COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

02/2010 VOL.53 NO.02

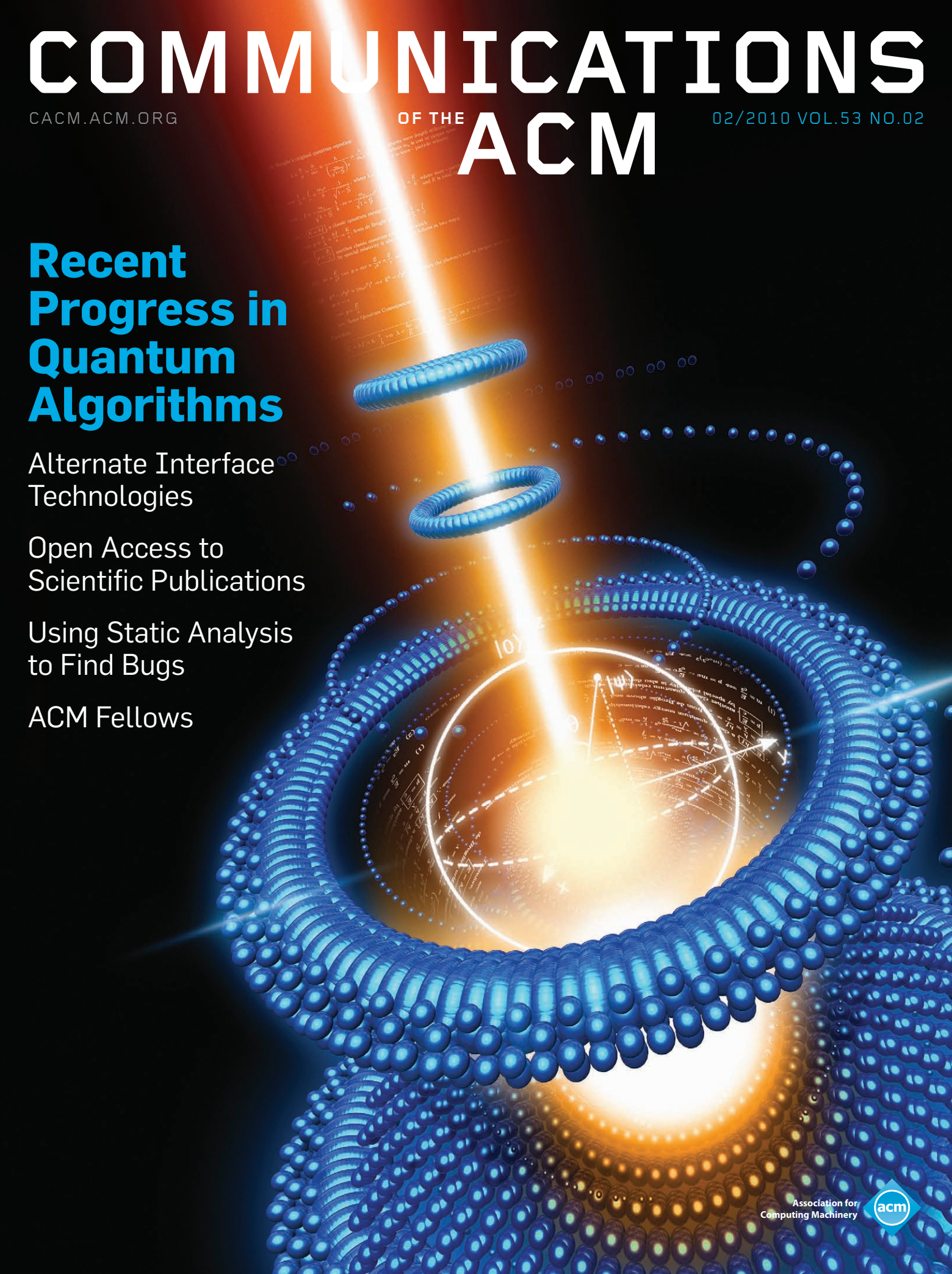
Recent Progress in Quantum Algorithms

Alternate Interface Technologies

Open Access to Scientific Publications

Using Static Analysis to Find Bugs

ACM Fellows



Association for Computing Machinery

24th IEEE INTERNATIONAL PARALLEL & DISTRIBUTED PROCESSING SYMPOSIUM

April 19-23, 2010 • Downtown Sheraton • Atlanta (Georgia) USA • www.ipdps.org

Sponsored by IEEE Computer Society Technical Committee on Parallel Processing.
In cooperation with ACM SIGARCH, IEEE Computer Society Technical Committee on Computer Architecture, and IEEE Computer Society Technical Committee on Distributed Processing

Hosted by Georgia Institute of Technology

IPDPS 2010 ATLANTA

IPDPS 2010

Join us in Atlanta at IPDPS 2010 where engineers and scientists from around the world will present their latest research findings in the fields of parallel processing and distributed computing. The deadline for discounted advance registration is February 28, 2010. The Advance Program at **www.ipdps.org** lists the technical sessions of contributed papers, invited speakers & topics, panel discussions and tutorials that are part of the IPDPS 2010 mid-week program, framed by the nineteen workshops listed here that will be held on Monday and Friday. Attendance at all workshops, along with receipt of their proceedings, is included in the registration fee for IPDPS. In addition, the PhD Forum, initiated under TCPP, continues in 2010 as a full-fledged IPDPS event, and there will be several commercial participants in Atlanta.

GENERAL CHAIR
David A. Bader, Georgia Institute of Technology

GENERAL VICE CHAIR
Alan Sussman, University of Maryland

PROGRAM CHAIR
Cynthia Phillips, Sandia National Laboratories

WORKSHOPS CHAIR
Ümit V. Çatalyürek, Ohio State University

PROCEEDINGS CO-CHAIRS
Yuanyuan Yang, State University of New York,
Stony Brook Xin Yuan, Florida State University

PhD FORUM GENERAL CHAIR
Manish Parashar, Rutgers University

PhD FORUM PROGRAM CHAIR
Luc Bougé, ENS Cachan, France

STEERING CO-CHAIRS
George Westrom, Discovery Science Center & FSEA
Viktor K. Prasanna, University of Southern California

IPDPS 2010 HOTEL

The Downtown Sheraton Atlanta Hotel is located in the heart of Georgia's capital city, minutes away from shopping and entertainment including the Georgia World Congress Center, Georgia Aquarium, Philips Arena, Centennial Olympic Park and the Georgia Dome. The special rate (see IPDPS Web) for IPDPS 2010 attendees is available three days before and after the conference and is guaranteed for reservations made before **March 31st**. Call +1-800-833-8624 and reference IEEE-IPDPS or go to IPDPS Web for online reservations.

IPDPS 2010 WORKSHOPS

Monday Workshops - 19 April 2010 - Atlanta	
HCW	Heterogeneity in Computing Workshop
RAW	Reconfigurable Architectures Workshop
HIPS	High-Level Parallel Programming Models & Supportive Environments
NIDISC	Nature Inspired Distributed Computing
HiCOMB	High Performance Computational Biology
APDCM	Advances in Parallel and Distributed Computing Models
CAC	Communication Architecture for Clusters
HPPAC	High-Performance, Power-Aware Computing
HPGC	High Performance Grid Computing
SMTPS	System Management Techniques, Processes, and Services
Friday Workshops - 23 April 2010 - Atlanta	
PDSEC	Parallel and Distributed Scientific and Engineering Computing
PMEO	Performance Modeling, Evaluation, and Optimisation of Ubiquitous Computing and Networked Systems
DPDNS	Dependable Parallel, Distributed and Network-Centric Systems
HOTP2P	Hot Topics in Peer-to-Peer Systems
MTAAP	Multi-Threaded Architectures and Applications
PDCoF	Parallel and Distributed Computing in Finance
LSPP	Large-Scale Parallel Processing
JSSPP	Job Scheduling Strategies for Parallel Processing
SSPS	Scalable Stream Processing Systems

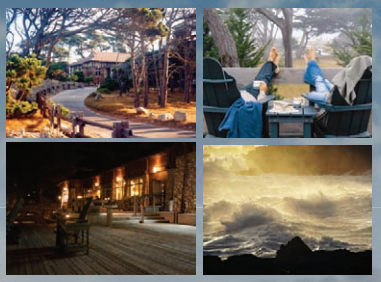
CALL FOR PAPERS

http://fdg2010.org

the International Conference on the

Foundations of Digital Games

June 19 - 21 2010



Asilomar Conference Grounds

Monterey, California, USA.

Submission Deadlines

Papers & Posters: *5th Feb*

Doctoral Consortium: *12th Feb*

Demos: *2nd April*

Infrastructure (Databases, Networks, Security)

Graphics & Interfaces

Game Studies

Learning in Games

Game Design

Artificial Intelligence

Computer Science & Games Education

Conference Chair:	**Ian Horswill**, Northwestern University
Program Chair:	**Yusuf Pisan**, University of Technology, Sydney
Doctoral Consortium Chair:	**Zoran Popovic**, University of Washington
Workshops Chair:	**Michael Mateas**, University of California, Santa Cruz
Panels Chair:	**Ian Bogost**, Georgia Institute of Technology
Tutorials Chair:	**Robin Hunicke**, That Game Company
Industrial Relations Chair:	**Hiroko Osaka**, Northwestern University
Local Arrangements Chair:	**Marilyn Walker**, University of California, Santa Cruz
Webmaster:	**Karl Cheng-Heng Fua**, Northwestern University

An Official Conference of the **Society for the Advancement of the Science of Digital Games**

SASDG

In Cooperation With ACM and its Special Interest Groups on Computer Science Education and Artificial Intelligence

SIGART

SIGCSE

acm

FDG 2010 Supported by:

Microsoft Research

COMMUNICATIONS OF THE ACM

Departments

Last Byte

News

Viewpoints

PHOTOGRAPH BY NICK KOZAK

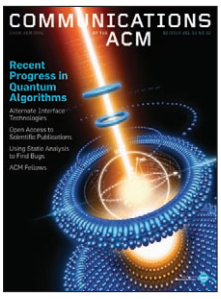
Association for Computing Machinery
Advancing Computing as a Science & Profession

02/2010
VOL. 53 NO. 02

Practice

Article development led by acmqueue
queue.acm.org

Review Articles



About the Cover: Hopes abound for a future with quantum computers and the algorithms that steer them. On page 84, Dave Bacon and Wim van Dam trace the latest discoveries in quantum algorithms that outperform their classical counterparts. Kenn Brown and Chris Wren, of Vancouver-based Mondolithic Studios, created a vision of quantum rings, dots, and algorithms encircling a Bloch sphere.

Contributed Articles

Research Highlights

Virtual Extension

As with all magazines, page limitations often prevent the publication of articles that might otherwise be included in the print edition. To ensure timely publication, ACM created *Communications*' Virtual Extension (VE).

VE articles undergo the same rigorous review process as those in the print edition and are accepted for publication on their merit. These articles are now available to ACM members in the Digital Library.

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

Communications of the ACM is the leading monthly print and online magazine for the computing and information technology fields. *Communications* is recognized as the most trusted and knowledgeable source of industry information for today's computing professional. *Communications* brings its readership in-depth coverage of emerging areas of computer science, new trends in information technology, and practical applications. Industry leaders use *Communications* as a platform to present and debate various technology implications, public policies, engineering challenges, and market trends. The prestige and unmatched reputation that *Communications of the ACM* enjoys today is built upon a 50-year commitment to high-quality editorial content and a steadfast dedication to advancing the arts, sciences, and applications of information technology.

ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.

Executive Director and CEO
John White
Deputy Executive Director and COO
Patricia Ryan
Director, Office of Information Systems
Wayne Graves
Director, Office of Financial Services
Russell Harris
Director, Office of Membership
Lillian Israel
Director, Office of SIG Services
Donna Cappo

ACM U.S. Public Policy Office
Cameron Wilson, Director

1100 Seventeenth St., NW, Suite 50
Washington, DC 20036 USA
T (202) 659-9711; F (202) 667-1066

Computer Science Teachers Association
Chris Stephenson
Executive Director
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (800) 401-1799; F (541) 687-1840

Association for Computing Machinery (ACM)
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (212) 869-7440; F (212) 869-0481

STAFF

DIRECTOR OF GROUP PUBLISHING
Scott E. Delman
publisher@cacm.acm.org

Executive Editor
Diane Crawford
Managing Editor
Thomas E. Lambert
Senior Editor
Andrew Rosenbloom
Senior Editor/News
Jack Rosenberger
Web Editor
David Roman
Editorial Assistant
Zarina Strakhan
Rights and Permissions
Deborah Cotton

Art Director
Andrij Borys
Associate Art Director
Alicia Kubista
Assistant Art Director
Mia Angelica Balaquiot
Production Manager
Lynn D'Addesio
Director of Media Sales
Jennifer Ruzicka
Marketing & Communications Manager
Brian Hebert
Public Relations Coordinator
Virgina Gold
Publications Assistant
Emily Eng

Columnists
Alok Aggarwal; Phillip G. Armour; Martin Campbell-Kelly; Michael Cusumano; Peter J. Denning; Shane Greenstein; Mark Guzdial; Peter Harsha; Leah Hoffmann; Mari Sako; Pamela Samuelson; Gene Spafford; Cameron Wilson

CONTACT POINTS

Copyright permission
permissions@cacm.acm.org
Calendar items
calendar@cacm.acm.org
Change of address
acmcoa@cacm.acm.org
Letters to the Editor
letters@cacm.acm.org

WEB SITE

http://cacm.acm.org

AUTHOR GUIDELINES

http://cacm.acm.org/guidelines

ADVERTISING

ACM ADVERTISING DEPARTMENT
2 Penn Plaza, Suite 701, New York, NY 10121-0701
T (212) 869-7440
F (212) 869-0481

Director of Media Sales
Jennifer Ruzicka
jen.ruzicka@hq.acm.org

Media Kit acmmediasales@acm.org

Subscriptions
An annual subscription cost is included in ACM member dues of $99 ($40 of which is allocated to a subscription to *Communications*); for students, cost is included in $42 dues ($20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is $100.

ACM Media Advertising Policy
Communications of the ACM and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current Advertising Rates can be found by visiting http://www.acm-media.org or by contacting ACM Media Sales at (212) 626-0654.

Single Copies
Single copies of *Communications of the ACM* are available for purchase. Please contact acmhelp@acm.org.

COMMUNICATIONS OF THE ACM

(ISSN 0001-0782) is published monthly by ACM Media, 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Periodicals postage paid at New York, NY 10001, and other mailing offices.

POSTMASTER

Please send address changes to
Communications of the ACM
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA

Association for Computing Machinery

Printed in the U.S.A.

DOI:10.1145/1646353.1646354

Diane Crawford

An Issue of Teamwork

This column was supposed to write itself. When Editor-in-Chief Moshe Vardi first asked me to tell readers how a typical issue of *Communications* comes together, I remember thinking "piece of cake."

After all, I've been part of that monthly process since Bill Gates earned his first billion. One column, coming up.

So, why have I been staring at a blank screen the last three days? The reason, I must admit, is the same reason I've enjoyed this work for so long. The real beauty of working on ACM's flagship publication is that *nothing* is ever the same. Unlike the majority of commercial and trade magazines, *Communications* rarely covers the same topic twice. Each issue truly follows a unique path to fruition.

In an effort to impart some sense of the production process, let's take this issue as an example. Many of the seeds for this edition were planted by mid-2009. Our cover story on quantum algorithms was invited by the EiC, but often the articles and viewpoints have been submitted and accepted after rigorous review by members of *Communications* Editorial Board (for details, see Author Guidelines http://cacm.acm.org/about-communications/author-center/author-guidelines). The News Board, working with Senior Editor Jack Rosenberger, teleconference monthly to determine the most important and timely stories to pursue for each issue. Managing Editor Tom Lambert works with the Viewpoints Board to ensure that section presents a compelling collection of columnists and commentary, delivering many diverse views on science and industry practices from around the world. Monthly features like CACM Online and Last Byte are also spirited by HQ editors.

The editorial lineup for each issue is a collaborative effort by the EiC, Board section chairs, and HQ staff. The EiC has the final word on all things editorial and makes the ultimate decision on the cover story and cover image. In a perfect world, *all* the manuscripts slated for each issue (save news stories) should be in the hands of HQ editors eight weeks prior to publication. That's when the real production cycle begins.

Pulling the Pieces Together

While the editorial elements have had months to simmer, the cover image and assorted graphical complements throughout this issue has only a few weeks to craft. The goal of the editorial artwork is to draw readers into an article by reflecting the subject in an intriguing way or by adding a creative spin to the subject. Given the assortment of topics in any edition, this is no small task. Therefore, one of the first steps in the process is a meeting with Group Publisher Scott Delman, staff editors, Art Director Andrij Borys, and Associate Art Director Alicia Kubista. Editors relay the gist of each article and volley ideas on how to best represent them graphically. Cover ideas take shape and artists and photographers are tapped. Over the weeks we watch early sketches turn into colorful works. Cover concepts can go through many iterations; often changing direction and detail as they progress.

While artwork is germinating, Lambert, Rosenberger, and Senior Editor Andrew Rosenbloom are editing every manuscript slated for the month. Working closely with the authors, their goal is to help make each article crisp and tight. Edited articles are sent to Assistant Art Director Mia Angelica Balaquiot, who flows the text into page galleys and redraws every submitted figure and table accompanying those articles to ensure a professional uniform presentation. Galleys are sent to the contact author for each article for one last look prior to publication. All final editorial tweaks are made; the design phase then takes over.

This is the juncture where editorial, artwork, and over 100 empty magazine pages merge. Borys and his team work under the ever-looming deadline to pull all the pieces together into a cohesive unit that flows from news stories, to viewpoints, to feature articles, to research, with lots of eye-catching elements in between. Staff editors read over every page one last time.

Advertising Sales Coordinator Jennifer Ruzicka submits the ad pages slated for the issue, and Production Manager Lynn D'Addesio coordinates every move between HQ and the printing facilities in Richmond, VA, where the final product is ultimately shipped.

As soon as one issue of *Communications* is "put to bed," we all move to the next one. There's art to discuss, authors to contact, and deadlines, always deadlines. The process may not always run this smoothly (ah, perfect world), but each issue always reflects a total team effort, and it is *always* a privilege to be part of that team.

Diane Crawford, EXECUTIVE EDITOR

ACM, Uniting the World's Computing Professionals, Researchers, Educators, and Students

Dear Colleague,

At a time when computing is at the center of the growing demand for technology jobs worldwide, ACM is continuing its work on initiatives to help computing professionals stay competitive in the global community. ACM's increasing involvement in activities aimed at ensuring the health of the computing discipline and profession serve to help ACM reach its full potential as a global and diverse society which continues to serve new and unique opportunities for its members.

As part of ACM's overall mission to advance computing as a science and a profession, our invaluable member benefits are designed to help you achieve success by providing you with the resources you need to advance your career and stay at the forefront of the latest technologies.

I would also like to take this opportunity to mention ACM-W, the membership group within ACM. ACM-W's purpose is to elevate the issue of gender diversity within the association and the broader computing community. You can join the ACM-W email distribution list at http://women.acm.org/joinlist.

ACM MEMBER BENEFITS:

- A subscription to ACM's newly redesigned monthly magazine, ***Communications of the ACM***
- Access to ACM's **Career & Job Center** offering a host of exclusive career-enhancing benefits
- **Free e-mentoring services** provided by MentorNet®
- **Full access to over 2,500 online courses** in multiple languages, and 1,000 virtual labs
- **Full access to 600 online books** from Safari® Books Online, featuring leading publishers, including O'Reilly (Professional Members only)
- **Full access to 500 online books** from Books24x7®
- Full access to the new ***acmqueue*** website featuring blogs, online discussions and debates, plus multimedia content
- The option to subscribe to the complete **ACM Digital Library**
- The **Guide to Computing Literature**, with over one million searchable bibliographic citations
- The option to connect with the **best thinkers in computing** by joining **34 Special Interest Groups** or **hundreds of local chapters**
- **ACM's 40+ journals and magazines** at special member-only rates
- ***TechNews***, ACM's tri-weekly email digest delivering stories on the latest IT news
- ***CareerNews***, ACM's bi-monthly email digest providing career-related topics
- ***MemberNet***, ACM's e-newsletter, covering ACM people and activities
- **Email forwarding service & filtering service**, providing members with a free acm.org email address and **Postini** spam filtering
- And much, much more

ACM's worldwide network of over 92,000 members range from students to seasoned professionals and includes many of the leaders in the field. ACM members get access to this network and the advantages that come from their expertise to keep you at the forefront of the technology world.

Please take a moment to consider the value of an ACM membership for your career and your future in the dynamic computing profession.

Sincerely,

Wendy Hall

President
Association for Computing Machinery

Association for Computing Machinery

Advancing Computing as a Science & Profession

Association for Computing Machinery

Advancing Computing as a Science & Profession

membership application & *digital library* order form

Priority Code: ACACM10

You can join ACM in several easy ways:

Online
http://www.acm.org/join

Phone
+1-800-342-6626 (US & Canada)
+1-212-626-0500 (Global)

Fax
+1-212-944-1318

Or, complete this application and return with payment via postal mail

Special rates for residents of developing countries:
http://www.acm.org/membership/L2-3/

Special rates for members of sister societies:
http://www.acm.org/membership/dues.html

Please print clearly

Name

Address

City | State/Province | Postal code/Zip

Country | E-mail address

Area code & Daytime phone | Fax | Member number, if applicable

Purposes of ACM

ACM is dedicated to:
1) advancing the art, science, engineering, and application of information technology
2) fostering the open interchange of information to serve both professionals and the public
3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature

ACM Code of Ethics:
http://www.acm.org/serving/ethics.html

choose one membership option:

PROFESSIONAL MEMBERSHIP:

- ❑ **ACM Professional Membership: $99 USD**
- ❑ **ACM Professional Membership plus the ACM Digital Library: $198 USD ($99 dues + $99 DL)**
- ❑ **ACM Digital Library: $99 USD (must be an ACM member)**

STUDENT MEMBERSHIP:

- ❑ **ACM Student Membership: $19 USD**
- ❑ **ACM Student Membership plus the ACM Digital Library: $42 USD**
- ❑ **ACM Student Membership PLUS Print *CACM* Magazine: $42 USD**
- ❑ **ACM Student Membership w/Digital Library PLUS Print *CACM* Magazine: $62 USD**

All new ACM members will receive an ACM membership card.
For more information, please visit us at www.acm.org

Professional membership dues include $40 toward a subscription to *Communications of the ACM*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Questions? E-mail us at acmhelp@acm.org
Or call +1-800-342-6626 to speak to a live representative

Satisfaction Guaranteed!

payment:

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc. in US dollars or foreign currency at current exchange rate.

❑ Visa/MasterCard ❑ American Express ❑ Check/money order

❑ Professional Member Dues ($99 or $198) $ _____

❑ ACM Digital Library ($99) $ _____

❑ Student Member Dues ($19, $42, or $62) $ _____

Total Amount Due $ _____

Card # Expiration date

Signature

DOI:10.1145/1646353.1646357

In the Virtual Extension

Communications' *Virtual Extension brings more quality articles to ACM members. These articles are now available in the ACM Digital Library.*

Reversing the Landslide in Computer-Related Degree Programs

Irma Becerra-Fernandez, Joyce Elam, and Susan Clemmons

Undergraduate and graduate enrollment in computer information system (CIS)-related coursework has been on a steady decline. The number of bachelor's degrees in computer science has fallen dramatically since 2004; a similar trend is also affecting academic programs that combine business and IT education. Rumors of CIS faculty layoffs compound the grim job outlook of CIS graduates, who fear not being able to find jobs in a market already plagued by challenges brought about by the dot com demise and IT outsourcing. How can CIS programs survive? This article details some successful intervention strategies that can be implemented to weather the impending crisis, and details how these interventions helped one institution reverse the downturn and reinvent the image of its CIS programs.

Practical Intelligence in IT: Assessing Soft Skills of IT Professionals

Damien Joseph, Soon Ang, Roger H.L. Change, and Sandra A. Slaughter

What qualities make a successful IT professional? This study develops and tests a measure of soft skills or practical intelligence of IT professionals, defined as intrapersonal and interpersonal strategies for managing self, career, and others. The instrument—SoftSkills for IT (SSIT)—elicits individuals' responses to IT work-related incidents and was administered to practicing IT professionals and inexperienced IT undergraduates. Results indicate that practical intelligence is measurable and SSIT discriminates between experienced and inexperienced IT professionals. This study concludes by identifying practical implications for selection, training, and development and proposes future research directions on assessing practical intelligence.

Wireless Insecurity: Examining User Security Behavior on Public Networks

Tim Chenoweth, Robert Minch, and Sharon Tabor

Wireless networks are becoming ubiquitous but often leave users responsible for their own security. The authors study whether users are securing their computers when using wireless networks. Automated techniques are used that scan users' machines after they associate with a university wireless network. Results show that over 9% of 3,331 unique computers scanned were not using a properly configured firewall. In addition, almost 9% had at least one TCP port open, with almost 6% having open ports with significant security implications. The authors also discuss cases where connected computers were compromised by Trojan programs, such as SubSeven and NetBus.

Informatics Creativity: A Role for Abductive Reasoning?

John Minor Ross

Analysts and programmers may be stymied when faced with novel tasks that seem beyond the reach of prior education and experience. Sometimes a solution appears, however, while considering something else seemingly altogether unrelated. Coming up with new ideas or creative approaches to overcome such problems may be less daunting once the role of adductive reasoning is considered.

Designs for Effective Implementation of Trust Assurances in Internet Stores

Dongmin Kim and Izak Benbasat

A study of 85 online stories offers a snapshot of how often Internet stores use trust assurances and what concerns they address. These findings will help business managers understand how other companies use trust assurances and help identify what can be improved within their organizations. For example, the authors determine that about 38% of total assurances were delivered in an ineffective way, which might cause shopping-cart-abandonment problems. The article offers design guidelines for trust assurances for Web developers based on the authors' analysis and previous studies.

Taking a Flexible Approach to ASPS

Farheen Altaf and David Schuff

There has been a recent revival of the ASP model through the notion of cloud computing and "software as a service." The purpose of this article is to better comprehend the Small to Medium Enterprise (SME) market for ASPs through an analysis of the factors that are most important to likely adopters. Through a survey of 101 SMEs, the authors find that cost, financial stability, reliability, and flexibility are all significantly associated with self-assessed likelihood of ASP adoption. Surprisingly, flexibility was negatively associated with likelihood of adoption, possibly indicating a perception that ASPs are not sought for their flexibility.

Managing a Corporate Open Source Software Asset

Vijay K. Gurbani, Anita Garvert, and James D. Herbsleb

Corporations have used open source software for a long time. But, can a corporation internally develop its software using the open source development models? It may seem that open source style development—using informal processes, voluntary assignment to tasks, and having few financial incentives—may not be a good match for commercial environments. This ongoing work demonstrates that under the right circumstances, corporations can indeed benefit from adopting open source development methodologies. This article presents findings on how corporations can structure software teams to succeed in developing commercial software using the open-source software development model.

Takes Two to Tango: How Relational Investments Improve IT Outsourcing Partnerships

Nikhil Mehta and Anju Mehta

As the recent economic crisis has shown, client-vendor partnership can quickly regress into a contractual arrangement with a primitive cost-cutting objective. Based on interviews with 21 vendor executives in India, the authors recommend that clients with a long-term vision for their IT outsourcing function may do well by developing mature partnerships with their vendors. To address the general lack of understanding about how to make it happen, the authors make provisional recommendations advising clients to make relational investments in selective areas. Key client benefits of such investments include improved service quality, cost savings, improved vendor sensitivity toward information security and privacy, and improved vendor capabilities to fulfill client's future IT needs.

BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish excerpts from selected posts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/1646353.1646358 http://cacm.acm.org/blogs/blog-cacm

Connecting Women and Technology

Guest blogger Valerie Barr writes about highlights of the ninth Grace Hopper Celebration of Women in Computing Conference, including keynote speeches by Megan Smith and Francine Berman.

From "Grace Hopper Conference Opening Session: Part 1"

http://cacm.acm.org/blogs/blog-cacm/43989

The theme of the ninth Grace Hopper Celebration (GHC) of Women in Computing is "Creating Technology for Social Good." This is a theme that has clearly resonated with many people as the conference totally sold out, with 1,608 attendees! There are 178 companies represented, 23 countries, and 728 students. There were more than 100 people who volunteered for the 16 committees that helped organize different aspects of the conference. One- quarter of the attendees, 430 people, are involved in presentations of panels, papers, workshops, and Birds of a Feather sessions. In addition to the usual conference type of activities, one of the sessions on Wednesday was a resumé review—the volunteer reviewers read more than 300 resumés.

A wonderful element of GHC is the emphasis on networking. At the conference opening on Thursday, Heidi Kvinge of Intel, the conference chair, challenged attendees to make the most of this aspect of the conference by introducing themselves to at least five new people per day. For the undergraduates in particular, Heidi gave a wonderful example of an elevator speech, demonstrating how they could capture all the key details about themselves in just a few sentences.

Heidi also acknowledged the support of SAP, which sponsors videoing at the conference. She showed the "I Am A Technical Woman" video that was made at GHC last year, which you can view at http://www.anitaborg.org/news/video. This is a great way to get a sense of what GHC is like, to understand the incredible energy at the conference. As per Heidi's request, please pass this video on to your friends and colleagues, and to anyone you know who has a daughter.

From "Grace Hopper Keynote 1: Megan Smith"

http://cacm.acm.org/blogs/blog-cacm/44258

Thursday's keynote address was by Megan Smith, vice president of new business development and general manager of Google.org. She has been at Google since 2003 and oversaw the acquisitions that resulted in Google Earth and Google Maps. In her talk Megan focused on the interconnectedness of CS, using four examples of areas that demonstrate this.

1. Interconnectedness of people around the world: When you look at Google query traffic worldwide, you see that there is almost no query traffic from Africa, though there is increasing SMS activity. For example, M-Pesa is a service in Kenya that allows full telephony-based money transfer. But the "real" commercial Internet is coming to Africa. Google is opening five new offices in Africa, bringing its total to seven. It will be doing maps and supporting all the "usual" Google apps, working with the Grameen AppLab, working on health-related applications, building on existing SMS efforts, and working to get NGO information on the Web.

2. Interconnectedness of data: People at Google have been generating real-time information about the spread of flu. They have used search logs to predict flu rates, based on the belief that the first thing people do when they get sick is start searching the Web. It turns out that they are 89% accurate on seasonal flu rates, based on verification with U.S. Center for Disease Control and Prevention (CDC) data. The benefit of this data-mining work is that Google can actually give the CDC real-time information more quickly than the CDC gets it from doctors and hospitals. Next, Google is working to get these applications into multiple languages—

it turns out that in many parts of the world it is becoming cheaper to collect data digitally than on paper, so the developing world can begin to move in this direction as well, using the data mining of digital data to gain information on trends.

3. Civil liberties: Events in Iran and Colombia have demonstrated the use of technology to mobilize people. The Alliance of Youth Movements Summit, held last year in New York City and soon to be held again, taught people how to create youth groups, and heavily utilized Webcasts and Facebook. Megan discussed the role that technology can play for people in "extreme" situations such as how SMS alerts can be used in parts of Africa to warn women about safe travel routes. She argued that technology can help speed up the improvement of life, particularly for women, in some parts of the world where there is still great danger. She also discussed the potential for improving education, such as creating opportunities for collaboration between schools across geographic and economic divides.

4. The environment: There are many CS opportunities in building the control systems involved for new energy-delivery approaches. For example, SolarBox is an application that will help groups of people organize to increase their buying power of solar panels in their neighborhood. Google's PowerMeter application will help people see power usage in their home. Studies show that once people know how much energy they are using, they usually decrease usage by 5%–15%.

Megan closed by saying that the 21st century will be all about these kinds of interconnectedness, and that there are many, many opportunities for people in CS to work on exciting, interesting, and relevant projects.

From "Grace Hopper Keynote 2: Fran Berman"

http://cacm.acm.org/blogs/blog-cacm/44532

The second keynote speaker was Fran Berman, vice president for research at Rensselaer Polytechnic Institute. Fran was formerly director of San Diego Supercomputer Center and has worked for years in the design and development of a national-scale cyberinfrastructure.

Fran's talk was entitled "Creating Technology for the Social Good: A Prologue." Her basic message was that science, engineering, and technology really matter when it comes to addressing and solving the most pressing problems facing society today.

As an example of a problem, and a solution born out of technology, she briefly discussed the area of safer environments through earthquake prediction. Basically, computer models are being developed to predict seismic activity. These models are then run on supercomputers, which generate output in the form of seismic predictions, showing where seismic activity will occur and how long it will last after an initial quake. This information is being used to develop new building codes, better disaster-response plans, and targeted retrofitting of older construction. Other examples Fran cited are the OLPC project to bring computers to children in the developing world and iRobot, which is developing robots suited for dangerous situations so that humans don't have to be exposed to danger and risk.

But Fran argues there is a major area that we have to address as the "prologue" to effectively addressing the large problems. That issue is data. We have to harness data, so that we can turn it into information and knowledge. This will help us create a strong foundation for efforts driven by science and engineering.

Electronic data is fragile. Much of it, such as wikis and Web sites, disappears quickly or is changed often. And there's a lot of it! There is currently more than a zettabyte of data. The U.S. Library of Congress alone has more than 295 terabytes of data. We are running out of room in which to store it all, so we have to be cognizant of the data life cycle and look at ways in which computer scientists can support the data life cycle. But we also have to recognize that the CS view of data is different than a librarian's view of data which, in turn, is different than an individual user's view of data.

So the key questions we need to think about are: What should we save? How should we save it? Who should pay for it?

Addressing these questions now is part of the process of creating a strong foundation for the technology work we will be doing in the years to come. Fran pointed out that we have to prepare today's students with technical skills, but that they also have to be prepared to understand international cultures, business, politics, and policy. Only then will they be ready to take on leadership roles in the years to come. Fran closed by saying that to create positive change we have to ask the hard questions, particularly about the representation of women and minorities in CS; create goals and metrics of success, and then hold people to them; publicly recognize the successes of our colleagues and students; and, when possible, use our role to create policy, set priorities, and handle resource allocation.

From "Final Thoughts About Grace Hopper Conference"

http://cacm.acm.org/blogs/blog-cacm/44533

My wrap-up from Grace Hopper—some Web sites and information about women and technology worldwide, much of it gleaned during the session "The 'F' Word: Feminism and Technology." The repeated message was that we have to see technology as a means to an end, not an end itself. If we want to build technology to help women, particularly in the developing world, we have to have the relevant context and involve women themselves in the development process. For example, in rural Pakistan the majority of women are illiterate, so a text-based Internet tool is useless. But an audiovisual medium, like one that is currently being used to provide information about health-care services, will be much more successful. While in the developed world we seem to always think of a computer solution, usually Web-based, to problems, these days the technology that will help women is most likely to involve mobile phones. This has been demonstrated in Africa by the Advancement through Interactive Radio project in which mobile phone technology allows women to participate in call-in programs on TV and radio, giving them a voice in community affairs which they had not previously had. C

Valerie Barr is the chair of the computer science department at Union College.

DOI:10.1145/1646353.1646359

David Roman

Where the Data Is

The vast Internet delivers only a sliver of the information the average American consumes each day, according to a recent report by the University of California, San Diego (http://hmi.ucsd.edu/howmuchinfo_research_report_consum.php). Less than 1% of the daily data diet comes from Web browsing, way behind the top three providers—video games, television, and movies, says Roger E. Bohn and James E. Short in "How Much Information? 2009 Report on American Consumers." That improbable finding is due to a statistical slight of hand that measures information consumption in bytes, giving streaming video supersize status. The numbers are less extreme though still slanted when Internet information is measured in hours (15.6% of total versus 75.1% for moving-picture media) or words (24.7% versus 47.6%). They are skewed further because they only measure information consumed at home, not work.

The report is full of extravagant, oddball stats: Americans consumed 1.3 trillion hours of information in 2008; those 3.6 zettabytes are 20 times more than could be stored at one time on all the world's hard drives. Some 70% of adults play computer games; 10 million subscribers watched 3.6 hours of video on average per month on mobile phones; and gaming PCs occupied less than 2% of data consumption hours but almost 40% of total bytes. And on it goes; data devoid of context. What else to make of its comparison of Lincoln's Gettysburg Address (1,293 bytes) with an episode of NBC's "Heroes" (10GB)?

The report sidesteps assigning "value" to information and allows that the Internet "provides a substantial portion of some kinds of information" but misses a key distinction: movies and computer games aren't consumed as a source of data. They are junk food, M&Ms to the Web's square meals. Yes, the Web also serves empty calories, and TV can serve healthy fare. But people do not use video-based media to finish work, communicate with friends, and help make decisions. The Internet plays all these roles. It is the primary source of news for 40% of adults (http://people-press.org/report/479/internet-overtakes-newspapers-as-news-source). Some 80% use it to socialize (http://www.ruderfinn.com/about/news/rf-s-new-study-of.html). It is a source of information on medical conditions and treatments for 61% of U.S. adults (http://www.pewinternet.org/Reports/2009/8-The-Social-Life-of-Health-Information.aspx). And 53% find help for financial decisions (http://www.ebri.org/pdf/surveys/mrcs/2007_factsheet_1.pdf) online.

Serious stuff, but hard to see amidst the 34GB you take in each day.

PHOTOGRAPH BY KEIRSTEN BALUKAS

ACM Member News

FIRST-EVER CS EDUCATION WEEK A SUCCESS

ACM and a handful of partners launched Computer Science Education Week in the U.S., which received a great deal of media attention and was celebrated at schools and businesses throughout the nation. Held from Dec. 5–11, the central hub for Computer Science Education Week was http://www.csedweek.org, which featured curriculum guides, posters, and research, and also provided a platform for users to share ideas through interactive links at Facebook, YouTube, and Twitter.

Computer Science Education Week received favorable media attention in *The New York Times*, *The Washington Post*, and elsewhere. And in a three-minute video posted on YouTube (http://www.youtube.com/CSEdWeek) ACM CEO John White discussed the importance of computer science, noting that "Computing is fueling countless advances, from improving communications and advancing health care to protecting national security and improving energy efficiency to helping understand the depths of the universe."

SIGSCE 2010

The theme of SIGCSE 2010 is "Making Contact," and the 41st ACM Technical Symposium on Computer Science Education will continue SIGCSE's tradition of bringing together colleagues from around the world to make contact with fellow educators via panels, poster and special sessions, workshops, Birds of a Feather sessions, and informal settings.

The keynote speakers are Sally Fincher of the University of Kent who will speak about useful sharing; Carl E. Wieman of the University of British Columbia who will discuss science education for the 21st century; and Intel's Michael Wrinn of Intel who will talk about parallel computing and industry and academic collaboration. SIGCSE 2010 will be held from March 10–13 in Milwaukee, WI. For more information, visit http://sigcse.org/sigcse2010.

Technology | DOI:10.1145/1646353.1646360 Kirk L. Kroeker

Alternate Interface Technologies Emerge

Researchers working in human-computer interaction are developing new interfaces to produce greater efficiencies in personal computing and enhance miniaturization in mobile devices.

HARDWARE ENGINEERS CONTINUE to pack more processing power into smaller designs, opening up an array of possibilities that researchers say will lead to human-computer interfaces that are more natural and efficient than the traditional hallmarks of personal computing. These smaller designs have given rise to new mobile platforms, where the barrier to further miniaturization no longer is the hardware itself but rather humans' ability to interact with it. Researchers working in human-computer interaction (HCI) are dedicating effort in both areas, developing interfaces that they say will unlock greater efficiencies and designing new input mechanisms to eliminate some of the ergonomic barriers to further miniaturization in mobile technology.

NanoTouch, a back-of-device input technology for very small screens on mobile devices and electronic jewelry. The technology demonstrated here by Patrick Baudisch was developed at Microsoft Research and Hasso Plattner Institute.

PHOTO COURTESY OF KAY HERSCHELMANN

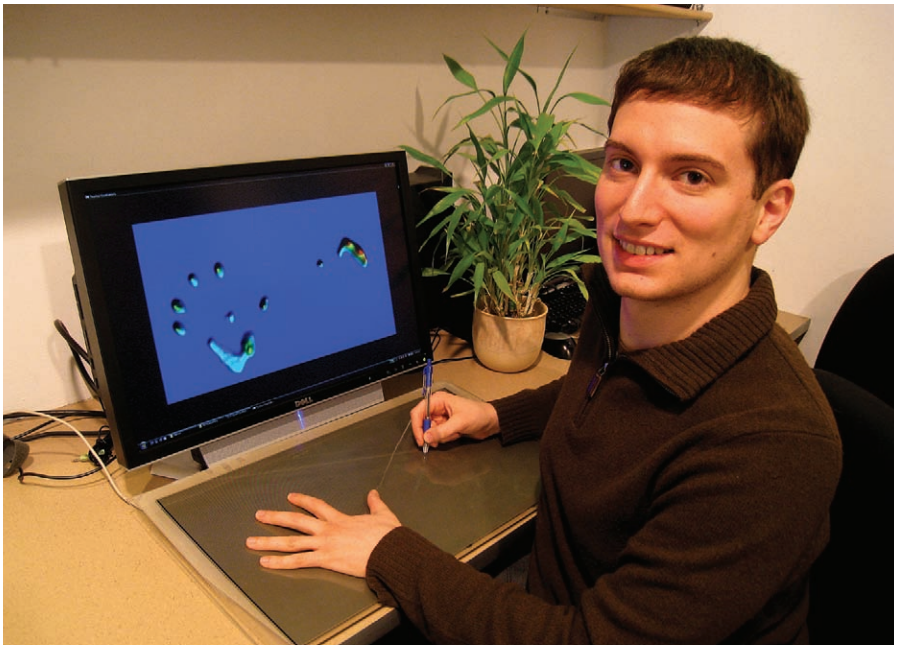
Patrick Baudisch, a computer science professor at Hasso Plattner Institute in Potsdam, Germany, points out that there are two general approaches to HCI, a field that draws on computer science, engineering, psychology, physics, and several design disciplines. One approach focuses on creating powerful but not always totally reliable interfaces, such as speech or gesture input. The other focuses on creating less complex, more reliable input techniques. Partial to the second approach, Baudisch argues that interfaces developed with simplicity and reliability in mind facilitate an uninterrupted engagement with the task at hand, increasing the opportunity for users to experience what psychologist Mihaly Csikszentmihalyi calls "optimal experience" or "flow."

Baudisch began his HCI career in the Large Display User Experience group at Microsoft Research, where he focused on how users could interact

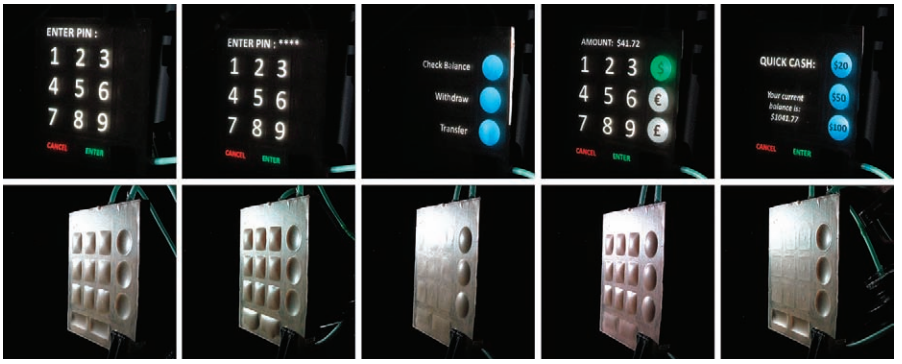
more effectively with wall displays and other large-format technologies that render traditional input techniques nearly useless. In his current work at the Hasso Plattner Institute, Baudisch focuses on projects designed to facilitate the transition from desktop to mobile computing. "There is a single true computation platform for the masses today," he says. "It is not the PC and not One Laptop Per Child. It is the mobile phone—by orders of magnitude. This is the exciting and promising reality we need to design for."

One example of an interface technology that Baudisch designed to facilitate this transition to mobile computing is NanoTouch. While current mobile devices offer advanced capabilities, such as touch input, they must be large enough to manipulate with fingers. The NanoTouch interface, which is designed to sidestep this physical constraint, makes the mobile device appear translucent and moves the touch input to the device's back side so that the user's fingers do not block the front display. Baudisch says NanoTouch eliminates the requirement to build interface controls large enough for big fingertips and makes it possible to interact with devices much smaller than today's handheld computers and smartphones.

In his most recent project, called RidgePad, Baudisch is working on a way to improve the recognition accuracy of touch screens. By monitoring not only the contact area between finger and screen, but also the user's fingerprint within that contact area, RidgePad reconstructs the exact angle at which the finger touches the display. This additional information allows for more specific touch calibration, and, according to Baudisch, can effectively double the accuracy of today's touch technology.



Interpolating force sensitive resistance (IFSR), a multitouch input technology developed at New York University's Media Research Lab. In this demo, Ilya Rosenberg demonstrates a 24-inch Touchco IFSR sensor that serves as an interactive desktop surface.



A prototype shape-shifting ATM display that can assume multiple graphical and tactile states. The display was developed at Carnegie Mellon University's Human-Computer Interaction Institute.

Increasing Human Performance

Another HCI researcher focusing on new interface technologies for mobility is Carnegie Mellon University's Chris Harrison, who points out that while computers have become orders of magnitude more powerful than they were a few decades ago, users continue to rely on the mouse and keyboard, technologies that are approximately 45 and 150 years old, respectively. "That's analogous to driving your car with ropes and sails," he says. "It's this huge disparity that gets me excited about input." Harrison, a graduate student in CMU's Human-Computer Interaction Institute, says that because computers have grown so powerful, humans are now the bottleneck in most operations. So the question for Harrison is how to leverage the excess computing power to increase human performance.

For one of Harrison's projects, increasing human performance benefitted from his observation that mobile devices frequently rest on large surfaces: Why not use the large surfaces for input? This line of thinking was the birthplace for Harrison's Scratch Input technology. The idea behind Scratch Input is that instead of picking up your media player to change songs or adjust volume, the media player stays where it is but monitors acoustic information with a tiny, built-in microphone that listens to the table or desk surface. To change the volume or skip to the next track, for example, you simply run your fingernail over the surface of the table or desk using different, recognizable scratch gestures. The media player captures the acoustic information propagating through the table's surface and executes the appropriate command.

In addition to developing Scratch Input, which Harrison says is now mature enough to be incorporated into commercial products, he and his colleagues have been working on multitouch displays that can physically deform to simulate buttons, sliders, arrows, and keypads. "Regular touch

TOP PHOTO COURTESY OF ILYA ROSENBERG; BOTTOM PHOTO COURTESY OF CHRIS HARRISON

screens are great in that they can render a multitude of interfaces, but they require us to look at them," says Harrison. "You cannot touch type on your iPhone." The idea with this interface technology, which Harrison calls a shape-shifting display, is to offer some of the flexibility of touch screens while retaining some of the beneficial tactile properties of physical interfaces.

Another interface strategy designed to offer new advantages while retaining some of the benefits of older technology is interpolating force-sensitive resistance (IFSR). Developed by two researchers at New York University, IFSR sensors are based on a method called force-sensitive resistance, which has been used for three decades to create force-sensing buttons for many kinds of devices. However, until Ilya Rosenberg and Ken Perlin collaborated on the IFSR project, it was both difficult and expensive to capture the accurate position of multiple touches on a surface using traditional FSR technology alone. "What we created to address this limitation took its inspiration from human skin, where the areas of sensitivity of touch receptors overlap, thereby allowing for an accurate triangulation of the position of a touch," says Perlin, a professor of computer science at NYU's Media Research Lab.

In IFSR sensors, each sensor element detects pressure in an area that overlaps with its neighboring elements. By sampling the values from the touch array, and comparing the output of neighboring elements in software, Rosenberg and Perlin found they could track touch points with an accuracy approaching 150 dots per inch, more than 25 times greater than the density of the array itself. "In designing a new kind of multitouch sensor, we realized from the outset how much more powerful a signal is when properly sampled," says Rosenberg, a graduate student in NYU's Media Research Lab. "So we aimed to build an input device that would be inherently anti-aliasing, down to the level of the hardware."

Recognizing the increased interest in flexible displays, electronic paper, and other technologies naturally suited to their core technology, Rosenberg and Perlin spun off their sensor technology into a startup called Touchco, and now are working with other companies to integrate IFSR into large-touch screens and flexible electronic displays. In addition, the team is looking into uses as diverse as musical instruments, sports shoes, self-monitoring building structures, and hospital beds.

"There is a single true computation platform for the masses today," says Patrick Baudisch, and it "is the mobile phone—by orders of magnitude. This is the exciting and promising reality we need to design for."

"It seems that many of the hurdles are largely ones of cultural and economic inertia," says Perlin. "When a fundamentally improved way of doing things appears, there can be significant time before its impact is fully felt."

As for the future of these and other novel input technologies, users themselves no doubt will have the final word in determining their utility. Still, researchers say that as input technologies evolve, the recognizable mechanisms for interfacing with computers will likely vanish altogether and be incorporated directly into our environment and perhaps even into our own bodies. "Just as we don't think of, say, the result of LASIK surgery as an interface, the ultimate descendents of computer interfaces will be completely invisible," predicts Perlin. "They will be incorporated in our eyes as built-in displays, implanted in our ears as speakers that properly reconstruct 3D spatial sound, and in our fingertips as touch- or haptic-sensing enhancers and simulators."

On the way toward such seamlessly integrated technology, it's likely that new interface paradigms will continue to proliferate, allowing for computer interactions far more sophisticated than the traditional mouse and keyboard. CMU's Harrison predicts that eventually humans will be able to walk up to a computer, wave our hands, speak to it, stare at it, frown, laugh, and poke its buttons, all as a way to communicate with the device. In Harrison's vision of this multimodal interfacing, computers will be able to recognize nuanced human communication, including voice tone, inflection, and volume, and will be able to interpret a complex range of gestures, eye movement, touch, and other cues.

"If we ever hope for human-computer interaction to achieve the fluidity and expressiveness of human communication, we need to be equally diverse in how we approach interface design," he says. Of course, not all tools and technologies will require a sophisticated multimodal interface to be perfectly functional. "To advance to the next song on your portable music player, a simple button can be fantastically efficient," says Harrison. "We have to be diligent in preserving what works, and investigate what doesn't." C

Further Reading

Baudisch, P. and Chu, G.
Back-of-device interaction allows creating very small touch devices. *Proceedings of the 27th International Conference on Human Factors in Computing Systems*, Boston, MA, April 2009.

Csikszentmihalyi, M.
Flow: The Psychology of Optimal Experience. HarperCollins, New York, 1990.

Erickson, T. and McDonald, D. W. (eds.)
HCI Remixed: Reflections on Works That Have Influenced the HCI Community. MIT Press, Cambridge, MA, 2008.

Harrison, C. and Hudson, S. E.
Scratch input: creating large, inexpensive, unpowered, and mobile finger Input surfaces. *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, Monterey, CA, October 2008.

Rosenberg, I. and Perlin, K.
The UnMouse pad: an interpolating multi-touch force-sensing input pad. *ACM Transactions on Graphics 28*, 3, August 2009.

Based in Los Angeles, **Kirk L. Kroeker** is a freelance editor and writer specializing in science and technology.

Science | DOI:10.1145/1646353.1646361 Alex Wright

Type Theory Comes of Age

Type systems are moving beyond the realm of data structure and into more complex domains like security and networking.

When the philosopher Bertrand Russell invented type theory at the beginning of the 20th century, he could hardly have imagined that his solution to a simple logic paradox—defining the set of all sets not in themselves—would one day shape the trajectory of 21st century computer science.

Once the province of mathematicians and social scientists, type theory has gained momentum in recent years as a powerful tool for ensuring data consistency and error-free program execution in modern commercial programming languages like C#, Java, Ruby, Haskell, and others. And thanks to recent innovations in the field, type systems are now moving beyond the realm of data structure and into more complex domains like security and networking.

First, a quick primer. In programming languages, a type constitutes a definition of a set of values (for example, "all integers"), and the allowable operations on those values (for example, addition and multiplication). A type system ensures the correct behavior of any program routine by enforcing a set of predetermined behaviors. For example, in a multiplication routine, a type system might guarantee that a program will only accept arguments in the form of numerical values. When other values appear—like a date or a text string—the system will return an error. For programmers, type systems help prevent undetected execution errors. For language implementers, they optimize execution and storage efficiency. For example, in Java integers are represented in the form of 32 bits, while doubles are represented as 64 bits. So, when a Java routine multiplies two numbers, the type system guarantees they are either integers or doubles. Without that guarantee, the runtime would need to conduct an expensive check to determine what kinds of numbers were being multiplied before it could complete the routine.

Benjamin C. Pierce, University of Pennsylvania.

What distinguishes a type system from more conventional program-level verification? First, a type system must be "decidable"; that is, the checking should happen mechanically at the earliest opportunity (although this does not have to happen at compilation time; it can also be deferred to runtime). A type system should also be transparent; that is to say, a programmer should be able to tell whether a program is valid or not regardless of the particular checking algorithm being used. Finally, a "sound" type system prevents a program from performing any operation outside its semantics, like manipulating arbitrary memory locations.

Languages without a sound type system are sometimes called unsafe or weakly typed languages. Perhaps the best-known example of a weakly typed system is C. While C does provide types, its type checking system has been intentionally compromised to provide direct access to low-level machine operations using arbitrary pointer arithmetic, casting, and explicit allocation and deallocation. However, these maneuvers are fraught with risk, sometimes resulting in programs riddled with bugs like buffer overflows and dangling pointers that can cause security vulnerabilities.

By contrast, languages like Java, C# , Ruby, Javascript, Python, ML, and Haskell are strongly typed (or "type safe"). Their sound type systems catch any type system violations as early as possible, freeing the programmer to focus debugging efforts solely on valid program operations.

Static and Dynamic Systems

Broadly speaking, type systems come in two flavors: static and dynamic. Statically typed languages catch almost all errors at compile time, while dynamically typed languages check most errors at runtime. The past 20 years have seen the dominance of statically typed languages like Java, C# , Scala, ML, and Haskell. In recent years, however, dynamically typed languages like Scheme, Smalltalk, Ruby, Javascript, Lua, Perl, and Python have gained in popularity for their ease of extending programs at runtime by adding new code, new data, or even manipulating the type system at runtime.

Statically typed languages have restrictions and annotations that make it possible to check most type errors at compile time. The information used by the type checker can also be used by tools that help with program text-editing and refactoring, which is a considerable advantage for large modular programs. Moreover, static type systems enable change. For example, when an important data structure definition is

PHOTOGRAPH BY SALLIE DEAN SHATZ

changed in a larger program, the type system will automatically point to all locations in the program that also need change. In a dynamically typed language it would be extremely difficult to make such changes in larger programs as it would be not known what other parts are affected by the change. On the other hand, some correct programs may be rejected by a static type system when it is not powerful enough to guarantee soundness.

In an effort to make static type systems more flexible, researchers have developed a number of extensions like interface polymorphism, a popular approach introduced by object-oriented languages like Simula, C++, Eiffel, Java, or C#. This method allows for inclusion between types, where types are seen as collections of values. So, an element of a subtype—say, a square—can be considered as an element of its supertype—say, a polygon—thus allowing the elements of different but related types to be used flexibly in different contexts.

Another form of polymorphism, found in almost all programming languages, is ad hoc polymorphism (also called overloading) where code behaves in different ways depending on the type. This approach has found its fullest expression in Haskell, thanks in part to the efforts of Philip Wadler, professor of theoretical computer science at the University of Edinburgh. "When we designed Haskell, it quickly became clear that overloading was important and that there was no good solution," says Wadler. "We needed overloading for equality, comparison, arithmetic, display, and input."

The Haskell system has evolved considerably over the years, thanks to the contributions of a far-flung group of contributors. "Once we'd come up with the initial idea of type classes, it led to a vast body of work, all sorts of clever researchers coming up with neat extensions to the system, or applying it do things that we'd never thought it could do," says Wadler. Today, Haskell ranks as the programming world's premier case study in ad hoc polymorphism.

The dream of unifying static and dynamic type systems has long fascinated researchers. Today, several computer scientists are probing the possibility of merging these approaches. Wadler is pursuing a promising line of research called blame calculus that attempts to incorporate both static and dynamic typing, while Erik Meijer, a language architect at Microsoft Research, proposes to use "static typing when possible, dynamic typing when necessary."

The theory for refinement types has existed for a long time, but recent progress in automatic theorem proving makes refinement types suddenly practical.

Security Type Systems

In recent years, researchers have also been exploring type systems capable of capturing a greater range of programming errors such as the public exposure of private data. These emerging type systems are known as security type systems. Whereas a traditional type system enforces rules by assigning values to data types, a security type system could apply the same principle of semantic checking to determine the owner of a particular piece of information. Those annotations could then help ensure the integrity of data flowing through the system. Two promising security research projects include the AURA programming language, developed by Steve Zdancewic, associate professor of computer science at University of Pennsylvania, and Jif, a Java-based security-typed language developed by Andrew Myers, associate professor of computer science at Cornell University.

Another interesting application of type checking involves hybridizing type systems and theorem provers. "Historically, there have been two parallel tracks in the software engineering world: type systems and theorem provers. The type systems track has always emphasized lightweight methods," says Benjamin C. Pierce, professor of computer science at the University of Pennsylvania, "but the formal methods people aren't interested in that. Today, they're starting to meet in the middle."

Pierce points to refinement types, which are types qualified by a logical constraint; an example is the type of even numbers, that is, the type of integers qualified by the is-an-even-number constraint. While the theory for refinement types has existed for a long time, only recent progress in automatic theorem proving makes refinement types suddenly practical. A promising security project was recently performed by Andrew D. Gordon, principal researcher at Microsoft Research Cambridge, and colleagues. They added a system of refinement types to the F# programming language and were able to verify security properties of F# implementations of cryptographic protocols by type checking.

While type theory has matured considerably over the past 100 years, it still remains an active research arena for computer scientists. As type systems move beyond the realm of data consistency and into headier computational territories, the underlying principles of type theory are beginning to shape the way researchers think about program abstractions at a deep—even philosophical—level. Bertrand Russell would be proud. C

Further Reading

Hindley, J.R.
Basic Simple Type Theory. **Cambridge University Press, New York, 2008.**

Pierce, B.
Types and Programming Languages. **MIT Press, Cambridge, MA, 2002.**

Flanagan, C.
Hybrid type checking. *SIGPLAN Notices 41*, 1, Jan. 2006.

Cardelli, L.
Type systems. *The Computer Science and Engineering Handbook*, Allen B. Tucker (ed.). CRC Press, Boca Raton, FL, 1996.

Church, A.
A formulation of the simple theory of types. *Journal of Symbolic Logic 5*, 2, 1940.

Alex Wright is a writer and information architect who lives and works in Brooklyn, NY. Daan Leijen and Wolfram Schulte of Microsoft Research contributed to the development of this article.

Society | DOI:10.1145/1646353.1646362 Sarah Underwood

Improving Disaster Management

Social networking, sophisticated imaging, and dual-use technologies promise improved disaster management, but they must be adopted by governments and aid agencies if more lives are to be saved in the wake of crises.

When the September 11, 2001 terrorist attacks ripped through the heart of New York City, the July 7, 2005 suicide bombings created chaos and mayhem in central London, and the Boxing Day 2004 Indian Ocean earthquake caused a tsunami that swept away more than 200,000 lives, information and communication technologies played a part in disaster response. Communication was key, but not always possible as infrastructure collapsed and mobile phone networks became overloaded, prompting renewed efforts to develop effective disaster management strategies.

Research organizations, relief agencies, and technology providers agree that technology can save lives in a disaster, but here consensus ends, with a rift between researchers pursuing the possibilities of Web 2.0 applications and field workers largely committed to their traditional toolkit of mobile and satellite phones.

"IT systems make it possible to handle large amounts of data to assess the situation after a disaster has struck, but we need to move in the direction of community response, developing media centers that accommodate citizen journalists," says Kathleen Tierney, professor of sociology and director of the Natural Hazards Research and Applications Information Center at the University of Colorado, Boulder. "Official agencies need to interact with citizen first responders and assess information being gathered in the field, rather than depending on information that is filtered through hierarchical organizations. The information may not be 100% accurate, but mobile phone pictures taken at the scene provide the most rapid information."

Tierney is a proponent of Web 2.0 applications, such as Twitter, blogs, and wikis, as a means of improving disaster response. "People's use of technology in crises is expanding rapidly, ahead of the use of technology by emergency management agencies," she says. "We need people in these agencies who are disaster and technology savvy. We also need opinion leaders and foundations that fund disaster assistance to think along new lines."

A man uses his mobile phone's camera to document the aftermath of the earthquake that devastated the city of Dujiangyan, in the province of Sichuan, China, on May 12, 2008.

The concept of community response plays into the thesis of Ramesh Rao, director of the California Institute for Telecommunications and Information Technology (Calit2) at the University of California, San Diego. Rao believes research into technological, sociological, and organizational issues is critical to the improvement of disaster response. "There is a great opportunity to use technology to improve disaster management, but at the moment people are in the way of that improvement," he says. "There is a lack of information sharing, which is not a technology problem, but a problem stemming from organizations wanting things this way."

One advance Rao suggests is dual use of technology. During peaceful times, dual-use technology, such as a mobile phone, operates as a everyday personal communications device, but during an emergency it transforms into an information sensor and disseminator. This overcomes aversion to using different communications equipment during a crisis and eliminates the time lag caused by government agencies collecting, processing, and distributing crisis-related data. Direct, firsthand reports from a disaster can provide a realistic picture, helping to avoid the confusion that can result from widespread, and not always accurate, television broadcasts.

Proving the value of dual-use technol-

PHOTOGRAPH BY NICK KOZAK

ogy, Calit2 has developed a peer-to-peer incident notification system that builds on the concept of human sensors. The human sensors collect and relay information about events, such as wildfires and traffic accidents, to first responders and the general public using mobile phones.

The notification system is available across all of California's major cities and is based on speech recognition, allowing commuters to call in and report incidents, or call in and listen about events that could disrupt their travel. Content is self-regulated with users flagging incidents that are irrelevant or abusive, and the notification system includes algorithms that rate users who report incidents. Conversely, the system can notify all users of an incident via a voice call or text message.

"If you see an accident and call 911, the police come, the local radio station picks up what has happened and transmits the problem, but that's often too late to allow commuters to get off the highway," Ganz Chockalingam, principal development engineer at Calit2 and developer of the notification system, explains. "The incident notification system is successful because there is no middle man and no time delay. Typically, we get about 1,000 calls a day, but during the California wildfires [in 2009] that went up to about 10,000 calls a day."

Unlike traditional disaster management systems that are inflexible and constrained by capacity, this peer-to-peer system can scale to deliver real-time information during a disaster as there is no single channel of information and no single point of information control. The project is currently funded by Calit2, although Chockalingam points out that costs are relatively low as much depends on user technology.

Disaster response needs to "move in the direction of community response, developing media centers that accommodate citizen journalists," says Kathleen Tierney.

New Thinking, New Tools

Stepping out of research labs and into the commercial world, disaster management development follows the path of improved global communication and social networking. ImageCat, a risk-management innovation company based in Long Beach, CA, works with government, industry, and research organizations to develop new thinking, tools, and services that are available to both government agencies and businesses, such as insurance companies that need to estimate losses after a disaster.

ImageCat also concentrates on remote sensing and geographic information systems. One recent development is a virtual disaster viewer, a Web-based system that uses remote sensors to gather information that can be displayed and used to assess the aftermath of disaster. The viewer is a social networking-type tool to which researchers and the public can add information. It was tested during the 2008 earthquake in the Sichuan province of China, with about 100 engineers accessing before and after satellite imagery to monitor the extent of damage in the region.

"The viewer allows users to conduct a virtual disaster survey without leaving their desks," says ImageCat CEO Ronald Eguchi. "In the future, people at the scene who take pictures with mobile phones will be able to upload them to the viewer. We are in discussion with the United Nations about how it could use the viewer in disaster management."

With a myriad of sensors around the world and optical and radar satellite images of an adequate resolution to see people on the ground, the possibilities of gathering and sharing data, such as earthquake, coastal, and hurricane images, are almost boundless and could support significant humanitarian relief efforts.

The benefits of satellite images in disaster response are not limited to countries that own satellites, however, and disaster-struck countries can activate a clause of the Charter of the United Nations that requires imagery to be made available to a nation in distress.

Eguchi believes much more can be done with technology to save lives in a disaster, but also recognizes the realities of aid agencies. "Technologies such as the virtual disaster viewer are new to aid agencies. Agencies need technology that is tested and validated, that adds value to what they do and demonstrates efficiency in getting the right information to the right people at the right time," he says.

UNICEF workers say the best in-field technologies for disaster management are simple to use, low maintenance,

ACM Awards News

2010 ACM Fellows Nominations

The designation "ACM Fellow" may be conferred upon those ACM members who have distinguished themselves by outstanding technical and professional achievements in informa¬tion technology, who are current professional members of ACM and have been professional members for the preceding five years. Any professional member of ACM may nominate another member for this distinction. *Nominations and endorsements must be submitted online no later than Sept. 1, 2010.* For Fellows Guidelines, go to http://awards.acm.org/html/fellow_nom_guide.cfm/

Nomination information organized by a principal nominator includes excerpts from the candidate's current curriculum vitae, listing selected publications, patents, technical achievements, honors, and other awards; a description of the work of the nominee, drawing attention to the contributions which merit designation as Fellow; and supporting endorsements from five ACM members. For the list of 2009's ACM Fellows, see p. 21.

lightweight, and cheap. Reflecting Rao's promotion of dual-use technologies that require minimal training, UNICEF is pioneering RapidSMS and a field-based communications system called Bee.

RapidSMS provides real-time transmission of data in a breaking emergency or in a long-standing disaster, allowing aid workers to monitor supplies and report on situations that require immediate response. At the moment, it is being used in a number of African countries to monitor nutrition, water, sanitation, and supply chains.

UNICEF's Bee is an open source emergency telecommunications system that provides Internet access in areas where infrastructure is nonexistent or unusable. It provides a telephony service and Wi-Fi access to applications such as ones that monitor health and track supplies. The Bee system requires no tools, and can be installed by a field worker and be operational within 30 minutes. Working with RapidSMS, it helps UNICEF provision supplies appropriately and gives workers immediate warnings of potential health risks and disease outbreaks. When an area stabilizes, the Bee system is left in place, acting as a base for a new communications infrastructure.

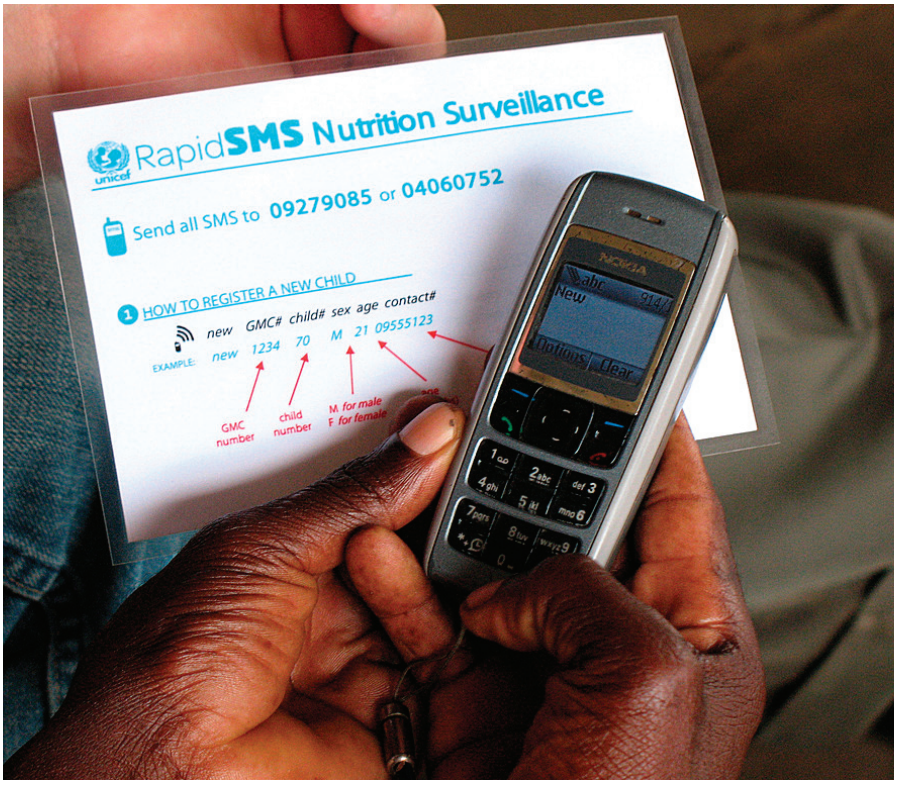
International humanitarian aid agency Mercy Corps also focuses on communication. "The driving force behind all the equipment we carry is communications and we use the cheapest services we can find," says Richard Jacquot, a member of Mercy Corps' global emergency operations team. "We all carry a BlackBerry, a cheap local mobile, a satellite phone such as the Inmarsat BGAN, sometimes a VHF radio set and a laptop equipped with applications, including Microsoft Office, email, and a Web browser. Where there is no network, or when a hostile government blocks access to local networks, we depend on satellite technology."

Far from the world of Web 2.0 collaboration tools in the war-torn regions of Africa and Asia, Jacquot's technology request is simple. "We have to carry too many tools, then we have to get authorization for them depending on where we are. Integrating everything into one device is too much to ask for, but some integration would be great," he says.

While those in the labs developing technology applications for disaster management and those in the field seeking simple, usable, and affordable solutions share the belief that technology can save more lives, it is important to not ignore the self-imposed threat created by technology development and nurtured by those who see it as a weapon with which to kill rather than a shield with which to protect.

"In the 2008 Mumbai hotel bombing, the terrorists were more effective in their use of technology than the officials," notes Rao. "They used information systems and cell phones, while the commandos sent in to clean up did not have cell phones, so they couldn't communicate with each other or people in the hotel.

"Official organizations are slow and their thinking is ossified," Rao says, "but that will change and technology will be better used to reduce the loss of lives in disaster management and support quicker recovery of communities and infrastructure."



In Malawi, UNICEF's RapidSMS system enables the registration of children and the monitoring of their nutritional status in an effort to stem the nation's high infant mortality rate.

PHOTOGRAPH BY UNICEF/ EVAN WHEELER

Further Reading

Dilmaghani, B. and Rao, R.
A systematic approach to improve communication for emergency response. *Proceedings of the 42nd Hawaii International Conference on System Sciences*, Waikoloa, HI, January 2009.

Lien, Y.N., Tsai, T.C., and Jang, H.C.
A MANET-based emergency communication and information system for catastrophic natural disasters. *2009 29th IEEE International Conference on Distributed Computing Systems*, Montreal, Canada, June 2009.

Palen, L. and Liu, S.
Citizen communications in crisis: anticipating a future of ICT-supported public participation. *Proceedings of the 25th International Conference on Human Factors in Computing Systems*, San Jose, CA, April 2007.

Shklovski, I., Palen, L., and Sutton, J.
Finding community through information and communication technology in disaster response. *Proceedings of the ACM 2008 Conference on Computer Supported Cooperative Work*, San Diego, CA, November 2008.

Showalter, P. and Lu, Y. (eds.)
Geospatial Techniques in Urban Hazard and Disaster Analysis. Springer, New York, 2009.

Tierney, K.
From the margins to the mainstream? disaster research at the crossroads. *Annual Review of Sociology*, Palo Alto, CA, August 2007.

Sarah Underwood is a technology writer based in Teddington, UK.

News | DOI:10.1145/1646353.1646381

ACM Fellows Honored

Forty-seven men and women are inducted as 2009 ACM Fellows.

The ACM Fellow Program was established by Council in 1993 to recognize and honor outstanding ACM members for their achievements in computer science and information technology and for their significant contributions to the mission of the ACM. The ACM Fellows serve as distinguished colleagues to whom the ACM and its members look for guidance and leadership as the world of information technology evolves.

The ACM Council endorsed the establishment of a Fellows Program and provided guidance to the ACM Fellows Committee, taking the view that the program represents a concrete benefit to which any ACM member might aspire, and provides an important source of role models for existing and prospective ACM Members. The program is managed by the ACM Fellows Committee as part of the general ACM Awards program administered by Calvin C. Gotlieb and James J. Horning. For details on Fellows nominations, see p. 19.

The men and women honored as ACM Fellows have made critical contributions toward and continue to exhibit extraordinary leadership in the development of the Information Age and will be inducted at the ACM Awards Banquet on June 26, 2010, in San Francisco, CA. These 47 new inductees bring the total number of ACM Fellows to 722 (see www.acm.org/awards/fellows/ for the complete listing of ACM Fellows).

Their works span all horizons in computer science and information technology: from the theoretical realms of numerical analysis, combinatorial mathematics and algorithmic complexity analysis; through provinces of computer architecture, integrated circuits and firmware spanning personal computer to supercomputer design; into the limitless world of software and networking that makes computer systems work and produces solutions and results that are useful—and fun—for people everywhere.

Their technical papers, books, university courses, computing programs, and hardware for the emerging computer/communications amalgam reflect the powers of their vision and their ability to inspire colleagues and students to drive the field forward. The members of the ACM are all participants in building the runways, launching pads, and vehicles of the global information infrastructure.

ACM Fellows

Hagit Attiya, Technion
David F. Bacon, IBM T.J. Watson Research Center
Ricardo Baeza-Yates, Yahoo! Research
Chandrajit L. Bajaj, University of Texas at Austin
Vijay Bhatkar, International Institute of Information Technology, Pune
José A. Blakeley, Microsoft Corporation
Gaetano Borriello, University of Washington
Alok Choudhary, Northwestern University
Nell B. Dale, University of Texas at Austin (Emerita)
Bruce S. Davie, Cisco Systems
Jeffrey A. Dean, Google, Inc.
Thomas L. Dean, Google, Inc.
Bruce R. Donald, Duke University
Thomas Erickson, IBM T. J. Watson Research Center
Gerhard Fischer, University of Colorado
Ian T. Foster, Argonne National Laboratory/ University of Chicago
Andrew V. Goldberg, Microsoft Research Silicon Valley
Michael T. Goodrich, University of California, Irvine
Venugopal Govindaraju, University at Buffalo, SUNY
Rajiv Gupta, University of California, Riverside
Joseph M. Hellerstein, University of California, Berkeley
Laurie Hendren, McGill University
Urs Hoelzle, Google, Inc.
Farnam Jahanian, University of Michigan
Erich L. Kaltofen, North Carolina State University
David Karger, Massachusetts Institute of Technology
Arie E. Kaufman, State University of New York at Stony Brook
Hans-Peter Kriegel, University of Munich (Ludwig-Maximilians-Universitaet Muenchen)
Maurizio Lenzerini, Sapienza Universitá di Roma
John C.S. Lui, The Chinese University of Hong Kong
Dinesh Manocha, University of North Carolina at Chapel Hill
Margaret Martonosi, Princeton University
Yossi Matias, Google, Inc.
Renee J. Miller, University of Toronto
John T. Riedl, University of Minnesota
Martin Rinard, CSAIL-MIT
Patricia Selinger, IBM Research
R. K. Shyamasundar, Tata Institute of Fundamental Research
Shang-Hua Teng, University of Southern California
Chandramohan A. Thekkath, Microsoft Corporation - Microsoft Research
Robbert van Renesse, Cornell University
Baba C. Vemuri, University of Florida
Paulo Veríssimo, University of Lisbon
Martin Vetterli, Ecole Polytechnic Federale de Lausanne (EPFL)
Kyu-Young Whang, Korea Advanced Institute of Science and Technology (KAIST)
Yorick Wilks, University of Sheffield
Terry Winograd, Stanford University

DOI:10.1145/1646353.1646363

M. Angela Sasse

Privacy and Security
Not Seeing the Crime for the Cameras?

Why it is difficult—but essential—to monitor the effectiveness of security technologies.

In terms of sales, remote surveillance camera systems—commonly known as closed-circuit television (CCTV)—are a huge success story. Billons of dollars are spent on CCTV schemes by governments in developed countries each year, and sales to commercial companies and home users have been increasing, too. CCTV can be used for many purposes—ranging from monitoring traffic flows on highways, to allowing visitors in zoos to observe newborn animals during their first few days without disturbing them. The vast majority of CCTV purchases are made with the aim of improving safety and security. The London Underground was the first public transport operator to install cameras on station platforms, so train drivers could check doors were clear before closing them. CCTV has come a long way since then: last summer, the technology writer Cory Doctorow noticed that a single London bus now has 16 cameras on it (see Figure 1). The advance from analog to digital technology had a major impact on CCTV: cameras are much smaller and cheaper, video is often transmitted wirelessly, and recordings are stored on hard disks, rather than tapes. Integration with other digital technologies offers further possibilities: image processing makes it possible to recognize automobile license plates automatically and match them against databases to check if a vehicle has been reported as stolen, or is uninsured. Advances in hardware—such as high-definition cameras—and image processing—such as the ability to process face and iris information from images taken at a distance, not detecting unattended objects—will enable a wide range of possible technology solutions (imagine the whole industry salivating).

The burgeoning sales figures and ubiquity of cameras suggest that surely CCTV technology must be effective.

The burgeoning sales figures and ubiquity of cameras suggest that surely CCTV technology must be effective. The U.K. government has invested heavily in CCTV over the past 15 years, making it the country with the highest CCTV camera-to-person ratio on earth (Greater London alone has one camera for every six citizens). A key driver for adoption was that local authorities seeking to combat crime could obtain government funds to purchase CCTV. In the public debate, this policy has been justified mainly with two arguments: "*the public wants it*," and "*surely it's obvious that it works*." As evidence for the latter, policymakers often point to high-profile (and often highly emotionally charged) cases:

- In 1993, CCTV images from a shopping mall camera showed police investigators that the murdered toddler James Bulger had been abducted by two teenagers, who were then apprehended and convicted.
- Images from London Transport

cameras led to the identification and apprehension of the four men who carried out the failed 7/21 "copycat" bombing attempts in 2005.

The still images from these cases (see Figure 2a/b) have become iconic—visual proof that CCTV works. Those who questioned its value in the public debate, and dared to mention the "p-word"—were largely dismissed as "privacy cranks," out of touch with the needs of policing and the wishes of ordinary citizens. But over the past two years, new doubts have been raised over the benefits:

▸ In summer 2008, a report by London police concluded that CCTV contributed to solving about 3% of street crimes. About £500 million ($700 million) has been spent on publicly funded CCTV in Greater London.

▸ In August 2009, a senior officer in the London police stated that, on an annual basis, about one crime was resolved for every 1,000 cameras in operation. He warned "*police must do more to head off a crisis in public confidence over the use of surveillance cameras.*"

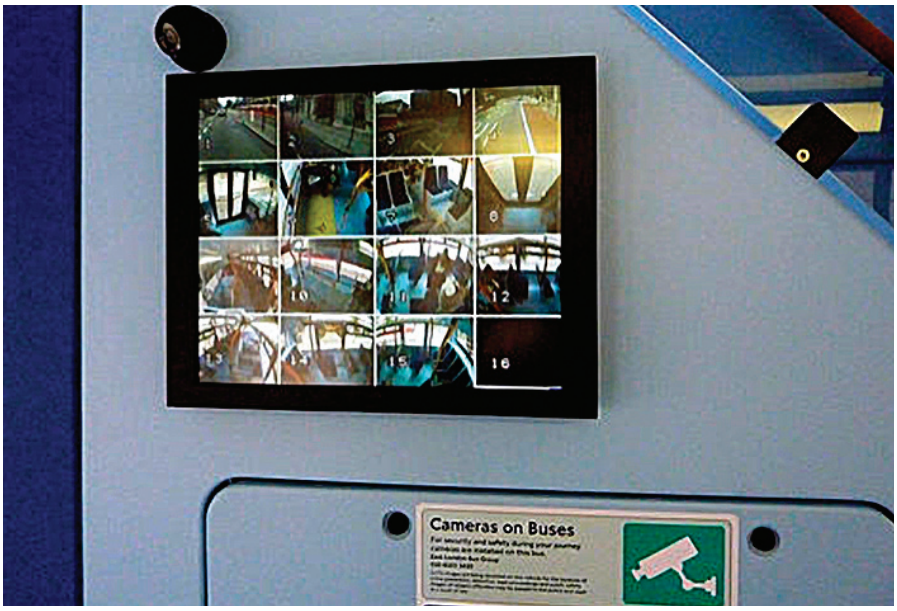
▸ In September 2009, John Bromley-Davenport, a leading criminal lawyer in Manchester, said images from CCTV did not prevent crime or help bring criminals to justice.[3] He prosecuted the killers of a man kicked to death outside a pub. The incident was recorded on CCTV, but police officers did not arrive in time to stop the attack, plus the quality of the recorded footage was too low to be used for identification purposes in court. (The killers were convicted on eyewitness evidence.) The chief executive of a company that helps police analyze CCTV footage estimated "*that about half of the CCTV cameras in the country are next to useless when it comes to safeguarding the public against crime and assisting the police to secure convictions.*" Bromley-Davenport said that large amounts of money spent on technology meant less money was available to have police officers on the street—and that police presence was what mattered for preventing crime.

▸ In October 2009, design college professor Mike Press called for a moratorium on further CCTV deployments in Scotland, because the technology was "*costly and futile* [...] *a lazy approach to crime prevention*" that was dangerous because it created "*a false sense of security, encouraging* [citizens] *to be careless with property and personal safety.*"[6]

Thus, the effectiveness of CCTV is being questioned in the country that has been a leading and enthusiastic adopter. Surely, this must ring alarm bells in the industry supplying such systems? Not really. The industry response is that more advanced technology will fix any problems. High-definition cameras, for instance, would provide better image quality and increase likelihood of identification. The same chief executive who said that half of all current cameras were useless suggests that "*intelligent cameras*" will improve effectiveness and reduce privacy invasion because they "*only alerting a police officer when a potential incident is taking place.*" London police experts also hope that "*future technology will boost conviction rates using CCTV evidence.*"

The proposals for new technology and effectiveness include building a national CCTV database of convicted offenders and unidentified suspects, and use of "*tracking technology developed by*

PHOTO COLLAGE BASED ON A PHOTOGRAPH BY TOM PAGE



A single London bus has 16 cameras mounted on it.

the sports advertising industry" to search footage for suspects and incidents. Since that technology is not quite ready, London police publish images of suspects on the Internet and ask the public for help. Recruitment of untrained members of the public to assist in CCTV monitoring is a growing trend:

- In a London housing project, residents have been given access to CCTV cameras, books of photos of individuals who had been warned not to trespass on the estate, and a phone number to call if they spotted any of them.
- In the tourist town of Stratford-on-Avon, residents and business can connect their own CCTV cameras to an Internet portal, and and volunteers who spot and report crimes can win prizes of up to £1,000.[a]
- Approximately $2 million has been spent on Webcams for virtual border surveillance at the Texas-Mexico border, enabling virtual local residents to spot and report illegal immigration.

The involvement of untrained members of the public in surveillance harbors many potential risks to privacy, public order, and public safety (e.g, vigilantism) that must be identified and considered. But even leaving those concerns aside, early indications from the last project suggest this not a quick fix to make CCTV more effective. The *El Paso Times*[6] reported in January 2009 that the program was not effective because only a dozen incidents had been reported. A spokesperson for the Governor of Texas responded that the problem was not with the technology, but the way its effectiveness was assessed. It may look like a weak argument, but it points to the key problem: How do you assess effectiveness of a security technology such as CCTV? How can you determine whether the results represent value for the money spent on technology, or privacy invasions that occur because of its existence?

The answer is conceptually simple: *effectiveness* of a particular deployment means that it achieves its stated purpose; *efficiency* means the desired results are worth more than the resources required to achieve them. But the execution of a study to measure them is a challenging and costly exercise. One of the few controlled studies to date was carried out in the clothing retail shops in 1999[1]:

- The *purpose* of installing the systems was clearly defined: reduce the stock losses through customer and staff theft.
- The *measures* for stock losses were clearly defined: the number and value of stock losses was monitored, and any reduction of losses calculated as a percentage of sales profits during the same period.
- Stock losses were measured four times—twice during a six-month period *before* and *after* the introduction of CCTV.
- The *efficiency* was calculated in terms of how many years the system would have to operate at the observed level of effectiveness to recover its investment.
- During the one-year period, they monitored for a number of *side effects* such as footfall, overall sales, customer assessment of shops, and so forth.

This illustrates that carrying out a meaningful assessment under controlled conditions requires significant resources and domain expertise, even for a conceptually simple study: the assessment was focused on a single crime, the monitoring environment was constant, and systems for measuring the impact were already in place. The results showed that stock losses were reduced significantly in the first three months of CCTV introduction—but then rose again. After six months, the average loss reduction was a near-insignificant £4—at an average capital expenditure of £12,000 per CCTV system, it would take 58 years to recoup the capital cost. In the end, only shops selling high-value fashion using high-end CCTV systems reduced stock

Still images from two cases that resulted in apprehension of perpetrators.

a Details of the rewards were revealed last December; see http://news.bbc.co.uk/1/hi/technology/8393602.stm

losses to a level that would mean their investment was recouped within two years. The authors concluded that anyone buying an off-the-shelf CCTV system may be wasting their money: only systems designed against a specific threat in a specific operating environment are effective.

A 2005 study of 13 CCTV systems funded by the U.K. government for crime prevention[2] concluded they had little or no impact on crime recorded by the police, or on citizens' perception of crime (based on victimization rates, fear of crime and other information collected via local surveys). A common problem was that those who bought the systems were unclear about the purpose of—and hence the technical and operating requirements for—the systems. Many projects were driven by an "*uncritical view that CCTV was 'a good thing' and that specific objectives were unnecessary*." Systems were bought because funding was available, or because a neighboring town had purchased one. There was no understanding of what CCTV could achieve, what types of problems it was best suited to alleviate, and which configuration and support technologies work best for which requirements. With buyers being unclear about objectives and lacking expertise, the systems were generally chosen by the salesperson—who tended to pick the system that suited the budget. In day-to-day operations, it turned out that many cameras were ineffective because they were badly placed, broken, dirty, or lighting was insufficient—problems that were previously identified in London Underground control rooms.[6] Both Gill and Spriggs[2] and McIntosh[6] also found that operator performance in the control room was hampered by a large number of disparate systems and information sources, and inefficient audio communication channels. Recent research by my own team[5] found these problems continue to affect operator performance, as do ever-increasing camera-to-operator ratios. Recorded video was generally too poor to be used for evidence. These problems suggest CCTV for crime prevention can only be effective as part of an overall set of measures and procedures designed to deal with specific problems. Effective communication and coordination between CCTV control rooms and those on the ground (police, shop and bar staff, private security forces) is key—and of course there must be sufficient staff on the ground to respond. And cameras need clear lines of sight and sufficient lighting. We found current practice is still a long way off: cameras were ineffective because of trees and shrubs growing in front, and autofocus cameras broken because they were pointed at flags and bunting.

The effectiveness of CCTV is being questioned in the country that has been a leading and enthusiastic adopter.

Current research shows that CCTV for crime prevention is largely ineffective. It is "lazy" to assume that installing technology solves the problem. It takes domain knowledge and attention to detail to make security technology work effectively—to date, this has been ignored, with expensive consequences.

References

1. Beck, A., and Willis, A. Context-specific measures of CCTV effectiveness in the retail sector. *Crime Prevention Studies 10* (1999), 251–269; http://www.popcenter.org/library/crimeprevention/volume_10/10-BeckWillis.pdf
2. Gill, M. and Spriggs, A. Assessing the impact of CCTV. Home Office Research Study 292. UK Home Office Research, Development and Statistics Directorate, February 2005; http://www.homeoffice.gov.uk/rds/pdfs05/hors292.pdf
3. Hope, C. 'Worthless' CCTV camera footage is not good enough to fight crime. *The Daily Telegraph*, (Aug. 26, 2009); http://www.telegraph.co.uk/news/newstopics/politics/6088086/Worthless-CCTV-camera-footage-is-not-good-enough-to-fight-crime-leading-QC-warns.html
4. Keval, H.U. and Sasse, M.A. "Not the usual suspects": A study of factors reducing the effectiveness of CCTV. To appear in *The Security Journal 23*, 2 (Apr. 2010); http://www.palgrave-journals.com/sj/journal/vaop/ncurrent/abs/sj20082a.html
5. Luff, P., Heath, C., and Jirotka, M. (2000): Surveying the scene: technologies for everyday awareness and monitoring in control rooms. *Interacting with Computers 13*, (2000), 193–228.
6. McIntosh, L. Soaring CCTV cameras 'are costly, futile and politically motivated'. *The Times* (Oct. 13, 2009); http://www.timesonline.co.uk/tol/news/uk/scotland/article6871833.ece?token=null&offset=12&page=2

M. Angela Sasse (a.sasse@cs.ucl.ac.uk) is Head of Information Security Research in the Department of Computer Science at University College London.

Calendar of Events

February 15–17
International Symposium on BioComputing 2010, Calicut, India,
Contact: Dan Tulpan,
Phone: 506-861-0958,
Email: dan.tulpan@nrc-cnrc.gc.ca

February 19–21
Symposium on Interactive 3D Graphics and Games, Bethesda, MD,
Sponsored: SIGGRAPH,
Contact: Chris Wyman,
Phone: 319-353-2549,
Email: cwyman@cs.uiowa.edu

February 22–23
Workshop on Mobile Opportunistic Networking, Pisa, Italy,
Sponsored: SIGMOBILE,
Contact: Sergio Polazzo,
Phone: 390957382370,
Email: polazzo@iit.unict.it

February 22–23
Multimedia Systems Conference Phoenix, Arizona,
Sponsored: SIGMM,
Contact: Wu-Chi Feng,
Phone: 503-725-2408,
Email: wuchi@cs.pdx.edu

February 25–27
India Software Engineering Conference, Mysore, India,
Contact: Srinivas Padmanabhuni,
Email: s_padmana@yahoo.com

February 26–27
International Conference and Workshop on Emerging Trends in Technology, Mumbai, India,
Contact: Poorva Girish Waingankar,
Email: poorva.waingankar@thekureducation.org

March 2–5
International Conference on Human Robot Interaction, Nara, Japan,
Sponsored: SIGCHI, SIGART,
Contact: Pamela J. Hinds,
Phone: 650-723-3843,
Email: phinds@stanford.edu

March 2–5
IEEE Pacific Visualization 2010, Taipei, Taiwan,
Contact: Shen Han-Wei,
Email: hwshen@cse.ohio-state.edu

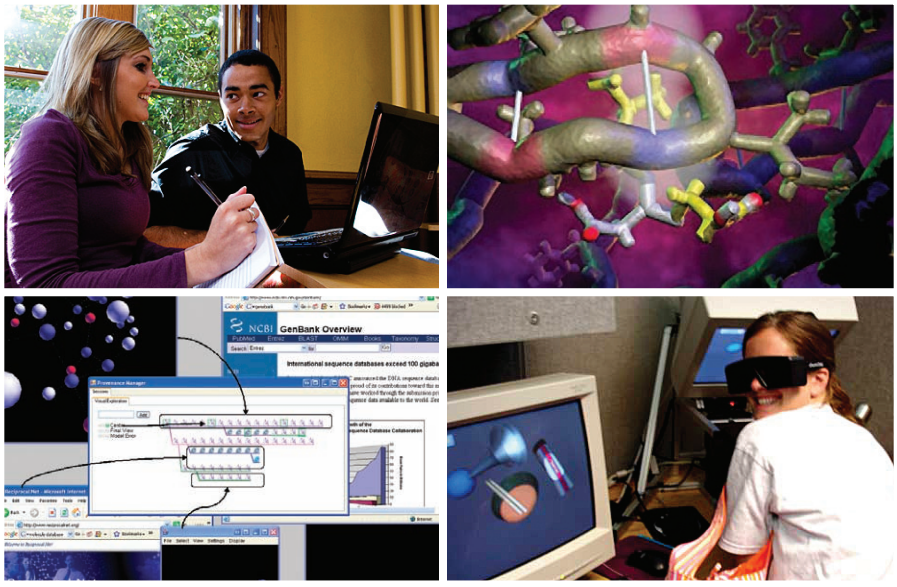
DOI:10.1145/1646353.1646364

Dennis P. Groth and Jeffrey K. MacKie-Mason

Education

Why an Informatics Degree?

Isn't computer science enough?



Informatics programs offer diverse applications, as shown in these scenes from the informatics program at Indiana University, Bloomington.

What is an informatics degree, and why? These are questions that have been posed to us on innumerable occasions for almost a decade by students, parents, employers, and colleagues, and when asked to prepare a *Communications* Education column to answer that question, we jumped at the opportunity.

The term "informatics" has different definitions depending on where it is used. In Europe, for instance, computer science is referred to as informatics. In the U.S., however, informatics is linked with applied computing, or computing in the context of another domain. These are just labels, of course. In practice, we are educating for a broad continuum of computing disciplines, applications, and contexts encountered in society today.

From Computer Science to Informatics

Computing provides the foundation for science, industry, and ultimately for the success of society. Computing education traditionally has focused on a set of core technological and theoretical concepts, and teaching these concepts remains critically important. Meanwhile, advances in computing occur and are driven by the need to solve increasingly complex problems in domains outside traditional computer science. Students, teachers, and scholars in other fields are keenly interested in computational thinking, and computing itself increasingly is informed by the challenges of other disciplines.

For example, to design good online auction technology, computer scientists found that they needed to understand how humans would select bidding strategies given the system design, and indeed how to design the system to motivate certain types of behavior (truthful value revelation, for example). This co-design problem led to fruitful interdisciplinary collaborations between computer scientists, economists and, increasingly, social psychologists. Likewise, designing successful technology for trust, privacy, reputation, and sharing in social computing environments requires both computer science and behavioral science.

These interactions between problem domain context and computational design are characteristic of the maturing of computer science. Computing is no longer owned solely by computer science, any more than statistics is owned solely by faculty in statistics departments. Computing and computational thinking have become ubiquitous, and embedded in all aspects of science, research, industry, government, and social interaction. Consider the flurry of excitement about "e-commerce" in the late 1990s. Quickly e-commerce moved from being seen as a new field to being absorbed in "commerce": the study of business communications, logistics, fulfillment, and strategy, for which the Internet and computing were just two technologies in a complex infrastructure.

How then does computing education need to change to respond to the new reality, and more importantly, to be equipped to respond to future developments? We must embrace the diversity of ways in which problems

IMAGES COURTESY OF INDIANA UNIVERSITY SCHOOL OF INFORMATICS AND COMPUTING

are solved through the effective use of computing, and we must better understand the diverse problem domains themselves.

The vision for informatics follows from the natural evolution of computing. The success of computing is in the resolution of problems, found in areas that are predominately outside of computing. Advances in computing—and computing education—require greater understanding of the problems where they are found: in business, science, and the arts and humanities. Students must still learn computing, but they must learn it in contextualized ways. This, then, provides a definition for informatics: informatics is a discipline that solves problems through the application of computing or computation, in the context of the domain of the problem. Broadening computer science through attention to informatics not only offers insights that will drive advances in computing, but also more options and areas of inquiry for students, which will draw increasing numbers of them to study computation.

The success of computing is in the resolution of problems, found in areas that are predominately outside of computing.

Informatics Programs

Computer science is focused on the design of hardware and software *technology* that provides computation. Informatics, in general, studies the intersection of people, information, and technology systems. It focuses on the ever-expanding, ubiquitous, and embedded relationship between information systems and the daily lives of people, from simple systems that support personal information management to massive distributed databases manipulated in real time. The field helps design new uses for information technology that reflect and enhance the way people create, find, and use information, and it takes into account the strategic, social, cultural, and organizational settings in which those solutions will be used.

In the U.S., informatics programs emerged over the past decade, though not always under the informatics name, and often in different flavors that bear the unique stamp of their faculty. Prominent examples include "Informatics" (Indiana University, University of Michigan, University of Washington, UC Irvine), "Human Computer Interaction" (Carnegie Mellon University), "Interactive Computing" (Georgia Tech), "Information Technology and Informatics" (Rutgers), and "Information Science and Technology" (Penn State). Some programs emerged primarily from computer science roots; others from information and social science roots. They do all generally agree on the centrality of the interaction of people and technology, and thus regardless of origin they are multidisciplinary and focus on computation in human contexts.

Informatics is fundamentally an interdisciplinary approach to domain problems, and as such is limited neither to a single discipline nor a single domain. This is evident in another type of diversity in such programs: some take a fairly broad approach, with several distinct tracks or application domains, which can range as widely as art and design, history, linguistics, biology, sociology, statistics and economics. Other programs are limited to a single application domain, such as bioinformatics (for example, Iowa State, Brigham Young, and UC Santa Cruz). Thus, informatics programs can have as many differences as they have commonalities. This has been reflected in some confusion and frustration about how to establish a community of interest. For example, there is an "iSchool" caucus (about 27 members), and a partially overlapping CRA (IT) Deans group (about 40 members). To illustrate some of the issues, we will describe two of the broader programs with which we are most familiar.

The School of Informatics and Computing at Indiana University Bloomington offers a traditional CS degree and an informatics degree, which was first offered in 2000. Its informatics curriculum is focused along three dimensions that are first presented in an introductory course: foundations, implications, and applications. Unlike most traditional computer science curricula, the introductory course does not focus on programming as the sole problem-solving paradigm. Instead, a number of skills, concepts, and problem solving techniques are introduced and motivated by context-based problems, including logical reasoning, basic programming, teamwork, data visualization, and presentation skills. Following this introduction, foundations courses include discrete math and logical reasoning, a two-course programming sequence, and a course on data and information representation, while implications courses include social informatics and human computer interaction. The foundations topics are similar to those in a computer science program; however, the ordering is quite different, in that programming comes last rather than first. This sequencing increases retention in the major because students have more time to develop their technical skills.

At Indiana, the interdisciplinary component of the curriculum is accomplished through a mixture of three methods: elective courses covering technology use and issues in specific problem domains; a required senior capstone project, aimed at solving a "real-world" problem; and a required cognate specialization of at least five courses in another discipline. There are currently over 30 different specializations from around 20 disciplines available, including: business, fine arts, economics, information security, biology, chemistry, telecommunications, and geography.

The School of Information (SI) at the University of Michigan has offered master's and Ph.D. degrees in Information since 1996. In 2008 SI joined with the Computer Science and Engineering Division, and the College of Literature, Science and Arts, to offer a joint undergraduate informatics degree. To enter the major, students are required to complete one prerequisite each in calculus, programming, statistics, and information science. They then take a 16-credit core in discrete math, data structures,

statistics, and information technology ethics. Each then selects a several-course specialization track, which is interdisciplinary but focuses on providing depth in a particular domain: computational informatics, information analysis, life science informatics, or social computing. This program establishes a strong foundation, domain depth and interdisciplinary training. However, to accomplish all of this, it also imposes on students the heaviest required-credit burden of any liberal arts major.

The equal participation by the Computer Science and Engineering Division in the Michigan degree emphasizes the ability to design an informatics program as a complement to a traditional computer science degree; indeed, the Computer Science and Engineering Division continues to offer two traditional CS bachelor's degrees (one in engineering, one in liberal arts). One advantage expected for the contextualized informatics degree is higher enrollment of women, and indeed, about half the class of declared majors is female. On the downside, managing a degree that spans three colleges and schools is challenging, with natural hurdles such as teaching budgets and credit approvals across units.

Looking Forward

Informatics curricula are young and developing, but have proven popular. Indiana has over 400 students in the major. In just its first year, Michigan attracted 40 undergraduate majors. Evidence comes also from successful courses offered outside a formal informatics program. For example, a computer scientist and an economist at Cornell enroll about 300 students annually in interdisciplinary "Networks," which counts toward the majors in Computer Science, Economics, Sociology, and Information Science.[a] At the University of Pennsylvania, "Networked Life" (taught by a computer scientist) attracts about 200 students, and satisfies requirements in three majors: Philosophy, Politics, and Economics; Science, Technology, and Society; and Computer and Information Science.[b]

Informatics enables students to combine passions for both computation and another domain. Since almost all domains now benefit from computational thinking, an informatics program can embrace students and concentrations in art and design, history, linguistics, biology, sociology, statistics, and economics. This diversity has costs, of course. One is that for now, in the early years, students and faculty must continuously explain "informatics" to potential employers. Another is providing strong enough foundations in both computation and another discipline to produce competitive, successful graduates.

The desire to deeply understand how computing works is what has drawn most researchers to study computer science. These same individuals are then invested with the responsibility to develop curricular programs and teach computing to the next generation of computing professionals. The current (and all future) generations of students entering the university have largely grown up in a world where computing is so commonplace that it is taken for granted. Many of them are less interested in how computing works than in how to make it work better in the solution of specific problems, drawn from virtually all other domains of human knowledge. There will always be a need for students who study computer science. Informatics provides a complementary path to reach other students for whom understanding and developing computation *contextually* is crucial to the problems that motivate them. Like mathematics, probability, and logic, in the future computation science will be taught embedded in many other areas. Indeed, informatics is a path within which the technical accomplishments of computer science, mathematics, and statistics become embedded in the ways we interact, imagine, and produce throughout the scope of human experience. C

Dennis P. Groth (dgroth@indiana.edu) is Associate Dean for Undergraduate Studies and Associate Professor of Informatics in the School of Informatics and Computing at Indiana University, Bloomington.

Jeffrey K. MacKie-Mason (jmm@umich.edu) is Associate Dean for Academic Affairs, the Arthur W. Burks Collegiate Professor of Information and Computer Science, and a professor of Economics and Public Policy in the School of Information at the University of Michigan.

a See http://www.infosci.cornell.edu/courses/info2040/2009sp/

b See http://www.cis.upenn.edu/~mkearns/teaching/NetworkedLife/

DOI:10.1145/1646353.1646365

Douglas Maughan

Inside Risks

The Need for a National Cybersecurity Research and Development Agenda

Government-funded initiatives, in cooperation with private-sector partners in key technology areas, are fundamental to cybersecurity technical transformation.

Communications' Inside Risks columns over the past two decades have frequently been concerned with trustworthiness of computer-communication systems and the applications built upon them. This column considers what is needed to attain new progress toward avoiding the risks that have prevailed in the past as a U.S. national cybersecurity R&D agenda is being developed. Although the author writes from the perspective of someone deeply involved in research and development of trustworthy systems in the U.S. Department of Homeland Security, what is described here is applicable much more universally. The risks of not doing what is described here are very significant.

—Peter G. Neumann

President Barack Obama greets White House Cyber Security Chief Howard A. Schmidt, who was appointed in December 2009.

OFFICIAL WHITE HOUSE PHOTO BY LAWRENCE JACKSON

Cyberspace is the complex, dynamic, globally interconnected digital and information infrastructure that underpins every facet of society and provides critical support for our personal communication, economy, civil infrastructure, public safety, and national security. Just as our dependence on cyberspace is deep, so too must be our trust in cyberspace, and we must provide technical and policy solutions that enable four critical aspects of trustworthy cyberspace: security, reliability, privacy, and usability.

The U.S. and the world at large are currently at a significant decision point. We must continue to defend our existing systems and networks. At the same time, we must attempt to be ahead of our adversaries, and ensure future generations of technology will position us to better protect critical infrastructures and respond to attacks from adversaries. Government-funded research and development must play an increasing role toward achieving this goal of national and economic security.

Background

On January 8, 2008, National Security Presidential Directive 54/Homeland Security Presidential Directive 23 formalized the Comprehensive National Cybersecurity Initiative (CNCI) and a series of continuous efforts designed to establish a frontline defense (reducing current vulnerabilities and preventing intrusions), which will protect against the full spectrum of threats by using intelligence and strengthening supply chain security, and shaping the future environment by enhancing our research, devel-

opment, and education, as well as investing in "leap-ahead" technologies.

No single federal agency "owns" the issue of cybersecurity. In fact, the federal government does not uniquely own cybersecurity. It is a national and global challenge with far-reaching consequences that requires a cooperative, comprehensive effort across the public and private sectors. However, as it has done historically, the U.S. government R&D community, working in close cooperation with private-sector partners in key technology areas, can jump-start the necessary fundamental technical transformation.

Partnerships

The federal government must reenergize two key partnerships to successfully secure the future cyberspace: the partnership with the educational system and the partnership with the private sector. The Taulbee Survey[2] has shown that our current educational system is not producing the cyberspace workers of the future and the current public-private partnerships are inadequate for taking R&D results and deploying them across the global infrastructure.

Education. A serious, long-term problem with ramifications for national security and economic growth is looming: there are not enough U.S. citizens with computer science (CS) and science, technology, engineering, and mathematics (STEM) degrees being produced. The decline in CS enrollments and degrees is most acute. The decline in undergraduate CS degrees portends the decline in master's and doctoral degrees as well. Enrollments in major university CS departments have fallen sharply in the last few years, while the demand for computer scientists and software engineers is high and growing. The Taulbee Survey[2] confirmed that CS (including computer engineering) enrollments are down 50% from only five years ago, a precipitous drop by any measure. Since CS degrees are a subset of the overall requirement for STEM degrees and show the most significant downturn, CS degree production can be considered a bellwether to the overall condition and trend of STEM education. The problems with other STEM degrees are equally disconcerting and require immediate and effective action. At the same time, STEM jobs are growing, and CS jobs are growing faster than the national average.

At a time when the U.S. experiences cyberattacks daily and as global competition continues to increase, the U.S. cannot afford continued ineffective educational measures and programs. Revitalizing educational systems can take years before results are seen. As part of an overall national cybersecurity R&D agenda, the U.S. must incite an extraordinary shift in the number of students in STEM education quickly to avoid a serious shortage of computer scientists, engineers, and technologists in the decades to come.

Public-Private Partnerships. Information and communications networks are largely owned and operated by the private sector, both nationally and internationally. Thus, addressing cybersecurity issues requires public-private partnerships as well as international cooperation. The public and private sector interests are dependent on each other and share a responsibility for ensuring a secure, reliable infrastructure. As the federal government moves forward to enhance its partnerships with the private sector, research and development must be included in the discussion. More and more private-sector R&D is falling by the wayside and, therefore, it is even more important that government-funded R&D can make its way to the private sector, given it designs, builds, owns, and operates most of the critical infrastructures.

Technical Agenda

Over the past decade there have been a significant number of R&D agendas published by various academic and industry groups, and government departments and agencies (these documents can be found online at http://www.cyber.st.dhs.gov/documents.html). A 2006 federal R&D plan identified at least eight areas of interest with over 50 project topics that were either being funded or should be funded by federal R&D entities. Many of these topic areas have been on the various lists for over a decade. Why? Because the U.S. has underinvested in these R&D areas, both within the government and private R&D communities.

The current public-private partnerships are inadequate for taking R&D results and deploying them across the global infrastructure.

The Comprehensive National Cyber Initiative (CNCI) and the President's Cyberspace Policy Review[3] challenged the federal networks and IT research community to figure out how to "change the game" to address these technical issues. Over the past year, through the National Cyber Leap Year (NCLY) Summit and a wide range of other activities, the U.S. government research community sought to elicit the best ideas from the research and technology community. The vision of the CNCI research community over the next 10 years is to "transform the cyber-infrastructure to be resistant to attack so that critical national interests are protected from catastrophic damage and our society can confidently adopt new technological advances."

The leap-ahead strategy aligns with the consensus of the U.S. networking and cybersecurity research communities: That the only long-term solution to the vulnerabilities of today's networking and information technologies is to ensure that future generations of these technologies are designed with security built in from the ground up. Federal agencies with mission-critical needs for increased cybersecurity, which includes information assurance as well as network and system security, can play a direct role in determining research priorities and assessing emerging technology prototypes.

The Department of Homeland Security Science and Technology Directorate has published its own roadmap in an effort to provide more R&D direction for the community. The Cybersecurity Research Roadmap[1] addresses a broad R&D agenda that is required to enable production of the technologies that will protect future information systems and

networks. The document provides detailed research and development agendas relating to 11 hard problem areas in cybersecurity, for use by agencies of the U.S. government. The research topics in this roadmap, however, are relevant not just to the governments, but also to the private sector and anyone else funding or performing R&D.

While progress in any of the areas identified in the reports noted previously would be valuable, I believe the "top 10" list consists of the following (with short rationale included):

1. Software Assurance: poorly written software is at the root of all of our security problems;
2. Metrics: we cannot measure our systems, thus we cannot manage them;
3. Usable Security: information security technologies have not been deployed because they are not easily usable;
4. Identity Management: the ability to know who you are communicating with will help eliminate many of today's online problems, including attribution;
5. Malware: today's problems continue because of a lack of dealing with malicious software and its perpetrators;
6. Insider Threat: one of the biggest threats to all sectors that has not been adequately addressed;
7. Hardware Security: today's computing systems can be improved with new thinking about the next generation of hardware built from the start with security in mind;
8. Data Provenance: data has the most value, yet we have no mechanisms to know what has happened to data from its inception;
9. Trustworthy Systems: current systems are unable to provide assurances of correct operation to include resiliency; and
10. Cyber Economics: we do not understand the economics behind cybersecurity for either the good guy or the bad guy.

Life Cycle of Innovation

R&D programs, including cybersecurity R&D, consistently have difficulty in taking the research through a path of development, testing, evaluation, and transition into operational environments. Past experience shows that transition plans developed and applied early in the life cycle of the research program, with probable transition paths for the research product, are effective in achieving successful transfer from research to application and use. It is equally important, however, to acknowledge that these plans are subject to change and must be reviewed often. It is also important to note that different technologies are better suited for different technology transition paths and in some instances the choice of the transition path will mean success or failure for the ultimate product. There are guiding principles for transitioning research products. These principles involve lessons learned about the effects of time/schedule, budgets, customer or end-user participation, demonstrations, testing and evaluation, product partnerships, and other factors.

In order to achieve the full results of R&D, technology transfer needs to be a key consideration for all R&D investments.

A July 2007 U.S. Department of Defense Report to Congress on Technology Transition noted there is evidence that a chasm exists between the DoD S&T communities and acquisition of a system prototype demonstration in an operational environment. DOD is not the only government agency that struggles with technology transition. That chasm, commonly referred to as the "valley of death," can be bridged only through cooperative efforts and investments by both research and acquisition communities.

There are at least five canonical transition paths for research funded by the federal government. These transition paths are affected by the nature of the technology, the intended end user, participants in the research program, and other external circumstances. Success in research product transition is often accomplished by the dedication of the program manager through opportunistic channels of demonstration, partnering, and sometimes good fortune. However, no single approach is more effective than a proactive technology champion who is allowed the freedom to seek potential utilization of the research product. The five canonical transition paths are:

- Department/Agency direct to Acquisition
- Department/Agency to Government Lab
- Department/Agency to Industry
- Department/Agency to Academia to Industry
- Department/Agency to Open Source Community

In order to achieve the full results of R&D, technology transfer needs to be a key consideration for all R&D investments. This requires the federal government to move past working models where most R&D programs support only limited operational evaluations and experiments. In these old working models, most R&D program managers consider their job done with final reports, and most research performers consider their job done with publications. In order to move forward, government-funded R&D activities must focus on the real goal: technology transfer, which follows transition. Current R&D principal investigators (PIs) and program managers (PMs) aren't rewarded for technology transfer. Academic PIs are rewarded for publications, not technology transfer. The government R&D community must reward government program managers and PIs for transition progress.

Conclusion

As noted in the White House Cyberspace Policy Review,[3] an updated national strategy for securing cyberspace is needed. Research and development must be a full partner in that discussion. It is only through innovation creation that the U.S. can regain its position as a leader in cyberspace. C

References

1. A Roadmap for Cybersecurity Research, Department of Homeland Security Science and Technology Directorate, November 2009; http://www.cyber.st.dhs.gov/documents.html
2. Taulbee Survey 2006–2007, Computing Research News 20, 3. *Computer Research Association*, May 2008.
3. White House Cyberspace Policy Review; http://www.whitehouse.gov/assets/documents/Cyberspace_Policy_review_final.pdf

Douglas Maughan (Douglas.Maughan@dhs.gov) is a program manager for cybersecurity R&D at the U.S. Department of Homeland Security in Washington, D.C.

DOI:10.1145/1646353.1646367

Michel Beaudouin-Lafon

Viewpoint
Open Access to Scientific Publications

The good, the bad, and the ugly.

In his July 2009 *Communications* editor's letter "Open, Closed, or Clopen Access?", editor-in-chief Moshe Vardi addressed the question of open access to this magazine and to ACM publications in general. Scientific publishing, like all areas of publishing, is undergoing major changes. One reason is the advent of the Internet, which fosters new types of publishing models. Another less-known factor is the exponential increase in the number of scientific publications (see the figure here), which has turned this area into a serious business. In this column, I take a look at commercial and Open Access publishing, and at the role that professional societies such as ACM can play in this evolving world.

PHOTOGRAPH BY ANNA CREECH. OPEN ACCESS LOGO BY PUBLIC LIBRARY OF SCIENCE

Commercial Publishing

Scientific publishing is a profitable business: at more than 30%, the operating profit margins of major commercial publishers are one of the highest across all businesses.[a] A major consequence has been a massive concentration of commercial editors of scientific, technical, and medical (STM) publications, with one giant (Elsevier) and a few big players (Springer, Thomson, Wiley). This concentration has coincided with sharp increases in subscription rates, and has generated razor-sharp business practices whereby, for example, publishers sell subscriptions to a bundle of titles that typically contain one or two good journals among a set of second-tier ones.

The quality of a journal is typically measured by its impact factor—the average number of citations to articles in this journal over a unit time (typically three years). Because of the competition among publishers, impact factors can be, and are, manipulated: Commercial publishers ask their editors-in-chief to "encourage" authors of accepted papers to include references to their journals. (Since they pay their editors-in-chief, it makes them more "receptive" to such requests.) The Web-based version of EndNote, the well-known

a See, for example, http://www.researchinformation.info/features/feature.php?feature_id=141

reference searching tool, facilitates references to publications indexed by ISI Web-of-Science, the division of Thomson that computes the very impact factors mentioned previously.

Over the years, commercial STM publishing has become a cutthroat business with cutthroat practices and we, the scientific and academic community, are the naive lambs, blinded by the ideals of science for the public good—or simply in need of more publications to advance our careers.

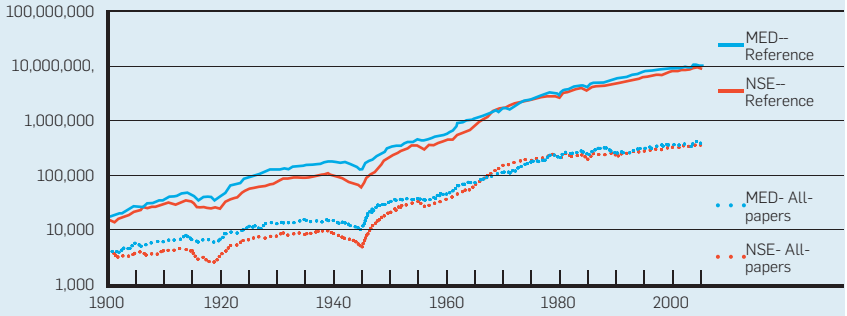
Fortunately, a number of researchers and academic leaders woke up one day and said: "We do not need commercial publishers. We want the results of our research, which is often funded by taxpayers' money, to be available for free to the public at large. With the Internet, the costs of publishing are almost zero, and therefore we can make this work." And so was born the white knight of STM publishing: Open Access.

Open-Access Publishing

But the proponents of Open Access quickly realized that online publishing is not free, nor cheap. Management, equipment, and access costs add up quickly. For example, ACM spends several million dollars every year to support the reliable data center serving the Digital Library[b] and to incorporate new data, improve cross-references, and develop new services.

Since Open Access needs funding, where can it come from?[c] An obvious answer is advertising, but it is not a sustainable option at least for now. A less obvious answer, but one that is quickly gaining momentum, is called *author charges* (or *publication fees*): since Open Access does not charge readers, authors will pay to publish their works. This should be painless for authors because they are also readers: it simply transfers charges from subscriptions to authorship. In fact, the proponents of this model explicitly encourage researchers to include author charges in their budgets when they apply for grants. The NIH explicitly supports Open Access and accepts such costs.

Exponential increase in the scientific production in the medical (MED) and natural sciences and engineering (NSE) fields. The vertical scale is logarithmic. The number of published articles for 2004 is about 500,000 and the number of references is about 10 million. Data provided by Yves Gingras.



But how much are authors ready to pay to publish an article? A few hundred dollars? The most prominent Open Access publisher, the Public Library of Science (PLOS), is a nonprofit organization that has received several million dollars in donations. Yet it charges between $1,350 and $2,900 per paper, depending on the journal.[d] In fact, many in the profession estimate that to be sustainable, the author-pay model will need to charge up to $5,000–$8,000 per publication.

Consider what this means. For example, I am the head of the Laboratory for Computer Science at Université Paris-Sud in France. We publish over 100 journal articles annually. At the conservative estimate of $2,500 per article, the author fees would cost us $250,000 per year. This is more than four times our current budget for journal subscriptions. And, of course, since not every publisher is going to turn to that model overnight, we would have to keep traditional subscriptions. At $5,000 per publication, my lab is broke.

Funding agencies are unlikely to cover these extra costs. If they do, it will be within the same overall budget, meaning less money for manpower, equipment, and travel. Also, how would funding agencies pay for papers published after the end of a grant, as is often the case with journal publications? How would researchers decide between two papers when budgets are tight? More than ever, the rich will be able to publish more and more easily than the poor. And even though Open Access publishers do have policies to lower or waive the fees for those who cannot pay, it is embarrassing just to have to ask.

In fact, those who benefit the most from this model are neither the scientific community nor the general public. They are the big pharmaceutical labs and the tech firms who publish very little but rely on the publication of scientific results for their businesses.[e] With author-pay, research will pay so that industry can get their results for free. Is this moral? The only other area in publishing where authors pay to get published is called the vanity press. Do we really want to enter that model?

Not surprisingly, commercial publishers have considered Open Access a potential threat. But they quickly realized that the author-pay model could work for them, too. Many publishers are already testing a dual-model: authors can publish an article without charge, in which case it is available to subscribers only, or with an author-charge, in which case it is available for free. This is the best of both worlds: charging *both* readers and authors!

So while Open Access was designed to provide an alternative to commercial publishing, it may well be consumed by it. Now, authors, not just readers, are the publishers' market.

b As an example, on Sept. 11, 2001, ACM was prepared to switch to a backup database in another location in the country to provide uninterrupted access to the Digital Library.

c See http://www.arl.org/sparc/publisher/incomemodels/ for a fairly complete list.

d See http://www.plos.org/journals/pubfees.html

e Elsevier has admitted to creating fake journals sponsored by pharmaceutical labs (see, for example, http://www.the-scientist.com/blog/display/55679/)

For example, I can easily imagine these publishers soon offering universities special deals with reduced author fees in exchange for exclusive rights to the publications of that university, jeopardizing academic freedom.

The Role of Professional Societies

Can we get out of this situation? Can we escape both the escalating subscription fees of commercial editors and the dangerous author fees of prominent Open Access publishers? It is important to understand that the scientific community is largely at fault: we sit on the editorial boards of the very journals published at exorbitant prices by commercial publishers,[f] and we submit our best articles to these journals.

The problem with the subscription model is not the model but the fees. Rob Kirby, of the UC Berkeley Math Department, has compared the cost-per-page of various mathematics journals, computed as the subscription price divided by the number of pages published annually.[g] In 1997, they ranged from $0.07 to $1.53. The cost per 10,000 characters, which better accounts for differences among journal formats, ranged from 30 cents to $3. Consistently, the cheaper journals are published by universities and societies; the most expensive ones by commercial publishers. In 2003, Donald Knuth, editor of *Journal of Algorithms*, wrote a long letter[h] to his editorial board explaining that the price per page of the journal had more than doubled since it had been acquired by Elsevier, while it had stayed stable over the previous period, when it was published by Academic Press. This led to a mass resignation of the board and the rebirth of the journal as *ACM Transactions on Algorithms*. Another well-known example is the *Journal of Machine Learning Research*, which became its own Open Access publisher for similar reasons. A number of journals have joined this trend,[i] but few have turned to Open Access.

Open Access is a valuable goal, but the scientific community is overly naive about the whole business of scientific publishing.

So, am I against Open Access? No. I think it is a noble goal, an achievable goal. But this goal should not blind us to the point of making a bad system even worse, of hurting research in the name of making its results freely available to everyone.

First, scientific publications can be affordable. The pricing of the ACM Digital Library is extremely low, even compared to other societies and nonprofit organizations. This is still not enough. The pricing model is adequate for the academic and industry audience but not for dissemination toward the public at large. As shown by the success of online stores such as iTunes, low-pricing can translate into large volumes. Commercial publishers charge non-subscribers up to $30 to download a single paper; ACM charges $15. What if it were 99 cents? While I am not saying that scientific publishing is a mass market like music, I do believe this would dramatically reduce the barrier to non-subscribers, in particular the general public, without significantly affecting the revenues from subscriptions.

Second, much of this debate has focused on cost. But free access is, to paraphrase the Free Software Movement, as much about free beer as it is about free speech. Many publishers, including ACM, allow their authors to publish copies of their articles on their personal Web page or on their institutional repository.[j] But the transfer of copyright is seen by some as a serious hindrance to open access, as it deprives authors from distribution rights. While copyright transfer offers authors protection (such as against plagiarism) and services (such as authorization to reprint), I believe switching to a licensing model such as Creative Commons could be beneficial.

The added value provided by publishers is twofold: reputation (the value of the imprimatur), and archiving (the guarantee that the work will be available forever). These allow publishers to provide services that self-publishing and even institutional repositories cannot provide, such as the author pages that were recently added to the ACM Digital Library. Little if any of this relies on the actual transfer of copyright. While publishers value the exclusivity granted by copyright transfer, users (authors and readers alike) value the services that make articles easier to find: indexing, cross-referencing, searching, and so forth. A proper licensing model could foster novel services for scientific dissemination, including by third parties, without challenging the primary values and revenue streams of publishers, in particular non-profit ones.

Conclusion

Open Access is a valuable goal, but the scientific community is overly naive about the whole business of scientific publishing. Societies and nonprofit organizations need to continue to lead the way to improve the dissemination of research results, but the scientific community at large must support them against the business-centric views of commercial publishers. Author fees are not a solution. Worse, they jeopardize the ecological balance of the research incentive structure. Finally, nonprofit publishers should take advantage of their unique position to experiment with sustainable evolutions of their publishing models. C

f I am an associate editor of an Elsevier-published journal.

g See http://math.berkeley.edu/~kirby/journals.html

h See http://www-cs-faculty.stanford.edu/~knuth/joalet.pdf

i For a list, see http://oad.simmons.edu/oadwiki/Journal_declarations_of_independence

j See section 2.5 of the ACM copyright policy, http://www.acm.org/publications/policies/copyright_policy, and the SHERPA/ROMEO list of publishers' copyright and self-archiving policies, http://www.sherpa.ac.uk/romeo/

Michel Beaudouin-Lafon (mbl@lri.fr) is a professor of computer science at Université Paris-Sud (France) and head of the Laboratory for Computer Science (LRI). He is also a former member of the ACM Council and the ACM Publications Board.

The author thanks Wendy Mackay and Bernard Rous for comments on earlier versions of this column.

DOI:10.1145/1646353.1646366

George V. Neville-Neil

Article development led by acmqueue
queue.acm.org

Kode Vicious
Taking Your Network's Temperature

A prescription for capturing data to diagnose and debug a networking problem.

Dear KV,

I posted a question on a mailing list recently about a networking problem and was asked if I had a tcpdump. The person who responded to my question—and to the whole list as well—seemed to think my lack of networking knowledge was some kind of affront to him. His response was pretty much a personal attack: If I couldn't be bothered to do the most basic types of debugging on my own, then I shouldn't expect much help from the list. Aside from the personal attack, what did he mean by this?

Dumped

Dear Dumped,

It is always interesting to me that when people study computer programming or software engineering they are taught to use the creative tools—editors to create code, compilers to take that code and turn it into an executable—but are rarely, if ever, taught how to debug a program. Debuggers are powerful tools, and once you learn to use one you become a far more productive programmer because, face it, putting `printf()`—or its immoral equivalent—throughout your code is a really annoying way to find bugs. In many cases, especially those related to timing issues, adding `print` statements just leads to erroneous results. If the number of people who actually learn how to debug a program during their studies is small, the number who learn how to debug a *networking* problem is minuscule. I actually don't know anyone who was ever directly taught how to debug a networking problem.

Some people—the lucky ones—are eventually led to the program you mention, tcpdump, or its graphical equivalent, wireshark, but I've never seen anyone try to teach people to use these tools. One of the nice things about tcpdump and wireshark is that they're multi-platform, running on both Unix-like operating systems and Windows. In fact, writing a packet-capture program is relatively easy, as long as the operating system you're working with gives you the ability to tap into the networking code or driver at a low enough level to sniff packets.

Those of us who spend our days banging our heads against networking problems eventually learn how to use

these tools, sort of in the way that early humans learned to cook flesh. Let's just say that though the results may have been edible, they were not winning any Michelin stars.

Using a packet-capture tool is, to a networking person, somewhat like using a thermometer is to a parent. It is likely that if you ever felt sick when you were a child at least one of your parents would take your temperature. If they took you to the doctor, the doctor would also take your temperature. I once had my temperature taken for a broken ankle—crazy, yes, but that doctor gave the best prescriptions, so I just smiled blithely and let him have his fun. That aside, taking a child's temperature is the first thing on a parent's checklist for the question "Is my child sick?" What on earth does this have to do with capturing packets?

By far the best tool for determining what is wrong with programs that use a network, or even the network itself, is the tcpdump tool. Why is that? Surely in the now 40-plus years since packets were first transmitted across the original ARPANET we have developed some better tools. The fact is we have not. When something in the network breaks, you want to be able to see the messages at as many layers as possible.

The other key component in debugging network problems is understanding the timing of what happens, which a good packet-capture program also records. Networks are perhaps the most nondeterministic components of any complex computing system. Finding out who did what to whom and when (another question parents often ask, usually after a fight among siblings) is extremely important.

All network protocols, and the programs that use them, have some sort of ordering that is important to their functioning. Did a message go missing? Did two or more messages arrive out of order at the destination? All of these questions can potentially be answered by using a packet sniffer to record network traffic, but *only if you use it*!

It's also important to record the network traffic as soon as you see the problem. Because of their nondeterministic nature, networks give rise to the worst types of timing bugs. Perhaps the bug happens only every so many hours, because of a rollover in a large counter; you really want to start recording the network traffic *before* the bug occurs, not after, because it may be many hours until the condition comes up again.

Using a packet-capture tool is, to a networking person, somewhat like using a thermometer is to a parent.

So, here are some very basic recommendations on using a packet sniffer in debugging a network problem. First, get permission (yes, it really is KV giving you this advice). People get cranky if you record their network traffic, such as instant messages, email, and banking transactions, and then post it to a mailing list. Just because some person in IT was dumb enough to give you root or admin rights on your desktop does not mean you should just record everything and send it off.

Next, record only as much information as you need to debug the problem. If you're new at this you'll probably have the program suck up every packet so you don't miss anything, but that's problematic for two reasons: the first is the previously mentioned privacy issue; and the second is that if you record too much data, finding the bug will be like finding a needle in a haystack—only you've never seen a haystack that big. Recording an hour of Ethernet traffic on your LAN can capture a few hundred million packets. No matter how good a tool you have, it's going to do a much better job at finding a bug if you narrow down the search.

If you do record a lot of data, don't try to share it all as one huge chunk. See how these points follow each other? Most packet-capture programs have options to say, "Once the capture file is full, close it and start a new one." Limiting files to one megabyte is a nice start.

Finally, do *not* record your data on a network file system. There is no better way to ruin a whole set of packet-capture files than by having them capture themselves.

So there you have it: a brief introduction to capturing data so you can debug a networking problem. Perhaps now you can get yelled at on a mailing list for something more egregious than not taking your network's temperature before calling the doctor.

KV

Related articles on queue.acm.org

Debugging in an Asynchronous World
Michael Donat
http://queue.acm.org/detail.cfm?id=945134

Kode Vicious Bugs Out
Kode Vicious
http://queue.acm.org/detail.cfm?id=1127862

A Conversation with Steve Bourne, Eric Allman, and Bryan Cantrill
http://queue.acm.org/detail.cfm?id=1413258

Dedication

I would like to dedicate this column to my first editor, Mrs. B. Neville-Neil, who passed away after a sudden illness on December 9th, 2009; she was 65 years old.

My mother took language, both written and spoken, very seriously. The last thing I wanted to hear upon showing her an essay I was writing for school was, "Bring me the red pen." In those days I did not have a computer; all my assignments were written longhand or on a typewriter and so the red pen meant a total rewrite. She was a tough editor, but it was impossible to question the quality of her work or the passion that she brought to the writing process. All of the things Strunk and White have taught others throughout the years my mother taught me, on her own, with the benefit of only a high school education and a voracious appetite for reading.

It is, in large part, due to my mother's influence that I am a writer today. It is also due to her influence that I review articles, books, and code on paper, using a red pen. Her edits and her unswerving belief that I could always improve are, already, keenly missed.

—George Vernon Neville-Neil III

DOI:10.1145/1646353.1646369

Dennis Shasha

Interview

An Interview with Michael Rabin

Michael O. Rabin, co-recipient of the 1976 ACM A.M. Turing Award, discusses his innovative algorithmic work with Dennis Shasha.

I FIRST ENCOUNTERED Michael Rabin in 1980 when I was a first-year Ph.D. student at Harvard University. It was four years after Rabin and Dana Scott won the ACM A.M. Turing award for their foundational work on deterministic and nondeterministic finite state automata. By 1980, however, Rabin's interests were all about randomized algorithms. His algorithms course was a challenge to all of us. Sometimes he presented a result he had worked out only two weeks earlier. When I had difficulty understanding, I would make puzzles for myself. As a consequence, when I published my first puzzle book *The Puzzling Adventures of Dr. Ecco*, I dedicated the book to Michael as one of my three most influential teachers. At Harvard, I enjoyed every encounter with Michael—in seminars and at parties. He was a great raconteur and a great joker. In 1994, journalist Cathy Lazere and I embarked on the writing of *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. The goal was to interview great living seminal thinkers of our field: Knuth, Rabin, Dijkstra, among others. Each thinker was associated with an epithet: Michael's was "the possibilities of chance."

—Dennis Shasha, Courant Institute, New York University,

PHOTOGRAPH COURTESY OF MICHAEL RABIN

Shasha: I'm going to try to get to a mixture of personal and technical recollections, let's start in Israel. You finished high school, and what were you thinking about doing after high school?

Rabin: I've been intensely interested in mathematics ever since I was 11 years old, when I was kicked out of class and told to stand in the hall. I encountered two ninth-graders who were working on geometry problems and asked them what they were doing. They said they had to prove such-and-such a statement, and then condescendingly said to me, "You think you can do it?" Even though I never studied geometry, I was able to solve that problem. The fact that you can by pure thought prove statements about the real world of lines, circles, and distances impressed me so deeply that I decided I wanted to study mathematics.

My father sent me to the best high school in Haifa, where I grew up, and in 10th grade I had the good luck to encounter a real mathematician, Elisha Netanyahu (the uncle of Benjamin Netanyahu) who was at the time a high school teacher. Later he became a professor at the Haifa Institute of Technology. He conducted once a week a so-called "mathematical circle," where he taught a selected group of students number theory, combina-

torics, and advanced algebra. Netanyahu lent me advanced mathematics books. Starting at age 15, I read Hardy and Wright's book *An Introduction to the Theory of Numbers*, the first volume in German of Landau's *Zahlentheorie*. I studied a wonderful book by G.H. Hardy called *Pure Mathematics*, which was mainly analyses, two volumes of Knopp's *Functions of a Complex Variable*, A. Speiser's book *Gruppentheorie*, and so on.

I finished high school at age 16½ because the war of independence broke out in Israel, and everybody in my class was drafted into the army. I continued to study mathematics on my own while in the army. Then I got in touch with Abraham Fraenkel, who was a professor of mathematics in Jerusalem and whose book on set theory I had studied. That was maybe in September of 1949. Fraenkel met and tested me on group theory, number theory, set theory, and decided that it would be quite worthwhile if I came to the university. He got in touch with army authorities, and I was discharged from the army to go and study at the university.

The system was that you went directly to a master's degree. I studied a lot of mathematics: set theory, algebra, functions of a complex variable, linear spaces, differential equations, probability as well as physics and applied mathematics. I was mainly interested in algebra. So in 1952 I wrote a master's thesis on the algebra of commutative rings, and solved an open problem due to Emmy Noether, giving a necessary and sufficient condition on a commutative ring for having the property that every ideal is a finite intersection of primary ideals. That was one of my first papers. It appeared in the *Comptes Rendus* (the proceedings) of the French Academy of Sciences.

Princeton Days and String Automata Theory

I went to Princeton University to study with Alonzo Church. At the time, only 13 Ph.D. students were admitted every year to the Mathematics Department. I found out the committee that was considering me assigned somebody to look at the master's thesis paper, and this is what led to my admission. My Ph.D. thesis was on computational (recursive) unsolvability of group theoretical problems, thus combining my interests in algebra and computability.

In 1957, IBM decided to go into research in a big way. They sent people to recruit promising students from top universities and top mathematics departments. Dana Scott and I were invited to spend the summer at IBM Research. Watson Labs did not exist at that time, and research was located at the Lamb Estate in Westchester County. This had previously been an insane asylum where, rumor had it, rich families would forcibly confine troublesome members.

At the Lamb Estate, Dana and I wrote the paper "Finite Automata and Their Decision Problems." Automata theory had started with a study by Walter Pitts and Warren McCulloch of what would now be called neural networks. They assumed that those neural networks essentially embodied a finite number of states, and talked about what is recognizable by such finite state neural networks, but they didn't have a complete characterization. Later, in 1956, S.C. Kleene invented regular languages: sets of strings that are obtained from finite sets by certain simple operations. Kleene showed that finite neural nets compute or recognize exactly the regular languages.

We decided to completely abstract away from neural nets and consider a finite-state machine that gets symbol inputs and undergoes state transitions. If upon being fed a string of symbols it passes from an initial state to one of a number of accepting states, then the string is accepted by the finite-state machine. The finite-state machine had no output except for saying yes/no at the end.

I set myself the goal of finding more and more applications of randomization in mathematics and computer science.

Dana and I asked ourselves in what ways could we expand the finite-state machine model. One of our extensions was to consider nondeterministic machines. We really did not have any deep philosophical reason for considering nondeterminism, even though as we now know nondeterminism is at the center of the P = NP question, a problem of immense practical and theoretical importance. For us, it was just one of the variants. We stipulated that the automaton when in state S and upon input symbol sigma can go into any one of a number of states S', S",..., comprising a certain subset of the set of states. The nondeterministic finite-state machine accepts a string if there is a possible computation, a possible sequence of transitions, leading to an accepting state. Then we proved that finite-state nondeterministic automata are exactly equal in power to deterministic automaton computations. [On a practical level, this means that the wildcard searches of say grep, perl, or python can be expressed as nondeterministic finite state automata and then can be translated into deterministic finite state automata.]

The corresponding question whether nondeterministic polynomial time Turing machine computations are equal in power to deterministic polynomial time Turing machine computations is the famous P = NP problem.

Employing nondeterministic automata, we were able to re-prove Kleene's result that finite state machines exactly accept regular languages. The proof became very easy. We also introduced and studied other extensions of finite automata such as two-way automata, multi-tape and linearly bounded automata. The latter construct appeared in our research report but did not find its way into the published paper.

Origins of Complexity Theory

The next summer I again went to the Lamb Estate. At that time there was a methodologically misled widely held view that computing, and what later became computer science, was a sub-field of information theory in the Shannon sense. This was really ill-con-

ceived because Shannon was dealing with the information content of messages. If you perform a lengthy calculation on a small input then the information content in the Shannon sense of the outcome is still small. You can take a 100-bit number and raise it to the power 100, so you get a 10,000-bit number. But the information content of those 10,000 bits is no larger than that of the original 100 bits.

John McCarthy posed to me a puzzle about spies, guards, and password. Spies must present, upon returning from enemy territory, some kind of secret password to avoid being shot by their own border guards. But the guards cannot be trusted to keep a secret. So, if you give them the password, the enemy may learn the password and safely send over his own spies. Here is a solution. You randomly create, say, a 100-digit number x and square it, and give the guards the middle 100 digits of x2. John von Neumann had suggested the middle square function for generating pseudo-random numbers. You give the number x to the spy. Upon being challenged, the spy presents x. The guard computes x2 and compares the middle square to the value he has. Every password x is used only once. The whole point is that presumably it is easy to calculate the middle square, but it is difficult, given the middle square, to find one of the numbers having that value as a middle square. So even if the guards divulge the middle square, nobody else can figure out the number the spy knows.

But how do you even define that difficulty of computing? And even more so, how do you prove it? I then set myself to study that problem. I wrote an article called "Degree of Difficulty of Computing a Function and Hierarchy of Recursive Sets." In that article, I wasn't able to solve the problem of showing that the von Neumann function is difficult to invert. This is really a special case of the P = NP problem. It hasn't been settled to this day. But I was able to show that for every computable function there exists another computable function that is more difficult to compute than the first one, regardless of the algorithm or programming language one chooses. It's similar to the minimum energy required for performing a physical task. If this phone is on the floor and I have to raise it, there is a minimum amount of work. I can do it by pulling it up, by putting a small amount of explosive, blowing it up here, but there is a certain inherent amount of work. This is what I was studying for computations.

The next significant application of my work on randomization was to cryptography.

I think this paper, no less than the Rabin/Scott paper, was a reason for my Turing Award. The ACM announcement of the Turing Award for Dana and for me mentioned the work on finite automata and other work we did and also suggested that I was the first person to study what is now called complexity of computations.

Randomized Algorithms: A New Departure

I went back to Jerusalem. I divided my research between working on logic, mainly model theory, and working on the foundations of what is now computer science. I was an associate professor and the head of the Institute of Mathematics at 29 years old and a full professor by 33, but that was completely on the merit of my work in logic and in algebra. There was absolutely no appreciation of the work on the issues of computing. Mathematicians did not recognize the emerging new field.

In 1960, I was invited by E.F. Moore to work at Bell Labs, where I introduced the construct of probabilistic automata. These are automata that employ coin tosses in order to decide which state transitions to take. I showed examples of regular languages that required a very large number of states, but for which you get an exponential reduction of the number of states if you go over to probabilistic automata.

Shasha: And with some kind of error bound?

Rabin: Yes, yes, that's right. In other words, you get the answer, but depending upon how many times you run the probabilistic automaton, you have a very small probability of error. That paper eventually got published in 1963 in *Information and Control*.

In 1975, I finished my tenure as Rector (academic head) of the Hebrew University of Jerusalem and came to MIT as a visiting professor. Gary Miller was there and had his polynomial time test for primality based on the extended Riemann hypothesis. [Given an integer n, a test for primality determines whether n is prime.] That test was deterministic, but it depended on an unproven assumption. With the idea of using probability and allowing the possibility of error, I took his test and made it into what's now called a randomized algorithm, which today is the most efficient test for primality. I published first, but found out that Robert Solovay and Volker Strassen were somewhat ahead with a different test. My test is about eight times faster than theirs and is what is now universally being used. In the paper I also introduced the distinction between what are now called Monte-Carlo and Las-Vegas algorithms.

In early 1976 I was invited by Joe Traub for a meeting at CMU and gave a talk presenting the primality test. After I gave that lecture, people were standing around me, and saying, "This is really very beautiful, but the new idea of doing something with a probability of error, however exponentially small, is very specialized. This business of witnesses for compostiness you have introduced is only useful for the primality test. It will never really be a widely used method." Only Joe Traub said, "No, no, this is revolutionary, and it's going to become very important."

From Trick to Fundamental Technique

From then on, I set myself the goal of finding more and more applications of randomization in mathematics and computer science. For example, in 1977 in my MIT class, I presented an algorithm for efficiently expressing an integer as the sum of four squares. Lagrange had proved in 1770 that ev-

ery integer can be so expressed, but there was no efficient algorithm for doing it. In 1977 I found an efficient randomized algorithm for doing that computation. That algorithm later appeared in a joint paper with Jeff Shallit in 1986 together with additional applications of randomization to number theoretical algorithms. Later, I turned my attention to distributed algorithms and found an approach using a random shared bit for solving Byzantine Agreement far more efficiently than previous approaches. Still later I applied randomization to asynchronous fault-tolerant parallel computations in collaboration with Jonatan Aumann, Zvi Kedem, and Krishna Palem.

Right now, if you look at STOC and FOCS [the major conferences in theoretical computer science] maybe a third to half of the papers are built around randomized algorithms. And of course you've got the wonderful book *Randomized Algorithms* by Rajeev Motwani and Prabhaker Raghavan.

Shasha: Let me back up and talk a little bit about the general uses of randomness in computer science. There seem to be at least three streams of use of randomness—yours, the use in communication (for example, exponential back off in the Ethernet protocol), and the use in genetic algorithms where random mutations and random recombination sometimes lead to good solutions of combinatorial problems. Do you see any unified theme among all those three?

Rabin: I would say the following: The use in the Ethernet protocol is in some sense like the use in Byzantine agreement. In Byzantine agreement, the parties want to reach agreement against improper or malicious opponents. In the Ethernet protocol, the participants want to avoid clashes, conflicts. That's somewhat similar. I don't know enough about genetic algorithms, but they are of the same nature as the general randomized algorithms. I must admit that after many years of work in this area, the efficacy of randomness for so many algorithmic problems is absolutely mysterious to me. It is efficient, it works; but why and how is absolutely mysterious.

More recently, I have become interested in protecting the privacy and secrecy of auctions.

It is also mysterious in another way because we cannot really prove that any process, even let's say radioactive decay, is truly random. Einstein rejected the basic tenets of Quantum Theory and said that God does not play dice with the universe. Randomized algorithms, in their pure form, must use a physical source of randomness. So it is cooperation between us as computer scientists and nature as a source of randomness. This is really quite unique and touches on deep questions in physics and philosophy.

Tree Automata

Let me return to a chapter of my work that I skipped before. After the work on finite automata by Dana Scott and me, two mathematicians, Richard Buchi and Calvin Elgot, discovered how the theory of finite automata could be used to solve decision problems in mathematical logic. They showed that the so-called Pressburger arithmetic decision problem could be solved by use of finite automata. Then Buchi went on to generalize finite automata on finite sequences to finite automata on infinite sequences, a very brilliant piece of work that he presented at the Congress on Logic, Philosophy, and Methodology in Science in 1960. In that paper, he showed that the so-called monadic second-order theory of one successor function is decidable. Let me explain very briefly what that means.

We have the integers, 0, 1, 2, 3. The successor function is defined as $S(x) = x+1$. The classical decision problems pertain to problems formulated within a predicate logic—the logic of relations and functions with the quantifiers of "exists" x and "for all" x, and the logical connectives. Monadic second-order theory means that you quantify over sets. Buchi demonstrated that the monadic second-order theory of

one successor function—where you are allowed to quantify over arbitrary subsets of integers—is decidable. His was the first result of that nature.

Buchi posed an open problem: is the monadic second-order theory decidable if you have two successor functions. For example, one successor function being 2x (double), and the other one being 2x+1. I don't know if he realized how powerful the monadic theory of two successor functions would turn out to be. I realized that this is an incredibly powerful theory. If you find a decision procedure for that theory, then many other logical theories can be demonstrated to be decidable.

I set myself to work on this problem. My formulation was a generalization from infinite strings to infinite binary trees. You consider a tree where you have a root, and the root has two children, a child on the left and a right child, and each of these has again two children, a left child and a right child. The tree branches out ad infinitum, forming an infinite binary tree. Consider that tree with the two successor functions, left child, right child, and study the logical theory of the tree with these two functions and quantification over arbitrary subsets of nodes of the tree.

In 1966, I came as visitor to IBM research at Yorktown Heights, and one of my goals to find an appropriate theory of automata on infinite binary trees and prove the decidability of the same problems that Dana Scott and I had shown to be decidable for finite automata on finite strings. I created the appropriate theory of automata on these infinite trees and showed that it was decidable. I consider this to be the most difficult research I have ever done.

A remarkable feature of that original proof is that even though we are dealing with finite automata, and with trees that are infinite but countable, the proof and all subsequent variants employ transfinite induction up to the first uncountable ordinal. Thus the proof is a strange marriage between the finite and countable with the uncountable.

This theory led to decision algorithms for many logical theories. That included decidability of nonstandard logics like modal logics and the temporal logics that are a tool for program verification, especially in the work of Amir Pnueli, Moshe Vardi, Orna Kupferman, and many others.

Great teaching and great science really flow together and are not mutually contradictory or exclusive of each other.

Keeping Secrets

The next significant application of my work on randomization was to cryptography. Ueli Mauer suggested a model of a cryptographic system that is based on what he called the bounded storage assumption. Namely, you have a very intense public source of random bits that everybody can observe, say beamed down from a satellite. The sender and receiver have a short common key that they establish say by meeting, and they use that key in order to select the same random bits out of the public source of randomness. Out of those bits, they construct so-called one-time pads. If you assume that the intensity of the random source is so large that no adversary can store more than, let us say, two-thirds of its bits, then the one-time pad is really completely random to the adversary and can be used for provably unbreakable encryption.

Mauer initially proved the unbreakability result under the assumption that the adversary stores original bits from the random source. However, there remained an open question: suppose your adversary could do some operations on the original bits and then store fewer bits than the number of source bits. Jonatan Aumann, Yan Zong Ding, and I showed that even if the adversary is computationally unbounded, one can still obtain unbreakable codes provided the adversary cannot store all his computed bits. This work created quite a stir and was widely discussed, even in the popular press.

Nowadays however, the bounded storage assumption may not be compelling because the capacity of storage has increased so dramatically. So I posited a new Limited Access Model. Suppose each of 10,000 participants independently store physically random pages. The sender and receiver use a common small random key to randomly select the same 30 page server nodes and from each node download the same randomly selected page. Sender and receiver now XOR those 30 pages to create a one-time pad they employ to encrypt messages. Assume that an adversary cannot listen to or subvert more than say 2,000 of the 10,000 page server nodes. Consequently, his probability of having obtained any particular random page is no more than 1/5, and, since the pages are XORed in groups of 30, his probability of having all of those pages is 1/5 to the power of 30. If the adversary is missing even one of those pages, then the one-time pad is completely random with respect to him, and consequently, if used to encrypt by XORing with the message, the encrypted message is also completely random for that adversary.

Privacy for Pirates and Bidders

In the late 1990s, [Dennis Shasha] came to me and suggested that we work together on devising a system for preventing piracy, to begin with, of software, but later on also music, videos, and so on—any kind of digitized intellectual property. We started by reinventing variants of existing methods, such as the use of safe co-processors and other methods that were actually current at the time and which, by the way, have all either been defeated or impose excessive constraints on use. We then invented a new solution. Our design protects privacy of legitimate purchasers and even of pirates while at the same time preventing piracy of protected digitized content. For example, an engineering firm wanting to win a contest to build a bridge can purchase software for bridge design without identifying itself. Thus competitors cannot find out that the firm

is competing for the project. The purchaser of digital content obtains a tag permitting use of the content. The tag is not tied to a machine identifier so that the tag and use of the content can be moved from machine to machine. Coupled with robust content-identifying software, use of protected content absent a corresponding tag is stopped. The deployment of this robust solution to the piracy problem requires the cooperation of system vendors.

More recently, I have become interested in protecting the privacy and secrecy of auctions. Working with Stuart Shieber, David Parks, and Chris Thorpe, we have a methodology that employs cryptography to ensure honest privacy-preserving auctions. Our protocols enable an auctioneer to conduct the auction in a way that is clearly honest, prevents various subversions of the auction process itself, and later on, when the auction is completed and the auctioneer has determined the winner or winners and how much they pay and how much they get of whatever is being auctioned in multi-item auctions, the auctioneer can publish a privacy-preserving proof for the correctness of the result. In the initial papers, we used the tool of homomorphic encryption. Later on, I had an idea that I then eventually implemented with Chris Thorpe and Rocco Servedio of an entirely new approach to zero knowledge proofs, which is computationally very efficient, does not use heavy-handed and computationally expensive encryption, and achieves everything very efficiently by use of just computationally efficient hash functions.

Teachers, Teaching, and Research

In the life of every creative scientist, you will find outstanding teachers that have influenced him or her and directly or indirectly played a role in their success. My first mentor was a very eminent logician and set theorist, Abraham Halevi Fraenkel. I wrote a master's thesis under the direction of the wonderful algebraist Jacob Levitski. He was a student of maybe the greatest woman mathematician ever, Emmy Noether, the creator of modern abstract algebra as we know it. In Jerusalem, I learned applied mathematics from Menachem Schiffer, a great mathematician and a great teacher.

Powerful algorithms are enabling tools for every computer innovation and application.

Shasha: Let's talk a little bit about the relationship that you've mentioned to me often between teaching and research, because you've won many awards for teaching as well as for science. How do you think academics should view their teaching responsibilities?

Rabin: This is really a very important question. There is this misconception that there is a conflict and maybe even a contradiction between great teaching and being able to do great science. I think this is completely incorrect, and that wonderful teaching, such as Schiffer's teaching, flows from a deep understanding of the subject matter. This is what enables the person also to correctly select what to teach. After all, the world of knowledge, even in specialized subjects, is almost infinite. So one great contribution is to select the topics, and the other great contribution is to really understand the essence, the main motifs, of each particular topic that the person presents, and to show it to the class in a way that the class gets these essential ideas. Great teaching and great science really flow together and are not mutually contradictory or exclusive of each other.

The Future

Shasha: You have contributed to so many areas of computer science. Talk about one or many, and where do you think it is going?

Rabin: Computer science research has undergone a number of evolutions during my research career, and because of the quick pace, one can even say revolutions. To begin with, there was Alan Turing's model of a computing machine, leading to the stored-program computer. Next there was a great emphasis on the study of various mathematical machines. Finite automata are models for sequential circuits. Nondeterministic and deterministic so-called pushdown automata play a pivotal role in the syntactic compilation of programming languages. For about 20 years or so, there was a great emphasis in research on automata, programming languages, and formal languages, also in connection of course with linguistics. There was also a considerable emphasis on efficient algorithms of various kinds. The book by Alfred Aho, John Hopcroft, and Jeff Ullman, and Donald Knuth' classical books are examples of this strong interest in algorithms. The study of algorithms will always remain centrally important. Powerful algorithms are enabling tools for every computer innovation and application.

Then emphasis started to shift toward issues of networking and communication. Even the people who created the Internet, email, and other forms of connectivity perhaps didn't fully realize where all that was going to lead. This evolution, revolution, explosion, started to accelerate, and over the past 10 years we went into a phase where the main use of computers is in the world of information creation, sharing, and dissemination. We have search engines, Wikipedia, Facebook, blogs, and many other information resources, available to everyone literally at their fingertips.

We are only at the beginning of this revolution. One can predict that there is going to be an all-encompassing worldwide network of knowledge where people are going to create, share, and use information in ways that never existed before, and which we can't fully foresee now. These developments are giving rise to a torrent of research in information organization and retrieval, in machine learning, in computerized language understanding, in image understanding, in cryptography security and privacy protection, in multi-agent systems, to name just a few fields. Computer science research will continue to be rich, dynamic, exciting, and centrally important for decades to come.

ACM's Online Books & Courses Programs!

Helping Members Meet Today's Career Challenges

NEW! Over 2,500 Online Courses in Multiple Languages Plus 1,000 Virtual Labs from Element K!

element k®
your learning solutions partner

ACM's new Online Course Collection includes over **2,500 online courses in multiple languages, 1,000 virtual labs, e-reference tools, and offline capability**. Program highlights:

The ACM E-Learning Catalog - round-the-clock access to 2,500+ online courses on a wide range of computing and business topics, in multiple languages.

Exclusive vLab® Virtual Labs - 1,000 unique vLab® exercises place users on systems using real hardware and software allowing them to gain important job-related experience.

Reference Tools - an e-Reference Library extends technical knowledge outside of the classroom, plus online Executive Summaries and quick reference cards to answer on-the-job questions instantly.

Offline Player - members can access assessments and self-study courses offline, anywhere and anytime, without a live Internet connection.

A downloadable Quick Reference Guide and a 15-minute site orientation course for new users are also available to help members get started.

The ACM Online Course Program is open to ACM Professional and Student Members.

---

600 Online Books from Safari

ACM members are eligible for a **special 40% savings** offer to upgrade to a Premium or Full Library subscription.
For more details visit:
http://pd.acm.org/books/about_sel.cfm

The ACM Online Books Collection includes **full access to 600 online books** from Safari® Books Online, featuring leading publishers including O'Reilly. Safari puts a complete IT and business e-reference library right on your desktop. Available to ACM Professional Members, Safari will help you zero in on exactly the information you need, right when you need it.

Safari® Books Online

500 Online Books from Books24x7

All Professional and Student Members also have **full access to 500 online books** from Books24x7®, in ACM's rotating collection of complete unabridged books on the hottest computing topics. This virtual library puts information at your fingertips. Search, bookmark, or read cover-to-cover. Your bookshelf allows for quick retrieval and bookmarks let you easily return to specific places in a book.

books24x7® a SkillSoft Company

Association for Computing Machinery

Advancing Computing as a Science & Profession

pd.acm.org
www.acm.org/join

DOI:10.1145/1646353.1646370

Article development led by acmqueue
queue.acm.org

Power-manageable hardware can help save energy, but what can software developers do to address the problem?

BY ERIC SAXE

Power-Efficient Software

THE RATE AT which power-management features have evolved is nothing short of amazing. Today almost every size and class of computer system, from the smallest sensors and handheld devices to the "big iron" servers in data centers, offer a myriad of features for reducing, metering, and capping power consumption. Without these features, fan noise would dominate the office ambience and untethered laptops would remain usable for only a few short hours (and only then if one could handle the heat), while data-center power and cooling costs and capacity would become unmanageable.

As much as we might think of power-management features as being synonymous with hardware, software's role in the efficiency of the overall system has become undeniable. Although the notion of "software power efficiency" may seem justifiably strange (as software doesn't directly consume power), the salient part is really the way in which software interacts with power-consuming system resources.

Let's begin by classifying software into two familiar ecosystem roles: resource managers (producers) and resource requesters (consumers). We will then examine how each can contribute to (or undermine) overall system efficiency.

The history of power management is rooted in the small systems and mobile space. By today's standards, these systems were relatively simple, possessing a small number of components, such as a single-core CPU and perhaps a disk that could be spun down. Because these systems had few resources, utilization in practice was fairly binary in nature, with the system's resources either being in use—or not. As such, the strategy for power managing resources could also be fairly simple, yet effective.

For example, a daemon might periodically monitor system utilization and, after appearing sufficiently idle for some time threshold, clock down the CPU's frequency and spin down the disk. This could all be done in a way that required little or no integration with the subsystems otherwise responsible for resource management (for example, the scheduler, file system, among others), because at zero utilization, not much resource management needed to be done.

By comparison, the topology of modern systems is far more complex. As the "free performance lunch" of ever-increasing CPU clock speeds has come to an end, the multicore revolution is upon us, and as a consequence, even the smallest portable devices present multiple logical CPUs that need to be managed. As these systems scale larger (presenting more power-manageable resources), partial utilization becomes more common where only part of the system is busy while the rest is idle. Of course, CPUs present just one example of a power-manageable system resource: portions of physical memory may (soon) be power manageable, with the same being true for storage and

I/O devices. In the larger data-center context, the system itself might be the power-manageable resource.

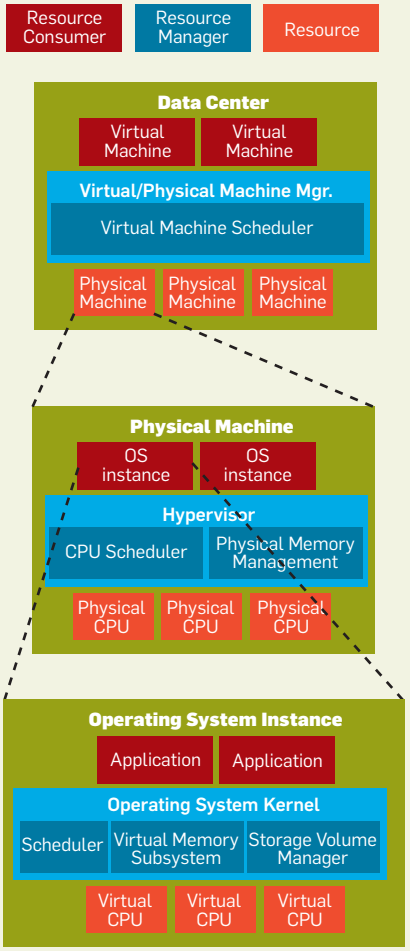
Effective resource management on modern systems requires that there be at least some level of resource manager awareness of the heterogeneity brought on by varying resource power states and, if possible, some exploitation of it. (Actually, effective resource management requires awareness of resource heterogeneity in general, with varying power states being one way in which that resource heterogeneity can arise.) Depending on what is being managed, the considerations could be spatial, temporal, or both.

Spatial Considerations

Spatial considerations involve deciding which resources to provision in response to a consumer's request in time. For an operating-system thread scheduler/dispatcher, this might determine to which CPUs runnable threads are dispatched, as well as the overall optimal distribution pattern of threads across the system's physical processor(s) to meet some policy objective (performance, power efficiency, and so on). For the virtual memory subsystem, the same would be true for how physical memory is used; for a file system/volume manager, the block allocation strategy across disks; and in a data center, how virtual machines are placed across physical systems. These different types of resource managers are shown in Figure 1.

One such spatial consideration is the current power state of available resources. In some sense, a resource's power states can be said to represent a set of trade-offs. Some states provide a mechanism allowing the system to

Figure 1. A hierarchy of resource managers.



trade off performance for power efficiency (CPU frequency scaling is one example), while others might offer (for idle resources) a trade-off of reduced power consumption versus increased recovery latency (for example, as with the ACPI C-states). As such, the act of a resource manager selecting one resource over another (based on power states) is an important vehicle for making such tradeoffs that ideally should complement the power-management strategy for individual resources.

The granularity with which resources can be power managed is another important spatial consideration. If multicore processors can be power managed only at the socket level, then there's good motivation to consolidate system load on as few sockets as possible. Consolidation drives up utilization across some resources, while quiescing others. This enables the quiesced resources to be power managed while "directing" power (and performance) to the utilized portion of the system.

Another factor that may play into individual resource selection and utilization distribution decisions are the characteristics of the workload(s) using the resources. This may dictate, for example, how aggressively a resource manager can consolidate utilization across the system without negatively impacting performance (as a result of resource contention) or to what extent changing a utilized resource's power state will impact the consumer's performance.

Temporal Considerations

Some resource managers may also allocate resources in time, as well as (or rather than) space. For example, a timer subsystem might allow clients to schedule some processing at some point (or with some interval) in the future, or a task queue subsystem might provide a means for asynchronous or deferred execution. The interfaces to such subsystems have traditionally been very narrow and prescriptive, leaving little room for temporal optimization. One solution is to provide interfaces to clients that are more descriptive in nature. For example, rather than providing a narrow interface for precise specification of what should happen and when:

```
int
schedule_timer((void)*
what(), time_t when);
```

a timer interface might instead specify what needs to be done along with a description of the constraints for when it needs to happen:

```
int
schedule_timer((void)*
what(), time_t about_when,
   time_t deferrable_by,
     time_t advancable_by);
```

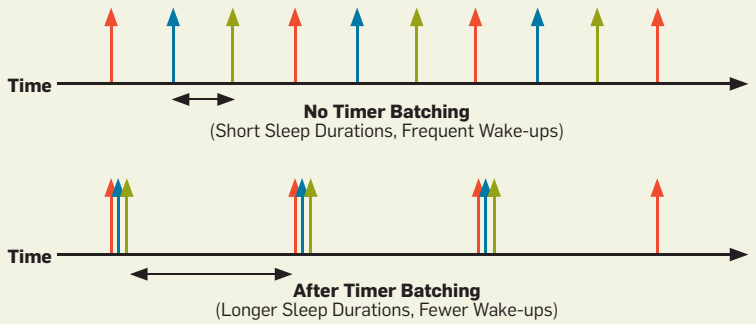
Analogous to consolidating load onto fewer sockets to improve spatial resource quiescence, providing some temporal latitude allows the timer subsystem to consolidate and batch process expirations. So rather than waking up a CPU *n* times over a given time interval to process *n* timers (incurring some overhead with each wakeup), the timer subsystem could wake the CPU once and batch process all the timers allowable per the (more relaxed) constraints, thus reducing CPU overhead time and increasing power-managed state residency (see Figure 2).

Efficient Resource Consumption

Clearly, resource managers can contribute much to the overall efficiency of the system, but ultimately they are forced to work within the constraints and requests put forth by the system's resource consumers. Where the constraints are excessive and resources are overallocated or not efficiently used, the benefits of even the most sophisticated power-management features can be for naught while the efficiency of the entire system stack is compromised.

Well-designed, efficient software is a thing of beauty showing good proportionality between utilization (and useful work done) and the amount of resources consumed. For utopian software, such proportionality would be perfect, demonstrating that when no work is done, zero resources are used; and as resource utilization scales high-

Figure 2. Benefits of batch processing periodic timers.



er, the amount of work done scales similarly (see Figure 3).

Real software is not utopian, though, and the only way to have software consume zero resources is not to run it at all. Even running very well-behaved software at the minimum will, in practice, require some resource overhead.

By contrast, inefficient software demonstrates poor proportionality between resource utilization and amount of work done. Here are some common examples:

- A process is waiting for something, such as the satisfying of a condition, and is using a timer to periodically wake up to check if the condition has been satisfied. No useful work is being done as it waits, but each time it wakes up to check, the CPU is forced to leave an idle power-managed state. What's worse, the process has decided to wake up with high frequency to "minimize latency" (see Figure 4).
- An application uses multiple threads to improve concurrency and scale throughput. It blindly creates as many threads as there are CPUs on the system, even though the application is unable to scale beyond a handful of threads because of an internal bottleneck. Having more threads means more CPUs must be awakened to run them, despite little to no marginal contribution to performance with each additional thread (see Figure 5).
- A service slowly leaks memory, and over time its heap grows to consume much of the system's physical memory, despite little to none of it actually being needed. As a consequence, little opportunity exists to power manage memory since most of it has been allocated.

Observing Inefficiency in the Software Ecosystem

Comprehensive analysis of software efficiency requires the ability to observe the proportionality of resource utilization versus useful work performed. Of course, the metric for "work done" is inherently workload specific. Some workloads (such as Web servers and databases) might be throughput based. For such workloads, one technique could be to plot throughput (for example, transactions per second) versus {cpu|memory|bandwidth|storage} resource consumption. Where a "knee" in the curve exists (resource utilization rises, yet throughput does not), there is an opportunity either to use fewer resources to do the same work or perhaps to eliminate a bottleneck to facilitate doing more work using the same resources.

For parallel computation workloads that use concurrency to speed up processing, one could plot elapsed computation time, versus the resources consumed, and using a similar technique identify and avoid the point of diminishing returns.

Rather than using workload-specific analysis, another fruitful technique is looking at systemwide resource utilization behavior at what should be zero utilization (system idle). By definition, the system isn't doing anything useful, so any software that is actively consuming CPU cycles is instantly suspect. PowerTOP is an open source utility developed

Figure 3. Good efficiency.

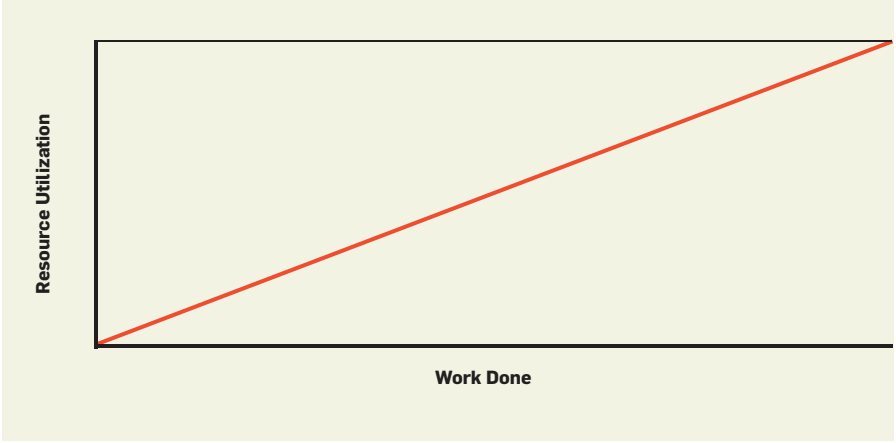


Figure 4. "Idle" inefficiency.

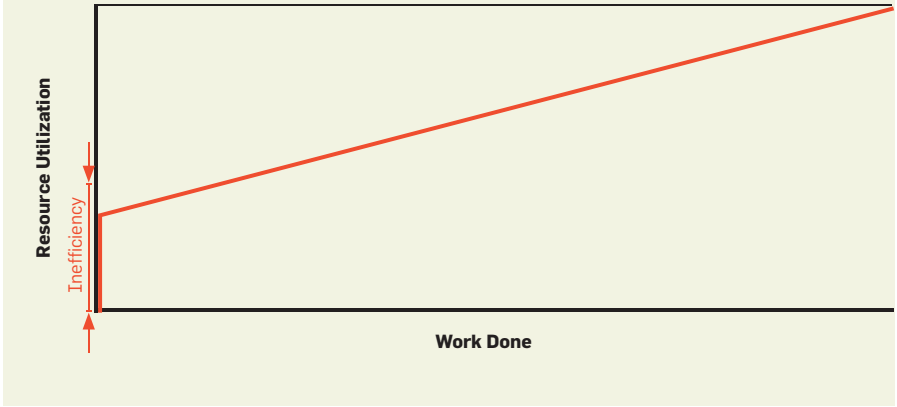
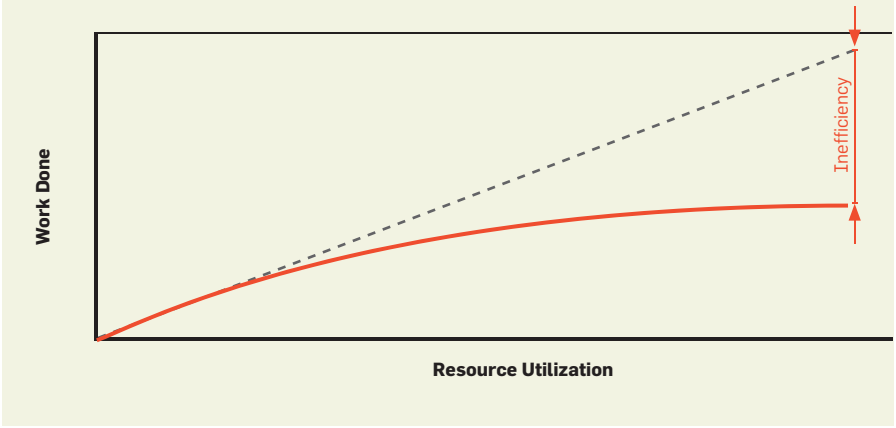


Figure 5. Scaling inefficiency.



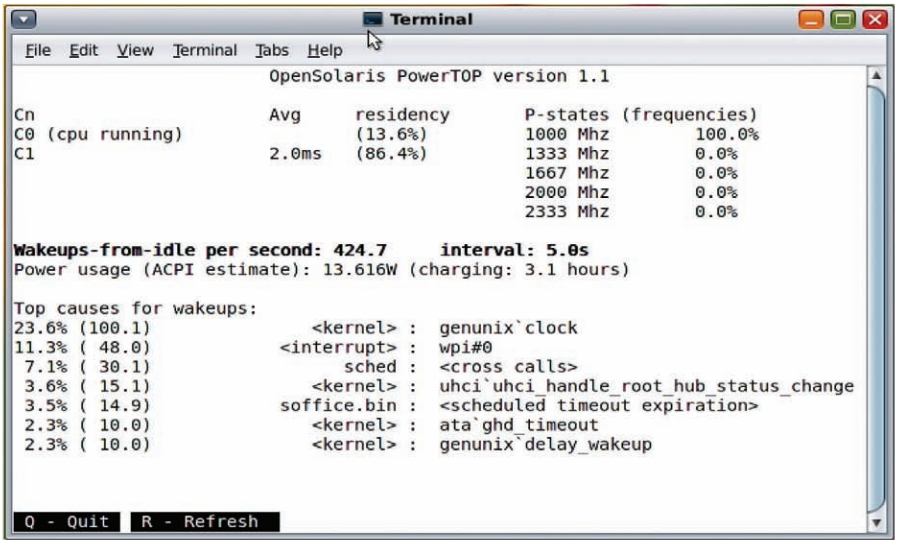


Figure 6. PowerTOP.

by Intel specifically to support this methodology of analysis (see Figure 6). Running the tool on what should be an otherwise idle system, one would expect ideally that the system's processors are power managed 100% of the time, but in practice, inefficient software (usually doing periodic time-based polling) will keep CPUs fractionally busy. PowerTOP shows the extent of the waste, while also showing which software is responsible. System users can then report the observed waste as bugs and/or elect to run more efficient software.

Designing Efficient Software

Efficiency as a design and optimization point for software might at first seem a bit foreign, so let's compare it with some others that are arguably more established: performance and scalability.

▸ Well-performing software maximizes the amount of useful work done (or minimizes the time taken to do it), given a fixed set of resources.

▸ Scalable software will demonstrate that performance proportionally increases as more resources are used.

Efficient software can be said to be both well performing and scalable, but with some additional constraints around resource utilization.

▸ Given a fixed level of performance (amount of useful work done or amount of time taken to do it), software uses the minimal set of resources required.

▸ As performance decreases, resource utilization proportionally decreases.

This implies that in addition to looking at the quantity and proportionality of performance *given the resource utilization*, to capture efficiency, software designers also need to consider the quantity and proportionality of resource utilization *given the performance*. If all this seems too abstract, here are some more concrete factors to keep in mind:

▸ When designing software that will be procuring its own resources, ensure it understands what resources are required to get the job done and yields them back when not needed to facilitate idle resource power management. If the procured resources will be needed on an intermittent basis, have the software try to leverage features that provide hints to the resource manager about when resources are (and are not) being used at least to facilitate active power management.

With respect to CPU utilization:

▸ When threads are waiting for some condition, try to leverage an event-triggered scheme to eliminate the need for time-based polling. Don't write "are we there yet?" software.

▸ If this isn't possible, try to poll infrequently.

▸ If it can't be eliminated, try at least to ensure that all periodic/polling activity is batch processed. Leverage timer subsystem features that provide latitude for optimization, such as coarsening resolution or allowing for timer advance/deferral.

With respect to memory utilization:

▸ Watch for memory leaks.

▸ Free or unmap memory that is no longer needed. Some operating systems provide advisory interfaces around memory utilization, such as madvise(3c) under Solaris.

With respect to I/O utilization

▸ If possible, buffer/batch I/O requests.

Driving Toward an Efficient System Stack

Every so often, evolution and innovation in hardware design brings about new opportunities and challenges for software. Features to reduce power consumption of underutilized system resources have become pervasive in even the largest systems, and the software layers responsible for managing those resources must evolve in turn—implementing policies that drive performance for utilized resources while reducing power for those that are underutilized.

Beyond the resource managers, resource consumers clearly have a significant opportunity either to contribute to or undermine the efficiency of the broader stack. Though getting programmers to think differently about the way they design software is more than a technical problem, tools such as PowerTOP represent a great first step by providing programmers and administrators with observability into software inefficiency, a point of reference for optimization, and awareness of the important role software plays in energy-efficient computing. C

Related articles on queue.acm.org

Powering Down
Matthew Garrett
http://queue.acm.org/detail.cfm?id=1331293

Maximizing Power Efficiency with Asymmetric Multicore Systems
Alexandra Fedorova, Juan Carlos Saez, Daniel Shelepov, and Manuel Prieto
http://queue.acm.org/detail.cfm?id=1658422

Modern System Power Management
Andrew Grover
http://queue.acm.org/detail.cfm?id=957774

Eric Saxe is a staff engineer in the Solaris Kernel Development Group at Sun Microsystems. Over the past 10 years at Sun, he has worked on a number of scheduler/dispatcher-related kernel components including the CMT (chip multithreading) scheduling subsystem, the Power Aware Dispatcher, and MPO (the Solaris NUMA framework), and he is the lead inventor for several related U.S. patents.

DOI:10.1145/1646353.1646371

Article development led by acmqueue
queue.acm.org

Contention for caches, memory controllers, and interconnects can be eased by contention-aware scheduling algorithms.

BY ALEXANDRA FEDOROVA, SERGEY BLAGODUROV, AND SERGEY ZHURAVLEV

Managing Contention for Shared Resources on Multicore Processors

MODERN MULTICORE SYSTEMS are designed to allow clusters of cores to share various hardware structures, such as LLCs (last-level caches; for example, L2 or L3), memory controllers, and interconnects, as well as prefetching hardware. We refer to these resource-sharing clusters as memory domains because the shared resources mostly have to do with the memory hierarchy. Figure 1 provides an illustration of a system with two memory domains and two cores per domain.

Threads running on cores in the same memory domain may compete for the shared resources, and this contention can significantly degrade their performance relative to what they could achieve running in a contention-free environment. Consider an example demonstrating how contention for shared resources can affect application performance. In this example, four applications—Soplex, Sphinx, Gamess, and Namd, from the Standard Performance Evaluation Corporation

(SPEC) CPU 2006 benchmark suite[6]—run simultaneously on an Intel Quad-Core Xeon system similar to the one depicted in Figure 1.

As a test, we ran this group of applications several times, in three different schedules, each time with two different pairings sharing a memory domain. The three pairing permutations afforded each application an opportunity to run with each of the other three applications within the same memory domain:

- Soplex and Sphinx ran in a memory domain, while Gamess and Namd shared another memory domain.
- Sphinx was paired with Gamess, while Soplex shared a domain with Namd.
- Sphinx was paired with Namd, while Soplex ran in the same domain with Gamess.

Figure 2 contrasts the best performance of each application with its worst performance. The performance levels are indicated in terms of the percentage of degradation from solo execution time (when the application ran alone on the system), meaning that the lower the numbers, the better the performance.

There is a dramatic difference between the best and the worst schedules, as shown in the figure. The workload as a whole performed 20% better with the best schedule, while gains for individual applications Soplex and Sphinx were as great as 50%. This indicates a clear incentive for assigning applications to cores according to the best possible schedule. While a contention-oblivious scheduler might accidentally happen upon the best schedule, it could just as well run the worst schedule. A contention-aware schedule, on the other hand, would be better positioned to choose a schedule that performs well.

This article describes an investigation of a thread scheduler that would mitigate resource contention on multicore processors. Although we began this investigation using an analytical modeling approach that would be difficult to implement online, we ultimately arrived at a scheduling method that can be easily implemented online with a modern operating system or even prototyped at the user level. To share a complete understanding of the problem, we describe both the offline and online modeling approaches. The article concludes with some actual performance data that shows the impact contention-aware scheduling techniques can have on the performance of applications running on currently available multicore systems.

To make this study tractable we made the assumption that the threads do not share any data (that is, they belong either to different applications or to the same application where each thread works on its own data set). If threads share data, they may actually

Figure 1. A schematic view of a multicore system with two memory domains representing the architecture of Intel Quad-Core Xeon processors.

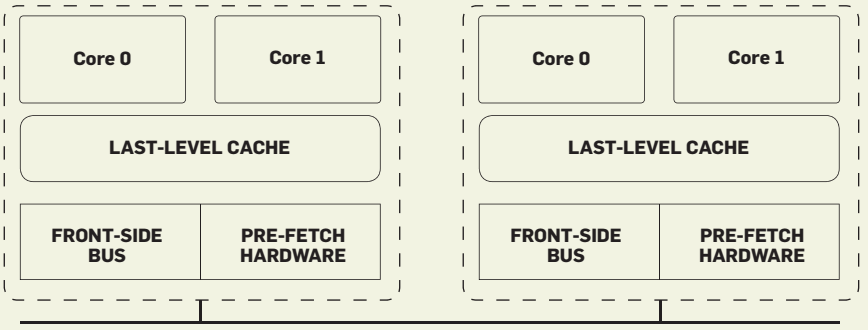


Figure 2. Percentage of performance degradation over a solo run achieved in two different scheduling assignments: the best and the worst. The lower the bar, the better the performance.

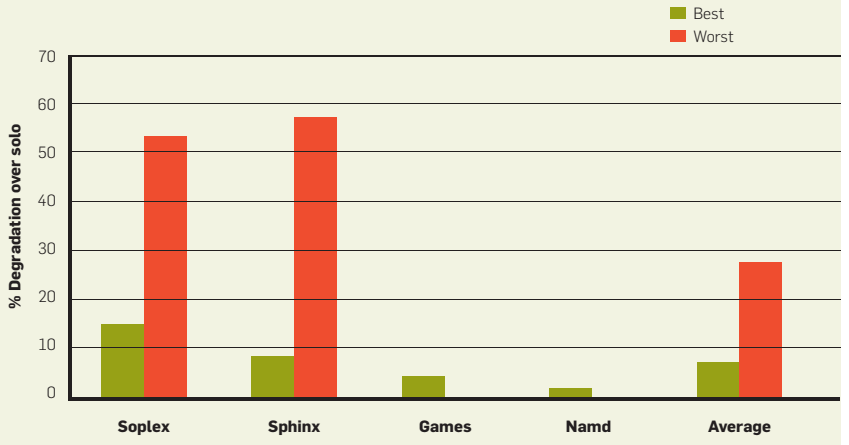
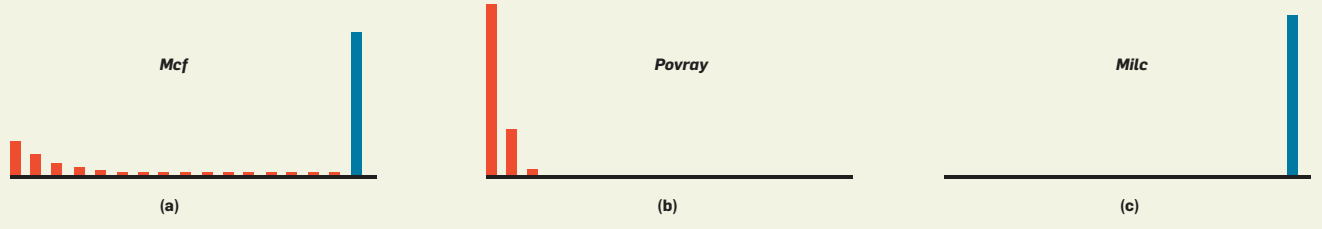


Figure 3. *Mcf* (a) is an application with a rather poor temporal locality, hence the low reuse frequency and high miss frequency. *Povray* (b) has excellent temporal locality. *Milc* (c) rarely reuses its data, therefore showing a very low reuse frequency and a very high miss frequency.



benefit from running in the same domain. In that case, they may access the shared resources cooperatively; for example, prefetch the data for each other into the cache. While the effects of cooperative sharing must be factored into a good thread-placement algorithm for multicores, this subject has been explored elsewhere.[9] The focus here is managing resource contention.

Understanding Resource Contention

To build a contention-aware scheduler, we must first understand how to model contention for shared resources. Modeling allows us to predict whether a particular group of threads is likely to compete for shared resources and to what extent. Most of the academic work in this area has focused on modeling contention for LLCs, as this was believed to have the greatest effect on performance. This is where we started our investigation as well.

Cache contention occurs when two or more threads are assigned to run on the cores of the same memory domain (for example, Core 0 and Core 1 in Figure 1). In this case, the threads share the LLC. A cache consists of cache lines that are allocated to hold the memory of threads as the threads issue cache requests. When a thread requests a line that is not in the cache—that is, when it issues a cache miss—a new cache line must be allocated. The issue here is that when a cache line must be allocated but the cache is full (which is to say whenever all the other cache lines are being used to hold other data), some data must be evicted to free up a line for the new piece of data. The evicted line might belong to a different thread from the one that issued the cache miss (modern CPUs do not assure any fairness in that regard), so an aggressive thread might end up evicting data for some other thread and thus hurting its performance.

Although several researchers have proposed hardware mechanisms for mitigating LLC contention,[3,7] to the best of our knowledge these have not been implemented in any currently available systems. We therefore looked for a way to address contention in the systems that people are running now and ended up turning our attention to scheduling as a result. Before building a scheduler that avoids cache contention, however, we needed to find ways to predict contention.

There are two schools of thought regarding the modeling of cache contention. The first suggests that considering the LLC miss rate of the threads is a good way to predict whether these threads are likely to compete for the cache (the miss rate under contention is an equally good heuristic as the solo miss rate). A miss rate is the number of times per instruction when a thread fails to find an item in the LLC and so must fetch it from memory. The reasoning is that if a thread issues lots of cache misses, it must have a large cache working set, since each miss results in the allocation of a new cache line. This way of thinking also maintains that any thread that has a large cache working set must suffer from contention (since it values the space in the cache) while inflicting contention on others.

This proposal is contradicted by followers of the second school of thought, who reason that if a thread hardly ever reuses its cached data—as would be the case with a video-streaming application that touches the data only once—it will not suffer from contention even if it brings lots of data into the cache. That is because such a thread needs very little space to keep in cache the data that it actively uses. This school of thought advocates that to model cache contention one must consider the memory-reuse pattern of the thread. Followers of this approach therefore created several models for shared cache contention based on memory-reuse patterns, and they have demonstrated that this approach predicts the extent of contention quite accurately.[3] On the other hand, only limited experimental data exists to support the plausibility of the other approach to modeling cache contention.[5] Given the stronger evidence in favor of the memory-reuse approach, we built our prediction model based on that method.

A memory-reuse pattern is captured by a memory-reuse profile, also known as the stack-distance[2] or reuse-distance profile.[1] Figure 3 shows examples of memory-reuse profiles belonging to applications in the SPEC CPU2006 suite. The red bars on the left show the reuse frequency (how often the data is reused), and the blue bar on the right indicates the miss frequency (how often that application misses in the cache). The reuse-frequency histogram shows the locality of reused data. Each bar represents a range of reuse distances—these can be thought of as the number of time steps that have passed since the reused data was last touched. An application with a very good temporal locality will have many high bars to the left of the histogram (Figure 3b), as it would reuse its data almost immediately. This particular application also has negligible miss frequency. An application with a poor temporal locality will have a flatter histogram and a rather high miss frequency (Figure 3a). Finally, an application that hardly ever reuses its data will result in a histogram indicating negligible reuse frequency and a very high miss frequency (Figure 3c).

Memory-reuse profiles have been used in the past to effectively model the contention between threads that share a cache.[3] These models, the details of which we omit from this article, are based on the shape of memory-reuse profiles. One such model, the SDC (stack distance competition) examines the reuse frequency of threads sharing the cache to determine which of the threads is likely to "win" more cache space; the winning thread is usually the one with the highest overall reuse frequency. Still, the SDC model, along with all the other models based on memory-reuse profiles, was deemed too complex for our purposes. After all, our goal was to use a model in an operating-system scheduler, meaning it needed to be both efficient and lightweight. Furthermore, we were interested in finding methods for approximating the sort of information memory-reuse profiles typically afford using just the data that's available at runtime, since memory-reuse profiles themselves are very difficult to obtain at runtime. Methods for obtaining these profiles online require unconventional hardware[7] or rely on hardware performance counters available only on select systems.[8]

Our goal, therefore, was to capture the essence of memory-reuse profiles in a simple metric and then find a way to approximate this metric using data that a thread scheduler can easily obtain online. To this end, we discovered that memory-reuse profiles are highly

successful at modeling contention largely because they manage to capture two important qualities related to contention: sensitivity and intensity. Sensitivity measures how much a thread suffers whenever it shares the cache with other threads. Intensity, on the other hand, measures how much a thread hurts other threads whenever it shares a cache with them. Measuring sensitivity and intensity appealed to us because together they capture the key information contained within memory-reuse profiles; we also had some ideas about how they could be approximated using online performance data. Before learning how to approximate sensitivity and intensity, however, we needed to confirm that these were indeed good bases for modeling cache contention among threads. To accomplish that, we formally derived the sensitivity and intensity metrics based on data in the memory-reuse profiles. After confirming that the metrics derived in this way did indeed accurately model contention, we could then attempt to approximate them using just online data.

Accordingly, we derived sensitivity S and intensity Z for an application using data from its memory-reuse profile. To compute S, we applied an aggregation function to the reuse-frequency histogram. Intuitively, the higher the reuse frequency, the greater an application is likely to suffer from the loss of cache space due to contention with another application—signifying a higher sensitivity. To compute Z, we simply used the cache-access rate, which can be inferred from the memory-reuse profile. Intuitively, the higher the access rate, the higher the degree of competition from the thread in question since the high access rate shows that it allocates new cache lines while retaining old ones. Details for the derivation of S and Z are described in another article.[10]

Using the metrics S and Z, we then created another metric called *Pain*, where the *Pain* of thread A due to sharing a cache with thread B is the product of the sensitivity of A and the intensity of B, and vice versa. A combined *Pain* for the thread pair is the sum of the *Pain* of A due to B and the *Pain* of B due to A, as shown here:

$$Pain(A|B) = S_A * Z_B$$
$$Pain(B|A) = S_B * Z_A$$
$$Pain(A,B) - Pain(A|B) + Pain(B|A)$$

Intuitively, $Pain(A|B)$ approximates the performance degradation of A when A runs with B relative to running solo. It will not capture the absolute degradation entirely accurately, but it is good for approximating relative degradations. For example, given two potential neighbors for A, the *Pain* metric can predict, which will cause a higher performance degradation for A. This is precisely the information a contention-aware scheduler would require.

The *Pain* metric shown here assumes that only two threads share a cache. There is evidence, however, showing this metric applies equally well when more than two threads share the cache. In that case, in order to compute the *Pain* for a particular thread as a consequence of running with all of its neighbors concurrently, the *Pain* owing to each neighbor must be averaged.

After developing the *Pain* metric based on memory-reuse profiles, we looked for a way to approximate it using just the data available online via standard hardware performance counters. This led us to explore two performance metrics to approximate sensitivity and intensity: the cache-miss rate and the cache-access rate. Intuitively these metrics correlate with the reuse frequency and the intensity of the application. Our findings regarding which metric offers the best approximation are surprising, so to maintain some suspense, we postpone their revelation until the section entitled "Evaluation of Modeling Techniques."

Figure 4. Computing the pain for all possible schedules. The schedule with the lowest *Pain* is chosen as the estimated best schedule.

Schedule [(A,B), (C,D)]: Pain = Average(Pain(A,B), Pain(C, D))
Schedule [(A,C), (B,D)]: Pain = Average(Pain(A,C), Pain(B, D))
Schedule [(A,D), (B,C)]: Pain = Average(Pain(A,D), Pain(B, C))

Figure 5. The metric for comparing the actual best and the estimated best schedule.

Estimated Best Schedule [(A,B), (C,D)]:
DegradationEst = Average(Degrad(A|B), Degrad(B|A), Degrad(C|D), Degrad(D|C))
Actual Best Schedule [(A,C), (B,D)]:
DegradationAct = Average(Degrad(A|C), Degrad(C|A), Degrad(B|D), Degrad(D|B))

$$\textit{Degradation over actual best} = \left(\frac{DegradationEst}{DegradationAct} - 1\right) \times 100\%$$

Using Contention Models in a Scheduler

In evaluating the new models for cache contention, our goal was to determine how effective the models would be for constructing contention-free thread schedules. We wanted the model to help us find the best schedule and avoid the worst one (recall Figure 2). Therefore, we evaluated the models on the merit of the schedules they managed to construct. With that in mind, we describe here how the scheduler uses the *Pain* metric to find the best schedule.

To simplify the explanation for this evaluation, we have a system with two pairs of cores sharing the two caches (as illustrated in Figure 1), but as mentioned earlier, the model also works well with more cores per cache. In this case, however, we want to find the best schedule for four threads. The scheduler would construct all the possible permutations of threads on this system, with each of the permutations being unique in terms of how the threads are paired on each memory domain. If we have four threads—A, B, C, and D—there will be three unique schedules: (1) {(A,B), (C,D)}; (2) {(A,C), (B,D)}; and (3) {(A,D), (B,C)}. Notation (A,B) means that threads A and B are co-scheduled in the same memory domain. For each schedule, the scheduler estimates the *Pain* for each pair: in schedule {(A,B), (C,D)} the scheduler would estimate Pain(A,B) and Pain(C,D) using the equations presented previously. Then it averages the *Pain* values of the pairs to estimate the *Pain* for the schedule as a whole. The schedule with the lowest *Pain* is deemed to be the estimated

best schedule. Figure 4 is a summary of this procedure.

The estimated best schedule can be obtained either by using the *Pain* metric constructed via actual memory-reuse profiles or by approximating the *Pain* metric using online data.

Once the best schedule has been estimated, we must compare the performance of the workload in the estimated best schedule with the performance achieved in the *actual* best schedule. The most direct way of doing this is to run the estimated best schedule on real hardware and compare its performance with that of the actual best schedule, which can be obtained by running all schedules on real hardware and then choosing the best one. Although this is the most direct approach (which we used for some experiments in the study), it limited the number of workloads we could test because running all possible schedules for a large number of workloads is time consuming.

To evaluate a large number of workloads in a short amount of time, we invented a semi-analytical evaluation methodology that relies partially on data obtained from tests on a real system and otherwise applies analytical techniques. Using this approach, we selected 10 benchmark applications from the SPEC CPU2006 suite to use in the evaluation. They were chosen using the minimum spanning-tree-clustering method to ensure the applications represented a variety of memory-access patterns.

We then ran all possible pairings of these applications on the experimental platform, a Quad-Core Intel Xeon system, where each Quad-Core processor looked like the system depicted in Figure 1. In addition to running each possible pair of benchmark applications on the same memory domain, we ran each benchmark alone on the system. This gave us the measure for the *actual degradation in performance* for each benchmark as a consequence of sharing a cache with another benchmark, as opposed to running solo. Recall that degradation relative to performance in the solo mode is precisely the quantity approximated by the *Pain* metric. So by comparing the scheduling assignment constructed based on the actual degradation to that constructed based on the *Pain* metric, we can evaluate how good the *Pain* metric is in finding good scheduling assignments.

In evaluating the new models for cache contention, our goal was to determine how effective the models would be for constructing contention-free thread schedules.

Having the actual degradations enabled us to construct the *actual* best schedule using the method shown in Figure 5. The only difference was that, instead of using the model to compute *Pain*(*A*,*B*), we used the actual performance degradation that we had measured on a real system (with *Pain* being equal to the sum of degradation of A running with B relative to running solo and the degradation of B running with A relative to running solo).

Once we knew the actual best schedule, we needed a way to compare it with the estimated best schedule. The performance metric was the average degradation relative to solo execution for all benchmarks. For example, suppose the estimated best schedule was {(A,B), (C,D)}, while the actual best schedule was {(A,C), (B,D)}. We computed the average degradation for each schedule to find the difference between the degradation in the estimated best versus the degradation in the actual one, as indicated in Figure 5. The notation *Degrad*(A|B) refers to the measured performance degradation of A when running alongside B, relative to A running solo.

This illustrates how to construct estimated best and actual best schedules for any four-application workload on a system with two memory domains so long as actual pair-wise degradations for any pair of applications have been obtained on the experimental system. Using the same methodology, we can evaluate this same model on systems with a larger number of memory domains. In that case, the number of possible schedules grows, but everything else in the methodology remains the same. In using this methodology, we assumed that the degradation for an application pair (A,B) would be the same whether it were obtained on a system where only A and B were running or on a system with other threads, such as (C,D) running alongside (A,B) on another domain. This is not an entirely accurate assumption since, with additional applications running, there will be a higher contention for the front-side bus. Although there will be some error in estimating schedule-average degradations under this method, the error is not great enough to affect

Figure 6. The percentage by which performance of schedules estimated to be best according to various modeling techniques varies from the actual best schedules. Low bars are good.

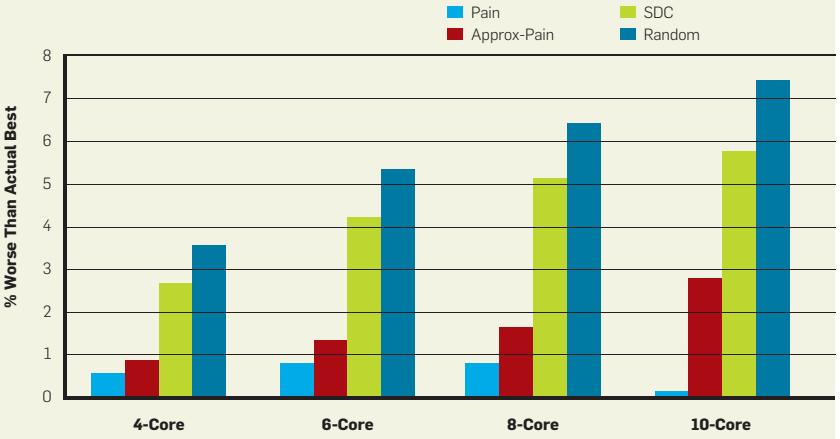


Figure 7. A breakdown of factors causing performance degradation due to contention for shared hardware on multicore systems based on tests using select applications in the SPEC CPU2006 suite.

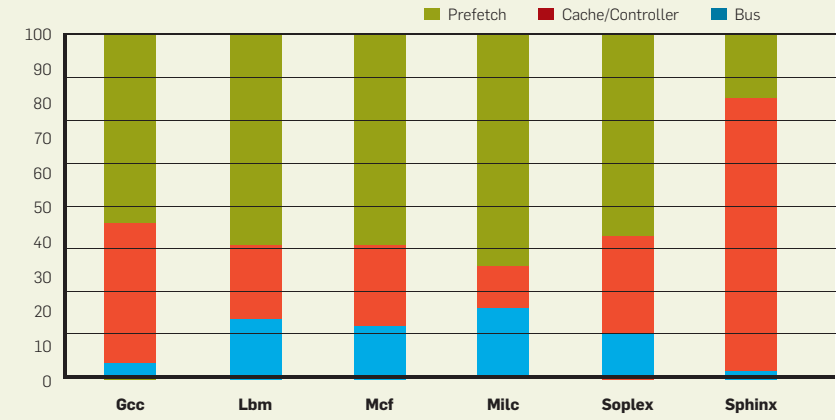
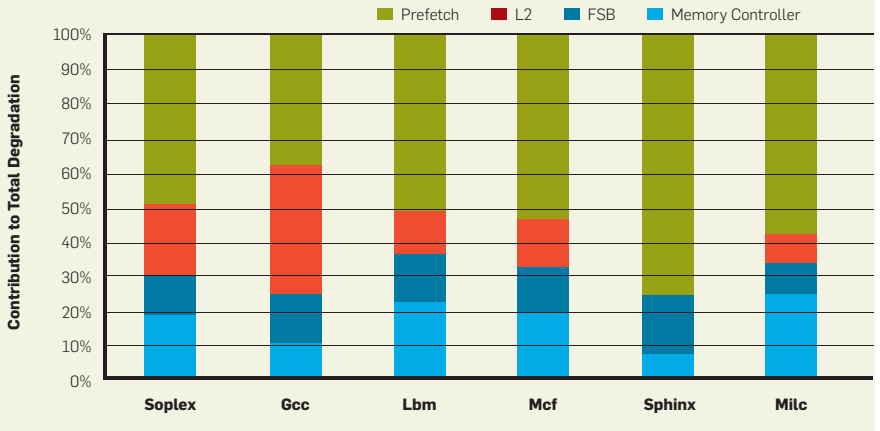


Figure 8. A breakdown of factors causing performance degradation due to contention for shared hardware on multicore systems based on tests using select applications in the SPEC CPU2006 suite. These experiments were performed on an Intel Xeon (Cloverton) processor. We also obtained data showing that cache contention is not dominant on AMD Opteron systems.



the evaluation results significantly. Certainly, the error is not large enough to lead to a choice of the "wrong" best schedule.

Evaluation of Modeling Techniques

Here, we present the results obtained using our semi-analytical methodology, followed by the performance results obtained via experiments only. Figure 6 compares the degradation over the actual best schedules (the method for which was indicated earlier) with estimated best schedules constructed using various methods. The blue bar indicating *Pain* is the model that uses memory-reuse profiles to estimate *Pain* and find the best schedule (the method for which was set forth in Figure 4). In the red bar indicating Approx-Pain, the *Pain* for a given application running with another is estimated with the aid of data obtained online (we explain which data this is at the end of this section); once Pain has been estimated, we can once again use the method shown in Figure 4. In SDC, a previously proposed model based on memory-reuse profiles[3] can be used to estimate the performance degradation of an application when it shares a cache with a co-runner. This estimated degradation can then be used in place of *Pain*(A|B); apart from that, the method shown in Figure 4 applies. Although SDC is rather complex for use in a scheduler, we compared it with our new models to evaluate how much performance was being sacrificed by using a simpler model. Finally, in Figure 6 the bar labeled Random shows the results for selecting a random-thread placement.

Figure 6 shows how much worse the schedule chosen with each method ended up performing relative to the actual best schedule. This value was computed using the method shown in Figure 4. Ideally, this difference from the actual best ought to be small, so in considering Figure 6, remember that low bars are good. The results for four different systems—with four, six, eight, and 10 cores—are indicated. In all cases there were two cores per memory domain. (Actual results from a system with a larger number of cores per memory domain are shown later.) Each bar represents the average for all the benchmark pairings that could be constructed out of our 10 representa-

tive benchmarks on a system with a given number of cores, such that there is exactly one benchmark per core. On four- and six-core systems, there were 210 such combinations, whereas an eight-core system had 45 combinations, and a 10-core system had only one combination. For each combination, we predicted the best schedule. The average performance degradation from the actual best for each of these estimated best schedules is reported in each bar in the figure.

The first thing we learned from the metric in Figure 5 was that the *Pain* model is effective for helping the scheduler find the best thread assignment. It produces results that are within 1% of the actual best schedule. (The effect on the actual execution time is explored later). We also found that choosing a random schedule produces significantly worse performance, especially as the number of cores grows. This is significant in that a growing number of cores is the expected trend for future multicore systems.

Figure 6 also indicates that the *Pain* approximated by way of an online metric works very well, coming within just 3% of the actual best schedule. At the same time, the SDC, a well-proven model from an earlier study, turns out to be less accurate. These results—both the effectiveness of the approximated *Pain* metric and the disappointing performance of the older SDC model—were quite unexpected. Who could have imagined that the best way to approximate the *Pain* metric would be to use the LLC miss rate? In other words, the LLC miss rate of a thread is the best predictor of both how much the thread will suffer from contention (its sensitivity) and how much it will hurt others (its intensity). As explained at the beginning of this article, while there was limited evidence indicating that the miss rate predicts contention, it ran counter to the memory-reuse-based approach, which was supported by a much larger body of evidence.

Our investigation of this paradox led us to examine the causes of contention on multicore systems. We performed several experiments that aimed to isolate and quantify the degree of contention for various types of shared resources: cache, memory controller, bus, prefetching hardware. The precise setup of these experiments is described in another study.[10] We arrived at the following findings.

First, it turns out that contention for the shared cache—the phenomenon by which competing threads end up evicting each others' cache lines from the cache—is not the main cause of performance degradation experienced by competing applications on multicore systems. Contention for other shared resources, such as the front-side bus, prefetching resources, and the memory controller are the dominant causes for performance degradation (see Figure 7). That is why the older memory-reuse model, designed to model cache contention only, was not effective in our experimental environment. The authors of that model evaluated it on a simulator that did not model contention for resources other than shared cache, and it turns out that, when applied to a real system where other types of contention were present, the model did not prove effective.

On the other hand, cache-miss rate turned out to be an excellent predictor for contention for the memory controller, prefetching hardware, and front-side bus. Each application in our model was co-scheduled with the Milc application to generate contention. Given limitations of existing hardware counters, it was difficult to separate the effects of contention for prefetching hardware itself and the effects of additional contention for memory controller and front-side bus caused by prefetching. Therefore, the impact of prefetching shows the combined effect of these two factors.

An application issuing many cache misses will occupy the memory controller and the front-side bus, so it will not only hurt other applications that use that hardware, but also end up suffering itself if this hardware is usurped by others. An application aggressively using prefetching hardware will also typically have a high LLC miss rate, because prefetch requests for data that is not in the cache are counted as cache misses. Therefore, a high miss rate is also an indicator of the heavy use of prefetching hardware.

In summary, our investigation of contention-aware scheduling algorithms has taught us that high-miss-rate applications must be kept apart. That is, they should not be co-scheduled in the same memory domain. Although some researchers have already suggested this approach, it is not well understood why using the miss rate as a proxy for contention ought to be effective, particularly in that it contradicts the theory behind the popular memory-reuse model. Our findings should help put an end to this controversy.

Based on this new knowledge, we have built a prototype of a contention-aware scheduler that measures the miss rates of online threads and decides how to place threads on cores based on that information. Here, we present some experimental data showing the potential impact of this contention-aware scheduler.

Implications

Based on our understanding of contention on multicore processors, we have built a prototype of a contention-aware scheduler for multicore systems called Distributed Intensity Online (DIO). The DIO scheduler distributes intensive applications across memory domains (and by intensive we mean those with high LLC miss rates) after measuring online the applications' miss rates. Another prototype scheduler, called Power Distributed Intensity (Power DI), is intended for scheduling applications in the workload across multiple machines in a data center. One of its goals is to save power by determining how to employ as few systems as possible without hurting performance. The following are performance results of these two schedulers.

Distributed Intensity Online. Different workloads offer different opportunities to achieve performance improvements through the use of a contention-aware scheduling policy. For example, a workload consisting of non-memory-intensive applications (those with low cache miss rates) will not experience any performance improvement since there is no contention to alleviate in the first place. Therefore, for our experiments we constructed eight-application workloads containing from two to six memory-intensive applications. We picked eight workloads in total, all consisting of SPEC CPU2006 applications, and then executed them under the DIO and the default Linux sched-

uler on an AMD Opteron system featuring eight cores—four per memory domain. The results are shown in Figure 8. The performance improvement relative to default has been computed as the average improvement for all applications in the workload (since not all applications are memory intensive, some do not improve). We can see that DIO renders workload-average performance improvements of up to 11%.

Another potential use of DIO is as a way to ensure QoS (quality of service) for critical applications since DIO essentially provides a means to make sure the worst scheduling assignment is never selected, while the default scheduler may occasionally suffer as a consequence of a bad thread placement. Figure 9 shows for each of the applications as part of the eight test workloads its worst-case performance under DIO relative to its worst-case performance under the default Linux scheduler. The numbers are shown in terms of the percentage of improvement or the worst-case behavior achieved under DIO relative to that encountered with the default Linux scheduler, so higher bars in this case are better. We can see that some applications are as much as 60% to 80% better off with their worst-case DIO execution times, and in no case did DIO do significantly worse than the default scheduler.

Figure 9. Performance of eight workloads under DIO relative to the default Linux scheduler.

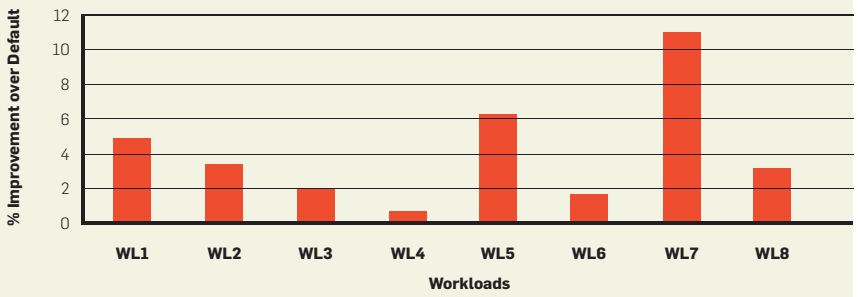


Figure 10. Worst-case performance for each of the applications included as part of the eight test workloads.

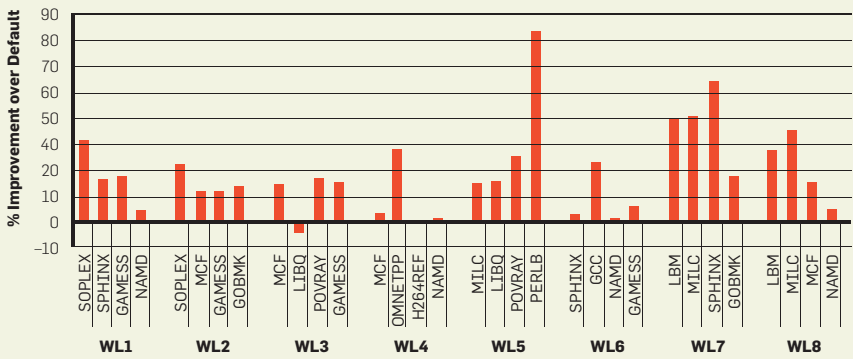
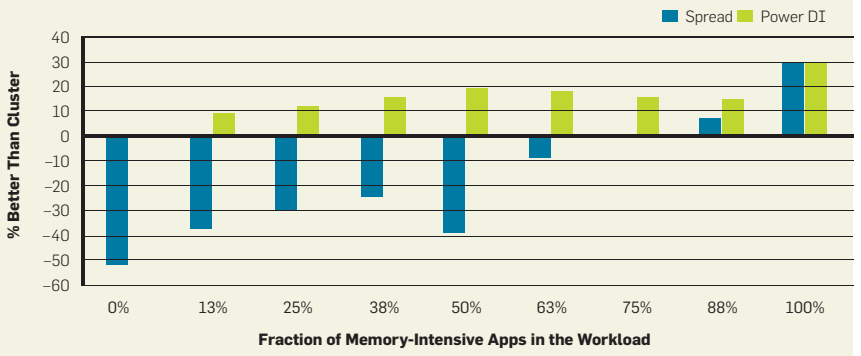


Figure 11. Percentage reduction in EDP.



Power Distributed Intensity. One of the most effective ways to conserve CPU power consumption is to turn off unused cores or entire memory domains in an active system. Similarly, if the workload is running on multiple machines—for example, in a data center—power savings can be accomplished by clustering the workload on as few servers as possible while powering down the rest. This seemingly simple solution is a double-edged sword, however, because clustering the applications on just a few systems may cause them to compete for shared system resources and thus suffer performance loss. As a result, more time will be needed to complete the workload, meaning that more energy will be consumed. In an attempt to save power it is also necessary to consider the impact that clustering can have on performance. A metric that takes into account both the energy consumption and the performance of the workload is the energy-delay product (EDP).[4] Based on our findings about contention-aware scheduling, we designed Power DI, a scheduling algorithm meant to save power without hurting performance.

Power DI works as follows: Assuming a centralized scheduler has knowledge of the entire computing infrastructure and distributes incoming applications across all systems, Power DI clusters all incoming applications on as few machines as possible, except for those applications deemed to be memory intensive. Similarly, within a single machine, Power DI clusters applications on as few memory domains as possible, with the exception of memory-intensive applications. These applications are not co-scheduled on the same memory domain with another application unless the other application has a very low cache miss rate (and thus a low memory intensity). To determine if an application is memory-intensive,

Power DI uses an experimentally derived threshold of 1,000 misses per million instructions; an application whose LLC miss rate exceeds that amount is considered memory intensive.

Although we did not have a data-center setup available to us to evaluate this algorithm, we simulated a multi-server environment in the following way. The in-house AKULA scheduling simulator created a schedule for a given workload on a specified data-center setup, which in this case consisted of 16 eight-core systems, assumed by the simulator to be Intel Xeon dual quad-core servers. Once the simulated scheduler decided how to assign applications across machines and memory domains within each machine, we computed the performance of the entire workload from the performance of the applications assigned by the scheduler to each eight-core machine. The performance on a single system could be easily measured via experimentation. This simulation method was appropriate for our environment, since there was no network communication among the running applications, meaning that inferring the overall performance from the performance of individual system was reasonable.

To estimate the power consumption, we used a rather simplistic model (measurements with the actual power meter are still under way) but captured the right relationships between power consumed in various load conditions. We assumed that a memory domain where all the cores are running applications consumes one unit of power. A memory domain where one out of two cores are busy consumes 0.75 units of power. A memory domain where all cores are idle is assumed to be in a very low power state and thus consumes 0 units of power. We did not model the latency of power-state transitions.

We constructed a workload of 64 SPEC CPU2006 applications randomly drawn from the benchmark suite. We varied the fraction of memory-intensive applications in the workload from zero to 100%. The effectiveness of scheduling strategies differed according to the number of memory-intensive applications. For example, if there were no memory-intensive applications, it was perfectly fine to cluster all the applications to the greatest extent possible. Conversely, if all the applications were memory intensive, then the best policy was to spread them across memory domains so that no two applications would end up running on the same memory domain. An intelligent scheduling policy must be able to decide to what extent clustering must be performed given the workload at hand.

Figure 10 shows the EDP for three different scheduling methods: Power DI, a naïve Spread method (which always spreads applications across machines to the largest extent possible), and the Cluster method (which in an attempt to save power always clusters applications on as few machines and as few memory domains as possible). The numbers are shown as a percentage reduction in the EDP (higher is better) of Power DI and Spread over Cluster.

We can see that when the fraction of memory-intensive applications in the workload is low, the naïve Spread method does much worse than the Cluster method, but it beats Cluster as that fraction increases. Power DI, on the other hand, is able to adjust to the properties of the workload and minimize EDP in all cases, beating both Spread and Cluster—or at least matching them—for every single workload.

Conclusion

Contention for shared resources significantly impedes the efficient operation of multicore systems. Our research has provided new methods for mitigating contention via scheduling algorithms. Although it was previously thought that the most significant reason for contention-induced performance degradation had to do with shared cache contention, we found that other sources of contention—such as shared prefetching hardware and memory interconnects—are just as important. Our heuristic—the LLC miss rate—proves to be an excellent predictor for all types of contention. Scheduling algorithms that use this heuristic to avoid contention have the potential to reduce the overall completion time for workloads, avoid poor performance for high-priority applications, and save power without sacrificing performance. C

Related articles on queue.acm.org

Maximizing Power Efficiency with Asymmetric Multicore Systems
Alexandra Fedorova, Juan Carlos Saez, Daniel Shelepov, and Manuel Prieto
http://queue.acm.org/detail.cfm?id=1658422

The Future of Microprocessors
Kunle Olukotun
http://queue.acm.org/detail.cfm?id=1095418

References

1. Berg, E. and Hagersten, E. StatCache: A probabilistic approach to efficient and accurate data locality analysis. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software* (2004), 20–27.
2. Cascaval, C., DeRose, L., Padua, D.A. and Reed, D. 1999. Compile-time based performance prediction. In *Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing* (1999), 365–379.
3. Chandra, D., Guo, F., Kim, S. and Solihin, Y. Predicting inter-thread cache contention on a multiprocessor architecture. In *Proceedings of the 11th International Symposium on High-performance Computer Architecture* (2005), 340–351.
4. Gonzalez, R. and Horowitz, M. Energy dissipation in general-purpose microprocessors. *IEEE Journal of Solid State Circuits 31*, 9 (1999), 1277–1284.
5. Knauerhase, R., Brett, P., Hohlt, B., Li, T. and Hahn, S. Using OS observations to improve performance in multicore systems. *IEEE Micro* (2008), 54–66.
6. SPEC: Standard Performance Evaluation Corporation; http://www.spec.org.
7. Suh, G., Devadas, S. and Rudolph, L. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the 8th International Symposium on High-performance Computer Architecture* (2002), 117.
8. Tam, D., Azimi, R., Soares, L. and Stumm, M. RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems* (2009), 121–132.
9. Tam, D., Azimi, R. and Stumm, M. Thread clustering: sharing-aware scheduling on SMP-CMP-SMT multiprocessors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems* (2007), 47–58.
10. Zhuravlev, S., Blagodurov, S. and Fedorova, A. Addressing shared resource contention in multicore processors via scheduling. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems* (2010).

Alexandra Fedorova is an assistant professor of computer science at Simon Fraser University in Vancouver, Canada, where she co-founded the SYNAR (Systems, Networking and Architecture) research lab. Her research interests span operating systems and virtualization platforms for multicore processors, with a specific focus on scheduling. Recently she started a project on tools and techniques for parallelization of video games, which has led to the design of a new language for this domain.

Sergey Blagodurov is a Ph.D. student in computer science at Simon Fraser University, Vancouver, Canada. His research focuses on operating-system scheduling on multicore processors and exploring new techniques to deliver better performance on non-uniform memory access (NUMA) multicore systems.

Sergey Zhuravlev is a Ph.D. student in computer science at Simon Fraser University, Vancouver, Canada. His recent research focuses on scheduling on multiprocessor systems to avoid shared resource contention as well as simulating computing systems.

DOI:10.1145/1646353.1646372

A translator framework enables the use of model checking in complex avionics systems and other industrial settings.

BY STEVEN P. MILLER, MICHAEL W. WHALEN, AND DARREN D. COFER

Software Model Checking Takes Off

ALTHOUGH FORMAL METHODS have been used in the development of safety- and security-critical systems for years, they have not achieved widespread industrial use in software or systems engineering. However, two important trends are making the industrial use of formal methods practical. The first is the growing acceptance of model-based development for the design of embedded systems. Tools such as MATLAB Simulink[6] and Esterel Technologies SCADE Suite[2] are achieving widespread use in the design of avionics and automotive systems. The graphical models produced by these tools provide a formal, or nearly formal, specification that is often amenable to formal analysis.

The second is the growing power of formal verification tools, particularly model checkers. For many classes of models they provide a "push-button" means of determining if a model meets its requirements. Since these tools examine all possible combinations of inputs and state, they are much more likely to find design errors than testing.

Here, we describe a translator framework developed by Rockwell Collins and the University of Minnesota that allows us to automatically translate from some of the most popular commercial modeling languages to a variety of model checkers and theorem provers. We describe three case studies in which these tools were used on industrial systems that demonstrate that formal verification can be used effectively on real systems when properly supported by automated tools.

PHOTOGRAPH: "FLUGHAFEN" BY HO-YEOL RYU

Model-Based Development

Model-based development (MBD) refers to the use of domain-specific, graphical modeling languages that can be executed and analyzed before the actual system is built. The use of such modeling languages allows the developers to create a model of the system, execute it on their desktops, analyze it with automated tools, and use it to automatically generate code and test cases.

Throughout this article we use MBD to refer specifically to software developed using synchronous dataflow languages such as those found in MATLAB Simulink and Esterel Technologies SCADE Suite. Synchronous modeling languages latch their inputs at the start of a computation step, compute the next system state and its outputs as a single atomic step, and communicate between components using dataflow signals. This differs from the more general class of modeling languages that include support for asynchronous execution of components and communication using message passing. MBD has become very popular in the avionics and automotive industries and we have found synchronous dataflow models to be especially well suited for automated verification using model checking.

Model checkers are formal verification tools that evaluate a model to determine if it satisfies a given set of properties.[1] A model checker will consider every possible combination of inputs and state, making the verification equivalent to exhaustive testing of the model. If a property is not true, the model checker produces a counterexample showing how the property can be falsified.

There are many types of model checkers, each with their own strengths and weaknesses. Explicit state model checkers such as SPIN[4] construct and store a representation of each state visited. Implicit state (symbolic) model checkers use logical representations of sets of states (such as Binary Decision Diagrams) to describe regions of the model state space that satisfy the properties being evaluated. Such compact representations generally allow symbolic model checkers to handle a much larger state space than explicit state model checkers. We have

used the BDD-based model checker NuSMV[5] to analyze models with over 10^{120} reachable states.

More recent model checkers, such as SAL[11] and Prover Plug-In,[9] use *satisfiability modulo theories* (SMT) solvers for reasoning about infinite state models containing real numbers and unbounded arrays. These checkers use a form of induction over the state transition relation to automatically prove that a property holds over all executable paths in a model. While these tools can handle a larger class of models, the properties to be checked must be written to support inductive proof.

The Translator Framework

As part of NASA's Aviation Safety Program (AvSP), Rockwell Collins and the University of Minnesota developed a product family of translators that bridge the gaps between some of the most popular commercial modeling languages and several model checkers and theorem provers.[8] An overview of this framework is shown in Figure 1.

These translators work primarily with the Lustre formal specification language,[3] but this is hidden from the users. The starting point for translation is a design model in MATLAB Simulink/Stateflow or Esterel Technologies SCADE Suite/Safe State Machines. SCADE Suite produces Lustre models directly. Simulink or Stateflow models can be imported using SCADE Suite or the Reactis[10] tool and a translator developed by Rockwell Collins. To ensure each Simulink or Stateflow construct has a well-defined semantics, the translator restricts the models that it will accept to those that can be translated unambiguously into Lustre.

Once in Lustre, the specification is loaded into an abstract syntax tree (AST) and a number of transformation passes are applied to it. Each transformation pass produces a new Lustre AST that is syntactically closer to the target specification language and preserves the semantics of the original Lustre specification. This allows all Lustre type checking and analysis tools to be used as debugging aids during the development of the translator. When the AST is sufficiently close to the target language, a pretty printer is used to output the target specification.

A model checker will consider every possible combination of inputs and state, making the verification equivalent to exhaustive testing of the model. If a property is not true, the model checker produces a counterexample showing how the property can be falsified.

We refer to our translator framework as a product family since most transformation passes are reused in the translators for each target language. Reuse of the transformation passes makes it much easier to support new target languages; we have developed new translators in a matter of days. The number of transformation passes depends on the similarity of the source and target languages and on the number of optimizations to be made. Our translators range in size from a dozen to over 60 passes.

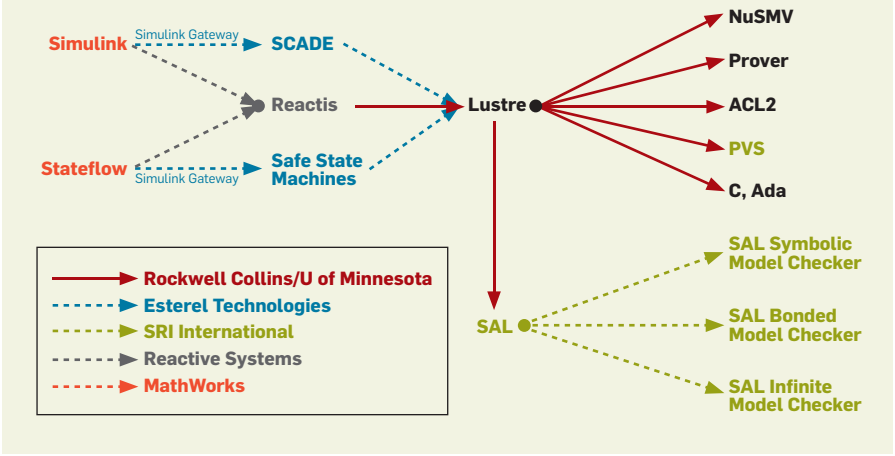
The translators produce highly optimized specifications appropriate for the target language. For example, when translating to NuSMV, the translator eliminates as much redundant internal state as possible, making it very efficient for BDD-based model checking. When translating to the PVS theorem prover, the specification is optimized for readability and to support the development of proofs in PVS. When generating executable C or Ada code, the code is optimized for execution speed on the target processor. These optimizations can have a dramatic effect on the target analysis tools. For example, optimization passes incorporated into the NuSMV translator reduced the time required for NuSMV to check one model from over 29 hours to less than a second.

However, some optimizations are better incorporated into the verification tools rather than the translator. For example, predicate abstraction[1] is a well-known technique for reducing the size of the reachable state space, but automating this during translation would require a tight interaction between our translator and the analysis tool to iteratively refine the predicates

Counterexample.

Step	1	2	3
Inputs			
Start	0	1	0
Clear	0	0	0
Door Closed	0	1	0
Steps to Cook	0	1	1
Outputs			
Mode	Setup	Cooking	Cooking
Steps Remaining	0	1	0

Figure 1. The translator framework.



based on the counterexamples. Since many model checkers already implement this technique, we have not tried to incorporate it into our translator framework.

We have developed tools to translate the counterexamples produced by the model checkers into two formats. The first is a simple spreadsheet that shows the inputs and outputs of the model for each step (similar to steps noted in the accompanying table). The second is a test script that can be read by the Reactis tool to step forward and backward through the counterexample in the Reactis simulator.

Our translator framework currently supports input models written in Simulink, Stateflow, and SCADE. It generates specifications for the NuSMV, SAL, and Prover model checkers, the PVS and ACL2 theorem provers, and C and Ada source code.

A Small Example

To make these ideas concrete, we present a very small example, the mode logic for a simple microwave oven shown in Figure 2. The microwave initially starts in *Setup* mode. It transitions to *Running* mode when the *Start* button is pressed and the *Steps Remaining* to cook (initially provided by the keypad entry subsystem) is greater than zero. On transition to *Running* mode, the controller enters either the *Cooking* or *Suspended* submode, depending on whether *Door Closed* is true. In *Cooking* mode, the controller decrements *Steps Remaining* on each step. If the door is opened in *Cooking* mode or the operator presses the *Clear* button, the controller enters the *Suspended* submode. From the *Suspended* submode, the operator can return to *Cooking* submode by pressing the *Start* button while the door is closed, or return to *Setup* mode by pressing the *Clear* button. When *Steps Remaining* decrements to zero, the controller exits *Running* mode and returns to *Setup* mode.

Since this model consists only of Boolean values (*Start*, *Clear*, *Door Closed*), enumerated types (mode), and two small integers (*Steps Remaining* and *Steps to Cook* range from 0 to 639, the largest value that can be entered on the keypad) it is well suited for analysis with a symbolic model checker such as NuSMV. A valuable property to check is that the door is always closed when the microwave is cooking. In CTL[1] (one of the property specification languages of NuSMV), this is written as:

```
AG(Cooking -> Door_Closed)
```

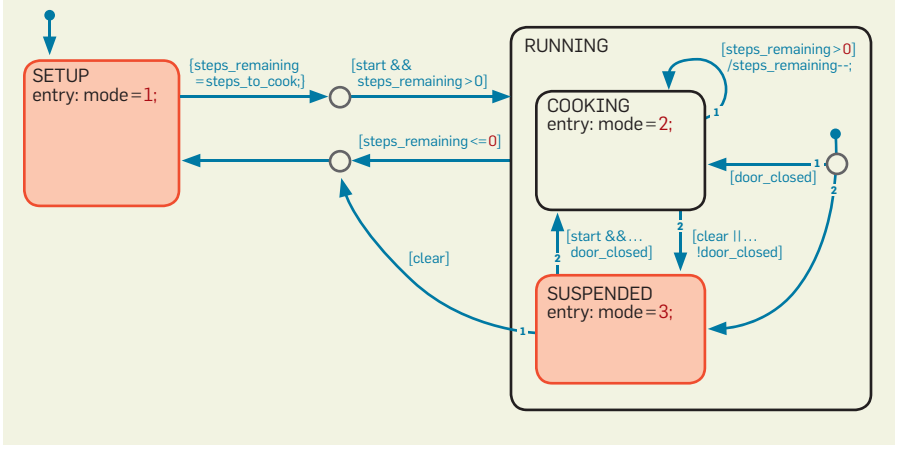
Translation of the model into NuSMV and checking this property takes only a few seconds and yields the counterexample shown in Table 1.

In step 2 of the counterexample, we see the value of *Start* change from 0 to 1, indicating the start button was pressed. Also in step 2, the door is closed and *Steps Remaining* takes on the value 1. As a result, the microwave enters *Cooking* mode in step 2. In step 3, the door is opened, but the microwave remains in *Cooking* mode, violating our safety property.

To better understand how this happened, we use Reactis to step through the generated counterexample. This reveals that instead of taking the transition from *Cooking* to *Suspended* when the door is opened, the microwave took the transition from *Cooking* to *Cooking* that decrements *Steps Remaining* because this transition has a higher priority (priority 1) than the transition from *Cooking* to *Suspended* (priority 2). Worse, the microwave would continue cooking with the door open until *Steps Remaining* becomes zero. Changing the priority of these two transitions and rerunning the model checker shows that in all possible states, the door is always closed if the microwave is cooking.

While this example is tiny, the two integers (*Steps Remaining* and *Steps to Cook*) still push its reachable state space to 9.8×10^6 states. Also note that the model checker does not necessarily find the "best" counterexample. It actually would have been clearer if the *Steps Remaining* had been set to a value larger than 1 in step 2. However, this counterexample is very typical. In the

Figure 2. Microwave mode logic.



production models that we have examined, very few counterexamples are longer than a few steps.

Case Studies

To be of any real value, model checking must be able to handle much larger problems. Three case studies on the application of our tools to industrial examples are described here. A fourth case study is discussed in Miller et al.[7]

ADGS-2100 Window Manager. One of the largest and most successful applications of our tools was to the ADGS-2100 Adaptive Display and Guidance System Window Manager.[13] In modern aircraft, pilots are provided aircraft status primarily through computerized display panels similar to those shown in Figure 3. The ADGS-2100 is a Rockwell Collins product that provides the heads-down and heads-up displays and display management software for next-generation commercial aircraft.

The Window Manager (WM) ensures that data from different applications is routed to the correct display panel. In normal operation, the WM determines which applications are being displayed in response to the pilot selections. However, in the case of a component failure, the WM also decides which information is most critical and routes this information from one of the redundant sources to the most appropriate display panel. The WM is essential to the safe flight of the aircraft. If the WM contains logic errors, critical flight information could be unavailable to the flight crew.

While very complex, the WM is specified in Simulink using only Booleans and enumerated types, making it ideal for verification using a BDD-based model checker such as NuSMV. The WM is composed of five main components that can be analyzed independently. These five components contain a total of 16,117 primitive Simulink blocks that are grouped into 4,295 instances of Simulink subsystems. The reachable state space of the five components ranges from 9.8×10^9 to 1.5×10^{37} states.

Ultimately, 563 properties about the WM were developed and checked, and 98 errors were found and corrected in early versions of the WM model. This verification was done early in the design process while the design was still changing. By the end of the project, the WM developers were checking the properties after every design change.

CerTA FCS Phase I. Our second case study was sponsored by the U.S. Air Force Research Laboratory (AFRL) under the Certification Technologies for Advanced Flight Critical Systems (CerTA FCS) program in order to compare the effectiveness of model checking and testing.[12] In this study, we applied our tools to the Operational Flight Program (OFP) of an unmanned aerial vehicle developed by Lockheed Martin Aerospace. The OFP is an adaptive flight control system that modifies its behavior in response to flight conditions. Phase I of the project concentrated on applying our tools to the Redundancy Management (RM) log-

Figure 3. Pilot display panels.

ic, which is based almost entirely on Boolean and enumerated types.

The RM logic was broken down into three components that could be analyzed individually. While relatively small (they contained a total of 169 primitive Simulink blocks organized into 23 subsystems, with reachable state spaces ranging from 2.1×10^4 to 6.0×10^{13} states), the RM logic was replicated in the OFP once for each of the 10 control surfaces on the aircraft, making it a significant portion of the OFP logic.

To compare the effectiveness of model checking and testing at discovering errors, this project had two independent verification teams, one that used testing and one that used model checking. The formal verification team developed a total of 62 properties from the OFP requirements and checked these properties with the NuSMV model checker, uncovering 12 errors in the RM logic. Of these 12 errors, four were classified by Lockheed Martin as severity 3 (only severity 1 and 2 can affect the safety of flight), two were classified as severity 4, two resulted in requirements changes, one was redundant, and three resulted from requirements that had not yet been implemented in the release of the software.

In similar fashion, the testing team developed a series of tests from the same OFP requirements. Even though the testing team invested almost half as much time in testing as the formal verification team spent in model checking, testing failed to find any errors. The main reason for this was that the demonstration was not a comprehensive test program. While some of these errors could be found through testing, the cost would be much higher, both to find and fix the errors. In addition, the errors found through model checking tended to be intermittent, near simultaneous, or combinatory sequences of failures that would be very difficult to detect through testing. The conclusion of both teams was that model checking was shown to be more cost effective than testing in finding design errors.

CerTa FCS Phase II. The purpose of Phase II of the CerTA FCS project was to investigate whether model checking could be used to verify large, numerically intensive models. In this study, the translation framework and model checking tools were used to verify important properties of the Effector Blender (EB) logic of an OFP for a UAV similar to that verified in Phase I.

Running a set of properties after each model revision is a quick and easy way to see if anything has been broken. We encourage our developers to "check your models early and check them often."

The EB is a central component of the OFP that generates the actuator commands for the aircraft's six control surfaces. It is a large, complex model that repeatedly manipulates a 3×6 matrix of floating point numbers. It inputs 32 floating point inputs and a 3×6 matrix of floating point numbers and outputs a 1×6 matrix of floating point numbers. It contains over 2,000 basic Simulink blocks organized into 166 Simulink subsystems, many of which are Stateflow models.

Because of its extensive use of floating point numbers and large state space, the EB cannot be verified using a BDD-based model checker such as NuSMV. Instead, the EB was analyzed using the Prover SMT-solver from Prover Technologies. Even with the additional capabilities of Prover, several new issues had to be addressed, the hardest being dealing with floating point numbers.

While Prover has powerful decision procedures for linear arithmetic with real numbers and bit-level decision procedures for integers, it does not have decision procedures for floating point numbers. Translating the floating point numbers into real numbers was rejected since much of the arithmetic in the EB is inherently nonlinear. Also, the use of real numbers would mask floating point arithmetic errors such as overflow and underflow.

Instead, the translator framework was extended to convert floating point numbers to fixed point numbers using a scaling factor provided by the OFP designers. The fixed point numbers were then converted to integers using bit-shifting to preserve their magnitude. While this allowed the EB to be verified using Prover's bit-level integer decision procedures, the results were unsound due to the loss of precision. However, if errors were found in the verified model, their presence could easily be confirmed in the original model. This allowed the verification to be used as a highly effective debugging step, even though it did not guarantee correctness.

Determining what properties to

verify was also a difficult problem. The requirements for the EB are actually specified for the combination of the EB and the aircraft model, but checking both the EB and the aircraft model exceeded the capabilities of the Prover Plug-In model checker. After consultation with the OFP designers, the verification team decided to verify whether the six actuator commands would always be within dynamically computed upper and lower limits. Violation of these properties would indicate a design error in the EB logic.

Even with these adjustments, the EB model was large enough that it had to be decomposed into a hierarchy of components several levels deep. The leaf nodes of this hierarchy were then verified using Prover Plug-In and their composition was manually verified using manual proofs. This approach also ensured that unsoundness could not be introduced through circular reasoning since Simulink enforces the absence of cyclic dependencies between atomic subsystems.

Ultimately, five errors in the EB design logic were discovered and corrected through model checking of these properties. In addition, several potential errors that were being masked by defensive design practices were found and corrected.

Lessons from the Case Studies

The case studies described here demonstrate that model checking can be effectively used to find errors early in the development process for many classes of models. In particular, even very complex models can be verified with BDD-based model checkers if they consist primarily of Boolean and enumerated types. Every industrial system we have studied contains large sections that either meet this constraint or can be made to meet it with some alteration.

For this class of models, the tools are simple enough for developers to use them routinely and without extensive training. In our experience, a single day of training and a low level of ongoing mentoring are usually sufficient. This also makes it practical to perform model checking early in the development process while a model is still changing.

Running a set of properties after each model revision is a quick and easy way to see if anything has been broken. We encourage our developers to "check your models early and check them often." The time spent model checking is recovered several times over by avoiding rework during unit and integration testing.

Since model checking examines every possible combination of input and state, it is also far more effective at finding design errors than testing, which can only check a small fraction of the possible inputs and states. As demonstrated by the CerTA FCS Phase I case study, it can also be more cost effective than testing.

Future Directions

There are many directions for further research. As illustrated in the CerTA FCS Phase II study, numerically intensive models still pose a challenge for model checking. SMT-based model checkers hold promise for verification of these systems, but the need to write properties that can be verified through induction over the state transition relation make them more difficult for developers to use.

Most industrial models used to generate code make extensive use of floating point numbers. Other models, particularly those that deal with spatial relationships such as navigation, make extensive use of trigonometric and other transcendental functions. A sound and efficient way of checking systems using floating point arithmetic and transcendental functions would be very helpful.

It can also be difficult to determine how many properties must be checked. Our experience has been that checking even a few properties will find errors, but that checking more properties will find more errors. Unlike testing for which many objective coverage criteria have been developed, completeness criteria for properties do not seem to exist. Techniques for developing or measuring the adequacy of a set of properties are needed.

As discussed in the CerTA FCS Phase II case study, the verification of very large models may be achieved by using model checking on subsystems and more traditional reasoning to compose the subsystems. Combining model checking and theorem proving in this way could be a very effective approach to the compositional verification of large systems.

Acknowledgments

The authors wish to thank Ricky Butler, Celeste Bellcastro, and Kelly Hayhurst of the NASA Langley Research Center, Mats Heimdahl, Yunja Choi, Anjali Joshi, Ajitha Rajan, Sanjai Rayadurgam, and Jeff Thompson of the University of Minnesota, Eric Danielson, John Innis, Ray Kamin, David Lempia, Alan Tribble, Lucas Wagner, and Matt Wilding of Rockwell Collins, Bruce Krogh of CMU, Vince Crum, Wendy Chou, Ray Bortner, and David Homan of the Air Force Research Lab, and Greg Tallant and Walter Storm of Lockheed Martin for their contributions and support.

This work was supported in part by the NASA Langley Research Center under contract NCC-01001 of the Aviation Safety Program (AvSP) and by the Air Force Research Lab under contract FA8650-05-C-3564 of the Certification Technologies for Advanced Flight Control Systems program (CerTA FCS) [88ABW-2009-2730].

References

1. Clarke, E., Grumberg, O. and Peled, D. *Model Checking.* The MIT Press, Cambridge, MA, 2001.
2. Esterel Technologies. SCADE Suite Product Description; http://www.estereltechnolgies.com/
3. Halbwachs, N., Caspi, P., Raymond, P and Pilaud, D. The synchronous dataflow programming language Lustre. In *Proceedings of the IEEE* 79, 9 (1991) 1305–1320.
4. Holzmann, G. *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley Professional, 2003.
5. IRST. The NuSMV Model Checker; http://nusmv.irst.itc.it/
6. The Mathworks Simulink Product Description; http://www.mathworks.com/
7. Miller, S., Anderson, E., Wagner, L., Whalen, M. and Heimdahl, M.P.E. Formal verification of flight critical software. In *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit* (San Francisco, CA, Aug. 15–18, 2005).
8. Miller, S., Tribble, A., Whalen, M. and Heimdahl, M.P.E. Proving the Shalls. *International Journal on Software Tools for Technology Transfer* (Feb. 2006).
9. Prover Technology. Prover Plug-In Product Description; http://www.prover.com/
10. Reactive Systems, Inc.; http://www.reactive-systems.com/
11. SRI International. Symbolic Analysis Laboratory; http://sal.csl.sri.com/
12. Whalen, M., Cofer, D., Miller, S. Krogh, B., and Storm, W. Integration of formal analysis into a model-based software development process. In *Proceedings of the 12th International Workshop on Formal Methods for Industrial Critical Systems* (Berlin, Germany, July 1–2, 2007).
13. Whalen, M., Innis, J., Miller, S. and Wagner, L. ADGS-2100 Adaptive Display & Guidance System Window Manager Analysis. NASA Contractor Report CR-2006-213952 (Feb. 2006); http://shemesh.larc.nasa.gov/fm/fm-collins-pubs.html/

Steven P. Miller (spmiller@rockwellcollins.com) is a principal software engineer in the Advanced Technology Center at Rockwell Collins, Cedar Rapids, IA.

Michael W. Whalen (whalen@cs.umn.edu) is the program director at the University of Minnesota Software Engineering Center, Minneapolis, MN.

Darren D. Cofer (ddcofer@rockwellcollins.com) is a principal systems engineer in the Advanced Technology Center at Rockwell Collins, Cedar Rapids, IA.

DOI:10.1145/1646353.1646374

How Coverity built a bug-finding tool, and a business, around the unlimited supply of bugs in software systems.

BY AL BESSEY, KEN BLOCK, BEN CHELF, ANDY CHOU, BRYAN FULTON, SETH HALLEM, CHARLES HENRI-GROS, ASYA KAMSKY, SCOTT MCPEAK, AND DAWSON ENGLER

A Few Billion Lines of Code Later

Using Static Analysis to Find Bugs in the Real World

IN 2002, COVERITY commercialized[3] a research static bug-finding tool.[6,9] Not surprisingly, as academics, our view of commercial realities was not perfectly accurate. However, the problems we encountered were not the obvious ones. Discussions with tool researchers and system builders suggest we were not alone in our naïveté. Here, we document some of the more important examples of what we learned developing and commercializing an industrial-strength bug-finding tool.

We built our tool to find generic errors (such as memory corruption and data races) and system-specific or interface-specific violations (such as violations of function-ordering constraints). The tool, like all static bug finders, leveraged the fact that programming rules often map clearly to source code; thus static inspection can find many of their violations. For example, to check the rule "acquired locks must be released," a checker would look for relevant operations (such as `lock()` and `unlock()`) and inspect the code path after flagging rule disobedience (such as `lock()` with no `unlock()` and double locking).

For those who keep track of such things, checkers in the research system typically traverse program paths (flow-sensitive) in a forward direction, going across function calls (inter-procedural) while keeping track of call-site-specific information (context-sensitive) and toward the end of the effort had some of the support needed to detect when a path was infeasible (path-sensitive).

A glance through the literature reveals many ways to go about static bug finding.[1,2,4,7,8,11] For us, the central religion was results: If it worked, it was good, and if not, not. The ideal: check millions of lines of code with little manual setup and find the maximum number of serious true errors with the minimum number of false reports. As much as possible, we avoided using annotations or specifications to reduce manual labor.

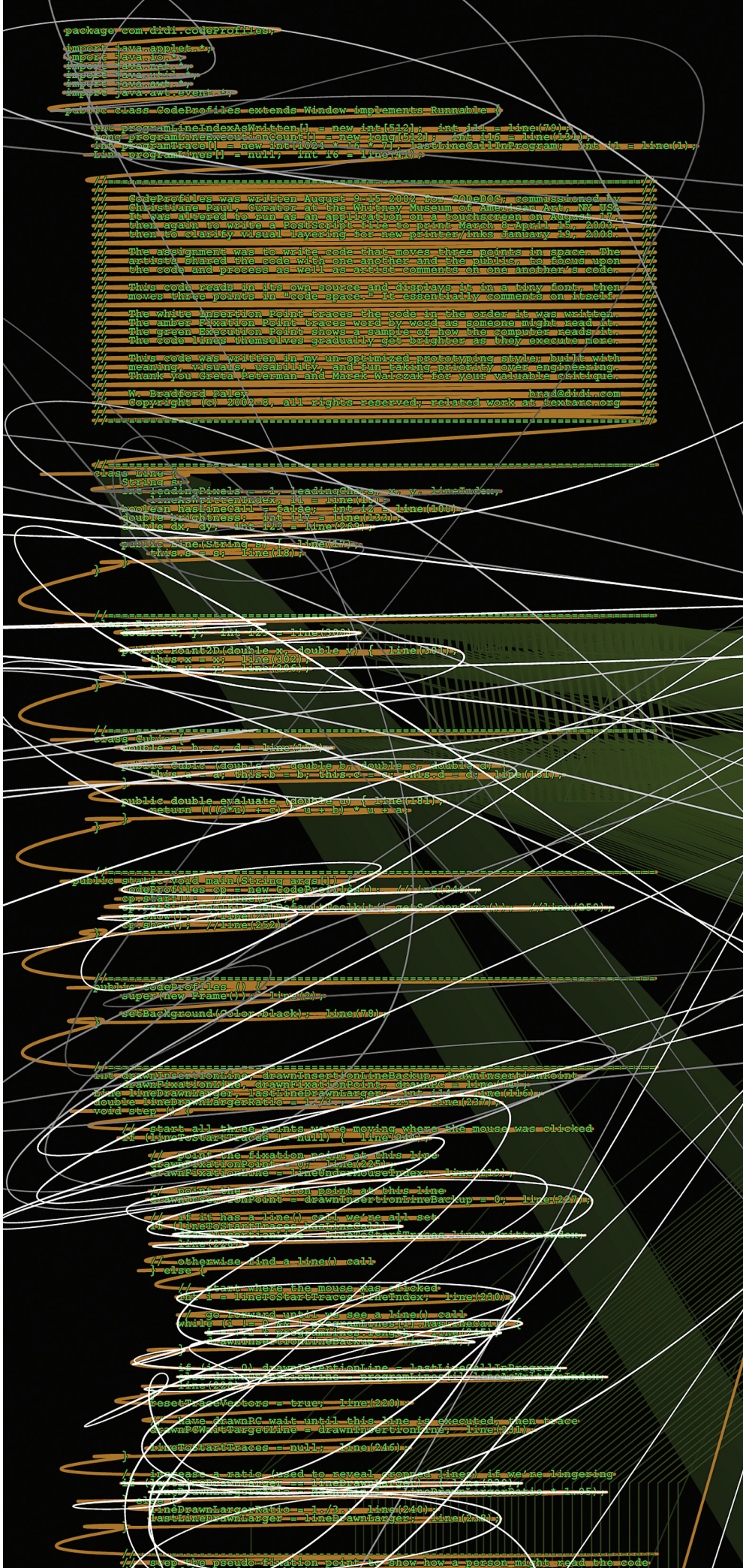
Like the PREfix product,[2] we were also unsound. Our product did not verify the absence of errors but rather tried to find as many of them as possible. Unsoundness let us focus on handling the easiest cases first, scaling up as it proved useful. We could ignore code constructs that led to high rates of false-error messages (false positives) or analysis complexity, in the extreme skipping problematic code entirely (such as assembly statements, functions, or even entire files). Circa 2000, unsoundness was controversial in the research community, though it has since become almost a de facto tool bias for commercial products and many research projects.

Initially, publishing was the main force driving tool development. We would generally devise a set of checkers or analysis tricks, run them over a few

million lines of code (typically Linux), count the bugs, and write everything up. Like other early static-tool researchers, we benefited from what seems an empirical law: Assuming you have a reasonable tool, if you run it over a large, previously unchecked system, you will always find bugs. If you don't, the immediate knee-jerk reaction is that something must be wrong. Misconfiguration? Mistake with macros? Wrong compilation target? If programmers must obey a rule hundreds of times, then without an automatic safety net they cannot avoid mistakes. Thus, even our initial effort with primitive analysis found hundreds of errors.

This is the research context. We now describe the commercial context. Our rough view of the technical challenges of commercialization was that given that the tool would regularly handle "large amounts" of "real" code, we needed only a pretty box; the rest was a business issue. This view was naïve. While we include many examples of unexpected obstacles here, they devolve mainly from consequences of two main dynamics:

First, in the research lab a few people check a few code bases; in reality many check many. The problems that show up when thousands of programmers use a tool to check hundreds (or even thousands) of code bases do not show up when you and your co-authors check only a few. The result of summing many independent random variables? A Gaussian distribution, most of it not on the points you saw and adapted to in the lab. Furthermore, Gaussian distributions have tails. As the number of samples grows, so, too, does the absolute number of points several standard deviations from the mean. The unusual starts to occur with increasing frequency.



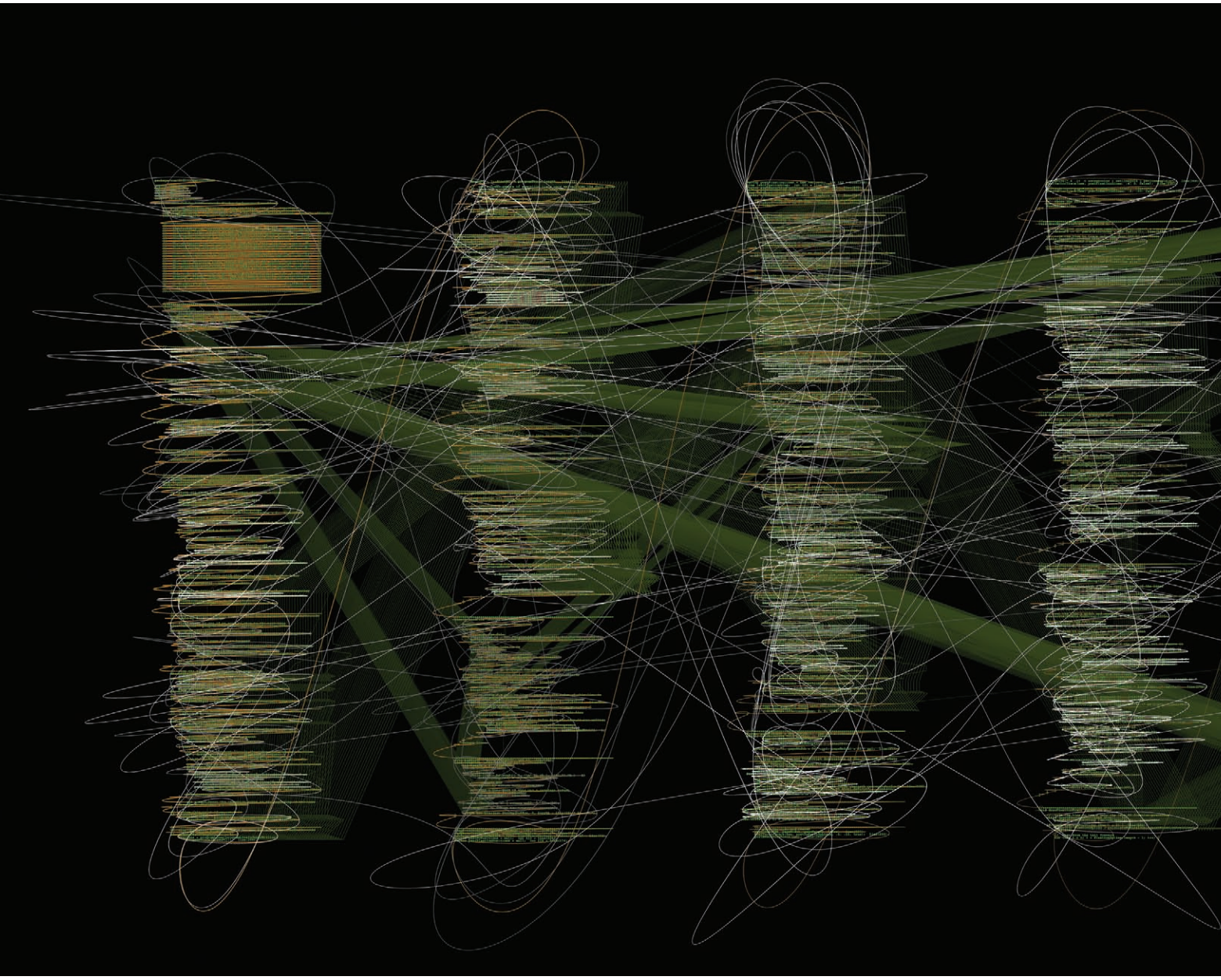
W. Bradford Paley's CodeProfiles was originally commissioned for the Whitney Museum of American Art's "CODeDOC" Exhibition and later included in MoMA's "Design and the Elastic Mind" exhibition. CodeProfiles explores the space of code itself; the program reads its source into memory, traces three points as they once moved through that space, then prints itself on the page.

For code, these features include problematic idioms, the types of false positives encountered, the distance of a dialect from a language standard, and the way the build works. For developers, variations appear in raw ability, knowledge, the amount they care about bugs, false positives, and the types of both. A given company won't deviate in all these features but, given the number of features to choose from, often includes at least one weird oddity. Weird is not good. Tools want expected. Expected you can tune a tool to handle; surprise interacts badly with tuning assumptions.

Second, in the lab the user's values, knowledge, and incentives are those of the tool builder, since the user and the builder are the same person. Deployment leads to severe fission; users often have little understanding of the tool and little interest in helping develop it (for reasons ranging from simple skepticism to perverse reward incentives) and typically label any error message they find confusing as false. A tool that works well under these constraints looks very different from one tool builders design for themselves.

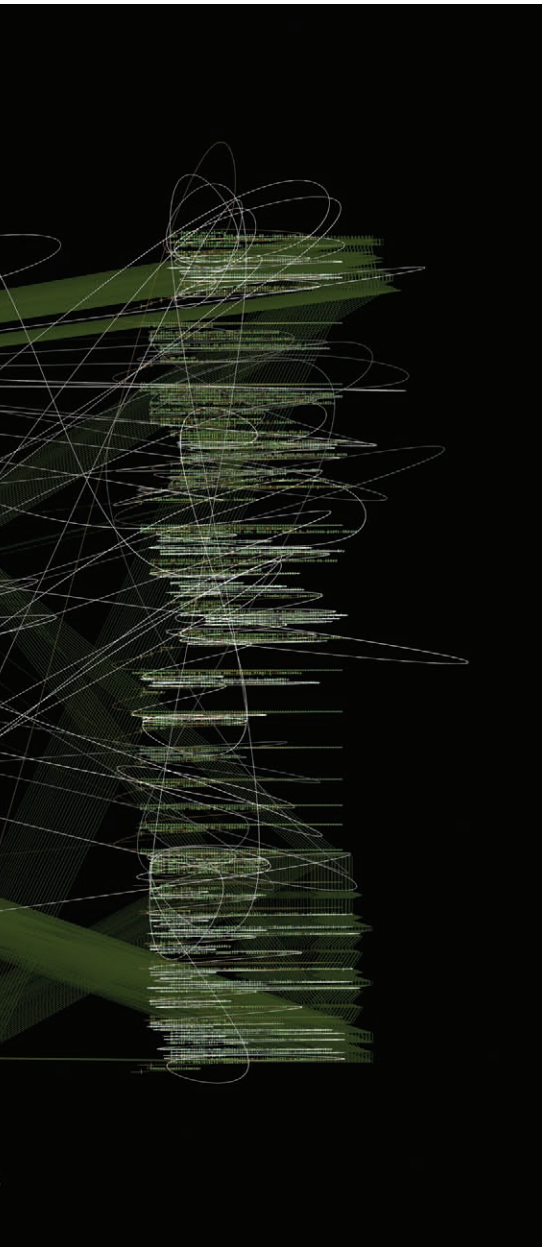
However, for every user who lacks the understanding or motivation one might hope for, another is eager to understand how it all works (or perhaps already does), willing to help even beyond what one might consider reasonable. Such champions make sales as easily as their antithesis blocks them. However, since their main requirements tend to be technical (the tool must work) the reader likely sees how to make them happy, so we rarely discuss them here.

Most of our lessons come from two different styles of use: the initial trial of the tool and how the company uses the tool after buying it. The trial is a pre-sale demonstration that attempts to show that the tool works well on a potential customer's code. We generally ship a salesperson and an engineer to the customer's site. The engineer configures the tool and runs it over a given code base and presents results soon after. Initially, the checking run would happen



in the morning, and the results meeting would follow in the afternoon; as code size at trials grows it's not uncommon to split them across two (or more) days.

Sending people to a trial dramatically raises the incremental cost of each sale. However, it gives the non-trivial benefit of letting us educate customers (so they do not label serious, true bugs as false positives) and do real-time, ad hoc workarounds of weird customer system setups.



"CODEPROFILES" BY W. BRADFORD PALEY

The trial structure is a harsh test for any tool, and there is little time. The checked system is large (millions of lines of code, with 20–30MLOC a possibility). The code and its build system are both difficult to understand. However, the tool must routinely go from never seeing the system previously to getting good bugs in a few hours. Since we present results almost immediately after the checking run, the bugs must be good with few false positives; there is no time to cherry pick them.

Furthermore, the error messages must be clear enough that the sales engineer (who didn't build the checked system or the tool) can diagnose and explain them in real time in response to "What about this one?" questions.

The most common usage model for the product has companies run it as part of their nightly build. Thus, most require that checking runs complete in 12 hours, though those with larger code bases (10+MLOC) grudgingly accept 24 hours. A tool that cannot analyze at least 1,400 lines of code per minute makes it difficult to meet these targets. During a checking run, error messages are put in a database for subsequent triaging, where users label them as true errors or false positives. We spend significant effort designing the system so these labels are automatically reapplied if the error message they refer to comes up on subsequent runs, despite code-dilating edits or analysis-changing bug-fixes to checkers.

As of this writing (December 2009), approximately 700 customers have licensed the Coverity Static Analysis product, with somewhat more than a billion lines of code among them. We estimate that since its creation the tool has analyzed several billion lines of code, some more difficult than others.

Caveats. Drawing lessons from a single data point has obvious problems. Our product's requirements roughly form a "least common denominator" set needed by any tool that uses nontrivial analysis to check large amounts of code across many organizations; the tool must find and parse the code, and users must be able to understand error messages. Further, there are many ways to handle the problems we have encountered, and our way may not be the best one. We discuss our methods more for specificity than as a claim of solution.

Finally, while we have had success as a static-tools company, these are small steps. We are tiny compared to mature technology companies. Here, too, we have tried to limit our discussion to conditions likely to be true in a larger setting.

Laws of Bug Finding

The fundamental law of bug finding is No Check = No Bug. If the tool can't check a system, file, code path, or given property, then it won't find bugs in it. Assuming a reasonable tool, the first order bound on bug counts is just how much code can be shoved through the tool. Ten times more code is 10 times more bugs.

We imagined this law was as simple a statement of fact as we needed. Unfortunately, two seemingly vacuous corollaries place harsh first-order bounds on bug counts:

Law: You can't check code you don't see. It seems too trite to note that checking code requires first finding it... until you try to do so consistently on many large code bases. Probably the most reliable way to check a system is to grab its code during the build process; the build system knows exactly which files are included in the system and how to compile them. This seems like a simple task. Unfortunately, it's often difficult to understand what an ad hoc, homegrown build system is doing well enough to extract this information, a difficulty compounded by the near-universal absolute edict: "No, you can't touch that." By default, companies refuse to let an external force modify anything; you cannot modify their compiler path, their broken makefiles (if they have any), or in any way write or reconfigure anything other than your own temporary files. Which is fine, since if you need to modify it, you most likely won't understand it.

Further, for isolation, companies often insist on setting up a test machine for you to use. As a result, not infrequently the build you are given to check does not work in the first place, which you would get blamed for if you had touched anything.

Our approach in the initial months of commercialization in 2002 was a low-tech, read-only replay of the build commands: run `make`, record its output in a file, and rewrite the invocations to their compiler (such as gcc) to instead call our checking tool, then rerun everything. Easy and simple. This approach worked perfectly in the lab and for a small number of our earliest customers. We then had the fol-

lowing conversation with a potential customer:

"How do we run your tool?"

"Just type 'make' and we'll rewrite its output."

"What's 'make'? We use ClearCase."

"Uh, What's ClearCase?"

This turned out to be a chasm we couldn't cross. (Strictly speaking, the customer used 'ClearMake,' but the superficial similarities in name are entirely unhelpful at the technical level.) We skipped that company and went to a few others. They exposed other problems with our method, which we papered over with 90% hacks. None seemed so troublesome as to force us to rethink the approach—at least until we got the following support call from a large customer:

"Why is it when I run your tool, I have to reinstall my Linux distribution from CD?"

This was indeed a puzzling question. Some poking around exposed the following chain of events: the company's make used a novel format to print out the absolute path of the directory in which the compiler ran; our script misparsed this path, producing the empty string that we gave as the destination to the Unix "cd" (change directory) command, causing it to change to the top level of the system; it ran "rm -rf *" (recursive delete) during compilation to clean up temporary files; and the build process ran as root. Summing these points produces the removal of all files on the system.

The right approach, which we have used for the past seven years, kicks off the build process and intercepts every system call it invokes. As a result, we can see everything needed for checking, including the exact executables invoked, their command lines, the directory they run in, and the version of the compiler (needed for compiler-bug workarounds). This control makes it easy to grab and precisely check all source code, to the extent of automatically changing the language dialect on a per-file basis.

To invoke our tool users need only call it with their build command as an argument:

```
cov-build <build command>
```

We thought this approach was bulletproof. Unfortunately, as the astute reader has noted, it requires a command prompt. Soon after implementing it we went to a large company, so large it had a hyperspecialized build engineer, who engaged in the following dialogue:

"How do I run your tool?"

"Oh, it's easy. Just type 'cov-build' before your build command."

"Build command? I just push this [GUI] button..."

A misunderstood explanation means the error is ignored or, worse, transmuted into a false positive.

Social vs. technical. The social restriction that you cannot change anything, no matter how broken it may be, forces ugly workarounds. A representative example is: Build interposition on Windows requires running the compiler in the debugger. Unfortunately, doing so causes a very popular windows C++ compiler—Visual Studio C++ .NET 2003—to prematurely exit with a bizarre error message. After some high-stress fussing, it turns out that the compiler has a use-after-free bug, hit when code used a Microsoft-specific C language extension (certain invocations of its #using directive). The compiler runs fine in normal use; when it reads the freed memory, the original contents are still there, so everything works. However, when run with the debugger, the compiler switches to using a "debug malloc," which on each free call sets the freed memory contents to a garbage value. The subsequent read returns this value, and the compiler blows up with a fatal error. The sufficiently perverse reader can no doubt guess the "solution."[a]

Law: You can't check code you can't parse. Checking code deeply requires understanding the code's semantics. The most basic requirement is that you parse it. Parsing is considered a solved problem. Unfortunately, this view is naïve, rooted in the widely believed myth that programming languages exist.

The C language does not exist; neither does Java, C++, and C#. While a language may exist as an abstract idea, and even have a pile of paper (a standard) purporting to define it, a standard is not a compiler. What language do people write code in? The character strings accepted by their compiler. Further, they equate compilation with certification. A file their compiler does

a Immediately after process startup our tool writes 0 to the memory location of the "in debugger" variable that the compiler checks to decide whether to use the debug malloc.

not reject has been certified as "C code" no matter how blatantly illegal its contents may be to a language scholar. Fed this illegal not-C code, a tool's C front-end will reject it. This problem is the tool's problem.

Compounding it (and others) the person responsible for running the tool is often not the one punished if the checked code breaks. (This person also often doesn't understand the checked code or how the tool works.) In particular, since our tool often runs as part of the nightly build, the build engineer managing this process is often in charge of ensuring the tool runs correctly. Many build engineers have a single concrete metric of success: that all tools terminate with successful exit codes. They see Coverity's tool as just another speed bump in the list of things they must get through. Guess how receptive they are to fixing code the "official" compiler accepted but the tool rejected with a parse error? This lack of interest generally extends to any aspect of the tool for which they are responsible.

Many (all?) compilers diverge from the standard. Compilers have bugs. Or are very old. Written by people who misunderstand the specification (not just for C++). Or have numerous extensions. The mere presence of these divergences causes the code they allow to appear. If a compiler accepts construct X, then given enough programmers and code, eventually X is typed, not rejected, then encased in the code base, where the static tool will, not helpfully, flag it as a parse error.

The tool can't simply ignore divergent code, since significant markets are awash in it. For example, one enormous software company once viewed conformance as a competitive disadvantage, since it would let others make tools usable in lieu of its own. Embedded software companies make great tool customers, given the bug aversion of their customers; users don't like it if their cars (or even their toasters) crash. Unfortunately, the space constraints in such systems and their tight coupling to hardware have led to an astonishing oeuvre of enthusiastically used compiler extensions.

Finally, in safety-critical software systems, changing the compiler often requires costly re-certification. Thus, we routinely see the use of decades-old compilers. While the languages these compilers accept have interesting features, strong concordance with a modern language standard is not one of them. Age begets new problems. Realistically, diagnosing a compiler's divergences requires having a copy of the compiler. How do you purchase a license for a compiler 20 versions old? Or whose company has gone out of business? Not through normal channels. We have literally resorted to buying copies off eBay.

This dynamic shows up in a softer way with non-safety-critical systems; the larger the code base, the more the sales force is rewarded for a sale, skewing sales toward such systems. Large code bases take a while to build and often get tied to the compiler used when they were born, skewing the average age of the compilers whose languages we must accept.

If divergence-induced parse errors are isolated events scattered here and there, then they don't matter. An unsound tool can skip them. Unfortunately, failure often isn't modular. In a sad, too-common story line, some crucial, purportedly "C" header file contains a blatantly illegal non-C construct. It gets included by all files. The no-longer-potential customer is treated to a constant stream of parse errors as your compiler rips through the customer's source files, rejecting each in turn. The customer's derisive stance is, "Deep source code analysis? Your tool can't even compile code. How can it find bugs?" It may find this event so amusing that it tells many friends.

Tiny set of bad snippets seen in header files. One of the first examples we encountered of illegal-construct-in-key-header file came up at a large networking company

```
// "redefinition of parameter 'a'"
void foo(int a, int a);
```

The programmer names `foo`'s first formal parameter `a` and, in a form of lexical locality, the second as well. Harmless. But any conformant compiler will reject this code. Our tool certainly did. This is not helpful; compiling no files means finding no bugs, and people don't need your tool for that. And, because its compiler accepted it, the potential customer blamed us.

Here's an opposite, less-harmless case where the programmer is trying to make two different things the same

```
typedef char int;
```

("Useless type name in empty declaration.")

And one where readability trumps the language spec

```
unsigned x = 0xdead _ beef;
```

("Invalid suffix '_beef' on integer constant.")

From the embedded space, creating a label that takes no space

```
void x;
```

("Storage size of 'x' is not known.")

Another embedded example that controls where the space comes from

```
unsigned x @ "text";
```

("Stray '@' in program.")

A more advanced case of a nonstandard construct is

```
Int16 ErrSetJump(ErrJumpBuf buf)
 = { 0x4E40 + 15, 0xA085; }
```

It treats the hexadecimal values of machine-code instructions as program source.

The award for most widely used extension should, perhaps, go to Microsoft support for precompiled headers. Among the most nettlesome troubles is that the compiler skips all the text before an inclusion of a precompiled header. The implication of this behavior is that the following code can be compiled without complaint:

```
I can put whatever I want here.
It doesn't have to compile.
If your compiler gives an error,
   it sucks.
#include  <some-precompiled-
   header.h>
```

Microsoft's on-the-fly header fabrication makes things worse.

Assembly is the most consistently troublesome construct. It's already non-portable, so compilers seem to almost deliberately use weird syntax, making it difficult to handle in a general way. Unfortunately, if a programmer uses assembly it's probably to write a widely used function, and if the programmer does it, the most likely place to put it is in a widely used

header file. Here are two ways (out of many) to issue a mov instruction

```
// First way
foo() {
      _ _asm mov eax, eab
      mov eax, eab;
}

// Second way
#pragma asm
_ _asm [ mov eax, eab mov
eax, eab ]
#pragma end _ asm
```

The only thing shared in addition to mov is the lack of common textual keys that can be used to elide them.

We have thus far discussed only C, a simple language; C++ compilers diverge to an even worse degree, and we go to great lengths to support them. On the other hand, C# and Java have been easier, since we analyze the bytecode they compile to rather than their source.

How to parse not-C with a C front-end. OK, so programmers use extensions. How difficult is it to solve this problem? Coverity has a full-time team of some of its sharpest engineers to firefight this banal, technically uninteresting problem as their sole job. They're never done.[b]

We first tried to make the problem someone else's problem by using the Edison Design Group (EDG) C/C++ front-end to parse code.[5] EDG has worked on how to parse real C code since 1989 and is the de facto industry standard front-end. Anyone deciding to not build a homegrown front-end will almost certainly license from EDG. All those who do build a homegrown front-end will almost certainly wish they did license EDG after a few experiences with real code. EDG aims not just for mere feature compatibility but for version-specific bug compatibility across a range of compilers. Its front-end probably resides near the limit of what a profitable company can do in terms of front-end gyrations.

Unfortunately, the creativity of compiler writers means that despite two decades of work EDG still regularly meets defeat when trying to parse real-world large code bases.[c] Thus, our next step is for each supported compiler, we write a set of "transformers" that mangle its personal language into something closer to what EDG can parse. The most common transformation simply rips out the offending construct. As one measure of how much C does not exist, the table here counts the lines of transformer code needed to make the languages accepted by 18 widely used compilers look vaguely like C. A line of transformer code was almost always written only when we were burned to a degree that was difficult to work around. Adding each new compiler to our list of "supported" compilers almost always requires writing some kind of transformer. Unfortunately, we sometimes need a deeper view of semantics so are forced to hack EDG directly. This method is a last resort. Still, at last count (as of early 2009) there were more than 406(!) places in the front-end where we had an #ifdef COVERITY to handle a specific, unanticipated construct.

EDG is widely used as a compiler front-end. One might think that for customers using EDG-based compilers we would be in great shape. Unfortunately, this is not necessarily the case. Even ignoring the fact that compilers based on EDG often modify EDG in idiosyncratic ways, there is no single "EDG front-end" but rather many versions and possible configurations that often accept a slightly different language variant than the (often newer) version we use. As a Sisyphean twist, assume we cannot work around and report an incompatibility. If EDG then considers the problem important enough to fix, it will roll it together with other patches into a new version.

So, to get our own fix, we must upgrade the version we use, often causing divergence from other unupgraded EDG compiler front-ends, and more issues ensue.

Social versus technical. Can we get customer source code? Almost always, no. Despite nondisclosure agreements, even for parse errors and preprocessed code, though perhaps because we are viewed as too small to sue to recoup damages. As a result, our sales engineers must type problems in reports from memory. This works as well as you might expect. It's worse for performance problems, which often show up only in large-code settings. But one shouldn't complain, since classified systems make things even worse. Can we send someone onsite to look at the code? No. You listen to recited syntax on the phone.

Bugs

Do bugs matter? Companies buy bug-finding tools because they see bugs as bad. However, not everyone agrees that bugs matter. The following event has occurred during numerous trials. The tool finds a clear, ugly error (memory corruption or use-after-free) in important code, and the interaction with the customer goes like thus:

"So?"

"Isn't that bad? What happens if you hit it?"

"Oh, it'll crash. We'll get a call." [Shrug.]

If developers don't feel pain, they often don't care. Indifference can arise from lack of accountability; if QA cannot reproduce a bug, then there is no blame. Other times, it's just odd:

"Is this a bug?"

"I'm just the security guy."

"That's not a bug; it's in third-party code."

"A leak? Don't know. The author left years ago..."

No, your tool is broken; that is not a bug. Given enough code, any bug-

b Anecdotally, the dynamic memory-checking tool Purify[10] had an analogous struggle at the machine-code level, where Purify's developers expended significant resources reverse engineering the various activation-record layouts used by different compilers.

c Coverity won the dubious honor of being the single largest source of EDG bug reports after only three years of use.

Lines of code per transformer for 18 common compilers we support.

160 QNX	280 HP-UX	285 picc.cpp
294 sun.java.cpp	384 st.cpp	334 cosmic.cpp
421 intel.cpp	457 sun.cpp	603 iccmsa.cpp
629 bcc.cpp	673 diab.cpp	756 xlc.cpp
912 ARM	914 GNU	1294 Microsoft
1425 keil.cpp	1848 cw.cpp	1665 Metrowerks

finding tool will uncover some weird examples. Given enough coders, you'll see the same thing. The following utterances were culled from trial meetings:

Upon seeing an error report saying the following loop body was dead code

```
foo(i = 1; i < 0; i++)
  ... deadcode ...
```

"No, that's a false positive; a loop executes at least once."

For this memory corruption error (32-bit machine)

```
int a[2], b;
memset(a, 0, 12);
```

"No, I meant to do that; they are next to each other."

For this use-after-free

```
free(foo);
foo->bar = ...;
```

"No, that's OK; there is no malloc call between the free and use."

As a final example, a buffer overflow checker flagged a bunch of errors of the form

```
unsigned p[4];
...
p[4] = 1;
```

"No, ANSI lets you write 1 past the end of the array."

After heated argument, the programmer said, "We'll have to agree to disagree." We could agree about the disagreement, though we couldn't quite comprehend it. The (subtle?) interplay between 0-based offsets and buffer sizes seems to come up every few months.

While programmers are not often so egregiously mistaken, the general trend holds; a not-understood bug report is commonly labeled a false positive, rather than spurring the programmer to delve deeper. The result? We have completely abandoned some analyses that might generate difficult-to-understand reports.

How to handle cluelessness. You cannot often argue with people who are sufficiently confused about technical matters; they think you are the one who doesn't get it. They also tend to get emotional. Arguing reliably kills sales. What to do? One trick is to try to organize a large meeting so their peers do the work for you. The more people in the room, the more likely there is someone very smart and respected and cares (about bugs and about the given code), can diagnose an error (to counter arguments it's a false positive), has been burned by a similar error, loses his/her bonus for errors, or is in another group (another potential sale).

...it's not uncommon for tool improvement to be viewed as "bad" or at least a problem.

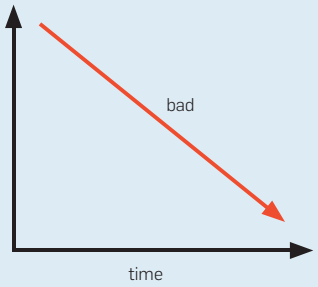
Further, a larger results meeting increases the probability that anyone laid off at a later date attended it and saw how your tool worked. True story: A networking company agreed to buy the Coverity product, and one week later laid off 110 people (not because of us). Good or bad? For the fired people it clearly wasn't a happy day. However, it had a surprising result for us at a business level; when these people were hired at other companies some suggested bringing the tool in for a trial, resulting in four sales.

What happens when you can't fix all the bugs? If you think bugs are bad enough to buy a bug-finding tool, you will fix them. Not quite. A rough heuristic is that fewer than 1,000 bugs, then fix them. More? The baseline is to record the current bugs, don't fix them but do fix any new bugs. Many companies have independently come up with this practice, which is more rational than it seems. Having a lot of bugs usually requires a lot of code. Much of it won't have changed in a long time. A reasonable, conservative heuristic is if you haven't touched code in years, don't modify it (even for a bug fix) to avoid causing any breakage.

A surprising consequence is it's not uncommon for tool improvement to be viewed as "bad" or at least a problem. Pretend you are a manager. For anything bad you can measure, you want it to diminish over time. This means you are improving something and get a bonus.

You may not understand technical issues that well, and your boss certainly doesn't understand them. Thus, you want a simple graph that looks like Figure 1; no manager gets a bonus for Figure 2. Representative story: At company X, version 2.4 of the tool found approximately 2,400 errors, and over time the company fixed about 1,200 of them. Then it upgraded to version 3.6. Suddenly there were 3,600 errors. The manager was furious for two reasons: One, we "undid" all the work his people

Figure 1. Bugs down over time = manager bonus.



had done, and two, how could we have missed them the first time?

How do upgrades happen when more bugs is no good? Companies independently settle on a small number of upgrade models:

Never. Guarantees "improvement";

Never before a release (where it would be most crucial). Counterintuitively happens most often in companies that believe the tool helps with release quality in that they use it to "gate" the release;

Never before a meeting. This is at least socially rational;

Upgrade, then roll back. Seems to happen at least once at large companies; and

Upgrade only checkers where they fix most errors. Common checkers include use-after-free, memory corruption, (sometimes) locking, and (sometimes) checkers that flag code contradictions.

Do missed errors matter? If people don't fix all the bugs, do missed errors (false negatives) matter? Of course not; they are invisible. Well, not always. Common cases: Potential customers intentionally introduced bugs into the system, asking "Why didn't you find it?" Many check if you find important past bugs. The easiest sale is to a group whose code you are checking that was horribly burned by a specific bug last week, and you find it. If you don't find it? No matter the hundreds of other bugs that may be the next important bug.

Here is an open secret known to bug finders: The set of bugs found by tool A is rarely a superset of another tool B, even if A is much better than B. Thus, the discussion gets pushed from "A is better than B" to "A finds some things, B finds some things" and does not help the case of A.

Adding bugs can be a problem; losing already inspected bugs is always a problem, even if you replace them with many more new errors. While users know in theory that the tool is "not a verifier," it's very different when the tool demonstrates this limitation, good and hard, by losing a few hundred known errors after an upgrade.

The easiest way to lose bugs is to add just one to your tool. A bug that causes false negatives is easy to miss. One such bug in how our early research tool's internal representation handled array references meant the analysis ignored most array uses for more than nine months. In our commercial product, blatant situations like this are prevented through detailed unit testing, but uncovering the effect of subtle bugs is still difficult because customer source code is complex and not available.

Churn

Users really want the same result from run to run. Even if they changed their code base. Even if they upgraded the tool. Their model of error messages? Compiler warnings. Classic determinism states: the same input + same function = same result. What users want: different input (modified code base) + different function (tool version) = same result. As a result, we find upgrades to be a constant headache. Analysis changes can easily cause the set of defects found to shift. The new-speak term we use internally is "churn." A big change from academia is that we spend considerable time and energy worrying about churn when modifying checkers. We try to cap churn at less than 5% per release. This goal means large classes of analysis tricks are disallowed since they cannot obviously guarantee minimal effect on the bugs found. Randomization is verboten, a tragedy given that it provides simple, elegant solutions to many of the exponential problems we encounter. Timeouts are also bad and sometimes used as a last resort but never encouraged.

Myth: More analysis is always good. While nondeterministic analysis might cause problems, it seems that adding more deterministic analysis is always good. Bring on path sensitivity! Theorem proving! SAT solvers! Unfortunately, no.

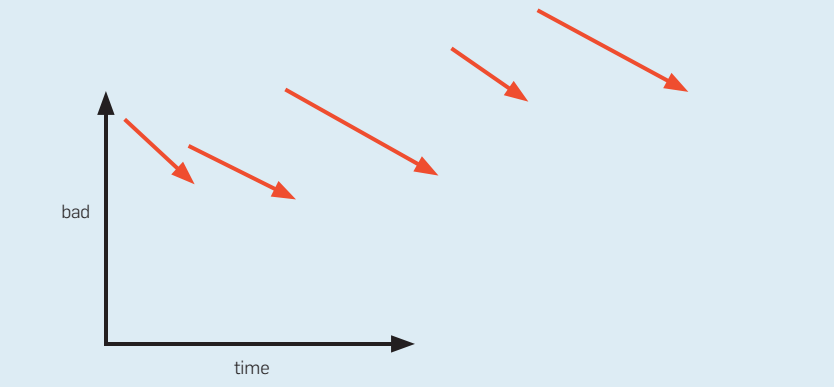
At the most basic level, errors found with little analysis are often better than errors found with deeper tricks. A good error is probable, a true error, easy to diagnose; best is difficult to misdiagnose. As the number of analysis steps increases, so, too, does the chance of analysis mistake, user confusion, or the perceived improbability of event sequence. No analysis equals no mistake.

Further, explaining errors is often more difficult than finding them. A misunderstood explanation means the error is ignored or, worse, transmuted into a false positive. The heuristic we follow: Whenever a checker calls a complicated analysis subroutine, we have to explain what that routine did to the user, and the user will then have to (correctly) manually replicate that tricky thing in his/her head.

Sophisticated analysis is not easy to explain or redo manually. Compounding the problem, users often lack a strong grasp on how compilers work. A representative user quote is "'Static' analysis'? What's the performance overhead?"

The end result? Since the analysis that suppresses false positives is invisible (it removes error messages rather than generates them) its sophistication has scaled far beyond what our research

Figure 2. No bonus.



system did. On the other hand, the commercial Coverity product, despite its improvements, lags behind the research system in some ways because it had to drop checkers or techniques that demand too much sophistication on the part of the user. As an example, for many years we gave up on checkers that flagged concurrency errors; while finding such errors was not too difficult, explaining them to many users was. (The PREfix system also avoided reporting races for similar reasons though is now supported by Coverity.)

No bug is too foolish to check for. Given enough code, developers will write almost anything you can think of. Further, completely foolish errors can be some of the most serious; it's difficult to be extravagantly nonsensical in a harmless way. We've found many errors over the years. One of the absolute best was the following in the X Window System:

```
if(getuid() != 0 && geteuid == 0) {
 ErrorF("only root");
 exit(1);
  }
```

It allowed any local user to get root access[d] and generated enormous press coverage, including a mention on Fox news (the Web site). The checker was written by Scott McPeak as a quick hack to get himself familiar with the system. It made it into the product not because of a perceived need but because there was no reason not to put it in. Fortunately.

False Positives

False positives do matter. In our experience, more than 30% easily cause problems. People ignore the tool. True bugs get lost in the false. A vicious cycle starts where low trust causes complex bugs to be labeled false positives, leading to yet lower trust. We have seen this cycle triggered even for true errors. If people don't understand an error, they label it false. And done once, induction makes the (n+1)th time easier. We initially thought false positives could be eliminated through technology. Because of this dynamic we no longer think so.

We've spent considerable technical effort to achieve low false-positive rates in our static analysis product. We aim for below 20% for "stable" checkers. When forced to choose between more bugs or fewer false positives we typically choose the latter.

Talking about "false positive rate" is simplistic since false positives are not all equal. The initial reports matter inordinately; if the first *N* reports are false positives (*N* = 3?), people tend to utter variants on "This tool sucks." Furthermore, you never want an embarrassing false positive. A stupid false positive implies the tool is stupid. ("It's not even smart enough to figure *that* out?") This technical mistake can cause social problems. An expensive tool needs someone with power within a company or organization to champion it. Such people often have at least one enemy. You don't want to provide ammunition that would embarrass the tool champion internally; a false positive that fits in a punchline is really bad.

Conclusion

While we've focused on some of the less-pleasant experiences in the commercialization of bug-finding products, two positive experiences trump them all. First, selling a static tool has become dramatically easier in recent years. There has been a seismic shift in terms of the average programmer "getting it." When you say you have a static bug-finding tool, the response is no longer "Huh?" or "Lint? Yuck." This shift seems due to static bug finders being in wider use, giving rise to nice networking effects. The person you talk to likely knows someone using such a tool, has a competitor that uses it, or has been in a company that used it.

Moreover, while seemingly vacuous tautologies have had a negative effect on technical development, a nice balancing empirical tautology holds that bug finding is worthwhile for anyone with an effective tool. If you can find code, and the checked system is big enough, and you can compile (enough of) it, then you will always find serious errors. This appears to be a law. We encourage readers to exploit it.

Acknowledgments

We thank Paul Twohey, Cristian Cadar, and especially Philip Guo for their helpful, last-minute proofreading. The experience covered here was the work of many. We thank all who helped build the tool and company to its current state, especially the sales engineers, support engineers, and services engineers who took the product into complex environments and were often the first to bear the brunt of problems. Without them there would be no company to document. We especially thank all the customers who tolerated the tool during its transition from research quality to production quality and the numerous champions whose insightful feedback helped us focus on what mattered. C

References

1. Ball, T. and Rajamani, S.K. Automatically validating temporal safety properties of interfaces. In *Proceedings of the Eighth international SPIN Workshop on Model Checking of Software* (Toronto, Ontario, Canada). M. Dwyer, Ed. Springer-Verlag, New York, 2001, 103–122.
2. Bush, W., Pincus, J., and Sielaff, D. A static analyzer for finding dynamic programming errors. *Software: Practice and Experience 30*, 7 (June 2000), 775–802.
3. Coverity static analysis; http://www.coverity.com
4. Das, M., Lerner, S., and Seigle, M. ESP: Path-sensitive program verification in polynomial time. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation* (Berlin, Germany, June 17–19). ACM Press, New York, 2002, 57–68.
5. Edison Design Group. EDG C compiler front-end; http://www.edg.com
6. Engler, D., Chelf, B., Chou, A., and Hallem, S. Checking system rules using system-specific, programmer-written compiler extensions. In *Proceedings of the Fourth Conference on Operating System Design & Implementation* (San Diego, Oct. 22–25). USENIX Association, Berkeley, CA, 2000, 1–1.
7. Flanagan, C., Leino, K.M., Lillibridge, M., Nelson, G., Saxe, J.B., and Stata, R. Extended static checking for Java. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (Berlin, Germany, June 17–19). ACM Press, New York, 2002, 234–245.
8. Foster, J.S., Terauchi, T., and Aiken, A. Flow-sensitive type qualifiers. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation* (Berlin, Germany, June 17–19). ACM Press, New York, 2002, 1–12.
9. Hallem, S., Chelf, B., Xie, Y., and Engler, D. A system and language for building system-specific, static analyses. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (Berlin, Germany, June 17–19). ACM Press, New York, 2002, 69–82.
10. Hastings, R. and Joyce, B. Purify: Fast detection of memory leaks and access errors. In *Proceedings of the Winter 1992 USENIX Conference* (Berkeley, CA, Jan. 20–24). USENIX Association, Berkeley, CA, 1992, 125–138.
11. Xie, Y. and Aiken, A. Context- and path-sensitive memory leak detection. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Lisbon, Portugal, Sept. 5–9). ACM Press, New York, 2005, 115–125.

Al Bessey, **Ken Block**, **Ben Chelf**, **Andy Chou**, **Bryan Fulton**, **Seth Hallem**, **Charles Henri-Gros**, **Asya Kamsky**, and **Scott McPeak** are current or former employees of Coverity, Inc., a software company based in San Francisco, CA.; http://www.coverity.com

Dawson Engler (engler@stanford.edu) is an associate professor in the Department of Computer Science and Electrical Engineering at Stanford University, Stanford, CA, and technical advisor to Coverity, Inc., San Francisco, CA.

d The tautological check `geteuid == 0` was intended to be `geteuid() == 0`. In its current form, it compares the address of `geteuid` to 0; given that the function exists, its address is never 0.

DOI:10.1145/1646353.1646373

The National Academy of Sciences recommends what the U.S. government should do to help maintain American IT leadership.

BY ERIC BENHAMOU, JON EISENBERG, AND RANDY H. KATZ

Assessing the Changing U.S. IT R&D Ecosystem

THE U.S. NATIONAL Academy of Sciences was established by President Abraham Lincoln in 1863 to provide unbiased assessment of scientific and technology policy issues facing the U.S. government (http://www.nas.edu). In 2006, the Academy's Computer Science and Telecommunication Board, under the sponsorship of the U.S. National Science Foundation, established a committee of experts in the fields of IT research (Randy Katz, Ed Lazowska, Raj Reddy), venture investment (Eric Benhamou, David Nagel, Arati Prabhakar), economics of innovation and globalization (Andrew Hargadon, Martin Kenney, Steven Klepper), and labor and workforce issues (Stephen Barley, Lucy Sanders) to assess the effects of changes in the U.S. IT R&D ecosystem (http://sites.nationalacademies.org/CSTB/index.htm). The committee took as its charter to examine the period from 1995 to the present, a time of rapid expansion and contraction of the field in response to the Internet boom, a precipitous stock-market collapse, increased competition through globalization of the industry, and an economy-disrupting terrorist attack and subsequent aftershocks to the economy, the full implications of which have yet to be understood.

While the committee's study focused on global developments and their implications, particularly for the U.S., the report, which was issued in February 2009, is of interest to researchers and policymakers in all of the world's IT-intensive economies, as the globalization of the industry accelerates with the rise of India and China as major centers.

Here, we summarize the report's observations, findings, and recommendations (http://www.nap.edu/catalog.php?record_id=12174), covering several main themes:

- IT's central role in the developed world;
- The committee's assessments of the IT R&D ecosystem, with inevitable U.S.-centric view, calling for increased and balanced investment in IT research by the U.S. government, with supporting investment in education and outreach to develop a high-skills/information-technology-aware work force, a commitment to reduce the "friction" the IT industry faces in the U.S. economy (such as in its highly litigious intellectual property system and regulatory regimes whose compliance unduly burdens small companies significantly more than larger ones) and expanded commitment to develop a leading-edge U.S. IT infrastructure; and
- The American Recovery and Reinvestment Act of 2009 and the actions taken by the Obama administration directly relevant to the recommendations of the committee.

IT Impact

Advances in IT and its applications represent a signal success for U.S. scientific, engineering, business, and

government over the past 50 years. IT has transformed, and continues to transform, all aspects of people's lives in the developed world, with increasing effects on the developing world, including in commerce, education, employment, health care, manufacturing, government, national security, communications, entertainment, science, and engineering. IT also drives the overall economy, both directly (the IT sector itself) and indirectly (other sectors powered by advances in IT).

To appreciate the magnitude and breadth of these effects, imagine spending a day in the developed world without IT. It would be a day without the Internet and all it enables: no diagnostic medical imaging; automobiles without electronic ignition, antilock brakes, and electronic stability control; no digital media (wireless telephones, high-definition televisions, MP3 audio, DVD video, computer animation, and videogames); aircraft unable to fly and travelers unable to navigate with benefit of the global positioning system; weather forecasters with no models; banks and merchants unable to transfer funds electronically; factory automation unable to function; and the U.S. military without technological supremacy. It would be, for most people, a "day the Earth stood still."

IT and its effect on the economy continue to grow in size and importance. According to estimates of the U.S. government's Bureau of Economic Analysis (http://www.bea.gov/), for 2006 the IT-intensive "information-communications technology producing" industries accounted for about 4% of the U.S. economy but contributed over 14% of real gross domestic product (GDP) growth. As a point of reference, U.S. federal funding in fiscal year 2008 for computer science research was around $3 billion, less than 0.025% of GDP. This substantial contribution to the economy reflects only a portion of the overall long-term benefits from IT research investment. It is in all governments' interest for these benefits to continue to grow and accrue.

Figure 1 is the "tire-tracks" diagram used to describe a large number of IT-industry sectors that have grown into billion-dollar industries through a combination of university and industrial research. Creation of these sectors

Figure 1. Examples of U.S.-government-sponsored IT R&D in the creation of commercial products and industries, the so-called tire-tracks diagram.

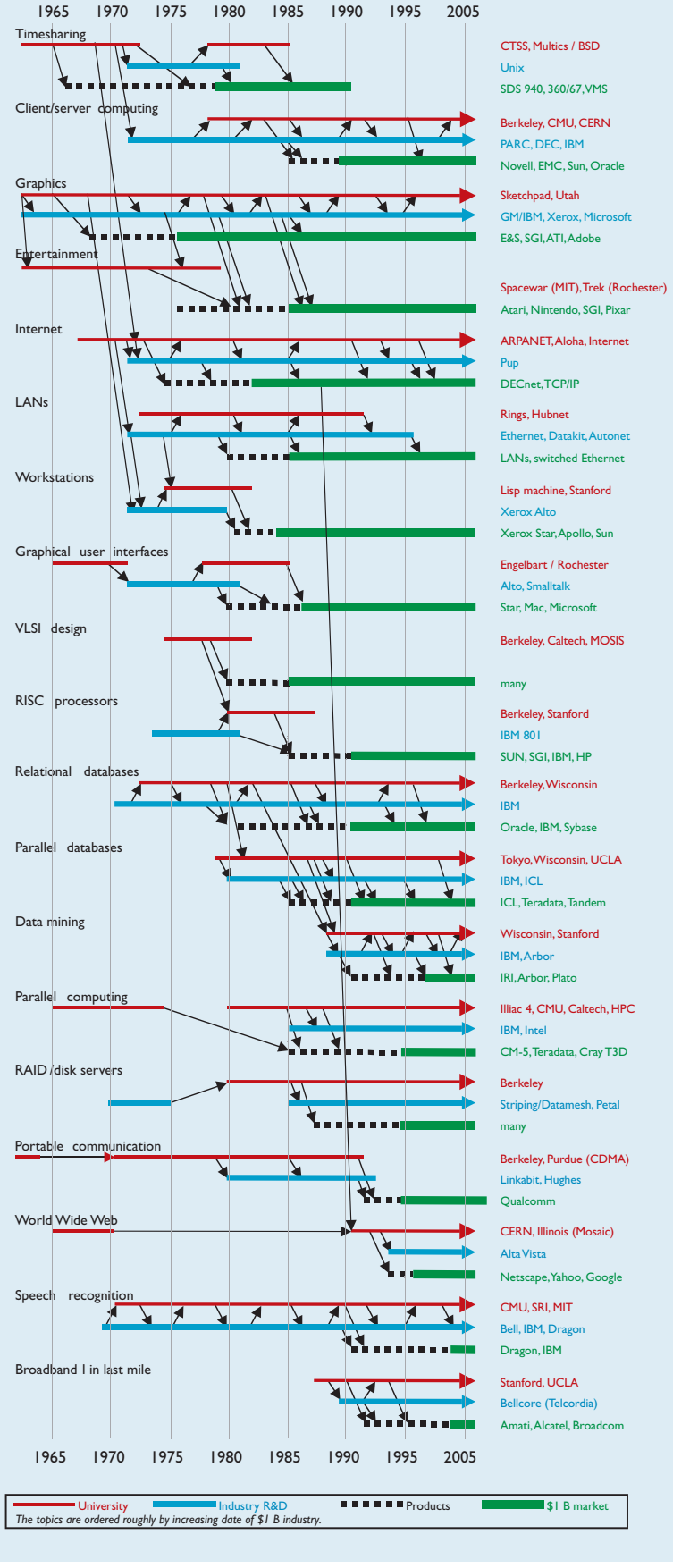
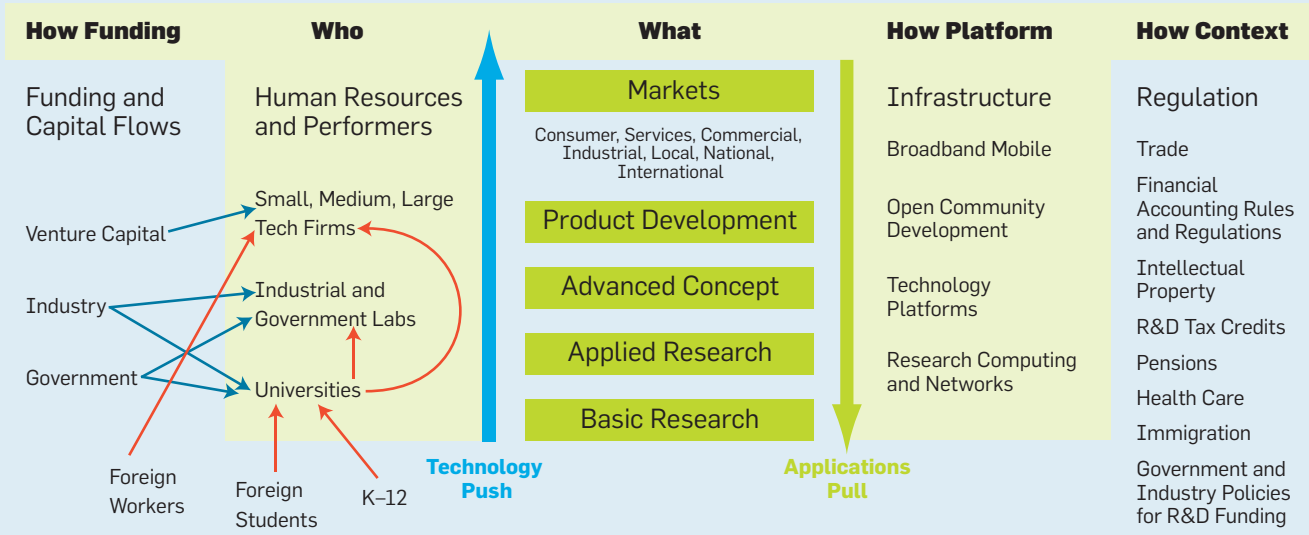


Figure 2. Key elements and relationships in the U.S. IT R&D ecosystem.



has been achieved, not just from the government investment in foundational research, but from a vibrant venture community able and willing to provide the risk capital to fund the commercialization of new ideas from which such industrial sectors are able to grow.

Assessing the Ecosystem

The U.S. IT R&D ecosystem was the envy of the world in 1995; Figure 2 outlines its essential elements: university and industrial research enterprises; emerging start-up companies and more mature technology companies; the industry that finances innovative firms; and the regulatory environment and legal frameworks. From the perspective of IT, the U.S. enjoyed a strong industrial base, the ability to create and leverage new technological advances, and an extraordinary system for creating world-class technology companies.

The period from 1995 to the present has been a turbulent one for the U.S. and the world, as characterized by:

- Irrational exuberance for IT stocks and the NASDAQ bust (2000);
- Y2K and the development of the Indian software industry;
- Aftereffects of the terror attacks of September 11, 2001;
- Financial scandals and bankruptcies (2001);
- Surviving after the bubble burst (2001–2004);
- Recovery (2005–2007); and
- Global economic financial crisis (2008).

These shocks took their toll, and in the view of the committee, government actions are necessary to sustain the U.S. IT R&D ecosystem. The U.S. government should:

- Strengthen the effectiveness of government-funded IT research;
- Remain the strongest generator of and magnet for technical talent;
- Reduce the friction that harms the effectiveness of the U.S. IT R&D ecosystem while maintaining other important political and economic objectives; and
- Ensure that the U.S. has an infrastructure for communications, computing, applications, and services that enables U.S. IT users and innovators to lead the world.

Potential high-value societal benefits of continued investment in IT include:

- Safer, robotics-enhanced automobiles;
- A more scalable, manageable, secure, robust Internet;
- Personalized and collaborative educational tools for tutoring and just-in-time learning;
- Personalized health monitoring;
- Augmented cognition to help people cope with information overload; and
- IT-driven advances in all fields of science and engineering.

Much has been learned from decades of experience about what constitutes an environment that fosters successful research and its transition to commercial formation. The following insights are extracted from a 2003 Academy report from the National Research Council called *Innovation in Information Technology* (The National Academies Press, Washington, D.C., 2003 http://www.nap.edu/openbook.php?isbn=0309089808):

On the results of research:

- U.S. international leadership in IT (vital to the country) springs from a deep tradition of research;
- The unanticipated results of research are often as important as the anticipated results; and
- The interaction of research ideas multiplies their effect; for example, concurrent research programs targeting integrated circuit design, computer graphics, networking, and workstation-based computing strongly reinforce and amplify one another.

On research as a partnership:

- The success of the IT research enterprise reflects a complex partnership among government, industry, and universities;
- The federal government has had and will continue to have an essential role in sponsoring fundamental research in IT (largely university-based) because it does what industry cannot do. Industrial and governmental in-

vestment in research reflects different motivations, resulting in differences in style, focus, and time horizon;

▸ Companies have little incentive to invest significant amounts in activities whose benefits spread quickly to their rivals. Fundamental research often falls into this category. By contrast, the vast majority of corporate R&D addresses product-and-process development; and

▸ Government funding for research has leveraged the effective decision making of visionary program managers and program-office directors from the research community, empowering them to take risks in designing programs and selecting grantees. Government sponsorship of research, especially in universities, also helps develop the IT talent used by industry, universities, and other parts of the economy.

On the economic payoff of research:

▸ Past returns on federal investment in IT research have been extraordinary for both U.S. society and the U.S. economy. The transformative effects of IT grow as innovations build on one another and as user know-how compounds. Priming that pump for tomorrow is today's challenge; and

▸ When companies create products using the ideas and work force that result from federally sponsored research, they repay the nation in jobs, tax revenue, productivity increases, and world leadership.

Inevitable Globalization of IT

Another significant trend, analyzed in detail in the report, is how the IT industry has become more globalized, especially with the dramatic rise of the economies of India and China, fueled in no small part by their development of vibrant IT industries. Moreover, India and China represent fast-growing markets for IT products, with both likely to grow their IT industries into economic powerhouses for the world, reflecting both deliberate government policies and the existence of strong, vibrant private-sector firms, both domestic and foreign. Ireland, Israel, Japan, Korea, and Taiwan, as well as some Scandinavian countries, have also developed strong niches within the increasingly globalized industry. Today, a product conceptualized and marketed in the U.S. might be designed to specifications in Taiwan, and batteries or hard drives obtained from Japan might become parts in a product assembled in China. High-value software and integrated circuits at the heart of a product might be designed and developed in the U.S., fabricated in Taiwan, and incorporated into a product assembled from components supplied from around the world.

Unfortunately, during a period of rapid globalization, U.S. national policies have not sufficiently buttressed the ecosystem or generated side effects that have reduced its effectiveness. This is particularly true of such areas as IT education, U.S. government IT research funding, and the regulations that affect the corporate overhead and competitiveness of innovative IT companies. As a result, the U.S. position in IT leadership today has eroded compared to that of prior decades, and IT leadership may pass to other nations within a generation unless the U.S. recommits itself to providing the resources needed to fuel U.S. IT innovation, removing important roadblocks that reduce the ecosystem's effectiveness in generating innovation and the fruits of innovation, and becoming a lead innovator and user of IT.

In 2009, the ecosystem also faced new challenges from a global economic crisis that continues to unfold. There has been a marked reduction in the availability of venture capital following losses in pension funds and endowments, as well as in initial public offerings by technology companies and a decline in mergers and acquisitions. There is also a steep decline in consumer confidence, suggesting that a consumer-driven recovery is unlikely in the near term. Significant layoffs and hiring cutbacks in IT firms and across the global economy seem all but certain to adversely affect the IT R&D ecosystem, undermining the partial recovery seen over the past few years. The magnitude, duration, and enduring effects on the ecosystem of the downturn are not yet clear.

Globalization is a broad and sweeping phenomenon that cannot be contained. If embraced rather than resisted, the committee concluded that it presents more opportunity than threat to the U.S. national IT R&D ecosystem. To thrive in this landscape, the U.S. should play to its strengths, notably its continued leadership in conceptualizing idea-intensive new concepts, products, and services the rest of the world desires and where the greatest increments of value-added are captured.

Toward this end, it is necessary for the U.S. to have the best-funded and most-creative research institutions; develop and attract the best technical and entrepreneurial talent among its own people, as well as those from around the world; make its economy the world's most attractive for forming new ventures and nurturing small, in-

PHOTOGRAPH COURTESY OF THE NATIONAL CENTER FOR COMPUTATIONAL SCIENCES

Ultrascale scientific computing capability, like the world's fastest supercomputer—Jaguar—in the U.S. Department of Energy's Oak Ridge National Laboratory, is considered a top priority among government science funding.

novative firms; and create an environment that ensures deployment of the most advanced technology infrastructures, applications, and services in the U.S. itself for the benefit of the nation's people, institutions, and firms.

Findings and Recommendations

Here, we describe the report's findings and recommendations in the context of the four objectives for government action described earlier:

Objective 1. Strengthen the effectiveness of federally funded IT research. University research is focused largely on basic research, while industrial research concentrates on applied R&D, meaning that much of the feedstock for long-term innovation is to be found in the nation's universities. As a result, support for university education and research is essential to generating the stream of innovations that nourish the rest of the ecosystem. Measures to enhance the productivity of university research funding, as well as that of other R&D funding, would increase the payoff from these investments.

Although the advances in IT over the past 50 years have been breathtaking, the field remains in its relative infancy, and continuing advances over the coming decades can be expected but only as long as the IT R&D ecosystem's capacity to sustain innovation is preserved and enhanced.

Current decisions about how the U.S. should make investments—both civilian and military—in basic IT research do not seem to reflect the full effect of IT on society and the economy. The government's own data indicates the U.S. lags behind Europe and Japan in civilian funding for IT R&D. Meanwhile, the European Union and China have aggressive plans for strengthening their global positions in IT through substantial and increasing IT R&D investment.

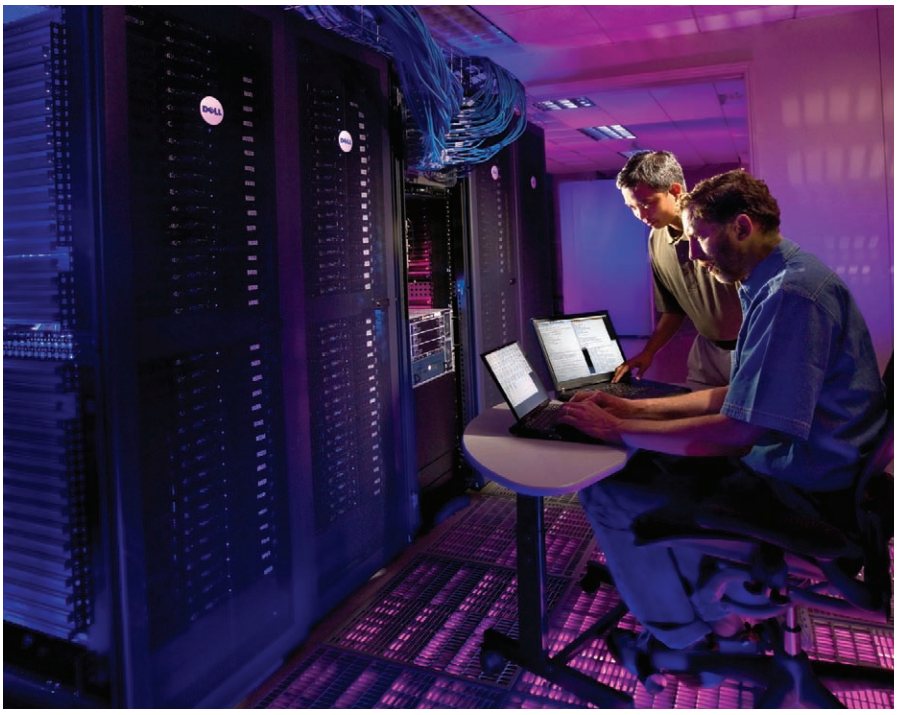
Regaining a leading position requires aggressive action, including ambitious targets for increased R&D investment. It is appropriate and necessary for the U.S. to correspondingly adjust its own IT R&D spending level, just as individual businesses, following best practices, track their global competitors' business models to avoid falling behind in global market share. Increased federal investment in IT research would reflect the importance of IT to the nation's society and economy as a whole and would allow the U.S. to build and sustain the already large positive effect of IT on its economy. The desirability of increased federal investment in IT R&D was also recognized in a 2007 report by the National Academies, *Rising Above the Gathering Storm: Energizing and Employing America for a Brighter Economic Future* (http://www.nap.edu/catalog.php?record_id=11463), and, to some extent, by provisions in the subsequently passed the America Competes Act of 2007 (http://thomas.loc.gov/cgi-bin/bdquery/z?d110:SN00761:@@@D&summ2=m&). Moreover, in its August 2007 report, the President's Council of Advisors on Science and Technology (PCAST, http://www.ostp.gov/pdf/nitrd_review.pdf) found an imbalance in the current federal R&D portfolio in that more long-term, large-scale, multidisciplinary R&D is needed. PCAST concluded that current interagency coordination processes for networking and IT R&D are inadequate for meeting anticipated national needs and for maintaining U.S. leadership in an era of global competitiveness.

Imagine spending a day in the developed world without IT.

A strategic reassessment of U.S. R&D priorities is needed, an analysis meriting the attention of first-tier scientists and engineers from academia, industry, and government. A strong focus on IT is important due to the special role of IT within science and engineering.

Toward this end, a means of delivering to the highest levels of the U.S. government the best possible advice on the transformational power of IT would help ensure that the nation invests at appropriate levels in IT research and this investment is made as efficiently and as effectively as possible, in part through improved coordination of federal R&D investments. This advice could be provided in a number of ways, including augmentation of the current presidential science-and-technology advisory structure, establishment of a high-level IT adviser to the President of the U.S., or reestablishment of an IT-specific presidential advisory committee (such as the President's Information Technology Advisory Committee, which operated 1997–2005).

Finding. A robust program of U.S. government-sponsored R&D in IT is vital to the nation.



Computer scientists at Sandia National Laboratories successfully demonstrated the ability to run more than a million Linux kernels as virtual machines. The Sandia research, two years in the making, was funded by the Department of Energy's Office of Science, the National Nuclear Security Administration's Advanced Simulation and Computing program, and by internal Sandia funding.

Finding. The level of U.S. government investment in fundamental research in IT continues to be inadequate.

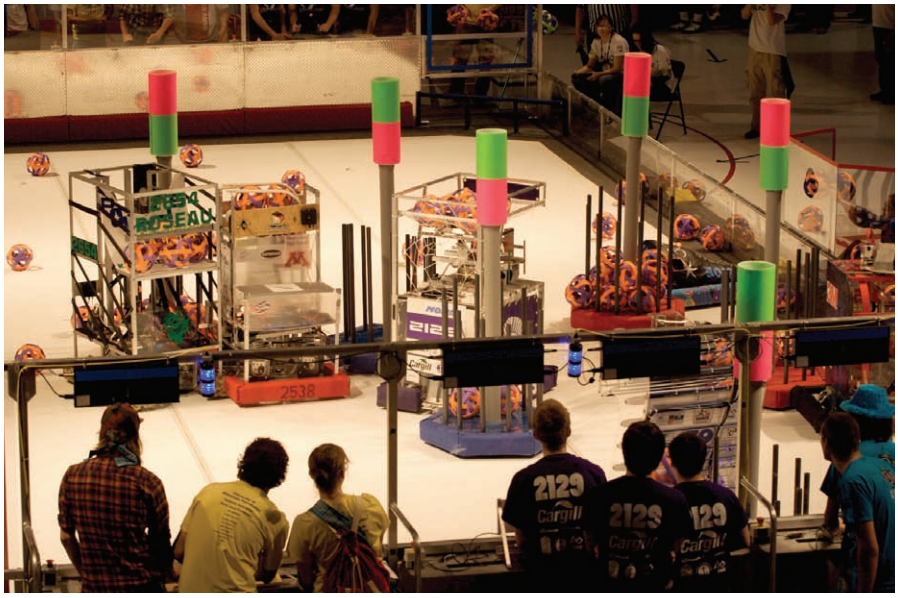
Recommendation. As the U.S. government increases its investment in long-term basic research in the physical sciences, engineering, mathematics, and information sciences, it should carefully assess the level of investment in IT R&D, mindful of economic return, societal effect, enablement of discovery across science and engineering, and other benefits of additional effort in IT and should ensure that appropriate advisory mechanisms are in place to guide investment within the IT R&D portfolio.

Objective 2. Remain the strongest generator of and magnet for technical talent. There is cause for concern that an undersized, insufficiently prepared work force for the IT industry will accelerate migration of higher-value activities out of the U.S. While the committee did not address the entire array of technology-sector wage-and-job-security issues, it is clear that without a work force that is knowledgeable with respect to technology and that has sufficient numbers of highly trained workers, the U.S. will find it difficult to retain the most innovation-driven parts of its own IT industry. Despite demand for such workers, the number of students specifying their intention to major in computing and information sciences has dropped significantly since 2003. The problem of declining enrollment in the computing disciplines (compared to projected demand) is compounded by the very low participation of underrepresented groups in IT.

The U.S. should rebuild its national IT educational pipeline, encouraging all qualified students, regardless of race, gender, or ethnicity, to enter the discipline. Without sustained, amplified intervention, the U.S. is unlikely to have an educational pipeline capable of yielding a revived and diverse IT work force over the next 10 years. To achieve the needed revitalization, the U.S. should pursue a multipronged approach: improve technology education at all levels from kindergarten through grade 12; broaden participation in IT careers by women, people with disabilities, and all minorities, particularly African-Americans, Hispanics, and Native Americans; and retain foreign students who have received advanced degrees in IT. Immigrants have been especially significant in high-technology entrepreneurship; for at least 25% of the U.S. engineering and technology companies started between 1995 and 2005, mostly in software, innovation, and manufacturing-related services, at least one key founder was born outside the U.S.

Finding. Rebuilding the computing-education pipeline at all levels requires overcoming numerous obstacles, in turn portending significant challenges for the development of future U.S. IT work-force talent.

Finding. Participation in IT of wom-



Robotics technologies—for defense, education, and manufacturing among many other applications—are always strong contenders for government funding. The FIRST Robotics Competition, here the Northstar Regional at the University of Minnesota, Minneapolis, challenged teams of young people and their mentors to solve a common problem using a standard "kit of parts."

TOP: PHOTOGRAPH COURTESY OF SANDIA NATIONAL LABORATORIES. BOTTOM: PHOTOGRAPH BY NIC MCPHEE

en, people with disabilities, and all minorities is especially low and declining. This low level of participation will affect the ability of the U.S. to meet its work-force needs and place it at a competitive disadvantage by not allowing it to capitalize on the innovative thinking of half its population.

Recommendation. To build such a skilled work force it needs to retain high-value IT industries, the U.S. should invest more in education and outreach initiatives to nurture and increase its IT talent pool.

Finding. Although some IT professional jobs will be offshored, at the time the committee completed its report in 2008, it found there were more IT jobs in the U.S. than at any time during the dotcom boom, even in the face of corporate offshoring trends. While this may no longer be true in the wake of the global recession, anecdotal evidence indicates strong demand for new graduates in computer science programs nationally.

Recommendation. The U.S. should increase the availability and facilitate the issuance of work and residency visas to foreign students who graduate with advanced IT degrees from U.S. educational institutions.

Objective 3. Reduce friction that harms the effectiveness of the U.S. IT R&D ecosystem. Such factors as intellectual property litigation and corporate governance regulations have become sources of increased friction in the conduct of business in the U.S. and can have the effect of making other countries more attractive places to establish the small, innovative companies that are essential components of the ecosystem. These issues are not simple; for example, in terms of corporate governance, the dampening effects of increased regulation must be weighed against the benefits of restoring and maintaining public confidence in equity markets. But to keep the U.S. attractive for new venture formation and to sustain the nation's unrivaled ability to transform innovative new concepts into category-defining products and services the world desires, the potential effects on the IT R&D ecosystem should be weighed in considering new measures or reforms in such areas as corporate governance and intellectual property litigation.

Finding. Fewer young, innovative IT companies are gaining access to U.S. public equity markets.

Recommendation. Congress and federal agencies (such as the Securities and Exchange Commission http://www.sec.gov/ and the Patent and Trademark Office http://www.uspto.gov/) should consider the effect of both current and proposed policies and regulations on the IT ecosystem, especially on young, innovative IT businesses, and consider measures to mitigate them where appropriate.

Phillip J. Bond, president of leading U.S. technology trade association TechAmerica, testified before the Senate Homeland Security and Government Affairs Subcommittee last April on advancing U.S. efforts toward the digital future.

PHOTOGRAPH COURTESY OF TECHAMERICA

Objective 4. Ensure that the U.S. has the infrastructure that enables U.S. IT users and innovators to lead the world. The U.S. has long enjoyed the position of being the largest market for IT; global demographics and relative growth rates suggest this advantage is unlikely to endure. Fortunately, although a healthy domestic IT market is an important element of a healthy domestic ecosystem, market size is not the only factor in leadership. The environment fostered by leading-edge users of technology (including those who can leverage research, innovate, and create additional value) creates the context for IT's next wave and its effective application. In such an environment, all sectors of society (including consumers, businesses, and governments) exploit and make the best use of advanced IT. But there are indications that the U.S. has indeed lost its leadership in the use of IT.

In particular, the U.S. broadband infrastructure is not as advanced or as widely deployed as that in many other countries. Should this situation persist, the U.S. will no longer be the nation in which the most innovative, most advanced technology and highest value-added products and services are conceptualized and developed.

Moreover, in addition to broadly fostering research and commercial innovation, government-sponsored R&D can help meet particular government demands. Though the government is no longer a lead IT user across the board, it continues to play an appropriate leadership role where federal-agency requirements are particular to their missions and commercial analogs are scarce or nonexistent.

Finding. The most dynamic IT sector is likely to be in the country with the most demanding IT customers and consumers.

Finding. In terms of nationwide availability, use, and speed of broadband, the U.S. (a former leader in the technology) has been losing ground compared with other nations.

Recommendation. The U.S. should establish an ambitious target for regaining and holding a decisive lead in the broad deployment of affordable gigabit-broadband services. Federal and state regulators should explore models and approaches that reduce regulatory and jurisdictional bottle-

necks and increase incentives for investment in these services.

Recommendation. Government (federal, state, local) should foster commercial innovation and itself make strategic investments in IT R&D and deployment so the U.S. retains a global lead position in areas where it has particular mission requirements.

American Recovery and Reinvestment Act of 2009

The recent global financial crisis has had a major effect on the U.S. IT industry, which appeared to be recovering following the shocks of the early part of the decade; 2008 was the first year in almost two decades when there were no IT initial public offerings on U.S. stock exchanges. Even high-flying Internet companies like Google have sustained layoffs, though modest. The effect is not limited to the U.S.; the slowdown is evident throughout the globalized IT industry.

The American Recovery and Reinvestment Act of 2009 (http://www.irs.gov/newsroom/article/0,,id=204335,00.html) provides significant funding for infrastructure and research investment as part of the U.S. government's recent economic stimulus package. The Act provides $2.5 billion for distance learning, telemedicine, and broadband infrastructure for rural communities. An additional $4.7 billion is available for broadband infrastructure projects, including for expanding public access to computers, such as through community colleges and public libraries. And an additional $4.5 billion is available to upgrade the electric grid for enhanced electricity delivery and energy reliability, and will likely make extensive use of IT in the process. The Department of Energy recently established the Advanced Research Projects Agency-Energy (http://arpa-e.energy.gov/), with initial funding of $400 million; initial science and technology research awards were announced in February 2009. Another $2 billion is available to coordinate deployment of advanced health IT. Finally, the National Science Foundation's annual budget is being enhanced by $2.5 billion, essentially increasing its annual budget by 50%. This provides a much-needed increment in foundational research funding in IT. While this is intended as a one-time increase in funding, we are optimistic that the Obama administration's and Congress's commitment to science and technology will continue.

The U.S. should play to its strengths, notably its continued leadership in conceptualizing idea-intensive new concepts, products, and services the rest of the world desires.

Acknowledgments

We thank the entire membership and staff of the Committee on Assessing the Impacts of Changes in the Information Technology Research and Development Ecosystem for their contributions to this report.

Members: Eric Benhamou (Co-Chair), Randy H. Katz (Co-Chair), Stephen R. Barley, Andrew B. Hargadon, Martin Kenney, Steven Klepper, Edward D. Lazowska, Lenny Mendoca, David C. Nagel, Arati Prabhakar, Raj Reddy, and Lucinda Sanders.

Staff: Jon Eisenberg, Joan D. Winston, Kristen Batch, and Morgan R. Motto.

Further Reading

Each of the following National Academies reports is available online and includes extensive bibliographies for further investigation of the issues discussed here.

Assessing the Impacts of Changes in the Information Technology R&D Ecosystem: Retaining Leadership in an Increasingly Global Environment. National Academies Press, Washington, D.C., 2009; http://www.nap.edu/catalog.php?record_id=12174

Computing the Future: A Broader Agenda for Computer Science and Engineering. National Academies Press, Washington, D.C., 1992; http://www.nap.edu/catalog.php?record_id=1982

Funding a Revolution: Government Support for Computing Research. National Academies Press, Washington, D.C., 1999; http://www.nap.edu/catalog.php?record_id=6323

Rising Above the Gathering Storm: Energizing and Employing America for a Brighter Economic Future. National Academies Press, Washington, D.C., 2008; http://www.nap.edu/catalog.php?record_id=11463

Eric Benhamou (eric@benhamouglobalventures.com) is chairman and CEO, Benhamou Global Ventures, LLC.

Jon Eisenberg (eisenbe@nas.edu) is Director of the Computer Science and Telecommunications Board of the National Academies.

Randy H. Katz (randy@cs.berkeley.edu) is The United Microelectronics Corporation Distinguished Professor in the Electrical Engineering and Computer Science Department at the University of California, Berkeley.

DOI:10.1145/1646353.1646375

What quantum algorithms outperform classical computation and how do they do it?

BY DAVE BACON AND WIM VAN DAM

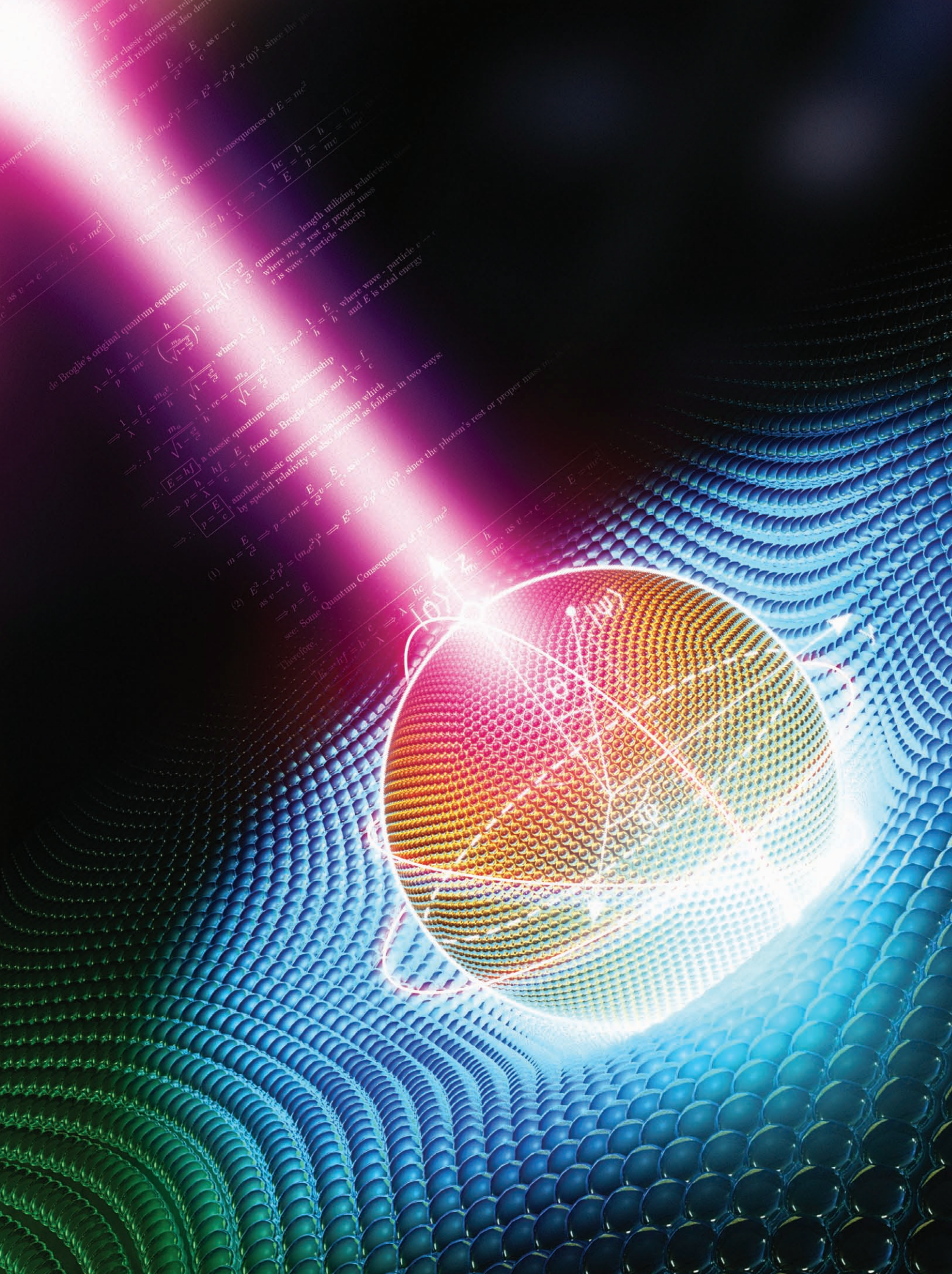
Recent Progress in Quantum Algorithms

IT IS IMPOSSIBLE to imagine today's technological world without algorithms: sorting, searching, calculating, and simulating are being used everywhere to make our everyday lives better. But what are the benefits of the more philosophical endeavor of studying the notion of an algorithm through the perspective of the physical laws of the universe? This simple idea, that we desire an understanding of the algorithm based upon physics seems, upon first reflection, to be nothing more than mere plumbing in the basement of computer science. That is, until one realizes that the pipes of the universe do not seem to behave like the standard components out of which we build a computer, but instead obey the counterintuitive laws of quantum theory. And, even more astoundingly, when one puts these quantum parts together, one gets a notion of the algorithm—the quantum algorithm—whose computational power appears to be fundamentally more efficient at carrying out certain tasks than algorithms written for today's, nonquantum, computers. Could this possibly be true: that there is a more fundamental notion of algorithmic efficiency for computers built from quantum components? And, if this is true, what exactly is the power of these quantum algorithms?

The shot that rang round the computational world announcing the arrival of the quantum algorithm was the 1994 discovery by Peter Shor that quantum computers could efficiently factor natural numbers and compute discrete logarithms.[24] The problem of finding efficient algorithms for factoring has been burning the brains of mathematicians at least as far back as Gauss who commented upon the problem that "the dignity of science seems to demand that every aid to the solution of such an elegant and celebrated problem be zealously cultivated." Even more important than the fact that such a simple and central problem has eluded an efficient algorithmic solution is that the lack of such an efficient algorithm has been used as a justification for the security of public key cryptosystems, like RSA encryption.[23] Shor's algorithm, then, didn't just solve a problem of pure academic interest, but instead ended up showing how quantum computers could break the vast majority of cryptographic protocols in widespread use today. If we want the content of our publicly key encrypted messages to remain secret not only now, but also in the future, then Shor's algorithm redefines the scope of our confidence in computer security: we communicate securely, today, given that we cannot build a large scale quantum computer tomorrow.

Given the encryption breaking powers promised by quantum computers, it was natural that, in the decade following Shor's discovery, research has focused largely on whether a

ILLUSTRATION BY MONDOLITHIC STUDIOS



quantum computer could be built. While there currently appear to be no fundamental obstacles toward building a large scale quantum computer (and even more importantly, a result known as the "threshold theorem"[1, 16–18, 25] shows that quantum computers can be made resilient against small amounts of noise, thereby confirming that these are not analog machines), the engineering challenges posed to build an RSA breaking quantum computer are severe and the largest quantum computers built to date have less than 10 quantum bits (qubits).[13, 19] But regardless of the progress in building a quantum computer, if we are to seriously consider our understanding of computation as being based upon experimental evidence, we will have to investigate the power of quantum algorithms. Christos Papadimitriou said in a recent interview[26] that the theory of computational complexity is such a difficult field because it is nearly impossible to prove what everyone knows from experience. How, then, can we even begin to gain an understanding of the power of quantum computers if we don't have one from which to gain such an experience? Further, and perhaps even more challenging, quantum algorithms seem to be exploiting the very effects that make quantum theory so uniquely counterintuitive.[6] Designing algorithms for a quantum computer is like building a car without having a road or gas to take it for a test drive.

In spite of these difficulties, a group of intrepid multidisciplinary researchers have been tackling the question of the power of quantum algorithms in the decades since Shor's discoveries. Here we review recent progress on the upper bounding side of this problem: what new quantum algorithms have been discovered that outperform classical algorithms and what can we learn from these discoveries? Indeed, while Shor's factoring algorithm is a tough act to follow, significant progress in quantum algorithms has been achieved. We concentrate on reviewing the more recent progress on this problem, skipping the discussion of early (but still important) quantum algorithms such as Grover's algorithm[12] for searching (a quantum algorithm that can search an unstructured space quadratically faster than the best classical algorithm), but explaining some older algorithms in order to set context. For a good reference for learning about such early, now "classic" algorithms (like Grover's algorithm and Shor's algorithm) we refer the reader to the textbook by Nielsen and Chuang.[21] Our discussion is largely ahistoric and motivated by attempting to give the reader intuition as to what motivated these new quantum algorithms. Astonishingly, we will see that progress in quantum algorithms has brought into the algorithmic fold basic ideas that have long been foundational in physics: interference, scattering, and group representation theory. Today's quantum algorithm designers plunder ideas from physics, mathematics, and chemistry, weld them with the tried and true methods of classical computer science, in order to build a new generation of quantum contraptions which can outperform their classical counterparts.

Quantum Theory in a Nutshell

Quantum theory has acquired a reputation as an impenetrable theory accessible only after acquiring a significant theoretical physics background. One of the lessons of quantum computing is that this is not necessarily true: quantum computing can be learned without mastering vast amounts of physics, but instead by learning a few simple differences between quantum and classical information. Before discussing quantum algorithms we first give a brief overview of why this is true and point out the distinguishing features that separate quantum information from classical information.

To describe a deterministic n-bit system it is sufficient to write down its configuration, which is simply a binary string of length n. If, however, we have n-bits that can change according to probabilistic rules (we allow randomness into how we manipulate these bits), we will instead have to specify the probability distribution of the n-bits. This means to *specify* the system we require 2^n positive real numbers describing the probability of the system being in a given configuration. These 2^n numbers must sum to unity since they are, after all, probabilities. When we observe a classical system, we will always find it to exist in one particular configuration (i.e. one particular binary string) with the probability given by the 2^n numbers in our probability distribution.

Now let's turn this approach to quantum systems, and consider a system made up of n qubits. Again, n qubits will have a configuration which is just a length n binary string. When you observe n qubits you will only see an n bit configuration (thus when you hear someone say that a qubit is both zero and one at the same time you can rely on your common sense tell them that this is absurd). But now, instead of describing our system by 2^n probabilities, we describe a quantum system by 2^n *amplitudes*. Amplitudes, unlike probabilities (which were positive real numbers and which summed to unity), are complex numbers which, when you take their absolute value-squared and add them up, sum to unity. Given the 2^n amplitudes describing a quantum system, if you observe the system, you will see a particular configuration with a probability given by the modulus squared of the amplitude for that configuration. In other words, quantum systems are described by a set of 2^n complex numbers that are a bit like square roots of probabilities (see Figure 1).

So far we have just said that there is this different description for quantum systems, you describe them by amplitudes and not by probabilities. But does this really have a consequence? After all the amplitudes aren't used so far, except to calculate probabilities. In order to see that yes, indeed, it does

Figure 1. Classical versus quantum information.

On the left, the classical bit is described by two nonnegative real numbers for its probabilities Pr(0) = 1/3 and Pr(1) = 2/3. The quantum bit on the right, instead, has two complex valued amplitudes that give the (same) probabilities by taking the absolute value-squared of its entries. When a quantum system has such a description with nonzero amplitudes, one says that the system is in a superposition of the 0 and 1 configurations.

Classical bit: $\begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix}$ Quantum bit: $\begin{bmatrix} \frac{-1}{\sqrt{3}} \\ \frac{1+i}{\sqrt{3}} \end{bmatrix}$

have a profound consequence, we must next describe how to update our description of a system as it changes in time. One can think about this as analyzing an *algorithm* where information in our computing device changes with time according to a set of specific recipe of changes.

For a classical probabilistic computing device we can describe how it changes in time by describing the conditional probability that the system changed into a new configuration given that it was in an old configuration. Such a set of conditional probabilities means that we can describe a probabilistic computing action by a stochastic matrix (a matrix whose entries are positive and whose columns sum to unity). A classical probabilistic algorithm can then be viewed as just a set of stochastic matrices describing how probabilities propagate through the computing device. If the classical probabilistic algorithm starts with n bits and ends with m bits, then the stochastic matrix describing the algorithm will be a 2^m by 2^n matrix.

What is the analogous procedure for a quantum system? Well instead of specifying conditional probabilities of a new configuration given an old configuration, in a quantum system you need to specify the conditional amplitude of a new configuration given an old configuration. In the quantum world, the matrix of conditional amplitudes has two major differences from the classical probabilistic setting. The first is that quantum systems evolve reversibly and thus the matrix is 2^n by 2^n (corresponding to the amplitude of every configuration to change into any other configuration). The second is that, in order to preserve the sum of the squares of those amplitudes, which should be 1 throughout, this matrix is a unitary matrix, meaning the entries of the matrix are complex numbers, and that the rows (and columns) of this matrix are orthonormal. Thus a quantum algorithm for a quantum system is given by a unitary matrix of conditional amplitudes.

What consequence does this change from probabilities to amplitudes and from stochastic matrices to unitary matrices have for the notion of an algorithm? This is, of course, the essential question at hand when considering quantum algorithms. In this survey we single out three major differences—quantum interference, the deep relationship between symmetries and quantum mechanics, and quantum entanglement—and show how they are related to recent progress in quantum algorithms.

Interference and the Quantum Drunkard's Walk

The first of our claimed differences between quantum computers and classical computers was that the former led to effects of quantum interference. What is interference and how can it lead to new efficient algorithms?

To illustrate the ideas of interference, consider a random walk on a line. The standard, classical drunkard's walk on a line refers to situation where the walker is allowed to step either forward or backward with equal probability every unit time step. When starting at position 0 at time zero, then after one time step there is an equal probability to be at locations +1 and –1. After the next time step, there is a one-fourth probability of being at positions –2 and 2 and one half probability of being at position 0. Notice here that the probability of reaching zero was the sum of two probabilities: the probability that the drunkard got to 0 via 1 and the probability that it got to 0 via –1. Random walks on structures more complicated than a line are a well-known tool in classical algorithms.

Suppose that we want to construct a quantum version of this drunkard's walk. To specify a quantum walk, we need, instead of a probability for taking a step forward or backward, an amplitude for doing this. However we also need to make sure that the unitary nature of quantum theory is respected. For example, you might think that the quantum analogy of a classical walk is to take a step forward and a step backward with amplitude one over the square root of two (since squaring this gives a probability of one half). If we start at 0, then after one step this prescription works: we have equal amplitude of one over square root of two of being at either 1 or –1. If we measure the walker after this first step, the probability of being at 1 or –1 is both one half. But if we run this for another time step, we see that we have an amplitude of ½ to be at –2 or 2 and an amplitude 1 to be at 0. Unfortunately if we square these numbers and add them up, we get a number greater than unity, indicating that the evolution we have described is not unitary.

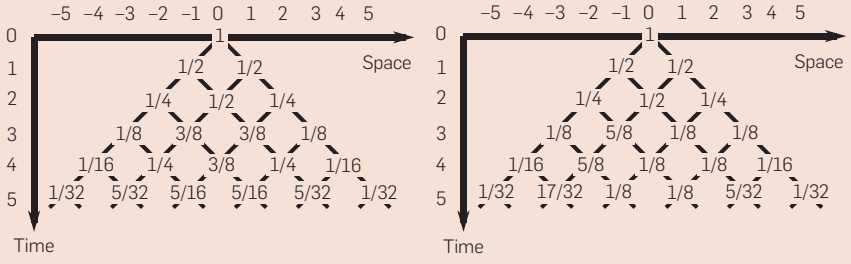
The solution to this problem is to let the drunkard flip a quantum coin at each time step, after which he steps in the direction indicated by the quantum coin. What is a quantum coin? A quantum coin is simply a qubit whose two configurations we can call "forward" and "backward" indicating the direction we are supposed to move after flipping the quantum coin. How do we flip such a coin? We apply a unitary transform. This unitary transform must specify four amplitudes. One choice of such a unitary transform that seems to mimic the drunkard's walk is to assign all conditional amplitudes a value of one over the square root of two, with the exception of the amplitude to change from the configuration "forward" to the configuration "backward," which, due to unitarity, we assign the amplitude negative one over square root of two. In other words the unitary transform we apply to flip the coin is specified by the transition matrix

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}. \qquad (1)$$

If we follow this prescription for a quantum random walk with the drunkard initially positioned at zero, one quickly sees that something strange happens. Consider, for instance, the probability distribution formed by the quantum walk had we measured the walker's position after three time steps (see Figure 2). Then the probability of getting to +1 for the drunkard is ⅛. For a classical walk the similar number would be ⅜. What is going on here? Well if you trace back how he could have gotten to +1 in three steps, you'll see that there are three paths it could have used to get to this position. In the classical world each of these is traversed with equal probability, adding a contribution of ⅛ for each step.

Figure 2. Classical (top) and quantum (bottom) random walks.

The probability of reaching a particular point in space and time, given that we measure the position at that time, is listed on the vertices.



But in the quantum world, two of these paths contribute equal but oppositely to the *amplitude* to get to this position. In other words these two paths interfere with each other. Because amplitudes, unlike probabilities, don't have to be positive numbers, they can add up in ways that cancel out. This is the effect known as quantum interference. It is the same interference idea which you see when two water waves collide with each other. But note an important difference here: amplitudes squared are probabilities. In water waves, the heights interfere, not anything related to the probabilities of the waves. This is the peculiar effect of quantum interference.

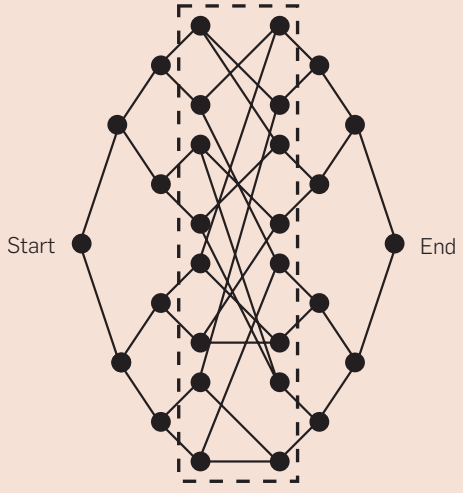
Quantum random walks were actually first described by physicists in 1993,[2] but only with the rise of interest in quantum computers was it asked whether these walks could be used as a computational tool. An alternative, continuous time version of these algorithms (tacking more closely to ideas in physics) has also been developed by Farhi and Gutmann.[9] Given these quantum random walks, a natural question is what does this have to do with algorithms? Well, the first observation is that quantum random walks behave in strange ways. For instance a well-known property of classical random walks on a line is that the expected standard deviation of a random walk as a function of the number of steps taken, T, scales like the square root of T. However, for a quantum random walk the standard deviation can actually spread linearly with T. Remarkably, this difference has been well known to physicists for a long time: it turns out that the quantum random walk defined above is closely related to the Dirac equation for a one-dimensional electron (the Dirac equation is a way to get quantum mechanics to play nicely with the special theory of relativity, and is a basic equation used in modern quantum field theory). This discovery that quantum algorithms seem to explore space quadratically faster than classical random walks has recently been shown to lead to quantum algorithms that polynomially outperform their classical cousins.

One example of an algorithm based upon quantum random walks is the algorithm for element distinctness due to Ambainis.[3] The element distinctness problem is, given a function f from $\{1, 2, ..., N\}$ to $\{1, 2, ..., N\}$ determine whether there exists two indices $i \neq j$ such that $f(i) = f(j)$. Classically this requires $\Omega(N)$ queries to the function f. Ambainis showed how a quantum random walk algorithm for this problem could be made to work using $O(N^{2/3})$ queries: an improvement which has not been achieved using any other quantum methods to date. Other algorithms that admit speedups of a similar nature by using quantum random walks are spatial search (searching a spatially d-dimensional space),[4] triangle finding,[20] and verifying matrix products.[7] Quantum random walks algorithms, then, are a powerful tool for deriving new quantum algorithms.

These examples all achieved polynomial speedups over the best possible classical algorithms. Given that quantum random walks can be used to polynomially outperform classical computers at some tasks, a natural question is whether quantum computers can be used to exponentially outperform classical computers. The answer to this question was first given by Childs et al.,[8] who showed that a quantum random walk could traverse a graph exponentially faster than any possible classical algorithm walking on this graph. In Figure 3 we show the graph in question: the crux of the idea is that a quantum algorithm, by constructively or destructively interfering, can traverse this graph, while a classical algorithm will always get stuck in the middle of the graph. Construc-

Figure 3. An example of a graph arising in the quantum random walk problem considered by Childs et al.[8]

In this problem one is given access to a function that takes as input a vertex and returns a list of the vertex's neighbors. The goal of the problem considered by Childs et al. is, by querying the function as few times as possible, traverse from the start vertex to the end vertex. The graphs considered are two full binary trees pasted together with a random cycle (in the example, the cycle resides inside the dashed box) whose roots are the start and end vertices. The quantum algorithm starts at the start vertex and then performs a quantum diffusion to the end vertex. The random cycle in the middle does not destroy this diffusion, since all paths contribute equally to this diffusion. For a graph of depth d, the quantum walk will find the end vertex by querying the local vertex function a polynomial number of times in d. The best classical algorithm can be shown to require querying the function for local vertex information exponentially many times in d.



tive interference refers to the condition where quantum evolution causes amplitudes to increase in absolute magnitude (and hence in probability) while destructive interference refers to where the evolution causes amplitudes to decrease in absolute magnitude (and hence decrease in probability). In spite of this success, the above problem, traversing this graph, does not appear to have a good algorithmic use. Thus a subject of great research interest today is whether there are quantum random walk algorithms that offer exponential speedups over classical algorithms for interesting algorithmic problems.

Quantum Algorithms and Game Playing

Quantum interference, the ability of multiple computational paths to add or detract amplitudes and thus lower and raise probabilities, is an effect well known to physicists. Given this, it is interesting to ask whether other techniques from physicists toolbox might also be of use in algorithms. A great example of this approach was the recent discovery by Farhi et al.[10] of a quantum algorithm that outperforms all possible classical algorithms for the evaluation of NAND tree circuits. This algorithm was derived, amazingly, by considering the scattering of wave packets off certain binary trees. As a quintessential physics experiment involves shooting one quantum system at another and observing the resulting scattered 'outputs,' physicists have developed a host of tools for analyzing such scattering experiments. It was this approach that led the above authors to the following important new quantum algorithm.

To illustrate the NAND tree problem consider the following two player game. The players are presented with a complete binary tree of depth k. On the leaves of the tree are labels that declare whether player A or player B wins by getting to this node. At the beginning of a match, a marker is placed at the root of the tree. Players take alternating turns moving this marker down a level in the tree, choosing one of the two possible paths, with the goal, of course, of ending up at a leaf labeled by the player's name. A natural question to ask is if it is always possible for player A, with its first move, to win the game. Evaluating whether this is the case can be deduced inductively in the following way. Suppose player A makes the last move. Then player A will be able to win if the marker is on a node with at least one of its children labeled "A wins" hence we should label such internal nodes with "A wins" as well. This line of reasoning holds in general for all internal nodes on which A makes a move: as soon as one of its children has the label "A wins," then the node inherits the same conclusion. On the other hand, if none of the children has this label, then we can conclude that "B wins." Player B will, of course, be reasoning in a similar manner. Thus we can see that player A will win, starting from a node of height two, only if both of the children of the node lead to positions where A wins. We can then proceed inductively using this logic to evaluate whether player A can always win the game with a move originating from the root of the tree. If we label the leaves where player A wins by 1 and where player B wins by 0, then we can compute the value of the root node (indicating whether player A can always win) by representing the interior layers of the tree by alternating layers of AND and OR gates. Further it is easy to see that one can transform this from alternating layers of AND and OR gates to uniform layers of NAND (negated AND) gates, with a possible flipping of the binary values assigned to the leaves.

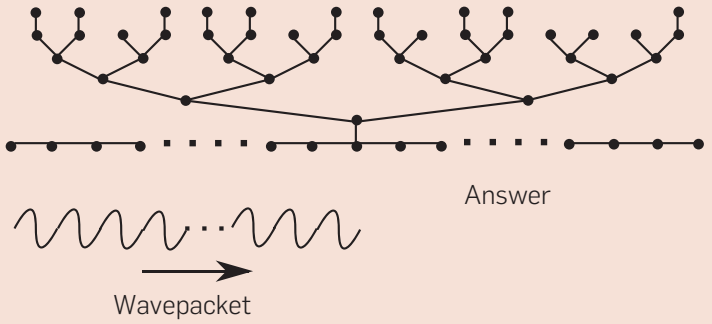
We have just shown that the problem of evaluating whether the first player has a series of moves that guarantees victory is equivalent to evaluating the value of a NAND tree circuit given a labeling the leaves of the tree. Further, if the player can evaluate any interior value of the NAND tree, then one can then use this to actually win the game. If such a procedure is available one can simply use the algorithm to evaluate the two trees and if one of them is always a win, take that move. Thus the problem of evaluating the value of the NAND tree is of central importance for winning this game. The NAND tree is an example of the more general concept of a game tree which is useful for study of many games such as Chess and Go. In these later games, more than two moves are available, but a similar logic for evaluating whether there is a winning strategy applies. This problem, of which the NAND tree circuit is the smallest example, is a central object in the study of combinatorial games.

One can now ask: how costly is it to evaluate the NAND tree: how many nodes does one need to query in order to compute the value of the NAND tree? One could evaluate every leaf and compute the root, but certainly this is wasteful: if you ever encounter a subtree which evaluates to 0, you know that the parent of this subtree must evaluate to 1. A probabilistic recursive algorithm is then easy to think up: evaluate a subtree by first evaluating randomly either its left or right subtree. If this (left or right) subtree is 0, then the original subtree must have value 1. If not, evaluate the other subtree. This method, known as alpha–beta pruning, has a long history in artificial intelligence research. For the NAND tree, one can show that by evaluating about $\Omega(N^{0.753})$ of the N leaves one can calculate the value of the NAND tree with high probability. It is also known that this value for the number of leaves needed to be queried is optimal.

For a long period of time it was uncertain whether quantum computers could perform better than this. Using standard lower bounding methods, the best lower bound which could be proved was a $O(N^{1/2})$, yet no quantum algorithm was able to achieve such a speedup over the best classical algorithm. Enter onto the scene the physicists Farhi, Goldstone, and Gutmann. These authors considered a continuous quantum random walk of a strange form. They considered a quantum random walk on the graph formed by a binary tree (of size related to the NAND tree being evaluated) attached to a long runway (see Figure 4). They then showed how, if one constructed an initial quantum system whose initial state was that of a quantum system moving to the right towards the binary tree, one could then obtain the value of the NAND tree by seeing whether such a quantum system scattered back off the binary tree, or passed through along the other side of the runway. The time required to see this scattering or lack of scattering was shown to be proportional to $O(N^{1/2})$. In other words, the NAND tree could be evaluated by using

Figure 4. The NAND tree algorithm of Farhi, Goldstone, and Gutmann.[10]

First, a tree is constructed where the presence or absence of leaves at the top of the tree corresponds to the binary input values to the NAND tree problem. Next, a wavepacket is then constructed which, if the tree were not attached, would propagate to the right. When the tree is attached, as shown, the value of the NAND tree can be determined by running the appropriate quantum walk and observing whether the wave packet passes to the right of the attached tree or is reflected backwards.



$O(N^{1/2})$ time by scattering a wave packet off of a binary tree representing the NAND tree problem. A few simple modifications can bring this in line with the standard computer scientists definition of a query algorithm for the NAND tree problem. Presto, out of a scattering experiment, one can derive a quantum algorithm for the NAND tree problem which gives a $O(N^{1/2})$ algorithm outperforming a classical computer science algorithm. Building upon this work, a variety of different trees with different branching ratios and degrees of being balanced have been explored showing quantum speedups. Indeed one remarkable aspect of much of this work is that while in many cases the classical versions of these problems do not have matching upper and lower bounds, in the quantum case matching upper and lower bounds can now be achieved.

Finding Hidden Symmetries

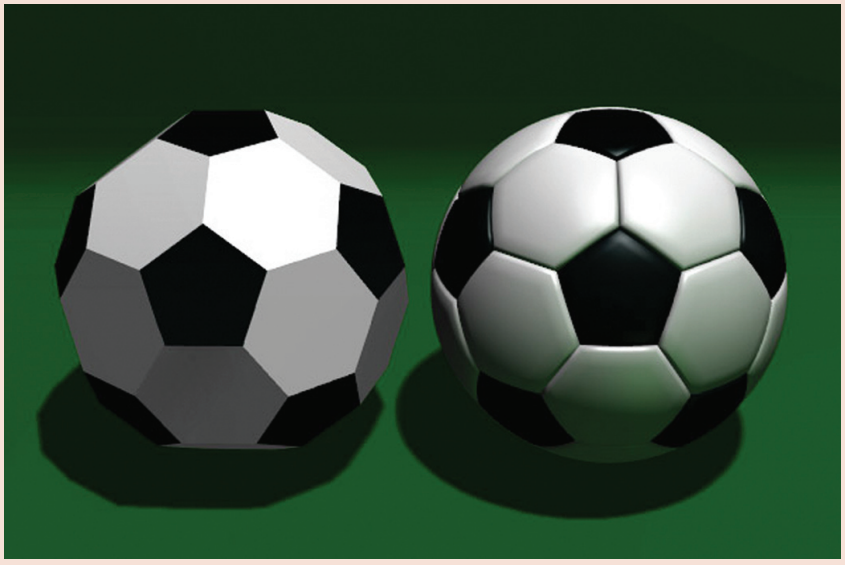
If interference is a quantum effect that leads to polynomial speedups, what about the quantum algorithms that appear to offer exponential speedups, like in Shor's algorithm for factoring or the quantum random walk algorithm of Childs et al. described here? Here it seems that just using interference by itself is not sufficient for gaining such extraordinary power. Instead, in the vast majority of cases where we have exponential speedups for quantum algorithms, a different candidate emerges for giving quantum computers power: the ability to efficiently find hidden symmetries. Here we review recent progress in algorithms concerning hidden symmetries. In many respects these algorithms date back to the earliest quantum algorithms, a connection we first briefly review, before turning to more modern ways in which this has influenced finding new quantum algorithms.

We say an object has *symmetry* if "we can do something to it without changing it." The things we can do are described by the elements of a group and the object itself is a function that is defined on the same group. That this does not have to be as abstract as it seems is illustrated in Figure 5 for the group of three-dimensional rotations and the icosahedral symmetry of a soccer ball.

Given a group G the symmetry of a function f defined on G can range from the trivial (when only the identity of G leaves f unchanged) to the maximum possible symmetry where f remains unchanged under all possible group operations. The most interesting cases happen when f is invariant under only a proper *subgroup* H of G and the task of finding this H, given f, is known as the *hidden subgroup problem*. For many different types of groups we know how to solve this problem efficiently on a quantum computer, while no classical algorithm can perform the same feat. We claim that this is because quantum computers can more efficiently exploit problems with hidden symmetries.

To illustrate how quantum computers are better suited to deal with symmetries, let's talk about the simplest symmetry one can talk about: the symmetry of flipping a bit. Consider the operation X of negating a bit and the identity operation I. If we perform X twice, we obtain the operation I of doing nothing at all, which shows that I and X together form a group. Next, consider representing how I and X

Figure 5. The symmetries of a soccer ball.



Of all the possible three-dimensional rotations that one can apply, only a finite number of them leave the image of a standard soccer ball unchanged. This subgroup, the icosahedral rotation group with its 60 elements, therefore describes the symmetries of the object; http://en.wikipedia.org/wiki/File: Trunc-icosa.jpg/

operate on a classical probabilistic bit. Such a binary system is described by a two-dimensional vector of probabilities, corresponding to the probability p_0 of being in 0 and p_1 of being in 1. The operations I and X can then be represented on this system as the two-by-two matrices

$$I \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } X \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

In group theoretic parlance, we say that these two matrices form a *representation* of the group, which effectively means that the multiplication among these matrices mimics the operation among the elements of the group that is being represented.

But now notice how the matrices for I and X act on the vector space $\mathbb{R}^2$. Naturally, the identity matrix I leaves all vectors unchanged, but the X matrix acts in a more interesting way. If X acts on the symmetric vector [1, 1], then, like I, it preserves this vector. If, on the other hand, X operates upon the vector [1, −1], then it multiplies this vector by −1. This new vector [−1, 1] still sits in the one-dimensional subspace spanned by the original [1, −1], but the direction of the vector has been reversed. In other words, the act of flipping a bit can naturally be represented down into its action upon two one-dimensional subspaces: on the first of these the group always acts trivially, while on the other it always acts by multiplying by the scalar −1. Now we can see why classical probabilistic information is at odds with this symmetry: while we can create a symmetric probability distribution $[\frac{1}{2}, \frac{1}{2}]$ wherein the bit flip X preserves this distribution, we cannot create the other probability distribution that transforms according to the multiplication by −1: doing so would require that we have negative probabilities. But wait, this is exactly what the amplitudes of quantum computers allow you to do: to prepare and analyze quantum information in all the relevant subspaces associated with group operations such as flipping a bit. Unlike classical computers, quantum computers can analyze symmetries by realizing the unitary transforms which directly show the effects of these symmetries. This, in a nutshell, is why quantum algorithms are better adapted to solve problems that involve symmetries.

The idea that symmetry is the excelsior of exponential quantum speedups now has considerable evidence in its favor and is one of the major motivators for current research in quantum algorithms. Shor's algorithm for factoring works by converting the problem of finding divisors to that of finding periods of a function defined over the integers, which in turn is the problem of determining the translational symmetries of this function. In particular Shor's algorithm works by finding the period of the function $f(x) = r^x \bmod N$ where r is a random number coprime with N, the number one wishes to factor. If one finds the period of this function, i.e. the smallest nonzero p such that $f(x) = f(x + p)$, then one has identified a p such that $x^p = 1 \bmod N$. If p is even (which happens with constant probability for random x), then we can express this equation as $(x^{p/2} + 1)(x^{p/2} - 1) = 0 \bmod N$. This implies that the greatest common divisor of $x^{p/2} + 1$ and N or the greatest common divisor of $x^{p/2} - 1$ and N is a divisor of N. One can then use the Euclidean algorithm to find a factor of N (should it exist). Thus one can efficiently factor assuming one can find the period p of $f(x)$. This fact was known before Shor's discovery; the task of determining the period p is what requires a quantum computer.

How then, can a quantum algorithm find the period p of a function f? The answer is: by exploiting the just described friendly relationship between quantum mechanics and group theory. One starts with a system of two quantum registers, call them left and right. These are prepared into a state where with equal amplitude the left register contains a value x and the right register carries the corresponding function value $f(x)$. The hidden symmetry of this state is captured by the fact that it remains unchanged if we would and p (or a multiple of p) to the left register; adding a nonmultiple of p will, on the other hand, change the state. To extract this hidden symmetry, let us view the amplitudes of the state as the values of a function from n bit strings to the complex numbers. We would like to use a quantum version of the Fourier transform to extract the symmetry hidden in this function. Why the Fourier transform? The answer to this is that the Fourier transform is intimately related to the symmetry of addition modulo N. In particular if we examine the process of addition where we have performed a Fourier transform before the addition and an inverse Fourier transform after the addition, we will find that it is now transformed from an addition into multiplication by a phase (a complex number z such that $|z| = 1$). Addition can be represented on a quantum computer as a permutation matrix: a matrix with only a single one per column and row of the matrix. If we examine how such a matrix looks in the basis change given by the Fourier transform, then we see that this matrix only has entries on the diagonal of the matrix. Thus the Fourier transform is exactly the unitary transform which one can use to "diagonalize the addition matrix" with respect to the symmetry of addition, which in turn is exactly the form of the symmetry needed for period finding.

The output of the quantum Fourier transformation will reveal to us which symmetries the state has, and by repeating this Fourier sampling a few times we will be able to learn the exact subgroup that the state hides, thus giving us the period p (and hence allowing us to factor). Crucially the quantum Fourier transform can be implemented on a number of qubits logarithmic in the size of the addition group, $\log N$, and in a time polynomial in $\log N$ as well. If one were to attempt to mimic Shor's algorithm on a classical computer, one would need to perform a Fourier transform on N classical pieces of data, which would require $N \log N$ time (using the fast Fourier transform). In contrast, because Shor's quantum algorithm acts on quantum amplitudes, instead of on classical configuration data, it leads to an efficient quantum algorithm for factoring.

This symmetry analysis results from the basics of the theory of group representation theory: symmetries are described by groups, and the elements of these groups can be represented by unitary matrices. This is something

that classical probabilistic computers cannot exploit: the only way to represent a group on a classical computer is to represent it as by deterministic permutation. But while a group can be represented by unitary matrices, no such representation is possible using stochastic matrices. This, at its heart, appears to be one of the key reasons that quantum computers offer exponential benefits for some problems over classical computers.

Given that Shor's algorithm exploits symmetry in such a successful way, it is natural to search for other problems that involve hidden symmetries. Following Shor's discovery it was quickly realized that almost all prior quantum algorithms could be cast in a unifying form as solving the hidden subgroup problem for one group or the other. For Shor's algorithm the relevant group is the group of addition modulo N. For the discrete logarithm problem the relevant group is the direct product of the groups of addition modulo N. Indeed it was soon discovered that for all finite Abelian groups (Abelian groups are those whose elements all commute with each other) quantum computers could efficiently solve the hidden subgroup problem. A natural follow-up question is: what about the non-Abelian hidden subgroup problem? And, even more importantly, would such an algorithm be useful for any natural problems, as the Abelian hidden subgroup problem is useful for factoring?

One of the remarkable facts about the problem of factoring is its intermediate computational complexity. Indeed, if one examines the decision version of the factoring problem, one finds that this is a problem which is in the complexity class NP and in the complexity class Co-NP. Because of this fact it is thought to be highly unlikely that it is NP-complete, since if it were, then the polynomial hierarchy would collapse in a way thought unlikely by complexity theorists. On the other hand, there is no known classical algorithm for factoring. Thus factoring appears to be of Goldilock's complexity: not so hard as to revolutionize our notion of tractability by being NP-complete, but not so easy as to admit efficient classical solution. There are, surprisingly, only a few problems which appear to fit into this category. Among them, however, are the problems of graph isomorphism and certain shortest-vector in a lattice problems. Might quantum computers help at solving these problems efficiently?

Soon after Shor's algorithm was phrased as a hidden subgroup problem, it was realized that if you could efficiently solve the hidden subgroup problem over the symmetric group (the group of permutations of n objects), then you would have an efficient quantum algorithm that solves the graph isomorphism problem. Further, Regev[22] showed how the hidden subgroup problem over the dihedral group (the group of symmetries of a regular polygon where one can not only rotate but also flip the object) relates to finding short vectors in a high dimensional lattice. Hence a hypothetical efficient quantum algorithm for this dihedral case could be used to solve such shortest vector problems. This in turn would break the public key cryptosystems that are based upon the hardness of these lattice problems, which are among the very few cryptosystems not broken by Shor's algorithm. As a result of these observations about the non-Abelian hidden subgroup problem, designing quantum algorithms for such groups has become an important part of the research in quantum computation. While a certain amount of progress has been achieved (by now we know of many non-Abelian groups over which the hidden subgroup problem can be solved efficiently), this problem remains one of the outstanding problems in the theory of quantum algorithms.

At the same time, going back to the Abelian groups, there has been quite some success in finding new applications of the quantum algorithm for the Abelian hidden subgroup problem, besides factoring and discrete logarithms. Hallgren[14] showed that there exists a quantum algorithm for solving Pell's equation (that is, finding integer solutions x, y to the cubic equation $x^2 - dy^2 = 1$, see Table 1), while Kedlaya[15] has described a quantum procedure that efficiently counts the number points of curves defined over finite fields. Furthermore, other efficient quantum algorithm has been found for, among other problems, determining the structure of black box groups, estimating Gauss sums, finding hidden shifts, and estimating known invariants.

Simulating Quantum Physics

A final area in which quantum algorithms have made progress goes back to the very roots of quantum computing and indeed of classical computing itself. From their earliest days, computers have been put to use in simulating physics. Among the difficulties that were

Table 1. Some examples of integer solutions (x, y) to Pell's equation $x^2 - dy^2 = 1$ for different values d.

Such solutions tell us what the units are of the number field $\mathbb{Q}[\div \sqrt{d}]$ (the rational numbers extended with the irrational $\div \sqrt{d}$) and thereby solve the unit group problem. Hallgren's result shows how this problem can be solved efficiently on a quantum computer, while no such algorithm is known for classical computers.

d	x	y
2	3	2
3	2	1
5	9	4
⋮		
13	649	180
14	15	4
⋮		
6,009	1,316,340,106,327,253,158	1,698,114,661,157,803,451
	9,259,446,951,059,947,388	6,889,492,378,831,465,766
	$4{,}013{,}975 \approx 1.3 \times 10^{44}$	$81{,}644 \approx 1.6 \times 10^{42}$
6,013	40,929,908,599	527,831,340
⋮		

soon encountered in such simulations was that quantum systems appeared to be harder to simulate than their classical counterparts. But, of course, somehow nature, which obeys quantum theory, is already carrying out "the simulation" involved in quantum physics. So, if nature is carrying out the simulation, then should we be able to design a computer that also can perform this simulation? This was in fact the seed of the idea that led to the original notion of quantum computing by Feynman.[11]

To put this in perspective, consider the problem of simulating classical physics. The miracle of reproducing classical physics on a classical computer is that you can use many 'particles' with small state spaces (bits) to mimic a few particles that have very large state spaces. For this to be possible it is required that the number of bit configurations, $2^{(\text{number of bits})}$, is at least as big as the number of possible states of the physical system (which is the size of the particle's state space exponentiated with the number of particles). As a result, we can simulate the solar system on a laptop.

Quantum computing does the same thing for quantum mechanical systems; now $2^{(\text{number of qubits})}$ is the dimension of the state space and it allows us to simulate other quantum physical systems that consists of few particles with exponentially large state spaces. Here however, it appears essential that we rely on quantum computing components in order to simulate the truly quantum mechanical components of a physical system. A crucial question therefore is: which physical systems are interesting to simulate in such a manner?

While the complete answer to this question is not known, a deeper look at quantum algorithms for simulating quantum physics is now being undertaken in several places. As an example, a group of physical chemists have recently compared how useful quantum computers would be for computing the energy level structure of molecular systems.[5] This is a classical problem of physical chemistry, and our inability to perform these calculations robustly for large molecules is a bottleneck in a variety of chemical and biological applications. Could quantum computers help for solving this problem and outperforming the best classical algorithms? One of the exciting findings in studying this problem was that a small quantum computer, consisting of only a few hundred qubits, could already outperform the best classical algorithms for this problem. This small number makes it likely that among the first applications of a quantum computer will not be factoring numbers, but instead will be in simulating quantum physics. Indeed, we believe that a quantum computer will be able to efficiently simulate my possible physical system and that it therefore has the potential to have a huge impact on everything from drug discovery to the rapid development of new materials.

Conclusion

The discovery that quantum computers could efficiently factor is, even today, difficult to really appreciate. There are many ways to get out of the conundrum posed by this discovery, but all of these will require a fundamental rewriting of our understanding of either physics or computer science. One possibility is that quantum computers cannot be built because quantum theory does not really hold as a universal theory. Although disappointing for quantum computer scientists, such a conclusion would be a major discovery about one of the best tested physical theories—quantum theory. Perhaps there is a classical algorithm for efficiently factoring integers. This would be a major computer science discovery and would blow apart our modern public key cryptography. Or perhaps, just perhaps, quantum computers really are the true model of computing in our universe, and the rules of what is efficiently computable have changed. These are the dreams of quantum computer scientists looking for quantum algorithms on the quantum machines they have yet to be quantum programmed. C

References

1. Aharonov, D., Ben-Or, M. Fault-tolerant quantum computation with constant error rate. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (1997). ACM, 176–188.
2. Aharonov, Y., Davidovich, L., Zagury, N. Quantum random walks. *Phys. Rev. A 48*, 167 (1993).
3. Ambainis, A. Quantum walk algorithm for element distinctness. *SIAM J. Comput. 37* (2007), 210.
4. Ambainis, A., Kempe, J., Rivosh, A. Coins make quantum walks faster. In *Proceedings of the 16th Annual ACM SIAM Symposium on Discrete Algorithms* (2005), 1099.
5. Aspuru-Guzik, A., Dutoi, A., Love, P.J., Head-Gordon, M. Simulated quantum computation of molecular energies. *Science 309*, 5741 (2005).
6. Bell, J.S. On the Einstein Podolsky Rosen paradox. *Physics 1*, (1964), 195.
7. Buhrman, H. Špalek, R. Quantum verification of matrix products. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms* (2006), 880.
8. Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.A. Exponential algorithmic speedup by quantum walk. In *Proceedings of the 35th ACM Symposium on Theory of Computing* (2003), 59–68.
9. Farhi, E., Gutmann, S. Quantum computation and decision trees. *Phys. Rev. A 58* (1998), 915.
10. Farhi, E., Goldstone, J., Gutmann, S. A quantum algorithm for the Hamiltonian NAND tree. Eprint arXiv:quant-ph/0702144, 2007.
11. Feynman, R. Simulating physics with computers. *Intl. J. Theor. Phys. 21* (1982), 467–488.
12. Grover, L. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computation* (New York, 1996). ACM, 212–219.
13. Häffner, H., Hänsel, W., Roos, C.F., Benhelm, J., al kar, D.C., Chwalla, M., Körber, T., Rapol, U.D., Riebe, M., Schmidt, P.O., Becher, C., Gühne, O., Dür, W., Blatt, R. Scalable multiparticle entanglement of trapped ions. *Nature 438* (2005), 643.
14. Hallgren, S. Polynomial-time quantum algorithms for pell's equation and the principal ideal problem. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computation* (New York, 2002). ACM, 653–658.
15. Kedlaya, K.S. Quantum computation of zeta functions of curves. *Comput. Complex. 15*, 1–19 (2006).
16. Kitaev, A. Quantum error correction with imperfect gates. In *Quantum Communication, Computing and Measurement* (New York, 1997). Plenum, 181–188.
17. Knill, E., Laflamme, R., Zurek, W.H. Resilent quantum computation. *Science 279* (1998), 342–345.
18. Knill, E., Laflamme, R., Zurek, W.H. Resilient quantum computation: error models and thresholds. *Proc. Roy. Soc. Lond. Ser. A 454* (1998), 365–384.
19. Leibfried, D., Knill, E., Seidelin, S., Britton, J., Blakestad, R.B., Chiaverini, J., Hume, D.B., Itano, W.M., Jost, J.D., Langer, C., Ozeri, R., Reichle, R., Wineland, D.J. Creation of a six-atom 'Schrödinger cat' state. *Nature 438* (2005), 639.
20. Magniez, F., Santha, M., Szegedy, M. Quantum algorithms for the triangle problem. In *Proceedings of the 16th Annual ACM SIAM Symposium on Discrete Algorithms* (2005), 1109.
21. Nielsen, M.A. Chuang, I.L. *Quantum Computation and Quantum Information*. Cambridge University Press, New York, 2000.
22. Regev, O. Quantum computation and lattice problems. In *43rd Symposium on Foundations of Computer Science* (IEEE Computer Society, 2002), 520–529.
23. Rivest, R.L., Shamir, A., Adleman, L. A method of obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21* (1978), 120–126.
24. Shor, P.W. Algorithms for quantum computation: Discrete log and factoring. In *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*. S. Goldwasser, ed. (Los Alamitos, CA, 1994). IEEE Computer Society, 124–134.
25. Shor, P.W. Fault tolerant quantum computation. In *Proceedings of the 37th Symposium on the Foundations of Computer Science* (Los Alamitos, CA, 1996), IEEE, 56–65.
26. Woehr, J. Online interview "A Conversation with Christos Papadimitriou". *Dr. Dobb's J.* July Dave Bacon is an assistant research professor in the Department of Computer Science and Engineering, Department of Physics, at the University of Washington, Seattle.

Dave Bacon (dabacon@cs.washington.edu) is an assistant research professor in the Department of Computer Science & Engineering, Department of Physics, University of Washington, Seattle, WA.

Wim van Dam (vandam@cs.ucsb.edu) is an associate professor in the Department of Computer Science, Department of Physics, University of California, Santa Barbara, Santa Barbara, CA.

Group Term Life Insurance**

10- or 20-Year Group Term Life Insurance*

Group Disability Income Insurance*

Group Accidental Death & Dismemberment Insurance*

Group Catastrophic Major Medical Insurance*

Group Dental Plan*

Long-Term Care Plan

Major Medical Insurance

Short-Term Medical Plan***

Who has time to think about insurance?

Today, it's likely you're busier than ever. So, the last thing you probably have on your mind is whether or not you are properly insured.

But in about the same time it takes to enjoy a cup of coffee, you can learn more about your ACM-sponsored group insurance program — a special member benefit that can help provide you financial security at economical group rates.

Take just a few minutes today to make sure you're properly insured.

Call Marsh Affinity Group Services at **1-800-503-9230** or visit **www.personal-plans.com/acm.**

MARSH
Affinity Group Services
a service of Seabury & Smith

research highlights

DOI:10.1145/1646353.1646378

Technical Perspective
Strange Effects in High Dimension

By Sanjoy Dasgupta

IN STUDYING THE genetic basis of a disease, it is now common to select a set of relevant genes G, and to measure how strongly they are expressed in cell samples from a group of patients, some healthy and some ill.[1] The expression level of a gene is mapped to a value in [−1,1]. Each patient's data is then a vector with one entry per gene, or equivalently, a point in $R^{|G|}$. The size of G is frequently in the thousands, which makes the data high-dimensional by present standards.

Analyzing data in a high-dimensional space R^d is rife with challenges. First, many statistical estimators are accurate only when the number of data points (call it *n*) is orders of magnitude larger than *d*. Some models, such as histograms, have error rates of the form $n^{-1/d}$, so that to halve the error, you need 2^d times as much data: bad news even for double-digit *d*. A second difficulty is computational, as typified by the ever-important *2-means* clustering problem: divide a data set into two groups, each with a designated center, to minimize the average squared distance from data points to their closest centers. Naive approaches run in time n^d, which is astronomical even for smallish *d*, and NP-hardness results have dampened hopes for dramatically better algorithms. As a result, such problems are attacked with heuristics that offer no guarantees on their solutions. Understanding the quality of these schemes is doubly tricky because they are often justified on intuitive grounds, inevitably informed by experiences of 2D and 3D space—while high dimensional spaces are full of strange and counterintuitive effects.

Such complications are collectively referred to as the *curse of dimension*: A superstition that the murky realm of high dimension brings bad luck. But mathematical analysis is starting to clear away the haze. A pleasant surprise is that the counterintuitive geometry of high-dimensional space, once properly characterized, can be exploited to defeat other facets of the curse.

Even a solid ball—the simplest of objects—has unusual aspects in high dimension. For d=2 or d=3, a set of points picked at random from the unit ball $B_d = \{x \text{ in } R^d: ||x|| \le 1\}$ will have some significant fraction near the origin, say within distance ½ of it. But we wouldn't find even one such point in high dimension unless we were to draw exponentially many points from the ball. This is because points at distance $r < 1$ from the origin constitute an r^d fraction of B_d, and this fraction goes rapidly to zero with rising dimension. For large *d*, the supposedly filled-in ball B_d is in fact well approximated by a thin, hollow shell, $\{x \text{ in } R^d: 1-\varepsilon \le ||x|| \le 1\}$ for $\varepsilon = O(1/d)$.

Here's another curiosity. Suppose we need lots of vectors in R^d that are orthogonal (at right angles) to each other. How many can we get? Exactly *d*, because this is the maximum number of linearly independent vectors. But if we only need them approximately orthogonal—with angles that need not be exactly 90 degrees, but $90\pm\varepsilon$—then we can find exponentially many vectors. A collection of $\exp(O(\varepsilon^2 d))$ vectors picked at random from the surface of B_d will with high probability satisfy the angle constraint.

The *curse of dimension*: A superstition that the murky realm of high dimension brings bad luck. But mathematical analysis is starting to clear away the haze. A pleasant surprise is that counterintuitive geometry can defeat other facets of the curse.

These examples hint at the strangeness of high-dimensional space. However, such effects do not directly help with data analysis because they pertain to very specialized sets of points—those chosen randomly from the unit ball—whereas real data sets might look different. The trick is to take an arbitrary data set and then *add* randomness to it in such a way that the outcome is helpful and predictable.

An early groundbreaking result of this kind was *Dvoretsky's theorem*.[2] Let K be a convex body in R^d: for instance, the feasible region of a linear program with d variables. K could be enormously complicated. But a random slice through K (of appropriate dimension) will with high probability be almost ellipsoidal. More precisely, it will contain an ellipsoid E and be contained within (1+o(1))E.

A more recent result is the *Johnson-Lindenstrauss theorem*.[3] Take any *n* points in Euclidean space of arbitrarily high dimension. If they are projected into a random subspace of O(log n) dimensions, distances between points will with high probability be almost perfectly preserved. Since clustering and other forms of data analysis frequently depend only on interpoint distances, the dimension of data sets can automatically be reduced to O(log n).

The following paper by Nir Ailon and Bernard Chazelle entitled "Faster Dimension Reduction" demonstrates an ingenious variant of this theorem that permits the projection to be achieved especially fast.

References

1. Alizadeh, A. et al. Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature 403* (2000), 503–511.
2. Dvoretzky, A. Some results on convex bodies and Banach spaces. In *Proceedings of the International Symposium on Linear Spaces* (Jerusalem, 1961), 123–160.
3. Johnson, W. and Lindenstrauss, J. Extensions of Lipschitz maps into a Hilbert space. *Contemporary Mathematics 26* (1984), 189–206.

Sanjoy Dasgupta is an associate professor in the Department of Computer Science and Engineering at the University of California, San Diego.

DOI:10.1145/1646353.1646379

Faster Dimension Reduction

By Nir Ailon and Bernard Chazelle

Abstract

Data represented geometrically in high-dimensional vector spaces can be found in many applications. Images and videos, are often represented by assigning a dimension for every pixel (and time). Text documents may be represented in a vector space where each word in the dictionary incurs a dimension. The need to manipulate such data in huge corpora such as the web and to support various query types gives rise to the question of how to represent the data in a lower-dimensional space to allow more space and time efficient computation. Linear mappings are an attractive approach to this problem because the mapped input can be readily fed into popular algorithms that operate on linear spaces (such as principal-component analysis, PCA) while avoiding the curse of dimensionality.

The fact that such mappings even exist became known in computer science following seminal work by Johnson and Lindenstrauss in the early 1980s. The underlying technique is often called "random projection." The complexity of the mapping itself, essentially the product of a vector with a dense matrix, did not attract much attention until recently. In 2006, we discovered a way to "sparsify" the matrix via a computational version of Heisenberg's Uncertainty Principle. This led to a significant speedup, which also retained the practical simplicity of the standard Johnson–Lindenstrauss projection. We describe the improvement in this article, together with some of its applications.

1. INTRODUCTION

Dimension reduction, as the name suggests, is an algorithmic technique for reducing the dimensionality of data. From a programmer's point of view, a d-dimensional array of real numbers, after applying this technique, is represented by a much smaller array. This is useful because many data-centric applications suffer from exponential blowup as the underlying dimension grows. The infamous curse of dimensionality (exponential dependence of an algorithm on the dimension of the input) can be avoided if the input data is mapped into a space of logarithmic dimension (or less); for example, an algorithm running in time proportional to 2^d in dimension d will run in linear time if the dimension can be brought down to log d. Common beneficiaries of this approach are clustering and nearest neighbor searching algorithms. One typical case involving both is, for example, organizing a massive corpus of documents in a database that allows one to respond quickly to similar-document searches. The clustering is used in the back-end to eliminate (near) duplicates, while nearest-neighbor queries are processed at the front-end. Reducing the dimensionality of the data helps the system respond faster to both queries and data updates. The idea, of course, is to retain the basic metric properties of the data set (e.g., pairwise distances) while reducing its size. Because this is technically impossible to do, one will typically relax this demand and tolerate errors as long as they can be made arbitrarily small.

The common approaches to dimensionality reduction fall into two main classes. The first one includes *data-aware* techniques that take advantage of prior information about the input, principal-component analysis (PCA) and compressed sensing being the two archetypical examples: the former works best when most of the information in the data is concentrated along a few fixed, unknown directions in the vector space. The latter shines when there exists a basis of the linear space over which the input can be represented sparsely, i.e., as points with few nonzero coordinates.

The second approach to dimension reduction includes *data-oblivious* techniques that assume no prior information on the data. Examples include sketches for data streams, locality sensitive hashing, and random linear mappings in Euclidean space. The latter is the focus of this article. Programmatically, it is equivalent to multiplying the input array by a random matrix. We begin with a rough sketch of the main idea.

Drawing on basic intuition from both linear algebra and probability, it may be easy to see that mapping high-dimensional data into a random lower-dimensional space via a linear function will produce an approximate representation of the original data. Think of the directions contained in the random space as samples from a population, each offering a slightly different view of a set of vectors, given by their projection therein. The collection of these narrow observations can be used to learn about the approximate geometry of these vectors. By "approximate" we mean that properties that the task at hand may care about (such as distances and angles between vector) will be slightly distorted. Here is a small example for concreteness. Let $a_1, \ldots, a_d$ be independent random variables with mean 0 and unit variance (e.g., a Gaussian $N(0, 1)$). Given a vector $x = (x_1, \ldots, x_d)$, consider the inner product $Z = \Sigma_i a_i x_i$: the expectation of Z is 0 but its variance is precisely the square of the Euclidean length of x. The number Z can be interpreted as a "random projection" in one dimension: the variance allows us to "read off" the length of x. By sampling in this way several times, we can increase our confidence, using the law of

A previous version of this paper appeared in *Proceedings of the 38th ACM Symposium on Theory in Computing* (May 2006, Seattle, WA).

large numbers. Each sample corresponds to a dimension. The beauty of the scheme is that we can now use it to handle many distances at once.

It is easy to see that randomness is necessary if we hope to make meaningful use of the reduced data; otherwise we could be given as input a set of vectors belonging to the kernel of any fixed matrix, thus losing all information. The size of the distortion as well as the failure probability are user-specified parameters that determine the target (low) dimension. How many dimensions are sufficient? Careful quantitative calculation reveals that, if all we care about is distances between pairs of vectors and angles between them—in other words, the Euclidean *geometry* of the data—then a random linear mapping to a space of dimension *logarithmic* in the size of the data is sufficient. This statement, which we formalize in Section 1.1, follows from Johnson and Lindenstrauss's seminal work.[25] The consequence is quite powerful: If our database contains n vectors in d dimensions, then we can replace it with one in which data contains only $\log n$ dimensions! Although the original paper was not stated in a computational language, deriving a naïve pseudocode for an algorithm implementing the idea in that paper is almost immediate. This algorithm, which we refer to as JL for brevity, has been studied in theoretical computer science in many different contexts. The main theme in this study is improving efficiency of algorithms for high-dimensional geometric problems such as clustering,[37] nearest neighbor searching,[24, 27] and large scale linear algebraic computation.[18, 28, 35, 36, 38]

For many readers it may be obvious that these algorithms are directly related to widely used technologies such as web search. For others this may come as a surprise: Where does a Euclidean space hide in a web full of textual documents? It turns out that it is very useful to represent text as vectors in high-dimensional Euclidean space.[34] The dimension in the latter example can be as high as the number of words in the text language!

This last example illustrates what a *metric embedding* is: a mapping of objects as points in metric spaces. Computer scientists care about such embeddings because often it is easier to design algorithms for metric spaces. The simpler the metric space is, the friendlier it is for algorithm design. *Dimensionality* is just one out of many measures of simplicity. We digress from JL by mentioning a few important results in computer science illustrating why embedding input in simple metric spaces is useful. We refer the reader to Linial et al.[29] for one of the pioneering works in the field.

- The Traveling Salesman Problem (TSP), in which one wishes to plan a full tour of a set of cities, with given costs of traveling between any two cities is an archetype of a computational hardness which becomes easier if the cities are embedded in a metric space,[14] and especially in a low-dimensional Euclidean one.[7, 30]
- Problems such as combinatorial optimization on graphs become easier if the nodes of the graph can be embedded in ℓ_1 space. (The space ℓ_p^d is defined to be the set $\mathbf{R}^d$ endowed with a norm, where the norm of a vector x (written as $\|x\|_p$) is given by $(\Sigma_{i=1}^{d} |x_i|^p)^{1/p}$. The metric is given by the distance between pairs of vectors x and y taken to be $\|x-y\|_p$.)
- Embedding into tree metrics (where the distance between two nodes is defined by the length of the path joining them) is useful for solving network design optimization problems.

The JL algorithm linearly embeds an input which is already in a high-dimensional Euclidean space ℓ_2^d into a lower-dimensional ℓ_p^k space for any $p \geq 1$, and admits a naïve implementation with $O(dk)$ running time per data vector; in other words, the complexity is proportional to the number of random matrix elements.

Our modification of JL is denoted FJLT, for *Fast-Johnson–Lindenstrauss-Transform*. Although JL often works well, it is the computational bottleneck of many applications, such as approximate nearest neighbor searching.[24, 27] In such cases, substituting FJLT yields an immediate improvement. Another benefit is that implementing FJLT remains extremely simple. Later in Section 3 we show how FJLT helps in some of the applications mentioned above. Until then, we concentrate on the story of FJLT itself, which is interesting in its own right.

1.1. A brief history of a quest for a faster JL

Before describing our result, we present the original JL result in detail, as well as survey results related to its computational aspects. We begin with the central lemma behind JL.[25] The following are the main variables we will be manipulating:

X—a set of vectors in Euclidean space (our input dataset). In what follows, we use the term *points* and *vectors* interchangeably.

n—the size of the set X.

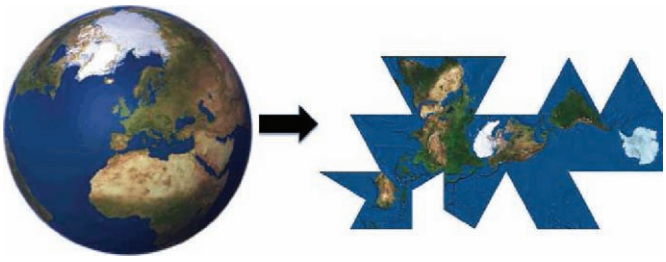
d—the dimension of the Euclidean space (typically very big).

k—the dimension of the space we will reduce the points in X to (ideally, much smaller than d).

ε—a small tolerance parameter, measuring to what is the maximum allowed distortion rate of the metric space induced by the set X in Euclidean m-space (the exact definition of distortion will be given below).

In JL, we take k to be $c\varepsilon^{-2} \log n$, for some large enough

Figure 1. Embedding a spherical metric onto a planar one is no easy task. The latter is more favorable as input to printers.



absolute constant c. We then choose a random subspace of dimension k in $\mathbf{R}^d$ (we omit the mathematical details of what a random subspace is), and define Φ to be the operation of projecting a point in $\mathbf{R}^d$ onto the subspace. We remind the reader that such an operation is linear, and is hence equivalently representable by a matrix. In other words, we've just defined a random matrix. Denote it by Φ.

The JL Lemma states that with high probability, for all pairs of points $x, y \in X$ simultaneously,

$$\sqrt{\frac{k}{d}}\,\|x-y\|_2\,(1-\varepsilon) \le \|\Phi x-\Phi y\|_2 \le \sqrt{\frac{k}{d}}\,\|x-y\|_2\,(1+\varepsilon). \quad (1)$$

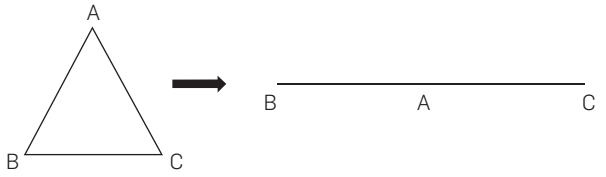
This fact is useful provided that $k < d$, which will be implied by the assumption

$$n = 2^{O(\varepsilon^2 d)}. \quad (2)$$

Informally, JL says that projecting the n points on a random low-dimensional subspace should, *up to a distortion of* $1 \pm \varepsilon$, preserve pairwise distances. The mapping matrix of size $\Phi = k \times d$ can be implemented in a computer program as follows: The first row is a random unit vector chosen uniformly in $\mathbf{R}^d$; the second row is a random unit vector from the space orthogonal to the first row; the third is a random unit vector from the space orthogonal to the first two rows, etc. The high-level proof idea is to show that for each pair $x, y \in X$ the probability of (1) being violated is order of $1/n^2$. A standard union bound over the number of pairs of points in X then concludes the proof.

It is interesting to pause and ask whether the JL theorem should be intuitive. The answer is both yes and no. Low-dimensional geometric intuition is of little help. Take an equilateral triangle ABC in the plane (Figure 2), no matter how you project it into a line, you get three points in a row, two of which form a distance at least twice the smallest one. The distortion is at least 2, which is quite bad. The problem is that, although the expected length of each side's projection is identical, the variance is high. In other words, the projected distance is rarely close to the average. If, instead of $d = 2$, we choose a high dimension d and project down to $k = c\varepsilon^{-2} \log n$ dimensions, the three projected lengths of ABC still have the same expected value, but crucially their (identical) variances are now very small. Why? Each such length (squared) is a sum of k independent random variables, so its distribution is almost normal with variance proportional to k (this is a simple case of the central limit theorem). This fact alone explains each factor in the expression for k: ε^{-2} ensures the desired distortion; $\log n$ reduces the error probability to $n^{-c'}$, for constant c' growing with c, which allows us to apply a union bound over all $\binom{n}{2}$ pairs of distances in X.

Figure 2. A triangle cannot be embedded onto a line while simultaneously preserving distances between all pairs of vertices.



Following Johnson and Lindenstrauss,[25] various researchers suggested simplifications of the original JL design and of their proofs (Frankl and Maehara,[20] DasGupta and Gupta,[17] Indyk and Motwany[24]). These simplifications slightly change the distribution from which Φ is drawn and result in a better constant c and simpler proofs. These results, however, do not depart from the original JL from a computational point of view, because the necessary time to apply Φ to a vector is still order of nk.

A bold and ingenious attempt to reduce this cost was taken by Achlioptas.[1] He noticed that the only property of Φ needed for the transformation to work is that $(\Phi_i \cdot x)^2$ be tightly concentrated around the mean $1/d$ for all unit vectors $x \in \mathbf{R}^d$, where Φ_i is the ith row of Φ. The distribution he proposed is very simple: Choose each element of Φ uniformly from the following distribution:

$$\begin{cases} \sqrt{3}/d & \text{with probability } 1/6; \\ 0 & 2/3; \\ -\sqrt{3}/d & 1/6. \end{cases}$$

The nice property of this distribution is that it is relatively sparse: on average, a fraction 2/3 of the entries of Φ are 0. Assuming we want to apply Φ on many points in $\mathbf{R}^d$ in a real-time setting, we can keep a linked list of all the nonzeros of Φ during preprocessing and reap the rewards in the form of a threefold speedup in running time.

Is Achlioptas's result optimal, or is it possible to get a super constant speedup? This question is the point of departure for this work. One idea to obtain a speedup, aside from sparsifying Φ, would be to reduce the target dimension k, and multiply by a smaller matrix Φ. Does this have a chance of working? A lower bound of Alon[5] provides a negative answer to this question, and dashes any hope of reducing the number of rows of Φ by more than a factor of $O(\log(1/\varepsilon))$. The remaining question is hence whether the matrix can be made sparser than Achlioptas's construction. This idea has been explored by Bingham and Mannila.[11] They considered sparse projection heuristics, namely, fixing most of the entries of Φ as zeroes. They noticed that in practice such matrices Φ seem to give a considerable speedup with little compromise in distortion for data found in certain applications. Unfortunately, it can be shown that sparsifying Φ by more than a constant factor (as implicitly suggested in Bingham and Mannila's work) will not work for all inputs. Indeed, a sparse matrix will typically distort a sparse vector. The intuition for this is given by an extreme case: If both Φ and the vector x are very sparse, the product Φx may be null, not necessarily because of cancellations, but more simply because each multiplication $\Phi_{ij} x_j$ is itself zero.

1.2. The random densification technique

In order to prevent the problem of simultaneous sparsity of Φ and x, we use a central concept from harmonic analysis known as the *Heisenberg principle*—so named because it is the key idea behind the Uncertainty Principle: a signal and its spectrum cannot be both concentrated. The look of

frustration on the face of any musician who has to wrestle with the delay from a digital synthesizer can be attributed to the Uncertainty Principle.

Before we show how to use this principle, we must stop and ask: what are the tools we have at our disposal? We may write the matrix Φ as a product of matrices, or, algorithmically, apply a chain of linear mappings on an input vector. With that in mind, an interesting family of matrices we can apply to an input vector is the *orthogonal* family of d-by-d matrices. Such matrices are *isometries*: The Euclidean geometry suffers no distortion from their application.

With this in mind, we precondition the random k-by-d mapping with a Fourier transform (via an efficient FFT algorithm) in order to isometrically densify any sparse vector. To prevent the inverse effect, i.e., the sparsification of dense vectors, we add a little randomization to the Fourier transform (see Section 2 for details). The reason this works is because sparse vectors are rare within the space of all vectors. Think of them as forming a tiny ball within a huge one: if you are inside the tiny ball, a random transformation is likely to take you outside; on the other hand, if you are outside to begin with, the transformation is highly unlikely to take you inside the tiny ball.

The resulting FJLT shares the low-distortion characteristics of JL but with a lower running time complexity.

2. THE DETAILS OF FJLT

In this section we show how to construct a matrix Φ drawn from FJLT and then prove that it works, namely:

1. It provides a low distortion guarantee. (In addition to showing that it embeds vectors in low-dimensional ℓ_2^k, we will show it also embeds in ℓ_1^k.)
2. Applying it to a vector is efficiently computable.

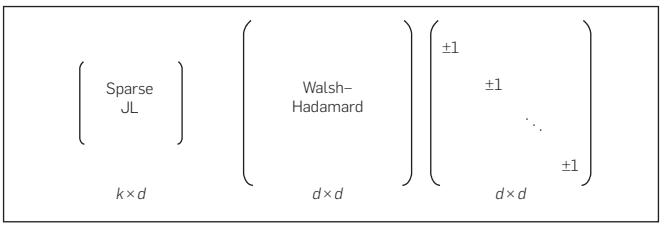
The first property is shared by the standard JL and its variants, while the second one is the main novelty of this work.

2.1. Constructing Φ

We first make some simplifying assumptions. We may assume with no loss of generality that d is a power of two, $d = 2^h > k$, and that $n\,\Omega\, d = \Omega(\varepsilon^{-1/2})$; otherwise the dimension of the reduced space is linear in the original dimension. Our random embedding $\Phi \sim \text{FJLT}(n, d, \varepsilon, p)$ is a product of three real-valued matrices (Figure 3):

$$\Phi = PHD.$$

Figure 3. FJLT.



The matrices P and D are chosen randomly whereas H is deterministic:

- P is a k-by-d matrix. Each element is an independent mixture of 0 with an unbiased normal distribution of variance $1/q$, where

$$q = \min\left\{\Theta\left(\frac{\varepsilon^{p-2}\log^p n}{d}\right), 1\right\}.$$

  In other words, $P_{ij} \sim N(0, 1/q)$ with probability q, and $P_{ij} = 0$ with probability $1 - q$.
- H is a d-by-d normalized Walsh–Hadamard matrix:

$$H_{ij} = d^{-1/2}(-1)^{\langle i-1,\, j-1\rangle},$$

  where $\langle i, j\rangle$ is the dot-product (modulo 2) of the m-bit vectors i, j expressed in binary.
- D is a d-by-d diagonal matrix, where each D_{ii} is drawn independently from $\{-1, 1\}$ with probability 1/2.

The Walsh–Hadamard matrix corresponds to the discrete Fourier transform over the additive group GF $(2)^d$: its FFT is very simple to compute and requires only $O(d \log d)$ steps. It follows that the mapping Φx of *any* point $x \in \mathbb{R}^d$ can be computed in time $O(d \log d + |P|)$, where $|P|$ is the number of nonzero entries in P. The latter is $O(\varepsilon^{-2} \log n)$ not only on average but also with high probability. Thus we can assume that the running time of $O(d \log d + qd\varepsilon^{-2} \log n)$ is worst-case, and not just expected.

The FJLT Lemma. *Given a fixed set X of n points in $\mathbf{R}^d$, $\varepsilon < 1$, and $p \in \{1, 2\}$, draw a matrix Φ from* FJLT. *With probability at least* 2/3, *the following two events occur*:

1. *For any $x \in X$,*
$$(1-\varepsilon)\alpha_p \|x\|_2 \le \|\Phi x\|_p \le (1+\varepsilon)\alpha_p \|x\|_2,$$
where $\alpha_1 = k\sqrt{2\pi^{-1}}$ and $\alpha_2 = k$.
2. *The mapping Φ: $\mathbf{R}^d \to \mathbf{R}^k$ requires*
$$O(d\log d + \min\{d\varepsilon^{-2}\log n, \varepsilon^{p-4}\log^{p+1} n\})$$
operations.

Remark: By repeating the construction $O(\log(1/\delta))$ times we can ensure that the probability of failure drops to δ for any desired $\delta > 0$. By *failure* we mean that either the first or the second part of the lemma does not hold.

2.2. Showing that Φ works

We sketch a proof of the FJLT Lemma. Without loss of generality, we can assume that $\varepsilon < \varepsilon_0$ for some suitably small ε_0. Fix $x \in X$. The inequalities of the lemma are invariant under scaling, so we can assume that $\|x\|_2 = 1$. Consider the random variable $u = HDx$, denoted by $(u_1, \ldots, u_d)^T$. The first coordinate u_1 is of the form $\Sigma_{i=1}^d a_i x_i$, where each $a_i = \pm d^{-1/2}$ is chosen independently and uniformly. We can use a standard tail estimate technique to prove that, with probability at least, say, 0.95,

$$\max_{x \in X} \|HDx\|_\infty = O\left(d^{-1/2}\sqrt{\log n}\right). \quad (3)$$

It is important to intuitively understand what (3) means. Bounding $\|HDx\|_\infty$ is tantamount to bounding the magnitude of all coordinates of HDx. This can be directly translated to a *densification* property. To see why, consider an extreme case: If we knew that, say, $\|HDx\|_\infty < 1$, then we would automatically steer clear of the sparsest case possible, in which x is null in all but one coordinate (which would have to be 1 by the assumption $\|x\|_2 = \|HDx\|_2 = 1$).

To prove (3), we first make the following technical observation:

$$\mathbf{E}[e^{tdu_1}] = \prod_i \mathbf{E}[e^{tda_i x_i}] = \prod_i \cosh(t\sqrt{d}x_i) \le e^{t^2 d\|x\|_2^2/2}.$$

Setting $t = sd$ above, we now use the technical observation together with Markov's inequality to conclude that, for any $s > 0$,

$$\begin{aligned}\mathbf{Pr}[|u_1| \ge s] &= 2\mathbf{Pr}[e^{sdu_1} \ge e^{s^2 d}] \\ &\le 2\mathbf{E}[e^{sdu_1}] / e^{s^2 d} \le 2e^{s^2 d\|x\|_2^2/2 - s^2 d} \\ &= 2e^{-s^2 d/2} \le 1/(20nd),\end{aligned}$$

for $s = \Theta(d^{-1/2}\sqrt{\log n})$. A union bound over all $nd \le n^2$ coordinates of the vectors $\{HDx | x \in X\}$ leads to (3). We assume from now on that (3) holds with s as the upper bound; in other words, $\|u\|_\infty \le s$, where $u = HDx$. Assume now that u is fixed. It is convenient (and immaterial) to choose s so that $m \stackrel{\text{def}}{=} s^{-2}$ is an integer.

It can be shown that $\|u\|_2 = \|x\|_2$ by virtue of both H and D (and their composition) being isometries (i.e., preserve ℓ_2 norms). Now define,

$$y = (y_1, \ldots, y_k)^T = Pu = \Phi x.$$

The vector y is the final mapping of x using Φ. It is useful to consider each coordinate of y separately. All coordinates share the same distribution (though not as independent random variables). Consider y_1. By definition of FJLT, it is obtained as follows: Pick random i.i.d. indicator variables $b_1, \ldots, b_d$, where each b_j equals 1 with probability q; then draw random i.i.d. variables $r_1, \ldots, r_d$ from $N(0, 1/q)$. Set $y_1 = \sum_{j=1}^d r_j b_j u_j$ and let $Z = \sum_{j=1}^d b_j u_j^2$. It can be shown that the conditional variable $(y_1 | Z = z)$ is distributed $N(0, z/q)$ (this follows a well known fact known as the 2-stability of the normal distribution). Note that all of $y_1, \ldots, y_k$ are i.i.d. (given u), and we can similarly define corresponding random i.i.d. variables $Z_1 (= Z), Z_2, \ldots, Z_k$. It now follows that the expectation of Z satisfies:

$$\mathbf{E}[Z] = \sum_{j=1}^d u_j^2 \mathbf{E}[b_j] = q. \tag{4}$$

Let u^2 formally denote $(u_1^2, \ldots, u_d^2) \in (\mathbf{R}^+)^d$. By our assumption that (3) holds, u^2 lies in the d-dimensional polytope:

$$\mathcal{P} = \left\{(a_1, \ldots, a_d) : 0 \le a_j \le \frac{1}{m} \text{ and } \sum_{j=1}^d a_j = 1\right\}.$$

Let $u^* \in \mathbf{R}^d$ denote a vector such that u^{*2} is a vertex of $\mathcal{P}$. By symmetry of these vertices, there will be no loss of generality in what follows if we fix:

$$u^* = (\underbrace{m^{-1/2}, \ldots, m^{-1/2}}_{m}, \underbrace{0, \ldots, 0}_{d-m}).$$

The vector u^* will be convenient for identifying extremal cases in the analysis of Z. By extremal we mean the most *problematic* case, namely, the sparsest possible under assumption (3) (recall that the whole objective of *HD* was to alleviate sparseness).

We shall use Z^* to denote the random variable Z corresponding to the case $u = u^*$. We observe that $Z^* \sim m^{-1}B(m, q)$; in words, the binomial distribution with parameters m, q divided by the constant m. Consequently,

$$\mathbf{var}(Z^*) = q(1-q)/m. \tag{5}$$

In what follows, we divide our discussion between the ℓ_1 and the ℓ_2 cases.

The ℓ_1 case: We choose

$$q = \min\{1/(\varepsilon m), 1\} = \min\left\{\Theta\left(\frac{\varepsilon^{-1}\log n}{d}\right), 1\right\}.$$

We now bound the moments of Z over the random b_j's.

LEMMA 1. *For any $t > 1$, $\mathbf{E}[Z^t] = O(qt)^t$, and*

$$(1-\varepsilon)\sqrt{q} \le \mathbf{E}[\sqrt{Z}] \le \sqrt{q}.$$

Proof: The case $q = 1$ is trivial because Z is constant and equal to 1. So we assume $q = 1/(\varepsilon m) < 1$. It is easy to verify that $\mathbf{E}[Z^t]$ is a convex function of u^2, and hence achieves its maximum at a vertex of $\mathcal{P}$. So it suffices to prove the moment upper bounds for Z^*, which conveniently behaves like a (scaled) binomial. By standard bounds on the binomial moments,

$$\mathbf{E}[Z^{*t}] = O(m^{-t}(mqt)^t) = Q(qt)^t,$$

proving the first part of the lemma.

By Jensen's inequality and (4),

$$\mathbf{E}[\sqrt{Z}] \le \sqrt{\mathbf{E}[Z]} = \sqrt{q}.$$

This proves the upper-bound side of the second part of the lemma. To prove the lower-bound side, we notice that $\mathbf{E}[\sqrt{Z}]$ is a concave function of u^2, and hence achieves its minimum when $u = u^*$. So it suffices to prove the desired lower bound for $\mathbf{E}[\sqrt{Z^*}]$. Since $\sqrt{x} \ge 1 + \frac{1}{2}(x-1) - (x-1)^2$ for all $x \ge 0$,

$$\begin{aligned}\mathbf{E}\left[\sqrt{Z^*}\right] &= \sqrt{q}\,\mathbf{E}\left[\sqrt{Z^*/q}\right] \\ &\ge \sqrt{q}\left(1 + \frac{1}{2}\mathbf{E}[Z^*/q - 1] - \mathbf{E}[(Z^*/q - 1)^2]\right).\end{aligned} \tag{6}$$

By (4), $\mathbf{E}[Z^*/q - 1] = 0$ and, using (5),

$$\mathbf{E}[(Z^*/q-1)^2] = \mathbf{var}(Z^*/q) = (1-q)/(qm) \leq 1/(qm) = \varepsilon.$$

Plugging this into (6) shows that $\mathbf{E}\left[\sqrt{Z^*}\right] \geq \sqrt{q}(1-\varepsilon)$, as desired. ✹

Since the expectation of the absolute value of $N(0, 1)$ is $\sqrt{2\pi^{-1}}$, by taking conditional expectations, we find that

$$\mathbf{E}[|y_1|] = \sqrt{2/q\pi}\,\mathbf{E}[\sqrt{Z}].$$

On the other hand, by Lemma 1, we note that

$$(1-\varepsilon)\sqrt{2/\pi} \leq \mathbf{E}[|y_1|] \leq \sqrt{2/\pi}. \quad (7)$$

Next, we prove that $\|y\|_1$ is sharply concentrated around its mean $\mathbf{E}[\|y\|_1] = k\mathbf{E}[|y_1|]$. To do this, we begin by bounding the moments of $|y_1| = |\Sigma_j b_j r_j u_j|$. Using conditional expectations, we can show that, for any integer $t \geq 0$,

$$\mathbf{E}[|y_1|^t] = \mathbf{E}[(q^{-1}Z)^{t/2}]\mathbf{E}[|U|^t],$$

where $U \sim N(0,1)$. It is well known that $\mathbf{E}[|U|^t] = (t)^{t/2}$; and so, by Lemma 1,

$$\mathbf{E}[|y_1|^t] = O(t)^t.$$

It follows that the moment generating function satisfies

$$\mathbf{E}[e^{\lambda|y_1|}] = 1 + \lambda\,\mathbf{E}[|y_1|] + \sum_{t>1}\mathbf{E}[|y_1|^t]\lambda^t/t! \leq 1 + \lambda\,\mathbf{E}[|y_1|] + \sum_{t>1} O(t)^t\lambda^t/t!.$$

Therefore, it converges for any $0 \leq \lambda < \lambda_0$, where λ_0 is an absolute constant, and

$$\mathbf{E}[e^{\lambda|y_1|}] = 1 + \lambda\,\mathbf{E}[|y_1|] + O(\lambda^2) = e^{\lambda\mathbf{E}[|y_1|]+O(\lambda^2)}.$$

Using independence, we find that

$$\mathbf{E}[e^{\lambda\|y\|_1}] = (\mathbf{E}[e^{\lambda|y_1|}])^k = e^{\lambda\mathbf{E}[\|y\|_1]+O(\lambda^2 k)}.$$

Meanwhile, Markov's inequality and (7) imply that

$$\mathbf{Pr}[\|y\|_1 \geq (1+\varepsilon)\mathbf{E}[\|y\|_1] \leq \mathbf{E}[e^{\lambda\|y\|_1}]/e^{\lambda(1+\varepsilon)\mathbf{E}[\|y\|_1]} \leq e^{-\lambda\varepsilon\mathbf{E}[\|y\|_1]+O(\lambda^2 k)} \leq e^{-\Omega(\varepsilon^2 k)},$$

for some $\lambda = \Theta(\varepsilon)$. The constraint $\lambda < \lambda_0$ corresponds to ε being smaller than some absolute constant. The same argument leads to a similar lower tail estimate. Our choice of k ensures that, for any $x \in X$, $\|\Phi x\|_1 = \|y\|_1$ deviates from its mean by at most ε with probability at least 0.95. By (7), this implies that $k\mathbf{E}[|y_1|]$ is itself concentrated around $\alpha_1 = k\sqrt{2/\pi}$ with a relative error at most ε; rescaling ε by a constant factor and ensuring (3) proves the ℓ_1 claim of the first part of the FJLT lemma.

The ℓ_2 case: We set

$$q = \min\left\{\frac{c_1 \log^2 n}{d}, 1\right\},$$

for a large enough constant c_1.

LEMMA 2. *With probability at least* $1 - \frac{1}{20n}$,

1. $q/2 \leq Z_i \leq 2q$ *for all* $i = 1, \ldots, k$; *and*
2. $kq(1-\varepsilon) \leq \sum_{i=1}^k Z_i \leq kq(1+\varepsilon)$.

Proof: If $q = 1$ then Z is the constant q and the claim is trivial. Otherwise, $q = c_1 d^{-1} \log^2 n < 1$. For any real λ, the function

$$f_\lambda(u_1^2, \ldots, u_d^2) = \mathbf{E}\left[e^{\lambda Z}\right]$$

is convex, hence achieves its maximum at the vertices of the polytope $\mathcal{P}$ (same as in the proof of Lemma 1). As argued before, therefore, $\mathbf{E}[e^{\lambda Z}] \leq \mathbf{E}[e^{\lambda Z^*}]$. We conclude the proof of the first part with a union bound on standard tail estimates on the scaled binomial Z^* that we derive from bounds on its moment generating function $\mathbf{E}[e^{\lambda Z^*}]$ (e.g., Alon and Spencer[6]). For the second part, let $S = \Sigma_{i=1}^k Z_i$. Again, the moment generating function of S is bounded above by that of $S^* \sim m^{-1}B(mk, q)$—all Z_i's are distributed as Z^*—and the desired concentration bound follows. ✹

We assume from now on that the premise of Lemma 2 holds for all choices of $x \in X$. A union bound shows that this happens with probability of at least 0.95. For each $i = 1, \ldots, k$ the random variable $y_i^2 q / Z_i$ is distributed as χ^2 with one degree of freedom. It follows that, conditioned on Z_i, the expected value of y_i^2 is Z_i/q and the moment generating function of y_i^2 is

$$\mathbf{E}\left[e^{\lambda y_i^2}\right] = \left(1 - 2\lambda Z_i/q\right)^{-1/2}.$$

Given any $0 < \lambda < \lambda_0$, for fixed λ_0, for large enough ξ, the moment generating function converges and is equal to

$$\mathbf{E}\left[e^{\lambda y_i^2}\right] \leq e^{\lambda Z_i/q + \xi\lambda^2 (Z_i/q)^2}.$$

We use here the fact that $Z_i/q = O(1)$, which we derive from the first part of Lemma 2. By independence, therefore,

$$\mathbf{E}\left[e^{\lambda\sum_{i=1}^k y_i^2}\right] \leq e^{\lambda\sum_{i=1}^k (Z_i/q + \xi\lambda^2\sum_{i=1}^k (Z_i/q)^2)};$$

and hence

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{k} y_i^2 > (1+\varepsilon)\sum_{i=1}^{k} Z_i/q\right] &= \mathbf{Pr}\left[e^{\lambda\sum_{i=1}^{k} y_i^2} > e^{(1+\varepsilon)\lambda\sum_{i=1}^{k} Z_i/q}\right] \\ &\le \mathbf{E}\left[e^{\lambda\sum_{i=1}^{k} y_i^2}\right] / e^{(1+\varepsilon)\lambda\sum_{i=1}^{k} Z_i/q} \\ &\le e^{-\varepsilon\lambda\sum_{i=1}^{k} Z_i/q + \xi\lambda^2\sum_{i=1}^{k}(Z_i/q)^2}.\end{aligned} \quad (8)$$

If we plug

$$\lambda = \frac{\varepsilon\sum_{i=1}^{k}(Z_i/q)}{2\xi\sum_{i=1}^{k}(Z_i/q)^2}$$

into (8) and assume that ε is smaller than some global ε_0, we avoid convergence issues (Lemma 2). By that same lemma, we now conclude that

$$\mathbf{Pr}\left[\sum_{i=1}^{k} y_i^2 > (1+\varepsilon)k\right] \le e^{-\Omega(\varepsilon^2 k)}.$$

A similar technique can be used to bound the left tail estimate. We set $k = c\varepsilon^{-2}\log n$ for some large enough c and use a union bound, possibly rescaling ε, to conclude the ℓ_2 case of the first part of the FJLT lemma.

Running Time: The vector Dx requires $O(d)$ steps, since D is diagonal. Computing $H(Dx)$ takes $O(d \log d)$ time using the FFT for Walsh–Hadamard. Finally, computing $P(H\,Dx)$ requires $O(|P|)$ time, where $|P|$ is the number of nonzeros in P. This number is distributed in $B(nk, q)$. It is now immediate to verify that

$$\mathbf{E}[|P|] = O(\varepsilon^{p-4}\log^{p+1} n).$$

A Markov bound establishes the desired complexity of the FJLT. This concludes our sketch of the proof of the FJLT lemma. ✹

3. APPLICATIONS

3.1. Approximate nearest neighbor searching

Given a metric space (U, d_U) and a finite subset (database) $P \subseteq U$, the problem of *ε-approximate nearest neighbor* (ε-ANN) searching is to preprocess P so that, given a query $x \in U$, a point $p \in P$ satisfying

$$d_U(x,p) \le (1+\varepsilon)d_U(x,q), \quad \text{for all} \quad q \in P,$$

can be found efficiently. In other words, we are interested in a point p further from x by a factor at most $(1 + \varepsilon)$ of the distance to its nearest neighbor.

This problem has received considerable attention. There are two good reasons for this: (i) ANN boasts more applications than virtually any other geometric problem[23]; (ii) allowing a small error ε makes it possible to break the curse of dimensionality.[24, 27]

There is abundant literature on (approximate) nearest neighbor searching.[8–10, 12, 13, 15, 16, 19, 21–24, 26, 27, 33, 39, 40] The early solutions typically suffered from the curse of dimensionality, but the last decade has witnessed a flurry of new algorithms that "break the curse" (see Indyk[23] for a recent survey).

The first algorithms with query times of poly(d, log n) and polynomial storage (for fixed ε) were those of Indyk and Motwani[24] in the Euclidean space case, and Kushilevitz et al.[27] in the Hamming cube case. Using JL, Indyk et al. provide a query time of $O(\varepsilon^{-2}d \log n)$ with $n^{O(\varepsilon^{-2})}$ storage and preprocessing. A discrete variant of JL was used by Kushilevitz et al. in the Hamming cube case. We mention here that the dimension reduction overwhelms the running time of the two algorithms. In order to improve the running time in both cases, we used two main ideas in Ailon and Chazelle.[2] The first idea applied to the discrete case. It used an observation related to the algebraic structure of the discrete version of JL used in Kushilevitz et al.[27] to obtain a speedup in running time. This observation was only applicable in the discrete case, but suggested the intuitive idea that a faster JL should be possible in Euclidean space as well, thereby motivating the search for FJLT. Indeed, by a straightforward application in Indyk et al.'s algorithm (with $p = 1$), the running time would later be improved using FJLT to $O(d \log d + \varepsilon^{-3}\log^2 n)$. Notice the *additive* form of this last expression in some function $f = f(d)$ and $g = g(n, \varepsilon)$, instead of a *multiplicative* one.

3.2. Fast approximation of large matrices

Large matrices appear in virtually every corner of science. Exact algorithms for decomposing or solving for large matrices are often inhibitively expensive to perform. This may change given improvements in matrix multiplication technology, but it appears that we will have to rely on matrix approximation strategies for a while, at least in the general case. It turns out that FJLT and ideas inspired by it play an important role in recent developments.

We elaborate on an example from a recent solution of Sarlós[36] to the problem of ℓ_2 regression (least square fit of an overdetermined linear system). Prior to that work (and ours), Drineas et al.[18] showed that, by downsampling (choosing only a small subset and discarding the rest) from the set of equations of the linear regression, an approximate solution to the problem could be obtained by solving the downsampled problem, the size of which depends only on the dimension d of the original solution space. The difficulty with this method is that the downsampling distribution depends on norms of rows of the left-singular vector matrix of the original system. Computing this matrix is as hard as the original regression problem and requires $O(m^2d)$ operations, with m the number of equations. To make this solution more practical, Sarlós observed that multiplying the equation matrix on the left by the $m \times m$ orthogonal matrix HD (as defined above in the definition of FJLT) implicitly multiplies the left-singular vectors by HD as well. By an analysis similar to the one above, the resulting left-singular matrix can be shown to have almost uniform row norm. This allows use of Drineas et al.'s ideas with uniform sampling of the equations. Put together, these results imply the first $o(m^2d)$ running time solution for worst-case approximate ℓ_2 regression.

In a recent stream of papers, authors Liberty, Martinsson, Rokhlin, Tygert and Woolfe[28, 35, 38] design and analyze fast algorithms for low-dimensional approximation algorithms of matrices, and demonstrate their application to the evaluation of the SVD of numerically low-rank matrices. Their schemes are based on randomized transformations akin to FJLT.

4. BEYOND FJLT

The FJLT result gives rise to the following question: What is a lower bound, as a function of n, d and ε, on the complexity of computing a JL-like random linear mapping? By this we mean a mapping that distorts pairwise Euclidean distances among any set of n points in d dimension by at most $1 \pm \varepsilon$. The underlying model of computation can be chosen as a linear circuit,[32] manipulating complex-valued intermediates by either adding two or multiplying one by (random) constants, and designating n as input and $k = O(\varepsilon^{-2} \log n)$ as output (say, for $p = 2$). It is worth observing that any lower bound in $\Omega(\varepsilon^{-2} \log n \min\{d, \log^2 n\})$ would imply a similar lower bound on the complexity of computing a Fourier transform. Such bounds are known only in a very restricted model[31] where constants are of bounded magnitude.

As a particular case of interest, we note that, whenever $k = O(d^{1/3})$, the running time of FJLT is $O(d \log d)$. In a more recent paper, Ailon and Liberty[3] improved this bound and showed that it is possible to obtain a JL-like random mapping in time $O(d \log d)$ for $k = O(d^{1/2-\delta})$ and any $\delta > 0$. Their transformation borrows the idea of preconditioning a Fourier transform with a random diagonal matrix from FJLT, but uses it differently and takes advantage of stronger measure concentration bounds and tools from error correcting codes over fields of characteristic 2. The same authors together with Singer consider the following inverse problem[4]: Design randomized linear time computable transformations that require the mildest assumptions possible on data to ensure successful dimensionality reduction. C

References

1. Achlioptas, D. Database-friendly random projections: Johnson–Lindenstrauss with binary coins. *J. Comput. Syst. Sci. 66*, 4 (2003), 671–687.
2. Ailon, N., Chazelle, B. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. *SIAM J. Comput. 39*, 1 (2009), 302–322.
3. Ailon, N., Liberty, E. Fast dimension reduction using rademacher series on dual bch codes. *Discrete and Computational Geometry* (2008).
4. Ailon, N., Liberty, E., Singer, A. Dense fast random projections and lean walsh transforms. *APPROX-RANDOM*, 2008, 512–522.
5. Alon, N. Problems and results in extremal combinatorics–I. *Discrete Math. 273*, 1–3 (2003), 31–53.
6. Alon, N., Spencer, *J. The Probabilistic Method*. John Wiley, 2nd edition, 2000.
7. Arora, S. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM 45*, 5 (1998), 753–782.
8. Arya, S., Mount, D.M. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (Austin, TX, United States, 1993), 271–280.
9. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM 45*, 6 (1998), 891–923.
10. Bern, M.W. Approximate closest-point queries in high dimensions. *Inf. Process. Lett. 45*, 2 (1993), 95–99.
11. Bingham, E., Mannila, H. Random projection in dimensionality reduction: Applications to image and text data. In *Knowledge Discovery and Data Mining*, 2001, 245–250.
12. Borodin, A., Ostrovsky, R., Rabani, Y. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the 31st Annual Symposium on the Theory of Computing* (*STOC*) (1999), 312–321.
13. Chan, T.M. Approximate nearest neighbor queries revisited. *Discrete Comput. Geometry 20*, 3 (1998), 359–373.
14. Christofides, N. Worst-case analysis of a new heuristic for the travelling salesman problem. *Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh*, 1976, 388.
15. Clarkson, K.L. An algorithm for approximate closest-point queries. In *Proceedings of the 10th Annual ACM Symposium on Computational Geometry* (*SoCG*) (1994), 160–164.
16. Clarkson, K.L. Nearest neighbor queries in metric spaces. *Discrete Comput. Geometry 22*, 1 (1999), 63–93.
17. DasGupta, S., Gupta, A. An elementary proof of the Johnson–Lindenstrauss lemma. *Technical Report, UC Berkeley*, 1999, 99–106.
18. Drineas, P., Mahoney, M.W., Muthukrishnan, S. Sampling algorithms for ℓ_2 regression and applications. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (Miami, FL, United States, 2006).
19. Farach-Colton, M., Indyk, P. Approximate nearest neighbor algorithms for Hausdorff metrics via embeddings. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science* (*FOCS*) (1999), 171–180.
20. Frankl, P., Maehara, H. The Johnson–Lindenstrauss lemma and the sphericity of some graphs. *J. Comb. Theory Ser. A 44* (1987), 355–362.
21. Indyk, P. On approximate nearest neighbors in non-Euclidean spaces. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science* (*FOCS*) (1998), 148–155.
22. Indyk, P. Dimensionality reduction techniques for proximity problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (2000), 371–378.
23. Indyk, P. Nearest neighbors in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry*. CRC, 2004.
24. Indyk P., Motwani, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing* (*STOC*) (1998), 604–613.
25. Johnson, W.B., Lindenstrauss, J. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math. 26* (1984), 189–206.
26. Kleinberg, J.M. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (*STOC*) (1997), 599–608.
27. Kushilevitz, E., Ostrovsky, R., Rabani, Y. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput. 30*, 2 (2000), 457–474.
28. Liberty, E., Woolfe, F., Martinsson, P.-G., Rokhlin, V., Tygert, M. Randomized algorithms for the low-rank approximation of matrices. *Proc. Natl. Acad. Sci.* (*PNAS*) *104*, 51 (2007), 20167–20172.
29. Linial, N., London, E., Rabinovich, Y. The geometry of graphs and some of its algorithmic applications. *Combinatorica 15*, 2 (1995), 215–245.
30. Mitchell, J.S.B. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem. *SIAM J. Comput. 28*, 4 (1999), 1298–1309.
31. Morgenstern, J. Note on a lower bound on the linear complexity of the fast fourier transform. *J. ACM 20*, 2 (1973), 305–306.
32. Morgenstern, J. The linear complexity of computation. *J. ACM 22*, 2 (1975), 184–194.
33. Muthukrishnan, S., Sahinalp, S.C. Simple and practical sequence nearest neighbors with block operations. In *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching* (*CPM*) (2002), 262–278.
34. Papadimitriou, C., Raghavan, P., Tamaki, H., Vempala, S. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the 17th Annual Symposium of Database Systems* (1998), 159–168.
35. Rokhlin, V., Tygert, M. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Science* (*PNAS*) *105*, 36 (2008), 13212–13217.
36. Sarlós, T. Improved approximation algorithms for large matrices via random projections. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science* (*FOCS*) (Berkeley, CA, 2006).
37. Schulman, L. Clustering for edge-cost minimization. In *Proceedings of the 32nd Annual Symposium on Theory of Computing* (*STOC*) (2000), 547–555.
38. Woolfe, F., Liberty, E., Rokhlin, V., Tygert, M. A fast randomized algorithm for the approximation of matrices. *Appl. Comput. Harmon. Anal. 25* (2008), 335–366.
39. Yianilos, P.N. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (1993), 311–321.
40. Yianilos, P.N. Locally lifting the curse of dimensionality for nearest neighbor search (extended abstract). In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*) (2000), 361–370.

Nir Ailon (nailon@gmail.com), Google Research.

Bernard Chazelle (chazelle@cs.princeton.edu), Department of Computer Science, Princeton University.

DOI:10.1145/1646353.1646376

Technical Perspective
Want to be a Bug Buster?

By Shekhar Y. Borkar

MICROPROCESSOR PERFORMANCE HAS increased exponentially, made possible by increasing transistor performance and doubling the number of transistors every two years to realize complex architectures. These chips with ever increasing complexity are not always fully functional on the first attempt, they need to be debugged quickly, bugs fixed, and reworked. We take this for granted, but did you ever wonder about how it is done?

Painfully, is the short answer!

There are two types of bugs: *functional* or *logic* bugs caused by design errors, and electrical bugs due to circuit marginalities, caused by unfavorable operating conditions such as temperature changes, and voltage drops. Although most of the functional bugs are caught during rigorous design validation and verification, it is virtually impossible to ensure that a design is bug-free before tape-out.

Circuit designers make great efforts to improve margins to avoid electrical bugs, but they too are difficult to avoid, and especially difficult to reproduce since they are manifested by various operating conditions. It is extremely important to find these bugs quickly post-fabrication, and current techniques are far too expensive and time-consuming. The novel technique described in the following paper by Sung-Boem Park and Subhasish Mitra entitled "Post-Silicon Bug Localization for Processors Using IFRA" provides the breakthrough.

When an error is detected it could be caused by one or more such bugs, and you need to identify what caused the error; root causing the error is not that straightforward. The error may be caused by encountering a bug during an instruction execution, or it may be thousands, or even billions, of instruction executions before. You must be a very good detective to diagnose and isolate the bug.

Post-silicon bug localization and isolation is time-consuming and costly because you have to reproduce the failures by returning the hardware to an error-free state, activating the failure-causing stimuli, and then reproduce the same failures—the most difficult task, considering complexities such as asynchronous signals and multiple clock domains.

If you are a detective and want to catch bad guys, then one of the first steps is to acquire the surveillance video from the scene to get the clues. If the surveillance camera is rolling around the clock, you will likely be required to watch the entire tape before coming upon the culprit; very tedious indeed.

Wouldn't it be nice if the camera rolled only when sensing an action taking place? Yes, but it would still require watching every activity recorded by the camera and would still be a time-consuming and wasteful task. Better yet, what if the camera recorded only suspicious activity, capturing all the necessary circumstantial evidence, and not necessarily the culprits? That would be an optimal solution and would make the detective's job a lot easier. This scenario is exactly what IFRA does.

IFRA implements circular buffers, capable of recording traces of instructions executed by the processor, and this process is controlled by failure detectors. These detectors use failure detection in the hardware, such as parity errors as well as soft-triggers that suspect an early symptom of a failure. These triggers stop further recording in the circular buffer, capturing the instruction trace of the suspected part of the instruction sequence for future analysis.

IFRA implements circular buffers, capable of recording traces of instructions executed by the processor, and this process is controlled by failure detectors.

The traditional method of isolating a bug is by comparing the captured instruction trace with a golden trace captured by a trusted simulator. Simulators are notoriously slow and the process is very time-consuming. The authors of this paper propose a novel concept of self-consistency to localize the bug by examining the instruction trace. For example, if an instruction uses a faulty operand then you do not need to know the exact value of the operand. It is sufficient to know that the instruction used a different value than the one that was produced for its use. The authors describe in detail how to diagnose and root cause the bug using the captured instruction trace and self-consistency.

Clearly, this is a very novel approach to post-silicon debug, and I would not be surprised to see it catch on quickly; but it's just a start. This paper describes how to debug a microprocessor, and this technique has great potential to go further and help debug of multicores, memory systems, analog circuits, and even complex SOCs.

Finally, as a critique, you may claim that IFRA adds hardware to the chip that is useful only for debugging. Not really. Transistors are inexpensive, so inexpensive that it is almost like incorporating a small logic analyzer or a tester on the chip itself to aid in debugging, and then turned off; you don't even notice it is there.

Is there better use for a transistor than to help bring products to you quickly and inexpensively? C

Shekhar Y. Borkar is an Intel Fellow and Director of Microprocessor Research, Intel Corp., Hillsboro, OR.

DOI:10.1145/1646353.1646377

Post-Silicon Bug Localization for Processors Using IFRA

By Sung-Boem Park and Subhasish Mitra

Abstract

IFRA, an acronym for Instruction Footprint Recording and Analysis, overcomes major challenges associated with a very expensive step in post-silicon validation of processors—pinpointing a bug location and the instruction sequence that exposes the bug from a system failure, such as a crash. Special on-chip recorders, inserted in a processor during design, collect instruction footprints—special information about flows of instructions, and what the instructions did as they passed through various microarchitectural blocks of the processor. The recording is done concurrently during the normal operation of the processor in a post-silicon system validation setup. Upon detection of a system failure, the recorded information is scanned out and analyzed offline for bug localization. Special self-consistency-based program analysis techniques, together with the test-program binary of the application executed during post-silicon validation, are used for this purpose. Major benefits of using IFRA over traditional techniques for post-silicon bug localization are (1) it does not require full system-level reproduction of bugs, and (2) it does not require full system-level simulation. Hence, it can overcome major hurdles that limit the scalability of traditional post-silicon validation methodologies. Simulation results on a complex superscalar processor demonstrate that IFRA is effective in accurately localizing electrical bugs with 1% chip-level area impact.

1. INTRODUCTION

Post-Silicon validation involves operating one or more manufactured chips in actual application environments to validate correct behaviors across specified operating conditions. According to recent industry reports, post-silicon validation is becoming significantly expensive. Intel reported a headcount ratio of 3:1 for design vs. post-silicon validation.[19] According to Abramovici et al.,[1] post-silicon validation may consume 35% of average chip development time. Yerramilli[25] observes that post-silicon validation costs are rising faster than the design costs.

Loosely speaking, there are two types of bugs that design and validation engineers worry about:

1. Bugs caused by the interactions between the design and the physical effects, also called *electrical bugs*.[10] Such bugs generally manifest themselves only under certain operating conditions (temperature, voltage, frequency). Examples include setup and hold time problems.
2. *Functional bugs*, also called *logic bugs*, caused by design errors.

Post-silicon validation involves four steps:

1. Detecting a problem by running a test program, such as OS, games, or functional tests, until a system failure occurs (e.g., system crash, segmentation fault, or exceptions).
2. Localizing the problem to a small region from the system failure, e.g., a bug in an adder inside an ALU of a complex processor. The stimulus that exposes the bug, e.g., the particular 10 lines of code from some application, is also important.
3. Identifying the root cause of the problem. For example, an electrical bug may be caused by power-supply noise slowing down a circuit path resulting in an error at the adder output.
4. Fixing or bypassing the problem by microcode patching,[7] circuit editing,[11] or, as a last resort, respinning using a new mask.

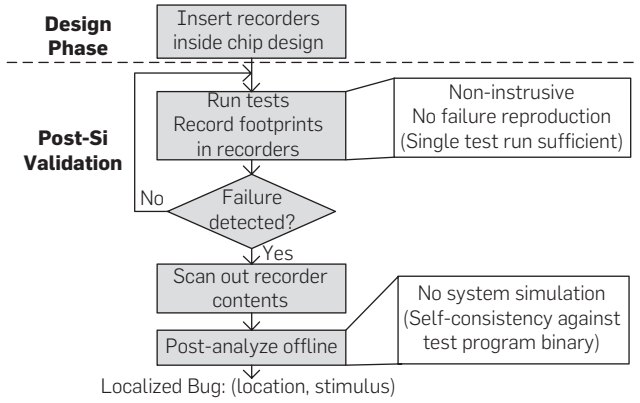
Josephson[9] points out that the second step, bug localization, dominates post-silicon validation effort and costs. Two major factors that contribute to the high cost of traditional post-silicon bug localization approaches are:

1. *Failure reproduction* which involves returning the chip to an error-free state, and re-executing the failure-causing stimulus (including test-program segment, interrupts, and operating conditions) to reproduce the same failure. Unfortunately, many electrical bugs are hard to reproduce. The difficulty of bug reproduction is exacerbated by the presence of asynchronous I/Os and multiple clock domains.
2. System-level simulation for obtaining golden responses, i.e., correct signal values for every clock cycle for the entire system (i.e., the chip and all the peripheral devices on the board) to compare against the signal values produced by the chip being validated. Running system-level simulation is typically 7–8 orders of magnitude slower than actual silicon.

Due to these factors, a functional bug typically takes hours to days to be localized vs. an electrical bug that requires days to weeks and more expensive equipments.[10]

A previous version of this paper appeared in the *Proceedings of the 45th ACM-IEEE Design Automation Conference* (2008, Anaheim, CA).

Figure 1. Post-silicon bug localization flow using IFRA.



IFRA, an acronym for Instruction Footprint Recording and Analysis, targets bug localization in processors. Figure 1 shows IFRA-based post-silicon bug localization flow. During chip design, a processor is augmented with low-cost hardware recorders (Section 2) for recording *instruction footprints*, which are compact pieces of information describing the flows of instructions (i.e., where each instruction was at various points of time), and what the instructions did as they passed through various design blocks of the processor. During post-silicon bug detection, instruction footprints are recorded in each recorder, concurrently with system operation, in a circular fashion to capture the last few thousand cycles of history before a failure.

Upon detection of a system failure, the recorded footprints are scanned out through a Boundary-scan interface, which is a standard interface present in most chips for testing purposes. Since a single run up to a failure is sufficient for IFRA to capture the necessary information (details in Section 2), failure reproduction is not required for localization purposes.

The scanned-out footprints, together with the test-program binary executed during post-silicon bug detection, are post-processed off-line using special analysis techniques (Section 3) to identify the microarchitectural block with the bug, and the instruction sequence that exposes the bug (i.e., the *bug exposing stimulus*). Microarchitectural block boundaries are defined specifically for IFRA. Examples include instruction queue control, scheduler, forwarding path, decoders, etc. IFRA post-analysis techniques do not require any system-level simulation because they rely on checking for self-consistencies in the footprints with respect to the test-program binary.

Once a bug is localized using IFRA, existing circuit-level debug techniques[4,9] can then quickly identify the root cause of bugs, resulting in significant gains in productivity, cost, and time-to-market.

In this paper, we demonstrate the effectiveness of IFRA for a DEC Alpha 21264-like superscalar processor model[6] because its architectural simulator[2] and RTL model[24] are publicly available. Such superscalar processors contain aggressive performance-enhancement features (e.g., execution of multiple instructions per cycle, execution of instructions out of program order, and prediction of branch targets and outcomes) that are present in many commercial high-performance processors.[22] Such features significantly complicate post-silicon validation. For simpler in-order processors (e.g., ARMv6, Intel Atom, SUN Niagra cores), IFRA can be significantly simplified.

There is little consensus about models of functional bugs.[8] Hence, we focus on electrical bugs that can be modeled as bit-flips (more details in Section 4). Extensive IFRA simulations demonstrate:

1. For 75% of injected electrical bugs, IFRA pinpointed their exact location (1 out of 200 microarchitectural blocks) and the time they were injected (1 out of over 1,000 cycles)—referred to as location–time pair. For 21% of injected bugs, IFRA correctly identified their location–time pairs together with 5 other candidates (out of over 200,000 possible pairs) on average. IFRA completely missed correct location–time pairs for only 4% of injected bugs.
2. The aforementioned results were obtained without relying on system-level simulation and failure reproduction.
3. IFRA hardware introduces a very small area impact of 1% (dominated by on-chip memory for storing 60KB of instruction footprints). If on-chip trace buffers[1] already exist for validation purposes, they can be reused to reduce the area impact. Alternatively, a part of data cache may also be used to reduce the area impact of IFRA.

Related work on post-silicon validation can be broadly classified as formal methods,[5] on-chip trace buffers for hardware debugging,[1] off-chip program and data tracing,[13] clock manipulation,[9] scan-aided techniques,[4] check-pointing with deterministic replay,[21] and online assertion checking.[1, 3] Table 1 presents a qualitative comparison of IFRA vs. existing post-silicon bug localization techniques. In Table 1, a technique is categorised as being intrusive if it can alter the functional/electrical behavior of the system which may prevent electrical bugs to get exposed.

Section 2 describes hardware support for IFRA. Section 3 describes off-line analysis techniques performed on the scanned-out instruction footprints. Section 4 presents simulation results, followed by conclusions in Section 5.

2. IFRA HARDWARE SUPPORT

The three hardware components of IFRA's recording infrastructure, for a superscalar processor, are indicated as shaded parts in Figure 2.

1. A set of distributed recorders, denoted by 'R' in Figure 2, with dedicated circular buffers. As an instruction passes through a pipeline stage, the recorder associated with that stage records information specific to that stage (Table 2). When no instruction passes through a pipeline stage for many cycles, consecutive idle cycles are compacted into a single entry in the corresponding recorder.

Table 1. IFRA vs. existing techniques.

Techniques	Formal Methods	Trace Buffer	Scan Methods	Clock Manipulation	Program and Data Tracing	Checkpoint and Replay	Assertion Checking	IFRA
Intrusive?	(+) No	Depends	(–) Yes		Depends	(–) Yes	Depends	(+) No
Failure reproduction?	(–) Yes						Depends	(+) No
System-level simulation?	(+) No	(–) Yes					(+) No	(+) No
Area impact?	(–) Yes		(+) No	(–) Yes	(+) No		(–) Yes	(–) 1%
Applicability?	(+) General				(–) Processor		Depends	(–) Processor

Figure 2. Superscalar processor augmented with recording infrastructure.

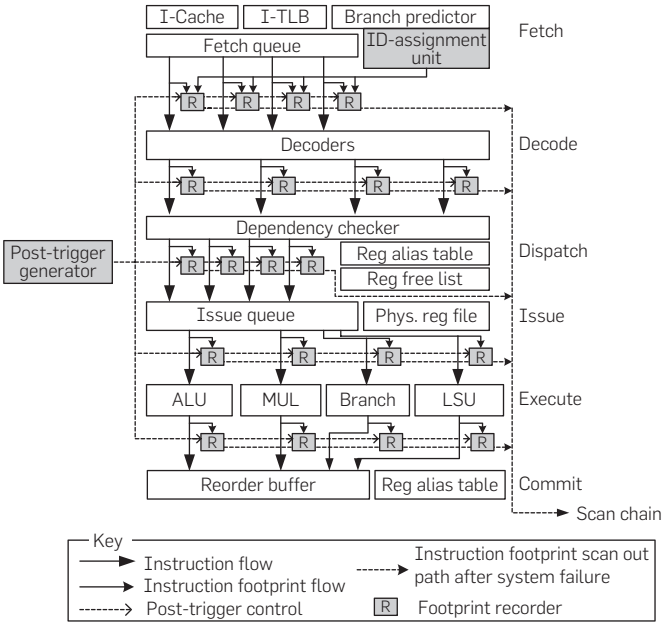


Table 2. Auxiliary information for each pipeline stage. The 2-bit and 3-bit residues are obtained by performing mod-3 and mod-7 operations on the original values, respectively.

	Auxiliary Information			
Pipeline Stage	Description	Bits per Entry	Number of Recorders	Entries per Recorder
Fetch	Program counter	32	4	1024
Decode	Decoded bits	4	4	1024
Dispatch	2-bit residue of register name	6	4	1024
Issue	3-bit residue of operands	6	4	1024
ALU, MUL	3-bit residue of result	3	4	1024
Branch	None	0	2	1024
LSU	3-bit residue of result; Memory address	35	2	1024
Commit	Fatal exceptions	4	1	1
Total storage required for all recorders: Each entry contains an additional 8-bit instruction ID (explained later).			60KB	

2. An ID (identification) assignment unit responsible for assigning and appending an ID to each instruction that enters the processor.
3. A post-trigger generator, which is a mechanism for deciding when to stop recording.

While an instruction, with an ID appended, flows through a pipeline stage, it generates an instruction footprint corresponding to that pipeline stage which is stored in the recorder associated with that pipeline stage. An instruction footprint corresponding to a pipeline stage consists of

1. The instruction's ID that was appended
2. *Auxiliary information* (Table 2) that tells us what the instruction did in the microarchitectural blocks contained in that pipeline stage

Synthesis results (using Synopsys Design Compiler with TSMC 0.13 microns library) show that the area impact of the IFRA hardware infrastructure is 1% on the Illinois Verilog Model[24] assuming a 2MB on-chip cache, which is typical of current desktop/server processors. The area cost is dominated by the circular buffers present in the recorders. Interconnect area cost is relatively low because the wires connecting the recorders (Figure 2) operate at slow speed, and a large portion of this routing reuses existing on-chip scan chains that are present for manufacturing testing purposes.

2.1. ID-assignment unit

For the recorded data to be useful for offline analysis, it is necessary to identify which of the trillions of instructions that passed through the processor, produced each of the recorded footprints. Hence, each footprint in a recorder must have an identifier or ID.

Simplistic ID assignment schemes have limited applicability. For example, assigning consecutive numbers to each incoming instruction, in a circular fashion, using very wide IDs is wasteful: using 40-bit IDs will increase the instruction footprint total storage to 160KB from 60KB. When IDs are too short, e.g., 8-bit IDs if there can be only 256 instructions in a processor at any one time, aliasing can occur for processors supporting out-of-order execution and pipeline

flushes (process of discarding instructions in the middle of execution to enforce a change in control flow). There can be multiple instructions with the same ID in a processor at any given time that may execute out of program order making it very difficult, if not impossible, to distinguish.

The PC (program counter) value cannot be used as an instruction ID for processors supporting out-of-order execution, because programs with loops may produce multiple instances of the same instruction with the same PC value. These multiple instances may execute out of program order.

It is difficult to use time-stamps or other global synchronization mechanisms as instruction IDs for processors supporting multiple clock domains and/or DVFS (dynamic voltage and frequency scaling) for power management.

Our special ID assignment scheme, described below, uses $\log_2 4n$ bits, where n is the maximum number of instructions in a processor at any one time (e.g., $n = 64$ for Alpha 21264). The first two rules assign consecutive numbers to incoming instructions and the third rule allows the scheme to work[18] under all the aforementioned circumstances: i.e., for processors supporting out-of-order execution, pipeline flushes, multiple clock domains and DVFS.

Instruction IDs are assigned to individual instructions as they exit the fetch stage and enter the decode stage. Since multiple instructions may exit the fetch stage in parallel at any given clock cycle, multiple IDs are assigned in parallel.

Instruction ID Assignment Scheme used by IFRA:

Rule 1: The first p instructions that exit the fetch stage in parallel are assigned IDs, 0, 1, 2, ..., $p - 1$.

Rule 2: Let ID X be the last ID that was assigned. If there are q instructions that exit the fetch stage in the current cycle in parallel, then q IDs, $X + 1 \pmod{4n}$, $X + 2 \pmod{4n}$, ..., $X + q \pmod{4n}$ are assigned to the q instructions.

Rule 3: If an instruction with ID Y causes a pipeline flush, then the ID X in Rule 2 is overwritten with the value of $Y + 2n \pmod{4n}$. As a result, ID of $Y + 2n + 1 \pmod{4n}$ is assigned to the first instruction that is fetched after the flush. The flush is caused either by a mispredicted branch or an exception.

2.2. Post-trigger generators

Suppose that a test program has been executing for billions of cycles and an electrical bug is exercised after 5 billion cycles from start. Moreover, suppose that the electrical bug causes a system crash after another 1 billion cycles (i.e., 6 billion cycles from the start). With limited storage, we are only interested in capturing the information around the time when the electrical bug is exercised. Hence, 5 billions of cycles worth of information before the bug occurrence may not be necessary. On the other hand, if we stop recording only after the system crashes, all the useful recorded information will be overwritten. Thus, we must incorporate mechanisms, referred to as *post-triggers*, for reducing *error detection latency*, the length of time between the appearance of an error caused by a bug and visible system failure.

Post-triggers targeting five different failure scenarios are listed in Table 2. A *hard post-trigger* fires when there is an evident sign of failure, and causes the processor operation to terminate. Classical hardware error detection techniques such as parity bits for arrays and residue codes for arithmetic units[20] as well as in-built exceptions, such as unimplemented instruction exceptions and arithmetic exceptions, belong to this category.

However, hard post-triggers mechanisms alone are not sufficient, e.g., two tricky scenarios described in the last two rows of Table 3. These two failure scenarios may be detected several millions of cycles after an error occurs, causing useful recorded information to be overwritten even with the existing error detection mechanisms. Hence, we introduce the notion of *soft post-triggers*.

A soft post-trigger fires when there is an early symptom of a possible failure. It causes the recording in all recorders to pause, but allows the processor to keep running. If a hard post-trigger for the failure corresponding to the symptom occurs within a pre-specified amount of time, the processor stops. If a hard post-trigger does not fire within the specified time, the recording resumes assuming that the symptom was false.

Segmentation fault (or segfault) requires OS handling and, hence, may take several millions of cycles to resolve. Null-pointer dereference is detected by adding simple hardware in the Load/Store unit. For other illegal memory accesses, TLB-miss is used as the soft post-trigger. If a segfault is not declared by the OS while servicing the TLB-miss, the recording is resumed on TLB-refill. On the other hand, if a segfault is returned, then a hard post-trigger is activated.

3. POST-ANALYSIS TECHNIQUES

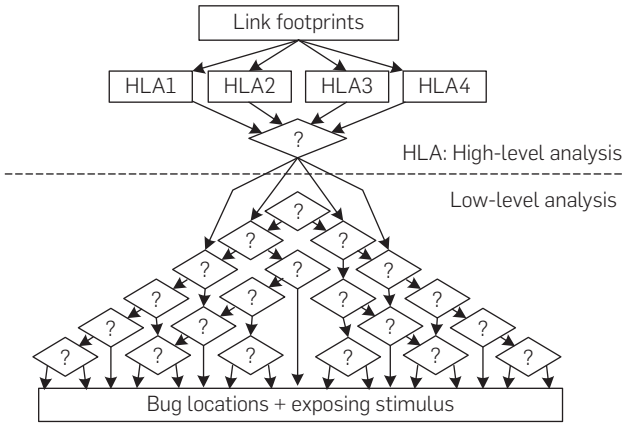
Once recorder contents are scanned out, footprints belonging to same instruction (but in multiple recorders) are identified and linked together using a technique called *footprint linking* (Section 3.1). The linked footprints are also mapped to the corresponding instruction in the test-program binary using the program counter value stored in the fetch-stage recorder (Table 2).

As shown in Figure 3, after the footprint linking, four high-level post-analysis techniques (Section 3.2) that are independent of microarchitecture are run. After which,

Table 3. Failure scenarios and post-triggers.

	Post-Triggers	
Failure Scenario	**Soft**	**Hard**
Array error	–	Parity check
Arithmetic error	–	Residue check
Fatal exceptions	–	In-built exceptions
Deadlock	Short (2 mem loads) instruction retirement gap	Long (2secs) instruction retirement gap[14]
Segfault	TLB-miss + TLB-refill	Segfault from OS; Address equals 0

Figure 3. Post-analysis summary: Park et al[18] describes the exact questions asked at each decision node.



low-level analysis (Section 3.3), represented as a decision diagram, asks a series of microarchitecture-specific questions until the final bug location–time pair(s) is obtained. The bug exposing stimuli are derived from the location–time pairs. Currently, the decision diagram is created manually based on the microarchitecture. Automatic generation of such decision diagrams is a topic of future research.

The post-analysis techniques rely on the concept of self-consistency which checks for the existence of contradictory events in collected footprints with respect to the test-program binary. While such checks are extensively used in fault-tolerant computing for error detection[12, 16, 23] the key difference here is that we use them for bug localization. Such application is possible because, unlike fault-tolerant computing, the checks are performed off-line enabling more complex analysis for localization purposes.

3.1. Footprint linking

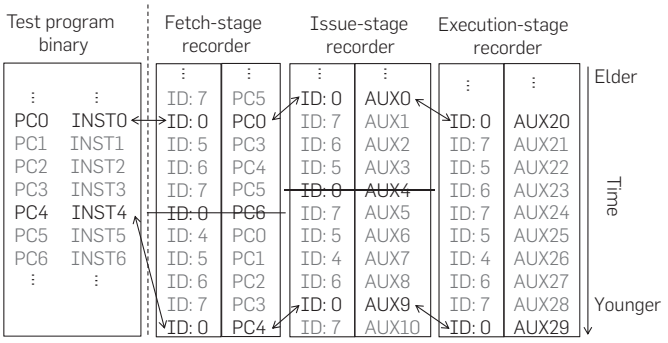
Figure 4 shows a part of a test program and the contents of three (out of many) recorders right after they are scanned out. As explained in Section 2, since we use short instruction IDs (8-bits for Alpha 21264-like processor), we end up having multiple footprints having the same ID in the same recorder and/or multiple recorders. For example, in Figure 4, ID 0 appears in three entries of the fetch-stage recorder, in two entries of the issue-stage recorder, and in three entries of the execution-stage recorder.

Which of these ID 0s correspond to the same instruction? This question is answered by the following special properties enforced by the ID assignment scheme presented in Section 2.1:

Property 1. All flushed instructions are identified by utilizing Rule 3 in our special ID assignment scheme (Section 2.1).

Property 2. If instruction A was fetched before instruction B, and they both have the same ID, then A will always exit any pipeline stage (and leave its footprint in the corresponding recorder) before B does for that same pipeline stage.

Figure 4. Instruction footprint linking, with a maximum number of 2 instructions in flight (i.e., $n = 2$).



In Figure 4, using the first property, footprints corresponding to flushed instructions are identified and discarded. After discarding, using the second property, the youngest ID 0s across all recorders are linked together, followed by linking of the second youngest ID 0s, and so on. Since the PC is stored in the fetch-stage recorder, we can link the instruction ID back to the test program binary to find the corresponding instruction.

3.2. High-level analysis

IFRA uses four high-level analysis techniques (1) data dependency analysis, (2) program control-flow analysis, (3) load-store analysis, and (4) decoding analysis.

Each analysis technique is applied separately. We are interested in the inconsistency that is closest to the electrical bug manifestation in terms of time (i.e., the eldest inconsistency). Thus, if multiple of them identify inconsistencies, then the reported inconsistencies are compared to see which one occurred the earliest. The high-level analysis technique with the earliest occurring inconsistency then decides the entry point into the decision diagram for low-level analysis. Here we briefly explain the control-flow analysis, one of the high-level analysis techniques, to illustrate the idea.

In the program control-flow analysis, four types of illegal transitions are searched in the PC sequence of the serial execution trace (obtained from fetch-stage recorder and test-program binary during footprint linking), starting from the eldest PC.

1. The PC does not increment by +4 except in the presence of a control flow transition instruction (e.g., branch, jump).
2. A PC jump does not occur in the presence of unconditional transition instruction.
3. The PC does not jump to the correct target in presence of direct transition (with target address that does not depend on a register value).
4. The PC does not jump to an address that is part of the executable address space (determined from the program binary) in the presence of register-indirect transition (with target address that depends on a register value).

If any illegal transition is found, the low-level analysis scrutinizes the PC register with the instruction that made an illegal transition.

3.3. Low-level analysis

The low-level analysis involves asking a series of microarchitectural-specific questions according to the decision diagram. We present a simple example by tracing one of the paths in the decision diagram.

Consider an example where a segfault (Section 2.2) during instruction access was detected, and the fourth illegal transition of the control-flow analysis was identified. We also assume that R5 shown in Figure 5 was the register used for the register-indirect transition. Instructions B and C have producer–consumer relationship: B writes its result in to register R0, and C uses a value from register R0.

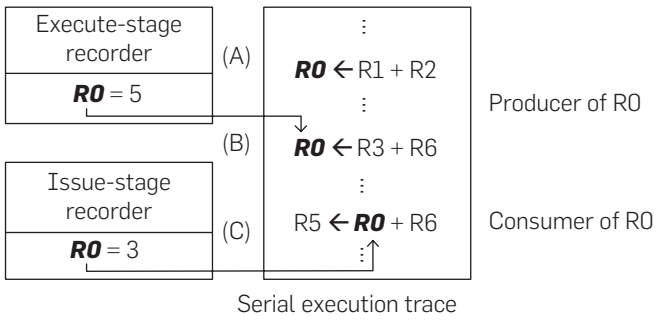
The first question in the decision diagram is whether C consumed the value B produced. The execute-stage recorder contains the residues of results and the issue-stage recorder contains the residues of operands of instructions. Comparing the two values during post-analysis shows that they do not match; i.e., B produced a value with residue of 5, while C received a value with residue of 3. This is clearly a problem.

The second question in the decision diagram is whether C and B used the same physical register to pass along the value. Analysis of the contents of the dispatch-stage recorder, which records the physical register name, reveals that B wrote its results into physical register P2, while C read its operand value from physical register P5, and they are not the same as shown in Figure 6.

There is again a problem, and the third question in the decision diagram asks whether C used a value produced by the previous producer (instruction that wrote its result into register R0 prior to the immediate producer) of register R0. Instruction A in Figure 7 is the previous producer of register R0 and analysis of the contents of the dispatch-stage recorder reveals that indeed that is the case.

Asking several more questions leads to the bug location and the exposing stimulus shown in Figure 8. The instruction trace between instruction A and instruction B is responsible for stimulating the bug, and the trace afterwards is responsible for propagating the bug to an observation point such as a soft post-trigger.

Figure 5. First question in the low-level analysis example: Did C consume the value B produced? Answer: No



4. RESULTS

We evaluated IFRA by injecting errors into a microarchitectural simulator[2] augmented with IFRA. For an Alpha 21264 configuration (4-way pipeline, 64 maximum instructions in-flight, 2 ALUs, 2 multipliers, 2 load/store

Figure 6. Second question asked in the low-level analysis example: Did C and B use the same physical register to pass along the value? Answer: No

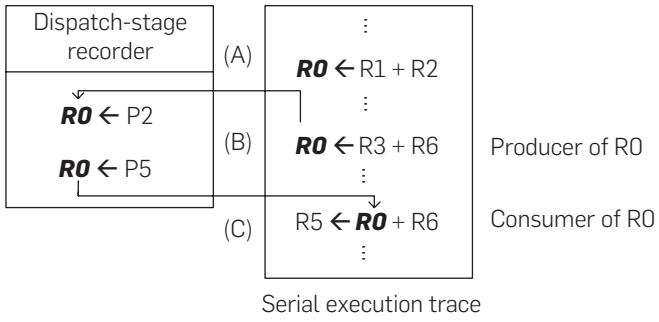


Figure 7. Third question asked in the low-level analysis example: Did C and A use the same physical register to pass along the value? Answer: Yes

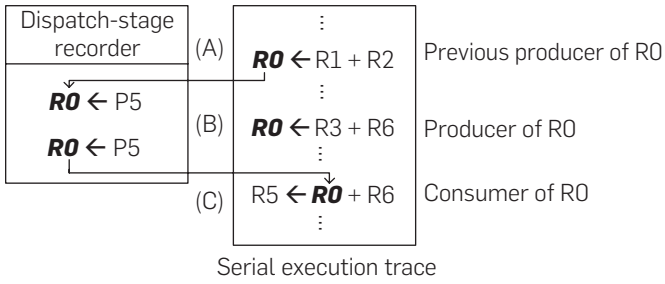
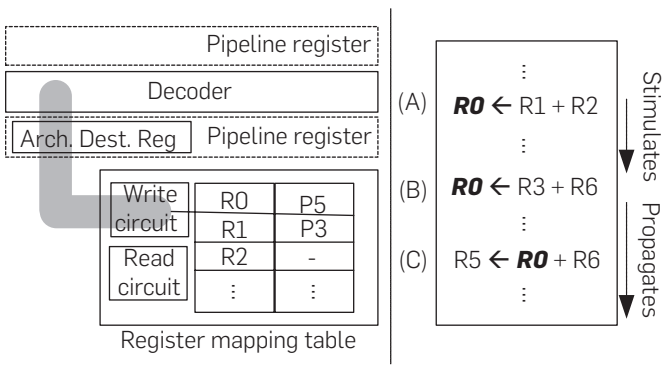


Figure 8. Bug location (enclosed in grey area – includes part of the decoder responsible for decoding the architectural destination register, the write circuitry into a register mapping table, and all the pipeline registers in between) shown on the left and the exposing stimulus shown on the right.



units), there are 200 different microarchitectural blocks (excluding array structures and arithmetic units since errors inside those structures are immediately detected and localized using parity and/or residue codes, as discussed in Section 2.2). Each block has an average size equivalent of 10K 2-input NAND gates. Seven benchmarks from SPECint2000 (bzip2, gcc, gap, gzip, mcf, parser, vortex) were chosen as validation test programs as they represent a variety of workloads. Each recorder was sized to have 1024 entries.

All bugs were modeled as single bit-flips at flip-flops to target hard-to-repeat electrical bugs. This is an effective model because electrical bugs eventually manifest themselves as incorrect values arriving at flip-fops for certain input combinations and operating conditions.[15]

Errors were injected in one of 1191 flip-flops [Park and Mitra[17]]. No errors were injected inside array structures since they have built-in parities for error detection.

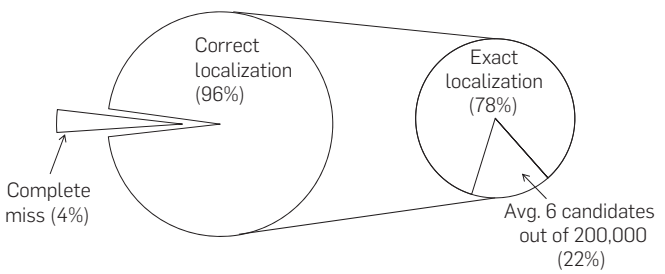
Upon error injection, the following scenarios are possible:

1. The error vanishes without any effect at the system level or produces an incorrect program output without any post-trigger firing. This case is related to the coverage of validation test programs and post-triggers, and is not the focus of this paper.
2. Failure manifestation with short error latency, where recorders successfully capture the history from error injection to failure manifestation (including situations where recording is stopped/paused upon activation of soft post-triggers).
3. Failure manifestation with long error latency, where 1024-entry recorders fail to capture the history from error injection to failure (including soft triggers).

Out of 100,000 error injection runs, 800 of them resulted in Cases 2 and 3. Figure 9 presents results from these two cases. The "*exactly located*" category represents the cases in which IFRA returned a single and correct location–time pair (as defined in Section 1). The "*candidate located*" category represents the cases in which IFRA returned multiple location–time pairs (called candidates) out of over 200,000 possible pairs (1 out of 200 microarchitectural blocks and 1 out of 1,000 cycles), and at least 1 pair was fully correct in both location and in time. The "*completely missed*" category represents the cases where none of the returned pairs were correct, even if either location or time is correct. In addition, we pessimistically report all errors that resulted in Case 3 as "completely missed." All error injections were performed after a million cycles from the beginning of the program in order to demonstrate that there is no need to keep track of footprints from the beginning.

It is clear from Figure 9 that a large percentage of bugs were uniquely located to correct location–time pair, while very few bugs were completely missed, demonstrating the effectiveness of IFRA.

Figure 9. IFRA bug localization summary.



5. CONCLUSION

IFRA targets the problem of post-silicon bug localization in a system setup, which is a major challenge in processor post-silicon design validation. There are two major novelties of IFRA:

1. High-level abstraction for bug localization using low-cost hardware recorders that record semantic information about instruction data and control flows concurrently in a system setup.
2. Special techniques, based on self-consistency, to analyze the recorded data for localization after failure detection.

IFRA overcomes major post-silicon bug localization challenges.

1. It helps bridge a major gap between system-level and circuit-level debug.
2. Failure reproduction is not required.
3. Self-consistency checks associated with the analysis techniques minimize the need for full system-level simulation.

IFRA creates several interesting research directions:

1. Automated construction of the post-analysis decision diagram for a given microarchitecture.
2. Sensitivity analysis and characterization of the interrelationships between post-analysis techniques, architectural features, error detection mechanisms, recorder sizes, and bug types.
3. Application to homogeneous/heterogeneous multi- and many-core systems, and system-on-chips (SoCs) consisting of nonprocessor designs.

Acknowledgment

The authors thank A. Bracy, B. Gottlieb, N. Hakim, D. Josephson, P. Patra, J. Stinson, H. Wang of Intel Corporation, O. Mutlu and S. Blanton of Carnegie Mellon University, T. Hong of Stanford University, and E. Rentschler of AMD for helpful discussions and advice. This research is supported in part by the Semiconductor Research Corporation and the National Science Foundation. Sung-Boem Park is also partially supported by Samsung Scholarship, formerly the Samsung Lee Kun Hee Scholarship Foundation.

References

1. Abramovici, M., et al. A reconfigurable design-for-debug infrastructure for SoCs. *Proc. Des. Automation Conf. July* (2006), 7–12.
2. Austin, T., et al. SimpleScalar: an infrastructure for computer system modeling. *Computer 35*, 2 (Feb. 2002), 56–67.
3. Bayazit, A.A., Malik, S. Complementary use of runtime validation and model checking. *Proc. Intl. Conf. Comput. Aided Des.* (2005), 1052–1059.
4. Caty, O., Dahlgren, P., Bayraktaroglu, I. Microprocessor silicon debug based on failure propagation tracing. *Proc. Intl. Test Conf. Nov.* (2005), 293–302.
5. De Paula, F.M., et al. BackSpace: formal analysis for post-silicon debug. *Proc. Formal Meth. Comput. Aided Des. Nov.* (2008), 1–10.
6. Digital Equipment Corporation, *Alpha 21264 Microprocessor Hardware Reference Manual*, July 1999.
7. Goddard, M.D., Christie, D.S. Microcode patching apparatus and method. U.S. Patent 5796974, Nov. 1995.
8. *International Technology Roadmap for Semiconductors*, 2007 ed.
9. Josephson, D. The good, the bad, and the ugly of silicon debug. *Proc. Des. Automation Conf. July* (2006), 3–6.
10. Josephson, D., Poehlman, S., Govan, V. Debug methodology for the McKinley processor. *Proc. Intl. Test Conf. Oct.–Nov.* (2001), 451–460.
11. Livengood, R.H., Medeiros, D. Design for (physical) debug for silicon microsurgery and probing of flip-chip packaged integrated circuits. *Proc. Intl. Test Conf. Sept.* (1999), 877–882.
12. Lu, D.J. Watchdog processors and structural integrity checking. *IEEE Trans. Comput. 31*, 7 (July 1982), 681–685.
13. MacNamee, C., Heffernan, D. Emerging on-chip debugging techniques for real-time embedded systems. *IEE Comput. Control Eng. J. 11*, 6 (Dec. 2000), 295–303.
14. Mahmood A., McCluskey, E.J. Concurrent error detection using watchdog processors—a survey. *IEEE Trans. Comput. 37*, 2 (Feb. 1988), 160–174.
15. McLaughlin R., Venkataraman, S., Lim, C. Automated debug of speed path failures using functional tests. *VLSI Test Symp. May* (2009), 91–96.
16. Oh, N., Shirvani, P.P., McCluskey, E.J. Control-flow checking by software signatures. *IEEE Trans. Reliability Mar.* (2002), 111–122.
17. Park S., Mitra, S. IFRA: instruction footprint recording and analysis for post-silicon bug localization in processors. *Proc. Des. Automation Conf. June* (2008), 373–378.
18. Park S., Hong, T., Mitra, S. Post-silicon bug localization in processors using instruction footprint recording and analysis (IFRA). *IEEE Trans. Comput. Aided Des. Integrated Circuits Syst. 28*, 10 (Oct. 2009), 1545–1558.
19. Patra, P. On the cusp of a validation wall. *IEEE Des. Test Comput. 24*, 2 (Mar. 2007), 193–196.
20. Sanda P.N., et al. Soft-error resilience of the IBM POWER6 Processor. *IBM J. Res. Dev. 52*, 3 (2008), 275–284.
21. Sarangi, S.R., Greskamp, B., Torrellas, J. CADRE: cycle-accurate deterministic replay for hardware debugging. *Intl. Conf. Dependable Syst. Netw. June* (2006), 301–312.
22. Shen, J.P., Lipasti, M.H. *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill, New York, 2005.
23. Siewiorek, D.P., Swarz, R.S. *Reliable Computer Systems—Design and Evaluation*, 3rd ed. A.K. Peters, Natick, 1998.
24. Wang, N.J., et al. Characterizing the effects of transient faults on a high-performance processor pipeline. *Proc. Intl. Conf. Dependable Syst. Netw. June–July* (2004), 61–70.
25. Yerramilli, S. Addressing post-silicon validation challenge: leverage validation & test synergy (invited address). *Intl. Test Conf.* Nov. (2006).

Sung-Boem Park Department of Electrical Engineering, Stanford University, Stanford, CA.

Subhasish Mitra Department of Electrical Engineering, Department of Computer Science, Stanford University, Stanford, CA.

CAREERS

Austin Peay State University
Assistant Professor/Computer Science

The Dept of Computer Science & Information Tech at Austin Peay State University invites applications for a tenure-track assistant professor position in computer science beginning August 2010. For more information, see http://bit.ly/674R4r

Carnegie Mellon University
Faculty Position in Design Research and Human Computer Interaction

Our world-class interdisciplinary Human-Computer Interaction Institute at Carnegie Mellon University expects to fill a tenure-track faculty position, starting August 2010. We are especially interested in an extraordinary faculty member who will advance our interdisciplinary research in design dimensions of human computer interaction.

Applicants should have the terminal degree in a discipline such as (but not limited to) design, architecture, computer science, psychology, HCI, cognitive science, or anthropology. We seek an outstanding educator and researcher who designs systems and who also implements systems and/or performs rigorous empirical laboratory or qualitative field studies. The candidate must be able to significantly advance research and theory in design and HCI. We will consider both junior and senior candidates. For more departmental information see www.hcii.cmu.edu.

Review of faculty applications will begin November 1, 2009, and will be accepted through January 1, 2010, or up until the position has been filled. Your application should include your CV, a statement of your research and teaching interests, pointers to a portfolio, 1-3 representative papers, and the names, positions, and email addresses of three or more individuals who may be asked to provide letters of reference. All applications should indicate citizenship and, in the case of non-US citizens, describe your current visa status. **Please send your application to Faculty Search Committee at hcii_facultysearch@cs.cmu.edu.**

Carnegie Mellon is an affirmative action/equal opportunity employer and we invite and encourage applications from women and minorities.

Coe College
Full-time Instructor

Coe College invites applications for a full-time one-year teaching appointment, with the possible extension to tenure-track, in computer science beginning in August of 2010. A masters degree in computer science or a related field is required (Ph.D. preferred). Candidates will be expected to demonstrate outstanding potential for teaching a wide range of undergraduate CS courses. Particular areas of interest include networking, software engineering, security, and operating systems. For more information, see http://www.coe.edu/aboutcoe/employment. Coe College is an Affirmative Action/Equal Opportunity Employer. Women and other underrepresented groups are particularly encouraged to apply.

Columbia College
Assistant/Associate Professor of Computer Science

Columbia College invites applications for a tenure-track assistant/associate professor position in computer science. Employment will begin August 2010. Applicants should be prepared to teach a broad range of undergraduate computer science courses. A Ph.D. in computer science is required and must be completed no later than August 2010. Normal teaching load is four courses (12 hours) each semester. Professional development, advising and service are expected.

Candidates must submit a letter of application, a current vita, statement of teaching philosophy, official graduate transcripts, three letters of reference and evidence of teaching effectiveness to: Executive Vice President and Dean for Academic Affairs, Columbia College, 1001 Rogers St., Columbia, MO 65216. AA/EOE. www.ccis.edu

Hawkes Learning Systems
Quality Assurance Engineer

Entry level position creating and using an automated testing system for our Quality Assurance and Development departments. Strong preference will be given to CS/Math double majors/minors. Salary and excellent benefits. Email resume to careers@hawkeslearning.com

National Taiwan University
Professor-Associate Professor-Assistant Professor

The Department of Computer Science and Information Engineering has faculty openings at all ranks beginning in August 2010. Highly qualified candidates in all areas of computer science/engineering are invited to apply. A Ph.D. or its equivalent is required. Applicants are expected to conduct outstanding research and be committed to teaching. Candidates should send a curriculum vitae, three letters of reference, and supporting materials before February 28, 2010, to Prof Kun-Mao Chao, Department of Computer Science and Information Engineering, National Taiwan University, No 1, Sec 4, Roosevelt Rd., Taipei 106, Taiwan.

National University of Singapore
School of Computing
Assistant/Associate/Full Professor

Applications are invited for tenure-track positions at the Assistant/Associate/Full Professor level. We are seeking outstanding candidates who are looking for new opportunities to advance their careers in the following areas:

- Computer Security
- Computer Graphics
- Games & Entertainment Technologies
- Computer Systems
- Networking
- Algorithms

NUS is a highly ranked research university with low teaching loads, excellent facilities, and intensive international collaboration. The Singapore government has recently earmarked over S$500 million for research and industrial projects focused on Digital Media and related areas. Significant funding opportunities abound for strong candidates. The School of Computing consists of active and talented faculty members working in a variety of areas, and attracts the best students (both undergraduate and graduate) in the region.

NUS offers highly competitive salaries, as well as generous benefits for housing and education. Singapore offers a vibrant international environment with low-taxes.

Review of applications will be immediate and will continue until June 30, 2010. Interested candidates are requested to send the following materials to csrec@comp.nus.edu.sg:

- Curriculum Vitae
- Research Statement
- Names of at least five references

Prince Mohammad Bin Fahd University
ARAMCO Endowed Chair in Technology and Information Management

Visit the complete description at:
http://www.pmu.edu.sa/cit/endowed.pdf
or contact Dr. Nassar Shaikh
(C)+966505820407; (O)+96638499300,
nshaikh@pmu.edu.sa

Vacancy open until filled. Review starts 26th October 2009.

The ideal candidate is a leader in technology and information management. Required is a doctorate degree in a closely related field; full professorship and significant experience in teaching, research and consultations; a record of publication and community outreach; experience as editor of a scholarly journal; international network and ability to attract participation in conferences held in Saudi Arabia.

Princeton University
Tenure-Track Assistant Professor

The Department of Computer Science at Princeton University invites applications for Assistant Professor. We are accepting applications in all areas of Computer Science, with particular emphasis on music, sound, and HCI.

Applications must demonstrate superior research and scholarship potential as well as teaching ability. A PhD or equivalent in Computer Science or related area is required. Successful candidates are expected to pursue an active re-

search program and to contribute significantly to the teaching programs of the department. Applicants should include a resume and the names of at least three people who can comment on the applicant's professional qualifications.

Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations. For general application information, how to self-identify, and to apply online please visit: https://jobs.cs.princeton.edu/

Rensselaer Polytechnic Institute

Department of Computer Science

Computer Science Postdoctoral Research Associates Positions

Postdoctoral Research Associates conduct advanced research, serve as a technical resource and may supervise grad students. Postdocs develop and maintain technical expertise required to perform original research both through self-direction and as part of a research team, and tailor research efforts to the needs of supporting groups from industry and/or government. The Rensselaer Polytechnic Institute Department of Computer Science has the following openings:

1. A 2 year position to accelerate the research aims of an NIH funded project. Expertise in image processing, linear algebra, graph theory, machine learning, and excellent system skills including programming is required. ***Please send application letter, CV and have three reference letters sent to cs-postdoc1@cs.rpi.edu.***
2. A 2 year position to accelerate the research aims of a DTRA funded project. Ability to model theoretically and simulate large scale networks and strong programming capabilities are required. Familiarity with social networks is a plus. ***Please send application letter, CV and have three reference letters sent to cs-postdoc2@cs.rpi.edu.***
3. A 2 or 3 year position to accelerate research aims of an ARL funded Social and Cognitive Network Academic Research Center. Ability to model theoretically and simulate large scale networks and strong programming capabilities are required. Familiarity with social networks, dynamic processes in networks and/or trust in networks is a plus. This position will spend a significant portion of his appointment at the research site of the Interdisciplinary Research Center at BBN Technologies in Boston, MA. ***Please send application letter, CV and have three reference letters sent to cs-postdoc3@cs.rpi.edu.***
4. A 3 year NIH funded Postdoctoral Fellowship, starting in spring 2010. A Ph.D. in mathematics or related field is required. Successful candidates should have expertise in stochastic models, optimization, numerical analysis and computation, finite element methods to work on a bioinformatics project. Basic understanding of biology is a plus. Candidate must have good programming skills and be familiar with matlab. ***Please send application letter, CV and have three reference letters sent to cs-postdoc4@cs.rpi.edu***

Candiates must have completed all Ph.D. requirements in the relevant field at the time of appointment.

We welcome candidates who will bring diverse intellectual, geographical, gender and ethnic perspectives to Rensselaer's work and campus communities. Rensselaer Polytechnic Institute is an Affirmative Action/Equal Opportunity Employer

Swarthmore College

Computer Science Department

Visiting Assistant Professor

Applications are invited for a two-year Visiting Assistant Professor position beginning August 2010.

Swarthmore College is a small, selective liberal arts college located in a suburb of Philadelphia. The Computer Science Department offers majors and minors in computer science at the undergraduate level. Applicants must have teaching experience and should be comfortable teaching a wide range of courses at the introductory and intermediate level. We are particularly interested in candidates who specialize in theory and algorithms or in systems areas, however, we will consider candidates from all areas of CS. A Ph.D. in CS by or near the time of appointment is preferred (ABD is required). We expect to begin interviewing in early February 2010.

See http://cs.swarthmore.edu/jobs for application submission information and more details about the position. Swarthmore College is an equal opportunity employer. Applications from women and members of minority groups are encouraged. Applications will be accepted until the position is filled.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Assistant Professorships (Tenure Track) in Computer Science

The Department of Computer Science (www.inf.ethz.ch) at ETH Zurich invites applications for assistant professorships (Tenure Track) in the areas of:

– Software Engineering
– Computer Graphics
– Computational Intelligence
– Human Computer Interaction

Applicants should have internationally recognized expertise in their field and pursue research at the forefront of Computer Science. The successful candidate should establish and lead a strong research program. He or she will be expected to supervise Ph.D. students and teach both undergraduate level courses (in German or English) and graduate level courses (in English).

The Department offers a stimulating and well-supported research and teaching environment. Collaboration in research and teaching is expected both within the Department and with other groups of the ETH domain and related institutions.

Assistant professorships have been established to promote the careers of younger scientists. The initial appointment is for four years with the possibility of renewal for an additional two-year period and promotion to a permanent position.

Please submit your curriculum vitae, a list of publications, names of at least three senior referees, and statements on future research and teaching activities **to the President of ETH Zurich, Prof. Dr. Ralph Eichler, ETH Zurich, Raemistrasse 101, 8092 Zurich, Switzerland, (or via e-mail to faculty-recruiting@sl.ethz.ch) no later than March 15, 2010.** With a view toward increasing the number of female professors, ETH Zurich specifically encourages qualified female candidates to apply.

Temple University

Tenure-Track, Open Rank

Applications are invited for a tenure-track, open rank, faculty position in the Department of Computer and Information Sciences at Temple University. Areas of interest include, but are not limited to Computer Systems, Wired and Wireless Networks, and Trustworthy and Reliable Computing. For senior rank candidates, applications should include curriculum vitae, a statement of recent achievements, and research, and teaching goals, up to three representative publications, and names and addresses of at least three references. Junior candidates should have three reference letters sent directly. Please submit applications online at http://academicjobsonline.org. For further information check http://www.cis.temple.edu. Review of candidates will begin on February 1, 2010 and will continue until the position is filled. Temple University is an equal opportunity, equal access, affirmative action employer.

Texas Christian University

Tenure-track position as Assistant/ Associate Professor

The Department of Computer Science at Texas Christian University seeks candidates at the Assistant or Associate Professor level for a tenure-track position in Computer Science to start on or after August 1, 2010. Applicants are expected to have an earned doctorate in Computer Science or related field, must have excellent verbal and written communication skills, and a strong commitment to teaching and

research. Responsibilities include teaching undergraduate computing courses in the Department's programs in Computer Information Technology and Computer Science; advising and mentoring majors; developing curricula in both programs; conducting research and engaged scholarship in computing fields; serving on college and university committees; and serving the computing community of educators, researchers, and professionals. Review of applications will begin immediately and continue until the position is filled. Salary is commensurate with qualifications and experience.

Interested candidates should submit a letter of application, curriculum vitae, a statement about teaching and research philosophy, and contact information for three persons from whom letters of recommendation can be requested. Please send information to:

Computer Science Search Committee
Department of Computer Science
TCU Box 298850
Texas Christian University
Ft. Worth, TX 76129

TCU is an EEO/AA employer. Women and minorities are encouraged to apply.

Toyota Motor Engineering and Manufacturing North America, Inc.

Senior Research/Principal Scientist & Research Engineer/Scientist

Senior Researcher:

▸ Apply special knowledge & talents to develop & execute new, independent research projects for robotic and mobile platform applications.

▸ Provide high-quality deliverables such as research/quick prototyping software, written & oral reports, IP, publications for peer-reviewed journals and conferences.

▸ Experience in artificial Intelligence, intelligent signal processing & sensor-fusion research.

▸ Experience in robotic research and testing.

▸ Ph.D. in a related field of study, preferably with 3 - 8 years of research experience.

▸ Strong programming skills (C/C++, Unix).

Researcher position:

▸ Apply special knowledge & talents to address outstanding research & development challenges in mid & long-term ITS (Intelligent Transportation Systems), robotics, and computer vision projects.

▸ Write dedicated software, design & carry out experiments on mobile platforms.

▸ Experience in computer vision and perception with one or more sensor modalities

▸ Good programming skills are essential.

▸ M.S. (Electrical/Computer Engineering or Computer Science) or Ph.D. in a related field of study.

Apply URL: http://www.toyota.com/toyota/about/jobs/JobsSearch.do

University of Campinas (UNICAMP)

Tenured Professor

The Faculty of Technology of UNICAMP invites applications for one tenured faculty position starting in early 2011. All areas of Computer Science and Engineer will be considered, but preference will be given to candidates with interest in programming language, discrete mathematics, computational modelling and graph theory. Successful candidates must have a strong commitment to academic excellence and teaching, and interest to establish and lead an independent and creative research group. Start-up resources and research infra-structure will be available.

Applicants should hold a PhD degree in Computer Science/Engineering or a closely related field, have excellent research and teaching record and leadership skills. Screening will begin in January, 1st 2010 and will continue until the position is filled. To apply please send (PDF only) curriculum vitae, including publication list, brief statements of research and teaching to info.ftecnol@reitoria.unicamp.br.

UNICAMP is considered one of the best research universities of Brazil. It has 33 thousand enrolled students, 48% of which are graduate students from 135 research programs, ranging from Music to Molecular Biology. UNICAMP is responsible for 15% of the research published in Brazil. The university is also responsible for the innovations that led to the development of some major technologies in Brazil like those in its fiber optics and telecommunication networks and biomedical industries. UNICAMP is located in Campinas, a well-known

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Dean
of Computer and Communication Sciences
at Ecole polytechnique fédérale de Lausanne (EPFL)

EPFL is conducting an international search for the Dean of the School of Computer and Communication Sciences, to take office by the fall of 2010.

EPFL, located in Lausanne (Switzerland), is a leading European University and a dynamically growing and well-funded institution fostering excellence and diversity. It has a highly international campus at an exceptionally attractive location and a first-class infrastructure. As technical university it covers computer & communication sciences, engineering, environmental, basic and life sciences, management of technology and financial engineering. It offers a fertile environment for research cooperation between different disciplines.

The School of Computer and Communication Sciences, with 42 faculty members, has experienced a strong development over the recent years to one of the top departments in its area in Europe. The School enrolls about 700 students in its bachelor and master programs in computer science and communication systems, has a highly competitive doctoral program with 300 PhD candidates recruited world-wide and hosts important industrial and research centers, such as the Swiss National Center of Excellence in Research in Mobile Information and Communication Systems and industry lablets by Nokia, SwissCom, and Logitech.

The Dean bears the overall responsibility for the school in matters of education, research, finance and organization and reports to the President of EPFL. The position offers competitive compensation and tenure at full professor level. Candidates should have an outstanding academic record, a strong vision for the development of the faculty in research, teaching, and technology transfer, proficiency in recruiting, and exceptional leadership, communication and management skills. EPFL will provide the means to realize a strategic development of the school over the coming years with the objective to establish world-class leadership in education and research.

The School of Computer and Communication Sciences Dean Search Committee invites letters of nomination, applications (vision statement, complete CV, and the name of up to 5 professional references), or expressions of interest. The screening of applications will start on **March 1st**, **2010**. Materials and inquiries should be addressed, preferably electronically (PDF format) to:

Prof. Karl Aberer
Chairman of the Search Committee
e-mail: karl.aberer@epfl.ch

More information on EPFL and the School of Computer and Communication Sciences can be found at **http://www.epfl.ch** and **http://ic.epfl.ch** respectively.

EPFL is committed to balance genders within its faculty, and strongly encourages women to apply.

technology town of Brazil, in the State of São Paulo, which accounts for 40% of Brazil's GNP.

University of Massachusetts Amherst

Department of Computer Science
Faculty Positions in Computer Science

The University of Massachusetts Amherst invites applications for two tenure-track faculty positions at the assistant professor level. Applicants must have a Ph.D. in Computer Science or related area and should show evidence of exceptional research promise.

For the first position, we seek Computer Science candidates able to collaborate with the departments of Linguistics and Psychology. The applicant should have a strong background in Natural Language Processing, preferably in the area of Syntax and Semantics. The applicant should also show a strong record of publication in Natural Language Processing and other areas that relate to Linguistics and Psychology, such as Machine Learning, Data Mining, and Information Retrieval. A history of interdepartmental collaboration is desirable but not required. (R36783)

For the second position, we seek Computer Science candidates that can collaborate with the departments of Sociology, Political Science, and Mathematics & Statistics. We seek a candidate in the area of Computational Social Science, or related areas of applied and theoretical computer science that could be relevant to the social sciences, such as social network analysis. (R36785)

The department is committed to the development of a diverse faculty and student body, and is very supportive of junior faculty, providing both formal and informal mentoring. We have a strong record of NSF CAREER awards and other early research funding. We lead the NSF-funded Commonwealth Alliance for Information Technology Education (CAITE) to design and carry out comprehensive programs that address underrepresentation in information technology (IT) education and the workforce.

The Department of Computer Science has 40 tenure and research track faculty and 180 Ph.D. students with broad interdisciplinary research interests. The department offers first-class research facilities. Please see http://www.cs.umass.edu for more information. The University provides an intellectual environment committed to providing academic excellence and diversity including mentoring programs for faculty. The College and the Department are committed to increasing the diversity of the faculty, student body and the curriculum. To apply, please send a cover letter referencing search R36783 or R36785 with your vita, a research statement, a teaching statement and at least three letters of recommendation.

We also invite applications for Research Faculty (R36769) and Research Scientist, Postdoctoral Research Associate, and Research Fellow (R36768) positions in all areas of Computer Science. Applicants should have a Ph.D. in Computer Science or related area (or an M.S. plus equivalent experience), and should show evidence of exceptional research promise. These positions are grant-funded; appointments will be contingent upon continued funding. To apply, please send a cover letter with your vita, a research statement and at least three letters of recommendation.

Electronic submission of application materials is recommended. Application materials may be submitted in pdf format to facrec@cs.umass.edu. Hard copies of the application materials may be sent to: Search {fill in number from above}, c/o Chair of Faculty Recruiting, Department of Computer Science, University of Massachusetts, Amherst, MA 01003.

We will begin to review applications on January 4, 2010 and will continue until available positions are filled. Salary and rank commensurate with education and experience; comprehensive benefits package. Positions to be filled dependent upon funding. Inquiries and requests for more information can be sent to: facrec@cs.umass.edu

The University of Massachusetts is an Affirmative Action/Equal Opportunity employer. Women and members of minority groups are encouraged to apply.

University of Toronto

Assistant Professor - Computer Science

The Department of Computer and Mathematical Sciences, University of Toronto Scarborough (UTSC), and the Graduate Department of Computer Science, University of Toronto, invite applications for a tenure-stream appointment at the rank of Assistant Professor, to begin July 1, 2010.

We are interested in candidates with research expertise in Computer Systems, including Operating Systems, Networks, Distributed Systems, Database Systems, Computer Architecture, Programming Languages, and Software Engineering.

Candidates should have, or be about to re-

Faculty Positions
Department of Software Design and Management
Kyungwon University

The Department of Software Design and Management at Kyungwon University in South Korea invites applications for a tenure-track position at the assistant professor or associate professor level.

Kyungwon University is located at Seongnam near Seoul. Further information about the university can be obtained at *http://www.kyungwon.ac.kr/english*.

The Department of Software Design and Management is a new department within the IT College. It is to launch in March 2010. It aims to become one of the world's top institutes for undergraduate software education. Dr. Won Kim, a world-renown pioneer in object-oriented and object-relational database technology, has joined Kyungwon University as IT Vice President and a lifetime professor to create, launch and grow the Department.

The Department seeks qualified candidates in one or more of the following areas:

- software engineering and architecture
- intelligent multimedia processing
- computer networking and communication

Applicant should have a strong passion for teaching, a Ph.D. degree in computer science from a reputable U.S. university, experience in teaching undergraduate computer science courses, and strong publication records.

How to Apply: Send a resume and cover letter, along with three letters of reference to *affairs@kyungwon.ac.kr*.

The Faculty of Engineering of the University of Freiburg, with its Departments of Computer Science and Microsystems Engineering, invites applications for the position of a

Full Professor (W3) of Computer Science

The successful candidate will be expected to establish a comprehensive research and teaching program in the area of algorithms and complexity with specialisation either in cryptography, optimization or parallel computing.

The University of Freiburg aims to increase the representation of women in research and teaching, and therefore expressly encourages women with suitable qualifications to apply for the post.

Information about the Department of Computer Science can be obtained from *www.informatik.uni-freiburg.de*.

Applications, including a curriculum vitae, publications list and statement of research interests should be sent by March 8, 2010 to the Dean of the Faculty of Engineering, University of Freiburg, Georges-Koehler-Allee 101, 79110 Freiburg, Germany (*www.tf.uni-freiburg.de*). Please send an electronic version of your application to the Dean's office and ask for our application form (*dekanat@tf.uni-freiburg.de*).

ceive, a Ph.D. in computer science or a related field. They must demonstrate an ability to pursue innovative research at the highest level, and a commitment to undergraduate and graduate teaching. Evidence of excellence in teaching and research is necessary. Salary will be commensurate with qualifications and experience.

The University of Toronto is an international leader in computer science research and education, and the Department of Computer and Mathematical Sciences enjoys strong ties to other units within the University. The successful candidate for this position will be expected to participate actively in the Graduate Department of Computer Science at the University of Toronto, as well as to contribute to the enrichment of computer science academic programs at the University's Scarborough campus.

Application materials, including curriculum vitae, research statement, teaching statement, and three to five letters of recommendation, should be submitted online at www.mathjobs.org, preferably well before our deadline of January 17, 2010.

PLEASE NOTE THAT WE ARE ONLY ACCEPTING APPLICATIONS AT: www.mathjob.org

For more information about the Department of Computer & Mathematical Sciences @ UTSC, please visit our home page www.utsc.utoronto.ca/~csms.

University of Toronto
Lecturer - Computer Science

The Department of Computer and Mathematical Sciences, University of Toronto Scarborough (UTSC), invites applications for a full-time position in Computer Science at the rank of Lecturer, to begin July 1, 2010.

We are especially interested in candidates who will help advance our curriculum in the areas of computer systems, computer architecture, and software engineering.

Appointments at the rank of Lecturer may be renewed annually to a maximum of five years. In the fifth year of service, Lecturers shall be reviewed and a recommendation made with respect to promotion to the rank of Senior Lecturer.

Responsibilities include lecturing, conducting tutorials, grading, and curriculum development in a variety of undergraduate courses.

Candidates should have a post-graduate degree, preferably a PhD, in Computer Science or a related field, and must demonstrate potential for excellence in teaching at the undergraduate level.

Salary will be commensurate with qualifications and experience.

Application materials, including curriculum vitae, a statement of career goals and teaching philosophy, evidence of teaching excellence, and a minimum of three reference letters should be submitted online at: www.mathjobs.org, preferably well before our deadline of March 1, 2010.

PLEASE NOTE THAT WE ARE ONLY ACCEPTING APPLICATIONS AT: www.mathjobs.org

For more information about the Department of Computer & Mathematical Sciences @ UTSC, please visit our home at: www.utsc.utoronto.ca/~csms

University of Toronto
Mendelzon Visiting Assistant Professor

The Department of Computer and Mathematical Sciences, University of Toronto Scarborough invites applications for a non-tenure-stream, two-year appointment as the Mendelzon Visiting Assistant Professor, to begin July 1, 2010.

We will consider applicants in all areas of computer science, but are especially interested in applicants who will help advance our curriculum in computer systems and software engineering.

The University of Toronto is an international leader in computer science research and education, and the Department of Computer and Mathematical Sciences enjoys strong ties to other units within the University.

The successful candidate for this position will be encouraged to engage in collaborative research with other computer science faculty at the university, as well as to contribute to the enrichment of computer science academic programs at the University's Scarborough campus.

Candidates should have, or be about to receive, a Ph.D. in computer science or a related field. They must demonstrate an ability to pursue innovative research, and a commitment to undergraduate teaching.

Application materials, including curriculum vitae, research statement, teaching statement, and three to five letters of recommendation, should be submitted online at www.mathjobs.org, preferably well before our deadline of January 17, 2010.

The University of Toronto is strongly committed to diversity within its community and especially welcomes applications from visible minority group members, women, Aboriginal persons, persons with disabilities, members of sexual minority groups, and others who may contribute to the further diversification of ideas. All qualified candidates are encouraged to apply; however, Canadians and permanent residents will be given priority.

The Mendelzon Visiting Assistant Professorship is a position created in memory of Alberto Mendelzon, FRSC, distinguished computer scientist, and former chair of the Department of Computer and Mathematical Sciences, University of Toronto Scarborough.

PLEASE NOTE THAT WE ARE ONLY ACCEPTING APPLICATIONS AT: www.mathjobs.org

For more information about the Department of Computer & Mathematical Sciences @ UTSC, please visit our home page: www.utsc.utoronto.ca/~csms

DOI:10.1145/1646353.1646380

Peter Winkler

Puzzled
Breaking Chocolate Bars

Welcome to three new puzzles. Solutions to the first two will be published next month; the third is (as yet) unsolved. In each, the issue is how your intuition matches up with the mathematics.

1. Charlie needs to break up chocolate bars in the process of baking s'mores for his children and their friends. Each bar is a rectangle scored into a five-by-nine array of squares. To break a bar into small squares, Charlie repeatedly picks up a single piece of chocolate and cracks it along a line (see Figure 1).

Charlie breaks up the first bar by first cracking it along its longest lines, resulting in five strips of nine squares each. He then dismantles each strip, square by square, for a total of $4 + 5 \times 8 = 44$ breaks. He breaks up the second bar the opposite way, first into eight strips of length five, then breaking up these strips for a total of $8 + 9 \times 4 = 44$ breaks, again. Darn. Can Charlie do better? What's the smallest number of breaks Charlie needs to reduce a chocolate bar to its constituent squares?

2. Charlie's children, Alice and Bobby, steal one of the bars and agree to play the following game: They place the bar on the table with its rows of nine squares aligned east-west (see Figure 2). Alice chooses any square and eats it, along with all the squares to the northeast (including straight north or east) of the chosen square. Bobby then chooses some remaining square and eats it, again along with all remaining squares to the northeast. The two then continue to alternate until the bar is completely gone.

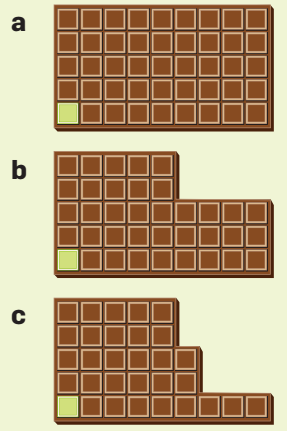
Alice could have started with the square at the southwest corner of the bar and thus consumed the whole bar in her first turn. But that square happens to be rotten. The object of the game is to force your opponent to eat the last square.

Prove that Alice indeed has a winning strategy.

3. What strategies enable Alice to win the game on any rectangular chocolate bar with more than a single square of chocolate?

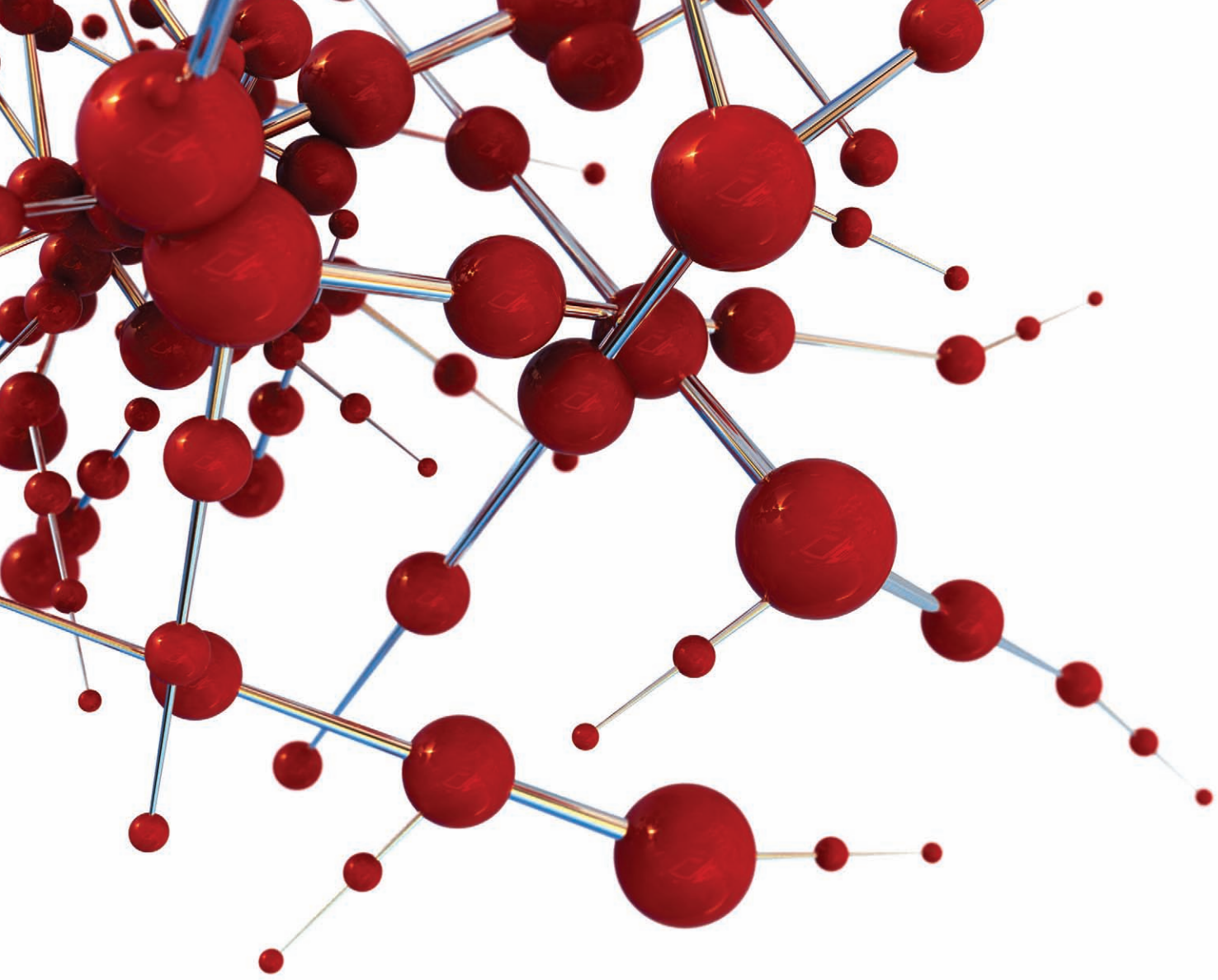
Figure 1. Charlie's first bar at the beginning, after four breaks, and after 12 breaks.

Figure 2. A possible opening move for Alice, and reply by Bobby.



Readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is Professor of Mathematics and of Computer Science and Albert Bradley Third Century Professor in the Sciences at Dartmouth College, Hanover, NH.



25TH ANNUAL ACM CONFERENCE ON SYSTEMS, PROGRAMMING, LANGUAGES, APPLICATIONS: SOFTWARE FOR HUMANITY

ACM SIGPLAN/SIGSOFT
announce
OOPSLA
is now part of
SPLASH

SPLASH

RENO 2010
OCTOBER 17-21

MARCH 25, 2010
Submission deadline for
OOPSLA Research Papers,
Practitioner Reports,
Educators' and Trainers' Symposium,
and proposals for Tutorials,
Workshops, Panels

APRIL 23, 2010
Onward! Papers and Essays

JUNE 24, 2010
Submission deadline for Posters,
Demonstrations, Doctoral Symposium,
Onward! Films, and Student Research
Competition and Volunteers

LOCATION
John Ascuaga's Nugget Hotel
Reno/Tahoe Nevada USA

COLOCATED CONFERENCES
Onward!
Dynamic Language Symposium (DLS)
Pattern Languages of Programs (PLoP)
and more

CONFERENCE CHAIR
William R. Cook, UT Austin
chair@splashcon.org

OOPSLA PROGRAM CHAIR
Martin Rinard, MIT
program@splashcon.org

For information, please contact
ACM Member Services Department
1-800-342-6626 (US & Canada)
+1-212-626-0500 (global)
info@splashcon.org

SPLASH/OOPSLA is sponsored by
ACM SIGPLAN and SIGSOFT

WWW.SPLASHCON.ORG