

Dr. Dobb's Journal

MARCH 2012

Next

ALSO INSIDE

PLINQ:
[Parallel Queries in .NET >>](#)

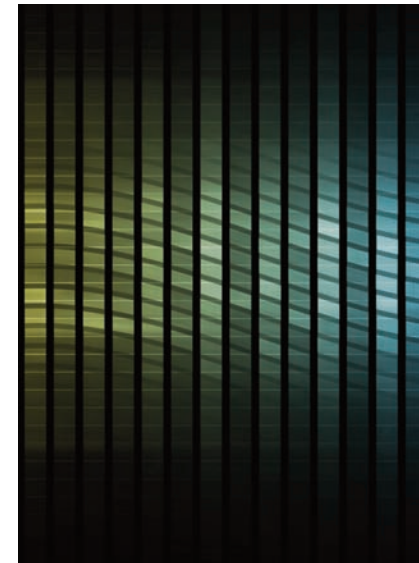
From the Vault:
[Concurrency Runtime \(CRT\) —
The Task Scheduler >>](#)

Creating a Custom Output Cache Provider in ASP.NET 4

Dr. Dobb's Journal

CONTENTS

March 2012



COVER STORY

8 Creating a Custom Output Cache Provider in ASP.NET 4

By Joydip Kanjilal

ASP.NET 4 provides support for cache extensibility through a provider model that you can write yourself. The primary benefit of this approach is that you can cache objects outside of the ASP.NET Worker Process. Getting the performance benefits of this customized cache control requires a few basic steps, presented in this article.

15 PLINQ: Parallel Queries in .NET

By Donis Marshall

LINQ queries execute when you iterate over the results, and they execute sequentially. With PLINQ, the iterations are performed in parallel, as tasks are scheduled on threads running in the .NET Framework 4 thread pool.

6 Editorial

By Andrew Binstock

33 From the Vault: Concurrency Runtime (CRT) — The Task Scheduler

By Lahlali Issam

Visual C++ 2010 came with new features and enhancements to simplify native programming. The Concurrency Runtime (CRT), for instance, is a framework that simplifies parallel programming and helps you write robust, scalable, and responsive parallel applications. This article explores the Task Scheduler, which schedules and coordinates tasks at runtime.

3 Letters

By you

42 Links

Snapshots of the most interesting items on drdobbs.com including an introduction to JRuby and a tour through the parallel hardware jungle.

43 Editorial and Business Contacts

More on DrDobbs.com

Working with Dropbox

Dropbox client applets allow only basic functionality for interacting with the file storage service. To get the full breadth of capabilities, you'll need to use their SDK and write your own code.

<http://drdobbs.com/tools/232600482>

Project of the Month: TestNG

Not every coder uses xUnit frameworks for testing. Many prefer TestNG, which gets around xUnit limitations, supports testing at multiple levels (unit, functional, integration, end-to-end), and greatly facilitates data-driven tests.

<http://drdobbs.com/testing/232600389>

Iterators and iostreams

A few weeks back, Mark Nelson was debating whether to use iterators or streaming operations for I/O on his data-compression code. He bemoaned the fact that the C++ iterators that perform stream I/O use the insertion and extraction operators, making them unsuitable for binary data compression. It looks like he spoke too soon...

<http://drdobbs.com/blogs/cpp/232600539>

Booting an Intel Architecture System, Part II: Advanced Initialization

Once the processor is running and memory has been initialized, timers and devices must be started up and a memory map laid out. Only then, can the OS be loaded.

<http://drdobbs.com/go-parallel/article/design/232400476>

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

Mailbag

RISCy moves, coding classes in the post-PC world,
and Delphi's whereabouts

When Did Intel Move to a RISC Core?

Whilst plying the interTubes to find a factoid for a piece I'm writing, I found your editorial on Intel/ARM (<http://is.gd/cDVCez>) wherein you say: "To be fair, though, RISC did score an architectural triumph in the heart of CISC. Intel x86 processors use a RISC-based instruction execution unit. CISC instructions are converted to an internal RISC format before being executed."

That has been my understanding for some time. What I can't find is which Intel x86 processor was the last to execute the instructions in silicon. Further, I'll suppose that the transition from full silicon to micro-code to full RISC behind a translator wasn't a single chip.

Do you recall? My guess would be the first 486 or last 386.

— Robert Young

Andrew Binstock responds: "I don't know the exact moment, because at the time Intel was being careful about releasing this information. However, two developments occurred at roughly the same time: 1) The ALU could perform a multiplication in half a clock cycle, and 2) L1 instruction caches were no longer sized according to KB, but by the

number of instruction traces they held. A quick look around suggests that this was well after the Pentium Pro began shipping, so a long while after the 486. According to Wikipedia (and consistent with my recollection), the double-pumped ALU shipped in the Pentium Net-Burst architecture, which was released right at the end of 2000."

Coding in the Post-PC World

In response to Mike Riley's blog series on post-PC coding (<http://is.gd/J1FjFz>), primarily by using a tablet on the front end and a cloud host on the back end, we received this letter:

I am about to take a web application development class in college that will be looking into (and using) Ruby on Rails, HTML5, JavaScript, jQuery, and possibly AJAX.

The professor has recommended using a Mac with TextMate; however, I of course own a massive desktop-replacement-sized gaming laptop that is getting to be too annoying to haul to class at the time.

I would like a tablet and think I would get more overall use out of it than purchasing a second laptop I really don't need. My question is, can today's tablets (iOS or Android) and their editors handle coding this type of material?

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

If I go with an iPad or a Tablet, I would also plan on getting a separate keyboard for input, so the on-screen keyboard limitations are a non-issue.

— **Andrew Norton**

Mike Riley replies: “The TextMate replacement I use on my iPad for Rails, HTML, etc. is an iOS app called ‘TexTastic.’ It supports TextMate bundles and natively supports the syntax you mentioned. Choices on the Android platform are more limited, especially when benchmarking text editors against TextMate, which is a Mac-only application. That said, the text editor I use the most on Android is a free app called ‘TextWarrior,’ which features highlighting for a variety of languages including HTML, Ruby, and Python. But if I had to recommend one platform over another, I would have to advocate the iPad for now. Perhaps once Android tablets start shipping with ICS (Android 4.0) on them and the apps start taking advantage of the new 4.0 API’s, it will be a different story. But that won’t be for at least another 6-8 months, maybe longer.

Of course, you will still need a server to copy the code to in order to execute it, so assuming your college allows you to connect your iPad to their WiFi and you can copy onto a server pre-configured with the frameworks you need to execute the code you’ve written, you should be good to go. The Bluetooth wireless keyboard I recommend is the Logitech Tablet keyboard (<http://is.gd/wvqZtD>) or the Apple wireless keyboard (<http://is.gd/nDIZQc>).”

In a follow-up note about using this set-up in college classes, Andrew Norton recounted: My professor looked at me like I was about to sail a ship off the edge of the flat-earth when I asked if anyone had tried taking the class using a tablet. His response was almost, but not quite, to the point of “It can’t be done.”

Part of this is because the course I’m taking is still relatively new for our campus, and I think the other part is that it may be such a radical shift in thinking or difference from the norm — to use tablets instead of laptops/desktops — that he couldn’t conceive how one would be able to do this. He cited a tablet’s lack of processing power and the potential problems of getting all of the software, application frameworks, and version controls, libraries, and model-view-controllers that are required for application development in our class to work together. The conversation was along the lines of “you can try, but don’t say I didn’t warn you if it doesn’t work...as someone still relatively new to coding, why risk your grade trying this out?”

To that point, the professor is right. However, I also think this leads to what Mike said in Part 4 of the Coding in a Post-PC environment series (<http://is.gd/QZXlwb>): “We still have a long way to go before developers will leave behind their laptops en masse and opt for this early glimpse of what the future holds for those who love to code.” Perhaps as an older man (in his late 60s), my professor is more comfortable with the tried-and-true traditional methods.

Another option one student in my systems analysis class is trying is using a tablet (in this case, an ASUS Transformer running Honeycomb, upgradable to ICS) as an interface to remote-desktop into his home PC. I have a desktop PC at home where I do most of my work at night anyway, and remoting-in may actually be a better solution to my problem of not wanting to bring a seldom used 17”, 8-lb gaming laptop to class to be used simply as an interface. It would also allow for trying out coding from a tablet, with a sort of “fail-safe” of remoting-in to my PC if I run into problems. We’ll see.

Any Problem with Delphi?

Regarding your editorial, “The Rise and Fall of Programming Languages in ” (<http://is.gd/ZsrmlT>), is there some deliberate boycott of

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

Delphi going on? Has the language become that obscure? Granted, in the Seattle area where I live and work (in the shadow of the Evil Empire), Delphi programmers are something of a rare breed. Yet, judging from the interest and newsgroups, the language is much more popular in Europe and Russia than “over here.” Why is that?

Speaking for myself, during a 20+-year career, I’ve been in and out of C, C++, and C#; I still maintain that I am far more productive in Delphi than any C-based language. Yes, it’s more than a language, it’s a Windows-slanted IDE with powerful language support, and a robust, easily read, easily understood syntax. Bottom line, I can deliver value to customers and users in less time with Delphi, and have readable, maintainable code to boot.

So what’s the problem?

— Lane Campbell

Andrew Binstock responds: “My article was about trends in language adoption. What happens in the Delphi universe is rarely the result of activities in other languages, and vice versa. Delphi is a *sui generis* language that is not part of any discernible trend, and so not part of my coverage. If you want to look at Delphi in isolation, only Tiobe has meaningful figures on its adoption. Those numbers show an increase year over year, but in the context of a larger downturn. If at some point Delphi’s fate is tied to advances or declines of other languages, it will certainly appear more prominently in my annual discussion of language trends.”

Have a correction or a thoughtful opinion on *Dr. Dobb’s* content? Let us know! Write to Andrew Binstock at alb@drdobbs.com. Letters chosen for publication may be edited for clarity and brevity. All letters become property of *Dr. Dobb’s*.



SHRINKWRAP YOUR APP

WITH AWARD-WINNING VERISIGN® CODE SIGNING

You developed the software. Now, deliver it with the same care and vigilance by using VeriSign® Code Signing. Why? Code signing not only protects the identity and reputation of the author, but it also verifies the authenticity and version of your software. Then, go a step further. VeriSign Code Signing can create a unique digital signature every time the code is signed. Plus, we support more certification programs and development platforms than any other Certificate Authority. Leverage the reputation of the most recognized and trusted name in online security.



Learn how VeriSign Code Signing can help make sure your applications are more trusted and adopted at www.Verisign.com/CodeSigning or call 1-866-893-6565.



Now from
VeriSign
Authentication Services

Copyright © 2011 Symantec Corporation. All rights reserved. Symantec, the Symantec Logo, and the Checkmark Logo are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. VeriSign and other related marks are the trademarks or registered trademarks of VeriSign, Inc. or its affiliates or subsidiaries in the U.S. and other countries and licensed to Symantec Corporation. Other names may be trademarks of their respective owners.

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

[EDITORIAL]

The Shrinking Expanding World of CI



Continuous integration is undergoing somewhat of a renaissance as continuous delivery (and its sidekick, DevOps) begin to find adoption in many enterprises. Simultaneously, the number of viable CI packages is shrinking quickly.

Of all the best practices enunciated by the Agile movement, perhaps no other makes as much sense in practice as continuous integration (CI). It is predicated on the simple theory that if you rebuild the project every time source code is checked in, developers can be informed immediately of changes that break the build. The shorter the loop between the break and notification to the developer, the more likely the developer will still remember the details of the changes, hunt down what's wrong, and get it fixed. This approach is easy to understand, straightforward to implement, and very effective. If you're not using continuous integration, you're working under a very substantial handicap.

A secondary, but important, benefit of CI is that by building the product regularly, the organization has in theory a working (if still incomplete) binary that it can use to demo the software if the need arises.

(Without warning, the CIO might want to see where things stand). Moreover, the usual problem of discovering build problems — one of the final stages in waterfall-like methodologies — is moved up the cycle, so that the effects of changes on the builders are known all the way along. This step removes all the frantic patching and rewriting that occurs in non-CI shops when a project is first tested.

The next step in CI, called "Continuous Delivery" after the Jolt Award-winning book that came out in 2010 (<http://is.gd/LZMzGT>), takes CI even further. It contends that the process automation of CI should be extended to delivering and installing the apps and running tests on the installed app. This added step gives the development organization an additional level of comfort. They can automate installation and verification with each new build, and know that the product subset they can deliver on any given day is completely ready to go should the client want to see it. The need for tighter integration between development and operations caused by the automation of continuous de-

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

livery is what forged the term “DevOps.” It is supposed to mean a closer relationship between developers and operators (a historically wide and sometimes acrimonious divide), but in fact, everyone seems to have different definitions of what DevOps really means.

Given the new emphasis on CI and continuous delivery, you’d think that the industry would be offering all kinds of new tools — somewhat akin to the sudden profusion of cloud technology vendors. But in fact, the reverse is happening. The high end is consolidating around two or three principal products, and the low-to-middle end is being dominated by two OSS solutions. The numerous other OSS CI products of the past have disappeared entirely or are simply awaiting word of their impending doom.

At the high end, there are IBM Rational BuildForge and Electric-Cloud’s Electric Commander. These two products regularly handle the build, test, deploy stages of codebases greater than 20 million LOCs. In fact, for both products that kind of line count is their sweet spot. In the next tier are UrbanCode’s DevOps Platform and Atlassian’s Bamboo, in that order, in so far as enterprise scale is concerned. There then comes a very large gap as we enter the domain of CI servers for SMBs and enterprise departments. Historically, these were served by open-source tools, especially Cruise Control. Other competitors, although somewhat less capable than Cruise Control, were Continuous, Hudson, LuntBuild, and others. Today, these are all more or less shrinking (LuntBuild seems to have ceased activity entirely) and being replaced by Jenkins (formerly Hudson).

Hudson was, for years, gaining ground on the other products listed here due to its ease of installation and numerous plugins. When it was

acquired by Oracle as part of the Sun purchase, there arose a dispute over the Hudson name. Most developers, including the principal developer, decided to fork the project and renamed it “Jenkins.” Hudson was then donated by Oracle to the Eclipse Foundation, where it is primarily maintained by Oracle contributors and staff from Sonatype (the Maven company). While the bulk of the new activity is occurring in the Jenkins fork, Hudson is undergoing a top-to-bottom cleanup. Which version you prefer depends mostly on personal choice as the differences between them are salient mostly to diehards within the respective communities. Either one will serve you well. And between them, they are clearing out all other competitors in the low to lower-middle strata of the market.

It’s not often that we see a market’s mission expand, while the number of its constituent products contracts. Nonetheless, I expect, that both trends will continue. Continuous delivery will reach farther into development process automation and the number of companies who can keep up with the new demands will continue to shrink. As a result, it makes sense to take time to choose your CI platform with particular care. CI builders can be difficult to swap out if you made the wrong choice.

— *Andrew Binstock is Editor in Chief for Dr. Dobbs and can be contacted at alb@drdobbs.com.*

[Comment](#)

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

Creating a Custom Output Cache Provider in ASP.NET 4

ASP.NET 4 provides support for cache extensibility through a provider model that you can write yourself. The primary benefit of this approach is that you can cache objects outside of the ASP.NET Worker Process. Getting the performance benefits of this customized cache control requires a few basic steps, presented here.

By Joydip Kanjilal

The performance of ASP.NET applications is a serious concern for Web application developers. There have been many improvements, though. The newer versions of .NET Framework have been optimized a lot. Added to all this, you have IIS improvements — better support for application pools, IIS compression, etc. However, we still use cache as a state management technique for storing relatively stable data. In doing so, redundant hits to databases are reduced and the application works faster. Proper usage of caching cuts down on network traffic and ensures that applications are more responsive.

The ASP.NET Caching Framework opens the door for aggressive output caching: It allows you to extend the default cache model and build your own cache providers. In this article, I discuss the basics of cache state management in ASP.NET and how to extend the ASP.NET Caching Framework to implement your own custom caching framework.

To work with the code examples in this article, you should have either Visual Studio 2010 or Visual Studio 11 installed on your system. As of this writing, the Developer Preview Version of Visual Studio 11 (<http://www.microsoft.com/download/en/details.aspx?id=27543>) is available for free download. The release date of Microsoft Visual Studio 11 has not yet been announced.

Caching: The Basics

Caching is a state management technique that optimizes your application's performance by enabling you to store relatively stable objects in memory and use them as needed without having to access the database or your file system to fetch the same data. If the data in your data store changes, you still can take advantage of cache dependency to ensure that data in the data store synchronizes with that in the memory.

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

MSDN (<http://msdn.microsoft.com/en-us/library/dd997357.aspx>) states: "Caching enables you to store data in memory for rapid access. When the data is accessed again, applications can get the data from the cache instead of retrieving it from the original source. This can improve performance and scalability. In addition, caching makes data available when the data source is temporarily unavailable."

Types of Caching in ASP.NET

There are three types of caching in ASP.NET:

- **Page Output Caching** — The output of the Web page is stored in cache so that the next request for the same page can be fetched directly from the cache rather than having to reload it again.
- **Page Fragment or Partial Page Caching** — As the names suggest, only fragments or portions of a Web page are cached.
- **Data Caching** — you can make use of the Cache API to store the frequently used objects of your application in the cache. So, the processing time of recreating such objects is eliminated.

Implementing your own Custom Cache Provider in ASP.NET

It should be noted that CPU and memory are finite resources. In earlier versions of ASP.NET, you were limited to using the default caching mechanism. That is, you did not have any control or any way to extend and build your own cache providers. ASP.NET 4 provides support for cache extensibility through its provider model. Why do we need this extensibility anyway? The most important benefit that you get from implementing your own cache provider is that you can have objects in the cache stored outside of the ASP.NET Worker Process.

Custom cache providers enable caching of data from various data stores that include disks, cloud storage, and distributed cache engines.

Microsoft documentation (<http://msdn.microsoft.com/en-us/magazine/gg650661.aspx>) states: "The ASP.NET output cache uses an in-memory storage mechanism and, until the .NET Framework 4, it wasn't possible to override or replace the default cache with your own implementation. With the new `OutputCacheProvider` type, it's now possible to implement your own mechanism for caching page output in ASP.NET."

Here's how you can specify output caching in an ASP.NET Web page:

```
<%@ OutputCache
    Duration="Specify duration to cache in milliseconds"
    Location="Any" VaryByParam="None" %>
```

You have quite a few vary parameters to choose from: `VaryByParam`, `VaryByHeader`, `VaryByCustom`, `VaryByControl` and `VaryByContentEncoding`.

In his blog on this topic, (<http://is.gd/sR9rg5>), Scott Guthrie states: "ASP.NET 4 adds an extensibility point to output caching that now enables developers to configure one or more custom output-cache providers. Output-cache providers can use any storage mechanism to persist output cached content. This makes it possible to easily create output-cache providers that store the cached content using any persistence mechanism — including local or remote disks, databases, cloud storage, and distributed cache engines (like memcached or Velocity)."

To implement your own Custom cache provider class, you need to extend the `OutputCacheProvider` abstract class of the `System.Web.Caching` namespace:

```
public abstract class OutputCacheProvider : ProviderBase
```

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

The `OutputCacheProvider` class contains several abstract methods that you can override to implement your version of a custom cache provider. These methods include the following:

- `Add()` — This method is used to add an item to the cache
- `Get()` — This method is used to retrieve an item from the cache
- `Remove()` — Used to remove an item from the cache
- `Set()` — This method is used to update an item previously added to the cache

Here is how the `OutputCacheProvider` class looks:

```
public abstract class OutputCacheProvider :
    ProviderBase
{
    public abstract object Get(string key);
    public abstract object Add(string key,
        object entry, DateTime utcExpiry);
    public abstract void Set(string key,
        object entry, DateTime utcExpiry);
    public abstract void Remove(string key);
}
```

To implement your own custom cache provider, override the methods of the `OutputCacheProvider` class shown above. It should be noted that custom cache providers support only *Absolute* cache expiration— there is no support for sliding expiration. It should also be mentioned here that cache expiration policies can be of two types: file-based and time-based. Also, if you have multiple cache providers available, you can allow your application to select any among them at runtime depending on the application's needs and the configuration set in the application's configuration file.

The `System.Web.Caching` namespace contains several classes, interfaces, and enumerations that help to manage your cached objects.



dtSearch®

Instantly Search Terabytes Of Text

- 25+ fielded & full-text search options
- dtSearch's own file parsers **highlight hits** in popular file & email types
- Spider supports static & dynamic data
- APIs for .NET, Java, C++, SQL, etc.
- Win / Linux (64-bit & 32-bit)

"Lightning Fast" – *Redmond Mag*

"Covers all data sources" – *eWeek*

"Returns results in less than a second"
– *InfoWorld*

www.dtSearch.com

Fully-Functional Evaluations

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

You can find more information on the `System.Web.Caching` namespace at the Microsoft site (<http://msdn.microsoft.com/en-us/library/system.Web.caching.aspx>).

With .NET 4, you can leverage caching in ASP.NET, Windows Forms, and WPF applications. The `System.Runtime.Caching` namespace has been enhanced a lot to facilitate this. You now can use `MemoryCache` and your application can have multiple `MemoryCache` instances at a given point in time. You can retrieve a reference to `ObjectCache` using the following statement:

```
static readonly ObjectCache Cache =
    MemoryCache.Default;
```

Next, you can add, remove, or edit your objects in the cache. The code sample below illustrates how you can store your objects using the reference you just created:

```
public static void Add(object obj, string key)
{
    Cache.Add(key, obj, DateTime.Now.AddDays(90));
}
```

The types in the `System.Runtime.Caching` namespace now provide support for caching in any application that runs on top of the .NET Framework — not just ASP.NET. A great feature indeed!

In this section, I'll discuss how we can leverage the cache extensibility feature in ASP.NET to implement the custom cache provider class.

The CustomCacheProvider Class

Our `CustomCacheProvider` class extends the abstract `OutputCacheProvider` class.

```
public class CustomCacheProvider : OutputCacheProvider
{
}
```

The following code listing illustrates our `CacheItem` class:

```
using System;

namespace DDJ.Caching
{
    [Serializable]
    internal class CacheItem
    {
        public string Key { get; set; }
        public byte[] Data { get; set; }
        public DateTime Expiration { get; set; }
    }
}
```

Implementing the Methods of the CustomCacheProvider Class

At first glance, here's how our `CustomCacheProvider` class looks:

```
using System.Web.Caching;
namespace DDJ.Caching
{
    public class CustomCacheProvider :
        OutputCacheProvider
    {
        public override object Add(string key, object entry,
            System.DateTime utcExpiry)
        {
            throw new System.NotImplementedException();
        }

        public override object Get(string key)
        {
            throw new System.NotImplementedException();
        }
    }
}
```

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

```

        public override void Remove(string key)
        {
            throw new System.NotImplementedException();
        }

        public override void Set(string key,
            object entry, System.DateTime utcExpiry)
        {
            throw new System.NotImplementedException();
        }
    }
}

```

The next step is to save your cached data to a persistent storage device, like a database or a file in the file system. There are plenty of articles available that discuss how you can write code to achieve this, so I'll skip this discussion here.

```

public void Save(CacheItem item, string key)
{
    // Write your custom code here to save your cached item to
    // a persistent storage device.
}

```

Now, your CustomCacheProvider class looks like this:

```

using System.Web.Caching;
using System.Collections.Generic;

namespace DDJ.Caching
{
    public class CustomCacheProvider : OutputCacheP-
        rovider
    {
        // Implement the Overriden methods of the
        // OutputCacheProvider class here.

        public void Save(CacheItem item, string key)
        {
            // Write your custom code here to save

```

```

        // your cached item to
        // a persistent storage device.
    }
}

```

Now that your CustomCacheProvider class is ready, the next step is to configure the cache provider. The next section discusses how to do this.

Configuring the Custom Cache Provider

To configure the custom cache provider, specify the details in your application's Web.config file. Here's how you can specify the configuration:

```

<configuration>
  <system.Web>
    <compilation debug="true" targetFramework="4.0"/>
    <caching>
      <outputCache defaultProvider="CustomOutputCache">
        <providers>
          <add name="CacheProviderName"
              type="CacheProviderCategory" />
        </providers>
      </outputCache>
    </caching>
  </system.Web>
</configuration>

```

So, if you have implemented a custom cache provider called FileCacheProvider, for example, you can specify this in the application's configuration file:

```

<configuration>
  <system.Web>
    <caching>

```

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

```

<outputCache defaultProvider="
    AspNetInternalProvider">
    <providers>
        <add name="FileCache"
            type="CustomCacheProvider" />
    </providers>
</outputCache>
</caching>
</system.Web>
</configuration>

```

Selecting the Custom Output Cache Provider Dynamically

Next, override the virtual method called `GetOutputCacheProviderName()` of the `HttpApplication` class in the `global.asax` file to ensure that the correct cache provider is selected at the application level.

```

public class Global : System.Web.HttpApplication
{
    public override string
        GetOutputCacheProviderName(HttpContext context)
    {
        return base.GetOutputCacheProviderName(context);
    }
}

```

It is also possible to select the output cache provider to be dynamically on a per-request basis by overriding the `GetOutputCacheProviderName()` method as shown in the code snippet below:

```

public class Global : System.Web.HttpApplication
{
    public override string
        GetOutputCacheProviderName(HttpContext context)
    {
        if (context.Request.Path.EndsWith("Default.aspx"))
            return "AspNetInternalProvider";
    }
}

```

```

        return base.GetOutputCacheProviderName(context);
    }
}

```

Now, your `Global.ascx.cs` file should look similar to this:

```

using System;
using System.Web;

namespace DDJ
{
    public class Global : System.Web.HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
        }

        void Application_End(object sender, EventArgs e)
        {
        }

        void Application_Error(object sender, EventArgs e)
        {
        }

        void Session_Start(object sender, EventArgs e)
        {
        }

        void Session_End(object sender, EventArgs e)
        {
        }

        public override string
            GetOutputCacheProviderName(HttpContext context)
        {

```

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

```

        if (context.Request.Path.EndsWith
            ("Default.aspx"))
            return "AspNetInternalProvider";

        return base.GetOutputCacheProviderName(context);
    }
}

```

“Caching can significantly reduce network latency and boost an application’s performance”

Note that the `Application_Start()`, `Application_End()`, `Application_Error()`, `Session_Start()`, and `Session_End()` are the default event handlers of the `System.Web.Global` class. You can also make use of the `RawUrl` property of the `Request` object to get the right custom cache provider dynamically as shown here:

```

public override string GetOutputCacheProviderName(HttpContext context)
{
    if (context.Request.RawUrl.ToUpper()
        .Contains("File"))
    {
        return "CustomFileOutputCacheProvider";
    }

    return base.GetOutputCacheProviderName(context);
}

```

And, you are done! You can now use your custom cache provider in your application.

Conclusion

Caching can significantly reduce network latency and boost an application’s performance. If you have a large number of concurrent requests to the same resource, caching can come in handy by allowing you to avoid redundant calls to the data stores. In this article, I explored the new features in the Cache API in .NET Framework 4, the ASP.NET Cache extensibility feature, and how these can be leveraged to implement your own custom cache provider.

References

These links provide additional information that you’ll find helpful:

- Extending ASP.NET Output Caching: <http://www.4guysfromrolla.com/articles/061610-1.aspx>
- Building and Using Custom OutputCache Providers in ASP.NET: <http://msdn.microsoft.com/en-us/magazine/gg650661.aspx>
- Cache Abstraction on ASP.NET: <http://Weblogs.asp.net/zowens/archive/2008/08/04/cache-abstraction.aspx>

— *Joydip Kanjilal is a frequent author on Microsoft technologies. He has won the Microsoft MVP Award for the last five years.*

[Comment](#)

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

PLINQ: Parallel Queries in .NET

LINQ queries execute when you iterate over results, and they execute sequentially. With PLINQ, the iterations are performed in parallel, as tasks are scheduled on threads running in the .NET Framework 4 thread pool

By Donis Marshall

Language-Integrated Query (LINQ) is a feature of Microsoft's .NET Framework that adds native data-query capabilities to .NET languages. PLINQ is a parallel version of LINQ. It was introduced with .NET Framework 4.0.

The objective of parallel programming is to maximize processor utilization with increased throughput in a multicore architecture. For a multicore computer, your application should recognize and scale performance to the number of available processor cores. LINQ queries execute when you iterate over the results, and they execute sequentially. With PLINQ, the iterations are performed in parallel, as tasks are scheduled on threads running in the .NET Framework 4 thread pool.

One of the best features of PLINQ is that it's easy to convert LINQ queries to PLINQ. You can simply add the `AsParallel` clause. Here is a simple LINQ query that returns a selection of books. The results are generated sequentially:

```
from book in books
where book.Publisher=="Lucerne Publishing"
orderby book.Title
select book;
```

Now, here's the same query updated for PLINQ. Note that the only addition to the code is the call to the `AsParallel` method of the `books` collection. This minor change, however, completely alters how the query is performed. When you iterate over the results, the query is performed with parallel tasks.

```
from book in books.AsParallel()
where book.Publisher=="Lucerne Publishing"
orderby book.Title
select book;
```

The following discussion will present an example that contrasts the productivity of standard LINQ and PLINQ. You can perform the example query either sequentially or in parallel. You'll display information to com-

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

pare the performance of both approaches. The task and thread identifiers are also displayed to highlight the underlying differences between parallel and sequential execution. Because the sequential version of the query does not use the Task Parallel Library (TPL), the task IDs are blank. In addition, the sequential version will execute on a single thread.

Perform a sequential query and a parallel query on an integer array and imperatively invoke the `Where` clause:

1. Create a console application for C# in Visual Studio 2010. Add using statements for both the `System.Threading` and `System.Diagnostics` namespaces.
2. Above the `Main` method, create a static method called `Normalize` that returns a `bool`. You'll call this method in the `Where` clause. In the `Normalize` method, display the current task and thread identifiers. Use the `Thread.SpinWait` method to simulate a real-world normalization operation. Return `true` to select and add the current element to the result collection.

```
static bool Normalize()
{
    Console.WriteLine(
        "Normalizing [Task {0} : Thread {1}]",
        Task.CurrentId,
        Thread.CurrentThread.ManagedThreadId
    );
    Thread.SpinWait(int.MaxValue);
    return true;
}
```

3. In `Main`, create an instance of the `Stopwatch` class. The `Stopwatch` is used to calculate the duration of both the sequential and the parallel versions of the PLINQ query. Also, define an integer array that has four elements. Here is the array you will query:

```
Stopwatch sw = new Stopwatch();
var intArray = new [] { 1, 2, 3, 4 };
```

4. Perform a sequential query on the array by using `LINQtoObjects`. Call the `Where` method. Evaluate a lambda expression and call the `Normalize` method as a parameter. The `Where` method — and consequently `Normalize` — will be called for each element of the array. Because the lambda expression returns true for each element, all elements of the array are included in the results. On the next line, repeat the query but use PLINQ. Add the `AsParallel` method. For now, comment out the parallel version of the query. You will initially run only the sequential query.

```
var result = intArray.Where((index)=>Normalize());
//var result = intArray.AsParallel().Where((index) =>
//Normalize());
```

5. Start the `Stopwatch`, and then iterate the results of the query. Display the results of the operation. Because of deferred execution, this is when the query actually executes.

```
foreach (int item in result)
{
    Console.WriteLine("Item={0}", item);
}
```

6. Call the `Stop` method on the `Stopwatch` class and display the duration:

```
sw.Stop();
Console.WriteLine("Elapsed time: {0}: seconds",
    sw.ElapsedMilliseconds / 1000);
```

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

Here is the entire program:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Diagnostics;
using System.Threading.Tasks;
namespace Performance
{
    class Program
    {
        static bool Normalize()
        {
            Console.WriteLine(
                "Normalizing [Task {0} : Thread {1}]",
                Task.CurrentId,
                Thread.CurrentThread.ManagedThreadId
            );
            Thread.SpinWait(int.MaxValue);
            return true;
        }

        static void Main(string[] args)
        {
            Stopwatch sw = new Stopwatch();
            var intArray = new [] { 1, 2, 3, 4 };
            var result = intArray.Where((index) =>
                Normalize());
            //var result =
            //intArray.AsParallel().Where((index) => Normalize());

            sw.Start();
            foreach (int item in result)
            {
                Console.WriteLine("Item={0}", item);
            }
            sw.Stop();
        }
    }
}

```

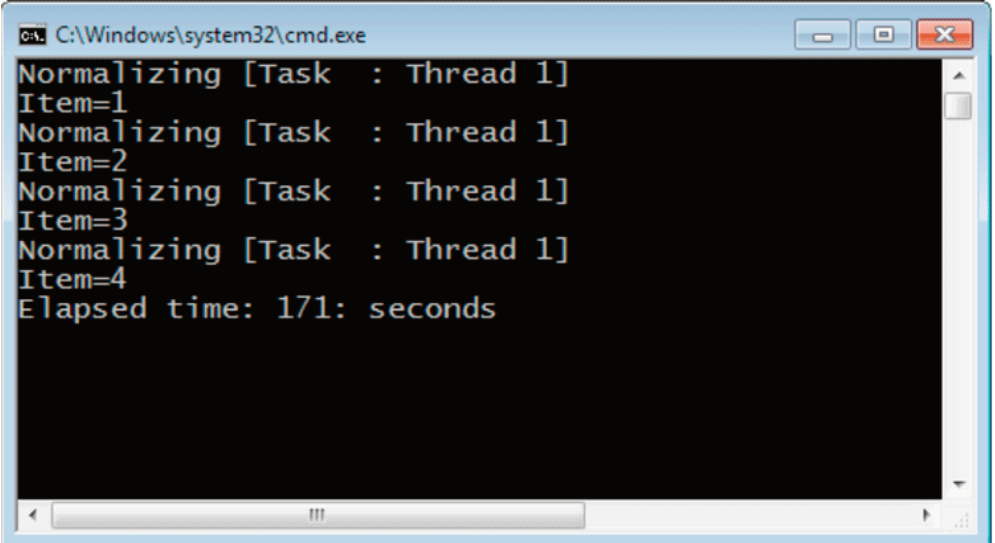
```

        Console.WriteLine("Elapsed time: {0}:
                           seconds", sw.ElapsedMilliseconds / 1000);
        Console.WriteLine("Press Enter to Continue");
        Console.ReadLine();
    }
}
}

```

Build and run the application. Because the statement containing the PLINQ query is commented, the code executes only the standard LINQ query. Each operation is therefore performed sequentially and on the same thread, which you can see from the output in the console window as shown in Figure 1. Because parallel tasks are not used, task IDs are not displayed. The duration is essentially the sum of running each of the operations in order.

Now uncomment the statement containing the PLINQ command and comment out the LINQ query instead. Rerun the application. This time, the results are entirely different. The `Where` method runs in parallel and



```

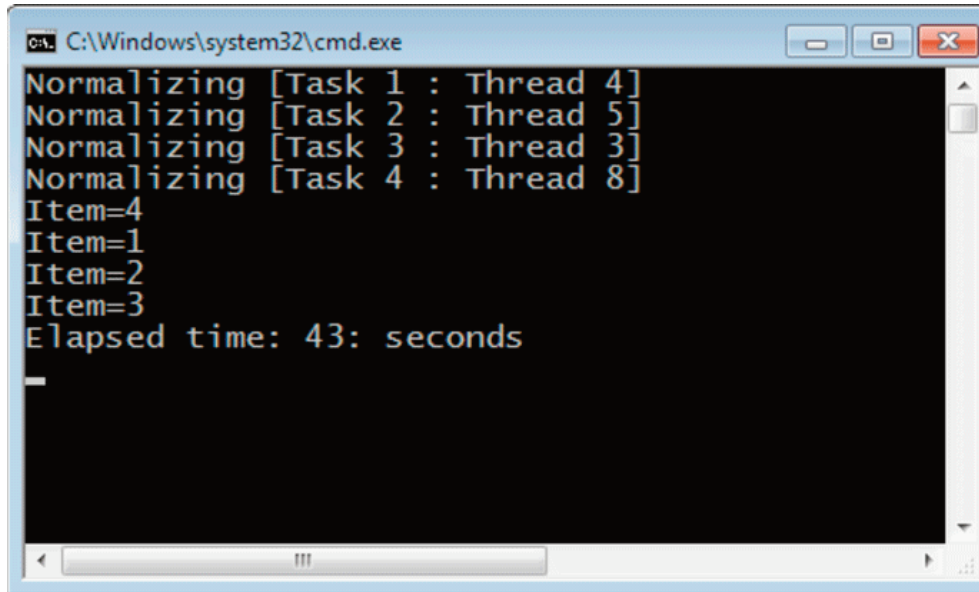
C:\Windows\system32\cmd.exe
Normalizing [Task : Thread 1]
Item=1
Normalizing [Task : Thread 1]
Item=2
Normalizing [Task : Thread 1]
Item=3
Normalizing [Task : Thread 1]
Item=4
Elapsed time: 171: seconds

```

Figure 1: Output of the serial version.

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)



```
C:\Windows\system32\cmd.exe
Normalizing [Task 1 : Thread 4]
Normalizing [Task 2 : Thread 5]
Normalizing [Task 3 : Thread 3]
Normalizing [Task 4 : Thread 8]
Item=4
Item=1
Item=2
Item=3
Elapsed time: 43: seconds
```

Figure 2: Output of the parallel version.

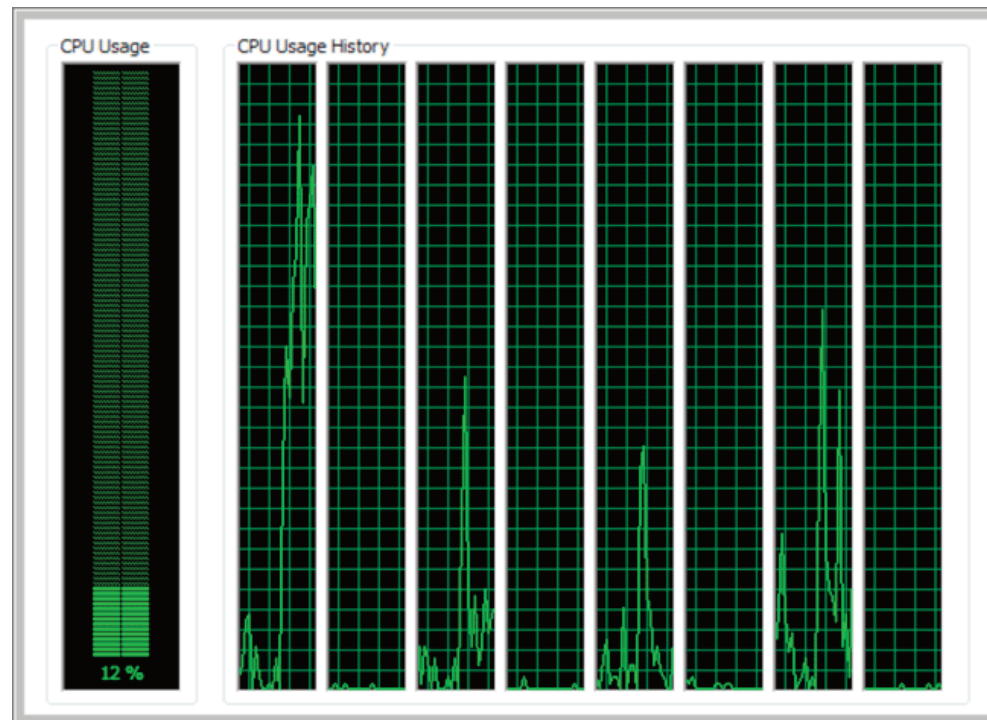


Figure 3: Processor utilization of the serial version.

on different threads, as shown in the output window in Figure 2. The PLINQ query leverages the multicore processor architecture; the results are specific to this example and the current hardware architecture. In this example, each iteration of a query operation is a different task. For this reason, the query runs faster.

The difference in performance of the two versions is depicted in the processor utilization of the Task Manager. Figure 3 is a screen shot of processor utilization from the LINQ version of the application. In this example, the average processor utilization is about 12 percent. Most of the processor computing capability is unused!

The PLINQ version of the application is much more efficient. In Figure 4 you can see a considerably higher utilization — more than 50 percent.

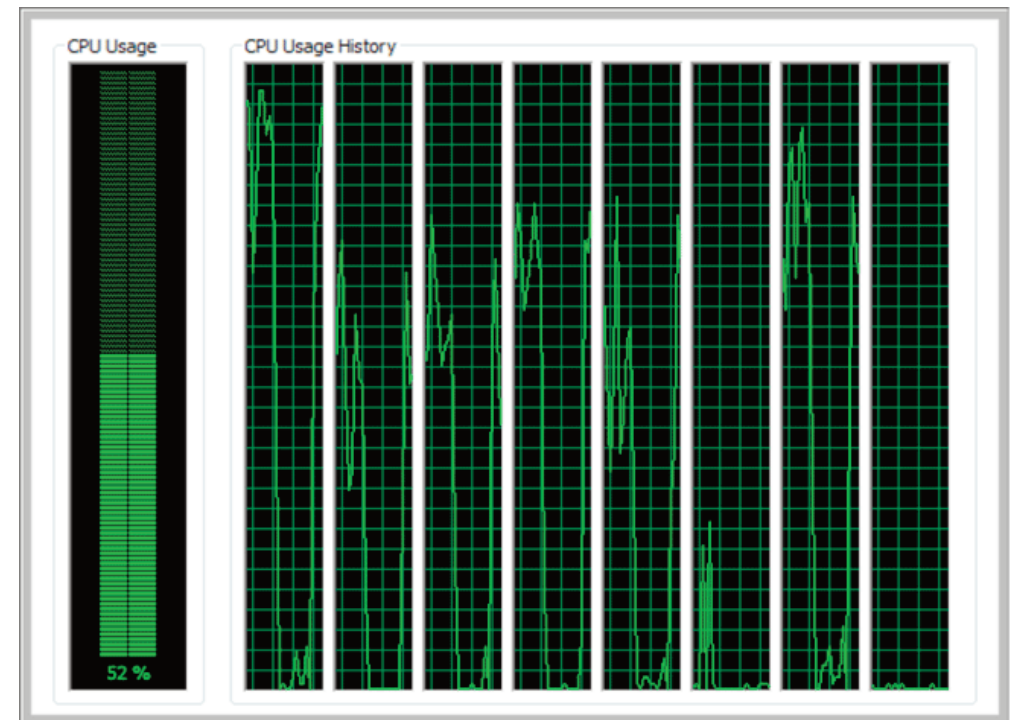


Figure 4: Processor utilization of the parallel version.

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

PLINQ Operators and Methods

You can modify the behavior of a PLINQ query with a variety of clauses and methods that are actually extension methods of `ParallelQuery<TSource>`. Most of these are the same clauses and methods available to LINQ. You can use these operators either independently or together to affect the behavior of a PLINQ query. However, PLINQ also introduces some new constructs, which are discussed in this section.

The ForAll Operator

You create a PLINQ query to parallelize your code. In most circumstances, the next step is to iterate the results by using a `foreach` or `for` method. At that time, the query is most likely performed by using deferred execution. The results are processed in iterations of the `foreach` loop. There is only one problem: The `foreach` loop is sequential. This is a classic “hurry-up-and-wait” scenario. After executing a PLINQ query, you might want to extend parallelism to handle the results in parallel as well.

LINQ’s `Parallel.ForEach` method is useful for parallelizing the same operation over a collection of values. It would appear natural to adhere to the same model to process the results of a PLINQ query. PLINQ returns a `ParallelQuery<TSource>` type, which represents multiple streams of data. However, `Parallel.ForEach` expects a single stream of data, which is then parsed into multiple streams. For this reason, the `Parallel.ForEach` method must recognize and convert multistream input to a single stream. There is a performance cost for this conversion.

The solution is the `ParallelQuery<TSource>.ForAll` method. The `ForAll` method directly accepts multiple streams, so it avoids the

overhead of the `Parallel.ForEach` method. Here is a prototype of the `ForAll` method. The first parameter is the target of the extension method, which is a `ParallelQuery` type. The last parameter is an `Action` delegate. For the `Action` delegate, you can use a delegate, a lambda expression, or even an anonymous method. The next element of the collection is passed as a parameter to the delegate.

```
public static void ForAll<TSource>(
    this ParallelQuery<TSource> source,
    Action<TSource> action
)
```

Here is a short demonstration that illustrates how to use the `ForAll` operator. In this example, you will perform a parallel query on a string array and then select and display strings longer than two characters in length.

Perform a parallel query of a string array

1. Create a console application for C# in Visual Studio. In the `Main` method, define a string array.

```
string [] stringArray = { "A", "AB", "ABC", "ABCD" };
```

2. Perform a PLINQ query on the string array. Select strings with a length greater than two.

```
var results=from value in stringArray.AsParallel()
where value.Length>2 select value;
```

3. Call the `ForAll` operator on the results. In the lambda expression, display the current item.

```
results.ForAll((item) => Console.WriteLine(item));
```

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

Here is the source code for the entire application:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ForAll
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] stringArray = { "A", "AB", "ABC", "ABCD" };
            var results = from value in stringArray.AsParallel()
                          where value.Length > 2 select value;
            results.ForAll((item) => Console.WriteLine(item));
            Console.WriteLine("Press Enter to Continue");
            Console.ReadLine();
        }
    }
}
```

The application will display ABC and ABCD as the result.

Using ParallelExecutionMode

So far, we have used the `AsParallel` method to convert LINQ to PLINQ. It is a simple change to a LINQ query that alters the semantics completely.

A PLINQ query is not guaranteed to actually execute in parallel. Overhead from executing the parallel query in parallel, such as thread-related costs, synchronization, and the parallelization code, can exceed the performance gain. Determining the relative performance benefit of the PLINQ query is an inexact science based on several factors. Here are some of the considerations that might affect the performance of a PLINQ query:

- Length of operations
- Number of processor cores
- Result type
- Merge options

One of the biggest factors is the duration of the parallel operations, such as the `Select` clause. Dependencies and the synchronization that results from them adversely affect the performance of any parallel solution. Furthermore, shorter operations might not be worth parallelizing, because the associated overhead might exceed the duration of the operation. For small operations, you could change the chunking to improve the balance of execution to overhead. Custom partitioners, including those that change the chunk size, are an option.

The number of processor cores might affect the performance of your parallel application, including PLINQ. However, you should typically ignore the number of processor cores, because that's mostly beyond your control. Maintaining hardware independence in your application is important for both scalability and portability.

PLINQ does not consider all of the above factors when deciding to execute a query in parallel. Based on the shape of the query and the clauses used, PLINQ decides to execute a query either in parallel or sequentially. You can override this default by using the `WithExecutionMode` clause with the `ParallelExecutionMode` enumeration as a parameter. The two options are `ParallelExecutionMode.ForceParallelism` and `ParallelExecutionMode.Default`. Use the `ParallelExecutionMode.ForceParallelism` enumeration to require parallel execution.

The `ParallelExecutionMode.Default` value defers to PLINQ for the appropriate decision on the execution mode. Here is an example that forces a parallel PLINQ query.

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

```
from item in data.AsParallel()
    .WithExecutionMode(
        ParallelExecutionMode.ForceParallelism
    )
select item;
```

Using WithMergeOptions

How the result of your query expression is handled can also affect performance. For example, the following PLINQ query returns a `List<T>` type. Converting the PLINQ to a list requires that the results be buffered to return an entire list.

```
intArray.AsParallel()
    .Where((value) => value > 5)
    .ToList();
```

As mentioned, for the above code, the results are buffered. In some circumstances, PLINQ might buffer the results, but that is mostly transparent to your code.

Using the .NET Framework 4 thread pool, PLINQ uses multiple threads to execute the query in parallel. The results of these parallel operations are then merged back into the joining thread. The merge option describes the buffering used when merging results from the various threads.

Here are the merge options as defined in the `ParallelMergeOptions` enumeration:

- **NotBuffered:** The results are not buffered. For operations such as the `ForAll` operation, `NotBuffered` is the default.
- **FullyBuffered:** The results are fully buffered, which can delay receipt of the first result.
- **AutoBuffered:** This option is similar to `NotBuffered`, except that the results are returned in chunks.
- **Default:** The default is `AutoBuffered`.

You can override the default buffer preference with the `WithMergeOptions` operator.

Using AsSequential

The difference between PLINQ and LINQ starts with the `AsParallel` clause. As we've seen, converting from LINQ to PLINQ is often as simple as adding the `AsParallel` method to a LINQ query. Here is a basic LINQ query:

```
numbers.Select(/* selection */ )
    .OrderBy( /* sort */ );
```

Here is a parallel version of the same query, with the required `AsParallel` method added.

```
numbers.AsParallel().Select
    (/* selection */).OrderBy( /* sort */ );
```

So far, you've seen the `AsParallel` method prefixed to only the `Select` clause. However, `Select` is only one of the LINQ clauses that can take the `AsParallel` method as a prefix. Starting at that clause, the remainder of the query is conducted in parallel. Methods preceding the `AsParallel` clause in the query statement execute sequentially. In this example, the `Select` clause executes sequentially, but the `GroupBy` and `OrderBy` clauses execute in parallel:

```
numbers.Select(/* selection */ )
    .AsParallel()
    .GroupBy( /* categorize */ )
    .OrderBy( /* sort */ );
```

`AsSequential` is the opposite of the `AsParallel` clause. `AsSequential` serializes portions of your LINQ query. You might choose

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

this to resolve dependencies in a PLINQ query. You can then use `AsSequential` to isolate the dependency and make a part of a PLINQ query sequential.

You might also decide that a portion of a PLINQ query is more efficiently run in parallel as opposed to sequentially.

Use `AsParallel` and `AsSequential` as gates for parallel and sequential execution, as shown in Figure 5. Although it is not common, a single PLINQ query can have multiple `AsParallel` and `AsSequential` clauses. Similar to the `AsParallel` clause, `AsSequential` can be used to prefix a LINQ method. From that position of the query forward, the remainder of the LINQ query executes sequentially — at least until it encounters an `AsParallel` clause. Figure 5 illustrates a PLINQ query with both `AsParallel` and `AsSequential` clauses. The `Select` and `Groupby` clauses execute in parallel, while the `OrderBy` clause is sequential.

Using `AsOrdered`

For some, an orderly universe is important. Unfortunately, this is contrary to the default behavior of PLINQ. The following code squares and renders the values of an integer list:

```
int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var result = from number in numbers.AsParallel()
    select number * number;
```

You might be surprised by the results, which are indeed squared but do not appear in the same order as the underlying list:

```
0 4 16 25 36 49 64 81 1 9
```

PLINQ creates tasks for the query. Each task is then scheduled and placed on a queue in the .NET Framework 4 thread pool. The tasks are

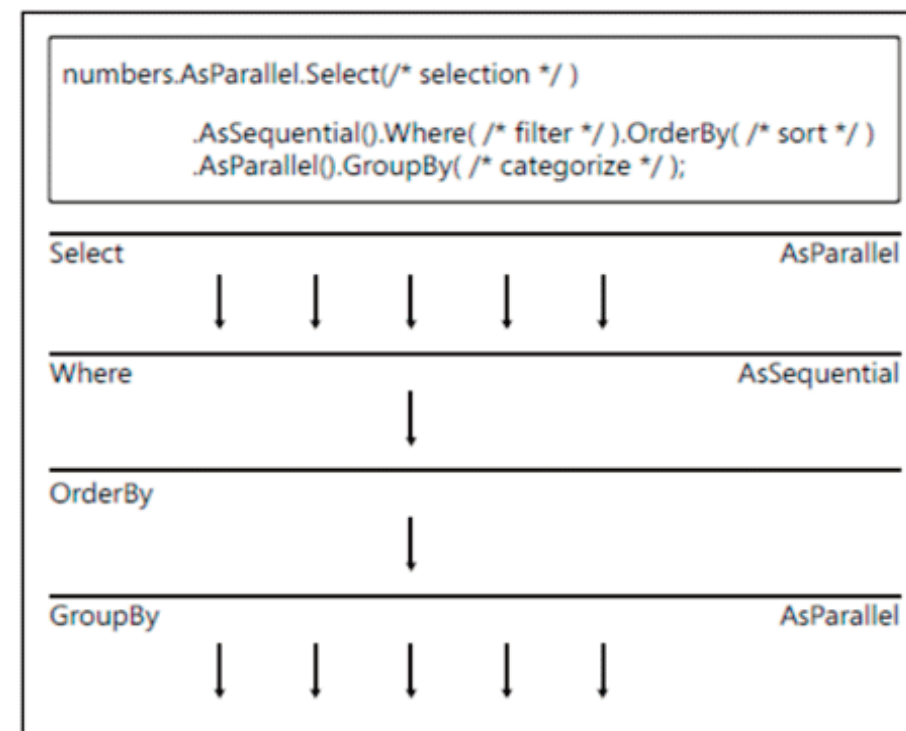


Figure 5: Execution in a PLINQ query with both `AsParallel` and `AsSequential` clauses.

then scheduled on processor cores and executed. But PLINQ does not guarantee the order in which tasks will execute, so it is likely, if not probable, that the list iteration is unordered.

If you prefer ordered results, use the `AsOrdered` clause. The PLINQ query still executes in an unordered fashion to improve performance and fully utilize the available processor cores. However, the results are buffered and then reordered at the completion of the query. This localizes the performance degradation to the `AsOrdered` clause.

Here is the modified query:

```
int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var result = from number in numbers.AsParallel()
    .AsOrdered() select number * number;
```

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

The results are now ordered:

0 1 4 9 16 25 36 49 64 81

Using WithDegreeOfParallelism

By default, PLINQ uses the available processor cores, up to a maximum of 64. The goal, of course, is to keep the available processor cores busy with active work, as shown in the graph in Figure 6. The graph depicts 100% utilization, which is ideal. This graph was taken during a PLINQ query where the parallel clauses were compute bound.

You can explicitly set the maximum number of processor cores with the `WithDegreeOfParallelism` clause. There are two primary reasons for using this clause. First, this is useful when operations are I/O

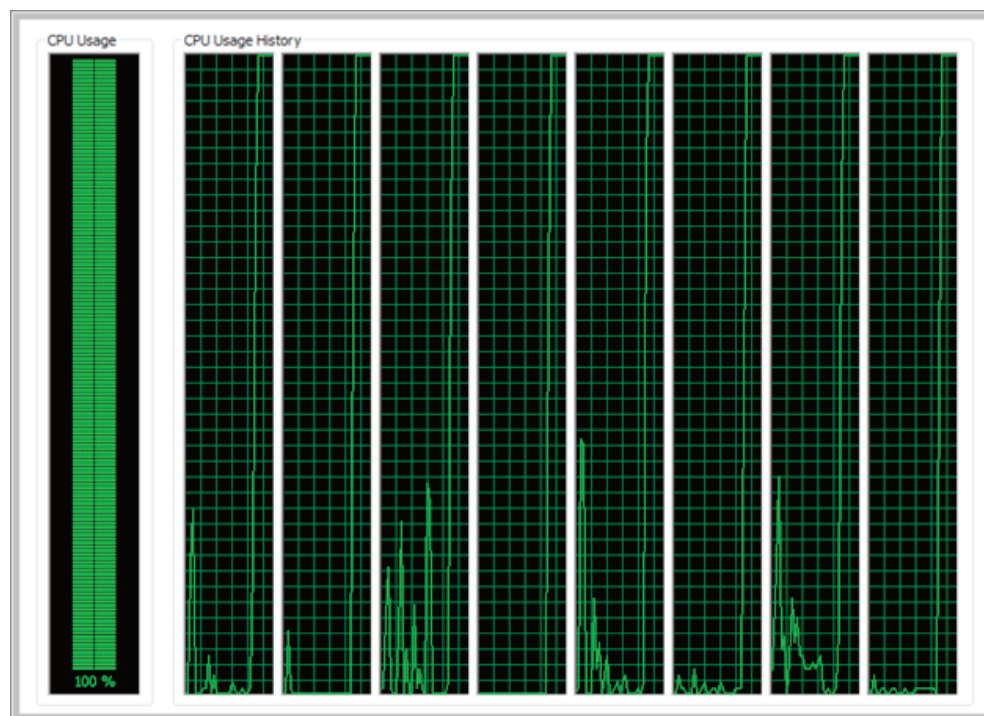


Figure 6: Full CPU utilization.

bound. I/O-bound threads are sometimes suspended, which causes processor cores to be underutilized. In this circumstance, you want to increase the degree of parallelism to exceed the number of processor cores. Conversely, you can decrease the number of tasks used in a PLINQ query with the `WithDegreeOfParallelism` clause. For example, you could create a more cooperative environment for other running applications by purposely reducing the degree of parallelism to less than the number of available cores.

Assuming eight available processor cores, the following code reduces the degree of parallelism. The amount of reduction depends on the number of available cores.

```
int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var result =
    numbers.AsParallel()
        .WithDegreeOfParallelism(2)
        .Select(number=>PerformSelect(number))
        .ToList();
```

Handling Exceptions

Unhandled exceptions raised in a PLINQ query are propagated to the joining thread. Because there can be one or more parallel tasks, multiple unhandled exceptions could occur concurrently. For that reason, unhandled exceptions are raised as an `AggregateException`. You can enumerate the `AggregateException.InnerExceptions` property to retrieve the original exceptions raised. This exception model is used in the following code to catch unhandled exceptions in PLINQ. In this example, you iterate an array of integers. Each element is then used as a divisor in a calculation. Unfortunately, a couple of the values in the array are zero. This throws the expected divide-by-zero exceptions. You will successfully catch and display the divide-by-zero exceptions.

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

Catch an exception in a PLINQ query as an AggregateException and display the results

1. Create a console application for C# in Visual Studio 2010. In the Main method, define an integer array. Notice that there are two zeros in the array list.

```
int [] intArray = { 5, 1, 2, 7, 4, 0, 6, 2, 9, 0 };
```

2. Perform a PLINQ query that iterates the integer array. In the Select clause, return a 1000/nth calculation.

```
var results = intArray.AsParallel()
    .Select(item => (int)1000 / (int) item);
```

3. Define a try and catch block. In the try block, iterate over the results by using a ForAll method.

```
try
{
    results.ForAll((item) => Console.WriteLine(item));
}
```

4. In the catch statement, catch an AggregateException. Then iterate over its InnerExceptions property and display the original underlying exceptions.

```
catch (AggregateException ex)
{
    foreach (var inner in ex.InnerExceptions)
    {
        Console.WriteLine(inner.Message);
    }
}
```

Here is the entire code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Exception
{
    class Program
    {
        static void Main(string[] args)
        {
            int [] intArray = { 5, 1, 2, 7, 4, 0, 6, 2, 9, 0 };
            var results = intArray.AsParallel()
                .Select(item => (int)1000 / (int) item);
            try
            {
                results.ForAll((item) =>
                    Console.WriteLine(item));
            }
            catch (AggregateException ex)
            {
                foreach (var inner in ex.InnerExceptions)
                {
                    Console.WriteLine(inner.Message);
                }
            }
            Console.WriteLine("Press enter to exit");
            Console.ReadLine();
        }
    }
}
```

Figure 7 shows the output from the application, which shows two unhandled exceptions. Why? There is one exception for each zero in the integer list. Each will cause an unhandled exception in a separate task.

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

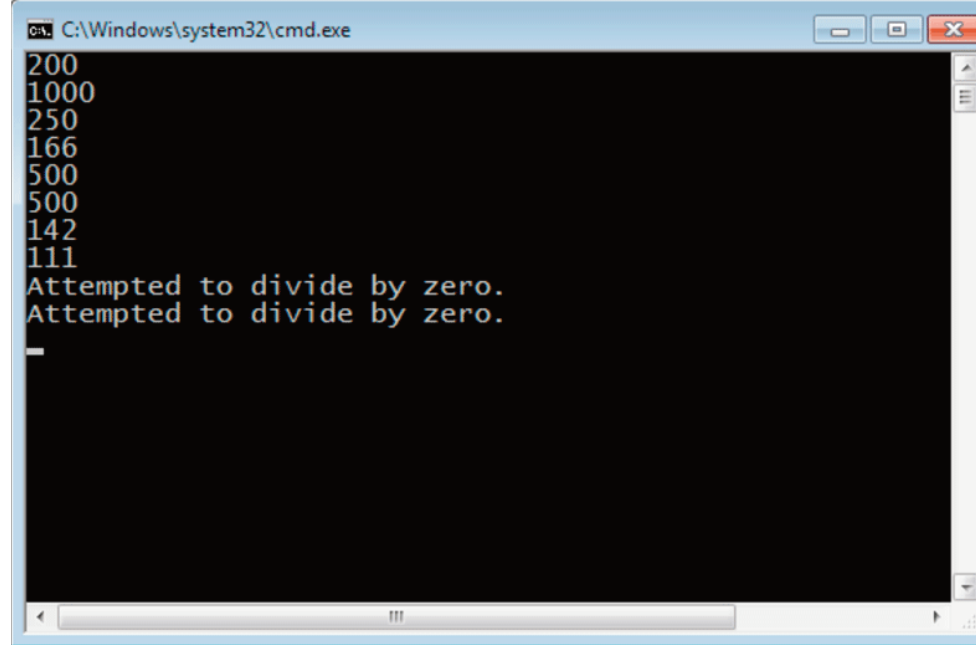


Figure 7: Output from the AggregateException example.

Cancellation

You cancel a PLINQ query with a `CancellationTokenSource` and `CancellationToken` object. `CancellationToken` is a property of the `CancellationTokenSource` class. Using the `WithCancellation` clause, you provide a cancellation token to a PLINQ query. You can then call `CancellationToken.Cancel` to cancel the query operation. When you cancel the operation, it throws an `OperationCanceledException` exception. Here are the basic steps of the cancellation model with PLINQ:

1. Define a `CancellationTokenSource`.
2. Execute the PLINQ query within a `try` block.

3. Add the `WithCancellation` clause to the PLINQ query. The only parameter is the `CancellationToken`.
4. Call the `CancellationTokenSource.Cancel` method to assert a cancellation. This will throw an `OperationCanceledException` in the PLINQ query.
5. In a `catch` block, catch and handle the `OperationCanceledException`.

Review the following example code. Note that it catches both the `OperationCanceledException` and `AggregateException`. A PLINQ query consists of parallel tasks, so it is quite possible for both an unhandled exception and a cancellation exception to occur on separate tasks of the same query. Therefore, you should be prepared to catch both exceptions. This example uses a separate thread to invoke the cancellation request.

```
static CancellationTokenSource cs
    = new CancellationTokenSource();
static void Main(string[] args)
{
    int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    (new Thread(new ThreadStart(Cancellation))).Start();
    try
    {
        var result = numbers.AsParallel()
            .WithCancellation(cs.Token)
            .Select(number => PerformSelect(number))
            .ToList();
    }
    catch (OperationCanceledException ex)
    {
        Console.WriteLine(ex.Message);
    }
    catch (AggregateException ex)
    {

```

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

```

        foreach (var inner in ex.InnerExceptions)
        {
            Console.WriteLine(inner.Message);
        }
    }
}

static void Cancellation()
{
    Thread.Sleep(4000); cs.Cancel();
}

```

Reduction

Reduction reduces a collection of elements to a singular value. For example, reduction could return an average value for a series of numbers. Other examples of reduction are calculating a summation, maximum value, or minimum value. In the .NET Framework, reduction is implemented as aggregation. LINQ provides a sequential implementation of aggregation, which runs on a single thread. This limits dependencies and is inherently thread-safe. LINQ provides explicit methods for the most common reductions, such as:

```

Sum
Average
Min
Max
Count

```

For other types of aggregation, you can implement a custom solution by using the `Aggregate` method.

The following is an example of reduction that uses LINQ. The reduction here is `Count`. It returns the number of positive values in a series of numbers.

```
var val=numbers.Where((number)=>number>-1).Count();
```

The next example is a custom aggregation and combines a collection of words as a sentence.

```

string [] words={
    "this",
    "is",
    "a",
    "string",
    "concatenation",
    "."
};
string punctuation = "., @# $ % ";
string delimiter=" ";
var sentence= words.Aggregate(
    (element1, element2) =>
    {
        delimiter =
            punctuation.Contains(element2) ? "" : " ";
        return element1 + delimiter + element2;
    }
);

```

The `Aggregate` method accepts a delegate as a parameter. The preceding example implements this delegate as a lambda expression. The delegate has two parameters: the partial result, and the next element of the collection. The return value is a partial result. But the parameters to the first `Aggregate` invocation are different. For that first call, the parameters are the first and second elements of the collection, which in this case would be the strings `this` and `is` from the string array. For subsequent iterations, the first parameter would contain the intermediate result.

Reduction in a PLINQ query merges the results from multiple threads. This creates a potential dependency on both the source collection and the result. For this reason, each thread uses thread-local

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

storage, which is non-shared memory, to cache partial results. When operations have completed, the separate partial results are combined into a final result. Like LINQ, PLINQ has the standard reductions, such as the `Sum`, `Average`, and `Count` methods. You simply need to add the `AsParallel` clause.

```
var val = numbers.AsParallel()
    .Where((number) => number > -1)
    .Count();
```

You can also perform custom aggregation by using the `Aggregate` method. The signature is different from the same method in LINQ. The following code is the simplest overload of the `ParallelEnumerable.Aggregate` method. The second parameter is the seed and is the first relevant parameter. Thread-local storage for threads used for caching partial results is initialized with the seed. The `updateAccumulatorFunc` function calculates the partial result for a thread. The `combineAccumulatorsFunc` function merges the partial results into a final result. The last parameter is `resultSelector`, which is used to perform a user-defined operation on the final results.

```
public static TResult Aggregate<TSource,
                                TAccumulate, TResult>(
    this ParallelQuery<TSource> source,
    TAccumulate seed,
    Func<TAccumulate, TSource, TAccumulate>
        updateAccumulatorFunc,
    Func<TAccumulate, TAccumulate, TAccumulate>
        combineAccumulatorsFunc,
    Func<TAccumulate, TResult> resultSelector
)
```

Next is the first example of a PLINQ query that uses the `Aggregate` method. Reduction is being used to calculate a factorial, which is an individual value. This example calculates the factorial of 5 (5!). The answer

is 120 (1 x 2 x 3 x 4 x 5). This example uses the `Enumerable.Range` method to return a collection of integer values ranging from 1 to 5. In the `Aggregate` method, the partial result of each thread is initialized to 1. If initialized to zero, the results would immediately be nullified, because the product of zero and anything else is zero.

```
int value=5;
var factorial=Enumerable.Range(1, value)
    .AsParallel()
    .Aggregate(1, (result, number)=>
        result*number, result=>result);
```

In the following exercise, we use the `Aggregate` method to calculate the dot product of two matrices. We then list all three matrices: the two input matrices and one result matrix.

Multiply corresponding points in two matrices by using the `Enumerable.Aggregate` method and save the product in a third matrix

1. Create a console application for C# in Visual Studio 2010. In the `Main` method, define a constant value of four. This is used to set the size of the one-dimensional matrices. In addition, declare and initialize two input matrices.

```
const int len = 4;
int[] first = new int [len] { 1, 2, 3, 4 };
int[] second = new int [len] { 5, 6, 7, 8 };
```

2. Starting at zero, create a range of integral values for indexes used with the matrices. Use the `AsParallel` clause to parallelize the `Aggregate` method.

```
var third=Enumerable.Range(0, first.Length).AsParallel()
```

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

3. You define an integer array as the first parameter, which is a reference. As such, each thread receives a reference to the same array. This is the result matrix and the initial value of each thread. Depending on how the array is used, this might not be thread-safe. In this circumstance, the different threads in the calculation will never access the same location in the result array. Therefore, it is thread-safe. The next parameter is a delegate. Insert a lambda expression that multiplies the two matrices, two elements at a time. The next parameter should merge the results of the various threads. Because the threads are sharing the same result matrix, a merge is not necessary. The final parameter is an action to be performed on the final result. In this example, there is no action.

```
.Aggregate(new int[len], (result, index) =>
{
    result.SetValue(first[index] * second[index], index);
    return result;
},
(total, subtotal) => total,
(result) => result);
```

4. Display the input and output matrices by using `Console.WriteLine`.

```
foreach (var element in first)
{
    Console.Write(element+" ");
}

Console.WriteLine();

foreach (var element in second)
{
    Console.Write(element+" ");
}
```

```
Console.WriteLine();
```

```
foreach (var element in third)
{
    Console.Write(element + " ");
}
```

Compile and run the program. Here is the entire application.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Matrix
{
    class Program
    {
        static void Main(string[] args)
        {
            const int len = 4;
            int[] first = new int [len] { 1, 2, 3, 4};
            int[] second = new int[len] { 5, 6, 7, 8};
            var third=Enumerable.Range(0, first.Length)
                .AsParallel().Aggregate(new int[len],
                    (result, index)=>
                    {
                        result.SetValue(first[index]*second[index],
                            index);
                        return result;
                    }, (result)=>result);

            foreach (var element in first)
            {
                Console.Write(element+" ");
            }

            Console.WriteLine();

            foreach (var element in second)
            {
```

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

```

        Console.Write(element+" ");
    }

    Console.WriteLine();

    foreach (var element in third)
    {
        Console.Write(element + " ");
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadLine();
}
}
}

```

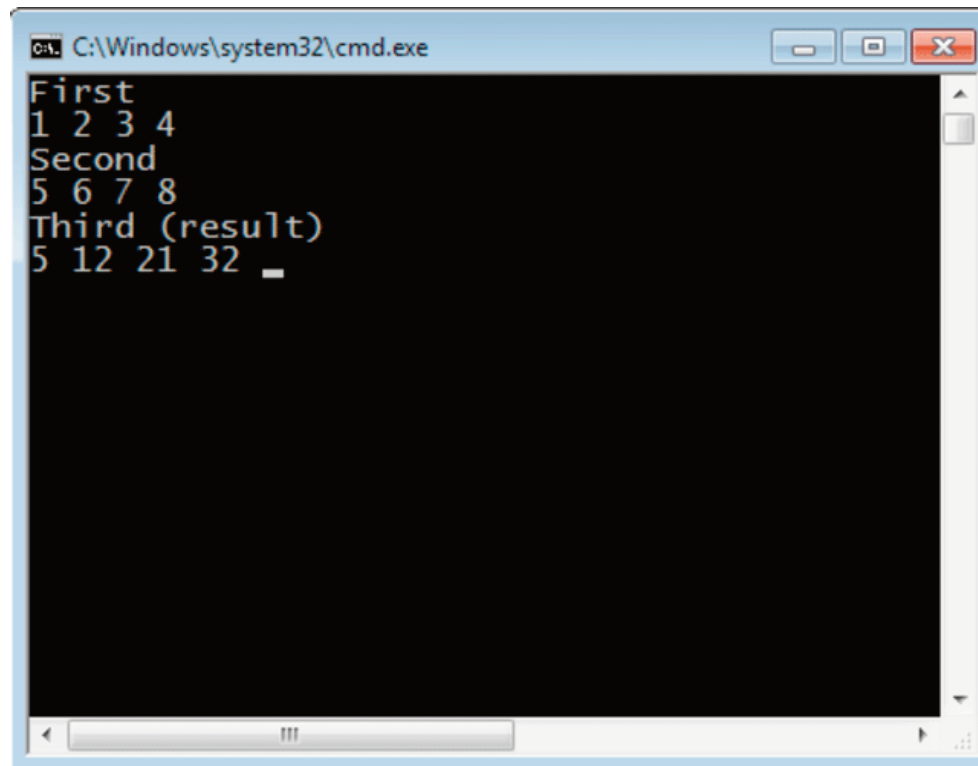


Figure 8: Calculating the dot product of two matrices.

Figure 8 shows the output for the application. It lists the three relevant matrices.

Using MapReduce with PLINQ

MapReduce is a special pattern of reduction used on large computer systems in which data collections can be distributed across a grid of computers. MapReduce allows you to perform an action on this highly dispersed collection of data. You probably do not own a large computer system with hundreds of computer nodes. Fortunately, the MapReduce pattern is also applicable and useful in smaller systems and multicore personal computers. MapReduce transforms a collection first to an ordered key-value pair, which is then reduced (typically based on the key) to a derivative collection.

Here are the basic steps of the MapReduce pattern in PLINQ:

1. Create or define the source, which must be a `ParallelQuery` type.
2. In the map operation, map the input source to an ordered and intermediate collection.
3. Finally, reduce the intermediate collection to the output collection.

PLINQ does not offer a direct implementation of the MapReduce pattern. Fortunately, Stephen Toub's "Parallel Programming with Microsoft .NET: Design Patterns for Decomposition and Coordination on Multicore Architectures" (<http://is.gd/MZ2NT2>) recommends a possible implementation of the MapReduce pattern in PLINQ. Here is the suggested implementation:

```

public static ParallelQuery<TResult>
    MapReduce<TSource, TMapped, TKey, TResult>(
        this ParallelQuery<TSource> source,

```

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

```
Func<TSource, IEnumerable<TMapped>> map,
Func<TMapped, TKey> keySelector,
Func<IGrouping<TKey, Tmapped>,
IEnumerable<TResult>
{
    return source.SelectMany(map)
        .GroupBy(keySelector)
        .SelectMany(reduce);
}
```

There are several parameters:

- The `source` parameter is the source collection.
- The `map` parameter is a delegate for the operation that maps the input collection to an intermediate collection.
- The `keySelector` parameter is a delegate for the operation that identifies the key. The intermediate collection is grouped on this key.
- The `reduce` parameter is also a delegate. It indicates where the reduction is performed — typically on the range of values associated with each key.

Here is code that reads Shakespearean sonnets and displays the word distribution, implemented with PLINQ. The source collection consists of four sonnets.

```
string [] files={
    @"C:\shakespeare\Sonnet 1.txt",
    @"C:\shakespeare\Sonnet 2.txt",
    @"C:\shakespeare\Sonnet 3.txt",
    @"C:\shakespeare\Sonnet 4.txt"
};
```

Define a PLINQ query with the collection of sonnets as the source. Call the PLINQ implementation of MapReduce to create a word count

and distribution from these files. The first parameter is a delegate implemented as a lambda expression. In the lambda expression, read the lines of text in each file, which are then split along delimiters into words. The next parameter identifies each word as a key. The result is an array of `KeyValuePair`s containing the word (key) and word count.

```
var counts = files.AsParallel().MapReduce(
    path => File.ReadLines(path).SelectMany(
        line => line.Split(delimiters)),
    word => word,
    group => new[] {
        new KeyValuePair<string,
            int>(group.Key, group.Count())
    }
);
```

You can now display the results.

```
foreach (var word in counts)
{
    Console.WriteLine(word.Key + " " + word.Value);
}
```

Compile and run the program. Here is the entire application.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace MapReduce
{
    static class PLINQ
    {
        public static ParallelQuery<TResult>
            MapReduce<TSource, TMapped, TKey, TResult>(
```

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

```

        this ParallelQuery<TSource> source,
        Func<TSource, IEnumerable<TMapped>> map,
        Func<TMapped, TKey> keySelector,
        Func<IGrouping<TKey, Tmapped>,
            IEnumerable<TResult>> reduce)
    {
        return source.SelectMany(map)
            .GroupBy(keySelector)
            .SelectMany(reduce);
    }
}

class Program
{
    static void Main(string[] args)
    {
        char [] delimiters={' ', ',', ';', '.'};
        string [] files={
            @"C:\shakespeare\Sonnet 1.txt",
            @"C:\shakespeare\Sonnet 2.txt",
            @"C:\shakespeare\Sonnet 3.txt",
            @"C:\shakespeare\Sonnet 4.txt"
        };
        var counts = files.AsParallel().MapReduce(
            path => File.ReadLines(path).SelectMany(
                line => line.Split(delimiters)),
            word => word,
            group => new[] {
                new KeyValuePair<string, int>(group.Key,
                    group.Count())
            }
        );
        foreach (var word in counts)
        {
            Console.WriteLine(word.Key + " " + word.Value);
        }
        //Console.WriteLine("Press enter to exit");
        Console.ReadLine();
    }
}

```

Figure 9 shows the partial results from the application.

```

praise 1
is 3
nothing 1
face 2
dies 1
This 1
prime 1
back 1
How 1
whose 1
bounteous 1
form 1
spend 1
new 1
thrifty 1
thysself 3
answer 1
he 1
sums 1
renewest 1

```

Figure 9: Partial output for the Shakespearean sonnets word-frequency application.

Wrapping Up

As an extension of LINQ, PLINQ implements data parallelism while supporting parallel queries across disparate data domains. PLINQ shares many of the same clauses and methods as LINQ. For this reason, if you are familiar with LINQ, you have the underlying knowledge of PLINQ. In most circumstances, just add the `AsParallel` clause to the LINQ query to transform it to PLINQ.

When iterating PLINQ results, use the `ParallelQuery.ForAll` method instead of the `for` and `foreach` methods. Serializing access to the results can negate some of the benefit to parallel processing. There are several important clauses in PLINQ:

- `ParallelExecutionMode` explicitly sets the PLINQ query to execute in parallel.

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

- `WithMergeOptions` sets the buffer mode for rendering the results of a PLINQ query.
- `AsSequential` mandates that subsequent LINQ clauses execute sequentially.
- `AsOrdered` orders the results (but the query itself is still executed in parallel).
- `WithDegreeOfParallelism` sets the number of concurrent tasks used in the PLINQ query.

Handle the `AggregateException` exception to catch exceptions raised in a PLINQ query. You can enumerate the `AggregateExceptions.InnerExceptions` attributes to inspect the underlying exceptions raised in one or more parallel tasks assigned to the PLINQ query.

You can cancel a PLINQ query by using the Cancellation model. Create an instance of the `CancellationTokenSource` and pass a cancellation token to the PLINQ query by using the `WithCancellation`

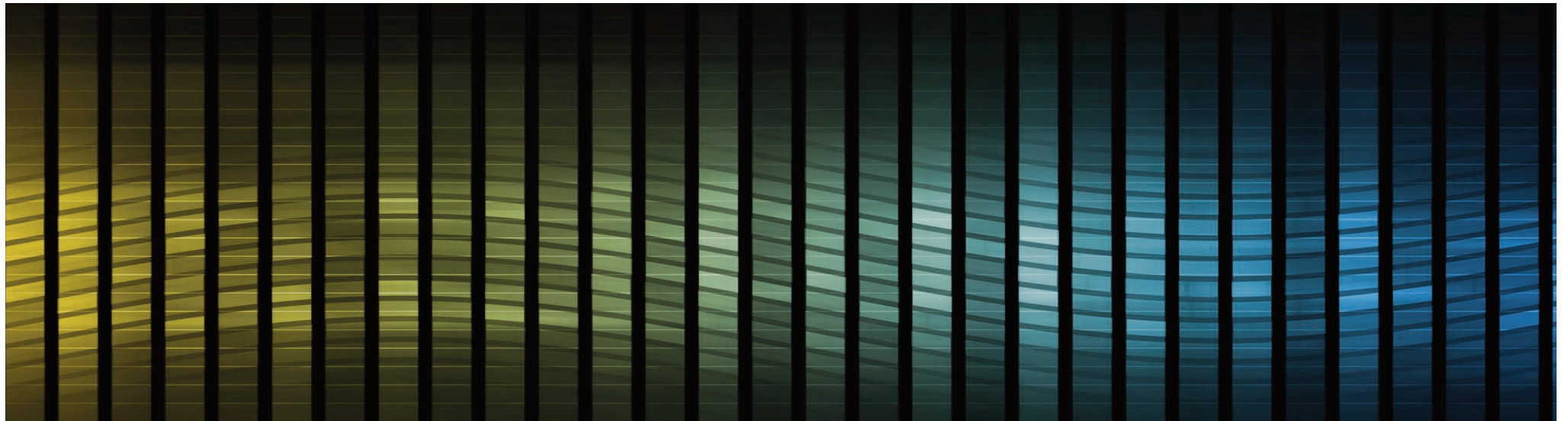
method. The PLINQ query can then be canceled with the `CancellationToken.Cancel` method.

Reduction reduces a collection to a value. PLINQ directly supports the most common reductions, such as `Sum`, `Average`, `Max`, `Min`, and `Count`. For a custom reduction operation, use the `ParallelEnumerable.Aggregate` method.

If you want, you can implement the MapReduce pattern for PLINQ.

For more on PLINQ, see “Parallel LINQ” at <http://drdobbs.com/go-parallel/article/212700663>.

This article was adapted from information in Donis Marshall's Parallel Programming with Microsoft Visual Studio 2010 Step by Step (<http://is.gd/KvH3gu>), published by Microsoft Press.

[Comment](#)

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

From the Vault

Concurrency Runtime (CRT): The Task Scheduler

This article from 2010 originally appeared on *Dr. Dobb's Go-Parallel* as part of a series on Visual Studio 2010. This installment walks you through scheduling and coordinating concurrent tasks at runtime. — DDJ

By Lahlali Issam

Visual C++ 2010 comes with new features and enhancements to simplify more native programming. The Concurrency Runtime (CRT), for instance, is a framework that simplifies parallel programming and helps you write robust, scalable, and responsive parallel applications. The CRT raises the level of abstraction so that you do not have to manage the infrastructure details that are related to concurrency. The Concurrency Runtime also enables you to specify scheduling policies that meet the quality of service demands of your applications.

Figure 1 presents the architecture of Concurrency Runtime Framework. In this article, I discuss the Task Scheduler layer and examine how it works internally. To do so, I use CppDepend (<http://www.cppdepend.com/>), an analysis tool that makes it easier for you to manage complex C/C++ (native, mixed, and COM) code base.

The Task Scheduler

The Task Scheduler schedules and coordinates tasks at runtime. A task is a unit of work that performs a specific job. The Task Scheduler manages the details that are related to efficiently scheduling tasks on computers that have multiple computing resources.

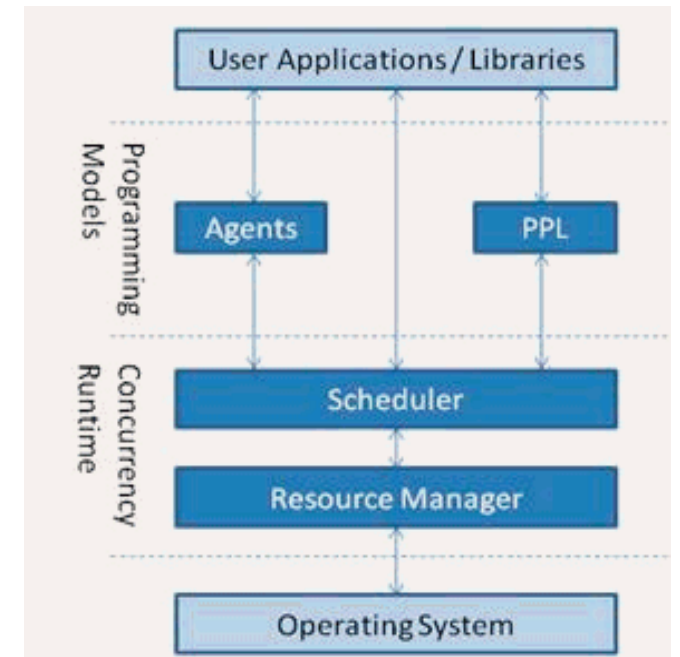


Figure 1

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

Windows provides a preemptive kernel-mode scheduler — a round-robin, priority-based mechanism that gives every task exclusive access to a computing resource for a given time period, then switches to another task. Although this mechanism provides “fairness” (every thread makes forward progress), it comes at some cost of efficiency. For example, many compute-intensive algorithms do not require fairness. Instead, it is important that related tasks finish in the least overall time. Cooperative scheduling enables an application to more efficiently schedule work.

Cooperative scheduling is a mechanism that gives every task exclusive access to a computing resource until the task finishes or until the task yields its access to the resource.

The user-mode cooperative scheduler enables application code to make its own scheduling decisions. Because cooperative scheduling enables many scheduling decisions to be made by the application, it reduces much of the overhead that is associated with kernel-mode synchronization.

The Concurrency Runtime (CRT) uses cooperative scheduling together with the preemptive scheduler of the operating system to achieve maximum usage of processing resources. In this article I examine the Task Scheduler design and lift its hood to see how it works internally. For information on the CRT Resource Manager, see Concurrency Runtime: The Resource Manager (<http://is.gd/D7zx6h>). Again, I use CppDepend to analyze the CRT source code.

Scheduler Design

The CRT provides the interface `Scheduler` to implement a specific scheduler adapted to application needs. Let’s examine classes that implement this interface:

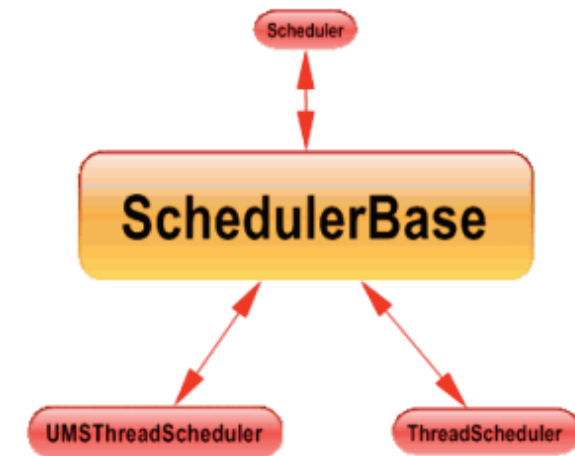


Figure 2

The CRT provides two implementations of the scheduler — `ThreadScheduler` and `UMSThreadScheduler`. As illustrated in the dependency graph in Figure 2, the `SchedulerBase` contains all common behavior of these two classes.

Is the `Scheduler` flexible? A good indicator of flexibility is to search for all abstract classes used by the `Scheduler`.

As shown in the dependency graph in Figure 3, the `Scheduler` uses many abstract classes. It enforces low coupling, and makes the scheduler more flexible, so adapting it to other needs is easy. To explain the role of each abstract class used by the `Scheduler`, I’ll discuss its responsibilities.

There are three major responsibilities assigned to the Task Scheduler:

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

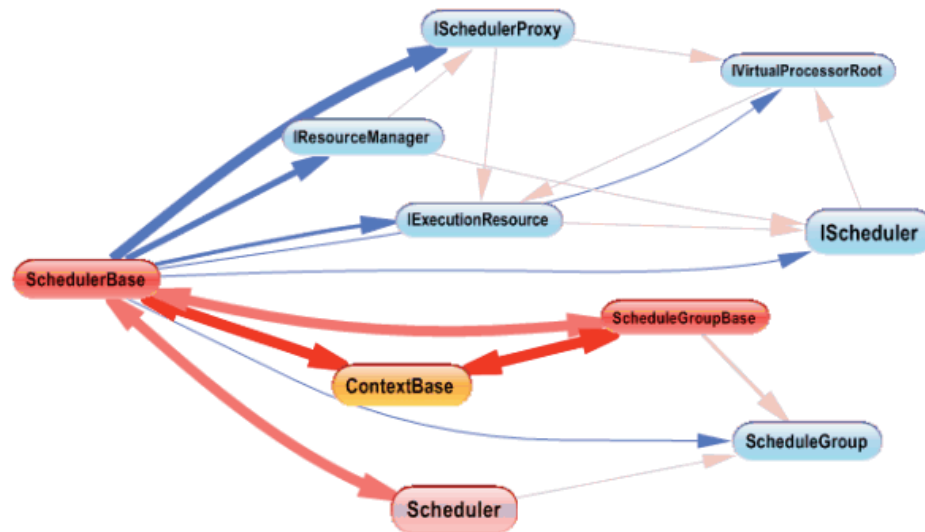


Figure 3

1. Getting resources (processors, cores, memory). When the scheduler is created, it asks for resources from the runtime Resource Manager, as explained in “CRT Concurrency Runtime: Resource Manager” (<http://is.gd/D7zx6h>). The `Scheduler` communicates with Resource Manager using `IResourceManager`, `ISchedulerProxy`, and `IScheduler` interfaces. Resources given by Resource Manager use scheduler policy to allocate resources to the `Scheduler`.

The policy as shown in Figure 4 is assigned when the `Scheduler` is created.

The CRT creates a default `Scheduler` if no `Scheduler` exists by invoking the `GetDefaultScheduler` method, and a default policy is used. The Task Scheduler enables applications to use one or more `Scheduler` instances to schedule work, and an application can invoke `Scheduler::Create` to add another `Scheduler` that uses a specific policy.

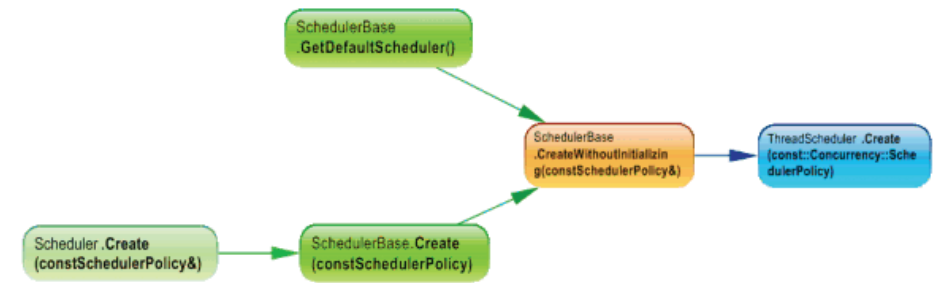


Figure 4

The `Concurrency::PolicyElementKey` enumeration defines the policy keys that are associated with the Task Scheduler (Figure 5).

For more information on policy keys, see “Scheduler Policies” (<http://help.outlook.com/en-us/140/ff829269.aspx>).

The following collaborations between the `Scheduler` and

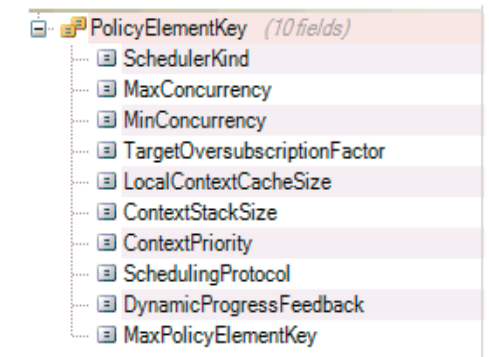


Figure 5

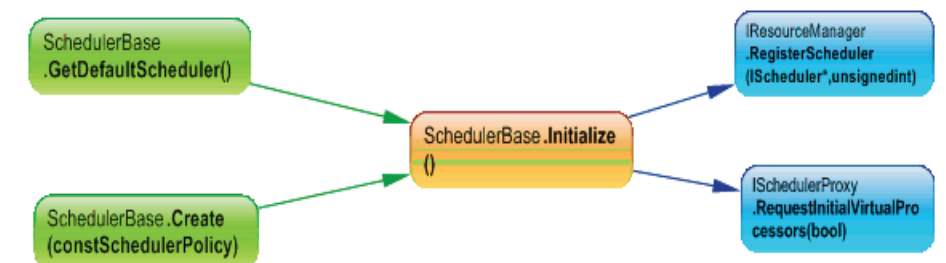


Figure 6

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

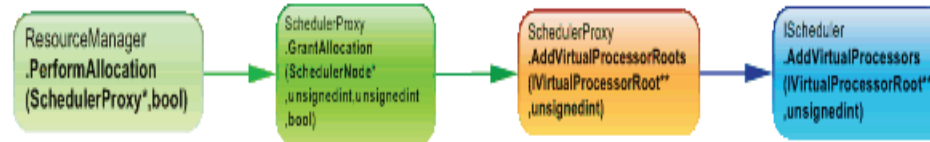


Figure 7

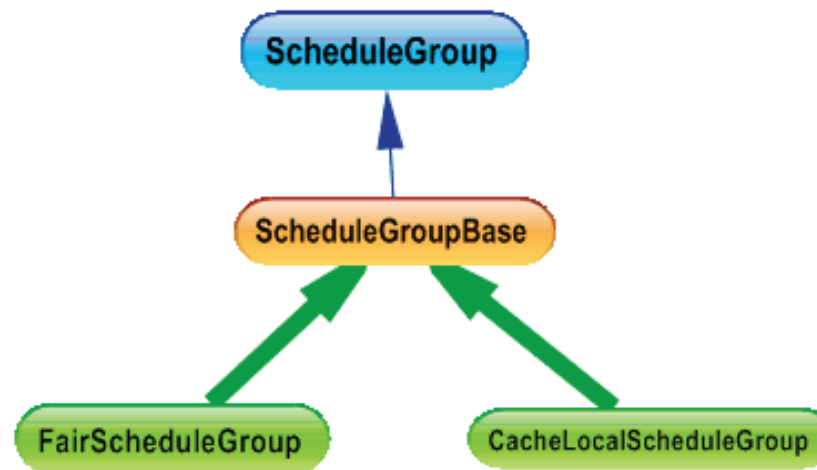


Figure 8

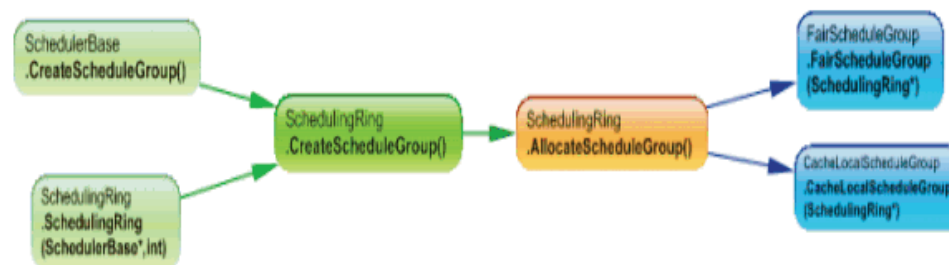


Figure 9

Resource Manager show the role of each interface concerned by the allocation.

- Ask for resource allocation (Figure 6):
- Getting resources from Resource Manager (Figure 7):

2. Managing task queues. When `Scheduler` is created, tasks can be assigned to it to be executed. The `Scheduler` stores tasks into queues to enforce the cohesion of classes — the queues are not managed directly by the `ThreadScheduler` class, but by `ScheduleGroupBase` class.

A schedule group affinitizes, or groups, related tasks together. Every `Scheduler` has one or more schedule groups. Use schedule groups when you require a high degree of locality among tasks; for example, when a group of related tasks benefit from executing on the same processor node.

In Figure 8, the runtime provides two kinds of `ScheduleGroup`: `FairScheduleGroup` and `CacheLocalScheduleGroup`, choosing between those two groups will be explained later.

Every `Scheduler` has a default schedule group for every scheduling node. The runtime creates one scheduling node for every processor package or Non-Uniform Memory Architecture (NUMA) node. If you do not explicitly associate a task with a schedule group, the `Scheduler` chooses which group to add the task to.

In Figure 9, the `schedulingRing` is responsible for managing `ScheduleGroups`. It contains a list of groups and creates them.

The `Schedule Group` contains three kind of queues:

- FIFO Queue
- Work-stealing Queue
- Unblocked Queue

IN THIS ISSUE

- Editorial >>
- ASP.NET >>
- PLINQ >>
- CRT >>
- Letters >>
- Links >>
- Table of Contents >>

methods	DepthOfIsUsing "Concurrency.d
Query target	
m_realizedChores	0
3 methods matched	
ScheduleGroupBase (3 methods)	
ScheduleTask(TaskProc,void*)	1
HasRealizedChores()	1
GetRealizedChore()	1

Figure 10

useful when you adapt existing code to use the scheduling functionality of the CRT.

A lightweight task is represented by the `RealizedChore` class, and the FIFO queue of the schedule group is represented by the field `m_realizedChores`. Let's search all methods using this queue (Figure 10).

Thus, we can add a lightweight task to the group by invoking `ScheduleGroupBase::ScheduleTask` or `Scheduler::ScheduleTask`. (See Figure 11.)

Next up, the work-stealing queue. While there's only one FIFO queue associated with the schedule group, the schedule group references a list of work stealing queues — for each worker thread, there's a local queue associated with it.



Figure 11

The FIFO Queue contains lightweight tasks. A lightweight task resembles the function that you provide to the Windows API `CreateThread` function. Therefore, light-

weight tasks are

A thread that is attached to a scheduler is known as an “execution context,” or just “context,” so actually this local queue is associated to `Context` class. The `Context` class provides a programming abstraction for an execution context and offers the ability to cooperatively block, unblock, and yield the current context.

And to be sure that only `Context` creates this kind of queue, you can directly access the `m_workQueues` field; see Figure 12.

So the `Context` is responsible for creating this queue, and for each context there's a local work-stealing queue associated to it. To illustrate the behavior of the work-stealing algorithm, imagine that you have two worker threads allocated to the scheduler. As explained before, for each worker thread there's a local queue associated with it; see Figure 13.

Imagine that you have three tasks in the queue of the

methods	DepthOfIsUsing "Concurrency.details.Sch
Query target	
m_workQueues	0
7 methods matched	
ContextBase (3 methods)	
CreateWorkQueue()	1
CreateStructuredWorkQueue()	1
ReleaseWorkQueue()	1
ScheduleGroupBase (4 methods)	
~ScheduleGroupBase()	1
RetireDetachedQueue(WorkQueue*)	1
HasUnrealizedChores()	1
StealUnrealizedChore()	1

Figure 12

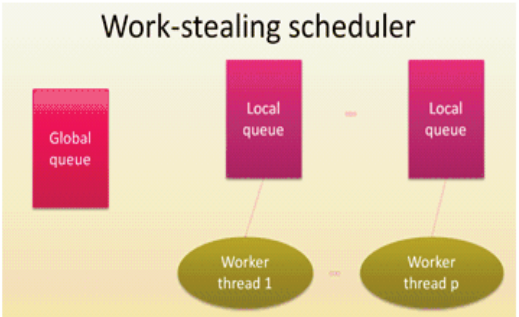


Figure 13

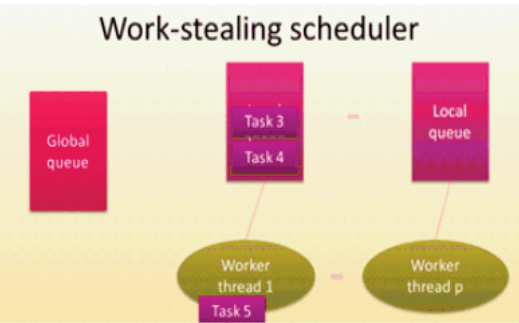


Figure 14

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)



Figure 15

the worker thread available; see Figure 15.

How can you create a task managed by a work-stealing queue?

To answer this question, let's examine the methods indirectly invoking the `CreateWorkQueue` method. As shown in Figure 16, this kind of task could be created by using the `task_group` class.

Using `task_group` to add a new task is better than using `Scheduler::ScheduleTask` to create lightweight tasks; indeed, the work-stealing algorithm better optimizes the use of virtual processors allocated to the `Scheduler`. However, `ScheduleTask` could be useful for migrating from the existing code with the `CreateThread` API.

Finally, there is the unblocked context queue. The `Context` class lets you block or yield the current execution context. Blocking or yielding is useful when the current context cannot continue because a resource



Figure 16

Worker Thread 1 — Tasks 3 and 4 are waiting to be executed while Task 5 is running; see Figure 14.

The `Dispatch` method finds that there is nothing in the queue, so Task 3 is moved, or "stolen" from its original queue, to be distributed to

is not available. The `Context::Block` method blocks the current context. A context that is blocked yields its processing resources so that the runtime can perform other tasks. The `Context::Unblock` method unblocks a blocked context.

When a context is unblocked and it's available to be executed, it's added to the runnable context queue. This queue is represented by the field `m_runnableContexts`.

Figure 17 shows some cases where the context is added to the runnable queue. The context is added to the queue when it's unblocked or a virtual processor is retired from the `Scheduler`.

3. Dispatching Tasks. The `Scheduler` tries to search for work to execute. "Work" can be:

- Unblocked context
- Lightweight tasks
- Tasks in work-stealing queues

Again, all these are stored in queues managed by Schedule Groups, and each group is managed by a scheduling ring.

When a virtual processor is allocated to the scheduler, a `ThreadProxy` class is created and associated with this processor. After the creation, the `Dispatch` method of the `ThreadProxy` is invoked.

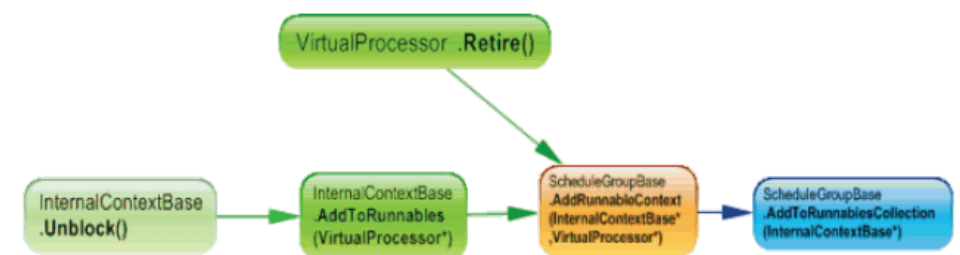


Figure 17

IN THIS ISSUE

[Editorial >>](#)
[ASP.NET >>](#)
[PLINQ >>](#)
[CRT >>](#)
[Letters >>](#)
[Links >>](#)
[Table of Contents >>](#)

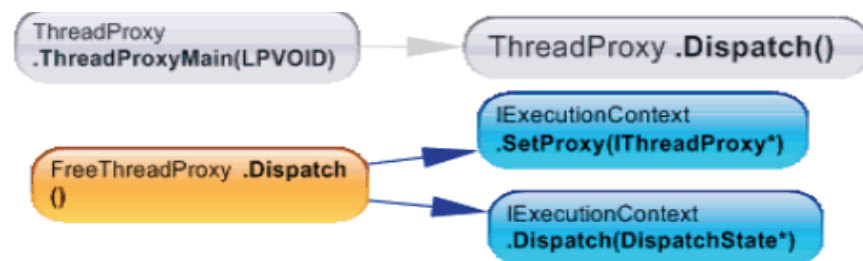


Figure 18



Figure 19

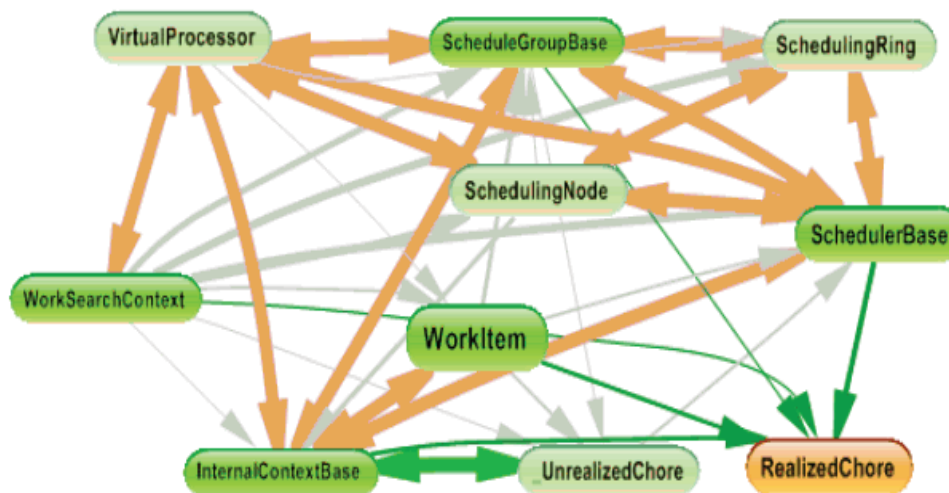


Figure 20

As shown in Figure 18, the CRT uses abstract classes to enforce low coupling, and the real dispatch invoked depends of the implementation chosen by the runtime — this choice is made according to the scheduler policy.

The concrete implementation of `Dispatch` invokes the `Dispatch` method of the execution context.

Figure 19 shows the all the methods invoked by concrete implementation of `Context::Dispatch`.

So the algorithm for searching the next task to execute is implemented by the `WorkSearchContext` class. Let's uncover all classes used directly by `WorkSearchContext` to achieve its goal; see Figure 20.

The responsibility of `WorkSearchContext` is to give us a `WorkItem` to execute; it could be `InternalContextBase`, `RealizedChore`, or `UnrealizedChore`. To understand the collaboration between these classes, let's search for methods directly used by `WorkSearchContext`; see Figure 21.

methods	DepthOfUse "Concurrency."
23 methods matched	N/A
GlobalMembers (2 methods)	
ScheduleGroupBase (3 methods)	
GetRunnableContext()	1
GetRealizedChore()	1
StealUnrealizedChore()	1
SchedulerBase (4 methods)	
GetNextSchedulingRing()	1
GetNextSchedulingRing(const SchedulingRing*)	1
GetFirstSchedulingNode(int*)	1
GetNextSchedulingNode(int*, int)	1
SchedulingNode (3 methods)	
GetScheduler()	1
GetFirstVirtualProcessor(int*)	1
GetNextVirtualProcessor(int*, int)	1
SchedulingRing (4 methods)	
WorkItem (1 method)	
VirtualProcessor (6 methods)	
UpdateRamblingState(bool, SchedulingRing*)	1
GetOwningNode()	1
GetOwningRing()	1
PreRunnableSearch()	1
StealLocalRunnableContext()	1
GetLocalRunnableContext()	1
Sum	23

Figure 21

IN THIS ISSUE

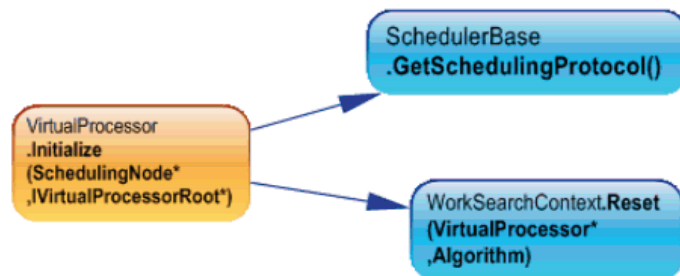
[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

Figure 22

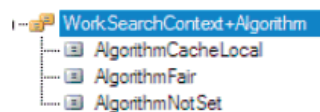


Figure 23

The WorkSearchContext iterates on SchedulingRing and ScheduleGroup classes via SchedulerBase methods. And for each ScheduleBase, you search for RunnableContext, realizedChore, or UnrealizedChore.

The WorkSearchContext class is created by the VirtualProcessor class, and as shown in Figure 22, the algorithm used is specified when the VirtualProcessor is initialized, and it asks the Scheduler for the SchedulingProtocol, which describes the scheduling algorithm that will be used for the Scheduler.

WorkSearchContext will discover the algorithm to use by passing it a value from Algorithm enum; see Figure 23.

So two algorithms are implemented by this class to find work.

The first is the Cache Local algorithm. This algorithm will look for runnable context within the current schedule group, then realized chores and unrealized chores. If there's no more work in the current schedule group, it looks for the next group in the same schedule ring,. And when it finishes all work in the current schedule ring, it looks in the next schedule ring.



Figure 24



Figure 25

So the scheduler prefers to continue to work on tasks within the current schedule group before moving to another schedule group.

This algorithm is implemented by the WorkSearchContext::SearchCacheLocal method, and as shown by this dependency graph, this method search invokes other methods to search for runnable contexts, RealizedChore or _UnrealizedChore; see Figure 24.

Another specificity of this algorithm is that the unblocked contexts are cached per virtual-processor and are typically scheduled in a last-in first-out (LIFO) fashion by the virtual processor that unblocked them. To verify this behavior, here's a dependency graph of methods invoked when searching for runnable context; see Figure 25.

This algorithm is the default algorithm used by the Scheduler if one is not specified.

The second algorithm is the Fair algorithm. In this case, the scheduler prefers to round-robin through schedule groups after executing each

IN THIS ISSUE

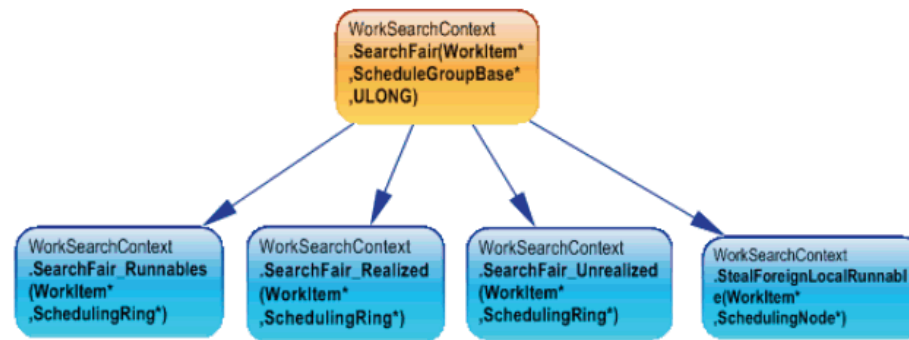
[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

Figure 26

task. Unblocked contexts are typically scheduled in a first-in-first-out (FIFO) fashion. Virtual processors do not cache unblocked contexts.

This algorithm is implemented by the `WorkSearchContext::SearchFair` method, and as shown by this dependency graph, this method search invokes other methods to search for runnable contexts, `RealizedChore` or `_UnrealizedChore`; see Figure 26.

From this analysis, it is fair to say that the CRT Task Scheduler appears to be well-designed, is characterized by a high cohesion and very low coupling, is very flexible, and provides by default an optimized algorithm to exploit better hardware resources.

— *Lahlali Issam is the developer of CppDepend code analysis tool for C/C++.*

[Comment](#)

IN THIS ISSUE

[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)

This Month on DrDobbs.com

Interesting items posted on
www.drdobbs.com over the past
month that you may have missed

WELCOME TO THE PARALLEL JUNGLE

Herb Sutter dives into the repercussions of parallel's reach from mobile devices, to the desktop, to clusters, and — at the highest level of granularity — to the cloud. This welter of different parallel implementations presents significant challenges for programming. The free lunch of sequential programming is well and truly over.

<http://drdobbs.com/parallel/232400273>

BY ANY OTHER NAME

The Arduino isn't a bad choice for its intended market segment. Al Williams just wishes it didn't teach bad habits.

<http://drdobbs.com/blogs/embedded-systems/232500607>

LANGUAGE OF THE MONTH: JRUBY

Ahead-of-time compilation, metaprogramming Java objects, real threads, and the ability to call Java directly from Ruby code are all ways in which JRuby improves on the standard Ruby distribution.

<http://drdobbs.com/jvm/232400456>

BOOTING AN INTEL ARCHITECTURE SYSTEM, PART I: EARLY INITIALIZATION

The boot sequence today is far more complex than it was even a decade ago. Here's a detailed, low-level, step-by-step walkthrough of the boot up.

<http://drdobbs.com/parallel/232300699>

LOW-POWER ARM-BASED BUILD SERVER DELIVERS

Nvidia's low-power, low-cost, ARM-based Trim-Slice is easy to set up and run, and it delivers as a build server.

<http://drdobbs.com/tools/232400398>

THE RISE AND FALL OF PROGRAMMING LANGUAGES IN 2011

Last year marked many changes in programming: Mobile devices emerged as a major programming platform and, at the other end of the spectrum, clouds became an established platforms for data and applications.

<http://drdobbs.com/mobile/232400093>

IN THIS ISSUE

- Editorial >>
- ASP.NET >>
- PLINQ >>
- CRT >>
- Letters >>
- Links >>
- Table of Contents >>

Dr.Dobb's

Andrew Binstock Editor in Chief, Dr. Dobb's
alb@drdobbs.com

Deirdre Blake Managing Editor, Dr. Dobb's
dblake@techweb.com

Amy Stephens Copyeditor, Dr. Dobb's
astephens@techweb.com

Sean Coady Webmaster, Dr. Dobb's
scoady@techweb.com

J.D.Hildebrand Editor at Large, Dr. Dobb's
jdhildebrand@drdobbs.com

Jon Erickson Editor in Chief Emeritus , Dr. Dobb's

CONTRIBUTING EDITORS

Scott Ambler
Mike Riley
Herb Sutter

DR DOBB'S
UBM TECHWEB
303 Second Street,
Suite 900, South Tower
San Francisco, CA 94107
1-415-947-6000

INFORMATIONWEEK

Rob Preston VP and Editor In Chief, Information-Week
rpreston@techweb.com 516-562-5692

John Foley Editor, InformationWeek
jpfoley@techweb.com 516-562-7189

Chris Murphy Editor, InformationWeek
cjmurphy@techweb.com 414-906-5331

Art Wittmann VP and Director, Analytics, InformationWeek
awittmann@techweb.com 408-416-3227

Alexander Wolfe Editor In Chief, Information-Week.com
awolfe@techweb.com 516-562-7821

Stacey Peterson Executive Editor, Quality, InformationWeek
speterson@techweb.com 516-562-5933

Lorna Garey Executive Editor, Analytics, InformationWeek
lgarey@techweb.com 978-694-1681

Stephanie Stahl Executive Editor, Information-Week
stahl@techweb.com 703-266-6030

Fritz Nelson VP and Editorial Director
fnelson@techweb.com 949-223-3608

David Berlind Chief Content Officer, TechWeb
dberlind@techweb.com 978-462-5315

REPORTERS

Charles Babcock Editor At Large
Open source, infrastructure, virtualization
cbabcock@techweb.com 415-947-6133

Thomas Claburn Editor At Large
Security, search, Web applications
tclaburn@techweb.com 415-947-6820

Paul McDougall Editor At Large
Software, IT services, outsourcing
pmcdougall@techweb.com

Marianne Kolbasuk McGee Senior Writer IT
management and careers
mmcgee@techweb.com 508-697-0083

J. Nicholas Hoover Senior Editor
Desktop software, Enterprise 2.0,
collaboration
nhoover@techweb.com 516-562-5032

Andrew Conry-Murray New Products and Business Editor
Information and content management
acmurray@techweb.com 724-266-1310

W. David Gardner News Writer
Networking, telecom
wdavidg@earthlink.net

Antone Gonsalves News Writer
Processors, PCs, servers
antoneg@pacbell.net

Eric Zeman
Mobile and Wireless
eric@zemanmedia.com

CONTRIBUTORS

Michael Biddick mbiddick@nwc.com
Michael A. Davis mdavis@nwc.com
Jonathan Feldman jfeldman@nwc.com
Randy George rgeorge@nwc.com
Michael Healey mhealey@nwc.com

EDITORS

Jim Donahue Chief Copy Editor
jdonahue@techweb.com

ART/DESIGN

Mary Ellen Forte Senior Art Director
mforte@techweb.com

Sek Leung Senior Designer
sleung@techweb.com

INFORMATIONWEEK ANALYTICS
analytics.informationweek.com

Art Wittmann VP and Director
awittmann@techweb.com 408-416-3227

Lorna Garey Executive Editor, Analytics
lgarey@techweb.com 978-694-1681

Heather Vallis Managing Editor, Research
hvallis@techweb.com 508-416-1101

INFORMATIONWEEK.COM

Benjamin Tomkins Managing Editor
btomkins@techweb.com 516-562-5336

Roma Nowak Senior Director,
Online Operations and Production
rnowak@techweb.com 516-562-5274

Tom LaSusa Managing Editor,
Newsletters
tlasusa@techweb.com

Jeanette Hafke Web Production Manager
jhafke@techweb.com

Joy Culbertson Web Producer
jculbertson@techweb.com

Nevin Berger Senior Director,
User Experience
nberger@techweb.com

Steve Gilliard Senior Director,
Web Development
sgilliard@techweb.com

Copyright 2012 United Business
Media LLC. All rights reserved.



INFORMATIONWEEK
ADVISORY BOARD

Dave Bent
Senior VP and CIO
United Stationers

Robert Carter
Executive VP and CIO
FedEx

Michael Cuddy
VP and CIO
Toromont Industries

Laurie Douglas
Senior CIO
Publix Super Markets

Dan Drawbaugh
CIO
University of Pittsburgh
Medical Center

Jerry Johnson
CIO
Pacific Northwest National
Laboratory

Kent Kushar
VP and CIO
E.&J. Gallo Winery

Carolyn LEclipse CDTon
Director, E-Services
California Office of the CIO

Jason Marchnard
Managing Director
Wells Fargo Securities

Randall Mott
Sr. Executive VP and CIO
Hewlett-Packard

Denis O'Leary
Former Executive VP
Chase.com

Mykolas Rambus
CEO
Wealth-X

M.R. Rangaswami
Founder
Sand Hill Group

Manjit Singh
CIO
Las Vegas Sands

David Smoley
CIO
Flextronics

Ralph J. Szygenda
Former Group VP and CIO
General Motors

Peter Whatnell
CIO
Sunoco

UBM TECHWEB

Tony L. Uphoff CEO

John Dennehy CFO

David Michael CIO

Joseph Braue Sr. VP,
Light Reading
Communications Network

Scott Vaughan CMO

Ed Grossman Executive
Vice President, Information-
Week Business Technology
Network

John Ecke VP and Group
Publisher, Financial
Technology Network,
InformationWeek
Government, and
InformationWeek
Healthcare

Martha Schwartz EVP,
Group Sales,
InformationWeek Business
Technology Network

Beth Rivera Senior VP,
Human Resources

Eric Lundquist VP and Edi-
torial Analyst, Information-
Week Business Technology
Network

David Berlind
Chief Content Officer,
TechWeb, and Editor in
Chief, TechWeb.com

Fritz Nelson VP and
Editorial Director,
InformationWeek Business
Technology Network, and
Executive Producer,
TechWeb TV

UNITED BUSINESS
MEDIA LLC

Pat Nohilly Sr. VP, Strategic
Development
and Business Administration

Marie Myers Sr. VP,
Manufacturing

INFORMATIONWEEK
VIDEO

informationweek.com/tv

Fritz Nelson Executive
Producer
fnelson@techweb.com

INFORMATIONWEEK
BUSINESS
TECHNOLOGY
NETWORK

DarkReading.com

Security
Tim Wilson, Site Editor
wilson@darkreading.com

IntelligentEnterprise.com

App Architecture
Doug Henschen,
Editor in Chief
dhenschen@techweb.com

NetworkComputing.com
Networking , Communica-
tions, and Storage
Mike Fratto, Site Editor
mfratto@techweb.com

PlugIntoTheCloud.com
Cloud Computing
John Foley, Site Editor
jpfoley@techweb.com

InformationWeek SMB
Technology for Small
and Midsize Business
Benjamin Tomkins,
Site Editor
btomkins@techweb.com

Dr. Dobb's
Good Stuff for Serious
Developers
Andrew Binstock
Editor in Chief
alb@drdobbs.com

IN THIS ISSUE[Editorial >>](#)[ASP.NET >>](#)[PLINQ >>](#)[CRT >>](#)[Letters >>](#)[Links >>](#)[Table of Contents >>](#)**DR. DOBB'S**

Sales Director, Michele Hurabiell
(415) 378-3540, mhurabiell@techweb.com

INFORMATIONWEEK BUSINESS TECHNOLOGY NETWORK

CMO, Scott Vaughan
(949) 223-3662, svaughan@techweb.com

EVP of Group Sales, InformationWeek Business Technology Network, Martha Schwartz
(212) 600-3015, mschwartz@techweb.com

Publisher's Assistant, Esther Rodriguez
(949) 223-3656, erodriguez@techweb.com

SALES CONTACTS—WEST

Western U.S. (Pacific and Mountain states) and Western Canada (British Columbia, Alberta)

Inside Sales Manager, Vesna Beso
(415) 947-6104, vbeso@techweb.com

Strategic Accounts

Account Director, Sandra Kupiec
(415) 947-6922, skupiec@techweb.com

Sales Assistant, Matthew Cohen-Meyer
(415) 947-6214, mmeyer@techweb.com

MARKETING

VP, Marketing, Winnie Ng-Schuchman
(631) 406-6507, wng@techweb.com

Marketing Manager, Monique Luttrell
(949) 223-3609, mluttrell@techweb.com

Marketing Director, Angela Lee-Moll
(516) 562-5803, aleemoll@techweb.com

Director, Client Marketing, Michelle Somers
(516) 562-7928, msomers@techweb.com

SALES CONTACTS—EAST

Midwest, South, Northeast U.S. and Eastern Canada (Saskatchewan, Ontario, Quebec, New Brunswick)

District Manager, Jenny Hanna
(516) 562-5116, jhanna@techweb.com

District Manager, Michael Greenhut
(516) 562-5044, mgreenhut@techweb.com

Account Manager, Cori Gordon
(516) 562-5181, cgordon@techweb.com

Inside Sales Manager East, Ray Capitelli
(212) 600-3045, rcapitelli@techweb.com

Sales Assistant, Elyse Cowen
(212) 600-3051, ecowen@techweb.com

Strategic Accounts

District Manager, Mary Hyland
(516) 562-5120, mhyland@techweb.com

Account Manager, Tara Bradeen
(212) 600-3387, tbradeen@techweb.com

Account Manager, Jennifer Gambino
(516) 562-5651, jgambino@techweb.com

Sales Assistant, Kathleen Jurina
(212) 600-3170, kjurina@techweb.com

Dr.Dobb's

Sales Director, Michele Hurabiell
(415) 378-3540, mhurabiell@techweb.com

AUDIENCE DEVELOPMENT

Director, Karen McAleer
(516) 562-7833, kmcaleer@techweb.com

BUSINESS OFFICE

General Manager, Marian Dujmovits

United Business Media LLC
600 Community Drive
Manhasset, N.Y. 11030 (516) 562-5000
Copyright 2012. All rights reserved.

Entire contents Copyright © 2012, Techweb/United Business Media LLC, except where otherwise noted. No portion of this publication may be reproduced, stored, transmitted in any form, including computer retrieval, without written permission from the publisher. All Rights Reserved. Articles express the opinion of the author and are not necessarily the opinion of the publisher. Published by Techweb, United Business Media, 303 Second Street, Suite 900 South Tower, San Francisco, CA 94107 USA 415-947-6000.

UBM TECHWEB

Tony L. Uphoff CEO

John Dennehy CFO

David Michael CIO

Joseph Braue Sr.VP, Light Reading Communications Network

Scott Vaughan CMO

Ed Grossman Executive Vice President, InformationWeek Business Technology Network

John Ecke VP and Group Publisher, Financial Technology Network, InformationWeek Government, InformationWeek Healthcare

Martha Schwartz EVP, Group Sales, InformationWeek Business Technology Network

Beth Rivera Senior VP, Human Resources

David Berlind Chief Content Officer, TechWeb, and Editor in Chief, TechWeb.com

Fritz Nelson VP, Editorial Director, InformationWeek Business Technology Network, and Executive Producer, TechWeb TV

Eric Lundquist VP and Editorial Analyst, InformationWeek Business Technology Network

UNITED BUSINESS MEDIA LLC

Pat Nohilly Sr.VP, Strategic Development and Business Admin.

Marie Myers Sr.VP, Manufacturing

