HOW DO DEVELOPERS USE PARALLEL LIBRARIES?

*To Appear @ FSE’12

Semih Okur & Danny Dig

University of Illinois at Urbana-Champaign
PARALLEL COMPUTING IS EVERYWHERE NOW
PARALLEL PROGRAMMING IS HARD

[From “The Practice of Parallel Programming” book cover]
PARALLEL LIBRARIES ARE WIDESPREAD

- C, C++: OpenMP, OpenMPI, Intel TBB
- Java: java.util.concurrent package
- C#: .NET Parallel API
WE KNOW LITTLE ABOUT HOW DEVELOPERS USE THESE LIBRARIES

- No empirical study on a large-scale so far

- This is bad news for 4 different communities
Developers
Developers

- Invest time to learn API?
- Missing educational resources
When should I use AsParallel() in linq/plinq

I'm looking to make use of the advantages of parallel programming in linq by using plinq, but I'm not sure I completely understand the use entirely apart from the fact it's going to make use of all CPU cores more efficiently so for a large query it might be quicker. Can I just call AsParallel() on linq calls to make use of the plinq functionality and it will always be quicker? Or should I only use it when there is a lot of data to query or process?

How to use Parallel.For?

I want to use Parallel Programming in my project (WPF). Here is my for loop code:

```csharp
for (int i = 0; i < results.Count; i++)
{
    product p = new product();

    Common.SelectedOldColor = p.Background;
    p.VideoInfo = results[i];
    Common.Products.Add(p, false);
    p.Drop_event += new product.DragDropEvent(p_Drop_Event);
    main.Children.Add(p);
}
```

It works without any problem. I want to write it with Parallel.For and I wrote this.
<table>
<thead>
<tr>
<th>Developers</th>
<th>Researchers</th>
</tr>
</thead>
<tbody>
<tr>
<td>✦ Invest time to learn API?</td>
<td></td>
</tr>
<tr>
<td>✦ Missing educational resources</td>
<td></td>
</tr>
<tr>
<td>Developers</td>
<td>Researchers</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>✦ Invest time to learn API?</td>
<td>✦ Make wrong assumptions</td>
</tr>
<tr>
<td>✦ Missing educational resources</td>
<td></td>
</tr>
<tr>
<td>Developers</td>
<td>Researchers</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>- Invest time to learn API?</td>
<td>- Make wrong assumptions</td>
</tr>
<tr>
<td>- Missing educational resources</td>
<td></td>
</tr>
</tbody>
</table>

**Library Designers**

Stephen Toub
Developers

- Invest time to learn API?
- Missing educational resources

Researchers

- Make wrong assumptions

Library Designers

- In danger of designing error-prone, hard to use APIs
Why

Developers

- Invest time to learn API?
- Missing educational resources

Researchers

- Make wrong assumptions

Library Designers

- In danger of designing error-prone, hard to use APIs

Tool Vendors
Developers

- Invest time to learn API?
- Missing educational resources

Researchers

- Make wrong assumptions

Library Designers

- In danger of designing error-prone, hard to use APIs

Tool Vendors

- Don’t know what to automate
Are developers embracing multi-threading?
How quickly do developers start using the new TPL & PLINQ libraries?
Which parallel constructs do developers use most often?
How do developers protect accesses to shared variables?
Which parallel patterns do developers embrace?
Which advanced features do developers use?
Do developers make their parallel code unnecessarily complex?
Are there constructs that developers commonly mis-use?
How

• Why C#

• What is our data?

• How do we gather information from the data?
WHY C#?

- One of most advanced parallel APIs: 4 different libraries
  - .Net 4.5 introducing new parallel constructs in 2012
- Wide range of platforms: desktop, server, mobile, and web applications.
  - Not only windows platforms but also iOS, linux, mac
- C# parallel programming is a breakout trend
RISING TREND: C# PARALLEL

![Google Trends for parallel programming](image)

### Related terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Trend</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel programming .net</td>
<td>Breakout</td>
</tr>
<tr>
<td>parallel programming c#</td>
<td>Breakout</td>
</tr>
<tr>
<td>parallel programming openmp</td>
<td>Breakout</td>
</tr>
<tr>
<td>openmp</td>
<td>+450%</td>
</tr>
<tr>
<td>parallel programming pdf</td>
<td>+250%</td>
</tr>
<tr>
<td>mpi programming</td>
<td>+200%</td>
</tr>
<tr>
<td>parallel programming mpi</td>
<td>+170%</td>
</tr>
<tr>
<td>parallel computing</td>
<td>+160%</td>
</tr>
<tr>
<td>parallel computing programming</td>
<td>+140%</td>
</tr>
<tr>
<td>parallel java</td>
<td>+80%</td>
</tr>
</tbody>
</table>
C# PARALLEL API

Threading

• Thread

```csharp
Thread thread = new Thread(ProcessFiles);
thread.Start();
thread.Join();
```

• synchronization constructs:
  locks (lock, ReaderWriterLock)
  signals(events, barrier)
  non-blocking (volatile, interlocked)
  blocking (Thread.sleep, Thread.join)
C# PARALLEL API

Collections.Concurrent

• Several thread-safe, scalable collections

BlockingCollection
ConcurrentDictionary
ConcurrentQueue
ConcurrentBag ...
C# PARALLEL API

PLINQ

• Parallel implementation of LINQ, data querying

```csharp
assembly.GetTypes().AsParallel()
  .Where(t => t.IsSubclassOf(typeof(ControllerBase)))
  .Select(t => new ...)
  .ForAll(t => controllersCache.Add(t.Name, t.Type));
```
C# PARALLEL API

TPL (Task-based Library)

• Parallel Class

```csharp
Parallel.For(0, 10, counter =>
    compute(counter, database, table))
```

• Task Class

```csharp
var task1 = TaskFactory.StartNew(() => processFiles())
var task2 = TaskFactory.StartNew(() => processImages())
Task.WaitAll(task1, task2);
```
C# PARALLEL API

2002

.NET 1.0 is released

Threading

Old

2010

.NET 4.0 is released

Collections.Concurrent
PLINQ
TPL

New
CORPUS OF DATA

- Downloaded all C# projects as of January 31st, 2012
- Ignored toy applications (less than 1000 SLOC)
- Compilable and targeting .NET 4.0
## CORPUS OF DATA

<table>
<thead>
<tr>
<th>Applications compilable and targeting .NET 4.0</th>
<th>Small (1K-10K)</th>
<th>Medium (10K-100K)</th>
<th>Large (&gt;100K)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications compilable and targeting .NET 4.0</td>
<td>6020</td>
<td>1553</td>
<td>205</td>
<td>7778</td>
</tr>
<tr>
<td>Multi-threaded Applications</td>
<td>1761</td>
<td>916</td>
<td>178</td>
<td>2855</td>
</tr>
</tbody>
</table>
## CORPUS OF DATA

<table>
<thead>
<tr>
<th></th>
<th>Small (1K-10K)</th>
<th>Medium (10K-100K)</th>
<th>Large (&gt;100K)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-threaded Apps</td>
<td>1761</td>
<td>916</td>
<td>178</td>
<td>2855</td>
</tr>
</tbody>
</table>

Focused on the apps adopted new parallel libraries: TPL & PLINQ

655 applications = 17.6M SLOC by 1609 programmers
AUTOMATED STATIC ANALYSIS

based on Microsoft Roslyn Project

• syntactic (lexical) analysis: traversing AST nodes
  used Syntax API of Roslyn

• semantic analysis: get type and object binding information
  used Symbol and Binding APIs of Roslyn
Q1: ARE DEVELOPERS EMBRACING MULTI-THREADING?

Should we learn how to use parallel libraries, or should we avoid them because they are a passing fad?

We built multicores but do developers take advantage of parallelism?
Q1: ARE DEVELOPERS EMBRACING MULTI-THREADING?

<table>
<thead>
<tr>
<th></th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications compilable and targeting .NET 4.0</td>
<td>6020</td>
<td>1553</td>
<td>205</td>
<td>7778</td>
</tr>
<tr>
<td>Multi-threaded Applications</td>
<td>1761</td>
<td>916</td>
<td>178</td>
<td>2855</td>
</tr>
</tbody>
</table>

87% 37%
Q1: HOW MANY APPS USE PARALLELISM VS CONCURRENCY?
PARALLELISM VS CONCURRENCY

throughput

responsiveness

[From: Tom Ball’s slides]
Q1: HOW MANY APPS USE PARALLELISM VS CONCURRENCY?

Parallelism

Parallel.For, PLINQ constructs
Main thread waits for the child threads to finish

Concurrency

FromAsync, TaskCompletionSource, UI event dispatching thread
Main thread does not wait for child threads

- Parallelism: 39%
- Concurrency: 74%
- Both: 12%
Q1: ARE DEVELOPERS EMBRACING MULTI-THREADING?

Many applications have embraced multi-threading, however many of them use it for concurrency rather than parallelism.

- Many developers will not be able to completely avoid multi-threaded programming
- Intel should be happy. Developers are taking advantage of multicores
Q2: HOW QUICKLY DO DEVELOPERS START USING THE NEW TPL & PLINQ LIBRARIES?

How long does it take for developers to start using new parallel libraries?
Q2: HOW QUICKLY DO DEVELOPERS START USING THE NEW TPL & PLINQ LIBRARIES?

- Purpose is to find out the tipping point for new parallel constructs
- Tipping point: magic moment when a trend spreads like wildfire
Q2: HOW QUICKLY DO DEVELOPERS START USING THE NEW TPL & PLINQ LIBRARIES?

- Selected the subset of applications that exist in the repository as of April 2010: 54 out of 655
- Analyzed monthly snapshots: 31.9MLOC, comprising 694 different versions
- Collected the usage details of TPL & PLINQ constructs from these versions
# TIPPING POINT

- **Small**: after 2-3 months
- **Medium**: after 4-5 months
- **Large**: after 8-9 months
Q2: HOW QUICKLY DO DEVELOPERS START USING THE NEW TPL & PLINQ LIBRARIES?

Applications of different size adopt the new parallel libraries differently

• Small applications are early adopters. They have higher density of parallel constructs

• Developers are better off looking for parallelism examples in small applications.
Q3: WHICH PARALLEL CONSTRUCTS DO DEVELOPERS USE MOST OFTEN?

Which constructs do developers prefer to use and which ones do not they prefer?

How can I become proficient quickly? Where can I find sample code?
Q3: WHICH PARALLEL CONSTRUCTS DO DEVELOPERS USE MOST OFTEN?

- Usage details for 4 Libraries, 138 classes, 1651 methods

- Detected each method call, class constructor call and got the type of the caller and callee by using Symbol and Binding APIs

- Type information: 100% precise
<table>
<thead>
<tr>
<th>Constructor</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>StartNew(System.Action)</code></td>
<td>1007</td>
</tr>
<tr>
<td><code>StartNew&lt;TResult&gt;(System.Func&lt;TResult&gt;)</code></td>
<td>199</td>
</tr>
<tr>
<td><code>StartNew(System.Action, CancellationToken)</code></td>
<td>60</td>
</tr>
<tr>
<td><code>StartNew(System.Action, TaskCreationOptions)</code></td>
<td>57</td>
</tr>
<tr>
<td><code>StartNew(System.Action&lt;object&gt;, object)</code></td>
<td>55</td>
</tr>
<tr>
<td><code>StartNew(System.Action&lt;object&gt;, object, TaskCreationOptions, TaskScheduler)</code></td>
<td>48</td>
</tr>
<tr>
<td><code>StartNew(System.Action&lt;object&gt;, object, TaskCreationOptions)</code></td>
<td>19</td>
</tr>
<tr>
<td><code>StartNew&lt;TResult&gt;(System.Func&lt;TResult&gt;, CancellationToken)</code></td>
<td>14</td>
</tr>
<tr>
<td><code>StartNew&lt;TResult&gt;(System.Func&lt;object, TResult&gt;, object)</code></td>
<td>12</td>
</tr>
<tr>
<td><code>StartNew(System.Action&lt;object&gt;, object, CancellationToken)</code></td>
<td>7</td>
</tr>
<tr>
<td><code>StartNew&lt;TResult&gt;(System.Func&lt;TResult&gt;, TaskCreationOptions)</code></td>
<td>7</td>
</tr>
<tr>
<td><code>StartNew&lt;TResult&gt;(System.Func&lt;TResult&gt;, CancellationToken, TaskCreationOptions, TaskScheduler)</code></td>
<td>7</td>
</tr>
<tr>
<td><code>StartNew&lt;TResult&gt;(System.Func&lt;object, TResult&gt;, object, CancellationToken)</code></td>
<td>6</td>
</tr>
<tr>
<td><code>StartNew(System.Action&lt;object&gt;, object, CancellationToken, TaskCreationOptions, TaskScheduler)</code></td>
<td>3</td>
</tr>
</tbody>
</table>
Q3: WHICH PARALLEL CONSTRUCTS DO DEVELOPERS USE MOST OFTEN?

- 67% (1114) of all method signatures are never used
- 10% of the API methods account for 90% of the total usage.
Q3: WHICH PARALLEL CONSTRUCTS DO DEVELOPERS USE MOST OFTEN?

Parallel library usage follows a power-law distribution: 10% of the API methods account for 90% of the total usage.

• Good news for developers who are just learning parallel libraries: they can focus on learning a relatively small subset of the library APIs and still be able to master a large number of parallelism scenarios.

• Library designers can think about never used methods. How to reduce API size?
Q4: HOW DO DEVELOPERS PROTECT ACCESSES TO SHARED VARIABLES?

Which synchronization types should we model in our algorithms and tools?
Q4: HOW DO DEVELOPERS PROTECT ACCESSES TO SHARED VARIABLES?

• Around 25 different synchronization constructs in 5 different categories:

  Locking, non-blocking, signaling, implicit, blocking
<table>
<thead>
<tr>
<th>Type</th>
<th>% in Types</th>
<th>Name</th>
<th>#</th>
<th>% in Type</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locking</td>
<td>39</td>
<td>lock (language feature)</td>
<td>6643</td>
<td>89</td>
<td>361</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ReaderWriterLockSlim</td>
<td>258</td>
<td>3</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monitor - Enter/Exit</td>
<td>245</td>
<td>3</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mutex</td>
<td>94</td>
<td>1</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Semaphore</td>
<td>75</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ReaderWriterLock</td>
<td>65</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SpinLock</td>
<td>31</td>
<td>0.4</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SemaphoreSlim</td>
<td>20</td>
<td>0.3</td>
<td>10</td>
</tr>
<tr>
<td>Non-Blocking</td>
<td>26</td>
<td>Volatile Accesses</td>
<td>3212</td>
<td>65</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interlocked Methods</td>
<td>1696</td>
<td>34</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thread.MemoryBarrier</td>
<td>50</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>Implicit</td>
<td>21</td>
<td>CC Operations</td>
<td>4021</td>
<td>100</td>
<td>283</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ManualResetEvent</td>
<td>671</td>
<td>38</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AutoResetEvent</td>
<td>647</td>
<td>37</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monitor - Wait/Pulse</td>
<td>168</td>
<td>10</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ManualResetEventSlim</td>
<td>167</td>
<td>10</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CountdownEvent</td>
<td>58</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Barrier</td>
<td>33</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Signaling</td>
<td>9</td>
<td>Thread.Join</td>
<td>382</td>
<td>38</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thread.Sleep</td>
<td>350</td>
<td>35</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Task.Wait</td>
<td>273</td>
<td>27</td>
<td>110</td>
</tr>
<tr>
<td>Blocking</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Q4: HOW DO DEVELOPERS PROTECT ACCESSES TO SHARED VARIABLES?

While locks are still very popular, developers use a wide variety of other synchronization constructs.

• Data-race detectors should also model these other synchronization constructs, not only locks!
Q5: WHICH PARALLEL PATTERNS DO DEVELOPERS EMBRACE?

Where can we find real world examples of parallel patterns?

Which parallel patterns should we support in our tools?
Q5: WHICH PARALLEL PATTERNS DO DEVELOPERS EMBRACE?

<table>
<thead>
<tr>
<th>Data Parallelism</th>
<th>Regular Parallel Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Aggregation (parallel dependent loops)</td>
</tr>
<tr>
<td>Task Parallelism</td>
<td>Regular fork join tasks</td>
</tr>
<tr>
<td></td>
<td>futures, task dependency</td>
</tr>
<tr>
<td></td>
<td>pipeline</td>
</tr>
<tr>
<td></td>
<td>dynamic task parallelism</td>
</tr>
</tbody>
</table>
**Q5: WHICH PARALLEL PATTERNS DO DEVELOPERS EMBRACE?**

<table>
<thead>
<tr>
<th>Data Parallelism (68%)</th>
<th>Regular Parallel Loops</th>
<th>954</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Aggregation (parallel dependent loops)</td>
<td>82</td>
</tr>
<tr>
<td>Task Parallelism (32%)</td>
<td>Regular fork join tasks</td>
<td>268</td>
</tr>
<tr>
<td></td>
<td>futures, task dependency</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>pipeline</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>dynamic task parallelism</td>
<td>18</td>
</tr>
</tbody>
</table>
Q5: WHICH PARALLEL PATTERNS DO DEVELOPERS EMBRACE?

Regular data parallelism is the most used parallel pattern in practice

• Our study also educates developers by showing real-world examples of parallel patterns
Q6: WHICH ADVANCED FEATURES DO DEVELOPERS USE?

Are developers using our fancy features?
Q6: WHICH ADVANCED FEATURES DO DEVELOPERS USE?

- More programmatic control than is possible with a thread or work item.

Tasks and the framework built around them provide a rich set of APIs that support waiting, cancellation, continuations, robust exception handling, detailed status, custom scheduling, and more.

Q6: WHICH ADVANCED FEATURES DO DEVELOPERS USE?

• All Parallel class methods (Invoke, For, ForEach) can take ParallelOptions as an argument.

• With ParallelOptions: (1) insert a cancellation token (2) limit the maximum concurrency (3) specify a custom task scheduler.

```csharp
ParallelOptions opts=
    new ParallelOptions {
        CancellationToken = (new CancellationTokenSource()).token;
        MaxDegreeOfParallelism = Environment.ProcessorCount;
        TaskScheduler = new QueuedTaskScheduler();
    }

Parallel.For(0, 100, options, i => { ... });
```
Q6: WHICH ADVANCED FEATURES DO DEVELOPERS USE?

- Of 852 method calls of Parallel class, only 3% use ParallelOptions.

- When ParallelOptions is used:
  - Custom TaskScheduler is used only once
  - 60% of the times developers overwrite MaxDegreeOfParallelism to be equal with the number of processors

```csharp
ParallelOptions opts = new ParallelOptions {
    MaxDegreeOfParallelism = Environment.ProcessorCount;
};
```
Q6: WHICH ADVANCED FEATURES DO DEVELOPERS USE?

The advanced features and optional arguments are rarely used in practice.

- To Library Designers: developers are not happy about the default degree of parallelism.
Q7: DO DEVELOPERS MAKE THEIR PARALLEL CODE UNNECESSARILY COMPLEX?

Parallel code is much more complex than sequential.

- Why is my parallel code so complex?
- Can we simplify the parallel code?
Q7: DO DEVELOPERS MAKE THEIR PARALLEL CODE UNNECESSARILY COMPLEX?

```csharp
var runDaemons = new Task(RunDaemonJobs, ..);
...
var runScheduledJobs = new Task(RunScheduledJobs, ..);
var tasks = new[] {runDaemons, ..., runScheduledJobs};
Array.ForEach(tasks, x => x.Start());
Task.WaitAll(tasks);
Parallel.Invoke(RunDaemonJobs, ..., RunScheduledJobs);
```

63 out of 268 regular fork/join task parallelism, the programmers could have used Parallel.Invoke
Q7: DO DEVELOPERS MAKE THEIR PARALLEL CODE UNNECESSARILY COMPLEX?

for (int i = 1; i <= threadCount; i++) {
    var copy = i;
    var taskHandle = Task.Factory.StartNew(
        () => DoInefficientInsert(...));
    tasks.Add(taskHandle);
}
Task.WaitAll(tasks);

Parallel.For(1, threadCount, (i) => DoInefficientInsert(...))

55 out of 189 cases could have used Parallel.For or Parallel.ForEach.
Q7: DO DEVELOPERS MAKE THEIR PARALLEL CODE UNNECESSARILY COMPLEX?

Despite the fact that parallel programs are already complex, developers make them even more complex than they need to be.

- Developers use more powerful constructs instead of the equivalent but simpler constructs.
- Refactoring that allow programmers to improve the readability of their parallel code have never been automated before, but are invaluable.
Q8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

- Why does not my code get any speedup with parallel constructs?
- How can we prevent the misuse of parallel constructs?
Q8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

All,

I am using the Parallels.ForEach as follows

```csharp
private void fillEventDifferencesParallels(IProducerConsumerCollection<IEvent> eve
{
    Parallel.ForEach<IEvent>(events, evt =>
    {
        IEvent originalEventInfo = originalEvents[evt.EventID];
        evt.FillDifferences(originalEventInfo);
    });
}
```

Ok, so the problem I'm having is I have a list of 28 of these (a test sample, this should be able to scale to 200+) and the FillDifferences method is quite time consuming (about 4s per call). So the Average time for this to run in a normal ForEach has been around 100-130s. When I run the same thing in Parallel, it takes the same amount of time and Spikes my CPU (Intel i5, 2 Core, 2 Threads per Core) causing the app to become sluggish while this query is running (this is running on a thread that was spawned by the GUI thread).
Q8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

Parallel.Invoke(() => i.ImportGPX(..));

- ImportGPX will execute in parallel with the main thread, when in fact it doesn't.
- 11% of all usages of Parallel.Invoke take only one action parameter in different applications.
Q8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

AsParallel() from PLINQ

assembly.GetTypes().AsParallel().
   Where(t => t.IsSubclassOf(...)).
   ForAll(t => controllersCache.Add(...));
Q8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

```csharp
foreach (var module in Modules.AsParallel())
    module.Refresh();
```

- The `foreach` proceeds sequentially.
- 27 cases in 19 applications (representing 12% of all `AsParallel` usages)
Q8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

Misuse of parallel constructs can lead to code with parallel syntax but sequential execution.

- Library designers may consider removing Parallel.Invoke version that takes only one action parameter.
SO WHAT?
http://LearnParallelism.NET
CONCLUSION

• First large-scale empirical study on the usage of parallel libraries

• We answered 8 research questions related to adoption, frequently (mis)used constructs and patterns.

• Implications for
  • Developers: [http://LearnParallelism.NET](http://LearnParallelism.NET)
  • Library Designers: awareness of API problems
  • Researchers & Tool Vendors: know what to automate