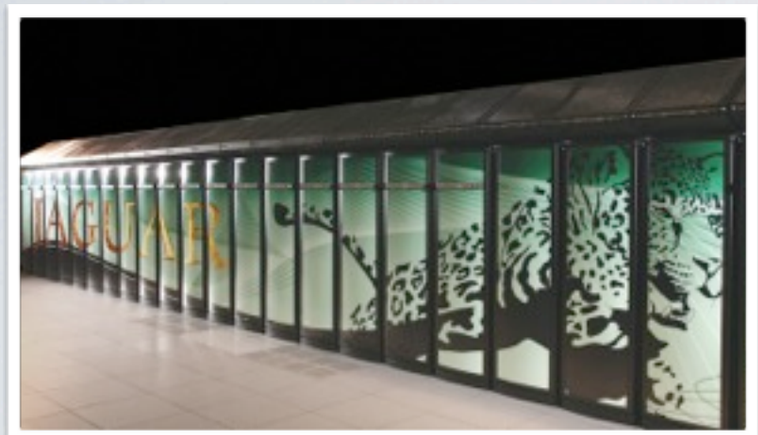
HOW DO DEVELOPERS USE PARALLEL LIBRARIES?

**To Appear @ FSE'12*

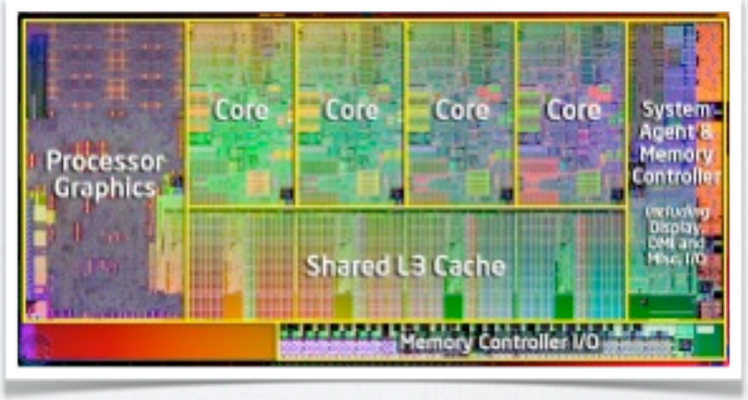
Semih Okur & Danny Dig

University of Illinois at Urbana-Champaign

PARALLEL COMPUTING IS EVERYWHERE NOW



2004



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

6

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

linq | parallel-processing | plinq

share | improve this question

asked **Nov 12 '10 at 10:59**
Froggy
31 •2

feedback

tagged

linq × 23966

parallel-processing × 3263

plinq × 154

asked **1 year ago**
viewed **1351 times**
active **1 year ago**

Linked

Nested Parallel.ForEach Loops on the same list?

How to use Parallel.For?

5

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

```
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.Visibility = System.Windows.Visibility.Hidden;
    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

tagged

c# × 374352

parallel-for × 27

asked **2 months ago**
viewed **119 times**
active **2 months ago**

Related

Could this WPF code benefit from Parallel.For and how?

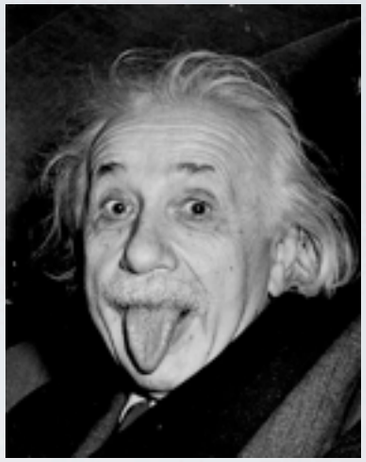
Do the parallel-for in .net 4.0 takes privilege of GPU computing automatically?

Why

Developers

- Invest time to learn API?
- Missing educational resources

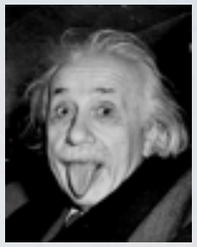
Researchers



Developers

- Invest time to learn API?
- Missing educational resources

Researchers

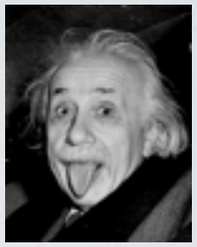


- Make wrong assumptions

Developers

- Invest time to learn API?
- Missing educational resources

Researchers



- Make wrong assumptions

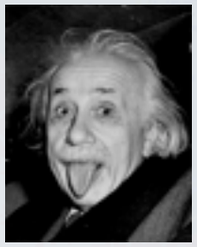
Library Designers

Stephen Toub

Developers

- Invest time to learn API?
- Missing educational resources

Researchers



- Make wrong assumptions

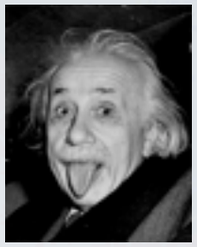
Library Designers

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

Developers

- Invest time to learn API?
- Missing educational resources

Researchers



- Make wrong assumptions

Library Designers

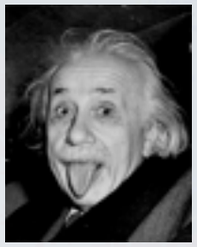
- In danger of designing error-prone, hard to use APIs

Tool Vendors

Developers

- Invest time to learn API?
- Missing educational resources

Researchers



- Make wrong assumptions

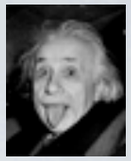
Library Designers

- 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

Web Search Interest: **parallel programming**. Worldwide, 2004 - present.

Related terms ?

Top | **Rising**

parallel programming .net	Breakout
parallel programming c#	Breakout
parallel programming openmp	Breakout
openmp	+450%
parallel programming pdf	+250%
mpi programming	+200%
parallel programming mpi	+170%
parallel computing	+160%
parallel computing programming	+140%
parallel java	+80%

How

C# PARALLEL API

Threading

- Thread

```
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`)

How

C# PARALLEL API

Collections.Concurrent

- Several thread-safe, scalable collections

```
BlockingCollection
ConcurrentDictionary
ConcurrentQueue
ConcurrentBag …
```

C# PARALLEL API

PLINQ

- Parallel implementation of LINQ, data querying

```
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

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

- Task Class

```
var task1 = TaskFactory.StartNew(() => processFiles())
var task2 = TaskFactory.StartNew(() => processImages())

Task.WaitAll(task1, task2);
```

C# PARALLEL API

2002

2010

.NET 1.0 is released

.NET 4.0 is released

Threading

Old

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

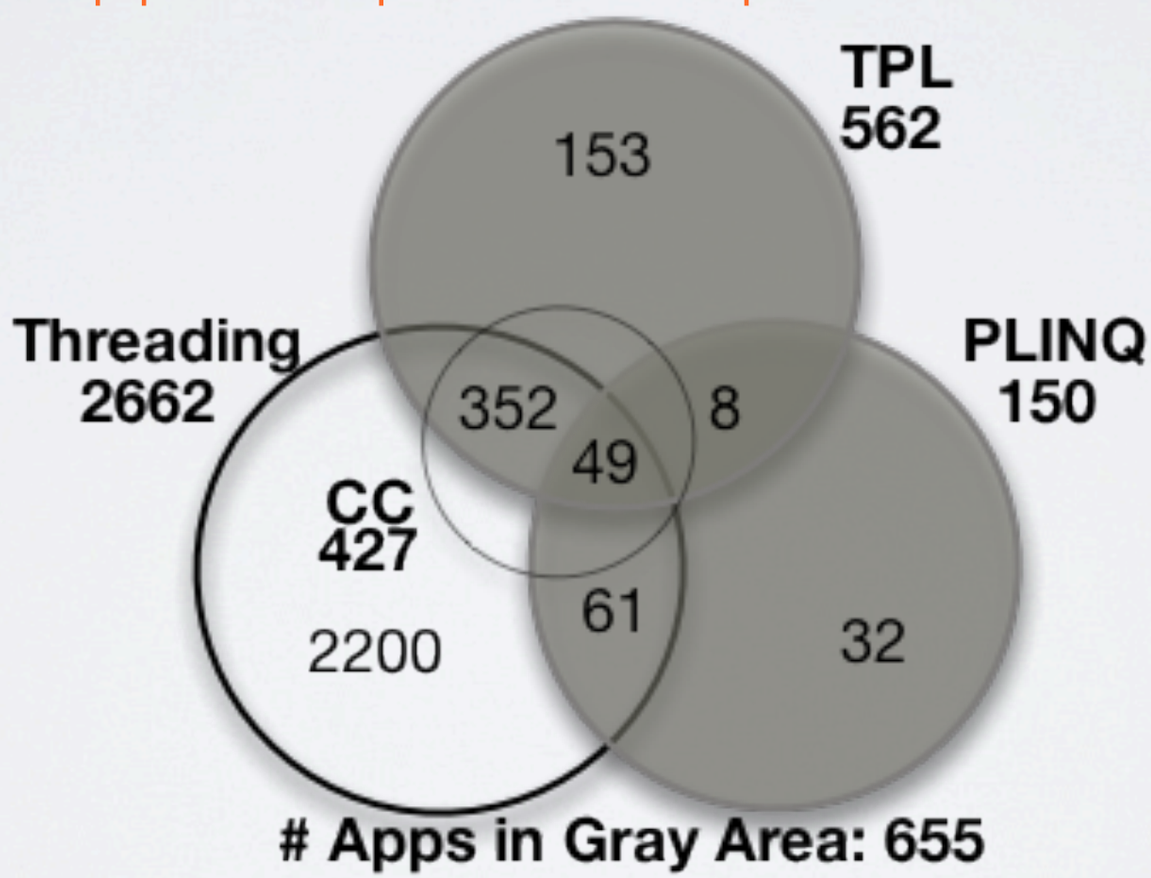
CORPUS OF DATA

	Small (1K-10K)	Medium (10K-100K)	Large (>100K)	Total
Applications compilable and targeting .NET 4.0	6020	1553	205	7778
Multi-threaded Applications	1761	916	178	2855

CORPUS OF DATA

	Small (1K-10K)	Medium (10K-100K)	Large (>100K)	Total
Multi-threaded Applications	1761	916	178	2855

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



655 applications = 17.6M SLOC by1609 programmers

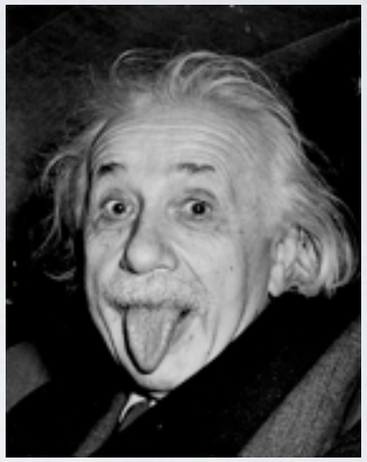
How

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?

	Small	Medium	Large	Total
Applications compilable and targeting .NET 4.0	6020	1553	205	7778
Multi-threaded Applications	1761	916	178	2855

87% 37%

What

Q1: HOW MANY APPS USE PARALLELISM VS CONCURRENCY?

stackoverflow Questions Tags Users Badges Unanswered Ask Questior

What is the difference between concurrency, parallelism and asynchronous methods?

8

2

Difference between concurrent programming and parallel programming

27

8

Concurrency vs Parallelism - What is the difference?

67

26

Concurrency vs Parallelism - What is the difference? Any examples

language-agnostic | concurrency | parallel-processing

share | improve this question

asked **Jun 26 '09 at 17:18**

StackUnderflow

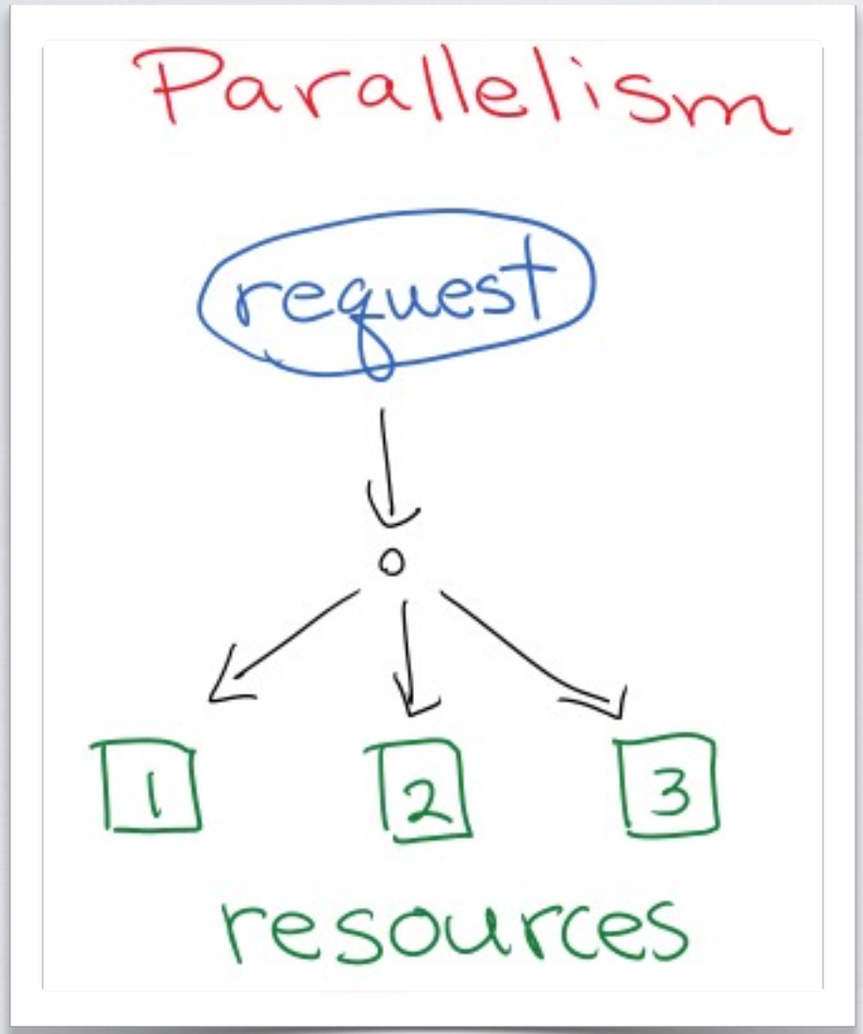
2,648 1 19 48

70% accept rate

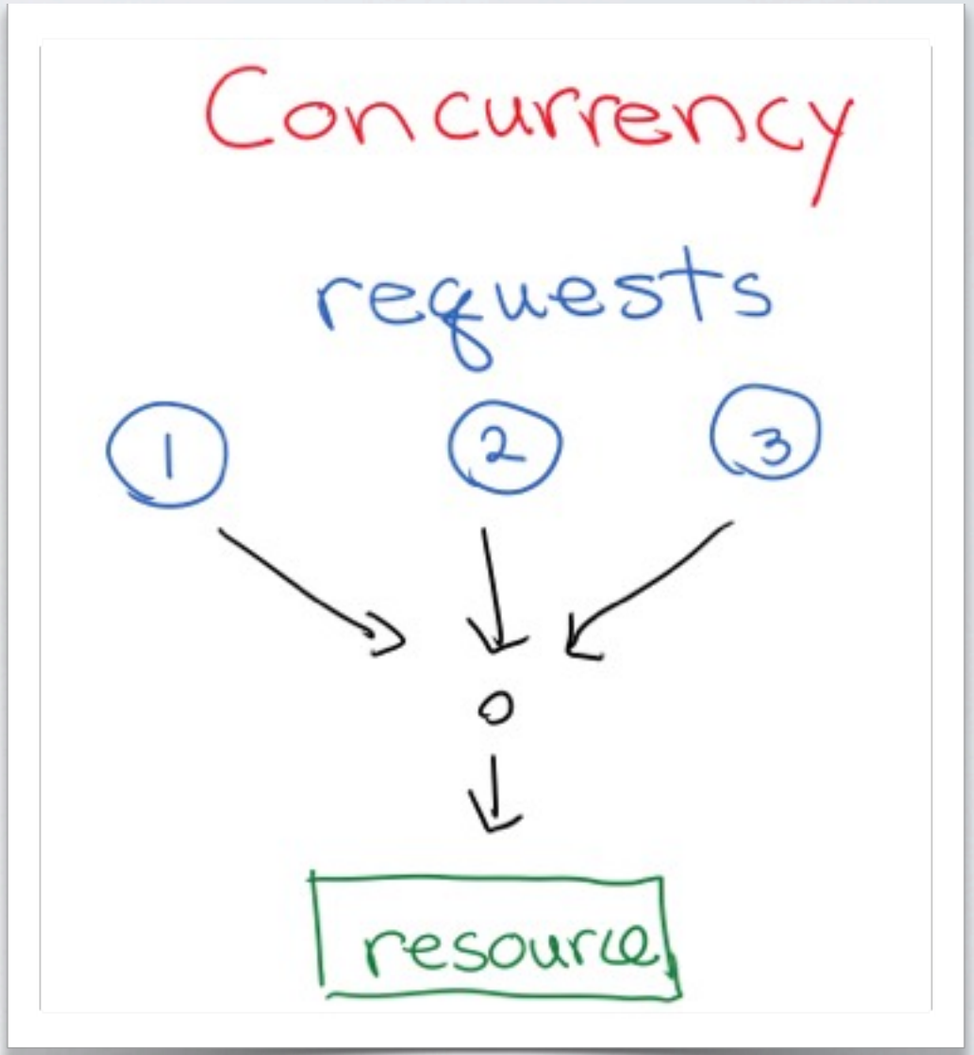
What

PARALLELISM VS CONCURRENCY

throughput



responsiveness



[From: Tom Ball's slides]

What

Q1: HOW MANY APPS USE PARALLELISM VS CONCURRENCY?

Parallelism

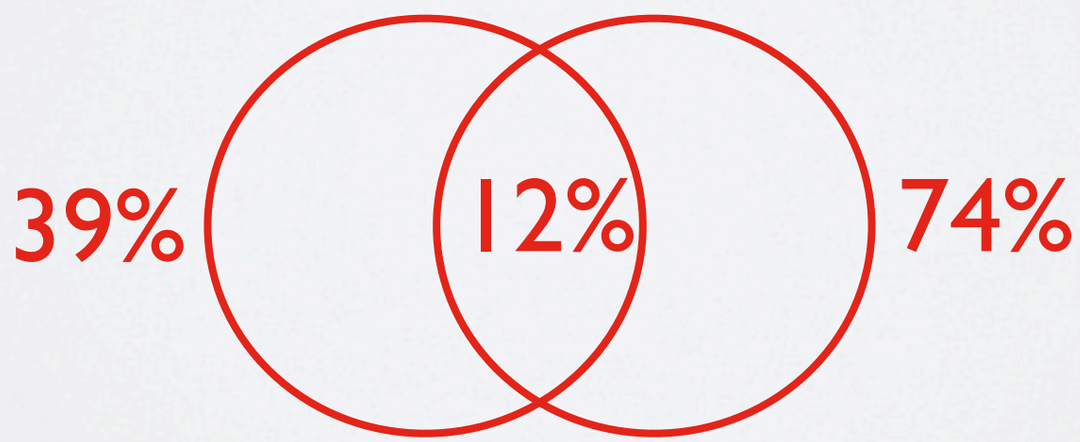
Parallel.For, PLINQ constructs

Main thread waits for the child threads to finish

Concurrency

FromAsync, TaskCompletion-Source, UI event dispatching thread

Main thread does not wait for child threads



What

Q1: ARE DEVELOPERS EMBRACING MULTI-THREADING?

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

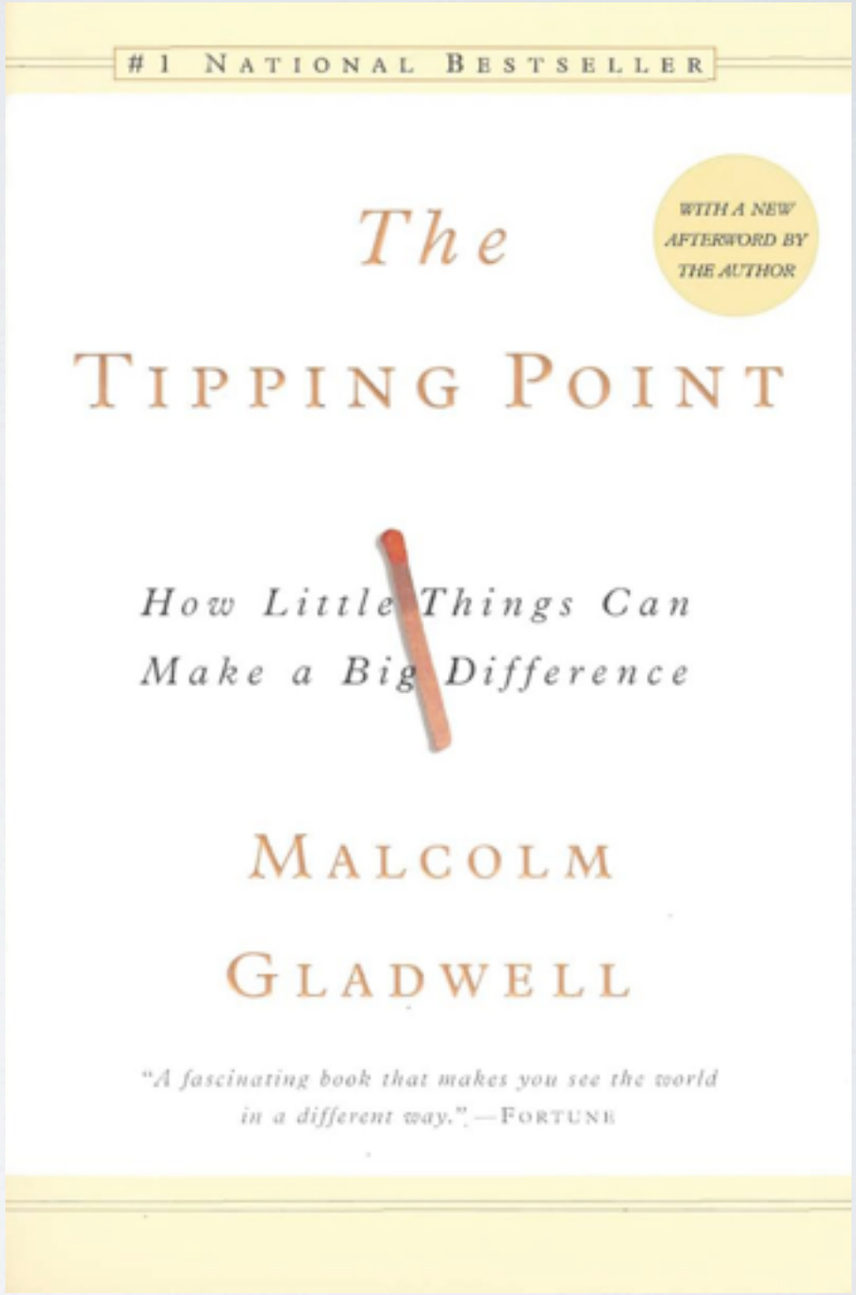
- Many developers will not be able to completely avoid multi-threaded programming
- Intel should be happy. Developers are taking advantage of multicores

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

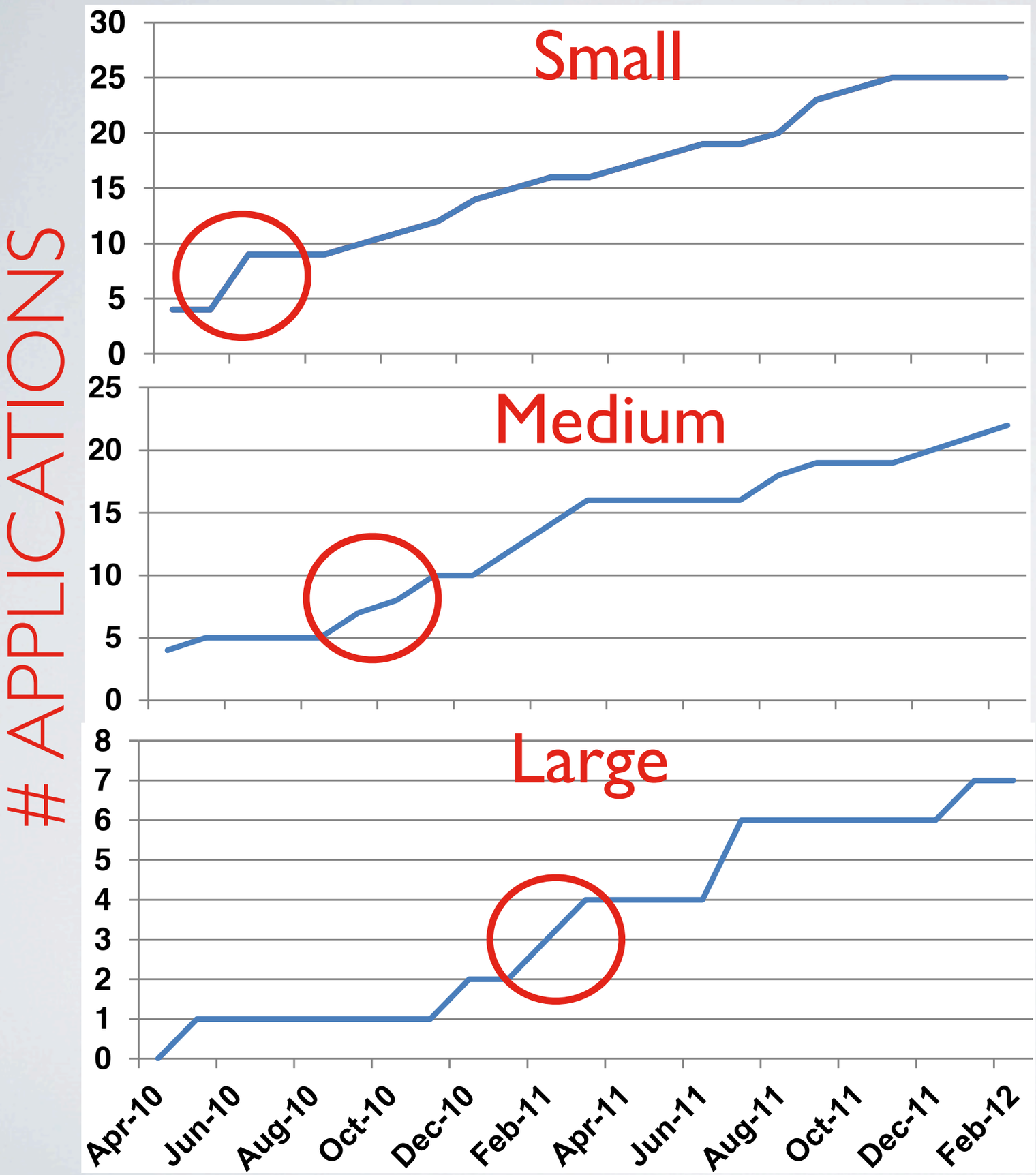


What

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



after 2-3 months

after 4-5 months

after 8-9 months

What

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?

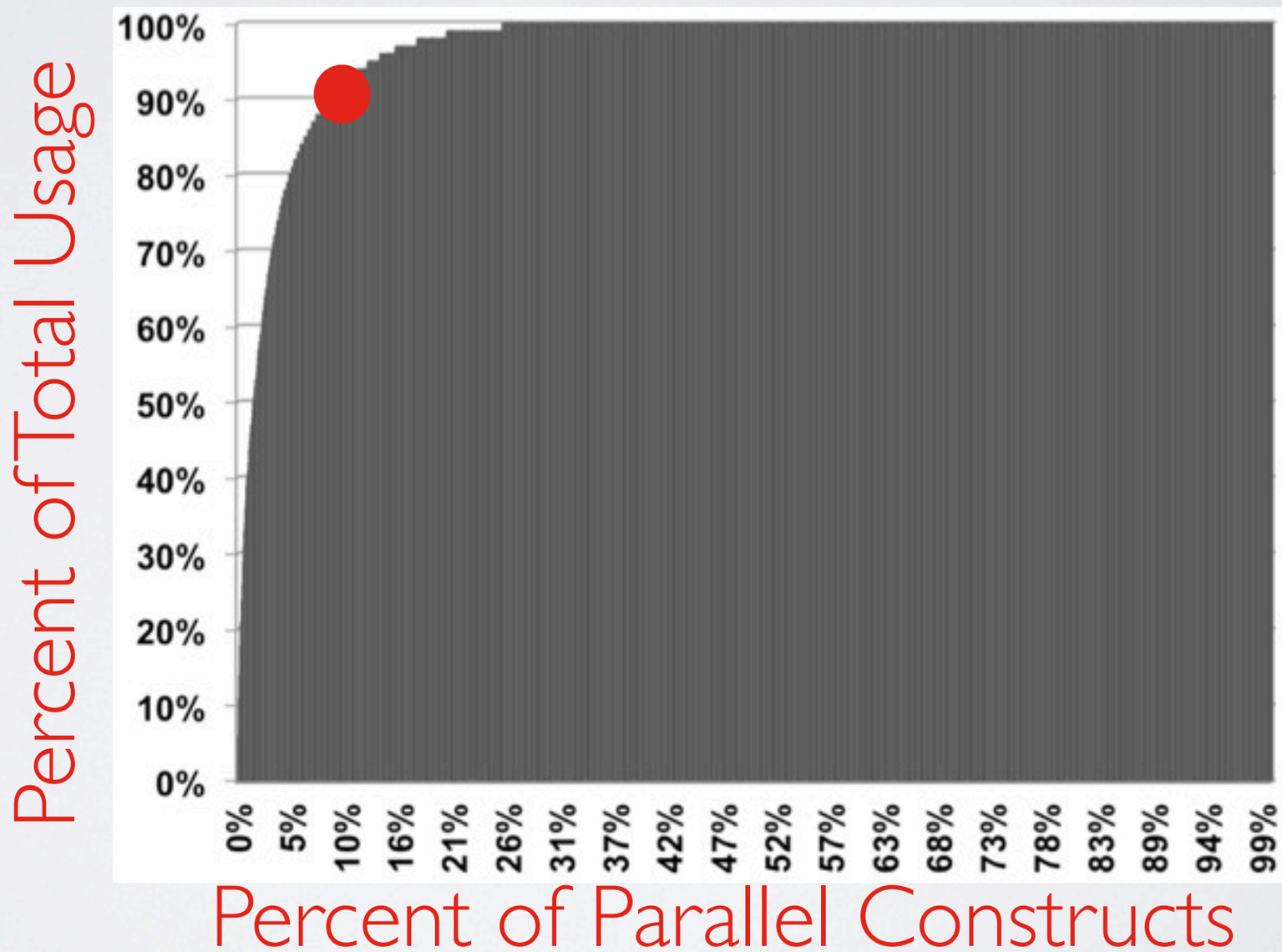
- Usage details for 4 Libraries, 138 classes, 1651 methods
- Detected each method call, class constructor call and got the type of the caller and callee by using Symbol and Binding APIs
- Type information: 100% precise

StartNew(System.Action)	1007
StartNew<TResult>(System.Func<TResult>)	199
StartNew(System.Action, CancellationToken)	60
StartNew(System.Action, TaskCreationOptions)	57
StartNew(System.Action<object>, object)	55
StartNew(System.Action, CancellationToken, TaskCreationOptions, TaskScheduler)	48
StartNew(System.Action<object>, object, TaskCreationOptions)	19
StartNew<TResult>(System.Func<TResult>, CancellationToken)	14
StartNew<TResult>(System.Func<object, TResult>, object)	12
StartNew(System.Action<object>, object, CancellationToken)	7
StartNew<TResult>(System.Func<TResult>, TaskCreationOptions)	7
StartNew<TResult>(System.Func<TResult>, CancellationToken, TaskCreationOptions, TaskScheduler)	7
StartNew<TResult>(System.Func<object, TResult>, object, CancellationToken)	6
StartNew(System.Action<object>, object, CancellationToken, TaskCreationOptions, TaskScheduler)	3

What

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.



What

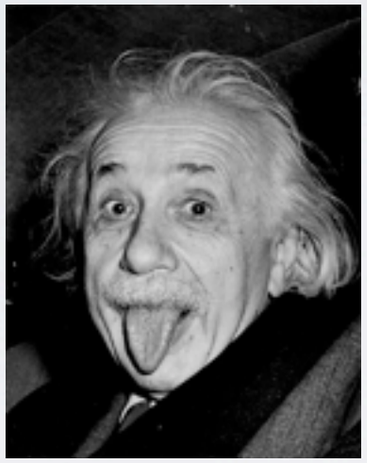
Q3: WHICH PARALLEL CONSTRUCTS DO DEVELOPERS USE MOST OFTEN?

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?

What

Q4: HOW DO DEVELOPERS PROTECT ACCESSES TO SHARED VARIABLES?

- Around 25 different synchronization constructs in 5 different categories:

  Locking, non-blocking, signaling, implicit, blocking

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

61%

What

Q4: HOW DO DEVELOPERS PROTECT ACCESSES TO SHARED VARIABLES?

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

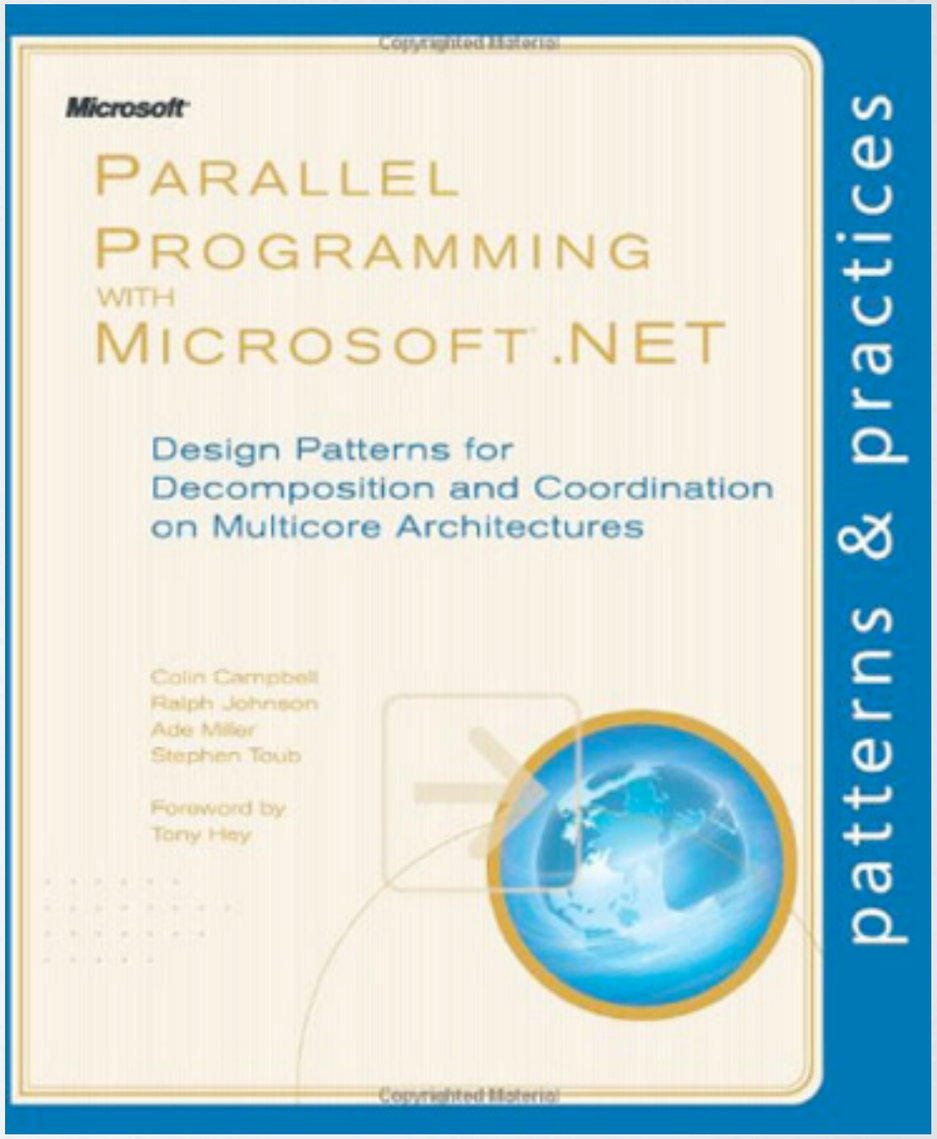
- Data-race detectors should also model these other synchronization constructs, not only locks!

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?

Data Parallelism	Regular Parallel Loops
	Aggregation (parallel dependent loops)
Task Parallelism	Regular fork join tasks
	futures, task dependency
	pipeline
	dynamic task parallelism



What

Q5: WHICH PARALLEL PATTERNS DO DEVELOPERS EMBRACE?

Data Parallelism (68%)	Regular Parallel Loops	954
	Aggregation (parallel dependent loops)	82
Task Parallelism (32%)	Regular fork join tasks	268
	futures, task dependency	155
	pipeline	41
	dynamic task parallelism	18

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.

[From http://msdn.microsoft.com/en-us/library/dd537609.aspx]

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.

```
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

```
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?

```
var runDaemons = new Task(RunDaemonJobs, ..);
...
var runScheduledJobs = new Task(RunScheduledJonbs, ..);
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?

What

Q8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

stackoverflow Questions Tags Users Badges Unanswered

Parallels.ForEach Taking same Time as Foreach

10

2

All,

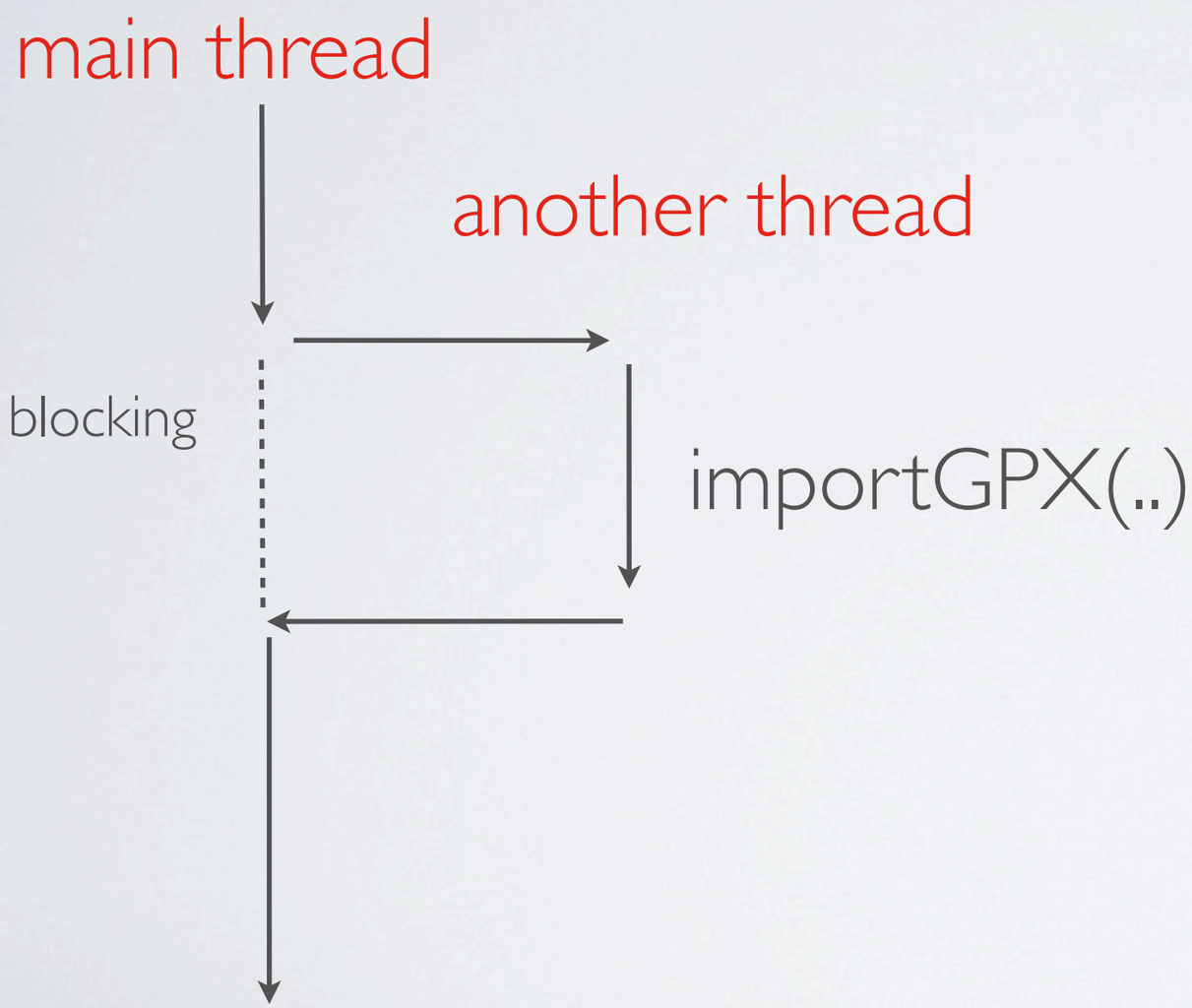
I am using the Parallels.ForEach as follows

```
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?

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

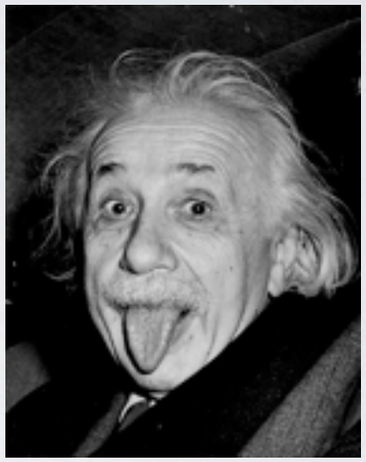
- The foreach proceeds sequentially.
- 27 cases in 19 applications (representing 12% of all AsParallel usages)

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
  - Library Designers: awareness of API problems
  - Researchers & Tool Vendors: know what to automate