HOW DO DEVELOPERS USE PARALLEL LIBRARIES?

Semih Okur & Danny Dig

University of Illinois at Urbana-Champaign

PARALLEL HARDWARE IS EVERYWHERE NOW

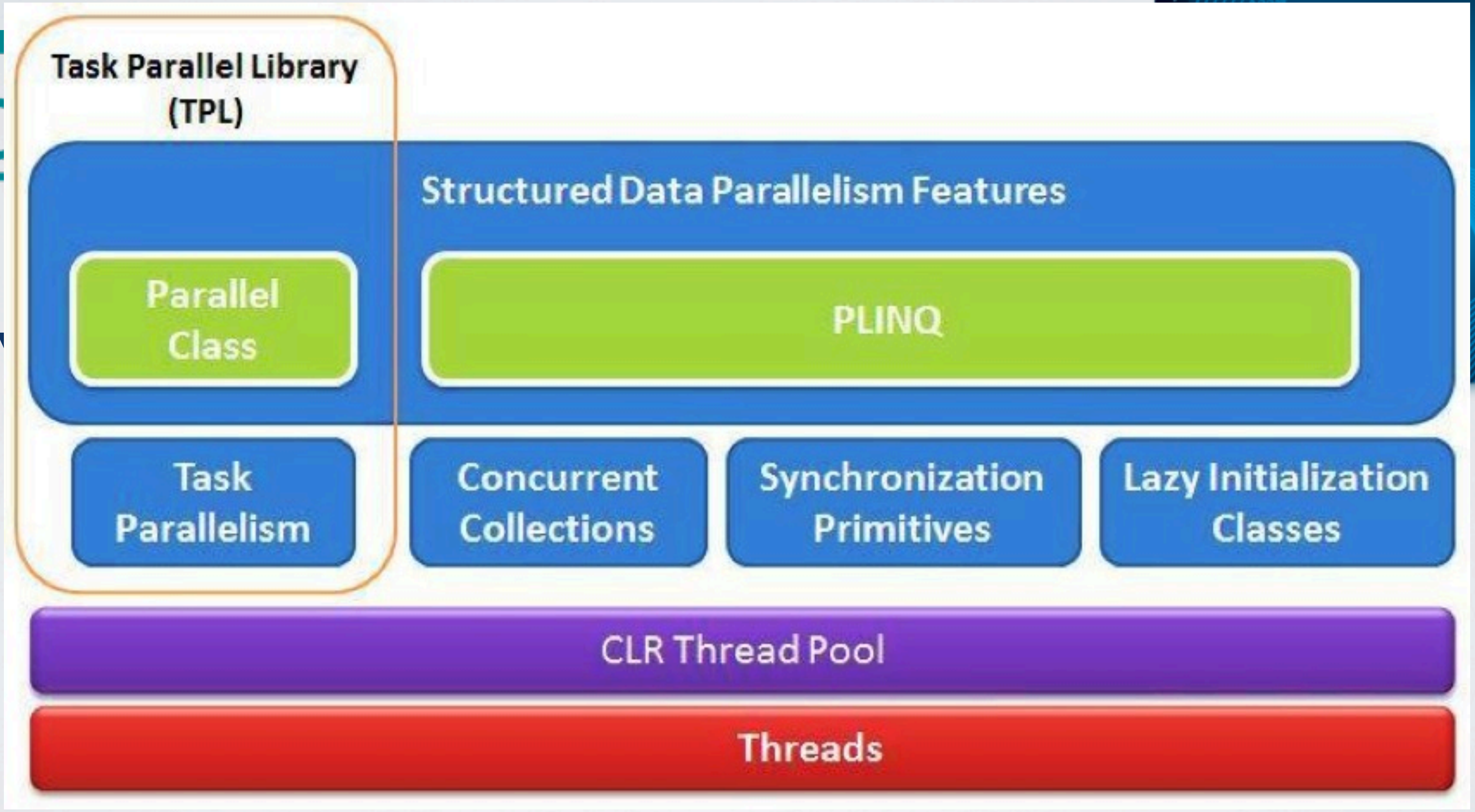
PARALLEL PROGRAMMING IS HARD

- Requires balancing two conflicting forces
  - (1) Thread-safe
  - (2) Scalable
- Makes the code more complex

[From "The Practice of Parallel Programming" book cover]

PARALLEL LIBRARIES ARE WIDESPREAD

ja


Task Parallel Library (TPL)
Structured Data Parallelism Features
Parallel Class
PLINQ
Task Parallelism
Concurrent Collections
Synchronization Primitives
Lazy Initialization Classes
CLR Thread Pool
Threads

WE KNOW NOTHING ABOUT HOW DEVELOPERS USE THESE LIBRARIES

- No empirical study on a large-scale so far

4 different communities strongly need such a study

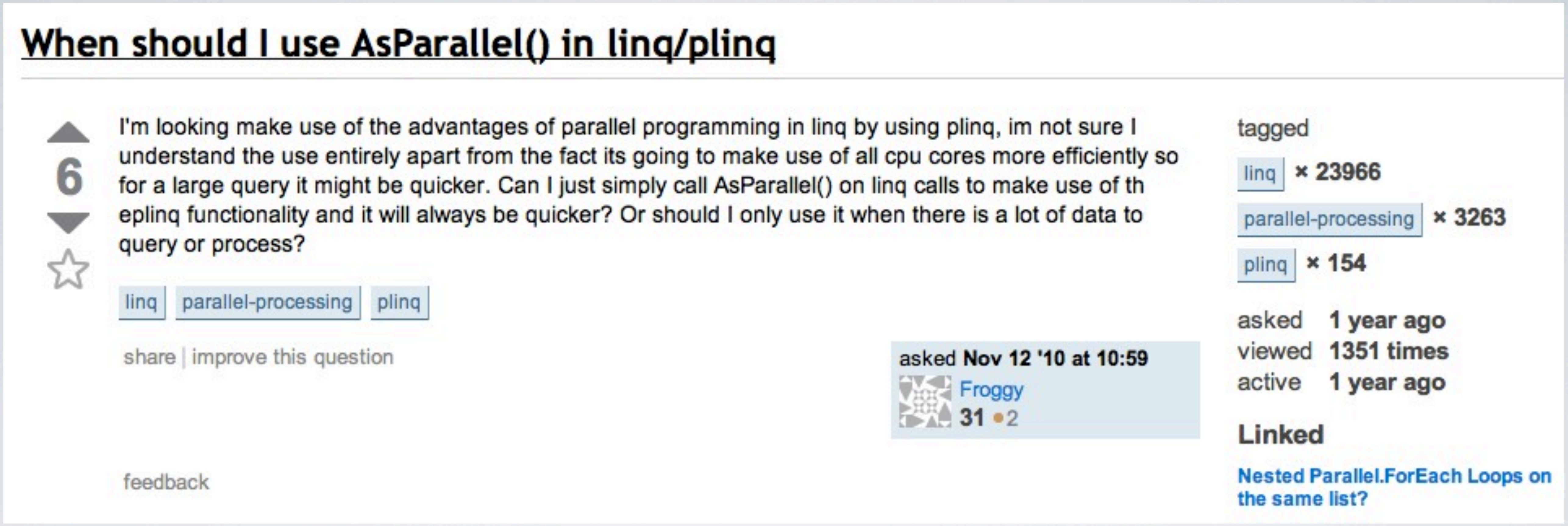
Without such a study

Developers

Without such a study

Developers

- Cannot decide whether to invest time to learn API
- Miss 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

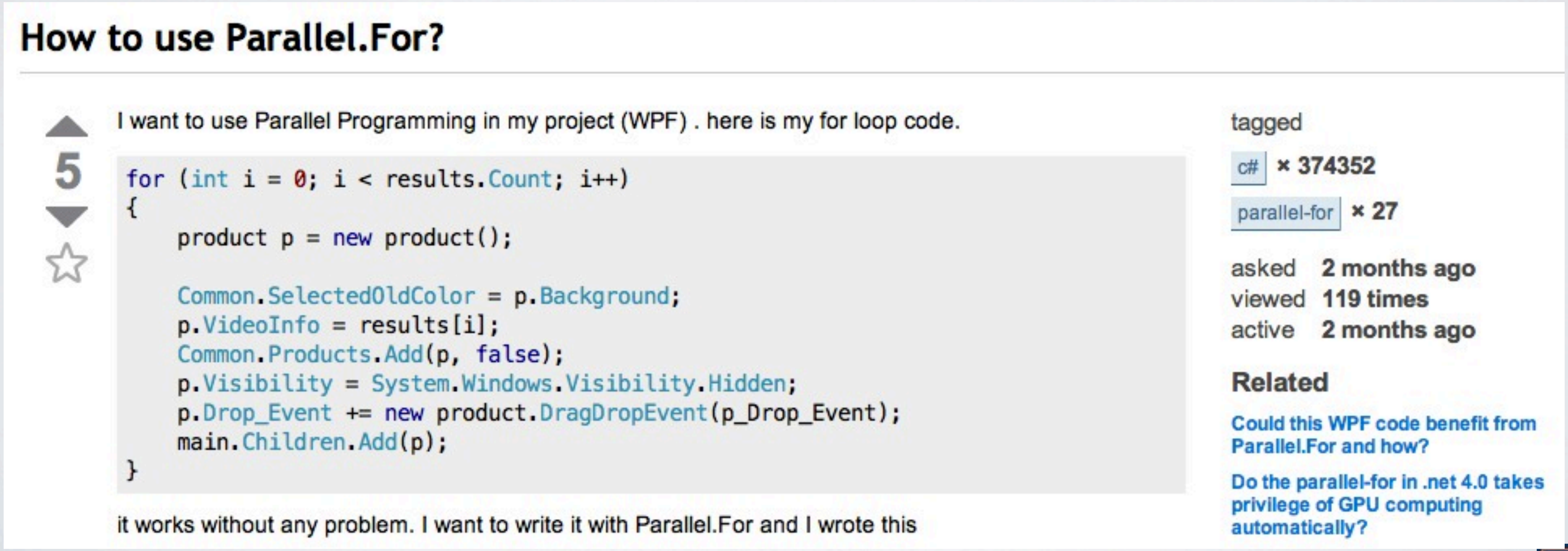
tagged

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?

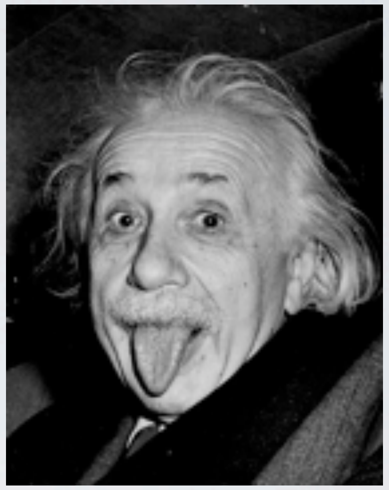
Do the parallel-for in .net 4.0 takes privilege of GPU computing automatically?

Without such a study

Developers

- Cannot decide whether to invest time to learn API
- Miss educational resources

Researchers

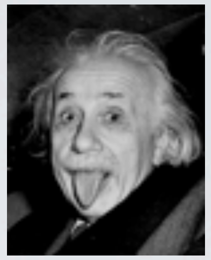


Without such a study

Developers

- Cannot decide whether to invest time to learn API
- Miss educational resources

Researchers



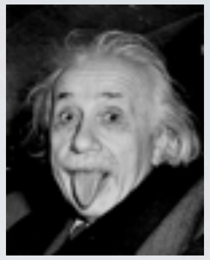
- Make wrong assumptions

Without such a study

Developers

- Cannot decide whether to invest time to learn API
- Miss educational resources

Researchers



- Make wrong assumptions

Library Designers

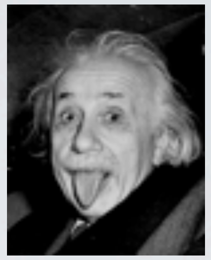
Stephen Toub

Without such a study

Developers

- Cannot decide whether to invest time to learn API
- Miss educational resources

Researchers



- Make wrong assumptions

Library Designers

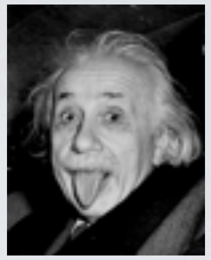
- In danger of designing error-prone, hard to use APIs

Without such a study

Developers

- Cannot decide whether to invest time to learn API
- Miss educational resources

Researchers



- Make wrong assumptions

Library Designers

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

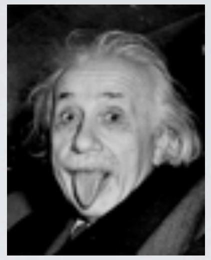
Tool Vendors

Without such a study

Developers

- Cannot decide whether to invest time to learn API
- Miss educational resources

Researchers



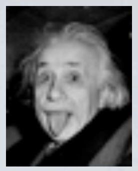
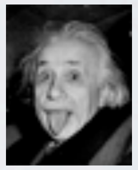
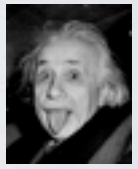
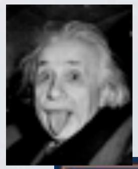
- Make wrong assumptions

Library Designers

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

Tool Vendors

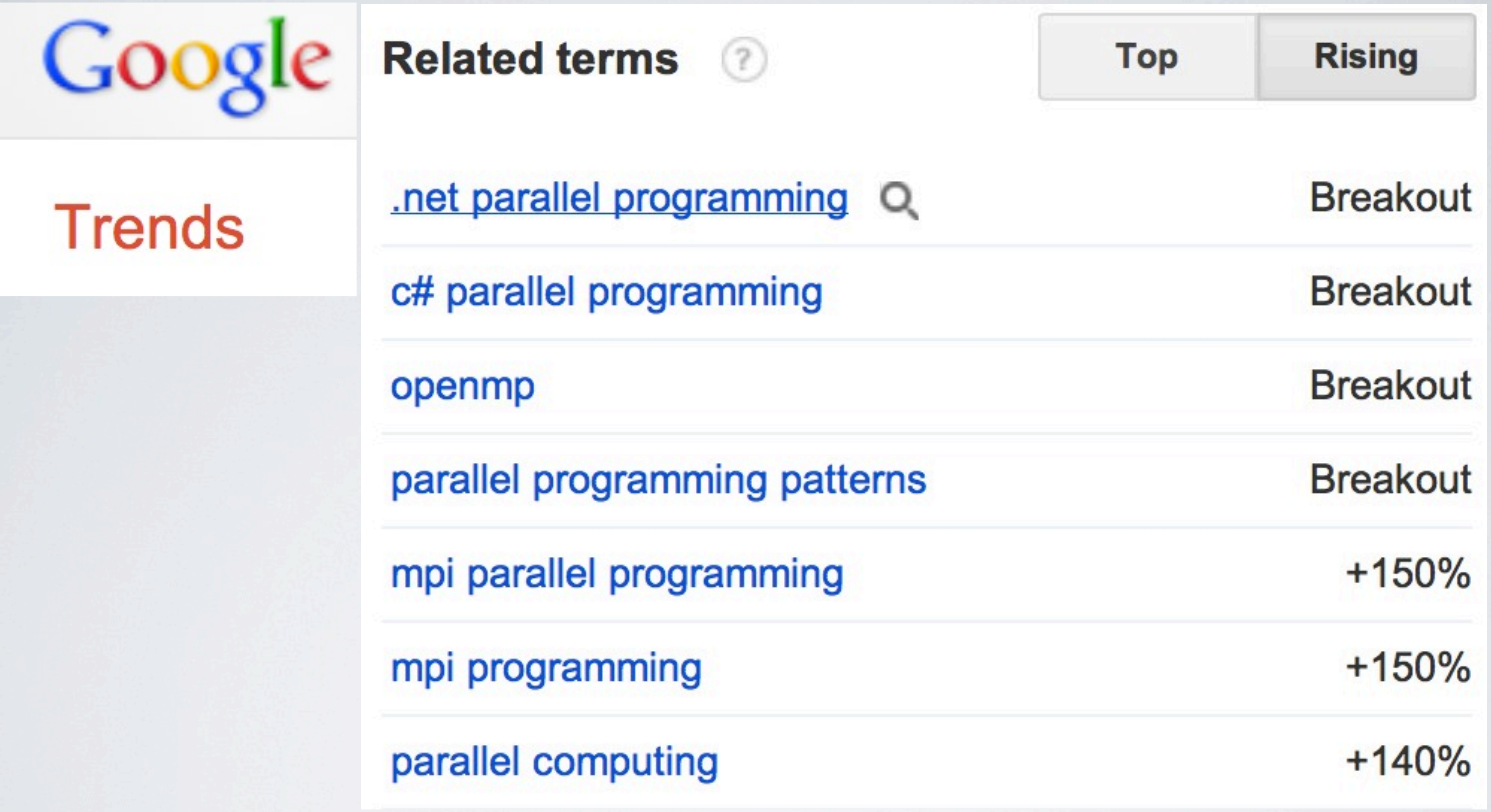
- Don't know what to automate

- Are developers embracing multi-threading? 
- How quickly do developers start using the newly released parallel libraries?
- Which parallel constructs do developers use most often?
- Which parallel patterns do developers embrace?
- Which advanced features do developers use?
- How do developers protect accesses to shared variables? 
- 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?

RISING TREND: C# PARALLEL



.net parallel programming	Breakout
c# parallel programming	Breakout
openmp	Breakout
parallel programming patterns	Breakout
mpi parallel programming	+150%
mpi programming	+150%
parallel computing	+140%

C# PARALLEL API

2002

2010

.NET 1.0 is released

Threading

Old

.NET 4.0 is released

Collections.Concurrent
PLINQ
TPL

New

CORPUS OF DATA

- Downloaded all C# projects as of January 31, 2012.

Filters:

1. Discarded toy applications (less than 1000 non-comment SLOC)

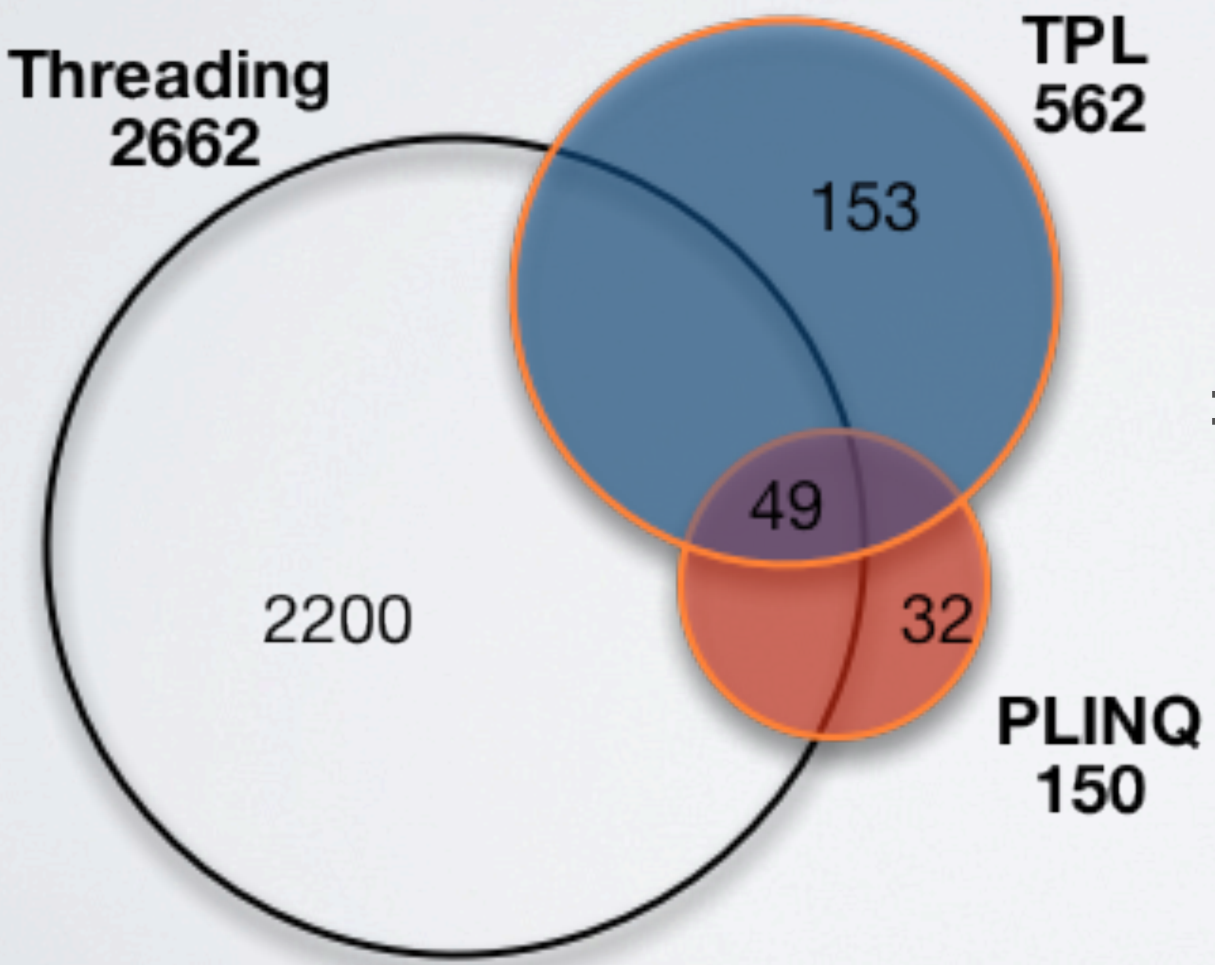
2. Compilable and targeting .NET 4.0

- After all, got 7778 applications

CORPUS OF DATA

Size According to SLOC =>	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



# Apps in Colored Area: 655

655 applications = 17.6M SLOC by 1609 programmers

AUTOMATED STATIC ANALYSIS

based on Microsoft Roslyn Project

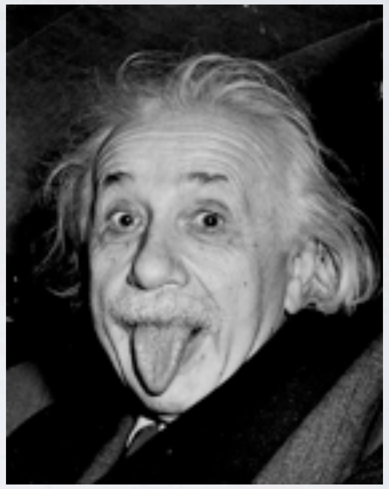
- syntactic (lexical) analysis: traversing AST nodes
  used Syntax API of Roslyn
- semantic analysis: get type and object binding information
  used Symbol and Binding APIs of Roslyn

Collected the usage details for each construct in 4 libraries
patterns based on heuristics

What

Present the results for 3 of our 8 research questions

RQ1: ARE DEVELOPERS EMBRACING MULTI-THREADING?



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

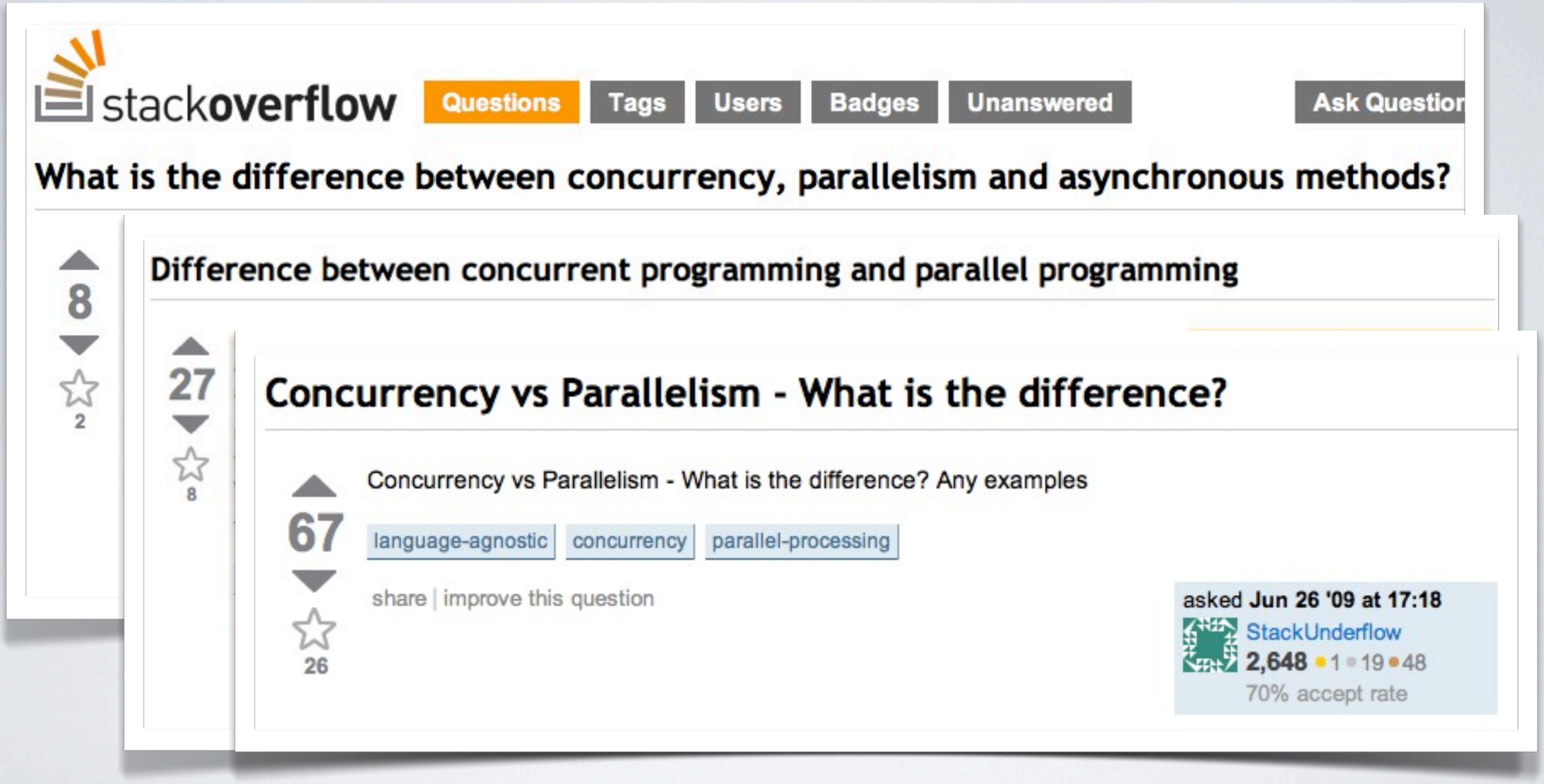
- We built multicores but do developers exploit parallelism?

RQ1: 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%

RQ1: HOW MANY APPS USE PARALLELISM VS CONCURRENCY?



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

Concurrency vs Parallelism - What is the difference? Any examples

language-agnostic concurrency parallel-processing

share | improve this question

26

asked Jun 26 '09 at 17:18
StackUnderflow
2,648 ●1 ●19 ●48
70% accept rate

What

RQ1: HOW MANY APPS USE PARALLELISM VS CONCURRENCY?

Parallelism

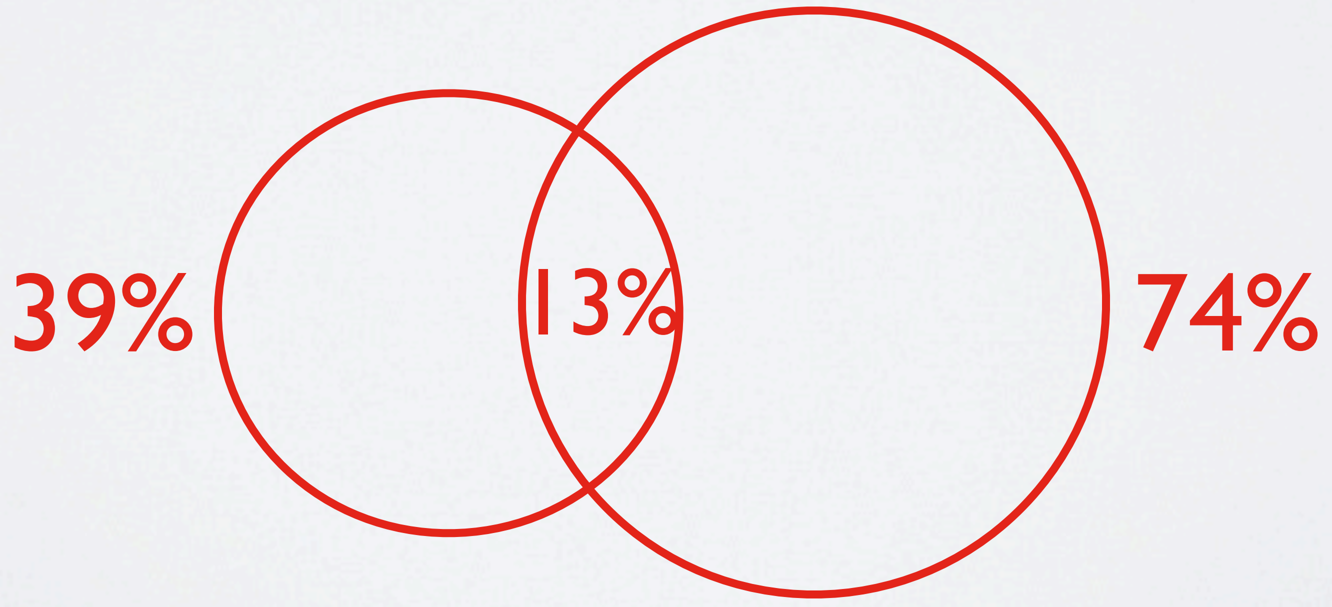
Parallel.For, PLINQ constructs

Parent thread forks worker threads and waits for them to finish

Concurrency

FromAsync, TaskCompletion-Source, UI event dispatching thread

Parent thread does not wait for worker threads



RQ1: ARE DEVELOPERS EMBRACING MULTI-THREADING?

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

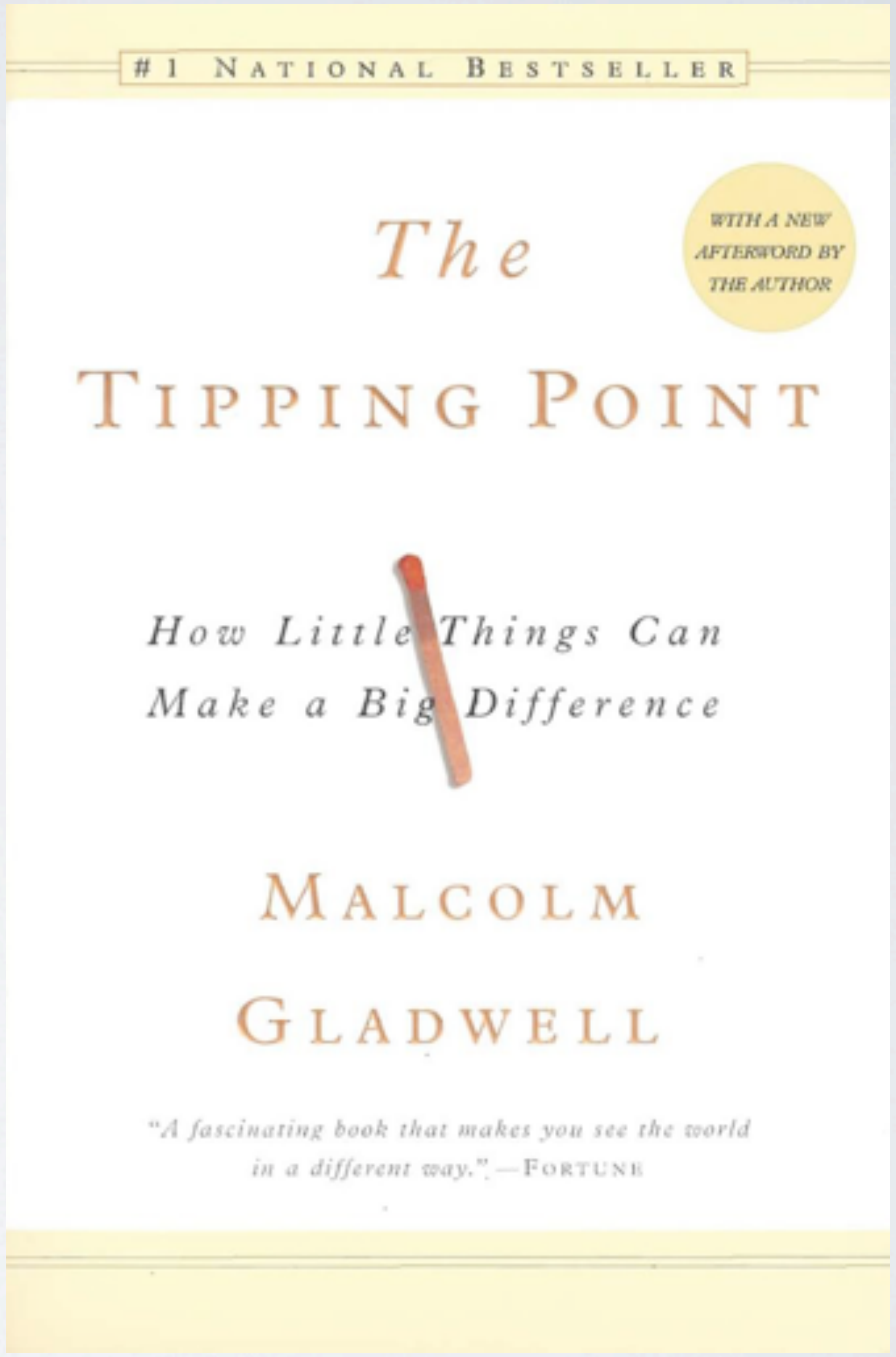
- Developers will not be able to avoid multi-threaded programming for much longer
- Intel should be happy. Developers are taking advantage of multicores

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?

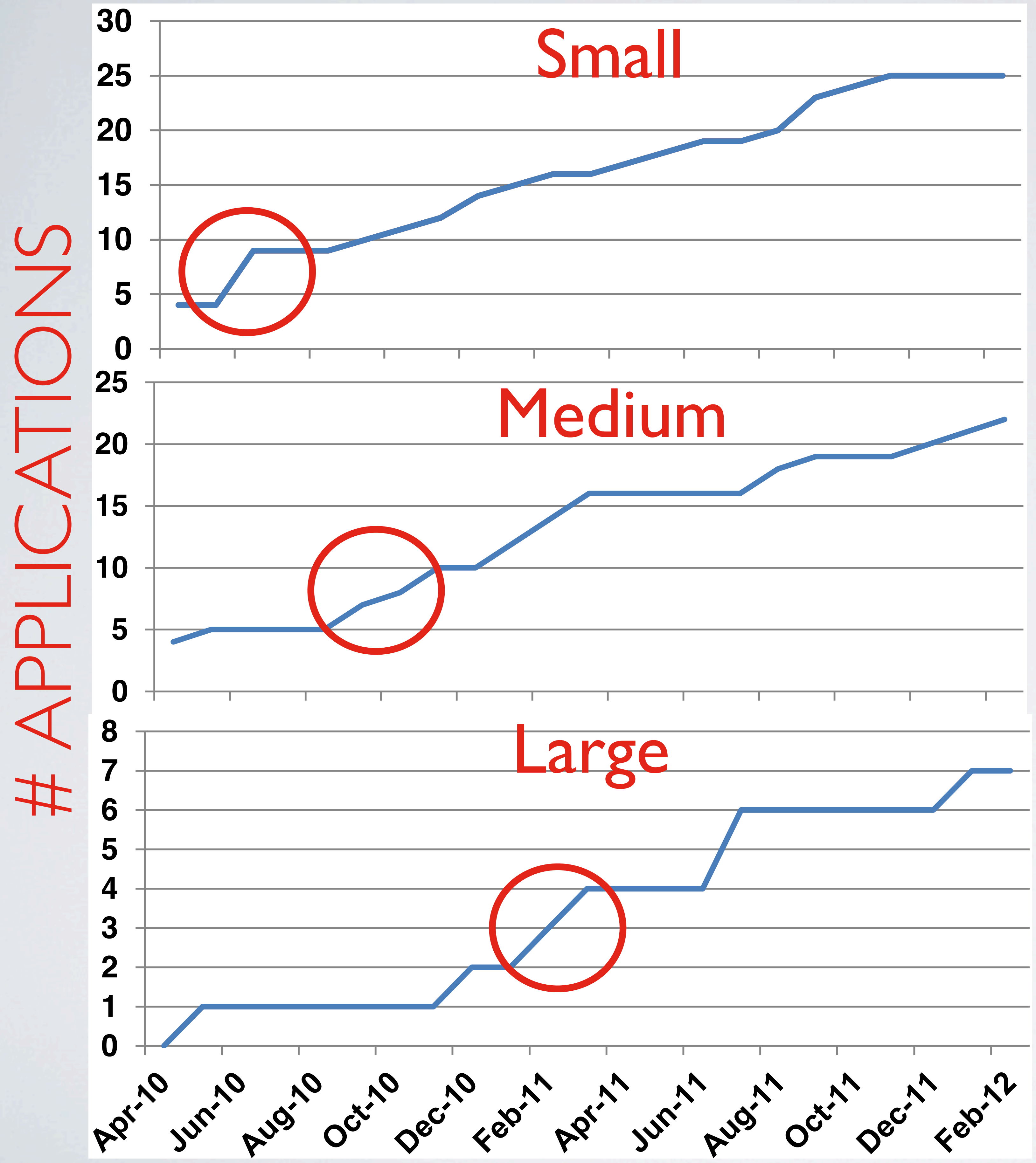
- Purpose is to find out the tipping point for new parallel constructs
- Tipping point: magic moment when a trend spreads like wildfire


#1 National Bestseller
With a new afterword by the author
The Tipping Point
How Little Things Can Make a Big Difference
Malcolm Gladwell
"A fascinating book that makes you see the world in a different way." —Fortune

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

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.

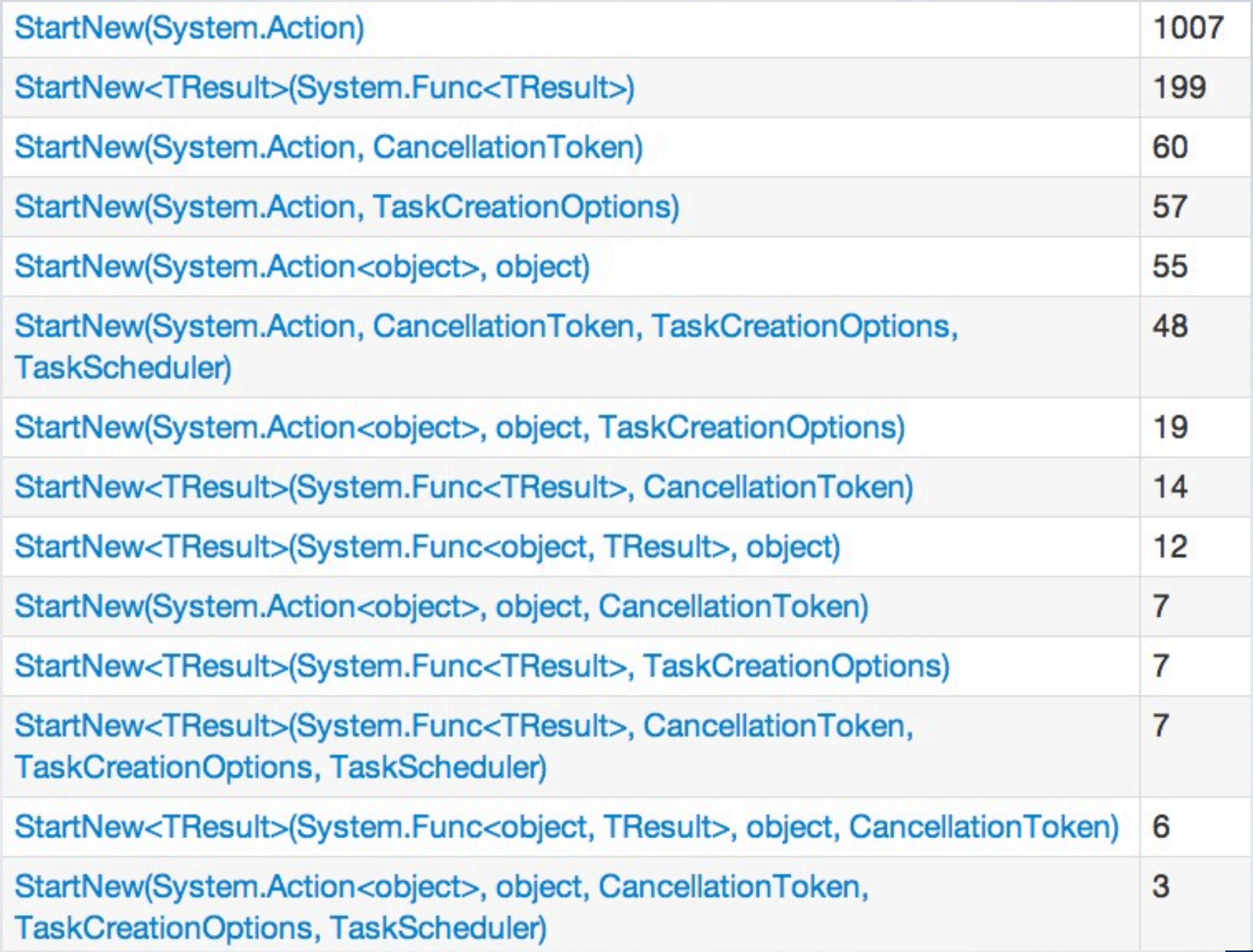
Q3: WHICH PARALLEL CONSTRUCTS DO DEVELOPERS USE MOST OFTEN?

- 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?

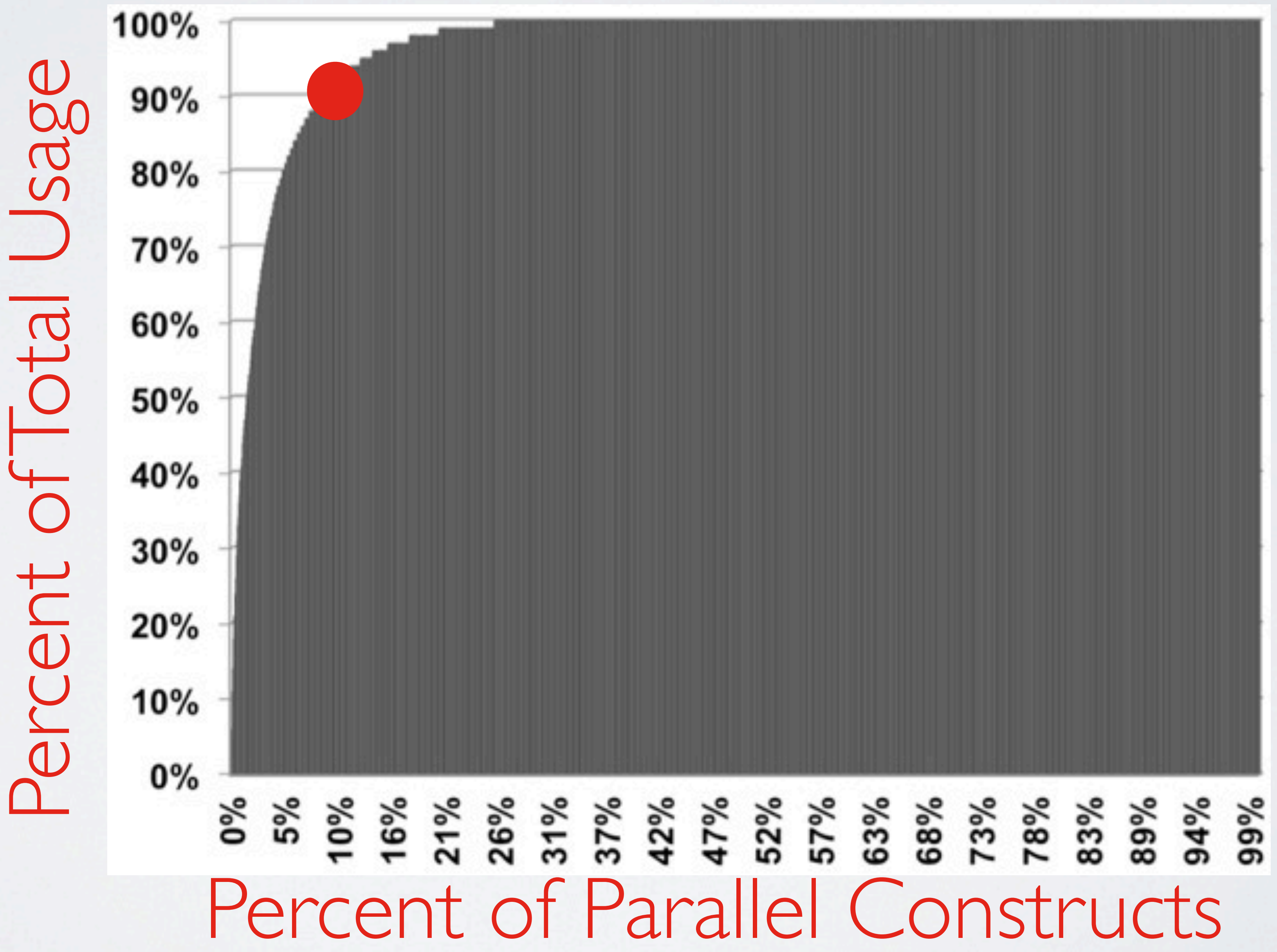
- Usage details for 4 Libraries, 138 classes, 1651 methods
- Detected each method call, class constructor call and got the type of the caller and callee by using Symbol and Binding APIs
- Type information: 100% precise



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

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.



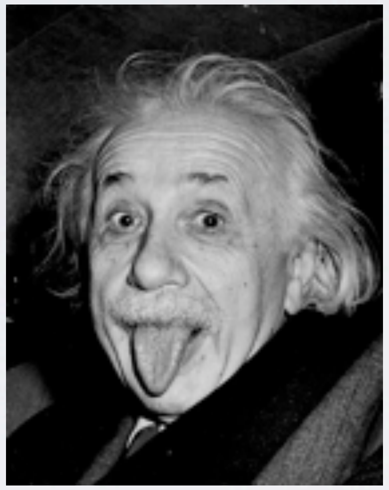
Q3: WHICH PARALLEL CONSTRUCTS DO DEVELOPERS USE MOST OFTEN?

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?

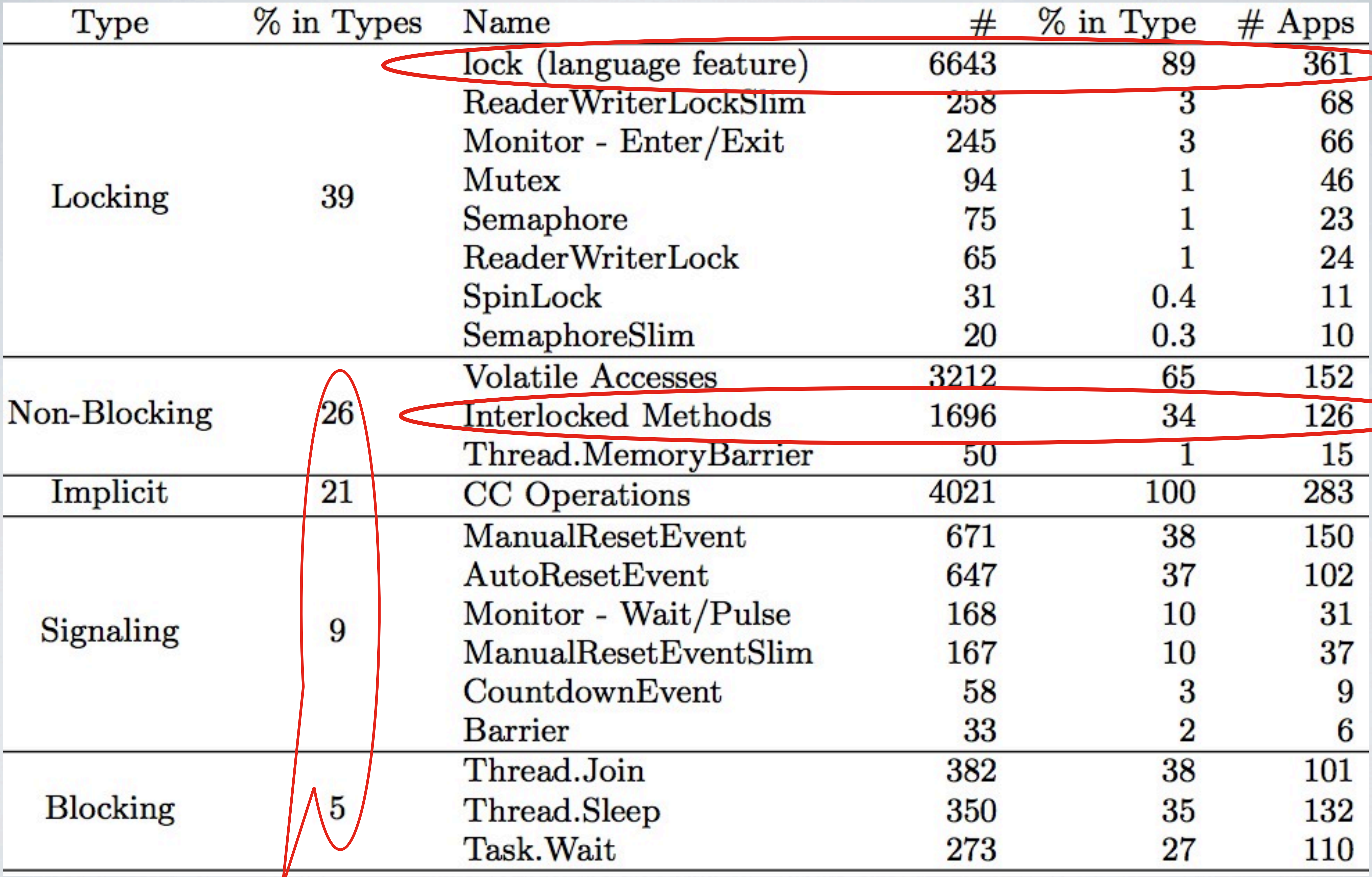


Which synchronization types should we model in our algorithms and tools?

Q4: HOW DO DEVELOPERS PROTECT ACCESSES TO SHARED VARIABLES?

- Around 25 different synchronization constructs in 5 different categories:

  Locking, non-blocking, signaling, implicit, blocking



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%

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!

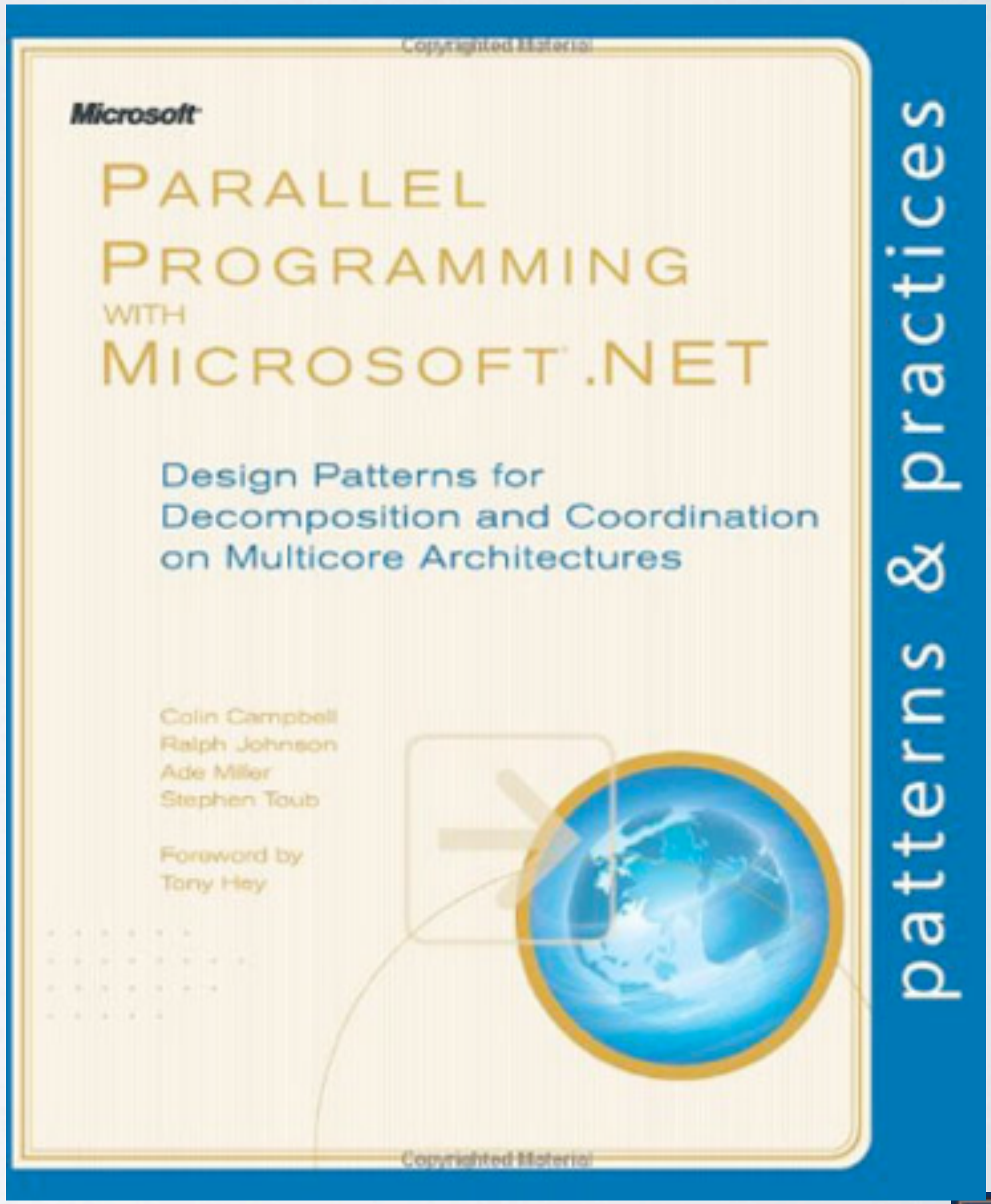
Q5: WHICH PARALLEL PATTERNS DO DEVELOPERS EMBRACE?

- 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


Microsoft
PARALLEL PROGRAMMING WITH MICROSOFT .NET
Design Patterns for Decomposition and Coordination on Multicore Architectures
Colin Campbell
Ralph Johnson
Ade Miller
Stephen Toub
Foreword by Tony Hey
patterns & practices

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?

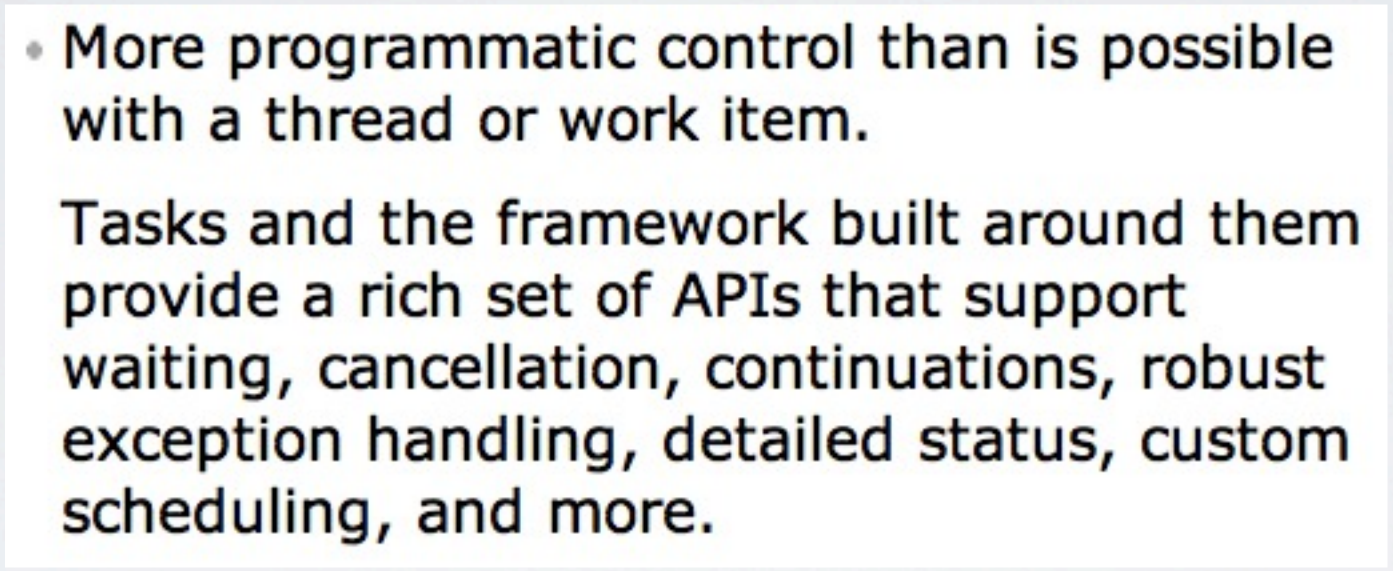
Regular data parallelism is the most used parallel pattern in practice

- 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.

RQ7: DO DEVELOPERS MAKE THEIR PARALLEL CODE UNNECESSARILY COMPLEX?

Parallel code is much more complex than sequential code.

- Why is my parallel code so complex?
- Can we simplify the parallel code?

RQ7: DO DEVELOPERS MAKE THEIR PARALLEL CODE UNNECESSARILY COMPLEX?

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

"ravendb" [github.com/ravendb/ravendb]

↓

```
Parallel.For(1,threadCount,(i)=>DoInsert(..))
```

29% of all cases creating tasks in loop could have used Parallel.For or Parallel.ForEach

RQ7: DO DEVELOPERS MAKE THEIR PARALLEL CODE UNNECESSARILY COMPLEX?

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

$\downarrow$

```
 Parallel.Invoke(RunDaemonJobs, ...,RunScheduledJobs);
```

63 out of 268 regular fork/join task parallelism, the programmers could have used Parallel.Invoke

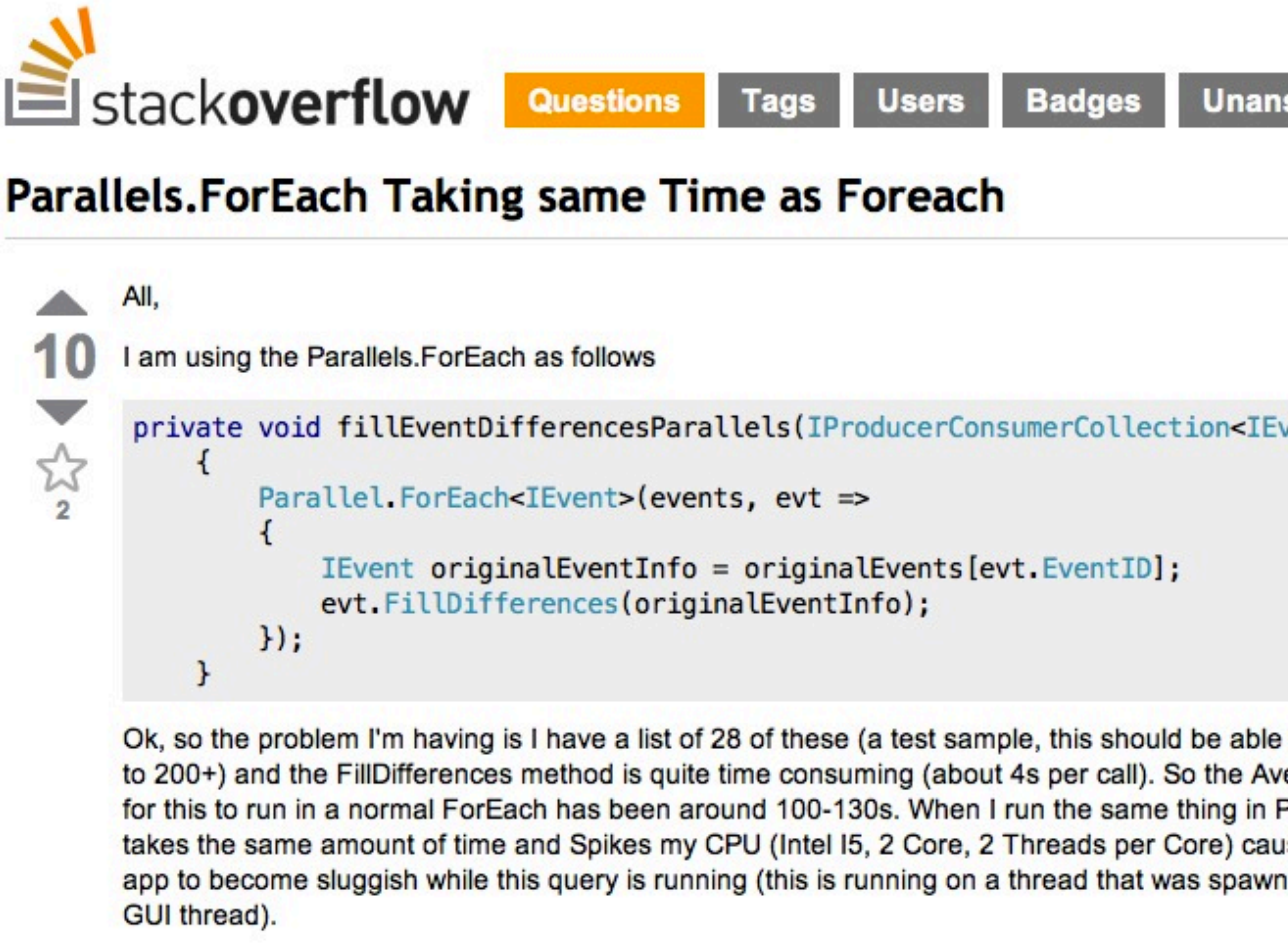
RQ7: DO DEVELOPERS MAKE THEIR PARALLEL CODE UNNECESSARILY COMPLEX?

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

- Refactorings to improve the readability of parallel code are desperately needed

RQ8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

- Why does not my code get any speedup with parallel constructs?
- What are these constructs that developers misuse?


stackoverflow Questions Tags Users Badges Unans

Parallels.ForEach Taking same Time as Foreach

10

2

All,

I am using the Parallels.ForEach as follows

```
private void fillEventDifferencesParallels(IProducerConsumerCollection<IEv
    {
        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 200+) and the FillDifferences method is quite time consuming (about 4s per call). So the Ave
for this to run in a normal ForEach has been around 100-130s. When I run the same thing in P
takes the same amount of time and Spikes my CPU (Intel I5, 2 Core, 2 Threads per Core) cau
app to become sluggish while this query is running (this is running on a thread that was spawn
GUI thread).

RQ8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

- Why does not my code get any speedup with parallel constructs?
- Which constructs do developers misuse?

RQ8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

AsParallel() from PLINQ

Correct

```
assembly.GetTypes().AsParallel().
      Where(t => t.IsSubclassOf(...)).
      ForAll(t => controllersCache.Add(...));
```

RQ8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

Incorrect

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

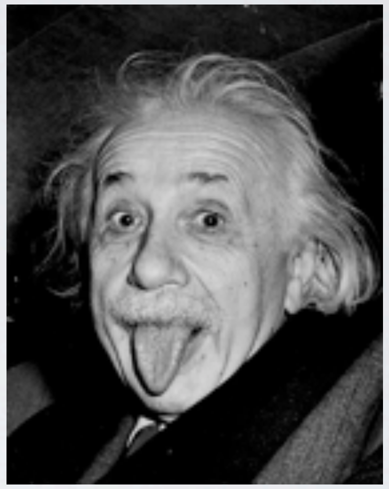
"profit" [profit.codeplex.com]

- The `foreach` proceeds sequentially.
- 12% of all `AsParallel` usages are incorrect

RQ8: ARE THERE CONSTRUCTS THAT DEVELOPERS COMMONLY MISUSE?

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

SO WHAT?

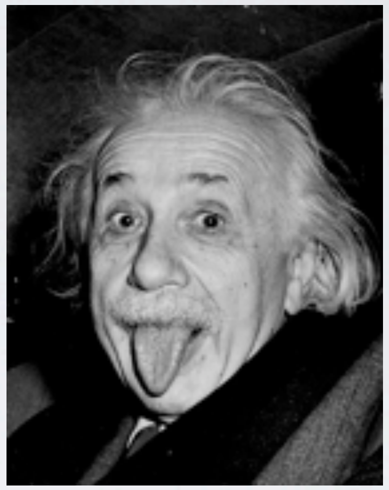


LIBRARY DESIGNERS

Learn how to make the APIs easier and less error-prone to use

Confirmed that our suggestions are useful and will influence the future development of the libraries

TOOL VENDORS & RESEARCHERS



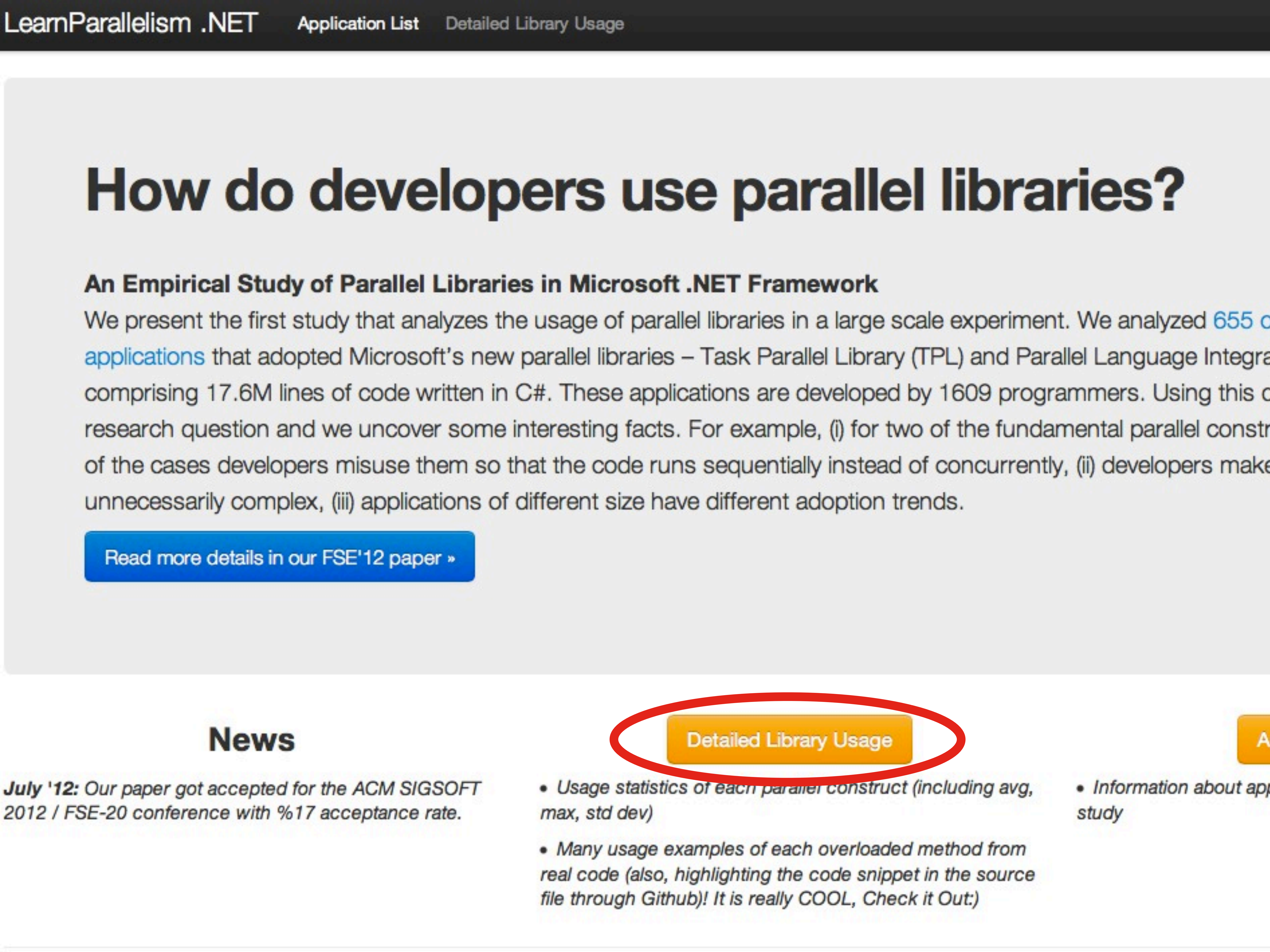
Focus on their efforts on widely (mis)used constructs and patterns

FOR DEVELOPERS

we created

http://LearnParallelism.NET

3700 unique organic visitors from 88 different countries generates 20231 page views in 4 months



How do developers use parallel libraries?

An Empirical Study of Parallel Libraries in Microsoft .NET Framework

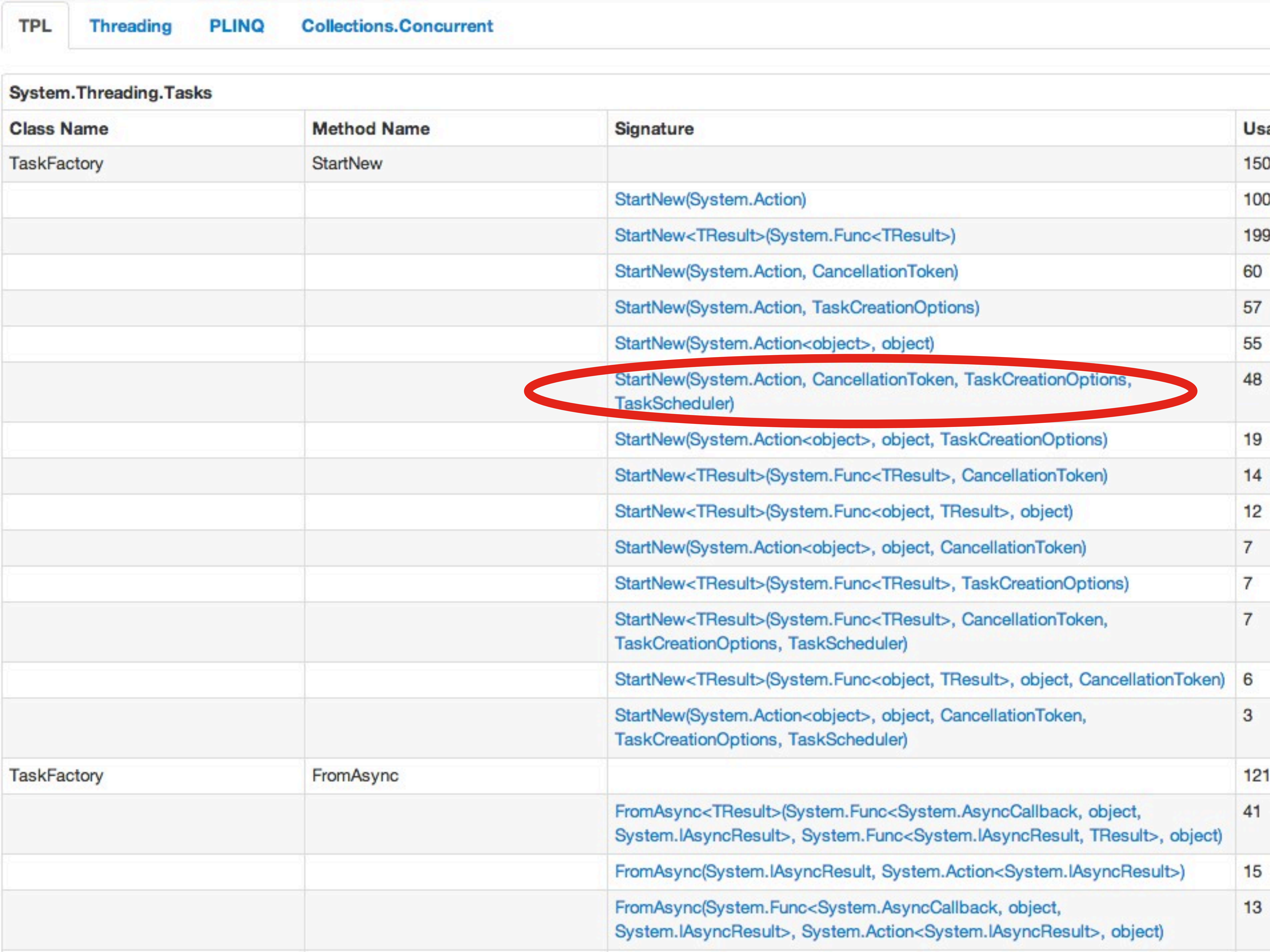
We present the first study that analyzes the usage of parallel libraries in a large scale experiment. We analyzed 655 c applications that adopted Microsoft's new parallel libraries – Task Parallel Library (TPL) and Parallel Language Integra comprising 17.6M lines of code written in C#. These applications are developed by 1609 programmers. Using this c research question and we uncover some interesting facts. For example, (i) for two of the fundamental parallel constr of the cases developers misuse them so that the code runs sequentially instead of concurrently, (ii) developers make unnecessarily complex, (iii) applications of different size have different adoption trends.

News

July '12: Our paper got accepted for the ACM SIGSOFT 2012 / FSE-20 conference with %17 acceptance rate.

- *Usage statistics of each parallel construct (including avg, max, std dev)*
- *Many usage examples of each overloaded method from real code (also, highlighting the code snippet in the source file through Github)! It is really COOL, Check it Out:)*

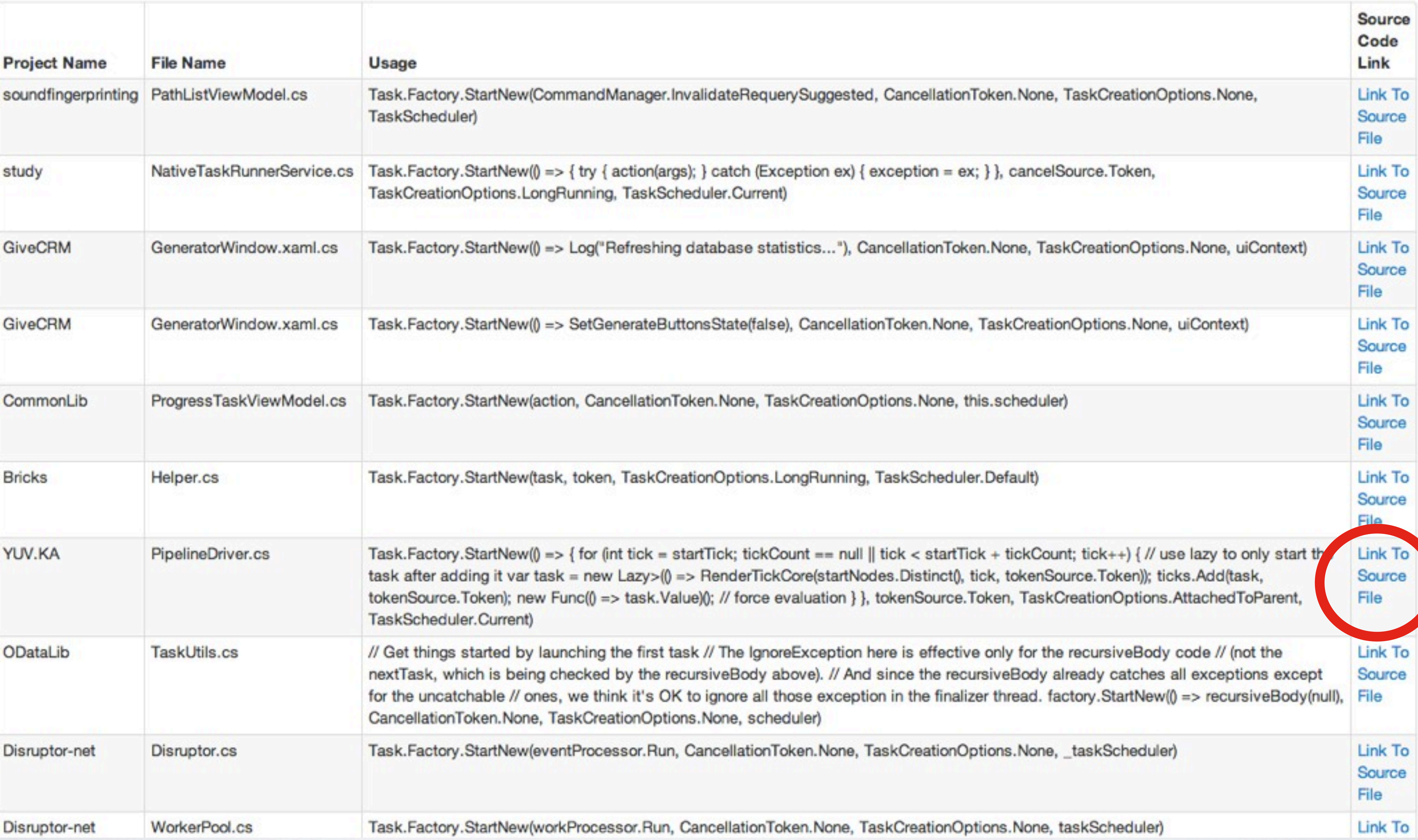
- *Information about ap study*



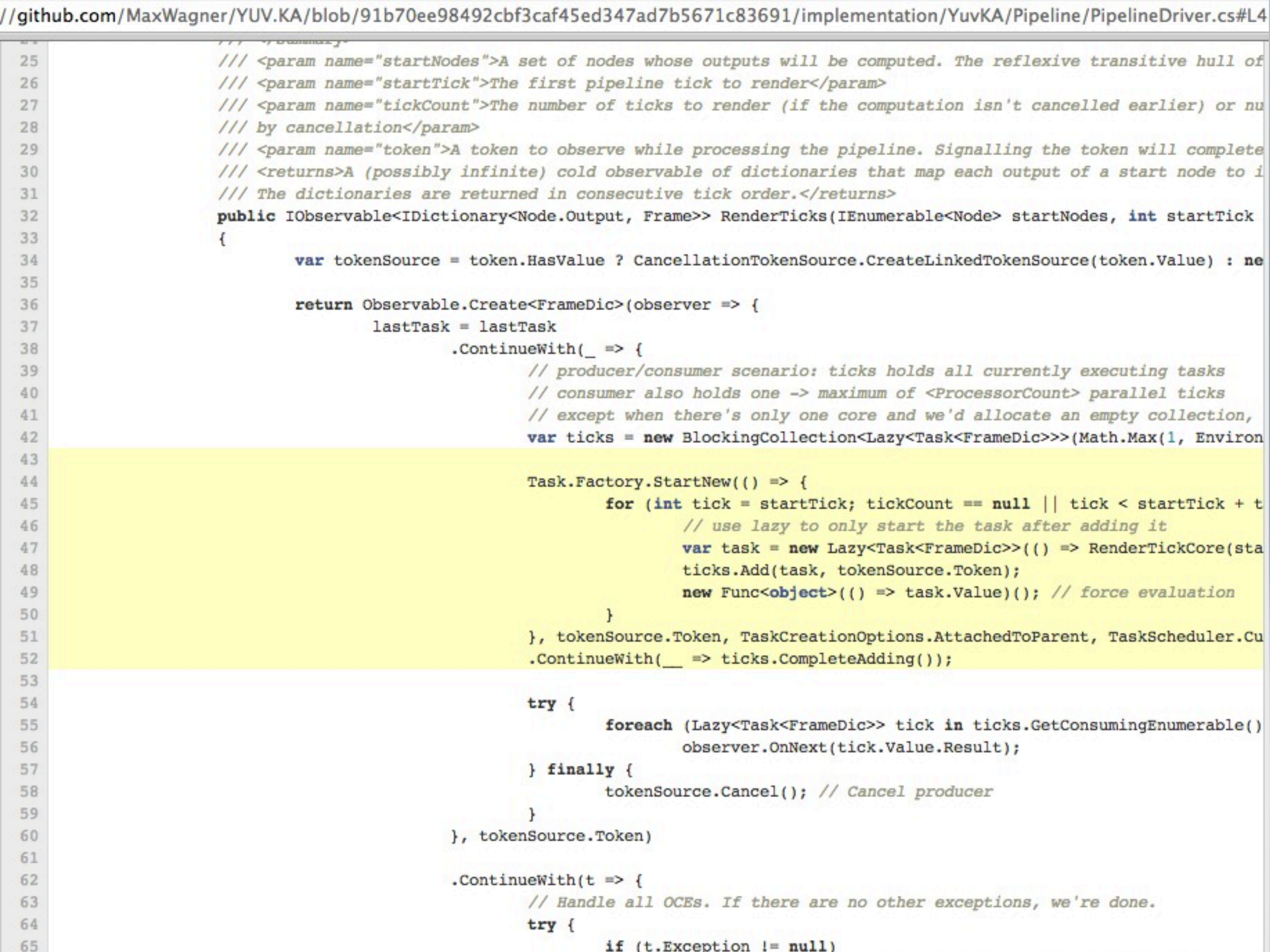
TPL Threading PLINQ Collections.Concurrent

System.Threading.Tasks

Class Name	Method Name	Signature	Usa
TaskFactory	StartNew		150
		StartNew(System.Action)	100
		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
TaskFactory	FromAsync		121
		FromAsync<TResult>(System.Func<System.AsyncCallback, object, System.IAsyncResult>, System.Func<System.IAsyncResult, TResult>, object)	41
		FromAsync(System.IAsyncResult, System.Action<System.IAsyncResult>)	15
		FromAsync(System.Func<System.AsyncCallback, object, System.IAsyncResult>, System.Action<System.IAsyncResult>, object)	13



Project Name	File Name	Usage	Source Code Link
soundfingerprinting	PathListViewModel.cs	Task.Factory.StartNew(CommandManager.InvalidateRequerySuggested, CancellationToken.None, TaskCreationOptions.None, TaskScheduler)	Link To Source File
study	NativeTaskRunnerService.cs	Task.Factory.StartNew(() => { try { action(args); } catch (Exception ex) { exception = ex; } }, cancelSource.Token, TaskCreationOptions.LongRunning, TaskScheduler.Current)	Link To Source File
GiveCRM	GeneratorWindow.xaml.cs	Task.Factory.StartNew(() => Log("Refreshing database statistics..."), CancellationToken.None, TaskCreationOptions.None, uiContext)	Link To Source File
GiveCRM	GeneratorWindow.xaml.cs	Task.Factory.StartNew(() => SetGenerateButtonsState(false), CancellationToken.None, TaskCreationOptions.None, uiContext)	Link To Source File
CommonLib	ProgressTaskViewModel.cs	Task.Factory.StartNew(action, CancellationToken.None, TaskCreationOptions.None, this.scheduler)	Link To Source File
Bricks	Helper.cs	Task.Factory.StartNew(task, token, TaskCreationOptions.LongRunning, TaskScheduler.Default)	Link To Source File
YUV.KA	PipelineDriver.cs	Task.Factory.StartNew(() => { for (int tick = startTick; tickCount == null || tick < startTick + tickCount; tick++) { // use lazy to only start th task after adding it var task = new Lazy>(() => RenderTickCore(startNodes.Distinct(), tick, tokenSource.Token)); ticks.Add(task, tokenSource.Token); new Func(() => task.Value)(); // force evaluation } }, tokenSource.Token, TaskCreationOptions.AttachedToParent, TaskScheduler.Current)	Link To Source File
ODataLib	TaskUtils.cs	// Get things started by launching the first task // The IgnoreException here is effective only for the recursiveBody code // (not the nextTask, which is being checked by the recursiveBody above). // And since the recursiveBody already catches all exceptions except for the uncatchable // ones, we think it's OK to ignore all those exception in the finalizer thread. factory.StartNew(() => recursiveBody(null), CancellationToken.None, TaskCreationOptions.None, scheduler)	Link To Source File
Disruptor-net	Disruptor.cs	Task.Factory.StartNew(eventProcessor.Run, CancellationToken.None, TaskCreationOptions.None, _taskScheduler)	Link To Source File
Disruptor-net	WorkerPool.cs	Task.Factory.StartNew(workProcessor.Run, CancellationToken.None, TaskCreationOptions.None, taskScheduler)	Link To



//github.com/MaxWagner/YUV.KA/blob/91b70ee98492cbf3caf45ed347ad7b5671c83691/implementation/YuvKA/Pipeline/PipelineDriver.cs#L4

```
/// </summary>
/// <param name="startNodes">A set of nodes whose outputs will be computed. The reflexive transitive hull of
/// <param name="startTick">The first pipeline tick to render</param>
/// <param name="tickCount">The number of ticks to render (if the computation isn't cancelled earlier) or nu
/// by cancellation</param>
/// <param name="token">A token to observe while processing the pipeline. Signalling the token will complete
/// <returns>A (possibly infinite) cold observable of dictionaries that map each output of a start node to i
/// The dictionaries are returned in consecutive tick order.</returns>
public IObservable<IDictionary<Node.Output, Frame>> RenderTicks(IEnumerable<Node> startNodes, int startTick
{
    var tokenSource = token.HasValue ? CancellationTokenSource.CreateLinkedTokenSource(token.Value) : ne

    return Observable.Create<FrameDic>(observer => {
        lastTask = lastTask
            .ContinueWith(_ => {
                // producer/consumer scenario: ticks holds all currently executing tasks
                // consumer also holds one -> maximum of <ProcessorCount> parallel ticks
                // except when there's only one core and we'd allocate an empty collection,
                var ticks = new BlockingCollection<Lazy<Task<FrameDic>>>(Math.Max(1, Environ

                Task.Factory.StartNew(() => {
                    for (int tick = startTick; tickCount == null || tick < startTick + t
                        // use lazy to only start the task after adding it
                        var task = new Lazy<Task<FrameDic>>(() => RenderTickCore(sta
                        ticks.Add(task, tokenSource.Token);
                        new Func<object>(() => task.Value)(); // force evaluation
                    }
                }, tokenSource.Token, TaskCreationOptions.AttachedToParent, TaskScheduler.Cu
                .ContinueWith(__ => ticks.CompleteAdding());

                try {
                    foreach (Lazy<Task<FrameDic>> tick in ticks.GetConsumingEnumerable()
                        observer.OnNext(tick.Value.Result);
                } finally {
                    tokenSource.Cancel(); // Cancel producer
                }
            }, tokenSource.Token)

            .ContinueWith(t => {
                // Handle all OCEs. If there are no other exceptions, we're done.
                try {
                    if (t.Exception != null)
```

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