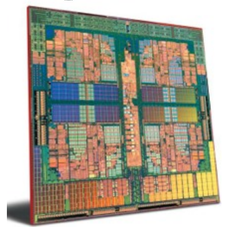
How do Developers Use Parallel Libraries?

Semih Okur, Danny Dig

Motivation

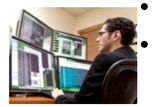
We are living in the multicore age



Parallel programming is hard

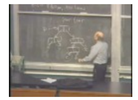
- TPL,Threading,PLINQ
- java.util.concurrent
- TBB, OpenMP

Yet, we know little about how practitioners use these libraries in practice



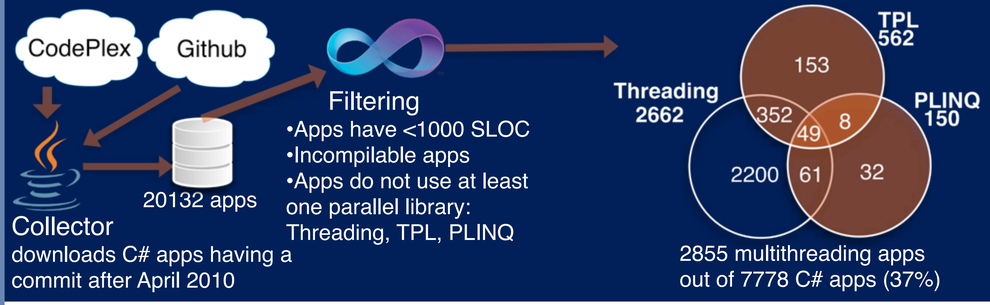
A huge community can benefit

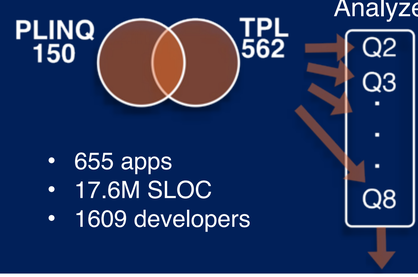
- Researchers
- Library developers
- Instructors
- Developers



Q1: Are developers embracing multi-threading?
Q2: How quickly do developers start using new parallel libraries?
Q3: Which parallel constructs do developers use most often?
Q4: How do developers protect accesses to shared variables?
Q5: Which parallel patterns do developers embrace?
Q6: Which advanced features do developers use?
Q7: Do developers make the parallel code unnecessarily complex?
Q8: Are there constructs that developers commonly misuse?

Methodology





- Implemented a specific analysis for each question
- Used Microsoft Roslyn APIs
- Q2 .. Q8 analyze 655 apps
- Only Q1 analyzes all 7778 apps
- Both syntactic and semantic analysis
- Detected the usage of parallel constructs (138 classes, 1651 methods) at 100% accuracy

Interesting Facts

10% of their "parallel" code runs sequentially!!

```
Parallel.Invoke(() => i.ImportGPX(null, GPXFile));
```

Parallel.Invoke executes in parallel the actions passed as arguments. We found 11% of all usages of this take one action parameter in different apps. Developers believe that ImportGPX will execute in parallel.

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

Any method called on the object that AsParallel() returns will execute in parallel. We found that 12% of all AsParallel are used as the iteration source of a sequential loop. Developers again believe that the code will run in parallel

Developers make their parallel code unnecessarily complex

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

↓

```
Parallel.Invoke(new ParallelOptions(CancellationToken =..token),
RunDaemonJobs , ..., RunScheduledJobs);
```

63 out of 268 cases could have used Parallel.Invoke instead of regular fork/join task parallelism

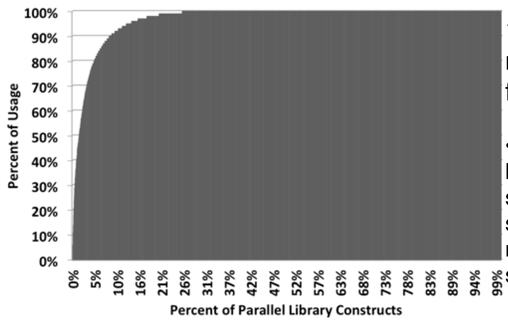
```
for( int i=1; i<=threadCount; i++) {
   var copy= I;
   var taskHandle= Task.Factory.StartNew(()=>
      DoInefficientInsert(server.Database.
            Configuration.ServerUrl, copy);
   tasks.Add(taskHandle);              }
Task.WaitAll(tasks);
```

↓

```
Parallel.For(1, threadCount, (i)=> DoInefficientInsert(
   server,Database.Configuration.ServerUrl, i) );
```

55 out of 189 cases could have used Parallel.For or Parallel.ForEach instead of regular for loop parallelism

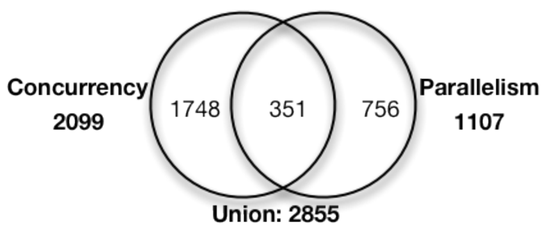
Parallel library usage follows a power-law distribution



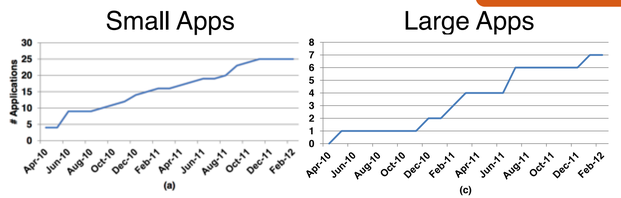
10% of the parallel API methods are responsible for 90% of all usages

•Beginners can focus on learning a relatively small subset of the library APIs and still be able to master a large number of parallelism scenarios

Cool Statistics

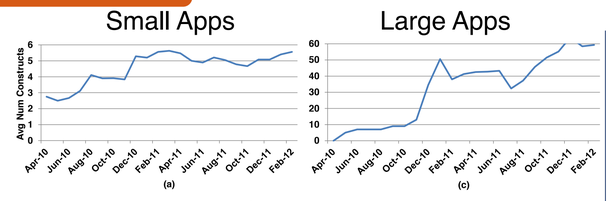


The purpose of multi-threading
74% for concurrency
39% for parallelism



Adoption of parallelism libraries

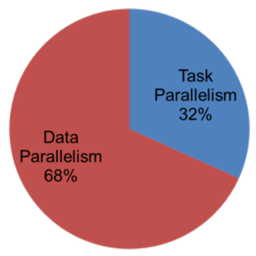
- More applications are becoming parallel



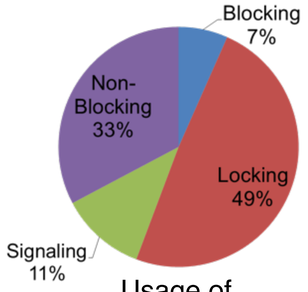
Average number of constructs per application

- Each application is becoming more parallel

The small apps are the early adopters of new libraries. Larger ones are late adopters.



Usage of Parallel Patterns



Usage of Synchronization Types

For more: LearnParallelism.NET