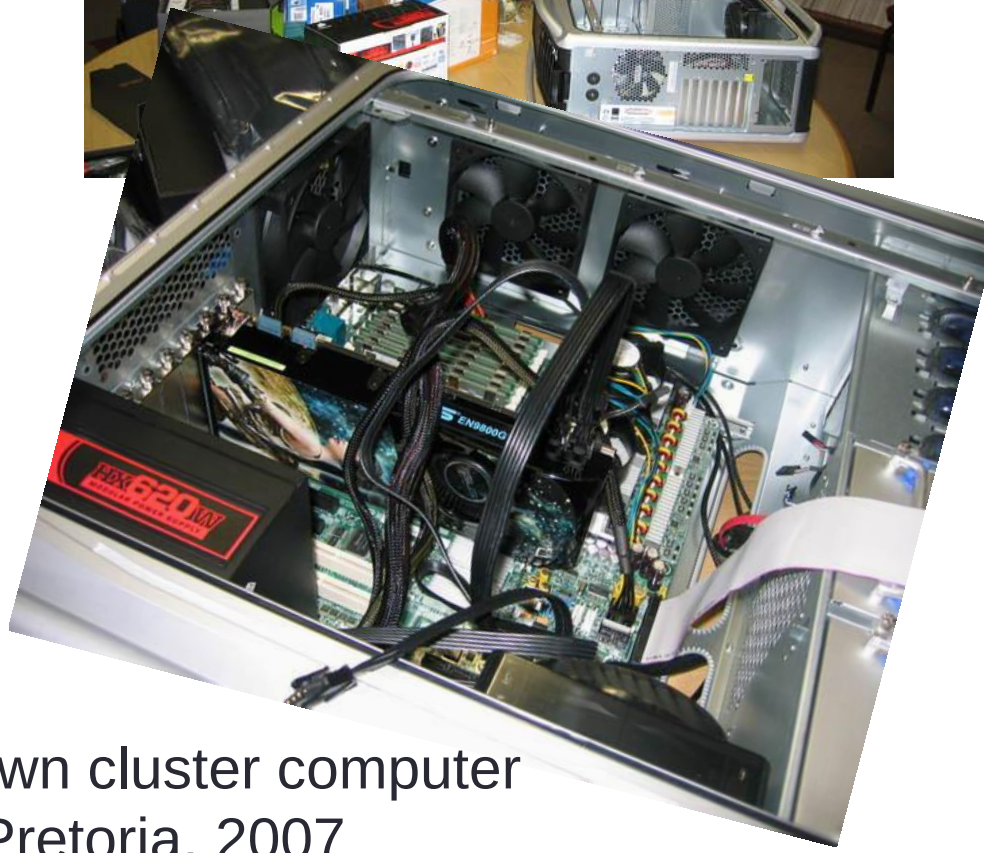
CONCURRENCY IN C# AND JAVA – WHY LANGUAGES MATTER

Judith Bishop
Microsoft Research Connections
Redmond, USA
jbishop@microsoft.co.za



Building our own cluster computer
University of Pretoria, 2007

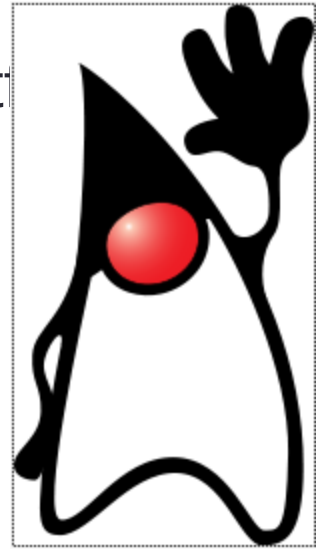
Language and OS Timeline

2001	2002	2003	2004	2005	2006	2007	2008	2009
C# 1.0		Spec#1.0.5	Java 1.5 Cω	C# 2.0 Spec# 1.0.6	Java 6 LINQ	C# 3.0		C#4.0 Working on Java 7
Windows XP .NET Mac OS X	Rotor 1.0		Mono 1.0	.NET 2	Windows Vista Rotor 2.0 Mac OS X on Intel	.NET 3.5 Mac OS X Leopard		Windows 7 .Net 4.0 Mac OS X Snow Leo. Eclipse 3.6

Some History

- **Eclipse** software development environmer
  extensible plug-in system.
  - From IBM Canada, 2001
  - Now in a Foundation
  - Free and open source under its own licencse
  - Strong community base

- Java programming language and its run-time platf
  - From Sun Microsystems in 1995
  - Now with Oracle
  - Free and Open Source under GNU public license
  - Strong research, education and developer base
  - Part of browser technology



The Development of C#

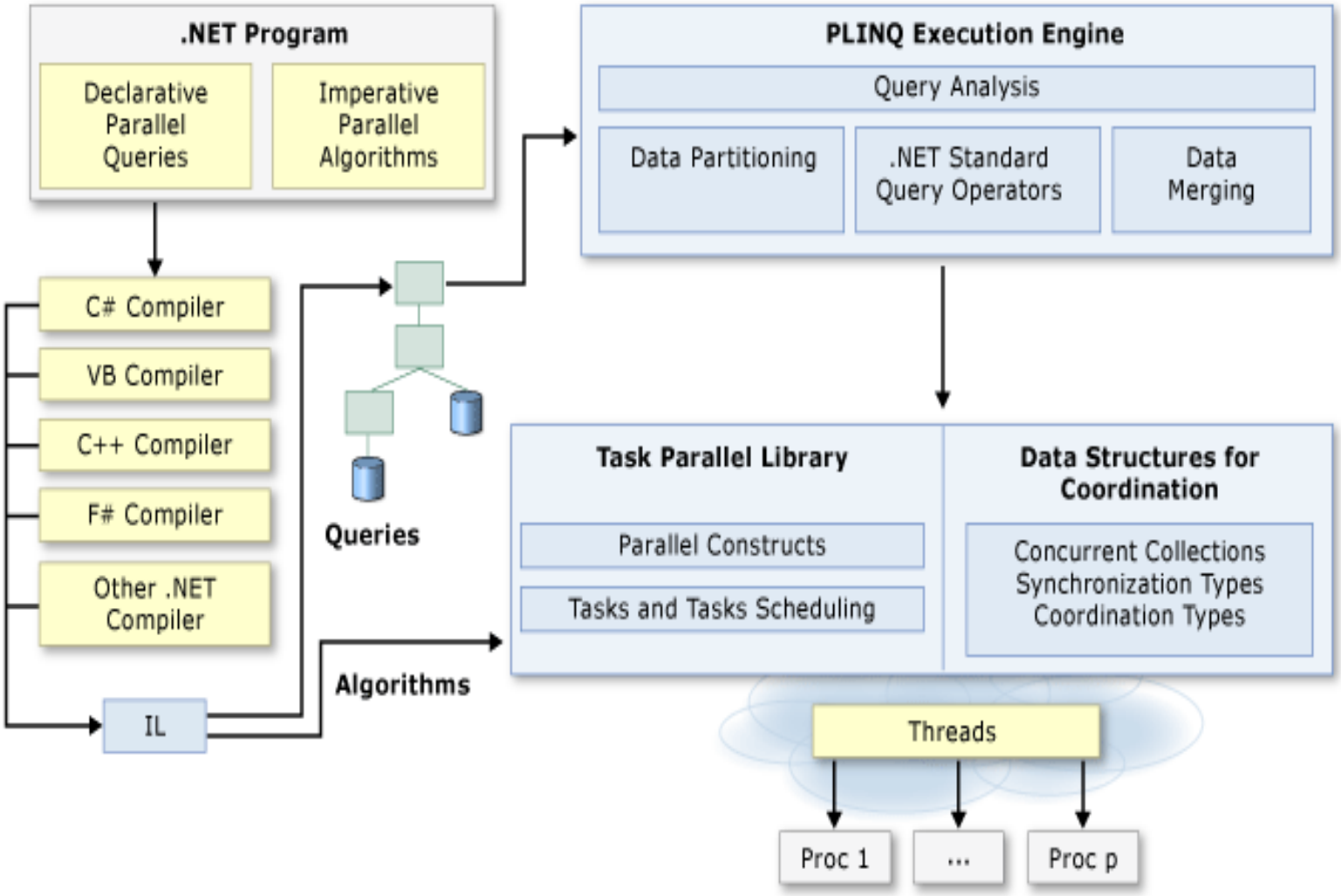
Table 1 The development of C#

C# 1.0 2001	*C# 2.0 2005*	*C# 3.0 2007*	*C# 4.0 2009*
structs properties foreach loops autoboxing delegates and events indexers operator overloading enumerated types with IO in, out and ref parameters formatted output	generics anonymous methods iterators partial types nullable types generic delegates	implicit typing anonymous types object and array initializers extension methods, lambda expressions query expressions (LINQ)	dynamic lookup named and optional arguments COM interop variance
API Serializable reflection		standard generic delegates	

.NET Parallel Landscape

- Activity within Microsoft and Microsoft Research from 2007 to support parallelism
- Supports all .NET languages, e.g. C#, F#, Visual Basic
- C++ requires a different set of libraries

1. **Stephens, Rod**, *Getting Started with the .NET Task Parallel Library*, DevX.com, http://www.devx.com/dotnet/Article/39204/1763/ Sept 2008
2. **Leijen, Daan, Wolfram Schulte and Sebastian Burckhar**dt. *The design of a task parallel library*. Proc. 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications OOPSLA '09, pp 227-242. DOI= http://doi.acm.org/10.1145/1640089.1640106
3. **Toub, Stephen**, *Patterns of parallel programming – understanding and applying patterns with the .NET framework 4 and Visual C#*, White Paper, pp 118, http://www.microsoft.com/downloads/details.aspx?FamilyID=86b3d32b-ad26-4bb8-a3ae-c1637026c3ee&displaylang=en, 2010
4. **Campbell, Colin, Ralph Johnson, Ade Miller and Stephen Toub**, *Parallel Programming with Microsoft® .NET: Design Patterns for Decomposition and Coordination on Multicore Architectures*, Microsoft Press, pp167, at http://parallelpatterns.codeplex.com/, 2010
5. **Freeman, Adam**, Pro *.NET 4 Parallel Programming in C# (Expert's Voice in .NET),* APress, pp328, 2010.



TPL – Task Parallel Library
PLINQ – Parallel Language Integrated Query

The Libraries

TPL

- Intended for speeding up processor-bound computations
- Uses **task** parallelism
- No need to equate tasks with threads or cores
- General abstraction mechanism for asynchronous operations

PLINQ

- Intended to use all cores efficiently
- Exposes **data** parallelism
- Extends LINQ which is based on SQL
- Query operators for objects

What you need for multicore programming

- Scale the degree of parallelism dynamically
- Use all the available cores
- Partition the work
- Balance the load
- Schedule the threads
- Support communication
- Support cancellation
- Manage state
- Other low-level details

Call TPL

We'd also like

- No changes to the language or compiler
- Simplicity and elegance

- Popular options:
- OpenMP – comment and pragma syntax
- MPI – library with lengthy parameter lists

MPI in C#

- MPI libraries exist for many languages
- Uses a Communication World
- Parameter lists can get long

```
static void Main(string[] args) {
  using (new MPI.Environment(ref args)) {
  Intracommunicator comm = Communicator.world;
  string[] hostnames =
        comm.Gather (MPI.Environment.ProcessorName, 0);
  if (comm.Rank == 0) {
      Array.Sort(hostnames);
      foreach(string host in hostnames)
          Console.WriteLine(host);
  }
 }
}
```

OpenMP

- Has special pragmas or comments and a special compiler that translates them
- Message passing with no shared memory

```
#pragma omp parallel shared(a,b,c,chunk) private(i)
  {

  #pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  }  /* end of parallel section */
```

Basic Methods

- Include the library `Systems.Threading.Tasks.Parallel`
- Use delegate expressions or lambda expressions

```
Parallel.For (0, 100, delegate(i) {
  // do some significant work, perhaps using i
});
```

- The body of the delegate forms the body of the loop and the task that is replicated onto cores.

Lambda expressions

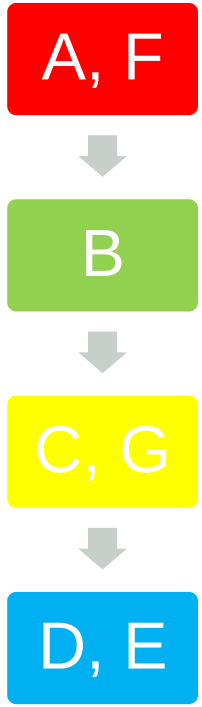
- Equivalent to delegate keyword
- New operator in C# - =>
- For is an overloaded method
- The layout of the brackets is a convention

```
Parallel.For (0, 10, i => {
  Process (i);
});

Parallel.For (0, 10, () => {
  // Do something
});
```

Loops over collections

- A collection is an enumerable structure e.g. array, list
- Sequential foreach construct does not require an index
- Parallel Foreach enables access to elements by all cores

```
// Sequential
foreach (var item in sourceCollection {
  Process(item);
});

Parallel.ForEach (sourceCollection, item => {
  Process(item)
});
```



Invoke method

- A set of tasks may be executed in parallel

```
static void ParQuickSort<T>(T[] domain, int lo, int hi)
  where T : IComparable<T> {
    if (hi - lo <= Threshold)
        InsertionSort(domain, lo, hi);
    int pivot = Partition(domain, lo, hi);
    Parallel.Invoke(
      delegate { ParQuickSort(domain, lo, pivot - 1); },
      delegate { ParQuickSort(domain, pivot + 1, hi); }
    );
}
```

Breaking out of a loop

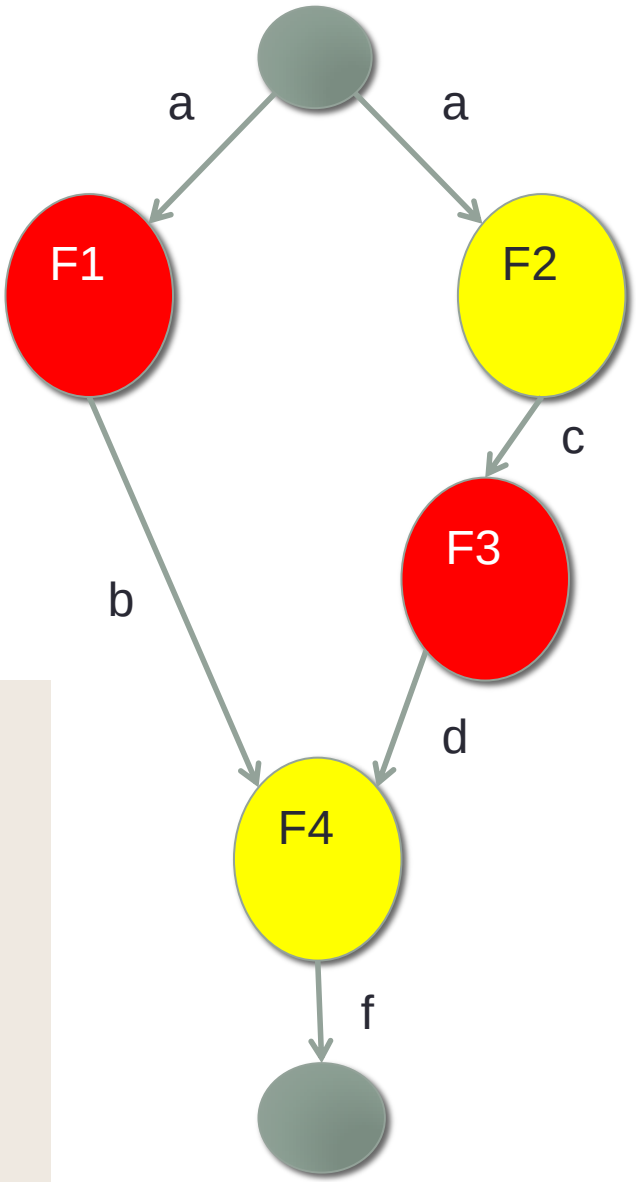
- ForEach has a parameter of type **ParallelLoopState**
- Methods are Break and Stop, with different semantics
- New in .NET 4.5 is timing out

```
ParallelLoopResult loopResult =
  Parallel.ForEach(studentList, student,
    ParallelLoopState loop) => {
    if (student.grade == 100) {
       loop.Stop(); return;
    }
    if (loop.IsStopped) return;
});
Console.WriteLine(
  "No-one achieved 100% is" + loopResult.IsCompleted);
```

Futures

- Futures are tasks that return results
- The classes are Task and TaskFactory
- Most of the synchronization is handled by TPL
- Met
- Hods for ContinueWhenAny etc

```
Task<int> futureB =
    Task.Factory.StartNew<int>(
       () => F1(a));
  int c = F2(a);
  int d = F3(c);
  int f = F4(futureB.Result, d);
return f;
```



PLINQ

- LINQ enables the querying of a variety of data sources in a type safe manner
- LINQ uses an advanced syntax with deferred execution
- PLINQ partitions the data source into segments to keep the cores busy
- Option to control the number of cores used

```
var source = Enumerable.Range(1, 40000);
// Opt-in to PLINQ with AsParallel
var specialNums = from num in source.AsParallel()
          .With DegreeOfParallelism(4)
    where Compute(num) > 0
    select num;
```

A collection of 17 special numbers

Uses of PLINQ

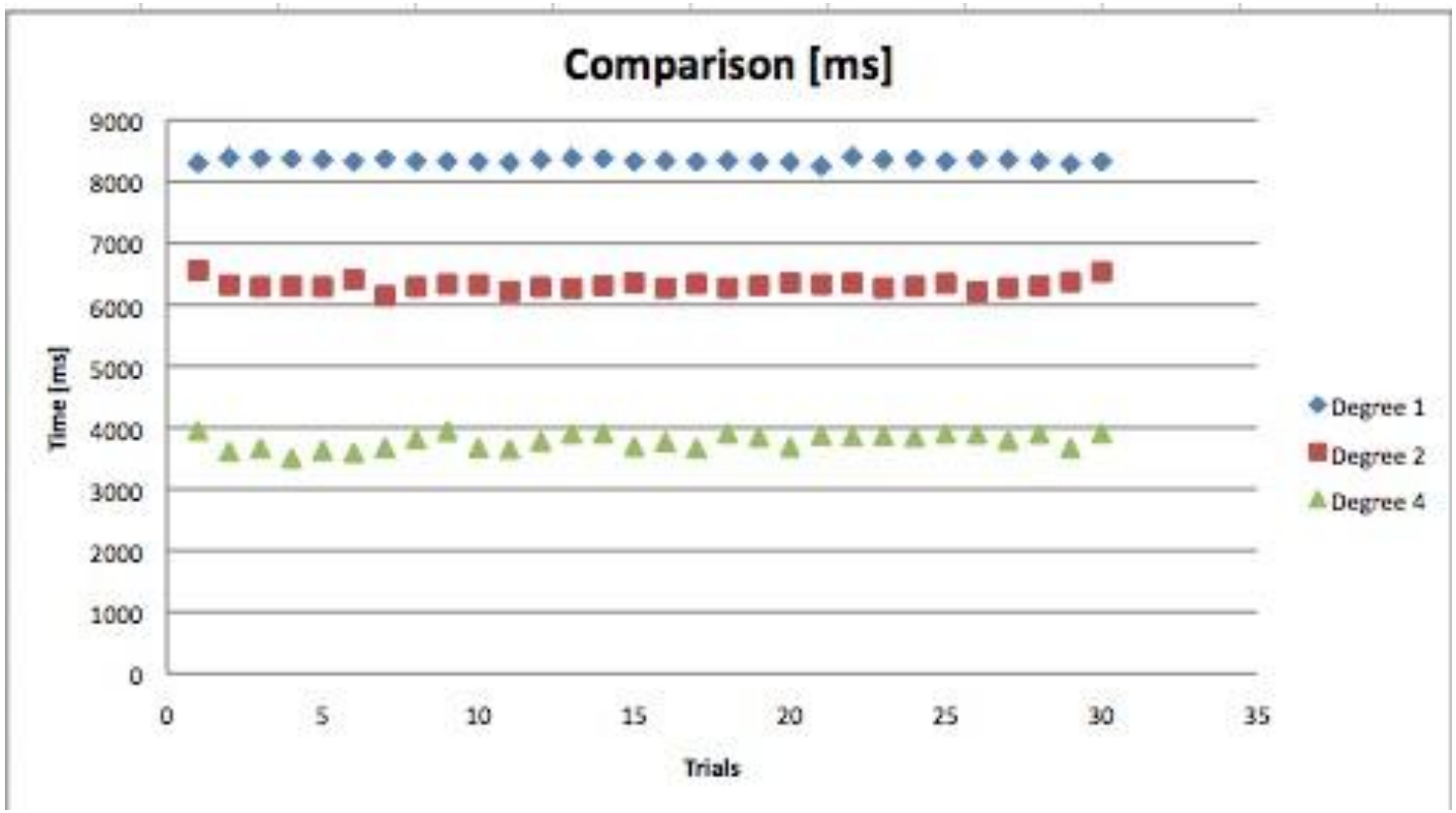
- Embarrassingly parallel applications
- Graphics and data processing
- Example – the Ray tracer

```
private IEnumerable<ISect>
Intersections
(Ray ray, Scene scene) {
  var things = from inter
     in obj.Intersect(ray)
     where inter != null
     orderby inter.Dist
     select obj;
  return things;
}
```

```
private IEnumerable<ISect>
Intersections
(Ray ray, Scene scene) {
  return scene.Things
  .Select(obj =>
      obj.Intersect(ray))
  .Where(inter =>
      inter != null)
  .OrderBy(inter =>
      inter.Dist);
}
```

Evaluation

Thanks to Luke Hoban and Luigi Drago



The machine was a 4x 6-core Intel Xeon CPU E7450 @ 2.4 GHz (total of 24 cores) with RAM: 96GB. The operating system was Windows Server 2008 SP2 64-bit (May 2009) running Visual Server 2010 with its C# compiler (April 2010).

Java Platform

- Lightweight threads since 1997
- java.util.concurrent package available since 2004
  - Includes thread pool, synchronizers, atomic variables
- JSR166y library in Java 7 in July 2011 contains a comprehensive parallel library
  - Fork, join and invokeAll methods
  - ForkJoinExecutor used for running tasks that only wait (not block)
  - Phasers that implement different types of barriers

Java example – Find Max Number

```
protected void compute() {
  if (problem.size < threshold)
     result = problem.solveSequentially();
  else {
     int midpoint = problem.size / 2;
     MaxWithFJ left = new MaxWithFJ(
       problem.subproblem(0, midpoint), threshold);
       MaxWithFJ right = new MaxWithFJ(
       problem.subproblem(midpoint + 1,
           problem.size), threshold);
       invokeAll(left, right);
       result = Math.max(left.result, right.result);
    }
```

How it all works

- A threadpool has to be explicitly created
- Java does not have delegates, so the FJ methods rely on specially named methods in a user class, e.g. compute

```
MaxWithFJ mfj = new MaxWithFJ(problem, threshold);
ForkJoinExecutor fjPool =
                    new ForkJoinPool(nThreads);
fjPool.invoke(mfj);
int result = mfj.result;
```

Speedup for FindMax on 500k array

	Threshold= 500k	Threshold= 50k	Threshold= 5k	Threshold= 500	Threshold=-50
Pentium-4 HT (2 threads)	1.0	1.07	1.02	.82	.2
Dual-Xeon HT (4 threads)	.88	3.02	3.2	2.22	.43
8-way Opteron (8 threads)	1.0	5.29	5.73	4.53	2.03
8-core Niagara (32 threads)	.98	10.46	17.21	15.34	

Good if lots of cheap operations OR few time-consuming operations

ParallelArray package

- Operations as methods similar to PLINQ
- Methods are passed objects that must contain an op method

```
System.out.println("Graduates this year:");
  processor
       .withFilter(isSenior)
       .withFilter(hasAcceptableGPA)
       .apply(printOutStudent);
  System.out.println("");
```

- The parallel array needs a forkjoin pool

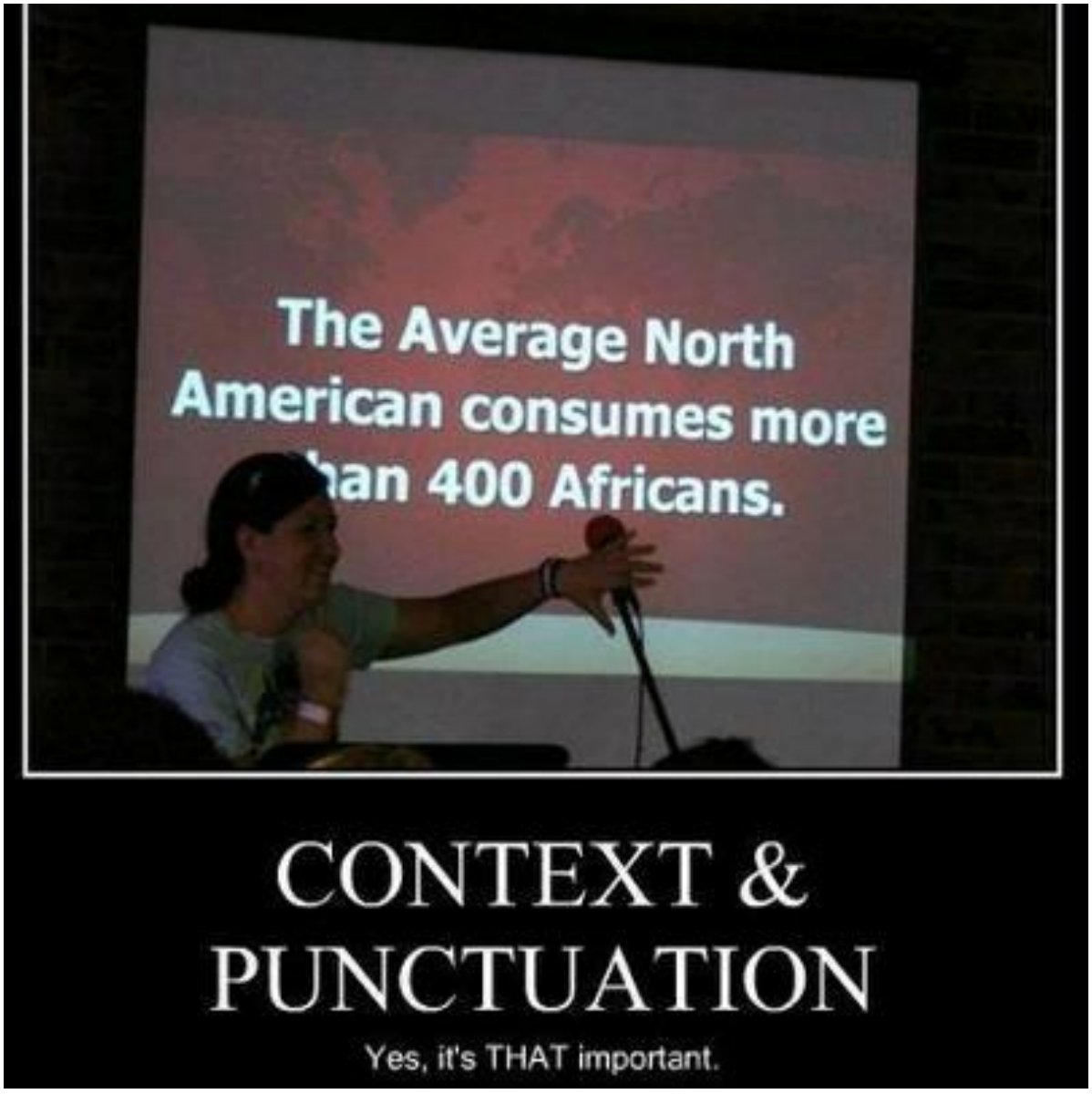
Java References

- Goetz, Brian, *Java theory and practice: Stick a fork in it*, http://www.ibm.com/developerworks/java/library/j-jtp11137.html, Nov 2007
- Holub, Allen, *Warning! Threading in a multiprocessor world*, JavaWorld, http://www.javaworld.com/jw-02-2001/jw-0209-toolbox.html, September 2001.
- http://download-llnw.oracle.com/javase/6/docs/api/java/util/concurrent/package-summary.html
- http://www.javac.info/jsr166z/jsr166z/forkjoin/ParallelArray.html
- Lea, Doug, *A java fork/join framework*. Java Grande, pp 36–43, 2000.
- Lea, Doug, *Concurrent programming in Java: design principles and patterns*, 2nd ed. Addison Wesley, 1999
- Lea, Doug, *The java.util.concurrent Synchronizer framework*, PODC Workshop on Concurrency and Synchronization in Java Programs, *CSJP'04*, July 26, 2004, St John's, Newfoundland, CA http://gee.cs.oswego.edu/dl/papers/aqs.pdf
- Neward, Ted, *Forking and Joining Java to Maximize Multicore Power*, DevX.com, http://www.devx.com/SpecialReports/Article/40982, February 2009

Language is not enough

- < 5% of developers familiar with the space
- Reputation for being difficult to get right
- Often don't understand the underlying issues
  - Sharing data
  - Blocking queues and messaging

How do we help developers be successful?

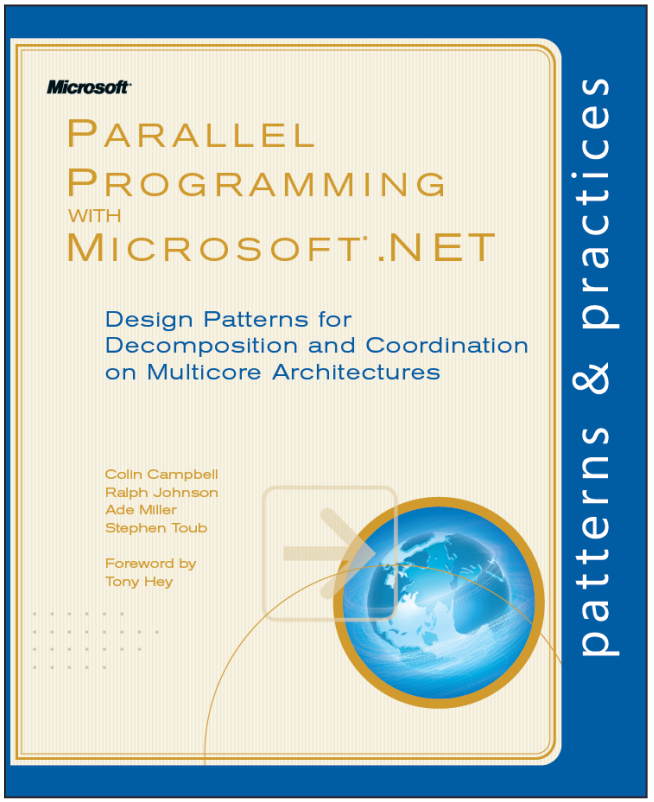

The Average North American consumes more
han 400 Africans.
CONTEXT & PUNCTUATION
Yes, it's THAT important.

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



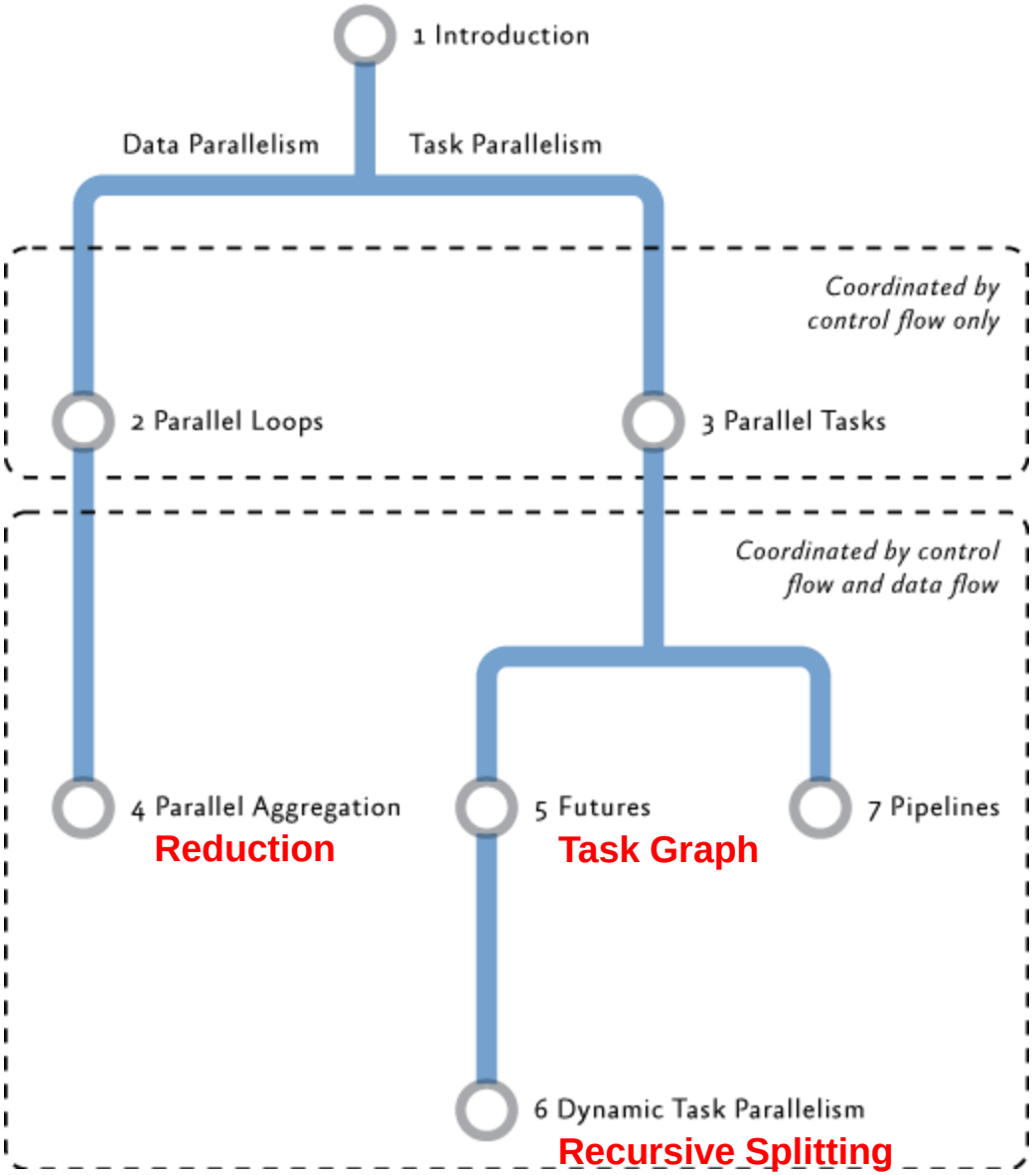
The Patterns

Picked 6 key patterns…

- Taken from OPL
- Most Common
- Supported by TPL

OPL is Berkeley's "Our Pattern Language"

http://parallelpatterns.codeplex.com
parlab.eecs.berkeley.edu/wiki/_media/patterns/opl_pattern_language-feb-13.pdf



Parallel Loops

```
Parallel.ForEach(accounts.AllAccounts, account => {
  Trend trend = SampleUtilities.Fit(account.Balance);
  double prediction = trend.Predict(
                        account.Balance.Length +
NumberOfMonths);
  account.ParPrediction = prediction;
  account.ParWarning = prediction < account.Overdraft;
});
```

```
accounts.AllAccounts.AsParallel().ForAll(account => {
  Trend trend = SampleUtilities.Fit(account.Balance);
  double prediction = trend.Predict(
              account.Balance.Length + NumberOfMonths);
  account.PlinqPrediction = prediction;
  account.PlinqWarning = prediction < account.Overd
});
```

PLINQ

Parallel Aggregation (map-reduce)

```
object lockObject = new object();
double sum = 0.0d;
Parallel.ForEach(sequence, () => 0.0d,  // Initialize
    (x, loopState, partialResult) =>
    {                                    // Sum locals
       return Normalize(x) + partialResult;
    },
                                         // In parallel
    (localPartialSum) =>  {             // Sum partials
       lock(lockObject) {
         sum += localPartialSum;
       }
    });
```

Parallel Aggregation

PLINQ

```
from x in sequence.AsParallel()
     select Normalize(x))
     .Aggregate(0.0d, (y1, y2) => y1 + y2);
```

```
vector<double> sequence;
combinable<double> sum([]()->double { return 0.0; });
parallel_for_each(sequence.begin(),sequence.end(),
[&sum](double x)
{
  sum.local() += Normalize(x);
});
return sum.combine(plus<double>());
```

C++/PPL

Map-reduce in PLINQ again

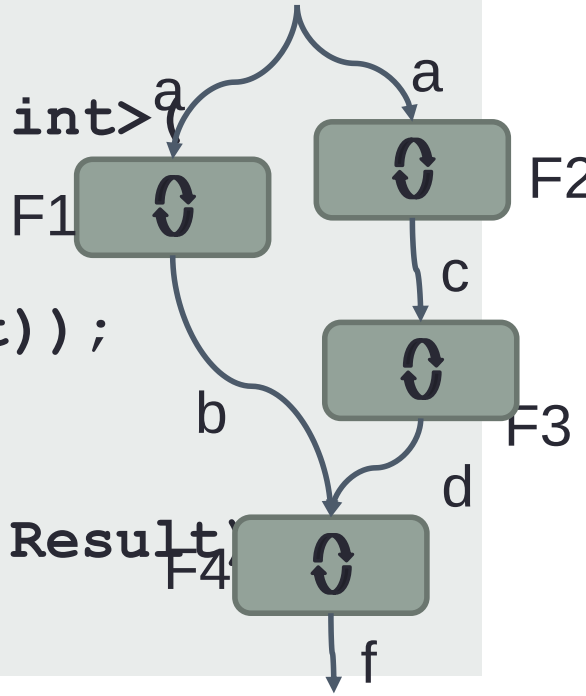
```
public IDMultisetItemList
PotentialFriendsPLinq(SubscriberID id, int maxCandidates)
{
  var candidates =
    subscribers[id].Friends.AsParallel()
    .SelectMany(friend => subscribers[friend].Friends)
    .Where(foaf => foaf != id &&
           !(subscribers[id].Friends.Contains(foaf)))
    .GroupBy(foaf => foaf)
    .Select(foafGroup => new IDMultisetItem(
            foafGroup.Key,foafGroup.Count()));
  return Multiset.MostNumerous(candidates,maxCandidates);
}
```

Parallel Tasks

```
Task t1 = Task.Factory.StartNew(DoLeft);
Task t2 = Task.Factory.StartNew(DoRight);

Task.WaitAll(t1, t2);

Parallel.Invoke(
    () => DoLeft(),
    () => DoLeft()
);
```

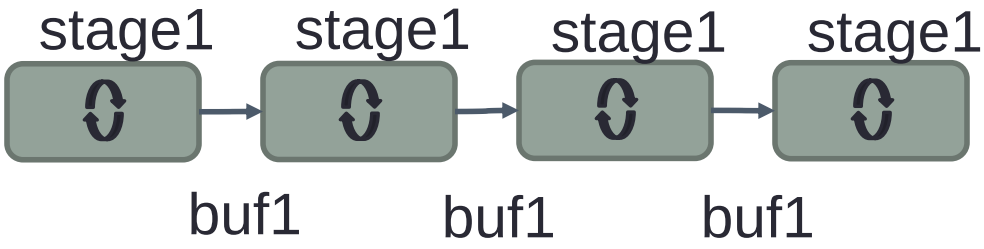
Futures

```
var futureB = Task.Factory.
              StartNew<int>(() => F1(a));
var futureD = Task.Factory.
              StartNew<int>(() => F3(F2(a)));

var futureF = Task.Factory.
              ContinueWhenAll<int, int>(
  new[] { futureB, futureD },
  (tasks) =>
  F4(futureB.Result, futureD.Result));

futureF.ContinueWith((t) =>
  Console.WriteLine("Result: " + t.Result)
);
```



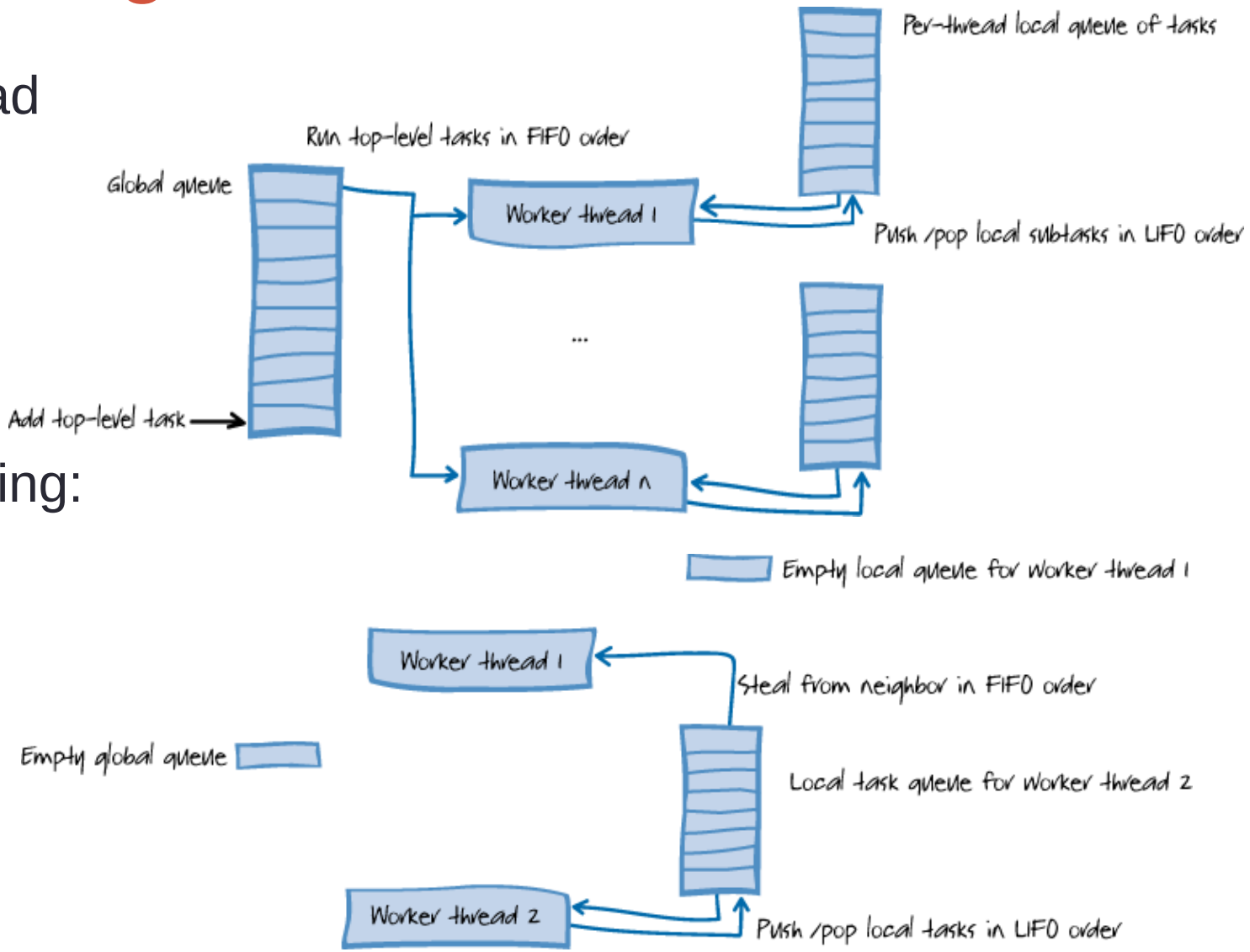
Pipelines

```
var buf1 = new BlockingCollection<string>(BufferSize);
var buf2 = new BlockingCollection<string>(BufferSize);
var buf3 = new BlockingCollection<string>(BufferSize);

var stage1 = Task.Factory.StartNew(() => ReadStrings(buf1, ...));
var stage2 = Task.Factory.StartNew(() => CorrectCase(buf1, buf2));
var stage3 = Task.Factory.StartNew(() => CreateSentences(buf2, buf3));
var stage4 = Task.Factory.StartNew(() => WriteSentences(buf3));

Task.WaitAll(stage1, stage2, stage3, stage4);

void CorrectCase(BlockingCollection<T> input,
                 BlockingCollection<T> output)
foreach (var item in input.GetConsumingEnumerable())
  {
    // …
    output.Add(result);
  }
  output.CompleteAdding();
}
```



Scheduling

Local thread queues:



Work stealing:

Conclusions

- Using higher level structures and libraries makes for
  - Ease of programming
  - Without sacrificing speedup
- The .NET TPL is the most advanced library todate and is being constantly improved behind the scenes
- .NET 5 has some new features too
- Thank you! Questions?