Practical
Parallel and Concurrent Programming

Course Overview

http://ppcp.codeplex.com/

These Course Materials Brought to You By

- Microsoft Research (MSR)
  - Research in Software Engineering (RiSE)
- University of Utah
  - Computer Science
- With support from
  - MSR External Research (Judith Bishop)
  - Microsoft Parallel Computing Platform (Stephen Toub, Sherif Mahmoud, Chris Dern)

Courseware Authors

- Thomas Ball, MSR Redmond
- Sebastian Burckhardt, MSR Redmond
- Ganesh Gopalakrishnan, Univ. Utah
- Joseph Mayo, Univ. Utah
- Madan Musuvathi, MSR Redmond
- Shaz Qadeer, MSR Redmond
- Caitlin Sadowski, Univ. California Santa Cruz

Acknowledgments

- This slide deck contains material courtesy of
  - Tim Harris, MSR Cambridge
  - Burton Smith, MSR Redmond

- The headshot of the alpaca used throughout the lectures is licensed under
  - the Creative Commons Attribution-Share Alike 2.0 Generic license
  - http://en.wikipedia.org/wiki/File:Alpaca_headshot.jpg

Overview

- **Context**
  - Trends
  - Applications
  - System and environment
- Concepts
- Units, Materials and Tools

Technology Trends

- Increasing *parallelism* in a "computer"
  - multi-core CPU
  - graphical processing unit (GPU)
  - cloud computing

- Increasing *disk capacity*
  - we are awash in interesting *data*
  - data-intensive problems require *parallel processing*

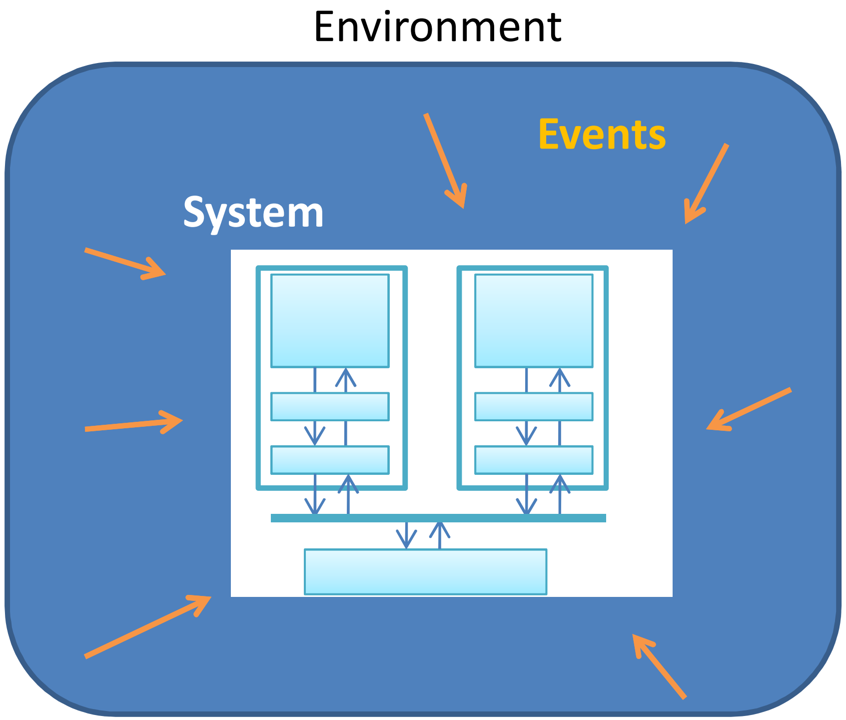
Technology Trends (2)

- Increasing *networks and network bandwidth*
  - wireless, wimax, 3G, …
  - collection/delivery of massive datasets, plus
  - real-time responsiveness to asynchronous events

- Increasing *number and variety of computers*
  - smaller and smaller, and cheaper to build
  - generating streams of asynchronous events

Parallelism and Concurrrency: System and Environment

- *Parallelism*: exploit system resources to speed up computation
- *Concurrency*: respond quickly/properly to events
  – from the environment
  – from other parts of system



Application Areas

- Entertainment/games
- Finance
- Science
- Modeling of real-world
- Health care
- Telecommunication
- Data processing
- …

Discuss application areas in context of

Trends

Parallelism/Concurrency

System/Environment

Practical Parallel and Concurrent Programming (PP&CP)

	Parallelism	Concurrency
Performance	Speedup	Responsiveness
Correctness	Atomicity, Determinism, Deadlock, Livelock, Linearizability, Data races, …	

Overview

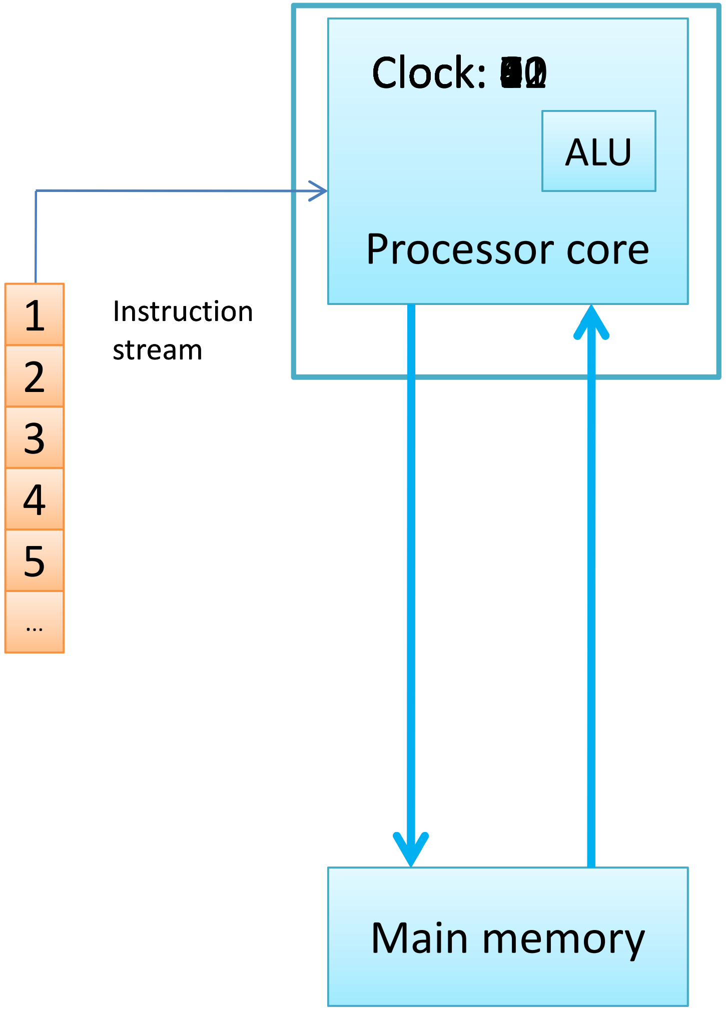
- Context
- **Concepts**
  1. Multi-core computer
  2. Speedup
  3. Responsiveness
  4. Correctness
- Units, Materials and Tools

Concept #1:
System = Multi-core Hardware

What is Today's Multi-core?

- What is the architecture?
- What are its properties?
  - Computation
  - Communication
    - Delivery guarantees
    - Latency
    - Throughput
  - Consistency
  - Caching

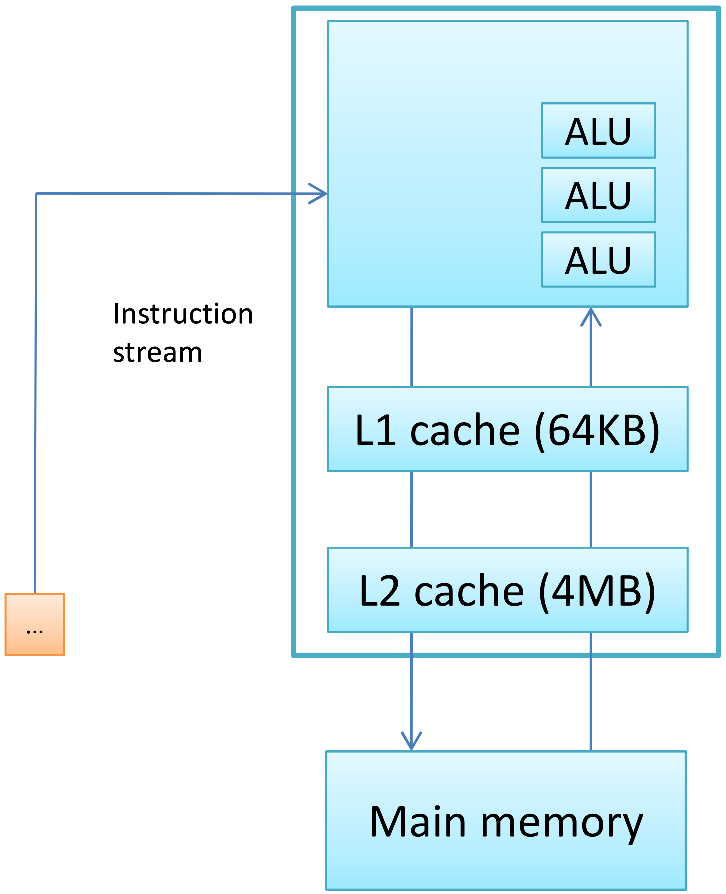
A simple microprocessor model ~ 1985



- Single h/w thread
- Instructions execute one after the other
- Memory access time ~ clock cycle time

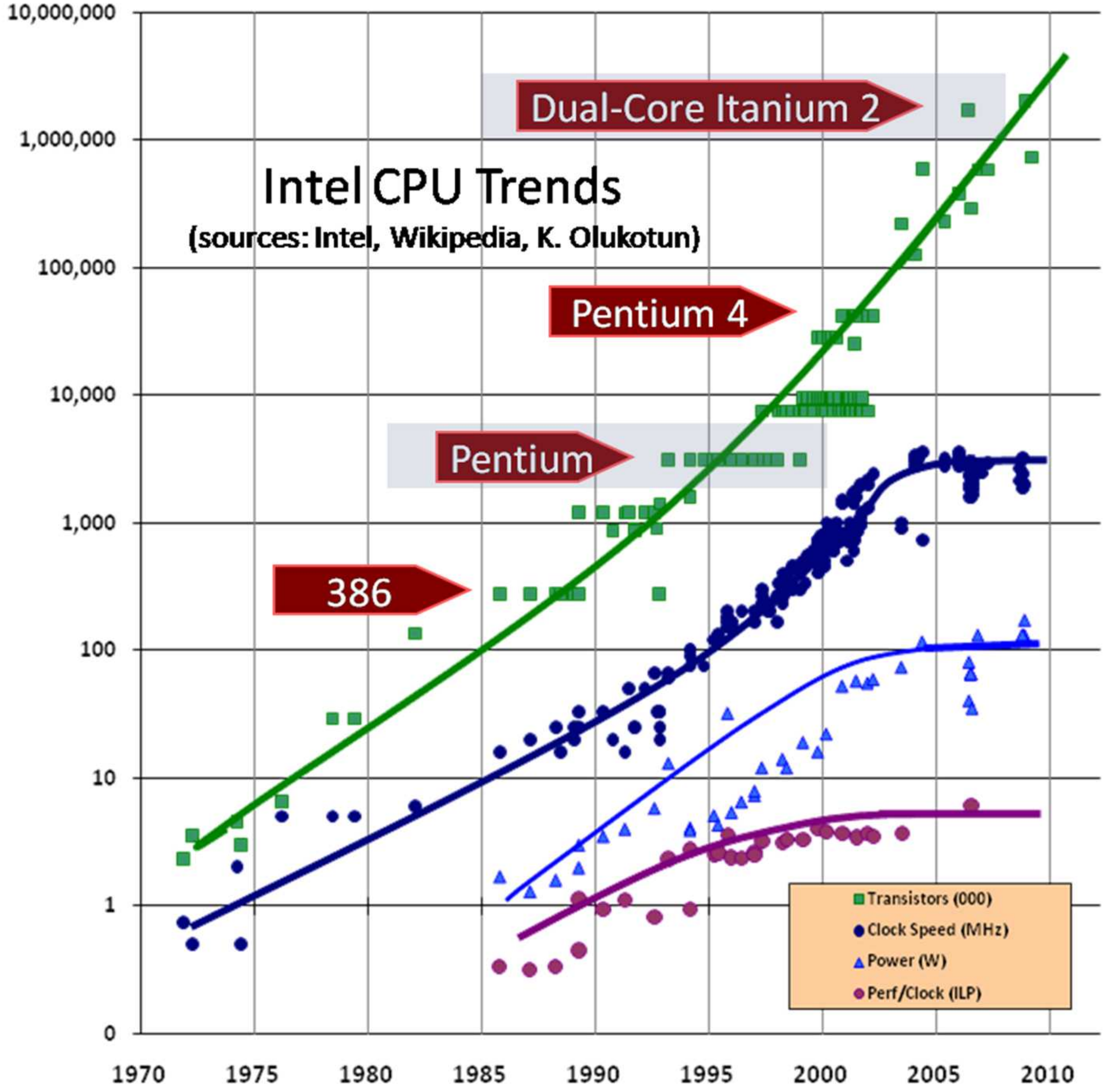
ALU: arithmetic logic unit

FastFwd Two Decades (circa 2005): Power Hungry Superscalar with Caches



Multiple levels of cache, 2 cycles for L1, 20 cycles for L2, 200 cycles for memory

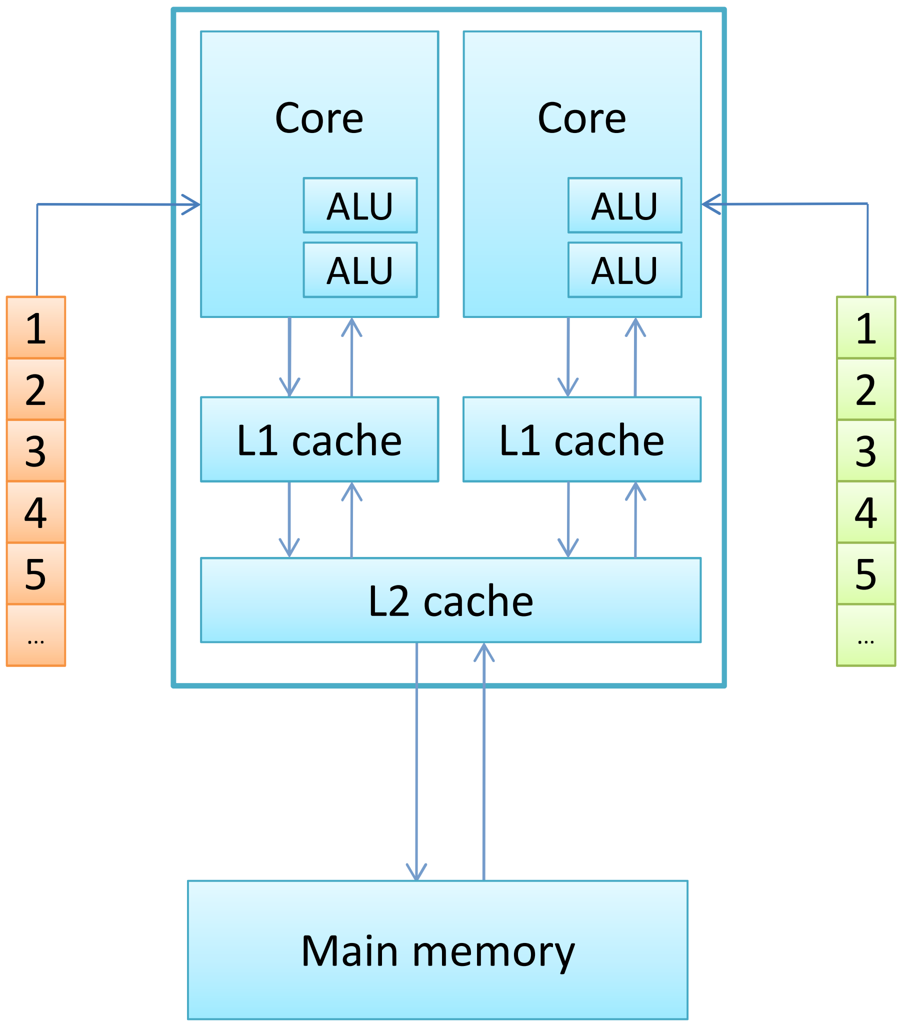
- Dynamic out-of-order
- Pipelined memory accesses
- Speculation


Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Dual-Core Itanium 2
Pentium 4
Pentium
386
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
1970
1975
1980
1985
1990
1995
2000
2005
2010
Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (ILP)

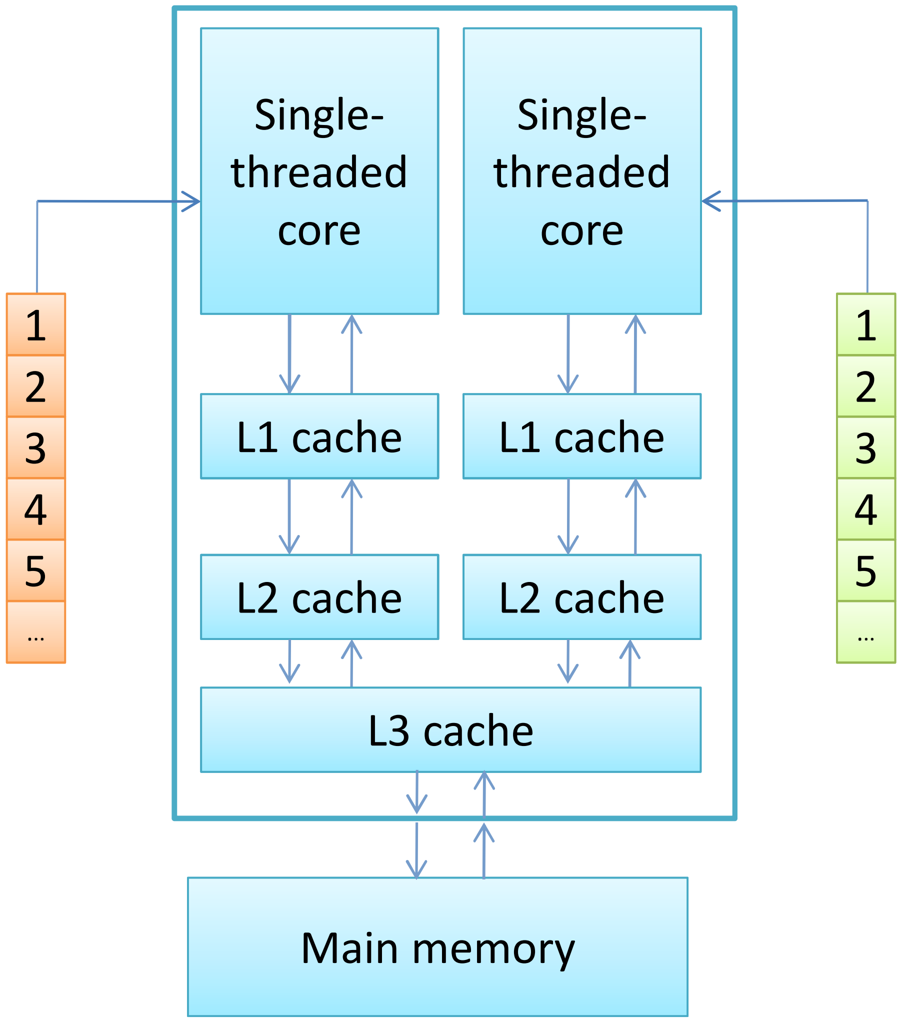
Power wall + ILP wall + memory wall = **BRICK WALL**

- *Power wall*
  - we can't clock processors faster
- *Memory wall*
  - many workload's performance is dominated by memory access times
- *Instruction-level Parallelism (ILP) wall*
  - we can't find extra work to keep functional units busy while waiting for memory accesses

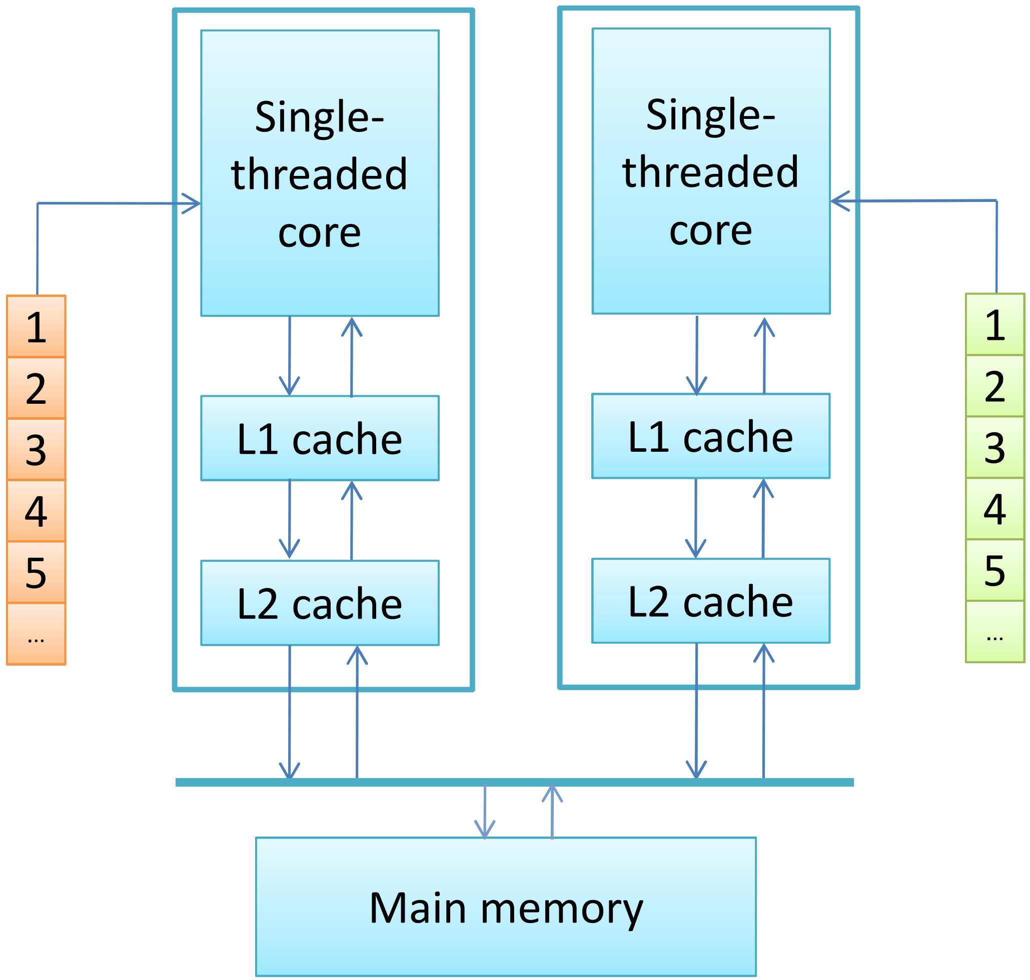
Multi-core h/w – common L2



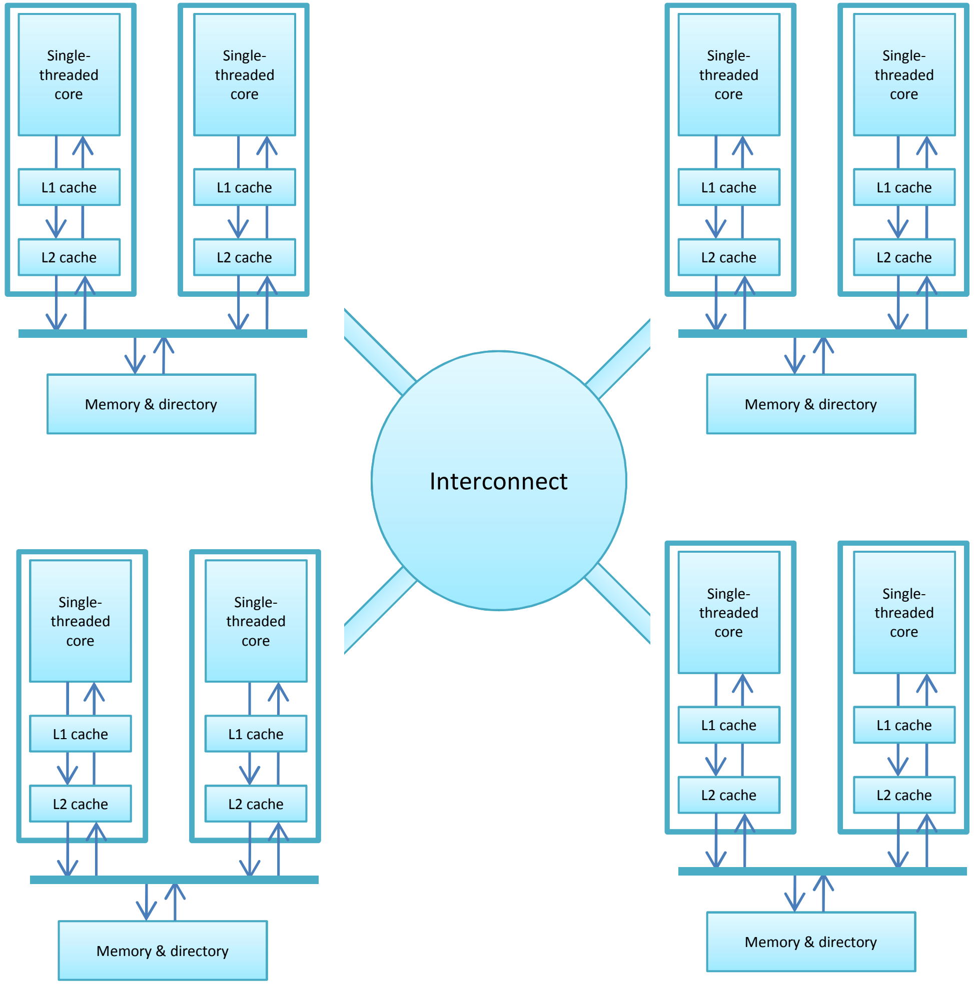
Multi-core h/w – additional L3



SMP multiprocessor



NUMA multiprocessor



Three kinds of parallel hardware

- Multi-threaded cores
  - Increase utilization of a core or memory b/w
  - Peak ops/cycle fixed
- **Multiple cores**
  - Increase ops/cycle
  - Don't necessarily scale caches and off-chip resources proportionately
- Multi-processor machines
  - Increase ops/cycle
  - Often scale cache & memory capacities and b/w proportionately

Concept #2: Speedup

Speedup Concerns

1. *Focus* on the longest running parts of the program first
   - be realistic about possible speedups
   - *different parts* may need to be parallelised with *different techniques*

2. *Understand* the different resource requirements of a program
   - computation, communication, and locality

3. *Consider* how data accesses interact with the memory system:
   - will the computation done on additional cores pay for the data to be brought to them?

Abstractions for Speedup

- Imperative parallelism
  - Parallel.For/ForEach
  - Lightweight *tasks* (not threads)

- Functional parallelism
  - Functional programming (F#)
  - Parallel Language Integrated Queries (PLINQ)
  - Array parallel algorithms (Accelerator)

- Concurrent components
  - for example, data structures that can efficiently accommodate many clients

Concept #3: Responsiveness

Responsiveness Concerns

1. Quick reaction to conditions over event streams

2. Handle multiple tasks at the same time

3. Don’t block essential tasks unnecessarily

4. Coordinate responses to requests

Abstractions for Responsiveness

- Asynchronous computation
  - lightweight *tasks* (not threads)
  - F#'s async
- Application-specific scheduling
- Complex event handling
  - IObservable
  - Reactive extensions (RX) to .NET
- Actors/agents

Concept #4:
Correctness

Correctness Concerns

- All those we have for sequential code
  - Assertions, invariants, contracts,
  - buffer overflows, null reference,
  - …
- Plus those related to parallelism/concurrency
  - Data races, deadlocks, livelocks, …
  - Memory coherence/consistency

Correctness Abstractions

- Atomicity
- Determinism
- Linearizability
- Serializibility
- Temporal logic

Outline

- Context
- Concepts
- **Units, Materials and Tools**

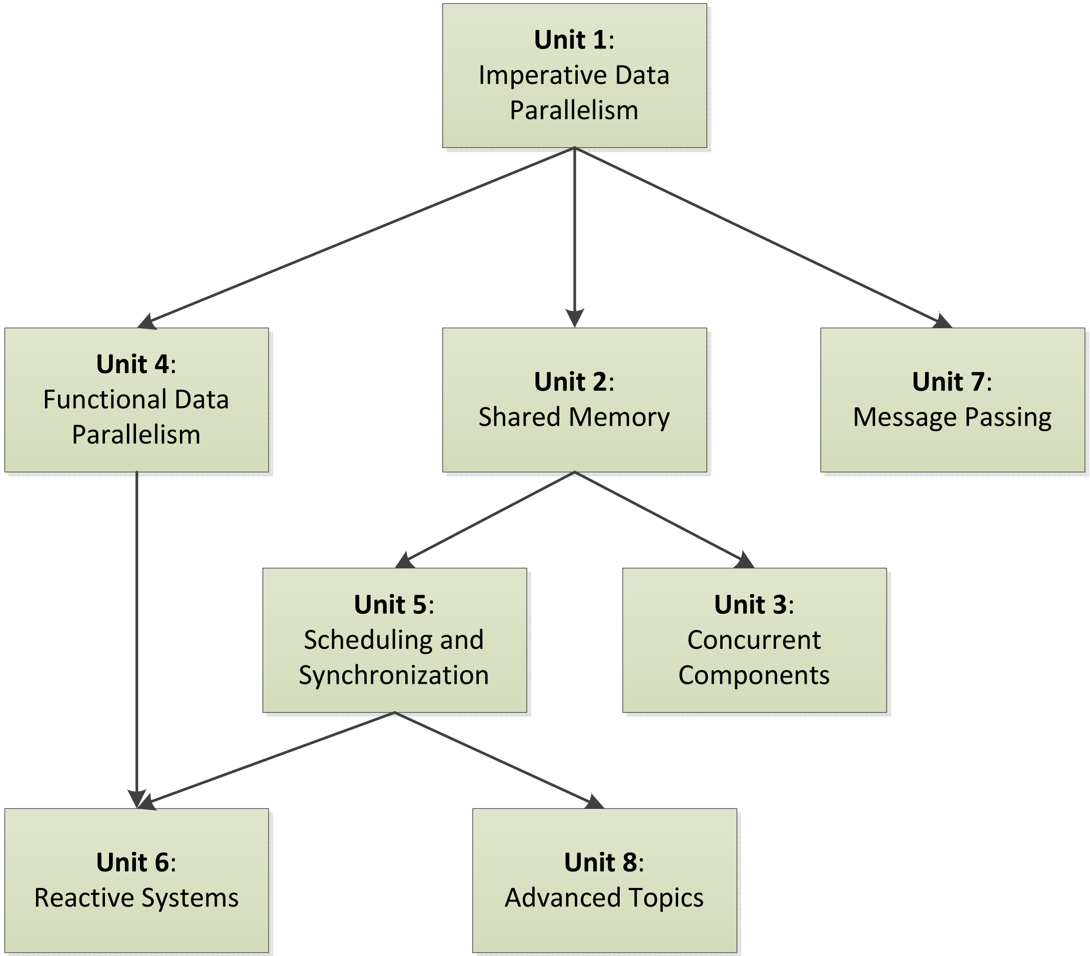
Units 1 – 4

- Unit 1: Imperative Data Parallelism
  - Data-intensive parallel programming (Parallel.For)
  - Concurrent Programming with Tasks
- Unit 2: Shared Memory
  - Data Races and Locks
  - Parallel Patterns
- Unit 3: Concurrent Components
  - Thread-Safety Concepts (Atomicity, Linearizability)
  - Modularity (Specification vs. Implementation)
- Unit 4: Functional Data Parallelism
  - Parallel Queries with PLINQ
  - Functional Parallel Programming with F#

Units 5 – 8

- Unit 5: Scheduling and Synchronization
  - From {tasks, DAGs} to {threads, processors}
  - Work-stealing
- Unit 6: Interactive/Reactive Systems
  - External vs. internal concurrency
  - Event-based programming
- Unit 7: Message Passing
  - Conventional MPI-style programming
- Unit 8: Advanced Topics
  - Parallelization, Transactions, Revisions

Unit Dependences



IDE, Libraries, Tools, Samples, Book

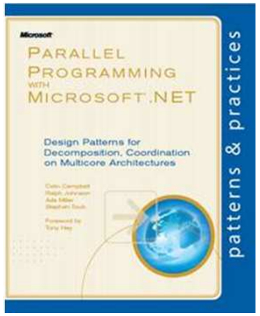
- **Visual Studio 2010**
  - C# and F# languages
  - **.NET 4:** Libraries for multi-core parallelism and concurrency
- **Other Libraries**
  - Accelerator
  - Code Contracts
  - Rx: Reactive Extensions for .NET
- **Alpaca**
  - A lovely parallelism and concurrency analyzer
  - Source code
- Code for all units, with Alpaca tests
- **Parallel Extensions Samples**
- **Free book: Parallel Programming with Microsoft .NET**

Icon Guide

Performance Concept

Correctness Concept

Code Concept

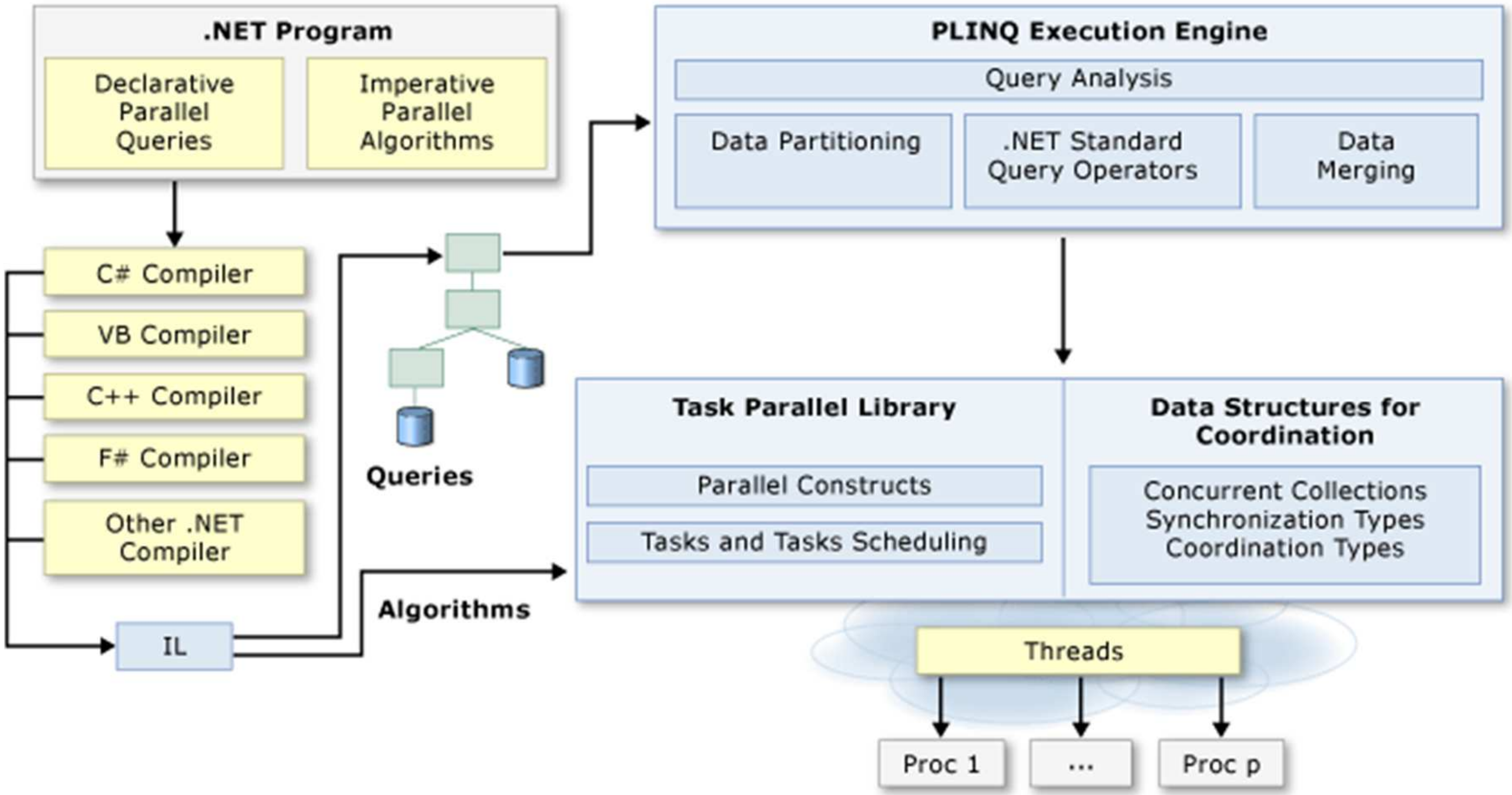


Discuss

Run

Aside

.NET 4 Libraries for Parallelism and Concurrency



Alpaca: A lovely parallelism and concurrency analyzer

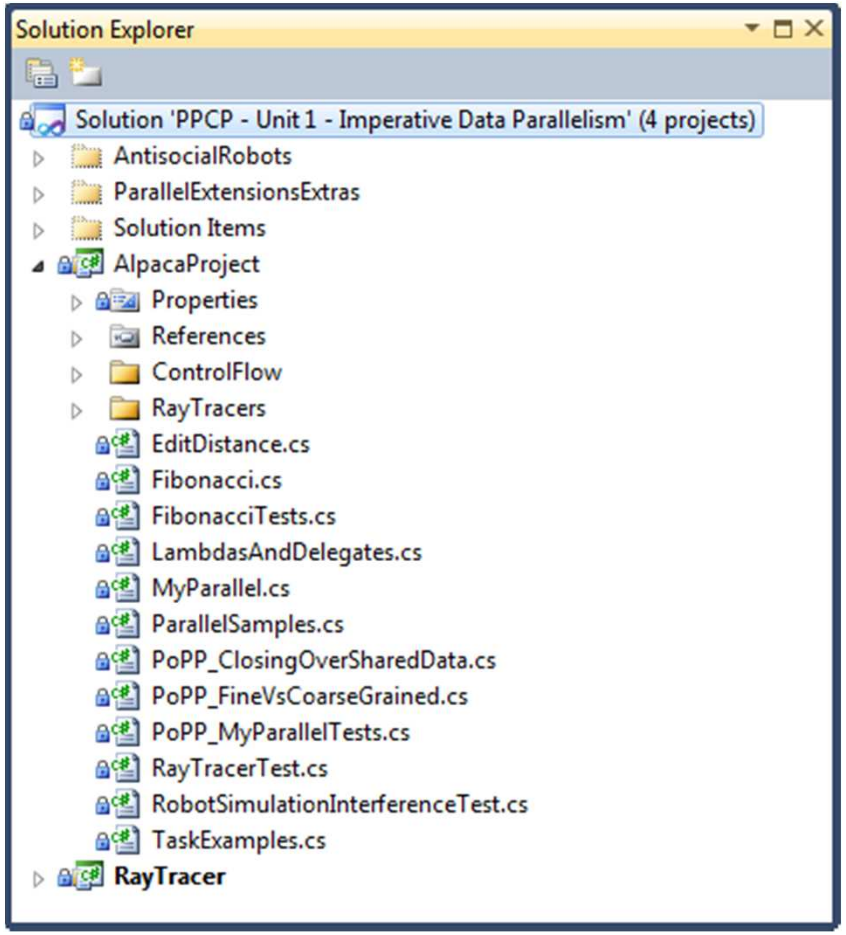
- Atttribute-based testing, for performance and correctness concepts
- [`UnitTestMethod`]
  - simply run this method normally, and report failed assertions or uncaught exceptions.
- [`DataRaceTestMethod`]
  - Run a few schedules (using CHESS tool) and detect data races.
- [`ScheduleTestMethod`]
  - Run all possible schedules of this method (with at most two preemptions) using the CHESS tool.
- [`PerformanceTestMethod`]
  - Like UnitTestMethod, but collect & graphically display execution timeline (showing intervals of interest)

Why Alpaca?

- Improve the learning experience for concurrent and parallel programming
- Vehicle for including instantly runnable sample code (incl. bugs)
- Unit tests: A quick way to validate / invalidate assumptions, about correctness or performance
- Provide simple graphical front end for various tools

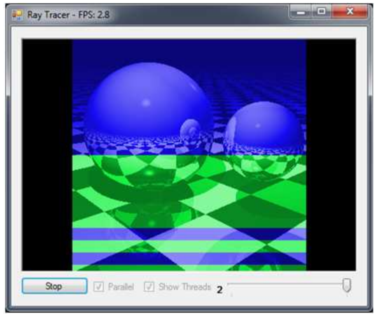
PPCP – Unit X - *.sln

- Each Unit has a VS2010 Solution
  - supporting examples
  - Alpaca Project

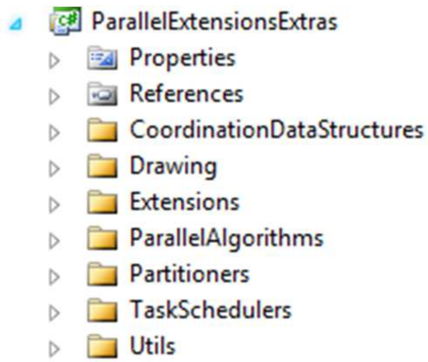


Parallel Extensions Samples

- http://code.msdn.microsoft.com/ParExtSamples

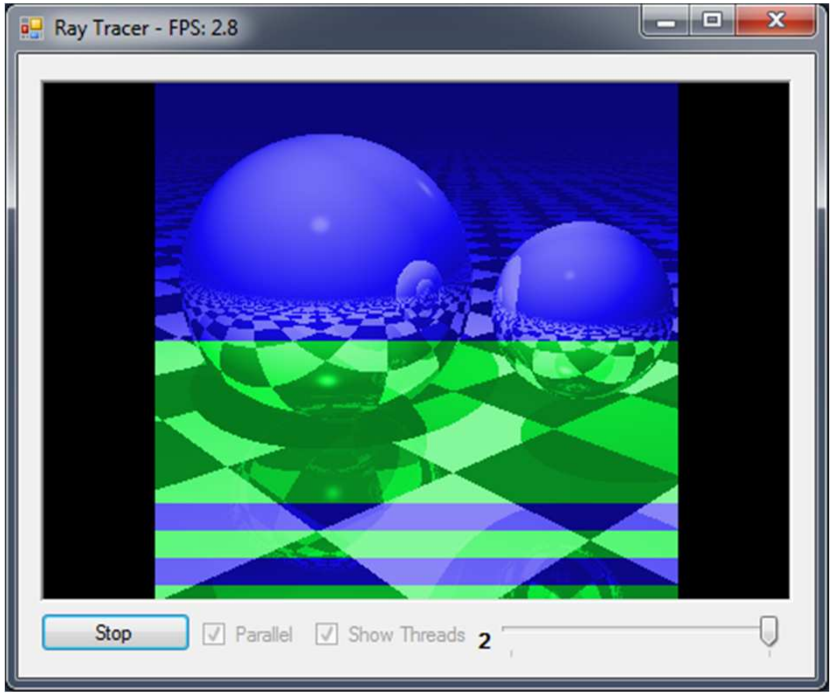


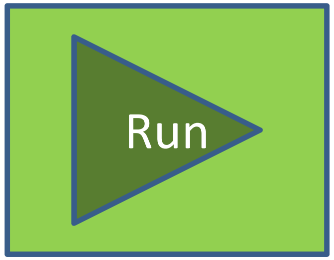
- **Over 15 Samples**
  - applications illustrating use of .NET 4
  - some included in courseware



- **ParallelExtensionsExtras.csproj**
  - helper classes built on .NET 4

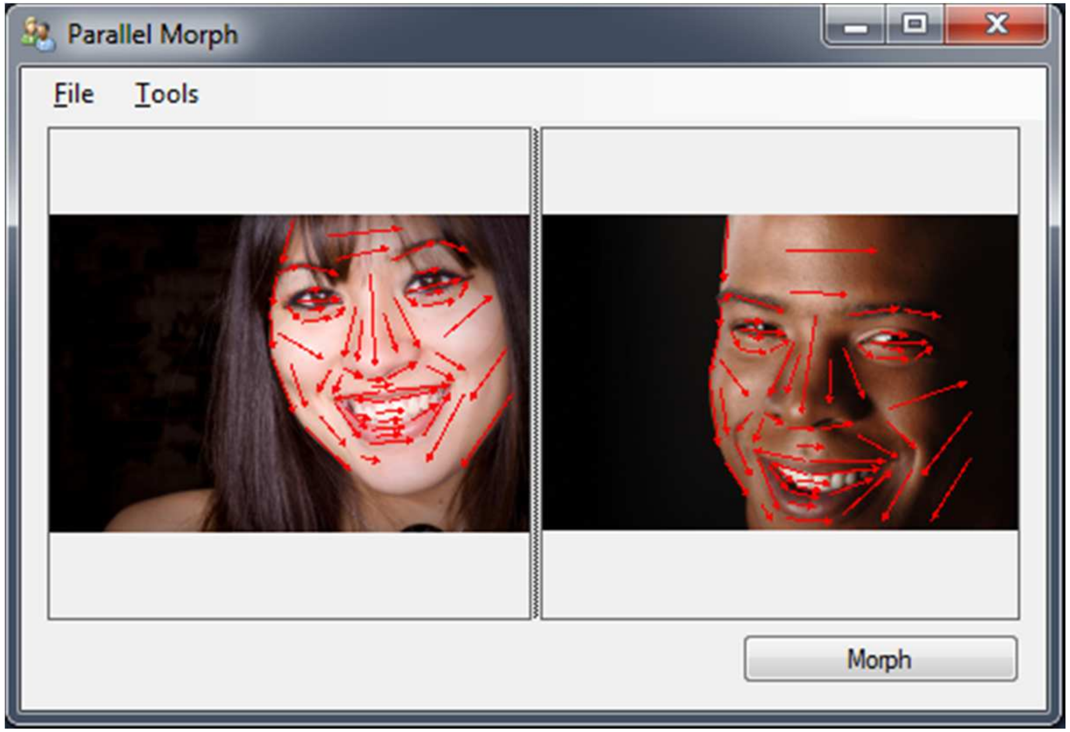
Sample: Ray Tracer

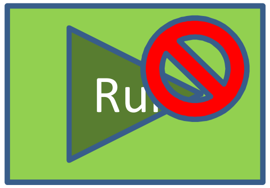




Animated, ray traced bouncing balls. Sequential and parallel implementations are provided, as is a special parallel implementation that colors the animated image based on which thread was used to calculate which regions.

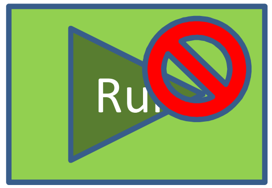
Sample: Image Morphing





Implements a morphing algorithm between two images. Parallelization is done using the Parallel class.

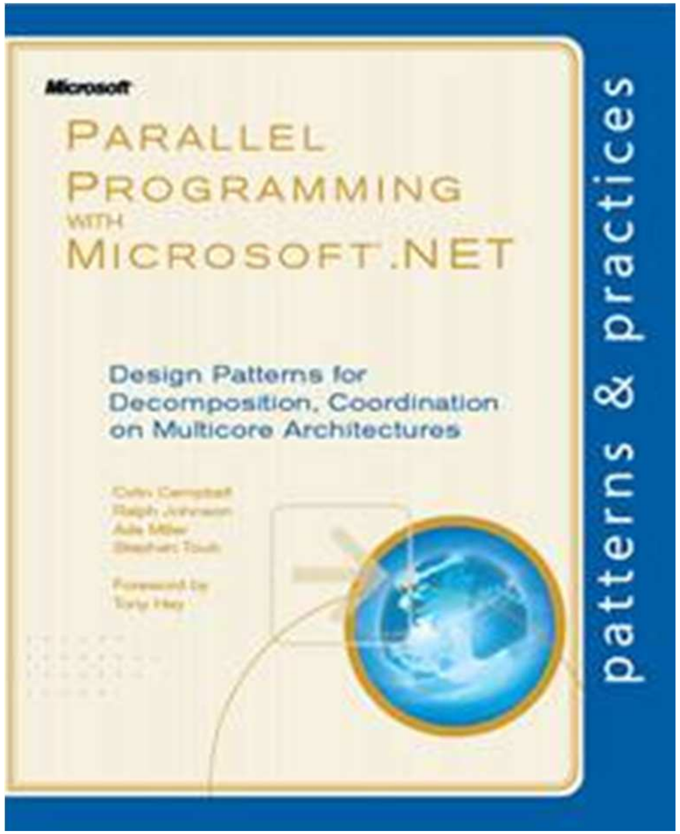
Sample: N-Body Simulation



Implements a classic n-body simulation using C# and WPF for the UI and using F# for the core computation. Parallelism is achieved using the Parallel class.

Free book:

Parallel Programming with Microsoft .NET



Design Patterns for Decomposition and Coordination on Multicore Architectures

Colin Campbell, Ralph Johnson, Ade Miller and Stephen Toub