Multicore Programming Primer
and Programming Competition

Introduction
“To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

-- E. Dijkstra, 1972 Turing Award Lecture
The First Software Crisis

- Time Frame: ’60s and ’70s

- Problem: Assembly Language Programming
  - Computers could handle larger more complex programs

- Needed to get Abstraction and Portability without losing Performance
How Did We Solve the First Software Crisis?

- High-level languages for von-Neumann machines
  - FORTRAN and C
- Provided “common machine language” for uniprocessors

<table>
<thead>
<tr>
<th>Common Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single flow of control</td>
</tr>
<tr>
<td>Single memory image</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Differences:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register File</td>
</tr>
<tr>
<td>ISA</td>
</tr>
<tr>
<td>Functional Units</td>
</tr>
</tbody>
</table>
The Second Software Crisis

- Time Frame: ’80s and ’90s

- Problem: Inability to build and maintain complex and robust applications requiring multi-million lines of code developed by hundreds of programmers
  - Computers could handle larger more complex programs

- Needed to get Composability, Malleability and Maintainability
  - High-performance was not an issue → left for Moore’s Law
How Did We Solve the Second Software Crisis?

● Object Oriented Programming
  ■ C++, C# and Java

● Also…
  ■ Better tools
    – Component libraries, Purify
  ■ Better software engineering methodology
    – Design patterns, specification, testing, code reviews
Today:
Programmers are Oblivious to Processors

● Solid boundary between Hardware and Software

● Programmers don’t have to know anything about the processor
  ■ High level languages abstract away the processors
    – Ex: Java bytecode is machine independent
  ■ Moore’s law does not require the programmers to know anything about the processors to get good speedups

● Programs are oblivious of the processor → work on all processors
  ■ A program written in ’70 using C still works and is much faster today

● This abstraction provides a lot of freedom for the programmers
The Origins of a Third Crisis

- Time Frame: 2005 to 20??
- Problem: Sequential performance is left behind by Moore’s law
- Needed continuous and reasonable performance improvements
  - to support new features
  - to support larger datasets
- While sustaining portability, malleability and maintainability without unduly increasing complexity faced by the programmer

→ critical to keep-up with the current rate of evolution in software
The March to Multicore: Moore’s Law

Image removed due to copyright restrictions.
General-purpose unicores have stopped historic performance scaling

- Power consumption
- Wire delays
- DRAM access latency
- Diminishing returns of more instruction-level parallelism
Power Consumption (watts)
Power Efficiency (watts/spec)


Watts/Spec:
- 0.7
- 0.6
- 0.5
- 0.4
- 0.3
- 0.2
- 0.1
- 0.0

Prof. Saman Amarasinghe, MIT.
Range of a Wire in One Clock Cycle

- 400 mm$^2$ Die
- From the SIA Roadmap

Prof. Saman Amarasinghe, MIT.
DRAM Access Latency

- Access times are a speed of light issue
- Memory technology is also changing
  - SRAM are getting harder to scale
  - DRAM is no longer cheapest cost/bit
- Power efficiency is an issue here as well

Images removed due to copyright restrictions.

\[
\begin{align*}
\muProc & \quad 60\%/yr. \\
(2X/1.5yr) & \\
DRAM & \quad 9\%/yr. \\
(2X/10\,\text{yrs}) &
\end{align*}
\]
Diminishing Returns

- **The ’80s: Superscalar expansion**
  - 50% per year improvement in performance
  - Transistors applied to *implicit* parallelism
    - pipeline processor (10 CPI --> 1 CPI)

- **The ’90s: The Era of Diminishing Returns**
  - Squeaking out the last implicit parallelism
    - 2-way to 6-way issue, out-of-order issue, branch prediction
    - 1 CPI --> 0.5 CPI
  - performance below expectations
  - projects delayed & canceled

- **The ’00s: The Beginning of the Multicore Era**
  - The need for Explicit Parallelism
Unicores are on the verge of extinction
Multicores are here

MIT Raw
16 Cores
Since 2002

Intel Montecito
1.7 Billion transistors
Dual Core IA/64

Intel Tanglewood
Dual Core IA/64

Intel Pentium D
(Smithfield)

Intel Dempsey
Dual Core Xeon

Intel Pentium Extreme
3.2GHz Dual Core

Intel Yonah
Dual Core Mobile

AMD Opteron
Dual Core

Sun Olympus and Niagara
8 Processor Cores

IBM Cell
Scalable Multicore

IBM Power 4 and 5
Dual Cores Since 2001

IBM Power 6
Dual Core

Intel Tejas & Jayhawk
Unicore (4GHz P4)

Cancelled

Prof. Saman Amarasinghe, MIT.
Multicores are Here

# of cores


1 2 4 8 16 32 64 128 256 512

4004 8080 8086 286 386 486 Pentium P2 P3 P4 Itanium Yonah Cell Xeon MP Tanglewood Power6 Power4 Xbox360 PA-8800 Opteron Tanglewood Raw Niagara XLR Octeon Cavium Octeon Raza XLR Tiflops Intel Tflops Cisco CSR-1 Picochip PC102 Ambric AM2045

Prof. Saman Amarasinghe, MIT.
Requirements and Outcomes

● Requirements
  ■ A good programmer with experience
  ■ Fluent in C

● Outcomes
  ■ Know fundamental concepts of parallel programming (both hardware and software)
  ■ Understand issues of parallel performance
  ■ Able to synthesize a fairly complex parallel program
  ■ Hands-on experience with the IBM Cell processor
The Project

- You proposed the projects
- We selected 7 teams
  - Mainly by the strength of the project proposals
- Seven Great Projects
  - Distributed Real-time Ray Tracer
  - Global Illumination
  - Linear Algebra Pack
  - Molecular Dynamics Simulator
  - Speech Synthesizer
  - Soft Radio
  - Backgammon Tutor
- Project Characteristics
  - Ambitious but accomplishable
  - Important and Relevant
  - Opportunity to sizzle
- Get them started ASAP!
A Note of Caution

- Cell processor is very new
- It is not an easy architecture to work with
- The tool chain is thin and brittle
- Most of the staff have limited experience
- Projects you are doing are of your own making. They aren’t canned exercises that are tried and proven.
- You will face unexpected problems.
- WE ARE ALL IN THIS TOGETHER!!
Grading

● Mini Quizzes 16%
  ■ At the beginning of each class day
  ■ 5 minutes each

● Lab Projects 24%

● Final Group Project 60%
Final Competition

- The competition will be decided on
  - Performance
  - Completeness
  - Algorithmic complexity
  - Demo and Presentation

- The winning team will
  - Get gift certificates ($150 each)
  - Be invited to IBM TJ Watson Research Center for a day
    - Tour of the facilities
    - Present your project
Staff

- Prof. Saman Amarasinghe
  - Interested in languages, compilers and computer architecture
  - Raw Processor (with Prof. Anant Agarwal)
  - StreamIt language
  - SUIF parallelizing compiler

- Dr. Rodric Rabbah
  - Currently a researcher at IBM Watson Research Center
  - Was a research scientist at CSAIL before that
  - Interested in compilers, computer architecture and FPGAs
Guest Lectures

- **Dr. Michael Perrone**
  - IBM Watson Research Center
  - Expert in Cell Architecture and Application Development

- **Prof. Alan Edelman**
  - Math and CS. Interested in parallel algorithms

- **Prof. Arvind**
  - Parallel architectures, compilers and languages

- **Dr. Bradley Kuszmaul**
  - Research scientist at CSAIL working on Cilk

- **Mike Acton**
  - Professional game developer

- **Bill Thies**
  - CSAIL PhD candidate
  - Architect of StreamIt
Lecture Organization

Extracting Parallelism

Implicit

- Superscalar Processors (start of Lecture 3)

Explicit

- Parallelizing Compilers (Lectures 11 & 12)

Languages

- StreamIt (Lecture 8)
- Star-P (Lecture 13)
- BlueSpec (Lecture 14)
- Cilk (Lecture 15)

Library

- Concurrency (Lecture 4)
- Design Patterns (Lectures 5, 6, 7)
<table>
<thead>
<tr>
<th></th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 8</td>
<td><strong>10:00 – 10:55</strong>&lt;br&gt;Lecture 1: Course Introduction</td>
<td>Recitation 1: Getting to Know Cell</td>
<td>Lecture 3: Introduction to Parallel Architectures</td>
<td>Project Reviews</td>
<td>Lecture 5: Parallel Programming Concepts</td>
</tr>
<tr>
<td></td>
<td><strong>11:05 – 12:00</strong>&lt;br&gt;Lecture 2: Introduction to Cell Processor</td>
<td>Lecture 4: Introduction to Concurrent Programming</td>
<td></td>
<td></td>
<td>Lecture 6: Design Patterns for Parallel Programming I</td>
</tr>
<tr>
<td>Jan 15</td>
<td><strong>10:00 – 10:55</strong>&lt;br&gt;Holiday</td>
<td>Recitation 2-3: Cell Programming Hands-On</td>
<td>Lecture 7: Design Patterns for Parallel Programming II</td>
<td>Recitation 4: Cell Debugging Tools</td>
<td>Lecture 9: Debugging and Performance Monitoring</td>
</tr>
<tr>
<td></td>
<td><strong>11:05 – 12:00</strong>&lt;br&gt;Lecture 8: StreamIt Language</td>
<td>Lecture 9: StreamIt Language</td>
<td></td>
<td></td>
<td>Lecture 10: Performance Optimizations</td>
</tr>
<tr>
<td>Jan 22</td>
<td><strong>10:00 – 10:55</strong>&lt;br&gt;Lecture 11: Classic Parallelizing Compilers</td>
<td>Recitation 5, 6: Cell Performance Monitoring Tools</td>
<td>Lecture 13: Star-P</td>
<td></td>
<td>Lecture 15: Cilk</td>
</tr>
<tr>
<td></td>
<td><strong>11:05 – 12:00</strong>&lt;br&gt;Lecture 12: StreamIt Parallelizing Compiler</td>
<td>Lecture 14: Synthesizing Parallel Programs</td>
<td></td>
<td></td>
<td>Lecture 16: Anatomy of a Game</td>
</tr>
<tr>
<td>Jan 29</td>
<td><strong>10:00 – 10:55</strong>&lt;br&gt;Lecture 17: The Raw Experience</td>
<td></td>
<td>Group Presentations</td>
<td></td>
<td>Awards &amp; Reception</td>
</tr>
<tr>
<td></td>
<td><strong>11:05 – 12:00</strong>&lt;br&gt;18: The Future</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>