Seminar 236832

Advanced Topics in Concurrent Programming

Erez Petrank
Agenda

• The (practical) problem of parallelism today.
• Basic terms.
• Administration.
Moore’s Law

• The number of transistors that can be inexpensively placed on an integrated circuit is increasing exponentially, doubling approximately every two years.
  [Gordon E. Moore, 1965]

• Also applies to: processing speed, memory capacity, and even the resolution of digital cameras.

• Predicted for “at least 10 years”, lasted at least 50…
Moore’s Law

Transistors Per Die

- Memory
- Microprocessor

Source: Intel


- 1K, 4K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M
- 4004, 8080, 8086, 80286, i386™, i486™, Pentium®, Pentium® Pro, Pentium® II, Pentium® III, Pentium®4, Pentium®®
Transistors and Performance

- Make transistors *smaller*, and then
- they get *closer* to each other, and then
- the *delay* in spreading their signal becomes *smaller*, and then
- we can *increase* the operating *frequency*. Hurray!

But then, power increases!
Processor Development

Transistors

Clock Speed

Power

Performance
For Software: Free Lunch is Over

- Want faster software? Buy a new computer...
- So software producers did not have to work hard for better performance.
Hardware Change

- Do not increase frequency.
- But there is more space!
- Put several “cores” on the same chip.
- Let the software guys parallelize their programs.
Current Status

• Multicores everywhere.
• But software is not ready!
  – Lots of existing single-threaded code.
  – Difficult to write parallel code.
  – Difficult to debug and ensure reliability.

• This is a major challenge for the computing world today.
Solution: Automatic Parallelization?

- **Automatic Parallelization** (compiler or source to source).
  - Lots of research and theory, mostly for Fortran (no pointers, no recursion)
  - But little success for general code
  - Some success for scientific computing (FFT, Matrix multiplication, etc.).
- Problem: existing dependencies in “typical” code foils parallelism.
- Seeming conclusion: developers must specify parallelism or at least give hints about it.
Solution: Design a New Language?

• **Design a new language** with an intuitive and robust expressiveness for parallelism.
  – How does one specify parallel execution?
  – How does synchronization go?

• Current programming languages typically use threads. For many programmers threads create synchronization problems, and the shared memory implies data races.

• Difficult to develop and debug.
Parallel Programming Environments in the 90’s

<table>
<thead>
<tr>
<th>Environment</th>
<th>Description</th>
<th>Environment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCFEI</td>
<td>CORRELATE</td>
<td>EPII</td>
<td>METAL</td>
</tr>
<tr>
<td>ACIS</td>
<td>CPS</td>
<td>GUARDI</td>
<td>MENTOR</td>
</tr>
<tr>
<td>ACT++</td>
<td>CRAY</td>
<td>HASKELL</td>
<td>MEGAPLUS</td>
</tr>
<tr>
<td>Active messages</td>
<td>CSF</td>
<td>HASKELL</td>
<td>MILLIPede</td>
</tr>
<tr>
<td>Add</td>
<td>CUADER</td>
<td>HASKELL</td>
<td>OPENMP</td>
</tr>
<tr>
<td>Advance</td>
<td>CEHUNN</td>
<td>JAVA</td>
<td>ORCA</td>
</tr>
<tr>
<td>AODAP</td>
<td>DAGGER</td>
<td>JAVAPAR</td>
<td>PAR++</td>
</tr>
<tr>
<td>AFAP</td>
<td>DAPPLE</td>
<td>MAESTRO</td>
<td>PET</td>
</tr>
<tr>
<td>ALWAN</td>
<td>Data-Parallel C</td>
<td>MATLAB</td>
<td>PCI</td>
</tr>
<tr>
<td>AM</td>
<td>DC</td>
<td>MATH</td>
<td>PHALAN</td>
</tr>
<tr>
<td>AMIB</td>
<td>DCE</td>
<td>MATH</td>
<td>PLUS</td>
</tr>
<tr>
<td>Apple/S</td>
<td>DOLL</td>
<td>MAESTRO</td>
<td>RIAL</td>
</tr>
<tr>
<td>Amore</td>
<td>DICE</td>
<td>Java</td>
<td>RIAL</td>
</tr>
<tr>
<td>ARTS</td>
<td>DIPC</td>
<td>Java</td>
<td>RIAL</td>
</tr>
<tr>
<td>Athapascan-llb</td>
<td>DOLIB</td>
<td>JavaSpace</td>
<td>RIAL</td>
</tr>
<tr>
<td>Aurora</td>
<td>DOT</td>
<td>JIDEL</td>
<td>RIAL</td>
</tr>
<tr>
<td>Automap</td>
<td>DOSMOS</td>
<td>Joyoc</td>
<td>RIAL</td>
</tr>
<tr>
<td>b.c. threads</td>
<td>DRL</td>
<td>Khoos</td>
<td>RIAL</td>
</tr>
<tr>
<td>Hloz</td>
<td>DSM-Threads</td>
<td>Karina</td>
<td>RIAL</td>
</tr>
<tr>
<td>BSIP</td>
<td>DAS</td>
<td>KOAN-Fortran-S</td>
<td>Nexus</td>
</tr>
<tr>
<td>BlockComm</td>
<td>ECO</td>
<td>LAM</td>
<td>Nexus</td>
</tr>
<tr>
<td>C++</td>
<td>Eiffel</td>
<td>LILAC</td>
<td>Nexus</td>
</tr>
<tr>
<td>C++/C++</td>
<td>EMD</td>
<td>LITE</td>
<td>Nexus</td>
</tr>
<tr>
<td>C++</td>
<td>EPL</td>
<td>JAVA</td>
<td>Nexus</td>
</tr>
<tr>
<td>CarlOS</td>
<td>Excelerator</td>
<td>WWWinds</td>
<td>Nexus</td>
</tr>
<tr>
<td>Cashmere</td>
<td>Excelerator</td>
<td>BSETL-Linda</td>
<td>Nexus</td>
</tr>
<tr>
<td>C4</td>
<td>Express</td>
<td>Paralleld</td>
<td>Nexus</td>
</tr>
<tr>
<td>C8+</td>
<td>Express</td>
<td>Paralleld</td>
<td>Nexus</td>
</tr>
<tr>
<td>ChnP</td>
<td>Elements</td>
<td>Paralleld</td>
<td>Nexus</td>
</tr>
<tr>
<td>Charlotte</td>
<td>FM</td>
<td>Objective-Linda</td>
<td>Nexus</td>
</tr>
<tr>
<td>Charm</td>
<td>FLASH</td>
<td>Objective-Linda</td>
<td>Nexus</td>
</tr>
<tr>
<td>Charm++</td>
<td>The FORCE</td>
<td>Objective-Linda</td>
<td>Nexus</td>
</tr>
<tr>
<td>C3</td>
<td>Fork</td>
<td>LiPS</td>
<td>Nexus</td>
</tr>
<tr>
<td>Cilk</td>
<td>Fortran M</td>
<td>LiPS</td>
<td>Nexus</td>
</tr>
<tr>
<td>CM-Fortran</td>
<td>FX</td>
<td>Locust</td>
<td>Nexus</td>
</tr>
<tr>
<td>Convexor</td>
<td>GA</td>
<td>Maplesoft</td>
<td>Nexus</td>
</tr>
<tr>
<td>Code</td>
<td>GAMMA</td>
<td>Maplesoft</td>
<td>Nexus</td>
</tr>
<tr>
<td>CCOOL</td>
<td>Glimedia</td>
<td>Manifold</td>
<td>Nexus</td>
</tr>
<tr>
<td>Concurrent</td>
<td>NOD</td>
<td>Machine</td>
<td>Nexus</td>
</tr>
<tr>
<td>Core++</td>
<td>NOD</td>
<td>Machine</td>
<td>Nexus</td>
</tr>
<tr>
<td>Core++</td>
<td>NOD</td>
<td>Machine</td>
<td>Nexus</td>
</tr>
<tr>
<td>Core++</td>
<td>NOD</td>
<td>Machine</td>
<td>Nexus</td>
</tr>
</tbody>
</table>

Taken from: “Patterns for Parallel Programming” T. Mattson and K. Kuetzer, 2004
Design a New Language?

• Some big designs are still in the work:
  – Transactional memories ?
  – SUN/ORACLE’s Fortress?
  – IBM’s X10?

• An open problem.
Help Experts Can Provide

- **Build libraries:**
  - Experts can work hard and design a linked-list with “excellent properties”.
  - Non-experts can just use that list in their programs to obtain a good behavior.

- **Properties:**
  - Efficiency (time, space, scalability, etc.)
  - Progress guarantees

- We will go over algorithm designs.
Efficiency

- Parallelism brings a new dimension to efficiency: synchronization overheads and delays.
- What does it mean for a parallel program to be “good”?
- Can we avoid using locks and just use low-level synchronization such as compare-and-swap?
- Various theories about lock-freedom, fairness, progress guarantee, etc.
- We will explore some issues in that end, focusing on lock-freedom.
Another Major Problem

• Debugging and Reliability

How difficult is it to find (and fix) a visible bug?

When benchmarks are running well on the developer platform, how many bugs are still hiding?
Debugging Parallel Programs

• Some bugs appear in a specific schedule and become visible only once every $n$ runs.
  – How large is $n$?

• Sometimes bugs produce different behaviors depending on the schedule.

• Can we ever be sure that a program is bug-free?
Debugging Tools

- Replay (same schedule)
- Randomly choose schedules
- Go over all “reasonable” schedules (and check what?)
- Race detection
- Determinism
- Etc.
Verification

• Verify that the program satisfies some properties.
• Very difficult (undecidable in the general case).
• Should we prove each code written? Can we verify a set of programs?
• Tools today are not ready for general developer use.
• Various logics: trade-offs between expressiveness and complexity.
• An important first step: type-safety.
• Verify high or low level code?
  – Verify the compiler?
  – Proof carrying code,
  – Typed assembly
Parallelizing Software: a Major Focus of Industry and Academia

• Large companies are trying various directions, (e.g., Intel, Oracle, IBM, Microsoft, Google, VMWare. We will discuss several directions in this seminar.

• Major conferences devote sessions for attempts to deal with the parallelism challenge. (See the resurrection of transactional memories.)
Major Focus of Industry and Academia

- Some money is being allocated…
- Example 1: DARPA (Pentagon’s Defense Advanced Research Projects Agency) has a large fund (~500M) for the next generation parallel platform. Surviving contractors were IBM and Cray.
  - Microsoft and Intel have committed a combined $20 million to the Berkeley and UIUC research centers over the next five years. An additional $8 million will come from UIUC, and UC Berkeley has applied for $7 million in funds from a state-supported program to match industry grants.
Motivation Summary

- Parallel platforms are widely available.
- We do not know how to efficiently develop, debug and verify parallel software.
- A major challenge for the computing world today.
- We will study solution directions,
  - and technical issues in the design of scalable parallel algorithms.
Some Basics

Platforms, Processes, Threads
Hardware

- Multi-processor (or multi-core),
- NUMA,
- Distributed system,
- and combinations.
Multiprocessor (Multicore)

Distributed Architecture
Hardware

- Main performance and scalability issue:
  - communication via bus or network.
  - memory access
- Main programming implication: communication and synchronization via message passing, or shared memory.
Basics: Software

- **A process**: a snapshot description of a program execution (excluding file contents, etc.).
- **Typically**: program counter, memory content (including local registers and global memory), and a process id.
- **Unix**: create via execution of a fork; synchronization via join, information transfer via sharing of parts of the memory, or sockets.
- **Windows**: `createProcess (...)`. 
Processes are Costly

• A new process (Unix) receives a copy of parents properties.
• Opening: naively, all memory copied.
  – Practically, copy-on-write.
• Switching: TLB invalidated.

• Making it lighter and faster: share memory… Threads!
Threads

• A process may contain several threads, which share its memory space.
• A thread consists of a thread id, program counter, registers, and a stack.
• Java: create a sub-class of Thread and override the “run()” method with the code.
• Communication using the shared memory.
• Synchronization via locks, monitors, and special instructions (compare-and-swap, etc.).
Comparison

- Processes are easily migrated (to execute remotely).
- Processes are isolated and thus less bug-prone.
- Threads are lighter to create and switch.
Summary: Basic Terms

• Computer platforms today typically consist of multicores or multiprocessors.
• They may connect to create a distributed system.
• Software for parallel computation is usually written by spawning processes or creating threads.
List of Topics

- Introduction
- Concurrent Algorithms
- Debugging
- Concurrent execution on GPUs
- Concurrent execution on non-volatile memory
- Server architecture
- Languages and Runtimes for Concurrent and Distributed Programming
- Systems Scalability
Administration

• You picked 18 out of 36 papers.
  • This means you influenced the material.
  • This means everybody had a choice.

• Next week Topics 1 and 2 (introduction)
  • אדר אמיר ויעל דואק

• There will be no class on the third week Nov. 9th.
• Topics 4 will happen in the first hour of the fourth week and I will give the second hour.
• Web page for announcements.
Presentation

- Each student will pass a lecture about one of the topics.
- A lecture should take 50 minutes, two lectures each week.
- PowerPoint or Keynote or PDF in English.
  - Note that the interface is VGA. (Check your connection ahead of time.)
- Lectures should be sent to me (erez@cs) in the evening before the seminar (by 19:00 Wednesday) or earlier.
- After the talk, the lecturer will modify the slides to create a final version and send it to me the next day (Thursday).
- The lectures will be posted on the course web-site.
- Any use of available material should be clearly stated (on each slide).
Grading

- Lecture (at least 85%),
- participation (at most 15%),
- You must attend…
  - Reduction of 5 points for each (unjustified) absence.
  - A justified absence implies a summary.
- The talk will be graded by:
  - Knowledge of the material.
  - Slides effectiveness.
  - Communication of the ideas.
  - Also, keeping the lecture time limits.
  - Also, following the procedure.
Talks

• Main goal: make people understand!
• Original presentations of most topics can be found on the web. You can use these slides as the basis for your presentations.
  – But the original presentations are for a different timing length and a different audience.
  – Provide some background and expand on interesting aspects.
• Practice giving the talk (on your friends / fellow students).
Registration

• Registration is manual
  – I will register undergraduates.
  – Graduate students register on their own.

• Assignments appear on the web. Let me know if there is a problem.

• Check for info and updates on:
  http://www.cs.technion.ac.il/~erez/courses/seminar18
To Conclude…

• 36 papers offered for choice.
• 18 papers will be presented (according to your choice).

• Send me your talks the evening before!

• See you next week.