EECS 570
Lecture 1
Parallel Computer Architecture
Winter 2022
Prof. Yatin Manerkar

http://www.eecs.umich.edu/courses/eecs570/

Slides developed in part by Profs. Austin, Adve, Falsafi, Martin, Narayanasamy, Nowatzyk, Peh, and Wenisch of CMU, EPFL, MIT, UPenn, U-M, UIUC
EECS 570 Class Info

• Instructor: Prof. Yatin Manerkar (manerkar@umich.edu)
  - Pronunciation: “Yuh-teen Muh-nair-cur” is pretty close 😊
  - https://web.eecs.umich.edu/~manerkar/

• Research interests:
  - Intersection of computer architecture/systems and formal methods
  - Formal modelling, verification, and synthesis of computing systems
  - Memory consistency models and cache coherence
  - Hardware security
  - Ethical AI
EECS 570 Class Info

• GSIs:
  - Joseph Nwabueze (jnnwa@umich.edu)
  - Alhad Daftardar (alhad@umich.edu)

• Class info:
  - URL: https://www.eecs.umich.edu/courses/eecs570/
  - Canvas for reading quizzes and reporting grades
  - Piazza for discussions and project coordination
Meeting Times

• Lecture:
  ❑ MW 1:30-2:50pm

• Discussion
  ❑ F 1:30-2:20pm
  ❑ Used for talking about programming assignments and projects
  ❑ Also potentially for make-up lectures
  ❑ Keep the slot free, but we won’t use it often

• Office Hours:
  ❑ Prof. Manerkar: Thursdays 10am-11am (Zoom link)
  ❑ GSIs: TBD (will be posted on course webpage)

• Q&A:
  ❑ Use Piazza for all technical questions
  ❑ Use email very sparingly
Who Should Take 570?

- Graduate Students (and seniors interested in research)
  - Computer architects to be
  - Computer system designers
  - Those interested in computer systems

- Required Background
  - Computer Architecture (e.g., EECS 470)
  - C/C++ Programming

- If you do not have the required background (especially a good base in computer architecture), you should strongly reconsider taking this course
  - Topics covered such as coherence and consistency are complex even for computer architects!
Grading (tentative)

- 2 Programming Assignments: 5% (PA1) and 10% (PA2)
- Reading Quizzes: 10%
- Midterm Exam: 25%
- Final Exam: 25%
- Final Research Project: 25%

- As you can see, the course requires a significant amount of work!
Grading (Contd.)

• Group studies are encouraged
• Group discussions are encouraged
• However,...
  □ All programming assignments must be results of individual work
  □ All reading quizzes must be done individually, questions/answers should not be posted publicly

• There is no tolerance for academic dishonesty. Please refer to the University Policy on cheating and plagiarism. Discussion and group studies are encouraged, but all submitted material must be the student's individual work (or in case of the project, individual group work).
Some Advice on Reading

• If you carefully read every paper from start to finish, it will take a very long time.

• Learn to skim past details that are not critical to the paper’s overall message.
Reading Quizzes

• You must take an online quiz for every paper
• Quizzes must be completed by class start via Canvas
• There will be 2 multiple choice questions
  □ The questions are chosen randomly from a list
  □ You only have 5 minutes
    ○ Not enough time to find the answer if you haven’t read the paper
  □ You only get one attempt
• Some of the questions may be reused on the midterm/final
• 4 lowest quiz grades (of about 40) will be dropped over the course of the semester (e.g., can skip some if you fall ill)
  □ Retakes/retries/reschedules will not be given for any reason
Final Project

• Original research on a topic related to the course
  - Goal: a high-quality 6-page workshop paper by end of term
  - 25% of overall grade
  - Done in groups of 4-5
  - Poster session in April

• Timeline will be posted on course website

• Available infrastructure
  - FeS2 and M5 multiprocessor simulators
  - GPGPUsim
  - Pin
  - Xeon Phi accelerators

• Suggested topic list will be distributed in a few weeks
Late Assignment Policy

• Applies only to PA1 and PA2
• Deduction of 20% per day, for up to 3 days
• After that, no late assignments will be accepted
• You will have ample time to work on the assignments, so please start early!
Regrade Policy

• Applies only to PA1, PA2, and midterm
• Regrade requests must be submitted in writing within one week from the day the assignment/midterm is handed back
• Regrade requests must specify clearly what the grading issue is
• On a regrade, the entire assignment will be regraded, and the grade may go up or down
Course Outline

• Unit I – Parallel Programming Models
  □ Message passing, shared memory (pthreads and GPU)

• Unit II – Synchronization
  □ Synchronization, Locks, Lock-free structures
  □ Transactional Memory

• Unit III – Coherency and Consistency
  □ Snooping bus-based systems
  □ Directory-based distributed shared memory
  □ Memory Consistency Models

• Unit IV – Interconnection Networks
  □ On-chip and off-chip networks

• Unit V – Applications & Accelerators
  □ ML, health, and data-center applications
  □ Custom hardware
Announcements

No discussion this Friday.

- Online quizzes (Canvas) on 1st readings due Monday, 1:30pm
- Please sign up for Piazza (sign up link in announcement on Canvas)
Readings

For Monday 1/10 *(quizzes due by 1:30pm)*


For Wednesday 1/12:

<table>
<thead>
<tr>
<th>Christina Delimitrou and Christos Kozyrakis. Amdahl's law for tail latency. Commun. ACM 61, July 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>H Kim, R Vuduc, S Baghsorkhi, J Choi, Wen-mei Hwu, Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU), Ch. 1</td>
</tr>
</tbody>
</table>
Parallel Computer Architecture

The Multicore Revolution
Why did it happen?
If you want to make your computer faster, there are only two options:

1. increase clock frequency

2. execute two or more things in parallel
   - Instruction-Level Parallelism (ILP)

Programmer specified explicit parallelism
The ILP Wall

- 6-issue has higher IPC than 2-issue, but not by 3x
  - Memory (I & D) and dependence (pipeline) stalls limit IPC

*Figure 4. IPC Breakdown for a single 2-issue
Figure 5. IPC Breakdown for the 6-issue processor.

Olukotun et al ASPLOS 96
Single-thread performance


Conclusion: Can’t scale MHz or issue width to keep selling chips

Hence, multicore!
The Power Wall

- Transistors (100,000's)
- Power (W)
- Performance (GOPS)
- Efficiency (GOPS/W)

Limits on heat extraction
Limits on energy-efficiency of operations
The Power Wall

- Transistors (100,000's)
- Power (W)
- Performance (GOPS)
- Efficiency (GOPS/W)

Limits on heat extraction
Stagnates performance growth
Limits on energy-efficiency of operations

Classic CMOS Dennard Scaling: the Science behind Moore’s Law

**Scaling:**
- Voltage: \( V / \alpha \)
- Oxide: \( t_{ox} / \alpha \)
- Wire width: \( W / \alpha \)
- Gate width: \( L / \alpha \)
- Diffusion: \( x_d / \alpha \)
- Substrate: \( \alpha N_A \)

**Results:**
- Higher Density: \( \sim \alpha^2 \)
- Higher Speed: \( \sim \alpha \)
- Power/ckt.: \( 1 / \alpha^2 \)
- Power Density: \( \sim \text{Constant} \)

Source: Future of Computing Performance: Game Over or Next Level?, National Academy Press, 2011

\[ P = C V^2 f \]

Post-classic CMOS Dennard Scaling

Post Dennard CMOS Scaling Rule

Scaling:

Voltage: \( \frac{V}{\alpha} \), \( V \)
Oxide: \( \frac{t_{ox}}{\alpha} \)
Wire width: \( \frac{W}{\alpha} \)
Gate width: \( \frac{L}{\alpha} \)
Diffusion: \( \frac{x_d}{\alpha} \)
Substrate: \( \alpha N_A \)

Results:

Higher Density: \( \sim \alpha^2 \)
Higher Speed: \( \sim \alpha \)
Power/skt: \( \frac{1}{\alpha^2} \) \( \frac{1}{1} \)
Power Density: \( \sim \alpha^2 \) \( \sim \text{Constant} \)

\[ P = C V^2 f \]

TODO: Chips w/ higher power (no), smaller (⊗), dark silicon (😊), or other (?)

Leakage Killed Dennard Scaling

Leakage:

- Exponential in inverse of $V_{th}$
- Exponential in temperature
- Linear in device count

To switch well

- must keep $V_{dd}/V_{th} > 3$

$V_{dd}$ can’t go down
Multicore: Solution to Power-constrained design?

Power = CV^2 F \quad F \propto V
Scale clock frequency to 80%

Now add a second core

Same power budget, but 1.6x performance!

But:

- Must parallelize application
- Remember Amdahl’s Law!
What Is a Parallel Computer?

“A collection of processing elements that communicate and cooperate to solve large problems fast.”

Almasi & Gottlieb, 1989
Spectrum of Parallelism

Bit-level | Pipelining | ILP | Multithreading Multiprocessing | Distributed
---|---|---|---|---
[Diagram of Bit-level Parallelism] | [Diagram of Pipelining] | [Diagram of ILP] | [Diagram of Multithreading Multiprocessing] | [Diagram of Distributed Processing]

EECS 370 | EECS 470 | EECS 570 | EECS 591

Why multiprocessing?

- Desire for performance
- Techniques from 370/470 difficult to scale further
Why Parallelism Now?

- These arguments are no longer theoretical

- All major processor vendors are producing multicore chips
  - Most machines today are already parallel machines
  - All programmers will be parallel programmers???

- New software model
  - Want a new feature? Hide the “cost” by speeding up the code first
  - All programmers will be performance programmers???

- Some may eventually be hidden in libraries, compilers, and high level languages
  - But a lot of work is needed to get there

- Big open questions:
  - How should the chips, languages, OS be designed to make it easier for us to develop parallel programs?
Multicore in Products

- “We are dedicating all of our future product development to multicore designs. ... This is a sea change in computing”

Paul Otellini, President, Intel (2005)

- All microprocessor companies switch to MP (2X cores / 2 yrs)

<table>
<thead>
<tr>
<th></th>
<th>Intel’s Nehalem-EX</th>
<th>Azul’s Vega</th>
<th>nVidia’s Tesla</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors/System</td>
<td>4</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>Cores/Processor</td>
<td>8</td>
<td>48</td>
<td>448</td>
</tr>
<tr>
<td>Threads/Processor</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Threads/System</td>
<td>64</td>
<td>768</td>
<td>1792</td>
</tr>
</tbody>
</table>
Revolution Continues..

Azul’s Vega 3 7300
54-core chip
864 cores
768 GB Memory
May 2008

Blue Gene/Q Sequoia
16-core chip
1.6 million cores
1.6 PB
2012

Sun’s Modular DataCenter ‘08
8-core chip, 8-thread/core
816 cores / 160 sq.feet

Lakeside Datacenter (Chicago)
1.1 milion sq.feet
~45 million threads
Multiprocessors Are Here To Stay

• Moore’s law is making the multiprocessor a commodity part
  □ 1B transistors on a chip, what to do with all of them?
  □ Not enough ILP to justify a huge uniprocessor
  □ Really big caches? $t_{\text{hit}}$ increases, diminishing $\%_{\text{miss}}$ returns

• Chip multiprocessors (CMPs)
  □ Every computing device (even your cell phone) is now a multiprocessor
Accelerator-Level Parallelism [Reddi and Hill 2019]

- Accelerators are specialized processing elements for different types of tasks
  - Machine learning, crypto, graphics (GPUs), etc

- Apple A12 System-on-Chip (SoC)
  - More than 40 accelerators [Wang and Shao 2019]
Heterogeneous Parallelism Across the Stack

- Parallelism has percolated up to high-level languages and compilers
  - Java, C/C++11, C#, etc. all have threads
- OS modified to best utilise parallel hardware
- New software toolchains for hardware accelerators
  - e.g. TensorFlow, PyTorch, TVM
- New landscape!
Parallel Programming Intro
Motivation for MP Systems

• Classical reason for multiprocessing:
  More performance by using multiple processors in parallel

  □ Divide computation among processors and allow them to work concurrently

  □ Assumption 1: There is parallelism in the application

  □ Assumption 2: We can exploit this parallelism
Finding Parallelism

1. Functional parallelism
   - Car: \{engine, brakes, entertain, nav, \ldots\}
   - Game: \{physics, logic, UI, render, \ldots\}
   - Signal processing: \{transform, filter, scaling, \ldots\}

2. Data parallelism
   - Vector, matrix, db table, pixels, \ldots

3. Request parallelism
   - Web, shared database, telephony, \ldots
Computational Complexity of (Sequential) Algorithms

- Model: Each step takes a unit time

- Determine the time (/space) required by the algorithm as a function of input size
Sequential Sorting Example

- Given an array of size $n$
- MergeSort takes $O(n \log n)$ time
- BubbleSort takes $O(n^2)$ time
- But, a BubbleSort implementation can sometimes be faster than a MergeSort implementation
- Why?
Sequential Sorting Example

- Given an array of size n
- MergeSort takes $O(n \log n)$ time
- BubbleSort takes $O(n^2)$ time
- But, a BubbleSort implementation can sometimes be faster than a MergeSort implementation
- The model is still useful
  - Indicates the scalability of the algorithm for large inputs
  - Lets us prove things like a sorting algorithm requires at least $O(n \log n)$ comparisons
We need a similar model for parallel algorithms