Lecture 1: Multicore/Manycore Revolution

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
Who Am I?

• Mohamed Zahran (aka Z)
• http://www.mzahran.com
• Research interest:
  – computer architecture
  – hardware/software interaction
• Office hours:
  – M-W 11am-12pm
  – or by appointment
• Office: WWH 320
Goals of This Course

• What are multicore/manycore processors?
• Why do we have them?
• What are the challenges in dealing with them?
• How to make the best use of them in our software?
• What will the future likely be both in hardware and software?
My wishes for this course

• Don’t be afraid of hardware
• Understand the hardware/software interaction
• Enjoy the challenge of making the best use of hardware to the benefit of software
• Enhance your way of thinking about parallelism and parallel programming models
• Build a vision about technology and its future
Grading

- Homework assignments: 25%
- Programming assignments: 25%
- Project: 50%
Policies: Assignments

• You must work alone on all assignments
  – Post all questions on the forums at NYU classes
  – You are encouraged to answer others’ questions, but refrain from explicitly giving away solutions.

• Hand-ins
  – All assignments submitted before the deadline through NYU classes by 11:55pm of the due day.
First ... A bit of history
Computer History

Eckert and Mauchly

- 1st working electronic computer (1946)
- To reprogram it you need to re-arrange the cords
- 18,000 Vacuum tubes
- 1,800 instructions/sec
- 3,000 ft$^3$
Computer History

Programming the ENIAC!
Computer History

• Von Neumann presented his idea of stored program concept.
• Maurice Wilkes built it.

EDSAC 1 (1949)

1st stored program computer
650 instructions/sec
1,400 ft³

http://www.cl.cam.ac.uk/UoCCL/misc/EDSAC99/
Computer History

• After the vacuum tubes, **transistors** were invented (1947) → 2\(^{\text{nd}}\) generations of computers

  • UNIVAC (UNIversal Automatic Computer)
  • Introduced in the 50s
Computer History

• From transistors to integrated circuits (IC) → 3rd generation of computers
• One IC can host hundreds (then thousands, then millions) of transistors → computers are getting smaller
Intel 4004 Die Photo

- Introduced in 1970
  - First microprocessor
- 2,250 transistors
- 12 mm²
- 108 KHz
Intel 8086 Die Scan

- 29,000 transistors
- 33 mm$^2$
- 5 MHz
- Introduced in 1979
  - Basic architecture of the IA32 PC
Intel 80486 Die Scan

- 1,200,000 transistors
- 81 mm²
- 25 MHz
- Introduced in 1989
  - 1ˢᵗ pipelined implementation of IA32
Pentium Die Photo

• 3,100,000 transistors
• 296 mm²
• 60 MHz
• Introduced in 1993
  – 1st superscalar implementation of IA32
Pentium III

- 9,500,000 transistors
- 125 mm²
- 450 MHz
- Introduced in 1999

[Image of Pentium III chip]

Pentium 4

- 55,000,000 transistors
- 146 mm²
- 3 GHz
- Introduced in 2000

http://www.chip-architect.com
How did the hardware evolve like that?

Let’s look at different waves (generations of architectures)
First Generation (1970s)

Single Cycle Implementation
Second Generation (1980s)

- Pipelining:
  - the hardware divided into stages
  - temporal parallelism
  - Number of stages increases with each generation

- Maximum CPI (Cycles Per Instruction) = 1
  - Due to dependencies
    (i.e. an instruction must wait for the result of another instruction)
Some Enhancements

- Cache Memory
  - Multi-level caches
- Virtual Memory
  - TLB
Third Generation (1990s)

- ILP (Instruction Level Parallelism)
- Spatial parallelism
- Executing several instructions at the same time is called superscalar capability.
- performance = instructions per cycle (IPC)
- Speculative Execution (prediction of branch direction) is introduced to make the best use of superscalar capability → This can make some instructions execute out-of-order!!
Fourth Generation (2000s)

Simultaneous Multithreading (SMT)  
(aka Hyperthreading Technology)
The Status-Quo

• We moved from single core to multicore to manycore:
  – for technological reasons ... As we will see shortly.

• Free lunch is over for software folks
  – The software will not become faster with every new generation of processors

• Not enough experience in parallel programming
  – Parallel programs of old days were restricted to some elite applications → very few programmers
  – Now we need parallel programs for many different applications
The Famous Moore’s Law
Moore’s law works because of ...

Dennard scaling

MOSFETs continue to function as voltage-controlled switches while all key figures of merit such as layout density, operating speed, and energy efficiency improve provided geometric dimensions, voltages, and doping concentrations are consistently scaled to maintain the same electric field.
Effect of Moore's law

- ~1986 - 2002 → 50% performance increase
- Since 2002 → ~20% performance increase

Hmmm ...

- Why do we care? 20%/year is still nice.
- What happened at around 2002?
- Can’t we have auto-parallelizing programs?
Why do we care?

• More realistic games
• Decoding the human genome
• More accurate medical applications

The list goes on and on ….

As our computational power increases → the number of problems we can seriously consider also increases.
Climate modeling
Protein folding
Drug discovery
Data analysis
People ask for more improvements → People get used to the software → Better Software → Hardware Improvement → Positive Cycle of Computer Industry
How Did These Advances Happen?

- Wishes
  - Software Community
  - Computer Architecture
  - Process Technology

- Design
  - Performance
  - Restrictions
  - Restrictions
  - Capabilities
Performance in the past achieved by:

- clock speed
- execution optimization
- cache

Performance now achieved by:

- hyperthreading
- multicore
- cache
Why did we build parallel machines (and continue to do so)?

(multicore, multiprocessors, multi-anything!)
Power Density

Moore’s law is giving us more transistors than we can afford!

Scaling clock speed (business as usual) will not work

This is what happened at around 2002!

Source: Patrick Gelsinger, Intel®
**Multicore Processors Save Power**

Power = $C \times V^2 \times F$

Performance = $Cores \times F$

Let’s have two cores

Power = $2 \times C \times V^2 \times F$

Performance = $2 \times Cores \times F$

But decrease frequency by 50%

Power = $2 \times C \times V^2/4 \times F/2$

Performance = $2 \times Cores \times F/2$

Power = $C \times V^2/4 \times F$

Performance = $Cores \times F$
# Inflection Point in Computing

<table>
<thead>
<tr>
<th>Late 20th Century</th>
<th>The New Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moore’s Law — 2× transistors/chip every 18-24 months</td>
<td>Transistor count still 2× every 18-24 months, but see below</td>
</tr>
<tr>
<td>Dennard Scaling — near-constant power/chip</td>
<td>Gone. Not viable for power/chip to double (with 2× transistors/chip growth)</td>
</tr>
<tr>
<td>The modest levels of transistor unreliability easily hidden (e.g., via ECC)</td>
<td>Transistor reliability worsening, no longer easy to hide</td>
</tr>
<tr>
<td>Focus on computation over communication</td>
<td>Restricted inter-chip, inter-device, inter-machine communication (e.g. Rent’s Rule, 3G, GigE); communication more expensive than computation</td>
</tr>
<tr>
<td>One-time (non-recurring engineering) costs growing, but amortizable for mass-market parts</td>
<td>Expensive to design, verify, fabricate, and test, especially for specialized-market platforms</td>
</tr>
</tbody>
</table>
A Case for Multicore Processors

- Can exploit different types of parallelism
- Reduces power
- An effective way to hide memory latency
- Simpler cores → easier to design and test → higher yield → lower cost
An intelligent solution

• Instead of designing and building faster microprocessors, put multiple processors on a single integrated circuit.
Now it’s up to the programmers

• Adding more processors doesn’t help much if programmers aren’t aware of them...

• ... or don’t know how to use them.

• Serial programs don’t benefit from this approach (in most cases).
The Need for Parallel Programming

Parallel computing: using multiple processors in parallel to solve problems more quickly than with a single processor.

Examples of parallel machines:

A cluster computer that contains multiple PCs combined together with a high speed network.

A shared memory multiprocessor (SMP) by connecting multiple processors to a single memory system.

A Chip Multi-Processor (CMP) contains multiple processors (called cores) on a single chip.
Cost and Challenges of Parallel Execution

• Communication cost
• Synchronization cost
• Not all problems are amenable to parallelization
• Hard to think in parallel
• Hard to debug
Attempts to Make Multicore Programming Easy

• 1st idea: The right computer language would make parallel programming straightforward
  – Result so far: Some languages made parallel programming easier, but none has made it as fast, efficient, and flexible as traditional sequential programming.
Attempts to Make Multicore Programming Easy

- 2\textsuperscript{nd} idea: If you just design the hardware properly, parallel programming would become easy.
  - Result so far: no one has yet succeeded!
Attempts to Make Multicore Programming Easy

- 3rd idea: Write software that will automatically parallelize existing sequential programs.
  - Result so far: Success here is inversely proportional to the number of cores!
The Multicore Software Triad
Programming Model

??

The Real Hardware
Conclusions

• We are always in need for more performance.

• The free lunch is over for software folks!

• Multicore/Manycore processors are here to stay, so we have to deal with them.
  – Power density pushed us to move to multicore before we are ready for it!

• Knowing about the hardware will make you way more efficient in software!