CS671 Parallel Programming in the Many-Core Era

Lecture 1: Introduction

Zheng Zhang

Rutgers University
CS671 Course Information

‣ Instructor information:
  instructor: zheng zhang
  website: www.cs.rutgers.edu/~zz124/
  office hours: 1pm-2pm Wednesday @ Core 310

‣ Important facts:
  lectures: tuesdays & thursdays 6:40pm - 8:00pm
  location: Core 305
  course page: www.cs.rutgers.edu/~zz124/cs671_fall2013/

‣ Basis for grades (tentative):
  in-class presentations: 30%
  assignments: 10%
  participation: 20%
  project: 40%
CS671 Class Taking Techniques

‣ A mixture of “I talk” and “you talk”
  * we will review a lot of papers
  * you think critically, participate and present papers

‣ No exams!!

‣ Projects
  * suggested topics
  * you can combine it with your own research
  * not necessarily coding
Why parallel?

Why two-core, four-core, ..., many-core?

Why now?
Thrust 1: Power Wall

- End of single core frequency scaling
Moore’s law

Moore’s law: the number of transistors per square inch doubles about every 18 months. (proposed in 1965)

Moore's law

source: wikipedia
Dennard Scaling Rule

- As transistors got smaller, the power density was constant.

Example: if there is a reduction in a transistor’s linear size by 2, the power it uses falls by 4 (with voltage and current both halving).

\[
\text{Power} = \text{Voltage} \times \text{Current} \\
\text{PowerDensity} = \frac{\text{Power}}{\text{Area}} \\
\text{Frequency} \propto \frac{1}{\text{Devicesize}}
\]

Robert H. Dennard: American electrical engineer and inventor. Most known for - the invention of DRAM (Dynamic Random Access Memory) and the development of scaling principles for miniaturization of MOS (Metal Oxide Semiconductor) transistors.
Microprocessor Power Density

V.S. Time

CPU Power = Capacitance x Voltage^2 x Freq

V.S. Device Size

A projection made in 1999
What can we do if we can’t supply enough power (or cool down the chip)?
What can we do if we can’t supply enough power (or cool down the chip)?

Power = Capacitance x Voltage$^2$ x Freq
Thrust 2: Memory Wall

- The CPU Memory Performance Gap
Thrust 3: Instruction-level Parallelism Wall

- Extensively exploited in the past few decade
  - Architecture level: out-of-order execution, super-scalar, VLIW
  - Compiler techniques, instruction scheduling, loop unrolling and etc.
“We will dedicate all of our future product designs to multicore environments”

-- CEO of Intel, Paul S. Otellini, 2004
What is happening at industry

Intel Knights Corner

AMD Heterogeneous APU

GreenArrays G4 4-C18 cores and GA144f18a – 144 Forth Cores

NVIDIA Many-core GPU

and more ......
Parallel Platform is Ubiquitous

Laptops

Desktops

Tablets

Servers

Phones

Network Devices

Wearable Device (Google Glasses) and more ......
Implications to programmers

“Parallelize or ... perish”

• How to write parallel programs?
  • “Writing parallel programs is hard…”
  • Learn programming models and tools

• How to write *efficient* parallel programs?
  • Understand performance tradeoffs in multi/many-core
  • Understand the interplay between compiler, OS, and hardware
Why parallel programming is challenging?

- Exploit parallelism (Amdal’s Law)
- Granularity
- Workload partitioning -- load balancing
- Memory management
- Communication and synchronization
- Energy efficiency
Topics in CS671

- GPGPUs
- Fundamental parallel programming models/language
- Automatic parallelization
- Memory management
- Power and energy efficiency
- Future parallel platforms
GPGPUs

- The most parallel platform on the market

- Programming GPU basics, performance bottlenecks, and optimization challenges, comparison with multi-core
Parallel Programming Models

- Exploiting as much parallelism as possible
- Amdal’s Law
  \[ S = \frac{1}{f_s + \frac{f_p}{N}} \]
  Where:
  \( f_s \) = serial fraction of code
  \( f_p \) = parallel fraction of code = \((1-f_s)\)
  \( N \) = number of processors

- Programming models/languages/framework
  Cilk, TBB, UPC, MapReduce, OpenMP, MPI, ......
  structured v.s. unstructured, regular v.s. irregular,
  shared memory model v.s. message passing, ......
Automatic Parallelization

- Let compiler/runtime library exploit the parallelism
  -- programmers still express the tasks in sequential semantics

- Static time parallelization v.s. dynamic parallelization
  speculative parallelization, polyhedral compilation models, work stealing, transactional memory, map-reduce, ...
Memory Management

- Shared memory hierarchy

- Program locality and theory
  reuse analysis, working set theory, hierarchical sharing
  performance modeling and prediction, complexity analysis
Thermal, Power and Energy efficiency

- Assume thermal and voltage control knobs are given to programmers, what can we do?
- Spatial & temporal scheduling *thermal control, sustainability and green computing*
Next class

- Programming GPGPU basics
- Need a volunteer for next Thursday class presentation -- email me by the end of this week