CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Instructors:
Randy H. Katz
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/fa10

Agenda

- Kinds of Parallelism
- Administrivia
- Technology Break
- Amdahl's Law

Agenda

- Kinds of Parallelism
- Administrivia
- Technology Break
- Amdahl's Law

Alternative Kinds of Parallelism: The Programming Viewpoint

- Job-level parallelism/process-level parallelism
  - Running independent programs on multiple processors simultaneously
  - *Example?*
- Parallel processing program
  - Single program that runs on multiple processors simultaneously
  - *Example?*

Alternative Kinds of Parallelism: Hardware vs. Software

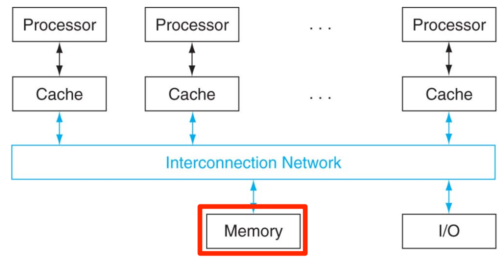
		Software	
		Sequential	**Concurrent**
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Xeon e5345 (Clovertown)	Windows Vista Operating System running on an Intel Xeon e5345 (Clovertown)

- Concurrent software can also run on serial hardware
- Sequential software can also run on parallel hardware
- Focus is on *parallel processing* software: sequential or concurrent software running on parallel hardware

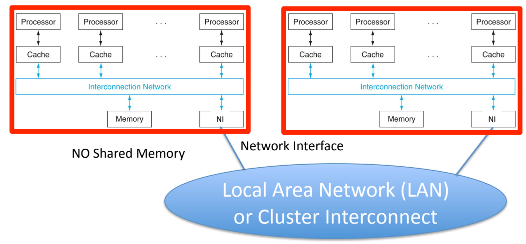
Alternative Kinds of Parallelism: The Processor Viewpoint

- Cluster: Set of computers interconnected across a local area network
  - Like a multiprocessor, but with no shared memory
- Multicore microprocessor
  - Microprocessor containing multiple processors ("cores") in a single IC, with shared memory
- *Our "warehouse-scale" computers are multicore clusters!*

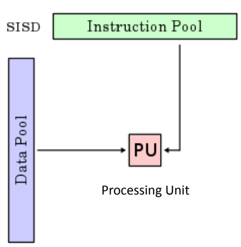
Alternative Kinds of Parallelism: Shared Memory Multiprocessor



Alternative Kinds of Parallelism: Computer Cluster

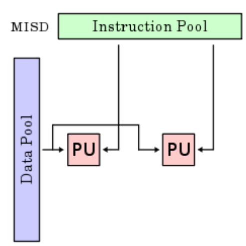


Alternative Kinds of Parallelism: Single Instruction/Single Data Stream



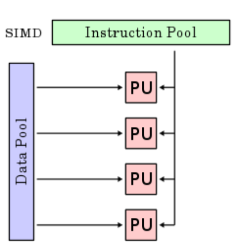
- Single Instruction, Single Data stream (SISD)
  - Sequential computer that exploits no parallelism in either the instruction or data streams. Examples of SISD architecture are traditional uniprocessor machines

Alternative Kinds of Parallelism: Multiple Instruction/Single Data Stream



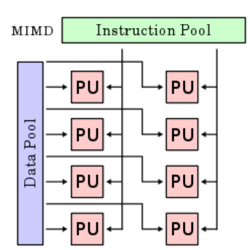
- Multiple Instruction, Single Data streams (MISD)
  - Computer that exploits multiple instruction streams against a single data stream for data operations that can be naturally parallelized. For example, certain kinds of array processors.
  - No longer commonly encountered, mainly of historical interest only

Alternative Kinds of Parallelism: Single Instruction/Multiple Data Stream



- Single Instruction, Multiple Data streams (SIMD)
  - Computer that exploits multiple data streams against a single instruction stream to operations that may be naturally parallelized, e.g., an array processor or Graphics Processing Unit (GPU)

Alternative Kinds of Parallelism: Multiple Instruction/Multiple Data Streams



- Multiple Instruction, Multiple Data streams (MIMD)
  - Multiple autonomous processors simultaneously executing different instructions on different data. Clusters/distributed systems are generally recognized to be MIMD architectures, either exploiting a *single shared memory space* or a *distributed memory space*

Flynn Taxonomy

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345 (Clovertown)

- In 2010, SIMD and MIMD most commonly encountered
- Most common parallel processing programming style: Single Program Multiple Data
  - Single program that runs on all processors of an MIMD
  - Cross-processor execution coordination through conditional expressions (save for Dave and thread parallelism)
- SIMD (aka hw-level *data parallelism*): specialized function units, for handling lock-step calculations involving arrays
  - Scientific computing, signal processing, multimedia (audio/video processing)

SIMD Architectures

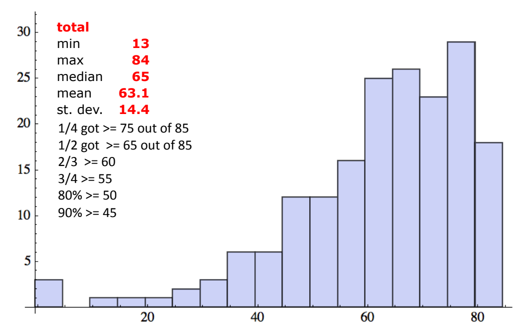
- *Data parallelism*: executing one operation on multiple data streams
- Example to provide context:
  - Multiplying a coefficient vector by a data vector (e.g., in filtering)

    $y[i] := c[i] \times x[i],\ 0 \le i < n$
- Sources of performance improvement:
  - One instruction is fetched & decoded for entire operation
  - Multiplications are known to be independent
  - Pipelining/concurrency in memory access as well
- More on SIMD on Monday

Agenda

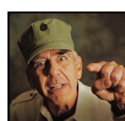
- Kinds of Parallelism
- Administrivia
- Technology Break
- Amdahl's Law

Midterm Results: Scores



Midterm Re-Grades: The Rules

- Written grading appeals only!
- Attend Tuesday's discussion to learn about exam solutions and grading scheme
- If you feel your exam was incorrectly graded, explain why and attach legible write-up to examination booklet
- You have ONE week from Tuesday to hand it in in class, lab, or discussion
- We want to be fair, and correct mistakes, but are unlikely to change partial credit as assigned
- NOTE: We reserve the right to re-examine the whole examination should you turn it in for a re-grade.



Course Kicks Into High Gear

- This week's EC2 Lab
- Next week's SIMD Lab
- EC2 Project #2 (Due Saturday, 10/23, 1 Second to Midnight)
- Following week's Thread Parallelism Lab

Agenda

- Kinds of Parallelism
- Administrivia
- Technology Break
- Amdahl's Law

Agenda

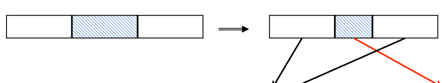
- Kinds of Parallelism
- Administrivia
- Technology Break
- Amdahl's Law

Big Idea: Amdahl's Law

- Speedup due to enhancement E is

$$\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}$$

- Suppose that enhancement E accelerates a fraction F (F <1) of the task by a factor S (S>1) and the remainder of the task is unaffected



Execution Time w/ E = Execution Time w/o E × [(1-F) + F/S]

Speedup w/ E = 1 / [(1-F) + F/S]

Big Idea: Amdahl's Law

Speedup =

Example: the execution time of half of the program can be accelerated by a factor of 2. What is the program speed-up overall?

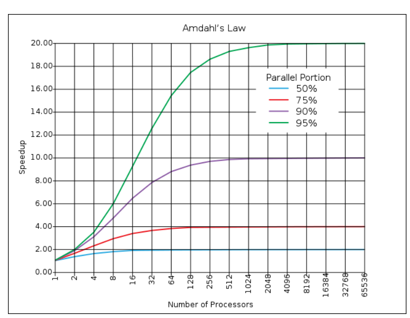
Big Idea: Amdahl's Law

$$\text{Speedup} = \frac{1}{\underbrace{(1 - F)}_{\text{Non-speed-up part}} + \underbrace{\frac{F}{S}}_{\text{Speed-up part}}}$$

Example: the execution time of half of the program can be accelerated by a factor of 2. What is the program speed-up overall?

$$\frac{1}{0.5 + \frac{0.5}{2}} = \frac{1}{0.5 + 0.25} = 1.33$$

Big Idea: Amdahl's Law



If the portion of the program that can be parallelized is small, then the speedup is limited

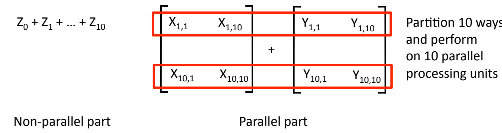
The non-parallel portion limits the performance

Example #1: Amdahl's Law

Speedup w/ E = 1 / [(1-F) + F/S]

- Consider an enhancement which runs 20 times faster but which is only usable 25% of the time
  Speedup w/ E = 1/(.75 + .25/20) = 1.31
- What if its usable only 15% of the time?
  Speedup w/ E = 1/(.85 + .15/20) = 1.17
- Amdahl's Law tells us that to achieve linear speedup with 100 processors, none of the original computation can be scalar!
- To get a speedup of 90 from 100 processors, the percentage of the original program that could be scalar would have to be 0.1% or less
  Speedup w/ E = 1/(.001 + .999/100) = 90.99

Parallel Speed-up Example



- 10 "scalar" operations (non-parallelizable)
- 100 parallelizable operations
- 110 operations

Example #2: Amdahl's Law

Speedup w/ E = 1 / [(1-F) + F/S]

- Consider summing 10 scalar variables and two 10 by 10 matrices (matrix sum) on 10 processors
  Speedup w/ E = 1/(.091 + .909/10) = 1/0.1819 = 5.5
- What if there are 100 processors ?
  Speedup w/ E = 1/(.091 + .909/100) = 1/0.10009 = 10.0
- What if the matrices are 100 by 100 (or 10,010 adds in total) on 10 processors?
  Speedup w/ E = 1/(.001 + .999/10) = 1/0.1009 = 9.9
- What if there are 100 processors ?
  Speedup w/ E = 1/(.001 + .999/100) = 1/0.01099 = 91

Scaling

- To get good speedup on a multiprocessor while keeping the problem size fixed is harder than getting good speedup by increasing the size of the problem.
  - *Strong scaling*: when speedup can be achieved on a multiprocessor without increasing the size of the problem
  - *Weak scaling*: when speedup is achieved on a multiprocessor by increasing the size of the problem proportionally to the increase in the number of processors
- Load balancing is another important factor. Just a single processor with twice the load of the others cuts the speedup almost in half

Summary

- Flynn Taxonomy of Parallel Architectures
  - *SIMD: Single Instruction Multiple Data*
  - *MIMD: Multiple Instruction Multiple Data*
  - SISD: Single Instruction Single Data
  - MISD: Multiple Instruction Single Data
- Amdahl's Law
  - Parallel code speed-up limited by the non-parallelized portion of the code
  - Strong scaling is hard to achieve