CSCI-GA.3033-017

Special Topics:

Multicore Programming

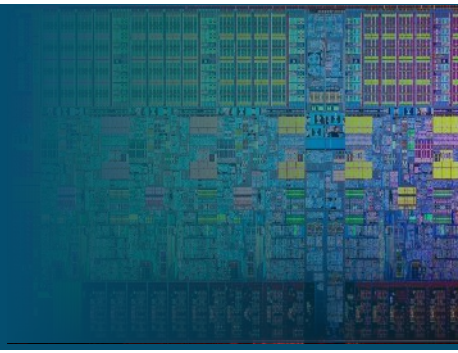
Lecture 10

Multicore Performance Evaluation

Christopher Mitchell, Ph.D.

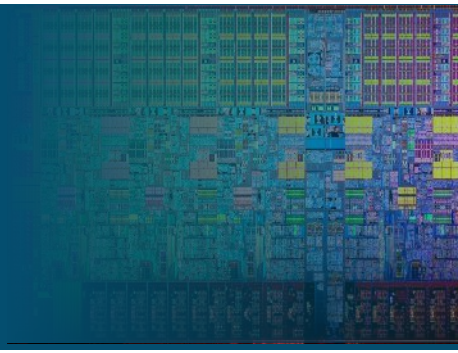
cmitchell@cs.nyu.edu || http://z80.me

Performance Evaluation Goals



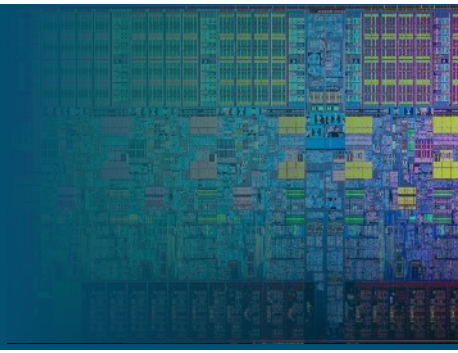
- Measure, report, and summarize performance
- Make intelligent choices
- Determine:
  - Why is some hardware better than others for different programs?
  - What factors of system performance are hardware related?
  - Does performance measure depends on application type?

Outline



- Performance Metrics
- Benchmarks
- Bottlenecks
- Presentation Discussion

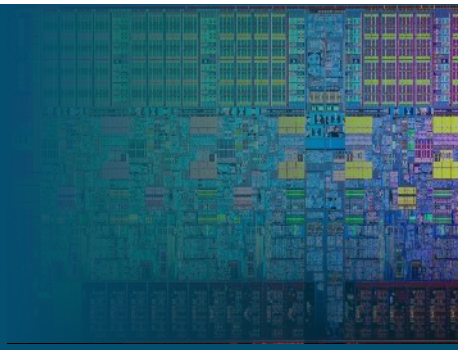
Simple Performance Metrics



- Response time (aka execution time)
  - The time between the start and completion of a task
- Throughput
  - Total amount of work completed in a given time

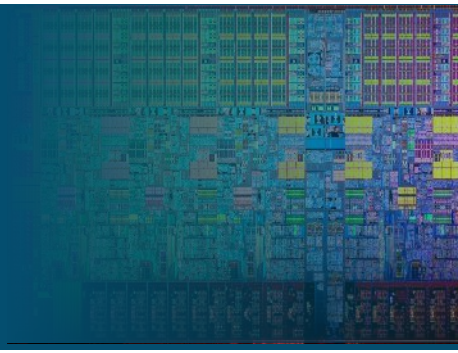
- What is the relationship between response time and throughput?

Computer Performance: Time



- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?
- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

Improving Response Time and Throughput



- How do the following affect response time and throughput?

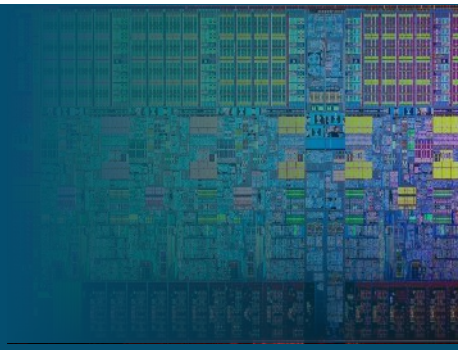
	Response Time	Throughput
Replace processor with faster processor		
Add additional processors to system that uses multiple processes for separate tasks		

Improving Response Time and Throughput

- How do the following affect response time and throughput?

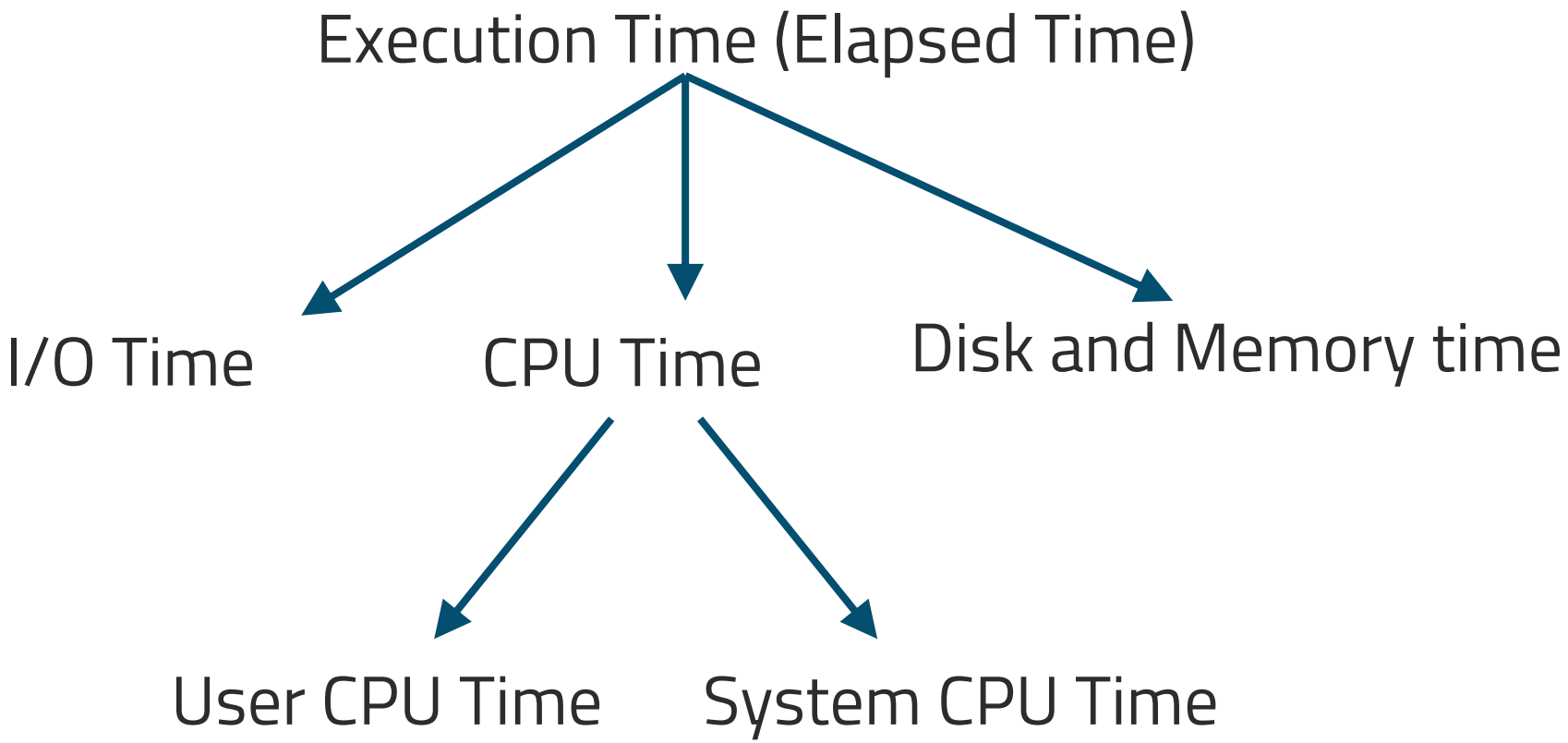
	Response Time	Throughput
Replace processor with faster processor		
Add additional processors to system that uses multiple processes for separate tasks		

Execution Time

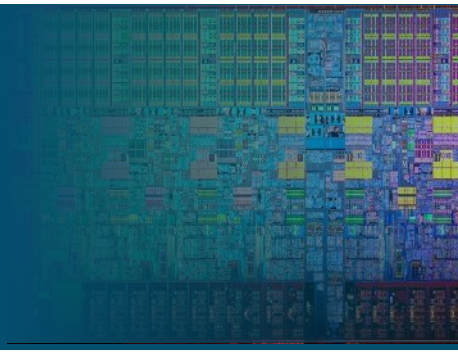


- Elapsed Time
  - Counts everything (disk and memory accesses, I/O , etc.)
  - A useful number, but often not good for comparison purposes
- CPU time
  - Doesn't count I/O or time spent running other programs
  - Can be broken up into system time, and user time
- Our focus: User CPU time
  - Time spent executing the lines of code that are "in" our program

Understanding Execution Time

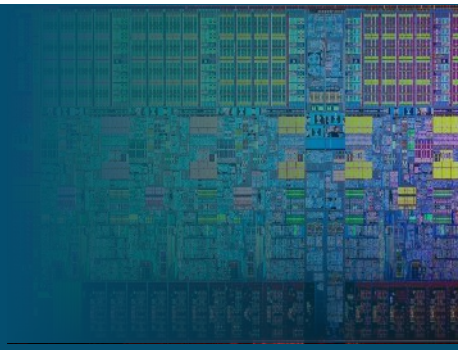


One Definition of (Machine) Performance



- For some program running on machine X,
  $Performance_X = 1 / Execution\_Time_X$
- "X is n times faster than Y"
  $Performance_X / Performance_Y = n$
- Example: time taken to run a program
  - 10s on A, 15s on B
  - $Execution\_Time_B / Execution\_Time_A$ = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

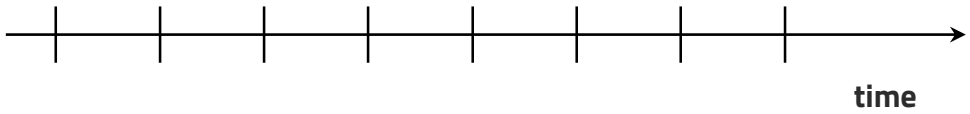
Clock Cycles



- Instead of reporting execution time in seconds, we often use cycles

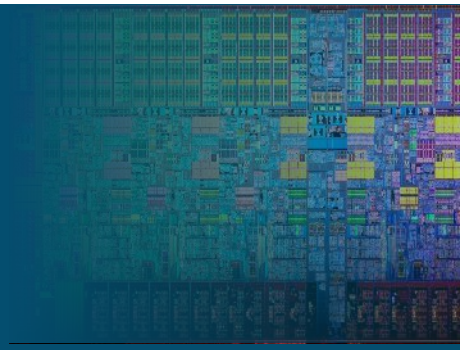
$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Clock "ticks" indicate when to start activities (one abstraction):



- Cycle time = time between ticks = seconds per cycle
- Clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)
- A 4 Ghz. clock has a $\frac{1}{4\times10^{9}} \times 10^{12} = 250\,\text{picoseconds (ps)}$ cycle time

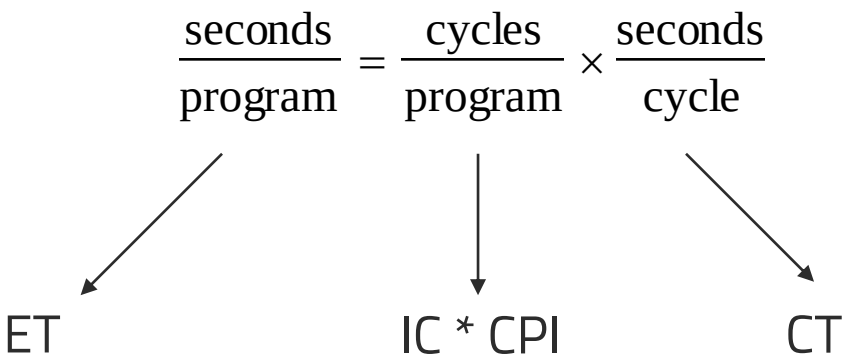
Improving Performance



$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- So, to improve performance (everything else being equal) you can either (increase or decrease?)
  - The # of required cycles for a program,
  - The clock cycle time, or said another way, the clock rate.

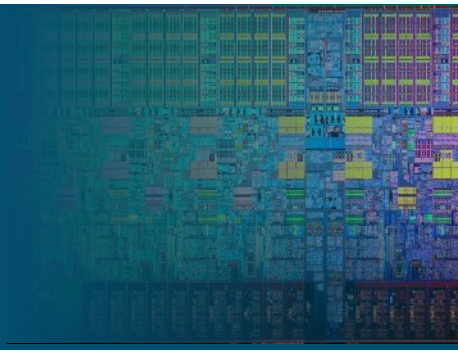
Relating Performance Quantities



$$ET = IC \times CPI \times CT$$

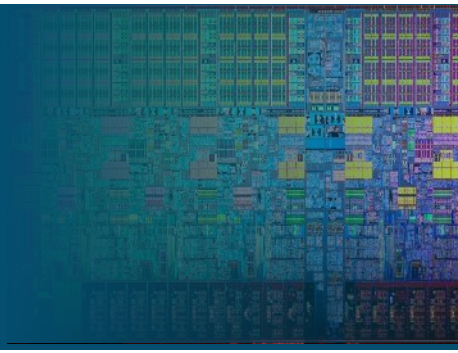
ET	Execution Time
CPI	Cycles Per Instruction
IC	Instruction Count
CT	Cycle Time

Comparing Machines



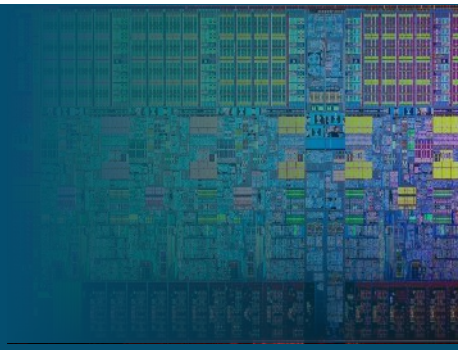
- If two machines have the same ISA, which of our quantities will always be identical?
  - Clock rate?
  - CPI?
  - Execution time?
  - # of instructions?
  - MIPS?

Example



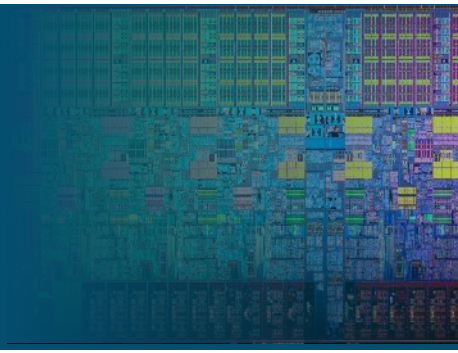
- Our favorite program runs in 10 seconds on computer A, which has a 4 GHz clock.
- We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds.
- The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate
  - This increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program.
- What clock rate should we tell the designer to target?

Now That We Understand Cycles…



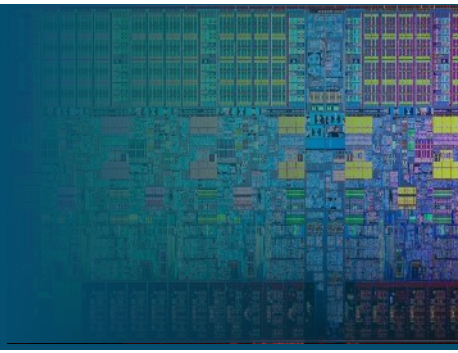
- A given program will require
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds
- We have a vocabulary that relates these quantities:
  - Cycle Time (seconds per cycle)
  - Clock Rate (cycles per second)
  - CPI (Cycles Per Instruction)
    - A floating point intensive application might have a higher CPI
  - MIPS (Millions of Instructions Per Second)
    - This is higher for a program using simple instructions

Performance



- Performance is determined by execution time
- Do any of the other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

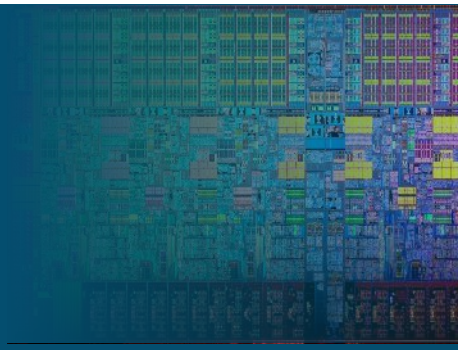
CPI Example



- Suppose we have two implementations of the same instruction set architecture (ISA), Machine A and Machine B. For some program:
  - Machine A has a clock cycle time of 250 ps and a CPI of 2.0
  - Machine B has a clock cycle time of 500 ps and a CPI of 1.2
- Which machine is faster for this program, and by how much?

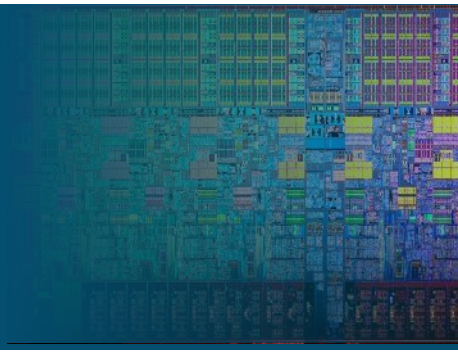
10^{-3} = milli, 10^{-6} = micro, 10^{-9} = nano, 10^{-12} = pico, 10^{-15} = femto

#Instructions Example



- A compiler designer is trying to decide between two code sequences for a particular machine.
- Based on the hardware implementation, there are three different classes of instructions: A (1 cycle), B (2 cycles), and C (3 cycles)
  - The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C
  - The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C.
- Which sequence will be faster? How much?
- What is the CPI for each sequence?

MIPS Example



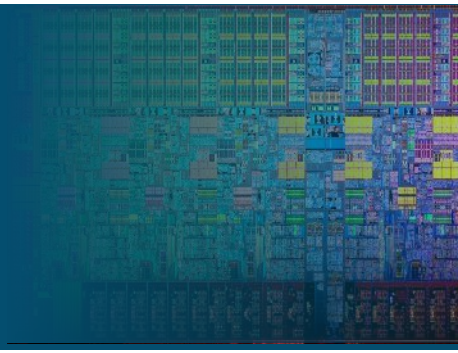
- Two different compilers are being tested for a 4 GHz machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

  The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

  The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- Which sequence will be faster according to MIPS?
- Which sequence will be faster according to execution time?

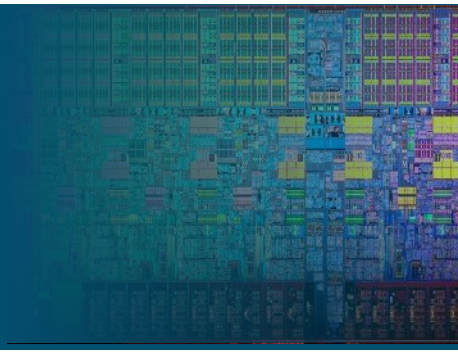
For Multithreaded Programs



- Should we use execution time, throughput, or both?
- IPC is not accurate here
  - Small timing variations may lead to different execution time
  - Order at which threads enter critical section may vary
  - Different interrupt timing may lead to different scheduling decisions

The total number of instructions executed may be different across different runs!

For Multithreaded Programs

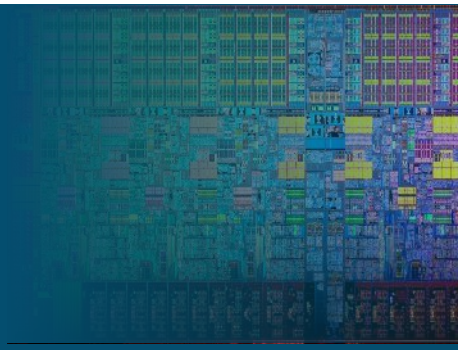


1. The total number of instructions executed may be different across different runs!

This effect increases with the number of cores

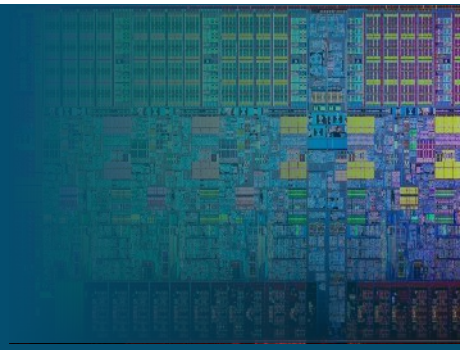
2. System-level code account for a significant fraction of the total execution time.

Your Program Does Not Run in A Vacuum



- At the very least, system software is present
- Multi-programming setting is very common in multicore settings
- Independent programs affect each other performance
  - Why?

Multiprogramming Metrics: Throughput



Normalized progress of program i

$$NP_i = \frac{T_i^{SP}}{T_i^{MP}},$$

T_i^{SP}: Time when running in isolation

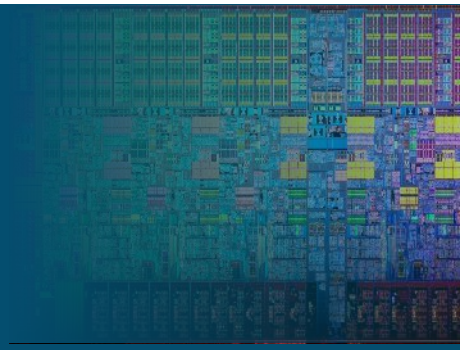
T_i^{MP}: Time when running with other programs

System throughput

$$STP = \sum_{i=1}^{n} NP_i = \sum_{i=1}^{n} \frac{T_i^{SP}}{T_i^{MP}}$$

Higher-is-better metric

Multiprogramming Metrics: Turnaround Time



Normalized Turnaround time of program i

$$NTT_i = \frac{T_i^{MP}}{T_i^{SP}}$$

T_i^{MP}: Time when running with other programs

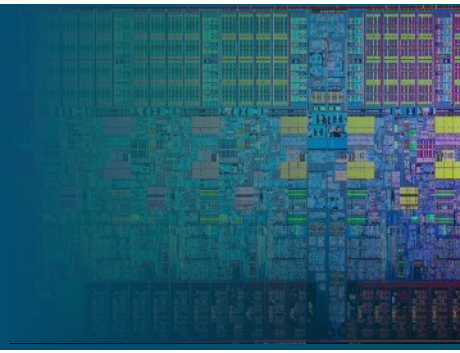
T_i^{SP}: Time when running in isolation

Average normalized turnaround time

$$ANTT = \frac{1}{n}\sum_{i=1}^{n} NTT_i = \frac{1}{n}\sum_{i=1}^{n} \frac{T_i^{MP}}{T_i^{SP}},$$

Lower-is-better metric

Other Metrics

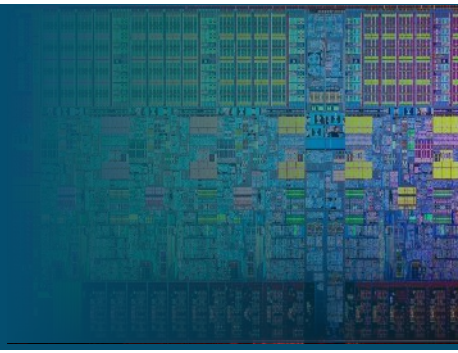


$$IPC\_throughput = \sum_{i=1}^{n} IPC_i$$

$$weighted\_speedup = \sum_{i=1}^{n} \frac{IPC_i^{MP}}{IPC_i^{SP}}$$

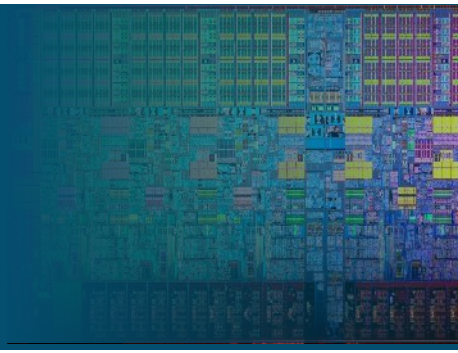
$$hmean = \frac{n}{\sum_{i=1}^{n} \frac{IPC_i^{SP}}{IPC_i^{MP}}}$$

Harmonic vs. Arithmetic Average



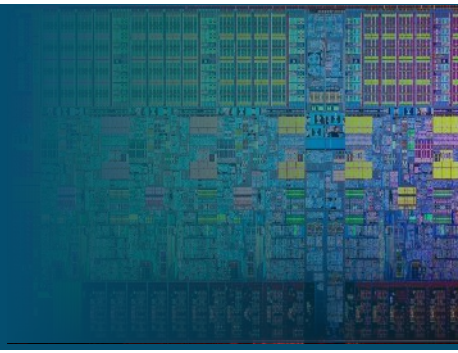
- Both used to compute an average (i.e. combine several measures) of a metric.
- Assume the metric is computed A/B
  - If A is weighted equally among all the benchmarks → then harmonic mean is meaningful.
  - If B → arithmetic mean
- Example: Suppose we gathered IPC of several benchmarks and want to combine them
  - If we execute all benchmarks for the same amount of instructions (e.g. 1 billion instructions) → harmonic (hmean)
  - If we execute all the benchmarks for the same amount of cycles → arithmetic mean

Multiprogramming vs. Multithreading



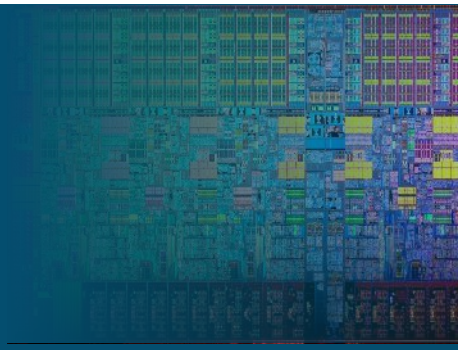
- Can we use the same metrics for multithreading that we apply to multiprogramming?

Benchmarks



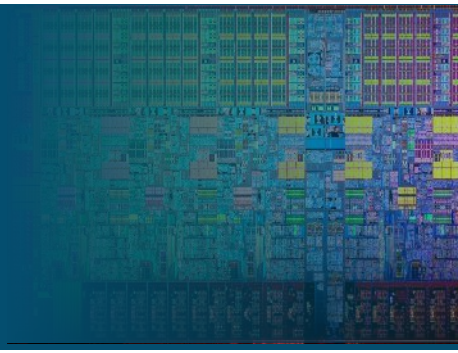
- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications
  - e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
  - Nice for architects and designers
  - Easy to standardize
- Parallel Benchmarks: PARSEC, Rodinia, SPLASH-2
- SPEC (System Performance Evaluation Cooperative)
  - Companies have agreed on a set of real program and inputs
  - Valuable indicator of performance (and compiler technology)

Why Benchmarks?



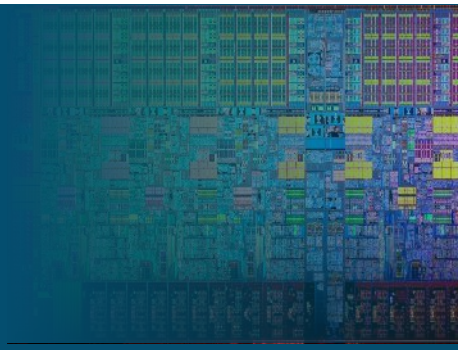
- Help designer explore architectural designs
- Identify bottlenecks
- Compare different systems
- Conduct performance prediction

Example: PARSEC



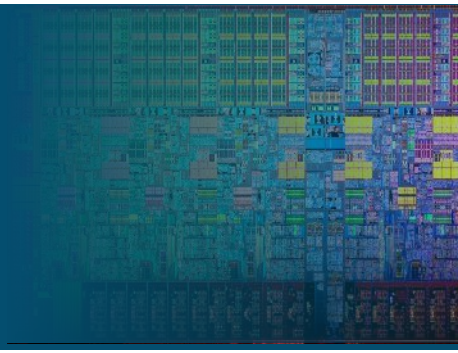
- PARSEC: Princeton Application Repository for Shared-Memory Computers
- Benchmark Suite for Chip-Multiprocessors
- Freely available at: http://parsec.cs.princeton.edu/
- Objectives:
  - Multithreaded Applications: Future programs must run on multiprocessors
  - Emerging Workloads: Increasing CPU performance enables new applications
  - Diverse: Multiprocessors are being used for more and more tasks
  - State-of-Art Techniques: Algorithms and programming techniques evolve rapidly

Example: PARSEC



Program	Application Domain	Parallelization
Blackscholes	Financial Analysis	Data-Parallel
Bodytrack	Computer Vision	Data-Parallel
Canneal	Engineering	Unstructured
Dedup	Enterprise Storage	Pipeline
Facesim	Animation	Data-Parallel
Ferret	Similarity Search	Pipeline
Fluidanimate	Animation	Data-Parallel
Freqmine	Data Mining	Data-Parallel
Streamcluster	Data Mining	Data-Parallel
Swaptions	Financial Analysis	Data-Parallel
Vips	Media Processing	Data-Parallel
x264	Video Encoding	Pipeline

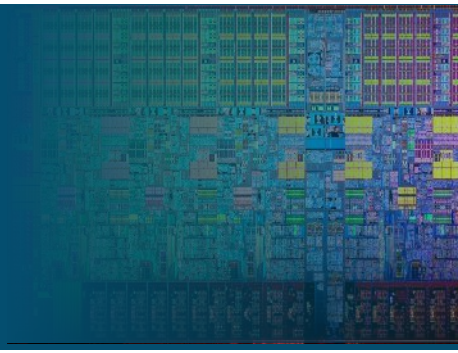
Example: Rodinia



- "A Benchmark Suite for Heterogeneous Computing: Multicore CPU and GPU"
- University of Virginia

Program	Application Domain	Parallelization
K-Means	Data Mining	Dense Linear Algebra
Needleman-Wunsch	Bioinformatics	Dynamic Programming
HotSpot	Physics Simulation	Structured Grid
Backpropagation	Pattern Recognition	Unstructured Grid
SRAD	Image Processing	Structured Grid
Leukocyte Tracking	Medical Imaging	Structured Grid
Breadth-First Search	Graph Algorithms	Graph Traversal
Stream Cluster	Data Mining	Dense Linear Algebra
Similarity Scores	Web Mining	MapReduce

Bottlenecks in Multithreaded Applications

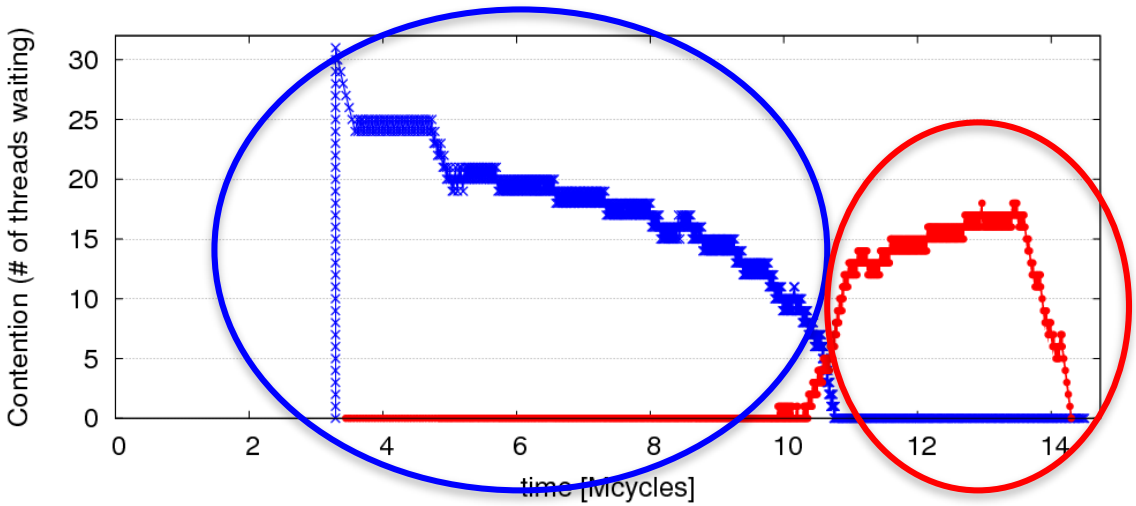


- What is a bottleneck?
- Any code segment for which threads contend (i.e. wait)
- Examples:
  - Amdahl's serial portions
    - Only one thread exists → on the critical path
  - Critical sections
    - Ensure mutual exclusion → likely to be on the critical path if contended
  - Barriers
    - Ensure all threads reach a point before continuing → the latest thread arriving is on the critical path
  - Pipeline stages
    - Different stages of a loop iteration may execute on different threads, slowest stage makes other stages wait → on the critical path

Observation: Limiting Bottlenecks Change Over Time

```
A = full linked list
B = empty linked list
do {
    A.lock();
    Traverse list A
    Remove X from A
    A.unlock();
    Compute on X
    B.lock();
    Traverse list B
    Insert X into B
    B.unlock();
} while (!A.empty());
```

Contention over time with 32 threads:

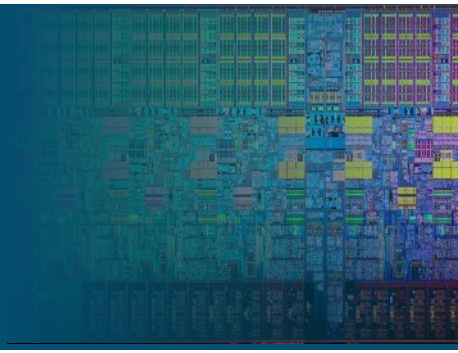


Lock A is bottleneck

Lock B is bottleneck

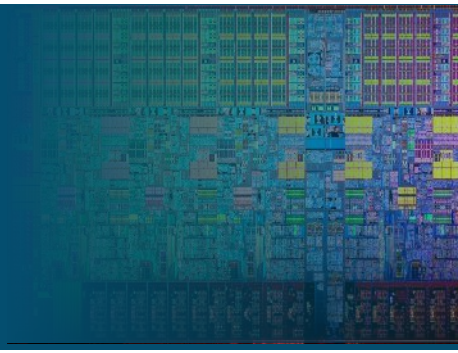
Bottleneck Identification and Scheduling in Multithreaded Applications … José A. Joao, M. Aater Suleman, Onur Mutlu, Yale N. Patt

Profiling Tools



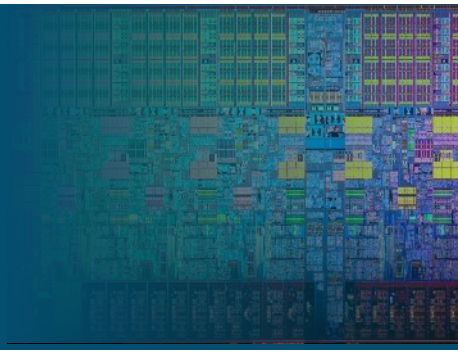
- Valgrind/Helgrind: http://valgrind.org/info/tools.html
- http://oprofile.sourceforge.net/about/
- http://www.rotateright.com/

Conclusions



- Performance evaluation is very important to assess programming quality as well as the underlying architecture and how they interact.
- The following capture some aspects of the system but do not represent overall performance: MIPS, #instructions, #cycles, frequency
- Execution time is what matters: system time, CPU time, I/O and memory time
- For parallel applications: system throughput and average normalized turn-around time are good measures.
- IPC (or CPI) is not a good measure for multithreaded applications

Presentation Discussion



- Presentations about research papers in last 1 or 2 classes
- Format and logistics
  - Groups of 1 or 2
  - Deliverables
    - 10-15 minutes in-class presentation, with or without slides
    - Minimum 1-page summary of the paper and its contributions
  - 20% of final grade
- Discuss relevant recent research on multithreaded or multicore programming, debugging, optimization, or hardware
  1. Summarize one paper that you pick
  2. Explain (in your own words) its novelty and contribution, in the context of its related work if necessary
- Proposal due by next class (November 17th)
  - Email me your paper and 2-3 sentences explaining what is novel and relevant about the paper