Lecture 8: Performance Evaluation

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
What Are We trying to do?

- Measure, Report, and Summarize Performance
- Make intelligent choices

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?
Let’s Start with Two Simple Metrics

• **Response time (aka Execution Time)**
  - The time between the start and completion of a task

• **Throughput**
  - Total amount of work done in a given time

What is the relationship between execution time and throughput?
Computer Performance: TIME, TIME, 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?
Try to solve this...

Do the following changes to the computer system increase throughput, decrease response time, or both?

- Replacing the processor with a faster version
- Adding additional processors to a system that uses multiple processors 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
Execution Time (Elapsed Time)

- I/O Time
- CPU Time
  - User CPU Time
  - System CPU Time
- Disk and Memory time
For some program running on machine X,

\[
\text{Performance}_X = \frac{1}{\text{Execution time}_X}
\]

"X is n times faster than Y"

\[
\frac{\text{Performance}_X}{\text{Performance}_Y} = n
\]

Example: time taken to run a program

- 10s on A, 15s on B
- \(\frac{\text{Execution Time}_B}{\text{Execution Time}_A} = \frac{15s}{10s} = 1.5\)
- 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 cycle time

\[
\frac{1}{4 \times 10^9} = 250 \text{ picoseconds (ps)}
\]
How to Improve 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.
ET = IC * CPI * CT

ET = Execution Time
CPI = Cycles Per Instruction
IC = Instruction Count
An Interesting Question

• If two machines have the same ISA which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?
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, but has informed us that 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?“
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 would be 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).

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
What machine is faster for this program, and by how much?

\[
\begin{align*}
&10^{-3} = \text{milli}, \quad 10^{-6} = \text{micro}, \quad 10^{-9} = \text{nano}, \quad 10^{-12} = \text{pico}, \quad 10^{-15} = \text{femto} \\
\end{align*}
\]
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: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).

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?
Your Program Does Not Run in A Vacuum

- System software at least is there
- Multi-programming setting is very common in multicore settings
- Independent programs affect each other performance (why?)
Some Metrics About Multiprogramming

Normalized progress of program $i$

\[ NP_i = \frac{T_{i}^{SP}}{T_{i}^{MP}}, \]

Time when running in isolation

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
Some Metrics About Multiprogramming

Normalized Turnaround time of program $i$

$NTT_i = \frac{T_{iMP}}{T_{iSP}}$

Time when running with other programs

Time when running in isolation

Average normalized turnaround time

$ANTT = \frac{1}{n} \sum_{i=1}^{n} NTT_i = \frac{1}{n} \sum_{i=1}^{n} \frac{T_{iMP}}{T_{iSP}}$

Lower-is-better metric
Other Metrics

\[ \text{IPC\_throughput} = \sum_{i=1}^{n} \text{IPC}_i \]

\[ \text{weighted\_speedup} = \sum_{i=1}^{n} \frac{\text{IPC}_i^{MP}}{\text{IPC}_i^{SP}} \]

\[ hmean = \frac{n}{\sum_{i=1}^{n} \frac{\text{IPC}_i^{SP}}{\text{IPC}_i^{MP}}} \]
Harmonic Vs Arithmetic

• 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 $\rightarrow$ then harmonic mean is meaningful.
  – If $B \rightarrow$ 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) $\rightarrow$ hmean
  – If we execute all the benchmarks for the same amount of cycles $\rightarrow$ arithmetic mean
What we saw for multiprogramming, how about multithreading?

The next few slides discuss performance measurement for multithreaded programs.
Speedup

- Number of cores = $p$
- Serial run-time = $T_{\text{serial}}$
- Parallel run-time = $T_{\text{parallel}}$

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$
Example

\[ S_p = \frac{100}{25} = 4.0, \]

Perfect parallelization!
Do it ever occur?

\[ S_p = \frac{100}{35} = 2.85, \]

perfect load balancing
Example (cont.)

\[ S_p = \frac{100}{40} = 2.5, \]  
load imbalance

\[ S_p = \frac{100}{50} = 2.0, \]  
load imbalance and sync cost

closest to real life parallel programs
# A Glimpse at the Top 500 (Nov 2018 List)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Cores</th>
<th>Rmax [TFlop/s]</th>
<th>Rpeak [TFlop/s]</th>
<th>Power [kW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE/SC/Oak Ridge National Laboratory (/site/48553) United States</td>
<td><strong>Summit</strong> - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband (/system/179397) IBM</td>
<td>2,397,824</td>
<td>143,500.0</td>
<td>200,794.9</td>
<td>9,783</td>
</tr>
<tr>
<td>2</td>
<td>DOE/NNSA/LLNL (/site/49763) United States</td>
<td><strong>Sierra</strong> - IBM Power System S922LC, IBM POWER9 22C 3.16GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband (/system/179398) IBM / NVIDIA / Mellanox</td>
<td>1,572,480</td>
<td>94,640.0</td>
<td>125,712.0</td>
<td>7,438</td>
</tr>
<tr>
<td>3</td>
<td>National Supercomputing Center in Wuxi (/site/50623) China</td>
<td><strong>Sunway TaihuLight</strong> - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway (/system/178764) NRCPC</td>
<td>10,649,600</td>
<td>93,014.6</td>
<td>125,435.9</td>
<td>15,371</td>
</tr>
<tr>
<td>4</td>
<td>National Super Computer Center in Guangzhou (/site/50365) China</td>
<td><strong>Tianhe-2A</strong> - TH-IVB- FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 (/system/177999) NUDT</td>
<td>4,981,760</td>
<td>61,444.5</td>
<td>100,678.7</td>
<td>18,482</td>
</tr>
</tbody>
</table>

Rmax: Maximal achieved  
Rpeak: Theoretical peak
Sources of Parallel Overheads

- Overhead of creating threads/processes
- Synchronization
- Load imbalance
- Communication
- Extra computation
- Memory access (for both sequential and parallel!)
Efficiency of a parallel program

\[ E = \frac{S}{p} = \frac{T_{\text{serial}}}{T_{\text{parallel}}} \]
Be Careful about $T$

- Both $T_{seq}$ and $T_{par}$ are wall-clock times, and as such they are not objective. They can be influenced by:
  - The skill of the programmer who wrote the implementations
  - The choice of compiler (e.g. GNU C++ versus Intel C++)
  - The compiler switches (e.g. turning optimization on/off)
  - The operating system
  - The type of filesystem holding the input data (e.g. EXT4, NTFS, etc.)
  - The time of day... (different workloads, network traffic, etc.)
Scalability

- **Scalability** is the ability of a (software or hardware) system to handle a growing amount of work efficiently.
- If we keep the efficiency fixed by increasing the number of processes/threads and without increasing problem size, the problem is **strongly scalable**.
- If we keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, the problem is **weakly scalable**.
Execution Time vs Throughput

• Shall we use execution time or 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!
Execution Time vs Throughput

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

This effect increases with the number of cores

System-level code account for a significant fraction of the total execution time
Benchmarks
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)
Role of Benchmarks

- help designer explore architectural designs
- identify bottlenecks
- compare different systems
- conduct performance prediction
Example: 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

<table>
<thead>
<tr>
<th>Program</th>
<th>Application Domain</th>
<th>Parallelization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackscholes</td>
<td>Financial Analysis</td>
<td>Data-parallel</td>
</tr>
<tr>
<td>Bodytrack</td>
<td>Computer Vision</td>
<td>Data-parallel</td>
</tr>
<tr>
<td>Canneal</td>
<td>Engineering</td>
<td>Unstructured</td>
</tr>
<tr>
<td>Dedup</td>
<td>Enterprise Storage</td>
<td>Pipeline</td>
</tr>
<tr>
<td>Facesim</td>
<td>Animation</td>
<td>Data-parallel</td>
</tr>
<tr>
<td>Ferret</td>
<td>Similarity Search</td>
<td>Pipeline</td>
</tr>
<tr>
<td>Fluidanimate</td>
<td>Animation</td>
<td>Data-parallel</td>
</tr>
<tr>
<td>Freqmine</td>
<td>Data Mining</td>
<td>Data-parallel</td>
</tr>
<tr>
<td>Streamcluster</td>
<td>Data Mining</td>
<td>Data-parallel</td>
</tr>
<tr>
<td>Swaptions</td>
<td>Financial Analysis</td>
<td>Data-parallel</td>
</tr>
<tr>
<td>Vips</td>
<td>Media Processing</td>
<td>Data-parallel</td>
</tr>
<tr>
<td>X264</td>
<td>Media Processing</td>
<td>Pipeline</td>
</tr>
</tbody>
</table>
Example: Rodinia

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

<table>
<thead>
<tr>
<th>Application / Kernel</th>
<th>Dwarf</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-means</td>
<td>Dense Linear Algebra</td>
<td>Data Mining</td>
</tr>
<tr>
<td>Needleman-Wunsch</td>
<td>Dynamic Programming</td>
<td>Bioinformatics</td>
</tr>
<tr>
<td>HotSpot*</td>
<td>Structured Grid</td>
<td>Physics Simulation</td>
</tr>
<tr>
<td>Back Propagation*</td>
<td>Unstructured Grid</td>
<td>Pattern Recognition</td>
</tr>
<tr>
<td>SRAD</td>
<td>Structured Grid</td>
<td>Image Processing</td>
</tr>
<tr>
<td>Leukocyte Tracking</td>
<td>Structured Grid</td>
<td>Medical Imaging</td>
</tr>
<tr>
<td>Breadth-First Search*</td>
<td>Graph Traversal</td>
<td>Graph Algorithms</td>
</tr>
<tr>
<td>Stream Cluster*</td>
<td>Dense Linear Algebra</td>
<td>Data Mining</td>
</tr>
<tr>
<td>Similarity Scores*</td>
<td>MapReduce</td>
<td>Web Mining</td>
</tr>
</tbody>
</table>
Conclusions

• Performance evaluation is very important to assess programming quality as well as the underlying architecture and how they interact.

• The following metrics capture some aspects of the system but do not represent overall performance: MIPS, #instructions, #cycles, frequency

• Scalability is also an important aspect for parallel programs in addition to speedup.

• IPC (or CPI) is not a good measure for multithreaded applications