Announcements

Assignment #0 due today
Trying to change the lecture room

Reader 1 handed out today
• Why supplementary papers?
Pipelining ( theirs)

- latch
- pipeline stage

- IF
- ID
- EX
- ME
- WB

Clock

Pipelining (Cont.)

Up to one result per clock cycle
  - e.g., lunch buffet, automotive manufacturing, etc.

But, performance may be limited
  - unequal segments
  - latch overhead
  - dependence
Memory Hierarchy

Principles of performance
- fast common case
- locality in accessing data and instructions
  - temporal locality — accesses to same data in near future
  - spatial locality — accesses to nearby data

Implementation facts
- on-chip faster than off-chip
- SRAM faster than DRAM faster than disk

Keep recently referenced data close to the processor

---

Memory Hierarchy

Registers
L1 Cache
L2 Cache
Main Memory
Disk
Memory Hierarchy (Cont.)

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Speed</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>&lt; 1KB</td>
<td>1-5 ns</td>
<td>9,600 MB/s</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>&lt; 256 KB</td>
<td>10 ns</td>
<td>3,200 MB/s</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>&lt; 8 MB</td>
<td>30 ns</td>
<td>800 MB/s</td>
</tr>
<tr>
<td>Memory</td>
<td>&lt; 4 GB</td>
<td>100 ns</td>
<td>133 MB/s</td>
</tr>
<tr>
<td>Disk</td>
<td>&gt; 1 GB</td>
<td>20 ms</td>
<td>4 MB/s</td>
</tr>
</tbody>
</table>

Performance & Cost

Performance metrics
- elapsed time vs CPU time

Iron law of performance

Benchmarks and benchmarking

Averaging

Amdahl’s law

Balance and bursty behavior

Cost and price
Performance Metrics

Time (latency)
- elapsed time vs processor time
Rate (bandwidth)
- performance = rate = work per time

Distinction is sometimes blurred

MIPS

MIPS = instruction count/(execution time x 10^6)
= clock rate/(CPI x 10^6)

Need a uniform measure of work
- instruction sets are not equivalent
- different programs use a different instruction mix
- instruction count is not a reliable indicator of work
  - some optimizations add instructions
  - instructions have varying work
Relative MIPS

Relative MIPS = \(\frac{\text{time}_{\text{reference}}}{\text{time}_{\text{new}}} \times \text{MIPS}_{\text{reference}}\)

Relative MIPS (e.g., Pentium MIPS) a bit better

But, very sensitive to reference machine!

How do compilers and OS compare?

Bottomline: maybe useful if same instruction set/compiler/OS

MFLOPS

MFLOPS = (FP ops/program) \times (\text{program/time}) \times 10^6

Not great because it

- ignores other instructions (e.g., load/store)
- depends on how FP-intensive program is
  - in the extreme some programs have no FP ops

Beware of peak MFLOPS!
Normalized MFLOPS (🅀)

Normalized FP: give cannonical # FP ops to prog

Normalized MFLOPS = (# cannonical FP ops/time) x 10^6

Not all machines have the same FP ops
- Cray does not implement divide
- Motorola has SQRT, SIN, and COS

Not all FP ops are same work
- adds usually faster than divide

Iron Law (🪤)

Time/prog = insts/prog x cycles/inst x sec/cycle

sec/cycle (a.k.a. cycle time, clock time)
- mostly determined by technology and CPU organization

cycles/inst (a.k.a. CPI)
- mostly determined by ISA and CPU organization
- overlap among instructions makes this smaller

insts/prog (a.k.a. instruction count)
- instructions executed NOT static code
- mostly determined by program, compiler, ISA
Controversial Example

Some have argued:
- CISC CPU time = $P \times 8 \times T = 8PT$
- RISC CPU time = $2P \times 2 \times T = 4 PT$
- RISC CPU time = CISC CPU time/2

The truth is much more complex!

Performance Comparison

Machine A is $n$ times faster than machine B iff
- $\frac{\text{perf}(A)}{\text{perf}(B)} = \frac{\text{time}(B)}{\text{time}(A)} = n$

Machine A is $x\%$ faster than machine B iff
- $\frac{\text{perf}(A)}{\text{perf}(B)} = \frac{\text{time}(B)}{\text{time}(A)} = 1 + \frac{x}{100}$

E.g., A 10s, B 15s
- $15/10 = 1.5 \Rightarrow A$ is 1.5 times faster than B
- $15/10 = 1 + 50/100 \Rightarrow A$ is 50% faster than B
Simple Example

<table>
<thead>
<tr>
<th>Op</th>
<th>Frequency</th>
<th>Cycle count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>43%</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>21%</td>
<td>1</td>
</tr>
<tr>
<td>Store</td>
<td>12%</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>24%</td>
<td>2</td>
</tr>
</tbody>
</table>

Assume stores can execute in 1 cycle by slowing clock 15%
Should this be implemented?

Simple Example (Cont.)

Old CPI = 0.43 + 0.21 x 2 + 0.12 x 2 + 0.24 x 2 = 1.36
New CPI = 0.43 + 0.21 + 0.12 + 0.24 x 2 = 1.24

Speedup = old time/new time
= (P x old CPI x T)/(P x new CPI x 1.15T)
= 1.36/1.24x1.15 = 0.97

Answer: don’t make the change!
Type of Benchmarks

Real programs
- representative of real workload
- only accurate way to characterize performance
- requires considerable work

Kernels
- “representative” program fragments
- good for focusing on individual features not big picture

Mixes
- instruction frequency of occurrence; calculate

Other Benchmarks

Toy benchmarks
- e.g., fibonacci, prime number
- little number

Synthetic benchmarks
- programs intended to give specific mix
- ignore dependences
- maybe ok for non-pipelined, non-cached, w/o optimizing compilers
**SPEC95 Benchmarks**

**Integer**
- go: plays a game of go
- m88ksim: simulates Motorola 88000 CPU
- gcc: GNU C compiler
- compress: data compress/decompress
- li: lisp interpreter
- jpeg: jpeg compression/decompression
- perl: perl script interpreter
- vortex: object-oriented database system

**SPEC95 Benchmarks**

**Floating point**
- tomcatv: vectorized mesh generator
- swim: shallow water model — finite difference
- su2cor: quantum physics
- hydro2d: galactic jets — navier stokes
- mgrid: multigrid solver for 3d field
- applu: partial differential equations
- turb3d: simulation of turbulence in a cube
- apsi: temperature and wind velocity
- fpppp: quantum chemistry
- wave5: n-body Maxwell’s

Caveat: Only useful if they match your workload!
Kernel Example

Inner product
DO L = 1, LP
  Q = 0.0
DO K = 1, N
  Q = Q + Z(K)*X(K)

Synthetic Benchmark Example

Dhrystone, Whetstone

X = 1.0
Y = 1.0
Z = 1.0
DO I = 1, N8
  CALL P3(X,Y,Z)

SUBROUTINE P3(X,Y,Z)
  X1 = X
  Y1 = Y
  X1 = T * (X1 - Y1)
  Y1 = T * (X1 + Y1)
  Z = (X1 + Y1)/T2
RETURN
**Synthetic Benchmark Example**

Dhrystone, Whetstone

\[ X = 1.0 \]
\[ Y = 1.0 \]
\[ Z = 1.0 \]

DO I = 1, N8

CALL P3(X,Y,Z)

**Mix Example**

Gibson Mix, developed in 1950’s at IBM

- load/store 31% branch 17%
- fixed add/sub 6% compare 4%
- float add/sub 7% float mult 4%
- float div 2% fixed mult 1%
- fixed div < 1% shifts 4%
- logical 2%

Generally speaking, these numbers are still valid today
SPEC Benchmarking Process

Steps:
- for each benchmark i, look up \( T_{\text{base},i} \)
- for each benchmark i, run target machine to get \( T_{\text{new},i} \)

- compute geometric mean: \[ \sqrt[n]{\prod_{i=1}^{n} \frac{T_{\text{base},i}}{T_{\text{new},i}}} \]

But,
- what workload does this match?
- how does geometric mean predict performance?

SPEC Benchmarking Process

Steps:
- extract benchmarks from applications
- choose performance metric
- execute benchmarks on candidate machines
- project performance in new machine