CSCE 4610: Computer Architecture

Homework #1  1.3, 1.7, 1.10, 1.14
Due: Jan 31, 2017

Review
Cost of a chip
Yield

Dies per wafer: \[
\frac{\pi \times ((\text{wafer\_diameter})/2)^2}{\text{chip\_area}} - \frac{\pi \times (\text{wafer\_diameter})}{\sqrt{2} \times \text{chip\_area}}
\]

Die yield= \[
\frac{1}{[1 + (\text{defects\_per\_unit\_area})^{\text{die\_area}}]}
\]

Power consumed: static and dynamic power
Energy = power * execution time
Energy x Delay product

\[
\text{Power}_{\text{dynamic}} = (1/2) \times (\text{Capacitive load}) \times (\text{threshold voltage})^{2} \times (\text{operating frequency})
\]

How do we define the performance of a processor?
Execution time for a program?
Wall clock or CPU time?
User CPU and System CPU Time

For now we will only use user CPU time = \[
\frac{\text{(instruction count)} \times (\text{CPI})}{\text{(Clock Rate)}}
\]

Note "cycle time" = 1/clock_rate. 1 Ghz clock means 1 ns per cycle
CPI: Average number of Cycles Per Instruction.
MIPS rating: (clock-rate)/CPI
MIPS rating can be misleading

Note the CPI may be impacted by many other aspects
For example consider a branch instruction.
CSCE 4610: Computer Architecture

If assuming a pipelined execution of instructions

Assume each stage of a pipeline takes 1 cycle

Once the pipeline is full, we can complete a new instruction every cycle
or CPI = 1 cycle

But what happens when we have a branch instruction?

We do not know what is the next instruction to fetch

Fetch must wait until branch decision is made – in EX stage

We may lose 2 or 3 cycles waiting for the decision

Those are often called pipeline bubbles or stalls

Cache Misses

Effect only load and store instructions

If no cache miss, say CPI = 2

If cache miss, we may have a CPI of 50

5% miss rate leads to 0.95*2+0.05*50 = 4.4

Remember the instruction frequencies from a previous example
CSCE 4610: Computer Architecture

ALU operations occur 43% of time and take 1 clock cycle to execute
Load instructions occur 21% of the time and need 2 cycles without cache misses
Store instructions occur 12% of the time and need 2 cycles without cache misses
Branch instructions occur 24% of the time and need 2 cycles

But if have 21% loads and 12% stores with 4.4 cycles with cache misses,
the new CPI is \(33\% \times 4.4 + 43\% \times 1 + 24\% \times 2 = 2.32\) CPI

compared to 1.57 CPI with no cache misses

How to report performance data?
- Execution time for one program
- Execution times for all programs
- Average execution time across all programs
- Weighted average etc.

Arithmetic Mean = \(\frac{1}{n} \sum_{i=1}^{n} (Time)_i\)

Assuming \(n\) programs

Weighted Arithmetic Mean = \(\sum_{i=1}^{n} (Weight)_i \times (Time)_i\)

Harmonic Mean = \(\frac{n}{\sum_{i=1}^{n} \frac{1}{(Time)_i}}\)

Let us look at an example. Here we are comparing execution time on 3 different computers using 2 programs.

<table>
<thead>
<tr>
<th>Program</th>
<th>Computer A</th>
<th>Computer B</th>
<th>Computer C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pgm P1</td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Pgm P2</td>
<td>1000</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>1001</td>
<td>110</td>
<td>40</td>
</tr>
</tbody>
</table>

Let us find weighted arithmetic average execution times and we will use 3 different weights

W1: P1=50%, P2=50%
W2: P1= 90.9%, P2= 9.1%
W3: P1= 99.9%, P2=0.1%
CSCE 4610: Computer Architecture

So which computer is best?

If we use W1, C is best, with W2, B is best and with W3 A is best

Can we think of a different way of computing averages?

Relative performance. For each program use a relative execution time, compared a “standard” computer.

The relative execution times can be used to compute an arithmetic (or weighted) means.

We can also compute Geometric Mean.  

Let us look our example using Geometric means. The relative performance of the 3 machines remain the same

Now, C is always the best
CSCE 4610: Computer Architecture

What programs to use in evaluating performance?

The programs that will be run in the field

Benchmark programs

Real programs that are common in an application domain
e.g. SPEC benchmarks (Spec CPU, Integer, float)
   SPECWeb, SPECvirt
   bio-informatics
   High-performance (SPECmp)
   BigDataBench

Program kernels:
e.g. Embedded kernels (EEMBC)
   NAS benchmarks, Livermore loops

Synthetic program mixes

CSCE 4610: Computer Architecture

How to collect performance data using benchmarks?
Actual Measurements and Simulations

If the architecture already exist, run programs and collect data
   Need to be careful in collecting data
   Instrumentation may skew data
   Performance Registers
   Software profiling techniques

Or develop simulations.
   Detailed simulations
   Trace driven simulations
   Monte Carlo simulations
The performance improvement to be gained from using some faster
mode of execution is limited by the fraction of the time the faster mode
can be used. -- the computer law of diminishing returns.

Example of page 47. We want to improve the performance for web browsing.
We can improve the performance for this functionality by a factor of 10

We are told that the actual computation for web services is 40% of all the time
-- that is we are only improving 40% of the execution time by a factor of 10

\[
\text{Speed up} = \frac{1}{0.6 + \left(\frac{0.4}{10}\right)} = 1.56
\]

Only 56% speed up

Amdahl’s Law. Consider another example (page 47).

25% of all operations are Floating Point; 2% are FPSQR
FPSQR takes 20 cycles (CPI)
All other FP operations take 4 cycles (CPI)
All other operations take 1.33 (CPI)

Two improvements proposed:
(a) Reduce FPSQR to 2 cycles
(b) Reduce all FP operations (except FPSQR) to 2 cycles.

Original CPI = 0.23*4+0.02*20+0.75*1.33 = 2.3175

(a) = 0.23*4+0.02*2+0.75*1.33 = 1.9575
improvement ratio = 1.84

(b) = 0.23*2 +0.02*20+0.75*1.33 =1.8575
improvement ratio = 1.248
Another Example
Consider two processors with different ways of implementing conditional instructions.

CPU-A: needs two instructions; A compare and a branch (e.g., SLT R3, R1, R2; BNZ R3, loop)

CPU-B: A single instruction to compare and branch (e.g., BLT R1, R2, Loop)

Branches take 2 cycles and all other instructions take 1 cycle
Frequency of branches = 20%

CPU-A’s clock is 25% faster – simpler instructions

Time on CPU-A = (Instr_Count)*(0.80*1+0.20*(2+1))*(Cycle_Time)
= (Instr_Count)*1.4*(Cycle_Time)

Time on CPU-B = (Inst_Count)*(0.8*1+0.2*2)*(1.25*Cycle_Time)
= (Instr_Count)*1.5*(Cycle_Time)

CPU-A is faster even if it needs more instructions!

Amdahl’s law was originally used for parallel processing

Let us designate the time to run a program on one core as $T_1$
Likewise the execution time when N cores are used as $T_N$

Speedup = ($T_1$/$T_N$)

The best speedup we can get is N.

But most programs cannot be fully parallelized.
Let us assume that a fraction $f$ of a program cannot be parallelized (serial portion).
What is $T_N$ in this case?

$$T_N = f*T_1 + (1-f)*T_1/N$$

In addition to serial portions of program, parallel program incur other overheads, due to communication, synchronization, etc.
Consider an example of how Amdahl’s law can be applied to parallel programs or multicore systems. See problem 1.18

1.18 [10/20/20/20/25] <1.10> When parallelizing an application, the ideal speedup is speeding up by the number of processors. This is limited by two things: percentage of the application that can be parallelized and the cost of communication. Amdahl’s law takes into account the former but not the latter.

a. [10] <1.10> What is the speedup with N processors if 80% of the application is parallelizable, ignoring the cost of communication?

b. [20] <1.10> What is the speedup with 8 processors if, for every processor added, the communication overhead is 0.5% of the original execution time.

c. [20] <1.10> What is the speedup with 8 processors if, for every time the number of processors is doubled, the communication overhead is increased by 0.5% of the original execution time?

d. [20] <1.10> What is the speedup with N processors if, for every time the number of processors is doubled, the communication overhead is increased by 0.5% of the original execution time?

e. [25] <1.10> Write the general equation that solves this question: What is the number of processors with the highest speedup in an application in which P% of the original execution time is parallelizable, and, for every time the number of processors is doubled, the communication is increased by 0.5% of the original execution time?

Remember speed up = $T_1/T_N$

Ideally, with N processors speed up = N and $T_N = T_1/N$

If only 80% can be parallelized

Execution time on N processors = $T_N = (20\%*T_1)+(80\%*T_1)/N$

Speed up = $1/[(80\%/N)+20\%] = N/(0.8 + 0.2*N)$

If we set N = 100, Speedup = 100/(0.8+20) = 4.8

(not even close to 100)

b) First let us assume that the program is fully parallelizable.

$T_N = T_1/N + N^0.005*0.5\%$

Speed up = $T_1/T_N = N/(1+0.005*N^2)$ and if N = 8, speed up = 8/(1+0.320) = 6.06

Now we also include that 20% which cannot be parallelized

$T_N = 80\%*T_1/N + 20\%* T_1 + T_1* N^0.5\%$

For N = 8, $T_N = 0.8*T_1/8 + 0.2*T_1 + T_1*0.005*8$

Speedup = $1/(0.1+0.2 + 0.04) = 2.94$
Another performance metric that is becoming very important is **Reliability**. This includes **Mean Time To a Failure (MTTF)** and **Mean Time To Repair/Replace (MTTR)**.

If we are looking for availability, we use **Mean Time To Repair (MTTR)**. Let us look at an example of a system (page 34):

- 10 disks each with 1,000,000 hours MTTF
- 1 ATA controller with 500,000 hours MTTF
- 1 power supply with 200,000 hours MTTF
- 1 fan with 200,000 hours MTTF
- 1 ATA cable with 1,000,000 hours MTTF

In general, the weakest link dictates overall reliability. The power supply or the fan is the weakest link in this system.

Assuming that all component failures are **independent**, we can find the system MTTF (page 35). Any one of the components may fail, so the system failure rate is:

\[
\text{System Failure rate} = 10 \times \frac{1}{1,000,000} + \frac{1}{500,000} + \frac{1}{200,000} + \frac{1}{200,000} + \frac{1}{1,000,000}
\]

\[
= 23,000 \text{ failures per } 1,000,000,000 \text{ hours}
\]

The system MTTF is the reciprocal of the failure rate:

\[
\text{System MTTF} = \frac{1}{23,000} \text{ hours} = 43,500 \text{ hour MTTF}
\]

**Reliability can be improved with redundancy**. We can replace with a backup when a component fails. An example using redundant power supply on page 35.

Here we simplified and assume independent failures (and detection capability) of the power supplies. If each power supply has a 200,000 hour MTTF, what is the MTTF of 2 power supplies used as redundant system?

The power supplies fail if both fail or failure rate = \(\frac{1}{200,000}\)^2
CSCE 4610: Computer Architecture

The book looks at a slightly more complex equation

What is the probability that after one power supply fails, that the second also fails before the first power supply is repaired or replaced

Let us assume that the time to replace or repair (MTTR) is 24 hours
A simplified version is shown in the book