CSCE 4610: Solutions To Exam-1

1. (30%) This problem is related to Amdahl’s law. Amdahl’s law says that, “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”. Consider an extension. Suppose we have two modes of a processor that are improved by $s_1$ times and $s_2$ times respectively; and these modes are used $f_1$ and $f_2$ fractions of times respectively.

a) Write an expression to show the overall performance gain (or speedup) due to both improvements.

b) As a concrete example, we speeded up the first mode by a factor 10 and the second mode by a factor 5; the first mode is used 50% of the time and the second mode is used 25% of the time. What is the speedup?

c) Now consider generalizing the above case with $n$ modes of a system that are improved: mode $i$ is speeded up by $s_i$ and used $f_i$ fraction of the time. Write an expression for speedup in this general case.

**Key.**

We will use 1 for the original execution time and compute new execution time with respect to this original execution time.

a). New execution time = $(1-f_1 - f_2) + f_1/s_1 + f_2/s_2$

Speedup = $1/[(1-f_1-f_2) + f_1/s_1 + f_2/s_2]$

b). New execution time = $0.25 + 0.5/10 + 0.25/5 = 0.35$

Speedup = $1/0.35 = 2.857$

c). New execution time = $[(1-f_1-f_2-...-f_n) + (f_1/s_1) + (f_2/s_2) + ... + (f_n/s_n)]$

speedup = $1/(new \ execution \ time)$

2. (20%) Consider a processor with only L1 cache with 32 byte blocks. Assume write-back cache and assume that at any given time 25% of cache lines are dirty (which means that on a cache miss dirty lines must be written back). Assume 2% miss rate to the cache, and on a miss you fetch 32 bytes from memory. Assume processor generates $10^9$ memory accesses per second. If the memory can support $10^9$ bytes per second bandwidth, what fraction of this bandwidth is utilized by the processor?

Hint. You need to find how many bytes per second are requested from memory when data is not found in L1, based on miss rates, number of accesses per second and the probability of dirty lines.

**Key.**

CPU generates $10^9$ memory requests per second, of this 2% are misses. So we will go to memory $0.02*10^9$ times per second. However, 25% of the time we have to write back data.

Thus the actual number of request to memory = $1.25*0.02*10^9 = 25*10^6$.

But on each request we transfer 32 bytes.
So the total number of bytes transferred to/from memory = \(32 \times 25 \times 10^6\) bytes = \(800 \times 10^6 = 0.8 \times 10^9\) bytes.

Memory can support \(1.0 \times 10^9\). So 80% of this bandwidth is used by the CPU, leaving only 20% for I/O request. In most case this is not sufficient for I/O.

3. (30%) Consider the following program in C.

```c
int i, j, b[100], c[100], a[100][100];
for (i=0; i<100; i++)
    c[i]=0.0;
for (j=0; j<100; j=j+1)
    c[i] = a[i][j]*b[j];
```

Let us only worry about memory accesses generated by the arrays \(a, b, c\) and each element of the array is 4-bytes long. We are using a cache with 16-byte blocks. We will also assume that cache is large enough to hold all arrays and there are no conflict misses.

(i) Assuming that the array elements are not initially in cache, what is the expected cache miss rate?

(ii) What would be miss rate if you are using 32 byte cache blocks?

Note, to compute miss rate you need to find number of misses and total number of accesses.

**Key.**

(i) For a, we cause \(100 \times \frac{100}{4} = 2500\) misses
   For \(b\) and \(c\) we cause \(100/4 = 25\) misses each
   Total misses = 2550 misses

Total number of access.

- We access is element of a only once for a total of \(100\times100\) accesses for a
- We access each \(c[i]\) 101 times for a total of \(101\times100\) accesses for \(c\)
- We access each \(b[j]\) 100 times for a total of \(100\times100\) accesses for \(b\)
  - Total accesses = 30,100
  - Miss rate = \(\frac{2550}{30,100} = 0.0847\) or 8.47%

(ii) If we are using 32 byte blocks, each block contains 8 4-byte data items.

- For a, we cause \(100 \times \frac{100}{8} = 1250\) misses
  - Or if we assume that inside \(j\) loop \(100 \text{ a["","[",","] elements are accessed}
  - Causing \(100/8\) or 12.5, rounded to 13 misses
  - Accesses to a cause \(100\times13\) misses = 1300

- For \(b\) and \(c\) we cause \(100/8 = 12.5\) but rounded off to 13 misses each
  - Total misses = 1276 misses or 1326 misses

Total number of accessed do not change.

- Miss rate = \(\frac{1276}{30100} = 0.0424\) or 4.24%
  - Or \(\frac{1326}{30100} = 4.4\%\)

As one would expect, half of the miss rate compared to using 16-byte blocks
4. (20%). As we saw in class, way-prediction can lead to energy savings, but may increase execution times, since on a mis-prediction when using way-prediction requires one additional cycle if the data is in one of the other ways. On the other hand, higher associativity may slow down the clock speed since we have to compare multiple tags.

Consider the following data.

a) The miss rate when using 4-way associative cache is 0.001.
b) When using way-prediction, we have a 90% success and it costs 1 ns. However on a mis-prediction, but if the data is in one of the other ways, it will cost 2 ns.
c) If we use conventional 4-way associative cache, the clock speed will be slower by 10% (compared with way-prediction). So, the hit time is 1.2 ns.
d) Miss penalty is 10 ns (with or without way-prediction).

i). What is average memory access time when using way-prediction?

ii). What is the average memory access time when using conventional 4-way associative cache (with slower clock)?

Key

i). Note on a hit we have two cases: if way-prediction is correct it costs 1 ns; if way-prediction is wrong it costs 2 ns. On a miss (0.1%), we take 10 ns

Average Memory access time = (hit-time) + (miss-rate)*(miss- penalty)  
= 1*[1*0.9+2*0.1] + 0.001*10 = 1.11

ii). Here the clock is slow, but we spend 1.1 ns on hit and 10 ns on miss.

Agerage memory access time = (hit-time) + (miss-rate)*(miss- penalty)  
= 1.1 ns + 0.001*10 = 1.11 ns

NOTE: There is a type in item (c). It states the clock is slowed by 10% but also states that the hit time is 1.2 ns. These two are inconsistent. I gave full credit if you used either 1.1 ns (10% slower clock) or 1.2 ns for hit time.

If you used 1.2 ns for hit, the average memory access for 4-way associative cache without way prediction = 1.2 ns + 0.001*10 = 1.21 ns