University of Western Ontario, Computer Science Department CS3350B, Computer Organization

Assignment 1 **Due: Monday, February 05, 2023**

General Guidelines: This assignment encompasses 14 pages, comprising 5 exercises, and carries a total of 100 marks. In instances requiring calculations, it is imperative to include your step-by-step workings. Final answers devoid of supporting workings will be deemed incorrect. It is mandated that each assignment submission and its associated responses reflect individual effort, completed independently. Any instances of plagiarism will be treated with utmost seriousness and may lead to severe consequences, including but not limited to a score of 0 on this assignment, expulsion from the course, or more severe disciplinary actions.

Submission Protocol: All responses for this assignment must be submitted electronically via Gradescope. Preferably, the answers should be typed. At a minimum, legible scanned copies (excluding photographs) of handwritten work are acceptable. Failure to provide easily readable or interpretable answers may result in them being marked as incorrect without the possibility of remarking. https://www.gradescope.ca/courses/9882

Useful Facts:

$$1\ GHz = 1 \times 10^9\ Hz$$

$$1\ byte = 8\ bits$$

$$1\ Kbyte\ (KB) = 1024\ bytes$$

Exercise 1: [15 Marks] Examine the subsequent pair of functions: *factRec* and *factIter*, both designed for calculating the factorial of a given number.

Evaluate the *instruction locality* inherent in each function. Determine whether there exists good or bad locality, and ascertain if the locality is spatial or temporal. Explain, and for the sake of discussion, focus on a specific execution scenario, such as when n = 5.

```
int factRec(int n) { if (n > 1)
      {
            return 1;
      } int n1 = factRec(n+1);
      return n1 * n;
}
```

```
int factIter(int n) { int fact = 1; for (int i = 1;
      i <= n; ++i) { fact = fact *
      i; } return fact;
}
```

Exercise 2. [15 Marks] A 16-bit computer featuring a streamlined memory hierarchy. This hierarchy encompasses a solitary cache and an unbounded backing memory. The cache operates on a 2-way set-associative model, with cache lines spanning 4 bytes and a total capacity of 32 bytes. Consider the ensuing sequence of memory *word addresses*:

8, 11, 9, 7, 15, 0, 22, 2, 6, 3

(a) Ascertain the set index and block offset for each address in the provided sequence, representing them in binary notation. Incorporate the *byte offset* as an integral part of the *block offset*. Assuming an initially empty cache, evaluate whether each reference in the address sequence yields a cache hit or a cache miss.

In the event of a cache miss, identify the specific type of cache miss—whether it is cold, conflict, or capacity. Utilize the table below to facilitate your responses:

Address	Index	Block Offset	Hit or Miss	Type of Miss
8				
31				
9				
7				
15				
20				
27				
2				
6				
0				

(b) Create a table with the relevant data for each set and line after considering the memory addresses in the provided sequence. Include the necessary information about the stored data in each cache line. (See “3350-L4-CacheExample”).

Exercise 3. [30 Marks] We will explore the impact of cache capacity on performance, focusing exclusively on the *data cache* and excluding instruction storage in the caches. Cache access time is directly linked to its capacity. For the sake of simplicity, let's assume that accessing the main memory takes 100ns, and in a specific program, 50% of instructions involve data access.

Two distinct processors, denoted as P1 and P2, are engaged in executing this program. Each processor is equipped with its own L1 cache.

	L1 size	L1 Miss Rate	L1 Hit Time
P1	64 KB	3.6%	1.26 ns
P2	128 KB	3.1%	2.17ns

(a) What is the AMAT for P1 and P2 assuming no other levels of cache?

(b) Assuming an ideal CPI of 2.0 for both processors and where the L1 hit time determines the cycle time, what is the CPI_{stall} for P1 and P2? Which processor is faster at executing this particular program?

Now consider the addition of an L2 cache to P1 with the following characteristics. The data from the previous table still holds.

L2 size	L2 Miss Rate	L2 Hit Time
16 MB	42%	21.24 s

(c) Determine the Average Memory Access Time (AMAT) for P1 with the introduction of an L2 cache. Does the inclusion of the L2 cache result in an improvement or degradation of the AMAT?

(d) Given an ideal CPI of 2.0, where the cycle time is dictated by the L1 hit time, calculate the Cycle Per Instruction (CPIstall) for P1 with the incorporation of an L2 cache.

(e) Which processor demonstrates superior speed now that P1 is equipped with an L2 cache? If P1 exhibits enhanced speed, what miss rate in its L1 cache would be required to match P2's performance? Alternatively, if P2 proves to be faster, what miss rate in its L1 cache would be necessary to equal P1's performance?

Exercise 4. [20 Marks] Consider a 64-bit computer featuring a simplified memory hierarchy. This hierarchy comprises a solitary cache and an unbounded backing memory. The cache is defined by the following attributes:

- Direct-Mapped, Write-through, Write allocate.
- Cache blocks are 8 words each.
- The cache has 128 sets.

(a) Consider the provided code snippet in the C programming language intended for execution on the previously described computer. Assume the following conditions: program instructions are not stored in the cache, arrays are aligned with the cache (starting at the beginning of a cache line), integers are 32 bits, and all other variables are exclusively stored in registers.

```
int N = 32859; int
A[N];
for (int i = 0; i < N; i += 2) {
      A[i] = A[i+1]; }
```

Determine the following:

(i) The number of cache misses.

(ii) The cache miss rate.

(iii) The type of cache misses that occur.

(b) Consider the following code fragment in the C programming language to be run on the described computer. Assume that: program instructions are not stored in cache, arrays are cache-aligned (the beginning of the array aligns with the beginning of a cache line), ints are 32 bits, and all other variables are stored only in registers.

```
int N = 32859; int
A[N]; int B[N];
for (int i = 0; i < N; ++i) { B[i] = A[i];
}
```

Determine the following:

(i) The number of cache misses.

(ii) The cache miss rate.

(iii) The type of cache misses that occur.

Exercise 5. [20 Marks] The Intel Core i7-8750H processor (more details here) has the following characteristics, taken from `/proc/cpuinfo`:

```
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
stepping        : 10
microcode       : 0xea
cpu MHz         : 2200.000
L1 cache size   : 32 KB
L2 cache size   : 256 KB
L3 cache size   : 9216 KB
physical id     : 0
siblings        : 12
core id         : 0
cpu cores       : 6
{...}
clflush size    : 64
cache line size : 64
```

Consider the following two functions `Normalize1` and `Normalize2` which take in a positive NxN matrix of `double`s and *normalizes* its entries to the range $(0,1]$. A program implementing these functions is available on OWL as `Normalize.c`.

After those two code segmenets, data is presented which was collected using the `perf` utility. This data shows runtime performance metrics of these functions executing on the Intel Core i7-8750H processor for various data sizes. In this data:

- "Normalize.bin 1 ..." executes the function Normalize1;
- "Normalize.bin 2 ..." executes the function Normalize2;
- the second command-line argument is the size N of the matrix;
- `LLC-loads` means "Last Level Cache loads", the number of accesses to L3; • `LLC-load-misses` means "Last Level Cache misses", the number of L3 cache misses;
- `cpu-cycles` is the number of CPU cycles elapsed during programing execution.

Using the knowledge learned so far in this course, the specification of the i7-8750H processor, the code fragments, and the `perf` data, answer the following questions.

(a) Why is the runtime execution of `Normalize1` faster than `Normalize2`? The values of which performance metrics from the `perf` data support your claims?

(b) Consider the miss rates of `Normalize1`. The miss rate drastically increases for values of N larger than 128. Explain why the increase occurs at this particular value of N. Give a reason why this increase is not a sharp "jump" but rather has an intermediate effect at $N = 512$.

(c) Consider the miss rates of `Normalize2`. The miss rate starts quite low but quickly increases for increasing values of N. Disucss why the miss rates for $N = 128$ and $N = 1024$ are misleading for describing the actual data locality of `Normalize2`. Which additional performance metrics would you record to get a more precise understanding of the data locality of Normalize2? Assume `perf` is capable of reporting any possible hardware event.

```
void Normalize1(double* A, int N) { double t, min = (double)
    RAND_MAX, max = 0.0; for (int i = 0; i < N; ++i) { for (int j
    = 0; j < N; ++j) { t = A[i*N + j]; if (t < min) min = t; if (t >
    max) max = t;
        }
    }

    double frac = 1 / (max - min); for (int i =
    0; i < N; ++i) { for (int j = 0; j < N; ++j) {
                A[i*N + j] = (A[i*N + j] - min) * frac;
        }
    }
}
void Normalize2(double* A, int N) { double t, min = (double)
    RAND_MAX, max = 0.0; for (int j = 0; j < N; ++j) {
        for (int i = 0; i < N; ++i) { t = A[i*N +
            j]; if (t < min) min = t; if (t >
            max) max = t;
```

```
            }
        }

        double frac = max - min; for (int j =
        0; j < N; ++j) {
            for (int i = 0; i < N; ++i) {
                                A[i*N + j] = (A[i*N + j] - min) / frac;
            }
        }
}
```

Runtime performance of Normalize1.

Performance counter stats for ’./Normalize.bin 1 256’:

4,802,912	cpu-cycles		
4,538	LLC-loads		
425	LLC-load-misses	#	9.37% of all LL-cache accesses

0.002047926 seconds time elapsed

Performance counter stats for ’./Normalize.bin 1 512’:

16,577,772	cpu-cycles		
5,409	LLC-loads		
668	LLC-load-misses	#	12.35% of all LL-cache accesses

0.004451773 seconds time elapsed

Performance counter stats for ’./Normalize.bin 1 1024’:

64,124,090	cpu-cycles		
14,016	LLC-loads		
6,100	LLC-load-misses	#	43.52% of all LL-cache accesses

0.016784636 seconds time elapsed

Performance counter stats for ’./Normalize.bin 1 1536’:

143,240,802	cpu-cycles		
24,864	LLC-loads		
14,732	LLC-load-misses	#	59.25% of all LL-cache accesses

0.038767478 seconds time elapsed

Performance counter stats for ’./Normalize.bin 1 2048’:

254,735,648	cpu-cycles		
40,122	LLC-loads		
24,405	LLC-load-misses	#	60.83% of all LL-cache accesses

0.065211502 seconds time elapsed

Performance counter stats for ’./Normalize.bin 1 2560’:

399,380,980 cpu-cycles
73,006 LLC-loads
50,260 LLC-load-misses # 68.84% of all LL-cache accesses

0.102457893 seconds time elapsed

Runtime performance of Normalize2·

Performance counter stats for ’./Normalize.bin 2 256’:

5,870,023 cpu-cycles
117,911 LLC-loads
287 LLC-load-misses # 0.24% of all LL-cache accesses

0.001692968 seconds time elapsed

Performance counter stats for ’./Normalize.bin 2 512’:

22,547,539 cpu-cycles
532,221 LLC-loads
7,504 LLC-load-misses # 1.41% of all LL-cache accesses

0.007333935 seconds time elapsed

Performance counter stats for ’./Normalize.bin 2 1024’:

121,908,975 cpu-cycles
2,047,279 LLC-loads
372,388 LLC-load-misses # 18.19% of all LL-cache accesses

0.031579656 seconds time elapsed

Performance counter stats for ’./Normalize.bin 2 1536’:

292,392,531 cpu-cycles
4,528,106 LLC-loads
1,384,196 LLC-load-misses # 30.57% of all LL-cache accesses

0.074422344 seconds time elapsed

Performance counter stats for ’./Normalize.bin 2 2048’:

694,112,174 cpu-cycles
8,338,987 LLC-loads
6,798,261 LLC-load-misses # 81.52% of all LL-cache accesses

0.173902216 seconds time elapsed

Performance counter stats for ’./Normalize.bin 2 2560’:

1,025,207,852	cpu-cycles		
12,349,636	LLC-loads		
10,425,562	LLC-load-misses	#	84.42% of all LL-cache accesses

0.258983051 seconds time elapsed