ETH login ID:
(Please print in capital letters)

Full name: _____

263-0007-00L: Advanced Systems Lab
ETH Computer Science, Spring 2020
Midterm Exam
Monday, July 08, 2020

Instructions

- Write your full name and login ID on the front.
- Make sure that your exam is not missing any sheets.
- No extra sheets are allowed.
- The exam has a maximum score of 100 points.
- No books, notes, calculators, laptops, cell phones, or other electronic devices are allowed.

Problem 1 (16 = 2+4+2+4+4)	
Problem 2 (15 = 6+6+3)	
Problem 3 (18 = 2+6+6+4)	
Problem 4 (14 = 2+2+3+5+2)	
Problem 5 (18 = 2+2+2+4+2+6)	
Problem 6 (19 = 2+2+2+2+4+2+5)	
Total (100)	

Problem 1: Operational Intensity (16 = 2+4+2+4+4)

Consider the computation $y^\top = (Ax)^\top A$ where x, y are column vectors of doubles of length n and A is an $n \times n$ matrix of doubles stored in row major order. No temporary arrays are used in this computation. `sizeof(double) = 8`. Make the following assumptions:

- A write-back/write-allocate and initially empty (cold) cache of size γ bytes.
- n is a multiple of the cache block of size $B = 64$ bytes.
- Arrays are aligned with the cache blocks.

In the derivations you can omit lower order terms (writing $\approx$ instead of $=$). Show your work.

1. Determine the flop count $W(n)$ of the computation.

   Solution:

   $$W(n) = 2n(2n-1) \approx 4n^2$$

2. Determine an upper bound for the operational intensity $I(n)$ of the computation considering only compulsory data movement. Consider only reads and ignore writes.

   Solution:

   $$Q(n) \geq 8(n^2 + 2n) \approx 8n^2$$
   $$I(n) \leq \frac{W(n)}{Q(n)} \approx \frac{4n^2}{8n^2} = \frac{1}{2}$$

3. For which sizes n would you expect the bound for $I(n)$ calculated in 2) to be roughly accurate (i.e., roughly equal to $I(n)$)?

   Solution:

   When the entire data fits in cache.

   $$8n^2 \leq \gamma$$
   $$n \leq \sqrt{\frac{\gamma}{8}}$$

4. Consider now that the computation is performed on a submatrix A' and subvector x' obtained by accessing A and x with a stride k. Assume that n is divisible by k. In Matlab notation, this is equivalent to $y'^{\top} = (A'x')^{\top}A'$ where $A' = A_{0:k:n^2-1}$, $x' = x_{0:k:n-1}$, and $y' = y_{0:k:n-1}$, so A' is an $n \times \frac{n}{k}$ matrix. For convenience, we also include the code of the computation:

```
1  /* Arrays y and tmp are initialized with zeros */
2  void f(double *A, double * x, double *y, double *tmp, int n, int k) {
3
4    // tmp = A*x (with stride k)
5    for (int i = 0; i < n; ++i)
6      for (int j = 0; j < n; j += k)
7        tmp[i] += A[i*n + j]*x[j];
8
9    // y = tmp^T*A (with stride k)
10   for (int j = 0; j < n; j += k)
11     for (int i = 0; i < n; ++i)
12       y[j] += tmp[i]*A[i*n + j];
13 }
```

Determine a new upper bound for the operational intensity $I(n)$ for the following cases considering only compulsory data movement. Consider only reads and ignore writes. Briefly explain your answer.

(a) For $k = 7$

Solution:

The stride is less than the cache block size. Thus, $Q(n)$ remains the same because the complete matrix A is still read into cache. On the other hand, $W(n)$ is reduced by a factor of 7. Thus, $I(n) \leq \frac{1}{14}$.

(b) For $k = 16$

Solution:

The stride is twice the cache block size. Thus, $Q(n)$ is reduced by a factor of 2. $W(n)$ is reduced by a factor of 16. Thus, $I(n) \leq \frac{1 \cdot 2}{2 \cdot 16} = \frac{1}{16}$.

Problem 2: Bounds for Sparse MVM (15 = 6+6+3)

We consider sparse matrix-vector multiplication, in double precision floating point, of the form $y = y + Ax$ where x, y are of length n and A is $n \times n$, sparse with k non-zero elements, and invertible. The computation is done with A in CSR (compressed sparse row) format which, as you known, uses three arrays (`values`, `col_idx` and `row_start`) to represent the sparse matrix. Indices are represented by (4-byte) integers.

Assume that the sparse MVM is executed on a computer with the following features:

- A cold (empty) cache with a cache block of size $B = 16$ bytes.
- A peak performance of 2 flops/cycle because it can execute 1 FMA/cycle.
- A read (memory) bandwidth of β bytes/cycle.

Answer the following. Show your work.

1. Compute a lower bound for the runtime and associated upper bound for the performance based on data movements by considering only compulsory misses. Consider only reads and ignore writes.

   Solution:

   Assuming compulsory misses only, everything is read once from main memory. Matrix A has k double elements and $k+n+1$ integer elements. Vectors x and y have n double elements each. A lower bound for the runtime is therefore:

   $$runtime \geq \frac{8(k+2n)+4(k+n+1)}{\beta} = \frac{12k+20n+4}{\beta}.$$

   For the performance:

   $$perf \leq \frac{2k\beta}{12k+20n+4} = \frac{k\beta}{6k+10n+2}.$$

2. Compute now an upper bound for the runtime and associated lower bound for the performance based only on data movements (since the computation is assumed to be memory bound) and assuming that all memory accesses lead to misses. Consider only reads and ignore writes.

   Solution:

   Since every memory access is a miss, to get the total bytes read from main memory, we count the number of memory accesses and multiply by the cache block (each access will bring the whole block to cache). All elements of the three arrays of A (the non-zero values, the column indices and the row array) are accessed once; thus, $2k+n+1$ misses. Vectors x and y lead to k and n misses respectively. Thus:

   $$runtime \leq \frac{16(3k+2n+1)}{\beta} = \frac{48k+32n+16}{\beta}.$$

$$perf \geq \frac{k\beta}{24k + 16n + 8}.$$

3. Using the bound calculated in 1) and assuming $k = 10n$, determine the values of β for which the Sparse MVM using CSR format is guaranteed to be memory bound. Show your derivation.

   Solution:

   This is achieved when the performance in 1) is less than the peak performance of the machine:

   $$\frac{k\beta}{6k + 10n + 2} \leq 2,$$

   which leads to

   $$\beta \leq 12 + \frac{20n + 4}{k} = 14 + \frac{2}{5n} \implies \beta \leq 14$$

Problem 3: Cache Mechanics (18 = 2+6+6+4)

Consider a 2-way set associative cache with a block size of 16 bytes and 4 sets. The replacement policy is LRU. The write policy is write-back/write-allocate. The cache is initially cold. Consider the following code:

```
1 void compute(double *A) {
2   double t1;
3   for (int i = 0; i < 8; ++i) {
4     t1 = A[i*s1];
5     A[i*s2] = t1 + 2.0;
6   }
7 }
```

Assume that A starts at memory address 0. All memory accesses happen in exactly the order that they appear and are inbound. Variables `t1` and `i` are kept in registers. Answer the following. Show your work. Hint: It helps to draw the cache.

1. What is the size of the cache in bytes?

   Solution:

   $2 \cdot 16 \cdot 4 = 128$ Bytes

2. For each of the following values of $s1$ and $s2$ and considering read and write misses do the following three things: i) determine the hit/miss pattern (something like HMMHM...); ii) draw the state of the cache after the loop has executed; iii) state what kind(s) of locality does this execution have.

   (a) $s1 = 2$ and $s2 = 4$?

   Solution:

   (i) M**H**,MM,**H**M,MM,**H**M,MM,**H**M,MM. Total of 12 misses.

   (ii)

Set	0	1
0	A_{16},A_{17}	A_{24},A_{25}
1	A_2,A_3	A_{10},A_{11}
2	A_{28},A_{29}	A_{12},A_{13}
3	A_6,A_7	A_{14},A_{15}

   (iii) Temporal

   (b) $s1 = 3$ and $s2 = 5$?

   Solution:

   (i) M**H**,MM,MM,MM,MM,**H**M,MM,**H**M. Total of 13 misses.

   (ii)

Set	0	1
0	A_{24},A_{25}	A_8,A_9
1	A_{18},A_{19}	A_{34},A_{35}
2	A_{20},A_{21}	A_{12},A_{13}
3	A_{30},A_{31}	A_{14},A_{15}

   (iii) Spacial and temporal.

3. What is the maximum number of cache misses for this code? Find a combination of $s1$ and $s2$ that achieves this maximum with $2 \leq s1 \leq s2 \leq 9$ and where all misses are compulsory misses.

   Solution:

   There are 16 memory accesses in total (8 reads and 8 writes). The first two accesses are always a compulsory miss followed by a hit (temporal locality). Thus at most we can have 15 misses. This is achieved choosing $s1 = 5, s2 = 9$ or $s1 = 7, s2 = 9$.

Problem 4: ILP (14 = 2+2+3+5+2)

Consider the AVX code below:

```
1  #include <immintrin.h>
2
3  /* Assume a, b and c are of size 4*n */
4  void f (double *a, double *b, double *c, unsigned int n) {
5      __m256d va, vb, v1, v2, v3, v4, v5;
6      size_t i;
7
8      for (i = 0; i < n; i++) {
9          va = _mm256_loadu_pd   (a + 4*i);
10         vb = _mm256_loadu_pd   (b + 4*i);
11
12         v1 = _mm256_add_pd     (va, vb);
13         v2 = _mm256_add_pd     (v1, vb);
14
15         v3 = _mm256_mul_pd     (va, va);
16         v4 = _mm256_permute_pd (v3, 0x5);
17
18         v5 = _mm256_add_pd     (v2, v4);
19
20         _mm256_storeu_pd       (c + 4*i, v5);
21     }
22 }
```

Assume that the working set fits in L1 cache and is already loaded (warm cache). Further, assume that the code is executed on a machine with the following latency, throughput, and port information:

Instruction	Latency [cycles]	Throughput (all ports) [instructions/cycle]	Port
_mm256_storeu_pd	1	1	4
_mm256_loadu_pd	1	2	2/3
_mm256_permute_pd	2	1	5
_mm256_mul_pd	5	2	0/1
_mm256_add_pd	4	1	1

Show your derivations or briefly explain your solutions.

1. Determine the flop count $W(n)$ of `f`.

   Solution:

$$W(n) = 4 \cdot (3 \times \texttt{\_mm256\_add\_pd} + 1 \times \texttt{\_mm256\_mul\_pd}) \cdot n = 16n \text{ flops}$$

2. Determine a lower bound (as tight as possible) for the runtime and an associated upper bound for the performance of `f` based on the instruction mix, ignoring dependencies between instructions (i.e., don't consider latencies and assume full throughput).

   Solution:

   Additions are the bottleneck. Thus, the runtime is at least $3n$ and the performance is at most $\frac{16}{3}$ flops/cycle.

3. Estimate the runtime (latency) of one loop iteration taking latency, throughput and dependency information into account. **Hint**: It may be useful to draw the DAG.

   Solution:

   The critical path in one iteration is 14 cycles.

4. Assume that we unroll the loop by a factor K and reorder the instructions such that all the $2K$ loads occur at the beginning of the new (unrolled) loop body, followed by the computations ($4K$ arithmetic SIMD instructions and K permutes), and finally the K stores. Assume that register pressure is not an issue (i.e., all loop variables can be held in registers). Take latency, throughput and dependency information into account and answer the following:

   (a) Estimate the runtime (latency) as a function of K of one loop iteration of this new unrolled loop.

   Solution:
   One iteration of the new unrolled loop takes $14 + (K - 1) = 13 + K$ cycles when additions are not the bottleneck (for small K). When additions become the bottleneck (for large K), one iteration takes $3K$ cycles plus 5 cycles (for the load/stores that cannot be scheduled in parallel with additions). Thus, one iteration takes $\max(13 + K, 3K + 5)$ cycles.

   (b) Based on the latency of one iteration determined in 4a), estimate a lower bound (as tight as possible) for the runtime of the unrolled version of `f`. Give the bound as a function of K and n. Assume that n is divisible by K and that loads and stores are always computed "in order" by the processor, meaning that a load from one iteration cannot be scheduled before a store from a previous iteration.

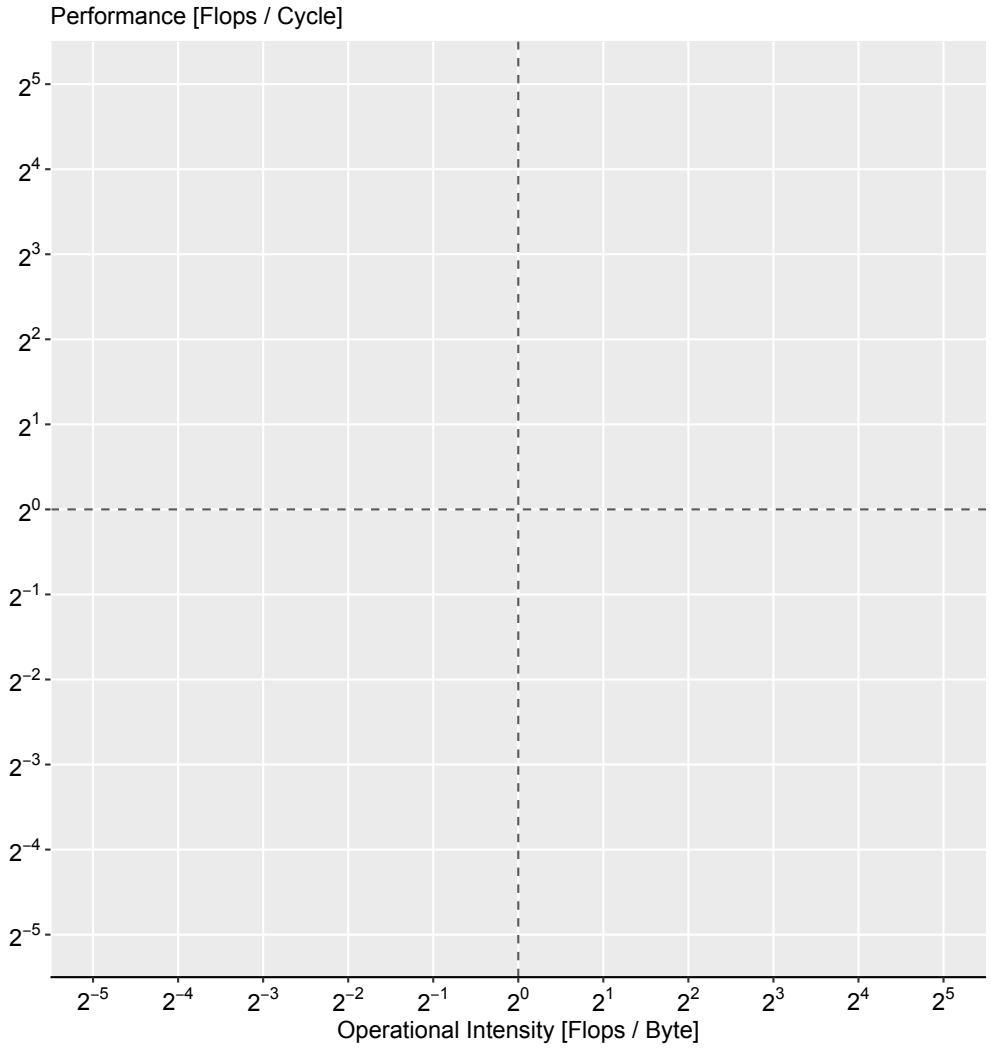
   Solution:
   We have a total of $\frac{n}{K}$ iterations. Since loads and stores have to be executed in order, one iterations has to be fully executed before the next starts. Thus, a lower bound for the runtime is

$$runtime \geq \max\left(n + 13\frac{n}{K}, 3n + 5\frac{n}{K}\right).$$

Problem 5: Roofline (18 = 2+2+2+4+2+6)

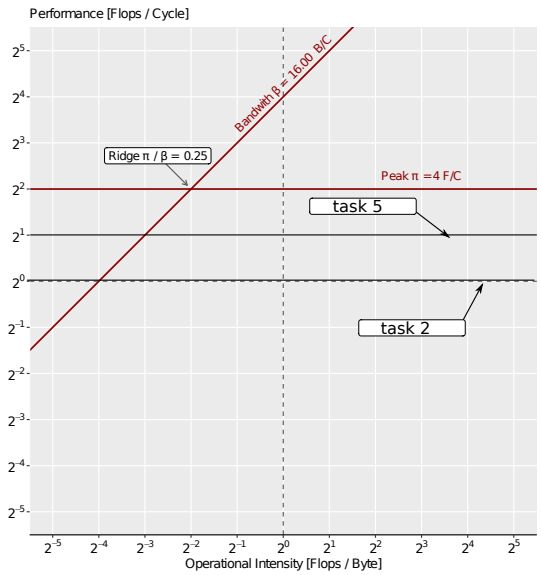
Assume a computer with the following features:

- A CPU that supports single precision floating point operations. The relevant port information is as follows:
  Port 1: fma, mul, add.
  Port 2: fma, mul.
- Each of these operations has a throughput of 1 per port.
- It does not support any SIMD operations.
- A write-back/write-allocate cache of size 1 MiB with cache block size $B = 64$ bytes.
- The read (memory) bandwidth is 4 floats per cycle. `sizeof(float) = 4`.



1. Draw the roofline plot for this computer into the above graph. Annotate the lines so we see your reasoning.

   Solution:



2. Consider the following code which computes the sum of three vectors x, y, and z (each has size n). Assume that x, y, and z are allocated sequentially in memory, one after the other, and the first element of x maps to the start of the first cache block:

```
1 void compute(float *x, float *y, float *z, int n){
2     for(int i=0; i < n; i++){
3         x[i] = x[i] + y[i] + z[i];
4     }
5 }
```

   Based on the instruction mix (i.e. ignoring dependencies), which performance is maximally achievable for this function and why? Draw an associated tighter horizontal roofline into the plot above. Assume that no FMA instructions are used.

   Solution:

   As it uses only additions, the maximum performance is 1 flops/cycle.

3. At what operational intensity $I(n)$ does this new horizontal roofline intersect with the memory roofline?

   Solution:

   Solving $\beta \cdot I = 1$, yields 1/16 flops/byte.

4. Assume the cache is fully associative and compute, for the code above, an upper bound for the operational intensity $I(n)$ considering compulsory misses only. Based on this $I(n)$, which peak performance is achievable? Consider only reads (i.e., ignore write-backs).

   Solution:

   $$W(n) = 2n, \;\; Q(n) \geq 12n$$
   $$I(n) \leq W/Q = 1/6$$

   The computation is compute bound. Thus, the peak achievable performance is 1 flop/cycle.

5. Assume the compiler is able to replace computation $a + b$ with $1 * a + b$ to utilise FMA instructions. Repeat exercise 4) taking this into consideration.

   Solution:

   The operational intensity remains the same, but the maximum performance based on the instruction mix is now 2 flops/cycle. Thus, π/b is now 1/8 flops/byte. The program is still compute bound, thus the max achievable performance is 2 flops/cycle.

6. Assume the cache is directly mapped, no FMA instructions are used, and the expressions are evaluated from left to right. Compute an upper bound (as tight as possible) for the operational intensity for $n = 2^{17}$. Consider only reads (i.e., ignore write-backs).

   Solution:

   There are conflicts between x and z as they are mapped to the same blocks. y does not have conflicts. The hit-miss pattern every 16 iterations is MMMM,HHMM,...,HHMM. Thus, x, y and z generate $\frac{17n}{16}$, $\frac{n}{16}$ and $\frac{16n}{16}$ misses respectively. In total, there are $\frac{34n}{16}$ misses. Thus, $Q \geq \frac{64 \cdot 34n}{16} = 136n$ bytes and $I \leq W/Q = 1/68$.

Problem 6: Sampler (19 = 2+2+2+2+4+2+5)

1. Assume that the peak performance π of computers doubles once every 4 years, whereas the memory bandwidth β doubles once every 8 years. Does this mean that more computations tend to become compute bound or memory bound in the sense defined by the roofline plot? Justify your answer.

   Solution:

   Performance increases faster than memory bandwidth over time. This means that the ratio π/β tends to become larger, leading to more computations becoming memory bound.

2. You are given a double type square matrix $A \in \mathbb{R}^{n \times n}$ with k non-zero entries. What is the largest k that makes it possible to save storage space by representing it in compressed sparse row (CSR) format, as opposed to the standard dense representation? Assume that indices are represented by (4-byte) integers. Show your work.

   Solution:

   The representation in dense format takes $8n^2$ bytes. The representation in CSR format takes $8k + 4(k + n + 1)$ bytes. Thus, we need

   $$12k + 4n + 4 < 8n^2$$

   $$k < \frac{2n^2 - n - 1}{3}$$

3. What is the advantage of using a write-back/write-allocate cache over a write-through/no-write-allocate cache?

   Solution:

   Subsequent writes to the same memory location does not generate memory traffic. Thus, resulting in less memory traffic for computations that exhibit good locality.

4. In Intel processors, caches use physical addresses, whereas instructions use virtual addresses. How is it then possible that L1 cache is still efficiently accessed?

   Solution:

   The lowest-order 12 bits of a memory address are the address within a 4096-byte page, so the lowest-order 12 bits of a virtual address and its corresponding physical address are the same. In the L1 cache, these 12 bits are used for the set index and the block offset. Since these bits do not require address translation, L1 cache look up can start simultaneously with TLB lookup.

5. Consider a computer with a direct-mapped TLB cache with 8 total entries. The page size is 1024 bytes. In the following code, assume that the vector x starts at address 0 in memory, and that $n = 128$. Determine memory locations (addresses) of $y[0]$ and $z[0]$ that would lead to the worst TLB hit-rate. State also this worst TLB hit-rate and justify your answer. Assume that expressions are evaluated from left to right.

```
1 void compute(float* x, float* y, float* z, int n){
2     for(int i=0; i < n; i++){
3         x[i] = x[i] + y[i] + z[i];
4     }
5 }
```

Solution:

The worst case is when starting addresses of x, y, z are in different pages but mapped to the same cache entrie. The TLB has 8 entries. Thus, each 8 KiB (2^{13} Bytes) are mapped to the same set. The worst case is when y starts at address 2^{13} and z at address 2^{14}. The hit rate in this case is about 1/4. To be more exact first iteration has a hit-miss pattern MMMM, and the following iterations HMMM. Each of vectors fully fit in a single page. Thus, we have 128*4 accesses and 127 hits. Hit rate = 127/(128*4) $\approx$ 1/4.

6. Consider an instruction with gap 2 and latency 3. How many such instructions can be fully executed in 10 cycles? Assume that there are no dependencies between them and that there is a single execution port and no other instructions are scheduled. Briefly justify your answer.

   Solution: 4

7. Does a fully associative cache always produce less or at most the same number of cache misses than a direct-mapped cache of the same size and with the same block size (assuming LRU replacement policy)? If yes, explain why this is the case. Otherwise, provide a counterexample.

   Solution:

   No. Considering a cache of 16 bytes and block size $B = 8$, and an array `double x[4]`. The access pattern `x[0], x[1], x[3], x[0]` generates 4 misses using a fully associative cache. The same access pattern produces 3 misses on a direct-mapped cache.