Last name, first name: ______________________

Student number: ______________________

263-0007-00L: Advanced Systems Lab
ETH Computer Science, Spring 2022
Midterm Exam
Wednesday, April 27, 2022

Instructions

- Write your full name and student number 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 (22 = 2+2+4+4+6+4)	
Problem 2 (13 = 3+2+4+4)	
Problem 3 (14 = 6+4+4)	
Problem 4 (16 = 2+2+6+6)	
Problem 5 (18 = 2+4+2+4+6)	
Problem 6 (17 = 6+4+7)	
Total (100)	

Problem 1: Sampler (22 = 2+2+4+4+6+4)

Be brief in your answers, no need to show derivations unless indicated otherwise.

1. Why can a CPU resolve write-after-read (WAR) and write-after-write (WAW) dependencies but not read-after-write (RAW)? How are these dependencies resolved?

   Solution: In RAWs the result is necessary for the computation whereas WAR and WAW are false in the sense that we just happened to write to the same register but there's no need to. The dependencies are resolved via register renaming.

2. Can a computation with an $O(1)$ operational intensity benefit from blocking for the caches?

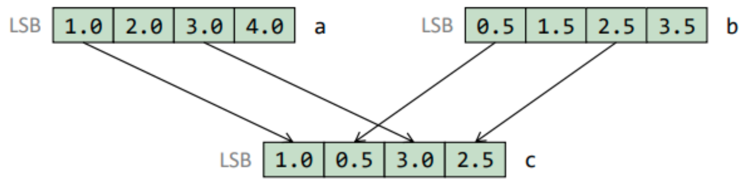
   Solution: Yes.

3. Answer the following regarding SIMD intrinsics. Use of pseudo code or descriptive pictures in your answer are both fine.

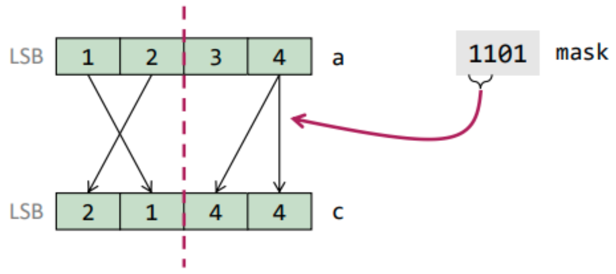
   (a) Provide the specification for **one** of the following intrinsics (you can choose which one). We will take the worst answer if two specifications are provided. Use the notation x_i to indicate the i-th element of a vector x.

   - `_mm256_permute_pd(__m256d a, int mask)` or
   - `_mm256_unpacklo_pd(__m256d a, __m256d b)`

   Solution:
   `c = _mm256_unpacklo_pd(a, b)`



   `c = _mm256_permute_pd(a, mask)`



(b) Explain what `_mm256_set1_ps` does.

Solution: Broadcast a single-precision (32-bit) floating-point value to all elements of a (256-bit) vector of single-precision floats.

4. Consider the computation $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} A_{ij}$ that sums all elements of a matrix of doubles A of size $n \times n$. Assume that matrix A is sparse with k non-zero elements and full rank. The computation is done with A in CSR (compressed sparse row) format. Indices are represented by (4-byte) integers. Determine a tight upper-bound for the operational intensity of the computation. Show your work.

Solution: Since we are adding all elements in the *values* array. There is no need to access the indices. Thus:

$$W(n) = k - 1$$
$$Q(n) \geq 8k$$
$$I(n) \leq \frac{k-1}{8k} \approx \frac{1}{8}$$

5. Consider a computer with a direct-mapped cache of size 1024 bytes and a block size of 64 bytes. In addition, it has a direct-mapped TLB with 32 entries. The page size is 2MiB. In the following code, assume that the vector x starts at address 0 in memory and that all memory accesses happen in exactly the order that they appear. Determine the smallest value for k that yields a TLB miss for every memory access but only yields compulsory misses in the cache. The cache and TLB are initially empty. Show your work.

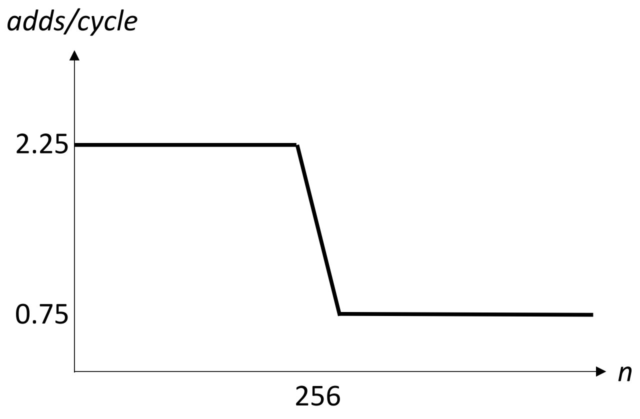
```c
void compute(double *x, unsigned int k){
  double t;
  for (int i = 0; i < 256; i += 1){
    t = x[i];
    x[i + k] = t + 0.1;
  }
}
```

Solution: In order to get only TLB misses, $x[i]$ and $x[i+k]$ should map to the same entry in the TLB. The page size is 2^{18} doubles and the TLB has 2^5 entries. Thus, they will map to the same entry when $k = 2^{18} \cdot 2^5 = 2^{23}$. With this k, $x[i]$ and $x[i+k]$ will contend for the same cache block, leading to conflict misses when acessing the cache. In order to get only compulsory misses in the cache, $x[i+k]$ should map to the block next to $x[i]$. This is achieved with $k = 2^{23} + 8$.

6. Consider the following function. `sizeof(float) = 4`.

```c
void vecsum(float *a, float *b, float *c, float d*, int n){
  for (int i = 0; i < n; i += 1) {
    d[i] = a[i] + b[i] + c[i] + d[i];
    }
}
```

It is run on an Intel-like single-core computer that can perform 3 additions per cycle, without SIMD vector executions. The machine only has one cache. For different input sizes n (starting with very small n), warm-cache measurement yields the following performance plot.



(a) Estimate the size of the cache in bytes. Note that the x-axis shows the input size n which is the length of each vector.

Solution: Cache size $\approx 4 \cdot 4 \cdot 256 = 4$KB.

(b) Estimate the read bandwidth β_{cache} in bytes/cycle to the cache.

Solution:
$\beta_{\text{cache}} = 12$ bytes/cycle.

Problem 2: Bounds (13 = 3+2+4+4)

Consider the following function:

```c
void compute(float* x, int n, int m){
    float v1, v2, v3;
    float c1 = 0.1;
    float c2 = 0.2;
    float c3 = 0.3;
    for(int i=0; i < m-1; i++){
        x[(i+1)*n] = 1.0;
        for(int j=0; j < n-2; j++){
            v1 = x[i*n + j];
            v2 = x[i*n + j+1];
            v3 = x[i*n + j+2];
            x[(i+1)*n+j+1] = (v1+c1)*(v2+c2)*(v3 OP c3); //OP provided in text
        }
        x[(i+1)*n + n-1] = x[i*n + n-1];
    }
}
```

Assume that the above code is executed on a computer with the following relevant latency, gap (inverse throughput), and port information:

Instruction	Latency	Gap (inverse throughput)	Port
	[cycles]	[cycles/instruction]	
add	3	1	0
mult	3	0.5	0/1
div	5	5	2

The processor does **not** support vector instructions. Further assume that:

1. You can ignore the latency and throughput of loads and stores, i.e., assume they have zero latency and infinite throughput.
2. The compiler does not apply any algebraic transformation: the operations are mapped to assembly instructions as shown.
3. Ignore integer operations.
4. A division counts as one floating-point operation.

Show enough detail with each answer so we understand your reasoning.

1. Determine the maximum theoretical floating-point peak performance in flops/cycle of the computer under consideration.

   Solution: Every five cycles the processor can schedule at most 5 additions, 5 multiplications and 1 division, resulting in a peak performance of $\frac{11}{5}$ flops/cycle.

2. Determine the exact flop count $W(n, m)$ of the `compute` function. Assume that OP (in line 12) counts as one floating-point operation.

   Solution:
   $W(n, m) = 5(n - 2)(m - 1) \approx 5nm$

3. Determine a lower bound (as tight as possible) for the runtime (in cycles) and an associated upper bound for the performance of the `compute` function based on the instruction mix, ignoring dependencies between instructions (i.e., don't consider latencies and assume full throughput). Consider the following two cases:

   (a) assume that OP is a **division** operation.
   Divisions are the bottle neck.

   $$T(n) \geq 5(n - 2)(m - 1)$$

   .

   $$P(n) \leq 1 \text{ flop/cycle}$$

   (b) assume that OP is a **multiplication** operation.

   $$T(n) \geq 2.5(n - 2)(m - 1)$$

   .

   $$P(n) \leq 2 \text{ flops/cycle}$$

4. Estimate a lower bound (as tight as possible) for the number of cycles that the computation in line 12 takes to complete. Take latency, throughput and dependency information into account and assume that OP is a **division** operation. Draw the corresponding DAG of the computation performed in line 12.

   Solution:
   10 cycles.

Problem 3: Operational Intensity (14 = 6+4+4)

Consider the following code implementing a strided matrix vector multiplication ($y = Ax+y$):

```c
void comp1(double *A, double *x, double *y, int n, int stride){
  for(int i = 0; i < n; i+= stride)
      for(int j = 0; j < n; j+= stride)
          y[i] += A[i*n + j] * x[j]
}
```

Assume the following:

- `sizeof(double)` = 8.
- A write-back/write-allocate cold cache
- The cache block size is 64 bytes.
- The stride is a power of two.
- n is a multiple of the stride: $n = ms$ for $m \in \mathbb{N}$.
- The flop count is $2 \cdot (\frac{n}{s})^2 = 2m^2$

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

1. Determine a hard upper bound for the operational intensity $I(n, s)$ [flops/byte] in terms of n and the stride denoted as s. Consider only compulsory misses for both reads and writes.

   Solution:

$$W(n, s) = 2 \cdot \left(\frac{n}{s}\right)^2 = 2 \cdot \frac{n^2}{s^2}$$

$$Q(n, s) \geq \begin{cases} (\frac{n}{s} \cdot B) \cdot \frac{n}{s} & \text{if } s \geq \frac{B}{8} \\ 8 \cdot n \cdot \frac{n}{s} & \text{if } s < \frac{B}{8} \end{cases}$$

$$I(n, s, B) \leq \begin{cases} \frac{2}{B} & \text{if } s \geq \frac{B}{8} \\ \frac{1}{4s} & \text{if } s < \frac{B}{8} \end{cases}$$

   For $B = 64$:

$$I(n, s) \leq \begin{cases} \frac{1}{32} & \text{if } s \geq 8 \\ \frac{1}{4s} & \text{if } s < 8 \end{cases}$$

2. You are given a computer which has 2 ports. Each port can execute one addition or one multiplication per cycle (no FMA, and no vector instructions). The memory bandwidth β is 24 bytes per cycle. For which strides is the computation memory bound in the sense of the roofline plot?

   Solution: The computation is memory bound when $I(n) < \frac{2}{24} = \frac{1}{12}$. Thus, the computation is memory bound for $s \geq 4$.

3. Consider a version that is strided across only one dimension:

```c
void comp2(double *A, double *x, double *y, int n, int stride){
  for(int i = 0; i < n; i+= stride)
      for(int j = 0; j < n; j++) // no stride here
          y[i] += A[i*n + j] * x[j]
}
```

For the same machine as in Task 2 and assuming that only compulsory misses happen during the computation, can comp2 ever be memory bound? Justify your answer.

Solution: No. The computation is memory bound when $I(n) < \frac{1}{12}$. However the operational intensity is $I(n, s) = \frac{1}{4}$ (see below). Thus, the computation is always compute bound.

$$W(n, s) = 2 * \left(\frac{n}{s}\right) \cdot n$$

$$Q(n, s) \approx 8 \cdot \frac{n^2}{s}$$

$$I(n, s) \approx \frac{1}{4}$$

Problem 4: Cache Mechanics (16 = 2+2+6+6)

You are given a write-back/write-allocate cache with 4 sets and LRU replacement policy. Its block size is 12 bytes, and the capacity is 96 bytes. Consider the following code which is the same as in Problem 2. `sizeof(float) = 4`.

```
void compute(float* x, int n, int m){
    float v1, v2, v3, v4;
    float c1 = 0.1;
    float c2 = 0.2;
    float c3 = 0.3;
    for(int i=0; i < m-1; i++){
        x[(i+1)*n] = 1.0;
        for(int j=0; j < n-2; j++){
            v1 = x[i*n + j];
            v2 = x[i*n + j+1];
            v3 = x[i*n + j+2];
            x[(i+1)*n + j+1] = (v1+c1)*(v2+c2)*(v3+c3);
        }
        v4 = x[i*n + n-1];
        x[(i+1)*n + n-1] = v4;
    }
}
```

Assume that array `x` starts at the memory address `0`. Variables `i`, `j`, `c1`, `c2` and `c3` are stored in registers. Memory accesses happen in exactly the order that they appear. Answer the following. Show your work. Hint: It helps to draw the cache.

1. How many floats fit into this cache?

   Solution: $96/4 = 24$ floats.

2. What is the associativity of this cache?

   Solution: $e = 96/12/4 = 2$ (2-way set associative).

3. For each of the following values of m and n do the following two things: i) determine the miss rate; ii) draw the state of the cache at the end of the computation. Show your work.

   (a) For $m = 2$ and $n = 12$:

   Solution:

   i. There are 8 compulsory misses in 43 accesses.
   $miss\_rate = 8/43$.

   ii.

Set	0	1
0	$x_{0,1,2}$	$x_{12,13,14}$
1	$x_{3,4,5}$	$x_{15,16,17}$
2	$x_{6,7,8}$	$x_{18,19,20}$
3	$x_{9,10,11}$	$x_{21,22,23}$

(b) For $m = 3$ and $n = 8$:

Solution:

i. Again, all data fits in cache. Thus, there are 8 compulsory misses in 54 accesses.
$miss\_rate = 8/54$.

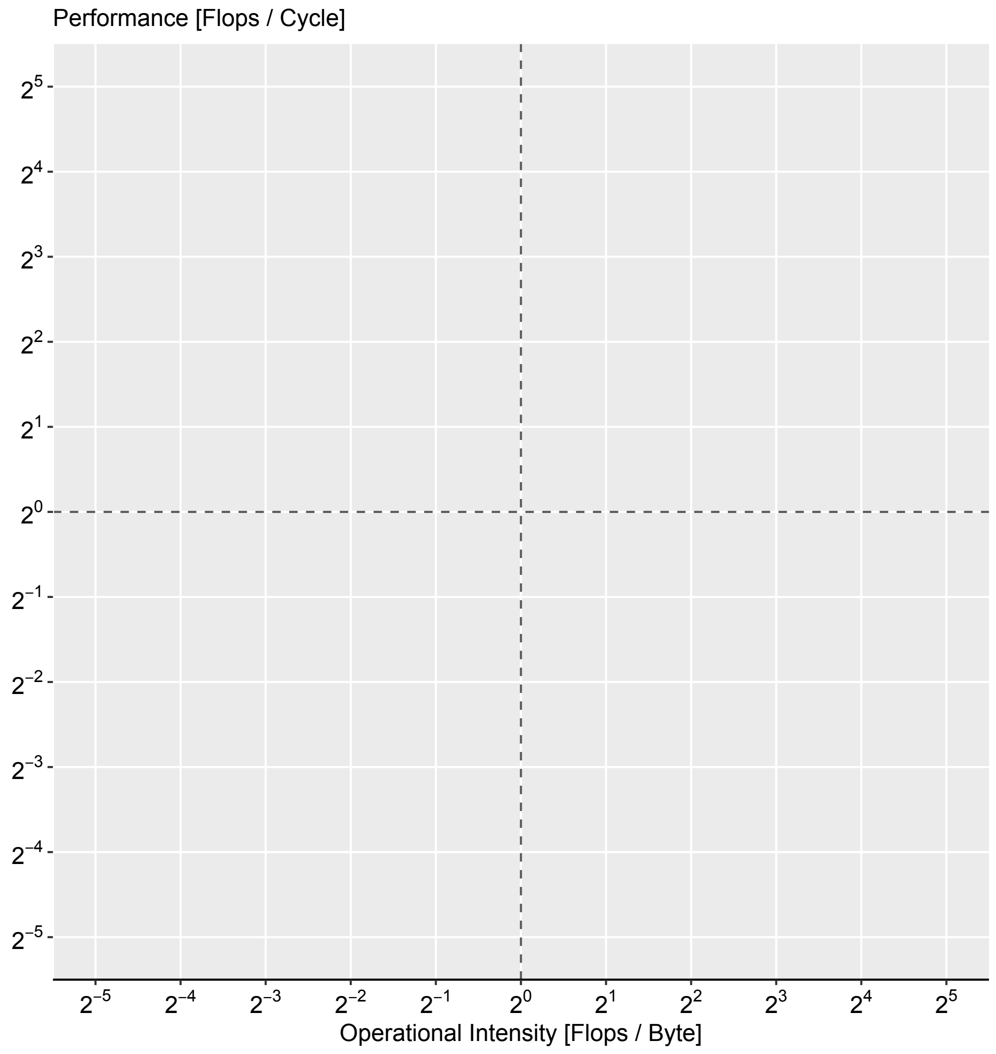
ii.

Set	0	1
0	$x_{0,1,2}$	$x_{12,13,14}$
1	$x_{3,4,5}$	$x_{15,16,17}$
2	$x_{6,7,8}$	$x_{18,19,20}$
3	$x_{9,10,11}$	$x_{21,22,23}$

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

Assume a computer with the following features:

- A CPU with the following ports:
  Port 1: FMA, MUL, ADD.
  Port 2: MUL.
  Port 3: ADD.
- Each of these operations has a throughput of 1 per port and a latency of 4 cycles.
- It does not support any SIMD operations.
- A write-back/write-allocate cache of size 2 MiB with cache block size $B = 64$ bytes. The cache is initially cold.
- The read (memory) bandwidth is 2 doubles per cycle. `sizeof(double) = 8`.



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

   Solution:

   $\pi = 4$ flops/cycle

   $\beta = 8 \cdot 2 = 16$ bytes/cycle

   $\pi/\beta = 1/4$ flops/byte

2. Consider the following computation where x, y and z are arrays. Assume that x, y, and z are cache-aligned allocated (i.e., the address of an array maps with the first element of a block in the first set of the cache):

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

   (a) Based only on the instruction mix, (i.e., ignoring all type of dependencies), which performance is maximally achievable for this function and why? Draw an associated tighter horizontal roofline into the plot above.

   Solution:

   Ignoring data dependencies, there are n FMAs, $2n$ MULs and $2n$ ADDs which are executed. Port 1 can execute n FMAs + $\frac{1}{3}$ ADDs + $\frac{1}{3}$ MULs, Port 2 can execute $\frac{5n}{3}$ MULs and Port 3 can execute $\frac{5n}{3}$ ADDs. Thus, $T(n) \geq \frac{5n}{3}$ cycles. $W(n) = 6n$, therefore a tight bound based on the instruction mix is $\frac{3 \cdot 6n}{5n} = \frac{18}{5} = 3.6$ flops/cycle.

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

   Solution:

   $I = \pi/\beta = \frac{18}{5 \cdot 16} = \frac{9}{40}$ flops/byte.

   (c) What is the performance bound if dependencies are also considered? Assume IEEE 754 arithmetic, i.e., operations cannot be reordered.

   Solution: Intraloop dependencies: `+ z[i]` ADD can only be scheduled after the first FMA has finished executing, that is, 4 cycles later. The `x[i+1] + y[i+1]` ADD can be scheduled at the same time as the FMA but the `* y[i]` MUL is dependent on the results of both operands, meaning it can be scheduled only 8 cycles after the beginning of the loop iteration. The last MUL `* z[i+1]` will be scheduled 12 cycles into the iteration and finish after 16 cycles in total.

Inter loop dependencies: the only independent operation that can be scheduled across loops is the `x[i+1] + y[i+1]` MUL. Everything else is dependent on the previous iteration.

In total, to schedule all instructions approximately $16n$ cycles are necessary, meaning that the tighter performance bound is $6n/16n = 3/8$ flops/cycle.

3. Assume the cache is fully associative and large enough to fit all the arrays. What is the upper bound for the operational intensity $I(n)$ considering all cache misses? Consider only reads (i.e., ignore write-backs). Based on this $I(n)$, which peak performance is achievable on the specified system taking into account instruction mix and dependencies (i.e., the setting of Task 2c)?

Solution:

$$W(n) = 6n, \quad Q(n) \geq 8 \cdot 3n$$

$$I(n) \leq W/Q = 1/4$$

The computation is memory bound in the setting of Task 2c when $I(n) \leq \frac{3}{8\cdot 16}$. Since $\frac{1}{4} > \frac{3}{8\cdot 16}$, the computation is compute bound and the peak achievable performance is 3/8 flops/cycle.

Problem 6: Cache Miss Analysis (17 = 6+4+7)

Consider the following function that uses a *i*-*j*-*k* loop and takes as input matrices A and B of size $n \times n$ and a vector x of size $5n$. A, B, x are not aliased. `sizeof(double) = 8`.

```c
/* NOTE: Assume that the notation A[i][j] is transformed to A[i*n + j].
 *       We use the notation A[i][j] for readability only. */
void f(double *A, double *B, double *x, int n){
  double t1, t2;
  for (int i = 0; i < n-1; i++)
    for (int j = 0; j < n-1; j++)
      for (int k = 0; k < n; k++) {
        t1 = A[k][ j ] * B[i+1][k];
        t2 = A[k][j+1] * B[ i ][k] + x[j + 4*k];
        A[k][j] = t1 + t2;
      }
}
```

Assume a fully associative write-back/write-allocate cache of size γ bytes with LRU replacement policy, and a cache block size of 64 bytes. Further, assume that n is divisible by 8. Assume an initially cold cache and answer the following. Show your work.

1. Assume that n is much larger than γ (i.e., $n \gg \gamma$) and that γ can fit all data in the innermost loop (i.e., $\gamma > 5 \cdot 64$). Consider cache misses from both reads and writes. Estimate the number of cache misses incurred when accessing each of the arrays as a function of n. In the derivations you can omit lower order terms (writing $\approx$ instead of $=$).

   (a) Misses when accessing A:

   $$Misses_A \approx \frac{9n^3}{8}$$

   (b) Misses when accessing B:

   $$Misses_B \approx \frac{n^3}{4}$$

   (c) Misses when accessing x:

   $$Misses_x \approx \frac{n^3}{2}$$

2. Determine the minimum value for γ, as precise as possible, such that the computation only has compulsory misses, i.e., a cache miss only occurs on the first access to a block. For this, assume that LRU replacement is not used and, instead, cache blocks are replaced as effectively as possible to minimize misses.

   $$\gamma \geq 8 \cdot (n^2 + 2n + 5n) = 8n^2 + 56n$$

3. Repeat Tasks 1 and 2 assuming that function `f` uses a *j*-*k*-*i* loop instead, i.e., the code now looks as follows:

```c
void f(double *A, double *B, double *x, int n){
  double t1, t2;
  for (int j = 0; j < n-1; j++)
    for (int k = 0; k < n; k++)
      for (int i = 0; i < n-1; i++) {
        t1 = A[k][ j ] * B[i+1][k];
        t2 = A[k][j+1] * B[ i ][k] + x[j + 4*k];
        A[k][j] = t1 + t2;
      }
}
```

   (a) Misses when accessing A:

$$Misses_A \approx \frac{9n^2}{8}$$

   (b) Misses when accessing B:

$$Misses_B \approx n^3$$

   (c) Misses when accessing x:

$$Misses_x \approx \frac{n^2}{2}$$

   (d) Minimum value of γ such that the computation only has compulsory misses:

$$\gamma \geq 8 \cdot (8n + 8 + n^2 + 4n + 8) = 8n^2 + 96n + 128$$