Last name, first name: _____

Student number: _____

263-0007-00L: Advanced Systems Lab
ETH Computer Science, Spring 2023
Midterm Exam
Wednesday, April 26, 2023

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 (20 = 2+2+2+4+4+6)	
Problem 2 (14 = 2+2+3+3+4)	
Problem 3 (14 = 3+2+6+3)	
Problem 4 (16 = 6+4+6)	
Problem 5 (18 = 2+1+3+2+4+6)	
Problem 6 (18 = 1+1+2+4+6+4)	
Total (100)	

Problem 1: Sampler (20 = 2+2+2+4+4+6)

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

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

2. Assume that `a`, `b`, `c` are variables of type `__m256d` with initial values `a`= $\{2, 4, 6, 8\}$ and `b`= $\{1, 5, 5, 9\}$. Determine the resulting value of `c` after the following operation: `c = _mm256_cmp_pd(a,b,_CMP_GT_OQ)`. Note that with macro `_CMP_GT_OQ`, the intrinsic will compute a "greater than" comparison.

3. You are given two **sparse** matrices A and B and two column vectors x, and y. You want to compute the MVMs: Ax and $y^T B$. How would you store the matrices to improve cache performance?

4. Consider the following two functions. Function `sum` computes the sum of all elements in an array x of size n and function `sum_vec` is a naive vector implementation of `sum`. Both functions compile without errors.

```
1 double sum(double* x, int n){
2     double a = 0.0;
3     for (int i = 0; i < n; i++)
4         a += x[i];
5     return a;
6 }
```

```
1 double sum_vec(double* x, int n){
2     __m256d a = _mm256_setzero_pd();
3     for (int i = 0; i < n; i += 4){
4         __m256d t0 = _mm256_load_pd(x+i);
5         a = _mm256_add_pd(a, t0);
6     }
7     return a[0] + a[1] + a[2] + a[3];
8 }
```

(a) Which two problems could an unsuspecting user of function `sum_vec` encounter?

(b) How would you improve the performance of function `sum` without using SIMD vector intrinsics?

5. Consider the following function. `sizeof(double) = 8`.

```
1 void f(double *x, unsigned int n, unsigned int s) {
2   double t = 0.0;
3   for (int i = 0; i < n; i += 1) {
4     t += x[i*s];
5   }
6 }
```

Assume that array x is cache-aligned (i.e., the address of the array maps with the first element of a block in the first set of the cache) and allocated sufficiently large such that all accesses in line 4 are within bounds. n is a multiple of 8. Assume that the cache is initially empty and the cache block size is $B = 64$ bytes. Determine a tight lower bound for the number of bytes transferred Q from memory to the cache as a function of n and s.

6. Does a 2-way set associative cache with LRU replacement policy 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? If yes, explain why this is the case. Otherwise, provide a counterexample.

Problem 2: Bounds (14 = 2+2+3+3+4)

Consider the following function:

```
1 void compute(double* x, double* y, double* z, int n) {
2     double a = 0.5;
3     double b = 0.3;
4     double c = 0.1;
5     for(int i = 0; i < n; i++){
6       // OP1 and OP2 are provided in text
7       z[i] = ((a + x[i]) * (b OP1 y[i])) OP2 (c OP1 z[i]);
8     }
9 }
```

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

Instruction	Latency [cycles]	Gap (inverse throughput) [cycles/instruction]	Port(s)
`add`	2	0.25	0,1,2,3
`mult`	3	0.33	0,1,2
`div`	6	4	3

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 their respective assembly instructions.
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.

2. Determine the exact flop count $W(n)$ of the `compute` function. Assume that OP1 and OP2 count as one floating-point operation each.

3. Assume that both OP1 and OP2 are **multiplication** operations. 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).

4. Repeat the previous task assuming now that OP1 is a **multiplication** and OP2 a **division**.

5. Estimate a lower bound (as tight as possible) for the number of cycles that the computation in line 7 takes to complete. Take latency, throughput and dependency information into account and assume that OP1 is a **division** operation and OP2 a **multiplication** operation. Draw the corresponding DAG of the computation performed in line 7.

Problem 3: Operational Intensity (14 = 3+2+6+3)

Assume the following for the below computations:

- `sizeof(double)` = 8.
- A write-back/write-allocate cold cache.
- No temporary arrays are used in the computations.

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

1. Consider the computation $E = AB - CD$ where A, B, C, D, E are $n \times n$ matrices of doubles. The computation is implemented using a triple-loop.

   (a) Determine a hard upper bound for the operational intensity $I(n)$ [flops/byte] considering only compulsory misses. Consider only reads, i.e., data movements from memory to cache. Recall that for this cache, arrays that are only written are also read and this read should be included.

   (b) Would the operational intensity of the computation be different on a write-through/no-write-allocate cache? Justify your answer and update $I(n)$ if it changed.

2. Consider the computation $z = z + A(x + y)$ where x, y, z are column vectors of doubles of length n. A is an $n \times n$ sparse matrix with $3n$ non-zero elements and invertible. The computation is done with A in CSR (compressed sparse row) format which, as you know, uses three arrays (`values`, `col_idx` and `row_start`) to represent the sparse matrix. Indices are represented by (4-byte) integers.

   (a) Determine a hard upper bound for the operational intensity $I(n)$ [flops/byte]. Consider only compulsory misses. Consider only reads, i.e., data movements from memory to cache.

   (b) Assume that the computation is performed in a computer with a memory bandwidth of $\beta = 32$ bytes/cycle and peak performance of π flops/cycle. For which values of π is the computation guaranteed to be memory-bound in the sense of the roofline model?

Problem 4: Cache Miss Analysis (16 = 6+4+6)

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

```
1 /* NOTE: Assume that the notation A[i][j] is transformed to A[i*n + j]. */
2 void f(double *A, double *B, double *C, int n) {
3     for(int j = 0; j < n; j++)
4         for(int k = 0; k < n; k+=2) // Incremented by 2
5             for(int i = 0; i < n; i++)
6                 C[i][j] += A[i][k]*B[j][k] + A[i][k+1]*B[j][k+1];
7 }
```

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. Determine the overall 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.

   (a) Misses when accessing A:

   (b) Misses when accessing B:

   (c) Misses when accessing C:

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.

3. Repeat Tasks 1 assuming that `f` uses a k-i-j loop. The code now looks as follows:

```
1 void f(double *A, double *B, double *C, int n) {
2     for(int k = 0; k < n; k+=2) // Incremented by 2
3         for(int i = 0; i < n; i++)
4           for(int j = 0; j < n; j++)
5                 C[i][j] += A[i][k]*B[j][k] + A[i][k+1]*B[j][k+1];
6 }
```

   (a) Misses when accessing A:

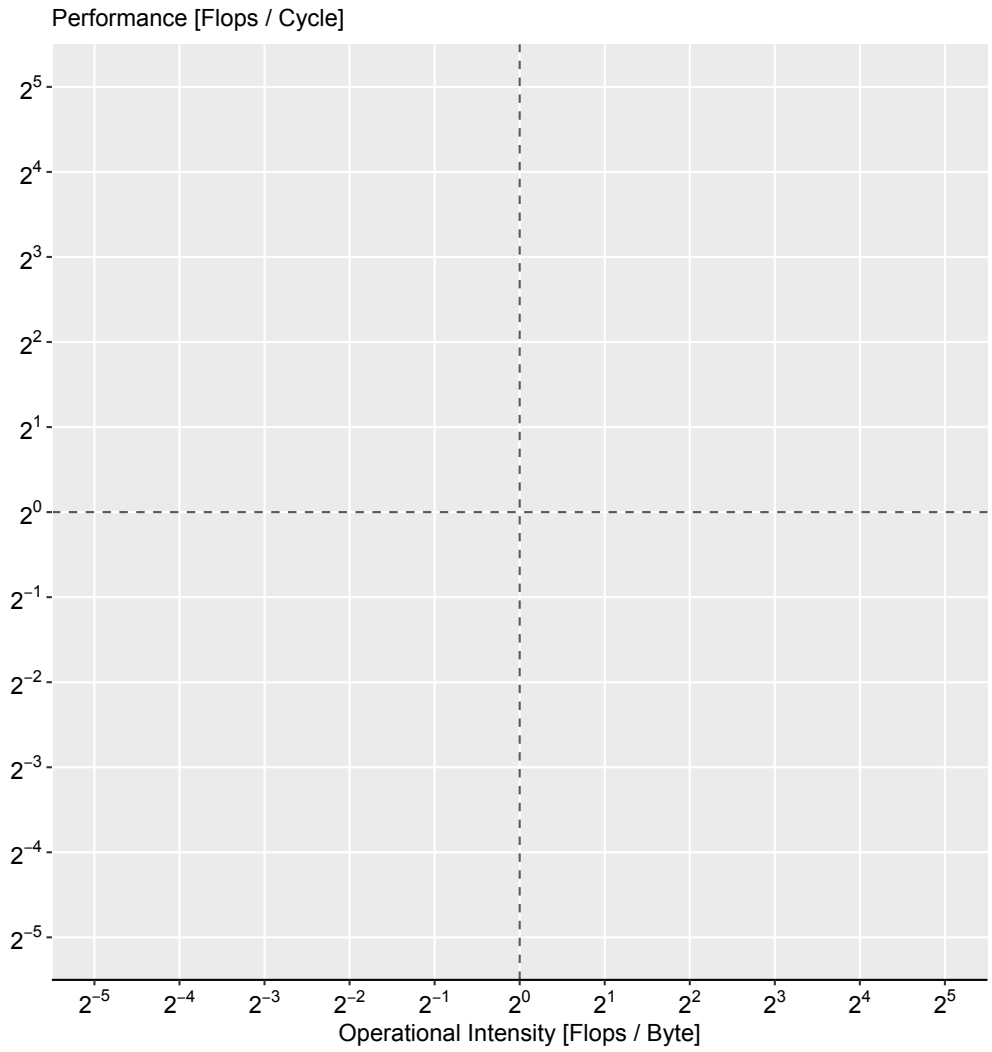
   (b) Misses when accessing B:

   (c) Misses when accessing C:

Problem 5: Roofline (18 = 2+1+3+2+4+6)

Assume a computer with the following features:

- A CPU with the following ports:
  Port 1: ADD, MUL, FMA.
  Port 2: ADD, MUL.
  Port 3: ADD, DIV.
- A division operation counts as one flop and has a latency of 8 cycles and a gap (inverse throughput) of 2 cycles.
- The remaining operations have 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. The cache is initially cold.
- The read (memory) bandwidth is $\beta_{read} = 32$ bytes per cycle.



1. Draw the roofline plot for this computer into the above graph based on maximum performance and read bandwidth. Annotate the lines so we see your reasoning.

2. Consider the following computation where x, y and z are arrays. Assume that x, y, and z are cache-aligned (i.e., the address of an array maps with the first element of a block in the first set of the cache). `sizeof(float) = 4`. Assume that FMAs are used whenever it is possible to fuse an addition with a multiplication:

```
1 void compute(float* x, float* y, float* z, int n){
2   for(int i = 1; i < n; ++i){
3     float a = x[i] * y[i] + y[i-1];
4     float b = z[i-1] / x[i-1];
5     z[i] = a * b;
6   }
7 }
```

   (a) Determine the exact flop count $W(n)$ of the compute function.

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

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

(d) What is the performance bound if dependencies are also considered? Assume that operations cannot be reordered (which is the case in the IEEE 754 standrad).

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 2d)?

Problem 6: Cache Mechanics (18 = 1+1+2+4+6+4)

Consider the following function that takes as input matrices A and B of size $N \times M$ stored in row major order. A and B are not aliased.

```
1 void compute(float* A, float *B, int N, int M){
2   for(int j = 0; j < M-3; j += 4)
3     for(int i = 0; i < N; i++)
4       for(int k = j; k < j+4; k++) {
5         float a = A[i*M + k];
6         float b = B[i*M + k];
7         B[i*M + k] = a + b;
8       }
9 }
```

Make the following assumptions:

- A write-back/write-allocate direct mapped cache of size $\gamma = 1$ KiB, and a cache block size of 64 bytes.
- Array A starts at memory address 0 and B starts at memory address $25 \cdot 2^6$, i.e., `&A[0] = 0` and `&B[0] = &A[400]`. With these addresses, both arrays are aligned to a cache block.
- Array A is cache-aligned, (i.e., the first element of A goes to the first element of a block in the first set of the cache)
- Memory accesses happen in exactly the order that they appear in lines 5–7.
- `sizeof(float)` $= 4$.

Consider $N = 6$ and $M = 48$ (thus A and B are not aliased), and answer the following. Justify your answers.

1. How many sets are in this cache?

2. Determine the cache set where array B starts.

3. How many memory accesses occur in the computation in total?

4. Determine the miss/hit pattern for A and B in lines 5-7 (something like Line 5: MMHH..., Line 6: MMMH..., Line 7: HHMM) for the first iteration of the outermost loop (i.e., for $j = 0$).

5. What is the hit-rate for all memory accesses to A and B in the entire computation? Recall that the hit-rate is $num\_hits/num\_accesses$.

6. Assume now that the cache is two-way set associative with LRU replacement policy. The cache size remains unchanged (i.e., $\gamma = 1$ KiB). What is the hit-rate for all memory accesses to A and B in the entire computation?