Last name, first name: ______________________

Student number: ______________________

263-0007-00L: Advanced Systems Lab
ETH Computer Science, Spring 2026
Midterm Exam
Wednesday, April 22, 2026

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, laptops, cell phones, or other electronic devices are allowed.
- Dedicated calculators are allowed, i.e., not the ones in your mobile phones.

Problem	Score
Problem 1 (2+4+4+3+5+5=23)	
Problem 2 (2+1+4+4+6=17)	
Problem 3 (4+2+2+4+4=16)	
Problem 4 (2+1+3+2+3+3=14)	
Problem 5 (1+1+4+4+6=16)	
Problem 6 (2+2+6+4=14)	
Total (100)	

Problem 1: Sampler (2+4+4+3+5+5=23)

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

1. A floating-point division on a certain CPU has latency 18 cycles and inverse throughput 6 cycles per instruction. What is the minimum number of independent divisions required so that the divider can be kept fully busy?

2. Two instructions `I0` and `I1` have inverse throughput of 2 and 3, respectively. What is the best (lowest) achievable inverse throughput of an even instruction mix of `I0` and `I1` (i.e., they occur in equal numbers) in the following two cases:

   - `I0` and `I1` share the same execution unit.

   - `I0` and `I1` share a port (but are in different execution units).

3. Consider the following idea for a rigorous definition of conflict misses for a program P run on a cache with parameters (S, E, B): make the cache fully associative (to take out all conflicts), run again, and take the difference. Formally:

   Conflict misses of P on (S, E, B) = Misses of P on (S, E, B) − Misses on $(1, SE, B)$.

   Explain why this definition works or why not.

4. Answer the following:

   - What is the translation lookaside buffer (TLB)?

   - Describe a memory access pattern in code where every data access results in a TLB miss.

5. Consider the computation $y = y + Ax$. y and x are vectors of size N and A is a sparse matrix of size $N \times N$ with a density of $0 < \alpha < 1$ (i.e., the fraction α of its elements are non-zero). A is stored in standard CSR format. Assume a memory bandwidth β bytes/cycle, cold cache with block size of 4 bytes, and all data types used are of size 4 bytes (`float` and `int`). Find bounds on the runtime **based only on data movement** in the following two cases. Consider only loads.

   (a) Determine, as precisely as possible, a lower bound on the runtime by considering only compulsory misses.

   (b) Estimate an upper bound on the runtime by also assuming that *every* access to x is a miss.

6. Given the following snippet of code:

```
1 void func(double *x, double s, int n) {
2   for (int i = 0; i < n; i++)
3     x[i] = s * x[i] + i;
4 }
```

Assume `n` is a multiple of 4 and `x` is 32-byte aligned. Rewrite the loop using AVX intrinsics. You can use the following intrinsics: `_mm256_load_pd`, `_mm256_mul_pd`, `_mm256_add_pd`, `_mm256_store_pd`, `_mm256_set_pd`, and `_mm256_set1_pd`. Do not optimize for ILP (in particular, no extra unrolling). Optimize the innermost loop for throughput. To make this faster, you can avoid declaring types and you can shorten the name of the intrinsics to only the instruction. For example, instead of `__m256d c = _mm256_add_pd(a,b)`, you can write `c = add(a,b)`.

Problem 2: Bounds (2+1+4+4+6=17)

Consider the following function:

```
1
2  void compute(double *x, double *y, int n) {
3      double X = 0;
4      double Y = 0;
5      for (int i = 0; i < n; i++) {
6          X = X + x[i];
7          Y = Y + y[i];
8          double XY = X * Y;
9
10         // OP is defined in text
11         double t0 = x[i] OP y[i];
12         y[i] = t0 OP XY;
13
14         double t1 = std::sqrt(XY);
15         double t2 = t1 + XY;
16         double t3 = Y * x[i];
17         x[i] = t2 + t3;
18     }
19 }
```

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

Instruction	Latency [cycles]	Inverse throughput [cycles/instruction]	Port(s)
add	5	1	0
mult	7	0.5	0,1
sqrt	15	5	1

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. Each instruction is mapped to its own execution unit,
4. Ignore integer operations,
5. A square root 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.

```
X = X + x[i];
Y = Y + y[i];
double XY = X * Y;

// OP is defined in text
double t0 = x[i] OP y[i];
y[i] = t0 OP XY;

double t1 = std::sqrt(XY);
double t2 = t1 + XY;
double t3 = Y * x[i];
x[i] = t2 + t3;
```

Instruction	Latency [cycles]	Inv. throughput [cycles/instr.]	Port(s)
add	5	1	0
mult	7	0.5	0,1
sqrt	15	5	1

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

3. Assume that OP is an **addition**. 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., do not consider latencies and assume full throughput).

4. Repeat the previous task assuming now that OP is a **multiplication**.

5. Estimate a lower bound (as tight as possible) for the latency (in cycles) of a single iteration of the loop. Take latency, throughput, ports, and dependency information into account and assume that OP is a **multiplication**. Draw the corresponding DAG of the computation. You can use the back of the previous page.

Problem 3: Operational Intensity (4+2+2+4+4=16)

Consider a system with the following specifications:

- *Peak Performance:* 2 Ports (each with ADD and MUL) $\pi = 2$ flops/cycle.
- *Memory Bandwidth:* $\beta = 20$ bytes/cycle.
- *Data Type:* `double` precision (`sizeof(double)=8`).
- *Cache Policy: Write-back/Write-allocate.* Assume n is large enough that we only consider compulsory misses (cold cache).
- *Cache Block Size:* $B = 64$ bytes.

Consider the following function executed on the above machine.

```
1 void kernel(double *A, double *B, int n) {
2     for (int i = 1; i < n - 1; i++) {
3         for (int j = 1; j < n - 1; j++) {
4             B[i*n + j] = 0.4 * A[i*n + j] +
5                          0.15 * (A[(i-1)*n + j] + A[(i+1)*n + j] +
6                                  A[i*n + (j-1)] + A[i*n + (j+1)]);
7         }
8     }
9 }
```

In the following questions, you can omit lower order terms.

1. Compute a hard upper bound on the operational intensity of `kernel` (in flops/bytes). Assume that loop indices (i, j) remain in registers. Consider both loads and stores.

2. Determine if the function is compute or memory bound on the machine specified above.

Consider the following implementation of `kernel` that uses a stride s (power of 2) in the innermost loop.

Listing 1: Strided kernel

```
1 void kernel_strided(double *A, double *B, int n, int s) {
2     for (int i = 1; i < n - 1; i++) {
3         for (int j = 1; j < n - 1; j += s) {
4             B[i*n + j] = 0.4 * A[i*n + j] +
5                          0.15 * (A[(i-1)*n + j] + A[(i+1)*n + j] +
6                                  A[i*n + (j-1)] + A[i*n + (j+1)]);
7         }
8     }
9 }
```

1. What is the work of this new function based on the stride.

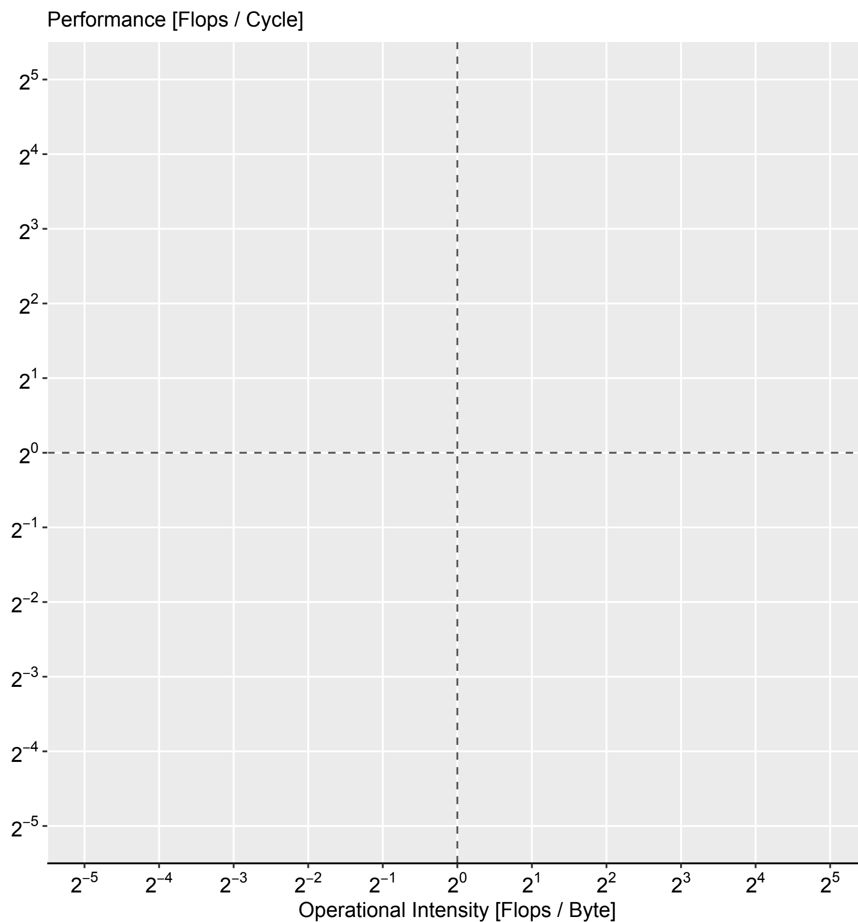
2. What is the data movement of this new function based on the stride.

3. For which values of s is this function memory-bound in the sense of the roofline model?

Problem 4: Roofline (2+1+3+2+3+3=14)

Assume a computer with the following features:

- A CPU with the following double-precision floating-point ports:
  Port 0: FMA, MUL, ADD.
  Port 1: MUL.
  Port 2: ADD, SQRT.
- A square root instruction counts as one flop and has a latency of 18 cycles and an inverse throughput of 6 cycles.
- All other instructions have a latency of 4 cycles and a throughput of 1 cycle per port.
- It supports 512-bit AVX operations with the same latency and throughput as the corresponding scalar operations.
- A write-back/write-allocate cache. The cache is initially cold.
- The load (memory) bandwidth is $\beta_{load} = 8$ bytes per cycle.
- `sizeof(double)=8`



Show enough detail in each answer so we can see your reasoning.

1. Draw the roofline plot for this computer into the above graph. Add labels to the lines so we see your reasoning. Draw 2 horizontal rooflines, one considering AVX and one without.

Consider the following computation where x, y, and z are arrays. Assume that x, y, and z are cache-aligned, i.e their first element map to the start of the first cache block.

```
1 void compute(double* x, double* y, double* z, double* w, int n) {
2     double acc = 0.0;
3     for (int i = 0; i < n; ++i) {
4         acc += x[i] * y[i];
5         w[i] = acc + sqrt(z[i]);
6     }
7 }
```

2. What is the exact flop count $W(n)$ of the compute function?

3. Based on the instruction mix (i.e., considering only throughput and ignoring dependencies), what performance is maximally achievable for this function and why? Draw an associated tighter horizontal roofline into the plot above. Assume that the FMA instruction unit is used optimally.

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

5. Now also consider the data dependencies in the code. Does the asymptotic throughput bound from question (3) change? Justify your answer by identifying the relevant dependency structure.

6. Assume that the cache is fully associative and large enough to fit all four arrays. Compute, for the code above, an upper bound for the operational intensity $I(n)$ considering compulsory misses only. Based on this $I(n)$, is the computation memory-bound or compute-bound in the scalar setting of part (3)?

Problem 5: Cache Mechanics (1+1+4+4+6=16)

Consider a machine with a direct-mapped write-back/write-allocate cache with blocks of size B=16 bytes and a total capacity of 128 bytes. Assume `sizeof(float)` = 4 bytes.

1. How many single-precision floating-point values can be stored in this cache?

2. How many cache sets does it have?

Consider the following code. Assume

- memory accesses occur in exactly the order that they appear, i.e load `value`, load `flag` and store `value`.
- scalar variables (iter, i) remain in registers and do not cause cache misses,
- *arr* is cache-aligned (first element goes into first cache block).
- All fields of the struct are stored contiguously in memory.

```
1  struct data_t {
2    float value;
3    float data[14];
4    float flag;
5  };
6
7  #define N 3
8
9  float comp(data_t* arr) {
10   for (int iter = 0; iter < 1000; iter++) {
11     for (int i = 0; i < N; i++) {
12       arr[i].value += arr[i].flag;
13     }
14   }
15 }
```

3. Determine the miss/hit pattern for *arr* (something like *arr*: MMHH...) at line 12 for all values of iter.

4. Repeat (3) considering the following definition of the struct `data_t`.

```
struct data_t {
  float value;
  float data[14];
  float flag;
  float padding[8];
};
```

5. Propose a new definition of the struct `data_t` that achieves at least the same hit-rate as in (4) and minimizes memory usage. Maintain the original `value`, `data`, and `flag` fields. *Hint: reorder the fields and insert a suitable padding.*

Problem 6: Cache Miss Analysis (2+2+6+4=14)

Consider a write-back/write-allocate cache with total capacity of 128 bytes and an array A of size 64 doubles. Assume `sizeof(double) = 8` bytes, cold cache, and A is cache-aligned ($A[0]$ is aligned to the first line of the cache).

Consider the following 6 accesses to array A.

$$A[0], A[10], A[35], A[2], A[34], A[1]$$

1. Assume the cache is direct-mapped (E=1). The miss-hit pattern of the above 6 accesses is `MMMMMH`. What is the block size of the cache?

2. Assume the cache is associative and has block size 4 doubles (B=4). The miss-hit pattern of the above 6 accesses is `MMMMHH`. What is the associativity of the cache?

Consider the following matrix computation, where C, A, and B are matrices of size $n \times b$, $n \times b$ and $b \times b$ respectively using a j-i-k loop.

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

Assume that the code is executed on a machine with a write-back/write-allocate fully-associative cache with blocks of size 64 bytes, a total capacity of γ doubles and with a LRU replacement policy. Assume that n and b are divisible by 8, cold caches, and that all matrices are cache-aligned. Justify all your answers.

```
1  //Same code as above so you do not have to flip pages.
2  void update(double *A, double *B, double *C, int n, int b) {
3      for(int j = 0; j < b; j++)
4        for(int i = 0; i < n; i+=2) // increased by 2!
5            for(int k = 0; k < b; k++)
6                  C[i * b + j] += A[i * b + k]*B[k * b + j] + A[(i+1) * b + k];
7  }
```

3. Assume that $2\gamma \ll n$ and $16b \ll \gamma$ and determine, as precisely as possible, the total number of cache misses that the computation has. For each of the matrices (A, B and C), state also the kind(s) of locality it benefits from to reduce misses.

   (a) Number of misses and types of localities for A:

   (b) Number of misses and types of localities for B:

   (c) Number of misses and types of localities for C:

4. Determine the minimum value for γ, as precise as possible, such that the computation only has compulsory misses. For this, assume that LRU replacement is not used and, instead, cache blocks are replaced as effectively as possible to minimize misses.