ETH login ID:
(Please print in capital letters)

Full name: _____

263-2300: How to Write Fast Numerical Code
ETH Computer Science, Spring 2019
Midterm Exam
Monday, April 15, 2019

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 (17 = 4+4+4+5)	
Problem 2 (16 = 5+2+4+3+2)	
Problem 3 (24 = 2+2+6+6+8)	
Problem 4 (14 = 6+8)	
Problem 5 (15 = 2+9+4)	
Problem 6 (14 = 5+2+3+2+2)	
Total (100)	

Problem 1: Bounds (17 = 4+4+4+5)

Consider an imaginary processor with the following throughputs for different floating point operations and the five ports on which they are executed:

Instruction	Max. throughput (all ports) [instructions/cycle]	Port number
`store`	1	0
`load`	2	1 and 2
`add/subtract`	1	3
`mult`	2	3 and 4

Consider the three functions `f1`, `f2`, and `f3` below that produce equivalent outputs:

```
1  void f1 (const double *x, const double *y, double *res, size_t N){
2    size_t i;
3    for(i=0; i < N; i++){
4      double a = x[i];
5      double b = y[i];
6      res[i] = a*a + b*b + 2*a*b;
7    }
8  }
9
10 void f2 (const double *x, const double *y, double *res, size_t N){
11   size_t i;
12   for(i=0; i < N; i++){
13     double a = x[i];
14     double b = y[i];
15     res[i] = (a+b) * (a+b);
16   }
17 }
18
19 void f3 (const double *x, const double *y, double *res, size_t N){
20   size_t i;
21   for(i=0; i < N; i++){
22     double a = x[i];
23     double b = y[i];
24     res[i] = (a-b) * (a-b) + 4*a*b;
25   }
26 }
```

Assume the following:

1. The working set of `f1`, `f2`, and `f3` fits L1 cache and is already loaded into L1.
2. No algebraic compiler transformations are applied: the operations are mapped to assembly instructions as shown
3. Ignore integer operations.

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

1. Determine for each function: a **lower bound** (as tight as possible) for the runtime (in cycles) and an associated **upper bound** for the performance of `f1`, `f2`, and `f3` based on the instruction mix (i.e, ignore dependencies between operations).

   `f1:`

   `f2:`

   `f3:`

2. Now we move the multiplier at port 3 to a new port numbered 5. Does this change any of the bounds? Explain.

Problem 2: Operational Intensity (16 = 5+2+4+3+2)

In the following computations, x, y, z are vectors of doubles of length n and A, B, X, Y are $n \times n$ matrices of doubles. No temporary arrays are used in these computations. We assume a write-back/write-allocate cache and a cold cache (of size γ bytes) at the start of the computation. sizeof(double) = 8. **In the derivations you can omit lower order terms** (writing $\approx$ instead of $=$). Show your work.

1. (a) Determine an upper bound for the operational intensity $I(n)$ of the computation $y = Ax + Bz + y$ considering only compulsory data movement.

   (b) Assume a processor with a peak performance $\pi = 4$ flops/cycle. What is the least memory bandwidth such that the computation could become compute bound?

2. (a) Determine an upper bound for the operational intensity $I(n)$ of the computation $Y = AX - XB$ considering only compulsory data movement.

   (b) For which sizes n would you expect $I(n)$ to be roughly accurate?

(c) Would the operational intensity of the computation be different on a write-through/no-write-allocate cache? If no, explain. If yes, give the bound for $I(n)$ in this case.

Problem 3: Cache Mechanics (24 = 2+2+6+6+8)

Consider the following code. You are given a 2-way set associative write-back/write-allocate cache with LRU replacement. Its block size is 16 bytes, and the capacity is 128 bytes.

Consider the following code; n is the common length of the arrays a, b, c.

```
1  void compute(double *a, double *b, double *c, int n)
2  {
3    // first loop
4    for(int i = 0; i < n; i++) {
5      double x = a[i];
6      double y = b[i];
7      c[i] = x + y;
8    }
9    // Show state of cache at this point
10   // second loop
11   for(int i = n-1; i >= 0; i--) {
12     double x = a[i];
13     double y = b[i];
14     c[i] = x + y;
15   }
16 }
```

Assume `a` starts at memory address `0`, `b` directly follows `a` in memory and `c` directly follows `b`. Memory accesses happen in exactly the order that they appear. Hint: It helps to draw the cache.

1. What is the capacity of the cache in doubles?

2. How many sets does the cache have?

3. For each of the following values of n do the following three things: i) determine the number of hits and misses for executing the first loop; ii) draw the state of the cache after the first loop; iii) determine the number of hits and misses in the second loop. Show your work.

(a) $n = 4$?

(b) $n = 8$?

(c) $n = 12$?

Problem 4: Blocking (14 = 6+8)

Given are two arrays A and B of integers, each of length n. `sizeof(int) = 4`. The longest common subsequence (LCS) can be computed using the following dynamic program that fills an $n \times n$ table L: (A, B, L are not aliased)

```
1  // assume sizeof(int) = 4
2  // assume all needed L[0][j] and L[i][0] are initialized to 0
3  void LCS(int **L, int *A, int *B, int n){
4    int i, j, k;
5
6    for (i = 1; i < n; i += 1) {
7      for (j = 1; j < n; j += 1){
8        L[i][j] = max(L[i-1][j], L[i][j-1], L[i-1,j-1]);
9        if (A[i-1] == B[j-1]) L[i][j] = L[i][j]+1;
10     }
11   }
12 }
```

Assume a fully-associative write-back/write-allocate cache of size γ bytes, a cache block size of 32 bytes, and only one cache. Further assume that n is much larger than γ. In the following we perform two cache miss analyses assuming an initially cold cache. **In the derivations you can omit lower order terms** (writing $\approx$ instead of $=$). Show your work.

1. Estimate the number of cache misses incurred by LCS as a function of n.

2. Now we try to reduce the number of misses by blocking the computation into blocks of size $b \times b$, b a multiple of 8. This means it now has the following loop structure (we ignore clean-up code if b does not divide n):

```
1  // assume sizeof(int) = 4
2  // assume all needed L[0][j] and L[i][0] are initialized to 0
3  void LCSblocked(int **L, int *A, int *B, int n){
4    int i, j;
5
6    for (i = 1; i < n; i += b) {
7      for (j = 1; j < n; j += b){
8        for (i1 = i; i1 < i+b-1; i1 += 1){
9          for (j1 = j; j1 < j+b-1; j1 += 1){
10           L[i1][j1] = max(L[i1-1][j1], L[i1][j1-1], L[i1-1,j1-1]);
11           if (A[i1-1] == B[j1-1]) L[i1][j1] = L[i1][j1]+1;
12     }
13   }
14 }
```

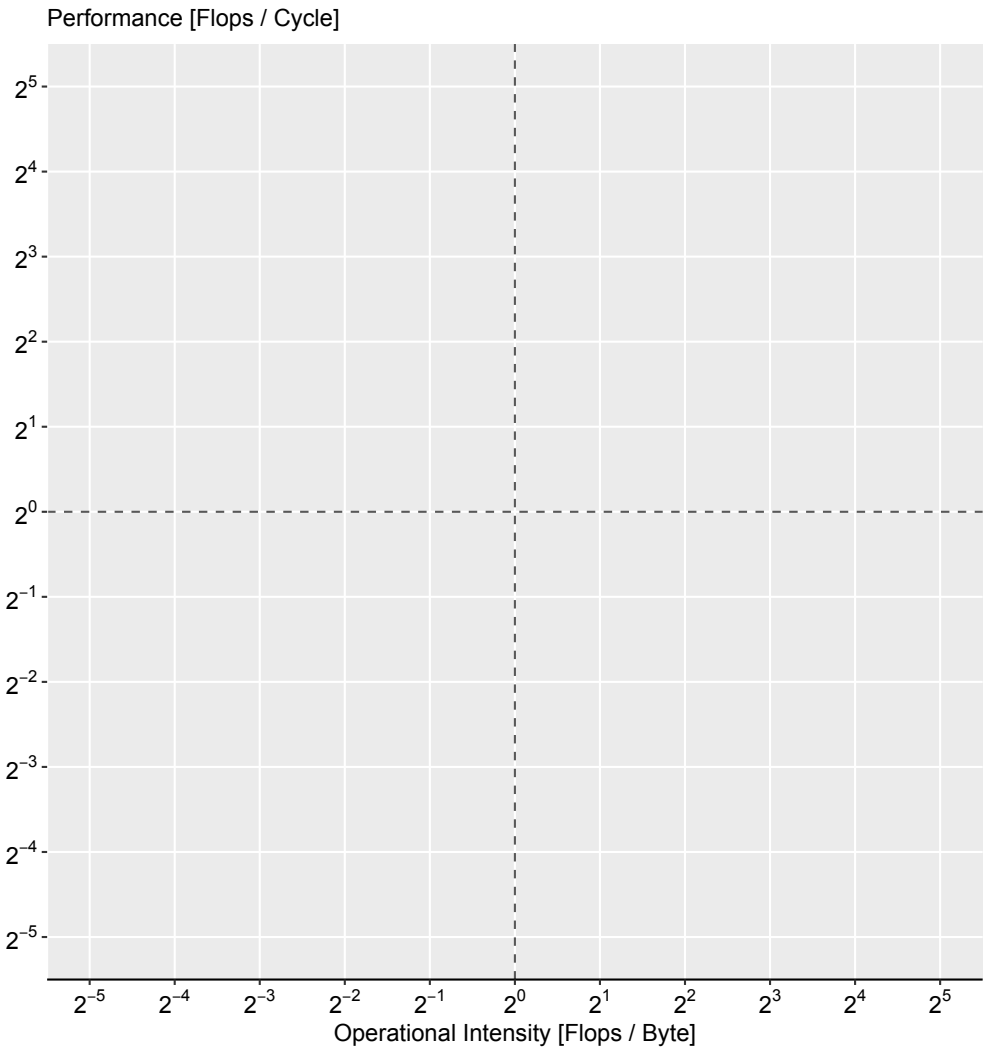
Estimate the number of cache misses incurred by LCSblocked. In doing so, upper bound the size of b so that you achieve good cache locality. Show also this bound.

Problem 5: Roofline and Sparse MVM (15 = 2+9+4)

We consider sparse matrix-vector multiplication, in double precision floating point, of the form $y = Ax + y$, where x, y are of length n and A is $n \times n$ and sparse. The computation is done with A in CSR (compressed sparse row) format where indices are represented by (4-byte) integers. You know that A is invertible and has $5n$ many entries.

The single-core computer you run on has a memory read bandwidth of 8 bytes/cycle and a peak performance of 2 because it can execute 1 FMA/cycle.

1. Use this information to draw a roofline plot for this processor.



2. From the above information determine a hard upper bound for the performance of the sparse MVM assuming a cold cache. Show your derivation and visualize the result in the above roofline plot.

3. Assume now that all indices in CSR are stored as 2-byte short integers. By how much can the performance bound be improved? Show your work.

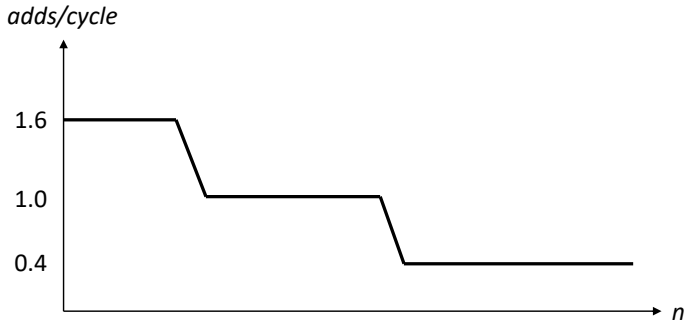
Problem 6: Sampler (14 = 5+2+3+2+2)

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

1. Consider the following function

```
1 void vecsum(float *a, float *b, float *c, float d*, float e*, int n){
2   int i;
3
4   for (i = 0; i < n; i += 1) {
5     a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
6   }
7 }
```

   It is run on an Intel-like single-core computer that can perform 2 additions per cycle, without SIMD vector executions. For different input sizes n (starting with very small n), warm-cache measurement yields the following performance plot.



   How many caches are there?

   Estimate the read bandwidths in bytes/cycle to the caches and memory.

2. Which types of dependencies can be resolved by renaming? How is renaming done dynamically by the processor?

3. Consider the following function

```
1 void vecsum(float *a, float *b, int n){
2   int i;
3
4   for (i = 0; i < n; i += 1) {
5     a[i] = a[i] + b[i];
6   }
7 }
```

   compiled in isolation. Can the compiler vectorize (use SIMD vector instructions for) this function? Explain.

4. Earlier Pentium processors had an L1 cache only half the size of Sandybridge or Haswell. When doubling the size, Intel decided to double the associativity. Why?

5. With a CPU that can perform 2 FMAs per cycle with a latency of 5 cycles, how many accumulators would you use for a dot-product to (hopefully) achieve the peak performance?