ETH login ID:
(Please print in capital letters)

Full name: _____

263-2300: How to Write Fast Numerical Code
ETH Computer Science, Spring 2016
Midterm Exam
Wednesday, April 20, 2016

Instructions

- Make sure that your exam is not missing any sheets, then write your full name and login ID on the front.
- 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 (8)	
Problem 2 ($20 = 5 + 3 + 3 + 3 + 6$)	
Problem 3 ($20 = 4 + 6 + 5 + 5$)	
Problem 4 ($24 = 12 + 12$)	
Problem 5 (8)	
Problem 6 ($20 = 4 + 8 + 8$)	
Total (100)	

Problem 1: Peak Performance (8)

Assume a processor that can execute at peak 1 floating point addition and 1 floating point multiplication per cycle. Further assume a function that requires a floating point additions and b floating point multiplications. What is the maximal percentage of peak performance that this function can achieve on this processor? Show your work.

Problem 2: Roofline (20=5+3+3+3+6)

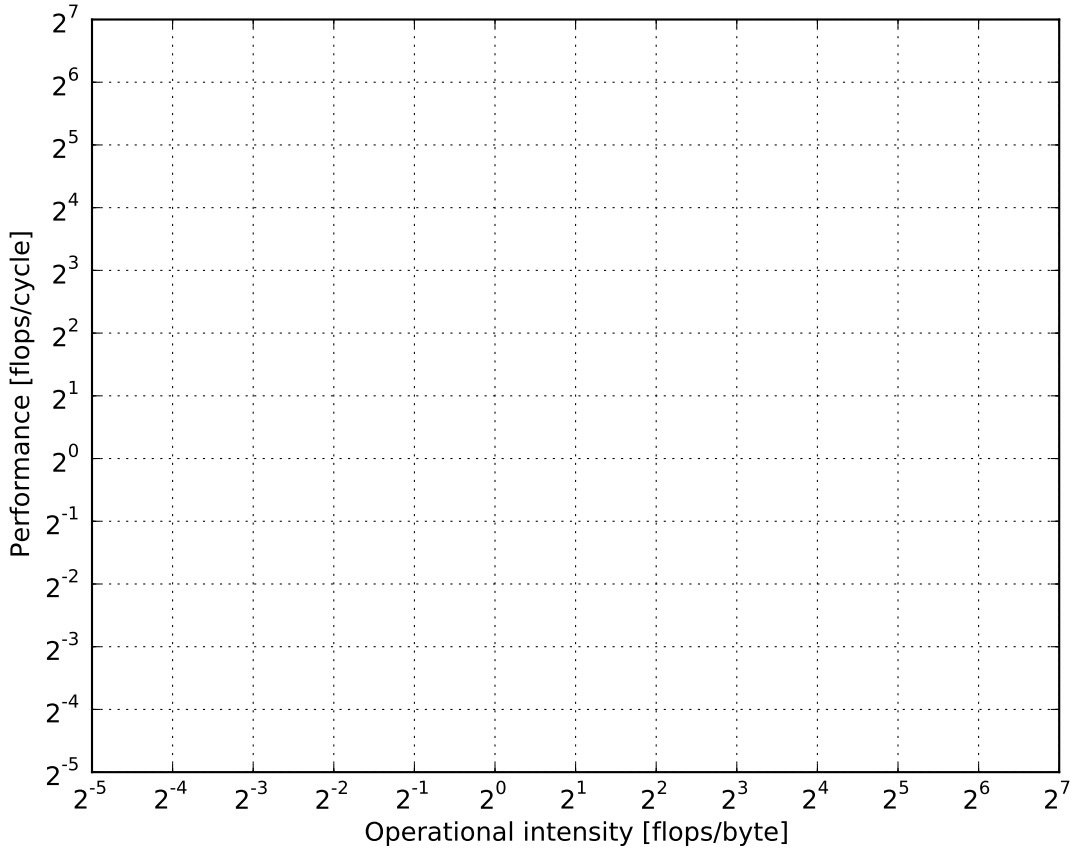
Assume a computer with the following features:

- The CPU can issue, in double precision, 2 (scalar) floating point multiplication and 2 (scalar) floating point additions/subtractions per cycle.
- The total bandwidth between CPU and main memory is 4 bytes/cycle.
- The last level cache is write-back/write-allocate with a size of 2 MB and a block size of 64 bytes.

Further we consider a function f implementing $C = AB + C$ in double precision, where each of A, B, C is an $n \times n$ triangular matrix, i.e., has $n(n+1)/2$ entries, and is stored contiguously in memory. The flop count for this computation is $n(n+1)(n+2)/3$. The function is executed with cold cache.

Note: slight approximations in the calculations are allowed. Explain enough so we see how you got the result.

1. Draw the roofline plot for this system.



2. Determine an upper bound for the operational intensity $I(n)$ of f. Consider reads and writes.

3. For which sizes n is this bound likely to be the actual intensity?

4. For which sizes n is the computation guaranteed memory bound?

5. Now assume the matrices A, B, C are spread in memory (meaning each is stored non-contiguously with large gaps). How does this change the answers to the three previous questions 2-4?

Problem 3: Performance Analysis (20=4+6+5+5)

The array x contains N vectors of length 64 stored consecutively. The function `n2norm` computes the 2-norm for each of these vectors and stores the results in array y:

```
1  void n2norm(const double * x, double * y, size_t N) {
2    int i;
3    double f1, f2;
4    for (i = 0; i < N; i++) {
5      f1 = 0.;
6      f2 = 0.;
7      for (j = 0; j < 64; j+=2) {
8        f1 += x[i*N+j]*x[i*N+j];
9        f2 += x[i*N+j+1]*x[i*N+j+1];
10     }
11     y[i] = sqrt(f1+f2);
12   }
13 }
```

We make the following assumptions:

- The CPU can issue in each cycle one floating point add and one floating point mult. Both have a latency of one cycle.
- In addition the CPU can issue every 32 cycles a square root. The latency is 32 cycles.
- The system has two levels of cache. Both are write-back/write-allocate.
- L1 cache: size 64 kB, read bandwidth 4 double/cycle.
- L2 cache: size 1 MB, read bandwidth 0.5 double/cycle.
- RAM: read bandwidth 0.25 double/cycle.
- The scalar variables i, j, $f1$, $f2$, and N are stored in registers.
- A double is 8 bytes.

Note: You are allowed to approximate and drop negligible terms.

1. Determine the flop count of this function.

2. Determine the maximal values of N for which the working set fits into

   (a) L1 cache

   (b) L2 cache

3. The performance of `n2norm` is measured as average over many executions. Sketch the expected performance plot for N up to 100'000. N is on the x-axis and the y-axis shows the percentage of floating point peak performance (between 0% and 100%) achieved. Provide enough details and also short explanations so we can verify your reasoning.

4. Now assume that the CPU can issue 2 fused multiply-add (FMA) instructions per cycle, and that each FMA has a latency of 1 cycle. Create a new performance plot analogous to part 3.

Problem 4: Cache Mechanics (24=12+12)

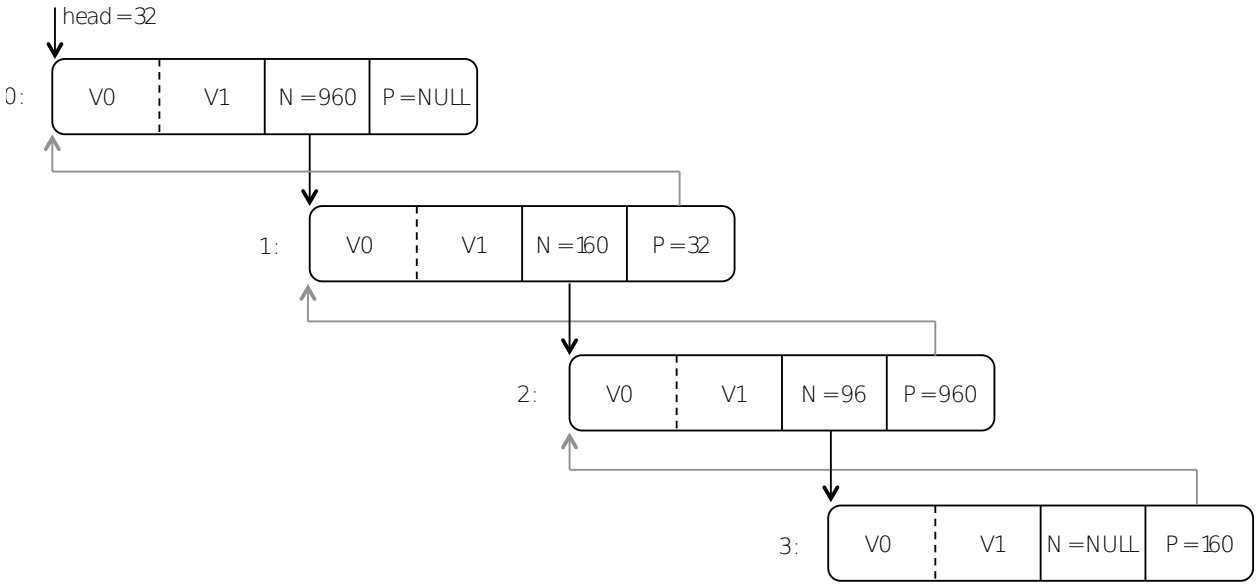
We consider a doubly linked list where nodes have the following structure:

```
1 struct Node {
2   double values[2];     // Labels: V0, V1
3   struct Node* next;    // Label:  N
4   struct Node* prev;    // Label:  P
5 };
```

Furthermore, we make the following assumptions:

- A double is 8 bytes.
- The system is a 64-bit machine (i.e., pointers hold 8 bytes) with a single cache of size 64 bytes.
- The cache is write-back/write-allocate.
- The cache has (number of sets, associativity, block size) $= (S, E, 16)$, i.e., the block size is 16 bytes.
- The cache is initially empty.
- The addresses of all nodes are divisible by 16 (16 byte aligned).
- Memory address 0 maps to the first block of the cache.

Now assume the execution of the code below on the following list:



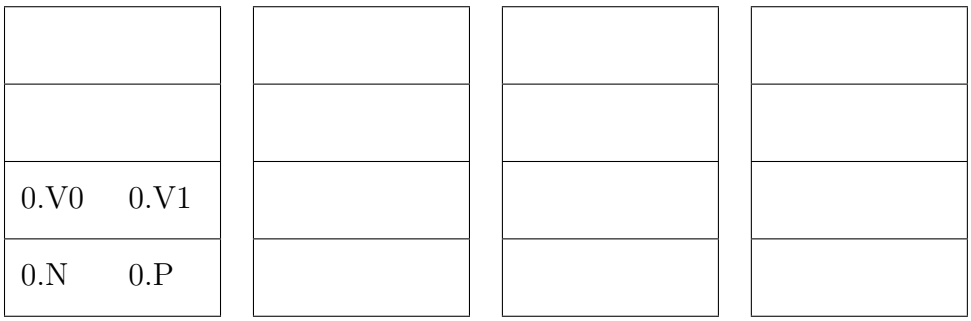
where every node is labeled with a number from 0 to 3.

```
1  struct Node* p = head; // head == 32
2  while (p->next != NULL) {
3    p->values[0] = p->values[0] + p->values[1];
4    p = p->next;                                  // SHOW CACHE
5  }
6  p->values[0] = p->values[0] + p->values[1];     // SHOW CACHE
7
8  while (p->prev != NULL) {
9    p->values[0] = p->values[0] - p->values[1];
10   p = p->prev;                                  // SHOW CACHE
11 }
12 p->values[0] = p->values[0] - p->values[1];     // SHOW CACHE
```

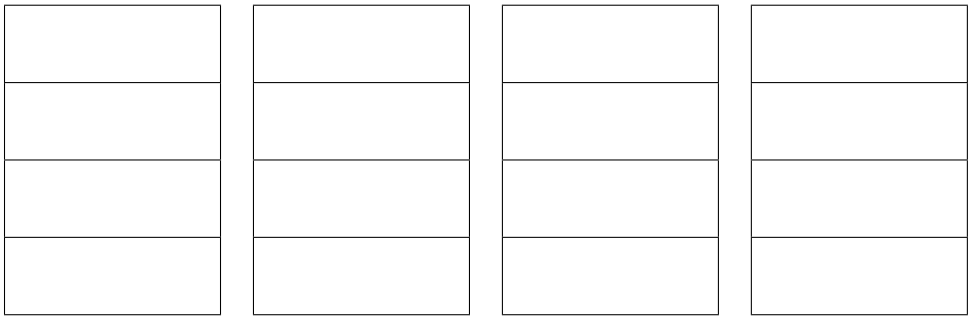
Assuming that pointers p and head are kept in register answer the following questions.

1. Assuming the cache has $S = 4$ and $E = 1$, execute the code above and show the status of the cache whenever requested (right after the execution of lines 4, 6, 10, and 12; so 8 times total). We draw the caches below so you can just fill in. Also provide the hit-miss sequence for the accesses right before the cache status under each cache.

   The first snapshot is shown as an example and for the notation you should use. 0 is the node label, $V0, V1$ are the values, P, N are prev and next. The second snapshot should be filled in the cache to the right of the first one.



MMHHH



2. Now assume the cache has $S = 2$ and $E = 2$ and an LRU replacement policy. Fill the drawn caches and provide the associated hit/miss sequences as before. Again fill the eight caches row by row (meaning, the second snapshot is to the right of the first one). We provide again the first snapshot as example.

0.V0	0.V1						
0.N	0.P						

MMHHH

Problem 5: TLB (8)

We consider the following CPU

- 4 KB page size
- `double` is 8 bytes
- A fully associative data TLB with 16 entries and LRU replacement.
- Single level, direct mapped cache with 32 sets and 32 bytes line size.

Consider the following code, run with initially cold cache and empty TLB.

```
1 void foo (double *x, int Iterations) { // Iterations is a large number
2   for (int N = 0; N < Iterations; N += 1) {
3     for(int i = 0; i < X; i += S) {
4       x[i] = x[i] * x[i] / 2.0;
5     }
6   }
7 }
```

Replace `X` and `S` in the code by integers such that this function causes more TLB misses than cache misses. Sketch the cache and explain enough so we see your reasoning. Then derive the TLB misses and cache misses as functions of `N` and/or `Iterations`. Assume that `x` is allocated such that it is big enough for your access pattern.

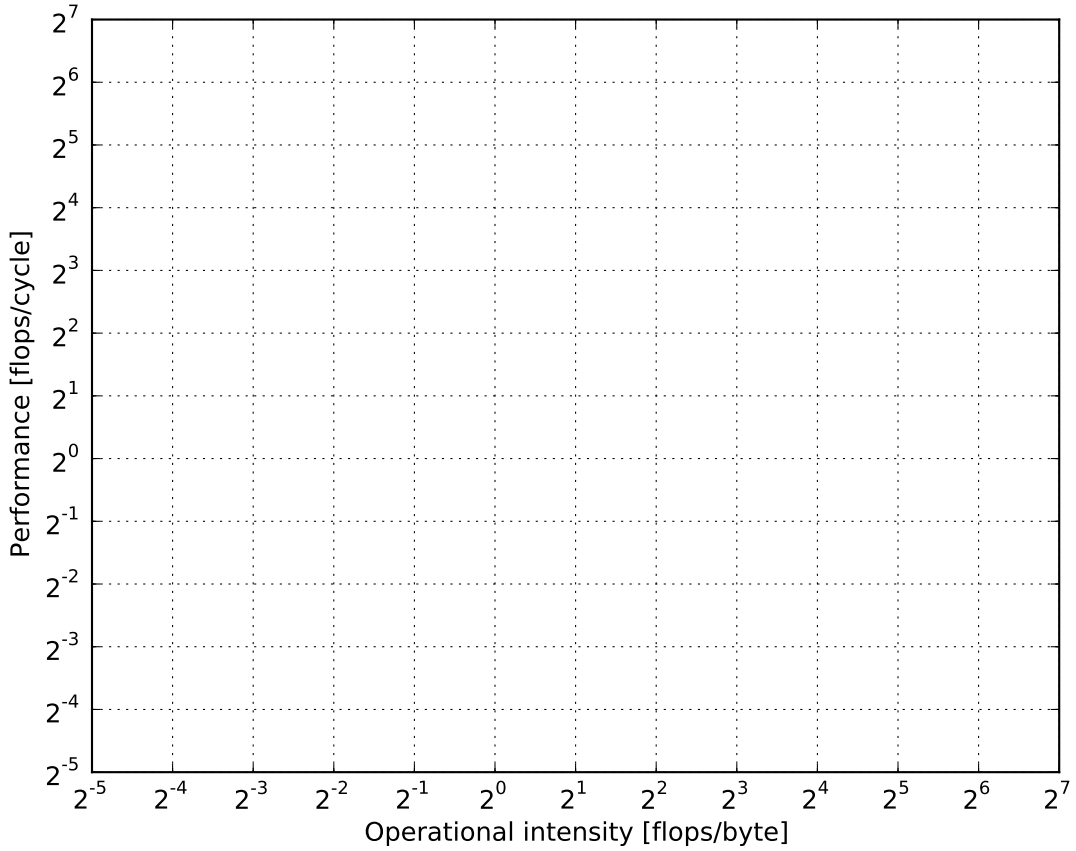
Problem 6 (20=4+8+8 points)

Assume you are using a system with the following features:

- A CPU that can issue 2 double precision multiplications and 1 double precision addition/subtraction per cycle. Latencies don't matter for this exercise.
- The interconnection between CPU and main memory (size 16 GB) has a maximal bandwidth of 8 bytes/cycle.
- The last level cache is write-allocate/write-back, direct mapped, has size 4 MB and block size of 64 bytes.

Answer the following two questions:

1. Draw the roofline plot for this system:



2. Consider the following code which computes entries in matrix g using matrix f (f, g have size $n \times n$). Assume that f and g are allocated sequentially one after the other and first block of f maps with first cache line:

```
1  void compute(double **f, double **g, int n) {
2    int i,j;
3    double s;
4    for(i = 0; i < n; i++) {
5      for(j = 0; j < n; j++){
6        s = f[0][i]*f[0][i] + f[0][j]*f[0][j];
7        g[i][j] = 0.5*(s - f[i][j]*f[i][j]);
8      }
9    }
10 }
```

Assume a cold cache, that the operators are left associative (expressions are evaluated from left to right), and that a double takes 8 bytes. Now compute for $n = 256$,

(a) The operational intensity (ignore write-backs).

(b) An upper bound (as tight as possible) for performance on the specified system.

You are allowed to make minor approximations. Show your work.

3. Now consider the following code which computes entries in matrix h using matrix f and g (f, g, h have size $n \times n$). Assume that f, g and h are allocated sequentially one after the other and first block of f maps with first cache line:

```
1  void compute(double **f, double **g, double **h, int n) {
2    int i,j,k;
3    for(i = 0; i < n; i++){
4      for(j = 0; j < n; j++){
5        for(k = 0; k < n; k++){
6          h[i][j] = h[i][j] + 0.25*g[i][k] + 0.75*f[k][j];
7        }
8      }
9    }
10 }
```

Assume a cold cache, that the operators are left associative (expressions are evaluated from left to right), and that a double takes 8 bytes. Now compute for $n = 1024$,

(a) The operational intensity (ignore write-backs).

(b) An upper bound (as tight as possible) for performance on the specified system.

You are allowed to make minor approximations. Show your work.