ETH login ID: | | | | | | | | | | | |
(Please print in capital letters)

Full name: ____________________

263-2300: How to Write Fast Numerical Code
ETH Computer Science, Spring 2014
Midterm Exam
Monday, April 14, 2014

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 (12)	
Problem 2 (15 = 3 + 12)	
Problem 3 (17 = 6 + 5 + 6)	
Problem 4 (15 = 3 + 3 + 3 + 3 + 3)	
Problem 5 (17 = 14 + 3)	
Problem 6 (24 = 3 + 3 + 3 + 3 + 12)	
Total (100)	

Problem 1 (12 points)

The function `vecsum` implements $z = x + y$, where z, x, and y are vectors of length N.

```
void vecsum(double * z, const double * x, const double * y, size_t N) {
  int i;
  for (i = 0; i < N; i++)
    z[i] = x[i] + y[i];
}
```

We make the following assumptions:

- The peak performance of the CPU is $\pi = 1$ add/cycle.
- The system has two levels of cache.
- All caches are write-back/write-allocate.
- L1 cache size: 64 kB; L1 read bandwidth: 1 double/cycle.
- L2 cache size: 2 MB; L2 read bandwidth: 0.5 double/cycle.
- RAM read bandwidth: 0.25 double/cycle.
- The variables i and N are stored in registers.
- A double is 8 bytes.

The performance of `vecsum` is measured as average over many executions. Sketch the expected performance plot for N up to 200'000. N is on the x-axis and the y-axis shows the percentage of peak performance (between 0% and 100%) achieved. Provide enough details and also short explanations so we can verify your reasoning.

Problem 2 (15 = 3 + 12 points)

Consider the following code, which processes an $M \times N$ matrix A. (Note that for this question it does not matter what the function does.)

```
void func(float A[M][N], float th) {

  int i,j,k,l;
  float r,c,t;

  srand(time(NULL));

  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++) {

      r = c = 0.f;

      for (k = j+1; k < N; k++) {
        t = (float)(rand())/RAND_MAX;
        c += t*a[i][k];
      }
      for (l = i+1; l < M; l++) {
        t = (float)(rand())/RAND_MAX;
        if (t > th)
          r += a[l][j];
        else
          r -= a[l][j];
      }

      a[i][j] += c*r;
    }
}
```

1. Define a detailed floating point cost measure $C(M, N)$ for the function `func`. Ignore integer operations, function calls, and comparisons.
2. Compute the cost $C(M, N)$.

Note: Lower-order terms (and only those) may be expressed using big-O notation (this means: as the final result something like $3n + O(\log(n))$ would be ok but $O(n)$ is not).

The following formula may be helpful: $\sum_{i=0}^{n-1}(n-i) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + O(n)$.

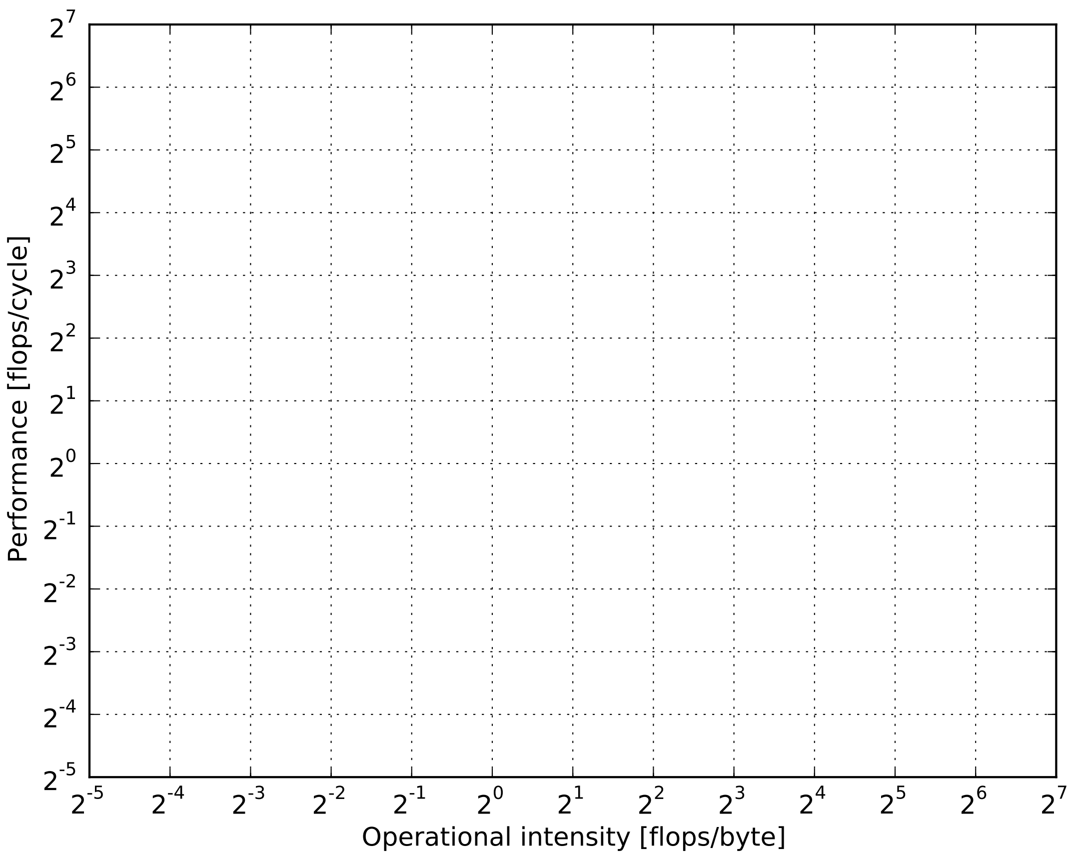
Problem 3 (17 = 6 + 5 + 6 points)

Assume you are using a system with the following features:

- A CPU that can issue 3 double precision multiplications and 1 double precision addition per cycle.
- The interconnection between CPU and main memory has a maximal bandwidth of 4 bytes/cycle.

Answer the following two questions:

1. Draw the roofline plot for this system. The units for x-axis and y-axis are performance in flops/cycle and operational intensity in flops/byte, both in log scale. The plot will contain two lines determining upper bounds on the achievable performance.



2. Consider the following code:

```
void  filter(double m[64][64], double r[62][62]) {

  for(i = 1; i < 63; i++)
    for(j = 1; j < 63; j++)
      r[i-1][j-1] = r[i-1][j-1]
                  + m[i-1][j-1] - m[i-1][j] + m[i-1][j+1]
                  - 2*m[i][j-1] + 2*m[i][j+1]
                  + m[i+1][j];

}
```

Assuming a cold write-back/write-allocate cache with block size $B > 8$ bytes and that the cache can hold the whole matrices m and r, compute the following. You can approximate 62 with 64 where convenient:

(a) The operational intensity of this code (ignore write-backs).

(b) An upper bound (as tight as possible) on performance.

Show your work.

Problem 4 ($15 = 3 + 3 + 3 + 3 + 3$ points)

Mark the following statements as true (T) or false (F). Explanations are not needed. We denote with $I(n)$ the operational intensity of a function executed on some input of size n. Wrong answers give negative points but you cannot get less than 0 points for this problem. You can leave questions unanswered.

☐ Doubling the cache size doubles I.

☐ Doubling the cache size can increase I.

☐ Assume that we can compute the cost (flop count) of an algorithm for a certain input of size n. Then, assuming a cold-cache scenario, we can compute a valid (possibly loose) upper bound for $I(n)$ for all possible C functions that implement this algorithm executed on that input.

☐ A function with a cost in $O(n^2)$ is certainly compute bound.

☐ A function with $I(n) \in O(1)$ is certainly memory bound.

Problem 5 (17 = 14 + 3 points)

Consider the following code for a matrix-matrix multiplication ($c = ab + c$):

```
// a, b, c are n x n matrices (data type double)
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

Assume the code is run on a system with a last-level cache of size C bytes and with a cache block size of $B = 32$ bytes. Further, we assume that $40n < C < 8n^2$.

1. Estimate the number of cache misses as a function of n. Ignore accesses to the matrix c and possible conflict misses.

2. Compute the operational intensity $I(n)$ based on the previous result.

Problem 6 ($24 = 3 + 3 + 3 + 3 + 12$ points)

We define an image as follow (of course you know that `sizeof(char) == 1`):

```
typedef struct {
  unsigned char   red;
  unsigned char   green;
  unsigned char   blue;
  unsigned char   alpha;
} Pixel;

Pixel image[N][16];
```

And we consider a system based on the following assumptions:

- The system has a write-back/write-allocate cache of size C bytes.
- The cache is 4-way set-associative.
- The cache block size is $B = 64$ bytes.
- The cache uses an LRU replacement policy.

Answer the following questions (note: the image is read Pixel by Pixel):

1. What is the miss rate if we read the whole `image` row-wise and $C = 64N$ bytes?

2. What is the miss rate if we read the whole `image` row-wise and $C = 4N$ bytes?

3. What is the miss rate if we read the whole `image` column-wise and $C = 64N$ bytes?

4. What is the miss rate if we read the whole `image` column-wise and $C = 4N$ bytes?

5. We now assume an image of size 4-by-4 pixels and a small cache with block size $B = 8$ bytes, one set, and a total size of 16 bytes. Provide the hit/miss sequence for the following code. All assignments are executed processing first the right-hand side from left to right. The entire sequence will have a length of $4 \times 6 = 24$.

```
int i,j;
for(i = 1; i < 3; i++)
  for(j = 1; j < 3; j++) {
    image[i][j].alpha = image[i-1][j].alpha * image[i][j-1].alpha;
    image[i][j].red   = image[i-1][j].red * image[i][j-1].red;
  }
```

Note it helps to sketch the cache and image.

bla