263-2300-00: How To Write Fast Numerical Code
Assignment 3: 100 points
Due Date: Th, March 19th, 17:00
http://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring15/course.html
Questions: fastcode@lists.inf.ethz.ch

Submission instructions (read carefully):

- (Submission)
  Homework is submitted through the Moodle system https://moodle-app2.let.ethz.ch/course/view.php?id=1317.
  Before submission, you must enroll in the Moodle course. Enrollment key is "263-2300".

- (Late policy)
  You have 3 late days, but can use at most 2 on one homework, meaning submit latest 48 hours after the due
  time. Note that each homework will be available for submission on the Moodle system 2 days after the deadline.
  However, if the accumulated time of the previous homework submissions exceeds 3 days, the homework will
  not count.

- (Formats)
  If you use programs (such as MS-Word or Latex) to create your assignment, convert it to PDF and name
  it homework.pdf. When submitting more than one file, make sure you create a zip archive that contains all
  related files, and does not exceed 10 MB. Handwritten parts can be scanned and included or brought (in time)
  to Alen's or Daniele's office. Late homeworks have to be submitted electronically.

- (Plots)
  For plots/benchmarks, be concise, but provide necessary information (e.g., compiler and flags) and always
  briefly discuss the plot and draw conclusions. Follow (at least to a reasonable extent) the small guide to
  making plots (soon in lecture).

- (Neatness)
  5% of the points in a homework are given for neatness.

Exercises:

1. *Cache mechanics (12 pts)* Consider an 8-way, 32KB cache with a cache block size of 64 bytes. Assume
   64 GB of byte-addressable RAM, on a 64-bit machine.

   (a) Determine the number of cache sets.

   (b) Determine the length of the cache tag for each cache line. How many of those bits will be effectively
       used on this particular machine ?

   (c) For a given `char * p` with address `0xDE147BA` (in hexadecimal format) calculate the cache tag
       value, set index, and block offset. Express the obtained values in hexadecimal format.

2. *Cache mechanics (12 pts)* Consider a direct mapped cache of size 16KB with block size of 16 bytes.
   Furthermore, the cache is write-back and write-allocate. Remember that `sizeof(int) == 4`. Assume
   that the cache starts empty and that local variables and computations take place completely within
   the registers and do not spill onto the stack.

   Now consider the following two implementations of a horizontal flip and copy of the matrix. Assume
   that the `src` matrix starts at address 0 and that the `dest` matrix follows immediately follows it.

   (a)
   ```
   void copy_n_flip_matrix1(int dest[ROWS][COLS], int src[ROWS][COLS]) {
       int i, j;
       for (i = 0; i < ROWS; i++)
         for (j = 0; j < COLS; j++)
           dest[i][COLS - 1 - j] = src[i][j];
   }
   ```

       i. What is the cache miss rate if ROWS = 64 and COLS = 64?

       ii. What is the cache miss rate if ROWS = 96 and COLS = 64?

(b)
```
void copy_n_flip_matrix2(int dest[ROWS][COLS], int src[ROWS][COLS]) {
    int i, j;
    for (j = 0; j < COLS; j++)
        for (i = 0; i < ROWS; i++)
            dest[i][COLS - 1 - j] = src[i][j];
}
```

    i. What is the cache miss rate if ROWS = 64 and COLS = 64?

    ii. What is the cache miss rate if ROWS = 96 and COLS = 64?

3. *Cache mechanics (20 pts)* In this problem, you will compare the performance of direct mapped and 4-way associative caches for the initialization of 2-dimensional arrays of data structures. Both caches have a size of 1024 bytes. The direct mapped cache has 64-byte blocks while the 4-way associative cache has 32-byte blocks. You are given the following definitions:

```
typedef struct{
    float irr[3];
    short theta;
    short phi;
} photon_t;
photon_t surface[16][16];
register int i, j, k;
```

Also assume that

- `sizeof(short) = 2` and `sizeof(float) = 4`
- `surface` begins at memory address 0
- Both caches are initially empty
- The array is stored in row-major order
- Variables i, j, k are stored in registers and any access to these variables does not cause a cache miss.

(a)
```
for (i = 0; i < 16; i ++) {
    for (j = 0; j < 16; j ++) {
        for(k = 0; k < 3; k ++) {
            surface[i][j].irr[k] = 0.;
        }
        surface[i][j].theta = 0;
        surface[i][j].phi = 0;
    }
}
```

    i. What fraction of the writes in the above code will result in a miss in the direct mapped cache?

    ii. What fraction of the writes will result in a miss in the 4-way associative cache?

(b)
```
for (i = 0; i < 16; i ++) {
    for (j = 0; j < 16; j ++) {
        for (k = 0; k < 3; k ++) {
            surface[j][i].irr[k] = 0;
        }
        surface[j][i].theta = 0;
        surface[j][i].phi = 0;
    }
}
```

    i. What fraction of the writes in the above code will result in a miss in the direct mapped cache?

    ii. What fraction of the writes will result in a miss in the 4-way associative cache?

4. *Roofline (25 pts)* Assume the following hardware parameters for an Intel CPU:

- Can issue one scalar add and two scalar multiplications per cycle.

- CPU frequency is 3.5 GHz.
- Last level cache (LLC) size is 8 MB and cache block size is 64 bytes.
- Memory bandwidth is 28 Gbyte/sec.

Draw a roofline plot for single precision floating point operations on the given hardware. The units for x-axis and y-axis are flops/byte and flops/cycle, respectively. Specifically, the plot should contain 2 lines:

(a) Upper bound based on peak performance π.

(b) Upper bound based on the maximal memory bandwidth β.

Provide enough detail (labels etc.) so we can check correctness.

Now consider running the following code on the platform above (all the matrices have size $N \times N$):

```
void compute1(float *A, float *B, float *C, size_t N) {
  int i,j,k;
  for(i = 0; i < N; i++)
    for(j = 0; j < N; j++)
      for(k=0; k < N; k++)
          C[i][j] = 0.4*C[i][j] + 0.6*A[i][k]*B[j][k];
}
```

(c) Is it possible to reach peak performance for `compute1`? If not, include a tighter performance bound specific for `compute1` in your roofline plot.

Again consider running the following code on the platform above (all the matrices have size $N \times N$):

```
void compute2(float *A, float *B, float *C, size_t N) {
    int i,j,k;
    for(i = 0; i < N; i++)
      for(j = 0; j < N; j++)
        for(k=0; k < N; k++)
            C[i][j] = C[i][j] + 0.6*A[i][k]*B[j][k];
}
```

(d) Is it possible to reach peak performance for `compute2`? If not, include a tighter performance bound specific for `compute2` in your roofline plot.

Finally consider the following code (all the matrices have size $N \times N$):

```
void compute3(float *A, float *B, float *C, size_t N) {
    int i,j,k;
    for(i = 0; i < N; i++)
      for(j = 0; j < N; j++)
        for(k=0; k < N; k++)
            C[i][j] = C[i][j] + 0.6*A[k][i]*B[k][j];
}
```

(e) Can the execution of `compute3(A,B,C,65536)` reach peak performance? If not, include a tighter performance bound specific for this execution of `compute3` in your roofline plot.

5. *Roofline and MMM (26 pts)*
   We consider a processor with the following parameters:

   - π: Peak performance in flops/cycle.

- β: Peak bandwidth in bytes/cycle.
- γ: Cache size in bytes (there is only one cache).

(a) Assume a function that is run for a given input size N. Show that if the dot of this function in the roofline plot is on the ridge point (the point where the performance bound and the bandwidth bound intersect) then $T_{\mathrm{comp}} = T_{\mathrm{mem}}$. Here, T_{comp} is the time required to execute the floating point ops, and T_{mem} is the time to transfer the needed data from and to memory. Under which condition is the reverse also true?

(b) Assume an implementation of matrix-matrix multiplication (MMM) with cost $W(N) = 2N^3$ flops for square matrices of size $N \times N$. Also assume that this implementation incurs the minimal number of cache misses possible and the dot in the roofline plot for all sizes lies on the ridge point. Further, assume now the use of another CPU with the same parameters except for peak performance, which is $\pi' = \alpha\pi$ flops/cycle, with $\alpha > 1$. On this CPU the MMM function will be memory bound. By which factor should the cache size be increased to bring the dot back to the ridge point? Explain.