

## 263-2300-00: How To Write Fast Numerical Code

Assignment 4: 120 points

Due Date: Th, April 11th, 17:00

<https://acl.inf.ethz.ch/teaching/fastcode/2019/>

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=10968>.
- (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. For example, submitting 1 hour late costs 1 late day. 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, Tyler's or Gagandeep's office.
- (Plots)  
For plots/benchmarks, **provide (concise) necessary information for the experimental setup (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 from the lecture.
- (Code)  
When compiling the final code, ensure that you use optimization flags (e.g. for GCC use the flag "-O3").
- (Neatness)  
5 points in a homework are given for neatness.

### Exercises:

1. *Cache mechanics (30 pts)* Consider the following code, executed on a machine with a write-back and write-allocate direct-mapped cache with blocks of size 16 bytes and a total capacity of 4096 bytes. This code takes a matrix and for every element, adds to it the element in the row above it. Memory accesses occur in exactly the order that they appear. The variables  $i, j, n, a$ , and  $b$  remain in registers and do not cause cache misses. Assume that  $A$  starts at address 0, and  $n$  is divisible by 4.

```
1 void add_rows(float * A, int n) {
2     for (int i = 1; i < n; i++) {
3         for (int j = 0; j < n; j++) {
4             float a = A[i*n + j];
5             float b = A[(i-1)*n + j];
6             A[i*n + j] = a + b;
7         }
8     }
9 }
```

Counting cache misses from both reads and writes, answer the following:

- (a) What is the cache miss rate if  $n = 256$ ?
- (b) What is the cache miss rate if  $n = 2048$ ?
- (c) What is the cache miss rate if  $n = 2052$ ?

2. *LRU vs MRU (20 pts)* When loading a block into cache, set-associative caches require a *policy* to determine which block to evict if the set is already full. A cache with a *least-recently used* (LRU) policy will evict the block that has been used least recently. Another possible heuristic is that of *most-recently used* (MRU), which will evict the block that has been used most recently. If the set is not already full, the policy does not evict anything.

Consider a 2-way set-associative write-back and write-allocate cache with blocksize 16 bytes. Assume that the cache is empty at the beginning of an operation. Is the cache miss rate for LRU always better than MRU? If yes, provide a proof. If no, provide a counter-example and provide enough detail to explain why MRU is better in this case.

3. *Rooflines (40 pts)* You are given a computer with the following parameters:

- It has a SIMD vector length of 4 single-precision floats.
- It has two execution ports that can execute floating point operations. Behind each port there is one execution unit that executes multiplications, one that executes additions, and one that executes FMAs.
- All execution units have a throughput of 1 operation/cycle for both scalar and vector operations.
- Each instruction has a latency of 1 cycle.
- The frequency of the CPU is 1 GHz.
- It has a read bandwidth from main memory of 12 GB/s.

- (a) Draw a roofline plot for the machine. Consider only single-precision floating point arithmetic. Consider only reads. Include a roofline for when vector instructions are not used and for when vector instructions are used.
- (b) Consider the following functions. For each, assume that vector instructions are not used, and derive hard upper bounds on its operational intensity and performance based on its instruction mix and compulsory misses. Ignore the effects of aliasing. Assume you write code that attains these bounds, and add the performance to the roofline plot (there should be two dots).

```

1 void computation_1(float *x1, float *x2, float *y1, float *y2,
2   float a, float b, float c, float d, int n)
3 {
4   for(int i = 0; i < n; i++) {
5     x1[i] = x1[i] + a * y1[i] + b * y2[i];
6     x2[i] = x2[i] + c * y1[i] + d * y2[i];
7   }
8 }

```

```

1 void computation_2(float *x1, float *x2, float *y1, float *y2,
2   float a, float b, float c, float d, int n)
3 {
4   for(int i = 0; i < n; i++) {
5     x1[i] = x1[i] + a + y1[i] + b + y2[i];
6     x2[i] = x2[i] + c + y1[i] + d + y2[i];
7   }
8 }

```

- (c) For each computation, what is the maximum speedup you could achieve by parallelizing it with vector intrinsics?

4. *Balance Principles (25 pts)* The I/O cost ( $Q$  in class) of an algorithm is the amount of data that is read from and written to main memory during the execution of the algorithm. The ordinary matrix-matrix multiplication (MMM) operation  $C := AB + C$  takes  $2n^3$  flops. When  $n$  is large and ignoring lower-order terms, the optimal I/O cost when executing MMM is  $\frac{2n^3}{\sqrt{m}}$  matrix elements transferred, where  $m$  is the number of elements that can fit in cache.

- (a) Given a machine with a peak performance of  $\pi$  double-precision flops/cycle and a cache size of  $\gamma$  bytes, what is the minimum value of  $\beta$  such that double-precision MMM can achieve peak performance, where  $\beta$  is the memory bandwidth in bytes/cycle?
- (b) If  $\gamma$  is doubled and  $\pi$  stays fixed, how does this change the minimum value of  $\beta$  required for MMM to achieve peak performance?
- (c) If  $\pi$  is doubled and  $\beta$  stays fixed, how does this change the minimum value of  $\gamma$  required for MMM to achieve peak performance?