263-0007-00: Advanced Systems Lab
Assignment 4: 120 points
Due Date: April 14th, 17:00
https://acl.inf.ethz.ch/teaching/fastcode/2022/
Questions: fastcode@lists.inf.ethz.ch

Academic integrity:
All homeworks in this course are single-student homeworks. The work must be all your own. Do not copy any parts of any of the homeworks from anyone including the web. Do not look at other students' code, papers, or exams. Do not make any parts of your homework available to anyone, and make sure no one can read your files. The university policies on academic integrity will be applied rigorously.

Submission instructions (read carefully):

- (Submission)
  Homework is submitted through the Moodle system https://moodle-app2.let.ethz.ch/course/view.php?id=17306.
- (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 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.
- (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.
- (Neatness)
  5 points in a homework are given for neatness.

Exercises

1. *Associativity (15 pts)*

   We consider a 2-way set associative cache given by $(S, E, B) = (8, 2, 64)$, where S is the number of sets and B the block size in bytes. To reduce the number of cache misses we design a second cache given by $(S', E', B') = (1, 16, 64)$. In words, this cache has the same size and block size but it is fully associative instead of 2-way set associative. We assume LRU replacement.

   We claim that for any code the number of cache misses on the second cache is equal or smaller than the number of misses on the first cache. Do you agree with this statement? If yes, provide a proof. If no, provide a counterexample (i.e., sketch a function operating on some array where it does not hold).

2. *Cache Mechanics (35 pts)*

   Consider the following code, executed on a machine with a direct-mapped write-back/write-allocate cache with blocks of size 16 bytes, a total capacity of 128 bytes and with a LRU replacement policy. Assume that memory accesses occur in exactly the order that they appear. The variables `i,j,t1` and `t2` remain in registers and do not cause cache misses. The matrix A is cache-aligned (first element goes into first cache block) and is stored in row major order. For this and the following exercises, assume a cold cache scenario. `sizeof(double)` = `sizeof(uint64_t)` = 8 bytes.

```
1  struct pair_t {
2      double   a;
3      double   b;
4      uint64_t u[3];
```

```
5  };
6
7  void comp(pair_t A[3][3]) {
8      double t1, t2;
9      // First loop
10     for (int i = 0; i < 2; i++) {
11         for (int j = 0; j < 3; j++) {
12             t1 = A[i+1][j].a;
13             t2 = A[i][(j+1)%3].a;
14             A[i][j].b = t1 + t2;
15         }
16     }
17     // Show state of cache at this point
18     // Second loop
19     for (int i = 0; i < 2; i++) {
20         for (int j = 0; j < 3; j++) {
21             t1 = A[(j+1)%3][i].a;
22             t2 = A[j][i+1].a;
23             A[j][i].b = t1 + t2;
24         }
25     }
26     // Show state of cache at this point
27 }
```

(a) Considering cache misses from both **reads and writes**, answer the following. Show your work.

i. Determine the number of hits and misses for executing the first loop (lines 10–16).

ii. Draw the state of the cache after the first loop, i.e, at line 17. See the example below that shows how to draw the cache.

iii. Determine the number of hits and misses for executing the second loop (lines 19–25).

iv. Draw the state of the cache after the second loop, i.e., at the end of the function at line 26.

(b) Repeat the previous task assuming now that the cache is 2-way set associative (the cache size and block size stay the same).

Example. The following example shows how we expect you to draw the cache. The example shows an initially empty cache with $(S, E, B) = (3, 2, 16)$ after $A[0][0].a$ was accessed. Note that this cache is different from the one specified in the exercise.

State of the cache after accessing $A[0][0].a$:

Set	0	1
0	$A_{00}.a$, $A_{00}.b$	
1		
2		

3. *Rooflines (40 pt)* Consider a processor with the following hardware parameters (assume 1GB = 10^9B):

- SIMD vector length of 256 bits.
- The following instruction ports that execute floating point operations:
  - Port 0 (P0): FMA, ADD, MUL
  - Port 1 (P1): FMA, ADD, MUL
  - Port 2 (P2): DIV

  Ports 0 and 1 can issue 1 instruction per cycle and each instruction has a latency of 1. Port 2 can only issue 1 division every two cycles with a latency of two, i.e., $Gap_{\text{div}} = 2$ and $Latency_{\text{div}} = 2$.
- One write-back/write-allocate cache with blocks of size 64 bytes.
- Read bandwidth from the main memory is 42 GB/s.
- Processor frequency is 3 GHz.

(a) Draw a roofline plot for the machine. Consider only double-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 performance and operational intensity (consider only **reads**) based on its **instruction mix** and **compulsory misses**. Assume that $s_1 = s_2 = 1$. Ignore the effects of aliasing and assume that no optimizations that change operational intensity are performed (the computation stays as is). FMAs are used to fuse an addition with a multiplication whenever applicable. All arrays are cache-aligned (first element goes into first cache set) and don't overlap in memory. Assume you write code that attains these bounds, and add the performance to the roofline plot (there should be two dots).

```
void comp1(double *x1, double *x2, double *y, double *z, int s1, int s2, int n) {
  double a = 0.1;
  for (int i = 0; i < n; i++) {
    x1[i] = a * (x1[i] + y[i]) * (x1[i] + z[s1*i]);
    x2[i] = a + (x2[i] + y[i]) * (x2[i] + z[s2*i]);
  }
}
```

```
void comp2(double *x1, double *x2, double *y, double *z, int s1, int s2, int n) {
  double a = 0.1;
  for (int i = 0; i < n; i++) {
    x1[i] = a + (x1[i] * y[i]) + (x1[i] * z[s1*i]);
    x2[i] = a / (x2[i] * y[i]) + (x2[i] * z[s2*i]);
  }
}
```

(c) Follow the same assumptions as in part (b) and derive hard upper bounds on the operational intensity and performance of each function assuming now that $s_1 = s_2 = 0$ (without vectorization). Add the new performance of each function to the roofline plot (two additional dots).

(d) For each computation in part (c), i.e., assuming that $s_1 = s_2 = 0$, what is the maximum speedup you could achieve by parallelizing it with vector intrinsics?

(e) Repeat part (b) assuming that $s1 = 2$ and $s2 = 16$. Add the new performance of each function to the roofline plot (two additional dots).

4. *Cache Miss Analysis (25 pts)*

Consider the following computation that performs a matrix multiplication $C = C + AB$ of square matrices A, B and C of size $n \times n$ using a k-i-j loop.

```
void mmm_kij(double *A, double *B, double *C, int n) {
    for(int k = 0; k < n; k++)
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                C[n*i + j] += A[n*i + k] * B[n*k + j];
}
```

Assume that the code is executed on a machine with a write-back/write-allocate fully-associative cache with blocks of size 64 bytes, a total capacity of γ doubles and with a LRU replacement policy. Assume that n is divisible by 8, cold caches, and that all matrices are cache-aligned.

(a) Assume that $\gamma << n$ and answer the following. Justify your answers.

i. Determine, as precise as possible, the total number of cache misses that the computation has.

ii. For each of the matrices (A, B and C), state the kind(s) of locality it benefits from to reduce misses.

(b) Repeat the previous task assuming now that $\gamma = 3n$ doubles.

(c) Determine the minimum value for γ, as precise as possible, such that the computation only has compulsory misses. For this, assume that LRU replacement is not used and, instead, cache blocks are replaced as effectively as possible to minimize misses.