263-0007-00: Advanced Systems Lab
Assignment 4: 120 points
Due Date: Th, April 9th, 17:00
https://acl.inf.ethz.ch/teaching/fastcode/2020/
Questions: fastcode@lists.inf.ethz.ch

1. *Cache Mechanics (40 pts)*

   Consider the following functions, executed on a machine with a write-back/write-allocate 2-way set associative cache with blocks of size 64 bytes, a total capacity of 4096 bytes and with a LRU replacement policy. These functions take a matrix A with $m = 2112$ columns and n rows. The largest element in each column of A is stored in an array v of size m. Memory accesses occur in exactly the order that they appear. The variables `i,j,m` and `n` remain in registers and do not cause cache misses. Array v is aligned to the cache block and the first element of A is immediately after the last element of v in memory, i.e., `A = v + m`. For this and the following exercises, assume a cold cache scenario.

```
1 void calculate1(float *A, float* v, int m, int n) {
2     for (int i = 0; i < n; ++i) {
3         for (int j = 0; j < m; ++j) {
4             v[j] = max(v[j], A[i * m + j]);
5         }
6     }
7 }
```

```
1 void calculate2(float *A, float* v, int m, int n) {
2     for (int j = 0; j < m; ++j) {
3         for (int i = 0; i < n; ++i) {
4             v[j] = max(v[j], A[i * m + j]);
5         }
6     }
7 }
```

   Counting cache misses from both reads and writes, answer the following (assume $m = 2112$):

   (a) What is the cache miss rate if $n = 8$ for both functions?

   Solution:

   i. In both functions, the number of memory accesses is $3mn$. In `calculate1`, array A is accessed sequentially and thus benefits from spacial locality. Since a cache block can store 16 floats, accessing array A produces $\frac{mn}{16}$ misses in total. Similarly, accessing array v produces $\frac{m}{16}$ misses in the first iteration of the outer loop. During the second and following iterations, the first half of v has been already evicted from the cache. Thus, we will not benefit from temporal locality and the total miss rate when accessing v is $\frac{mn}{16}$. The total miss rate of `calculate1` is therefore $\frac{\mathbf{1}}{\mathbf{24}} \approx \mathbf{0.041}$.

   ii. In `calculate2`, array A has a stride access pattern in the inner loop but there are no cache conflicts. Thus, the number of misses remains $\frac{mn}{16}$. Accesses to array v benefit from temporal locality in the inner loop, producing $\frac{m}{16}$ misses in total. The miss rate is therefore $(\frac{8m}{16} + \frac{m}{16}) \cdot \frac{1}{3 \cdot 8 \cdot m} = \frac{\mathbf{3}}{\mathbf{128}} \approx \mathbf{0.023}$.

   (b) What is the cache miss rate if $n = 16$ for both functions?

   Solution:

   i. The miss rate remains the same for `calculate1`, i.e., $\frac{\mathbf{1}}{\mathbf{24}}$.

   ii. In function `calculate2`, array v benefits from temporal locality in the inner loop, producing $\frac{m}{16}$ misses in total. In the first 16 iterations of the outer loop, the first elements of v are loaded into set 0 of the cache and the elements read from matrix A are loaded to the sets $4, 8, 12, ..., 28, 0, 4..., 28, 0$. Since the cache is 2-way set associative, there is a conflict in set 0 (array v, row 8 and row 16 of A). Thus, the $8th$ and $16th$ rows of A always produce misses ($2m$ in total). The remaining 14 rows generate $\frac{14m}{16}$ misses in total. The miss rate is therefore $(\frac{m}{16} + \frac{14m}{16} + 2m) \cdot \frac{1}{3 \cdot 16 \cdot m} = \frac{\mathbf{47}}{\mathbf{3 \cdot 16^2}} \approx \mathbf{0.061}$.

(c) What is the minimum value of n in which function `calculate2` achieves its highest miss rate?

Solution:

Function `calculate2` achieves its highest miss rate when reading from matrix A results always in a cache miss. Starting from $n = 23$, the elements read from matrix A during the first 16 iterations are loaded to the sets $4, 8, ..., 28, 0, 4..., 28, 0, 4, ..., 28$. There are cache conflicts when accessing all rows of A. The miss rate in this case is $(\frac{m}{16} + 23m) \cdot \frac{1}{3 \cdot 23 \cdot m} \approx \frac{1}{3}$. Thus, function `calculate2` achieves its highest miss rate when $\mathbf{n \geq 23}$.

(d) Suggest an improvement over `calculate2` that achieves a lower miss rate than `calculate1` for the n that you calculated in (c).

Solution:

Implementing blocking over the rows of A. For example, processing A in $8 \times m$ blocks would achieve a similar miss rate than in part (a), i.e. around $\mathbf{\frac{3}{128} \approx 0.023}$.

2. *Stride access (20 pts)*

Consider the following function, executed on the same machine as in the previous exercise. The array v is aligned with the cache block size. Memory accesses occur in exactly the order that they appear in the code. Thus, no optimizations are performed that reduce the memory accesses or reorder computations. The variables `i,j,n,sum` and `stride` remain in registers and do not cause cache misses.

```
1  double calculate(double *A, double* v, int n, int stride)
2  {
3      double sum = 0;
4      for (int j = 0; j < 2; ++j) {
5          for (int i = 0; i < n; ++i) {
6              sum += v[i*stride];
7          }
8      }
9      return sum;
10 }
```

Counting cache misses answer the following:

(a) Assuming $stride = 4$, $n > 1$ and n even, what is the maximum value of n that achieves the highest hit rate?

Solution:

Working with a two-power-strided working set is like having a smaller cache. The cache can store 512 doubles and has block size of 8 doubles. With $stride = 4$, function `calculate` presents spatial locality and temporal locality when $n \leq \frac{512}{4} = \mathbf{128}$. For this case, the first iteration of the outer loop generates one miss and one hit (spacial locality) for every two elements loaded. The second iteration generates only hits (temporal locality) since all data is in the cache. The hit rate is therefore $\frac{3}{4}$.

(b) Under the same assumptions, what is the minimum value of n that achieves the lowest hit rate.

Solution:

As already mentioned, the outer loop always generates one miss and one hit for every two elements loaded. The lowest hit rate is found when for the second iterations, the data to be accessed is no longer in the cache (no temporal locality). This happens when $n \geq 191$. Since the problem specifies to assume n even, the minimum value is $n = \mathbf{192}$. In this case, the first 64 elements accessed in the first iteration of the outer loop are evicted from the cache and have to be loaded again in the second iteration. This process will evict the next 64 elements due to the LRU policy. The process is repeated, generating a hit rate of $\frac{1}{2}$.

(c) Repeat parts (a) and (b), now assuming $stride = 32$.

Solution:

i. Similar to (a), with $stride = 32$, function `calculate` presents temporal locality when $n \leq \frac{512}{32} = \mathbf{16}$. It doesn't have spatial locality. Thus, the first iteration of the outer loop generates only misses. The second iteration generates only hits (temporal locality) since all data is in the cache. The highest hit rate is therefore $\frac{1}{2}$.

ii. The lowest hit rate is found when for the second iterations, the data to be accessed is no longer in the cache (no temporal locality). This happens when $n \geq \mathbf{24}$. In this case, all accesses are misses in the first iteration. In the second iteration, the data is no longer in the cache. Thus, every memory access is a miss and the hit rate is zero.

(d) Assuming $stride = 16$ and $n = 40$. Determine the hit rate of the function.

Solution:
There are 80 memory accesses in total. There are no hits in the first iteration of the outer loop. In the second iteration, the first 8 accesses are misses, then 8 hits, then 8 misses, and the pattern continues. In total there are 16 hits and the hit rate is $\frac{16}{80} = \mathbf{\frac{1}{5}}$.

3. *Rooflines (40 pts)*

You are given a computer with the following parameters:

- It has a SIMD vector length of 4 double-precision floats.
- It has three execution ports that can execute floating point operations (P0, P1 and P2). Behind P0 and P1 there is one execution unit that executes multiplications, one that executes additions, and one that executes FMAs. In P2 there is only one execution unit that executes additions. The mapping of execution units to ports is as follows:
  - **P0**: FMA, ADD, MUL.
  - **P1**: FMA, ADD, MUL.
  - **P2**: ADD.
- All execution units have a throughput of 1 operation/cycle for both scalar and vector operations.
- Each instruction has a latency of 1 cycle.
- One cache with 64-byte cache block size.
- It has a read bandwidth from main memory of 32 bytes/cycle.

(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. Ignore the effects of aliasing and assume that no optimizations that change operational intensity are performed (the computation stays as is). For `computation3`, FMAs are used to fuse an addition with a multiplication. Both arrays (x and y) are cache-aligned (first element goes into first cache block). Assume you write code that attains these bounds, and add the performance to the roofline plot (there should be three dots).

```
1 void computation1(double *x, double *y, double c1, double c2, double c3, int n) {
2     for(int i = 0; i < n; i++) {
3         x[i] = x[i] + c1 + y[i] + c2 + y[i+4] + c3;
4     }
5 }
```

```
1 void computation2(double *x, double *y, double c1, double c2, double c3, int n) {
2     for(int i = 0; i < n; i++) {
3         x[i] = x[i] * c1 * y[i] * c2 * y[i+4] * c3;
4     }
5 }
```

```
1 void computation3(double *x, double *y, double c1, double c2, double c3, int n) {
2     for(int i = 0; i < n; i++) {
3         x[i] = x[i] + c1 * y[i] + c2 * y[i+4] + c3;
4     }
5 }
```

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

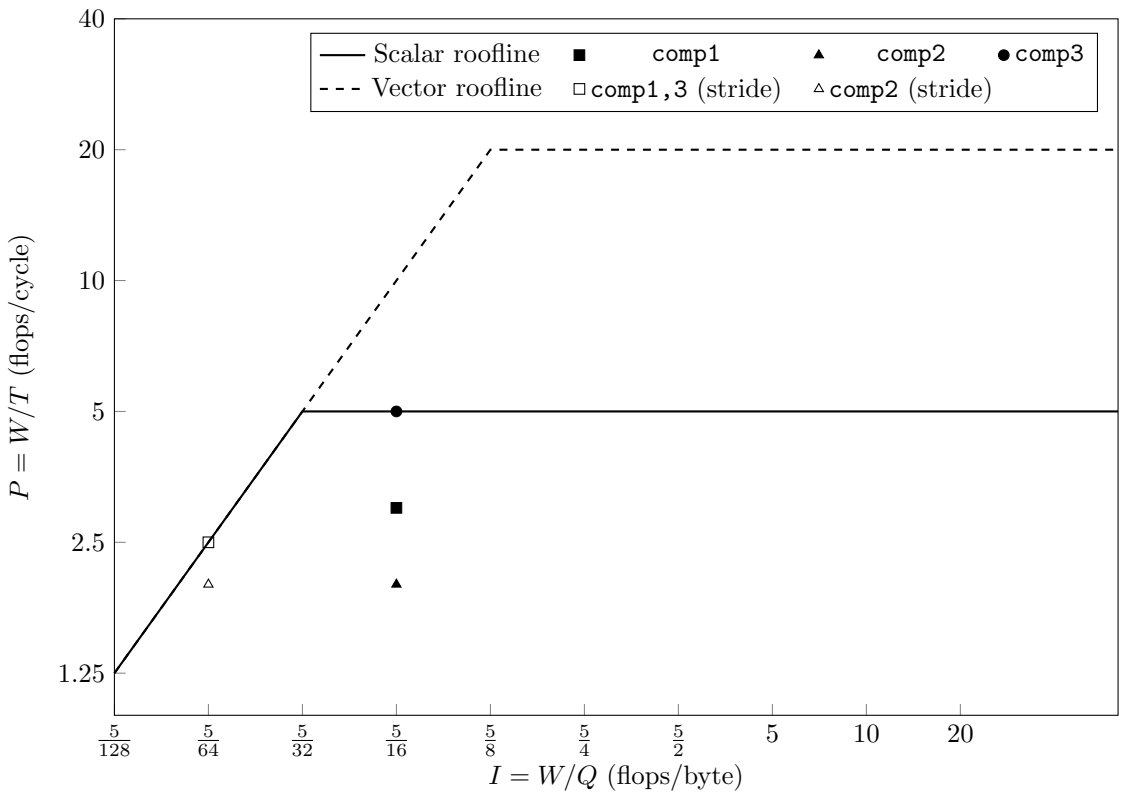
(d) For each of the three functions, consider the following modification in the memory access pattern (strided access). We only show `computation1`, but assume the same modification in `computation2` and `computation3`. Array y has an according larger size.

```
1 void computation1(double *x, double *y, double c1, double c2, double c3, int n) {
2     for(int i = 0; i < n; i++) {
3         x[i] = x[i] + c1 + y[7*i] + c2 + y[7*i+4] + c3;
4     }
5 }
```

Follow the same assumptions as in part (b) and derive hard upper bounds on the operational intensity and performance of each function considering this new access pattern (without vectorization). Add the new performance of each function to the roofline plot (three additional dots).

Solution:

Roofline plot:



(a) The maximum performance is achieved when executing 1 ADD and 2 FMAs per cycle. Thus, 5 flops/cycle and 20 flops/cycle for scalar and vectorized code respectively. The rooflines are $P \leq \pi_s$, $P \leq \pi_v$ and $P \leq I \cdot \beta$, where $\pi_s = 5$, $\pi_v = 20$ and $\beta = 32$.

(b) The number of flops is $W(n) = 5n$. Considering only reads and compulsory misses, a lower bound on the number of bytes transferred is $Q(n) \geq 8(2n + 4) \geq 16n$. The operational intensity is therefore $I(n) \leq \frac{5}{16}$. The performance for `computation1`, `computation2` and `computation3` is 3, 2 and 5 flops/cycle respectively.

(c) The operational intensity stays the same and the computation becomes memory bound. The maximum theoretical performance for $I(n) \leq \frac{5}{16}$ is 10 flops/cycle. The maximum performance of `computation1` and `computation2` and `computation3` based on the instruction mix is 12, 8 and

20 flops/cycle. Thus, `computation2` achieves a speed up of 4. `computation1` and `computation3` reach the roofline (10 flops/cycle). Thus, their speedup is 3.3 and 2 respectively.

(d) With this access pattern and considering that the block size is 64 bytes, $Q(n) \geq 8(8n+4) \geq 64n$. Thus, the operational intensity becomes $I(n) \leq \frac{5}{64}$. Here, the computation becomes memory bound and it hits the roofline for `computation1` and `computation3` with a performance of 2.5 flops/cycle. The performance of `computation2` stays the same (2 flops/cycle).

4. *Skinny matrix-matrix multiplication (15 pts)*

Let A be a $m \times m$ square matrix, and B, C be $m \times n$ rectangular matrices. The matrix-matrix multiplication (MMM) operation $C := AB + C$ takes $2m^2n$ flops. Consider the case of a "skinny" matrix where m is large and $n \leq \sqrt{M}$, where M is the number of elements that can fit in cache. Assume that the elements of the matrices are double-precision floating point numbers. Consider also an algorithm that implements MMM for skinny matrices using a one-dimensional partitioning of A and C as follows:

$$C \rightarrow \begin{pmatrix} C_1 \\ \hline \vdots \\ \hline C_\alpha \end{pmatrix}, \quad A \rightarrow \begin{pmatrix} A_1 \\ \hline \vdots \\ \hline A_\alpha \end{pmatrix}$$

where each C_i is $m_c \times n$ and fits in the cache, each A_i is $m_c \times m$, and α is the number of partitions such as $\alpha = \frac{m}{m_c}$. You can assume that m is multiple of m_c. Then, the algorithm computes each suboperation $C_i \mathrel{+}= A_iB$ by a sequence of outer products using one column of A_i and one row of B at a time. For completeness, we show the pseudo-code to compute one suboperation $C_i \mathrel{+}= A_iB$:

```
1 void suboperation_Ci(Ci, Ai, B):
2     for h = 0 to m-1
3         for j = 0 to m_c-1
4             for k = 0 to n-1
5                 Ci[j][k] += Ai[j][h] * B[h][k]
```

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 its execution. Assume a machine with a single cache and answer the following.

(a) Determine a tight lower bound of the I/O cost (in bytes) for the skinny MMM algorithm considering only compulsory misses, i.e., assuming that everything fits in the cache (no capacity misses) and fully associative cache (no conflict misses). Consider reads and writes. Write the lower-bound as a function of m and n.

Solution:
Considering only compulsory misses, every element of A, B and C has to be loaded from main memory and every element of C is written to main memory. Thus, a lower bound of the I/O cost is $Q(m,n) \geq 8 \cdot (3mn + m^2) = 24mn + 8m^2$

(b) Assume now that the cache can only store one partition C_i, one row of B (n elements) and one element of A at a time. Determine a tighter lower bound of the I/O cost as a function of m, n and M, considering compulsory and capacity misses. Assume a fully associative cache (no conflict misses).
Hint: C is read and written once to main memory, and each element of A is read once. The only matrix that is read multiple times from main memory due to capacity misses is B.

Solution:
Matrix B has to be loaded every time a suboperation is performed to compute a block C_i. Thus, $Q(m,n,M) \geq 8 \cdot (2mn + m^2 + mn(\frac{m}{m_c})) \geq 16mn + 8m^2 + \frac{8m^2n^2}{M}$