263-0007-00: Advanced Systems Lab
Assignment 3: 100 points
Due Date: Th, March 26th, 17:00
https://acl.inf.ethz.ch/teaching/fastcode/2020/
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=12308 and through Code Expert https://expert.ethz.ch/mycourses/SS20/asl for coding exercises.
- (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.
- (Code)
  The code has to be submitted through Code Expert https://expert.ethz.ch/mycourses/SS20/asl.
- (Neatness)
  5% of the points in a homework are given for neatness.

Note:
There is no need to submit anything in Moodle for this homework. All is submitted through Code Expert.

Exercises

1. *Matrix-vector multiplication (50 pts)*

   In this exercise, we consider the following computation that resembles a matrix-vector multiplication operation with some additional computations:

```
1   #define NR 32
2   void kernel_base(double* A, double *x, double *y) {
3     for (int i = 0; i < NR; ++i) {
4       for (int j = 0; j < NR; ++j) {
5         y[i] += (j + 2) * A[i*NR + j] * x[j];
6       }
7     }
8
9     for (int i = 0; i < NR; ++i) {
10      if (y[i] >= 0) {
11        y[i] += 1;
12      }
13    }
14  }
```

Your task is to implement this computation with vector intrinsics using the **AVX2** (with FMA) instruction set. Optimize it as much as you can.

This is part of the supplied code in Code Expert:

- Read and understand the code. It enables you to register functions with the same signature, which will be timed in a microbenchmark fashion.
- You may apply any optimization that produces the same result in exact arithmetic.
- You can create new functions in `kernel.cpp` with the same signature as *kernel_base* and register it to the timing framework through the *register_function* function in `kernel.cpp`. Let it run and, if it verifies, it will print the estimated performance in flops per cycle.
- Implement in function *kernel_fast* the implementation that achieves the best runtime. This is the one that will be autograded by Code Expert.
- For this task, the Code Expert system compiles the code using GCC 8.3.1 with the following flags: `-O3 -fno-tree-vectorize -mavx2 -mfma`
- Don't forget to click on the "Submit" button when you finish your implementation.

Note that your implementation has to be completely vectorized. Thus, you should use vector intrinsics for reading from memory, performing the necessary computations, and for writing the result back to memory. Implementations without vectorization will not be counted as valid.

2. *Walsh-Hadamard transform (45 pts)*

Introduction:
In this exercise we consider the Walsh-Hadamard transform applied to the column vectors of a matrix A, i.e., $C := WHT_n \cdot A$. The Hardamard matrix WHT_n is a $n \times n$ square matrix whose entries are either +1 or -1. For this exerciese we only consider the case of $n = 8$. The Hardamard matrix can be factorized into terms which result in a more efficient implementation of the transformer. For WHT_8, the factors are the following:

$$T_1 = \begin{pmatrix} 1 & & & & 1 & & & \\ & 1 & & & & 1 & & \\ & & 1 & & & & 1 & \\ & & & 1 & & & & 1 \\ 1 & & & & -1 & & & \\ & 1 & & & & -1 & & \\ & & 1 & & & & -1 & \\ & & & 1 & & & & -1 \end{pmatrix} \qquad T_2 = \begin{pmatrix} 1 & & 1 & & & & & \\ & 1 & & 1 & & & & \\ 1 & & -1 & & & & & \\ & 1 & & -1 & & & & \\ & & & & 1 & & 1 & \\ & & & & & 1 & & 1 \\ & & & & 1 & & -1 & \\ & & & & & 1 & & -1 \end{pmatrix}$$

$$T_3 = \begin{pmatrix} 1 & 1 & & & & & & \\ 1 & -1 & & & & & & \\ & & 1 & 1 & & & & \\ & & 1 & -1 & & & & \\ & & & & 1 & 1 & & \\ & & & & 1 & -1 & & \\ & & & & & & 1 & 1 \\ & & & & & & 1 & -1 \end{pmatrix}$$

Using these factors, the Walsh-Hadarman transform applied to the column vectors of a matrix A can be computed as $C := T_3 \cdot (T_2 \cdot (T_1 \cdot A))$. Note that using these factors, the total number of floating point operations that have to be performed is only 4 additions and 4 subtractions for each factor. Thus, 24 flops in total to transform one column vector of A. To illustrate this, we show the four additions and four subtractions needed to compute the result of multiplying the first factor T_1 to a column vector a_i:

$$T_1 \cdot a_i = \begin{pmatrix} 1 & & & & 1 & & & \\ & 1 & & & & 1 & & \\ & & 1 & & & & 1 & \\ & & & 1 & & & & 1 \\ 1 & & & & -1 & & & \\ & 1 & & & & -1 & & \\ & & 1 & & & & -1 & \\ & & & 1 & & & & -1 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix} = \begin{pmatrix} a_1 + a_5 \\ a_2 + a_6 \\ a_3 + a_7 \\ a_4 + a_8 \\ a_1 - a_5 \\ a_2 - a_6 \\ a_3 - a_7 \\ a_4 - a_8 \end{pmatrix}$$

Task:

Your task is to implement the Walsh-Hadamard transform applied to the column vectors of a matrix A. To this end follow these guidelines:

- Implement the Walsh-Hardmard transform using its factors, i.e. implement $C := T_3 \cdot (T_2 \cdot (T_1 \cdot A))$. You may rearrange the factors in any other order of your preference, but you are not allowed to multiply factors by themselves. For example, implementing $C := T_3 \cdot (T_{1,2} \cdot A)$ where $T_{1,2} = T_1 \cdot T_2$ is not allowed.
- Avoid performing the actual matrix-vector multiplication of the column vectors of A with the factors $T_{1,2,3}$. Instead, translate the application of each factor to series of additions and subtractions (as in the example above). At the end, your code should mainly consists of addition/subtraction operations.
- Use vector intrinsics in your implementation using the **AVX2** (with FMA) instruction set. Optimize it as much as you can.
- It is **not allowed** to store elements from different column vectors of A into the same vector variable (e.g. of type `__m256d`). For example, given two column vectors a_j and a_k for arbitrary columns j and k with $j \neq k$, and a vector variable v of type `__m256d`. v should not contain elements of a_j and a_k simultaneously.
- Your implementation has to be completely vectorized. Thus, you should use vector intrinsics for reading from memory, performing the necessary computations, and for writing the result back to memory. Implementations without vectorization will not be counted as valid.

This is part of the supplied code in Code Expert:

- Similar to the previous task, the code enables you to register functions with the same signature, which will be timed in a microbenchmark fashion.
- You can create new functions in `wht.cpp` with the same signature as *wht_base* and register it to the timing framework through the *register_function* function in `wht.cpp`. Let it run and, if it verifies, it will print the estimated performance in flops per cycle.
- You can assume that matrices A and C are stored in column-major order. Thus, the columns of these matrices are contiguous in memory.
- Implement in function *wht_fast* the implementation that achieves the best estimated performance. This is the one that will be autograded by Code Expert.
- For this task, the Code Expert system compiles the code using GCC 8.3.1 with the following flags: `-O3 -fno-tree-vectorize -mavx2 -mfma`

Information regarding Code Expert

- Note that Code Expert will return an estimate of the performance of your optimized functions. This may differ from the real performance since you are allowed to reduce the operation count during your optimizations.
- The result in percentage given by Code Expert does not necessarily translates into the final grade of the exercise. This is just an estimate of the performance of your optimized implementation compared to a baseline (e.g. the theoretical peak performance). Thus, for some cases, it may not be even possible to achieve 100%.

- Don't forget to click on the "Submit" button when you finish an exercise.
- The CPU running the programs submitted to Code Expert is an Intel Xeon Silver 4210 Processor. This is a Cascade Lake processor but you can assume the same latency and throughput information as Skylake.