University of Basel

Lecturer:
Prof. Dr. Florina M. Ciorba

Computer Lab. U1.001
Spiegelgasse 1,
CH-4051 Basel

Assistants:
Aurélien Cavelan, Ph.D.
Danilo Guerrera, M.Sc.
Ahmed Eleliemy, M.Sc.
Ali Mohammed, M.Sc.

http://informatik.unibas.ch/fs2018/lecture-high-performance-computing/

High Performance Computing (17164-02) **Spring Semester 2018**

Assignment 1: Code optimization *(20+ + 3 Points)*

Starting Date: March 01, 2018

Deadline: March 14, 2018 - 23:59:59

Objectives:
1- Develop cache-aware applications.
2- Exploit the vector processing units in a CPU using the compiler optimization flags.
3- Understand how data organization in the memory can affect the application performance.

1 Cache aware programming *(8 Points)*

Given the sequential matrix multiplication program *T1.c*, you are required to develop a cache-aware matrix multiplication version.

a. Implement the TODO section in *T1.c*. This matrix multiplication version should be a cache-aware version (Hint: You can partition matrices into smaller submatrices to fit in the cache memory or you can try any other strategy that may help). *(2 Points)*

b. Explain your strategy to convert the given matrix multiplication to be cache-aware. *(2 Points)*

c. Plot the relation between the following matrix sizes N = {200, 400, 600, 800, 1000} and the execution time for the cache-aware version and the given version. *(2 Points)*

d. Use the command "perf stat -e cache-misses ./your_program" to report the total number of cache misses encountered by executing both versions for the previous matrix sizes. *(2 Points)*

2 Memory access patterns *(3 Points)*

There are 6 possible loop orderings to organize the three loops of the matrix multiplication in *mult_matr.c*. One of them we have already used: **IJK**.

a. Implement the TODO section in *T2.c* to provide another loop ordering that minimizes the execution time. *(1 Point)*

b. Using the provided code in *T2.c*, report the performance of both versions in (GFLOP/s) where the matrix size N = 2000. **Do not use any optimization flags.** *(1 Point)*

c. Explain why your proposed loop ordering is more efficient than the given one. *(1 Point)*

3 Data organization in the memory *(5 Points)*

In this task, you have to define matrices A, B, and C as 1D arrays of size $[N * N]$ instead of 2D arrays of size $[N][N]$.

a. Rewrite *T3_2D.c* as *T3_1D.c* such that A, B, and C in *T3_2D.c* are defined as 1D arrays instead of 2D arrays. **Make all the required changes in the code.** *(3 Points)*

b. Plot the relation between matrix sizes N = {200, 400, 600, 800, 1000} and the execution time for both versions. *(2 Points)*

4 Auto-vectorization *(4 Points)*

a. Use the following command to compile *T4.c* with the O3 optimization flag:
icc -O3 -qopt-report=1 -qopt-report-routine=multiply -qopt-report-file=report.optrpt to obtain the optimization report. *(2 Points)*

b. Modify the *multiply* function in such a way that the compiler can vectorize and **provide** the new optimization report. *(2 Points)*

5 Bonus task *(3 Points)*

a. Combine all optimization techniques in tasks 1, 2, 3, and 4 to provide the most efficient matrix multiplication version. *(1 Point)*

b. Plot the relation between the execution time and the matrix sizes N = {200, 400, 600, 800, 1000} of your optimized version comparing to the base version given in *T1.c*. *(2 Points)*

Please make sure that your optimizations do not affect the correctness of the results.
You should use the miniHPC cluster to obtain all the results.
The delivered solution should be in one tar file.