CSE 625 Parallel Programming Project 2

October 4, 2022 (100 points)

Due: October 20 (Thur) midnight (Submit your project report to the Blackboard.)

Readings: 04\_C++\_11\_MultiThreading\_1\_rev.pdf and 03\_AVX.pdf

CodeBlocks Project: Matrix\_2

Computer Used

Try to use the computers in DC119 to do project 2. DC119 computers have the following specs:

CPU Intel i7 8700K CPU @3.76GHz

AVX2 (256-bit MM registers)

6 cores / 12 threads

12 MB Intel Smart Cache (L3-cache)

RAM 32 GB RAM

GPU GTX 1080Ti (Pascal GPU)

3584 CUDA cores  
 2,816 kilobytes of L2-Cache  
 11 GB Memory

If you cannot use DC119 computers to do project 2, state the specs of the computer (like the specs given above) you use for project 2 in your project report. Make sure that the computer you use has sufficient CPU (and GPU for future projects) capability to do a reasonable study of pralllel computing applications.

Assignments

1 Study the Martix\_2 CodeBlocks project.

* 1. (5 points) Explain row-majoy memory layout of using one-dimentional array to represent two-dimentional matrix.

1.2 (20 points) Use the Matrix\_2 project to measure square matrix multiplication   
 runtimes of these three matrix multiplication functions (defined in matrix.cpp):  
  
 matrixMul\_RowMajor,

matrixMul\_tmm, and

matrixMul\_AVX\_tmm.

Measure the runtimes of matrix multiplication for matrix sizes of 200, 400, 800, 1600, 3200, 6400 and 12,800 put the timing results in the following table. The speed-up is measured relative to the matrixMul\_RowMajor (the baseline) runtimes.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 200 | 400 | 800 | 1,600 | 3,200 | 6,400 | 12,800 |
| matrixMul\_  RowMajor |  |  |  |  |  |  |  |
| matrixMul\_ tmm |  |  |  |  |  |  |  |
| Speed-up |  |  |  |  |  |  |  |
| matrixMul\_ AVX\_tmm |  |  |  |  |  |  |  |
| Speed-up |  |  |  |  |  |  |  |

In the project report, include:

[1] The timing table result

[2] Explaining how tmm and AVX\_tmm matrix multiplications speed up the baseline

multiplication

[3] A couple of screenshots of your timing tests

2 In this problem, you need to use Python to compute matrix multiplication. Given two   
 numpy 2D matrices as given below:   
  
 import numpy as np  
 s = 800  
 mat1 = np.random.random((s, s)).astype(np.float32)

mat2 = np.random.random((s, s)).astype(np.float32)

compute the matrix multiplication of mat1 and mat2 in Python and measure its  
 execution times using two computing methods.

2.1 (15 points) The first method is to implement the multiplication using Python for-  
 loops. Implement the matrix multiplication and measure its execution times for   
 various sizes of s. Put the timing results in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 200X200 | 400X400 | 800X800 |
| For-loop Timing |  |  |  |

2.2 (10 points) The second method is to compute matrix multiplication using numpy matrix multiplication function matmul (the @ operator) and measure its execution times for various sizes of s. Put the timing results in the following table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 200X200 | 400X400 | 800X800 | 1600X1600 | 3200X3200 | 6400X6400 | 12,800X12,800 |
| matmul Timing |  |  |  |  |  |  |  |

In the project report, include:

[1] Listing of Python for-loop matrix multiplication script,

[2] Listing of Numpy matrix multiplication script, and

[3] The timing table results

3(25 points) Write a C++ sequential dot product function and an AVX dot product   
 function of two float vectors. Test and compare the time performance and accuracy of   
 the results of these two functions using vectors of sizes 6,400,000 and 64,000,000,   
 respectively. In the project report, include:

[1] Listing of the sequential and AVX dot product functions

[2] Test results given in the following table

|  |  |  |
| --- | --- | --- |
|  | 6,400,000 | 64,000,000 |
| Sequential time |  |  |
| AVX time |  |  |
| Sequential result |  |  |
| AVX result |  |  |

[3] Explain why the AVX result is accurate for 64,000,000, but the sequential result is   
 not

4(25 points) Implement a C++ multithreading matrix multiplication function in C++   
 based on the matrixMul\_RowMajor function defined in matrix.cpp of the Matrix\_2   
 project. Use lambda expression in the implementation. Test the time performance on   
 3200x3200 and 6400x6400 float matircs, respectively.

In the project report, include:

[1] Listing of the multithreading matrix multiplication function

[2] Showing the timing results in the following table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 4 threads | 6 threads | 8 threads | 16 threads | 20 threads |
| 3200x3200 |  |  |  |  |  |
| 6400x6400 |  |  |  |  |  |

You can change the number of threads in the table as you consider necessary to perform the timing tests.

[3] Discuss what could be the number of threads that gives fastest result.