CS 231 Lab 3

Part 3

Report

Saksham Rathi
Second-Year Computer Science Undergraduate
22B1003

Contents

1 Approach from the start

In this problem, we were supposed to first take four scalars and then four matrices. Then we calculated two intermediate matrices as

$$B_1 = s_1 \times M_1 + s_2 \times M_2$$

$$B_2 = s_3 \times M_3 + s_4 \times M_4$$

Then we took the hadamard product of these two matrices:

$$F[i][j] = B_1[i][j] \times B_2[i][j]$$

Finally, we calculate the alternate sum:

$$\textbf{Final Answer} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (-1)^{i+j} F[i][j]$$

These tasks had to be performed using two approaches through nested for loops.

- **Row Major :** First iterate through the rows and then for each row, iterate through all the columns
- **Column Major :** First iterate through the columns and then for each column, iterate through all the rows

2 Optimizations

We were also supposed to perform optimizations on the codes. I did the following optimizations:

1. **Cache Locality :** Whenever an address is demanded by the Operating System, the DRAM stores a certain contiguous chunk after that address into the Cache for fast access by the OS. So, the number of cycles required to access a certain bytes after that becomes faster. I have made use of this fact extensively in my submissions. So, in the row major forms, the next address demanded is contiguous to the previous one with a difference of 8 bytes. So, the number of cycles needed to run the Row Major form are very less as compared to Column Major form for larger inputs.
2. **Unroll :** Due to two nested for loops, performance gets significantly reduced. To decrease the intrinsic calculations and also avoiding the change of PC through jmp statements, one can perform 4 or more computations in a single iteration of for loop. So, for the Row Major forms, I completely removed the inner for loop and just used a single for loop for iterating over the entire matrix. (Since, it is contiguous.) This has significantly improved the overall performance of the code.
3. **Avoiding stalls :** When a register is used for consecutive instructions, there is a chance of stall in pipelines. This causes a significant delay, thereby hindering performance. This has been taken care of wherever possible.
4. **Minimizing push and pop instructions :** Push and popping unused registers will also hinder the performance of the code. So, I tried to minimize them as much as possible. This required building the code on minimum number of registers possible.

3 Data Obtained

Firstly I obtained the TSC (time stamp counter) of Mac Docker using the code provided. So, the TSC was roughly **1005 MHz**.

Then, I ran the column major and row major codes on various values of N. I also created a code in Python (numpy) and measured it's time using time command to show the speed of x86 codes. x86 codes are being run for 10 times for each operation, but still they are far more fast than python. So here is the data:

RM = Row Major
CM = Column Major
Time = $\frac{\text{Cycles}}{\text{TSC}}$

Input Size (N)	**RM Cyles**	**RM Time (μs)**	**CM Cyles**	**CM Time (μs)**	**Python**
1	69912	69.5	82112	81.70	1.01s
2	67375	67.04	108495	107.96	1.15s
4	69945	69.597	66654	66.32	1.39s
8	72641	72.28	75608	75.23	0.52s
16	70433	70.08	71079	70.73	1.11s
32	82350	81.94	90241	89.79	0.99s
64	109229	108.68	119050	118.46	0.81s
128	219975	218.88	285995	284.57	1.43s
256	695741	692.28	1153062	1147.33	1.00s
512	2640725	2627.59	6081291	6051.04	1.16s
1024	10479008	10426.88	59176483	58882.07	2.01s
2048	41706325	41498.83	293295925	291836.74	7.77s

4 Plots

The above plot is for the time taken (in μs) by Row Major and Column Major forms for various matrix dimensions. The x-axis is on logarithmic scalar. The Row Major form dominates over the Column Major form for large inputs, thus depicting the concept of Cache Locality.

The second plot 4 shows the number of cycles taken by Row Major and Column Major forms vs the dimension of matrices on a logarithmic scale.

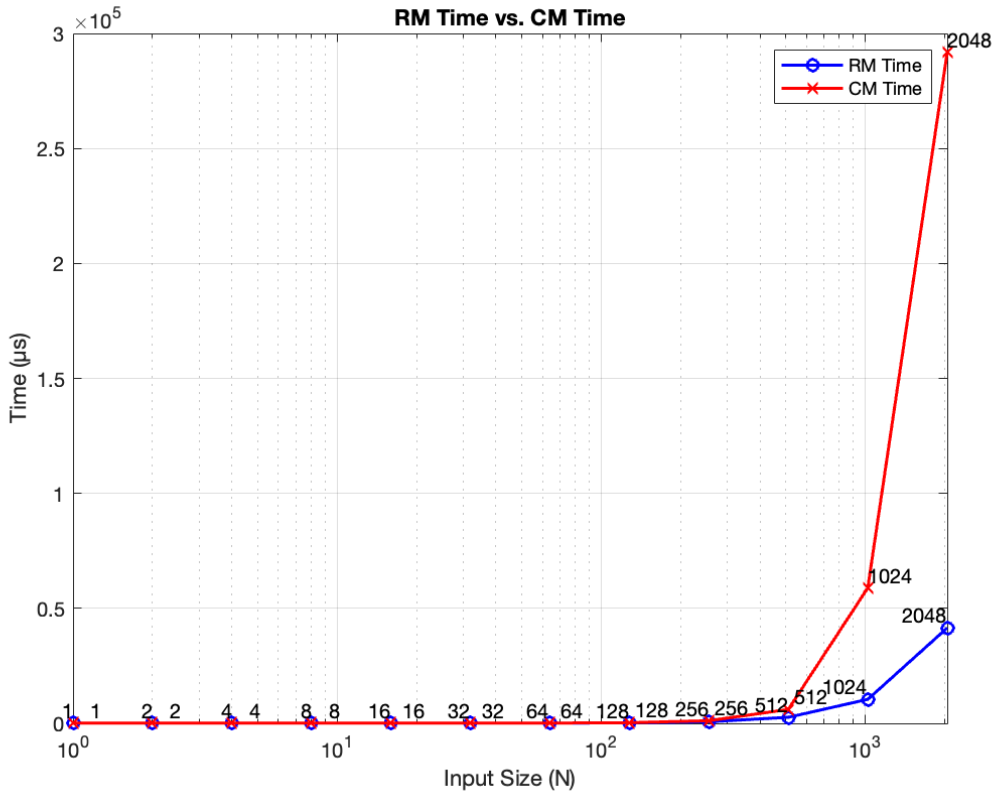

Figure 1: Plot of Time vs Matrix Dimension

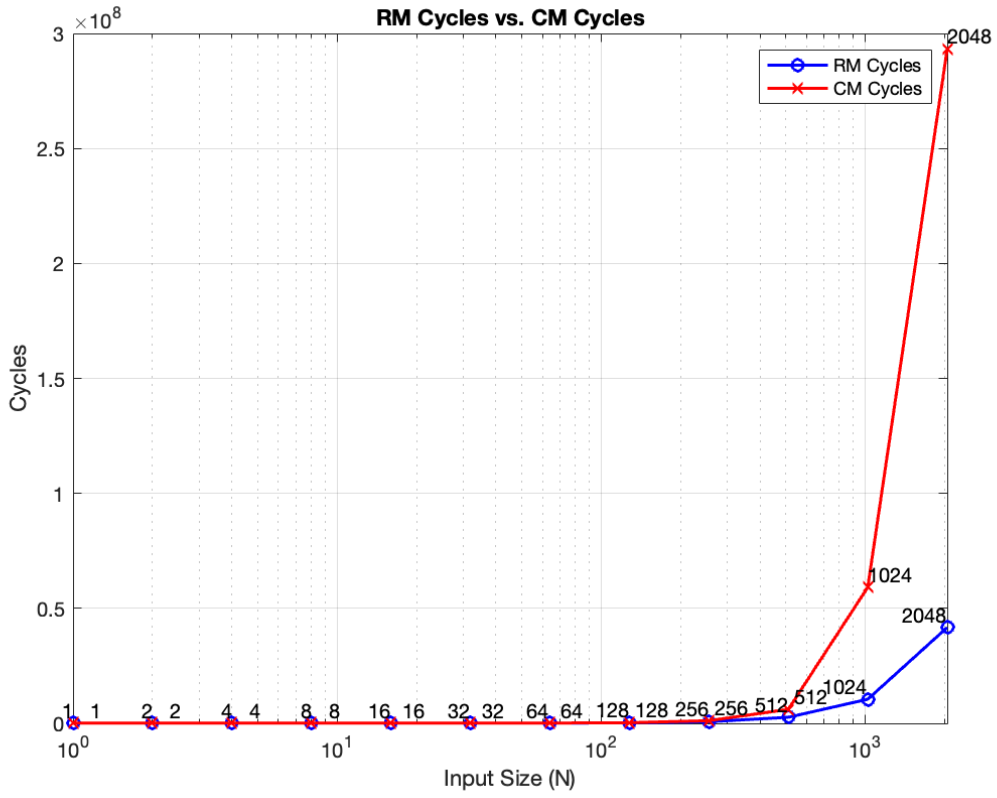

Figure 2: Plot of Number of Cycles vs Matrix Dimension