CS231 Lab 3 Part 3 Report

Arohan Hazarika

October 23, 2023

1 Approach to solve the problem

1.1 my_LINCOMB_COL.asm

I used r11 for the column index (i) and r13 for the row index (j) in this assembly code. The outer loop, controlled by r11, iterated over columns, and the inner loop, controlled by r13, cycled through rows. For each element, I calculated memory offsets for mat1 and mat2, loaded values into r14 and r15, performed scalar multiplications, stored the result back in memory, and adjusted memory offsets and counters

1.2 my_SUM_COL.asm

I used r11 for the column index (i) and r13 for the row index (j). The outer loop iterates over columns, and the inner loop iterates over rows. For each element, I load the value of mat1[j][i] into r14, determine if the sum of i and j is even or odd, and apply a multiplier of 1 or -1 accordingly(which is done by doing **and** operation and then comparing the last bit) . The result is added to rax to update the sum. This code computes the sum based on the parity of the row and column indices and stores the result in rax.

1.3 my_PRODUCT_COL.asm

In this assembly code, r11 represents the column index (i), and r13 is used for the row index (j). The code efficiently performs matrix multiplication. The outer loop iterates over columns (i), and the inner loop iterates over rows (j). For each element, it calculates memory offsets, loads values from mat1 and mat2 into r14 and r15, respectively. Then, it multiplies r14 and r15 to compute the product, which is stored in r14.

1.4 matrix-testbench.asm

Here I have used mmap for memory allocation using syscall.

1.5 Note

Similar to columns, my_LINCOMB_ROW.asm, my_SUM_ROW.asm and my_PRODUCT_ROW.asm also work where the operations are done by taking summations along rows.

2 Observations

Dimension	Cycles (row)	Cycles (col)	TSC	Time (row)	Time (col)
128	689334	933302	2112.006 MHz	$3.26\text{x}10^{-4}$s	$4.42\text{x}10^{-4}$s
256	1598583	3130307	2112.006 MHz	$7.57\text{x}10^{-4}$s	$1.48\text{x}10^{-3}$s
512	5779194	35582071	2112.006 MHz	$2.74\text{x}10^{-3}$s	$1.68\text{x}10^{-2}$s
1024	25845810	271023334	2112.006 MHz	$1.22\text{x}10^{-2}$s	0.1283 s
2048	124105883	1332558834	2112.006 MHz	0.058 s	0.631 s

Table 1: Observed data

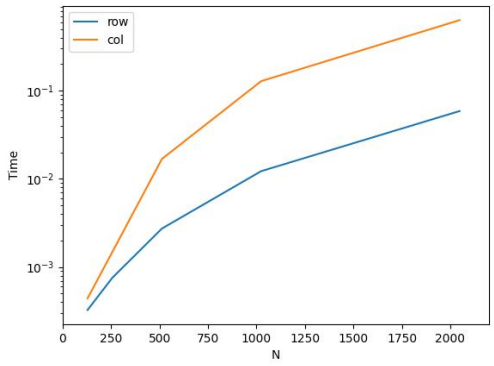

Figure 1: Graph (X axis = size, Y axis = time (in seconds))

Explanation

The time T in seconds is given by the formula: $\text{T} = \frac{\textit{Number of clock cycles}}{\textit{TSC (in Hz)}}$

From the graph we can see row operations are taking less time compared to column operations, the main reason for this is contiguous access of row elements in memory whereas column accesses take more time as the elements in the same column are stored in places scattered in the memory. Moreover cache is one more reason, as we know when we access an element from memory, the neighboring elements are also brought along with it and stored in cache and there the row operations have an advantage over column operations.