Matrix Analysis Runtime Experiment Project 4 COP 4634 Systems & Networks 1 Fall 2016

Ashley Miller/Torrey Bettis
3 November 2016

Memory management is one of the most important and also most overlooked aspects of programming. Improper handling of memory can lead to leaks which can cause systems to run inefficiently, or to crash all together. A two dimensional matrix will be utilized in this experiment. It will consist of 20480 rows and 4096 columns, with a page size of 4096 bytes. Due to the massive size of this matrix, it will be declared globally. Operations will be performed on this matrix, and the data collected to theorize about speed of operation, and memory management techniques.

The read and write functions were iterated through 10 times using remote access to the SSH server, with optimization turned off. This data was written directly to a .CSV file for statistical processing. The outcomes from this trial ran on the SSH server can be seen in figure 1 and table 1. As can be seen the difference between row major operations and column major operations is quite large when ran on the SSH server.

Table 1 Runtime using SSH Server

Time(s)				
Trial	Row Write	Column Write	Row Read	Column Read
1	0.376708	4.818609	0.295873	3.39107
2	0.27665	4.820149	0.295877	3.378282
3	0.287748	4.820703	0.295674	3.369895
4	0.276748	4.84395	0.295873	3.354449
5	0.276706	4.847084	0.295612	3.361505
6	0.276609	4.84626	0.295624	3.354319
7	0.276647	4.817222	0.295775	3.353346
8	0.276802	4.817072	0.295952	3.36417
9	0.276755	4.812495	0.300857	3.355989
10	0.276681	4.825629	0.295584	3.361393
Ave	0.2878054	4.8269173	0.2962701	3.3644418
Median	0.276748	4.820703	0.295873	3.361505
Standard Deviation	0.03142962	0.01274753	0.00153403	0.011598436

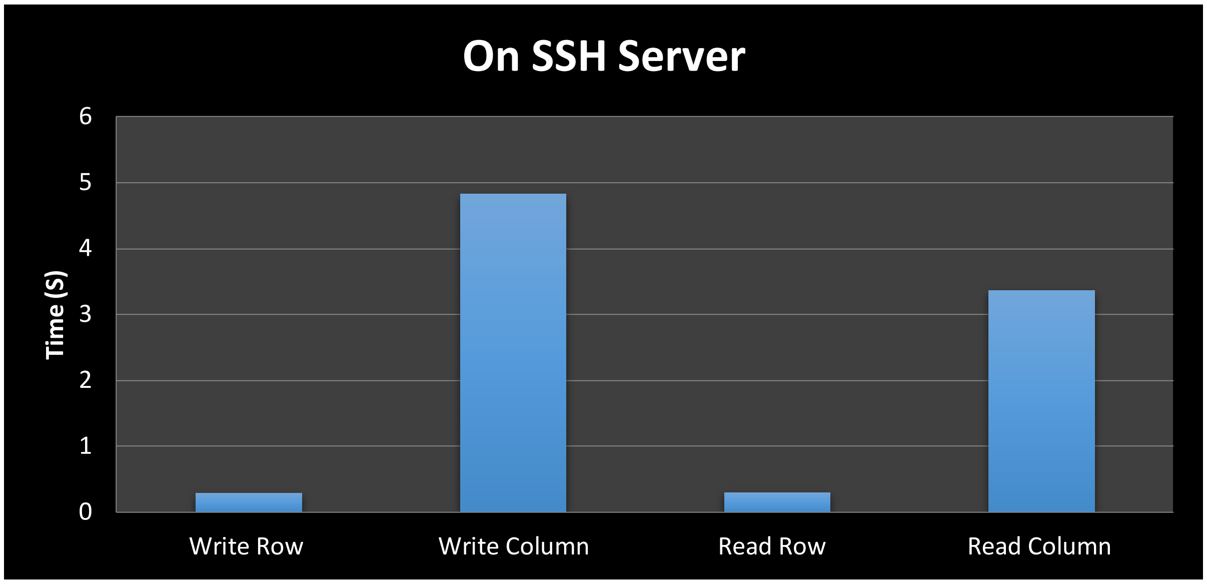

Figure 1 Graph of operation times when run on SSH server

The local machine used was a 3.2 GHz quad core i7 with 14 GB of RAM, loaded with Windows 7. Microsoft Excel and NetBeans were the only programs open during the experiment. The matrix row operations on the local machine were very similar to those ran on the SSH server, although faster for column operations. In both cases the row operation outperformed the column operations as was to be expected. We surmise this was due to page size, and the page faults that occurred during the operations. Since the page size was equal to row size, we expected one page fault per row operations. Column size was five times the page size so we expected many more page faults, and this can be seen in the time difference between the two operations. As also can be seen, the difference compiler optimization has on run times.

Table 2 Runtime using SSH Local Machine

Times(ms)				
Pass	**Row Write**	**Column Write**	**Row Read**	**Column Read**
1	0.212161	2.017093	0.224641	1.073287
2	0.210601	2.017093	0.229321	1.101367
3	0.215281	2.045173	0.232441	1.082647
4	0.213721	2.035813	0.230881	1.074847
5	0.218401	2.042053	0.232441	1.084207
6	0.212161	2.009293	0.226201	1.088887
7	0.213721	2.038933	0.229321	1.079527
8	0.209041	2.038933	0.232441	1.082647
9	0.210601	2.049853	0.229321	1.079527
10	0.215281	2.048293	0.230881	1.044967
Mean	0.213097	2.034253	0.229789	1.079191
Median	0.213097	2.038933	0.229789	1.079527
Standard Deviation	0.002628959	0.013706711	0.002520257	0.013638442

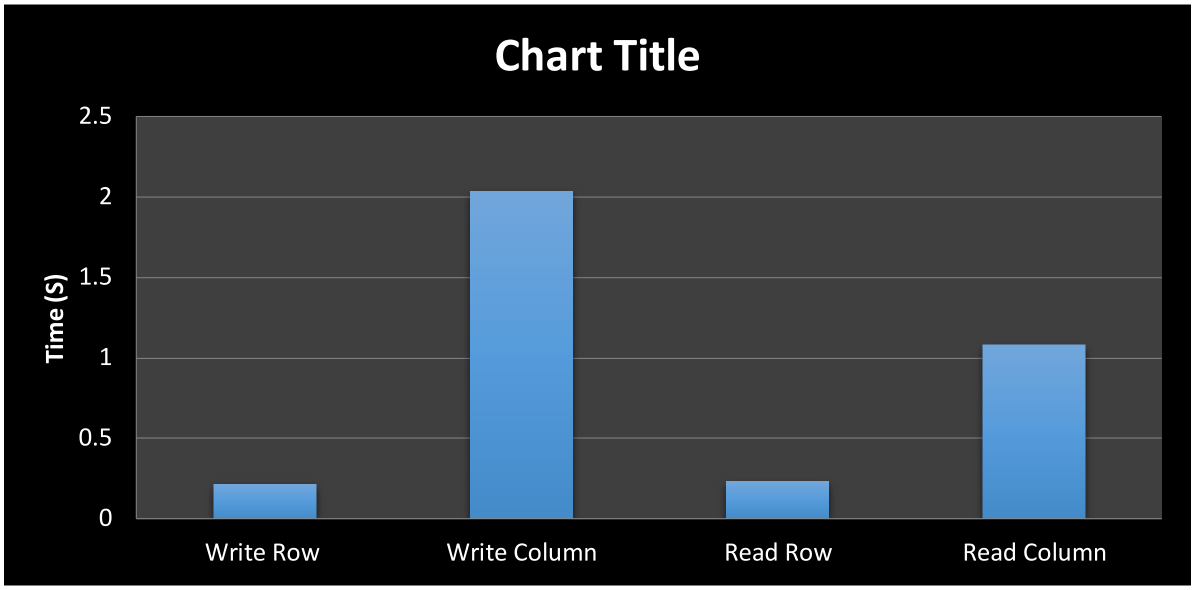

Figure 2 Graph of operation times when run on Local Machine

This experiment brought to focus how important a part memory access plays in process efficiency. It is easily observed that row operation outperforms column operations. Page size, and the faults associated with them are a major factor in processing time. The columns hold five page lengths, so it has page faults before it increments to the next column, which in itself creates a page fault. This accounts for the greater run time for column based operations. This experiment has shown that memory management plays an important role in programming, and how it is used must be taken into consideration when code is created.