CMDA 3634
Lab 05 Report

Russell J. Hewett

1. Answer the following questions about your laptop computer. You will need to refer to Internet resources for this information. You must cite your sources. Note, the mechanisms to find the answers to these questions are different for Windows vs macOS. Find this information for your laptop, not for the VM.

   - What brand, model, and version is your processor?
   - What size are the L1, L2, L3 (if it exists) caches?
   - How many bytes per cache line?
   - How big is your main memory?

   ANSWER: Some data from `https://en.wikichip.org/wiki/intel/core_i7/i7-7700hq` and `http://www.cpu-world.com/CPUs/Core_i7/Intel-Core%20i7%20i7-7700HQ.html`.

	Host OS	Guest OS
Brand	Intel	Intel
Model	Core i7	Core i7
Version	7700HQ	7700HQ
L1	32KB (data)	32KB (data)
L2	256KB/core	256KB
L3	6MB	611KB
Cache-line	64B	N/A
Main Memory	16GB	4GB

2. In the command line on your virtual machine, you can check information about the processor by checking the contents of the /proc/cpuinfo file:

   ```
   > cat /proc/cpuinfo
   ```

   Copy this information into a table. Does this match your laptop's processor?

   ANSWER:

   See above. Everything matches except for the main memory, which we configured to be smaller in the VM.

3. In your own words, describe the difference between the two calculations being timed in `matvecColOrient.c` and `matvecRowOrient.c`. Before running any code, can you make any predictions about their relative performance for different sized matrices?

   ANSWER:

   The row-oriented calculation loops over rows first, then columns and the column-oriented calculation loops over columns first, then the rows. The row-oriented version should be faster because the matrix array is indexed in row-major order.

4. We will analyze square matrices for this exercise, although the code allows for rectangular matrix studies. Say that you were to perform a matrix-vector multiplication $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, and $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^N$.

(a) Write a general formula for how many bytes of data you expect this to use (similar to the $16N$ expression used in the random-order axpy example in class).

(b) For your system, at what value for N do you expect to exceed L1 cache?

(c) For your system, at what value for N do you expect to move out of lower level (L2 or L2+L3) cache?

ANSWER:

(a) The $\boldsymbol{A}$ has N^2 4-byte floats and $\boldsymbol{x}$ ans $\boldsymbol{b}$ have N 4-byte floats. Thus, the total data used would be $4(2N + N^2)$ Bytes, ignoring loop indicies.

(b) $4(2N + N^2)B = 32 * 1024B \implies N \approx 89$.

(c) $4(2N + N^2)B = (256 + 32) * 1024B + 6144 * 1024B \implies N \approx 1279$.

5. Using the experiment script `cache_row_orient.sh`, record the results for `matvecRowOrient.c` in the table below, to analyze the cache performance for this data access pattern.

ANSWER:

nRows	**nCols**	**time (s)**	L1 read miss %	LL read miss %	L1 write miss %	LL write miss %
10	10	0.0000003800	1.6%	1.3%	2.9%	2.6%
20	20	0.0000013000	0.5%	0.4%	1.7%	1.4%
40	40	0.0000052200	0.2%	0.1%	0.6%	0.6%
80	80	0.0000255400	0.0%	0.0%	0.3%	0.2%
160	160	0.0000896800	0.3%	0.0%	0.3%	0.1%
320	320	0.0003911600	0.3%	0.0%	0.2%	0.1%
640	640	0.0015848800	0.3%	0.0%	0.2%	0.1%
1280	1280	0.0066128600	0.3%	0.2%	0.2%	0.1%
2560	2560	0.0249078800	0.3%	0.3%	0.2%	0.2%
5120	5120	0.1010645400	0.6%	0.3%	0.2%	0.2%

6. Using the experiment scripts `cache_col_orient.sh`, record the results for `matvecColOrient.c` in the table below, to analyze the cache performance for this data access pattern.

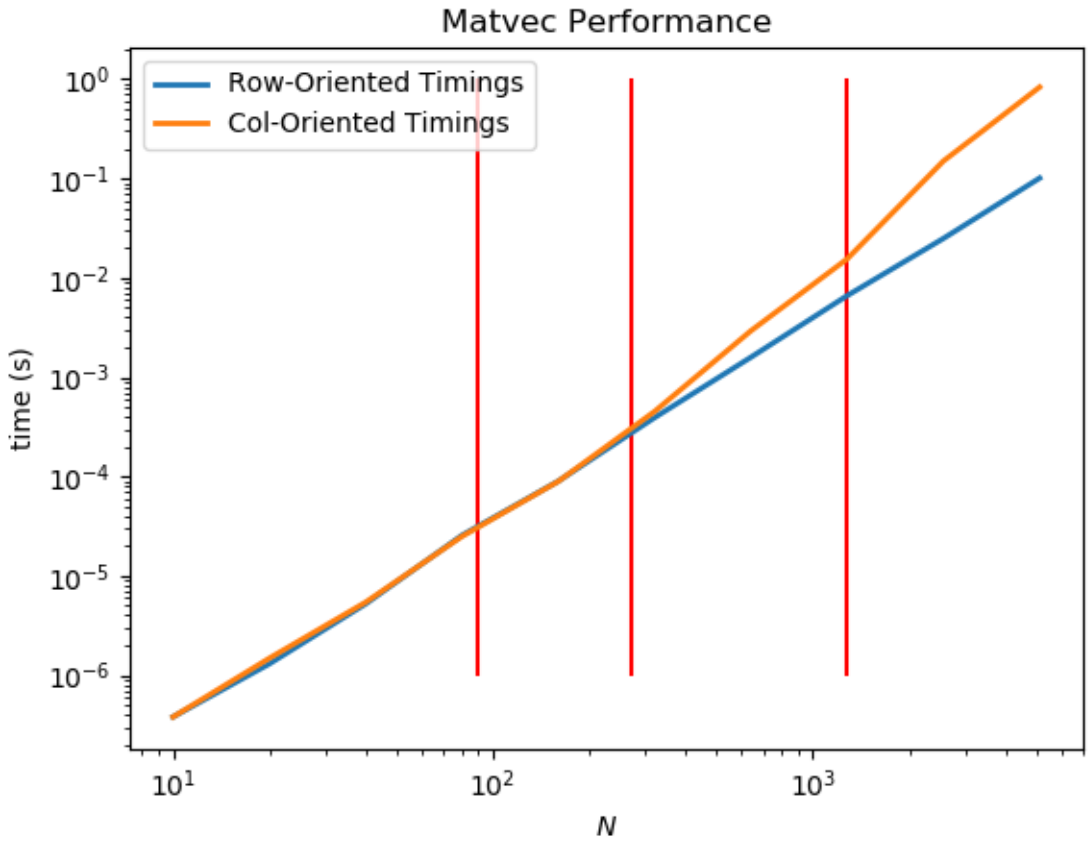
ANSWER:

nRows	**nCols**	**time (s)**	L1 read miss %	LL read miss %	L1 write miss %	LL write miss %
10	10	0.0000003800	1.6%	1.3%	2.9%	2.6%
20	20	0.0000014800	0.5%	0.4%	1.7%	1.4%
40	40	0.0000054200	0.2%	0.1%	0.6%	0.6%
80	80	0.0000249200	0.0%	0.0%	0.3%	0.2%
160	160	0.0000892000	0.3%	0.0%	0.2%	0.1%
320	320	0.0004509200	4.9%	0.0%	0.2%	0.1%
640	640	0.0029247200	4.9%	0.0%	0.2%	0.1%
1280	1280	0.0154923400	4.9%	0.2%	0.2%	0.1%
2560	2560	0.1498281800	4.9%	0.3%	0.2%	0.2%
5120	5120	0.8327762400	4.9%	4.9%	0.2%	0.2%

7. Using your favorite plotting software (Excel, Python Matplotlib, Plot.ly, Matlab, etc.) plot the timing results for each of the above tables. You should plot N on the x-axis and the run-time on the y-axis. Both axes should use a logarithmic scale.

   Be sure to add any files/code used to produce the plot in your `lab05/report` folder. Your plot must have a legend indicating which series corresponds to which experiment.

ANSWER:



Code is in `plot_data.py`

8. In the previous plot, where are the noticeable changes in the timing trends? At the point(s) where the timing trends change, what is causing this change? Support your argument with your memory size predictions and your cachegrind data. You may need to take more samples. Be sure to include any additional code/files used to produce those figures in your `lab05/report` folder.

   ANSWER:

   The red lines in the plot indicate the values of the cache boundaries (L1, L2, L3) from problem 4. In these areas we notice a change in the timing trends for the column-oriented algorithm. For the row-oriented algorithm we do not see these trends as the access patterns map well to the cache and prefetching behavior.

9. How many processors does your virtual machine have? How do you know? Include a screenshot if it's useful.

   ANSWER:

   The VM is configured to 1 processor. This info came from `lscpu`.

10. In the table below, record your timing results for the serial matrix-vector multiply (with row-major access) and for the openmp matrix-vector multiply (with row-major access). Also, record the ratio of the times.

    Did you get as much speedup as you expected based on the number of processors? Why do you think this happened?

    ANSWER:

nRows	nCols	time (s) without omp	time (s) with omp	serial time / omp time
10	10	0.0000003773	0.0000009947	0.38
20	20	0.0000020441	0.0000046239	0.44
40	40	0.0000057851	0.0000060627	0.95
80	80	0.0000363203	0.0000222913	1.63
160	160	0.0002472032	0.0001031512	2.40
320	320	0.0008854729	0.0004679907	1.89
640	640	0.0035193957	0.0016044020	2.19
1280	1280	0.0097061886	0.0065218148	1.49
2560	2560	0.0276844027	0.0270949682	1.02
5120	5120	0.1148509079	0.1049633594	1.09

Initially we did not achieve a speedup of 2, as expected, because the startup cost of the second OpenMP thread were higher than the cost of the single thread doing work (and the problem fit into cache). Until we exceeded the rough boundary of the cache, we obtained nearly the 2x speedup that was anticipated. After the problem went to main memory, the performance reduced again, beacuse we crossed into the main memory boundary.

11. Other than the instructor or TAs, who did you receive assistance from on this assignment?

    ANSWER:

    No one.