High Performance Computing

Homework #4

Due: Tuesday March 15 2016 by 11:59 PM

Email-based help Cutoff: 5:00 PM on Mon, March 15 2016

Name: Henry Ni

Experimental Platform

The experiments documented in this report were conducted on the following platform:

Component	*Details*
CPU Model	Intel Xeon X5550 (model 26)
CPU/Core Speed	2.67GHz
Main Memory (RAM) size	24725392 kB (~ 24GB)
Operating system used	CentOS 6.3
Interconnect type & speed (if applicable)	Not applicable
Was machine dedicated to task (`yes`/`no`)	Yes (via a `qsub` job)
Name and version of C++ compiler (if used)	g++ (GCC) 4.9.2
Name and version of Java compiler (if used)	None
Name and version of other non-standard software tools & components (if used)	None

Regular C++ Matrix Multiplication Runtime Observations

Document the statistics collated from your experiments conducted using the supplied C++ version of matrix multiplication in the table below. In Table 2 report just mean and 95% CI values (not the five raw timing values) for the standard C++ version of matrix multiplication.

MATRIX_SIZE	Samples	C++ Execution Time (*Avg±CI* sec)	Peak Memory (KB)
500	5	.47 ± .1245	28768
1000	5	10.212 ± .4256	99344
2000	5	78.282 ± 17.4588	381104
2500	5	133.256 ± 34.0206	592272
3000	5	354.16 ± 3.3193	1155280

Table 1: Execution timings for simple matrix multiplication implemented in C++. The average execution times and 95% confidence intervals (CI) have been computed from five independent runs of the test program `MatMul`.

Block Matrix Multiplication Runtime Observations

Document the statistics collated from your experiments conducted using the block matrix multiplication version in the table below. In Table 3 report just mean and 95% CI values (not the five raw timing values) for the C++ block matrix multiplication version.

MATRIX_SIZE	Samples	C++ Execution Time (*Avg±CI* sec)	Peak Memory (KB)
500	5	.196 ± .0581	28768
1000	5	1.48 ± .11	99344
2000	5	11.816 ± .4462	381104
2500	5	22.906 ± .9766	592272
3000	5	36.208 ± 1.2834	1155280

Table 2: Execution timings for block matrix multiplication implemented in C++. The average execution times and 95% confidence intervals (CI) have been computed from five independent runs of the test program `MatMul`.

Using the data in the column titled `Execution Time` plot the graph comparing regular and Block Matrix multiplication (using Excel/LibreOffice) and copy-paste the chart into this report replacing the chart already shown. You may use the chart below as a graphical template for developing your chart.

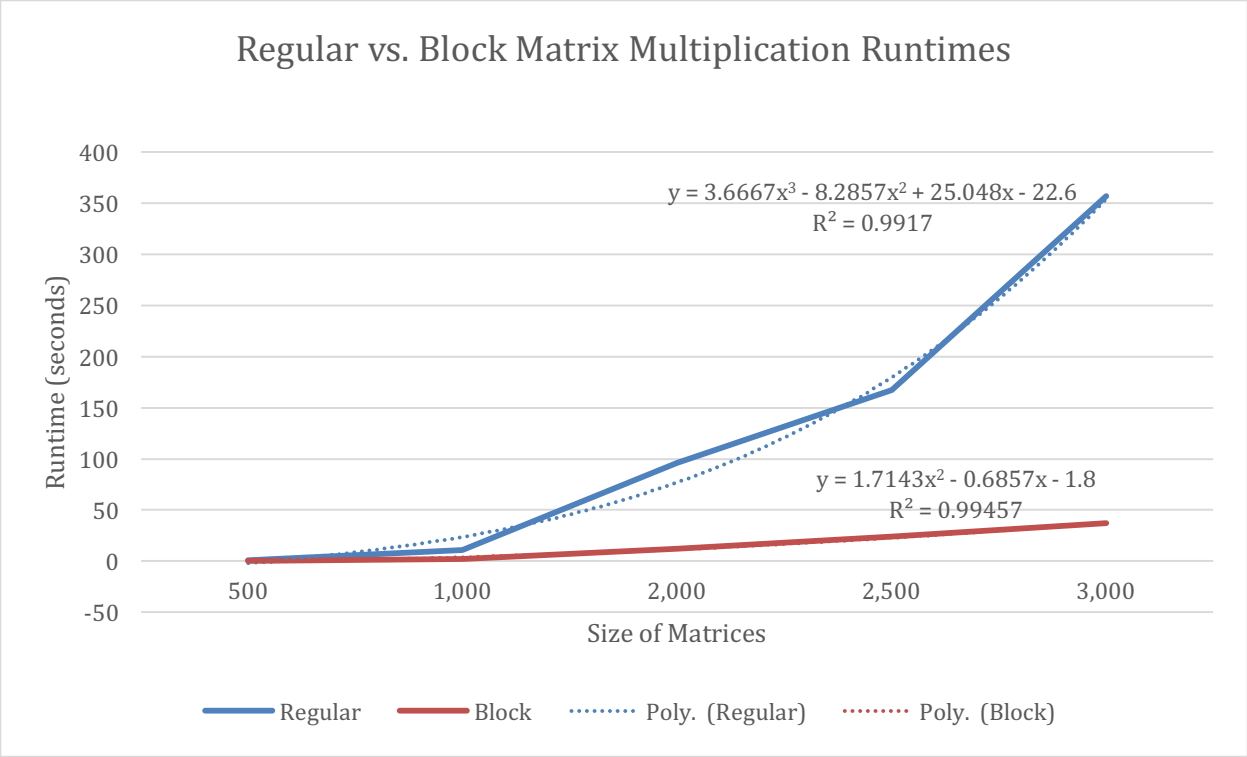

Figure 1: Plot comparing execution timings of matrix multiplication and Block Matrix Multiplication

Using the data from the above chart indicate the following information for the regular and block matrix multiplication runtimes:

Time complexity for C++ version: (This is the equation for the trend line)	Cubic ($y = 3.6667x^3 - 8.2857x^2 + 25.048x - 22.6$)
Time complexity for Block Matrix version: (This is the equation for the trend line)	Quadratic ($y = 1.7143x^2 - 0.6857x - 1.8$)

Inferences & Discussions

Now, using the data from tables along with the graphs document your inferences contrasting matrix multiplication and block matrix multiplication using the evidence gathered through the experiments (use as much space as you need). Discuss the change in time complexity between the two implementations using the trend line equations. Compare and contrast on the performance trends and memory usage of the two implementations. Include inference as to why one version performs better than the other.

Between the two implementations, block matrix multiplication far outperformed regular matrix multiplication in runtime speed, mostly due to the cache-friendly nature of block multiplication. By performing operations only in localized sub-matrices, we were able to exploit caching behavior and optimize for reference locality, reducing cache misses and keeping costly fetches to a minimum. In fact, this is so effective our block implementation has quadratic complexity while the regular implementation has cubic runtime complexity. We noticed the same behavior in the lab exercise in which we compared row major vs column major matrix operations and this comparison focuses on similar concepts.

However, as far as memory usage is concerned both versions load the same matrix into memory so they will use roughly the same amount of memory, as seen in the data above. Not much can be done except to make sure to pass matrices in by reference instead of value during each function call.