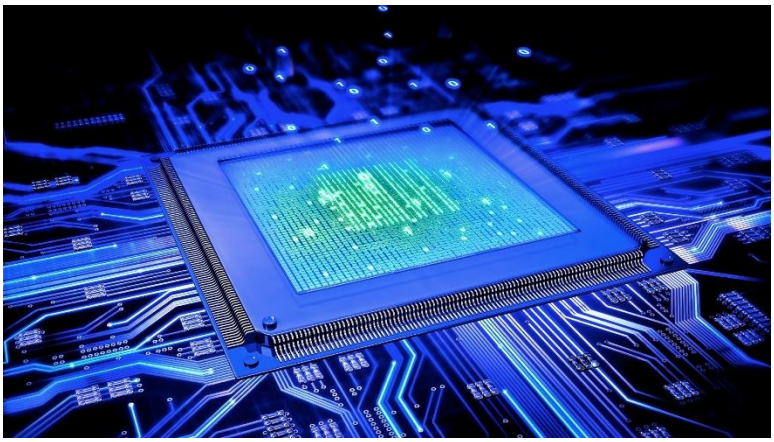

Homework #2

CPE 631

KYLE RAY

February 5, 2018

Contents

Purpose

To demonstrate the differences in performance between gcc or g++ and the intel provided compiler icc or icpc. Also, to become familiar with timing analysis and other tools such as the Performance Application Programming Interface (PAPI).

Setup

The programs for this homework assignment were created and executed on the Blackhawk machine located at the University of Alabama in Huntsville. The programs contain C++ code and were compiled with g++ and icpc.

Problem 1

PROGRAM DESCRIPTION

C++ application that multiplies two squared matrices A and B of size N (NxN) doubles. The program accepts the size of the matrices via a command line argument, then fills the matrices with random values and proceeds to calculate the resulting matrix via matrix multiplication. The NxN resulting matrix is then written to a binary file.

PART A RESULTS

The tables below contain the overall execution time results, which were gathered using the built-in time script in Linux, for running the matrix multiplication program with different sizes of N consisting of 512, 1024, 1536, and 2048. Table 1 houses the results for program execution using the g++ compiler with the -O3 optimization and contains the results for using the icpc compiler with the -fast optimization.

Table 1. Execution Times for g++ and icpc Matrix Multiplication

Matrix Size	512	1024	1536	2048
Execution Time (g++) in seconds	0.251	2.343	19.37	143.494
Execution Time (icpc) in seconds	0.216	0.779	2.602	6.691

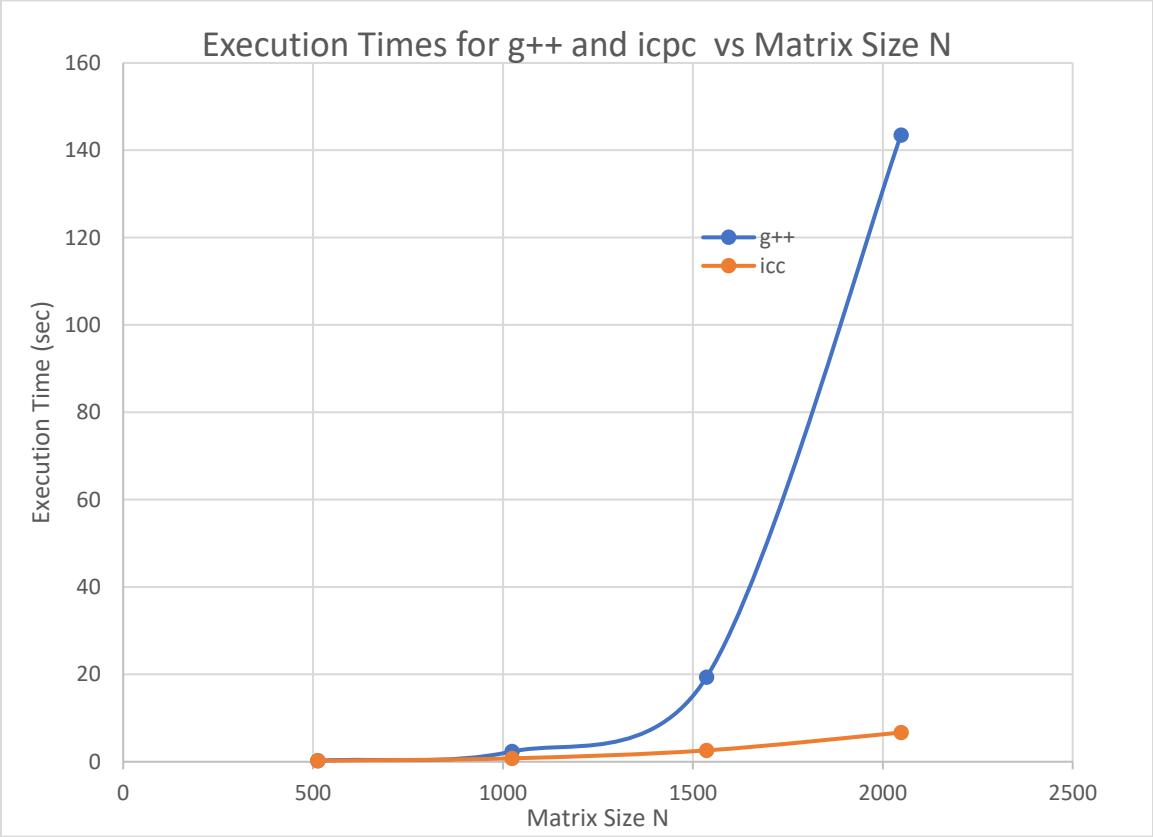

Figure 1. Execution Times for Each Compiler

Comparing the two tables it is easy to see that with a square matrix of 512 x 512 both compilers have roughly the same execution time; however, the intel compiler completely out performs the g++ version for the remainder of the test scenarios. The results do make sense given that the Blackhawk machine is using an Intel(R) Xeon(R) CPU E5-2643 0 @ 3.30GHz processor and the compiler would be more optimized towards intel processors than would g++.

PART B RESULTS

The tables below contain the data collected when measuring the execution time of the critical loop, the part of the code that is performing the matrix multiplication. The execution time of the critical loop was recorded using the clock methods provided in time.h for the serial versions.

Table 2. Critical Loop Execution Times for g++ and icpc Matrix Multiplication

Matrix Size	512	1024	1536	2048
Execution Time (g++) in seconds	0.23	2.28	18.77	142.87
Execution Time (icpc) in seconds	0.09	0.72	2.49	5.87

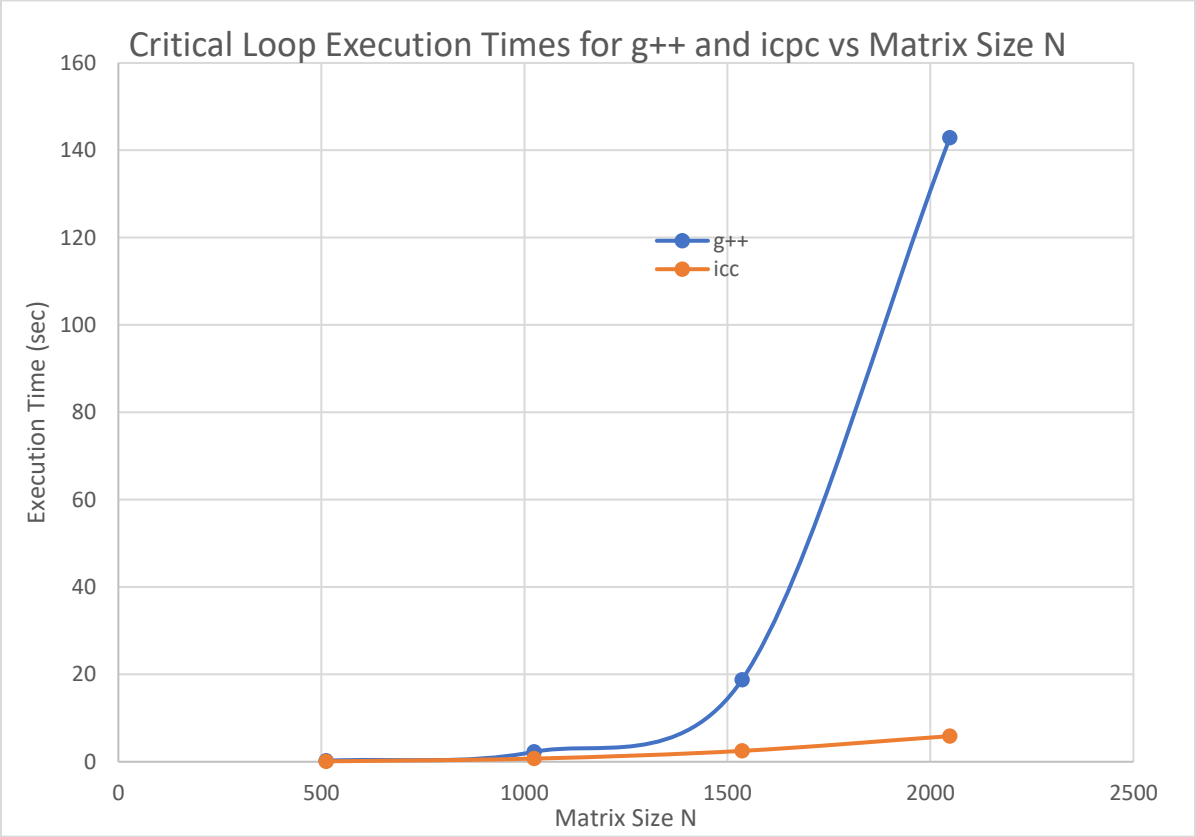

Figure 2. Critical Loop Execution Times for Each Compiler

Again, the intel compiler has better performance over the g++ compiler. It is interesting that regarding the g++ compiler the overall execution time is not that much larger than the critical loop execution time, in the case of matrix size 512 0.251 vs. 0.23; however, the icpc compiled version overall execution times hold a greater difference from the critical loop measurements, in the case of matrix size 512 0.216 vs. 0.09.

Problem 2

PROGRAM DESCRIPTION

Modified the existing matrix multiplication program from problem 1 to utilize OpenMP to parallelize the critical loop to increase overall performance.

RESULTS

The table below shows the results for critical loop execution time for both the g++ and icpc compilers using OpenMP. When measuring the execution time of the critical loop in the OpenMP programs the internal omp_get_wtime method from the OpenMP library was used.

Table 3. OpenMP Critical Loop Execution Times for g++ and icc Matrix Multiplication

Matrix Size	512	1024	1536	2048
Execution Time (g++) in seconds	0.144	0.473	1.405	35.625
Execution Time (icpc) in seconds	0.163	0.346	1.055	35.333

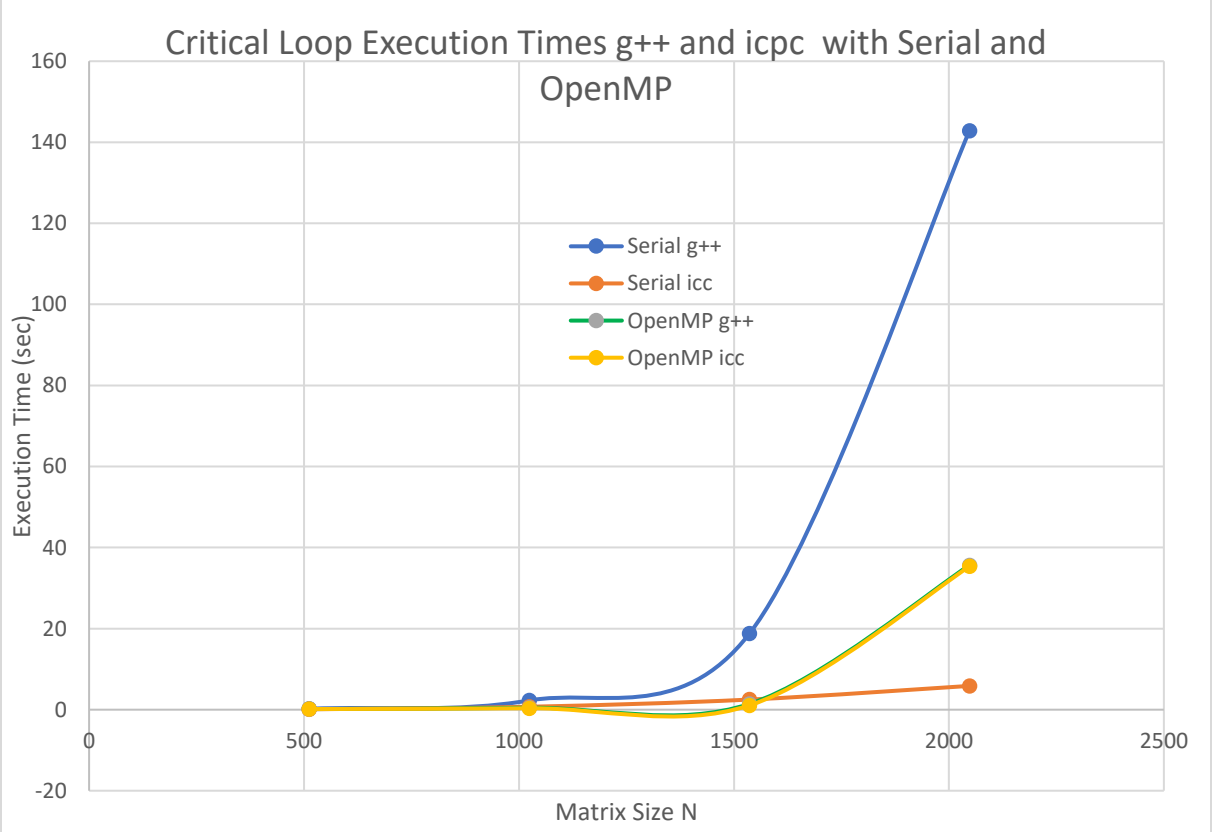

Figure 3. Critical Loop Execution Times for Each Compiler with OpenMP

It is interesting that utilizing OpenMP to parallelize the critical loop greatly benefitted the performance of the g++ version of the program, which was expected, yet seemed to hinder the performance of the icpc version. It seems that the overhead associated with introducing OpenMP to the icpc compiled version decreased the performance significantly. In the end, the OpenMP implementation results in nearly identical execution times for both compilers. In regards execution time as the performance metric the serial icpc version is still in the lead.

Problem 3

PROGRAM DESCRIPTION

Implement PAPI interface calls to measure number of instructions executed, total number of clock cycles, level 1 data cache misses, and the total number of L2 misses in the critical loop.

RESULTS

Refer to the figures below for the results of the experiment. Tabulated data of the results can be found in the Appendix A of this report.

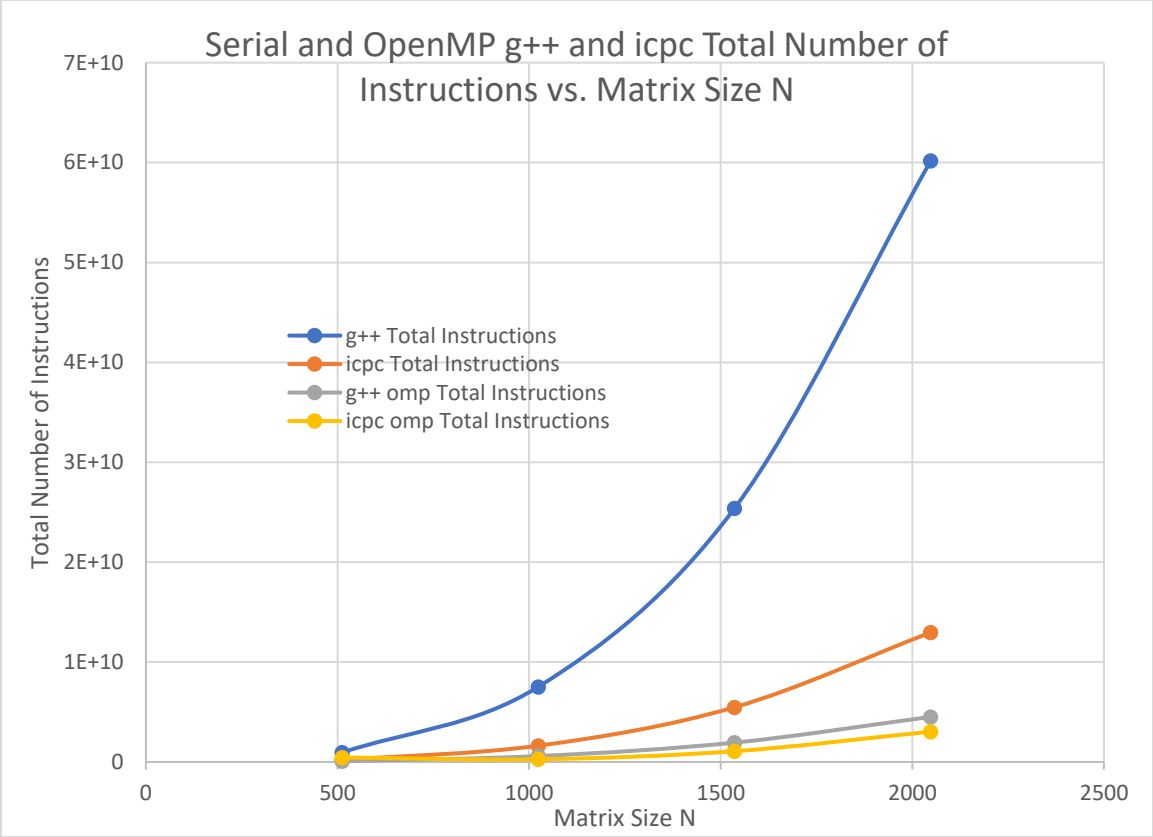

Figure 4. Total Number of Instructions for Each Matrix Size N

Adding the OpenMP macros to the g++ version of the code seemed to decrease the total number of instructions by a significant amount from the serial version. The icpc compiled version of the program observed a decent decrease in total number of instructions as well.

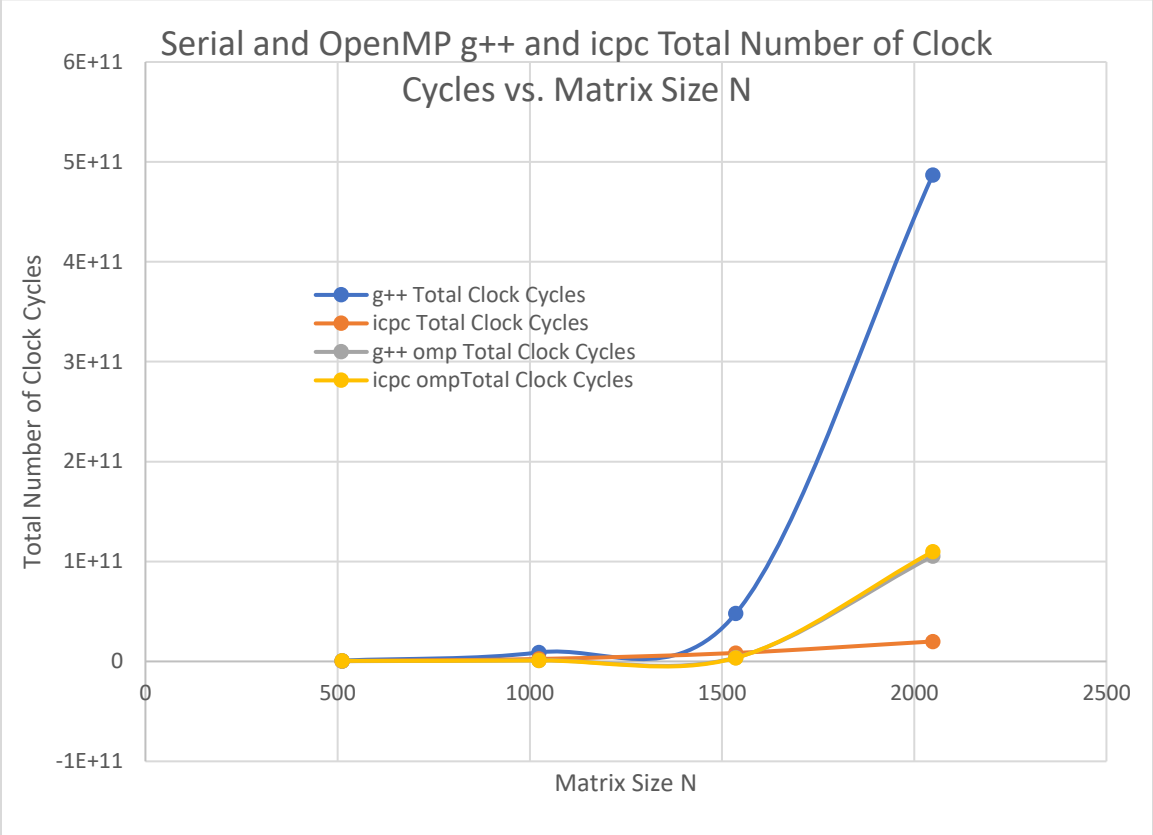

Figure 5. Total Number of Clock Cycles for Each Matrix Size N

Overall for the g++ compiled program the total number of clock cycles decreased when using OpenMP to parallelize the matrix multiplication, drastically for larger matrices. It seems that the overhead that is brought on by OpenMP increased the overall clock cycles for the icpc compiled program. As expected, this plot is very similar to the execution time of the critical loop shown in Figure 3 above.

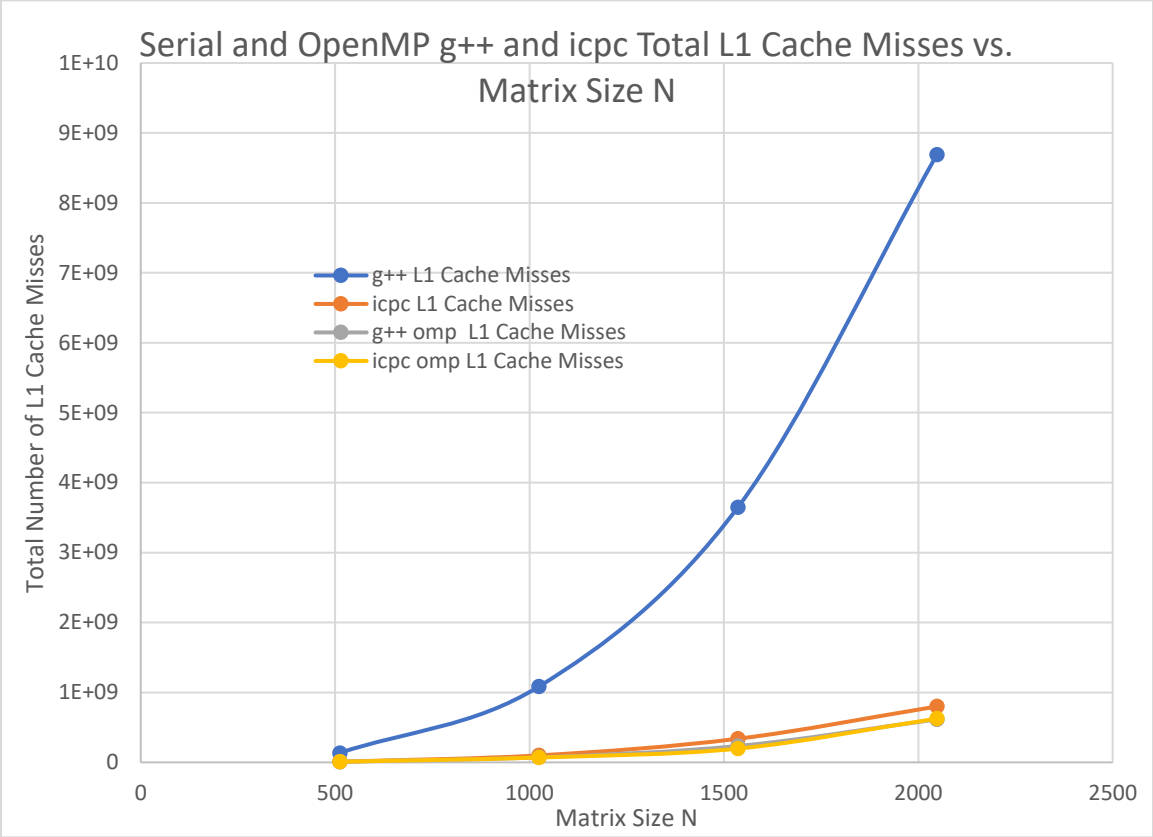

Figure 6. Total Number of L1 Cache Misses for Each Matrix Size N

According to Figure 6. g++ observes a significant amount more of L1 Caches misses than does the intel version. It is very interesting how much of an impact using OpenMP with g++ decreased the overall cache misses to be nearly on par with that of the intel compiled programs.

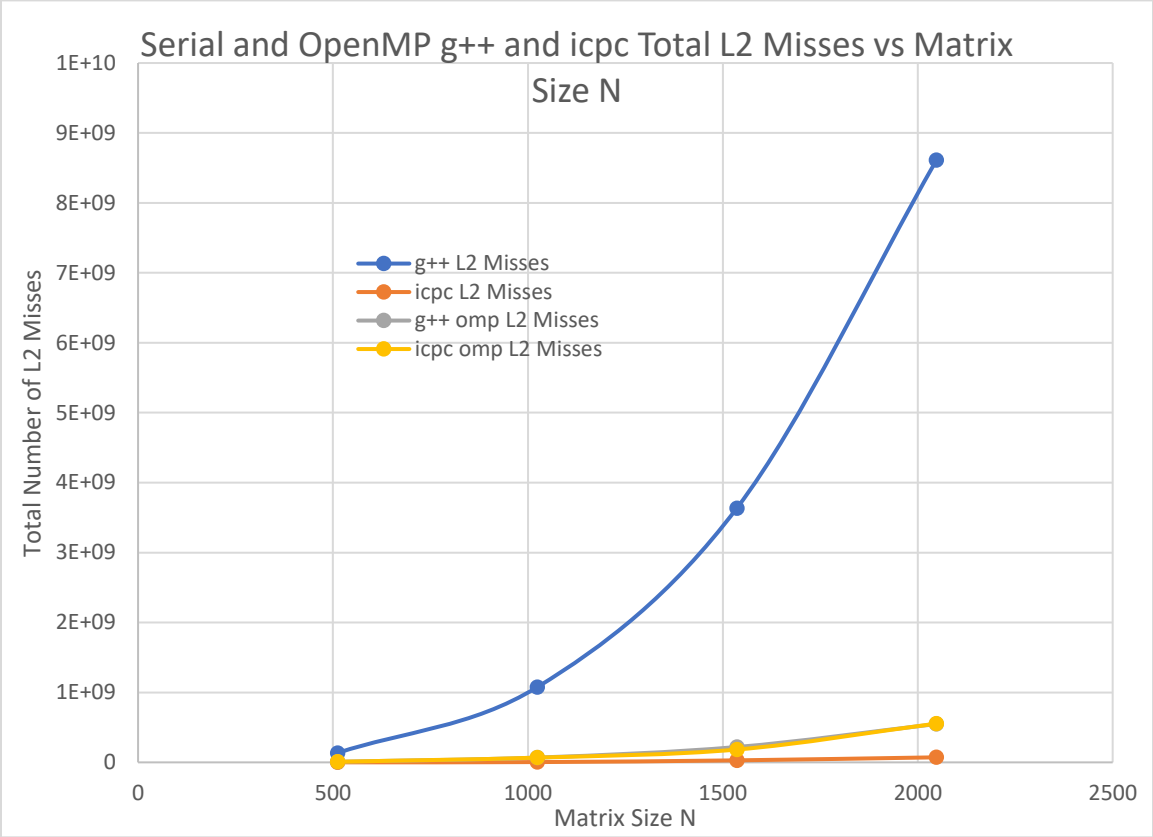

Figure 7. Total Number of L2 Misses for Each Matrix Size N

Conclusion

Throughout this experiment g++ and icpc compilers have been tested using one of their built-in options for optimizing code, -O3 for g++ and -fast for icpc, on a simple square matrix multiplication application. Using varying matrix sizes and collecting performance metrics such as overall execution time, critical loop execution time both serial and parallel, instruction count, total clock cycles, etc. In every test case the intel provided compiler has overall better performance over its g++ counterpart. This is to be expected as g++ is a more generalized and open source compiler and all the tests ran were on a machine using an intel processor, which it would be safe to assume that the more expensive intel compiler is better suited for optimization for intel hardware. Utilizing OpenMP to parallelize the application added significant performance increases for the g++ application yet provided mixed results when used with the intel compiler. It would be wise to compare the other compiler optimization options offered. In conclusion, the g++ compiler is a good general compiler but if it is performance and not cost that is required then the intel compiler would be of better use.

Appendix A

PAPI RESULTS

Table 4. Total Number of Instructions for g++ and icpc (Serial and OpenMP)

Matrix Size N	g++ Total Instructions	icpc Total Instructions	g++ omp Total Instructions	icpc omp Total Instructions
512	941889827	206269731	81036686	450205005
1024	7525642911	1630296363	593699039	266822077
1536	25388413065	5480040865	1937587494	1077253236
2048	60167454370	12963463450	4513004564	3035769914

Table 5. Total Number of Clock Cycles for g++ and icpc (Serial and OpenMP)

Matrix Size N	g++ Total Clock Cycles	icpc Total Clock Cycles	g++ omp Total Clock Cycles	icpc ompTotal Clock Cycles
512	827511765	300440670	285131510	595673689
1024	9052470116	2479660695	1110980107	979073123
1536	48124152643	8511245480	3892381958	3598842174
2048	4.86807E+11	19989495770	1.05328E+11	1.09851E+11

Table 6. Total Number of L1 Cache Misses for g++ and icpc (Serial and OpenMP)

Matrix Size N	g++ L1 Cache Misses	icpc L1 Cache Misses	g++ omp L1 Cache Misses	icpc omp L1 Cache Misses
512	135003153	8666359	9109507	8543531
1024	1083636666	99778901	72150691	69564609
1536	3647433847	337573040	233587144	196180169
2048	8691588513	799168072	614757460	625807404

Table 7. Total Number of L2 Misses for g++ and icpc (Serial and OpenMP)

Matrix Size N	g++ L2 Misses	icpc L2 Misses	g++ omp L2 Misses	icpc omp L2 Misses
512	134869945	247781	8722714	8486858
1024	1075662802	3975057	68493551	67376128
1536	3635232852	29013466	219651666	183303016
2048	8611548215	72037797	548136598	554814135