Parallelized (non-tiled):

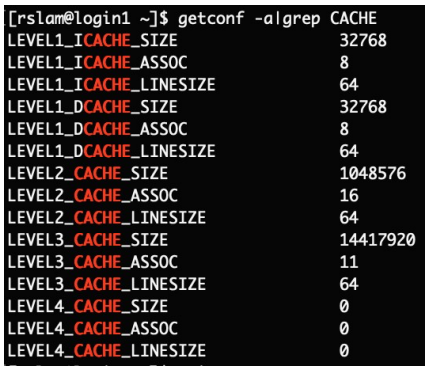
To parallelize the standard matrix multiplication, the OMP directives were added to evenly distribute the workloads amongst all the threads. Additionally, to further optimize the matrix-matrix multiplication, a condition was added to determine if the second matrix, matrix B, was a square matrix. If that condition passed, the program would transpose the second matrix, to take advantage of locality.

Additionally, with respect to the OMP directives, beyond adding a parallelized for loop and a reduction, not much else had to be implemented, as the default scheduling assigned is static, which is ideal for dense matrix multiplication. Because of the structure of a dense matrix, the workload can be equally distributed amongst all threads, to optimize the amount of work each thread will compute.

Parallelized Tiled:

To determine cache line size, the command, "getconf -a|grep CACHE", was executed into the wave supercomputer terminal. The command outputs the cache: sizes, associativity, and line sizes. As a result, to determine the blocksize, I computed the total L1 cache line size and the L2 cache line size, 64 + 64, which sums to a total cache line size of 128.

The output of the above command is shown below:



I also tried other numbers and found that either a cache block size too large or small would severely impact the performance of the tiling matrix multiplication negatively. As for the tiled matrix multiplication itself, to parallelize it, I had three outer loops to compute the tile size, and three inner loops to conduct the multiplication of the tiles. For the OMP directives, I included a collapse for the loops responsible for establishing the tiles, and a parallel for loop with a reduction on "sum" to compute the actual dot product on the established tiles. There

wasn’t a specified a scheduling type, because OMP defaults the scheduling type to static, which distributes an equal workload amongst the threads, which I felt was ideal for dense matrix multiplication.

Additionally, although there wasn’t a dramatic performance improvement from this, variables and indexes were computed at the outermost scope, to minimize resource utilization from unnecessary computation.

Tables:

Non-Parallelized Default Matrix Multiplication (Baseline)	
Dimensions	Time to Complete
1000x1000x1000	~ 1.24249 seconds
1000x2000x5000	~ 19.5882 seconds
9000x2500x3750	~ 185.546 seconds

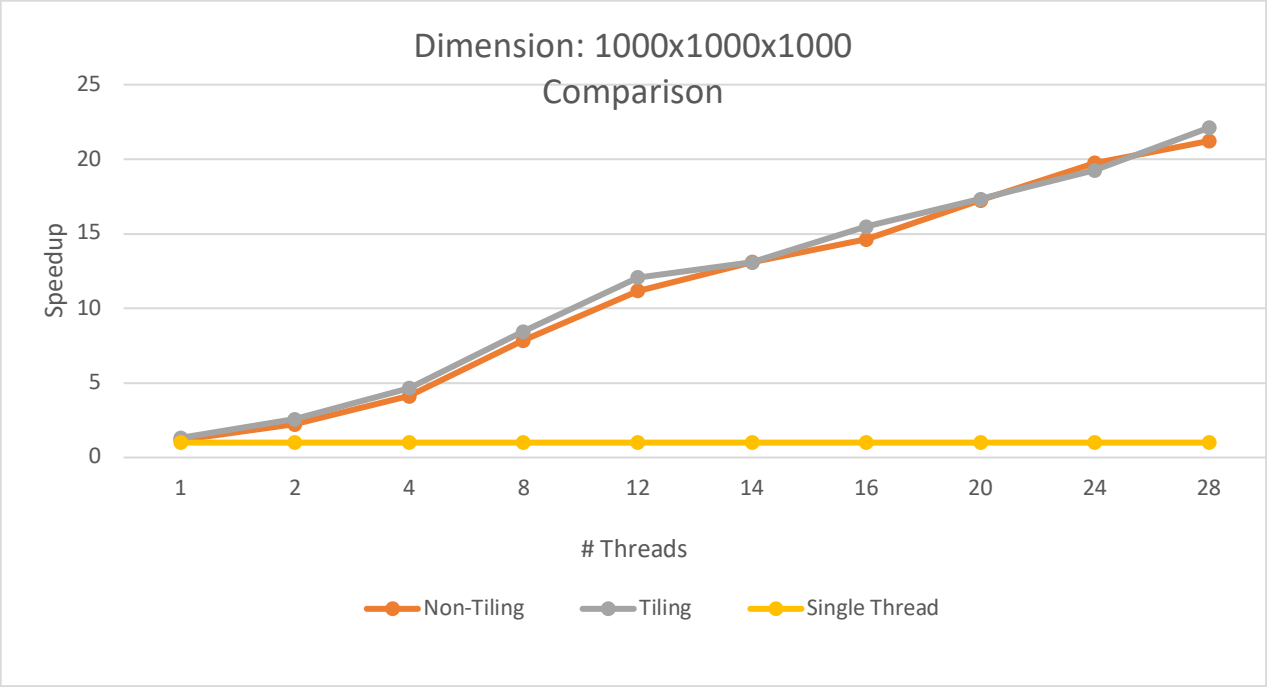
Dimensions: 1000x1000x1000			
Threads	Non-Tiling	Tiling	Single Thread
1	1.0646	0.942448	1.24249
2	0.558476	0.484346	1.24249
4	0.30112	0.26758	1.24249
8	0.158109	0.146883	1.24249
12	0.11114	0.102949	1.24249
14	0.0948803	0.0949097	1.24249
16	0.084982	0.0802652	1.24249
20	0.0720523	0.0717189	1.24249
24	0.0628789	0.0645167	1.24249
28	0.0585422	0.056187	1.24249

Dimensions: 1000x2000x5000			
Threads	Non-Tiling	Tiling	Single Thread
1	19.7464	9.43239	19.5882
2	10.7861	4.78676	19.5882
4	5.63109	2.59712	19.5882
8	3.0213	1.40793	19.5882
12	2.06997	1.01015	19.5882
14	1.79145	0.920979	19.5882
16	1.63951	0.789956	19.5882

20	1.42762	0.724285	19.5882
24	1.17834	0.657048	19.5882
28	0.890613	0.584853	19.5882

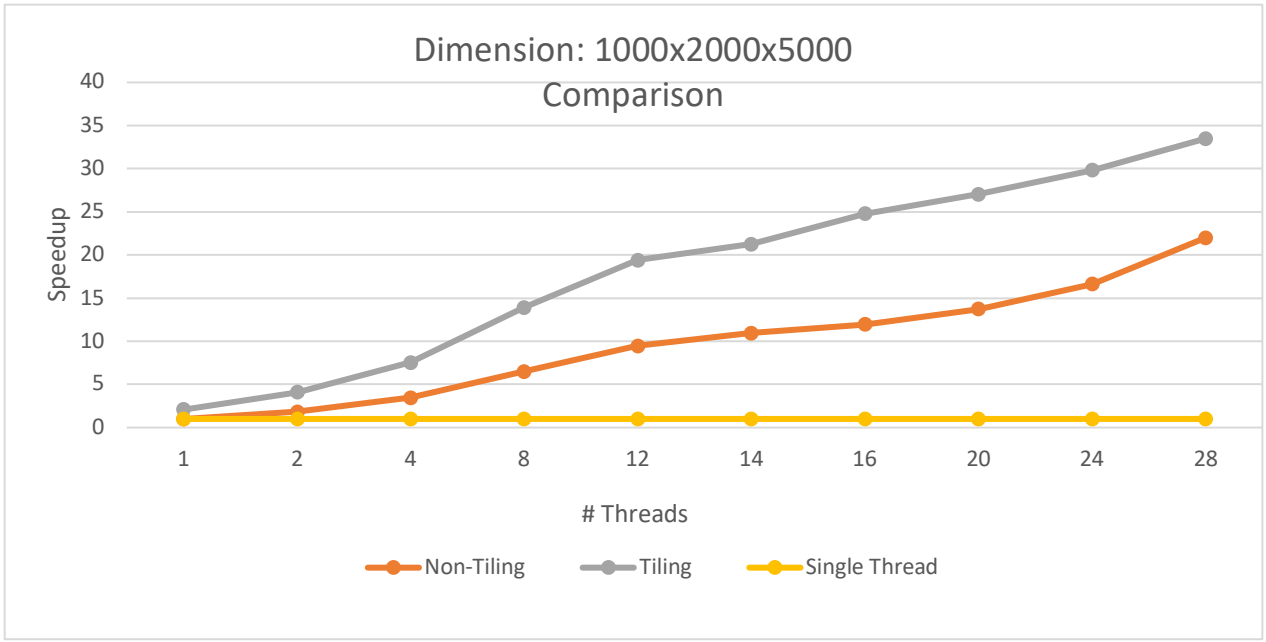
Dimensions: 9000x2500x3750			
Threads	Non-Tiling	Tiling	Single Thread
1	182.101	80.0023	185.546
2	94.8713	40.616	185.546
4	45.4885	21.9145	185.546
8	23.02	11.8637	185.546
12	16.7564	8.46575	185.546
14	13.9113	7.75357	185.546
16	12.7485	6.52814	185.546
20	10.2093	5.78067	185.546
24	8.16814	5.10533	185.546
28	6.96754	4.43156	185.546

Scaling Charts:



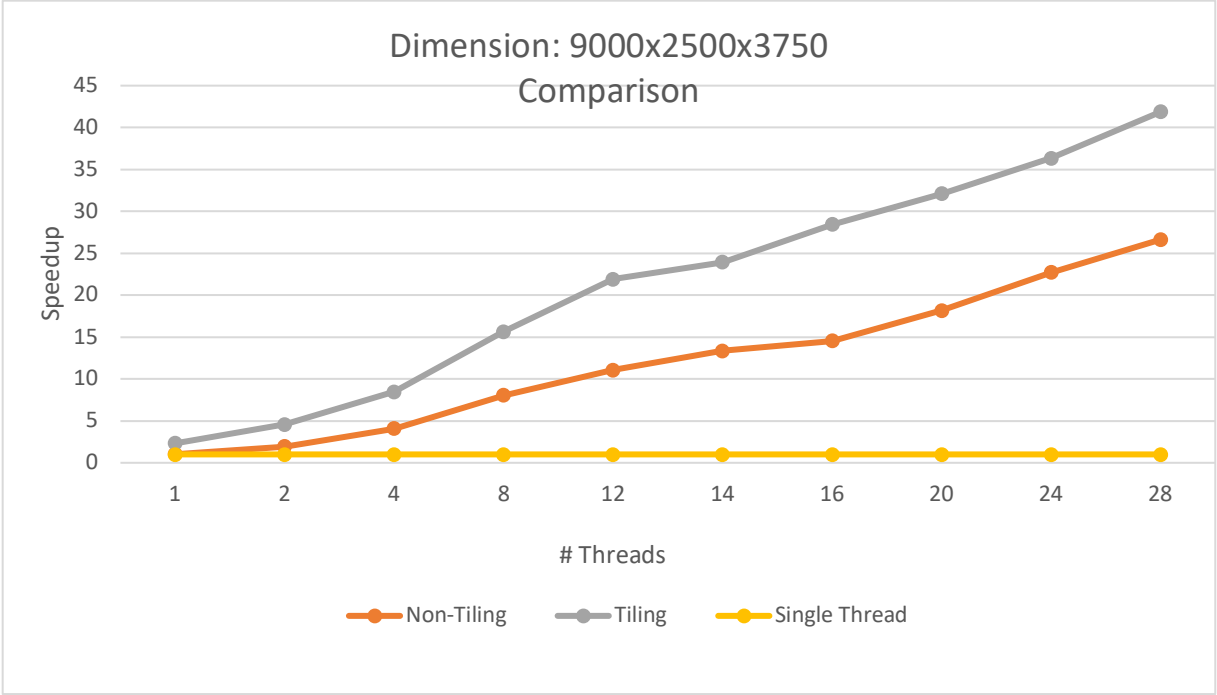
Dimension: 1000x1000x1000		
Tiling	Single Thread	Speedup
1.31836451	1.24249	1.31836451x
2.56529423	1.24249	2.56529423x
4.64343374	1.24249	4.64343374x
8.45904563	1.24249	8.45904563x
12.0689856	1.24249	12.0689856x
13.0912857	1.24249	13.0912857x
15.4798094	1.24249	15.4798094x
17.3244431	1.24249	17.3244431x
19.2584246	1.24249	19.2584246x
22.1134782	1.24249	22.1134782x

Dimension: 1000x1000x1000		
Non-Tiling	Single Thread	Speedup
1.0646	1.24249	1.16709562x
0.558476	1.24249	2.22478674x
0.30112	1.24249	4.12622875x
0.158109	1.24249	7.85843943x
0.11114	1.24249	11.1795033x
0.0948803	1.24249	13.0953422x
0.084982	1.24249	14.6206255x
0.0720523	1.24249	17.2442795x
0.0628789	1.24249	19.7600467x
0.0585422	1.24249	21.2238351x



Dimension: 1000x2000x5000		
Tiling	Single Thread	Speedup
9.43239	19.5882	2.0766953x
4.78676	19.5882	4.09216255x
2.59712	19.5882	7.5422776x
1.40793	19.5882	13.9127655x
1.01015	19.5882	19.3913775x
0.920979	19.5882	21.2688889x
0.789956	19.5882	24.7965709x
0.724285	19.5882	27.0448787x
0.657048	19.5882	29.8124338x
0.584853	19.5882	33.4925186x

Dimension: 1000x2000x5000		
Non-Tiling	Single Thread	Speedup
19.7464	19.5882	0.99198841x
10.7861	19.5882	1.81605956x
5.63109	19.5882	3.47858052x
3.0213	19.5882	6.48336809x
2.06997	19.5882	9.4630357x
1.79145	19.5882	10.9342711x
1.63951	19.5882	11.9475941x
1.42762	19.5882	13.7208781x
1.17834	19.5882	16.6235552x
0.890613	19.5882	21.9940648x



Dimension: 9000x2500x3750		
Tiling	Single Thread	Speedup
80.0023	185.546	2.31925832x
40.616	185.546	4.56829821x
21.9145	185.546	8.46681421x
11.8637	185.546	15.6398088x
8.46575	185.546	21.9172548x
7.75357	185.546	23.9303959x
6.52814	185.546	28.4224909x
5.78067	185.546	32.0976634x
5.10533	185.546	36.343586x
4.43156	185.546	41.8692289x

Dimension: 9000x2500x3750		
Non-Tiling	Single Thread	Speedup
182.101	185.546	1.01891807x
94.8713	185.546	1.95576534x
45.4885	185.546	4.07896501x
23.02	185.546	8.06020851x
16.7564	185.546	11.0731422x
13.9113	185.546	13.3377901x
12.7485	185.546	14.5543397x
10.2093	185.546	18.1742137x
8.16814	185.546	22.7158202x
6.96754	185.546	26.6300588x

Analysis:

After conducting experiments, which involved varying multiple parameters, such as, how different numbers of threads affects the efficiency of the computation, the dimensions of the

matrices being multiplied, differing methods of conducting matrix-matrix multiplication, and how all these factors correlate with one another, various conclusions have been determined. To begin with, with an increased number of threads, **there was a drastic increase in efficiency of computation**, which can be observed from the tables and charts above. However, as seen the charts, the productivity of threads **seemed to have diminished returns after 14**, which is a common trend witnessed throughout the charts.

Also, with an increase of size in the inputted matrices, the runtime for computation was much higher, which can be observed in the Y-axis of the charts. For example, with the 1000x1000x1000 matrix, the serial computation only took about 1.2 seconds. However, with the 9000x2500x3750 matrix, the time of computation was over 180 seconds. **Not only was it determined that the larger the size of the inputted matrices, matrix A and B, the longer the run-time, but also, the greater impact more threads had on the computation.** For example, we begin to see diminishing returns in the 1000x1000x100 matrix at around 8 threads, however, for the 9000x2500x3750 matrix, the impact of threads begins to lessen after about 20 threads. **This is because the larger the number of elements, the greater the workload for a single thread. Naturally, if we have more elements in a dense matrix, we need more threads to equally divide the workload.**

Furthermore, it was observed that implementing tiling also greatly benefitted the computation, the larger the inputted matrices were. When observing the speed up tables, we can see that as the matrix sizes increases, there is a significant speedup from the serial algorithm, when using tiling. Although all the implementations utilize a 2D thread friendly contiguous array, which exploits spatial locality, tiling exploits temporal locality. Because of this, the larger the input matrices, the more likely we are to get a hit, and reuse data.

A common characteristic observed amongst all the charts, was a general decrease in runtime, with an increase in threads. It was also observed that there was a certain threshold in the number of threads for each matrix dimension, in which returns of computational efficiency greatly diminished. **The reason for this is because although an increase of threads leads to a division of the overall workload, there is a certain point in which bottlenecks begin to form.** As a result, a common characteristic between the experiments is that regardless of the shapes of the dense matrices, there is an optimal number of threads to conduct the computations. In summation, as discovered in the experiments**, tiling greatly impacts the computational**

efficiency in dense matrix-matrix multiplication due to exploitation of locality, **the larger the matrices, the greater number of threads are needed to reduce runtime to an optimal amount**, and **regardless of the shape or size of a dense matrix, there is a point in which an increase in threads provides significantly diminished results**.