Parallel Matrix Multiplication

Parallel Programming

December 27, 2022

1 Introduction

The CPU analysis of the A^TB and the A^TB^T code was performed on the $kp307$ and the $kp326$ clusters respectively. The GPU performance analysis of the A^TB and the A^TB^T code was performed on the $kp359$ machine. All performance numbers presented in the figures and the subsections are reported in GFLOPs.

2 CPU Performance Analysis

The CPU parallel code uses various code files. The dynamic switching mechanism for both the matrix computations is implemented in the ${}_par.c$ files. The remaining parallel files implement individual optimizations and the main file must be modified to run them individually. For compilation, create a new directory and run **gcc -fopenmp -O3 -o** $< execname >$ ***.c** for any of the presented matrix codes.

2.1 Analysis for A^TB

For the current problem, it was found that the ikj permutation offered high speedups. The reasoning might lie into the j loop's ability to be vectorized due to an access stride of 1 and no data dependence. Furthermore, attempts to tile the complete ikj permutation did not offer significant speedups. However, individual tiling of i loop or the outer ik loop led to high improvements in performance. Tiling of j loop also did not lead to the best performance amongst all configurations but this configuration became a necessity when the value of N_i was found to be less than 16 in which case high values of N_j offered more parallel work for the j loop. Due to the j loop's ability to be vectorized and high access stride for the k or the i loops, we instead maintained the j loop as the innermost loop for these matrix sizes and strip mined this loop to enable parallelization for the outer tiled jj loop.

A summary of the configurations which led to the highest performance for different problem size parameters is summarized below:

- ik tiling with k unrolled by a factor of 2 for $N_k > 256$ and $N_i > 64$ and $N_j > 64$
- i tiling with k loop unrolling for $N_i > 64$

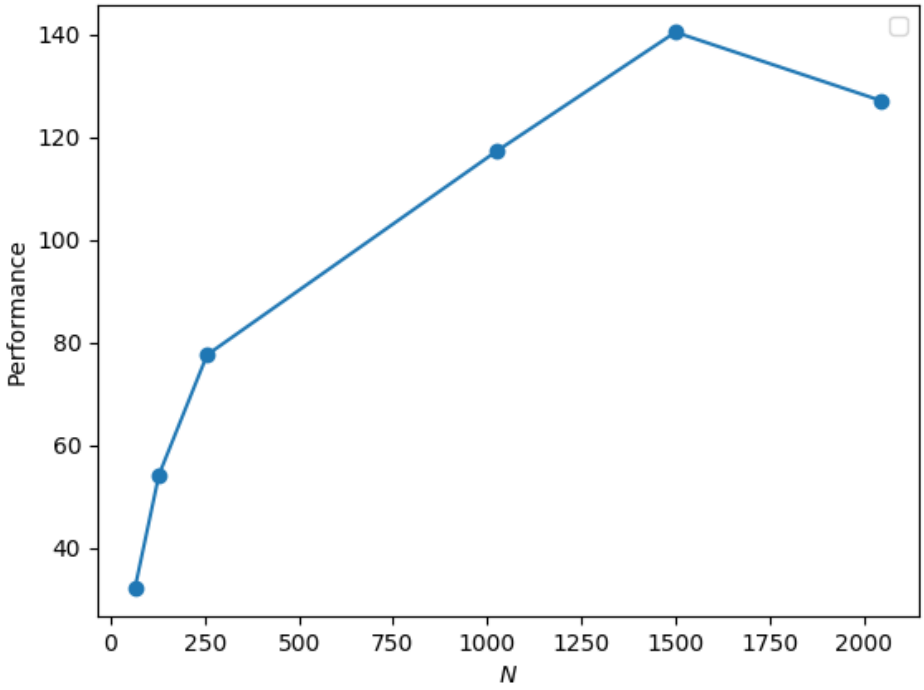

Figure 1: Performance variation with N where $N_i = N_j = N_k = N$ for the $A^T B$ CPU code for ik tiling configuration

- Parallel Reduction code for $N_i <= 64$, $N_j <= 64$ and $N_k >= 2048$

When values of N_i or N_j could not offer high parallelization, attempt was made to have a parallel reduction mechanism. More specifically, the k loop iterations were divided amongst threads and each thread maintained a private matrix. The global shared matrix was updated atomically in another loop. The approach could only offer high speedups beyond 30 GFLOPs at $N_k > 2048$.

Figure 1 shows the ability for the tiled ik configuration to achieve high GFLOP counts at high problem size parameters. The tiled configuration routinely clocked more than 100 GFLOPs for $N > 1024$ where $N_i = N_j = N_k = N$.

2.2 Analysis for $A^T B^T$

For the current problem, through data dependency analysis, it was found that there was no clear way to vectorize the loop even after performing suitable loop permutations. The k loop index carried an output dependence due to which it could not be parallelized. This left one with two configurations namely ikj and jki. It was found that despite no vectorization benefit, the ikj permutation outperformed the jki permutation for most matrix configurations. Both loop tiling and loop unrolling were tested to check the obtained benefits and heavy benchmarking was performed. A few configurations were found to yield maximum

performance at different problem size parameters. These configurations are shown as follows:

- Complete ikj tiling with k loop unrolling at $N_i, N_j, N_k > 1024$
- ij tiling with k loop unrolling at $N_i, N_j > 64$
- jk tiling at $N_k, N_j > 1024$ with i loop outermost
- k tiling with k loop unrolling or Plain k loop unrolling at $N_k > 8$
- Parallel Reduction Code based on the Histogram Parallelization concept for $N_i * N_j < 64$ and $N_k > 1024$

Furthermore, it was found that even with tiled code, on many occasions, standard tiling on the ikj permutation did not yield any significant benefit. For example, standard tiling for the kj configuration did not yield high GFLOP values. However, tiling for the jk loop in the ijk permutation with the k loop unrolled yielded a massive performance benefit with performance exceeding 100 GFLOPs for some variation of problem size parameters. The ij tiling configuration was even more convoluted where the outer tiled ij loops were followed by the inner kij permutation. The reasoning for this behavior might lie in data locality analysis where the tiled ij configuration might offer high cache hits if kept in the innermost loop with the outer tiled i loop parallel.

Another important factor to note is the tilesize of the tiled loops. It was found that performance improved until a tile block size equal to 16 beyond which the performance deteriorated. Furthermore, for the current problem, k loop unroll factors until 8 were giving high performance beyond which the performance deteriorated. These observations are only valid for the $kp326$ machine on which the computation was performed. Heavy performance variations were observed across a different machine where a tilesize of 8 was found to give good results. The reasoning might stem into the cache sizes available on a specific machine and whether a certain k loop unroll factor starts causing additional register overhead depending on the machine characteristics.

When both N_i and N_j were less than 8, parallelizing any one of these loops did not yield high scalability. For these configurations, a separate reduction concept was used whereby the k iterations were divided in parallel. A separate private matrix was used to fill data for each one of the threads followed by an atomic update of the final shared matrix. Although the approach looks scalable in theory but it only scaled well if N_k was greater than 1024. In general, if the values of N_i was greater than 64, the i loop alone provided high parallelization work since the number of threads on most machines are limited to around 55.

Figure 2 shows the performance variations observed as N increases where $N_i = N_j = N_k = N$. It is observed that the complete ikj tiling maintains more than 50 GFLOPs and offers high speedups beyond 100 GFLOPs for $N > 1500$. Additional benchmark data collected for various configurations is also attached towards the end of this report.

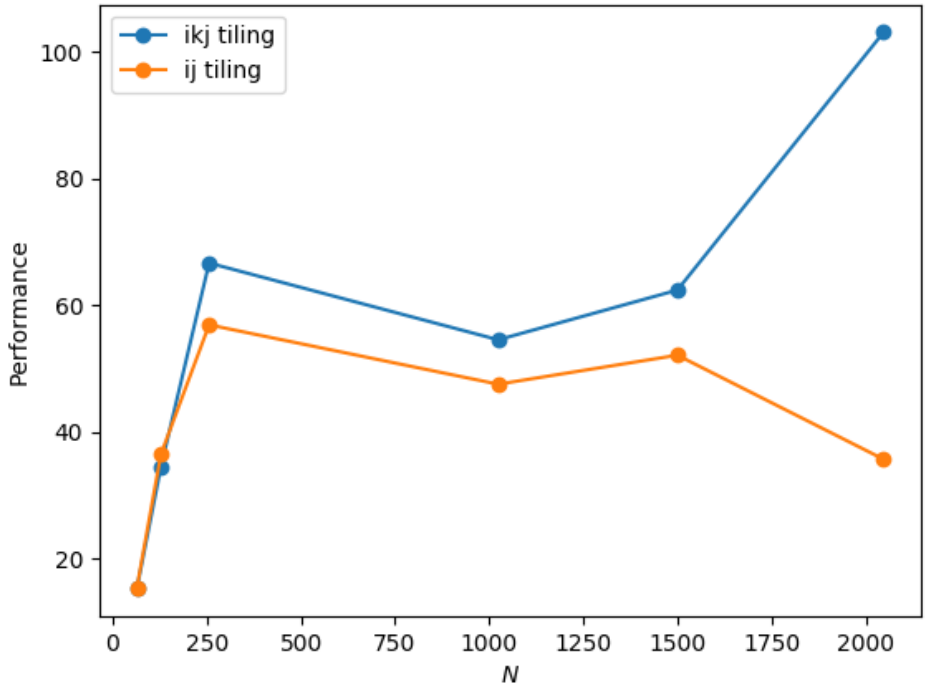

Figure 2: Performance variation with N where $N_i = N_j = N_k = N$ for the $A^T B^T$ CPU code

3 GPU Performance Analysis

Several GPU code versions were developed to analyze the effects of different levels of loop unrolling and utilization of shared memory as problem size parameters varied. These versions further fall into two different categories. The first category which we refer to as **Block** kernels use square shared memory matrices. For this category, both the matrices A and B are utilized for shared memory and a $BSIZE \times BSIZE$ shared memory matrix is created for each one of these. The second category used blocksize of 1 in the Y direction and is referred to as **Slim** kernel in the next subsections. For this category, only one shared memory array is created corresponding to either A or B depending on whether there is repeated access to the matrix for a single block of threads. Loop unrolling was carried out for both of these versions for each one of the three loops and a separate version was created which had all of the three loops unrolled.

A dynamic switching selection was performed for the different code versions depending on the problem size parameters. A separate function **launcher** performs this switching selection for both the matrix codes in the **main.cu** file. To compile the code versions, create a fresh directory for the **atb** or the **atbt** codes one is trying to analyze and run the command, **nvcc -O3 -o atb *.cu** with the submitted main file.

For each one of the code version described in the subsections, thread divergence issues were encountered due to heavy use of conditionals required to handle arbitrary values of problem

size parameters. These issues became a major bottleneck when N_i or N_j was less than 16. To minimize thread divergence, we take the route of decreasing the block size parameter when N_i or N_j were less than 32 in order to maintain correctness for these parameters.

3.1 Analysis for $A^T B$

All code versions in this subsection mapped the i loop to the Y direction of the thread block and the j loop to the X direction of the thread block. Loop unrolling and the utilization of shared memory were found to significantly affect the performance of the current problem. Figure 3 shows a benchmark at $N_i = N_j = N_k = 1024$ where the performance of the loop unrolled shared memory code is found to completely outperform the other code versions. Furthermore, even for the slim version, the three loop unrolled version outperforms the basic implementation.

Sensitivity of the developed code versions on the values of problem size parameters was studied carefully. Various benchmarks were gathered and it was found that overall performance for all developed code versions improved with increase in N_i and N_j. However, increase in N_k did not show a clear relationship. Benchmark data was collected at $N_i = N_j = 128$ which can also be seen in Figure 4. For the slim configuration, the performance increased rapidly but only until a certain point after which it showed a sharp jump downwards and then either remained the same or marginally increased. For the Block configuration, the performance showed a similar trend but without any downward spike while always remaining less than the Slim configuration. Furthermore, this uneven relationship was not valid at higher values of N_i and N_j where the performance always showed an upward trend. As a result of this observation, the switching selection implemented in code takes care to use the Slim kernels when N_i and N_j are small and N_k is large.

Benchmarks were also gathered at $N_i = N_k = 128$ with varying N_j. Figure 5 shows the performance behavior with N_j for the selected parameters. It was found that the Block based shared memory implementation increases exponentially and completely outperforms the slim versions beyond a certain value of N_j. The same behavior was viewed for other values of N_i and N_k with higher values leading to an even further improvement in the increase factor. Hence, care was taken to use the Block based configurations for high values of N_j. Similar behavior was also observed as the value of N_i varied with fixed N_j and N_k.

3.2 Analysis for $A^T B^T$

All code versions in this subsection mapped the i loop to the X direction of the thread block and the j loop to the Y direction of the thread block. It was found that loop unrolling and the usage of shared memory play a much bigger role for this problem. More specifically, the usage of block based shared memory almost always outperformed the basic implementation for the current problem. Loop unrolling of the j and the k loops led to an overall much higher performance gain at different problem size parameters. Furthermore, at $N_i > 1024$, $N_j > 1024$ and $N_k > 1024$, the three way unrolling of all loops outperformed the other implementations by a much bigger margin. An example benchmark is provided in Figure 6 where the shared memory kernel with all three loops unrolled shows more than 1500

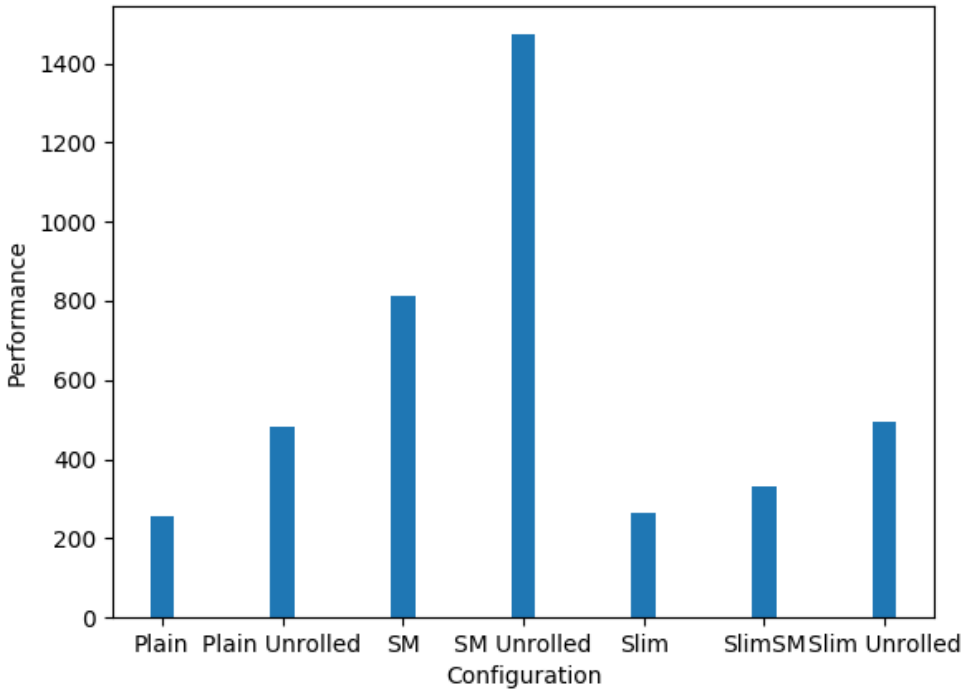

Figure 3: Performance of different configurations at $N_i = N_j = N_k = 1024$. **Plain**: No optimizations, **Plain unroll**: Unrolling of all three loops, **SM**: GPU kernel with square shared memory arrays, **SM unroll**: SM kernel with unrolling **Slim**: GPU kernel with $Blocksize = 1$ in the Y direction, **Slim SM**: Slim kernel with shared memory, **Slim unroll**: Slim kernel with unrolling

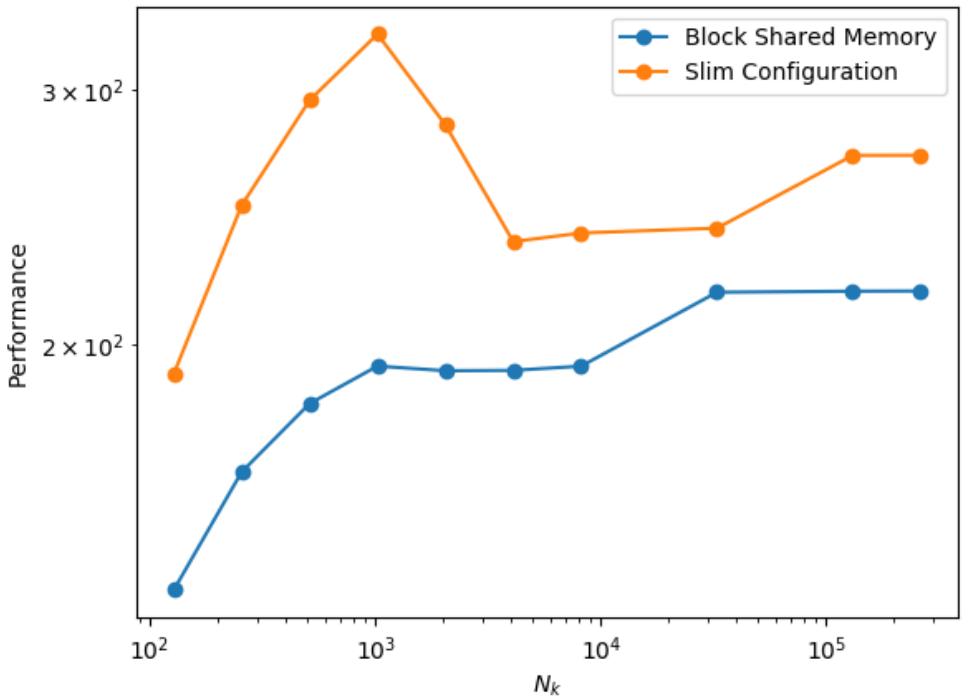

Figure 4: Performance variation with N_k where $N_i = N_j = 128$. **Block Shared Memory**: GPU kernel with square shared memory arrays, **Slim**: GPU kernel with $BSIZE = 1$ in Y direction

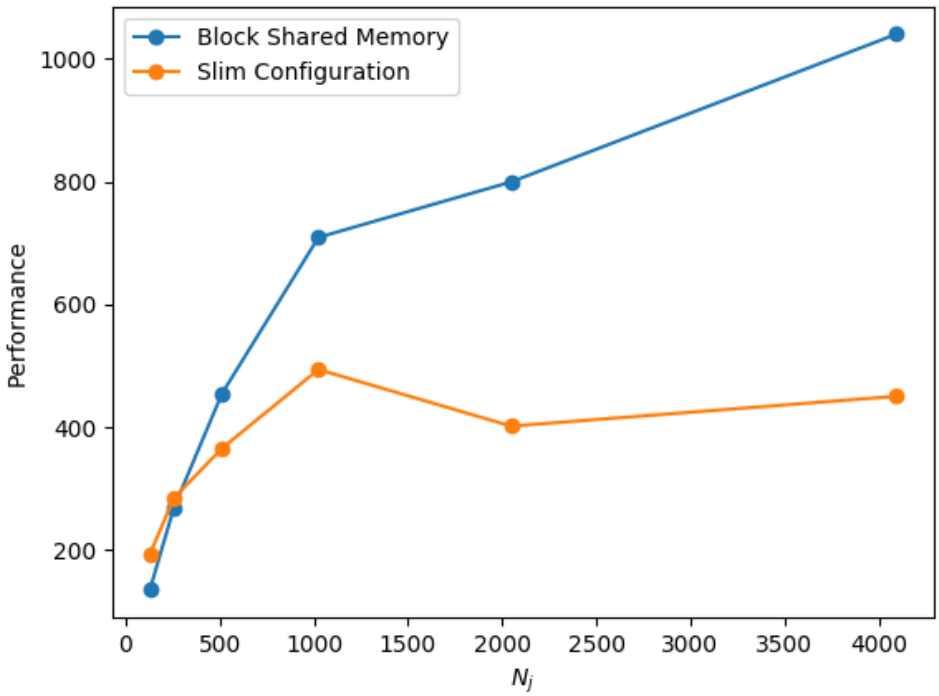

Figure 5: Performance variation with N_j where $N_i = N_k = 128$. **Block Shared Memory**: GPU kernel with square shared memory arrays, **Slim**: GPU kernel with $BSIZE = 1$ in Y direction

GFLOP performance value outperforming the rest by a very significant margin. Figure 9 further demonstrates this observation by showing the variation of performance of both the naive kernel and the shared memory kernel with loop unrolling. The shared memory kernel with loop unrolling shows a much higher performance improvement as N_j increases.

In order to analyze the selection of configuration at different problem size parameters, a sensitivity analysis was performed. More specifically, variation of performance of each one of the code versions was studied as only one of N_i, N_j or N_k varied by keeping the other two fixed. It was found that when any one of N_i or N_j or N_k was equal to 128, it was not possible to analytically determine whether the **Slim** and the **Block** versions performed better. Figure 7 shows the best performing configuration around the point when any one of these parameters is equal to 128.

To be more precise, it was found that increases in any one of the three parameters improved the performance of both the **Slim** and the **Block versions**. However, relative performance improvement varied. It was also found that for highly skewed input matrices A or B around this point, the performance of **Slim** kernels performed better. As can be observed from Figure 7, **Slim** kernel outperformed the **Block** version when Nk increased around this point while the **Block** version outperformed when N_j increased around this point. Hence, the value 128 was considered an inflection point from a performance viewpoint.

Figure 8 further validates this observation where both the shared memory kernel and the slim kernels show equivalent performance at $N_i = 512, N_j = 128, N_k = 32$.

As a result of the above observations, the value 128 is frequently used to make decisions related to which kernel must be utilized within the submitted code file.

4 CPU vs GPU Comparisons

The CPU codes provided high speedups but their speedups were limited for high values of problem size parameters. The highest CPU performance values were limited to around $130-140$ GFLOPs for both the matrix computations. On the other hand, GPU speedups went beyond 1500 GFLOPs routinely for high values of problem size parameters. Furthermore, due to the ease with which atomic operations can be implemented and lower latency offered by these operations on CPUs, it was possible to incorporate parallelization of highly skewed matrices where $N_i < 16$, $N_j < 16$ and $N_k > 2048$. This was made possible by having private matrices for each one of the threads and then using atomic operations to update the shared matrix. A similar optimization was attempted for GPU codes but it must be noted that barrier synchronization for GPU codes is only valid within a specific thread block. Across thread blocks, CUDA provides atomic operations but the latency of these operations is excessive and did not yield very significant gains even for such skewed matrices. Hence, it was found to be very difficult to achieve high speedup values for these matrices on GPUs. Furthermore, little material was found to be present online on performing sum reduction of an array on a GPU. Most material performed reduction within a specific thread block and then relied on CPUs to perform the final addition across thread blocks. Hence, skewed matrices were prone to thread divergence and were taken care of by having lower sized thread

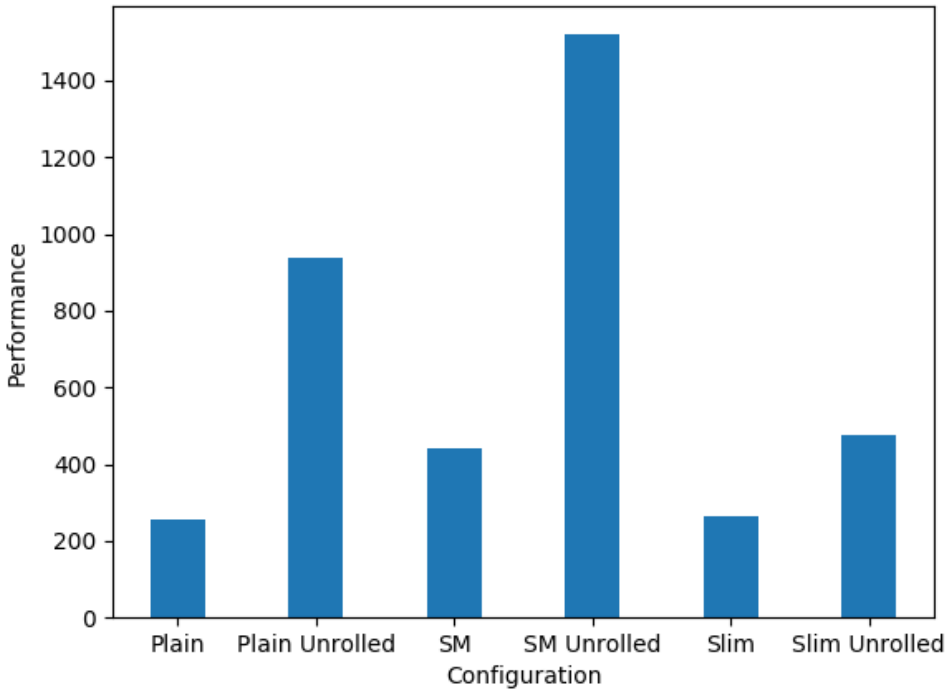

Figure 6: Performance of different configurations at $N_i = N_j = N_k = 1024$. **Plain**: No optimizations, **Plain unroll**: Unrolling of all three loops, **SM**: GPU kernel with square shared memory arrays, **SM unroll**: SM kernel with unrolling **Slim**: GPU kernel with $Blocksize = 1$ in the Y direction, **Slim unroll**: Slim kernel with unrolling

Ni	Nj	Nk	Best Performing Configuration
128	128	128	Slim kernel with SM
256	128	128	Slim kernel with SM
128	256	128	Slim kernel with SM
128	128	256	Slim kernel with SM
128	128	512	Slim kernel with SM
128	128	1024	Slim kernel with SM
128	128	2048	Slim kernel with SM
128	128	8192	Slim kernel with SM
128	128	8192*16	Slim kernel with SM
512	128	128	SM kernel with j unroll
128	512	128	SM kernel with j unroll
128	512	256	SM kernel with j unroll
128	512	512	SM kernel with j unroll
128	512	64	SM kernel with j unroll
128	512	32	Slim SM kernel
512	128	32	Both SM kernel and slim SM kernel
1024	128	32	Both SM kernel and slim SM kernel
2048	128	32	SM kernel and SM kernel with k unroll
128	1024	32	Slim SM kernel
128	2048	32	Both slim SM kernel and SM kernel
128	4096	32	SM kernel

Figure 7: Best Performing configuration at different problem size parameters for $A^T * B^T$. **SM kernel**: GPU kernel using Square shaped shared memory array, **Slim Kernel**: GPU kernel with $Blocksize = 1$ in the Y direction, **j unroll**: J loop unrolled, **k unroll**: K loop unrolled

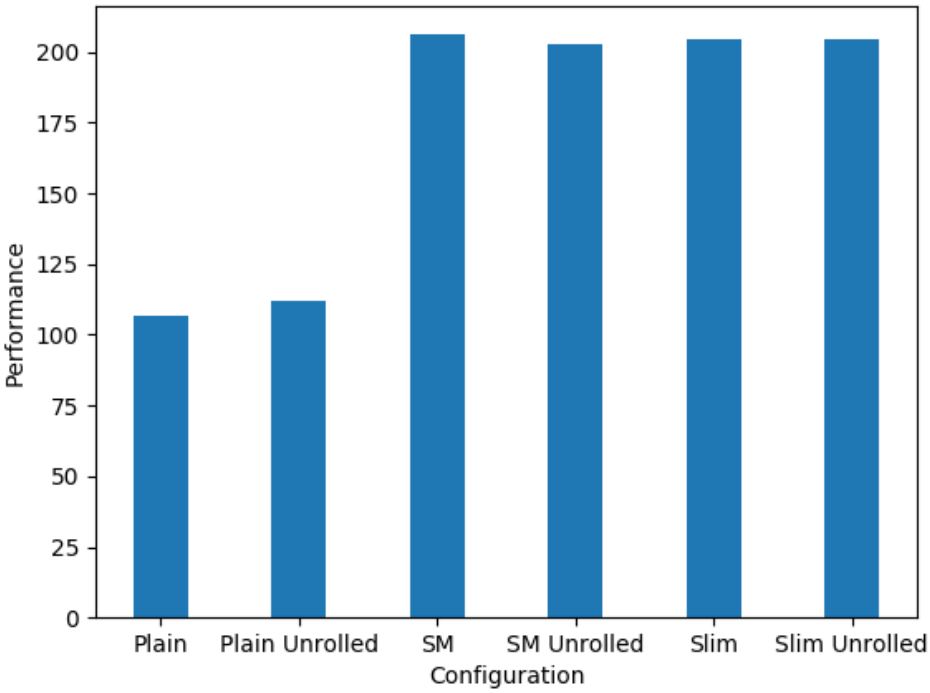

Figure 8: Performance of different configurations at $N_i = 512, N_j = 128, N_k = 32$. **Plain**: No optimizations, **Plain unroll**: Unrolling of all three loops, **SM**: GPU kernel with square shared memory arrays, **SM unroll**: SM kernel with unrolling **Slim**: GPU kernel with $Blocksize = 1$ in the Y direction, **Slim unroll**: Slim kernel with unrolling

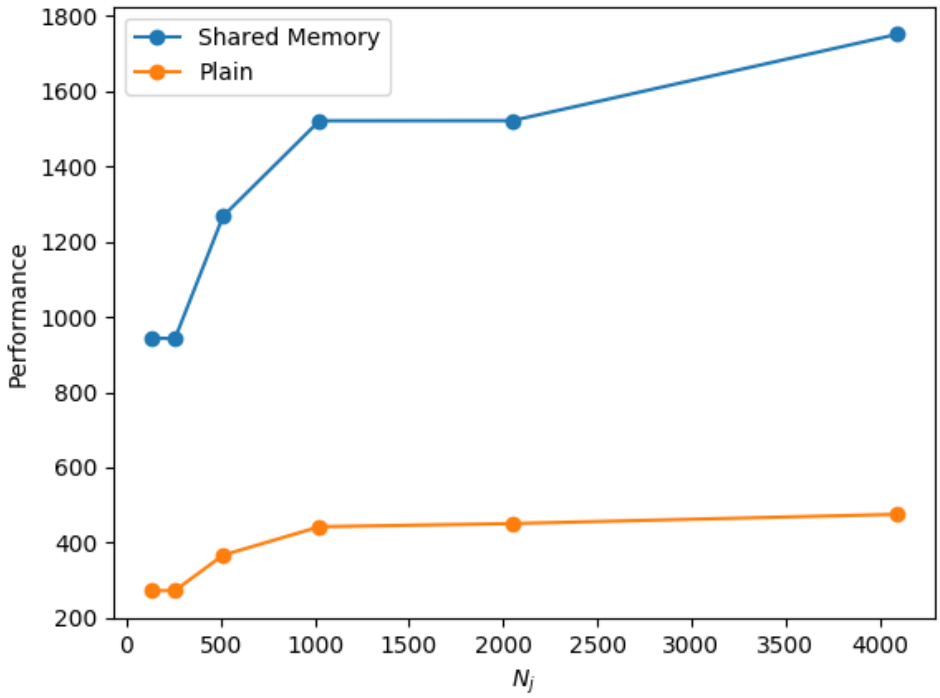

Figure 9: Performance variation with N_j at $N_i = N_k = 1024$. **Plain**: No shared memory utilization, **Shared Memory**: GPU kernel with square shared memory arrays

blocks with the slim configuration.

5 Appendix

This section shows code used to perform benchmarking on CHPC cluster and the data collected as a result of this effort.

```
from os import listdir
from os.path import isfile, join
import os
import re
import time

def getfilenames(dir_path, filenames, mainfiles):

    filenames[dir_path] = []

    for f in listdir(dir_path):

        if isfile(join(dir_path, f)):

            if( f.endswith( "main.cu" ) ):
                mainfiles[dir_path] = join( dir_path, f )
            else:
                filenames[dir_path].append( join(dir_path, f) )

        else:

            getfilenames( join(dir_path, f), filenames, mainfiles )


def runscript():

    dir_path = os.path.dirname(os.path.realpath(__file__))

    filenames = dict()
    mainfiles = dict()

    Ni = "1024"
    Nj = "1024"
    Nk = "1024"

    getfilenames( dir_path, filenames, mainfiles )
    # getfilenames(dir_path + "/SM", filenames)
    # getfilenames(dir_path + "/Plain", filenames)

    for dirval, filevals in filenames.items():

        r = re.compile(".*\.cu")
        newlist = list(filter(r.match, filevals)) # Read Note below
        # print(newlist)

        compile_command_prefix = "nvcc -O3 -o "
```

```
        compile_command_mid = " " + mainfiles[dirval] + " "

        for filename in newlist:

            execfilename = filename[:-3]
            run_cmd = execfilename + " " + Ni + " " + Nj + " " + Nk
            print(run_cmd)

            compile_command = compile_command_prefix +
            execfilename + compile_command_mid + filename
            # print(compile_command)
            os.system( compile_command )

            time.sleep(3)
            os.system( run_cmd )



def runscript1():

    dir_path = os.path.dirname(os.path.realpath(__file__))

    filenames = dict()
    mainfiles = dict()

    Ni = "128"
    Nj = ""
    Nk = "128"

    getfilenames(dir_path, filenames, mainfiles)
    # getfilenames( dir_path + "/Plain_ijkunroll", filenames, mainfiles )
    # getfilenames(dir_path + "/SM", filenames)
    # getfilenames(dir_path + "/Plain", filenames)

    vals = [128, 256, 512, 1024, 2048, 4096]
    # vals = [8192, 8192*4, 8192*16, 8192*32]

    for val in vals:

        Nj = str(val)

        for dirval, filevals in filenames.items():

            r = re.compile(".*\.cu")
            newlist = list(filter(r.match, filevals)) # Read Note below
            # print(newlist)

            compile_command_prefix = "nvcc -O3 -o "
            compile_command_mid = " " + mainfiles[dirval] + " "

            for filename in newlist:

                execfilename = filename[:-3]
                run_cmd = execfilename + " " + Ni + " " + Nj + " " + Nk
```

```
                print(run_cmd)

                compile_command = compile_command_prefix + \
                execfilename + compile_command_mid + filename
                # print(compile_command)
                os.system( compile_command )

                time.sleep(3)
                os.system( run_cmd )


    # print(filenames)



if __name__ == "__main__":
    runscript1()
```

CPU Performance Data for ATB Operation

- **bs: Tilesize value for the specified tilinng configuration**
- **ijk or ikj: Loop permutation**
- **Best performance shown on 1/2/4/8/10/15/20/27/55 threads**

Matrix Configuration	Permutation	Unroll Loop, factor	Parallel Loop	Tiling	Performance
ni = nj = 1024*8, nk = 16	ikj	None	i	None	Best Performance (GFLOPS): 4.19 8.35 16.02 31.78 39.18 56.88 69.63 52.98 78.56
ni = nj = 1024*8, nk = 16	ikj	j, 2	i	None	Best Performance (GFLOPS): 3.21 5.34 7.72 13.81 13.41 20.23 27.00 19.25 38.15
ni = nj = 1024*8, nk = 16	ikj	k, 2	i	None	Best Performance (GFLOPS): 4.79 9.52 18.35 36.35 44.95 66.46 44.46 57.54 78.13
ni = nj = 1024*8, nk = 16	ikj	k, 4	i	None	Best Performance (GFLOPS): 4.63 9.13 17.85 35.41 43.61 63.48 80.19 58.32 80.54
ni = nj = 1024*8, nk = 16	ikj	i, 2	i	None	Best Performance (GFLOPS): 4.33 6.34 8.21 14.78 18.47 26.92 35.96 23.47 41.58
ni = nj = 1024*8, nk = 16	ikj	i, 4	i	None	Best Performance (GFLOPS): 1.57 2.75 4.61 7.32 7.04 10.36 12.85 11.56 17.97
ni = nj = 1024*8, nk = 16	ikj	K,2	i	Ik, bs = 4	Best Performance (GFLOPS): 4.56 9.04 17.44 34.63 42.08 58.45 50.05 51.44 76.01

ni = nj = 1024*8, nk = 16	ikj	K,2	i	I, bs = 2	Best Performance (GFLOPS): 4.80 9.54 18.25 36.27 44.37 63.02 72.29 79.99 79.97
ni = nj = 1024*8, nk = 16	ikj	K,2	i	I, bs = 4	Best Performance (GFLOPS): 4.72 9.07 17.88 35.43 43.73 64.14 55.70 57.30 79.03
ni = nj = 1024*8, nk = 16	ikj	K,2	i	I, bs = 8	Best Performance (GFLOPS): 4.77 9.30 18.20 36.13 44.33 65.06 46.58 61.20 78.60
ni = nj = 4096, nk = 64	ikj	None	i	None	Best Performance (GFLOPS): 4.47 8.91 17.65 35.21 43.33 64.22 85.83 76.46 123.34
ni = nj = 4096, nk = 64	ikj	K,2	i	None	Best Performance (GFLOPS): 5.27 10.49 20.77 41.39 51.56 76.08 66.94 71.42 140.61
ni = nj = 4096, nk = 64	ikj	K,4	i	None	Best Performance (GFLOPS): 5.36 10.67 20.87 41.31 51.00 74.74 49.82 66.45 132.45
ni = nj = 4096, nk = 64	ikj	None	i	I, bs = 4	Best Performance (GFLOPS): 4.49 8.95 17.73 35.34 43.22 64.07 85.51 62.24 119.51
ni = nj = 4096, nk = 64	ikj	K,2	i	I, bs = 4	Best Performance (GFLOPS): 5.27 10.47 20.79 41.43 51.32 75.54 67.28 73.55 140.13
ni = nj = 4096, nk = 64	ikj	K,2	i	I, bs = 8	Best Performance (GFLOPS): 5.06 10.05 19.90 39.72 48.71 70.71 93.34 69.34 128.91

ni = nj = 4096, nk = 64	ikj	K,2	i	ik, bs = 4	Best Performance (GFLOPS): 5.01 9.94 19.81 39.56 48.77 71.90 94.13 80.29 123.24
ni = nj = 4096, nk = 64	ikj	K,2	i	ik, bs = 8	Best Performance (GFLOPS): 5.13 10.18 20.20 40.16 49.23 71.92 94.26 69.91 126.25
ni = nj = 2048, nk = 256	ikj	None	i	None	Best Performance (GFLOPS): 5.15 10.34 20.66 40.85 51.02 75.04 47.59 63.23 126.13
ni = nj = 2048, nk = 256	ikj	K,2	i	None	Best Performance (GFLOPS): 5.57 11.09 21.98 43.61 54.08 80.33 68.16 73.00 144.05
ni = nj = 2048, nk = 256	ikj	K,4	i	None	Best Performance (GFLOPS): 4.89 9.77 19.40 38.49 48.17 71.52 94.41 66.75 131.64
ni = nj = 2048, nk = 256	ikj	None	i	I tiled with bs = 4	Best Performance (GFLOPS): 5.29 10.60 21.13 41.76 51.34 75.09 100.89 136.93 122.98
ni = nj = 2048, nk = 256	ikj	K, 2	i	I tiled with bs = 4	Best Performance (GFLOPS): 5.58 11.10 22.09 43.49 53.71 78.73 104.19 74.45 138.96
ni = nj = 2048, nk = 256	ikj	K, 2	i	I tiled with bs = 8	Best Performance (GFLOPS): 5.57 11.08 22.09 43.13 52.88 75.71 66.47 70.31 139.46

CPU Performance Data for ATBT Operation

- **bs: Tilesize value for the specified tilinng configuration**
- **ijk or ikj: Loop permutation**
- **Best performance shown on 1/2/4/8/10/15/20/27/55 threads**

Matrix Configuration	Permutation	Unroll Loop, factor	Tiling	Performance
ni = nj = 1024*8, nk = 16	ikj	None	None	Best Performance (GFLOPS): 0.80 1.55 3.05 5.73 7.13 9.14 10.19 12.26 21.72
ni = nj = 1024*8, nk = 16	ikj	k, 2	None	Best Performance (GFLOPS): 1.48 2.92 5.78 10.96 13.38 18.43 21.66 22.00 40.40
ni = nj = 1024*8, nk = 16	ikj	k, 4	None	Best Performance (GFLOPS): 2.24 4.42 8.72 16.80 20.99 28.95 35.60 38.46 68.93
ni = nj = 1024*8, nk = 16	ikj	k, 8	None	Best Performance (GFLOPS): 3.50 6.86 13.57 26.04 32.07 46.00 47.98 61.72 78.99
ni = nj = 1024*8, nk = 16	ikj	k, 16	None	Best Performance (GFLOPS): 1.96 3.90 7.69 15.13 19.30 28.22 37.51 23.27 47.69
ni = nj = 1024*8, nk = 16	ikj	K,2	Ikj, bs = 4	Best Performance (GFLOPS): 1.38 2.76 5.48 10.94 13.66 20.37 26.67 36.32 37.34
ni = nj = 1024*8, nk = 16	ikj	K,4	Ikj, bs = 4	Best Performance (GFLOPS): 1.45 2.89 5.75 11.51 14.35 21.44 28.35

				22.30 40.69
ni = nj = 1024*8, nk = 16	ikj	K,8	Ikj, bs = 4	Best Performance (GFLOPS): 1.04 2.08 4.15 8.30 10.36 15.47 20.48 18.35 27.49
ni = nj = 1024*8, nk = 16	ikj	K, 2	Ikj, bs = 8	Best Performance (GFLOPS): 2.52 5.03 9.99 19.94 24.64 36.22 25.34 32.01 59.53
ni = nj = 1024*8, nk = 16	ikj	K, 4	Ikj, bs = 8	Best Performance (GFLOPS): 2.86 5.71 11.40 22.74 28.09 40.99 53.29 36.06 63.95
ni = nj = 1024*8, nk = 16	ikj	K, 8	Ikj, bs = 8	Best Performance (GFLOPS): 2.87 5.71 11.43 22.72 28.15 40.98 36.56 38.18 64.58
ni = nj = 1024*8, nk = 16	ikj	None	Ij, bs = 4	Best Performance (GFLOPS): 1.38 2.74 5.47 10.94 13.58 20.30 26.94 20.29 40.36
ni = nj = 1024*8, nk = 16	ikj	K,2	Ij, bs = 4	Best Performance (GFLOPS): 1.81 3.61 7.24 14.43 18.03 26.93 35.50 26.87 50.22
ni = nj = 1024*8, nk = 16	ikj	K,4	Ij, bs = 4	Best Performance (GFLOPS): 1.99 3.96 7.90 15.79 19.69 29.28 38.83 29.37 57.04
Ni = 1024, nj = 1024*8, nk = 16	ikj	K, 8	Ij, bs = 4	Best Performance (GFLOPS): 2.06 4.10 8.16 16.14 20.08 29.93 39.79 30.32 59.99
ni = nj = 1024*8, nk = 16	ikj	None	Ij, bs = 8	Best Performance (GFLOPS): 2.17 4.34 8.65 17.28 21.47 31.96 23.95

				29.02 57.45
ni = nj = 1024*8, nk = 16	ikj	K,2	Ij, bs = 8	Best Performance (GFLOPS): 2.69 5.37 10.73 21.35 26.49 39.36 51.93 38.14 68.89
ni = nj = 1024*8, nk = 16	ikj	K,4	Ij, bs = 8	Best Performance (GFLOPS): 2.99 5.96 11.84 23.65 29.56 43.74 39.08 42.49 70.04
Ni = 1024, nj = 1024*8, nk = 16	ikj	K, 8	Ij, bs = 8	Best Performance (GFLOPS): 3.18 6.33 12.61 25.08 31.35 46.25 59.53 45.11 72.72
ni = nj = 1024*8, nk = 16	ikj	None	Ij, bs = 16	Best Performance (GFLOPS): 2.55 4.93 9.44 18.11 22.78 34.14 43.18 34.26 60.83
ni = nj = 1024*8, nk = 16	ikj	K,2	Ij, bs = 16	Best Performance (GFLOPS): 3.48 6.92 13.78 27.49 33.80 49.46 40.92 46.71 67.27
ni = nj = 1024*8, nk = 16	ikj	K,4	Ij, bs = 16	Best Performance (GFLOPS): 3.85 7.67 15.34 30.49 37.36 55.25 39.72 53.21 67.86
Ni = 1024,*8 nj = 1024*8, nk = 16	ikj	K, 8	Ij, bs = 16	Best Performance (GFLOPS): 3.98 7.93 15.72 31.27 38.45 54.34 59.23 69.42 69.08
ni = nj = 4096, nk = 64	ikj	None	None	Best Performance (GFLOPS): 0.91 1.79 3.54 6.90 8.42 12.55 16.55 14.47 20.30
ni = nj = 4096, nk = 64	ikj	k, 2	None	Best Performance (GFLOPS): 1.76 3.51 6.89 13.55 16.59 24.01 31.60

				23.86 42.01
ni = nj = 4096, nk = 64	ikj	k, 4	None	Best Performance (GFLOPS): 2.21 4.39 8.69 17.13 21.29 31.38 41.04 32.48 54.15
ni = nj = 4096, nk = 64	ikj	k, 8	None	Best Performance (GFLOPS): 2.30 4.52 8.84 16.30 18.94 27.16 33.85 36.45 56.49
ni = nj = 4096, nk = 64	ikj	k, 16	None	Best Performance (GFLOPS): 1.70 3.37 6.67 13.08 16.43 24.63 32.20 24.85 44.89
ni = nj = 4096, nk = 64	ikj	K,2	Ikj, bs = 4	Best Performance (GFLOPS): 1.23 2.44 4.85 9.66 11.98 17.72 23.44 18.22 36.38
ni = nj = 4096, nk = 64	ikj	K,4	Ikj, bs = 4	Best Performance (GFLOPS): 1.31 2.60 5.18 10.33 12.81 18.93 25.13 19.13 37.21
ni = nj = 4096, nk = 64	ikj	K,8	Ikj, bs = 4	Best Performance (GFLOPS): 0.97 1.92 3.83 7.65 9.50 14.17 18.75 17.29 26.92
ni = nj = 4096, nk = 64	ikj	K, 2	Ikj, bs = 8	Best Performance (GFLOPS): 2.38 4.74 9.48 18.88 23.24 34.38 45.95 31.99 60.92
ni = nj = 4096, nk = 64	ikj	K, 4	Ikj, bs = 8	Best Performance (GFLOPS): 2.79 5.55 11.08 22.08 27.09 40.22 28.63 35.58 68.25
ni = nj = 4096, nk = 64	ikj	K, 8	Ikj, bs = 8	Best Performance (GFLOPS): 2.72 5.41 10.81 21.50 26.44 39.22 52.42

				47.90 71.44
ni = nj = 4096, nk = 64	ikj	K, 2	Ikj, bs = 16	Best Performance (GFLOPS): 3.42 6.79 13.50 26.87 33.01 47.79 65.62 47.96 91.37
ni = nj = 4096, nk = 64	ikj	K, 4	Ikj, bs = 16	Best Performance (GFLOPS): 3.77 7.52 14.92 29.80 36.59 52.81 72.63 51.35 98.81
ni = nj = 4096, nk = 64	ikj	K, 8	Ikj, bs = 16	Best Performance (GFLOPS): 3.85 7.68 15.27 30.38 37.37 53.91 40.50 53.01 99.84
ni = nj = 4096, nk = 64	ikj	None	Ij, bs = 4	Best Performance (GFLOPS): 1.11 2.13 4.29 8.20 10.33 15.66 21.10 19.34 29.46
ni = nj = 4096, nk = 64	ikj	K,2	Ij, bs = 4	Best Performance (GFLOPS): 1.41 2.73 5.29 10.61 13.28 19.99 26.47 35.17 38.31
ni = nj = 4096, nk = 64	ikj	K,4	Ij, bs = 4	Best Performance (GFLOPS): 1.68 3.28 6.32 12.54 15.02 22.37 29.54 21.88 42.16
ni = nj = 4096, nk = 64	ikj	K, 8	Ij, bs = 4	Best Performance (GFLOPS): 1.77 3.44 6.46 13.02 16.16 24.27 31.51 26.35 50.80
ni = nj = 4096, nk = 64	ikj	None	Ij, bs = 8	Best Performance (GFLOPS): 1.74 3.42 6.76 13.35 16.90 25.05 32.75 25.14 47.76
ni = nj = 4096, nk = 64	ikj	K,2	Ij, bs = 8	Best Performance (GFLOPS): 2.08 4.10 8.15 16.07 19.94 30.21 38.93

				27.15 53.14
ni = nj = 4096, nk = 64	ikj	K,4	Ij, bs = 8	Best Performance (GFLOPS): 2.44 4.82 9.53 18.88 23.38 34.68 46.08 42.89 68.19
ni = nj = 4096, nk = 64	ikj	K, 8	Ij, bs = 8	Best Performance (GFLOPS): 2.72 5.42 10.57 21.06 25.83 37.99 51.13 42.74 79.79
ni = nj = 4096, nk = 64	ikj	None	Ij, bs = 16	Best Performance (GFLOPS): 2.26 4.49 8.79 17.36 21.68 31.32 42.12 30.53 57.15
ni = nj = 4096, nk = 64	ikj	K,2	Ij, bs = 16	Best Performance (GFLOPS): 2.80 5.54 10.70 21.19 26.03 38.75 52.23 47.41 75.09
ni = nj = 4096, nk = 64	ikj	K,4	Ij, bs = 16	Best Performance (GFLOPS): 3.26 6.46 12.82 25.23 30.86 45.26 60.26 58.19 93.67
ni = nj = 4096, nk = 64	ikj	K, 8	Ij, bs = 16	Best Performance (GFLOPS): 3.75 7.52 15.06 29.70 36.30 52.93 42.74 56.62 107.99
ni = nj = 2048, nk = 256	ikj	k, 2	None	Best Performance (GFLOPS): 1.78 3.50 6.94 13.61 16.77 24.70 31.11 21.16 40.81
ni = nj = 2048, nk = 256	ikj	k, 4	None	Best Performance (GFLOPS): 2.11 4.13 8.12 15.88 19.67 29.25 38.02 27.13 50.28
ni = nj = 2048, nk = 256	ikj	k, 8	None	Best Performance (GFLOPS): 2.21 4.36 8.58 16.81 20.81 30.86 40.22

				30.90 59.10
ni = nj = 2048, nk = 256	ikj	k, 16	None	Best Performance (GFLOPS): 1.44 2.84 5.63 11.19 14.03 20.78 27.47 21.70 37.74
ni = nj = 2048, nk = 256	ikj	K,2	Ikj, bs = 4	Best Performance (GFLOPS): 1.22 2.41 4.79 9.57 11.77 17.67 23.45 19.53 35.14
ni = nj = 2048, nk = 256	ikj	K,4	Ikj, bs = 4	Best Performance (GFLOPS): 1.28 2.53 4.99 9.95 12.28 18.18 24.31 18.25 35.08
ni = nj = 2048, nk = 256	ikj	K,8	Ikj, bs = 4	Best Performance (GFLOPS): 0.96 1.91 3.80 7.60 9.40 13.90 18.66 17.07 26.73
ni = nj = 2048, nk = 256	ikj	K, 2	Ikj, bs = 8	Best Performance (GFLOPS): 2.38 4.74 9.46 18.79 23.13 33.38 45.72 33.47 63.77
ni = nj = 2048, nk = 256	ikj	K, 4	Ikj, bs = 8	Best Performance (GFLOPS): 2.79 5.55 11.10 22.10 27.19 39.21 53.89 38.16 72.48
ni = nj = 2048, nk = 256	ikj	K, 8	Ikj, bs = 8	Best Performance (GFLOPS): 2.74 5.45 10.80 21.55 26.51 38.27 28.91 35.43 69.57
ni = nj = 2048, nk = 256	ikj	None	Ikj, bs = 16	Best Performance (GFLOPS): 2.51 5.00 9.99 19.95 22.73 31.76 39.64 32.71 44.40
ni = nj = 2048, nk = 256	ikj	K, 2	Ikj, bs = `16	Best Performance (GFLOPS): 3.53 7.05 13.98 27.80 34.29 49.23 63.44

				57.75 79.05
ni = nj = 2048, nk = 256	ikj	K, 4	Ikj, bs = `16	Best Performance (GFLOPS): 3.92 7.84 15.64 31.03 38.39 55.22 52.16 47.12 87.77
ni = nj = 2048, nk = 256	ikj	K, 8	Ikj, bs = 16	Best Performance (GFLOPS): 4.05 8.09 16.08 32.10 39.46 56.94 73.23 101.09 90.08
ni = nj = 2048, nk = 256	ikj	None	Ij, bs = 4	Best Performance (GFLOPS): 1.11 2.15 4.25 8.45 10.46 15.51 20.63 18.92 27.80
ni = nj = 2048, nk = 256	ikj	K,2	Ij, bs = 4	Best Performance (GFLOPS): 1.41 2.74 5.43 10.71 13.21 19.58 26.08 19.28 36.56
ni = nj = 2048, nk = 256	ikj	K,4	Ij, bs = 4	Best Performance (GFLOPS): 1.66 3.24 6.43 12.70 15.75 23.10 30.71 23.26 39.98
ni = nj = 2048, nk = 256	ikj	K, 8	Ij, bs = 4	Best Performance (GFLOPS): 1.75 3.45 6.79 13.31 16.18 23.61 31.26 26.20 49.24
ni = nj = 2048, nk = 256	ikj	None	Ij, bs = 8	Best Performance (GFLOPS): 1.67 3.25 6.39 12.76 15.69 22.70 30.91 24.43 43.09
ni = nj = 2048, nk = 256	ikj	K,2	Ij, bs = 8	Best Performance (GFLOPS): 2.02 3.97 7.81 15.50 19.16 27.74 37.97 27.07 48.69
ni = nj = 2048, nk = 256	ikj	K,4	Ij, bs = 8	Best Performance (GFLOPS): 2.22 4.41 8.73 17.33 21.46 30.84 42.37

				33.81 64.43
ni = nj = 2048, nk = 256	ikj	K, 8	Ij, bs = 8	Best Performance (GFLOPS): 2.73 5.32 10.53 20.89 25.59 36.47 49.89 40.11 75.33
ni = nj = 2048, nk = 256	ikj	None	Ij, bs = 16	Best Performance (GFLOPS): 2.22 4.38 8.66 17.26 21.26 30.71 39.39 30.39 50.79
ni = nj = 2048, nk = 256	ikj	K,2	Ij, bs = 16	Best Performance (GFLOPS): 2.56 5.07 10.07 20.01 24.64 35.40 36.21 35.89 68.18
ni = nj = 2048, nk = 256	ikj	K,4	Ij, bs = 16	Best Performance (GFLOPS): 3.17 6.31 12.26 24.28 29.94 43.21 55.29 75.91 83.25
ni = nj = 2048, nk = 256	ikj	K, 8	Ij, bs = 16	Best Performance (GFLOPS): 3.74 7.39 14.71 28.73 35.06 50.31 46.95 51.65 89.67
Ni = nj = 2048, nk = 1024	ikj	None	None	Best Performance (GFLOPS): 0.72 1.43 2.85 5.59 6.91 10.32 13.57 17.70 11.03
Ni = nj = 2048, nk = 1024	ikj	k, 2	None	Best Performance (GFLOPS): 1.38 2.76 5.43 10.72 13.35 19.83 26.42 34.95 24.05
Ni = nj = 2048, nk = 1024	ikj	k, 4	None	Best Performance (GFLOPS): 1.70 3.40 6.76 13.42 16.67 24.66 32.53 43.25 39.48
Ni = nj = 2048, nk = 1024	ikj	k, 8	None	Best Performance (GFLOPS): 1.85 3.71 7.31 14.58 18.05 26.87 35.64

				28.73 49.33
Ni = nj = 2048, nk = 1024	ikj	k, 16	None	Best Performance (GFLOPS): 1.26 2.52 5.01 9.99 12.47 18.44 24.34 21.67 34.97
Ni = nj = 2048, nk = 1024	ikj	K,2	Ikj, bs = 4	Best Performance (GFLOPS): 1.17 2.33 4.63 9.22 11.32 16.79 22.54 30.90 33.68
Ni = nj = 2048, nk = 1024	ikj	K,4	Ikj, bs = 4	Best Performance (GFLOPS): 1.23 2.46 4.91 9.79 12.05 17.91 23.99 32.86 36.06
Ni = nj = 2048, nk = 1024	ikj	K,8	Ikj, bs = 4	Best Performance (GFLOPS): 0.92 1.85 3.69 7.35 9.08 13.63 18.33 24.70 25.42
Ni = nj = 2048, nk = 1024	ikj	K, 2	Ikj, bs = 8	Best Performance (GFLOPS): 2.39 4.76 9.49 19.01 23.38 33.78 46.54 42.03 61.39
Ni = nj = 2048, nk = 1024	ikj	K, 4	Ikj, bs = 8	Best Performance (GFLOPS): 2.69 5.34 10.69 21.35 26.22 37.86 52.17 38.33 68.51
Ni = nj = 2048, nk = 1024	ikj	K, 8	Ikj, bs = 8	Best Performance (GFLOPS): 2.61 5.22 10.45 20.82 25.60 37.15 50.97 45.44 67.85
Ni = nj = 2048, nk = 1024	ikj	None	Ikj, bs = 16	Best Performance (GFLOPS): 2.51 5.00 9.99 19.95 22.73 31.76 39.64 32.71 44.40
Ni = nj = 2048, nk = 1024	ikj	K, 2	Ikj, bs = 16	Best Performance (GFLOPS): 2.65 5.29 10.50 20.87 25.62 36.83 47.39

				43.60 63.07
Ni = nj = 2048, nk = 1024	ikj	K, 4	Ikj, bs = 16	Best Performance (GFLOPS): 2.84 5.66 11.29 22.46 27.55 39.80 51.27 40.61 71.84
Ni = nj = 2048, nk = 1024	ikj	K, 8	Ikj, bs = 16	Best Performance (GFLOPS): 3.01 6.02 12.00 23.87 29.24 41.97 53.79 44.27 77.32
Ni = nj = 2048, nk = 1024	ikj	None	Ij, bs = 4	Best Performance (GFLOPS): 1.02 2.03 4.02 7.99 9.85 14.52 18.98 22.39 24.38
Ni = nj = 2048, nk = 1024	ikj	K,2	Ij, bs = 4	Best Performance (GFLOPS): 1.36 2.74 5.41 9.98 13.16 19.28 25.14 21.07 34.11
Ni = nj = 2048, nk = 1024	ikj	K,4	Ij, bs = 4	Best Performance (GFLOPS): 1.62 3.14 6.28 12.15 15.18 22.75 28.40 27.13 36.89
Ni = nj = 2048, nk = 1024	ikj	K, 8	Ij, bs = 4	Best Performance (GFLOPS): 1.72 3.43 6.85 13.10 16.20 23.68 31.66 26.41 44.22
Ni = nj = 2048, nk = 1024	ikj	K,2	Ij, bs = 8	Best Performance (GFLOPS): 1.92 3.82 7.59 15.01 18.47 26.73 36.64 38.74 42.92
Ni = nj = 2048, nk = 1024	ikj	K,4	Ij, bs = 8	Best Performance (GFLOPS): 2.15 4.31 8.60 16.98 20.90 30.01 40.69 34.70 57.63
Ni = nj = 2048, nk = 1024	ikj	K, 8	Ij, bs = 8	Best Performance (GFLOPS): 2.55 5.10 10.08 20.05 24.58 35.70 47.93

				43.17 64.17
Ni = nj = 2048, nk = 1024	ikj	None	Ij, bs = 16	Best Performance (GFLOPS): 1.93 3.83 7.68 15.35 18.87 27.31 34.87 29.43 39.77
Ni = nj = 2048, nk = 1024	ikj	K,2	Ij, bs = 16	Best Performance (GFLOPS): 2.30 4.59 9.03 17.83 22.30 32.00 40.90 41.44 50.57
Ni = nj = 2048, nk = 1024	ikj	K,4	Ij, bs = 16	Best Performance (GFLOPS): 2.71 5.40 10.50 21.30 26.08 37.21 46.47 34.86 59.32
Ni = nj = 2048, nk = 1024	ikj	K, 8	Ij, bs = 16	Best Performance (GFLOPS): 2.98 5.92 11.85 23.29 28.73 41.50 52.97 50.99 69.41
ni = nj = 1024, nk = 1024	ikj	None	None	Best Performance (GFLOPS): 0.89 1.76 3.51 6.88 8.55 12.44 16.36 15.68 21.21
ni = nj = 1024, nk = 1024	ikj	j, 2	None	Best Performance (GFLOPS): 0.89 1.76 3.50 6.87 8.48 12.52 16.29 11.28 20.91
ni = nj = 1024, nk = 1024	ikj	I, 2	None	Best Performance (GFLOPS): 1.68 3.33 6.63 12.98 16.19 23.91 30.43 26.73 35.43
ni = nj = 1024, nk = 1024	ikj	k,2	None	Best Performance (GFLOPS): 1.70 3.36 6.69 13.11 16.28 24.09 30.86 20.31 39.21
ni = nj = 1024, nk = 1024	ikj	k,4	None	Best Performance (GFLOPS): 1.99 3.95 7.88 15.56 19.26 28.44 37.23

				26.72 47.48
ni = nj = 1024, nk = 1024	ikj	k,8	None	Best Performance (GFLOPS): 2.15 4.30 8.58 17.07 21.17 31.38 41.02 29.35 58.23
ni = nj = 1024, nk = 1024	ikj	K, 2	Ijk tiled with bs = 4	Best Performance (GFLOPS): 1.24 2.47 4.93 9.86 12.13 17.52 24.16 18.19 34.81
ni = nj = 1024, nk = 1024	ikj	K,4	Ijk tiled with bs = 4	Best Performance (GFLOPS): 1.29 2.58 5.14 10.28 12.64 18.40 25.24 19.85 35.80
ni = nj = 1024, nk = 1024	ikj	K,8	Ijk tiled with bs = 4	Best Performance (GFLOPS): 0.94 1.88 3.74 7.48 9.20 13.29 18.35 13.32 25.71
ni = nj = 1024, nk = 1024	ikj	K,2	Ijk tiled with bs = 8	Best Performance (GFLOPS): 2.47 4.93 9.86 19.67 24.20 34.81 44.61 34.71 61.81
ni = nj = 1024, nk = 1024	ikj	K,4	Ijk tiled with bs = 8	Best Performance (GFLOPS): 2.75 5.50 10.97 21.91 26.88 38.84 49.91 69.76 66.91
ni = nj = 1024, nk = 1024	ikj	K,8	Ijk tiled with bs = 8	Best Performance (GFLOPS): 2.71 5.42 10.81 21.57 26.49 38.07 32.36 35.04 64.50
ni = nj = 1024, nk = 1024	ikj	None	Ijk tiled with bs = 16	Best Performance (GFLOPS): 2.51 5.00 9.99 19.95 22.73 31.76 39.64 32.71 44.40
ni = nj = 1024, nk = 1024	ikj	K,2	Ijk tiled with bs = 16	Best Performance (GFLOPS): 2.88 5.74 11.49 22.92 26.18 36.66 45.84

				33.18 55.32
ni = nj = 1024, nk = 1024	ikj	K, 4	Ijk tiled with bs = 16	Best Performance (GFLOPS): 2.97 5.94 11.87 23.68 27.03 37.79 47.28 62.71 61.42
ni = nj = 1024, nk = 1024	ikj	K, 8	Ijk tiled with bs = 16	Best Performance (GFLOPS): 3.13 6.26 12.49 24.97 28.35 39.93 49.95 40.30 65.10
ni = nj = 1024, nk = 1024	ikj	None	Ijk tiled with bs = 32	Best Performance (GFLOPS): 1.82 3.64 7.25 14.48 14.56 19.41 28.86 21.77 26.27
ni = nj = 1024, nk = 1024	ikj	None	Ij tiled with bs = 4	Best Performance (GFLOPS): 1.02 2.06 4.07 8.10 9.95 14.51 19.88 17.97 24.17
ni = nj = 1024, nk = 1024	ikj	K, 2	Ij tiled with bs = 4	Best Performance (GFLOPS): 1.37 2.77 5.46 10.88 13.34 19.37 26.46 23.80 33.47
ni = nj = 1024, nk = 1024	ikj	K, 4	Ij tiled with bs = 4	Best Performance (GFLOPS): 1.62 3.23 6.37 12.66 15.52 22.83 30.96 26.95 37.57
ni = nj = 1024, nk = 1024	ikj	K, 8	Ij tiled with bs = 4	Best Performance (GFLOPS): 1.76 3.50 6.95 13.57 16.69 24.05 32.81 24.54 44.17
ni = nj = 1024, nk = 1024	ikj	none	Ij tiled with bs = 8	Best Performance (GFLOPS): 1.54 3.09 6.18 12.33 15.20 21.86 28.01 20.20 37.87
ni = nj = 1024, nk = 1024	ikj	K, 2	Ij tiled with bs = 8	Best Performance (GFLOPS): 1.91 3.84 7.66 15.31 18.71 26.94 34.62

				22.81 43.79
ni = nj = 1024, nk = 1024	ikj	K, 4	Ij tiled with bs = 8	Best Performance (GFLOPS): 2.15 4.30 8.59 17.17 21.04 30.39 38.76 39.09 57.81
ni = nj = 1024, nk = 1024	ikj	K, 8	Ij tiled with bs = 8	Best Performance (GFLOPS): 2.60 5.18 10.29 20.23 24.66 35.55 45.63 34.58 64.68
ni = nj = 1024, nk = 1024	ikj	none	Ij tiled with bs = 16	Best Performance (GFLOPS): 1.97 3.93 7.84 15.62 17.63 24.72 30.84 26.83 38.70
ni = nj = 1024, nk = 1024	ikj	K, 2	Ij tiled with bs = 16	Best Performance (GFLOPS): 2.32 4.64 9.27 18.50 21.14 29.54 36.86 35.57 48.25
ni = nj = 1024, nk = 1024	ikj	K, 4	Ij tiled with bs = 16	Best Performance (GFLOPS): 2.72 5.44 10.85 21.63 24.70 34.54 30.60 33.50 57.73
ni = nj = 1024, nk = 1024	ikj	K, 8	Ij tiled with bs = 16	Best Performance (GFLOPS): 3.03 6.05 12.00 24.00 27.43 38.47 47.77 38.51 61.29
ni = nj = 1024, nk = 1024	ikj	None	Ik tiled with bs = 4	Best Performance (GFLOPS): 0.89 1.77 3.52 6.90 8.35 12.12 16.43 15.27 20.12
ni = nj = 1024, nk = 1024	ikj	None	Ik tiled with bs = 8	Best Performance (GFLOPS): 0.91 1.79 3.43 6.79 8.33 11.73 15.03 13.74 19.84
ni = nj = 1024, nk = 1024	ikj	K, 2	Ik tiled with bs = 16	Best Performance (GFLOPS): 1.75 3.43 6.67 12.94 14.97 20.58 24.98

				33.08 31.66
ni = nj = 1024, nk = 1024	ikj	K, 2	Ik tiled with bs = 4	Best Performance (GFLOPS): 1.71 3.43 6.78 13.17 15.96 23.04 30.95 22.03 36.66
ni = nj = 1024, nk = 1024	ikj	K, 2	Ik tiled with bs = 8	Best Performance (GFLOPS): 1.74 3.42 6.52 12.91 15.93 22.59 28.65 20.92 31.40
ni = nj = 1024, nk = 1024	ikj	none	I tiled with bs = 4	Best Performance (GFLOPS): 0.89 1.77 3.52 6.88 8.44 12.13 16.08 15.36 19.82
ni = nj = 1024, nk = 1024	ikj	none	I tiled with bs = 8	Best Performance (GFLOPS): 0.92 1.79 3.45 6.84 8.38 11.90 15.11 11.46 19.93
ni = nj = 1024, nk = 1024	ikj	none	I tiled with bs = 16	Best Performance (GFLOPS): 0.92 1.79 3.47 6.90 7.82 10.76 13.26 13.10 17.11
ni = nj = 1024, nk = 1024	ikj	none	j tiled with bs = 4	Best Performance (GFLOPS): 0.52 1.04 2.05 4.05 5.04 7.51 9.85 13.02 11.18
ni = nj = 1024, nk = 1024	ikj	none	j tiled with bs = 8	Best Performance (GFLOPS): 0.76 1.48 2.92 5.74 7.01 10.53 13.61 9.50 16.18
ni = nj = 1024, nk = 1024	ikj	none	j tiled with bs = 16	Best Performance (GFLOPS): 0.88 1.72 3.40 6.59 8.08 12.05 15.44 10.56 18.11
ni = nj = 1024, nk = 1024	ikj	none	k tiled with bs = 4	Best Performance (GFLOPS): 0.88 1.77 3.47 6.88 8.45 12.50 16.12 11.35

				20.85
ni = nj = 1024, nk = 1024	ikj	none	k tiled with bs = 8	Best Performance (GFLOPS): 0.91 1.79 3.48 6.91 8.51 12.39 16.14 11.41 20.69
ni = nj = 1024, nk = 1024	ikj	none	k tiled with bs = 16	Best Performance (GFLOPS): 0.91 1.79 3.54 6.71 8.47 12.38 16.14 11.36 21.10
ni = nj = 1024, nk = 1024	ikj	none	kj tiled with bs = 4	Best Performance (GFLOPS): 0.78 1.57 3.12 6.21 7.70 11.48 15.17 14.62 24.22
ni = nj = 1024, nk = 1024	ikj	none	kj tiled with bs = 8	Best Performance (GFLOPS): 1.44 2.84 5.64 11.16 13.75 20.64 27.35 22.42 41.31
ni = nj = 1024, nk = 1024	ikj	none	kj tiled with bs = 16	Best Performance (GFLOPS): 1.76 3.49 6.89 13.67 16.98 25.29 33.41 22.67 43.93