Jessica Salguero, UIN: 937005006
CSCE 735, Summer 2025

Major Project: Parallelizing Cholesky Factorization Algorithm on a GPU

Question 1:

1. (75 points) Develop a parallel implementation of the Cholesky factorization algorithm that runs on the GPU. Your code should correctly compute R using multiple GPU threads and should demonstrate speedup over the single-thread code. 10 points out of 75 are reserved for implementations that use more than one multiprocessor on the GPU to compute the factorization (recall that one needs to create more than one block of threads in the grid in order to use more than one multiprocessor).

Successfully implemented multi-thread GPU Cholesky factorization!

```
n = 128, parallel
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 128, GPU execution time:   2.8027 ms
```

Jessica Salguero, UIN: 937005006
CSCE 735, Summer 2025

Question 2

2. (25 points) Develop a report that describes the parallel performance of your code and its dependence on the grid structure used for kernel invocation. This will require you to conduct a variety of experiments with varying matrix sizes and grid structures to understand the behavior of your program on the GPU. Discuss any design choices you made to improve the parallel performance of the code and how they relate to actual observations. Include any insights you obtained while working on this project. Lastly, include a brief description of how to compile and execute the code.

Single Thread Implementation Results

```
n = 128, single thread
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 128, GPU execution time:  85.7579 ms

n = 256, single thread
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 256, GPU execution time: 598.4850 ms

n = 512, single thread
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 512, GPU execution time: 4607.2690 ms
```

```
n = 1024, single thread
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 1024, GPU execution time: 37129.6523 ms

n = 2048, single thread
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 2048, GPU execution time: 322810.7188 ms

n = 4096, single thread
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 7.5.
 ... multiprocessor count  = 40
 ... max threads per multiprocessor = 1024
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1590 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 4096, GPU execution time: 3343299.5000 ms
```

Parallel Multi-thread Implementation Results

```
n = 128, parallel
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 128, GPU execution time:   2.8027 ms
```

```
n = 256, parallel
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 256, GPU execution time:  19.6465 ms


n = 512, parallel
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 512, GPU execution time: 103.5960 ms


n = 1024, parallel
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 7.5.
 ... multiprocessor count  = 72
 ... max threads per multiprocessor = 1024
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1620 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 1024, GPU execution time: 310.1369 ms

n = 2048, parallel
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 7.5.
 ... multiprocessor count  = 72
 ... max threads per multiprocessor = 1024
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1620 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 2048, GPU execution time: 1111.9585 ms
```

Jessica Salguero, UIN: 937005006
CSCE 735, Summer 2025

```
n = 4096, parallel
------------------------------------------------------------
Number of GPU devices found = 1
[Device: 0] Compute Capability 8.0.
 ... multiprocessor count  = 108
 ... max threads per multiprocessor = 2048
 ... max threads per block = 1024
 ... max block dimension   = 1024, 1024, 64 (along x, y, z)
 ... max grid size         = 2147483647, 65535, 65535 (along x, y, z)
 ... warp size             = 32
 ... clock rate            = 1410 MHz
------------------------------------------------------------
+++  Matrix successfully factored
Matrix size: 4096, GPU execution time: 7789.8955 ms
```

Question 2 Table

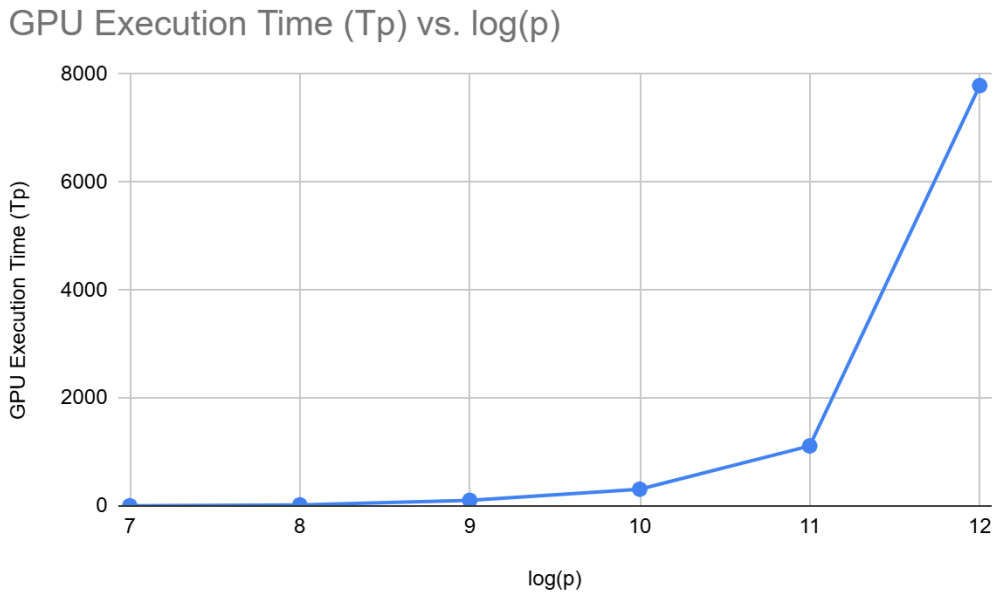
Threads (p)	log(p)	Single Thread Execution Time (T1)	GPU Execution Time (Tp)	SpeedUp (T1/Tp)	Efficiency (Sp/p)
128	7	85.7579	2.8027	30.59831591	0.239049343
256	8	598.485	19.6465	30.46267783	0.118994835 3
512	9	4607.269	103.596	44.47342561	0.086862159 4
1024	10	37129.6523	310.1369	119.7202019	0.116914259 7
2048	11	322810.7188	1111.9585	290.3082433	0.141752071 9
4096	12	3343299.5	7789.8955	429.1841271	0.104781281

Design Choices for GPU Parallelization

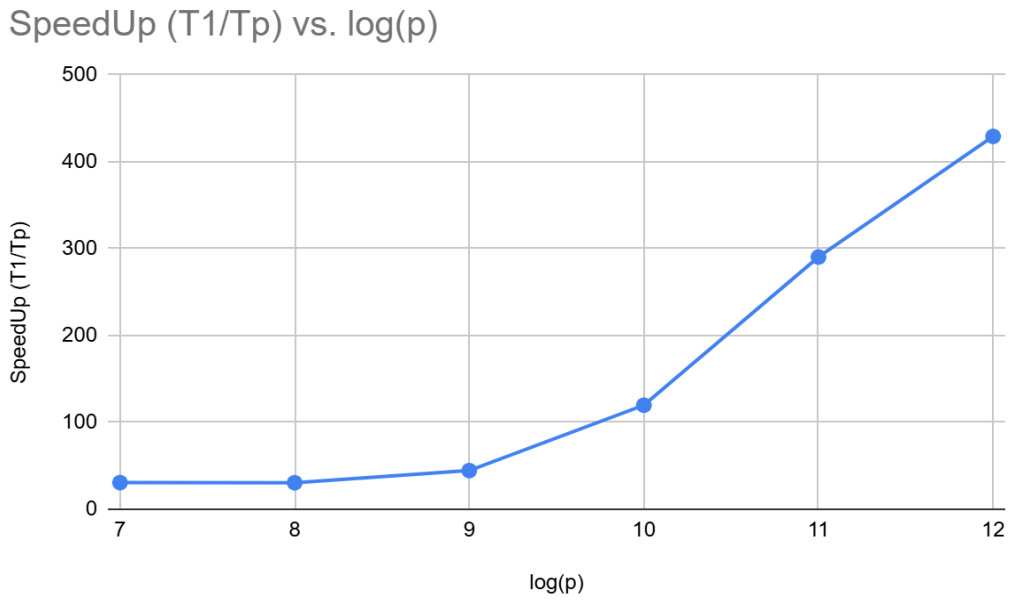
To parallelize Cholesky's factorization using GPUs and multiple threads, I had to create a new CUDA kernel device (parallel_cholesky). This device splits up the matrix into block sections and distributes them across multiple threads. This device can be split into a few main sections while looping over the matrix by column: computing the diagonal, dividing each element by the shared diagonal, updating the matrix, and a sanity check to make sure that everything below the diagonal is zero. One of the most important aspects to make sure that the calculations were correct was including __syncthreads() calls after each section; this ensured that all threads had finished and caught up before moving onto the following step and prevented race conditions where threads access incorrect/non-updated variables. Additionally, the total number of threads and their respective blocks were determined based on total matrix size. The matrix was split into block sizes of 256; this block size provided a good balance of overhead and performance improvement.

Jessica Salguero, UIN: 937005006
CSCE 735, Summer 2025

Performance Figures

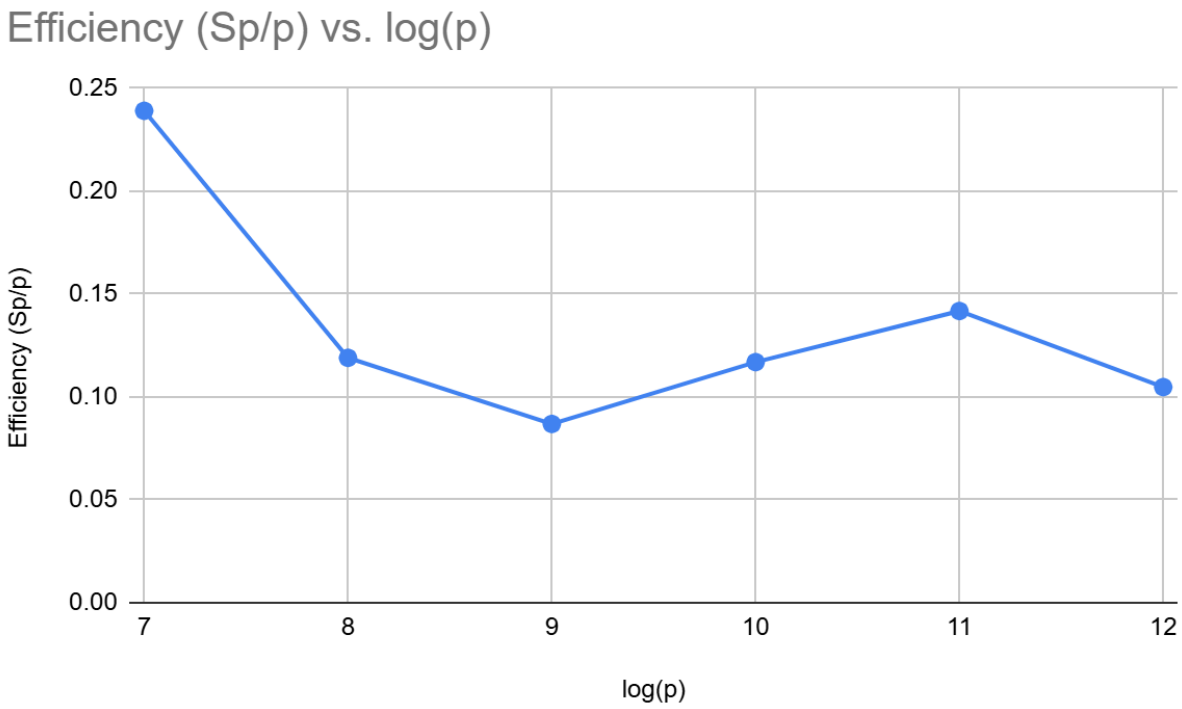


This figure looks at the trend of execution time as the matrix size and number of threads increased. As expected, we can see that the parallel implementation using multiple threads increases as matrix size and number of threads increases. However, there is a much larger increase in execution time at the log value 12 (equivalent to 4096). Since 4096 is the maximum matrix size, implementation is not optimized for this value, which explains the drastic increase.



This figure looks at the speedUp of this multithread implementation when compared to the original code using a single thread. Again, there is a clear trend that as the matrix size and number of threads increases, speedUp also increases. This suggests that within the tested implementation, additional threads resulted in improved performance.

Jessica Salguero, UIN: 937005006
CSCE 735, Summer 2025



Lastly, this figure examines efficiency as the number of threads and matrix size increases. As the number of threads and matrix size increases, efficiency tends to trend downward. This decrease in efficiency suggests that the additional threads for parallelization are able to improve performance times, but the execution will not be as efficient since there is increased overhead required for maintaining communication. Additionally, the use of __syncthreads() requires some early-finishing threads to idle while waiting for others to catch up, decreasing the overall efficiency but making sure that shared variables are correct.

Jessica Salguero, UIN: 937005006
CSCE 735, Summer 2025

Compile & Execute Code

When compiling and executing the code, there are a few important things that need to be done:

- You need to request a GPU node
- Load necessary modules for intel/2023a and CUDA/12.2
- Create executable: nvcc -o cholesky_gpu_parallel.exe cholesky_gpu_parallel_SalgueroJessica.cu
- When running the executable, you need to specify matrix size: ./cholesky_gpu_parallel.exe <n>

To compile and execute the code, I used slurm and sbatch scripts. This is what my sbatch script looked like:

```
#!/bin/bash
##ENVIRONMENT SETTINGS; CHANGE WITH CAUTION
#SBATCH --export=NONE                #Do not propagate environment
#SBATCH --get-user-env=L             #Replicate login environment
#
##NECESSARY JOB SPECIFICATIONS
#SBATCH --job-name=cholesky_gpu_parallel       #Set the job name to "JobExample2"
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --time=1:00:00               #Set the wall clock limit to 1hr
#SBATCH --nodes=1                    #Request 1 node
#SBATCH --ntasks=1
#SBATCH --mem=32G                     #Request 32GB per node
#SBATCH --output=output.major-project.%j      #Send stdout/err to "output.[jobID]"
#
##OPTIONAL JOB SPECIFICATIONS
##SBATCH --mail-type=ALL              #Send email on all job events
##SBATCH --mail-user=email_address    #Send all emails to email_address
#
##First Executable Line
#
module load intel/2023a CUDA/12.2
#
nvcc -o cholesky_gpu_parallel.exe cholesky_gpu_parallel_SalgueroJessica.cu
#
echo "n = 4096, parallel"
./cholesky_gpu_parallel.exe 4096
```