Toronto Metropolitan University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering
& Architectural Science

Course Title:	Advanced Computer Architecture
Course Number:	COE 818
Semester/Year (e.g.F2016)	W2023

Instructor:	Dr. Arghavan Asad

Assignment/Lab Number:	4
Assignment/Lab Title:	Using NVIDIA GPU

Submission Date:	March 13, 2023
Due Date:	March 13, 2023

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Youssef	Helana	500766171	1	H.Y.

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: https://www.torontomu.ca/content/dam/senate/policies/pol60.pdf

Lab 4: Using NVIDIA GPU

1. Objective:

We create a parallel matrix multiplication application in this lab. The matrix multiplication application is run on CUDA, and we discover that adding more threads improves speed.

2. Introduction:

NVIDIA created CUDA as a general-purpose parallel computing framework for use with its own GPUs. By utilising GPU capability, CUDA enables programmers to accelerate computationally demanding programmes. Several GPU APIs, like OpenCL, have been proposed. There are moreover other competitive GPUs from firms like AMD. Several application domains, including deep learning, are dominated by the CUDA and NVIDIA GPU combination [1].

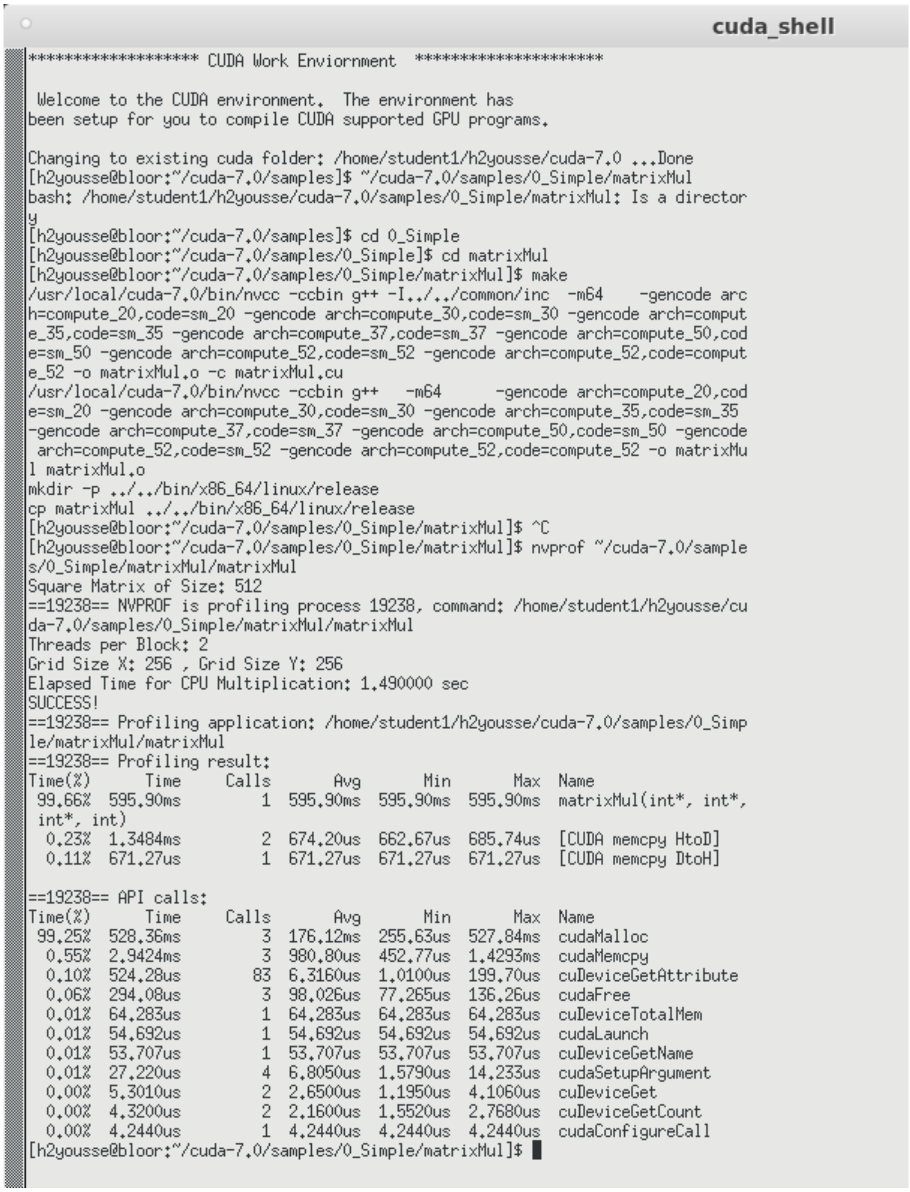
We create a parallel matrix multiplication application in this lab. The matrix multiplication application is run on CUDA, and we discover that adding more threads improves speed. We assess the application's performance when the number of threads per block is changed from 2 to 32 in multiples of 2. We use square matrices of sizes 512 x 512 and 1024 x 1024 to compare the performance of the matrix multiplication application in each case.

3. Background

Using Device Query to Determine the Optimal Threads Per Block

It is essential to examine the characteristics of the CUDA device in the system that is used to run the application to choose the best number of threads per block for our matrix multiplication application. We make use of the Device Query example found in the folder /cuda-7.0/samples/1 Utilities/deviceQuery.

The CUDA devices that are present in the system are listed by their attributes in this example. The device characteristics of the system that we utilize to run the matrix multiplication application are shown in Figure 1.



```
****************** CUDA Work Enviornment  ********************

 Welcome to the CUDA environment.  The environment has
been setup for you to compile CUDA supported GPU programs.

Changing to existing cuda folder: /home/student1/h2yousse/cuda-7.0 ...Done
[h2yousse@bloor:~/cuda-7.0/samples]$ ~/cuda-7.0/samples/0_Simple/matrixMul
bash: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul: Is a director
y
[h2yousse@bloor:~/cuda-7.0/samples]$ cd 0_Simple
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple]$ cd matrixMul
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arc
h=compute_20,code=sm_20 -gencode arch=compute_30,code=sm_30 -gencode arch=comput
e_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,cod
e=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=comput
e_52 -o matrixMul.o -c matrixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,cod
e=sm_20 -gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35
-gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode
 arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMu
l matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ ^C
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/sample
s/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 512
==19238== NVPROF is profiling process 19238, command: /home/student1/h2yousse/cu
da-7.0/samples/0_Simple/matrixMul/matrixMul
Threads per Block: 2
Grid Size X: 256 , Grid Size Y: 256
Elapsed Time for CPU Multiplication: 1.490000 sec
SUCCESS!
==19238== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simp
le/matrixMul/matrixMul
==19238== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.66%  595.90ms         1  595.90ms  595.90ms  595.90ms  matrixMul(int*, int*,
 int*, int)
  0.23%  1.3484ms         2  674.20us  662.67us  685.74us  [CUDA memcpy HtoD]
  0.11%  671.27us         1  671.27us  671.27us  671.27us  [CUDA memcpy DtoH]

==19238== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.25%  528.36ms         3  176.12ms  255.63us  527.84ms  cudaMalloc
  0.55%  2.9424ms         3  980.80us  452.77us  1.4293ms  cudaMemcpy
  0.10%  524.28us        83  6.3160us  1.0100us  199.70us  cuDeviceGetAttribute
  0.06%  294.08us         3  98.026us  77.265us  136.26us  cudaFree
  0.01%  64.283us         1  64.283us  64.283us  64.283us  cuDeviceTotalMem
  0.01%  54.692us         1  54.692us  54.692us  54.692us  cudaLaunch
  0.01%  53.707us         1  53.707us  53.707us  53.707us  cuDeviceGetName
  0.01%  27.220us         4  6.8050us  1.5790us  14.233us  cudaSetupArgument
  0.00%  5.3010us         2  2.6500us  1.1950us  4.1060us  cuDeviceGet
  0.00%  4.3200us         2  2.1600us  1.5520us  2.7680us  cuDeviceGetCount
  0.00%  4.2440us         1  4.2440us  4.2440us  4.2440us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ █
```

Figure 1: Properties of the CUDA Device in the System Used to Run the Matrix Multiplication Application

You can see that the matrix multiplication application is running on a CUDA device that uses 48 cores per multiprocessor in the device's characteristics. Also take note of the warp size, which is 32. We must thus make sure to utilise a number of threads per block that is a multiple of the warp size (32) and the number of cores per multiprocessor (48 cores per multiprocessor) when determining the ideal number of threads per block to run the matrix multiplication application [2].

As the number of threads per block is increased in the range of 2 to 16 threads per block, we anticipate the scalability of the matrix multiplication application to grow (in multiples of 2). We anticipate that the scalability of the matrix multiplication application would decline when the number of threads per block is increased from 16 to 32 threads per block since 32 threads per block is not a multiple of 48 cores per multiprocessor.

4. Methodology

We create a straightforward CUDA application for matrix multiplication. The matrix multiplication programme is executed one step at a time. The matrix multiplication application is then run in parallel with thread counts per block ranging from 2 to 32 in multiples of 2. We keep track of how long it takes to finish each matrix multiplication. Also, we keep track of how long it takes to perform matrix multiplications for each of the scenarios previously discussed using square matrices of sizes 512 x 512 and 1024 x 1024.

The appendix contains the code for the matrix multiplication application. In the results part of the appendix, Table I summarizes the results that were shown after the application was run.

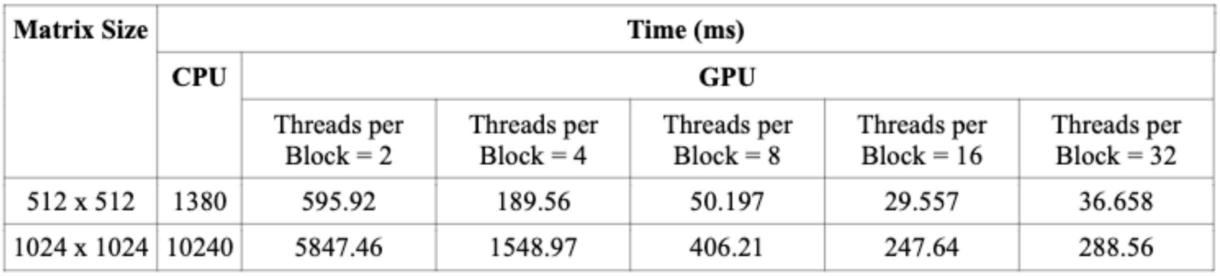
5. Results

A) Total Time Needed to Finish Matrix Multiplication

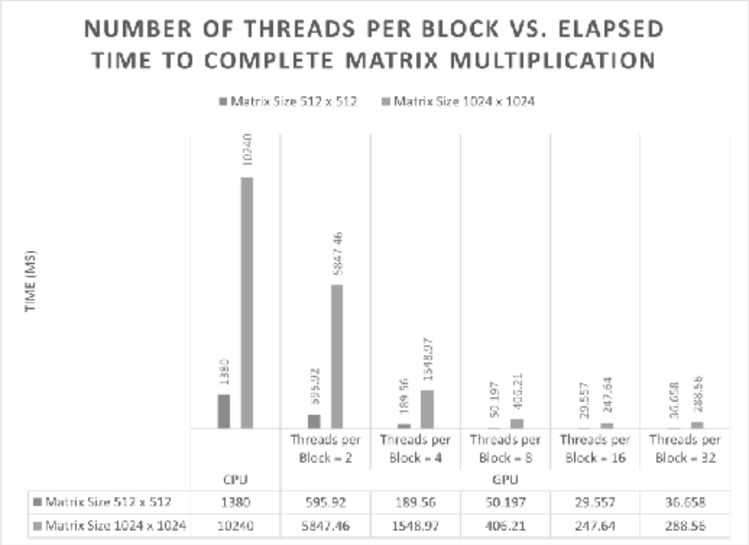
Figure 2 and Table I both show the amount of time needed to complete each of the matrix multiplications. The results show that a considerable time improvement was achieved by raising the number of threads per block from 2 threads per block to 16 threads per block in multiples of 2.

Moreover, going from 16 to 32 threads per block did not speed up the process. This is because 32 threads per block is not a multiple of the 48 cores that our multiprocessor has. Hence, 16 threads per block is the ideal number of threads for our matrix multiplication application.

Table I: Elapsed Time to Complete Matrix Multiplication



Matrix Size	**Time (ms)**					
	CPU	**GPU**				
		Threads per Block = 2	Threads per Block = 4	Threads per Block = 8	Threads per Block = 16	Threads per Block = 32
512 x 512	1380	595.92	189.56	50.197	29.557	36.658
1024 x 1024	10240	5847.46	1548.97	406.21	247.64	288.56



	CPU	Threads per Block = 2	Threads per Block = 4	Threads per Block = 8	Threads per Block = 16	Threads per Block = 32
		GPU				
■ Matrix Size 512 x 512	1380	595.92	189.56	50.197	29.557	36.658
■ Matrix Size 1024 x 1024	10240	5847.46	1548.97	406.21	247.64	288.56

Figure 2: Number of Threads per Block vs. Elapsed Time to Complete Matrix Multiplication

B) Scalability

The scalability is determined by comparing the amount of time needed to multiply a matrix sequentially on a CPU to the amount of time needed to multiply the same matrix in parallel on a GPU utilising x threads per block, as shown in (1):

$$Scalability = \frac{T_{seq}}{T_{parallel}(x\ Threads\ per\ Block)} \tag{1}$$

For the situations of square matrices of size 512 x 512 and square matrices of size 1024 x 1024, Table II shows the scalability of the matrix multiplication application when the number of threads per block is increased from 2 to 32 in multiples of 2. The statistics presented in Table II are illustrated in Figure 3.

The results in Table II and Figure 3 make it clear that the matrix multiplication application is more scalable as the number of threads per block rises, from 2 to 16 threads per block, in multiples of 2. It should be noted that scaling the matrix multiplication application increases the number of threads per block from 16 to 32 but limits its scalability. This is because 32 threads per block is not a multiple of the 48 cores that our multiprocessor has. Hence, 16 threads per block is the ideal number of threads for our matrix multiplication application.

Table II: Scalability of Matrix Multiplication Application

Matrix Size	Scalability					
	CPU	GPU				
		Threads per Block = 2	Threads per Block = 4	Threads per Block = 8	Threads per Block = 16	Threads per Block = 32
512 x 512	1	2.315747	7.280017	27.49168	46.68945	37.64526
1024 x 1024	1	1.751188	6.610845	25.20864	41.35035	35.48655

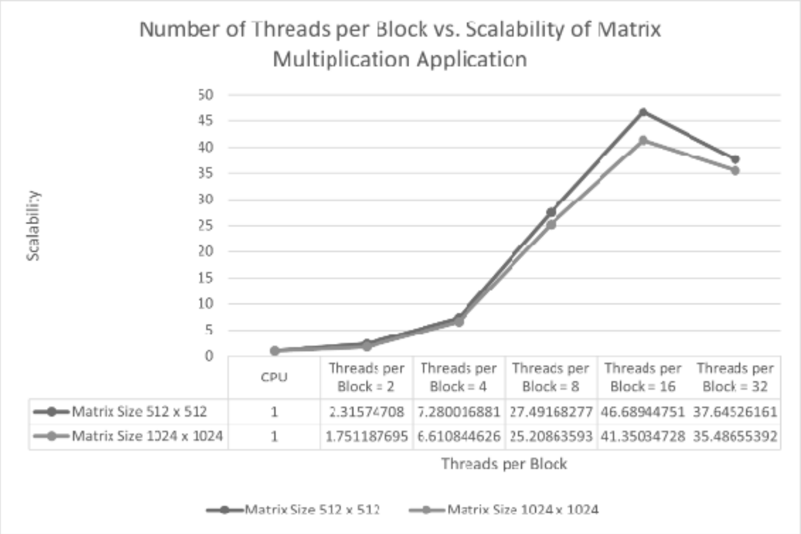


Figure 3: Number of Threads per Block vs. Scalability of Matrix Multiplication Application

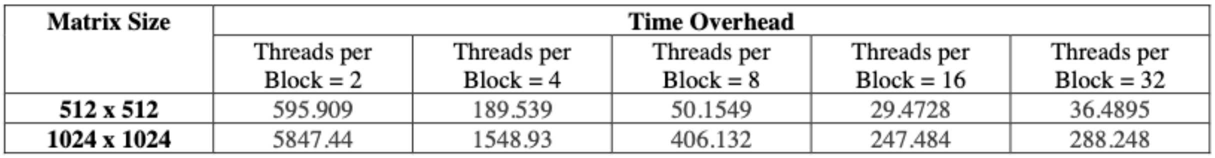
C) Time Overhead

The time overhead is calculated using (2):

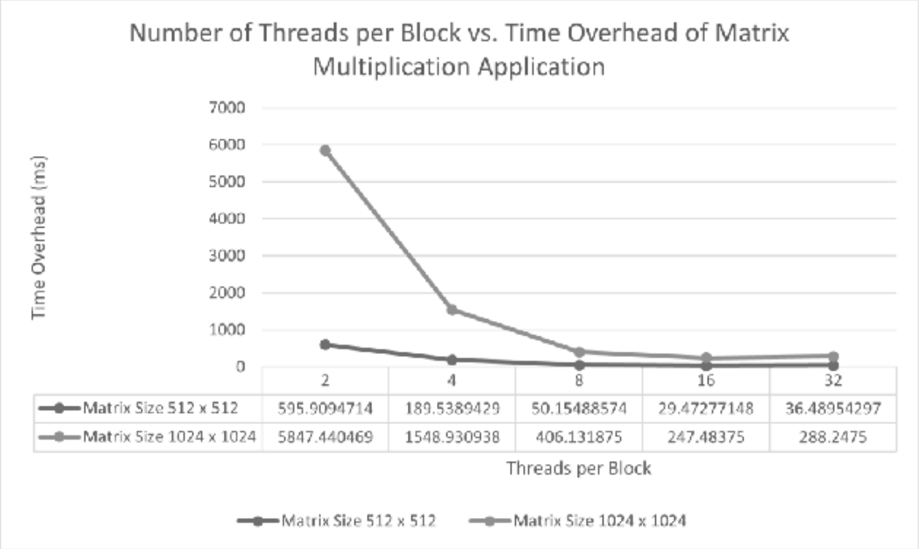
$$T_{overhead} = T_{parallel}(Threads\ per\ Block) - \frac{T_{seq}}{Total\ Number\ of\ Threads} \qquad (2)$$

As the number of threads per block increases from 2 to 16 threads per block, in multiples of 2, the matrix multiplication application becomes more scalable, as shown by the findings in Table II and Figure 3. It should be noted that scaling the matrix multiplication application results in a 16 to 32 thread increase per block but limits its potential to scale. This is because our multiprocessor only has 48 cores, although there are 32 threads per block. This means that the ideal number of threads for our matrix multiplication application is 16 threads per block.

Table III: Time Overhead of Matrix Multiplication Application



Matrix Size	Time Overhead				
	Threads per Block = 2	Threads per Block = 4	Threads per Block = 8	Threads per Block = 16	Threads per Block = 32
512 x 512	595.909	189.539	50.1549	29.4728	36.4895
1024 x 1024	5847.44	1548.93	406.132	247.484	288.248



	2	4	8	16	32
Matrix Size 512 x 512	595.9094714	189.5389429	50.15488574	29.47277148	36.48954297
Matrix Size 1024 x 1024	5847.440469	1548.930938	406.131875	247.48375	288.2475

Figure 4: Number of Threads per Block vs. Time Overhead of Matrix Multiplication Application

6. Conclusion

The statistics in Table II and Figure 3 clearly show that as the number of threads per block rises from 2 to 16, the scalability of the matrix multiplication application also increases.
It should be noted that scaling the matrix multiplication application increases the number of threads per block from 16 to 32 but limits its scalability. This is because 32 threads per block is not a multiple of the 48 cores that our multiprocessor has. Hence, 16 threads per block is the ideal number of threads for our matrix multiplication application.

The time overhead of the matrix multiplication application is greatest when 2 threads per block are used, as can be seen from the data shown in Table III and shown in Figure 4. When the number of threads per block is changed, in multiples of 2, from 2 to 16 threads per block, the matrix multiplication application's time overhead decreases. When 16 threads per block are used, the matrix multiplication application's time overhead is at its lowest. It should be noted that the time overhead grows as the number of threads per block rises from 16 to 32. This is because 32 threads per block is not a multiple of the 48 cores that our multiprocessor has.

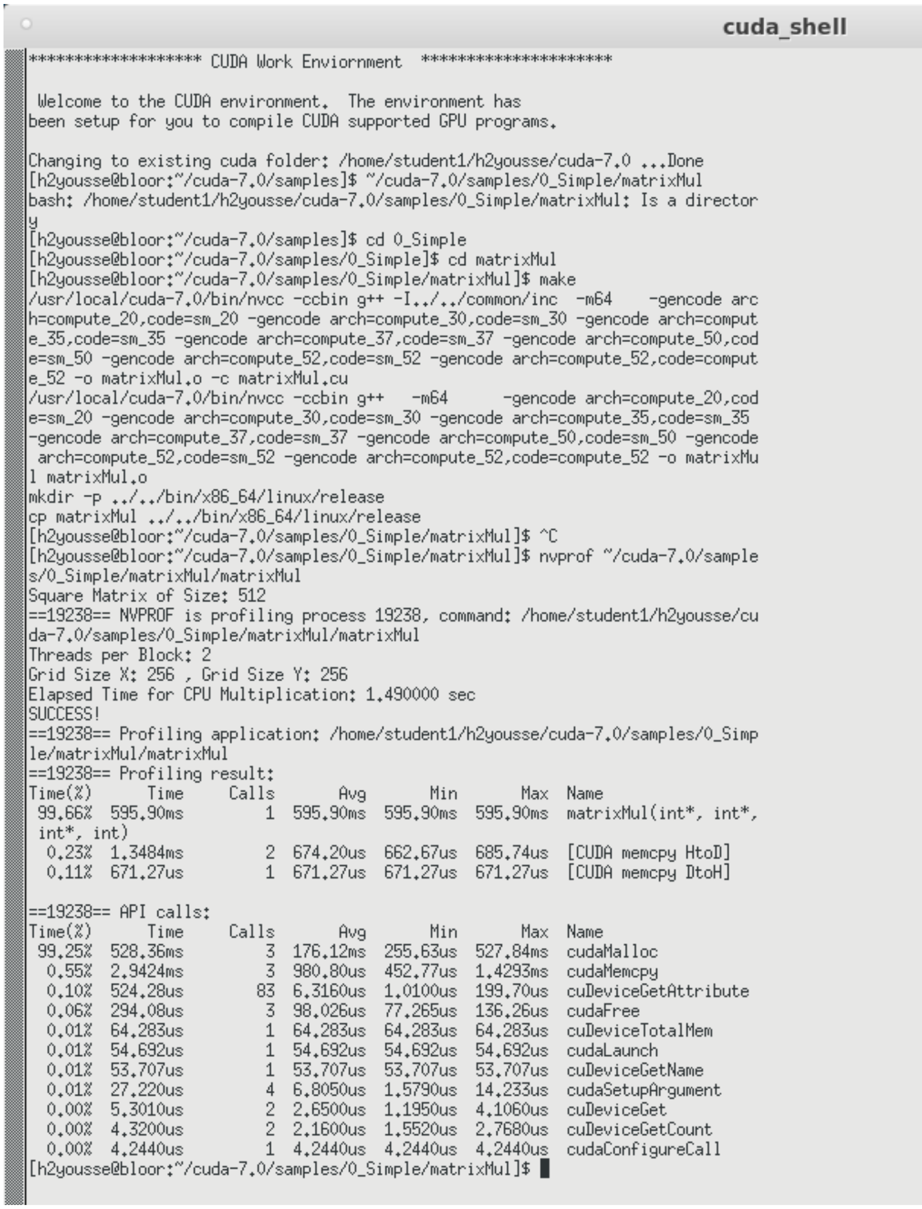
7. References

[1] M. Heller, “What is CUDA? Parallel Programming for GPUs,” InfoWorld, 2018. https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html#:~:text=CUDA is a parallel computing, parallelizable part of the computation.

[2] C. Frederick, “The optimal number of threads per block in CUDA programming,” Research Gate, 2017. https://www.researchgate.net/post/The_optimal_number_of_threads_per_block_in_CUDA_programming.

8. Appendix

A) Console Results



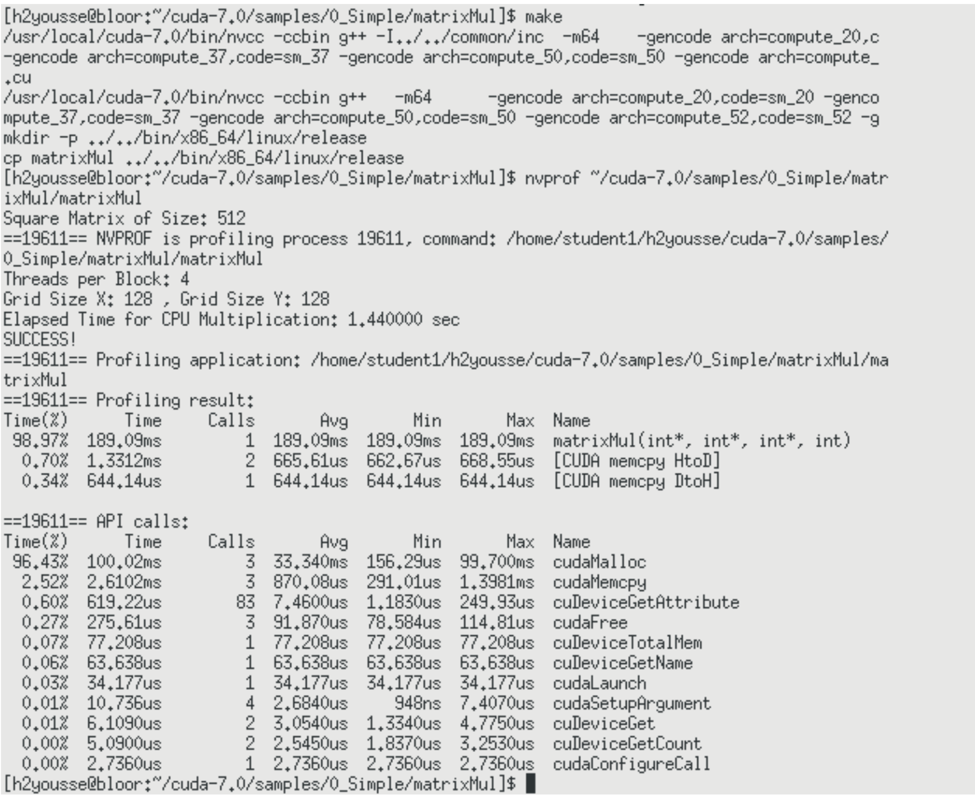
```
****************** CUDA Work Enviornment  ********************

 Welcome to the CUDA environment.  The environment has
been setup for you to compile CUDA supported GPU programs.

Changing to existing cuda folder: /home/student1/h2yousse/cuda-7.0 ...Done
[h2yousse@bloor:~/cuda-7.0/samples]$ ~/cuda-7.0/samples/0_Simple/matrixMul
bash: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul: Is a director
y
[h2yousse@bloor:~/cuda-7.0/samples]$ cd 0_Simple
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple]$ cd matrixMul
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arc
h=compute_20,code=sm_20 -gencode arch=compute_30,code=sm_30 -gencode arch=comput
e_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,cod
e=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=comput
e_52 -o matrixMul.o -c matrixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,cod
e=sm_20 -gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35
-gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode
 arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMu
l matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ ^C
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/sample
s/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 512
==19238== NVPROF is profiling process 19238, command: /home/student1/h2yousse/cu
da-7.0/samples/0_Simple/matrixMul/matrixMul
Threads per Block: 2
Grid Size X: 256 , Grid Size Y: 256
Elapsed Time for CPU Multiplication: 1.490000 sec
SUCCESS!
==19238== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simp
le/matrixMul/matrixMul
==19238== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.66%  595.90ms         1  595.90ms  595.90ms  595.90ms  matrixMul(int*, int*,
 int*, int)
  0.23%  1.3484ms         2  674.20us  662.67us  685.74us  [CUDA memcpy HtoD]
  0.11%  671.27us         1  671.27us  671.27us  671.27us  [CUDA memcpy DtoH]

==19238== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.25%  528.36ms         3  176.12ms  255.63us  527.84ms  cudaMalloc
  0.55%  2.9424ms         3  980.80us  452.77us  1.4293ms  cudaMemcpy
  0.10%  524.28us        83  6.3160us  1.0100us  199.70us  cuDeviceGetAttribute
  0.06%  294.08us         3  98.026us  77.265us  136.26us  cudaFree
  0.01%  64.283us         1  64.283us  64.283us  64.283us  cuDeviceTotalMem
  0.01%  54.692us         1  54.692us  54.692us  54.692us  cudaLaunch
  0.01%  53.707us         1  53.707us  53.707us  53.707us  cuDeviceGetName
  0.01%  27.220us         4  6.8050us  1.5790us  14.233us  cudaSetupArgument
  0.00%  5.3010us         2  2.6500us  1.1950us  4.1060us  cuDeviceGet
  0.00%  4.3200us         2  2.1600us  1.5520us  2.7680us  cuDeviceGetCount
  0.00%  4.2440us         1  4.2440us  4.2440us  4.2440us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ █
```

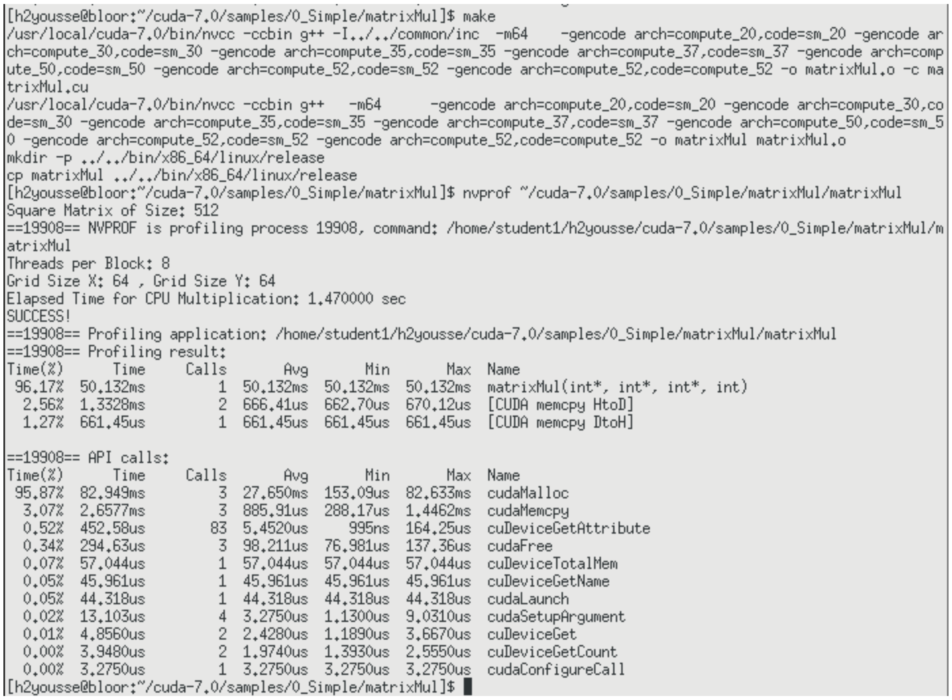
Figure 5: Console Results for Matrix Multiplication Application Using Matrices of Size 512 x 512 and 2 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,c
-gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode arch=compute_
.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -genco
mpute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -g
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matr
ixMul/matrixMul
Square Matrix of Size: 512
==19611== NVPROF is profiling process 19611, command: /home/student1/h2yousse/cuda-7.0/samples/
0_Simple/matrixMul/matrixMul
Threads per Block: 4
Grid Size X: 128 , Grid Size Y: 128
Elapsed Time for CPU Multiplication: 1.440000 sec
SUCCESS!
==19611== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/ma
trixMul
==19611== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.97%  189.09ms         1  189.09ms  189.09ms  189.09ms  matrixMul(int*, int*, int*, int)
  0.70%  1.3312ms         2  665.61us  662.67us  668.55us  [CUDA memcpy HtoD]
  0.34%  644.14us         1  644.14us  644.14us  644.14us  [CUDA memcpy DtoH]

==19611== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.43%  100.02ms         3  33.340ms  156.29us  99.700ms  cudaMalloc
  2.52%  2.6102ms         3  870.08us  291.01us  1.3981ms  cudaMemcpy
  0.60%  619.22us        83  7.4600us  1.1830us  249.93us  cuDeviceGetAttribute
  0.27%  275.61us         3  91.870us  78.584us  114.81us  cudaFree
  0.07%  77.208us         1  77.208us  77.208us  77.208us  cuDeviceTotalMem
  0.06%  63.638us         1  63.638us  63.638us  63.638us  cuDeviceGetName
  0.03%  34.177us         1  34.177us  34.177us  34.177us  cudaLaunch
  0.01%  10.736us         4  2.6840us     948ns  7.4070us  cudaSetupArgument
  0.01%  6.1090us         2  3.0540us  1.3340us  4.7750us  cuDeviceGet
  0.00%  5.0900us         2  2.5450us  1.8370us  3.2530us  cuDeviceGetCount
  0.00%  2.7360us         1  2.7360us  2.7360us  2.7360us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ █
```

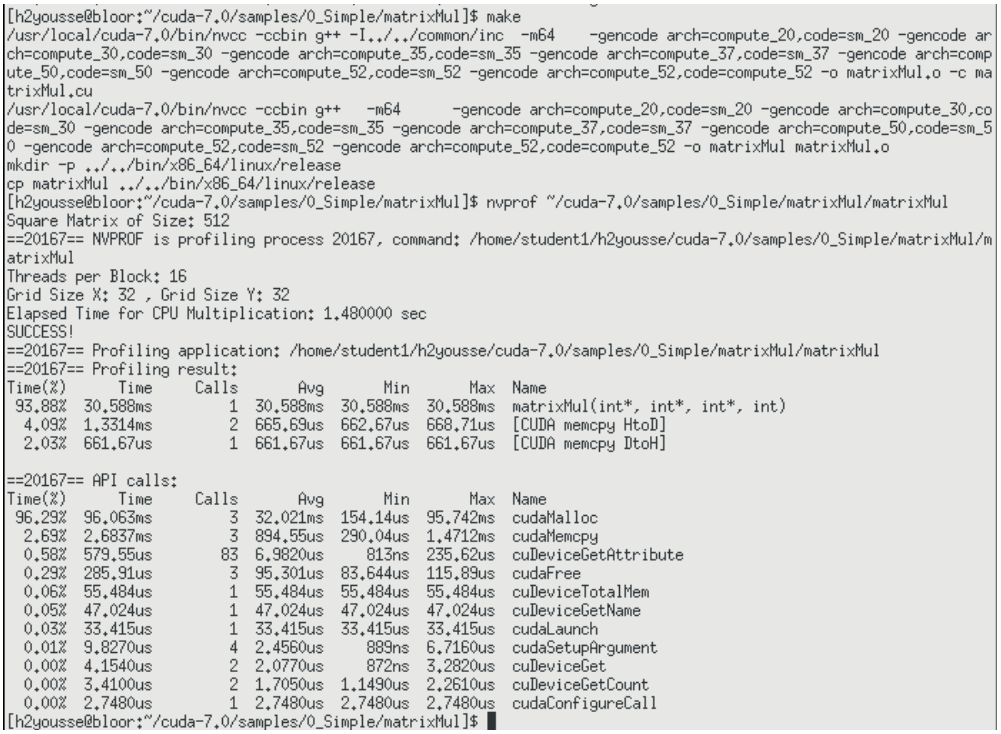
Figure 6: Console Results for Matrix Multiplication Application Using Matrices of Size 512 x 512 and 4 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,code=sm_20 -gencode ar
ch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=comp
ute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul.o -c ma
trixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,co
de=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_5
0 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 512
==19908== NVPROF is profiling process 19908, command: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/m
atrixMul
Threads per Block: 8
Grid Size X: 64 , Grid Size Y: 64
Elapsed Time for CPU Multiplication: 1.470000 sec
SUCCESS!
==19908== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
==19908== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.17%  50.132ms         1  50.132ms  50.132ms  50.132ms  matrixMul(int*, int*, int*, int)
  2.56%  1.3328ms         2  666.41us  662.70us  670.12us  [CUDA memcpy HtoD]
  1.27%  661.45us         1  661.45us  661.45us  661.45us  [CUDA memcpy DtoH]

==19908== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 95.87%  82.949ms         3  27.650ms  153.09us  82.633ms  cudaMalloc
  3.07%  2.6577ms         3  885.91us  288.17us  1.4462ms  cudaMemcpy
  0.52%  452.58us        83  5.4520us     995ns  164.25us  cuDeviceGetAttribute
  0.34%  294.63us         3  98.211us  76.981us  137.36us  cudaFree
  0.07%  57.044us         1  57.044us  57.044us  57.044us  cuDeviceTotalMem
  0.05%  45.961us         1  45.961us  45.961us  45.961us  cuDeviceGetName
  0.05%  44.318us         1  44.318us  44.318us  44.318us  cudaLaunch
  0.02%  13.103us         4  3.2750us  1.1300us  9.0310us  cudaSetupArgument
  0.01%  4.8560us         2  2.4280us  1.1890us  3.6670us  cuDeviceGet
  0.00%  3.9480us         2  1.9740us  1.3930us  2.5550us  cuDeviceGetCount
  0.00%  3.2750us         1  3.2750us  3.2750us  3.2750us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ █
```

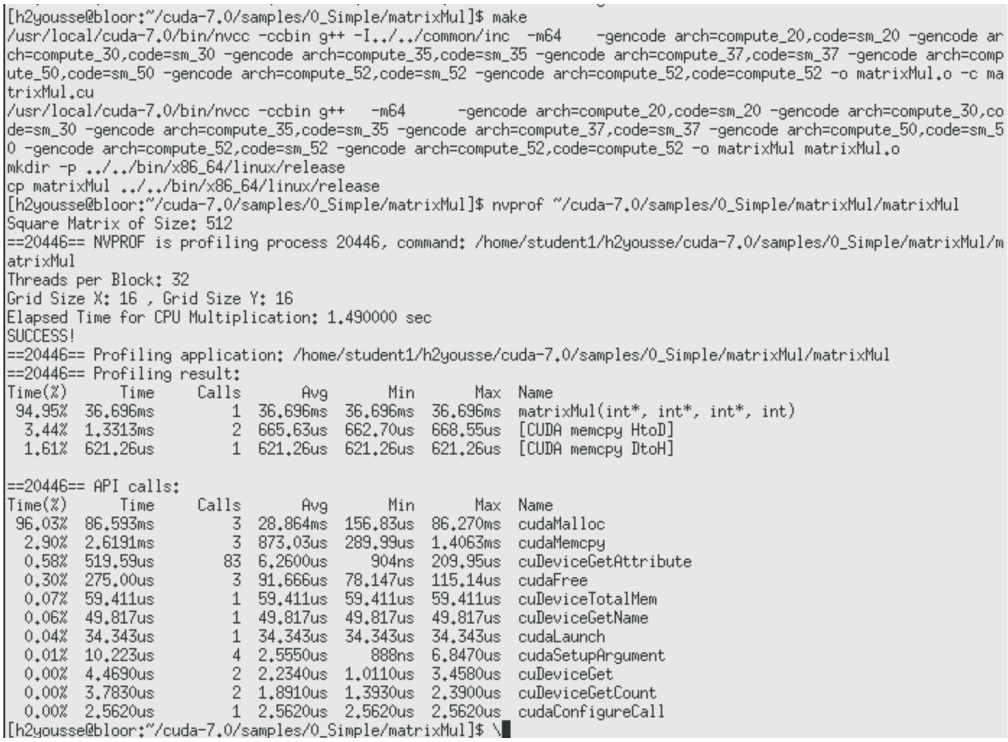
Figure 7: Console Results for Matrix Multiplication Application Using Matrices of Size 512 x 512 and 8 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,code=sm_20 -gencode ar
ch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=comp
ute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul.o -c ma
trixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,co
de=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_5
0 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 512
==20167== NVPROF is profiling process 20167, command: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/m
atrixMul
Threads per Block: 16
Grid Size X: 32 , Grid Size Y: 32
Elapsed Time for CPU Multiplication: 1.480000 sec
SUCCESS!
==20167== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
==20167== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 93.88%  30.588ms         1  30.588ms  30.588ms  30.588ms  matrixMul(int*, int*, int*, int)
  4.09%  1.3314ms         2  665.69us  662.67us  668.71us  [CUDA memcpy HtoD]
  2.03%  661.67us         1  661.67us  661.67us  661.67us  [CUDA memcpy DtoH]

==20167== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.29%  96.063ms         3  32.021ms  154.14us  95.742ms  cudaMalloc
  2.69%  2.6837ms         3  894.55us  290.04us  1.4712ms  cudaMemcpy
  0.58%  579.55us        83  6.9820us     813ns  235.62us  cuDeviceGetAttribute
  0.29%  285.91us         3  95.301us  83.644us  115.89us  cudaFree
  0.06%  55.484us         1  55.484us  55.484us  55.484us  cuDeviceTotalMem
  0.05%  47.024us         1  47.024us  47.024us  47.024us  cuDeviceGetName
  0.03%  33.415us         1  33.415us  33.415us  33.415us  cudaLaunch
  0.01%  9.8270us         4  2.4560us     889ns  6.7160us  cudaSetupArgument
  0.00%  4.1540us         2  2.0770us     872ns  3.2820us  cuDeviceGet
  0.00%  3.4100us         2  1.7050us  1.1490us  2.2610us  cuDeviceGetCount
  0.00%  2.7480us         1  2.7480us  2.7480us  2.7480us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ █
```

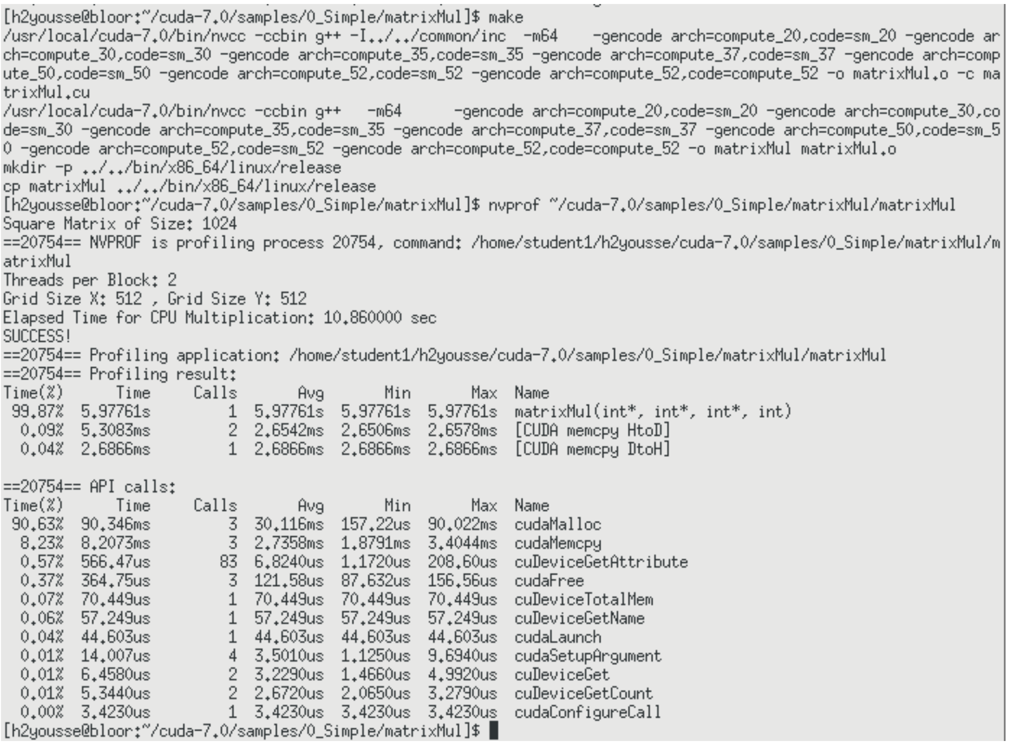
Figure 8: Console Results for Matrix Multiplication Application Using Matrices of Size 512 x 512 and 16 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,code=sm_20 -gencode ar
ch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=comp
ute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul.o -c ma
trixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,co
de=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_5
0 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 512
==20446== NVPROF is profiling process 20446, command: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/m
atrixMul
Threads per Block: 32
Grid Size X: 16 , Grid Size Y: 16
Elapsed Time for CPU Multiplication: 1.490000 sec
SUCCESS!
==20446== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
==20446== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 94.95%  36.696ms         1  36.696ms  36.696ms  36.696ms  matrixMul(int*, int*, int*, int)
  3.44%  1.3313ms         2  665.63us  662.70us  668.55us  [CUDA memcpy HtoD]
  1.61%  621.26us         1  621.26us  621.26us  621.26us  [CUDA memcpy DtoH]

==20446== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.03%  86.593ms         3  28.864ms  156.83us  86.270ms  cudaMalloc
  2.90%  2.6191ms         3  873.03us  289.99us  1.4063ms  cudaMemcpy
  0.58%  519.59us        83  6.2600us     904ns  209.95us  cuDeviceGetAttribute
  0.30%  275.00us         3  91.666us  78.147us  115.14us  cudaFree
  0.07%  59.411us         1  59.411us  59.411us  59.411us  cuDeviceTotalMem
  0.06%  49.817us         1  49.817us  49.817us  49.817us  cuDeviceGetName
  0.04%  34.343us         1  34.343us  34.343us  34.343us  cudaLaunch
  0.01%  10.223us         4  2.5550us     888ns  6.8470us  cudaSetupArgument
  0.00%  4.4690us         2  2.2340us  1.0110us  3.4580us  cuDeviceGet
  0.00%  3.7830us         2  1.8910us  1.3930us  2.3900us  cuDeviceGetCount
  0.00%  2.5620us         1  2.5620us  2.5620us  2.5620us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ \█
```

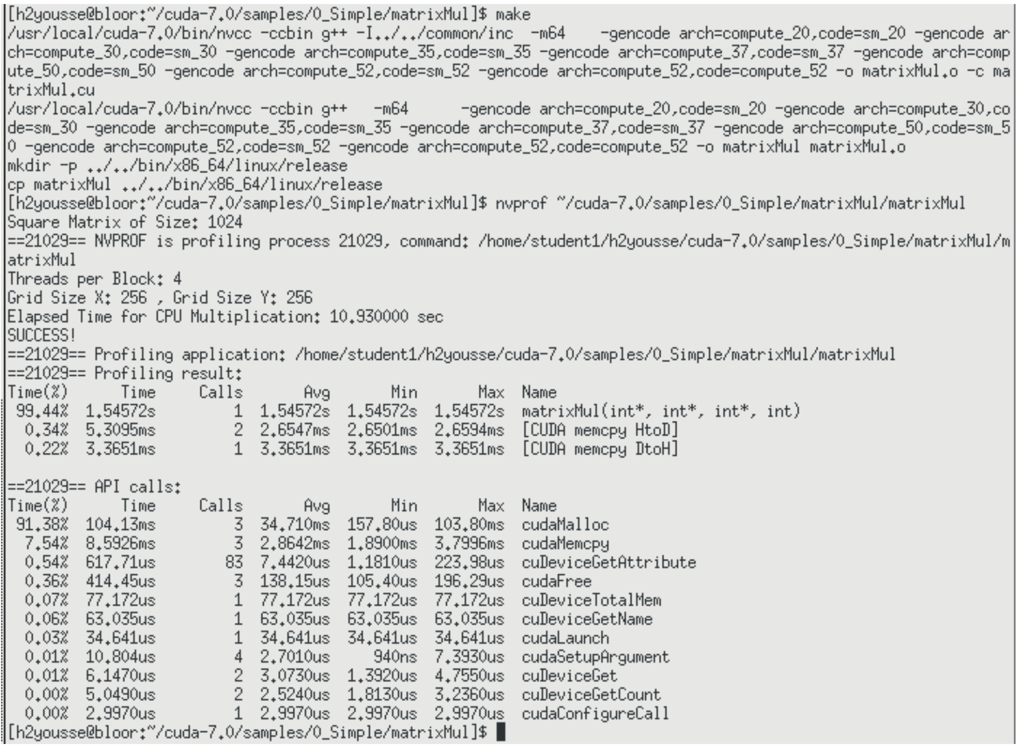
Figure 9: Console Results for Matrix Multiplication Application Using Matrices of Size 512 x 512 and 32 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,code=sm_20 -gencode ar
ch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=comp
ute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul.o -c ma
trixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,co
de=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_5
0 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 1024
==20754== NVPROF is profiling process 20754, command: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/m
atrixMul
Threads per Block: 2
Grid Size X: 512 , Grid Size Y: 512
Elapsed Time for CPU Multiplication: 10.860000 sec
SUCCESS!
==20754== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
==20754== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.87%  5.97761s         1  5.97761s  5.97761s  5.97761s  matrixMul(int*, int*, int*, int)
  0.09%  5.3083ms         2  2.6542ms  2.6506ms  2.6578ms  [CUDA memcpy HtoD]
  0.04%  2.6866ms         1  2.6866ms  2.6866ms  2.6866ms  [CUDA memcpy DtoH]

==20754== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 90.63%  90.346ms         3  30.116ms  157.22us  90.022ms  cudaMalloc
  8.23%  8.2073ms         3  2.7358ms  1.8791ms  3.4044ms  cudaMemcpy
  0.57%  566.47us        83  6.8240us  1.1720us  208.60us  cuDeviceGetAttribute
  0.37%  364.75us         3  121.58us  87.632us  156.56us  cudaFree
  0.07%  70.449us         1  70.449us  70.449us  70.449us  cuDeviceTotalMem
  0.06%  57.249us         1  57.249us  57.249us  57.249us  cuDeviceGetName
  0.04%  44.603us         1  44.603us  44.603us  44.603us  cudaLaunch
  0.01%  14.007us         4  3.5010us  1.1250us  9.6940us  cudaSetupArgument
  0.01%  6.4580us         2  3.2290us  1.4660us  4.9920us  cuDeviceGet
  0.01%  5.3440us         2  2.6720us  2.0650us  3.2790us  cuDeviceGetCount
  0.00%  3.4230us         1  3.4230us  3.4230us  3.4230us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$
```

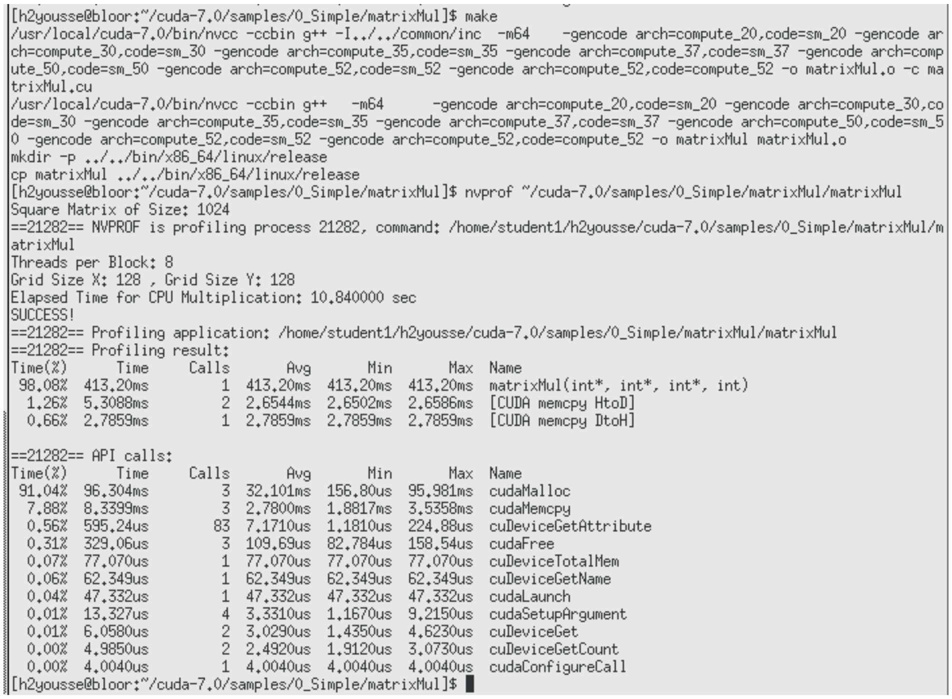
Figure 10: Console Results for Matrix Multiplication Application Using Matrices of Size 1024 x 1024 and 2 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,code=sm_20 -gencode ar
ch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=comp
ute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul.o -c ma
trixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,co
de=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_5
0 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 1024
==21029== NVPROF is profiling process 21029, command: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/m
atrixMul
Threads per Block: 4
Grid Size X: 256 , Grid Size Y: 256
Elapsed Time for CPU Multiplication: 10.930000 sec
SUCCESS!
==21029== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
==21029== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.44%  1.54572s         1  1.54572s  1.54572s  1.54572s  matrixMul(int*, int*, int*, int)
  0.34%  5.3095ms         2  2.6547ms  2.6501ms  2.6594ms  [CUDA memcpy HtoD]
  0.22%  3.3651ms         1  3.3651ms  3.3651ms  3.3651ms  [CUDA memcpy DtoH]

==21029== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 91.38%  104.13ms         3  34.710ms  157.80us  103.80ms  cudaMalloc
  7.54%  8.5926ms         3  2.8642ms  1.8900ms  3.7996ms  cudaMemcpy
  0.54%  617.71us        83  7.4420us  1.1810us  223.98us  cuDeviceGetAttribute
  0.36%  414.45us         3  138.15us  105.40us  196.29us  cudaFree
  0.07%  77.172us         1  77.172us  77.172us  77.172us  cuDeviceTotalMem
  0.06%  63.035us         1  63.035us  63.035us  63.035us  cuDeviceGetName
  0.03%  34.641us         1  34.641us  34.641us  34.641us  cudaLaunch
  0.01%  10.804us         4  2.7010us     940ns  7.3930us  cudaSetupArgument
  0.01%  6.1470us         2  3.0730us  1.3920us  4.7550us  cuDeviceGet
  0.00%  5.0490us         2  2.5240us  1.8130us  3.2360us  cuDeviceGetCount
  0.00%  2.9970us         1  2.9970us  2.9970us  2.9970us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$
```

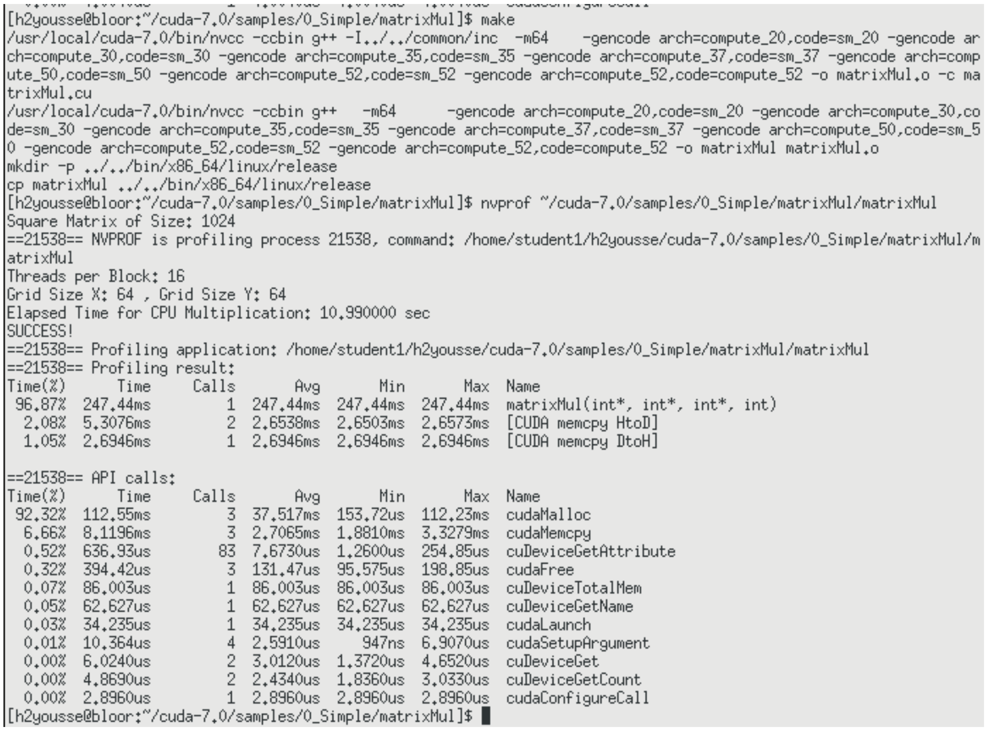
Figure 11: Console Results for Matrix Multiplication Application Using Matrices of Size 1024 x 1024 and 4 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,code=sm_20 -gencode ar
ch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=comp
ute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul.o -c ma
trixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,co
de=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_5
0 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 1024
==21282== NVPROF is profiling process 21282, command: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/m
atrixMul
Threads per Block: 8
Grid Size X: 128 , Grid Size Y: 128
Elapsed Time for CPU Multiplication: 10.840000 sec
SUCCESS!
==21282== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
==21282== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.08%  413.20ms         1  413.20ms  413.20ms  413.20ms  matrixMul(int*, int*, int*, int)
  1.26%  5.3088ms         2  2.6544ms  2.6502ms  2.6586ms  [CUDA memcpy HtoD]
  0.66%  2.7859ms         1  2.7859ms  2.7859ms  2.7859ms  [CUDA memcpy DtoH]

==21282== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 91.04%  96.304ms         3  32.101ms  156.80us  95.981ms  cudaMalloc
  7.88%  8.3399ms         3  2.7800ms  1.8817ms  3.5358ms  cudaMemcpy
  0.56%  595.24us        83  7.1710us  1.1810us  224.88us  cuDeviceGetAttribute
  0.31%  329.06us         3  109.69us  82.784us  158.54us  cudaFree
  0.07%  77.070us         1  77.070us  77.070us  77.070us  cuDeviceTotalMem
  0.06%  62.349us         1  62.349us  62.349us  62.349us  cuDeviceGetName
  0.04%  47.332us         1  47.332us  47.332us  47.332us  cudaLaunch
  0.01%  13.327us         4  3.3310us  1.1670us  9.2150us  cudaSetupArgument
  0.01%  6.0580us         2  3.0290us  1.4350us  4.6230us  cuDeviceGet
  0.00%  4.9850us         2  2.4920us  1.9120us  3.0730us  cuDeviceGetCount
  0.00%  4.0040us         1  4.0040us  4.0040us  4.0040us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ █
```

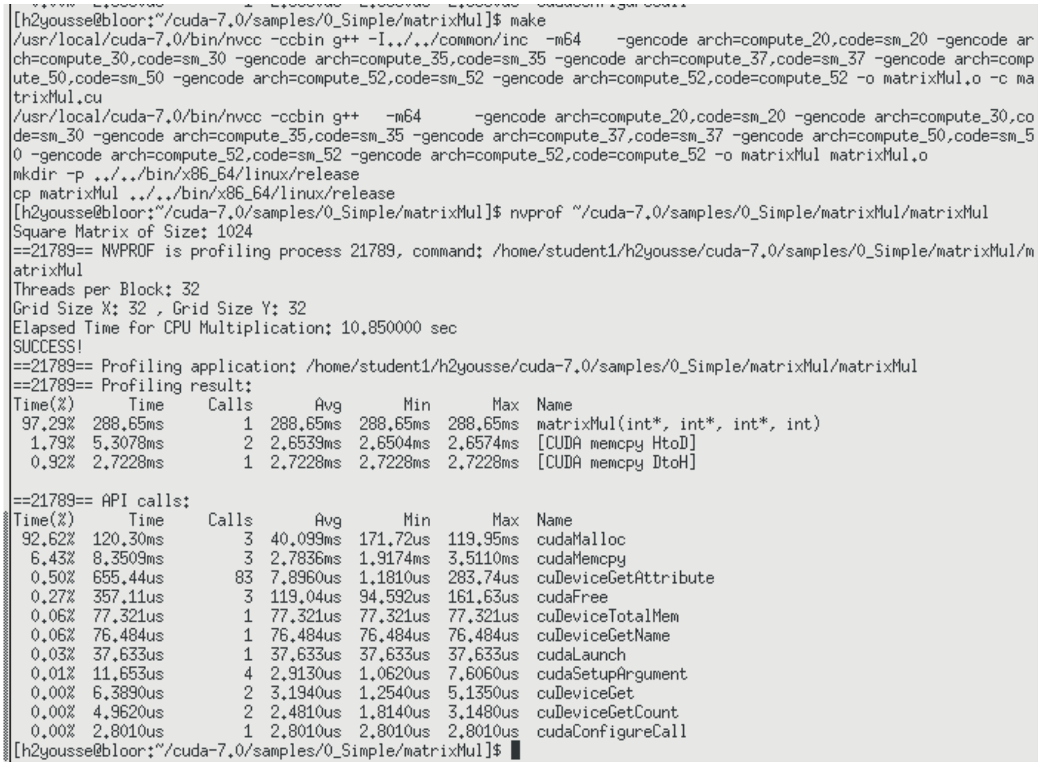
Figure 12: Console Results for Matrix Multiplication Application Using Matrices of Size 1024 x 1024 and 8 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,code=sm_20 -gencode ar
ch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=comp
ute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul.o -c ma
trixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,co
de=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_5
0 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 1024
==21538== NVPROF is profiling process 21538, command: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/m
atrixMul
Threads per Block: 16
Grid Size X: 64 , Grid Size Y: 64
Elapsed Time for CPU Multiplication: 10.990000 sec
SUCCESS!
==21538== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
==21538== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.87%  247.44ms         1  247.44ms  247.44ms  247.44ms  matrixMul(int*, int*, int*, int)
  2.08%  5.3076ms         2  2.6538ms  2.6503ms  2.6573ms  [CUDA memcpy HtoD]
  1.05%  2.6946ms         1  2.6946ms  2.6946ms  2.6946ms  [CUDA memcpy DtoH]

==21538== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 92.32%  112.55ms         3  37.517ms  153.72us  112.23ms  cudaMalloc
  6.66%  8.1196ms         3  2.7065ms  1.8810ms  3.3279ms  cudaMemcpy
  0.52%  636.93us        83  7.6730us  1.2600us  254.85us  cuDeviceGetAttribute
  0.32%  394.42us         3  131.47us  95.575us  198.85us  cudaFree
  0.07%  86.003us         1  86.003us  86.003us  86.003us  cuDeviceTotalMem
  0.05%  62.627us         1  62.627us  62.627us  62.627us  cuDeviceGetName
  0.03%  34.235us         1  34.235us  34.235us  34.235us  cudaLaunch
  0.01%  10.364us         4  2.5910us     947ns  6.9070us  cudaSetupArgument
  0.00%  6.0240us         2  3.0120us  1.3720us  4.6520us  cuDeviceGet
  0.00%  4.8690us         2  2.4340us  1.8360us  3.0330us  cuDeviceGetCount
  0.00%  2.8960us         1  2.8960us  2.8960us  2.8960us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ █
```

Figure 13: Console Results for Matrix Multiplication Application Using Matrices of Size 1024 x 1024 and 16 Threads per Block



```
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ make
/usr/local/cuda-7.0/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode arch=compute_20,code=sm_20 -gencode ar
ch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=comp
ute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul.o -c ma
trixMul.cu
/usr/local/cuda-7.0/bin/nvcc -ccbin g++   -m64      -gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,co
de=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_5
0 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52 -o matrixMul matrixMul.o
mkdir -p ../../bin/x86_64/linux/release
cp matrixMul ../../bin/x86_64/linux/release
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ nvprof ~/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
Square Matrix of Size: 1024
==21789== NVPROF is profiling process 21789, command: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/m
atrixMul
Threads per Block: 32
Grid Size X: 32 , Grid Size Y: 32
Elapsed Time for CPU Multiplication: 10.850000 sec
SUCCESS!
==21789== Profiling application: /home/student1/h2yousse/cuda-7.0/samples/0_Simple/matrixMul/matrixMul
==21789== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 97.29%  288.65ms         1  288.65ms  288.65ms  288.65ms  matrixMul(int*, int*, int*, int)
  1.79%  5.3078ms         2  2.6539ms  2.6504ms  2.6574ms  [CUDA memcpy HtoD]
  0.92%  2.7228ms         1  2.7228ms  2.7228ms  2.7228ms  [CUDA memcpy DtoH]

==21789== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 92.62%  120.30ms         3  40.099ms  171.72us  119.95ms  cudaMalloc
  6.43%  8.3509ms         3  2.7836ms  1.9174ms  3.5110ms  cudaMemcpy
  0.50%  655.44us        83  7.8960us  1.1810us  283.74us  cuDeviceGetAttribute
  0.27%  357.11us         3  119.04us  94.592us  161.63us  cudaFree
  0.06%  77.321us         1  77.321us  77.321us  77.321us  cuDeviceTotalMem
  0.06%  76.484us         1  76.484us  76.484us  76.484us  cuDeviceGetName
  0.03%  37.633us         1  37.633us  37.633us  37.633us  cudaLaunch
  0.01%  11.653us         4  2.9130us  1.0620us  7.6060us  cudaSetupArgument
  0.00%  6.3890us         2  3.1940us  1.2540us  5.1350us  cuDeviceGet
  0.00%  4.9620us         2  2.4810us  1.8140us  3.1480us  cuDeviceGetCount
  0.00%  2.8010us         1  2.8010us  2.8010us  2.8010us  cudaConfigureCall
[h2yousse@bloor:~/cuda-7.0/samples/0_Simple/matrixMul]$ █
```

Figure 13: Console Results for Matrix Multiplication Application Using Matrices of Size 1024 x 1024 and 16 Threads per Block