BOISE STATE UNIVERSITY

(BSU, USA)

Name: Brian KYANJO
Course: Parallel Scientific Computing

Project Number: 3
Date: April 19, 2021

Problem description

l have explored different ways to compute matrix multiplication on a GPU using CUDA. The multiplication of two matrices A and B (we assume they are square) is another square matrix C, such that:

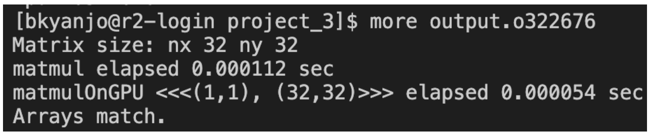
$$C_{ij} = \sum_{k=0}^{N-1} A_{ik} B_{kj} \tag{1}$$

where N is the matrix size (i.e. $A, B, C \in \mathbb{R}^{N \times N}$), the first and second index in the subscript enumerates row and column of a matrix respectively.

Task 1 - Monolithic approach

1. The provided serial code for matrix-matrix multiplication has been used to create a CUDA code which follows the CUDA programing model (allocate arrays on the device, move data to the device, execute kernel, copy result to host, free memory on the device) named **matmul_task1.cu**

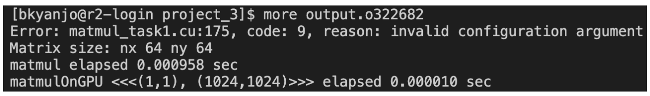
2. A problem size of N = 32 with one block of 1024 threads has been used to run the kernel (**matmulOnGPU**). And it was confirmed that the kernel produces the same results as the serial code **matmul** as depicted in figure 1 below.



```
[bkyanjo@r2-login project_3]$ more output.o322676
Matrix size: nx 32 ny 32
matmul elapsed 0.000112 sec
matmulOnGPU <<<(1,1), (32,32)>>> elapsed 0.000054 sec
Arrays match.
```

Figure 1: Shows the output from the serial code and the kernel

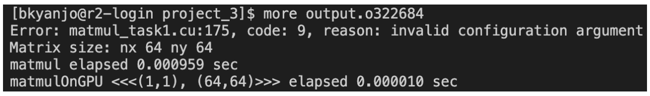
3. When a problem size ($N = 64$) larger than $N = 32$ was used, with one block and 1024 threads, the code prints out an execption shown in figure 2



```
[bkyanjo@r2-login project_3]$ more output.o322682
Error: matmul_task1.cu:175, code: 9, reason: invalid configuration argument
Matrix size: nx 64 ny 64
matmul elapsed 0.000958 sec
matmulOnGPU <<<(1,1), (1024,1024)>>> elapsed 0.000010 sec
```

Figure 2: $N > 32$

When the number of threads (4096) are increased to match the new problem size ($N = 64$) leaving the grid size constant, again the code prints out an execption as shown in figure 3.



```
[bkyanjo@r2-login project_3]$ more output.o322684
Error: matmul_task1.cu:175, code: 9, reason: invalid configuration argument
Matrix size: nx 64 ny 64
matmul elapsed 0.000959 sec
matmulOnGPU <<<(1,1), (64,64)>>> elapsed 0.000010 sec
```

Figure 3: N matching the number of threads.

4. The entire CUDA code (including memory transfers) was timed and compared with the serial code. The timing results are shown in table 1.

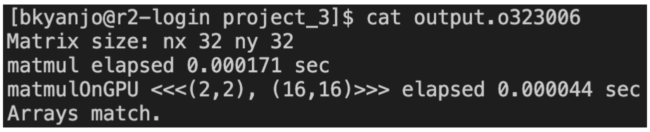
Codes	N	Elapsed time
Serial	32	0.000176
Cuda	32	0.180669

Table 1: Timing results from both Serial and Cuda codes.

Device acceleration does not give any benefit for a single small matrix multiplication, since much time is spent on transfering the data between the host and the device, which is not accounted for in the serial code.

Task 2 - 2D grid of 2D blocks decomposition

1. The 2D-2D matrix-multiplication algorithm in CUDA (**matmul_task2.cu**) was implemented and we confirmed that it gives the same result as the serial code. Its output has been displayed in figure 4 below.



```
[bkyanjo@r2-login project_3]$ cat output.o323006
Matrix size: nx 32 ny 32
matmul elapsed 0.000171 sec
matmulOnGPU <<<(2,2), (16,16)>>> elapsed 0.000044 sec
Arrays match.
```

Figure 4: N matching the number of threads.

2. A large problem size $N = 2^{11}$ has been used, with a few block configurations as shown in table 2. The timing results displayed in table 2 for Cuda, is kernel time only. All the results obtained are correct and they are saved in a file name **task2.dat**, and its comparision with serial code is depicted in table 2.

Block Configuration	Elapsed time (Cuda) sec	Elapsed time (Serial) sec
32×32	0.056697	58.261000
16×16	0.059276	58.703160
8×8	0.071871	58.481528
4×4	0.151849	58.427382
1×1	1.555273	58.463656

Table 2: Timing results from both Serial and Cuda codes.

The table 2 above represents that Cuda timing results, raise as the block configuration decreases with a fixed big grid ($2^{11} \times 2^{11}$). However, its still much better than the matrix multiplication in the serial code, since the time taken by the serial code is almost a minute. Hence, we conclude by choosing Cuda with any block configuration, because it will still be much computationally cheap and fast compared to the serial code.

Task 3 - 1D-1D decomposition

1. The 1D-1D decomposition for CUDA matrix multiplication named **matmul_task2.cu** was implemented and compared with the serial code as shown in figure 5.

```
[bkyanjo@r2-login project_3]$ more output.o323745
Matrix size: nx 32 ny 32
matmul elapsed 0.000118 sec
matmulOnGPU <<<(1,1), (1,32)>>> elapsed 0.000088 sec
Arrays match.
```

Figure 5: 1D-1D decomposition for CUDA matrix multiplication.

2. The same problem size as in Task 2 has been used and only the kernel is timed for different configurations of threads per block (i.e. 1, 16, 64, 256, 1024), and correct results **(task3_2.dat)** where obtained. Results are compared with Task 2 as shown in figure 6.

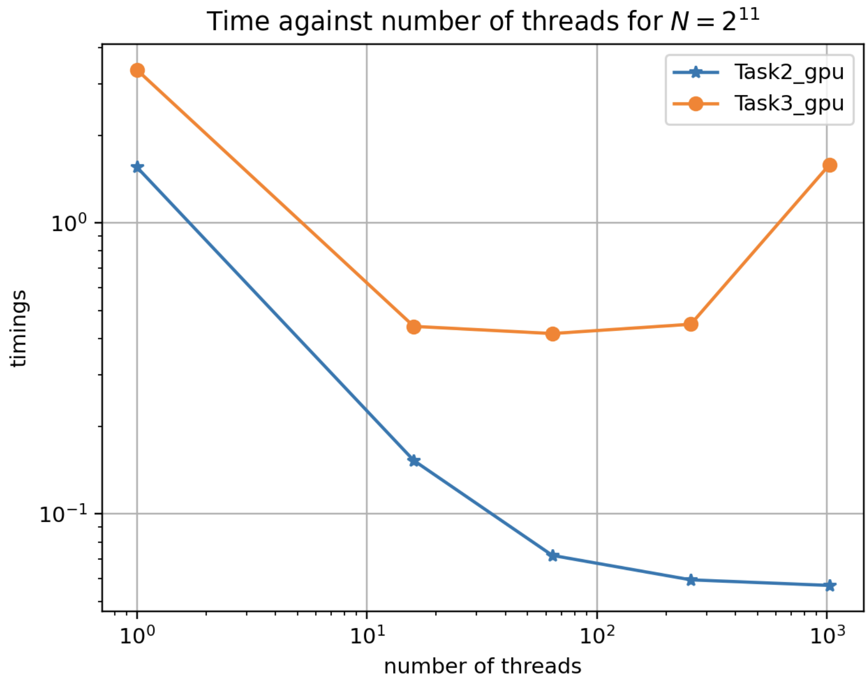


Figure 6: Task2 and Task3 kernel timings.

3. According to figure 6, task2 (blue) performs much better than task3(orange) computationwise as the number of threads increases. On this note i would choose a 2D-2D decomposition approach with 1024 by 1024 configuration, since the more the threads the fast and cheaper the code becomes. I conclude that 2D-2D performs better than 1D-1D, with all configurations.

Mastery

1. The serial code together with CUDA kernel code (**mastery_1.cu**) performing matrix-vector product $b = Ax$ has been implemented, and its correctness has been tested as shown in figure 7 . I used a 1D-1D block decomposition, since it gave the least simulation time, durring all runs i performed on the other decompositions.

   Explain your choice of block-thread decomposition.

```
[bkyanjo@r2-login project_3]$ more output.o324215
Matrix size: nx 32 ny 32
matvecmul elapsed 0.000004 sec
matvecmulOnGPU <<<(1,1), (32,1)>>> elapsed 0.000043 sec
Arrays match.
```

Figure 7: Matrix vector multiplication

2. The serial code together with CUDA kernel code (**mastery_2.cu**) performing euclidian norm of a vector has been implemented, and its correctness has been tested as shown in figure 8 .

```
[bkyanjo@r2-login project_3]$ more output.o324208
Vector size 32
euclideanOnHost elapsed 0.000012 sec
euclideanOnGPU <<<(1,256), (0,0)>>> Time elapsed 0.000049 sec
Results match.
```

Figure 8: Euclidean norm.

3. The normalization kernel (**mastery_3.cu**), that divides each entry of vector b by its Euclidian norm has been implemented and results are proved as shown in figure 9.

```
[bkyanjo@r2-login project_3]$ more output.o324388
Vector size 32
normOnHost elapsed 0.000000 sec
normOnGPU <<<(1,256), (0,0)>>> Time elapsed 0.000048 sec
Arrays match.
```

Figure 9: Normalisation of vector b.

4. The serial power iteration code together with its kernel (**mastery_serial_gpu.cu**) has been implemented. And the results are depicted in 10

```
[bkyanjo@r2-login project_3]$ cat output.o324895
Matrix size: nx 32 ny 32
Arrays match.
```

Figure 10: Power iteration for both serial and its kernel.

5. Both the serial code and the CUDA code has been timed, and the results are depicted in the figure 11 below.

```
[bkyanjo@r2-login project_3]$ cat output.o324897
N = 2048        serial_time = 2.970818  kernel_time = 5.507358
```

Figure 11: Power iteration for both serial and its kernel.

   I have used 2D-2D decomposition block thread decompostion, since it proved to give the optimal results for all runs i have performed, and also makes more threads available per block.

6. No, since computation time for serial is still almost half of the kernel one. I think this can be improved by reducing on the amount of data transfers between the device and the host, since it seems to be more expensive than the actual calculations.