PARALLEL COMPUTING

ASSIGNMENT 1
SOLVING LOWER TRIANGULAR SYSTEM OF LINEAR EQUATIONS

FILES INCLUDED

1. openMPversion.c
2. cudaVersion1.cu
3. cudaVersion2a.cu
4. cudaVersion2b.cu
5. README.txt

STEPS TO RUN THE SUBMISSION

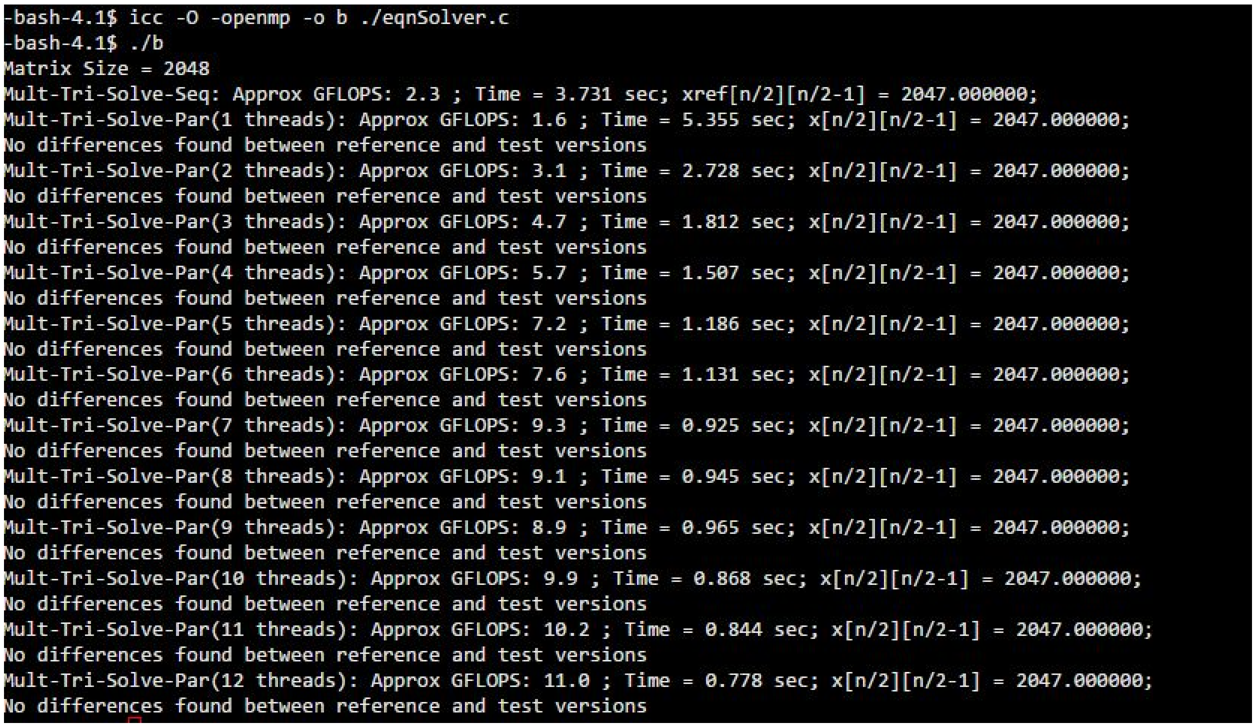
(Instructions to run the OpenMP version.)

1. Compile the file using the following command:
   ```
   icc -O -openmp ./openMPversion.cu -o openmpver
   ```
2. Run the output .
   ```
   ./openmpver
   ```

(Instructions to run the cuda version.)

1. Compile the file using the following command:
   ```
   nvcc -O -o cuda1 ./cudaversion.cu
   ```
2. Run the output .
   ```
   ./cuda1
   ```

OpenMP version



```
-bash-4.1$ icc -O -openmp -o b ./eqnSolver.c
-bash-4.1$ ./b
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 2.3 ; Time = 3.731 sec; xref[n/2][n/2-1] = 2047.000000;
Mult-Tri-Solve-Par(1 threads): Approx GFLOPS: 1.6 ; Time = 5.355 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(2 threads): Approx GFLOPS: 3.1 ; Time = 2.728 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(3 threads): Approx GFLOPS: 4.7 ; Time = 1.812 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(4 threads): Approx GFLOPS: 5.7 ; Time = 1.507 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(5 threads): Approx GFLOPS: 7.2 ; Time = 1.186 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(6 threads): Approx GFLOPS: 7.6 ; Time = 1.131 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(7 threads): Approx GFLOPS: 9.3 ; Time = 0.925 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(8 threads): Approx GFLOPS: 9.1 ; Time = 0.945 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(9 threads): Approx GFLOPS: 8.9 ; Time = 0.965 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(10 threads): Approx GFLOPS: 9.9 ; Time = 0.868 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(11 threads): Approx GFLOPS: 10.2 ; Time = 0.844 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
Mult-Tri-Solve-Par(12 threads): Approx GFLOPS: 11.0 ; Time = 0.778 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
```

OBSERVATIONS:

1. Requested number of threads is being allocated by the openMP compiler.
2. The serial version is taking around 3.
3. The performance is improving for increase in the number of threads.
4. The performance is peaking around 11.0 GFLOPS.
5. The increase in performance is not entirely non-decreasing. we observe a slight reduce in performance when we run the algorithm with 9 threads.

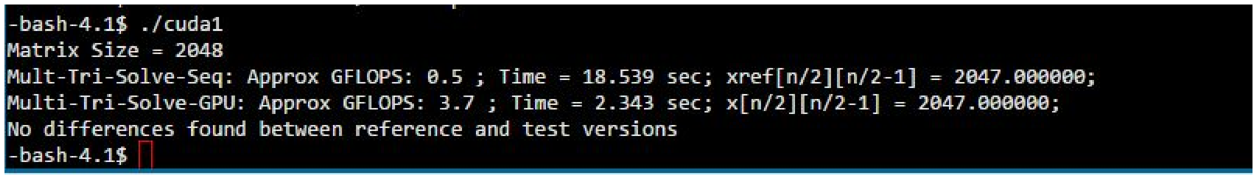
CUDA Version

APPROACH 1 :

The dimensions of the grid and block were tweaked in order to study the performance. There was no much improvement beyond 3.7 GFLOPs.

OBSERVATIONS:

1. The serial version was taking around 18.54s with performance of around 0.5 GFLOPs.
2. For a grid dimension of <32,32> and block dimension of <2,2> a peak performance of 3.7 GFLOPs.



```
-bash-4.1$ ./cuda1
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.539 sec; xref[n/2][n/2-1] = 2047.000000;
Multi-Tri-Solve-GPU: Approx GFLOPS: 3.7 ; Time = 2.343 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
-bash-4.1$
```

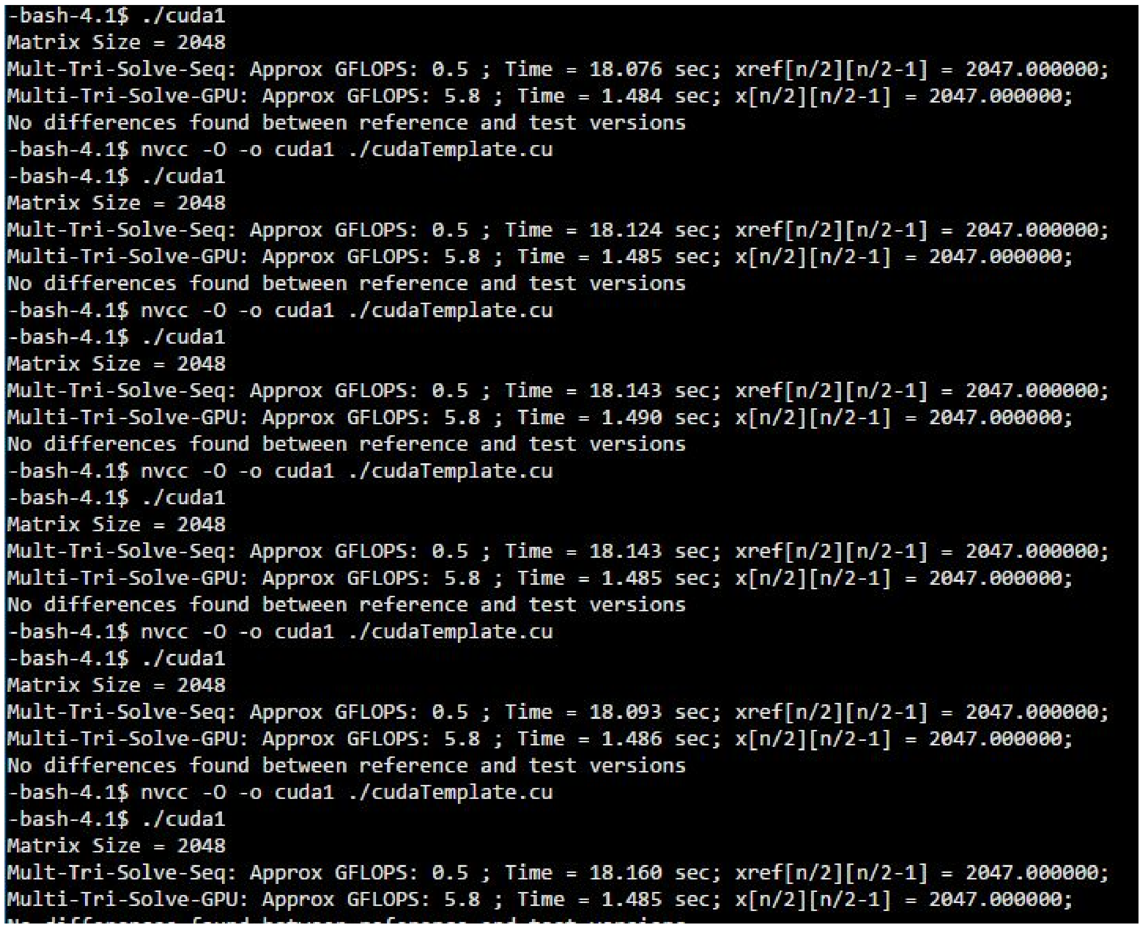
3. But this is a not a suitable way of trying improve performance. We can explore other ways like changing access patterns.

APPROACH 2 :

The matrices were transposed to change the access pattern. Now the access pattern of the matrices is column wise. The transpose function was called in the cpu. The overhead cost paid for transposing the elements of the matrix is marginal.

OBSERVATIONS:

1. We observed that access pattern along column had much better performance as compared to the row-wise access pattern. This is due to the fact the data required for all the threads that were running simultaneously were loaded at one go in case of column-wise access pattern.
2. A peak performance of 5.8 GFLOPs was observed even for different grid and block dimensions.



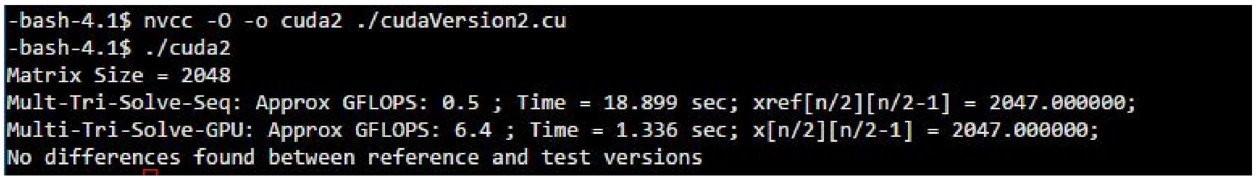
```
-bash-4.1$ ./cuda1
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.076 sec; xref[n/2][n/2-1] = 2047.000000;
Multi-Tri-Solve-GPU: Approx GFLOPS: 5.8 ; Time = 1.484 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
-bash-4.1$ nvcc -O -o cuda1 ./cudaTemplate.cu
-bash-4.1$ ./cuda1
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.124 sec; xref[n/2][n/2-1] = 2047.000000;
Multi-Tri-Solve-GPU: Approx GFLOPS: 5.8 ; Time = 1.485 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
-bash-4.1$ nvcc -O -o cuda1 ./cudaTemplate.cu
-bash-4.1$ ./cuda1
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.143 sec; xref[n/2][n/2-1] = 2047.000000;
Multi-Tri-Solve-GPU: Approx GFLOPS: 5.8 ; Time = 1.490 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
-bash-4.1$ nvcc -O -o cuda1 ./cudaTemplate.cu
-bash-4.1$ ./cuda1
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.143 sec; xref[n/2][n/2-1] = 2047.000000;
Multi-Tri-Solve-GPU: Approx GFLOPS: 5.8 ; Time = 1.485 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
-bash-4.1$ nvcc -O -o cuda1 ./cudaTemplate.cu
-bash-4.1$ ./cuda1
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.093 sec; xref[n/2][n/2-1] = 2047.000000;
Multi-Tri-Solve-GPU: Approx GFLOPS: 5.8 ; Time = 1.486 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
-bash-4.1$ nvcc -O -o cuda1 ./cudaTemplate.cu
-bash-4.1$ ./cuda1
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.160 sec; xref[n/2][n/2-1] = 2047.000000;
Multi-Tri-Solve-GPU: Approx GFLOPS: 5.8 ; Time = 1.485 sec; x[n/2][n/2-1] = 2047.000000;
```

APPROACH 3 :

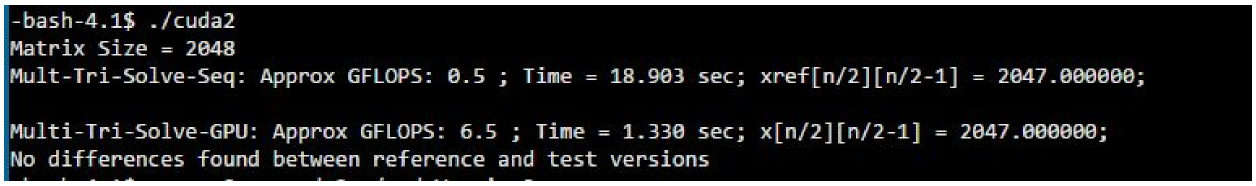
The matrices were transposed to change the access pattern. Now the access pattern of the matrices is column wise. **This time the transpose function was called as a kernel function.** Since the transpose function is a kernel function, the cost we are paying for this overhead is very minimal as compared to the previous case.

OBSERVATIONS:

1. We observed a much better performance as compared to that we observed in approach 2.
2. A peak performance of **6.5 GFLOP**s was observed even for different grid and block dimensions.



```
-bash-4.1$ nvcc -O -o cuda2 ./cudaVersion2.cu
-bash-4.1$ ./cuda2
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.899 sec; xref[n/2][n/2-1] = 2047.000000;
Multi-Tri-Solve-GPU: Approx GFLOPS: 6.4 ; Time = 1.336 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
```



```
-bash-4.1$ ./cuda2
Matrix Size = 2048
Mult-Tri-Solve-Seq: Approx GFLOPS: 0.5 ; Time = 18.903 sec; xref[n/2][n/2-1] = 2047.000000;

Multi-Tri-Solve-GPU: Approx GFLOPS: 6.5 ; Time = 1.330 sec; x[n/2][n/2-1] = 2047.000000;
No differences found between reference and test versions
```

OVERALL INFERENCES:

1. Changing the loop order is impossible, since there are inherent loop dependencies.
2. The only way we can leverage maximum parallelism from GPUs is by changing the access order.