Priyanka Karuppuch Samy

729007151

HW5

1. Compilation and execution
    - The zip file contains a GPR_final_lu.cu, a.out. The compilation is done using the command nvcc -O3 GPR_final_lu.cu.
    - The output screenshot is also attached.
    - When compiled, a new a.out will be created and it can be run using the command ./a.out #m #rstar[0] #rstar[1].
    - Eg: ./a.out 30 0.5 0.5
    - To enable the disable running on GPU, the GPU function calls can commented out and some code segments has to be enabled. This is clearly mentioned in the comments in file GPR_final_lu.cu.

2. Strategy to Parallelize LU
    - LU factorization produces an upper triangular and lower triangular matrix. To do this there are 3 loops required. The outer loop couldn't be parallelized because of a variable that has to be executed in order. Thus, I just parallelized inner loops.
    - All the updates and calculations for L and U matrices are done in the device. Once done, I copied back L and U for the next steps on host.
    - As described in the HW, I copied the matrices U and K to the device to run on it and compute tI + K.
    - Once that computation was done, since the code I had written was highly modular, I had to make another copy of U and L to the device and then performed LU decomposition on the device. I then copied it on
    - I didn't parallelize the solver routine.
    - The number of floating point operations are going to be the same as HW3 because, I just edited that code to accommodate for GPU devices.
    - Each block has 32 threads to use as many multiprocessors as possible. So, the number of blocks are going to be, based on n. The formula I used for number of blocks is n / threadsperblock + 1.

```
[priyanka1331@ada4 Cuda_HW5]$ ./a.out 10 0.5 0.5
LU exec time CPU: 8.90821ms
Solver routine time CPU: 0.414246ms

Total time CPU: 0.146466s

fstar=1.00762
[priyanka1331@ada4 Cuda_HW5]$ ./a.out 20 0.5 0.5
LU exec time CPU: 568.57ms
Solver routine time CPU: 5.92ms

Total time CPU: 4.92238s

fstar=1.00718
[priyanka1331@ada4 Cuda_HW5]$ ./a.out 30 0.5 0.5
LU exec time CPU: 6473.58ms
Solver routine time CPU: 29.545ms

Total time CPU: 54.9573s

fstar=1.00631
[priyanka1331@ada4 Cuda_HW5]$ ./a.out 40 0.5 0.5
LU exec time CPU: 36403.5ms
Solver routine time CPU: 92.8491ms

Total time CPU: 308.808s

fstar=1.00129
```

```
[priyanka1331@ada4 Cuda_HW5]$ ./a.out 10 0.5 0.5
LU exec time GPU: 20.7026ms
Solver routine time GPU: 0.414842ms

Total time GPU: 0.175158s

fstar=1.00762
[priyanka1331@ada4 Cuda_HW5]$ ./a.out 20 0.5 0.5
LU exec time GPU: 405.161ms
Solver routine time GPU: 5.92394ms

Total time GPU: 4.82886s

fstar=1.00718
[priyanka1331@ada4 Cuda_HW5]$ ./a.out 30 0.5 0.5
LU exec time GPU: 1836.33ms
Solver routine time GPU: 29.5622ms

Total time GPU: 51.0897s

fstar=1.00631
[priyanka1331@ada4 Cuda_HW5]$ ./a.out 40 0.5 0.5
LU exec time GPU: 5802.39ms
Solver routine time GPU: 92.8192ms

Total time GPU: 282.485s

fstar=1.00129
```

3. Flop Rate Calculation of solver and factorization routines:

- Actual GPU FLOPS = 1.7TFLOPS
- M = 40
- Number of operations in Factorization routine is 2046720000*4(subtractions and multiplication are 2 different operations) + (1600*1600)div operation for finding fac = 8189440000
- Execution time : 1 core – LU routine – 36403 ms
- Execution time : 31 cores – LU routine – 5802 ms
- FLOPS for 31 cores in LU routine – (8189440000/ 5802 ms) = 1.4 GFlops
- Speed up on Factorization routine = 6.27
- Efficiency on Factorization routine = 20%

The FLOPs is lower than that of the peak FLOPs which is 1.7TFLOPS mainly because we were not able to use all the cores in the GPU and also memory access from DRAM of GPU is going to significantly slow down the speed.

I haven't done any parallelization in solver routine.

- Speed up and efficiency achieved overall:
  m = 40
  n = 1600
  no of blocks = 1600/32 + 1

  #threads = 1, time : 308.808 s
  #threads = 1632, time: 282.485 s
  Speed up = 1.095
  Efficiency = 4% (using number of cores – using blocks)

The speed up and efficiency overall is significantly lower compared to LU routine is because of copying data back and forth to the device.