Introduction To Parallel Computing

Report 3

Cuda Programming

STEPS TO RUN THE SUBMISSION

(Instructions to run on Ohio Supercomputer.)

1. Compile the file using the following command:
   nvcc -O -o cuda_1 ./cuda_1.cu

2. Run the output .
   ./cuda_1

3. Compile and run the serial version.
   g++ serial_1.cpp
   ./a.out

Part 1

A[i][j] = A[i-1][j+1] + A[i][j+1];

Number of floating point operations = 100*4096*4096 = 1600 MFLOP = 1.6 GFLOP approx

Run time of the serial version : 8.23 seconds
GFLOPs for the serial version : 0.194 GFLOPs

Time Taken for the parallel version :

	<gridDIm, blockDIm>	<Time in Seconds>	<GFLOPs>
1.	<(241,241),(17,17)>	1.477	1.088
2.	<(257,257),(16,16)>	1.287	1.25
3.	<(129,129),(32,32)>	1.456	1.103
4.	<(513,513),(8,8)>	0.899	1.78
5.	<(1025,1025),(4,4)>	1.316	1.22

The code seems to be working faster when the number of threads per block is 64 threads.

Part 2

Multiplying the Matrix with the transpose of itself.

Number of floating point operations : 4096*4096*4096 = 64 Giga floating point operations.

Run time of the serial version : The serial version was taking lot of time to run for an array of size 4096*4096. So I had to kill the process. Instead I ran the serial version of the code for a much smaller array and extracted the GFLOPs for the same. This should give us a rough idea to compare it with the parallel version. (Assuming serial version code scales up in a linear fashion with respect to the size of the array.)
Time Taken running the serial version : 34.49 seconds (array size = 2048*2048);
GFLOPs for the serial version : 0.11 GFLOPs

Time Taken for the parallel version :

	<gridDIm, blockDIm>	<Time in Seconds>	<GFLOPs>
1.	<(2048,2048),(2,2)>	38.581	1.68
2.	<(1024,1024),(4,4)>	20.114	3.18
3.	<(512,512),(8,8)>	14.8	4.32
4.	<(256,256),(16,16)>	40.252	1.59
5.	<(128,128),(32,32)>	39.15	1.63

The code seems to be working faster when the number of threads per block is 64 threads.

Part 3

Reversing the contents of a matrix and verifying the reversal..

1. One block of 32 threads
   It is taking 20.0329 milli-seconds.

2. One block of 1024 threads.
   Unable to reverse with one block of 1024 threads.

3. Four blocks of 1024 threads each.
   It is taking 1.515 milli-seconds.

Calculating Running Time on Device

I am using the library function of cuda to calculate time. Following are the functions being used.

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start);
cudaMemcpy(d_A, h_A, memSize,cudaMemcpyHostToDevice);
cudaMemcpy(d_C, h_C, memSize,cudaMemcpyHostToDevice);
initArray<<< dimGrid, dimBlock >>>(d_A,d_C);
cudaMemcpy(h_C, d_C, memSize,cudaMemcpyDeviceToHost);
cudaEventRecord(stop);
cudaEventSynchronize(stop);
float milliseconds = 0;
cudaEventElapsedTime(&milliseconds, start, stop);
cout<<"Time taken to complete parallel version : "<<milliseconds<<"
    milliseconds"<<endl;
```

Observations

1. Both in part 1 and part2 , we observed that the cuda code is running optimum when number of threads per block is 64.
2. Parallel version is definitely running faster as compared to the serial version.
3. For the part 3, The cuda program is running faster for 4 blocks having 1024 threads each.