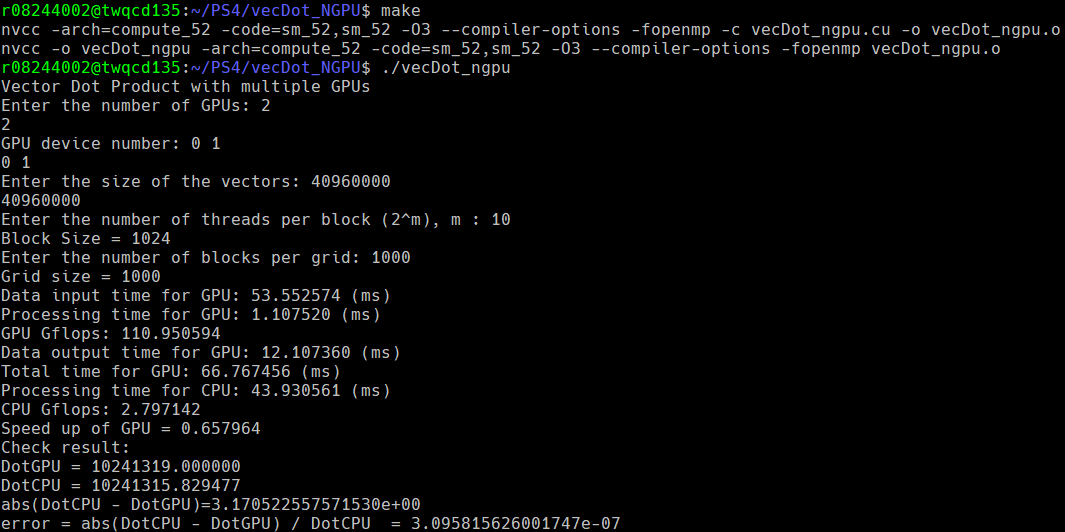
**CUDA Parallel Programming Problem Set 4**

R08244002 Shin-Rong Tsai

* **Dot product with multi-GPU**

1. **Result**

Save the code in *vecDot\_ngpu.cu*. Write another code with for loops to loop through all the block size and grid size in order to find the optimal grid size and block size, save in *vecDot\_ngpu\_Optimize.cu*.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| 10 | 363.16 | 236.51 | 145.50 | 105.31 | 80.36 | 75.16 | 64.84 | 62.75 | 62.84 | 63.10 |
| 100 | 92.11 | 81.49 | 68.71 | 64.82 | 62.39 | 61.48 | 61.06 | 60.77 | 63.42 | 62.92 |
| 1000 | 77.12 | 65.25 | 63.48 | 61.37 | 60.76 | 60.67 | 61.22 | 61.10 | 62.56 | 62.72 |
| 10000 | 74.76 | 65.08 | 62.83 | 61.55 | 61.15 | 61.47 | 61.62 | 62.99 | 63.26 | 63.78 |

Total time used by GPU­ (ms):

Total time used by GPU (ms):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| 10 | 646.95 | 391.97 | 238.41 | 150.93 | 110.84 | 90.35 | 79.89 | 74.77 | 72.39 | 71.49 |
| 100 | 134.26 | 102.76 | 87.39 | 78.44 | 74.12 | 72.16 | 71.74 | 71.57 | 71.58 | 71.73 |
| 1000 | 90.28 | 80.72 | 76.31 | 72.83 | 71.50 | 71.42 | 71.37 | 71.38 | 71.30 | 71.39 |
| 10000 | 87.63 | 80.67 | 75.63 | 72.65 | 71.82 | 71.60 | 71.72 | 71.65 | 71.56 | 71.84 |

1. **Discussion**
2. 2-GPU slightly matches the results from Problem Set 2, the more , the less it should read from device memory, which means faster.

If compared with 1-GPU, it somewhat saturates or even worse, when grid size is 10000, and block size more than 256. Possible reasons would be the post-processing and the preliminary work, which is to add up all the elements in the array of length equal to grid size, transferring and controlling the work flow of the program, copying data, might take up more time.

I think this is caused by we are using 2 OpenMP threads, the resources are split to two. The OpenMP thread cannot handle that much stuff and the bandwidth reaches the limit.

So from my result, for 2-GPU and consider total time used, the optimal grid size would be around 1000, and the corresponding optimal block size would be around 64.

1. Compare time used by GPU only in 1 GPU and 2 GPU:

Time used by GPU only (ms):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| 10 | 295.34 | 163.59 | 80.85 | 40.88 | 20.85 | 10.57 | 5.39 | 2.85 | 1.75 | 1.19 |
| 100 | 32.43 | 16.64 | 8.91 | 4.51 | 2.32 | 1.35 | 1.13 | 1.10 | 1.09 | 1.09 |
| 1000 | 11.54 | 5.65 | 3.49 | 1.80 | 1.19 | 1.11 | 1.09 | 1.10 | 1.12 | 1.12 |
| 10000 | 9.57 | 5.01 | 2.92 | 1.55 | 1.16 | 1.09 | 1.08 | 1.09 | 1.13 | 2.01 |

Time used by GPU only (ms):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| 10 | 577.82 | 322.80 | 168.82 | 81.77 | 41.70 | 21.14 | 10.76 | 5.69 | 3.16 | 2.28 |
| 100 | 64.82 | 33.35 | 17.84 | 9.00 | 4.64 | 2.65 | 2.22 | 2.18 | 2.17 | 2.15 |
| 1000 | 21.03 | 11.48 | 7.16 | 3.70 | 2.35 | 2.20 | 2.16 | 2.16 | 2.17 | 2.19 |
| 10000 | 18.10 | 11.06 | 6.13 | 3.18 | 2.30 | 2.17 | 2.15 | 2.14 | 2.15 | 2.35 |

From the above two tables, which exclude the effects caused by OpenMP multi-threads, only GPU is considered. The time used by 2-GPU is half of the 1-GPU time used, as expected, since the array to be dealt with is cut in half. And both of their results also matches Problem Set 2.

But for block size 1024 and grid size 10000, it’s probably because that the architecture of the GPU cannot handle it.

1. Race Condition:

When we add up the result from different blocks, if we do it with OpenMP threads, it will access same variable and encounter race condition. So I declared another array to store the results from each OpenMP threads, then sum up the array after OpenMP ends. I guess that it will speed up more if we are using more GPUs.