EE 217 GPU Final Project Report

Jinwei Zhang SID: 862114004

**1. Objective**

The project objective is to accelerate the computation of an algorithm in the advanced MRI reconstruction using GPGPUs. The algorithm in the form of serial computation was proposed in the article and shown in Fig. 1.

[Sam S. Stone, Justin P. Haldar, Stephanie C. Tsao, Wen-Mei W. Hwu, Zhi-Pei Liang, and Bradley P. Sutton. "Accelerating Advanced MRI Reconstructions on GPUs." In Computing Frontiers, 2008.](http://www.sciencedirect.com/science/article/pii/S0743731508000919)

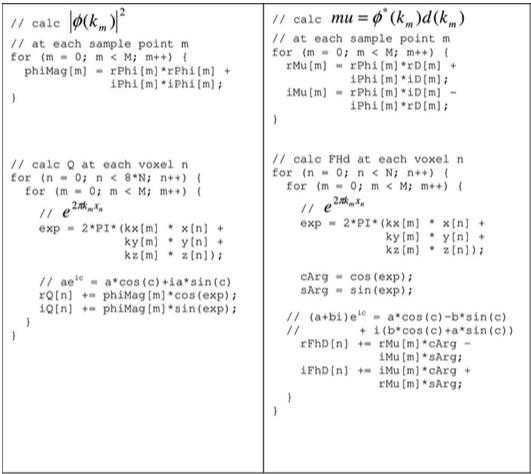


Figure 1. Q algorithm

**2. Computation of *phiMag* using GPU.**

The algorithm contains two parts, one is computing *phiMag* and the other is computing *Q* array. For the first part, *phiMag* can be computed using GPU in the similar way as the *vector-add* program. Each element of phiMag is assigned to a thread respectively. Considering the number of elements of *phiMag* is usually the multiplications of 1024, I assign 1024 threads in a block to minimize the control divergence in a block. Fig. 2 shows the kernel code of computing *phiMag*.

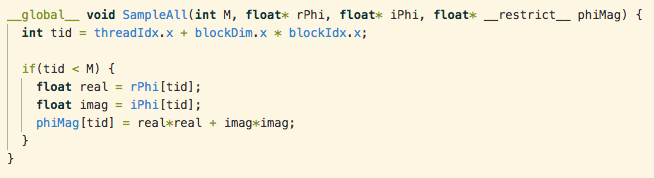


Fig. 2. Kernel code of computing *phiMag*.

**2. Computation of *Q* using GPU.**

The serial code of computing Q consists of two for-loops, seeking for both computation speed and hardware efficiency, the outer loop is substituted with GPU threads where each thread is responsible for one voxel. Inside each thread, the inner for-loop is executed in serial. Fig. 3 shows the code on host to allocate memory space on GPU global memory and copy data to GPU device and copy data back to host memory addresses.

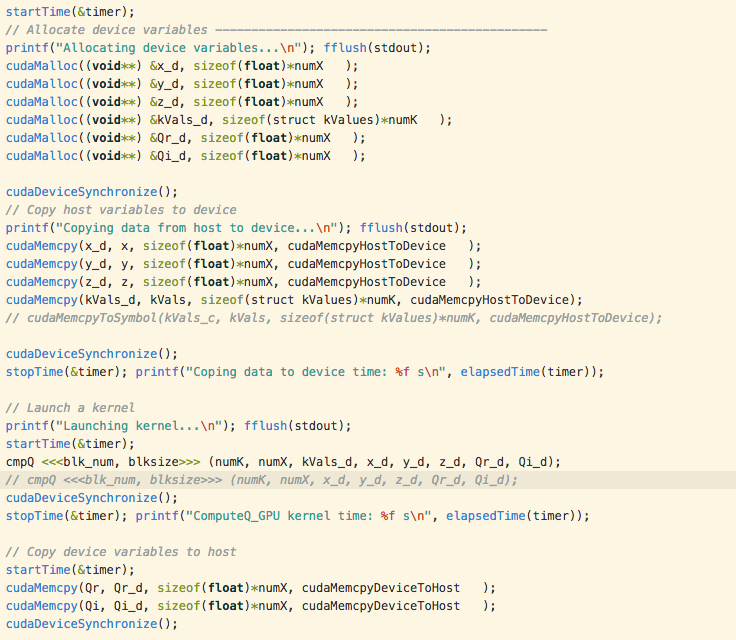


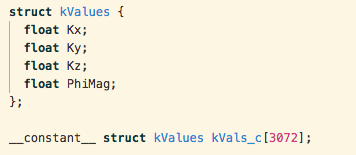
Fig. 3

**3. Handling the Memory Bandwidth Limitation**

There are two large pieces of data involved with the kernel function, k-values (kx, ky and kz) and voxel space values (x, y and z). In order to minimize the number of cycles of reading and writing to memory space and to reduce memory access times, we can extensively take advantage of constant memory and shared memory address spaces.

Due the number of voxel space values are dynamic and have a large range (e.g. from 323 to 1283), it is difficult to allocate voxel space values into constant memory or shared memory. However, k-value arrays are suitable data to be allocated into constant or dynamically allocated into shared memory on GPU.

1) Using constant memory for k-values array







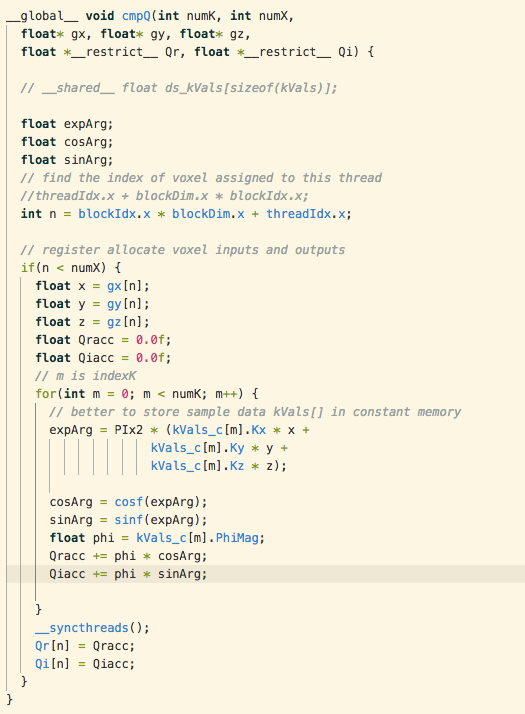
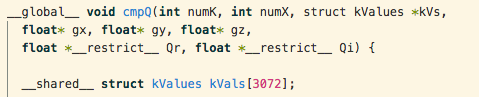


Fig. 4

2) Using shared memory for k-values array

We need to notice that each block needs to load the entire k-value array to its shared memory. In order to fast load, each thread is responsible for loading (numK-1)/threadID.x+1 elements into shared memory. For example, there are 1024 threads in a block and numK is 3072, then each thread loads 3 elements. To be more specific, thread-0 loads the 1st, 1024th and 2047th elements of k-value array to shared memory. Fig. 5 shows the kernel code considering using shared memory for storing k-value array.



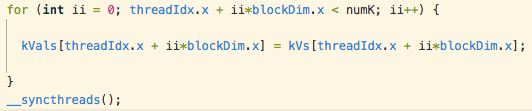
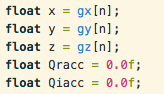


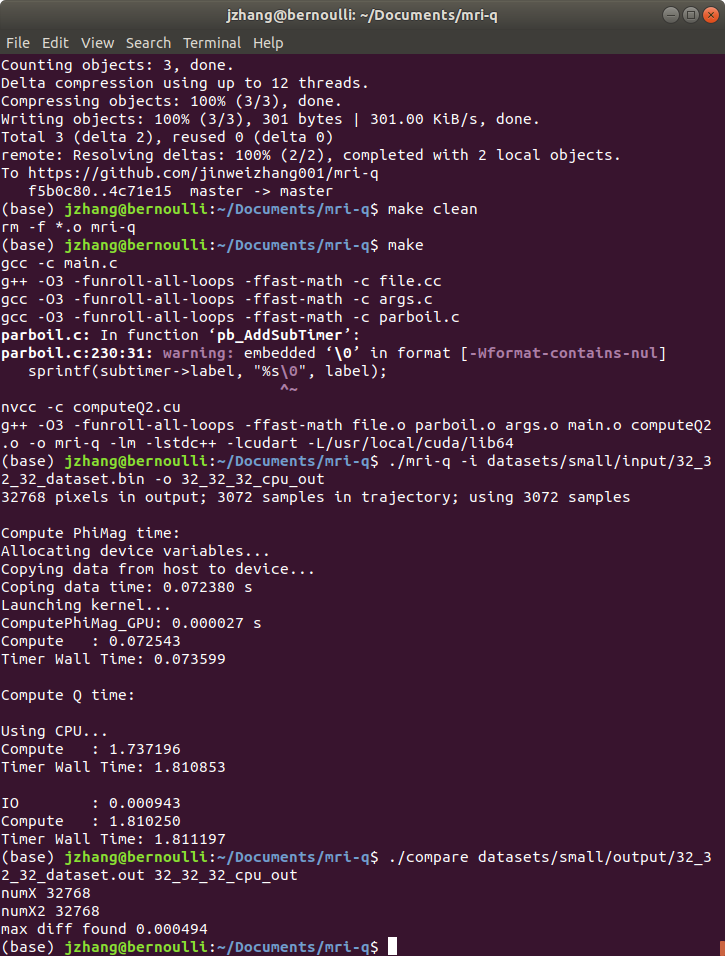
Fig. 5. Using shared memory for storing k-value array.

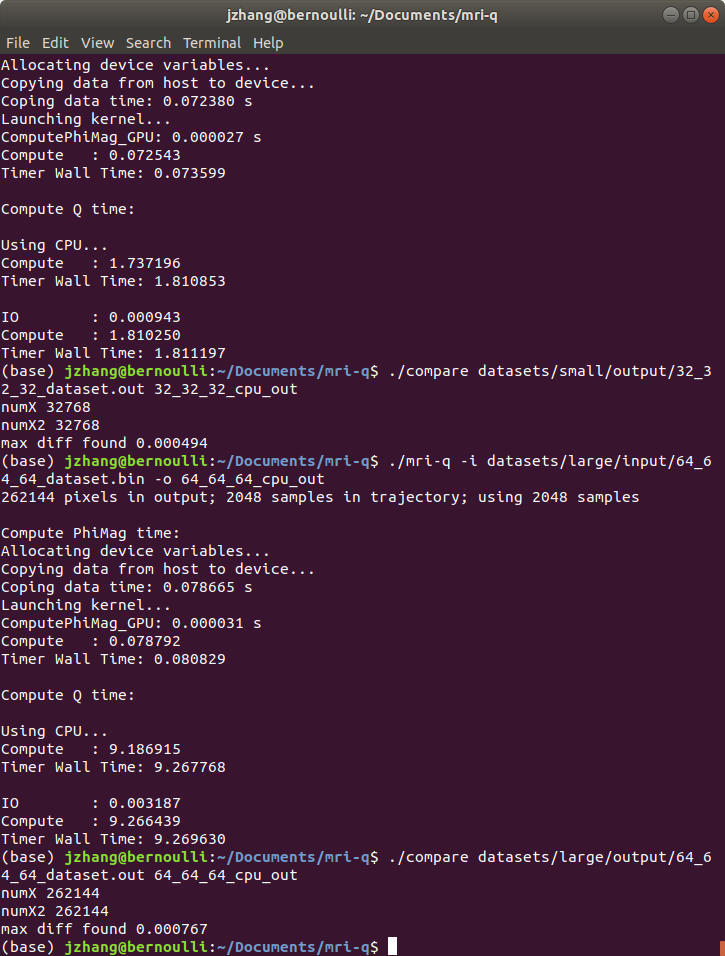
3) Using register for storing voxel space values (x, y and z) to reduce global memory accesses.

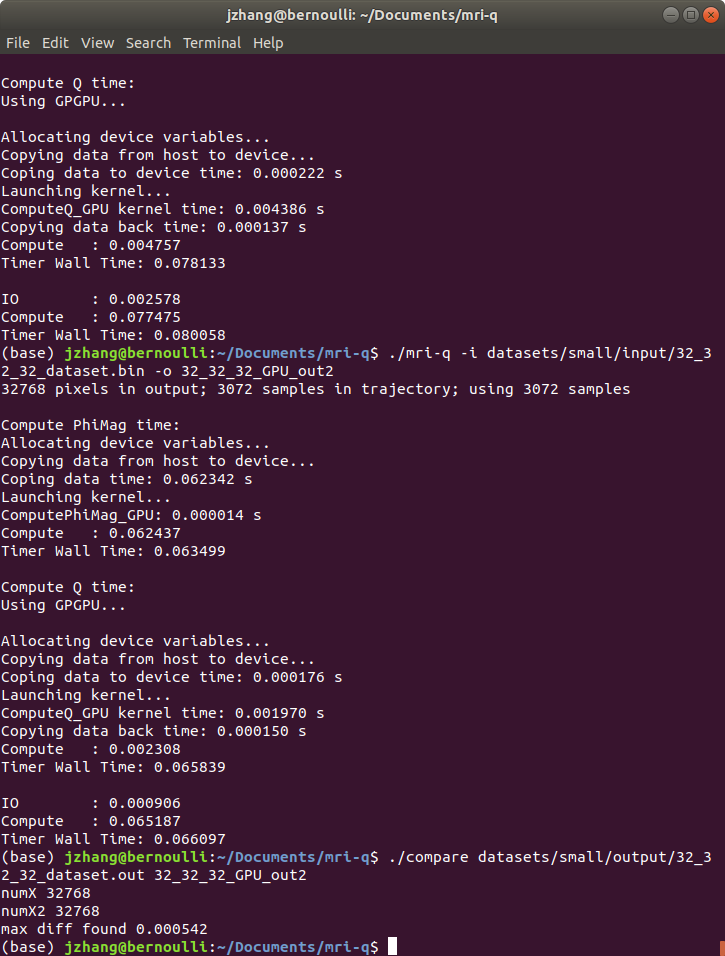


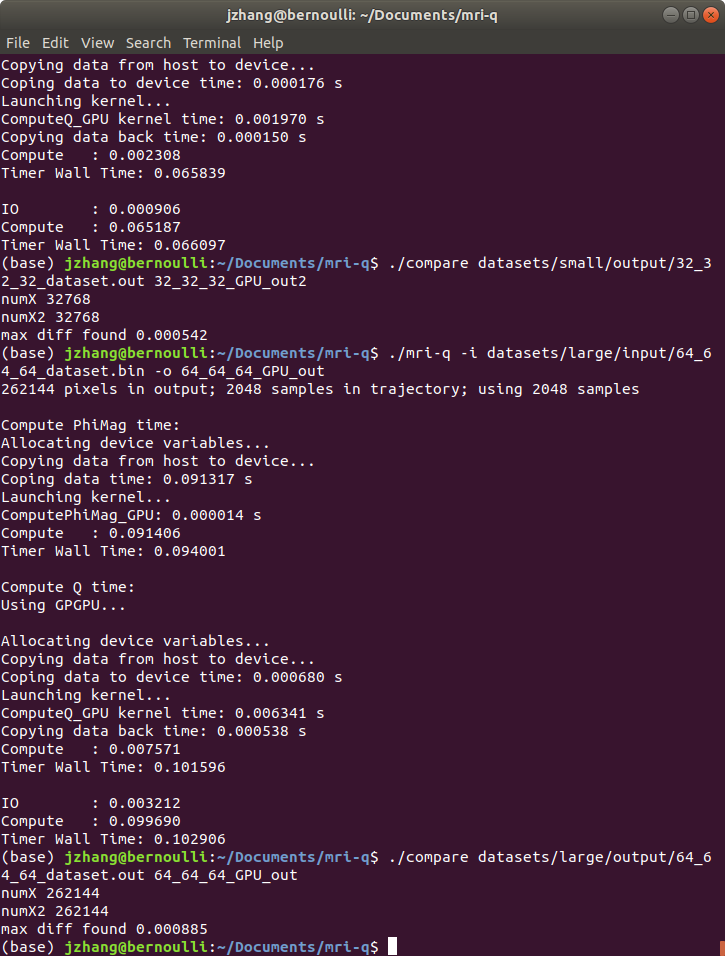
**4. Summary of Acceleration by GPU**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Dimension | CPU Serial Code | GPU + Global memory | GPU + Constant memory | GPU + Shared memory |
| 32x32x32 | 1.737 sec | 0.00265 sec | 0.0109 sec | 0.00231 sec |
| 64x64x64 | 9.266 sec | 0.00852 sec | 0.0350 sec | 0.00757 sec |
| Acceleration |  | 1087X | 265X | 1224X |

**** 32x32x32, CPU, 1.737s for computing Q

**** 64x64x64, CPU, 9.266s for computing Q.

32x32x32, GPU, shared memory, register used, 0.0023s for computing Q.

 64x64x64, GPU, shared memory, register used, 0.00757s for computing Q.