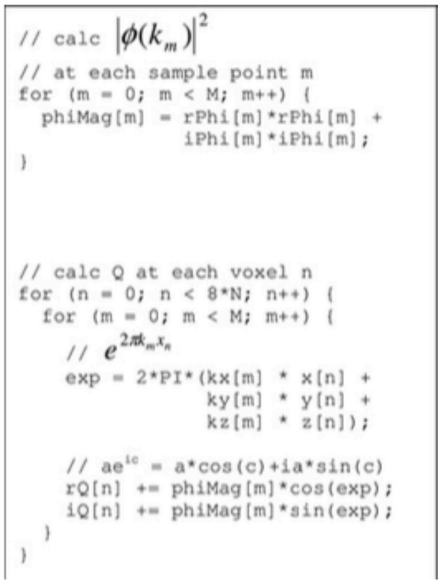
# EE 217 GPU Final Project Report

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.

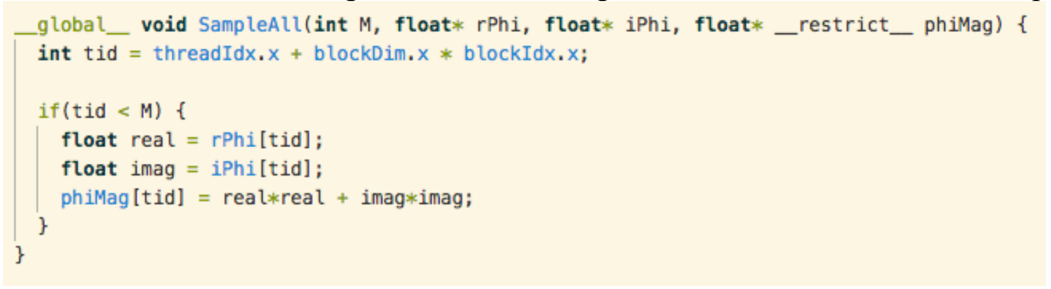


```
// calc $|\phi(k_m)|^2$
// at each sample point m
for (m = 0; m < M; m++) {
  phiMag[m] = rPhi[m]*rPhi[m] +
              iPhi[m]*iPhi[m];
}



// calc Q at each voxel n
for (n = 0; n < 8*N; n++) {
  for (m = 0; m < M; m++) {
    // $e^{2\pi k_m x_n}$
    exp = 2*PI*(kx[m] * x[n] +
                ky[m] * y[n] +
                kz[m] * z[n]);

    // $ae^{ic}$ = a*cos(c)+ia*sin(c)
    rQ[n] += phiMag[m]*cos(exp);
    iQ[n] += phiMag[m]*sin(exp);
  }
}
```

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*.

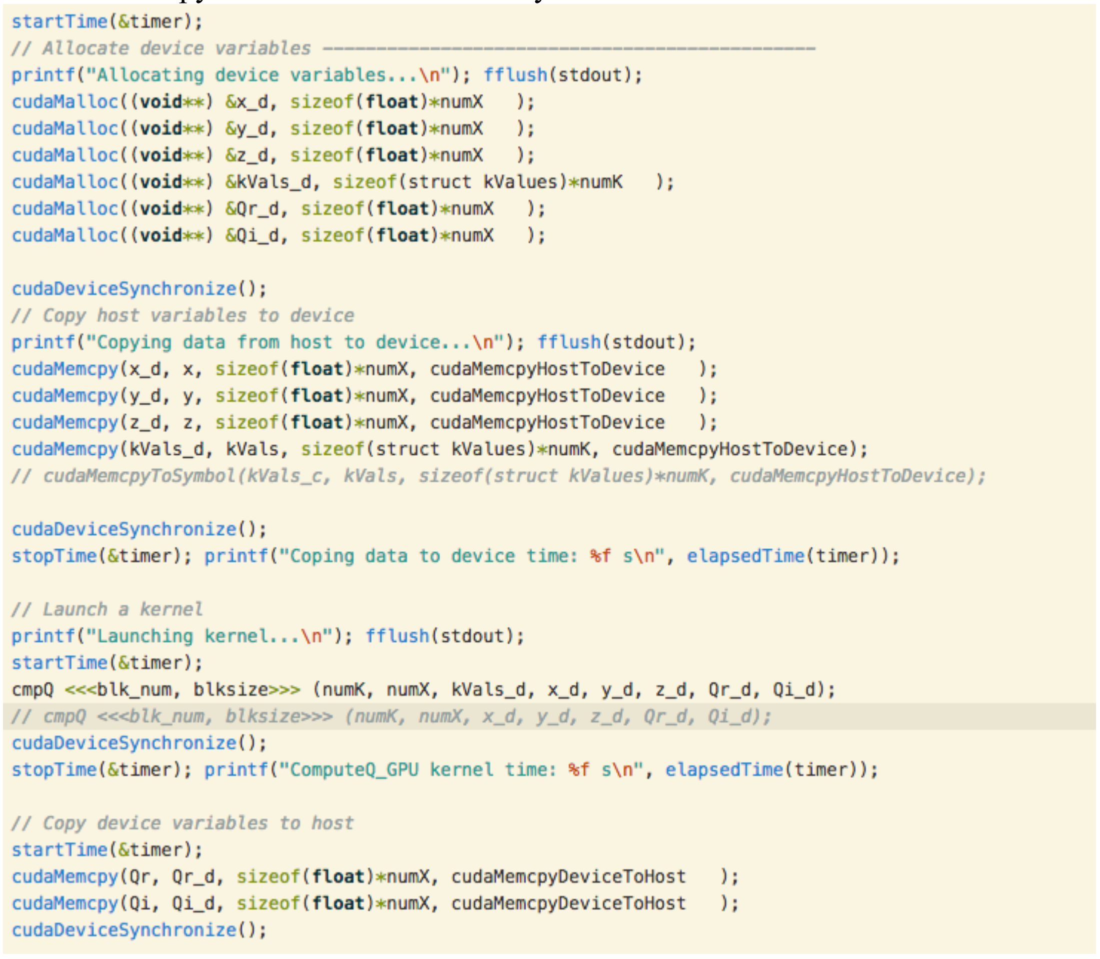


```
__global__ void SampleAll(int M, float* rPhi, float* iPhi, float* __restrict__ phiMag) {
  int tid = threadIdx.x + blockDim.x * blockIdx.x;

  if(tid < M) {
    float real = rPhi[tid];
    float imag = iPhi[tid];
    phiMag[tid] = real*real + imag*imag;
  }
}
```

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.



```
startTime(&timer);
// Allocate device variables ----------------------------------------------
printf("Allocating device variables...\n"); fflush(stdout);
cudaMalloc((void**) &x_d, sizeof(float)*numX   );
cudaMalloc((void**) &y_d, sizeof(float)*numX   );
cudaMalloc((void**) &z_d, sizeof(float)*numX   );
cudaMalloc((void**) &kVals_d, sizeof(struct kValues)*numK   );
cudaMalloc((void**) &Qr_d, sizeof(float)*numX   );
cudaMalloc((void**) &Qi_d, sizeof(float)*numX   );

cudaDeviceSynchronize();
// Copy host variables to device
printf("Copying data from host to device...\n"); fflush(stdout);
cudaMemcpy(x_d, x, sizeof(float)*numX, cudaMemcpyHostToDevice   );
cudaMemcpy(y_d, y, sizeof(float)*numX, cudaMemcpyHostToDevice   );
cudaMemcpy(z_d, z, sizeof(float)*numX, cudaMemcpyHostToDevice   );
cudaMemcpy(kVals_d, kVals, sizeof(struct kValues)*numK, cudaMemcpyHostToDevice);
// cudaMemcpyToSymbol(kVals_c, kVals, sizeof(struct kValues)*numK, cudaMemcpyHostToDevice);

cudaDeviceSynchronize();
stopTime(&timer); printf("Coping data to device time: %f s\n", elapsedTime(timer));

// Launch a kernel
printf("Launching kernel...\n"); fflush(stdout);
startTime(&timer);
cmpQ <<<blk_num, blksize>>> (numK, numX, kVals_d, x_d, y_d, z_d, Qr_d, Qi_d);
// cmpQ <<<blk_num, blksize>>> (numK, numX, x_d, y_d, z_d, Qr_d, Qi_d);
cudaDeviceSynchronize();
stopTime(&timer); printf("ComputeQ_GPU kernel time: %f s\n", elapsedTime(timer));

// Copy device variables to host
startTime(&timer);
cudaMemcpy(Qr, Qr_d, sizeof(float)*numX, cudaMemcpyDeviceToHost   );
cudaMemcpy(Qi, Qi_d, sizeof(float)*numX, cudaMemcpyDeviceToHost   );
cudaDeviceSynchronize();
```

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 $32^3$ to $128^3$), 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

```
struct kValues {
  float Kx;
  float Ky;
  float Kz;
  float PhiMag;
};

__constant__ struct kValues kVals_c[3072];
```

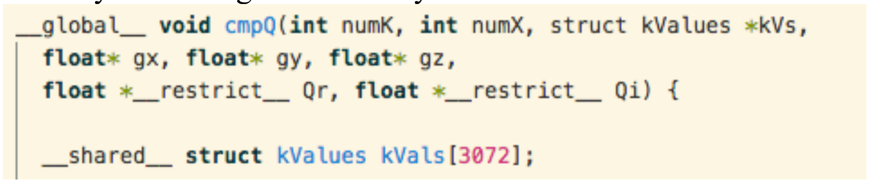
```
cudaMemcpyToSymbol(kVals_c, kVals, sizeof(struct kValues)*numK, cudaMemcpyHostToDevice);
```

```
cmpQ <<<blk_num, blksize>>> (numK, numX, x_d, y_d, z_d, Qr_d, Qi_d);
```

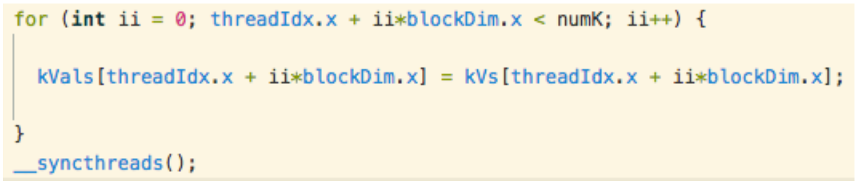
```
__global__ void cmpQ(int numK, int numX,
  float* gx, float* gy, float* gz,
  float *__restrict__ Qr, float *__restrict__ Qi) {

  // __shared__ float ds_kVals[sizeof(kVals)];

  float expArg;
  float cosArg;
  float sinArg;
  // find the index of voxel assigned to this thread
  //threadIdx.x + blockDim.x * blockIdx.x;
  int n = blockIdx.x * blockDim.x + threadIdx.x;

  // register allocate voxel inputs and outputs
  if(n < numX) {
    float x = gx[n];
    float y = gy[n];
    float z = gz[n];
    float Qracc = 0.0f;
    float Qiacc = 0.0f;
    // m is indexK
    for(int m = 0; m < numK; m++) {
      // better to store sample data kVals[] in constant memory
      expArg = PIx2 * (kVals_c[m].Kx * x +
                       kVals_c[m].Ky * y +
                       kVals_c[m].Kz * z);

      cosArg = cosf(expArg);
      sinArg = sinf(expArg);
      float phi = kVals_c[m].PhiMag;
      Qracc += phi * cosArg;
      Qiacc += phi * sinArg;

    }
    __syncthreads();
    Qr[n] = Qracc;
    Qi[n] = Qiacc;
  }
}
```

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.



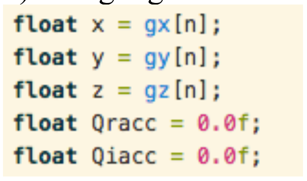
```
__global__ void cmpQ(int numK, int numX, struct kValues *kVs,
  float* gx, float* gy, float* gz,
  float *__restrict__ Qr, float *__restrict__ Qi) {

  __shared__ struct kValues kVals[3072];
```



```
for (int ii = 0; threadIdx.x + ii*blockDim.x < numK; ii++) {

  kVals[threadIdx.x + ii*blockDim.x] = kVs[threadIdx.x + ii*blockDim.x];

}
__syncthreads();
```

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

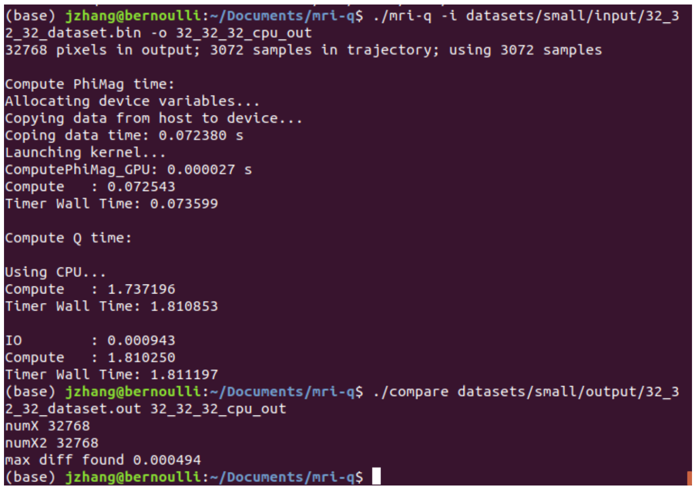
3) Using register for storing voxel space values (x, y and z) to reduce global memory accesses.



```
float x = gx[n];
float y = gy[n];
float z = gz[n];
float Qracc = 0.0f;
float Qiacc = 0.0f;
```

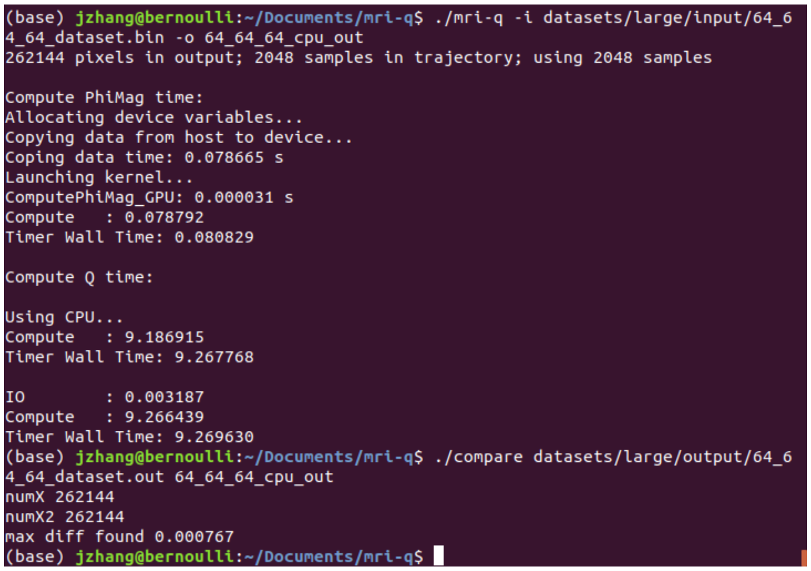
## 4. Summary of Acceleration by GPU

| 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 |



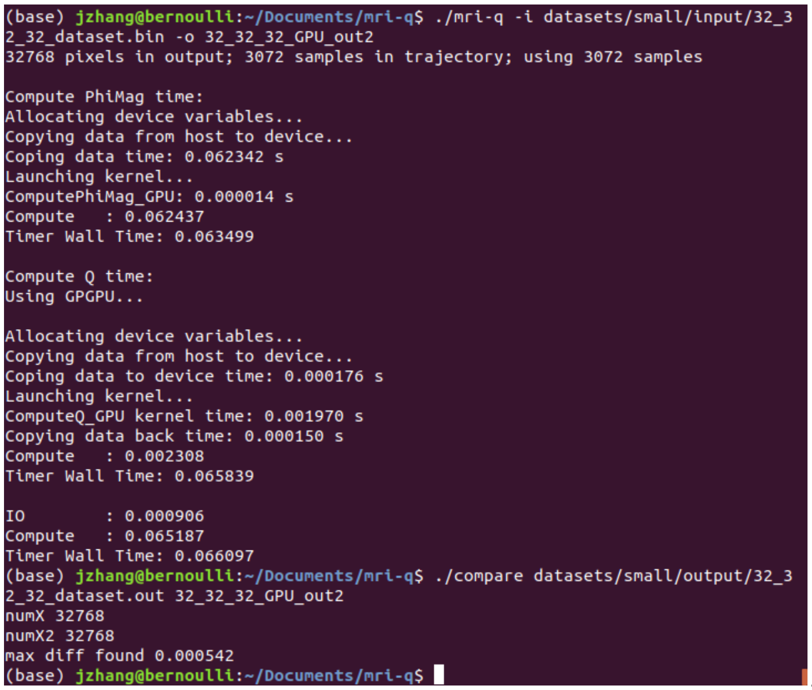
```
(base) jzhang@bernoulli:~/Documents/mri-q$ ./mri-q -i datasets/small/input/32_3
2_32_dataset.bin -o 32_32_32_cpu_out
32768 pixels in output; 3072 samples in trajectory; using 3072 samples

Compute PhiMag time:
Allocating device variables...
Copying data from host to device...
Coping data time: 0.072380 s
Launching kernel...
ComputePhiMag_GPU: 0.000027 s
Compute    : 0.072543
Timer Wall Time: 0.073599

Compute Q time:

Using CPU...
Compute    : 1.737196
Timer Wall Time: 1.810853

IO         : 0.000943
Compute    : 1.810250
Timer Wall Time: 1.811197
(base) jzhang@bernoulli:~/Documents/mri-q$ ./compare datasets/small/output/32_3
2_32_dataset.out 32_32_32_cpu_out
numX 32768
numX2 32768
max diff found 0.000494
(base) jzhang@bernoulli:~/Documents/mri-q$
```

32x32x32, CPU, 1.737s for computing Q



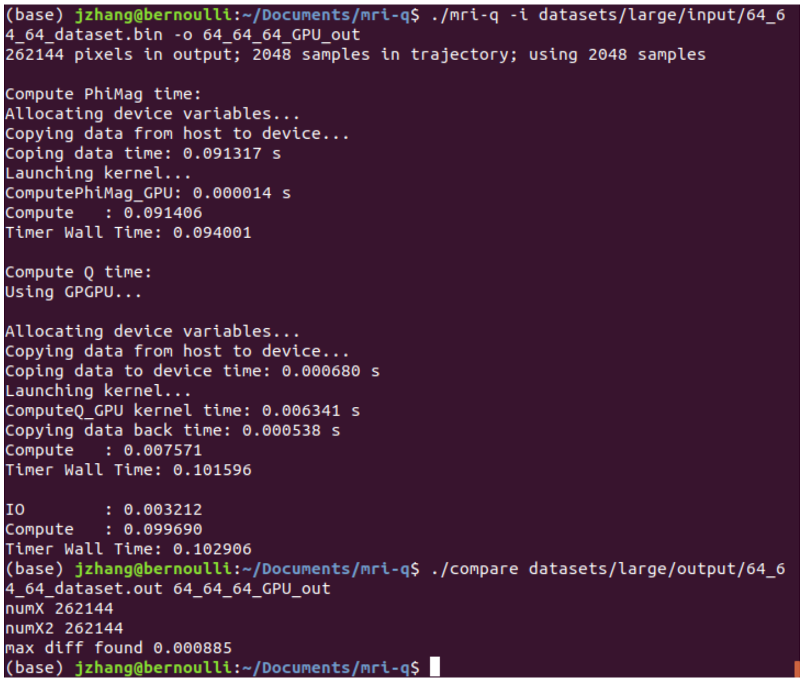
```
(base) jzhang@bernoulli:~/Documents/mri-q$ ./mri-q -i datasets/large/input/64_6
4_64_dataset.bin -o 64_64_64_cpu_out
262144 pixels in output; 2048 samples in trajectory; using 2048 samples

Compute PhiMag time:
Allocating device variables...
Copying data from host to device...
Coping data time: 0.078665 s
Launching kernel...
ComputePhiMag_GPU: 0.000031 s
Compute    : 0.078792
Timer Wall Time: 0.080829

Compute Q time:

Using CPU...
Compute    : 9.186915
Timer Wall Time: 9.267768

IO         : 0.003187
Compute    : 9.266439
Timer Wall Time: 9.269630
(base) jzhang@bernoulli:~/Documents/mri-q$ ./compare datasets/large/output/64_6
4_64_dataset.out 64_64_64_cpu_out
numX 262144
numX2 262144
max diff found 0.000767
(base) jzhang@bernoulli:~/Documents/mri-q$
```

64x64x64, CPU, 9.266s for computing Q.



```
(base) jzhang@bernoulli:~/Documents/mri-q$ ./mri-q -i datasets/small/input/32_3
2_32_dataset.bin -o 32_32_32_GPU_out2
32768 pixels in output; 3072 samples in trajectory; using 3072 samples

Compute PhiMag time:
Allocating device variables...
Copying data from host to device...
Coping data time: 0.062342 s
Launching kernel...
ComputePhiMag_GPU: 0.000014 s
Compute   : 0.062437
Timer Wall Time: 0.063499

Compute Q time:
Using GPGPU...

Allocating device variables...
Copying data from host to device...
Coping data to device time: 0.000176 s
Launching kernel...
ComputeQ_GPU kernel time: 0.001970 s
Copying data back time: 0.000150 s
Compute   : 0.002308
Timer Wall Time: 0.065839

IO        : 0.000906
Compute   : 0.065187
Timer Wall Time: 0.066097
(base) jzhang@bernoulli:~/Documents/mri-q$ ./compare datasets/small/output/32_3
2_32_dataset.out 32_32_32_GPU_out2
numX 32768
numX2 32768
max diff found 0.000542
(base) jzhang@bernoulli:~/Documents/mri-q$
```

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



```
(base) jzhang@bernoulli:~/Documents/mri-q$ ./mri-q -i datasets/large/input/64_6
4_64_dataset.bin -o 64_64_64_GPU_out
262144 pixels in output; 2048 samples in trajectory; using 2048 samples

Compute PhiMag time:
Allocating device variables...
Copying data from host to device...
Coping data time: 0.091317 s
Launching kernel...
ComputePhiMag_GPU: 0.000014 s
Compute   : 0.091406
Timer Wall Time: 0.094001

Compute Q time:
Using GPGPU...

Allocating device variables...
Copying data from host to device...
Coping data to device time: 0.000680 s
Launching kernel...
ComputeQ_GPU kernel time: 0.006341 s
Copying data back time: 0.000538 s
Compute   : 0.007571
Timer Wall Time: 0.101596

IO        : 0.003212
Compute   : 0.099690
Timer Wall Time: 0.102906
(base) jzhang@bernoulli:~/Documents/mri-q$ ./compare datasets/large/output/64_6
4_64_dataset.out 64_64_64_GPU_out
numX 262144
numX2 262144
max diff found 0.000885
(base) jzhang@bernoulli:~/Documents/mri-q$
```

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