Programming Assignment 3 CSE 5441 Bobo Shi

1.

First attempted was put it simply with CUDA to get parallel GPU performance, assigning each thread to calculate each element.

Next, I optimized it by using shared memory. I use square block and share both A and B in block (as it used in slides "introduction to GPU programming" Page 26). I call it ShMem-SquareBlock.

Then I use shared memory by use a line block (like dimBlock(64, 1)), I call it ShMem-LineBlock.

Finally, I did the unrolling i for shared-LineBlock, I call it ShMem-LineBlock-unroll-i.

Base GFLOPs is 0.2 GFLOPs

simple kernel with dimBlock(x,y)	GFLOPs
dimBlock (4,4)	9.1
dimBlock (4,8)	15.8
dimBlock (4,16)	11.3
dimBlock (8,8)	28.8
dimBlock (8,32)	25.9
dimBlock (16,16)	39.9
dimBlock (16,32)	42.8
dimBlock (32,2)	39.7
dimBlock (32,4)	48.7
dimBlock (32,8)	41.6
dimBlock (32,16)	45.3
dimBlock (32,32)	51.5

ShMem-SquareBlock	GFLOPs
ShMem-SB(4,4)	6.3
ShMem-SB (8,8)	34.3
ShMem-SB (16,16)	58.9
ShMem-SB (32,32)	61
(ShMem-SB 64,64)	519.0

ShMem-LineBlock	GFLOPs
ShMem-LB-16	8.8
ShMem-LB-32	18.7

ShMem-LB-64	29.1
ShMem-LB-128	26.5
ShMem-LB-256	24.3
ShMem-LB-512	25.4
ShMem-LB-1024	30.3
ShMem-LB-2048	519.1

ShMem-LineBlock-unroll-i	GFLOPs
ShMem-LB-unroll-i-16	15.3
ShMem-LB-unroll-i-32	32.9
ShMem-LB-unroll-i-64	52.2
ShMem-LB-unroll-i-128	51.6
ShMem-LB-unroll-i-256	47.5
ShMem-LB-unroll-i-512	49.1
ShMem-LB-unroll-i-1024	56.2
ShMem-LB-unroll-i-2048	519.9

You may noticed that I got the very weird and unbelievable value 519.0, 519.1, 519,9. For all those values, the number of threads per block is more than 1024, which is the limit for CUDA. I don't know why they can get the correct value by exceeding the limits of the CUDA and get crazy good performance. I think there is some trick thing there like the Cache thing. Besides those crazy number, the best performance is 61 GFLOPs. The code is:

ShMem-SquareBlock-(32,32):

```
#define bSize (32)
#define bSizeX (bSize)
#define bSizeY (bSize)
#define bNumX (N/bSizeX)
#define bNumY (N/bSizeY)

__global__ void p1_kernel(double *A, double *B, double *C) {
   int bx=blockIdx.x; int by=blockIdx.y;
   int tx=threadIdx.x; int ty=threadIdx.y;

   int i = bSizeY*by+ty;
   int j = bSizeX*bx+tx;

   //int aBegin=N*bSizeY*by;
   //int bBegin=bSizeX*bx;
```

```
    __shared__ double As[bSizeX][bSizeY];
    __shared__ double Bs[bSizeX][bSizeY];

    int alnd=i*N+tx;
    int blnd=j+ty*N;

    double Bsub=B[i*N+j];

   for (int kt = 0; kt < N; kt+=bSizeX){

      As[ty][tx]=A[alnd];
      Bs[ty][tx]=B[blnd];
      __syncthreads();
        for (int k= (i>kt ? i : kt); k< kt+ bSizeX; ++k)
           Bsub+=As[ty][k-kt]*Bs[k-kt][tx];
      __syncthreads();
       alnd+=bSizeX;
       blnd+=bSizeY*N;
     }
    C[i*N+j]=Bsub;
}
```

And I also want to put ShMem-LineBlock-unroll-i here because it alse gives good performance.

ShMem-LineBlock-unroll-i:

```
__global__ void p1_kernel(double *A, double *B, double *C) {
    int bx=blockIdx.x; int by=blockIdx.y;
    int tx=threadIdx.x;// int ty=threadIdx.y;

    int i = by*2;
    int j = bSizeX*bx+tx;

    __shared__ double As0[bSizeX],As1[bSizeX];

    double Bsub0=B[i*N+j];
        double Bsub1=B[(i+1)*N+j];
        Bsub0 += A[i*N+i]*B[i*N+j];
        for(int kt=0; kt<N; kt+=bSizeX){
                As0[tx]=A[kt+tx+N*i];
                As1[tx]=A[kt+tx+N*(i+1)];
                __syncthreads();
                for(int k=(i>=kt ? i+1 : kt); k<kt+bSizeX; k++){
```

```
                double Br=B[k*N+j];
                Bsub0 += As0[k-kt]*Br;
                Bsub1 += As1[k-kt]*Br;
            }
            __syncthreads();
        }
    C[i*N+j]=Bsub0;
    C[(i+1)*N+j]=Bsub1;
}
```

2.

First I attempted to put simple kernel program, assigning each thread to calculate each element and then test different block size.

Then I use shared memory by use a line block (like dimBlock(4, 1)), I call it ShMem-LineBlock.

Unroll is wrote but never worked. No time to debug that.

simple	GFLOPs
dimBlock(2,2)	4.6
dimBlock (2,4)	5.1
dimBlock (2,8)	6.3
dimBlock (2,16)	6.4
dimBlock (4,4)	6.9
dimBlock (4,8)	9.3
dimBlock (4,16)	9.1
dimBlock (4,32)	8.8
dimBlock (8,4)	8
dimBlock (8,8)	7.6
dimBlock (8,16)	7.7
dimBlock (8,32)	7.2
dimBlock (8,64)	7.8
dimBlock (16,4)	5.1
dimBlock (16,8)	4.9
dimBlock (16,16)	4.7
dimBlock (16,32)	4.5
dimBlock (32,1)	3.5
dimBlock (32,2)	2.4
dimBlock (32,4)	2.2
dimBlock (32,8)	2.3
dimBlock (32,16)	2.4

ShMem-LineBlock	GFLOPs
ShMem-LB-4	2.7
ShMem-LB-8	3.1
ShMem-LB-16	1.6

The best performance for this problem is 9.3 GFLOPs by using simple kernel with dimBlock (4,8)
Codes for simple kernel:

```
#define bSizex (4)
#define bSizey (8)
#define bNumx (N/bSizex)
#define bNumy (N/bSizey)

__global__ void p1_kernel(double *A, double *B, double *C) {
  int bx=blockIdx.x; int by=blockIdx.y;
  int tx=threadIdx.x; int ty=threadIdx.y;
  int bSizeX= bSizex;
  int bSizeY= bSizey;

  int k = bSizeY*by+ty;
  int l = bSizeX*bx+tx;
  int N2=N*N;

  if(k<l){
    double Csub=C[k*N+l];
    for (int i=0; i<N; i++)
      for (int j=0; j<N; j++)
        Csub+=0.5*(A[l*N2+i*N+j]*B[k*N2+i*N+j]+A[k*N2+i*N+j]*B[l*N2+i*N+j]);
    C[k*N+l]=Csub;
  }
}
```

code for ShMem-LineBlock

```
__global__ void p1_kernel(double *A, double *B, double *C) {
  int bx=blockIdx.x; int by=blockIdx.y;
  int tx=threadIdx.x;

  int k = by;
  int l = bSizeX*bx+tx;
  int N2=N*N;
```

```
    __shared__ double As[bSizeX][N];
    __shared__ double Bs[bSizeX][N];

    double Csub = C[k*N+l];
    for (int is=0; is<N; is+= bSizeX){
        for (int j=0; j<N; j++){
            As[tx][j] = A[k*N2+(is+tx)*N+j];
            Bs[tx][j] = B[k*N2+(is+tx)*N+j];
        }
        __syncthreads();
        if(k<l){
            for (int i=is; i<is+bSizeX; i++)
                for (int j=0; j<N; j++)
                    Csub+=0.5*(As[i-is][j]*B[l*N2+i*N+j]+A[l*N2+i*N+j]*Bs[i-is][j]);
        }
    __syncthreads();
    }
    C[k*N+l]=Csub;
}
```