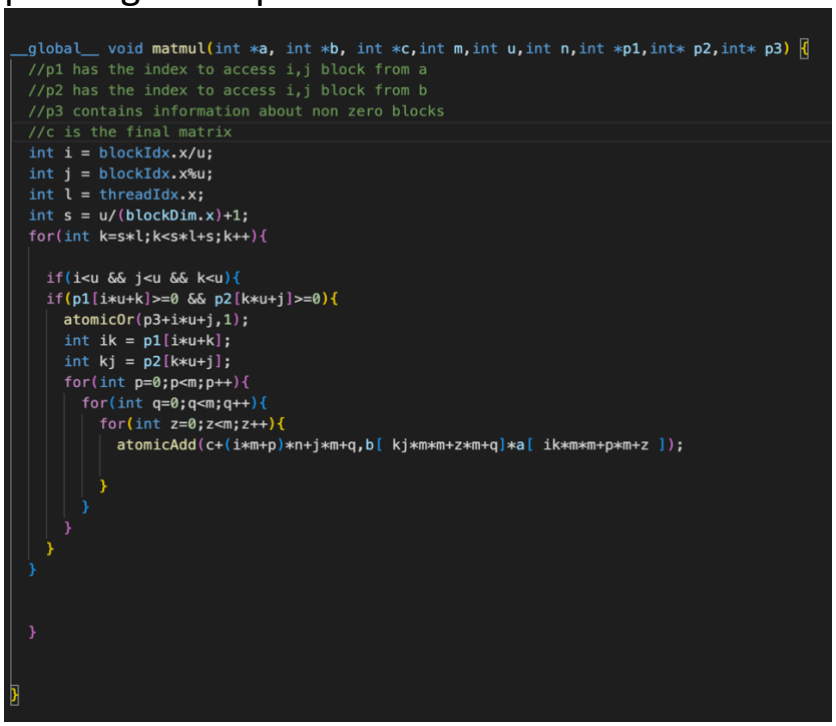
Assignment 4 Report

By Khyateeswar Naidu Nalla
2019CS10376

Approach 1:

I first stored the input matrices in the form of blocks i.e had an array which contains the nonzero blocks continuously and every block here is indexed by another array. Now I sent these 4 matrices to gpu, here the multiplication is done such that each block calculates the final block of matrix so n/m*n/m blocks are used for computation and each thread in that block distributes the multiplication into nonzero matrix block multiplication which adds to that block. And then we send the output matrix to cpu and we output to the output file.I used OPENMP for speeding up the output pushing to output file.



```
__global__ void matmul(int *a, int *b, int *c,int m,int u,int n,int *p1,int* p2,int* p3) {
  //p1 has the index to access i,j block from a
  //p2 has the index to access i,j block from b
  //p3 contains information about non zero blocks
  //c is the final matrix
  int i = blockIdx.x/u;
  int j = blockIdx.x%u;
  int l = threadIdx.x;
  int s = u/(blockDim.x)+1;
  for(int k=s*l;k<s*l+s;k++){

    if(i<u && j<u && k<u){
    if(p1[i*u+k]>=0 && p2[k*u+j]>=0){
      atomicOr(p3+i*u+j,1);
      int ik = p1[i*u+k];
      int kj = p2[k*u+j];
      for(int p=0;p<m;p++){
        for(int q=0;q<m;q++){
          for(int z=0;z<m;z++){
            atomicAdd(c+(i*m+p)*n+j*m+q,b[ kj*m*m+z*m+q]*a[ ik*m*m+p*m+z ]);

          }
        }
      }
    }
  }


  }

}
```

Approach 2:

I made some changes to the above approach.Instead of the sending the whole output matrix from gpu ram to cpu ram. I used shared memory to calculate the final block corresponding that block and then converted the non zero blocks to char array and then send the char array to cpu ram where we can directly output it.This is helps in reducing output time.

```
__global__ void matmul(int *a, int *b,char* cf,int m,int u,int n,int *p1,int* p2,int *nz,int* fnz) {
//p1 has the index to access i,j block from a
//p2 has the index to access i,j block from b
// nz contains the no of nonzero blocks
//cf is the char array to send to output

__shared__ int c[64];// array for storing the block computed by this block
for(int h=0;h<64;h++){
  c[h]=0;
}
__shared__ int p3;
p3=0;

__syncthreads();

int i = blockIdx.x/u;
int j = blockIdx.x%u;
int l = threadIdx.x;
int s = u/(blockDim.x)+1;

for(int k=s*l;k<s*l+s;k++){

  if(i<u && j<u && k<u){

  if(p1[i*u+k]>=0 && p2[k*u+j]>=0){
    atomicOr(&p3,1);
    int ik = p1[i*u+k];
    int kj = p2[k*u+j];
    for(int p=0;p<m;p++){
      for(int q=0;q<m;q++){
        for(int z=0;z<m;z++){

          atomicAdd(  &c[p*m+q],b[ kj*m*m+z*m+q]*a[ ik*m*m+p*m+z ]);

        }
      }
    }
  }


}
}

__syncthreads();


  if(l==0 && i<u && j<u){
      if( p3>0 ){
          p3 = atomicAdd(nz,1);
          int off = (p3-1)*(4*m*m+8)+12;
          conv(i,cf,off);
          conv(j,cf,off+4);
          int sd=0;
          atomicAdd(fnz,1);
          for(int a=0;a<m;a++){
              for(int b=0;b<m;b++){
                  conv(c[a*m+b],cf,off+8+sd);
                  sd=sd+4;
              }
          }
      }
      if(i==0 && j==0){
        conv(n,cf,0);
        conv(m,cf,4);

      }
  }


}
```

N=30000
M=8
Varying k

K(No of nonzero blocks of each matrix)	Time taken(s)
20000	6
50000	46
100000	61
500000	90
1000000	