SANSKAAR PATNI
CSE C 23 180905134
PP LAB 6
Programs on Matrix using CUDA

1a. Matrix Addition- Each Thread computes a row of output matrix

CODE

```
%%cu
#include <stdio.h>
#include <stdlib.h>
__global__ void rowAdd(int *A, int *B, int *C,int w) {
int rowId=threadIdx.x;
int colId;
for(colId=0;colId<w;colId++){
   C[rowId * w +colId]= A[rowId * w +colId] +B [rowId * w +colId];
}
}
int main() {
// Matrices should have the same dimensions for addition
int h=2;
int w=3;

int *d_A, *d_B, *d_C;

int size= h * w * sizeof(int);

cudaMalloc((void **)&d_A, size);
cudaMalloc((void **)&d_B, size);
cudaMalloc((void **)&d_C, size);
 int A[h][w] = {
   {1,2,5},
   {3,4,6}
    };
int B[h][w] = {
   {1,2,3},
   {4,5,6}
   };
int C[h][w] = {
   {0,0,0},
   {0,0,0}
   };
```

```
// Copy inputs to device
cudaMemcpy(d_A, &A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, &B, size, cudaMemcpyHostToDevice);

// Launch add() kernel on GPU
rowAdd<<<1,h>>>(d_A, d_B, d_C,w);

// Copy result back to host
cudaError err = cudaMemcpy(&C, d_C, size, cudaMemcpyDeviceToHost);
 if(err!=cudaSuccess) {
     printf("CUDA error copying to Host: %s\n",
cudaGetErrorString(err));
 }
printf("1a. Matrix Addition Each Thread computes a row of output
matrix\n");
printf("Resultant matrix C after addition is\n");
for(int k=0;k<h;k++){
   for(int f=0;f<w;f++){
       printf("%d\t",C[k][f]);
   }
   printf("\n");
}
// Cleanup
cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
return 0;
}
```

SCREENSHOT

```
1a. Matrix Addition Each Thread computes a row of output matrix
Resultant matrix C after addition is
2       4       8
7       9       12
```

```
[12]  1 %%cu
```

1b. Matrix Addition- Each Thread computes a column of output matrix

CODE

```
%%cu
#include <stdio.h>
```

```
#include <stdlib.h>
__global__ void colAdd(int *A, int *B, int *C,int h) {
 int colId=threadIdx.x;
 int w=blockDim.x;
 int rowId=0;
 for(rowId=0;rowId<h;rowId++){
     C[rowId * w + colId]= A[rowId*w+ colId] + B[rowId*w + colId];
 }
}
int main(){
// Matrices should have the same dimensions for addition
int h=2;
int w=3;

int *d_A, *d_B, *d_C;

int size= h * w * sizeof(int);

cudaMalloc((void **)&d_A, size);
cudaMalloc((void **)&d_B, size);
cudaMalloc((void **)&d_C, size);
 int A[h][w] = {
   {1,2,5},
   {3,4,6}
    };
int B[h][w] = {
   {1,2,3},
   {4,5,6}
   };
int C[h][w] = {
   {0,0,0},
   {0,0,0}
   };

// Copy inputs to device
cudaMemcpy(d_A, &A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, &B, size, cudaMemcpyHostToDevice);

colAdd<<<1,w>>>(d_A, d_B, d_C,h);

// Copy result back to host
cudaError err = cudaMemcpy(&C, d_C, size, cudaMemcpyDeviceToHost);
 if(err!=cudaSuccess) {
```

```
     printf("CUDA error copying to Host: %s\n",
cudaGetErrorString(err));
 }
printf("1b. Matrix Addition Each Thread computes a column of output
matrix\n");
printf("Resultant matrix C after addition is\n");
for(int k=0;k<h;k++){
   for(int f=0;f<w;f++){
       printf("%d\t",C[k][f]);
   }
   printf("\n");
}
// Cleanup
cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
return 0;
}
```

SCREENSHOT

```
1b. Matrix Addition Each Thread computes a column of output matrix
Resultant matrix C after addition is
2       4       8
7       9       12
```

1c. Matrix Addition- Each Thread computes an element of output matrix

CODE

```
%%cu
#include <stdio.h>
#include <stdlib.h>
__global__ void eleAdd(int *A, int *B, int *C) {
   int rowId=threadIdx.y;
   int colId=threadIdx.x;
   int w=blockDim.x;
   C[rowId * w + colId]= A[rowId * w + colId] + B[rowId * w + colId];
}
int main() {
 int h=2;
 int w=3;
 int *d_A, *d_B, *d_C;
```

```
 int size = h*w*sizeof(int);

 cudaMalloc((void **)&d_A, size);
 cudaMalloc((void **)&d_B, size);
 cudaMalloc((void **)&d_C, size);

 // Setup input values
 int A[h][w] = {
     {1,2,5},
     {3,4,6}
     };
 int B[h][w] = {
     {1,2,3},
     {4,5,6},
     };
 int C[h][w] = {
     {0,0,0},
     {0,0,0}
     };
 // Copy inputs to device
 cudaMemcpy(d_A, &A, size, cudaMemcpyHostToDevice);
 cudaMemcpy(d_B, &B, size, cudaMemcpyHostToDevice);

 dim3 threadstr(w,h);
 dim3 numBlocks(1,1);

 eleAdd<<<numBlocks,threadstr>>>(d_A, d_B, d_C);

 // Copy result back to host
 cudaError err = cudaMemcpy(&C, d_C, size, cudaMemcpyDeviceToHost);
   if(err!=cudaSuccess) {
       printf("CUDA error copying to Host: %s\n",
cudaGetErrorString(err));
   }
 printf("1c. Matrix Addition Each Thread computes an element of output
matrix\n");
 printf("Resultant matrix C after addition is\n");
 for(int k=0;k<h;k++){
     for(int f=0;f<w;f++){
         printf("%d\t",C[k][f]);
     }
     printf("\n");
 }
```

```
 // Cleanup
 cudaFree(d_A);
 cudaFree(d_B);
 cudaFree(d_C);
 return 0;
}
```

SCREENSHOT

```
1c. Matrix Addition Each Thread computes an element of output matrix
Resultant matrix C after addition is
2       4       8
7       9       12
```

```
[ ]   1 %%cu
```

2a. Matrix Multiplication- Each Thread computes a row of output matrix

CODE

```
%%cu
#include <stdio.h>
#include <stdlib.h>
__global__ void rowMul(int *A, int *B, int *C,int wb,int wa) {
int rowIdA=threadIdx.x;
int i,sum,colIdB;
for(colIdB=0;colIdB<wb;colIdB++){
   sum=0;
   for(i=0;i<wa;i++){
       sum+=A[rowIdA*wa+i]*B[i*wb+colIdB];
   }
   C[rowIdA * wb +colIdB]=sum;
}
}
int main() {
int ha=2;
int wa=3;
int hb=3;
int wb=3;

// device copies of variables A, B & C
int *d_A, *d_B, *d_C;

int size = sizeof(int);
// Allocate space for device copies of A, B, C
```

```
int sizeA= ha*wa*size;
int sizeB= hb*wb*size;
int sizeC= ha*wb*size;

cudaMalloc((void **)&d_A, sizeA);
cudaMalloc((void **)&d_B, sizeB);
cudaMalloc((void **)&d_C, sizeC);
// Setup input values
int A[ha][wa] = {
   {1,2,5},
   {3,4,6}
    };
int B[hb][wb] = {
   {1,2,3},
   {4,5,6},
   {7,8,9}
   };
int C[ha][wb] = {
   {0,0,0},
   {0,0,0}
   };
// Copy inputs to device
cudaMemcpy(d_A, &A, sizeA, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, &B, sizeB, cudaMemcpyHostToDevice);

rowMul<<<1,ha>>>(d_A, d_B, d_C,wb,wa);
// Copy result back to host
cudaError err = cudaMemcpy(&C, d_C, sizeC, cudaMemcpyDeviceToHost);
 if(err!=cudaSuccess) {
     printf("CUDA error copying to Host: %s\n",
cudaGetErrorString(err));
 }
printf("2a. Matrix Mutiplication Each Thread computes a row of output
matrix\n");
printf("Resultant matrix C after multiplication is\n");
for(int k=0;k<ha;k++){
   for(int f=0;f<wb;f++){
       printf("%d\t",C[k][f]);
   }
   printf("\n");
}
// Cleanup
cudaFree(d_A);
```

```
cudaFree(d_B);
cudaFree(d_C);
return 0;
}
```

SCREENSHOT

```
2a. Matrix Mutiplication Each Thread computes a row of output matrix
Resultant matrix C after multiplication is
44      52      60
61      74      87
```

```
1 %%cu
2 #include <stdio.h>
```

2b. Matrix Multiplication- Each Thread computes a column of output matrix

CODE

```
%%cu
#include <stdio.h>
#include <stdlib.h>
__global__ void colMul(int *A, int *B, int *C,int ha,int wa) {
int colIdB=threadIdx.x;
int j,sum,rowIdA;
int wb=blockDim.x;
for(rowIdA=0;rowIdA<ha;rowIdA++){
   sum=0;
   for(j=0;j<wa;j++){
       sum+=A[rowIdA*wa+j]*B[j*wb+colIdB];
   }
   C[rowIdA * wb +colIdB]=sum;
}
}
int main() {
int ha=2;
int wa=3;
int hb=3;
int wb=3;

// device copies of variables A, B & C
int *d_A, *d_B, *d_C;

int size = sizeof(int);
// Allocate space for device copies of A, B, C
```

```
int sizeA= ha*wa*size;
int sizeB= hb*wb*size;
int sizeC= ha*wb*size;

cudaMalloc((void **)&d_A, sizeA);
cudaMalloc((void **)&d_B, sizeB);
cudaMalloc((void **)&d_C, sizeC);
// Setup input values
int A[ha][wa] = {
   {1,2,5},
   {3,4,6}
    };
int B[hb][wb] = {
   {1,2,3},
   {4,5,6},
   {7,8,9}
   };
int C[ha][wb] = {
   {0,0,0},
   {0,0,0}
   };
// Copy inputs to device
cudaMemcpy(d_A, &A, sizeA, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, &B, sizeB, cudaMemcpyHostToDevice);

colMul<<<1,wb>>>(d_A, d_B, d_C,ha,wa);
// Copy result back to host
cudaError err = cudaMemcpy(&C, d_C, sizeC, cudaMemcpyDeviceToHost);
 if(err!=cudaSuccess) {
     printf("CUDA error copying to Host: %s\n",
cudaGetErrorString(err));
 }
printf("2b. Matrix Mutiplication Each Thread computes a column of
output matrix\n");
printf("Resultant matrix C after multiplication is\n");
for(int k=0;k<ha;k++){
   for(int f=0;f<wb;f++){
       printf("%d\t",C[k][f]);
   }
   printf("\n");
}
// Cleanup
cudaFree(d_A);
```

```
cudaFree(d_B);
cudaFree(d_C);
return 0;
}
```

SCREENSHOT

```
70 return 0;
71 }
```

```
2b. Matrix Mutiplication Each Thread computes a column of output matrix
Resultant matrix C after multiplication is
44      52      60
61      74      87
```

2c. Matrix Multiplication- Each Thread computes an element of output matrix

CODE

```
%%cu
#include <stdio.h>
#include <stdlib.h>
__global__ void eleMul(int *A, int *B, int *C,int wa) {
   int rowIdA=threadIdx.y;
   int colIdB=threadIdx.x;
   int wb=blockDim.x;
   int i,sum=0;
   for(i=0;i<wa;i++){
       sum+=A[rowIdA*wa+i]*B[i*wb+colIdB];
   }
   C[rowIdA * wb +colIdB]=sum;
}
int main() {
int ha=2;
int wa=3;
int hb=3;
int wb=3;

int *d_A, *d_B, *d_C;

int size = sizeof(int);
// Allocate space for device copies of A, B, C
int sizeA= ha*wa*size;
int sizeB= hb*wb*size;
int sizeC= ha*wb*size;

cudaMalloc((void **)&d_A, sizeA);
```

```
cudaMalloc((void **)&d_B, sizeB);
cudaMalloc((void **)&d_C, sizeC);

// Setup input values
int A[ha][wa] = {
   {1,2,5},
   {3,4,6}
    };
int B[hb][wb] = {
   {1,2,3},
   {4,5,6},
   {7,8,9}
   };
int C[ha][wb] = {
   {0,0,0},
   {0,0,0}
   };
// Copy inputs to device
cudaMemcpy(d_A, &A, sizeA, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, &B, sizeB, cudaMemcpyHostToDevice);

dim3 threadstr(wb,ha);
dim3 numBlocks(1,1);

eleMul<<<numBlocks,threadstr>>>(d_A, d_B, d_C,wa);
// Copy result back to host
cudaError err = cudaMemcpy(&C, d_C, sizeC, cudaMemcpyDeviceToHost);
 if(err!=cudaSuccess) {
     printf("CUDA error copying to Host: %s\n",
cudaGetErrorString(err));
 }
printf("2c. Matrix Mutiplication Each Thread computes an element of
output matrix\n");
printf("Resultant matrix C after multiplication is\n");
for(int k=0;k<ha;k++){
   for(int f=0;f<wb;f++){
       printf("%d\t",C[k][f]);
   }
   printf("\n");
}
// Cleanup
cudaFree(d_A);
cudaFree(d_B);
```

```
cudaFree(d_C);
return 0;
}
```

SCREENSHOT

```
2c. Matrix Mutiplication Each Thread computes an element of output matrix
Resultant matrix C after multiplication is
44      52      60
61      74      87
```