Practical No. 8

Study and Implementation of Grids and Blocks of 2 and 3 dimensions in CUDA C, GPU Memory and Profiling

Name: Shubham Mali

PRN: 2018BTECS00007

Batch: B4

%%writefile cuda11.cu #include<stdio.h> #include<cuda.h>

#define row1 2 /\* Number of rows of first matrix \*/ #define col1 3 /\* Number of columns of first matrix \*/ #define row2 3 /\* Number of rows of second matrix \*/ #define col2 2 /\* Number of columns of second matrix \*/

global void matproduct(int \*l,int \*m, int \*n)

{

int x=blockIdx.x; int y=blockIdx.y; int k;

n[col2\*y+x]=0; for(k=0;k<col1;k++)

{

n[col2\*y+x]=n[col2\*y+x]+l[col1\*y+k]\*m[col2\*k+x];

}

}

int main()

{

int a[row1][col1]={{1,2,3},{4,5,6}};

int b[row2][col2]={{1,2},{3,4},{5,6}};

int c[row1][col2]; int \*d,\*e,\*f;

int i,j;

printf("\n Enter elements of first matrix of size 2\*3\n");

cudaMalloc((void \*\*)&d,row1\*col1\*sizeof(int)); cudaMalloc((void \*\*)&e,row2\*col2\*sizeof(int)); cudaMalloc((void \*\*)&f,row1\*col2\*sizeof(int));

cudaMemcpy(d,a,row1\*col1\*sizeof(int),cudaMemcpyHostToDevice); cudaMemcpy(e,b,row2\*col2\*sizeof(int),cudaMemcpyHostToDevice);

dim3 grid(col2,row1);

/\* Here we are defining two dimensional Grid(collection of blocks) structure. Syntax is dim3 grid(no. of columns,no. of rows) \*/

matproduct<<<grid,1>>>(d,e,f);

cudaMemcpy(c,f,row1\*col2\*sizeof(int),cudaMemcpyDeviceToHost); printf("\nProduct of two matrices:\n ");

for(i=0;i<row1;i++)

{

for(j=0;j<col2;j++)

{

printf("%d\t",c[i][j]);

}

printf("\n");

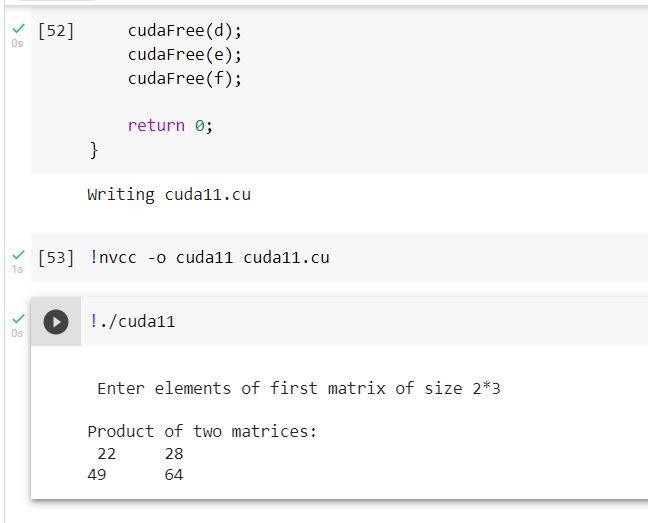
}

cudaFree(d); cudaFree(e);

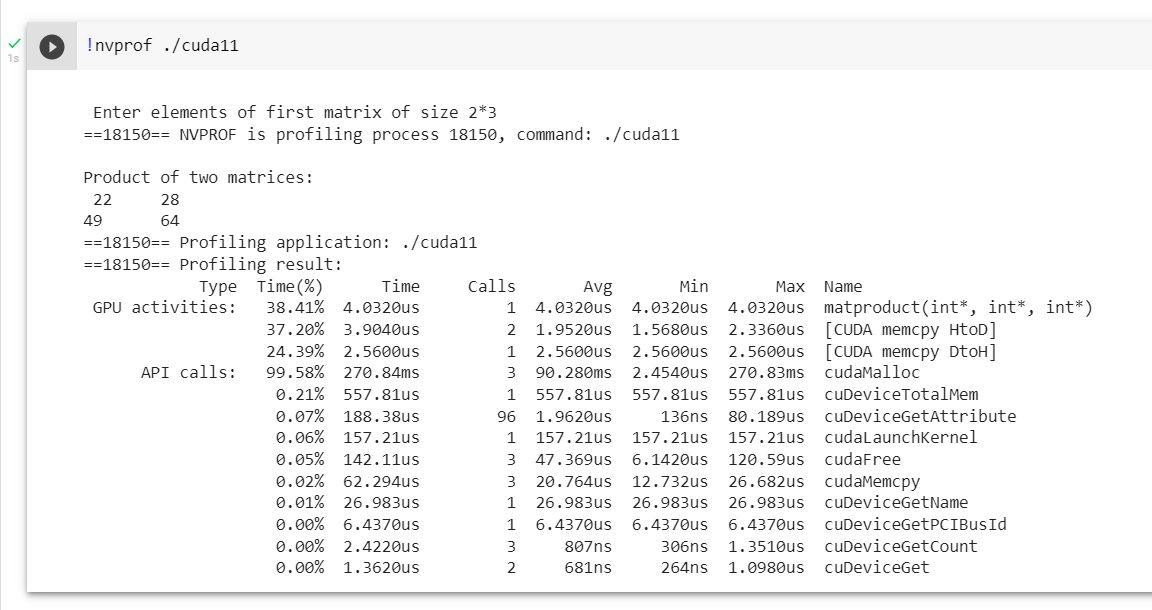
cudaFree(f);

return 0;

}



# After profiling:



**Optimized code:**

%%writefile cuda5.cu #include<stdio.h> #include<cuda.h>

#define row1 2 /\* Number of rows of first matrix \*/ #define col1 3 /\* Number of columns of first matrix \*/ #define row2 3 /\* Number of rows of second matrix \*/ #define col2 2 /\* Number of columns of second matrix \*/

global void matproductsharedmemory(int \*l,int \*m, int \*n)

{

int x=blockIdx.x; int y=blockIdx.y;

shared int p[col1];

int i;

int k=threadIdx.x; n[col2\*y+x]=0;

p[k]=l[col1\*y+k]\*m[col2\*k+x];

syncthreads(); for(i=0;i<col1;i++) n[col2\*y+x]=n[col2\*y+x]+p[i];

}

int main()

{

int a[row1][col1]={1,2,3,4,5,6};

int b[row2][col2]={1,2,3,4,5,6}; int c[row1][col2];

int \*d,\*e,\*f; int i,j;

cudaMalloc((void \*\*)&d,row1\*col1\*sizeof(int)); cudaMalloc((void \*\*)&e,row2\*col2\*sizeof(int)); cudaMalloc((void \*\*)&f,row1\*col2\*sizeof(int));

cudaMemcpy(d,a,row1\*col1\*sizeof(int),cudaMemcpyHostToDevice); cudaMemcpy(e,b,row2\*col2\*sizeof(int),cudaMemcpyHostToDevice);

dim3 grid(col2,row1);

/\* Here we are defining two dimensional Grid(collection of blocks) structure. Syntax is dim3 grid(no. of columns,no. of rows) \*/

matproductsharedmemory<<<grid,col1>>>(d,e,f); cudaMemcpy(c,f,row1\*col2\*sizeof(int),cudaMemcpyDeviceToHost);

printf("\n Product of two matrices:\n "); for(i=0;i<row1;i++)

{

for(j=0;j<col2;j++)

{

printf("%d\t",c[i][j]);

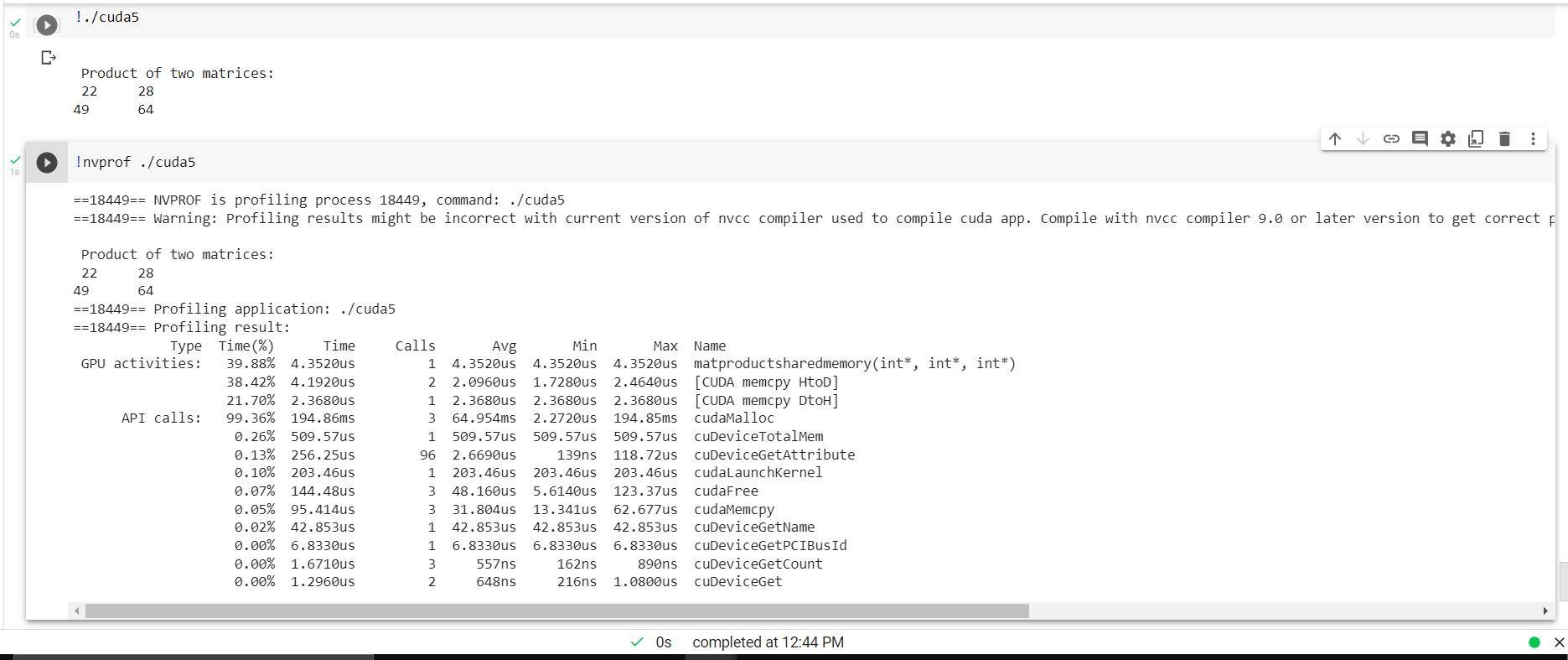
}

printf("\n");

}

cudaFree(d); cudaFree(e); cudaFree(f);

return 0;

}

# Write a CUDA C program to demonstrate the use of different GPU memories.

* + **Use of private memory.**

%%cu

#include<stdio.h> #include<cuda.h>

global void arradd(int \*x,int \*y, int \*z)

{

int id=blockIdx.x;

/\* blockIdx.x gives the respective block id which starts from 0 \*/ z[id]=x[id]+y[id];

}

int main()

{

int a[6];

int b[6];

int c[6];

int \*d,\*e,\*f; int i;

printf("\n Six elements of first array: "); for(i=0;i<6;i++)

{

a[i]=i;

printf("%d ",a[i]);

}

printf("\n Six elements of second array: "); for(i=0;i<6;i++)

{

b[i]=i+1; printf("%d ",b[i]);

}

cudaMalloc((void \*\*)&d,6\*sizeof(int)); cudaMalloc((void \*\*)&e,6\*sizeof(int)); cudaMalloc((void \*\*)&f,6\*sizeof(int));

cudaMemcpy(d,a,6\*sizeof(int),cudaMemcpyHostToDevice); cudaMemcpy(e,b,6\*sizeof(int),cudaMemcpyHostToDevice);

arradd<<<6,1>>>(d,e,f); cudaMemcpy(c,f,6\*sizeof(int),cudaMemcpyDeviceToHost);

printf("\nSum of two arrays:\n "); for(i=0;i<6;i++)

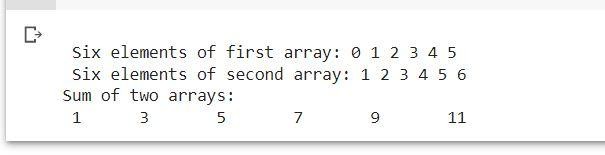
{

printf("%d\t",c[i]);

}

cudaFree(d); cudaFree(e); cudaFree(f);

return 0;

}

# Use of shared memory.

%%cu

#include<stdio.h> #include<cuda.h> #define row1 2

#define col1 3

#define row2 3

#define col2 2

global void matrixproduct(int \*l,int \*m, int \*n)

{

int x=blockIdx.x; int y=blockIdx.y;

shared int p[col1];

int i;

int k=threadIdx.x; n[col2\*y+x]=0; p[k]=l[col1\*y+k]\*m[col2\*k+x];

syncthreads();

for(i=0;i<col1;i++) n[col2\*y+x]=n[col2\*y+x]+p[i];

}

int main()

{

int a[row1][col1]={1,2,3,4 ,5,6};

int b[row2][col2]={1,2,3,4,5,6}; int c[row1][col2];

int \*d,\*e,\*f; int i,j;

printf("\*\*\*\*This is an example of shared memory in cuda\*\*\*"); printf("\n first matrix of size 2\*3\n");

for(i=0;i<row1;i++)

{

for(j=0;j<col1;j++)

{

printf("%d\t",a[i][j]);

}

}

printf("\n second matrix of size 3\*2\n"); for(i=0;i<row2;i++)

{

for(j=0;j<col2;j++)

{

printf("%d\t",b[i][j]);

}

}

cudaMalloc((void \*\*)&d,row1\*col1\*sizeof(int)); cudaMalloc((void \*\*)&e,row2\*col2\*sizeof(int)); cudaMalloc((void \*\*)&f,row1\*col2\*sizeof(int));

cudaMemcpy(d,a,row1\*col1\*sizeof(int),cudaMemcpyHostToDevice); cudaMemcpy(e,b,row2\*col2\*sizeof(int),cudaMemcpyHostToDevice);

dim3 grid(col2,row1); matrixproduct<<<grid,col1>>>(d,e,f);

cudaMemcpy(c,f,row1\*col2\*sizeof(int),cudaMemcpyDeviceToHost);

printf("\n Product of two matrices:\n "); for(i=0;i<row1;i++)

{

for(j=0;j<col2;j++)

{

printf("%d\t",c[i][j]);

}

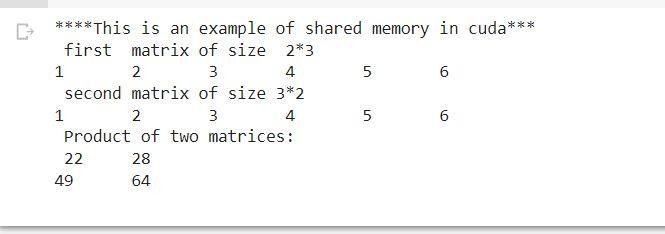
printf("\n");

}

cudaFree(d);

cudaFree(e); cudaFree(f);

return 0;

}

# Use of global memory.

%%writefile cuda11.cu #include<stdio.h> #include<cuda.h>

#define row1 2 /\* Number of rows of first matrix \*/ #define col1 3 /\* Number of columns of first matrix \*/ #define row2 3 /\* Number of rows of second matrix \*/ #define col2 2 /\* Number of columns of second matrix \*/

global void matproduct(int \*l,int \*m, int \*n)

{

int x=blockIdx.x; int y=blockIdx.y; int k;

n[col2\*y+x]=0; for(k=0;k<col1;k++)

{

n[col2\*y+x]=n[col2\*y+x]+l[col1\*y+k]\*m[col2\*k+x];

}

}

int main()

{

int a[row1][col1]={{1,2,3},{4,5,6}};

int b[row2][col2]={{1,2},{3,4},{5,6}};

int c[row1][col2]; int \*d,\*e,\*f;

int i,j;

printf("\n Enter elements of first matrix of size 2\*3\n");

cudaMalloc((void \*\*)&d,row1\*col1\*sizeof(int)); cudaMalloc((void \*\*)&e,row2\*col2\*sizeof(int)); cudaMalloc((void \*\*)&f,row1\*col2\*sizeof(int));

cudaMemcpy(d,a,row1\*col1\*sizeof(int),cudaMemcpyHostToDevice); cudaMemcpy(e,b,row2\*col2\*sizeof(int),cudaMemcpyHostToDevice);

dim3 grid(col2,row1);

/\* Here we are defining two dimensional Grid(collection of blocks) structure. Syntax is dim3 grid(no. of columns,no. of rows) \*/

matproduct<<<grid,1>>>(d,e,f);

cudaMemcpy(c,f,row1\*col2\*sizeof(int),cudaMemcpyDeviceToHost); printf("\nProduct of two matrices:\n ");

for(i=0;i<row1;i++)

{

for(j=0;j<col2;j++)

{

printf("%d\t",c[i][j]);

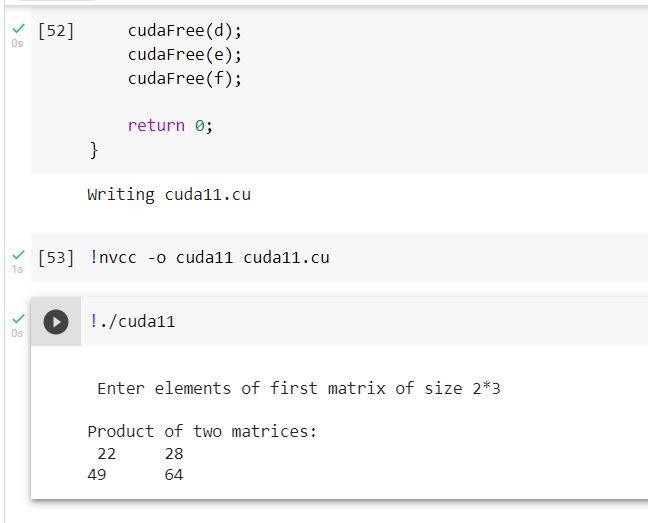
}

printf("\n");

}

cudaFree(d); cudaFree(e); cudaFree(f);

return 0;

}