```
#include <stdio.h>

// cuda headers
#include <cuda.h>
#include "helper_timer.h"

// macros
#define BLOCK_WIDTH 32

// global variables
int *hostA=NULL;
int *hostB=NULL;
int *hostC=NULL;
int *gold=NULL;

int *deviceA=NULL;
int *deviceB=NULL;
int *deviceC=NULL;

float timeOnCPU = 0.0f;
float timeOnGPU = 0.0f;

// cuda kernel function
__global__ void matMulGPU(int *A,int *B,int *C,int numARows,int numAColumns,int numBColumns,int numCColumns)
{
    // variable declarations
    int row=blockIdx.y * blockDim.y + threadIdx.y;
    int column=blockIdx.x * blockDim.x + threadIdx.x;
    // code
    if((row < numARows) && (column < numBColumns))
    {
        int value=0.0;
        for(int k=0; k < numAColumns; k++)
        {
            int a=A[row * numAColumns + k];
            int b=B[k * numBColumns + column];
            value += a*b;
        }
        C[row * numCColumns + column]=value;
    }
}

int main(int argc,char *argv[])
{
    // function declarations
    void InitA(int *data,int,int);
    void InitB(int *data,int,int);
    void matMulCPU(int*, int*, int*, int, int, int, int);
    void cleanup(void);

    // variable declarations
    int numARows=BLOCK_WIDTH;
    int numAColumns=BLOCK_WIDTH;
    int numBRows=BLOCK_WIDTH;
    int numBColumns=BLOCK_WIDTH;
```

```
56
57      int numCRows=numARows;
58      int numCColumns=numBColumns;
59
60      int numGoldRows=numARows;
61      int numGoldColumns=numBColumns;
62
63      int sizeA = numARows * numAColumns * sizeof(int);
64      int sizeB = numBRows * numBColumns * sizeof(int);
65      int sizeC = numCRows * numCColumns * sizeof(int);
66      int sizeGold = numGoldRows * numGoldColumns * sizeof(int);
67
68      cudaError_t result = cudaSuccess;
69
70      // code
71      // host memory allocation
72      hostA=(int *)malloc(sizeA);
73      if(hostA==NULL)
74      {
75          printf("Host Memory allocation is failed for hostA matrix.\n");
76          cleanup();
77          exit(EXIT_FAILURE);
78      }
79
80      hostB=(int *)malloc(sizeB);
81      if(hostB==NULL)
82      {
83          printf("Host Memory allocation is failed for hostB matrix.\n");
84          cleanup();
85          exit(EXIT_FAILURE);
86      }
87
88      hostC=(int *)malloc(sizeC);
89      if(hostC== NULL)
90      {
91          printf("Host Memory allocation is failed for hostC matrix.\n");
92          cleanup();
93          exit(EXIT_FAILURE);
94      }
95
96      gold=(int *)malloc(sizeGold);
97      if(gold== NULL)
98      {
99          printf("Host Memory allocation is failed for gold matrix.\n");
100         cleanup();
101         exit(EXIT_FAILURE);
102     }
103
104     // printing matrix dimensions and sizes
105     printf("The Dimensions Of Matrix 'hostA' Are : %d x %d                ↵
          \n",numARows,numAColumns);
106     printf("The Dimensions Of Matrix 'hostB' Are : %d x %d                ↵
          \n",numBRows,numBColumns);
107     printf("The Dimensions Of Matrix 'hostC' Are : %d x %d                ↵
          \n",numCRows,numCColumns);
108
```

```
109     printf("The Dimensions Of Matrix 'gold' Are : %d x %d
          \n",numGoldRows,numGoldColumns);
110
111     printf("Size Of Matrix hostA = %d\n",sizeA);
112     printf("Size Of Matrix hostB = %d\n",sizeB);
113     printf("Size Of Matrix hostC = %d\n",sizeC);
114
115     printf("Size Of Matrix gold = %d\n",sizeGold);
116
117     // fill source matrices
118     InitA(hostA,numARows,numAColumns);
119     InitB(hostB,numBRows,numBColumns);
120
121     // device memory allocation
122     result=cudaMalloc((void **)&deviceA,sizeA);
123     if(result!=cudaSuccess)
124     {
125         printf("Device Memory allocation is failed for deviceA matrix.\n");
126         cleanup();
127         exit(EXIT_FAILURE);
128     }
129
130     result=cudaMalloc((void **)&deviceB,sizeB);
131     if(result!=cudaSuccess)
132     {
133         printf("Device Memory allocation is failed for deviceB matrix.\n");
134         cleanup();
135         exit(EXIT_FAILURE);
136     }
137
138     result=cudaMalloc((void **)&deviceC,sizeC);
139     if(result!=cudaSuccess)
140     {
141         printf("Device Memory allocation is failed for deviceC matrix.\n");
142         cleanup();
143         exit(EXIT_FAILURE);
144     }
145
146     // copy data from host matrices into device matrices
147     result=cudaMemcpy(deviceA,hostA,sizeA,cudaMemcpyHostToDevice);
148     if(result!=cudaSuccess)
149     {
150         printf("Host to Device Data Copy is failed for deviceA matrix.\n");
151         cleanup();
152         exit(EXIT_FAILURE);
153     }
154
155     result=cudaMemcpy(deviceB,hostB,sizeB,cudaMemcpyHostToDevice);
156     if(result!=cudaSuccess)
157     {
158         printf("Host to Device Data Copy is failed for deviceB matrix.\n");
159         cleanup();
160         exit(EXIT_FAILURE);
161     }
162
163     // CUDA kernel configuration
```

```
164     dim3 dimGrid=dim3(ceil((int)numBColumns/(int)BLOCK_WIDTH),ceil((int)
          numARows/(int)BLOCK_WIDTH),1);
165     dim3 dimBlock=dim3(BLOCK_WIDTH,BLOCK_WIDTH,1);
166
167     // CUDA kernel for matrix multiplication
168     StopWatchInterface* timer = NULL;
169     sdkCreateTimer(&timer);
170     sdkStartTimer(&timer);
171
172     matMulGPU <<<dimGrid, dimBlock >>>(deviceA, deviceB, deviceC, numARows,
          numAColumns, numBColumns, numCColumns);
173
174     sdkStopTimer(&timer);
175     timeOnGPU = sdkGetTimerValue(&timer);
176     sdkDeleteTimer(&timer);
177     timer = NULL;
178
179     // copy data from device matrix into host matrix
180     result=cudaMemcpy(hostC,deviceC,sizeC,cudaMemcpyDeviceToHost);
181     if(result!=cudaSuccess)
182     {
183         printf("Device to Host Data Copy is failed for hostC matrix.\n");
184         cleanup();
185         exit(EXIT_FAILURE);
186     }
187
188     // matrix multiplication on host
189     matMulCPU(hostA, hostB, gold, numARows, numAColumns, numBColumns,
          numCColumns);
190
191     // comparison
192     int breakValue = -1;
193     bool bAccuracy = true;
194     for (int i = 0; i < numCRows * numCColumns; i++)
195     {
196         int val1 = gold[i];
197         int val2 = hostC[i];
198         if (val1 != val2)
199         {
200             bAccuracy = false;
201             breakValue = i;
202             break;
203         }
204     }
205
206     char str[128];
207     if (bAccuracy == false)
208         sprintf(str, "Comparison of CPU and GPU Matrix Multiplication is not
              accurate at array index %d", breakValue);
209     else
210         sprintf(str, "Comparison of CPU and GPU Matrix Multiplication is
              accurate");
211
212     printf("Time taken for Matrix Multiplication on CPU = %.6f\n", timeOnCPU);
213     printf("Time taken for Matrix Multiplication on GPU = %.6f\n", timeOnGPU);
214     printf("%s\n", str);
```

```
215
216     // cleanup
217     cleanup();
218
219     return(0);
220 }
221
222 void InitA(int *data,int row,int col)
223 {
224     int num=1;
225     // code
226     for(int i=0;i<row;i++)
227     {
228         for(int j=0;j<col;j++)
229         {
230             *(data + i * col + j)=num;
231             num++;
232         }
233     }
234 }
235
236 void InitB(int *data,int row,int col)
237 {
238     int num=BLOCK_WIDTH;
239     // code
240     for(int i=0;i<row;i++)
241     {
242         for(int j=0;j<col;j++)
243         {
244             *(data + i * col + j)=num;
245             num--;
246         }
247     }
248 }
249
250 void matMulCPU(int* A, int* B, int* C, int numARows, int numAColumns, int
      numBColumns, int numCColumns)
251 {
252     // code
253     StopWatchInterface* timer = NULL;
254     sdkCreateTimer(&timer);
255     sdkStartTimer(&timer);
256
257     for (int i = 0; i < numARows; ++i)
258     {
259         for (int j = 0; j < numBColumns; ++j)
260         {
261             int value = 0.0f;
262             for (int k = 0; k < numAColumns; ++k)
263             {
264                 int a = A[i * numAColumns + k];
265                 int b = B[k * numBColumns + j];
266                 value += a * b;
267             }
268             C[i * numCColumns + j] = value;
269         }
```

```
270     }
271
272     sdkStopTimer(&timer);
273     timeOnCPU = sdkGetTimerValue(&timer);
274     sdkDeleteTimer(&timer);
275     timer = NULL;
276 }
277
278 void cleanup(void)
279 {
280     // code
281     if (deviceC)
282     {
283         cudaFree(deviceC);
284         deviceC = NULL;
285     }
286
287     if (deviceB)
288     {
289         cudaFree(deviceB);
290         deviceB = NULL;
291     }
292
293     if (deviceA)
294     {
295         cudaFree(deviceA);
296         deviceA = NULL;
297     }
298
299     if (gold)
300     {
301         free(gold);
302         gold = NULL;
303     }
304
305     if (hostC)
306     {
307         free(hostC);
308         hostC = NULL;
309     }
310
311     if (hostB)
312     {
313         free(hostB);
314         hostB = NULL;
315     }
316
317     if (hostA)
318     {
319         free(hostA);
320         hostA = NULL;
321     }
322 }
323
```