```
1  // header files
2  // standard headers
3  #include <stdio.h>
4
5  // cuda headers
6  #include <cuda.h>
7  #include "helper_timer.h"
8
9  // global variables
10 //const int iNumberOfArrayElements = 5;
11 const int iNumberOfArrayElements = 11444777;
12
13 float* hostInput1 = NULL;
14 float* hostInput2 = NULL;
15 float* hostOutput = NULL;
16 float* gold = NULL;
17
18 float* deviceInput1 = NULL;
19 float* deviceInput2 = NULL;
20 float* deviceOutput = NULL;
21
22 float timeOnCPU = 0.0f;
23 float timeOnGPU = 0.0f;
24
25 // CUDA kernel
26 __global__ void vecAddGPU(float* in1, float* in2, float* out, int len)
27 {
28     // code
29     int i = blockIdx.x * blockDim.x + threadIdx.x;
30
31     if (i < len)
32     {
33         out[i] = in1[i] + in2[i];
34     }
35 }
36
37 // entry-point function
38 int main(void)
39 {
40     // function declarations
41     void fillFloatArrayWithRandomNumbers(float*, int);
42     void vecAddCPU(const float*, const float*, float*, int);
43     void cleanup(void);
44
45     // variable declarations
46     int size = iNumberOfArrayElements * sizeof(float);
47     cudaError_t result = cudaSuccess;
48
49     // code
50     // host memory allocation
51     hostInput1 = (float*)malloc(size);
52     if (hostInput1 == NULL)
53     {
54         printf("Host Memory allocation is failed for hostInput1 array.\n");
55         cleanup();
56         exit(EXIT_FAILURE);
```

```
57      }
58
59      hostInput2 = (float*)malloc(size);
60      if (hostInput2 == NULL)
61      {
62          printf("Host Memory allocation is failed for hostInput2 array.\n");
63          cleanup();
64          exit(EXIT_FAILURE);
65      }
66
67      hostOutput = (float*)malloc(size);
68      if (hostOutput == NULL)
69      {
70          printf("Host Memory allocation is failed for hostOutput array.\n");
71          cleanup();
72          exit(EXIT_FAILURE);
73      }
74
75      gold = (float*)malloc(size);
76      if (gold == NULL)
77      {
78          printf("Host Memory allocation is failed for gold array.\n");
79          cleanup();
80          exit(EXIT_FAILURE);
81      }
82
83      // filling values into host arrays
84      fillFloatArrayWithRandomNumbers(hostInput1, iNumberOfArrayElements);
85      fillFloatArrayWithRandomNumbers(hostInput2, iNumberOfArrayElements);
86
87      // device memory allocation
88      result = cudaMalloc((void**)&deviceInput1, size);
89      if (result != cudaSuccess)
90      {
91          printf("Device Memory allocation is failed for deviceInput1 array.
              \n");
92          cleanup();
93          exit(EXIT_FAILURE);
94      }
95
96      result = cudaMalloc((void**)&deviceInput2, size);
97      if (result != cudaSuccess)
98      {
99          printf("Device Memory allocation is failed for deviceInput2 array.
              \n");
100         cleanup();
101         exit(EXIT_FAILURE);
102     }
103
104     result = cudaMalloc((void**)&deviceOutput, size);
105     if (result != cudaSuccess)
106     {
107         printf("Device Memory allocation is failed for deviceOutput array.
              \n");
108         cleanup();
109         exit(EXIT_FAILURE);
```

```
110     }
111
112     // copy data from host arrays into device arrays
113     result = cudaMemcpy(deviceInput1, hostInput1, size,
          cudaMemcpyHostToDevice);
114     if (result != cudaSuccess)
115     {
116         printf("Host to Device Data Copy is failed for deviceInput1 array.
              \n");
117         cleanup();
118         exit(EXIT_FAILURE);
119     }
120
121     result = cudaMemcpy(deviceInput2, hostInput2, size,
          cudaMemcpyHostToDevice);
122     if (result != cudaSuccess)
123     {
124         printf("Host to Device Data Copy is failed for deviceInput2 array.
              \n");
125         cleanup();
126         exit(EXIT_FAILURE);
127     }
128
129     // CUDA kernel configuration
130     dim3 dimGrid = dim3((int)ceil((float)iNumberOfArrayElements / 256.0f), 1,
          1);
131     dim3 dimBlock = dim3(256, 1, 1);
132
133     // CUDA kernel for Vector Addition
134     StopWatchInterface* timer = NULL;
135     sdkCreateTimer(&timer);
136     sdkStartTimer(&timer);
137
138     vecAddGPU << <dimGrid, dimBlock >> > (deviceInput1, deviceInput2,
          deviceOutput, iNumberOfArrayElements);
139
140     sdkStopTimer(&timer);
141     timeOnGPU = sdkGetTimerValue(&timer);
142     sdkDeleteTimer(&timer);
143     timer = NULL;
144
145     // copy data from device array into host array
146     result = cudaMemcpy(hostOutput, deviceOutput, size,
          cudaMemcpyDeviceToHost);
147     if (result != cudaSuccess)
148     {
149         printf("Device to Host Data Copy is failed for hostOutput array.\n");
150         cleanup();
151         exit(EXIT_FAILURE);
152     }
153
154     // vector addition on host
155     vecAddCPU(hostInput1, hostInput2, gold, iNumberOfArrayElements);
156
157     // comparison
158     const float epsilon = 0.000001f;
```

```
159     int breakValue = -1;
160     bool bAccuracy = true;
161     for (int i = 0; i < iNumberOfArrayElements; i++)
162     {
163         float val1 = gold[i];
164         float val2 = hostOutput[i];
165         if (fabs(val1 - val2) > epsilon)
166         {
167             bAccuracy = false;
168             breakValue = i;
169             break;
170         }
171     }
172
173     char str[128];
174     if (bAccuracy == false)
175         sprintf(str, "Comparison of CPU and GPU Vector Addition is not within
              accuracy of 0.000001 at array index %d", breakValue);
176     else
177         sprintf(str, "Comparison of CPU and GPU Vector Addition is within
              accuracy of 0.000001");
178
179     // output
180     printf("Array1 begins from 0th index %.6f to %dth index %.6f\n",
          hostInput1[0], iNumberOfArrayElements - 1, hostInput1
          [iNumberOfArrayElements - 1]);
181     printf("Array2 begins from 0th index %.6f to %dth index %.6f\n",
          hostInput2[0], iNumberOfArrayElements - 1, hostInput2
          [iNumberOfArrayElements - 1]);
182     printf("CUDA Kernel Grid dimension = %d,%d,%d and Block dimension = %d,%d,
          %d\n", dimGrid.x, dimGrid.y, dimGrid.z, dimBlock.x, dimBlock.y,
          dimBlock.z);
183     printf("Output Array begins from 0th index %.6f to %dth index %.6f\n",
          hostOutput[0], iNumberOfArrayElements - 1, hostOutput
          [iNumberOfArrayElements - 1]);
184     printf("Time taken for Vector Addition on CPU = %.6f\n", timeOnCPU);
185     printf("Time taken for Vector Addition on GPU = %.6f\n", timeOnGPU);
186     printf("%s\n", str);
187
188     // cleanup
189     cleanup();
190
191     return(0);
192 }
193
194 void fillFloatArrayWithRandomNumbers(float* arr, int len)
195 {
196     // code
197     const float fscale = 1.0f / (float)RAND_MAX;
198     for (int i = 0; i < len; i++)
199     {
200         arr[i] = fscale * rand();
201     }
202 }
203
204 void vecAddCPU(const float* arr1, const float* arr2, float *out, int len)
```

```
205 {
206     // code
207     StopWatchInterface* timer = NULL;
208     sdkCreateTimer(&timer);
209     sdkStartTimer(&timer);
210 
211     for (int i = 0; i < len; i++)
212     {
213         out[i] = arr1[i] + arr2[i];
214     }
215 
216     sdkStopTimer(&timer);
217     timeOnCPU = sdkGetTimerValue(&timer);
218     sdkDeleteTimer(&timer);
219     timer = NULL;
220 }
221 
222 void cleanup(void)
223 {
224     // code
225     if (deviceOutput)
226     {
227         cudaFree(deviceOutput);
228         deviceOutput = NULL;
229     }
230 
231     if (deviceInput2)
232     {
233         cudaFree(deviceInput2);
234         deviceInput2 = NULL;
235     }
236 
237     if (deviceInput1)
238     {
239         cudaFree(deviceInput1);
240         deviceInput1 = NULL;
241     }
242 
243     if (gold)
244     {
245         free(gold);
246         gold = NULL;
247     }
248 
249     if (hostOutput)
250     {
251         free(hostOutput);
252         hostOutput = NULL;
253     }
254 
255     if (hostInput2)
256     {
257         free(hostInput2);
258         hostInput2 = NULL;
259     }
260 
```

```
261     if (hostInput1)
262     {
263         free(hostInput1);
264         hostInput1 = NULL;
265     }
266 }
267
```