```
1  // headers
2  #include <stdio.h>
3
4  #include <cuda.h> // for CUDA
5
6  #include "helper_timer.h"
7
8  // global variables
9  // odd number 11444777 is deliberate illustration ( Nvidia OpenCL Samples )
10 int iNumberOfArrayElements=5;
11
12 float *hostInput1=NULL;
13 float *hostInput2=NULL;
14 float *hostOutput=NULL;
15 float *gold=NULL;
16
17 float *deviceInput1=NULL;
18 float *deviceInput2=NULL;
19 float *deviceOutput=NULL;
20
21 float timeOnCPU;
22 float timeOnGPU;
23
24 // *** CUDA KERNEL DEFINITION ***
25 // global kernel function definition
26 __global__ void vecAdd(float *in1,float *in2,float *out,int len)
27 {
28     // variable declarations
29     int i=blockIdx.x * blockDim.x + threadIdx.x;
30     // code
31     if(i < len)
32     {
33         out[i]=in1[i]+in2[i];
34     }
35 }
36
37 int main(int argc,char *argv[])
38 {
39     // function declarations
40     void fillFloatArrayWithRandomNumbers(float *, int);
41     void vecAddHost(const float *, const float *, float *, int);
42     void cleanup(void);
43
44     // code
45     // allocate host-memory
46     hostInput1=(float *)malloc(sizeof(float) * iNumberOfArrayElements);
47     if(hostInput1== NULL)
48     {
49         printf("CPU Memory Fatal Error = Can Not Allocate Enough Memory For Host
           Input Array 1.\nExitting ...\n");
50         cleanup();
51         exit(EXIT_FAILURE);
```

```
52      }
53
54      hostInput2=(float *)malloc(sizeof(float) * iNumberOfArrayElements);
55      if(hostInput2== NULL)
56      {
57          printf("CPU Memory Fatal Error = Can Not Allocate Enough Memory For Host
              Input Array 2.\nExitting ...\n");
58          cleanup();
59          exit(EXIT_FAILURE);
60      }
61
62      hostOutput=(float *)malloc(sizeof(float) * iNumberOfArrayElements);
63      if(hostOutput== NULL)
64      {
65          printf("CPU Memory Fatal Error = Can Not Allocate Enough Memory For Host
              Output Array.\nExitting ...\n");
66          cleanup();
67          exit(EXIT_FAILURE);
68      }
69
70      gold=(float *)malloc(sizeof(float) * iNumberOfArrayElements);
71      if(gold== NULL)
72      {
73          printf("CPU Memory Fatal Error = Can Not Allocate Enough Memory For Gold
              Output Array.\nExitting ...\n");
74          cleanup();
75          exit(EXIT_FAILURE);
76      }
77
78      // fill above input host vectors with arbitary but hard-coded data
79      fillFloatArrayWithRandomNumbers(hostInput1,iNumberOfArrayElements);
80      fillFloatArrayWithRandomNumbers(hostInput2,iNumberOfArrayElements);
81
82      // allocate device-memory
83      cudaError_t err=cudaSuccess;
84      err=cudaMalloc((void **)&deviceInput1,sizeof(float) *
          iNumberOfArrayElements);
85      if(err!=cudaSuccess)
86      {
87          printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.
              \nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
88          cleanup();
89          exit(EXIT_FAILURE);
90      }
91
92      err=cudaMalloc((void **)&deviceInput2,sizeof(float) *
          iNumberOfArrayElements);
93      if(err!=cudaSuccess)
94      {
95          printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.
              \nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
96          cleanup();
```

```
 97          exit(EXIT_FAILURE);
 98      }
 99
100      err=cudaMalloc((void **)&deviceOutput,sizeof(float) *
           iNumberOfArrayElements);
101      if(err!=cudaSuccess)
102      {
103          printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.
               \nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
104          cleanup();
105          exit(EXIT_FAILURE);
106      }
107
108      // copy host memory contents to device memory
109      err=cudaMemcpy(deviceInput1,hostInput1,sizeof(float) *
           iNumberOfArrayElements,cudaMemcpyHostToDevice);
110      if(err!=cudaSuccess)
111      {
112          printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.
               \nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
113          cleanup();
114          exit(EXIT_FAILURE);
115      }
116
117      err=cudaMemcpy(deviceInput2,hostInput2,sizeof(float) *
           iNumberOfArrayElements,cudaMemcpyHostToDevice);
118      if(err!=cudaSuccess)
119      {
120          printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.
               \nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
121          cleanup();
122          exit(EXIT_FAILURE);
123      }
124
125      // cuda kernel configuration
126      dim3 DimGrid=dim3(ceil(iNumberOfArrayElements/256.0),1,1);
127      dim3 DimBlock=dim3(256,1,1);
128
129      // start timer
130      StopWatchInterface *timer = NULL;
131      sdkCreateTimer(&timer);
132      sdkStartTimer(&timer);
133
134      vecAdd<<<DimGrid,DimBlock>>>
           (deviceInput1,deviceInput2,deviceOutput,iNumberOfArrayElements);
135
136      // stop timer
137      sdkStopTimer(&timer);
138      timeOnGPU = sdkGetTimerValue(&timer);
139      sdkDeleteTimer(&timer);
140
141      // copy device memory to host memory
```

```
142     err=cudaMemcpy(hostOutput,deviceOutput,sizeof(float) *
          iNumberOfArrayElements,cudaMemcpyDeviceToHost);
143     if(err!=cudaSuccess)
144     {
145         printf("GPU Memory Fatal Error = %s In File Name %s At Line No. %d.
              \nExitting ...\n",cudaGetErrorString(err),__FILE__,__LINE__);
146         cleanup();
147         exit(EXIT_FAILURE);
148     }
149
150     // results
151     vecAddHost(hostInput1, hostInput2, gold, iNumberOfArrayElements);
152
153     // compare results for golden-host
154     const float epsilon = 0.000001f;
155     bool bAccuracy=true;
156     int breakValue=0;
157     int i;
158     for(i=0;i<iNumberOfArrayElements;i++)
159     {
160         float val1 = gold[i];
161         float val2 = hostOutput[i];
162         if(fabs(val1-val2) > epsilon)
163         {
164             bAccuracy = false;
165             breakValue=i;
166             break;
167         }
168     }
169
170     if(bAccuracy==false)
171     {
172         printf("Break Value = %d\n",breakValue);
173     }
174
175     char str[125];
176     if(bAccuracy==true)
177         sprintf(str,"%s","Comparison Of Output Arrays On CPU And GPU Are Accurate
              Within The Limit Of 0.000001");
178     else
179         sprintf(str,"%s","Not All Comparison Of Output Arrays On CPU And GPU Are
              Accurate Within The Limit Of 0.000001");
180
181     printf("1st Array Is From 0th Element %.6f To %dth Element %.6f\n",hostInput1
          [0], iNumberOfArrayElements-1, hostInput1[iNumberOfArrayElements-1]);
182     printf("2nd Array Is From 0th Element %.6f To %dth Element %.6f\n",hostInput2
          [0], iNumberOfArrayElements-1, hostInput2[iNumberOfArrayElements-1]);
183     printf("Grid Dimension = (%d,1,1) And Block Dimension = (%d,1,1)
          \n",DimGrid.x,DimBlock.x);
184     printf("Sum Of Each Element From Above 2 Arrays Creates 3rd Array As :\n");
185     printf("3nd Array Is From 0th Element %.6f To %dth Element %.6f\n",hostOutput
          [0], iNumberOfArrayElements-1, hostOutput[iNumberOfArrayElements-1]);
```

```
186     printf("The Time Taken To Do Above Addition On CPU = %.6f (ms)\n",timeOnCPU);
187     printf("The Time Taken To Do Above Addition On GPU = %.6f (ms)\n",timeOnGPU);
188     printf("%s\n",str);
189
190     // total cleanup
191     cleanup();
192     return(0);
193 }
194
195 void cleanup(void)
196 {
197     // code
198
199     // free allocated device-memory
200     if(deviceInput1)
201     {
202         cudaFree(deviceInput1);
203         deviceInput1=NULL;
204     }
205
206     if(deviceInput2)
207     {
208         cudaFree(deviceInput2);
209         deviceInput2=NULL;
210     }
211
212     if(deviceOutput)
213     {
214         cudaFree(deviceOutput);
215         deviceOutput=NULL;
216     }
217
218     // free allocated host-memory
219     if(hostInput1)
220     {
221         free(hostInput1);
222         hostInput1=NULL;
223     }
224
225     if(hostInput2)
226     {
227         free(hostInput2);
228         hostInput2=NULL;
229     }
230
231     if(hostOutput)
232     {
233         free(hostOutput);
234         hostOutput=NULL;
235     }
236
237     if(gold)
```

```
238     {
239         free(gold);
240         gold=NULL;
241     }
242 }
243
244 void fillFloatArrayWithRandomNumbers(float *pFloatArray, int iSize)
245 {
246     // code
247     int i;
248     const float fScale = 1.0f / (float)RAND_MAX;
249     for (i = 0; i < iSize; ++i)
250     {
251         pFloatArray[i] = fScale * rand();
252     }
253 }
254
255 // "Golden" Host processing vector addition function for comparison purposes
256 void vecAddHost(const float* pFloatData1, const float* pFloatData2, float*
      pFloatResult, int iNumElements)
257 {
258     int i;
259
260     StopWatchInterface *timer = NULL;
261     sdkCreateTimer(&timer);
262     sdkStartTimer(&timer);
263
264     for (i = 0; i < iNumElements; i++)
265     {
266         pFloatResult[i] = pFloatData1[i] + pFloatData2[i];
267     }
268
269     sdkStopTimer(&timer);
270     timeOnCPU = sdkGetTimerValue(&timer);
271     sdkDeleteTimer(&timer);
272 }
273
```