```cpp
// header files
// standard headers
#include <stdio.h>
#include <math.h> // fabs()

// OpenCL headers
#include <CL/opencl.h>

#include "helper_timer.h"

// global variables
//const int iNumberOfArrayElements = 5;
const int iNumberOfArrayElements = 11444777;

cl_platform_id oclPlatformID;
cl_device_id oclDeviceID;

cl_context oclContext;
cl_command_queue oclCommandQueue;

cl_program oclProgram;
cl_kernel oclKernel;

float *hostInput1=NULL;
float *hostInput2=NULL;
float *hostOutput=NULL;
float* gold = NULL;

cl_mem deviceInput1=NULL;
cl_mem deviceInput2=NULL;
cl_mem deviceOutput=NULL;

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

// OpenCL kernel
const char *oclSourceCode =
"__kernel void vecAddGPU(__global float *in1, __global float *in2, __global
  float *out,int len)" \
"{" \
"int i=get_global_id(0);" \
"if(i < len)" \
"{" \
"out[i]=in1[i]+in2[i];" \
"}" \
"}";

// entry-point function
int main(void)
{
    // function declarations
    void fillFloatArrayWithRandomNumbers(float*, int);
    size_t roundGlobalSizeToNearestMultipleOfLocalSize(int, unsigned int);
    void vecAddCPU(const float*, const float*, float*, int);
    void cleanup(void);

```

```
56      // variable declarations
57      int size = iNumberOfArrayElements * sizeof(float);
58      cl_int result;
59
60      // code
61      // host memory allocation
62      hostInput1 = (float*)malloc(size);
63      if (hostInput1 == NULL)
64      {
65          printf("Host Memory allocation is failed for hostInput1 array.\n");
66          cleanup();
67          exit(EXIT_FAILURE);
68      }
69
70      hostInput2 = (float*)malloc(size);
71      if (hostInput2 == NULL)
72      {
73          printf("Host Memory allocation is failed for hostInput2 array.\n");
74          cleanup();
75          exit(EXIT_FAILURE);
76      }
77
78      hostOutput = (float*)malloc(size);
79      if (hostOutput == NULL)
80      {
81          printf("Host Memory allocation is failed for hostOutput array.\n");
82          cleanup();
83          exit(EXIT_FAILURE);
84      }
85
86      gold = (float*)malloc(size);
87      if (gold == NULL)
88      {
89          printf("Host Memory allocation is failed for gold array.\n");
90          cleanup();
91          exit(EXIT_FAILURE);
92      }
93
94      // filling values into host arrays
95      fillFloatArrayWithRandomNumbers(hostInput1, iNumberOfArrayElements);
96      fillFloatArrayWithRandomNumbers(hostInput2, iNumberOfArrayElements);
97
98      // get OpenCL supporting platform's ID
99      result = clGetPlatformIDs(1, &oclPlatformID, NULL);
100     if (result != CL_SUCCESS)
101     {
102         printf("clGetPlatformIDs() Failed : %d\n", result);
103         cleanup();
104         exit(EXIT_FAILURE);
105     }
106
107     // get OpenCL supporting CPU device's ID
108     result = clGetDeviceIDs(oclPlatformID, CL_DEVICE_TYPE_GPU, 1,
          &oclDeviceID, NULL);
109     if (result != CL_SUCCESS)
110     {
```

```
111         printf("clGetDeviceIDs() Failed : %d\n", result);
112         cleanup();
113         exit(EXIT_FAILURE);
114     }
115
116     // create OpenCL compute context
117     oclContext = clCreateContext(NULL, 1, &oclDeviceID, NULL, NULL, &result);
118     if (result != CL_SUCCESS)
119     {
120         printf("clCreateContext() Failed : %d\n", result);
121         cleanup();
122         exit(EXIT_FAILURE);
123     }
124
125     // create command queue
126     oclCommandQueue = clCreateCommandQueue(oclContext, oclDeviceID, 0,
          &result);
127     if (result != CL_SUCCESS)
128     {
129         printf("clCreateCommandQueue() Failed : %d\n", result);
130         cleanup();
131         exit(EXIT_FAILURE);
132     }
133
134     // create OpenCL program from .cl
135     oclProgram = clCreateProgramWithSource(oclContext, 1, (const char **)
          &oclSourceCode, NULL, &result);
136     if (result != CL_SUCCESS)
137     {
138         printf("clCreateProgramWithSource() Failed : %d\n", result);
139         cleanup();
140         exit(EXIT_FAILURE);
141     }
142
143     // build OpenCL program
144     result = clBuildProgram(oclProgram, 0, NULL, NULL, NULL, NULL);
145     if (result != CL_SUCCESS)
146     {
147         size_t len;
148         char buffer[2048];
149         clGetProgramBuildInfo(oclProgram, oclDeviceID, CL_PROGRAM_BUILD_LOG,
              sizeof(buffer), buffer, &len);
150         printf("Program Build Log : %s\n", buffer);
151         printf("clBuildProgram() Failed : %d\n", result);
152         cleanup();
153         exit(EXIT_FAILURE);
154     }
155
156     // create OpenCL kernel by passing kernel function name that we used
          in .cl file
157     oclKernel = clCreateKernel(oclProgram, "vecAddGPU", &result);
158     if (result != CL_SUCCESS)
159     {
160         printf("clCreateKernel() Failed : %d\n", result);
161         cleanup();
162         exit(EXIT_FAILURE);
```

```
163     }
164
165     // device memory allocation
166     deviceInput1=clCreateBuffer
          (oclContext,CL_MEM_READ_ONLY,size,NULL,&result);
167     if(result!=CL_SUCCESS)
168     {
169         printf("clCreateBuffer() Failed For 1st Input Array : %d\n",result);
170         cleanup();
171         exit(EXIT_FAILURE);
172     }
173
174     deviceInput2=clCreateBuffer
          (oclContext,CL_MEM_READ_ONLY,size,NULL,&result);
175     if(result!=CL_SUCCESS)
176     {
177         printf("clCreateBuffer() Failed For 2nd Input Array : %d\n",result);
178         cleanup();
179         exit(EXIT_FAILURE);
180     }
181
182     deviceOutput=clCreateBuffer
          (oclContext,CL_MEM_WRITE_ONLY,size,NULL,&result);
183     if(result!=CL_SUCCESS)
184     {
185         printf("clCreateBuffer() Failed For Output Array : %d\n",result);
186         cleanup();
187         exit(EXIT_FAILURE);
188     }
189
190     // set 0 based 0th argument i.e. deviceInput1
191     result=clSetKernelArg(oclKernel,0,sizeof(cl_mem),(void *)&deviceInput1);
192     if(result != CL_SUCCESS)
193     {
194         printf("clSetKernelArg() Failed For 1st Argument : %d\n",result);
195         cleanup();
196         exit(EXIT_FAILURE);
197     }
198
199     // set 0 based 1st argument i.e. deviceInput2
200     result=clSetKernelArg(oclKernel,1,sizeof(cl_mem),(void *)&deviceInput2);
201     if(result != CL_SUCCESS)
202     {
203         printf("clSetKernelArg() Failed For 2nd Argument : %d\n",result);
204         cleanup();
205         exit(EXIT_FAILURE);
206     }
207
208     // set 0 based 2nd argument i.e. deviceOutput
209     result=clSetKernelArg(oclKernel,2,sizeof(cl_mem),(void *)&deviceOutput);
210     if(result != CL_SUCCESS)
211     {
212         printf("clSetKernelArg() Failed For 3rd Argument : %d\n",result);
213         cleanup();
214         exit(EXIT_FAILURE);
215     }
```

```
216
217     // set 0 based 3rd argument i.e. len
218     result=clSetKernelArg(oclKernel,3,sizeof(cl_int),(void *)
          &iNumberOfArrayElements);
219     if(result != CL_SUCCESS)
220     {
221         printf("clSetKernelArg() Failed For 4th Argument : %d\n",result);
222         cleanup();
223         exit(EXIT_FAILURE);
224     }
225
226     // write abve 'input' device buffer to device memory
227     result=clEnqueueWriteBuffer
          (oclCommandQueue,deviceInput1,CL_FALSE,0,size,hostInput1,0,NULL,NULL);
228     if(result != CL_SUCCESS)
229     {
230         printf("clEnqueueWriteBuffer() Failed For 1st Input Device Buffer : %d
            \n",result);
231         cleanup();
232         exit(EXIT_FAILURE);
233     }
234
235     result=clEnqueueWriteBuffer
          (oclCommandQueue,deviceInput2,CL_FALSE,0,size,hostInput2,0,NULL,NULL);
236     if(result != CL_SUCCESS)
237     {
238         printf("clEnqueueWriteBuffer() Failed For 2nd Input Device Buffer : %d
            \n",result);
239         cleanup();
240         exit(EXIT_FAILURE);
241     }
242
243     // kernel configuration
244 //    size_t localWorkSize=5;
245     size_t localWorkSize=256;
246     size_t globalWorkSize;
247     globalWorkSize=roundGlobalSizeToNearestMultipleOfLocalSize(localWorkSize,
          iNumberOfArrayElements);
248
249     // start timer
250     StopWatchInterface *timer = NULL;
251     sdkCreateTimer(&timer);
252     sdkStartTimer(&timer);
253
254     result=clEnqueueNDRangeKernel
          (oclCommandQueue,oclKernel,1,NULL,&globalWorkSize,&localWorkSize,0,NULL,
          NULL);
255     if(result != CL_SUCCESS)
256     {
257         printf("clEnqueueNDRangeKernel() Failed : %d\n",result);
258         cleanup();
259         exit(EXIT_FAILURE);
260     }
261
262     // finish OpenCL command queue
263     clFinish(oclCommandQueue);
```

```
264
265     // stop timer
266     sdkStopTimer(&timer);
267     timeOnGPU = sdkGetTimerValue(&timer);
268     sdkDeleteTimer(&timer);
269
270     // read back result from the device (i.e from deviceOutput) into cpu
          variable (i.e hostOutput)
271     result=clEnqueueReadBuffer
          (oclCommandQueue,deviceOutput,CL_TRUE,0,size,hostOutput,0,NULL,NULL);
272     if(result != CL_SUCCESS)
273     {
274         printf("clEnqueueReadBuffer() Failed : %d\n",result);
275         cleanup();
276         exit(EXIT_FAILURE);
277     }
278
279     // vector addition on host
280     vecAddCPU(hostInput1, hostInput2, gold, iNumberOfArrayElements);
281
282     // comparison
283     const float epsilon = 0.000001f;
284     int breakValue = -1;
285     bool bAccuracy = true;
286     for (int i = 0; i < iNumberOfArrayElements; i++)
287     {
288         float val1 = gold[i];
289         float val2 = hostOutput[i];
290         if (fabs(val1 - val2) > epsilon)
291         {
292             bAccuracy = false;
293             breakValue = i;
294             break;
295         }
296     }
297
298     char str[128];
299     if (bAccuracy == false)
300         sprintf(str, "Comparison of CPU and GPU Vector Addition is not within
              accuracy of 0.000001 at array index %d", breakValue);
301     else
302         sprintf(str, "Comparison of CPU and GPU Vector Addition is within
              accuracy of 0.000001");
303
304     // output
305     printf("Array1 begins from 0th index %.6f to %dth index %.6f\n",
          hostInput1[0], iNumberOfArrayElements - 1, hostInput1
          [iNumberOfArrayElements - 1]);
306     printf("Array2 begins from 0th index %.6f to %dth index %.6f\n",
          hostInput2[0], iNumberOfArrayElements - 1, hostInput2
          [iNumberOfArrayElements - 1]);
307     printf("OpenCL Kernel Global Work Size = %lu and LOcal Work Size = %lu\n",
           globalWorkSize, localWorkSize);
308     printf("Output Array begins from 0th index %.6f to %dth index %.6f\n",
          hostOutput[0], iNumberOfArrayElements - 1, hostOutput
          [iNumberOfArrayElements - 1]);
```

```
309     printf("Time taken for Vector Addition on CPU = %.6f\n", timeOnCPU);
310     printf("Time taken for Vector Addition on GPU = %.6f\n", timeOnGPU);
311     printf("%s\n", str);
312
313     // cleanup
314     cleanup();
315
316     return(0);
317 }
318
319 void fillFloatArrayWithRandomNumbers(float* arr, int len)
320 {
321     // code
322     const float fscale = 1.0f / (float)RAND_MAX;
323     for (int i = 0; i < len; i++)
324     {
325         arr[i] = fscale * rand();
326     }
327 }
328
329 void vecAddCPU(const float* arr1, const float* arr2, float *out, int len)
330 {
331     // code
332     StopWatchInterface* timer = NULL;
333     sdkCreateTimer(&timer);
334     sdkStartTimer(&timer);
335
336     for (int i = 0; i < len; i++)
337     {
338         out[i] = arr1[i] + arr2[i];
339     }
340
341     sdkStopTimer(&timer);
342     timeOnCPU = sdkGetTimerValue(&timer);
343     sdkDeleteTimer(&timer);
344     timer = NULL;
345 }
346
347 size_t roundGlobalSizeToNearestMultipleOfLocalSize(int local_size, unsigned
      int global_size)
348 {
349     // code
350     unsigned int r = global_size % local_size;
351     if(r == 0)
352     {
353         return(global_size);
354     }
355     else
356     {
357         return(global_size + local_size - r);
358     }
359 }
360
361 void cleanup(void)
362 {
363     // code
```

```
364     if(deviceOutput)
365     {
366         clReleaseMemObject(deviceOutput);
367         deviceOutput=NULL;
368     }
369
370     if(deviceInput2)
371     {
372         clReleaseMemObject(deviceInput2);
373         deviceInput2=NULL;
374     }
375
376     if(deviceInput1)
377     {
378         clReleaseMemObject(deviceInput1);
379         deviceInput1=NULL;
380     }
381
382     if(oclKernel)
383     {
384         clReleaseKernel(oclKernel);
385         oclKernel=NULL;
386     }
387
388     if(oclProgram)
389     {
390         clReleaseProgram(oclProgram);
391         oclProgram=NULL;
392     }
393
394     if(oclCommandQueue)
395     {
396         clReleaseCommandQueue(oclCommandQueue);
397         oclCommandQueue=NULL;
398     }
399
400     if(oclContext)
401     {
402         clReleaseContext(oclContext);
403         oclContext=NULL;
404     }
405
406     if(hostOutput)
407     {
408         free(hostOutput);
409         hostOutput=NULL;
410     }
411
412     if(hostInput2)
413     {
414         free(hostInput2);
415         hostInput2=NULL;
416     }
417
418     if(hostInput1)
419     {
```

```
420         free(hostInput1);
421         hostInput1=NULL;
422     }
423 }
424
```