```c
// header files
// standard headers
#include <stdio.h>

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

// global variables
const int iNumberOfArrayElements = 5;

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;

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

// 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 cleanup(void);

    // variable declarations
    int size = iNumberOfArrayElements * sizeof(float);
    cl_int result;

    // code
    // host memory allocation
    hostInput1 = (float*)malloc(size);
    if (hostInput1 == NULL)
    {
        printf("Host Memory allocation is failed for hostInput1 array.\n");
        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      // filling values into host arrays
76      hostInput1[0] = 101.0;
77      hostInput1[1] = 102.0;
78      hostInput1[2] = 103.0;
79      hostInput1[3] = 104.0;
80      hostInput1[4] = 105.0;
81
82      hostInput2[0] = 201.0;
83      hostInput2[1] = 202.0;
84      hostInput2[2] = 203.0;
85      hostInput2[3] = 204.0;
86      hostInput2[4] = 205.0;
87
88      // get OpenCL supporting platform's ID
89      result = clGetPlatformIDs(1, &oclPlatformID, NULL);
90      if (result != CL_SUCCESS)
91      {
92          printf("clGetPlatformIDs() Failed : %d\n", result);
93          cleanup();
94          exit(EXIT_FAILURE);
95      }
96
97      // get OpenCL supporting CPU device's ID
98      result = clGetDeviceIDs(oclPlatformID, CL_DEVICE_TYPE_GPU, 1,
          &oclDeviceID, NULL);
99      if (result != CL_SUCCESS)
100     {
101         printf("clGetDeviceIDs() Failed : %d\n", result);
102         cleanup();
103         exit(EXIT_FAILURE);
104     }
105
106     // create OpenCL compute context
107     oclContext = clCreateContext(NULL, 1, &oclDeviceID, NULL, NULL, &result);
108     if (result != CL_SUCCESS)
109     {
110         printf("clCreateContext() Failed : %d\n", result);
```

```
111         cleanup();
112         exit(EXIT_FAILURE);
113     }
114
115     // create command queue
116     oclCommandQueue = clCreateCommandQueue(oclContext, oclDeviceID, 0,
          &result);
117     if (result != CL_SUCCESS)
118     {
119         printf("clCreateCommandQueue() Failed : %d\n", result);
120         cleanup();
121         exit(EXIT_FAILURE);
122     }
123
124     // create OpenCL program from .cl
125     oclProgram = clCreateProgramWithSource(oclContext, 1, (const char **)
          &oclSourceCode, NULL, &result);
126     if (result != CL_SUCCESS)
127     {
128         printf("clCreateProgramWithSource() Failed : %d\n", result);
129         cleanup();
130         exit(EXIT_FAILURE);
131     }
132
133     // build OpenCL program
134     result = clBuildProgram(oclProgram, 0, NULL, NULL, NULL, NULL);
135     if (result != CL_SUCCESS)
136     {
137         size_t len;
138         char buffer[2048];
139         clGetProgramBuildInfo(oclProgram, oclDeviceID, CL_PROGRAM_BUILD_LOG,
            sizeof(buffer), buffer, &len);
140         printf("Program Build Log : %s\n", buffer);
141         printf("clBuildProgram() Failed : %d\n", result);
142         cleanup();
143         exit(EXIT_FAILURE);
144     }
145
146     // create OpenCL kernel by passing kernel function name that we used
          in .cl file
147     oclKernel = clCreateKernel(oclProgram, "vecAddGPU", &result);
148     if (result != CL_SUCCESS)
149     {
150         printf("clCreateKernel() Failed : %d\n", result);
151         cleanup();
152         exit(EXIT_FAILURE);
153     }
154
155     // device memory allocation
156     deviceInput1=clCreateBuffer
          (oclContext,CL_MEM_READ_ONLY,size,NULL,&result);
157     if(result!=CL_SUCCESS)
158     {
159         printf("clCreateBuffer() Failed For 1st Input Array : %d\n",result);
160         cleanup();
161         exit(EXIT_FAILURE);
```

```
162     }
163
164     deviceInput2=clCreateBuffer
          (oclContext,CL_MEM_READ_ONLY,size,NULL,&result);
165     if(result!=CL_SUCCESS)
166     {
167         printf("clCreateBuffer() Failed For 2nd Input Array : %d\n",result);
168         cleanup();
169         exit(EXIT_FAILURE);
170     }
171
172     deviceOutput=clCreateBuffer
          (oclContext,CL_MEM_WRITE_ONLY,size,NULL,&result);
173     if(result!=CL_SUCCESS)
174     {
175         printf("clCreateBuffer() Failed For Output Array : %d\n",result);
176         cleanup();
177         exit(EXIT_FAILURE);
178     }
179
180     // set 0 based 0th argument i.e. deviceInput1
181     result=clSetKernelArg(oclKernel,0,sizeof(cl_mem),(void *)&deviceInput1);
182     if(result != CL_SUCCESS)
183     {
184         printf("clSetKernelArg() Failed For 1st Argument : %d\n",result);
185         cleanup();
186         exit(EXIT_FAILURE);
187     }
188
189     // set 0 based 1st argument i.e. deviceInput2
190     result=clSetKernelArg(oclKernel,1,sizeof(cl_mem),(void *)&deviceInput2);
191     if(result != CL_SUCCESS)
192     {
193         printf("clSetKernelArg() Failed For 2nd Argument : %d\n",result);
194         cleanup();
195         exit(EXIT_FAILURE);
196     }
197
198     // set 0 based 2nd argument i.e. deviceOutput
199     result=clSetKernelArg(oclKernel,2,sizeof(cl_mem),(void *)&deviceOutput);
200     if(result != CL_SUCCESS)
201     {
202         printf("clSetKernelArg() Failed For 3rd Argument : %d\n",result);
203         cleanup();
204         exit(EXIT_FAILURE);
205     }
206
207     // set 0 based 3rd argument i.e. len
208     result=clSetKernelArg(oclKernel,3,sizeof(cl_int),(void *)
          &iNumberOfArrayElements);
209     if(result != CL_SUCCESS)
210     {
211         printf("clSetKernelArg() Failed For 4th Argument : %d\n",result);
212         cleanup();
213         exit(EXIT_FAILURE);
214     }
```

```
215
216     // write abve 'input' device buffer to device memory
217     result=clEnqueueWriteBuffer
          (oclCommandQueue,deviceInput1,CL_FALSE,0,size,hostInput1,0,NULL,NULL);
218     if(result != CL_SUCCESS)
219     {
220         printf("clEnqueueWriteBuffer() Failed For 1st Input Device Buffer : %d
              \n",result);
221         cleanup();
222         exit(EXIT_FAILURE);
223     }
224
225     result=clEnqueueWriteBuffer
          (oclCommandQueue,deviceInput2,CL_FALSE,0,size,hostInput2,0,NULL,NULL);
226     if(result != CL_SUCCESS)
227     {
228         printf("clEnqueueWriteBuffer() Failed For 2nd Input Device Buffer : %d
              \n",result);
229         cleanup();
230         exit(EXIT_FAILURE);
231     }
232
233     // kernel configuration
234     size_t global_size=5; // 1-D 5 element array operation
235     result=clEnqueueNDRangeKernel
          (oclCommandQueue,oclKernel,1,NULL,&global_size,NULL,0,NULL,NULL);
236     if(result != CL_SUCCESS)
237     {
238         printf("clEnqueueNDRangeKernel() Failed : %d\n",result);
239         cleanup();
240         exit(EXIT_FAILURE);
241     }
242
243     // finish OpenCL command queue
244     clFinish(oclCommandQueue);
245
246     // read back result from the device (i.e from deviceOutput) into cpu
          variable (i.e hostOutput)
247     result=clEnqueueReadBuffer
          (oclCommandQueue,deviceOutput,CL_TRUE,0,size,hostOutput,0,NULL,NULL);
248     if(result != CL_SUCCESS)
249     {
250         printf("clEnqueueReadBuffer() Failed : %d\n",result);
251         cleanup();
252         exit(EXIT_FAILURE);
253     }
254
255     // display results
256     int i;
257     for(i=0;i<iNumberOfArrayElements;i++)
258     {
259         printf("%f + %f = %f\n",hostInput1[i],hostInput2[i],hostOutput[i]);
260     }
261
262     // cleanup
263     cleanup();
```

```
264
265     return(0);
266 }
267
268 void cleanup(void)
269 {
270     // code
271     if(deviceOutput)
272     {
273         clReleaseMemObject(deviceOutput);
274         deviceOutput=NULL;
275     }
276
277     if(deviceInput2)
278     {
279         clReleaseMemObject(deviceInput2);
280         deviceInput2=NULL;
281     }
282
283     if(deviceInput1)
284     {
285         clReleaseMemObject(deviceInput1);
286         deviceInput1=NULL;
287     }
288
289     if(oclKernel)
290     {
291         clReleaseKernel(oclKernel);
292         oclKernel=NULL;
293     }
294
295     if(oclProgram)
296     {
297         clReleaseProgram(oclProgram);
298         oclProgram=NULL;
299     }
300
301     if(oclCommandQueue)
302     {
303         clReleaseCommandQueue(oclCommandQueue);
304         oclCommandQueue=NULL;
305     }
306
307     if(oclContext)
308     {
309         clReleaseContext(oclContext);
310         oclContext=NULL;
311     }
312
313     if(hostOutput)
314     {
315         free(hostOutput);
316         hostOutput=NULL;
317     }
318
319     if(hostInput2)
```

```
320     {
321         free(hostInput2);
322         hostInput2=NULL;
323     }
324 
325     if(hostInput1)
326     {
327         free(hostInput1);
328         hostInput1=NULL;
329     }
330 }
331 
```