List 4.10: Task parallel model - kernel taskParallel.cl

```
__kernel void taskParallelAdd(__global float* A, __global float* B, __gl
obal float* C)
{
    int base = 0;

    C[base+0] = A[base+0] + B[base+0];
    C[base+4] = A[base+4] + B[base+4];
    C[base+8] = A[base+8] + B[base+8];
    C[base+12] = A[base+12] + B[base+12];
}

__kernel void taskParallelSub(__global float* A, __global float* B, __gl
obal float* C)
{
    int base = 1;

    C[base+0] = A[base+0] - B[base+0];
    C[base+4] = A[base+4] - B[base+4];
    C[base+8] = A[base+8] - B[base+8];
    C[base+12] = A[base+12] - B[base+12];
}

__kernel void taskParallelMul(__global float* A, __global float* B, __gl
obal float* C)
{
    int base = 2;

    C[base+0] = A[base+0] * B[base+0];
    C[base+4] = A[base+4] * B[base+4];
    C[base+8] = A[base+8] * B[base+8];
    C[base+12] = A[base+12] * B[base+12];
}

__kernel void taskParallelDiv(__global float* A, __global float* B, __gl
obal float* C)
{
    int base = 3;

    C[base+0] = A[base+0] / B[base+0];
    C[base+4] = A[base+4] / B[base+4];
    C[base+8] = A[base+8] / B[base+8];
    C[base+12] = A[base+12] / B[base+12];
}
```

List 4.11: Task parallel model - host taskParallel.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
3.
4.  #ifdef __APPLE__
5.  #include <OpenCL/opencl.h>
6.  #else
7.  #include <CL/cl.h>
8.  #endif
9.
10. #define MAX_SOURCE_SIZE (0x100000)
11.
12. int main()
13. {
14.     cl_platform_id platform_id = NULL;
15.     cl_device_id device_id = NULL;
16.     cl_context context = NULL;
17.     cl_command_queue command_queue = NULL;
18.     cl_mem Amobj = NULL;
19.     cl_mem Bmobj = NULL;
20.     cl_mem Cmobj = NULL;
21.     cl_program program = NULL;
22.     cl_kernel kernel[4] = {NULL, NULL, NULL, NULL};
23.     cl_uint ret_num_devices;
24.     cl_uint ret_num_platforms;
25.     cl_int ret;
26.
27.     int i, j;
28.     float* A;
29.     float* B;
30.     float* C;
31.
32.     A = (float*)malloc(4*4*sizeof(float));
33.     B = (float*)malloc(4*4*sizeof(float));
34.     C = (float*)malloc(4*4*sizeof(float));
35.
36.
37.     FILE *fp;
38.     const char fileName[] = "./taskParallel.cl";
39.     size_t source_size;
40.     char *source_str;
41.
42.     /* Load kernel source file */
43.     fp = fopen(fileName, "rb");
44.     if (!fp) {
45.             fprintf(stderr, "Failed to load kernel.\n");
46.             exit(1);
47.     }
48.     source_str = (char *)malloc(MAX_SOURCE_SIZE);
49.     source_size = fread(source_str, 1, MAX_SOURCE_SIZE, fp);
50.     fclose(fp);
51.
52.     /* Initialize input data */
53.     for (i=0; i < 4; i++) {
```

```
            for (j=0; j < 4; j++) {
                    A[i*4+j] = i*4+j+1;
                    B[i*4+j] = j*4+i+1;
            }
    }

    /* Get platform/device information */
    ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);

    ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_DEFAULT, 1, &device_
id, &ret_num_devices);

    /* Create OpenCL Context */
    context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);


    /* Create command queue */
    command_queue = clCreateCommandQueue(context, device_id, CL_QUEUE_OUT
_OF_ORDER_EXEC_MODE_ENABLE, &ret);

    /* Create buffer object */
    Amobj = clCreateBuffer(context, CL_MEM_READ_WRITE, 4*4*sizeof(float),
NULL, &ret);
    Bmobj = clCreateBuffer(context, CL_MEM_READ_WRITE, 4*4*sizeof(float),
NULL, &ret);
    Cmobj = clCreateBuffer(context, CL_MEM_READ_WRITE, 4*4*sizeof(float),
NULL, &ret);

    /* Copy input data to memory buffer */
    ret = clEnqueueWriteBuffer(command_queue, Amobj, CL_TRUE, 0, 4*4*size
of(float), A, 0, NULL, NULL);
    ret = clEnqueueWriteBuffer(command_queue, Bmobj, CL_TRUE, 0, 4*4*size
of(float), B, 0, NULL, NULL);

    /* Create kernel from source */
    program = clCreateProgramWithSource(context, 1, (const char **)&sourc
e_str, (const size_t *)&source_size, &ret);
    ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);


    /* Create task parallel OpenCL kernel */
    kernel[0] = clCreateKernel(program, "taskParallelAdd", &ret);

    kernel[1] = clCreateKernel(program, "taskParallelSub", &ret);

    kernel[2] = clCreateKernel(program, "taskParallelMul", &ret);

    kernel[3] = clCreateKernel(program, "taskParallelDiv", &ret);


    /* Set OpenCL kernel arguments */
```

```
90.     for (i=0; i < 4; i++) {
91.             ret = clSetKernelArg(kernel[i], 0, sizeof(cl_mem), (void *)&
    Amobj);
92.             ret = clSetKernelArg(kernel[i], 1, sizeof(cl_mem), (void *)&
    Bmobj);
93.             ret = clSetKernelArg(kernel[i], 2, sizeof(cl_mem), (void *)&
    Cmobj);
94.     }
95.
96.     /* Execute OpenCL kernel as task parallel */
97.     for (i=0; i < 4; i++) {
98.             ret = clEnqueueTask(command_queue, kernel[i], 0, NULL, NULL)
    ;
99.     }
100.
101.            /* Copy result to host */
102.            ret = clEnqueueReadBuffer(command_queue, Cmobj, CL_TRUE, 0,
    4*4*sizeof(float), C, 0, NULL, NULL);
103.
104.            /* Display result */
105.            for (i=0; i < 4; i++) {
106.                    for (j=0; j < 4; j++) {
107.                            printf("%7.2f ", C[i*4+j]);
108.                    }
109.                    printf("\n");
110.            }
111.
112.            /* Finalization */
113.            ret = clFlush(command_queue);
114.            ret = clFinish(command_queue);
115.            ret = clReleaseKernel(kernel[0]);
116.            ret = clReleaseKernel(kernel[1]);
117.            ret = clReleaseKernel(kernel[2]);
118.            ret = clReleaseKernel(kernel[3]);
119.            ret = clReleaseProgram(program);
120.            ret = clReleaseMemObject(Amobj);
121.            ret = clReleaseMemObject(Bmobj);
122.            ret = clReleaseMemObject(Cmobj);
123.            ret = clReleaseCommandQueue(command_queue);
124.            ret = clReleaseContext(context);
125.
126.            free(source_str);
127.
128.            free(A);
129.            free(B);
130.            free(C);
131.
132.            return 0;
133.    }
```