List 4.8: Data parallel model - kernel dataParallel.cl

```
__kernel void dataParallel(__global float* A, __global float* B, __globa
l float* C)
{
  int base = 4*get_global_id(0);
  C[base+0] = A[base+0] + B[base+0];
  C[base+1] = A[base+1] - B[base+1];
  C[base+2] = A[base+2] * B[base+2];
  C[base+3] = A[base+3] / B[base+3];
}
```

List 4.9: Data parallel model - host dataParallel.c

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

#ifdef __APPLE__
#include <OpenCL/opencl.h>
#else
#include <CL/cl.h>
#endif

#define MAX_SOURCE_SIZE (0x100000)

int main()
{
  cl_platform_id platform_id = NULL;
  cl_device_id device_id = NULL;
  cl_context context = NULL;
  cl_command_queue command_queue = NULL;
  cl_mem Amobj = NULL;
  cl_mem Bmobj = NULL;
  cl_mem Cmobj = NULL;
  cl_program program = NULL;
  cl_kernel kernel = NULL;
  cl_uint ret_num_devices;
  cl_uint ret_num_platforms;
  cl_int ret;

  int i, j;
  float *A;
  float *B;
  float *C;

  A = (float *)malloc(4*4*sizeof(float));
  B = (float *)malloc(4*4*sizeof(float));
  C = (float *)malloc(4*4*sizeof(float));

  FILE *fp;
```

```
37.  const char fileName[] = "./dataParallel.cl";
38.  size_t source_size;
39.  char *source_str;
40.
41.  /* Load kernel source file */
42.  fp = fopen(fileName, "r");
43.  if (!fp) {
44.    fprintf(stderr, "Failed to load kernel.\n");
45.    exit(1);
46.  }
47.  source_str = (char *)malloc(MAX_SOURCE_SIZE);
48.  source_size = fread(source_str, 1, MAX_SOURCE_SIZE, fp);
49.  fclose(fp);
50.
51.  /* Initialize input data */
52.  for (i=0; i < 4; i++) {
53.    for (j=0; j < 4; j++) {
54.            A[i*4+j] = i*4+j+1;
55.            B[i*4+j] = j*4+i+1;
56.    }
57.  }
58.
59.  /* Get Platform/Device Information
60.  ret = clGetPlatformIDs(1, &platform_id, &ret_num_platforms);
61.  ret = clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_DEFAULT, 1, &device_i
   d, &ret_num_devices);
62.
63.  /* Create OpenCL Context */
64.  context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);
65.
66.  * Create command queue */
67.  command_queue = clCreateCommandQeue(context, device_id, 0, &ret);
68.
69.  * Create Buffer Object */
70.  Amobj = clCreateBuffer(context, CL_MEM_READ_WRITE, 4*4*sizeof(float),
   NULL, &ret);
71.  Bmobj = clCreateBuffer(context, CL_MEM_READ_WRITE, 4*4*sizeof(float),
   NULL, &ret);
72.  Cmobj = clCreateBuffer(context, CL_MEM_READ_WRITE, 4*4*sizeof(float),
   NULL, &ret);
73.
74.  /* Copy input data to the memory buffer */
75.  ret = clEnqueueWriteBuffer(command_queue, Amobj, CL_TRUE, 0, 4*4*sizeo
   f(float), A, 0, NULL, NULL);
76.  ret = clEnqueueWriteBuffer(command_queue, Bmobj, CL_TRUE, 0, 4*4*sizeo
   f(float), B, 0, NULL, NULL);
77.
78.  /* Create kernel program from source file*/
79.  program = clCreateProgramWithSource(context, 1, (const char **)&source
   _str, (const size_t *)&source_size, &ret);
80.  ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
```

```
81.
82.   /* Create data parallel OpenCL kernel */
83.   kernel = clCreateKernel(program, "dataParallel", &ret);
84.
85.   /* Set OpenCL kernel arguments */
86.   ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&Amobj);
87.   ret = clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *)&Bmobj);
88.   ret = clSetKernelArg(kernel, 2, sizeof(cl_mem), (void *)&Cmobj);
89.
90.   size_t global_item_size = 4;
91.   size_t local_item_size = 1;
92.
93.   /* Execute OpenCL kernel as data parallel */
94.   ret = clEnqueueNDRangeKernel(command_queue, kernel, 1, NULL,
95.   &global_item_size, &local_item_size, 0, NULL, NULL);
96.
97.   /* Transfer result to host */
98.   ret = clEnqueueReadBuffer(command_queue, Cmobj, CL_TRUE, 0, 4*4*sizeof
    (float), C, 0, NULL, NULL);
99.
100.    /* Display Results */
101.    for (i=0; i < 4; i++) {
102.            for (j=0; j < 4; j++) {
103.                    printf("%7.2f ", C[i*4+j]);
104.            }
105.            printf("\n");
106.    }
107.
108.
109.    /* Finalization */
110.    ret = clFlush(command_queue);
111.    ret = clFinish(command_queue);
112.    ret = clReleaseKernel(kernel);
113.    ret = clReleaseProgram(program);
114.    ret = clReleaseMemObject(Amobj);
115.    ret = clReleaseMemObject(Bmobj);
116.    ret = clReleaseMemObject(Cmobj);
117.    ret = clReleaseCommandQueue(command_queue);
118.    ret = clReleaseContext(context);
119.
120.    free(source_str);
121.
122.    free(A);
123.    free(B);
124.    free(C);
125.
126.    return 0;
127. }
```