APPENDIX -1
An introduction to OpenCL

OpenCL Parallelism Concept	CUDA Equivalent
Kernel	Kernel
Host program	Host program
NDRange (index space)	Grid
Work item	Thread
Work group	Block

FIGURE A.1: Mapping between OpenCL and CUDA data parallelism model concepts.

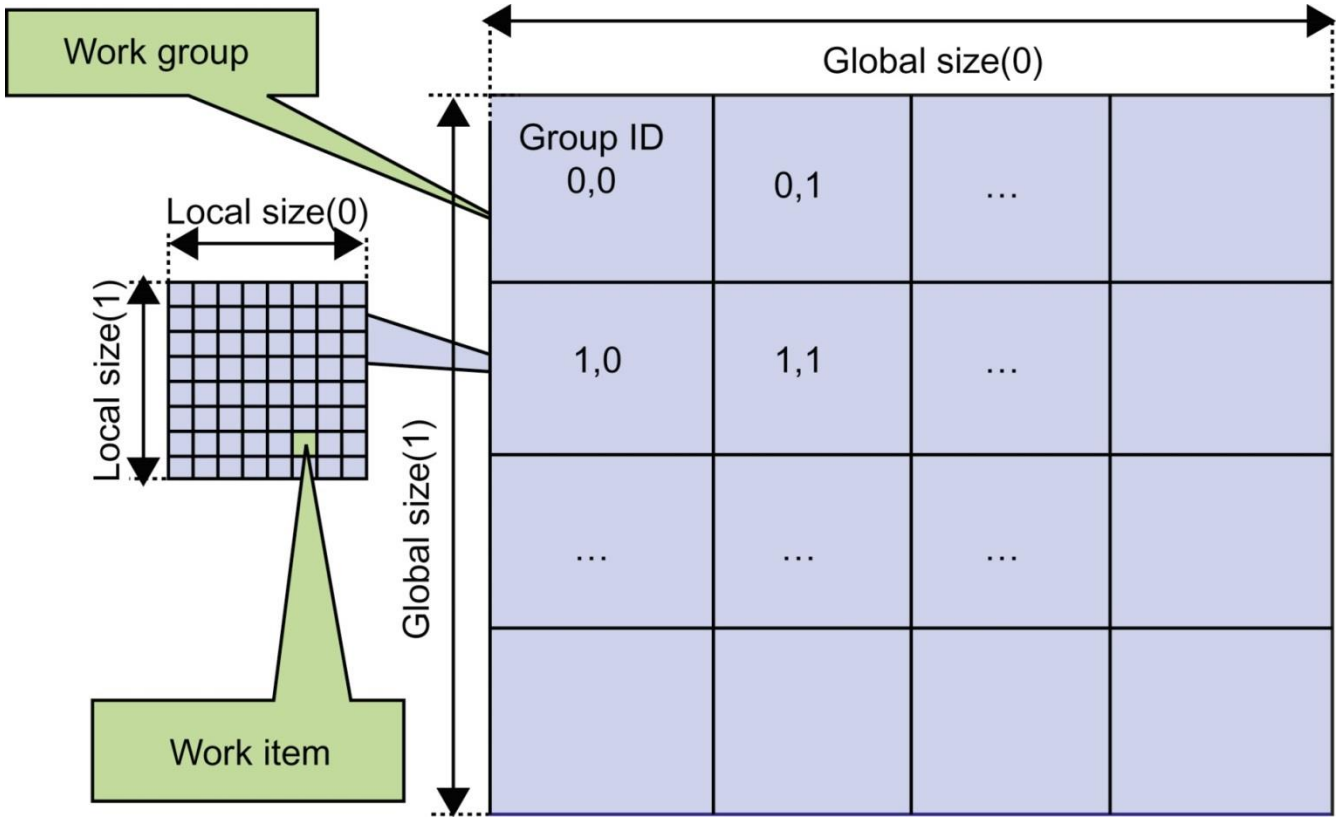

FIGURE A.2: Overview of the OpenCL parallel execution model.

OpenCL API Call	Explanation	CUDA Equivalent
`get_global_id(0)`	Global index of the work item in the `x` dimension	`blockIdx.x*blockDim x+threadIdx.x`
`get_local_id(0)`	Local index of the work item within the work group in the `x` dimension	`threadIdx.x`
`get_global_size(0)`	Size of NDRange in the `x` dimension	`gridDim.x*blockDim.x`
`get_local_size(0)`	Size of each work group in the `x` dimension	`blockDim.x`

FIGURE A.3: Mapping of OpenCL dimensions and indices to CUDA dimensions and indices.

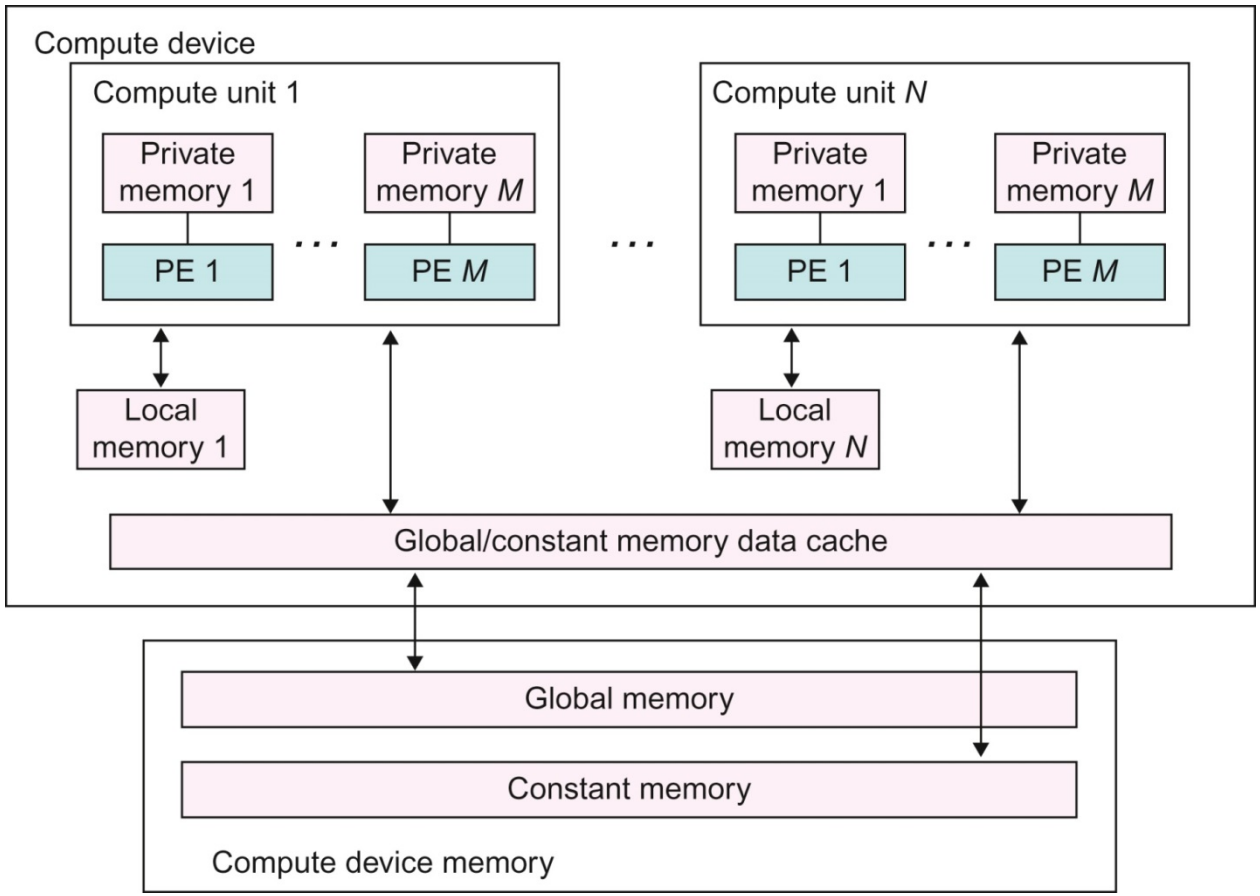

FIGURE A.4: Conceptual OpenCL device architecture. The host is not shown.

Memory Type	Host Access	Device Access	CUDA Equivalent
Global memory	Dynamic allocation; read/write access	No allocation; read/write access by all work items in all work groups, large and slow but may be cached in some devices.	Global memory
Constant memory	Dynamic allocat ion; read/write access	Static allocation; r ead-only access by all work items.	Constant memory
Local memory	Dynamic allocation; no access	Static allocation; s hared read-write access by all work items in a work group.	Shared memory
Private memory	No alloc ation; no access	Static allocation; read/write access by a single work item.	Registers and local memory

FIGURE A.5: Mapping between OpenCL and CUDA memory types.

```
__kernel void  vadd(__global const float *a,
     __global const float *b, __global float *result) {

     int i = get_global_id(0);
     result[i] = a[i] + b[i];
}
```

FIGURE A.6: A simple OpenCL kernel.

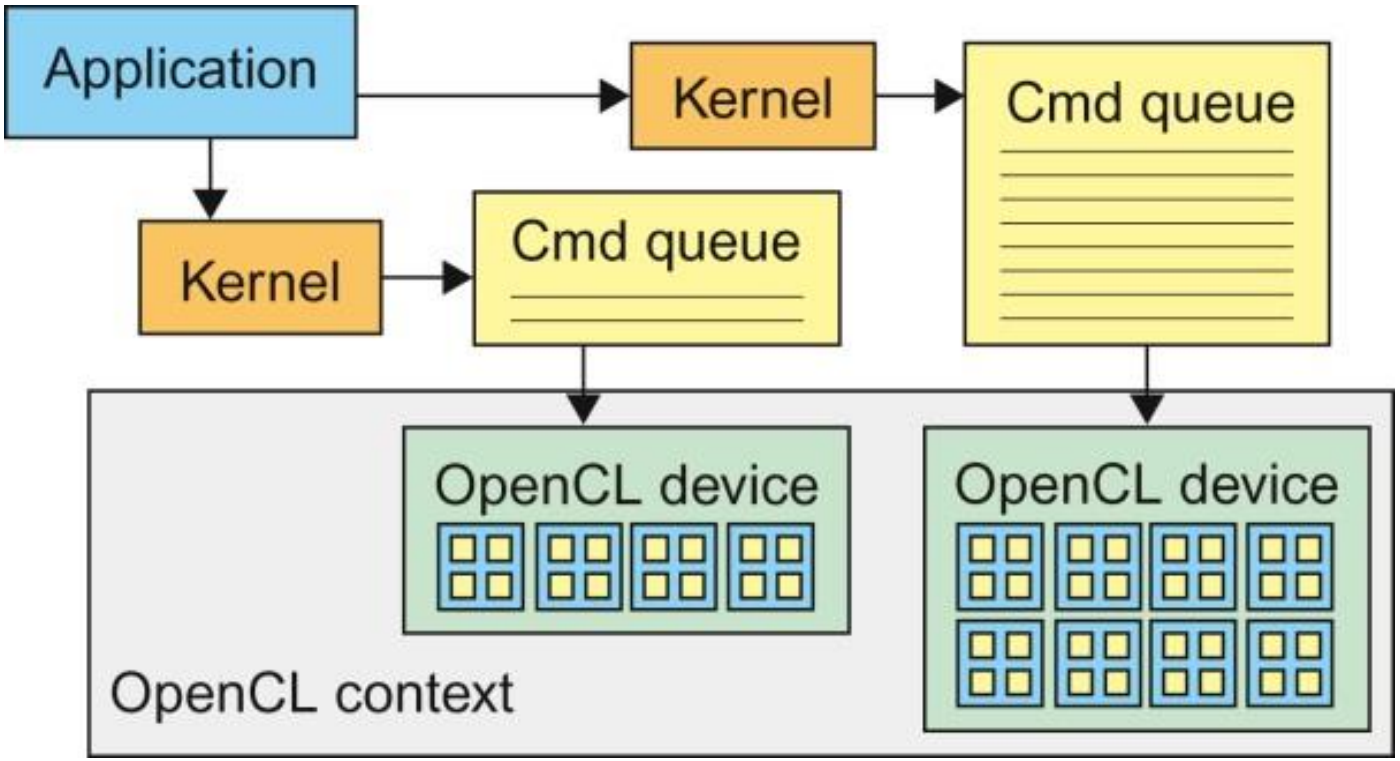

FIGURE A.7: OpenCL contexts are needed to manage devices.

```
…
1. cl_int clerr = CL_SUCCESS;

2. cl_context clctx=clCreateContextFromType(0, CL_DEVICE_TYPE_ALL,
        NULL, NULL, &clerr);

3. size_t parmsz;
4. clerr= clGetContextInfo(clctx, CL_CONTEXT_DEVICES, 0, NULL, &parmsz);

5. cl_device_id* cldevs= (cl_device_id *) malloc(parmsz);
6. clerr= clGetContextInfo(clctx, CL_CONTEXT_DEVICES, parmsz, cldevs, NULL);

7. cl_command_queue clcmdq=clCreateCommandQueue(clctx, cldevs[0], 0, &clerr);
```

FIGURE A.8: Creating OpenCL context and command queue.

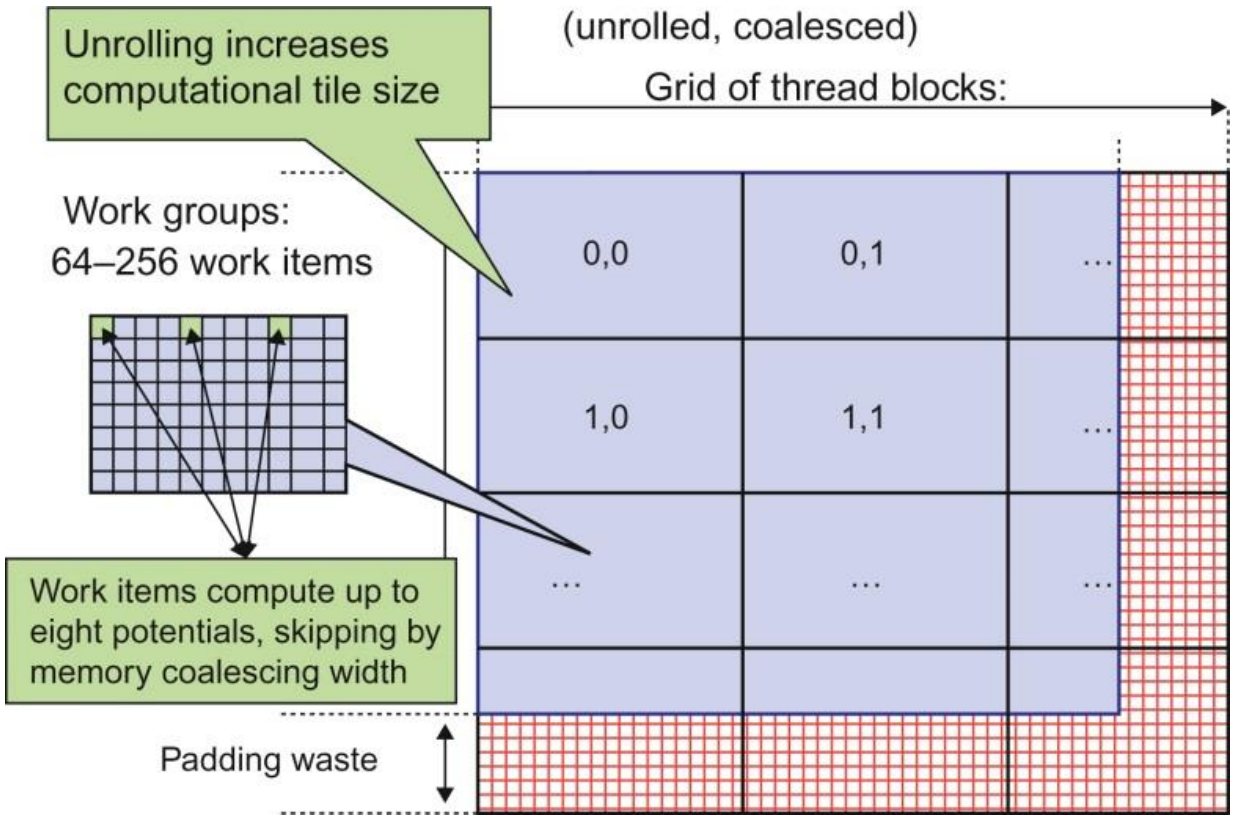

FIGURE A.9: DCS kernel version 3 NDRange configuration.

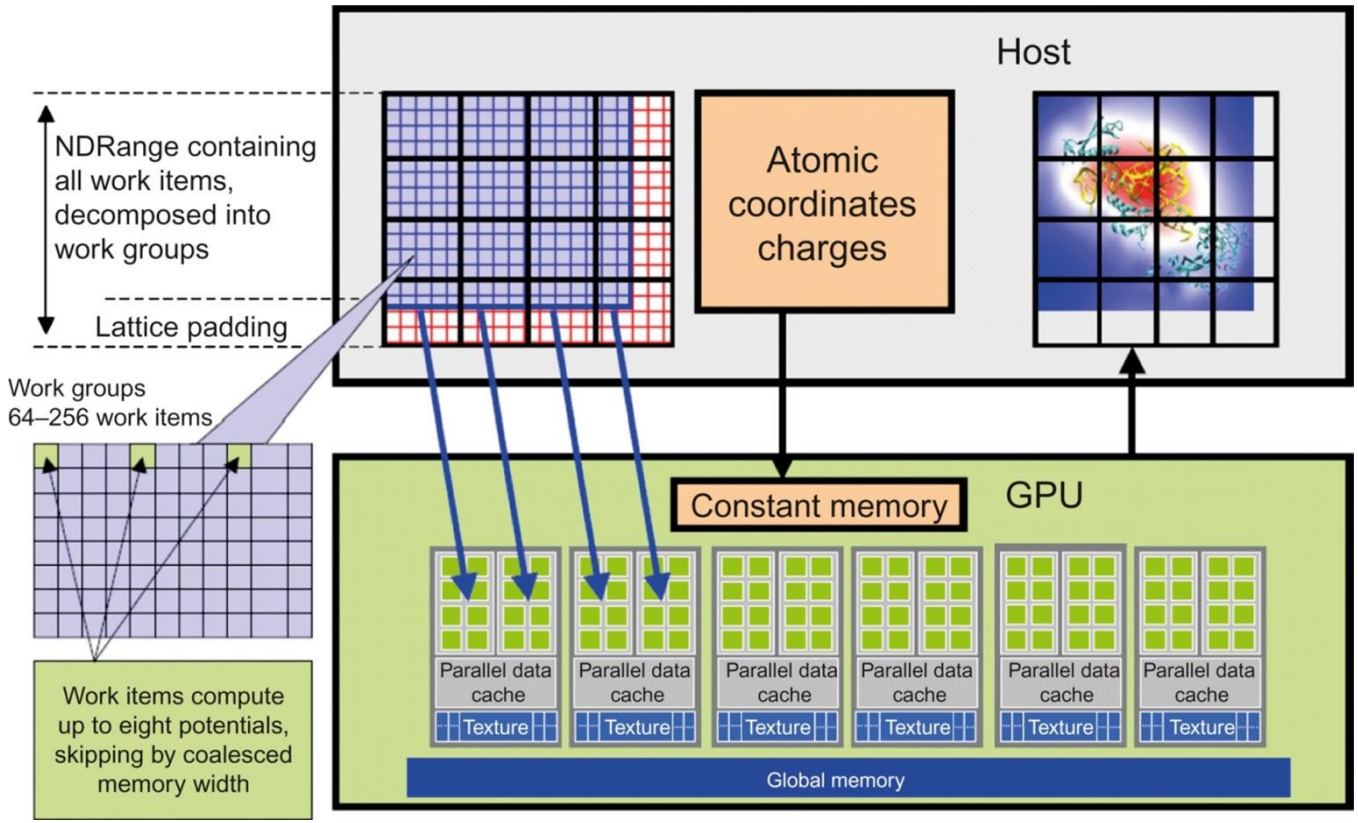

FIGURE A.10: Mapping DCS NDRange to OpenCL Device.

```
Device
OpenCL:
__kernel voidclenergy(…) {
unsigned int xindex= get_global_id(0);
unsigned int yindex= get_global_id(1);
unsigned int outaddr= get_global_size(0) * UNROLLX
*yindex+xindex;

CUDA:
__global__ void cuenergy(…) {
Unsigned int xindex= blockIdx.x *blockDim.x +threadIdx.x;
unsigned int yindex= blockIdx.y *blockDim.y +threadIdx.y;
unsigned int outaddr= gridDim.x *blockDim.x *
UNROLLX*yindex+xindex
```

FIGURE A.11: Data access indexing in OpenCL and CUDA.

```
…
for (atomid=0; atomid<numatoms; atomid++) {
float dy = coory -atominfo[atomid].y;
float dyz2= (dy * dy) + atominfo[atomid].z;
float dx1 = coorx -atominfo[atomid].x;
float dx2 = dx1 + gridspacing_coalesce;
float dx3 = dx2 + gridspacing_coalesce;
float dx4 = dx3 + gridspacing_coalesce;
float charge = atominfo[atomid].w;
energyvalx1 += charge* native_rsqrt(dx1*dx1 + dyz2);
energyvalx2 += charge* native_rsqrt(dx2*dx2 + dyz2);
energyvalx3 += charge* native_rsqrt(dx3*dx3 + dyz2);
energyvalx4 += charge* native_rsqrt(dx4*dx4 + dyz2);
}
```

FIGURE A.12: Inner loop of the OpenCL DCS kernel.

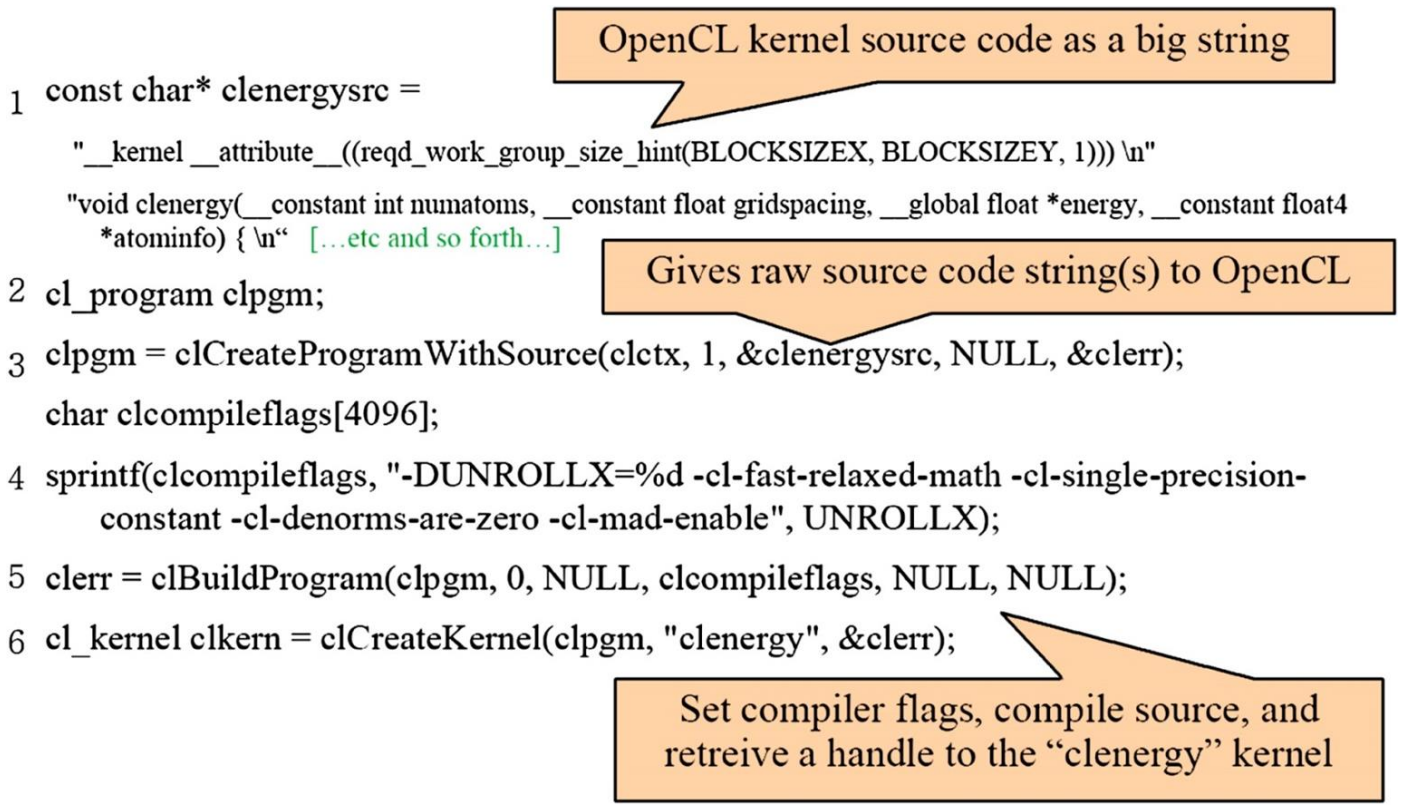

FIGURE A.13 : Building OpenCL kernel.

```
1.  doutput= clCreateBuffer(clctx, CL_MEM_READ_WRITE,volmemsz,
        NULL, NULL);
2.  datominfo= clCreateBuffer(clctx, CL_MEM_READ_ONLY,
        MAXATOMS *sizeof(cl_float4), NULL, NULL);
…
3.  clerr= clSetKernelArg(clkern, 0,sizeof(int), &runatoms);
4.  clerr= clSetKernelArg(clkern, 1,sizeof(float), &zplane);
5.  clerr= clSetKernelArg(clkern, 2,sizeof(cl_mem), &doutput);
6.  clerr= clSetKernelArg(clkern, 3,sizeof(cl_mem), &datominfo);
7.  cl_event event;
8.  clerr= clEnqueueNDRangeKernel(clcmdq,clkern, 2, NULL,
        Gsz,Bsz, 0, NULL, &event);
9.  clerr= clWaitForEvents(1, &event);
10. clerr= clReleaseEvent(event);
…
11. clEnqueueReadBuffer(clcmdq,doutput, CL_TRUE, 0,
        volmemsz, energy, 0, NULL, NULL);
12. clReleaseMemObject(doutput);
13. clReleaseMemObject(datominfo);
```

FIGURE A.14: OpenCL Host code for kernel launch and parameter passing.