13. fejezet

OpenCL – OpenGL Interop

Grafikus Processzorok Tudományos Célú Programozása

Berényi Dániel – Nagy-Egri Máté Ferenc

Az Interoperabilitás fogalma

- Minden külső eszközt kezelő API valahogy azonosítja a felé átadott memóriákat
- A probléma ott kezdődik, amikor több API-nak kellene együttműködnie, amik ugyan eszközön tárolnak memóriákat
- Az interop lehetővé teszi, hogy ne kelljen visszamásolni a gazda oldalra memóriát, csak azért, hogy kicseréljük rajta az API azonosítót.

Az Interoperabilitás fogalma

A legtöbb GPU-s Compute API lehetővé teszi valamelyik grafikus API-val való interop-ot:

OpenCL esetében a következőkkel:

- OpenGL
- DirectX 9-10-11

OpenCL- OpenGL interop

Most az OpenCL-OpenGL Interop legfontosabb lépéseit mutatjuk be

OpenCL- OpenGL interop

Wigner GPU Lab

Context létrehozás:

Először általában a grafikus API-t kell inicializálni és abból kell létrehozni a compute API contextust:

```
cl_int status;
auto glContext = wglGetCurrentContext();
auto gldc = wglGetCurrentDC();
cl_context_properties cps[] =
{
   CL_GL_CONTEXT_KHR,   (cl_context_properties) glContext,
   CL_WGL_HDC_KHR,      (cl_context_properties) gldc,
   CL_CONTEXT_PLATFORM, (cl_context_properties)platform,  0 };
auto context = clCreateContext(cps, 1, &device, 0, 0, &status);
```

OpenCL- OpenGL interop

Context létrehozás:

Először általában a grafikus API-t kell inicializálni és abból kell létrehozni a compute API contextust (ez a példa most Windows):

A korábban létrehozott grafikus kontextus és eszköz kontextus lekérdezése

```
cl_int status;
auto glContext = wglGetCurrentContext();
auto gldc = wglGetCurrentDC();
cl_context_properties cps[] =
{
   CL_GL_CONTEXT_KHR,   (cl_context_properties) glContext,
   CL_WGL_HDC_KHR,      (cl_context_properties) gldc,
   CL_CONTEXT_PLATFORM, (cl_context_properties)platform,  0 };
auto context = clCreateContext(cps, 1, &device, 0, 0, &status);
```

OpenCL- OpenGL interop

Hasonlóan linux alatt:

```
cl_int status;
auto glContext = glXGetCurrentContext();
auto gldc = glXGetCurrentDisplay();
cl_context_properties cps[] =
{
    CL_GL_CONTEXT_KHR,   (cl_context_properties) glContext,
    CL_GLX_DISPLAY_KHR,  (cl_context_properties) gldc,
    CL_CONTEXT_PLATFORM, (cl_context_properties)platform,  0
};
auto context = clCreateContext(cps, 1, &device, 0, 0,
&status);
```

OpenCL- OpenGL interop

Hasonlóan Apple esetén:

```
cl_int status;

auto kCGLContext = CGLGetCurrentContext();
auto kCGLShareGroup = CGLGetShareGroup(kCGLContext);
cl_context_properties cps[] =
{
    CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
    (cl_context_properties) kCGLShareGroup,  0
};
auto context = clCreateContext(cps, 0, 0, 0, 0, &status);
```

OpenCL- OpenGL interop

A követező lépés a bufferek megosztása:

Itt is először a grafikus API-ban hozzuk létre a buffert, és utána azt adjuk át az OpenCL-nek, hogy ő is csináljon belőle egy buffert.

OpenCL- OpenGL interop

```
GLuint glid;

glGenBuffers(1, &glid);
glBindBuffer(GL_ARRAY_BUFFER, glid);
glBufferData(GL_ARRAY_BUFFER, size, pointer,
GL_STATIC_DRAW);

cl_int status;
auto clid = clCreateFromGLBuffer( context,
CL_MEM_READ_WRITE, glid, &status );
```

OpenCL- OpenGL interop

```
GLuint glid;

glGenBuffers(1, &glid);
glBindBuffer(GL_ARRAY_BUFFER, glid);
glBufferData(GL_ARRAY_BUFFER, size, pointer,
GL_STATIC_DRAW);

cl_int status;
auto clid = clCreateFromGLBuffer( context,
CL_MEM_READ_WRITE, glid, &status );
```

OpenGL buffer azonosító generálása

OpenCL- OpenGL interop


```
GLuint glid;

glGenBuffers(1, &glid);
glBindBuffer(GL_ARRAY_BUFFER, glid);
glBufferData(GL_ARRAY_BUFFER, size, pointer,
GL_STATIC_DRAW);


cl_int status;
auto clid = clCreateFromGLBuffer( context,
CL_MEM_READ_WRITE, glid, &status );
```

Az OpenGL buffer típusának és adatainak megadása

OpenCL- OpenGL interop

```
GLuint glid;


glGenBuffers(1, &glid);
glBindBuffer(GL_ARRAY_BUFFER, glid);
glBufferData(GL_ARRAY_BUFFER, size, pointer,
GL_STATIC_DRAW);
```

Az OpenCL buffer létrehozása az OpenGL bufferből

```
cl_int status;
auto clid = clCreateFromGLBuffer( context,
CL_MEM_READ_WRITE, glid, &status );
```

OpenCL- OpenGL interop

Mivel mindkét API használhatja ugyan azt a buffert, intézkednünk kell, hogy ez egyszerre ne történhessen meg.

Mielőtt az OpenCL használhatná a buffert, el kell kérni az OpenGL-től, és miután végeztünk a használatával vissza kell neki adni.

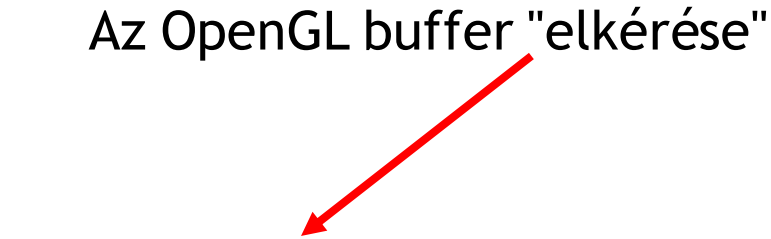
OpenCL- OpenGL interop

```
clEnqueueAcquireGLObjects(queue, 1, &clid, 0, nullptr, nullptr);

clEnqueueNDRangeKernel( ... );

clEnqueueReleaseGLObjects( queue, 1, &clid, 0, nullptr, nullptr );
```

OpenCL- OpenGL interop

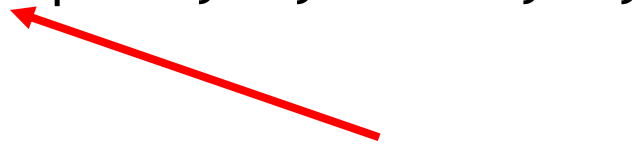



```
clEnqueueAcquireGLObjects(queue, 1, &clid, 0, nullptr, nullptr);

clEnqueueNDRangeKernel( ... );

clEnqueueReleaseGLObjects( queue, 1, &clid, 0, nullptr, nullptr );
```

Használat



OpenCL- OpenGL interop

De végig az OpenCL azonosítót használjuk, hogy a bufferre hivatkozzunk, hiszen ezek mint OpenCL hívások!

```
clEnqueueAcquireGLObjects(queue, 1, &clid, 0, nullptr, nullptr);

clEnqueueNDRangeKernel( ... );

clEnqueueReleaseGLObjects( queue, 1, &clid, 0, nullptr, nullptr );
```

OpenCL- OpenGL interop

Egy teljes lépés a programban valahogy így néz ekkor ki:

```
clEnqueueAcquireGLObjects(...);
clEnqueueNDRangeKernel(...);
clEnqueueReleaseGLObjects(...);
clFinish(...);

glUseProgram(...);
glBindVertexArray(...);
glDrawArrays(...);
glFinish();
```

OpenCL- OpenGL interop

Egy teljes lépés a programban valahogy így néz ekkor ki:

```
clEnqueueAcquireGLObjects(...);
clEnqueueNDRangeKernel(...);
clEnqueueReleaseGLObjects(...);
clFinish(...);
```

OpenCL elkéri, ír a bufferbe, elengedi, és megvárjuk, amíg minden elindított OpenCL parancs véget ér.

```
glUseProgram(...);
glBindVertexArray(...);
glDrawArrays(...);
glFinish();
```

OpenGL rajzolási beállítások, rajzolás, és szintén megvárjuk, amíg minden befejeződik