OpenCL bővítmények

OpenCL bővítmények

- Bővítmény rendszer
  - Az OpenGL bővítményeihez hasonló
  - A specifikiáció természetes fejlődése
    - Gyártó specifikus bővítmény
    - Általános bővítmények
    - Core szabvány
- Textúra támogatás (Image support)
- Együttműködés a grafikus API-val
- OpenCL C++ binding

OpenCL textúra támogatás

- Image object
  - 1D / 2D / 3D textúrák
  - 4 elemű vektorok
  - Lineáris interpoláció
  - Címzési módok

- Textúra támogatás

```
cl_int clGetDeviceInfo(...)
```

  - CL_DEVICE_IMAGE_SUPPORT

OpenCL textúra támogatás

- Image object létrehozása

```
cl_mem clCreateImage2D(...)
```

```
cl_mem clCreateImage3D(...)
```

  - image formátum
  - pitch: egy sor tárolásához szükséges byte méret
    - `0` ha a host_pointer NULL
    - `>= szélesség * elem` méret byteban ha a host_pointer adott
    - csak a betöltéshez szükséges, a belső formátum más lehet!

OpenCL textúra támogatás

- Textúra formátum leírás

```
typedef struct _cl_image_format {
        cl_channel_order image_channel_order;
        cl_channel_type  image_channel_data_type;
} cl_image_format;
```

- Csatorna sorrend
  - CL_R, CL_A
  - CL_INTENSITY
  - CL_LUMINANCE
  - CL_RG, CL_RA
  - CL_RGB
  - CL_RGBA, CL_ARGB, CL_BGRA

OpenCL textúra támogatás

- Textúra adat formátum
  - CL_SNORM_INT8 / 16
  - CL_UNORM_INT8 / 16
  - CL_UNORM_SHORT_565 / 555
  - CL_UNORM_INT_101010
  - CL_SIGNED_INT8 / 16 / 32
  - CL_UNSIGNED_INT8 / 16 / 32
  - CL_HALF_FLOAT
  - CL_FLOAT

OpenCL textúra támogatás

- Támogatott formátumok lekérdezése

```
cl_int clGetSupportedImageFormats(cl_context context,
                                  cl_mem_flags flags,
                                  cl_mem_object_type image_type,
                                  cl_uint num_entries,
                                  cl_image_format* image_formats,
                                  cl_uint* num_image_formats)
```

  - image_type: 2D/3D image object
  - image_formats: a támogatott formátumok listája

OpenCL textúra támogatás

- Olvasás Image objektumból

```
cl_int clEnqueueReadImage(...)
```

- Irás Image objektumba

```
cl_int clEnqueueWriteImage(...)
```

  - origin[3]: kezdő koordináták
  - region[3]: másolandó méret
  - row_pitch / slice_pitch: reprezentációs méret

OpenCL textúra támogatás

- Másolás Image objektumok között

```
cl_int clEnqueueCopyImage(...)
```

  - src_origin[3]: forrás koordináták
  - dst_origin[3]: cél koordináták
  - region[3]: másolandó terület mérete
- Másolás Image és Buffer objektum között

```
cl_int clEnqueueCopyImageToBuffer(...)
```

```
cl_int clEnqueueCopyBufferToImage(...)
```

OpenCL textúra támogatás

- Sampler objektum

```
cl_sampler clCreateSampler(cl_context context,
                           cl_bool normalized_coords,
                           cl_addressing_mode addressing_mode,
                           cl_filter_mode filter_mode,
                           cl_int* errcode_ret)
```

  - normalized_coords: címzési mód
  - addressing_mode: túlcímzés kezelése
    - REPEAT, CLAMP_TO_EDGE, CLAMP, NONE
  - filter_mode: textúra szűrés
    - NEAREST, LINEAR

OpenCL textúra támogatás

- Referencia számláló növelése / csökkentése

```
cl_int clRetainSampler(cl_sampler sampler)
```

```
cl_int clReleaseSampler(cl_sampler sampler)
```

- Sampler információk lekérdezése

```
cl_int clGetSamplerInfo(cl_sampler sampler,
                        cl_sampler_info param_name,
                        size_t param_value_size,
                        void* param_value,
                        size_t *param_value_size_ret)
```

  - CPU oldalon létrehozott sampler tulajdonságai

OpenCL textúra támogatás

- Image object
  - Csak olvasható: __read_only
  - Csak írható: __write_only
  - Olvasás és írás nem támogatott egyszerre!

- Sampler object
  - Host oldalon létrehozott sampler
  - Globális konstans sampler

```
const sampler_t samplerName = NORMALIZED_COORDS |
                              ADDRESS_MODE      |
                              FILTER_MODE;
```

OpenCL textúra támogatás

- Olvasás az Image objektumból

```
<o_típus> read_image{f,i,ui}(image2d_t image,
                             sampler_t sampler,
                             <c_típus> coord)
```

  - 4 elemű vektor az Image típusának megfelelően
  - {f, i, ui}: float, int, unsigned int
  - coord: a sampler címzésének megfelelő koordináták

OpenCL textúra támogatás

- Írás Image objektumba

```
void write_image{f,i,ui}(image2d_t image,
                         <coord_típus> coord,
                         <color_típus> color)
```

  - {f, i, ui}: float, int, unsigned int
  - coord: cél koordináták
  - color: a beírandó színvektor

OpenCL textúra támogatás

- Image objektum információk
  - Image dimenziók

```
int get_image_width(image{2,3}d_t image)
```

```
int get_image_height(image{2,3}d_t image)
```

```
int get_image_depth(image3d_t image)
```

```
int{2,4} get_image_dim(image{2,3}d_t image)
```

  - Image formátum

```
int get_image_channel_data_type(image{2,3}d_t image)
```

```
int get_image_channel_order(image{2,3}d_t image)
```

Együttműködés a grafikus API-val

- Célja az átjárás megteremtése
  - OpenGL és DirectX támogatás
  - Megoszthatóak
    - Általános buffer objektumok (pl. vertex buffer)
    - Textúrák
    - Render bufferek
  - A megosztandó objektumokat a grafikus API hozza létre
    - OpenCL-beli használat előtt zárolni kell
  - Az objektum használata kizárólagos!

Együttműködés a grafikus API-val

- OpenGL és OpenCL kontextus megosztás
  - GL_SHARING_EXTENSION
  - OpenGL kontextus információk

```
cl_int
clGetGLContextInfoKHR(const cl_context_properties *props,
                      cl_gl_context_info param_name,
                      size_t param_value_size,
                      void* param_value,
                      size_t* param_value_size_ret)
```

    - CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR
    - CL_DEVICES_FOR_GL_CONTEXT_KHR

Együttműködés a grafikus API-val

- OpenGL és OpenCL kontextus megosztás
  - OpenCL kontextus létrehozás

```
cl_context
clCreateContext(const cl_context_properties *props,
                                cl_uint num_devices,
                                const cl_device_id *devices,
                                void (*pfn_notify)(...),
                                void *user_data,
                                cl_int *errcode_ret)
```

    - Tulajdonságok:
      - CL_GL_CONTEXT_KHR: OpenGL kontextus
      - CL_WGL_HDC_KHR: az OpenGL kontextus HDC-je
      - CL_CONTEXT_PLATFORM: platform_id

Együttműködés a grafikus API-val

- Kontextus megosztása

```
InitGL();
cl_platform platform = createPlatform();
cl_device_id device_id = createDevice(platform, CL_DEVICE_TYPE_GPU);
cl_context sharedContext = 0;

if(CheckSharingSupport(device_id)){
  cl_context_properties props[] = {
    CL_GL_CONTEXT_KHR, (cl_context_properties)wglGetCurrentContext(),
    CL_WGL_HDC_KHR, (cl_context_properties)wglGetCurrentDC(),
    CL_CONTEXT_PLATFORM, (cl_context_properties)platform,
    0
  };

  sharedContext =
            clCreateContext(props, 1, &device_id, NULL, NULL, &err);
}
```

Együttműködés a grafikus API-val

- Buffer objektumok megosztása

```
cl_mem clCreateFromGLBuffer(cl_context context,
                            cl_mem_flags flags,
                            GLuint bufobj,
                            cl_int* errcode_ret)
```

- Image objektumok megosztása

```
cl_mem clCreateFromGLTexture2D(cl_context context,
                               cl_mem_flags flags,
                               GLenum texture_target,
                               GLint miplevel,
                               GLuint texture,
                               cl_int* errcode_ret)
```

Együttműködés a grafikus API-val

- Render buffer megosztása

```
cl_mem clCreateFromGLRenderBuffer(cl_context context,
                                  cl_mem_flags flags,
                                  GLuint renderbuffer,
                                  cl_int* errcode_ret)
```

- Az OpenCL objektumok tulajdonságai
  - Létrehozáskor aktuális értékek alapján
  - Nem követik az OpenGL objektum változásait!
    - Amennyiben változik újra meg kell osztani!

Együttműködés a grafikus API-val

- Buffer objektum megosztása
  - OpenGL vertex buffer mint OpenCL memória objektum

```
GLuint vbo;

glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);

cl_mem vboCL;

vboCL = clCreateFromGLBuffer(sharedContext, CL_MEM_WRITE_ONLY,
                             vbo, NULL);
```

Együttműködés a grafikus API-val

- Objektum lefoglalása

```
cl_int clEnqueueAcquireGLObjects(cl_command_queue command,
                                 cl_uint num_objects,
                                 const cl_mem* mem_objects,
                                 ...)
```

- Objektum felszabadítása

```
cl_mem clEnqueueReleaseGLObjects(cl_command_queue command,
                                 cl_uint num_objects,
                                 const cl_mem* mem_objects,
                                 ...)
```

  - Minden használat előtt le kell foglalni
  - Használat után fel kell szabadítani

Együttműködés a grafikus API-val

- Szinkronizáció OpenGL és OpenCL között
  - Nincs explicit szinkronizáció!
    - Szüksége lenne mindkét API támogatására
  - Mindkét API oldalán a csővezeték kiürítése
    - OpenGL: glFinish()
    - OpenCL: clFinish()
  - Implementáció függően más megoldás is lehet
    - glFlush() és clEnqueueBarrier()

Együttműködés a grafikus API-val

- Buffer objektum használata
  - OpenGL vertex buffer mint OpenCL memória objektum

```
// OpenGL hívások
glFinish();

clEnqueueAcquireGLObjects(command, 1, &vboCL,
                          0, NULL, NULL);

// Kernel paraméterek beállítása és kernel végrehajtás
clFinish();

clEnqueueReleaseGLObjects(commands, 1, &vboCL,
                          0, NULL, NULL);

// OpenGL hívások
```

OpenCL C++ binding

- Wrapper osztályok az OpenCL API fölé
- Direkt módon használja a C API-t
- www.khronos.org/registry/cl/

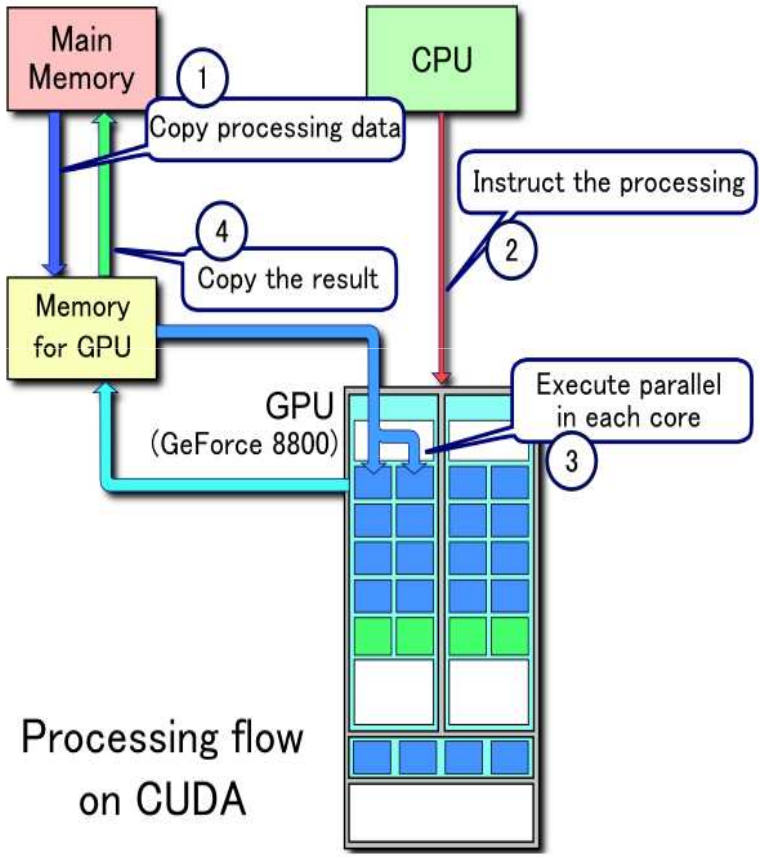
```
cl::Context context(CL_DEVICE_TYPE_GPU);
std::vector devices = context.getInfo();
cl::Program::Sources source(1, std::make_pair(srcString, srcSize));
cl::Program program(context, source);
program.build(devices);
cl::CommandQueue command(context, devices[0]);
cl::Buffer data(context, CL_MEM_READ_WRITE, size);
cl::Kernel kernel(program, „kernel”);
cl::KernelFunctor func = kernel.bind(command, cl::NDRange(count));
func(data, count).wait();
...
```

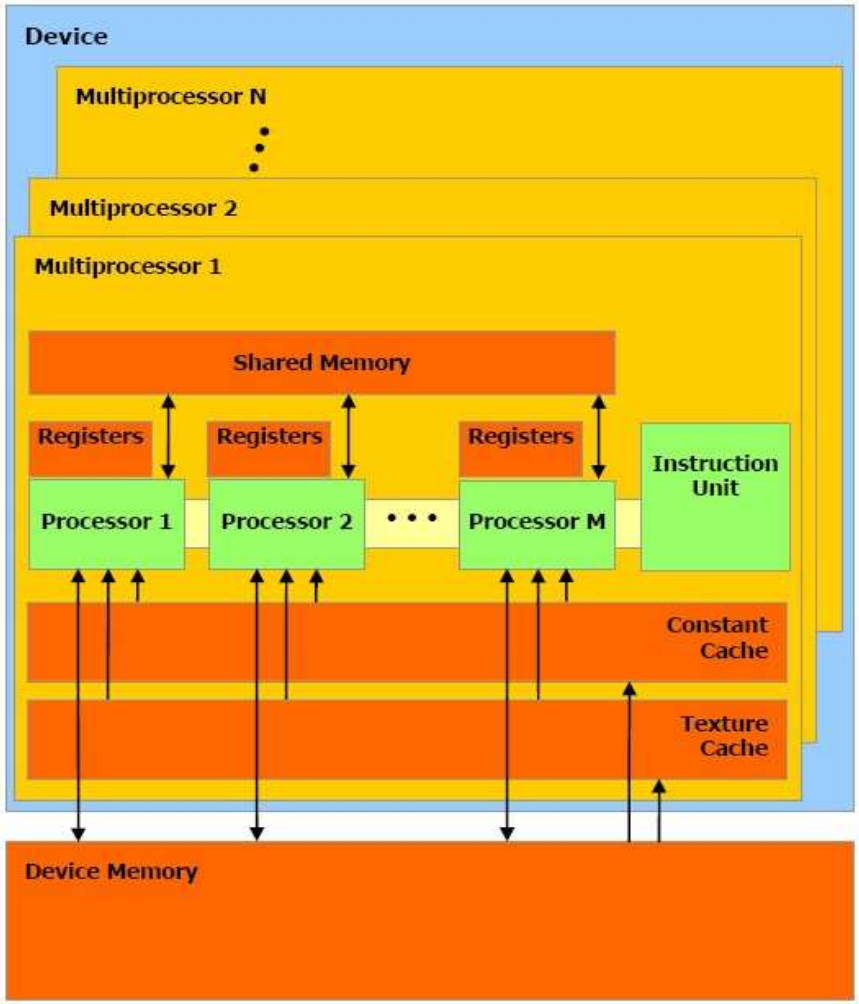
CUDA

CUDA

- CUDA mint architektúra
  - Párhuzamos feldolgozásra optimalizált architektúra
- CUDA mint GPGPU keretrendszer
  - Runtime és Driver API
  - CUDA C/C++
  - NVCC fordító
- CUDA ecosystem
  - CUBLAS
  - CUFFT
  - CUSPARSE
  - CURAND
  - Thrust

CUDA architektúra





CUDA keretrendszer

- Driver API
  - Alacsony szintű hívások
  - Hasonló koncepcióra épül mint az OpenCL
    - Device, Context, Module, Function
    - Heap memory, CUDA Array, Texture, Surface
- Runtime API
  - Magas szintű felületet nyújt a programozáshoz
  - Támogatja a host és device függvények keverését
  - Automatikus keretrendszer menedzsment

CUDA C/C++

- Támogatja a C/C++ szabvány jelentős részét
  - Adatgyűjtő osztályok
  - Osztályok származtatása
  - Osztály sablonok
  - Függvény sablonok
  - Funktorok
- Nem támogatja
  - Futásidejű típus információk (RTTI)
  - Kivételek
  - C++ Standard Library

NVCC fordító

- A fordítás menete
  - A forráskód szétválasztása host és device kódra
  - A host kód kiegészítése CUDA specifikus kódrészekkel
    - A továbbiakban a host fordító dolgozik vele
  - A device kód fordítása a megfelelő architektúrára
    - Az NVIDIA device fordító hozza létre belőle a binárist
  - A host és device binárisok összeszerkesztése

CUDA példa

```
#include <cuda.h>

__global__ void square(int* dataGPU, int dataSize){
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  dataGPU[index] = dataGPU[index] * dataGPU[index]
}

int main(int argc, char* argv[]){
  const int dataSize = 1024;
  int* dataCPU = (int*)malloc(sizeof(int)*dataSize);
  for(int i = 0; i < dataSize; ++i){
    dataCPU[i] = i;
  }

  int* dataGPU;
  cudaMalloc(&dataGPU, sizeof(int)*dataSize);
  cudaMemcpy(dataGPU, dataCPU, sizeof(int)*dataSize, cudaMemcpyHostToDevice);

  int threadsPerBlock = 256;
  int blocksPerGrid = 4;
  square<<<blocksPerGrid, threadsPerBlock>>>(dataGPU, dataSize);

  cudaMemcpy(dataCPU, dataGPU, sizeof(int)*dataSize, cudaMemcpyDeviceToHost);

  int wrongCount = 0;
  for(int i = 0; i < dataSize; ++i){
    if(dataCPU[i] != i * i) wrongCount++;
  }
  printf(„Number of wrong squares: %d\n”, wrongCount);
  cudaFree(dataGPU);
}
```