Uniwersytet Wrocławski

Programowanie Równoległe
Wykład, 20.01.2014

CUDA praktycznie 3
OpenGL Interoperability

Maciej Matyka
Instytut Fizyki Teoretycznej

Plan CUDA w praktyce

- Wykład 1: CUDA w praktyce
- Wykład 2: Cuda + Opengl
- **Wykład 3: Cuda + Opengl Interoperability**

Kompilacja LINUX

- Kompilacja CUDA + GLUT+GLEW (OpenGL)

```
> nvcc main.cu -lGL -lGLU -lGLEW -lglut
> ./a.out
```

Tak było

- Kod OpenGL

```
void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT |
    GL_DEPTH_BUFFER_BIT);

    glLoadIdentity();
    glColor3f(1,1,1);
    glPointSize(5.0);

    glBegin(GL_POINTS)
     glVertex3f(px_cpu, py_cpu,0);
    glEnd();
    glutSwapBuffers();
}
```

C4

macierz+ format

rysowanie

- Efektywność? GPU -> CPU -> OpenGL -> GPU ...

Jak to działa?

Źle, bo dane wędrują:

- **cpu → gpu** (obliczenia w kernelu CUDA)
- **gpu → cpu** (do wizualizacji)
- **cpu → gpu** (glBegin/glEnd)

Jak pozbyć się kopiowania danych między cpu/gpu?

Jak zrobić to lepiej?

OpenGL Interoperability- API do współdziałania CUDA z OpenGL

Inaczej: zestaw procedur do mapowania danych między OpenGL, a CUD

OpenGL Interoperability

- dane trzymamy w buforze na karcie VBO (Vertex Buffer Object)

```
glGenBuffers(1, &positionsVBO);
glBindBuffer(GL_ARRAY_BUFFER, positionsVBO);
unsigned int size = width * height * 4 * sizeof(float);
glBufferData(GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

OpenGL Interoperability

- dane trzymamy w buforze na karcie VBO (Vertex Buffer Object)

```
glGenBuffers(1, &positionsVBO);
glBindBuffer(GL_ARRAY_BUFFER, positionsVBO);
unsigned int size = width * height * 4 * sizeof(float);
glBufferData(GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

- następnie dane te są rejestrowane jako zasób CUDA

```
laGraphicsGLRegisterBuffer(&positionsVBO_CUDA,
                positionsVBO,                                cudaGraphicsMapFlagsWriteDisca
```

Obliczenia (VBO)

```
    cudaGraphicsResourceGetMappedPointer((void**)&positions,
                                         &num_bytes,
                                         positionsVBO_CUDA));
// Execute kernel
    dim3 dimBlock(16, 16, 1);
    dim3 dimGrid(width / dimBlock.x, height / dimBlock.y, 1);
    createVertices<<<dimGrid, dimBlock>>>(positions, time,
                                          width, height);
```

Obliczenia (VBO)

```
    cudaGraphicsResourceGetMappedPointer((void**)&positions,
                                         &num_bytes,
                                         positionsVBO_CUDA));
// Execute kernel
    dim3 dimBlock(16, 16, 1);
    dim3 dimGrid(width / dimBlock.x, height / dimBlock.y, 1);
    createVertices<<<dimGrid, dimBlock>>>(positions, time, width, height);
```

```
__global__ void createVertices(    float4* positions, float time,
                                   unsigned int width, unsigned int height)
{
    unsigned int x = blockIdx.x * blockDim.x + threadIdx.x;
    unsigned int y = blockIdx.y * blockDim.y + threadIdx.y;
    // Calculate uv coordinates
    float u = x / (float)width;
    float v = y / (float)height;
    ... (w =...)
    // Write positions
    positions[y * width + x] = make_float4(u, w, v, 1.0f);
}
```

Rendering

```
// Render from buffer object
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBindBuffer(GL_ARRAY_BUFFER, positionsVBO);
     glVertexPointer(4, GL_FLOAT, 0, 0);
     glEnableClientState(GL_VERTEX_ARRAY);
     glDrawArrays(GL_POINTS, 0, width * height);
     glDisableClientState(GL_VERTEX_ARRAY);
```

Zadanie na dzisiaj

Przepisz program przesuwający punkty na wersję CUDA-GL-Interoperability

Skorzystaj z:

http://docs.nvidia.com/cuda/cuda-c-programming-guide/#opengl-interoperability