

# Programowanie Równoległe

## Wykład, 16.12.2014

CUDA + OpenGL

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

- Korzystamy z pracowni 426
- Komputery mają zainstalowane karty nVidia GeForce
- Środowisko CUDA jest zainstalowane
- Kompilacja (plik program.cu)
  - > **nvcc program.cu**
  - > **./a.out**
- Kompilacja CUDA + GLUT (OpenGL)
  - > **nvcc main.cpp kernels.cu -lglut -lGLU**
  - > **./a.out**

# Projekt wieloplukowy w CUDA

kernels.cu

kernels.h

main.cpp (`#include <kernels.h>`)

- Kernele GPU trzymam w \*.cu
- Interfejsy C++ do kerneli GPU w \*.cu
- Deklaracje interfejsów trzymam w \*.h
- W plikach C++ ładuję ww deklaracje interfejsów
- A w praktyce?

# Projekt wieloplukowy CUDA

## main.cpp

```
#include <cuda.h>
#include <cuda_runtime.h>
#include <stdio.h>

#include "kernels.h"

int main(void)
{
    call_helloworld();

    ...

    printf("%s\n", napis_host);
}
```



# Projekt wieloplukowy CUDA

## main.cpp

```
#include <cuda.h>
#include <cuda_runtime.h>
#include <stdio.h>

#include "kernels.h"

int main(void)
{
    call_helloworld();

    ...

    printf("%s\n", napis_host);
}
```

## kernels.h

```
void call_helloworld(void);
```



# Projekt wieloplukowy CUDA

kernels.cu

main.cpp

```
#include <cuda.h>
#include <cuda_runtime.h>
#include <stdio.h>

#include "kernels.h"

int main(void)
{
    call_helloworld();

    ...

    printf("%s\n", napis_host);
}
```

```
#include <cuda.h>

__device__ char napis_device[14];
__constant__ __device__ char hw[] = "Hello
World!\n\0";

__global__ void helloWorldOnDevice(void)
{
    int idx = threadIdx.x;
    napis_device[idx] = hw[idx];
}

void call_helloworld(void)
{
    helloWorldOnDevice <<< 1, 15 >>> ();
}
```

+rozklady cuda api

kernels.h

```
void call_helloworld(void);
```



# Motywacja

- 1 CPU vs GPU ([\*anims\*](#))



# Graficzne hello world

- Punkt materialny na ekranie (OpenGL)
- Jego pozycja odczytywana z pamięci GPU
- Kernel zmienia pozycję punktu
  - np. ruch jednostajny  $x = x + vdt$
  - warunki cykliczne:  $x = x - L$ , gdy  $x > L$

# Otwarcie okna OpenGL

- Minimalny kod w GLUT (GL Utility Toolkit)
- Schemat:

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DEPTH |
        GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("CUDA GL");

    C1

    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutIdleFunc(idleFunction);

    // enter GLUT event processing cycle
    glutMainLoop();
}
```

# Otwarcie okna OpenGL

- Minimalny kod w GLUT (GL Utility Toolkit)
- Schemat:

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DEPTH |
        GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("CUDA GL");
```

C1

```
// register callbacks
glutDisplayFunc(renderScene);
glutReshapeFunc(changeSize);
glutIdleFunc(idleFunction);

// enter GLUT event processing cycle
glutMainLoop();
}
```

```
void changeSize(int w, int h)
{
    float ratio = 1.0 * w / h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, w, h);
    gluOrtho2D(-1,1,-1,1);
    glMatrixMode(GL_MODELVIEW);
}
```

# Otwarcie okna OpenGL

- Minimalny kod w GLUT (GL Utility Toolkit)
- Schemat:

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DEPTH |
        GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("CUDA GL");

    C1

    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutIdleFunc(idleFunction);

    // enter GLUT event processing cycle
    glutMainLoop();
}
```

```
void changeSize(int w, int h)
{
    float ratio = 1.0 * w / h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, w, h);
    gluOrtho2D(-1,1,-1,1);
    glMatrixMode(GL_MODELVIEW);
}
```

```
void idleFunction(void)
{
    C2    C3
    glutPostRedisplay();
}
```

# Otwarcie okna OpenGL

- Minimalny kod w GLUT (GL Utility Toolkit)
- Schemat:

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DEPTH |
        GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("CUDA GL");

    C1

    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutIdleFunc(idleFunction);

    // enter GLUT event processing cycle
    glutMainLoop();
}
```

```
void changeSize(int w, int h)
{
    float ratio = 1.0 * w / h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, w, h);
    gluOrtho2D(-1,1,-1,1);
    glMatrixMode(GL_MODELVIEW);
}
```

```
void idleFunction(void)
{
    C2    C3
    glutPostRedisplay();
}
```

```
void renderScene(void)
{
    C4
    glutSwapBuffers();
}
```

# Struktura programu

przesunięcie  
punktu (**GPU**)

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DEPTH |
        GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowPosition(100, 100);
    glutInitWindowSize(320, 320);
    glutCreateWindow("CUDA GL");

    C1

    // register callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutIdleFunc(idleFunction);

    // enter GLUT event processing cycle
    glutMainLoop();
}

void changeSize(int w, int h)
{
    float ratio = 1.0 * w / h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, w, h);
    gluOrtho2D(-1, 1, -1, 1);
    glMatrixMode(GL_MODELVIEW);
}

void idleFunction(void)
{
    C2 C3
    glutPostRedisplay();
}

void renderScene(void)
{
    C4
    glutSwapBuffers();
}
```

skopiowanie x,y  
(**GPU**->**CPU**)

ustawienie  
współrzędnych  
w pamięci **GPU**

narysowanie punktu  
na ekranie (**CPU**)

# Współrzędne punktu

- Deklaracja zmiennych w pamięci GPU
- Słowo kluczowe `__device__`

```
__device__ float px_gpu;  
__device__ float py_gpu;
```

# Współrzędne punktu

- Deklaracja zmiennych w pamięci GPU
- Słowo kluczowe `__device__`

```
__device__ float px_gpu;  
__device__ float py_gpu;
```

- Proste jądro ustawiające wartości px/py:

```
__global__ void setParDevice(float x, float y)  
{  
    px_gpu = x;  
    py_gpu = y;  
}  
  
void call_setParDevice (float x, float y)  
{  
    setParDevice <<< 1, 1 >>> (x, y);  
}
```



# Współrzędne punktu

- Deklaracja zmiennych w pamięci GPU
- Słowo kluczowe `__device__`

```
__device__ float px_gpu;  
__device__ float py_gpu;
```

- Proste jądro ustawiające wartości px/py:

```
__global__ void setParDevice(float x, float y)  
{  
    px_gpu = x;  
    py_gpu = y;  
}  
  
void call_setParDevice (float x, float y)  
{  
    setParDevice <<< 1, 1 >>> (x, y);  
}
```

- Ustawienie punktu:

```
call_setParDevice(0,0);
```

c1

# Przesunięcie punktu

- Przesuwamy składową  $x$  wg wzoru:  
$$x = x + dx * dt;$$

# Przesunięcie punktu

- Przesuwamy składową x wg wzoru:  
$$x = x + dx * dt;$$

```
__global__ void moveParDevice(float dxdt)
{
    px_gpu = px_gpu + dxdt;
    if(px_gpu>1) px_gpu = px_gpu - 2;
}

void call_movepar(float dxdt)
{
    moveParDevice <<< 1, 1 >>> (dxdt);
}
```

# Przesunięcie punktu

- Przesuwamy składową x wg wzoru:

$$x = x + dx * dt;$$

```
__global__ void moveParDevice(float dxdt)
{
    px_gpu = px_gpu + dxdt;
    if(px_gpu>1) px_gpu = px_gpu - 2;
}

void call_movepar(float dxdt)
{
    moveParDevice <<< 1, 1 >>> (dxdt);
}
```

- Wywołanie funkcji:

```
call_movepar(0.04);
```

C2

# Ustawienie punktu - cudaMemcpy

```
cudaError_t cudaMemcpyFromSymbol(  
    const void *dst,  
    const char *symbol,  
    size_t count,  
    size_t offset,  
    enum cudaMemcpyKind kind)
```

Uwaga: kopiowanie wykonujemy wykonując kod z pliku .cu

# Narysowanie punktu na ekranie

- 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?

## Możliwe rozszerzenia:

- 1. Więcej punktów (tablica `px[]`, `py[]`)
- 2. Przesunięcie punktów w pętli
- 3. Wielowątkowe przesunięcie punktów
- 4. Oddziaływanie grawitacyjne z centrum przyciągania (liczenie pierwiastka na punkt)
- 5. Porównanie wydajności CPU->GPU



**Uwaga:** przy bardzo dużej ilości punktów takie rysowanie punkt po punkcie z pomocą glBegin/glEnd będzie nieefektywne i trzeba skorzystać z tzw. buforów wierzchołków.

Słowa kluczowe do poszukiwań lepszego rozwiązania:

„**CUDA / OpenGL Interoperability**”. Wykorzystanie tych technik pozwoli na wyświetlenie nawet milionów punktów w czasie rzeczywistym (!).

Ćwiczenia 15.12.2014 – czas 2 godziny

Startując z dostarczonego szablonu (plik glut-szablon.zip):

- 1. Dopisz przesuwanie punktu w CUDA.**
- 2. Dodaj 10000 punktów z losowymi ustawieniami początkowymi.**
- 3. Dodaj grawitację i oddziaływanie punktów (odbicia od ścian, albo przyciąganie).**

Za zadanie zostanie wystawiona ocena na koniec zajęć.

- <http://www.sdsc.edu/us/training/assets/docs/NVIDIA-02-BasicsOfCUDA.pdf>