GPGPU programming on example of CUDA

Maciej Matyka

Institute of Theoretical Physics
University of Wrocław
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
Outline

CPU
   CPU Architecture

GPU
   GPU Architecture

CUDA Architecture
   Existing GPGPU frameworks

GPU programming
   Data types and kernel
   Grid, blocks, threads, memory

Examples
   Parallel HelloWorld
   N body problem (no interaction) $O(N)$
   N body problem with interaction $O(N^2)$

Summary
   Out of the main track
Outline

CPU

CPU Architecture

GPU

GPU Architecture

CUDA Architecture

Existing GPGPU frameworks

GPU programming

Data types and kernel
Grid, blocks, threads, memory

Examples

Parallel HelloWorld
N body problem (no interaction) $O(N)$
N body problem with interaction $O(N^2)$

Summary

Out of the main track
CPU Architecture

CPU = Central Processing Unit

- **ALU** - Arithmetic Logic Unit
- **Registers** - integral memory of CPU (data)
- **IR** - instruction register (processor instructions)
- **PC** - program counter (point in processor instruction list)
CPU working scheme

CPU was assumed to be sequential\(^1\):

- **fetch** - get an instruction to be executed (PC)
- **decode** - translate the instruction to elementary CPU instructions
- **execute** - execute all instructions using ALU
- **writeback** - save result (i.e. using registers)

\(^1\)http://en.wikipedia.org/wiki/Central_processing_unit
Parallel computing and CPU:

- **instrukcji** level (instruction pipelining, superscalar pipeline)
  - use of latency of parts of fundamental sequence

- **threads** level (Multiple Instructions-Multiple Data)
  - from few processors (1960) to multi-core (2001 r.)

- **data** level (Single Instruction-Multiple Data) - vector CPUs, useful to special kind of problems

Source: 'Wikipedia'
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
Graphics pipeline (1)

Source: 'The Cg Tutorial', nVidia
Graphics pipeline (2)

Source: 'The Cg Tutorial', nVidia
GPU - Graphics Processor Unit

GPU processor was optimized to run parts of graphics pipeline:

- the same procedure on many many elements
- SIMD (Single Instruction-Multiple Data) architecture
- transistors used to run many floating point operations (more than to optimize memory access like in CPUs)
- programmable shaders (vertex/pixel shaders)
- high level shader languages (CG, HLSL, GLSL)
GPU and CPU architectures

Source: 'nVidia CUDA, Programming Guide 2.2', nVidia
Performance (1)

Source: 'nVidia CUDA, Programming Guide 2.2', nVidia
Performance (2)

Source: ‘nVidia CUDA, Programming Guide 2.2’, nVidia
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
Frameworks

- Begin of **GPGPU**, calculations with programable shader units
- Today - two frameworks dedicated to GPU programming (ATI, nVidia)

- More: **OpenCL** - open standard for parallel computations (Khronos group)
What is that CUDA, actually?

CUDA, **Compute Unified Device Architecture:**

- high level language based on c/c++
- programming API
- dedicated compiler (nvcc)
- memory model

In future CUDA will be maintained from the level of any language (Source: ‘nVidia CUDA, Programming Guide 2.2’, nVidia).
Memory model in CUDA architecture

Different memory types in CUDA:

- shared memory (16kb)
- texture cache
- constant cache
- device memory (gb)
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
Outline

**CPU**
- CPU Architecture

**GPU**
- GPU Architecture

**CUDA Architecture**
- Existing GPGPU frameworks

**GPU programming**
- Data types and kernel
  - Grid, blocks, threads, memory

**Examples**
- Parallel HelloWorld
- N body problem (no interaction) $O(N)$
- N body problem with interaction $O(N^2)$

**Summary**
- Out of the main track
CUDA Programming(1)

- new vector data types:

  char1, char2, char3, char4, int1, uint1, int2, uint2, int3, uint3,
  int4, uint4, float1, float2, float3, float4, double1, double2,
  dim3 (+ many more)
CUDA Programming(2)

- example for float3:

```c
1 float3 w=make_float3(0.1, 2.2, 4.0);
2 float a = w.x + w.z;  // a = 4.1
```

- vector data types don’t have implemented arithmetic operators

- one may try this (speed?):

```c
1 __device__ float3 operator+(const float3 &a,
2 const float3 &b)
3 {
4   return make_float3(a.x+b.x, a.y+b.y, a.z+b.z);
5 }
```

- in practice - one should be aware about memory access!
Kernel, declaration and call (1)

- in CUDA programs for GPU are called *kernels*
- code of kernels are placed in *.cu* files
- *Kernel* is declared in similar way to c functions
- keyword describing level from which kernel may be called (from CPU function, other kernel, etc.)
Kernel, declaration and call (2)

- example declaration:

```c
__global__ void doSomethingOnDevice(...)
{
  ...
}
```

where:
- `__global__` compiled for GPU, called by CPU
- `__device__` compiled for GPU, called by GPU
- `__host__` compiled for CPU, called by CPU
Kernel, declaration and call (3)

- **Kernel** is called in similar way as c function
- additional declaration of number of blocks in a grid and threads in the block

1. `<<< ... , ... >>>`

- example call:

```c
1  int blockSize = 128;
2  int nBlocks = N/blockSize + (N % blockSize == 0 ? 0 : 1);
3
4  doSomethingOnDevice <<< nBlocks, blockSize >>> (...);
```
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
Grid, blocks, threads...

1. \texttt{doSomethingOnDevice \textless\textless nBlocks, blockSize \textgreater\textgreater (\ldots);}

- creates grid of blocks (grid)
- \texttt{nBlocks} of blocks each
- each block consists of \texttt{blockSize} threads

Example →

- \texttt{nBlocks} = \texttt{dim3}(3, 2, 1);
- \texttt{blockSize} = \texttt{dim3}(4, 3, 1);

Source: 'nVidia CUDA, Programming Guide 2.2', nVidia
Data declaration, memory (1)

- constants in GPU memory, i.e.
  ```c
  __constant__ float3 Gravity = {0,-1.0,0};
  ```

- constants in device memory, i.e.
  ```c
  __device__ float3 velocity = {0,0,0};
  ```

- memory shared between threads in a block
  ```c
  __shared__ float3 table[N];
  ```

- access to data of type __device__ from CPU, example:
  ```c
  float3 vel_host;
  const char *symbol="velocity";
  cudaMemcpyFromSymbol (vel_host, symbol,
  sizeof(float3), 0, cudaMemcpyDeviceToHost);
  ```
Memory allocation and copying (CPU ↔ GPU)

CPU
1 \texttt{float \*a\_h = (float \*)malloc(size);}

GPU
1 \texttt{float \*a\_d;}
2 \texttt{cudaMalloc((void \**) \&a\_d, size);}

Copying CPU → GPU
1 \texttt{cudaMemcpy(a\_d, a\_h, size, cudaMemcpyHostToDevice);}

Copying CPU ← GPU
1 \texttt{cudaMemcpy(a\_h, a\_d, size, cudaMemcpyDeviceToHost);}
Outline

CPU
- CPU Architecture

GPU
- GPU Architecture

CUDA Architecture
- Existing GPGPU frameworks

GPU programming
- Data types and kernel
- Grid, blocks, threads, memory

Examples
- Parallel HelloWorld
- N body problem (no interaction) $O(N)$
- N body problem with interaction $O(N^2)$

Summary
- Out of the main track
Outline

CPU
- CPU Architecture

GPU
- GPU Architecture

CUDA Architecture
- Existing GPGPU frameworks

GPU programming
- Data types and kernel
- Grid, blocks, threads, memory

Examples
- Parallel HelloWorld
  - N body problem (no interaction) $O(N)$
  - N body problem with interaction $O(N^2)$

Summary
- Out of the main track
Hello World

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

__device__ char napis_device[14];

__global__ void helloWorldOnDevice(void)
{
    napis_device[0] = 'H';
    napis_device[1] = 'e';
    ...  
    napis_device[12] = '\n';
    napis_device[13] = 0;
}

int main(void)
{
    helloWorldOnDevice <<< 1, 1 >>> ();
    cudaThreadSynchronize();

    char napis_host[14];
    const char *symbol="napis_device";
    cudaMemcpyFromSymbol (napis_host, symbol, sizeof(char)*13, 0, cudaMemcpyDeviceToHost);

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

Result: 'Hello World!' written by GPU.

Maciej Matyka

GPGPU programming on example of CUDA
Parallel HelloWorld

Parallel version of 'Hello World!':

- no changes in functionality
- we split the task between cores
- let each 'Hello World!' letter will be written by different core!

(Together with Paweł Czubiński)
Parallel version of Hello World

Kernel split to 14 blocks 1 thread each:

```c
1  helloWorldOnDevice <<< 14, 1 >>> ();
```

'Hello World! \n 0.'
Hello World

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

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

int main(void)
{
    helloWorldOnDevice <<< 14, 1 >>> ();
    cudaThreadSynchronize();
    ...
}
```

- Result: 'Hello World!' written by GPU in parallel!
- Each letter written by different core (if there were free resources available).
Outline

CPU
   CPU Architecture

GPU
   GPU Architecture

CUDA Architecture
   Existing GPGPU frameworks

GPU programming
   Data types and kernel
   Grid, blocks, threads, memory

Examples
   Parallel HelloWorld
   N body problem (no interaction) \( O(N) \)
   N body problem with interaction \( O(N^2) \)

Summary
   Out of the main track
N body problem (no interaction) $O(N)$

- $N$ non interacting material points
- gravity force $g = (0, g_y, 0)$
- + attraction to specified point
- computational loop:
  1. force calculation
  2. integration of equations of motion
- a little number of FLOPS per memory access!
- linear complexity growth - $O(N)$
Solution 1. CPU

```c
for(int i=0; i < N; i++)
{
    DIST = sqrt( (_ATR.x - pos[i].x)*(_ATR.x - pos[i].x)
                  + (_ATR.y - pos[i].y)*(_ATR.y - pos[i].y));

    if(DIST)
    {
        ATRF.x = KD * ((_ATR.x - pos[i].x))/DIST;
        ATRF.y = KD * ((_ATR.y - pos[i].y))/DIST;
    }

    vel[i].x = vel[i].x + (G.x + ATRF.x) * dt;
    vel[i].y = vel[i].y + (G.y + ATRF.y) * dt;

    pos[i].x = pos[i].x + vel[i].x * dt;
    pos[i].y = pos[i].y + vel[i].y * dt;

    if(pos[i].x >1) {pos[i].x = 1; vel[i].x = -vel[i].x;}
    if(pos[i].y >1) {pos[i].y = 1; vel[i].y = -vel[i].y;}
    if(pos[i].x <-1) {pos[i].x = -1; vel[i].x = -vel[i].x;}
    if(pos[i].y <-1) {pos[i].y = -1; vel[i].y = -vel[i].y;}
}
```
Solution 2. GPU, float3

```c
__global__ void movepartOnDevice
(float3* posvbo, float3 *vel, ...)
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;

    float3 posidx;
    float3 velidx;

    (...)

    if (idx < _N)
    {
        posidx = posvbo[idx];
        velidx = vel[idx];

        // -------------------------------------
        // Here: loop as in CPU code, exchange
        // pos[i] -> posidx
        // vel[i] -> velidx
        // -------------------------------------

        posvbo[idx] = posidx;
        vel[idx] = velidx;
    }
}
```

CPU | GPU | CUDA Architecture | GPU programming | Examples | Summary
---|---|---|---|---|---
N body problem (no interaction) $O(N)$

- $x2$
- memory access!
Coalesced memory access

Left: coalesced float memory access, resulting in a single memory transaction.
Right: coalesced float memory access (wrong way), resulting in 16 memory transactions.

Figure 5-1. Examples of Coalesced Global Memory Access Patterns

Figure 5-2. Examples of Global Memory Access Patterns That Are Non-Coalesced for Devices of Compute Capability 1.X
N body problem (no interaction) $O(N)$

### Solution 3. GPU, float4

```c
__global__ void movepartOnDevice
(float4* posvbo, float4 *vel, ...)
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;

    float4 posidx;
    float4 velidx;
    (...)

    if (idx < _N)
    {
        posidx = posvbo[idx];
        velidx = vel[idx];

        // -------------------------------------
        // Here: loop as in CPU code, exchange
        // pos[i] -> posidx
        // vel[i] -> velidx
        // -------------------------------------

        posvbo[idx] = posidx;
        vel[idx] = velidx;
    }
}
```

- x10
- order of magnitude!
Complexity $O(N)$, OpenGL visualization, GPU vs CPU

- 0.5 milion of points
- realtime visualization
- OpenGL
- GL Utility Toolkit (GLUT)
- Vertex Buffer Object (VBO)
N body problem (no interaction) $O(N)$

Complexity $O(N)$, comparison

- linear complexity
- GPU $\rightarrow$ device memory $>$ CPU $\rightarrow$ host memory.
- coalesced access $\rightarrow$ x10!
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
N body problem with interaction $O(N^2)$

- $N$ interacting point masses
- pairwise interaction
- computational loop:
  1. force calculation (double)
  2. integration of equations of motion
N body problem with interaction $O(N^2)$

**Solution 1. CPU, N=32000**

```c
(...)

for(int i=0; i < N; i++)
  for(int j=0; j < N; j++)
  {
    n.x = pos[j].x - pos[i].x;
    n.y = pos[j].y - pos[i].y;

    DIST = sqrt(n.x * n.x + n.y * n.y + EPS2);

    // fg = n*G*m1m2/r**3

    forc[i].x += (n.x * BIGG / DIST*DIST*DIST);
    forc[i].y += (n.y * BIGG / DIST*DIST*DIST);
  }
(...)
```
Solution 2. GPU, N=32000

```c
__global__ void moveparticlesOnDevice
(float4* posvbo, float4 *vel, ...)
{
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    (....)
    if (idx<_N)
    {
        posidx = posvbo[idx];
        forc.x = 0;
        forc.y = 0;
        for(int j=0; j < _N; j++)
        {
            // fg = n*G*m1m2/r**3
            // between posidx, and posvbo[j]
            // (analogy to CPU code)
            (....)
        }
    }
    (....)
}
```
N body problem with interaction $O(N^2)$

N body algorithm with shared memory

Effective algorithm of N body for CUDA:

- Lars Nyland, Mark Harris, Jan Prins
  Fast N-Body Simulation with CUDA
  GPU Gems 3

- use of shared memory
- each thread calculates N forces
- $p$ - number of threads in a block
- each thread runs a loop which calculates N/p forces ($p$ times)
- these N/p forces are calculated using data stored in shared memory
- thread/block - place in memory mapping
Solution 3. GPU - shared memory, N=32000

- x100!
- 2 orders of magnitude!
N body problem with interaction $O(N^2)$

Complexity $O(N^2)$, OpenGL visualization, GPU vs CPU

- 5835 points
- realtime visualization
- OpenGL
- GL Utility Toolkit (GLUT)
- Vertex Buffer Object (VBO)
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) $O(N)$
  N body problem with interaction $O(N^2)$

Summary
  Out of the main track
Summary

Summary:

- memory access optimization - crucial
- algorithm rebuilding - needed (work for programmers)
- shared memory - 16kb of registers (!)
- a lot of work - worth to do.
- Future? GTX300 - complete change of architecture? (!)

Attention: In that seminar we didn't say anything about compute capabilities, atomic operations, threads synchronization, vertex and pixel bufor, textures, loops unrolling, memory bank conflicts, warps etc..

Source: 'nVidia CUDA, Programming Guide 2.2', nVidia
Sources

- [http://4.bp.blogspot.com/_hqgVFA7RYE4/SkAI1b2QGII/AAAAAAAABd4/Q5mhupFHCA/s400/Hun+Sen+shaking+hand+with+Abhisit+(Reuters).jpg](http://4.bp.blogspot.com/_hqgVFA7RYE4/SkAI1b2QGII/AAAAAAAABd4/Q5mhupFHCA/s400/Hun+Sen+shaking+hand+with+Abhisit+(Reuters).jpg)
- [http://www.ipipan.gda.pl/ wiesiek/pjwstk/08-09/ark/wyklad-03.pdf](http://www.ipipan.gda.pl/ wiesiek/pjwstk/08-09/ark/wyklad-03.pdf)
- [http://arstechnica.com/hardware/reviews/2008/06/nvidia-geforce-gx2-review.ars](http://arstechnica.com/hardware/reviews/2008/06/nvidia-geforce-gx2-review.ars)
- 'nVidia CUDA, Programming Guide 2.2', nVidia
- Wikipedia
- Zbigniew Koza, 'CUDA na Twoim biurku, czyli słów kilka na temat Compute Unified Device Architecture', Wykład
Outline

CPU
  CPU Architecture

GPU
  GPU Architecture

CUDA Architecture
  Existing GPGPU frameworks

GPU programming
  Data types and kernel
  Grid, blocks, threads, memory

Examples
  Parallel HelloWorld
  N body problem (no interaction) \(O(N)\)
  N body problem with interaction \(O(N^2)\)

Summary
  Out of the main track
Simple model of granular matter:

N. Bell, Y. Yu and P. J. Mucha,
Particle-Based Simulation of Granular Materials,
Eurographics/ACM SIGGRAPH (2005)

spring interaction between masses

Realtime, CUDA, 4096 grains, [run]
The End

Thank you for your attention!