GPU Programming

Prof. Gilberto Díaz
gjdiazt@uis.edu.co

Universidad Industrial de Santander
Bucaramanga – Santander
Motivation

• The insatiable market demand for real-time, high-definition 3D graphics capability.

• Led the development of Programmable Graphics Processing Units (GPU, Graphics Processor Unit)
Motivation

- Today's GPUs have evolved into devices with sophisticated capabilities:
  - Parallel processing.
  - Multi threads.
  - High bandwidth communication to RAM.
  - Huge amount of processor units.
Motivation

Source: https://www.nextplatform.com
Currently, any system has more than one processor (**multi-core systems**) in the same silicon chip.

- GPUs have a big amount of cores (**many core systems**).
Hardware Specifications

- The performance difference is due to:
  - In a GPU most of the transistors are dedicated to processing.
  - In a CPU there are many transistors dedicated to control: data and instruction flow, cache memory, etc.
GPUs are designed for problems where data parallelism is the best approach.

One instruction

Many Data

Processor

Many results
• Programs running on a GPU are generally arithmetic expression intensive.

• Thus, memory latency can be compensated with computations instead of large caches.
Parallel Processing

- It maps each data element to threads
Data Parallelism

\[
\begin{pmatrix}
a_{00} & a_{01} & a_{02} \\
a_{10} & a_{11} & a_{12} \\
a_{20} & a_{21} & a_{22}
\end{pmatrix}
\times
\begin{pmatrix}
b_{00} & b_{01} & b_{02} \\
b_{10} & b_{11} & b_{12} \\
b_{20} & b_{21} & b_{22}
\end{pmatrix}
= 
\begin{pmatrix}
c_{00} & c_{01} & c_{02} \\
c_{10} & c_{11} & c_{12} \\
c_{20} & c_{21} & c_{22}
\end{pmatrix}
\]
**Parallel Processing**

**Data Parallelism**

\[
\begin{bmatrix}
a_{00} & a_{01} & a_{02} \\
a_{10} & a_{11} & a_{12} \\
a_{20} & a_{21} & a_{22} \\
\end{bmatrix}
\times
\begin{bmatrix}
b_{00} & b_{01} & b_{02} \\
b_{10} & b_{11} & b_{12} \\
b_{20} & b_{21} & b_{22} \\
\end{bmatrix}
= 
\begin{bmatrix}
c_{00} & c_{01} & c_{02} \\
c_{10} & c_{11} & c_{12} \\
c_{20} & c_{21} & c_{22} \\
\end{bmatrix}
\]
Data Parallelism

\[
\begin{bmatrix}
a_{00} & a_{01} & a_{02} \\
a_{10} & a_{11} & a_{12} \\
a_{20} & a_{21} & a_{22} \\
\end{bmatrix}
\times
\begin{bmatrix}
b_{00} & b_{01} & b_{02} \\
b_{10} & b_{11} & b_{12} \\
b_{20} & b_{21} & b_{22} \\
\end{bmatrix}
= 
\begin{bmatrix}
c_{00} & c_{01} & c_{02} \\
c_{10} & c_{11} & c_{12} \\
c_{20} & c_{21} & c_{22} \\
\end{bmatrix}
\]
Data Parallelism

\[
\begin{align*}
C_{00} &= a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} \\
C_{01} &= a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} \\
C_{02} &= a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\
\cdots \\
C_{22} &= a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}
\end{align*}
\]
Parallel Processing

Data Parallelism

\[
C_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}
\]

\[
C_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}
\]

\[
C_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}
\]

\[
C_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}
\]
Application Areas

- Image processing.
- Video encoding.
- Patterns recognition.
- Artificial intelligence.
- Scientific computing.
Compute Unified Device Architecture (CUDA)

Prof. Gilberto Díaz
gjdiazt@uis.edu.co

Universidad Industrial de Santander
Bucaramanga – Santander
Programación Paralela y Distribuida
CUDA comes with a software environment that allows developers to use C++ as a high-level programming language.

Definition

- CUDA is a technology that enables the massively parallel processing power of NVIDIA GPUs.

- It is a programming model developed by NVIDIA for its GPUs.
CUDA was introduced in November 2006 with a new programming model and a particular set of instructions.
Characteristics

- No knowledge of APIs is needed for GPU programming.
- Minimum effort in learning new programming elements.
- Provides access to the GPU instruction set.
Characteristics

- Provides direct access to the GPU memory.
- The programming model is designed to take advantage of the parallel resources of current systems.
Requirements

- Programming in CUDA requires an NVIDIA graphics card.
  - A100
  - HDR100
  - Tesla V100
Corresponding driver for the graphics card.

CUDA toolkit (compiler and libraries).
CUDA Goals

CUDA was developed with the following goals in mind:

- Provide a small set of extensions for traditional languages like C.
- Support for heterogeneous computations so that applications can use both the GPU and the CPU.
  - The serial portion in the CPU
  - The parallel portion on the GPU
The CPU and GPU are treated as separate devices, each with their own memory space.
Introduction

CUDA enables the computing power of graphics cards through APIs such as:

- OpenCL
- DirectX Compute

Or through high level languages like:

- C/C++
- Fortran
- Python, etc.
Architecture

NVIDIA Hardware
Parallel Compute Engines

Operating System Support

CUDA driver
PTX (ISA)

Application DirectX
HLSL Compute Shaders
DirectX Compute

Application OpenCL
OpenCL C Compute Kernels
OpenCL Driver

Application CUDA
C for CUDA Compute Kernels
C for CUDA Compute functions
C Runtime For CUDA

Application C, C++, Fortran, Python

C, C++, Fortran, Python

CUDA C Runtime

HLSL Compute Shaders
OpenCL C Compute Kernels
C for CUDA Compute Kernels
C for CUDA Compute functions
C Runtime For CUDA

CUDA poses three fundamental abstractions:

- A hierarchy of groups of threads (threads).
- Shared memory.
- Synchronization barrier.
• These abstractions provide two specific types of parallelism embedded within two other types of parallelism:

**CPU**
- Data parallelism. (Coarse grain)
- Functional parallelism

**GPU**
- Data parallelism (Fine grain)
- Thread parallelism
Programming Model

• This provides the programmer with a guide to the process of partitioning the problem into relatively thick subproblems.

• Each subproblem can be solved independently by using thread blocks.
• Then, each subproblem can be divided into smaller pieces.

• That can be solved in parallel, collaboratively, by the threads within each block.
CUDA thread hierarchy

Grid

0,0 0,1 0,2
1,0 1,1 1,2
2,0 2,1 2,2

Block

0,0 0,1
1,0 1,1

Thread
# Variable Types

<table>
<thead>
<tr>
<th>Variable declaration</th>
<th>Memory</th>
<th>Scope</th>
<th>Lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int var;</code></td>
<td>register</td>
<td>thread</td>
<td>thread</td>
</tr>
<tr>
<td><code>int array_var[10];</code></td>
<td>local</td>
<td>thread</td>
<td>thread</td>
</tr>
<tr>
<td><code>__shared__ int shared_var;</code></td>
<td>shared</td>
<td>block</td>
<td>block</td>
</tr>
<tr>
<td><code>__device__ int global_var;</code></td>
<td>global</td>
<td>grid</td>
<td>application</td>
</tr>
<tr>
<td><code>__constant__ int constant_var;</code></td>
<td>constant</td>
<td>grid</td>
<td>application</td>
</tr>
</tbody>
</table>
#define N 20
float c[N][N];

void mulmat(float a[N][N], float b[N][N]){
    int i,j,k;
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            for(k=0; k<N; k++)
                c[i][j] = c[i][j] + a[i][k]*b[k][j];
}

Example
__global__ void MatrixMult(float *Md, float *Nd, float *Pd, int Width) {
    // Cálculo del índice de fila de Pd y M
    int Row blockIdx.y * TILE_WIDTH + threadIdx.y;
    // Cálculo del índice de columna de Pd y N
    int Col blockIdx.x * TILE_WIDTH + threadIdx.x;
    float Pvalue = 0;
    for (int k = 0; k < Width; ++k) {
        Pvalue += Md[Row*Width+k] * Nd[k*Width+Col];
        Pd[Row*Width*Col] = Pvalue;
    }
}
CUDA Installation Process

Prof. Gilberto Díaz
gjdiazt@uis.edu.co

Universidad Industrial de Santander
Bucaramanga – Santander
Programación Paralela y Distribuida
CUDA Installation Process

Download cuda-toolkit.

cd /usr/local/src

CUDA Installation Process

Install the software

./cuda_12.1.1_530.30.02_linux.run

Enter install path (default /usr/local/cuda, '/cuda' will be appended): ENTER
CUDA Installation Process

Add environment variable

/etc/profile (global)
.bashrc (user)

```bash
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/usr/local/cuda/lib
```
Compiling Programs

```
nvcc sourceCode.cu -o execName

./execName
```