Nvidia GPU Computing Era

Berlusconi Mattia 754720
Mauri Andrea 754721
Contents

• Why GPU computing?
• Previous architectures
• CUDA
• Fermi architecture
Why GPU computing?

- GPU faced **parallelism** problem from the very beginning
- Even the **first architectures** were designed to exploit parallelism
- The first step was **exposing** the architecture to the developer
- The real evolution was creating a **new architecture**
The graphic pipeline

**INPUT**: list of geometric primitives

**OUTPUT**: final picture

The **canonical graphic pipeline** is composed by these stages:

1. Transformation
2. Pre-vertex lighting
3. Viewing transformation
4. Primitive generation
5. Projection transformation
6. Clipping
7. Viewport transformation
8. Rasterization
9. Texturing
10. Display

Vertex and Fragment operations were **not programmable**
Previous architectures

G80 Architecture

• First step towards GPU general purpose computing
• Single unified steaming multiprocessor
• Scalar thread processor
• Single Instruction Multiple Thread execution model
• Shared memory and barrier synchronization for inter-thread communication
• C language support
Previous Architectures

**GT200 Architecture**

- Only a refinement of the G80 architecture
- Increased the number of the stream processor’s core (CUDA core)
- Register size doubled
- Double point float precision added
CUDA

• **Compute Unified Device Architecture**

• **Allow to use GPU for general purpose computing**

• **C programs with a few simple extensions can be executed on GPU**

• **Programs written for single thread execution will run in many parallel threads**
CUDA programs consist of a function **kernel**. Instances of a kernel are executed on many parallel threads. Threads are organized in **threads block**. Thread blocks executing the same kernel are grouped in a **grid**.

- Each thread has a private local memory.
- Each block has a per-block shared memory.
- Each grid has a per-application global memory.
Fermi architecture

- Up to 512 CUDA cores
- The cores are organized in 16 Streaming Multiprocessors of 32 cores
- GigaThread deploy thread blocks to SM
Each SM features:

- 32 CUDA cores
- 16 load/store units
- 4 special function units
- 64KB configurable memory
- 128KB register file
- An instruction cache
- 2 multi-thread scheduler (warp scheduler)
• 16 load/store permit 16 memory operation per clock
• SFUs are able to compute transcendental functions
• A SM can schedule a group 32 parallel threads called “warp”
• A SM can issue and execute concurrently 2 warps
• Each warp scheduler issue one instruction from a warp to a group of 16 core, 16 l/s units or 4 SFUs
• The 64KB memory can be configured in 2 way
CUDA core

Each Cuda core feature:

- An ALU unit
- A FPU unit

CUDA core implements standard and double precision
PTX

- **Parallel Thread Execution Instruction Set Architecture**
- Pseudo – Assembly language
- Support operation of a parallel thread processor
- PTX instructions are translated into machine language by a compiler contained in the GPU driver
PTX – Unified Space Address

- Single continuous address space that unifies:
  - Local thread private
  - Block shared
  - Global

- 40bit of unified address space
- 64bit instruction set for future growth
Memory system

- Per processor configurable L1 cache
- Shared L2 cache
- Main DRAM memory
SM memory has 2 possible configurations:
- 48KB of shared memory and 16KB L1 cache
- 16KB of shared memory and 48KB L1 cache

L2 cache is shared among all the GPU and provides a 768KB

The main memory is a GDDR5 DRAM and support up to 6GB
ECC Support

- Enhance data integrity in high performance computer environment
- Detect and correct single-bit soft error
- Detect double and multi bit error and report them
GigaThread

- At chip level distributes the **thread block** to various SM
- At SM level distributes **warps** to the corresponding functional unit

- Allow **fast context switching**
- Allow **concurrent kernel execution**
Thanks for the attention