Course Notes on CUDA

CS599

03/23/2011

Yongqiang Li

Introduction of the speaker

Bachelor degree was obtained from UIUC.

Master degree was obtained from UC Berkeley.

He is a leader in one group of CUDA R&D.

He was employed in CUDA,2006.

He is responsible for development of DirectX 9 and 10 Drivers.

Background and Motivation of CUDA

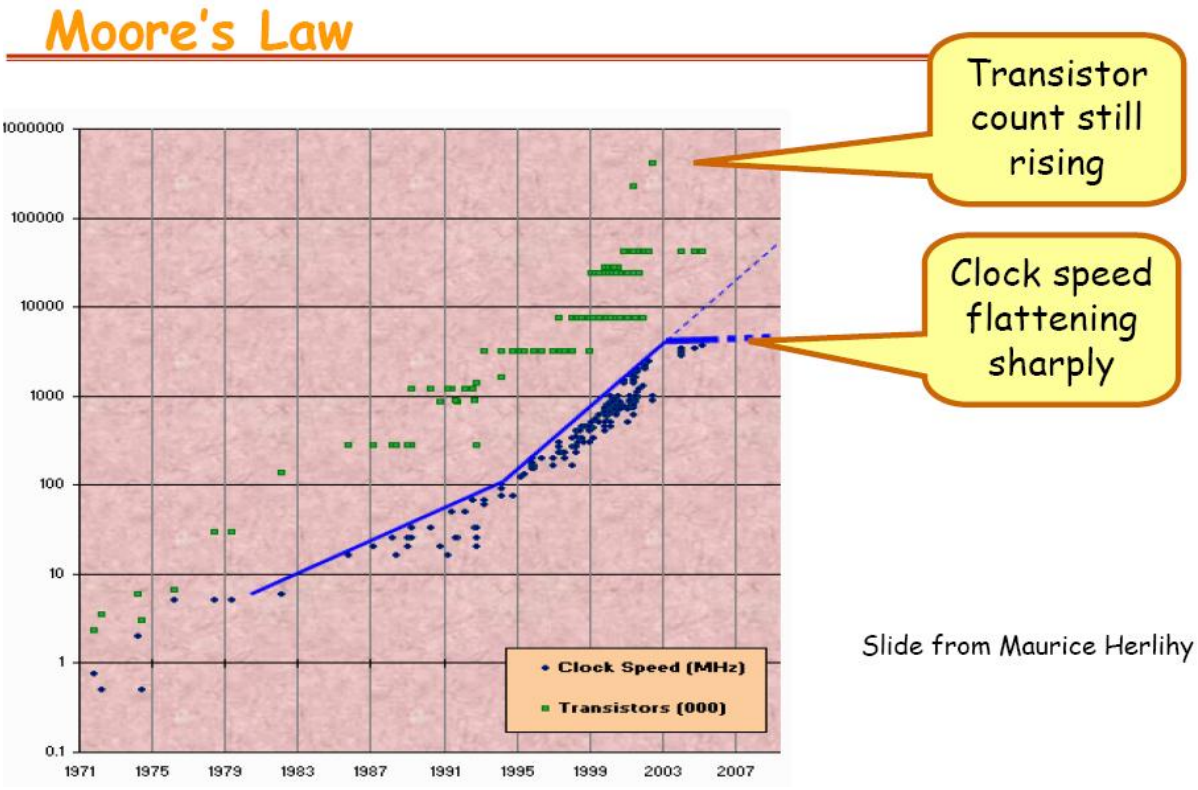

Fig1. Moore's Law, cited from slide of CS503, 2010, USC

Transistor count in a chip doubles every 18 months. This performance doubling stopped not to be true about five years ago. Because clock speed flattened sharply as shown in Figure 1. In addition, the more transistors, the more power is cost. It is not power saving to add more transistors to augment the performance of a computer. Parallelism costs less power than add more cores. Therefore parallelism is the way forward.

GPU Programming Trend

CIDA is an extension to C for programming GPUs

Tesla server processes

NVIDIA GPUs in super powerful computers

Top 500 LINPACK benchmark -> 3 out of top 4 use NVIDIA GPUs

Green 500 -> 2 out of 4 use NVIDIA

Real-world Application Utilizes CUDA to improve Performance

ANSYS, Matlab, Adobe, Autodesk…

Historical GPU Computing Using OpenGL

Programming Graphics Pipeline(see Monday lecture)

GPGPU

OpenGL Programming treats each pixel of a texture as a thread

OPENGL limitations

Threads have specific output

Floating-point arithmetic only

Limited instruction count

Limited control flow

No cross-thread communication

GPU Architecture (low latency or high Throughput)

CPU <- low latency

GPU<-more transistors to computation

GPU hides latency with computation

Two main components

Global memory(currently up to 6 GB)

Streaming Multiprocessor(SM) -> most interesting

L2 -> cache , repeated units

Fermi: 32 CUDA cores

CUDA CORE

2 warp schedulers

32K 32-bit registers

64K shared mem+L1 cache

4 special – function Units

Cos, sin, square root, hardware to do the computation

Fermi: Memory system

CUDA Programming: Launching Threads

Serial code -> host (CPU)

Parallel code -> device(GPU)

: C with a few key words

Function qualifier

host

global

device

Kernel: functions called by the host that executes on GPU

Thread blocks allow scalability

The shallower the function is, the better. GPU function should be short, the wider threads are, the better.

Maximum # of blocks

Some case: 64,000.

# of threads: 512

CUDA Programming Example: AXPY

Memory hierarchy

Thread: 1) Registers

2) Local memory

Block of threads: shared memory

All blocks, Global memory (local, peer devices, host)

Per-Block Shared Memory

shared

Latency : a few clock cycles

bandwidth: 1.03 TB / s

: Device Global Memory

cudaMalloc();

cudaMemSet();

cudaFree();

Latency: 400~800 cycles

bandwidth: 156 GB/s

: Host, Peer, Global Memory

Latency: many cycles

bandwidth: 6GB/s

:Host, Peer Global Memory

Latency: many cycles

bandwidth: 6 GB/s

Memory hierach: Peer Topologies

1). Global Memory Gotches,malloc() X

2).Host cannot access device memory

PCIE latency is horrific

CUDA Programming Example: GEMM

GPGPU Tools or Libraries

BLAs

FFT

Textbook and lecture notes:

http://courses.engr.illinois.edu/ece498/al/

Debugs

1). Windows: Parallel N sight

2). Linux:cuda-gdb

3).Runs on the GPU

Questions:

Q:How to schedule parallel tasks to different threads in GPU?

A:You can assign different tasks to the different blocks or threads.

Q:Does CUDA support dynamic arrays?

A:Not yet.

Q: What's the maximal # of blocks and threads?

A: Threads # are 512, maximal # of blocks depends on different GPU, i.e. 64,000(the range of 32 bit interger).