Introduction

Jan Lemeire
Dept. ETRO
September 23th 2014

Jan Lemeire (jan.lemeire@vub.ac.be)

- Graduated as Engineer in 1994 at VUB
- Worked for 4 years for 2 IT-consultancy companies
- 2000-2007: PhD at the VUB while teaching as assistant
  - Subject: probabilistic models for the performance analysis of parallel programs
- Since 2008: postdoc en parttime professor at VUB, department of electronics and informatics (ETRO)
  - Teaching 'Informatics' for first-year bachelors; 'parallel systems' and 'advanced computer architecture' to masters
- Since October 2012: also teaching for engineers industrial sciences ('industrial engineers')
- Projects, papers, phd students in parallel processing (performance analysis, GPU computing) & data mining/machine learning (probabilistic models, causality, learning algorithms)
- http://parallel.vub.ac.be

Goals of course

- Understand architecture of modern parallel systems.
- Employ software technologies for parallel programming.
- Design efficient and two-fold generic parallel solutions.
  - For a wide variety of parallel systems & broad class of similar algorithms.
  - Sharpen your low-level and high-level IT skills.
- Understand their performance.
- Make successful technology. Understand economics.
The Free Lunch is Over

Why You Don't Have 10GHz Today?
- heat/surface is the problem (power wall)
- 12 ns would mean electric paths of 10 atoms wide

Moreover:
- memory bottleneck
- instruction level parallelism (ILP) wall

What about Moore's Law?
- increase of Clock speed: stopped
- increase of Transistors: ongoing

It's now about the number of cores!

http://www.gotw.ca/publications/concurrency-ddj.htm

Floating-Point Operations per Second for the CPU and GPU

Collection of graphical cards

FASTRA 8 cards = 8x128 processors = 4000 euro

Similar performance as University's supercomputer (512 regular desktop PCs) that costed 3.5 million euro in 2005

http://fastra.ua.ac.be
GPU peak performance has grown aggressively. Hardware has kept up with Moore’s law.

Source: NVIDIA

2010
350 Million triangles/second
3 Billion transistors GPU

1995
3,000 triangles/second
800,000 transistors GPU

Why are GPUs faster?

- Light-weight threads, supported by the hardware
  - Thread processors, up to 96 threads per processor
  - Context switch can happen in 1 cycle!

- No caching mechanism, branch prediction, ...
  - GPU does not try to be efficient for every program, does not spend transistors on optimization
  - Simple straight-forward sequential programming should be abandoned...

- Less higher-level memory:
  - GPU: 16KB shared memory per SIMD multiprocessor
  - CPU: L2 cache contains several MB’s

- Massively floating-point computation power

- Transparent system organization
  - Modern (sequential) CPUs based on simple Von Neumann architecture

So...

GP-GPUs: Graphics Processing Units for General-Purpose programming
Usage

- Copy data from CPU to GPU
- Start kernel within CPU-program (C, java, Matlab, python, ...)
  - Several kernels can be launched (pipelined)
  - Handled on the GPU one by one or in parallel
- Figure

GPU Architecture

- In the GTX 280, there are 10 Thread Processing Clusters
  - Each has 3 Streaming Multiprocessors, which we will refer to as **multiprocessors (MPs)**.
  - Each MP has 8 Thread Processors. We will refer to these as **Scalar Processors (SP)**.
  - 240 processor cores and 30 MPs in total!
- One double-precision unit (DP) per MP

Host (CPU) – Device (GPU)

- Hypertransport and Intel's Quickpath currently 25.6 GB/s
- PCIe x16 4 GB/s
- PCIe x16 Gen2 8 GB/s peak
- Nvidia Tesla C2050: 1030.4 GB/s
1 Streaming Multiprocessor

The Same Instruction is executed on Multiple Data (SIMD)
width of pipeline: 8 - 32

GPU vs CPU: NVIDIA 280 vs Intel i7 860

<table>
<thead>
<tr>
<th></th>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>16,384 (32-bit) / multi-processor(^3)</td>
<td>128 reservation stations</td>
</tr>
<tr>
<td>Peak memory bandwidth</td>
<td>141.7 Gb/sec</td>
<td>21 Gb/sec</td>
</tr>
<tr>
<td>Peak GFLOPs</td>
<td>562 (float) / 77 (double)</td>
<td>50 (double)</td>
</tr>
<tr>
<td>Cores</td>
<td>240 (scalar processors)</td>
<td>4/8 (hyperthreaded)</td>
</tr>
<tr>
<td>Processor Clock (MHz)</td>
<td>1296</td>
<td>2800</td>
</tr>
<tr>
<td>Memory</td>
<td>1Gb</td>
<td>16Gb</td>
</tr>
<tr>
<td>Local/shared memory</td>
<td>16Kb/TPC(^2)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

\(^1\)http://ark.intel.com/Product.aspx?id=41316
\(^2\)TPC = Thread Processing Cluster (24 cores)
\(^3\)30 multi-processors in a 280

Performance: GFlops?

- GPUs consist of MultiProcessors (MPs) grouping a number of Scalar Processors (SPs)
- Nvidia GTX 280:
  - 30MPs x 8 SPs/MP x 2FLOPs/instr/SP x 1 instr/clock x 1.3 GHz = 624 GFlops
- Nvidia Tesla C2050:
  - 14 MPs x 32 SPs/MP x 2FLOPs/instr/SP x 1 instr/clock x 1.15 GHz (clocks per second) = 1030 GFlops

Other limit: bandwidth

- Nvidia GTX 280:
  - 1.1 GHz memory clock
  - 141 GB/s
- Nvidia Tesla C2050:
  - 1.5 GHz memory clock
  - 144 GB/s
**Example**: real-time image processing

Images of 20 MegaPixels

CPU gives only 4 fps
next generation machines need 50 fps
GPUs deliver 70 fps

---

**Pixel transformation**

- Performance on Tesla C2050
- 1 pixel is represented by 1 byte [0-255]
- Integer operations: performance is half of floating point operations
- **Two different implementations:**
  - FPN1: 1 pixel per thread
  - FPN4: 4 pixels per thread (treat 4 bytes as 1 ‘word’)

<table>
<thead>
<tr>
<th>$P_{\text{mem}}$ (bytes/s)</th>
<th>115 GB/s</th>
<th>$P_{\text{ops}}$ (ops/s)</th>
<th>500 Gops/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI (bytes/pix)</td>
<td>1/5</td>
<td>Ops/pix</td>
<td>5+4 (FPN1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5+1 (FPN4)</td>
</tr>
<tr>
<td>$P_{\text{mem} \times CI}$ (pix/s)</td>
<td>23 Gpix/s</td>
<td>$P_{\text{ops}}/(\text{Ops/pix})$</td>
<td>56 Gpix/s (FPN1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>83 Gpix/s (FPN4)</td>
</tr>
</tbody>
</table>

---

**But... nothing is for free**

- **Harder to program!**
  - Hardware architecture should be taken into account
  - Optimization is important
  - Additional complexity in code
  - Harder to debug, maintain, ...

- Algorithms should contain inherently massively fine-grained parallelism