The GPU on heterogeneous and high performance computing

Manuel Ujaldón Martínez
Computer Architecture Department
University of Malaga (Spain)
Outline

1. The GPU on heterogeneous computing [9]
2. The GPU on high-performance computing [7]
3. Programming the GPU for heterogeneous and high-performance computing [25]
4. A comparison versus shared and distributed memory platforms [16]
5. TESLA [10]
I. The GPU on heterogeneous computing

Multi-core architectures: 4-8 cores

Many-core architectures: 240-512 cores
GPUs have evolved to the point where many real world applications are easily implemented on them and run significantly faster than on multi-core systems. Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs.

Jack Dongarra
Professor, University of Tennessee
Author of Linpack
We’ve all heard ‘desktop supercomputer’ claims in the past, but this time it’s for real: NVIDIA and its partners will be delivering outstanding performance and broad applicability to the mainstream marketplace. Heterogeneous computing is what makes such a breakthrough possible.

Burton Smith
Technical Fellow, Microsoft
Formerly, Chief Scientist at Cray
The central axis for the next decade: Heterogeneous computing

Use CPU and GPU

Every processor executes those parts where it gets more efficient than its counterpart
The best of each computational model

Where GPUs are ahead:

- Very fast caches.
- Fine-grained branches.
- Many paradigms for executing threads and processes.
- High performance on a single thread of execution.
- Efficient I/O system.

TASK PARALLELISM

Where GPUs are ahead:

- DRAM is very fast.
- Hardware specialized in math operations.
- Microprogramming using shaders.
- High performance when executing parallel tasks.
- I/O constraints.

DATA PARALLELISM
The CPU-GPU on a cooperative environment

- High communication bandwidth:
  - External, up to 10 GB/s (PCI-express).
  - Internal, even more than 100 GB/s (video memory GDDR).
Scopes of application for the heterogeneous model

- Highly parallel computing
- Control and communication
- Productivity-based applications
- Data intensive applications

GPU (Parallel computing)

CPU (Sequential computing)

Oil & Gas
Finance
Medical
Biophysics
Numerics
Audio
Video
Imaging
How to find the optimal cooperation point in heterogeneous computing
Performance on heterogeneous computing: Discovery

- Computational chemistry: 4.6 days (CPU only), 27 minutes (CPU+GPU)
- Neurologic modelling: 2.7 days (CPU only), 30 minutes (CPU+GPU)
- Cell phones simul. in RF: 8 hours (CPU only), 13 minutes (CPU+GPU)
- 3D ultrasounds in tomography: 3 hours (CPU only), 16 minutes (CPU+GPU)
II. The GPU on high performance computing
Motivation

Architectural models based on simple cores are a valid alternative for building multiprocessors, and the industry has realized about it:

- Architectures are more scalable and very innovative, as they always find a place on the lucrative video-games marketplace.
- Leisure industry, in continuous growing, guarantees viability.
- Software is ubiquitous and popular, maintaining a solid group of programmers who can migrate towards GPGPU tasks.
- The performance/cost ratio is far better than supercomputers based on out-of-order cores (CPUs coming from Intel/AMD). An example follows...
Comparative (end of 2008)

CalcUA (supercomputer from Sun Microsystems, USA):
- Hardware: 256 dual-core CPUs.
- Cost: 5 million dollars.

FASTRA (Belgium):
- Hardware: 8 GPUs on a desktop PC.
- Cost: 6000 dollars.

Time spent on a 3D reconstruction in tomography
What high performance computing never had

Popularity and low cost: GPUs are ubiquitous and cheap.

Source: Nvidia (2008)
What high performance computing always had

- Programming challenges: Programming model is based on a predefined number of layers, on top of which efficiency is compromised.

- Distance between hardware and software: The block diagram has a huge potential, but software tools exploit just a tiny part of it.
Three lessons imported from graphics processing

- Do not rely on frequency for scalability, but on replicated cores, which can better take advantage of smaller transistors.
- Scalable multi-cores shine last decade inside GPUs much earlier than within CPUs, showing the right way.
- Sustained data bandwidth remains as one of the keys for high-performance. Some vendors (IBM-Cell, Nvidia-CUDA) offer an explicit management of the memory hierarchy, SRAM in particular.
How the GPU shows us those lessons

- To face the challenge of real-time images, GPU had to innovate to develop pioneering multicore chips.
- During those transitions, they built more general-purpose platforms than originally thought.
- Once the special-purpose was committed to general-purpose, programmer finds those elements he is used to: Data structures, branches, ...
- And that is how CUDA and OpenCL arrive, allowing the blooming current state of GPU programming.
III. Programming the GPU for heterogeneous and high-performance computing
A platform for heterogeneous and high-performance computing: BALE

- Purchased by the Ohio Supercomputing Center (June ‘07).
- It consists of 18 nodes running Linux, each endowed with:
  - Two dual-core Opteron CPUs from AMD @ 2.6 GHz.
  - 8 Gbytes of DRAM.
  - 750 Gbytes of disc space.
  - Two GPU Quadro FX 5600 by Nvidia, each with 1.5 Gbytes of video memory.
  - Nodes interconnected through Infiniband.
The BALE supercomputer: Cluster of 55+18 nodes, hybrid CPU-GPU

<table>
<thead>
<tr>
<th>Regular Nodes</th>
<th>Visualization Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td><strong>GPU</strong></td>
</tr>
<tr>
<td>Single-Socket, Dual-Core CPU, Single GPU</td>
<td>Dual-Socket, Dual-Core CPU, Dual GPU</td>
</tr>
</tbody>
</table>

**Configuration Details**

- **CPU**: AMD Opteron 6100 Series
- **GPU**: NVIDIA GeForce 459
- **Memory**: 8 GB DDR3
- **Storage**: 730 GB
- **Network**: 10 GbE interconnect
- **File System**: Parallel Virtual File System
Block diagram for each of the 18 visualization nodes

[Diagram of a block diagram for visualization nodes, showing various components such as PCI-E slots, CPU, NPF3050, ALC262, SATAII ports, EIDE(ATA/133) X1, USB2.0/10 ports 6 at rear, 3 at front, 1 internal, Dual GbE PHY 88E1121, 1394 rear(1)+front(2), TSB43AB22A, LPC, LPCROM, SIO6, SMSC 5307, Floppy, PS/2, Serial.]
Comparative CPU/GPU performance on computational raw power and memory BW.

<table>
<thead>
<tr>
<th>Processor</th>
<th>CPU (AMD)</th>
<th>GPU (Nvidia)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural model</td>
<td>Opteron X2 2218</td>
<td>Quadro FX 5600</td>
</tr>
<tr>
<td>Clock frequency</td>
<td>2.6 GHz</td>
<td>600 MHz / 1.35 GHz</td>
</tr>
<tr>
<td>Number of cores</td>
<td>2 cores</td>
<td>128 stream processors</td>
</tr>
<tr>
<td>Peak computational power</td>
<td>2 cores x 4.4 GFLOPS = 8.8 GFLOPS</td>
<td>madd(2 FLOPS) x128 SP x 1.35 GHz = 345.6 GFLOPS</td>
</tr>
<tr>
<td>Total (18 nodes x 2 sockets)</td>
<td>316.8 GFLOPS</td>
<td>ii 12.4 TFLOPS !!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th>CPU (AMD)</th>
<th>GPU (Nvidia)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity and type</td>
<td>8 Gbytes DDR2</td>
<td>1.5 Gbytes GDDR3</td>
</tr>
<tr>
<td>Clock frequency</td>
<td>2x 333 MHz</td>
<td>2x 800 MHz</td>
</tr>
<tr>
<td>Bus width</td>
<td>128 bits (dual channel)</td>
<td>384 bits</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>10.8 Gbytes/sg.</td>
<td>76.8 Gbytes/sg.</td>
</tr>
</tbody>
</table>
Languages and tools to be used by programmers

- **Within the context of GPU programming:**
  - 2003: OpenGL and Cg (vector C for shaders - Nvidia).
  - 2006: CUDA (GPGPU for NVIDIA platforms).
  - 2009: OpenCL (GPGPU for a wide range of graphics platforms).

- **Within the context of CPU parallel programming:**
  - pthreads on multi-core platforms.
  - MPI/OpenMP on multisocket platforms.

- Let’s see how all this evolves based on our own experience....
First, CPU code is ported from C into Cg
Second, we develop multicore versions on CPUs and manycore on GPUs.

<table>
<thead>
<tr>
<th>Programming techniques</th>
<th>Programming effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU/Host programming: Matlab or C++</td>
<td>Dual core CPU</td>
</tr>
<tr>
<td>Graphics programming: OpenGL+Cg or CUDA</td>
<td>Dual chip GPU</td>
</tr>
</tbody>
</table>
Third, we move to multinode platforms.
Last, (multi-core + many-core) x multi-node for a massive parallelism
Route followed by our first application:
Image feature extraction

**CODES:**
- MATLAB
- C++
- Cell Programming

**HARDWARE PLATFORM:**
- Single core CPU
- Dual core CPU
- Multi-node CPU and/or GPU chip
- Dual core GPU

**PROGRAMMING TECHNIQUES:**
- CPU/Host programming: Matlab or C++
- Graphics programming: OpenGL+Cg or CUDA
- Multicore programming: pthreads or OpenMP
- Data distribution/partitioning: Manual for using DataCutter
- Parallelism & communication: Message Passing or MPI

**Programming effort**
Route followed by our second application: Segmentation and image clustering
Route followed by our third application: Registration and 3D reconstruction
Segmentation and image clustering

Large-scale high resolution image

Image tiles (40x zoom)

Classification map

Computational Units

CPU  ...  GPU  ...  PS 3

Assign a classification tag

- Cancer tissue
- Healthy tissue
- Background
- Undetermined
The global process

Color conversion: GPU-like operators

Data mining: CPU-like operators

Training

Verify
Sequence of operations to carry out

- **Color conversion:** From RGB to LA*B*
  - Typical GPU operator, but CUDA shared memory still offers potential improvement.

- **Co-occurrence matrix for computing statistical features:**
  - Low data reuse.
  - Counters increment enforces memory load/stores.

- **LBP (Local Binary Pattern) operator:**
  - Uses a 3x3 pixels neighborhood window.

- **Histogram:**
  - Similar characterization than co-occurrence matrices.
## Experimental results on a single node (BALE supercomputer)

<table>
<thead>
<tr>
<th>Image size</th>
<th>Resolution (in pixels)</th>
<th>Number of 1K x 1K tiles on image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>32980 x 66426</td>
<td>33 x 65 = 2145</td>
</tr>
<tr>
<td>Medium</td>
<td>76543 x 63024</td>
<td>75 x 62 = 4659</td>
</tr>
<tr>
<td>Large</td>
<td>109110 x 80828</td>
<td>107 x 79 = 8453</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image size</th>
<th>On the CPU</th>
<th>On the GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Matlab</td>
<td>C++</td>
</tr>
<tr>
<td>Small</td>
<td>2h 57’ 29”</td>
<td>43’ 40”</td>
</tr>
<tr>
<td>Medium</td>
<td>6h 25’ 45”</td>
<td>1h 34’ 51”</td>
</tr>
<tr>
<td>Large</td>
<td>11h 39’ 28”</td>
<td>2h 51’ 23”</td>
</tr>
</tbody>
</table>
GPU cluster performance: Single/Multiple nodes, different image sizes

Single Node Image Analysis

Parallel Image Analysis

- Cg
- DC-Cg
- CUDA
- DC-CUDA
- CUDA (2 GPU)
- DC-CUDA (2 GPU)

Time (s)

SMALL  MEDIUM  LARGE

629.42 s

SMALL  MEDIUM  LARGE
Almost linear speed-up is achieved in all versions, with more stable results on medium and large images.
Concluding results for heterogeneous computing and massive parallelism

- With respect to the computational load of the hospital (8-10 biopsy samples per patient, 400 patients per year).
- On a single CPU, total processing time is:
  - 3.4 months, using C++ (2005).
- On a single GPU, the time gets reduced to:
  - 5.3 days, using Cg (2006).
  - 2.4 days, using CUDA (2007).
- Parallelizing the code on BALE supercomputer:
  - Less than 2 hours, using 16 nodes CPU-GPU (2008).
- Total speed-up exceeds six orders of magnitude.
Conclusions

- CUDA on GPUs and DataCutter on BALE allow us to exploit many levels of parallelism in biomedical image processing: Multinode, SMP and thread-level, SIMD and ILP.
- Our applications consist of a number of kernels which behave differently on CPUs and GPUs depending on features like arithmetic intensity, memory access or data reuse.
- In addition to the great speed-up achieved on a single node thanks to the GPU, almost linear speedup is obtained when all coexisting methods are combined on a CPU-GPU multiprocessor.
Our third application: Register and 3D reconstruction

Our main challenge is the huge volume of data to process:

- High resolution images (0.7-1.5 GB.)
- ~1200 images for each mouse.
- > 3 TB. for each mouse.
Final result

Placenta reconstruction
Scalability on BALE for heterogeneous computing

Combined CPU-GPU scalability

<table>
<thead>
<tr>
<th>Execution time (secs.)</th>
<th>mammary 1</th>
<th>mammary 2</th>
<th>mammary 3</th>
<th>mammary 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CPU + 1 GPU</td>
<td>306.02</td>
<td>131.13</td>
<td>579.80</td>
<td>227.51</td>
</tr>
<tr>
<td>2 CPUs + 2 GPUs</td>
<td>252.02</td>
<td>83.89</td>
<td>450.45</td>
<td>152.13</td>
</tr>
<tr>
<td>4 CPUs + 2 GPUs</td>
<td>102.81</td>
<td>85.83</td>
<td>213.09</td>
<td>51.72</td>
</tr>
<tr>
<td>8 CPUs + 4 GPUs</td>
<td>57.99</td>
<td>45.36</td>
<td>117.37</td>
<td>34</td>
</tr>
<tr>
<td>16 CPUs + 8 GPUs</td>
<td>32.24</td>
<td>26.74</td>
<td>71.28</td>
<td>18.54</td>
</tr>
<tr>
<td>32 CPUs + 16 GPUs</td>
<td>22.62</td>
<td>18.87</td>
<td>38.07</td>
<td>11.89</td>
</tr>
</tbody>
</table>
Performance on each BALE node

Dual core / socket / node performance

<table>
<thead>
<tr>
<th>Processors used</th>
<th>Execution time (secs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CPU core 0 GPU chips core alone</td>
<td>1,304,04</td>
</tr>
<tr>
<td>2 CPU cores 0 GPU chips cores on same socket</td>
<td>715,44</td>
</tr>
<tr>
<td>2 CPU cores 0 GPU chips cores on different sockets</td>
<td>568,63</td>
</tr>
<tr>
<td>4 CPU cores 0 GPU chips cores on different nodes</td>
<td>293,73</td>
</tr>
<tr>
<td>1 CPU core 1 GPU chip core alone</td>
<td>579,80</td>
</tr>
<tr>
<td>2 CPU cores 2 GPU chips cores on same socket</td>
<td>450,45</td>
</tr>
</tbody>
</table>
GPU influence on BALE performance

![Bar chart showing GPU influence on performance](image)

- No GPUs
- Half number of GPUs than CPU cores
- Same number of GPUs and CPUs

Execution time (secs.) vs. Number of CPU cores

- 1 core: 367.15
- 2 cores: 715.44
- 4 cores: 450.45
- 8 cores: 293.73
- 16 cores: 112.66
- 32 cores: 117.37
- 64 cores: 65.39
- 128 cores: 81.36
- 256 cores: 71.28
- 512 cores: 38.01
- 1024 cores: 47.45
- 2048 cores: 38.07
- 4096 cores: 26.61

1,304.04
IV. A comparison versus shared and distributed memory machines
Introduction. How a BlueGene/L supercomputer is built

**System**
- 64 Racks, 64x32x32
- 180/360 TFLOPS
- 32 TB

**Rack**
- 32 nodes

**Node**
- (32 chips 4x4x2)
- 16 compute, 0-2 I/O
- 2.8/5.6 TFLOPS
- 512 GB

**Board**
- 2 chips, 1x2x1
- 90/180 GFLOPS
- 16 GB
- 5.6/11.2 GFLOPS
- 1.0 GB

**Chip**
- 2 processors
- 2.8/5.6 GFLOPS
- 4 MB

**Introduction.**

How a BlueGene/L supercomputer is built.
The Q-norm algorithm

- It consists of a normalization process by statistical quantiles for a serie of genetic samples which comprises 470 vectors and 6.5 millions of data.
Phases of Q-norm process

1) Load data from file.
2) Sort each R column to produce X, and annotate I[G][E]=p (where p is the index of the original column).
3) Compute A[G], the average value for each row.
4) Assign the average value to each entries on X vector sorted.
5) Sort each output column O[g][E] by the I[g][E] index to reproduce the original permutation.
Q-norm: Pseudo-code

\[ nE = \text{LoadProject}(\text{fname}, \text{fList}); \]  // Step 1: I/O

\begin{verbatim}
for (i=0;i< nE;i++) { // Step 2: Sort columns
    \text{LoadFile}(\text{fList}, i, \text{dataIn});
    \text{Qnorm1}(\text{dataIn}, \text{dIndex}, \text{fList}[i].nG);
    \text{PartialRowAccum}(\text{AvG}, \text{dataIn}, \text{nG});
    // Manage index in memory or disc
}

for (i=0;i<nG;i++) // Step 3: Compute averages
    \text{AvG}[i].Av /=\text{AvG}[i].num;
\end{verbatim}

// Produce output file with vector data sorted for each column
for (i=0;i<nE;i++)
{
    \text{Obtain column index from disc}
    for (j=0;j<nG;j++) { // Prepare output vector
        \text{dataOut}[\text{dIndex}[j]]=\text{AvG}[j].Av;
        // File position to write output vector
    }
}

\begin{itemize}
    \item Input \textit{R}[G][E]
    \item Order by col \textit{X}[G] reusable
    \item Row average \textit{A}[G]
    \item \textbf{O}[i, l[i, j]] = A[i]
\end{itemize}
Hardware platforms where we execute the code

- **Distributed memory multiprocessor (“Pablo”):**
  - Cluster of 256 blades JS20 from IBM (512 CPUs). Each blade has:
    - 2 CPUs PowerPC 970FX @ 2 GHz.
    - 4 Gbytes of memory, for a total of 1 Tbyte of distributed memory.
  - Interconnection network: MIRYNET.

- **Shared memory multiprocessor (“Picasso”):**
  - HP Superdome of 16 nodes (128 cores). On each node:
    - 4 CPUs Intel Itanium-2 (alias “Montecito”) dual-core @ 1.6 GHz.
  - Total memory available: 384 Gbytes.
  - Interconnection network: Vendor specific.

- **A regular PC in our lab (“Antonio Banderas”):**
  - CPU: Intel Core 2 Quad Q9450, 2.66 GHz, 1.33 GHz FSB, 12 MB L2.
  - GPU: GeForce 9800 GX2, 600/1500 MHz, 512 MB de DDR3 a 2 GHz.
  - Hard disks: 2 x 72 GB (RAID 0) WD Raptors 10000 RPM.
Pablo (1/8 of former Mare Nostrum - 2004).
Cost around 600.000 €
Picasso (new HP Superdome - 2008).
Cost around 2 million euros
Picasso: Block diagram

Cell Controller

CPU

Cell 0

DIMM x2

DIMM x2

DIMM x2

DIMM x2

DIMM x2

DIMM x2

DIMM x2

DIMM x2

DIMM x2

8.5 GB/s (6.8)
(144@533 MT/s)

4.3 GB/s (4.0)
(72@533 MT/s)

10.8 GB/s
(7.5 duplex, 4.2 simplex)
(2x20@3.0 GT/s diff)

12 slot I/O Backplane
2 @ 266 MHz
6 @ 133 MHz
4 @ 86 MHz

I/O System Bus Adapter

Host Adapter 0

PCI-X

Host Adapter 11

PCI-X

Cell 1

Cell 2

Cell 3

Cell 4

Cell 5

Cell 6

Cell 7

Cell 8

Cell 9

Cell 10

Cell 11

Cell 12

Cell 13

Cell 14

Cell 15

Total Peak (Sust) BW per Cell
17.0 GB/s (13.6) CPUs (1.3x)
17.0 GB/s (16.0) Memory (2.1x)
34.6 GB/s (27.3) Crossbar (4.2x)
10.8 GB/s (7.5) I/O (4.1x)

System Aggregate:
>1 TFlop/s w/ Monte Carlo
106.2 GB/s bisection BW
120 GB/s I/O BW: >10,000 TB/day
GPU Antonio Banderas: GeForce 9800 GX2 (Nvidia - 2008)

- Dual GPU GeForce 8800 (G80).
- Dual printed circuit board.
- Dual video memory 512 MB and 256 bits.
- A single PCI-express connector.
We use a single GPU, but there are up to six available on three PCI-express slots. This guarantees a tremendous potential for growing in the near future.
Q-norm: A prototype for parallel implementations

<table>
<thead>
<tr>
<th>Step: Input</th>
<th>A: order &amp; local average</th>
<th>Global averaging</th>
<th>CD: Store normalize</th>
</tr>
</thead>
</table>
OpenMP version for Picasso
(shared memory multiprocessor)

nE = LoadProject(fname, fList); // Step 1: I/O

for (i=0;i< nE;i++) { // Step 2: Sort columns
    LoadFile(fList, i, dataIn);
    Qnorm1(dataIn, dIndex, fList[i].nG);
    PartialRowAccum(AvG, dataIn , nG);
    // Manage index in memory or disc

for (i=0;i<nG;i++) // Step 3: Compute averages
    AvG[i].Av /=AvG[i].num;

    // Produce output file with vector data sorted for each column
    for (i=0;i<nE;i++)
    {
        Obtain column index from disc
        for (j=0;j<nG;j++) { // Prepare output vector
            dataOut[dIndex[j]]=AvG[j].Av;
            // File position to write output vector
        }
    }

#pragma omp parallel shared
From, To, Range
// Open general parallel section

WE ONLY HAVE TO ADD THESE TWO DIRECTIVES TO THE ORIGINAL CODE

#pragma omp parallel shared
From, To, Range
Message-passing version for Pablo (distributed memory multiprocessor)

Master
Slave(s)

Get Parameters, Initialize
CalculateBlocks(nP,IniBlocks)
Broadcast(IniBlocks)

Start with params
Receive (Block)
while(!ENDsignal) {
  for each experiment in block {
    LoadExperiment(Exp);
    SortExperiment(Exp);
    AcumulateAverage(Exp);
  }
}

while (ThereIsBlocks) {
  Receive(ResultBlock,who)
  AverageBlock(ResultBlock)
  if(!SendAllBlocks) {
    CalculateNextBlock(NextBlock)
    Send(who,NextBlock)
  }
}

Broadcast(ENDsignal)

Receive(Block);

AverageBlock(ResultBlock);
Send(ResultBlock);

ReportResults
CUDA version for Antonio Banderas

```c
__global__ void QSortGPU(int *dataIn, int *dIndex) // The heavy kernel on GPU
{
    // Index to the element to be computed by each thread
    int idx = (blockIdx.x * blockDim.x) + threadIdx.x;
    // Sort the dataIn array and return dIndex as output permutation
}

__host__ void main() // The host program running on CPU
{
    // N is the number of input samples; p is the number of gene expression values
    // Bi (threads/block) and NBi (number of blocks) determine parallelism
    int N = LoadProject(fname,fList); // p is read from file similarly
    for (i=0; i<N; i++) {
        LoadFromFile(fList,i,dataIn); CPUtoGPU(dataIn); // Move data from file to GPU
        B1 = 512; // NB1 depends on p and B1
        QSortGPU <<<NB1,B1>>>(p,*dataIn,*dIndex); // CUDA kernel invocation
        GPUMtoCPU(p,dIndex); WriteToFile(p,dIndex); // Move data from GPU back to file
        RowAccum <<<NB2,B2>>>(p,*dataIn,*dataAvg); // 2nd CUDA kernel
    }
    GlobalAvg <<<(NB3,B3)>>>(*dataAvg,N); // 3rd CUDA kernel
    GPUMtoCPU(*dataAvg);
    // For all samples, read *dIndex file and update *dataIn file according to *dataAvg
}
```
Execution times for Q-norm (secs.)

Pablo (a fraction of former Mare Nostrum).
Including I/O time

Picasso (HP Superdome).
Including I/O time

Antonio Banderas
(PC with a GeForce 9800 GX2)

<table>
<thead>
<tr>
<th>PC</th>
<th>CPU only</th>
<th>incl. GPU</th>
<th>Acel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing</td>
<td>807</td>
<td>115</td>
<td>7.03x</td>
</tr>
<tr>
<td>I/O</td>
<td>179</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>Commun.</td>
<td>0</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>986</td>
<td>319</td>
<td>3.09x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># CPUs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pablo &amp; speed-up</td>
<td>1708</td>
<td>867</td>
<td>469</td>
<td>339</td>
<td>285</td>
<td>221</td>
<td>220</td>
<td>175</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speed-up</td>
<td>1.97x</td>
<td>3.64x</td>
<td>5.03x</td>
<td>5.99x</td>
<td>7.71x</td>
<td>7.76x</td>
<td>9.76x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Picasso &amp; speed-up</td>
<td>210</td>
<td>108</td>
<td>72</td>
<td>54</td>
<td>43</td>
<td>37</td>
<td>31</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speed-up</td>
<td>1.94x</td>
<td>2.91x</td>
<td>3.67x</td>
<td>4.82x</td>
<td>5.64x</td>
<td>6.66x</td>
<td>7.32x</td>
<td>10.9x</td>
<td>13.6x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GPU parallelization break-up

The CUDA block size is crucial, but GPU does not shine because Q-norm lacks of arithmetic intensity

<table>
<thead>
<tr>
<th>Threads per block</th>
<th>GPU time</th>
<th>Improvement</th>
<th>Overall improv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>204,38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>170,24</td>
<td>17%</td>
<td>17%</td>
</tr>
<tr>
<td>128</td>
<td>141,21</td>
<td>18%</td>
<td>31%</td>
</tr>
<tr>
<td>256</td>
<td>122,13</td>
<td>13%</td>
<td>40%</td>
</tr>
<tr>
<td>512</td>
<td>114,92</td>
<td>6%</td>
<td>44%</td>
</tr>
</tbody>
</table>

Q-norm is I/O bounded

<table>
<thead>
<tr>
<th>Transfers</th>
<th>Secs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. From disk to CPU (SAS, 2 RAID 0)</td>
<td>162,32</td>
</tr>
<tr>
<td>2. From CPU to GPU (PCI-express 2)</td>
<td>13,89</td>
</tr>
<tr>
<td>3. From GPU to CPU (PCI-express 2)</td>
<td>11,51</td>
</tr>
<tr>
<td>4. From CPU to disc (asynchronous)</td>
<td>16,70</td>
</tr>
<tr>
<td>Total I/O time (1+4)</td>
<td>179,02</td>
</tr>
<tr>
<td>Total CPU-GPU comm. time (2+3)</td>
<td>25,40</td>
</tr>
<tr>
<td>Total transfer time (1+2+3+4)</td>
<td>204,42</td>
</tr>
</tbody>
</table>
To download codes

- All three versions analyzed, as well as related documents can be downloaded from our Web site located at:
  - [https://chirimoyo.ac.uma.es/qnorm](https://chirimoyo.ac.uma.es/qnorm)
V. TESLA
A new profile for Nvidia users

GeForce® Gamers and leisure
Quadro® Professional Design
Tesla High Performance Computing

All based on the same microarchitecture
A Tesla card versus a GeForce and a Quadro

- **Similarities:**
  - Connected to a PCI-e socket on motherboard.
  - Accessible under Windows/Linux/MacOS.

- **Differences:**
  - Cost around 1000 €.
  - It does not allow you to play games (GeForce), nor rendering (Quadro), as it lacks of a DVI graphics output.
What is TESLA?

- A high-end GPU oriented to general-purpose computing.
- The GPU architecture is shared with GeForce models, but video memory is ahead (analogy with Intel Xeon versus commodity models).
- It has several predetermined configurations:
  - Low-end user: PCI-express card on a simple PC motherboard.
  - Mid-end user: Server grouping four cards on a 1U-rack basis.
  - Graphics supercomputer with multiple servers interconnected.
- Each server consumes around 800 W. (and noisy: 70 dB.).
- It requires a host (CPU) which supplies computational kernels and the file system.
How Tesla contributes to high-performance computing

- To small research labs, graphics cards for general-purpose computing at an affordable cost.
- To big companies and research centers, servers grouping 2x2 cards on a 1U rack for an initial cost around 9000€.
There is a hardware platform for each end user

- Millions of researchers: Tesla graphics card (Less than 5000 dollars)
- Thousand of researchers: Cluster of Tesla servers (Between 50,000 and 1,000,000 dollars)
- Millions of researchers: Large-scale clusters (More than a million $)
Performance/cost ratio is very attractive

Cluster of Tesla coprocessing nodes

Tesla personal supercomputer

A traditional CPU server

Performance

Cost

1x

< 10000 €

10000 - 1M €

100x

10000x

100x cost

100x performance

100x cost

100x performance
... and power consumption is more efficient

18X better at performance/watt ratio versus a CPU
Compared to a typical CPU high-end server

Building a 100TF datacenter

4 CPU cores
0.07 Teraflop
$2000
400 W
1429 CPU servers

$3.1 M
571 KW

4 GPUs: 960 cores
4 Teraflops
$8000
800 W
25 CPU servers
25 Tesla systems

$0.31 M
27 KW

10x lower cost
21x lower power
TESLA server connection to a supercomputing rack
Tesla products and roadmap for 2010

**Large Data Sets**
- **Tesla C2070**
  - 520-630 Gflop DP
  - 6GB Memory, ECC
  - $3,999
- **Tesla S2070**
  - 2.1-2.5 Tflop DP
  - 6 GB / GPU, ECC
  - $18,995
- **Tesla M-Fermi OEM Server Module**
  - 6GB

**8x Peak Double Precision Performance**
- **Tesla C2050**
  - 520-630 Gflop DP
  - 3 GB Memory, ECC
  - $2,499
- **Tesla S2050**
  - 2.1-2.5 Tflop DP
  - 3 GB / GPU, ECC
  - $12,995
- **Tesla M-Fermi OEM Server Module**
  - 3GB

**Best Single Precision Performance / $**
- **Tesla C1060**
  - 933 Gflop SP
  - 4 GB Memory
  - $1,299
- **Tesla S1070**
  - 4.14 Tflop SP
  - 345 Gflop DP
  - 4 GB / GPU
  - $7,995
- **Tesla M1060 Module**
  - 933 Gflop SP
  - 78 Gflop DP
  - 4 GB Memory
Servers based on TESLA (2010)

- Commercial cluster: >3000 GPUs
- Government: >2000 GPUs
- Chinese Academy of Sciences - Industrial Process Institute: 828 GPUs
- Tokyo Institute of Technology Supercomputing Center: 680 GPUs
- NCSA – National Center for Supercomputing Applications: 384 GPUs
- Seismic processing: 256 GPUs
- Pacific Northwest National Labs – Biomedical research: 256 GPUs
- CSIRO – Australian National Supercomputing Center: 252 GPUs
- Riken – Japanese Astrophysical research: 220 GPUs
- Seismic processing: 200 GPUs
- Chinese Academy of Sciences – Institute of Modern Physics: 200 GPUs

Source: Nvidia
Acknowledgements

From Biomedical Informatics Dept., Ohio State University (US):
- Tim Hartley, M.Sc., Umit Catalyurek, PhD.
- Lee Cooper, M.Sc., Kun Huang, PhD.
- Olcay Sertel, M.Sc., Metin Gurcan, PhD.

From the Ohio Supercomputer Center (US):
- Dennis Sessanna, Don Stredney, PhD.

From the Computer Engineering Dept., Jaume I Univ. (Spain):
- Francisco Igual, M.Sc.

From my Department in Malaga:
- Antonio Ruiz, M.Sc.
- Andrés Rodríguez, PhD.
Thanks!

- My coordinates in Spain:
  - E-mail: ujaldon@uma.es
  - Web page: http://www.ac.uma.es/~ujaldon