Discussion on

NVIDIA's Compute Unified Device Architecture (CUDA)

Stan Tomov
04/13/2011

(see also: http://www.nvidia.com/object/cuda_get.html)
Why talking about CUDA?

- Hardware Trends

<table>
<thead>
<tr>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREASE IN PARALLELISM</td>
</tr>
<tr>
<td>INCREASE IN COMMUNICATION COST (vs COMPUTATION)</td>
</tr>
</tbody>
</table>
Evolution of GPUs

- GPUs continue to improve due to ever increasing computational requirements, as better games are linked to
  - **Faster** and more **realistic graphics**
    (more accurate and complex physics simulations)

- To acquire ever
  - **More power** (1 TFlop/s in single, 140 GB/s memory bandwidth)
  - **More functionality** (support fully IEEE double precision, multithreading, pointers, asynchronicity, levels of memory hierarchy, etc.)
  - **More programmability** (with CUDA no need to know graphics to program for GPUs; can use CUDA libraries to benefit from GPUs without knowing CUDA)

- Towards **hybrid architectures**, integrating (in varying proportions) two major components
  - **Multicore** CPU technology
  - Special purpose hardware and accelerators, especially **GPUs**

  as evident from major chip manufacturers, such as Intel, AMD, IBM, and NVIDIA
## Current NVIDIA GPUs

<table>
<thead>
<tr>
<th></th>
<th>GeForce GTX 280</th>
<th>GeForce GTX 260</th>
<th>Tesla C1060</th>
<th>Tesla S1070</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Form Factor</strong></td>
<td>Dual slot card</td>
<td>Dual slot card</td>
<td>Dual slot card</td>
<td>Rackmount</td>
</tr>
<tr>
<td><strong>TPCs</strong></td>
<td>10</td>
<td>8</td>
<td>10</td>
<td>4x10</td>
</tr>
<tr>
<td><strong>SMs</strong></td>
<td>30</td>
<td>24</td>
<td>30</td>
<td>4x30</td>
</tr>
<tr>
<td><strong>SPs</strong></td>
<td>240</td>
<td>192</td>
<td>240</td>
<td>4x240</td>
</tr>
<tr>
<td><strong>Graphics Freq.</strong></td>
<td>602MHz</td>
<td>576MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Processor Freq.</strong></td>
<td>1296MHz</td>
<td>1242MHz</td>
<td>1300MHz</td>
<td>1500MHz</td>
</tr>
<tr>
<td><strong>Memory Freq.</strong></td>
<td>1107MHz</td>
<td>999MHz</td>
<td>800MHz</td>
<td>800MHz</td>
</tr>
<tr>
<td><strong>Memory Bandwidth</strong></td>
<td><strong>141.7 GB/s</strong></td>
<td>127.9 GB/s</td>
<td>102.4 GB/s</td>
<td>4x102.4 GB/s</td>
</tr>
<tr>
<td><strong>Memory Capacity</strong></td>
<td>1GB</td>
<td>896MB</td>
<td>4GB</td>
<td>4x4GB</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>236W TDP</td>
<td>183W TDP</td>
<td>160W &quot;Typical&quot;</td>
<td>700W &quot;Typical&quot;</td>
</tr>
<tr>
<td><strong>SP GFLOP/s (wo/MUL)</strong></td>
<td>622.1</td>
<td>476.9</td>
<td>624.0</td>
<td>4x720.0</td>
</tr>
<tr>
<td><strong>SP GFLOP/s (w/MUL)</strong></td>
<td><strong>933.1</strong></td>
<td>715.4</td>
<td>936.0</td>
<td>4x1080.0</td>
</tr>
<tr>
<td><strong>DP GFLOP/s</strong></td>
<td><strong>77.8</strong></td>
<td>59.6</td>
<td>78.0</td>
<td>4x72.0</td>
</tr>
</tbody>
</table>

(Source: “NVIDIA's GT200: Inside a Parallel Processor”)
How to learn CUDA?

1) Get the hardware and install the latest NVIDIA drivers, CUDA Toolkit, and CUDA SDK

2) Compile, run, and study some of the projects of interest that come with the NVIDIA SDK

3) Do Homework 9, Part II
   **Note:** This exercise is on **Hybrid computing**, something that we encourage. It shows you don't need to know CUDA in order to benefit GPUs – just to design your algorithms on high level, splitting the computation between CPU and GPU, and using CUDA kernels for the GPU part.

4) Develop user specific CUDA kernels (whenever needed)
   [ read the CUDA Programming guide & study projects of interest]
GPUs for HPC

- Programmability
- Performance
- Hybrid computing
Programmability

CUDA Software Stack

CUBLAS, CUFFT, MAGMA, ...

C like API
How to program in parallel?

- There are many parallel programming paradigms, e.g.,

```
master/worker  divide and conquer  pipeline  work pool  data parallel (SPMD)
```

- In reality applications usually combine different paradigms

- CUDA and OpenCL have roots in the data-parallel approach (now adding support for task parallelism)

Programming model

A highly multithreaded coprocessor
  * thread block
    ( a batch of threads with fast shared memory executes a kernel )
  * Grid of thread blocks
    ( blocks of the same dimension, grouped together to execute the same kernel; reduces thread cooperation )

__global__ void MatVec(...) {
  // Block index
  int bx = blockIdx.x;
  int by = blockIdx.y;

  // Thread index
  int tx = threadIdx.x;
  int ty = threadIdx.y;
  ...
}

// set the grid and thread configuration
Dim3 dimBlock(3,5);
Dim3 dimGrid(2,3);

// Launch the device computation
MatVec<<<dimGrid, dimBlock>>>(...);
Performance

High performance derives from

- High parallelism
  [ 240 processing elements ]

- Shared memory + registers (16K / Block)
  [ allows for memory reuse ]

- High bandwidth to memory
  [ 141.7 GB/s ]

- CPU- GPU Interface: PCI Express x 16
  [ up to 4 GB/s peak per direction
    up to 8 GB/s concurrent bandwidth ]

Memory model [ numbers for GTX 280 ]
An Example of Memory Reuse
(through use of shared memory)

[ see sgemm example file from lecture 10 ]
Hybrid Computing

Excelling in Graphics Rendering

An operation that

- Requires enormous computational power
- Allows for high parallelism
- Stresses more on high bandwidth vs low latency (because of a deep graphics pipeline)

Powerful General Purpose GPU

Computational pattern, common with many applications (but still not all applications map well)

Hybrid GPU + CPU Computing

Split the computation to fully exploit the power that each of the hybrid components offers (GPUs + multicores)
An approach for multicore+GPUs

- Split algorithms into **tasks** and **dependencies** between them, e.g., represented as DAGs.
- Schedule the execution in parallel without violating data dependencies.

Algorithms as DAGs (small tasks/tiles for homogeneous **multicore**)

Hybrid CPU+GPU algorithms (small tasks for multicores and large tasks for GPUs)

e.g., in the PLASMA library for Dense Linear Algebra
http://icl.cs.utk.edu/plasma/
e.g., in the MAGMA library for Dense Linear Algebra
http://icl.cs.utk.edu/magma/
Discussion

• Dense Linear Algebra
  − Matrix-matrix product
  − LAPACK with CUDA
• Sparse Linear Algebra
  − Sparse matrix-vector product
• Projects using CUDA
Conclusions

- **GPU computing**
  Significantly outperform current multicores on many real world applications (illustrated for DLA which has been traditionally of HP on x86 architectures)
  - New algorithms needed (increased parallelism and reduced communication)
  - Speed vs accuracy trade-offs
  - Autotuning

- **Hybrid GPU+CPU computing**
  There are still applications – or at least part of them – that do not map well on GPU architectures and would benefit much more a hybrid one

- **Architecture trends**
  Towards heterogeneous/hybrid designs, integrating (in varying proportions) two major components
  - Multicore CPU technology
  - Special purpose hardware and accelerators, especially GPUs