HPC with Multicore and GPUs

Stan Tomov
Electrical Engineering and Computer Science Department
University of Tennessee, Knoxville

CS 594 Lecture Notes
March 13, 2013
Outline

- **Introduction**
  - Hardware trends
- **Challenges of using multicore+GPUs**
- **How to code for GPUs and multicore**
  - An approach that we will study
- **Introduction to CUDA**
- **Conclusions**
Speeding up Computer Simulations

Better numerical methods

Exploit advances
in hardware

http://www.cs.utk.edu/~tomov/cflow/

e.g. a posteriori error analysis:
solving for much less DOF but achieving the same accuracy

Performance Development in Top500

- Manage to use hardware efficiently for real-world HPC applications
- Match LU benchmark in performance!
Why multicore and GPUs?

Hardware trends

- Multicore

- GPU Accelerators

Power is the root cause of all this

A hardware issue just became a software problem

(Source: slide from Kathy Yelick)

<table>
<thead>
<tr>
<th>Form Factor</th>
<th>GeForce GTX 280</th>
<th>GeForce GTX 260</th>
<th>Tesla C1060</th>
<th>Tesla S1070</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCs</td>
<td>10</td>
<td>8</td>
<td>10</td>
<td>4x10</td>
</tr>
<tr>
<td>SMs</td>
<td>30</td>
<td>24</td>
<td>30</td>
<td>4x30</td>
</tr>
<tr>
<td>SPs</td>
<td>240</td>
<td>192</td>
<td>240</td>
<td>4x240</td>
</tr>
<tr>
<td>Graphics Freq.</td>
<td>602MHz</td>
<td>576MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor Freq.</td>
<td>1296MHz</td>
<td>1242MHz</td>
<td>1300MHz</td>
<td>1600MHz</td>
</tr>
<tr>
<td>Memory Freq.</td>
<td>1107MHz</td>
<td>999MHz</td>
<td>800MHz</td>
<td>800MHz</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>141.7GB/s</td>
<td>127.9GB/s</td>
<td>102.4GB/s</td>
<td>4x102.4GB/s</td>
</tr>
<tr>
<td>Memory Capacity</td>
<td>1GB</td>
<td>8GB</td>
<td>4GB</td>
<td>4GB</td>
</tr>
<tr>
<td>Power</td>
<td>236W TDP</td>
<td>183W TDP</td>
<td>160W &quot;Typical&quot;</td>
<td>700W &quot;Typical&quot;</td>
</tr>
<tr>
<td>SP GFLOP/s (wo/MUL)</td>
<td>622.1</td>
<td>476.9</td>
<td>624.0</td>
<td>4x420.0</td>
</tr>
<tr>
<td>SP GFLOP/s (w/MUL)</td>
<td>933.1</td>
<td>715.4</td>
<td>936.0</td>
<td>4x1080.0</td>
</tr>
<tr>
<td>DP GFLOP/s</td>
<td>77.8</td>
<td>55.6</td>
<td>78.0</td>
<td>4x72.0</td>
</tr>
</tbody>
</table>

(Source: “NVIDIA’s GT200: Inside a Parallel Processor”)
Main Issues

- **Increase in parallelism** *1
  How to code (programming model, language, productivity, etc.)?

- **Increase in commun. cost (vs computation)** *2
  How to redesign algorithms?

- **Hybrid Computing** *3
  How to split and schedule the computation between hybrid hardware components?

Despite issues, **high speedups** on HPC applications are reported using GPUs (from NVIDIA CUDA Zone homepage).

**CUDA architecture & programming:** *1
- A data-parallel approach that scales
- Similar amount of efforts on using CPUs vs GPUs by domain scientists demonstrate the GPUs' potential

**Processor speed** improves 59% / year but memory bandwidth by 23% latency by 5.5% *2

**e.g., schedule small non-parallelizable tasks** on the CPU, and large and parallelizable on the GPU *3
Evolution of GPUs

**GPUs:** excelling in graphics rendering

- **Scene model** → Graphics pipelined computation → Final image
- Repeated fast over and over: e.g. TV refresh rate is 30 fps; limit is 60 fps

This type of computation:
- Requires **enormous computational power**
- Allows for **high parallelism**
- Needs **high bandwidth vs low latency**
  (as low latencies can be compensated with deep graphics pipeline)

Obviously, this pattern of computation is common with many other applications
Challenges of using multicore+GPUs

• **Massive parallelism**
  Many GPU cores, serial kernel execution
  [ e.g. 240 in the GTX280; up to 512 in *Fermi* – to have concurrent kernel execution ]

• **Hybrid/heterogeneous architectures**
  Match algorithmic requirements to architectural strengths
  [ e.g. small, non-parallelizable tasks to run on CPU, large and parallelizable on GPU ]

• **Compute vs communication gap**
  Exponentially growing gap; persistent challenge
  [ Processor speed improves 59%, memory bandwidth 23%, latency 5.5% ]
  [ on all levels, e.g. a GPU Tesla C1070 (4 x C1060) has compute power of O(1,000) Gflop/s but GPUs communicate through the CPU using O(1) GB/s connection ]
How to Code for GPUs?

- Complex question
  - Language, programming model, user productivity, etc

- Recommendations
  - **Use CUDA / OpenCL**
    - [already demonstrated benefits in many areas; data-based parallelism; move to support task-based]
  
  - **Use GPU BLAS**
    - [high level; available after introduction of shared memory – can do data reuse; leverage existing developments ]

  - **Use Hybrid Algorithms**
    - [currently GPUs – massive parallelism but serial kernel execution; hybrid approach – small non-parallelizable tasks on the CPU, large parallelizable tasks on the GPU ]

*Typical order of acceleration:*

dense matrix-matrix $O(1) \times$
dense matrix-vector $O(10) \times$
sparse matrix-vector $O(100) \times$
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/
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.

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

E.g., in the MAGMA library for Dense Linear Algebra
http://icl.cs.utk.edu/magma/
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)

Compute Unified Device Architecture (CUDA) Software Stack

(CPU) → Application → CUDA Libraries → CUDA Runtime → CUDA Driver → (GPU)

CUBLAS, CUFFT, MAGMA, ...

C like API

(Source: NVIDIA CUDA Programming Guide)
CUDA Memory Model

(Grid)

Block (0, 0)
- Shared Memory
- Registers
- Thread (0, 0)
- Local Memory

Block (1, 0)
- Shared Memory
- Registers
- Thread (0, 0)
- Local Memory

(Source: NVIDIA CUDA Programming Guide)
CUDA Programming Model

- **Grid of thread blocks**
  (blocks of the same dimension, grouped together to execute the same kernel)

- **Thread block**
  (a batch of threads with fast shared memory executes a kernel)

- **Sequential code launches asynchronously GPU kernels**

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

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

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

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

    ...}
```

(Source: NVIDIA CUDA Programming Guide)
Conclusions

- Hybrid Multicore+GPU computing:
  - Architecture trends: towards heterogeneous/hybrid designs
  - Can significantly accelerate linear algebra [vs just multicores];
  - Can significantly accelerate algorithms that are slow on homogeneous architectures