HPC with Multicore and GPUs

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

COSC 594 Lecture Notes
March 21, 2018
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 and the cs954 project/library

- Conclusions
Speeding up Computer Simulations

Better numerical methods

Exploit advances in hardware

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

Hardware trends

- **Multicore**

- **GPU Accelerators**
  old GPUs; compare 77 Gflop/s with today’s P100, reaching 4,700 Gflop/s in DP

---

**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></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>1600MHz</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>141.7GB/s</td>
<td>127.9GB/s</td>
<td>102.4GB/s</td>
<td>4x102.4GB/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>933.1</td>
<td>715.4</td>
<td>936.0</td>
<td>4x1080.0</td>
</tr>
<tr>
<td><strong>DP GFLOP/s</strong></td>
<td>77.8</td>
<td>59.6</td>
<td>76.0</td>
<td>4x72.0</td>
</tr>
</tbody>
</table>

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

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

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

- **Hybrid Computing**
  - 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)

<table>
<thead>
<tr>
<th>CUDA architecture &amp; programming:</th>
<th>Processor speed improves 59% / year but memory bandwidth by 23% latency by 5.5%</th>
<th>e.g., schedule small non-parallelizable tasks on the CPU, and large and parallelizable on the GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>- A data-parallel approach that scales</td>
<td>- Similar amount of efforts on using CPUs vs GPUs by domain scientists demonstrate the GPUs' potential</td>
<td></td>
</tr>
</tbody>
</table>
Evolution of GPUs

**GPUs:** excelling in graphics rendering

- Scene model \(\xrightarrow{\text{streams of data}}\) Graphics pipelined computation \(\xrightarrow{}\) Final image

Repeated fast over and over: e.g. TV refresh rate is 30 fps; limit is 60 fps

- Currently, can be viewed as **multithreaded multicore vector units**

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
This is the programming model of MAGMA

**TASK-BASED ALGORITHMS**

MAGMA uses task-based algorithms where the computation is split into tasks of varying granularity and their execution scheduled over the hardware components. Scheduling can be static or dynamic. In either case, small non-parallelizable tasks, often on the critical path, are scheduled on the CPU, and larger more parallelizable ones, often Level 3 BLAS, are scheduled on the GPUs.

**PERFORMANCE & ENERGY EFFICIENCY**

MAGMA LU factorization in double precision arithmetic

![Performance Chart]

- **CPU**
  - Intel Xeon E5-2650 v3 (Haswell)
  - 2x10 cores @ 2.30 GHz

- **K40**
  - NVIDIA K40 GPU
  - 15 MP x 192 @ 0.88 GHz

- **P100**
  - NVIDIA Pascal GPU
  - 56 MP x 64 @ 1.19 GHz

![Energy Efficiency Chart]
MAGMA main thrusts and use

MAGMA – research vehicle on LA algorithms and libraries for new architectures

for architectures in
   { CPUs + Nvidia GPUs (CUDA),
     CPUs + AMD GPUs (OpenCL),
     CPUs + Intel Xeon Phis,
     manycore (native: GPU or KNL/CPU),
     embedded systems, combinations, and
     software stack, e.g., since CUDA x }

for precisions in
   { s, d, c, z,
     adding half real and complex,
     mixed, … }

for interfaces
   { heterogeneous CPU or GPU,
     native, … }

• LAPACK
• BLAS
• Batched LAPACK
• Batched BLAS
• Sparse

• MAGMA for CUDA
  GPU Center of Excellence (GCOE) for 8th year

• MAGMA for Xeon Phi
  Intel Parallel Computing Center (IPCC)
  5th year collaboration with Intel on Xeon Phi

• MAGMA in OpenCL
  Collaboration with AMD

  Number of downloads for 2016 so far is 6,205
  (around 3K per release; to be ~10K for 2016)

  MAGMA Forum: 2,727 + 312 (3,039) posts in
  742 + 75 (817) topics, 1,038 + 317 (1,355) users

  MAGMA is incorporated in MATLAB (as of the R2010b),
  contributions in CUBLAS and MKL,
  AMD, Siemens (in NX Nastran 9.1), ArrayFire,
  ABINIT, Quantum-Espresso, R (in HiPLAR & CRAN),
  SIMULIA (Abaqus), MSC Software (Nastran and Marc),
  Cray (in LibSci for accelerators libsci_acc),
  Nano-TCAD (Gordon Bell finalist),
  Numerical Template Toolbox (Numscale), and others.
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**)

E.g., in the **PLASMA** library for Dense Linear Algebra
http://icl.cs.utk.edu/plasma/
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**)

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

*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/
Design algorithms to use mostly BLAS 3

Level 1, 2 and 3 BLAS

Nvidia P100, 1.19 GHz, Peak DP = 4700 Gflop/s

- **dgemm** BLAS Level 3
- **dgemv** BLAS Level 2
- **daxpy** BLAS Level 1

\[ C = C + A \times B \]
\[ y = y + A \times x \]
\[ y = \alpha \times x + y \]

Graph showing performance for different matrix sizes and vector sizes.

- 4503 Gflop/s
- 52 Gflop/s
- 31x speedup
How to program in parallel?

There are many parallel programming paradigms (to be covered w/ Prof. George Bosilca), 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

(Source: NVIDIA CUDA Programming Guide)
CUDA Hardware Model

(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 Program Sequential Execution

// 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;

  . . .
}
Jetson TK1

- A full-featured platform for embedded applications
- It allows you to unleash the power of 192 CUDA cores to develop solutions in computer vision, robotics, medicine, security, and automotive;
- You have accounts on astro.icl.utk.edu (for Ozgur Cekmer and Yasser Gandomi) rudi.icl.utk.edu (for Yuping Lu and Eduardo Ponce)
- Board features
  - Tegra K1 SOC
    - Kepler GPU with 192 CUDA cores
    - 4-Plus-1 quad-core ARM Cortex A15 CPU
  - 2 GB x16 memory with 64 bit width
  - 16 GB 4.51 eMMC memory
  - ...

https://developer.nvidia.com/jetson-tk1
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