HPC with Multicore and GPUs

Stanimire Tomov and Jack Dongarra
Innovative Computing Laboratory
University of Tennessee, Knoxville

TAMU visit
December 3-4 College Station, TX
Outline

• Introduction
  – Hardware to Software Trends

• LAPACK to MAGMA / PLASMA Libraries
  – Challenges and approach
  – One-sided and two-sided factorizations

• Performance Results

• 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!

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

http://www.cs.utk.edu/~tomov/cflow/
Moore’s Law is Alive and Well

Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanović
Slide from Kathy Yelick
But Clock Frequency Scaling Replaced by Scaling Cores / Chip

15 Years of exponential growth ~2x year has ended

Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanović
Slide from Kathy Yelick
Performance Has Also Slowed, Along with Power

Power is the root cause of all this

A hardware issue just became a software problem

Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanović
Slide from Kathy Yelick
A multicore chip is a single chip (socket) that combines two or more independent processing units that provide independent threads of control.

- **Power ∝ Voltage^2 x Frequency** (V^2F)
- **Frequency ∝ Voltage**
- **Power ∝ Frequency^3**

<table>
<thead>
<tr>
<th></th>
<th>Cores</th>
<th>V</th>
<th>Freq</th>
<th>Perf</th>
<th>Power</th>
<th>PE (Bops/watt)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Superscalar</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>&quot;New&quot; Superscalar</strong></td>
<td>1X</td>
<td>1.5X</td>
<td>1.5X</td>
<td>1.5X</td>
<td>3.3X</td>
<td>0.45X</td>
</tr>
<tr>
<td><strong>Multicore</strong></td>
<td>2X</td>
<td>0.75X</td>
<td>0.75X</td>
<td>1.5X</td>
<td>0.8X</td>
<td>1.88X</td>
</tr>
</tbody>
</table>

(Bigger # is better)

50% more performance with 20% less power

Preferable to use multiple slower devices, than one superfast device
Why GPU-based Computing?

- Hardware Trends

**Hardware**

INCREASE IN PARALLELISM

INCREASE IN COMMUNICATION COST (vs COMPUTATION)

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

Hybrid / Heterogeneous Designs

Multicore + GPUs
Evolution of GPUs

GPUs: excelling in graphics rendering

Scene model → Streams of data → 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
# 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>Form Factor</td>
<td>Dual slot card</td>
<td>Dual slot card</td>
<td>Dual slot card</td>
<td>Rackmount</td>
</tr>
<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><strong>240</strong></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>1500MHz</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><strong>141.7GB/s</strong></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>896MB</td>
<td>4GB</td>
<td>4x4GB</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>4x720.0</td>
</tr>
<tr>
<td>SP GFLOP/s (w/MUL)</td>
<td><strong>933.1</strong></td>
<td>715.4</td>
<td>936.0</td>
<td>4x1080.0</td>
</tr>
<tr>
<td>DP GFLOP/s</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”)
HPC applications and **high speedups** reported using GPUs
(from NVIDIA CUDA Zone homepage)

CUDA architecture & programming:
- A data-parallel approach that scales
- Similar amount of efforts on using CPUs vs GPUs by domain scientists demonstrate the GPUs' potential
Matrix Algebra on GPU and Multicore Architectures (MAGMA)

- **MAGMA**: a new generation linear algebra (LA) libraries to achieve the fastest possible time to an accurate solution on hybrid/heterogeneous architectures, starting with current multicore+MultiGPU systems. 
  
  *Homepage*: http://icl.cs.utk.edu/magma/

- **MAGMA & LAPACK**
  - **MAGMA** - based on LAPACK and extended for hybrid systems (multi-GPUs + multicore systems);
  - **MAGMA** - designed to be similar to LAPACK in functionality, data storage and interface, in order to allow scientists to effortlessly port any of their LAPACK-relying software components to take advantage of the new architectures
  - **MAGMA** - to leverage years of experience in developing open source LA software packages and systems like LAPACK, ScaLAPACK, BLAS, ATLAS as well as the newest LA developments (e.g. communication avoiding algorithms) and experiences on homogeneous multicores (e.g. PLASMA)

- **Support**
  - NSF, Microsoft, NVIDIA [ now CUDA Center of Excellence at UTK on the development of Linear Algebra Libraries for CUDA-based Hybrid Architectures ]

- **MAGMA developers**
  - University of Tennessee, *Knoxville*; University of California, *Berkeley*; University of Colorado, *Denver*
Challenges of using 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 ]
GPUs for HPC

- Programmability
- Performance
- Hybrid computing
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 ]
LAPACK to Multicore

A New Generation of Software:
Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

<table>
<thead>
<tr>
<th>Software/Algorithms follow hardware evolution in time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (70’s)</td>
</tr>
<tr>
<td>(Vector operations)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- Level-1 BLAS operations</td>
</tr>
<tr>
<td>LAPACK (80’s)</td>
</tr>
<tr>
<td>(Blocking, cache friendly)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- Level-3 BLAS operations</td>
</tr>
<tr>
<td>ScaLAPACK (90’s)</td>
</tr>
<tr>
<td>(Distributed Memory)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- PBLAS Mess Passing</td>
</tr>
<tr>
<td>PLASMA (00’s)</td>
</tr>
<tr>
<td>New Algorithms</td>
</tr>
<tr>
<td>(many-core friendly)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- a DAG/scheduler</td>
</tr>
<tr>
<td>- block data layout</td>
</tr>
<tr>
<td>- some extra kernels</td>
</tr>
</tbody>
</table>

Those new algorithms
- have a very low granularity, they scale very well (multicore, petascale computing, ... )
- removes a lots of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out-of-core)
- rely on fast kernels

Those new algorithms need new kernels and rely on efficient scheduling algorithms.

“delayed update” to organize successive Level 2 BLAS as a single Level 3 BLAS

Split BLAS into tasks and represent algorithms as DAGs; new algorithms where Level 2 BLAS panel factorizations are now Level 3 BLAS
LAPACK to multicore+GPU (MAGMA)

1) Development of **NEW ALGORITHMS** (parallelism, hybrid, optimal communication)

2) **HYBRIDIZATION** of LAPACK/new algorithms
   - Represent LAPACK/new algorithms as a collection of **TASKS** and **DEPENDENCIES** among them
   - Properly **SCHEDULE** the tasks' execution over the multicore and the GPU

3) Development of GPU BLAS **KERNELS**

4) **AUTO-TUNED** implementations

![Diagram of algorithms as DAGs](image1.png)

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

![Diagram of hybrid CPU+GPU algorithms](image2.png)

- Hybrid CPU+GPU algorithms
  - (small tasks for multicore and large tasks for GPUs)
One-Sided Dense Matrix Factorizations (LU, QR, and Cholesky)

- Panels (Level 2 BLAS) are factored on CPU using LAPACK
- Trailing matrix updates (Level 3 BLAS) are done on the GPU using “look-ahead” (to overlap CPUs work on the critical path with the GPUs large updates)

**Example:** Left-Looking Hybrid Cholesky factorization

```
MATLAB code | LAPACK code | Hybrid code
-------------|-------------|-------------
[2] B = chol(B, 'lower') | spotrf('L', B, ha(j, j), lda, info) | cublasGetMatrix(nb, 0, 4, dA[j, j], *lda, hwork, nb)
```
One-sided hybrid factorizations

QR factorization in single precision arithmetic, CPU interface

Performance of MAGMA vs MKL

MAGMA QR time breakdown

- MAGMA
- MKL 8 cores
- MKL 1 core

GPU : NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)
CPU : Intel Xeon dual socket quad-core (8 cores @2.33 GHz)

GPU BLAS : CUBLAS 2.2, sgemm peak: 375 GFlop/s
CPU BLAS : MKL 10.0, sgemm peak: 128 GFlop/s

[ for more performance data, see http://icl.cs.utk.edu/magma ]
Two-sided matrix factorizations

- Two-sided factorization, e.g. consider Hessenberg reduction
  \[ Q A Q' = H \]
  \( H \) – upper Hessenberg, \( Q \) – orthogonal similarity transformation

- Importance
  
  **One-sided factorizations**
  - bases for linear solvers
  
  **Two-sided factorizations**
  - bases for eigen-solvers

- Block algorithm
  
  \( Q \) – a product of n-1 elementary reflectors
  
  \[ Q = H_1 H_2 \ldots H_{n-1} \quad H_i = I - \tau_i v_i v'_i \]
  
  \[ H_1 \ldots H_{nb} = I - V T V' \] (WY transform; the bases for delayed update or block algorithm)

- Can we accelerate it?
  
  [similarly to the one-sided using hybrid GPU-based computing]
  
  [to see much higher acceleration due to a removed bottleneck]
Homogeneous multicore acceleration?

There have been difficulties in accelerating it on homogeneous multicore systems.

**Hessenberg factorization in double precision arithmetic, CPU interface**

**Performance of MAGMA vs MKL**

- **CPU**: Intel Xeon dual socket quad-core (8 cores @2.33 GHz)
- **CPU BLAS**: MKL 10.0, dgemm peak: 65 GFlop/s

- MKL 8 cores
- MKL 1 core

One core is enough to saturate the bus. Performance does not scale.
The Bottleneck

Reduction times in seconds for $N = 4,000$

<table>
<thead>
<tr>
<th># cores</th>
<th>1</th>
<th>8</th>
<th>1+GPU</th>
<th>8+GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3 BLAS</td>
<td>25 (30%)</td>
<td>4</td>
<td>3.5 (60%)</td>
<td>2.7</td>
</tr>
<tr>
<td>Level 2 BLAS</td>
<td>59 (70%)</td>
<td>59</td>
<td>2.3 (40%)</td>
<td>2.3</td>
</tr>
</tbody>
</table>

No improvement

Level 3 BLAS update
[80% flops; ~30% of the run time]

Level 2 BLAS update
[20% flops; ~70% of the run time]

$y_j = A_j v_j$
[Line 5 of Algorithm 2]
Hybrid computing acceleration?

- Intuitively, yes, as **matrix-vector product is fast on GPUs**
- How to organize a hybrid computation?

![DGEMV Performance Graph](image)

**GPU**: GeForce GTX 280  
(240 Cores @ 1.30 GHz)

**Bandwidth**:  
Theoretical: 141 GB/s

Achieved > **100 GB/s**
Task Splitting & Task Scheduling

Task scheduling:
- Multicore+GPU
- GPU
- Multicore

Critical path

1. Copy $dP_i$ to CPU
2. Copy $v_j$ to GPU
3. Copy $y_j$ to CPU
4. Copy $Y$ to CPU

Work

GPU
Performance

Hessenberg factorization in double precision arithmetic, CPU interface

Performance of MAGMA vs MKL

Acceleration is much higher than one-sided (O(10)x) due to removed bottleneck

GPU: NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)  GPU BLAS: CUBLAS 2.3, dgemm peak: 75 GFlop/s
CPU: Intel Xeon dual socket quad-core (8 cores @2.33 GHz)  CPU BLAS: MKL 10.0 , dgemm peak: 65 GFlop/s

[ for more performance data, see http://icl.cs.utk.edu/magma ]
The symmetric case

- The same approach, using a fast DSYMV on the GPU

**DSYMV Performance**

**GPU**: GeForce GTX 280
(240 Cores @ 1.30 GHz)

**Bandwidth**:
Theoretical: 141 GB/s

Approach can be applied similarly to other types of matrices as the fast Level 2 BLAS needed is available
Linear Solvers

Solving $Ax = b$ using LU factorization

Intel(R) Xeon(R) E541@2.34GHz / 8 Cores + GTX 280 @1.30GHz / 240 Cores

- **Direct solvers**
  - Factor and do triangular solves in the same, working precision

- **Mixed Precision Iterative Refinement**
  - Factor in single (i.e. the bulk of the computation in fast arithmetic) and use it as preconditioner in simple double precision iteration, e.g.
  $$x_{i+1} = x_i + (LU_{SP})^{-1} P (b - A x_i)$$

![Graph showing performance of different solvers](image)
Extension to Multicore and Multi GPUs

Cholesky factorization in SP
Strong Scalability

HOST: 4x AMD Opteron core @1.8GHz
GPUs: 4x C1060 (240 cores each @1.44GHz)

Leveraging PLASMA developments:
- 2 level nested parallelism
  - **coarse**: PLASMA tiled algorithm and static scheduling
  - **fine**: tasks/tiles are redefined for hybrid 1 core+GPU computing
- Defining a “Magnum tiles approach”

[ with Hatem Ltaief, Rajib Nath, Peng Du, and Jack Dongarra ]
Conclusions

- Hybrid Multicore+GPU computing:
  - Architecture trends: towards heterogeneous/hybrid designs
  - Can significantly accelerate DLA [vs just multicores];
    - for both one and two-sided factorizations
  - Can significantly accelerate algorithms that are slow on homogeneous architectures