Dense Linear Algebra Solvers for Multicore with GPU Accelerators

Stanimire Tomov, Rajib Nath, Hatem Ltaief, and Jack Dongarra
Innovative Computing Laboratory
University of Tennessee, Knoxville

IEEE IPDPS 2010
High-level Parallel Programming Models and Supportive Environments (HIPS)
April 19-23, 2010, Atlanta, GA
Outline

• Introduction
  – Hardware to Software Trends

• The MAGMA library
  – Challenges and approach
  – One-sided factorizations and solvers
  – Two-sided factorizations

• 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!

![Graph showing error tolerance vs degrees of freedom](http://www.cs.utk.edu/~tomov/cflow/)

- Example of *a posteriori* error analysis: solving for much less DOF but achieving the same accuracy

![Performance Development in Top500 chart](http://www.cs.utk.edu/~tomov/cflow/)
Clock Frequency Scaling
Replaced by Scaling Cores/Chip

Data from Kunle Olukotun, Lance Hammon, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanović
Slide from Kathy Yelick
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**
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/](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
MAGMA 0.2

- LU, QR, Cholesky (S, C, D, Z)
- Linear solvers
  - In working precision, based on LU, QR, and Cholesky
  - Mixed-precision iterative refinement
- CPU and GPU interfaces
- Two-sided factorizations
  - Reduction to upper Hessenberg form
    (bi/tri-diagonalization developed)
- MAGMA BLAS
  - Routines critical for MAGMA (GEMM, SYRK, TRSM, GEMV, SYMV, etc.)
Challenges

- **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
  [ 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 ]

---

**GPU vs CPU GEMM**

**GPU vs CPU GEMV**

**GPU**: GTX280 (240 cores @ 1.30GHz, 141 GB/s)

**CPU**: 2 x 4 cores Intel Xeon @ 2.33GHz, 10.4 GB/s
## 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><strong>LINPACK (70’s)</strong>&lt;br&gt;(Vector operations)</td>
</tr>
<tr>
<td><strong>LAPACK (80’s)</strong>&lt;br&gt;(Blocking, cache friendly)</td>
</tr>
<tr>
<td><strong>ScalAPACK (90’s)</strong>&lt;br&gt;(Distributed Memory)</td>
</tr>
<tr>
<td><strong>PLASMA (00’s)</strong>&lt;br&gt;New Algorithms&lt;br&gt;(many-core friendly)</td>
</tr>
</tbody>
</table>

"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 panel factorizations use **localized** (over tiles) elementary transformations

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.
LAPACK to MAGMA
(multicore with GPU accelerators)

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

2) **HYBRIDIZATION** of linear algebra algorithms
   - Represent the 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
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
[1] B = B - A'*A
[2] B = chol(B, 'lower')
[3] D = D - C*A'
[4] D = D\B

LAPACK code
ssyrk_(['L', 'N'], &nb, Bj, &mone, hA[j,0], ...)  
spotrf_(['L', &nb, hA[j,j], lda, info])  
sgemm_(['N', 'T', Bj], ...)  
strsm_(['R', 'L', 'T', 'N', Bj], ...)  

Hybrid code
cublasSsyrk_(['L', 'N', nb, j, mone, da[j,0], ...])  
cublasGetMatrix(nb, nb, 4, da[j,j], &lda, hwork, nb)  
cublasSsgemm_(['N', 'T', j, ...])  
cublasSpotrf_(['L', &nb, hwork, &nb, info])  
cublasSetMatrix(nb, nb, 4, hwork, nb, da[j,j], &lda)  
cublasStrsm_(['R', 'L', 'T', 'N', j, ...])
```
One-sided hybrid factorizations

QR factorization in single precision arithmetic, CPU interface

Performance of MAGMA vs MKL

MAGMA QR time breakdown

Matrix size x 1000

GFlop/s

0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320

1 2 3 4 5 6 7 8 9 10

Matrix size x 1000

Time

0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

CPU

MAGMA

MKL 8 cores

MKL 1 core

GPU

CPU

CPU+GPU

GPU

Overhead

MAGMA : 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 ]
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.
    
    $x_{i+1} = x_i + (LU_{SP})^{-1} P (b - A x_i)$
Extension to Multicore and Multi GPUs

MAGNUM tile algorithms for multiGPUs

Hybrid (CPU and GPU) operations on MAGNUM tiles
Performance using MultiGPUs

Cholesky factorization in SP
Strong Scalability

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

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**”
Two-sided matrix factorizations

- Two-sided factorizations
  \[ Q A Q' = H \]
  \( H \) – upper Hessenberg / bidiagonal / tridiagonal,
  \( Q \) – 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
  \[ Q = H_1 H_2 \ldots H_{n-1}, \quad H_i = I - \tau_i v_i v_i' \]
  \( H_1 \ldots H_{n_b} = I - V T V' \) (*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?

Hessenberg factorization in double precision arithmetic, CPU interface
Performance of MAGMA vs MKL

- There have been difficulties in accelerating it on homogeneous multicores
The Bottleneck

Hessenberg factorization

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

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

\[ y_j = A_j v_j \]

bidiagonalization & tridiagonalization have even more Level 2 BLAS (50%)

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%) / 4</td>
<td>3.5 (60%) / 2.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2 BLAS</td>
<td>59 (70%) / 59</td>
<td>2.3 (40%) / 2.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No improvement
Hybrid computing acceleration?

- Intuitively, yes, as matrix-vector product is fast on GPUs (e.g., sgemv up to 66 Gflop/s, ssymv up to 102 GFlop/s)

- How to organize a hybrid computation?

**DGEMV Performance**

![Graph showing DGEMV Performance](image)

- **Achieved > 100 GB/s**

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

**Bandwidth**:
- GPU: 141 GB/s
- CPU: 10.4 GB/s
Task Splitting & Task Scheduling

Task scheduling:
- Multicore+GPU
- GPU
- Multicore

Critical path:
- P₀ → G₀ → M₀
- P₁ → G₁ → M₁

Work flow:
1. Copy dPᵢ to CPU
2. Copy vᵢ to GPU
3. Copy yⱼ to CPU
4. Copy dY to CPU

CPU usage:
- C P U

GPU usage:
- G P U

Data transfer:
- dWork
- dV

Task representation:
- Mᵢ
- Pᵢ
- Gᵢ
Performance

Hessenberg factorization in double precision arithmetic, CPU interface

Performance of MAGMA vs MKL

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

(performance in single precision arithmetic)

GPU Performance

Multicore Performance

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>GPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>26 x</td>
<td>22 x</td>
</tr>
<tr>
<td>5184</td>
<td>12 x</td>
<td></td>
</tr>
<tr>
<td>9088</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13024</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GPU: NVIDIA GeForce GTX 280 (240 cores @ 1.30GHz)

CPU: Intel Xeon dual socket quad-core (8 cores @ 2.33 GHz)

GPU BLAS: CUBLAS 2.3, dgemm peak: 75 GFlop/s

CPU BLAS: MKL 10.0, dgemm peak: 65 GFlop/s
Conclusions

- Linear algebra can be significantly accelerated using GPUs
- Described a hybridization methodology to achieve this acceleration
  - high level model
  - Leverage prior developments
- Hybridization can be used for a wide set of fundamental linear algebra algorithms
  - Linear and eigen/singular-value solvers
  - Incorporated in the MAGMA library
    http://icl.cs.utk.edu/magma/