Solving general dense linear systems on hybrid multicore-GPU systems

Adrien RÉMY, Marc BABOULIN

June 14, 2012
Outline

Mono GPU hybrid solvers
- LU factorization
- CPU/GPU algorithms
- Results

Multi GPU
- Method
- Results

Ongoing Work
- GMAC
Mono GPU hybrid solvers
  LU factorization
  CPU/GPU algorithms
  Results
Multi GPU
  Method
  Results
Ongoing Work
  GMAC
Mono GPU hybrid solvers

LU factorization

CPU/GPU algorithms

Results

Multi GPU

Method

Results

Ongoing Work

GMAC
The issue of pivoting in linear systems

- General square system $Ax = b$, solved by **Gaussian Elimination**
- Difficulties when zero or small diagonal elements → interchange rows: **partial pivoting (GEPP)**
- GEPP is implemented in most numerical libraries (LAPACK...). Used in the LINPACK benchmark for TOP500 list
- Factorization $PA = LU$, where $P$ is a permutation matrix
- No floating point operation is performed in pivoting but it involves irregular movements of data
- **Communication overhead due to pivoting**: $O(n^2)$ comparisons
Pivoting is expensive

Figure: Cost of partial pivoting in LU factorization (MAGMA)
CPU 1 × Quad-Core Intel Core2 Q9300 @ 2.50 GHz - GPU C2050 @ 1.15 GHz
Right-looking block LU factorization

- **Factorization** → $A = L \ast U$
- **Pivoting** → $P \ast A = L \ast U$

1. Panel (block column) is factored using Gaussian elimination.
2. Permutations are applied to trailing submatrix.
3. Solve triangular system to compute the $b$ first rows.
4. Update trailing submatrix.
Mono GPU hybrid solvers

LU factorization

CPU/GPU algorithms

Results

Multi GPU
Method
Results

Ongoing Work

GMAC
LU implementation for GEPP in MAGMA

Figure: Block splitting in hybrid LU factorization
Initial matrix has been transferred to the GPU.

Current iteration:

- Current panel (1) is downloaded to the CPU
- (1) is factored by the CPU using GEPP and the result is sent back to the GPU
- The GPU updates (2) (next panel)
- The updated panel (2) is sent back to the CPU to be factored while the GPU updates the rest of the matrix (3) (look-ahead)

Communication issues:

- Only panels are transferred between CPU and GPU ($O(n \times b)$ data vs $O(n \times n \times b)$ computation in the updates)
- Total overlap of the panel computation by the updates for $n$ large enough.
PRBT Solver

- PRBT (Partial Random Butterfly Transformation) is an LU solver based on randomization (see [Baboulin et al., TOMS, to appear]).
- Using the PRBT solver, we solve the general linear system $Ax = b$ by the following steps:

Algorithm 1 Solving $Ax = b$ with PRBT

1. Compute randomized matrix $A_r = U^TAV$, with $U$ and $V$ recursive butterflies.
2. Factorize $A_r$ with GENP.
3. Solve $A_r y = U^Tb$.
4. Solution is $x = Vy$.

- Properties:
  - Randomization is cheap ($O(n^2)$ operations)
  - GENP is fast (take advantage of the GPU)
  - Accuracy is in practice similar to GEPP (with iterative refinement)
Panel factorization (on CPU) based on **tournament pivoting** (see [Grigori et al., SIMAX 2011])

Implemented as a reduction operation:

- Partition the panel in blocks
- Select in parallel a set of local pivots using PP
- Perform tournament on the local sets to select global pivots
- Global pivots are moved to diagonal positions and GENP is performed on the entire panel
**Figure:** Panel factorization on CPU using Tall and Skinny LU with tournament pivoting
Hybrid version of CALU (H-CALU)

Figure: Hybrid LU factorization (4 panels).
Mono GPU hybrid solvers
   LU factorization
   CPU/GPU algorithms
   Results
Multi GPU
   Method
   Results
Ongoing Work
   GMAC
Performance results

- Hybrid CPU/GPU algorithms implemented following MAGMA development guidelines
- GPU device: NVIDIA Fermi Tesla S2050 (448 CUDA cores)
  Multicore host: 4 × 12-Core AMD Opteron 6172 Magny-Cours @ 2.1 GHz, 128GB memory, theoretical peak 403.2 Gflop/s (8.4 Gflop/s per core) in double precision
- **Panel factorization**: comparisons against MKL multithreaded
- **Hybrid LU solvers**: We compare MAGMA, PRBT and H-CALU
Comparison of CPU multi-threaded panel factorizations

Matrix size = 5120, panel size = 256

Matrix size = 10240, panel size = 320

Matrix size = 15360, panel size = 512

Matrix size = 21504, panel size = 768
Performance on square matrices

![Graph showing performance on square matrices with matrix size on the x-axis and Gflop/s on the y-axis, comparing different algorithms: magma_dgetrf, H-CALU, and PRBT. The graph demonstrates the performance trend across various matrix sizes.]
Tests on accuracy

- We compare 3 solvers:
  - MAGMA/GEPP
  - H-CALU
  - PRBT (2 recursions)

- We report componentwise backward error
  \[ \omega = \max_i \frac{|Ax - b|_i}{(|A| \cdot |x| + |b|)_i} \]

- Iterative refinement is systematically added

- Test matrices from the LAPACK tester:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Diagonal</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Upper triangular</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Lower triangular</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Random, ( \kappa = 2 )</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>First column zero</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>Last column zero</td>
<td></td>
</tr>
</tbody>
</table>
### Table: Componentwise Backward Error

<table>
<thead>
<tr>
<th>Matrix Type</th>
<th>MAGMA LU (magma_dgetrf)</th>
<th>H-CALU</th>
<th>PRBT</th>
<th>No pivoting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>1.42e-16(1)</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>1.32e-16</td>
<td>1.32e-16</td>
<td>4.02e-16(3)</td>
<td>6.19e-16</td>
</tr>
<tr>
<td>3</td>
<td>1.85e-16</td>
<td>1.85e-16</td>
<td>2.46e-16(3)</td>
<td>2.14e-16</td>
</tr>
<tr>
<td>4</td>
<td>2.16e-16</td>
<td>2.76e-16</td>
<td>2.93e-16(2)</td>
<td>1.13e-11</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>2.10e-16</td>
<td>3.76e-16</td>
<td>2.64e-16(3)</td>
<td>2.94e-12</td>
</tr>
<tr>
<td>9</td>
<td>2.70e-16</td>
<td>6.37e-16</td>
<td>1.16e-13(1)</td>
<td>1.41e-13</td>
</tr>
<tr>
<td>10</td>
<td>7.60e-14</td>
<td>7.40e-14</td>
<td>4.01e-14(2)</td>
<td>2.42e-11</td>
</tr>
<tr>
<td>11</td>
<td>2.27e-16</td>
<td>2.11e-16</td>
<td>2.41e-16(2)</td>
<td>2.90e-11</td>
</tr>
</tbody>
</table>
Mono GPU hybrid solvers
  LU factorization
  CPU/GPU algorithms
  Results

Multi GPU
  Method
  Results

Ongoing Work
  GMAC
Mono GPU hybrid solvers
   LU factorization
   CPU/GPU algorithms
   Results

Multi GPU
   Method
   Results

Ongoing Work
   GMAC
Multi GPU implementation

Original matrix is distributed among the GPUs (1D block cyclic)

Figure: LU factorization using 3 GPUs
Multi GPU implementation

First panel is sent to the CPU

Figure: LU factorization using 3 GPUs
Multi GPU implementation

Figure: LU factorization using 3 GPUs
Multi GPU implementation

Factored panel is sent to the GPUs

**Figure:** LU factorization using 3 GPUs
Multi GPU implementation

GPUs update trailing submatrices

Figure: LU factorization using 3 GPUs
Multi GPU implementation

Figure: LU factorization using 3 GPUs
Multi GPU implementation

CPU factors new the panel while GPUs still update trailing submatrices

Figure: LU factorization using 3 GPUs
Mono GPU hybrid solvers
   LU factorization
   CPU/GPU algorithms
Results
Multi GPU
   Method
Results
Ongoing Work
GMAC
Performance results

Multi GPU LU factorization

- No pivoting (optimized for PRBT)
- Partial pivoting

<table>
<thead>
<tr>
<th>Speedup</th>
<th>Number of GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>1</td>
</tr>
<tr>
<td>1.4</td>
<td>2</td>
</tr>
<tr>
<td>1.6</td>
<td>3</td>
</tr>
<tr>
<td>1.8</td>
<td>4</td>
</tr>
</tbody>
</table>

- Size = 5120
- Size = 10240
- Size = 15360
- Size = 20480
Mono GPU hybrid solvers
  LU factorization
  CPU/GPU algorithms
  Results

Multi GPU
  Method
  Results

Ongoing Work
  GMAC
Going to a bigger scale

- Using clusters of GPUs
- Distributed memory version of LU factorization with multiple GPUs **in progress**
- Using GMAC for managing communication
Mono GPU hybrid solvers
   LU factorization
   CPU/GPU algorithms
   Results

Multi GPU
   Method
   Results

Ongoing Work
   GMAC
What is GMAC?

- GMAC [I. Gelado et al. ASPLOS’10](Global Memory for ACcelerators) provides unified virtual address for CUDA
- Simplify the CPU code
- Single virtual address space for CPUs and GPUs
- Provide advanced CUDA features for free:
  - Asynchronous data transfer
  - Pinned memory
  - GPU to GPU communication
  - Get access to any GPU from any CPU thread
- Collaboration with Wen-Mei Hwu (University of Illinois at Urbana-Champaign)
Efficient and accurate solvers for hybrid architectures:
- Solutions for multicore accelerated with **one GPU**
- Solutions for multicore accelerated with **several GPUs**
- Give similar accuracy results on most test cases

Difference between the solvers comes from the pivoting strategy for factoring the panel

Distributed version in progress to use clusters of GPUs
Related papers

[1] M. Baboulin, S. Donfack, J. Dongarra, L. Grigori, A. Rémy, S. Tomov,
A class of algorithms for solving general dense linear systems on CPU/GPU parallel machines.

[2] M. Baboulin, D. Becker, J. Dongarra,
A Parallel Tiled Solver for Dense Symmetric Indefinite Systems on Multicore Architectures.
Proceedings of IPDPS 2012.

[3] M. Baboulin, J. Dongarra, J. Herrmann, S. Tomov,
Accelerating linear system solutions using randomization techniques.
To appear in ACM Transactions on Mathematical Software (TOMS), LAPACK Working Note 246.

An asymmetric distributed shared memory model for heterogeneous parallel systems.

[5] S. Tomov, J. Dongarra, M. Baboulin,
Towards dense linear algebra for hybrid GPU accelerated manycore systems.

[6] M. Baboulin, J. Dongarra, S. Tomov,
Some issues in dense linear algebra for multicore and special purpose architectures.