Fast Solvers for Linear Systems on the GPU

GPU Technology Conference (GTC), San Jose, March 23-27, 2014
Session S4299
Kees Vuik, Rohit Gupta, Martijn de Jong
Martin van Gijzen, Auke Ditzel (MARIN), Auke van der Ploeg (MARIN)
March 25th, 2014, c.vuil@tudelft.nl, GTC S4299
Contents

1. Problem description
2. Preconditioners
   • RBB preconditioner
   • Truncated Neumann Series (TNS)
   • Deflation
3. Numerical results
4. Conclusions
Main question:

Can ILU preconditioners be combined with GPU’s?
1. Problem description: ship simulator

Linearized Variational Boussinesq for interactive waves:

\[
\frac{\partial \zeta}{\partial t} + \nabla \cdot (\zeta \mathbf{U} + h \nabla \varphi - hD \nabla \psi) = 0, \quad (1a)
\]

\[
\frac{\partial \varphi}{\partial t} + \mathbf{U} \cdot \nabla \varphi + g \zeta = -P_s, \quad (1b)
\]

\[
\mathcal{M} \psi + \nabla \cdot (hD \nabla \varphi - N \nabla \psi) = 0. \quad (1c)
\]

After discretization (FVM for space, Leapfrog for time):

\[
A \vec{\psi} = \mathbf{b}, \quad (2)
\]

\[
\frac{dq}{dt} = Lq + \mathbf{f}. \quad (3)
\]
Properties of the system matrix

The system matrix $A$ is given by a 5-point stencil:

$$
egin{bmatrix}
0 & -\frac{\Delta y}{\Delta x} \mathcal{N}_W & \frac{\Delta x}{\Delta y} \mathcal{N}_N + \frac{\Delta y}{\Delta x} \mathcal{N}_E + \Delta x \Delta y \mathcal{M}_C + \frac{\Delta x}{\Delta y} \mathcal{N}_S + \frac{\Delta y}{\Delta x} \mathcal{N}_W & 0 \\
-\frac{\Delta y}{\Delta x} \mathcal{N}_W & \frac{\Delta x}{\Delta y} \mathcal{N}_N + \frac{\Delta y}{\Delta x} \mathcal{N}_E + \Delta x \Delta y \mathcal{M}_C + \frac{\Delta x}{\Delta y} \mathcal{N}_S + \frac{\Delta y}{\Delta x} \mathcal{N}_W & 0 \\
0 & -\frac{\Delta y}{\Delta x} \mathcal{N}_S & -\frac{\Delta x}{\Delta y} \mathcal{N}_S & -\frac{\Delta y}{\Delta x} \mathcal{N}_E \\
\end{bmatrix}.
$$

Matrix $A$ is:

- real-valued, sparse (5-point, pentadiagonal)
- diagonally dominant (not very strong for small mesh sizes)
- symmetric positive definite (SPD)
- quite large
Problem Description: Bubbly Flow

Mass-Conserving Level-Set method for Navier Stokes

\[ -\nabla \cdot \left( \frac{1}{\rho(x)} \nabla p(x) \right) = f(x), \ x \in \Omega \]  (4)

\[ \frac{\partial}{\partial n} p(x) = 0, \ x \in \partial \Omega \]  (5)

• Pressure-Correction equation is discretized to \( Ax = b \).

• Most time consuming part is the solution of this SPSD system
2. Preconditioners: RRB

The RRB-solver:

- is a PCG-type solver (Preconditioned Conjugate Gradient)
- uses as preconditioner: the RRB preconditioner

RRB stands for “Repeated Red-Black”.

The RRB preconditioner determines an incomplete factorization:

\[ A = LDL^T + R \quad \Longrightarrow \quad M = LDL^T \approx A \]
Preconditioners: RRB

As the name RRB reveals: multiple levels

Therefore the RRB-solver has good scaling behaviour (Multigrid)

Method of choice because:

• shown to be robust for all of MARIN’s test problems

• solved all test problems up to 1.5 million nodes within 7 iterations(!)
Special ordering (1)

An $8 \times 8$ example of the RRB-numbering process

All levels combined:
Special ordering (2)

Effect on sparsity pattern of matrix $A$:

Lexicographic  

becomes

RRB-numbering
Special ordering (3)

Sparsity pattern of matrix $A$ versus $L + D + L^T$

(recall preconditioner $M = LDL^T$)

In the blue shaded areas fill-in has been dropped (lumping)
CUDA implementation (1)

Besides the typical Multigrid issues such as idle cores on the coarsest levels, in CUDA the main problem was getting “coalesced memory transfers”.

Why is that?

Recall the RRB-numbering: the number of nodes becomes $4 \times$ smaller on every next level:
CUDA implementation (2)

New storage scheme: $r_1/r_2/b_1/b_2$

Nodes are divided into four groups:
Preconditioners: TNS

Truncated Neumann Series Preconditioning$^a,^b$

\[ M^{-1} = K^T D^{-1} K, \text{ where } K = (I - LD^{-1} + (LD^{-1})^2 + \cdots) \]

$L$ is the strictly lower triangular of $A$, and $D=\text{diag}(A)$.

1. More terms give better approximation.

2. In general the series converges if $\|LD^{-1}\|_\infty < 1$.

3. As much parallelism as Sparse Matrix Vector Product.

---


Preconditioners: Deflation

Removes small eigenvalues from the spectrum of $M^{-1}A$. The linear system $Ax = b$ can be solved by the splitting,

$$x = (I - P^T)x + P^T x \text{ where } P = I - AQ. \tag{6}$$

$$\Leftrightarrow Pb = PA\hat{x}. \tag{7}$$

$$Q = ZE^{-1}Z^T, \quad E = Z^T AZ.$$  

$Em = a1$ is the coarse system

$Z$ is an approximation of the 'bad' eigenvectors of $M^{-1}A$.

For our experiments $Z$ consists of piecewise constant vectors.
Preconditioners: Deflation

Operations involved in deflation\textsuperscript{a, b}.

- \( a_1 = Z^T p. \)
- \( m = E^{-1} a_1. \)
- \( a_2 = AZm. \)
- \( \hat{w} = p - a_2. \)

where, \( E = Z^T AZ \) is the Galerkin Matrix and \( Z \) is the matrix of deflation vectors.


3. Numerical results: ship simulator

- Including: 2D Poisson, Gelderse IJssel (NL), Plymouth Sound (UK)
- Realistic domains up to 1.5 million nodes
Numerical results: ship simulator

![Graph showing speed up numbers for realistic test problems.](image)

**Speed up numbers for the realistic test problems.**
Numerical results: Bubbly flow

\[ Speedup = \frac{T_{CPU}}{T_{GPU}} \]  \hspace{1cm} (8)

- Number of Unknowns = 128^3.
- Tolerance set to 10^{-6}.
- Density Contrast is 10^{-3}

Naming deflation vectors

- SD-i -> Sub-domain deflation with i vectors.
- LS-i -> Level-Set deflation with i vectors.
- LSSD-i -> Level-Set Sub-domain deflation with i vectors.
Numerical results: Bubbly flow

9 bubbles - 64 Sub-domains

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU-CUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DICCG(0)</td>
<td>DICCG(0)</td>
<td>DICCG(TNS)</td>
</tr>
<tr>
<td>SD-64</td>
<td>SD-63</td>
<td>LSSD-135</td>
</tr>
<tr>
<td>Number of Iterations</td>
<td>472</td>
<td>603</td>
</tr>
<tr>
<td>Total Time</td>
<td>81.39</td>
<td>13.61</td>
</tr>
<tr>
<td>Iteration Time</td>
<td>81.1</td>
<td>10.61</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
<td>7.64</td>
</tr>
</tbody>
</table>
Numerical results: Bubbly flow

9 bubbles - 512 Sub-domains

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU-CUSP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DICCG(0)</td>
<td>DPCG(TNS)</td>
</tr>
<tr>
<td>Number of Iterations</td>
<td>67</td>
<td>81</td>
</tr>
<tr>
<td>Total Time</td>
<td>12.51</td>
<td>4.56</td>
</tr>
<tr>
<td>Iteration Time</td>
<td>12.18</td>
<td>1.56</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
<td>7.81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.52</td>
</tr>
</tbody>
</table>
Paralution\textsuperscript{a} results: Bubbly flow

Intel Xeon dual quad-core CPU versus NVIDIA K20 GPU

<table>
<thead>
<tr>
<th></th>
<th>ME-ILU-J</th>
<th>ILU(0)</th>
<th>FSAI</th>
<th>MCSGS</th>
<th>TNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>210</td>
<td>142</td>
<td>163</td>
<td>203</td>
<td>175</td>
</tr>
<tr>
<td>CPU Time</td>
<td>9.324</td>
<td>17.1</td>
<td>19.53</td>
<td>9.843</td>
<td>8.61</td>
</tr>
<tr>
<td>GPU Time</td>
<td>2.74</td>
<td>5.165</td>
<td>6.68</td>
<td>2.841</td>
<td>2.25</td>
</tr>
</tbody>
</table>

Comparing Deflated PCG for different preconditioners

Grid Size - $128^3$. 8 bubbles.

\textsuperscript{a}http://www.paralution.com/
4. Conclusions

- ILU type preconditioners can be used on GPU’s by a Neumann series approach or a careful reordering.
- Deflation type preconditioners are very suitable for GPU’s.
- The combination of Neumann series and Deflation preconditioners leads to robust and fast solvers on the GPU.
- A special ordering of a red black reordering can lead to speedup of a factor 30-40 on the GPU.
Main question

Can ILU preconditioners be combined with GPU’s?
Main question

Can ILU preconditioners be combined with GPU’s?

YES
References


- Rohit Gupta thesis defence is in the second half of 2014

For CV etc.: http://ta.twi.tudelft.nl/nw/users/rohit/