Mathematical Methods in Fluid Dynamics and Simulation of Giant Oil and Gas Reservoirs

3-5 September 2012
Swissotel The Bosphorus, Istanbul, Turkey
Fast and robust solvers for pressure systems on the GPU

Kees Vuik

Delft University of Technology

http://ta.twi.tudelft.nl/users/vuik/
Contents

1. Large Jumps

2. Large Grids

3. Large Computers

4. Summary
1. Large Jumps

Model Problem

\[-\nabla(\mathbf{K}\nabla u) = f\]

- \(K = 1\)
- \(K = 10^{-3}\)
- \(K = 1\)
- \(K = 10^{-3}\)

Test case taken from [Vuik et al., 1999]
Ax = b
A is sparse and SPD
Condition number of A is large, due to large contrast in permeability

Applications

• Reservoir simulations
• Porous media flow
• Fictitious domain methods
Convergence of CG

Convergence behavior of CG without preconditioning
Convergence of CG

Convergence behavior of CG without preconditioning
Convergence of CG

Convergence behavior of ICCG
Idea: remove the bad eigenvectors from the error/residual.

Krylov

Preconditioned Krylov

Block Preconditioned Krylov

Block Preconditioned Deflated Krylov
DICCG

\[ k = 0, \hat{r}_0 = Pr_0, \quad p_1 = z_1 = L^{-T}L^{-1}\hat{r}_0; \]

while \( \|\hat{r}_k\|_2 > \varepsilon \) do

\[ k = k + 1; \]

\[ \alpha_k = \frac{(\hat{r}_{k-1}, z_{k-1})}{(p_k, PAp_k)}; \]

\[ x_k = x_{k-1} + \alpha_k p_k; \]

\[ \hat{r}_k = \hat{r}_{k-1} - \alpha_k PAp_k; \]

\[ z_k = L^{-T}L^{-1}\hat{r}_k; \]

\[ \beta_k = \frac{(\hat{r}_k, z_k)}{(\hat{r}_{k-1}, z_{k-1})}; \]

\[ p_{k+1} = z_k + \beta_k p_k; \]

end while
Geometry oil flow problem
### Results oil flow problem

#### Varying $\sigma_{\text{shale}}$

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>ICCG $\lambda_{\text{min}}$</th>
<th>iter</th>
<th>DICCG $\lambda_{\text{min}}$</th>
<th>iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-3}$</td>
<td>$1.5 \cdot 10^{-2}$</td>
<td>26</td>
<td>$6.9 \cdot 10^{-2}$</td>
<td>20</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>$2.2 \cdot 10^{-4}$</td>
<td>59</td>
<td>$7.7 \cdot 10^{-2}$</td>
<td>20</td>
</tr>
<tr>
<td>$10^{-7}$</td>
<td>$2.3 \cdot 10^{-6}$</td>
<td>82</td>
<td>$7.7 \cdot 10^{-2}$</td>
<td>20</td>
</tr>
</tbody>
</table>

#### Varying accuracy

<table>
<thead>
<tr>
<th>accuracy</th>
<th>ICCG iter</th>
<th>CPU</th>
<th>DICCG iter</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-5}$</td>
<td>82</td>
<td>18.9</td>
<td>20</td>
<td>6.3</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>78</td>
<td>18.0</td>
<td>12</td>
<td>4.1</td>
</tr>
<tr>
<td>$10^{-1}$</td>
<td>75</td>
<td>17.2</td>
<td>2</td>
<td>1.2</td>
</tr>
</tbody>
</table>
A groundwater flow problem

The pressure in groundwater satisfies the equation:

$$-\nabla \cdot (A \nabla u) = F,$$

where the coefficients and geometry of the problem are:
A groundwater flow problem

The low permeable layer ($A = 10^{-5}$) and the jump in permeabilities between the two sand sections lead to a 'small' eigenvalue.
DG Methods
DG methods are like FVM, but then based on piecewise polynomials.

DG for elliptic problems: [Arnold et al., 2002], [Riviére, 2008]
Coarse corrections

The main idea is to speed up CG using coarse corrections based on $p = 0$

DG matrix $A$ with $p > 0$

$$A^{-1} \approx Q := R^T (RAR^T)^{-1} R$$

prolongation

restriction

DG matrix $RAR^T$ with $p = 0$

Original idea of spectral multigrid: [Rønquist and Patera, 1987]
Deflation variant

We can switch to deflation by simply skipping a smoothing step.

The result $z$ of applying two-level deflation to a vector $r$:

\[
\begin{align*}
  z^{(1)} &= \omega M^{-1} r, \\
  z^{(2)} &= z^{(1)} + Q(r - Az^{(1)}), \\
  z &= z^{(2)} + \omega M^{-T}(r - Az^{(2)}),
\end{align*}
\]

apply smoother $\omega M^{-1}$  
apply coarse correction $Q$  
apply smoother $\omega M^{-T}$

Requirement: $M + M^T \omega A$ is SPD  
$M$ is SPD  
assuming we pre-process the CG start vector once:

\[
x_0 \rightarrow Qb + (I - AQ)^T x_0,
\]

The result is equivalent to a CG method with an SPD preconditioner: [Tang et al., 2009]
### Layered problem

<table>
<thead>
<tr>
<th>degree mesh size</th>
<th>$A=20^2$</th>
<th>$N=40^2$</th>
<th>$N=80^2$</th>
<th>$N=160^2$</th>
<th>$A=20^2$</th>
<th>$N=40^2$</th>
<th>$N=80^2$</th>
<th>$N=160^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
<td>975</td>
<td>1264</td>
<td>1567</td>
<td>2314</td>
<td>1295</td>
<td>1490</td>
<td>1921</td>
<td>3110</td>
</tr>
<tr>
<td>block Jacobi (BJ)</td>
<td>243</td>
<td>424</td>
<td>788</td>
<td>1285</td>
<td>244</td>
<td>425</td>
<td>697</td>
<td>1485</td>
</tr>
<tr>
<td>two-level prec., 2x BJ</td>
<td>46</td>
<td>43</td>
<td>43</td>
<td>44</td>
<td>55</td>
<td>56</td>
<td>56</td>
<td>57</td>
</tr>
<tr>
<td>two-level defl., 1x BJ</td>
<td>43</td>
<td>45</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
</tbody>
</table>

**CG stopping criterion:** \[ \frac{\|b-Ax_k\|_2}{\|b\|_2} \leq 10^{-6} \]

*Diagonal-scaling is applied beforehand*
Groundwater flow

<table>
<thead>
<tr>
<th>degree mesh</th>
<th>p=2</th>
<th>p=3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N=20²</td>
<td>N=40²</td>
</tr>
<tr>
<td>two-level prec. (ω = 1)</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>two-level prec. (ω = 0.7)</td>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td>two-level defl. (ω = 1)</td>
<td>52</td>
<td>54</td>
</tr>
</tbody>
</table>

Test case taken from [Vuik et al., 2001]
Variational Boussinesq model (VBM) as proposed by Gert Klopman

Linearized VBM equations:

\[ \frac{\partial \zeta}{\partial t} + \nabla \cdot (\zeta \mathbf{U} + h \nabla \varphi - h \mathcal{D} \nabla \psi) = 0, \]

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

\[ \mathcal{M} \psi + \nabla \cdot (h \mathcal{D} \nabla \varphi - h \mathcal{N} \nabla \psi) = 0. \]
3. Large computers (GPU, MSc, Martijn de Jong)

- Equidistant rectangle grid (not mandatory)
- Finite volume method (FVM) for space
- Leapfrog method for time integration

After discretization:

\[ S\vec{\psi} = b, \]

\[ \frac{dq}{dt} = Lq + f. \]

Solve them consecutively in real-time
Matrix $S$ is given by a 5-point stencil:

$$
\begin{bmatrix}
0 & \frac{\Delta y}{\Delta x} \bar{N}_W & \frac{\Delta x}{\Delta y} \bar{N}_N & -\frac{\Delta x}{\Delta y} \bar{N}_E & 0 \\
-\frac{\Delta y}{\Delta x} \bar{N}_W & 0 & \frac{\Delta x}{\Delta y} \bar{N}_N + \Delta x \Delta y M_C & \frac{\Delta x}{\Delta y} \bar{N}_S + \frac{\Delta y}{\Delta x} \bar{N}_W & -\frac{\Delta y}{\Delta x} \bar{N}_E \\
\end{bmatrix}
$$

Matrix $S$ is:

- real-valued, sparse (5-point, pentadiagonal)
- diagonally dominant (not very strong for small mesh sizes)
- symmetric positive definite (SPD)
- quite large (in the order of millions by millions)
The RRB-solver:

- is a PCG-type solver (preconditioned conjugated gradients)
- uses as preconditioner: the RRB-method

RRB stands for “Repeated Red-Black”.

The RRB-method determines an incomplete factorization:

\[ S = LDL^T + R \quad \Rightarrow \quad M = LDL^T \approx S \]
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(!)
An $8 \times 8$ example of the RRB-numbering process

(1)

(2)

(3)

(4)

All levels combined:
Effect on sparsity pattern of matrix $S$:

- Lexicographic
- RRB-numbering

becomes
Special ordering

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

(recall preconditioner $M = LDL^T$)

In the blue shaded areas fill-in has been dropped (lumping)
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:

![Diagram showing RRB-numbering with nodes from 29 to 64]

(1)  (2)  (3)  (4)

57  49  58  50

61  63

54

50
CUDA implementation

- Data is read from and written to the device’s global memory via 32-, 64- of 128-byte memory transfers
- Example: reading data with a stride
CUDA implementation

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

Nodes are divided in four groups:
The $r1/r2/b1/b2$-storage scheme

• is applied on every next coarser level till the point that the remaining level is smaller than $32 \times 32$ elements; the last levels are solved in one go on 1 streaming multiprocessor (SM) exploiting the benefits of cache

• almost comes for free (only at the beginning and ending of CG we have some overhead due to reordering of the data)

• allows for coalesced memory read and write operations throughout the entire CG algorithm which yields optimal throughput
Kernel throughput up to 250 GB/s (thanks to cache)

- Solver speed up is up to $30 \times$ for realistic problems of 1.5 million nodes and up to $40 \times$ for even larger problems (> $2048 \times 2048$ nodes)

- Time needed? Merely 10 milliseconds for 7 CG-iterations (vs. 300 ms for C++)

- The fast CUDA solver allows real-time simulation

- Also the RRB-preconditioner can be constructed in real-time and hence varying bathymetry across time is supported
• Building blocks for fast and robust solvers for pressure systems on the GPU are given

• Deflation can reduce the condition number, number of iterations, and CPU time considerably

• High accuracy DG methods can greatly reduce the grid size

• Deflation type solvers lead to scalable solvers for DG problems

• RRB solver leads to scalable convergence for Poisson type problems

• Clever reordering leads to speed up of a factor 40 on the GPU
References

Thank You