A direct-iterative hybrid block linear solver for discontinuous-Galerkin finite-element equations

Steven Murphy\textsuperscript{1}  Iain Duff\textsuperscript{2}

\textsuperscript{1}CERFACS, Toulouse & University of Nottingham

\textsuperscript{2}RAL, UK & CERFACS, Toulouse

18 June 2013
Talk structure

1. Introduction
2. Block direct solver
3. Sequential hybrid solver
4. Parallel hybrid solver
• Have sparse block matrices with dense blocks
• Can come from discontinuous-Galerkin methods
• Block structure has a direct correspondence to the finite element mesh
Motivation

- Generally unsymmetric
- Diagonal blocks correspond to finite elements
- Off diagonal blocks correspond to face boundaries
- Diagonal blocks are dense, square, non singular and are of size $(p + 1)^d$
- With $p$-refinement the sizes of the diagonal blocks vary
- With $h$-refinement the block structure varies
Storage Format

- Can represent the block structure in reduced format by letting each block correspond to a single entry in a reduced matrix.
- Store all blocks in dense format, to facilitate application of BLAS and LAPACK routines.
- Structure of the blocks may be analysed, with the results expanded to the full system.

Pattern of the matrix Poisson 1

nz = 20186
Using block ANALYSE

- Can utilise this block sparse structure to speed direct solver
- Multifrontal solvers work in three phases: ANALYSE, FACTORIZE, SOLVE
- Use the ANALYSE from the HSL solver MA57 to analyse just the block structure
- Expand the ordering and tree data from MA57 ANALYSE back to full matrix before proceeding to FACTORIZE
<table>
<thead>
<tr>
<th>Full Problem Size, $N$</th>
<th>Full Problem $NE$</th>
<th>Block Problem Size, $BN$</th>
<th>Block Problem $BNE$</th>
<th>Average block size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1225</td>
<td>45793</td>
<td>49</td>
<td>140</td>
<td>25.0</td>
</tr>
<tr>
<td>2125</td>
<td>94226</td>
<td>85</td>
<td>254</td>
<td>25.0</td>
</tr>
<tr>
<td>4000</td>
<td>165274</td>
<td>160</td>
<td>468</td>
<td>25.0</td>
</tr>
<tr>
<td>4696</td>
<td>227302</td>
<td>172</td>
<td>508</td>
<td>27.3</td>
</tr>
<tr>
<td>6168</td>
<td>327911</td>
<td>208</td>
<td>610</td>
<td>29.7</td>
</tr>
<tr>
<td>7624</td>
<td>467562</td>
<td>244</td>
<td>734</td>
<td>31.2</td>
</tr>
<tr>
<td>10511</td>
<td>729935</td>
<td>319</td>
<td>975</td>
<td>32.9</td>
</tr>
<tr>
<td>12946</td>
<td>971365</td>
<td>379</td>
<td>1158</td>
<td>34.2</td>
</tr>
</tbody>
</table>

- Test with a set of matrices from a DG problem
- Problems from a higher order DG method for Poisson’s equation in 2D.
- Notice a speedup when replacing MA57 with MA57 with block ANALYSE
- Speedup for ANALYSE, FACTORIZE and SOLVE is much greater than speedup for ANALYSE
• Substantial reduction in the number of integers required to store the factorized matrix
• Moderate reduction in the real storage required

Murphy, Duff

A direct-iterative hybrid block linear solver
- Same ordering, from MA57 run on blocks, was used to get a moderate speed up in MA41
- Ordering preserves block structure, though tree data was not preserved when moving from MA57 ANALYSE to MA41 FACTORIZE
What do we mean by hybrid?

- Sparse linear solvers classified as either direct or iterative
- Each has its own advantages and disadvantages

**Direct**

- Robust and numerically stable
- Accurate solutions
- No preconditioning required
- Can require a lot of memory for large problems

**Iterative**

- Greater control over memory
- Require preconditioning
- Can be highly tuned to the problem
- With good preconditioner can be fast and low memory
Goal

- Seek to create a solver which combines the best of iterative and direct solvers
- Want to use parameters to control the extent to which it’s a direct solver and an iterative solver
- Look to find where to set those parameters in order to get best compromise between speed and memory
Hybrid Solver - Method

- To solve the linear system $Ax = b$
- The solve uses an outer GMRES loop, preconditioned by an overlapping additive Schwarz preconditioner

$$AM^{-1}(Mx) = b$$

- Where $M$ is a preconditioner constructed from a series of reduced matrices $A_i$ and their restriction operators $R_i$

$$M^{-1} = \sum_i R_i A_i^{-1} R_i^T$$

- The choice of the $R_i$ and $A_i$ determined by multifrontal solver analysis of the matrix structure
Defining the domain decomposition

- Seek an overlapping partition \( \{ \Omega_i \} \) of the computational domain, \( \Omega \):

\[
\bigcup_{i} \Omega_i = \Omega \enspace , \enspace \bigcap_{i} \Omega_i \neq \emptyset
\]

- Tree data from modified MA57 ANALYSE run on block structure can define this partition
- Parameter MERGE controls the sizes of the \( \Omega_i \)
- Parameter OVERLAP to controls the overlap between neighbouring \( \Omega_i \)
- Build preconditioner by building the reduced matrices \( A_i \) corresponding the \( \Omega_i \) and factorizing them
An example of the effect of the overlap parameter
A domain decomposition on a larger domain
- OVERLAP = 1 greatly improves speed. Increasing it further makes little difference
- Comparison of the hybrid solver against SPARSEKIT ILUT preconditioner, given same memory
Parallel implementation

- How well suited to a parallel implementation is this method?
- 3 primary operations to be performed: matrix vector product, preconditioning and inner product

Matrix vector product: \( z \leftarrow Ax \)

Precondition: \( z \leftarrow \sum_i R_i A_i^{-1} R_i^T x \)

Inner product: \( z \leftarrow \langle x, x \rangle \)

- Divide the problem between the MPI processes seeking to facilitate performing these operations
Parallel implementation

- Have 2 goals: minimise communication and balance work
- Tried 2 approaches to split problem between processes

**Greedy load balance**
- Use same sequential algorithm to split the computational domain into subdomains \( \Omega = \bigcup_i \Omega_i \)
- Use greedy algorithm to divide \( \Omega_i \) between processes

**Metis partition**
- Use Metis on block structure to partition domain between processes \( \Omega = \bigcup_i \Omega_p \)
- Use sequential algorithm on each \( \Omega_p \) to get final overlapping \( \Omega_i \)
Parallel implementation

- Tested the parallel matrix vector product and preconditioning operations on a matrix $N \approx 35000$
- Found that without first using Metis to divide the problem between processors, each scaled very poorly
- Also found that increasing the OVERLAP between subdomains was hugely detrimental to performance
- Can be explained by considering the communication

<table>
<thead>
<tr>
<th>Number Processes</th>
<th>OVERLAP</th>
<th>Greedy algorithm communication Mat Vec</th>
<th>Greedy algorithm communication Precon</th>
<th>Metis communication Mat Vec</th>
<th>Metis communication Precon</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>28696</td>
<td>28696</td>
<td>2262</td>
<td>2262</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>28696</td>
<td>34899</td>
<td>2262</td>
<td>4804</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>54934</td>
<td>54934</td>
<td>5646</td>
<td>5646</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>54934</td>
<td>94661</td>
<td>5646</td>
<td>12008</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>69328</td>
<td>69328</td>
<td>10886</td>
<td>10886</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>69328</td>
<td>153086</td>
<td>10886</td>
<td>24366</td>
</tr>
</tbody>
</table>
Conclusions

- Block structure can be utilized to speed direct solves
- Hybrid linear solver using a multifrontal ANALYSE to partition domain
- Small overlap between subdomains necessary to speed convergence, yet...
- Overlap greatly increases communication for parallel implementation
- Still a work in progress...
Thank you for listening!
Any questions?