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

Steven Murphy[1] Iain Duff[2]

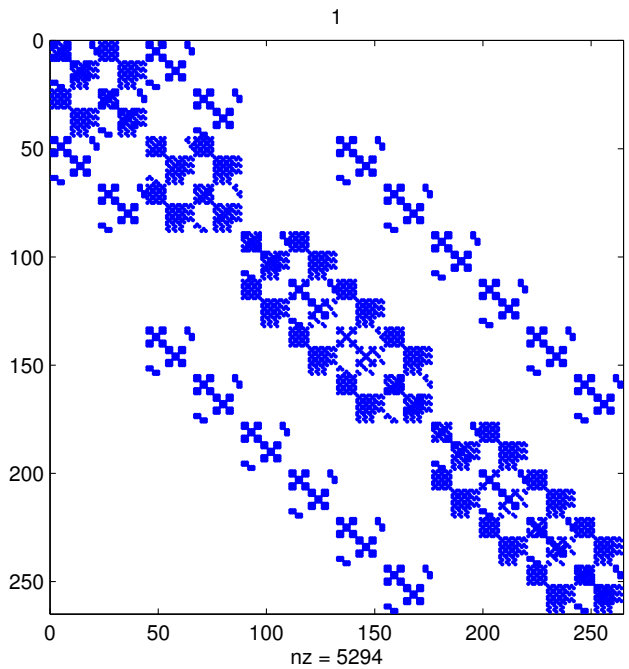
[1]CERFACS,Toulouse & University of Nottingham

[2]RAL, UK & CERFACS,Toulouse

18 June 2013

# Talk structure

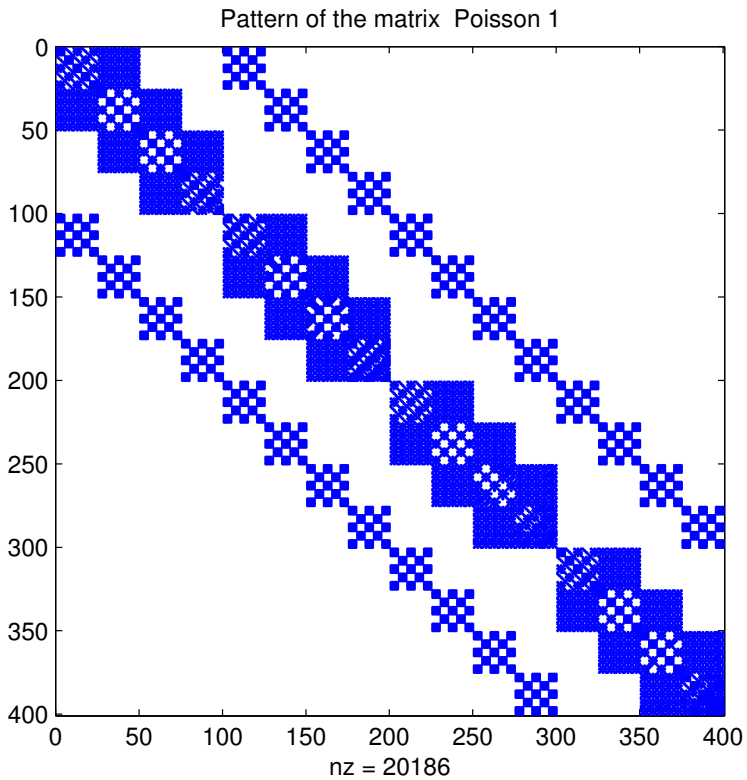
# Motivation




- Have sparse block matrices with dense blocks
- Can come from discontinuous-Galerkin methods
- Block structure has a direct correspondence to the finite element mesh

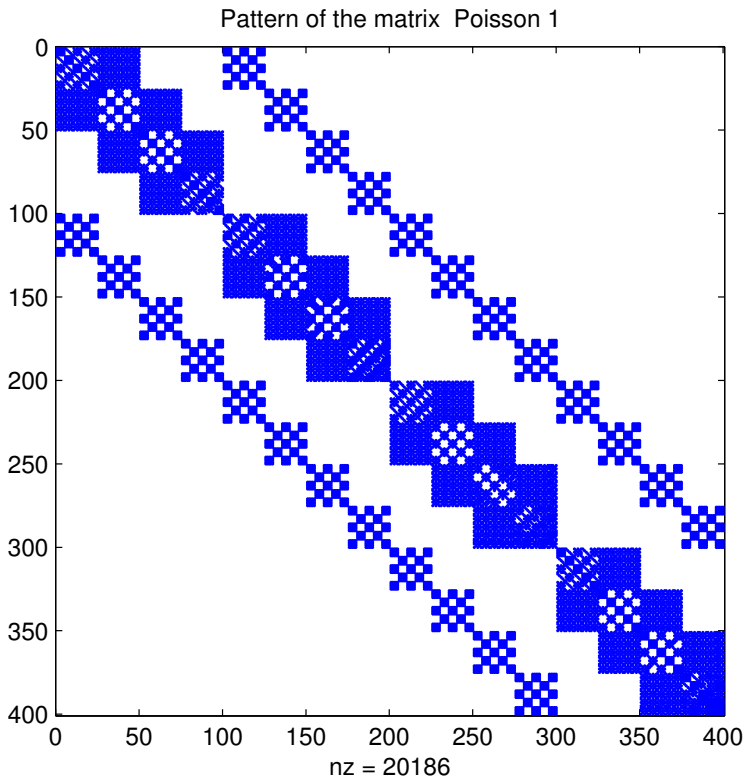
# Motivation

- 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



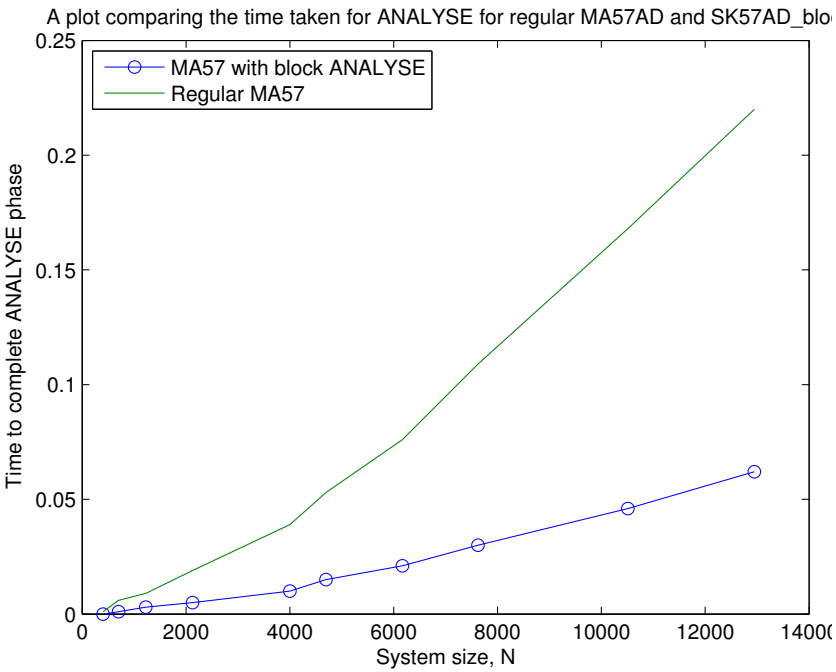
# Using block ANALYSE

- 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

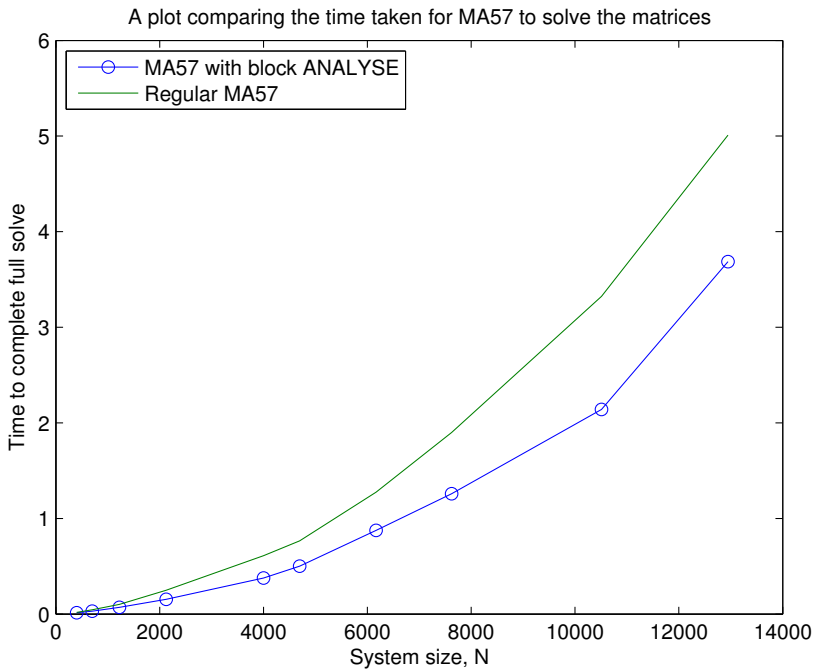
| Full Problem Size, $N$ | Full Problem $NE$ | Block Problem Size, $BN$ | Block Problem $BNE$ | Average block size |
|---:|---:|---:|---:|---:|
| 1225 | 45793 | 49 | 140 | 25.0 |
| 2125 | 94226 | 85 | 254 | 25.0 |
| 4000 | 165274 | 160 | 468 | 25.0 |
| 4696 | 227302 | 172 | 508 | 27.3 |
| 6168 | 327911 | 208 | 610 | 29.7 |
| 7624 | 467562 | 244 | 734 | 31.2 |
| 10511 | 729935 | 319 | 975 | 32.9 |
| 12946 | 971365 | 379 | 1158 | 34.2 |

- Test with a set of matrices from a DG problem
- Problems from a higher order DG method for Poisson's equation in 2D.

## ANALYSE time

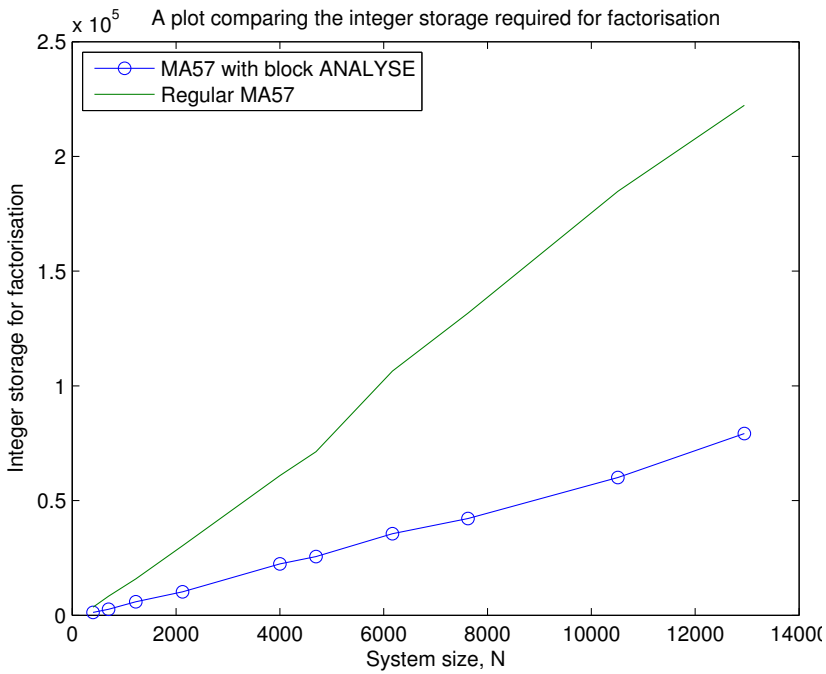



## ANALYSE, FACTORIZE and SOLVE time

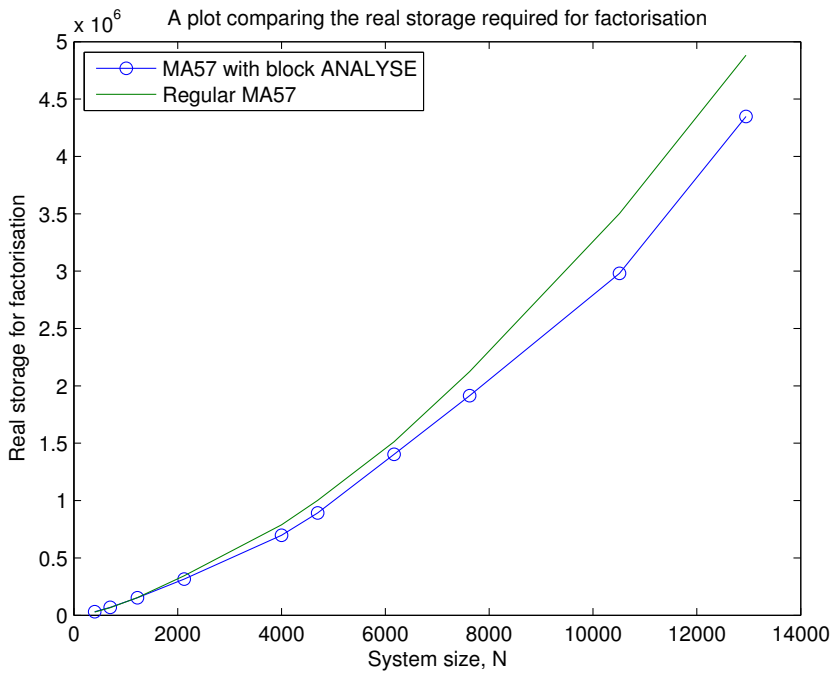



- Notice a speedup when replacing MA57, with MA57 with block ANALYSE
- Speedup for ANALYSE, FACTORIZE and SOLVE is much greater than speedup for ANALYSE

## Integer storage for matrix factorization

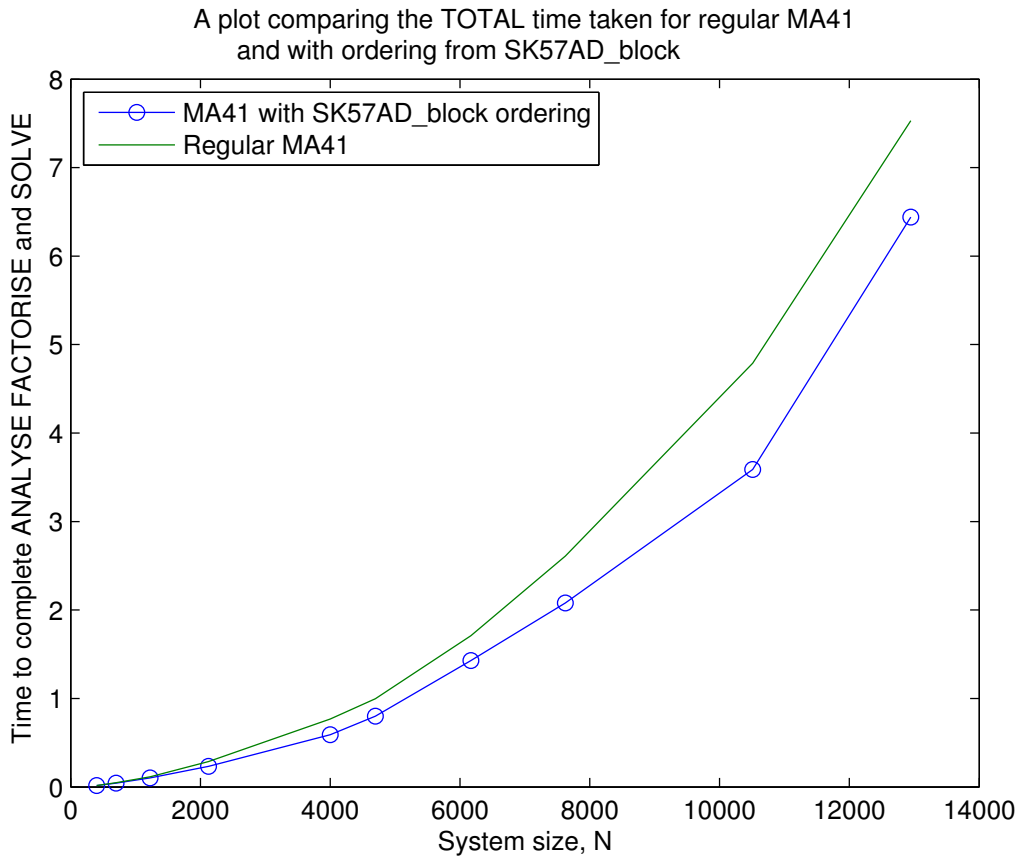



## Real storage for matrix factorisation




- Substantial reduction in the number of integers required to store the factorized matrix
- Moderate reduction in the real storage required

- 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

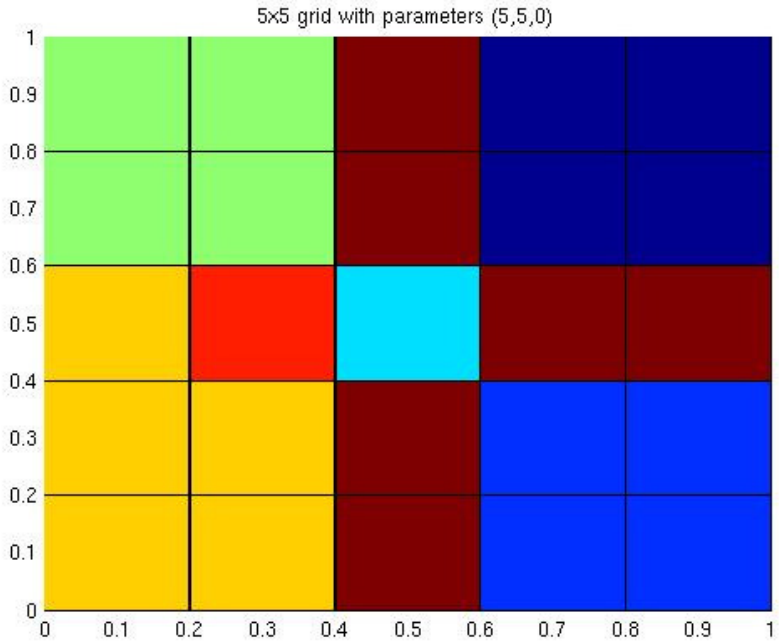
# Defining the domain decomposition

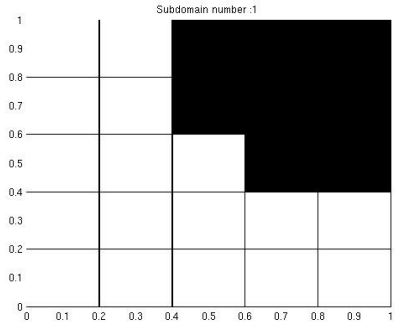
- Seek an overlapping partition $\{\Omega_i\}$ of the computational domain, $\Omega$:

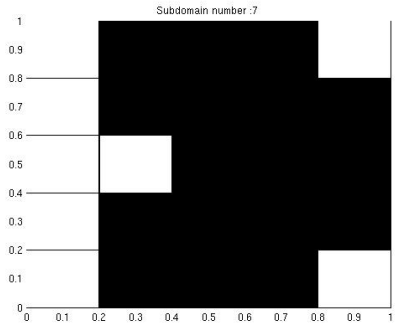
$$\bigcup_i \Omega_i = \Omega \quad , \quad \bigcap_i \Omega_i \neq \phi$$

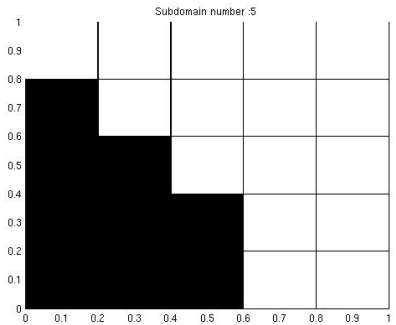
- Tree data from modified MA57 `ANALYSE` run on block structure can define this partition
- Parameter MERGE controls the sizes of the $\Omega_i$
- Parameter OVERLAP to controls the overlap between neighbouring $\Omega_i$
- Build preconditioner by building the reduced matrices $A_i$ corresponding the $\Omega_i$ and factorizing them

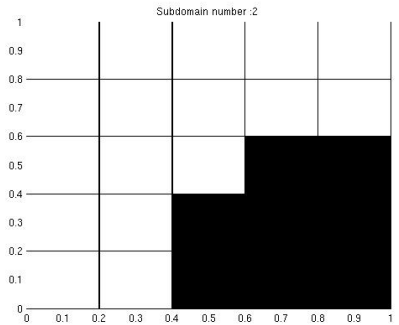
# An example of the effect of the overlap parameter



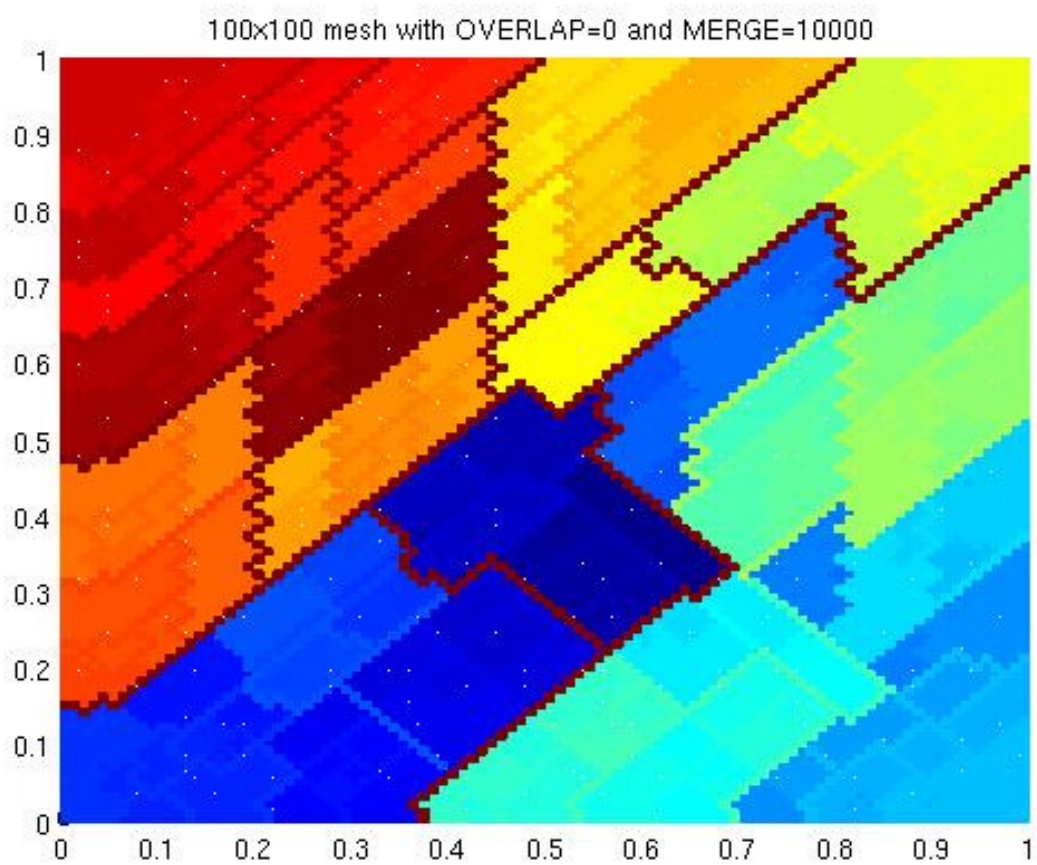



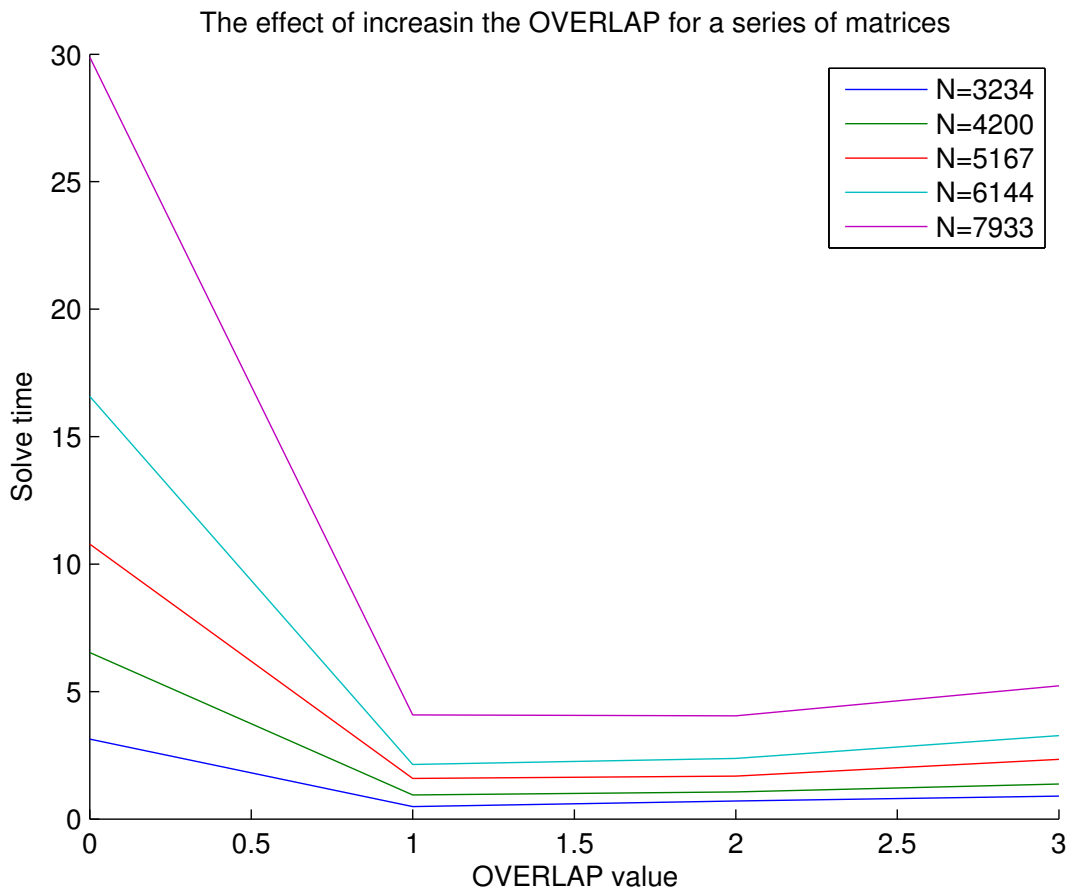



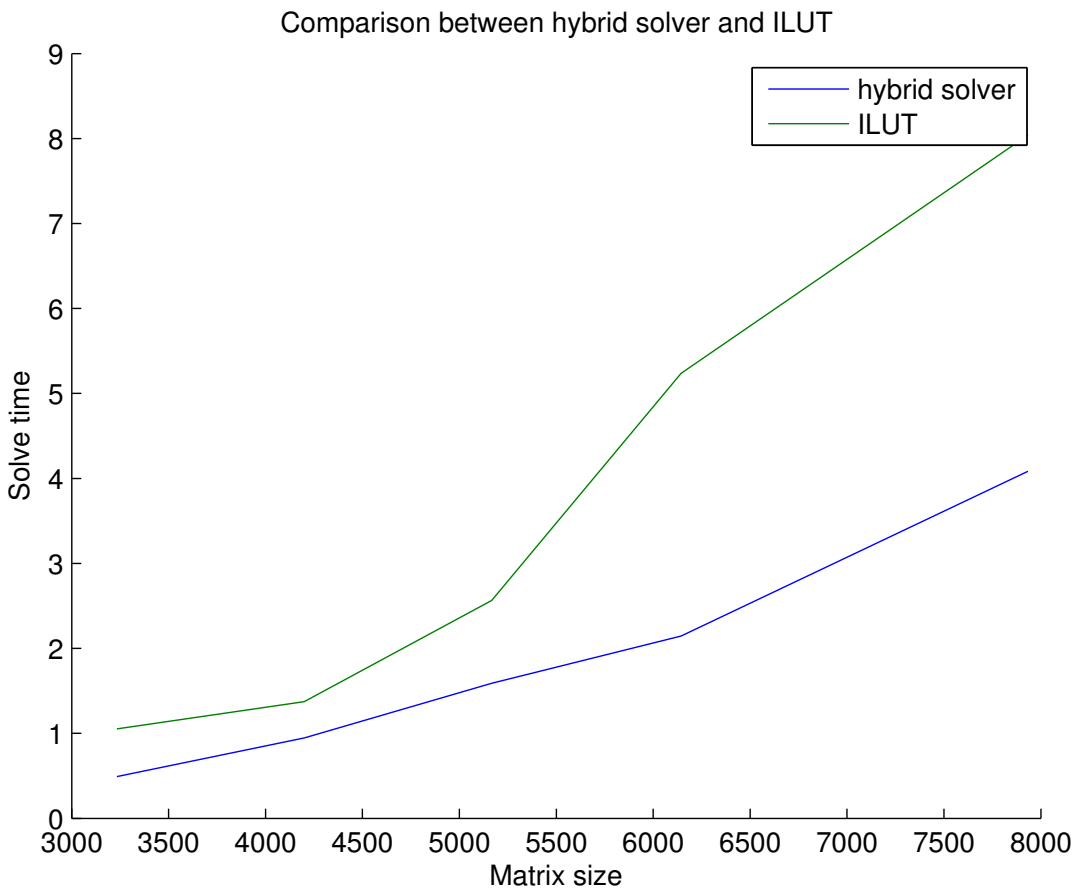



# A domain decomposition on a larger domain






- 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

$$\begin{aligned} \text{Matrix vector product} \quad & z \leftarrow Ax \\ \text{Precondition} \quad & z \leftarrow \sum_i R_i A_i^{-1} R_i^T x \\ \text{Inner product} \quad & z \leftarrow \langle x, x \rangle \end{aligned}$$

- 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

| Number Processes | OVERLAP | Greedy algorithm communication | | Metis communication | |
|---|---|---|---|---|---|
| | | Mat Vec | Precon | Mat Vec | Precon |
| 2 | 1 | 28696 | 28696 | 2262 | 2262 |
| 2 | 2 | 28696 | 34899 | 2262 | 4804 |
| 4 | 1 | 54934 | 54934 | 5646 | 5646 |
| 4 | 2 | 54934 | 94661 | 5646 | 12008 |
| 8 | 1 | 69328 | 69328 | 10886 | 10886 |
| 8 | 2 | 69328 | 153086 | 10886 | 24366 |

# 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?