A parallel ILU strategy for solving Navier-Stokes equations on an unstructured 3D mesh

Ø. Staff & S. Ø. Wille
Faculty of Engineering, Oslo University College, Norway

Abstract

An iterative algorithm for solving a mixed finite element formulation of Navier-Stokes equations on a distributed memory computer is presented. The solver is a Krylov subspace method with a parallel preconditioner suitable for high latency clusters. Nodes are pivoted to minimize the number of synchronization points in each solver iteration.

An unstructured mesh is decomposed into non-overlapping subdomains. Each node is given a category depending on which subdomains it is a member of and on the subdomains of its neighboring nodes in the mesh. Based on these categories, an *a priori* pivoting suited for parallel solution is constructed. The solver requires approximately the same number of iterations as good serial solvers with a similar preconditioner.

The incomplete LU (ILU) preconditioning and subsequent solve is performed on a global matrix implicitly formed as a sum of all subdomain matrices. Communication overhead is kept low by generating a schedule to send information to neighboring subdomains as soon as dependencies in the matrix are resolved. Results will be shown to indicate that this is a viable strategy on computer clusters built with cheap off the shelf components.
Keywords: ILU, preconditioning, parallel, unstructured mesh, CFD, Navier-Stokes.

1 Introduction

Simulations on single processors are often limited by CPU speed and available central memory. Even fairly modest three dimensional problems can surpass what

WIT Transactions on Engineering Sciences, Vol 52, © 2006 WIT Press
www.witpress.com, ISSN 1743-3533 (on-line)
doi:10.2495/AFM06004

can be solved on a single processor in a reasonable amount of time. Parallel multiprocessing is a frequently used strategy for overcoming such limits [1, 2, 3].

This work presents a parallel incomplete LU (ILU) factorization strategy developed for solving a stationary and incompressible formulation of the Navier-Stokes equations. The solver utilizes Taylor-Hood elements, *a priori* pivoting and segregation of variables. An unstructured mesh is split into a number of subdomains. Earlier work [4] required the subdomains be slices of the global domain with unconnected interfaces. A more general algorithm discussed her allows for arbitrary domain partitioning. This generalization is only partially motivated by a wish for increased efficiency by reducing the interface areas between subdomains. When scaling beyond a handful of subdomains, it becomes difficult to find a partitioning of an unstructured mesh with unconnected interfaces. Thus, a more robust algorithm is desirable.

2 Navier-Stokes equations

The model problem is the stationary Navier-Stokes system

$$\rho \mathbf{u} \cdot \nabla \mathbf{u} - \mu \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \qquad \text{in } \Omega \subset \mathbb{R}^3 \tag{1}$$

$$-\nabla \cdot \mathbf{u} = 0 \qquad \text{in } \Omega \tag{2}$$

with homogeneous Dirichlet boundary conditions

$$\mathbf{u} = \mathbf{0} \qquad \text{on } \Gamma = \partial\Omega \tag{3}$$

where $\mathbf{u}$ is the velocity vector, p is the pressure, ρ is the density and μ is the viscosity coefficient.

3 Parallel strategy

An unstructured mesh is decomposed into non-overlapping subdomains matching the number of available processors (figure 3). Each processor then performs an independent ILU on the degrees of freedom internal to the subdomain. A global ILU matrix is then implicitly created by synchronizing the degrees of freedom on interfaces between subdomains. All communication is done through message passing.

Unlike traditional domain decomposition techniques, no special regard is given to solving the interface degrees of freedom (e.g. by constructing a Schur complement). Instead, an implicit global matrix is created through node pivoting and the equivalent of a serial ILU of this matrix is performed (figure 2). The ILU fill-in strategy is to fill in at degrees of freedom connected through an element, and thus the ILU matrix will have the exact same structure as the finite element matrix.

The actual matrix constructed for each subdomain is pivoted to have three parts (figure 1). Velocity and pressure degrees of freedom are separated in each part, and each subdomain is given an arbitrary number. Then internal degrees of freedom

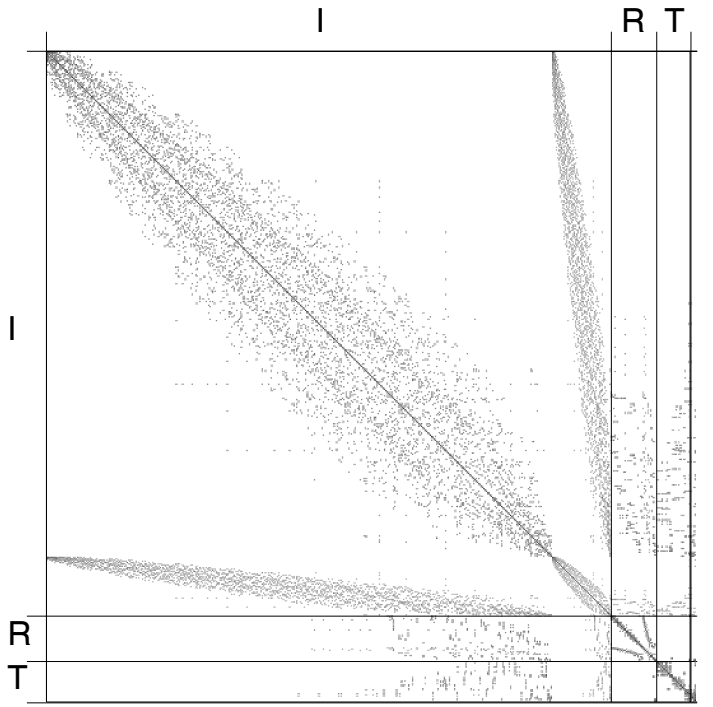

Figure 1: The structure of the solver matrix on a single processor. Each non-zero value in the matrix is marked with a dot. Nodes internal to the submesh are labeled I, those on receive interfaces are labeled R and transmit interfaces are labeled T. The submatrices connecting receive and transmit nodes are almost entirely zero. This allows much of the synchronization between subdomains to happen in parallel.

are pivoted first (I), followed by degrees of freedom on an interface to a lower numbered subdomain (R) and finally those on an interface to a higher numbered subdomain (T). Unless some element connects two interfaces, the submatrices R and T are guaranteed to be independent.

As an example, assume a mesh sliced into subdomains 1, 2 and 3 with subdomains 1 and 3 connected to the center slice 2 (figure 2). Performing a normal serial ILU on the resulting global matrix can then be performed in three steps:

1. Factorize A^1_{II}, A^2_{II} and A^3_{II}.
2. Send A^1_{TT} to subdomain 2 and add it to A^1_{RR}. Do the same with A^2_{TT} and A^3_{RR}.
3. Factorize A^2_{RR} and A^3_{RR}.

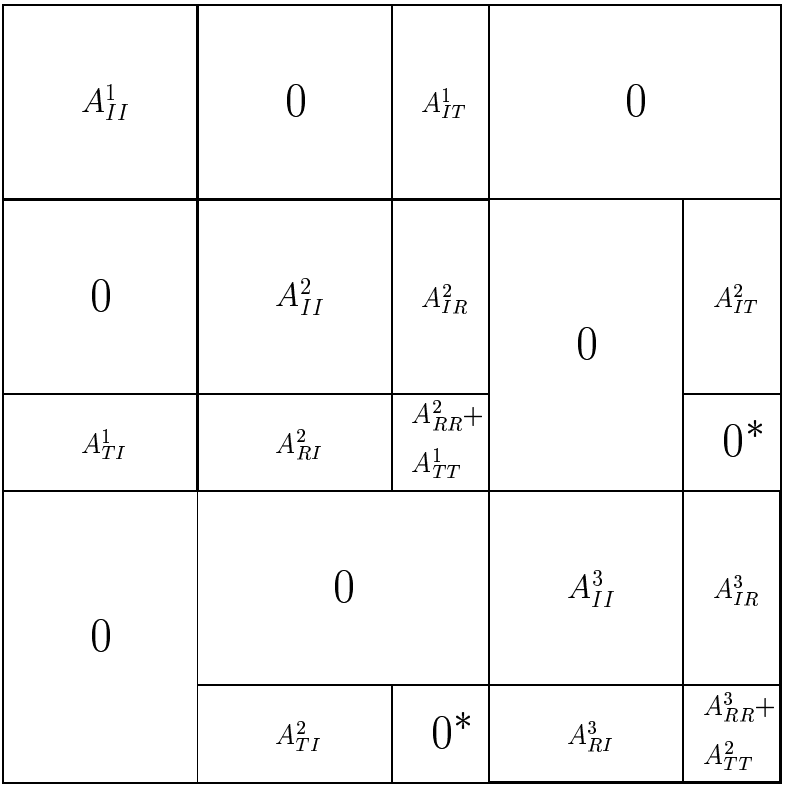

Figure 2: The implicitly generated matrix when running on three processors connected with interfaces between 1 and 2 and between 2 and 3. Running the parallel solver is equivalent to solving this matrix in a standard serial fashion. That is, the parallel solver realizes a serial ILU of this matrix. The starred zeros are zero only under the assumption that the two interfaces are independent and share no single element.

Given independent interfaces between subdomains this parallelization is always possible. However, in the degenerate case there will be no internal degrees of freedom and the parallel potential will be almost nonexistent.

The other complication stems from connected interfaces. Almost any partitioning will have some corners where interfaces meet, and although it will affect relatively few nodes, it is an important situation to handle.

With *a priori* knowledge of the matrix structure for any given node pivoting, lists of all possible dependencies in the ILU factorization are generated. The values of row i depends only on rows $j = [1, \ldots, i-1]$ where $a_{ij} \neq 0$. Consequently, when $k \leq i-1$ rows have been factorized and

$$a_{ij} = 0 \ \forall \ j \in [k, \ldots, i-1] \tag{4}$$

then row i can be factorized. Parallel ILU factorization is possible when $k < i-1$.

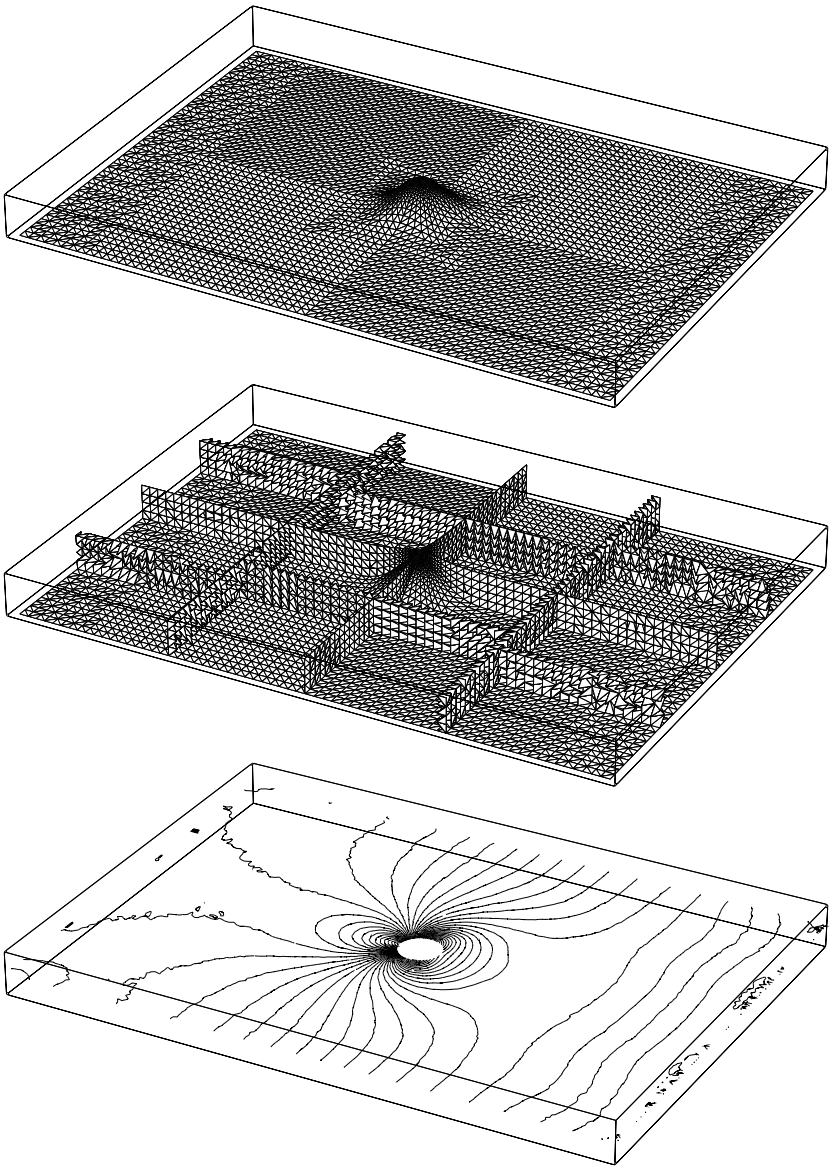

Figure 3: A mesh of a box with an elevated bottom. The middle figure is the mesh divided on 16 processors and the bottom shows pressure isobars of a steady state solution at Reynolds number 400. The flow is driven by a parabolic inflow at the right hand side, the sides and top have slip boundary conditions and the bottom has zero velocity. The walls in the middle mesh are the interface nodes between processors.

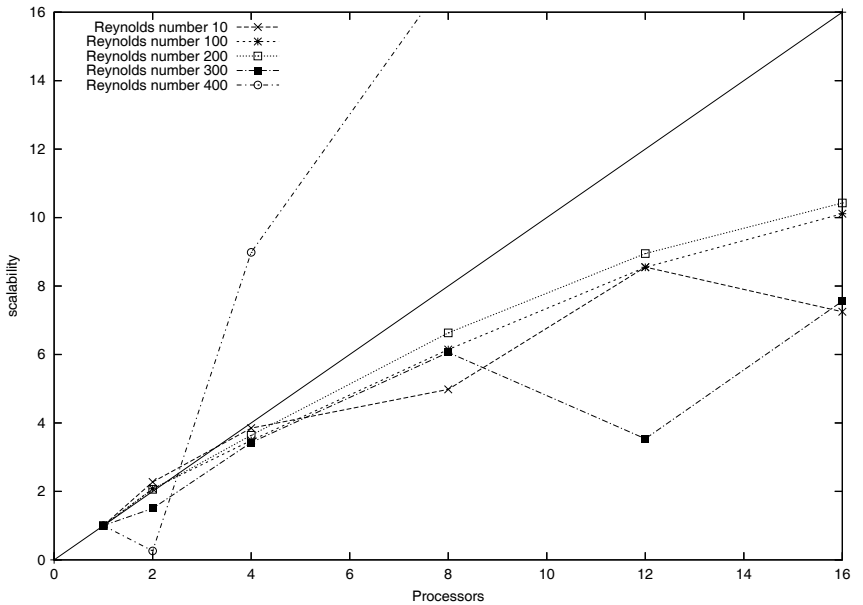

Figure 4: Scalability of the solver running on up to 16 CPUs. Runtime on 16 processors at Reynolds number 300 was 38.9 s.

The following node pivot strategy is designed to maximize the parallel potential between interfaces. For non-connected interfaces, it will produce a pivot identical to previous work [4].

1. For each node, build a list of which subdomains the node is a part of. Order the list in decreasing numerical order.
2. Sort all nodes in increasing order by lexically comparing subdomain lists. When subdomain lists are otherwise identical, the shortest list is considered smaller and ordered first. (The ordering is identical to the sorting of names in a phone book.)
3. For each node, find which other types of subdomain lists it is connected to by traversing the mesh. The node is dependent upon nodes with lexically smaller subdomain lists.
4. Sort nodes with identical subdomain lists in increasing order by their highest dependency.
5. Traverse the nodes in pivoted order. Whenever the node subdomain list changes, find the nodes which now have all their dependencies satisfied. These are guaranteed not to be affected by further factorization. From this construct a schema of transfer events to be used in the actual ILU factorization and CG iterations.

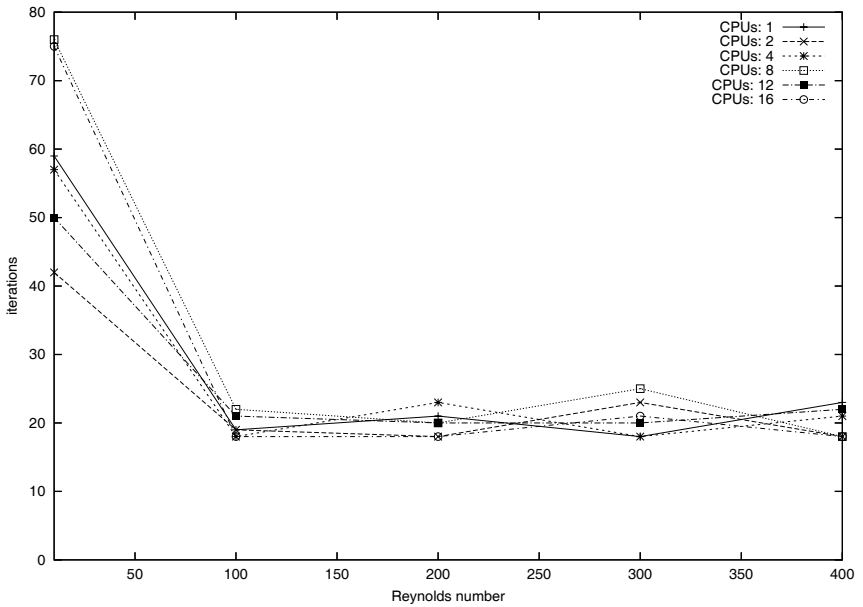

Figure 5: Required number of iterations to achieve $||Ax - b|| < 5.0 \cdot 10^{-6}$. Continuation is performed by scaling the previous solution when the velocity is increased. Each Reynolds number is calculated with five Newton iterations per refinement.

4 Results

Tests were performed on a Linux cluster with 16 3GHz Pentium 4 CPUs. The nodes were connected through a 100Mbit switched Ethernet and communication was performed using TCP/IP sockets.

The problem tested was a box with a bump in the middle (figure 3). The box had dimensions 7x5x1m, and the elevation of the bottom was a normal distribution with $\sigma = 0.5$. Flow at the inlet was set to a parabolic profile, the sides and top had slip boundary conditions, the bottom had no slip and the outlet had free velocity in the flow direction.

The initial mesh had 115,632 elements at Reynolds number 10. Subsequent meshes were adaptively grown to approximately 170,000 elements at Reynolds number 300 and 480,000 elements at Reynolds number 400. Scalability was around 10 for 16 CPUs (figure 4) and shows an increasing trend for the larger problems.

The final problem ran out of real memory on both one and two CPUs with corresponding abysmal single-CPU performance and super-linear scalability. In absolute numbers, the problem still performed well on 16 CPUs. From Reynolds

number 300 to 400, the 16 CPU mesh grew from 173,670 to 481,782 elements (factor 2.8). Runtime only increased from 38.9s to 87.8s (factor 2.3) clearly indicating that the relative parallel overhead decreases as the problem size is increased.

Each run on a given number of CPUs was performed independently. The initial difference this caused in the ILU pivot affected both iteration counts and small differences in the adaptive mesh. However, there is no obvious correlation between iteration count and the number of processors (figure 5). The quality of the preconditioner was not significantly impacted by the parallelization in the tests performed.

5 Conclusion

The presented parallel ILU preconditioner is suitable for implementation on cheap distributed memory, high latency clusters. Scalability is weak in the sense that per processor efficiency decreases as the number of processors is increased on any given problem. On the other hand, the speedup is quite good even on fairly small problems with runtimes less than a minute.

There is reason to believe the algorithm will scale to larger number of processors for constant per-processor element counts. Communication happens only between subdomains with shared nodes. Consequently, adding processors will not significantly impact the parallel overhead of any one processor.

References

[1] Bauer, A.C. & Patra, A.K., Performance of parallel preconditioners for adaptive hp fem discretization of incompressible flows. *Commun Numer Methods Eng*, **18**, pp. 305–313, 2002.

[2] Johnson, A. & Tezduyar, T., Methods for 3D computation of fluid-flow interactions in spatially periodic flows. *Computer Methods in Applied Mechanics and Engineering*, **190**, pp. 3201 – 3221, 2001.

[3] Gropp, W.D., Kaushik, D.K., Keyes, D.E. & Smith, B.F., Analyzing the parallel scalability of an implicit unstructured mesh cfd code. *Lect Note Comput Sci*, **1970**, pp. 395–404, 2001.

[4] Wille, S.Ø., Staff, Ø. & Loula, A., Block and full matrix ILU preconditioners for parallel finite element solvers. *Computer methods in applied mechanics and engineering*, **191(13-14)**, pp. 1381 – 1394, 2002.