Abstract

A class of parallel multiple front solution algorithms is described for solving linear systems arising from discretization of boundary value and evolution problems. The basic substructuring approach and frontal algorithm on each subdomain are first modified to ensure stable factorization in situations where ill-conditioning may occur due to differing material properties or the use of high-degree finite elements (p methods). Next, the method is extended to apply to parallel distributed-memory multiprocessor systems with the final reduced (small) Schur's complement problem solved on a single processor. A modified algorithm that implements a recursive partitioning approach on the subdomain interfaces is then developed. Both algorithms are implemented and tested in a least-squares finite element scheme for viscous incompressible flow computation using a p-finite element method.

Contents

List of figures **iii**

List of tables **iv**

1 Introduction **1**

2 Parallel multiple front algorithm **3**

2.1 Substructuring 3

2.2 Subdomain problem 4

3 Frontal implementation **9**

4 Partitioning the interface **11**

5 Matrix representation **16**

6 Data structure and implementation **18**

7 Numerical experiments **19**

8 Conclusion **29**

Acknowledgments **31**

A Hierarchic basis functions **31**

References **33**

List of Figures

4.1 Partitioning of a finite element mesh to six subdomains. 12

4.2 Separators in the graph of global interfaces. 13

4.3 Vertices of interface graphs for subdomains $1, 2$, and 4. 13

7.1 CPU Time for Mesh I and III with quadratic basis functions. 22

7.2 CPU Time for Mesh I ($p = 2$ and 7). 22

7.3 Effect of increasing polynomial orders (Mesh I) on 8 and 16 Processors. 23

7.4 Frontal solver performance for Mesh I ($p = 5$, 17820 equations). . . . 23

7.5 6-Subdomain partition of the 4:1 contraction geometry. 25

7.6 15-Subdomain partition of the 4:1 contraction geometry. 26

7.7 CPU time for different polynomial orders. 27

7.8 Performance of distributed Schur complement solution (p=6, 4:1 contraction). 27

7.9 Serial and parallel Schur complement solution timings (p=6, 4:1 contraction). 28

7.10 Performance of parallel Schur complement algorithm (p=6, $4:1$ Contraction). 29

7.11 31-Subdomain partition of the lid-driven cavity problem. 30

7.12 Performance of parallel Schur complement algorithm (p=5, lid-driven cavity). 30

A.1 2D hierarchic basis functions. 32

List of Tables

7.1 Global Schur's complement and subdomain unknowns for Meshes I and III (p=2). 22

7.2 Schur's complement problem size for Mesh I on eight processors (Lagrange). . . 24

7.3 Schur's complement problem size for Mesh I on eight processors (Legendre). . . 24

1 Introduction

The emergence of several powerful multiprocessor supercomputers and uniform message passing interface protocols for tightly coupled workstation clusters has generated considerable interest in the development and application of parallel algorithms. In analysis applications using finite-difference or finite-element schemes, much of the computational cost is associated with solving sparse linear systems. Domain decomposition has proven to be an effective and robust strategy to exploit parallelism across subdomains, and this is the approach considered in the present work.

Since iterative methods such as the conjugate gradient (CG) algorithm are generally easier to parallelize and scale than sparse direct methods, considerable attention has been focused on the performance of iterative methods on distributed memory multiprocessors. Most of these studies are limited to solving well-behaved, sparse, diagonally dominant, symmetric positive-definite linear systems arising from low-degree finite elements or analogous finite-volume/difference schemes. Relatively little attention has been paid to solving poorly conditioned linear systems with an irregular sparsity pattern that may arise out of high-degree p-version finite-element formulations, nor has the parallel preconditioning problem been fully resolved for these systems. For example, Babuška *et al.* (1992) consider a domain decomposition strategy for an hp-finite element formulation applied to elliptic problems and using a preconditioned CG method for shared-memory parallel processors. Barragy *et al.* (1994) present a parallel element-by-element scheme for elliptic partial differential equations and discuss some aspects of the preconditioning and iterative performance issues for these schemes in Carey and Barragy (1989) and Barragy and Carey (1991). A family of domain decomposition preconditioners has also been proposed by Mandel (1990) and Mandel (1991) for p-version finite-element matrices. However, there are many classes of problems for which iterative methods converge slowly, are unreliable, or breakdown and parallel elimination methods are more reliable and efficient. In fact, for real applications in many areas, iterative methods may have significant difficulties. Notably, this is the case in least-squares finite element formulations of the incompressible viscous flow equations for Newtonian and viscoelastic flows and motivates the present research on parallel direct methods using substructuring and element-by-element multifrontal techniques.

Almost all studies on parallel direct solution of hp-finite element systems consider symmetric positive definite matrices for which diagonal pivoting and distributive Cholesky decomposition can be employed. For example, Guo and Myers (1996) proposed a dynamic load distribution algorithm for mapping hp-finite element clusters onto processors using the mesh topology and a parallel distributed Cholesky factorization scheme for solving the resulting linear system. The algorithm is suitable

for parallel direct solution of linear elasticity problems.

In the present paper, we extend the parallel multiple front approach (DUFF AND SCOTT, 1994) to the solution of linear systems arising from p-finite element formulations and apply the algorithm to viscous flow simulations. We have added several intermediate steps for subdomain and element reordering, and load balancing via weighted vertices based on the p-levels of the element basis functions to the existing capabilities of Metis (KARYPIS AND KUMAR, 1995) to achieve our subdomain partitioning for parallel elimination. An element-by-element frontal LU factorization with threshold pivoting is then applied to element matrices of each subdomain. Performing the LU factorization step allows us to solve non-symmetric matrix problems such as those arising from the mixed Galerkin formulation of the Navier-Stokes equations. In general, this step eliminates most of the internal unknowns in each subdomain. However, it should be noted that, because of the pivoting criterion used in our scheme, some of the internal unknowns in each subdomain may not be eliminated and therefore become part of the resulting Schur's complement problem. As explained later, this is necessary for stability of the factorization for systems such as those obtained using high p bases and mixed methods for incompressible viscous flow computations. The global Schur's complement problem can be assembled and solved on one processor. The respective subdomain solutions are then extracted and sent to the individual processors for parallel recovery of the local eliminated variables. However, solution of the entire Schur's complement problem on one processor presumes that the reduced problem is small and hence, only allows coarse granularity of the present algorithm. In a modified algorithm we introduce recursive partitioning of the interface and global Schur's complement problem. This leads to improved scalability properties, as shown in the supporting numerical experiments.

The outline of the discussion is as follows: We first describe the basic domain decomposition and substructuring problem from a linear algebra standpoint. Then we give details of the frontal implementation, including strategies related to pivoting and reduced fill-in. Our approach for recursively partitioning the interface and the associated graph theory is then presented. Some supporting numerical experiments for p-finite element solution of viscous flow problems and parallel performance studies conclude the treatment.

2 Parallel multiple front algorithm

2.1 Substructuring

Consider a mesh defined on a domain Ω and a partition to a set of k open subdomains $\Omega = \bigcup_{i=1}^{k} \Omega_i$ with associated subdomain meshes such that $\Omega_i \bigcap \Omega_j = 0$ and the common interfaces are $\overline{\Omega_i} \bigcap \overline{\Omega_j} = \Gamma_{ij}$ for $1 \leq i, j \leq k$. Using this decomposition, the discretized problem can be expressed as a partitioned algebraic system with the partitioning defined by the subdomains.

A typical substructuring algorithm involves factorization of matrix blocks for grid points interior to the respective subdomains in order to eliminate internal unknowns in terms of interface unknowns. This is straightforward when the subdomain block matrices are nonsingular. However, Gunzburger and Nicolaides (1984) have shown, for instance, that in the standard mixed Galerkin formulation for the incompressible Navier-Stokes problem with discontinuous pressure subspaces, each subdomain block matrix will have a single local pressure null vector and therefore will be singular with a one-dimensional null space. To circumvent this difficulty, they propose a complex algorithm using pseudo-inverses and many matrix manipulation steps.

In other cases, such as the least-squares mixed formulation of the incompressible Navier-Stokes equations with equal interpolation bases for pressure, velocity, and stresses, the subdomain block matrices are nonsingular. However, depending on the choice of element basis functions and the particular problem, these block matrices may not be completely factorizable because of ill-conditioning that can be attributed to the presence of very small pivots arising from the basis functions. We propose a strategy that is based on frontal elimination with pivoting within each subdomain and that may involve only a partial elimination.

To illustrate the main ideas, consider decomposition of a domain Ω to two subdomains Ω_1 and Ω_2. Let us regroup implicitly the unknowns so that $\mathbf{u_1}$ and $\mathbf{u_2}$ correspond to only those internal unknowns that can be eliminated from subdomains Ω_1 and Ω_2, respectively, within stability limits and prescribed pivoting criteria. Then let $\mathbf{u_S}$ denote the interface unknowns and remaining internal unknowns from the two subdomains. We write the resulting partitioned system as

$$\begin{bmatrix} \mathbf{J_{11}} & \mathbf{0} & \mathbf{J_{1S}} \\ \mathbf{0} & \mathbf{J_{22}} & \mathbf{J_{2S}} \\ \mathbf{J_{S1}} & \mathbf{J_{S2}} & \mathbf{J_{SS}} \end{bmatrix} \begin{bmatrix} \mathbf{u_1} \\ \mathbf{u_2} \\ \mathbf{u_S} \end{bmatrix} = \begin{bmatrix} \mathbf{r_1} \\ \mathbf{r_2} \\ \mathbf{r_S} \end{bmatrix}, \tag{2.1}$$

where the coefficient matrix corresponds to, say, the Jacobian matrix for a nonlinear

system solve or the constant coefficient matrix for a linear system. Note that the specific form of $\mathbf{u_S}$ and hence the structure of the reduced Schur's complement problem for $\mathbf{u_S}$ cannot be determined in advance but is only known after the partial interior elimination is complete in Ω_1 and Ω_2.

2.2 Subdomain problem

Let us now consider the part of the algebraic linear system corresponding to a subdomain Ω_s written as

$$\mathbf{Ju} = \mathbf{r} + \boldsymbol{\sigma}. \tag{2.2}$$

Here $\mathbf{J}$ is the assembled Jacobian matrix from individual elemental matrices of the subdomain, $\mathbf{r}$ is likewise the assembled residual for subdomain Ω_i, and $\boldsymbol{\sigma}$ corresponds to the remaining contributions from the adjacent subdomains. (We have suppressed the subdomain index for notational simplicity.)

Next, let us introduce another local ordering of the subdomain nodal unknowns so that those eliminated by the action of the incomplete frontal factorization are numbered first as $\mathbf{u_e}$. The remaining unknowns, $\mathbf{u_f}$, correspond to those that are left in the front for the subdomain. This simple reordering of the nodal components can be represented using a permutation matrix $\mathbf{Q^T}$ by

$$\tilde{\mathbf{u}} = \mathbf{Q^T u}, \tag{2.3}$$

where $\tilde{\mathbf{u}} = [\mathbf{u_e} \;\; \mathbf{u_f}]$ and $\mathbf{Q^T}$ is defined by elementary operations on the identity matrix.

In the same way, the components of the vector $\mathbf{r}$ in (2.2) can be reordered to $\tilde{\mathbf{r}}$ with

$$\tilde{\mathbf{r}} = \mathbf{P^T r}, \tag{2.4}$$

as in (2.3), where $\mathbf{P^T}$ is defined in a manner similar to $\mathbf{Q^T}$ to denote the effect of row interchanges. It should be noted that the permutation matrices $\mathbf{P}$ and $\mathbf{Q}$ are the same only if diagonal pivoting is used for factorization of $\mathbf{J}$. Otherwise, matrices $\mathbf{P}$ and $\mathbf{Q}$ reflect the order of row and column pivoting, respectively, during the frontal factorization of $\mathbf{J}$. Under this reordering, equation (2.2) becomes

$$\tilde{\mathbf{J}}\tilde{\mathbf{u}} = \tilde{\mathbf{r}} + \tilde{\boldsymbol{\sigma}}, \tag{2.5}$$

where $\tilde{\boldsymbol{\sigma}}^T$ has zeros in those entries corresponding to all interior degrees of freedom and

$$\tilde{\mathbf{J}} = \mathbf{P^T J Q}. \tag{2.6}$$

When diagonal pivoting is used, that is, $\mathbf{P^T} = \mathbf{Q}$, the symmetry of $\tilde{\mathbf{J}}$ is preserved. The elimination process induces a corresponding block partitioning which can be expressed conveniently as

$$\tilde{\mathbf{J}} = \begin{bmatrix} \mathbf{J_{ee}} & \mathbf{J_{ef}} \\ \mathbf{J_{fe}} & \mathbf{J_{ff}} \end{bmatrix}. \tag{2.7}$$

Here $\mathbf{J_{ee}}$ denotes the block of $\mathbf{J}$ associated with the nodal unknowns to be eliminated, and $\mathbf{J_{ff}}$ is associated with those in the remaining front. The off-diagonal blocks $\mathbf{J_{ef}}$ and $\mathbf{J_{fe}}$ describe the interaction between eliminated and remaining degrees of freedom.

The system (2) under this partitioning becomes

$$\begin{aligned} \mathbf{J_{ee}u_e} + \mathbf{J_{ef}u_f} &= \mathbf{r_e} \\ \mathbf{J_{fe}u_e} + \mathbf{J_{ff}u_f} &= \mathbf{r_f} + \boldsymbol{\sigma}_\mathbf{f}, \end{aligned} \tag{2.8}$$

where the nonzero entries in $\boldsymbol{\sigma}_f$ correspond to all unknowns on the interface.

Formally, the first equation in (2.8) can be used to express $\mathbf{u_e}$ in terms of $\mathbf{u_f}$ as

$$\mathbf{u_e} = \mathbf{J_{ee}}^{-1}\left[\mathbf{r_e} - \mathbf{J_{ef}u_f}\right] \tag{2.9}$$

so that the remaining equation in (2.8) reduces to the Schur's complement problem for the subdomain interface,

$$\mathbf{S_{ff}u_f} = \mathbf{b_f}, \tag{2.10}$$

with the Schur's complement matrix $\mathbf{S_{ff}}$ defined by

$$\mathbf{S_{ff}} = \mathbf{J_{ff}} - \mathbf{J_{fe}J_{ee}^{-1}J_{ef}} \tag{2.11}$$

and

$$\mathbf{b_f} = \mathbf{r_f} + \boldsymbol{\sigma}_\mathbf{f} - \mathbf{J_{fe}J_{ee}^{-1}r_e}. \tag{2.12}$$

Of course, this system cannot be set up explicitly because the unknowns defining $\mathbf{u_f}$ are not known a priori. Moreover, even if the system were set up, it obviously could not be solved because entries in $\boldsymbol{\sigma}_\mathbf{f}$ are not known. However, the operations in (10) and (11) can be conveniently expressed for our purposes in terms of the block factorization

$$\begin{bmatrix} \mathbf{J_{ee}} & \mathbf{J_{ef}} \\ \mathbf{J_{fe}} & \mathbf{J_{ff}} \end{bmatrix} = \begin{bmatrix} \mathbf{L_{ee}} & \mathbf{0} \\ \mathbf{L_{fe}} & \mathbf{S_{ff}} \end{bmatrix} \begin{bmatrix} \mathbf{U_{ee}} & \mathbf{U_{ef}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{2.13}$$

Further details of the LU factorization and matrix front modification as well as pivoting and fill-in reduction strategies are described later in the section on Frontal Implementation. Substituting the factored form, (2.13), in (2.8),the resulting problem can be rewritten as the pair of systems

$$\begin{bmatrix} \mathbf{L_{ee}} & \mathbf{0} \\ \mathbf{L_{fe}} & \mathbf{S_{ff}} \end{bmatrix} \begin{bmatrix} \mathbf{y_e} \\ \mathbf{y_f} \end{bmatrix} = \begin{bmatrix} \mathbf{r_e} \\ \mathbf{r_f} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\sigma}_\mathbf{f} \end{bmatrix}. \tag{2.14}$$

and

$$\begin{bmatrix} \mathbf{U_{ee}} & \mathbf{U_{ef}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{u_e} \\ \mathbf{u_f} \end{bmatrix} = \begin{bmatrix} \mathbf{y_e} \\ \mathbf{y_f} \end{bmatrix} \tag{2.15}$$

Let us now return briefly to the global problem and note that the global system could be obtained by assembling the local subdomain systems. Let $\mathbf{B_s}$ denote the boolean matrix specifying the global connectivity for subdomain s. Then the local subdomain matrix $\mathbf{J_s}$ can be mapped into the global system by $\mathbf{B_s^T J_s B_s}$. Here $\mathbf{J_s}$ is of size $n_s \times n_s$ and $\mathbf{B_s}$ is an $n_s \times n_G$ matrix, where n_s, n_G are the number of local and global degrees of freedom, respectively. Similarly, $\mathbf{r_s}$, $\boldsymbol{\sigma}_\mathbf{s}$ map to $\mathbf{B_s^T r_s}$ and $\mathbf{B_s^T}\boldsymbol{\sigma}_\mathbf{s}$. Then the assembled global system for $\mathbf{u_G}$ can be expressed as the sum of mapped subdomain contributions

$$\sum_{s=1}^{S} \left(\mathbf{B_s^T J_s B_s}\right) \mathbf{u_G} = \sum_{s=1}^{S} \mathbf{B_s^T r_s} + \sum_{s=1}^{S} \mathbf{B_s^T}\boldsymbol{\sigma}_\mathbf{s}. \tag{2.16}$$

The last term in (2.16) reduces to the global sum of interface flux jumps; that is,

$$\sum_{s=1}^{S} \mathbf{B_s^T}\boldsymbol{\sigma}_\mathbf{s} = \sum_{i=1}^{I} [\boldsymbol{\sigma}_\mathbf{i}] = \mathbf{0}, \tag{2.17}$$

which follows from the weak variational statement in the Galerkin method and from $\mathbf{C^o}$ approximation of the fluxes in the least-squares method.

Hence, we need not explicitly treat $\boldsymbol{\sigma}_\mathbf{f}$ and therefore can proceed with a parallel factorization step as follows: The interface variables for each subdomain are first flagged in a pre-front routine. The block matrices $\mathbf{L_{ee}}$, $\mathbf{L_{fe}}$,$\mathbf{U_{ee}}$,$\mathbf{U_{ef}}$, and $\mathbf{S}$ are then generated by an element-by-element frontal solution scheme with threshold pivoting within the front. The remaining front contains not only the interface variables but also those remaining internal variables that did not pass the threshold pivoting criterion. Since conditioning depends upon the choice of the element basis functions (as well as other factors such as subdomain mesh size), it is not surprising that different bases will lead to different front sizes. In some of our numerical test cases we find that a

significant fraction of the interior unknowns still remain in the front at the end of the factorization step.

For clarity of exposition, let us first assume that the global Schur's complement problem ($\mathbf{S_F}\ \mathbf{g_F} = \mathbf{b_F}$) has been assembled from subdomain contributions, (2.10), onto processor zero and solved to determine the global vector $\mathbf{g_F}$. From $\mathbf{g_F}$ we can extract the subdomain front solution vectors $\{\mathbf{u_f^s}\}, s = 1, 2, \cdots, S$. Consider a typical subdomain with front vector solution $\mathbf{u_f}$. Then the recovery procedure (recall eq. (2.9)) for the remaining interior variables consists of a subdomain back-substitution step in (2.15).

For a given LU-decomposition of the subdomain Jacobian matrix $\tilde{\mathbf{J}}$, we can modify (2.12) as

$$\mathbf{b_f} = \mathbf{r_f} + \boldsymbol{\sigma}_\mathbf{f} - \mathbf{J_{fe}}\mathbf{J_{ee}^{-1}}\mathbf{r_e} = \mathbf{r_f} + \boldsymbol{\sigma}_\mathbf{f} - \mathbf{L_{fe}}\mathbf{y_e} \tag{2.18}$$

The above step replaces the matrix multiplication operations to a forward substitution step for calculation of the Schur's complement vector $\mathbf{b_f}$. Since the local subdomain elimination is local, the above steps can be made concurrently across the processors in a distributed environment. For purposes of overlapping computation with communication, the subdomain Schur's complement matrix $\mathbf{S_{ff}}$ may be sent to processor zero via a nonblocking "send" call while the computation for vector $\mathbf{b_f}$ is in progress. A more detailed description of the parallel multiple front solution algorithm follows as

Algorithm [parallel multiple front algorithm]

for s = 0 **to** S subdomains in parallel
if s = 0 **then** [processor **zero**]
  Receive $\mathbf{S_{ff}}$ and $\mathbf{b_f}$ from processors {1,2,$\cdots$,S} and assemble
    the global Schur's complement problem.
  Solve the global assembled Schur's complement system.
  Extract and send subdomain solution $\mathbf{y_e}$ to each subdomain.
else
  Construct matrices $\mathbf{L_{ee}}$,$\mathbf{L_{fe}}$,$\mathbf{U_{ee}}$,$\mathbf{U_{ef}}$,$\mathbf{S_{ff}}$ using subdomain frontal factorization.
  Generate permutation matrices $\mathbf{P}$ and $\mathbf{Q}$ from row and column pivoting sequences
    during frontal factorization.
  Send subdomain Schur's complement matrix $\mathbf{S_{ff}}$ to processor **zero** using a
    nonblocking data transfer protocol such as **MPI_Isend**.
  Compute $\mathbf{b_f} \leftarrow \mathbf{r_f} - \mathbf{L_{fe}}\mathbf{y_e}$ using forward substitution.
  Send subdomain Schur's complement right-hand-side $\mathbf{b_f}$ to processor
    zero using **MPI_Isend**.
  Receive $\mathbf{y_e}$ from processor **zero** using a **blocking** receive call

such as MPI_recv. [forces synchronization with global Schur's complement solution step]

Solve for internal variables $\mathbf{u_e}$.

endif

endfor

Remarks

The only serial part of the algorithm is the final solution of the global reduced Schur's complement problem. Clearly, this step may dramatically degrade performance and efficiency of the parallel algorithm as the problem scales since increasing the number of subdomains results in a large number of interface variables (DUFF AND SCOTT, 1994); and the size of the assembled Schur's complement problem increases accordingly (KEUNINGS, 1995). This algorithm is therefore best suited for coarse-grain parallelism. Of course, one can also solve the resulting global Schur's complement problem in parallel using a parallel dense direct or iterative solution algorithm.

The above comments are especially true for elements with low order polynomial bases such as linear and quadratic basis functions. For example, given a fixed mesh of quadratic finite elements, the ratio of interface and subdomain internal unknowns quickly grows with an increase in the number of partitions. However, with sufficiently high order finite elements on the same mesh, there are more internal unknowns in any subdomain than interface unknowns in the global Schur's complement problem. Therefore, the parallel efficiency can be improved significantly by using high-order hierarchic polynomial basis functions keeping the mesh fixed. We briefly describe the hierarchic polynomial bases used in the present study in the Appendix. This is also demonstrated in our numerical experiments presented in a later section. However, it should be noted that fixing the mesh size to a constant and increasing the polynomial degree of the approximation functions may not always lead to an optimal rate of convergence. On the other hand, the above algorithm may be quite practical on a tightly coupled system of workstations. To illustrate these ideas, let us consider a rectangular $m \times n$ mesh of quadrilaterals. Let p denote the uniform polynomial degree of the approximation functions in all the elements of the mesh and k the number of subdomain partitions. Note that there are many possible ways of dividing a mesh of $m \times n$ elements into k-subdomain partitions. For example, a simple strategy is to use vertical partitioning of the elements into k vertical subdomain strips. With such an arrangement, we can readily calculate the number of internal and interface unknowns for each subdomain and also the number of unknowns for the global Schur's complement problem, assuming that only the subdomain interface unknowns constitute

the global Schur's complement system. The present parallel multiple front algorithm based on a serial solution of the global Schur's complement problem is acceptable when the size of the global Schur's complement problem is less than the average subdomain problem size. Algebraically this can be written for the above partitioning as

$$k \leq \frac{\sqrt{1+4b}+1}{2}, \tag{2.19}$$

where $b = \frac{p^2mn+mn+np-1}{mp+1}$. Such estimates, however, cannot be as easily calculated for more general unstructured mesh and subdomain partitions. The Schur's complement problem is more dense than the typical subdomain problem. Obviously the overall efficiency will be better if the Schur's complement problem is significantly smaller than the average subdomain problem.

3 Frontal implementation

The convenient sparse block structure of the subdomain matrix $\tilde{\mathbf{J}}$ in (2.5) does not arise naturally. Instead, the matrix is stored as a collection of smaller dense finite-element matrices so that their assembling sum is equivalent to a permutation of the matrix displayed on the left side of (2.5). Nevertheless, the block LU factorization of $\tilde{\mathbf{J}}$ can be obtained without ever using a block structure. The finite element matrix contributions can be assembled into a larger dense matrix, denoted $\mathbf{F}$, where all dense kernel factorization operations take place. The so-called frontal matrix $\mathbf{F}$ is constantly modified at each step of the factorization process with a rank-one update operation such that the frontal matrix at the k-th step is obtained by

$$\mathbf{F}^{(k)} = \mathbf{F}^{(k-1)} - \alpha^{(k)}\mathbf{x}\mathbf{y}, \tag{3.1}$$

where $\alpha^{(k)}$ is the inverse of the pivot element in the matrix, $\mathbf{x}$ is the vector of multipliers, and $\mathbf{y}$ is the row to be subtracted. $\mathbf{F^{(k)}}$ is organized into four dense blocks

$$\mathbf{F}^{(k)} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}. \tag{3.2}$$

The blocks $\mathbf{A}$ and $\mathbf{C}$ store fully summed columns, and blocks $\mathbf{A}$ and $\mathbf{B}$ store fully summed rows. Pivots can be chosen from row and column entries in $\mathbf{A}$ according to the criteria discussed below. We proceed with rank-one updates until no further equation can be eliminated or the dimension of $\mathbf{D}$ equals the number of interface equations (in which case, $\mathbf{A}$ will be null). The remaining block $\mathbf{D}$ is the Schur

complement of $\mathbf{J_{ee}}$ in $\tilde{\mathbf{J}}$, and it is elsewhere denoted as $\mathbf{S_{ff}}$. The block factorization (2.13) is possible since the columns of $[\mathbf{L_{ee}}\ \ \mathbf{L_{fe}}]^T$ are the multipliers $\alpha^{(k)}\mathbf{x}$ and the rows of $[\mathbf{U_{ee}}\ \ \mathbf{U_{ef}}]$ are the subtracting rows $\mathbf{y}^T$.

It should be noted that the dimension of $\mathbf{D}$ is equal to or greater than the number of subdomain interface equations. It will be greater if all finite elements are assembled into $\mathbf{F}$; $\mathbf{A}$ cannot be reduced to a null matrix because some of its entries have unacceptable pivots. Since pivots can be chosen anywhere in $\mathbf{A}$, row and column permutations are actively performed, and therefore a rigorous account of the above procedure must include the effects of two permutation matrices.

The criteria for selection of pivots follows ZLATEV (1980), except that we have not found advantages in using an associated fill-in reducing strategy. A pivot at the k-th step of factorization is selected in $\mathbf{A}$ by searching the rows of $\mathbf{A}$ and $\mathbf{C}$ for each column of $\mathbf{A}$ in the increasing order of their global indices until the first occurrence of the threshold criterion,

$$|a_{i,j}| \geq u\left(\max|a_{i,j}|, \max|c_{i,j}|\right), \tag{3.3}$$

is satisfied. A global index of rows and columns of $\mathbf{F}$ points to the corresponding sparse row and column indices in $\tilde{\mathbf{J}}$. Therefore the pivot search is biased towards the original ordering of the equations. The threshold parameter u is chosen as 0.1. No absolute pivot tolerance is enforced, but the largest and smallest pivot absolute value is monitored. In our experience, if $\tilde{\mathbf{J}}$ is properly equilibrated, an acceptable pivot can be found in a few column searches, typically less than 5. Equilibration is found to be mandatory—otherwise this pivoting strategy can take a substantial amount of computing time to search and most importantly cause prohibitive fill-in. See (DE ALMEIDA *et al.*, 1999) for details on equilibration and fill-in control strategies based on dropping.

The entries in the multiplier column $\alpha^{(k)}\mathbf{x}$ and subtracting row $\mathbf{y}$ are dropped according to a threshold dropping criterion. The root-mean-square value (RMS) of each row and column is computed, and all entries whose absolute value is less than 10^{-7} times the RMS value are dropped. We find that this dropping strategy often reduces fill-in without affecting the quadratic convergence of the outer Newton iteration loop. Ordering of the matrix $\tilde{\mathbf{J}}$ is of the utmost importance. Here we employ the wavefront reducing method of SLOAN (1989) modified by Scott's (1995) procedure II to avoid subdomain interfacial elements entering the front at early stages of elimination. Since the equations corresponding to the block $\mathbf{J_{ff}}$ in (2.13) can not be eliminated, they should enter the frontal matrix last. This can be effectively achieved by resequencing the finite elements so that the elements on the interfaces are numbered last within a subdomain. Scott's approach II augments the dual finite-element graph of the mesh with one vertex and edges connected to all interfacial vertices. By

taking this additional vertex as a pseudo peripheral vertex, a second pseudo peripheral vertex is computed and used as the starting vertex in Sloan's element reordering Sloan (1989). Once the reordering is completed, the additional vertex is eliminated. Since the additional vertex was the last vertex to be renumbered, all interfacial vertices connected to it will be contained in the last level structure of the graph, and therefore they are labeled last.

4 Partitioning the interface

Since we adopt a graph theoretical approach to describe the interface partitioning problem, a few basic concepts from graph theory are first presented. A *graph* $G = (V, E)$ consists of a set V of *vertices* along with a set E of *edges* where an *edge* is a pair (v_1, v_2) of distinct vertices in V. A *subgraph* $\tilde{G} = (\tilde{V}, \tilde{E})$ of $G = (V, E)$ is a graph which consists of some or all vertices of G and some of the edges of $G : \tilde{V} \subseteq V, \tilde{E} \subset E$. The *subgraph* is a *section graph* when $\tilde{V}$ consists of only some of the vertices of G and $\tilde{E}$ consists of all edges (v_1, v_2) that are in $\tilde{V}$:

$$\begin{aligned} \tilde{V} &\subset V \\ \tilde{E} &= \left\{ (v_1, v_2) \in E \ \| \ v_1 \in \tilde{V} \ \ and \ \ v_2 \in \tilde{V} \right\} \end{aligned} \tag{4.1}$$

A graph is *connected* if every pair of vertices is connected by a path. Otherwise, the graph is disconnected. Usually a disconnected graph consists of two or more connected components. We only consider connected graphs of finite elements for the present study. A separator is a set of vertices that renders a connected graph disconnected when that set of vertices along with their incident edges are removed from the graph. A vertex is called a *cutvertex* when it is itself a separator. For our purposes, we define degree of a cutvertex as the number of subdomains incident to it and denote it as $deg(C_i)$ for a cutvertex C_i.

Now let us consider a partitioning of a finite-element mesh to subdomains. We wish to represent the global interfaces among the neighboring subdomains in terms of interface graphs. Depending on the partitioning scheme used for the subdomains, there will be a number of interfaces joining only at certain points in 2D and along lines in 3D. For clarity of exposition, we will restrict our discussion to 2D domains for this study. However, the basic ideas are easily extended to 3D geometries as well. To illustrate the main ideas, let us consider the partitioning in figure 4.1. It should be noted that the point at which two interfaces intersect in 2D coincides with a physical node of a finite element sharing these two interface segments. This node then becomes a cutvertex or a separator in the interface graph. For example, the separators of the

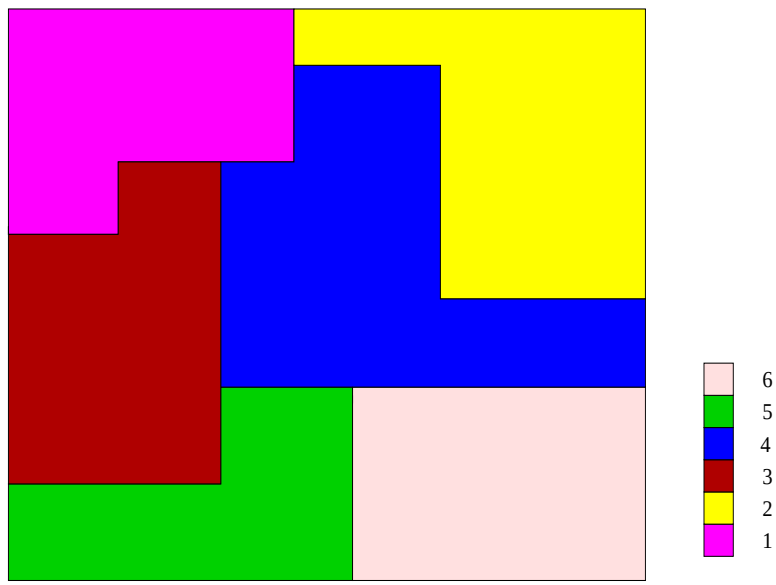

Fig. 4.1: *Partitioning of a finite element mesh to six subdomains.*

graph of global interfaces for the partitioning in figure 4.1 are shown in figure 4.2. The degree of any cutvertex can be calculated from the number of subdomains incident to it, for example, $deg(C_4) = 2$, $deg(C_5) = 3$, etc. The interface segment between any two end separators consists of nodes of finite elements incident to that interface. Let us denote the set of all nodes between two separators i and j along an interface by v_{ij}. Using these definitions, we can represent all the vertices of an interface graph, as shown in figure 4.3 for subdomains 1, 2, and 4.

The interface partitioning strategy can then be simply explained using this example. Let us assume that the interfaces incident with subdomains $1, 2$, and 4 in figure 4.3 belong to the same partition. Obviously, not all of the unknowns on these interface segments can be solved independently of the other partitions. The following rules, however, are always satisfied.

1. *If either of the cutvertices i or j of an interface graph can be eliminated, then the intermediate set of vertices v_{ij} is also a candidate for elimination.*

2. *A separator or cutvertex can be eliminated only when its degree has been reduced to 2 by merging incident subdomains for a particular interface partition.*

The graph of the interface for subdomain 1 can be represented as $G^{(1)} = (V^{(1)}, E^{(1)})$ where the set of all vertices on the interface is $V^{(1)} = \{C_1, v_{12}, C_2, v_{23}, C_3, v_{34}, C_4\}$. Note that the degree of any interface nodal subset v_{ij} is always 2. However, the cutvertices C_i have different degrees depending on the original decomposition of the

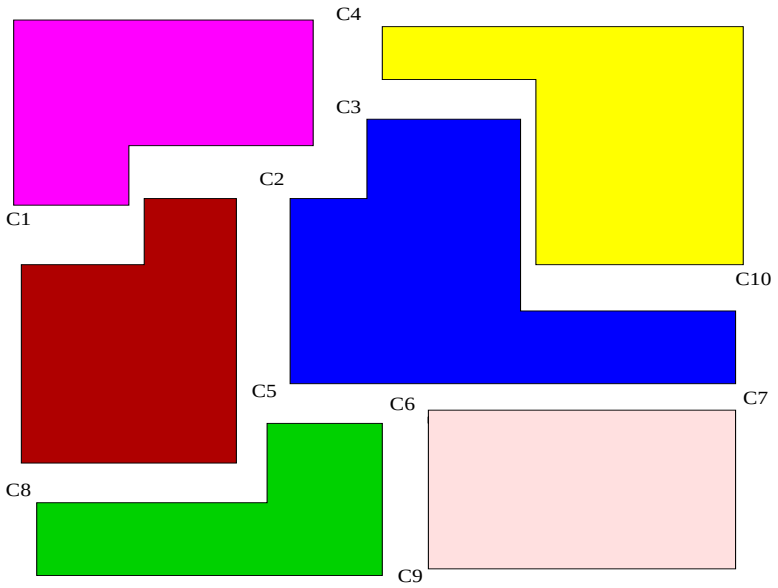

Fig. 4.2: *Separators in the graph of global interfaces.*

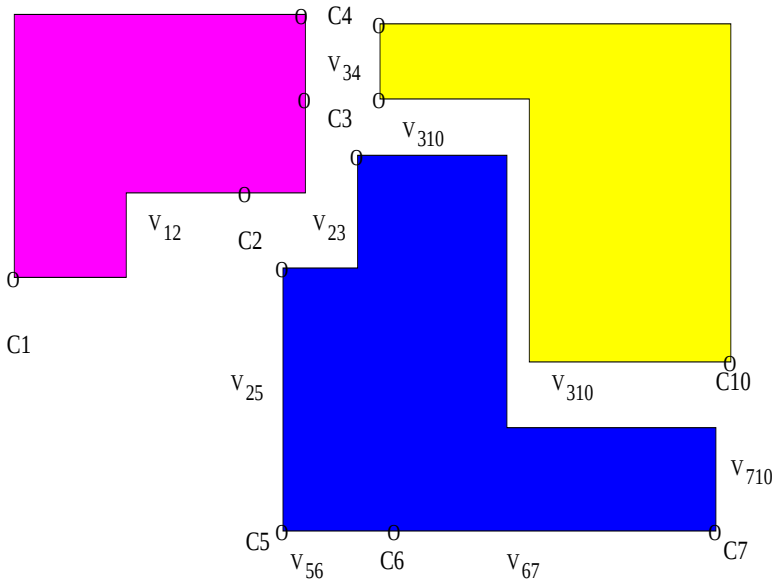

Fig. 4.3: *Vertices of interface graphs for subdomains* 1, 2, *and* 4.

finite-element mesh.

The entire set of vertices $V^{(1)}$ is non-reducible; that is they cannot be eliminated independently of the other partitions. Next, consider the set of vertices on the interface of subdomain 2, $V^{(2)} = \{C_4, v_{34}, C_3, v_{310}, C_{10}\}$, and make $V^{(2)}$ part of the set of vertices that cannot be eliminated. Next, we search for common separators of $V^{(1)}$ and $V^{(2)}$; (C_3 and C_4). The separator C_4 can be readily eliminated (following *Rule 2* above) since its degree is 2. Then, by *Rule 1*, the intermediate vertices given by the set v_{34} can also be eliminated. Since $deg(C_3) = 3$, the separator C_3 cannot be eliminated at this stage. However, once subdomains 1 and 2 are merged, we can reduce the degree of C_3 by one so that $deg(C_3) = 2$. For example, it can be eliminated if subdomain 4 is also merged with subdomains 1 and 2.

After subdomains 1 and 2 are merged, the sets of internal and boundary unknowns on the resulting interface are

$$\begin{aligned} I &= \{C_4, v_{34}\} \\ N &= \{C_1, v_{12}, C_2, v_{23}, C_3, v_{310}, C_{10}\}\,, \end{aligned} \tag{4.2}$$

respectively. The internal vertices given by the set I can be readily eliminated and a resulting Schur complement problem can be written in terms of the unknowns corresponding to the vertices in N. We describe the implication for the matrix problem in a later section. A merging of the interface of subdomain 4 to the current sets will then result in a new list for both I and N as follows:

$$\begin{aligned} I &= \{C_4, v_{23}, v_{34}, C_3, v_{310}, C_{10}\} \\ N &= \{C_1, v_{12}, C_2, v_{25}, C_5, v_{56}, C_6, v_{67}, C_7\}\,. \end{aligned} \tag{4.3}$$

The above example assumes that the interfaces of subdomains 1, 2, and 4 form one single partition of the global interface problem. If one wishes to solve the global interface problem on two interface partitions or interface subregions in parallel, the rest of the interface segments incident with subdomains 3, 5, and 6 will be part of the second interface subregion and can be treated in the same manner as above. For both interface subregions, we can eliminate the respective internal unknowns and write the resulting Schur complement problem in terms of the remaining unknowns. The resulting final "reduced" Schur complement problem can then be assembled and solved on one processor. Clearly, the above approach involves two partitioning and elimination cycles, the first for subdomains with interfaces and the second for interface subregions. The final 'reduced' Schur complement problem involves only a very small subset of the original problem.

We now present the algorithm for creating lists I and N for each interface partition.

Algorithm 1

```
for k = 0 to n interface partitions
    generate a list of m subdomains (S^(k)) for partition k
    for i = 0 to m subdomains
        N^(k) ← N^(k) ∪ V^(i) (initialization)
        if i > 0 then (merging of subdomains)
        find s minimal separators between sets N^(k) and V^(i)
        for j = 0 to s separators
            if deg(C_j) = 2 then (eliminate)
              I^(k) ← I^(k) ∪ C_j ∪ v_jl (jl: connected vertices)
              remove C_j and v_jl from N^(k)
            else
              deg(C_j) = deg(C_j) − 1
            endif
        endfor
    endfor
endfor
```

For a given number of subdomains, the number of interface partitions to be divided among processors for the solution of the global Schur complement problem is not fixed. For elimination of common vertices, at least two subdomains must be merged for each interface segment. Benner *et al.* (1987) present an algorithm based on nested dissection in which two subdomains are merged in parallel to eliminate the incident interface in n-steps for 2^n subdomains. The disadvantages of this method are: the parallelism is progressively reduced and that their scheme is best suited to structured meshes. Moreover, computing n nested-dissection steps in the multifrontal algorithm on a distributed-memory multiprocessing system using message passing requires many communication cycles and may not scale at all beyond a few subdomains.

Recently, a family of linear solvers based on selective orderings have been proposed. Notably, SPOOLES (Ashcraft *et al.*, 1999) from Boeing Phantom Works employs multilevel nested dissection and multisection (a hybrid algorithm of nested dissection and minimum-degree ordering) to find an ordering in the graph of the matrix to be factored. Subsequently, the vertices in the elimination graph are grouped together to form fundamental supernode trees which can be solved in parallel. This approach requires renumbering the vertex labels of the entire graph and, therefore,

may not be convenient for the type of large distributed finite-element graphs we consider here. Another recent publication (LI *et al.*, 1999) explores the notion of unsymmetric supernodes to perform most of the numerical computation in dense matrix kernels. The multi-threaded version of the solver known as SuperLU runs on shared-memory machines and employs a depth-first search along with symmetric structural reductions to speed up the symbolic factorization and improve cache performance. Later, we present a number of examples to illustrate the effect of partitioning the Schur complement problem over different processor configurations for a given number of subdomains using the new algorithm described here.

5 Matrix representation

The specific problem of merging adjacent subdomain interfaces and the resulting "reduced Schur complement" problem is considered next. For the time being, let us assume that only interface unknowns contribute to the Schur complement problem for any subdomain. Consider two neighboring subdomains, $\Omega_{(p)}$ and $\Omega_{(q)}$, sharing an interface segment Γ_{pq}. The subdomain Schur complement problems are

$$\mathbf{S^{(p)}u^{(p)} = b^{(p)} + r^{(p)}} \tag{5.1}$$

$$\mathbf{S^{(q)}u^{(q)} = b^{(q)} + r^{(q)}}, \tag{5.2}$$

corresponding to the interfaces $\partial\Omega_{(p)}$ and $\partial\Omega_{(q)}$, respectively. The interface unknowns are denoted as $\mathbf{u^{(p)}}$ and $\mathbf{u^{(q)}}$. The subdomain Schur complement matrices $\mathbf{S^{(p)}}$ and $\mathbf{S^{(q)}}$ are generated using a frontal algorithm in each of the subdomains p and q. Note that both subdomains p and q may have interfaces shared with other neighboring subdomains. The contribution of the neighboring interfaces is represented by the vectors $\mathbf{r^{(p)}}$ and $\mathbf{r^{(q)}}$ in (5.1) and (5.2). Let us assume that these two subdomain interfaces are merged to form a global interface partition. The algorithm shown earlier can be employed to generate the necessary lists $\mathcal{N}$ and I for merging the interfaces. The unknowns corresponding to the shared interface Γ_{pq} will be part of I and the rest of the unknowns part of $\mathcal{N}$. This induces a block representation of the Schur complement problem, ((5.1) and (5.2)), in each subdomain. First, we introduce a local ordering of the subdomain interface nodal unknowns so that those on Γ_{pq} are numbered as $\mathbf{u_e}$. The remaining unknowns on each subdomain interface are labelled as $\mathbf{u_f^{(p)}}$ and $\mathbf{u_f^{(q)}}$, respectively. In the absence of any other subdomain interfaces, the lists I and $\mathcal{N}$ can then be expressed as

$$I = \{\mathbf{u_e}\}$$
$$\mathcal{N} = \left\{\mathbf{u_f^{(p)}}, \mathbf{u_f^{(q)}}\right\}. \tag{5.3}$$

This reordering of the interface unknowns can be represented using a permutation matrix $\mathbf{Q}_{(.)}$ as

$$(\mathbf{u_e}\ ,\ \mathbf{u_f^{(p)}})^T = \mathbf{Q_{(p)}^T}\ \mathbf{u^{(p)}} \tag{5.4}$$
$$(\mathbf{u_e}\ ,\ \mathbf{u_f^{(q)}})^T = \mathbf{Q_{(q)}^T}\ \mathbf{u^{(q)}}, \tag{5.5}$$

respectively, for the two subdomains.

Under these reorderings, (5.1) and (5.2) become

$$\tilde{\mathbf{S}}^{(p)}\tilde{\mathbf{u}}^{(p)} = \tilde{\mathbf{b}}^{(p)} + \tilde{\mathbf{r}}^{(p)} \tag{5.6}$$
$$\tilde{\mathbf{S}}^{(q)}\tilde{\mathbf{u}}^{(q)} = \tilde{\mathbf{b}}^{(q)} + \tilde{\mathbf{r}}^{(q)}, \tag{5.7}$$

respectively, for subdomains p and q, where

$$\tilde{\mathbf{S}}^{(p)} = \mathbf{Q_{(p)}^T}\mathbf{S^{(p)}}\mathbf{Q_{(p)}},\ \ \tilde{\mathbf{b}}^{(p)} = \mathbf{Q_{(p)}^T}\mathbf{b^{(p)}},\ \ \tilde{\mathbf{r}}^{(p)} = \mathbf{Q_{(p)}^T}\mathbf{r^{(p)}} \tag{5.8}$$

and

$$\tilde{\mathbf{S}}^{(q)} = \mathbf{Q_{(q)}^T}\mathbf{S^{(q)}}\mathbf{Q_{(q)}},\ \ \tilde{\mathbf{b}}^{(q)} = \mathbf{Q_{(q)}^T}\mathbf{b^{(q)}},\ \ \tilde{\mathbf{r}}^{(q)} = \mathbf{Q_{(q)}^T}\mathbf{r^{(q)}}, \tag{5.9}$$

Matrices $\tilde{\mathbf{S}^{(p)}}$ and $\tilde{\mathbf{S}^{(q)}}$ then have the following block structure:

$$\tilde{\mathbf{S}}^{(p)} = \begin{bmatrix} \mathbf{S_{ee}^{(p)}} & \mathbf{S_{ef}^{(p)}} \\ \mathbf{S_{fe}^{(p)}} & \mathbf{S_{ff}^{(p)}} \end{bmatrix} \tag{5.10}$$

and

$$\tilde{\mathbf{S}}^{(q)} = \begin{bmatrix} \mathbf{S_{ee}^{(q)}} & \mathbf{S_{ef}^{(q)}} \\ \mathbf{S_{fe}^{(q)}} & \mathbf{S_{ff}^{(q)}} \end{bmatrix}. \tag{5.11}$$

Note that the matrices $\mathbf{S_{ee}^{(p)}}$ and $\mathbf{S_{ee}^{(q)}}$ represent the respective contributions of subdomains p and q for the interface unknowns on the common interface segment

Γ_{pq} to the global Schur complement matrix. We now perform a summation of these two matrices; that is, we compute $\mathbf{S_{ee}} = \mathbf{S_{ee}^{(p)}} + \mathbf{S_{ee}^{(q)}}$. This step requires interprocessor communication between subdomains p and q. This is also the assembled Schur complement matrix corresponding to the unknowns $\mathbf{u_e} \in I$ on Γ_{pq}. Let us denote the unknowns associated with the list $\mathcal{N}$ by $\mathbf{u}_\phi$ as a result of merging the interfaces of subdomains p and q. As a result, the merged Schur complement matrix can be expressed as

$$\begin{bmatrix} \mathbf{S_{ee}} & \mathbf{S_{e\phi}} \\ \mathbf{S_{\phi e}} & \mathbf{S_{\phi\phi}} \end{bmatrix} \begin{bmatrix} \mathbf{u_e} \\ \mathbf{u}_\phi \end{bmatrix} = \begin{bmatrix} \mathbf{b_e} \\ \mathbf{b}_\phi \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{r}_\phi \end{bmatrix}, \tag{5.12}$$

where the blocks $\mathbf{S_{e\phi}}$, $\mathbf{S_{\phi e}}$, and $\mathbf{S_{\phi\phi}}$ consist of contributions from subdomain interfaces other than p and q and, hence, cannot be eliminated at this stage. However, the matrix block $\mathbf{S_{ee}}$ corresponding to the unknowns $\mathbf{u_e}$ is fully summed since Γ_{pq} consists of only the interface between subdomains p and q. The vector $\mathbf{r}_\phi$ corresponds to the flux term contribution of $\mathbf{u}_\phi$ and unassembled interface unknowns from neighboring subdomains. Finally, the unknowns $\mathbf{u_e}$ can be eliminated from (5.12) and the reduced Schur complement problem for the merged subdomain interfaces can be expressed as

$$\left[\mathbf{S}_{\phi\phi} - \mathbf{S_{e\phi}}\mathbf{S_{ee}^{-1}}\mathbf{S_{\phi e}}\right]\mathbf{u}_\phi = \mathbf{b}_\phi + \mathbf{r}_\phi - \mathbf{S_{e\phi}}\mathbf{S_{ee}^{-1}}\mathbf{b_e}. \tag{5.13}$$

The subdomain interfaces can be partitioned into a number of global interface segments by merging neighboring subdomains. Each global interface partition generates a 'reduced' Schur complement problem, as given in (5.13), which consists of only a fraction of the original Schur complement problem. These reduced Schur complement matrices may now be easily assembled onto one processor and solved for the global cutvertices.

6 Data structure and implementation

The algorithm described in the earlier sections has been implemented in a general purpose p-adaptive finite-element program termed Pfinics. A given finite-element mesh is divided into a specified number of subdomains using Metis (KARYPIS AND KUMAR, 1995) and a set of pre-processing tools available in Pfinics. This pre-processing phase performs the following tasks :

1. prepare a graph of element connectivity with element centroids as vertices.

2. partition element graph using a multilevel k-way partitioning scheme available via Metis.
3. reorder the subdomains (output of Metis) to minimize the front-width of the global assembled Schur complement matrix.
4. for each subdomain in the mesh, find internal and interface element numbers; reorder local subdomain mesh to reduce front-width of subdomain frontal factorization; generate send/receive lists of local nodes for each neighboring subdomain; find local boundary conditions from global finite element mesh; prepare input file for Pfinics solver.
5. each subdomain mesh is then assigned to a processor within a portable distributed object-oriented framework for clusters of workstations and distributed memory supercomputers such as the Cray T3E.

Communication among processors is handled using the standard message-passing interface MPI (Gropp *et al.*, 1994) protocols. For instance, all data are sent to processors using a non-blocking protocol called MPI_Isend. This allows a processor to proceed to the next level of computation without waiting for the receiving processor to read the data. To prevent bottlenecks, all processors receive data using a blocking MPI_recv call. The resulting software is portable from workstation clusters to supercomputers and current MPP architectures. A nested linked list (PEList) keeps track of neighboring processor numbers (PE's) for each processor in the system and local interface node numbers shared with each neighbor. Another linked list (ElemList) maintains internal and interface element numbers for each subdomain. This information is used in the subdomain frontal solver to indicate which unknowns are fully summed.

An advanced memory management model is used in Pfinics, resulting in truly dynamic memory allocation and deletion as required for adaptive p-finite-element methods. More information on data structure and an object-oriented implementation can be found in Bose and Carey (1997).

7 Numerical experiments

As a representative application to demonstrate the ideas, we consider steady flow of a viscous incompressible fluid. The global laws of conservation of mass and momentum in a region Ω of isothermal flow may be written in dimensionless form as

$$\nabla \cdot \mathbf{v} = 0 \tag{7.1}$$

$$\mathbf{v} \cdot \nabla \mathbf{v} = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \frac{\mathbf{g}}{\mathbf{W_g}}, \tag{7.2}$$

where $p, \mathbf{v}, \boldsymbol{\tau}$, and $\mathbf{g}$ denote a scalar pressure field, velocity vector, stress tensor and body force vector, respectively, at a spatial point $\mathbf{x} \in \Omega$. The constitutive equation for a Newtonian fluid is given by

$$\boldsymbol{\tau} = \frac{1}{Re} \left(\nabla \mathbf{v} + \nabla \mathbf{v}^{\mathbf{T}} \right), \tag{7.3}$$

where W_g ($= \frac{{v_0}^2}{g_o l}$) is defined as an inertial buoyancy number and $Re = \frac{\rho v_o l}{\mu}$ is the Reynolds number. The parameters ρ, μ, and v_0 are density, viscosity, and a characteristic velocity, respectively.

The boundary $\partial\Omega$ consists of non-overlapping parts $\partial\Omega_d$ and $\partial\Omega_n$ such that $\overline{\partial\Omega_d \cup \partial\Omega_n} = \partial\Omega$. The typical boundary conditions are

$$\mathbf{v} = \mathbf{v_0} \qquad on\ \partial\Omega_d \tag{7.4}$$

$$\boldsymbol{\tau} \cdot \mathbf{n} = \mathbf{t_0} \qquad on\ \partial\Omega_n, \tag{7.5}$$

Here we present a least-squares finite-element formulation for (7.1)-(7.3) but emphasize that the parallel multiple front algorithm can be employed to solve the discretized matrix problem arising out of other weak formulations, such as Galerkin's method. A review of least-squares finite-element methods for fluid flow and transport for a range of problems including convection-diffusion, Navier-Stokes, and viscoelastic flows is given in CAREY *et al.* (1998).

A least-squares minimization functional for admissible fields $\mathbf{v}$, p and stress components $(\tau_{xx}, \tau_{xy}, \tau_{yy})$ in Ω can be constructed by introducing the corresponding residuals $\mathbf{R}_f$ for (7.1)-(7.3) as follows:

$$\mathcal{F} = \| \mathbf{R_f} \|^2_{L^2(\Omega)}, \tag{7.6}$$

where the residuals $\mathbf{R}_f$ $(Re \geq O(1))$ for admissible approximation fields $\mathbf{v_h}$, p_h, and $\tau_\mathbf{h}$ in the finite-element spaces are simply

$$\mathbf{R}_f = \begin{bmatrix} \nabla \cdot \mathbf{v}^h \\ \mathbf{v}^h \cdot \nabla \mathbf{v}^h + \nabla p^h - \nabla \cdot \boldsymbol{\tau}^h - \frac{\mathbf{g}}{\mathbf{W_g}} \\ \boldsymbol{\tau}^h - \frac{1}{Re} \left(\nabla \mathbf{v}^h + \nabla \mathbf{v}^{h^T} \right) \end{bmatrix}. \tag{7.7}$$

Taking variations leads to a nonlinear algebraic system to be solved for the nodal solution values of velocities, pressure, and stresses. The usual consistency conditions

associated with the Galerkin approach do not have to be enforced. (It is well known that the mixed Galerkin method is not stable for certain basis combinations. For example, locking to $\mathbf{v}_h = \mathbf{0}$ occurs for the linear-velocity, constant-pressure triangle and spurious pressure modes can occur for other choices of bases (Carey and Oden, 1986). These restrictions on the bases do not apply to the least-squares mixed finite-element formulation.) Consequently, in the results presented below, we use $\mathbf{C}^o$ equal-order bases for all variables for convenience (although other choices are possible and may be preferable).

We first consider a common two-dimensional viscous-flow benchmark problem of isothermal driven cavity flow. A unit tangential velocity is applied at the top wall with no slip on the sides and bottom. Our purpose is to study the effects of h and p-refinement on the performance of the present algorithm and do a parallel scalability study. All numerical experiments were performed on a CRAY T3E system at the Advanced Computing Facility of the University of Texas at Austin.

We used three different uniform discretizations, namely, 10×10, 16×16, and 20×20 grids denoted as Meshes I, II, and III, respectively, in all subsequent discussion. The three respective meshes are partitioned into 3, 7, and 15 subdomains. Since we currently use processor zero in the communication group for solving the assembled global Schur's complement problem, we run the above cases on 4, 8, and 16 processors, respectively.

Figure 7.1 shows timing results on 4, 8, and 16 processors of the T3E for both Mesh I and Mesh III with quadratic basis functions ($p = 2$). The present algorithm does well for up to eight processors for these two cases but does not scale beyond eight processors. In Table 7.1, we see that for 16 processors, the size of the assembled global Schur's complement problem is larger than the maximum number of unknowns in any subdomain (denoted within brackets in the table). The total CPU time in this case is dominated by the time required to solve the global Schur's complement problem. However, for good scaling with increasing number of processors, the size of the global Schur's complement problem must remain small. With p-enrichment of the element basis functions on a fixed mesh, the ratio of eliminated to remaining unknowns increases progressively. Now the time required for subdomain Jacobian matrix factorization and forward/back-solve steps is again more than that required for the assembled global Schur's complement problem. This results in better timing results, as seen in Figure 7.2, which compares CPU times for Mesh I with polynomial orders of 2 and 7, respectively.

The effect of increasing problem size on a fixed number of processors is shown in Figure 7.3. Note that for p equal to 2, both the 8 and 16-processor cases take approximately the same amount of time. However, as the polynomial order is increased, the

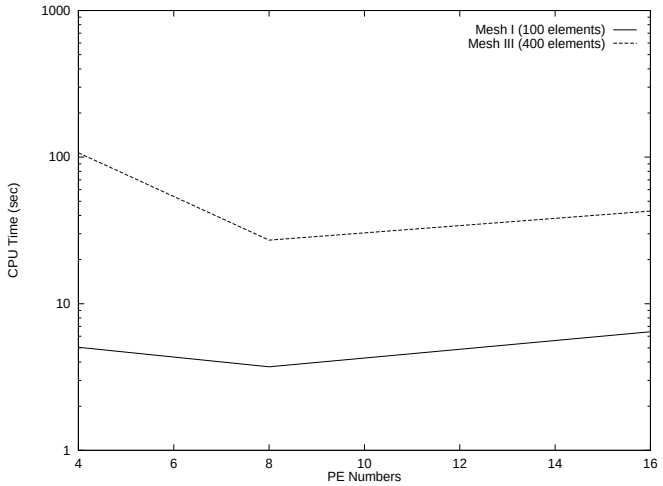

Fig. 7.1: *CPU Time for Mesh I and III with quadratic basis functions.*

Table 7.1: *Global Schur's complement and subdomain unknowns for Meshes I and III (p=2).*

	Number of Processors (npes)		
	$npes = 4$	$npes = 8$	$npes = 16$
$MeshI$	234 (978)	480 (474)	834 (282)
$MeshIII$	462 (822)	918 (1614)	1578 (822)

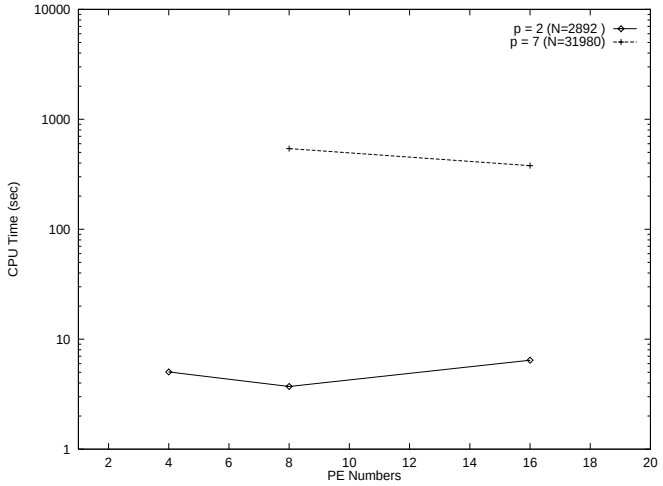

Fig. 7.2: *CPU Time for Mesh I (p = 2 and 7).*

timing results with 16 processors improve substantially. Obviously, the subdomain solution time for frontal factorization and for forward and backward solution steps scales well as the number of processors is increased, as shown in Figure 7.4 for Mesh I and polynomial order $p = 5$. However, because of the increasing number of interface unknowns, the frontal solver for the assembled global Schur's complement problem takes more time as the number of processors is increased. This bottleneck can be significantly reduced by devising an algorithm for parallel solution of the interface (i.e., Schur's complement) problem.

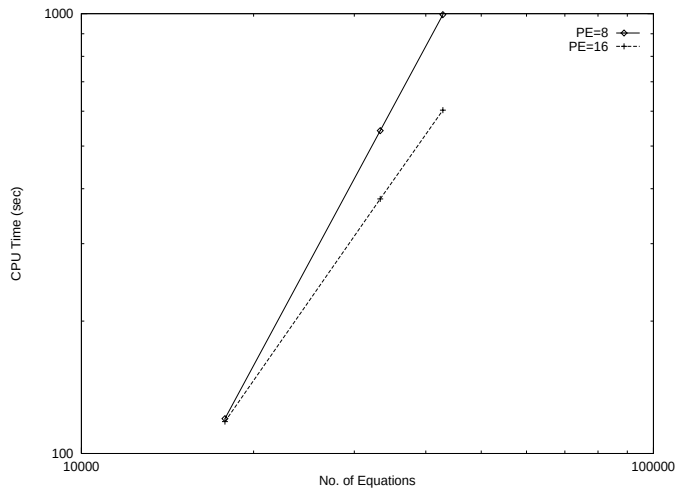

Fig. 7.3: *Effect of increasing polynomial orders (Mesh I) on 8 and 16 Processors.*

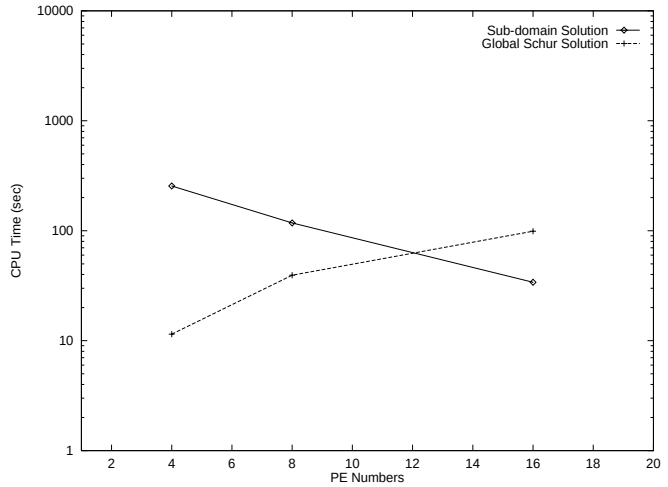

Fig. 7.4: *Frontal solver performance for Mesh I ($p = 5$, 17820 equations).*

As the polynomial degree of the basis is raised, matrix conditioning deteriorates and the rate of convergence of iterative solvers is adversely affected (Bramble *et al.*,

1986). Matrix conditioning in direct solvers can also be a problem; for example, this can significantly affect the stability of a matrix factorization, owing to the presence of very small pivots along fully assembled rows or columns. This is a more serious issue for substructuring since only a small sub-mesh is used in each subdomain factorization and the number of assembled rows or columns in each subdomain is much smaller than that in the fully assembled global domain. As pointed out earlier, the present algorithm prescribes a conservative pivoting criterion and some equations in the subdomain matrix are left uneliminated if no acceptable pivots have been found.

The choice of polynomial basis functions for a particular finite-element formulation also affects the condition number of the element matrices, and as a result, the subdomain Jacobian matrix. For example, consider the previous example involving a finite-element formulation of the incompressible Navier-Stokes equations based on least-squares minimization using hierarchic Lagrange and Legendre polynomials.

Table 7.2: *Schur's complement problem size for Mesh I on eight processors (Lagrange).*

	Processor Number (PE)							
p	$PE=0$	$PE=1$	$PE=2$	$PE=3$	$PE=4$	$PE=5$	$PE=6$	$PE=7$
2	480	114 25.7%	162 34.2%	114 26.1%	102 22.1%	162 36.9%	216 46.8%	126 27.3%
5	3249	545 23.2%	773 30.6%	420 17.9%	545 21.8%	572 24.4%	823 34.4%	807 33.5%
7	7263	1055 23.6%	1714 35.8%	746 16.7%	1049 22.1%	1302 29.2%	1292 28.6%	1821 40.0%

Table 7.3: *Schur's complement problem size for Mesh I on eight processors (Legendre).*

	Processor Number (PE)							
p	$PE=0$	$PE=1$	$PE=2$	$PE=3$	$PE=4$	$PE=5$	$PE=6$	$PE=7$
2	480	114 25.7%	162 34.2%	114 26.1%	102 22.1%	162 36.9%	216 46.8%	126 27.3%
5	1200	276 11.7%	396 15.7%	276 11.8%	246 9.9%	396 16.9%	540 22.6%	306 12.7%
7	1680	384 8.6%	552 11.5%	384 8.6%	342 7.2%	552 12.4%	756 16.8%	426 9.4%

The data in Table 7.2 are from a partition of Mesh I to seven subdomains. We consider tensor-product polynomial basis functions for elements in the mesh. The numbers in the second row for each p-level denote the percentage of Schur's complement unknowns with respect to total number of unknowns in a typical subdomain.

Although the software can handle non-uniform p-refinement, for simplicity, the polynomial order for this study is uniform over all subdomains. This also facilitates load balancing among processors. Notice that for $p = 2$, both Tables 7.2 and 7.3 show the same number of Schur's complement unknowns $\mathbf{u}_{\mathbf{f}}^{(\mathbf{i})}$ for all subdomains. However, for higher p levels, Lagrange polynomials contribute many internal unknowns to the subdomain Schur's complement problem, whereas hierarchical basis functions consisting of Legendre polynomials contribute none. Therefore, for Legendre polynomials, the Schur's complement problem for each subdomain consists of only interface unknowns. This is evident from the above table. The present algorithm requires that the size of the Schur's complement problem be small compared with the size of the subdomain problem. For Lagrange polynomials, the size of the Schur's complement problem grows exponentially with polynomial order, resulting in poorer efficiency of the present algorithm. Moreover, it should be noted that the percentage of Schur's complement unknowns with respect to total number of unknowns in any subdomain decreases as the order of the basis functions increases, only when Legendre polynomials are used.

We also consider least-squares finite-element solution of channel flow into a planar 4:1 contraction as a second example. We refer to (Carey *et al.*, 1998) for the finite-element formulation and related issues. Due to symmetry, only one-half of the planar contraction is modeled. Complete details of the flow problem and graphs of the solution field are given by Bose (1997). Here we focus on the performance of the recursive interface partitioning algorithm and parallel scaling. Representative samples of partitionings of the original domain to subdomains are given in figures 7.5 and 7.6.

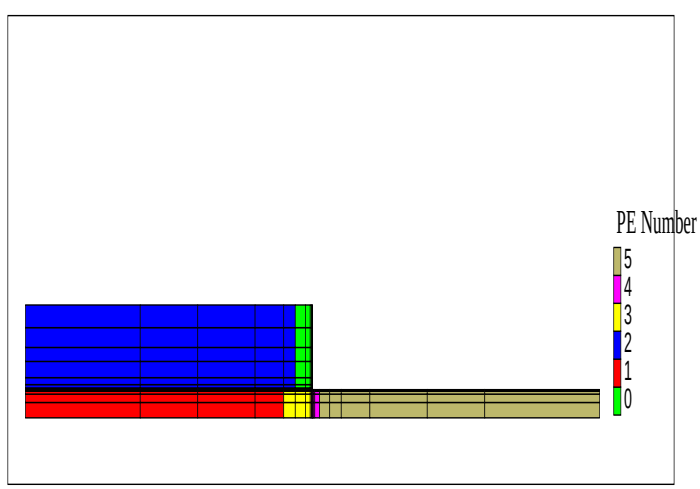

Fig. 7.5: *6-Subdomain partition of the 4:1 contraction geometry.*

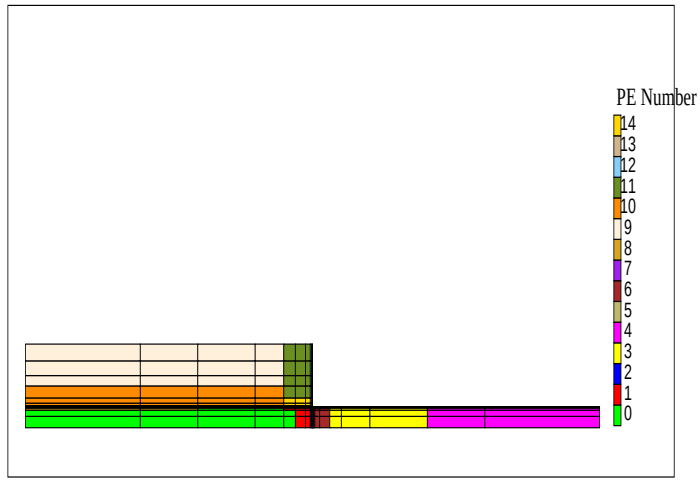

Fig. 7.6: *15-Subdomain partition of the 4:1 contraction geometry.*

Figure 7.7 shows timing results for parallel computations with uniform p-refinement on a fixed mesh. Note that for $p = 2$ (i.e., quadratic basis functions), the parallel multiple front algorithm does not scale beyond nine processors. For higher p-orders, however, we get better performance. For example, the bottleneck in the Schur complement solution step does not appear for $p = 5$ up to 15 processors. For polynomial enrichment using hierarchical bases, most of the extra unknowns are added in the interior of an element. Therefore, for coarse granularity systems and uniform p-refinement on a fixed mesh, the size of the subdomain matrix problem is large compared with that of the global Schur complement problem. For this reason, solving the entire global Schur complement problem on one processor is still acceptable. However, for a fixed mesh and fixed p-order distribution, as the number of subdomains and processors is increased, the size of the interface problem becomes large and eventually an unacceptable bottleneck occurs.

Figure 7.8 shows the performance of the new distributed Schur complement solution algorithm embedded within the parallel multiple front solution scheme for $p = 6$. A linear speedup curve based on the first data point is provided for comparison. The time required for solution of the Schur's complement problem is also presented. The improvement in parallel efficiency for the present algorithm can be explained from the relatively bounded curve for the Schur's complement solution time for the processor configurations we have tested so far. Figure 7.9 compares the solution time for the Schur's complement problem for the two algorithms. Note that the size of the Schur's complement problem is too large to solve on one processor because of memory limitation when the number of subdomains is sufficiently large (e.g., 18 for this example). This is further motivation for solving the Schur's complement

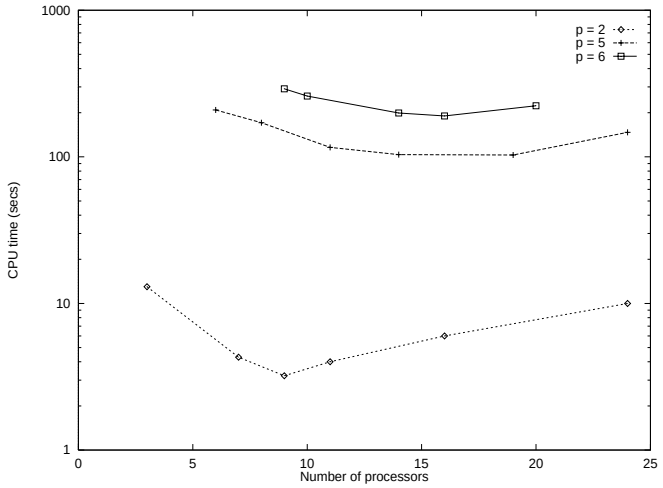

Fig. 7.7: *CPU time for different polynomial orders.*

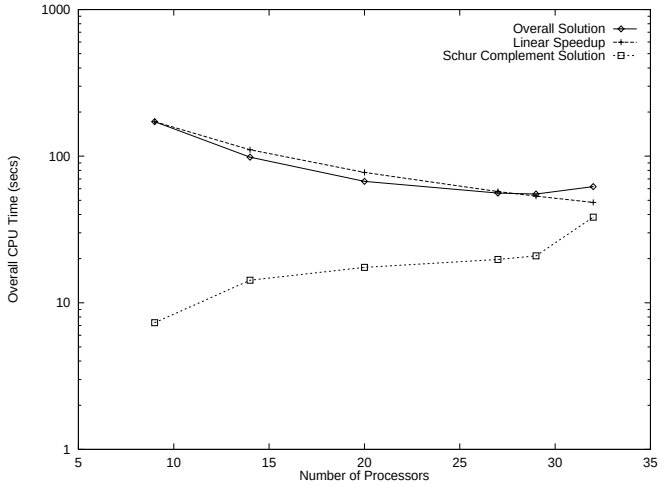

Fig. 7.8: *Performance of distributed Schur complement solution (p=6, 4:1 contraction).*

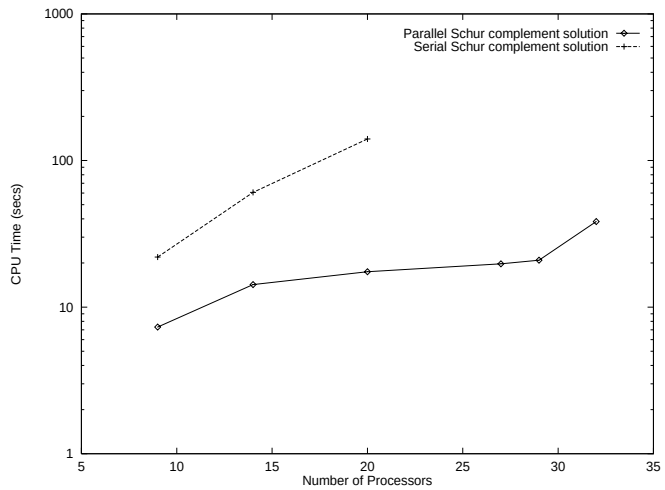

Fig. 7.9: *Serial and parallel Schur complement solution timings (p=6, 4:1 contraction).*

problem distributively across a number of processors using either the present scheme or a parallel iterative scheme.

Next, we comment on the partitioning of the Schur's complement problem across a number of processors. Since the global interface segments are eliminated via merging of subdomain interfaces, the greater the number of subdomain interfaces that are put on the same interface partition (for the Schur's complement solution step), the fewer the number of unknowns in the reduced Schur's complement problem. The minimum number of subdomains that must be merged to form a global interface partition is 2, and therefore, one can employ half the number of original processors for solution of the Schur's complement problem. However, this increases the size of the reduced Schur's complement problem, and indeed it may be efficient to employ just a few processors for solving the Schur complement problem, as evidenced in figure 7.10 for an original finite-element mesh ($p = 6$) divided into 31 subdomains. Note that as the number of interface partitions is increased, the reduced Schur's complement problem gets larger since fewer subdomains are merged for each interface partition. For the problem sizes we have considered in the present study, it was sufficient to employ only four or five processors for the Schur's complement solution step as indicated by the local minimum. It is not straightforward to decide upon an optimum number of interface partitions for the Schur complement problem. However, the objective is to balance the time required for solving the merged interface problems in parallel with the time required for solution of the reduced Schur complement problem on one processor. Note that the timings presented in figure 7.10 are still better than that required for solution of the Schur complement problem on one processor even though

we get optimum performance when only three or four partitions are used.

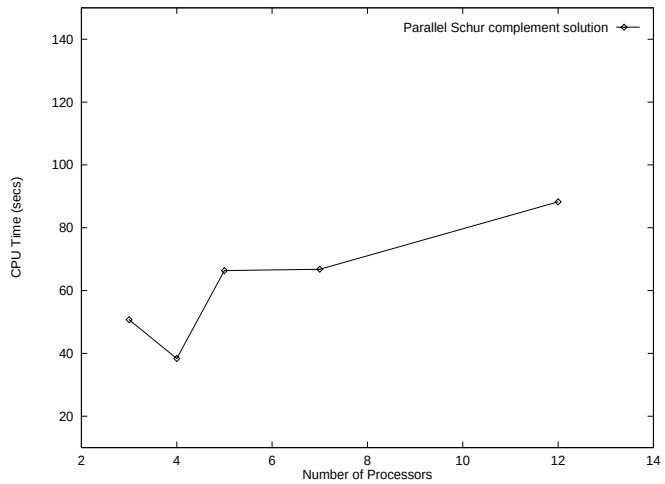

Fig. 7.10: *Performance of parallel Schur complement algorithm (p=6,* $4:1$ *Contraction).*

Finally, the parallel Schur's complement solution algorithm is applied to our first flow problem, that is, isothermal cavity flow in a unit square. We show a representative partition to 31 subdomains of the original domain of 20×20 elements with a uniform polynomial order of 5, as shown in figure 7.11. For this partitioning, we present the performance of the parallel algorithm in figure 7.12 on five processors (i.e., four interface partitions). It should be noted that the idle processors during the parallel Schur complement solution step may be returned to the resource pool for other processes. Such dynamic resource management is necessary for distributed systems based on networks of workstations and is an active area of research. Overall, these two numerical experiments demonstrate an order of magnitude of improvement in parallel efficiency and scalability over the serial Schur's complement solution algorithm.

8 Conclusion

We have presented a class of multiple front algorithms based on domain decomposition for parallel direct solution of algebraic systems resulting from discretization of boundary-value problems in continuum mechanics. The basic algorithm is suitable for coarse grain parallelism, for example, tightly coupled workstation clusters and low-end multiprocessor systems. Its efficiency improves with p-refinement on a given mesh provided the size of the global Schur's complement problem is kept small compared with the average size of the subdomain problems. The solution of the global

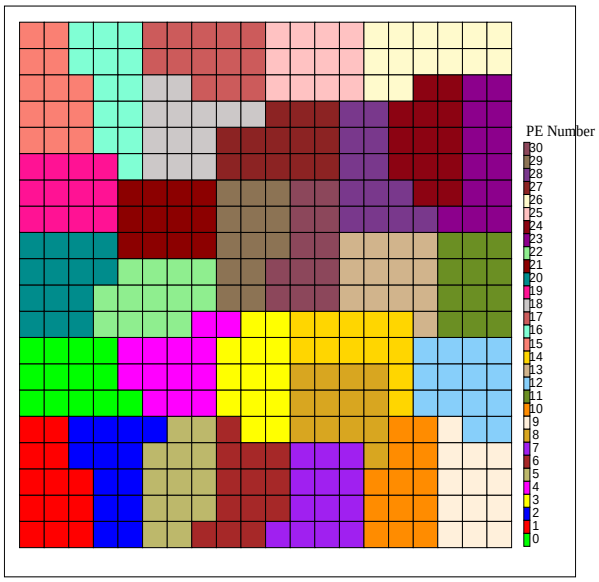

Fig. 7.11: *31-Subdomain partition of the lid-driven cavity problem.*

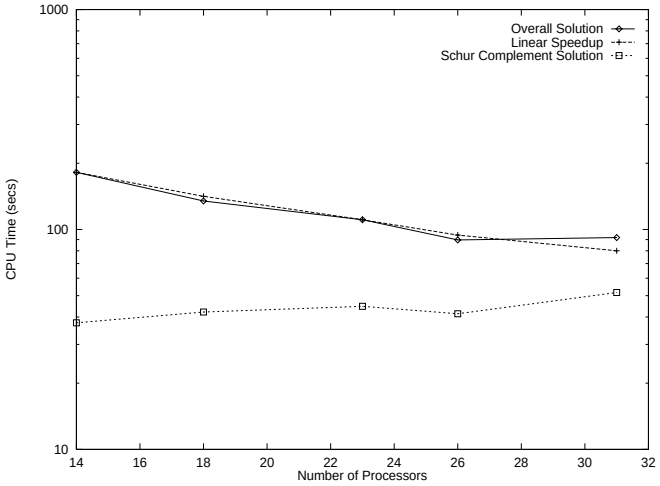

Fig. 7.12: *Performance of parallel Schur complement algorithm (p=5, lid-driven cavity).*

Schur's complement problem, a bottleneck for increasing number of subdomains can be significantly improved by partitioning the subdomain interfaces and distributing them among neighboring processors, rather than solving the entire global Schur's complement problem on a single processor.

The new algorithm is based on a graph theoretical representation of the subdomain interfaces. The separators of the global interface graph are identified, and a list of subdomains per global interface partition is generated. Only neighboring subdomains are merged onto a partition so that local neighbor-to-neighbor communication is required. Once the common vertices of subdomain interfaces are eliminated, one is left with a *reduced* Schur's complement problem defined over only a fraction of the original interface unknowns. The procedure can be continued recursively until the final reduced problem is sufficiently small. This final very small system can then be assembled onto a single processor and solved. The performance of the algorithm in terms of total CPU time closely follows the linear speed-up curve for up to 32 processors for the test problems considered. This is a significant improvement over the previous parallel direct-solution algorithms based on domain decomposition.

Acknowledgments

This research has been supported in part by NASA (contract NCCSS-154) and ASCI (contract B347883).

A Hierarchic basis functions

In the following, we briefly discuss the hierarchic polynomial basis functions on quadrilaterals used in the present work. For further details, see Szabo and Babuška (1991). These may be generated as tensor products of one-dimensional families of polynomials. Such 1D hierarchic polynomials for $\xi \in [-1, 1]$ can be defined as

$$\begin{aligned} \mathcal{N}_{(1)}(\xi) &= \frac{1}{2}(1-\xi) \\ \mathcal{N}_{(3)}(\xi) &= \frac{1}{2}(1+\xi) \\ \mathcal{N}^{(i)}{}_{(2)}(\xi) &= \phi_i(\xi) \end{aligned} \tag{A.1}$$

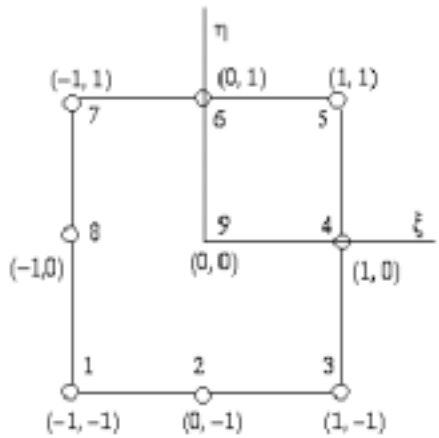

Fig. A.1: *2D hierarchic basis functions.*

where the functions ϕ_i are polynomials of degree i and in the case of Lagrange polynomials as

$$\phi_i(\xi) \;=\; \left(\xi^i - s\right) \qquad i \geq 2 \tag{A.2}$$

where $s = 1$ for i even and $s = \xi$ for i odd. Similarly, for Legendre polynomials

$$\phi_i(\xi) = \frac{1}{\sqrt{2(2i-1)}}\left(P_i(\xi) - P_{i-2}(\xi)\right) \tag{A.3}$$

where the Legendre polynomials $P_i(\xi)$ are computed using the recursion

$$(n+1)P_{n+1}(\xi) = (2n+1)\xi P_n(\xi) - nP_{n-1}(\xi) \tag{A.4}$$

Taking the tensor product, the basis functions for the reference square can be grouped as follows (see Figure A.1)

<u>Vertex Basis Functions</u>

$$\begin{aligned}
\mathcal{N}_1(\xi,\eta) &= \frac{1}{4}(1-\xi)(1-\eta) \\
\mathcal{N}_3(\xi,\eta) &= \frac{1}{4}(1+\xi)(1-\eta) \\
\mathcal{N}_5(\xi,\eta) &= \frac{1}{4}(1+\xi)(1+\eta) \qquad (A.5) \\
\mathcal{N}_7(\xi,\eta) &= \frac{1}{4}(1-\xi)(1+\eta)
\end{aligned}$$

(A.6)

Edge Basis Functions

$$\begin{aligned}
\mathcal{N}_2(\xi,\eta) &= \frac{1}{2}\phi_i(\xi)(1-\eta), \quad i = 2,3,\cdots,p \\
\mathcal{N}_4(\xi,\eta) &= \frac{1}{2}(1+\xi)\phi_i(\eta), \quad i = 2,3,\cdots,p \\
\mathcal{N}_6(\xi,\eta) &= \frac{1}{2}\phi_i(\xi)(1+\eta), \quad i = 2,3,\cdots,p \qquad \text{(A.7)} \\
\mathcal{N}_8(\xi,\eta) &= \frac{1}{2}(1-\xi)\phi_i(\eta), \quad i = 2,3,\cdots,p
\end{aligned}$$

(A.8)

Interior Basis Functions

$$\mathcal{N}_9(\xi,\eta) = \phi_i(\xi)\phi_j(\eta), \quad i,j = 2,3,\cdots,p \qquad \text{(A.9)}$$

In our adaptive p methodology, the polynomial degree may differ on adjacent elements. The polynomial degree p on an element edge is taken to be the smaller p-value of the two adjoining elements sharing that edge. In the hierarchic framework, C^0 continuity of field variables across element edges follows easily since the degree can be constrained by simply setting the corresponding higher-order derivatives as degrees of freedom on an edge to zero. This can be enforced by assembling only those edge hierarchic basis functions that correspond to polynomial orders less than or equal to the specified degree p. This results in a smaller linear system of equations.

References

de Almeida, V. F., Chapman, A. M., and Derby, J. J. **1999** On equilibration and sparse factorization of matrices arising in finite element solutions of partial differential equations. *Numer. Methods Partial Different. Equ.* **0** 0–0. In press.

Ashcraft, C., Pierce, D., Wah, D. K., and Wu, J. **1999** Spooles 2.2: An object oriented software library for solving sparse linear systems of equations. Reference manual, Boeing Shared Services Group, Seattle, WA 98124, U.S.A. Available at URL: `http://www.netlib.org/linalg/spooles/`.

Babuška, I., Elman, H. C., and Markley, K. **1992** Parallel implementation of the hp-version of the finite element method on a shared-memory architecture. *SIAM J. Sci. Stat. Comput.* **13** 1433–59.

Barragy, E. and Carey, G. F. **1991** Preconditioners for high degree elements. *Comput. Meths. Appl. Mech. Engrg.* **93** 97–110.

Barragy, E., Carey, G. F., and de Geijn, R. V.

Guo, . Z. and Myers, K. W. **1996** A parallel solver for the hp-version of finite element methods. *Comput. Meths. Appl. Mech. Engrg.* **133** 229–46.

Karypis, G. and Kumar, V. **1995** A fast and high quality multilevel scheme for partitioning irregular graphs. Research Report 95-035, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455-0132, U.S.A. Available via WWW at URL: `http://www.cs.umn.edu/~karypis`.

Keunings, R. **1995** Parallel finite element algorithms applied to computational rheology. *Comput. Chem. Engrg.* **19** 647–69.

Li, . S., Demmel, J., and Gilbert, J. **1999** On some pivotal strategies in gaussian elimination by sparse technique. *SIAM J. Matrix Anal. Appl.* Submitted.

Mandel, J. **1990** Iterative solvers by substructuring for the p-version finite element method. *Comput. Meths. Appl. Mech. Engrg.* **80** 117–28.

Mandel, J. **1991** Domain decomposition preconditioning for p-version finite elements with high aspect ratios. *Appl. Num. Meth.* **8** 441–47.

Scott, J. A. **1995** Element resequensing for the use with a multiple front algorithm. Research Report RAL 95-027, Computing and Information Systems Department, Oxon OX11 0QX, U.K.

Sloan, S. W. **1989** A FORTRAN program for profile and wavefront reduction. *Int. J. Num. Meth. Eng.* **28** 2651–79.

Szabo, B. and Babuška, I. **1991** *Finite Element Analysis*. John Wiley and Sons, New York, New York.

Zlatev, Z. **1980** On some pivotal strategies in gaussian elimination by sparse technique. *SIAM J. Numer. Anal.* **17** 18–30.