INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2003; **56**:1569–1592 (DOI: 10.1002/nme.627)

A class of parallel multiple-front algorithms on subdomains

A. Bose[1], G. F. Carey[2,*,†] and V. F. de Almeida[3]

[1]*EECS, The University of Michigan, Ann Arbor, MI 48109, U.S.A.*
[2]*TICAM, The University of Texas at Austin, Austin, TX 78712-1085, U.S.A.*
[3]*Oak Ridge National Laboratory, Oak Ridge, TN 37831-6181, U.S.A.*

SUMMARY

A class of parallel multiple-front solution algorithms is developed for solving linear systems arising from discretization of boundary value problems 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 implemented on distributed-memory multiprocessor systems with the final reduced (small) Schur complement problem solved on a single processor. A novel algorithm that implements a recursive partitioning approach on the subdomain interfaces is then developed. Both algorithms are implemented and compared in a least-squares finite-element scheme for viscous incompressible flow computation using h- and p-finite element schemes. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: solvers; parallel; multi-front; sub-domains

1. INTRODUCTION

In analysis applications using 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 [1–6]. 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. However, partial elimination on subdomains can also pose significant difficulties. Gunzburger and Nicolaides [7] 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

*Correspondence to: G. F. Carey, TICAM, The University of Texas at Austin, Austin, Texas 78712-1085, U.S.A.
†E-mail: carey@cfdlab.ae.utexas.edu

Contract/grant sponsor: NASA; contract/grant number: NCCSS-154
Contract/grant sponsor: ASCI; contract/grant number: B347883

Received 30 March 2000
Revised 7 March 2002
Accepted 24 June 2002

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 non-singular. 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.

In the present work, we first adopted the multiple-front approach [8] with subdomain and element reordering to address some of the above difficulties. We then added an intermediate step of load balancing via weighted vertices based on the p-levels of the element basis functions, to the existing capabilities of Metis [9] to achieve our subdomain partitioning for parallel elimination. An element-by-element frontal LU factorization with threshold pivoting [10] is then applied to element matrices of each subdomain. This results in some of the internal unknowns in each subdomain remaining in the equations yet to be eliminated and therefore they become part of the resulting Schur complement problem. For non-element problems, Mallya *et al.* [11] also addressed the problem of pivoting within each block. Solution of the entire Schur complement problem on one processor presumes that the reduced problem is small and hence, only allows coarse granularity of the present algorithm. Recently, Scott [12] introduced a message passing interface (MPI)-based multiple-front solution method (part of the Harwell Subroutine Library or HSL) in which the reduced Schur complement problem from subdomains are solved on a single host via a frontal algorithm. The host performs a reordering of these matrices (similar to element reordering schemes) before using the HSL module MA42 (the original frontal solver) for frontal solution of the interface problem. The multiple-front algorithm in Reference [12] is applied to finite-element meshes. In a modified algorithm which we consider the major contribution of the present work, we introduce recursive partitioning of the interface and consequently, the global Schur 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. The key contribution concerning our approach for recursively partitioning the interface is then described in detail via an illustrative example. Supporting numerical experiments for p-finite-element solution of viscous flow problems and parallel performance studies conclude the treatment.

2. PARALLEL MULTIPLE-FRONT ALGORITHM

In this section, we briefly describe the multiple-front algorithm with partial pivoting as a motivation of our present work. More information on multiple-front algorithms can be found in References [8, 13]. 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 $\Omega_i \cap \Omega_j = 0$ and common interfaces $\overline{\Omega_i} \cap \overline{\Omega_j} = \Gamma_{ij}$ for $1 \leqslant i, j \leqslant k$. Using this decomposition, the discretized problem can be expressed as a partitioned algebraic system with the partitioning defined by the subdomains. To fix the main ideas related to the subdomain interface treatment, let us consider the part of the algebraic linear system corresponding to a subdomain Ω_s written as

$$\mathbf{J}\mathbf{u} = \mathbf{r} + \boldsymbol{\sigma} \tag{1}$$

Here $\mathbf{J}$ is the assembled Jacobian matrix from individual elemental matrices of the subdomain, $\mathbf{r}$ is likewise the assembled residual for the subdomain Ω_s 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 $\tilde{\mathbf{u}} = \mathbf{Q^T u}$, 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 (1) can be reordered to $\tilde{\mathbf{r}}$ with $\tilde{\mathbf{r}} = \mathbf{P^T r}$, where $\mathbf{P^T}$ denotes the effect of row interchanges. (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 (1) becomes

$$\tilde{\mathbf{J}}\tilde{\mathbf{u}} = \tilde{\mathbf{r}} + \tilde{\boldsymbol{\sigma}} \tag{2}$$

where $\tilde{\boldsymbol{\sigma}}$ has zeros in those entries corresponding to all interior degrees of freedom and $\tilde{\mathbf{J}} = \mathbf{P^T J Q}$. The elimination process induces a corresponding block partitioning

$$\begin{aligned} \mathbf{J_{ee}u_e} + \mathbf{J_{ef}u_f} &= \mathbf{r_e} \\ \mathbf{J_{fe}u_e} + \mathbf{J_{ff}u_f} &= \mathbf{r_f} + \boldsymbol{\sigma_f} \end{aligned} \tag{3}$$

where the non-zero entries in $\boldsymbol{\sigma_f}$ correspond to all unknowns on the interface.

Formally, the first equation in (3) can be used to express $\mathbf{u_e}$ in terms of $\mathbf{u_f}$ so that the remaining equation in (3) reduces to the Schur complement problem for the subdomain interface

$$\mathbf{S_{ff}u_f} = \mathbf{b_f} \tag{4}$$

with $\mathbf{S_{ff}} = \mathbf{J_{ff}} - \mathbf{J_{fe}J_{ee}^{-1}J_{ef}}$, $\mathbf{b_f} = \mathbf{r_f} + \boldsymbol{\sigma_f} - \mathbf{J_{fe}J_{ee}^{-1}r_e}$. 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 $\boldsymbol{\sigma_f}$ are not known. However, the associated operations can be conveniently expressed for our purposes in terms of the block factorization

$$\begin{bmatrix} \mathbf{J_{ee}} & \mathbf{J_{ef}} \\ \mathbf{J_{fe}} & \mathbf{J_{ff}} \end{bmatrix} = \begin{bmatrix} \mathbf{L_{ee}} & \mathbf{0} \\ \mathbf{L_{fe}} & \mathbf{S_{ff}} \end{bmatrix} \begin{bmatrix} \mathbf{U_{ee}} & \mathbf{U_{ef}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{5}$$

and the resulting problem for (3) can be rewritten as the pair of systems

$$\begin{bmatrix} \mathbf{L_{ee}} & \mathbf{0} \\ \mathbf{L_{fe}} & \mathbf{S_{ff}} \end{bmatrix} \begin{bmatrix} \mathbf{y_e} \\ \mathbf{y_f} \end{bmatrix} = \begin{bmatrix} \mathbf{r_e} \\ \mathbf{r_f} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\sigma_f} \end{bmatrix} \tag{6}$$

and

$$\begin{bmatrix} \mathbf{U_{ee}} & \mathbf{U_{ef}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{u_e} \\ \mathbf{u_f} \end{bmatrix} = \begin{bmatrix} \mathbf{y_e} \\ \mathbf{y_f} \end{bmatrix} \tag{7}$$

Next, 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 $\mathbf{B}_\mathbf{s}^\mathbf{T}\mathbf{r_s}$ and $\mathbf{B}_\mathbf{s}^\mathbf{T}\boldsymbol{\sigma}_\mathbf{s}$. Then the assembled global system for $\mathbf{u_G}$ can be expressed as the sum of mapped subdomain contributions

$$\sum_{s=1}^{S}(\mathbf{B}_\mathbf{s}^\mathbf{T}\mathbf{J}_\mathbf{s}\mathbf{B}_\mathbf{s})\mathbf{u_G} = \sum_{s=1}^{S}\mathbf{B}_\mathbf{s}^\mathbf{T}\mathbf{r_s} + \sum_{s=1}^{S}\mathbf{B}_\mathbf{s}^\mathbf{T}\boldsymbol{\sigma}_\mathbf{s} \tag{8}$$

The last term in (8) reduces to the global sum of interface flux jumps, and

$$\sum_{s=1}^{S}\mathbf{B}_\mathbf{s}^\mathbf{T}\boldsymbol{\sigma}_\mathbf{s} = \sum_{i=1}^{I}[\boldsymbol{\sigma}_\mathbf{i}] = \mathbf{0} \tag{9}$$

follows from the weak variational statement in the Galerkin method and from $\mathbf{C^0}$ approximation of the fluxes in the least-squares method in Section 4.

Hence, we need not explicitly treat $\boldsymbol{\sigma}_\mathbf{f}$ and 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 $\mathbf{S_{ff}}$ are then generated in parallel across subdomains 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 element basis (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 subdomain factorization step.

For clarity of exposition, let us first assume that the global Schur complement problem ($\mathbf{S_F g_F} = \mathbf{b_F}$) has been assembled from subdomain contributions onto a single processor (e.g. processor zero) and solved to determine the global vector $\mathbf{g_F}$. From $\mathbf{g_F}$ we can extract the subdomain front solution vectors $\{\mathbf{u}_\mathbf{f}^\mathbf{s}\}$, $s = 1, 2, \ldots, S$ by use of a parallel subdomain back-substitution step in (7).

For a given LU-decomposition of the subdomain Jacobian matrix $\tilde{\mathbf{J}}$, we can rewrite $\mathbf{b_f}$ from (4) and (6) as

$$\mathbf{b_f} = \mathbf{r_f} + \boldsymbol{\sigma}_\mathbf{f} - \mathbf{J_{fe}}\mathbf{J_{ee}^{-1}}\mathbf{r_e} = \mathbf{r_f} + \boldsymbol{\sigma}_\mathbf{f} - \mathbf{L_{fe}}\mathbf{y_e} \tag{10}$$

to obtain a forward substitution step for calculation of the Schur complement vector $\mathbf{b_f}$. Since the 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 complement matrix $\mathbf{S_{ff}}$ may be sent to processor zero via a non-blocking 'send' call while the computation for vector $\mathbf{b_f}$ is in progress:

Algorithm (parallel multiple-front algorithm)
for s = 0 **to** S subdomains in parallel
if s = 0 **then** [processor **zero**]
  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.
  Receive $\mathbf{S_{ff}}$ and $\mathbf{b_f}$ from processors $\{1, 2, \ldots, \mathrm{S}\}$ and assemble the global Schur complement problem.
  Solve the global assembled Schur 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 complement matrix $\mathbf{S_{ff}}$ to processor **zero** using a **non-blocking** 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 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 complement solution step]
Solve for internal variables $\mathbf{u_e}$.
endif

endfor

Remarks
As shown above, processor zero can be used for one of the subdomain problems as well as for solving the assembled Schur complement problem. At the time of writing the software for this algorithm, the authors were limited to the small memory size of a Cray T3E system. Both the subdomain and the assembled Schur complement problems could not be allocated on processor zero. As a result, the implementation assumed a number of subdomains one less than the number of requested processors, and processor zero was set aside to solve the assembled Schur complement problem only. Furthermore, for some of the example problems, solving the entire problem on a single processor was not possible due to memory constraints. For other distributed-shared memory systems this is not a problem.

The only significant serial part of the above algorithm is the final solution of the global reduced Schur 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 (see Reference [8]); and the size of the assembled Schur complement problem increases accordingly [14]. This algorithm is therefore best suited for coarse grain parallelism. Of course, one can also solve the resulting global Schur complement problem in parallel using a parallel dense direct or iterative solution algorithm. A more interesting approach for very large problems is to continue the parallel partitioning idea to the interface as described next.

3. INTERFACE TREATMENT

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 $\tilde{V}: \tilde{V} \subset V$, $\tilde{E}=\{(v_1,v_2)\in E \parallel v_1 \in \tilde{V} \textit{ and } v_2 \in \tilde{V}\}$. A graph is *connected* if every pair of vertices is connected by a path. Otherwise, the graph is disconnected. Usually a disconnected

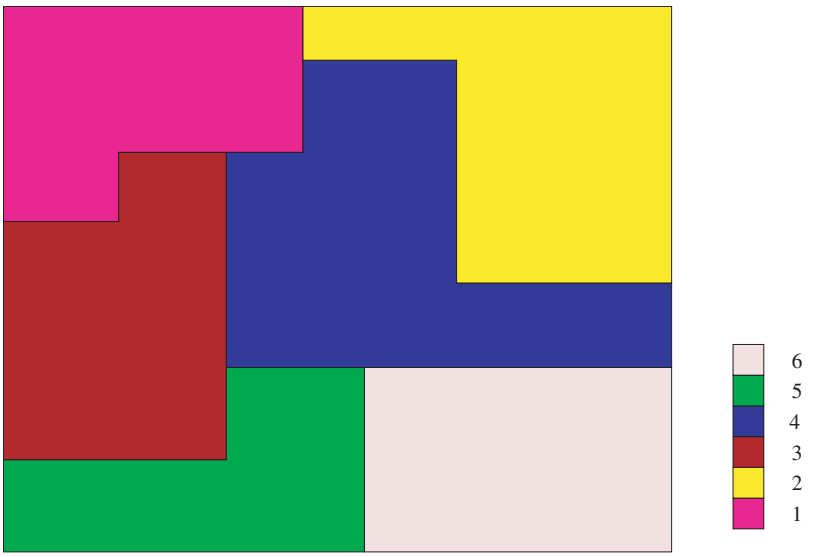

Figure 1. Partitioning of a finite element mesh to six subdomains.

graph consists of two or more connected components. We only consider connected graphs of finite elements for the present study. However, the proposed algorithm can be extended to the case of a disconnected graph with two or more connected components. 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 $\deg(C_i)$ of a cutvertex C_i.

3.1. Partitioning the interface

Now let us consider a partitioning of a finite-element mesh to subdomains. We wish to represent the global interfaces among the neighbouring 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, examples and implementation details to 2D domains for this study. However, the basic ideas can be extended to 3D geometries as well, although the implementation will clearly be more complicated. To illustrate the main ideas, let us consider the partitioning in Figure 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; i.e. if a connectivity graph is constructed out of all interface vertices, this node will become a separator of this graph. For example, the separators of the graph of global interfaces for the partitioning in Figure 1 are shown in Figure 2. The degree of any cutvertex can be calculated from the number of subdomains incident to it, e.g. $\deg(C_4) = 2$, $\deg(C_5) = 3$, etc. The interface segment between any two end separators consists of nodes of finite-elements incident to that interface.

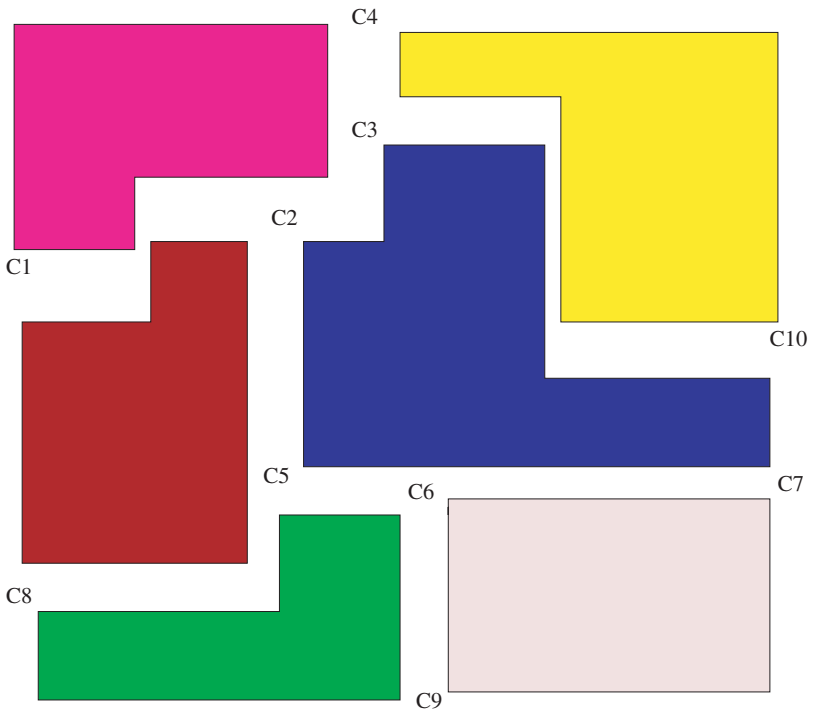

Figure 2. Separators in the connectivity graph of global interfaces.

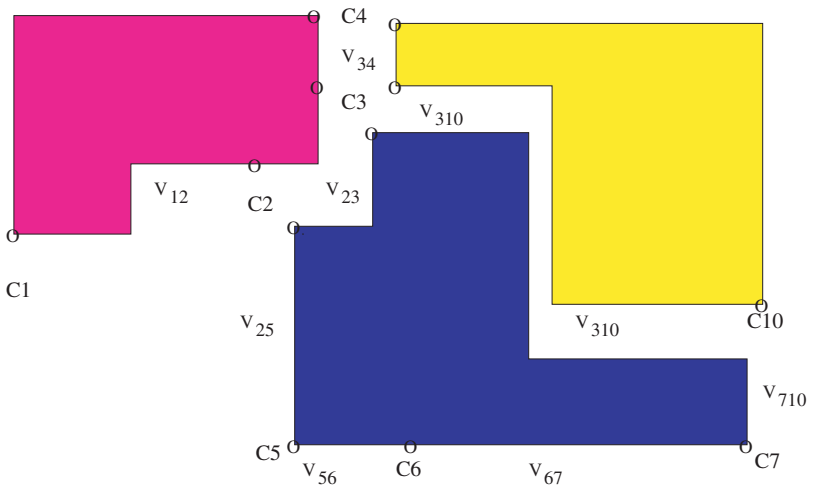

Figure 3. Vertices of interface graphs for subdomains 1, 2 and 4.

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 3 for subdomains 1, 2 and 4.

The interface partitioning strategy can then be most clearly explained using an example. Let us assume that the interfaces incident with subdomains 1, 2 and 4 in Figure 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 in 2D:

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.

We should point out that the above rules specifically apply to 2D geometries. Similar rules can be formulated for the case of 3D geometries as well. 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 finite-element mesh.

The entire set of vertices $V^{(1)}$ is non-reducible; i.e. they cannot be eliminated independently of the other partitions. Now 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 $V^{(2)}$; i.e. 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: $I = \{C_4, v_{34}\}$, $N = \{C_1, v_{12}, C_2, v_{23}, C_3, v_{310}, C_{10}\}$, 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: $I = \{C_4, v_{23}, v_{34}, C_3, v_{310}, C_{10}\}$, $N = \{C_1, v_{12}, C_2, v_{25}, C_5, v_{56}, C_6, v_{67}, C_7\}$.

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. In an ideal case it would involve only the remaining corner vertice.

The algorithm for creating lists I and N for each interface partition in a two-dimensional geometry follows:

Algorithm (Interface list construction for 2D geometries)
for $k=1$ **to** n interface partitions
  generate a list of m subdomains ($S^{(k)}$) for partition k
  for $i=1$ **to** m subdomains
    $\mathcal{N}^{(k)} \leftarrow \mathcal{N}^{(k)} \cup V^{(i)}$ (initialization)
    if $i>1$ **then** (merging of subdomains)
    find s minimal separators between sets $\mathcal{N}^{(k)}$ and $V^{(i)}$
    for $j=1$ **to** s separators
      if $\deg(C_j)=2$ then (eliminate)
        $I^{(k)} \leftarrow I^{(k)} \cup C_j \cup v_{jl}$ (jl: connected vertices)
        remove C_j and v_{jl} from $\mathcal{N}^{(k)}$
      else
        $\deg(C_j)=\deg(C_j)-1$
      endif
    endif
    endfor
  endfor
endfor

Remark
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.* [15] 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. However, in this approach the parallelism is progressively reduced and 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 well beyond a few subdomains.

Even though our presentation of the proposed algorithm addresses only 2D geometries, the most promising application of this algorithm is for the case of 3D geometries in which the size of the interface is likely to grow and the need for interface partitioning becomes almost essential. It should be noted that for the case of 3D, the interface is no longer a line. Instead, it may be represented by a parametric surface with two parameters in a local co-ordinate system. The separators in the graph of global interfaces will not be simple end-points along interfaces, rather they will be a set of vertices along the tangential directions of an interface. The degree of each of these vertices will still be the number of subdomains incident to it; and all interface vertices that are enclosed within this set of separators can be eliminated by merging subdomains. For 3D, the two rules and the algorithm for interface list construction need to be modified accordingly.

Recently, a family of linear solvers based on selective orderings have been proposed. Notably, SPOOLES [16] 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 [17] 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 of the Schur complement problem over different processor configurations for a given number of subdomains using the new algorithm described here.

In closing this section, we note that both the subdomain and interface partitioning problems can be combined into a single partitioning problem in which a given geometry is partitioned into a number of subdomains with the constraint that the resulting minimum interfaces can be merged for maximum overlap. Such a constrained partitioning problem may prove to be difficult for complex unstructured finite-element meshes. However, by using iterative and heuristic methods, locally optimal solutions for the combined partitioning problem may be possible, certainly for structured meshes.

3.2. *Merging subdomain interfaces*

The specific problem of merging adjacent subdomain interfaces and the resulting 'reduced Schur complement' problem is considered next. This section does not depend on our specific treatment of 2D geometries as shown above, and therefore is more general in treating both 2D and 3D meshes. For the time being, let us assume that only interface unknowns contribute to the Schur complement problem for any subdomain. Consider two neighbouring subdomains $\Omega_{(p)}$ and $\Omega_{(q)}$ sharing an interface segment Γ_{pq}. The subdomain Schur complement problems are

$$\begin{aligned} \mathbf{S}^{(\mathbf{p})}\mathbf{u}^{(\mathbf{p})} &= \mathbf{b}^{(\mathbf{p})} + \mathbf{r}^{(\mathbf{p})} \\ \mathbf{S}^{(\mathbf{q})}\mathbf{u}^{(\mathbf{q})} &= \mathbf{b}^{(\mathbf{q})} + \mathbf{r}^{(\mathbf{q})} \end{aligned} \tag{11}$$

corresponding to the interfaces $\partial\Omega_{(p)}$ and $\partial\Omega_{(q)}$, respectively. The interface unknowns are denoted as $\mathbf{u}^{(\mathbf{p})}$ and $\mathbf{u}^{(\mathbf{q})}$. The subdomain Schur complement matrices $\mathbf{S}^{(\mathbf{p})}$ and $\mathbf{S}^{(\mathbf{q})}$ 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 neighbouring subdomains. The contribution of the neighbouring interfaces is represented by the vectors $\mathbf{r}^{(\mathbf{p})}$ and $\mathbf{r}^{(\mathbf{q})}$ in Equations (11). 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 (11) in each subdomain. First, we introduce a local ordering of the subdomain interface nodal unknowns so that those on Γ_{pq} are numbered as $\mathbf{u}_\mathbf{e}$. The remaining unknowns on each subdomain interface are labelled as $\mathbf{u}_\mathbf{f}^{(\mathbf{p})}$ and $\mathbf{u}_\mathbf{f}^{(\mathbf{q})}$, respectively. In the absence of any other subdomain interfaces, the lists I and $\mathcal{N}$ can then be expressed as $I = \{\mathbf{u}_\mathbf{e}\}$, $\mathcal{N} = \{\mathbf{u}_\mathbf{f}^{(\mathbf{p})}, \mathbf{u}_\mathbf{f}^{(\mathbf{q})}\}$.

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

$$\begin{aligned}(\mathbf{u_e}, \mathbf{u_f^{(p)}})^{\mathrm{T}} &= \mathbf{Q}_{\mathbf{(p)}}^{\mathbf{T}} \mathbf{u}^{(p)} \\ (\mathbf{u_e}, \mathbf{u_f^{(q)}})^{\mathrm{T}} &= \mathbf{Q}_{\mathbf{(q)}}^{\mathbf{T}} \mathbf{u}^{(\mathbf{q})}\end{aligned} \tag{12}$$

respectively, for the two subdomains. Under these reorderings, Equations (11) become

$$\begin{aligned}\tilde{\mathbf{S}}^{(p)} \tilde{\mathbf{u}}^{(p)} &= \tilde{\mathbf{b}}^{(p)} + \tilde{\mathbf{r}}^{(p)} \\ \tilde{\mathbf{S}}^{(q)} \tilde{\mathbf{u}}^{(q)} &= \tilde{\mathbf{b}}^{(q)} + \tilde{\mathbf{r}}^{(q)}\end{aligned} \tag{13}$$

respectively, for subdomains p and q where

$$\tilde{\mathbf{S}}^{(p)} = \mathbf{Q}_{\mathbf{(p)}}^{\mathbf{T}} \mathbf{S}^{\mathbf{(p)}} \mathbf{Q}_{\mathbf{(p)}}, \quad \tilde{\mathbf{b}}^{(p)} = \mathbf{Q}_{\mathbf{(p)}}^{\mathbf{T}} \mathbf{b}^{\mathbf{(p)}}, \quad \tilde{\mathbf{r}}^{(p)} = \mathbf{Q}_{\mathbf{(p)}}^{\mathbf{T}} \mathbf{r}^{\mathbf{(p)}} \tag{14}$$

and

$$\tilde{\mathbf{S}}^{(q)} = \mathbf{Q}_{\mathbf{(q)}}^{\mathbf{T}} \mathbf{S}^{\mathbf{(q)}} \mathbf{Q}_{\mathbf{(q)}}, \quad \tilde{\mathbf{b}}^{(q)} = \mathbf{Q}_{\mathbf{(q)}}^{\mathbf{T}} \mathbf{b}^{\mathbf{(q)}}, \quad \tilde{\mathbf{r}}^{(q)} = \mathbf{Q}_{\mathbf{(q)}}^{\mathbf{T}} \mathbf{r}^{\mathbf{(q)}} \tag{15}$$

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

$$\tilde{\mathbf{S}}^{(p)} = \begin{bmatrix} \mathbf{S}_{\mathbf{ee}}^{\mathbf{(p)}} & \mathbf{S}_{\mathbf{ef}}^{\mathbf{(p)}} \\ \mathbf{S}_{\mathbf{fe}}^{\mathbf{(p)}} & \mathbf{S}_{\mathbf{ff}}^{\mathbf{(p)}} \end{bmatrix} \tag{16}$$

and

$$\tilde{\mathbf{S}}^{(q)} = \begin{bmatrix} \mathbf{S}_{\mathbf{ee}}^{\mathbf{(q)}} & \mathbf{S}_{\mathbf{ef}}^{\mathbf{(q)}} \\ \mathbf{S}_{\mathbf{fe}}^{\mathbf{(q)}} & \mathbf{S}_{\mathbf{ff}}^{\mathbf{(q)}} \end{bmatrix} \tag{17}$$

Note that the matrices $\mathbf{S}_{\mathbf{ee}}^{\mathbf{(p)}}$ and $\mathbf{S}_{\mathbf{ee}}^{\mathbf{(q)}}$ 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, i.e. we compute $\mathbf{S}_{\mathbf{ee}} = \mathbf{S}_{\mathbf{ee}}^{\mathbf{(p)}} + \mathbf{S}_{\mathbf{ee}}^{\mathbf{(q)}}$. This step requires inter-processor 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

$$\begin{bmatrix} \mathbf{S_{ee}} & \mathbf{S}_{\mathbf{e}\phi} \\ \mathbf{S}_{\phi\mathbf{e}} & \mathbf{S}_{\phi\phi} \end{bmatrix} \begin{bmatrix} \mathbf{u_e} \\ \mathbf{u}_\phi \end{bmatrix} = \begin{bmatrix} \mathbf{b_e} \\ \mathbf{b}_\phi \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{r}_\phi \end{bmatrix} \tag{18}$$

where the blocks $\mathbf{S}_{\mathbf{e}\phi}$, $\mathbf{S}_{\phi\mathbf{e}}$ 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 neighbouring subdomains. Finally, the unknowns

$\mathbf{u}_e$ can be eliminated from (18) and the 'reduced' Schur complement problem for the 'merged' subdomain interfaces can be expressed as

$$[\mathbf{S}_{\phi\phi} - \mathbf{S}_{e\phi}\mathbf{S}_{ee}^{-1}\mathbf{S}_{\phi e}]\mathbf{u}_\phi = \mathbf{b}_\phi + \mathbf{r}_\phi - \mathbf{S}_{e\phi}\mathbf{S}_{ee}^{-1}\mathbf{b}_e \tag{19}$$

The subdomain interfaces can be partitioned into a number of global interface segments by merging neighbouring subdomains. Each global interface partition generates a 'reduced' Schur complement problem as given in (19) 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.

3.3. Implementation

The schemes have 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 [9] 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 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 elements [18] to reduce the front width of subdomain frontal factorization; generate send/receive lists of local nodes for each neighbouring 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.

Communication among processors is handled using the standard MPI [19] protocols; e.g. 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 neighbouring processor numbers (PE's) for each processor in the system and local interface node numbers shared with each neighbour. 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. A novel memory management model is used in Pfinics, resulting in 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 Reference [20]. Although the object-oriented framework of Pfinics is general enough to accommodate 2D and 3D elements, the code has only been implemented for solving 2D geometries so far, and therefore, only 2D flow problems are presented in the section for numerical examples.

4. 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

$$\nabla \cdot \mathbf{v} = 0 \tag{20}$$

$$\mathbf{v} \cdot \nabla \mathbf{v} = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \frac{\mathbf{g}}{W_g} \tag{21}$$

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 $\mathbf{x} \in \Omega$ and W_g $(= v_0{}^2/g_o l)$ is an inertial-buoyancy flow parameter. The constitutive equation for a Newtonian fluid is taken in the form

$$\boldsymbol{\tau} = \frac{1}{Re}(\nabla \mathbf{v} + \nabla \mathbf{v}^{\mathrm{T}}) \tag{22}$$

for $Re = \rho v_o l/\mu$ the Reynolds number. The parameters ρ, μ, v_0 and l are density, viscosity, a characteristic velocity and length scale, 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: $\mathbf{v} = \mathbf{v}_0$ on $\partial\Omega_d$, $\tau \cdot \mathbf{n} = \mathbf{t_0}$ on $\partial\Omega_n$.

Here we employ a least-squares finite-element formulation for Equations (20)–(22) 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 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 Equations (20)–(22) as follows:

$$\mathscr{F} = \|\mathbf{R_f}\|^2_{L^2(\Omega)} \tag{23}$$

where the residuals $\mathbf{R}_f$ $(Re \geqslant O(1))$ for admissible approximation fields $\mathbf{v}^h$, p^h, $\boldsymbol{\tau}^h$ in the finite-element spaces are simply

$$\mathbf{R}_f = \begin{bmatrix} \nabla \cdot \mathbf{v}^h \\ \mathbf{v}^h \cdot \nabla \mathbf{v}^h + \nabla p^h - \nabla \cdot \boldsymbol{\tau}^h - \dfrac{\mathbf{g}}{\mathbf{W_g}} \\ \boldsymbol{\tau}^h - \dfrac{1}{Re}\left(\nabla \mathbf{v}^h + \nabla \mathbf{v}^{h^T}\right) \end{bmatrix} \tag{24}$$

Taking variations leads to a non-linear 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 $\mathbf{v}_h = 0$ occurs for the linear velocity, constant-pressure triangle and spurious pressure modes can occur for other choices of bases [21]. These restrictions on the bases do not apply to the least-squares mixed finite-element formulation.) Consequently, in the results presented later we use $\mathbf{C}^o$ equal order bases for all variables for convenience (although other choices are possible and may be preferable).

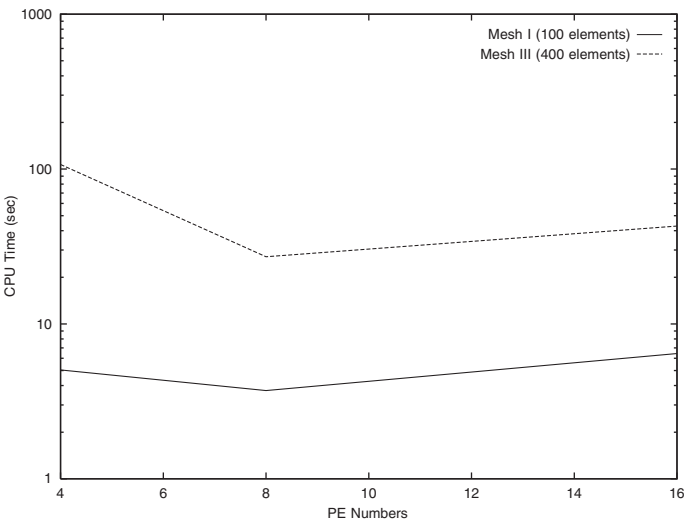

Figure 4. CPU time for Meshes I and III with quadratic basis functions for 4, 8, 16 processors.

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 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 show results for two different uniform discretizations, namely 10×10 and 20×20 grids denoted as Meshes I and III, respectively, in all subsequent discussion. Other experiments on an intermediate mesh (Mesh II) were also carried out but the results are similar and hence are not included here. The respective meshes are partitioned into 3 and 15 subdomains. Since we use processor zero in the communication group for solving the assembled global Schur complement problem, we therefore run the above cases on 4 and 16 processors, respectively. We first present timing results using the single-processor Schur complement solution algorithm. Figure 4 shows timing results on 4, 8 and 16 processors of the T3E for the two meshes with quadratic basis functions ($p=2$). The base parallel algorithm does well for up to 8 processors for these two cases but does not scale beyond 8 processors. In Table I, we see that for 16 processors, the size of the assembled global Schur 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 complement problem serially. Clearly, for good scaling with increasing number of processors, the size of the global Schur 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 complement problem. This results in better timing results as seen in Figure 5 which compares CPU times for Mesh I with polynomial orders of 2 and 7, respectively.

Table I. Sizes of global Schur complement and subdomain problems (in brackets) for Meshes I and III ($p = 2$).

	Number of processors (PEs)		
	PEs = 4	PEs = 8	PEs = 16
Mesh I	234 (978)	480 (474)	834 (282)
Mesh III	462 (822)	918 (1614)	1578 (822)

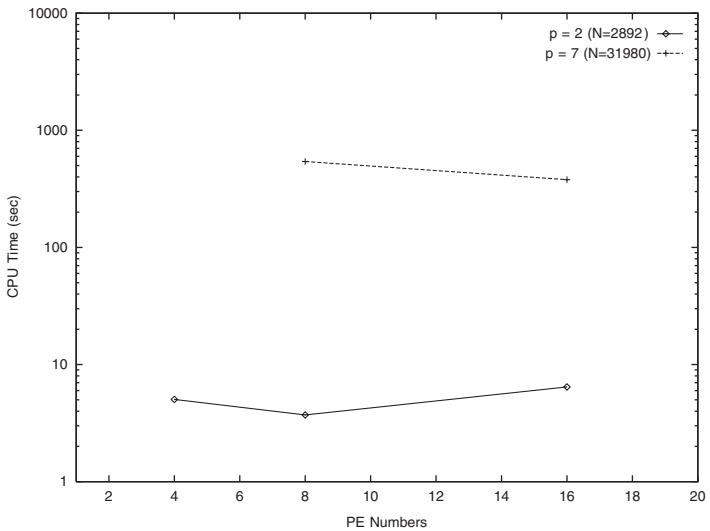

Figure 5. CPU time for mesh I ($p = 2$ and 7).

The effect of increasing problem size on a fixed number of processors is shown in Figure 6. 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 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, 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 complement problem takes more time as the number of processors is increased. This bottleneck can be significantly reduced by the algorithm for parallel solution of the interface problem as demonstrated later.

As the polynomial degree of the basis is raised, matrix conditioning deteriorates and the rate of convergence of iterative solvers is adversely affected [22]. Matrix conditioning in

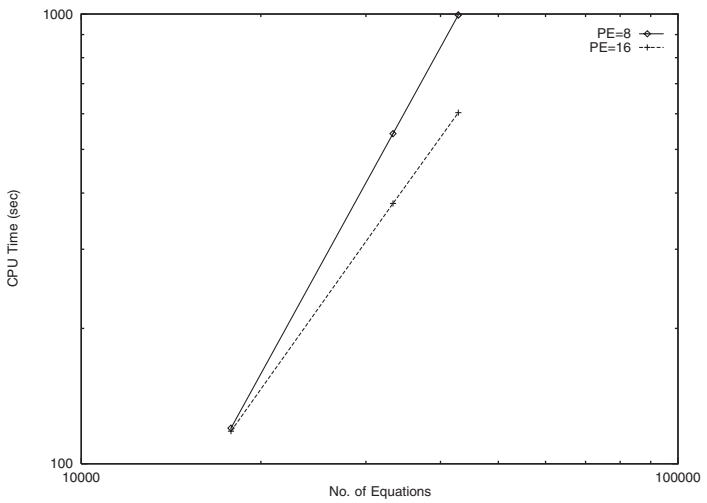

Figure 6. Effect of increasing polynomial orders (Mesh I) on 8 and 16 processors.

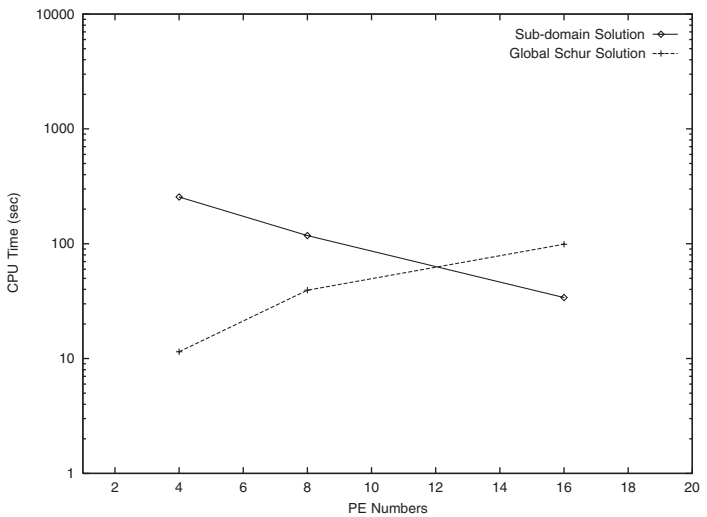

Figure 7. Frontal solver performance for mesh I ($p = 5$, 17 820 equations).

direct solvers can also be a problem; e.g. 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 submesh 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 (details in Reference [23]).

Table II. Schur complement problem size for Mesh I on 8 processors, percentage shown in terms of subdomain size (Lagrange polynomials).

	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 III. Schur complement problem size for Mesh I on 8 processors, percentage shown in terms of subdomain size (Legendre polynomials).

	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 2.4%	756 16.8%	426 9.4%

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. e.g. 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.

The data in Table II 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 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 II and III show the same number of Schur 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 complement problem, whereas hierarchical basis functions constructed from Legendre polynomials contribute none. Therefore, for Legendre polynomials, the Schur complement problem for each subdomain consists of only interface unknowns. This

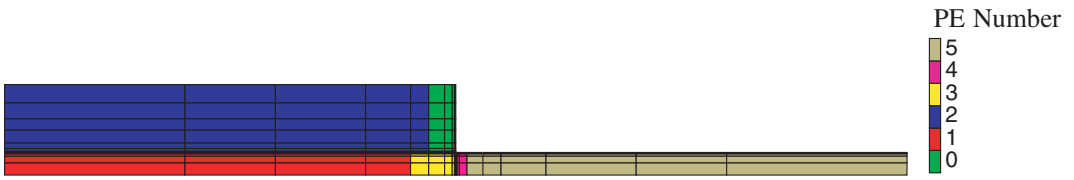

Figure 8. Six-subdomain partition of the 4:1 contraction geometry.

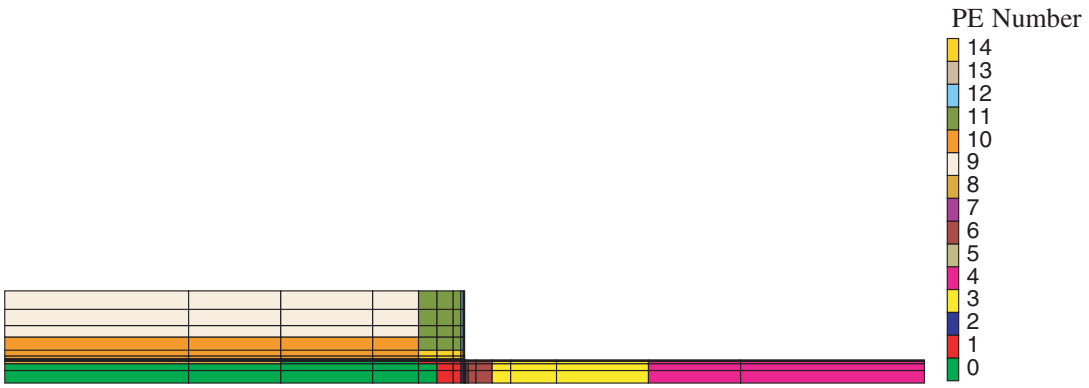

Figure 9. Fifteen-subdomain partition of the 4:1 contraction geometry.

is evident from the above table. For Lagrange polynomials, the size of the Schur complement problem grows exponentially with polynomial order, resulting in poorer efficiency of the algorithm. Moreover, the percentage of Schur complement unknowns with respect to total number of unknowns in any subdomain decreases as the order of the basis functions increases, when Legendre polynomials are used.

Before we present the performance of the distributed Schur complement solution algorithm for the driven cavity problem, let us consider another widely used numerical experiment as a second example: least-squares finite-element solution of channel flow into a planar 4:1 contraction. Our objective is to show the limitations of the current approaches for both numerical examples and compare them with our distributed Schur complement solution algorithm. We refer to Reference [24] for the finite-element formulation and related issues. Owing 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 in Reference [25]. Here we focus on the comparative performance and parallel scaling of the new recursive interface partitioning algorithm relative to the previous algorithm. Representative samples of partitionings of the original domain to subdomains are given in Figures 8 and 9.

Figure 10 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 9 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,

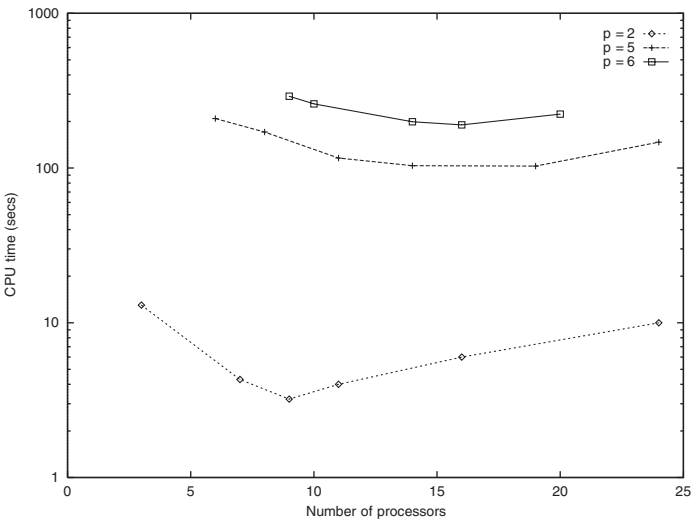

Figure 10. CPU time for different polynomial orders.

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 to 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-degree 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 11 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 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 complement solution time for the processor configurations we have tested so far. Figure 12 compares the solution time for the Schur complement problem for the two algorithms. Note that the size of the Schur 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 complement 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 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 complement solution step), the fewer the number of unknowns in the reduced Schur 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 complement problem. However, this increases the size of the reduced Schur complement problem, and indeed it may be efficient

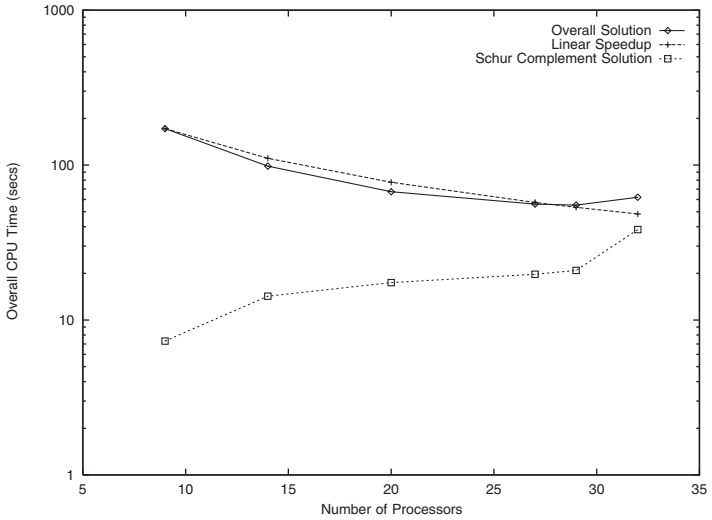

Figure 11. Performance of distributed Schur complement solution ($p = 6$, 4:1 contraction).

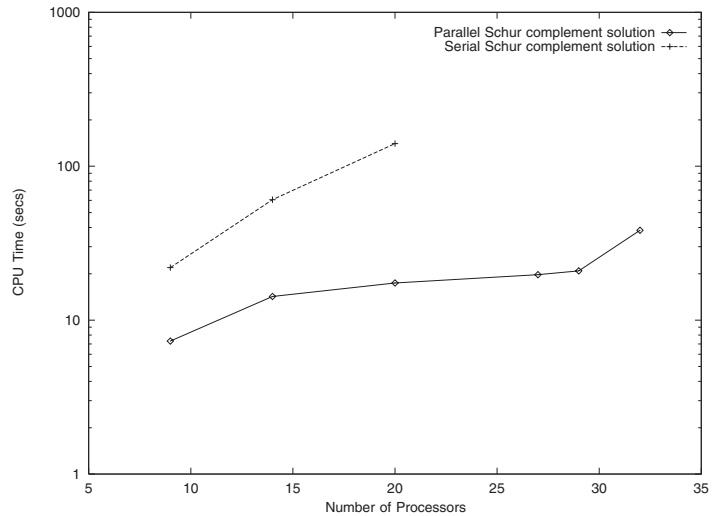

Figure 12. Serial and parallel Schur complement solution timings ($p = 6$, 4:1 contraction).

to employ just a few processors for solving the Schur complement problem as evidenced in Figure 13 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 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 complement solution step as indicated by the local minimum. It is

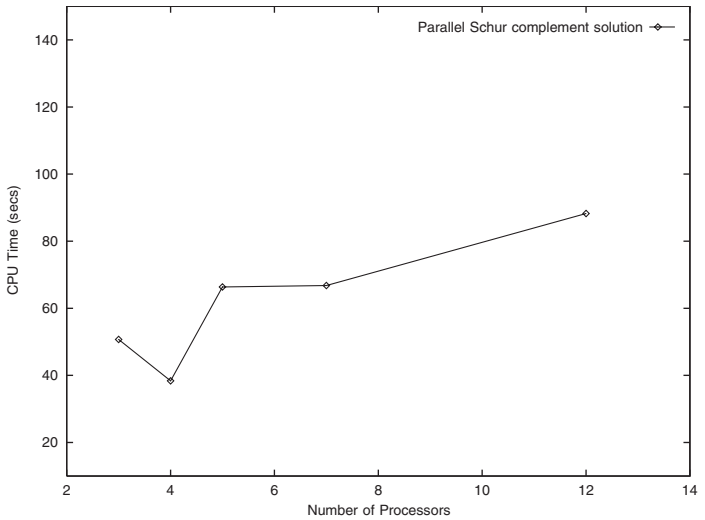

Figure 13. Performance of parallel Schur complement algorithm ($p = 6$, 4:1 contraction).

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 13 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.

Finally, the parallel Schur complement solution algorithm is applied to our first flow problem, i.e. isothermal cavity flow. 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 14. For this partitioning, we present the performance of the parallel algorithm in Figure 15 on 5 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 complement solution algorithm.

5. CONCLUSION

We have presented a multiple-front elimination algorithm based on domain decomposition for parallel direct solution of algebraic systems resulting from discretization of boundary value problems in continuum mechanics. The basic algorithm concludes with a serial Schur complement solve and therefore is suitable for coarse grain parallelism, e.g. for tightly coupled workstation clusters and low-end multi-processor systems. Its efficiency improves with p-refinement on a given mesh provided the size of the global Schur complement problem

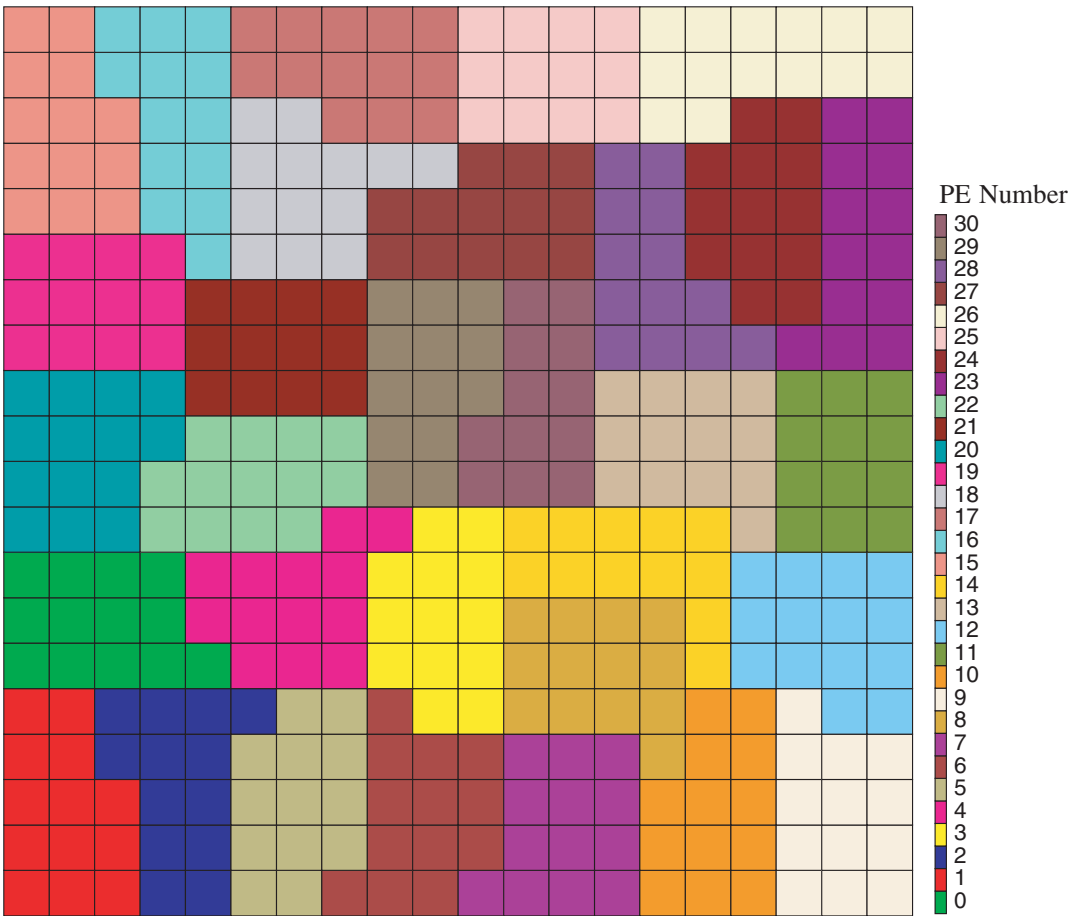

Figure 14. Thirty-one-subdomain partition of the lid driven cavity problem.

is kept small compared with the average size of the subdomain problems. The solution of this global Schur complement problem, a bottleneck for increasing number of subdomains, can be significantly improved by partitioning the subdomain interfaces and distributing them among neighbouring processors as shown in our examples. This leads to a scalable algorithm that employs 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 neighbouring subdomains are merged onto a partition so that local neighbour-to-neighbour communication is required. Once the common vertices of subdomain interfaces are eliminated, one is left with a *reduced* Schur 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.

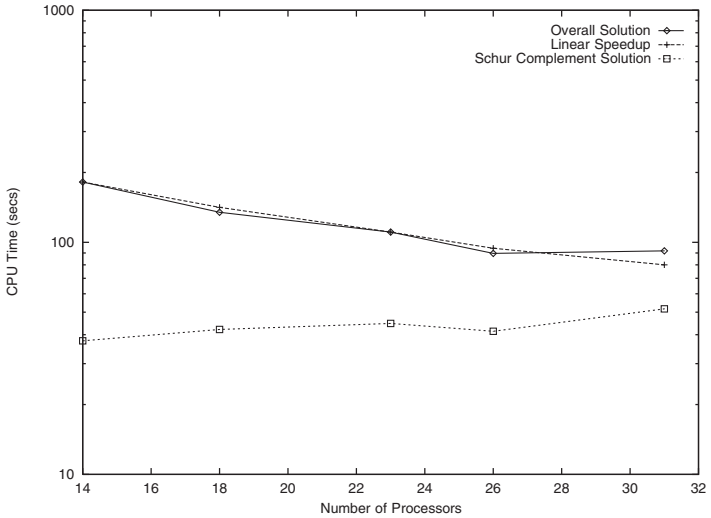

Figure 15. Performance of parallel Schur complement algorithm ($p = 5$, lid driven cavity).

ACKNOWLEDGEMENTS

This research has been supported in part by NASA (contract NCCSS-154) and ASCI (contract B347883). The authors express their appreciation to the reviewers for their constructive comments and suggestions.

REFERENCES

1. Carey GF, Barragy E. Basis function selection and preconditioning high degree finite element methods. *BIT* 1989; **29**:794–804.
2. Mandel J. Iterative solvers by substructuring for the *p*-version finite element method. *Computer Methods in Applied Mechanics and Engineering* 1990; **80**(1/3):117–128.
3. Barragy E, Carey GF. Preconditioners for high degree elements. *Computer Methods in Applied Mechanics and Engineering* 1991; **93**:97–110.
4. Mandel J, Lett GS. Domain decomposition preconditioning for *p*-version finite elements with high aspect ratios. *Applied Numerical Mathematics* 1991; **8**(4/5):411–425.
5. Babuska I, Elman HC, Markley K. Parallel implementation of the *hp*-version of the finite element method on a shared-memory architecture. *SIAM Journal on Scientific and Statistical Computing* 1992; **13**(6):1433–1459.
6. Barragy E, Carey GF, Van de Geijn R. Performance and scalability of finite element analysis for distributed parallel computation. *Journal of Parallel and Distributed Computing* 1994; **21**:202–212.
7. Gunzburger MD, Nicolaides RA. Issues in the implementation of substructuring algorithms for the Navier–Stokes equations. *Advances in Computer Methods and Partial Differential Equations*, vol. 5. IMACS: New Brunswick, NJ, 1984; 57–63.
8. Duff IS, Scott JA. The use of multiple-fronts in Gaussian elimination. *RAL Report* 94-040, 1994a.
9. Karypis G, Kumar V. *METIS*: unstructured graph partitioning and sparse matrix ordering system. *Technical Report*, Department of Computer Science, The University of Minnesota, 1995.
10. Duff IS, Scott JA. The use of multiple-fronts in Gaussian elimination. *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*. SIAM: Philadelphia, PA, 1994b; 567–571.
11. Mallya JU, Zitney SE, Chaudhary S, Stadtherr MA. Parallel frontal solver for large scale process simulation and optimization. *AIChE Journal* 1997; **43**:1032–1040.
12. Scott JA. A parallel frontal solver for finite element applications. *International Journal for Numerical Methods in Engineering* 2001; **50**(5):1131–1144.
13. Escaig Y, Vayssade M, Touzot G. Une methode de decomposition de domaines multifrontale multiniveaux. *Revue Europ. des Elem. Finis* 1994; **3**:311–337.

14. Keunings R. Parallel finite element algorithms applied to computational rheology. *Computers and Chemical Engineering* 1995; **19**(6/7):647–669.
15. Benner RE, Montry GR, Weigand GG. Concurrent multifrontal methods: shared memory, cache, and frontwidth issues. *International Journal of Supercomputer Applications* 1987; **1**(3):26–44.
16. Ashcraft C, Grimes R. SPOOLES: an object-oriented sparse matrix library. *Proceedings of the 1999 SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, Texas, 22–27 March 1999.
17. Li XS, Demmel J, Gilbert J. An asynchronous parallel supernodal algorithm for sparse Gaussian elimination. *SIAM Journal on Matrix Analysis and Applications* 1999; **20**(4):915–952.
18. Scott JA. Element resequencing for use with a multiple front algorithm. *International Journal for Numerical Methods in Engineering* 1996; **39**:3999–4020.
19. Lusk E, Skjellum A *et al. MPI: A Message-Passing Interface Standard*. MPI Forum, 21 April 1994.
20. Bose A, Carey GF. A class of data structures and object-oriented implementation for finite element methods on distributed memory systems. *Computer Methods in Applied Mechanics and Engineering* 1999, **171**(1):109–121.
21. Carey GF, Oden JT. *Finite Elements*: *Fluid Mechanics*. Prentice-Hall: USA, 1986.
22. Bramble JH, Pasciak JE, Schatz AH. The construction of preconditioners for elliptic problems by substructuring I. *Mathematics of Computation* 1986; **47**:103–134.
23. de Almeida VF, Chapman AM, Derby JJ. On equilibration and sparse factorization of matrices arising in finite element solutions of partial differential equations. *Numerical Methods for Partial Differential Equations* 2000; **16**:12–29.
24. Carey GF, Pehlivanov AF, Shen Y, Bose A, Wang KC. Least-squares finite elements for fluid flow and transport. *International Journal for Numerical Methods in Fluids* 1998; **27**(1–4):97–107.
25. Bose A. Parallel *p*-adaptive finite element methods for viscous incompressible non-Newtonian flows. *Ph.D. Dissertation*, The University of Texas at Austin, 1997.

