CHAPTER 27

Parallel Substructuring Algorithms in Structural Analysis, Direct and Iterative Methods*

Petter E. Bjørstad†
Jon Brækhus‡
Anders Hvidsten§

Abstract

This paper reviews the development that has occurred in order to adopt a large scale structural analysis package, SESAM[1], to the rapid changes in computer architecture as well as to the algorithmic advances that has been made in the past few years. We describe a parallel implementation of the sophisticated direct solution algorithm, based on processing substructures in parallel, but also allowing a high degree of parallelism in the computation within each substructure. We further discuss the use of iterative solution strategies. The paper concludes that iterative techniques should be further investigated and that the current knowledge about such methods makes them attractive for certain special classes of problems already today. We provide a few numerical examples in support of our conclusions.

1 Introduction

Structural analysis provided the problems, motivation and pioneering work that led to the development of the powerful finite element method. This field is still the most important application area of finite element based analysis. There exist a number of highly successful commercial codes that can be used to analyze the behavior of almost any kind of structure under rather general conditions. In almost all of these codes the technique called substructuring [1] is used in order to simplify modeling and post processing. The physical domain is divided into several disjoint pieces called substructures, and each substructure can be assembled separately. The global stiffness matrix is then usually formed in order to solve the equations.

One commercial code, SESAM, [29],[3] pioneered the further use of this concept, by also taking advantage of the substructures (in particular identical ones) in the solution

*This work was supported by NTNF, contract IT0228.20484
†Institutt for Informatikk, University of Bergen, Thormøhlensgate 55, N-5008 Bergen, NORWAY
‡Veritas Sesam Systems, Box 300, Veritasveien 1, N-1322 Høvik, NORWAY.
§Veritas Sesam Systems, Box 300, Veritasveien 1, N-1322 Høvik, NORWAY.
[1]SESAM is marketed by Veritas Sesam Systems Inc.

algorithm. In this code the solution procedure is carried out by the elimination of interior unknowns from each substructure, followed by the calculation of Schur complements corresponding to the unknowns on interior interfaces between substructures.

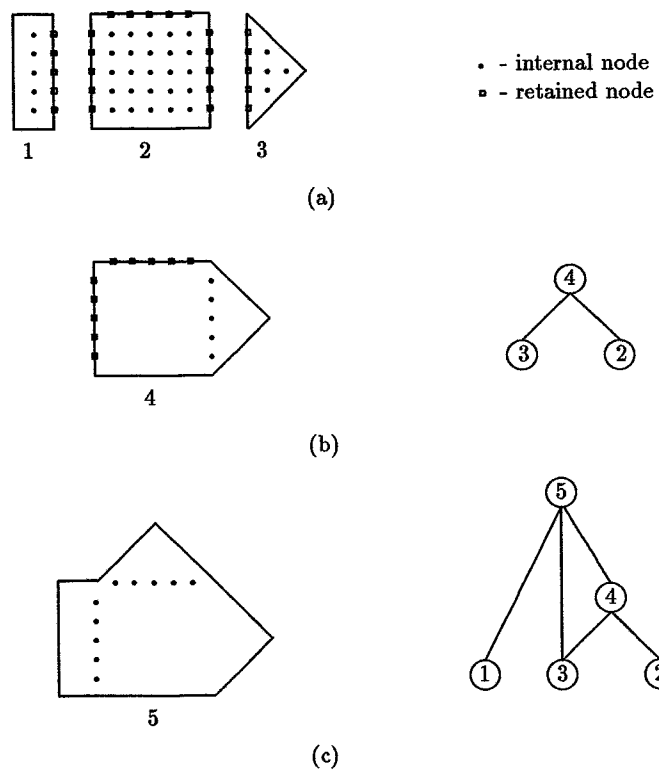

Figure 1: *The multilevel substructure technique.*

This strategy is implemented in a recursive fashion, thus making a multilevel substructure implementation. At any given level in this procedure, the unknowns can be uniquely divided into two

disjoint sets, the internal variables and the retained or external variables. The latter group of variables can be interpreted as boundary values or fixed degrees of freedom at this level of the algorithm. Before the algorithm proceeds to the next level, all internal variables are eliminated. At the next level the retained variables from the previous level are again split into two sets and this process repeats until one reaches the highest level where all remaining variables (or unknowns) in the problem will be in the internal set.

The organization of a finite element model, as well as the computational procedure outlined above, can be represented as in Figure 1. The dots represent *internal* nodes, these are nodes which cannot be shared with other substructures. The small squares are *retained* nodes, which are nodes retained for the purpose to be coupled together with retained nodes from other substructures. Retained nodes at a given level may become internal or

specified nodes at the next (upward) level. Figure 1(b) shows an intermediate level in the assembly process where substructure 2 and one instance of substructure 3 are coupled together to form the new substructure 4. The retained nodes on the interface between these two substructures become internal nodes in substructure 4, while the remaining retained nodes from substructure 2 keep their status. The substructure tree of this intermediate level is shown to the right in 1(b). Finally, Figure 1(c) shows the entire computational model where substructure 1 and a rotated instance of substructure 3 are matched with the remaining retained nodes of substructure 4 to form substructure 5, the *structure level.* On the structure level there are only internal variables. A small, but realistic model of a tubular joint assembled from 9 substructures is pictured in 2, together with the simple, two-level substructure tree.

The leaves of the tree (not shown in Figure 1 or 2, corresponding to 'zero-level' substructures), represent basic finite elements, the branches indicate dependencies. Each node in the tree as well as subtrees can also be directly associated with physical components of the structure that is modeled. Identical subtrees show identical substructures from which the global structure is built. This tree can also be used to illustrate the computational procedure, and in particular, the coarse grain parallel possibilities of both direct and iterative algorithms. The Cholesky factorization of the global stiffness matrix can be viewed in terms of a block algorithm, by processing all the different branches of the tree starting at the first level substructures and working up to the root. (Note that only one copy of identical branches or sub-branches are processed in this scheme.) Similarly, given any particular right hand side vector (load on the structure), one can find the displacement

(and strain) at a specified point in the structure, by traversing the tree in the opposite direction from the root and along the branches that lead to the corresponding element in the finite element model. This implies flexibility when the solution is of particular interest in certain areas of the structure. In order to find the solution everywhere, all branches including identical ones, must be traversed in the back substitution phase.

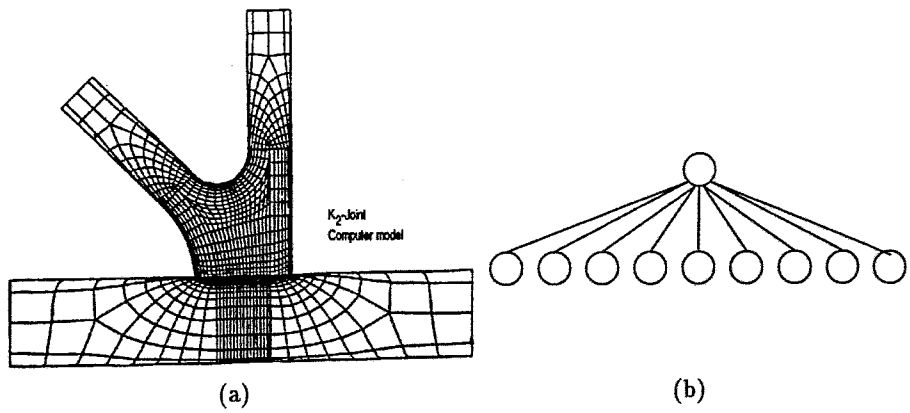

Figure 2: *Two level computational model.*

2 A direct, parallel substructure algorithm

The original ideas for an efficient, parallel, direct solver implemented in the SESAM system, were described in [4]. At the end of 1989, many of these proposals had been carefully investigated and implemented by Anders Hvidsten as described in his thesis [21]. We will in the following sections report on the most significant findings from this large project. In order to provide sufficient flexibility in a rapidly changing parallel computing environment, the target parallel machine was defined to consist of one or more computational nodes $N_i, i = 1, n$ individually capable of running the SESAM code and in addition, satisfy the criteria:

- N_i has local memory
- N_i has $m_i \geq 1$ processors
- N_i is connected via some network to all nodes $N_j, j = 1, n$
- N_i has access to all secondary storage devices in the system
- N_i has identical binary data representation

The majority of modern computing environments satisfy these requirements, the last one can of course be omitted at the expense of running an appropriate filter program.

This computer model makes the set of possible systems that can run SESAM considerably larger than before. As will be discussed below, different parts of the parallel algorithm will be attractive on different computer models. The coarse grain level can benefit from computers connected in a fairly low bandwidth network, provided that there is local data storage.

The parallel algorithms that are implemented inside each substructure, on the other hand, depends on somewhat higher interprocessor communication bandwidth. More importantly, these algorithms depend on high bandwidth to the relevant storage devices in the system. These algorithms are currently restricted to the elimination of internal variables and the computation of the corresponding Schur complement within each substructure. This is the only computational task that has a complexity growing faster than linearly in the number of unknowns. The algorithms are all organized as block linear algebra routines, and the task granularity corresponds to the computation of a resulting block, often involving a considerable number of Blas-3 level operations [14] with a program tunable block size parameter. Unlike the LAPACK [13] [2] software, all matrix blocks are square and of the same size (typically 50-100), with the exception of a few rectangular 'remainder blocks' of smaller dimensions. The potential for fine grain parallel computation of the Blas-3 kernels are left to the possible parallel hardware that may exist internally to what we define as a computational node.

We have worked with computer systems that are suitable for either the coarse grain or the medium grain parallel level. There are currently very few computer systems that are able to take full advantage of both parallel levels simultaneously.

The potential reduction in elapsed computing time for a model like Figure 2 is not very dramatic, but for larger models having hundreds of thousands of unknowns in a substructure tree with many branches, resulting in fairly large, dense computational tasks at the higher levels, the gain can be considerable. Figure 3 shows curves that represent upper bounds on the speedup that can be obtained by applying different parallelization strategies to compute the tubular joint model in Figure 2. As the substructure matrices

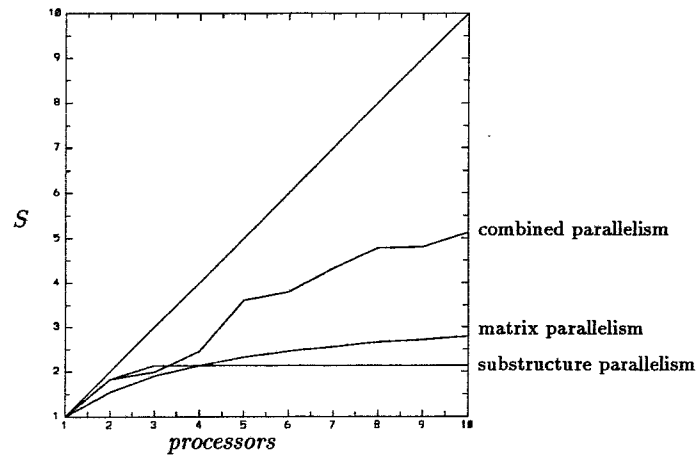

Figure 3: *Upper bounds on the speedup with substructure and block parallelization strategies, for the model in the previous figure, S is the speedup.*

involved in this calculation, are fairly small and in order to get an upper bound that is independent of the actual block size, we have assumed that the individual matrix problems achieve linear speedup as a function of the number of processors used. It is also assumed that the processors involved to obtain these bounds are identical. The best speedup line in figure 3 shows the upper bound for the speedup when combining the substructure and the matrix parallel strategies.

2.1 Distributed data structures

In order to process individual substructures in parallel, each substructure must have a separate and complete data structure that contains all information necessary to complete its calculation. The restructuring of the data structures was a major step motivated by the development of parallel algorithms, but this reorganization of data has many other advantages as well. The ability to stop and restart a computer analysis as well as secondary storage management have been improved. (These features actually help the introduction of parallel computing into a conservative user community!) A main advantage with the substructure approach is that the entire calculation can be made parallel, including a large number of statements that would be virtually impossible to parallelize in any other way.

2.2 The pool of tasks concept and inter-process communication

Since different substructures in the same model, typically have very different sizes and our computer model allows for computational nodes with unequal speed, the optimal assignment of substructures to computing resources is difficult. (Indeed, both of the above problems are NP-hard to solve [20].) Our strategy uses a *pool of tasks* [24] where information about all the tasks including precedence relations, time estimates and the computing system

is stored. This approach ensures a symmetric and well balanced load, there is no predetermined job scheduling.

In the active pool process implementation there is one process that executes a pool algorithm and a set of computational processes that use primitives from Table 1 to communicate with the active pool process.

generic name	description
create	Create a process.
open	Open a communication channel to a process.
send	Send a message to a process.
recv	Receive a message from a process.
close	Close the communication channel to a process.

Table 1: *The required communication primitives for the active pool process.*

The generic pool process that is executed by the pool of tasks, will loop until a user specified condition is no longer true.

The first entry point in this algorithm is a *recv*, which blocks the execution of the pool process until a computational process sends a request to it. Depending on the type of this incoming request, the pool process performs separate actions.

The implementation of the active pool primitives is based on a *send* construct that can send a message to a specified process on a specified node of a computing environment in addition to

associate a message type with each message to send. The corresponding *recv* construct must be able to distinguish between accepting messages with any message type and to block execution until it receives a message of a specified message type. The message passing primitives conforms to the Intel iPSC/1-2 *send* and *recv* constructs [22].

A typical process picture throughout the execution of the two-level parallel algorithm will be of the form shown in Figure 4, with two sets of processes attached to matrix computations. The figure shows the substructure pool P_s and two (temporary) matrix pools P_m together with the computational processes they control. The process p_1^1 has turned itself into a temporary (matrix) pool process after it has created the two matrix processes p_2^1 and p_2^2. It is also assisted in the matrix computation by the substructure process p_1^2. Process p_1^5 controls in a similar way two separate processes to perform another matrix computation. The substructure process p_1^3 is either in a non-matrix part of a substructure computation or it is idle. This approach will dynamically add idle processors as they become available to ongoing matrix computations in a very smooth way.

A large analysis can start with many low level substructures being processed in parallel and end with many processors working together at the higher level matrix computations. As the matrix problems in the low level substructures often are sparse, while the higher level problems become dense, this strategy gives good processor utilization at all levels.

The number of processes that should be assigned to substructure calculations and the additional number that can be dedicated to matrix calculations, will depend on both the size of the computation and of the type of the computing environment in question.

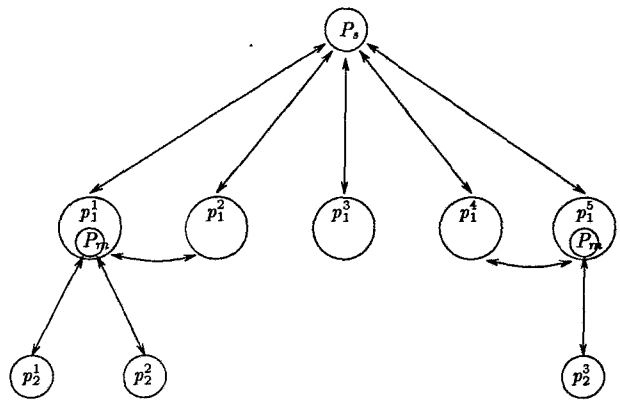

Figure 4: *Two-level process picture.*

For example, in a computing environment where the separate nodes are shared memory multiprocessors that are connected in a cluster, one strategy would be to assign substructures to the computing nodes, and have each node spawn matrix processes locally. Because of the normally large imbalance of the transfer speed between the set of shared memory multiprocessors and between the processors within each multiprocessor, it may not help to assign idle substructure computing nodes to matrix processing.

Consider also a hypercube type computer with disk devices connected to separate nodes like in Figure 5. A natural way to create processes in this environment is to define

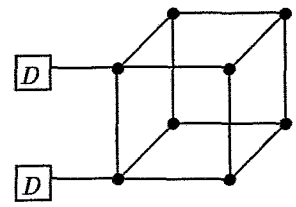

Figure 5: *Computing environment with two disk devices D.*

subsets of processes, i.e. subcubes, where each subcube executes one substructure process and a number of matrix processes. The number of matrix processes should match the dimension of the actual subcube. A computing environment partitioned into many small subcubes at the beginning of a run, can automatically reconfigure itself into larger and larger subcubes as the computation traverses towards the top of the associated substructure assembly tree.

A third example of a computing environment where one can apply the two level scheme, is a set of workstations connected through some high speed network. If all the

nodes in this system are equivalent with respect to the substructure computations, then a natural concept is to create one substructure process on each node in the environment and let these processes start to compute substructures. At the start of each matrix computation, the node creates an additional matrix process to ensure progress with the actual matrix task. When the substructure pool is (temporarily) empty, the scheduling scheme can decide on a switching between the substructure computation algorithm and the matrix computation algorithm for the substructure process that experience the empty pool.

2.3 Example calculation of a tubular joint.

We report on a calculation of a tubular joint, considerably more complex than the one in Figure 2. The joint has 16 first level substructures and a total of 9852 unknowns. In Table 2 we list the results of 2 runs. The first run is using the sequential version of SESAM, tuned for Cray computers. We report on elapsed times on a machine with no other activity. The second run uses 4 SUN 4/75 (Sparc-2) workstations and the parallel version of the program, restricted to do substructures in parallel only.

	First level	Second level	Total
Cray X-MP	171 s	146 s	317 s
4 SUN 4/75	263 s	721 s	984 s

Table 2: Comparison of supercomputer and workstations analyzing a tubular joint.

Table 2 shows that the workstations come close to a factor of 3 compared to the Cray X-MP on this example. In the parallel computation of first level substructures the workstations perform at 65 percent of the speed of a Cray. In a more realistic production environment (having several jobs on the Cray) we have observed elapsed times that are 2 to 3 times longer on the supercomputer, making the two systems quite similar with respect to 'time to solution'. A contributing factor to the performance of the workstations in this example, is the total bandwidth to secondary storage devices in the two computing environments. The Cray employs one (expensive) fast disk, while the 4 workstations each use a standard, local SCSI disk. On the Cray, this application is very I/O bound, while the workstations achieve a much higher cpu utilization. The example also shows the need for the matrix parallel algorithms since three workstations are idle more than 70 percent of the

time. Currently, the computer network based on Ethernet technology, severely limits the potential of todays workstations in this respect. The matrix parallel algorithms depend on a reasonable balance

between communication speed and computational speed. Unfortunately, due to the rapid microprocessor development, the next generation FDDI based network, will only improve this situation marginally. Based on this experience, we see a very cost effective computing alternative provided that clusters of high performance microprocessors with sufficient internal communication bandwidth, become available.

3 Iterative substructure algorithms

The renewed interest from the numerical analysis community in substructuring as a way of breaking up the solution of elliptic problems into problems on smaller domains has provided a new understanding and a theoretical foundation for the use of iterative methods in this process. An early numerical analysis paper in this direction is [12]. More recent work is described in [15,16], [6], [9], [7], [10], [11] see also the references listed in these papers. In order for iterative methods to gain acceptance in the structural analysis community, the method must fulfill many requirements.

A very important consideration is the effort needed to incorporate new algorithms into existing industrial software. This means that an algorithm should be based on accessing information that has been - or easily can be computed. Another important consideration is the ability to solve for multiple right hand sides in an efficient manner. The detection of errors due to for example incorrect use of a preprocessing program and the ease of porting the program to a wide range of computer systems, are two other goals that must be satisfied.

It is interesting to note that a block implementation of the conjugate gradient method employing diagonal scaling is described in the SESAM user manual dated 1972. Unfortunately, although this block conjugate gradient algorithm is written and programmed entirely in terms of matrix operations, the inversion of the block matrices that generalizes the scalar parameters, is done by just inverting the diagonal elements. (The programmer obviously just wanted to create a block algorithm based on processing multiple right hand sides, but he came very close to discovering the block conjugate gradient algorithm well before its first description in the literature [25].)

The existing SESAM substructuring method does not use overlap between substructures. It may therefore be natural to first study the

iterative substructuring methods as a basis to implement preconditioned conjugate gradient algorithms in SESAM. The methods of particular interest are those which operate on matrices that already exist within the SESAM framework. The implementation of the Neumann-Dirichlet iterative substructuring algorithm [7] in SESAM, see Bjørstad and Hvidsten [5], is an example of a method that builds a preconditioner from matrices readily at hand. That work was the first example of a new algorithmic theory on iterative procedures implemented in an existing commercial structural analysis code. Although the algorithm has its practical limitations, it verifies that the modular design of SESAM makes it possible to proceed in the direction of iterative substructuring algorithms. The implementation of new iterative substructuring and iterative refinement strategies into large scale industrial codes may offer an attractive software evolution path because it can be achieved without a substantial reorganization and rewriting of the software.

There exist several techniques to reorganize the implicit matrix representation from Figure 1 into the framework of different preconditioning strategies. One technique is to reduce the number of levels of the substructure assembly tree associated with a particular matrix, by collapsing subtrees within the global tree. With this collapsing technique, retained nodes from substructures are coupled together and moved into the set of internal nodes of a new and larger substructure. In the extreme case, the collapsing of all substructures in a computational model, will produce the global stiffness matrix for the entire model.

Another technique to shorten multilevel trees is to compute some of the Schur complements, for example all Schur complements up to a certain level l. The highest

level substructures with their Schur complements computed then become the new first level substructures. Figure 6 shows a four-level substructure assembly tree which has been collapsed into a new two-level tree by applying both the top and the bottom collapsing strategies. The boxes in Figure 6 describe which of the substructures from the original substructure assembly that are collapsed into the new substructure levels.

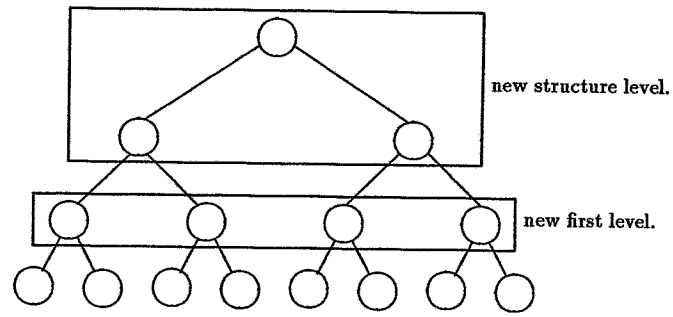

Figure 6: *The substructure collapsing technique*

3.1 Hierarchical change of basis

The hierarchical change of basis proposed by Yserentant [33] is an attractive way to precondition finite element problems in two dimensions. In the context of substructures in the plane, an attractive idea is to apply the change of basis only to the retained variables (see Figure 1), that is

directly to the Schur complements that are computed at the first level. A conjugate gradient iteration can then be used to solve the next level problem, accessing the matrix in its unassembled form, including the hierarchical transformation matrices. It is easy to see that the hierarchical basis change derived from the retained variables between two substructures, applied to the corresponding Schur complement is equivalent to apply the hierarchical change of basis to the entire substructure and then compute the resulting Schur complement. The former approach is clearly the most attractive. A few results from this approach (first presented at the Salishan Sparse Matrix Conference in May 1989), are given in Table 3.

level	unknowns	dim(S)	iter.	$\kappa(S)$	iter.	$\kappa(T^TST)$
2	42	14	8	18.76	8	7.89
3	210	30	18	37.67	16	11.41
4	930	62	33	79.42	28	14.65
5	3906	126	62	161.06	38	18.25
6	16002	254	120	323.26	45	22.25

Table 3: *Hierarchical basis change in the Schur complement, membrane elements.*

Although this technique appears attractive, the method has limited applicability since the problem must be at most 2-dimensional for these results to hold. A careful discussion of this technique applied to the Laplace

model problem is given in [28]. J. Xu has proposed a nodal basis preconditioner that overcome some of the limitations of the hierarchical basis preconditioner [31,32], see also [8]. This method can be applied in a similar way and may be the basis for a more general preconditioning technique.

3.2 Approximate Schur complements

The preconditioner implemented and studied in this section, may be thought of as building an approximate Schur complement. It is based on an idea in Dryja and Widlund [18] to zero out off-diagonal blocks in the Schur complement matrices. This method is also related to the Bramble, Pasciak and Schatz method [9]. In the context of the model problem described in [9], the main difference in these two methods, is the description of the coarse global system. In [9], a separate coarse 'wire basket' is defined with piecewise linear functions between the vertices, while in this preconditioner, the global problem is

obtained by carrying out the standard block Cholesky elimination with some blocks within the matrices explicitly set to zero. Mandel [23] describes a similar preconditioning strategy where the coarse global system is defined by average values from each substructure.

In the framework of the multilevel substructuring technique, the preconditioner described here may zero out off-diagonal blocks at all levels of multilevel substructure trees or at certain levels only.

Let K be the stiffness matrix corresponding to internal nodes at a certain level. In 2 dimensions, the block structure of K is defined such that each block consists of those unknowns which are on the common interface between two substructures from a lower level. The preconditioner is constructed by replacing off-diagonal blocks in K by zero-blocks before the usual block Cholesky elimination step. Applying this recursively at every level of a hierarchy gives successively coarser approximations to the exact Schur complement. An argument showing why this preconditioner still might be a good approximation to the original Schur complement, is that the interfaces at higher levels depend on the global behavior of their descendant substructures and that they are not strongly dependent on the more local information that has been replaced by zero. To ensure that all the modified Schur complements on all levels are positive definite, it is sufficient to impose the restriction that the structure level matrix has full rank. For all intermediate levels, the grouping of unknowns into internal and retained sets, and restricting the removal of blocks to the matrix K, will guarantee this.

Widlund [30] shows that for regions divided into substructures like the two-level models in Figure 7, the condition number of the preconditioned system grows proportional to $(1+log(\frac{H}{h}))^2$ where H is the dimension of a typical substructure and h is the diameter of a typical finite element within a substructure. An interesting question is whether the same logarithmic bound holds when the technique is applied recursively in multilevel hierarchies. The numerical results for Model 3 in Figure 7 indicate that this may be true.

Taking model 1 in Figure 7 as an example, the stiffness matrix for the edge and

vertex system of equations, has the form:

$$S = \begin{pmatrix} S_{11} & S_{12} & 0 & S_{14} & S_{15} \\ S_{12}^T & S_{22} & S_{23} & 0 & S_{25} \\ 0 & S_{23}^T & S_{33} & S_{34} & S_{35} \\ S_{14}^T & 0 & S_{34}^T & S_{44} & S_{45} \\ S_{15}^T & S_{25}^T & S_{35}^T & S_{45}^T & S_{55} \end{pmatrix},$$

S_{55} is associated with the vertex node common to all four substructures and S_{ii} , $i = 1, .., 4$, with the edges between two and two substructures. Constructing our preconditioner from S gives the sparse matrix:

$$\hat{S}_z = \begin{pmatrix} S_{11} & 0 & 0 & 0 & S_{15} \\ 0 & S_{22} & 0 & 0 & S_{25} \\ 0 & 0 & S_{33} & 0 & S_{35} \\ 0 & 0 & 0 & S_{44} & S_{45} \\ S_{15}^T & S_{25}^T & S_{35}^T & S_{45}^T & S_{55} \end{pmatrix}.$$

In order to estimate the cost of the preconditioner as compared to the standard block Cholesky elimination, consider a model problem defined on the unit square.

In the first partitioning strategy each square is recursively partitioned into smaller squares in the following way. Each square at a given level j in the partitioning process is divided into four equal new squares. This process is repeated l times, resulting in $m = 2^l$ substructures along each side of the original square. Model 3 in 7 is partitioned with this strategy using $l = 2$. The number of unknowns along each substructure edge at level j in the partitioning process is $N = \frac{n}{2^j}$ and there are 4^j squares at level j where n is the number of unknowns along each side of the original square.

The number of operations needed to compute the preconditioned

and the original factorization at all the levels can be estimated. At level $j = 0$ (the structure level) there is only a Cholesky factorization to perform. The operation counts for the Cholesky factorization are $\frac{4}{3}N^3$ and $\frac{4^3}{3}N^3$ for the preconditioned and the original system respectively. For the levels $j > 0$, the additional number of operations needed to compute the Schur complements for the preconditioned and the original system are counted to be $\frac{76}{3}N^3 + O(N^2)$ and $\frac{319}{3}N^3 + O(N^2)$ respectively. Summing over all l partitioning levels show that the operation count of the preconditioner compared with the block Cholesky factorization approach ratio

$\frac{104}{767}$ as l increases. To form the preconditioner is therefore in the limit approximately eight times cheaper than the direct factorization. This ratio decreases from 16 in the case of one level.

Consider the following alternative partitioning strategy without recursive partitioning. The original square again has n unknowns along each side and is divided into 4^l equally sized smaller squares. Model 2 in Figure 7 is partitioned with this strategy taking $l = 2$. The number of unknowns along each edge of a square when there is a total of 4^l squares, is $N = \frac{n}{2^l}$. The number of interior edges of the partitioned square is $2^{2l+1} - 2^{l+1}$ and the block bandwidth of the stiffness matrix of the edge systems of equations is $2^{l+1} - 1$. The operation count to compute the Cholesky factor of this banded matrix is of $O(N^3 2^{4l})$ if $2^l \gg 1$.

With the latter partitioning strategy, the preconditioner gives a block diagonal system of dimension $N \cdot (2^{2l+1} - 2^{l+1})$ where N is the dimension of the diagonal blocks.

In addition, there is a small dense stiffness system corresponding to the vertex nodes that must be factored. The operation count for the vertex system is of $O(2^{6l})$. As long as the factorization cost of this vertex system is small compared to the cost of the block diagonal system, the operation count of the preconditioner is of $O(N^3 2^{2l})$. Hence the ratio of the operation count between the preconditioned and the original system is of $O(2^{-2l})$

when the vertex system can be ignored.

Comparing the two partitioning strategies, shows that the latter method gives a reduction in the number of operations, compared with a direct solution method, which is proportional to the number of substructures.

The approximate Schur complement preconditioner has been applied to the 2-dimensional model problems in Figures 7 and 8. The models are defined on the unit square and they all have the same underlying finite element mesh. The thickness of the elements are 0.05 and the Poisson ratio $\nu = 0.3$. Both 4 node membrane elements and 4 node shell elements are used to generate the stiffness matrices for the system of equations to be preconditioned in the tests. The membrane elements have 2 unknowns per node while the shell elements have 5 unknowns per node, three displacement unknowns and two in-plane rotational unknowns. The differences between the separate models are the different groupings of finite elements into substructures and the assembly of substructures into computational models. The different grouping strategies give rise to different numbers of retained unknowns along the internal edges.

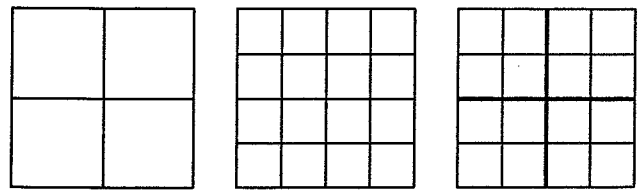

Figure 7: *The 2D Model Domains: Squares.*

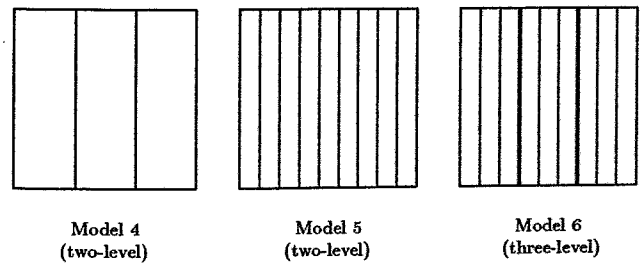

Figure 8: *The 2D Model domains: Strips.*

The initial guess of the solution vector x has been $x = 0$ in all the tests. The

stopping criteria is

$$\frac{1}{\lambda_{min}} \frac{\| r\|_2}{\| x\|_2} < \epsilon = 1.0 \cdot 10^{-10}$$

where λ_{min} is obtained from the representation of the Krylov subspace at hand in the conjugate gradient iteration and r is the residual [19]. For the membrane elements, this stopping criteria produced the same iteration count as the component based stopping criteria

$$\max_i \frac{| x_i - x_i^{direct} |}{| x_i^{direct} |} < \epsilon = 1.0 \cdot 10^{-10}$$

where x_i^{direct} is found from a Cholesky factorization of the problem. With shell elements, the amplitudes in the rotational variables are several orders of magnitude less than in the displacement variables. .This results in a larger number of iterations to obtain a given tolerance, using the last criteria.

The results from the tests are organized in the following three paragraphs where each paragraph presents results from a group of models.

Model 1 consist of 4 first level substructures assembled into a two level computational model. Table 4 shows the preconditioning results for Model 1 with membrane elements and Table 5 shows the corresponding results with shell elements. In each table the condition numbers $\kappa(S)$ and $\kappa(S_z^{-1}S)$ for the original and the preconditioned system are tabulated as a function of decreasing mesh size h. For the membrane elements, it can be seen that the condition number $\kappa(S_z^{-1}S)$ grows proportional to $(log(h^{-1}))^2$. membrane element test For the shell elements the situation is slightly different. The condition num-

N	h	$\kappa(S)$	I	$\kappa(S_z^{-1}S)$	I
37	.0500	17.91	18	5.69	11
69	.0278	33.24	27	7.96	12
133	.0147	64.00	39	11.00	13
261	.0075	125.59	55	14.82	15

Table 4: *Preconditioning results for Model 1 using membrane elements with two unknowns per node. N is the number of nodes along the edges. $\kappa(S)$ and $\kappa(S_z^{-1}S)$ are condition numbers of the original and the preconditioned system. I is the iteration count.*

bers are reduced significantly in all the tests but going from $h = 0.05$ to $h = 0.0278$ shows a doubling of the condition number. However, in the next refinement step, the logarithmic growth is visible. A more detailed investigation of the matrix entries in S and S_z shows

N	h	$\kappa(S)$	I	$\kappa(S_z^{-1}S)$	I
37	.0500	25360.52	236	78.44	17
69	.0278	45980.56	399	175.34	19
133	.0147	87924.25	>500	226.84	22

Table 5: *Preconditioning results for Model 1 using shell elements with five unknowns per node. N is the number of nodes along the edges. $\kappa(S)$ and $\kappa(S_z^{-1}S)$ are condition numbers of the original and the preconditioned system. I is the iteration count.*

that within the off diagonal blocks that have been replaced by zeroes, there are entries

in the range $10^{-2} - 10^{10}$. This is pictured in Figure 9 where the x entries in the matrix $|S - S_z|$ are plotted for Model 1 with $h = 0.05$. The peaks represent the missing couplings between nodes that are adjacent to the vertex node. It can be seen that this error is rapidly decaying as the distance from the vertex increases. One can improve the preconditioner

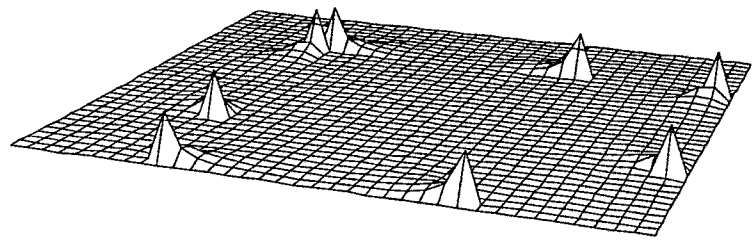

Figure 9: *x entries in matrix $|S - S_z|$ for Model 1 with $h = 0.05$. Scaling $= 5.0 \cdot 10^{-10}$.*

by moving adjacent nodes from each of the internal edges into a new extended cross set $\tilde{S}_{55}$. This strategy will reduce the largest entries in the matrix $|S - S_z|$. Table 6 shows the decrease in $\kappa(S_z^{-1}S)$ and in the number of iterations, as the number of nodes in the extended cross is increased. The test is done with a mesh parameter $h = 0.0075$ for the membrane element model.

N_c	l	$\kappa(S_z^{-1}S)$	I
1	*vertex*	14.82	15
5	.02	7.96	13
13	.05	4.56	10
25	.10	2.92	8

Table 6: *Preconditioning results for Model 1 with extended vertex set. N_c is the number of nodes in the new vertex set. l is the linear size of the new vertex.*

Both Model 2 and Model 3 are based on the same 16 first level square substructures. In Model 2 all these substructures are combined into the structure level while in Model 3 there is an intermediate level, grouping four and four substructures together. The structure level substructure is here represented by the thick lined edges built from the four substructures at the intermediate level. With Model 2, the preconditioner is similar to the Bramble, Pasciak and Schatz preconditioner [9]. With the three-level model, Model 3, there are three different ways to introduce zeros in the matrix. First, one can remove the coupling between the edges of the four intermediate substructures (thin lines). The second option is to remove the coupling between the edges in the highest level substructure (thick lines). The third option is to apply the preconditioner at both levels simultaneously. Fig-

ure 10 plots the growth in the condition number for the iteration matrix with the various preconditioning options arising from Model 2 and Model 3 as a function of $(log(h^{-1}))^2$. The condition numbers grow linearly as expected.

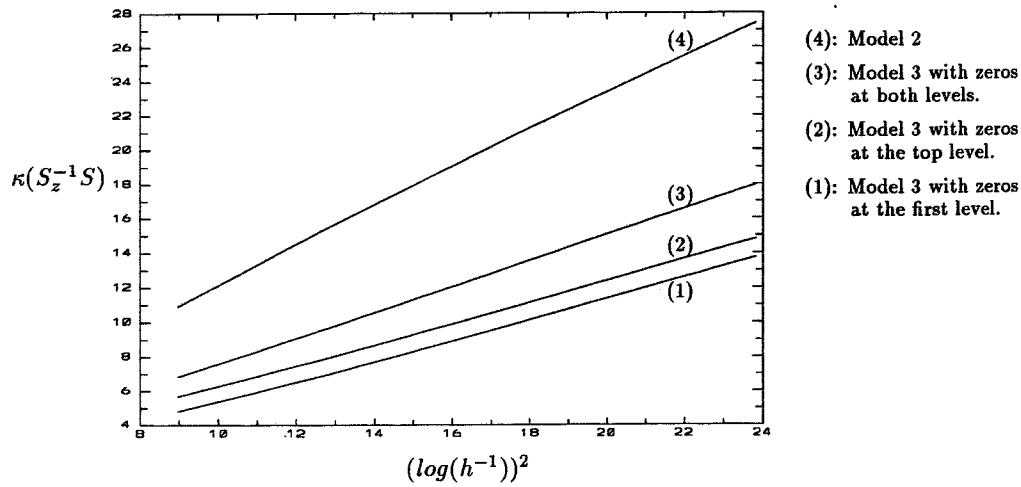

Figure 10: *The condition number $\kappa(S_z^{-1}S)$ as a function of $(log(h^{-1}))^2$ for Model 2 and Model 3 meshed with membrane elements.*

Model 4-6 are assembled from substructures being strips. The matrix picture of the Schur complements assembled from strips are by nature different from those assembled from squares. This is so because there is a constant distance between the edges in all strip models. Figure 11 plots the entries of the matrix $| S - S_z |$ for Model 4. By comparing this plot with the one in Figure 9, the different nature of strip and square models can be seen. Note that the scaling of the entries in the two plots differs with a factor of ten. Tables 7 and 8 show the condition number and the iteration count for the various preconditioners applied to the strip models.

		Model 4	
N	h	$\kappa(S)$	$\kappa(S_z^{-1}S)$
38	.0500	14.4	2.74
70	.0278	25.19	2.74
134	.0147	46.22	2.74

Table 7: *Preconditioning results for Model 4 using membrane elements. N is the number of nodes along the edges. $\kappa(S)$ and $\kappa(S_z^{-1}S)$ are the condition numbers of the original and the preconditioned system.*

Observe from Table 7 and 8 that for a given preconditioner, the condition number is independent of the mesh size. The refinement introduces new couplings with exactly the same magnitude in the strip case. In the square models, however, this approach leads to stronger coupling as the mesh is refined. Note also that introducing zeros at both levels in Model 6 gives identical condition numbers to the first case considered. This is expected

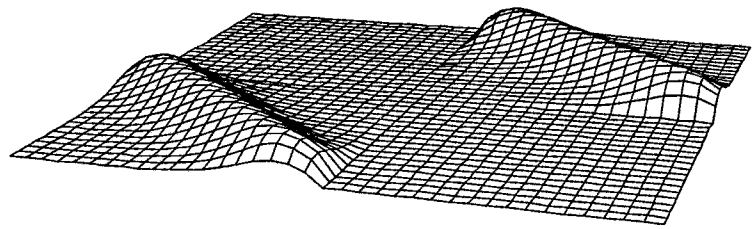

Figure 11: *x entries in the matrix* $\mid S - S_z \mid$ *for Model 4 with* $h = 0.05$ *Scaling* $= 5.0 \cdot 10^{-9}$.

			Model 5	Model 6		
		$\kappa(S)$	$\kappa(S_z^{-1}S)$	$\kappa(S_z^{-1}S)$		
N	h			opt1	opt2	opt3
152	.0500	40.91	24.90	9.32	2.47	9.32
280	.0278	58.97	24.90	9.32	2.47	9.32

Table 8: *Preconditioning results for Model 5 and Model 6 using membrane elements. N is the number of nodes along the edges.* $\kappa(S)$ *and* $\kappa(S_z^{-1}S)$ *are the condition numbers of the original and the preconditioned system. opt1, op2, opt3 are the different options referred to in the text.*

since two first level substructures that are decoupled will remain decoupled at all higher levels.

3.3 Additive Schwarz methods

Domain decomposition preconditioners based on ideas from Schwarz [26], has received much attention, see [16],[17] and the references in these papers. Until recently, most of this work related to model problems and scalar elliptic equations. In his thesis Barry Smith [27] developed and tested a method for two and three dimensional linear elasticity. He proved the optimality of the method and carried out extensive tests using the SESAM structural analysis program to provide the underlying stiffness matrices. This method appears very robust and quite promising, it is however, somewhat more complex to incorporate in a structural analysis code in its full generality. The preconditioner can be built in a systematic way, but unfortunately, the components that enter, are not already explicitly available in the SESAM code.

4 Conclusions

We have shown how a redesign of a large industrial finite element code for parallel processing, can be carried out. The resulting code has the capabilities to employ two different levels of parallel algorithms, dynamically switching between them, in order to achieve good parallel efficiency in all phases of the computation.

The development of a parallel version further makes the incorporation of iterative algorithms in parts of the solution process, easier. Several such algorithms based on preconditioned conjugate gradients, have been discussed and tested using different finite element formulations. The understanding of how to construct good preconditioner has improved during this project and are now ready to be tested out on realistic models.

With the preconditioned conjugate gradient method as a basis to solve systems of equations in SESAM, new methods to construct

preconditioners should still be investigated. The methods must have optimal or near optimal convergence rate properties.

We plan to pursue the incorporation of these and similar ideas for important classes of frequently analyzed structures. Almost all models of tubular joints like the one in Figure 2 consist of a potentially very large number of first level substructures combined with a very compute intensive structure level matrix. This is a very attractive special problem class where the combination of direct elimination of the first level matrices

(in parallel) combined with an optimal iterative solver like the one proposed by Barry Smith [27], may challenge the traditional use of direct algorithms.

References

[1] K. BELL, B. HATLESTAD, O. E. HANSTEEN, AND P. O. ARALDSEN, *NORSAM, a programming system for the finite element method. Users manual, Part 1, General description.*, NTH, Trondheim, 1973.

[2] C. BISHOF, J. DEMMEL, J. J. DONGARRA, J. D. CROZ, A. GREENBAUM, S. HAMMARLING, AND D. SORENSEN, *LAPACK working note no. 5 provisional contents*, tech. report, Argonne National Laboratory, September 1988.

[3] P. E. BJØRSTAD, *SESAM'80: A modular finite element system for analysis of structures.*, in PDE Software: Modules, Interfaces and Systems, B. Enquist and T. Smedsaas, eds., Amsterdam, 1984, North Holland, pp. 19–27.

[4] ——, *A large scale, sparse, secondary storage, direct linear equation solver for structural analysis and its implementation on vector and parallel arcchitectures.*, Parallel Computing, (1987).

[5] P. E. BJØRSTAD AND A. HVIDSTEN, *Iterative methods for substructured elasticity problems in structural analysis*, in Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Philadelphia, 1988, SIAM.

[6] P. E. BJØRSTAD AND O. B. WIDLUND, *Solving elliptic problems on regions partitioned into substructures*, in Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., New York, 1984, Academic Press, pp. 245–256.

[7] ——, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1093–1120.

[8] J. BRAMBLE, J. PASCIAK, AND J. XU, *Parallel multilevel preconditioners correction: A unifying approach*, Math. Comp., (1991).

[9] J. H. BRAMBLE, J. E. PASCIAK, AND A. H. SCHATZ, *The construction of preconditioners for elliptic problems by substructuring, I*, Math. Comp., 47 (1986), pp. 103–134.

.0] ——, *The construction of preconditioners for elliptic problems by substructuring, II*, Math. Comp., 49 (1987), pp. 1–16.

.1] ——, *The construction of preconditioners for elliptic problems by substructuring, III*, Math. Comp., 51 (1988), pp. 415 – 430.

.2] P. CONCUS, G. H. GOLUB, AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic PDE*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., New York, 1976, Academic Press, pp. 309–332.

[13] J. DEMMEL, J. J. DONGARRA, J. D. CROZ, A. GREENBAUM, S. HAMMARLING, AND D. SORENSEN, *Prospectus for the development of a linear algebra library for high-performance computers*, tech. report, Argonne National Laboratory, September 1987.

[14] J. DONGARRA, J. D. CROZ, I. DUFF, AND S. HAMMARLING, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Soft., X (1990).

[15] M. DRYJA, *A method of domain decomposition for 3-D finite element problems*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Philadelphia, 1988, SIAM.

[16] ——, *An additive Schwarz algorithm for two- and three- dimensional finite element elliptic problems*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Philadelphia, 1989, SIAM.

[17] M. DRYJA AND O. B. WIDLUND, *An additive variant of the Schwarz alternating method for the case of many subregions*, Tech. Report 339, also Ultracomputer Note 131, Department of Computer Science, Courant Institute, 1987.

[18] ——, *On the optimality of an additive iterative refinement method*, in Multigrid Methods, J. Mandel, S. McCormick, J. J.E. Dendy, C. Farhat, G. Lonsdale, S. V. Parter, J. W. Ruge, and K. Stüben, eds., Philadelphia, 1989, SIAM. Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods, April 9-14, 1989.

[19] G. H. GOLUB AND C. F.VAN LOAN, *Matrix Computations*, Johns Hopkins Univ. Press, 1989. Second Edition.

[20] R. GRAHAM, E. LAWLER, J. LENSTRA, AND A. RINNOY KAN, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Annals of Discrete Mathematics, 5 pp. 287-326, (1979). Nort-Holland Publishing Company.

[21] A. HVIDSTEN, *A parallel implementation of the finite element program SESTRA*, PhD thesis, Department of Informatics, University of Bergen, Norway, 1990.

[22] INTEL CORPORATION, *iPSC/2 USERS GUIDE*, March 1989.

[23] J. MANDEL, *On block diagonal and schur complement preconditioning*, Tech. Report TE 31089, University of Colorado at Denver, 1200 Larimer Street, Denver, CO 80204, October 1989.

[24] P. MØLLER-NIELSEN AND J. STAUNSTRUP, *Problem heap: A paradigm for multipocessor algorithms*, Parallel Computing Vol. 4 No. 1 pp. 63-74, (1986).

[25] D. P. O'LEARY, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.

[26] H. A. SCHWARZ, *Gesammelte Mathematische Abhandlungen*, vol. 2, Springer, Berlin, 1890, pp. 133–143. First published in Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich, volume 15, 1870, pp.272–286.

[27] B. F. SMITH, *Domain decomposition algorithms for the partial differential equations of linear elasticity*, PhD thesis, Courant Institute of Mathematical Sciences, New York, September 1990.

[28] B. F. SMITH AND O. B. WIDLUND, *A domain decomposition algorithm based on a change to a hierarchical basis*, Tech. Report 473, Department of Computer Science, Courant Institute, November 1989. submitted to SIAM J. Sci. Stat. Comput.

[29] VERITAS SESAM SYSTEMS, P.O. BOX 300, N-1322 HØVIK, NORWAY, *SESAM technical description*, april 1989.

[30] O. B. WIDLUND, *Iterative substructuring methods: Algorithms and theory for elliptic problems in the plane*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Philadelphia, 1988, SIAM.

[31] J. XU, *Theory of Multilevel Methods*, PhD thesis, Cornell University, 1989.

[32] ——, *Iterative methods by space decomposition and subspace correction: A unifying approach*, Tech. Report AM 67, Department of Mathematics, Penn State University, August 1990.

[33] H. YSERENTANT, *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379– 412.