Parallel SAS Multicluster Algorithms for Solving Linear Systems with Reflexive Coefficient Matrices

Hsin–Chu Chen

Center for Supercomputing Research & Development
University of Illinois at Urbana–Champaign
Urbana, Illinois 61801

Abstract

The SAS domain decomposition is an efficient scheme for the numerical solution of a wide class of discretized physical problems on either sequential or vector computers. It is also an amazingly parallelizable approach for multiprocessors like the Cray X–MP, the Cray–2, and the Alliant FX/80, or for multicluster machines of the Cedar type. The efficiency and parallelizability of this scheme for solving symmetric elasticity problems and slightly perturbed asymmetric problems have been demonstrated using an Alliant minisupercomputer which has only two levels of parallelism. For multiprocessors with parallelism of three levels such as the Cedar, this method offers much more freedom for the problem to be solved using all levels of parallelism. In this paper, we provide some strategies for the implementation of this domain decomposition method on multiprocessors of the Cedar type. Three parallel multicluster algorithms for solving linear systems with reflexive coefficient matrices are presented. The main difference among these algorithms lies in the use of the global memory and the cluster memories of the Cedar multicluster processor. The performance of these algorithms on the Cedar using the linear system derived from a 3D elasticity problem with two planes of reflexive symmetry is reported.

1 Introduction

Domain decomposition methods have recently been a research area under intensive studies (see [GGMP88, CGPW89] for references). Among all the methods, the SAS (symmetric–and–antisymmetric) domain decomposition is an efficient scheme for the numerical solution of a wide class of discretized physical problems on either sequential or vector computers. It is also an amazingly parallelizable approach for multiprocessors like the Cray X–MP, the Cray–2, and the Alliant FX/80 or for multicluster machines of the Cedar type. Unlike other approaches that use the same ideas of symmetry and antisymmetry [NoPe87a, NoPe87b, BrDM88, DoSm89], the SAS decomposition method exploits special properties of reflexive matrices and takes advantage of them to decompose a physical problem with reflexive symmetry into several independent subproblems. The efficiency and parallelizability of this scheme for solving symmetric and slightly perturbed asymmetric elasticity problems [ChSa87, Chen88, ChSa89a, ChSa89b] have been demonstrated using an Alliant FX/8 which has only two levels of parallelism. For multiprocessors with parallelism on three levels such as the Cedar, this method offers a great deal of freedom for a problem to be solved using all levels of parallelism. In this paper, we provide some general strategies for the implementation of this domain decomposition method on multiprocessors of this type. Although multicluster algorithms can be developed for essentially the entire analysis, including the generation of element stiffness (mass) matrices, the assemblage of the system stiffness (mass) matrix, and the solution process of the linear system or the generalized eigenvalue problem resulting from the finite element discretization, we shall emphasize only the solution process of solving the linear system in this paper. Three parallel multicluster algorithms for solving linear systems with reflexive coefficient matrices are presented. The main difference among these algorithms lies in the use of the global memory and the cluster memories of the Cedar multicluster processor. The performance of these algorithms on the Cedar for a 3D elasticity problem with reflexive symmetry is reported.

2 SAS Decomposition Method

The basic idea of the SAS domain decomposition method originates from the traditional symmetric-and-antisymmetric approach where structure engineers take advantage of the symmetry of physical problems by imposing proper boundary conditions along the line or plane of symmetry [BlKa66, Szil74, WeJo87]. The application of the SAS approach to problems which are SAS-decomposable but yet do not have clear physical meaning is made possible by the introduction of a special class of matrices referred to as reflexive matrices. Furthermore, this approach can also be applied to problems which are not SAS-decomposable via decomposing or splitting techniques. One way of doing this is to decompose the matrix, obtained from some sort of discretization, into a reflexive matrix and an antireflexive one, the counterpart of the reflexive matrix. Note that a matrix $A \in \mathcal{C}^{n \times n}$ is said to be reflexive (or antireflexive) with respect to P if $A = PAP$ (or if $A = -PAP$) where P is some reflection matrix (symmetric signed permutation matrix) of order n. A linear system $Ax=b$ or an eigenvalue problem $Av = \lambda v$ is also said to be reflexive if the coefficient matrix A is reflexive. Since it is not our main focus to discuss these matrices in this paper, we only state three simple but important properties associated with the SAS approach. Let P be some reflection matrix of order n.

Property 1: Given a nonsingular linear system $Ax = f$, $A \in \mathcal{C}^{n \times n}$, and $f, x \in \mathcal{C}^n$, if A is reflexive with respect to some reflection matrix P, then $x = Px$ (or $x = -Px$) if and only if $f = Pf$ (or $f = -Pf$).

Property 2: Any vector $b \in \mathcal{C}^n$ can be decomposed into two parts, u and v with $u = Pu$ and $v = -Pv$, such that $u+v = b$.

Property 3: Any matrix $A \in \mathcal{C}^{n \times n}$ can be decomposed into two parts, U and V with $U = PUP$ and $V = -PVP$, such that $U+V = A$.

Properties 1 and 2 provide all the important information we need to decompose a linear system $Ad = f$ with A reflexive with respect to P into two subsystems once P is given. Property 3 is useful for certain problems whose coefficient matrices, though not reflexive, are very close to reflexive matrices.

The SAS decomposition of the reflexive linear system $Ad = f$ into two subsystems is equivalent to applying a linear transformation with an orthogonal matrix X (X depends on the form of P) to the original reflexive system to yield a new system $\tilde{A}\tilde{d} = \tilde{f}$ where $\tilde{A} = X^T A X$, $\tilde{f} = X^T f$, and $\tilde{d} = X^T d$ such that the matrix $\tilde{A}$ consists of two disjoint diagonal submatrices, $\tilde{A}_1$ and $\tilde{A}_2$ for example. If the decomposed subsystems are also reflexive, further decompositions can be performed to yield more independent subsystems. See [Chen88] and [ChSa89a] for more information.

In the context of elasticity problems, the SAS decomposition of a reflexive linear system into subsystems corresponds physically to that of a single domain into subdomains with proper boundary conditions imposed implicitly on the interfaces. An example of this decomposition for 3D elasticity problems with reflexive symmetry will be shown in section 4. Now let N_s be the number of subdomains decomposed by the SAS domain decomposition method and $\tilde{A}_i$, $i = 1, N_s$, be the decomposed submatrices. The three major steps involved in this approach for solving the reflexive linear system $Ad = f$ arising from the static analysis of elasticity problems with reflexive symmetry can be described in the following basic algorithm:

1. the decomposition of the matrix A into $\tilde{A}_i$ and the right-hand side vector f into $\tilde{f}_i$;

2. the solution of the decomposed subsystems $\tilde{A}_i \tilde{d}_i = \tilde{f}_i$; and

3. the retrieval of the solution vector d from $\tilde{d}_i$

where $\tilde{d}_i$ and $\tilde{f}_i$, $i = 1, ..., N_s$, are the i^{th} partition of $\tilde{d}$ and $\tilde{f}$, respectively.

3 SAS Multicluster Algorithms on the Cedar

The Cedar computer system developed at the Center for Supercomputing Research and Development at the University of Illinois is a cluster–based multiprocessor [DKLS86]. This multiprocessor currently consists of 4 clusters of processors with 8 computation elements (CE's) in each cluster and of a global memory system. Each cluster is a modified Alliant FX/8 with a cluster memory and a data cache shared by all 8 CE's and is equipped with a global interface hardware. The 8 CE's in each cluster are connected to the global interface units and to the data cache through a cross–bar switch. The global interface units in each cluster are then connected to the global memory system via the global network system and the data cache is connected to the cluster memory via a memory bus. One of the main features of the Cedar architecture is its hierarchical memory system. Due to the hierarchical nature of its structure, the Cedar allows for exploitation of parallelism on three different levels: large–grain parallelism among clusters, medium–grain parallelism among processors within each cluster, and fine–grain parallelism using vectorization in each processor. The SAS approach is well suited for the Cedar machine when the number of subdomains is a multiple of the number of clusters because it exploits the parallelism on the large–grain level.

To implement the three main steps mentioned in the previous section on Cedar, we need first to decide where to generate and where to store the initial data: the matrix A (or $\tilde{A}$) and the vector f (or $\tilde{f}$) since the memory system is not centralized. This gives rise to several options. The first is to have all data generated and stored in the global memory. The second option is to generate and store all initial data in the cluster memory of the master cluster (a master cluster is defined in this paper as the cluster that initiates the process of the job). The third is to have each cluster generate and store its own data $\tilde{A}_i$ and $\tilde{f}_i$. There are other options. For instance, the mixed use of global memory and cluster memory.

Each option has its own advantages and disadvantages. The first option allows for direct access to the data by each cluster without the need of explicit copying. However, it may be more desirable for a cluster to fetch the data from its own cluster memory so that the cache structure can be exploited. Storing the data initially in global memory of course does not prevent us from copying them to the cluster memory of any particular cluster before computations in that cluster actually begin. The second option is useful when the third option is difficult to apply and when the cluster memory is much larger than the global memory. The third option is likely to be the most efficient one since each cluster avoids fetching the data, to a very large extent, from locations outside its own cluster memory. The need of synchronization among clusters during the generation of the data, however, makes the program more difficult to develop.

In the following, we present three multicluster algorithms implemented on the Cedar machine: SDOGM, SDOGC, and CTSKDB. These algorithms implement only the first two options of memory allocations. The implementation of the third option is still under development and will be reported later. For the sake of illustration, we denote the i^{th} cluster by C_i with C_1 being the master cluster. The cluster memory of C_i is denoted by CM_i and the global memory denoted by GM. We also assume that the number of subproblems, N_s, decomposed by the SAS approach is the same as that of clusters available for simplicity. The matrix $\tilde{A}$ and f are assumed already generated and available in GM for the algorithms SDOGM and SGOGC and available in CM_1 for the algorithm CTSKDB.

The first algorithm presented is SDOGM. This algorithm employs the idea of the first option of memory allocations and uses the Cedar Fortran constructs SDOALL [Guzz87, EHJP90] to assign one linear subsystem to one cluster.

Algorithm SDOGM:

1. C_1 decomposes f into $\tilde{f}$ with data in GM.
2. C_i solves $\tilde{A}_i \tilde{d}_i = \tilde{f}_i$ in parallel using multiple clusters, with $\tilde{A}_i$, $\tilde{f}_i$, and $\tilde{d}_i$ in GM, $i = 1, \ldots, N_s$.
3. C_1 retrieves d from $\tilde{d}_i$ with data in GM.

Similar to SDOGM, the algorithm SDOGC shown below uses SDOALL to fork all independent tasks among clusters. The difference between SDOGC and SDOGM is in the second step of the algorithms where SDOGC explicitly copies the data needed before and after solving the subsystems and then solves the linear subsystems using only cluster memory.

Algorithm SDOGC:

1. C_1 decomposes f into $\tilde{f}$ with data in GM.
2. C_i copies $\tilde{A}_i$ and $\tilde{f}_i$ from GM to CM_i in parallel using multiple clusters, $i = 1, \ldots, N_s$;

   C_i solves $\tilde{A}_i \tilde{d}_i = \tilde{f}_i$ in parallel using multiple clusters, with $\tilde{A}_i$, $\tilde{f}_i$, and $\tilde{d}_i$ in CM_i, $i = 1, \ldots, N_s$; and

   C_i copies $\tilde{d}_i$ from CM_i to GM in parallel using multiple clusters, $i = 1, \ldots, N_s$.
3. C_1 retrieves d from $\tilde{d}_i$ with data in GM.

Unlike SDOGM and SDOGC, the algorithm CTSKDB uses mainly the cluster memory. The global memory is used only for data transfer between the master cluster and other clusters. The Cedar Fortran constructs CTSKSTART and CTSKWAIT, instead of SDOALL, are employed to spread tasks among all clusters.

Algorithm CTSKDB:

1. C_1 decomposes f into $\tilde{f}$ with data in CM_1.
2. C_1 writes $\tilde{A}_i$ and $\tilde{f}_i$ from CM_1 to GM, while C_i copies at almost the same time $\tilde{A}_i$ and $\tilde{f}_i$ from GM to CM_i block by block using a double buffering technique, $i = 2, \ldots, N_s$;

   C_i solves $\tilde{A}_i \tilde{d}_i = \tilde{f}_i$ in parallel using multiple clusters, with $\tilde{A}_i$ and $\tilde{f}_i$ in CM_i and $\tilde{d}_i$ in GM, $i = 1, \ldots, N_s$; and

   C_1 copies $\tilde{d}_i$ from GM back to CM_1.
3. C_1 retrieves d from $\tilde{d}_i$ with data in CM_1.

Note that in these three algorithms, the decomposition of the right-hand side vector and the retrieval of the solution are not performed in multicluster mode because the computations in these two steps are negligible compared to solving the linear subsystems. It should also be mentioned that the decomposition of A into $\tilde{A}$ does not enter the algorithms because the matrix A need not be actually formed. Instead, we assemble the system stiffness matrix $\tilde{A}$ directly from the decomposed element stiffness matrices. Were it not the case, the decomposition of A into $\tilde{A}$ would be included.

4 Performance of the SAS Multicluster Algorithms

4.1. The problem. To test the performance of the multicluster algorithms that employ the SAS approach on the Cedar for 3D elasticity problems with reflexive symmetry, we consider the static finite element analysis of an isotropic prismatic beam with Young's modulus equal to 1000.0 and Poisson's ratio equal to 0.3. The beam, as shown in Figure 1, is fixed on the plane $x = 0$ and free on all other surfaces. Let the displacement and body force vectors be denoted by δ and $\mathbf{b}$, respectively, where

$$\delta^T = \left[\delta_1, \delta_2, \delta_3\right] \text{and } \mathbf{b}^T = \left[b_1, b_2, b_3\right].$$

Here the subscripts 1, 2, and 3 represent the

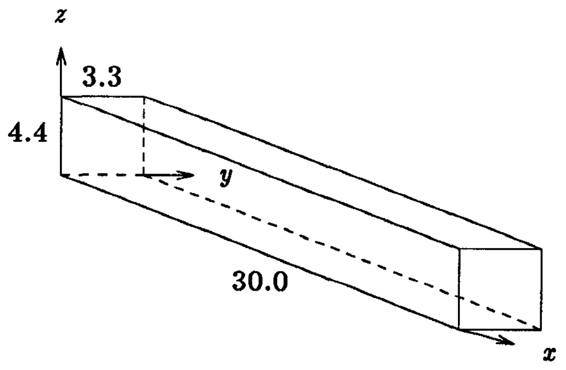

Fig.1. A prismatic beam.

three Cartesian directions x, y, and z respectively. The governing differential equations for such a problem [TiGo70, Breb83] can be expressed as

$$\mathcal{L}^T D \mathcal{L} \delta + \mathbf{b} = \mathbf{0} \quad in \ R^3$$

where D is the material property matrix, of order 6×6, of the elastic beam [Wang53, Sege76] and $\mathcal{L}$ is the differential operator:

$$\mathcal{L}^T = \begin{bmatrix} \partial/\partial x & 0 & 0 & \partial/\partial y & \partial/\partial z & 0 \\ 0 & \partial/\partial y & 0 & \partial/\partial x & 0 & \partial/\partial z \\ 0 & 0 & \partial/\partial z & 0 & \partial/\partial x & \partial/\partial y \end{bmatrix}.$$

The beam with dimensions 30.0×3.3×4.4 is discretized into a 16x11x11 uniform grid where 16, 11, and 11 are the numbers of grid spacings along the x, y, and z axis respectively. This beam, assumed to be subject to some asymmetric external loadings, is implicitly partitioned into four subdomains: I, J, K, and L, along the two planes of symmetry as shown in Figure 2. Note that we do not place any node on the planes of symmetry in our grid. Note also that the SAS decomposition does not depend on the symmetry of external loadings. Let m be the number of nodes in each subdomain. Let further I_i be the i^{th} node in subdomain I, $i = 1, m$, and its corresponding reflexive nodes in subdomains J, K, and L be denoted by J_i, K_i, and L_i respectively. For the sake of easy decompositions, we employ the following reflexive ordering for the nodes: $I_i = i$, $J_i = i + m$, $K_i = i + 2m$, and $L_i = i + 3m$. With this reflexive ordering, it can be shown [Chen88, ChSa89a] that the system

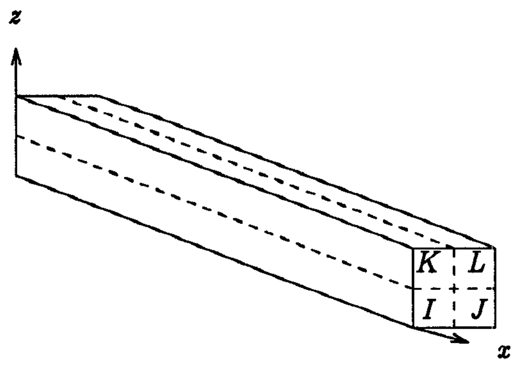

Fig.2. Implicitly partitioned subdomains.

stiffness matrix A has the following properties: $A = PAP$ and $A = QAQ$ where P and Q are some reflection matrices. If the displacements in the vector d, including the boundary degrees of freedom whose displacements are zero, are ordered in such a way that the j^{th} degree of freedom, $j = 1, 2,$ and 3, at the i^{th} node, $i = 1, ..., 4m$, is represented by d_k, the k^{th} component of d where $k = 3(i-1)+j$, then P and Q take the following forms

$$P = \begin{bmatrix} 0 & 0 & F & 0 \\ 0 & 0 & 0 & F \\ F & 0 & 0 & 0 \\ 0 & F & 0 & 0 \end{bmatrix}, \quad F = I_m \otimes D_z \qquad (1)$$

and

$$Q = \begin{bmatrix} 0 & E & 0 & 0 \\ E & 0 & 0 & 0 \\ 0 & 0 & 0 & E \\ 0 & 0 & E & 0 \end{bmatrix}, \quad E = I_m \otimes D_y \qquad (2)$$

where

$$D_y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad D_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix},$$

and I_m is the identity matrix of dimension m. Now by uniformly partitioning the matrix A into A_{ij}, $i, j = 1, 4$ (a matrix of block order 4) and by taking

$$X = \frac{1}{\sqrt{2}} \begin{bmatrix} I & 0 & -F & 0 \\ 0 & I & 0 & -F \\ F & 0 & I & 0 \\ 0 & F & 0 & I \end{bmatrix} \qquad (3)$$

and

$$Y = \frac{1}{\sqrt{2}} \begin{bmatrix} I & -E & 0 & 0 \\ E & I & 0 & 0 \\ 0 & 0 & I & -E \\ 0 & 0 & E & I \end{bmatrix}, \qquad (4)$$

it is not difficult to show that

$$\tilde{A} = Y^T X^T A X Y$$

$$= \tilde{A}_1 \oplus \tilde{A}_2 \oplus \tilde{A}_3 \oplus \tilde{A}_4 \qquad (5)$$

where $\oplus$ is the direct sum and

$$\tilde{A}_1 = (A_{11}+A_{13}F) + (A_{12}+A_{14}F)\,E\ ,$$

$$\tilde{A}_2 = (A_{22}+A_{24}F) - (A_{21}+A_{23}F)\,E\ ,$$

$$\tilde{A}_3 = (A_{33}-A_{31}F) + (A_{34}-A_{32}F)\,E\ ,$$

$$\tilde{A}_4 = (A_{44}-A_{42}F) - (A_{43}-A_{41}F)\,E\ .$$

To take advantage of this uncoupling structure resulting from the decomposition in solving the linear system $Ad = f$, we solve instead the transformed system $\tilde{A}\tilde{d} = \tilde{f}$ or, equivalently,

$$(Z^T AZ)(Z^T d) = (Z^T f) \qquad (6)$$

where $Z = XY$. In other words, this decomposition allows the solution process of the linear system to be handled in parallel by all four clusters on the Cedar, one subsystem per cluster. Note that the matrix Z is orthogonal because X and Y are both orthogonal. Therefore, the retrieval of the solution d from $\tilde{d}$, $Z^T d = \tilde{d}$, does not require a linear system solver since d is simply equal to $Z\tilde{d}$. Furthermore, by inspecting the nonzero entries of X and Y it is clear that this step does not even involve any arithmetic multiplications or divisions except the factor of 2, which can be dropped from both sides of the equality sign of Eq. (6) as well. This decomposition method is thus multiplication (division) free in both the decomposition and retrieval steps. Note also that the decomposition is performed by similarity transformation. Accordingly, this approach is immediately applicable for solving the eigenvalue problem $Av = \lambda v$ or the generalized eigenvalue problem $Av = \lambda Bv$ if the matrix B has the same reflexive nature as A.

4.2. Performance results on the Cedar.

The finite element discretization described in the previous subsection yields an algebraic linear system of order 7344 which includes the boundary degrees of freedom whose displacements are zero. The linear system is decomposed into 4 independent subsystems using the SAS approach. The bandwidth of each subsystem depends only on the ordering of the nodes in subdomain I. Recall that once the ordering of nodes in subdomain I is determined, the ordering of nodes in all the other subdomains is fixed. In our experiments, a natural ordering of the nodes in subdomain I that yields a half-bandwidth, (bandwidth+1)/2, of the stiffness matrix of each SAS-decomposed subsystem equal to 132 is employed.

In the following, we report the performance related to the three important steps stated in the Cedar algorithms presented in Section 3. These three steps account for more than 80% of the total CPU time consumed on the Alliant FX/80 when only one CE is employed. The performance of the algorithms is tested using a direct band solver which is parallelized and vectorized during the Gaussian elimination process. The vectorization in the elimination step is performed on the update of the entries in a given row and the parallelization is carried out by updating multiple rows in parallel, one row per processor. We compare the results obtained from the three multicluster algorithms run on the 4×8 Cedar (the first number is the number of clusters and the second the number of processors in each cluster) with 64 MB of global memory and 32 MB of cluster memory in each cluster.

The wall-clock time spent in the three main steps, including data transfer from the global memory to each cluster memory or from one cluster memory to the others, if any, for the multicluster algorithms SDOGC, CTSKDB, and DOCM is shown in Table 1. As seen from Table 1, the best 4-cluster algorithm is SDOGC which runs 1.46 times as fast as CTSKDB and 1.27 times as fast as SDOGM, using all 32 processors

Table 1
Wall-clock time in seconds

CEDAR	No. of Proc-	Algorithm		
	essors	SDOGM	CTSKDB	SDOGC
1 * 1	1	167.69	62.75	66.94
1 * 2	2	91.51	35.59	41.70
1 * 3	3	66.13	26.81	32.72
1 * 4	4	53.51	22.35	28.81
1 * 6	6	40.69	18.29	25.27
1 * 8	8	34.47	16.52	23.69
2 * 8	16	17.76	14.70	11.95
4 * 8	32	9.52	10.98	7.50

Table 2
Concurrency speedup for each individual algorithm

CEDAR	No. of Proc-essors	Algorithm		
		SDOGM	CTSKDB	SDOGC
1 * 1	1	1.00	1.00	1.00
1 * 2	2	1.83	1.76	1.61
1 * 3	3	2.54	2.34	2.05
1 * 4	4	3.13	2.81	2.32
1 * 6	6	4.12	3.43	2.65
1 * 8	8	4.86	3.80	2.83
2 * 8	16	9.44	4.27	5.60
4 * 8	32	17.14	5.71	8.93

available. The superiority of employing the multicluster approach is clearly shown in the last three rows in Table 1, in which the wall-clock time spent using 1 clusters of all 8 processors is reduced by a factor of more than 1.94 using 2 clusters and reduced by a factor of more than 3.15 when all 4 clusters are used for algorithms SDOGC and SDOGM. The slowdown of the reduction in the wall-clock time for the algorithm CTSKDB using more than one cluster is mainly due to the data transfer from the master cluster to the other cluster(s). Note that there is no data transfer either between the global memory and the cluster memories or between cluster memories in CTSKDB when only one cluster is employed. This is also the reason why CTSKDB is the most efficient one among these three algorithms when only one cluster is used. For the algorithm CTSKDB to be more efficient, the speed of data transfer must be improved.

The concurrency speedup, defined to be the ratio of the wall-clock time spent using one CE to that using multiple CE's, for each individual algorithm is shown in Table 2. From Table 2, we notice that the parallel execution of the algorithm SDOGM yields a concurrency speedup of 17.14, a satisfactory parallel performance. Although SDOGM is not the most efficient algorithm on the Cedar machine in this experiment, this high speedup clearly indicates the potential of the multicluster algorithm for a shared memory multicluster processor that has very small cluster memory or has no cluster memory at all. Figures 3 and 4 plot the results of the wall-clock time and the concurrency speedups respectively for each of the three algorithms.

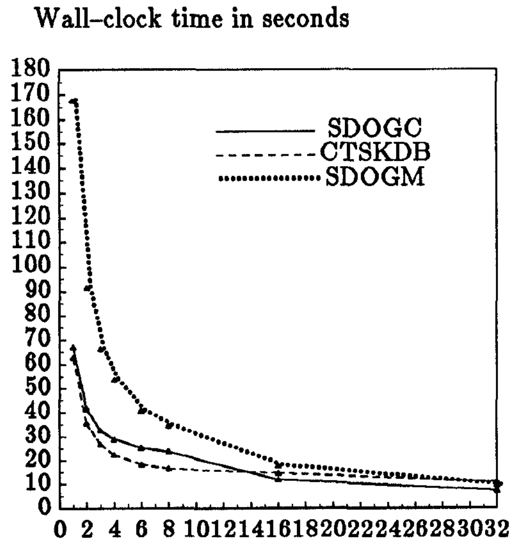

Fig.3. Performance of the SAS method (wall-clock time).

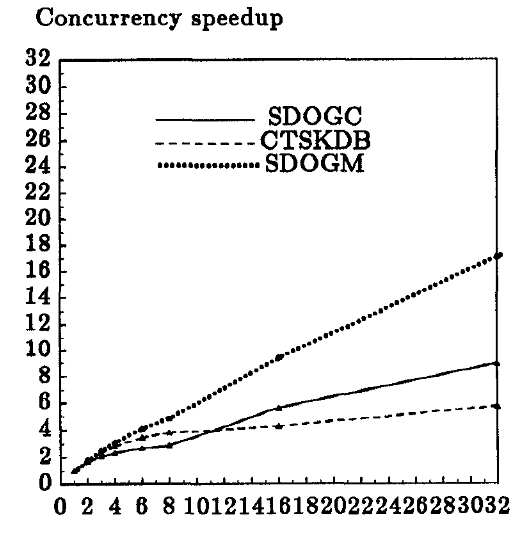

Fig.4. Performance of the SAS method (concurrency speedup).

5 Summary

In this paper, we have presented three multicluster algorithms for solving linear systems obtained from the application of the SAS decomposition method to 3D elasticity problems with reflexive symmetry. The algorithms all employ the SAS approach to decompose the reflexive linear system into several smaller and independent subsystems and solve them via a direct linear system solver. The main difference among these algorithms lies in the use of the global memory and the cluster memories of the Cedar multicluster processor. The advantages and disadvantages of these algorithms and their potential have been discussed. The parallel performance of these algorithms has also been tested on the 4×8 Cedar multiprocessor using a 3D elasticity problem with reflexive symmetry.

Among the three algorithms tested, the one that stores and solves the linear system in the global memory yields reasonably good concurrency speedup. Although it is not currently the most efficient one in our experiments on the current Cedar configuration, this is an indication to the high potential of this algorithm for multicluster parallel computers that have a large shared global memory and small cluster memories, In the case of a small shared global memory and large cluster memories, one can still take full advantage of the architecture by generating and assemblying the subsystems directly in cluster memories in parallel, one subsystem per cluster. This is likely to be the most efficient one because most of the the data transfer can be eliminated. The computer code employing this approach is currently under development.

6 Acknowledgments

This research was supported primarily by the U.S. Department of Energy under Grant No. DOE-DE-FG02-85ER25001, with additional support from the National Science Foundation under Grant No. NSF-CCR-87-17942. The author would like to thank Rudolf Eigenmann, Jay Hoeflinger, and Greg Jaxon for many helpful discussions on the Cedar Fortran language.

REFERENCES

[BlKa66]
S. Blaszkowiak and Z. Kaczkowski, Iterative Methods in Structural Analysis (translated by A. Kacner and Z. Olesiak), Pergamon Press, Oxford, 1966.

[BrDM88]
F. Brezzi, C.C. Douglas, and L.D. Marini, "Parallel Domain Reduction Method," IBM Research Report RC 13778, 1988.

[Breb83]
C.A. Brebbia, "Basic Principles," in C.A. Brebbia, T. Futagami, and M. Tanaka (editors), Boundary Elements, Spring-Verlag, New York, 1983. *Proceedings of the Fifth International Conference*, pp.3–28, Hiroshima, Japan, November 1983.

[CGPW89]
T. Chan, R. Glowinski, G.A. Meurant, J. Periaux, and O. Widlund (editors), Domain Decomposition Methods, SIAM, Philadelphia, 1989. Proceedings of the Second International Symposium on Domain Decomposition Methods, Los Angeles, California, January, 1988.

[Chen88]
H-C. Chen, The SAS Domain Decomposition Method for Structural Analysis, *CSRD Tech. Rept 754*, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1988.

[ChSa87]
H-C. Chen and A. Sameh, "Numerical Linear Algebra Algorithms on the Cedar System," Parallel Computations and Their Impact on Mechanics (Ed. A.K. Noor), *The American Society of Mechanical Engineers*, AMD-Vol.86, 1987, pp. 101–125.

[ChSa89a]
H-C. Chen and A. Sameh, "A Matrix Decomposition Method for Orthotropic Elasticity Problems," *SIAM J. Matrix Anal. Appl.*, Vol.10, No.1, pp. 39–64, January 1989.

[ChSa89b]
H-C. Chen and A. Sameh, "A Domain Decomposition Method for 3D Elasticity Problems," Applications of Supercomputers in Engineering: Fluid Flow and Stress Analysis Applications (Ed. C.A. Brebbia and A. Peters), Computational Mechanics Publications, Southampton University, UK, Sep. 1989, pp. 171–188.

[DKLS86]
E. Davidson, D. Kuck, D. Lawrie, and A. Sameh, "Supercomputing Tradeoffs and the Cedar System," *CSRD Tech. Rept 577*, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1986.

[DoSm88]
C.C. Douglas and B.F. Smith, "Using Symmetries and Antisymmetries to Analyze a Parallel Multigrid Algorithm: the Elliptic Boundary Value Problem Case," *SIAM J. Numer. Anal.*, Vol.26, No.6, pp. 1439–1461, December 1989.

[EHJD90]
R. Eigenmann, J. Hoeflinger, G. Jaxon, and D. Padua, "Cedar Fortran and Its Compiler," *CSRD Tech. Rept 966*, Center for Supercomputing Research and Development, University of Illinois at Urbana–Champagin, 1990.

[GGMP88]
R. Glowinski, G.H. Golub, G.A. Meurant, and J. Periaux (editors), Domain Decomposition Methods for Partial Differential Equations, SIAM, Philadelphia, 1988. Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Paris, France, January 1987.

[Guzz87]
M.D. Guzzi, "Cedar Fortran Programming Handbook," *CSRD Tech. Rept 601*, Center for Supercomputing Research and Development, University of Illinois at Urbana–Champaign, 1987

[Melo63]
R.J. Melosh, "Structural Analysis of Solids," *ASCE J. of Structural Division*, Vol.89, No.ST4, August, 1963, pp. 205–223.

[NoPe87a]
A.K. Noor and J.M. Peters, "Preconditioned Conjugate Gradient Techniques for the Analysis of Symmetric Anisotropic Structures", *IJNME*, Vol.24, pp. 2057–2070, 1987.

[NoPe87b]
A.K. Noor and J.M. Peters, "Model-size Reduction for the Nonlinear Dynamic Analysis of Quasi-symmetric Structures," *Engineering Computations*, Vol.4, No.4, pp. 178–189, September 1987.

[Sege76]
L.J. Segerlind, Applied Finite Element Analysis, Wiley, New York, 1976.

[Szil74]
R. Szilard, Theory and Analysis of Plates—Classical and Numerical Methods, Prentice Hall, New Jersey, 1974.

[TiGo70]
S.P. Timoshenko and J.N. Goodier, Theory of Elasticity, McGraw–Hill, New York, 1st ed. 1934, 2nd ed. 1951, 3rd ed. 1970.

[Wang53]
C.T. Wang, Applied Elasticity, McGraw–Hill, New York, 1953.

[WeJo87]
W. Weaver, Jr. and P.R. Johnson, Structural Dynamics by Finite Elements, Prentice–Hall, New Jersey, 1987.