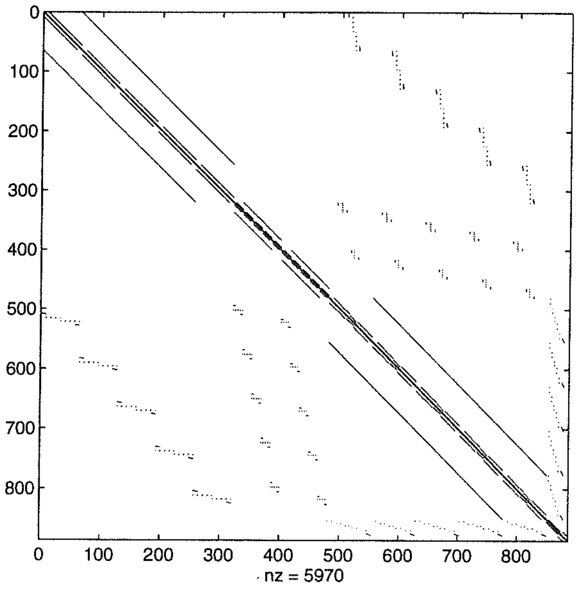
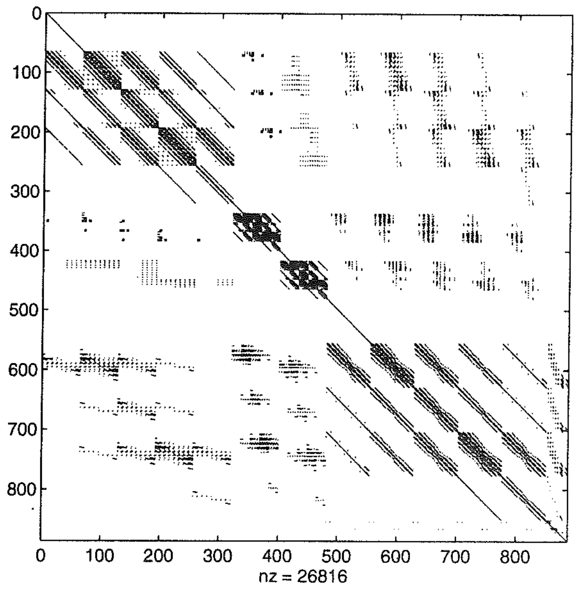
Centro Svizzero di
Calcolo Scientifico

Swiss Center for
Scientific Computing

Parallel Sparse Approximate Inverse Preconditioner

V. R. Deshpande
P. Messmer
M. J. Grote
W. B. Sawyer





TECHNICAL REPORT

TR-96-14

May 1996

Parallel Sparse Approximate Inverse Preconditioner

V. R. Deshpande[1] M. J. Grote[2]
P. Messmer[1] W. B. Sawyer[1]

May 1996

Abstract. A parallel implementation of a sparse approximate inverse (SPAI) preconditioner for distributed memory parallel processors (DMPP) is presented. The fundamental SPAI algorithm is known to be a useful tool for improving the convergence of iterative solvers for ill-conditioned linear systems. The parallel implementation (PARSPAI) exploits the inherent parallelism in the SPAI algorithm and the data locality on the DMPPs, to solve structurally symmetric (but non-symmetric) matrices, which typically arise when solving partial differential equations (PDEs). Some initial performance results are presented which suggest the usefulness of PARSPAI for tackling such large size systems on present day DMPPs in a reasonable time.
The PARSPAI preconditioner is implemented using the Message Passing Interface (MPI) and is embedded in the parallel library for unstructured mesh problems (PLUMP).

Keywords. SPAI, PARSPAI, PDEs, preconditioner, PLUMP, parallel architectures, DMPPs, MPI

This work was performed as part of the *Joint CSCS/NEC Collaboration in Parallel Processing* and will be presented at the *Third International Workshop on Parallel Algorithms for Irregularly Structured Problems (Irregular'96)*, Santa Barbara, USA.

[1] Swiss Center for Scientific Computing (CSCS/SCSC),
CH-6928 Manno, Switzerland
[2] Courant Institute of Mathematical Sciences, New York, NY 10012, USA

Contents

List of Figures

List of Tables

1 Introduction

We consider the linear system of equations

$$Ax = b, \quad x, b \in \mathbb{R}^n \ . \tag{1}$$

Here A is a large and sparse matrix and may be non-symmetric. Due to the size of A, direct solvers become prohibitively expensive because of the amount of work and storage required. As an alternative we consider iterative methods such as CGS, GMRES, BCG, and BI-CGSTAB applied to the normal equations [2]. Given the initial guess x_0, these algorithms compute iteratively new approximations x_k to the true solution $x = A^{-1}b$. The iterate x_m is accepted as a solution if the residual $r_m = b - Ax_m$ satisfies $\|r_m\|/\|b\| \leq \text{tol}$. In general, the convergence is not guaranteed or may be extremely slow. Hence, the original problem (1) must be transformed into a more tractable form, by applying a preconditioning matrix M either to the right or to the left of the linear system

$$AMy = b \ , \quad x = My \ , \quad \text{or} \quad MAx = Mb \ . \tag{2}$$

M should be chosen such that AM (or MA) is a good approximation of the identity I. As the ultimate goal is to reduce the total execution time, both the computation of M and the matrix-vector product My should be done in parallel. Since the matrix-vector product must be performed at each iteration, the number of nonzero entries in M should not greatly exceed that in A.

The most successful preconditioning methods in reducing solver iterations, e.g., incomplete LU factorizations or SSOR, are notoriously difficult to implement on a parallel architecture, especially for unstructured matrices. ILU, for example, can lead to breakdowns. In addition, ILU computes M implicitly, namely in the form $M = U_{\text{approx}}^{-1} L_{\text{approx}}^{-1}$, and its application therefore involves solving upper and lower triangular sparse linear systems, which are inherently sequential operations. Polynomial preconditioners with $M = p(A)$, on the other hand, are inherently parallel, but do not lead to as much improvement in the convergence as ILU. For a complete discussion see [2].

A relatively new approach is to minimize $\|AM - I\|$ in the Frobenius norm, an approach with inherent parallelism, because the columns m_k of M can be computed independently of one another. Indeed, since

$$\|AM - I\|_F^2 = \sum_{k=1}^{n} \|(AM - I)e_k\|_2^2 \ , \tag{3}$$

the solution of (3) separates into n independent least squares problems

$$\min_{m_k} \|Am_k - e_k\|_2 \ , \quad k = 1, \ldots, n \ , \tag{4}$$

where $e_k = (0, ..., 0, 1, 0, ..., 0)^T$. Thus, we can solve (4) in parallel and obtain an explicit approximate inverse M of A. If M is sparse, (4) reduces to n small least squares problems, which can be solved very quickly [9, 14]. Thus M is computed explicitly and is then applied with a sparse matrix-vector multiplication — an operation which can also be performed in parallel.

The difficulty lies in determining a good sparsity structure of the approximate inverse, otherwise the solution of (4) will not yield an effective preconditioner. Yeremin et al. compute a factorized sparse approximate inverse [13, 12, 14], but only consider fixed sparsity patterns. Simon and Grote [9] solve (4) explicitly, but only allow for a banded sparsity pattern in M. The approach of Cosgrove, Diaz, and Griewank [5], Chow and Saad [4], and Grote and Huckle [11, 10] all suggest methods which capture the sparsity pattern of the main entries of A^{-1} automatically and at a reasonable cost, but stop short of an actual parallel implementation. Gould and Scott [8] present results of a simulated parallel implementation based on a shared-memory paradigm.

In this paper we build on the sequential version of Grote and Huckle [11, 10] and assume that A is (nearly) structurally symmetric (true for partial differential equations solved on meshes). The resulting PARSPAI algorithm offers a high degree of data locality, and its implementation on a distributed memory parallel processor (DMPP) architecture with the Message Passing Interface (MPI) [15] is described in detail. In Sect. 2 we review the SPAI algorithm, which computes a sparse approximate inverse of A. In Sect. 3 we discuss some of the numerous considerations in the parallel implementation. We briefly describe the PARSPAI algorithm in Sect. 4 and present indications about the preconditioner's quality and performance results on a DMPP in Sect. 5. Finally we draw some conclusions about its usefulness in the parallel solution of partial differential equations PDEs on very large unstructured meshes.

2 SPAI Algorithm Review

The SPAI algorithm is explained in detail in [11, 10]; we summarize it here briefly for the sake of completeness.

The matrix M is the solution of the minimization problem (4). Since the columns of M are independent of one another, the algorithm for only one of them, m_k, is sufficient. An initial sparsity of M is assumed, i.e., $\mathcal{J}$ is the set of indices j such that $m_k(j) \neq 0$. In reality the initial $\mathcal{J}$ could be very simple, e.g., $\mathcal{J} = \{k\}$. The reduced vector of unknowns $m_k(\mathcal{J})$ is denoted by $\hat{m}_k$. Next, let $\mathcal{I}$ be the set of indices i such that $A(i, \mathcal{J})$ is not identically zero, denote the resulting submatrix $A(\mathcal{I}, \mathcal{J})$ by $\hat{A}$, and define $\hat{e}_k = e_k(\mathcal{I})$. Solving (4) for m_k is equivalent to solving

$$\min_{\hat{m}_k} \|\hat{A}\hat{m}_k - \hat{e}_k\|_2 \tag{5}$$

for $\hat{m}_k$. The $|\mathcal{I}| \times |\mathcal{J}|$ least squares problem (5) is extremely small because A and M are very sparse matrices. Equation (5) is solved, e.g., with the QR decomposition (among other methods — see [7] for details) for each $k = 1, \ldots, n$ and $m_k(\mathcal{J}) = \hat{m}_k$. This yields an approximate inverse M, which minimizes $\|AM - I\|_F$ for the given sparsity structure.

The sparsity pattern $\mathcal{J}$ is then augmented to obtain a more effective preconditioner by reducing the current error $\|AM - I\|_F$, i.e., reducing the norm of the residual

$$r = A(.,\mathcal{J})\,\hat{m}_k - e_k \; . \tag{6}$$

If $r = 0$, m_k is exactly the k-th column of A^{-1} and cannot be improved upon. Otherwise, since A and m_k are sparse, most components of r are still zero. $\mathcal{L}$ is the set of remaining indices ℓ for which $r(\ell) \neq 0$ (typically equal to $\mathcal{I}$). To every $\ell \in \mathcal{L}$ corresponds an index set $\mathcal{N}_\ell$, which consists of the indices of the nonzero elements of $A(\ell, .)$ that are not in $\mathcal{J}$ yet. The potential new candidates to augment $\mathcal{J}$ are contained in

$$\tilde{\mathcal{J}} = \bigcup_{\ell \in \mathcal{L}} \mathcal{N}_\ell \; . \tag{7}$$

New indices $j \in \tilde{\mathcal{J}}$ are selected to achieve the largest possible reduction in $\|r\|_2$.

Grote and Huckle [11, 10] consider for each $j \in \tilde{\mathcal{J}}$ the one-dimensional minimization problem

$$\min_{\mu_j} \|r \,+\, \mu_j A e_j\|_2 \; , \tag{8}$$

whose solution is $\mu_j = -\frac{r^T A e_j}{\|Ae_j\|_2^2}$. They then calculate for each j the 2-norm ρ_j of the new residual $r + \mu_j A e_j$, namely,

$$\rho_j^2 = \|r\|_2^2 - \frac{(r^T A e_j)^2}{\|Ae_j\|_2^2} \; , \tag{9}$$

and take those j which lead to smallest ρ_j, e.g., those whose ρ_j is less than the mean $\bar{\rho}$.

Gould and Scott in [8] suggest the more precise but expensive minimization of the

$$\min_{z,\mu_j} \|Az \,-\, e_k \,+\, \mu_j A e_j\|_2 \; . \tag{10}$$

Using the augmented set of indices $\mathcal{J}$, the sparse least squares problem (4) is solved again. This yields a better approximation m_k of the k-th column of A^{-1}. This process is repeated for each $k = 1, \ldots, n$ until the residual satisfies a prescribed tolerance $\|r\|_2 \leq \epsilon$ or a maximum amount of fill-in f_i has been reached in m_k . [1]

The SPAI (Sparse Approximate Inverse) Algorithm:

For every column m_k of M:

(a) Choose an initial sparsity $\mathcal{J}$, e.g., $\mathcal{J} = \{k\}$.

(b) Compute the row indices $\mathcal{I}$ of the corresponding nonzero entries and the QR decomposition (5) of $A(\mathcal{I}, \mathcal{J})$. Then compute the solution m_k of the least squares problem (4), and its residual r given by (6).

While $\|r\|_2 > \varepsilon$ and $|\mathcal{J}| \leq f_i$:

(c) Set $\mathcal{L}$ equal to the set of indices ℓ for which $r(\ell) \neq 0$.

(d) Set $\tilde{\mathcal{J}}$ equal to the set of all new column indices of A that appear in all $\mathcal{L}$ rows but not in $\mathcal{J}$.

[1]If the process is not stopped, the algorithm will eventually compute the k-th column of A^{-1} .

(e) For each $j \in \tilde{\mathcal{J}}$ solve the minimization problem (8).

(f) For each $j \in \tilde{\mathcal{J}}$ compute ρ_j given by (9), and delete from $\tilde{\mathcal{J}}$ all but the most profitable indices.

(g) Determine the new indices $\tilde{\mathcal{I}}$ and update the QR decomposition using (5). Then solve the new least squares problem, compute the new residual $r = Am_k - e_k$, and set $\mathcal{I} = \mathcal{I} \cup \tilde{\mathcal{I}}$ and $\mathcal{J} = \mathcal{J} \cup \tilde{\mathcal{J}}$.

3 Considerations for Parallel Implementation

While the least squares minimizations in (4) for each k clearly can be performed independently on different processing elements (PE)s, each PE must have access to the data required to solve its subproblem. Thus the parallel implementation on a shared-memory machine is more straightforward than that on a DMPP, on which the algorithm is not communication-free or even necessarily minimal in communication.

To consider problems of interest with n very large (e.g., $> 100{,}000$) on a DMPP, it must be assumed that the matrix A and all the n-vectors used in the calculation are distributed over all PEs. An expedient way is to distribute the vector element-wise over all PEs, and distribute the rows of A in the same manner.

One possible approach is the use of the ITPACK [2] format in distributed form (DIS) (see Fig. 1) which is used in the Parallel Library for Unstructured Mesh Problems [3] PLUMP. This distribution assures that the kernel operation for the solver $y \leftarrow Ax$ can be implemented relatively simply, and that communication can be partially overlapped with computation.

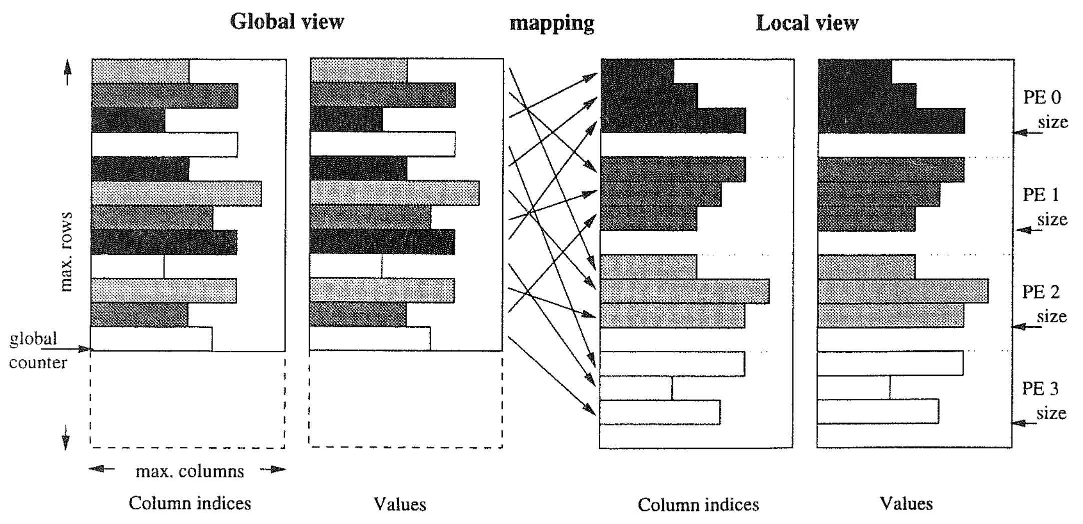


Figure 1: The distributed ITPACK storage assumes that there is a maximum number of matrix entries per row, and that the graph is adequately partitioned such that no PE fills its local two-dimensional array. In addition to the matrix value array, there is a corresponding array with column indices for each matrix entry.

Given A in DIS format, step **(d)** in the SPAI algorithm implies that the residuals r or at least the set $\mathcal{J}$ from any given PE's ongoing column m_k calculation has to be broadcasted to all other PEs, since for this PE it is not known a priori whether $r^T Ae_j$ in (9) will be non-zero, i.e., whether non-zero positions specified by the set $\mathcal{L}$ will correspond to non-zero entries in the j^{th} column of A. Since all PEs are concurrently working on a different m_k, this would mean a frequent all-to-all communication which would necessarily incur a large amount of unscalable overhead.

For a very large class of problems, namely the solution of partial differential equations over a (possibly unstructured) mesh, the above-mentioned problem is not as difficult to overcome as it may seem. The problem to be solved is to find the function values $u(x) \in \mathbb{R}$ at any point $x \in \mathbb{R}^n$ in the domain Ω, where u fulfills

$$Lu = f \; . \tag{11}$$

Here $f(x)$ is defined in Ω, and suitable boundary conditions are applied on $\partial\Omega$. A feature of such problems is that, when Ω is discretized into a finite number of mesh points and the operator L described through a discrete operator, they lead to a nearly or entirely *structurally symmetric* matrix A, i.e., $A(i,j) \neq 0 \rightarrow A(j,i) \neq 0$, even though $A(i,j) \neq A(j,i)$. Such problems provide an opportunity to efficiently determine which columns have to be evaluated, since now the index set $\mathcal{N}_l$ from (7) consists of the indices of the nonzero elements of $A(.,l)$ (as well as $A(l,.)$) which all reside on one PE for given l. If, on the other hand, A is sparse but not inherently structurally symmetric, the matrix can be stored in a structurally symmetric form by storing a zero in $A(i,j)$ for $A(j,i) \neq 0$. This method requires additional memory (at most twice as much), but provides important graph information for later use. Usually in PDE problems, completing the structural symmetry is only necessary for boundary vertices and therefore comes at small cost of memory.

In fact, for such a PDE problem, for a given $\mathcal{J}$ the columns of the matrices which need to be evaluated correspond to the set of the first and second nearest neighbors in the connectivity graph of the matrix (see Fig. 2). As the set $\mathcal{J}$ grows in order to make the preconditioner more precise, the set $\mathcal{I}$ expands much like the propagation of a wavefront.

To ensure that the "wavefront" crosses a processor boundary as rarely as possible, we assume that the mesh (which is highly correlated with the connectivity graph in Fig. 2) is partitioned over the PEs in an efficient way, such that the load assigned to each PE is roughly even, and the number of cut edges (or the "surface area") of the individual partitions is minimum. As there is extensive literature on this subject [1, 16] we do not expound on this topic here. [2] If the mesh is partitioned cleverly on many PEs, the PE calculating m_k will, even in the worst case — e.g., if a graph vertex is on a PE boundary — communicates with one or a few other PEs, depending on how the wavefront progresses. Even if A is an ill-conditioned matrix from a PDE problem, the wavefront is not expected to propagate too far through the graph.

[2]Highly-efficient graph partitioning software is available in the public domain, e.g., MeTiS available from `http://www.cs.umn.edu/~karypis/metis/metis.html`

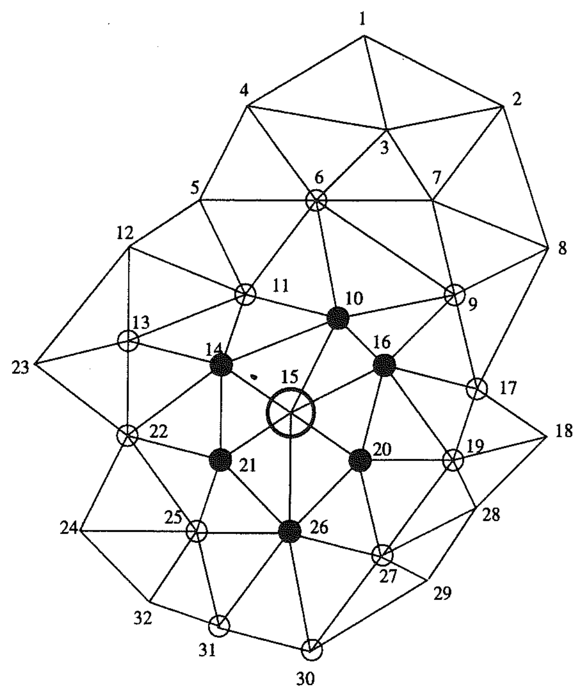


Figure 2: In structurally symmetric problems the vertices $\tilde{\mathcal{J}}$ to evaluate are the set of first (in black) and second (in white) neighbors of $\mathcal{J}$ (here $\mathcal{J}$ is the vertex marked 15).

In the continuous case, the solution of (11) with suitable boundary conditions can be represented in terms of its Green's function $G(x; y)$, which is defined by

$$u(x) = \int_\Omega G(x; y) f(y) dy \ . \tag{12}$$

The Green's function is the continous counterpart of A^{-1}, and the convolution in (12) corresponds to the product $A^{-1}b$ in the discrete case. Therefore, the k^{th} column of A^{-1} is a discrete approximation of $g(y) = G(y, x_k)$. Since $G(x, y)$ typically decays rapidly with increasing $\|x - y\|$, e.g., like $\mathcal{O}(1/\|x - y\|)$ for the Laplacian in $\mathbb{R}^3$, there is a reasonable hope that a sparse approximation of A^{-1} exists, when A comes from the discretization of a differential operator.

Firstly, the nearest neighbors of the graph vertex k are the best candidates for minimizing $\|Am_k - e_k\|_2$. Secondly, we expect the wavefront to fade out as it propagates, thus limiting the range of partitions (and thus PEs) which have to be addressed.

Tests show that it is sufficient to consider the first nearest neighbors of the set $\mathcal{J}$, i.e., the set $\mathcal{L} - \mathcal{J}$, as possible extensions to the set $\mathcal{J}$. If that set provides a preconditioner of insufficient quality, the second nearest neighbors will most likely be evaluated in subsequent propagations of the wavefront. This is the basis of the simplification in the parallel code which affects the quality and performance of the preconditioner compared to the original SPAI implementation.

4 Realization of PARSPAI

The PARSPAI algorithm has been implemented in FORTRAN 77 using MPI. It is embedded in the PLUMP library which provides a complete environment to describe a matrix A in DIS format, solve the system of equations for a given finite element mesh problem and later to support dynamic refinement of the mesh. The implementation of PARSPAI is modular enough to allow its usability with other distributed data formats with little effort. The current implementation of PARSPAI can be outlined as follows:

On every PE for a set of m_k of M:

```
(A) Choose an initial sparsity $\tilde{\mathcal{J}}$, e.g., $\tilde{\mathcal{J}} = \{k\}$.
While $\|r\|_2 > \epsilon$ and $|\mathcal{J}| \le f_i$ and number of iterations $\le \gamma$ :
    (B) Given the set $\tilde{\mathcal{J}}$, determine the new set $\tilde{\mathcal{I}}$ to update the matrix $\hat{A}$, using
        the local entries of $A$ or receiving them from other PEs. Set $\mathcal{I} = \mathcal{I} \cup \tilde{\mathcal{I}}$ and
        $\mathcal{J} = \mathcal{J} \cup \tilde{\mathcal{J}}$.
    (C) Solve the least squares problem (5) and set $\mathcal{L}$ equal to the set of indices $\ell$ for
        which $r(\ell) \neq 0$.
    (D) Send the residual $r = Am_k - e_k$ and the set $\mathcal{L}$ to other PEs requiring them to
        compute $\rho_j$.
    (E) Determine the local set $\tilde{\mathcal{J}}$ and for each $j \in \tilde{\mathcal{J}}$ compute $\rho_j$ given by (9). Gather
        $\rho_j$ from other PEs involved and compute $\bar{\rho}$.
    (F) Set $\tilde{\mathcal{J}} = \{j \in \tilde{\mathcal{J}} | \rho_j \le \beta \cdot \bar{\rho}\}$
```

In the above description, γ denotes the maximum number of graph extensions for one m_k.

The PEs which receive the residual r and the set $\mathcal{J}$ determine independently the sets $\mathcal{N}_l$ and the solutions of the minimization problem (8). Where SPAI increases $\tilde{\mathcal{J}}$ at most by a fixed number s during one iteration, PARSPAI considers all j with $\rho_j \le \beta \cdot \bar{\rho}$ for a given parameter β. This allows to determine $\tilde{\mathcal{J}}$ on the involved PEs and avoids additional communication of $\tilde{\mathcal{J}}$.

In the above procedure, steps **(B)** and **(D)** involve exchange of variable amounts of information among the different PEs. This can be realized by using the MPI function `MPI_Alltoallv`. But as mentioned earlier, only nearest neighbors are considered for graph extension. Hence a better approach would be to determine the subset of all PEs which actually require the requisite information and then exchange it using MPI point-to-point communication primitives. The current implementation follows this approach and to improve performance non-blocking send and receive operations are used. Although no comparison has been made, the performance of the two approaches may vary, depending on the underlying MPI library.

Substantial saving in computational effort could be achieved by solving the least squares problem only for the updated part of $\hat{A}$ [11, 10] instead of solving the problem for the entire updated matrix as in the current implementation.

Table 1: Number of iterations for ORSIRR_2, using CGS and varying ϵ: unpreconditioned ($M=I$) and preconditioned ($0.1\leq\epsilon\leq0.5$, $\gamma=4$), comparison between SPAI and PARSPAI.

		PARSPAI			
	SPAI	1 PE	2 PE	4 PE	8 PE
$M=I$	653	722	868	691	881
$\epsilon=0.5$	70	55	55	58	71
$\epsilon=0.4$	41	41	37	40	54
$\epsilon=0.3$	32	33	33	32	46
$\epsilon=0.2$	18	30	29	32	45
$\epsilon=0.1$	–	27	29	29	44

Table 2: Number of iterations for ORSIRR_2, using CGS for varying γ ($\epsilon=0.1$)

	1 PE	2 PE	4 PE	8 PE
$\gamma=2$	148	151	165	123
$\gamma=3$	55	63	55	95
$\gamma=4$	27	29	29	44
$\gamma=5$	17	18	22	27

5 Results and Discussion

The implementation of PARSPAI as per its quality and performance was investigated on a wide variety of problems, including test matrices from the Harwell-Boeing Collection [6]. The NEC Cenju-3, a distributed-memory machine with up to 128 nodes, each having a R4400 processor and 64MB memory was used for this purpose with the NEC-CSCS MPI library.

The following section discusses the results of three numerical experiments done on specific test cases of the Harwell-Boeing Collection and on sparse matrices as resulting from PDEs on a mesh.

Table 1 shows the convergence characteristics for the ORSIRR_2 886 × 886 matrix from an oil reservoir simulation, for SPAI (taken from [11, 10]) and PARSPAI using CGS as the solver. The initial guess was always $x_0=0$, $\tilde{r}_0=r_0=b$, and the stopping criterion

$$\frac{\|b-Ax_m\|_2}{\|b\|_2}\leq 10^{-8}, \quad x_m=My_m.$$

In line with the sequential SPAI algorithm, the number of iterations decreases as ϵ is reduced, confirming the robustness of the algorithm with respect to this parameter.

As per the discussion in Sec. 3, the performance of PARSPAI depends on the maximum number of graph extensions γ. The numerical experiments (Tab. 2) confirm that increasing γ reduces the number of iterations.

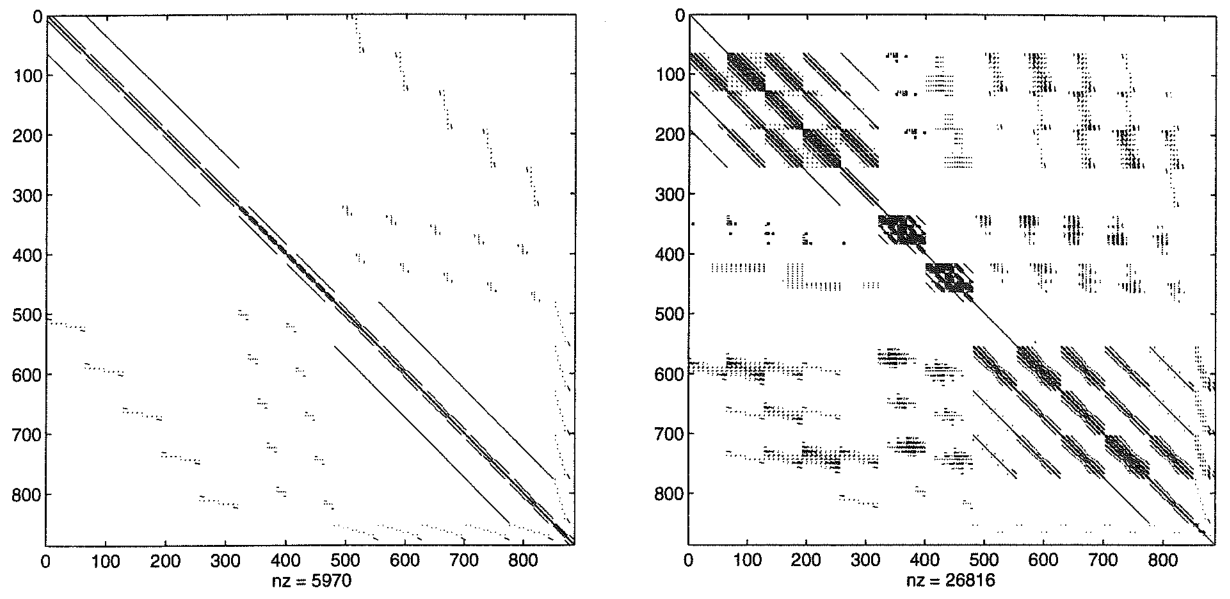


Figure 3: ORSIRR_2 test case from the Harwell-Boeing collection: The matrix itself (left) and the preconditioner as computed by PARSPAI with $\gamma=4$ and $\epsilon=0.4$.

The condition number obtained using PARSPAI ($\text{cond}_2(AM)=84.8$) compare favorably with that of SPAI ($\text{cond}_2(AM)=74.2$) for the same value of $\epsilon=0.4$. The sparsity pattern and the amount of fill-in however varies considerably as seen in Fig. 3, reflecting again the different approaches of augment $\tilde{\mathcal{J}}$.

The quality of the parallel preconditioner was also assessed by evaluating the eigenvalue spectrum of the preconditioned system AM. Since the methods to determine $\tilde{\mathcal{J}}$ are not identical, the eigenvalue spectrum of SPAI and PARSPAI differ, as shown in Fig. 4.

The speedup for building the preconditioner using PARSPAI for ORSIRR_2 is shown in Fig. 5. In spite of the small size of the test matrix, the results indicate good scaling behavior of PARSPAI. Since the number of iterations remains fairly constant with an

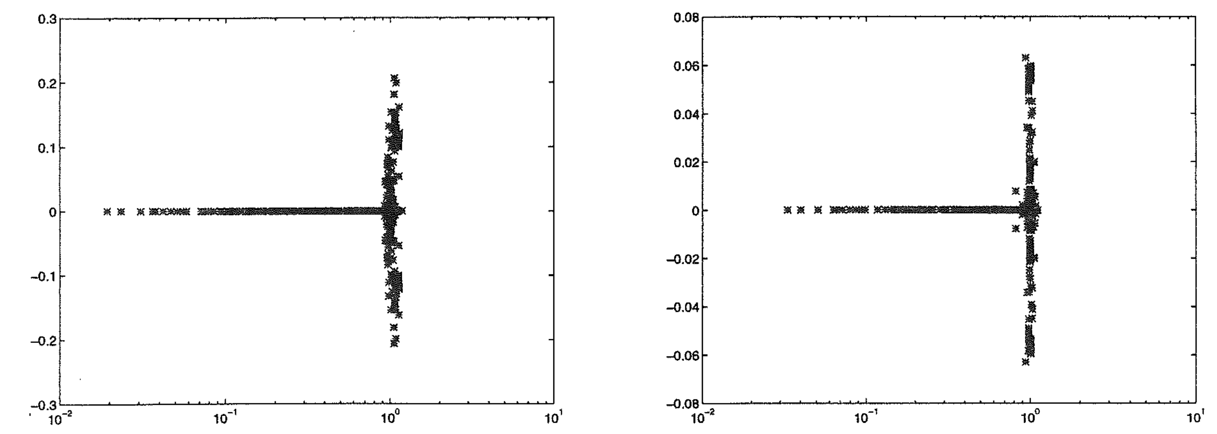


Figure 4: The eigenvalue spectra of AM_{SPAI} (left) and AM_{PARSPAI} (right) showing the similar quality of both SPAI and PARSPAI ($\epsilon=0.4$ and $\gamma=4$) .

Table 3: Number of iterations for SHERMANx, using CGS and varying γ (ϵ=0.1).

	SHERMAN1	SHERMAN3	SHERMAN5
γ=2	71	428	72
γ=4	40	254	58

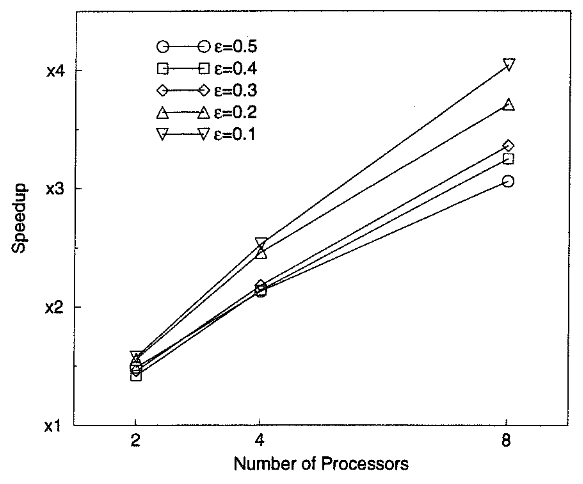


Figure 5: Speedup for building the preconditioner of the ORSIRR_2 matrix for $0.1 \leq \epsilon \leq 0.5$ ($\gamma = 4$).

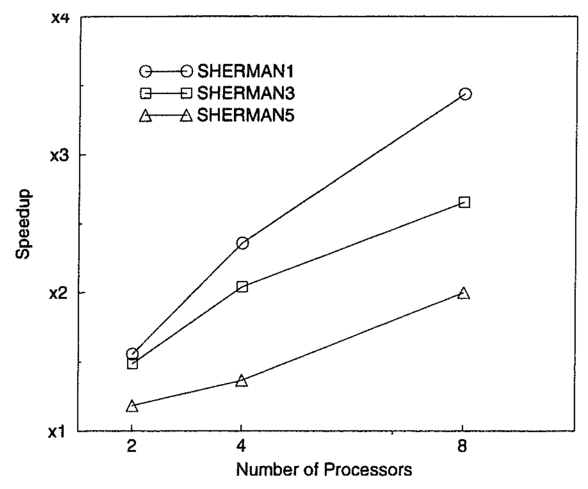


Figure 6: Speedup for building the preconditioner of the SHERMANx matrices for ϵ=0.1, $\gamma = 4$

increase in the number of PEs (Tab. 1), and because of the known scaling behavior of the matrix-vector product in the CGS solver [3], a good speedup in the preconditioning phase translates into reduced overall execution time.

As problem of medium size, the SHERMANx black oil simulators were chosen. The considered set consists of

SHERMAN1: a black oil simulator, shale barrier, $10 \times 10 \times 10$ grid, 1 unknown, of size n=1,000 and with nz=3,750 nonzero elements.

SHERMAN3: a black oil, IMPES simulation, $35 \times 11 \times 13$ grid, 1 unknown, n=5,005 and nz=20,033.

SHERMAN5: a fully implicit black oil simulator, $16 \times 23 \times 3$ grid, 3 unknowns, n=3,312 and nz=20,793.

Table 3 indicates the convergence results for CGS solver and varying γ on 8 PEs. Again, the dependency on the parameter γ is significant. Speedup for all three matrices is shown in Fig. 6, indicating the good scaling behavior for this size of problems.

The scaling behavior of the preconditioner in these test cases is promising for tackling larger size systems which were investigated and the results are described in [3]. A good partitioning of the mesh, however, is important to exploit the data locality on each processor and to ensure a good compute-to-communication ratio on the DMPPs. The

performance of the preconditioner could be substantially improved by optimizing the communication, tuning the different parameters, and by improving the basic algorithm used in PARSPAI, as suggested in Sec. 4. The results in this paper are indicative of the synergistic coupling of the preconditioner to the underlying data structure for achieving good performance on large size problems.

6 Conclusions

Iterative methods themselves are not difficult to parallelize or necessarily communication-bound on parallel machines, as they only require vector operations and global communication of single values for scalar products and norms. Parallelizing the required matrix-vector product and calculating a preconditioner are the difficult tasks involved. By limiting ourselves to non-symmetric matrices which are structurally symmetric, e.g., those which result from the solution of PDEs on a computational mesh, we have exploited the data locality and the inherent parallelism in the SPAI algorithm.

With the PLUMP library, in which PARSPAI is integrated, the matrix-vector product and the preconditioner are provided in a transparent way. We have presented some initial benchmarks which indicate that parallel iterative solvers along with the PARSPAI preconditioner will be able to tackle very large and ill-conditioned problems beyond the reach of single processor machines and current sparse direct solvers.

Acknowledgments We graciously acknowledge the NEC Corporation for financial support during the realization of this project.

References

[1] S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. Technical Report RNR-092-033, NASA Ames Research Center, Moffett Field, CA 94035, November 1992.

[2] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *TEMPLATES for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM Publications, 1994.

[3] Oliver Bröker, Vaibhav Deshpande, Peter Messmer, and William Sawyer. Parallel Library for Unstructured Mesh Problems. Technical Report CSCS-TR-96-15, Centro Svizzero di Calcolo Scientifico, CH-6928 Manno, Switzerland, May 1996.

[4] E. Chow and Y. Saad. Approximate Inverse Preconditioners for General Sparse Matrices. In *Proc. Colorado Conf. on Iterative Meth.*, 1994.

[5] J. D. F. Cosgrove, J. C. Diaz, and A. Griewank. Approximate Inverse Preconditionings for Sparse Linear Systems. *Intern. J. Computer Math.*, 14:91–110, 1992.

[6] I. Duff, R. G. Grimes, and J. Lewis. *User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)*. Available from http://math.nist.gov:80/MatrixMarket/collections/hb.html.

[7] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins, second edition, 1989.

[8] N. I. M. Gould and J. A. Scott. On Approximate-Inverse Preconditioners. Technical Report RAL 95-026, Rutherford Appleton Laboratory, 1995.

[9] M. Grote and H. Simon. Parallel Preconditioning and Approximate Inverses on the Connection Machine. In *Proc. of the Scalable High Performance Computing Conference (SHPCC), Williamsburg, VA*, pages 76–83. IEEE Comp. Sci. Press, 1992.

[10] Marcus J. Grote and Thomas Huckle. Parallel Preconditioning with Sparse Approximate Inverses. *SIAM Journal on Scientific Computing*. In press.

[11] Marcus J. Grote and Thomas Huckle. Effective Parallel Preconditioning with Sparse Approximate Inverses. In *Proc. SIAM Conf. on Parallel Processing for Scientific Comp., San Francisco*, pages 466–471. SIAM, 1995.

[12] L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin. Factorized Sparse Approximate Inverse (FSAI) Preconditionings for Solving 3D FE Systems on Massively Parallel Computers II. In R. Beauwens and P. de Groen, editors, *Iterative Meth. in Lin. Alg., Proc. of the IMACS Internat. Sympos., Brussels*, pages 311–312, 1991.

[13] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized Sparse Approximate Inverse Preconditionings. *SIAM Journal on Matrix Analysis and Applications*, 14(1):45–58, 1993.

[14] Ju. B. Lifshitz, A. A. Nikishin, and A. Yu. Yeremin. Sparse Approximate Inverse Preconditionings for Solving 3D CFD Problems on Massively Parallel Computers. In R. Beauwens and P. de Groen, editors, *Iterative Meth. in Lin. Alg., Proc. of the IMACS Internat. Sympos., Brussels*, pages 83–84, 1991.

[15] Message Passing Interface Forum. MPI: a message-passing interface standard (version 1.1). Revision of article appearing in the *International Journal of Supercomputing Applications*, 8(3/4):157–416, 1994, June 1995.

[16] C. Walshaw, M. Cross, M. Everett, and S. Johnson. A Parallelisable Algorithm for Partitioning Unstructured Meshes. In Alfonso Ferreira and Jose D. P. Rolim, editors, *Parallel Algorithms for Irregular Problems: State of the Art*, chapter 2, pages 25–44. Kluwer Academic Publishers, Dordrecht, Netherlands, August 1995. Collection of extended papers from Irregular'94 conference. [ISBN: 0-7923-3623-2].

1995

TR-95-03	C. Clémençon, K. Decker, V. Deshpande, A. Endo, J. Fritscher, N. Masuda, A. Müller, R. Rühl, W. Sawyer, B. J. N. Wylie, and F. Zimmermann: Tool-Supported Development of Parallel Application Kernels. (April 1995)
TR-95-04	Y. Seo, T. Kamachi, K. Suehiro, M. Tamura, A. Müller, and R. Rühl: Kemari: a Portable HPF System for Distributed Memory Parallel Machines. (June 1995)
TR-95-05	A. Endo and B. J. N. Wylie: Annai/PMA Instrumentation Intrusion Management of Parallel Program Profiling. (November 1995)
TR-95-06	P. Ackermann and U. Meyer: Prototypes for Audio and Video Processing in a Scientific Visualization Environment based on the MET++ Multimedia Application Framework. (June 1995)
TR-95-07	M. Guggisberg, I. Pontiggia and U. Meyer: Parallel Fractal Image Compression Using Iterated Function Systems. (May 1995)

1996

TR-96-01	W. P. Petersen: A General Implicit Splitting for Stabilizing Numerical Simulations of Langevin Equations. (February 1996)
TR-96-02	C. Clémençon, K. M. Decker, V. R. Deshpande, A. Endo, J. Fritscher, P. A. R. Lorenzo, N. Masuda, A. Müller, R. Rühl, W. Sawyer, B. J. N. Wylie, F. Zimmermann: Tools-supported HPF and MPI Parallelization of the NAS Parallel Benchmarks. (April 1996)
TR-96-03	B. J. N. Wylie and A. Endo: Annai/PMA Multi-level Hierarchical Parallel Program Performance Engineering. (April 1996)
TR-96-04	C. Clémençon, A. Endo, J. Fritscher, A. Müller, and B. J. N. Wylie: Annai Scalable Run-time Support for Interactive Debugging and Performance Analysis of Large-scale Parallel Programs. (April 1996)
TR-96-05	M. Roth: A Visualization System for Turbomachinery Flow. (April 1996)
TR-96-06	W. P. Petersen: Some evaluations of Random Number Generators in real*8 Format. (April 1996)
TR-96-07	Tetsuya Takaishi: Heavy quark potential and effective actions on blocked configurations. (April 1996)
TR-96-08	Norio Masuda and Frank Zimmermann: PRNGlib: A Parallel Random Number Generator Library. (May 1996)
TR-96-09	Marc C. Hohenadel and Pascal Pagny: X-OpenWave User's Manual. (May 1996)
TR-96-10	Edgar A. Gerteisen: Automatized Generation of Block-Structured Meshes for a Parametric Geometry. (May 1996)
TR-96-11	V. Deshpande, W. Sawyer, and D. W. Walker: An MPI Implementation of the BLACS. (May 1996)
TR-96-12	Ramprasad Sampath, Josef Fritscher, and Brian J. N. Wylie: Port of the Annai tool environment to workstation clusters. (May 1996)
TR-96-13	Paulo A. R. Lorenzo, Andreas Müller, Yoshimichi Murakami, and Brian J. N. Wylie: High Performance Fortran interfacing to ScaLAPACK. (May 1996)