ACES JOURNAL, VOL. 16, NO. 3, NOV. 2001

Implementing the Parallel Approximate Schur Complement Preconditioner

D.H. Malan and A.C. Metaxas
EUG, Department of Engineering, Cambridge, CB2 1TP, UK
dhm@eng.cam.ac.uk, acm@eng.cam.ac.uk

Abstract

A modified implementation of Saad's parallel approximate Schur complement preconditioner is presented that is shown to be faster than the original algorithm. This is evaluated for the solution of large sparse matrices as generated by the finite element time domain method. Results indicate that the preconditioner scales well with an increase in processors and problem size.

1 Introduction

The linear finite element time domain (FETD) method [1] is a powerful technique for high frequency electromagnetics analysis. It allows the use of an unstructured mesh to model the geometry, and can produce results simultaneously at a range of frequencies by using a pulse as excitation. It also does not suffer from the matrix ill conditioning that plagues the frequency domain finite element method when applied to closed resonant cavities.

$$\nabla \times \frac{1}{\mu} \nabla \times \mathbf{E} + \sigma_e \frac{\partial \mathbf{E}}{\partial t} + \epsilon \frac{\partial^2 \mathbf{E}}{\partial t^2} = -\frac{\partial \mathbf{J}_s}{\partial t}. \qquad (1)$$

In the present FETD implementation, the vector wave equation (1) is discretised using Whitney tetrahedral edge elements. $\mathbf{E}$ is the electric field, μ is magnetic permeability, ϵ and σ_e are respectively real permittivity and effective conductivity, and $\mathbf{J}_s$ represents free source current density. The standard Galerkin procedure is applied to obtain:

$$[S]e + [T_\sigma]\frac{\partial e}{\partial t} + [T_\epsilon]\frac{\partial^2 e}{\partial t^2} = s. \qquad (2)$$

e is a vector of unknown edge values, and s is a vector that is obtained from source current densities. The square matrices S, T_σ and T_ϵ are in general large and sparse, with around sixteen non-zero entries per row, and typically several hundred thousand rows for a household sized applicator.

To discretise the temporal derivatives the Newmark method is used, which yields the recurrence relation:

$$[A]e^{t+1} = [P_1]e^t + [P_2]e^{t-1} + k_1 s^{t+1} + k_2 s^t + k_3 s^{t-1}. \qquad (3)$$

The Newmark parameters and the time step length are used to derive the constants k_n, and are also used to obtain the sparse matrices A, P_1 and P_2 as linear combinations of the S, T_σ and T_ϵ matrices in equation 2. At each time step, the FETD method requires the solution of the sparse system in equation 3, with the right hand side assembled as a linear combination of the previous and present electric field and source vectors and the new source vector.

For large problems the A matrix can have in excess of a million rows, precluding the use of a direct solver. The matrix is also real, sparse, symmetric, positive definite and very well conditioned. An iterative solution method is therefore used, which can be rendered much more efficient with the right preconditioning. Ideally, a good preconditioner should significantly decrease the iterations required, introduce little computational overhead, and be capable of effective implementation in parallel.

The approximate Schur complement preconditioner proposed by Saad [7] presents an attractive choice. It is purely local in its setup phase, and in application requires exactly the same communication routines as for a distributed matrix vector product. In this paper the practical implementation of this preconditioner will be discussed, and its performance when used with the FETD method will be accessed.

2 Approximate Schur complement preconditioner

As shown in equation 3, at each time step of the FETD method a linear system of the form $Ax = b$ has to be solved with a new b vector. For solution in a parallel environment the system is distributed row-wise over p processors, as shown in equation 4. The rows held on each processor are considered as one domain, with matrix entries designated as *boundary* if they fall in columns of which the like numbered rows are held on different processors, or *local* otherwise. Rows of the linear system are called boundary if they contain any boundary entries.

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix} \qquad (4)$$

Each block-row of the above system is reordered so that the boundary rows occur last. Additionally, the columns of the whole system are rearranged so that within the diagonal block of each domain the columns with numbers corresponding to boundary rows are stored last. Bearing in mind that the local rows are per definition only connected to rows within that domain, the matrix blocks and vector sections can then be written as:

$$A_{ii} = \begin{bmatrix} A_{\alpha\alpha} & A_{\alpha\beta} \\ A_{\beta\alpha} & A_{\beta\beta} \end{bmatrix} \quad A_{\substack{ij \\ j\neq i}} = \begin{bmatrix} 0 & 0 \\ 0 & E_{ij} \end{bmatrix}$$
$$x_i = \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix} \qquad b_i = \begin{bmatrix} b_\alpha \\ b_\beta \end{bmatrix} \tag{5}$$

E_{ij} is a sparse matrix with number of rows equal to the number of boundary rows in domain i, and number of columns equal to the number of boundary rows in domain j. It represents the coupling between domains i and j, and will be zero for non-adjacent domains. $A_{\alpha\alpha}$, $A_{\alpha\beta}$, $A_{\beta\alpha}$ and $A_{\beta\beta}$ are sparse matrices, containing the self and inter-connecting matrix entries of the local and boundary rows. For the vectors, α and β subscripts indicate entries corresponding to local and boundary rows of the linear system, respectively.

On each domain, internal unknowns may be eliminated independently, leaving a reduced system in terms of the boundary unknowns. The diagonal blocks of the reduced system are the Schur complement matrices of the A_{ii} matrices, found as $A_{SC_i} = A_{\beta\beta} - A_{\beta\alpha}A_{\alpha\alpha}^{-1}A_{\alpha\beta}$. If the reduced system is multiplied on both sides by a diagonal block matrix consisting of the inverses of the Schur complement matrices, further algebra finally yields a system of the form [3]:

$$\begin{bmatrix} I & \tilde{A}_{SC_1}^{-1}E_{12} & \cdots & \tilde{A}_{SC_1}^{-1}E_{1p} \\ \tilde{A}_{SC_2}^{-1}E_{21} & I & \cdots & \tilde{A}_{SC_2}^{-1}E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{A}_{SC_p}^{-1}E_{p1} & \tilde{A}_{SC_p}^{-1}E_{p2} & \cdots & I \end{bmatrix} \begin{bmatrix} x_{\beta_1} \\ x_{\beta_2} \\ \vdots \\ x_{\beta_p} \end{bmatrix} = \begin{bmatrix} R_{SC_1}\tilde{A}_1^{-1}b_1 \\ R_{SC_2}\tilde{A}_2^{-1}b_2 \\ \vdots \\ R_{SC_p}\tilde{A}_p^{-1}b_p \end{bmatrix} \tag{6}$$

The ˜ sign indicates that approximations are used. $R_{SC_i} = [0 \;\; I]$ is a rectangular matrix that for domain i restricts a vector of values on that domain to the boundary values. To solve this linear system iteratively the left hand side needs not to be formed explicitly, only its effect in matrix-vector products is required. If an approximation of the $\tilde{A}_{SC_i}^{-1}$ matrix is implicitly available as an incomplete LU factorisation, the matrix vector product is found from:

$$y_{\beta_i} = x_{\beta_i} + \tilde{U}_{SC_i}^{-1}\tilde{L}_{SC_i}^{-1}\sum_{\substack{j=1 \\ j\neq i}}^{p} E_{ij}x_{\beta_j} \tag{7}$$

To compute this equation, the x_{β_j} values have to be retrieved from the processors where they are held. Indeed, the communication required is identical to that required to compute a full matrix vector product. The incomplete factors of the Schur complement matrix are obtained by doing an incomplete factorisation of the complete A_i matrix on each domain, and then (see Appendix) using only restricted portions of the factors: $\tilde{L}_{SC} = R_{SC}\tilde{L}_iR_{SC}^T$, and $\tilde{U}_{SC} = R_{SC}\tilde{U}_iR_{SC}^T$.

The approximate Schur complement preconditioner therefore first requires the calculation of an incomplete factorisation of the diagonal block on each domain. With this available, the application of the preconditioner can be summarised as [7]:

1. On each domain, receive a right hand side vector: $b_i = \begin{bmatrix} b_{\alpha_i} \\ b_{\beta_i} \end{bmatrix}$

2. Compute the right hand side of the approximate Schur complement system (eq. 6) on every domain from a forward and backward sweep of the incomplete factors, retaining only the boundary values of the result: $\tilde{b}_{SC_i} = R_{SC_i}\tilde{U}_i^{-1}\tilde{L}_i^{-1}\begin{bmatrix} b_{\alpha_i} \\ b_{\beta_i} \end{bmatrix}$

3. Solve the reduced system of equation 6 with a global iterative method, computing matrix vector products by exchanging boundary values and then using equation 7.

4. Exchange the final boundary values, and update the boundary section of the right hand side using: $b_{\beta_i}^u = b_{\beta_i} - \sum_{\substack{j=1 \\ j\neq i}}^{p} E_{ij}x_{\beta_j}$

5. Using the updated right hand side, compute the estimate of the solution on each domain as: $x_i = \tilde{U}_i^{-1}\tilde{L}_i^{-1}\begin{bmatrix} b_{\alpha_i} \\ b_{\beta_i}^u \end{bmatrix}$

An implementation of this preconditioner is available in the P-Sparslib library [6].

3 Fast approximate Schur complement implementation

Saad's implementation of the approximate Schur complement preconditioner in the P-Sparslib library follows the algorithm in the previous section very closely. Both the second and the last steps of the algorithm appear to require a forward and backward substitution with the entire $\tilde{L}_i$ and $\tilde{U}_i$ incomplete factors of the diagonal blocks, and are implemented as such in the P-Sparslib code. In fact, substantial computational savings can be effected by reusing information computed at the start. In the second step, the backward sweep with the $\tilde{U}_i$ factor can be terminated once the boundary values have been computed, thus allowing the local values found from the forward sweep to be retained. The calculations in the second step are rewritten as:

$$\begin{bmatrix} b_{f_i} \\ \tilde{b}_{SC_i} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & \tilde{U}_{SC_i}^{-1} \end{bmatrix} \tilde{L}_i^{-1} b_i \tag{8}$$

The vector b_{f_i} is used as temporary storage for the result of the forward sweep on the local variables. Since only the boundary section of the right hand side vector is changed in step four, in the final step the b_{f_i} values can be used to execute a forward sweep of only the boundary values, followed by a backward sweep of the entire system. The final step calculation can therefore be rewritten as:

$$\begin{bmatrix} x_{\alpha_i} \\ x_{\beta_i} \end{bmatrix} = \tilde{U}_i^{-1} \begin{bmatrix} I & 0 \\ 0 & \tilde{L}_{SC_i}^{-1} \end{bmatrix} \begin{bmatrix} b_{f_i} \\ b_{\beta_i}^{u} \end{bmatrix} \tag{9}$$

For a large matrix the boundary values can constitute less than ten percent of the local system. By this modified procedure almost an entire forward and backward sweep is eliminated per preconditioning step, allowing significantly faster execution.

4 Results

Typical matrices from applying the FETD method to a microwave cavity [4] are used to evaluate the preconditioner. The cavity is modelled at 2.45 GHz, and consists of a rectangular metal enclosure, several wavelengths in all dimensions, fed via a waveguide. Excitation is via a current sheet in the waveguide; all metal boundaries are treated as perfect conductors. An intricately shaped load material with permittivity $30 - 10j$ is placed inside the cavity.

An unstructured mesh of Whitney tetrahedral edge elements is used to discretise the domain. The matrix partitioning is done using the Metis [2] graph partitioning package, using vertex weighting to ensure the same number of non-zero entries per domain, and edge weighting to reduce the numerical value of boundary entries.

FGMRES with restart of 20 is used as outer accelerator - since the preconditioner changes at every step this can detract from the performance of a standard technique like CG, so a flexible method is preferred. Four unrestarted GMRES iterations on the reduced system are used per preconditioning step. In all cases the relative residual is reduced to less than 10^{-8}; with no preconditioning except diagonal scaling this takes 54 iterations to achieve for the largest matrix. It was found that the preconditioner produced the shortest execution times with incomplete factorisations containing 2.5 times more entries than the original diagonal blocks, hence this is used in all cases.

By reducing the element size in the mesh, a series of matrices is created such that consecutive matrices differ in size by a factor of two. By solving the smallest using one processor and then doubling the number of processors for every subsequent matrix, the amount of data per processor is kept approximately constant.

Procs. (P)	Entries (E)	Its. (I)	Its. on 1
1	1,246,876	3	3
2	2,516,166	3	3
4	4,759,275	3	3
8	9,867,575	4	4
16	20,509,258	5	4

Procs.	Slow	Fast (F)	F/I/(E/P)
1	2.36	1.41	3.76E-7
2	3.24	2.12	5.63E-7
4	2.98	2.09	5.86E-7
8	4.56	3.29	6.66E-7
16	6.39	4.66	7.26E-7

Table 1: Timing results for fixed problem size per processor.

Table 1 shows the timing results on a Cray T3E if the number of non-zero entries per processor (E/P) is kept to around 1,250,000. The *Its.* column indicates how many FGMRES iterations were required, while the *Slow* and *Fast* columns show the time in seconds needed to solve the system using the original and improved formulation of the approximate Schur complement preconditioner.

The most significant figures are in the last column, showing the time per iteration divided by the number of entries per processor. Ideally, this time should remain constant. For the present preconditioner some increase may be expected when moving from 1 to 2 processors, since on one processor no iterations on a reduced system are carried out. From 2 to 16 processors the time increases by around 29 percent, which is due to increased communication, as well as to a slow increase in the relative size of the reduced system.

It appears that the advantage of the fast algorithm decreases as more processors are used: from a difference of 34% for two processors to only around 27% for sixteen. This is due to a relative increase in communication time over calculation time. For low numbers of processors, the average number of adjacent processors will increase sharply as more processors are used, though this will become much less significant for very large numbers of processors. In addition, it may be noted that for any configuration where communication time dominates, either due to slow inter-processor links or a small number of entries per processor, this preconditioner will tend to be outperformed by simpler local methods like block ILUT. The reverse is true if there is a relative increase in calculation time - either through faster inter-processor links, or more entries per processor, when the reduction in iteration count offered by the approximate Schur complement preconditioner leads to significantly shorter execution times.

The *Its. on 1* column indicates the number of iterations required on one processor. It can be seen that in terms of iterations the preconditioner scales very well, with an additional iteration required only in the sixteen processor case.

5 Conclusions

For matrices from electromagnetic FETD analysis, the approximate Schur complement preconditioner gives a reduction in iteration count similar to ILU on one processor. It is simple to implement in parallel, requiring only routines for a local incomplete factorisation, the exchange of boundary variables and an accelerator. A small modification to the algorithm allows a significantly faster implementation. For constant processor load on two or more processors, execution times increase slowly as more processors are used.

A Appendix

For a matrix partitioned as in equation 5, a useful insight [5] is that an inverse of the Schur complement matrix is present inside the inverse of the complete matrix:

$$\begin{bmatrix} A_{\alpha\alpha} & A_{\alpha\beta} \\ A_{\beta\alpha} & A_{\beta\beta} \end{bmatrix}^{-1} = \begin{bmatrix} A_{\alpha\alpha}^{-1} & -A_{\alpha\alpha}^{-1}A_{\alpha\beta}A_{SC}^{-1} \\ 0 & A_{SC}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -A_{\beta\alpha}A_{\alpha\alpha}^{-1} & I \end{bmatrix} = \begin{bmatrix} * & * \\ -A_{SC}^{-1}A_{\beta\alpha}A_{\alpha\alpha}^{-1} & A_{SC}^{-1} \end{bmatrix} \tag{10}$$

An asterisk indicates a term of no interest. This identity allows the incomplete factorisation of the Schur complement to be found from the incomplete factorisation of the diagonal block on each domain.

To see how the right hand side of equation 6 is obtained, note that the right hand side of the reduced system, found by eliminating the local variables on a domain and then multiplying by the inverse of the Schur complement matrix, is equal to:

$$A_{SC_i}^{-1}(b_{\beta_i} - A_{\beta\alpha_i}A_{\alpha\alpha_i}^{-1}b_{\alpha_i}) \tag{11}$$

From equation 10 it can be seen that this is the same as:

$$R_{SC_i}A_i^{-1}b_i \tag{12}$$

References

[1] D.C. Dibben and A.C. Metaxas. Finite element time domain analysis of multimode applicators using edge elements. *Journal of Microwave Power and Electromagnetic Energy*, 29(4):242–251, 1994.

[2] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Scientific Computing*, 20(1):359–392, 1998.

[3] D.H. Malan. *Parallel Finite Element Analysis for Microwave Heating Systems.* PhD thesis, University of Cambridge, 2000.

[4] D.H. Malan and A.C. Metaxas. Implementing a finite-element time-domain program in parallel. *IEEE Antennas and Propagation Magazine*, 42(1):105–109, 2000.

[5] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 20 Park Plaza, Boston MA, USA, 1996.

[6] Y. Saad and A. Malevsky. P-sparslib: A portable library of distributed memory sparse iterative solvers. Technical Report UMSI-95-180, University of Minnesota, Minneapolis, MN, 1995.

[7] Y. Saad and M. Sosonkina. Distributed Schur complement techniques for general sparse linear system. *SIAM Journal on Scientific Computing.* To appear.