



Preconditioning finite element subsurface flow solutions on distributed memory parallel computers

Andrew M. Binley

Centre for Research on Environmental Systems, Lancaster University, Lancaster, UK, LA1 4YQ

&

Malcolm F. Murphy

Department of Mathematics, Lancaster University, Lancaster, UK, LA1 4YQ

Due to their low cost and high expansion capabilities, distributed memory parallel computers, consisting of relatively large arrays of processor elements with local memory, offer great potential for solving large scale three-dimensional subsurface flow problems. We consider here two possible solution methods for such detailed modelling using the finite element method. One is based on the commonly used diagonally scaled conjugate gradient method. The other uses an element-by-element preconditioner in the conjugate gradient method which exploits the natural parallelism in the finite element method. Results using both methods are documented for saturated and variably saturated flow simulations carried out on a single Intel i860 processor and an array of Inmos transputer/Intel i860 processors at Lancaster.

Keywords: parallel computing, groundwater flow, finite element method, conjugate gradient method, element-by-element preconditioning.

INTRODUCTION

Distributed memory parallel computers of Multiple Instruction Multiple Data (MIMD) type, consisting of relatively large arrays of processor elements with local memory, offer great potential for solving large scale three-dimensional subsurface flow problems due to their low cost and high expansion capabilities. Such computers may be available as local workstations, introducing the potential for dedicated systems for practical predictive purposes. Processor arrays of this type are inexpensive in comparison with their shared memory counterparts due to the relatively low level of sophistication needed for the system software. Unfortunately, many numerical algorithms originally written for computers of a serial nature will gain little benefit from parallel computers, however, the literature on parallel solution methodologies is increasing and providing many algorithms which will make detailed three-dimensional flow and transport modelling feasible on low cost parallel machines. Grid based methods are ideally suited to such computers. We are concerned here

with the application of the finite element method, which can be demonstrated to possess natural parallel properties.

Traditionally, finite element algorithms consist of two main stages. First, the formation of local elemental equations and accumulation of the global system of equations. Second, the solution of the global series of linear equations $Ax = b$, where x is often the head at a particular time instant. For unsaturated or variably saturated problems these linear equations may be the result of some linearisation strategy, for example Picard iteration and in which case the system of equations must be solved at several iteration levels per time step as A and b are functions of the pressure head.

In the finite element method the process of forming the local element equations and accumulating the global system is essentially parallel in nature. Implementation of this stage on a distributed parallel computer is trivial due to the independent nature of the calculations. All that is required is a *farming* process which distributes element data to *slave* processors in a network, each forming individual element matrices and returning the local equations to some controlling *master* processor, although the resultant matrix equation may be too large for three-dimensional problems to be stored explicitly.

Unfortunately, the parallel solution of the resulting system of global equations is not as straightforward, however, a number of suitable schemes, both direct and iterative, have been suggested in the literature.

Methods for the direct solution of the system $Ax = b$ offer several advantages over iterative schemes, their robustness makes them particularly attractive although their greater storage requirements has led to a deterioration in popularity for large scale multi-dimensional problems. Parallel direct solution schemes, however, are available, providing a more practical approach for low memory distributed parallel processors. One such method is the multifrontal method⁷ based on the idea of eliminating equations from the global system as soon as they are fully formed. This formation and elimination procedure can be initiated from several sources of the domain and proceed in a concurrent manner, although some careful planning of nodal sequences may be needed to ensure even balancing of work amongst a processor array. The method has been successfully implemented on shared memory⁸ and distributed memory¹⁸ computers.

In comparison to direct methods, iterative solutions often require much less storage but may require more processing time for matrices with non-ideal properties, however, because an initial guess of the solution vector is often possible the number of iterations required may not be too large. Schmid & Braess²⁰ discuss the merits of direct and iterative solvers with particular emphasis on solving groundwater problems.

Parallel computation is much easier to achieve with iterative methods in comparison with the direct approach. A number of methods have been proposed in the literature (see for example Carey & Jiang³) which utilise the independent nature of the accumulation of local finite element matrices in a parallel procedure for the iterative solution of the $Ax = b$ problem. These are referred to generally as element-by-element (EBE) methods and involve distributed calculation of steps in a normal iterative solution strategy by accumulating the contributions to the steps on an element by element basis, thus eliminating the need for storage of a global system. The local matrices are therefore only formed when needed in the solution stage. In order to reduce storage requirements in an iterative solution, the local element equations would need formation at each iteration level, however, for unsaturated elements this may be a computationally expensive option. Carey *et al.*⁴ discuss methods by which EBE schemes may also be vectorised.

The most popular iterative solution methods for groundwater problems are multigrid and conjugate gradient based algorithms. Multigrid methods, in which the problem is solved at different levels of domain discretisation, appear well suited for parallel computation, Chan & Tuminaro⁵, amongst others, discuss some implementation strategies. The conjugate gradient (CG)

method (see Hestenes & Stiefel⁹) is essentially an optimisation method which treats the solution of n unknowns in a linear system of equations as the problem of finding the minimal residual vector in n dimensional space. The CG method and variants, which at present appear to be the most widely implemented subsurface flow solution algorithms, can also be easily decomposed for parallel computation.

PARALLEL IMPLEMENTATION OF THE CONJUGATE GRADIENT METHOD

The CG algorithm is simple in nature requiring only one matrix-vector product and a number of vector products per iteration. The method has the great advantage, over most iterative schemes, that iterative acceleration parameters are not required as the method is 'self accelerating'. Parallelisation of the method is relatively trivial, as demonstrated by Lyzenga *et al.*¹⁵ by decomposing the domain into a series of subregions, or subdomains. By assigning a single processor to a subdomain, the subdomain's contribution to the global system of equations may be formed in the normal manner. The vector products may be performed on each processor and transmitted to a controlling manager processor which accumulates the subdomain contributions in order to evaluate the global vector products. All that is then required is a synchronised exchange of boundary information from each subdomain to neighbouring subdomains so that the matrix vector product may be accumulated. The manager process may then evaluate global parameters and broadcast these to the elements of the processor array. Domain decomposition is completely flexible as each element of the mesh may be regarded as a subdomain, given a large processor array.

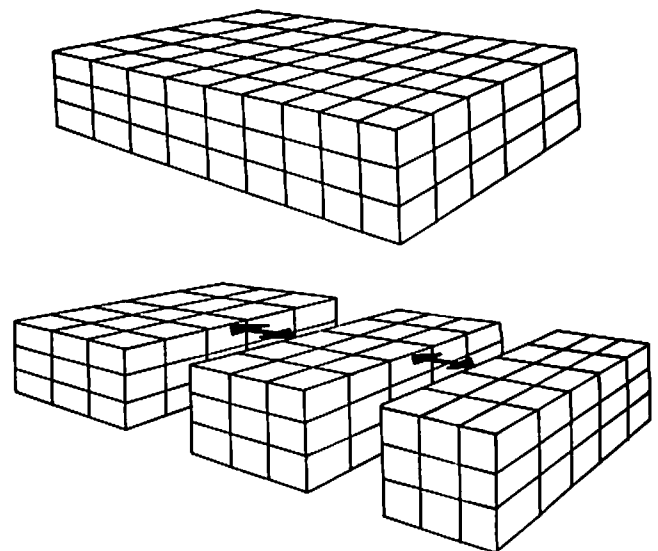


Fig. 1. Domain decomposition of a three-dimensional finite element mesh.

For the general case, communication between subdomains for the determination of the matrix vector product must be carried out as a series of communication steps as a number of subdomains may contribute to a single row of the matrix. If the domain is decomposed so that each node point is common to no more than two subdomains, communication overheads are reduced to a minimum and if each subdomain consists of a similar number of nodes load balancing is easy to achieve. Domain decomposition of a three-dimensional, not necessarily regular, mesh can then easily be achieved by separating the mesh into a series of vertical or horizontal slices, as shown in Fig. 1. Clearly, decomposition of more complex meshes, such as those based on tetrahedral elements, would not be as trivial.

The basic CG method is often considered inefficient due to the large number of iterations required for convergence, however, this can be significantly reduced by diagonally scaling the system of equations so that the $\mathbf{Ax} = \mathbf{b}$ problem is written as:

$$(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{D}^{1/2}\mathbf{x} = \mathbf{D}^{-1/2}\mathbf{b}$$

where \mathbf{D} is the diagonal of \mathbf{A} .

As with the basic CG method, the diagonally scaled version is easy to implement in a distributed parallel environment, as for example detailed in Binley². Additional communication between processors is needed at the start of each CG method solution, in comparison with the basic CG method, however, the effect is minimal provided the number of iterations needed to solve the equations is not low.

PARALLEL IMPLEMENTATION OF THE PRECONDITIONED CONJUGATE GRADIENT METHOD

The major drawback with the CG method, even with diagonal scaling, is the large number of iterations often needed to solve large problems, in particular those dealing with highly non-uniform material characteristics. This behaviour is even more significant when the solution is performed on a distributed memory parallel computer as the inter-processor communication, required for each iteration, may result in substantial overheads, especially if messages have to be routed through long paths within a processor array. A more popular form of the CG method uses a preconditioner matrix to accelerate the convergence rate. With a suitable preconditioner we may then expect the solution to become less 'communication bound' even at the expense of greater computational effort within each iteration.

The preconditioned conjugate gradient (PCG) method can be expressed as:

Let \mathbf{x}_0 be the initial solution estimate to $\mathbf{Ax} = \mathbf{b}$, then

the initial residual vector is defined by $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$, then define $\mathbf{Bg}_0 = \mathbf{r}_0$ and $\mathbf{d}_0 = \mathbf{g}_0$, then set $k = 0$.

Then do until some stopping criterion is satisfied,

$$\alpha_k = \left(\frac{\mathbf{r}_k^T \mathbf{g}_k}{\mathbf{d}_k^T \mathbf{A} \mathbf{d}_k} \right)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{d}_k$$

$$\text{solve } \mathbf{B} \mathbf{g}_{k+1} = \mathbf{r}_{k+1}$$

$$\beta_k = \left(\frac{\mathbf{r}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{r}_k^T \mathbf{g}_k} \right)$$

$$\mathbf{d}_{k+1} = \mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$$

$$k = k + 1$$

The literature reveals various recommendations for the form of the preconditioner matrix \mathbf{B} for serial calculation (see for example, Axelsson¹; Meyer *et al.*¹⁷; Hill¹⁰). Preconditioning may be efficient for certain classes of problems, for example those of a steady state nature, where convergence properties of the overall system of equations is poor, although in many cases the extra effort needed to precondition may not prove to be efficient. Pini & Gambolati¹⁹, based on a number of trial problems, suggest that the diagonal of \mathbf{A} may be the most efficient preconditioner for many problems. In a number of cases, such as those dealing with complex boundary conditions, for example third type, convergence of a diagonally scaled solution may not arise (Dougherty, personal communication, 1992). In such instances preconditioning is clearly essential.

Element-by-element (EBE) preconditioning

Winget & Hughes²¹ discuss a preconditioner which has natural parallel properties and show for their heat conduction problems significant speed up on a vector computer making the solution more efficient than a diagonally preconditioned CG method solution.

Consider the accumulation of finite element matrices \mathbf{A}_e over n_e elements to form the global stiffness matrix \mathbf{A} . The Winget & Hughes preconditioner is based on element-by-element factorisation of these local matrices and uses the following decomposition for Crout factorisation of a symmetric system of equations:

$$\mathbf{B} = \mathbf{W}^{1/2} \times \prod_{e=1}^{e=n_e} L(\tilde{\mathbf{A}}_e) \times \prod_{e=1}^{e=n_e} D(\tilde{\mathbf{A}}_e) \\ \times \prod_{e=n_e}^{e=1} L^T(\tilde{\mathbf{A}}_e) \times \mathbf{W}^{1/2}$$

where: $\mathbf{W} = \text{diag}(\mathbf{A})$, $L(\tilde{\mathbf{A}}_e)$ and $D(\tilde{\mathbf{A}}_e)$ are the lower and diagonal factors, respectively, of $\tilde{\mathbf{A}}_e$, such that $L(\tilde{\mathbf{A}}_e)D(\tilde{\mathbf{A}}_e)L^T(\tilde{\mathbf{A}}_e) = \tilde{\mathbf{A}}_e$, the *Winget regularised element*

matrix which is given by:

$$\tilde{A}_e = I + W^{-1/2}(A_e - \text{diag}(A_e))W^{-1/2}$$

which is carried out to ensure that \tilde{A}_e is positive-definite.

Note that A_e and \tilde{A}_e are of a size $n_p \times n_p$, where n_p is the total number of node points, however, since the operations in the above only act on the terms of each finite element equation, the global matrix A , does not need to be stored, although the global diagonal of $A(=W)$ is needed for the diagonal scaling above.

As an example to illustrate the use of the procedure, consider a finite element mesh consisting of three elements. The individual element matrices A_1 , A_2 and A_3 are formed in the normal way but stored in an element-by-element manner rather than forming a global matrix. During this operation the global diagonal vector W and global right-hand side vector b are formed. Once the element integrations are complete we may then modify the element matrices using the *Winget regularisation* procedure above. The lower and diagonal factors of all elements (L_1, L_2, L_3 and D_1, D_2, D_3 , respectively) may then be calculated, again on an element-by-element basis.

Then, during the PCG solution of the system of linear equations we require the solution of $Bg_k = r_k$, which because of our selected form for B we may express as:

$$Bg_k = W^{1/2}L_1L_2L_3D_1D_2D_3L_3^TL_2^TL_1^TW^{1/2}g_k = r_k$$

We may then obtain the solution g_k , with the following steps:

global diagonal scaling:

$$y_0 = W^{-1/2}r_k$$

forward reduction:

$$y_1 = L_1^{-1}y_0$$

$$y_2 = L_2^{-1}y_1$$

$$y_3 = L_3^{-1}y_2$$

elemental diagonal scaling:

$$\hat{y}_3 = D^{-1}y_3$$

back substitution:

$$\bar{y}_3 = L_3^{-T}\hat{y}_3$$

$$\bar{y}_2 = L_2^{-T}\bar{y}_3$$

$$\bar{y}_1 = L_1^{-T}\bar{y}_2$$

global diagonal scaling:

$$g_k = W^{-1/2}\bar{y}_1$$

Note that in the above the diagonal factorisation is done in one stage using $D = D_1D_2D_3$. Also note that in the above $L^{-T} = (L^T)^{-1}$.

During the PCG solution we also require the

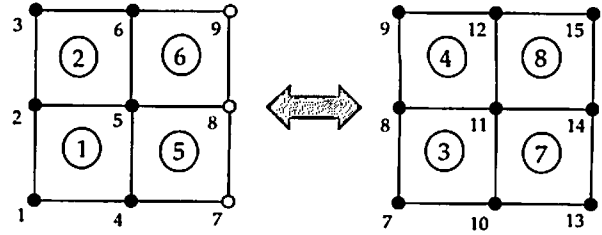


Fig. 2. Parallel EBE preconditioning: eight element mesh decomposition.

matrix-vector product Ad_k , again this may be done on an element-by-element basis.

The EBE preconditioner of Winget & Hughes has two important properties. First, for computations where storage is limited the element factors may be formed only when needed. Although this is likely to increase execution time significantly it would make large three-dimensional solutions possibly on small workstations in-core. The second property, which is of interest here, is that the forward and backward substitution stages may be executed on blocks of elements concurrently provided all the elements within a block are not connected. Hughes *et al.*¹¹ applied such a procedure on a Cray X-MP/48 using block sizes of 64 to exploit the vector hardware of the Cray computer. Based on the timings of various structural analysis problems their solution appears to vectorise well.

The EBE preconditioner also appears ideally suited to distributed array processing. For example, consider the finite element mesh containing eight elements decomposed into two subdomains and numbered as shown in Fig. 2. Each subdomain is allocated to a separate processor in the array and forms its element matrices together with its lower and diagonal factors.

Consider then the first stage of the solution of $Bg_k = r_k$ which involves global diagonal scaling and forward reduction. The vector r_k and W is known for all nodes within each subdomain. We may then proceed with the following operations concurrently:

On processor 1: $y_0 = W^{-1/2}r_k, y_1 = L_1^{-1}y_0, y_2 = L_2^{-1}y_1$
 On processor 2: $y_0 = W^{-1/2}r_k, y_3 = L_3^{-1}y_0, y_4 = L_4^{-1}y_3$

A transmission of terms common to the boundary of the two subdomains (that is, for nodes 7, 8, and 9 in Fig. 2) is then needed. This involves sending the relevant part of y_4 from processor 2 to processor 1 and updating the subdomain boundary terms in y_2 .

We then continue with the following steps executed concurrently:

On processor 1: $y_5 = L_5^{-1}y_4, y_6 = L_6^{-1}y_5$
 On processor 2: $y_7 = L_7^{-1}y_4, y_8 = L_8^{-1}y_7$

When this sequence is complete processor 1 must transmit the subdomain boundary terms in y_6 to processor 2 to update the terms common to the both subdomains thus modifying y_8 on processor 2.

At this stage elemental diagonal scaling is carried out:

$$\text{On processor 1: } \hat{y}_6 = \mathbf{D}_1^{-1} \mathbf{D}_2^{-1} \mathbf{D}_5^{-1} \mathbf{D}_6^{-1} y_6$$

$$\text{On processor 2: } \hat{y}_8 = \mathbf{D}_3^{-1} \mathbf{D}_4^{-1} \mathbf{D}_7^{-1} \mathbf{D}_8^{-1} y_8$$

Back substitution is then performed in two stages, as in forward reduction. First:

$$\text{On processor 1: } \bar{y}_6 = \mathbf{L}_6^{-T} \hat{y}_6, \bar{y}_5 = \mathbf{L}_6^{-T} \bar{y}_6$$

$$\text{On processor 2: } \bar{y}_8 = \mathbf{L}_8^{-T} \hat{y}_8, \bar{y}_7 = \mathbf{L}_7^{-T} \bar{y}_8$$

Which is followed by a transmission of subdomain boundary terms in \bar{y}_5 on processor 1 to processor 2 to update \bar{y}_7 . The second stage of back substitution is then performed by:

$$\text{On processor 1: } \bar{y}_2 = \mathbf{L}_2^{-T} \bar{y}_5, \bar{y}_1 = \mathbf{L}_1^{-T} \bar{y}_2$$

$$\text{On processor 2: } \bar{y}_4 = \mathbf{L}_4^{-T} \bar{y}_1, \bar{y}_3 = \mathbf{L}_3^{-T} \bar{y}_4$$

Subdomain boundary terms in \bar{y}_3 on processor 2 must then be sent to processor 1 to update \bar{y}_1 .

Finally global scaling takes place within the two subdomains:

$$\text{On processor 1: } \mathbf{g}_k = \mathbf{W}^{-1/2} \bar{y}_5$$

$$\text{On processor 2: } \mathbf{g}_k = \mathbf{W}^{-1/2} \bar{y}_3$$

A procedure for implementation of the above on a distributed memory parallel computer for a finite element mesh decomposed into any number of slices is outlined in the appendix. This procedure could also be generalised to deal with mesh decomposition into domains of arbitrary size and shape (such as meshes containing various element shape types) although the additional communication overheads would probably be too restrictive.

EXAMPLES USING SERIAL COMPUTATION

Before examining the usefulness of the EPE-PCG for parallel computation, we consider here the application of the method on a serial computer to two subsurface flow problems and use the diagonally scaled CG method (hereafter referred to as DCG) for comparison. We wish to explore the sensitivity of preconditioning to media heterogeneity and finite element shape.

Example 1: Steady state saturated flow in heterogeneous media

For this example we determine the solution of the problem:

$$\nabla \cdot (K_s \nabla H) = 0$$

in a unit cube. Where H are the hydraulic heads and K_s is the saturated hydraulic conductivity.

This problem has been used as a benchmark finite difference problem on global memory parallel computers by Meyer *et al.*⁷ and more recently by Dougherty⁶.

We adopt here a similar problem specification, namely, Dirichlet boundary conditions subject to $H = 0$ on one surface of the cube, $H = 1$ on the opposite face; initial solution vector $H = 0.9$ for all internal nodes. In our test K_s derived from a spatially uncorrelated field, log transformed and given by:

$$K_s = 0.1e^{\sigma Z},$$

where Z is taken from a $N[0,1]$ distribution and σ is the standard deviation of the log hydraulic conductivities.

For both DCG and EBE-PCG we use the same criteria for convergence of the resulting $\mathbf{Ax} = \mathbf{b}$ problem, namely:

$$\frac{\|\mathbf{r}_k\|_2}{\|\mathbf{b}\|_2} \leq \epsilon$$

where ϵ is the acceptable tolerance.

For these experiments execution was timed on a single Intel i860 processor for the solution stage of the code (formation of finite element equations being omitted from the timing stage). For all experiments $\epsilon = 10^{-6}$. Note that the results are taken from single realisations of variability. Over the range of heterogeneity considered we observed little difference in results between realisations. Figure 3(a) shows the effect of heterogeneity on convergence of the two methods for a $31 \times 31 \times 31$ grid. As σ increases the EBE preconditioning becomes more attractive, as indicated by the reduced execution time shown in Fig. 3(b), although for reasonably natural levels of variability in hydraulic conductivity there appears little advantage in preconditioning with the EBE method. Figure 4 shows the near log-linear convergence rate of both methods for the case where $\sigma = 2$.

Example 2: Flow in a variably saturated porous medium

Our second example is based on the solution of a steady state variably saturated flow problem. For such problems heterogeneity caused by the nonlinear relationship between hydraulic conductivity and pressure head may demonstrate effectiveness of a preconditioner. Also of interest for realistic applications is the effect of element shape on solution performance. The mesh of cubic elements in the previous example is clearly not typical for real applications, often for subsurface problems the horizontal dimensions of the region will be much larger than the vertical dimension. For many problems dealing with unsaturated flow discretisation in the vertical may be several orders less than in the horizontal in order accurately to describe the large vertical head gradients. Recently, Lee & Wathen¹³ compared the EBE-PCG and DCG methods for two-dimensional linear elliptic problems. They were able to show that the ratio of condition numbers for the two methods varies with grid shape in such a way that an

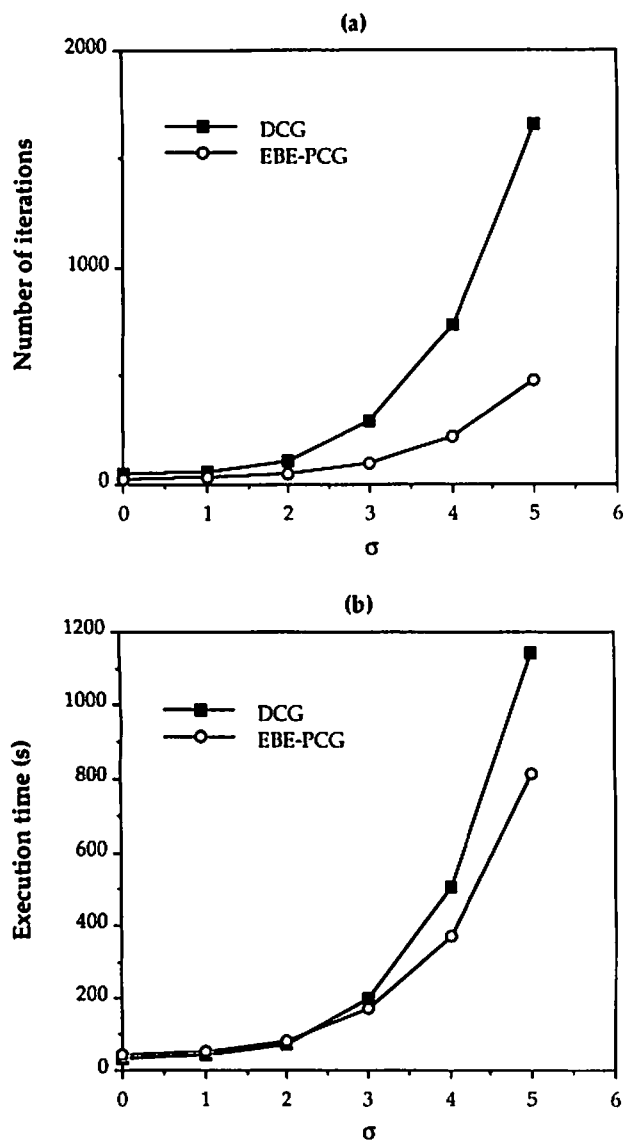


Fig. 3. Comparison of DCG and EBE-PCG for example 1, $31 \times 31 \times 31$ grid. (a) Effect of heterogeneity on number of iterations to converge. (b) Effect of heterogeneity on total execution time.

optimum region for EBE preconditioning can be seen. Because of the generality in their analysis, Lee & Wathen were unable to reveal performance comparisons in terms of computational effort or even the total number of iterations to converge. We use this example here then to extend Lee & Wathen's work by investigating the effect of changing finite element shape on the convergence properties of a particular solution for variably saturated flow, with the aim of assessing the practical implications of the variation in condition number ratio between the two methods.

The governing equation of flow is:

$$\nabla \cdot (K_s K_r(\psi) \nabla H) = 0$$

where: K_s is the saturated hydraulic conductivity, K_r is the relative hydraulic conductivity ($0 < K_r \leq 1$), ψ is

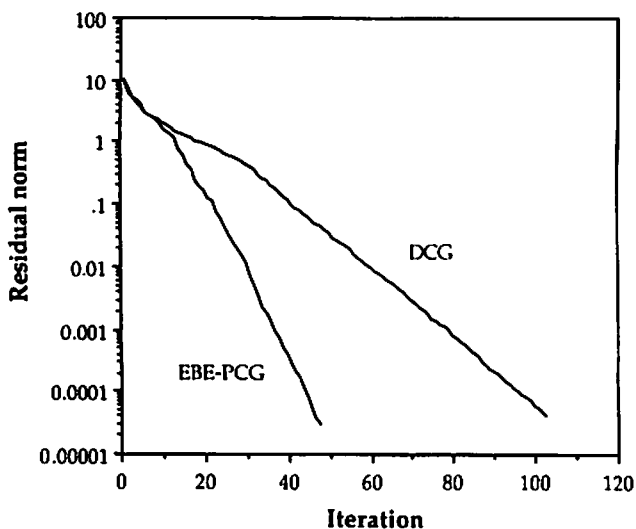


Fig. 4. Example decay in error norm with iterations for DCG and EBE-PCG methods ($31 \times 31 \times 31$ grid, $\sigma = 2$).

pressure head, H is hydraulic head ($= \psi + z$), z is elevation.

For this example the flow domain is a cuboid of size $10\text{m} \times w\text{m} \times w\text{m}$ with boundary conditions shown in Fig 5. The domain is discretised into a grid of $11 \times 11 \times 11$ node points (with uniform element size) and we will examine here the effect of changing the length w on the convergence properties of the DCG and EBE-PCG methods. For the purpose of this comparison we define the element aspect ratio as an element's longest dimension divided by its shortest. The initial solution guess is set to $H = 10\text{m}$ for all internal nodes within the domain.

The unsaturated hydraulic conductivity is specified as:

$$K_r(\psi) = \frac{a}{a + |\psi|^b}$$

where: $a = 10\text{m}^4$, $b = 4.0$.

We use a Galerkin finite element scheme based on eight node linear brick elements. Eight point Gauss quadrature is used to evaluate the resulting element integrals, the unsaturated hydraulic conductivity $K_r(\psi)$ being evaluated at each gauss point within the element. The scheme used is suitable for general isoparametric elements as the Jacobian of the element local coordinate system is also evaluated at each Gauss point. Consequently, a significant amount of computation is required in evaluating the element equations unlike the previous example. We have not refined the algorithm to work just for this simple problem so that an estimate of the computational efficiency of the DCG and EBE-PCG methods for realistic three-dimensional problems can be made.

A Picard iteration scheme is used to linearise the governing equations and we adopt a tolerance of 0.001 m here for this scheme. Table 1 lists the total number of iterations required for a series of Picard

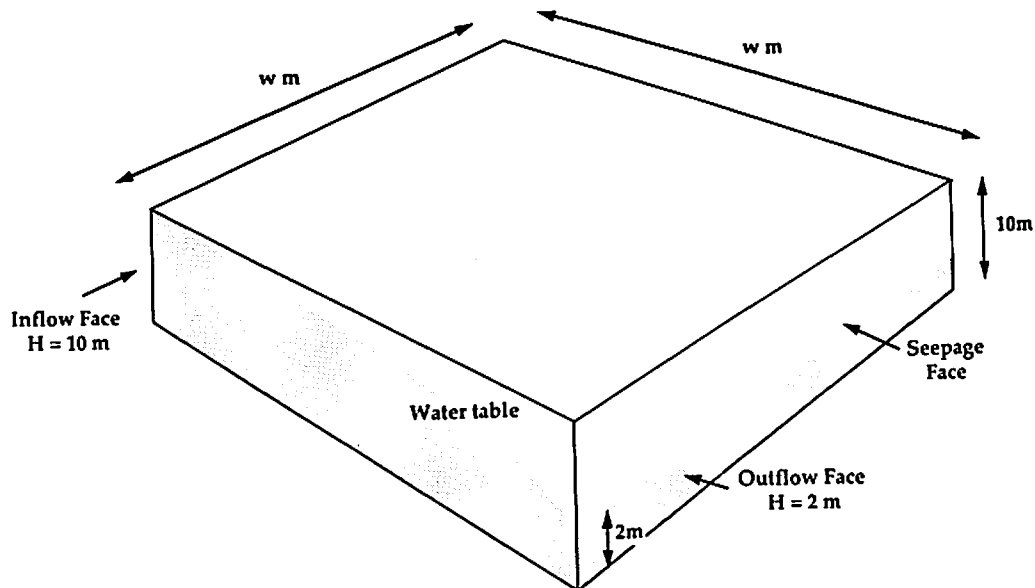


Fig. 5. Definition of variably saturated problem example 2.

iterations for this problem, for a range of aspect ratios, using the DCG and EBE-PCG methods. Also listed are the execution times on a single Intel i860 processor for the entire Picard solution stage of the code. The stage includes formation of local elemental equations for each Picard iteration and solution of resulting linear equations. We use the same criteria for convergence of the conjugate gradient method as the previous example, with $\epsilon = 10^{-6}$ for all experiments.

Figure 6 shows the relative performance (in terms of execution time) of the EBE method as a preconditioner for this example compared with the DCG method. A region in which the EBE-PCG method shows optimal performance can be seen clearly for small (less than 10) aspect ratios. Within this region the EBE preconditioner only shows a maximum improvement of some 10% over the diagonally scaled method. Furthermore, the relative performance of the EBE-PCG method deteriorates significantly for larger, more practical, aspect ratios.

Table 1. Performance of DCG and EBE-PCG methods for Example 2

Aspect ratio	DCG		EBE-PCG	
	Iterations	Time(s)	Iterations	Time(s)
1.0	95	8.09	42	7.95
1.5	131	9.52	47	8.41
2.0	161	10.75	58	9.47
2.5	196	11.94	68	10.39
3.0	248	15.08	93	13.54
4.0	312	17.67	108	16.83
5.0	389	21.60	165	21.09
10.0	730	35.18	382	41.30
20.0	1092	49.58	758	76.30
30.0	1397	61.71	1022	100.87
40.0	1676	72.81	1121	110.10
100.0	2118	89.48	1299	126.66

PARALLEL COMPUTATION

The previous two examples have shown some difficulties in selecting suitable preconditioning. The choice of method will often be problem dependent and we have shown two sensitive factors influencing the performance of the EBE method when compared to diagonal scaling. We have not observed over a number of trial problems any failure of the EBE-PCG method to converge and always see some improvement in convergence rate (measured in terms of the number of iterations to converge) when compared to the DCG method. Clearly, such a measure of improvement should be viewed with caution as the EBE preconditioner requires significantly

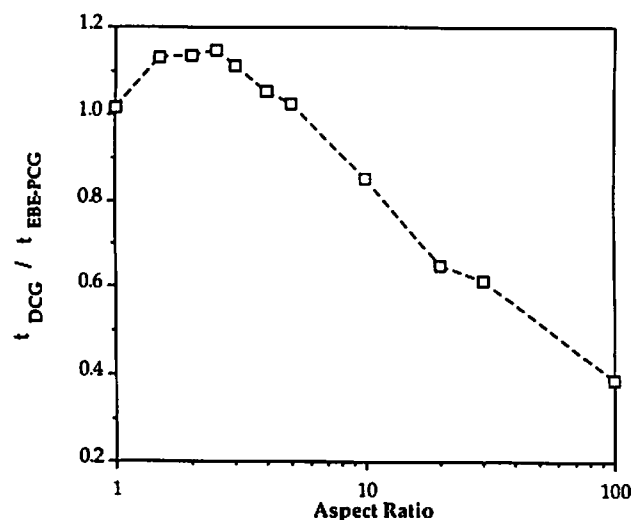


Fig. 6. Effect of element aspect ratio on execution times of DCG and EBE-PCG methods. $t_{DCG}/t_{EBE-PCG}$ is ratio of DCG execution time to that of EBE-PCG.

more work per iteration. On a distributed memory parallel computer, message passing will be invoked at each iteration and so one should attempt to minimise communication at the expense of more computational effort on each processing element. As stated earlier, for large processor arrays, sensitive to costly message routing throughout the array, we may see some improvement in the EBE method, compared to the DCG method.

Parallel computer hardware

Our investigation uses a distributed memory parallel computer based on the Inmos transputer and the Intel i860. The computer is essentially a series of 11 16 Mb Intel i860 modules, supplied by Transtech Devices, contained in a Sun workstation. Each i860 is attached to a 4 Mb T805 transputer which acts as a communication processor for that i860. This version of the transputer contains four inbuilt high speed communication links permitting a wide variety of processor configurations. This configuration was purchased as a departmental facility and can be easily expanded with any number of additional processors. This expansion facility is one of the key advantages of such computers as relatively low cost start-up systems may be purchased and then expanded at a later date as projects and funds permit, unlike other fixed configuration distributed computers.

The processors were networked using a ring topology. Other arrangements, for example tree, hypercube, etc., may be more efficient for particular numbers of processors but for a general configuration this was considered to be optimal and least biased in benchmarking processor array performance.

Software environment

Software is provided at two levels. At the system level, a process manager is executed on each processor in the network, with the task of routing messages throughout the array and providing a link to the host computer file system. At the user level, software is available in the form of libraries. Subroutines within the libraries allow a number of tasks to be performed, the most common of which are the sending and receiving of messages to and from another process in the network. For the parallel EBE-PCG procedure documented in the appendix two Fortran programs were written, one to execute as a *master* process, the other to be executed on each (*slave*) processor in a selected array of processors. For the DCG solution we use an equivalent pair of codes, as reported by Binley². Note that in both cases the *master* process does very little work and thus may be executed on the processor array host computer. In our examples we use one of the i860 modules for this task.

Comparison of methods

We use the saturated flow in heterogenous media example, described above, to report on performance trials using parallel DCG and EBE-PCG codes. Note that the procedure for parallel solution is identical for variably saturated and saturated flow. For various processor array sizes we ran a number of problems of different size and degrees of heterogeneity. As the EBE method of preconditioning is based on a product of element factors, the solution will vary to some degree with the order of operation. This order for serial computation will differ from parallel solution, however, we did not observe any great sensitivity to this ordering during our trials.

Figure 7 shows the speed-up of the parallel solution for both methods on a number of processor array sizes for a $31 \times 31 \times 31$ problem with $\sigma = 2$. We used this level of heterogeneity for comparison of methods as serial execution times were similar for both solution methods. The effect of the reduced number of iterations, hence communication overheads, in the EBE method is clearly shown by a high parallel efficiency, in comparison to the diagonally scaled solution.

These results suggest that for larger problems the EBE-PCG method would be far superior to the DCG method within a parallel framework simply because the size of the messages would increase and hence communication overheads would be more significant. In fact, as the mesh size increases we actually see very little improvement as the preconditioning using element factors becomes slightly less effective as the mesh size increases. This is shown in Fig. 8 in terms of the ratio of iterations to solve the two methods varying with mesh size, again with $\sigma = 2$.

In Fig. 9 the variation in execution times of the DCG parallel algorithm for various problem sizes (with $\sigma = 2$)

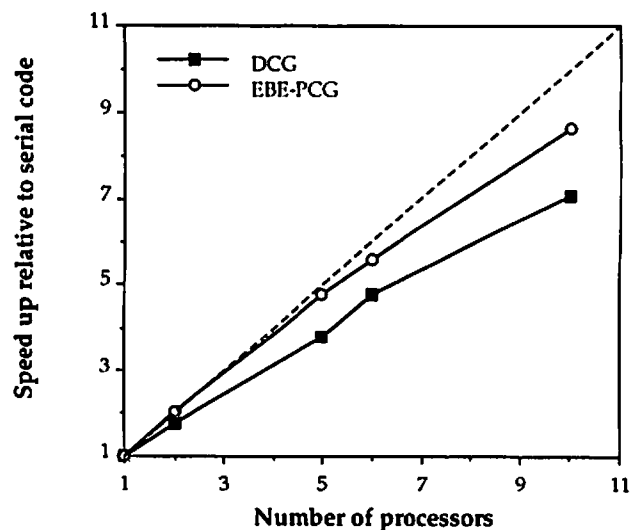


Fig. 7. Parallel speed-up of DCG and EBE-PCG methods, for $31 \times 31 \times 31$ grid, $\sigma = 2$.

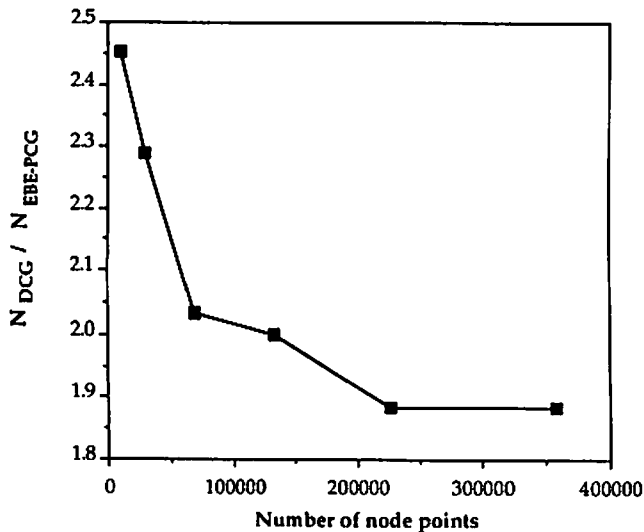


Fig. 8. Effect of grid size on number of iterations to converge. $N_{DCG}/N_{EBE-PCG}$ is ratio of DCG iterations to that of EBE-PCG.

on an array of 10 processors is shown (EBE-PCG times are similar due to the effects noted above). Since we have constrained our domain decomposition to slicing in one plane then we would not expect times in between the markers in Fig. 9 to follow a linear (on log scales) relationship, as indicated by the dashed lines. To show this effect the execution time for a $35 \times 35 \times 35$ grid (42875 node points) has been included in Fig. 9. Even though load balancing is not optimum for this grid the effect is not great.

Treating the relationship in Fig. 9 as a simple power law expression, that is, execution time is proportional to N^a we find, for our example, $a = 1.18$, the deviation from the theoretical value of $4/3$ being due to communication overheads. Note that a grid consisting

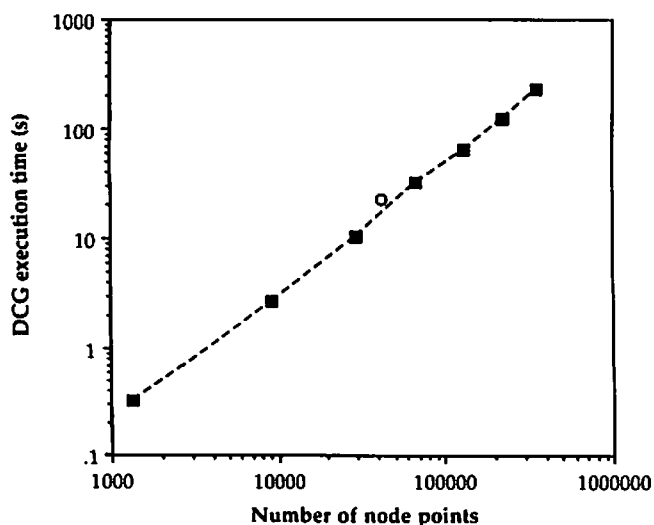


Fig. 9. Execution times for parallel DCG solution on 10 processors for various grid sizes (all timings for $\sigma = 2$). Solid markers refer to perfectly load balanced problem. Open circle refers to uneven load balanced problem ($35 \times 35 \times 35$ grid).

of an equal number of node points in each dimension, as in the example here, is rather artificial. For realistic applications we would 'slice' our mesh along planes normal to the grid dimension with the largest number of nodes, as in Fig. 1, thus reducing communication overheads.

CONCLUDING REMARKS

Recent advances in computer hardware and numerical algorithms now make three-dimensional modelling of flow in variably saturated environments a practical option. The relatively low cost of distributed parallel computers presents the possibility of dedicated powerful computers for predictive purposes. The finite element method is inherently parallel and easy to solve in an efficient manner on such machines using the conjugate gradient method.

Preconditioning the conjugate gradient solution to reduce the number of iterations for solution helps to avoid a 'communication bound' algorithm on distributed memory computers and thus improve parallel performance. The choice of suitable preconditioners for general flow problems, however, is not clear. Popular methods on serial and vector computers, such as Incomplete Cholesky Factorisation (see Meijerink & van der Vorst¹⁶), may not be efficient for distributed memory parallel computers.

We have examined a preconditioning method based on element-by-element factorisation of individual finite element equations, which shows natural parallel properties. Our results suggest that the EBE-PCG method shows greater parallel efficiency than the traditional diagonally scaled CG method, although the general efficiency of this preconditioner may not be suitable for general flow problems. For highly heterogeneous systems the EBE based method appears to be superior but for finite element grids containing distorted elements the EBE-PCG timings were disappointing, even though convergence rates were higher than the DCG method. Recognising the limited number of trials reported here, further experiments are recommended to evaluate fully the EBE preconditioning algorithm.

Despite these results, the basic EBE method can be improved, possibly to such an extent to make it a practical option. First, on parallel arrays consisting of processors with vector architecture, such as the Intel i860, the preconditioning stage is relatively easy to vectorise. This is achieved by decomposing each subdomain into blocks of non-adjacent elements and operating forward reduction and back substitution stages in a concurrent manner. We decided not to attempt this here, emphasising the parallel properties of the algorithms. A second improvement may be achieved by writing the preconditioner as a product of factors of 'super-elements' rather than single elements. These

super-elements would consist of a block of adjacent elements, the size of the block varying from one to the number of elements in a subdomain. As the block size increases, the convergence rate would improve at the expense of greater effort in factorising larger super-element matrices. Such a scheme would be equivalent to a direct solution at the limit of block size equal to the total number of elements in the mesh. This more general EBE method has been referred to as 'Clustered Element-By-Element' by Liou & Tezduyer¹⁴ and shows great promise for general domain decomposition methods. Excessive computational overheads due to factorisation of the super-element matrices need not arise as the factors need only be calculated once for an entire simulation. This applies also to unsaturated problems if one adopts simple element integration schemes, such as those used by Huyakorn *et al.*¹² in determining what they refer to as 'influence coefficients'.

We strongly urge continuing trials to assess the suitability of EBE methods for flow (and transport) solutions. The natural vector and parallel properties of EBE methods are obvious, the suitability for typical groundwater finite element schemes is still not clear and has yet to be assessed.

ACKNOWLEDGEMENTS

We are grateful to the Natural Environment Research Council and the University Grants Committee Earth Science Review, 1987, for recognising our needs for local parallel computing facilities. We would also like to thank the three anonymous reviewers for their constructive comments.

REFERENCES

1. Axelsson, O., A survey of preconditioned iterative methods for linear systems of algebraic equations. *B.I.T.*, **25** (1985) 166–87.
2. Binley, A.M., Exploiting parallelism in finite element models of flow in variably saturated porous media. In *Computational Methods in Water Resources IX, Vol. 1: Numerical Methods in Water Resources*, ed. Russel *et al.* Computational Mechanics Publications, 1992 pp. 687–94.
3. Carey, G.F. & Jiang, B.N., Element-by-element linear and non-linear solution schemes. *Comm. Appl. Numer. Methods*, **2** (1986) 145–53.
4. Carey, G.F., Barragy, E., McLay, R. & Sharma, M., Element-by-element vector and parallel computations. *Comm. Appl. Numer. Methods*, **4** (1988) 299–307.
5. Chan, R.F. & Tuminaro, R.S., A survey of parallel multigrid algorithms. In *Parallel Computations and Their Impact on Mechanics*, ed. A.K. Noor, A.S.M.E., 1987, pp. 155–70.
6. Dougherty, D.E., Hydrologic applications of the Connection Machine CM-2. *Water Resources Res.*, **27** (1991) 3137–47.
7. Duff, I.S. & Reid, J.K., The multifrontal solution of unsymmetric sets of linear equations. Report CSS 133, Computer Science and Systems Division, Harwell Laboratory, Oxon, 1983.
8. Duff, I.S., Gould, N.I.M., Lescrenier, M. & Reid, J.K., The multifrontal method in a parallel environment, Report CSS 211, Computer Science and Systems Division, Harwell Laboratory, Oxon, 1987.
9. Hestenes, M.E. & Stiefel, E., Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, **49** (1952) 409–36.
10. Hill, M.C., Solving groundwater flow problems by Conjugate-Gradient methods and the Strongly Implicit Procedure., *Water Resources Res.*, **26**(9) (1990) 1961–9.
11. Hughes, T.J.R., Ferencz, R.M. & Hallquist, J.O., Large-scale vectorized implicit calculations in solid mechanics on a Cray XMP/48 utilizing the EBE preconditioned conjugate gradients., *Comp. Methods Appl. Mech. Eng.*, **61** (1987) 215–48.
12. Huyakorn, P.S., Springer, E.P., Givanasen, V. & Wadsworth, T.D., A three-dimensional finite-element model for simulating water flow in variably saturated porous media. *Water Resources Res.*, **22** (1986) 1790–1808.
13. Lee, H.-C. & Wathen, A.J., On element-by-element preconditioning for general elliptic problems., *Comp. Methods Appl. Meth. Eng.*, **92** (1991) 215–29.
14. Liou, J. & Tezduyer, T.E., A clustered element-by-element method for finite element computations. In *Proceedings Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, ed. Glowinski *et al.* SIAM, 1991, pp. 140–50.
15. Lyzenga, G.A., Raefsky, A. & Hagar, B.H., Finite elements and the method of conjugate gradients on a concurrent processor. In *Proceedings 1985 ASME International Computers in Engineering*, Boston, 1985, pp. 401–5.
16. Meijerink, J.A. & van der Vorst, H.A., An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-Matrix. *Mathematics of Computation*, **31** (1977) 148–62.
17. Meyer, P.D., Valocchi, A.J., Ashby, S.F. & Saylor, P.E., A numerical investigation of the conjugate gradient method as applied to three-dimensional groundwater flow problems in randomly heterogeneous porous media. *Water Resources Res.*, **25**(6) (1989) 1440–6.
18. Miles, R.G. & Havard, S.P., Multifronts and transputer networks for solving fluid mechanical finite element systems. *Int. J. Numer. Methods Fluids*, **9** (1989) 731–40.
19. Pini, G. & Gambolati, G., Is simple diagonal scaling the best preconditioner for conjugate gradients on supercomputers? *Advances in Water Resources*, **13** (1990) 147–53.
20. Schmid, G. & Braess D., Comparison of fast equation solvers for groundwater flow problems. In *Groundwater Flow and Quality Modelling*, ed. E. Custido *et al.*, D. Reidel Publishing Co., 1988, pp 173–88.
21. Winget, J.M. & Hughes, T.J.R., Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies. *Comp. Methods Appl. Mech. Eng.*, **52** (1985) 711–815.

APPENDIX: EBE-PCG PARALLEL SOLUTION

To outline the complete procedure, consider a processor array consisting of n slave processors with connections to left and right neighbours and a master processor. A

three-dimensional mesh is decomposed into a series of subdomains so that any finite element is assigned to one subdomain but several node points will be common to two subdomains (as in Fig. 1). We then assign the number of active nodes within a subdomain equal to the total number of nodes in that subdomain minus the number of nodes common to the subdomain on the right, then, within all (except the rightmost) subdomains the number of active nodes will be less than the total number of nodes. We then follow the procedure below on our *master - slaves* network. Within each subdomain the elements are numbered from 1 to n_e , n_{eL} is the number of elements connected to the subdomain on the left and n_{eR} is the number of elements connected to the subdomain on the right.

Slave process for subdomain i ($i = 1, 2, \dots, n$ numbered left to right)

Stage 1: Formation of equations

- for each element e in subdomain form equations and store A_e
- for all nodes accumulate global diagonal W and global right hand side b
- send non-active terms of W to subdomain $i + 1$
- receive non-active terms in W from subdomain $i - 1$ and add to W
- send terms in W common to subdomain $i - 1$ to subdomain $i - 1$
- receive updated non-active terms in W from subdomain $i + 1$

Stage 2: Calculate LDL^T factors

- for each element e calculate the *Winget regularised element matrix* given by:

$$\tilde{A}_e = I + W^{-1/2}(A_e - \text{diag}(A_e))W^{-1/2}$$
- for each element e determine lower L_e and diagonal D_e factors of \tilde{A}_e
- calculate diagonal scaling vector $D = D_1 D_2 \dots D_n$
- send non-active terms of D to subdomain $i + 1$
- receive non-active terms in D from subdomain $i - 1$ and accumulate D with product
- send terms in D common to subdomain $i - 1$ to subdomain $i - 1$
- receive updated non-active terms in D from subdomain $i + 1$ (the subdomain now contains the complete D for all nodes)

Stage 3: Initialisation of PCG

- x = initial solution vector
- $k = 0$
- for all nodes $r = b - Ax$ calculated element-by-element
- send non-active terms in r to subdomain $i + 1$
- receive non-active terms in r from subdomain $i - 1$ and add to r
- send terms in r common to subdomain $i - 1$ to subdomain $i - 1$
- receive updated non-active terms in r from subdomain $i + 1$ (the subdomain now contains the complete r for all nodes)
- $b2 = b.b$ over all active nodes
- send $b2$ to *master* processor

Stage 4: Initial solution of $Bg = r$

- Global scaling:
 - For all nodes: $g = W^{-1/2}r$
- Forward reduction:
 - carry out the sequence: $g = L_1^{-1}g, g = L_2^{-1}g, \dots, g = L_{n_eL}^{-1}g$
 - send common terms in g to subdomain $i - 1$
 - receive common terms in g from subdomain $i + 1$ and update g
 - carry out the sequence: $g = L_{n_eL+1}^{-1}g, g = L_{n_eL+2}^{-1}g, \dots, g = L_{n_e}^{-1}g$
 - send common terms in g to subdomain $i + 1$
 - receive common terms in g from subdomain $i - 1$ and update g
- Diagonal scaling:
 - For all nodes: $g = D^{-1}g$
- Back substitution:
 - carry out the sequence: $g = L_{n_e}^{-T}g, g = L_{n_e-1}^{-T}g, \dots, g = L_{n_e-n_eR+1}^{-T}g$
 - send common terms in g to subdomain $i + 1$
 - receive common terms in g from subdomain $i - 1$ and update g
 - carry out the sequence: $g = L_{n_e-n_eR}^{-T}g, g = L_{n_e-n_eR-1}^{-T}g, \dots, g = L_1^{-T}g$
 - send common terms in g to subdomain $i - 1$
 - receive common terms in g from subdomain $i + 1$ and update g
- Global scaling:
 - For all nodes: $g = W^{-1/2}g$
- For all nodes : $d = g$

- $rg = \mathbf{r} \cdot \mathbf{g}$ for all active nodes
- send rg to *master* processor

Stage 5: Start of iteration cycle with matrix vector product

- $\mathbf{Ad} = \mathbf{Ad}$ for all nodes
- send non-active terms in \mathbf{Ad} to subdomain $i+1$
- receive non-active terms in \mathbf{Ad} from subdomain $i-1$ and add \mathbf{Ad}
- send common terms in updated \mathbf{Ad} to subdomain $i-1$
- receive updated non-active terms in \mathbf{Ad} from subdomain $i+1$
- send terms in \mathbf{Ad} common to subdomain $i-1$ to subdomain $i-1$
- receive updated non-active terms in \mathbf{Ad} from subdomain $i+1$ (the subdomain now contains the complete \mathbf{Ad} for all nodes)

Stage 6: Calculate α and update \mathbf{x} and \mathbf{r}

- $dAd = \mathbf{dAd}$ for all nodes
- send dAd to *master* processor
- receive α from *master*
- $\mathbf{x} = \mathbf{x} + \alpha \mathbf{d}$ and $\mathbf{r} = \mathbf{r} - \alpha \mathbf{Ad}$ for all nodes

Stage 7: Check for convergence

- $r2 = \mathbf{r} \cdot \mathbf{r}$ for all active nodes
- send $r2$ to *master* processor
- receive signal from *master* processor
- if signal is convergence signal then exit

Stage 8: Update \mathbf{g}

- Repeat stage 4 (omitting last three steps) for solution of $\mathbf{Bg} = \mathbf{r}$

Stage 9: Update \mathbf{d} with new β and repeat cycle

- $rg = \mathbf{r} \cdot \mathbf{g}$ for all active nodes

- send rg to *master* processor
- receive β from *master* processor
- $\mathbf{d} = \mathbf{r} + \beta \mathbf{d}$ for all nodes
- $k = k + 1$, go to stage 5

Note that in each subdomain the vector of heads is always known for all nodes in that subdomain even though several operations only act on the active set of nodes. It should also be noted that, provided facilities are available within the computer environment, the communication send signals may be initialised but do not need to be completed before the next operation. Thus the *slave* process is not delayed if a neighbour is not perfectly synchronised as the message is retained in a buffer whilst the *slave* continues with other communication or calculation steps. This could also be done in receive mode although normally the statement following a receive depends on the contents of that message.

Master process

Stage 1: Initialise

- receive $b2$ from all *slaves*, accumulate and calculate $\|\mathbf{b}\|_2$
- receive rg from all *slaves* and accumulate

Stage 2: Calculation of α

- $rg_{old} = rg$
- receive dAd from all *slaves* and accumulate
- calculate $\alpha = dAd/rg$ and send to all *slaves*

Stage 3: Check convergence

- receive $r2$ from all *slaves*, accumulate and calculate $\|\mathbf{r}\|_2$
- check convergence and send signal to *slaves*
- exit if converged

Stage 4: Calculation of β

- receive rg from all *slaves* and accumulate
- calculate $\beta = rg/rg_{old}$ and send to all *slaves*
- go to stage 2