Block Iterative Methods for Three-Dimensional Groundwater Flow Models

K.J.Neylon

September 1991

Submitted to the

Department of Mathematics,

University of Reading,

in partial fulfilment of the requirements for the

Degree of Master of Science

Some block iterative methods which are applied to the matrices resulting from the finite element approximation of a three dimensional non-linear partial differential equation are compared. The nodal ordering is shown to have an important important effect on the convergence of these methods. The possibility of implementing the methods on a transputer system is discussed.

I would like to thank Dr.M.J.Baines and Dr.N.K.Nicholls, of the Mathematics

List of Figures

List of bles

Not tion

Symbol	Meaning
e_j	unit gravitational vector
h	size of element as proportion of whole region
H	size of rectangular prism element in the z-direction
k	iteration number
k_{rw}	relative permeability with respect to the water phase
K_{ij}	saturated hydraulic conductivity tensor
l	size of rectangular prism element in the x-direction
N_J	three dimensional linear basis function
m	size of rectangular prism element in the y-direction
p	number of processors in transputer network
s	number of diagonal blocks in the matrix system
S_w	water saturation
x_1, x_2, x_3	Cartesian co-ordinate directions
$x\ ,\ y\ ,\ z$	Cartesian co-ordinate directions
ψ	pressure head
ψ_J	approximate nodal pressure head on the finite element grid
$\rho(M)$	spectral radius of matrix M
ω	over-relaxation parameter
$\mathcal{R}$	set of real numbers

In general, there are two approaches when solving matrix problems - direct and iterative. Direct methods solve the problem in a known (finite) number of operations and errors in the solution arise entirely from rounding errors introduced in the computation. Iterative methods generate a sequence of approximate solutions (iterates) which tend towards the solution of the problem; these methods usually exploit any sparsity in the matrix more than direct methods.

This dissertation is concerned with block iterative methods for the solution of large sparse systems of equations arising from a three dimensional finite element model of groundwater flow, consideration is given to the possibility of implementing these algorithms on a parallel computer. The investigations carried out can be divided into three main areas.

The first area of investigation is the choice of node numbering scheme. At present, many three dimensional finite element models use a single node numbering scheme over the whole region, chosen without any consideration given to the predominant direction of flow in the region (e.g. the vertical slicing approach

The matrix systems used during the course of this project are generated by finite element approximations to simple three dimensional groundwater flow prob-

$\overline{}$ ij rw $\overline{}$ j w s $\overline{}^{w}$ $\overline{}$

ij

rw

j

w

S_s is the specific storage coefficient

ϕ is the porosity

t is time

and x_1 , x_2 , x_3 are Cartesian co-ordinate directions.

In order to simplify this equation, the range of problems considered is restricted to those which are in steady state (i.e. no time dependence) and have no sources or sinks. Under these conditions (2.1) becomes

$$\frac{\partial}{\partial x_i}\left[K_{ij}k_{rw}\left(\frac{\partial \psi}{\partial x_j}+e_j\right)\right]=0 \qquad i,j=1,2,3 \tag{2.2}$$

This equation is non-linear since both the relative permeability and the saturated hydraulic conductivity tensor are functions of the solution (the pressure head) in the unsaturated part of the region [4].

2.2 Finite Element Approximation

The standard Galerkin method [2, 12] is used to generate the three dimensional finite element approximation to (2.2). A trial function, $\tilde{\psi}$, approximating the unknown, ψ, is selected of the form

$$\tilde{\psi}(x_i)=\sum_{J=1}^{n}\psi_J N_J(x_i) \qquad i=1,2,3$$

where ψ_J is the approximate nodal value of $\tilde{\psi}$ on the finite element grid

N_J is a three dimensional linear basis function

n is the number of nodes in the finite element grid

The partial differential equation (2.2) is multiplied by a test function, w, where

$$w=N_I(x_i) \qquad i=1,2,3 \qquad I=1,2,...,n$$

is the same basis function as for the trial space, and then integrated over the whole region. Green's theorem is applied to transform the second derivative term

(which is not defined for linear basis functions) to a first derivative term (which

IJ J I
J

IJ ij rw $\frac{I}{i}$ $\frac{J}{j}$
e R^e

I ij rw $\frac{I}{i}$ j
e R^e

IJ

I

IJ J

2

$local$ $local$ th

$local$ $local$

i.e. $(i_{local}, j_{local}) \Rightarrow (i_{global}, j_{global})$

Since a node at position (i_{global}, j_{global}) will be a part of more than one element, the contribution from each element must be combined to give the global matrix entry.

2.2.1 Computation of Element Matrices

Only rectangular prism elements are used in the discretisation of the region - hence only rectangular regions can be modelled. These elements are chosen so that they are all of the same size and are aligned with the global Cartesian coordinates (with the y-direction taken as vertically upwards). A typical element is shown in Figure 2.1. The element seepage matrices, $[A]^e$, and element right

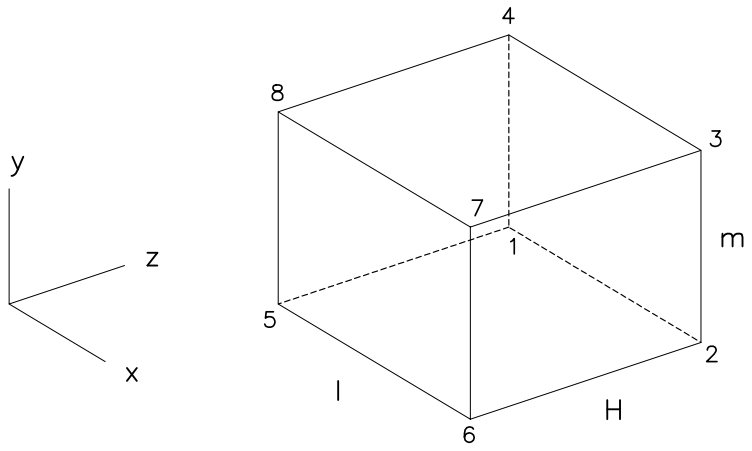


Figure 2.1: Dimensions and Orientation of lements

hand side vectors, $\{F\}^e$, are generated using the influence coefficient technique described in [9]. This technique uses the fact that the elements are simple and regular to calculate the coefficients in the element matrices and vectors for a general element. It cannot be used effectively on regions with more widely varying

types of element geometry where numerical quadrature must be used to generate the coefficients for each element independently.

The expressions for the seepage matrix and right hand side vectors (2.4),(2.5) are simplified by taking the hydraulic properties to be constant within each element (and equal to the centroidal value). This corresponds to one-point three-dimensional Gaussian quadrature on a rectangular region which introduces an error of $0(h^2)$ i.e. the same order of accuracy as the finite element approximation. The element seepage matrix is given by

$$\begin{aligned}
[A]^e &= \langle K_{xx} k_{rw} \rangle \frac{mH}{2l} [A^{xx}]^e + \langle K_{yy} k_{rw} \rangle \frac{lH}{2m} [A^{yy}]^e \\
&+ \langle K_{zz} k_{rw} \rangle \frac{lm}{2H} [A^{zz}]^e + \langle K_{xy} k_{rw} \rangle \frac{H}{2} [A^{xy}]^e \\
&+ \langle K_{yz} k_{rw} \rangle \frac{l}{2} [A^{yz}]^e + \langle K_{zx} k_{rw} \rangle \frac{m}{2} [A^{zx}]^e
\end{aligned}$$

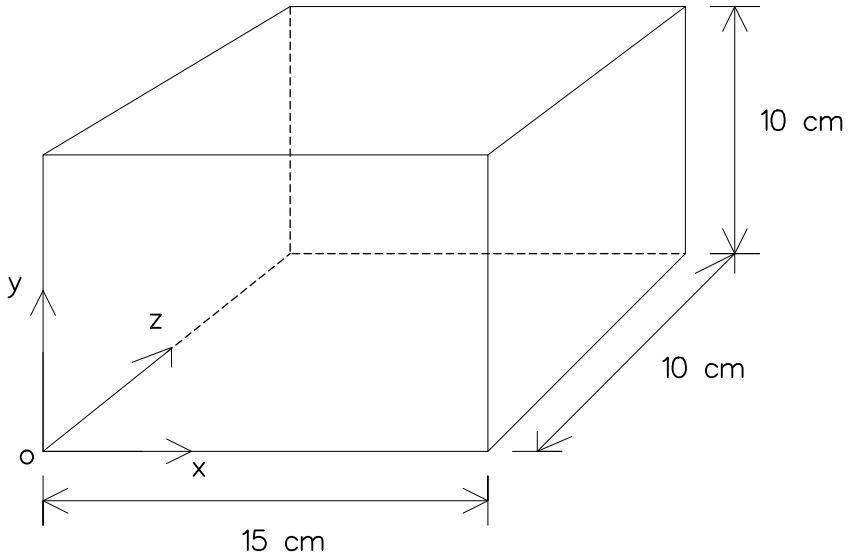
where $\langle \cdot \rangle$ denotes the centroidal value and, because the global coordinate system is oriented such that y is vertically upwards, then the element right hand side vector is given by

$$\{F\}^e = -\langle k_{rw} \rangle [\langle K_{xy} \rangle \{F^x\}^e + \langle K_{yy} \rangle \{F^y\}^e + \langle K_{zy} \rangle \{F^z\}^e],$$

The influence coefficient matrices and vectors are as given in Appendix A.

2.3 Sample Problem

The problem used to examine the nodal ordering and block iterative methods in this dissertation is a modified form of an unsaturated flow example given in [3]. This example is essentially a two-dimensional problem, to be solved by a three-dimensional finite-element package by keeping all the characteristics of the problem constant across the width of the region and using a single element across the width.


10 cm
y
z
10 cm
o
x
15 cm

In this chapter, the block matrix theory used during the dissertation is outlined. Some standard sequential iterative block matrix algorithms are given and the performance of these is compared for some sample problems.

Where appropriate, the theory is directed at a symmetric, block tri-diagonal

$$
\begin{array}{ccccccc}
1 & 1 & & & & -1 & -1 \\
T_1 & 2 & 2 & & & -2 & -2 \\
 & & & & & & \\
 & & T_{s-2} & s-1 & s-1 & -s-1 & -s-1 \\
 & & & T_{s-1} & s & -s & -s
\end{array}
$$

In general, if A is an complex matrix with eigenvalues, $_i$ $= 1 \ldots$ then

$$(\ ) = \max_{1 \le i \le n} \ _i$$

is the spectral radius of the matrix A.

[13]

The asymptotic rate of convergence of a linear stationary iterative method (such as those described in Section 3.2) with iteration matrix A, $_\infty(\ )$, is

$$_\infty(\ ) = \ \ln \ (\ )$$

It is a measure of the rapidity of convergence of the iterative method.

For a square matrix, A, the condition number, $(\ )$, is

$$(\ ) = \ ^{-1}$$

$$- \quad -$$

$$_2 \qquad _2 \ ^{-1} \ _2$$

Consistent Ordering [6]

The $s \times s$ block matrix, A, consisting of blocks $A_{i,j}$ $(i, j = 1, \ldots, s)$ is consistently ordered if for some t there exist disjoint non-empty subsets $S_1, \ldots, S_t$ of $\{1, 2, \ldots, s\}$ such that

$$\bigcup_{i=1}^{t} S_i = \{1, 2, \ldots, s\}$$

and such that if $A_{i,j} \neq 0$ with $i \neq j$ and S_k is the subset containing i, then $j \in S_{k+1}$ if $j > i$, and $j \in S_{k-1}$ if $j < i$.

p-cyclic Matrix [13]

An $n \times n$ complex matrix A is weakly cyclic of index k (> 1) if there exists an $n \times n$ permutation matrix, P, such that PAP^T is of the form

$$PAP^T = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & A_{1,k} \\ A_{2,1} & 0 & & & 0 & 0 \\ 0 & A_{3,2} & \ddots & & & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & A_{k,k-1} & 0 \end{pmatrix}$$

If the block Jacobi matrix of A (Section 3.2.2) is weakly cyclic of index p (≥ 2) then A is a p-cyclic matrix. It can be shown that a block tri-diagonal matrix is a consistently ordered 2-cyclic matrix [13].

Block Diagonal Dominance

An $s \times s$ block matrix, A, consisting of blocks $A_{i,j}$ $(i, j = 1, \ldots, s)$ is block

diagonally dominant [5] if, for $i = 1, \ldots, s$

$$\|A_{i,i}^{-1}\|_\infty \sum_{\substack{j=1 \\ j \neq i}}^{s} \|A_{i,j}\|_\infty \leq 1$$

This definition can be modified to give an expression for the *degree of block diagonal dominance*, σ, of a block tridiagonal matrix as follows.

$$\sigma = \max_{i=1,.,s} \left\{ \|A_{i,i}^{-1}\|_\infty (\|A_{i,i-1}\|_\infty + \|A_{i,i+1}\|_\infty) \right\} \qquad A_{1,0} = A_{s,s+1} = 0$$

The smaller the value of σ, the stronger the block diagonal dominance.

3.2 lassical Iterative Methods for Block Matrices

The classical iterative techniques [13] for solving the linear system,

$$A\underline{x} = \underline{b} \qquad \underline{b} \in \mathcal{R}^n, A \in \mathcal{R}^{n \times n} \tag{3.2}$$

are based on splittings of the matrix, A. The standard notation for a splitting is

$$A = M - N \qquad (M \quad \text{invertible}) \tag{3.3}$$

giving rise to the iterative method

$$M\underline{x}^{k+1} = N\underline{x}^k + \underline{b} \qquad k \geq 0 \tag{3.4}$$

where $\underline{x}^{k+1}$ is the next iterate based on the previous iterate, $\underline{x}^k$.
$M^{-1}N$ is the iteration matrix and it can be shown [5] that $\rho(M^{-1}N) < 1$ is a necessary and sufficient condition for the method to converge to the true solution, $\underline{x} = A^{-1}\underline{b}$, for any initial iterate, $\underline{x}^0$.

The matrix A may be decomposed as follows.

$$A = \quad -L - U \tag{3.5}$$

where consists of the diagonal blocks of A

consists of the sub-diagonal blocks of A

consists of the super-diagonal blocks of A.

The relationship between the splitting (3.3) and the decomposition (3.5) of the matrix governs which of the classical block iterative methods is used.

If the relationship between the splitting and the decomposition of the matrix is

$$= \qquad = \quad +$$

then (3.4) becomes the block Jacobi (BJ) method.

$$_^{k+1} = (\quad + \quad)_^{k} + _$$

th

$$_i \; _i^{k+1} \qquad _i \qquad \overset{T}{}_{i-1} \; _{i-1}^{k} \qquad _i \; _{i+1}^{k}$$

i

3.2.2 Block Gauss-Seidel Method

If

$$M = \quad - L \quad , \quad N = U$$

then (3.4) becomes the block Gauss-Seidel (BGS) method.

$$(\quad - L)\underline{x}^{k+1} = U\underline{x}^{k} + \underline{b}$$

Applying this iteration technique to the i^{th} row of blocks in (3.1) gives

$$A_i\underline{x}_i^{k+1} = \underline{b}_i - (B_i\underline{x}_{i+1}^{k} + B_{i-1}^{T}\underline{x}_{i-1}^{k+1}) \qquad i = 1, .., s \tag{3.7}$$

Since the block tridiagonal matrix is a consistently ordered p-cyclic matrix (Section 3.1) and the diagonal blocks are non-singular then, from [13], if λ is a non-zero eigenvalue of the BGS iteration matrix ($\mathcal{L}_1$) and $\mu^2 = \lambda$, then μ is an eigenvalue of the BJ iteration matrix, B. Thus the BJ method converges if and only if the BGS iteration method converges, and if both converge then

$$\rho(\mathcal{L}_1) = \{\rho(B)\}^2 < 1$$

and consequently

$$R_\infty(\mathcal{L}_1) = 2R_\infty(B) \tag{3.8}$$

so in order to achieve the same accuracy, roughly twice as many iterations of the BJ method are required as for the BGS method.

The BGS method (3.7) is much less amenable to parallelisation than the BJ method because the right-hand side has a term from the current iteration, hence the i block systems must be solved in sequence.

3.2.3 Block Successive Over-Relaxation Method

If the solutions from the previous iteration, $\underline{x}_k$, and the latest iteration, $\tilde{\underline{x}}^{k+1}$, in the BGS method are combined with the use of an over-relaxation parameter, ω,

then the result is the next iterate in the block successive over-relaxation (BSOR) method, $\underline{x}^{k+1}$ i.e.

$$\underline{x}^{k+1} = (1-\omega)\underline{x}^{k} + \omega\tilde{\underline{x}}^{k+1}$$

This is equivalent to the relationships

$$M = \frac{1}{\omega}(\quad - \omega L) \quad , \quad N = \frac{1}{\omega}\{(1-\omega) \quad + \omega U\}$$

so applying this iteration technique to the i^{th} row of blocks in (3.1) gives

$$A_i\underline{x}_i^{k+1} = \omega\underline{b}_i - \{(1-\omega)A_i\underline{x}_i^{k} + \omega(B_i\underline{x}_{i+1}^{k} + B_{i-1}^{T}\underline{x}_{i-1}^{k+1})\} \qquad i = 1,..,s \qquad (3.9)$$

If the value of the over-relaxation parameter is set to one, i.e. $\omega = 1$, then the block successive over-relaxation method is the same as the BGS method. As with the BGS method, this method is not easily parallelised because the right-hand side has a term from the current iteration so the i block systems must be solved in sequence.

From [13], if A is a consistently ordered p-cyclic matrix then the corresponding BSOR iteration will only converge if

$$0 < \omega < \left(\frac{p}{p-1}\right)$$

so since a block tri-diagonal matrix is a consistently ordered 2-cyclic matrix then the BSOR method (3.9) will only converge if

$$0 < \omega < 2$$

The value of ω which gives the fastest rate of convergence is ω_{opt}, i.e.

$$\min_{\omega}\{\rho(\mathcal{L}_\omega)\} = \rho(\mathcal{L}_{\omega_{opt}})$$

where $\mathcal{L}_\omega$ is the BSOR iteration matrix. Since BGS is a special case of BSOR then

$$\rho(\mathcal{L}_{\omega_{opt}}) \leq \rho(\mathcal{L}_1)$$

ω_{opt} 1

ω_{opt} 1

so $\phi(x)$ is minimised when

$$(\underline{x} - A^{-1}\underline{b})^T A(\underline{x} - A^{-1}\underline{b}) = 0$$

$$\text{i.e.} \qquad \underline{x} = A^{-1}\underline{b}$$

and the minimum value of the functional is

$$\phi(\underline{x}) = -\frac{1}{2}\, \underline{b}^T A^{-1}\, \underline{b} \quad \square$$

Thus minimising the functional (3.12) and solving (3.2) are equivalent problems. It is possible to minimise $\phi(\underline{x})$ by the steepest descent method [5]. At a point, $\underline{x}^k$, the functional $\phi^k \;\; (= \phi(\underline{x}^k) \;)$ decreases most rapidly in the direction of $-\nabla\phi^k$ which is the residual, $\underline{r}^k = \underline{b} - A\underline{x}^k$. The new iterate, $\underline{x}^{k+1}$, is given by

$$\underline{x}^{k+1} = \underline{x}^k + \alpha \underline{r}^k \qquad\qquad \alpha \in \mathcal{R}$$

where α is chosen such that ϕ^{k+1} is minimised. This is achieved by setting

$$\frac{\partial \phi^{k+1}}{\partial \alpha} = 0$$

which gives

$$\alpha = \frac{(\underline{r}^k)^T \underline{r}^k}{(\underline{r}^k)^T A \underline{r}^k}$$

However, if the condition number of A, $\kappa(A)$, is large then the level curves of ϕ are very elongated hyperellipsoids and minimisation corresponds to finding the lowest point on a relatively flat, steep-sided valley; in the steepest descent method, we are forced to traverse back and forth across the valley rather than down the valley, the gradient directions that arise during the iteration are close thus making the progress towards the minimum point slow.

To overcome this problem, we successively minimise ϕ along a set of search directions $\{\underline{p}^0, \underline{p}^1, \ldots\}$ which do not necessarily correspond to the residual vectors $\{\underline{r}^0, \underline{r}^1, \ldots\}$. Then

$$\underline{x}^{k+1} = \underline{x}^k + \alpha \underline{p}^k$$

and again ϕ^{k+1} is minimised when

$$\frac{\partial \phi^{k+1}}{\partial \alpha} = 0$$

which gives

$$\alpha = \frac{(\underline{p}^k)^T \underline{r}^k}{(\underline{p}^k)^T A \underline{p}^k}$$

In order to ensure a reduction in the size of ϕ, $\underline{p}^k$ must not be orthogonal to $\underline{r}^k$,

$$\text{i.e.} \quad (\underline{p}^k)^T \underline{r}^k \neq 0$$

and the search directions are taken as A-conjugate to all the previous search directions in order to ensure convergence [5].

$$\text{i.e.} \quad P_{k-1}^T A \underline{p}^k = 0 \quad \text{where} \quad P_{k-1} = \{\underline{p}^0, \ldots, \underline{p}^{k-1}\} \in \mathcal{R}^{n \times k}$$

This is the basis of the CG method. From [5], the required search directions, $\underline{p}^k$, are found by taking

$$\underline{p}^k = \underline{r}^k + \beta \underline{p}^{k-1}$$

where

$$\beta = \frac{(\underline{r}^k)^T \underline{r}^k}{(\underline{r}^{k-1})^T \underline{r}^{k-1}}$$

3.3.1 Convergence

In exact arithmetic, the solution is obtained in at most n steps so the method has the property of finite termination (and would then be considered a direct method as opposed to an iterative method). However, in the practical situation of finite precision arithmetic, rounding errors lead to a loss of orthogonality among the residuals and finite termination is not mathematically guaranteed.

In any case, when the CG method is applied, n is usually so big that $O(n)$ iterations represents an unacceptable amount of work. Hence it is customary

to regard the method as a genuinely iterative technique with termination based upon an iteration maximum, $_{max}$ and the residual norm.

$$A \quad T$$

$$k \quad A \qquad 0 \quad A \quad \frac{2}{2} \quad k$$

2

k

2

2

$n \times n$

-1

$$-1 \qquad -1$$

$$-1 \qquad -1$$

- M^{-1} is a good approximation to A^{-1} so that $\tilde{A}$ is "close" to the identity
- M is computationally simple and inexpensive to invert due to the occurrence of M^{-1} in the preconditioned CG algorithm.

In this dissertation, the pre-conditioner is taken as the diagonal blocks of the matrix A to enable the BCG method to be compared to the classical block iterative methods (which can be viewed as having the same diagonal block pre-conditioner) in Chapter 5.

3.3.3 Pre-conditioned Block Conjugate Gradient Algorithm

Applying the preconditioned CG method to the block tri-diagonal structure (3.1) leads to the following pre-conditioned block conjugate (BCG) algorithm.

$$\underline{p}_i^0 = (M^{-1}\underline{r}^0)_i = (M^{-1}(\underline{b} - A\underline{x}))_i \qquad i = 1, \ldots, s$$

$k = 0$

while $\| \underline{r}^k \| \neq 0$ and $k < k_{max}$

$$k = k + 1$$

$$\alpha_k = \frac{\sum_{i=1}^{s} (\underline{r}_i^k)^T (M^{-1}\underline{r}^k)_i}{\sum_{i=1}^{s} (\underline{p}_i^k)^T (A\underline{p}^k)_i}$$

$$\underline{x}_i^{k+1} = \underline{x}_i^k + \alpha_k \underline{p}_i^k \qquad i = 1, \ldots, s$$

$$\underline{r}_i^{k+1} = \underline{r}_i^k - \alpha_k (A\underline{p}^k)_i \qquad i = 1, \ldots, s$$

$$\beta_k = \frac{\sum_{i=1}^{s} (\underline{r}_i^{k+1})^T (M^{-1}\underline{r}^{k+1})_i}{\sum_{i=1}^{s} (\underline{r}_i^k)^T (M^{-1}\underline{r}^k)_i}$$

$$\underline{p}_i^{k+1} = (M^{-1}\underline{r}^{k+1})_i + \beta_k \underline{p}_i^k \qquad i = 1, \ldots, s$$

Chapter

Investigation of Nodal Ordering

In this chapter, the effect of the nodal ordering on the matrix structure generated by the finite element approximation described in Chapter 2 is examined. Two different ordering approaches are investigated on a sample problem which is solved by a block Jacobi iteration (Chapter 3). Using the results of this investigation, some recommendations are given for the nodal ordering in the discretisation of the region.

4.1 Discretisation of Region

In order to investigate the effect of the orientation of the node numbering scheme with respect to the predominant direction of flow, two different ordering approaches are used.

The first is a vertical slicing method. In this method, the region is divided by equally spaced vertical slices; then the vertical block sub-regions between pairs of adjacent slices are divided into equally sized rectangular prism elements. The nodes in the region are numbered on each slice in turn by natural ordering with the incrementation being faster in the vertical direction. An example of this is shown in Figure 4.1.

4.2 Investigation of onvergence

The system of equations (2.3) is non-linear. In order that this system can be solved by matrix methods, a non-linear iteration technique is used. The system is linearised about the latest non-linear iterate to generate the matrix system to be solved.

The area of interest in these tests is the solution of the linearised system. Hence the non-linear iteration is not performed to convergence, only a single non-linear iteration being used and the convergence of the inner iteration method (used to solve the linearised system) is examined.

Due to the slicing methods used for the nodal ordering in the region, the global seepage matrix will have a symmetric and block tridiagonal structure as shown in (3.1). The connection between nodes on a slice in the approximate equation (2.3) appears as an entry in the diagonal blocks. The connection between nodes on adjacent slices appears as an entry in the super- and sub-diagonal blocks. If the direction of flow is in the direction of the slicing then the entries in the diagonal blocks will be large relative to those in the off-diagonal blocks and vice-versa.

The problem described in Section 2.3 is selected because it is an example of unsaturated flow. From [1], flow in the unsaturated region is predominantly in the vertical direction (as opposed to saturated flow which is predominantly in the horizontal direction). Hence, the block diagonal dominance should be stronger with the vertical slicing and so the convergence of block iterative methods should be faster with this slicing.

From (2.4), it can be seen that the two main influences on the relative sizes of the entries in the seepage matrix are K_{ij} and k_{rw}. The effects of these two are examined independently.

rw

$initial$

rw

ij

xx

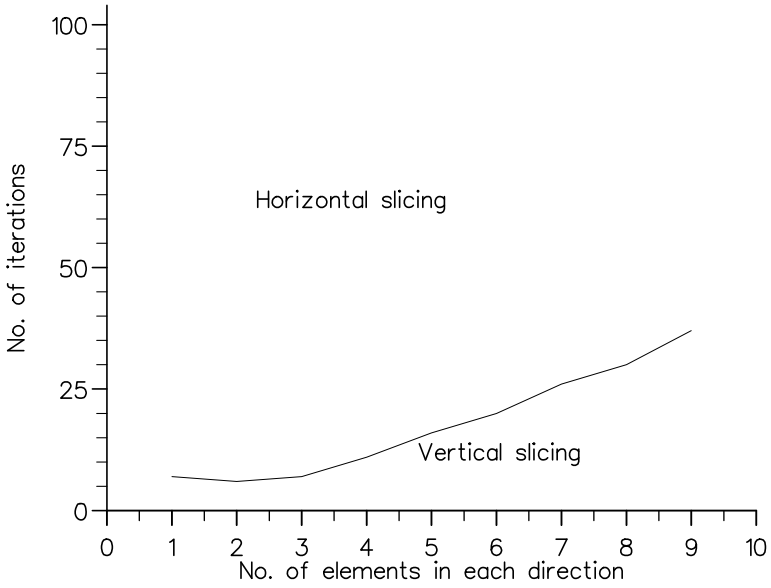
yy

zz

xy

yz

zx


100
75
50
25
0
No. of iterations
Horizontal slicing
Vertical slicing
0
1
2
3
4
5
6
7
8
9
10
No. of elements in each direction

4.2.2 Effect of k_{rw}

In order to eliminate the effect of K_{ij} in these tests, its components are taken as

$$\begin{aligned} K_{xx} &= 1.0 \quad \text{cm/day} \\ K_{yy} &= 1.0 \quad \text{cm/day} \\ K_{zz} &= 1.0 \quad \text{cm/day} \\ K_{xy} &= 0.0 \quad \text{cm/day} \\ K_{yz} &= 0.0 \quad \text{cm/day} \\ K_{zy} &= 0.0 \quad \text{cm/day} \end{aligned}$$

Since k_{rw} manifests itself as a non-linear effect, then a non-linear iteration must be performed to produce the second non-linear iterate (which more closely resembles the true form of the solution) and this is used to generate the linearised matrix system to be solved by the block iterative algorithm. The results of the tests are shown in Figure 4.4. This test case will be used to generate matrices in later parts of this dissertation; the test case with vertical slicing will be referred to as Problem 3 and the test case with horizontal slicing will be referred to as Problem 4.

From Figure 4.4 it can be seen that, again, the vertical slicing gives a faster rate of convergence. The difference in convergence rates is an indirect result of the boundary conditions imposed on the problem. For this problem, the boundary conditions cause the vertical flow to dominate. During the non-linear iteration, the flow from the previous iteration is used to calculate the linearised relative permeability throughout the region. The magnitude of this effect is governed by the non-linearity of the constitutive relations.

or, in the notation of (3.1),

$$\infty = \max_{i=1,\ldots,s} \quad {}^{-1}_{i} \quad {}^{T}_{i-1} \quad \infty + \quad {}^{-1}_{i} \quad {}_{i} \quad \infty \qquad {}_{0} = \quad {}_{s} = 0$$

so, comparing with (4.1), it can be seen that

$$\infty =$$

Therefore the stronger the block diagonal dominance of the pre-conditioned matrix, the smaller the -norm of the block Jacobi iteration matrix. It is conjectured that, for the types of matrices generated in this dissertation; the smaller the -norm of the block iteration matrix, the smaller its spectral radius and hence the faster the asymptotic rate of convergence of the classical iteration methods.

The direct connection between the norm of the iteration matrix and its spec-

1 2 3 4

1 3

2 4

Matrix	σ	$\rho(B)$	N
A_1	0.8859	0.4382	6
A_2	0.9648	0.5957	12
A_3	0.9746	0.7215	17
A_4	1.0069	0.8147	27

Table 4.1: Degree of Block Diagonal Dominance for xample Matrices

stronger block diagonal dominance and faster convergence, as expected from the previous results in this chapter .

4.4 Recommendations for Nodal Ordering Approach

Plane slicing techniques (with the same number of nodes on each slice) are used in this project primarily because of coding convenience. As stated in [9], mesh grading capability is somewhat limited with this approach, particularly when the flow region contracts or expands in the direction normal to the plane of the slice.

In practice, a more general spatial discretisation will be required to model complex geometries and flows. The generalisations required would be to have

- a varying number of nodes on each slice
- various different types of geometry of element (e.g. tetrahedral)
- no requirement that the faces of an element which lie on adjacent slices be parallel.

With these generalisations (particularly the last one), the horizontal and vertical slicing terminology loses meaning. However the convergence concepts out-

In this chapter, the performance of the block iterative methods described in Chapter 3 on the problems described in Chapter 4 is discussed.

It has already been stated that the matrices generated by the methods described in Chapters 2 and 4 give rise to symmetric block tridiagonal matrices (3.1). Hence only the diagonal and super-diagonal blocks need be generated and stored in the program.

ij ij

ij

where is the number of elements in the y-direction for vertical slicing, and the number of elements in the z-direction for horizontal slicing. This is not easily seen from the example matrices given in Appendix B because these are too small.

When applying the block iterative methods of Chapter 3, there is a matrix system corresponding to each of the diagonal blocks to solve at each iteration.

T

T

opt opt

opt

It is noted that if the results from Problem 1 and Problem 2, and the results from Problem 3 and Problem 4, are compared for each of the block iteration methods, then the matrices arising from the vertical slicing node ordering converge faster than those from the horizontal slicing nodal ordering, showing that the results of Chapter 4 apply to all the iteration methods and not just the block Jacobi method.

Size of Problem	Number of iterations				
	BJ	BGS	BSOR = 1 5	BSOR($_{opt}$) = $_{opt}$	BCG
		= M 1			

				^{opt} $_{opt}$	

Size of Problem $n \quad n \quad n$	Number of iterations				
	BJ	BGS	BSOR $\omega = 1.5$	BSOR(ω_{opt}) $\omega = \omega_{opt}$	BCG
2	17	11	12	6 (1.20)	5
3	27	17	12	8 (1.29)	9
4	48	29	13	11 (1.43)	10
5	74	45	15	13 (1.53)	13
6	90	55	21	16 (1.57)	16
7	121	74	30	18 (1.64)	18
8	139	86	35	21 (1.66)	22
9	172	107	46	24 (1.69)	24

Size of Problem $n \quad n \quad n$	Number of iterations				
	BJ	BGS	BSOR $\omega = 1.5$	BSOR(ω_{opt}) $\omega = \omega_{opt}$	BCG
2	27	14	12	7 (1.34)	6
3	43	24	12	9 (1.40)	9
4	80	43	15	13 (1.53)	12
5	111	64	25	16 (1.62)	15
6	142	76	31	18 (1.64)	18
7	184	99	42	21 (1.70)	21
8	200	112	48	23 (1.72)	24
9	225	136	61	27 (1.75)	27

Ch pter 6

r llel Implement tion of Block M trix Algorithms

Block algorithms access data from memory in large chunks or blocks; in parallel computer architectures, input/output is usually more expensive than floating point computations, making block algorithms attractive for these machines. In this chapter, the possibility of parallel implementation of some of the block matrix methods described in Chapter 3 is discussed.

The block matrix algorithms are directed at a transputer type architecture. In the first part of this chapter, the transputer is described and some of the important properties required for the design of algorithms for this type of machine are identified.

6.1 Description of Transputer Architecture

6.1.1 The Transputer

A transputer (INMOS 1985) is a single chip microprocessor [7], designed as a building block for parallel processors. To facilitate this it has memory and

four links to connect one transputer to another, all on a single VLSI chip. One important feature of VLSI technology is that communication between devices is very much slower than communication within a device. In transputer parts, all components execute concurrently; each of the four links and the floating point co-processor can all perform useful work while the processor is executing other instructions.

Transputers are designed to implement the parallel programming language OCCAM very efficiently. However, due to the large amount of resources residing in FORTRAN code, parallel FORTRAN is available, e.g. the 3L Parallel FORTRAN package [11].

6.1.2 Transputer Networks

The four links on a transputer allow several interconnection topologies to be implemented. Some examples of these are shown in Figure 6.1. The network topology of a system is a hardware feature. In a network, one transputer functions as the 'host', by which the network is directed. The host is installed in a PC and does not take part in the computational process itself.

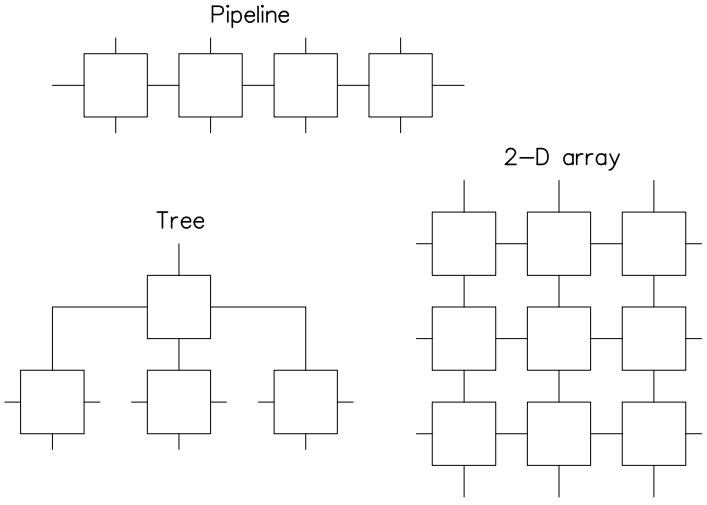
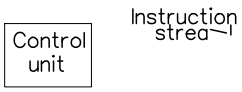
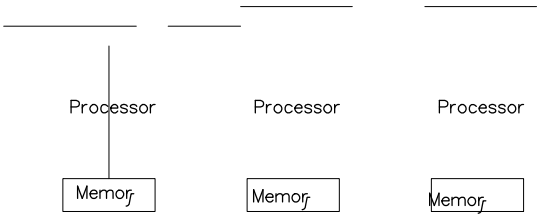


Figure 6.1: Transputer Network Topologies


Control unit
Instruction stream


Processor
Processor
Processor
Memory
Memory
Memory

between tasks, they communicate via channels which are one way communication paths with a fixed starting point and a fixed finishing point. A complete application is viewed as a collection of one or more tasks, a single transputer can execute more than one task. A task may contain several concurrent threads.

A thread is a FORTRAN subroutine. Threads are dynamically created and destroyed during execution and are potentially heirarchical (i.e. a thread may be created from within another thread). Threads have shared common blocks.

$—_{0,i}$ $—_i$

T_{i-1} i i

$—_i^k$

3. Receive $\underline{\ }^{k}_{i-1}$ from task 1 and $\underline{\ }^{k}_{i+1}$ from task $+1$.

4. Modify the right hand side vector, i.e.

$$\tilde{\underline{\ }}_{i} = \underline{\ }_{i} \quad (\ {}^{T}_{i-1}\underline{\ }^{k}_{i-1} + \ {}_{i}\underline{\ }^{k}_{i+1})$$

using banded matrix-vector multiplication.

5. Performpl) $\quad {}_{i} \quad {}^{T}$

$${}^{T}\underline{\ }^{k+1}_{i} \qquad \underline{\ }_{i}$$

$\underline{\ }^{k+1}_{i}$

Another possibility is to pass all the results along the chain to a single transputer (where convergence is tested) after each iteration. However, this means that the bulk of the execution time of the program is consumed by the communication and so is not efficient.

Another approach is to pass the results along the chain to the first (non-host) processor where the convergence test is applied by a separate task - if convergence has occurred then this task informs the host to terminate the process and output the results. Since each transputer communicates with its neighbours at the end of each iteration then convergence is only tested after every $(p-1)$ iterations so some unnecessary iterations can be performed. In order to implement this approach, the block iteration task algorithm would have to be altered to incorporate the passing of data along the chain after each iteration; this is omitted from the task algorithm given for the block Jacobi algorithm so that the parallelism of the block iterative method is more clear (for this reason, no termination criterion will be given for the other parallel block iterative methods described in this chapter).

6.2.3 Block Gauss-Seidel Method

This method is not as easily parallelised since each of the block matrix equations uses the latest iterate from the previous block matrix equation. However, adapting the parallel point GS method from [10] for a block algorithm gives a possible approach for the parallel implementation of the BGS method.

Consider a general task in the middle of the chain of tasks. The previous task calculates its $(k+1)^{th}$ iterate and passes it to this task. This task then calculates its own $(k+1)^{th}$ iterate and passes it onto the next task. Up to this point there has been no departure from the sequential BGS method described in Section 3.2.2. However, the current task also passes its $(k+1)^{th}$ iterate back to the previous task which can now calculate its $(k+2)^{th}$ iterate. In this way, more

than one task is active at any particular instant with different iteration levels being performed concurrently.

In this approach, some of the blocks will do extra iterations after the iteration which gives the solution to the required tolerance, so it appears that some processor time is wasted. However, in a parallel applications, the execution time is the chief measure of performance and, since the extra iterations are done at the same time as the required iterations then these extra iterations do not represent any waste in the parallel performance of the method. The algorithm for task i in the middle of the system, is given after the end of this paragraph; the same assumptions as for the BJ method in Section 6.2.2 are made here.

Block Gauss-Seidel Method : Task i Algorithm

1. $k = 0$
2. Receive $\underline{x}_{i-1}^{k+1}$ from task $i-1$.
3. Receive $\underline{x}_{i+1}^{k}$ from task $i+1$.
4. Modify the right hand side vector, i.e.

$$\tilde{\underline{b}}_i = \underline{b}_i - (B_{i-1}^T \underline{x}_{i-1}^{k+1} + B_i \underline{x}_{i+1}^{k})$$

   using banded matrix-vector multiplication.

5. Perform a banded Cholesky decomposition on A_i i.e. $A = L \quad L^T$ (unless this has already been computed at a previous iteration).
6. Solve

$$L \quad L^T \underline{x}_i^{k+1} = \tilde{\underline{b}}_i$$

   for $\underline{x}_i^{k+1}$ by banded forward substitution, division by the diagonal and banded backward substitution.

7. Send $\underline{x}_i^{k+1}$ to task $i+1$.

8. Send $\underline{x}_i^{k+1}$ to task $i-1$.

9. $k = k + 1$

10. Return to step 2.

See Section 6.2.2 for the termination criterion for the method.

6.2.4 Block Successive Over-relaxation Method

The approach used for the parallelisation of the BGS method in the previous sub-section can also be used for the parallelisation of the BSOR method. The only difference in the task algorithm will be in step 4 which now reads as follows.

- Modify the right hand side vector, i.e.

$$\tilde{\underline{b}}_i = \omega \underline{b}_i - \{(1-\omega)A_i \underline{x}_i^k + \omega(B_i \underline{x}_{i+1}^k + B_{i-1}^T \underline{x}_{i-1}^{k+1})\}$$

  using banded matrix-vector multiplication.

6.2.5 Block Conjugate Gradient Method

If the sequential preconditioned BCG algorithm given in Section 3.3.3 is examined it can be seen that, during each iteration, the calculation of the parameters α and β require the results from all the blocks to be assembled in a single sum. To implement this algorithm in its current form on the type of transputer network available would cause the same difficulties as those described for the convergence test (see Section 6.2.2). Since the choices of α and β in the sequential algorithm ensure local minimisation of the functional over the space of A-conjugate search directions then any departure from these values will cause the method to lose this property. However, the iterated sequence generated by the BCG method with different choices of α and β may still converge to the solution of the system.

One possible approach is to have separate and for each block. In this case the algorithm becomes

$$
\begin{aligned}
&{}^{0}_{i} = (\quad {}^{-1}\,{}^{0})_i = (\quad {}^{-1}(\quad)) _i && = 1 \ldots \\
&= 0 \\
&\text{while} \quad {}^{k} = 0 \text{ and } \quad {}_{max} \\
&\qquad = \quad + 1 \\
&\qquad {}_{k,i} = \frac{({}^{k}_{i})^T(\quad {}^{-1}\,{}^{k})_i}{({}^{k}_{i})^T(\quad {}^{k})_i} && = 1 \ldots \\
&\qquad {}^{k+1}_{i} = {}^{k}_{i} + \quad {}_{k,i}\,{}^{k}_{i} && = 1 \ldots \\
&\qquad {}^{k+1}_{i} = {}^{k}_{i} \quad {}_{k,i}(\quad {}^{k})_i && = 1 \ldots \\
&\qquad {}_{k,i} = \frac{({}^{k+1}_{i})^T(\quad {}^{-1}\,{}^{k+1})_i}{({}^{k}_{i})^T(\quad {}^{-1}\,{}^{k})_i} && = 1 \ldots \\
&\qquad {}^{k+1}_{i} = (\quad {}^{-1}\,{}^{k+1})_i + \quad {}_{k,i}\,{}^{k}_{i} && = 1 \ldots
\end{aligned}
$$

Note this this is no longer the conjugate gradient method but instead is a related method based on that procedure. This algorithm was found to converge only for the small test cases (e.g. 3 elements in each direction) indicating that the loss of conjugacy in this new method is too great.

$_{k,i}$ $_{k,i}$

$_{k,i}$ $_{k,i}$

$_{k,i}$ $_{k,i}$

opt

$$\left[\quad xx\right]^e = \frac{2}{3}\begin{matrix} xx & \frac{1}{2} & xx \\ \frac{1}{2} & xx & xx \end{matrix} \qquad \left[\quad yy\right]^e = \frac{2}{3}\begin{matrix} yy & \frac{1}{2} & yy \\ \frac{1}{2} & yy & yy \end{matrix}$$

$$\left[\quad zz\right]^e = \frac{1}{2}\begin{matrix} zz & zz \\ zz & zz \end{matrix} \qquad \left[\quad xy\right]^e = \frac{2}{3}\begin{matrix} xy & \frac{1}{2} & xy \\ \frac{1}{2} & xy\ T & xy \end{matrix}$$

$$yz\ e \quad - \begin{matrix} yz & yz \\ yz\ T & yz \end{matrix} \qquad zx\ e \quad - \begin{matrix} zx & zx \\ zx\ T & zx \end{matrix}$$

where

$$ {}^{xx} = \frac{1}{6} \begin{matrix} 2 & 2 & 1 & 1 \\ 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \end{matrix} \qquad {}^{yy} = \frac{1}{6} \begin{matrix} 2 & 1 & 1 & 2 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 \end{matrix} $$

$$ {}^{zz} = \frac{1}{9} \begin{matrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{matrix} \qquad {}^{xy} = \frac{1}{2} \begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix} $$

$$ {}^{yz} = \frac{1}{3} \begin{matrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{matrix} \qquad {}^{\hat{}yz} = \frac{1}{3} \begin{matrix} 0 & 0 & 1 & 2 \\ 0 & 0 & 2 & 1 \\ 1 & 2 & 0 & 0 \\ 2 & 1 & 0 & 0 \end{matrix} $$

$$ {}^{zx} = \frac{1}{3} \begin{matrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 2 & 0 \\ 1 & 0 & 0 & 2 \end{matrix} \qquad {}^{\hat{}zx} = \frac{1}{3} \begin{matrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 \\ 0 & 1 & 2 & 0 \end{matrix} $$

A.2 In uence oefficients for

Element Right Hand Side Vector

$$\{F^x\}^e = \left\{ \begin{array}{c} f^x \\ f^x \end{array} \right\} \qquad \{F^y\}^e = \left\{ \begin{array}{c} f^y \\ f^y \end{array} \right\} \qquad \{F^z\}^e = \left\{ \begin{array}{c} f^z \\ -f^z \end{array} \right\}$$

where

$$f^x = \frac{mH}{2} \left\{ \begin{array}{c} -1 \\ 1 \\ 1 \\ -1 \end{array} \right\} \qquad f^y = \frac{lH}{2} \left\{ \begin{array}{c} -1 \\ -1 \\ 1 \\ 1 \end{array} \right\} \qquad f^z = \frac{lm}{2} \left\{ \begin{array}{c} -1 \\ -1 \\ -1 \\ -1 \end{array} \right\}$$

Appendix B

Small Example Matrices

In this appendix, the global seepage matrices arising from the test problems described in Chapter 4, with $2 \times 2 \times 2$ elements used to discretise the region, are given. (The entries in these matrices are only given to 2 decimal places.)

Matrix A_1 is the result of Problem 1 (Section 4.2.1)

Matrix A_2 is the result of Problem 2 (Section 4.2.1)

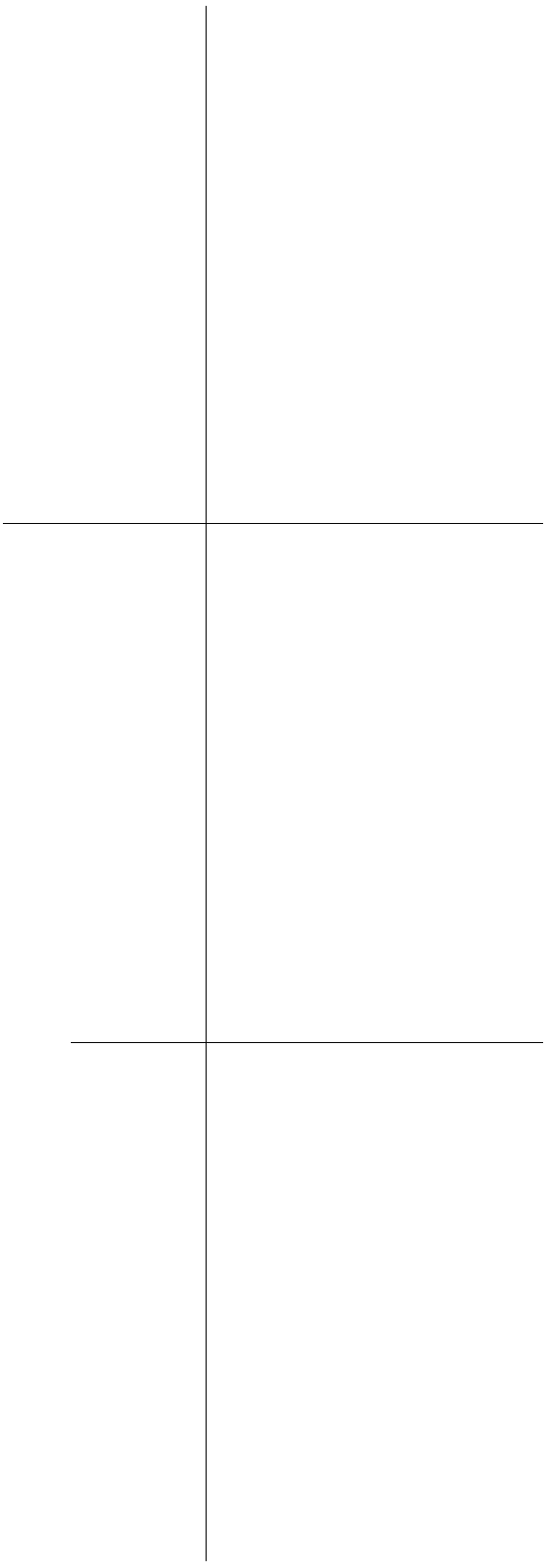
Matrix A_3 is the result of Problem 3 (Section 4.2.2)

Matrix A_4 is the result of Problem 4 (Section 4.2.2)

These matrices are used in Chapter 4 to illustrate the effect on the block diagonal dominance and asymptotic rate of convergence of different node ordering approaches. The matrices are also used to illustrate the structure of the global seepage matrices in Chapter 5.

$$A_1 = \left(\begin{array}{rrrrr|rrrrr|rrrrr}
0.08 & -0.01 & -0.02 & -0.03 & 0.00 & 0.03 & -0.01 & -0.01 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.01 & 0.16 & -0.03 & -0.03 & -0.03 & -0.01 & 0.05 & -0.02 & -0.03 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.02 & -0.03 & 0.16 & -0.02 & 0.00 & -0.01 & -0.02 & 0.05 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.03 & -0.03 & -0.02 & 0.31 & -0.02 & -0.02 & -0.03 & -0.02 & 0.11 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.03 & 0.00 & -0.02 & 0.16 & 0.00 & -0.02 & 0.00 & -0.02 & 0.05 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
\hline
0.03 & -0.01 & -0.01 & -0.02 & 0.00 & 0.16 & -0.02 & -0.03 & -0.07 & 0.00 & 0.03 & -0.01 & -0.01 & -0.02 & 0.00 \\
-0.01 & 0.05 & -0.02 & -0.03 & -0.02 & -0.02 & 0.31 & -0.07 & -0.06 & -0.07 & -0.01 & 0.05 & -0.02 & -0.03 & -0.02 \\
-0.01 & -0.02 & 0.05 & -0.02 & 0.00 & -0.03 & -0.07 & 0.31 & -0.04 & 0.00 & -0.01 & -0.02 & 0.05 & -0.02 & 0.00 \\
-0.02 & -0.03 & -0.02 & 0.11 & -0.02 & -0.07 & -0.06 & -0.04 & 0.63 & -0.04 & -0.02 & -0.03 & -0.02 & 0.11 & -0.02 \\
0.00 & -0.02 & 0.00 & -0.02 & 0.05 & 0.00 & -0.07 & 0.00 & -0.04 & 0.31 & 0.00 & -0.02 & 0.00 & -0.02 & 0.05 \\
\hline
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & -0.01 & -0.01 & -0.02 & 0.00 & 0.08 & -0.01 & -0.02 & -0.03 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.01 & 0.05 & -0.02 & -0.03 & -0.02 & -0.01 & 0.16 & -0.03 & -0.03 & -0.03 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.01 & -0.02 & 0.05 & -0.02 & 0.00 & -0.02 & -0.03 & 0.16 & -0.02 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.02 & -0.03 & -0.02 & 0.11 & -0.02 & -0.03 & -0.03 & -0.02 & 0.31 & -0.02 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.02 & 0.00 & -0.02 & 0.05 & 0.00 & -0.03 & 0.00 & -0.02 & 0.16
\end{array}\right)$$

$$
A_2 = \left[\begin{array}{cccccc|cccccc|ccc}
0.08 & 0.03 & 0.00 & -0.02 & -0.01 & 0.00 & -0.01 & -0.01 & 0.00 & -0.03 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.03 & 0.16 & 0.03 & -0.01 & -0.03 & -0.01 & -0.01 & -0.02 & -0.01 & -0.02 & -0.07 & -0.02 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.03 & 0.08 & 0.00 & -0.01 & -0.02 & 0.00 & -0.01 & -0.01 & 0.00 & -0.02 & -0.03 & 0.00 & 0.00 & 0.00 \\
-0.02 & -0.01 & 0.00 & 0.16 & 0.05 & 0.00 & -0.03 & -0.02 & 0.00 & -0.02 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.01 & -0.03 & -0.01 & 0.05 & 0.31 & 0.05 & -0.02 & -0.07 & -0.02 & -0.02 & -0.04 & -0.02 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.01 & -0.02 & 0.00 & 0.05 & 0.16 & 0.00 & -0.02 & -0.03 & 0.00 & -0.02 & -0.02 & 0.00 & 0.00 & 0.00 \\
\hline
0.01 & 0.01 & 0.00 & 0.03 & 0.02 & 0.00 & 0.16 & 0.05 & 0.00 & 0.03 & 0.03 & 0.00 & 0.03 & 0.02 & 0.00 \\
0.01 & 0.02 & 0.01 & 0.02 & 0.07 & 0.02 & 0.05 & 0.31 & 0.05 & 0.03 & 0.06 & 0.03 & 0.02 & 0.07 & 0.02 \\
0.00 & 0.01 & 0.01 & 0.00 & 0.02 & 0.03 & 0.00 & 0.05 & 0.16 & 0.00 & 0.03 & 0.03 & 0.00 & 0.02 & 0.03 \\
0.03 & 0.02 & 0.00 & 0.02 & 0.02 & 0.00 & 0.03 & 0.03 & 0.00 & 0.31 & 0.11 & 0.00 & 0.02 & 0.02 & 0.00 \\
0.02 & 0.07 & 0.02 & 0.02 & 0.04 & 0.02 & 0.03 & 0.06 & 0.03 & 0.11 & 0.63 & 0.11 & 0.02 & 0.04 & 0.02 \\
0.00 & 0.02 & 0.03 & 0.00 & 0.02 & 0.02 & 0.00 & 0.03 & 0.03 & 0.00 & 0.11 & 0.31 & 0.00 & 0.02 & 0.02 \\
\hline
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & 0.02 & 0.00 & 0.02 & 0.02 & 0.00 & 0.16 & 0.05 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.07 & 0.02 & 0.02 & 0.04 & 0.02 & 0.05 & 0.31 & 0.05 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.03 & 0.00 & 0.02 & 0.02 & 0.00 & 0.05 & 0.16
\end{array}\right]
$$



$$A_4 = \left(\begin{array}{cccccc|cccccc|ccc}
1.10 & -0.13 & 0.00 & 0.25 & -0.21 & 0.00 & -0.13 & -0.40 & 0.00 & -0.21 & -0.28 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.13 & 2.15 & -0.12 & -0.21 & 0.49 & -0.20 & -0.40 & -0.24 & -0.38 & -0.28 & -0.42 & -0.26 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.12 & 1.05 & 0.00 & -0.20 & 0.24 & 0.00 & -0.38 & -0.12 & 0.00 & -0.26 & -0.20 & 0.00 & 0.00 & 0.00 \\
0.25 & -0.21 & 0.00 & 1.65 & -0.19 & 0.00 & -0.21 & -0.28 & 0.00 & -0.19 & -0.60 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.21 & 0.49 & -0.20 & -0.19 & 3.25 & -0.18 & -0.28 & -0.42 & -0.26 & -0.60 & -0.37 & -0.58 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.20 & 0.24 & 0.00 & -0.18 & 1.60 & 0.00 & -0.26 & -0.20 & 0.00 & -0.58 & -0.18 & 0.00 & 0.00 & 0.00 \\
\hline
-0.13 & -0.40 & 0.00 & -0.21 & -0.28 & 0.00 & 1.86 & -0.21 & 0.00 & 0.42 & -0.36 & 0.00 & -0.15 & -0.19 & 0.00 \\
-0.40 & -0.24 & -0.38 & -0.28 & -0.42 & -0.26 & -0.21 & 3.63 & -0.20 & -0.36 & 0.83 & -0.34 & -0.19 & -0.29 & -0.18 \\
0.00 & -0.38 & -0.12 & 0.00 & -0.26 & -0.20 & 0.00 & -0.20 & 1.78 & 0.00 & -0.34 & 0.40 & 0.00 & -0.18 & -0.14 \\
-0.21 & -0.28 & 0.00 & -0.19 & -0.60 & 0.00 & 0.42 & -0.36 & 0.00 & 2.88 & -0.33 & 0.00 & -0.14 & -0.45 & 0.00 \\
-0.28 & -0.42 & -0.26 & -0.60 & -0.37 & -0.58 & -0.36 & 0.83 & -0.34 & -0.33 & 5.71 & -0.32 & -0.45 & -0.28 & -0.45 \\
0.00 & -0.26 & -0.20 & 0.00 & -0.58 & -0.18 & 0.00 & -0.34 & 0.40 & 0.00 & -0.32 & 2.83 & 0.00 & -0.45 & -0.14 \\
\hline
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.15 & -0.19 & 0.00 & -0.14 & -0.45 & 0.00 & 1.23 & -0.14 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.19 & -0.29 & -0.18 & -0.45 & -0.28 & -0.45 & -0.14 & 2.46 & -0.14 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & -0.14 & 0.00 & -0.45 & -0.14 & 0.00 & -0.14 & 1.23
\end{array}\right)$$

Bibliogr phy

[1] D.M. Cooper (Institute of Hydrology), Private Communication, (1991)

[2] A.J. Davies, **The Finite Element Method : A First Approach**, OUP, (1986)

[3] **FLAMINCO Documentation : A Three-Dimensional Finite-Element Code for Analysis Water Flow and Solute Transport in Saturated/Unsaturated Porous Media**, Geo. Trans., Inc., (Sept. 1987)

[4] R.A. Freeze and J.A. Cherry, **Groundwater**, Prentice-Hall, (1979)

[5] G.H. Golub and C.F. Van Loan, **Matrix Computations**, 2nd d., John Hopkins University Press, (1989)

[6] L.A. Hageman and D.M. Young, **Applied Iterative Methods**, Academic Press, (1981)

[7] R.W. Hockney and C.R. Jesshope, **Parallel Computers 2 : Architecture, Programming and Algorithms**, Adam Hilger, Bristol, (1988)

[8] P.S. Huyakorn and B.M. Thompson, **Techniques for Making Finite Elements Competitive in Modelling Flow in Variably Saturated Porous Media**, Water Resources Research. 20, pp. 1099–1115, (Aug. 1984)

[9] P.S. Huyakorn, .P. Springer, V. Guvanasen and T.D. Wadsworth, **A Three-Dimensional Finite Element Model for Simulating Water**

Flow in Variably Saturated Porous Media Flow Systems, Water Resources Research. 22, pp. 1790–1808, (Dec. 1986)

[10] J.J. Modi, **Parallel Algorithms and Matrix Computation**, Clarendon Press, Oxford, (1988)

[11] **Parallel FORTRAN User Guide**, 3L Ltd., Software Version 2.1.3, (Dec. 1990)

[12] G. Strang and G.S. Fix, **An Analysis of the Finite Element Method**, Prentice-Hall Int., (1973)

[13] R.S. Varga, **Matrix Iterative Analysis**, Prentice-Hall Int., London, (1962)