A Multi-Level Domain Decomposition Algorithm
Suitable for the Solution of Three-Dimensional
Elliptic Partial Differential Equations

by

John Killough

Technical Report 86-7, April 1986

\footnotetext{1A Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, Rice University.}
A THREE-DIMENSIONAL DOMAIN DECOMPOSITION ALGORITHM
FOR THE NUMERICAL SOLUTION OF ELLIPTIC AND PARABOLIC
PARTIAL DIFFERENTIAL EQUATIONS

BY

JOHN EDWIN KILLOUGH

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

DOCTOR OF PHILOSOPHY

APPROVED, THESIS COMMITTEE

Mary Fajjett Wheeler,
Professor of Mathematical Sciences
Chairperson

Edward A. Tapia,
Professor of Mathematical Sciences

Clarence A. Miller,
Professor of Chemical Engineering

Roland Glowinski,
Professor of Mathematics
University of Houston

HOUSTON, TEXAS
APRIL, 1986

A THREE-DIMENSIONAL DOMAIN DECOMPOSITION ALGORITHM
FOR THE NUMERICAL SOLUTION OF ELLIPTIC AND PARABOLIC
PARTIAL DIFFERENTIAL EQUATIONS

BY

JOHN EDWIN KILLOUGH

ABSTRACT

A three-dimensional, nonsymmetric, domain decomposition algorithm is developed. The algorithm is based upon the use of a lower dimensional problem as a correction to the preconditioned generalized conjugate residual method using the domain decomposition technique as the overall preconditioner. For the finite difference solution of elliptic and parabolic partial differential equations with symmetric and nonsymmetric rough coefficients, the method is both robust and efficient.

Three problems, including a highly heterogeneous example, an example from the SPE/SIAM comparative solution project, and a nonsymmetric parabolic reservoir simulation example, are presented to validate the method. Several comparisons are made with other well-known preconditioners including incomplete LU and reduced system/incomplete LU
factorizations. For the examples considered the domain decomposition technique was the most efficient in a nonparallel environment; in a parallel computational environment the algorithm was a factor of four faster than the other techniques. An analysis is made concerning both vector and parallel computational aspects of the domain decomposition method. Numerical experiments show the effect of the number of domains, subproblem tolerances, and subproblem preconditioners on the convergence rate and computational work for the algorithm. The convergence rate is shown to be only slightly dependent on the number of subdomains. The effect of subproblem tolerances on the method is also small. Reduced system ILU(0) had the best computational efficiency for use in subproblem solutions. Corrections using the lower dimensional problem, known as line corrections, is shown to be necessary for the rapid convergence of the method. Finally, the three dimensional domain decomposition algorithm was efficiently implemented in parallel computational environments using multitasking and microtasking on both the CRAY X/MP 48 and IBM 3090/400 parallel supercomputers.

ACKNOWLEDGEMENTS

I would like to express my appreciation to my wife and family, who sacrificed much in order for me to pursue my graduate studies. I would also like to express my gratitude to Atlantic Richfield Company for giving me the freedom to complete my studies. I am appreciative of the assistance of Cray Research and the IBM Corporation in performing the parallel experiments included in Section 6 of the thesis.

I especially would like to thank Mary Fanett Wheeler without whose encouragement and advice this thesis would have not been possible. The help and encouragement of John R. Wallis proved to be invaluable in completing the work in this document.
TABLE OF CONTENTS

Section                                      Page

1. INTRODUCTION............................................. 1
2. BACKGROUND............................................... 3
   The Preconditioned Conjugate Gradient Method......... 4
   Substructuring and the Capacitance Matrix Method...... 6
   Capacitance Matrix/PCG Domain Decomposition....... 12
   Full Matrix/PCG Domain Decomposition............... 16
   The Generalized Conjugate Residual Method........... 23
3. THE THREE-DIMENSIONAL FINITE DIFFERENCE
   DOMAIN DECOMPOSITION ALGORITHM...................... 26
   Line Correction Preconditioners.................... 26
   Domain Decomposition with Mixed
   Neumann/Dirichlet boundary Conditions.............. 29
   The Block Gauss-Seidel Algorithm................... 31
   The Three-Dimensional Algorithm................... 32
   Algorithm Variants...................................... 34
4. COMPUTATIONAL ASPECTS OF THE DOMAIN
   DOMAIN DECOMPOSITION ALGORITHM.................... 39
   Vectorization Aspects of the Method................ 40
   Application of Parallel Processing to the Domain
   Decomposition Algorithm............................ 41
5. EXAMPLE PROBLEMS FOR DOMAIN DECOMPOSITION
   COMPUTATIONAL EXPERIMENTS............................ 47
   A Symmetric Highly Heterogeneous Example............ 47
   The SIAM/SPE Comparative Solution Example........... 48
   A Matrix from a Reservoir Simulation Model........... 49
6. COMPUTATIONAL EXPERIMENTS............................. 51
   Comparison of Domain Decomposition
   With Other Preconditioners......................... 52

TABLE OF CONTENTS (CONT.)

Effects of the Number of Domains on
Convergence Rate and Efficiency....................... 59
Effect of Subproblem Tolerances........................ 67
Evaluation of Line Corrections for Domain
Decomposition........................................... 70
Comparisons of Subproblem Preconditioners............ 79
Comparisons of Various Domain Decomposition
Techniques............................................. 82
Comparisons with Block Jacobi Methods............... 95
Comparison of Symmetric and Nonsymmetric
Preconditioning......................................... 103
Neumann and Dirichlet Boundary Conditions for
Edge Subdomains........................................ 109
Application To a Petroleum Reservoir Simulator...... 111
Parallel/Vector Applications of the Domain
Decomposition Algorithm............................. 113

7. CONCLUSIONS............................................. 118

REFERENCES.................................................. 120
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>Example Two-Dimensional Finite Difference Grid With Standard Ordering.</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>Matrix Structure for Five Point Finite Difference Scheme with Standard Ordering.</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Example Two-Dimensional Finite Difference Grid With Capacitance Matrix Method Ordering.</td>
<td>9</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>Matrix Structure for Five Point Finite Difference Scheme with Capacitance Matrix Ordering.</td>
<td>10</td>
</tr>
<tr>
<td>Figure 5.</td>
<td>Example Two-Dimensional Grid for the Bramble et al. Method Showing Standard Ordering.</td>
<td>17</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>Schematic of Standard Ordering for Five Point Finite Difference Scheme.</td>
<td>18</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>Example Two-Dimensional Grid for the Bramble et al. Method Showing Domain Decomposition Ordering.</td>
<td>19</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>Schematic of Five Point Finite Difference Matrix With Bramble et al. Ordering.</td>
<td>21</td>
</tr>
<tr>
<td>Figure 9.</td>
<td>Comparison of Relative Work for the Vertex, CG, and Jacobi/CG Methods for the SIAM/SPE Example</td>
<td>54</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>Comparison of Relative Work for the Vertex, CG, and Jacobi/CG Methods for the Heterogeneous Example</td>
<td>55</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>Comparison of Relative Work for the Vertex Method and Other Preconditioned GCR Methods for the SIAM/SPE Example</td>
<td>57</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>Comparison of Relative Work for the Vertex Method and Other Preconditioned GCR Methods for the Heterogeneous Example</td>
<td>58</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>Effect of the Number of Domains on the Convergence Rate for the Vertex Method for the SIAM/SPE Example.</td>
<td>60</td>
</tr>
<tr>
<td>Figure 14.</td>
<td>Comparison of Vector Work for Various Numbers of Domains for the Vertex Method for the SIAM/SPE Example.</td>
<td>61</td>
</tr>
<tr>
<td>Figure 15.</td>
<td>Comparison of Scalar Work for Various Numbers of Domains for the Vertex Method for the SIAM/SPE Example.</td>
<td>63</td>
</tr>
<tr>
<td>Figure 16.</td>
<td>Effect of the Number of Domains on the Convergence Rate for the Vertex Method for the Heterogeneous Example.</td>
<td>64</td>
</tr>
<tr>
<td>Figure 17.</td>
<td>Comparison of Vector Work for Various Numbers of Domains for the Vertex Method for the Heterogeneous Example.</td>
<td>65</td>
</tr>
<tr>
<td>Figure 18.</td>
<td>Comparison of Scalar Work for Various Numbers of Domains for the Vertex Method for the Heterogeneous Example.</td>
<td>66</td>
</tr>
<tr>
<td>Figure 19.</td>
<td>Effect of Subproblem Tolerance on Convergence Rate for the SIAM/SPE Example.</td>
<td>68</td>
</tr>
<tr>
<td>Figure 20.</td>
<td>Effect of Subproblem Tolerance on Relative Work for the SIAM/SPE Example.</td>
<td>69</td>
</tr>
<tr>
<td>Figure 21.</td>
<td>Effect of Subproblem Tolerance on Convergence Rate for the Heterogeneous Example.</td>
<td>71</td>
</tr>
<tr>
<td>Figure 22.</td>
<td>Effect of Subproblem Tolerance on Relative Work for the Heterogeneous Example.</td>
<td>72</td>
</tr>
<tr>
<td>Figure 23.</td>
<td>Effect of Optimal Line Corrections Versus No Line Corrections for the SIAM/SPE Example.</td>
<td>73</td>
</tr>
<tr>
<td>Figure 24.</td>
<td>Effect of X-, Y-, or Z-line Corrections on Convergence Rate for the SIAM/SPE Example.</td>
<td>74</td>
</tr>
<tr>
<td>Figure 25.</td>
<td>Effect of Combination Line Corrections on the Convergence Rate for the SIAM/SPE Example.</td>
<td>75</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>26</td>
<td>Effect of Optimal Line Corrections Versus No Line Corrections for the Heterogeneous Example</td>
<td>77</td>
</tr>
<tr>
<td>27</td>
<td>Effect of X-, Y-, or Z-line Corrections on Convergence Rate for the Heterogeneous Example</td>
<td>78</td>
</tr>
<tr>
<td>28</td>
<td>Effect of Combination Line Corrections on the Convergence Rate for the Heterogeneous Example</td>
<td>80</td>
</tr>
<tr>
<td>29</td>
<td>Comparison of Convergence Rates for Various Subproblem Preconditioners for the SIAM/SPE Example</td>
<td>81</td>
</tr>
<tr>
<td>30</td>
<td>Comparison of Relative Work for Various Subproblem Preconditioners for the SIAM/SPE Example</td>
<td>83</td>
</tr>
<tr>
<td>31</td>
<td>Comparison of Convergence Rates for Various Subproblem Preconditioners for the Heterogeneous Example</td>
<td>84</td>
</tr>
<tr>
<td>32</td>
<td>Comparison of Relative Work for Various Subproblem Preconditioners for the Heterogeneous Example</td>
<td>85</td>
</tr>
<tr>
<td>33</td>
<td>Comparison of Convergence Rates for Block and Vertex Methods for the SIAM/SPE Example</td>
<td>87</td>
</tr>
<tr>
<td>34</td>
<td>Comparison of Relative Work for Block and Vertex Methods for the SIAM/SPE Example</td>
<td>89</td>
</tr>
<tr>
<td>35</td>
<td>Comparison of Convergence Rates for Block and Vertex Methods for the Heterogeneous Example</td>
<td>90</td>
</tr>
<tr>
<td>36</td>
<td>Comparison of Relative Work for Block and Vertex Methods for the Heterogeneous Example</td>
<td>91</td>
</tr>
<tr>
<td>37</td>
<td>Comparison of Convergence Rates for Block and Vertex Strips for the SIAM/SPE Example</td>
<td>93</td>
</tr>
<tr>
<td>38</td>
<td>Comparison of Relative Work for Block and Vertex Strips for the SIAM/SPE Example</td>
<td>94</td>
</tr>
<tr>
<td>39</td>
<td>Comparison of Convergence Rates of Vertex Strips for the Heterogeneous Example</td>
<td>96</td>
</tr>
<tr>
<td>40</td>
<td>Comparison of Relative Work of Vertex Strips for the Heterogeneous Example</td>
<td>97</td>
</tr>
<tr>
<td>41</td>
<td>Comparison of Convergence Rates for Block Strips for the Heterogeneous Example</td>
<td>98</td>
</tr>
<tr>
<td>42</td>
<td>Comparison of Relative Work for Block Strips for the Heterogeneous Example</td>
<td>99</td>
</tr>
<tr>
<td>43</td>
<td>Comparison of Convergence Rates for the Simultaneous and Sequential Block Methods for the SIAM/SPE Example</td>
<td>101</td>
</tr>
<tr>
<td>44</td>
<td>Comparison of Relative Work for the Simultaneous and Sequential Block Methods for the SIAM/SPE Example</td>
<td>102</td>
</tr>
<tr>
<td>45</td>
<td>Comparison of Convergence Rates for the Vertex and Simultaneous/Sequential Block Methods for the Heterogeneous Example</td>
<td>104</td>
</tr>
<tr>
<td>46</td>
<td>Comparison of Relative Work for the Vertex and Simultaneous/Sequential Block Methods for the Heterogeneous Example</td>
<td>105</td>
</tr>
<tr>
<td>47</td>
<td>Comparison of Convergence Rates for Symmetric and Nonymmetric CG/GCR Vertex Method for the Heterogeneous Example</td>
<td>107</td>
</tr>
<tr>
<td>48</td>
<td>Comparison of Relative Work for Symmetric and Nonymmetric CG/GCR Vertex Method for the Heterogeneous Example</td>
<td>108</td>
</tr>
<tr>
<td>49</td>
<td>Effect of Edge Neumann and Dirichlet Boundary Conditions on the Convergence Rate of the Vertex Method</td>
<td>110</td>
</tr>
</tbody>
</table>
Section 1
INTRODUCTION

The following sections present the development and validation of a three-dimensional, nonsymmetric, preconditioned generalized conjugate residual algorithm which uses the domain decomposition technique for preconditioning. Section 2 presents the background of the domain decomposition methods which have been published to date. These methods fall into two categories: capacitance matrix methods and full matrix methods. Also presented in Section 1 are the preconditioned conjugate gradient and generalized conjugate residual iterative accelerations. Section 3 develops the three-dimensional algorithm. The method of line corrections is presented along with the nonsymmetric domain decomposition technique. Several algorithm variants are also presented.

Section 4 discusses the computational aspects of the algorithm for both vector and parallel computer architectures. Since one of the main advantages of the domain decomposition approach is its applicability to parallel computing, emphasis is placed on implementation of the algorithm in a parallel environment. In the next section the example problems for the numerical experiments are presented: a highly heterogeneous example, an example from the SPE/SIAM comparative solution project, and an example from a production petroleum reservoir simulation model. The computational experiments given in Section 6 demonstrate the robustness and efficiency of the three-dimensional domain decomposition technique. Parameter studies show the most effective way in which to implement the numerous algorithm variants. The implementation of the algorithm in the production reservoir simulation model is presented along with an approach for combining the method with Newton-Raphson iterations for the solution of nonlinear equations. Finally, computational results are given in Section 6 for the implementation of the algorithm on two vector/parallel supercomputers: the CRAY X-MP/48 and the IBM 3090/400.
Section 2

BACKGROUND

Solution of the following elliptic partial differential equation:

\[- \nabla \cdot (a \nabla u) = f \text{ on } \Omega \quad (1)\]

\[u = g \text{ on } \partial \Omega\]

is often performed numerically using either finite differences or finite elements. Either of these numerical solutions leads to a system of linear equations with an associated positive definite matrix $A$. The following closely related system of partial differential equations from petroleum reservoir simulation can yield a nonsymmetric matrix:

\[- v \cdot C_0 \nabla P_0 + q_0 = \frac{\partial}{\partial t} (S_0 B_0) \]

\[- v \cdot C_w \nabla (P_0 - P_{cow}) + q_w = \frac{\partial}{\partial t} (S_w B_w) \quad (2)\]

\[- v \cdot C_g \nabla (P_0 + P_{cgo}) - v \cdot R_s C_0 \nabla P_0 + q_g + R_s q_0 = \frac{\partial}{\partial t} (S_g B_g + R_s S_0 B_0) \]

on $\Omega$,

\[\frac{\partial P_0}{\partial n} = 0 \quad \text{on } \partial \Omega,\]

where,

\[P_0 = \text{oil phase pressure},\]

\[C_0, C_w, C_g = \text{phase transmissibilities},\]

\[q_0, q_w, q_g = \text{phase source and sink terms},\]

\[S_0, S_w, S_g = \text{phase saturations},\]

\[B_0, B_w, B_g = \text{phase density terms},\]

\[P_{cow}, P_{cgo} = \text{capillary pressure terms},\]

\[R_s = \text{gas-oil mass transfer term}.\]

The system of equations is often reduced to a single finite difference equation involving the unknown $P_0$, the oil phase pressure, using the IMPES reduction scheme. This single equation varies from either elliptic to parabolic in nature depending on the compressibility of the fluids in the reservoir.

For either equations (1) or (2), the resultant matrix for the finite difference or finite element numerical solution is large, banded, and sparse. Solution of this large, sparse system often becomes the most difficult and computationally intense part of obtaining a numerical solution to the resulting elliptic or parabolic partial differential equations.

The Preconditioned Conjugate Gradient Method

Recently, preconditioned conjugate gradient or generalized conjugate residual methods have become popular techniques for the solution of these systems of equations. (See for example, references 2-5.) Given the following
system of equations to solve:
\[ A \mathbf{x} = \mathbf{b} \]  
(3)

the preconditioned conjugate gradient method can be described as follows:

For \( k = 0, 1, 2, \ldots \) until convergence:

1. Solve \( M \mathbf{z}^k = \mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k \)
   where \( M \) = preconditioning matrix

2. \( \beta_k = \langle M \mathbf{z}^k, \mathbf{z}^k \rangle / \langle M \mathbf{z}^{k-1}, \mathbf{z}^{k-1} \rangle \),
   with \( \beta_0 = 0 \)

3. \( \mathbf{p}^k = \mathbf{z}^k + \beta_k \mathbf{p}^{k-1} \)

4. \( \alpha_k = \langle M \mathbf{z}^k, \mathbf{z}^k \rangle / \langle A \mathbf{p}^k, \mathbf{p}^k \rangle \)

5. \( \mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k \)

6. \( \mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k A \mathbf{p}^k \)

For this method to be efficient, several conditions are required. First, the preconditioner \( M \) as well as the matrix \( A \) should be symmetric, positive definite. The following system of equations should be easy to solve:

\[ M \mathbf{x} = \mathbf{b} \]  
(5)

since solution of this system requires a substantial portion of the work associated with the preconditioned conjugate gradient method (PCG). Conversely, if the condition number \( \kappa \) of the system \( M^{-1}A \) is sufficiently small, insuring only a few iterations for PCG to converge, then the preconditioned system (5) may be somewhat more complicated.

The substance of this thesis deals with a complex preconditioner which involves several steps to solve the linear system of equations associated with (3).

Substructuring and the Capacitance Matrix Methods

Capacitance matrix methods or substructuring have been used for some time in the finite element and finite difference literature for the solution of large systems associated with structural and circuit analysis. The main benefit derived from these techniques was the reduction in computational storage when smaller systems are solved. The basic idea can be described as follows. For a finite difference grid with orderings as shown in Figure 1, the resultant matrix would appear as depicted in Figure 2.

However, if the orderings are performed as shown in Figure 3 with the subdomain \( \Omega_1 \) ordered first, followed by \( \Omega_2 \) and finally the boundary \( \Gamma_3 \) between the two domains, a different structure of the matrix results as shown in Figure 4. The capacitance matrix can then be formed by a block Gaussian elimination procedure with the eliminations for the two subdomains \( \Omega_1 \) and \( \Omega_2 \) corresponding to the solutions of smaller two-dimensional problems. The original matrix corresponding to Figure 4 can be described as follows:

\[
\begin{bmatrix}
K_{11} & K_{13} \\
K_{22} & K_{23} \\
K_{T} & K_{T}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
=
\begin{bmatrix}
b_1 \\
b_2 \\
b_3
\end{bmatrix}
\]  
(3)
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure 1. Example Two-Dimensional Finite Difference Grid With Standard Ordering

\[
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \\
\times & \times & \times & \\
\times & \times & \\
\times & \\
\end{bmatrix}
\]

Figure 2. Matrix Structure for Five Point Finite Difference Scheme with Standard Ordering
Figure 3. Example Two-Dimensional Finite Difference Grid With Capacitance Matrix Method Ordering

Figure 4. Matrix Structure for Five Point Finite Difference Scheme with Capacitance Matrix Ordering
The next step in the capacitance matrix method is to form the Schur complement or capacitance matrix \( \bar{K}_{33} \) by Gaussian elimination as follows:

\[
\begin{bmatrix}
K_{11} & K_{13} \\
K_{22} & K_{23}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2
\end{bmatrix}
\]  \( (4) \)

where,

\[
\bar{K}_{33} = K_{33} - K_{13} K_{11}^{-1} K_{13} - K_{23} K_{22}^{-1} K_{23}
\]

\[
\bar{b}_3 = b_3 - K_{13} K_{11}^{-1} b_1 - K_{23} K_{22}^{-1} b_2
\]  \( (5) \)

The capacitance matrix system is given by the following equations:

\[
\bar{K}_{33} x_3 = \bar{b}_3
\]  \( (6) \)

The second and third terms of the right hand side of equation (5) correspond to Dirichlet boundary conditions with the solutions on the subdomains being applied to the boundary domain \( \Gamma_3 \) through the coupling terms \( K_{13} \) and \( K_{23} \).

Although the capacitance matrix or Schur complement \( \bar{K}_{33} \) belongs to a subspace of the original problem, its structure is generally substantially fuller than the original matrix \( A \). In the past the capacitance matrix methods solved this dense system using direct methods. Once the solution of the subsystem (6) is known, the entire system can be solved easily by back substitution providing the inverses \( K_{11}^{-1} \) and \( K_{22}^{-1} \) (or equivalent) are still available.

**Capacitance Matrix/PCG Domain Decomposition**

Several authors have recently developed techniques in which a variant of the capacitance matrix method becomes an iterative solution for the capacitance system using conjugate gradients.\(^{11-13, 15} \) The basic idea of the technique is to guess an initial solution on the boundary \( \Gamma_3 \) and then to iterate to the solution of the capacitance system (6) using conjugate gradients. The following steps outline the algorithm for a capacitance matrix approach using conjugate gradients:

1. **Initialization**
   
   - Set \( x_3 = \lambda_0 \) \( (\lambda_0 = 0 \) for most schemes \)
   - Solve \( K_{ii} w_i = b_i - K_{13} \lambda_0 \) for \( i=1,2 \) \( (7) \)
   - Form the residue of the capacitance system:
     
     \[ r^0 = b_3 - K_{13} K_{11}^{-1} (b_1 - K_{13} \lambda_0) - K_{23} K_{22}^{-1} (b_2 - K_{23} \lambda_0) \]  \( (8) \)
   - Precondition:
     
     \[ M z^0 = r^0 \]
     
     \[ z^0 = M^{-1} r^0 \]

2. **Iteration**

   For \( k=0,1,2,\ldots \) until convergence
   
   - Form \( \bar{K}_{33} p^k \) \((p^0 = z^0)\)
   - By solving:
     
     \[ K_{ii} w_i^k = K_{13} p^k \]  \( \) for \( i=1,2 \)
   - And forming the following sum:
     
     \[ \bar{K}_{33} p^k = K_{33} p^k - K_{13} w_1^k - K_{23} w_2^k \]  \( (9) \)
Set,
\[
\begin{align*}
  a_k &= \frac{\langle z^k, r^k \rangle}{\langle p^k, \tilde{R}_3^k p^k \rangle}, \\
  \lambda_k^{k+1} &= \lambda_k^k - \langle \tilde{R}_3^k p^k, \lambda_k^k \rangle, \\
  r_k^{k+1} &= r_k^k - a_k \tilde{R}_3^k p^k, \\
  \beta_k &= \frac{\langle r_k^{k+1}, z^k \rangle}{\langle z^k, z^k \rangle}, \\
  z_k^{k+1} &= \lambda_k^{k+1} r_k^{k+1}, \quad \text{and} \quad p_k^{k+1} = z_k^{k+1} + \beta_k p^k.
\end{align*}
\]

In general, each iteration of the capacitance matrix/PCG algorithm requires one solution of the smaller subproblems on domains \(\Omega_1\) and \(\Omega_2\) to form the product of the capacitance matrix \(\tilde{R}_3^k\) and the vector \(p^k\). In addition, the initialization step requires a solution of the Dirichlet problem on each of the subdomains.

The main differences among the reported methods in the literature lie in the treatment of the preconditioner \(M\) for the conjugate gradient method. Glowinski et al.\textsuperscript{11} and Fischler\textsuperscript{16}, simply choose the original matrix \(R_3\) as the preconditioning matrix. Since this matrix is sparse and, in the case of the example, a tridiagonal matrix corresponding to a one-dimensional finite difference grid, it is easy to solve the preconditioning step for \(z^k\). The use of this preconditioner is intuitively appealing since no assumption as to the smoothness of the coefficients of the original differential equation is made.

If one appeals to the theory of differential equations, a better preconditioner can be developed. Since the capacitance matrix represents a solution on a subspace of the original problem on the boundary between the two subdomains for the finite difference example, the relevant Sobolev subspace is \(H^1(\Gamma_3)\) with corresponding norm defined as follows:

\[
\|w\|_{H^1(\Omega)} = \left( \sum_{k=1}^{P} \hat{a}_k^2 \lambda_k^k \right)^{\frac{1}{2}}
\]  

where \(\hat{a}_k\) are the Fourier coefficients of the function \(w\) with respect to the eigenvectors of the differential operator defined on the boundary \(\Gamma_3\). The \(\lambda_k\) are the corresponding eigenvalues for the operator. The value \(p\) is the dimension of the subsystem on the boundary. A convenient operator to choose on the boundary is the finite difference operator \(K\) which corresponds to the following tridiagonal system for a case in which the coefficients are constant:

\[
K = \begin{bmatrix}
-2 & 1 \\
1 & -2 & 1 \\
& & & & & & & & & & \\
& & & & & & & & & & \\
& & & & & & & & & & \\
& & & & & & & & & & \\
& & & & & & & & & & \\
& & & & & & & & & & \\
& & & & & & & & & & \\
& & & & & & & & & & \\
\end{bmatrix}
\]

The method developed by Dryja\textsuperscript{13,17} corresponds to using \(K\) as the preconditioner \(M\) for the conjugate gradient method.
applied to the capacitance matrix system of equations. Golub et al.\textsuperscript{15} developed an additional preconditioner based on the observation that the capacitance matrix $\tilde{K}_{33}$ is close to being a Toeplitz form. This observation led to a modification of Dryja's preconditioner as follows:

$$M = (4K + K^2)^{\frac{1}{2}}$$  \hspace{1cm} (12)

where $K$ is as defined by equation (11). Bjorstad and Widlund\textsuperscript{12} developed a preconditioner for the system (6) by splitting the matrix $K_{33}$ into two pieces corresponding to contributions from each of the subdomains $\Omega_1$ and $\Omega_2$ as follows:

$$K_{33} = K_{33}^{(1)} + K_{33}^{(2)}$$  \hspace{1cm} (13)

The preconditioner then consists of two steps. First, the following system is solved for $\omega^k$:

$$\begin{pmatrix} K_{33}^{(1)} & K_{13} \end{pmatrix} \begin{pmatrix} K_{11}^{-1} & K_{13} \end{pmatrix} \omega^k = r^k$$  \hspace{1cm} (14)

The resultant solution vector is then used as the right hand side for the following equation:

$$\begin{pmatrix} K_{33}^{(2)} & K_{23} \end{pmatrix} K_{22}^{-1} K_{23} \omega^k = z^k$$  \hspace{1cm} (15)

The resultant $z^k$ is simply the preconditioned solution to which conjugate gradient acceleration is applied. The solution of the system (14) requires less work than the solution of the capacitance matrix and in fact corresponds to the solution of the two-dimensional system formed by $\Omega_1$ and $\Gamma_3$ with coupling to $\Omega_2$ through Dirichlet boundary conditions. In their work Bjorstad and Widlund\textsuperscript{12} showed that theoretically the preconditioner given by (14) and (15) results in the smallest condition number for $M^{-1}A$. They also showed that the Golub preconditioner is nearly the same as (14) and (15) and that the Dryja preconditioner is slightly worse. The Glowinski method is shown to have a condition number that depends on the number of nodes associated with the boundary $\Gamma_3$.

Full Matrix/PCG Domain Decomposition

The capacitance matrix methods depend on a near exact solution on each of the subdomains to form the product of the matrix $\tilde{K}_{33}$ and the search vector $p^k$. The expense of this solution for nonuniform coefficient cases led to the methods of Bramble, Pasciak, and Schatz\textsuperscript{14,18,19} in which an approximate solution is obtained on each of the subdomains using an efficient technique such as FFT. Since exact solutions are not obtained, the preconditioned conjugate gradient iteration is applied to the entire matrix $A$ rather than to the subsystem associated with $\tilde{K}_{33}$. The basic idea of the method can be better understood by again considering a finite difference example for the homogenous Neumann problem on a rectangle. Figure 5 depicts the associated finite difference grid with standard ordering. The associated matrix for this ordering for five point finite differences is shown in Figure 6. Figure 7 shows the
Figure 5. Example Two-Dimensional Grid for the Bramble et al. Method Showing Standard Ordering

Figure 6. Schematic of Standard Ordering for Five Point Finite Difference Scheme
ordering for the Bramble et al. method and Figure 8 gives a schematic of the associated matrix. Using the notation adopted for the previous example, the following matrix equation results from the reordered coefficients:

\[
\begin{bmatrix}
K_{11} & K_{15} & x_1 & b_1 & 0_1 \\
K_{22} & K_{25} & x_2 & b_2 & 0_2 \\
& & K_{33} & x_3 & b_3 & 0_3 \\
& & K_{44} & K_{45} & x_4 & b_4 & 0_4 \\
K_{15} & K_{25} & K_{35} & K_{45} & K_{55} & x_5 & b_5 & \Gamma
\end{bmatrix}
\]  

(16)

The boundary matrix \(K_{55}\) can be further subdivided into matrices involving the edges and the vertex in the middle of the edges:

\[
\begin{bmatrix}
K_{1E}^{1E} & K_{15}^{1E} & x_{1E} & b_{1E} & \Gamma_{1E} \\
K_{2E}^{2E} & K_{25}^{2E} & x_{2E} & b_{2E} & \Gamma_{2E} \\
K_{3E}^{3E} & K_{35}^{3E} & x_{3E} & b_{3E} & \Gamma_{3E} \\
K_{4E}^{4E} & K_{45}^{4E} & x_{4E} & b_{4E} & \Gamma_{4E} \\
K_{15}^{E} & K_{25}^{E} & K_{35}^{E} & K_{45}^{E} & K_{55}^{E} & x_v & b_v & \Gamma_v
\end{bmatrix}
\]  

(17)

If Gaussian elimination were now used on (16), the method would simply be a Schur complement method; however, the Bramble et al. variant is to form approximations to the
solutions of the systems on the subdomains using FFT methods and then to use these approximations as part of the preconditioner for the full matrix problem. The steps in the method can be qualitatively described as follows:

1) Reorder the system with large subdomains first, the edges between the subdomains next, and the vertices between the domains last.

2) On each of the subdomains solve for $W_p$, the approximate FFT solution to the Dirichlet problem with homogeneous boundary conditions.

3) Given the solutions $W_p$ on the large subdomains, find the solutions on the edges between domains $(\Gamma_{ij})$ by solving the one-dimensional problems on each edge. This corresponds to solving a homogeneous Dirichlet problem on the edges.

4) Find a solution for the vertex (vertices) $W_v$ by solving a coarse mesh finite difference problem over the entire grid. This step can be done independently of the edge solutions.

5) Extend the solutions on the edges and vertices by solving the associated Dirichlet problems on the 2-D subdomains for $W_H$, the discrete harmonic on the subdomains.

6) Set the overall solution on the subdomains equal to
\( W_H + W_P \), the sum of the harmonic solution and the particular solution.

7) Use the solutions from steps 4 and 6 above as the preconditioned solution \( z^k \) for the PCG algorithm for the full matrix \( A \).

Using the notation of Bramble et al., the preconditioner \( M \) for the algorithm can be expressed in matrix form as follows:

\[
\begin{pmatrix}
B_{PP} & 0 & 0 \\
B_{EP} & I & \tilde{V}
\end{pmatrix}
\begin{pmatrix}
B_{PP} & B_{EP} & B_{VP} \\
B_{TP} & B_{EE} & 0 \\
B_{VP} & 0 & \tilde{V}
\end{pmatrix}
= \begin{pmatrix}
B_{PP}^2 & B_{PP}B_{EP} & B_{PP}B_{VP} \\
B_{TP}B_{EP} & B_{EE} & 0 \\
B_{VP}B_{VV} & 0 & \tilde{V}
\end{pmatrix}
\tag{18}
\]

where the subscripts \( P, E, \) and \( V \) refer to the two-dimensional subdomains, the edges between subdomain, and the vertices, respectively. As shown in equation 18, the preconditioner \( M \) is symmetric as required by the PCG method. In fact, this algorithm can be thought of as a variation on a symmetric form of the block Gauss-Seidel method for preconditioning.

The matrices \( \tilde{B}_{EE} \) and \( \tilde{B}_{VV} \) represent the modifications to the original matrices such that the edge solution involves a solution using the \( H^2 \) norm which requires the eigenvalues and eigenvectors of the edge system. The vertex matrix is modified to represent the coarse grid finite difference solution discussed above.

The Generalized Conjugate Residual Method

Up to this point in the discussion the matrix \( A \) has been assumed to be symmetric, positive definite. However, in many instances \( A \) is not symmetric. (For example, the matrix arising from equations (2).) For the solution of these nonsymmetric systems a variant of the preconditioned generalized conjugate residual algorithm known as ORTHOMIN(\( k \)) has become perhaps one of the most widely used methods. This method as originally developed by Vinsome\(^{21}\) can be described as follows:

For \( n = 0,1,2, \ldots \) until convergence:

1. \( l = \text{mod}(n+1, k) \)
2. \( m = \text{min}(n, k) \)
3. Solve \( Mz^n = r^n \)

   where \( M = \) preconditioning matrix

4. Form \( u = A z^n \)
5. For \( i < l \) or \( l < i \leq m \)

   Form \( a_i = z_i < p^i, u > \)

6. \( q^l = v - \sum_{i=0}^{l-1} a_i p^i - \sum_{i=l+1}^{m} a_i p^i \)
7. \( r^l = l < p^l, p^l > \)
8. \( \omega = \gamma^l < r^n, p^l > \)
9. \( x^{n+1} = x^n + \omega q^l \)
10. \( r^{n+1} = r^n - \omega p^l \)

The value of \( k \) is generally limited to a maximum of ten which has been shown empirically to be sufficient for reasonable convergence of the acceleration technique.
Larger values of $k$ could of course be used; however, computer memory usually limits the size used since $k$ vectors must be saved. The cost of the limitation on $k$ is that convergence of the nonpreconditioned method cannot be guaranteed. In practice this is not a concern, since far fewer than $n$ iterations are performed where $n$ corresponds to the size of the matrix $A$. A small choice of $k$ may sometimes reduce the rate of convergence or lead to oscillations, however.

The next section presents a nonsymmetric domain decomposition preconditioner which depends on the use of ORTHOMIN($k$) generalized conjugate residual acceleration.

Section 3

THE THREE-DIMENSIONAL FINITE DIFFERENCE DOMAIN DECOMPOSITION ALGORITHM

Development of the three-dimensional domain decomposition algorithm presented in this thesis was based on three separate observations. First, the domain decomposition methods discussed in Section 2 appeared to have several characteristics which would be useful for parallel computer applications of the algorithm. Second, an acceleration method which had been used in the petroleum industry for the solution of matrices with nonstandard couplings between finite difference cells appeared to have promise as an accelerator for the domain decomposition methods. Finally, incomplete LU factorizations, which had gained prominence in the petroleum reservoir simulation literature, could be used as approximate solutions for the subdomains instead of exact solutions or approximate FFT solutions. The following section describes in more detail the manner in which these pieces of the algorithm were joined to form the three-dimension method presented as the subject of this thesis.

Line Correction Preconditioners

The concept of line corrections was first introduced by
Watts in 1971. The basic idea of the method is to sum the residuals in a given direction and to solve a reduced problem such that the solution forces the sum of the residuals in the given direction to be zero. A somewhat different but equivalent approach is to assume that the solution has no variation in a given direction of the grid. This situation might arise if coefficients in a given direction were extremely large, but the method is not restricted to such cases. If the solution is indeed constant, for example, in the z-direction, then for the following elliptic partial differential equation:

\[ \nabla \cdot a(x,y,z) \nabla u = f(x,y,z) \quad (19) \]

the following equation is true:

\[ u(x,y,z) = u(x,y) \quad (20) \]

Because of this, the differential equation (20) can be integrally averaged in the z-direction to produce a system of reduced dimension:

\[
\frac{1}{z_{\text{max}}} \int_0^{z_{\text{max}}} \nabla \cdot a \nabla u \, dz = \frac{1}{z_{\text{max}}} \left[ \frac{\partial}{\partial x} \left( \int_0^{z_{\text{max}}} a \, dz \right) \frac{\partial}{\partial x} u + \frac{\partial}{\partial y} \left( \int_0^{z_{\text{max}}} a \, dz \right) \frac{\partial}{\partial y} u \right] = \int_0^{z_{\text{max}}} f \, dz 
\]

(21)

The resultant two-dimensional equation when expressed in finite difference form is obviously substantially faster to solve than the original three-dimensional system since the associated matrix has a substantially reduced bandwidth and number of equations. When expressed in matrix form equation (21) is equivalent to the following:

\[
(C^T A C) x = C^T b
\]

(22)

The size of the matrix \(C\) is \(N_{xyz}\) rows by \(N_{xy}\) columns where \(N_{xyz}\) and \(N_{xy}\) are the number of grid points in the two-dimensional areal reduced problem and \(N_{xyz}\) is the number of unknowns in the original set of three-dimensional finite difference equations, respectively. The dimension of the vector \(x\) in equation (22) is simply that of the reduced two-dimensional areal system, \(N_{xy}\). If \(I_{N_{xy}}\) is a representation of the identity matrix of dimension \(N_{xy}\) then the matrix \(C\) is given by the following equation:

\[
C = \begin{bmatrix}
I_{N_{xy}} & \\
I_{N_{xy}} & N_x \text{ times} \\
& N_x \\
I_{N_{xy}}
\end{bmatrix}
\]

(23)

The two-dimensional solution vector \(x\) is expanded to the
entire grid by applying the matrix operator C to x. For line corrections in the z-direction this corresponds to adding the same correction to each of the grid points in a given column of the three-dimensional grid.

For numerous examples it has been shown empirically that the use of line corrections generally substantially accelerates the convergence of an iterative procedure for solution of the linear system A for the finite difference representation of the differential equation (19). Combined with the preconditioned generalized conjugate residual method, the use of line corrections has proven to be quite robust. The theory of this technique originally developed by Watts\(^{22}\) indicates that the line corrections tend to eliminate certain low frequency eigenvalues associated with eigenvectors aligned in the direction of the correction.

A natural use of line corrections comes about in the application of the technique to domain decomposition techniques such as that developed by Bramble et al. Instead of applying somewhat arbitrary connections between the vertices, the line correction technique provides a more rigorous and perhaps simpler method to join the vertices and the solutions in all of the subdomains.

**Domain Decomposition with Mixed Neumann/Dirichlet Boundary Conditions**

The domain decomposition technique which will be used in this thesis is based on solution of the block centered finite difference approximation to a partial differential equation of the following form with homogeneous Neumann-type boundary condition:

\[
- \nabla \cdot a \nabla u = f(u) + \frac{\partial u}{\partial t} \quad \text{on } \Omega \times (0,T) \\
\frac{\partial u}{\partial n} = 0 \quad \text{on } \partial \Omega \times (0,T) \\
u(\cdot,0) = 0 \quad \text{on } \Omega
\]  

(24)

As noted above, other applications of domain decomposition techniques have looked at the Dirichlet boundary conditions; however, most of the conclusions which will be drawn are equally applicable to this closely related problem with Neumann boundary conditions. For the solution using domain decomposition as a preconditioner the resulting matrix will have mixed boundary conditions for the differential equation: Neumann-type on the original boundary of the domain and Dirichlet-type on the boundaries between the separate subdomains. The use of Dirichlet boundary conditions on the subdomains leads to matrices for the subproblems which have nice properties, i.e., they are more diagonally dominant.

Solution of the three-dimensional partial differential equation (24) results in subdomains which are of dimension
one greater than the techniques discussed in Section 2. For the Bramble et al. technique, the large subproblems become three-dimensional and the edges becomes two-dimensional cross-sections.

The Block Gauss-Seidel Algorithm

The use of a nonsymmetric acceleration scheme such as ORTHOMIN(k) allows the preconditioner as well as the original matrix A to be nonsymmetric. For the Bramble et al. method the preconditioner M can be written as follows:

\[ M = \begin{bmatrix}
  B_{pp} & 0 & 0 \\
  B_{EP} & B_{EE} & 0 \\
  B_{VP} & B_{EV} & B_{VV}
\end{bmatrix} \quad (25) \]

After the solutions on the large subdomains represented by \( B_{pp} \), the edges are solved with Dirichlet boundary conditions using the \( V_p \) solutions. The vertices are then solved with Dirichlet boundary conditions from both the edges and the subdomains. For the case of a seven-point finite difference operator, the solutions on the vertices decouple completely from the subdomains so that the matrix \( B_{vp} \) is zero.

The use of the Gauss-Seidel approach has a major advantage over previous domain decomposition methods. For each iteration of the algorithm only a single solution for the large three-dimensional subproblems is required. As discussed previously, other techniques have required both a forward and backward solution of the large subdomains for symmetric preconditioning of the algorithm.

The Three-Dimensional Algorithm

A combination of both the line corrections and the block Gauss-Seidel preconditioning results in the overall iteration for three-dimensional domain decomposition as follows:

For \( k=0,1,2, \ldots \) until convergence,

1) Given the system of linear equations \( Ax^k = b \) with residual \( r^k = b - Ax^k \), perform line preconditioning as given above in equation (22):

\[ \text{Solve,} \quad c^T A c \tilde{w}^k = c^T r^k \quad (26) \]

The matrix \( C \) will be defined later but it is similar to that defined in equation (23).

2) Expand the solution \( \tilde{w} \) as follows:

\[ \tilde{w}^k = c \tilde{w}^k \quad (27) \]

and form new residuals \( r^k = r^k - A \tilde{w}^k \).

3) Perform block Gauss-Seidel preconditioning using a domain decomposition ordering such as that of Bramble, et al. by solving the following system:

\[ \begin{bmatrix}
  B_{pp} & 0 & 0 \\
  B_{EP} & B_{EE} & 0 \\
  0 & B_{EV} & B_{VV}
\end{bmatrix}
\begin{bmatrix}
  z_p \\
  z_e \\
  z_v
\end{bmatrix}

= \begin{bmatrix}
  r_p^k \\
  r_e^k \\
  r_v^k
\end{bmatrix} \quad (28) \]

with the original matrix \( A \) ordered as follows:
\[
A = \begin{bmatrix}
B_{PP} & B_{EP} & 0 \\
B_{EP}^T & B_{EE} & B_{EV} \\
0 & B_{EV} & B_{VV}
\end{bmatrix}
\] (29)

The subdomains decouple from the vertices (\(B_{PV} = 0\)) since a seven point (three-dimensional) finite difference operator is being used.

4) For the preconditioned solution \(x^k\) perform generalized conjugate residual acceleration using the algorithm described in Section 2 above and form the new solution vector \(x^{k+1}\).

5) Continue iteration in 1) if the tolerance on the 2-norm of the residuals has not been satisfied.

For the preconditioning step, solution of the subdomains can be performed in parallel, followed by parallel solution on the edges and finally solution on the vertices. As indicated by the block Gauss-Seidel matrix, the solutions at each stage become the Dirichlet boundary conditions for the subsequent stage. The theoretical convergence rate and efficiency of the algorithm can only rely on the loosely associated theorems which have been proved by several of the authors in Section 2. Since a nonsymmetric approach forms the basis of the method, little theory exists for the technique that was incorporated. Validation of the method will rely on empirical results which are given in subsequent sections of this thesis.

Algorithm Variants

The complexity of the domain decomposition algorithm with line corrections allows for innumerable variants on the technique. Many of these variants were attempted during the course of this work. The possibilities fall into two main categories: variants on line corrections and variants on domain decomposition methods.

For line corrections, the most elementary possibility is to vary the frequency with which the corrections are applied, for example, all iterations versus corrections every \(n\)-th iteration. For the sample problems in the subsequent sections it was found that application of corrections every iteration proved to be the most efficient and robust technique.

The direction in which the correction should be applied can have a dramatic effect on the rate of convergence of the technique. For one of the sample problems only \(y\)-line corrections were useful; \(x\)- and \(z\)-line corrections had little effect on convergence and in fact were detrimental to the rate of convergence. The obvious possibility exists to have a skew line correction for problems in which the dominant coefficient direction does not align with the grid. Although this possibility could be easily implemented, no attempt was made to investigate its advantages.

For problems in which there is no obvious dominant
coefficient direction, an approach was taken in which
corrections were made in all coordinate directions \( x, y, \) and
\( z \). Three variants of this approach were found to be useful:
sequential corrections applied in a Gauss-Seidel manner and
independent additive corrections with and without weighting.
From the point of view of parallel computing the independent
corrections approach is the most satisfactory since each of
the corrections can be performed on a separate processor.
The matrix form of these corrections can be stated in a
manner similar to the single line correction given
previously. If the transformation matrix \( C \) for a given
direction \( \gamma \) is given by \( C_\gamma \) with corresponding unknown
and constant vectors \( x_\gamma \) and \( b_\gamma \), then the sequential correction
is given by solution of the following equation tensor
equation:

\[
\begin{bmatrix}
N_{xy} & N_{yz} & N_{xz} \\
N_{xy} & C_x & C_y \\
N_{yz} & C_x & C_y \\
N_{xz} & 0 & C_x \\
\end{bmatrix}
\begin{bmatrix}
\phi_{xy}^k \\
\phi_{yz}^k \\
\phi_{xz}^k \\
\phi_{xy}^k \\
\phi_{yz}^k \\
\phi_{xz}^k \\
\phi_{xy}^k \\
\phi_{yz}^k \\
\phi_{xz}^k \\
\end{bmatrix} =
\begin{bmatrix}
C_z^T r^k \\
C_x^T r^k \\
C_y^T r^k \\
C_z^T r^k \\
C_x^T r^k \\
C_y^T r^k \\
C_z^T r^k \\
C_x^T r^k \\
C_y^T r^k \\
\end{bmatrix}
\]  

(30)

where, \( N_{xy} \) represents the product of the number of unknowns
in the \( x \) and \( y \) directions and so forth. The overall line
correction is found by expanding \( \phi_{xy}^k \):

\[
z^k = C_y^T x_y^k
\]  

(31)

where \( z^k \) represents the \( k \)th iterate of the preconditioned
solution \( z \).

The independent line correction solutions are somewhat
easier to represent. The following three equations are
solved separately for \( \tilde{w}_z^k \), \( \tilde{w}_x^k \), and \( \tilde{w}_y^k \):

\[
\begin{align*}
C_z^T A C_z \tilde{w}_z^k &= C_z^T r^k \\
C_x^T A C_x \tilde{w}_x^k &= C_x^T r^k \\
C_y^T A C_y \tilde{w}_y^k &= C_y^T r^k
\end{align*}
\]  

(32)

The solutions are then summed with weightings \( \phi_\gamma \) to obtain
the overall line correction:

\[
z^k = \phi_z C_z \tilde{w}_z^k + \phi_x C_x \tilde{w}_x^k + \phi_y C_y \tilde{w}_y^k
\]  

(33)

Variants on the domain decomposition algorithm which
were investigated were of two varieties: the Bramble,
Pasciak, and Schatz method\textsuperscript{14} or "vertex" method and the
Bjorstad and Widlund-type\textsuperscript{12} method or "block" method. For
both of these approaches the nonsymmetric method is used,
i.e., a block Gauss-Seidel method.

For the vertex technique there are several
possibilities for variations on the basic method. One
approach which was investigated was the variation of
boundary conditions for the edge-subdomain coupling from
Dirichlet to Neumann. This variation results in a slightly
perturbed matrix from that given originally in equation
(28) as follows:
where the matrix $\bar{B}_{EE}$ represents an alteration of the original matrix $B_{EE}$ by zeroing the diagonal contributions from the subdomains, i.e., the coefficients of $B_{PE}$ which were added to diagonal of $B_{EE}$ in forming the finite difference matrix.

For the block technique the edge solutions are eliminated and combined with the large subdomains. The iteration progresses in a red/black fashion or checkerboard fashion. First the systems for all of the red subdomains are solved. Next all of the black subdomain solutions are obtained. The matrix of this method can be viewed as successive use of block Gauss-Seidel preconditioning as follows:

$$M = \begin{bmatrix} B_{11} & 0 \\ B_{12} & B_{22} \end{bmatrix}$$

where $B_{11}$ refers to the system of equations associated with the coupled red subdomain and edges and $B_{22}$ represent the black subdomains.

One of the major differences between the method of this thesis and those of other authors is the use of approximate factorizations for the solutions on any of the subdomains, edges, or even for the line corrections. Both Kuznetsov\textsuperscript{23} and Gonzalez\textsuperscript{24} have also looked at domain decomposition with approximate factorization for the subproblems. Empirical data indicated that there was little degradation of the technique even when only one iteration of an approximate factorization such as ILU(0) with ORTHOMIN(k) acceleration was used for any of the subproblems associated with the technique. For the technique presented in this work four separate approximate factorizations were investigated for both the vertex and block methods: ILU(0), ILU(1), RS/DIAG, and RS/ILU(0) (See for example reference 3.) where RS represents reduced system with A3 (red/black) ordering and DIAG indicates diagonal scaling.

One of the obvious choices for the method is to vary the number of domains for either the vertex or block approaches. In addition to this variation, a technique was investigated in which the number of domains was varied from one iteration to the next with the idea that additional error would be reduced through the use of nonoverlapping domains. The concept of domains representing separate layers rather than an areal splitting was also investigated. Both of these latter techniques showed a degradation in performance when compared to the other domain decomposition techniques described above.
Section 4
COMPUTATIONAL ASPECTS OF THE DOMAIN DECOMPOSITION ALGORITHM

One of the main concerns of the domain decomposition-type approach is the efficiency with which the algorithm can be implemented on state-of-the-art parallel/vector computers. Two such computers were investigated: the CRAY X-MP/48 and the IBM 3090/400. Both the CRAY and IBM computers were equipped with vector/pipeline hardware on each of their four multi-processors and represent the fastest available hardware in the industry in multiprocessor supercomputers. The design of these machines is based on the globally shared resources (memory) concept although either machine has local nonshared registers and/or cache available to each of the four CPU's.

Although the concept of domain decomposition has been touted as an efficient approach for parallel computing, little has been done other than on hypothetical machines to demonstrate the power of the method. A recent publication by Keyes and Gropp\textsuperscript{25} does demonstrate the possibility of using domain decomposition on the commercially available INTEL hypercube.

Even though implementation of the algorithm on a globally shared resource parallel architecture represents the least difficulty compared to implementation on other parallel architectures, several caveats were encountered during the implementation of the three-dimensional algorithm on the two vector/parallel computers used for experiments.

Vectorization Aspects of the Method

The vectorization aspect poses a significant difficulty for domain decomposition. Although actual vectorization of a substantial portion of the algorithm is straightforward, the breaking up of the domain into smaller pieces poses a significant threat to vector efficiency since reduced vector length in the smaller subdomains means reduced vector efficiency. This is especially the case for situations in which RS/ILU(0) is used for the approximate factorization of the submatrices because the maximum vector length is halved initially for RS/ILU(0). Additionally, for three-dimensional problems this halving can become critical since a great amount of work is done with vector length of one-half of the x-direction (or equivalently the first direction in the ordering) dimension. For even large problems (> 20,000 grid points) the x-direction dimension can still be limited to much less than fifty with the resultant vector length in the approximate factorization for subproblems approaching 12 with RS/ILU(0). For many vector processors a short vector of length 12 can cause a reduction in vector processing efficiency of more than fifty percent.
The overall result of this reduction in vector efficiency is to severely limit the number of subdomains that can be used in domain decomposition if the number of parallel processors is small (<16). Experiments performed in subsequent sections verify this claim.

**Application of Parallel Processing to the Domain Decomposition Algorithm**

Implementation of any algorithm in a parallel processing environment requires a thorough understanding of the data structures within the program (FORTRAN) for the algorithm. Data resides on two levels as far as a FORTRAN application program is concerned in the parallel environment: global data known to all processors and local data known only to a single processor. Implementation of parallel processing has taken on two distinct structures for the IBM and CRAY computers which were investigated. Both IBM and CRAY offer multitasking for implementation of parallel processing. Multitasking essentially requires that the user provide separate routines for the tasks which are to run in parallel. These routines may of course simply be several copies of a single routine with different data fed by the main task to each subtask. In addition to multitasking, CRAY Research has recently provided "microtasking" which allows the user to "parallelize" within a given subroutine or collection of subroutines even down to the individual FORTRAN statement level.

Both approaches have their advantages and disadvantages as far as data structure is concerned. By requiring that parallel tasks be a subroutine or group of subroutines, multitasking eliminates much of the data analysis. For the IBM implementation, no COMMON may be passed from the main task to the subtask although the subtasks may have COMMON in which data local to the subtasks is known. All data from the main task which must be known by the subtasks must be passed as arguments to the main subroutine of the subtask. COMMON may exist within the subtasks but the data within this COMMON is known only by the given subtask and not shared with other tasks. In the CRAY implementation of multitasking all COMMON is globally shared data known to all processors. TASK COMMON is used to distinguish data which is to be locally shared by a given task.

Although the IBM implementation is somewhat restrictive, the restrictions provide a structured way of analyzing data for local and global nature. When data are passed through COMMON as in the case of CRAY multitasking, data "scoping" becomes more difficult. For large global COMMON, recognition of data types is complicated. The obvious advantage of the ability to use COMMON is that all data does not need to be passed as arguments to the subtasks.
With microtasking the scoping of data becomes even more difficult. To understand this some introduction to microtasking is required. The basic structure of microtasking is called the "fray" which is generally the subroutine level. Within the fray are located different control structures which define the separate tasks which are to be performed by the different processors. Certain tasks within the fray are performed by all processors and are therefore not under a control structure. Local data within these noncontrolled tasks is known to all processors since such tasks are performed by all processors. Within a control structure local data is known only to the processor which performs the task. Since a given processor may not perform a task defined by the control structure, data local to the control structure will only be known to the processor which executes that control structure. For this reason algorithms executed correctly on a single processor may develop incorrect results or even divergence when executed in a parallel multiprocessor environment if data scoping is performed incorrectly by the user.

For complex algorithms such as domain decomposition with large numbers of local variables within control structures, data scoping can become a major difficulty. This scoping is further complicated if subroutines within a control structure must share data. For a given task it is desirable for COMMON data to be shared but it is not desirable for this data to be globally known by all tasks as is the case for global COMMON. For this reason TASK COMMON must be defined for such data on the CRAY while for the IBM multitasking all common within a task is assumed only to be locally known.

The obvious advantage of microtasking is that several different parallel tasks may be defined within a given frayed subroutine with simple preprocessor directives. For the domain decomposition algorithm presented here the directives which were used fall into three categories: the individually defined task, the DO loop controlled task, and the fray continued from one subroutine to the next. Again each of these categories is simply defined by directives that appear as comment cards to the compiler. All parallel processing which was performed in subsequent experiments on the CRAY used microtasking for its implementation.

Definition of parallel tasks on the IBM 3090/400 was accomplished through the use of the multitasking facility (MTF). As indicated above separate subroutines must be written to perform the tasks under MTF. In addition to all data being passed through parameters to the subtasks from the main task, MTF does not allow the same data to be altered by more than one task. With microtasking no such restriction applies. Although this construct is more
general, it requires the use of the GUARD directive to prevent a variable from being updated simultaneously by more than one processor.

In addition to the local and global nature of data, data movement and storage becomes a concern in the implementation of domain decomposition. Since the coefficients used in each of the subdomains remain essentially unaltered in the internal Dirichlet boundary condition case, it might be argued that data movement and storage should be minimal problems; however, this is not the case. First to implement the subdomain solutions, it is necessary to apply the boundary conditions in the matrices for the subproblems by zeroing the appropriate coefficients. This can be done in at least two ways. One solution is simply to duplicate the coefficient matrix for the small subproblems and then to apply the boundary conditions. Another method would be to use the unaltered coefficient matrix for the original problem and to logically test for application of boundary conditions on the smaller problems. Neither solution is wholly satisfactory, but for the examples of this thesis the former approach was used with duplication of storage for coefficients.

To implement approximate factorizations on the subdomains additional storage was required. For the first iteration the approximate factorization of each subproblem was stored. For subsequent iterations, since only the right hand side constant vector of the equation is changing, this factorization of the first iteration could still be used in a forward substitution/back elimination process to obtain preconditioners for the subproblems. The savings in computation time by elimination of the factorizations was substantial but without a significant cost in storage requirements for the technique. For the case of ILU(1) factorization, for example, the storage requirements are approximately twenty-five times the dimension of the subproblems. Overall storage requirements approach about fifty times the dimension of the overall problem — about twice the standard requirements for an approximate factorization preconditioned conjugate residual method. If the subproblems were to be performed sequentially, then backing storage could be used to reduce requirements; however, for a parallel implementation of the technique this option is not available since data for all parallel tasks must reside in global memory for the globally shared resource architecture.
Section 5
EXAMPLE PROBLEMS FOR DOMAIN DECOMPOSITION COMPUTATIONAL EXPERIMENTS

Three example problems were investigated to validate the three-dimensional domain decomposition algorithm and its variants: a highly heterogeneous problem, a sample case from the SIAM/SPE comparative linear equation solution project, and an example from an IMPES petroleum reservoir simulation model.

A Symmetric Highly Heterogeneous Example

The highly heterogeneous example problem was constructed using a finite difference grid of dimension 35 by 35 by 15 in the x-, y-, and z-directions, respectively, for a total dimension of 18,375 grid cells. A block centered finite difference difference approach was used to solve the incompressible, two-phase, water-oil, reservoir fluid flow equations similar to those presented in equation (2). The IMPES reduction scheme was used to obtain a single set of coupled linear equations for the pressure in the porous media. Properties of the reservoir and fluids are given in Table 1. Permeabilities and source and sink terms were varied randomly in an exponential fashion for the permeabilities and in a linear fashion for the source and sink terms. To preserve consistency, positive and negative sources and sinks were varied as pairs so that the overall net fluid produced from and injected into the system was zero. Since an incompressible system was used, one point in the matrix at the center of the grid was fixed at zero pressure to avoid the otherwise singular system. Several experiments were made to determine the overall effect of changing reservoir conditions. The use of random sources and sinks for each successive time step in this example caused the solutions for each time step to differ substantially from the previous time step. The associated matrix for each time step appeared as difficult to solve as that of the first time step. For this reason all experiments reported in this thesis are for the solution at the initial conditions of the example.

The SIAM/SPE Comparative Solution Example

Problem number three was used as the example from the SPE/SIAM comparative solution. This example was also a finite difference matrix with grid dimensions of 35 by 11 by 13 for an overall grid of 5,005 cells. This example is characterized by extreme heterogeneities especially in the direction corresponding to the second dimension or y-direction. Many cells within the grid were either void with zero porosity and permeability or were characterized by extremely low or high permeabilities. Details of this
example are given by Sherman. A Matrix from a Reservoir Simulation Model

To investigate the application of the algorithm to a production reservoir simulation model, the technique was implemented in an IMPES-type, three-dimensional, three-phase, compressible, black-oil type reservoir simulator currently in use in an industrial environment. The model was again block-centered finite differences with homogeneous Neumann boundary conditions. The grid dimensions were 34 by 90 by 5 for a total of 15,300 points. The main purpose of this example served to demonstrate that the technique was viable in a realistic application in which the linear equation solution is part of an overall nonlinear iteration scheme.

TABLE 1
RESERVOIR PROPERTIES FOR THE HIGHLY HETEROGENEOUS EXAMPLE

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Block Dimensions</td>
<td>100 feet by 100 feet by 20 feet</td>
</tr>
<tr>
<td>Porosity</td>
<td>25 percent</td>
</tr>
<tr>
<td>Water Viscosity</td>
<td>1 centipoise</td>
</tr>
<tr>
<td>Oil Viscosity</td>
<td>10 centipoise</td>
</tr>
<tr>
<td>Permeability</td>
<td>1000 millidarcies</td>
</tr>
<tr>
<td>Random variation applied as follows:</td>
<td></td>
</tr>
<tr>
<td>$K = 0.001 R_{random} K$</td>
<td></td>
</tr>
<tr>
<td>Vertical permeability reduced by 0.1</td>
<td></td>
</tr>
<tr>
<td>Relative Permeabilities:</td>
<td></td>
</tr>
<tr>
<td>Water:</td>
<td>$k_{rw} = S_w^2$</td>
</tr>
<tr>
<td>Oil:</td>
<td>$k_{ro} = (1 - S_w^2)$</td>
</tr>
<tr>
<td>Oil-Water Capillary Pressure (psi):</td>
<td>$P_{caw} = 15 - (1 - S_w)^2$</td>
</tr>
<tr>
<td>Water Density</td>
<td>61.92 lb/ft$^3$</td>
</tr>
<tr>
<td>Oil Density</td>
<td>53.28 lb/ft$^3$</td>
</tr>
<tr>
<td>Initial Water Saturation</td>
<td>0.0</td>
</tr>
<tr>
<td>Source and sink locations:</td>
<td></td>
</tr>
<tr>
<td>1. $x = 1$ $y = 1$ $z = 1$ $-10$ Relative Strength $= +500.$</td>
<td></td>
</tr>
<tr>
<td>2. $x = 35$ $y = 35$ $z = 6$ $-15$ Relative Strength $= -500.$</td>
<td></td>
</tr>
<tr>
<td>3. $x = 23$ $y = 23$ $z = 5$ $-14$ Relative Strength $= +500.$</td>
<td></td>
</tr>
<tr>
<td>4. $x = 14$ $y = 14$ $z = 3$ $-12$ Relative Strength $= -500.$</td>
<td></td>
</tr>
<tr>
<td>5. $x = 14$ $y = 14$ $z = 3$ $-12$ Relative Strength $= +500.$</td>
<td></td>
</tr>
<tr>
<td>6. $x = 27$ $y = 5$ $z = 4$ $-13$ Relative Strength $= -500.$</td>
<td></td>
</tr>
<tr>
<td>7. $x = 13$ $y = 27$ $z = 2$ $-11$ Relative Strength $= +500.$</td>
<td></td>
</tr>
<tr>
<td>8. $x = 35$ $y = 5$ $z = 6$ $-15$ Relative Strength $= -500.$</td>
<td></td>
</tr>
<tr>
<td>9. $x = 5$ $y = 35$ $z = 4$ $-13$ Relative Strength $= +500.$</td>
<td></td>
</tr>
<tr>
<td>10. $x = 21$ $y = 4$ $z = 2$ $-11$ Relative Strength $= -500.$</td>
<td></td>
</tr>
</tbody>
</table>
Section 6
COMPUTATIONAL EXPERIMENTS

The following chapter describes the experiments which were performed to validate the three-dimensional domain decomposition technique and to investigate the effects of various parameters on the method. The first section compares the method with other methods which are currently in use for the three examples. Next, various parameter effects are investigated for domain decomposition for the SIAM/SPE and heterogeneous examples. Finally, results are given for the application of parallel computation to domain decomposition. All examples were performed in a vector/pipeline environment on a Cray X-MP/24 unless otherwise noted so that the comparisons are on a basis of state-of-the-art computing. Some examples were performed in scalar simply to demonstrate the vector efficiency of the algorithm. The work estimates given in the following figures are based on normalizing the CPU time for the experiments. The CPU time per iteration was calculated for each of the examples in a given figure. These times were then divided by the CPU time per iteration for the base case of the figure to obtain the factor by which to multiply the iteration count for the various examples.

Comparison of Domain Decomposition with Other Preconditioners

Six separate methods were compared to the three-dimensional domain decomposition algorithm. These methods fall into two categories: preconditioned conjugate gradient (PCG) and preconditioned generalized conjugate residual (PGCR) methods. Since the preconditioned conjugate gradient method has limited applicability, the PCG methods which were investigated are somewhat simplistic: no preconditioning and Jacobi (diagonal) preconditioning. Although simplistic, these methods do give a basis from which to compare the other methods. Four preconditioners were investigated for the generalized conjugate residual method (ORTHOMIN(k)): incomplete factorizations ILU(0) and ILU(1), and reduced system with incomplete factorization ILU(0) and Jacobi (diagonal) preconditioning. (See for example reference 27.) These methods will be subsequently referred to as ILU(0), ILU(1), RS/ILU(0), and RS/DIA. All of the comparisons are with the three-dimensional domain decomposition with line corrections and using the 2 by 2 domain, vertex method similar to that described by Bramble et al. Except where indicated RS/ILU(0) was used to obtain approximate solutions for the lines and subdomains. For the line corrections a residual reduction of 0.01 was used as tolerance whereas for the subdomains and edges only one or two iterations were
performed for the SIAM/SPE example or heterogeneous example, respectively. Y-line corrections were used for the SIAM/SPE problem and simultaneous x-, y-, and z-lines with summed corrections were used in the heterogeneous example for the base cases.

Figure 9 compares the three-dimensional domain decomposition technique with the unpreconditioned CG and Jacobi preconditioned CG methods for the SIAM/SPE example. As shown in the figure, the relative efficiency of the domain decomposition method in a vector/pipeline computational environment is about a factor of three better than the Jacobi CG technique and is orders of magnitude better than the nonpreconditioned method. For the highly heterogeneous example Figure 10 demonstrates that the Jacobi CG method fairs much better. The CG method is not competitive; however, the Jacobi CG method is only twenty percent less efficient than the domain decomposition method. Although the number of arithmetic operations to convergence for Jacobi CG may be a factor of two or three greater than methods with better preconditionings, the vector efficiency of the Jacobi CG method is substantially greater than more robust but less vectorizable preconditioners. Overall, these examples do show that the domain decomposition algorithms of this thesis is more efficient than the two CG methods.

Figure 9. Comparison of Relative Work for the Vertex, CG, and Jacobi/CG Methods for the SIAM/SPE Example
methods presented here.

Figure 11 gives a comparison for the SIAM/SPE example of domain decomposition with four preconditioners for the conjugate residual method: ILU(1), ILU(0), RS/ILU(0), and RS/DIAG. Clearly, both domain decomposition and RS/ILU(0) are substantially more efficient for this example. RS/DIAG failed to converge and ILU(0) had great difficulty converging. Although ILU(1) did converge in fewer iterations than RS/ILU(0) (83 versus 98 iterations) the high percentage of scalar work in ILU(1) dictate that its efficiency per iteration is much less than that of RS/ILU(0). The convergence rate of the domain decomposition method was extremely good with only 11 iterations required. The small size of the three-dimensional subdomains for this example caused the vector efficiency to be reduced for domain decomposition. For larger subdomains the comparison should be even more dramatic.

The example for the heterogeneous case demonstrates this even more clearly as shown in Figure 12. For this 35 by 35 by 15 example domain decomposition is again clearly more efficient by almost a factor of two over RS/ILU(0). The additional gain in efficiency compared to the SIAM/SPE example is due somewhat to the larger domains (17 by 17 areally) used for the three-dimensional subproblems. For
Figure 11. Comparison of Relative Work for the Vertex Method and Other Preconditioned GCR Methods for the SIAM/SPE Example

Figure 12. Comparison of Relative Work for the Vertex Method and Other Preconditioned GCR Methods for the Heterogeneous Example
this example the ILU(1) preconditioner performs somewhat better than RS/ILU(0) but still thirty percent less efficiently than domain decomposition.

Effects of the Number of Domains on Convergence Rate and Efficiency

Several experiments were run comparing the effects of the number of domains on the three-dimensional domain decomposition algorithm. For the SIAM/SPE example, Figure 13 shows the effect on convergence rate for five groupings of domains: 2 by 2, 3 by 3, 4 by 3, 6 by 2, and 6 by 3. There is no question that the convergence rate does depend inversely on the number of domains although this dependence is slight. This result is in agreement with the theory of Widlund since a preconditioner similar to that of Glowinski was used for the algorithm of these examples. The dominance of the y-direction transmissibilities for this example is evident in that the 6 by 2 domains converge more rapidly than either the 3 by 3 or 4 by 3 cases with fewer and the same number of domains respectively. The number of domains in the y-direction must dominate. Relative work comparisons on a vector basis in Figure 14 show that 2 by 2 is clearly best followed by the 3 by 3 case. The 3 by 3 case is able to overcome its slower convergence rate with vectorization efficiencies that are greater than the more rapidly convergent 6 by 2 case since the vector length in

Figure 13: Effect of the Number of Domains on the Convergence Rate for the Vertex Method for the SIAM/SPE Example
the x-direction dominates the work for RS/ILU(0). This is more clearly demonstrated in Figure 15 in the comparison of the case on a scalar basis where the 6 by 2 and 3 by 3 case are shown to have the same relative efficiencies. The 4 by 3 and 6 by 3 cases appear to be equivalent in both convergence rate and efficiency.

For the heterogeneous example seven different cases with differing numbers of domains were compared: 2 by 2, 3 by 3, 3 by 4, 4 by 3, 4 by 4, 4 by 6, and 6 by 4. As shown in Figure 16 there are three distinct groupings of cases for the convergence rate. The convergence rate dependence on the number of domains can again be observed. The 2 by 2 case is clearly best, followed by a group of four cases (3 by 3, 4 by 3, 3 by 4, and 4 by 4), and last a group of the 4 by 6 and 6 by 4 cases. The dependence of efficiency on vector length is clearly demonstrated in Figure 17. The 4 by 3 and 4 by 4 cases separate from the 3 by 3 and 3 by four cases because of the substantially shorter x-direction vector length in the former two cases. The 4 by 6 and 6 by 4 cases maintain about the same poor efficiency because of the extremely short vector lengths in either of the cases. Figure 18 shows that all of the cases with the exception of the 2 by 2 case have about the same scalar efficiency. Any differences are probably due to data movement for either the scalar or vector cases.

Figure 14. Comparison of Vector Work for Various Numbers of Domains for the Vertex Method for the SIAH/SPE Example
Figure 15. Comparison of Scalar Work for Various Numbers of Domains for the Vertex Method for the SIAM/SPE Example

Figure 16. Effect of the Number of Domains on the Convergence Rate for the Vertex Method for the Heterogeneous Example
Effect of Subproblem Tolerances

Figures 19-22 depict the effect of varying the tolerance with which subproblems are solved in the domain decomposition algorithm. For the SIAM/SPE example Figure 19 shows that the accuracy with which subproblems are solved has little effect on convergence rate. For the example shown here the optimal case had tolerances of 0.01 for the lines correction solutions and a single iteration of RS/IIIJ(0) for the 3D and edge solutions. As shown in the figure increasing the accuracy on the subdomains increased the rate of convergence slightly. Decreasing the accuracy of the line solutions had little effect on the convergence rate even when a tolerance of 0.1 was used. A tolerance of 0.01 on the subdomains with a tolerance of 10^{-6} for the lines resulted in a convergence rate the same as the optimal case. Figure 20 shows more clearly the effect of the change in tolerance on overall efficiency of the method. Although reduction in the subdomain tolerances resulted in a slight improvement in convergence rate, the associated computational cost resulted in a factor of about 2.5 reduction in efficiency as shown in Figure 20. Reduction in the accuracy of the subdomain solutions resulted in an efficiency improvement of about a factor of two. Finally, reduction of the line correction accuracy gives further efficiency improvement.

Figure 19. Effect of Subproblem Tolerance on Convergence Rate for the SIAM/SPE Example
Figure 21 shows results for the tolerance effects for the heterogeneous example. Similar conclusions can be drawn for this case. Increase in the accuracy results in only a slight increase in the convergence rate, but Figure 22 shows that again, the cost in computational efficiency for the slight convergence rate improvement is large.

Evaluation of Line Corrections for Domain Decomposition

Since the use of line corrections for domain decomposition is one of the major contributions of this work, an evaluation of their utility is necessary. Figures 23–28 show the effects of various line corrections on the convergence rate of the three-dimensional domain decomposition algorithm for the two example problems. For the SIAM/SPE example, the effect of y-line corrections is dramatic as shown in Figure 23. The case with no line corrections converges only slowly compared to the y-line correction case. For this example, Figure 24 shows that x and z-line corrections have little effect. Figure 25 shows the effect of various combinations of x, y, and z-line corrections on the convergence rate of the SPE/SIAM example. The xyz correction refers to the case in which x, y, and z-line corrections are calculated simultaneously. The corrections are then summed and added to the overall solution vector. A variant on this approach is to weight
Figure 21. Effect of Subproblem Tolerance on Convergence Rate for the Heterogeneous Example

Figure 22. Effect of Subproblem Tolerance on Relative Work for the Heterogeneous Example
Figure 34. Effect of Y-, Z- or 3-Line Corrections on Convergence Rates for the SIM/SP Example

Figure 35. Effect of Optimal Line Corrections Versus No Line Corrections for the SIM/SP Example

SIAM/SP Example

Comparison of Convergence Rates

Effect of Line Corrections

Iteration Number

Residual Reduction (L2)

X-Line

Y-Line (Optimal)

Z-Line

Line Correction

No Correction

Line Correction
each of the corrections before adding them to the solution vector. The xyz $L^2$ weighted scheme refers to a method by which a weighted average using the $L^2$ norms of the residuals in each of the two-dimensional correction matrix equations is used.

Finally, the xyz sequential scheme solves for the x-line corrections first, updates the residuals, and then solves for the y-line corrections, and updates the residuals, and so forth.

As shown in Figure 25 although the y-line technique is optimal for this example, the xyz sequential corrections give nearly the same rate of convergence. The xyz $L^2$ weighted method is somewhat worse than the optimal, followed by the xyz simultaneous method. These latter two techniques, although less efficient for the SPE/SIAM example, have the advantage of allowing the three line corrections to be performed in parallel.

Figure 26 shows the convergence rate for the heterogeneous example with and without line corrections. The optimal case shown here corresponds to the xyz simultaneous method described above. As shown in the figure, the convergence rate without line corrections is extremely slow after a residual reduction of $10^{-2}$ is reached. Figure 27 compares x, y, and z-line corrections with the optimal for the heterogeneous example. As shown
Figure 26. Effect of Optimal Line Corrections Versus No Line Corrections for the Heterogeneous Example

Figure 27. Effect of $X$, $Y$, or $Z$-line Corrections on Convergence Rate for the Heterogeneous Example
the z-line correction appears to be the best of the three directions, but it is still somewhat worse than the optimal. Although the example was constructed to avoid a dominant direction for the coefficients, the orientation of source and sink terms vertically may have improved the ability of the z-line corrections to give a better starting solution for the problem. Finally, Figure 28 shows that the XYZ sequential and simultaneous correction techniques give similar results for convergence rate. The L^2 weighted scheme was somewhat worse. The fact that the XYZ simultaneous correction approach can be solved in parallel makes it the obvious method of choice for this example.

Comparisons of Subproblem Preconditioners

Figures 29-32 show the effect of the technique used for solution of the subproblems on convergence rate and efficiency for the two example problems. Four techniques were investigated: ILU(0), ILU(1), RS/DIA and RS/ILU(0). Each of these methods was accelerated by using ORTHOMIN(k) during the course of the iterations. For the SIAM/SPE example a tolerance of 0.01 residual reduction or 10 iterations was used for the line corrections. Only a single iteration was allowed for either the three-dimensional or edge subproblems. As shown in Figure 29 ILU(1) and RS/ILU(0) behave similarly as expected since they are almost equivalent techniques. ILU(0) performs reasonably close to Figure 28. Effect of Combination Line Corrections on the Convergence Rate for the Heterogeneous Example
Figure 29. Comparison of Convergence Rates for Various Subproblem Preconditioners for the SIAH/SPE Example

the other methods whereas RS/DIAG is not competitive. Figure 30 gives an indication of the relative efficiencies of the various preconditioners for the subproblems. Since RS/ILU(0) solves a system of half the dimension of ILU(1) and since RS/ILU(0) can be vectorized as opposed to ILU(1), RS/ILU(0) is clearly more efficient. Even though ILU(0) is relatively efficient on a per iteration basis, this cannot overcome the loss of efficiency caused by the greater number of iterations that it requires to solve the system. RS/DIAG falls far behind the other method for this example.

Similar results were obtained for the heterogeneous example as shown in Figures 31 and 32. For this case both RS/ILU(0) and ILU(1) are again the best in convergence rate and efficiency but the overall efficiency of RS/ILU(0) dominates. For this more difficult example, ILU(0) is not as competitive in either convergence rate or efficiency. RS/DIAG comes close in overall efficiency to ILU(0) for this example.

Comparisons of Various Domain Decomposition Techniques

Several variants of the domain decomposition algorithm were investigated. From the discussion in Section 2, two types of domain decomposition methods with conjugate residuals applied to the overall system can be derived. As indicated previously these two methods are based on the work of Bjorstad and Widlund, and Bramble, Pasciak, and
Figure 30. Comparison of Relative Work for Various Subproblem Preconditioners for the SIAN/SPE Example

Figure 31. Comparison of Convergence Rates for Various Subproblem Preconditioners for the Heterogeneous Example
Schatz\textsuperscript{14} and will be referred to as the "block" and "vertex" methods respectively. The vertex method is the same as that used in the examples prior to this section. The block method involves only red/black large subdomain solutions; unlike the vertex method, no edge or vertex solutions are required. This seeming advantage of the block method is lost when parallel processing considerations are taken into account. Because only half of the domains are solved at a given time, the degree of parallelism for the block method is half of that of the vertex method for the same number of major subdomains.

Figures 33 and 34 compare the 2 by 2 vertex method domains with the block method (2 by 2 and 4 by 2 domains) for the SIAM/SPE example. As shown in Figure 33 the convergence rates of both the block and vertex methods are essentially the same. It is interesting to note that there appears to be little degradation of the block method with increasing numbers of domains. The results of Figure 33 demonstrate that the block method as implemented here tends to obey the theory proposed by Widlund et al. That is, the convergence rate of the block method appears to be the best indicating a smaller condition number for the preconditioned system $M^{-1}A$. There appears to be little degradation of the method with the number of domains.
Figure 34 shows that the relative work for the block 2 by 2 methods is comparable. The minor advantage shown for the vertex method results from the slightly smaller three-dimensional subproblems which are solved. The block method involves solutions of three-dimensional subproblems of areal dimensions one greater than the corresponding vertex method since the edge solutions are included in the three-dimensional subdomains. The degradation shown for the block 4 by 2 domain example is simply due to the loss in vector efficiency caused by the decreased x-dimension for the three-dimensional subproblems.

Figures 35 and 36 compare the block and vertex methods for the heterogeneous example. Figure 35 again demonstrates the claim that the block method shows little degradation with the number of domains and that the block method converges more rapidly than the vertex method. The relative work comparisons shown in Figure 36 indicate that the work is again dominated by the maximum vector length in the x-direction for the subdomains. The block and vertex 2 by 2 domain methods are comparable in efficiency. The block 2 by 4 case is only slightly worse, but the block 4 by 2 and 4 by 4 cases are substantially worse due to their short x-direction vector lengths.

A natural variant on the domain decomposition method is to use strips to enhance the vector length in the
Figure 34. Comparison of Relative Work for Block and Vertex Methods for the CMU/SD Example

Figure 35. Comparison of Convergence Rates for Block and Vertex Methods for the Heterogeneous Example
preconditioners. For example, the number of domains in one of the areal directions is unity for the strip methods. For the vertex technique the strip application results in the elimination of the vertex solutions. The iteration proceeds from line corrections to strip domains to edges and finally to conjugate residual acceleration. The block domain decomposition method remains unaltered.

Figure 37 compares the convergence rate for the block and vertex strip methods for the SIAM/SPE example. The relatively small dependence of the block method on the number of domains is demonstrated whereas the vertex method shows a much greater dependence on the number of domains. The convergence rate of the block 4 by 1 strip case is slightly better than the vertex 2 by 2 base case. The convergence rate of the block 1 by 4 case is somewhat less than the best cases again demonstrating the strong coupling in the y-direction for this example. The relative work comparisons shown in Figure 38 indicate that the vector length of the preconditioners dominates the computational work of the methods. Although the convergence rate of the block 1 by 4 case was substantially less than the vertex 2 by 2 base case, the increased vector lengths for the block 1 by 4 case improve its efficiency so that it becomes comparable to the vertex 2 by 2 case. The orderings of the additional cases show a dependence on both the maximum vector length in
the x-direction and the strong coupling in the y-direction for the SIAM/SPE example.

Figures 39-42 compare the vertex and block strip methods with the vertex base case for the heterogeneous example. Figure 39 demonstrates the strong dependence of the vertex method on the number of strips. The 4 by 1 and the 1 by 4 cases show comparable rates of convergence as do the 7 by 1 and 1 by 7 cases. The 2 by 2 case is clearly more rapidly convergent. The relative work comparisons shown in Figure 40 indicate the effect of vector length in the x-direction on the efficiency of the method. For the 1 by 4 and 1 by 7 cases the increased vector length improves the efficiency so that these cases are comparable to the vertex 2 by 2 case. The degradation of the 4 by 1 and 7 by 1 cases due to vector length is clearly shown in Figure 40.

Figure 41 compares convergence rate of the block strip method with the vertex 2 by 2 base case for the heterogeneous example. The minor dependence of the block method on the number of domains (strips) is evident from the figure. Figure 42 indicates that the work for the different cases is again highly dependent on the vector length. Because of the increased vector length the 1 by 4 case is substantially better than the vertex 2 by 2 base case and the 1 by 7 case becomes comparable to the base case.

Comparisons with Block Jacobi Methods

---

**Figure 39. Comparison of Convergence Rates of Vertex Strips for the Heterogeneous Example**
Figure 41. Comparison of Convergence Rates for Block Strips for the Heterogeneous Example

Figure 40. Comparison of Relative Work of Vertex Strips for the Heterogeneous Example
Because of the success of the block method demonstrated above it appeared evident that further investigation should be done to improve the parallelism of the block method. An obvious choice is to revert from the block line corrected Gauss-Seidel procedure used above to a block Jacobi preconditioner with line corrections. This corresponds to a change in the preconditioning matrix $M$ from the following form for the block Gauss-Seidel method:

$$M = \begin{bmatrix} B_{11} & 0 \\ B_{12} & B_{22} \end{bmatrix}$$  \hspace{1cm} (36)$$

to the block Jacobi matrix:

$$M = \begin{bmatrix} B_{11} & 0 \\ 0 & B_{22} \end{bmatrix}$$  \hspace{1cm} (37)$$

In equations 36 and 37 $B_{11}$ refers to the system of equations associated with the coupled red subdomain and edges and $B_{22}$ represent the black subdomains.

Figures 43 and 44 compare the block Jacobi (Block 4x1 SIM) preconditioner with line corrections with the vertex 2 by 2 base case and the block 4 by 1 Gauss-Seidel (Block 4x1 SEQ) method. As shown in Figures 43 and 44, the block Jacobi method is about a factor of three less efficient than either the block Gauss-Seidel or vertex base case. The vertex 2 by 2 base case remains the method of choice for two reasons. First, the vertex method appears
Figure 43. Comparison of Convergence Rates for the Simultaneous and Sequential Block Methods for the SIAM/SPE Example

Figure 44. Comparison of Relative Work for the Simultaneous and Sequential Block Methods for the SIAM/SPE Example
to be the most efficient of the methods. Second, the vertex method has the highest degree of parallelism, at least for situations limited to a small number of processors (§4). For the heterogeneous example the comparison of the block Jacobi preconditioner with the block Gauss-Seidel and vertex base case is more favorable as shown in Figures 45 and 46. Although the convergence rate for the Jacobi method is about fifty percent less efficient than that for the other two cases, the long vector efficiency of the 1 by 4 strips improves the performance of the Jacobi case so that it is only twenty-five percent less efficient than the base case. However, the block Gauss-Seidel 1 by 4 case remains about fifty percent more efficient than the Jacobi case.

If the number of processors in a parallel environment is large such that there is an excess of processors to the number of jobs in a given system, it appears that the block Jacobi method with line corrections may be the method of choice. Even though the total computation time for the method will be greater than the other methods discussed here, the elapsed time could be significantly less because of the high degree of parallelism and large granularity of the block Jacobi approach.

Comparison of Symmetric and Nonsymmetric Preconditioning

The final comparison of domain decomposition techniques

Figure 45. Comparison of Convergence Rates for the Vertex and Simultaneous/Sequential Block Methods for the Heterogeneous Example
involves symmetric and nonsymmetric block preconditioning for the method. Theory predicts that the conjugate gradient method would perform poorly for the nonsymmetric block Gauss-Seidel vertex method. However, if the preconditioner is made to be symmetric as given in equation (22), can the acceleration provided by the improved performance of the conjugate method overcome the increased cost? Figures 47 and 48 demonstrate the answer to this question. These figures show a comparison of the vertex 2 by 2 base case (NONSYN GCR) to symmetric preconditioners for both the conjugate gradient and generalized conjugate residual accelerations (SYM CG and SYM GCR, respectively) and a block Gauss-Seidel vertex case with conjugate gradient acceleration (NONSYM CG). As shown in the figures, symmetrization of the preconditioner does improve the convergence rate of either the conjugate (CG) or generalized conjugate residual (GCR) accelerations; however, the improvement is not sufficient to overcome the inefficiency of the symmetric preconditioner. Figure 48 shows that the work associated with the symmetric preconditioner is almost a factor of two greater than the nonsymmetric case. For this reason the nonsymmetric GCR method is substantially better than the symmetric GCR case even though the convergence rate is somewhat better. The improved convergence rate of the symmetric CG case is
Figure 47. Comparison of Convergence Rates for Symmetric and Nonsymmetric CG/GCR Vertex Method for the Heterogeneous Example

Figure 48. Comparison of Relative Work for Symmetric and Nonsymmetric CG/GCR Vertex Method for the Heterogeneous Example
sufficient to cause it to break even with the nonsymmetric CG case; however, the overall convergence rate of either of these cases is still sufficiently poor that the GCR method is clearly better, especially with nonsymmetric Gauss-Seidel preconditioning.

Comparison of Neumann and Dirichlet Boundary Conditions for Edge Subdomains

Figure 49 compares the use of Neumann boundary conditions for the edge subdomain solutions, as described in Section 3, with the base case using Dirichlet boundary conditions. Although the initial iteration of the Neumann case is slightly better, the subsequent convergence rate is substantially slower overall than the Dirichlet case. For this example, the same number of iterations were used for each of the edge approximate solutions. Since the Neumann problems were less diagonally dominant than the Dirichlet, with the same number of iterations the solutions are somewhat worse for the Neumann case. However, an additional experiment showed that the convergence rate for the Neumann case was unchanged when the solution tolerances for the edges were reduced to $10^{-6}$ from the original $10^{-2}$. Because of the increased amount of work necessary for the Neumann case and because of its poorer rate of convergence, the Dirichlet technique is considered the best for the edge solutions.

---

Figure 49. Effect of Edge Neumann and Dirichlet Boundary Conditions on the Convergence Rate of the Vertex Method
Application to a Petroleum Reservoir Simulator

The vertex 2 by 2 domain decomposition technique was incorporated into a production industrial reservoir simulation model as described in the previous chapter. Results from experiments, although somewhat disappointing, do serve as a basis for application of the domain decomposition algorithm. For the 15,300 cell model the domain decomposition method was less efficient in converging to the same residual compared to RS/ILU(0) for the full system by a factor that ranged from 2.81 to 3.41 for an average of about a factor of 3.20. The discrepancy between these results and those of the two examples can easily be explained. All of the examples indicated that the convergence rate of the domain decomposition method with line corrections was linear after the first iteration with a constant reduction from one iteration to the next in the range 0.4 - 0.6. This indicates that the method although convergent, will require almost the same number of iterations, regardless of the condition number of the matrix, to converge to a given tolerance. For the example problems the diagonal dominance was extremely small (with likely large condition number) thus making the convergence of any technique difficult. For the case of the production reservoir model, the diagonal dominance was large and therefore RS/ILU(0) converged rapidly, often requiring fewer than ten iterations. Since for the same tolerance the domain decomposition method required about six iterations and each iteration was about three times more costly than the equivalent RS/ILU(0) full system iteration, the overall result was the poor performance that was observed. One of the main reasons for the poor performance was also the short vectors for the 2 by 2 vertex scheme.

The tests on the production model did indicate a possible area for further application of the domain decomposition method. For the reservoir model it was noted that the nonlinear portion of the algorithm consumed a considerable portion of the computing time. With domain decomposition, the possibility exists to incorporate the nonlinear iteration into the overall subdomain iterations. In this manner further parallelization of the entire computational problem can be performed.

To test this hypothesis, only a single iteration of the domain decomposition algorithm was performed prior to updating the nonlinear portion of the overall solution strategy. Since a single iteration for this example resulted in a residual reduction of greater than an order of magnitude, it was anticipated that the nonlinear convergence would not be adversely affected. Results for a about forty time steps of the simulation indicated that this approach was viable. Compared to a full matrix RS/ILU(0) scheme with
residual reduction tolerance of 0.01, the single iteration domain decomposition method gave convergence rates for the nonlinear iterations which were nearly the same. Overall the computation time for the example was only eight percent slower than the RS/ILU(0) full matrix solution.

Parallel/Vector Applications of the Domain Decomposition Algorithm

To investigate the performance of the line corrected domain decomposition algorithm, the vertex 2 x 2 method, heterogeneous case was chosen. Three major levels of parallelism exist for the vertex method: the parallel x, y, and z-line corrections, the parallel three-dimensional subdomains, and the parallel edge solutions. For this example, the vertex method was converted to multitasking for the CRAY X-MP/48 implementation and MTF multitasking was implemented for the IBM 3090/400. Timings were made for the three major levels of parallelism as well as an overall timing. Table 2 shows the ratio of CPU time to elapsed time for the two computers on which the algorithm was evaluated. Two measurements appear in parentheses for the CRAY timings. These represent optimizations which were performed over the original IBM timings. For the edge solutions it was noted that the IBM timings had been made with different numbers of iterations for each of the domains. This heterogeneous multitasking results in inefficiencies since one or more of the processors with the smallest amount of work must wait for the other processors to complete their tasks. The overall result is a reduction in efficiency of about a factor of two for the edge solutions. Since almost thirty percent of the CPU time is spent in the line corrections, the overall improvement is significantly limited.

Theoretically, the best improvement for this case with ideal parallel implementation for all of the major subtasks is slightly over three. The times of 2.39 and 2.31 for the CRAY and IBM computers indicate that considerable efficiency improvements could be made in the algorithms over the best case of 80% for the times in the table.

The parenthetical numbers in the table indicate that it is possible to make such improvements. First, for the edge solutions if the number of iterations is kept constant for all solutions, the ratio of CPU to elapsed time improves from 1.9 to 3.4. Next multitasking can be implemented on the portions of the algorithms other than the three main tasks. In particular, the generalized conjugate residual algorithm and residual calculations between steps of the Gauss-Seidel algorithm can be performed in parallel. With these improvements in the parallel implementation of the algorithms, the overall ratio of CPU to elapsed time improved from 2.39 to 2.98 or almost ninety percent of the theoretical maximum of 3.4 for zero overhead of parallel
computations. Computation rates for the CRAY X-MP/48 were approximately 100 million floating point operations per second compared to about 35 million for the single processor case.

To achieve an overall reduction in elapsed time to a ratio of greater than 3.5 would require the the line corrections be done in parallel. An obvious method to use for the line corrections is simply to implement a two-dimensional version of domain decomposition. A block Jacobi 2 by 2 method was implemented for the example problem for the line corrections with somewhat disappointing results. Convergence rates for the line correction portion of the solution were poor even with the addition of the one-dimensional solution for line corrections.

The results of this section indicate the direction for the domain decomposition approach to take in the future when larger numbers of processors will be available. Since the majority of the CPU time is spent in the solutions for the three-dimensional subdomains, smaller tolerances could be used for this portion of the algorithm. Similarly, the tolerances could be relaxed for the line corrections. As shown in the experiments with the subdomains, these changes would tend to slightly improve the convergence rate of the algorithm for about a factor of two decrease in computational efficiency. The advantage of this approach is to significantly improve the parallel performance for the algorithm. Experiments on the IBM processor resulted in ratios near 3.6 for the three-dimensional portion of the solutions when tolerances were increased. For the CRAY these ratios were greater than 3.9. If at some point in the future the expense of an additional processor is small, the extra cost of reduced efficiency of the method with smaller three-dimensional tolerances will be offset by the ability to achieve performance of the algorithm near the theoretical maximum for a given parallel environment. Further computations on future hardware may be necessary to substantiate this conjecture.
Table 2
Comparison of Parallel Implementations of Domain Decomposition

<table>
<thead>
<tr>
<th></th>
<th>CRAY X-MP/48</th>
<th>IBM 3090/400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Corrections</td>
<td>2.015</td>
<td>1.72</td>
</tr>
<tr>
<td>3D Subdomains</td>
<td>3.721</td>
<td>3.40</td>
</tr>
<tr>
<td>Edge Solutions</td>
<td>1.90 (3.45)</td>
<td>1.55</td>
</tr>
<tr>
<td>Total Iteration</td>
<td>2.39 (2.98)</td>
<td>2.31</td>
</tr>
</tbody>
</table>

Section 7
CONCLUSIONS

A three-dimensional domain decomposition algorithm for the numerical solution of finite difference approximations of elliptic or parabolic partial differential equations was developed. Numerical experiments indicated that the algorithm was both robust and efficient. Comparisons on a class of difficult problems demonstrated that the method was more efficient than other techniques such as preconditioned generalized conjugate residual methods with incomplete factorization or reduced system incomplete factorization preconditioners. Two variants of the method appeared to be the optimum for the numerical experiments which were performed: the Bramble, Pasciak, and Schatz-type or vertex method and the Bjorstad and Widlund-type or block method. Overall, the vertex method appeared to have the best performance for both efficiency and robustness.

Computational experiments showed that the vertex approach as implemented had a slight dependence on the number of domains while the block method showed little dependence. The vertex method appeared to be somewhat more efficient as far as parallel implementation of domain decomposition. That is, for a given number of domains the
vertex method has a higher degree of parallelism. For the same degree of parallelism, the vertex method exhibits better vector efficiency; the block method requires twice the number of three-dimensional subdomains with a resultant overall shorter average vector length.

Implementation of strips for the domains showed some vector efficiency improvements for the heterogeneous example. For the example from the SIAM/SPE comparative solution project, strips showed poorer rates of convergence especially when oriented in the most advantageous way for vector efficiency. Nonetheless, there appears to be a class of problems on which the strip technique may be the optimum.

Parallel implementations of the algorithm on the CRAY X-MP/48 and IBM 3090/400 showed that the method could perform efficiently in a globally shared resources environment. A factor of three reduction in elapsed time was observed on the CRAY X-MP/48 for a microtasked implementation of the vertex algorithm. Corresponding computational rates were improved from about thirty-five million to one hundred million floating point operations per second. Further improvements of the parallel performance depend on a more efficient line correction scheme or equivalently a technique in which the highly parallel, large granularity three-dimensional subdomains dominate.

REFERENCES


19. Pasciak, J. E., manuscript on a three-dimensional preconditioner using substructures, in progress.