M A T H E M A T I C S D E P A R T M E N T

The Implementation of a Direct Domain Decomposition Method on a Small Transputer System

K.J. Neylon

Numerical Analysis Report 3/92

U N I V E R S I T Y O F R E A D I N G

The Implementation of a Direct Domain Decomposition Method on a Small Transputer System

K.J. Neylon

Numerical Analysis Report 3/92

Department of Mathematics
P.O. Box 220
University of Reading
Whiteknights
Reading
RG6 2AX
United Kingdom

Abstract

Domain decomposition methods for the solution of BVPs divide the problem into smaller sub-problems at the discretisation stage by dividing the domain into sub-domains. On each of these sub-domains, the corresponding sub-problems can be solved separately before being combined to give the solution on the complete domain. This has obvious advantages in parallel computation where each sub-problem can be solved simultaneously on separate processors. Also, non-concurrency based benefits exist for particular types of problems.

Most domain decomposition methods have some iterative feature or involve the solution of a dense matrix system. In this report, a direct domain decomposition method which involves matrix systems with the normal discretisation sparsity pattern is described. Numerical results from the implementation of this method on a two transputer parallel system are presented and execution time values given show that the method does indeed parallelise well. However, it is also found that a restrictively small time step is needed for the error introduced by the decomposition to be of the same order as the error of the underlying discretisation.

Contents

1 Introduction

The need to perform extremely large numerical calculations in reasonable time scales has led to the need for greater and greater computing power. One way of fulfilling this need is to increase the performance of individual processing units above the level currently attainable in semiconductor based devices by the use of radical new materials (e.g. superconductors) but, as yet, much work is needed before this becomes a practical possibility.

Since it is relatively cheap to mass-produce powerful semiconductor based processors, another way of increasing power is to use separate processors on a single application at the same time to share the computational task - parallel computation. All modern 'super-computers' are parallel machines. There are two important distinctions when performing a simple classification of a parallel machine; the degree of granularity and the type of memory access. There may be a small number of sophisticated processors (coarse granularity), each of which can perform reasonably complex computational tasks independently, or there may be many simple processors (fine granularity), each of which can only perform small computational tasks. Some machines operate with a shared memory environment where each processor has direct access to a common memory. Others have a distributed memory associated with individual processors and, for a processor to gain access to data held in another processor's memory, these processors must communicate directly.

This report is concerned with an application for solving partial differential equations on small transputer systems. These systems have a coarse grained, distributed memory architecture. For an application to be suited to this type of system, there must be at least a coarse degree of concurrency inherent in the algorithm.

Domain decomposition methods present a means of imposing coarse concurrency on existing sequential PDE solvers. In these methods, the basic idea is to break up the domain into sub-domains, solve each sub-domain problem separately, and match the local solutions together. On a parallel computer, the sub-domain problems can be distributed to the different processors and solved simultaneously.

In this report, the application of one particular domain decomposition method on a two processor transputer system is described. Numerical experiments are conducted on two example problems and timing values are obtained. A restriction on the time step for reasonable accuracy with this method is also identified.

2 Brief Description of Transputer System

2.1 Hardware

A transputer is a single chip microprocessor designed as a building block for parallel processors. To facilitate this, each transputer has both memory and four links (which allow it to be connected to other transputers to form a system). The four links allow many inter-connection topologies to be implemented.

The transputer system used for the work in this report consists of five T800 transputers which are linked in an open-ended pipeline topology (Figure 1), so each transputer can only communicate directly with its two nearest neighbours. Note that an open-ended n-stage pipeline network has $(2n + 1)$ free (or unused) links and therefore does not make optimum use of the transputer communication facilities. The host system is a PC.

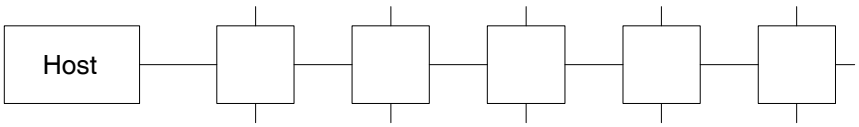

Figure 1: Open-ended Pipeline Topology

2.2 Software

Transputers are designed to implement the parallel programming language OCCAM very efficiently. However, due to the large amount of financial (and man-programming-hour) resources residing in FORTRAN code, parallel FORTRAN packages are also available for transputers. 3L Parallel FORTRAN, [9], is installed on the Reading system and all the parallel programming used in the work described in this report has been carried out using this package. There are two types of software parallelisation available in 3L Parallel FORTRAN, tasks and threads - the most important of which is the task.

A task is a self-contained FORTRAN program. Task structure is static (i.e. tasks are not created or destroyed dynamically during execution) and no hier-

archy exists (i.e. there is no facility for 'sub-tasks' within tasks). A complete application is viewed as a collection of one or more tasks. There is no memory sharing between tasks. They communicate with each other via channels which are one-way communication paths with fixed starting and finishing points. The task structure has the advantage that it is easy to adapt the application to execute on any number of transputers (up to the number of tasks in the application) since the distribution of tasks on the available transputers is controlled by a single piece of software - the configurer - which in turn is controlled by a small user-generated input file - the configuration file.

A task may contain several concurrent threads - a thread is a FORTRAN subroutine. These are created and destroyed during execution and are potentially hierarchical (i.e. a thread may be created from within another thread). Threads have shared common blocks. For programming ease, only the tasks structure was used in the programming of the applications in this report.

2.3 Considerations for Algorithm Design

An important feature of transputers is that communication between them is very much slower than communication within a particular transputer, so communication between transputers should be kept to a minimum (i.e. each transputer should do a small amount of communicating and a large amount of processing). Also, since transputers have only local memory and, in a pipeline network, they may only communicate directly with their two nearest neighbours, then data required by a transputer from a non-adjacent transputer must be passed along the chain through the intermediate transputers - this type of communication is very inefficient and so should be avoided. This is not a problem in the applications described in this report since only two tasks (and hence only two processors at most) are used.

2.4 Timing of Applications

The mechanism used for obtaining the timing values for the execution of the applications during this report is the low priority clock on the transputers combined with the TIMER utility in the 3L Parallel FORTRAN package. The low priority clock cycles at 15625 ticks/second and has a wrap round period of 2^{32} ticks (just over three days).

3 Domain Decomposition

3.1 Domain Decomposition Ideas

Domain decomposition methods for the numerical solution of partial differential equations involve dividing the region in which the equations are to be solved into (possibly overlapping) sub-regions. When the problem on the whole region is discretised (e.g. by the finite difference method or the finite element method), the interior unknowns of one sub-region are not directly coupled to those of another sub-region - they are only coupled via the variables in the overlap region. The number of overlap variables is typically small compared to the overall number of unknowns. By careful treatment of these overlap variables, the problems on the sub-regions can become completely decoupled - hence these problems can be solved simultaneously on separate processors of a parallel computer.

From [4], domain decomposition methods can also have benefits outside a parallel computing environment:

- The decomposition of a problem into sub-problems is advantageous when solving very large problems on a machine with limited storage.
- Some very efficient numerical techniques exist (e.g. discrete Fourier methods), although many of these can only be applied to problems on domains with regular geometries. The domain decomposition method can be used to divide a problem on an irregular region into sub-problems on regular subdomains (in this application, the domain decomposition method is similar to the capacitance matrix technique [2]).
- When generating a numerical grid to discretise a geometrically complex domain in two or three dimensions, the whole region can be split into distinct sub-domains on which it is easier to apply algebraic or PDE grid generation techniques [3]. This is the motivation behind the multiblock domain decomposition philosophy currently in favour in the aeronautics industry [1].
- In many practical problems, the nature of the governing equations changes throughout the domain or the equations have different parameters in different regions, in which case the idea of sub-dividing the region to allow different solution techniques to be more easily implemented in the different regions comes very naturally.

Only the parallel implementation of one particular domain decomposition method is discussed in this report - these other benefits are not investigated.

3.2 Common Domain Decomposition Methods

Many domain decomposition approaches for the solution of boundary value problems have some iterative feature. In general these are based on work originally done by Schwarz [8] who proposed solving the Poisson equation,

$$\nabla^2 u = f \quad \text{in} \quad \Omega \quad , \quad u = g \quad \text{on} \quad \partial\Omega$$

by dividing Ω into two overlapping sub-domains, Ω_1 and Ω_2 (Figure 2), and

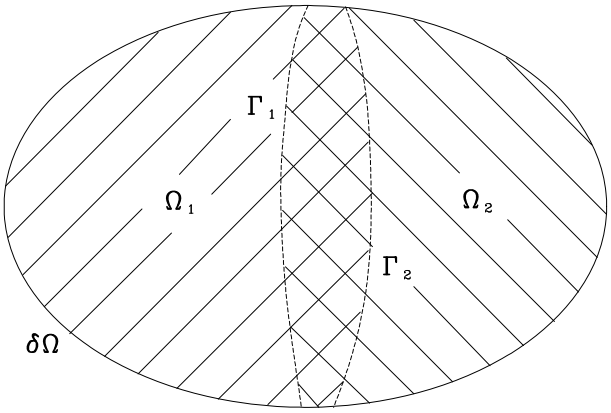

Figure 2: Schwarz Decomposition

solving the two sub-domain problems. The solution on each sub-domain is used to update the overlap boundary data for the adjacent domain. For the decomposition shown in Figure 2, an initial guess, α, may be specified for the solution on Γ_1. The problem on Ω_1 is then,

$$\nabla^2 u_1 = f \quad \text{in} \quad \Omega_1$$

with boundary conditions,

$$u_1 = g \quad \text{on} \quad \partial\Omega_1 \cap \partial\Omega$$

$$u_1 = \alpha \quad \text{on} \quad \Gamma_1$$

The solution u_1^0 on Ω_1 provides boundary data β on Γ_2 for the solution of a similar problem on Ω_2 which in turn leads to an improved iterate on Γ_1 - the procedure can then be repeated. Convergence of this method can be proven by the maximum principle. (This procedure is easily parallelised by providing initial starting iterates on Γ_1 and Γ_2 simultaneously and solving concurrently.)

Other decomposition methods involve condensing out a system of equations for the overlap variables from the complete discrete system by forming the Schur

complement of the matrix for the complete problem - this involves forming and solving a relatively small but dense matrix system for the overlap variables (as opposed to the sparse matrix system generally generated by the discretisation of the PDE) and then using the results of this as boundary conditions for the (now distinct) sub-regions.

The domain decomposition approach used in this report [7] involves no iteration and there are no dense matrix systems to generate, store and solve for the overlap variables.

3.3 A Direct Domain Decomposition Method

To introduce the domain decomposition method from [7], consider the discretised form of a partial differential equation (which applies in a region Ω), given by a system of linear equations for the nodal unknowns,

$$A\mathbf{u} = \mathbf{f} \tag{1}$$

The region Ω can be partitioned into two distinct sub-regions, Ω_1 and Ω_2, as shown in Figure 3, with

$$A\mathbf{u}_1 = \mathbf{f}_1 \quad \text{and} \quad A\mathbf{u}_2 = \mathbf{f}_2 \tag{2}$$

where

$$\mathbf{f}_1 = \begin{cases} \mathbf{f} & \in \Omega_1 \\ \mathbf{0} & \in \Omega_2 \end{cases}$$

and

$$\mathbf{f}_2 = \begin{cases} \mathbf{0} & \in \Omega_1 \\ \mathbf{f} & \in \Omega_2 \end{cases}$$

Note that Ω could have been partitioned into any number of distinct subdomains but the case with two sub-regions will serve to illustrate the method.

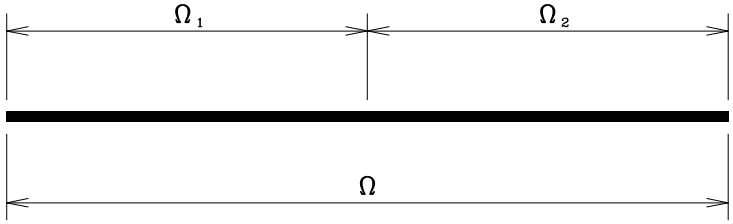

Figure 3: Non-overlapping Partitioning with Two Domains

Since the systems are linear, then by super-position the solution to (1) is

$$\mathbf{u} = \mathbf{u}_1 + \mathbf{u}_2$$

This method for finding the solution to (1) involves solving two systems each the size of the original system, so there appears to be no benefit in this approach - even when implemented on a parallel machine. However for problems which, when discretised, give rise to strictly diagonally dominant matrices, the solution to

$$A\mathbf{u}_i = \mathbf{f}_i$$

decays to zero outside Ω_i away from the partitioning boundary. For example, the numerical solution of a parabolic problem, the 1-D heat equation,

$$u_t = u_{xx} \quad \text{on} \quad \Omega = [0,1] \quad , \quad t \geq 0$$

with initial condition,

$$u(x,0) = \sin\left(\frac{\pi x}{2}\right)$$

and boundary conditions,

$$u(0,t) = 0 \quad , \quad u(1,t) = 1$$

using two distinct domains,

$$\Omega_1 = [0, \tfrac{1}{2}] \quad , \quad \Omega_2 = [\tfrac{1}{2}, 1]$$

by a Crank-Nicolson finite difference method with three point central spatial discretisation and $\Delta x = \Delta t = 10^{-3}$ after one time step is shown in Figure 4.

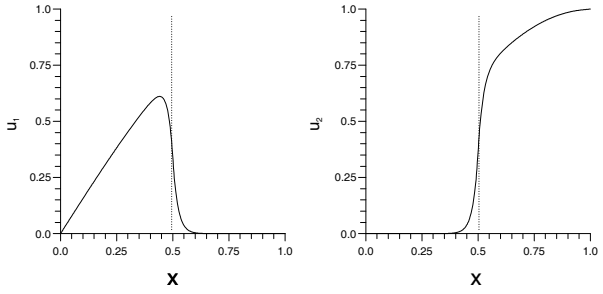

Figure 4: Decay of Solution away from Partitioning Boundary

This figure demonstrates the decay of the solution to the sub-problems (2) away from the partitioning boundary for this particular example, i.e. u_1 decays

to zero rapidly in Ω_2 (left plot) as does u_2 in Ω_1 (right plot). Therefore, if an artificial internal boundary with homogeneous Dirichlet boundary condition is placed somewhere in Ω, far enough away from Ω_i for the exact solution to be in some sense 'small' there, then the solution can be obtained as the sum of the solutions on the sub-regions. The problem is now being solved on two overlapping sub-regions - the notation for this partitioning is shown in Figure 5.

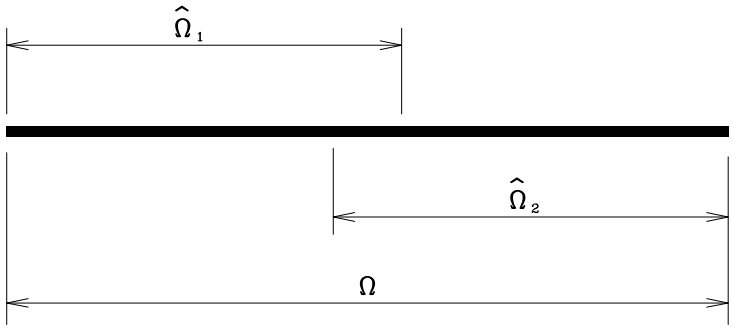

Figure 5: Overlapping Partitioning with Two Domains

Formalising this, let $\hat{A}_1$ and $\hat{A}_2$ be the matrices arising from the discretisation of the problem in two overlapping sub-regions, $\hat{\Omega}_1$ and $\hat{\Omega}_2$, and let $\hat{\mathbf{f}}_1$ and $\hat{\mathbf{f}}_2$ be the restrictions of $\mathbf{f}_1$ and $\mathbf{f}_2$ to $\hat{\Omega}_1$ and $\hat{\Omega}_2$, i.e.

$$\hat{\mathbf{f}}_1 = \mathbf{f}_1 \in \hat{\Omega}_1 \quad , \quad \hat{\mathbf{f}}_2 = \mathbf{f}_2 \in \hat{\Omega}_2$$

After solving the systems,

$$\hat{A}_1 \hat{\mathbf{u}}_1 = \hat{\mathbf{f}}_1 \quad , \quad \hat{A}_2 \hat{\mathbf{u}}_2 = \hat{\mathbf{f}}_2 \tag{3}$$

let $\mathbf{u}_i$ be the prolongation of $\hat{\mathbf{u}}_i$ to Ω obtained by augmenting it with $\mathbf{0}$ outside $\hat{\Omega}_i$, i.e.

$$\mathbf{u}_1 = \begin{cases} \hat{\mathbf{u}}_1 & \in \hat{\Omega}_1 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

and

$$\mathbf{u}_2 = \begin{cases} \hat{\mathbf{u}}_2 & \in \hat{\Omega}_2 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

If the size of the overlap region has been chosen so that the solution on $\hat{\Omega}_i$ is the same (to some prescribed tolerance) as the solution on Ω using distinct domains, then the solution to (1) is given by

$$\mathbf{u} \approx \mathbf{u}_1 + \mathbf{u}_2$$

When applying this domain decomposition method numerically, the aim is to make the error introduced by the domain decomposition of the same order as the

error caused by the discretisation of the problem.

The class of problems which give rise to matrices which are strictly diagonally dominant is not restricted to parabolic problems. For example, the elliptic problem known as the Helmholtz equation,

$$\nabla^2 u - cu = f \qquad (c > 0)$$

can also have this property.

Note that no iteration is required in this method and both the matrices in (3) are as sparse as the original matrix in (1). If the overlapping region is small compared to the size of the whole region then the computational work done in solving the two sub-systems is not very much more than that used to solve the original equation. The sub-systems can be generated and solved simultaneously on separate processors in a parallel computer.

4 Numerical Experiments

4.1 Test Problems

The test problem is the constant coefficient, two-dimensional heat equation,

$$u_t - \nabla^2 u = f(x,y,t) \quad , \quad t \geq 0 \tag{4}$$

on the unit square,

$$\Omega = \{0 \leq x \leq 1 \ , \ 0 \leq y \leq 1\}$$

with initial condition,

$$u(x,y,0) = u_i(x,y)$$

and Dirichlet boundary conditions,

$$u(x,y,t) \ \text{ given on } \ \partial\Omega$$

Problem A :

$$f(x,y,t) = (2\pi^2 - \alpha)e^{-\alpha t}\sin(\pi x)\sin(\pi y)$$

$$u(x,y,t) = 0 \ \text{ on } \ \partial\Omega$$

$$u_i(x,y) = \sin(\pi x)\sin(\pi y)$$

This has exact solution,

$$u(x,y,t) = e^{-\alpha t}\sin(\pi x)\sin(\pi y)$$

Problem B :

$$f(x,y,t) = \left(\frac{5\pi^2}{4} - \alpha\right) e^{-\alpha t}\sin\left(\frac{\pi x}{2}\right)\sin(\pi y)$$

$$u(x,y,t) = 0 \ \text{ on } \ \begin{cases} x = 0 \\ y = 0, 1 \end{cases}$$

$$u(x,y,t) = e^{-\alpha t}\sin(\pi y) \ \text{ on } \ x = 1$$

$$u_i(x,y) = \sin\left(\frac{\pi x}{2}\right)\sin(\pi y)$$

This has exact solution,

$$u(x,y,t) = e^{-\alpha t}\sin\left(\frac{\pi x}{2}\right)\sin(\pi y)$$

4.2 Discretisation of Test Problem

A Crank-Nicolson finite difference scheme on an regular Cartesian mesh with a five point spatial stencil is used to discretise (4), which then becomes,

$$\begin{aligned}
&\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} \\
&\quad - \left\{ \frac{1}{2} \left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{(\Delta x)^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{(\Delta y)^2} \right) \right. \\
&\quad \left. + \frac{1}{2} \left(\frac{u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}}{(\Delta x)^2} + \frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{(\Delta y)^2} \right) \right\} \\
&\qquad = \frac{1}{2} \left(f_{i,j}^{n} + f_{i,j}^{n+1} \right) \qquad (5)
\end{aligned}$$

where Δx is the mesh size in the x-direction
Δy is the mesh size in the y-direction
Δt is the time step
$u_{i,j}^{n} \approx u(i\Delta x, j\Delta y, n\Delta t)$
and $f_{i,j}^{n} = f(i\Delta x, j\Delta y, n\Delta t)$

The Crank-Nicolson scheme is popular because of its second order accuracy both spatially and temporally, i.e. the truncation error, $\tau_{i,j}^{n}$, is given by

$$\tau_{i,j}^{n} = \mathrm{O}(\Delta x)^2 + \mathrm{O}(\Delta y)^2 + \mathrm{O}(\Delta t)^2$$

and also because it is unconditionally stable, i.e. no restriction need be placed on the size of the time step used to ensure that the numerical solution is bounded. Rearranging (5) by grouping the values at the current time step on the right hand side gives

$$u_{i,j}^{n+1} + \Phi_{i,j}^{n+1} = u_{i,j}^{n} - \Phi_{i,j}^{n} + \frac{\Delta t}{2}(f_{i,j}^{n} + f_{i,j}^{n+1})$$

where

$$\Phi_{i,j}^{n} = -\nu_y u_{i,j-1}^{n} - \nu_x u_{i-1,j}^{n} + 2(\nu_x + \nu_y) u_{i,j}^{n} - \nu_x u_{i+1,j}^{n} - \nu_y u_{i,j+1}^{n}$$

$$\nu_x = \frac{\Delta t}{2(\Delta x)^2} \quad \text{and} \quad \nu_y = \frac{\Delta t}{2(\Delta y)^2}$$

If natural ordering is used to number the interior (i.e. non-boundary) nodes in the region with the increment being faster in the y-direction, then the resulting matrix system for the interior unknowns is

$$A\mathbf{u} = \mathbf{f} \qquad (6)$$

where A is a symmetric, strictly diagonally dominant, penta-diagonal matrix of band-width N_y (and N_y is the number of interior nodes in the y-direction).

4.3 Details of Application of Method

A two domain overlapping partitioning (Figure 5) is used with the overlap region being,

$$(\hat{\Omega}_1 \cap \hat{\Omega}_2) = \left\{ \tfrac{1}{2} - \gamma \leq x \leq \tfrac{1}{2} + \gamma, 0 \leq y \leq 1 \right\} \quad , \quad \gamma > 0$$

i.e. the overlap is symmetric about $x = \frac{1}{2}$.

On the transputer system, only the overlap variables are passed between the transputers, fulfilling the objective of low communication for a distributed memory machine if the overlap region is not large relative to the size of the whole region.

The matrix systems (6) resulting from the discretisation are large, sparse, symmetric and strictly diagonally dominant. These are solved using the preconditioned conjugate gradient (CG) method [6], with the preconditioner being the diagonal entries of the matrices. At each time step the previous non-updated solution on each domain is used as the initial estimate in the CG iteration on that domain; in the notation of (3) convergence is taken to have occurred when

$$\frac{\|\hat{A}_i \hat{\mathbf{u}}_i - \hat{\mathbf{f}}_i\|_2}{\|\hat{\mathbf{f}}_i\|_2} \leq 10^{-7}$$

In the numerical tests in the remainder of this section, the mesh size is taken as four times smaller in the x-direction than in the y-direction (this extra fineness in the x-direction is used to ensure that the decay of the solution in the overlap region on the two sub-regions is resolved). The size of the overlap is taken to be a quarter of the size of the whole region (i.e. $\gamma = \frac{1}{8}$).

Since the truncation error of the discretisation is second order both spatially and temporally, then the time step is taken to be of the same order as the mesh size. i.e. $\Delta t = \mathrm{O}(\Delta x)\{= \mathrm{O}(\Delta y)\}$. Various problem sizes (i.e. various numbers of nodes in the discretisation) are used and the value of the exponential decay parameter (α) in the test problems is adjusted so that the solution is half its original size after 10 time steps.

Figures 6 to 11 which follow show the solutions by the direct domain decomposition method for a 1600 node case in both Problems A and B after $0, 5$, and 10 time steps. These figures show the expected form of the complete solution (i.e. the original shape decays uniformly). The rapid decay of the solutions in the overlap region on each domain region is also clearly shown.

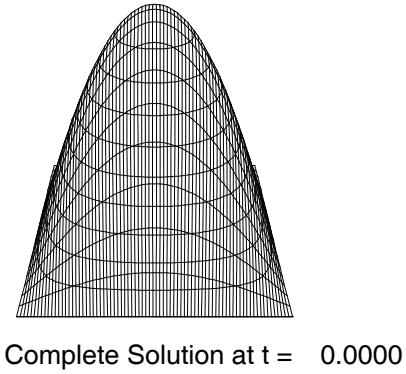

Figure 6: Initial Data for Problem A (1600 nodes)

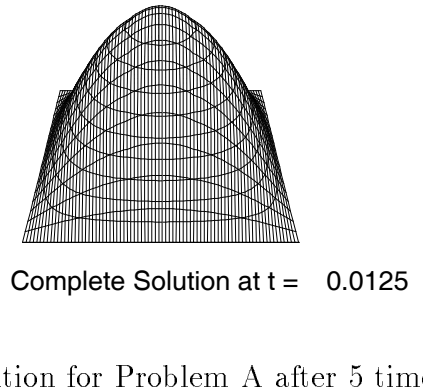

Figure 7: Solution for Problem A after 5 time steps

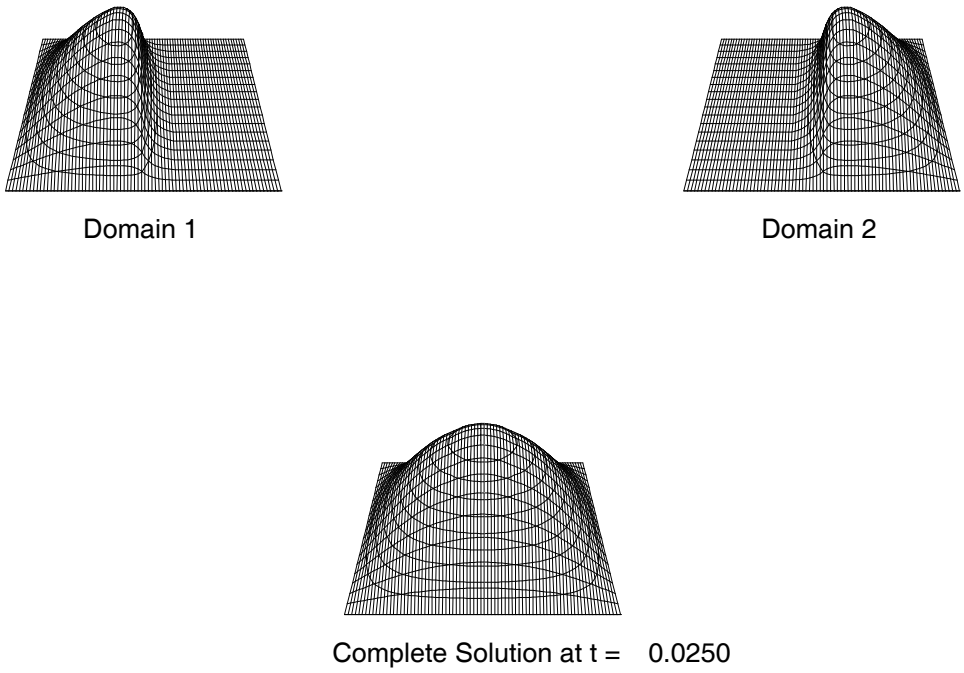

Figure 8: Solution for Problem A after 10 time steps

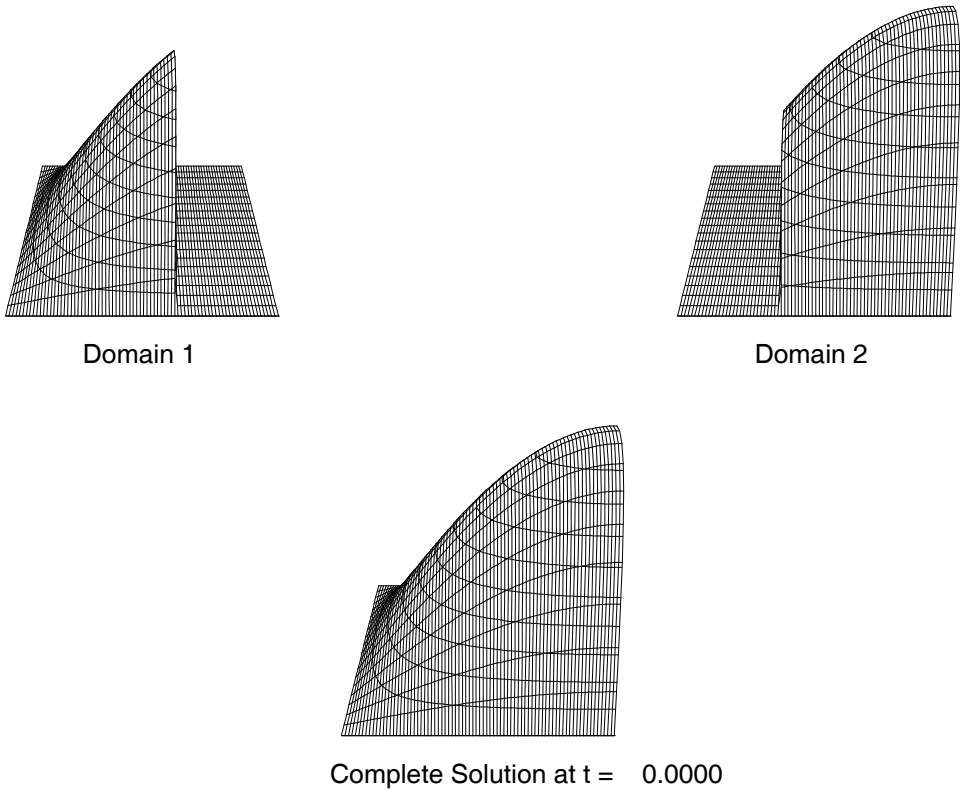

Figure 9: Initial Data for Problem B (1600 nodes)

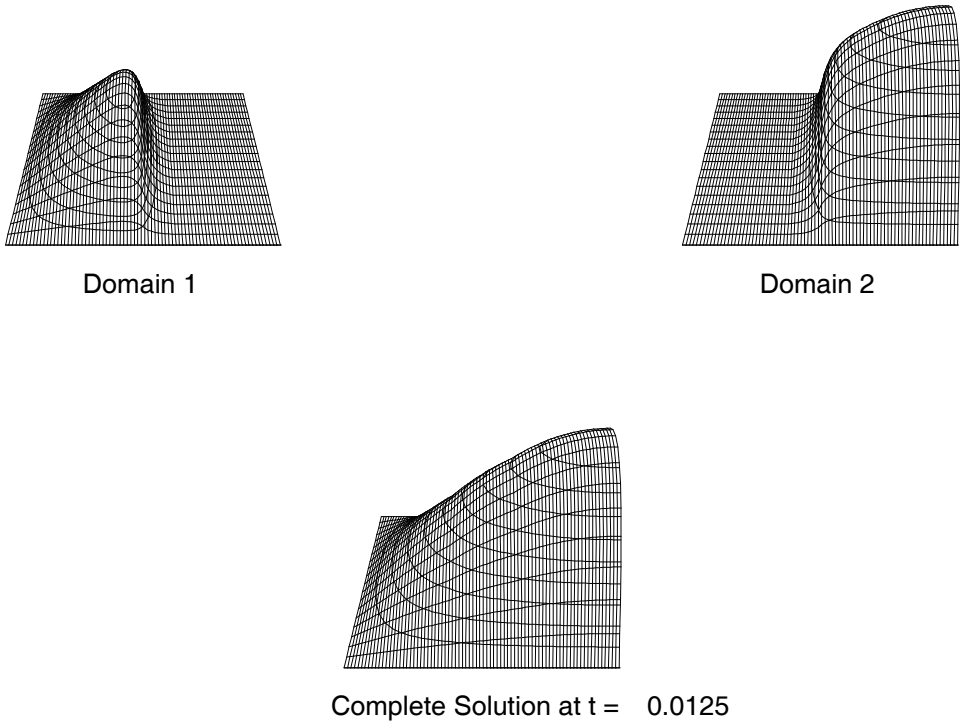

Figure 10: Solution for Problem B after 5 time steps

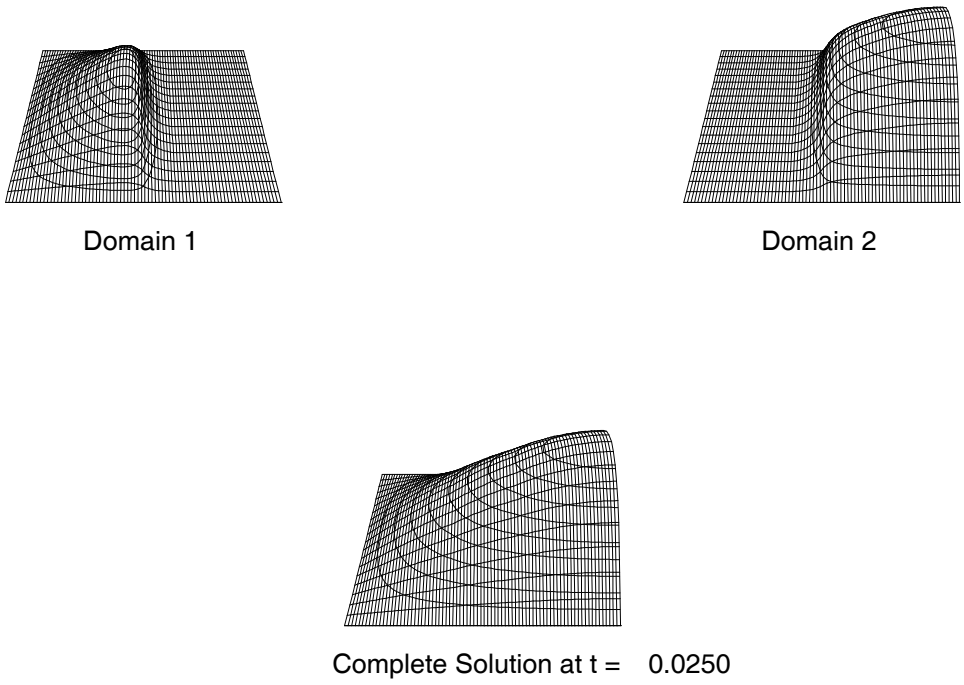

Figure 11: Solution for Problem B after 10 time steps

4.4 Parallel Implementation on Transputer System

Three different solution approaches are used on the transputer system :

1. Solution without domain decomposition on a single transputer (CN). The problem is coded as a single task (i.e. self-contained program), spatially discretised by a single mesh over the whole region and the resulting matrix system is solved by CG iteration at each time step.
2. Solution on two domains on a single transputer (DD_1). The problem is coded as a single task and the complete region is divided into two overlapping domains, each of which is discretised and the resulting matrix systems at each time step are solved separately and then the overlap region in both domains is updated using the solution from the other domain.
3. Solution on two domains on two transputers (DD_2). The problem is coded as two tasks, each of which is dedicated to solving the problem on a single sub-domain and communicates the overlap variables to the other for updating at the end of every time step.

In order to obtain the timing values for a single time step, the applications are run several times for 10 time steps, then the average elapsed time is divided by the number of time steps. The timing values for the three approaches on Problems A and B are shown in Tables 1 and 2 respectively.

Problem Size	Solution Technique		
	CN	DD_1	DD_2
400	0.443	1.270	0.641
800	1.056	3.295	1.663
1600	2.667	8.409	4.234
3200	6.422	20.271	10.215
6400	15.872	48.875	24.615

Table 1: Execution time (in seconds) per time step for Problem A

From Table 1, it can be seen that for Problem A, the solution without domain decomposition (CN) is faster than the domain decomposition method, even when this is implemented on two transputers (DD_2). However assuming that the communication overheads do not increase too much, these timing values suggest that if the region were to be divided into four overlapping domains (instead of the current two) and four transputers were used to solve the resulting matrix systems

Problem Size	Solution Technique		
	CN	DD_1	DD_2
400	1.270	1.305	0.644
800	3.023	3.433	1.673
1600	7.428	8.736	4.235
3200	17.628	21.294	10.360
6400	42.287	51.448	25.076

Table 2: Execution time (in seconds) per time step for Problem B

simultaneously, then the execution times with the domain decomposition method would be comparable with those without domain decomposition. This has not been attempted because the existing pipeline topology of the available transputer system makes the implementation of applications on more than two transputers unwieldy. It is planned that the hardware configuration of the transputers will be adjusted in future to overcome this problem. Table 2 shows that the solution on two transputers is faster than the solution without domain decomposition for Problem B. Note that the execution time for DD_2 is approximately half the execution time for DD_1 for both Problems A and B and all problem sizes, which shows that the application is reasonably well balanced on the transputer system - the domain decomposition method has been correctly parallelised.

If η is the ratio of the execution time of the two-transputer application to the method with no domain decomposition, i.e.

$$\eta = \frac{\text{Execution time for } DD_2}{\text{Execution time for CN}}$$

then the values of η for Problems A and B are given in Table 3.

Problem Size	η	
	Problem A	Problem B
400	1.447	0.507
800	1.575	0.553
1600	1.588	0.570
3200	1.591	0.588
6400	1.551	0.593

Table 3: Execution Time Ratio for Parallel Implementation

Generally, η increases with problem size showing that the increase in computational work is outweighed by the increase in communication (of the overlap variables) between the transputers at each time step in DD_2, resulting in the performance of DD_2 decreasing relative to CN.

In terms of execution time, the domain decomposition method is less successful on Problem A than on Problem B. This is because, in Problem A, the number of CG iterations used per time step to solve the discretised equations on each sub-domain is about 3-4 times greater than the number for the complete region (Table 4). Even though the discrete problems on the sub-domains are smaller, this manifests itself in DD_1 being about three times slower than CN (Table 1). For Problem B, a comparable number of CG iterations is needed on each sub-domain as on the complete region (Table 5), so the domain decomposition is more effective.

Problem Size	Region		
	Domain 1	Domain 2	Complete Region
400	23	23	8
800	30	30	10
1600	37	37	13
3200	45	45	16
6400	54	54	21

Table 4: Number of CG iterations per time step for Problem A

Problem Size	Region		
	Domain 1	Domain 2	Complete Region
400	23	25	29
800	30	33	35
1600	37	41	42
3200	46	50	50
6400	55	60	60

Table 5: Number of CG iterations per time step for Problem B

The difference in the ratio of the number of CG iterations needed to solve on the whole region and the number needed to solve on the sub-domains for the two problems is caused by a special property of Problem A combined with the initial iterate used at each time step in the CG iteration. Since the domain is a unit

square, then the homogeneous Dirichlet boundary condition in Problem A causes the implicit part of the five-point discrete differential operator (i.e. the matrix A in (6)) to have a fundamental eigenvector,

$$\{\mathbf{e}_1\}_{i+(j-1)N_x} = \sin(i\pi\Delta x)\sin(j\pi\Delta y)$$

where

$$1 \leq i \leq N_x \qquad \text{and} \qquad 1 \leq j \leq N_y$$

This fundamental eigenvector is the same as the spatial components of both the solution to the discrete problem and the right hand side of the discrete differential equation.

From [6], at the k^{th} iteration, the CG method finds an approximation, $\tilde{\mathbf{u}}_k$, to the solution, $\mathbf{u}$, of the matrix system (6) by minimising the functional,

$$\phi(\tilde{\mathbf{u}}_k) = \tfrac{1}{2}\tilde{\mathbf{u}}_k^T A\tilde{\mathbf{u}}_k - \tilde{\mathbf{u}}_k^T\mathbf{f}$$

over the Krylov space spanned by,

$$\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\}$$

where $\mathbf{r}_k$ is the residual vector defined by,

$$\mathbf{r}_k = A\tilde{\mathbf{u}}_k - \mathbf{f}$$

As already stated in Section 4.3, the initial iterate at each time step is taken as the solution from the previous time step. Hence for Problem A, $\tilde{\mathbf{u}}_0$ (and subsequently $\mathbf{r}_0$) also has the form of the eigenvector. So for the complete region in Problem A, the CG method looks for the solution in the set of vectors which spans the spatial component of the solution, which leads to extremely fast convergence. When the region is decomposed into sub-domains, this property is no longer true and the sub-problems require many more iterations for convergence.

This effect is observed even if the initial iterate is taken to be zero at all points in the region because $\mathbf{r}_0$ still has the form of the eigenvector. However, if $\tilde{\mathbf{u}}_0$ is taken to be unity at all interior points, then $\mathbf{r}_0$ no longer has the form of the eigenvector and the CG method converges at the same rate for the problem on the complete region as for the problems on the sub-domains.

Problem A is therefore an unfair test of the parallel performance of the domain decomposition method because of the fast convergence properties of the problem on the complete region.

4.5 Time Step Restriction

As already stated in Section 2, the error introduced by the domain decomposition should be of the same order as the error caused by the discretisation so that the decomposition does not corrupt the numerical solution. Figure 12 shows the ratio of the error in the discretisation (found by comparing the CN solution with the exact solution) and the error in the decomposition solution (again found by comparison with the exact solution) for all points in the region for both Problems A and B (and a problem size of 1600 nodes).

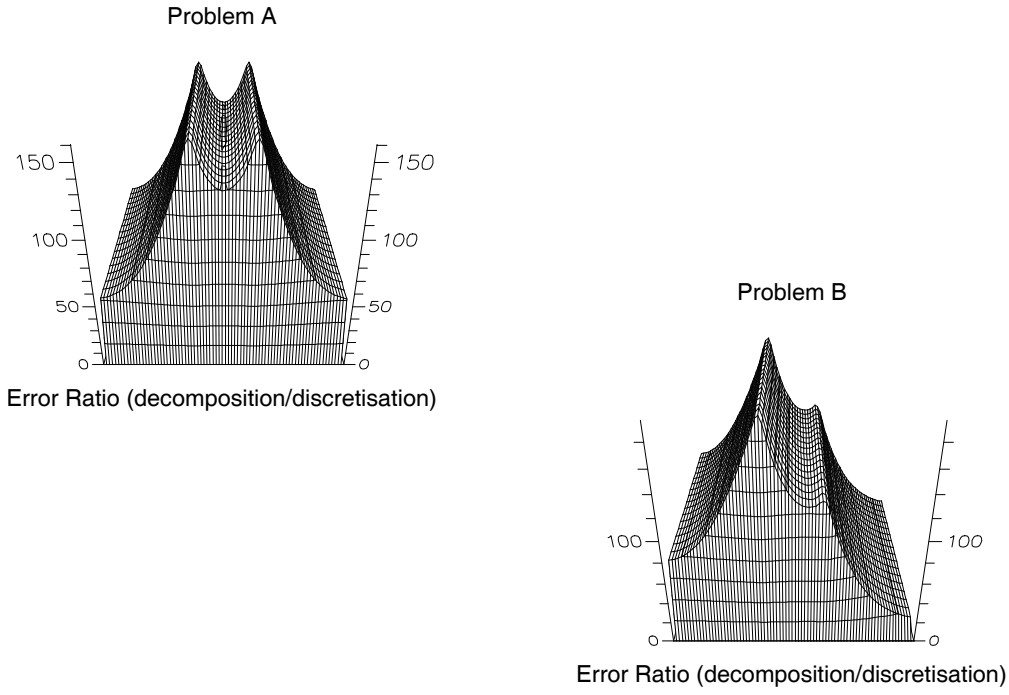

Figure 12: Ratio of Decomposition Error to Discretisation Error after 10 time steps

Table 6 shows the ratio of these errors for all the problem sizes. This ratio increases with problem size (apart from the 6400 node case where, since all the calculations were performed in single precision, the theoretical discretisation error is not achieved due to machine round-off errors), showing that the error caused by the decomposition decreases at a slower rate than the discretisation error.

As stated in Section 3.3, the feature of the discretised form of the PDEs considered which causes the decay of the solutions for the sub-problems in the overlap

Problem Size	$\left\|\frac{\text{Decomposition Error}}{\text{Discretisation Error}}\right\|_\infty$	
	Problem A	Problem B
400	38.4	42.1
800	70.3	80.2
1600	160.6	195.6
3200	433.6	416.1
6400	45.2	51.7

Table 6: Ratio of Decomposition Error to Discretisation Error

regions is the strict diagonal dominance of the resulting matrices. Demko [5] proved exponential decay for the inverses of banded, strictly diagonally dominant matrices in the form of the following proposition.

PROPOSITION : *Let $A = I - B$ be a band matrix with $b_{ij} = 0$ if $|i-j| > m$. Assuming that $\|B\|_q = r < 1$ for some $1 \leq q \leq \infty$, then the entries, α_{ij}, of A^{-1} satisfy*

$$|\alpha_{ij}| \leq \frac{r^{\frac{|i-j|}{m}}}{1-r} \quad , \quad where \quad r \leq \|B\|_q$$

The actual decay rate of the entries in the inverse for the matrices in Problems A and B will be faster than this result suggests because (i) the discretisation is such that the matrices are sparse (as well as being banded), and (ii) the entries on the out-riding diagonals are smaller than those on the super- and sub-diagonals. Neither of these additional matrix structures are taken in to account in the bound above. For the partitioning used in these numerical experiments, the decay rate required by the decomposition method is one such that the solution in the middle of the overlap is only weakly connected by an entry in the inverse to the solution at the edge of the overlap.

In general, m is determined by the mesh size and the distance of the inverse entry of interest from the diagonal is determined by both the mesh size and the size of the overlap. The required size of the entry is determined by the size of the RHS vector, as seen in Figure 12, where the error is larger at the end of the overlap from the 'larger' solution region in the asymmetric case (Problem B).

For a given size of overlap and mesh, the main influence on the decay is $\|B\|_q$. A value of $\|B\|_q$ within some tolerance, θ, is required for the connecting entry in the inverse to be sufficiently small. For the matrix system resulting from (5),

$$\|B\|_\infty = 4(\nu_x + \nu_y) = 2\Delta t \left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} \right) \leq \theta$$

Since $\Delta y = 4(\Delta x)$ for the meshes used in the examples, it is sufficient that the time step satisfies

$$\Delta t \leq \mathrm{O}(\Delta x)^2$$

so that the domain decomposition solution does not corrupt the underlying finite difference one. Figure 13 shows the same error ratio as Figure 12 but with a time step of $\Delta t = \mathrm{O}(\Delta x)^2$. From this it can be seen that the discretisation error is of the same size as the decomposition error when this time step restriction is imposed.

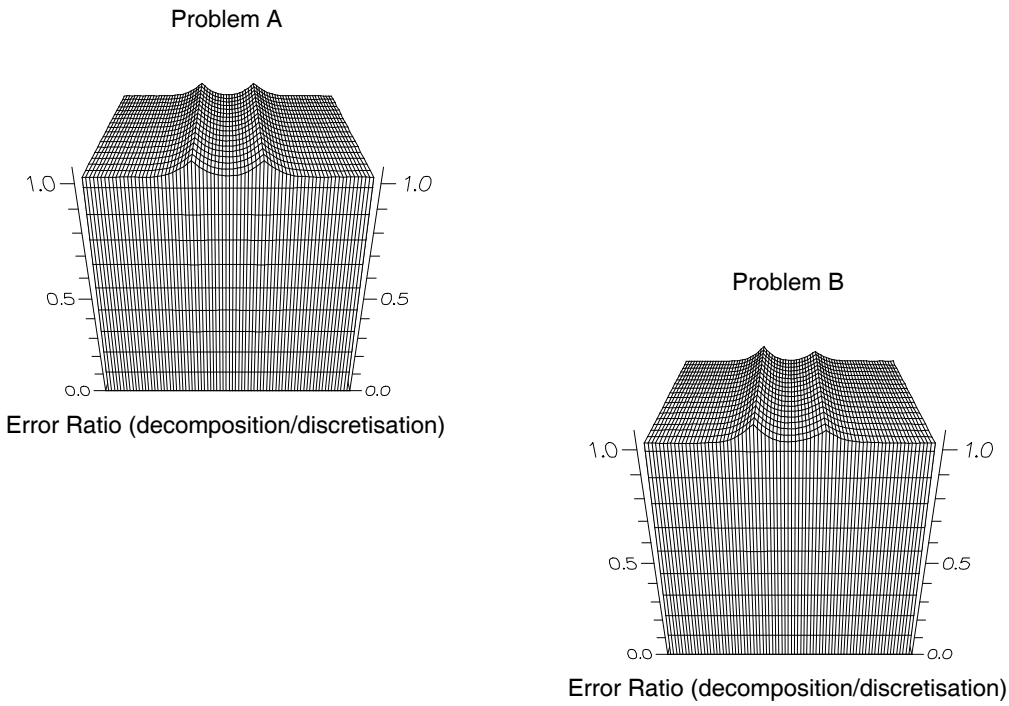

Figure 13: Ratio of Decomposition Error to Discretisation Error after 10 time steps

So, although the discretisation method is unconditionally stable and accuracy considerations imply only a first order time step restriction, error results from the

implementation of the domain decomposition indicate a second order time step restriction.

4.6 Position of Partition and Balancing of Application

Problem A is symmetric about the decomposition partitioning, hence the problem sizes are equal on the two sub-domains and the same number of CG iterations are required to solve the problems on these sub-domains (Table 4). When this problem is solved by DD_2, there is minimal latency time i.e. the time which one transputer spends idle during a time step while waiting for the other transputer to complete its computational task. DD_2 for Problem A is therefore a well balanced application.

Problem B is asymmetric about the decomposition partitioning and, although the problem sizes are equal on the two sub-domains, the solution on domain 2 requires more CG iterations (Table 5). Hence the transputer solving the problem on domain 1 has to idly wait during each time step until the transputer solving the problem on domain 2 has completed these extra iterations.

x co-ordinate of	Number of CG iterations		$\|$Decomposition Error/
centre of overlap	Domain 1	Domain 2	Discretisation Error$\|_\infty$
0.4	37	42	216.8
0.5	37	41	195.6
0.6	38	40	183.3

Table 7: Effect of Overlap Position

Table 7 shows the number of CG iterations required in both sub-domains for Problem B (with a problem size of 1600 nodes) for various overlap positions. From this table it can be seen that it is possible to decrease the latency time by moving the overlap from the centre ($x = 0.5$) in the positive x direction. This has the effect of increasing the size of the problem on domain 1 and decreasing it on domain 2, so the problem on domain 1 requires more work per iteration (and fewer iterations) than the problem on domain 2. A repositioning of the overlap of this form could be made to improve the execution time efficiency of DD_2 on Problem B still further but this avenue of investigation has not been pursued in detail in this report.

Table 7 also shows the maximum value of the error ratio described in the previous section for various overlap positions. This shows that an added advantage in the use of repositioning the overlap to decrease the latency time is that the error ratio can be decreased.

5 Conclusions

A direct domain decomposition method for the numerical solution of boundary value problems in which the discretisation leads to strictly diagonally dominant matrices has been presented.

This method has been used to decrease the execution time for test problems by allowing efficient implementation on a two-transputer parallel computer system. However, it has also been demonstrated that, when using this method, a second order time step restriction should be applied so that the error introduced by the decomposition does not seriously add to the underlying discretisation error. With this in mind, it may be preferable to use an explicit Euler temporal discretisation which requires less computational work since it does not need a (numerical) matrix inversion at each time step. The scheme is then only first order accurate in time but there is a second order time step restriction for stability.

The investigations carried out on the method in this report are by no means exhaustive and many issues are open to research, e.g. the 'optimal' size and position of the overlap for general problems and the effect of increasing the number of domains (and also the number of transputers).

6 Acknowledgements

K.J. Neylon is supported financially by a NERC CASE studentship with the collaborating body being the Institute of Hydrology, Wallingford. He wishes to thank Dr. M.J. Baines and Dr. N.K. Nichols, of the Mathematics Department, University of Reading, for their advice and assistance during this work.

All the parallel computations were carried out on a transputer system which is currently on loan to the Mathematics Department and Construction Management and Engineering Department, University of Reading, from the Rutherford Appleton Laboratory as part of the SERC/DTI Initiative in the Engineering Application of Transputers (loan ref. TR1/217).

References

[1] Aves, M.A., (1991)
"Multigrid Multiblock Computation of Steady Compressible Flows",
Ph.D. Thesis, University of Reading

[2] Buzbee, B.L., Dorr, F.W., George, J.A., and Golub, G.H., (1971)
"The Direct Solution of the Discrete Poisson Equation on Irregular Regions",
SIAM J. Numer. Anal., **8**, pp 722-736

[3] Carey, G.F., (1989)
"Parallel Sub-Domain and Element-by-Element Techniques",
In *Parallel Supercomputing : Methods, Algorithms and Applications*,
(Ed. G.F. Carey), Wiley, pp 57-75

[4] Chan, T.F., and Resasco, D.C., (1985)
"A Domain-Decomposed Fast Poisson Solver on a Rectangle",
Research Report YALEU/DCS/RR-409

[5] Demko, S., (1977)
"Inverses of Band Matrices and Local Convergence of Spline Projections",
SIAM J. Numer. Anal., **14**, pp 616-619

[6] Golub, G.H., and Van Loan, C.F., (1989)
"Matrix Computations", (2nd Edition),
John Hopkins University Press

[7] Meurant, G., (1990)
"A Domain Decomposition Method for Parabolic Equations",
To appear in a special issue of Applied Numerical Mathematics

[8] Schwarz, H.A., (1869)
"Über einige Abbildungsaufgaben",
Ges. Math. Abh., **11**, pp 65-83

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