PARALLELIZATION AND GRAPHICAL USER INTERFACE OF AUTO94

XIANJUN WANG

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

JULY 1994

ISBN 0-315-97615-2

Abstract

Parallelization and Graphical User Interface of AUTO94

Xianjun WANG

Certain parallel algorithms arising in the numerical analysis of bifurcation problems are studied in this thesis. In particular, two sparse linear solvers are investigated in detail. These two sparse solvers differ mainly in communication schemes. In addition, one of them uses a restricted row and column pivoting strategy, while the other does not. The condensation of parameters and its associated backsubstitution algorithms in the sparse solvers are almost fully parallelized. These mainly determine the parallel performance of the sparse solvers, particularly when the size of the problem is big. This thesis stresses the communication schemes of these algorithms. They have been implemented in AUTO94P, which currently runs on the Intel Delta at the California Institute of Technology. All numerical timing results in this thesis were obtained on the Delta. To simplify the use of AUTO94P and its sequential version AUTO94, a graphical user interface (GUI), based on the X window system, has been designed and implemented. The testing and verification of the GUI was done on various Silicon Graphics and Sun workstations. This thesis also documents the installation, functionality and extensibility of the GUI.

Acknowledgments

First of all, I would like to thank my supervisor, professor Eusebius J. Doedel, for his great patience and kindness in guiding and supporting my work for many years.

I would like to thank professors T. D. Bui, Lixin Tao and Tao Li for their advice.

Thanks to Mr. Quanlin Gu, Mr. Deming Li, Mr. Xinming Yu and Mr. Pankaj Kamthan for their helpful discussions and friendship during my many years at Concordia University.

Support for this work has come from many sources including the Centre de Recherche Informatique de Montréal and the Center for Research on Parallel Computing at the California Institute of Technology.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we first give an introduction to parallel programming, and then describe the thesis work and structure.

1.1 Parallel Programming

1.1.1 Basic Concepts

In essence, the task of the programmer is that *given a specification, develop a program that satisfies the specification. A parallel program* is simply a collection of cooperating programs that together satisfy a given specification[9, 10]. A sequential program can be thought as a "parallel program" without any cooperation. Unlike a parallel program, a sequential program executes its statements sequentially on a singlc processor. However, a parallel program executes its statements concurrently on many processors. A *multicomputer*[30, 69] is often referred to as a *message-passing concurrent computer.* Figure 1 shows a simple connection of a message-passing network. It has n computers, called *nodes*, connected by a communication network. Each of the n nodes executes programs concurrently, and coordinates its activities with the other nodes by message exchanging, namely, sending and receiving messages. *Distributed*

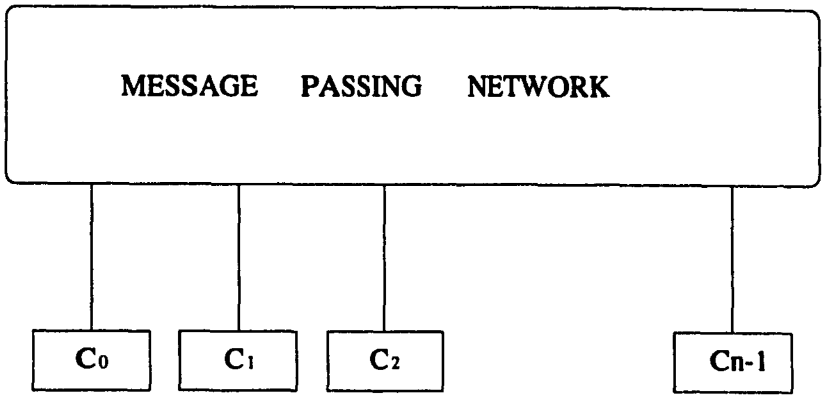


Figure 1: N Computing Nodes

memory systems, i.e., multicomputers, physically separate the memory for each node. Unlike *shared memory systems*, distributed memory systems have no global memory addresses, each node has its own local private memory which can not be accessed by other nodes. The aim of the parallel programming on distributed memory systems is to concurrently execute programs in each node efficiently. However, parallel programming is extremely hard, much harder than sequential programming[29]. This is because that it does not enforce deterministic execution. In addition, most message passing systems do not enforce information hiding and provide a global name space of processes. This makes it difficult to develop modular programs and reusable libraries. But parallel programming can be "easier" if one works in the domain of the problem to be solved, the specification domain, rather than on coding for some particular machine. Currently many universities and institutions are currently committed to this research direction, for example, the Center for Research on Parallel Computing (CRPC) [49], a consortium of Argonne National Laboratory, the California Institute of Technology, Los Alamos National Laboratory, Syracuse University, Rice University and the University of Tennessee. There are many research directions being focussed at CRPC, two of them are *Fortran parallel programming* such as Fortran M[29] and *Compositional programming* such as PCN[28] and compositional $C++$[8].

1.1.2 Overview of Numerical Parallel Computing

Numerical parallel computing is playing an increasingly important role in computer science and offers great promise for future progress of computer technology. Numerical computations in linear algebra and differential equations pose important problems in the field of parallel computing. Surveys of parallel algorithms for linear algebra computations and for solution of differential equations are given in [31, 43, 44] and in [60], respectively. In this section, we briefly outline the present status of a few numerical methods for differential equations on parallel computer. We restrict the overview to *direct methods*, in view of its direct relation to this thesis work.

Gauss Elimination: Gauss elimination is one of the frequently used methods for solving linear systems. Its complexity is $O(n^3)$, where n is the dimension of the matrix. Several early papers, for example [27, 52], considered in detail the implementation of Gauss elimination. An important aspect of the analysis in some of the above papers is the derivation of precise timing formulas which show the effect of the start-up times for vector operations. An introduction can be found in [44]. A detailed analysis of the computational complexity of factorization algorithms can be found in [51]. Recent results can be found in [16, 17].

Givens Reduction: The difficulties with implementing interchange strategies on parallel computer suggest that orthogonal reductions to triangular form may have advantages. It was first observed by Gentleman [35] that orthogonal reduction to triangular form by Givens or Householder transformations has a certain natural parallelism. The algorithm for the Givens reduction is given in detail in [68], where it is also shown that it is slightly more efficient in a parallel environment than Householder transformations.

Tridiagonal System: Tridiagonal systems are often solved by a *recursive doubling* method, a procedure which, in the simplest case, expresses the $2i$th element in a sequence in terms of the ith. Thus for $N = 2^k$, the Nth component can be computed in $logN$ steps. Various methods for solving triangular system can be found in [17, 18], and other related methods, such as block tridiagonal system, can be found in [48].

Nested Dissection: Nested dissection for vector computers was first discussed in [6] in the context of rather general rectangular finite elements, and estimates are given for the number of vector operations required for the factorization, assuming dissection is carried to completion. The appropriate level of dissection becomes an interesting question for a vector computer. Another result discussed in [39] deals with the general problem of how effectively an algorithm translates into vector operations. For parallel arrays, a careful analysis of nested dissection has been given in [32], where the author considers an MIMD array with nearest neighbor connections and assumes a processor for each node in the discretization. The algorithm uses a pipelined version of Givens rotations as a building block. It is shown that the nested dissection runs in $C(N + rlogN)$ time for r right hand sides. The constant C is fairly large and may result in the algorithm not being competitive with other methods for a single right hand side. Other nested dissection methods can be found in [57].

1.2 Description of the Thesis Work

In this section, we formulate an ordinary differential equation (ODE) computational problem that arises in the AUTO package [21]. A major part of this thesis concerns the design and implement of two efficient parallel algorithms to solve the ODE problem on distributed memory systems. We also introduce a network transparent graphical user interface (GUI) for AUTO based on the X window system. The outline of this thesis is given at the end of this section.

1.2.1 The AUTO Package

AUTO is a set of FORTRAN routines dealing with numerical analysis of bifurcation problems. It can be used as a tool for investigating problems arising in various fields such as applied mathematics, mechanical engineering, chemical engineering, biological science, etc. AUTO was first written in 1979. It was based on a related program written in 1976 at the California Institute of Technology. A first publication referring to the package by its current name appeared in [20]. At this time AUTO86 [21] is the most widely distributed version. A useful tutorial appeared in [22, 23]. AUTO94P, a new parallel version of AUTO developed by the author in cooperation with E. J. Doedel, is an experimental parallel program that currently runs on the Intel Delta system at the California Institute of Technology. It required significant changes in the linear equation solver. AUTO94 is the sequential version of AUTO94P. A graphical user interface was added to simplify the use of the package.

1.2.2 The Linearized System

We consider the first order system of ordinary differential equations described in [21].

$$\frac{du}{dt} = f(u, \lambda) \tag{1}$$

where $t \in [0,1], u \in R^n$ and $\lambda \in R^{n_\lambda}$, subject to boundary conditions

$$b_i(u_0, u_1, \lambda) = 0 \qquad i = 1, 2, \cdots, n_b \tag{2}$$

and integral constraints

$$\int_0^1 q_i(u, \lambda)dt = 0 \qquad i = 1, 2, \cdots, n_q \tag{3}$$

In order for the above problem to be well posed, it is necessary that $n_\lambda = n_b + n_q - n + 1$. In this case there will be one free parameter, so that the equations will normally yield curves of solution. Note that the boundary conditions are nonlinear, (see [64] for linear boundary conditions and see [1, 3, 5, 50, 54, 65] for related methods), i.e., we are dealing with a very general ODE problem. An efficient sequential numerical algorithm for solving the above ODE system can be found in [21]. The corresponding algorithm was implemented in AUTO86. More precisely, define a mesh

$$\{0 = t_0 < t_1 < \cdots < t_N = 1\}, \quad \Delta t_j \equiv t_{j+1} - t_j, \quad (0 \le j \le N - 1) \tag{4}$$

and for each j introduce the Lagrange basis polynomials

$$\{w_{j,i}(t)\} \quad j = 0, 1, 2, \cdots, N - 1 \quad i = 0, 1, 2, \cdots, m \tag{5}$$

defined by

$$w_{j,i}(t) = \prod_{k=0,k\neq i}^{m} \frac{t - t_{j+\frac{k}{m}}}{t_{j+\frac{i}{m}} - t_{j+\frac{k}{m}}} \quad t_{j+\frac{i}{m}} \equiv t_j + \frac{i}{m}\Delta t_j \tag{6}$$

The collocation method now consists of finding

$$p_j(t) = \sum_{i=0}^{m} w_{j,i}(t) u_{j+\frac{i}{m}} \tag{7}$$

such that

$$p_j'(z_{j,i}) = f(p_j(z_{j,i}), \lambda) \quad i = 1, 2, \cdots, m \quad j = 0, 1, 2, \cdots, N-1 \tag{8}$$

where in each subinterval $[t_{j-1}, t_j]$ the points $\{z_{j,i}\}_{i=1}^{m}$ are the zeros of the mth degree Legendre polymonials relative to that subinterval. With the above choice of basis, u_j and $u_{j+\frac{i}{m}}$ are to approximate the solution $u(t)$ of the continuous problem at t_j and $t_{j+\frac{i}{m}}$ respectively. The discrete boundary conditions are $b_i(p_1(0), p_N(1), \lambda) = 0, i = 1, 2, \cdots, n_b$, i.e.,

$$b_i(u_0, u_N, \lambda) = 0 \qquad i = 1, 2, \cdots, n_b \tag{9}$$

The integrals can be discretized by a quadrature formula. In view of the discretization of the differential equation (1), the natural choice is the composite quadrature formula obtained by approximate integration over each of the subintervals $[t_{j-1}, t_j]$. This gives

$$\sum_{j=0}^{N-1} \sum_{i=0}^{m} w_{j,i} q_k(u_{j+\frac{i}{m}}, \lambda) = 0 \qquad k = 1, 2, \cdots, n_q \tag{10}$$

where the quantities $w_{j,i}$ are the Lagrange quadrature coefficients. Apart from a scaling factor these are independent of j. Since pseudo-arclength continuation, see [22, 23, 47] for details, is used for the computation of branches of solutions to (1), we need to adjoin the equation

$$\theta_u^2 \int_0^1 (u(t) - u_0(t))^* \dot{u}_0(t) dt + \theta_\lambda^2 (\lambda - \lambda_0)^* \dot{\lambda}_0 - \Delta s = 0 \tag{11}$$

where (u_0, λ_0) is the previously computed point on the solution branch and $(\dot{u}_0, \dot{\lambda}_0)$ is the normalized direction of the branch at that point. Upon discretization the pseudo-arclength equation becomes

$$\theta_u^2 \sum_{j=0}^{N-1} \sum_{i=0}^{m} w_{j,i} (u_{j+\frac{i}{m}} - (u_0)_{j+\frac{i}{m}})^* (\dot{u}_0)_{j+\frac{i}{m}} + \theta_\lambda^2 (\lambda - \lambda_0)^* \dot{\lambda}_0 - \Delta s = 0 \tag{12}$$

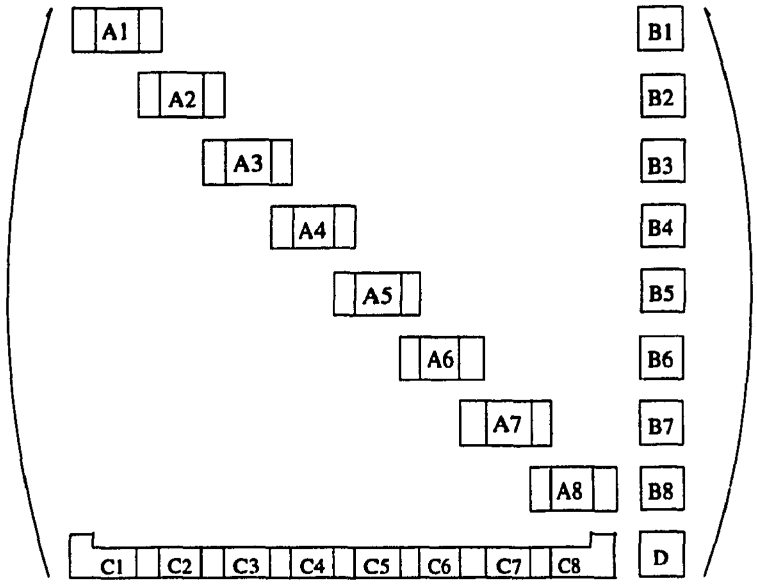


Figure 2: Structure of the Jacobian matrix J

The complete set of discrete equations for taking one step along a branch of solutions therefore consists of solving the system of $mnN + n_b + n_q + 1$ nonlinear equations (8)-(12) for the unknowns $\{u_{j+\frac{i}{m}}\} \in R^{mnN+n}, \lambda \in R^{n_\lambda}$. This is done by a Newton or Newton-Chord iteration. After linearization (Newton method) [21], a linearized sparse matrix J as shown in Figure 2 is obtained. The matrix J is structured and sparse with borders at the bottom and on the right. The corresponding linearized system has the form

$$Jx = f \tag{13}$$

The above linearized system (13) is solved in AUTO several times during each Newton step when computing solutions of ordinary differential equations. Moreover, the entire computation for a given problem can take many steps. When the problem size is big, solving the linearized system (13) becomes the dominant computation of the AUTO package. Practical results show that AUTO often spends more than 70% of its total computation on setting up and solving the linearized system. This percentage increases as the problem size increases. Thus efficiently solving this system is important. One of the main contribution of this thesis is to study and develop efficient parallel algorithms to solve the linearized system on distributed memory systems.

1.2.3 The Graphical User Interface of AUTO94

The purpose of the graphical user interface (GUI) is to simplify the use of AUTO. It provides the user with a convenient computational environment on a wide ranges of UNIX platforms. It is network transparent and has a three dimensional feeling. Figure 33 shows the general frame of the interface. The design of the GUI interface is based on the X window system [58] and the implementation is based on Motif [58, 59, 61, 62, 67]. A detailed description of the GUI is presented in Chapter 6.

1.2.4 Outline of the Thesis

As pointed out in section 1.2.2, efficiently solving the linearized system is important in the AUTO package. This thesis will pay particular attention to the study and development of parallel algorithms for solving the linearized system. Since the communication between processors on distributed memory systems is one of the key points in designing parallel algorithms, we will mainly stress the communication schemes of the parallel algorithms studied in the thesis. The purpose of studying these parallel algorithms was to develop AUTO94P, a new parallel version of AUTO. This thesis devotes three chapters, namely Chapter 2,3 and 4, on the design and implementation of the parallel algorithms. Chapter 2, which addresses dense matrix computation, serves as preliminary study for Chapters 3 and 4. Both Chapters 3 and 4 focus on the linear solver for the system (13). The major difference between these chapters is that Chapter 3 focusses on the linear solver without pivoting strategy and Chapter 4 with pivoting strategy. Both parallel algorithms of Chapters 3 and 4 are implemented in AUTO94P. Numerical timing results are included in Chapters 3 and 4. All numerical results were obtained on the Intel Touchstone Delta, 512 Intel iPSC/i860 nodes connected by mesh network, located at the California Institute of Technology. Chapter 5 describes some implementation issues of AUTO94P. A new sequential version of AUTO called AUTO94, was also developed in cooperation with E. J. Doedel. It is somewhat more convenient to use than AUTO86. In particular, it includes a graphical user interface , which is described in Chapter 6. Chapter 7 contains concluding remarks.

Chapter 2

Dense Matrix Computations

In this chapter, we focus on parallel dense matrix computations. The work in this chapter provides an introduction to the problems studied in Chapters 3 and 4. It includes a brief overview of dense matrix computations, outlines of both sequential and parallel LU-decomposition algorithms, and a description of parallel implementation of the algorithm. Numerical timing results for the parallel algorithm are also included.

2.1 Overview

Dense matrix computations are of such central importance that they are usually among the first algorithms implemented in any new computing environment. The need for high performance on common operations such as matrix multiplication and solving systems of linear equations has had a strong influence on the design of many architectures, compilers, etc., and such computations have become standard benchmarks for evaluating the performance of new computer systems. LU-decomposition is one of the important problems in dense matrix computations. Its complexity is $O(n^3)$, where n is the size of the problem or dimension of the matrix. The main focus in this chapter is to design a parallel implementation of the LU-decomposition algorithm. A survey of parallel algorithms for dense matrix computations is given in [31]. Notable successes in attaining very high performance can be found in [4].

2.2 LU-Decomposition Algorithm

The LU-decomposition algorithm is based on a reformulation of the classical result[16, 17, 18, 41]:

Theorem 2.2.1 *For any real $M \times N$ matrix A, there exists an $M \times M$ permutation matrix R and an $N \times N$ permutation matrix C such that*

$$RAC^T = L_1 U_1$$

where L_1 is $M \times M$ unit lower triangular and U_1 is $M \times N$ upper triangular.

From Theorem 2.2.1, we can obtain the following[16]:

Theorem 2.2.2 *For any real $M \times N$ matrix A, there exists an $M \times M$ permutation matrix R and an $N \times N$ permutation matrix C such that*

$$A = LC^T I_{N,M} RU$$

where RLC^T is lower triangular and RUC^T is upper triangular. Both L and R are $M \times N$ matrices. The matrix $I_{N,M}$ is $N \times M$ identity matrix.

The above LU-decomposition is used as the mathematical basis for the parallel LU-decomposition algorithms that we describe in the next section.

2.3 LU-Decomposition Algorithm for Distributed Memory Systems

Parallel implementation of the LU-decomposition algorithm can be categorized by the distribution of the coefficient matrix over the concurrent processes and by the pivoting strategy. Various combinations of row and column oriented distribution with pivoting

```
{ Initialize the permutation index array }
for $i = 0, 1, \cdots, m-1$ do { concurrently}
    $fr(i) = i$ { feasible row index}
    $fc(i) = i$ { feasible column index }
end
$fr_0 = 0$ {initial feasible row index}
$fc_0 = 0$ {initial feasible column index }
for $k = 0, 1, \cdots, n-1$ do { concurrently}
    { search for the local pivot $a_{lpiv}$ at step $k$ in each node }
    $a_{lpiv} = max(| a(fr(i), fc(j)) |)$    $i \in (fr_0, m-1)$    $j \in (fc_0, n-1)$
    { find the global pivot by recursive doubling procedure }
    $a_{gpiv} = max((a_{lpiv})_p)$    $p = 0, 1, \cdots, P-1$
    if $a_{gpiv} \in P_p$ then { i.e., $a_{gpiv}$ is in node p }
        $fr_0 = fr_0 + 1$ { in node $P_p$ only }
        $fc_0 = fc_0 + 1$ { in node $P_p$ only }
        send pivot row to all other nodes { broadcast pivot row }
        do Gauss elimination in node $P_p$
    else { nodes do not hold $a_{gpiv}$ at step $k$ }
        receive pivot row
        do Gauss elimination
    endif
end
```

Table 1: Parallel LU-Decomposition Algorithm with Complete Pivoting

can be found in [7, 11, 33, 34, 56]. Here we present a design and implementation of the LU-decomposition based on row oriented distribution with complete pivoting. The implementation is done on the Intel Touchstone Delta system. The goal is to develop an efficient linear algebra library for distributed memory system. Specific design criteria are that the library components must be easy to integrate into a larger user program and that the communication cost should be minimized. Assume a $n \times n$ matrix A is partitioned into $(A_0, A_2, \cdots, A_{P-1})$. For simplicity, assume that $A_p \in R^{m \times n}$, where $p = 0, 1, 2, \cdots, P-1$. The parallel LU-decomposition algorithm is outlined in Table 1.

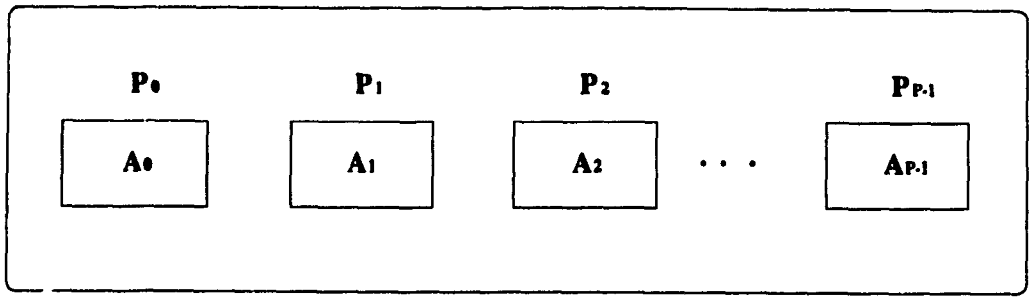


Figure 3: Data Distribution of Matrix A

2.4 Parallel Implementation

Here we describe data distributition, pivoting strategy, communication schemes and the backsubstitution process after LU-decomposition. These are the main factors that affect the performance of the parallel implementation.

2.4.1 Data Distribution

Our goal is to evenly distribute the data so that the task can be balanced over all processors. We use row oriented distribution because it represents a family of application problems. Assume the total number of processors is P, each processor is identified by a unique number between 0 and $P-1$. Partition the $n \times n$ matrix A as $(A_0, A_1, \cdots, A_{P-1})^T$, where $A_p, 0 \leq p \leq P-1$, is in processor p. This type of distribution is typical and useful. It can balance the loads over all processors and the best performance can be expected. Figure 3 shows the data distribution graphically.

2.4.2 Pivoting Strategy

In order to prevent numerical instability, pivoting is often needed. It is proven that only LU-decomposition with *complete pivoting* is numerically stable, i.e., for any matrix the computed LU-decomposition is the LU-decomposition of a nearby matrix. Complete pivoting searches all feasible entries of a matrix for the one that is largest in absolute value, namely, during each elimination step, the pivot element of a matrix A is obtained by searching all the feasible entries in the matrix. This strategy is dynamic because the decision which entry should be the pivot in the $k-th$ step of the decomposition is postponed until the pivot is actually needed. For the sake of efficiency, one can restrict the searching region to obtain *row pivoting* or *column pivoting*, but the drawback is that its the numerical stability is not as good as that of complete pivoting. In our parallel implementation, a complete pivoting strategy is used.

2.4.3 Communication Schemes

In the implementation, we organized all processors into a one dimensional grid. The origin and destination of messages are identified by the processor identification number together with the type of the message. The determination of the pivot is done by the *recursive doubling* procedure which will be described later in this thesis (see section 4.2.3). The sending and receiving of the pivot row at each elimination step is done by a global broadcast. The partial solutions obtained during the backsubsitution process are also sent to other processors via global broadcast.

2.4.4 Backsubstitution

To obtain the solution of a linear system after the LU-decomposition, backsubstitutions have to be performed. The complexity of the backsubstitution is $O(n^2)$ compared with $O(n^3)$ of the LU-decomposition. The parallel backsubstitution process needs communications between different processors, because at step k of the backsubstitution, we need to determine which processor should do the backsubstitution and broadcast the partial solutions at step k to all other nodes. The cost of the communication is minor, since the entire backsubstitution process is of order $O(n^2)$.

Number of Nodes	Execution time	Speed-up	Efficiency
1	15.22	1	100%
8	2.14	7.11	88.90%
16	1.59	9.57	59.83%

Table 2: $A = (a_{ij}) = (cos((i+1)(j+1))) \qquad i,j = 1,2,\cdots,256$

2.5 Numerical Results

The performance of a parallel algorithm depends on many factors, such as the pivoting strategy and the matrix behavior. For timing purpose, we hope that the pivot entry is uniformly distributed. Thus we use the following matrix similar to the one used in [16].

$$A = (a_{ij}) = (cos((i+1)(j+1)))$$

where $0 \leq i,j \leq N$. The symmetry of this matrix is never used in the algorithm and the implementation. This matrix is easy to generate and behaves well numerically. We tested the parallel algorithm on the Intel Delta. The timing results for $N = 256$ is shown in Table 2. Assume that the execution time for one node is T_1 and T_p for P nodes. We define the speed-up, denoted by ω, and the efficiency, denoted by η, as follows

$$\omega = \frac{T_1}{T_p} \tag{14}$$

$$\eta = \frac{T_1}{PT_p} \tag{15}$$

Chapter 3

A Parallel Sparse Solver without Pivoting

In this chapter, we focus on the design and implementation of a sparse solver without pivoting on distributed memory systems. This parallel sparse solver has been implemented in AUTO94P on the Intel Hypercube and Mesh architecture machines.

3.1 Overview of Sparse Matrix Computations

A large family of applications, when formulated as a mathematical model, leads to sparse matrices. Different applications have different representations mathematically, but a common feature is that most of the sparse matrices are structured. Exploiting the structure of a particular sparse matrix is often very important, because it can significantly speed up the computation [14, 46]. For example, some systems can have a block-tridiagonal [45], almost block-diagonal [63], banded [24, 25] or a positive definite [40] structure. The LU-decomposition algorithm for sparse matrices does not differ too much from the dense one, except that the access to matrix entries is considerably more difficult. The address of an entry $a_{i,j}$ is no longer computable with an easy integer expression. Instead the entry is located by a search through a sparse matrix representation. In sparse matrix representation, one has to distinguish between structured and unstructured sparse matrices. Often, exploitation of a particular structure leads to simpler and more efficient representations [15, 53]. However, even if the full matrix A has a particular structure, the local matrices in A may not. It is possible to construct a particular distribution for some matrices such that the local structure is taken into account, but we shall study the case, where the local matrices are dense.

3.2 The Sparse Linear System

In this chapter, we consider the sparse linear system

$$Jx = f \tag{16}$$

formulated in Chapter 1 (section 1.2.2). The structure of the Jacobian matrix J is shown in Figure 2. An efficient sequential algorithm for solving this system can be found in [21]. This system is solved many times in AUTO when computing solutions of ordinary differential equations. When the problem size is big, solving this linear system becomes the dominant computation of the AUTO package. Thus efficiently solving this system is important. In the following sections of this chapter, we describe a parallel algorithm for solving the above system without pivoting.

3.3 Direct and Iterative Method

Due to computer memory limitations, many iterative methods have been designed and implemented for various types of problems. These methods typically use some kind of iteration process to correct an initial approximate solution until a better one has been found. We shall not use iterative methods. Instead, we use a direct method based on Gauss elimination, because: **a)** Direct methods are generally more efficient for computing solutions of ordinary differential equations. **b)** Direct methods are much more robust than iterative methods. **c)** Our direct method produces stability and bifurcation information as a by-product.

```
begin
    Partition strategy
    Condensation of parameters.
    Nested dissection.
    Solving the small system.
    Backsubstitution process one.
    Backsubstitution process two.
end
```

Table 3: Outline of the Parallel Algorithm without Pivoting

3.4 The Parallel Algorithm

3.4.1 Overview

The parallel algorithm consists the following parts: partition strategy, condensation of parameters, nested dissection, solving the small system and two backsubstitution processes. We describe the parallel design of each part below. In particular, we stress the communication scheme of the algorithm. We use a *direct method* as explained above in section 3.3, and in this chapter we do not consider any pivoting strategy for the Gauss elimination process. The pivoting strategy will be considered in the next chapter. The main idea of the communication scheme will be presented in graphic charts. Although the idea is illustrated by considering 8 processors, the communication scheme has no limits on the number of processors. The outline of the algorithm is shown in Table 3.

3.4.2 Partition Strategy

To achieve load balance, data should be distributed as evenly as possible, because the performance of a parallel algorithm for distributed memory systems is largely influ enced by it. Consider the sparse linear system (16), whose Jacobian matrix J is shown in Figure 2, with right hand side $f = (F_1, F_2, \cdots, F_8, FC)^T$. Assume the total number

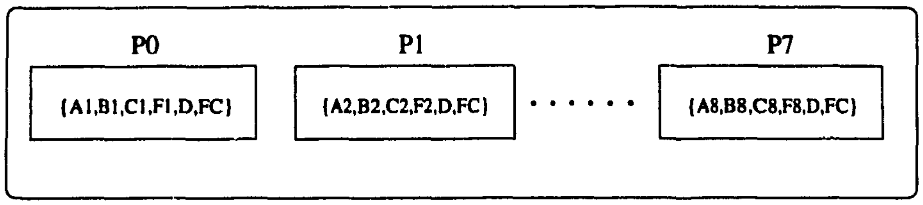


Figure 4: Data Distribution for the Sparse System

of processors is P, here $P = 8$. We define one *data unit* as $\{A_i, B_i, C_i, F_i, D, FC\}$, where $i = 1, 2, \cdots, 8$. Our partition strategy is given below. Similar to the dense matrix LU-decomposition, we partition the Jacobian matrix J and the right hand side f into P data groups. Each data group contains k (here $k = 1$) data units. In the case where the number of data units is not divisible by P, some data groups contain $k + 1$ data units. In other words, the difference between any two data groups is at most one data unit. The best case is no difference between any two groups. Of course, we can not get perfect balance, because the initialization of the data groups is not totally independent. The distribution of the Jacobian matrix J in Figure 2 and the right hand side f is shown in Figure 4. Note that the matrix D and FC in Figure 2 are shared by all the processors. This means that each processor keeps one copy of D and FC.

3.4.3 Condensation of Parameters

After distribution of the sparse matrix J and the right hand side f, each processor holds one part of the matrix J and of the right hand side f. Gauss elimination procesure to the Jacobian matrix J can be applied concurrently in each of the processors. This is because no communication occurs until at the bottom of the sparse matrix J during the elimination process. At the bottom, we need two types of communications to update the C_i's, D and FC in all processors. The first communication type is for updating the C_i's, because the right part of C_i in node p_i overlaps with the left part of C_{i+1} in node p_{i+1}. The second communication type is for updating D and FC, because they are shared by all nodes. Updating the C_i's is done in two steps.

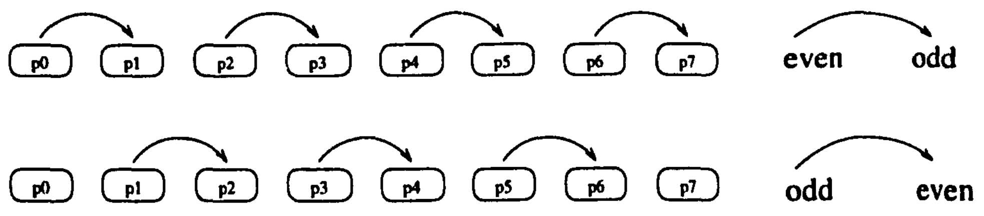


Figure 5: Communications between Odd and Even Nodes

In the first step, odd nodes send messages to even nodes; in the second step, even nodes send messages to odd nodes. The communications between odd and even nodes are indicated in Figure 5. Note that there are only two startup times for the entire communication. This communication is scalable [42] since it is independent of the number of nodes. The communication cost is the same, roughly two startup times, no matter how many nodes are used in the computation. More precisely, assume the total number of nodes is P, and define the following set

$$I_P = \{0,1,2,\cdots,P-1\} \tag{17}$$
$$L_P = \{1,2,3,\cdots,log_2^P\} \tag{18}$$

and map $M_0 : I_P \longrightarrow B_P$ where set $B_P = \{odd, even\}$. M_0 can be formulated as

$$M_0(i_p) = \begin{cases} odd & \text{if } mod(i_p,2) = 0 \\ even & \text{if } mod(i_p,2) = 1 \end{cases}$$

where $i_p \in I_P$. Table 4 outlines the communication between neighboring nodes. To update D and FC, a global sum is needed. This global sum is done by a recursive doubling procedure which takes log_2^P steps (see [19] pp95–108), where P is the total number of nodes. The recursive doubling procedure is shown in Figure 6, where only four nodes are used for simplicity. Depending on the outcome of the final sum, we used two types of recursive doubling. One is where the left most node hold the final results, as shown in the left part of Figure 6, where node p_0 holds the final result. The other is where the right most node holds the final results, as shown in the right part of Figure 6, where p_3 holds the final results. To outline the recursive doubling

```
begin
    if ($M_0(i_p)$ == odd && $i_p < P - 1$) then
        send the data to my right neighbor node $i_p + 1$
    else
        receive the data from my left neighbor node $i_p - 1$
    endif
    if ($M_0(i_p)$ == even && $i_p < P - 1$) then
        send the data to my right neighbor node $i_p + 1$
    else
        receive the data from my left neighbor node $i_p - 1$
    endif
end
```

Table 4: Neighboring Node Communications

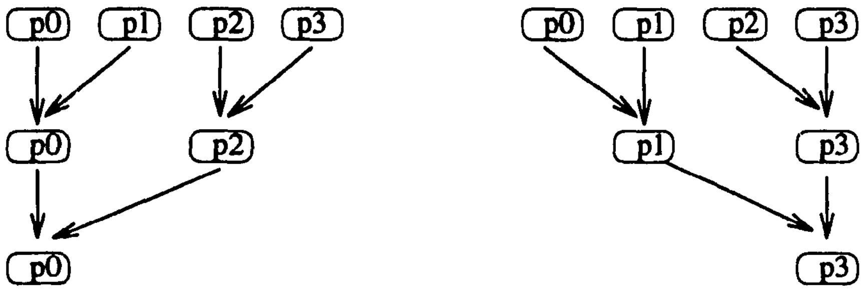


Figure 6: Recursive Doubling for 4 Nodes

procedure in our implementation, define the set $B = \{T, F\}$ and maps M_{even}, M_{odd} such that

$$M_{odd} : I_P \times L_P \longrightarrow B \tag{19}$$

$$M_{even} : I_P \times L_P \longrightarrow B \tag{20}$$

More precisely,

$$M_{odd}(i_p, l_p) = \begin{cases} T & \text{if } mod(iam_p, 2) = 0 \\ F & \text{if } mod(iam_p, 2) = 1 \end{cases} \tag{21}$$

$$M_{even}(i_p, l_p) = \begin{cases} T & \text{if } mod(iam_p, 2) = 1 \\ F & \text{if } mod(iam_p, 2) = 0 \end{cases} \tag{22}$$

where $iam_p = \frac{i_p}{2^{l_p - 1}} \cap I_P$. Further define maps

$$N_{odd}(i_p, l_p) = \begin{cases} i_p + 2^{l_p-1} & \text{if } i_p + 2^{l_p-1} \in I_P \\ \emptyset & \text{otherwise} \end{cases} \tag{23}$$

$$N_{even}(i_p, l_p) = \begin{cases} i_p - 2^{l_p-1} & \text{if } i_p - 2^{l_p-1} \in I_P \\ \emptyset & \text{otherwise} \end{cases} \tag{24}$$

where $\emptyset$ means undefined. Table 5 shows the outline of the recursive doubling in our implementation. A special recursive doubling algorithm can be found in [26]. After condensation of parameters, the Jacobian matrix J in Figure 2 become that shown in Figure 7. The shaded areas in Figure 7 will be considered in the nested dissection part. Table 6 is an outline of the condensation of parameters process.

3.4.4 Nested Dissection

After condensation of parameters, a nested dissection process is needed. For further details on nested dissection, see [36, 37]. Related results can be found in [38, 55, 66]. The communication scheme in this process is different from the one when pivoting is taking into account. The nested dissection with pivoting is described in the next chapter. Here we consider nested dissection without pivoting. Extracting the shaded areas in Figure 7, we obtain the matrix shown in Figure 8. Here A_1, A_2, C_1, C_2 and B

```
for $l_p := 1, 2, \cdots, log_2^P$ do begin
    if $(M_{odd}(i_p, l_p) == T)$ then
        send data to node $N_{odd}(i_p, l_p)$
    endif
    if $M_{even}(i_p, l_p) == T)$ then
        receive data from node $N_{even}(i_p, l_p)$
        do the summation
    endif
end
```

Table 5: The Recursive Doubling Algorithm

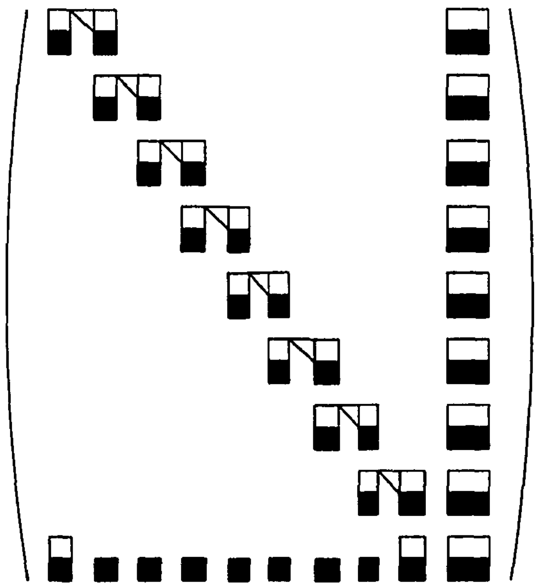


Figure 7: The Jacobian J after Condensation of Parameters

```
begin
    {for each node $p_i \in I_P$ do in parallel}
    set $D$ and $FC$ to zero if $p_i > 0$
    otherwise keep them unchanged
    for $k := 0, 1, \cdots, Max(k)$ do begin
        do Gauss elimination
    end
    update Ci's via neighboring node communication
    update D and FC by recursive doubling procedure
end
```

Table 6: Outline of the Condensation of Parameters without Pivoting

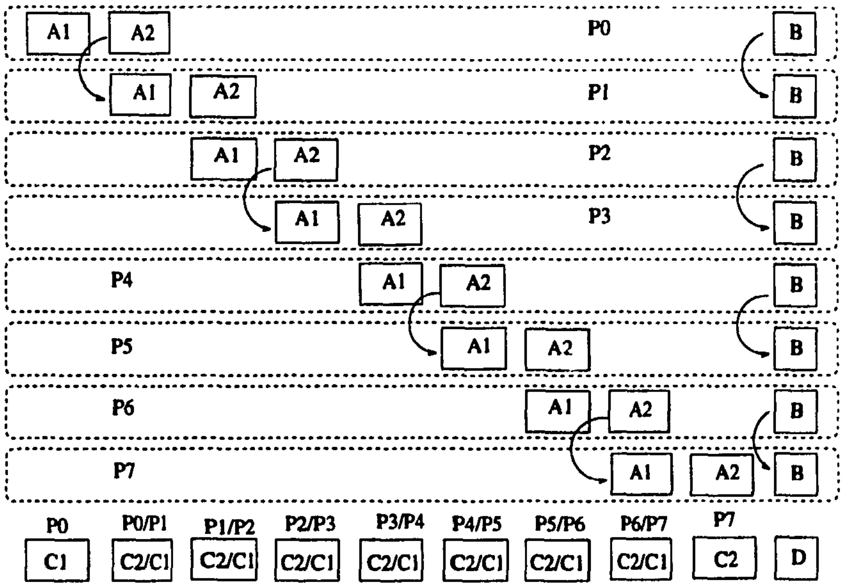


Figure 8: Initial State of the Nested Dissection

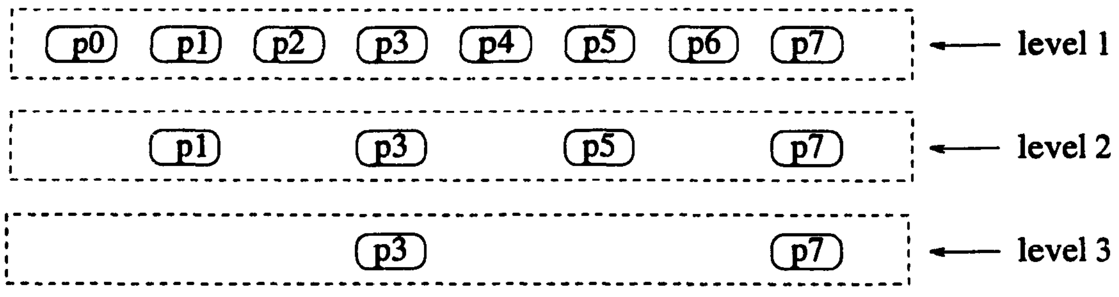


Figure 9: Busy Nodes during Each Recursion Level

are in one processor. The notation C_2/C_1 in Figure 8 means that C_2 is in node p_i and C_1 in node p_{i+1}, where $i = 0, 1, \cdots, 6$. The idea is to reflect the fact that C_2 is the same as C_1, but that they are in different processors. It needs to be mentioned that the values of A_1, A_2, C_1, C_2 and B are not the same for different nodes although we use the same notations for all nodes. We use a recursive doubling procedure for the nested dissection process. Thus, when the total number of processors is 8, the number of the recursion levels is 3. Figure 9 shows the busy nodes for each recursion level. At the first level, the communication is between two neighboring nodes. For example, in Figure 8, A_2 in node p_0 is sent to node p_1, because the Gauss elimination of A_1 in p_1 needs information from A_2 in p_0. A similar communication has to be done for the B's, C_i's and F_i's. At the first level of the recursive procedure, all nodes are working. After the elimination at the first level, the matrix is as shown in Figure 10, where the shaded areas denote fill-in. At the second level, half of the nodes will be idle, while the other half is doing eliminations. The situation is graphically illustrated in Figure 9. At the second level, communication is not between neighbors only; instead a group of nodes communicates with each other. The number of nodes participating in the communication in each group is $2^{k-1}+1$, where k is the recursion level. Figure 10 shows the communication scheme. After the elimination at level 2, the matrix is shown in Figure 11. Level 3 is similar as level 2, except the number of nodes in each communication group is more than before. The final state, after the nested dissection, is shown in Figure 12.

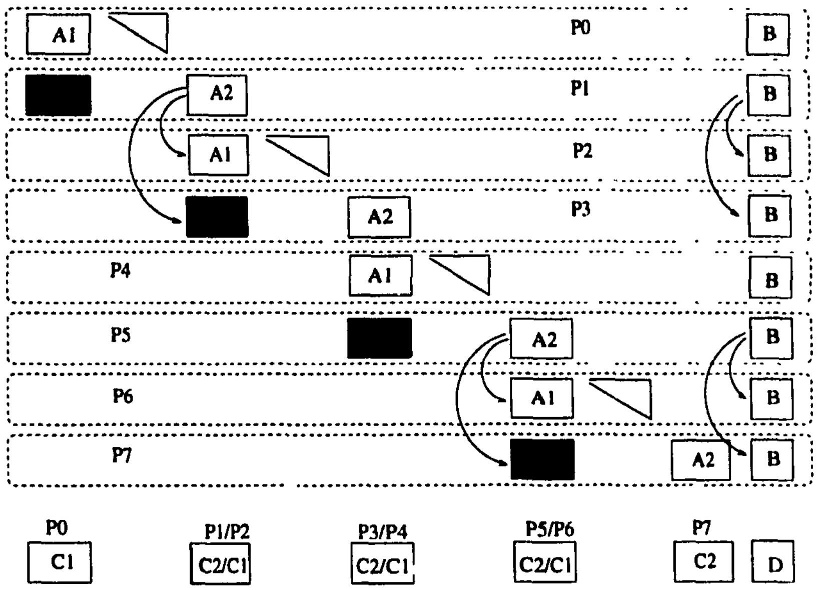


Figure 10: Initial State of Level 2

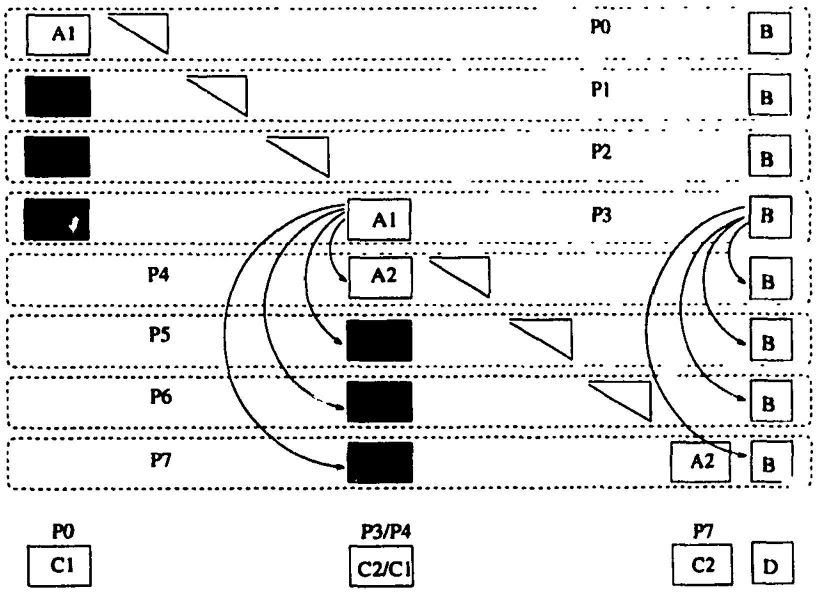


Figure 11: Initial State of Level 3

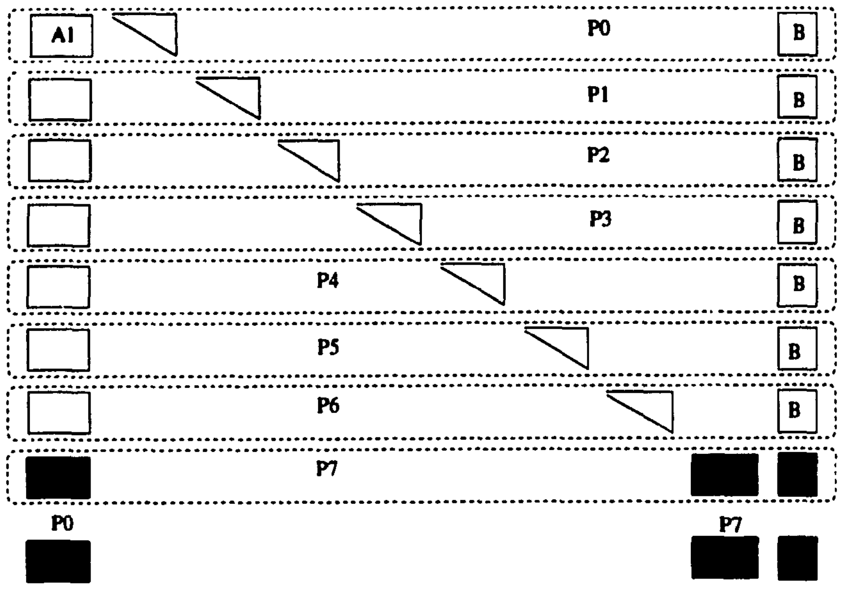


Figure 12: Final State after the Nested Dissection

3.4.5 Solving the Small System

The shaded area in Figure 12 is a relatively small square matrix. It is generally nonsingular. This square matrix with the corresponding right hand side is solved by Gauss elimination with complete pivoting. The solving of the square system is done by one node, here node p_7. The shaded area C_1 in p_0 in Figure 12 has to be sent to node p_7, before node p_7 can compute the solution of the small system.

3.4.6 Backsubstitution

After solving the small system, we can obtain the solution for the full matrix shown in Figure 8 by a backsubstitution process. In order to do the backsubstitution, the solution of the small system has to be sent to all other nodes. This can be done by a broadcast from node p_7. Figure 13 shows the broadcast. After the broadcast, we can do the backsubstitution in each node concurrently, and hence compute the solution of the nested system. In order to obtain the solution of the full system 16, another backsubstitution has to be done. This final backsubstitution process does not need

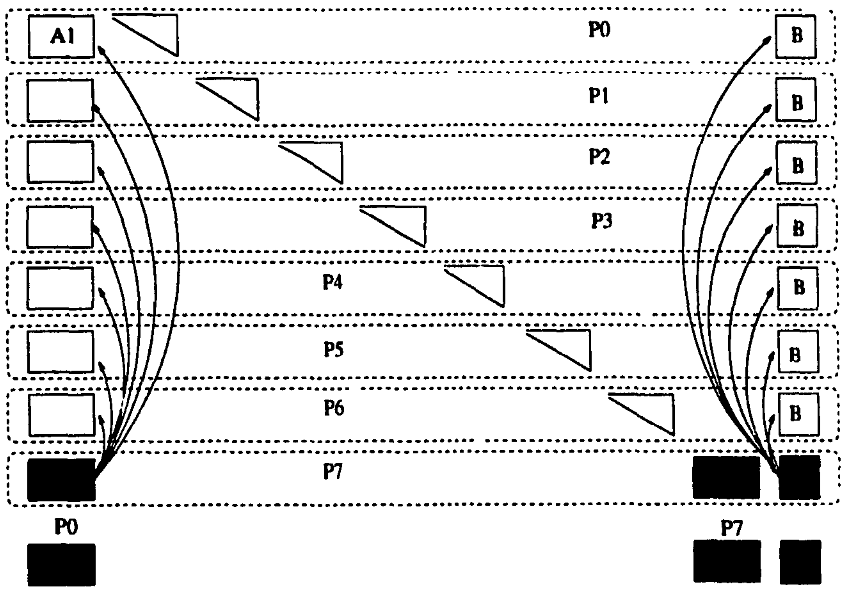


Figure 13: Broadcast Solution from the Small System

any communication and can be done concurrently in all nodes. The full solution has now been obtained, but it is scattered over all the nodes. Thus a concatenation is needed. This concatenation process is identical to that in the pivoting case, and will be described in the next chapter (section 4.2.8).

3.5 Timing Results

AUTO94P without pivoting has been tested on the Intel Hypercube and Mesh machines. The numerical timing results reported here are obtained on the Intel Delta; 512 Intel iPSC/860 nodes connected by a mesh network. The Delta system is located at Caltech. The example used for the numerical experiments is *tim.f*, which is one of the drivers in the AUTO94 package used solely for timing purposes. It defines a simple first order system of ordinary differential equations with boundary conditions. The dimension of the system is a variable. The parameter $NDIM$ (dimension of the system) may be assigned by any even value within the modifiable limit stated in

AUTO94 user manual. The system of the equations is

$$u_1' = u_2 \tag{25}$$

$$u_2' = -\lambda \sum_{i=0}^{25} \frac{u_1^i}{i!} \tag{26}$$

where $u_1 \in R^n$, $u_2 \in R^n$, $n = NDIM/2$ and λ is the continuation parameter. The boundary conditions are

$$u_1(0) = 0 \tag{27}$$

$$u_1(1) = 0 \tag{28}$$

The starting solution at $\lambda = 0$ is $u_1(x) = u_2(x) = 0$, $x \in [0,1]$. There are no integral conditions except a pseudo-arclength integral, which is always there. The computation is such that for each run of the problem, there will be 10 decompositions and 10 backsubstitutions in the linear equation solver. We use the efficiency formula (14) defined in Chapter 2 and the relative efficiency formula (29) defined below to measure the performance of AUTO94P without pivoting.

$$\eta = \frac{mT_m}{nT_n}, \tag{29}$$

where T_m, T_n are the execution times obtained by using m, n nodes, respectively.

3.5.1 Timing Results Including I/O Time

Timing results in this section include "I/O time". Specifically, all execution times in the following tables are taken from the first node (node p_0). The first node does all the I/O operations, thus its execution time is a little bit longer than that of other nodes.

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.113E+03	1	100%
2	0.611E+02	1.85	92.47%
4	0.359E+02	3.15	78.69%
8	0.222E+02	5.09	63.63%
16	0.138E+02	8.19	51.18%
32	0.116E+02	9.74	30.44%
64	0.103E+02	10.97	17.14%

Table 7: Without Pivoting 1: NDIM=12, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.603E+03	1	100%
2	0.316E+03	1.91	95.41%
4	0.168E+03	3.59	89.73%
8	0.944E+02	6.39	79.85%
16	0.558E+02	10.81	67.54%
32	0.332E+02	18.16	56.76%
64	0.268E+02	22.50	35.16%

Table 8: Without Pivoting 1: NDIM=24, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
8	0.562E+03	1	100%
16	0.306E+03	1.84	91.83%
32	0.176E+03	3.19	78.83%
64	0.106E+03	5.30	66.27%

Table 9: Without Pivoting 1: NDIM=48, NTST=64, NCOL=4, NMX=10

3.5.2 Timing Results Excluding I/O time

Timing results in this section do not include "I/O time". The execution time in the following tables is the first node's execution time minus the "average I/O time". The average I/O times for all tables in this section are shown in Table 10. For simplicity, we do not count the work-load difference between nodes, which also affects the execution. The average I/O time is calculated by the following formula

$$T_{average} = \frac{T_{minimum} + T_{maximum}}{2} \tag{30}$$

In Table	Minimum I/O time	Maximum I/O time	Average I/O time
11	1.00	3.50	2.25
12	1.00	3.50	2.25
13	3.00	7.00	5.00

Table 10: Without Pivoting: Average I/O Time

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.111E+03	1	100%
2	0.589E+02	1.88	94.23%
4	0.337E+02	3.29	82.34%
8	0.199E+02	5.58	69.72%
16	0.116E+02	9.57	59.81%
32	0.935E+01	11.87	37.10%
64	0.805E+01	13.79	21.55%

Table 11: Without Pivoting 2: NDIM=12, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.601E+03	1	100%
2	0.314E+03	1.91	95.70%
4	0.166E+03	3.62	90.51%
8	0.923E+02	6.51	81.39%
16	0.536E+02	11.21	70.08%
32	0.310E+02	19.39	60.58%
64	0.246E+02	24.43	38.17%

Table 12: Without Pivoting 2: NDIM=24, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
8	0.557E+03	1	100%
16	0.301E+03	1.85	92.52%
32	0.171E+03	3.26	81.43%
64	0.101E+03	5.51	68.93%

Table 13: Without Pivoting 2: NDIM=48, NTST=64, NCOL=4, NMX=10

Chapter 4

A Parallel Sparse Solver with Pivoting

In this chapter, we focus on the design and the implementation of a sparse solver with pivoting on distributed memory systems. This parallel sparse solver has been implemented in AUTO94P on the Intel Hypercube and Mesh architecture machines.

4.1 Introduction

In this chapter, we again consider a direct solution algorithm for the sparse linear system (16) in Chapter 1 (section 1.2.2). To enhance numerical stability, we now use a pivoting strategy. We use the same partitioning strategy as described in Chapter 3.

4.2 The Parallel Algorithm

4.2.1 Overview

The top level of the parallel algorithm is similar to that in Chapter 3. It is outlined in Table 14. We describe the parallel design of each part of Table 14, except that we

```
begin
    Partitioning strategy
    Condensation of parameters with pivoting.
    Nested dissection with pivoting.
    Solving the small system.
    Backsubstitution of the nested dissection.
    Backsubstitution of the condensation of parameters.
end
```

Table 14: Outline of the Parallel Algorithm with Pivoting

do not repeat the partitioning strategy. Similar to Chapter 3, we present the communication schemes in graphic charts with emphasis on pivoting. To avoid duplication, we do not repeat parts similar to those in Chapter 3, particularly for the condensation of parameters. Although the idea is illustrated by considering 8 processors, the communication scheme has no limits on the number of processors.

4.2.2 Pivoting Strategy

To enhance numerical stability [2, 63], we use a pivoting strategy. The simplest strategy is to have no strategy at all. In the $k-th$ elimination step, entry $a(k,k)$ of matrix A_i is chosen as pivot. The pivot is always the current entry or a preset pivot. This type of pivoting, the static strategy, suffers from numerical instability. In some cases this may be an acceptable risk, in other cases enough might be known about the matrix to guarantee that the errors remain small. In a dynamic strategy, the decision which entry should be the pivot in the $k-th$ step of the elimination is postponed until the pivot is actually needed. It is desirable to use a dynamic strategy. It is proven that LU-decomposition with complete row and column pivoting is numerically stable. For any matrix the computed LU-decomposition is the LU-decomposition of a nearby matrix. We use restricted row and column pivoting in our parallel algorithm. Complete pivoting searches all feasible entries for the one that is

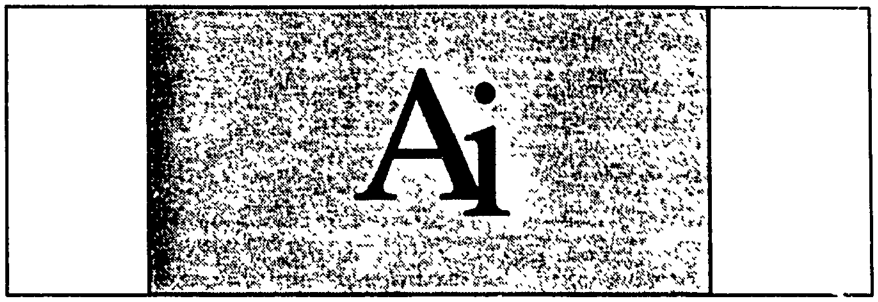


Figure 14: The Pivot Window for A_i

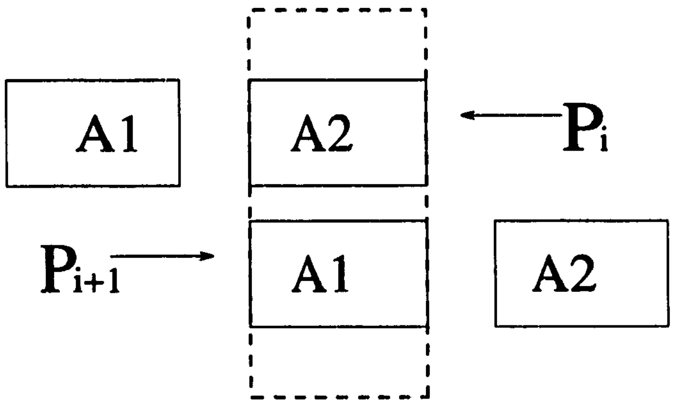


Figure 15: The Pivot Window for Nested Dissection

largest in absolute value. This requires a search over $(N-k)^2$ entries, where N is the dimension of the matrix being considered. Since the Jacobian matrix, as shown in Figure 2, is sparse, it is reasonable to restrict the pivot search region to the region where the entries are non-zero. During the condensation of parameters process, we use a pivot search window at each elimination step. In Figure 14, the pivot window for A_i is the shaded area. The corresponding pivot window in nested dissection is shown in Figure 15. It needs to be mentioned here that the pivot window in the nested dissection does not reside in any node alone but resides either in two adjacent neighboring nodes or in two nodes with distance of 2^k, where k is the current recursion level. Thus, during any recursion level of the nested dissection, say level k, message

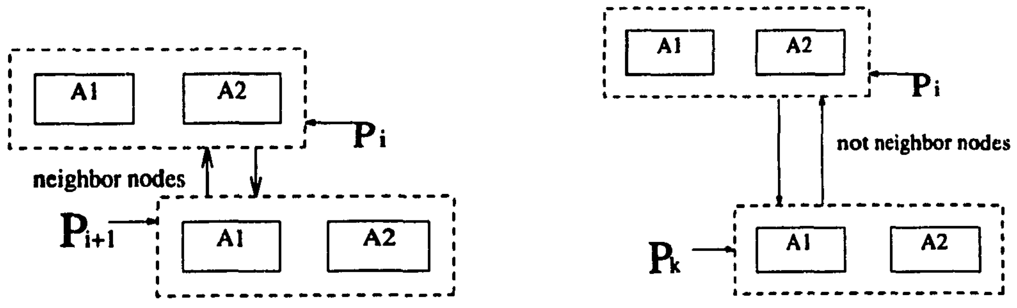


Figure 16: Communication within the Pivot Window

passing between two nodes with distance 2^k is needed. The information exchanges are indicated in Figure 16. The efficiency of the above pivoting strategy is obviously better than the complete row and column one, due to the restriction of the searching region.

4.2.3 Condensation of Parameters

Starting from the initial state of the Jacobian matrix, as shown in Figure 2, condensation of parameters transforms the matrix into the form shown in Figure 17. Similar to that in Chapter 3, the elimination process is done concurrently in each processor, except at the bottom of the Jacobian matrix J. The communications at the bottom have been described in the Chapter 3 (section 3.4.3). The difference here is that, during the elimination process in each node, a local pivot within the pivot window mentioned in section 4.2.2 is searched. In addition, the communications at the bottom of the Jacobian matrix J are delayed until the nested dissection process to improve the degree of parallelism or granularity. With the above communication delay, we save two start-up times. More precisely, assume that T_s is the start-up time for each simple communication and that a global sum takes a factor of log_2^P times a simple startup time, where P is the total number of processors. We roughly save the following communication time in the condensation of parameters process:

$$T_{save} = (log_2^P + 2)T_s$$

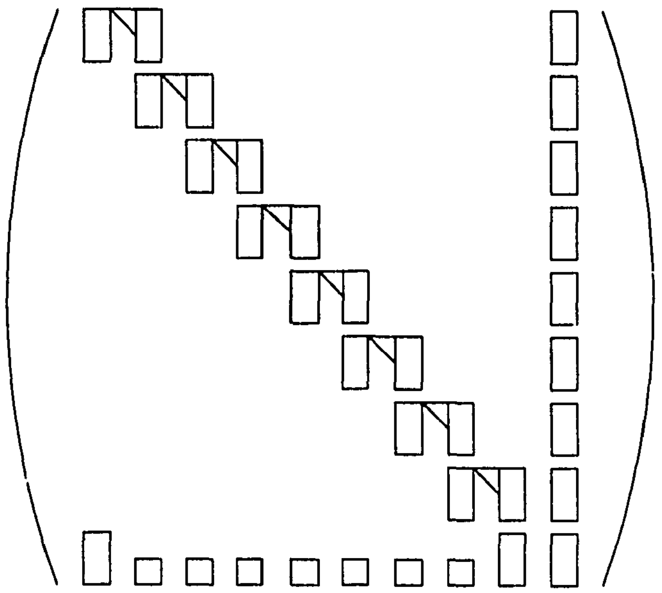


Figure 17: The Jacobian J after Condensation of Parameters

Here P is the total number of nodes. Table 15 is an outline of the condensation of parameters process.

4.2.4 Nested Dissection

The nested dissection part plays an important role in the full algorithm. It is complicated in the sense that both pivoting and elimination are not local. They both require communication between specified nodes. Assume that P is the total number of nodes allocated to a task. We know that each node p_i has a unique processor identification number. We denote this node identification number by pid_i. This pid_i is typically assigned by the operating system, depending on the system. Assume I_P, L_P are defined as before and define the following set

$$\begin{aligned} B &= \{T, F\} && (31)\\ PID &= \{pid_i \mid i \in I_P\} && (32)\\ & && (33) \end{aligned}$$

```
begin
    {for each node $p_i \in I_P$ do in parallel}
    set $C, D$ and $FC$ to zero if $p_i > 0$
    otherwise keep them unchanged
    for $k := 0, 1, \cdots, Max(k)$ do begin
        {Pivoting strategy and Bookkeeping}
        do pivot search
        do index exchange
        do elimination
    end
end
```

Table 15: Outline of the Condensation of Parameters With Pivoting

It is convenient, and necessary in practice, to map the system assigned PID to individually organized I_P, such as a linear mapping. One can define the map as

$$M_{si} : PID \longrightarrow I_P \tag{34}$$

One way to construct the map is to sort all pid_i in PID in nondecreasing order and renumber each element by an integer starting from zero. Depending on the application, the construction of the map is often done differently. To better describe the communication method, we need to define some maps. The first two are $M_i, i = 1, 2$ such that

$$M_i : I_P \times L_P \longrightarrow B \quad i = 1, 2 \tag{35}$$

We define the two maps as follows.

$$M_1(i_p, l_p) = \begin{cases} T & \text{if } mod(iam_p, 2) = 0 \\ F & \text{if } mod(iam_p, 2) = 1 \end{cases} \tag{36}$$

$$M_2(i_p, l_p) = \begin{cases} T & \text{if } mod(iam_p, 2) = 1 \\ F & \text{if } mod(iam_p, 2) = 0 \end{cases} \tag{37}$$

where, as before,

$$iam_p = \left\{ \frac{i_p}{2^{l_p - 1}} \right\} \cap I_P \tag{38}$$

```
begin
    {for each $p_i \in I_P$ do in parallel}
    for $k := 0, 1, \cdots, Max(k)$ do begin
        {Pivoting strategy and Bookkeeping}
        {Here pivot search is local}
        do pivot search
        do elimination
    end
end
```

Table 16: Nested Dissection Process 1

Secondly, define $N_i, i = 1, 2$ such that

$$N_i : I_P \times L_P \longrightarrow I_P \tag{39}$$

We define $N_i, i = 1, 2$ as follows.

$$N_1(i_p, l_p) = \begin{cases} i_p + 2^{1_p - 1} & \text{if } i_p + 2^{1_p - 1} \in I_P \\ \emptyset & \text{otherwise} \end{cases} \tag{40}$$

$$N_2(i_p, l_p) = \begin{cases} i_p - 2^{1_p - 1} & \text{if } i_p - 2^{1_p - 1} \in I_P \\ \emptyset & \text{otherwise} \end{cases} \tag{41}$$

where $\emptyset$ means undefined. Now we can describe the algorithm in two parts. Part one is outlined in table 16 and part two in table 17. Below we describe our communication mechanism graphically. The nested dissection is carried out similar to the recursive doubling procedure described before. During each level of recursion, half of the total number of nodes sit idly. Assume that the total number of *data units*, defined in Chapter 3 (section 3.4.2), is 24. After the data distribution over 8 nodes described in Chapter 3, each node holds 3 *data units* We first look at what happens in any one of the nodes, say in p_i. This part of the algorithm is outlined in table 16. The elimination in each node is sequential. For now, we don't consider shared data, such as matrix D and vector FC. The result of this elimination is shown in Figure 18

```
begin
    {for each node $p_i \in I_P$, do the following in parallel}
    {Here pivot search is not local}
    for $k := 1, \cdots, max(k)$ do begin
        for $l_p := 1$ to $log_2^P$ do begin
            if $(M_1(i_p, l_p) == T)$ then
                Search local pivot element $PIV_1$
                receive $PIV_2$ from node $N_1(i_p, l_p)$
                Determine $PIV = max(PIV_1, PIV_2)$
                if $(PIV == PIV_1)$ then
                    send pivot row to node $N_1(i_p, l_p)$
                else
                    send current row to node $N_1(i_p, l_p)$
                endif
                do eliminations
            endif
            if $(M_2(i_p, l_p) == T)$ then
                Search local pivot element $PIV_2$
                send local pivot row to node $N_2(i_p, l_p)$
                if $(PIV_2 != PIV)$ then
                    receive pivot row from node $N_2(i_p, l_p)$
                else
                    receive current row from node $N_2(i_p, l_p)$
                endif
                do eliminations
            endif
        end
    end
end
```

Table 17: Nested Dissection Process 2

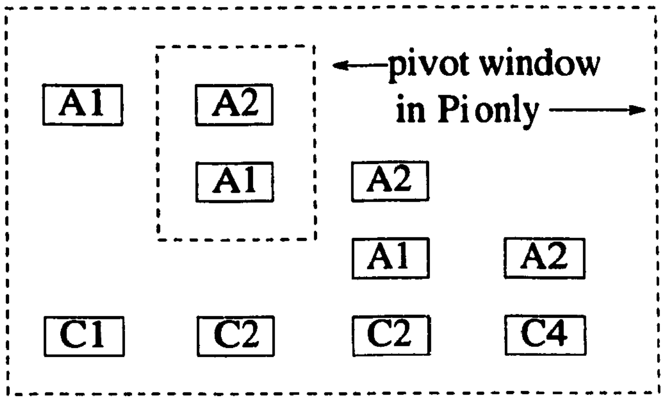


Figure 18: The Initial State in Node p_i

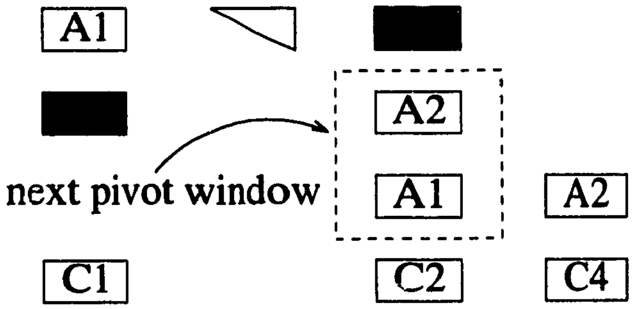


Figure 19: The Intermediate State in Node p_i

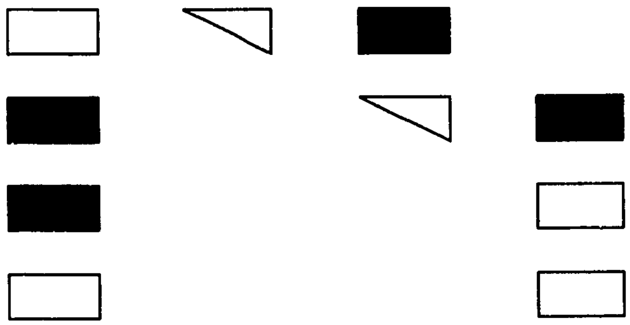


Figure 20: The Final State in Node p_i

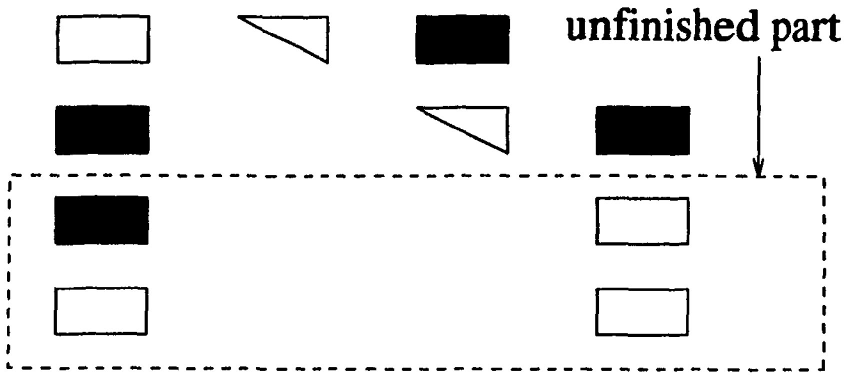


Figure 21: Enclosed in the Dashed-line Bo^v is the Unfinished Part

to Figure 20. The shading denotes fill-in due to the elimination and pivoting. So far, there is no communication yet. Now we look at part two of the nested dissection corresponding to the outline in Table 17. After each node finishes the first part of the nested dissection outlined in Table 16, extracting the unfinished part shown in Figure 21 in each node, we have the situation shown in Figure 22. Since we have 8 nodes, the number of recursion levels is 3. In Figure 22, we have also shown the communications during level 1 of the recursion. Note that the communications from condensation of parameters have been merged here in order to improve the degree of the parallelism. All communications indicated by an arrow arc are done in parallel. In other words, they only need one start up time T_s plus the time to send or receive certain data between any two nodes. The pivot window covers neighboring nodes for level 1 but not in levels 2 and 3, as one can see from Figures 23 and 24. In total we need 3 startup times to complete the elimination for 8 nodes. The elimination result and the communications are indicated for each level of the recursive process in Figure 22 to Figure 24. The final result is indicated in Figure 25. The shared data, such as the matrix D and the vector FC, are updated by a global sum similar to Figure 6. Note that only the last node holds the global sum of D and FC, as will be explained below.

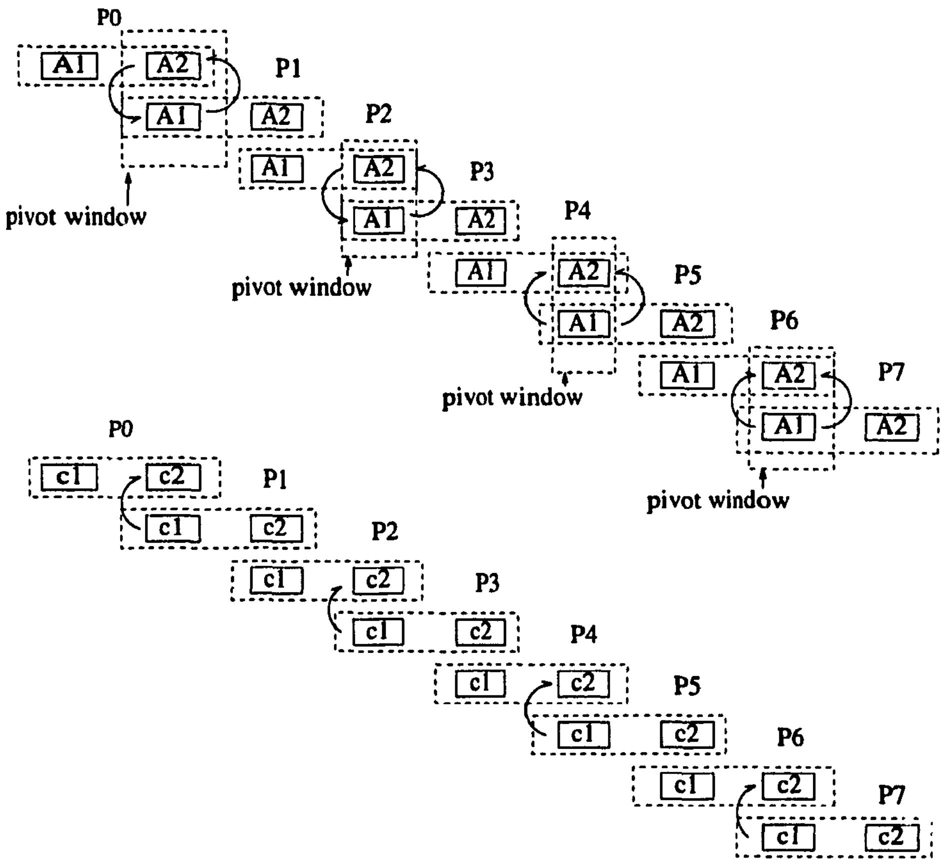


Figure 22: Level 1

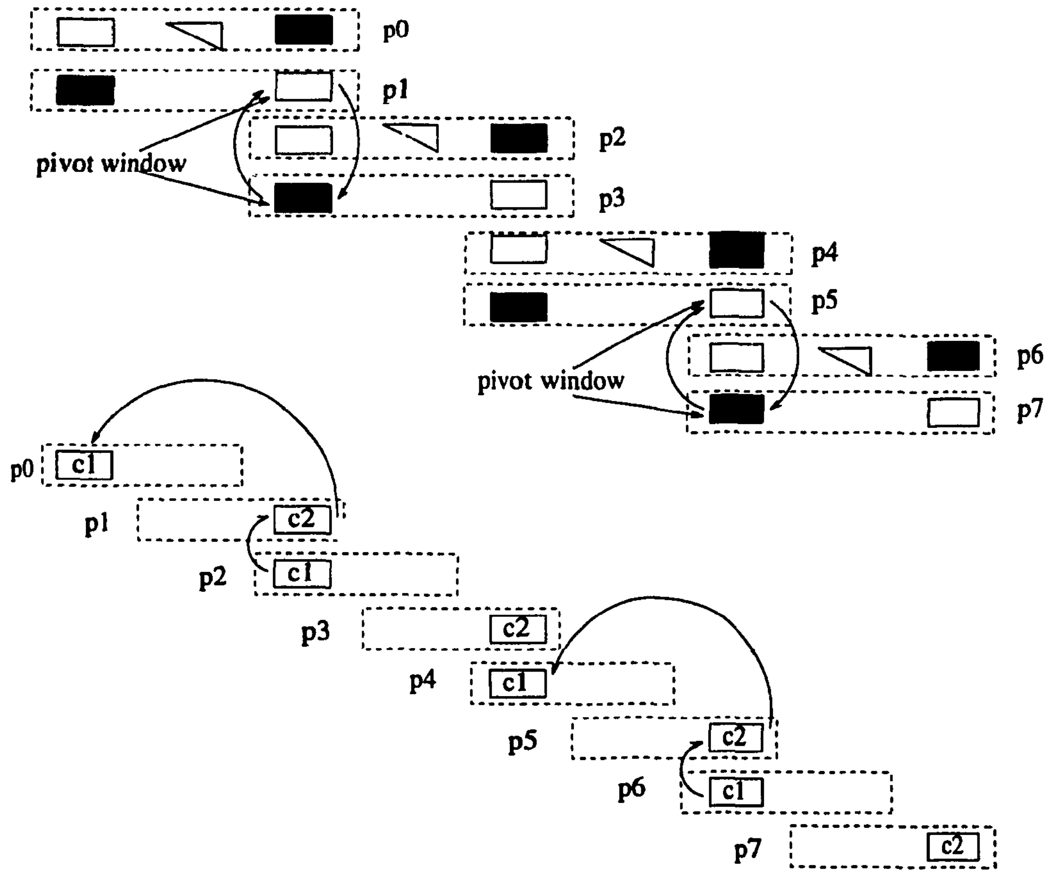


Figure 23: Level 2

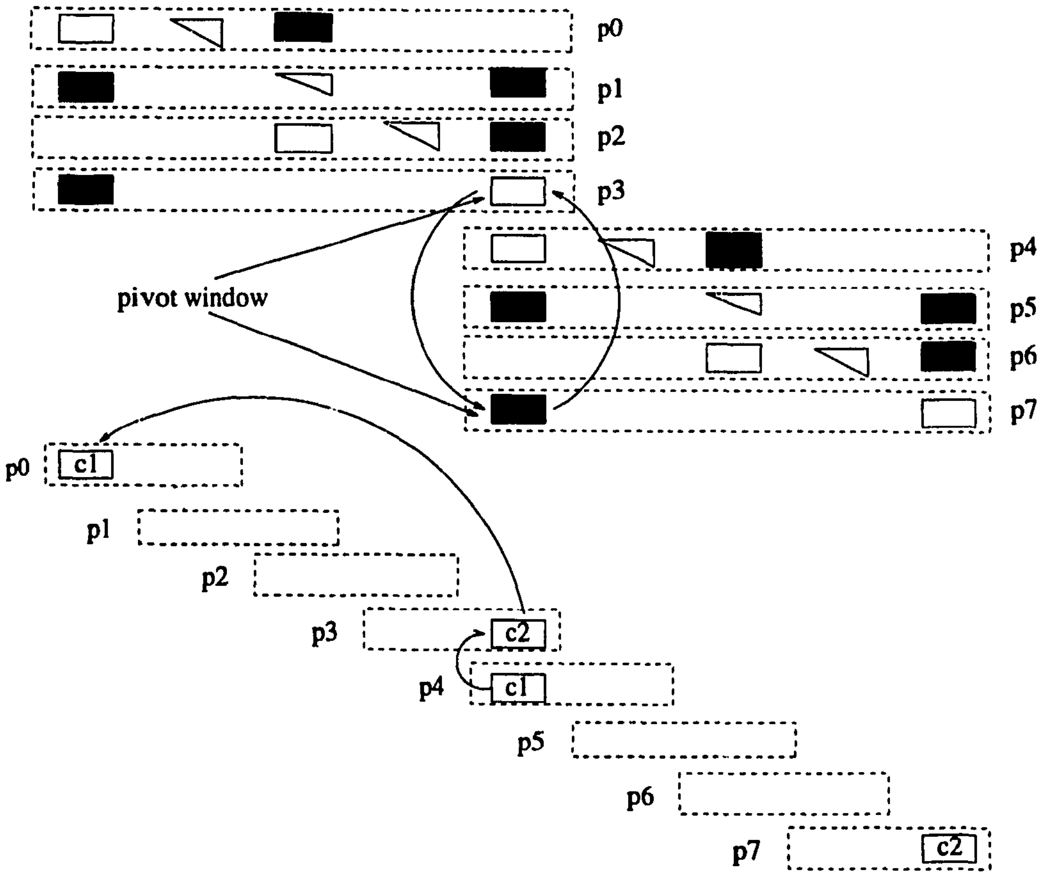


Figure 24: Level 3

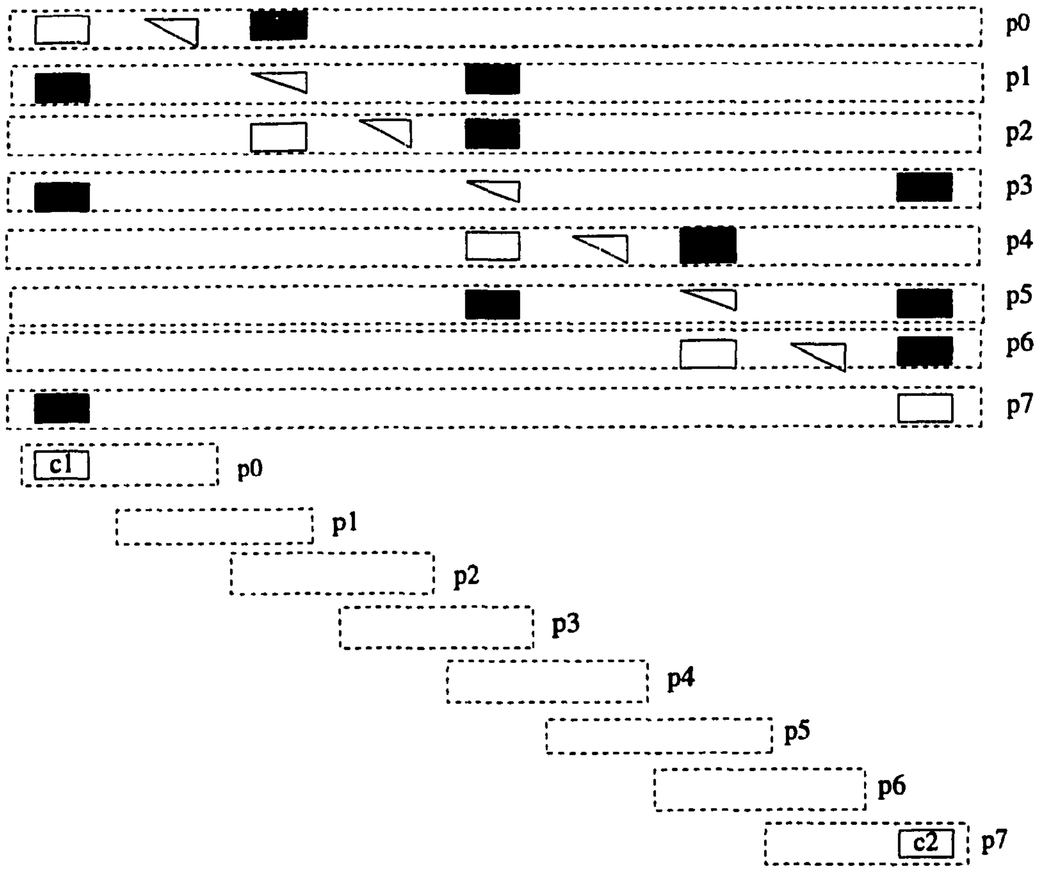


Figure 25: The Final Result of the Nested Dissection Process

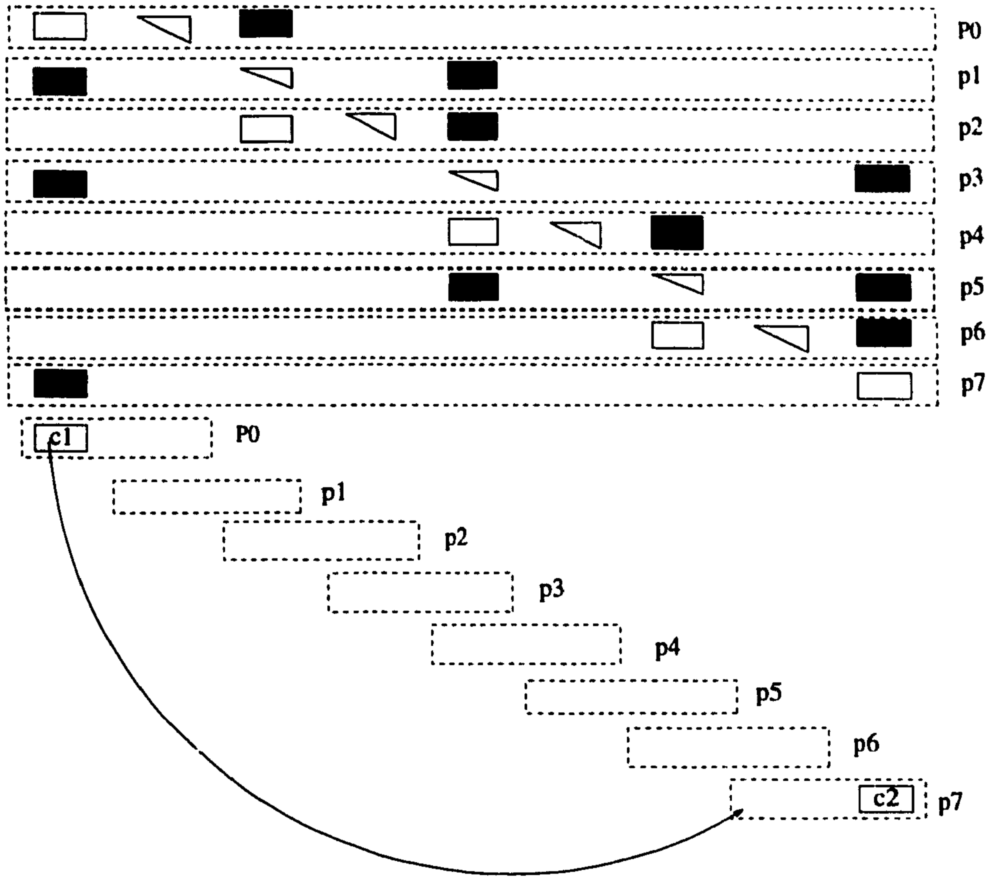


Figure 26: Send c_1 from Node p_0 to Node p_7

4.2.5 Solving the Small System

After the nested dissection, we need to solve a relatively small square system described in Chapter 3 (section 3.4.5). In Figure 25, we send C_1 from node p_0 to node p_7 as indicated in Figure 26. We can extract the square system enclosed by a big dashed line box as indicated in Figure 27 in node p_8. The shaded area implicitly represents the Poincaré map that we need to preserve in the AUTO package. This small square system is solved by node p_7 using complete pivoting. Thereafter node p_7 broadcasts the solution to all the other nodes as indicated in Figure 28. The outline for this part is in Table 18.

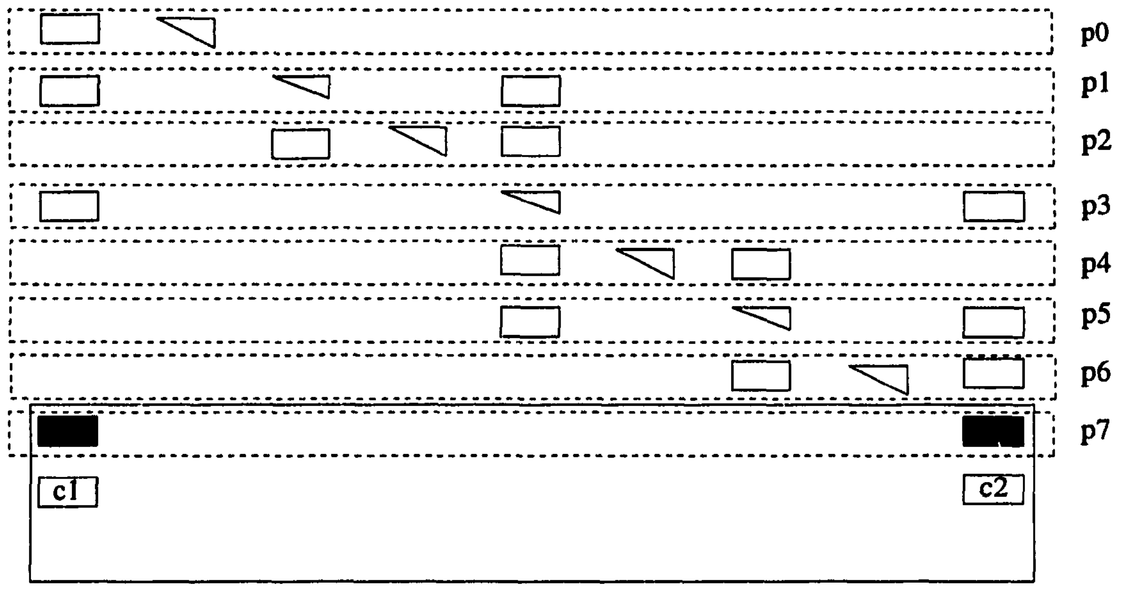


Figure 27: The Square Matrix Enclosed in the Solid-line Box

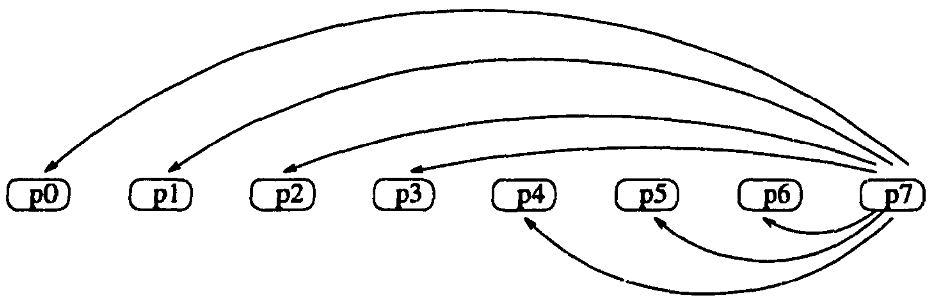


Figure 28: Broadcast of the Partial Solution from Node p_7 to All Other Nodes

```
begin
    { for each node $i_p \in I_P$ do }
    if ($i_p$ == 0) then
        send data to the last node $i_{P-1}$
    endif
    if ($i_p$ == $i_{P-1}$) then
        receive data from node $i_0$
        Solve the small system by
        Gauss elimination with complete pivoting
    endif
    if ($i_p$ == $i_{P-1}$) then
        broadcast the solution
    else
        receive the solution
    endif
end
```

Table 18: Solving the Small System

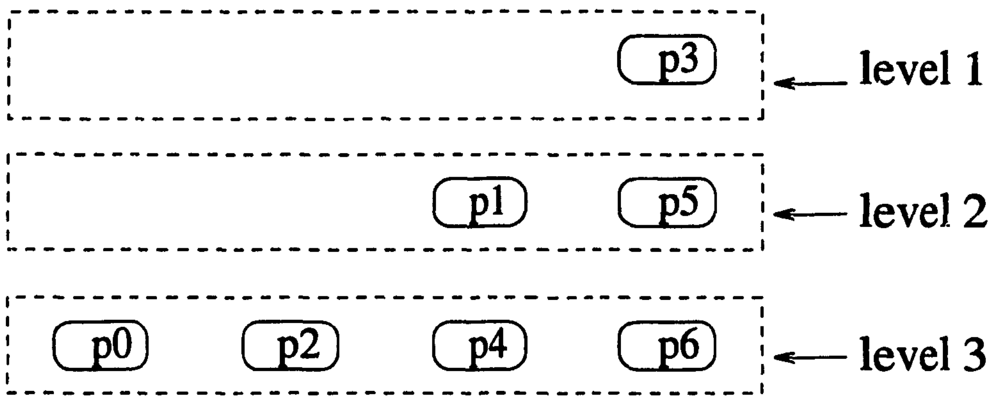


Figure 29: Busy Nodes for Each Level during Backsubstitute

4.2.6 Backsubstitution for the Nested Dissection

After solving the small square system with complete pivoting, we need to do backsubstitution. There are two backsubstitution processes. The first one is associated with the nested dissection process. The second one is associated with condensation of parameters. The first one is a recursive procedure similar to the nested dissection. It requires log_2^P levels or steps. The number of working nodes here is reversed compared with that of the nested dissection. The busy nodes are indicated in Figure 29 level by level. Initially, only one node works. Thereafter the number of working nodes is doubled with each increase of the recursion level. At each level some communications are needed as will be illustrated by pictures. The algorithm is outlined below. Assume the ınaps $M_i(i_p, l_p), i = 1,2$ are defined as in (36). Define sets $Nb_i(i_p, l_p) \subset I_P, i = 1,2$ such that

$$Nb_1(i_p,l_p) = \{i_p \mid M_1(i_p,l_p-1) = T \quad and \quad l_p > 1\} \tag{42}$$

$$Nb_2(i_p,l_p) = \{i_p \mid M_1(i_p,l_p+1) = T \quad and \quad l_p < P-1\} \tag{43}$$

The backsubstitution process is now outlined in table 19. The backsubstitution process can be interpreted graphically as follows. After broadcasting the solution from the small square system, we can solve part of the solution corresponding to the shaded triangular area in node p_3 in Figure 30. In order to solve the part of the solution

```
begin
    { for each $i_p \in I_P$ do in parallel}
    for $l_p := log_2^P, \cdots, 1, step\ \ -1$ do begin
        if $(M_1(i_p, l_p) == T)$ then
            if $(l_p < P - 1)$ then
                receive solution from nodes $Nb_2(i_p, l_p)$
            endif
            do local backsubstitution only in the last block row
            if $(l_p > 1)$ then
                send the solution to the
                following nodes $Nb_1(i_p, l_p)$
            endif
        endif
    end
    do the backsubstitution for the remaining block rows locally
end
```

Table 19: Backsubstitution Process 1

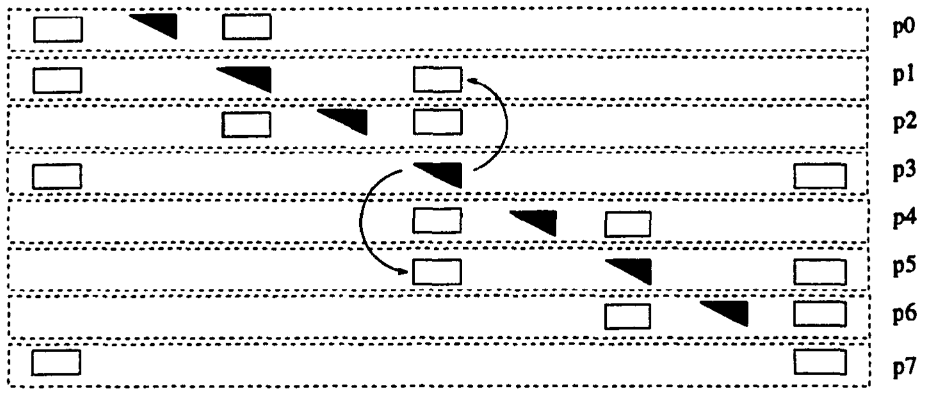


Figure 30: Backsubstitution Level 1

```
begin
    { for each $i_p \in I_P$ do in parallel}
    backsubstitution locally in each node $i_p$
end
```

Table 20: Backsubstitution Process 2

corresponding to the shaded triangular area in nodes p_1 and p_5, we need to send the solution from node p_3 to node p_1 and to node p_5 as shown in Figure 30. This completes level 1 in the backsubstitution process. Similarly one can complete level 2 as indicated in Figure 31. No communication is needed for level 3.

4.2.7 Backsubstitution for the Condensation of Parameters

After the above backsubstitution process associated with the nested dissection, one can solve the whole solution of the linearized system in parallel as indicated in Figure 2. This part of the computation does not involve any communication. It is a simple backsubstitution that we omit describing here. The outline can be found in table 20.

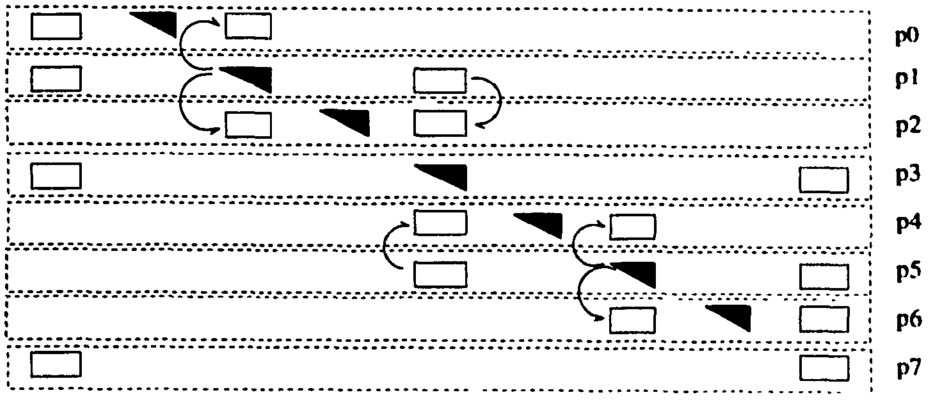


Figure 31: Backsubstitution Level 2

4.2.8 Merging the Solutions

So far we have computed the solution to the system indicated in Figure 2. However, the solution is still scattered over all nodes, and we need to concatenate them. This is done by a global operation as indicated in Figure 32. It consists of two steps. The

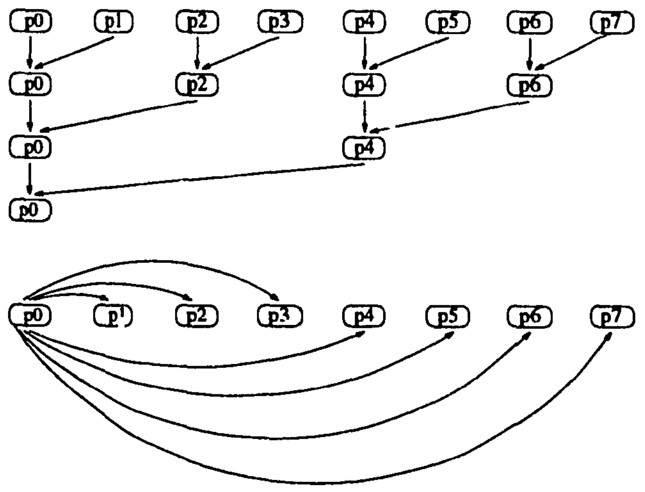


Figure 32: Concatenation of the Solutions

first step is to let node p_0 collect all solutions from the other nodes, the second step is to broadcast the full solution.

4.3 Timing results

AUTO94P with pivoting has been tested on the Intel Hypercube and Mesh machines. The numerical timing results reported here are obtained on the Intel Delta. We use the same example as in Chapter 3 for the timing purpose here. The efficiency formula (14) defined in Chapter 2 and the relative efficiency formula (29) defined in Chapter 3 are used to measure the performance of AUTO94P with pivoting.

4.3.1 Timing Results Including I/O Time

Timing results in this section include I/O time. Specifically, all execution times in the following tables are taken from the first node (node 0). The first node does all the AUTO I/O operations. Thus its execution time is a little bit longer than that of other nodes.

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.14816E+03	1	100%
2	0.76882E+02	1.92	96.36%
4	0.41966E+02	3.53	88.26%
8	0.24917E+02	5.95	74.33%
16	0.16852E+02	8.79	54.95%
32	0.12519E+02	11.83	36.98%
64	0.11386E+02	13.01	20.33%

Table 21: With Pivoting 1: NDIM=12, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.29486E+03	1	100%
2	0.15434E+03	1.91	95.52%
4	0.82070E+02	3.59	89.82%
8	0.46864E+02	6.29	78.65%
16	0.34325E+02	8.59	53.69%
32	0.21596E+02	13.65	42.67%
64	0.19698E+02	14.97	23.39%

Table 22: With Pivoting 1: NDIM=12, NTST=128, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
2	0.30752E+03	1	100%
4	0.16642E+03	1.85	92.39%
8	0.93223E+02	3.30	82.47%
16	0.62372E+02	4.93	61.63%
32	0.44666E+02	6.88	43.03%
64	0.36942E+02	8.32	26.01%

Table 23: With Pivoting 1: NDIM=12, NTST=256, NCOL=4, NMX=10

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.86821E+03	1	100%
2	0.43959E+03	1.98	98.75%
4	0.22831E+03	3.80	95.07%
8	0.12090E+03	7.18	89.77%
16	0.69891E+02	12.42	77.64%
32	0.43812E+02	19.82	61.93%
64	0.33428E+02	25.97	40.58%

Table 24: With Pivoting 1: NDIM=24, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
4	0.45262E+03	1	100%
16	0.13188E+03	3.43	85.80%
32	0.83715E+02	5.41	67.58%
64	0.57496E+02	7.87	49.20%

Table 25: With Pivoting 1: NDIM=24, NTST=128, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
8	0.47475E+03	1	100%
32	0.16136E+03	2.94	73.55%
64	0.11212E+03	4.23	52.93%

Table 26: With Pivoting 1: NDIM=24, NTST=256, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
8	0.77154E+03	1	100%
16	0.40720E+03	1.89	94.74%
32	0.22908E+03	3.37	84.20%
64	0.14745E+03	5.23	65.41%

Table 27: With Pivoting 1: NDIM=48, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
16	0.79527E+03	1	100%
32	0.43987E+03	1.81	90.40%
64	0.26383E+03	3.01	75.36%

Table 28: With Pivoting 1: NDIM=48, NTST=128, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
32	0.84541E+03	1	100%
64	0.50268E+03	1.68	84.09%

Table 29: With Pivoting 1: NDIM=48, NTST=256, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
2	0.14787E+04	1	100%
4	0.75186E+03	1.97	98.34%
8	0.39168E+03	3.78	94.38%
16	0.21482E+03	6.88	86.04%
32	0.12844E+03	11.51	71.95%

Table 30: With Pivoting 1: NDIM=48, NTST=32, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
16	0.15305E+04	1	100%
32	0.88878E+03	1.72	86.10%

Table 31: With Pivoting 1: NDIM=96, NTST=32, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
64	0.10121E+04	1	100%

Table 32: With Pivoting 1: NDIM=96, NTST=64, NCOL=4, NMX=10

4.3.2 Timing Results Excluding I/O Time

Timing results in this section do not include "I/O time". The execution time in the following tables is the first node's execution time minus the "average I/O time". We use formula (30) in Chapter 3 to calculate the the average I/O time. The average I/O times for all tables in this section are shown in Table 33. For simplicity, we do not consider work-load difference between different nodes. This also makes the execution time differ from node to node. The average I/O times for all tables in this section are shown in the Table 33 below.

In Table	Minimum I/O time	Maximum I/O time	Average I/O time
34	1.01	1.88	1.45
35	1.89	3.65	2.77
36	3.72	6.94	5.33
37	1.87	3.83	2.85
38	3.59	7.25	5.42
39	9.53	15.02	12.27
40	1.88	4.09	2.99
41	3.89	7.01	5.45
42	7.09	13.67	10.38
43	13.81	30.23	22.02
44	3.4	7.1	5.25
45	13.47	13.47	13.47

Table 33: With Pivoting: Average I/O Time

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.14671E+03	1	100%
2	0.75432E+02	1.94	97.25%
4	0.40516E+02	3.62	90.53%
8	0.23467E+02	6.25	78.15%
16	0.15402E+02	9.53	59.53%
32	0.11069E+02	13.25	41.42%
64	0.99360E+01	14.77	23.07%

Table 34: With Pivoting 2: NDIM=12, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.29209E+03	1	100%
2	0.15157E+03	1.93	96.35%
4	0.79300E+02	3.68	92.08%
8	0.44094E+02	6.62	82.80%
16	0.31555E+02	9.26	57.85%
32	0.18826E+02	15.52	48.49%
64	0.16928E+02	17.25	26.96%

Table 35: With Pivoting 2: NDIM=12, NTST=128, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
2	0.30219E+03	1	100%
4	0.16109E+03	1.88	93.80%
8	0.87893E+02	3.44	85.95%
16	0.67042E+02	4.51	56.34%
32	0.39336E+02	7.68	48.01%
64	0.31612E+02	9.56	29.87%

Table 36: With Pivoting 2: NDIM=12, NTST=256, NCOL=4, NMX=10

Number of Nodes	Execution time	Speed-up	Efficiency
1	0.86536E+03	1	100%
2	0.43674E+03	1.98	99.07%
4	0.22546E+03	3.84	95.95%
8	0.11805E+03	7.33	91.63%
16	0.67041E+02	12.91	80.67%
32	0.40962E+02	21.13	66.02%
64	0.30578E+02	28.30	44.22%

Table 37: With Pivoting 2: NDIM=24, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
4	0.44720E+03	1	100%
16	0.12646E+03	3.54	88.41%
32	0.78295E+02	5.71	71.40%
64	0.52076E+02	8.59	53.67%

Table 38: With Pivoting 2: NDIM=24, NTST=128, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
8	0.46248E+03	1	100%
32	0.14909E+03	3.10	77.55%
64	0.99850E+02	4.63	57.90%

Table 39: With Pivoting 2: NDIM=24, NTST=256, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
2	0.14757E+04	1	100%
4	0.74887E+03	1.97	98.53%
8	0.38869E+03	3.80	94.91%
16	0.21183E+03	6.97	87.08%
32	0.12545E+03	11.76	73.52%

Table 40: With Pivoting 2: NDIM=48, NTST=32, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
8	0.76609E+03	1	100%
16	0.40175E+03	1.91	95.34%
32	0.22363E+03	3.43	85.64%
64	0.14200E+03	5.40	67.44%

Table 41: With Pivoting 2: NDIM=48, NTST=64, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
16	0.78489E+03	1	100%
32	0.42949E+03	1.83	91.37%
64	0.25345E+03	3.10	77.42%

Table 42: With Pivoting 2: NDIM=48, NTST=128, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
32	0.82339E+03	1	100%
64	0.48066E+03	1.71	85.65%

Table 43: With Pivoting 2: NDIM=48, NTST=256, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
16	0.15252E+04	1	100%
32	0.88351E+03	1.73	86.31%

Table 44: With Pivoting 2: NDIM=96, NTST=32, NCOL=4, NMX=10

Number of Nodes	Execution time	Relative Speed-up	Relative Efficiency
64	0.99863E+03	1	100%

Table 45: With Pivoting 2: NDIM=96, NTST=64, NCOL=4, NMX=10

Chapter 5

Some Implementation Issues of AUTO94P

5.1 Introduction

The two sparse linear solvers described in Chapters 3 and 4 were inplemented in AUTO94P. The initial implementation of AUTO94P was done on the Gamma machine, an Intel Hypercube machine with 64 Intel iPSC/860 nodes. Thereafter it was ported to the Delta system, an Intel mesh machine with 512 iPSC/860 nodes. Both machines are located at Caltech. The Delta has a peak performance of 32 Gflops. Communication between the processors on both machines is done by message passing. Communication can be either synchronous (blocking) or asynchronous (nonblocking). Some of the basic communication primitives on these machines [12, 13] are shown in Table 46. The detailed hardware configuration for the Intel Touchstone Delta [13] is shown in Table 47.

Type	Syntax	Description
synchronous	csend(Type,Data,Len,Node,Pid)	send a message and wait for completion
	crecv(Type,Data,Len)	receive a message and wait for completion
asynchronous	MsgId=isend(Type,Data,Len,Node,Pid)	send a message
	MsgType=irecv(Type,Data,Len)	receive a message
	msgdone(Id)	determine message done or not
	msgwait(Id)	wait for message to complete

Table 46: Primitives on the Gamma and Delta machines

Hardware:

Console:

80386,387 (16 MHz), 8 MB memory, 380 MB disk
60 MB cartridge tape, 1.2MB floppy, Ethernet

Mesh:

513 computational nodes
39 i/o nodes (80386 16 MHz, 8 MB/node)
i860 (40MHz) 4KB inst. cache
8KB data cache, 2-way set assoc.
16 MB memory/node (cache miss=10 ticks)
64 disk = 90 Gigabytes of total disk-space
2 SCSI disks/node (1.4 GB/disk), 14 Exabyte tapes
12 service nodes (80386 16 MHz, 8 MB/node)
2 Ethernet nodes (80386 16 MHz, 8 MB/node)
4 HIPPI nodes (i860 40 MHz, 32 MB/node)

Software:

UNIX Sys V 3.2, NX/M Rel 1.4
FORTRAN and C from the Portland Group (Release 3.0)
Express from Parasoft, PCN

Table 47: Intel Touchstone Delta Configuration

5.2 Node Organization for the Implementation

Assume that the number of nodes is fixed. Each node has a unique identifier, which is used as an address in the exchange of messages. In our implementation, the nodes are organized as a one dimensional grid. User identification for each node is then a number p between 0 and $P-1$, where P is the number of nodes. We assume that the number of nodes is a power of 2, because the initial implementation was done on the Intel Hypercube machine where the number of nodes allocated is always a power of 2.

5.3 I/O Strategy of AUTO94P

Since in our case, all nodes share one common I/O file and because the common file can be very big for a large size problems, our policy is to let all nodes do the reading concurrently. Each node has its own file pointer, so that they don't affect each other. For writing, we only allow one node to do the operation, simply because we only need one output file. Here we assigned node p_0 to do all writing. Thus the total execution time for node p_0 is longer than that of all the other nodes. More precisely, the I/O control is as follows. Each node maintains its own file pointer. File access requests are honored on a first-come, first-served basis. If two nodes write to the same place in the file, the second node overwrites the data written by the first node. Because the nodes do not have to communicate with each other, the best performance is obtained.

5.4 Interface Routines of AUTO94P

In order to facilitate porting AUTO94 to other systems, a set of interface routines is provided. The idea is to separate the system dependent primitive calls, so that porting can focus on the interface, assuming organization of the communication scheme is preserved. In principle, one only needs to concentrate on this set of interface routines. Depending on the particular multicomputer system, one may need to synthesize some of the primitives in our implementation. Refer to the appropriate system manual to find out how to do this. A possible simple case is that only the syntax of the primitives needs changes, with little consideration of their semantics. Since multicomputer systems, especially distributed memory systems, may differ significantly both in architecture and in system software, an absolutely isolated interface is hard to provide. But a least our interface separates the primitives that have to be changed when porting to other systems. Depending on the network structure or the way the processor elements are connected, the current communication organization may not be the best suitable to that particular system, and hence the performance may be affected.

Chapter 6

Graphical User Interface for AUTO94

6.1 X Window System

The X Window System is a hardware and operating system independent window system [58]. It was developed jointly by MIT and Digital Equipment Corporation, and has been adopted by the computer industry as a standard for graphics applications. X controls a bit-mapped display in which each pixel on the screen is individually controllable. This allows applications to draw pictures as well as text. The unique feature of X is that it is based on a network protocol instead of on system-specific procedure and system calls. This network protocol enables X to be ported to different computer architectures and operating systems; it also allows programs to run on one architecture or operating system while display on another. Because of its unique design, X can make a network of different computers cooperate. For example, a computationally intensive application might run on a supercomputer, but take input from and display output on a work station connected across a local area network. To the user, the application would simply appear to be running on the work station.

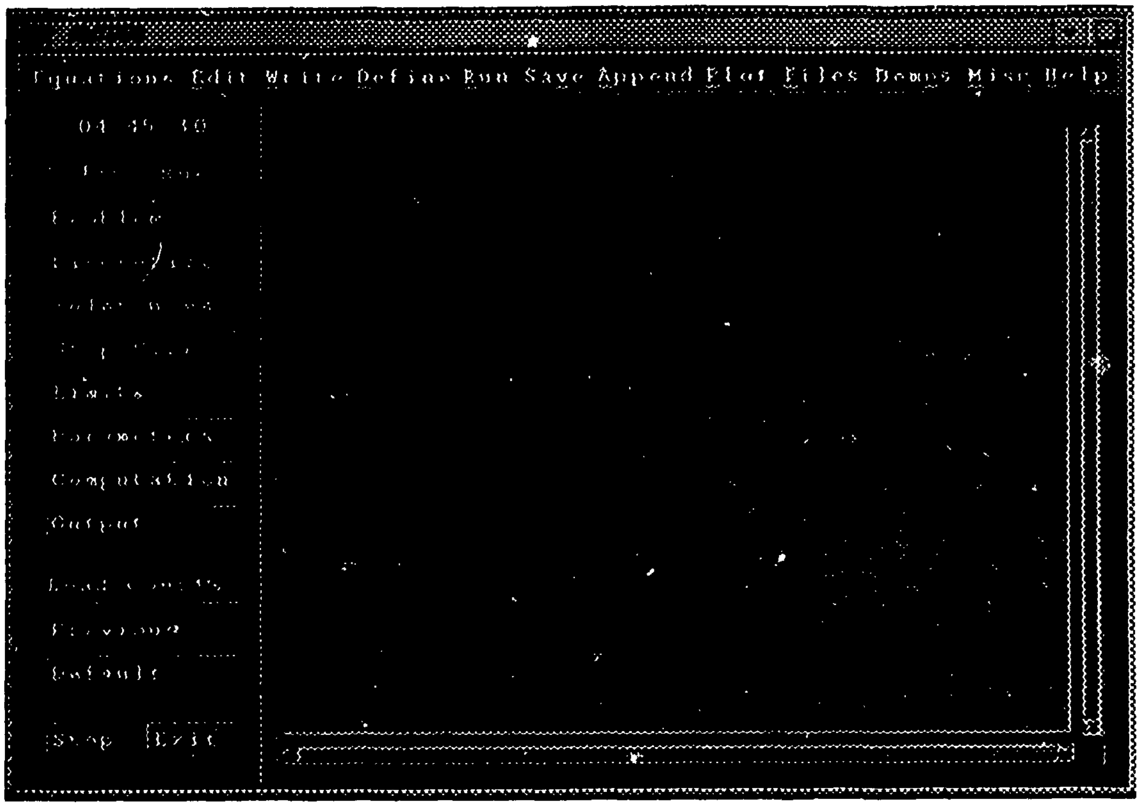

Equations Edit Write Define Run Save Append Plot Files Demos Misc Help
Parameters
Computation
Output
Load Consts
Previous
Default
Stop Exit

Figure 33: The Graphical User Interface for AUTO94

6.2 Overview of the GUI

The graphical user interface (GUI) of AUTO94 is based on the X window system. It is written in Motif [58, 59, 61, 62, 67]. The point-and-click access provides the user with a convenient interface. It has a three dimensional feeling and is network transparent. The GUI interface has been tested on SGI and SUN platforms. It should, in principle, run on any Unix system. For the GUI interface, the LEFT or the FIRST button of the mouse is used most frequently, the RIGHT or the THIRD button of the mouse is also used in some cases, however, the MIDDLE or the SECOND button of the mouse is never used. Unless mentioned otherwise, use the LEFT or the FIRST button of the mouse. The appearance of the interface is shown in Figure 33.

6.3 New Features of AUTO94

AUTO94 inherits all capabilities of AUTO86. In addition, it: **a)** changed the internal structures, **b)** eliminated the preprocessor, **c)** reduced the number of user-supplied subroutines, **d)** replaced the eigenvalue solver by EISPACK [70], which is included in the package, and **e)** repalced the linear solver for ordinary differential equations, the new li[illegible] solver does Gauss elimination with a restricted row and column pivoting, **f)** added a graphical user interface which simplified the use of the package.

6.4 Installation

AUTO94 is packed as a tar file called *auto.tar.Z*. After obtaining the tar file, the following has to be done:

1. uncompress the tar file by typing
   uncompress auto.tar

2. unbundle the tar file by typing
   tar xvfo auto.tar

3. compile the package by typing
   make sgi
   for Silicon Graphics platforms, or
   make
   for SUN and other platforms

4. remove unnecessary files by typing
   make clean

5. set the environment variable AUTO_DIR by adding
   setenv AUTO_DIR $HOME/auto/94
   to the *.cshrc*, assuming **auto** is the top directory which contains the package.

6. set AUTO aliases for keyboard commands by adding the following to the *.cshrc* file

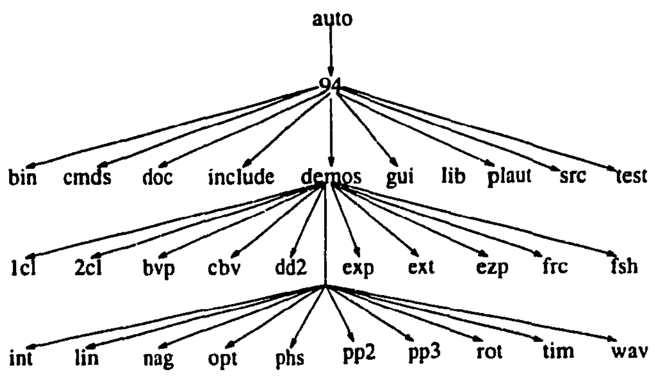
source $AUTO_DIR/cmds/@auto.alias

Note that these keyboard commands are not used by the GUI interface, they can be used if X windows are unavailable.

7. remove the tar file by typing

   rm auto.tar

The AUTO94 directory tree is shown in Figure 34, where **auto** is the top directory. Each directory contains a *README* file which briefly tells what the directory contains and what can be done in the directory. Compilation can be done separately in each directory, when applicable, by typing the command **make** in that directory.



auto
94
bin cmds doc include demos gui lib plaut src test
1cl 2cl bvp cbv dd2 exp ext ezp frc fsh
int lin nag opt phs pp2 pp3 rot tim wav

Figure 34: The AUTO94 Directory Tree

6.5 Setting Up the X Resources for the GUI

On a color monitor, the graphical user interface can be color configured. On a black and white monitor, there is no need to set up the X resource file. In this case, the appearance of the interface will be black and white. Included in the package, are two X resource files, *Xdefaults.1* and *Xdefaults.2*, in directory $AUTO_DIR/gui. The simplest way to get a color appearance is to type the command

cp $AUTO_DIR/gui/Xdefaults.1 $HOME/.Xdefaults

or

cp $AUTO_DIR/gui/Xdefaults.2 $HOME/.Xdefaults

depending on which color selection one prefers. In addition, one can set up a customized **.Xdefaults** file by editing **Xdefaults.1** or **Xdefaults.2** to produce various color appearances of the interface. Color names can be found in the file *rgb.txt* under */usr/lib/X11* on any UNIX system with Motif installed.

6.6 Features of the GUI

The features of the GUI are summarized below:

1. The entire set of AUTO-constants can be interactively manipulated in a popup window (the full AUTO-constants panel).

2. AUTO-constants, according to their purpose, are also grouped into sub-panels. Each of the sub-panels can be interactively manipulated consistently with the full AUTO-constants panel.

3. A default AUTO-equations file *aut.f* and a default AUTO-constants file *r.aut* can be loaded by a simple mouse click action.

4. Automatic or separate loading of a previous AUTO-constants file when an existing AUTO-equations file is loaded.

5. An AUTO-equations editor is provided to create or modify the AUTO-equations file.

6. Recompilation of the executable is done only when necessary.

7. Computational result can be plotted in a popup graphics window.

8. Reset restart files interactively.

9. Interactively manipulate output data files.

10. Run AUTO demos by point-and-click access.

11. On-line help on all AUTO-constants, browsing of demos, their help files, and the full AUTO user manual.
12. On-line activation of the editors **xedit** and **emacs**.
13. Terminate any computational run by pressing a **Stop** button.
14. Network transparency.
15. Ability to port to any UNIX platforms.
16. Convenient clock for timing.
17. Easy installation and setup of the package.

6.7 Functionalities of the Menu Bar

The menu bar of the graphical user interface contains the pull down menus **Equations, Edit, Write, Define, Run, Save, Append, Plot, Files, Demos, Misc** and **Help**. Some of these contains further sub-pull down menus. All are described below.

6.7.1 Equations Menu

The **Equations** pulldown menu contains the menu items **Old, New** and **Default** as shown in Figure 35. Selecting item **Old** will popup a file selector in which one can select an existing AUTO-equations file. These are called *name.f* where *name* is the equation name. The corresponding AUTO-constants file, called *r.name*, will be loaded automatically if it exists. The AUTO-equations file, when selected, will be loaded into the AUTO-equations editor for further modification and its *r.name* file will be loaded into the AUTO-constants panels described below. Selecting item **New** will popup a prompt window in which one is asked to enter the abbreviated *name* of the equation. The AUTO-equations file will be saved later as *name.f*. Selecting item **Default** will automatically load the default AUTO-equations file, called *aut.f*,

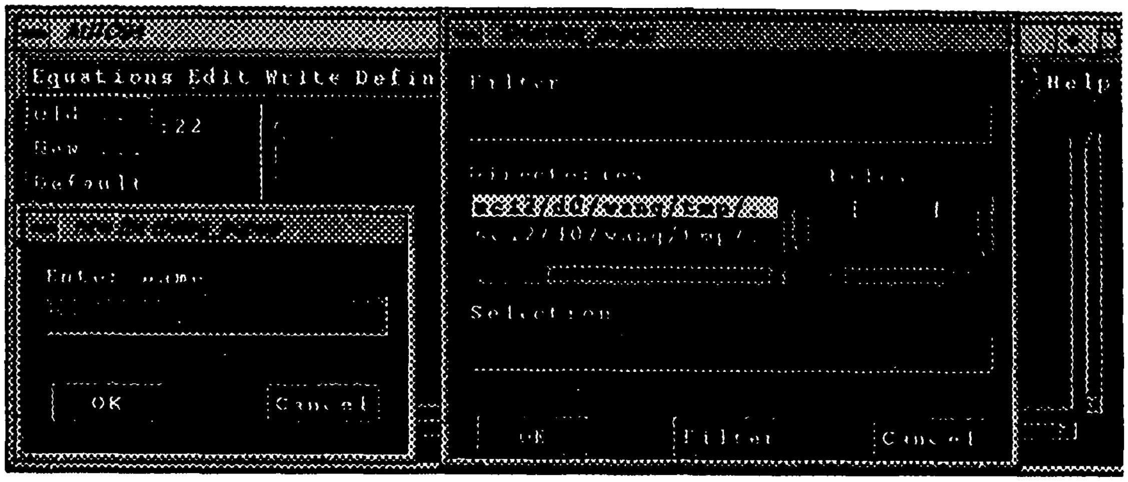


Figure 35: The Equations Menu

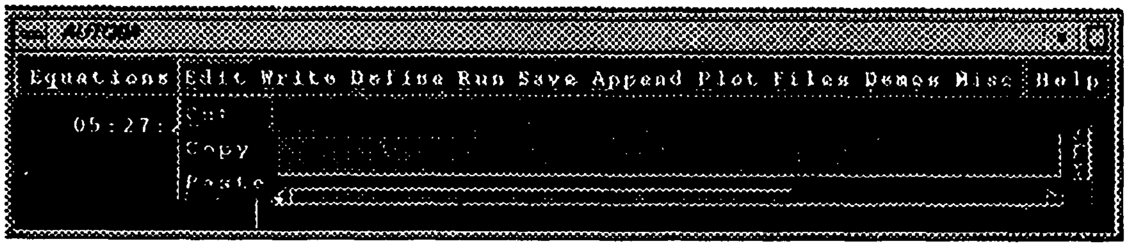

AUTO94
Equations Edit Write Define Run Save Append Plot Files Demos Misc Help
Cut
Copy
Paste

Figure 36: The Edit Menu

and the corresponding default AUTO-constants file *r.aut*. No matter which item one selects, the equation *name* will be used as the current *name* from then on.

6.7.2 Edit Menu

The **Edit** pulldown menu contains the menu items **Cut, Copy** and **Paste** as shown in Figure 36. Selecting item **Cut** will delete the highlighted portion of the text, which is put into a buffer for later pasting. Selecting item **Copy** will copy the highlighted portion of the text into a buffer for pasting. The highlighted portion of the text remains where it is. Selecting item **Paste** will copy the text in the buffer to the location where the mouse cursor is pointing to.

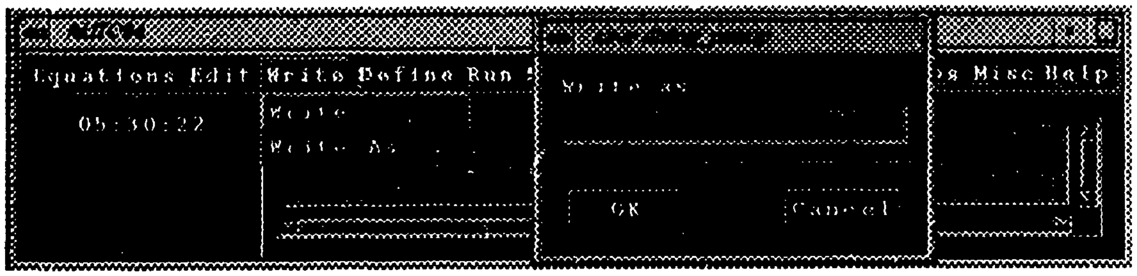


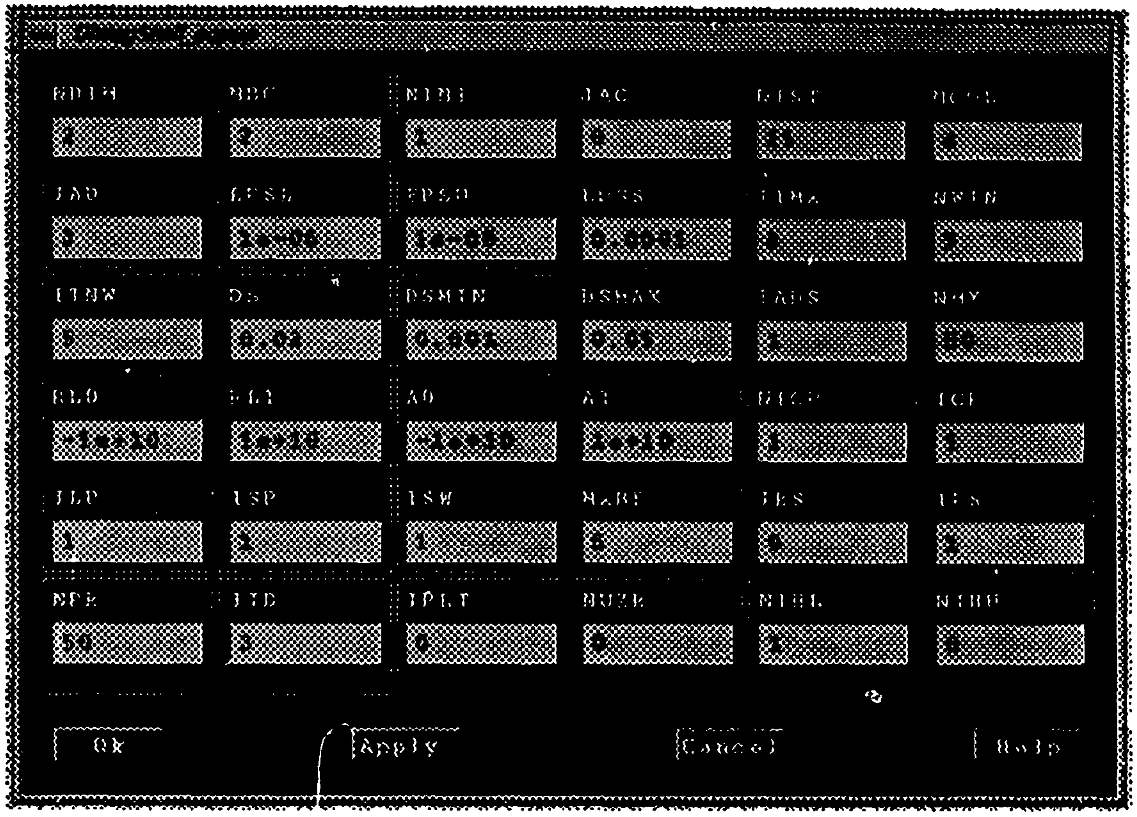
Figure 37: The Write Menu

6.7.3 Write Menu

The **Write** pulldown menu contains the menu items **Write** and **Write As** as shown in Figure 37. Selecting the item **Write** will save the AUTO-equations file in the AUTO-equations editor in the current directory as *name.f*, where *name* is the current equation name. Selecting item **Write As** will popup a prompt window in which one needs to enter a new *name*. Once the new *name* is entered, the AUTO-equations file in the AUTO-equations editor will be saved as the new *name.f* and *name* becomes the equation name.

6.7.4 Define Menu

The **Define** menu contains no items except itself. It is a cascade button. Pressing **Define** will popup the full AUTO-constants panel in which one can manipulate all AUTO-constants. There are 36 text fields in the full panel as shown in Figure 38. Each has a label to indicate to which AUTO-constant it corresponds. Most text fields can be activated by pointing the mouse cursor to it and clicking. When activating the text field for **ICP, NUZR, NTHL** and **NTHU**, another separate window will popup in which one can enter or modify certain data. They are shown in Figure 39.



NDIM	NBC	NINT	JAC	NTST	NCOL
1	2	1	0	15	4
IAD	EPSL	EPSU	EPSS	ITMX	NWTN
3	1e-06	1e-06	0.0001	8	3
ITNW	DS	DSMIN	DSMAX	IADS	NMX
5	0.01	0.001	0.05	1	50
RL0	RL1	A0	A1	NICP	ICP
-1e+10	1e+10	-1e+10	1e+10	1	1
ILP	ISP	ISW	MXBF	IRS	IPS
1	1	1	5	0	1
NPR	IID	IPLT	NUZR	NTHL	NTHU
50	2	0	0	1	0

Figure 38: The Full AUTO-constants Panel

The values of the AUTO-constants **JAC, NCOL, ILP, ISP, ISW, IPS** and **IID** are restricted. To prevent incorrect user choices, their text fields are not editable. In these cases, one has to press the RIGHT button of the mouse to activate an asociated popup menu in which a value can be selected by releasing the RIGHT button of the mouse on it. Figure 40 shows **JAC** as an example. At the bottom of the full AUTO-constants panel are four push buttons labeled **Ok, Apply, Cancel** and **Help**. Pressing the **Ok** button will save the current AUTO-constants in the panel and popdown the panel. Pressing the **Apply** button saves the AUTO-constants without panel popdown. Pressing the **Cancel** button will popdown the panel. Pressing the **Help** button will popup an item selection window in which all AUTO-constants are selectable. After an item is selected, one can press the **Ok** button at the bottom of

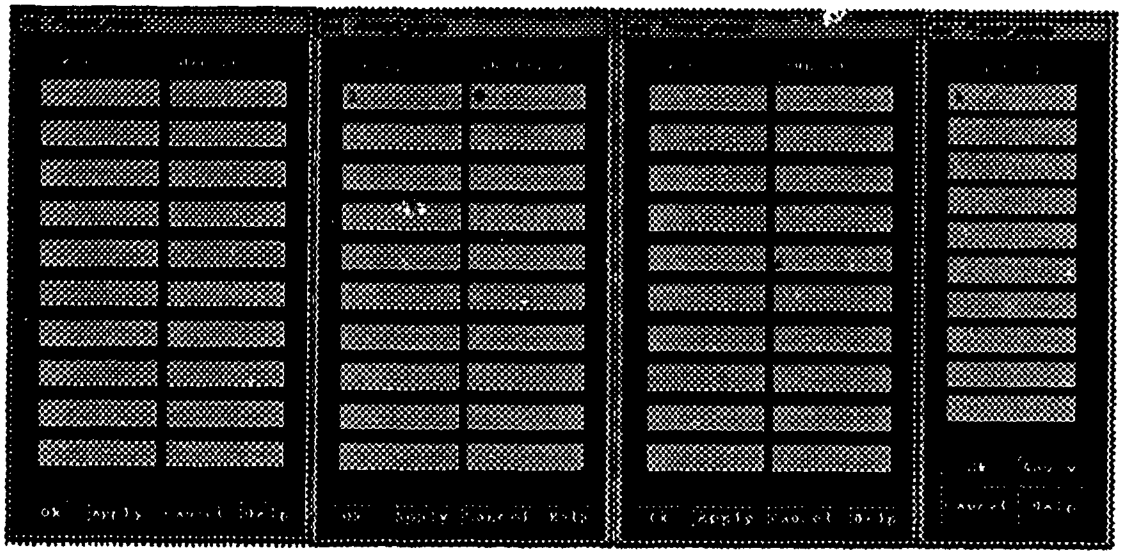
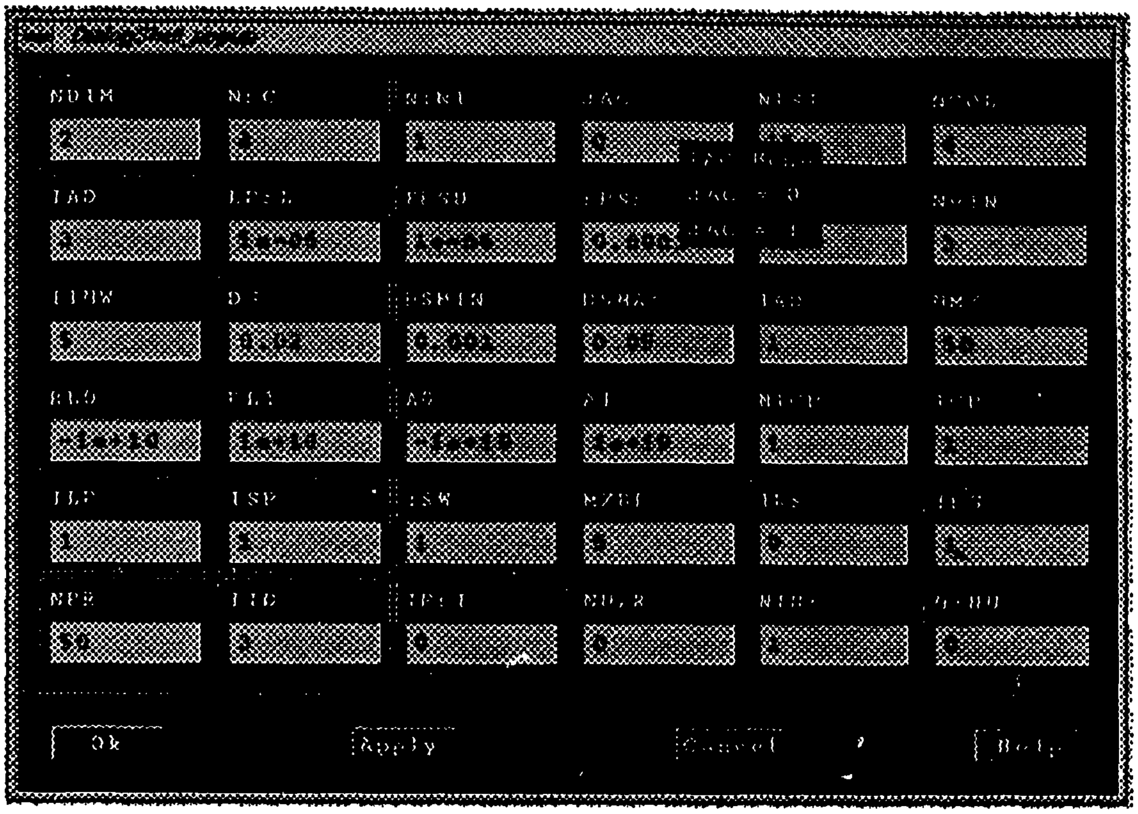


Figure 39: The Popup Windows for UZR, THL, THU and ICP

the selection window to display the requested information. To popdown the window containing the help information, press the **Cancel** button. The **Ok**, **Apply**, **Cancel** and **Help** buttons in the popup windows of **ICP**, **NUZR**, **NTHL** and **NTHU** have the same functionalities.



NDIM	NBC	NINT	JAC	NTST	NCOL
2	[illegible]	1	0	[illegible]	4
IAD	EPSL	EPSU	EPSS	[illegible]	NWTN
3	1e-06	1e-06	0.000	[illegible]	3
ITNW	DS	DSMIN	DSMAX	IADS	NMX
5	0.01	0.001	0.05	1	50
RL0	RL1	A0	A1	[illegible]	[illegible]
-1e+10	1e+10	-1e+10	1e+10	1	[illegible]
ILP	ISP	ISW	NTHL	[illegible]	[illegible]
1	2	1	0	0	[illegible]
NPR	IID	IPLT	NUZR	[illegible]	[illegible]
50	2	0	0	1	0

Figure 40: The Popup Menu for JAC in the Full AUTO-constants Panel

6.7.5 Run Menu

The **Run** menu contains no items except itself. It is a cascade button. Pressing **Run** will generate the AUTO-constants file as *r.name* where *name* is the current *name*, and start execution. If the AUTO-equations file has never been compiled or if it has been modified, then compilation will be done first. Before starting execution, the AUTO-equations file has to be written (**Write** button) and all AUTO-constants have to be defined.

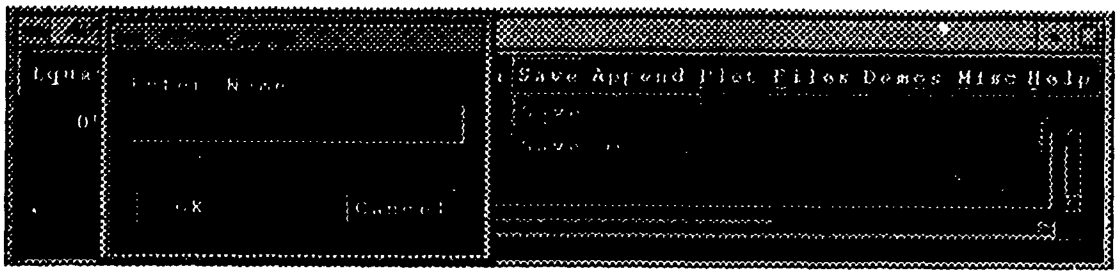


Figure 41: The Save Menu

6.7.6 Save Menu

The **Save** pulldown menu contains the two menu items **Save** and **Save As** as shown in Figure 41. Selecting the item **Save** will save the AUTO output files *fort.7, fort.8* and *fort.9* as *p.name, q.name* and *d.name*, respectively, where *name* is the current equation name. Selecting the item **Save As** will popup a prompt window in which one will be asked to enter a *name*. After the *name* is entered, *fort.7, fort.8* and *fort.9* will be saved as *p.name, q.name* and *d.name*, respectively.

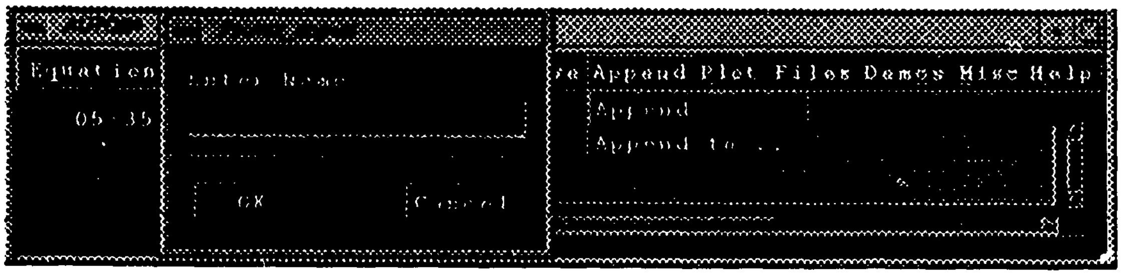


Figure 42: The Append Menu

6.7.7 Append Menu

The **Append** pulldown menu contains the two menu items **Append** and **Append to** as shown in Figure 42. Selecting the item **Append** will append the AUTO output

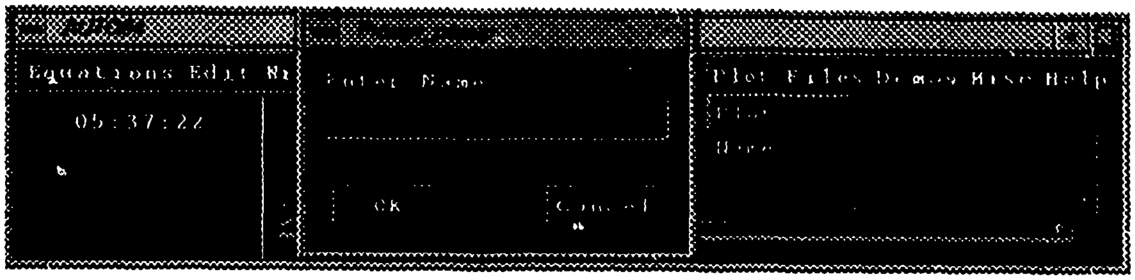


Figure 43: The Plot Menu

files *fort.7, fort.8* and *fort.9* to *p.name, q.name* and *d.name*, respectively, where *name* is the current equation *name*. Selecting the item **Append to** will popup a prompt window in which one will be asked to enter a *name*. After the *name* is entered, *fort.7, fort.8* and *fort.9* will be appended to *p.name, q.name* and *d.name*, respectively.

6.7.8 Plot Menu

The **Plot** pulldown menu contains the menu items **Plot** and **Name** as shown in Figure 43. Selecting the item **Plot** will activate the plotting program *PLAUT* for the data files *p.name* and *q.name*, where *name* is the current equation name. For example, Figure 44 shows a diagram for the demo *pp2*. Selecting the item **Name** will popup a prompt window in which one will be asked to enter a *name*. After the *name* is entered, the plotting program *PLAUT* will be activated for the files *p.name* and *q.name*.

6.7.9 Files Menu

The **Files** pulldown menu contains the menu items **Restart, Copy, Append, Move, Delete** and **Clean** as shown in Figure 45. Selecting the item **Restart** will popup a prompt window in which one will be asked to enter the restart *name*. For example, if one enters *xxx*, then the restart data for the immediately following run will be read from *q.xxx*. Selecting any one of the items **Copy, Append** and **Move** will popup a prompt window in which one will be asked to enter two *names*, say.,

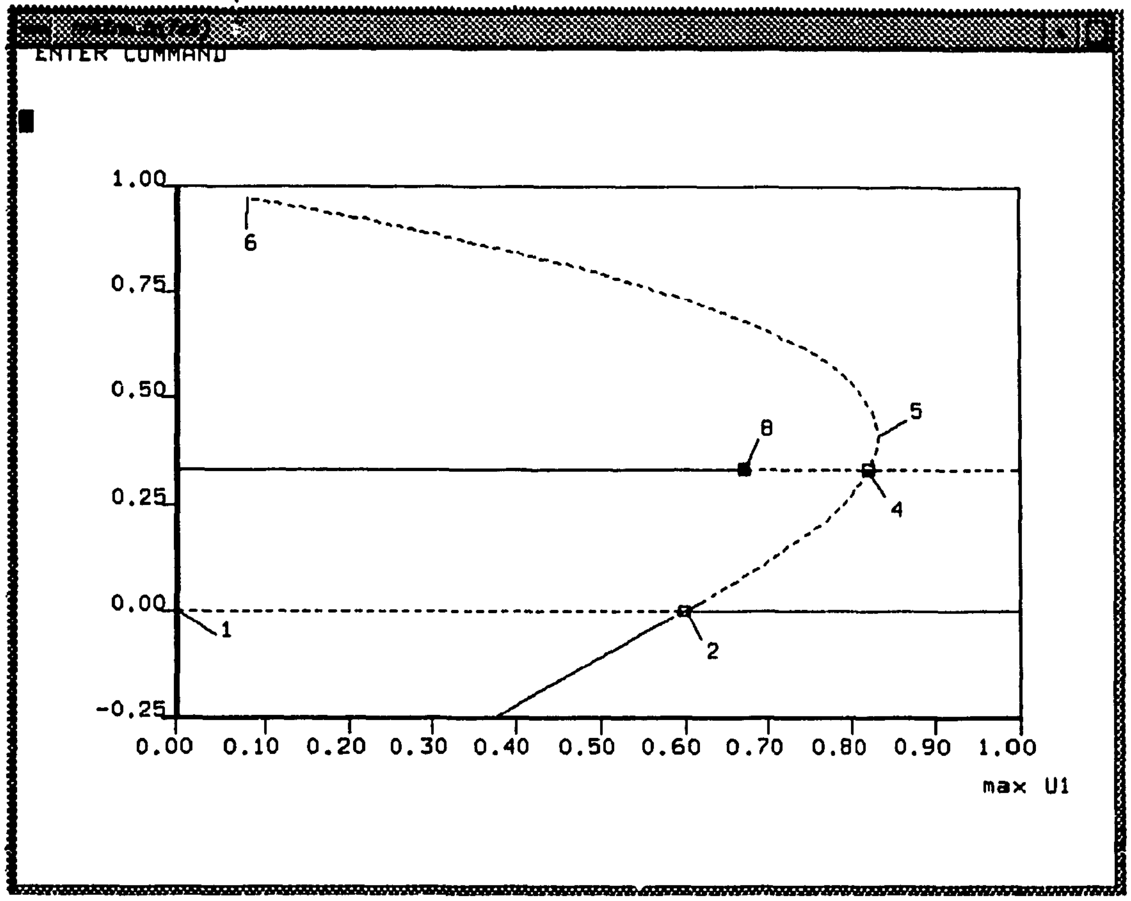

ENTER COMMAND

Figure 44: The Diagram for pp2.f

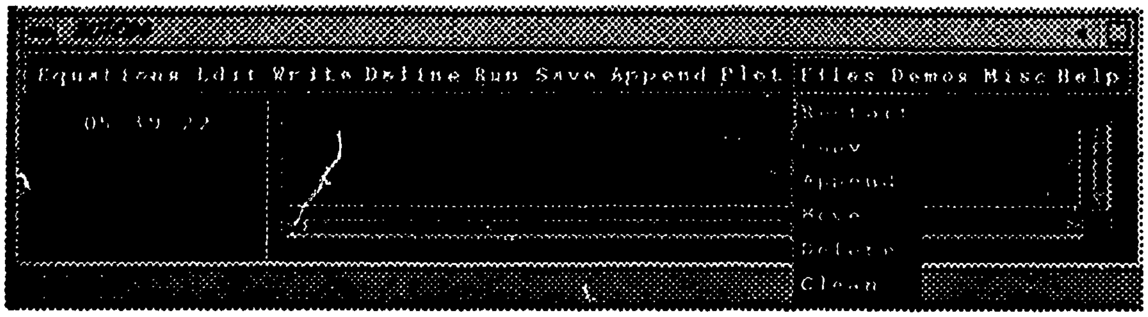

Equations Edit Write Define Run Save Append Plot Files Demos Misc Help
Restart
Copy
Append
Move
Delete
Clean

Figure 45: The Files Menu

name1 in the left text field and *name2* in the right text field, as shown in Figure 46. For convenience, the current *name* is always displayed as *name1* in the left text field. This current default *name* can be changed. Once *name1* and *name2* are entered, the AUTO data files *p.name1, q.name1* and *d.name1* will be copied, appended or moved to *p.name2, q.name2* and *d.name2*, respectively. The AUTO-constants file *r.name1* will be also copied or moved to *r.name2* for **Copy** or **Move**, respectively. Selecting the item **Delete** will popup a prompt window in which one can enter the *name* of the AUTO data files to be deleted. Once *name* is entered, *p.name, q.name* and *d.name* will be deleted. Selecting the item **Clean** will clean the current directory. More specifically, the files **.o, fort.** and **.exe* will be deleted.

6.7.10 Demos Menu

The **Demos** pulldown menu contains two menu items as shown in Figure 47, namely **Select** and **Reset.** Selecting the item **Select** will popup a selection window in which one can select one of the demos. After selection, one can either run or browse the selected demo by pressing the **Run** or **Browse** button at the bottom of the selection window. Pressing the **Browse** button will popup a window in which the source code of the selected demo can be paged. On-line help for each of the demos is given by pressing the **Help** button.

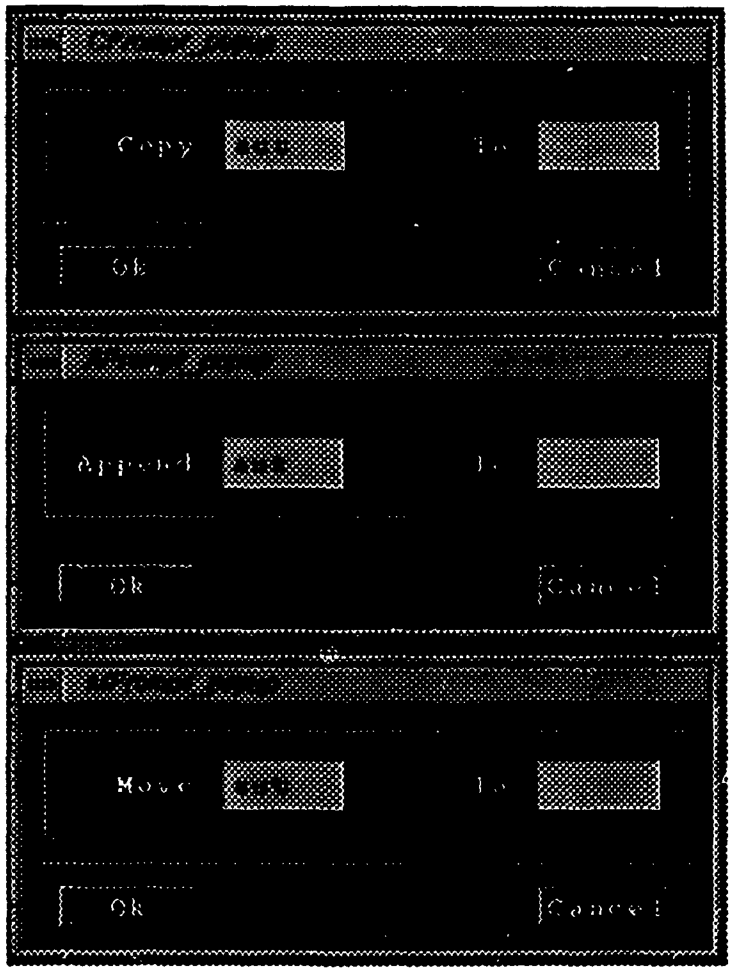


Figure 46: The Popup Windows for Copy, Append and Move

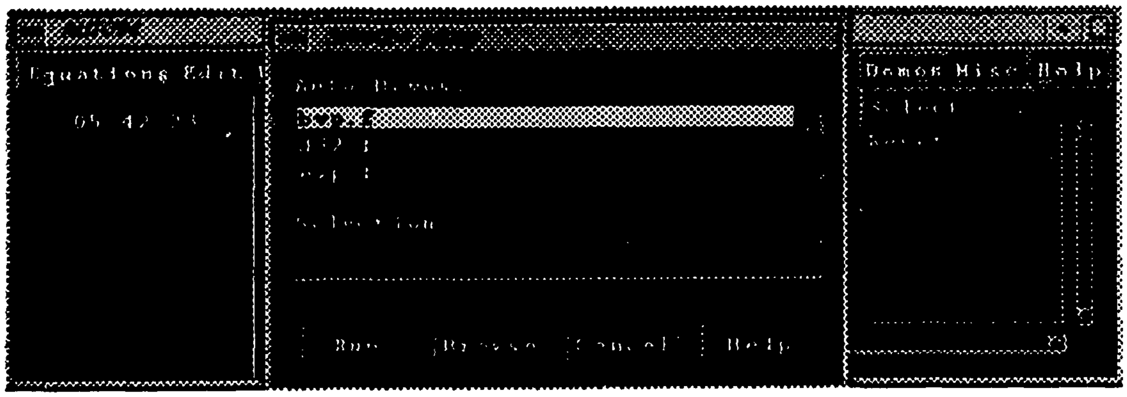


Figure 47: The Demos Menu

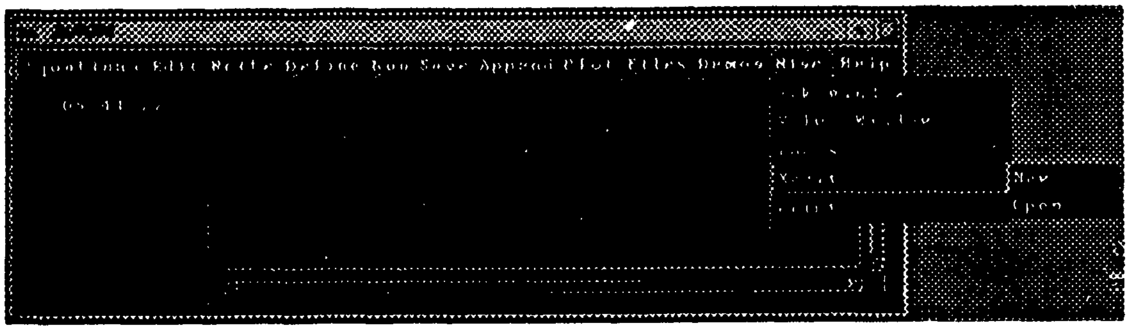


Figure 48: The Misc Menu

6.7.11 Misc Menu

The **Misc** pulldown menu contains the menu items **Tek Window, VT102 Window, Emacs, Xedit** and **Print** as shown in Figure 48. The items **Emacs** and **Xedit** contain sub-menus **New** and **Open**. Selecting **Tek Window** or **VT102 Window** will popup a Tektronix window or VT102 window, respectively. The Tektronix window can be used to run the AUTO plotting program PLAUT which requires a Tektronix window, the VT102 window can be used to issue commands or invoke the **vi** editor. **Emacs** and **Xedit** can be selected to invoke *emacs* and *xedit*, respectively. One can either edit a new file or an existing file by selecting the **New** or **Open** item in the sub-menus. Selecting the item **Print** will send the AUTO-equations file in the current AUTO-equations editor to the printer connected to the syst[illegible] The printing command is defined in the header file *GuiConsts.h* under *$AUTO_DIR/include* directory. This printing command may need to be changed for different systems. Files other than the AUTO-equations file in the current AUTO-equations editor can not be printed by selecting this **Print** item.

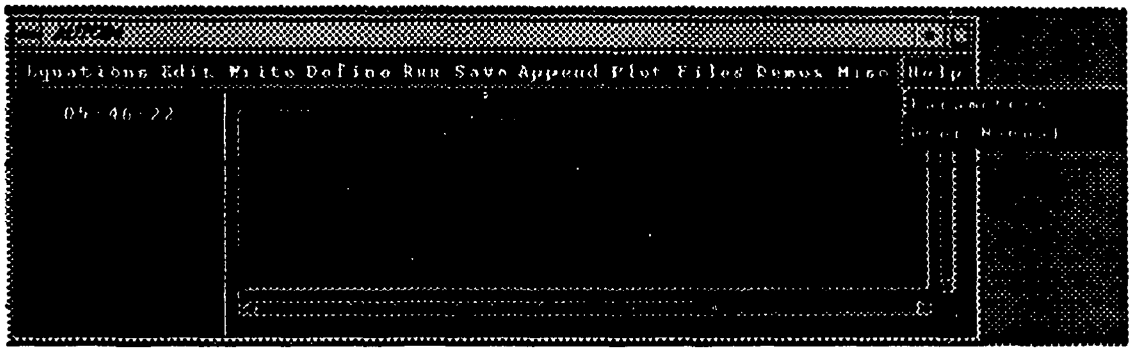

Equations Edit Write Define Run Save Append Plot Files Demos Misc Help
05:46:22
Parameters
User Manual

Figure 49: The Help Menu

6.7.12 Help Menu

The **Help** pulldown menu contains the menu items **Parameters** and **User Manual** as shown in Figure 49. Selecting **Parameters** will popup on-line help on AUTO-constants as already described in section 6.7.4. Selecting **User Manual** will allow the user to page through the AUTO94 user manual.

6.8 Functionalities of Other Buttons

On the main layout of the graphical user interface, there are a number of other push buttons in addition to those on the menu bar, namely, **Problem, Discretize, Tolerances, Step Size, Limits, Parameters, Computation, Output, Previous, Default, Stop** and **Exit**. In addition, there is an equation editor in which the

AUTO-equations file *name.f* can be edited. Their description follows below.

6.8.1 The Equation Editor

The equation editor is a full screen editor for editing the AUTO-equations *name.f*, the user-file containing the Fortran subroutines that define the equations. One can use the mouse to move the cursor around. Text in the editor can be paged up or down by pressing the key <PageUp> or <PageDown>, respectively. There are two scroll bars for the editor, namely, the vertical and the horizontal one. One can use them to scroll the text in the editor. Any portion of the text in the editor can be highlighted by positioning the mouse cursor at the beginning of the text and dragging the mouse to the end of the text while holding the LEFT mouse button. Text highlighted can be cut, copied or pasted by selecting one of the items in the **Edit** pulldown menu on the menu bar. In addition, the highlighted portion of the text can be cut or inserted by pressing the <Delete> or <Insert> key.

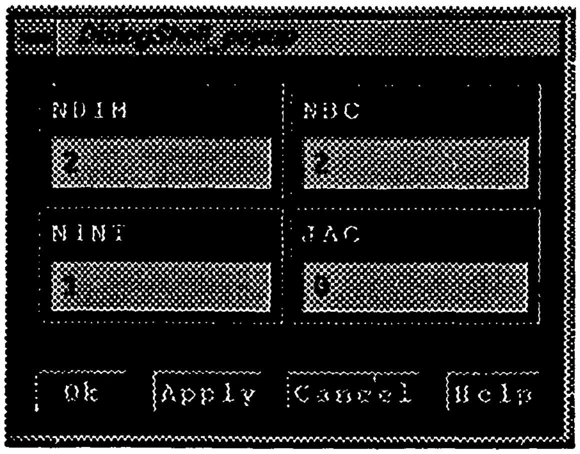

NDIM 2
NBC 2
NINT 1
JAC 0

Figure 50: The Problem Window

6.8.2 Problem Button

Pressing the **Problem** button will display the popup window shown in Figure 50, in which the AUTO-constants **NDIM, NBC, NINT** and **JAC** can be manipulated.

Pressing the **Problem** button by using the RIGHT button of the mouse will display a popup menu which shows what AUTO-constants can be modified by the popup window. The functionalities of the popup window are similar to that of the full AUTO-constants panel described in section 6.7.4, except that the current popup window can only manipulate AUTO-constants **NDIM, NBC, NINT** and **JAC.**

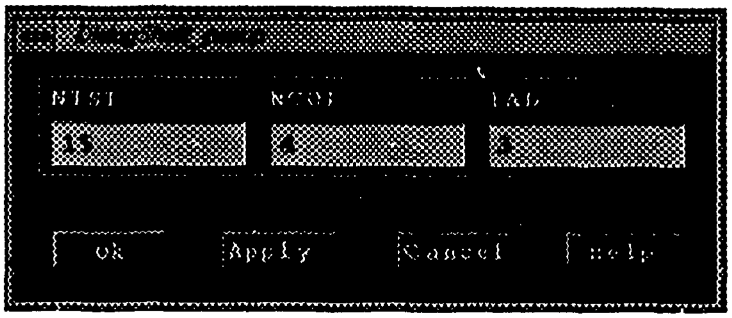


Figure 51: The Discretize Window

6.8.3 Discretize Button

Pressing the **Discretize** button will display the popup window shown in Figure 51, in which the AUTO-constants **NTST, NCOL** and **IAD** can be manipulated. Pressing the **Discretize** button by using the RIGHT button of the mouse will display a popup menu which shows what AUTO-constants can be modified by the popup window. The functionalities of the popup window are similar to that of the full AUTO-constants panel described in section 6.7.4, except that the current popup window can only manipulate the AUTO-constants **NTST, NCOL** and **IAD.**

6.8.4 Tolerances Button

Pressing the **Tolerances** button will display the popup window shown in Figure 52, in which the AUTO-constants **EPSL, EPSU, EPSS, ITMX, NWTN** and **ITNW** can be manipulated. Pressing the **Tolerances** button by using the RIGHT button of the mouse will display a popup menu which shows what AUTO-constants can be

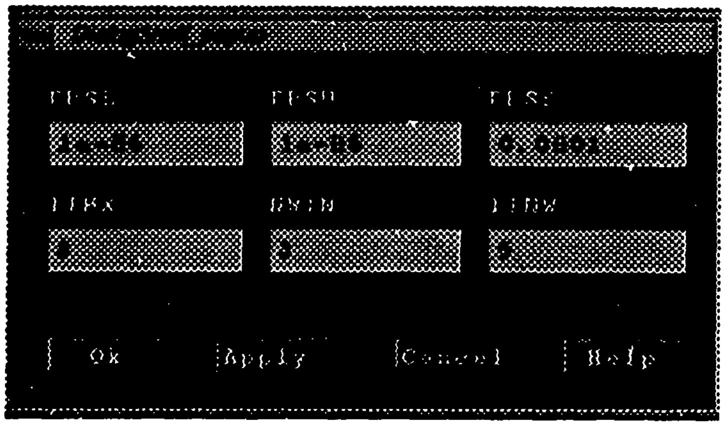


Figure 52: The Tolerances Window

modified by the popup window. The functionalities of the popup window are similar to that of the full AUTO-constants panel described in section 6.7.4, except that the current popup window can only manipulate the AUTO-constants **EPSL, EPSU, EPSS, ITMX, NWTN** and **ITNW.**

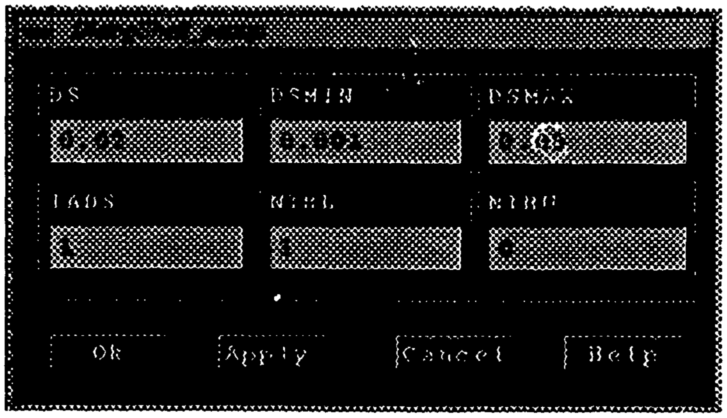


Figure 53: The Step Size Window

6.8.5 Step Size Button

Pressing the **Step Size** button will display the popup window shown in Figure 53, in which the AUTO-constants **DS, DSMIN, DSMAX, IADS, NTHL** and **NTHU** can be manipulated. Pressing the **Step Size** button by using the RIGHT button of the mouse will display a popup menu which shows what AUTO-constants can be modified by the popup window. The functionalities of the popup window are similar to that of the full AUTO-constants panel described in section 6.7.4, except that the current popup window can only manipulate the AUTO-constants **DS, DSMIN, DSMAX, IADS, NTHL** and **NTHU**.

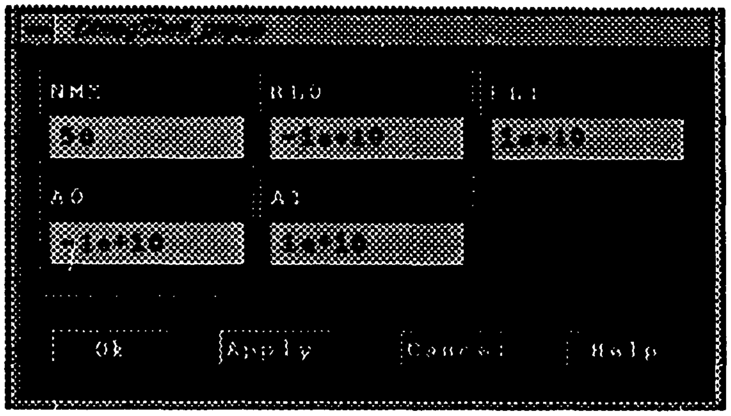


Figure 54: The Limits Window

6.8.6 Limits Button

Pressing the **Limits** button will display the popup window shown in Figure 54, in which the AUTO-constants **NMX, RL0, RL1, A0** and **A1** can be manipulated. Pressing the **Limits** button by using the RIGHT button of the mouse will display a popup menu which shows what AUTO-constants can be modified by the popup window. The functionalities of the popup window are similar to that of the full AUTO-constants panel described in section 6.7.4, except that the current popup window can only manipulate the AUTO-constants **NMX, RL0, RL1, A0** and **A1**.

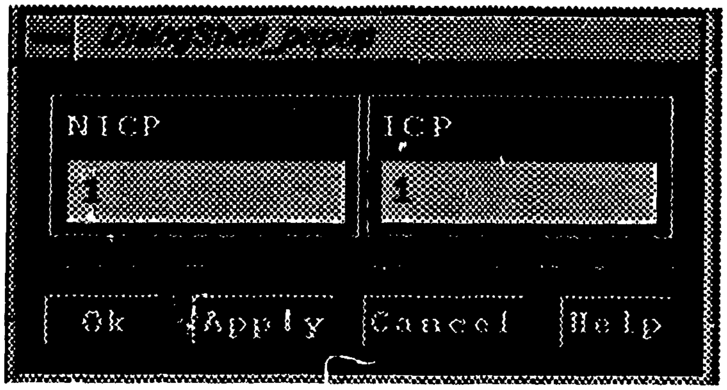


Figure 55: The Parameter Window

6.8.7 Parameters Button

Pressing the **Parameters** button will display the popup window shown in Figure 55, in which the AUTO-constants **NICP** and **ICP** can be manipulated. Pressing the **Parameters** button by using the RIGHT button of the mouse will display a popup menu which shows what AUTO-constants can be modified by the popup window. The functionalities of the popup window are similar to that of the full AUTO-constants panel described in section 6.7.4, except that the current popup window can only manipulate the AUTO-constants **NICP** and **ICP**.

6.8.8 Computation Button

Pressing the **Computation** button will display the popup window shown in Figure 56, in which the AUTO-constants **ILP, ISP, ISW, MXBF, IRS** and **IPS** can be manipulated. Pressing the **Computation** button by using the RIGHT button of the mouse will display a popup menu which shows what AUTO-constants can be modified by the popup window. The functionalities of the popup window are similar to that of the full AUTO-constants panel described in section 6.7.4, except that the current popup window can only manipulate the AUTO-constants **ILP, ISP, ISW, MXBF, IRS** and **IPS**.

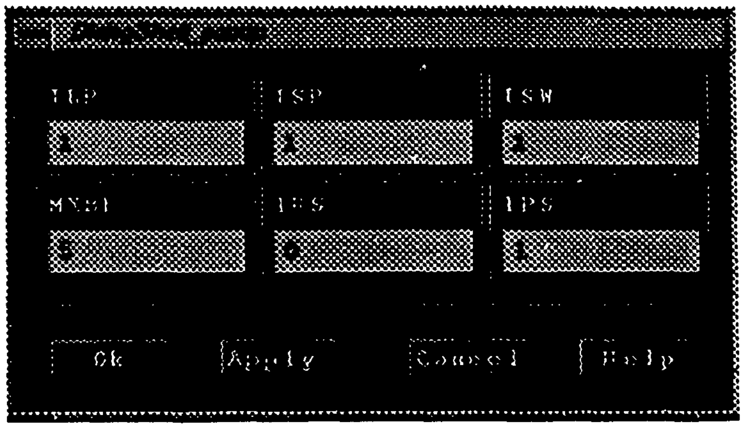


Figure 56: The Computation Window

6.8.9 Output Button

Pressing the **Output** button will display the popup window shown in Figure 57, in which the AUTO-constants **NPR, IID, IPLT** and **NUZR** can be manipulated. Pressing the **Output** button by using the RIGHT button of the mouse will display a popup menu which shows what AUTO-constants can be modified by the popup window. The functionalities of the popup window are similar to that of the full AUTO-constants panel described in section 6.7.4, except that the current popup window can only manipulate the AUTO-constants **NPR, IID, IPLT** and **NUZR**.

6.8.10 Previous Button

Pressing the **Previous** button will popup a file selector in which one can select any existing AUTO-constants file (*r.name*). The selected file will be loaded in all panel windows that manipulate AUTO-constants.

6.8.11 Default Button

The functionality of the **Default** button is similar to that of the **Previous** button, except that in this case, the default AUTO-constants file *r.aut* is loaded.

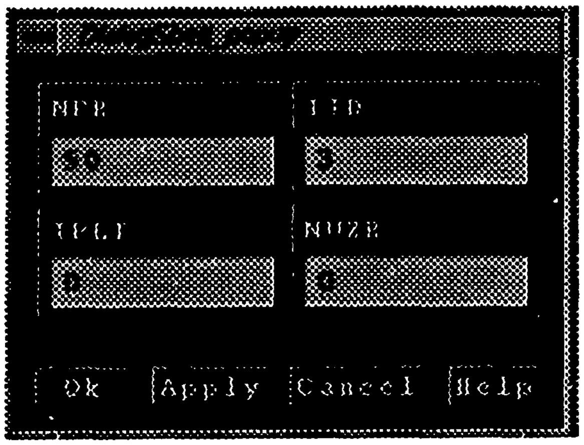

NPR
50
IID
3
IPLT
0
NUZR
0

Figure 57: The Output Window

6.8.12 Stop Button

Pressing the **Stop** button will terminate the execution of the running program.

6.8.13 Exit Button

Pressing the **Exit** button will popup a question window with a *Yes* or *No* choice for exiting from AUTO94.

6.9 Demos

In the **Demos** pulldown menu on the menu bar, pressing the item **Select** will popup a demo selector window. After selecting, for example, *exp.f*, press the **Run** button to execute all runs of the demo *exp*. The screen output is

```
        f77 -O -c exp.f
        f77 -O exp.o -o exp ../../lib/*.o
Demo exp.f is started
exp.f : first run
BR    PT TY LAB    PAR(1)         L2-NORM        MAX U(1)        MAX U(2)
```

```
 1    1 EP   1  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
 1    9 UZ   2  9.999996E-01  3.388826E-01  1.404988E-01  5.493525E-01
 1   12 UZ   3  3.000000E+00  1.521270E+00  6.401164E-01  2.319603E+00
 1   14 LP   4  3.513831E+00  2.781940E+00  1.186263E+00  4.000000E+00
 1   16 UZ   5  3.000000E+00  4.554630E+00  1.975141E+00  6.103383E+00
 1   22 UZ   6  9.999999E-01  9.157868E+00  4.089818E+00  1.084694E+01
 1   50 EP   7  2.239591E-05  3.677775E+01  1.723320E+01  3.720814E+01
TOTAL TIME    0.751E+01
exp.f : second run
BR   PT TY LAB    PAR(1)        L2-NORM       MAX U(1)      MAX U(2)
 1   50 EP   8  1.191140E-22  1.245716E+02  5.939144E+01  1.215573E+02
TOTAL TIME    0.192E+02
Demo exp.f is done
```

After execution, one can clean the selected demo by selecting the **Reset** item in the **Demos** pulldown menu in the menu bar. This will reset the selected demo to its original state. One can run the other demos similarly.

6.10 Restrictions on NICP, NUZR, NTHL and NTHU

AUTO94 has modifiable maximum values **NDIMX, NCOLX, NTSTX, NBCX, NINTX, NPARX, NBIFX** and **NUZRX**. for the AUTO-constants **NDIM, NCOL, NTST, NBC, NINT, NPAR, NBIF** and **NUZR**, respectively. These maximum values are defined in the header file *auto.h* under directory *$AUTO_DIR/include*. To change these maxima one should edit the file *auto.h* and recompile AUTO94. In addition to the above, the graphical user interface of AUTO94 independently has modifiable maximum values for the display of AUTO-constants **NICP, NUZR, NTHL** and **NTHU**. These maxima are defined by constants **MAX_NICP. MAX_NUZR, MAX_NTHL** and **MAX_NTHU** in the header file *GuiConsts.h* under directory *$AUTO_DIR/include*. The default maximum value is 10. To change this, one should edit the file *GuiConsts.h* and recompile AUTO94. In some application, the value of **NICP, NUZR, NTHL** or **NTHU** may be large than can be accommodated in the corresponding display windows. In such a case it is recommonded that the user d·ectly enter these values in the AUTO-constants file *r.name*, rather than by using the graphical user interface.

6.11 Keyboard Commands

A set of keyboard commands is also provided for terminals on which X windows are not available. These commands are defined in the file *@auto.alias* under the directory *$AUTO_DIR/cmds*. Their functionalities are described below. It is assumed that the working directory contains an AUTO-equations file, here *pp2.f*, and a corresponding AUTO-constants file, here *r.pp2*. These commands are described below.

@r - used to run AUTO94. For example, to run *pp2.f* with AUTO-constants file *r.pp2* and restart data file *q.pp2* if needed, type

@r pp2

- to run *pp2.f* with AUTO-constants file *r.pp2* and restart data file *q.xxx*, type

@r pp2 xxx

- to run *pp2.f* with AUTO-constants file *r.yyy* and restart data file *q.xxx*, type

@r pp2 xxx yyy

@sv - used to save the AUTO output data files *fort.7, fort.8* and *fort.9*. For example, to save the output data files of *pp2.f*, one can type the command

@sv pp2

This will save *fort.7, fort.8* and *fort.9* as ***p.pp2, q.pp2*** and ***d.pp2***, respectively.

@ap - used to append the latest AUTO output files *fort.7, fort.8* and *fort.9* to previously saved outputs, respectively. For example, to append the latest output files of *pp2.f* to previous output, one can type the command

@ap pp2

This will append *fort.7, fort.8* and *fort.9* to ***p.pp2, q.pp2*** and ***d.pp2***, respectively.

- to append the existing output data files *p.tmp,q.tmp,d.tmp*, to *p.pp2,q.pp2,d.pp2*, respectively, type

@ap tmp pp2

@p - used to plot the output files. For example, to plot the contents of the output data files *p.pp2* and *q.pp2*, type the command

@p pp2

- to plot *fort.7* and *fort.8*, type

@p

@cp - used to copy output data and AUTO-constants files. For example, to copy from *pp2* to *tmp*, one can type the command

@cp pp2 tmp

This will copy *p.pp2, q.pp2, d.pp2* and *r.pp2* to *p.tmp, q.tmp, d.tmp* and *r.tmp*, respectively.

@mv - used to rename previously saved output and AUTO-constants files. For example, to rename *pp2* to *tmp*, type the command

@mv pp2 tmp

This will rename *p.pp2, q.pp2, d.pp2* and *r.pp2* to *p.tmp, q.tmp, d.tmp* and *r.tmp*, respectively.

@dl - used to delete saved output files. For example, to delete the output files *pp2*, type the command

@dl pp2

This will delete *p.pp2, q.pp2* and *d.pp2*, respectively.

@df -used to delete *fort.** files.

@cl - used to clean object, executable and Fortran output files. The command **@cl** will delete **.o, *.exe* and *fort.**.

@dm - used to copy AUTO demo files to the current directory, for example, type

@dm pp2

to copy *pp2.f* and *r.pp2.** from the AUTO demo directory to the current directory.

6.12 Updating the On-line Help

All on-line help for the AUTO-constants and demos are defined in the header file *GuiGlobal.h* under directory *$AUTO_DIR/include*. These help messages can be updated. The graphical user interface has a modifiable maximum length for the help message. For Silicon Graphic platforms, the maximum message length is set to 10k or 10240 bytes. To change the maximum message length, for example, from 10k to 20k, one can simply replace the following line in the **Makefile** under *$AUTO_DIR/gui*

```
CC = cc -Wf,-XNl10240 -O
```

by

```
CC = cc -Wf,-XNl20480 -O
```

For other platforms, the maximum message length is restricted to the system defined maximum string literal length.

Chapter 7

Concluding Remarks

In this thesis, we studied two parallel algorithms for sparse linear systems in direct connection with AUTO94P. We stressed the communication schemes of these parallel algorithms. We also discussed some implementation issues. These two sparse linear algorithms were implemented in AUTO94P. More precisely, the condensation of parameters and its associated backsubstitution algorithms are almost fully parallelized, the nested dissection and its associated backsubstitution algorithms are implemented by recursive doubling procedures, and the relatively small system after nested dissection is solved sequentially. The parallel performance of AUTO94P is mainly determined by the algorithms for sparse linear systems. More precisely, the efficiency of AUTO94P increases as the dimension of the system increases, because when the size of the problem is big, the condensation of parameters algorithm dominates the total computation time, while the communication time remains relatively small. The parallel performance of AUTO94P also increases as the number of mesh or collocation points increases, since the number of mesh or collocation points determines the degree of parallelism or the granularity of the package. The performance of AUTO94 slowly decreases as the total number of computational nodes increases, because the nested dissection and its associated backsubstitution processes in the sparse solver are implemented by recursive doubling procedures, so that the number of messages in the communication is logarithmically increasing as one increases the number of computational nodes. AUTO94P has been implemented on some distributed memory systems. Since the total necessary memory is distributed over many processors, one can investigate much bigger size problems than before in an efficient way. The numerical results obtained on the Intel Delta support the above conclusion. For a 24 dimensional system of ordinary differential equations with 64 mesh points and 4 collocation points, the efficiency of AUTO94P is 80.67% for 16 nodes and 66.02% for 32 nodes on the Delta. To further enhance the parallel performance of AUTO94P, future work could aim at the scalability of the nested dissection process and its associated backsubstitution process, as this is one of the major factors that slowly decreases the parallel performance of AUTO94P. Other work could include optimizing memory management, I/O enhancement, pipelining and parallelizing the less critical parts of the package. The graphical user interface (GUI) of AUTO94 simplifies the use of the package. It provides the users with a convenient computational environment, in particular, it allows the users to run AUTO94 on a powerful remote machine and displays the results on a local workstation. Further developement of the GUI could aim at 3D visualization of numerical results.

References

[1] U. Ascher, J. Christiansen, and R. D. Russell. "Collocation software for boundary value ODE's". *ACM TOMS*, 7(2):209 pages, 1981.

[2] U. Ascher, R. M. M. Mattheij, and R. D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[3] U. Ascher and R. D. Russell. "Reformulation of boundary value problems into standard form". *SIAM Review*, 23(2):238–254, 1981.

[4] J. Browne, J. Dongarra, A. Karp, K. Kennedy, and D. Kuck. "1988 Gordon Bell Prize". *IEEE Software*, 6:78–85, May 1989.

[5] R. Bulirsch, J. Stoer, and P. Deuflhard. "Numerical Solution of Nonlinear Two Point Boundary Value Problems". *Numer. Math. Handbook Series Approx.*, 1976.

[6] D. Calahan. "Complexity of Vectorized Solution of Two Dimensional Finite Element Grids". Technical Report Systems Engineering Laboratory Report 91, Univ. Michigan, Ann Arbor, 1975.

[7] R. M. Chamberlain. "An Alternative View of LU Factorization with Partial Pivoting on a Hypercube Multiprocessor". In M. T. Heath, editor, *Hypercube Processors.* SIAM, Philadelphia, PA, 1987.

[8] K. Mani Chandy and Carl Kesselman. "Compositional C++: Compositional Parallel Programming". Technical Report Caltech-CS-TR-92-13, California Institute of Technology, Pasadena, CA 91125, July 1992.

[9] K. Mani Chandy and J. Misra. *Parallel Program Design, A Foundation.* Addison Wesley, 1988.

[10] K. Mani Chandy and Stephen Taylor. *An Introduction to Parallel Programming.* California Institute of Technology, 1992.

[11] E. Chu and J. A. George. "Gaussian Elimination with Partial Pivoting and Load Balancing on a Multiprocessor". *Parallel Computing*, 5:65–74, 1987.

[12] Intel Corporation. "Touchstone Delta Fortran System Calls Reference Manual". Technical report, Intel Corporation, 1991.

[13] Intel Corporation. "Touchstone Delta System User's Guide". Technical report, Intel Corporation, 1991.

[14] Germund Dahlquist and Ake Bjorck. *Numerical Methods.* Prentice-hall, Inc, Englewood Cliffs, New Jersey, 1974.

[15] D.Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, Englewood Cliffs, 1989.

[16] Eric F. Van de Velde. "Experiments with Multicomputer LU-Decomposition". Technical Report CRPC-89-1, California Institute of Technology, Pasadena, CA 91125, April 1989.

[17] Eric F. Van de Velde. "Multicomputer Matrix Computations: Theory and Practice". Technical Report CRPC-89-2, California Institute of Technology, Pasadena, CA 91125, March 1989.

[18] Eric F. Van de Velde. *Introduction to Concurrent Scientific Computing.* California Institute of Technology, DRAFT, 1992.

[19] Henk A. Van der Vorst and Paul van Dooren. *Parallel Algorithms for Numerical Linear Algebra (Volume I).* North-Holland, Amsterdam,New York, Oxford, Tokyo, 1990.

[20] E. J. Doedel. "AUTO: A Program for the Automatic Bifurcation Analysis of Autonomous Systems". *Proc. 10th Manitoba Conf. On Num. Math. and Comp.*, pages 265–284, 1980.

[21] E. J. Doedel. "AUTO: Software for Continuation and Bifurcation Problems in Ordinary Differential Equations". Technical report, Applied Mathematics, California Institute of Technology, Pasadena, CA 91125, May 1986.

[22] E. J. Doedel, H. B. Keller, and J. P. Kernévez. "Numerical Analysis and Control Of Bifurcation Problems, Part I". *Int. J. Bifurcation and Chaos*, 3:493–520, 1991.

[23] E. J. Doedel, H. B. Keller, and J. P. Kernévez. "Numerical Analysis and Control Of Bifurcation Problems, Part II". *Int. J. Bifurcation and Chaos*, 4:745–772, 1991.

[24] J. J. Dongarra and L. Johnson. "Solving Banded Systems on a Parallel Processor". *Parallel Computing, North-Holland*, 5:219–246, 1987.

[25] J. J. Dongarra and A. H. Sameh. "On Some Parallel Banded System Solvers". *Parallel Computing, North-Holland*, 1:223–235, 1984.

[26] Omer Egecioglu, Cetin K. Koc, and Alan J. Laub. "A Recursive Doubling Algorithm for Solution of Tridiagonal Systems on Hypercube Multiprocessors". *Journal of Computational and Applied Mathematics*, 27:95–108, 1989.

[27] K. Fong and T. Jordan. "Some Linear Algebraic Algorithms and Their Performance on the CRAY-1". Technical Report LA-6774, Los Alamos National Laboratory, Los Alamos, NM, 1977.

[28] Ian Foster and Steven Tuecke. "Parallel Programming with PCN". Technical Report ANL-91/32, Argonne National Laboratory, Argonne, IL 60439, September 1991.

[29] Ian T. Foster and K. Mani Chandy. "FORTRAN M: A Language for Modular Parallel Programming". Technical Report Preprint MCS-P327-0992, Argonne National Laborary, Argonne, IL 60439, June 1992.

[30] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems on Concurrent Processors.* Prentice Hall, 1988.

[31] K. Gallivan, R. Plemmons, and A. Sameh. "Parallel Algorithms for Dense Linear Algebra Computations". *SIAM Review*, 32:54–135, 1990.

[32] D. Gannon. "A Note on Pipelining a Mesh-connected Multiprocessor for Finite Element Problem by Nested Dissection". *1980 Int. Conf. Par. Proc.*, pages 197–204, 1980.

[33] G. A. Geist and M. T. Heath. "Matrix Factorization on a Hypercube Multiprocessor". In M. T. Heath, editor, *Hypercube Processors.* SIAM, Philadelphia, PA, 1986.

[34] G. A. Geist and C. H. Romine. "LU Factorization Algorithms on Distributed Memory Multiprocessor Architectures". *SIAM Journal on Scientific and Statistical Computing*, 9(4):639–649, 1988.

[35] W. Gentleman. "Error Analysis of the QR Decomposition by Givens Transformations". *Lin. Alg. Appl.*, 10:189–197. 1975.

[36] A. George. "Nested Dissection of a Regular Finite Element Mesh". *SIAM J. Numer. Anal.*, 10:345–363, 1973.

[37] A. George and J. W. H. Liu. "An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems". *SIAM J. Numer. Anal.*, 15:1053–1069, 1978.

[38] A. George and J. W. H. Liu. "The Evolution of the Minimum Degree Ordering Algorithm". *SIAM Review*, 31:1-19, 1989.

[39] A. George, W. Poole, and R. Voigt. "Analysis of dissection algorithms for vector computers". *Comput. Math. Appl.*, 4:287–304, 1978.

[40] Alan George, Michael Heath, Joseph Liu, and Esmond Ng. "Solution of Sparse Positive Definite Systems on a Hypercube". *Journal of Computational and Applied Mathematics*, 27:129–156, 1989.

[41] G. H. Golub and C. F. Van Loan. *Matrix Computation*. Johns Hopkins Baltimore, MD, 1983.

[42] Anshul Gupta and Vipin Kumar. "Scalability of Parallel Algorithms for Matrix Multiplication". Technical Report Technical Report TR 91-54, Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, November 1991.

[43] Michael T. Heath, Esmond Ng, and Barry W. Peyton. "Parallel Algorithms for Sparse Linear Systems". *SIAM Review*, 33(3):420–460, 1991.

[44] D. Heller. "A Survey of Parallel Algorithms in Numerical Linear Algebra". *SIAM Review*, 20:740–777, 1978.

[45] Jorn Hofhaus and Eric F. Van de Velde. "Multicomputer Programs for Solving a Large Number of Block Tridiagonal Systems". to appear.

[46] Fritz John. *Partial differential equations (4th edition)*. Springer-Verlag, New York, Heidelberg, Berlin, 1982.

[47] H. B. Keller. "Numerical solution of bifurcation and nonlinear eigenvalue problems". In *Applications of Bifurcation Theory*, pages 359–384. Academic Press, New York, N.Y., 1977. P. H. Rabinowitz, ed., Mathematics Research Center Publication 8.

[48] Herbert B. Keller. "Domain Decomposition for Two-point Boundary Value Problems". Technical Report CRPC-90-7, California Institute of Technology, Pasadena, CA 91125, August 1990.

[49] Ken Kennedy. "Center for Research on Parallel Computation: An Introduction". *Parallel Computing Research*, 1:1–6, January 1993.

[50] M. Kubiček and M. Marek. *Computational Methods in Bifurcation Theory and Dissipative Structures.* Springer Verlag, 1983.

[51] S. Kumar and J. Kowalik. "Parallel Factorization of a Positive Definite Matrix on MIMD Computers". *1984 Int. Conf. Par. Proc.*, pages 410–416, 1984.

[52] J. Lambiotte. *"The Solution of Linear System of Equations on a Vector Computer"*. PhD thesis, Univ. Virginia, Charlottesville, Viginia, 1975.

[53] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays. Trees . Hypercubes.* Morgan Kaufmann Publishers, San Mateo, California, 1992.

[54] M. Lentini and V. Pereyra. "'An Adaptive Finite Difference Solver for Nonlinear Two Point Boundary Problems with Mild Boundary Layers'. *SIAM J. Numer. Anal.*, 14, 1977.

[55] J. W. H. Liu. "Modification of the Minimum Degree Algorithm by Multiple Elimination". *ACM Trans. Math. Software*, 11:141–153, 1985.

[56] C. B. Moler. "Matrix Computation on a Hypercube Multiprocessor". In M. T. Heath, editor, *Hypercube Processors.* SIAM, Philadelphia, PA, 1986.

[57] Mo Mu and J. R. Rice. "A Grid-based subtree-subcube assignment strategy for solving partial differential equations on hypercubes". *SIAM J. Sci. Stat. Comput.*, 13(3):826–839, May 1992.

[58] Adrian Nye and Tim O'Reilly. *X Volume 4: X Toolkit Intrinsics Programming Manual.* O'Reilly & Associates Inc, 1990.

[59] Adrian Nye and Tim O'Reilly. *X Volume 6: Motif Programming Manual.* O'Reilly & Associates Inc, 1993.

[60] James M. Ortega and Robert G. Voigt. "Solution of Parallel Differential Equations on Vector and Parallel Computers". *SIAM Review*, 27(2):149–237, 1985.

[61] OSF. *OSF/Motif: Programmer's Guide.* Open Software Foundation, 1991.

[62] OSF. *OSF/Motif: Programmer's Reference.* Open Software Foundation, 1991.

[63] Marcin Paprzyck and Ian Gladwell. "Solving Almost Block Diagonal Systems on Parallel Computers". *Parallel Computing, North-Holland*, 17:133–153, 1991.

[64] Marcin Paprzycki. "A Parallel Chopping Algorithm for ODE Boundary Value Problems". To appear.

[65] V. Pereyra. "Deferred Corrections Software and its Application to Seismic Ray Tracing". In K. Bohmer and H. Stetter, editors, *Defect Correction Methods*, volume 5, pages 211–226. Computing Suppl., 1984.

[66] D. Rose. "A Graph Theoretic Study of the Numerical Solution of Sparse Positive Definite Systems of Linear Equations". In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.

[67] Rand J. Rost. *X and Motif Quick Reference Guide.* Digital Press, 1990.

[68] A. Sameh and D. Kuck. "On Stable Parallel Linear System Solvers". *J. ACM*, 25:81–91, 1978.

[69] Charles L. Seitz, Jakov Seizovic, and Wen-King Su. "The C Programmer's Abbreviated Guide to Multicomputer Programming". Technical Report Caltech-CS-TR-88-1, California Institute of Technology, Pasadena, CA 91125, January 1989.

[70] B. Smith, J. Boyle, J. Dongarra, B. Garbow, Y. Ikebe, Klema, and C. Moler. "Matrix Eigensystem Routines: EISPACK Guide". Technical report, Springer-Verlag, 1976. Lecture Notes in Computer Science 6.