AUTO94P: An Experimental Parallel Version of AUTO

Eusebius Doedel
Xianjun Wang

CRPC-TR95599
May 1995

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Revised: January, 1996. Also available as *CRPC-95-3* from the Center for Research on Parallel Computation at the California Institute of Technology.

AUTO94P : An Experimental Parallel Version of AUTO

X. J. Wang and E. J. Doedel

January 19, 1996

CRPC-95-3
Center for Research on Parallel Computing
California Institute of Technology
Pasadena CA 91125

Abstract

A detailed description is given of the parallel algorithms used in AUTO94P, an experimental parallel version of the software AUTO for the numerical bifurcation analysis of systems of ordinary differential equations. Timing results and user instructions for the Intel Delta are included. The sequential version of the software, AUTO94, is fully described in [8]. For a related tutorial paper see [5, 6].

1 Introduction

In this report we give a detailed presentation of the parallel algorithms used in AUTO94P, an experimental version of the software package AUTO for the bifurcation analysis of systems of ordinary differential equations. The latest version of the standard sequential software, AUTO94, is described in [8], where user instructions and many illustrative examples are given. A description of the numerical algorithms used in AUTO, as well as related algorithms, can be found in [5, 6]. To obtain a copy of AUTO94 or for information on AUTO94P send email to doedel@cs.concordia.ca.

We consider the first order system of ordinary differential equations [6].

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

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. An efficient sequential numerical algorithm for solving the above ODE system is described in [6]. The corresponding algorithm was implemented in AUTO86 [7] as well as in the more recent AUTO94 [8]. More precisely, define a mesh

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

and for each j introduce the Lagrange basis polynomials

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

defined by

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

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 [5, 6, 11] 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

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

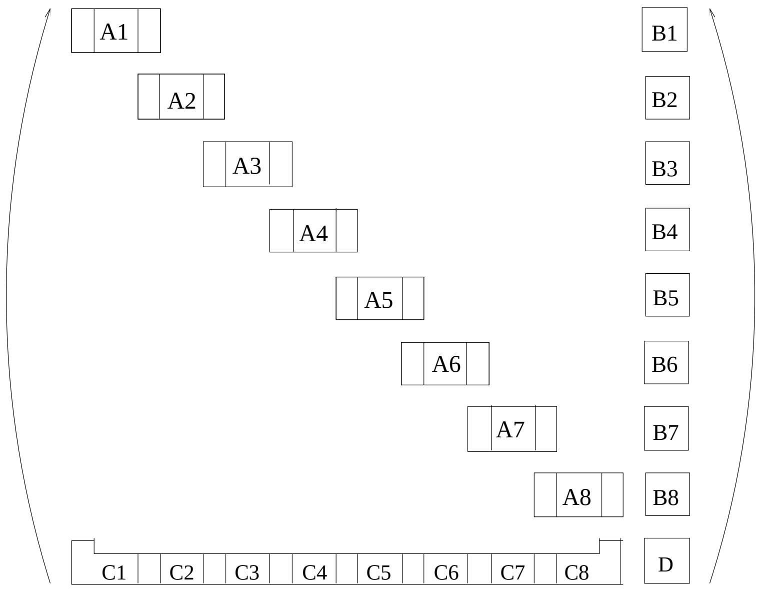

Figure 1: Structure of the Jacobian matrix J

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 via Newton's method the matrix J in Figure 1 is obtained. This matrix 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 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, both on sequential and parallel machines.

We shall only consider direct methods here, as opposed to iterative methods. Direct methods are generally more efficient and robust for the case of ordinary differential equations. More importantly, our direct solvers produce asymptotic-stability and bifurcation information as a by-product. In Section 2 a parallel direct solver for the linearized system (13) that does not use pivoting is described. Without a pivoting strategy the algorithms simplify considerably. However, in practical problems pivoting is frequently necessary to maintain numerical stability. Thus, in Section 3, the parallel algorithm is generalized to include pivoting. Both Sections 2 and 3 include timing results. Some implementation issues are addressed in Section 4, and details on how to use AUTO94P are given in Section 5.

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

Table 1: Outline of the Parallel Algorithm without Pivoting

2 A Parallel Sparse Solver without Pivoting

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. In this section we do not consider any pivoting strategy in the Gauss elimination process. Pivoting will be considered in Section 3. The communication scheme will be illustrated in graphic charts for the special case of 8 processors, although it applies to any number of processors. The outline of the algorithm is shown in Table 1.

2.1 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 influenced by it. Consider the sparse linear system (13), whose Jacobian matrix J is shown in Figure 1, with right hand side $f = (F_1, F_2, \cdots, F_8, FC)^T$. Assume the total number 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 dense matrix LU-decomposition, we partition the Jacobian matrix J and the right

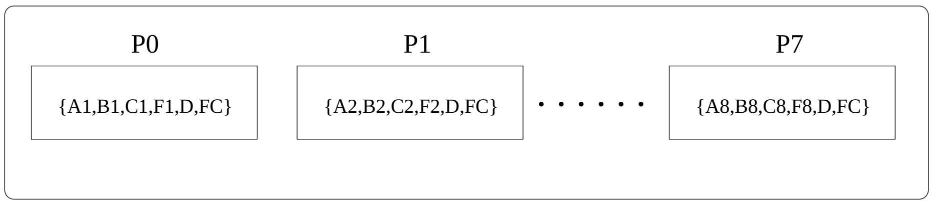

Figure 2: Data Distribution for the Sparse System

hand side f into P data groups. Each data group contains k (here $k = 1$) data units. In case the number of data units is not divisible by P, some data groups contain $k + 1$ data units. Thus the difference between any two data groups is at most one data unit. The best case is no difference at all, but we cannot always get perfect balance. The distribution of the Jacobian matrix J in Figure 1 and the right hand side f is shown in Figure 2. Note that the matrix D and FC in Figure 1 are shared by all processors. This means that each processor keeps a copy of D and FC.

2.2 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. The Gauss elimination procedure for the Jacobian matrix J can be applied concurrently in each of the processors, since no communication occurs until the bottom of the sparse matrix J is reached. 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. 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 3. Note that there are only two startup times for the entire communication. This communication is scalable [10] 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

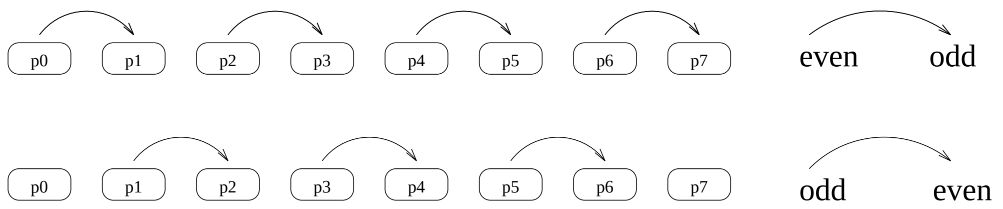

Figure 3: Communications between Odd and Even Nodes

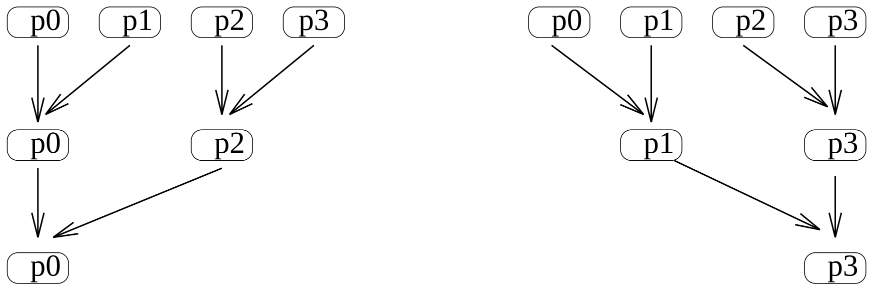

Figure 4: Recursive Doubling for 4 Nodes

precisely, assume the total number of nodes is P, and define the following set

$$I_P = \{0, 1, 2, \cdots, P-1\} \quad (14)$$

$$L_P = \{1, 2, 3, \cdots, log_2\ P\} \quad (15)$$

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 2 outlines the communication between neighboring nodes. To update D and FC, a global sum is needed. This is done by a recursive doubling procedure [4], which takes $log_2\ P$ steps, where P is the total number of nodes. The recursive doubling procedure is shown in Figure 4, where only four nodes are used for simplicity. Depending on the outcome of the final sum, we use two types of recursive doubling. One is where the

```
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 2: Neighboring Node Communications

left most node holds the final results, as shown in the left part of Figure 4, 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 4, where p_3 holds the final results. To outline the recursive doubling procedure in our implementation, define the set $B = \{T, F\}$ and maps M_{even}, M_{odd} such that

$$M_{odd} : I_P \times L_P \longrightarrow B \tag{16}$$

$$M_{even} : I_P \times L_P \longrightarrow B \tag{17}$$

More precisely,

$$M_{odd}(i_p, l_p) = \begin{cases} T & \text{if } mod(iam_p, 2) = 0 \\ F & \text{if } mod(iam_p, 2) = 1 \end{cases} \tag{18}$$

$$M_{even}(i_p, l_p) = \begin{cases} T & \text{if } mod(iam_p, 2) = 1 \\ F & \text{if } mod(iam_p, 2) = 0 \end{cases} \tag{19}$$

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

$$N_{odd}(i_p, l_p) = \begin{cases} i_p + 2^{l_p-1} & \text{if } i_p + 2^{l_p-1} \in I_P \\ \emptyset & \text{otherwise} \end{cases} \tag{20}$$

```
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 3: The Recursive Doubling Algorithm

$$N_{even}(i_p, l_p) \quad = \quad \begin{cases} i_p - 2^{l_p - 1} & \text{if } i_p - 2^{l_p - 1} \in I_P \\ \emptyset & \text{otherwise} \end{cases} \tag{21}$$

where $\emptyset$ means undefined. Table 3 shows the outline of the recursive doubling in our implementation. After condensation of parameters, the Jacobian matrix J in Figure 1 becomes the one shown in Figure 5. The shaded areas in Figure 5 will be considered below in the nested dissection process. Table 4 is an outline of the condensation of parameters process.

2.3 Nested Dissection

After condensation of parameters a nested dissection [9] procedure is used. Here we consider nested dissection without pivoting. The communication scheme for the pivoting case is different, and described in Section 3. Extracting the shaded areas in Figure 5, we obtain the matrix shown in Figure 6. Here A_1, A_2, C_1, C_2 and B are in one processor. The notation C_2/C_1 in Figure 6 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

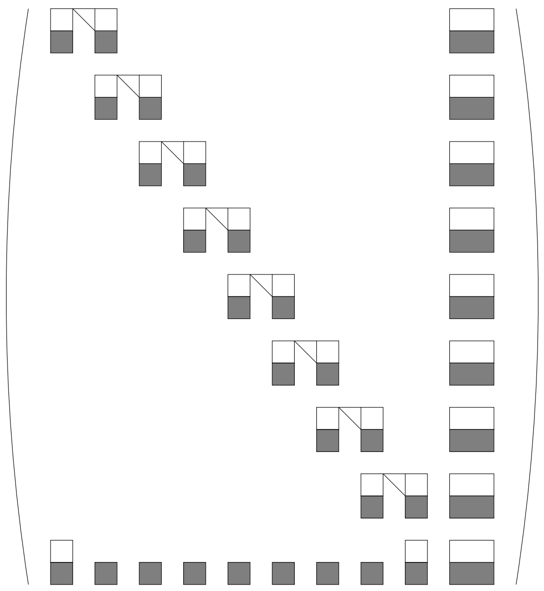

Figure 5: The Jacobian J after Condensation of Parameters

```
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 4: Outline of the Condensation of Parameters without Pivoting

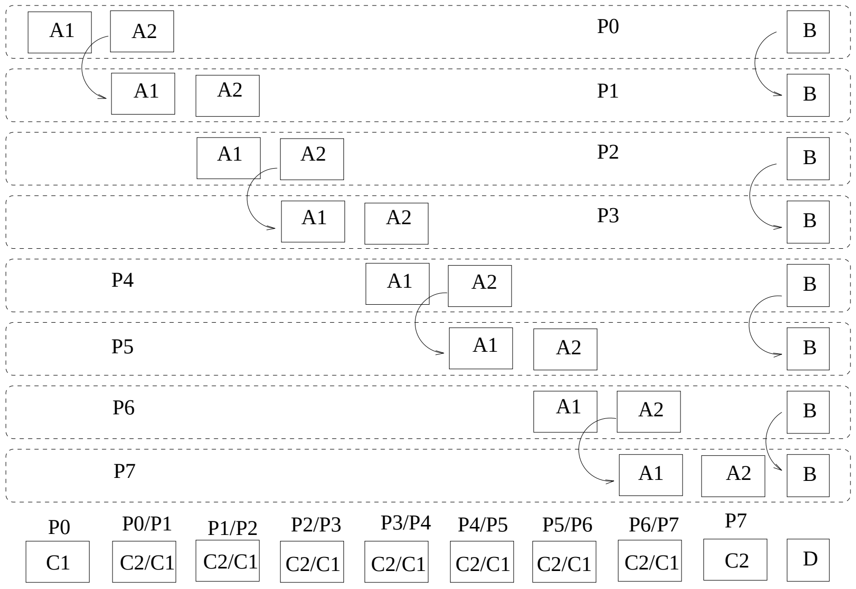

Figure 6: Initial State of the Nested Dissection

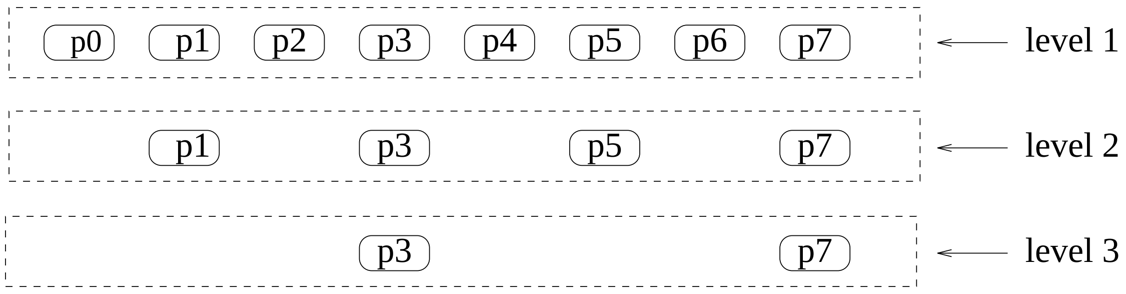

Figure 7: Busy Nodes during Each Recursion Level

of the recursion levels is 3. Figure 7 shows the busy nodes for each recursion level. At the first level, the communication is between two neighboring nodes. For example, in Figure 6, 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 8, 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 7. 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 8 shows the communication scheme. After the elimination at level 2, the matrix is shown in Figure 9. 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 has been completed, is shown in Figure 10.

2.4 Solving the Small System

The shaded area in Figure 10 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

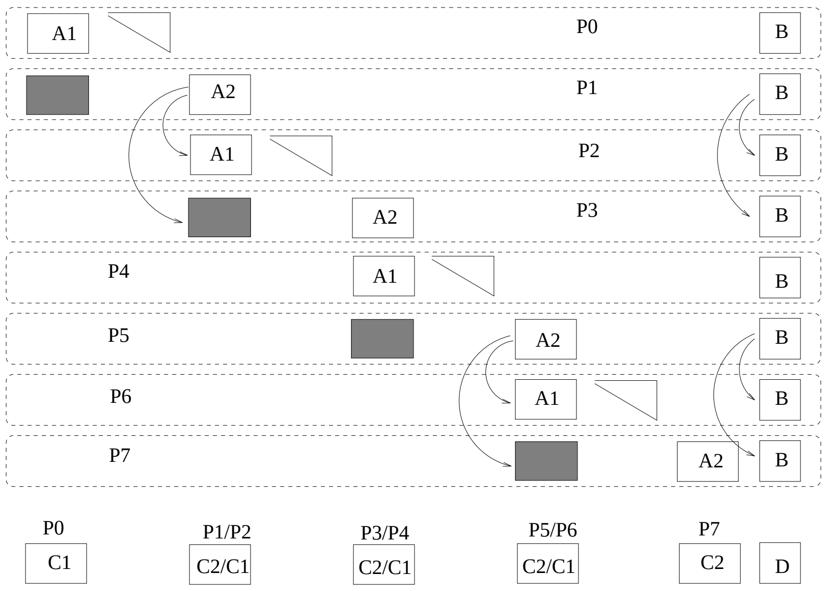

Figure 8: Initial State of Level 2

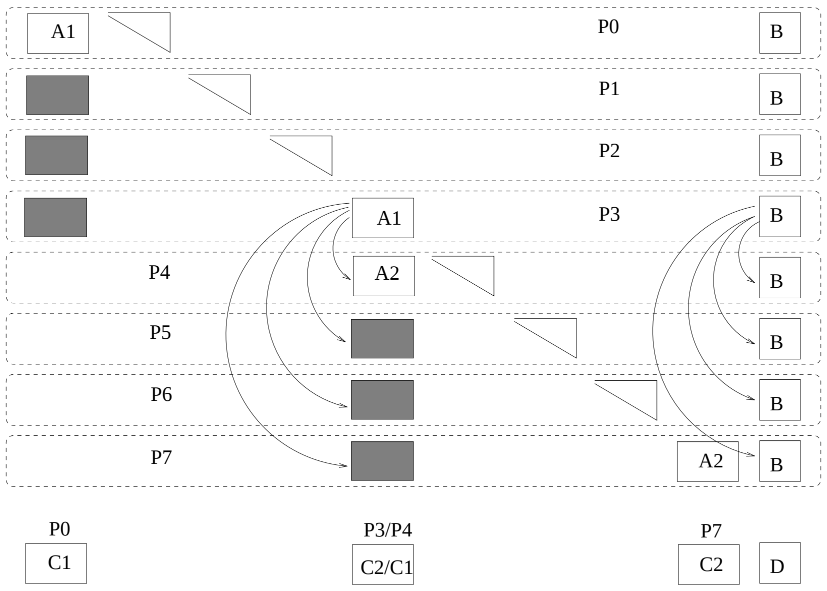

Figure 9: Initial State of Level 3

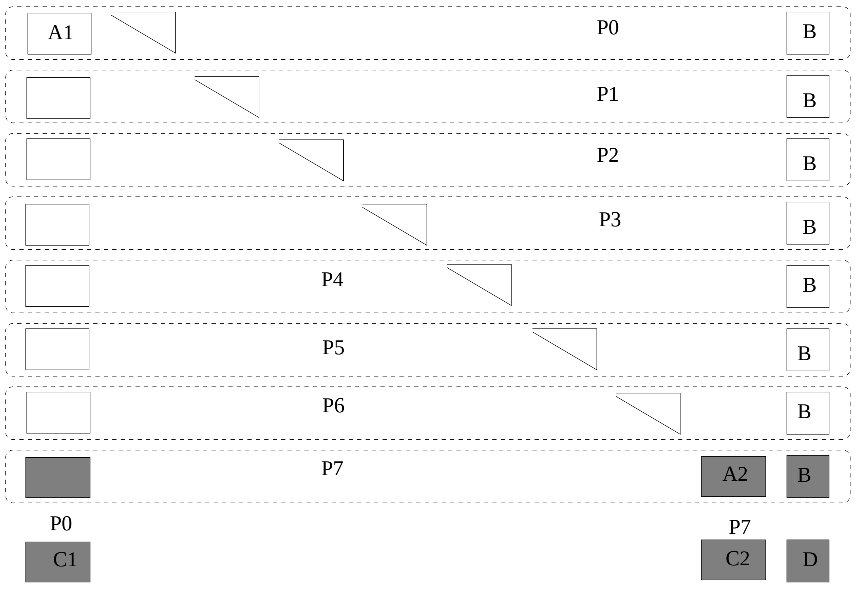

Figure 10: Final State after the Nested Dissection

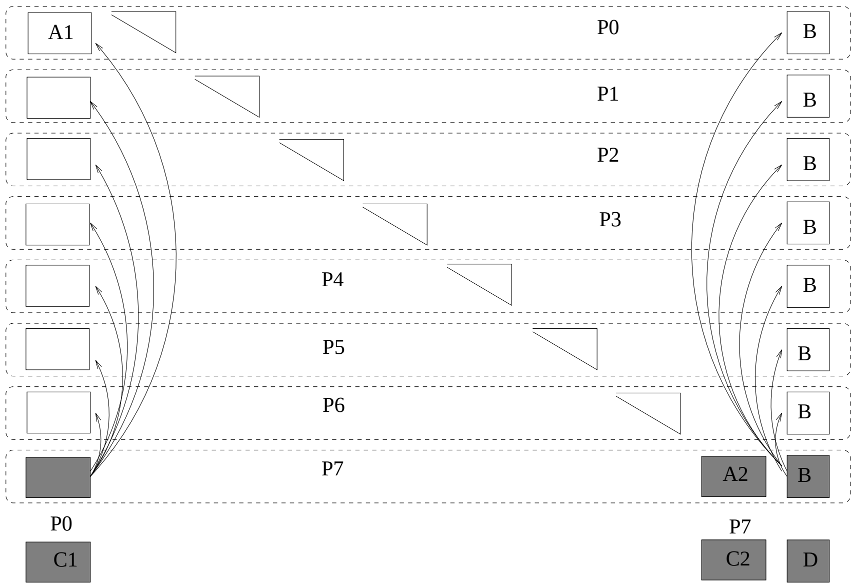

Figure 11: Broadcast Solution from the Small System

p_0 in Figure 10 has to be sent to node p_7, before node p_7 can compute the solution of the small system.

2.5 Backsubstitution

After solving the small system, we can obtain the solution for the full matrix shown in Figure 6 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 11 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 (13), another backsubstitution has to be done. This final backsubstitution process does not need 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 Section 3.7.

2.6 Timing Results

AUTO94P without pivoting has been tested on the Intel Hypercube and Mesh machines. The numerical timing results reported here were obtained on the Intel Delta; 512 Intel iPSC/860 nodes connected by a mesh network. The example used for the numerical experiments is *tim.f*, a demo in AUTO94, which is useful 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 NDIMX [8]. The system of the equations is

$$u_1' = u_2 \tag{22}$$

$$u_2' = -\lambda \sum_{i=0}^{25} \frac{u_1^i}{i!} \tag{23}$$

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

$$u_1(0) = 0 \tag{24}$$

$$u_1(1) = 0 \tag{25}$$

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 (26) and the relative efficiency formula (27) below to measure the performance. Assume that the execution time for one node is T_1 and T_p for P nodes. We define the efficiency, denoted by η, as follows

$$\eta = \frac{T_1}{PT_p} \tag{26}$$

and the relative efficiency as

$$\eta_r = \frac{mT_m}{nT_n}, \tag{27}$$

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

2.6.1 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 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 5: 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 6: Without Pivoting 1: NDIM=24, NTST=64, NCOL=4, NMX=10

2.6.2 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 8. 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

$$T_{average} = \frac{T_{minimum} + T_{maximum}}{2} \tag{28}$$

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

In Table	Minimum I/O time	Maximum I/O time	Average I/O time
9	1.00	3.50	2.25
10	1.00	3.50	2.25
11	3.00	7.00	5.00

Table 8: 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 9: 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 10: 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 11: Without Pivoting 2: NDIM=48, NTST=64, NCOL=4, NMX=10

```
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 12: Outline of the Parallel Algorithm with Pivoting

3 A Parallel Sparse Solver with Pivoting

The top level of the parallel algorithm outlined in Table 12 is similar to that in Section 2.

We show the parallel design of each part of Table 12, except that we do not repeat the partitioning strategy. We present the communication schemes in graphic charts with emphasis on pivoting. To avoid duplication, we do not repeat parts similar to those in Section 2, particularly for the condensation of parameters. Although the ideas are illustrated for the case of 8 processors, the communication scheme applies to any number of processors.

3.1 Pivoting Strategy

To enhance numerical stability [1, 12], we use restricted row and column pivoting in our parallel algorithm. During the condensation of parameters process, we use a pivot search window at each elimination step. In Figure 12, the pivot window for A_i is the shaded area. The corresponding pivot window in nested dissection is shown in Figure 13. It needs to be mentioned here that the pivot window in the nested dissection does not reside in a single node, but rather resides either in two neighboring nodes or in two nodes at distance 2^k, where k is the current recursion level. Thus, during any recursion level of the nested dissection, say level k, message passing between

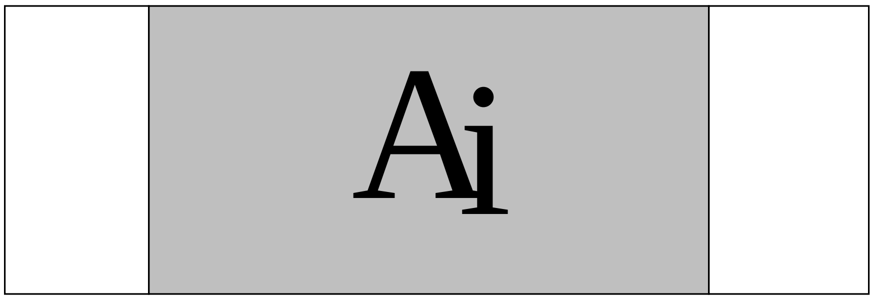

Figure 12: The Pivot Window for A_i

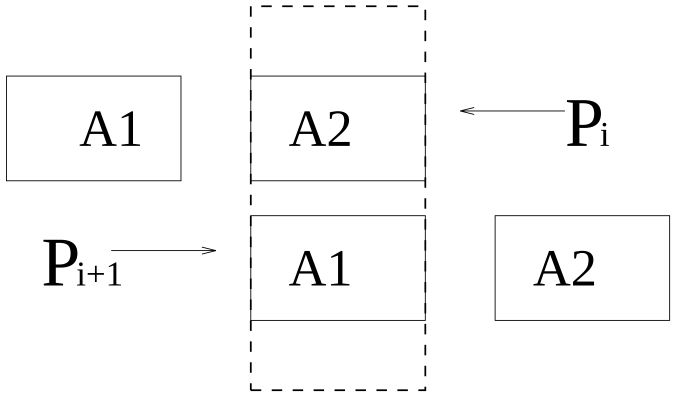

Figure 13: The Pivot Window for Nested Dissection

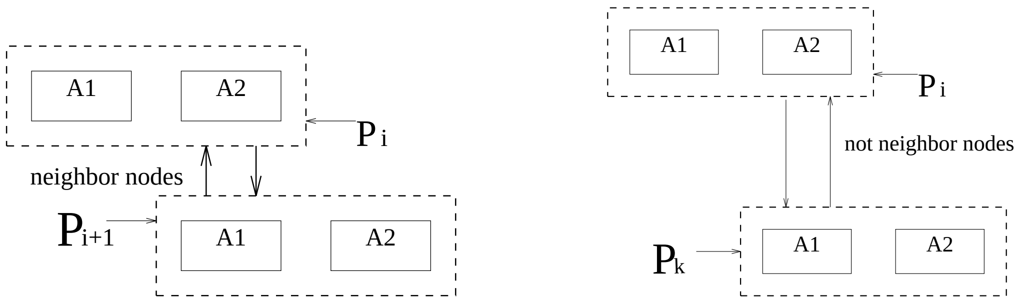

Figure 14: Communication within the Pivot Window

two nodes at distance 2^k is needed. The information exchanges are indicated in Figure 14. The efficiency of the above pivoting strategy is obviously better than complete row and column pivoting, due to the restriction of the search region.

3.2 Condensation of Parameters

Starting from the initial state of the Jacobian matrix, as shown in Figure 1, condensation of parameters transforms the matrix into the form shown in Figure 15. As in Section 2, 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 Section 2.2. The difference here is that, during the elimination process in each node, a local pivot within the pivot window mentioned in section 3.1 is searched. In addition, the communications at the bottom of the Jacobian matrix J are delayed until the nested dissection process in order to improve the degree of parallelism. 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:

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

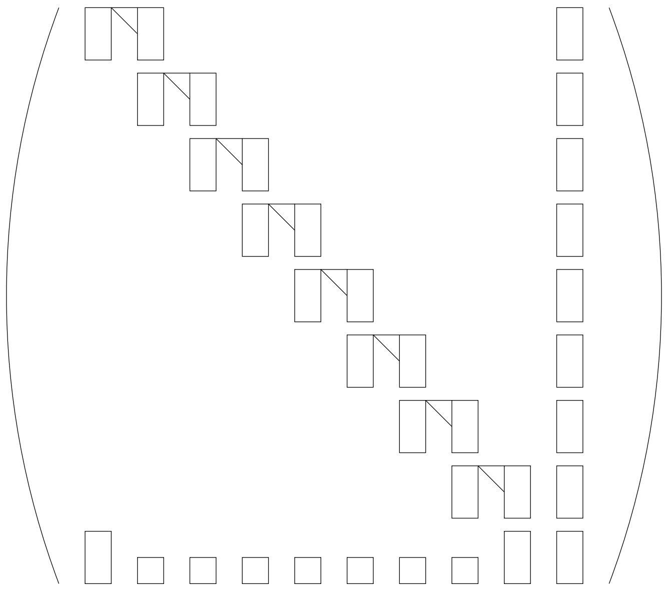

Figure 15: The Jacobian J after Condensation of Parameters

```
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 13: Outline of the Condensation of Parameters With Pivoting

Here P is the total number of nodes. Table 13 is an outline of the condensation of parameters process.

3.3 Nested Dissection

The nested dissection 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

$$\begin{aligned} B &= \{T, F\} && (29) \\ PID &= \{pid_i \quad | \quad i \in I_P\} && (30) \\ & && (31) \end{aligned}$$

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

$$M_{si} : PID \longrightarrow I_P \tag{32}$$

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

$$M_i : I_P \times L_P \longrightarrow B \quad i = 1, 2 \tag{33}$$

We define the two maps as follows.

$$M_1(i_p, l_p) = \begin{cases} T & \text{if } mod(iam_p, 2) = 0 \\ F & \text{if } mod(iam_p, 2) = 1 \end{cases} \tag{34}$$

$$M_2(i_p, l_p) = \begin{cases} T & \text{if } mod(iam_p, 2) = 1 \\ F & \text{if } mod(iam_p, 2) = 0 \end{cases} \tag{35}$$

where, as before,

$$iam_p = \left\{ \frac{i_p}{2^{l_p - 1}} \right\} \cap I_P \tag{36}$$

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

$$N_i : I_P \times L_P \longrightarrow I_P \tag{37}$$

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

$$N_1(i_p, l_p) = \begin{cases} i_p + 2^{l_p - 1} & \text{if } i_p + 2^{l_p - 1} \in I_P \\ \emptyset & \text{otherwise} \end{cases} \tag{38}$$

$$N_2(i_p, l_p) = \begin{cases} i_p - 2^{l_p - 1} & \text{if } i_p - 2^{l_p - 1} \in I_P \\ \emptyset & \text{otherwise} \end{cases} \tag{39}$$

where $\emptyset$ means undefined. Now we can describe the algorithm in two parts. Part one is outlined in Table 14 and part two in Table 15. Below we describe our communication mechanism graphically. The nested dissection is carried out in a way similar to the recursive doubling procedure described

```
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 14: Nested Dissection Process 1

before. During each level of recursion, half of the total number of nodes are idle. Assume that the total number of *data units*, defined in Section 2.1, is 24. After the data distribution over 8 nodes described in Section 2 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 14. The elimination in each node is sequential. For now, we don't consider shared data, such as the matrix D and the vector FC. The result of the elimination is shown in Figure 16 to Figure 18. 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 15. After each node finishes the first part of the nested dissection outlined in Table 14, extracting the unfinished part shown in Figure 19 in each node, we have the situation shown in Figure 20. Since we have 8 nodes, the number of recursion levels is 3. In Figure 20, 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 21 and 22. In total we need 3 startup times to complete the

```
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 15: Nested Dissection Process 2

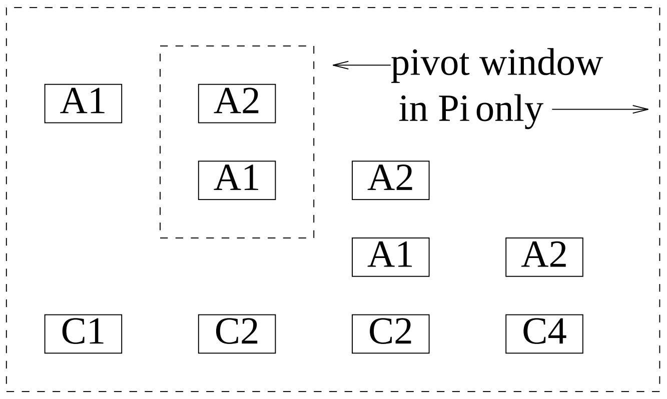

Figure 16: The Initial State in Node p_i

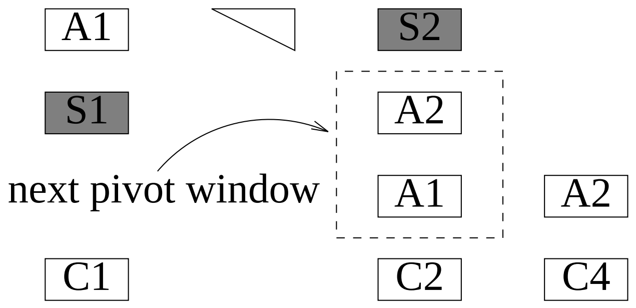

Figure 17: The Intermediate State in Node p_i

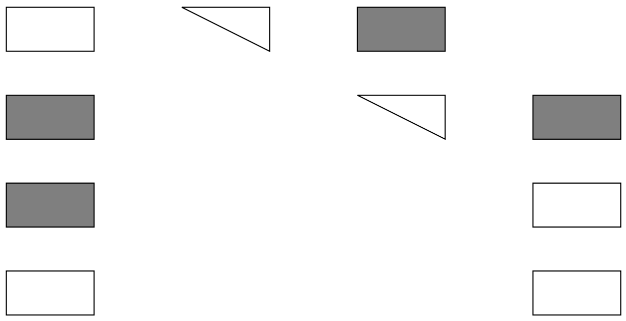

Figure 18: The Final State in Node p_i

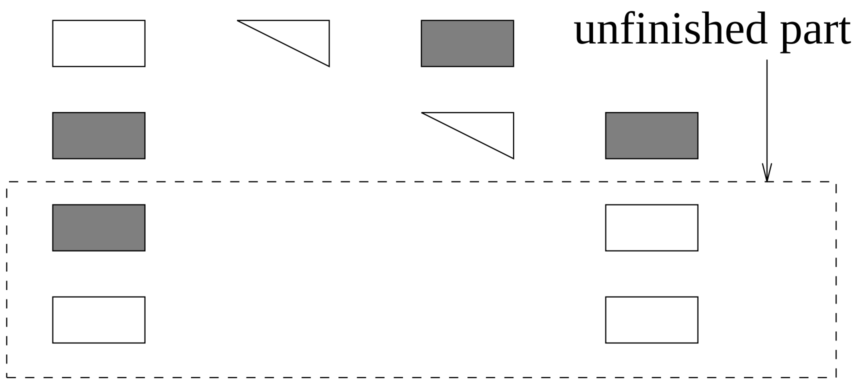

Figure 19: Enclosed in the Dashed-line Box is the Unfinished Part

elimination for 8 nodes. The elimination result and the communications are indicated for each level of the recursive process in Figure 20 to Figure 22. The final result is indicated in Figure 23. The shared data, such as the matrix D and the vector FC, are updated by a global sum similar to Figure 4. Note that only the last node holds the global sum of D and FC, as will be explained below.

3.4 Solving the Small System

After the nested dissection, we need to solve a relatively small square system described in Section 2.4. In Figure 23, we send C_1 from node p_0 to node p_7 as indicated in Figure 24. We can extract the square system enclosed by a big dashed line box as indicated in Figure 25 in node p_8. The shaded area implicitly represents the Poincaré map that is needed in AUTO [6]. 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 26. The outline for this is in Table 16.

3.5 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 27 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 maps $M_i(i_p, l_p), i = 1, 2$ are defined as in (34). Define sets $Nb_i(i_p, l_p) \subset I_P, i = 1, 2$ such that

$$\begin{aligned} Nb_1(i_p, l_p) &= \{i_p \mid M_1(i_p, l_p - 1) = T \quad and \quad l_p > 1\} \qquad (40) \\ Nb_2(i_p, l_p) &= \{i_p \mid M_1(i_p, l_p + 1) = T \quad and \quad l_p < P - 1\} \qquad (41) \end{aligned}$$

The backsubstitution process is now outlined in Table 17. The backsubsti-

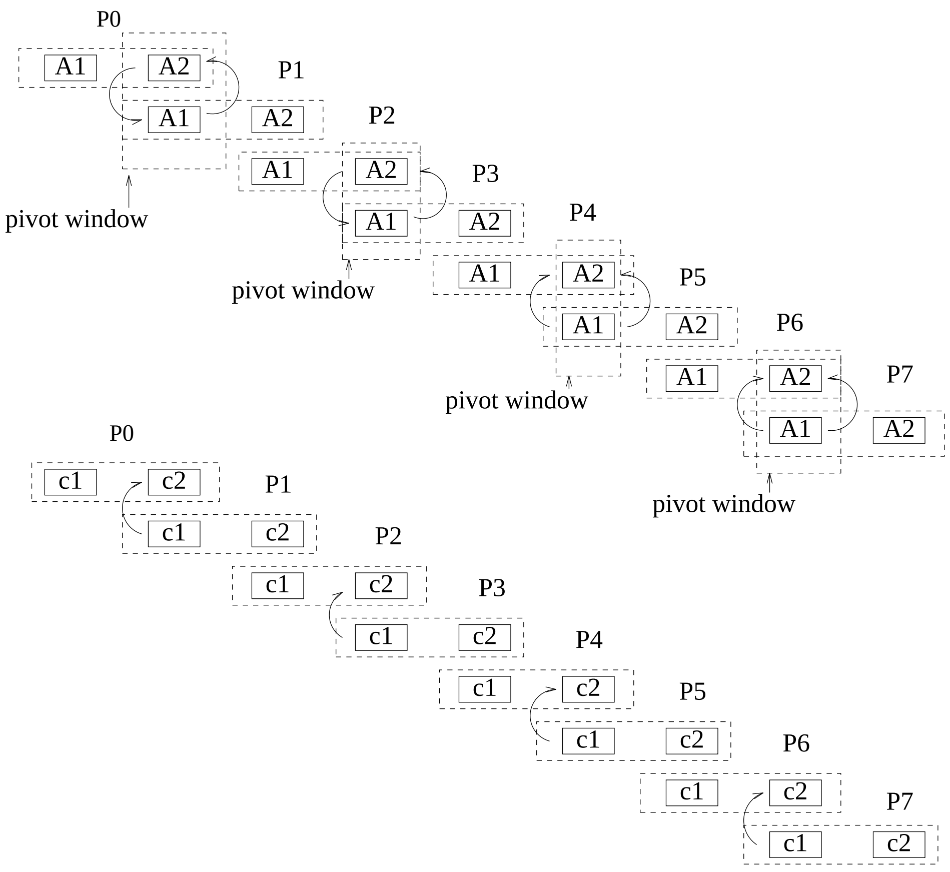

Figure 20: Level 1

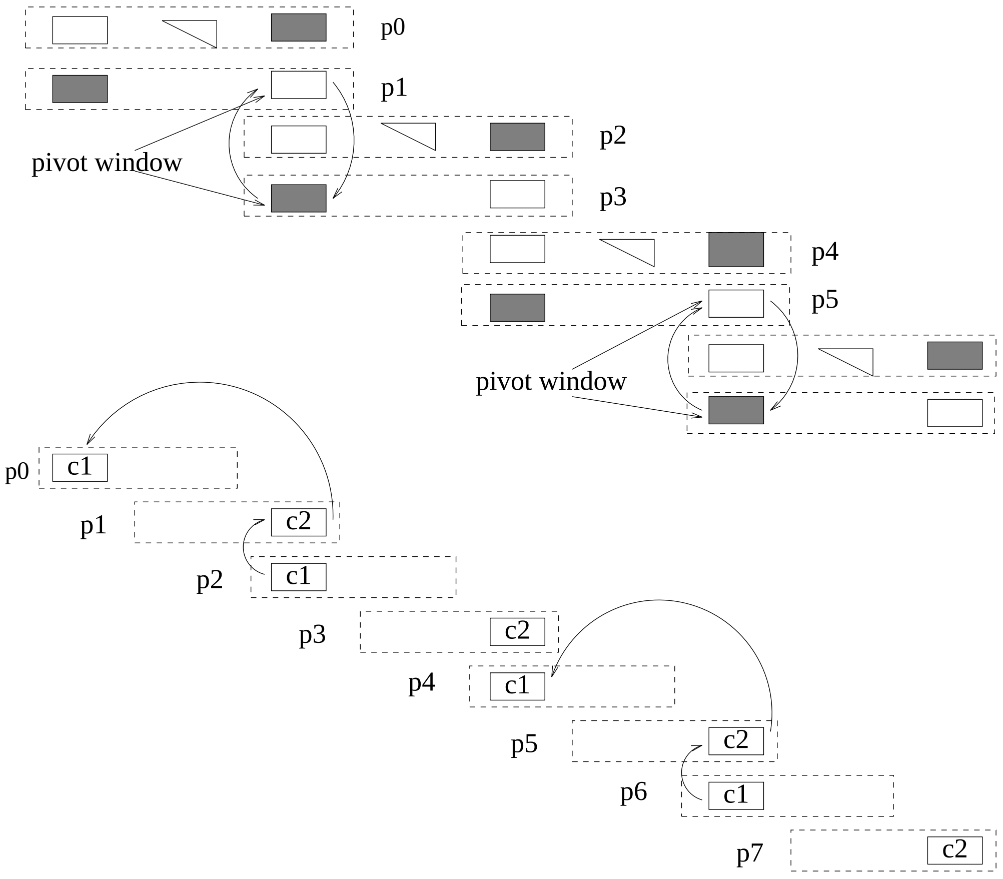

Figure 21: Level 2

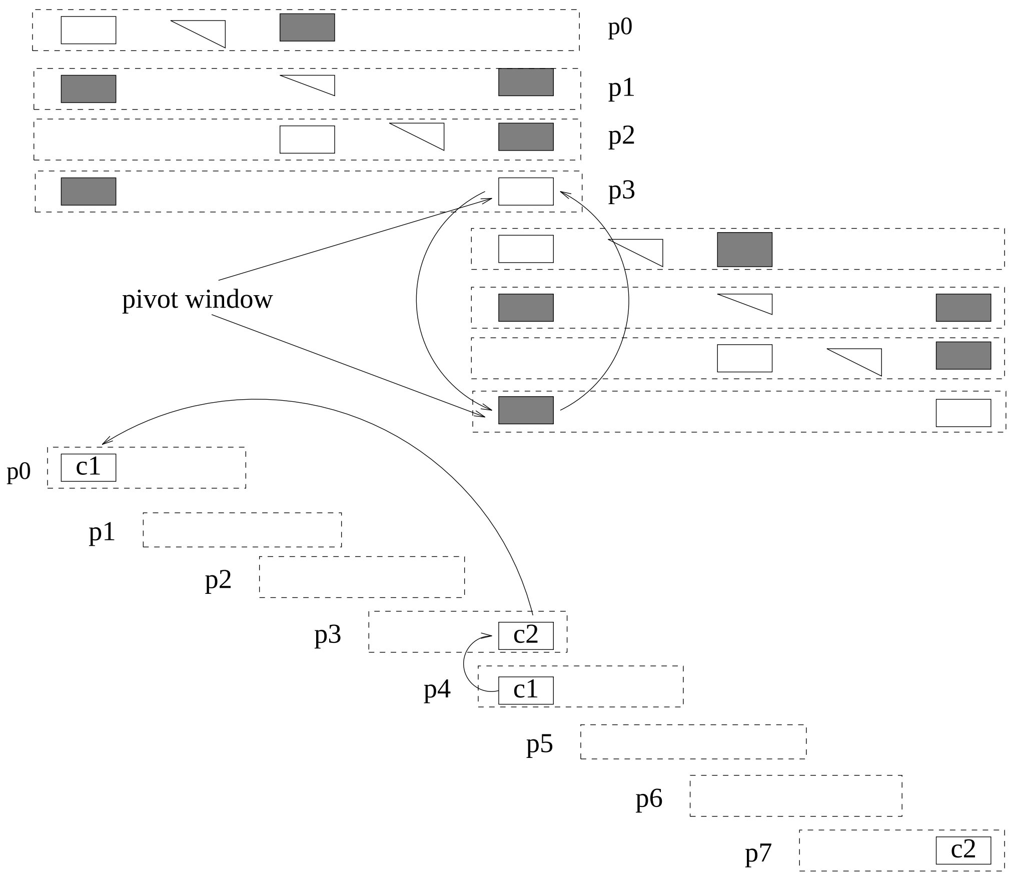

Figure 22: Level 3

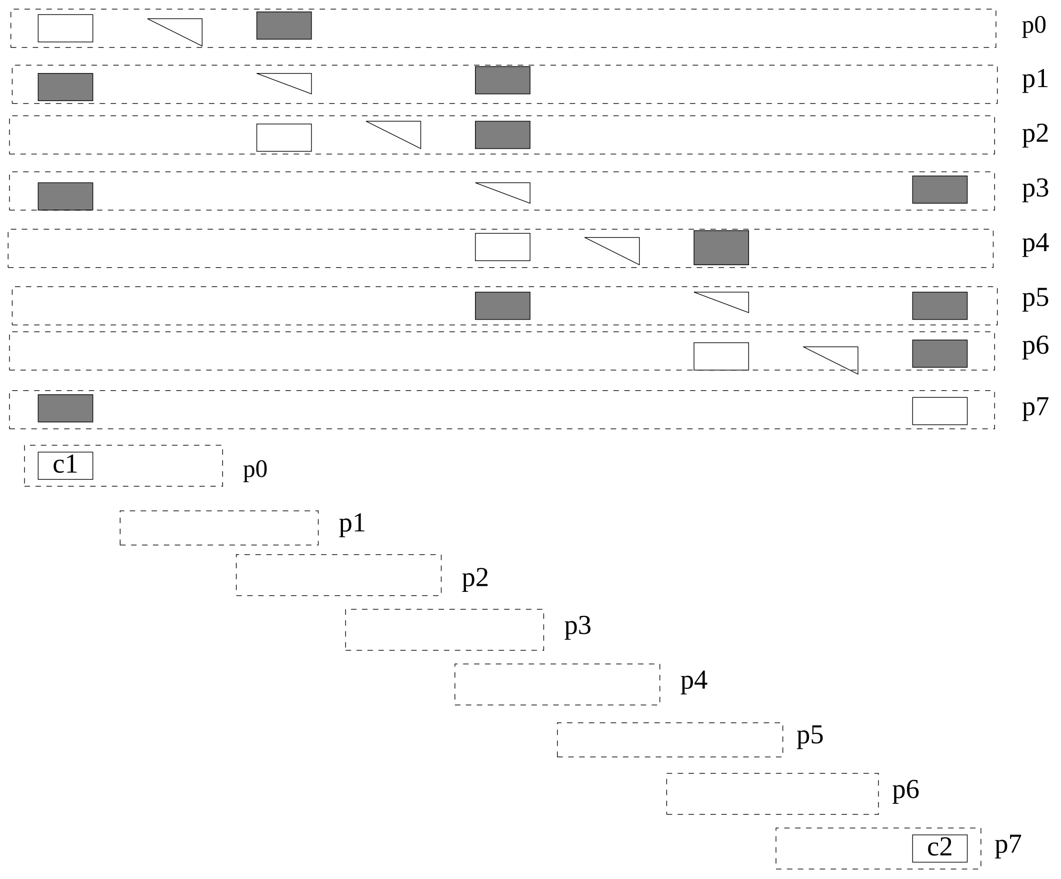

Figure 23: The Final Result of the Nested Dissection Process

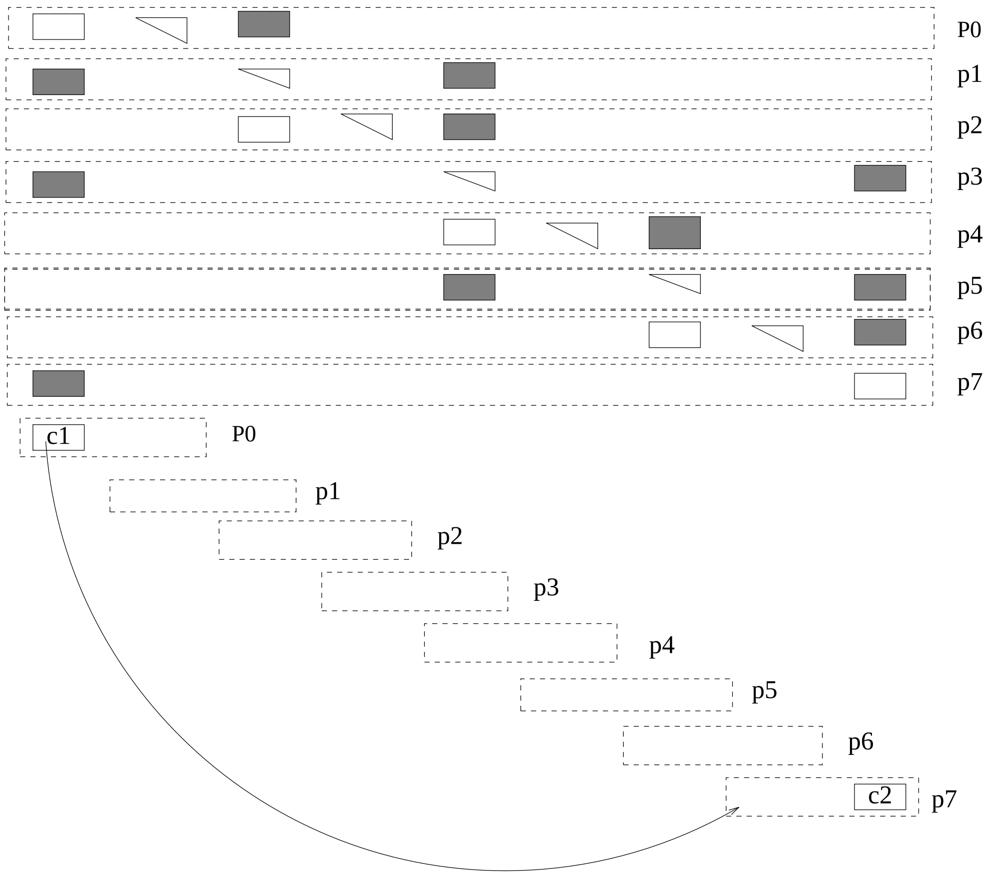

Figure 24: Send c_1 from Node p_0 to Node p_7

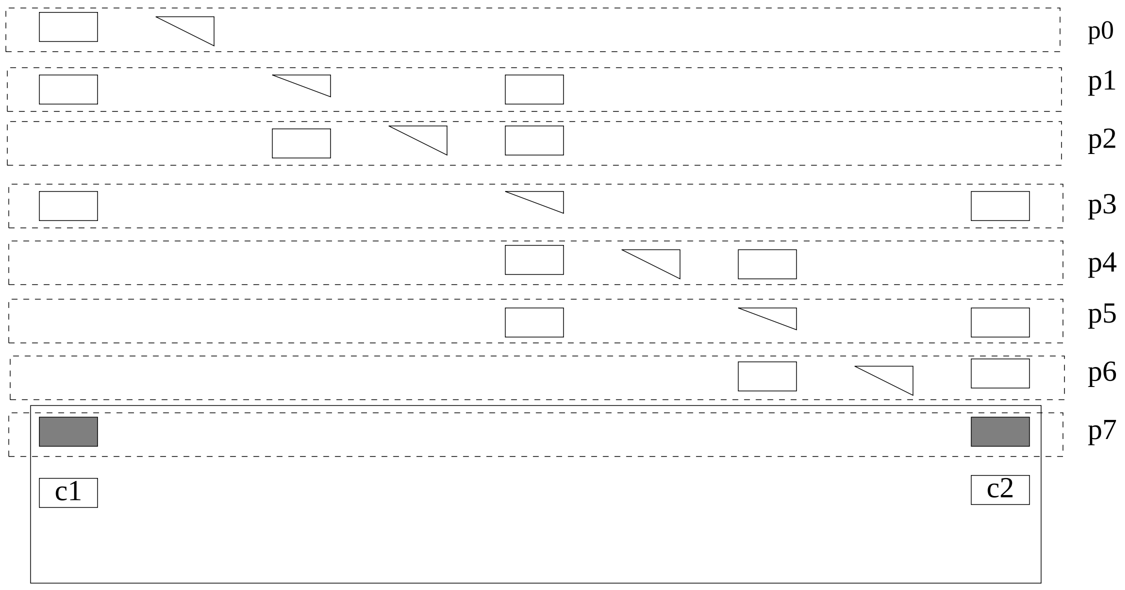

Figure 25: The Square Matrix Enclosed in the Solid-line Box

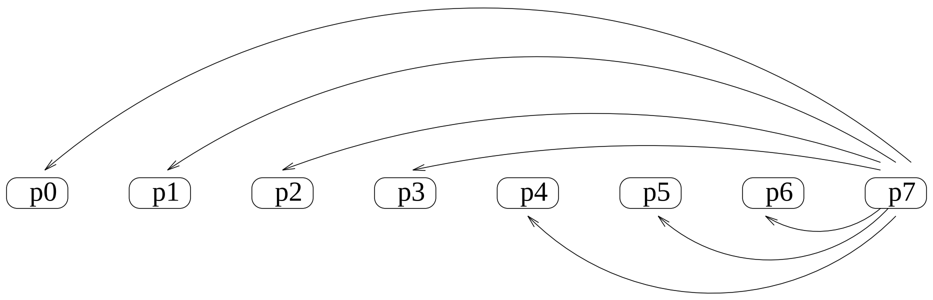

Figure 26: Broadcast of the Partial Solution from Node p_7 to All Other Nodes

```
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 16: Solving the Small System

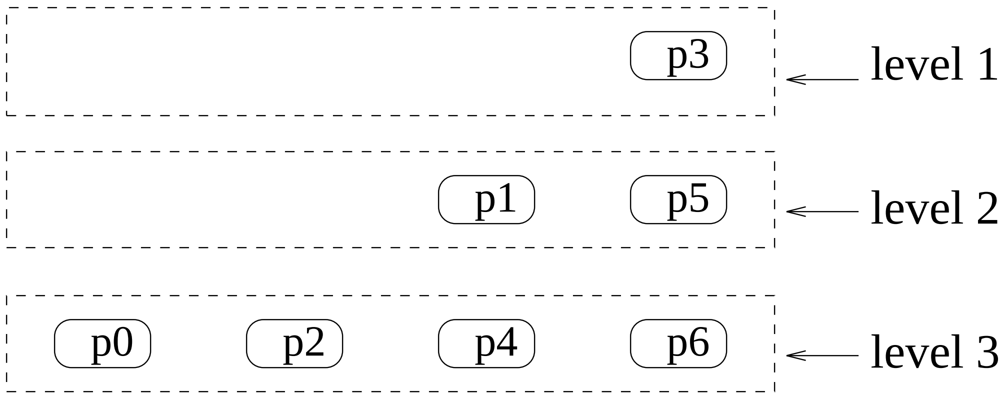

Figure 27: Busy Nodes for Each Level during Backsubstitute

```
begin
    { for each $i_p \in I_P$ do in parallel}
    for $l_p := log_2\ P, \cdots, 1, step\quad -1$ 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 17: Backsubstitution Process 1

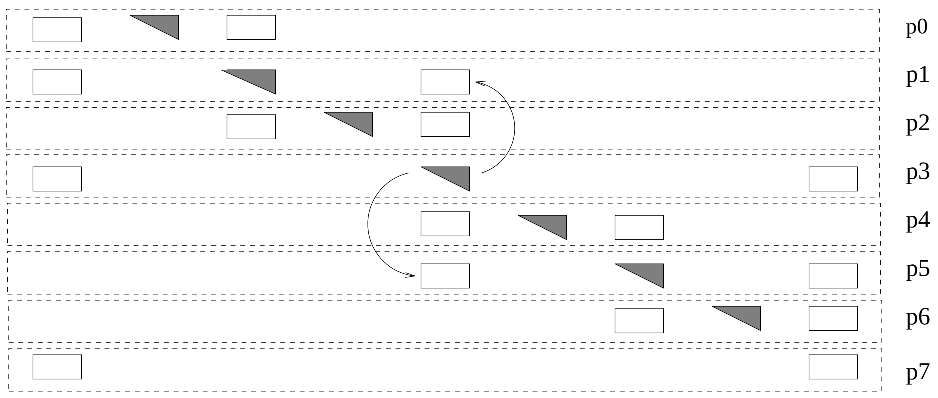

Figure 28: Backsubstitution Level 1

tution process can be interpreted graphically as follows. After broadcasting the solution from the small square system, we can obtain part of the solution corresponding to the shaded triangular area in node p_3 in Figure 28. In order to solve the part of the solution 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 28. This completes level 1 in the backsubstitution process. Similarly one can complete level 2 as indicated in Figure 29. No communication is needed for level 3.

3.6 Backsubstitution for the Condensation of Parameters

After the above backsubstitution process associated with the nested dissection, one can obtain the whole solution of the linearized system in parallel as indicated in Figure 1. 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 18.

3.7 Merging the Solutions

So far we have computed the solution to the system indicated in Figure 1. However, the solution is still scattered over all nodes, and we need to

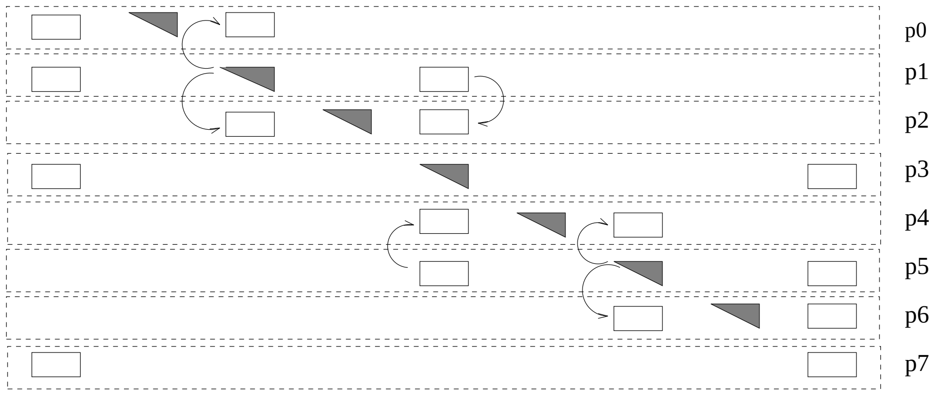

Figure 29: Backsubstitution Level 2

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

Table 18: Backsubstitution Process 2

concatenate them. This is done by a global operation as indicated in Figure 30. It consists of two steps. The first step is to let node p_0 collect all solutions

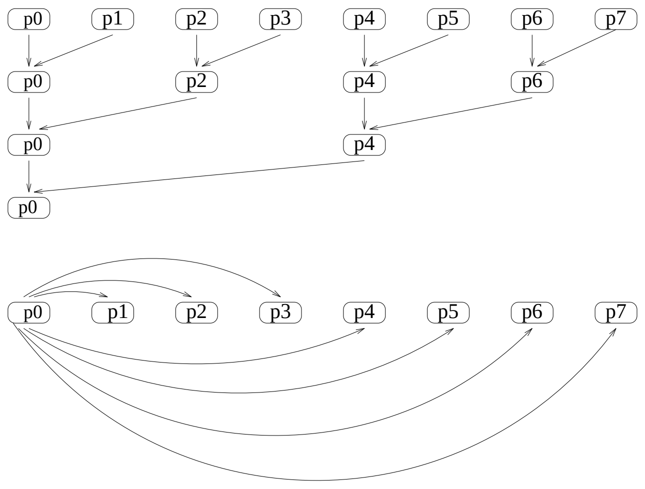

Figure 30: Concatenation of the Solutions

from the other nodes, the second step is to broadcast the full solution.

3.8 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 Section 2 for the timing purpose here. The efficiency formula (26) and the relative efficiency formula (27) are used to measure the performance.

3.8.1 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 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 19: 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 20: 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 21: 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 22: 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 23: 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 24: 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 25: 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 26: 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 27: 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 28: 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 29: 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 30: With Pivoting 1: NDIM=96, NTST=64, NCOL=4, NMX=10

3.8.2 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 (28) to calculate the the average I/O time. The average I/O times for all tables in this section are shown in Table 31. 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 31 below.

In Table	Minimum I/O time	Maximum I/O time	Average I/O time
32	1.01	1.88	1.45
33	1.89	3.65	2.77
34	3.72	6.94	5.33
35	1.87	3.83	2.85
36	3.59	7.25	5.42
37	9.53	15.02	12.27
38	1.88	4.09	2.99
39	3.89	7.01	5.45
40	7.09	13.67	10.38
41	13.81	30.23	22.02
42	3.4	7.1	5.25
43	13.47	13.47	13.47

Table 31: 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 32: 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 33: 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 34: 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 35: 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 36: 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 37: 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 38: 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 39: 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 40: 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 41: 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 42: 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 43: With Pivoting 2: NDIM=96, NTST=64, NCOL=4, NMX=10

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 44: Primitives on the Gamma and Delta machines

4 Implementation Notes

4.1 Introduction

The two sparse linear solvers described in Sections 2 and 3 are 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. 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 [2, 3] are shown in Table 44.

4.2 Node Organization

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.

4.3 I/O Strategy

Since in our case all nodes share a 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.

4.4 Interface Routines

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 our interface does separate 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.

5 How to Run AUTO94P

5.1 Overview

This section describes how to install AUTO94P on *delilah.ccsf.caltech.edu* and how to run it on *delta1.ccsf.caltech.edu*. The first machine is a SunOS UNIX workstation and the second is the Intel Delta. We also assume the parallel Fortran compiler is the cross compiler for i860 microprocessor from the Portland Group (Release 3.0). The instructions below must be accurately followed.

5.2 Installation

AUTO94P is packed as a tar file *auto94p.tar.Z*. After obtaining *auto94p.tar.Z*, put it in the home directory.

1. Uncompress *auto94p.tar.Z* by typing
   uncompress auto94p.tar

2. Unbound *auto94p.tar* by typing
   tar xvfo auto.tar
   This will create a directory *auto94p* containing the AUTO files.

3. Enter the directory *$HOME/auto94p*, and type
   make
   to compile the entire package.

4. Remove unnecessary files by typing
   make clean

The above four steps will complete the installation of AUTO94P.

5.3 How to Run AUTO94P

We use the demo *exp.f* from AUTO94 to illustrate how to run AUTO94P on the Delta.

1. **Compile** *exp.f* on *delilah.ccsf.caltech.edu*.

2. **Transfer** *exp.exe, r.exp.1, r.exp.2* from *delilah.ccsf.caltech.edu* to *delta1.ccsf.caltech.edu*.

3. **Load** *exp.exe* on *delta1.ccsf.caltech.edu.*

A detailed description follows.

Compile: The compilation should be done on *delilah.ccsf.caltech.edu.* First copy *exp.f, r.exp.1, r.exp.2* from *$HOME/auto94p/demos/exp* to *$HOME/auto94p/work* by typing
cp $HOME/auto94p/demos/exp/*exp* $HOME/auto94p/work
Then enter the work directory by typing
cd $HOME/auto94p/work
Then modify the *Makefile* under directory *$HOME/auto94p/work.*
Specifically, replace the line
SRC = name
by
SRC = exp
Finally compile *exp.f* by typing
make
This will generate *exp.exe*.

Transfer: In directory *$HOME/auto94p/work* on *delilah.ccsf.caltech.edu*, type
ftp delta1.ccsf.caltech.edu
then enter the *login name* and the *password*.
Use binary mode for the transfer by typing
bin
Finally do the following
put exp.exe
mput r.*
put fort.003
Note that **fort.003** is an empty file that must be created.

Load: Login on Delta by typing
telnet delta1.ccsf.caltech.edu
After entering the *login name* and the *password*, type
cp r.exp.1 fort.002
Now the first run of *exp.f* is ready.
To load *exp.exe* to a 2x2 mesh (4 nodes), type
mexec "-t(2,2)" -f exp.exe

Save *fort.007, fort.008, fort.009* as *p.exp, q.exp, d.exp* respectively, by typing

```
cp fort.007 p.exe
cp fort.008 q.exe
cp fort.009 d.exe
```

This completes the first run. To do the second run, type

```
cp r.exp.2 fort.002
cp q.exp fort.003
```

To load *exp.exe* to a 2x2 mesh, type

```
mexec "-t(2,2)" -f exp.exe
```

Append *fort.007, fort.008, fort.009* to *p.exp, q.exp, d.exp* respectively

```
cat fort.007 >> p.exp
cat fort.008 >> q.exp
cat fort.009 >> d.exp
```

This completes the second run.

References

[1] 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.

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

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

[4] Eric F. Van de Velde. *Concurrent Scientific Computing.* Springer-Verlag, 1994.

[5] 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.

[6] 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.

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

[8] E. J. Doedel, X. J. Wang, and T. F. Fairgrieve. "AUTO94: Software for Continuation and Bifurcation Problems in Ordinary Differential Equations". Technical report, CRPC-95-1, Center for Research on Parallel Computing, California Institute of Technology, Pasadena, CA 91125, 1995.

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

[10] 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.

[11] 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.

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