1986

Partitioning PDE Computations: Methods and Performance Evaluation

Catherine E. Houstis

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
86-614

PARTITIONING PDE COMPUTATIONS: METHODS AND PERFORMANCE EVALUATION

Catherine E. Houstis*
Department of Electrical Engineering
Elias N. Houstis**
John R. Rice**
Department of Computer Science
Purdue University
CSD-TR 614
June 2, 1986

ABSTRACT

We consider modeling, predicting and evaluating the performance of methods for solving PDEs in parallel architectures. We have developed a method for coarse grain partitioning of computations for parallel architectures and we apply it to three PDE applications: (a) Cholesky factorization, (b) spline collocation, and (c) an application complete from processing text input to plotting the PDE solution. Our partitioning method is oriented to minimizing interprocessor communication and we review some "uniform" architectures and models of their communication. We apply this method to the three applications implemented on the FLEX/32 multicomputer. We review the architecture on the FLEX/32 and the results of applying the partitioning method to computation running on the FLEX/32. We observe that the FLEX/32 does not have any communication bottleneck and probably will not suffer substantial performance degradation if the processor speeds are increased by a factor of 10. Our partitioning method works reasonably well even here where communication costs are negligible. The coarse grain structure of two of these applications is not highly parallel and we observe speedups of about k/2 for k processors. The other application is highly parallel and we observe optimal speedups for any number of processors as the problem size increases.

---

*This research supported in part by NSF grant DMC-8508684.
**This research supported in part by AFOSR grant 84-0385.
This paper will appear in the January 1987 issue of Parallel Computing.

1. MODELING THE PERFORMANCE OF ALGORITHM / ARCHITECTURE PAIRS

We consider a Partial Differential Equation (PDE) application A to be a computation with four properties: processing requirements, memory requirements, communication requirements and precedence (or synchronization) between the subcomputations. Visualize the computation broken into *computational modules* which are nodes of a precedence graph for the computations. Note the processing and memory requirements at each node of the graph. Note the communication requirements along each link or edge of the graph. This annotated graph is called G(A). Observe that links representing communication may need to be added to G(A) even though they are redundant as far as precedence is concerned.

We consider a machine to have three components: processing elements, memory elements and communication paths (an interconnection network). Similarly, the machine can be represented by an annotated graph G(M). In this paper, we consider machines that have a set of identical processors (which may have local memories), a set of identical common memories, and a queueing delay function which represents the communication performance characteristics of the machine. Intuitively, one may consider these as machines with a rather uniform architecture, one that scales simply with the number of processors and memories.

Assume that the graph G(A) is given and that an appropriate choice of problem size and granularity has been used in partitioning A to create G(A). Assume also that G(M) is given. In general, we want to map G(A) onto G(M) so the computational modules (nodes) of G(A) are associated with processors in G(M) and communication links (edges) of G(A) are associated with data paths in G(M). We assume here that the parallelism of G(A) is no larger than that of G(M). Normally the number of nodes in G(A) is much larger than the number of processors in G(M) and the computation is *scheduled* by assigning the computational modules to processors. This produces a new graph $G'(A)$. The three graphs, G(A), $G'(A)$ and G(M) along with their associated parameters (e.g. speed of processors, amount of arithmetic at a node) are a model of the PDE computation A to be carried out on machine M. One key question is how well this model predicts the performance of typical algorithm/architecture pairs. A second key question is how to map G(A) into G(M) so as to obtain $G'(A)$.

In general, the mapping problem has three somewhat independent steps:

1. Reduce the parallelism of the application to that of the machine.
2. Schedule the computational modules so that the application runs efficiently.
3. Given steps 1 and 2 are done, imbed the application into the machine.

The mapping problem is known to be NP complete [JENN77] and to find the optimal mapping for any of these three steps is also NP complete. We propose to use fast, heuristic algorithms here which are intended to produce good mappings at low cost.

Due to the lack of space, we discuss the parallelism reduction in depth elsewhere [HOUS86], and assume here that this step is not needed. Given that steps 1 and 2 are done, step 3 is fairly easy for the class of machines considered here. Thus, this paper concentrates on the problem of scheduling the application so that it runs efficiently and on evaluating the performance of these schedules.

Our overall plan for making performance evaluations is as follows. We first devise fast, heuristic algorithms which carry out steps 1, 2 and 3 above. Note that the order of steps 1 and 2 is not necessarily fixed, we assume in the discussions that step 1 precedes step 2 so as to simplify the discussion but this is not an essential ingredient to our approach. Our heuristics use communication delay models for "uniform" architectures in order to simplify the complexity of the mapping problem. Once we have the application mapped onto the machine and scheduled to provide good efficiency, we then run the program on an actual machine (we use a FLEX/32 multiprocessor) or use a simulation package, (for example, SIMON[FUJI85]) to "execute" a "compressed" revision of the program and to estimate its performance.

The performance of a distributed system depends very much on how well the architecture and the algorithms are matched [KLEIN85]. A methodology for predicting multiprocessor performance from the systems point of view and the RP3 architecture is presented in [NORT85]. The problem of mapping has been studied in [BOKH81] and [BERM84,85] for the finite element machine and CHIP architectures, respectively. In [WILL83] various objective criteria necessary for assigning processes to processors are examined and tested in a distributed environment. A different approach for modeling communications complexity of parallel algorithms is presented in [GANN84]. Our approach to mapping PDE computations onto parallel machines (i.e. step 2 of the mapping process) is described in [HOUS83] and our approach to compressing the parallelism is described in [HOUS86].

In this paper we give models of three PDE applications (Section 2) and discuss how to model communications for various machine architectures (Section 3). We then describe the FLEX/32 architecture and its performance (Section 4) and apply our mapping algorithm to the three PDE applications (Section 5). The observed performance for these applications is discussed (Section 6) and compared with predictions from the model. Some conclusions are stated in Section 7.

2. APPLICATION MODELING WITH THREE EXAMPLES

For the modeling of an application, we assume an initial partitioning of the computation into a number of communicating computational modules. If the communication paths among the modules are not known a priori (i.e. real time applications) then the partition is modeled by a stochastic AND-EOR directed graph SG(A). In [HOUS84] a stochastic analysis is applied to reduce SG(A) into a deterministic graph G(A) which

comprises the input to the mapping model.

If the communication paths among modules are known a priori then we use complexity analysis and/or simulation to obtain values for the parameters of the computation graph G(A). In the case of simulation, we use the data flow language SIMON [FUJI85] and concurrent C and FORTRAN languages [FLEX85] to specify the computation modules and their communication and synchronization requirements. In order to execute SIMON programs we use our non-shared memory multiprocessor simulator [FUJI85]. Concurrent C or FORTRAN programs are run on FLEX/32 multiprocessor system. Next, we consider three scientific computations to be used for the evaluation of our methodology.

2.1 Application I: Cholesky decomposition of symmetric matrices

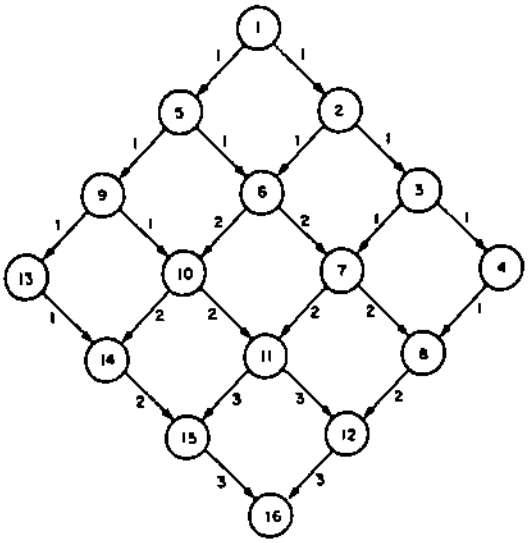
We consider the parallelization of Cholesky decomposition of symmetric matrices. Figure 1 presents the computation of each module in the example of a parallel Cholesky decomposition algorithm [O'LEA85]. Figure 2 depicts the corresponding graph G(A) and the values of the various workload parameters (node processing time and blocking time, communication traffic among nodes) obtained by setting SIMON's switching delay to zero. The *blocking time* or algorithm *synchronization delay* of a module is the time that the module must wait for its inputs before its computation can start.

```
K:=0
WHILE (TRUE) BEGIN
    K:=K+1;
    IF (K = i and K = j) then
        BEGIN
        X:=sqrt(X):
        IF(j ≠ N) PUT (OUT_EAST,X);
        IF(i ≠ N) PUT (OUT_SOUTH,X);
        BREAK;
        END
    ELSE IF (K=j) THEN
        BEGIN
        GET (IN-SOUTH,Y);
        X=X/Y
        IF (j ≠ N) PUT (OUT_EAST,X);
        IF (i ≠ N) PUT (OUT_SOUTH,Y);
        BREAK;
        END
    ELSE IF (K=j) THEN
        BEGIN
        GET (IN_EAST,Y);
        IF (j ≠ N) PUT (OUT_EAST,Y);
        IF (i ≠ N) PUT (OUT_SOUTH,X);
        BREAK;
```

```
ELSE BEGIN
    GET (IN_SOUTH,Y);
    GET (IN_EAST,Z);
    X=X*Y*Z;
    IF (i ≠ N) PUT (OUT_SOUTH,Y);
    IF (j ≠ N) PUT (OUT_EAST,Z);
    END
END WHILE
```

Figure 1. Computation of node (i,j) in SIMON data flow language for a parallel Cholesky decomposition algorithm.

A matrix is partitioned into blocks so that the maximum degree of parallelism is equal to the number of processors. If smaller blocks are used, then the parallelism reduction would be applied but we do not consider this case here. This application provides an example where the parallelism is mostly less than maximum and thus where the speedup should be considerably less than the number of processors. See [O'LEA86] for further discussions. That paper shows that optimal speedup is achievable with a speedup of N/9 for N processors.



Module processing times
$\underline{m}$ = (24, 30, 30, 25, 27, 31, 37, 32, 27, 34, 38, 39, 21, 28, 35, 45)

Module blocking times
$\underline{b}$ = (0, 4, 14, 24, 15, 27, 37, 48, 33, 45, 61, 66, 51, 64, 76, 88)

Figure 2. Precedence graph G(A) of the parallel Cholesky decomposition algorithm for a 4 by 4 block decomposition of a symmetric matrix. The assumed numbering of modules is specified in each node. The communication traffic is indicated as a weight on the links of the graph.

2.2 Application II: Spline collocation methods for elliptic PDEs

We consider the numerical solution of an elliptic PDE problem defined by the second order elliptic operator

$$Lu \equiv \alpha u_{xx} + 2\beta u_{xy} + \gamma u_{yy} + \delta u_x + \epsilon u_y + \varsigma u = f,$$

on $\Omega \subset \mathbb{R}^2$ subject to boundary conditions

$$Bu \equiv 0 \text{ on the boundary of } \Omega.$$

We use the line collocation method based on cubic splines studied in [HOUS84], [VANA85] to discretize the above PDE problem. For simplicity, we consider the Helmholtz equation with constant coefficients and an iterative method corresponding to SOR for solving the underlying linear equations. This discretization is defined by the block equations

$$\mathbf{A1x} = (1-\omega)\,\mathbf{A1x} - \omega\mathbf{A2y} + \omega\mathbf{F}, \tag{2.1a}$$

$$\mathbf{Q2y} = (1-\omega)\mathbf{Q_{2}y} + \omega\mathbf{Q1x} - \omega\mathbf{G}. \tag{2.1b}$$

where

A1 = diag (T,T, ..., T) with T=trid (1,−2,1),

$Q2 =$ diag (S,S, ..., S) with S=trid (1,4,1),

$A2 = P \cdot A1$, $Q1 = P \cdot Q2$,

P = permutation matrix of the ordering of unknowns,

x = solution vector in horizontal mesh lines.

y = solution vector in vertical mesh lines.

ω = overrelaxation parameter

F,G = constant matrices from the right side f.

The discretization (2.1) can be considered as the application of a set of equations to a data set. For example, the first block of equations (2.1a) is applied to data associated with the horizontal mesh lines (see Fig. 3a) and the (2.1b) equations are applied to data related to the vertical mesh lines (see Fig. 3b). One partition technique is to divide the data sets (mesh lines) into parts, instead of the equations, and assume each part is assigned to a single processor. Figure 4 depicts the parallel spline collocation algorithm based on the above data set decomposition for a two processor system. The generalization is straight forward. In Section 6 we present the performance of this parallel algorithm for a shared memory architecture.

2.3. Application III. Large scale PDE computation.

This PDE application is complete from the description of a problem in a very high level language through the graphical display of the computed solution. Figure 5 shows a schematic diagram with the steps of the computation, it has degree of parallelism 6. Not shown in Figure 5 are vertical data communication paths between corresponding nodes on the fan-in/fan-out of the tree structure in the center. The PDE problem solved is a general one on a non-rectangular domain. A multifront method is used based on a nested dissection partition of the domain. Gauss elimination is used to eliminate unknowns in the interior of each domain. The numbers shown at each node are the "computation units" for a particular instance of this problem corresponding to using a 20 by 20 mesh, a finite element method with cubic basis functions, and a 40 by 40 plotting grid. The domain boundary has 3 pieces and, at the fourth step, all but one processor are working on the interior of the domain. A computational unit here is about 1000 arithmetic operations plus associated memory and control operations.

Figure 6 shows the same graph with the additional communication paths. These paths transfer LU factorizations of subblocks of the linear systems from the initial solve phase to the back substitution phrase. The numbers shown are data transfers required in units of 1000 words.

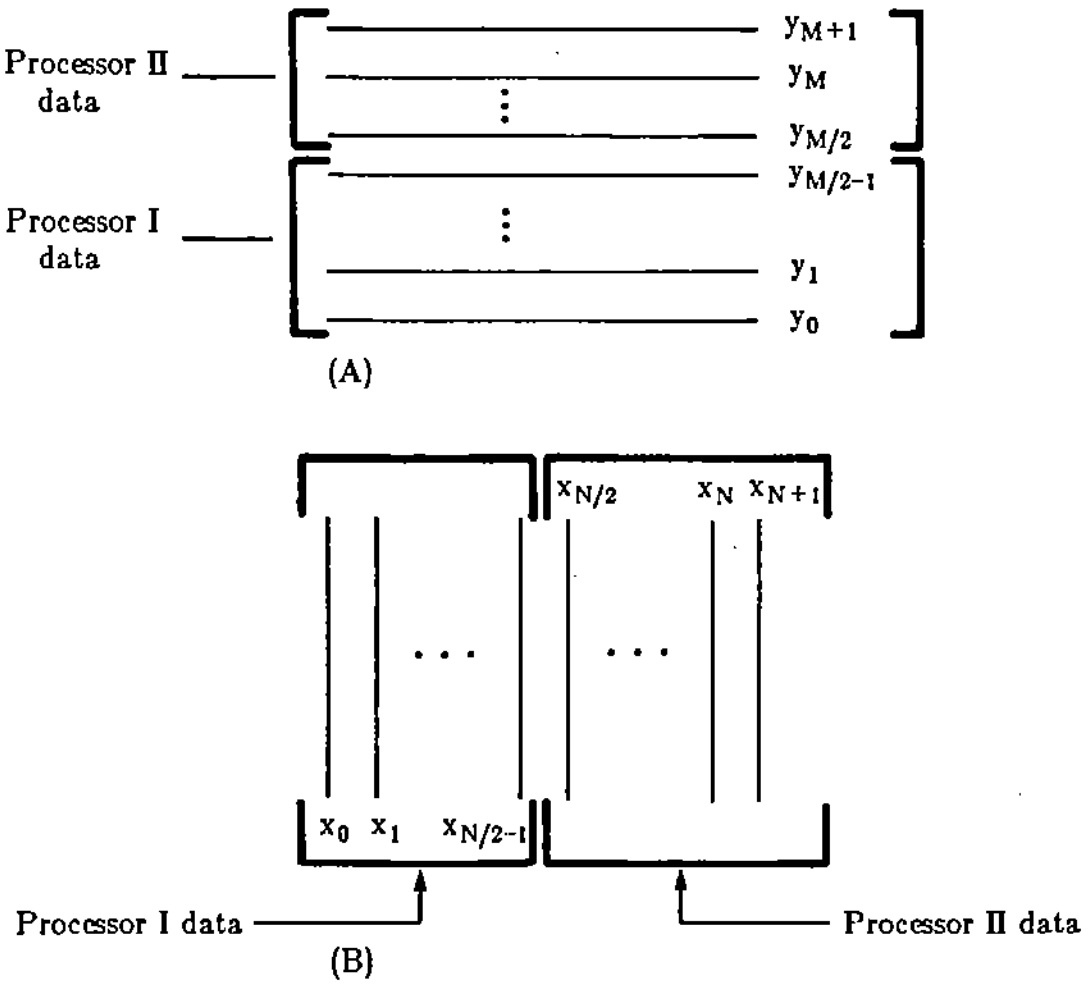


Figure 3. Partitions of the mesh line data of (2.1) into (A) horizontal mesh lines and (B) vertical mesh lines. The assignment of the lines is shown for the case of two processors.

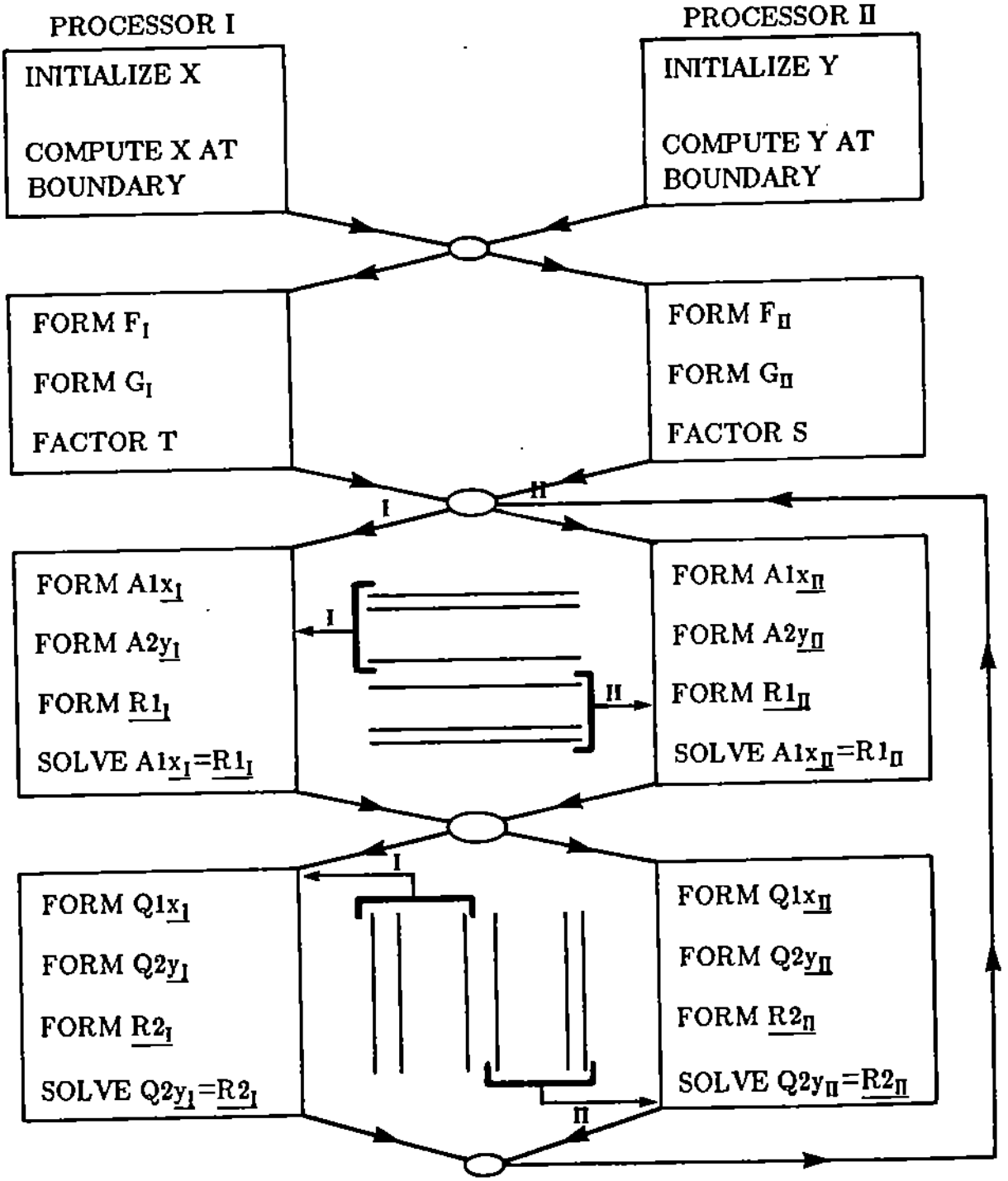
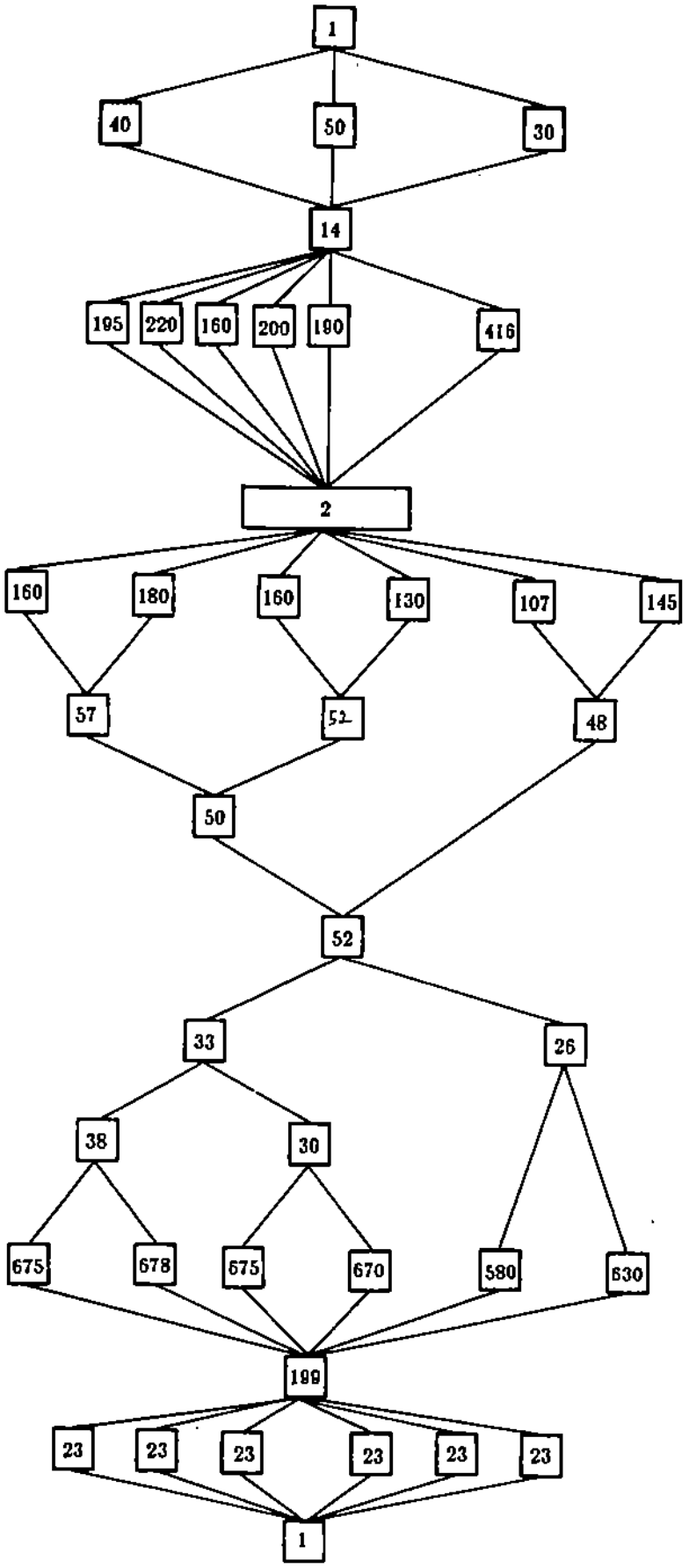


Figure 4. Schematic of the parallel spline collocation algorithm and iteration (2.1). The decomposition for two processors is shown based on alternately using on data sets the horizontal and vertical mesh lines.



Read Problem Description

Process 3 Boundary Pieces

Merge Boundary Data Structures
Identify All Elements

Process Interior/Boundary
Elements

Create 6 Frontal Areas

Interior Assembly + Elimination

Merger and Elimination
in Neighboring Areas

Merger and Elimination
in Neighboring Areas

Merger and Elimination
in Neighboring Areas

Backsolve for Neighboring
Unknowns

Backsolve for Neighboring
Unknowns

Backsolve for Unknowns
Tabulate Answer for Plot
Tabulate Flow Field

Create Plot Data Structure

Compute Contours, Colors
for 6 Views

Figure 5. Annotated graph G(A) for Application III for 6 processors. The numbers at the nodes are units of computation to be done.

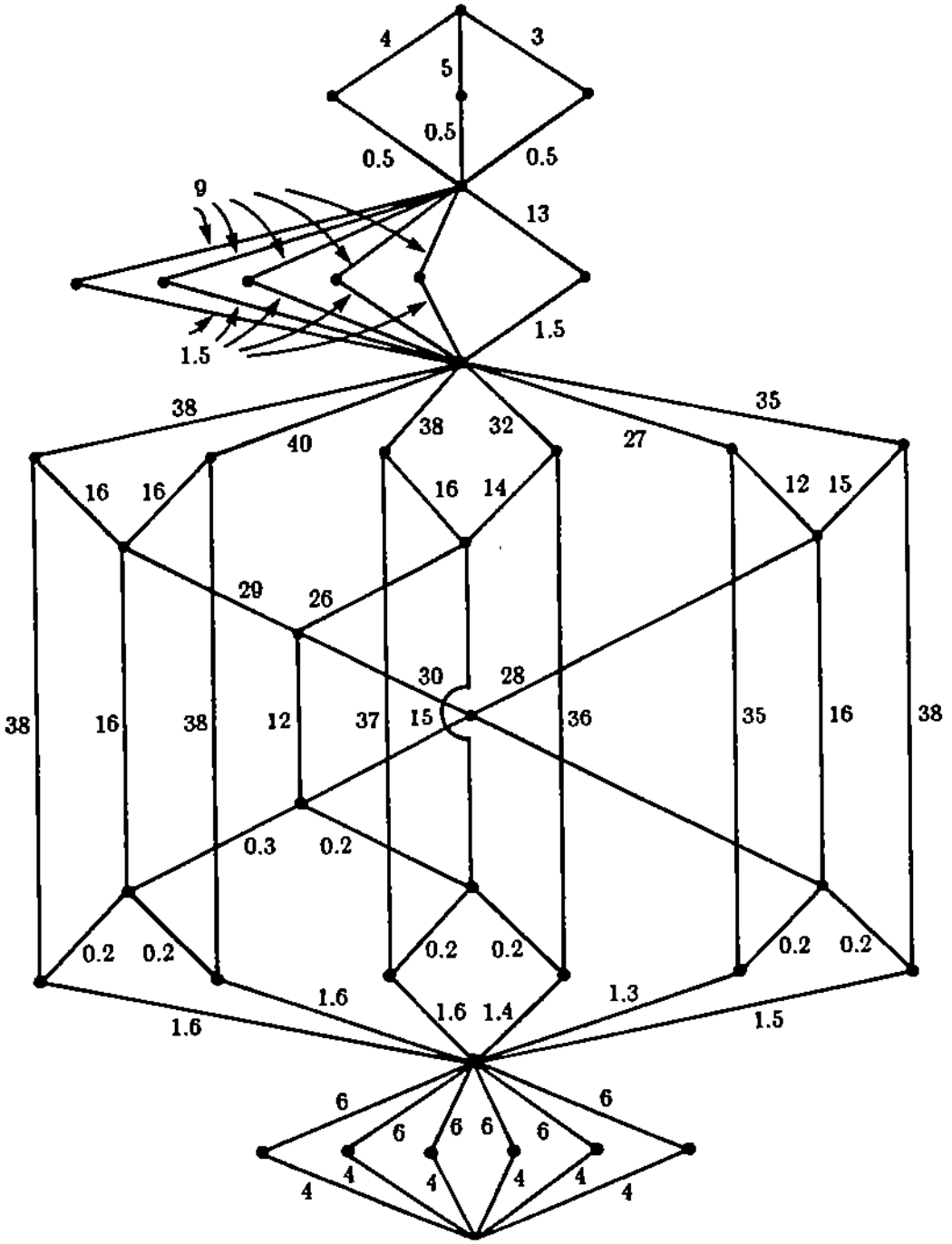


Figure 6. The graph of Figure 5 showing additional communication edges and the number of units to be communicated along each edge.

3. COMMUNICATION MODELING IN MULTIPROCESSOR ARCHITECTURES

Our methodology for reducing the initial interconnection graph G(A) is based on minimizing its communication cost. Thus, it is crucial to be able to predict the communication *cost* of G(A) running on a specific multiprocessor system architecture. We consider architectures for which there is a realistic simple model of the communication cost at the message passing level. Such performance models of multiprocessor systems are given in [MAR83], [KRU83]. The performance measure is a delay function which provides the expected queueing delay a message suffers in traversing the interconnection network of the system. Queueing delay is computed as a function of the system's communication interconnection network utilization.

Let

c_{ij} = total traffic (# of messages) between modules m_i and m_j,

k,ℓ = processors assigned to computational modules, m_i, m_j

and

$d_{k,\ell}$ = interprocessor distance in the interconnection network of the considered architecture,

then the *interprocessor communication cost* or queueing delay is given by

$$D = D(u)$$

where $u = u(c_{ij}, d_{k,\ell}$, interconnection network characterization) denotes the utilization. When two modules are assigned to different processors their utilization of the interconnection network is computed as follows:

$$u = \frac{c_{i,j} * d_{k,\ell}}{C * T}$$

where C = capacity of interconnection network and T = time frame during which each parallel processor is running.

The delay obtained is only approximate because of the various simplifying assumptions involved in the analytical modeling of the performance characteristics (queueing, utilization) of the multiprocessor system. In all of the models we consider, two assumptions are made: (a) every processor accesses the interconnection network at the same rate, and (b) every processor accesses the various common memory modules with the same probability. These assumptions lead to a queueing model which is mathematically tractable. Intuitively, these assumptions imply that the communication requirements of G(A) are nearly uniform. If G(A) is one of many applications running in the same system, these assumptions may be realistic even if the algorithm has somewhat

non-uniform communication patterns. A precise investigation on this question is under way.

We consider the performance analysis of five different multiprocessor parallel architectures: (1) Single bus, common memory, (2) Single bus, distributed common memory modules, (3) Multiple bus, distributed common memory modules, (4) Omega interconnection, distributed memory modules, (5) Omega interconnection processor to processor multiprocessor system.

3.1 Single bus common memory architecture

In this architecture, (Figure 7(i)), k processors (P_i) are connected to an external common memory (CM) via a global bus (GB). Each P_i has a local memory (LM_i) connected to its own local bus (LB_i). The system has been analyzed in [MAR83] as a "machine repairman" problem. For given bus utilization level, say u_1, the average queueing delay per information transfer unit is given by $D_1(u_1) = (k-u_1/\rho_1)/u_1$. The parameter ρ_1 characterizes the communication of the architecture and it is found by solving the nonlinear algebraic equation.

$$u_1 = \frac{P_k(\rho_1)-1}{P_k(\rho_1)} \quad \text{where} \quad P_k(\rho_1) = \sum_{j=0}^{k} \rho_1^j k!/(k-j)!.$$

The workload characterization of the application is given by $\rho_p = \rho_1/2$ and represents the ratio λ_p/μ where λ_p is each P_i's access rate to the bus which in turn depends on the communication pattern of G(A) and $1/\mu$ is the average memory transfer requirement which depends on the average message length.

3.2 Single bus distributed memory architecture

This architecture (Figure 7(ii)), is obtained from the previous one by distributing the common memory to each processor. The local memory of each P_i is logically divided in private memory PM_i and common memory CM_i and accessed by a double port.

This system has been analyzed in [MAR83]. The average queueing delay per information transfer unit as a function of the bus utilization is given by

$$D_2(u_2) = \frac{k-u_2/\rho_2}{u_2} \quad \text{where } \rho_2 = \frac{\rho_p}{\rho_p+1}$$

$$u_2 = \frac{P_k(\rho_2)-1}{P_k(\rho_2)} \quad \text{where } P_k(\rho_2) = \sum_{j=0}^{k} \rho_2^j \frac{k!}{(k-j)!}$$

3.3 Multiple bus distributed common memory modules architecture

A multiple bus distributed common memory modules architecture is shown in Figure 7(iii). The P_i's have their own LM_i's and communicate via common memory modules. Multiple global busses are used by the P_i's to access CM_i's. Such a system is referred to as a k×m×b system where k is the number of P_i's, m is the number of CM_i's and b is the number of global busses used. We assume that $k \geq m > b$.

This system has been analyzed in [MAR82] using several approximate models based on a Markov chain approach. The model we are adopting here provides performance characteristics which are a lower bound to the characteristics of the actual system.

The average queueing delay per information transfer unit is given by

$$D_3(u_3) = \frac{k-u_3/\rho_3}{u_3} \quad \text{where } \rho_3 = \rho_p$$

$$u_3 = \frac{P_k(\rho_3)-1}{P_k(\rho_3)} \text{ where } P_k(\rho_3) = \sum_{i=1}^{k} \frac{\rho_3^{k-j}\frac{k!}{(i-1)!}\prod_{\ell=1}^{k-i}\beta_\ell^{-1}}{1+\sum_{j=0}^{k-1}(\rho_3^{k-j}\frac{k!}{j!}\prod_{\ell=1}^{k-j}\beta_\ell^{-1})}$$

$$\text{and } \beta_\ell = \frac{\sum_{j=1}^{b-1} j p_j(\ell) + b\sum_{j=0}^{\ell-b}[p_b(j+b)p_{m-b}(\ell-2b-j+m)]}{\sum_{j=1}^{b-1} p_j(\ell) + \sum_{j=0}^{\ell-b}[p_b(j+b)p_{m-b}(\ell-2b-j+m)]}, \ell \geq 0$$

where

$$p_j(\ell) = p_j(\ell-j) + p_{j-1}(\ell-j) + \ldots + p_1(1-j) + p_0(1-j)$$

with initial conditions

$$p_j(\ell) = 0 \quad \ell < j$$
$$p_0(\ell) = 0 \quad \ell > 0$$
$$p_j(\ell) = 1 \quad j \geq 0$$

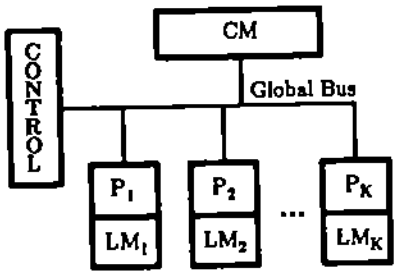
3.4 Omega interconnection distributed memory modules

In Figure 7(iv), an example of an Omega interconnected architecture is shown where the Omega network is drawn for k = 8 processors in the system, and 2x2 switches.

This system has been analyzed by [KRU83], as the interconnection network of the ultracomputer [KRU83]. The same analysis applies to a general class of multistage

Architecture | **Delay function**

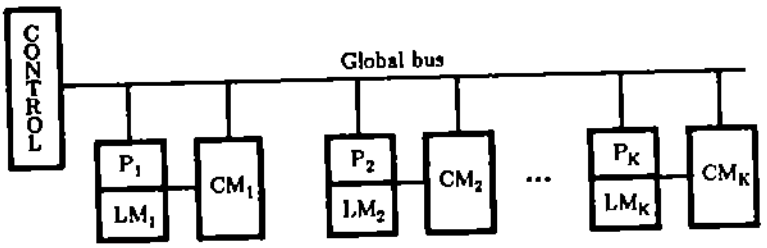
(i) Single bus common memory



$\rho_p = \rho/2$

$D = (k-u/\rho)/u$

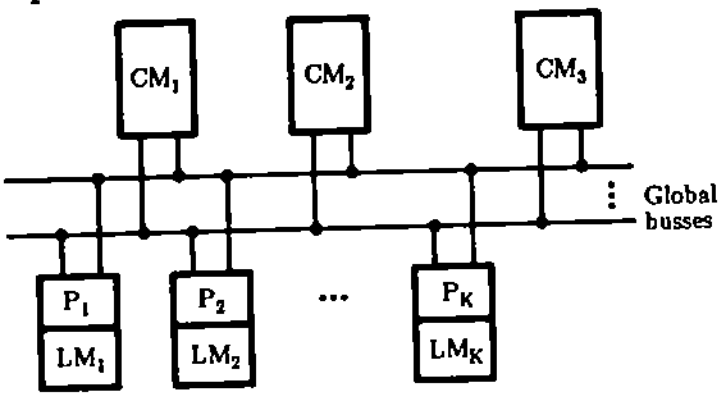
(ii) Single bus distributed common memory



$\rho_p = \frac{\rho}{1-\rho}$

$D = (k-u/\rho)/u$

(iii) Multiple bus distributed common memory



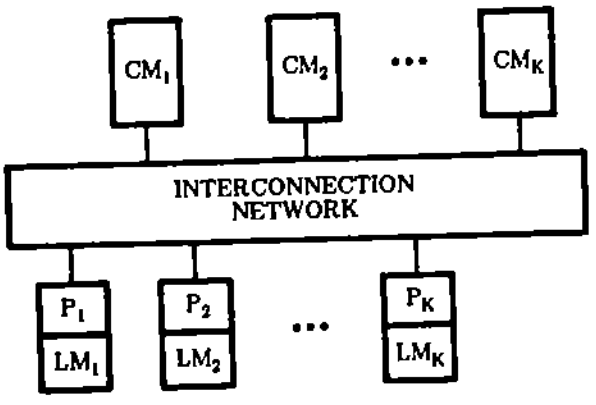
$\rho_p = \rho$

$D = (k-u/\rho)/u$

(iv) Omega Interconnection distributed common memory

$\rho_p = \rho$

$D = \log_n k(t_r + t_c \frac{m^2(1-1/n)\rho}{2(1-n\rho)}) + (m-1)t_r$



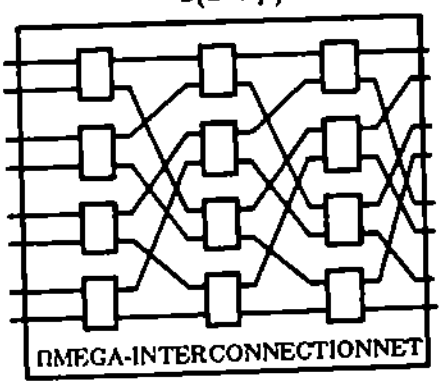


Figure 7. The delay function and workload parameter (ρ_p) as a function of utilization (u) and parameter (ρ) for four shared memory architectures with k processors. In case (iii) m = # of common memories, b = # of common busses and $k \geq m > b$. In case (iv) n×n is the size of the switch, t_r = transit time of a switch, t_c = cycle time of a switch, m = # of packets in a msg; ρ = average # of msgs entered by each P_i/cycle time.

interconnection networks called Banyan networks [KRU83]. Some of these are the Omega, Delta or indirect cube interconnection networks. In our case, we have considered n×n switches where the capacity of the buffers is infinite. Then the average delay per message in traversing the interconnection network is simply its delay per stage times the number of stages ($\log_n k$) in the interconnection.

$$D_4(\rho_4) = \log_n k(t_r + t_c \frac{m^2(1-\frac{1}{n})\rho_4}{2(1-m\rho_4)}) + (m-1)t_c$$

and

$$u_4 = \rho_4 k$$

where k = number of processors in the system
t_r = transit delay of a switch
t_c = cycle time of a switch
n = size of switch (n×n)
m = number of packets per message
ρ_4 = average number of messages entered by each P_i per cycle time

An exact calculation of $D_4(\rho_4)$ is possible since $\rho_4 = u/k$ and can be directly substituted into the delay function.

3.5 Omega Interconnection PE to PE architecture

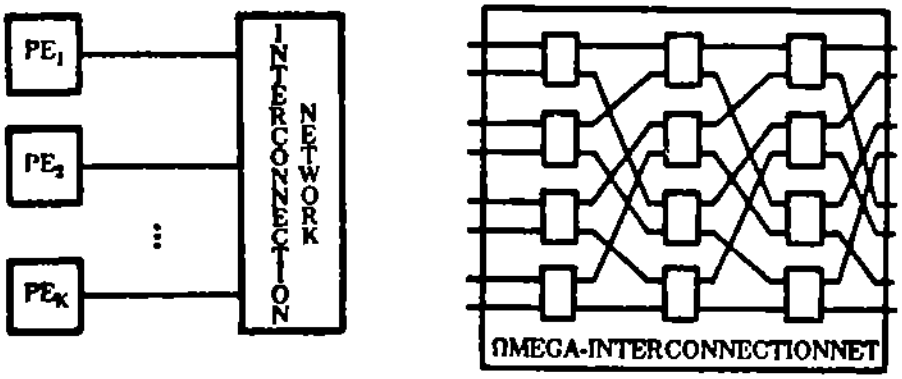
In every architecture considered so far we have $d_{k,\ell} = 1$. Figure 8 shows an architecture for which $d_{k,\ell} \neq 1$. The organization of the processors and memories differs from that in Figure 7(iv) in that each memory is associated with each processor forming a single element called the processing element (PE). Thus a PE to PE configuration is shown in Figure 8. Note that this time the distance between the processors is $d_{k,\ell} = 1,2$. For the average queueing delay per message through the omega interconnection a formula is given which is the same as in architecture (3.4) but it is stated in more general terms by [NOR85] and is as follows:

$$D = \frac{u}{2(1-u)} \times m \times st \times (1-\frac{1}{u})$$

where u = utilization of the interconnection by a processor per cycle time
m = number of packets per message
st = number of network stages
$n \times n$ = size of switch

The above formula is applicable for the general class of Banyan interconnection

Architecture



Delay function

$$D = \frac{u}{2(1-u)} \times m \times st \times (1-\frac{1}{u})$$

Figure 8. The delay function for nonshared memory interconnection networks u = utilization of the interconnection by a processor per cycle time, m = number of packets per message, st = number of network stages, n×n = size of switch.

networks, delta, omega, indirect cube.

4. THE FLEX/32 ARCHITECTURE AND PERFORMANCE

The FLEX/32 is a collection of computers that can each execute independent instruction streams and can separately access memory. The memory is shared and distributed in several modules as illustrated in Figure 9. There are several choices of boards and they can be mixed.

The shared memory computers communicate by using the common memory. The intercommunication hardware for FLEX/32 is depicted in Figure 10 and consists of two common busses and ten local busses. Common busses are nonmultiplexed, synchronous with a clock rate of 12 Mhz, and with address width and data width of 32 bits. The communication mechanism is implemented by the *Common Control Card* (CCC) and the *Common Access Cards* (CAC). The CAC card provides a local bus to common interfaces, contains 64 to 512 kbytes of common memory and hardware for interprocess synchronization. The CCC card also provides a local bus to common bus interface and arbitrates conditional critical region communication. Only one CCC is needed per machine, additional CCC's allow the FLEX/32 to be partitioned into completely independent machines.

4.2 The Purdue FLEX/32 configuration

The department of Computer Science at Purdue University operates a FLEX/32 multiprocessor system with 7 computers based on National Semiconductor NS 32032 processors each supported by floating point unit. Six computers have 1 Mbyte of local memory and one has 4 Mbytes. The system has 6 CAC cards with 512 Kbytes of common memory each. Each computer can operate in serial mode under UNIX and in multicomputing or parallel mode under the MMOS operating system [FLEX85]. The system is programmed in Concurrent C and Concurrent FORTRAN 77.

4.2 Purdue FLEX/32 system performance

Some parameters that have been used to characterize the performance of MIMD machines [FOX85] are:

k = number of processors (or computers)
t_{comm} = time to communicate one word between processors
t_{fl} = time for single precision floating point calculation
t_{dfl} = time for double precision floative point calculation
t_{int} = time for integer calculation
t_{ext} = time to read or write one word from external devices

The values of these parameters for the Purdue FLEX/32 are given in Table 1. No value is given for t_{fl} because the software does not currently allow one access to single

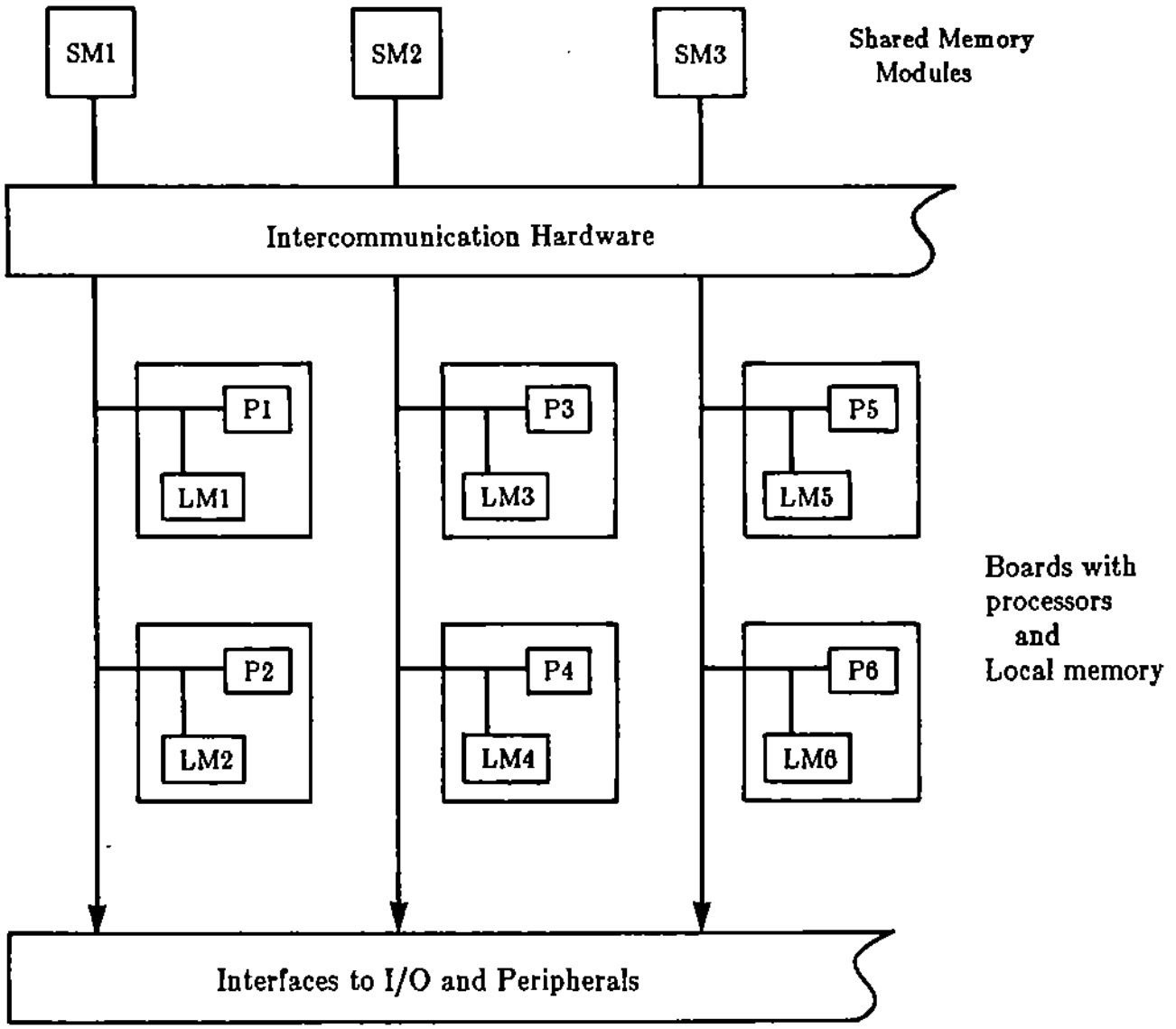


Figure 9. Schematic of the FLEX/32 architecture. There may be up to ten shared memory modules and twenty processor boards.

precision calculations.

The FLEX/32 processors (NS 32032) were compared with VAX 780 single processor computer. It appears that each FLEX/32 computer based on NS 32032 is 1/3 of VAX 780 for double precision floating point calculations and 1/2.65 for integer arithmetic. This agrees closely with comparisons to the VAX 780 by other NS32032 based machines we have. The LINPACK subroutine DCHDC for Cholesky decomposition for symmetric matrices was also used to compare the performance of the FLEX/32-NS32032 processor and the VAX 780. Table 2 gives the performance of LINPACK double precision routine in C under FLEX/32-UNIX and MMOS. The data indicate that MMOS is slightly faster. Table 2 shows that a Fortran implementation of DCHDC is slightly more efficient than the C implementation. Table 2 also shows that a single processor on the FLEX/32 runs DCHDC faster than expected when compared to a VAX 780. The ratios of execution times average about 2.1 while the ratios of processor speeds vary from 2.6 to 3.3. Once again we see that predicting performance of even arithmetic dominated applications cannot be done reliably just by using processor speeds.

Extensive experiments have been performed to measure the communication cost through the common memory. For the Purdue FLEX/32 configuration communication delay appears to be completely independent of the number of processors that are active no matter what they are doing.

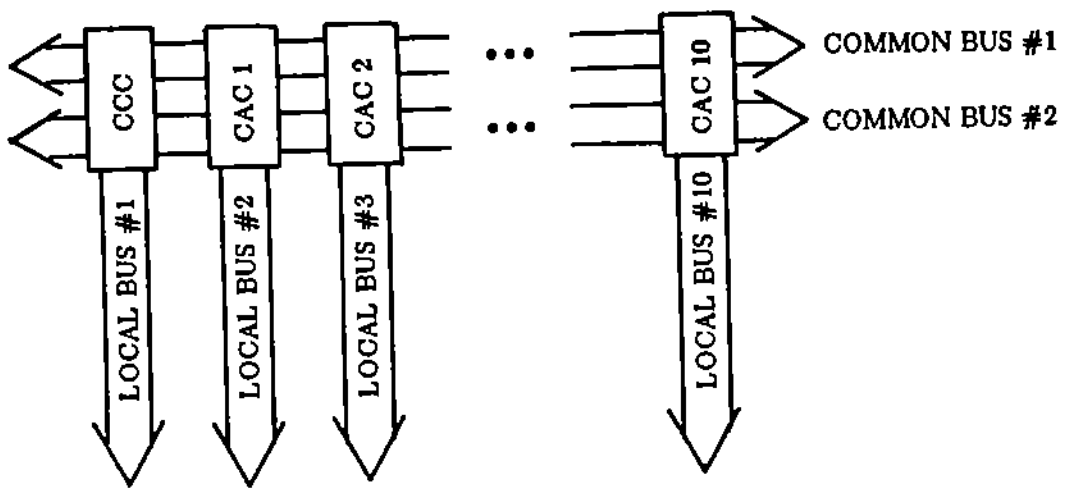


Figure 10. FLEX/32 intercommunication hardware. The common bus operates at a 384 megabit/sec and the local buses have half that capacity.

Parameter	FLEX/32 value μsec	Comments
t_{comm}	6.6	
t_{dfl}	31.5	operands in shared memory
	25.13	operands in local memory
t_{int}	17.05	operands in shared memory
	14.66	operands in local memory
t_{ext}	140	"write" operations
	80	"read" operations

Table 1. Hardware parameters for the Purdue FLEX/32.

N	UNIX-C	MMOS-C	UNIX/MMOS	UNIX-F77	FLEX32/VAX780
32	0.62	0.58	1.063	0.58	2.32
64	3.88	3.52	1.103	3.63	2.18
96	12.08	10.74	1.125	11.15	2.18
128	27.18	24.16	1.125	25.33	2.21
160	51.83	45.72	1.134	48.18	2.16
192	87.92	77.34	1.136	82.56	1.99
224	137.72	120.94	1.138	125.32	2.05
256	204.63	178.48	1.146	190.28	2.08
288	286.38	251.96	1.137	268.25	2.15
320	390.33	342.96	1.138	367.02	2.16

Table 2. Execution times in seconds on a single processor of the LINPACK subroutine DCHDC for Cholesky decomposition of symmetric matrices of order N. Columns 1, 2 and 3 show the effect of running C programs under the two operating systems, UNIX and MMOS. Column 4 shows the time for a Fortran 77 version and the last column gives the ratio of execution time for the same program on a VAX 780.

5. MAPPING COMPUTATION MODELS TO ARCHITECTURES

We now consider how to obtain the schedule graph G'(A) from the original graph G(A). We use a heuristic algorithm based on the minimization of communication cost. This cost is estimated from the data about communication in G(A) and the average communication delay models such as given in Section 2. The details of this algorithm are given in [HOUS83] and we briefly summarize it here for completeness. There are numerous constraints implicit in the mapping problem which complicate a complete mathematical formulation considerably. However, experience shows that this algorithm runs fast and the run time is normally linear in the size of G(A) and the number of processors. The reduction can be viewed as a clustering process that tries to minimize *communication time for data and computation variables* by clustering modules together. This clustering must satisfy the following constraints:

(i) *resource constraints:*
- Every computational module must fit into the memory assigned.
- Data blocks must fit into the memory assigned.
- Computations must have enough processor time.

(ii) *parallelism constraint:*
- Parallel computational modules can not be clustered, they are assigned to different processors.

(iii) *artificial constraint:*
- Processing time on each processor is limited to T (a parameter).

The *time frame* parameter T is used implicitly to calculate the utilization (u) of the interconnection network of the machine M due to intermodule communication traffic. The reduction of T forces more and more processors to be used.

We give the results of applying our method to map Applications 1 and 3 to the FLEX/32 architecture. Application 2 is not considered because there is an obvious mapping for it that gives optimal speedup.

5.1 The clustering algorithm

The solution of the reduction problem for a specific time frame T is achieved by the following *heuristic clustering strategy:*

Start

Assign one processor per computation module
Assign data blocks to 'closest' memory

Iteration

1. Select a pair of computation modules for possible merger into one processor which gives maximum reduction in objective function (communication cost).

2. If no constraints are violated, then merges the two modules, otherwise remove the pair from list of eligible pairs.

Experience suggests that this heuristic strategy "solves" the reduction problem in approximately linear time. If the graph G(A) can be separated into disjoints subgraphs by synchronization nodes (nodes with parallelism = 1), then the subgraphs are treated separately and the resulting subclusters joined in arbitrary order. The input to the clustering algorithm is as follows:

Algorithm specification

- the application graph G(A). Recall that this graph explicitly contains requirements for processing, memory and communication plus the precedence of the computation modules. From this information one may easily derive the synchronization delay or blocking times of the modules.

Architecture characteristics

- communication models (delay function)
- interconnection network bandwidth
- G(M) if different than G(A)

Resource constraint

- time frame parameter T

5.2 Mapping application I to the FLEX/32 architecture

The Cholesky parallel algorithm [O'LEA85] was implemented in Concurrent C [FLEX85] for block symmetric matrices. The size of each block is denoted by N^2 and the number of blocks is k^2 where k represents the number of active processors. The graph of the applications is depicted in Figure 2 for 4 by 4 block matrix. The FLEX/32 communication delay function is $t_{comm} * k * c_{i,j}$ where $c_{i,j}$ is the number of units of data transferred between modules i and j. Table 3 gives the mappings obtained versus k. Figure 11 shows the clusters for the 5 by 5 case.

This application allows us to test the claim that the heuristic mapping algorithm has run time proportional to the number of nodes and processors. In this case the number of processors is k and the nodes k^2. We have for this algorithm

k	2	3	4	5	6
execution time (sec.)	0.68	1.48	4.47	8.53	19.81
100 * (execution time)/k^3	8.5	5.5	7.0	6.8	9.2

k	Cluster #1	Cluster #2	Cluster #3	Cluster #4	Cluster #5	Cluster #6
2	1,2	3,4				
3	1,2,3	4,7,8,9	5,6			
4	1,2,3,4,15,16	5,9,11,12,13	6,7,8	10,14		
5	1,2,3,4,5	6,11,16,18,21, 23,24,25	7,8,9,10	12,18,19,20,22	13,14,15	
6	1,2,3,4,5, 6,22,23,24	7,13,19,25,31	8,9,10,11,12 29,30	14,20,26,28,32 34,35,36	15,16,17,18	21,27,33

Table 3. The clustering of the nodes of G(A) produced to obtain G'(A) by the heuristic algorithm. For k processors the nodes are numbered using the same pattern as in Figure 2 and then assigned to clusters as indicated in this table.

5.3. Mapping application III to the FLEX/32 architecture

The application of the heuristic mapping algorithm to the graph G(A) shown in Figures 5 and 6 gives the clusters shown in Figure 12. Each node of G(A) is replaced by the cluster number to which it belongs in G'(A). The interlacing of the clusters makes the display such as in Figure 11 too complex. The time to compute this clustering is 5.484 seconds.

5.4 On the behavior of the mapping algorithm

When the time frame parameter T changes, different clusterings in G(A) can be obtained. Let

T_{PAR} = the shortest time frame for which all allocated processors can run the application A in parallel,

T_{MIN} = shortest time frame for which the application A can run under the imposed resource utilization constraints.

Our observation has been that $T_{MIN} \neq T_{PAR}$ when the intermodular communication cost is very low. For example, in the case of Cholesky decomposition (Figure 2) there are ten different clusterings which are summarized in Figure 13. Normally, $T_{MIN} = T_{PAR}$ and the mapping algorithm produces a unique solution. If we define as an *optimum clustering* for G(A) as the one with minimum elapsed time and minimum communication cost then it is easy to verify the following observations.

Lemma: If $T_{MIN} \neq T_{PAR}$ then the optimum clustering is the one that corresponds to time frame $T = T_{PAR}$ or the one with minimum number of clusters.

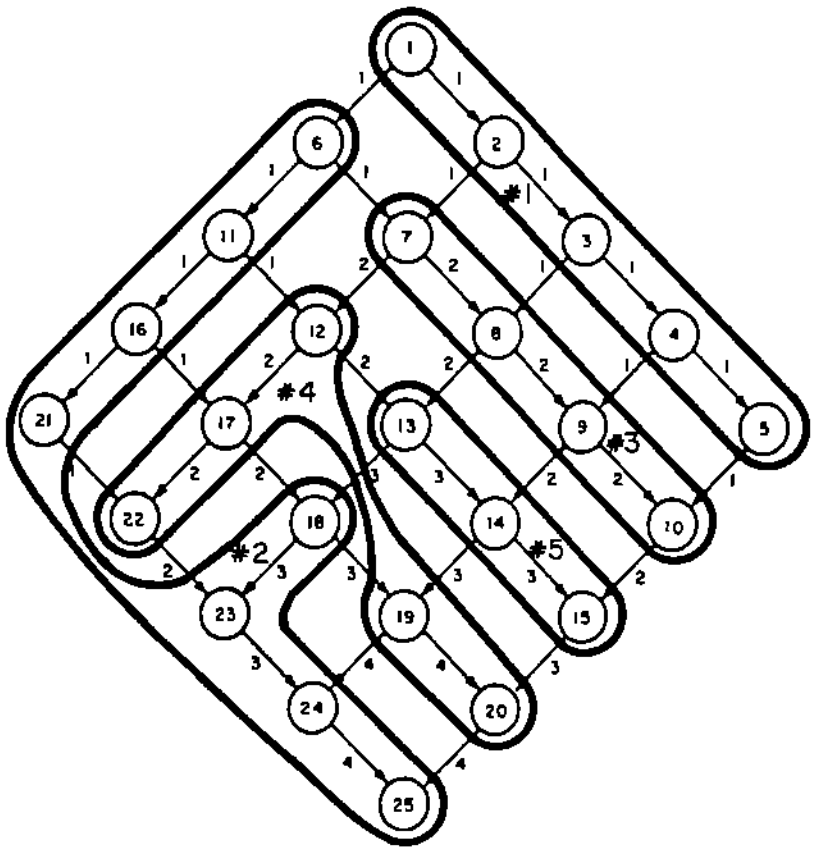


Figure 11. The clusters obtained by the mapping of G(A) into G'(A) for the 5 by 5 Cholesky computation with N=240. The predicted elapsed time is 194 seconds to make this computation.

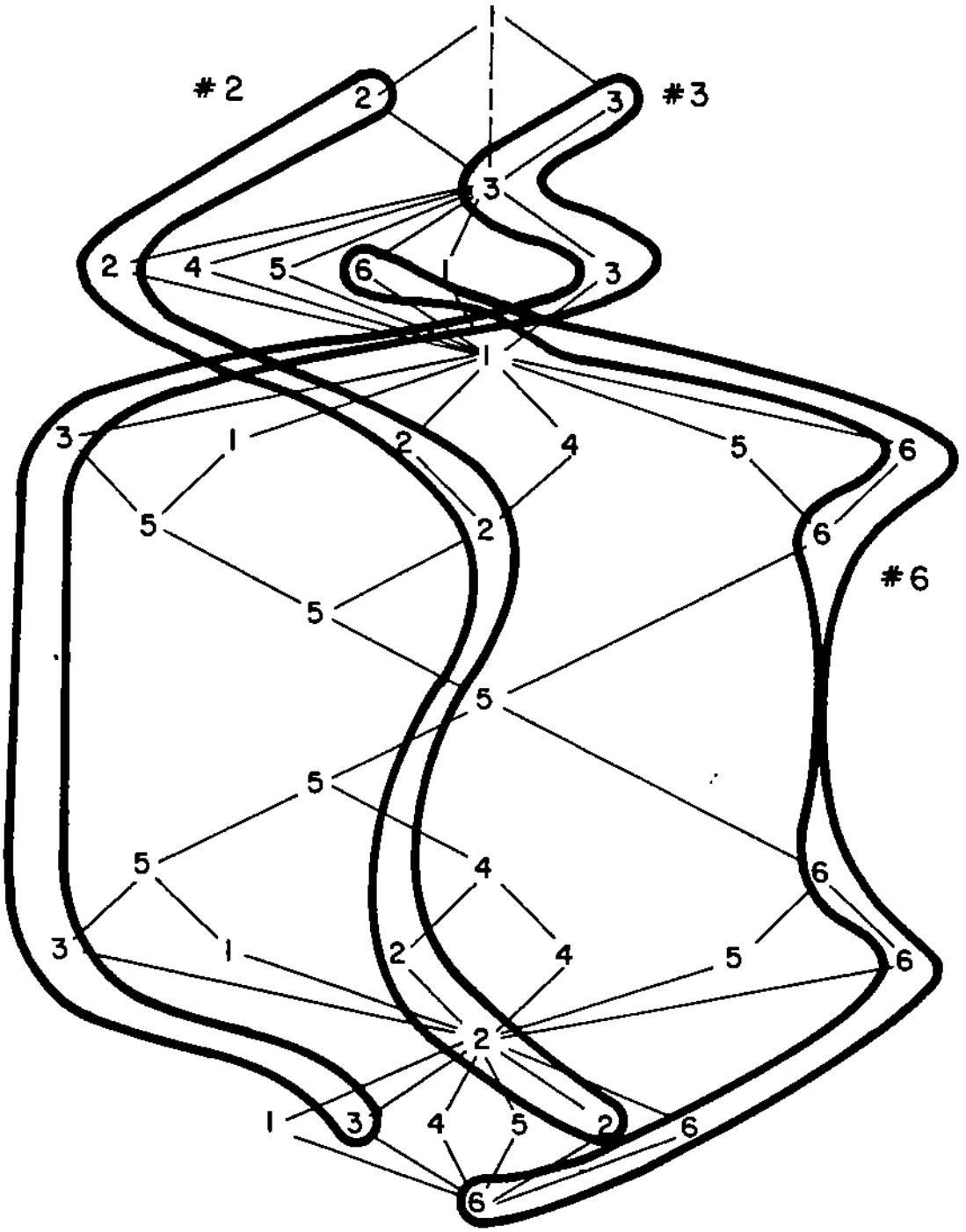


Figure 12. Partition of the graph G(A) of Applications III into 6 clusters using the heuristic algorithm with T=66.37. The numbers at the nodes indicate to which cluster that node belongs. Clusters 2, 3 and 6 are also indicated by the heavy lines.

For the different clusterings $G'(A)$ of Figure 13, the elapsed time versus the time frame T are plotted in Figure 14. These results were obtained using simulation on the FLEX/32.

This observation is supported by our experimental results. Table 4 shows the effect of using more processors than required for Application I (the 4 by 4, N=240 case). Communication is so fast in the FLEX/32 compared to computation that one sees little effect in actuality. We artificially increased the communication requirements by a factor of 100 and those results are shown in column 3 of Table 4. Then the adverse effect of having too many processors is evident.

The current implementation of the graph reduction phase is capable, by using an iterative procedure, to identify all possible solutions and the breakpoints as in Figure 13. Also, it is possible to estimate the elapsed time provided we are able to predict the blocking time of each module in G(A) due to the precedence of computations. The workload analysis based on SIMON or FLEX/32 provides this information. It turns out that T_{PAR} is a close *upper bound* of the elapsed time when the blocking time of each module is incorporated as part of the processing time of the modules. We believe that this approach to predicting the elapsed time is reasonable since the blocking time is solely an attribute of the application algorithm. The importance of the elapsed time is twofold. (a) For the same application the performance of different multiprocessor system architectures can be compared by comparing their elapsed times. The advantage of parallel processing can also be investigated by comparing the elapsed times on a parallel architecture system to a uniprocessor system. (b) Given different initial partitions G(A) of the same application A, their performance can be compared by looking at the elapsed time of the different partitions running on the same multiprocessor system.

6. PERFORMANCE OF APPLICATION/ARCHITECTURE PAIRS

In this section we present actual performance data for the three applications on the FLEX/32. We analyze the speedup obtained and correlate this with the computation structure and with the predicted values from our allocation algorithm.

6.1 Application I/FLEX32 pair

Consider an N by N matrix and the Cholesky decomposition described in Section 2.1 and assume that is to be implemented using k processors. One way is to partition the matrix into k^2 blocks. Then the computational complexity of this algorithm is at least $0(N^3/k)$ and the communication complexity is at least $0(N_2/k)$. This algorithm was implemented in concurrent C and ran on the FLEX/32 using different values of N and k. The mapping was determined by the allocation algorithm assuming a linear delay model $t_{comm} \times k \times c_{i,j}$. Table 5 shows the speedup and efficiency obtained. The entry for k=8 was obtained by partial simulation. Efficiency is the speedup obtained

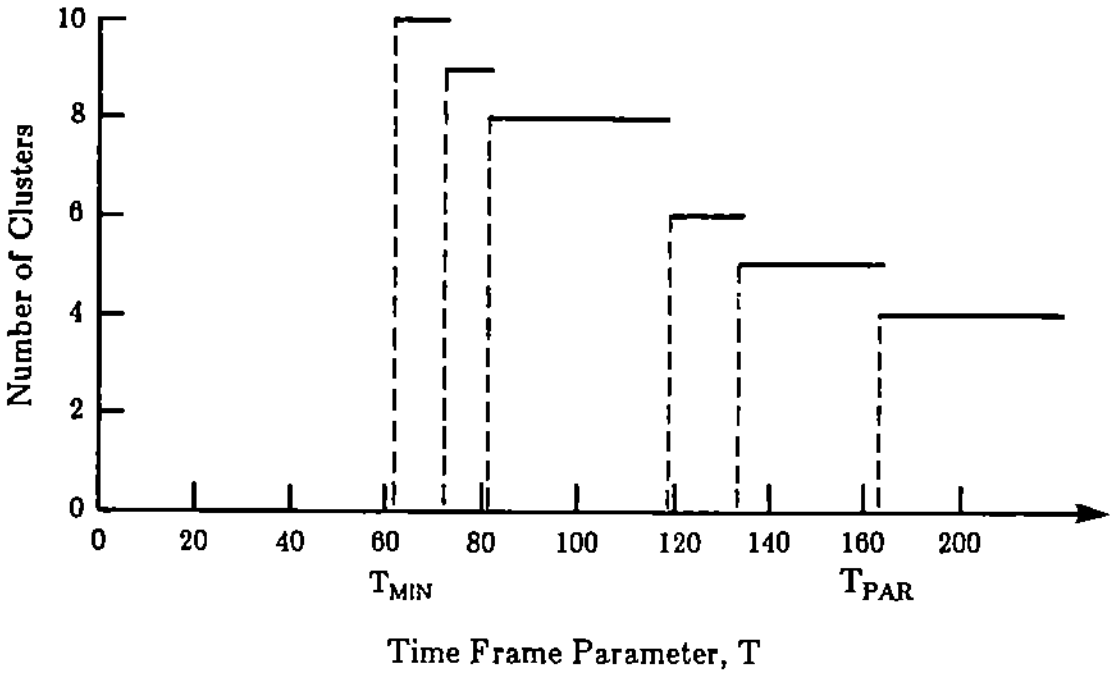


Figure 13 The number of clusters obtained for different values of the time frame parameter T for the Cholesky decomposition application of Figure 2 with N=240.

Number of processors	Elapsed time	Elapsed time with communications requirements increased by 100.
4	228.68	747.7
5	229.44	801.6
6	230.24	838.0
7		910.1

Table 4. Effect of having more processors than required for the parallelism of an application in the case of Cholesky decomposition with a 4 by 4 system and N=240 (see Figure 2).

divided by k. It is worth noting that the communication complexity is reduced with respect in a factor close to the speedup.

Number of active processors	Elapsed Time	Speedup	Efficiency
1	408.46		
2	347.10	1.18	.59
3	276.98	1.48	.49
4	228.67	1.79	.45
5	194.36	2.1	.42
6	169.54	2.4	.40
8	102.38	4.0	.49

Table 5. Elapsed time, speedup and efficiency obtained for Application I on the FLEX/32.

One should not expect speedup of about k (or efficiency almost 1) for this application. As seen in Figure 2, during the computation the parallelism of G(A) grows steadily to its maximum and then decreases steadily back to 1. Thus a course grained partitioning such as we use should use, on the average, about half the processors so we expect the efficiency to be about 0.5 which corresponds well with the observed results.

In order to study the effect of communication cost on the computation we have artificially increased it by a factor of 10 and 100. This increases the computational time for a 4×4 Cholesky decomposition graph N=240 and 4 active processors as follows:

Communication Cost	Elapsed time	Ratio to normal case
Normal	228.67	-
10 times normal	277.39	1.2
100 times normal	747.67	2.7

One of the objectives of our methodology is to predict the performance of some algorithm/architecture pairs. It appears that T_{PAR} can be used for this purpose. Table 6 shows the predicted elapsed time for various computational graphs of Application I or FLEX/32 system.

k	Predicted Elapsed time	Predicted Speedup	Measured Speedup
2	316	1.29	1.18
3	243	1.68	1.48
4	162	2.52	1.79
5	159	2.57	2.10
6	140	2.90	2.40

Table 6. Predicted and measured speedup. The predicted time is T_{PAR} from the allocation algorithm, the measured time from Table 5.

6.2 Application II/FLEX32 pair

A straight forward estimation of the computational complexity of the spline collocation method indicates that it requires $O(N^2)$ arithmetic operations per iteration where 1/N is the size of the partition in the x and y directions. Ideally, distributing the work among k processors reduces the computational complexity in a parallel algorithm to $O(N^2/k)$. The performance curve in Figure 14 indicates ideal speedup for large N. Table 7 presents the timings obtained with the FLEX/32 with a single processor and the VAX 780. These data indicate that the VAX 780 is about twice as fast as a FLEX/32 processor. This agrees with the data in Table 2 and supports the conjecture that the VAX 780 is not as much faster than FLEX/32 processor as the raw speeds of the processors would suggest.

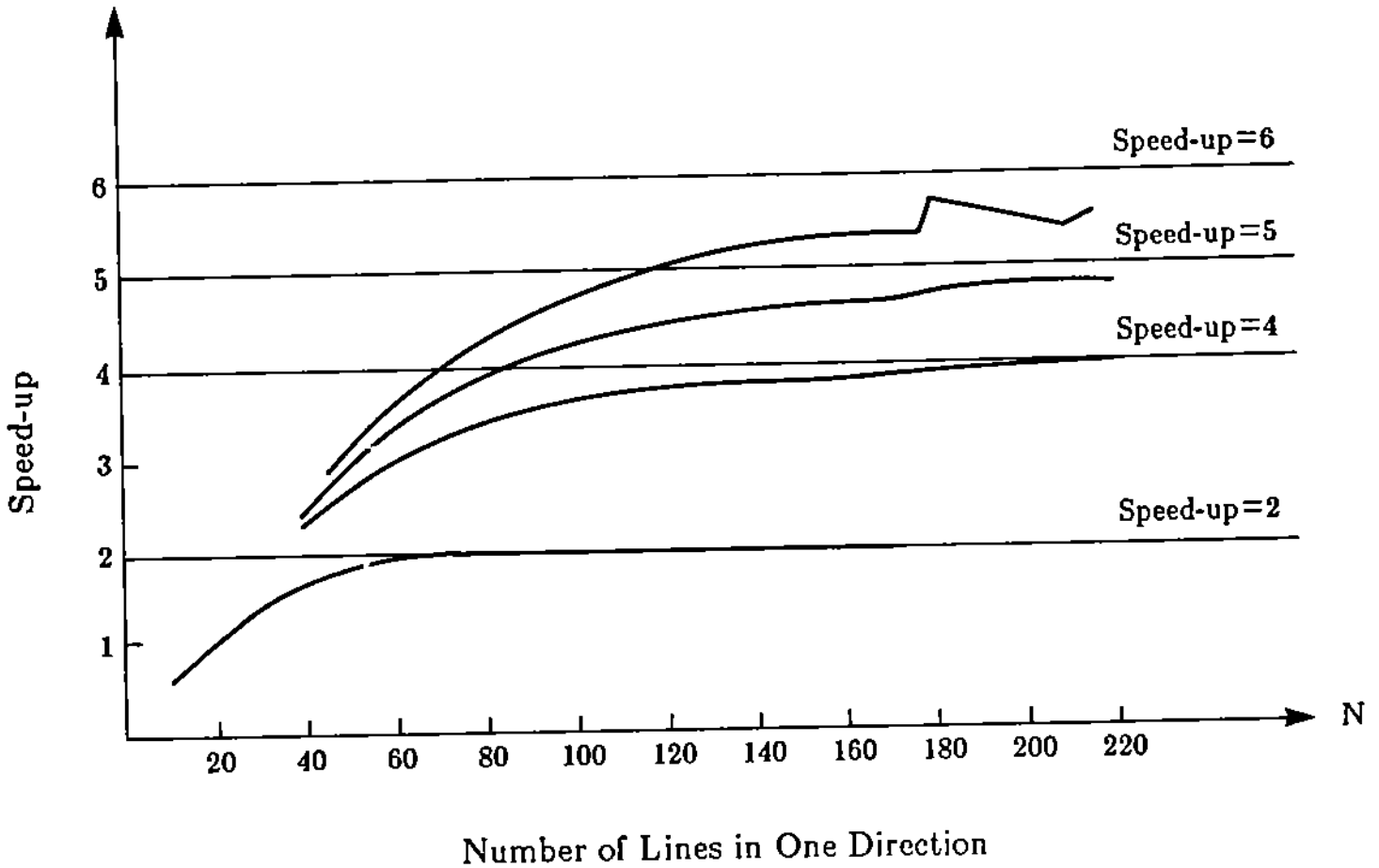


Figure 14 Measured speedup for Application II on the FLEX/32. The rough nature of the curve for 6 processors is due to variations from properties of neighboring values of N (e.g. at N=174, 176 and 180).

N	FLEX-UNIX	FLEX-MMOS	VAX-UNIX	FLEX-MMOS/VAX-UNIX
12	0.30	0.24	0.08	3.00
24	1.15	0.82	0.32	2.56
48	4.18	3.18	1.38	2.30
96	15.70	12.52	5.85	2.14
120	23.85	19.54	9.32	2.09
150	38.03	30.52	14.20	2.15
180	52.83	46.61*	20.93	2.23
210	72.29	58.29*	28.43	2.05
216	78.65	63.43*	29.92	2.12

* predicted values

Table 7. Time to execute one iteration on a single processor computer under different operating systems. The computation is one iteration of the spline collocation method (see equation 2.1) for an N by N mesh.

We observe that the computational complexity and communication complexity are both $O(N^2/k)$ here. For the FLEX/32 the ratio

$$t_{comm}/t_{diff} = 6.6/31.5 = 0.21$$

is quite small which is necessary to obtain the high efficiency observed here. For the particular case N=180 we have the following results

k = number of processors	1	2	3	4	5	6
speedup	1	2.00	2.47	3.88	4.77	5.65

The speedup curves as shown in Figure 14 are typical of those seen in other applications and for other machines. This same data is redisplayed in Figure 15, we plot the number of processors times the elapsed execution time for one iteration of this computation. If we had perfect speedup, then these curves would be constants. We observe that they have a slight rise, almost linear. In particular, each of these has risen about 3.5 at k=6. This suggests that

$$\text{Elapsed time} = C_1N^2/k + C_2 = 0.00135\,N^2/k + 0.75k$$

which gives.

$$\text{Speedup} = k(1 + (k-1)C_2/(C_1N^2))$$

Thus there is a very plausible fixed overhead of C_2 in the elapsed time per processor. The significance of this behavior is that C_2 is an overhead independent of N and thus

one that becomes less significant as the problem size increases.

6.3 Application III/FLEX32 pair

Application III requires 252.86 seconds to run on a single FLEX/32 processor. A brief examination of Figure 6 suggests that a substantial speedup is possible for 6 processors, but somewhat less than 6 is expected. There is a drop in parallelism in the center of the graph (where the multifrontal method merges) which involves substantial computation and the "create plot data structure" node is sequential and sizable. When this application is run with 6 processors on the FLEX/32 using the clustering of Figure 12, it takes 72.03 seconds. This is a speedup of 3.34.

An analysis of Figure 5 shows that the maximum possible speedup is 3.92 if one assumes that communication is instantaneous. Recall that our heuristic algorithm does not attempt to minimize elapsed time directly, so the speedup obtained is reasonable.

The time frame parameter T used to achieve the clustering is 66.37. This gives the predicted speedup of 252.86/66.37 = 3.81 which is between maximum possible and the observed value.

7. CONCLUSIONS

We see that the heuristic partitioning algorithm works well even for a machine like the FLEX/32 where communication costs are negligible compared to computational costs. The applications considered have a high correlation between communication and computational costs which is not surprising, this is probably fairly common. The seven processor FLEX/32 at Purdue exhibits no communication degradation even with all processors continually accessing shared memory. We expect this situation to continue if the number of processors is increased to the one cabinet limit (20 processors). Our projection is that the FLEX/32 boards can be upgraded with processor ten times faster than the current ones without adversely impacting most computations. We are surprised to see on two unrelated, arithmetic dominated applications that the FLEX/32 runs half as fast as a VAX 780 even though the VAX processors are 3 times faster.

Optimal speedup requires that the degree of parallelism be maximal most of the time. Our approach only exploits coarse grain parallelism and two of the applications can use both coarse and fine grain parallelism. Even so, we observe good speedup of about k/2 for k processors. The third application (spline collocation) naturally has high parallelism and, for any number of processors, we observe optimal speedup as the problem size increases. In this case the lack of perfect speedup seems to be due to a small constant "start up cost" per processor. This small cost has a large apparent effect on traditional "speedup versus size" graphs.

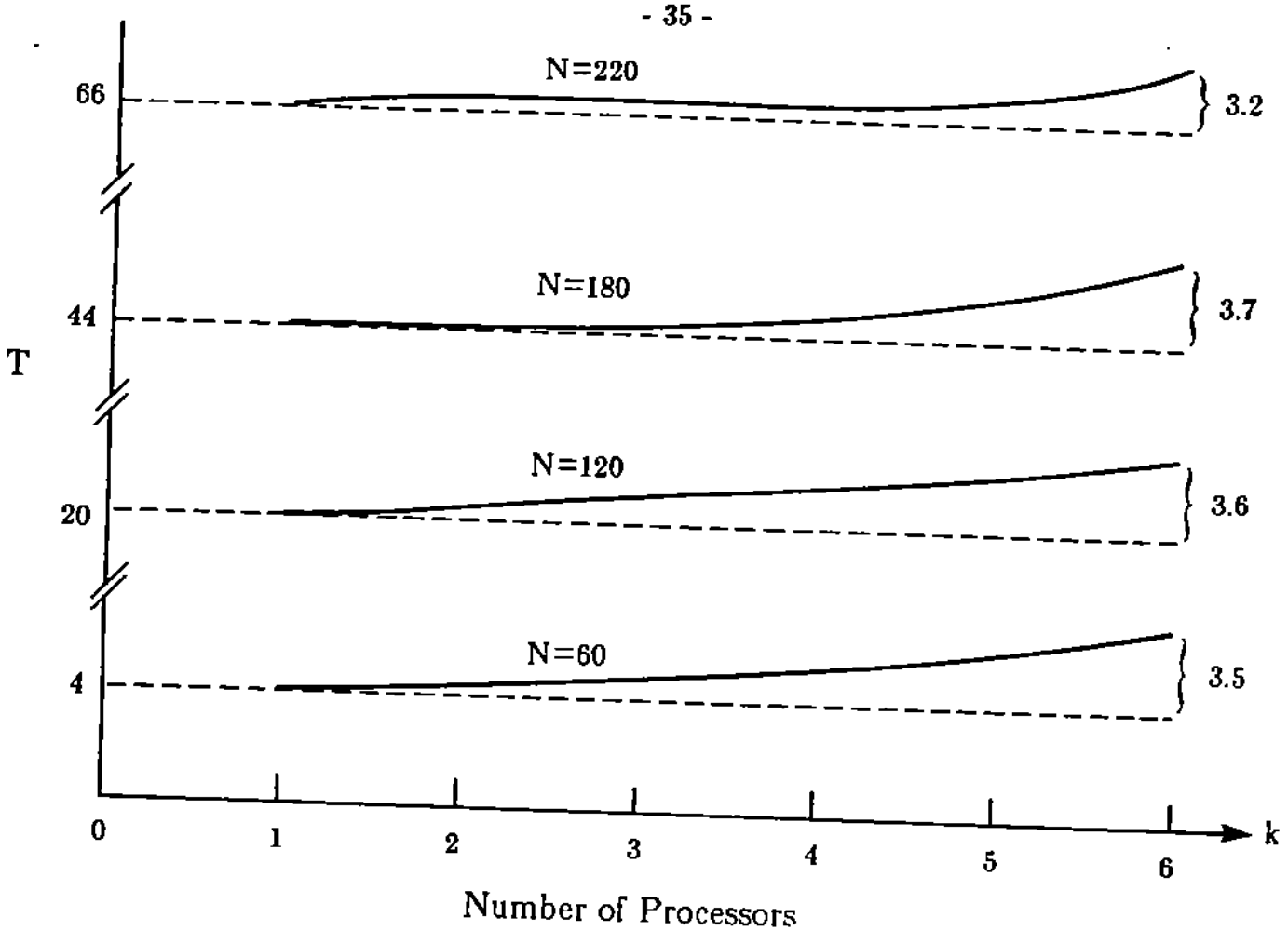


Figure 15. Plot of k times elapsed computing time for Application II for N=60, 120, 180 and 220. Note that the difference from a constant in these curves at k=6 is almost the same.

8. REFERENCES

[BERM84] Berman, F. and Snyder, L., "On mapping parallel algorithms in parallel architectures," *Proc. Internat. Conf. Parallel Processing,* (1984) 307-309.

[BERM85] Berman, F., Goodrich, M., Koelbel, C., Robinson, III, W. J., and Showell, K., "Prep-p: A mapping preprocessor for chip computers," Proc. Inter. Conf. Parallel Processing, (1985) 731-733.

[BOKH81] Bokhari, Shamid, H., "On the Mapping Problem," *IEEE Computers,* C-30, (1981) 207-214.

[FLEX85] Flexible computer corporation, *FLEX/32 Multicomputer system overview,* Doc. No. 030-0000-002, Second Edition, June (1985).

[FOX85] Fox, G, The performance of the Caltech Hypercube in scientific calculations, Caltech Technical Report, April 5 (1985).

[FUJI85] Fujimoto, R. M., "The SIMON simulation and development system," *Summer Computer Simulation Conference,* 1985 (Univ. of Utah).

[GANN84] Gannon, D. and J. Von Rosendale, On the communication complexity of parallel numerical algorithms, *IEEE Trans. Computer,* Vol. C-33, No. 12, pp. 1180-1194, December 1984.

[HOUS84] Houstis, Catherine E., "Allocation of real time application in distributed systems," (1984) submitted for publication.

[HOUS83] Houstis, C. E., Houstis, E. N., and Rice, J., "Partitioning and Allocation of PDE Computation to Distributed Systems," *PDE Software: Modules Interfaces and Systems,* Edited by B. Enguist and T. Smedsaas, North Holland, (1983), 67-85.

[HOUS86] Houstis, E. N. and Rice, J. R., Heuristic methods for reducing the parallelism in computational graphs, Technical Report, Computer Science, Purdue University, 1986.

[HWAN84] Hwang, Kai, and Briggs, Fayé, *Computer Architecture and Parallel Processing,* McGraw-Hill, 1984.

[JENN77] Jenny, C. J., "Process partitioning on distributed systems," Digest of paper *National Telecommunications Conference,* (1977) 31:1-31:10.

[KLEI85] Kleinrock, Leonard, "Distributed systems," *Communications ACM,* 28, (1985), 1200-1213.

[KRUSS83] Kruskal, C., and Snir, M., "The performance of multistage interconnection nets for multiprocessing," *IEEE Trans. Computers,* (1982), 1091-1098.

[MARS83] Marsan, M. A., and Gerla, M., "Markov models for multiple bus multiprocessor systems," *IEEE Trans. Computers,* C-32, (1983) 239-248.

[MARS83] Marsan, M. A. Balbo, G., and Conte, G., "Comparative performance analysis of single bus multiprocessor architectures," *IEEE Trans. Computers,* C-31, (1983) 1179-1191.

[NORT85] Norton, Alan and G. F. Pfister, "A methodology for predicting multiprocessor performance," Proc. Internat. Conf. Parallel Processing, (1985) 772-778.

[O'LEA86] O'Leary, D. P. and G. W. Stewart, "Data-flow algorithms for parallel matrix computations," *Communication ACM*, 28, (1985) 840-853.

[O'LEA86] O'Leary, D. P. and G. W. Stewart, "Assignment and scheduling in parallel matrix factorization" Lin. Alg. Appl., (1986), to appear.

[WILL83] Williams, E. A., "Assigning processes to processors in distributed systems," *Proc. Internal. Conf. Parallel Processing*, (1983) 404-406.