1989

The algorithm mapper: A system for modeling and evaluating parallel applications/archicture pairs

C. E. Houstis

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

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

S. M. Samartzis

D.L. Alexandrakis

Report Number:
89-854

A USER GUIDE TO THE ALGORITHM MAPPER
THE ALGORITHM MAPPER: A SYSTEM FOR MODELING AND EVALUATING PARALLEL APPLICATIONS/ARCHITECTURE PAIRS

C. E. Houstis
E. N. Houstis
J. R. Rice
S. M. Samartzis
D. L. Alexandrakis

CSD-TR-854
February 1989

A USER GUIDE TO THE ALGORHTM MAPPER

THE ALGORITHM MAPPER: A SYSTEM FOR MODELING AND EVALUATING PARALLEL APPLICATIONS/ARCHITECTURE PAIRS

*C.E. Houstis**

Computer Science Department
University of Crete
Heraklin, Greece

*E.N. Houstis**, J.R. Rice***
*S.M. Samartzis**, and D.L. Alexandrakis***

Computer Science Department
Purdue University
Technical Report CSD-TR-854
CAPO Report CER-89-5
January 1989

ABSTRACT

We present a methodology for evaluating the performance of application programs on distributed computing systems. An application A is represented by an annotated graph $G(A)$ giving its requirements for processing, memory and communication, plus the precedence between computation modules. Machines are represented by a similar graph $G(M)$ and the methodology is to map $G(A)$ into $G(M)$. The mapping problem is subdivided into three steps (s1) a reduction of the parallelism of $G(A)$ to that of $G(M)$, (s2) scheduling the computational modules to (nearly) minimize communication costs, and (s3) actual layout of resulting graph into $G(M)$. The two technical problems addressed here are (1) using communication delay models to simplify $G(M)$ and the step (s3), (2) scheduling the modules (step (s2)). Our communication models apply well to "uniform" architectures, we explicitly consider the following four: single bus and common memory, single bus and distributed memory, multiple bus and distributed common memory, Banyan interconnection and distributed common memory.

* This research was supported by NSF grant DMC-85080684A1 and ESPRIT Project 1588. Most of this was completed while she was with the Electrical Engineering Department of Purdue University.
** This research was supported by ARO grant DAAG29-83-K-0026 and AFOSR grant 84-0385.

1. INTRODUCTION

Parallel processor systems are efficiently utilized when the computations they are assigned can be performed in parallel and they are mapped in such a way as to maximize their speed up. Such systems can be interconnected in a variety of ways, which can be roughly classified as shared memory and non-shared memory. In shared memory systems, processors usually have their own local memory and communicate by contending for common resources, such as an interconnection network and shared memory. Shared memory can be either distributed among the processors in the form of shared memory modules, or it can be common. The interconnection network along with the shared memory can be regarded as the system's communication network. Interconnection networks are commonly from the class of multiple bus systems, ranging from the simplest configuration of a single bus up to a highest bandwidth configuration of a crossbar switch. The class of Banyan networks is also common. In the case of the multiple bus interconnection, the distance between the processors is clearly one, since the interconnection provides a direct connection from every processor to every other processor (through the common memory). In the Banyan case, the distance between processors can also be regarded as one, since on the average the routes between processors uses the same number of intermediate switches.

In non-shared memory architectures, a variety of interconnections exist. For example, processors can be placed at the nodes of a grid or at the nodes of a cube, etc. In these cases, the distance between processors is not necessarily one and it depends on the routes chosen between processors and the geometry of the architecture.

A number of methodologies exist in the literature which address the problem of mapping computations to parallel systems and they can be divided into two categories; (a) the methods that assume implicitly or explicitly that the distance between processors is one and (b) the methods that assume that this distance is different from one. Our methodology explicitly assumes the distance between processors is one and thus we review mainly such methodologies.

We first state the *mapping problem.* We consider an application A to be a computation with four properties: processing requirements, memory requirements, communication requirements and precedence (or synchronization) between the subcomputations. We visualize the computation broken into *computational modules* which are nodes of a precedence graph for the computations. We note the processing and memory requirements at each node of the graph. We note the communication requirements along each link or edge of the graph. This annotated graph is called G(A).

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 general, the mapping problem has three somewhat independent steps:

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

We apply our methodology to several applications, mainly numerical or real-time.

2. MODELING APPLICATION/ARCHITECTURE PAIRS

2.1 Review of Existing Modeling Methodologies

Most of the existing algorithms have addressed Step 1, [CHU 80], [CHU 87], [EFE 82], [HAES 80], [GYLY 76]. In general, the approaches used can be classified as graph theoretic, integer programming and heuristic methods.

In [CHU 80], a graph theoretic and an integer programming approach is used to solve the scheduling or allocation problem, which is defined as the assignment of M modules to N identical processors, so that interprocessor communication is minimized. The communication network connecting the processors is not described explicitly and the solution implicitly assumes a shared memory architecture.

In [CHU 87] a heuristic approach to a task allocation for distributed systems is presented. The issues discussed are very similar to what we have examined. The objective function in [CHU 87] is the minimization of the maximum processor loading. This has produced a number of differences in their analysis and solution of the problem. They are also not concerned about parallel module execution.

A heuristic approach to module allocation by minimizing interprocessor communication subject to a load balancing constraint is suggested in [EFE 82]. This approach clusters modules into processors and contains a mechanism to solve Step 2 of the mapping problem as follows: N processors are assumed and the modules are clustered heuristically into possibly $N + K$ clusters. A second heuristic, a module reassignment algorithm, is to divide the K clusters among the N processors when possible, by balancing their load.

In [HAES 80], a graph theoretic approach is used. N modules in the application graph are initially associated with N available processors and are regarded as resident modules. The algorithm then divides the rest of the graph into N clusters, which are centered at the resident modules by minimizing communication between processors. In [GYLY 76], a two module clustering algorithms are used to search for "eligible" pairs of modules, eligible in the sense that when they are assigned to the same processor, the greatest possible interprocessor

communication is eliminated. This "fusion" continues until all eligible pairs are fused. The implicit assumption here is that the number of processors is equal to the number of clusters obtained. A shared memory architecture is also implicitly assumed, since the distance between processors is not considered.

For shared memory architectures, the approaches based on graph theoretic methods [CHU 80], [JENN 82] and integer programming methods [CHU 80], result in fairly complex algorithms which prohibit their use for applications with large graphs. The heuristic approaches [EFE 82], [GYLY 76], are more promising and simple to use.

In all of the above models, the interprocessor communication is measured in terms of the amount of data transferred among modules assigned to different processors. These models fail to incorporate the performance characteristics of the parallel system. In our model, data transfers are assigned a communication cost which includes he queueing delay incurred by communicating processors. This requires the performance analysis of the parallel systems architecture. The resulting performance measure is the queueing delay versus utilization of the communication network. Two different classes of interconnection networks have been examined, the Banyan and multiple bus. We have experimentally shown that queueing delay affects considerably the allocation decision, thus our approach incorporates the system's architecture into the mapping heuristics. The use of performance models simplifies the systems involvement into the problem. Moreover, it provides the means of examining the performance of application/architecture pairs. The third step in the mapping problem for shared memory architectures is simplified, since the distance between processors is one, and it is an arbitrary assignment of the allocation of modules obtained in Step 2 to the systems processors, provided that memory requirements are satisfied.

The approach we propose for shared memory architectures is computationally simple, applies to large and general graphs and it can easily be extended to non-shared memory architectures. Moreover, we have examined a few small problems (Cholesky decomposition) where we know the optimum schedule and we see that our algorithm produces this optimum schedule. The extension of this work for non-shared memory architectures is underway.

3. METHODOLOGY FOR SHARED MEMORY ARCHITECTURES

We have given three steps for the solution of the mapping problem. We deal primarily with Step 1 and Step 2, since Step 3 simplifies when shared memory architectures are used. In Step 1, a heuristic algorithm is used to schedule the application computation modules and data blocks into parallel clusters. The algorithm minimizes queueing delay among processors by assigning module pairs with the most communication to the same processor, provided (a) they do not have to be executed in parallel, (b) they do not overutilize the processor, and (c) they fit

into its local memory. The output of this step is a number of clusters of modules that has the same parallelism as the application. Step 2 is the reduction of the clusters obtained in Step 1 to the number of available processors in the system. This is necessary mainly for two reasons: first, the application's parallelism in general can be much higher than the number of available processors. Second, there are architectures in which the number of processors can be adjusted to equal the number of clusters obtained. Multiple bus interconnection architectures present such a feasibility. In the Banyan interconnection system, the number of processors can be increased (or decreased) only by powers of 2. Thus if the number of clusters obtained is not a power of 2, then it is cost effective to use a number of processors equal to the next higher power of 2, less than the number of clusters. In this case, Step 2 is unavoidable. Step 3 is an arbitrary assignment of the clusters obtained in Step 2 to the system's processors, provided memory constraints are satisfied.

The mapping problem requires the modeling of the application and of the parallel system. The computations of an application are assumed to be partitioned and they are modeled by a precedence directed logic graph. This graph can be stochastic when the data flow in the application is not known a priori as is often the case is in real time applications or it can be deterministic as often in the case in numerical applications. Next we describe both a stochastic and a deterministic graph.

3.1 A Data Flow Graph Representation

We denote by $G = (M,L)$ a directed parallel, cyclic and weighted AND - EOR logic graph. Throughout, we assume that the partition of each application is represented by such a graph. Let $M = \{m_i,\ i = 1, 2, \ldots, N_P\}$ be a set of weights corresponding to each program (module) of the graph. Each m_i represents the execution time requirement of the i-th program. For each pair of programs (i,j), we denote by p_{ij} the probability of passing control from i to j and γ_{ij} the expected amount of data transferred. Data transfers are measured in an ITUs (Information Transfer Unit) which are the smallest unit of information whose queueing delay due to communication can be determined. Each link in the graph L is associated with the weights $L = \{l_{ij} = (\gamma_{ij},p_{ij});\ i,j = 1, \ldots, N_P,\ i \neq j\}$. Note that L also represents the precedence graph for the application. If program i passes information to program j, then i precedes j in the execution. Information may flow in both directions between a pair of program modules, we assume the application is such that no infinite loops exist. Note that G is a stochastic graph.

The synchronization requirements of the partitioned application are defined in terms of an AND or EOR logic in the I/O of each node. It is worth noticing that an AND logic on the output links of a node indicate the potential parallel execution of the modules associated with these links. Figure 1, presents an example of a stochastic model that includes internal data blocks

referenced by each module. These internal data blocks are associated by the code and are not part of the data from the application. We want to determine the total processing time requirements of the application. For this we apply a Markov Chain analysis [LOW 73] to the stochastic graph G and transform it into a deterministic graph G'. The transformation G' is defined by (M',L') where

$$M' = \{m_i' = m_i f_i;\ i = 1, \ldots, N_P\},$$

$$L' = \{l_{ij}' = f_i p_{ij} \gamma_{ij} + f_j p_{ji} \gamma_{ji} \text{ if } i > j,$$

$$l_{ij}' = 0 \text{ if } i \leq j \text{ for } i,j = 1, 2, \ldots, N_P\},$$

For each program i, f_i is the number of times it is executed, m_i' indicates its total processing time requirement, while l_{ij}' is the total amount of ITUs transferred between programs i and j. The parallelism of the application (represented by AND logic) is implemented in a matrix form by the precedence matrix $\Delta = \{\delta_{ij};\ \delta_{ij} = 1$ if i and j programs can be executed in parallel; $\delta_{ij} = 0$ otherwise$\}$. The *degree of parallelism of the application* is the maximum number of program modules that can be executed in parallel. Our methodology uses the parallelism as assigned by the user. We call this the *assigned degree of parallelism*, it can be less than the actual degree of parallelism. The assigned degree of parallelism may be interpreted as the amount of parallelism that the user wishes to maintain in the computation. The information in the Δ matrix is in the G graph and can be easily obtained from it. Notice that the matrices L', M' and Δ constitute part of the input to the allocation algorithms to be described.

3.2 Performance Analysis of the Parallel System Architecture

Performance models of parallel multiprocessor systems are used to derive the queueing delay processors incur in communicating among themselves. This delay is due to two factors, (a) accessing the common interconnection network and (b) accessing the common memory modules. Several system architectures have been considered namely;

System 1:	Single bus and common shared memory system.	[HOUS 87b]
System 2:	Single bus and distributed shared memory system.	[HOUS 87b]
System 3:	Multiple bus and distributed memory system.	[HOUS 87d]
System 4:	Banyan switch and distributed shared memory system.	[HOUS 87d]

Figure 1. An example of a stochastic data flow graph model.

A comparative performance analysis of these architectures has been performed, and in all cases, the interconnection network queueing *Delay* versus the *utilization* $D(u)$, has been computed. The performance analysis leads to a number of performance measures. The main variable is the average number of *Active Processors*, *AP*, in the system, i.e., processors doing computation in their local memory. From the *AP*, $D(u)$ can be computed as discussed in Section 4.2.3. The *speed up*, *S*, of the application running on the machine must be bounded as follows

$$1 \leq S \leq AP.$$

3.3 Performance Measures of the Algorithm Mapper

The *algorithm mapper* produces schedules of modules to processors and a number of performance measures, which relate to the system use and the efficient execution of the application. The main measures computed are directly related to the analytical systems model, the first is the *average processor utilization* u_p. To define u_p we introduce

k = the number of processors in the system,
A_j = the set of all module indices in $G(A)$ assigned to processor j,
u_p^i = utilization of processor $i = \sum_{j \in A_i} m_j$,

and then define

$$u_p = \frac{1}{k} \sum_{i=1}^{k} u_p^i.$$

Moreover, the average processor utilization u_p, relates to the active processors in the system AP as follows,

$$u_p = \frac{AP}{k}.$$

Our second performance measure in the *speed up* S defined in terms of T_{seq} = execution time of the application by a single processor (sequential execution) and T_{REAL} = execution time of the application by the parallel system. We define

$$S = T_{seq}/T_{REAL}.$$

4. THE *ALGORITHM MAPPER* RESOURCE ALLOCATION SYSTEM

The algorithm mapper software system is composed of three main parts (a) a preprocessor, which takes the partitioned application and produces the information about its graphical representation and the input data to the mapping heuristics, (b) the heuristic algorithms for the mapping problem, and (c) a user friendly interface. The interface is interactive and displays the input and output of the mapping heuristics on a SUN workstation employing color graphics.

The algorithm mapper can be a part of a parallel machine operating system, namely the scheduler and can be applied at load time. It preassumes a partition of the application which we have obtained by applying mathematical techniques to redesign (when necessary) the application's computation with the objective of parallelizing it. Compiler techniques are also applicable when a known computation is investigated for parallelism.

4.1 The Allocation Algorithm

In Step 1, we formulate and solve the module allocation problem. It is stated formally as a constrained minimization problem as follows:

Input: (a) An application which consists of communicating program modules and data blocks.
(b) Specifications of a given distributed system: processor speeds, memory module sizes and $D(u)$ characterizing the interconnection network queueing delay vs. utilization.

Problem: Allocate the application modules and data in order to minimize the queueing delays due to interprocessor communications into (1) clusters of modules allocated to individual processors and (2) clusters of data allocated to memory modules. This is subject to the

(a) *Distributed System Constraints:*
   (i) size of memory modules,
   (ii) processor utilization capacity,

(b) *Application Constraints:*
   (i) a fixed time allowed for executing the application,
   (ii) program modules that can be executed in parallel will be executed in parallel.

Note that the objective of minimum processing time will be achieved as a result of parallel processing and minimizing the queueing delays due to interprocessor communications.

The details of the mathematical formulation of the allocation algorithm can be found in [HOUS 88a,b].

4.1.1 The output of the allocation algorithm

ALLOCATION SOLUTION AND WORKLOAD STATISTICS
number of modules is 41
estimated elapsed time is 70.0 & capacity is 151515.2 time units
allowed real time for processing is 70.0 time units

*** PROCESSOR UTILIZATION ***

id	total (%) utilization	proc (%) utilization	processes assigned									
1	51.57	51.20	1	3	15	20	26	30	36			
2	69.76	69.60	2	7	13	19	22	23	24	25	26	27
3	24.19	23.75	4	5	11							
4	50.20	50.00	6	16	31	30						
5	45.27	45.13	8	17	32	40	41					
6	55.73	55.40	9	10	21	27	33	35				
7	67.74	65.67	10	12	14	29	34	39				
Average:	52.07	51.55										

*** APPLICATION'S COMMUNICATION & PROCESSING COST ***
(in time units)

id	memory inter-module communication cost	interprocessor communication cost	total cluster processing time	total processor workload
1	0	39000	35.8	39335.8
2	0	650	46.8	6548.8
3	0	46500	16.6	46516.6
4	0	28900	35.6	28935.8
5	0	14000	31.6	14831.6
6	0	35200	36.8	35238.8
7	0	220000	46.0	220046.0
Average:	0.00	55028.57	36.1	55064.7

delay per item 0.000007 *** APPLICATION'S COMMUNICATION REQUIREMENTS ***
(in time units)

id	data reference via interconnection	interprocessor data transfer
1	0.0	39300.0
2	0.0	8500.0
3	0.0	46500.0
4	0.0	20900.0
5	0.0	14800.0
6	0.0	35200.0
7	0.0	220000.0

id	memory module			instructions blocked allocated								
	instruction	size	utilization									
1	1	100.00	7.00	1	3	15	20	26	30	36		
2	1	100.00	10.00	2	7	13	19	22	23	25	26	37
3	1	100.00	3.00	4	5	11						
4	1	100.00	4.00	6	16	31	30					
5	1	100.00	5.00	0	17	32	40	41				
6	1	100.00	6.00	9	10	21	27	33	35			
7	1	100.00	6.00	10	12	14	29	34	39			

id	memory module			instructions blocked allocated								
	instruction	size	utilization									
1	1	100.00	7.00	1	3	15	20	26	30	36		
2	1	100.00	10.00	2	7	13	19	22	24	25	26	37
3	1	100.00	3.00	4	5	11						
4	1	100.00	4.00	6	16	31	30					
5	1	100.00	5.00	0	17	32	40	41				
6	1	100.00	6.00	9	10	21	27	33	35			
7	1	100.00	6.00	10	12	14	29	34	39			

*** there are no common datablocks ***

Table 1. Output of the allocation algorithm, the variable *id* is the index of the program module or data block.

A sample output of the algorithm is given in Table 1. The output includes the *total (%) utilization* of each processor which is the processing time plus communication time of modules assigned to a processor.

When the time frame T changes, a different clustering of $G(A)$ is obtained. Ideally, we would like to find the shortest time frame T for which the application can be run. Let T_{PAR} = *the shortest time frame for which the machine can run the application A in parallel.*

From the output of the allocation algorithm, we can compute the average number of active processors, AP, (see Section 3.3) as follows

$$AP = \sum_{i=1}^{k} u_p^i$$

We can obtain T_{PAR} and T_{REAL} from the algorithm and then the following bounds hold

$$1 \leq S = T_{seq}/T_{REAL} \leq T_{seq}/T_{PAR} \leq AP.$$

4.2 The Algorithm Mapper System

The *algorithm mapper* is a software system which maps any application to a shared parallel memory architecture system. It is made up of a preprocessor, the heuristic allocation algorithm called ALLOC, and a user friendly graphical interface.

The allocation algorithm was initially implemented in Pascal [STEV 82] for a single time frame and a hypothetical system queueing delay function. All input data were assumed known. No preprocessor or user interface was available. The code was inefficient and did not completely solve the mapping problem.

The present allocation algorithm ALLOC is implemented in the language C with a variable time frame. A library of performance functions has been added of four multiprocessor architectures, namely (a) single bus with shared common memory, (b) single bus with distributed shared memory modules (two port memory), (c) multiple bus with shared memory modules, and (d) processors and shared memory with a Banyan interconnection. In the case of a multiple bus with shared memory module architecture, the number of busses can be set equal to one, thus obtaining a single bus with distributed shared memory architecture or set equal to the number k of processors, thus obtaining a crossbar interconnection of processors with distributed shared memory.

The time frame T is varied. Initially a large value of T is used, which decreases until T_{PAR} is obtained, i.e., the shortest time frame for which the number of clusters obtained equals the application parallelism in Step 1, or after the parallelism reduction Step 2, it equals the number of available processors in the system.

After an allocation has been obtained and the clusters formed, the execution of the application is simulated in order to obtain T_{REAL}, i.e., the actual real time required to execute the application architecture. This simulation routine is also in the C program.

4.2.1 The algorithm mapper preprocessor

The preprocessor PALLOC is an interface between the user and the allocation program ALLOC which automatically creates its input data file. The preprocessor is necessary because the amount of data needed by ALLOC is very large, especially for large graphs.

PALLOC is written in the C programming language and currently runs on a DEC VAX-11/780 computer under the Berkeley operating system (4.3 BSD). All applications programs for PALLOC are written in C. There is also a version of the preprocessor for applications written in FORTRAN.

PALLOC uses either an abstraction of the actual application or an instrumented version of it or a combination. In any case, the application must be partitioned into subroutines corresponding to the program modules to be used in the parallel implementation. Further, the data communicated between these modules must be explicitly specified as to destination and size (in bytes). The execution time of the code in a program module can either be specified explicitly or the actual code is compiled and timed during a sequential execution of the application. In summary, PALLOC determines the execution times and total communication of modules from this special version of the application. The details about the features of the input to PALLOC can be found in [HOUS 88b].

4.2.2 The algorithm mapper graphics user interface

AllocTool is a graphical interface to the allocation algorithm. It helps the user to specify the computation graph, to enter the required data for the algorithm, and to display the results in a graphical form. In general, for a specific application, the user has to do the following steps to use the allocation algorithm:

(a) Run *AllocTool* (which is trivial).

(b) Draw the application data flow graph.

(c) Specify the various data that are required for the algorithm.

(d) Run the allocation algorithm and display the results.

(e) (Optional) Store the application description in a file for later use.

Steps (b) and (c) can be replaced by loading an application data file previously prepared.

Figure 2. Sample output of the graphics user interface of AllocTool. Colors are used in the large CANVAS FRAME (center) to indicate the processor assignment (or numbers for black and white workstations). Processor utilization is shown at the upper right, the control *PANEL* at the upper left.

AllocTool uses the Sun View library routines and should work on any Sun workstation. In the following paragraphs, we use the terminology of Sun View library when referring to windows and specific items within these windows. These terms from Sun View are in *ITALICS*.

The tool is composed of two basic *FRAMES* (windows). The first one is the control *PANEL*, that controls the functions of the tool and the second is a frame containing a *CANVAS* window for images and a small *PANEL* on the top for diagnostic messages. In [HOUS 88b], we describe the operation of these windows.

4.2.3 Shared memory architecture models

Four different shared memory architectures (see Section 3.2) are analyzed and their queueing delay functions $D(u)$ presented. In all cases the overall system organization is the same and it is introduced first.

The systems are composed of k processors and k memory modules, (although we are assuming that the number of processors is the same as the number of memory modules, the same analysis can be applied when the number of processors is less than the number of memory modules). Each processor has its own private memory module, where the program and the data are stored. If processor i wants to communicate with processor j, it prepares a message and sends it to memory module j where it can be accessed by processor j.

The performance analysis of these architectures is documented in detail in [HOUS 87], [HOUS 88b]. The main performance measure obtained is the *average number of active processors, AP*. We have also obtained the *average queueing delay per message, $D(u)$*, and the *average utilization* of the communication network. We summarize these results below.

System 1: Single bus and shared common memory architecture.

Set $\rho = \lambda/\mu$, where λ is the message generation rate of a processor and $1/\mu$ is the average length both from an exponential distribution, then

$$AP_1 = \frac{\sum_{j=0}^{k} \frac{\rho^j k!}{(k-j)!} - 1}{\rho \sum_{j=0}^{k} \frac{\rho^j k!}{(k-j)!}},$$

and

$$u = \rho \times AP_1/2, \quad D_1(u) = (k - AP_1)/u.$$

System 2: Single bus and distributed shared memory architecture (double port memory modules).

Set

$$\rho = \frac{\lambda}{\lambda + 2\mu}$$

then we have

$$AP_2 = \frac{(1-\rho)\sum_{j=0}^{k} \frac{\rho^j k!}{(j-k)!}}{\rho \sum_{j=0}^{k} \frac{\rho^j k!}{(j-k)!}},$$

$$u = AP_2\rho/(1-\rho),$$

$$D_2(u) = (k - AP_2)/(1-\rho))/u.$$

System 3: *Multiple bus and distributed shared memory modules architecture.* It is of order $k \times m \times b$ where k is the number of processors, m the number of memories and b the number of busses.

Set $\rho = \lambda/\mu$ and define the array $p_j(l)$ by

$$p_j(l) = p_j(l-j) + p_{j-1}(l-j) + \cdots + p_1(l-j) + p_0(l-j)$$

with initial conditions

$$\begin{aligned} p_j(l) &= 0 \quad l < j \\ p_0(l) &= 0 \quad l > 0 \\ p_j(l) &= 1 \quad j \geq 0 \end{aligned}$$

Then set

$$\beta_l = \frac{\sum_{j=1}^{b-1} j p_j(l) + b \sum_{j=0}^{l-b} [p_b(j+b) p_{m-b}(l-2b-j+m)]}{\sum_{j=1}^{b-1} p_j(l) + \sum_{j=0}^{l-b} [p_b(j+b) p_{m-b}(l-2b-j+m)]}, \quad l \geq 0$$

and

$$P_k' = \left[1 + \sum_{j=0}^{k-1} \left[\rho^{k-j} \frac{k!}{j!} \prod_{l=1}^{k-j} \beta_l^{-1}\right]\right]^{-1}$$

$$P_k = \sum_{i=1}^{k} \rho^{k-i} \frac{k!}{i!} \prod_{j=0}^{k-i} \beta_j^{-1} P_k' \text{ for } i = 1 \text{ to } k-1.$$

We then have

$$AP_3 = \sum_{i=1}^{k} P_i$$

Further we have

$$u = 1 - P_0$$

$$D_3(u) = (k - u/\rho)/u.$$

System 4: Banyan switch and distributed shared memory architecture.

Set $\rho = \lambda/\mu$ and

$$P_0 = \left[\sum_{i=0}^{k} \frac{k!}{(k-i)!} \rho^i \prod_{j=1}^{i} \frac{1}{c(j)}\right]^{-1}, \quad P_i = P_0 \frac{k!}{(k-i)!} \rho^k \prod_{j-1}^{i} \frac{i}{c(j)},$$

where $c(j) = f_{\log_4}(i)$, $f_l(i) = \phi(f_{l-1}(i))$ and $\phi(i) = \begin{cases} \dfrac{i(2k - 0.5i - 1.5)}{2(k-1)} & i > 1 \\ 1 & i = 1. \end{cases}$

Then we have

$$AP_4 = \sum_{i=1}^{k} iP_i$$

With $u(x) = \max(x, 0)$, let

$$L = \sum_{i=1}^{k} u(i - c(i))P_i,$$

then we have

$$u = 1 - P_0$$

and

$$D_4(u) = (k + L\rho)/(k - L).$$

4.3 A Parallel Implementation of the Algorithm Mapper

The allocation algorithm runs initially for a large value of the *time frame* T and subsequently the value of T is decreased until the number of clusters equals the parallelism of the graph in Step 1, or equals the number of processors in Step 2 (see the discussion at the end of Section 1). The smallest value of T for which Step 1 or Step 2 is completed is T_{PAR}. Note that in each iteration of the algorithm, the allocation heuristic is executed producing a number of clusters and schedule of modules for the current value of T. When T changes, the output of the allocation algorithm changes as well. Moreover, the output of one iteration is not used in the next iteration. This attribute makes the algorithm a suitable candidate for parallel execution of

the multisection extension of the bisection algorithm for locating zeros of functions. See [MIRA 69], [RICE 71] for further details. We have implemented a parallel version using the Sequent, a parallel machine which houses (in the configuration we used at the Computer Science Department at Purdue) 20 processors.

Any number $k \leq 20$ processors can be used. Our particular version for the parallel algorithm mapper is as follows.

1. Estimate T as best one can.
2. Take as initial interval $[a,b] = [T/2,T]$ hoping that the guess is good enough for this interval to contain T_{PAR}. Set $T_0 = 0$.
3. Divide $[a,b]$ into $k-1$ equal subintervals by the values $T_i = a + (i-1)(b-a)/(k-1)$, $i = 1, 2, \ldots, k$.
4. Assign the ith processor the value T_i and run the allocation program with it.
5. Let $[T_j,T_{j+1}]$ be the interval such that (a) T_{j+1} gives the smallest number of clusters, and (b) the number of clusters for T_j is larger than that of T_{j+1}. If $j \neq 1$, take $[a,b]$ to be $[T_j,T_{j+1}]$. If $j = 1$, take $[a,b]$ to be $[\max(T_0,2T_1 - T_k),T_1]$. If $b - a \leq \epsilon$ = convergence tolerance, then go to 6, otherwise go to 3.
6. Set $T = a$ and produce output. If the number of clusters is less than the parallelism of the application, then find that interval $[T_l,T_{l+1}]$ where l is the smallest index where the number of clusters exceeds those of T. Set $T_0 = T_l$, $a = T_l$, $b = T_k$ and go to 3.

A little analysis shows that this algorithm has speed up of $\log_2(k+1)$ which is worthwhile for small values of k.

4.4 Time Complexity and Optimality of the Algorithm Mapper

The time complexity of the allocation algorithm is approximately proportional to the number of links in the application graph. This is a consequence of the search for the links that carry high communication and the formation of lists of such links until a merge can be performed. After each merge, the same search starts over again. In Table 2, timing data from three representative applications are shown. These data strongly suggest that the time complexity is no worse than linear in the number of links in $G(A)$.

Application	Number of links	Time	Ratio
Real time	27	1.18	22.9
PDE collocation	77	4.1	18.8
Robot arm	230	21.1	10.9

Table 2. Experimental data on the complexity of the allocation algorithm. Time for execution is given in seconds on a VAX 11/780 along with the ratio of links to time.

A symmetric and computationally simple partition of a Cholesky decomposition application is used to test the optimality of the algorithm mapper. A (4×4) partition of the application graph is discussed in detail in Section 5.1. The degree of parallelism of the graph is four and four clusters were obtained for System 2 with four processors. By using simulation, T_{PAR} has been shown to be the minimum possible elapsed time for this application (see Figure 4). Moreover, the processor utilizations were well balanced (see Table 3).

5. EXAMPLE STUDIES USING THE ALGORITHM MAPPER

Several applications have been tested to confirm the analysis and heuristics.

These applications are, (1) a Cholesky decomposition algorithm for the solution of linear equations, (2) a partial differential Equation (PDE) problem, using the collocation method, and (3) the solution to the Newton-Euler equations, for the mechanical movement of a robotic elbow manipulator. In all three cases, a partitioning of the application is given. The partitions are from previous work: PDE collocation application, [HOUS 87]; Cholesky decomposition, [OLEA 85]; robot elbow application, [KASH 85].

5.1 Cholesky Decomposition Application

We consider the parallelization of Cholesky algorithm for the factorization of symmetric matrices [OLEA 85]. We initially used a data flow language SIMON [FUJI 85] to specify the computational modules and their communication and synchronization requirements. We then executed these programs using the SIMON non-shared memory multiprocessor simulator. Figure 3 shows the 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 (i.e., assume there are no communication delays). 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. The *processing time* is the time to execute the code in a module. These times are given in the vectors b and m of Figure 3. The application was also

run by our preprocessor and similar input data for the allocation were obtained.

The matrix is partitioned into blocks so that the maximum assigned degree of parallelism is equal to the number of processors available, so we have as many processors as the assigned degree of parallelism of the application. A single bus multiprocessor and distributed shared memory architecture is used. Thus only Step 1 of the mapping problem needs to be taken. We have run this application for various assigned degrees of parallelism (number of processors), we only report on the case of assigned degree of parallelism four here. Table 3 shows the output of the allocation algorithm. Note that no internal datablocks are necessary in this application, since no reference to data is necessary. By default the number of internal datablocks are set equal to one per module of size zero.

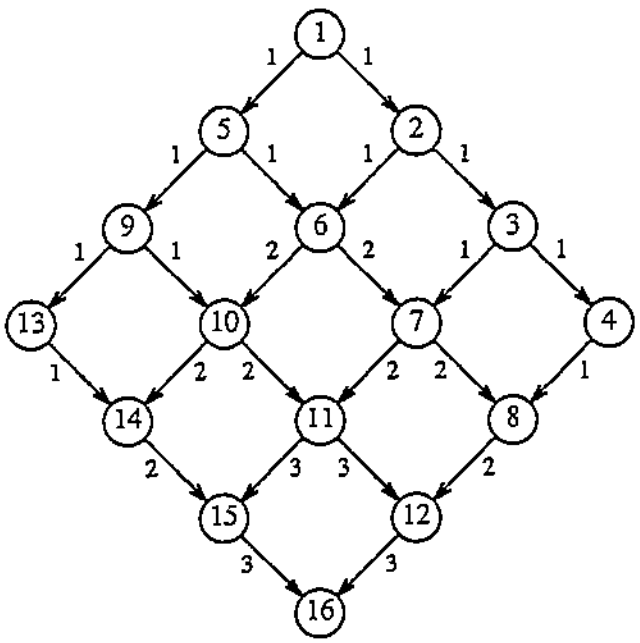

Figure 3. Procedence graph $G(A)$ of the parallel Cholesky decomposition algorithm for a 4 by 4 block decomposition of a symmetric matrix. The numbering of modules is specified in each node and the processing times and blocking times are given in this order. The communication traffic is indicated as a weight on the links of the graph. (Module processing times m = (24, 30, 30, 24, 27, 31, 32, 27, 34, 38, 39, 21, 28, 34, 45); Module blocking times b = (0, 4, 14, 24, 15, 27, 37, 48, 33, 45, 61, 66, 51, 64, 76, 88).

From Table 3 we see that $T_{PAR} = 250$, $T_{seq} = 622$ (139+152+170+161), and $T_{REAL} = 336$. The processor utilization u_p^i is (total cluster processing time)/(time frame T), for example $u_p^1 = 139/250 = 0.566$. The number AP of active processors is calculated from $\sum u_p^i$ to be 2.488. The speed up is $S = T_{seq}/T_{REAL} = 1.851$. We see that the following relationship holds.

$$1 \leq S = T_{seq}/T_{REAL} = 1.851 \leq T_{seq}/T_{PAR} = 2.488 \leq AP = 2.488$$

When the time frame T is decreased below T_{PAR}, the result is more and more clusters. We have used simulation to verify (see Figure 4) that the minimum elapsed time of this application does indeed occur at $T = T_{PAR}$. See [HOUS 87] for more details.

Table 3. Output of a 4 × 4 case of the Cholesky decomposition application of Figure 3.

Figure 4. Simulation results for the elapsed time and average transmission delay of the different $G'(A)$'s obtained for the Cholesky 4×4 decomposition application.

5.2 PDE Collocation Application

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. The complete graph as displayed by the algorithm mapper is shown in Figure 5 (the communication has been scaled by a factor of 1000). We again use a single bus multiprocessor and distributed shared memory architecture with the number of processors equal to the assigned degree of parallelism of the application. Thus only Step 1 of the mapping problem needs to be taken. In Figure 6 the output from the algorithm mapper of the allocation algorithm is shown. All modules having the same number are allocated to the same processor.

Table 4 shows another form of the output of the algorithm mapper system. No internal data blocks exist in this application, thus datablocks by default are set to equal one per module of size zero.

Figure 5. The complete graph of the PDE collocation application as displayed by the algorithm mapper system. This is at the input stage, the communication along edges is shown and the computational modules numbered.

Figure 6. The output of the algorithm mapper system for the PDE collocation application. The allocation of computational modules to processors is indicated by the numbering of the nodes. Nodes with the same number are allocated to the same processor.

Table 4. Additional output of the algorithm mapper system for the PDE collocation application.

Similar calculations can be made from Table 4 as with the previous application. Thus $T_{PAR} = 70$, $T_{seq} = 252.587$, $T_{REAL} = 71$ and $AP = .512 + .6968 + .2375 + .5 + 4513 + .554 + .6567 = 3.608$. The speed up is 3.54 and we have

$$1 \leq S = T_{seq}/T_{REAL} = 3.54 \leq T_{seq}/T_{PAR} = 3.6 \leq AP = 3.608$$

See [HOUS 87] for more details.

5.3 Robotic Elbow Manipulator Application

In [KASH 85] a partition of a robot elbow manipulator computation is given at the equation level, i.e., the computational modules represent the solution of an equation. We use this partition with slight modifications. Figure 7 shows the a precedence graph of this application; the numbers assigned to the modules identify them. Modules are organized on different levels and modules on the same level can be executed in parallel. The execution times of modules are given in [KASH 85] in *m*sec for an Intel 8087 processor. The partition in [KASH 85] requires little communication among modules and communication is not considered there. We have modified this by including in the execution time, t_e, of a module, both the processing time t_x and communication time t_y, that is $t_e = t_x + t_y$. We also assume that synchronization delay is included in t_x. If a module has several outgoing links, we distribute the communication times t_y uniformly among them. We have modified this application to obtain three applications with different behaviors. We take the execution time t_e to be constant and divide into communication and computation so that the values of the *computation/communication ratio* $r = t_x/t_y$ are 1/10, 1/1 and 10/1. The latter corresponds closely to the original application.

We vary the computation/communication ratio r of this application to study its effect and to show various properties of the allocation algorithm. Usually a fine partition requires more communication than a coarse partition, but at the same time, the number of modules is increased or decreased respectively. By varying r, we change the partition grain without affecting the number of modules. Let c_p be the number of clusters obtained by the algorithm in Step 1, then in general c_p is greater than or equal to the parallelism of the application.

For this application, a comparative study has been performed for two architectures, the multiple bus and Banyan switch, both with distributed shared memory.

Figure 7. The precedence graph of the 105 computational modules of the robot elbow manipulator. The maximum assigned degree of parallelism is 11.

Schedules for the Multiple Bus and Distributed Shared Memory Architecture

This application, shown in Figure 7, is allocated to a $k \times m \times b$ multiple bus and distributed shared memory (see Section 4.2.3) architecture. We choose the number k of memory modules (m) and processors (k) to be equal. A $13 \times 13 \times 1$ and a $13 \times 13 \times 8$ system are used. The number c_p of parallel clusters obtained is equal to the number of processors in the system, as indicated in Figure 10.

Sample schedules of the results are illustrated in Figures 8,9. We give a schedule of assigned modules to each processor, the processor utilization u_p^i ($i = 1,2,\ldots,13$), and the optimal T_{PAR} obtained. We also give for each processor the *total processor utilization* which includes both the processing and communication delay times required by all modules assigned to the same processor.

In Figure 10, we present a summary of the statistics of schedules produced by the algorithm mapper for the multiple bus system architecture. We have used u_p^i to be the *average total processor utilization*. In [HOUS 88b], details on every schedule as illustrated in Figures 8,9 are included.

Comparing the results in Figure 10, we observe for the case of $r = 1,\ 1/10$, that T_{PAR} is shorter for the 8 bus system than for the 1 bus system. This is expected, since the 8 bus system has higher bandwidth and thus less delay. In addition, in each case different processor schedules have been obtained. This is the effect that queueing delay has on the scheduling. In the case of $r = 10/1$, queueing delay does not play a significant role in scheduling and the 1 bus and 8 bus systems have identical schedules. We have measured the utilization for this case and we find that it is low. Thus, for low utilization values, there is no significant difference in the queueing delay for 1 and 8 bus systems. We note that processor utilizations are low due to the parallelism constraint between modules, we cannot assign more modules to increase their utilization.

In Figure 11, a schedule is shown which is based on minimizing the amount of communication among modules without assigning a cost to it, i.e., the queueing delay $D(u)$ is zero. A different schedule is produced as compared to the same $r = 1/10$ case for the 1 bus system in Figure 8. The significant queueing delay in the 1 bus system also increases T_{PAR} substantially. Thus queueing delay has a direct effect on scheduling.

system: $13 \times 13 \times 1$, $r = 1/10$, $T_{PAR} = 4054$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	10.88	3.05	1 27 28 61 66 67 72 92 98 104
2	18.94	4.26	2 7 24 36 37 42 48 49 54 78 87
3	14.82	4.62	3 9 58 64 70 81 85 96
4	15.86	2.06	4 10 16 46
5	15.86	2.06	5 11 17 47
6	15.86	2.17	6 12 18 60
7	16.29	2.15	8 14 15 29
8	10.16	2.24	13 25 43 57
9	18.93	5.67	19 20 21 22 23 26 31 32 34 35
10	18.93	5.40	30 39 45 51 55 75 79 83 89 94 100
11	9.80	5.76	38 44 50 73 74 82 88 93 99 102 105
12	19.29	7.80	40 41 52 53 76 77 80 84 90 91 95 100
13	15.31	5.25	56 59 62 65 68 71 86 97 103

Figure 8. Schedule of module assignments of the robot elbow manipulator for the 13×13×1 multiple bus architecture and low computation/communication ratio, $r = 1/10$.

system: $13 \times 13 \times 1$, $r = 10/1$, $T_{PAR} = 3304$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	12.97	12.38	1 27 30 61 67
2	42.08	40.16	2 6 12 18 37 49 60 87
3	89.98	88.03	3 9 39 41 45 51 53 75 79 83 89 91 94 100
4	26.67	25.31	4 10 16 46
5	26.67	25.31	5 11 17 47
6	67.46	65.20	7 8 14 15 24 29 36 42 48 54 78
7	25.25	24.76	13 43 57 63 69
8	70.90	69.60	19 20 21 22 23 26 31 32 33 34 35
9	34.31	33.29	25 28 55 66 72 92 98 104
10	71.10	70.70	38 44 50 73 74 82 88 93 99 102 105
11	74.85	74.00	40 52 76 77 80 84 90 95 101
12	65.36	64.37	56 59 62 65 68 71 86 97 103
13	51.73	51.17	58 64 70 81 85 96

Figure 9. Schedule of module assignments of the robot elbow manipulator for the 13×13×1 multiple bus system architecture and high computation/communication ratio, $r = 10/1$.

Multiple Bus Architecture (*kxmxb*)		
$r = 1$	13×13×1 $T_{PAR} = 2206$ $u_p^t = 54.41$ $u_p = 40.82$ $c_p = 13$	13×13×8 $T_{PAR} = 2125$ $u_p^t = 53.56$ $u_p = 42.38$ $c_p = 13$
$r = 1/10$	$T_{PAR} = 4054$ $u_p^t = 15.45$ $u_p = 4.03$ $c_p = 13$	$T_{PAR} = 3881$ $u_p^t = 15.10$ $u_p = 4.21$ $c_p = 13$
$r = 10/1$	$T_{PAR} = 3304$ $u_p^t = 50.71$ $u_p = 49.56$ $c_p = 13$	$T_{PAR} = 3304$ $u_p^t = 50.67$ $u_p = 49.56$ $c_p = 13$

Figure 10. Summary of performance statistics of 13 processor schedules for the multiple bus architecture (1 bus or 8 busses system).

system: none, $r = 1/10$, $T_{PAR} = 3880$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	8.78	2.34	1 13 25 43 57 63 69
2	16.38	4.45	2 7 24 36 37 42 48 49 54 78 87
3	13.12	4.82	3 9 58 64 70 81 85 96
4	13.37	2.15	4 10 16 46
5	13.37	2.15	5 11 17 47
6	13.39	2.27	6 12 18 60
7	13.73	2.25	8 14 15 29
8	16.69	5.92	19 20 21 22 23 26 31 32 33 34 35
9	11.24	6.04	27 28 61 66 67 72 73 92 93 98 99 100 101 102 103 104 105
10	25.97	6.20	30 39 45 51 55 75 79 82 83 89 94
11	15.76	3.11	38 44 50 74 88
12	20.24	8.05	40 41 52 53 76 77 80 84 90 91 95
13	12.55	5.06	56 59 62 65 68 71 86 97

Figure 11. Schedule of module assignments of the robot elbow manipulator. The queueing delay is set to zero and a low computation/communication ratio, $r = 1/10$ is used.

Schedules: Banyan Switch and Distributed Shared Memory Architecture

The results of applying the allocation algorithm to this application with the Banyan switch and distributed shared memory architecture are summarized In Figure 12. A sample output is shown in Figure 13. Twelve or thirteen parallel clusters are obtained. Our observation has been that for symmetric application graphs or almost symmetric graphs, the number of clusters

obtained is equal to the assigned degree of parallelism of the application. When the application graph does not possess any symmetry, like the one here, the number of clusters may be greater than the graph's assigned degree of parallelism. This is due to the allocation algorithm's mechanism of module merging which must satisfy the parallelism and time frame constraints at the same time. This is a limitation of our heuristic algorithm, since it does not exhaust all possible schedules.

Banyan switch, 8 processor system	
$r = 1$	$T_{PAR} = 2212$ $u_p' = 53.85$ $u_p = 38.27$ $c_p = 12$
$r = 1/10$	$T_{PAR} = 3383$ $u_p' = 14.98$ $u_p = 3.32$ $c_p = 13$
$r = 10/1$	$T_{PAR} = 3306$ $u_p' = 53.42$ $u_p = 44.53$ $c_p = 13$

Figure 12. Summary of performance statistics of cluster schedules for the Banyan system before parallelism reduction is applied.

system: 8 processor, Banyan switch, $r = 10/1$, $T_{PAR} = 3306$

Cluster id	Total (%) utilization	Utilization (%)	Modules assigned
1	12.95	12.37	1 27 30 61 67
2	42	40.14	2 6 12 18 37 49 60 87
3	89.88	87.99	3 9 39 41 45 51 53 75 79 83 89 91 94 100
4	26.62	25.30	4 10 16 46
5	26.61	25.30	5 11 17 47
6	67.36	65.17	7 8 14 15 24 29 36 42 48 54 78
7	25.23	24.75	13 43 57 63 69
8	70.83	69.57	19 20 21 22 23 26 31 32 33 34 35
9	34.27	33.27	25 28 55 66 72 92 98 104
10	71.05	70.67	38 44 50 73 74 82 88 93 99 102 105
11	74.79	73.96	40 52 76 77 80 84 90 95 101
12	65.30	64.34	56 59 62 65 68 71 86 97 103
13	51.69	51.14	58 64 70 81 85 96

Figure 13. Schedule of module assignments of the robot elbow manipulator for the Banyan switch architecture and high computation/communication ratio, $r = 10/1$.

5.4 Reduction of Parallelism in the Robot Application

Here we investigate the possibility of using the algorithm mapper system's allocation algorithm to reduce the parallelism of an application. This corresponds to Step 2 of the mapping problem as described in Section 1. We use the robot application and the Banyan switch architecture. The number of parallel clusters obtained in Figure 12 are twelve or thirteen, and we now assume that only an 8 processor system is available and thus we need to reduce the number of clusters to 8 from 12 or 13. To reduce parallelism we use the same heuristic allocation algorithm with a simple modification, we eliminate the parallelism constraint. We use as the input to the heuristic algorithm for Step 2 the clusters obtained in Step 1. Each of these clusters is regarded as a single module and the communication between clusters forms the communication between modules. Thus the graph output from Step 1 is the input to Step 2, except that the parallelism constraints are removed. The communication cost is found as in Step 1. Since parallelism between the modules of the new graph is not a constraint, it is always feasible to cluster the modules into a predetermined number of processors (in this case 8), by adjusting appropriately the time frame T parameter. The results for the three cases of r used in Step 1 are summarized in Figure 15. A sample output of the algorithm is shown in Figure 14. Note the module 1 of Figure 14 corresponds to cluster 1 of Figure 13 and so on for the rest of the modules. We also note that processor utilizations are much higher than in Step 1.

The time frame T_{PAR} may increase or decrease when the parallelism is reduced. If communication dominates the work, then T_{PAR} should be smaller, because with the parallelism reduction, less communication is required. One sees this to be the case when $r = 1/10$ (compare Figures 12 and 15). If computation dominates the work, then T_{PAR} should be larger because there are fewer processors to do the computation. One sees this to be the case when $r = 10/1$ (compare Figures 12 and 15). In an intermediate case where communication and computation are of similar amounts, then the effect on T_{PAR} is unclear. One sees for this particular application that T_{PAR} is increased slightly (compare Figure 12 and 15).

system: 8 processor, Banyan switch, $r = 10/1$, $T_{PAR} = 3706$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned		
1	74.25	73.10	1	8	
2	83.97	80.94	2	4	5
3	80.18	78.49	3		
4	89.75	87.81	6	9	
5	88.92	88.06	7	11	
6	63.38	63.04	10		
7	58.25	57.40	12		
8	46.11	45.62	13		

Figure 14. Parallelism reduction for the schedules shown in Figure 13 for the robot application and Banyan switch architecture. The number of clusters is reduced from 13 to 8.

Banyan switch, 8 processor system	
$r = 1$	$T_{PAR} = 2528$ $u_p^t = 61.49$ $u_p = 51.03$ $c_p = 8$
$r = 1/10$	$T_{PAR} = 1278$ $u_p^t = 65.84$ $u_p = 24.57$ $c_p = 8$
$r = 10/1$	$T_{PAR} = 3706$ $u_p^t = 73.09$ $u_p = 71.80$ $c_p = 8$

Figure 15. Summary of performance statistics of processor schedules for the Banyan switch after parallelism reduction is applied.

For comparison purposes, we also reduce the 13 cluster schedule obtained for the 13 processor multiple bus architectures to 8 clusters for an 8 processor system. The results are summarized in Figure 16. Detailed schedules are reported in [HOUS 88b].

5.5 Performance Evaluation of Applications/Architecture Pairs

We now compare the performance of the Banyan switch architecture and multiple bus architecture for the robot elbow manipulator application. Three versions of the application are considered with computation/communication ratio values of $r = 1/1$, $1/10$ and $10/1$. Both

architectures have distributed shared memory and 8 processors, the multiple bus architectures also have 1 or 8 busses. The primary comparison is on the basis of T_{PAR}, the shortest parallel execution time. We also show (1) the average total processor utilization u_p^t which, includes both processor time and queueing waits for communication, (2) the speed up, and (3) the *efficiency* [SIEG 82] = (speed up)/k.

Figure 16 shows that the smallest T_{PAR} value and the highest processor utilizations were obtained for $r = 1/10$, as one expects. The $8 \times 8 \times 8$ multiple bus architecture has a higher bandwidth than the Banyan network, and the schedule for an $8 \times 8 \times 8$ multiple bus architecture has the best performance. This demonstrates the usefulness of the mapping methodology in matching applications to architectures. Speed ups and efficiency factors for the 8 processor systems are presented in Figure 17.

Ratio r	Banyan Switch Architecture	Multiple Bus Architecture
$r = 1$	$T_{PAR} = 2529$ $u_p^t = 61.50$	(1 bus) $T_{PAR} = 2421$ $u_p^t = 73.68$ (8 busses) $T_{PAR} = 2421$ $u_p^t = 73.70$
$r = 1/10$	$T_{PAR} = 1278$ $u_p^t = 65.84$	(1 bus) $T_{PAR} = 1207$ $u_p^t = 67.05$ (8 busses) $T_{PAR} = 1134$ $u_p^t = 66.75$
$r = 10/1$	$T_{PAR} = 3706.3$ $u_p^t = 73.10$	(1 bus) $T_{PAR} = 3804$ $u_p^t = 71.22$ (8 busses) $T_{PAR} = 3804$ $u_p^t = 71.22$

Figure 16. Performance comparison of multibus and Banyan architectures for the robot elbow manipulator application. Three values of the computation/communication ratio r are used. All architectures have 8 processors.

Ratio r	Speed up			Efficiency		
	Banyan	Multiple bus		Banyan	Multiple bus	
$r = 1/1$	4.630	(1 bus)	4.836	.578	(1 bus)	.604
		(8 busses)	4.836		(8 busses)	.604
$r = 1/10$	1.665	(1 bus)	1.763	.208	(1 bus)	.220
		(8 busses)	1.877		(8 busses)	.234
$r = 10/1$	5.744	(1 bus)	5.596	.718	(1 bus)	.699
		(8 busses)	5.596		(8 busses)	.699

Figure 17. Comparison of speed ups and efficiency for the 8 processor Banyan switch and multiple bus architectures.

From data in [KASH 85] the elapsed (sequential) time in a uniprocessor system, T_{seq}, can be calculated. In a uniprocessor system the communication cost is zero, so the sequential time depends heavily on the ratio r of computation to communication. Then for each value of r, we have considered that we get T_{seq} as shown in Figure 18.

$r = 1$	$T_{seq} = 11709$
$r = 1/10$	$T_{seq} = 2128$
$r = 10/1$	$T_{seq} = 21281$

Figure 18. Sequential elapsed time of the application for values of r.

In Figures 19 and 20 speed ups and efficiencies are given for the multiple bus architectures under the assumption that the number of processors equals the number of parallel clusters, i.e., after Step 1 of the algorithm is performed. This data is derived directly from the schedules in Section 5.3. Note that for $r = 1/10$ and the one or eight bus system, the speed up is actually less than 1, which indicates that the parallel system does worse than a single processor system. Thus, a partition where the communication is ten times the computation, is the worst partition of the three we have examined. This shows how our methodology helps us evaluate the various partitions of an application.

Ratio r	Speed up	Efficiency
$r = 1$	5.308	.408
$r = \frac{1}{10}$	.525	.040
$r = \frac{10}{1}$	6.441	.495

Figure 19. Speed up and efficiency data for the schedules of Figure 10 for the 13×13×1 multiple bus and distributed shared memory architecture.

Ratio r	Speed up	Efficiency
$r = 1$	5.510	.423
$r = \frac{1}{10}$	.548	.042
$r = \frac{10}{1}$	6.443	.495

Figure 20. Speed up and efficiency data for the schedules of Figure 10 for the 13 ×13×8 multiple bus and distributed shared memory architecture.

We illustrate the performance analysis further for those schedules (see Figure 16) where the assigned degree of parallelism has beed reduced. The speedup S and average number of active processors AP, are shown in Figures 21 and 22. Note that the bounds discussed in Sections 3.2 and 4.1.1 hold. In the cases where communication is limited, i.e., $r = 10/1$ and $r = 1/1$, its cost is also negligible, thus $S = AP$. When $r = 1/10$, then the communication cost in terms of queueing delay is not negligible any more, and it affects the T_{PAR} results. Thus $S = T_{seq}/T_{PAR}$ is lower than in the previous two cases. By definition, communication delay is not included in AP, thus $AP \geq S$ as stated previously and observed in both Figures 21 and 22. We may also calculate using T_{REAL}. For example for the schedule of Figure 9, we obtain AP = 6.4428, $T_{seq} = 21281$, $T_{PAR} = 3304$, $T_{REAL} = 6113$ and the following relationships.

$$1 \leq S = T_{seq}/T_{REAL} = 3.481 \leq T_{seq}/T_{PAR} = 6.440 \leq AP = 6.4428$$

Ratio r	$8 \times 8 \times 1$ system		$8 \times 8 \times 8$ system	
	AP	T_{seq}/T_{PAR}	AP	T_{seq}/T_{PAR}
$r = 10/1$	5.5969	5.5968	5.5969	5.596
$r = 1/1$	4.8351	4.835	4.8351	4.836
$r = 1/10$	2.4144	1.763	2.3133	1.877

Figure 21. Speed up and active processors for the schedules of Figure 16 for the multiple bus and distributed shared memory architecture. The assigned degree of parallelism has been reduced from 13 to 8 in these schedules.

Ratio r	AP	T_{seq}/T_{PAR}
$r = 10/1$	5.7446	5.744
$r = 1/1$	4.6310	4.630
$r = 1/10$	1.9658	1.665

Figure 22. Speed up and active processors for the schedules of Figure 15 for Banyan switch and distributed shared memory architecture. The assigned degree of parallelism has been reduced from 13 to 8 in the schedules.

6. SUMMARY

We have formulated the mapping problem and described our algorithm mapper system and methodology. Four architectures have been considered and three realistic applications have been evaluated for them. These applications were Cholesky decomposition algorithm, a PDE collocation solution and a robot arm manipulation computation. The four architectures considered are: (1) single bus and shared common memory, (2) single bus and distributed shared memory, (3) multiple bus and distributed shared memory, and (4) Banyan switch and distributed shared memory. The allocation methodology has made use of the parallel architecture performance models to assign a cost to communication between parallel processors. This cost is the queueing delay in communicating messages between modules assigned to different processors. The allocation algorithm used is based on a merging heuristic that minimizes communication between processors in assigning parallel modules to different processors. The approach has also been used to evaluate various partitions of a single application.

We have evaluated the performance of application/architecture pairs. Different partitions of the same application may "fit" better on different architectures. Our methodology provides the means for this evaluation. We have made extensive experimentation with the robot arm application with 105 modules. We see a dramatic improvement in execution speed up using even a suboptimal allocation method such as ours. In some small systems and applications, we are able to demonstrate optimality of results, but we do not expect this in general. The parallel architectures we have studied have shared memory and our current approach depends on this fact. We are pursuing a similar approach for nonshared memory architectures.

7. REFERENCES

[ABRA 86] Abraham, S.G. and Davidson, E.S., "Task Assignment Using Network Flow Methods for Minimizing Communication in *n*-Processor Systems", Technical Report, CSRD Rpt. No. 598, Center of Supercomputing Research and Development, National Center of Supercomputing Applications, University of Illinois at Urbana-Champaign, 1986.

[ALLE 78] Allen, A.O., *Probability Statistics and Queueing Theory*, Academic Press, 1978.

[BATC 76] Batcher, K.E., "The Flip Network in STARAN", *Int'l Conf. on Parallel Processing*, 1976, pp. 65–71.

[BATR 78] D.P. Batra, *Architectural Implications of Problem Partitioning for Distributed Processor Systems*, Ph.D. Thesis, Computer Science Department, Northwestern University, Evanston, Illinois, 1978.

[BUKL 79] Bukles, B.P and Hardin, D.M., "Partitioning and Allocation of Logical Resources in a Distributed Computing Environment", Tutorial: *Distributed System Design*, IEEE Computer Society, 1979.

[CHU 80] Chu, W.W., Holloway, L.J., Lan, M.T. and Efe, K., "Task Allocation in Distributed Data Processing", *Computer*, 1980, pp. 57–69.

[CHU 87] Chu, W.W., Lance M-T, Lan, "Task Allocation and Precedence Relations for Distributed Real-Time Systems", *IEEE Trans. Computer Engineering*, Vol. C, 1987, pp. 667–679.

[EFE 82] Efe, K., "Heuristic Models of Task Assignment Scheduling in Distributed Systems", *Computer*, Vol. 15, 1982, pp. 50–56.

[FUJI 85] Fujimoto, R.M., "The SIMON Simulation and Development System", *Summer Computer Simulation Conference*, 1985.

[GANN 86] Gannon D. and Von Rosendale, J., "On the Communication Complexity of Parallel Numerical Algorithms", *IEEE Trans. Computers*, (to appear).

[GILB 87] Gilbert, J.R. and Zmijewski, E., "A Parallel Graph Partitioning Algorithm for a Message-Passing Multiprocessor", Technical Report, TR 87-803, Department of Computer Science, Cornell University, Ithaca, N.Y., 1987.

[GYLE 76] Gylys, V.B. and Edwards, J.A., "Optimal Partitioning of Workload for Distributed Systems", *Proceeding Compcon*, 1976, pp. 353–357.

[GOTT 83] Gottlieb, A., et al., "The NYU Ultracomputer-Designing on MIMD Shared Memory Parallel Computer", *IEEE Trans. Computers*, C-33, 1984, pp. 1180–1194.

[HAES 80] Haessig K. and Jenny, C.J., "Partitioning and Allocation Computational Objects in Distributed Computing Systems", *Proc. of IFIP Congress*, 1980, pp. 593–598.

[HOUS 81] Houstis, C.E., Houstis E.N. and Rice, J.R. "Partitioning and Allocation of PDE Computation to Distributed Systems", in: B. Engquist and T. Smedsass, eds. *PDE Software: Modules Interfaces and Systems*, North-Holland, Amsterdam, 1981, pp. 67–85.

[HOUS 82] Houstis, C.E., "Software Partitioning in a Distributed Environment", Technical Report, College of Engineering, University of South Carolina, 1982.

[HOUS 84] Houstis, E.N, Rice, J.R. and Vavalis E.A., "Spline Collocation Methods for Elliptic Partial Differential Equations", in: R. Vichnevetsky and R.S. Stepleman, eds. *Advances in Computer Methods for Partial Differential Equations V*, IMACS, Rutgers University, 1984, pp. 191–194.

[HOUS 87a] Houstis, C.E., Houstis, E.N. and Rice, J.R., "Partitioning PDE Computations: Methods and Performance Evaluation", *J. Parallel Computing*, Vol. 5, 1987, pp. 141–163.

[HOUS 87b] Houstis, C.E., "Allocation of Real-Time Applications to Distributed Systems", *Int'l Conf. Parallel Processing*, 1987, pp. 863–866.

[HOUS 87c] Houstis, C.E., "Distributed Processing Performance Evaluation", *Third International Conference on Data Communication Systems and Their Performance*, L.F.M. de Moaves, E. de Souse e Silva and L.F.G. Soaves, eds., Rio de Janeiro, Brazil, 1987, pp. 391–406.

[HOUS 87d] Houstis, C.E. and Aboelaze, M., "The Mapping of Applications to Multiple Bus and Banyan Interconnected Multiprocessor Systems: A Case Study",

Supercomputing, 1987, pp. 514–543.

[HOUS 88a] Houstis, C.E., "Allocation of Real Time Applications to Distributed Systems", Under review in the IEEE Trans. on Software Engineering.

[HOUS 88b] Houstis, C.E., Houstis, E.N., Rice, J.R., Samartzis, S.M. and Alexandrakis, D.L., "The Algorithm Mapper: A System for Modeling and Evaluating Parallel Applications/Architecture Pairs", Technical Report CSD-TR-793, Computer Science Department, Purdue University, West Lafayette, IN 47907, August 1988.

[HWAN 84] Hwang, K. and Briggs, F., *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, 1984.

[GYLY 76] Gylys, V.B. and Edwards, J.A., "Optimal Partitioning of Workload for Distributed Systems", *Proceeding Compcon*, 1976, pp. 353–357.

[JENN 77] Jenny, C.J., "Process Partitioning on Distributed Systems", Digest of Papers, NTC 1977, pp. 31:1–31-10.

[JENN 82] Jenny, C.J., "On the Placement of Files and Processes in a System With Distributed Intelligence", *Proceedings of International Zurich Seminar on Digital Communications*, 1982, pp. B1.1-8.

[KASH 85] Kashara, H. and Narita, S., "Parallel Processing of Robot-Arm Control Computation on a Multiprocessor System", *IEEE J. Robotics Automation*, Vol. RA-1, 1985, pp. 104–113.

[KLEI 85] Kleinrock, L., "Distributed Systems", *Comm. ACM*, Vol. 28, 1985, pp. 1200–1213.

[KRUS 83] Kruskal, C. and Snir, M., "The Performance of Multistage Interconnection Nets for Multiprocessing", *IEEE Trans. Computers*, Vol. C-32, 1983, pp. 1091–198.

[LAWR 75] Lawrie, D., "Access and Alignment of Data in an Array Processor", *IEEE Trans. Computers*, Vol. C-24, 1975, pp. 1145–1155.

[LI 81] Li, H., "The Impact of Process Intercommunication on the Global Bus Architecture", *IEEE Proc. Real-Time Systems*, 1981, pp. 29–31.

[LOW 73] Lowe, T.C., "Analysis of an Information System Model With Transfer Penalties", *IEEE Trans. Computers*, Vol. C-22, 1973, pp. 269–280.

[MA 81] Ma, P., Lee, E.Y.S. and Tsuchiya, M., "On the Design of a Task Allocation Scheme for Time-Critical Applications", *IEEE Proc. Real-Time Systems*, 1981, pp. 121–126.

[MARS 83] Marsan, M.A., Balbo, G. and Conte, G., "Comparative Performance Analysis of Single Bus Multiprocessor Architectures", *IEEE Trans. Computers*, Vol. C-31, 1983, pp. 1179–191.

[MARS 83] Marsan, M.A. and Gerla, M., "Markov Models for Multiple Bus Multiprocessor Systems", *IEEE Trans. Computers*, Vol. C-32, 1983, pp. 239–248.

[MIRA 69] Miranker, W.L., "Parallel Methods for Approximating the Root of a Function", *IBM J. Res. Develop.*, Vol. 13, 1969, pp. 297–301.

[NORT 85] Norton, A. and Pfister, G.F., "A Methodology for Predicting Multiprocessor Performance", *Proc. Int'l Conf. Parallel Processing*, 1985, pp. 772–778.

[OLEA 85] O'Leary, D.P. and Stewart, G.W., "Data-Flow Algorithms for Parallel Matrix Computations", *Comm. ACM*, 28, 1985, pp. 840–853.

[OLEA 87] O'Leary, D.P. and Stewart, G.W., "Assignment and Scheduling in Parallel Matrix Factorization", *Linear Algebra Appl.*, 77, 1986, pp. 275–300.

[PATE 79] Patel, J.H., "Processors-Memory Interconnections for Multiprocessors", *Proc. 6th Ann. Symp. Computer Arch.*, 1979, pp. 168–177.

[RICE 71] Rice, J.R., "Matrix Representations of Nonlinear Equation Iterations – Application to Parallel Computation", *Math. Comp.*, Vol. 25, 1971, pp. 639–647.

[SARK 86] Sarkov, V. and Hennessey, J., "Compile-Time Partitioning and Scheduling of Parallel Programs", *Proceedings of the SIGPLAN 1986 Symposium on Compiler Instructions*, ACM, 1986, pp. 17–36.

[SIEG 78] Siegel, H.J. and Smith, H.D., "Study of Multistage SIMD Interconnection Networks", *5th Annual Symposium on Computer Architecture*, 1978, pp. 223–229.

[SIEG 78] Siegel, H.J., McMillen, R.J. and Mueller, P.T., Jr., "A Survey of Interconnection Methods for Reconfigurable Parallel Processing Systems", *Int'l Conf. on Parallel Processing*, 1978, pp. 9–17.

[SIEG 82] Siegel, L., Siegel, H.J. and Swain, P.H., "Performance Measures for Evaluating Algorithms for SIMD Machines", *IEEE Trans. Software Engineering*, Vol., SE-8, 1984, pp. 319–331.

[SIEG 85] Siegel, H., *Interconnection Networks for Large-Scale Parallel Processing*, Heath and Company, 1985.

[STEV 82] Stevens, R.M., *A Pascal Program Which Partitions Programs for a Multiprocessor System*, Masters Thesis, Electrical and Computer Engineering, University of South Carolina, 1982.

[STON 77] Stone, H.S., "Multiprocessor Scheduling With the Aid of Network Flow Algorithms", *IEEE Trans. Software Engineering*, Vol. SE-3, 1977, pp. 85–93.

[STON 78] Stone, H.S. and Bokhari, S.H., "Control of Distributed Processes", *Computer*, Vol. 11, 1978, pp. 971–976.

[WILL 83] William, E.A., "Assigning Processors to Processors in Distributed Systems", *IEEE Conf. Parallel Processing*, 1983, pp. 404–406.