THE MAPPING OF APPLICATIONS TO MULTIPLE BUS AND BANYAN INTERCONNECTED MULTIPROCESSOR SYSTEMS: A CASE STUDY

Catherine E. Houstis *
Mokhtar Aboelaze
Electrical Engineering Department
Purdue University

Abstract

We study the mapping of a robot elbow manipulator application, to two different classes of multiprocessor systems the multiple bus and Banyan interconnected systems. A comparative performance analysis of the two systems is performed. The application is partitioned into communicating computational modules and three different partitions of it are approximated. Fast heuristic algorithms are used to produce assignments of modules to processors. A number of performance measures are also employed to evaluate the matching of application/architecture pairs.

1. INTRODUCTION

In previous work [HOUS83], [HOUS87a], we have outlined a mapping methodology which has been applied to the computations involved in the solution of partial differential equations. We continue this study here for a different application, a robot arm elbow manipulator and two different system architectures, multiple bus and banyan interconnected parallel multiprocessor systems.

The mapping problem arises when the number of computational modules required by the application exceeds the number of processors available or when the interconnection structure of the application's computational modules, differs from the interconnection structure of the parallel machine [BERM84]. In [BOKH81], the mapping problem is defined as the assignment of modules to processors and the problem of maximizing the number of pairs of communicating modules that fall on pairs of directly connected processors. It is also shown, that the mapping problem is equivalent to a graph isomorphism problem, or to a bandwidth reduction problem. In either case, an exact algorithm for the mapping problem is unlikely to be found. Research in this area has concentrated on efficient heuristics which give good solutions in most cases, [BERM85], [BOKH81], [JENN77], [ABRA86], [GILB87].

The application we consider here, is the solution of the Newton Euler equation for the motion of a six degree of freedom manipulator whose joints are all rotational, i.e., an elbow manipulator. The computations involved are modelled by a precedence graph, where each node in the graph represents a computational module's computation and memory requirements and each link represents communication requirements between modules. The decomposition of the applications computation is given in [KASA85], and it is broken down in such a way that parallelism is used to maximum advantage.

*This research was supported by NSF grant DMC-8508684A1.

The parallel multiprocessor system architectures, are composed of processors each having a local memory and shared memory modules and they are interconnected via multiple busses or a Banyan switch. The systems are homogeneous, i.e., they have identical processors and identical local or shared memories. The processors are assumed uniprogrammed. Communication is performed via message passing between the processors by using common access to shared memories and the interconnection network. The communication network of the multiprocessor system is represented by its performance characteristic which is the communication Queueing Delay vs its Utilization. This requires its performance analysis. We have used one of the multiples bus interconnection models presented in [MARS83], and performed a similar performance analysis. We have also performed a comparable analysis for the Banyan interconnection network.

The solution of the mapping problem, involves a number of steps which can be performed somewhat independently [HOUS87a]. (a) Schedule the computational modules into parallel clusters, (b) reduce the number of parallel clusters to the number of parallel processors in the machine then (c) imbed the application into the machine. We concentrate on a heuristic algorithm, that is used in step (a) and also in step (b) with a simple modification. Step (c) for the interconnection architectures used and homogeneous systems is trivial since there is a communication link between all processors, or processors can be considered equally distant. An arbitrary assignment of clusters obtained in step (b) to processors is sufficient. We have discussed this step (c) in [HOUS87c]. Thus we shall concentrate on steps (a), and (b).

In [HOUS83], [HOUS87a] only step (a) of this approach has been demonstrated for applications whose precedence graphs had no more than 50 communicating modules and a bus interconnected system architecture. In this work, a much larger graph is considered that has 105 nodes and two different system architectures are tested. The architectures are chosen so that the parallelism reduction, (step (b)), can be evaluated. We consider on one hand, a multiple bus architecture where the number of processors can be increased (or decreased) by one. In this case, the application's parallelism can be fully exploited since the number of processors can always be adjusted to equal the number of parallel clusters obtained in step (a). Then step (a) is a sufficient solution to the mapping problem. If the system has a fixed number of processors which are less than the clusters obtained, then the parallelism reduction step (b) is necessary. In a Banyan interconnected system, the number of processors can be increased (or decreased) only in powers of 2. Thus, if the number of clusters obtained is not a power of 2 then it is cost effective to use a system where the number of processors is the highest power of 2 and less than the number of clusters. In this case, the parallelism reduction, step (b), is unavoidable.

In a number of heuristic algorithms dealing with step (a) [CHU80] [EFE82], [JENN77], [STO78], [GYL76], [BERM87] an assignment of computational modules to processors is produced by minimizing the communication required among processors. We assign a cost to this communication which is the interconnection system's Queueing Delay. We claim that a different schedule is produced when the system's Queueing Delay is involved and that it is a more realistic schedule. We demonstrate this in Section 4.2. The use of Queueing Delay models simplifies the complexity of the mapping problem. A number of performance measures are also calculated which indicate how good *applications/architecture pairs* are *matched.*

In Section 2, the performance analysis of two interconnection networks is given. In Section 3, the application is presented and how different partitions of the application are

obtained. In Section 4, step (a) of the mapping problem is summarized and the results obtained using the two systems and the considered application are given. In Section 5, the parallelism reduction, step (b), is performed and the resulting schedules are presented. A comparison of the performance of the two systems is also demonstrated. In Section 6, conclusions about this methodology, application and systems are discussed.

2. COMPARATIVE PERFORMANCE ANALYSIS OF INTERCONNECTION NETWORKS

In steps (a) and (b) of the mapping problem, the objective is to exploit the parallelism of the multiprocessor system and that of the application in order to obtain optimal speed ups for the application. For this, two conditions must be met. (1) The application's computations must be decomposed into smaller sub-computations that can run in parallel, and at the same time the amount of data transfer between these sub-computations must be kept at a minimum. We further discuss this problem in the next section. (2) The system overhead must also be kept at a minimum. Our work has concentrated on the second condition. The system overhead comes mainly from the communication conflicts of the processors at the different memory modules or at the interconnection network.

In the systems we consider, processors have approximately comparable capabilities. All processors share access to a common memory module, I/O channels, and peripheral devices. Most importantly, the entire system is controlled by a single integrated operating system providing interactions between processors and their programs at various levels. Besides the shared memories and I/O devices, each processor has its own local memory. Interprocessor communications can be done via the shared memories. In this mode of operation, each processor executes a program stored in its local memory, on a set of data stored in the same local memory. When processor i wants to communicate with another processor j, either requesting a set of data or providing to the other processor with some needed data, processor i forms a message and sends it to the common memory module of processor j, via the interconnection network.

There are many physical configurations for the interconnection network. The simplest form is the single bus, where a single bus is used by all the processors to access any memory module. The single bus is very inexpensive, but the bandwidth of such a network is usually low, since only one processor can use the bus at any time, and thus it is inadequate even for a small number of processors. At the other end of the spectrum is the crossbar switch. In a crossbar interconnection, any processor can access any memory module, given that no other processor is accessing the same memory module. Although the crossbar interconnection offers the highest possible bandwidth, it is very expensive, and considering todays technology, it is the most expensive part in the system. Moreover, it will be very hard to justify its use especially for large systems.

In this paper, we study two classes of interconnection networks. The multiple bus network and the multistage banyan network. Notice that the single bus and the crossbar network are both a special case of the multiple bus network.

2.1 Banyan network

The banyan networks have been proposed mainly for SIMD "Single Instruction Multiple Data" machines, where more than one processor executes the same instruction on a different set of data. However, they have been used successfully in MIMD "Multiple Instructions Multiple Data" machines. They can be divided into two main categories, Single stage and Multistage networks.

Single stage, also called recirculating networks, because data may have to circulate through more than one processor to reach their final destination. Fig. 1, shows the different configurations for the single stage cube network.

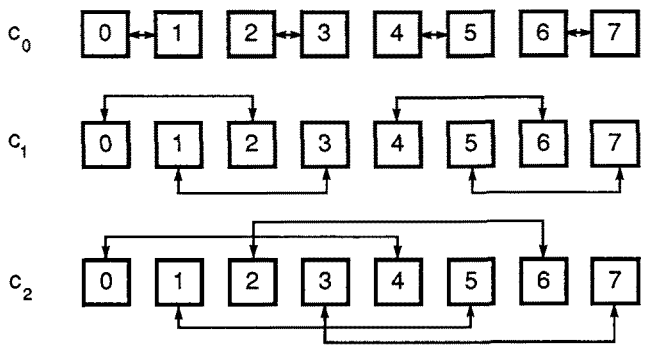

Fig. 1. Single Stage Cube Network.

Multi stage networks, connect k processors, via log$k$ stages and allow communication between any two processors to take place in log$k$ steps. The multistage networks can be characterized by three parameters, [SIEG85], interchange box, topology and control structure.

Interchange box is a two input two output switch, used as the basic building block for the multistage interconnection network. Fig. 2, shows an interchange box with two inputs marked a_1 and a_2 and two outputs marked b_1 and b_2. There are four configurations for the interchange box (1) straight, where $a_1 \rightarrow b_1$, $a_2 \rightarrow b_2$; (2) exchange, where $a_1 \rightarrow b_2$, $a_2 \rightarrow b_1$; (3) upper broadcast, where $a_1 \rightarrow b_1$, $a_2 \rightarrow b_1$; and (4) lower broadcast, where $a_1 \rightarrow b_2$, $a_2 \rightarrow b_2$. A two function interchange box can only take the configuration of straight and exchange, while a four function interchange box can take any one of the four configurations [SIEG78].

Topology is the actual connection between the different interchange boxes in the different stages. One topology is the indirect binary k-cube, shown in Fig. 3, for k=8. The input to the first stage is numbered 0 through 7. Any interchange box in stage i is connected to two inputs that differ in the i^{th} bit of their binary representation. Straight configuration for an interchange box will connect two numbers having the same i^{th} bit, while exchange configuration will connect two numbers that are different in their i^{th} bit. The indirect

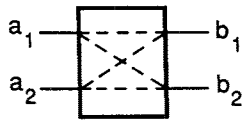

Fig. 2. Interchange box.

binary k cube network is used in the staran network [BATC76]. Another network is the Omega network, [LAWR75], shown in Fig. 4. The omega network is based on the perfect shuffle interconnection network, which routes data from position i whose binary representation is $i_{k-1} \cdots i_1 i_0$ to position $s(i)$ whose binary representation is $i_{k-2} \cdots i_1 i_0 i_{k-1}$.

Control structure. The control structure of the network can be either individual stage control or individual box control. In the individual stage control , the same control signal is used to set all the boxes in the stage, so all the boxes in one stage should have the same configuration. It is clear that the individual box control is more efficient, however, it will require every message to have a header to determine its destination. The individual boxes should have control circuitry to interpret the destination header and set its own configuration. In the rest of this paper, when we mention a banyan network we mean a multistage network with individual box control.

2.2 Modes of operation

We first introduce the organization of the system. The system is 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 the 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 the memory module number j where this memory can be accessed by processor j. We are assuming an asynchronous communication, so any processor can be in any one of three states.

1) The processor is executing a program in its local memory;
2) The processor is sending a message to another processor;
3) The processor is blocked waiting for the interconnection network to deliver a message to another processor.

Processors in the first stage are considered active processors, i.e., processors doing useful work not blocked or communicating with other processors. We do not account for the time taken by any processor to read a message sent by another processor to its memory module, since this is considered to be part of the program executed by the processor.

We are assuming that the time between the generation of messages is exponentially distributed random variable with mean $\frac{1}{\lambda}$, and the length of the message is an exponentially distributed random variable with mean $\frac{1}{\mu}$. We are also assuming that an access request from processor i is directed to memory module j with probability $p_{ij}=1/m$ where m is the

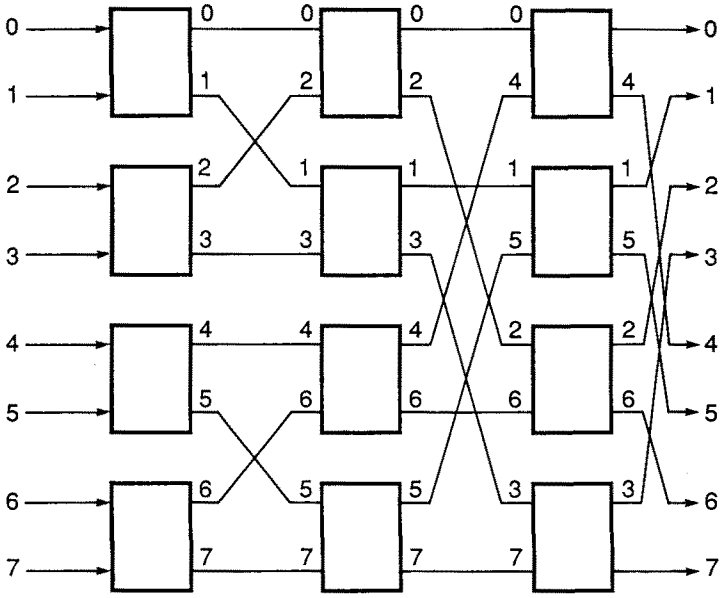

Fig. 3. Binary k-cube network (k=8).

number of memory modules (m=k). If any processor needs to send a message to another processor, it stops executing its program. Then, if the interconnection network can establish a path from the source processor to the destination module it will do so instantaneously with no delay and the sending processor will begin to send its message. When the message is completed the sending processor will return to its active state. If there is contention at the interconnection network or destination module the processor is put in queue, "blocked", waiting for the contention to be resolved and then transmits its message.

2.3 Performance Analysis of a Banyan Network

There have been some attempts to analyze the banyan network, [PATE79] [KRUS83]. However, in their analysis they considered the system to be synchronized. At the beginning of each cycle, every processor generates a message with a certain probability and they calculated the probability of message acceptance. In this paper, we assume the system to be asynchronous. The length of the message generated by any processor is an exponentially distributed random variable, and the time interval between the generation of two consecutive messages by the same processor is also an exponentially distributed random variable. We shall calculate the expected number of active processors, and the average delay encountered by an average message.

We are representing the system as an **M/M/c/K/K** queueing system "Machine repair model with K machines and c repairmen", [ALLE78]. This model, assumes a population of k identical devices (processors), each of which has an operating time of O time units between

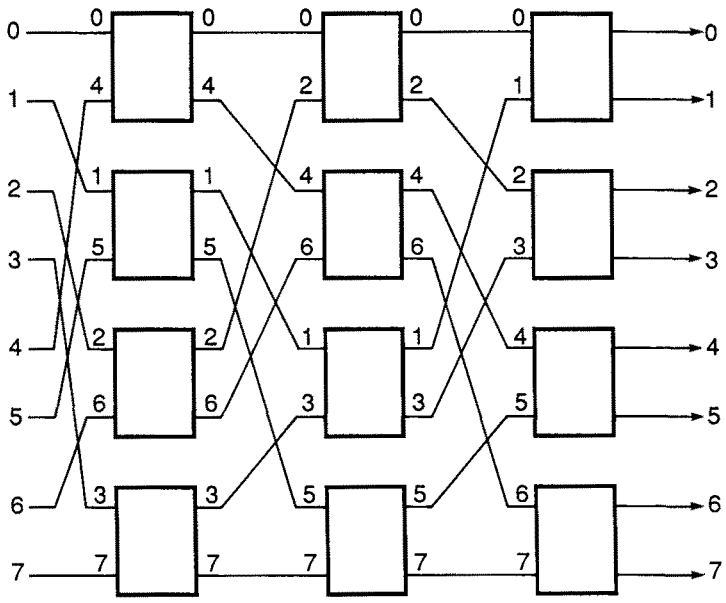

Fig. 4. Omega network.

breakdowns, O having a exponential distribution with average value of $\frac{1}{\lambda}$. The repairman (communicatioin network) repairs the machines with an average repair time of $\frac{1}{\mu}$ time units. Fig. 5, shows the state transition diagram for the **M/M/c/K/K** queueing system with a service rate c_i at state i. Notice that the service rate will depend upon how many customers (processors) are requesting service.

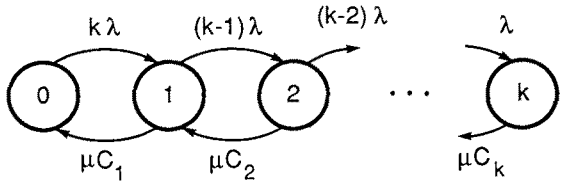

Fig. 5. State transition diagram for the **M/M/c/K/K** queueing system.

To calculate c_i, we have to know how many requests will go through the network if i processors request to send messages at the same time. Consider the first stage of the banyan network as shown in Fig. 3, or 4, where there are $\frac{k}{2}$ switches, each switch connected to 2 inputs. If one input is active, i.e., it is requesting to send a message and the other input is

idle, then the average throughput of this switch is 1. If two inputs are active at the same time, then the average throughput of this switch is 1.5 [PATE79]. Thus, the expected number of messages to go through one stage of the banyan network given that there are i requests equals the expected number of switches with one active input $\times$ 1 + the expected number of switches with two active inputs $\times$ 1.5. The probability that any switch chosen at random has one active input is p_1, where

$$p_1 = \frac{2i(k-i)}{k(k-1))} .$$

The probability that any switch chosen at random has two active inputs is p_2, where

$$p_2 = \frac{i(i-1)}{k(k-1)} .$$

Since the number of switches in one stage is $k/2$, then the expected number of messages to go through one stage given i requests is $\phi(i)$, where

$$\phi(i) = (p_1+1.5p_2)\frac{k}{2} \qquad i \geq 1$$

$$\phi(1) = 1$$

In a banyan network with k inputs we have log$k$ stages. We can calculate the average number of messages to go through the network in a recursive manner. If $c(i)$ is the *average* number of messages to go through the network given that there are i requests, then let $f_\ell(i)$ = expected number of messages that will go through the ℓ-th stage given that the input in the first stage will be i. Then,

$$f_\ell(i) = \phi(f_{\ell-1}(i))$$

$$\text{with } f_1(i) = \phi(i) = (p_1+1.5p_2)\frac{k}{2} = \frac{i(2k-0.5i-1.5)}{2(k-1)}$$

$$\text{and } c(i) = f_{\log k}(i)$$

After obtaining the service rate for the different states of the transition diagram in Fig. 5, the probability of being at state i is, P_i, where

$$P_i = P_0 \frac{k!}{(k-i)!} \left(\frac{\lambda}{\mu}\right)^k \prod_{j=1}^{i} \frac{1}{c(j)}$$

with P_0

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

The expected number of processors waiting in a queue is L_q, where

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

and

$$u(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

The actual request rate is $\bar{\lambda}$, where

$$\overline{\lambda} = \frac{k}{\frac{1}{\lambda}+W_q+\frac{1}{\mu}}$$

where W_q is the average time a processor spends in the queue. After applying Little's formula, we obtain

$$W_q = \frac{L_q}{\overline{\lambda}} = (\frac{1}{\lambda}+W_q+\frac{1}{\mu})\frac{L_q}{k}$$

solving for W_q, we obtain

$$W_q = \frac{L_q(\frac{1}{\lambda}+\frac{1}{\mu})}{k-L_q}$$

By normalizing the time with respect to $\frac{1}{\mu}$ and noticing that the total delay is the sum of the waiting time and the service time, we obtain

$$D = \frac{k+L_q\rho}{k-L_q}, \quad where \quad \rho=\frac{\lambda}{\mu}.$$

D is the *average delay per message*. The *interconnection network utilization*, U is

$$U = 1 - P_0$$

and the number of Active Processors (AP) is

$$AP = \sum_{j=1}^{k} jP_j$$

2.4 Performance analysis of a multiple bus system

In a multiple bus system, we assume k processors and m memory modules connected through a b bus interconnection network, as shown in Fig. 6. When processor i needs to access memory module j, processor i will check to see if there is an available bus. If one is found and memory module j is free (no other processor is accessing it), then a path from processor i to memory module j is established immediately with zero delay. If there is no available bus or there is another processor accessing memory module j, then processor i is blocked in a queue waiting for a bus, or for memory module j, or for both. Marsan and Gerla, [MARS83], analyzed the Markov chain of such a system. They concluded that the exact Markov chain is not easy to handle, because the number of states will increase very rapidly with the system size. They introduced four approximations with moderate computational complexity. In this work, we use an approximation which is very similar to approximation C2 in their work, which gives a lower bound for the average number of active processors. We analyze the system using this approximate model, and we use P_i, where P_i is the probability of having i processors requesting access to the memory (either accessing a memory module or waiting in a queue). Using the machine repairman model with k servers, [ALLE78], the expected number of processors waiting in a queue is l_q, where

$$l_q = \sum_{i=0}^{k} u(i-c(i))P_i.$$

P_i is given in [MARS83] and is

$$P_i = \left[\frac{\lambda}{\mu}\right]^{k-i} \frac{k!}{i!} \prod_{j=0}^{k-i} \beta_j^{-1} P_k$$

and

$$P_k = \left[1 - \sum_{j=1}^{k-1} \left[(\frac{\lambda}{\mu})^{k-j} \frac{k!}{j!} \prod_{i=0}^{k-j} \beta_i^{-1}\right]\right]^{-1}$$

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

where

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

with initial conditions

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

In Fig. 7, 8 the performance of the two systems is plotted for 8 processor systems.

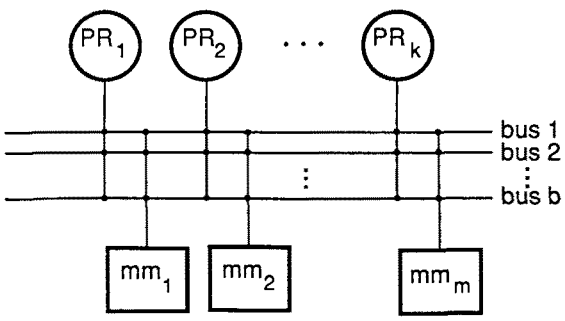

Fig. 6. A *kxmxb* multiple bus system.

3. A ROBOT ELBOW MANIPULATOR APPLICATION

The representation of an applications computations by a precedence graph requires a proper partitioning of the application into computational modules. Partitioning techniques via a system's compiler are presented in [SARK86]. There are applications which are amenable to mathematical decomposition techniques [OLEAR85], [BOKH81], [MARI87]. In the applications we have studied, [HOUS87a], knowledge of the application allow the use of mathematical decomposition to partition it and identify potential parallelism among modules. An integration of compiler techniques and decomposition techniques are needed for an appropriate partition.

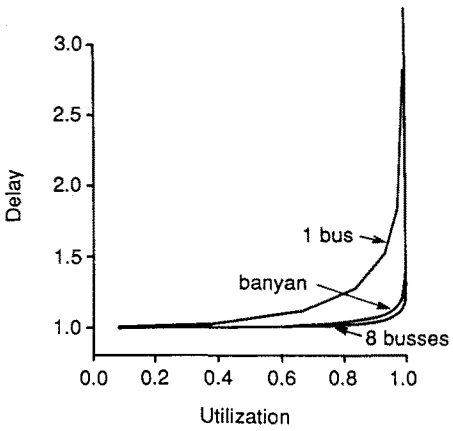

Fig. 7. Queueing delay vs. Utilization of interconnection networks.

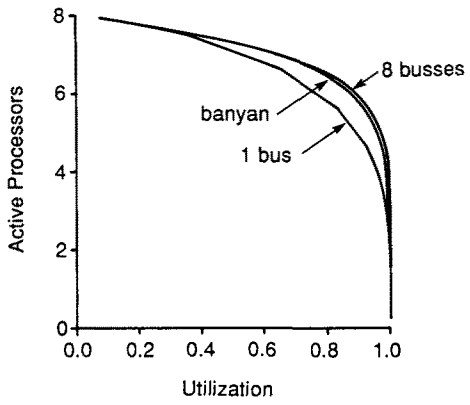

Fig. 8. Active Processors vs. Utilization of interconnection nets.

Once a partition is decided, the execution times of modules and communication times among modules have to be determined. In [HOUS87a] a parallel simulation language, SIMON, [FUJI85], was used to obtain these times along with the synchronization delay of each module. Since synchronization delay is an attribute of the application, it has been included in the graph by adding it to the processing time of its corresponding module. If communication paths among modules are not a priori known, then the partition is modeled by a stochastic logic graph [HOU87b]. A stochastic analysis of this graph results in a precedence graph as described above.

In [KASA85] a partitioning of the robot elbow manipulator computations is given at the equation level, i.e., computational modules represent the solution of an equation. We use this partition with slight modifications. In Fig. 9, a precedence graph of the application is shown. The numbers assigned to the modules are to simply identify them. Modules are grouped on different levels and parallel modules are shown by being drawn at the same level. The execution times of modules are given on [KASA85] in μsec for an Intel 8087 processor. The partition given in [KASA85] is such that very little communication is required among modules and it is considered negligible. We have modified this by assuming that the execution time, t_e, of a module, includes both processor's processing time and communication time. If t_x (μsec) is the processing time and t_y (μsec) is the communication time of a module, then $t_x+t_y=t_e$. We assume that synchronization delay is included in t_x. We assign t_y according to a uniform distribution to the outgoing links of its corresponding module. Thus, we have been able to investigate the effect of varying the ratio, r, between the amount of processing and communication of a module, where $r = t_x/t_y$. Three values of r have been used $r = 1, .1, 10$. We note at this point that by using r in this way, we only approximate what actually happens to a partition when the communication increases and processing decreases or vice versa. Usually, more communication is present when a partition is finer, i.e., module size (processing time) gets smaller but at the same time the number of modules is increased. By changing r, we

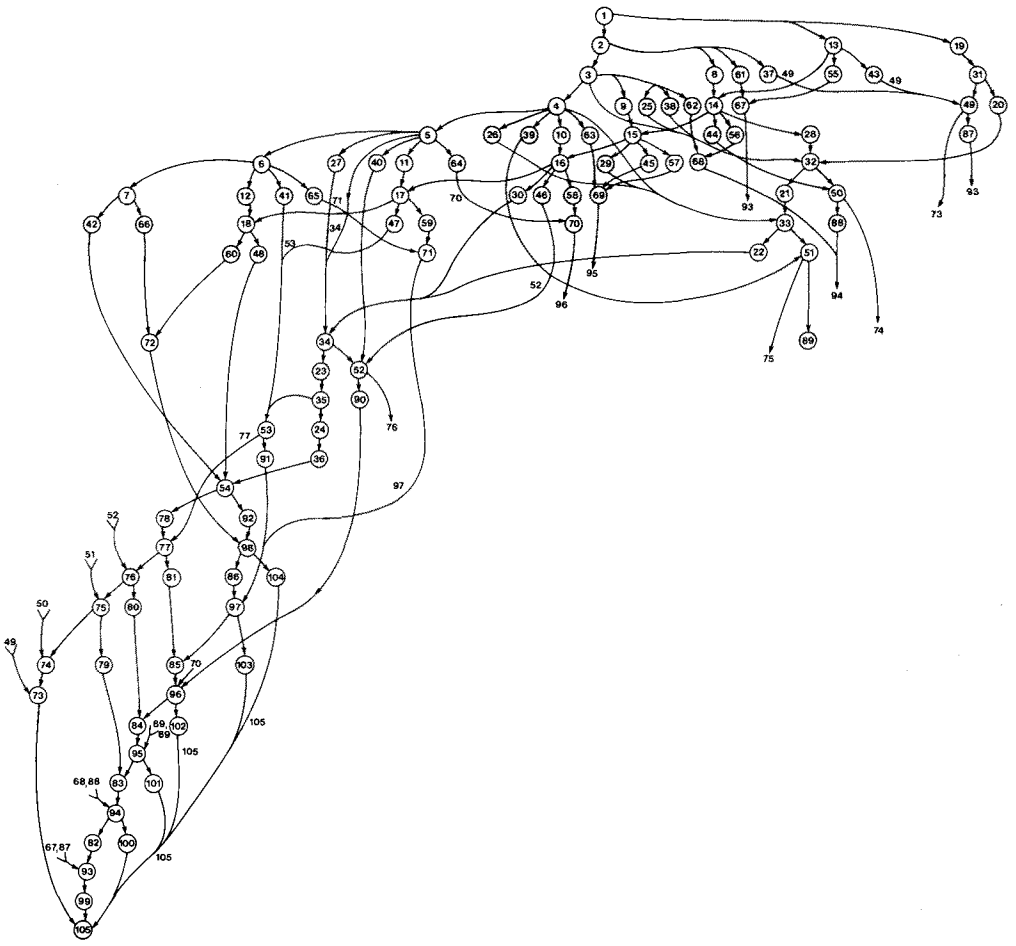

Fig. 9. The precedence graph of the robot elbow manipulator.

change the partition grain but without increasing or decreasing the number of modules.

From the communication requirements between a pair of modules, c_{ij}, we calculate the corresponding utilization of the interconnection network U, where

$$U = \frac{c_{ij}}{C \times T} .$$

where C is the capacity of the interconnection network and T is the time frame during which each parallel processor is running. This time frame will be described in the following section

and it represents a time constraint within which the application is required to complete execution.

4. SCHEDULING TO MINIMIZE COMMUNICATION DELAY

The first step of the mapping problem deals with the scheduling of modules into parallel clusters. A heuristic algorithm is used which is based on the minimization of communication delay between modules. The algorithm's input is the information represented in the application's precedence graph, the multiprocessor system specifications (processor speed, memory availability, bandwidth) and the Queueing Delay vs Utilization characteristic of the interconnection network. A detailed presentation of the algorithm's parameters and heuristic technique used is presented in [HOUS87b]. Here we summarize for completion purposes. The merit of this heuristic is that it is fast, that is, its computational complexity increases only *linearly* with the number of links in the application. The scheduling problem is a constraint minimization problem as follows:

(i) **resource constraints:**

— Every computational module must fit into the memory assigned.

— Computations must have enough processor time.

(ii) **parallelism constraint:**

— parallel modules cannot be assigned to the same processor.

(iii) **artificial constraint**

— Processing time on each processor is limited to T (a parameter).

objective function:

— minimize the queueing delay of the multiprocessor system interconnection network, due to the application's communication requirements.

The *time frame T* is used to calculate the utilization of the interconnection network according to equation (1), for every c_{ij}. The corresponding queueing delay is then found from Fig. 7 depending on which interconnection network architecture is used. T is also used to calculate the processor's utilization every time a pair of candidate modules is merged, (see next section), and check if constraint (i) is satisfied. After the module assignment to processors is completed, the final processor utilizations are calculated using the final value of T.

4.1 The heuristic algorithm

For a specified *time frame* value of T

Start

Assign one processor per module.

Iteration

(a) make up a list of eligible pairs of modules which are ordered according to longest amount of required communication and thus queueing delay among them.

(b) If no constraints are violated merge the pair of modules which will result in the maximum reduction of the objective function. If there is more than one pair then merge the

pair that results in the least processor loading; if there is still more than one, select a pair at random.

If we define by n_p the potential number of parallel modules in the application graph, and by c_p the number of parallel clusters obtained by the algorithm, then $c_p \geq n_p$. The initial value of T is large since the assignments of one module per processor represents the worst possible queueing delay in the systems. The value of T is decreased until $T = T_{PAR}$ where T_{PAR} is the shortest time for which the allocated processors can run the application in parallel. It turns out that T_{PAR} is a close *upper bound* of the elapsed time of the application when synchronization delay for each module is incorporated as part of the processing time of modules [HOUS87a]. T_{PAR} is the *optimal time frame*. Further reduction of T results in more and more processors to be used.

4.2 Results

The robot elbow manipulator application, shown in Fig. 9, was allocated to a $k \times m \times b$ parallel multiple bus system architecture, where k is the number of processors, m is the number of common memory modules ($m=k$), and b is the number of common busses. In our application $n_p = 11$ and $c_p = 12$ or 13. A 13x13x1 and a 13x13x8 systems have been used. An 8 processor Banyan interconnected system was also used. In the case of a multiple bus architecture the number of parallel clusters obtained is equal to the number of processors in the system. Outputs of the heuristic algorithm for different values of the ratio r, (r = processing time/communication time per module), and the two multiple bus systems are shown in Figures 10a,b,c and Figures 11a,b,c. A schedule of assigned modules to each processor, its utilization and the optimal the frames T_{PAR} obtained are given. We are also showing the total processor utilization which includes the processing and communication delay required by all modules assigned cluster to a processor. A number of observations can be made by comparing Fig. 10 and Fig. 11.

First, we observe that optimal time frames are shorter in the case of an 8 bus system as compared to the 1 bus system, for $r = 1, 1/10$. This is expected, since the 8 bus system has higher bandwidth and thus less delay and there is enough communication between modules to affect the scheduling decision. Note that in both cases, different processor schedules have been obtained. In the case $r = 10/1$, communication delay does not play a significant role in scheduling (only total utilizations are slightly different when we compare the output of the 8 busses system, to the output of the 1 bus system). As a result identical schedules have also been obtained. From Fig. 7, one can see that unless communication between modules is such that it results in over 50% utilization of the interconnection network there is no significant difference in queueing delay between the 1 bus and the 8 busses system. We note that processor utilizations are low. This is due to the parallelism constraint of modules that limits their scheduling possibilities.

In Fig. 12, a schedule is shown which is based on minimizing the communication among modules without assigning a cost to it, namely, its corresponding queueing delay (no system is assumed). Note that the schedule produced is different as compared to the same case schedule in Fig. 10b (1 bus system). Thus, involving queueing delay in the scheduling decision instead of simply the amount of communication among modules produces different results. If we compare Fig. 12 to its corresponding case in Fig. 11b (8 busses system), then one observes that the same schedule has been obtained. This is due to the fact that a 13x13x8 system is practically a crossbar switch unless the communication between parallel

modules is very high and thus resulting in interconnection utilization values greater than 80%, which is not the case in this application. One should not forget that if there are modules with high communication among them they are scheduled into the same processor unless they are parallel modules. This eliminates most of the high communication between modules.

system: 13×13×1, $r = 1/1$, $T_{PAR} = 2206$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	51.13	45.33	1 57 61 63 66 67 69 72 92 98 104
2	70.31	68.67	2 37 49 59 65 71 73 86 87 93 97 99 100 101 102 103 105
3	50.50	36.49	3 7 9 24 36 42 48 54 78
4	35.34	20.85	4 10 16 46
5	35.34	20.85	5 11 17 47
6	36.35	21.98	6 12 18 60
7	36.60	21.76	8 14 15 29
8	51.65	38.30	13 41 43 53 56 62 68 91
9	71.26	57.34	19 20 21 22 23 26 31 32 33 34 35
10	83.36	64.59	25 30 40 52 76 77 80 84 90 95
11	51.77	32.41	27 38 44 50 74 88
12	58.79	46.69	28 55 58 64 70 81 85 96
13	75.02	55.52	39 45 51 75 79 82 83 89 94

Fig. 10a. Schedule of module assignments of the robot elbow manipulator.

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

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 95 101
13	15.31	5.25	56 59 62 65 68 71 86 97 103

Fig. 10b. Schedule of module assignments of the robot elbow manipulator.

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

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

Fig. 10c. Schedule of module assignments of the robot elbow manipulator.

system: 13×13×8, $r = 1/1$, $T_{PAR} = 2125$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	58.87	58.35	1 61 66 67 72 73 87 92 93 98 99 100 101 102 103 104 105
2	44.27	30.12	2 27 37 41 49 53 91
3	81.15	62.35	3 9 39 45 51 75 79 82 83 89 94
4	32.92	21.65	4 10 16 46
5	32.92	21.65	5 11 17 47
6	34.00	22.82	6 12 18 60
7	74.58	55.76	7 8 14 15 24 29 36 42 48 54 78
8	61.06	52.24	13 43 56 59 62 65 68 71 86 97
9	70.36	59.53	19 20 21 22 23 26 31 32 33 34 35
10	27.30	22.12	25 57 63 69
11	57.89	48.47	28 55 58 64 70 81 85 96
12	76.96	64.71	30 40 52 76 77 80 84 90 95
13	44.00	31.29	38 44 50 74 88

Fig. 11a. Schedule of module assignments of the robot elbow manipulator.

system: 13×13×8, $r = 1/10$, $T_{PAR} = 3881$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	8.86	2.34	1 13 25 43 57 63 69
2	16.53	4.45	2 7 24 36 37 42 48 49 54 78 87
3	13.22	4.82	3 9 58 64 70 81 85 96
4	13.51	2.15	4 10 16 46
5	13.51	2.15	5 11 17 47
6	13.53	2.27	6 12 18 60
7	13.88	2.25	8 14 15 29
8	16.83	5.93	19 20 21 22 23 26 31 32 33 34 35
9	11.30	6.04	27 28 61 66 67 72 73 92 93 98 99 100 101 102 103 104 105
10	26.23	6.21	30 39 45 51 55 75 79 82 83 89 94
11	15.92	3.12	38 44 50 74 88
12	20.40	8.06	40 41 52 53 76 77 80 84 90 91 95
13	12.65	5.06	56 59 62 65 68 71 86 97

Fig. 11b. Schedule of module assignments of the robot elbow.

system: 13×13×8, $r = 10/1$, $T_{PAR} = 3304$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	12.96	12.38	1 27 30 61 67
2	42.02	40.16	2 6 12 18 37 49 60 87
3	89.93	88.03	3 9 39 41 45 51 53 75 79 83 89 91 94 100
4	26.63	25.31	4 10 16 46
5	26.63	25.31	5 11 17 47
6	67.40	65.20	7 8 14 15 24 29 36 42 48 54 78
7	25.24	24.76	13 43 57 63 69
8	70.87	69.60	19 20 21 22 23 26 31 32 33 34 35
9	34.28	33.29	25 28 55 66 72 92 98 104
10	71.08	70.70	38 44 50 73 74 82 88 93 102 105
11	74.82	74.00	40 52 76 77 80 84 90 95 101
12	65.33	64.37	56 59 62 65 68 71 86 97 103
13	51.72	51.17	58 64 70 81 85 96

Fig. 11c. Schedule of module assignments of the robot elbow manipulator.

system: no system, $r = 1/10$, $T_{PAR} = 3880$

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

Fig. 12. Schedule of module assignments of the robot elbow manipulator.

From the available data, one can calculate performance measures such as speed up and efficiency [SIEG82], where

$$\text{speed up} = \frac{\text{elapsed time in uniprocessor system}}{\text{elapsed time in multiprocessor system}}$$

and if there are k processors in the multiprocessor system then

$$\text{efficiency} = \frac{\text{speed up}}{k}$$

On tables 1,2 speedups and efficiency measures are given for the different system architectures.

		Speed up	Efficiency
13×13×1 multiple bus system architecture	$r = 1$	5.308	.408
	$r = \frac{1}{10}$	.525	.040
	$r = \frac{10}{1}$	6.441	.495

Table 1: Speed up and efficiency data for the schedules of Fig. 10a,b,c.

		Speed up	Efficiency
13×13×8 multiple bus system architecture	$r = 1$	5.510	.423
	$r = \frac{1}{10}$	.548	.042
	$r = \frac{10}{1}$	6.443	.495

Table 2: Speed up and efficiency data for the schedule of Fig. 11a,b,c.

Step (a) was also applied when the parallel system has a Banyan interconnection network. The results are shown in Fig. 13a,b,c. 12 or 13 parallel clusters have been obtained. The number of clusters depends primarily on the parallelism of the application, (n_p), and the system involved in the scheduling decision. Our observation has been that for symmetric graphs or almost symmetric graphs, [HOUS87a], the number of clusters obtained is equal to the of parallelism of the graph. When the application graph does not possess any symmetry, like the one we used here, the numbers of clusters is greater or equal to the graph's parallelism. This is due to the mechanism of module merging which satisfies the parallelism and time frame constraint. It can be regarded as a limitation of our heuristic algorithm since it does not exhaust all possible schedules (NP complete).

system: 8 processor, Banyan Interconnection, $r = 1/1$, $T_{PAR} = 2212$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	49.54	45.20	1 57 61 63 66 67 69 72 92 98 104
2	74.22	72.99	2 37 49 59 65 71 73 82 86 87 93 97 99 102 103 105
3	40.67	26.44	3 6 9 12 18 60
4	31.62	20.79	4 10 16 46
5	31.62	20.79	5 11 17 47
6	71.65	53.56	7 8 14 15 24 29 36 42 48 54 78
7	28.82	22.60	13 30 43 56 62 68
8	67.58	57.18	19 20 21 22 23 26 31 32 33 34 35
9	60.56	52.20	25 39 45 51 75 79 83 89 94 100
10	46.79	32.32	27 38 44 50 74
11	55.60	46.55	28 55 58 64 70 81 85 96
12	87.66	78.64	40 41 52 53 76 77 80 84 90 91 95 101

Fig. 13a. Schedule of module assignments of the robot elbow manipulator.

system: 8 processor, Banyan Interconnection, $r = 1/10$, $T_{PAR} = 3883$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	8.79	2.34	1 13 25 43 57 63 69
2	16.39	4.45	2 7 24 36 37 42 48 49 54 78 87
3	13.38	2.15	4 10 16 46
4	13.38	2.15	4 10 16 46
5	13.38	2.15	5 11 17 47
6	13.40	2.27	6 12 18 60
7	13.75	2.25	8 14 15 29
8	16.71	5.92	19 20 21 22 23 26 31 32 33 34 35
9	11.24	6.20	30 39 45 51 55 75 79 82 83 89 94
10	26.00	6.20	30 39 45 51 55 75 79 82 83 89 94
11	15.77	3.11	38 44 50 74 88
12	20.26	8.05	40 41 52 53 76 77 80 90 91 95
13	12.56	5.06	56 59 62 65 68 71 86 97

Fig. 13b. Schedule of module assignments of the robot elbow manipulator.

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

Processor id	Total (%) utilization	Utilization (%)	Modules assigned
1	12.95	12.37	1 27 30 61 67
2	42.00	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

Fig. 13c. Schedule of module assignments of the robot elbow manipulator.

5. REDUCTION OF PARALLELISM

In [HOUS87c] a number of heuristic algorithms have been discussed for the reduction of parallelism, step (b). Here we apply one of these heuristics and perform step (b).

Parallelism reduction in necessary when the number of parallel clusters obtained in step (a) is higher than the parallel processors in the system. In general, the number of clusters depends to a great extend on the parallelism of the application, and it is unlikely that it is

equal to the number of available processors. There are multiprocessor system architectures where the number of processors can easily be adjusted to equal the number of clusters obtained. Multiple bus interconnection architectures present such feasibility as discussed in the previous section. In such case step (a) is sufficient.

In this work we are interested to test the performance of our method for a Banyan interconnected architecture. Since the number of parallel clusters obtain in Fig. 13a,b,c are 12 or 13 it is cost effective to assume that an 8 processor system is available and thus we need to reduce the number of clusters from 12 or 13, to 8. In the reduction of parallelism, step (b), we use the same heuristic algorithm as in step (a) with a simple modification, which is to eliminate the parallelism constraint. This is feasible since the input to the heuristic algorithm for the reduction step is the number of parallel clusters obtained in step (a). Each of these clusters is regarded as a single module and the communication between clusters forms the communication between modules. Thus, we start with a new graph which is the output of step (a). The communication cost is found as in step (a) by using the Queueing Delay vs Utilization of an 8 processor Banyan interconnected network as shown in Fig. 7. Since parallelism between the modules of the new graph is eliminated, 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 constraint. The processor utilizations in step (a) are low and because of that, in order to get 8 clusters, we have to increase or decrease T until the desired number of clusters are obtained. The results in every case corresponding to step (a) are presented in Fig. 14a,b,c. Note that module 1 of Fig. 14 corresponds to cluster 1 of Fig. 13 and so on for the rest of the modules. We also note, that processor utilizations are much higher than in step (a) which results in a lower time frame in every case.

system: 8 processors, Banyan Interconnection, $r = 1/1$, $T_{PAR} = 2528$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned		
1	43.34	39.54	1		
2	64.94	63.86	2		
3	82.49	59.51	3	4	5
4	62.68	46.86	6		
5	75.82	65.44	7	9	
6	59.12	50.02	8		
7	40.93	28.27	10		
8	62.66	54.76	12		

Fig. 14a. Parallelism reduction for the schedules shown in Fig. 13a.

system: 8 processor, Banyan Interconnection, $r = 1/10$, $T_{PAR} = 1083$

Processor id	Total (%) utilization	Utilization (%)		Modules assigned	
1	85.16	29.63	1	8	
2	81.49	48.77	2	9	11
3	89.81	25.01	3	4	
4	74.58	15.86	5	6	
5	50.97	8.06	7		
6	86.10	22.24	10		
7	74.40	28.88	12		
8	46.12	18.13	13		

Fig. 14b. Reduction parallelism for the schedules shown in Fig. 13b.

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

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		

Fig. 14c. Reduction parallelism for the schedules shown in Fig. 13c.

system: 8×8×1, $r = 1$, $T_{PAR} = 2421$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned		
1	82.02	78.66	1	13	
2	89.42	72.26	2	8	
3	71.30	54.71	3		
4	79.46	58.01	4	5	6
5	65.54	48.93	7		
6	61.79	52.23	9		
7	89.07	76.18	10	12	
8	50.84	42.53	11		

Fig. 15a. Reduction of parallelism for the schedule shown in Fig. 10a.

system: 8×8×1, $r = 1/10$, $T_{PAR} = 1207$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned		
1	89.60	33.44	1	12	
2	56.68	14.31	2		
3	84.05	22.44	3	4	
4	70.06	14.23	5	6	
5	48.02	7.23	7		
6	57.31	19.05	8		
7	87.81	49.40	9	10	11
8	42.88	16.27	13		

Fig. 15b. Reduction of parallelism for the schedules shown in Fig. 10b.

system: 8×8×1, $r = 1/10$, $T_{PAR} = 3804$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned		
1	72.34	71.22	1	8	
2	81.81	78.86	2	4	5
3	78.12	76.47	3		
4	87.44	85.56	6	9	
5	86.64	85.79	7	11	
6	61.75	61.42	10		
7	56.76	55.92	12		
8	44.93	44.45	13		

Fig. 15c. Reduction of parallelism for schedules shown in Fig. 10c.

system: 8×8×8, $r = 1$, $T_{PAR} = 2421$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned		
1	82.00	78.66	1	13	
2	89.32	72.26	2	8	
3	71.20	54.71	3		
4	79.34	58.01	4	5	6
5	65.44	48.93	7		
6	61.73	52.23	9		
7	89.00	76.18	10	12	
8	50.79	42.53	11		

Fig. 16a. Reduction of parallelism for schedules shown in Fig. 11a.

system: 8×8×8, $r = 1/10$, $T_{PAR} = 1134$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned	
1	80.11	28.30	1	8
2	56.81	15.23	2	
3	84.99	24.21	3	7
4	46.46	7.37	4	
5	69.94	15.15	5	6
6	84.27	41.92	9	10
7	54.74	10.66	11	
8	56.74	22.16	13	

Fig. 16b. Reduction of parallelism for schedules shown in Fig. 11b.

system: 8×8×8, $r = 10/1$, $T_{PAR} = 3804$

Processor id	Total (%) utilization	Utilization (%)	Modules assigned		
1	72.34	71.22	1	8	
2	81.81	78.86	2	4	5
3	78.12	76.47	3		
4	87.44	85.56	6	9	
5	86.64	85.79	7	11	
6	61.75	61.42	10		
7	56.76	55.92	12		
8	44.93	44.45	13		

Fig. 16c. Reduction of parallelism for schedules shown in Fig. 11c.

For comparison purposes we reduced the 13 cluster obtained in the case of the multiple bus architecture to 8 assuming an 8 processor system. The results are shown in Fig. 15 and Fig. 16. We compare the two architectures in Fig. 17 by showing time frames and average total processor utilizations u_p^t. Note that the *best match between the application* we studied *and the two systems* considered *depends on the ratio r.*

Banyan Interconnection	Multiple Bus Interconnection
(8 processor system) $r = 1/1$ $T_{PAR} = 2529$ $u_p^t = 67.07$	(8×8×1) $T_{PAR} = 2421$ $u_p^t = 73.68$ (8×8×8) $T_{PAR} = 2421$ $u_p^t = 73.70$
$r = 1/10$ $T_{PAR} = 1083$ $u_p^t = 74.83$	(8×8×1) $T_{PAR} = 1207$ $u_p^t = 1134$ (8×8×8) $T_{PAR} = 1134$ $u_p^t = 71.35$
$r = 10/1$ $T_{PAR} = 3706.3$ $u_p^t = 73.10$	(8×8×1) $T_{PAR} = 3804$ $u_p^t = 71.22$ (8×8×8) $T_{PAR} = 3804$ $u_p^t = 71.22$

Fig. 17. Performance comparison of multibus and Banyan architecture for the robot elbow manipulator application.

The smallest time frame and the highest processor utilization were obtained for $r = 1/10$, and a Banyan interconnection system. The 8×8×8 multiple bus system has a higher bandwidth than the Banyan network, and yet the schedule produced when a Banyan system was considered results in best performance. From this we conclude the usefulness of the mapping methodology in matching application to architectures. Speed ups and efficiency factors for the 8 processor systems are presented in Fig. 18.

Speed up				Efficiency		
(8 processor systems)						
	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
$r = 1/10$	1.965	(1 bus)	1.763	.245	(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

Fig. 18. Comparison of speedups and efficiency for 8 processor Banyan and multiple bus interconnected systems.

7. CONCLUSIONS

The mapping of applications to multiprocessor systems provides a very good indication of how well the application is matched to the system. We have examined a robot elbow manipulator application which was partitioned into communicating modules at the equation level. A precedence graph representation was used. We have approximated different partitions of the same application by changing the ratio of processing to communication on a per module basis. Two classes of multiprocessor systems have been examined, the multiple bus systems and the Banyan multistage networks. They have been represented by their specification and their Queueing Delay vs. Utilization of their interconnection networks. A fast heuristic algorithm was used for the two different steps of the mapping method. In the first step, schedules of modules for every processors were produced. A full advantage of the application's parallelism was taken and the available system could match that. In the second step, a reduction of parallelism of the application was used to reduce the number of clusters to the number of available processors. For this step, new schedules were also produced. In the third step, the layout of the clusters to the processors is required. For the multiprocessor systems we have used it is a simple assignment of clusters to processors, since all processors are equally distant.

Our results indicated that the best match between an application and a system depends on the partition used, i.e., depends on r. The highest speed up was obtained when r was the highest and the parallelism of the application was fully exploited. This was possible only in the case of the multiple bus architecture systems. The processor utilization obtained were low. Thus, *for the particular application studied* best speed ups were obtained at the expense of underutilizing the system.

When the parallelism reduction step was applied (one less constraint) then the speed ups were lower but the processor utilizations were higher. After the application of this step it was possible to compare the match of the two different classes of systems to the application. Our results indicated again a dependence or r. For r=1 the multiple bus system produced the best speed up and at the same time highest average processor utilization while for r=.1,

10 the Banyan network produced best speed ups and the highest average processor utilizations.

Our results indicate the usefulness of the applied methodology to the solution of the mapping problem. It provides performance measures which closely agree with our initiation and at the same time it provides the assignments of modules to processors which are not at all obvious.

8. REFERENCES

[ABRA86] Abraham, S. G., 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, September 1986.

[ALLE78] Allen, A. O., *Probability Statistics and Queueing Theory,* Academic Press, 1978.

[BATC76] Batcher, K. E., "The Flip Network in STARAN," *1976 Int'l. Conf. on Parallel Processing,* Aug. 1976, pp. 65-71.

[BERM84] Berman, F. Snyder, L., "On mapping parallel algorithms in parallel architectures," *Proceedings of the International Conference on Parallel Processing,* 1984, pp. 307-309.

[BERM85] Berman, F., Goodrich, M., Koelbel, C., Robinson, III, W. J. Showell, K., "Prep-p: A mapping preprocessor for chip computers," *Proceedings of the International Conference on Parallel Processing,* 1985, pp. 731-733.

[BERM87] Berman, F., Haden, P., "A compartive study of mapping algorithms for an automated parallel programming environment," Computer Science Technical Report Number CS-088, University of California, San Diego, La Jolla, CA 92093.

[BOKH81] Bokhari, Shamid, H., "On the Mapping Problem," *IEEE Transactioins on Computers,* Vol. C-30, No. 3, 1981, pp. 207-214.

[CHU80] W. W. Chu, L. J. Holloway, M. T. Lan, K. Efe, "Task Allocation in Distributed Data Processing," *Computer,* November 1980, pp. 57-69.

[EFE82] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *Computer,* Vol. 15, No. 6, June 1982, pp. 50-56.

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

[GILB87] Gilbert, J. R., 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., January 1987.

[GYL76] V. B. Gylys, J. A. Edwards, "Optimal Partitioning of Workload for Distributed Systems," *Proceeding Compcon,* Fall 1976, pp. 353-357.

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

[HOUS87a] Houstis, C. E., Houstis, E. N., Rice, J. J., "Partitioning PDE Computation: Methods and Performance Evaluation," invited paper, *Journal of Parallel Computing,* No. 4, 1987.

[HOUS87b] Houstis, Catherine, "Allocation of Real-Time Applications to Distributed Systems," accepted for presentation in the *1987 International Conference on Parallel Processing,* August 1987.

[HOUS87c] Houstis, C. E., "Distributed Processing Performance Evaluation," accepted for presentation to the *Third International Conference on Data Communication Systems and Their Performance,* Rio di Janeiro, Brazil, June 22-25, 1987.

[JENN77] Jenny, C. J., "Process Partitioning on Distributed Systems," Digest of paper *National Telecommunications Conference,* 1977, pp. 31:1-31:10.

[KASA85] Kasahara, H., Narita, S., "Parallel Processing of Robot-Arm Control Computation on a Multiprocessor System," *IEEE Journal of Robotics and Automation,* Vol. RA-1, No. 2, June 1985, pp. 104-113.

[KLEI85] Kleinrock, Leonard, "Distributed Systems," *Communications of ACM,* Vol. 28, Number 11, 1985, pp. 1200-1213.

[KRUS83] Kruskal, C., Snir, M., "The performance of multistage Interconnection Nets for multiprocessing," *IEEE Transactions on Computers,* Vol. C-32, No. 12, 1983, pp. 1091-198.

[LAWR75] Lawrie, D., "Access and Alignment of Data in an Array Processor," *IEEE Transactions on Computers,* Vol. C-24, No. 12, Dec. 1975, pp. 1145-1155.

[MARS83] Marsan, M. A., Gerla, M., "Markov models for Multiple Bus Multiprocessor Systems," *IEEE Transactions on Computers,* Vol. C-32, No. 3, 1983, pp. 239-248.

[O'LEA85] O'Leary, D. P. and G. W. Stewart, "Data-flow algorithms for parallel matrix computations," *Communication of ACM,* Vol. 28, 1985, pp. 840-853.

[PATE79] Patel, J. H., "Processors-Memory Interconnections for Multiprocessors," in *Proc. 6th Ann. Symp. Computer Arch.,* pp. 168-177, Apr. 1979.

[MARI87] Marinescu, D., Rice, J., "Domain oriented analysis of DOE splitting algorithms," to appear in the *Journal of Information Science,* July 1987.

[SARK86] Sarkov, V., Hennessy, J., "Compile-time Partitioning and Scheduling of Parallel Programs," *Proceedings of the SIGPLAN 1986 Symposium on Compiler Instructions,* June 1986, pp. 17-36.

[SIEG78] Siegel, H. J. and Smith, H. D., "Study of Multistage SIMD Interconnection Networks," *5th Annual Symposium on Computer Architecture,* Apr. 1978, pp. 223-229.

[SIEG78] Siegel, H. J., McMillen, R. J., and Mueller, P. T., Jr., "A Survey of Interconnection methods for Reconfigurable Parallel Processing Systems," *1978 Int'l Conf. on Parallel Processing,* Aug. 1978, pp. 9-17.

[SIEG82] Siegel, L., Siegel, H. J., Swain, P. H., "Performance Measures for evaluating algorithms for SIMD machines," *IEEE Transaction on Software Engineering,* Vol. SE-8, No. 4, July 1984, pp. 319-331.

[SIEG85] Siegel, H., *Interconnection Networks for Large-Scale Parallel Processing,* Heath and Company, 1985.

[STO78] Stone, H. S., Bokhari, S. H., "Control of Distributed Processes," *Computer,* Vol. 11, No. 7, July 1978, pp. 97-106.