[10] P. Ramanathan and K. G. Shin, "Reliable broadcast in hypercube multicomputers," *IEEE Trans. Comput.*, vol. C-37, pp. 1654–1657, Dec. 1988.
[11] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Comput.*, vol. C-37, pp. 867–872, July 1988.
[12] S. B. Tien, C. S. Raghavendra, and M. A. Sridhar, "Reconfiguration embedded task graphs in faulty hypercubes by automorphisms," in *Proc. Hawaii Int. Conf. Syst. Sci.*, 1990, pp. 91–100.
[13] S. B. Tien and C. S. Raghavendra, "Algorithms and bounds for shortest paths and diameter in faulty hypercubes," in *Proc. 28th Allerton Conf. Commun., Contr., Comput.*, Univ. Illinois, Urbana-Champaign, Oct. 1990.n, pp. 216–225.
[14] P. J. Yang, S. B. Tien, and C. S. Raghavendra, "Embedding of rings and meshes onto faulty hypercubes," Tech. Rep., Dep. Elec. Eng., Univ. South. Calif., 1991.

Assignment of Task Modules in Hypercube Multicomputers with Component Failures for Communication Efficiency

Bing-rung Tsai and Kang G. Shin

Abstract—**The problem of assigning task modules within a hypercube multicomputer with possible link failures is investigated. A concept of *indirect optimization* is introduced and a function, called *communication traffic,* is proposed as the objective of optimization. The assignments obtained from optimizing this function are shown to significantly improve the actual communication performance measure, called *communication turnaround time,* over random assignments.**

Index Terms—**Task assignment, communication traffic and cost, NP-hard problem, link failures, fault-tolerant routing algorithms.**

I. Introduction

While the abundance of nodes in a hypercube multicomputer allows for executing tasks that require a large number of nodes, internode communication is still a major bottleneck in achieving the overall speedup. To achieve communications efficiency, considerable efforts have been made to improve routing algorithms and switching mechanisms, which are basically concerned with system-level implementations. Communication efficiency must also be improved on a per-task basis by exploiting the communication locality among task modules.

To assign task modules for an "optimal" performance, the run-time behavior of these modules must be known *a priori* to some extent. However, as stated in the Halting Problem in computing theory, there is no way to predict the exact run-time behavior of a program before it is actually executed. In case of distributed computation, it is also very difficult to predict the timing of communication events before a set of task modules are actually executed.

Manuscript received July 8, 1991; revised January 15, 1992 and July 20, 1992. This work was supported in part by the Office of Naval Research under Grants N00014-85-K-0122 and N00014-91-J-1115. Any opinions, findings, and recommendations in this paper are those of the authors and do not reflect the views of the ONR.

The authors are with Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122.

IEEE Log Number 9209042.

In the graph-mapping approach (e.g., [3]) the timing aspects of module communication are ignored, and a simple objective function is proposed for optimization. It is generally difficult to relate this objective to any of the well-known performance measures, such as task execution time. By contrast, any more complicated approach (e.g., [5]) requires a substantial amount of knowledge of the run-time behavior of task modules, which may not be available unless the task is tested thoroughly beforehand.

Our primary goal in this correspondence is to optimize communication performance. We use a relatively simple objective function and verify (with simulations) that optimizing this function actually leads to better communication performance, especially for assigning communication-bound tasks. Focusing on communication performance differentiates our work from others related to more generic aspects of task assignment. Taking a communication-oriented approach to the task assignment problem is hardly a limitation, since internode communication is of the utmost importance to the performance and fault-tolerance of any distributed system.

This correspondence is organized as follows. In Section II, we present the basic system model and assumptions used. Our problem is also formally stated there. In Section III, the NP-hardness of minimizing communication traffic is stated first in order to justify the use of heuristic algorithms. Several heuristic algorithms are then used to find good suboptimal solutions. These algorithms are tested extensively for various inputs to assess the quality of the assignments obtained from them. We then simulate these algorithms to verify the actual quality of the assignments found by minimizing communication traffic. The effects of inaccuracy in describing the task behavior are also discussed there. Section IV deals with the case where an alternative fault- tolerant routing scheme is used. The correspondence concludes with Section V.

II. Preliminaries

The communication volume between each pair of modules is expressed in the number of packets to be exchanged between them. A message may be composed of a number of packets. Intermodule communications are assumed to be accomplished via message passing. A message is routed from the source to the destination via a fault-free shortest path under circuit or message switching.

Since most existing hypercubes do not support a per-node multiprogramming environment, it is assumed that at most one module is assigned to a node, i.e., the mapping between nodes and modules is one-to-one. For a task with M modules such that $2^{n-1} < M < 2^n$ for some integer n, one can add some "dummy" modules and make it a task with 2^n modules. So, we will henceforth assume $M = 2^n$ where n is the dimension of the subcube allocated by the host to execute the task, and thus the mapping of modules into subcube nodes is one-to-one and onto.

For a network of nodes, we define a *communication event between modules* (CEBM) as an instance that a *module* needs to send a message to another module, while defining a *communication event between nodes* (CEBN) as an instance of a *node* needing to send a message to some other node. In circuit switching, these two are indistinguishable. In message switching, however, a single CEBM can become several CEBN's. For example, when a pair of modules reside in two different nodes which are 2 hops apart, in circuit switching a CEBM from one module to the other is just a CEBN from one node to the other node. For message switching, however, this CEBM becomes two CEBN's: one from the source to the intermediate node,

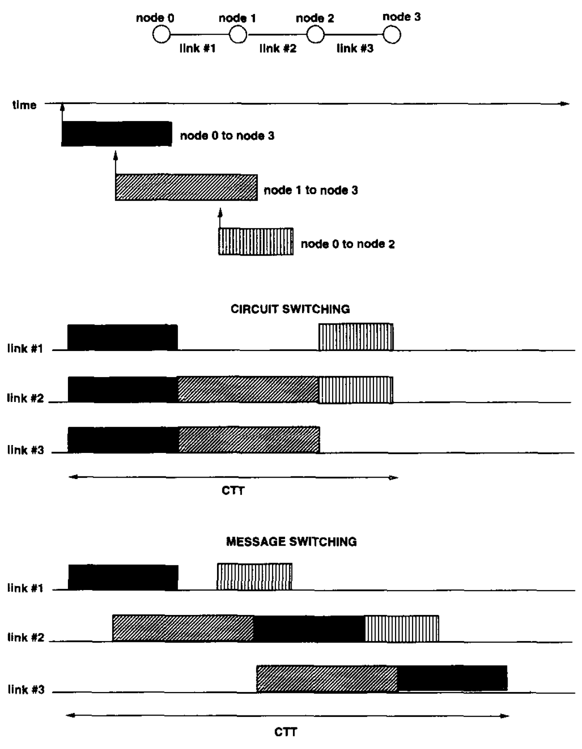

Fig. 1. An example to demonstrate the definition of CTT.

and the other from the intermediate node to the destination. We said there is an *outstanding* CEBN if a message is to be sent by a node. An outstanding CEBN is said to be *processed* if it is sent from the source node to a neighboring destination node. An outstanding CEBN may not be processed immediately due to the limited link resources available. A CEBN is said to be *blocked* if it is not processed immediately.

The goodness of a task assignment for hypercubes is measured by the *communication turnaround time* (CTT), which is the time span from the first CEBN becoming outstanding to all CEBN's being processed. As an illustrative example, in Fig. 1 we have a simple network of 4 nodes with 3 CEBM's. The status of each link during the execution under both circuit and message switching is shown in this figure. Note that the computation time needed is invariant among different assignments, since at most one module is assigned to each node. Therefore, CTT is the main source of difference in the completion time of a task.

CTT cannot be easily described as a mathematical function, and the exact value of CTT depends largely on the timing of communication events, thus making it impractical to use any direct optimization of CTT. However, as we shall see, for communication-bound tasks, minimizing a certain simple function can usually minimize CTT.

The *communication cost* in executing a set of task modules is defined as the sum of time units during which links are kept busy with the messages among these modules. In other words, it is a measure of the total communication resources used by an instance of task execution measured in time units.

Suppose $c(h)$ is the number of time units links are kept busy with a packet sent over a path of h hops. The sum of time units links are kept busy for related purposes other than packet transmission—such as establishing a connection—and are assumed to be negligible. For message-switched hypercubes, $c(h) = hc(1)$, but this relation may not be accurate for circuit- switched hypercubes. However, if the "call request" signal to hunt for a free path occupies each link only for a very short time, then this expression would be a good approximation even for circuit-switched hypercubes.

By defining $c(1)$ as a *unit* of communication traffic (i.e., the link usage by one packet traversing one link), the communication traffic resulting from executing a task under assignment $\boldsymbol{a}$ becomes: $k(\boldsymbol{a}) = \sum_{1 \le i \le n} iv_i$, where v_i is the number of packets traversing over i links. One can easily see that $cost_{com}(\boldsymbol{a}) \propto k(\boldsymbol{a})$.

In both types of switching, communication traffic is proportional to the total link occupation time, two communicating modules placed far apart will require more communication resources, and there is a higher possibility that some other instances of communication will be blocked and/or delayed, which in turn leads to an increase of CTT. Therefore, reduction of communication traffic is crucial to the CTT associated with communication-bound tasks.

When introducing the notion of communication cost and communication traffic, we deliberately avoided the low-level timing details. We only consider the total number of packets to be sent/received between a pair of task modules during the whole mission time, thus allowing for a simple objective function that can be translated into a simple combinatorial optimization problem.

The following notation will be used throughout the correspondence.

n: the dimension of a subcube available for executing the task under consideration.

$\boldsymbol{U}$: an $M \times M$ communication volume matrix, where U_{ij} is the communication volume from m_i to m_j expressed in number of packets, and M is the number of task modules. As mentioned earlier, we will assume $M = 2^n$ unless specified otherwise. Note that $U_{ii} = 0 \forall^i$, since a module does not send messages to itself.

$\boldsymbol{a}$: a $1 \times M$ vector denoting an assignment, the ith component of which represents the fact that m_i is assigned to a node whose address is $a_i, 0 \le i \le M-1$.

$D(n_i, n_j)$: the distance (i.e., the length of a shortest path) between node n_i and n_j, and is dependent upon the routing algorithm used. For now, we will assume $D(n_i, n_j) = D(n_j, n_i)$. (The case where $D(n_i, n_j)$ can be different from $D(n_j, n_i)$ will be discussed in Section VI.) Before making a module assignment, $D(n_i, n_j)$'s are calculated for a subcube assigned to the task under consideration with a shortest-path routing algorithm. Note that the distance between a pair of nodes may be greater than their Hamming distance, and depends on the number of faulty links and the routing scheme used.

III. Optimization Algorithms and Performance Evaluation

Although the objective function we proposed is simple in nature, optimizing it is still a difficult problem, as formally stated in the following theorem.

Theorem 1: Given an $M \times M$ task communication volume matrix $\boldsymbol{U}$ where $M = 2^n$, it is NP-hard to find an optimal mapping $\boldsymbol{a}$ of an M-module task onto an n-dimensional fault-free hypercube.

The theorem can be proved by restricting to the fault-free hypercube embedding problem discussed in [4]. The proof is presented in [6] and will not be repeated here.

Thus, there is no known polynomial-time algorithm to find an optimal mapping/assignment. Note that minimizing CTT, rather than communication traffic itself, is our ultimate goal. As we shall see, good heuristic algorithms will suffice in most situations. An optimal solution that minimizes communication traffic is usually computationally expensive, and may only improve slightly over fast algorithms in terms of minimizing CTT, our actual objective.

One simple greedy heuristic which has been tested to work well in fault-free cases [6] is given below. Consider each task as a weighted graph with vertices representing modules and edge weights representing communication volumes. For any two nodes x and y under the shortest-path routing, $D(x, y) = D(y, x)$. Therefore, it is sufficient to use an undirected graph with $U_{ij} + U_{ji}$ as the weight on the edge connecting m_i and m_j. We want to find a Hamiltonian cycle in this task graph with as high a total edge-weight as possible, and then embed this cycle into a Hamiltonian cycle in the hypercube. A Hamiltonian cycle in a fault-free hypercube can be easily found with Gray-code enumeration.

In an injured hypercube with faulty links, however, there may not be any Hamiltonian cycle available for embedding. So, we define a *weighted relaxed* (WR) Hamiltonian cycle in an injured hypercube (with no disconnected node) as a relaxed version of Hamiltonian cycle, such that two nodes x and y can be linked in the cycle via a *virtual edge* which may be a path from x to y through some intermediate nodes. The weight on each virtual edge of the cycle is the number of physical edges on it. The greedy algorithm embeds the Hamiltonian cycle in the task graph with the maximum weight (found by a greedy approach) into the minimum-weight WR Hamiltonian cycle in the injured hypercube (also found via a greedy approach).

Two other (more complex) heuristic algorithms are also implemented and tested: a bottom-up approach algorithm similar to the one proposed in [3], and a top-down approach proposed in [2]. Both of these algorithms are modified to handle cases with broken links. A third nondeterministic approach using the simulated annealing method is also implemented and tested, where 2-opting is used as the perturb function.

To compare the quality of the assignments found by these algorithms with respect to communication traffic, we simulated these algorithms using input tasks with randomly generated communication volumes among their modules.

Each algorithm was executed for 1000 randomly generated tasks where U_{ij}'s are characterized by a normally distributed random variable with mean μ and variance σ^2. Changing the value of μ is found to have little effect on the relative performance of assignments found with different algorithms as long as the ratio σ/μ remains constant. It is also found that, as σ/μ approaches zero, the difference in communication traffic between random assignments and those assignments found with the above three algorithms gets smaller, while the difference gets larger as σ/μ increases. This is consistent with the fact that when all U_{ij}'s are identical, all assignments will lead to an identical communication traffic, and all assignment algorithms will perform

For the input tasks used to obtain the plots in Fig. 2, U_{ij}'s are characterized with $\mu = 20$ and $\sigma = 15$; the horizontal axis depicts the number of faulty links while the vertical axis represents communication traffic. In this figure, "A1" represents the greedy algorithm, while "A2" represents the communication traffic achieved with either top-down or bottom-up algorithms, whichever yields smaller communication traffic. This is to enhance the readability of the plots since the performance of the top-down and bottom-up algorithms turns out to be very close to each other.

It can be seen from the above result that the greedy approach performs surprisingly well. Complex (i.e., top-down and bottom-up) approaches outperform the simple greedy approach only by a small margin. Furthermore, as the number of faulty links increases, the gap between the two curves gets narrower. This can be explained by the fact that both the top-down and bottom-up approaches are best suited for fault-free (thus regular) hypercubes. For hypercubes with faulty links, the interconnection structure is no longer symmetric or regular. in such a case, the partitioning mechanism in the top-down approach and the combining mechanism in the bottom-up approach must use less accurate heuristic decisions, hence degrading the performance.

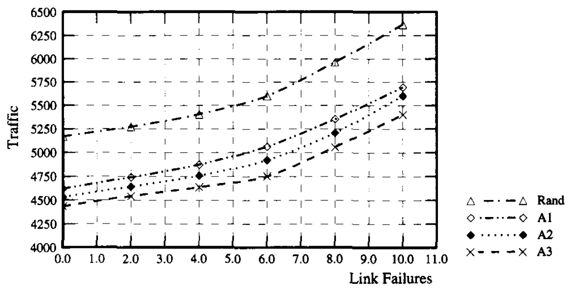

Fig. 2. Comparison of communication traffic.

TABLE I
TIMING COMPARISONS FOR VARIOUS ALGORITHMS

	CPU Time			
Size	Greedy	Top-Down	Bottom-Up	S-Annealing
$n = 3, M = 8$	.057	4.2	7.8	125.2
$n = 4, M = 16$	.178	15.3	28.3	433.6
$n = 5, M = 32$	1.215	172.4	297.6	2537.1

The simulated annealing approach ("A3"), on the other hand, has shown more consistent performances. Its advantages over other algorithms become more pronounced as the cube size and the number of link failures increase. Therefore, we can conclude that this approach is more adaptable to irregular structures.

In Table I, we show the relative timings of various algorithms used. The algorithms are tested on a DEC 5000 workstation running Ultrix operation system. Although we have only shown the performance data for problem size of $n = 4, M = 16$, the relative performances of different algorithms are found to be consistent at least up to the problem size of $n = 8, M = 256$.

To demonstrate why minimizing communication traffic can be effective, we also need to compare the CTT's of those assignments found with different algorithms. Our simulation model for this purpose is described below.

Timing: A *time unit* is selected as the time required to send a packet over a single communication link.

Routing Algorithm and Mechanism:

- Link failures are detected before task assignment and execution. Each message is routed through a fault-free shortest path determined prior to the execution of this task. We assume there are no additional link failures during the execution of this task.
- Under message switching, the routing mechanism at an intermediate node on a path will take a certain amount of time to forward a message from one link to the next. We assume this time to be relatively small and absorbed into the length of the corresponding message.
- The propagation delay on a communication path is assumed to be negligible.

Task Communication Behavior:

- T, given for each task, denotes the time span between the arrival time of the first and the last CEBM's. The arrival times of CEBM's are uniformly distributed in [0, T]. Hence, for a given task assignment, a larger T represents the task being more computation-bound, while a smaller T represents the task being more communication-bound.
- L_{msg} denotes the maximum message length measured in number of packets. The communication volume between each pair of modules is randomly grouped into messages of lengths within [1, L_{msg}].

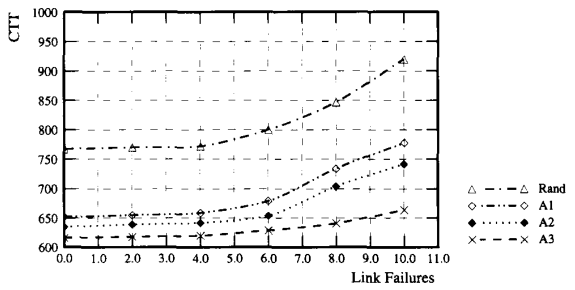

Fig. 3. Comparison of CTT's.

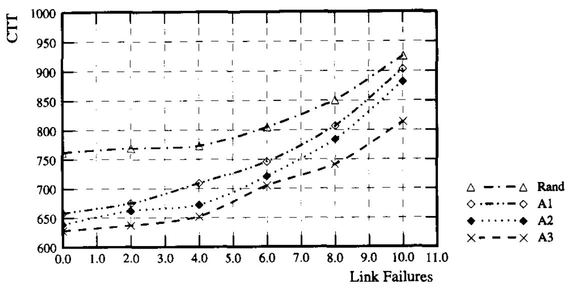

Fig. 4. Comparison of CTT's with inaccurate U_{ij}'s.

TABLE II
EFFECTS OF CHANGING T UNDER MESSAGE SWITCHING

	$n = 3, M = 8$				$n = 4, M = 16$				$n = 5, M = 32$			
T	Rand	A1	A2	A3	Rand	A1	A2	A3	Rand	A1	A2	A3
10	220	176	176	173	767	652	635	616	2456	2007	1994	1924
25	220	176	175	173	767	653	636	616	2469	2011	1995	1928
50	221	177	177	175	769	655	636	617	2474	2026	2005	1940
100	222	182	178	176	770	655	637	618	2475	2028	2013	1949
200	308	306	305	302	772	657	639	620	2476	2037	2025	1961
300	311	308	305	303	775	661	642	622	2478	2048	2031	1993

Message Scheduling and Queueing: If a link is busy when it is to be used for transmitting an incoming message, the message is stored in an FIFO queue at the source end of the link. When more than one message requests the use of the same link at a time, one of them is randomly chosen to use the link. This selection procedure is repeated until all requests are honored.

The goal of our simulation is to comparatively evaluate the goodness of different assignments under the same execution environment, but not to compare the performance of different system implementations. So, the simulation results should not be used to determine the relative performance of different switching methods or routing algorithms.

The assignments found are fed into an event-driven simulator to evaluate their performance in a close to real-world environment. The results are plotted in Fig. 3 for message switching systems. Input tasks used here are the same as those used for Fig. 2. We set $L_{msg} \epsilon$ [1, 5], and $T = 100$. Results for circuit-switched hypercubes are found to be similar in most situations and thus are not presented.

The effects of changing T under the same assignment for a given task are shown in Table II for message switching without link failures. The results are found to be similar to those under circuit switching. For the cases of $n = 4$, $M = 6$, and $n = 5$, $M = 32$, changing T in the range [10, 300] does not have any significant impact on the relative performance of assignments found with different algorithms. The assignments found with all of the above algorithms have shown substantial improvements over random assignments $\forall T \in [10, 300]$. This is because the network gets saturated with messages when $T = 300$.

In case of $n = 3$, $M = 8$, the network becomes less congested at $T \approx 160$ and the differences of CTT's among different assignment algorithms start to diminish. So, we can conclude that minimizing communication traffic yields a peak improvement when the task to be assigned is communication-bound and the communication network may become highly congested during the execution of this task. For $n = 4$, $M = 16$, the T value which results in small performance differences is approximately 750, while for $n = 5$, $M = 32$, it is about 2250. However, when T is relatively small and the network is not near saturation, the difference in message queue length can be made smaller by using the assignments obtained from the minimization of communication traffic. Depending on system implementation, the performance of a node may also be influenced by the length of message queue it has to maintain.

The effects of changing L_{msg} are more subtle than changing T. Generally, shorter message lengths result in better performances in circuit-switched hypercubes, while for message-switched hypercubes, changing the message length does not affect system performance notably if the overall communication traffic is fixed.

Our simulation results have indicated that different switching techniques do not matter much to system performance for communication-bound tasks. Circuit switching is shown to have only a slightly better performance than message switching for the same task assignments. However, as mentioned earlier, the actual performance will depend on system implementation, and thus, the simulation results should not be used to compare the effectiveness of the two switching methods.

When the number of faulty links grows within our preset range (i.e., less than one third of all links), CTT also increases. For smaller hypercubes, such as $n = 3$, introducing even one more faulty link can make a significant difference in CTT. This effect gets more pronounced when the number of link failures becomes larger, as one can see in Fig. 3. As the cube size increases, there will be more fault-free links, hence making lesser impacts of a single link failure on system performance.

Although the proposed assignment scheme requires only minimal information of run-time task behaviors, we still need the communication matrix to assign a task. It is obvious that unless the task has been fully tested and each message length is exactly calculated, the entries in the communication matrix cannot be absolutely accurate. To study the effects of an inaccurate communication matrix, we repeated the simulation for evaluating CTT while introducing uncertainties in the communication matrix. in Fig. 4, the input tasks are essentially the same as those in Fig. 3, but there is a maximum of 20% error in each U_{ij}, i.e., during an instance of actual task execution, the number of packets exchanged between m_i and m_j is $U_{ij} \pm 0.2U_{ij}$. From Fig. 4, one can see that inaccuracies in U_{ij}'s affect communication performance, especially when the cube size and number of link failures are large. However, when the number of link failures is less than one- sixth of all links, the overall performances of various assignment algorithms are still quite close to those in the case with exact U_{ij}'s.

IV. AN ALTERNATIVE ROUTING ALGORITHM

Thus far, we have assumed that the hypercube is implemented with a routing scheme which routes messages from the source to the destination via fixed, shortest paths determined before the execution of each task. However, there are several practical problems with this assumption. For instance, all faulty links must be known before making a task assignment, which may not always be possible. Also, if additional link failures occur after the assignment, the execution of the task may become unsuccessful.

To overcome these problems, we must use a routing algorithm that is more adaptive to system changes. For instance, the DFS routing scheme proposed in [1] is an adaptive fault-tolerant routing algorithm which uses only a limited amount of global link status information. Under this algorithm, the system does not require *a priori* link status information, and communications can be completed even if some unexpected link failures occur during task execution as long as all nodes involved remain connected. However, due to the adaptive nature of the DFS routing algorithm, it is difficult to predict the length of the path used for routing a message during task execution, especially in the presence of link failures. So, $D(x, y)$ cannot be accurately estimated, thus making it difficult to minimize the overall communication traffic. Furthermore, under some routing scheme like the DFS routing, due to the lack of global link status information, the length of the path chosen for communication from node x to node y may not be the same as the one chosen for that from y to x. For example, suppose we have a 3-cube with three broken links, 00$*$, 0$*$0, and $*$01. Then the length of path chosen under the DFS routing from 000 to 111 is 3. But the path chosen to route messages from 111 to 000 is 111 $\rightarrow$ 110 $\rightarrow$ 001 $\rightarrow$ 101 $\rightarrow$ 001 $\rightarrow$ 110 $\rightarrow$ 010 $\rightarrow$ 011 $\rightarrow$ 001 $\rightarrow$ 000, which has a length of 9. The routing schemes with this nature are said to be *asymmetric*. In most cases, a routing scheme becomes asymmetric only in the presence of faulty components.

Based on the above observations, one may jump to a conclusion that there is no way to minimize the communication traffic of an assignment, and hence it will be impossible to improve communication efficiency by appropriately placing task modules. However, as our simulation results show below, use of the proposed objective function, even by assigning task modules to the nodes as if there were no faulty links, can still significantly improve communication performance over random assignments when the number of faulty links is within a certain range.

Three assignment strategies are compared in our simulation. The first is the usual random assignment. The second is to apply the greedy algorithm to the hypercube without knowing which links are faulty. The third assumes perfect knowledge of link failures and how each message will be routed during the execution. This strategy is an unrealistic, ideal case, which gives an upper bound of performance improvement with communication traffic, whereas the second strategy provides a lower bound. In real applications, depending on the knowledge available during the task assignment phase, the performance should lie somewhere between these two extremes.

Fig. 5 shows the communication traffic of the assignments under the DFS routing for the same set of input tasks as in Fig. 2. "S1" represents the assignments found with no knowledge of faulty links, while "S2" represents those found with complete knowledge of faulty links and the routing paths of all messages. It can be easily seen that under the DFS routing, the overall communication traffic is higher than the routing algorithm used before. Nevertheless, the assignments "S1" still generate smaller communication traffic than random assignments, although the improvement becomes insignificant as the number of faulty links increases.

The same set of input tasks used in Fig. 3 are employed again for event-driven simulations, except that the DFS routing is used here. Since the DFS routing is designed based on the operating principles of message switching, we only simulate the hypercubes implemented with this switching method.

The measured CTT's of these assignments are plotted in Fig. 6. It is found that, without knowledge of faulty links, assignment "S1" still improves over random assignments with a margin of a least 10%

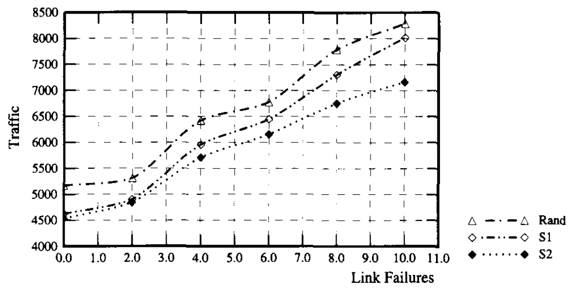

Fig. 5. Comparison of communication traffic under the DFS routing.

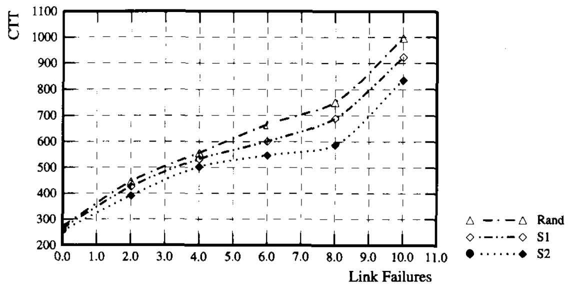

Fig. 6. Comparison of CTT's under the DFS routing.

when the number of faulty links is more than one-eighth of the total links. This margin increases as the number of faulty links increases, but starts to level off when the percentage of faulty links approaches 33%. The assignment "S2" shows even larger improvements and improves over random assignments with a steadily increasing margin as the number of link failures increases.

By comparing Fig. 6 to Fig. 3, one can see that, although the DFS routing results in an overall higher communication traffic, it results in smaller CTT's when the number of faulty links is relatively small. This is due to the fact that the DFS routing chooses communication paths in a more "spread out" fashion and causes less congestion than the shortest fixed-path scheme used before. This advantage diminishes after the number of faulty links grows beyond one-fifth of all links. When the percentage of faulty links reaches 25%, the DFS routing begins to yield larger CTT's than the shortest path routing. This is because paths available between nodes are becoming fewer, so messages cannot be spread out to more paths under the DFS routing. Also, the greater communication traffic overhead of the DFS routing starts to have dominant effects. Note, however, that implementation details will be crucial in actual applications, and these simulation results should not be used to judge the relative merits of different routing algorithms.

V. Concluding Remarks

Using a simple objective function, we formulated and solved the problem of mapping a task which is composed of multiple interacting modules into a hypercube with possible faulty links. The goal was to optimize communication performance, measured in communication turnaround time. Due to the difficulties in optimizing this objective directly, a function called communication traffic is proposed. By minimizing this function, we could find assignments with the optimal communication performance using heuristic combinatorial techniques. Several algorithms that find assignments by minimizing communication traffic are implemented and comparatively evaluated. The assignments found with these algorithms are also evaluated with simulations. It has been shown that for communication-bound tasks, they have significant improvements over random assignments with respect to an actual communication performance measure, i.e., the communication turnaround time.

We also analyzed the case where an alternative routing algorithm like the DFS routing is used. Our task assignment criterion is again shown to work well in this case.

Although we have focused our attention on hypercube multicomputers, the objective function we developed can be generalized to other distributed systems with different interconnection topologies. In fact, when we consider hypercubes with faulty links, they are actually no longer hypercubes, but they are subgraphs of hypercubes. For systems with other interconnection topologies, as long as they adopt message switching or circuit switching and the length of the path chosen by the routing scheme between each pair of nodes is known before a task assignment, or assignment criterion can be applied to these architectures.

REFERENCES

[1] M.-S. Chen and K. G. Shin, "Depth-first search approach for fault-tolerant routing in hypercube multicomputers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, pp. 152–159, Apr. 1990.

[2] F. Ercal, J. Ramanujam, and P. Sadayappan, "Task allocation onto a hypercube by recursive mincut bipartitioning," in *Proc. 3rd Conf. Hypercube Concurrent Comput. Appl.*, Jan. 1988, pp. 210–221.

[3] S. Horiike, "A task mapping method for a hypercube by combining subcubes," in *Proc. 5th Distrib. Memory Comput. Conf.*, Apr. 1990, pp. 909–914.

[4] D. W. Krumme, K. N. Venkataraman, and G. Cybenko, "Hypercube embedding is NP-complete," in *Proc. 1st Conf. Hypercube Concurrent Comput. Appl.*, Aug. 1985, pp. 148–157.

[5] V. M. Lo, "Temporal communication graphs: A new graph theoretic model for mapping and scheduling in distributed memory systems," in *Proc. 6th Distrib. Memory Comput. Conf.*, Apr. 1991, pp. 248–252.

[6] B.-R. Tsai and K. G. Shin, "Communication-oriented assignment of task modules in hypercube multicomputers," in *Proc. 12th Int. Conf. Disrib. Comput. Syst.*, June 1992, pp. 38–45.

A Multiaccess Frame Buffer Architecture

D. T. Harper, III

***Abstract*—Many current graphical display systems are based around a memory array commonly known as a frame buffer. In these systems, the frame buffer contains the array of pixels currently being displayed. Updates to the display are accomplished by modifying the values in the frame buffer. This brief contribution demonstrates how the performance of frame buffer based systems can be improved by decreasing the number of accesses to the frame buffer memory array. The proposed architecture, referred to as a multiaccess frame buffer, allows parallel access to constant area rectangles of the array of pixels stored in the frame buffer rather than the row oriented accesses required by most current frame buffer architectures. By allowing more general types of access, a given update can be performed with fewer frame buffer accesses.**

***Index Terms*—Frame buffers, Parallel memory architecture, computer graphics, interleaved memories, computer architecture.**

Manuscript received July 22, 1991; revised January 13, 1993. This work was supported by the National Science Foundation under Grant CCR-8908612 and by the Texas Advanced Research Program under Grant 9741-004.

The author is with the Department of Electrical Engineering, The University of Texas at Dallas, Richardson, TX 75983-0688.

IEEE Log Number 9212765.

I. INTRODUCTION

As the demand for larger and faster graphical display systems increases, the limiting factor in raster display performance is becoming the rate at which the frame buffer can be updated. Because large amounts of memory are required to implement display systems, it is usually impractical, due to cost and memory density considerations, to use high-performance static RAM technology. Therefore, it is necessary to use parallelism in the display system architecture to achieve high memory bandwidths.

Recent work by Gharachorloo *et al.* has evaluated several techniques that have been proposed to increase display system performance [5]. Within the category of frame buffer rasterization techniques, the authors distinguish between techniques based on the number of pixels processes in parallel and whether or not processors are embedded in the memory array as has been proposed in the Pixel Planes [4] and scan-line access memory (SLAM) architectures [2]. Although these systems are capable of very high performance, they also require significant custom hardware.

Rasterization techniques that use standard memory chips achieve parallelism on a lower scale by reading and writing multiple pixels in parallel. Most raster displays limit parallel access to consecutive pixels on a single row of the display. This is the architecture of choice because pixels must be read from the frame buffer in row order to drive the video circuitry. A disadvantage of these architectures is understood by considering the display of a vertical line. In this case, one write to the frame buffer must be made for each pixel on the line. To improve the performance of the system on vertical lines, Ostapko [11] has proposed on architecture that permits reading and writing of rows and columns. For similar reasons, Sproull *et al* . [13] and Gupta *et al.* [6] have proposed an architecture that permits writing to 8×8 rectangles; their intent is to provide an efficient mechanism ti fill areas while also providing some degree of parallelism during row (vertical line) and column (horizontal line) access to the memory. Whelan [14] has proposed a display system architecture that permits a rectangular area to be filled in parallel—the system effectively broadcasts a common value to an array of pixels; it does not allow different values to be written to different locations in the rectangle.

In this brief contribution, a frame buffer architecture is proposed which consists of $N = 2^n$ independent memory banks or modules. The array of pixel values that is to be displayed is stored in the modules. The size of the pixel array is assumed to be $R \times C$ where C is a power of two ($C = 2^c$). For example, for a typical 1280×1024 pixel display device, $R = 1280, C = 1024$, and $c = 10$. The exact value of R does not affect the frame buffer architecture and is not considered further.

The proposed system permits parallel access to any $N/2$ pixel rectangle within the $R \times C$ array which has dimensions that are powers of 2 (that is, the rectangle that 2^p rows and 2^q columns and $p+q = n-1$). Thus, the architecture supports full parallel access to rows (rectangles with $q = 0$), columns (rectangles with $q = n-1$), and any other constant area rectangle that has a column dimension equal to a power of 2 and less than N (Fig. 1). The rectangles, also referred to as *blocks* in this brief contribution, may begin at any position within the pixel array with the only constraint on orientation being that all pixels which lie in a single row of the block, also lie within a single row of the pixel array. The architecture will be referred to as a *multiaccess* frame buffer.