1-1-1987

The Gradient Model Load Balancing Method

Frank C. H. Lin

Robert M. Keller
Harvey Mudd College

The Gradient Model Load Balancing Method

FRANK C. H. LIN AND ROBERT M. KELLER, MEMBER, IEEE

Abstract—**A dynamic load balancing method is proposed for a class of large-diameter multiprocessor systems. The method is based on the "gradient model," which entails transferring backlogged tasks to nearby idle processors according to a pressure gradient indirectly established by requests from idle processors. The algorithm is fully distributed and asynchronous. Global balance is achieved by successive refinements of many localized balances. The gradient model is formulated so as to be independent of system topology.**

Index Terms—**Applicative systems, computer architecture, data flow, distributed systems, load balancing, multiprocessor systems, reduction architecture.**

I. Introduction

LOAD balancing enables a multiprocessor system to distribute tasks effectively to various processing nodes such that the aggregate throughput of the system is maximized. Throughput is normally measured by the system response time, but it is difficult to use this *a posteriori* measure to improve performance dynamically. Hence, many load balancing studies rely on a secondary measure, that of processor utilization, to govern load-balancing. The intuitive idea is that if processor utilization can be increased without undue overhead, then response time will be improved.

Conceptually, a load balancing algorithm implements a mapping function from tasks to processors. A static mapping may be exercised during program compilation or loading [6], [1], [5], [17], [7], [18]. Dynamic balancing [15], [14], [17], on the other hand, deals with decisions relating to the mapping of tasks during the computation itself. The effectiveness of a static balancing method hinges on the accuracy of the assignment function whereas the effectiveness of a dynamic balancing depends on the efficiency of the task migration techniques.

This study focuses on dynamic load balancing issues of loosely coupled, large-scale applicative systems [8], [1], [10], [11], [20]. A program is started by generating a task at one processor. For a parallel application, this task will "spawn" additional tasks, which in turn continue spawning to build up a work backlog, requiring further dispersal of the work load.

Section II mentions related load balancing strategies. The gradient model for load migration and balancing is described in Section III. Variations of the gradient model in heterogeneous systems are also presented. An implementation of the proposed model on an applicative multiprocessor system is described in Section IV, followed by the presentation of simulation results.

Manuscript received January 31, 1986; revised June 16, 1986.

F. C. H. Lin is with ESL Inc., 495 Java Drive, Sunnyvale, CA 94088.

R. M. Keller was with the Department of Computer Science, University of Utah, Salt Lake City, UT 84112. He is now with Quintus Computer Systems, Inc., Mountain View, CA 94041.

IEEE Log Number 8611359.

II. Related Research

Kratzer [14] suggested a swap-bid protocol for distributed load balancing. When a processor receives a "status update message," it finds the best possible task movement to/from another node or a swap of tasks with respect to a performance estimation heuristic.

Load balancing in the Purdue Engineering Computer Network [7] is implemented by a deterministic balancing policy. A load average, which represents the degree of idleness, is maintained in each network machine's kernel. Before a command is processed, the load averages from every network machine are obtained and the one with the minimum load average is chosen.

Ni [15] proposed a load balancing method for a small scale point-to-point multiprocessor system. An idle processor sends a request message to its neighbors. The neighbors respond with a busy or not-busy status indication. The idle processor then selects a target neighbor and sends it a draft message. The target processor may either respond with a new task or respond with a too-late message if the new task has been drafted by another processor. Tasks can be migrated at most one hop away from the originating host. A modified version of the draft protocol was recently published [16].

Stankovic [19] suggested using an expert system for heuristic scheduling algorithms. This is a broad and ambitious approach and no details of such an expert system have been described up to now.

Turning to applicative systems, the data flow machine proposed by [4] used a round-robin centralized scheduler to arbitrate operation packets to processing units. The AMPS project [8] employed a tree structure which recursively balanced tasks onto the processor tree. Load balancing was handled by the nonleaf processors, with tasks shifted from one subtree to another in order to reduce load differential between adjacent subtrees.

Gostelow [6] proposed a token-ring network where each node had four processing elements and one shared local memory. New tasks were mapped onto processors by a system-wide hash function. Several hash functions were studied and it was concluded that system performance increased if program locality could be enforced by the hash function.

0098-5589/87/0100-0032$01.00 © 1987 IEEE

The ZAPP system [1] attempted to match the creation of new tasks to available processors while executing a divide and conquer algorithm. This too used the idea of stealing a task from a neighbor for load balancing.

III. The Gradient Model

The gradient model is a localized load balancing method where every processor interacts only with its immediate neighbors. A global balancing is achieved by propagation and successive refinement of local load information. An idle or underutilized processor initiates the load balancing activities by *demanding* more work load. The demand is indirectly relayed through the system in a manner to be described. A demand is fulfilled by the arrival of a task or tasks from other heavily loaded processors.

A. Gradient Surface

The gradient model employs a two-tiered load balancing algorithm. The first step is to let each individual processor determine its own loading condition. The time-varying *load state* of a processor may be *light*, *moderate*, or *heavy*. Colloquially, if a processor is light, it wishes to have more load given to it. If it is heavy, it wishes to get rid of some of its present load. If neither of these conditions holds, then it is moderate.

Definition: The *distance d* between two processors i and j, of a multiprocessor network is the length of the shortest path between i and j. The *diameter* of a multiprocessor network N is the maximum distance between any two nodes of N, i.e.,

$$\text{diameter } (N) := \max \{d_{i,j} \text{ for all } i, j \text{ in } N\}$$

Definition: The *gate* of a processor i is a binary function g_i. A gate is open if the node is lightly loaded. Otherwise it is closed. In the gradient model, g_i is defined as:

$$g_i := 0 \qquad \text{if gate } i \text{ is open}$$

$$g_i := w_{\max} \qquad \text{if gate } i \text{ is closed}$$

where $w_{\max} = \text{diameter } (N) + 1$.

Intuitively, a processor welcomes the influx of new tasks by opening its gate. The second step of the gradient model load balancing method is to establish a system-wide gradient surface to facilitate task migrations. The gradient surface is represented by the aggregate value of all *proximities*.

Definition: The *proximity* of a processor i, w_i, is the minimum distance between the processor and a lightly loaded node in the system. If there is no light node in a system, w_i is defined as $w_{\max}$. This means,

$$w_i := \min \{d_{i,k} \text{ over } k \text{ where } g_k = 0\}$$

$$\text{if there exists a } k \text{ such that } g_k = 0$$

or

$$w_i := w_{\max} \qquad \text{if for all } k,\ g_k = w_{\max}.$$

The proximity of a light node is zero. The proximity of its immediate neighbors is one, indicating that these nodes are one hop away from a light processor. The proximity of the neighbor's neighbor is two, etc. A system without any light node can be considered as a system with a light node at a distance beyond the diameter of the network.

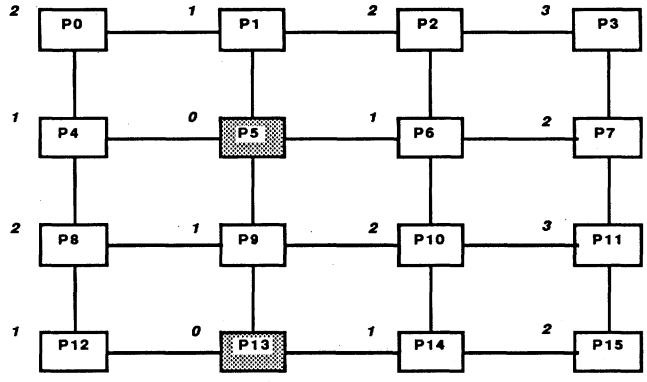


Fig. 1. Proximity distribution and gradient surface.

Definition: The *gradient surface* (GS) of a network is the collection of proximities of all processors. GS = $[w_1, w_2, w_3, \cdots, w_n]$.

Given a neighboring relation and a gate distribution, the gradient surface of a system is stable and determinate. As an example, Fig. 1 depicts a system with a 4 × 4 rectangular configuration. Assume nodes 5 and 13 are lightly loaded; the proximity function of each processor is shown in italic. These values comprise a gradient surface.

B. Gradient Surface Approximation

The gradient surface has multiple attributes. First, it is a network-wide indication of all under-utilized processors. Second, it carries an implicit request for work load. Third, it serves as a minimum distance routing pointer for directing unprocessed tasks. However, formulation of the gradient surface requires the knowledge of all proximities. Accurate calculation of a proximity requires gate values of all processors, which are not readily available in a distributed environment. In this section, we suggest a distributed measurement, termed *propagated pressure*, to approximate the proximity function.

Definition: The propagated pressure of a processor p_i is defined by the following equation.

$$p_i = \min \{g_i, 1 + \min \{p_j \text{ over } j, \text{ where } d_{i,j} = 1\}\}$$

A lightly loaded processor has propagated pressure of zero. Propagated pressure of a moderate or heavy processor is computed by adding one to the minimum propagated pressure of neighbors. Since $g_i <= w_{\max}$, $w_{\max}$ is also the upper bound of propagated pressures.

Definition: The *pressure surface* (PS) of a network is the collection of propagated pressures of all processors. PS = $[p_1, p_2, p_3, \cdots, p_n]$.

Definition: A pressure surface is *apparently stable* if the last value of each propagated pressure is equal to its newly computed value.

Theorem: When an apparently stable pressure surface is reached, $p_i = w_i$.

Proof: If there is no light node in the system, by definition,

$$w_i = w_{\max} \quad \text{for all } i$$

$$p_i = \min \{w_{\max}, 1 + \min \{w_{\max}, w_{\max}, \cdots\}\}$$

$$= \min \{w_{\max}, 1 + w_{\max}\}$$

$$= w_{\max}$$

$$= w_i.$$

If there exists a light node k, in the system, by definition, $w_k = 0$. If node i is distance n away from node k, $w_j = n$.

case i: $n = 0$, i.e., $k = i$

$$p_i = \min \{0, \text{some positive number}\} = 0$$

$$w_i = \min \{d_{i,i}\} = 0 \quad \text{because } g_i = 0$$

$$p_i = w_i$$

case ii: Assume that node i is distance n away from k and $p_i = w_i$. We try to prove that given a node j at distance $n + 1$, $p_j = w_j$. Without loss of generality, we assume that nodes i and x are immediate neighbors of j (Fig. 2).

$$p_j = \min \{g_j, 1 + \min \{p_i, p_x\}\}$$

$$= \min \{w_{\max}, 1 + \min \{w_i, p_x\}\}$$

$$= 1 + \min \{w_i, p_x\} \quad \text{since } 1 + w_i <= w_{\max}$$

In an apparently stable pressure surface, one of the following three conditions holds:

(1) $p_x = 1 + p_j$;

Since $p_j = 1 + p_i = 1 + w_i$, $p_x = 1 + 1 + w_i$,

hence $p_x > w_i$

(2) $p_x = p_j$;

$p_x = 1 + w_i$; hence $p_x > w_i$

(3) $p_x = p_j - 1$;

$p_x = 1 + w_i - 1$; hence $p_x = w_i$

In any events, $p_x > = w_i$

Therefore, $p_j = 1 + \min \{w_i, p_x\} = 1 + w_i = w_j$

Q.E.D.

The gradient model uses the calculated propagated pressure to approximate the proximity function. Excessive tasks from heavily loaded nodes is routed to the neighbor of the least propagated pressure. There is no ultimate destination assigned to a task when it is moving in the system. The proximate destination of a task is designed such that a localized balancing is easily achieved. Ultimate balancing of the system is accomplished through multiple overlapped local balancing.

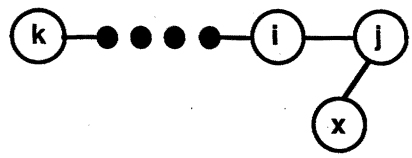


Fig. 2. Neighboring relations among k, i, j, and x.

Such a load migration procedure continues until one of the following conditions is satisfied: 1) the task arrives at the light node, or 2) some other tasks arrive at the light node and close the gate. If there exists some other underutilized node in the system, a new gradient surface is reshaped. The task is then redirected toward the new nearest light processor.

In mathematics terms, the gradient model load balancing scheme is a form of *relaxation*. The single hop migration of tasks is a successive approximation method toward a global balancing of a system.

C. Saturation

After all processors become busy, there is no need to further balance the load because processors have been fully utilized. Any load balancing activity during this period can only increase system overhead and reduce throughput.

Definition: A system is *saturated* if none of the processors are lightly loaded. In other words, a system is saturated if all proximities are equal to $w_{\max}$.

The use of a ceiling value $w_{\max}$ is shown to prevent propagated pressure from becoming unbounded unnecessarily in a fully loaded system. Another use of the $w_{\max}$ is to detect saturation and reduce futile task movements. The saturation state persists until some processor becomes lightly loaded again. The third usage of $w_{\max}$ is in the area of fault isolations. When a processor fails, its neighbors can mandatorily set the pressure of the failed processor to be $w_{\max}$ which stops the moving of new tasks toward the faulty node [13].

D. Algorithm

Based on the gradient model, a distributed load balancing algorithm for each node of a system can be devised.

LOOP

Processor i determines its internal loading state p_i

CASE state **OF**

light:

Set $p_i = 0$.

Ignore the pressure information from neighbors.

moderate:

$p_i = 1 + \min \{p_j\}$ over all neighbor j.

If $p_i > w_{\max}$ **then** $p_i = w_{\max}$

(*saturation*)

heavy:

$p_i = 1 + \min \{p_j\}$ over all neighbor j.

If $p_i > w_{\max}$

then $p_i = w_{\max}$ (*saturation*)

else **if** $\min \{p_j\} < p_i$,

then transfer one task to node j, where p_j is the minimum.

END CASE

Broadcast p_i to all neighbors, if p_i has changed from last update.

END LOOP.

The execution of this algorithm is fully asynchronous and distributed. All processors independently update their pressure by using the most recent information from the neighbors. It should be noted that a light node sets its own pressure to 0 and immediately delivers this information to the neighbors, since 0 is the minimum proximity. This allows the lightly loaded processor to trigger the load migration as soon as possible.

The algorithm is a localized load balancing measure. The dynamics of local balancing represent a step-wise refinement toward a global load balancing. At any instant there are some subregions of a system adjusting their regional gradient distributions and load balancing their work load.

E. Heterogeneous Systems

A heterogeneous multicomputer system may be composed of different processor types, varying processing power, or many kinds of communication links. The gradient model can be enhanced to accommodate this class of heterogeneous systems.

1) Processor Types: When a multicomputer system has more than one type of processors, some tasks may have to be evaluated by certain kind of processors. A group addressing scheme is used to facilitate this type of application.

A task with a group address as the destination is treated as a regular task. When a task is absorbed by a processor, the node verifies the group address with the destination identification. The task is reinjected into the system if the address mismatches. This try and reinject method is costly if the size of a group is relatively small. The approach becomes more attractive as the size of the group grows. One example of using this technique is in a system with interleaving floating point processors. A compute-bound task may designate the group address for floating point processors as the destination. Since the floating point processors are interleaved with regular processors, the overhead of absorbtion and reinjection could be minimal.

Another approach is to formulate multiple gradient surfaces, one surface for each type of processors. Tasks of different types are balanced by different gradient surfaces. In general, multiple load managers within a processor are needed.

2) Processing Power: The first step of the load balancing algorithm is for the processor to assess its own loading condition to be either light, moderate, or heavy. The load state is devised to be a pure internal measure of a processor. As a result, the load computation methods of different nodes need not be identical. A system composed of processors with different processing power may use different criteria for determining a processor state. For example, a node with twice the processing power may require twice as many tasks to become heavily loaded.

Once a node determines its load state, the gradient model makes no distinction among different processors. The decoupling of internal load measurement from network-wide balancing is an essential feature of the gradient model.

3) Communication Link: The proximity function, which computes the minimum "length" between two processors, implicitly assumes identical cost for each communication link. In a system with heterogeneous communication capabilities, the proximity function and the pressure approximation is better served with a cost function.

$$p_i = \min \{g_i, \min \{c_{i,j} + p_j \text{ over } j, \text{ where } d_{i,j} = 1\}\}$$

i.e.,

$$p_i = \min \{g_i, c_{i,j} + p_j \text{ over } j, \text{ where } d_{i,j} = 1\}\}$$

where $c_{i,j}$ is the communication cost between node i and j. Compared to earlier *pressure* definition, it is obvious that the homogeneous communication is a special case where $c_{i,j} = 1$ for all i and j.

IV. An Implementation

The gradient model load balancing scheme has been implemented in the simulator for **Rediflow,** a loosely coupled applicative system proposed by researchers at the University of Utah [10]–[12]. Each processor is closely paired with a memory, and a network of packet switches is used to communicate between these pairs. The combination of a processor–memory pair and a packet switch for information transfer is called an *Xputer*.

A. The Rediflow Simulator

The Rediflow simulator is based on a graph-reduction model of computation and driven by programs written in an applicative language, Function Equation Language (FEL) [9], so named because its expressions are literally equations describing functions and objects.

The simulator permits the specification of various parameters, including the number of Xputers, the amount of memory, the configuration of the Xputers, the switch capacities, the communication bandwidth, and others. The loading status of an Xputer is computed as a function of the backlog of tasks and the amount of memory in use. The internal load measurement is equated to

$$\text{number-of-tasks} + \text{memprs}/(1 - \text{memory-in-use})$$

where memprs may be specified as a simulator parameter. In simulations reported here, memprs is set to 0.01.

In the simulator, an Xputer is light if its internal load measurement falls below a settable low threshold, and heavy if it rises beyond a settable high threshold. The setting of low and high threshold used here is 2 and 3, respectively.

The default size of an APPLY packet is 20 bytes. This

is sufficient to carry the characterizing information of code-block pointer, argument location, and result location. (Longer arguments and results are handled as descriptors to structures.) A DATA packet has 10 bytes and a pressure update packet has only 1 byte. The function code of an APPLY packet may be disseminated to all processors before the execution starts. Otherwise, the code may be transferred along with the APPLY packet and cached in the destination node. Communication delays between two switches is another adjustable parameter. Initially, the communication channel speed was set to 10 Mbits per second for each channel.

B. Simulation Results

The performance of the Rediflow architecture is evaluated using an *introspective* model. The speedups of a multiprocessor system are measured against a single processor with the same technological assumptions, architecture, and evaluation model.

The simulation is event-driven. Messages sent between switches are serviced in time-stamp order. Since we are simulating mostly determinate programs with an invariable number of noncommunication operations and each of a similar duration, so *speedup* can be computed on the fly, as

speedup = total_time_of_operations / simulation_time.

which is equivalent to the reciprocal of the parallel execution time divided by the sequential execution time.

As an example, we use a highly parallel test program which is a purified divide and conquer algorithm, summing the leaves of a binary tree with nodes numbered 1–1024. With the syntax of the functional language FEL, the program DC1024 is shown as follows.

```
{
result DC[1,1024]
DC[m,n] =
        {
        result      if m > = n
                    then m
                    else DC[m,med]+DC[med+1,n]
        med = (m+n) div 2
        }
}
```

The program DC1024 is run on the Rediflow simulator with an increasing number of Xputers. Given a fixed number of Xputers, DC1024 is exercised on several configurations. Different topologies used in the simulation are depicted in Fig. 3. The speedup of the simulation versus the size and topology of the system is shown in Fig. 4.

It is no surprise that wrapped topology performs better than the nonwrapped configuration, since the average distance between any two Xputers in the former is only about half that in the latter. Both the task packets and pressure updates benefit from the shorter communication distance.

The hypercube configuration appears to be the most efficient topology of these alternatives. This is to be expected, since for a given number of nodes, this configuration has the smallest diameter. The smaller the diameter, the faster that saturation can be detected. However, the simple wraparound grid configuration performs reasonably well. This is encouraging, because the grid is a logical choice for wafer scale VLSI implementations.

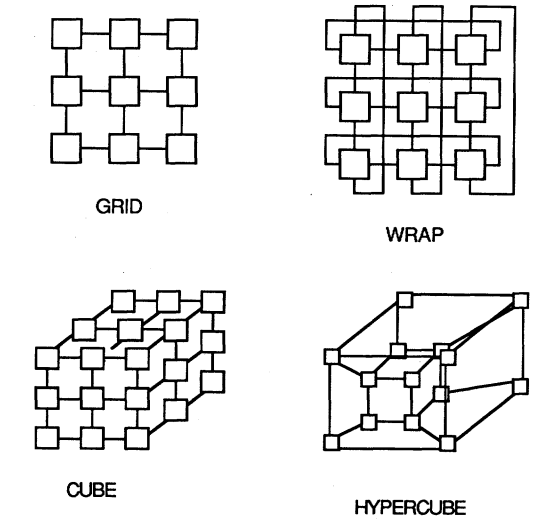


Fig. 3. Topologies used in simulation.

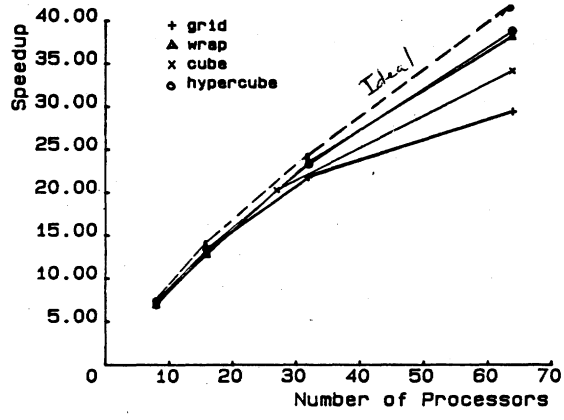


Fig. 4. Speedup of DC1024.

1) Idealized Balancing: To assess the effectiveness of a load balancing scheme, one needs to identify an ideal balancing method and compare it with the given method. A shared-memory model with a centralized scheduling queue would seem to provide the ultimate in load balancing. It must be assumed that communication cost is negligible. An underutilized processor requests further work load from this central facility. For comparison to the load-balancing rule under various configurations, the idealized case is shown in Fig. 4 as the upper dashed curve.

2) Trajectory: Periodic snapshots of the Xputer utilizations shown in Fig. 5 are compiled from DC1024 running with a 4 × 4 wrapped grid configuration. The time interval between samples is 5000 simulation time units. The processor and switch utilizations are shown as interval averages. The switch utilization curve also depicts the breakdown of switch traffic between user data packets and system pressure updating packets. It shows that the up-

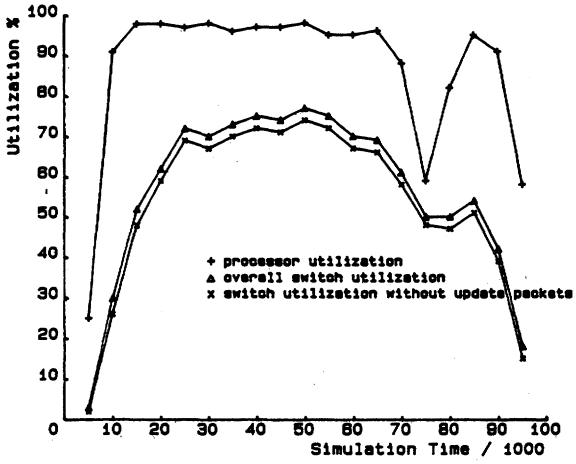


Fig. 5. Trajectory of processor and switch utilizations.

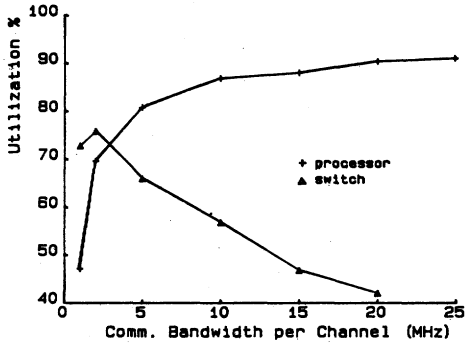


Fig. 6. Utilization of different communication bandwidths.

dating packets use only a fraction of communication bandwidth in this example.

Processor utilization rises fairly rapidly during early stages of the simulation cycle. It indicates the effectiveness of spreading work load over idle Xputers. The slope of the rising curve reflects the eagerness of load spreading. This rate is controllable by adjusting the high and low load thresholds. Note that the processor utilization dips when the simulation time is around 70 000 time unit. The reason is that the system is engaged in a garbage collection operation where memory utilization changes significantly.

3) Communication Speed: The results from the above simulations show that the switch utilization is well within our technology assumption, which assumes each communication channel has a 10 Mbit per second data transfer rate. Since communication overhead is a critical gauge in any multiprocessing system, this section examines the effect of communication bandwidth by changing the speed settings in the Rediflow simulator. The test program is still DC1024 with wrapped 4 × 4 grid configuration. Fig. 6 shows the impact of communication bandwidth on processor and switch utilizations. The speedup of the system is depicted in Fig. 7.

Slow communication channels, e.g., 1 MHz, significantly impede the system throughput, since the processors are only utilized half of the time. The speedup increases as the communication bandwidth improves. However,

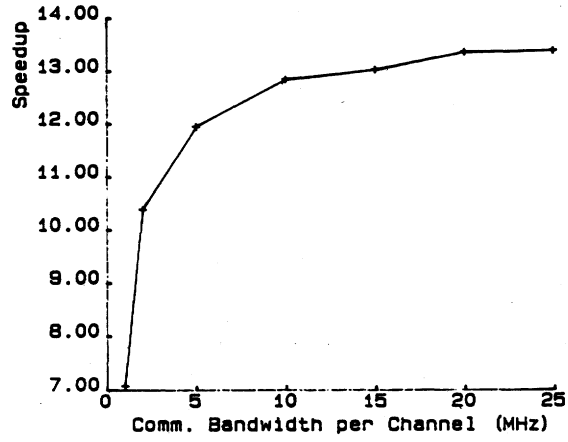


Fig. 7. Speedups of different communication bandwidths.

when the channel capacity exceeds the 10 MHz rate, the system starts to approach the throughput upper bound. This simulation shows that a communication channel speed of 10 MHz seems adequate for this combination of processors, switches, and task granularities.

Similar, although less extensive, simulation studies have been conducted on more practical examples, such as matrix multiplication using quad-trees, histogramming for image processing, logic programming, and *n*-queens search, etc. The DC1024 example used here seems to be representative, in terms of the amount of usable concurrency present, and the resulting system performance.

V. Summary

The load balancing problem is crucial in multiprocessor systems having large numbers of processors and which spawn many concurrent tasks. Any balancing scheme requiring a centralized action seems impractical when the system scales up. Applications with spontaneous task generation also make it difficult to prenegotiate a balanced distribution.

In this study, a distributed load balancing scheme, called the gradient model, is devised. The model is based on a demand-driven principle which requires the under-utilized processors to dynamically initiate load balancing requests. A system-wide gradient surface is formed as a result of these requests. Overloaded processors respond to requests by migrating unevaluated tasks down the gradient surface toward under-utilized processors.

A global balance state is achieved computationally by successive approximation of many localized balances. The concept of saturation is introduced to discourage futile load migration when the system is fully utilized.

The Rediflow simulator, which simulates a proposed applicative system, incorporates the gradient model load balancing mechanism. Various architectural tradeoffs have been studied with the simulation. Simulation studies suggest that the gradient model performs satisfactorily under reasonable technological assumptions.

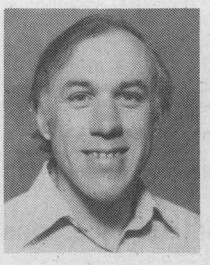
References

[1] F. W. Burton and M. R. Sleep, "Executing functional programs on a virtual tree of processors," in *Proc. 1981 Conf. Functional Pro-*

gramming Languages and Computer Architecture, Oct. 1981, pp. 187–194.
[2] Y. Chow and W. H. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput.*, vol. C-28, pp. 354–361, May 1979.
[3] A. L. Davis and R. M. Keller, "Dataflow program graphs," *Computer*, vol. 15, no. 2, pp. 26–41, Feb. 1982.
[4] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," in *Proc. 2nd Annu. Symp. Comput. Architecture*, IEEE, 1974.
[5] K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *Computer*, vol. 15, no. 6, pp. 50–56, June 1982.
[6] K. P. Gostelow and R. E. Thomas, "Performance of a simulated dataflow computer," *IEEE Trans. Comput.*, vol. C-29, pp. 905–919, Oct. 1980.
[7] K. Hwang *et al.*, "A UNIX-based local computer network with load balancing," *Computer*, vol. 15, pp. 55–64, Apr. 1982.
[8] R. M. Keller, G. Lindstrom, and S. Patil, "A loosely-coupled applicative multi-processing system," in *Proc. AFIPS*, June 1979, pp. 613–622.
[9] R. M. Keller, "Function-equation language programmer's guide," Dep. Comput. Sci., Univ. Utah, AMPS Tech. Memo. 7, Apr. 1982.
[10] R. M. Keller, F. C. H. Lin, and J. Tanaka, "Rediflow multiprocessing," in *Proc. CompCon '84*, IEEE, Feb. 1984, pp. 410–417.
[11] R. M. Keller and F. C. H. Lin, "Simulated performance of a reduction-based multiprocessor," *Computer*, vol. 17, no. 7, pp. 70–82, July 1984.
[12] R. M. Keller, "Rediflow architecture prospectus," UUCS-85-105, Dep. Comput. Sci., Univ. Utah, Tech. Rep. UUCS-85-105, Aug. 1985.
[13] F. C. H. Lin and R. M. Keller, "Distributed recovery in applicative systems," in *Proc. Int. Conf. Parallel Processing*, Aug. 1986.
[14] A. Kratzer and D. Hammerstrom, "A study of load levelling," in *Proc. CompCon*, IEEE, Fall 1980, pp. 647–654.
[15] L. M. Ni, "A distributed load balancing algorithm for point-to-point local computer networks," in *Proc. CompCon*, IEEE, Fall 1982, pp. 116–123.
[16] L. M. Ni, C. W. Xu, and T. B. Gendreau, "Drafting algorithm—A dynamic process migration protocol for distributed systems," in *Proc. 5th Int. Conf. Distributed Comput. Syst.*, IEEE, Denver, CO, May 1985, pp. 539–546.
[17] D. D. Sharp and P. L. Crews, "Work distribution in a bus-structured fully distributed processing system," in *Proc. CompCon*, IEEE, Sept. 1983, pp. 42–49.
[18] J. A. Stankovic and I. S. Sidhu, "An adaptive bidding algorithm for processes, clusters and distributed groups," in *Proc. 4th Int. Conf. Distributed Comput. Syst.*, IEEE, San Francisco, CA, May 1984, pp. 49–59.
[19] J. A. Stankovic, "A perspective on distributed computer systems," *IEEE Trans. Comput.*, vol. C-33, pp. 1102–1115, Dec. 1984.
[20] S. R. Vegdahl, "A survey of proposed architectures for the execution of functional languages," *IEEE Trans. Comput.*, vol. C-33, pp. 1050–1071, Dec. 1984.

Frank C. H. Lin received the B.S.E.E. degree from the National Taiwan University in 1973, the M.S.E.E. degree from Utah State University, Logan, in 1978, and the Ph.D. degree in computer science from the University of Utah, Salt Lake City, in 1985.

He is a Program Manager with ESL Inc., a subsidiary of TRW, in California. He was with CalComp Electronics Inc. from 1975 to 1976. From 1978 to 1986, he was with Sperry Corporation in Salt Lake City where he last served as manager of the Research and Advanced Technology Group. His current research interests are parallel architectures, heterogeneous systems, data flow machines, and fault-tolerant computing.



Robert M. Keller (S'64–M'66) received the B.S. and M.S.E.E. degrees from Washington University, St. Louis, MO, and the Ph.D. degree from the University of California, Berkeley, all in electrical engineering and computer science.

From 1970 to 1976 he was an Assistant Professor of Electrical Engineering at Princeton University. From 1976 to 1986 he was Associate Professor, then Professor, of Computer Science at the University of Utah, Salt Lake City. He has held visiting appointments at Stanford University and Lawrence Livermore National Laboratories. He is currently Director of Research at Quintus Computer Systems, Inc. in Mountain View, CA. His research contributions are in the area of theory of concurrent processing, parallel program verification, parallel computer architecture, and implementation of functional languages. His current research interests deal with numerous topics relating to logic programming and multiprocessing.