Computers Math. Applic. Vol. 26, No. 12, pp. 17–24, 1993
Printed in Great Britain

0898-1221/93 $6.00 + 0.00
Pergamon Press Ltd

Optimal Task Allocation in Hypercube Multiprocessor Ensembles

C. C. PRICE*
Department of Computer Science
Stephen F. Austin State University
Nacogdoches, TX 75962, U.S.A.

M. SALAMA
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109, U.S.A.

(Received October 1992; revised and accepted June 1993)

Abstract—Criteria are established to determine the optimal policy for allocating a set of uniform tasks onto a multiprocessor hypercube ensemble. It is shown that the optimal policy depends on the ratio of computation to intertask communication required by the distributed program, and that based on this ratio, tasks should be placed either all on one processor or uniformly distributed over the largest possible hypercube.

Keywords—Scheduling, Multiprocessors, Computer system design, Task allocation, Intertask communication.

1. INTRODUCTION

In a multiprocessor environment, programs are decomposed into multiple tasks, which in turn must be distributed among the available processors for execution. An optimal task allocation policy may then seek to complete execution of the entire program in the shortest possible total time. As one may expect, however, both the algorithmic requirements of the program, and the specific features of the computer architecture will influence the optimality of the task allocation policy. In this paper, attention is focussed on the optimal allocation of systems of uniform tasks on a class of parallel processors with a hypercube interconnection topology [1]. A system of tasks is said to be uniform if each task requires the same amount of processing resources. As will be seen later, many algorithms in computational mathematics can be decomposed into uniform tasks. The hypercube class of architecture is of special importance because it provides a tractable alternative to the more desirable fully-connected networks. Practicality of the latter diminishes quickly as the number of processors in the system increases.

The general problem of decomposing a given program into multiple tasks has been investigated formally (e.g., [2,3]). However, for the hypercube multiprocessor configuration under consideration, success with the decomposition problem has been attributed to the use of algorithm-specific knowledge, rather than to the use of complex software [4]. But regardless of the means by which a large program has been decomposed, this paper concentrates on the complementary issue of

This paper presents research results carried out in part at the Jet Propulsion Laboratory, under contract with the National Aeronautics and Space Administration (NASA).

*Author to whom all correspondence should be addressed.

Typeset by $\mathcal{AMS}$-TEX

allocating individual tasks to processors. We develop an analytical model for the optimal allocation of uniform task systems on a hypercube-interconnected multiprocessor with the goal of minimizing the maximum schedule length (including execution and communication times) over all processors. And although analytical models cannot reflect all details of real computer systems, nor the time-varying behavior of actual distributed programs, they can nevertheless provide insights and guidelines for the management of distributed computations.

In Section 2 of this paper, we state our task allocation problem and cite relevant research that motivated our interest in this problem. Section 3 contains a brief description of the hypercube multiprocessor structure. Section 4 describes the attributes of the type of uniform tasks under consideration and gives a specific illustrative example. In Section 5, we present our results which specify the allocation of tasks to the available processors such that the total completion time for the task system is minimized. It will be shown that to take appropriate advantage of parallelism, the optimal policy depends on the ratio of computation to intertask communication required by the program, and that based on this ratio, tasks should be placed either all on one processor or uniformly distributed over all available processors in the hypercube. An example for solving a large system of simultaneous equations by Jacobi iterations illustrates the results.

2. THE GENERAL TASK ALLOCATION PROBLEM

In a distributed multiprocessor environment, performance may be measured by the total execution and interprocessor communication times required to complete a program consisting of multiple tasks. In such an environment, one must distinguish between the completion time (or schedule length) of each individual processor comprising the ensemble, and that of the whole ensemble. The first may vary from one processor to another, and is defined as the sum of execution plus communication times of all tasks assigned to the processor in question. However, the ensemble's completion time (or schedule length) is the total execution plus communication times on the processor for which this sum is maximal over all processors. We are interested in determining a static allocation for which the maximum schedule length is minimal over all possible allocations.

The task system under consideration is comprised of m independent tasks to be executed on n processors. The execution time for task i on processor j is denoted by x_{ij}; the amount of communication from task i to task k is c_{ik}; and the cost per data unit sent from processor j to processor r is denoted by d_{jr}. The performance objective is then to minimize the ensemble's schedule length:

$$S = \max_{1 \le j \le n} \left\{ \sum_{i=1}^{m} \sum_{k=1}^{m} \sum_{r=1}^{n} (x_{ij}\, e_{ij} + c_{ik}\, d_{jr}\, e_{ij}\, e_{kr}) \right\}, \tag{1}$$

where $e_{ij} = 1$ whenever task i is assigned to processor j, and $= 0$ otherwise. For systems with identical processors in which communication costs do not occur, i.e., $c_{ik} = 0$, optimal assignments cannot be found efficiently [5]. The only case for which optimal assignments can be determined efficiently is where tasks are uniform. If arbitrary intertask communication costs are introduced, then even the uniform task case becomes NP-hard [5].

The intractability of the general task allocation problem for a fully connected network of processors has been established, but special cases have been investigated by Linneman [6] and Lo [5]. In particular, for uniform tasks and constant intertask communication costs, criteria are given that determine optimal allocations of tasks. Numerous task allocation methods have been based on performance criteria other than the schedule length in (1). Some of these are surveyed in [7]. The special case of two processors is of interest because of its tractability as well as its applicability [8–10]. Optimal allocation policies for a statistical model are investigated in [11] for a random graph program and processors that share a common bus. Our own investigations have led to the study of the hypercube multiprocessor configuration. None of the task allocation schemes reported in the literature are directly applicable to the placement of computational tasks across such an ensemble of processors.

3. THE HYPERCUBE MULTIPROCESSOR ENSEMBLE

Because the schedule length for a system of communicating tasks depends on the time required to transfer data or signals between pairs of processors, any task allocation scheme must take into account the particular topology of the multiprocessor configuration. A successful alternative to the fully-connected network is the hypercube configuration [1]. This arrangement consists of identical processors whose interconnection pattern is illustrated by the 16-processor ensemble in Figure (1). In general, a p-order hypercube consists of $n_p = 2^p$ processors. Any of the n_p processors, say processor j, has p nearest neighbors which are directly connected to it by one link (edge). Of the remaining $(n_p - p - 1)$ neighbors, only one processor is located at the maximum distance p links away. The connection distance to any other neighbor r falls in the range $1 < d_{jr} < p$. Task communication in the hypercube computer occurs via message-passing rather than through shared memory. Each processing node in the hypercube has the same "view" of the network as any other node; and since the link bandwidth can be made to increase as the number of processors increases, such a network has no unique nodes and no bottlenecks. These characteristics make the hypercube an appropriate ensemble for a variety of combinatorial, numerical, and sensory applications, and well-suited to simulating physical processes and natural phenomena.

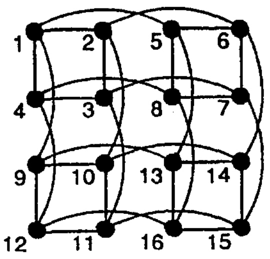

Figure 1. 16-processor hypercube ensemble.

In typical software implementations, the program code and data are partitioned for distribution among processors. Each processor has its own memory, but may also recall data stored in the memory of other processors. The software allocation policy developed herein specifies the conditions under which the cost of interprocessor communication outweighs the advantage of multiprocessing.

4. UNIFORM TASKS

The task system under consideration consists of m uniform tasks; each requiring the same execution time $x_{ij} = x$ for a typical task i on a typical processor j, and each requiring the same amount of communication $c_{ik} = c$ between any pair of tasks i and k. Variations in communication costs arise only due to variation in distances between the pairs of processors in which the tasks reside. Furthermore, there are no precedence relationships between tasks.

Uniform tasks having these attributes are special cases of the more general non-uniform tasks. Nevertheless, they arise in a number of practical problems; notably in iterative numerical techniques. Examples include the solution of simultaneous equations by Jacobi and conjugate gradient, the extraction of eigenpairs by iterations, and the solution of many physics and engineering problems by relaxation [4]. To illustrate, consider the solution of a large system of N simultaneous equations:

$$A_{ij}U_j = B_i, \qquad i,j = 1,\dots,N. \tag{2}$$

Alternatively, equation (2) may be partitioned into m blocks: a_{ij}, u_j, and b_i of sizes $(N/m) * (N/m)$, $(N/m) * 1$, and $(N/m) * 1$, respectively. The choice of the partition size $(N/m) \geq 1$ is usually dictated by the memory capacity of the individual processors to which the computational tasks are to be distributed. The partitioned equations can be solved by a sequence of Jacobi iterations. During a typical $(r+1)$ iteration, one computes an improved solution u_i^{r+1} from:

$$u_i^{r+1} = a_{ii}^{-1} b_i - \sum_{j=1, j\neq i}^{m} a_{ii}^{-1} a_{ij} u_j^r, \qquad i = 1, \ldots, m. \tag{3}$$

Performing the matrix operations in equation (3) on a typical i^{th} row partition during a typical $(r+1)$ iteration constitutes a uniform task with no precedence constraints. Each task consists of two successive steps: (a) computing u_i^{r+1} from (3), then (b) sending u_i^{r+1} to all other processors and receiving all other u_q^{r+1}, $i \neq q$ from other processors. Within a given iteration, the resources required to update u_i^{r+1} for any row partition i are equal, and no precedence exists among row partitions. Therefore, it is possible to perform these steps on row partition i on processor k in parallel with performing them on row partition q on processor ℓ. Notice that step (b) makes the data needed to perform equation (3) for the next iteration available to all processors. Once steps (a) and (b) are completed for iteration $(r+1)$, the iteration index is advanced by one, and all processors repeat the same steps for iteration $(r+2)$ on the same row partitions they were initially assigned. While some matrix operations will need to be performed only once at the first iteration, subsequent typical iterations on the ith partition involve (N^2/m) floating point operations (each consisting of a multiplication and an addition), and the communication of (N/m) numbers to all other processors. In the current state-of-the-art hypercube hardware, the ratio μ of a floating point operation execution rate to the communication of a floating point number between two nearest neighbor processors falls in the range $\mu \simeq 0.1$ to 2.0, depending on the packet length. Thus, in the specific example of equation (3), $x = (N^2/m)$, $c = (N/m\mu)$, and $(x/c) = \mu N$. Other algorithms are characterized by different (x/c) values.

5. ALLOCATION OF UNIFORM TASKS IN THE HYPERCUBE

Without loss of generality, let $m = k * n$ for some integer $k \geq 1$, since if this is not true, the system performance will be that of m' tasks, where m' is the smallest integer for which $m' = k * n > m$ for some k. If all m tasks are distributed uniformly among $n = n_p = 2^p$ processors, then the schedule length S_p can be written as:

$$S_p = \left(\frac{m}{n_p}\right) x + K_p \left(\frac{m}{n_p}\right)^2 c, \tag{4}$$

where

$$K_p = \sum_{j=1}^{n_p} d_{ij} \quad \text{or,} \quad K_p = \frac{p}{2} n_p. \tag{5}$$

For example, for the second order hypercube, $p = 2$, $n_p = 4$, $K_p = 4$, and we have the uniform allocation $S_2 = (m/4)x + (m^2/4)c$. In order to determine an optimal allocation, consider the schedule length $S_0 = mx$, which would result if all m tasks were placed on one processor (i.e., a zero order hypercube). Clearly if the communication cost c is high enough, it may be worthwhile to forego the apparent advantages of multiple processing. Specifically, the tasks should not be distributed when $S_0 \leq S_p$, or when:

$$(x/c) \leq F_p \quad \text{where} \quad F_p = K_p \left(\frac{m}{n_p^2 - n_p}\right). \tag{6}$$

On the other hand, an n_p processor ensemble has the advantage over a single processor when (x/c) exceeds the factor F_p. This is illustrated in Figure (2) for a few small order hypercubes, where it is shown that $S_3 < S_2 < S_1$ for all (x/c) ratios. Furthermore, for certain ranges of (x/c) values, S_0 is actually the smaller schedule length. The "break even" points between S_0 and S_p, $p = 1, 2, 3$, are identified in the figure as F_1,F_2, and F_3, respectively. Thus, (x/c) determines the relative merit of S_0 and S_p for a given $n_p = 2^p$. However, in order to establish an optimal allocation policy, we must show that no intermediate or nonuniform allocation can be better than S_0 or S_p. In particular, it remains to be shown that: (a) for any n_p, the use of all n_p processors uniformly is better than using any embedded sub-hypercube uniformly, and (b) for any given number of processors, a uniform (or balanced) allocation is better than any nonuniform allocation. Such an optimal policy is defined by the following two theorems.

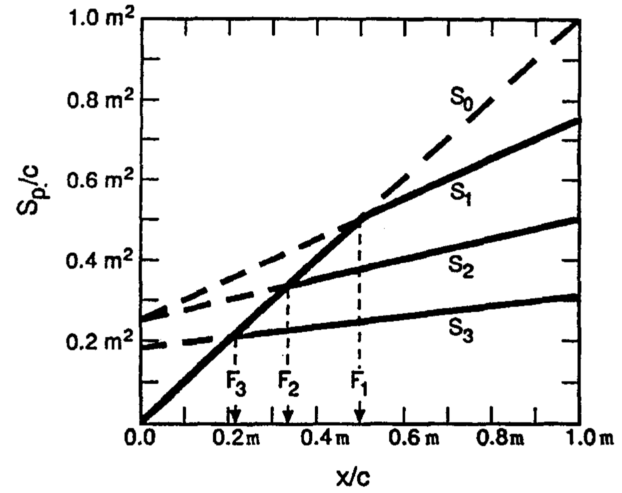

Figure 2. An illustration of the relative merits of single and multiple processors.

THEOREM 1. *For a uniform task allocation and for all (x/c), the schedule length S_p on a p-order hypercube is less than or equal to the schedule length S_{p-1} on an embedded sub-hypercube of order $(p-1)$; i.e., $S_p \leq S_{p-1}$, for $p \geq 2$.*

PROOF. The increase in the schedule length resulting from a uniform allocation of a given number of m tasks on a hypercube whose order has been reduced from p to $(p-1)$, where $p \geq 2$, is $\Delta S_p = S_{p-1} - S_p$. From equations (4,5) , it follows that:

$$\Delta S_p = \left(\frac{m}{2^p}\right)\left[x + \left(\frac{p}{2} - 1\right) mc\right].$$

For $p \geq 2$, both coefficients of x and c in the above equation are always positive (or zero) regardless of the relative magnitudes of x and c. Thus, it is always true that $S_1 > S_2 > S_3 \cdots$. ∎

Thus, the optimal hypercube for uniform distribution of tasks is either the zero-order hypercube or the largest p-order hypercube, depending on whether (x/c) is less or greater than F_p. As an illustration of the above conclusion, consider the following two cases on various orders of hypercubes. Let $m = 32$, $c = \frac{1}{32}$, and $x = 1$, so that $(x/c) = 32$. For $m = 32$ tasks, the largest order hypercube is $p = 5$, and $F_p = 2.58 < (x/c)$. For $p = 0, 1, \ldots, 5$, we find: $S_0 = 32$, $S_1 = 24$, $S_2 = 16$, $S_3 = 10$, $S_4 = 6$, and $S_5 = 3.5$. Clearly, in this example the shortest schedule results from the largest order hypercube. On the other hand, if $m = 32$, $c = \frac{1}{2}$, and $x = 1$, then $F_p = 2.58 > (x/c) = 2$. Note here that for $p \geq 2$, the schedule length decreases for successively higher order hypercubes, but is never less than for $p = 0$. Thus, $S_1 = 144$, $S_2 = 136$, $S_3 = 100$, $S_4 = 66$, and $S_5 = 41$, but $S_0 = 32$.

The second theorem establishes that for any fixed number of n_p processors, the shortest possible schedule results from a uniform distribution of tasks. Notice for example that a variation on the uniform allocation, say for $n_p = 4$, occurs by moving one task from one processor (e.g., Processor 3) to another processor (e.g., Processor 1). The resulting schedule, S_2', which is governed by Processor 1 is now greater than S_2, and is given by: $S_2' = (\frac{m}{n_p}+1)x+\left[4(\frac{m}{n_p})^2+2(\frac{m}{n_p})-2\right]c$. For Processors 2 and 4, the schedule length remains unchanged. The processor with the longest task execution schedule is also the one that incurs the greatest communication cost, and is therefore the one that determines the schedule length for the entire task system. This is in fact just an illustration of the following more general theorem.

THEOREM 2. *If all n_p processors within a p-order hypercube are to be utilized by an m task system, then the schedule resulting from uniform distribution of the tasks is shorter than or equal to that associated with any non-uniform allocation.*

PROOF. Since no node is unique in a hypercube ensemble, one may arbitrarily select any node, j^*, to be the most heavily loaded processor in a non-uniform allocation policy. Let $b = (m/n_p)$ be the number of tasks in a uniform allocation, and let t be the number of extra tasks on Processor j^*. Recall that the greatest distance d_{j^*r} between Processor j^* and any of the remaining $(n_p - 1)$ processors is equal to the hypercube order p, and that there is only one processor located at this p distance, while all other $(n_p - 2)$ processors are located at distances $1 \le d_{j^*r} < p$. In the best possible non-uniform allocation scenario, all extra $t \le b$ tasks on Processor j^* are taken from the processor at the greatest distance $d_{j^*r} = p$ from j^*, with the hope of decreasing the communication costs over that distance. The schedule length for this non-uniform allocation is:

$$S^* = (b+t)[x + K_p bc - ptc].$$

Comparing the schedule lengths S_p and S^* for the uniform and non-uniform allocations, the inequality $S_p < S^*$ holds whenever

$$\left(\frac{x}{c}\right) + K_p b - p(b+t) > 0.$$

The ratio (x/c) is positive. The term $[K_p b - p(b+t)] = [(K_p - p)b - pt]$ is also positive when $t \le b$, since from equation (5): $(K_p - p) \ge p$ for $p \ge 2$. It is straightforward to show that any allocation in which the $t \le b$ extra tasks do not all come from the maximum distance p results in a schedule length that is greater than S^*. In general, if k tasks are moved to Processor j^* from the processor at distance p, and $(t-k)$ tasks are moved from processors at distances less than p, then the shortest schedule is achieved when $k = t$ and monotonically becomes worse as k decreases.

If $t > b$, then the best possible non-uniform allocation scenario dictates that the first b of the t extra tasks on Processor j^* be taken from the processor at distance p, and that the remaining $(t-b)$ tasks be taken from processors at a distance less than p. In this case, the schedule length S^{**} of Processor j^* will be

$$S^{**} = (b+t)\left[x + K_p bc - pbc - (p-1)(t-b)c\right].$$

The inequality $S_p < S^{**}$ holds whenever

$$\left(\frac{x}{c}\right) > \left[(t-b)\left(p-1-\frac{b}{t}\right) - \left(\frac{2^p}{2}-2\right)pb\right],$$

and this can be shown to hold when $(x/c) > F_p$. Otherwise, the optimal policy is to allocate all tasks to one processor according to Theorem 1. Thus, as with $t \le b$, the best possible non-uniform allocation of $t > b$ tasks yields a worse schedule length than the uniform allocation.

As t becomes larger, the relative advantage of S_p over S^{**} diminishes but remains positive; the reason being that as more processors are vacated, there is a clustering of tasks on or near

Processor j^*. The clustering tends to reduce the communication time but extends the execution time on Processor j^* and its nearest neighbors. However, since $(x/c) > F_p$, we know that S_0 is not optimal, so we must not place all tasks on Processor j^*. Also we must not uniformly allocate the tasks to embedded sub-hypercubes, since Theorem 1 has established that $S_p < S_{p-1}$ for uniform allocations. ∎

An illustration of how one might use the results derived herein for an appropriate distribution of computational tasks in the hypercube environment can be provided by revisiting the example in Section 4. Assuming a third order hypercube (i.e., $n_p = 8$), equation (6) gives $F_3 = .2143m$, where m = number of uniform tasks. Considering the solution of $N = 2048$ simultaneous equations by the Jacobi scheme of equation (3), we have $(x/c) = \mu N$, which ranges from $0.1N = 204.8$ to $2N = 4096$. Two extreme options are now examined:

Option 1: Divide the 2048 equations into 8 equal partitions, each consisting of 256 equations. Thus $m = 8$, and $F_3 = 1.714$.

Option 2: Divide the 2048 equations into 1024 equal partitions, each consisting of 2 equations. Thus $m = 1024$, and $F_3 = 219.443$.

In the first option, $(x/c) > F_3$, and all 8 processors should be used. However, with the second option, if the packet length is short, then $(x/c) \simeq 204.8 < F_3$, and a single processor would be more advantageous to use over the eight processors.

6. SUMMARY

In this paper, attention was focussed on the optimal assignment of m uniform tasks on n_p identical processors configured in a p-order hypercube ensemble. We have established that whenever the ratio of execution to communication (x/c) does not exceed the factor F_p, then communication costs dominate execution costs to the extent that all tasks should execute serially on a single processor to avoid communication costs and therefore achieve minimal schedule length. Otherwise, these tasks should be allocated uniformly on the largest possible hypercube. These two extremes were shown to yield shorter schedule lengths than any other non-uniform allocation. It is interesting to note that for the random graph model of [11], conclusions similar to those stated above were obtained based on statistical arguments although they differed in locating the point of optimality.

Optimal allocation policies have not been established for more general task systems with heterogeneous intertask communication patterns, or for task systems with precedence constraints. Heuristic algorithms for scheduling such tasks on the hypercube have been developed and tested on a variety of problems [12]. However, much work remains to be done in order to fully characterize the behavior of general distributed programs and the appropriate utilization of processing power within multiprocessor computing systems.

REFERENCES

1. C.L. Seitz, The cosmic cube, *Comm. ACM* **28** (1), 22–23 (January 1985).
2. B. Lint and T. Agerwala, Communication issues in the design and analysis of parallel algorithms, *IEEE Trans. Software Engr.* **SE-7** (2), 174–188 (March 1981).
3. M.R. Paige, On partitioning program graphs, *IEEE Trans. Software Engr.* **SE-3** (6), 386–393 (November 1977).
4. G.C. Fox, Using the Caltech hypercube, *IEEE Software*, 73 (July 1985).
5. V.M. Lo, Task assignment to minimize completion time, In *IEEE Proc., Fifth International Conf. on Distributed Computing Systems*, pp. 329–336, (May 1985).
6. V. Linneman, *Deterministic Processor Scheduling with Communication Costs*, Fachbereich Informatik, Universitat Frankfurt, Germany, (January 1984).

7. C.C. Price, Task allocation in distributed systems: A survey of practical strategies, In *Proc. ACM'82 Conf.*, pp. 176–181, (October 1982).
8. K. Fuchs and D. Kafura, Memory-constrained task scheduling on a network of dual processors, *J. of ACM* **32** (1), 102–129 (January 1985).
9. G.S. Rao, H.S. Stone and T.C. Hu, Assignment of tasks in a distributed processor system with limited memory, *IEEE Trans. Computers* **C-28** (4), 291–299 (April 1979).
10. H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, *IEEE Trans. Software Engr.* **SE-3** (1), 85–93 (January 1977).
11. B. Indurkhya, H.S. Stone and L. Xi-Cheng, Optimal partitioning of randomly generated distributed programs, *IEEE Trans. Software Engr.* **SE-12** (3), 483–495 (March 1986).
12. C.C. Price and M. Salama, Scheduling of precedence-constrained tasks on multiprocessors, *The Computer Journal* **33** (3), 219–229 (1990).