Algorithms For Static Task Assignment And Symmetric Contraction In Distributed Computing Systems

Virginia M. Lo
lo@cs.uoregon.edu

CIS-TR-88-06
May 5, 1988

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE
UNIVERSITY OF OREGON

Abstract

In this paper, we look at the mapping problem, which was posed within the domain of parallel processing, and we redefine that problem for use in distributed computing systems whose underlying communication medium is a broadcast medium such as ethernet. We describe an efficient algorithm which can be utilized to find optimal assignments of tasks to processors for a wide variety of distributed algorithms when *symmetric contraction* of the algorithm is necessary. We also describe a heuristic algorithm for use in finding suboptimal assignments of tasks to processors for arbitrary distributed computations. Both algorithms model the mapping problem as the Graph Partitioning Problem. Our algorithms utilize an efficient algorithm for finding maximum weight matchings to find an assignment of tasks to processors which minimizes the total interprocessor communication cost while meeting a constraint on the number of tasks assigned to each processor.

1. Introduction

Researchers in the area of distributed computing are actively seeking ways to take advantage of the potential of these systems for parallel computing. The last few years have seen intensification of efforts in the design of large-grained, loosely coupled parallel algorithms and in the design of systems software (operating systems, compilers) which support parallel computation. One problem that occurs in both distributed and parallel systems is the problem of assigning tasks in a distributed (parallel) computation to the processors in a distributed (parallel) system. This problem has been referred to as the **static task assignment problem** and also as the **mapping problem**, with the former appellation more commonly used in the distributed computing community and the latter terminology more commonly used in the parallel processing community. For both types of systems, the goals of task assignment include reducing interprocessor communication, load balancing, and parallelism; indeed, many approaches to the solution of this problem can be applied successfully to either type of system. However, there are differences in the architectures of distributed systems and parallel processors that affect the design of task assignment algorithms.

In this paper, we look at the mapping problem, which was posed within the domain of parallel processing, and we define an analogous problem for distributed computing systems whose underlying communication medium is a broadcast medium such as ethernet. We demonstrate that in such a system, an appropriate goal for task assignment algorithms is minimization of the total interprocessor communication costs while meeting a constraint on the number of tasks assigned to each processor. We describe an efficient algorithm which can be utilized to find optimal assignments of tasks to processors for a wide variety of distributed algorithms when *symmetric contraction* of the algorithm is necessary. We also describe a heuristic algorithm for use in finding suboptimal assignments of tasks to processors for arbitrary distributed computations. Both algorithms model the task assignment problem as the Graph Partitioning Problem. Our algorithms utilize an efficient algorithm for finding maximum weight matchings to find an assignment of tasks to processors.

We consider only distributed systems which consist of a collection of homogeneous multiple computers, each with local CPU, memory, and other hardware resources, and which communicate through an ethernet bus. A distributed computation or task force consists of a set of communicating tasks to be assigned to processors in the distributed system. We utilize the graph theoretic model [16] for these computations in which each task is modeled as a node in the graph and communicating tasks are connected by an edge whose weight equals the communication cost incurred if the tasks are assigned to different processors. We assume that communication between tasks assigned to the same processor is negligible.

In section 2 we describe the problem in more detail and discuss its relation to the contraction problem from parallel processing. In section 3 we present our algorithms as well as simulation results for the heuristic Algorithm H. The last section contains conclusions and a discussion of further work in this area.

2. Static Task Assignment and Contraction in Ethernet-based Distributed Systems

The general problem we wish to address is the assignment of tasks to processors in order to minimize interprocessor communication while meeting constraints on the number of tasks assigned to each processor. This statement of the task assignment problem [1] is useful in two slightly different contexts: (1) for initial assignment of tasks to processors (static task assignment) and (2) for assignment of tasks to fewer processors than the distributed algorithm was initially designed for, either at the time of initial assignment or dynamically at execution time due to node failure, node withdrawal, or phase transitions in the distributed computation. Node withdrawal occurs in distributed systems comprised of a network of personal workstations. At any time, one of the workstations may become unavailable to the distributed computation, either because the 'owner' withdraws it from the pool or because the load on that workstation becomes heavy.

These problems are similar to the mapping problem posed by [2] for parallel processoring systems. In parallel processors, the point-to-point nature of the communication network offers a multiplicity of interconnection options - hypercube, meshes, shuffle-exchange networks, cube-connected cycles, trees, etc. and the potential for parallel (non-interfering) communication on disjoint paths through the network. As a result, the mapping problem involves two phases, ***contraction*** and ***layout.*** Contraction involves reducing the graph G which represents the parallel algorithm to a smaller graph G' in the same family, causing several task nodes in G to become associated with one node in the reduced graph G'. (Thus, if G is a tree, it must be contracted to a smaller tree G'.) In the layout phase, the reduced graph is then mapped to the interconnection hardware, with at most one task node per processor, taking into account the mapping of edges in the computation graph to edges in the processor network [2]. At execution time, the tasks assigned to a given processor are multiplexed on that processor. In an ethernet-based system, there is only one communication pathway, and all interprocessor communication competes for the same resource. As a result, the layout phase is not relevant for distributed systems (i.e., it is not necessary to map computation edges to network edges) nor is it necessary to maintain any specific interconnection structure in the contracted graph.

In addition, the criteria evaluating contractions in a distributed system is different than that proposed for parallel processing systems. In the latter systems, evaluation of contraction algorithms utilizes one or more of the following metrics: [2]

- the average number of tasks and edges from G absorbed into one node of G',
- the maximum number of tasks and edges from G absorbed into one node of G',
- the average number of edges of G mapped to an edge in G',
- the maximum number of edges of G mapped to an edge in G'.

These objective functions, particularly the latter two objective functions, are useful when more than one communication link exists and communication can occur independently and in parallel on these links. In ethernet systems, a more appropriate metric with respect to communication is the total sum of interprocessor communication costs incurred by an assignment. By minimizing this quantity, the overall overhead of IPC and also the contention for ethernet is minimized thereby improving the response time of the distributed computation. However, it is a well-known fact that

[1] Several different optimality criteria have been studied in the graph theoretic approach to static task assignment including minimization of the total sum of execution and communication costs [11, 13, 15, 16], minimization of IPC followed by load leveling [5], and sum-bottleneck optimization [15].

[2] These performance metrics are based on the assumption that $c_{ij} = 1$ for all i, j; these metrics can be extended in a natural way for arbitrary c_{ij}.

minimization of IPC conflicts with the goals of load balancing and parallelism. We compensate for these needs by constraining the number of tasks assignable to each processor. When many tasks are assigned to one processor they contend for the resources of that processor and incur overhead due to process switching, management of shared buffers, etc. Bounding the number of tasks per processor contributes to load balancing by limiting contention of this nature.

Thus, in an ethernet-based distributed system, task assignment requires contraction of the computation task graph to a reduced graph in order to minimize the total interprocessor communication costs while maintaining a bound on the number of tasks assigned to each processor. Henceforth, we restrict our discussion to the context of ethernet-based distributed systems and we will use the terms *task assignment, mapping,* and *contraction* interchangeably.

Because we seek different goals for contraction in distributed systems, contraction techniques devised by parallel processing researchers may not be appropriate for our problem. For example, Berman's technique of *truncation* for complete binary trees yields an assignment with total IPC cost of 12 whereas an optimal assignment according to our criteria has cost equal to 3 (see Figure 1 below).

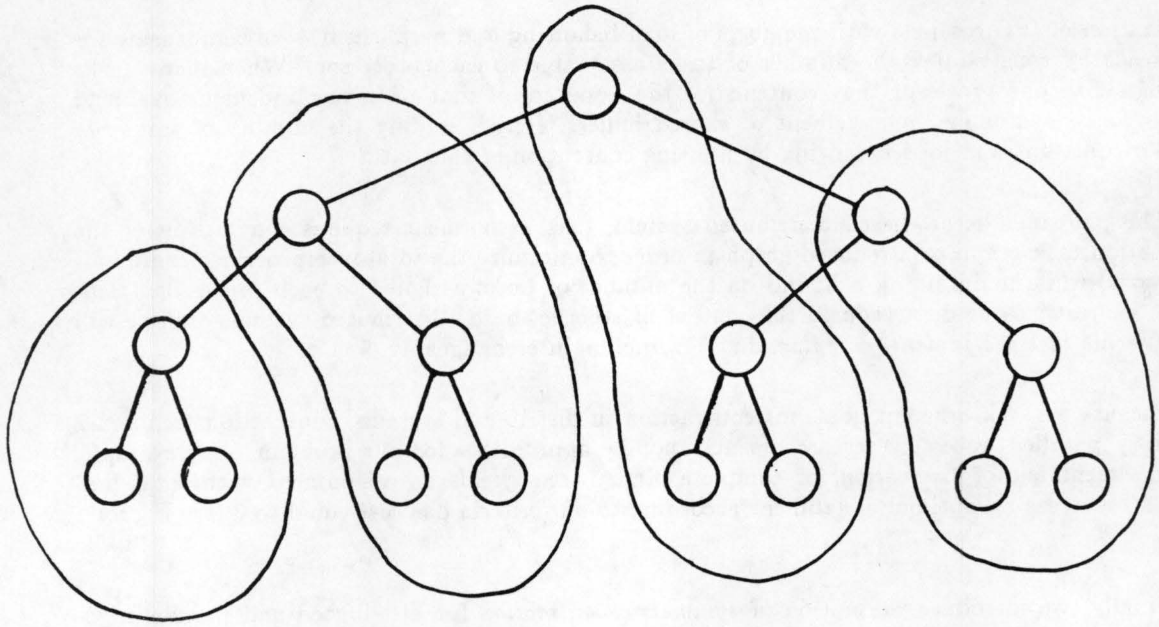
Finally, we introduce the notion of *symmetric contraction* for distributed and parallel algorithms which are regular in structure. These algorithms consist of a number of **identical** tasks that operate on data that has been partitioned among the tasks. When contraction is performed, it is necessary to merge tasks in a symmetric fashion in order to preserve parallelism. More specifically, if a distributed algorithm consisting of k identical tasks is to be contracted for assignment to fewer than k processors, it is necessary to contract an equal number of tasks to each processor. Because the tasks on a given processor are multiplexed, an unequal contraction is undesirable because it constrains all processors to the speed of the slowest processor in the system. This necessity for symmetric contraction has been noted in [14] and in our own survey of distributed and parallel algorithms. We shall see that this fact provides compelling motivation for the use of Algorithm M (described below) to find optimal contractions of distributed computations.

3. Description of the Algorithms

We have developed two algorithms for the assignment of tasks to processors whose goals is minimization of total IPC under a common constraint on the number of tasks per processors. Algorithm M finds optimal assignments in polynomial time for a restricted group of distributed computations. Algorithm M is ideally suited for contraction of computations with the regular structure described above and for the assignment of "small" computations in a "big" distributed system. We illustrate its application to a distributed algorithm for the simplex method of linear programming. Algorithm H is an efficient heuristic which finds possibly suboptimal assignments for arbitrary distributed computations. Simulation results show the performance of this algorithm to be good, yielding an optimal assignment in 81.1% of the cases simulated.

3.1. Equivalence to the Graph Partitioning Problem

We restrict our attention to homogeneous systems with n identical processors. Let $P = \{p_1, p_2, ..., p_n\}$ be the set of n processors, $T = \{t_1, t_2, ..., t_k\}$ be a set of k communicating tasks (i.e., a distributed computation) to be assigned to the processors. Let c_{ij} be the cost of communication between tasks t_i and t_j if they are assigned to different processors. Interprocess communication cost is assumed to be negligible when communicating tasks are assigned to the same processor. Let B, $\lceil \frac{k}{n} \rceil \leq B \leq k$, be a common bound on the maximum number of tasks allowed


(a) Contraction with IPC = 3

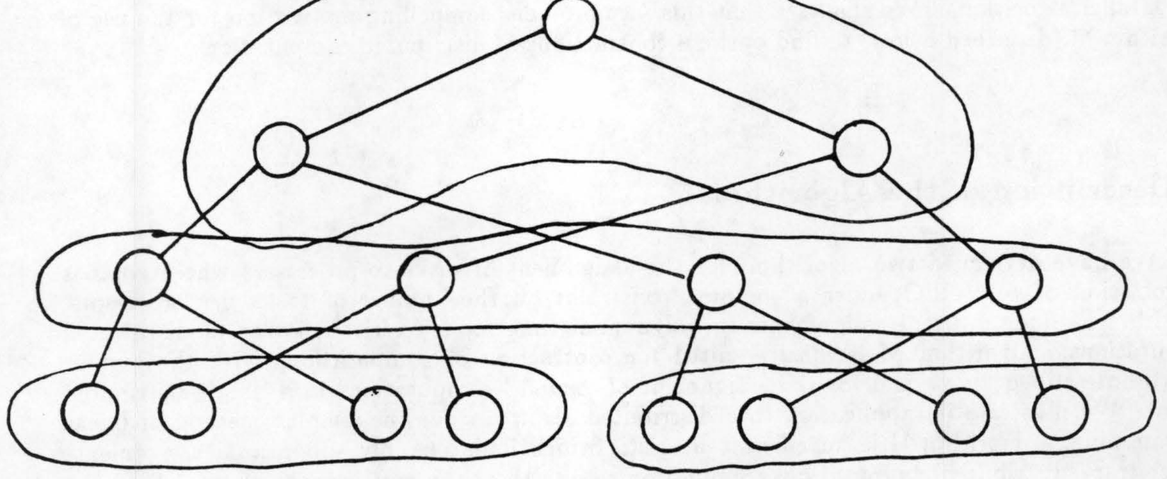

(b) Berman-Snyder Contraction - IPC = 12

Figure 1: Contraction to Minimize IPC *versus* Berman-Snyder Contraction
(For simplicity, all edges are assumed to have $c_{ij} = 1$)

on each processor. We define an **optimal assignment** as one which minimizes the total interprocessor communication costs incurred under the constraint that $k_q \leq B$ for all processors p_q, $1 \leq q \leq n$, where k_q is the number of tasks assigned to processor p_q.

The task-processor system described above can be modeled as a graph $G = (V,E)$ in which each task is represented as a vertex in V. An edge is constructed for each pair of communicating tasks and given a weight equal to the communication cost c_{ij}. The problem of finding an assignment of tasks to processors which minimizes IPC under a constraint on the number of tasks per processors is equivalent to the Graph Partitioning Problem with all node weights equal to one.

Graph Partitioning Problem: Given Graph $G = (V,E)$, weights $w(v)$ for each $v \in V$ and $l(e)$ for each $e \in E$, and positive integers B and J, find a partition of V into disjoint sets $V_1, V_2, \cdots V_n$ such that $\sum_{v \in V_i} w(v) \leq B$ for $1 \leq i \leq n$ and such that if E' contained in E is the set of edges that have their two endpoints in two different sets V_i, then $\sum_{e \in E'} l(e) \leq J$.

The Graph Partitioning Problem and the Graph Partitioning Problem with all vertex weights equal to one have been shown to be *NP-complete* [8]. Thus, our task assignment problem is also *NP-complete*.

3.2. Algorithm M, An Optimal Algorithm for Task Assignment

Algorithm M can be used to find optimal assignments in polynomial time when the number of tasks is less than or equal to twice the number of processors and when each processor may be assigned at most two tasks. These constraints may sound rather limiting at first, but we show that there exist many distributed computations for which these constraints hold. Algorithm M utilizes a polynomial time algorithm for finding a *maximum weight matching* in graphs. An algorithm of complexity $O(ke \log k)$ where e is the number of edges and k the number of nodes in the network is described in [7].

We first prove that for systems in which the number of tasks is less than or equal to twice the number of processors and in which each processor may be assigned at most two tasks, an optimal solution can be found in polynomial time. This proof involves two parts: (a) construction of a maximal matching in a graph corresponding to the task assignment problem, and (b) proof that a maximal matching yields an assignment which minimizes IPC while meeting the constraint of at most two tasks per processor.

Theorem 1: Consider a system with n identical processors and with the number of tasks $k \leq 2n$. Let $B = 2$ be the maximum number of tasks allowed on each processor. Then an assignment which minimizes total interprocessor communication costs under the constraint of at most 2 tasks per processor can be found in polynomial time.

Proof:

(a): Construct a graph G with a node representing each task and an edge between each pair of task nodes t_i and t_j with weight c_{ij}. We note from graph theory [9] that a *matching* in such a graph is a set of edges in which no two edges have a node in common and the *weight of the matching* is the sum of the weights of those edges that are in the matching. Furthermore, a *maximum weight matching* for G is one whose weight is maximum among all matchings for G.

A matching can be used to define an assignment of tasks to processors with at most two tasks per processor as follows:

(1) Let each pair of tasks t_i and t_j connected by an edge e in the matching be assigned to a distinct processor p_q.

(2) If there exist tasks not connected by an edge in the matching, arbitrarily arrange them in pairs and assign these pairs to distinct processors p_q such that no other tasks are assigned to p_q.

(3) If a single task remains unassigned, assign it to any processor p_q such that no other tasks are assigned to p_q.

Since the number of tasks $k \leq 2n$, there will be a sufficient number of processors to perform the above assignment. Because of the way the assignment is made, no processor will be assigned more than two tasks.

(b): We now prove that a maximum weight matching corresponds to an assignment which minimizes the total interprocessor communication costs under the constraint that no processor is assigned more than two tasks. Let f be an assignment of tasks to processors and let

$$C_f = \sum_{f(t_i) \neq f(t_j)} c_{ij} \text{ and}$$

$$\overline{C}_f = \sum_{f(t_i) = f(t_j)} c_{ij}$$

In other words, C_f is the total IPC incurred by assignment f, and $\overline{C}_f$ is the sum of communication costs between tasks assigned to the same processor. Then

$$C_{TOT} = \sum_{1 \leq i,j \leq k} c_{ij} = C_f + \overline{C}_f$$

Since C_{TOT} (the grand total of all communication costs on all edges in the graph) is fixed over all assignments, an assignment which minimizes C_f also maximizes $\overline{C}_f$.

Now consider a maximum weight matching for G and construct an assignment f from the matching as described above. We will show that the weight of the maximum weight matching is precisely equal to $\overline{C}_f$, the sum of the communication costs on edges between tasks assigned to the same processor. Each edge in the maximum weight matching has a weight c_{ij} which contributes to the sum $\overline{C}_f$ since the two tasks t_i and t_j are assigned to the same processor by step (1). Each pair of tasks selected by step (2) above has a weight c_{ij} which equals 0. (If not, that edge could be added to the maximum weight matching to produce another matching with greater weight.) Thus $\overline{C}_f$ is precisely equal to the weight of the matching. It follows then that an assignment defined by a maximum weight matching for G maximizes $\overline{C}_f$ and thus minimizes C_f, the communication costs incurred by assignment f. As discussed above, an assignment which minimizes C_f is optimal.

As stated above, maximum weight matchings and thus optimal assignments can be found in polynomial time. **Q.E.D.**

Algorithm M:

- Construct a matching in G using a polynomial time algorithm for finding maximum weight matchings.
- Construct an assignment according to the steps described in part (a) of Theorem 1.

Algorithm M is well-suited for the problem of dynamic contraction in ethernet-based distributed systems for regular distributed computations. As discussed earlier, parallelism is maintained in regular distributed computations by contracting an equal number of tasks to each processor. For many distributed algorithms, contraction which assigns two tasks per processor is acceptable and even desirable. In these cases, the number of tasks is precisely equal to twice the number of processors, and Algorithm M can be used to find an optimal contraction by setting $B = 2$. Figure 2 illustrates the contraction of a regular distributed algorithm for the simplex method of linear programming. This algorithm was developed by [6] for execution on the Charlotte Distributed Operating System which consists of 20 Vax 11/750. Other regular algorithms for which contraction to $\frac{k}{2}$ processors is useful include Jacobi iterative method for solving LaPlace equations on a rectangle, successive over-relaxation iterative method for solution of linear systems of equations, Nelson's version of Horowitz and Zorat's matrix multiplcation algorithm. These algorithms all appear in [14].

We also note that many existing distributed systems, such as those in use at academic and research institutions, consist of 40-50 nodes. Algorithm M can thus be used for the optimal assignment of distributed computations consisting of up to twice that many tasks. We claim (but do not substantiate now) that there are a significant number of distributed algorthms that are within these size constraints. In addition, for many distributed algorithms, such as the simplex algorithm, the number of tasks is a user option and can therefore be specified to be in the range necessary for Algorithm M.

3.3. Algorithm H, a Heuristic Algorithm for Task Assignment

Theorem 1 suggests the following heuristic, polynomial-time algorithm for task systems with an arbitrary number of tasks, n identical processors, and bound B, $\lceil \frac{k}{n} \rceil \leq B \leq k$, on the maximum number of tasks per processor.

Algorithm H: This algorithm reduces the original task graph to one containing $\leq 2n$ nodes, each with no more than $\lceil \frac{B}{2} \rceil$ tasks per node. Algorithm M can then be used to produce an optimal assignment for the reduced graph, but this assignment may be suboptimal for the original graph.

- Construct a graph G with a node for each task t_i and an edge between each pair of nodes t_i and t_j with weight c_{ij}.

- If $k \leq 2n$, then Theorem 1 applies and Algorithm M can be invoked to obtain an optimal assignment.

- If $k > 2n$, then tasks are grouped into clusters utilizing the Sort Greedy Algorithm (described below) with a limit $\lceil \frac{B}{2} \rceil$ on the maximum size of a cluster, where $\lceil \frac{k}{n} \rceil \leq B \leq k$. Sort Greedy continues to form clusters until the number of clusters is less than or equal to $2n$.

- A new graph G' is constructed with a node corresponding to each cluster and an edge between the pair of clusters r_1 and r_2 with weight

$$w_{12} = \sum_{\substack{t_i \epsilon r_1 \\ t_j \epsilon r_2}} c_{ij}$$

Symmetric Contraction of A Distributed Algorithm for the Simplex Method *

The system to be solved is represented by a matrix M. and a solution is obtained through repeated iterations on *M.* At each iteration, it is necessary to (1) select a pivot column from *M,* (2) select a pivot row from *M,* and (3) perform operations on each row in *M* using the values of the elements in the pivot row.

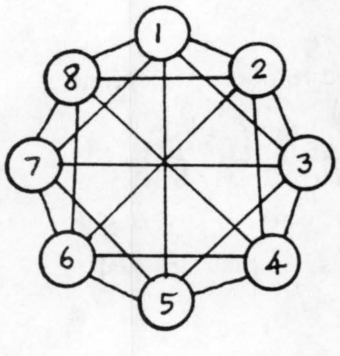
Given m rows in *M,* we distribute the work to p processes by assigning each **calculator process** m/p contiguous rows. The selection of the pivot column can be done locally by each calculator process. However, the selection of the pivot row involves choosing the "best" row of the m rows in *M.* One alternative for achieving the distributed voting to select the pivot row utilizes latin squares.

Latin squares voting: Each calculator process has an ordered list of all the other calculator processes such that no two lists contain the same process in the same ordinal position in the list. During each round, a given calculator sends its best row (either its own best, in round 0, or the best seen so far, in later rounds) to the next calculator on its list. During that round, it also receives a message from some other calculator. The lists can be arranged so that after $\log p$ rounds (base 2), each calculator knows the best row (assuming the number of processes p is a power of 2).

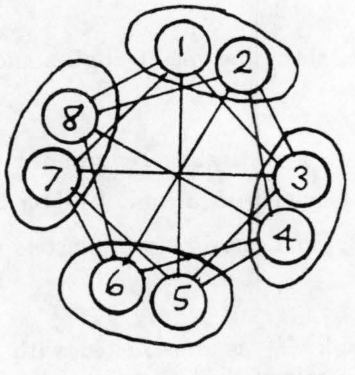
A task graph representing the distributed simplex algorithm designed for 8 tasks is shown below. The communication edges are defined by the latin squares configuration also shown in the figure. Because of the regular nature of the distributed algorithm, $c_{ij} = C$ for all i,j. If contraction of this algorithm is necessary, contraction to 4 processors preserves the parallelism in the algorithm because of the identical nature of the tasks. Thus Algorithm M can be invoked with B=2, k=8, and n=4. to find an optimal assignment with total IPC of 20*C units.

Lists for the Calculator Processes

	1	2	3	4	5	6	7	8
Round 0	2	3	4	5	6	7	8	1
Round 1	3	4	5	6	7	8	1	2
Round 2	5	6	7	8	1	2	3	4



(a) before contraction



(b) optimal contraction

Figure 2: Contraction of a Distributed Algorithm for the Simplex Method

* Only the bare bones of the algorithm are presented here. The full algorithm can be found in [6].

- Since the number of nodes in this new graph G' is less than or equal to $2n$, Algorithm M can be used to produce an assignment of clusters to processors which minimizes the total inter-cluster communication costs while keeping the number of tasks on each processor less than or equal to the bound B.

Subroutine Sort Greedy: This subroutine is a greedy algorithm which groups tasks into clusters of size $\leq \lceil \frac{B}{2} \rceil$ such that the total number of clusters is $\leq 2n$.

```
• Initially, each task in $G$ is in a task group by itself.

• Construct a list $L$ which contains the edges of $G$ sorted in non-increasing order.

• While there are unmarked edges remaining

        • • Find the next unmarked edge $e = (t_i, t_j)$ in the list $L$. Mark it.
            $G_i$ is the task group containing $t_i$.
            $G_j$ is the task group containing $t_j$.

        • • If $|G_i| + |G_j| \lceil \frac{B}{2} \rceil$ then

                • • • Merge the two groups:
                      $G_{new} = G_i \bigcup G_j$

                • • • Mark all the edges between tasks in
                      $G_i$ and tasks in $G_j$

        • • Else do not merge $G_i$ and $G_j$.
```

Algorithm H is a polynomial time algorithm. The complexity of Subroutine Sort Greedy is $O(e \log e)$ where e is the number of edges in the computation graph. The maximal matching is found using Algorithm M in polynomial time as discussed earlier.

3.4. Simulation Results

In order to evaluate the performance of Algorithm H in finding suboptimal task assignments, simulation runs were performed on a variety of typical task forces. Altogether, 90 task forces were simulated with the number of tasks ranging from 4 to 35 and the number of processors ranging from 2 to 5. Optimal assignments were computed using a branch and bound backtracking algorithm.

The data used in the simulations are organized into four categories. Dataset 1 (*Clustered*) consists of randomly generated task-processor systems in which tasks form clusters. Communication costs between tasks within the same cluster are on the average larger than communication costs between tasks in different clusters. Dataset 2 (*Sparse*) consists of randomly generated task-processor configurations in which the communication matrix is sparse. In particular, the communication costs are nonzero for only $\frac{1}{6}$ of the $\binom{k}{2}$ possible pairs of tasks. Dataset 3 (*Actual*) consists of data representing actual task forces derived from numerical and matrix algorithms, operating systems programs, and general applications programs. In this dataset, specific information about the number of tasks and/or about which pairs of tasks communicate with each other was available in the literature. Estimates of execution and communication costs were made from information such as the number and type of messages passed between tasks, from the function of the tasks, and from raw data on these costs. Dataset 4 (*Structured*) consists of task forces whose task graphs have the structure of a ring, a pipe, a tree, or a lattice. Details about these datasets can be found in [11].

Algorithm H performed extremely well, finding an optimal assignment in 81.1% of cases. Table 1 below shows the ratio $\frac{T_H}{T_O}$ where T_H is the cost of an assignment found by Algorithm H, while T_O is the cost of an optimal assignment.

Table 1: Distribution of Ratio T_H/T_O by Dataset

Dataset	Total No. of Simulations	Percent of Simulations					
		(Optimal) = 1.00	≤ 1.10	≤ 1.20	≤ 1.30	≤ 1.40	≤ 1.50
All Data	90	81.1	91.1	95.5	96.6	98.8	100.0
(1) *Clustered*	19	47.4	79.0	94.8	100.0	100.0	100.0
(2) *Sparse*	16	87.5	87.5	87.5	87.5	93.8	100.0
(3) *Actual*	22	90.9	100.0	100.0	100.0	100.0	100.0
(4) *Structured*	33	90.9	93.9	96.9	96.9	100.0	100.0

4. Conclusion

This work represents a contribution to the current research effort to utilize the potential of distributed computing systems for parallel computation. To summarize, we have presented two algorithms for the assignment of tasks to processors in order to minimize interprocessor communication costs under the constraint of a bound on the number of tasks assigned to each processor. Algorithm M finds optimal assignments for systems in which the number of tasks is $\leq$ twice the number of processors. Algorithm H is a heuristic for arbitrary task-processor configurations. This model of the task assignment problem and the use of Algorithms M and H are suitable for assignment of tasks and for contraction of tasks in distributed systems which have ethernet as the underlying communication medium.

We are looking at a number of extensions to the work described in this paper. First, because Algorithm H utilizes a greedy type algorithm to reduce the task graph, it is clear that poor assignments may result when the task graph has uniform communication costs. For example, in the binary tree task graphs of Figure 1, an optimal assignment has cost 3 while Algorithm H could yield a poor assignment with cost 8. (Algorithm H could also find the optimal assignment but is not guaranteed to.) Thus, we are interested in refining Algorithm H to handle the case of uniform communication costs. We are also interested in finding algorithms tailored to regular graph structures such as trees, rings, lattices; for these restricted graphs, it may be possible to find optimal algorithms. In the longer run, we are interested multiphase contraction as described in [14] and decentralized dynamic contraction. Many distributed algorithms involve multiple execution phases with a distinct communication pattern associated with each phase. Decentralized dynamic contraction involves local detection of the need for contraction at execution time and achievement of contraction through negotiation among processors rather than through a centralized controller. Both of these problems are related to our interest in process migration in distributed computing systems;

in fact, dynamic contraction is essentially carefully-orchestrated process migration. We are currently engaged in a study of parallel and distributed algorithms to determine the role that characteristics of these algorithms can take in guiding process migration in distributed systems.

References

[1] Y. Artsy, H.Y. Chang, and R. Finkel, "Processes Migrate in Charlotte", University of Wisconsin Dept. of Computer Science Technical Report No. 655, August 1986.

[2] F. Berman and L. Snyder, "On Mapping Parallel Algorithms into Parallel Architectures", *Journal of Parallel and Distributed Computing", Vol. 4 No. 5, Oct. 1987, pp. 549-458.*

[3] S.H. Bokhari, "Partitioning Problems in Parallel, Pipelined and Distributed Computing", *IEEE Transactions on Computing,* can't find which issue now, but will find it, 1987.

[4] W. W. Chu, L. J. Holloway, M. T. Lan, and Kemal Efe, "Task Allocation in Distributed Data Processing," *IEEE Computer,* Nov. 1980, pp. 57-69.

[5] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *IEEE Computer,* June 1982, pp. 50-56.

[6] R. A. Finkel, "Large-grain Parallelism - Three Case Studies", in *The Characteristics of Parallel Algorithms,* edited by L.H. Jamieson, D.B. Gannon, and R.J. Douglas, MIT Press, 1987, pp. 21-64.

[7] Z. Galil, S. Micali, and H. Gabow, "Priority Queues with Variable Priority and an $O(EV\log V)$ Algorithm for Finding a Maximal Weighted Matching in General Graphs", *23rd Annual Symposium on Foundations of Computer Science,* Nov. 1982, pp. 255-261.

[8] L. Hyafil and R.L. Rivest, "Graph Partitioning and Constructing Optimal Decision Trees are Polynomial Complete Problems", Report No. 33, IRIA-Laboria, Rocquencourt, France, 1973.

[9] E. Lawler, *Combinatorial Optimzation, Networks and Matroids,* Holt, Rinehart, and Winston, 1976.

[10] V. M. Lo and J. W. S. Liu, "Task Assignment in Distributed Multiprocessor Systems" *Proceedings of the 1981 IEEE International Conference on Parallel Processing,* 1981, pp. 358-360.

[11] V. M. Lo, "Task Assignment in Distributed Systems,", Dept. of Computer Science, University of Illinois, Ph.D. Thesis, October 1983.

[12] V. M. Lo, "Task Assignment to Minimize Completion Time", *IEEE 5th International Conference on Distributed Computing Systems,* May 1985.

[13] V. M. Lo, "Heuristics for Static Task Assignment in Distributed Systems", in press for *IEEE Transactions on Computers.* (Also University of Oregon Technical Report CIS-TR-86-13.)

[14] P. A. Nelson, "Parallel Programming Paradigms", University of Washington Computer Science Technical Report No. 87-07-02, July 1987.

[15] H. S. Stone and S. H. Bokhari, "Control of Distributed Processes," *IEEE Computer,* July 1978, pp. 97-106.

[16] H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Trans. on Software Engineering,* Vol. SE-3, No. 1, Jan. 1977, pp. 85-93.