A Min-Max Algorithm For Data Assignment in Parallel Computation

Julius Dichter
Computer Science and Engineering
University of Bridgeport
Bridgeport CT 06601

Howard Sholl
Computer Science and Engineering
Booth Research Center
University of Connecticut
Storrs CT 06269

Abstract

One of the key factors that affects performance of parallel computations is interprocess communication. It is often desirable to minimize communication to optimize the response time of a parallel or distributed computation. A computation may take any of a number of execution paths. This is due to the natural branching conditions which typically occur in computations. Depending on the actual execution path, assignment of data elements to processes may vary. We present the Min-Max algorithm which determines the minimum upper bound for data communication cost no matter which path executes.

1 Introduction

There has been much written in the literature about ways to minimize data communication between processes in parallel and distributed computations [1-5,8,9,12]. Typical approaches have included the data parallel approach in which identical processes execute in parallel on different data partitions [9]. This method works only if the problem naturally fits into a data parallel algorithm. Load balancing has been used widely [1,4,5,12]. The assumption is that by keeping all processors equally busy, the execution time will be minimized. This can be effective, but the underlying assumption is that the processes and data are paired such that data sharing will not be significant factor. Compile-time strategies have been developed using analysis of processes' read and write requirements [8]. This is an adapted cache coherence strategy to data sharing, requiring specialized instructions, and useful only in shared-memory systems. In [2], we presented an *Optimal Data Assignment Algorithm* (ODA). Given a set of processes, a data set, a data-access matrix, and some system cost parameters, ODA determines the assignment which minimizes the total interprocess data communication cost in the computation. The ODA algorithm is effective when the computation execution path is known *a priori* to run time. However, when a computation branches, it effectively prunes off part of the equation used for data assignment. If we factor in the branching probabilities to the read and write costs, and apply the ODA algorithm, the result would minimize only the average cost. In the extended computation structure model, control and data graphs offer a view of the serial and parallel computation flow as well as existing interprocess data dependencies [11]. These are likely to exist in computations with functional partitioning. We use this knowledge along with specified or observed branching probabilities to place data, that is, associating it with a process or processes.

2 Accounting for Branching

In Fig 1, a control graph is shown for a computation with 8 processes $\{p_1..p_8\}$ with specified branching probabilities and read-write requirement quantities ("*r* " and "*w*") for a single data element. There are four possible execution paths: path 1 = $\{p_1,p_3,p_4\}$, path 2 = $\{p_1,p_3,p_5\}$, path 3 = $\{p_2,p_6,p_8\}$, path 4 = $\{p_2,p_7,p_8\}$. To assign the data element optimally, we need to consider the branching probabilities as well as the read-write requirements. We can assume that loops have been absorbed into the graph, and reside in individual processing nodes. To compute the average read requirement for a process *i*, $r_{i,avg}$ we compute

$$r_{i,avg} = \sum_{path \in \mathbf{P}_i} r_i * \Pr(path)$$

$\mathbf{P}$ is the set of all possible paths, while $\mathbf{P_i}$ is a subset of $\mathbf{P}$ such that it contains all paths which include process *i*. The computations for $w_{i,avg}$ are done in a similar manner. The results of computing all $r_{i,avg}$ and $w_{i,avg}$ are shown in Fig 2. We can apply the ODA algorithm directly to Fig 2 to generate the optimal data assignment for the average case. Actual data communication costs will vary with the actual execution path taken during a particular execution. Therefore we have optimized only the average case, which may never occur.

3 ODA Algorithm Conditions

Essentially the ODA algorithm [2] defines two conditions, First Assign (*FA*) and Add (*ADD*) for each data element. We assign a data element to any *single* process satisfying *FA* (we are guaranteed that such a process exists). Then any process satisfying *ADD,* also has the data element assigned. We have proved that doing so will guarantee the minimum data communication cost. We simply repeat this for all data elements in the data set. The ODA parameters are L, R, α, and β. While L and R are the local and remote *read* costs respectively, αL and βR are the local and remote *write* costs respectively. **S** is the set of all processes and **A** is the set of all assigned processes, i.e. $\mathbf{A} \subseteq \mathbf{S}$. Let us assume that L << R and that $\alpha = \beta = 1.5$. Using the ODA algorithm yields conditions as follows.

First Assign (*FA*) a data element to a single process *i* satisfying

$$\text{MAX}_{\, i \in S} \{r_{i,avg} + 1.5{*}w_{i,avg}\} \qquad (1)$$

Add (*ADD*) the data element to any other process *i* satisfying

$$\frac{r_{i,avg}}{1.5} + w_{i,avg} > \sum_{i \in S} w_{i,avg} = \omega_{avg} \qquad (2)$$

Let us define $\boldsymbol{add}_i = r_{i,avg}/1.5 + w_{i,avg}$, that is, the ***ADD*** value of process *i*. Similarly, $\boldsymbol{fa}_i = r_{i,avg}+1.5{*}w_{i,avg}$, the ***FA*** value of process *i*. Computing ω_{avg} in Fig 2 gives 19.16. Process 7 satisfies *FA* because $\boldsymbol{fa}_7 > \boldsymbol{fa}_i$, $i \neq 7$. Since no process $\boldsymbol{add}_i > \omega_{avg} = 19.16$, the ODA result is simply **A** = {p_7}. We can then perform this process for all data elements in the data set, thus obtaining a total minimum cost data assignment on average. Given an assignment **A**, we can compute the communication cost, for all possible data paths. We will count local access as free. With β = 1.5, interprocess reads will cost R and writes 1.5*R, where R is the remote read cost. This gives a cost vector CV| **A**={p_7} = [66,57,48,25], where cost is in units of R. For example, 66 corresponds to the communication cost of path 1. Similarly, we have a path execution probability vector PV = [0.24,0.16,0.18,0.42]. The PCV (Probabilistic cost vector) for a given assignment set **A** is just the vector obtained by multiplying the corresponding elements of CV|**A** and PV. This gives a PCV| {p_7} = [15.84, 9.12, 8.64, 10.5]. Summing the components of PCV, we get 44.1, the expected cost of the data assignment **A** = {p_7}. However, by examining CV, we see that 44.1 does not match any of the actual costs given in CV|**A**={p_7}. A fair question to ask would be whether the computation could guarantee to bound communication costs to, say *k*. If we know that

$$\text{MAX}\,\{CV|\mathbf{A}\} < k \qquad (3)$$

then the answer is *yes*. If (3) is not satisfied, we cannot answer the question with certainty. Above, CV|{p_7}, costs for paths 1 and 2 are 66 and 57 respectively. We note that in Fig 2, $\mathbf{fa}_1$ = 10.4. It is the penultimate value behind $\mathbf{fa}_7$ = 10.92. This is significant because it shows that the initial assignment to process 1 is nearly as good as to process 7, giving a cost vector CV|{p_1} = [40,35,48,51]. This shows that selecting contrary to the ODA algorithm may produce preferable upper bound costs. In comparing **A** = {p_1} with **A** = {p_7}, path 3 cost is unchanged, paths 1 and 2 improve, only path 4 has a higher cost than with **A** = {p_7}. It is typical to try to minimize total cost. However when such cost is composed of a set of costs, which, in turn are interdependent, minimizing one may have an adverse effect on some others. In such cases, the approach often used is called Min-Max. It is a way to minimize an upper bound. In [10], Schwartz used a Min-Max approach to minimize upper bound delay through a network having a fixed bandwidth, and exponentially distributed traffic through various system concentrators.

4 Data Assignment Schemes

While assigning data according to the ODA Algorithm may be a good choice, it will only result in optimal communication cost on average. Below we list three assignment schemes, each with slightly different goals:

1. Assign data to minimize the average communication cost

2. Assign data to minimize the upper bound on communication cost

3. Assign data dynamically during execution

To demonstrate these three approaches, we will return to the example in Figs 1 and 2. In Table 1, we show CV, PCV, and E[Cost] for three possible assignments. Scheme 1 simply uses the ODA which results in **A** = {p_7}. owing scheme 2, we would choose assignment set **A** = {p_1}, due to its lower upper bound cost, that is MAX{CV|{p_1}} = 51 < MAX{CV|{p_7}} = 66. The maximum CV component is given by MAX {CV | **A**}. The assignment **A** = {p_3} is particularly good for path 3, therefore we might want to reassign the data to p_3 if branch 3 goes to the left.

Data Assignment	CV	PCV	E[Cost]
A = {p7}	[66,57,48,25]	[15.84,9.12,8.64,10.5]	44.1
A = {p1}	[40,35,48,51]	[9.6,5.6,8.64,21.42]	45.26
A = {p6}	[66,57,25,51]	[15.84,9.12,4.5,21.42]	50.88

Table 1

In the next section, we develop scheme 2, and in future research we will develop scheme 3. This deals with data migration, whereby data may be reassigned during the run time to enhance performance.

5 Minimizing Upper Bounds

In order to meet soft deadlines, we must be able to determine whether we can keep the communication cost within some constraint k. We need to determine if $\exists$ **A** which satisfies (3). If such an assignment exists, then we determine that the deadline can be met with certainty, otherwise it cannot. The search space for data assignment is $O(2^{mn})$, where m is the number of processes and n is the number of data elements [4]. In Fig 1, the number of possible assignments is approximately 2^{80}. Any reasonable approach must avoid an exhaustive assignment search. In the next section, we simplify the search by considering path processes specifically and all others generically. We check paths individually, then the total computation.

6 The Min-Max Approach

The Min-Max assignment algorithm provides a least upper bound guarantee for communication cost. Meeting a deadline is simply a relaxed Min-Max solution. Therefore we will first develop the Deadline Assignment Algorithm, then extend it to the Min-Max solution. A wall clock deadline for a computation does not specifically address the communication cost. Yet the performance of some computations is tied closely to data communication costs. Furthermore, given a deadline, we may be able to determine that the communication must be bounded by some value, k. To determine whether communication constraint k can be met, we can consider each path independently, then attempt to find a solution satisfying all paths. It is well-known that when a problem with global constraints is partitioned and solved independently subject to local constraints, those results may not be trivially combined into a global solution [6]. If all paths *can* meet the deadline, then the computation as a whole may potentially be able to do it. Thus we need to determine whether an assignment which allows all paths to satisfy the deadline exists. If it does exist, we need to be able to specify it.

6.1 The Deadline Assignment Algorithm

In order to present such an algorithm, we define the following terms:

A - The optimal data assignment set for the entire computation
$\mathbf{A}_i$ - Optimal data assignment set for path i
P - The number of paths in a computation
m - The number of processes in a computation
k - The communication deadline
P_{Ai} - The set of assignments which allow path i to meet the deadline
P_{ACi} - Set of *all* possible assignments to path i

The algorithm can be stated as follows:

I. Compute $\mathbf{A}_i$ for each path using the ODA algorithm

II. If for $\forall$ i, the communication cost of Path $i|A_i < k$, go to III, otherwise go to V

III. Compute P_{Ai} for each path

IV. Determine if there exists a single assignment P_{Ai} for all i. If such an assignment exists, then report and terminate the algorithm. Otherwise, proceed to step V

V. Report that the deadline cannot be met. Terminate algorithm, or compute the Min-Max Assignment

The above algorithm finds an assignment such that communication time is bounded by k. Steps I and II are straightforward. For Steps III and IV, we can compute P_{Ai} using only processes in path i. Later we can extend P_{Ai} with all other processes without regard to their identity. If an assignment which satisfies a deadline for path i includes a set of processes, **Q**, *not* in path i, then there exist other satisfying assignments with any $|\mathbf{Q}|$ processes not in path i substituted for the elements of **Q**. We will use the term *abstract process* to refer to non-path processes. For example, let us say that a computation has 10 processes, p_1 through p_{10}. Let path 1 $=\{p_1, p_3, p_5\}$and path 2 $=\{p_2, p_7, p_9, p_{10}\}$. If, for a particular deadline, P_{A1} includes $\{p_1\}$ and $\{p_1, \mathbf{p_2}\}$, then p_2 is an abstract process because any other process *not* in path 1 can be substituted in its place. We can then write $\{p_1, \mathbf{abs_1}\}$, an abstract set, to represent the 7 assignments which satisfy the deadline. If we can also include the abstract set $\{p_1, \mathbf{abs_1}, \mathbf{abs_2}\}$, then the two abstract processes represent 21 assignments which satisfy the deadline. For example, consider $\mathbf{P_{Ai}}$'s from paths 1 and 2. If we are matching up $\{\{p_1, \mathbf{abs_1}, \mathbf{abs_2}\}\} \in P_{A1}$ and $\{\{p_2, \mathbf{abs_3}, \mathbf{abs_4}\}\} \in P_{A2}$, then 21 assignments are being matched to 15 others in a trivial manner: p_1 matches abs_3; p_2 matches to abs_1; and abs_2 and abs_4 match. The complete rules for matching sets are given in [3].

6.2 The Min-Max Algorithm

The Min-Max Algorithm can be used as an extension to the Deadline algorithm, or alone. When used as an extension its steps can be numbered VI and VII. When used alone we can refer to them as I and II.

VI. Compute P_{ACi} for all paths

VII. Determine an assignment $\theta \in P_{ACi}$ for all i such that its MAX $\{CV \mid \theta\}$ is minimized or it is within suitable tolerance level from the constraint k.

P_{ACi} is similar to P_{Ai} except that it includes *all* possible assignments. Let path i contain 4 processes, and the whole computation contain 10 processes. Therefore P_{ACi} will have 2^4-1= 15 non-abstract assignments , but 2^{10}-1 = 1023 total assignments. However, only 15 non-abstract and 96 abstract assignments are necessary. In general, if a path i has m_i processes and the entire computation has m total processes, the number of total assignments is given by

$$\sum_{1\le j\le m_i} \binom{m_i}{j} + (m - m_i) \sum_{0\le j\le m_i} \binom{m_i}{j}$$

where the first term represents the non-abstract assignments and the second the abstract ones. For example, let us consider a path i, which has $m_i = 3$ processes, and the computation has m = 20 processes. Then the total number of assignments necessary for path i is just 143, and not 2^{20}. Finding the data assignment is just a matter of matching sets. In practice the total number of assignments in P_{ACi} should be less than (4) because we can run the ODA algorithm on the computation, generate $CV|\mathbf{A_{ODA}}$ and use it to so that $MAX\{CV| P_{ACi}\} < MAX\{CV|\mathbf{A_{ODA}}\}$. In other words, using ODA we can find the best average assignment. Then we can use its maximum CV element to limit the number of assignments actually placed in the P_{ACi}'s. Finding a global solution consists of determining a good initial solution, and using it in a branch-and-bound algorithm. Branch-and-bound algorithms are very useful for solving computationally intensive problems [7]. The result of the branch and bound search will be the Min-Max assignment. We may stop early by settling for an assignment within an acceptable tolerance from k.

7 Min-Max Example

Let us consider the computation depicted in Figs 1 and 2. Let us impose a deadline of k = 51 time units. In Table 2, we used the ODA Algorithm to compute the optimal assignments for each path independently.

$Path_i$	A_i	Cost
$Path_1$	$A_1 = \{p1\}$	40
$Path_2$	$A_2 = \{p1\}$	31
$Path_3$	$A_3 = \{p6\}$	25
$Path_4$	$A_4 = \{p7\}$	25

Table 2

This satisfies steps I and II of the algorithm. Step III requires that we compute $\mathbf{P_{Ai}}$ for each path. The results for the $\mathbf{P_{Ai}}$'s are summarized below. The numbers following the assignments represent the actual communication costs.

$\mathbf{P_{A1}}$ = {({p_1},**40**); ({p_3},**51**); ({p_4},**41**); ({p_1,p_4},**45**)}

$\mathbf{P_{A2}}$ = {({p_1},**31**); ({p_3},**42**); ({p_5},**41**); ({p_1,p_3},**37**); ({p_3,p_5},**47**); ({p_1,p_5},**36**); ({p_1,p_3,p_5} ,**42**)}

$\mathbf{P_{A3}}$ = {({p_2},**40**); ({p_6},**25**);({p_8},**35**); ({p_2,p_6},**37**); ({p_6,p_8},**36**); ({p_2,p_8},**47**); {(p_2,p_6,p_8},**48**); ({p_6,abs_{31}},**49**); ({abs_{32}},**48**)}

$\mathbf{P_{A4}}$ = {({p_2},**39**); ({p_7},**25**); ({p_8},**38**);{{p_2,p_7},**46**); {p_7,p_8},**45**); ({abs_{41}},**51**)}

A global solution is an assignment which appears in all four local solutions. If a solution exists, it must be one of the elements from $\mathbf{P_{A1}}$, either a specific or abstract assignment. It is simply an assignment for which all paths have a communication cost less than k. The element {p_1} matches exactly in $\mathbf{P_{A2}}$. In $\mathbf{P_{A3}}$ it matches the abstract set {abs_{32}}, while in $\mathbf{P_{A4}}$ it matches the set {abs_{41}}. We conclude that the deadline k = 51 can be met with the assignment **A** = {p_1}. In a similar fashion, **A** = {p_3} can also satisfy the deadline. We need not compute it however, since we only need to determine if the deadline can be satisfied. This method is graphically depicted in Fig 3. From this figure we can also see that the deadline k = 45 cannot be met. Suppose we now want to find the Min-Max assignment. We can set k = 0, use ODA to bound the communication costs of assignments in P_{ACi}, and generate *all* solutions. The lowest cost solution is the Min-Max solution. In this case, k = 51 *is* the Min-Max solution. The Min-Max Algorithm simply exhausts all assignments which perform better than the ODA one in the worst case.

8 Conclusions

We have developed a method of guaranteeing an upper bound for interprocess communication in parallel and distributed computations. An algorithm was presented to achieve a Min-Max solution such that the minimum upper bound was found. To reduce the large data assignment search space, abstract sets were used to simplify the set matching problem. This research has wide applications in computations where branching can significantly affect the data communication, as well as in computations utilizing functional partitioning. As such our results can be applied to a wide variety of High-Performance and Real-Time Computing problems.

Control Graph Showing Read-Write Needs for a Data Element

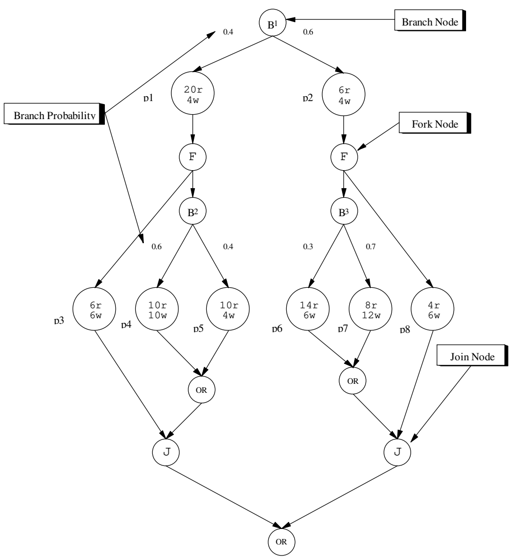

Figure 1

Control Graph Showing Branching Factored Into Read-Write Need

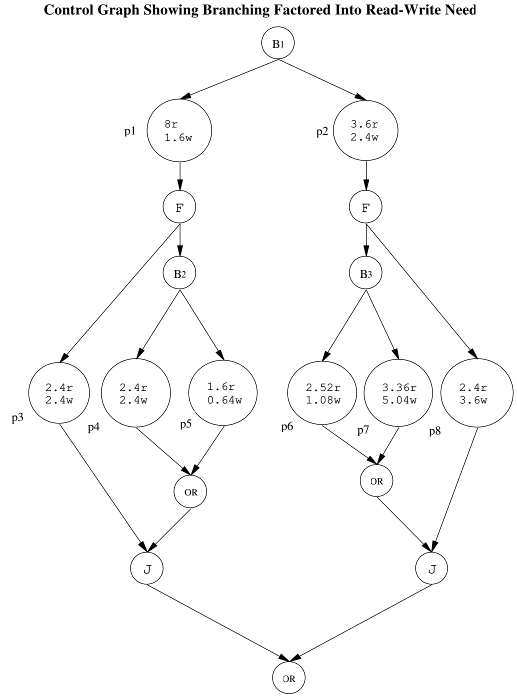

Figure 2

P_{A1} = {({p_1},**40**); ({p_3},**51**); ({p_4},**41**); ({p_1,p_4},**45**)}

X X

P_{A2} = {({p_1},**31**); ({p_3},**42**); ({p_5},**41**); ({p_1,p_3},**37**); ({p_3,p_5},**47**); ({p_1,p_5},**36**); ({p_1,p_3,p_5} ,**42**)}

P_{A3} = {({p_2},**40**); ({p_6},**25**);({p_8},**35**); ({p_2,p_6},**37**); ({p_6,p_8},**36**); ({p_2,p_8},**47**); {(p_2,p_6,p_8},**48**); {p_6,abs_{31}},**49**); ({abs_{32}},**48**)}

P_{A4} = ({p_2},**39**); ({p_7},**25**); ({p_8},**38**); ({p_2,p_7},**46**); ({p_7,p_8},**45**); ({abs_{41}},**51**)}

Figure 3

References

[1] S. Chowdhury, "The Greedy Load Sharing Algorithm", Journal of Parallel and Distributed Computing 9 (1990) 93-99.

[2] J. Dichter and H. Sholl, “A New Optimal Algorithm For Data Assignment in Parallel Computation”, ISCA International Conference on Computers and Their Applications, March 1996, San Francisco.

[3] J. Dichter and H. Sholl, “Guaranteeing Data Communication Upper Bounds in Computations with Branching”, Technical Report, University of Connecticut, BRC/CSE-TR-96-08, June 1996.

[4] S. Garg, "Joint Utilization of Control and Data Partitioning in Determining Performance Optimum Software", Ph.D. Thesis, University of Connecticut, 1993.

[5] R. Hanxleden, L.R. Scott, "Load Balancing on Message Passing Architectures", Journal of Parallel and Distributed Computing, 13, 312-324, 1991

[6] I. Pramanic and J.Kuhl, “An Inherently Parallel Method for Heuristic Problem Solving: Part I - General Framework”, IEEE Transactions on Parallel and Distributed Systems, October, 1995.

[7] P. Ma, “A Model to Solve Time-Critical Application Problems in Distributed Computer Systems”, IEEE Computer, January 1984.

[8] F. Mounes-Toussi, D. Lilja, “The Potential of Compile-Time Analysis to Adapt the Cache Coherence Enforcement Strategy to the Data Sharing Characteristics”, IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 5, May 1995.

[9] N. Nedeljkovic, M. Quinn, "Data-Parallel Programming on a Network of Heterogeneous Workstations", Proceedings IEEE First International Symposium on High Performance Distributed Computing September 1992.

[10] Mischa Schwartz,Computer_Communication Network Design and Analysis, Prentice Hall, Inc, 1977.

[11] H. Sholl and B. Qin, “A Modified Computation Model to Predict the Time Costs of Parallel Computation”, International Computer Science Conference, Dec 1988, Hong Kong.

[12] T. Znati, R. Melhem, "A Uniform Framework for Dynamic Load Balancing Strategies in Distributed Processing Systems", Journal of Parallel and Distributed Computing 23(1994)246-255