A New Optimal 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

The performance of a parallel or distributed computation depends heavily on minimizing communication between the cooperating processes. When concurrent processes possibly residing on different processors require access to the same data, delay is introduced when a process needs to remotely access its data. While data replication minimizes this delay for reading data, any writing of data must suffer the penalty of data coherency maintenance. In this work, we present a new algorithm defining an optimal assignment of data to a parallel or distributed computation. The algorithm can be executed in linear time in the number of processes requiring access to a single data item. It is proven that the algorithm is optimal in reducing total data communication costs. We define the conditions for adding a process into a non-empty assignment set and those for adding an initial process. We show that both conditions are necessary for optimal assignment and that the conditions can occur independently. The algorithm is completely general and provides flexibility so that it may be applied to any system with various cost parameters. It is well-suited to shared memory systems as well as multicomputers and distributed platforms.

1 Introduction

The performance of a parallel and distributed computation depends on many factors, including the amount of parallelism, the number of processors, and the quantity of communication between the cooperating processes. When processes and data are independent, it means that each process may proceed at its own rate without any communication from any other process. More often, however, processes need to share data. That implies that any number of processes may need to read and/or write the same data element. In this paper, we define a datum as the unit of granularity. A datum, or data element, therefore, may be a single data element or a block of data elements. The data assignment problem has been studied widely [1,2,4-6,8,10,11,13]. Data assignment is the process of associating the data element to a particular process. When a datum is assigned to set of processes, those processes read the data locally, while the unassigned processes read it remotely. However, any process writing the data must ensure its coherency by writing it to all assigned processes. Optimal data assignment is an optimization problem. In this paper, we present an algorithm to optimally distribute the data for an arbitrary computation over a variety of parallel and distributed platforms.

In previous research into the data assignment problem various approaches were taken. In many approaches the data assignment was in inherent part of the load balancing [1,6,13]. Whenever a new process was forked, it, along with its data set was assigned to the "best" processor. The definitions of *best* vary from the least busy to the winner of a bidding procedure. Data parallel approaches [11] are similar. Processes reside on different processors, and equivalent computations are performed by each processor on a data partition. Sometimes physical processors execute several processes in a data parallel computation. Typically the data is partitioned so that each processor is data-independent, hence data allocation is not an issue. In [10], cache coherence strategies are adapted to data sharing. The data allocation decisions are performed at compile time. The compiler analyzes the code and marks the data blocks as to whether coherence enforcement is needed. In short, the compiler can issue three types of write instructions, write-only (if coherence cannot be a problem), write-invalid, or write-update. The last two allow the compiler to switch between the temporally more efficient coherence strategy: invalidating or updating. This approach is limited to shared-memory systems, and therefore does not generalize multicomputers or distributed systems. The problem of database file allocation was investigated in [8]. A probabilistic solution was found to assign files at sites for a distributed database. In this work, configuring the connections for the underlying communication network was a constraint. Therefore transmission delays and great disparities between file sizes affect the assignment. In [4,5] the data assignment is specifically addressed. These works deal specifically with MIMD architectures and multicomputers, and assumptions on the relative costs of local and remote accesses are inherent. Therefore variations in MIMD implementations, such as cache write strategies, cannot be accurately modeled. In our approach, we individually parameterize the relative speed of reading and writing locally and remotely. This allows us to model various systems more accurately. We describe two conditions necessary for data assignment. The first qualifies an

initial process for assignment, the second qualifies subsequent processes. The union of processes satisfying the conditions results in the optimal assignment. In short, our approach extends solutions to a wider category of cases. In addition, our algorithm has an optimal complexity.

2 The Cost of Data Assignment

Let us consider the cost of a data assignment for an arbitrary computation. We will define *cost* as access to data locally or access by interprocess communication. In general there will be *n* data items, but for now we can consider one. For the computation, let us define the following:

S - set of all processes requiring the data element
A - set of the processes assigned the data element ($A \subseteq S$)
$|S| = m$ and $|A| = a$.
L, R - local/remote read access cost
LU, RU local/remote write access cost

Write cost are greater than read costs. We will define LU and RU as multiples of their read counterparts giving

$LU = \alpha L$ and
$RU = \beta R$, where α and $\beta > 1$.

To model the cost of data assignment we need to account for local access of assigned processes and remote access of unassigned processes. Additionally assigned processes have remote accesses when they write data assigned to other processes. Therefore the size of the assignment set **A** will have a substantial bearing on the cost of the assignment. Let r_i and w_i be the number of read and write operations required by process *i* to data d_k. The total cost of the assignment of data element d_k, is given by

$$L\sum_{i\in A} r_i + LU\sum_{i\in A} w_i + R\sum_{i\in S-A} r_i + RU[a\sum_{i\in S-A} w_i + (a-1)\sum_{i\in A} w_i]$$

which can be restated as

$$L[\sum_{i\in A}(r_i + \alpha w_i)] + R[\sum_{i\in S-A} r_i + \beta a\sum_{i\in S} w_i - \beta\sum_{i\in A} w_i] \quad (1)$$

It is clear from that adding a new process *i* to the assignment set **A** monotonically increases the left term of (1), but it is not clear how it affects the second term. In the next section, we investigate the behavior of data assignments for read-only (RO), write-only (WO), and read-write (RW) data. We then show how the data assignment can be done so that it is optimal, that is, it achieves minimum cost.

3 Optimizing Data Assignment Cost

3.1 Simple Cases: Read-only and Write-only data

We consider read-only data (RO) and write-only data (WO). RO is a data element which is never written by any process. This is a program input. WO is a data element which is never read by any process, a program output.
In the RO data case, (1) reduces to the following

$$Cost_{d_k} = L\sum_{i\in A} r_i + R\sum_{i\in S-A} r_i$$

This implies that the cost is minimum for

$$MIN\{L,R\}\sum_{i\in S} r_i$$

In the WO data case, (1) simplifies to the following

$$Cost_{d_k} = (\alpha L - \beta R)\sum_{i\in A} w_i + \beta Ra\sum_{i\in S} w_i$$

The following two lemmas show how optimal assignment should be done for cases where $L < R$ or $L = R$. Proofs can be found in [2].

Lemma 1.1
To achieve optimal response time in a parallel or distributed computation where L and R are costs of local and remote access respectively, a read-only data element should be fully replicated in all processes requiring access if $L < R$. If $L = R$, the data element can be assigned to any non-empty subset of processes.

Lemma 1.2
To achieve optimal response time in a parallel or distributed computation where L and R are costs of local and remote access respectively, a write-only data element should be assigned to the process with the greatest write need if $\alpha L < \beta R$. If $\alpha L = \beta R$, the data element can be assigned to any single process.

3.2 General Case: Read-Write data

The case for RW data must include an initial boundary condition. In the following discussion, conditions for adding a new process to a non-empty assignment set **A** are given. Later we consider how the selection of the initial choice for assignment. can affect the resulting overall data assignment
Below we simplify (1) for the general case

$$Cost_{d_k} = \beta Ra\omega + \sum_{i\in A}[Lr_i - (\beta R - \alpha l)w_i] + R\sum_{i\in S-A} r_i \quad (2)$$

Let us assume that the assignment set **A** is non-empty. Then we can develop conditions for adding any new process *i* into **A**. We need to carefully look at the terms of (2). The terms $\beta R a\omega$ and $\Sigma_{i\in A} Lr_i$ increase with each new process assignment. The term $R\Sigma_{i\in S-A} r_i$ monotonically decreases (2) by Rr_i. The last term, $\Sigma_{i\in A}(\beta R - \alpha L)w_i$ will either monotonically increase or decrease depending on the sign of $\beta R - \alpha L$. In general, we can always define the conditions for adding a process (*ADD*) as the one in which the terms which decrease cost are greater than the terms which increase it. This gives rise to three cases. In case 1, $\beta R > \alpha L$, the condition for adding a process *i* is given by

$$R(r_i + \beta w_i) > \beta R\omega + L(r_i + \alpha w_i)$$

Similar conditions can be given for case 2, $\beta R = \alpha L$

$$Rr_i > \beta R\omega + Lr_i$$

In case 3, $\beta R < \alpha L$, the condition can be expressed as

$$Rr_i > (\alpha L - \beta R)w_i + \beta R\omega + Lr_i$$

The important point to make about the *ADD* conditions is that each one shows that the condition for adding any process *i* into the non-empty assignment set **A** can be considered independently and in an arbitrary order. Once L, R, α, and β are fixed for any system and ω is computed, the addition of a process does not affect the eligibility of any other process.

3.3 Selecting the Initial Assignment Process

Having defined the *ADD* conditions, we now consider the conditions required for initial assignment into **A**, first assign conditions, *FA*. It is important to note that the initial element assigned into **A**, need not, in fact, cannot satisfy *ADD* conditions. That is because (2) is not properly defined when **A** = { }. We need to consider the choice of the first process to be assigned into **A**. It is natural to select the process which minimizes cost. But it is important to ensure that such a greedy choice promotes the absolute minimum total cost, and does not settle in a local minimum. It is well-known that a greedy algorithm selects from a local minimum, but may work adversely in achieving a global optimum [9,12]. For example, in the Traveling Salesman problem, using the heuristic of selecting the closest city to visit next in no way helps to obtain a minimum length tour of all cities [3]. The greedy approach assigns an initial process *j* which minimizes (2) with the constraint that $a = 1$. Once again this gives rise to three cases.

Let us assume that only process *i* has been assigned into **A**. Then the conditional cost is

$$Cost_{d_k} \mid A=\{p_i\} = \beta R\omega + (L - R)r_i - (\beta R - \alpha L)w_i \quad (3)$$

A process satisfying *FA* must minimize (3). We examine *FA* conditions for the three necessary cases.

Case 1 : $\beta R > \alpha L$

Since $\beta R\omega$ is a constant, we really need to minimize the two right-most terms of (3). We also need to know the value of (L - R). If L = R, then (3) is minimized for a process *j* when

$$w_j > \forall_{i\neq j} w_i$$

If L < R, then (3) is minimized when we maximize

$$(R - L)r_i + (\beta R - \alpha L)w_i$$

Case 2 : $\beta R = \alpha L$

This is a simple case where we just need to minimize $(L - R)r_i$ If L = R, then an arbitrary process can be assigned yielding equivalent results. If L < R, then the process with the maximum read requirement, r_i, should be assigned.

Case 3 : $\beta R < \alpha L$

In this case, we need to minimize

$$(\alpha L - \beta R)w_i + (L + R)r_i$$

If L = R, then the process with the minimum write requirement should be assigned, i.e. process *i* such that $w_j < \forall_{i\neq j} w_i$. If L < R, then (3) is minimized when we minimize

$$(\alpha L - \beta R)w_i - (R - L)r_i$$

The above cases define the *FA* conditions. A greedy approach is taken in assigning the first process into the assignment set **A**. Next, we show that this approach combined with assigning subsequent processes according to the *ADD* conditions will yield an optimal data assignment.

It is important to reiterate that the *ADD* conditions are not affected by how many processes have been assigned, that is, they are invariant given that an initial assignment has been made. *ADD* is a condition that qualifies processes to be added into the assignment set **A** given that it is non-empty, while *FA* is a condition that identifies processes qualifying to be initially assigned into **A** to achieve a minimum cost. Processes satisfying *FA* or *ADD* are identified as p_{FA} and p_{ADD} respectively. The next section presents the optimal assignment for a RW data element, followed by the optimal data assignment algorithm.

4 Optimal Data Assignment Algorithm

4.1 Optimal Assignment for RW Data Element

Theorem 1.1

To achieve optimal response time in a parallel or distributed computation for a RW data item, the assignment set **A** should be initialized with any single process, p_{FA}. Then all other processes can be considered for assignment sequentially and in an arbitrary order. Any subsequent process *i* can be added if it satisfies *the ADD* condition. Initial assignment of a process other than p_{FA}, cannot result in a lower assignment cost, but may result in a greater cost.

The proof, given in [2], shows how *FA* and *ADD* conditions for all cases can occur independently. As a result, if a process satisfying *FA* is added first, an initial optimal cost will be achieved. Then, every other process satisfying *ADD* can be assigned in an arbitrary order due to the *invariant* characteristic of *ADD*. The following summarizes *ADD* and *FA* for all possible cases.

Case 1 : $\beta R > \alpha L$

(L=R)

FA: p_i that satisfies $MAX_{i \in S}\{w_i\}$
ADD: p_i that satisfies $(\beta-\alpha)w_i > \beta\omega$

(L<R)

FA: p_i that satisfies $MAX_{i \in S}\{(R-L)r_i + (\beta R - \alpha L)w_i\}$
ADD: p_i that satisfies $(R-L)r_i + (\beta R - \alpha L)w_i > \beta R\omega$

Case 2 : $\beta R = \alpha L$

(L=R)

FA: An arbitrary process can be considered as p_{FA}
ADD: $0 > \beta R\omega$

(L<R)

FA: p_i that satisfies $MAX_{i \in S}\{r_i\}$
ADD: p_i that satisfies $(R-L)r_i > \beta R\omega$

Case 3 : $\beta R < \alpha L$

(L=R)

FA: p_i that satisfies $MAX_{i \in S}\{w_i\}$
ADD: p_i that satisfies $(\beta-\alpha)w_i > \beta\omega$

(L<R)

FA: p_i that satisfies $MAX_{i \in S}\{(R-L)r_i + (\beta R - \alpha L)w_i\}$
ADD: p_i that satisfies $(R-L)r_i + (\beta R - \alpha L)w_i > \beta R\omega$

It is clear from the above cases that *ADD* is a stronger condition: if it can be satisfied by a set of processes $S_{ADD} \subseteq S$, then there exists a non-empty set $S_{FA} \subseteq S_{ADD}$ satisfying *FA*. Assigning any single process $p_{FA} \in S_{FA}$ into the empty assignment set **A** produces the an initial highest gain. Since the set of processes satisfying *ADD* is not affected by any previous assignment, each $p_{ADD} \in S_{ADD}$ can be assigned into **A,** in an arbitrary order, yielding the optimal assignment for a data element.

4.2 The Optimal Data Assignment Algorithm

Based upon lemmas 1.1 and 1.2 and theorem 1.1, we now state the algorithm for optimal data assignment.

I. A read-only data (RO) should be assigned according to Lemma 1.1
II. A write-only data (WO) should be assigned according to Lemma 1.2
III. A read-write data (RW) should be assigned as follows

   i. Given system parameters L, R, α, β, calculate αL, βR, and ω
   ii.. Assign to the assignment set **A** any single process satisfying *FA*
   iii. Assign to the assignment set **A** any process satisfying *ADD*

4.3 A Special Case of the Algorithm

In the case of distributed computation across a network or a message-passing multicomputer, it is usually the case that R = KL where K >> 1. Also typically

$$\frac{LU}{L} \doteq \frac{RU}{R} \doteq \alpha, \beta$$

If we take the limit of (2) as L → 0, we obtain the cost of data assignment for our special case

$$Cost_{d_k} = \beta Ra\omega - \sum_{i \in A} \beta R w_i + R \sum_{i \in S-A} r_i \qquad (4)$$

ADD can be determined from (4) giving

$$\frac{r_i}{\beta} + w_i > \omega$$

Then a process *i* satisfies *FA* whenever

p_i satisfies $MAX_{i \in S}\{r_i + \beta w_i\}$

It is clear that β, the ratio RU/R, can have a significant effect on data assignment. In particular a cache write policy of write-through would produce significantly larger β than write-back.

	p_1,r	p_1,w	p_2,r	p_2,w	p_3,r	p_3,w	p_4,r	p_4,w	p_5,r	p_5,w	p_6,r	p_6,w
d_1	10	0	8	0	4	0	2	0	6	0	9	0
d_2	0	6	0	0	0	8	0	10	0	5	0	6
d_3	30	2	10	4	29	6	25	8	43	0	0	0
d_4	15	1	6	3	12	1	3	4	18	3	2	2
d_5	10	0	12	0	9	0	0	7	2	1	6	1

Table 1

In write-through main memory access cost would be incurred on each write [7]. A detailed example using the *FA* and *ADD* conditions just obtained for the special case is presented in the final section.

5 The Total Data Set Assignment Cost

Theorem 1.1 gives the cost for the data assignment for a single data item from the data set. In general, we are interested in the total cost of optimal assignment for the entire data set. The data set **D** has *n* data elements in it. Therefore we can generalize (2) for the entire data set $\mathbf{D} = \{d_k \mid 1 \le k \le n\}$. Overall optimal assignment will result when each of the *n* data elements has been assigned optimally. For a particular data element *i*, optimal assignment can be characterized by finding an optimal assignment set, $\mathbf{A}_{i\,OPT}$ with its corresponding cardinality, a_i. The optimal assignment cost for an entire data set is given by

$$\sum_{1 \le k \le n} [\,[L \sum_{i \in A_{k_{OPT}}} r_{i,d_k} + \alpha w_{i,d_k}] \; +$$

$$R\,[\sum_{i \in S_k - A_{k_{OPT}}} r_{i,d_k} + \beta a_k \sum_{i \in S_k} w_{i,d_k} - \beta \sum_{i \in A_{k_{OPT}}} w_{i,d_k}]\,]$$

5.1 Complexity of Total Data Assignment

In [2], a proof is given for the following lemma.

Lemma 1.3
The complexity of the optimal data assignment algorithm is O(mn), where *m* is the total number of processes involved in the computation and *n* is the number of data elements contained in the data set **D**.

6 Data Assignment Example

As an example, let us consider a distributed computation across a network or in a message-passing multicomputer. It is usually the case that R = KL where K >> 1. When local access is negligible, then *ADD* and *FA* are given in section 4.3. Let us define a data access matrix as a table with a row for every data element and two columns for every process, a read and write requirement. Every intersection of a row and column will give either a read or write requirement of a process for a particular data item. For example, $process_3$ requires 12 reads to $data_4$. We can assume either that all data have equal sizes, or that the access counts have been normalized in the table. For example if a datum has a size of 1 word and requires 10 reads, while another datum has a size of 20 words with 5 reads, then read counts would be 10 and 100 respectively. Also note that d_1 is a read-only datum (RO), while d_2 is a write-only datum (WO). Table 1 shows a data access matrix for a computation.

To show the effect of β on *FA* and *ADD* conditions, we will consider the processes which satisfy them for β = 1,2,3. For data element d_i, let $\mathbf{A}_{i,FA}$ be the set including $p_{FA,}$ and $\mathbf{A}_{i,ADD}$ is the set of all p_{ADD} processes. Then $\mathbf{A}_i$ is simply $\mathbf{A}_{i,FA} \cup \mathbf{A}_{i,ADD}$.

β = 1

$\mathbf{A}_{1,FA}$= {p1} $\mathbf{A}_{1,ADD}$ = {p1,p2,p3,p4,p5,p6} $\mathbf{A}_1$ = {p1,p2,p3,p4,p5,p6}
$\mathbf{A}_{2,FA}$= {p4} $\mathbf{A}_{2,ADD}$ = { } $\mathbf{A}_2$ = {p4}
$\mathbf{A}_{3,FA}$= {p5} $\mathbf{A}_{3,ADD}$ = {p1,p3,p4,p5} $\mathbf{A}_3$ = {p1,p3,p4,p5}
$\mathbf{A}_{4,FA}$= {p5} $\mathbf{A}_{4,ADD}$ = {p1,p5} $\mathbf{A}_4$ = {p1,p5}
$\mathbf{A}_{5,FA}$= {p2} $\mathbf{A}_{5,ADD}$ = {p1,p2} $\mathbf{A}_5$ = {p1,p2}

β = 2

$\mathbf{A}_{1,FA}$= {p1} $\mathbf{A}_{1,ADD}$ = {p1,p2,p3,p4,p5,p6} $\mathbf{A}_1$ = {p1,p2,p3,p4,p5,p6}
$\mathbf{A}_{2,FA}$= {p4} $\mathbf{A}_{2,ADD}$ = { } $\mathbf{A}_2$ = {p4}
$\mathbf{A}_{3,FA}$= {p5} $\mathbf{A}_{3,ADD}$ = {p1,p3,p4,p5} $\mathbf{A}_3$ = {p1,p3,p4,p5}
$\mathbf{A}_{4,FA}$= {p5} **$\mathbf{A}_{4,ADD}$ = { }** **$\mathbf{A}_4$ = {p5}**
$\mathbf{A}_{5,FA}$ = {p4} **$\mathbf{A}_{5,ADD}$ = { }** **$\mathbf{A}_5$ = {p4}**

β = 3

$\mathbf{A}_{1,FA}$= {p1} $\mathbf{A}_{1,ADD}$ = {p1,p2,p3,p4,p5,p6} $\mathbf{A}_1$ = {p1,p2,p3,p4,p5,p6}
$\mathbf{A}_{2,FA}$= {p4} $\mathbf{A}_{2,ADD}$ = { } $\mathbf{A}_2$ = {p4}
$\mathbf{A}_{3,FA}$ = {p4} **$\mathbf{A}_{3,ADD}$ = { }** **$\mathbf{A}_3$ = {p4}**
$\mathbf{A}_{4,FA}$= {p5} $\mathbf{A}_{4,ADD}$ = { } $\mathbf{A}_4$ = {p5}
$\mathbf{A}_{5,FA}$= {p4} $\mathbf{A}_{5,ADD}$ = { } $\mathbf{A}_5$ = {p4}

To show the effect of β on assignment, any set which is underlined differs from the similar set obtained using a former value of β. The example demonstrates that as β increases, it

becomes more and more difficult for processes to satisfy *ADD*. This is clear when we examine *FA* and *ADD* conditions obtained in section 4.3, when β is large. In such cases the import of the read accesses is diminished and that of writes is stressed. The assignment tends to go to the single process with the highest write access requirement.

7 Conclusions

We have demonstrated an algorithm for optimally assigning a data set over the processes in a parallel or distributed computation. The algorithm is general. It can be used in shared-memory systems where local and remote access are equivalent, as well as in systems where the cost of remote access far exceeds that of local one. Our example shows how the algorithm can be applied to typical message-passing system. Due to its low complexity, this algorithm can be used to be a part of a tool to automate the data assignment problem even for systems with numerous processors and large data sets.

References

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

[2] J. Dichter, H. Sholl, "Data Communication Optimization in Parallel Computing", Technical Report, University of Connecticut, BRC/CSE-TR-95-14, November 1995.

[3] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.

[4] S. Garg, H Sholl, R. Ammar, "Determining Optimum Software Partitioning and Data Allocation on a MIMD System to Minimize Completion Time", Journal of Computing and Information, Vol 1, No 1, 1994.

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

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

[7] J. Hennesy, D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, San Francisco California, 1994.

[8] K. Irani, N. Khabbaz, "A Methodology for the Design of Communication Networks and the Distribution of Data in Distributed Supercomputer Systems", IEEE Tranactions on Computers, Vol C-31, No. 5, May 1982.

[9] B.M.E.Moret, H.D. Shapiro, Algorithms from P to NP, The Benjamin Cummings Publishing Company, Inc., 1991.

[10] 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.

[11] 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.

[12] P. Purdom, C Brown, The Analysis of Algorithms, Holt, Rinehart and Winston, Inc., 1985.

[13] 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.