Communication Cost Estimation and Global Data Partitioning for Distributed Memory Machines

S. R. Prakash Y. N. Srikant*

Department of Computer Science & Automation
Indian Institute of Science
Bangalore, 560 012, INDIA.

Abstract

Estimating communication cost involved in executing a program on distributed memory machines is important for evaluating the overheads due to repartitioning. We present a scheme which will work with reasonable efficiency for arrays with at most 3 dimensions. Hyperplane Partitioning technique given by [10] is extended to complete programs by estimating the communication cost by the scheme presented in this work.

1 Introduction

Any strategy for automatic data partitioning needs a way of estimating communication cost involved when we implement the strategy on a target architecture. Many researchers have developed tools [3, 4, 5] that estimate the performance of a parallel program with explicit communication at compile-time. In [2], they employ "training sets" approach to estimate the performance of a parallel program. In [4], profiling is done to get parallel program parameters such as load distribution, communication overhead and data locality. In [9, 5], they use pattern-matching on array references to generate high-level collective communication.

We present here a (compile-time) scheme to estimate the communication cost involved in executing a loop on a parallel machine. This scheme works with reasonable efficiency only for dimensions upto 3. Even then, it would be very useful to know whether a given loop will be suitable for execution on parallel machine. If not we can decide to run the loop on a single processor. This is due to the fact that if the communication cost increases beyond some "threshold", the performance degrades to such an extent that it would be better to run the loop sequentially. We employ intersections of regions to estimate costs which are more accurate than profiling or pattern-matching schemes that are seen in literature. Profiling needs more time to estimate and pattern-matching schemes are limited to the data base of patterns available and matching things might turn out costly.

Schemes to get good data partition for arrays in a loop by inducing best iteration space partitions are presented in [10] with respect to our communication model. Here we present the overall algorithm adopted in this paper and details can be seen in [10]. We call such distribution of data space using one hyperplane as *Hyperplane Distribution*. We consider only such distributions in this work. We will show how we can extend the scheme for sequence of loops in an execution path of a program. We will first attempt to find the best distribution for such a path and later address the issue of finding the best distribution for the entire program.

2 Hyperplane Partitioning

In this section we will see very briefly the scheme called *Hyperplane Partitioning*, which is used in this work. The details can be seen in [10]. The method is as follows.

- First, for every pair of references, the dependence equation is computed. The dependence equation gives the next index point which depends on the present index point.
- For each pair, the direction of dependence are computed. The direction of dependence gives the direction of the vector along which the dependence vectors align eventually (far from the origin).

*srikant@csa.iisc.ernet.in

1094-7256/97 $10.00 © 1997 IEEE

- From these directions of dependences, we compute a Hyperplane which is used to partition the iteration space into as many number of partitions as there are logical processors. The Hyperplane will be the one which is best fit for the dependence directions obtained.
- These dependence directions induce data space partitions in every array used in the loop. The direction of dependences are then induced into the data spaces of all the arrays and we fit a Data Hyperplane in these Data spaces of all arrays.
- Each logical processor executes different partition in parallel keeping corresponding data partitions locally, and synchronization and non-local accesses are handled at runtime.

3 Communication Cost Estimation

For some loops, the computations will be inherently sequential and parallelizing such loops, would make them slower than sequential execution. So, it would be advantageous if we could estimate the communication cost involved in running the loop on a given machine with given data and iteration space distributions *at compile time* itself so that we can decide whether to run the loop parallel or to run it on a single processor.

In this work, we estimate the communication cost involved in running a given loop with given partitions (iteration space and data space). We will explain the essential idea by giving an example.

Observation 3.1 (Necessity of Communication) *While running a loop on distributed-memory machine, communication is done for every inter-processor dependency. So, to estimate communication involved we have to estimate the number of inter-processor dependencies.*

Example 1: Consider a loop with two array references $X[2i+j, i+j]$ and $X[i, j]$, with loop bounds $(0, 0)$ and $(20, 20)$. Figure 1 shows how the iteration space is to be partitioned obtained by the scheme presented in [10]. ∎

Fig. 1 shows how the number of such dependencies can be estimated. For each partition, corresponding to each processor, the section which contains the index points which depend on some index point of this processor, is computed. We call such sections as 'dependency sections'. For example, we can see that section 'trans_iter$i$.grp' is the dependency section for 'iter$i$.grp' which is the iteration space partition owned by processor i $(0 \leq i \leq 3)$.

Now, the let us see what is the communication involved for a processor k. Consider the area of intersection between the partition section for processor k and its dependency section. For all the index points in this intersection, we can see that there is no communication involved for processor k because both the index point and the index point which depends on this belong to same processor. So, overall communication involved is just the number of points which lie in the non-intersecting region as shown in the Fig 2. So, communication involved will be equal to the number of points lying in the region IBJFGH. As can be seen in [1], the number of integer points (index points) are equal to the area of the section. So, the area of IBJFGH gives the communication involved for processor k. Similarly, we can compute communication involved for other processors. Sum total will give us the total cost of communication for executing the loop. But, since the partitions are run in parallel the total time required to do the communication would be maximum of times required by these processors.

Actually, not all index points in that region corresponds to communication cost. The reason will be explicit in the Example 2.

Example 2: Consider a loop with array references $X[i+2j, i+j]$ and $X[i+j, j-i]$ with loop bounds being $(0, 0)$ and $(20, 20)$. Then, as can be seen in the Figure 3, not all integer points correspond to communication, because for some of the index points in the partition section, there will be no corresponding index points in the dependency section. For example, for the index point $(0, 5)$ the corresponding point in the dependency section would be $(2.5, 7.5)$ which is not actually an index point. So, for $(0, 5)$, there is no communication involved. Similarly, we can see that for $(1, 7)$ and $(2, 3)$ there are no legal index points in dependency section, and hence no communication for these points. We need to estimate the number of index points in the partition section such that corresponding index point in the dependency section is the integer vector. Lemma 3.1 will prove useful in estimating such a quantity. ∎

Lemma 3.1 (Integer vector) *There are atleast* $(ub_0 - lb_0) * (ub_1 - lb_1)/(l_1 * l_2)$ *values* (i, j) *for which the vector* $(ai + bj, ci + dj)$ *will be an integer vector, where* $lb_0 \leq i < ub_0, lb_1 \leq j < ub_1$ *and* l_1 *and* l_2 *are least common multiples of* (a_2, c_2) *and* (b_2, d_2) *respectively, and* $a = a_1/a_2, b = b_1/b_2, c = c_1/c_2, d = d_1/d_2$ *all the fractions being in reduced form (i.e, with gcd of numerator and denominator being 1)*

Proof We see that whenever a_2 and c_2 divides i and b_2 and d_2 divide j simultaneously, the given vector will

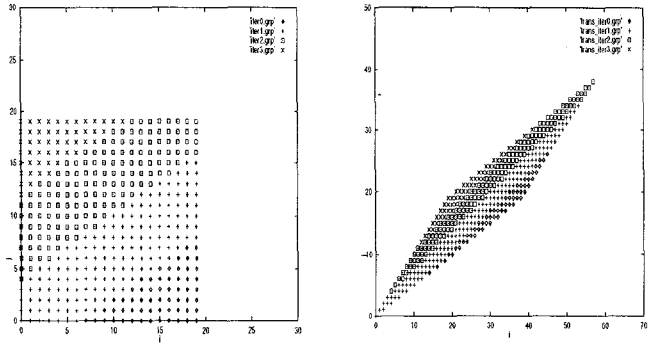

Figure 1: Iteration Partitions and Dependency sections for Example 1

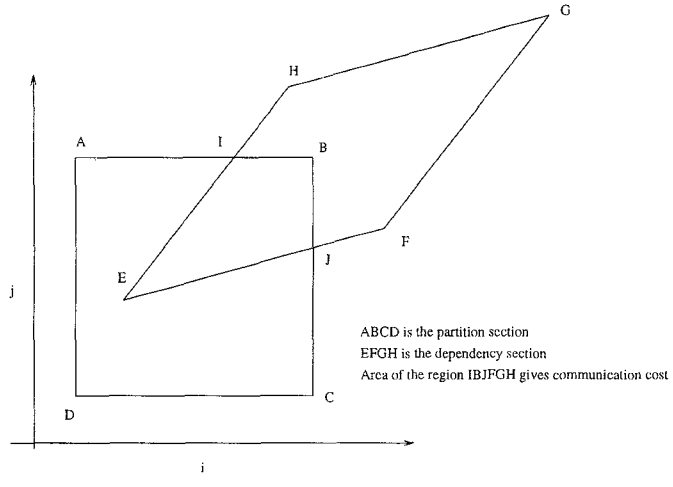

Figure 2: Communication involved for a partition

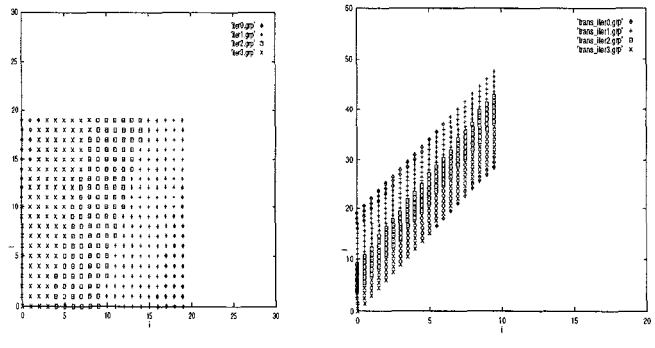

Figure 3: Iteration Partitions and Dependency Sections for Example 2

be an integer vector. It is easy to see that this happens when least common multiple of these numbers divide i and j respectively. They do it $(ub_0 - lb_0) * (ub_1 - lb_1)/(l_1 * l_2)$ number of times in the given range. ∎

Using the above lemma we can estimate fairly accurately the least number of integer points that would give legal index points in the dependency sections also. Using this factor, we can compute the communication cost involved. We note here that finding exactly the size of the solution space for a mixed integer programming is more difficult and we have left that as future research.

Once the communication cost is estimated, we can either leave it to the user to decide whether to go ahead with parallelization or not, or perhaps the compiler can decide by itself with some knowledge. The compiler can parallelize the loop if the communication cost is not greater than some threshold, which is found by running few programs on the target machine.

4 Global Data Partitioning

In this section we see how we can partition the data space of a program globally .i.e, for the complete program.

4.1 Data Distribution Tree

Consider a path in a program with five loops as shown in the Figure 4. In the Figure, each node in the graph represent a loop in the program. Figure 4 also gives the corresponding distribution tree for the sequence of loops. Each edge represents one possible way of distribution between two nodes and each node represents a loop with a particular distribution. Such a tree is called 'Data Distribution Tree(DDT)'.

Definition 4.1 (Best Local Distribution) *Best Local Distribution(BLD) for a loop L, is the hyperplane distribution for the loop L.*

Definition 4.2 (Static Distribution) *Static Distribution(SD) for a sequence of loops, $L_1, L_2 \ldots, L_k$ is the hyperplane distribution for all the loops $L_1, L_2 \ldots, L_k$ taken together. In other words, it is the Hyperplane Distribution for the given loops considering all the array references in these loops together.*

Definition 4.3 (Data Distribution Tree(DDT)) *A DDT for a given sequence of loops in a program is a directed tree where each node represents a loop in a program with a particular hyperplane distribution, and each outgoing edge for the node, I, represents a possible way of distribution for the loop I. Leaf nodes are called 'End Nodes'. Any path from the root (first loop) to any end node represents a possible way of distribution/s for the given sequence of loops.*

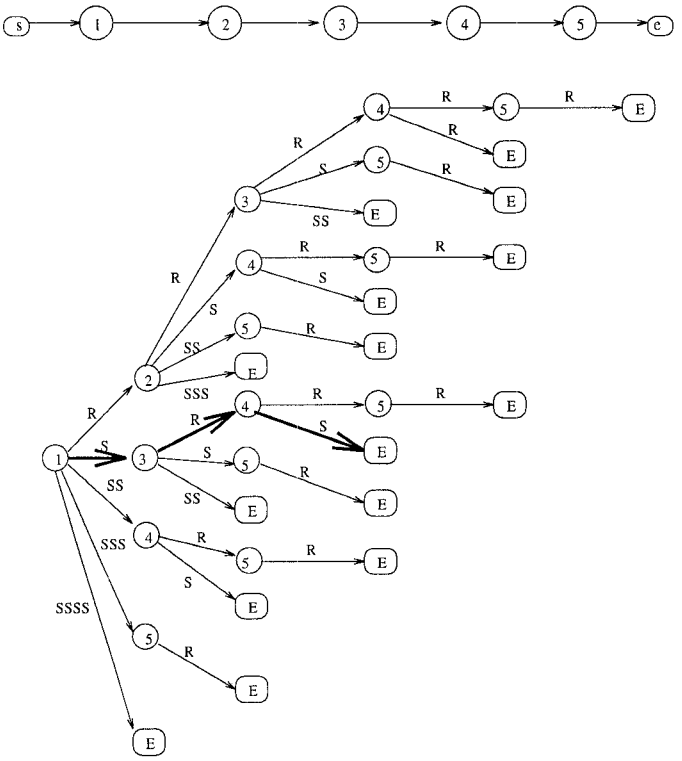

Figure 4: A sequence of loops and its Data Distribution Tree

Example 3: Consider Figure 4. We now explain how to construct such a DDT for a given sequence of loops. Consider the loop L_1. There are 5 possibilities of distributions for this loop as follows :

'R' Execute L_1 using BLD and come to L_2 to make further decisions.

'S' Execute L_1 and L_2 using SD for L_1 and L_2, and come to L_3 to make further decisions.

'SS' Execute L_1, L_2 and L_3 with SD for L_1, L_2 and L_3 and come to L_4 to make further decisions.

'SSS' Execute L_1, L_2, L_3 and L_4 with SD for L_1, L_2, L_3 and L_4 together, and come to loop L_5 for further decisions.

'SSSS' Execute L_1, L_2, L_3, L_4 and L_5 with SD for L_1, L_2, L_3, L_4 and L_5 and come to the End Node.

If we apply all possibilities for every node we get the tree as the one shown in Figure 4. ∎

Each of the paths from the node 1 to any End Nodes represent possibly way of distribution/s and redistribution/s along the execution path of the program. We need to find the *best* (least cost) path from the node 1 to any End Node in the DDT for the given sequence of loops.

Example 4: Let us suppose the best path is the one highlighted in the Figure 4. Then it says to get the best performance the loops L_1 and L_2 are to be run with SD for loops L_1 and L_2, execute loop L_3 with BLD and redistribute to run the loops L_4 and L_5 with SD for loops L_4 and L_5. ∎

To get the least cost path from the node 1 to any End Node we need to assign costs for every node in the DDT. We see that cost for every edge,E outgoing from node L_i, in the DDT can be estimated as given below :

CE = 0
if (label (E)='R') then
    CE = C(L_i,BLD) + CC(L_i)
else if (label(E)) = 'SS ...S' (k in no.)) then
    CE = C($L_i L_{i+1} L_{i+2}, \ldots, L_{i+k}$,SD) + CC($L_{i+k}$)

In the above algorithm, C(Loops,Type), gives the cost of executing the sequence of loops, Loops, with distribution, 'Type' for the sequence of loops in the case of 'SD' and for the single loop in case of 'BLD'. CC(L_k) gives the communication cost for redistributing after the L_k in the sequence to suit the distribution for the next loop.

Theorem 4.1 (Number of Distributions) *The number of times the algorithm for estimating communication cost is run is $\mathcal{O}(n^2)$, where n is the number of loops in the given sequence of loops.*

Proof We see that the number of distinct distributions considered is exactly equal to the number of times the communication estimator is run, because for every distribution for a loop we need to run the estimator. Now, the number of distributions considered
$= n$ (for 'R' labels for every loop) +
$(n-1)$ (#SD possibilities for Loop L_1) +
$(n-2)$ (#SD possibilities for Loop L_2) + ⋯
$(n-k)$ (#SD possibilities for Loop L_k) + ⋯
1 (#SD possibilities for Loop L_{n-1}) =
n(n+1)/2 = $\mathcal{O}(n^2)$ ∎

Example 5: Consider the sequence of loops given in Figure 4. We see that number of cost estimations to be done for the L_1 is 5 (1 for BLD, 4 for SDs for loops (L_1, L_2),(L_1, L_2, L_3),(L_1, L_2, L_3, L_4) and $(L_1, L_2, L_3, L_4, L_5)$). Similarly for loops L_2, L_3, L_4, L_5 it is 4,3,2 and 1 respectively and thus total number of times the cost estimation has to be done is $1 + 2 + 3 + 4 + 5 = 15 = n(n+1)/2$, for $n = 5$. The other point to note is that even though we have exponential number of edges in the DDT, we need to compute only $\mathcal{O}(n^2)$ since the same edge cost repeats many times as can be seen in Figure 4 for L_3 where the edges $3-4, 3-5, 3-E$ repeats twice. ∎

4.2 Communication Cost for the Sequence of Loops

We will see how we can estimate the communication cost for a sequence of loops once we have constructed DDT for the sequence of loops. Obviously we need to find the least cost path from the start to the end node of a DDT.

4.2.1 Redistribution Cost

We assume in this work that communication time,t involved in sending a message of length l between any two processors, follows the equation,

$$t = \alpha l + \beta \tag{1}$$

It has been seen [7] that the value of α and β change with message length l. But it is true for any message length that, $r = \beta/\alpha$, will be of order of 10^3, because of which redistribution with better distribution for the next loop becomes a reasonable alternative in the algorithm.

We assume that redistribution after execution of a loop is done as follows:

- First, all processors except 0 send their chunk of data to processor 0.
- Then, processor 0 will send again the new chunks according to new distribution to all the processors except itself.

Thus, with this communication model we can compute the redistribution cost, R using the equation, $R = 2 * (N_p - 1) * (\alpha * M/N_p + \beta)$, where N_p is the number of processors available, M is total data space (sum of sizes of the array) and α and β are as given in Equation 1. We note here that redistribution cost is independent of the way the distribution is done and so it is the same for any edge in the DDT.

4.2.2 Total Cost for the Sequence of Loops

Once the DDT is constructed we can get the total cost of the program by the shortest path algorithm of Dijkstra[6]. The shortest path algorithm from source to destination will take $\theta(N + e)$ time where N is the number of nodes and e is the number of edges.

4.3 The Communication Cost for a Program

Now, we will see how same idea can be extended to a general program.

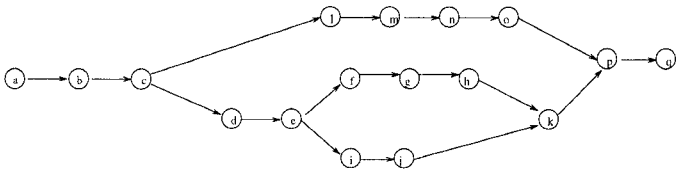

Figure 5: The structure of a general program with loops

Consider a program with the structure shown in the Figure 5. The branches are due to conditionals. For such a program we can run the above algorithm, for every sequence of loops to decide the way the arrays are to be distributed between loops. For example, for the program given in the Figure 5, there are 7 sequences, viz,$\{a,b,c\}$,$\{c,l,m,n,o,p\}$, $\{c,d,e\}$,$\{e,f,g,h,k\}$,$\{e,i,j,k\}$,$\{k,p\}$,$\{p,q\}$. Since we have minimized communication cost on every path, the overall cost of the program also gets reduced. If we run the algorithm for every path in the program from the start to the end node in the program it would minimize the cost still better, but that would take exponential amount of time since there could be exponential number of paths from the start to the end of a program. We can compute total communication cost by considering the probabilities at the junctions. At junctions there could be conflicting requirements for a loop. These can be handled ad hoc by satisfying arbitrarily any one. This needs to be studied further to get a better scheme.

5 Conclusions

Estimating communication cost for programs running on distributed memory machines is a difficult but important problem. We have a scheme to estimate the communication cost with Hyperplane Partitioning of the iteration space of a loop. Using this estimation we present a scheme for Global data partitioning of a complete program. This work needs to be extended to make cost estimation more accurate and more versatile to handle any number of dimensions of arrays.

References

[1] Ananth Agarwal, David Kranz, and Venkat Natarajan. Automatic partitioning of parallel loops and data distribution for distributed shared-memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 6(9):943–962, September 1995.

[2] Vasanth Balasundaram, Geoffrey Fox, Ken Kennedy, and Ulrich Kremer. A static performance estimator to guide data partitioning decisions. In *3rd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pages 213–223, Williamsburg, Va., April 1991.

[3] Siddhartha Chatterjee, John R. Gilbert, Robert Schreiber, and Shang-Hua Teng. Automatic array alignment in data-parallel programs. In *20th Annual ACM Symposium on Principles of Programming Languages*, pages 260–272, Charleston, S.C., January 1993.

[4] Thomas Fahringer, Roman Blasko, and Hans P. Zima. Automatic performance prediction to support parallelization of Fortran programs for massively parallel systems. In *1992 ACM International Conference on Supercomputing*, pages 347–356, Washington, D.C., July 1992.

[5] Manish Gupta and Prithviraj Banerjee. Compile-time estimation of communication costs of programs. *Journal of Programming Languages*, 5(4), September 1994.

[6] Ellis Horowitz and Sartaj Sahni. *The Fundamentals of Computer Algorithms*. Galgotia Publishers, New Delhi, 1984.

[7] R Beivide J Miguel, A Arruabarrena and J A Gregorio. Assessing the performance of the new IBM SP2 communication subsystem. *IEEE Parallel & Distributed Technology*, 81(2):12–22, Winter 1996.

[8] Ravi Kannan. An approximate random polynomial time algorithm to compute volume of complex polyhedron. *Journal of the ACM*, 6(5):1–17, January 1991.

[9] Jingke Li and Marina C. Chen. Generating explicit communication from shared-memory program references. In *Supercomputing '90*, pages 865–876, New York, November 1990.

[10] Prakash S R and Y N Srikant. Data partitioning technique for general loops for distributed memory machines. Tech rept, IISc, Dept. of Computer Science and Automation, April 1997.