01 Jan 1990

Distributed Evaluation of an Iterative Function for All Object Pairs on a SIMD Hypercube

Fikret Erçal
Missouri University of Science and Technology, ercal@mst.edu

Recommended Citation

F. Erçal, "Distributed Evaluation of an Iterative Function for All Object Pairs on a SIMD Hypercube," *Proceedings of the Fifth Distributed Memory Computing Conference, 1990*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1990.
The definitive version is available at https://doi.org/10.1109/DMCC.1990.555407

Distributed Evaluation of an Iterative Function for All Object Pairs on an SIMD Hypercube

Fikret Erçal
Department of Computer Engineering and Information Sciences
Bilkent University, Ankara, TURKEY

Abstract

An efficient distributed algorithm for evaluating an iterative function on all pairwise combinations of C objects on an SIMD hypercube is presented. The algorithm achieves uniform load distribution and minimal, completely local interprocessor communication.

1 Introduction

The problem addressed here is the following: Given a set of C objects uniformly distributed among the processors of an SIMD hypercube, and an operation on pairs of objects which may possibly modify the objects, is there a way to efficiently evaluate the operation iteratively on all the possible $C(C-1)/2$ pairwise combinations of the C objects in a distributed fashion ? This problem arises for example in the context of parallel $k-$way graph partitioning on a hypercube [1], and in the scheduling of a round-robin tournament between C players using $C/2$ courts, where the paths between courts form a hypercube interconnection. Matches between players are to be scheduled so that the courts are maximally utilized and the players do minimal walking between courts.

In an earlier study [4], a distributed solution to the problem for an MIMD hypercube was presented, and shown to be optimal with respect to processor utilization and communication. In this paper, we solve the same problem for an SIMD hypercube. Two important constraints in the iterative application of the function make the otherwise trivial problem a non-trivial one : 1) the objects might get modified by the application of the operation, (i.e. not read-only) and 2) the result of the current step depends on the state of the objects after the previous step (iterative). Since the operation can change the objects, a consisteny problem arises if multiple copies of the same object exist simultaneously in the distributed system. Therefore, only one copy of an object must be allowed in the system.

The key to an efficient distributed pairwise combining algorithm is the appropriate scheduling of communication of the objects between the processors so that all possible pairs

0-8186-2113-3/90/0000/0364$01.00 © 1990 IEEE

meet exactly once, and no redundant computations occur. To achieve this, we require each processor to communicate with only its nearest neighbors, and do some usefull work after each communication. We present a fully distributed algorithm which maximally utilizes the system and uses minimal interprocessor communication. The algorithm comprises $p+1$ phases, where p is the dimension of the hypercube. Each phase consists of two subphases - an **object-circulation** subphase, and a **window-fragmentation** subphase. **Object-circulation** subphase make use of the SIMD data circulation algorithm given in [2] with a simple modification to handle variable window sizes.

The paper is organized as follows : In section 2, we present a fully distributed algorithm using only local inter-processor communication for solving the pairwise-evaluation problem on an SIMD hypercube. In section 3, the algorithm is shown to be optimal. Section 4 concludes the paper with a brief discussion.

2 Distributed Pairwise-Evaluation on an SIMD Hypercube

We use the following notation in specifying the algorithm:

Given a processor numbered k, $0 \leq k \leq P-1$

$b_d(k)$: d−th bit of the binary representation of k
$N^d(k)$: the neighbor processor whose binary representation differs from k in only the d−th bit
$C1_k, C2_k$: objects assigned to processor k
$P = 2^p$: the number of hypercube processors
$C = 2^c$: the total number of objects

Pairwise-Evaluation Algorithm listed below evaluates a given function for all $C(C-1)/2$ pairwise combinations of C objects using $C/2$ processors. Initially, each processor P_k contains two of the C objects, labeled $C1_k$ and $C2_k$, with no two processors containing the same object. The processors alternate between computation and communication, with each processor repeatedly performing: 1) a pairwise operation on the two locally held objects, and, 2) communication of one of the objects to a neighbor processor, in turn receiving some other object from a neighbor.

SIMD Distributed Pairwise-Evaluation Algorithm :

Processor P_k executes:

```
1. for d ← p to 0 do
2.   for s ← 1 to 2^d − 1 do
3.       operate on the pair (C1_k, C2_k);
4.       send(C2_k, N^{h(d,s)}(k));
5.       recv(C2_k, N^{h(d,s)}(k));
6.   endfor
7.       operate on the pair (C1_k, C2_k);
8.       if (d > 0) then
9.         if (b_{d-1}(k) = 1) then
10.          send(C1_k, N^{(d-1)}(k));
11.          recv(C1_k, N^{(d-1)}(k));
12.        else
13.          send(C2_k, N^{(d-1)}(k));
14.          recv(C2_k, N^{(d-1)}(k));
15.        endif
16.      endif
17. endfor
```

The key requirement is that the objects be moved between the processors in such a way

that each possible pair of objects comes together exactly once to enable the application of the pairwise operation on that pair. The algorithm has $p+1$ phases (indexed by "d"), where p is the number of dimensions of the hypercube. Each phase consists of two subphases - an **object-circulation** subphase where processors circulate their objects in closed windows (lines 2-6), and a **window-fragmentation** subphase where each window subdivides into two isolated windows (lines 8-16). The window structure thus changes from phase to phase, with 2^{p-d} independent windows of size 2^d being formed during phase d, as illustrated for a 4-dimensional hypercube in Fig. 1.

For an MIMD hypercube, object-circulation in a window of size 2^d can be easily done by repeatedly performing, 2^d-1 times, a circular shift of 1 among the processors belonging to the same window. On the other hand, due to the central control in an SIMD hypercube, optimal circulation requires a special exchange sequence X_d as described in [3]. This sequence is defined recursively as in the following:

$$X_1 = 0, \quad X_d = X_{d-1}, d-1, X_{d-1} \quad (d > 1)$$

For example, $X_3 = 0,1,0,2,0,1,0$. Using X_d sequence, object circulation in a window of size 2^d is achieved by first circulating data in windows of size 2^{d-1} in parallel using X_{d-1} sequence, then performing a data exchange across the two windows (along bit d-1), and finally circulating the exchanged data in the two windows again using X_{d-1} sequence. In the algorithm given above, the notation $h(d,i)$ is used to denote the $i-th$ number in the sequence X_d, $1 \leq i \leq 2^d$. As an example, $h(3,1) = 0$, $h(3,2) = 1$, $h(3,3) = 0$, and $h(3,4) = 2$.

During a phase, corresponding to one iteration of the d-loop of the algorithm, each processor keeps one of its objects (C1) local, while it repeatedly receives, transforms and passes on the second object (C2). Considering phase p, with all processors communicating in one single window, at the end of the 2^p-1 steps in the first part (the object-circulation subphase) of the phase, all objects constituting the various $C1_k$'s (denoted $\mathcal{CS}1$) would have been matched up with respect to every object in the $\mathcal{CS}2$ set (and the pairwise operation performed on each such generated pair). Thus the only pairings between objects that have not yet been formed are between the members of the $\mathcal{CS}1$ set and likewise, mutually among the members of $\mathcal{CS}2$. The window-fragmentation subphase of phase p involves pairwise communication exchanges between each processor and its neighbor whose address differs in the highest bit. During this subphase, each processor P_k with highest address bit of one ($b_{p-1}(k)$=1), swaps its C1 object for the C2 object of its partner processor(P_l, with $b_{p-1}(l)$=0). Thus, after this communication subphase, all processors P_k with ($b_{p-1}(k)$=1), will only have objects from the original $\mathcal{CS}2$ set, while all processors with ($b_{p-1}(k)$=0) will have all the objects comprising the original $\mathcal{CS}1$ set. This subphase is labeled the "window-fragmentation subphase" because the window gets fragmented into two smaller windows and no communica-

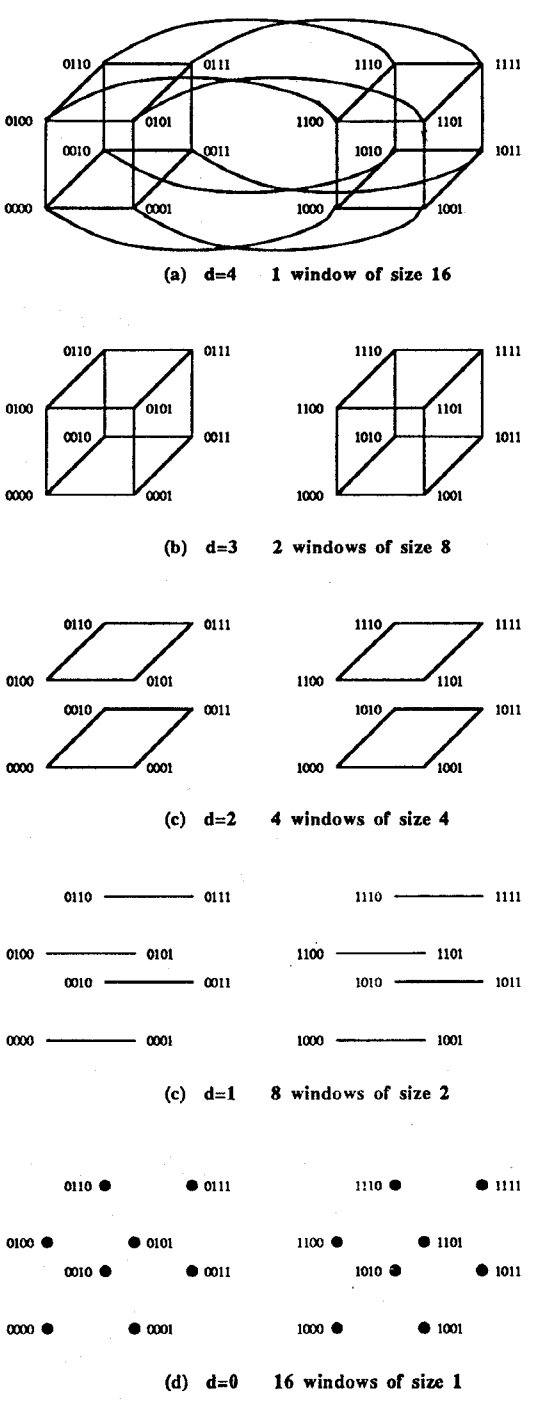

Figure 1: Illustration of window formation in different phases of the Distributed PC algorithm

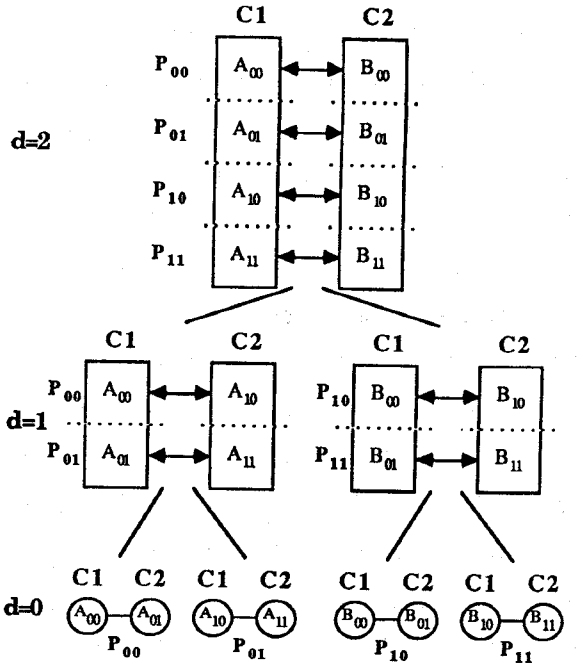

Figure 2: Illustration of Distributed PC algorithm on a 2-D hypercube (4 processors)

tion takes place thereafter between the processors in the "highest-bit-1" window and those in the "highest-bit-0" window. Thus in phase $(p-1)$, two windows of size 2^{p-1} are formed for the object-circulation subphase and communication occurs between processors differing in their $(p-2)$th bit during the window-fragmentation subphase.

During each phase of the algorithm, new object-pairs meet at the processors, for application of the pairwise operation. The algorithm guarantees that during an outer pass, no pair of objects is ever matched up more than once. Fig. 2 is used to illustrate this "**no-repetition**" property of the algo-

rithm. In order to focus on the nature of the window-fragmentation subphase, the effects of the alternating object-circulation subphase are intentionally omitted. Eight objects are shown, mapped onto four processors, two objects per processor. During phase 2 ($d = 2$), the application of the object-circulation subphase results in the generation of all possible pairwise combinations with one object from $\mathcal{CS}1$ (A_{00},A_{01},A_{10},A_{11}) and the other from $\mathcal{CS}2$ (B_{00},B_{01},B_{10},B_{11}). Ignoring for now the actual permutation of the C2 objects that will result at the end of the object-circulation subphase, and assuming it to be as shown, the window-fragmentation subphase of phase 2 will result in the state shown for $d = 1$. Processors P_{00} and P_{01} are left with objects A_{00},A_{01},A_{10},A_{11}, whereas P_{10} and P_{11} now have objects B_{00},B_{01},B_{10},B_{11}. After the window-fragmentation phase of phase 2, P_{0*} and P_{1*} do not ever again communicate with each other. Since no pairwise combinations involving two A-objects had occurred during phase 2, and since none of the B-objects can any longer meet any of the A-objects, all pairs of objects that align at any processor are unique combinations that have not occurred earlier. The same property clearly holds recursively, as illustrated in the figure.

In the next section, we formally prove the correctness of the distributed algorithm.

3 Proof of Optimality

Lemma 1 *The total number of pairwise object combinations that occur during execution of the algorithm is* $C(C-1)/2$.

Proof: Each processor performs one pairwise comparison during every step of every phase of the algorithm, as is clear from the algorithm specification. The number of steps in phase d is 2^d. Hence the total number of pairwise combinations tried is:

$$\begin{aligned} 2^p * \sum_{d=p}^{0} 2^d &= 2^p * (2^{(p+1)} - 1) \\ &= P(2P-1) = C(C-1)/2 \end{aligned}$$

□

Lemma 2 *Given any objects C_i and C_j, the combination (C_i,C_j) can occur at most once during execution of the algorithm.*

Proof: Let d be the earliest phase that the combination (C_i,C_j) occurs. Obviously, at most one such match can occur during the object-circulation subphase of phase d. For such a match to occur, one of them must belong to the C1-object-set and the other to the C2-object-set. Since they belong to different object-sets, during the window-fragmentation subphase of phase d, C_i and C_j will necessarily end up in processors P_k, P_l, where l and k differ at least in bit $d-1$, and hence P_k and P_l belong to different windows. Obviously, they cannot get matched in any later phase $d' < d$. Hence at most one match (C_i,C_j) can occur during an outer pass. □

Theorem 1 *Given any two objects C_i and C_j, the pairwise combination (C_i,C_j) occurs exactly once during execution of the algorithm.*

Proof: Theorem 1 follows immediately from lemma 1 and lemma 2. By lemma 1, a total of $C(C-1)/2$ pairwise combinations occur, and by lemma 2, no combination (C_i,C_j) can occur more than once. Since the number of possible distinct combinations of object pairs is $C(C-1)/2$, all possible matches must occur exactly once during execution of the algorithm. □

Theorem 1 implies that as regards to computation, the algorithm is optimal since every processor is busy during each computational step and no duplicate computations occur. With respect to communication too, under the constraint of computational load balancing and uniform data distribution, each processor can only contain two objects, and after performing the pairwise operation on its currently held pair, it will have to send out at least one object and receive one object in order to perform useful computation at the next step. The algorithm causes only one object to be sent and one object received by each processor at each step, i.e., the algorithm performs minimal communication.

4 Discussion

An efficient distributed algorithm for evaluating an iterative function on all pairwise combinations of C objects on an SIMD hypercube is presented, and it is shown to achieve uniform load distribution and minimal, completely local inter-processor communication.

In case that $C > 2P$, the algorithm can be extended in a straightforward fashion. For $C = MP$, $M = 2k$, $k > 1$, groups of $M/2$ objects should be considered in place of single objects in the presented algorithm. Now, instead of a single pairwise operation, $(M/2)^2$ pairwise operations are performed at each step of the algorithm between member partitions of the two $(M/2)$-ary object-groups in a processor. With such a $(M/2)-ary$ group of objects in place of single objects, the algorithm for distributed PC is essentially the same as above, except for an additional set of operations between the components of each $(M/2)-ary$ group of objects.

References

[1] P. Sadayappan, F. Ercal and J. Ramanujam, "Parallel Graph Partitioning on a Hypercube," *Proc. of Fourth Conf. on Hypercube Concurrent Computers and Applications*, March, 1989.

[2] S.Ranka and S. Sahni, *Hypercube Algorithms For Image Processing and Pattern Recognition,* Bilkent University Lecture Notes, Springer-Verlag, in press.

[3] E. Dekel, D. Nassimi, and S. Sahni, "Parallel Matrix and Graph Algorithms," *SIAM Journal on Computing,* 1981, pp. 657-675.

[4] P. Sadayappan, F. Ercal and J. Ramanujam, "Distributed Generation of Pairwise Combinations on a Hypercube," in *Proceedings of Parallel Computing 89,* Leiden, The Netherlands, August 1989.