INFORMATIK

The Factor Algorithm for All-to-all Communication on Clusters of SMP Nodes

Peter Sanders[1] and Jesper Larsson Träff

MPI–I–2002–1-008 February 2002

FORSCHUNGSBERICHT RESEARCH REPORT

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Stuhlsatzenhausweg 85 66123 Saarbrücken Germany

Authors' Addresses

Peter Sanders
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
sanders@mpi-sb.mpg.de, http://www.mpi-sb.mpg.de/~sanders/

Jesper Larsson Träff
C&C Research Laboratories, NEC Europe Ltd.
Rathausallee 10, 53757 Sankt Augustin, Germany
traff@ccrl-nece.de

Abstract

We present an algorithm for *all-to-all personalized communication*, in which every processor has an individual message to deliver to every other processor. The machine model we consider is a cluster of processing nodes where each node, possibly consisting of several processors, can participate in only one communication operation with another node at a time. The nodes may have different numbers of processors. This general model is important for the implementation of all-to-all communication in libraries such as MPI where collective communication may take place over arbitrary subsets of processors. The algorithm is simple and optimal up to an additive term that is small if the total number of processors is large compared to the maximal number of processors in a node.

Keywords

parallel computing, collective communication, personalized all-to-all, MPI, hierachical interconnection networks

1 Introduction

A successful approach to parallel programming is to write a sequential program executing on all processors and delegate interprocessor communication and coordination to a communication library such as MPI [13]. Within this approach, many parallel computations can be conveniently expressed in terms of a small number of *collective communication operations*, where "collective" means that a subset of processors is cooperating in a nontrivial way. One such frequently used collective communication operation is *regular personalized all-to-all message exchange*: Each of p processors has to transmit a personalized message to itself and each of $p-1$ other processors, i. e., for every pair of processor indices i and j a message m_{ij} has to be sent from processor i to processor j. In *regular* all-to-all exchange, all messages are assumed to have the same length. Examples of subroutines using all-to-all communication are matrix transposition and FFT.

This paper presents an algorithm for regular all-to-all communication on clusters of processing nodes where each node may consist of several processors. We assume that only a single processor from each node can be involved in inter-node communication at a time. Prime examples of such hierarchical systems are clusters of SMP nodes, where processor groups of 2–16 processors communicate via a shared memory, and where some medium to large number of nodes are interconnected via a commodity interconnection network. For example, the Earth Simulator[2] and the NEC SX-6 supercomputer[3] have up to 8 processors per node; the IBM SP POWER3 allows up to 16 processors per node.[4]

The difficult case is when nodes have differing numbers of processors participating in the all-to-all exchange. This situation must be handled efficiently in a high-quality communications library because it arises naturally if a job is assigned only part of the machine, or if the exchange is only among a subset of the processors in a job.

We use a simple machine model that allows an efficient implementation portable over a large spectrum of platforms. The nodes are assumed to be fully connected. Communication is *single ported* in the sense that at most one processor per node can communicate with a processor on another node at a time. The single-ported assumption is valid for current interconnection technologies like Myrinet, Giganet, the Scalable Coherent Interface (SCI), or for the crossbar switch used on the NEC machines and the Earth Simulator.

[2] `http://www.es.jamstec.go.jp/`

[3] `http://www.ess.nec.de/sx-6.html`

[4] `http://www-1.ibm.com/servers/eserver/pseries/hardware/largescale/sp.html`

Our algorithm for all-to-all communication extends a well-known algorithm for non-hierarchical systems based on *factoring* the complete graph into matchings. The new algorithm is optimal with respect to the time a processor spends waiting or transmitting data up to an additive term that is bounded by the time needed for data exchange inside a node. This time is comparatively small if the total number of processors is large compared to the maximum number of processors in a node. Our algorithm runs in phases, in each phase getting rid of nodes with the minimum number of processors among the surviving nodes. The main issue is to balance the communication volume of nodes with many processors over the phases so that the number of communication steps is minimized.

Related Work

All-to-all communication has been studied intensively, and we mention only a sample of the known results. Most work focuses on non-hierarchical systems with specific interconnection networks [11, 5, 14]. Trade-offs between communication volume and number of communication start-ups were studied in [3, 5], which achieve algorithms that are faster for small messages.

A well-known version of the factor algorithm in which processor j communicates with processor j `xor` i in step i works if the number of processors is a power of two. General 1-factorizations of the complete graph [9, 6] can be used for arbitrary numbers of processors (see Section 2). 1-factorizations have also been used for constructing decompositions of the complete graph into permutations that can be efficiently executed on mesh-like networks [11, 5], and for all-to-all communication on fully connected networks [10].

Our variant of the single-ported communication model is similar to the *telephone model* [2, 1]. Collective communication on hierarchical systems has recently received some attention [12, 8, 7]. Huse [7] reports experiments with an implementaion of (homogeneous) all-to-all which ensures that only one processor per node is involved in inter-node communication at a time. Algorithmic details and properties are not stated.

2 The non-hierarchical factor algorithm

The basis for our algorithm is a well-known algorithm for the single-ported, non-hierarchical case. We briefly give our version of the algorithm here. It exploits the existence of a 1-*factorization* of the complete graph. Since our construction requires to include self loops into the graph, whereas the usual construction has no self-loops. We give the construction and the proof here.

Interestingly, self-loops *simplify* the construction. [5]

Lemma 1 *Let G be the complete graph with p vertices including self-loops. G is* 1-factorizable*, i. e., $G = (V, E)$ can be decomposed into p subgraphs $G^i = (V, E_i), i = 0, \ldots, p-1$ in which each vertex has degree 1 (*1-factors*).*

PROOF: Let $V = \{0, \ldots, p-1\}$.

The ith factor $G^i = (V, E_i)$, is constructed as follows. For $u \in V$ define $v^i(u) = (i-u) \bmod p$. Define $E_i = \{(u, v^i(u)) | u \in V\}$. Since $v^i(v^i(u)) = (i-((i-u) \bmod p)) \bmod p = u$ all vertices have degree exactly one. Furthermore, any edge $(u, v) \in E$ will find itself in some factor, namely in factor $G^{(u+v) \bmod p}$. In particular, the self-loop (u, u) will find itself in $G^{2u \bmod p}$. ■

We can now formulate the non-hierarchical *factor algorithm* that is the basis for our hierarchical algorithm explained in the next section. It requires p communication rounds for any number of p processors. In the ith round, processors u and v that are neighbors in G^i are *paired* and exchange messages m_{uv} and m_{vu}.

for $i = 0, \ldots, p-1$ **do** // round
  Let $G^i = (V, E_i)$ be the ith "factor"
  for $(u, v) \in E_i$ **pardo** exchange(u, v) // step

procedure exchange(u, v):
// use a total processor ordering $u < v$ to order exchange between u and v
if $u < v$ **then** send m_{uv} from u to v and receive m_{vu} from v to u **else**
if $u > v$ **then** receive m_{vu} from v to u and send m_{uv} from u to v **else**
  copy m_{uu} from source buffer of u to destination buffer of u

The factor algorithm is optimal in the sense that in every step, each processor sends data it needs to send or receives data it needs to receive.

3 All-to-all communication on clustered, hierarchical systems

We now generalize the factor algorithm to clustered, hierarchical systems. Let N be the number of processor nodes, and let G denote the N-node complete graph with self-loops. Let G_A denote the subgraph of G induced by a subset of nodes A, and G_A^i the ith 1-factor of G_A. We use U and V to

[5] Without self-loops some special treatment for even P saves one factor.

Algorithm ClusteredAllToAll:

$A \leftarrow \{0, \ldots, N-1\}$ // set of *active* nodes
done $\leftarrow 0$
while $A \neq \emptyset$ **do** // phase
  loop invariant: $\forall (U, V) \in G : \forall u \in U, v \in V :$
    $U \preceq V \wedge 0 \leq l(u) < \text{done} \Rightarrow m_{uv}$ and m_{vu} have been delivered
  current $\leftarrow \min\{\text{size}(U) \mid U \in A\}$
  for $i = 0, \ldots, |A| - 1$ **do** // round
    for all $(U, V) \in G_A^i$ where $U \preceq V$ **pardo**
      for each $u \in U, \text{done} \leq l(u) < \text{current}$ **do**
        for each $v \in V, \text{done} \leq l(v) < \text{size}(V)$ **do** // step
          if $U = V$ **then** send m_{uv} from u to v
          else exchange(u, v)
  done $\leftarrow$ current
  $A \leftarrow A \setminus \{U \mid \text{size}(U) = \text{done}\}$

Figure 1: The clustered, hierarchical factor algorithm.

denote processor nodes of the system, and u and v for individual processors. By size(U) we denote the number of processors in node U, and by $l(u)$ the *local index* of processor u within its node, $0 \leq l(u) < \text{size}(U)$ for $u \in U$. To specify what messages should be exchanged when two nodes U and V are paired we impose a node ordering as follows:

$$U \preceq V \quad \text{if} \quad \text{size}(U) < \text{size}(V), \text{or } \text{size}(U) = \text{size}(V) \wedge U \leq V$$

where $U \leq V$ relates to an arbitrary total ordering of the nodes. The algorithm is shown in Figure 1 using this notation.

The outermost loop iterates over a number of phases, each of which considers a 1-factorization of the set of *active nodes* A that have not yet exchanged all their messages. The second loop iterates over the 1-factors G_A^i of G_A. The parallel loop considers all node pairs (U, V) that are neighbors in the given 1-factor G_A^i. The node ordering $U \preceq V$ is used to conveniently describe the message exchange between processors on node U and processors on nodes V necessary for reestablishing the invariant for the outermost loop after 'done' has been increased to 'current'. When $U = V$, the bidirectional exchange is replaced by a unidirectional send because otherwise, intra-node messages would be transmitted twice. Sending m_{uu} from u to u means copying m_{uu} from source buffer to destination buffer of u. Figure 2

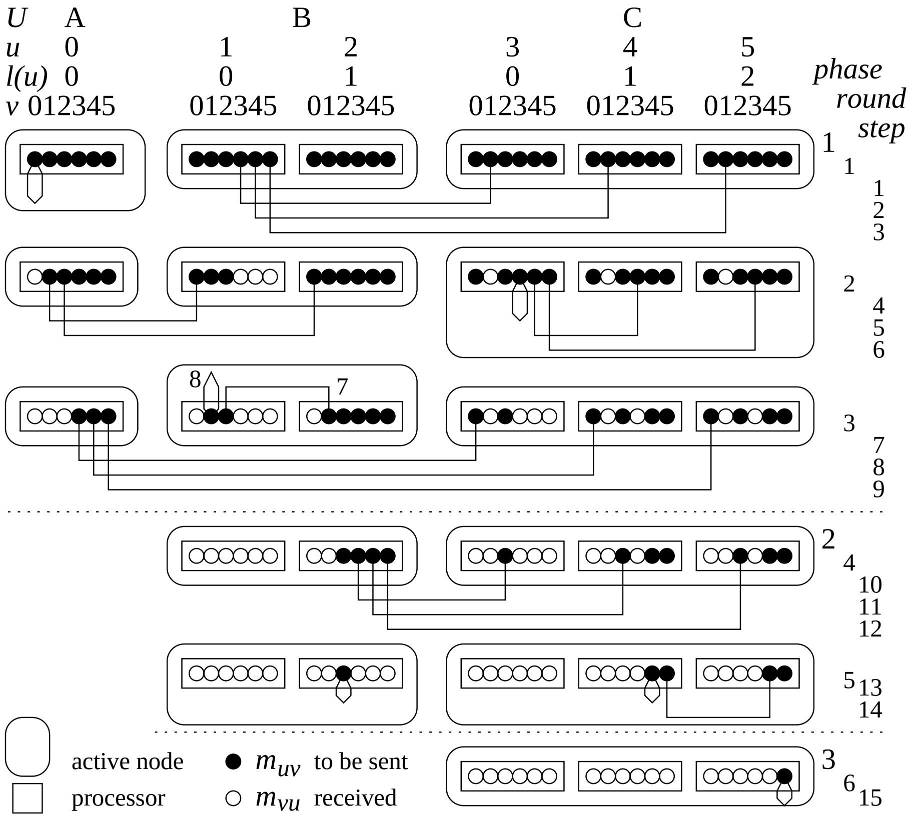

Figure 2: Execution of algorithm ClusteredAllToAll for three nodes with size 1, 2, and 3 respectively. The algorithms goes through 3 phases, and 3, 2 and 1 rounds respectively are required, for a total of 15 steps.

gives an example of the operation of the ClusteredAllToAll algorithm, and the following theorem formally states its correctness and in which sense it is close to optimal.

Theorem 1 *Algorithm ClusteredAllToAll performs a personalized all-to-all exchange in a number of steps equal to the maximal number of messages that the processors in a node have to send.*

PROOF: (Outline) Regarding correctness, let $0 = S_0 < S_1 < \ldots < S_k$ be the sequence of different node sizes. The algorithm performs k phases. In phase i nodes U with $\text{size}(U) \geq S_i$ are active. In particular, the outer loop terminates. Furthermore, at the end of the algorithm done $= \max_{U \in \{0,\ldots,N-1\}} \text{size}(U)$. The loop invariant implies that all messages have been exchanged.

The bound on the number of steps follows since all nodes with the maximum number of processors are participating in a communication in every step. ■

The reason why the algorithm is not optimal in all cases is that a node paired with itself communicates only unidirectionally in each step. If at the same step two other nodes with maximum number of processors are paired, they communicate bidirectionally and hence take longer to complete a round. This is not optimal since at least in some cases there are schedules which avoid such situations. However, there are only few such inefficient steps: Consider a node U with maximal number $n = |U|$ of processors. Our algorithm performs pn steps. At most n^2 of these steps — a fraction of p/n — can be inefficient for node U. Hence, the inefficient steps are few compared to the efficient steps for $p \gg n$.

Although the algorithm was formulated for single-ported communication using 1-factorizations, generalizations to multi-ported communications are possible. The same basic scheme applies if the complete graph is decomposed into graphs with degrees at most k or into permutations (directed cycles). Decomposition into permutations is particularly interesting since several all-to-all algorithms for non-fully connected networks are known that are based on this approach [11, 5, 14].

4 Implementation issues

The algorithm is easy to implement, and has been used for an implementation of the `MPI_Alltoall` function of the Message Passing Interface (MPI) [13] for clusters of SMP nodes.

On most SMP clusters intra-node communication is considerable faster than communication between processors on different nodes. This can be exploited to reduce the completion time somewhat by doing more (intra-node) work in rounds where a node is paired with itself, thus possibly saving rounds in subsequent phases where $|A|$ is either even or one.

A prototype implementation has been done within the framework of the well-known MPICH implementation [4]. For each process, the data structure representing the set of processes that can communicate with each other – the *communicator* construct of MPI – is extended with an array containing the processes sorted after SMP node id, and an array of node sizes. This information can be computed once and for all when the communicator is created. Our algorithm thus only performs simple loops, array lookups and message passing operations.

Since we do not have access to a machine with large nodes yet we only performed preliminary performance studies on a small Giganet SMP cluster with 6 nodes of 4 processors each. We experimented with various distributions of processes among the nodes (regular, organ pipe, alternating full/small nodes, various random distributions). We made comparisons with the trivial, native MPICH algorithm for `MPI_Alltoall` which posts p non-blocking, concurrent send and receive operations on each processor. The new algorithm never performs worse and achieves at least 10% and sometimes up to 20% higher bandwidth when the number of active processes per SMP varies. We remark that the implementation within MPICH permits only crude control over the utilization of the inter-node network, since we cannot immediately access a centralized, per-node queue of communication requests. It is likely that more efficient, low-level, centralized contention control will lead still to better performance for our algorithm.

References

[1] Bar-Noy, Kipnis, and Schieber. Optimal multiple message broadcasting in telephone-like communication systems. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 100:1–15, 2000.

[2] D. Barth and P. Fraigniaud. Approximation algorithms for structured communication problems. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 180–188, Newport, Rhode Island, June 22–25, 1997. SIGACT/SIGARCH and EATCS.

[3] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1143–1156, 1997.

[4] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable imlementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[5] S. E. Hambrusch, F. Hameed, and A. A. Khokar. Communication operations on coarse-grained mesh architectures. *Parallel Computing*, 21:731–751, 1995.

[6] F. Harary. *Graph Theory.* Addison-Wesley, 1967.

[7] L. P. Huse. MPI optimization for SMP based clusters interconnected with SCI. In *7th European PVM/MPI User's Group Meeting*, volume 1908 of *Lecture Notes in Computer Science*, pages 56–63, 2000.

[8] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'2000)*, pages 377–384, 2000.

[9] D. König. *Theorie der endlichen und unendlichen Graphen*. Akademische Verlagsgesellschaft, 1936.

[10] P. Sanders and R. Solis-Oba. How helpers hasten h-relations. *Journal of Algorithms*, 2001. To appear.

[11] D. S. Scott. Efficient all-to-all communication patterns in hypercube and mesh topologies. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 398–403, 1991.

[12] S. Sistare, R. vandeVaart, and E. Loh. Optimization of MPI collectives on clusters of large-scale SMPs. In *Supercomputing*, 1999. `http://www.supercomp.org/sc99/proceedings/techpap.htm#mpi`.

[13] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The Complete Reference*, volume 1, The MPI Core. MIT Press, second edition, 1998.

[14] Y. Yang and J. Wang. Optimal all-to-all personalized exchange in self-routable multistage networks. *IEEE Transactions on Parallel and Distributed Systems*, 11(3):261–274, 2000.