Heuristic Search of Optimal Reduction Schedule in Heterogeneous Cluster Environments

Pangfeng Liu Tzu-Hao Sheng
Chih-Hsuae Yang
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan, R.O.C.
{pangfeng,sth88,ych88}@cs.ccu.edu.tw

Abstract

Network of workstation (NOW) is a cost-effective alternative to massively parallel supercomputers. As commercially available off-the-shelf processors become cheaper and faster, it is now possible to build a PC or workstation cluster that provides high computing power within a limited budget. However, a cluster may consist of different types of processors and this heterogeneity within a cluster complicates the design of efficient collective communication protocols, and it is a very hard combinatorial problem to design an optimal reduction protocol. Nevertheless, we show that a simple exchange lemma can greatly reduce seraph space so that we can find the optimal solution efficiently. Combined with standard branch-and-nound technique the search space is reduced to be less than one percent of original size for most practical cluster size.

1 Introduction

Network of workstation (NOW) is a cost-effective alternative to massively parallel supercomputers [2]. As commercially available off-the-shelf processors become cheaper and faster, it is now possible to build a PC or workstation cluster that provides high computing power within a limited budget. High performance parallelism is achieved by dividing the computation into manageable subtasks, and distributing these subtasks to the processors within the cluster. These off-the-shelf high-performance processors provide a much higher performance to cost ratio so that high performance cluster can be built inexpensively. In addition, the processors can be conveniently connected by industry standard network components. For example, fast ethernet technology provides up to 100 mega bits per second of bandwidth with inexpensive fast ethernet adaptors and hubs.

In parallel to the development of inexpensive and standardized hardware components for NOW, system software for programming on NOW is also advancing rapidly. For example, the *Message Passing Interface* (MPI) library has evolved into a standard for writing message-passing parallel code [4, 5, 1]. An MPI programmer uses a standardized high-level programming interface to exchange information among processes, instead of native machine-specific communication library. An MPI programmer can write highly portable parallel code and run it on any parallel machine (including network of workstation) that has MPI implementation.

Most of the literature on cluster computing emphasizes on *homogeneous* cluster – a cluster consisting of the same type of processors. However, we argue that heterogeneity is one of the key issue that must be addressed in improving parallel performance of NOW. First it is always the case that one wish to connect as many processors as possible into a cluster to increase parallelism and reduce execution time. Despite the increased computing power, the scheduling management of such a *heterogeneous network of workstation* (HNOW) becomes complicated since these processors will have different performance in computation and communication from one another. Secondly, since most of the processors that are used to build a cluster are commercially off-the-shelf products, they will very likely be outdated by faster successors before they become unusable. Very often a cluster will consist of "leftovers" from the previous installation, and "new comers" that are recently purchased. The issue of heterogeneity is both scientific and economic.

Any workstation cluster, be it homogeneous or heterogeneous, requires efficient collective communication [3]. For example, a barrier synchronization is often placed between two successive phases of computation to make sure that all pro-

cessors finish the first phase before anyone goes to the next. In addition, a scatter operation distributes input data from the source to all the other processors for parallel processing, then a global reduction operation combines the partial solutions obtained from individual processors into the final answer. The efficiency of these collective communication will affect the overall performance, sometimes dramatically.

Heterogeneity of a cluster complicates the design of efficient collective communication protocols on it. When the processors send and receive messages at different rates, it is difficult to synchronize them so that the message can arrive at the right processor at the right time for maximum communication throughput. On the other hand, in homogeneous NOW every processor requires the same amount of time to transmit a message. For example it is straightforward to implement a reduction operation as a series of sending and receiving messages, and in each phase we reduce the number of processors that have received the combined result from other processors. In a heterogeneous environment it is no long clear how we should proceed to complete the same task.

It is a very hard optimization problem to find an optimal reduction schedule for heterogeneous clusters. In practice we often resort to dynamic programming or branch-and-bound search[3]. This paper shows that a simple technique called *processor exchange* is very effective in reducing the number of nodes we have to search in the branch-and-bound process. In addition, we introduce a simple *slowest-node-first* schedule which, despite that it does not guarantee optimality, performs reasonably well in practice.

The rest of the paper is organized as follows. Section 2 describe the communication model in our treatment of reduction problem in HNOW. Section 3 describes the concept of earliest possible schedule and Section 4 describes a slowest-node-first heuristic for reduction in heterogeneous cluster. Section 5 states the exchange lemma, Section 6 shows the experimental results, and Section 7 concludes.

2 Communication Model

The communication model is defined as follows. The system consists of a set of n processors $\{p_0, \cdots p_{n-1}\}$, each is capable of direct point-to-point communication to one another. A processor p_i is characterized by its transmission time $t(p_i)$, i.e. the time it takes for p_i to send a message to any other processor.

In order to make the communication model realistic, we further assume that two communications cannot overlap, i.e. neither the sender nor the receiver can engage in any other communication at the same time. This assumption is based on that in practice most of the workstation clusters are connected by non-share-able communication media, like ethernet. As a result, the algorithms designed under this model will be useful in practice.

Based on the communication model, we define the reduction problem. Suppose each processor in the system has an unit of information, and we would like to combine all these informations into the final answer, how do we schedule the processors so that the reduction takes the least amount of time? For example, each processor may have a number and we want compute the total sum of these numbers by message passing. For a homogeneous system a simple tree algorithm can be used so that during each iteration half of the processors send their information to the other where the information will be combined. The algorithm takes $\log n$ rounds to combine all the informations since the number of active processors reduces by half per iteration. However, due to the variance in communication speed, this algorithm cannot guarantee minimum reduction time.

We formally define the reduction as a scheduling problem. We observe that during the reduction process exactly $n-1$ messages will be sent by $n-1$ different processors. As a result the slowest processor should not send any message – instead it should only receive message and compute the final reduction result. Based on this observation, we define a reduction schedule as mapping function from a processor to the time that it starts sending its message. Formally we define the scheduling function s so that for any processor p[1], p starts and completes sending its message at time $s(p)$ and $s(p) + t(p)$ respectively. The interval $[s(p), c(s,p)]$ is defined as the *transmission window* of p, where $c(s,p) = s(p) + t(p)$ is the *completion time* of p under a schedule s.

Due to the constraint that a processor can participate a single communication at any given time, we must distinguish valid schedule from invalid ones. To formalize this constraint, we define two sets of processors for any schedule at any given time. Let $A(s,t)$ be the number of processors that are still actively sending their messages, and $C(s,t)$ be the number of processors that have completed sending their messages at time t under schedule s.

[1]Except the slowest processor, which does not send any message.

$$
\begin{aligned}
A(s,t) &= |\{p_i | s(p_i) \leq t < c(s,p_i)\}| \\
C(s,t) &= |\{p_i | t \geq c(s,p_i)\}| \\
2A(s,t) + C(s,t) &\leq n \qquad (1)
\end{aligned}
$$

A schedule function s is valid if and only if for all time t Inequality 1 is true. The inequality means that at any time t, the total number of senders and receivers, and those processors that have sent out their messages, should not be more than the number of processors in the system. Figure 1 show a valid schedule and the corresponding A and C functions.

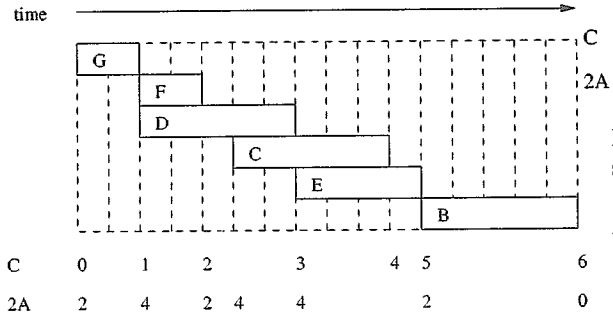

Figure 1: An valid schedule example showing the A and C functions. The system consist of seven processors – indicated by the uppercase letters from A to G. The transmission time for these seven processors are 10, 5, 5, 5, 4, 2, 2, respectively.

3 Earliest-Possible Schedule

This section describe a technique called *earliest possible* scheduling that can "normalize" all the possible valid schedules. We use this canonical form to simplify the discussion of finding the optimal schedule.

An earliest possible (EP) schedule is one in which all the communication are initiated as earlier as possible. A new communication can be initiated as soon as the number of *free processors* reaches 2 – one for the sender and one for the receiver.

Note that it is trivial to convert any valid schedule into an EP schedule without increasing the total time – we just move the transmission window of each processor as early as possible. As a result EP schedule servers as a canonical form in which we will limit our search of optimal schedule. Since the EP algorithm can completely determine a schedule once the order of processors in the sequence P is fixed, we only have to consider up to $(n-1)!$ different schedules. Figure 2 shows the EP schedule that is converted from the schedule in Figure 1, and Figure 3 shows that the algorithm EP will assign non-decreasing start time to the processors in the order they appear in the input processor sequence P.

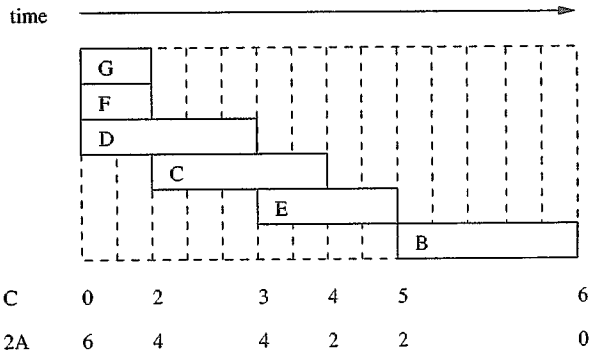

Figure 2: The EP schedule converted from the schedule in Figure 1

```
Algorithm EP(P)
{
   i = 1
   time = 0
   free = the number of processors in P
   Active = empty set
   Complete = empty set

   while (i <= n-1)
      do while free >= 2
         set time to be the start time of
         the ith processor in P;
         add P_i into Active;
         i = i + 1;
         free = free - 2;

      find the processor p among Active that
      has the smallest completion time;
      time = the completion time of p;
      move q from Active to Complete;
      free = free + 1;
}
```

Figure 3: The earliest possible scheduling algorithm.

4 Slowest-Node-First Scheduling

We introduce a simple scheduling method called *slowest-node-first* (SNF) for the reduction problem. SNF simply sorts the processors in P in non-increasing order, and give the sorted sequence $P = (p_1, p_2, ...p_{n-1})$[2] to algorithm EP. In the

[2] $t(p_i) \geq t(p_j)$ if $i < j$

next section we show that this simple technique is very effective in obtaining a good reduction schedule. Figure 4 shows the slowest-node-first schedule with the same cluster as in Figure 2.

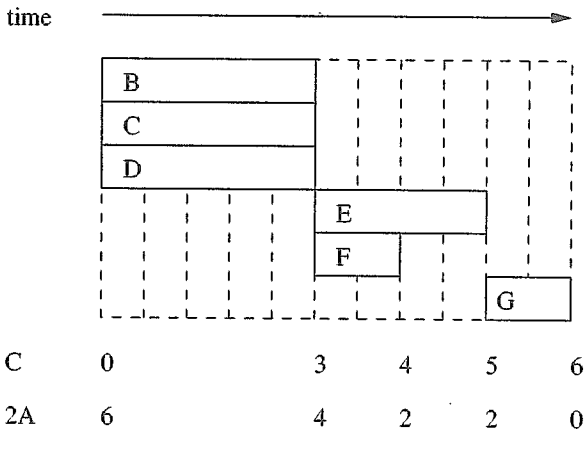

Figure 4: Slowest-node-first scheduling result from the same cluster as in Figure 1 and 2.

The rational of having the slowest processors to send message first is as follows. At the beginning of the reduction process, we would like to overlap as many communication as possible. Intuitively we let all the slow processors send first so that they will overlap with each other, instead of having to wait for each other at the end of the reduction.

5 Exchange Lemma

This section describe the exchange lemma that clarifies our intuition that slow processors should send first, as we did in SNF scheduling.

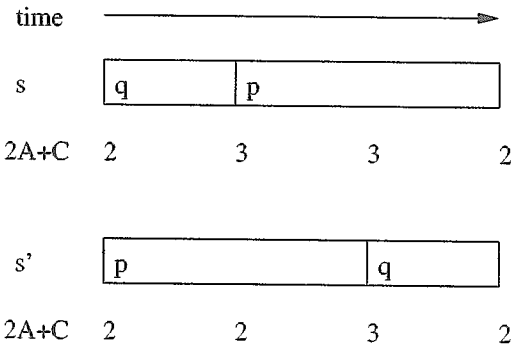

Figure 5: An illustration on the contribution to the sum of C and $2A$.

Lemma 1 *Let s be a valid schedule that $s(p) = c(s,q)$, i.e. p starts right after q ends. If $t(p) > t(q)$ then we can exchange p and q so that the modified schedule s' in which $s'(p) = s(q)$ and $s'(q) = c(s',p)$, is also valid.*

Proof. From Figure 5 we observe that the contribution of P_i and P_j to the sum of C and A functions is always higher in s than in s', therefore if s can satisfy Ineqality 1, so can s'. ■

From Lemma 1 it follows immediately that in the search of optimal reduction schedule we can neglect those schedules that have a slower sender p waiting for any faster processor q to complete. There are two cases to consider. First if $s(p) = c(s,q)$ then by Lemma 1 we can switch p and q. On the other hand, if there is a gap between the transmission window of p and q, then we can delay the transmission of q so that it ends right where p starts. We can do so because there is no new transmission initiated between $s(p)$ and $s(q)+t(q)$. As a result we will consider only those schedules that all the senders wait for slower processors only.

Corollary 1 *There exists an optimal schedule such that every processor waits for slower processors only.*

Figure 6 shows a counterexample that SNF always gives optimal reduction time, even in a simple cluster consisting of two types of processors. This cluster has 4 slow processors with transmission time x, and 8 fast processors with transmission time 1. The alternative schedule requires $2x + 1$ time when $1.5 \leq x < 2$, or 4 when $1 < x < 1.5$. In contrast SNF requires $x + 3$ time for both cases, and has a longer reduction time for for all x between 1 and 2.

6 Experimental Results

As we describe in Section 3, any valid schedule can be converted into a earliest possible schedule, which in turn can be described by a processor sequence. As a result we can find an optimal reduction schedule among the $(n-1)!$ possible processor sequences, where n is the number of processors in the cluster. However, $(n-1)!$ is quite a large number and we apparently cannot try all of these permutations for practical value of $(n-1)!$, even by a straightforward branch-and-bound procedure. To overcome this problem, we show that by using the exchange lemma (Lemma 1) we can dramatically reduce the search space.

We use three techniques to reduce the number of processor sequences we have to consider. First

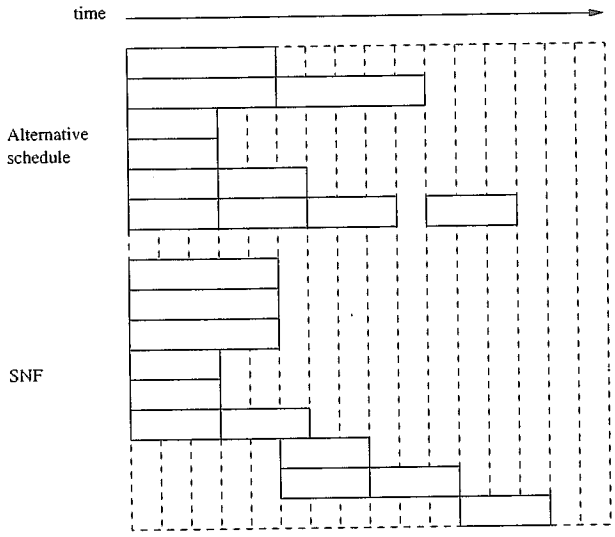

Figure 6: A counterexample that SNF always gives optimal reduction time in a cluster consisting of two types of processors.

of all, we examine the sequences in such an order that those sequences with slower processors appearing first will be examined first. Formally we define the *priority* of a sequence to be the number processors that have longer transmission time than the next processor in the sequence. In other words, the slowest-processor-first schedule has the highest priority, and will be considered first.

The second technique is to apply the exchange lemma so that we do not have to examine all the permutations. Note that by Lemma 1 one we find that a slower processor has to wait for a faster processor, we can stop the search at that subtree immediately. Notice that it is possible for several senders to complete simultaneously so that more than one processor can start at the same time. In that case if any sender is faster than any of those processor that can start when the sender completes, then we can drop this partial solution, and dramatically reduce the search space. Also notice that from the exchange lemma it is trivia to argue that the last sender must be the fastest processor in the system. That makes the total possible permutation to be $(n-2)!$

Finally, we use generic branch-and-bound technique to traverse the search tree. If the time of a partially examined sequence is already larger than the current optimal sequence, then the entire subtree is pruned. This technique is most effective when the difference among processor speed is large.

The input cluster configurations are generated as follow. We assume that the number of classes in a cluster is between 3 and 6. This assumption is practical since processors are usually purchased in batches. For a given class number, we build their relative communication speed into a table. Then we vary the cluster size from 6 to 16. Each processor will randomly pick a communication speed from that speed table. For each cluster size we repeat the experiments for 50 times and compute the average number of tree nodes the algorithm has to search in order to find the optimal solution.

We quantify the search efficiency of our algorithm as follows. For a naive algorithm that scan through all the possible permutations, it has to go over $P = \frac{(n-1)!}{m_1!m_2!...m_k!}$ permutations, where n is the number of processors and m_i is the number of processors in the ith class among the k different processor classes. For an algorithm that scans p solutions before finding the optimal one, we define *search efficiency* to be $1 - \frac{p}{P}$. As indicated by Table 1, our algorithm has extremely high efficiency, especially for large cluster sizes. In addition, for cluster with more than 16 processors, the naive algorithm will simply take too much time, while our algorithm can still find the optimal solution efficiently.

processor #	classes number			
	3	4	5	6
6	53.22	56.48	58.36	63.82
7	64.28	72.74	77.41	71.38
8	71.28	79.24	82,26	84.18
9	86.78	91.88	93.33	94.48
10	89.99	93.80	94.70	96.42
11	94.32	96.84	97.29	98.71
12	94.62	97.58	97.48	98.82
13	96.40	98.63	99.09	99.62
14	96.78	98.64	99.40	99.65
15	97.70	99.45	99.84	99.83
16	98.21	99.51	99.84	99.96

Table 1: Efficiency of the search algorithm.

7 Conclusion

This paper shows that a simple exchange lemma can greatly reduce seraph space so that we can efficiently find the optimal reduction schedule within a heterogeneous environment. Combined with standard branch-and-bound technique the search space is reduced to be less than one percent of original size for most practical cluster size.

It will be interesting to extend this technique to other communication protocols and models. For example, in our model the communication time is determined completely by the sender. In a more

complex model the communication time may be determined by both the send and the receiver. In addition, in a switch-based multi-port system a processor may participate up to a small number of communication simultaneously. These questions are very important in designing collective communication protocols in heterogeneous clusters, and will certainly be the focus of further investigations in this area.

References

[1] *Message Passing Interface Forum.* Mar 1994.

[2] T. Anderson, D. Culler, and D. Patterson. A case for networks of workstations (now). In *IEEE Micro*, Feb 1995.

[3] M. Banikazemi, V. Moorthy, and D.K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of International Parallel Processing Conference*, 1998.

[4] J. Bruck et al. Efficient message passing interface(mpi) for parallel computing on clusters of workstations. *Journal of Parallel and Distributed Computing*, Jan 1997.

[5] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi: a message passing interface standard. Technical report, Argonne National Laboratory and Mississippi State University.