Parallel Programming Techniques for Distributed Memory Computers

E. L. G. Saukas
Summa Technologies, Inc.
São Paulo, SP, Brazil
einar at summa-tech. com
http://www.ime.usp.br/ ~ einar

S. W. Song *
Inst. de Matemática e Estatística
Universidade de São Paulo
R. do Matão, 1010
São Paulo, SP 05508-900, Brazil
song at ime. usp. br
http://www.ime.usp.br/ ~ song

Abstract

The CGM (Coarse Grained Multicomputer) model is a very simple parallel programming model. A CGM algorithm consists of an alternating sequence of computation rounds and communication rounds. The communication cost is modeled by the number of communication rounds. The objective is to design algorithms that require a small number of communication rounds. We discuss two important issues and make considerations of practical importance, based on our previous experience in the design and implementation of several parallel algorithms. The first issue is the amount of data transmitted in each communication round, in addition to minimizing the number of communication rounds. The second issue is about the trade-off between the number of communication rounds which the CGM attempts to minimize and the time taken by each communication round. These two issues have guided us to present an efficient algorithm for the selection problem, used here as illustration.

1 Coarse-Grained Multicomputer

The CGM (Coarse Grained Multicomputer) parallel programming model [2, 3] is similar to Valiant's BSP (Bulk-Synchronous Parallel) model [7]. It is simpler and uses only two parameters, n and p, where n is the size of the input and p the number of processors each with $O(n/p)$ local memory. The term *coarse grained* means the local memory size is large, usually we require $n/p > p$.

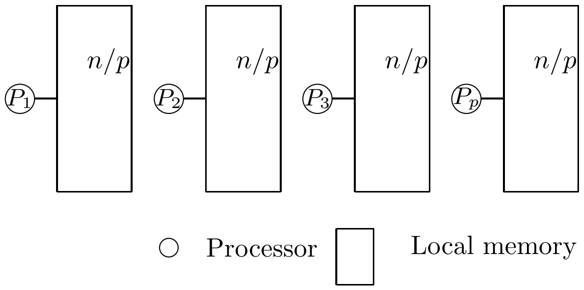

Figure 1: A distribute memory multicomputer

All processors are connected by a router that can deliver messages in a point to point fashion. To make the model as general as possible, we do not make any assumption on the particular interconnection network utilized to connect the processors. A CGM program or algorithm consists of an alternating sequence of *computation rounds* and *communication rounds* separated by barrier synchronizations.

In a computation round, we usually use the best possible sequential algorithm in each processor to process its data locally. In a communication round each processor can exchange a total of $O(n/p)$ data with other processors.

In the CGM model the communication cost of a parallel algorithm is modeled by the number of communication rounds. The objective is to design algorithms that require a small number of communication rounds, usually constant or some function of p, for example $O(\log p)$. As p is usually small compared

*Member of IFIP Working Group 10.3 (Concurrent Systems). Research partially supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) Proc. No. 98/06138-2, CNPq Proc. No. 52 3778/96-1, and CNPq/NSF Collaborative Research Program Proc. No. 680037/99-3.

to n the resulting CGM algorithm is expected to be efficient when implemented in practice.

Notice that the CGM model is particularly suitable in cases where the overall computation speed is considerably larger than the overall communication speed, and the problem size is considerably larger than the number of processors, which is usually the case in practice.

2 Programming Techniques and Issues

In this paper we emphasize some important issues and make considerations of practical importance on the CGM model, based on our previous experience in the design and implementation of several parallel algorithms. We illustrate our discussion with some examples.

The CGM model is a simple parallel computing model intended to work for a multicomputer with processors connected by any interconnection network. The actual communication time, however, depends on the communication characteristics and parameters of the particular interconnection network. To make the model as independent as possible, the communication cost in the CGM model takes into account only the number of communication rounds. The goal is to produce algorithms that require a small number of such rounds.

In one communication round in the CGM model, we allow the exchange of $O(n/p)$ data among the processors. The first issue we want to consider is the amount of data transmitted in a communication round. For a practical implementation to be successful we should attempt to minimize this amount, even when it is already within the maximum allowed by the CGM model (which is $O(n/p)$). We will illustrate this in Section 3 with two programs for the problems of selection and simultaneous selection. These two programs have been implemented by taking this issue into consideration, with very significant result.

The second issue has to do with the trade-off between the number of communication rounds which the CGM attempts to minimize and the time taken by each communication round. Even though in the CGM model we want to minimize the number of communication rounds so that we do not have to care about the particular interconnection network, in a practical implementation, we do know the characteristics of the times taken by communication in a particular interconnection network. Although the CGM model minimizes the number of communication rounds, ultimately our goal is to minimize the overall execution time.

In summary, we consider the following important issues in the design of an efficient parallel algorithm.

1. The amount of data transmitted in each communication round, in addition to minimizing the number of communication rounds.

2. Trade-off between the number of communication rounds which the CGM attempts to minimize and the time taken by each communication round.

3 Amount of data transmitted

In this section we illustrate the first issue on the importance of minimizing the amount of data transmitted in a communication round. We illustrate this by an example of a program we developped and implemented to solve the selection problem [4, 5].

Consider a sequence X of n distinct elements. The rank of x in X, denoted by $rank(x, X)$, is the number of elements of X that are less than or equal to x.

Given an integer k, where $1 \leq k \leq n$, the selection problem is to obtain an element x of X such that $rank(x, X)$ is equal to k. When $k = 1$ and $k = n$, we have the special cases of determining the minimum and maximum of X, respectively. When $k = \lceil n/2 \rceil$, we have the important case of finding the median of X.

It is well known that the selection problem can be solved by a sequential algorithm in linear time [1]. A parallel algorithm based on the CGM model can be devised as follows (we omit the details and refer the reader to [4, 5]). Partition the input data successively to reduce the total amount of input elements from $O(n)$ to $O(n/p)$. These remaining elements can then be processed sequentially in a single processor in linear $O(n/p)$ time.

We use the following partitioning strategy. Choose an element M in sequence A with the following property: $n/4 \leq rank(M, A) \leq (3n)/4$. Partition A into three subsequences A_l, A_e, A_g, respectively with elements less than, equal to, and greater than M. We have $|A_l| \leq (3n)/4$ and $|A_g| \leq (3n)/4$. Based on the value of k relative to the sizes of the three subsequences A_l, A_e, A_g, we either find the kth element, or continue the selection with a smaller subsequence. If k corresponds to the first partition A_l, then we continue the selection using subsequence A_l and discard the rest (see Figure 2); the new problem size will therefore be reduced by at least 1/4. If k corresponds to the middle partition A_e, then we have found the answer: the kth element is M. If k corresponds to the last partition A_g, then we continue the selection using subsequence A_g to select the element of rank $k - (|A_l| + |A_e|)$; again the new problem size will be reduced by at least 1/4. Thus after $O(\log p)$ reductions, we will have only $O(n/p)$ data remaining, that will fit into one processor to be processed sequentially.

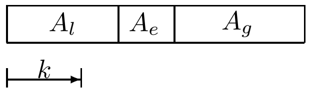

Figure 2: Position of k relative to the sizes of A_l, A_e, and A_g

It can be shown that to find such an element M with the above property we can use the median of the medians of the blocks [1]. At each round the elements are partitioned into p blocks and medians of each block are obtained.

Now we come to the important question of the amount of data exchanged in each communication round. To use the median of medians to partition the data, we would require equal block sizes at each round. Thus redistribution of data among processors would be needed at each round. Redistribution of data can be done in each round by exchanging a total of $O(n/p)$ data. This is actually allowed in the CGM model. However, we now consider the issue of minimizing the amount of data transmitted in each round. We would like to avoid this costly load balancing step in each communication round. Instead we would like to be able to continue the partitioning without having to do the data redistribution. We have thus come up with the use of the weighted median of medians as M, instead of just using the median of medians. The detailed algorithm can be found in [4].

The issue of amount of data transmitted in each round has guided us to attempt to use the weighted median of the blocks' medians to do the partitioning, thus avoiding redistribution of data. With this we get a substantial saving in communication time. This turns out to be crucial in the nice experimental results which we reproduce in Figure 3, where we considered the cases of input elements distributed at random among the processors and also distributed in the worst possible way (regarding parallelization).

The basic selection problem has been extended to a second algorithm to solve q simultaneous selection problems on the same sequence of elements. Again

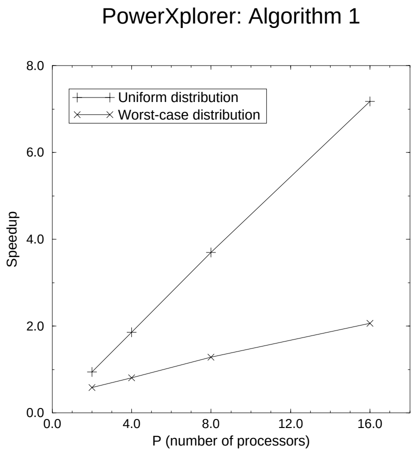

Figure 3: Speedup of the basic selection algorithm for $n = 10^{20}$

here we utilize the weighted median of medians with the same purpose of avoiding load rebalancing at each round. Experimental results are shown in Figure 4.

4 Trade-off number of rounds $\times$ cost of each round

We now consider the trade-off between the number of communication rounds which the CGM attempts to minimize and the cost of each communication round. Even though in the CGM model we want to minimize the number of communication rounds so that we do not have to care about the particular interconnection network, in a practical implementation, we do know the machine characteristics and the communication times in a particular interconnection network.

In the selection algorithm of [4] we introduce a design parameter that can be adjusted for a particular parallel machine, thus taking care of the above mentioned trade-off. The idea is as follows. As presented earlier, we first reduce the amount of input data from n to n/p so it fits into one processor. This is done by the following iteration:

Algorithm 1 *CGM algorithm for obtaining the kth smallest element, assuming p processors each with local memory of size $O(n/p)$.*

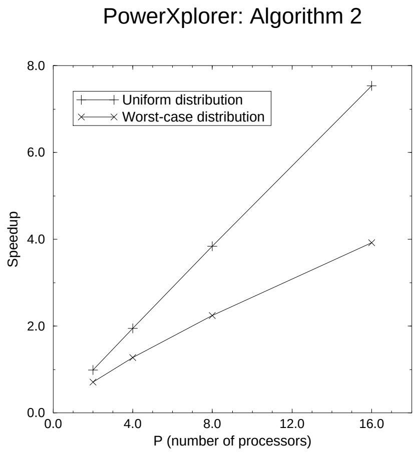

Figure 4: Speedup of the simultaneous selection algorithm for $n = 10^{20}$ and $q = 16$

Input: *Sequence A of n elements distributed among the p processors, each processor i with $n_i = O(n/p)$ elements, and an integer k, $1 \leq k \leq n$.*

Output: *An element a_i of A such that $rank(a_i, A) = k$.*

(1) Set $N := n$

(2) Repeat until $N \leq n/(cp)$ Apply partitioning strategy to discard unnecessary data and other steps ...

In the algorithm, c is a parameter that can be chosen freely. Its effect is as follows. The number of communication rounds is $O(\log cp)$. In the final communication round, the amount of data transmitted is $O(\frac{n}{cp})$. Thus increasing c increases the number of communication rounds, while decreasing the amount of data transmitted in the final round. By choosing an appropriate value for c, the algorithm can be adjusted experimentally for machines with low communication bandwidth. Thus, by adjusting this parameter, we can obtain the best practical result on a specific machine, so that the overall execution time is minimized. Notice that this is an empirical result since the value of parameter c is dependent on the particular machine.

5 Conclusion

We summarize our experience in the design of parallel algorithms using the CGM model by presenting two important design issues: the amount of data exchanged in each communication round and the tradeoff between the number of communication rounds (which the CGM model tries to minimize) and the cost of the communication round. By considering the two issues, illustrated by an example, we achieve the ultimate goal of minimizing the overall execution time.

Acknowledgments

We want to thank the referees for their comments.

References

[1] Blum, M., Floyd, R. W. Pratt, V., Rivest, R. L. and Tarjan, R. E. "Bounds for selection," *Journal of Computer and System Sciences*, Vol. 7, 1973, pp. 448–461.

[2] Dehne, F., Fabri, A. and Rau-Chaplin, A. Scalable parallel geometric algorithms for coarse grained multicomputers. Proc. ACM 9th Annual Computational Geometry (1993) 298–307.

[3] Dehne, F. "Guest Editor's Introduction" *Algorithmica*, (Special Issue on "Coarse grained parallel algorithms), Vol. 24, No. 3/4, July/August 1999, pp. 173-176

[4] Saukas, E. L. G. and Song, S. W. "A Note on Parallel Selection on Coarse Grained Multicomputers," *Algorithmica*, Vol. 24, 1999, pp. 371-380.

[5] Saukas, E. L. G. and Song, S. W. "Efficient Selection Algorithms on Distributed Memory Computers," *ACM/IEEE Supercomputing'98.* Orlando, FL, November 1998.

[6] Saukas, E. L. G. and Song, S.W. "A parallel algorithm for solving tridiagonal linear systems on coarse grained multicomputer". *IX Brazilian Symposium on Computer Architecture and High-Performance Processing.* October 8 to 10, 1997, pp. 463 – 474.

[7] Valiant, L. G. et al. General purpose parallel architectures. Handbook of Theoretical Computer Science, Edited by J. van Leeuwen, MIT Press/Elsevier (1990) 943–972.