Chapter 26

A NOTE ON COARSE GRAINED PARALLEL INTEGER SORTING

A. Chan and F. Dehne
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6[*]
{achan,dehne}@scs.carleton.ca

Abstract We observe that for $n/p \geq p$, which is usually the case in practice, there exists a very simple, deterministic, optimal coarse grained parallel integer sorting algorithm with 24 communication rounds (6 $\frac{n}{p}$-relations and 18 p-relations), $O(n/p)$ memory per processor and $O(n/p)$ local computation. Experimental data indicates that the algorithm has very good performance in practice.

Keywords: BSP, coarse grained parallel algorithms, integer sorting.

26.1 INTRODUCTION

Goodrich [4] presented a deterministic sorting algorithm for the BSP [5] and closely related CGM model [2, 3]. Given $O(n)$ data items stored on a p processor BSP/CGM, $O(n/p)$ data items per processor, these items can be sorted in $O(\frac{\log n}{\log(h+1)})$ communication rounds (h-relations), for $h = \Theta(n/p)$, with $O(\frac{n \log n}{p})$ local computation, using $O(n/p)$ memory per processor. For $n/p \geq p^{\epsilon}$, $\epsilon > 0$, $O(\frac{\log n}{\log(h+1)}) = O(1)$. That is, for this case, the algorithm requires $O(1)$ communication rounds.

We are interested in the problem of sorting $O(n)$ *integers* in the range $1, \ldots, n^c$, for fixed constant c, stored on a p processor BSP/CGM, n/p data items per processor. The sort algorithm in [4] is based on Cole's merge sort [1]. The $O(\frac{n \log n}{p})$ local computation in [4] is due to a constant number of local

[*]Research partially funded by the Natural Sciences and Engineering Research Council of Canada.

sorts. Hence, by applying radix sort for the integer case, it is easy to obtain $O(n/p)$ local computation without increasing the number of communication rounds.

In this paper we observe that for $n/p \geq p$, which is usually the case in practice, there exists a very simple, deterministic, optimal BSP/CGM integer sorting algorithm with 24 communication rounds (6 $\frac{n}{p}$-relations and 18 p-relations), $O(n/p)$ memory per processor and $O(n/p)$ local computation. Experimental data indicates that the algorithm has very good performance in practice.

26.2 THE ALGORITHM

We first present a BSP/CGM integer sorting algorithm for $\frac{n}{p} \geq p^2$, which serves as our base case, and then extend it to the case $\frac{n}{p} \geq p$. The integer sorting algorithm for $\frac{n}{p} \geq p^2$, described next, follows the well known *deterministic sample sort* method combined with *radix sort* for the sequential sorting steps. We are making the algorithm descriptions fairly detailed in order to allow an analysis that includes estimates of constant factors.

Algorithms 1: Sorting n integers on a p processors BSP/CGM with $\frac{n}{p} \geq p^2$. **Input:** n integers in the range $1, \ldots, O(n^c)$, for fixed constant c, stored on a p processor BSP/CGM, n/p integers per processor. $\frac{n}{p} \geq p^2$. **Output:** The integers are permuted into sorted order.

1. Each processor sorts locally its n/p integers, using radix sort.
2. Each processor selects from its locally sorted integers a sample of p integers with ranks $i\frac{n}{p^2}$, $0 \leq i \leq p-1$. We refer to these selected integers as *local samples*. All local samples are sent to processor P_1. (Note that P_1 is to receive in total p^2 integers. This is possible since $\frac{n}{p} \geq p^2$.)
3. P_1 sorts the p^2 local samples received in Step 2 (using radix sort) and selects a sample of p integers with ranks ip, $0 \leq i \leq p-1$. We refer to these selected integers as the *global samples*. The p global samples are then broadcast to all processors.
4. Based on the received global samples, each processor P_i partitions its n/p integers into p buckets $B_{i,1}, \ldots, B_{i,p}$ where $B_{i,j}$ are the local integers with value between the $(j-1)$-th and j-th global sample.
5. In one (combined) h-relation, every processor P_i, $1 \leq i \leq p$, sends $B_{i,j}$ to processor P_j, $1 \leq j \leq p$. Let R_j be the set of integers received by processor P_j, $1 \leq j \leq p$, and let $r_i = |R_i|$.
6. Every processor P_i, $1 \leq i \leq p$, locally sorts R_i using radix sort.

7. A global "balancing shift" operation which distributes all integers evenly among the processors without changing their order is performed as follows: Every processor P_i, $1 \leq i \leq p$, sends r_i to P_1. Processor P_1 calculates for each P_j an array A_j of p numbers indicating how many of its integers have to be moved to the respective processors. In one h-relation, every A_j is sent to P_j, $1 \leq i \leq p$. The "balancing shift" is then performed in a subsequent single h-relation according to the A_j values.

— end of Algorithm —

Theorem 1 *Algorithm 1 sorts n integers in the range $1, \ldots, O(n^c)$, for fixed constant c, stored on a p processor BSP/CGM, n/p integers per processor, $\frac{n}{p} \geq p^2$, using 6 communication rounds (2 $\frac{n}{p}$-relations and 4 p^2-relations), $O(\frac{n}{p})$ local memory per processor, and $O(\frac{n}{p})$ local computation.*

Proof. Correctness: The algorithm follows the well known deterministic sample sort method combined with radix sort for the sequential sorting steps. Let $S_{i,j}$ be the set of integers on P_i, at the end of Step 1, with rank between $j\frac{n}{p^2}$ and $(j+1)\frac{n}{p^2}$, $0 \leq j \leq p-1$. The main observation is that each R_k contains at most $3p$ sets $S_{i,j}$.
Complexity: The local memory and local computation are bounded by the sequential local radix sorts. For the communication, we observe that Steps 2 and 3 require one p^2-relation, each, Step 5 requires one $\frac{n}{p}$-relation, and Step 7 requires two p^2-relations and one $\frac{n}{p}$-relation. □

We now describe the algorithm for the case $\frac{n}{p} \geq p$. The basic idea is to partition the p processors into $\sqrt{p}$ groups $G_1, \ldots, G_{\sqrt{p}}$ of $\sqrt{p}$ processors, each. The main operation consists of permuting the integers such that all integers stored in G_i are smaller than all integers stored in G_j for all $i < j$. We can then apply Algorithm 1 to each group. Again, we are making the description of Algorithm 2 fairly detailed in order to allow an analysis that includes estimates of constant factors.

Algorithms 2: Sorting n integers on a p processors machine when $\frac{n}{p} \geq p$. **Input:** n integers in the range $1, \ldots, n^c$, for fixed constant c, stored on a p processor BSP/CGM, n/p integers per processor. $\frac{n}{p} \geq p$. **Output:** The integers are permuted into sorted order.

1. Group the p processors into $\sqrt{p}$ groups $G_1, \ldots, G_{\sqrt{p}}$ of $\sqrt{p}$ processors, each. For each group, apply Algorithm 1 to sort the integers within the group.

2. Each processor sends its smallest integer to processor P_1. We will refer to these items as the *local minima*. (Note that P_1 is to receive p local minima in total, and that this is possible since $\frac{n}{p} \geq p$.)

3. P_1 sequentially sorts all local minima and selects the $\sqrt{p}$ integers with rank $i\sqrt{p}$, $0 \leq i \leq \sqrt{p}$, referred to as *global splitters*. These $\sqrt{p}$ global splitters are broadcast to all processors (using 2 p-relations).

4. Based on the received global splitters, each processor P_i partitions its n/p integers into $\sqrt{p}$ buckets $B'_{i,1}, \ldots, B'_{i,\sqrt{p}}$ where $B'_{i,j}$ contains the local integers with value between the $(j-1)$-th and j-th global splitter.

5. Every processor P_i, $1 \leq i \leq p$, sends $B'_{i,j}$ to a processor in group G_j, $1 \leq j \leq \sqrt{p}$. Let R'_j be the set of integers received by processors in group G_j, $1 \leq j \leq \sqrt{p}$, and let $r'_j = |R'_j|$. The routing schedule for this operation is determined as follows: First, within each group G_i the size, $t_{i,j}$, of each integer set sent to group G_j, $1 \leq j \leq \sqrt{p}$, is computed. All $t_{i,j}$, $1 \leq i, j \leq \sqrt{p}$, are sent to one *leading* processor per group. Furthermore, within each group, the sizes of all $B'_{i,j}$ are also sent to the leading processor. Each leading processor can then compute a routing schedule for its group and broadcast it to the processors in its group.

6. Each group G_j, $1 \leq j \leq \sqrt{p}$, sorts R'_j using Algorithm 1.

7. A global "balancing shift" operation which distributes all integers evenly among the processors without changing their order is performed analogous to Step 6 of Algorithm 1 but with a "two phase" scheme analogous to the routing schedule computation in Step 5 of Algorithm 2.

— end of Algorithm —

Theorem 2 *Algorithm 2 sorts n integers in the range $1, \ldots, n^c$, for fixed constant c, stored on a p processor BSP/CGM, n/p integers per processor, $\frac{n}{p} \geq p$, using 24 communication rounds (6 $\frac{n}{p}$-relations and 18 p-relations), $O(\frac{n}{p})$ local memory per processor, and $O(\frac{n}{p})$ local computation.*

Proof. Correctness: For each group G_i of processors, we have $n' = \frac{n}{\sqrt{p}}$ integers and $p' = \sqrt{p}$ processors. Therefore, $\frac{n'}{p'} = \frac{n/\sqrt{p}}{\sqrt{p}} = \frac{n}{p} \geq p = {p'}^2$, and Algorithm 1 is applicable for each group. Let S'_i be the set of integers stored at P_i after Step 1. The second main observation is that each R'_j contains at most $3\sqrt{p}$ sets S'_i.
Complexity: The local memory and local computation are bounded by the sequential local radix sorts. For the communication, we observe that Step 1

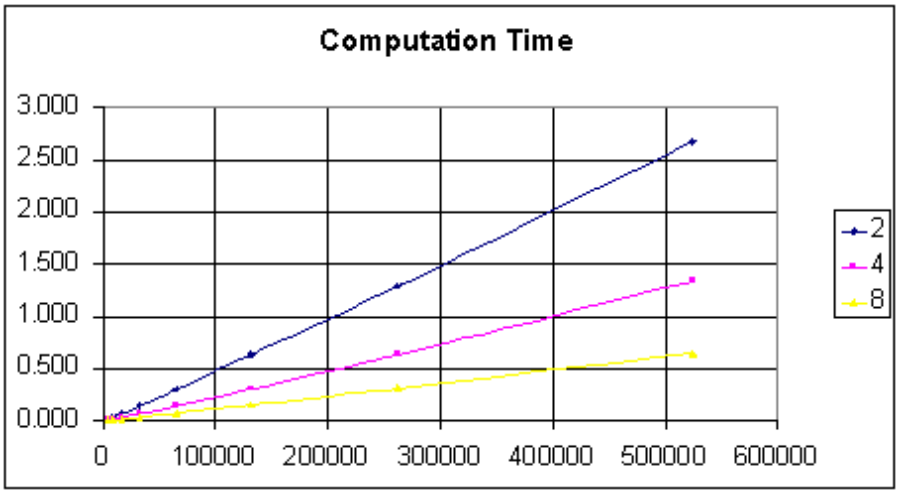

Figure 26.1 Computation Times (In Seconds) For Different Numbers Of Data Items. The Three Curves Represent Configurations Of 2, 4, And 8 Processors, Respectively.

requires 2 $\frac{n}{p}$-relations and 4 p-relations, Step 2 requires 1 p-relation, Step 3 requires 2 p-relations, Step 5 requires 1 $\frac{n}{p}$-relation and 3 p-relations, Step 6 requires 2 $\frac{n}{p}$-relations and 4 p-relations, and Step 7 requires 1 $\frac{n}{p}$-relations and 3 p-relations. □

26.3 IMPLEMENTATION AND EXPERIMENTS

We have implemented Algorithm 1 in MPI and tested it on a multiprocessor Pentium platform running LINUX. The communication between the processors is performed through an Ethernet switch.

It is interesting to observe that, even on this low cost architecture, our algorithm shows good performance. Figures 26.1, 26.2, and 26.3 show the computation, communication, and total times (in seconds), respectively, for different numbers of data items. The three curves in each figure represent configurations of 2, 4, and 8 processors, respectively. As expected, the communication times are fairly similar. The computation times and total running times are almost linear, and we observe close to linear speedup. Note that, linear speedup is archived even with a reasonably small workload. (Many algorithms achive linear speedups only for very high workloads.)

In summary, we observe that the algorithm shows very good performance in practice.

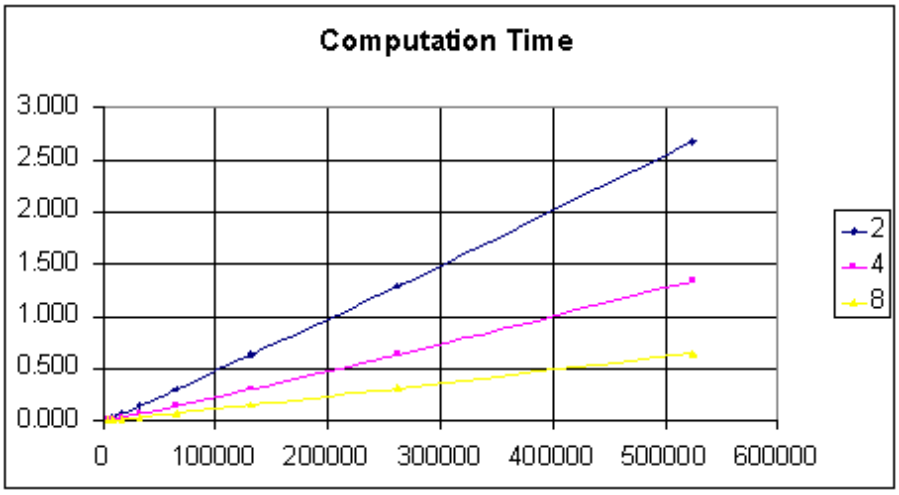

Figure 26.2 Communication Times (In Seconds) For Different Numbers Of Data Items. The Three Curves Represent Configurations Of 2, 4, And 8 Processors, Respectively.

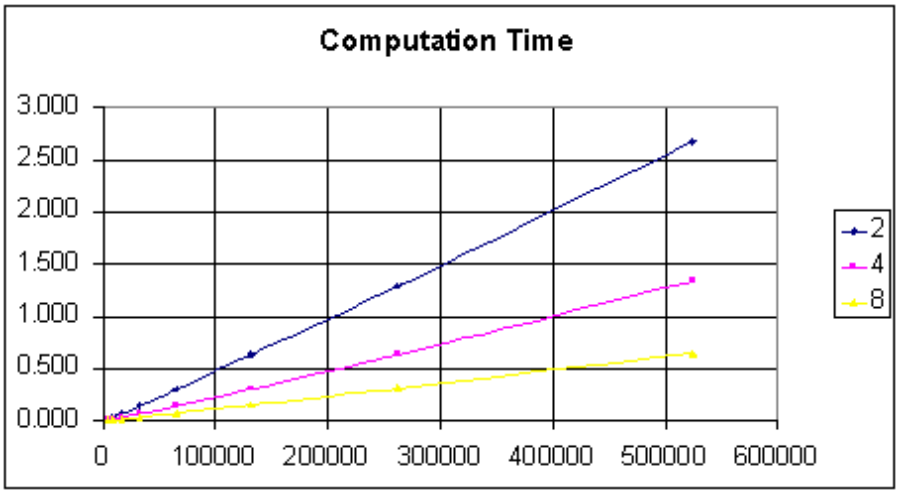

Figure 26.3 Total Running Times (In Seconds) For Different Numbers Of Data Items. The Three Curves Represent Configurations Of 2, 4, And 8 Processors, Respectively.

26.4 CONCLUSION

We have described a very simple, deterministic, optimal BSP/CGM integer sorting algorithm that assumes $n/p \geq p$, which is usually the case in practice. Our algorithm description is fairly detailed in order to allow an analysis that includes estimates of constant factors. The algorithm requires 24 communication rounds (6 $\frac{n}{p}$-relations and 18 p-relations), $O(n/p)$ memory per processor and $O(n/p)$ local computation. For theoretical interest, it is easy to see that the algorithm can be generalized to run with $O(1/\epsilon)$ rounds for $n/p \geq p^{\epsilon}$, $\epsilon > 0$. Experimental data indicates that the algorithm has very good performance in practice.

References

[1] R. Cole, "Parallel merge sort," SIAM J. Comput., 17(4), pp. 770–785, 1988.

[2] F. Dehne, A. Fabri, and A. Rau-Chaplin, "Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers," in *Proc. ACM 9th Annual Computational Geometry*, pp. 298–307, 1993

[3] F. Dehne, X. Deng, P. Dymond, A. Fabri and A.A. Kokhar, "A randomized parallel 3D convex hull algorithm for coarse grained parallel multicomputers", in *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 1995.

[4] M.T. Goodrich, "Communication Efficient Parallel Sorting", in Proc. 28th Annual ACM Symp. on Theory of Computing (STOC'96), 1996.

[5] L.G. Valiant, "A Bridging Model for Parallel Computation." *Communications of the ACM*, Vol. 33, pp. 103–111, 1990.