Regular Paper

Parallel Selection Algorithms for CGM and BSP Models with Application to Sorting*

Akihiro Fujiwara,† Michiko Inoue†† and Toshimitsu Masuzawa††

In this paper, we present two deterministic selection algorithms with application to sorting for the CGM and BSP models. The first is a parallel algorithm whose computation time is cost optimal, which runs with $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log\log n))$ communication rounds for $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon > 0$. The second is a parallel algorithm whose number of communication rounds is optimal, which runs with $O(\frac{n}{p}\log p)$ computation time and a constant number of communication rounds for $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon > 0$. In addition, we apply the second selection algorithm to sorting. The sorting algorithm runs with $O(\frac{n}{p}\log n)$ computation time and a constant number of communication rounds, where $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon \geq 2$.

1. Introduction

The selection problem is to find the kth-smallest element in a given totally ordered set of n elements for a given parameter k ($1 \leq k \leq n$). (In the case of $k = \lceil \frac{n}{2} \rceil$, the element is called the median.) Since the selection problem is a basic one that plays an important role in computer science, many selection algorithms have been proposed. For sequential computing, an optimal selection algorithm, whose time complexity is $O(n)$, was proposed by Blum, et al.[4]. Many parallel algorithms have also been proposed for the problem, mainly in PRAM models or network-dependent models (e.g., the mesh model and the hypercube model).

However, the architectures of recent parallel computers are quite different from the above models. A typical parallel computer now consists of a set of processor modules that run asynchronously. Each module has the latest processor and a local memory which is considerably larger than $O(1)$. The modules are connected by some fast interconnection network, but the cost of one communication (send or receive) is considerably larger than the cost of one internal computation. On the other hand, the PRAM and network-dependent models assume synchronous processing and a small amount memory of per processor. In addition, communication costs are not evaluated precisely in the models. Thus the above models are not suited for recent parallel computers, and in many cases, their algorithms are not efficient on parallel computers.

For practical purposes, some parallel computation models have been proposed for the recent parallel computers. Among them, the Bulk-Synchronous Parallel (BSP) model, which was proposed by Valiant[12], has received considerable attention. The Coarse Grained Multicomputer (CGM) model, which was proposed by Dehne, et al.[5], is essentially the same as the BSP model except for the following points. In the BSP model, communication issues are abstracted by using two parameters, L and g, which denote the latency of the network and the communication throughput ratio, respectively. In the CGM model, on the other hand, communication costs are determined by the number of communication rounds.

In this paper, we consider selection algorithms for these models. For the BSP model, some randomized selection algorithms have been proposed. Gerbessiotis, et al.[6] proposed a randomized algorithm that runs in $O(\frac{n}{p} + T_{ppf}(p))$ time with high probability, where $T_{ppf}(p)$ is the time required for a parallel prefix operation of p processors. Bäumker, et al.[3] proposed another randomized algorithm. If $n = \Omega(p\log^4 n)$ holds, the algorithm runs in $O(\frac{n}{p} + L\log p)$ computation time and $O(\frac{g}{B}\sqrt{\frac{n}{p}} + (L+g)\log p)$ communication time

† Department of Computer Science and Electronics, Kyushu Institute of Technology
†† Graduate School of Information Science, Nara Institute of Science and Technology

* Research supported in part by the Scientific Research Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan (Scientific Research of Priority Area(B)10205218).

for $B \leq \sqrt{\frac{n}{p}}$ with high probability☆. In addition, some deterministic and randomized selection algorithms[1),2)] have also been proposed for BSP-like models. These two algorithms have been used experimentally on real parallel machines.

Recently, two deterministic selection algorithms were proposed for the above two models. Saukas and Song[11)] proposed a deterministic CGM algorithm for the selection problem. Their algorithm runs with $O(\frac{n}{p}\log p)$ computation time and $O(\log p)$ communication rounds for $\frac{n}{p} \geq p$ in the CGM model. Ishimizu, et al.[9)] proposed parallel selection algorithms on the BSP and BSP* models. The complexities of their BSP algorithm are $O(\frac{n}{p}+d\log p\log\log n+L\frac{\log p\log\log n}{\log d})$ for the computation time and $O(g\frac{n}{p}+(gd+L)\frac{\log p\log\log n}{\log d})$ for the communication time. Although their BSP algorithm is cost optimal in some cases, it needs $O(\log p\log\log n)$ communication rounds, whose number depends on the input size n; that is, the number of communication rounds grows with respect to n.

Finding an optimal algorithm on BSP and CGM models is equivalent to minimizing the number of communication rounds so as to be independent of the input size, as well as minimizing its computation time. In this paper, we first propose two selection algorithms in which the number of communication rounds is independent of n, and next apply one of these algorithms to sorting. According to a definition of the CGM model, the complexities of the algorithms are evaluated according to two parameters, the *computation time* and the *number of communication rounds*☆☆. We assume that *h-relation*, with $h = O(\frac{n}{p})$, is permitted in each communication round; that is, each processor can send $O(\frac{n}{p})$ data and receive $O(\frac{n}{p})$ data in a round.

Our first selection algorithm runs with $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log\log n))$ communication rounds in the CGM model. The algorithm achieves cost optimality with respect to the computation time. In the algorithm, we assume that $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon > 0$. Notice that this assumption holds for almost all real parallel computers. According to a definition of the BSP model, the algorithm runs in $O(\frac{n}{p} + (L+g\frac{n}{p})\times\min(\log p, \log\log n))$ time in the BSP model. Previous deterministic BSP/CGM algorithms for the selection problem need communication rounds whose number depends on n, or $O(\log p)$ communication rounds and $O(\frac{n}{p})$ computation time per round, to our knowledge. Thus, our algorithm is the first deterministic selection algorithm that achieves a total computation time of $O(\frac{n}{p})$ with a number of communication rounds that is independent of n.

In the second selection algorithm, we aim to optimize the number of communication rounds. Using Goodrich's sorting algorithm[8)], we can solve the selection with $O(\frac{n}{p}\log p)$ computation time and a constant number of communication rounds in the case of $n = p^{\epsilon}$ and $\epsilon > 0$. Our second selection algorithm runs with the same complexity for $\frac{n}{p} \geq p^2$. By combining Goodrich's sorting algorithm and our algorithm, we can solve the selection problem with the above complexity for $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon > 0$ in the CGM model. Since the number of processors on parallel machines is usually fixed, the second algorithm may be faster than the first in some situations. In the BSP model, the algorithm runs in $O(\frac{n}{p}\log p + L + g\frac{n}{p})$ time.

Furthermore, we created a sorting algorithm by modifying the second selection algorithm. The algorithm sorts n elements with $O(\frac{n}{p}\log n)$ computation time and a constant number of communication rounds in the CGM model and with $O(\frac{n}{p}\log n+L+g\frac{n}{p})$ time in the BSP model, for $\frac{n}{p} \geq p^2$. Compared with Goodrich's sorting algorithm[8)], the number of processors used by our algorithm is restricted to no more than $n^{\frac{1}{3}}$ and the other complexities are the same. However, our algorithm is simple and seems to be faster in some situations.

This paper is organized as follows. In Section 2, we give brief descriptions of the models and primitive operations. In Section 3, we present our cost optimal selection algorithm. In Section 4, we present the second selection algorithm, whose number of communication rounds is optimal, and describe its application to sorting. Section 5 concludes the paper.

2. Preliminaries

2.1 Models

The *rank* of an element in a totally ordered set is the number of elements smaller than or

☆ Their algorithm assumes the use of the BSP* model, which is an extended version of the BSP model.

☆☆ We characterize the algorithm in the BSP model by these two complexities and the BSP parameters. The method is described in Section 2.

equal to the element. The selection problem is the problem of finding an element whose rank is k in a given set of n elements for a given parameter k ($1 \leq k \leq n$). For simplicity, we assume that all elements are distinct. For CGM and BSP models, we assume that the input elements are evenly distributed on p processors $P_0, P_1, \ldots P_{p-1}$, that is, each processor stores $\lceil \frac{n}{p} \rceil$ or $\lceil \frac{n}{p} \rceil - 1$ input elements at the beginning of an algorithm.

Both CGM and BSP models consist of three parts: a set of processor modules, a communication network for module-to-module communication and a synchronizer that synchronizes all or a subset of processors in barrier style. In this paper, we assume that a computation on these models consists of a sequence of *supersteps*. In each superstep, each processor executes *a local computation round* followed by *a communication round*. In a local computation round, each processor computes without any communication with other processors. On the other hand, each processor sends and receives data only in a communication round. Therefore data received in a communication round cannot be used in a local computation round in the same superstep. After all processors have completed their supersteps, the synchronizer makes all processors start the next superstep.

The complexities of algorithms for the CGM model are measured according to two parameters, the *computation time* and the *number of communication rounds*. Let S be the number of supersteps of a computation. The number of communication rounds T_{comm} of the algorithm is equal to the number of supersteps of the computation; that is, $T_{comm} = S$. Let $comp_i$ be the maximum computation time among all local computation rounds in a superstep i, and let $comm_i$ be the maximum number of data sent or received by one processor among all communication rounds in a superstep i. We define the computation time T_{comp} as the sum of $comp_i$ and $comm_i$ for all supersteps; that is,

$$T_{comp} = \sum_{i=1}^{S} (comp_i + comm_i).^{\star}$$

We also assume that each processor can send $O(\frac{n}{p})$ elements and receive $O(\frac{n}{p})$ elements in each superstep. Although a "packing requirement"** is assumed in some papers for CGM algorithms, we do not use such an assumption in this paper, because doing so increases the power of the model.

In the BSP model, two parameters, g and L, are used to denote communication costs. The parameter g denotes the ratio of the computation to the communication throughput, and L denotes the minimal time taken to perform a synchronization. The running time of an algorithm in the BSP model is easily obtained from these two parameters and complexities in the CGM model: The running time of the algorithms is $O(T_{comp} + (L + g \times \frac{n}{p}) \times T_{comm})$, where T_{comp} and T_{comm} denote the computation time and the number of communication rounds in the CGM model, respectively.

2.2 Primitive Operations

In the following, we describe three primitive operations used in this paper.

1. **Broadcast:** Broadcast is an operation to send one element stored in a processor to all of the other processors.
2. **Prefix operation:** Let $\oplus$ be a binary associative operator. Given a sequence of elements $(a_0, a_1, \ldots, a_{p-1})$ such that each element a_i is stored in a processor P_i, the prefix operation computes the value $s_i = a_0 \oplus a_1 \oplus \cdots \oplus a_i$ for each processor P_i. (In this paper, we only use prefix sums.)
3. **Load balancing:** Let A be a set of u elements that is partitioned into p subsets $A_0, A_1, \ldots, A_{p-1}$. We assume that each processor P_i stores a subset A_i and $u_i = |A_i|$. Load balancing is an operation to distribute all elements in A evenly among all processors; that is, after the load balancing operation, each processor stores $\lceil \frac{u}{p} \rceil$ or $\lceil \frac{u}{p} \rceil - 1$ elements of A and every element is stored in exactly one processor.

We can execute the load balancing operation by using the prefix sums as follows.

(1) Each processor P_i computes u_i.
(2) Compute the prefix sums of u_i ($0 \leq i \leq p-1$) by using all processors. Let PS_i be the result for a processor P_i.
(3) Let $A_i = \{a_0^i, a_1^i, \ldots, a_{u_i-1}^i\}$. Each processor P_i sends each element a_j^i to a pro-

* Although in many papers the computation time is defined as $T_{comp} = \sum_{i=1}^{S} comp_i$, we assume that the cost of one sending or receiving operation is not less than the cost of one internal operation.

** The "packing requirement" means that all data sent from a given processor to the same processor in a communication round are packed into one long message.

cessor $P_{\lceil (PS_i - u_i + j + 1)/\frac{u}{p} \rceil - 1}$.

Since each processor sends at most $O(u_i)$ elements and receives at most $O(\frac{u}{p})$ elements except for prefix sums computation in the above three steps, we obtain the following lemma:

Lemma 1 The load balancing of u elements can be performed in $O(\max(u_0, u_1, \ldots, u_{p-1}) + \frac{u}{p} + T_{ppf}^{comp}(p))$ computation time and $T_{ppf}^{comm}(p)$ communication rounds, using p processors in the CGM and BSP models, where $T_{ppf}^{comp}(p)$ and $T_{ppf}^{comm}(p)$ are the computation time and the number of communication rounds of the prefix operation, respectively.

3. Cost Optimal Selection Algorithm

3.1 Basic Idea

Our first selection algorithm is based on a well-known strategy, "median of medians." In the strategy, the median, whose rank is $\lceil \frac{m}{2} \rceil$ among m elements, is used as a pivot to split elements. The strategy was proposed as a sequential algorithm by Blum, et al.[4], and have been utilized in some parallel algorithms[1),2)]. The following algorithm has a similar structure to these algorithms; however, the above algorithms are considered only for $p << n$, and are not cost optimal even for $\frac{n}{p} = p$.

The following is an overview of our algorithm. In the description, we find the kth-smallest element.

Selection algorithm based on strategy "median of medians"

Step 1: Set $s = 1$, $m_s = n$, $k_s = k$, and repeat the following phase until $m_s \leq \max(\frac{n}{p}, \frac{n}{\log n})$.
(m_i denotes the number of remaining elements in a phase i.)

(1) On each processor, find the median of all elements on the processor.

(2) Select the median of the medians. (Details of this substep are described in the following subsection.) Let MM be the median of the medians.

(3) Broadcast MM to all processors.

(4) Split the elements on each processor P_i into two subsets, L_i and U_i. The subset L_i contains elements that are smaller than MM, and the subset U_i contains elements that are larger than MM.

(5) Compute $SUM_L = \sum_{i=0}^{p-1} |L_i|$ by using all processors. According to the following three conditions, discard elements on each processor P_i and set k_{s+1} or end the algorithm.

(5-a) Discard elements contained in $U_i \cup \{MM\}$ and set $k_{s+1} = k_s$. (If $k_s \leq SUM_L$)

(5-b) Output MM and end the algorithm. (If $k_s = SUM_L + 1$)

(5-c) Discard elements contained in $L_i \cup \{MM\}$ and set $k_{s+1} = k_s - (SUM_L + 1)$. (If $k_s > SUM_L + 1$)

(6) Execute the load balancing operation for remaining elements on all processors. Compute the number m_{s+1} of the remaining elements. Finally, set $s = s + 1$.

Step 2: If $\frac{n}{p} \leq \frac{n}{\log n}$, sort all elements by using all processors and find the k_sth-smallest element. Otherwise, gather all elements on one processor and execute a sequential selection algorithm on the processor.

The key point of this strategy is to ensure that $m_{s+1} \leq \frac{1}{\delta} m_s$ for a constant $\delta > 1$. If this condition is satisfied, $m_{s+1} \leq (\frac{1}{\delta})^s n$ holds for each phase s in Step 1. Consequently the number of remaining elements becomes less than $\max(\frac{n}{p}, \frac{n}{\log n})$ after $O(\min(\log p, \log\log n))$ phases. We prove this in the following subsection.

We can perform (1) and (4) of Step 1 in $O(\frac{m_s}{p})$ time on each processor. Therefore we can perform Step 1 in $O(\sum_{s=1}^{\min(\log p, \log\log n)} \lceil \frac{m_s}{p} \rceil)$ $= O(\frac{n}{p} + \min(\log p, \log\log n))$ computation time and a constant number of communication rounds except for the broadcast, prefix sums, and pivot computation in (2). The computation time $O(\frac{n}{p} + \min(\log p, \log\log n))$ is equal to $O(\frac{n}{p})$ for $\frac{n}{p} \geq p^\epsilon$ and $\epsilon > 0$.

We can perform Step 2 in $O(\frac{n}{p})$ computation time and a constant number of communication rounds, for the following reasons. If $\frac{n}{\log n}$ elements remain, we use Goodrich's sorting algorithm[8)]. The algorithm sorts m elements in $O(\frac{m \log m}{p})$ computation time and a constant number of communication rounds for $\frac{m}{p} \geq p^\epsilon$ and $\epsilon > 0$. Therefore, we can sort the remaining elements in $O(\frac{\frac{n}{\log n} \log \frac{n}{\log n}}{p}) = O(\frac{n}{p})$ computation time and a constant number of communication rounds. If $\frac{n}{p}$ elements remain, we can find a result with the same complexity by using the optimal sequential algorithm[4)].

Consequently, we can perform the algorithm in $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log\log n))$ communication rounds

if three operations—the broadcast, prefix sums, and pivot computation in (2)—can be performed in $O(\lceil \frac{m_s}{p} \rceil)$ computation time and $O(1)$ communication rounds in each phase s. In the following, we describe first the details of the broadcast and prefix sums, and finally the details of the pivot computation.

3.2 Broadcast and Prefix Sums

In this subsection, we show that the broadcast and prefix sums can be executed in $O(\frac{n}{p \log n})$ computation time. Since $\frac{n}{p \log n} \leq \frac{m_s}{p}$ holds for every phase s, we can perform these operations in $O(\frac{m_s}{p})$ computation time for each phase.

We use a *d-ary tree* proposed by Gerbessiotis and Valiant[7]. The d-ary tree is an undirected tree that satisfies the following conditions:

- Each non-leaf node has exactly d children.
- All leaves are at level $\lceil \frac{\log p}{\log d} \rceil$.

Using the d-ary tree, we can perform the broadcast and prefix sums in $O(d \times \frac{\log p}{\log d})$ computation time and $O(\frac{\log p}{\log d})$ communication rounds[10]. To perform these operations with the complexity described above, we set $d = \lceil \frac{n}{p \log n} \rceil$, and prove that $\frac{\log p}{\log d} = O(1)$.

$$
\begin{aligned}
&\frac{\log p}{\log \lceil \frac{n}{p \log n} \rceil} \\
&\leq \frac{\log p}{\log \frac{n}{p \log n}} \\
&\leq \frac{\log n^{\frac{1}{1+\epsilon}}}{\log \frac{n}{n^{\frac{1}{1+\epsilon}} \log n}} \quad \left(\text{from } \frac{n}{p} \geq p^{\epsilon} \right) \\
&= \left(\frac{1}{1+\epsilon} \right) \frac{\log n}{\log \frac{n^{\frac{\epsilon}{1+\epsilon}}}{\log n}} \\
&< \frac{\log n}{c \log n - \log \log n} \quad \left(c = \frac{\epsilon}{1+\epsilon} \right) \\
&= O(1)
\end{aligned}
$$

Therefore we can perform the broadcast and prefix sums of each phase in $O(\frac{n}{p \log n})$ computation time and $O(1)$ communication rounds.

3.3 The Median of Medians Computation

We compute the median of medians (MM) by using the d-ary tree ($d = \lceil \frac{n}{p \log n} \rceil$) from leaves to the root. In the following, we describe the computation of MM. Before this computation, each processor stores one median, and we set $d = \lceil \frac{n}{p \log n} \rceil$.

Algorithm for computing MM

Set $g = 1, l_g = p$ and repeat the following phase until $l_g = 1$. (l_g denotes the number of remaining medians in a phase g.)

After all phases are complete, set MM to the remained median.

Step 1: Gather the medians on $\left\lfloor \frac{l_g}{d} \right\rfloor$ processors so that the number of elements on the processors differs by at most 1. To complete this step, we execute the following two operations:

(1) Each processor P_i sends its median to processor P_j such that $j = \lfloor \frac{i}{d} \rfloor$.

(2) A processor $P_{\lfloor \frac{l_g}{d} \rfloor + 1}$, which stores $d-1$ or fewer medians, sends the gathered medians to $P_0, P_1, \ldots, P_{\lfloor \frac{l_g}{d} \rfloor}$ evenly.

Step 2: On each processor where the gathered medians are stored, find a median of the medians on the processor by using a sequential selection algorithm, and discard all medians on the processor except for the obtained median of the medians. Compute the number l_g of the remaining medians by using all processors and set $g = g + 1$ on each processor.

We can prove that the above computation requires $O(\frac{n}{p \log n})$ computation time and $O(1)$ communication rounds in a similar manner to our proof of the broadcast and prefix sums.

The remaining task is to prove that the obtained MM ensures that $m_{s+1} \leq \frac{1}{\delta} m_s$ for a constant $\delta > 1$ in every phase s of the algorithm. (Remember that m_s denotes the number of remaining elements at the beginning of phase s in the selection algorithm.) In our selection algorithm, we split m_s elements into two subsets, $\bigcup_{i=0}^{p-1} L_i$ and $\bigcup_{i=0}^{p-1} U_i$, and set m_{s+1} to one of $\sum_{i=0}^{p-1} |L_i|$ and $\sum_{i=0}^{p-1} |U_i|$, in each phase s. In the following, we prove that $\sum_{i=0}^{p-1} |L_i| \leq (1 - \frac{1}{2^{r+1}}) m_s$ holds for $m_{s+1} = \sum_{i=0}^{p-1} |L_i|$, where r is a constant which denotes the number of communication rounds of the MM computation.

(Proof) Let l_r denote the number of remaining medians at the beginning of the final phase of the computation. Since $l_j \leq l_{j+1} \times d$ and $l_1 = p$, $l_r \times d^{r-1} \geq p$ holds. (Note that we use a d-ary tree such that $d = \lceil \frac{n}{p \log n} \rceil$.)

Let EU_g be the number of remaining medians that are not less than MM, at the beginning of phase g of the pivot computation. From the definition of the median, $EU_r = \lceil \frac{l_r}{2} \rceil$ holds. Since

$$EU_g = EU_{g+1} \times \lceil \tfrac{d}{2} \rceil,$$

$$\begin{aligned} EU_1 &= \left\lceil \frac{l_r}{2} \right\rceil \times \left\lceil \frac{d}{2} \right\rceil^{r-1} \\ &\geq \frac{l_r}{2} \times \left(\frac{d}{2}\right)^{r-1} \\ &= \frac{1}{2^r} \times \left(l_r \times d^{r-1}\right) \\ &\geq \frac{1}{2^r} \times p \end{aligned}$$

Therefore, there exist at least $\frac{1}{2^r}p$ processors on which medians contained in $U_i \cup \{MM\}$ are stored at the beginning of the computation of MM. On each processor which has a median contained in $U_i \cup \{MM\}$, the number of elements that are not less than MM is $\left\lceil \frac{\frac{m_s}{p}}{2} \right\rceil$. Consequently,

$$\begin{aligned} \left(\sum_{i=0}^{p-1} |U_i|\right) + 1 &\geq EU_1 \times \left\lceil \frac{\frac{m_s}{p}}{2} \right\rceil \\ &= \frac{1}{2^{r+1}} m_s. \end{aligned}$$

Since $m_{s+1} = \sum_{i=0}^{p-1} |L_i| = (\sum_{i=0}^{p-1} |U_i|) + 1$ from the partition in the algorithm, $m_{s+1} \leq \left(1 - \frac{1}{2^{r+1}}\right) m_s$ holds for each s. □

We can also prove that $m_{s+1} \leq \frac{1}{\delta} m_s$ in the case of $m_{s+1} = \sum_{i=0}^{p-1} |U_i|$ in the same manner. In consequence, we obtain the following theorem.

Theorem 1 We can solve the selection with $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log\log n))$ communication rounds by using p processors in CGM and BSP models for $\frac{n}{p} > p^{\epsilon}$ and $\epsilon > 0$.

4. Algorithm with Constant Communication Rounds

4.1 Selection Algorithm

We now give the second selection algorithm that runs with $O(\frac{n}{p} \log p)$ computation time and a constant number of communication rounds for $\frac{n}{p} \geq p^2$.

The basic idea of our second algorithm is as follows. In the algorithm, we reduce the number of input elements from n to $\frac{n}{p}$ with a constant number of communication rounds. To achieve the reduction, we use p^2 pivots. (We use only one pivot in each phase of the previous algorithm.) We select p pivots from each processor, and merge the pivots into one sorted sequence. By computing the ranks of the pivots for all input elements, we can find a pair of neighboring pivots such that the kth element is between them. Once the neighboring pivots have been discovered, we discard input elements that are not between them. Since the number of remaining elements becomes at most $\lceil \frac{n}{p^2} \rceil$ on each processor, we can gather all remaining elements in one processor and execute the optimal sequential selection algorithm in $O(p \times \left\lceil \frac{n}{p^2} \right\rceil) = O(\frac{n}{p})$ computation time and a constant number of communication rounds.

In the following, we give an overview of the algorithm.

Selection algorithm with constant communication rounds

Step 1: On each processor P_i, compute a sorted sequence $PV_i = (pv_0^i, pv_1^i, \ldots, pv_{p-1}^i)$ such that pv_j^i is the element whose rank is $\lceil j \times \frac{n}{p^2} \rceil$ in a set of elements on P_i.

Step 2: Compute a sorted sequence $PV = (pv_0, pv_1, \ldots, pv_{p^2-1})$ whose elements are $PV_0 \cup PV_1 \cup \ldots \cup PV_{p-1}$. (We compute PV so that every processor stores a copy of PV.)

Step 3: For each pivot in PV, compute the rank of the pivot among all input elements. (We assume that the results are stored in a sequence $R = (r_0, r_1, \ldots, r_{p^2-1})$, and we compute R so that every processor stores a copy of R.)

Step 4: On each processor P_i, execute the following steps. First, find a pair of neighboring pivots such that the kth element is between them; that is, find a pair of pivots (pv_{j-1}, pv_j) such that $r_{j-1} \leq k \leq r_j$. After finding the pair, compute the rank of pv_{j-1} in elements on the processor, and set L_i to the rank minus 1. (L_i denotes the number of elements that are smaller than pv_{j-1} on the processor P_i.) Discard elements that are not between the pair of pivots.

Step 5: Gather all remaining elements and $L_0, L_1, \ldots, L_{p-1}$ on one processor, and find an element whose rank is $k - \sum_{i=0}^{p-1} L_i$ by using a sequential selection algorithm.

The details of the algorithm are as follows. We can perform Step 1 in $O(\frac{n}{p} \log p)$ time by computing the selection recursively: First we find two pivots whose ranks are $\lceil \lfloor \frac{p}{2} \rfloor \times \frac{n}{p^2} \rceil$ and $(\lceil \lfloor \frac{p}{2} \rfloor + 1) \times \frac{n}{p^2} \rceil$. According to the two pivots, we split the elements into three subsets: The first subset is a set of elements that are smaller than the lower pivot, the second subset is a set

of elements that are larger than the upper pivot, and the third subset is a set of the remaining elements. We compute the two pivots recursively for the first and second subsets.

In Step 2, each processor P_i broadcasts all elements in PV_i. We can perform the broadcast in $O(p \times p) = O(p^2) = O(\frac{n}{p})$ time and one communication round because each processor sends $p \times p = p^2 \leq \frac{n}{p}$ pivots and receives the same number of pivots. After the broadcast, each processor stores sequences $PV_0, PV_1, \ldots, PV_{p-1}$. On each processor, we can compute the sorted sequence PV by merging the sequences, using an optimal sequential sorting algorithm. Since both the number of sorted sequences and the size of each sequence are p, we can sort in $O(p^2 \log p) = O(\frac{n}{p} \log p)$ time on each processor.

In Step 3, we first compute ranks of the pivots in PV for elements on each processor. To compute the ranks, we execute the following steps on each processor P_i. First we find a pair of neighboring pivots for each element on the processor such that the element is between them, by the binary search. We can find the pair of pivots in $O(\log p)$ time for each element since the size of PV is p^2. After finding the pairs for all elements on the processor, we compute, for each pair of neighboring pivots, the number of elements between the two pivots. Let $E_i = (e_0^i, e_1^i, \ldots, e_{p^2-1}^i)$ be a sequence such that e_j^i is the number of elements on a processor P_i between pv_{j-1} and pv_j, except for e_0^i, which denotes the number of elements smaller than pv_0. By computing the prefix sums of E_i, we can compute $R_i = (r_0^i, r_1^i, \ldots, r_{p^2-1}^i)$, which are the ranks of the pivots for elements on P_i. We set $r_j^i = \sum_{g=0}^{j} e_g$ for each j on each processor P_i.

Next we compute ranks $R = (r_0, r_1, \ldots, r_{p^2-1})$ such that $r_0 = \sum_{i=0}^{p-1} r_0^i$, $r_1 = \sum_{i=0}^{p-1} r_1^i$, $\ldots$, $r_{p^2-1} = \sum_{i=0}^{p-1} r_{p^2-1}^i$. To compute these sums evenly on each processor, each processor P_i sends ranks $r_{j\times p}^i, r_{j\times p+1}^i, \ldots, r_{j\times p+p-1}^i$ to a processor P_j. From the received ranks, P_j can compute $r_{j\times p}, r_{j\times p+1}, \ldots, r_{j\times p+p-1}$. After computing a subset of R on each processor, all subsets are broadcast and thus each processor can compute R by merging the received subsets. We can perform all of the above computations of R in $O(\frac{n}{p} \log p)$ computation time and a constant number of communication rounds, because each processor sends and receives $p^2 \leq \frac{n}{p}$ elements and computes locally in $O(p^2 + \frac{n}{p} \log p) = O(\frac{n}{p} \log p)$ computation time.

We can perform Step 4 in $O(\frac{n}{p})$ computation time, because $|R| = p^2 \leq \frac{n}{p}$. Since the number of the remaining elements on each processor is at most $\lceil \frac{n}{p^2} \rceil$ after Step 4, we can perform Step 5 in $O(p + \frac{n}{p^2} \times p) = O(\frac{n}{p})$ time and a constant number of communication rounds.

Consequently, we can solve the selection in $O(\frac{n}{p} \log p)$ computation time and the constant number of communication rounds for $\frac{n}{p} \geq p^2$. Since Goodrich's sorting algorithm[8] can sort m elements in $O(\frac{m}{p} \log m)$ computation time and a constant number of communication round for $\frac{m}{p} \geq p^\epsilon$ and $\epsilon > 0$, the computation time of Goodrich's algorithm is $O(\frac{m}{p} \log p)$ in the case of $\frac{m}{p} = p^\epsilon$ and $\epsilon > 0$. Therefore we obtain the following theorem by combining our selection algorithm and Goodrich's sorting algorithm.

Theorem 2 We can solve the selection with $O(\frac{n}{p} \log p)$ computation time and a constant number of communication rounds by using p processors in CGM and BSP models for $\frac{n}{p} \geq p^\epsilon$ and $\epsilon > 0$.

4.2 Application to Sorting

In the following, we present a sorting algorithm that is an extension of the second selection algorithm. We assume that the inputs and outputs of the sorting are evenly distributed among a set of p processors $P_0, P_1, \ldots, P_{p-1}$; that is, each processor has $\lceil \frac{n}{p} \rceil$ or $\lceil \frac{n}{p} \rceil - 1$ elements at the beginning and at the end of the algorithm.

An overview of the sorting algorithm is as follows.

Sorting algorithm based on selection

Step A: Find $p-1$ elements $T = (t_0, t_1, \ldots, t_{p-2})$ whose ranks are $\left(\lceil \frac{n}{p} \rceil, \lceil 2 \times \frac{n}{p} \rceil, \ldots, \lceil (p-1) \times \frac{n}{p} \rceil\right)$, respectively. To find these elements, an extension of the second selection algorithm is used.
(We compute $T = (t_0, t_1, \ldots, t_{p-2})$ so that every processor stores a copy of T.)

Step B: Each processor sends elements on the processor so that the processor P_0 receives elements smaller than t_0, the processor P_{p-1} receives elements larger than t_{p-2}, and the other processors P_i $(1 \leq i \leq p-2)$ receive elements between t_{i-1} and t_i.

Step C: Sort elements on each processor.

To execute Step A, we use the second se-

lection algorithm with Steps 4 and 5 modified as follows. In the description, we use $T = (t_0, t_1, \ldots, t_{p-2})$, which is used in the above sorting algorithm, and two sets, $PV = (pv_0, pv_1, \ldots, pv_{p^2-1})$ and $R = (r_0, r_1, \ldots, r_{p^2-1})$, which are obtained in Steps 2 and 3 of the second selection algorithm.

Extension of the second selection algorithm for sorting

Step 4: On each processor P_i, execute the followings:

(i) For each j $(0 \leq j \leq p-2)$, find a pair of neighboring pivots in PV, (pv_{x_j-1}, pv_{x_j}), such that t_j is between them; that is, compute x_j that satisfies $r_{x_j-1} \leq \lceil j \times \frac{n}{p} \rceil \leq r_{x_j}$.

(ii) Make $p-1$ subsets of elements TE_0^i, $TE_1^i, \ldots, TE_{p-2}^i$ so that each TE_j^i contains all elements on the processor P_i between $pv_{x_{j-1}}$ and pv_{x_j}.

(iii) Compute the ranks of pv_{x_0-1}, pv_{x_1-1}, …, $pv_{x_{p-2}-1}$ in elements on each processor, and set l_i^j to the rank of pv_{x_j-1} minus 1. (l_j^i denotes the number of elements that are smaller than pv_{x_j-1} on each processor P_i.)

Step 5: Each processor P_i sends (TE_0^i, l_0^i), (TE_1^i, l_1^i), …, (TE_{p-2}^i, l_{p-2}^i), to processors P_0, P_1, …, P_{p-2}, respectively. After receiving all data, each processor P_i $(0 \leq i \leq p-2)$ finds an element t_i whose rank is $\lceil (i+1) \times \frac{n}{p} \rceil - \sum_{g=0}^{p-1} l_i^g$ in $\bigcup_{g=0}^{p-1} TE_i^g$ on the processor. Finally, all t_i $(0 \leq i \leq p-2)$ are broadcast so that every processor stores a copy of T.

In Steps 4 and 5, the broadcast, prefix sums, and sequential sorting are used, and at most $p^2 \leq \frac{n}{p}$ elements are sent and received. Therefore, we can perform these steps in $O(\frac{n}{p} \log \frac{n}{p} + p^2) = O(\frac{n}{p} \log n)$ computation time and a constant number of communication rounds.

We can perform Step B by sorting elements on each processor, computing the ranks of the elements among R, and sending and receiving at most $\lceil \frac{n}{p} \rceil$ elements. Therefore, we can perform Step B in $O(\frac{n}{p} \log n)$ computation time and a constant number of communication rounds. Step C can obviously be performed in the same complexity.

In consequence, we obtain the following theorem.

Theorem 3 We can sort n elements with $O(\frac{n}{p} \log n)$ computation time and a constant number of communication rounds by using p processors in CGM and BSP models for $\frac{n}{p} \geq p^2$.

5. Conclusions

In this paper, we have proposed two selection algorithms and described their application to sorting for the CGM and BSP models. The first selection algorithm runs with $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log \log n))$ communication rounds, and the second selection algorithm runs with $O(\frac{n}{p} \log p)$ computation time and a constant number of communication rounds. We also presented a sorting algorithm that runs with $O(\frac{n}{p} \log n)$ computation time and a constant number of communication rounds for $\frac{n}{p} i \geq p^2$.

Acknowledgments We would like to thank the reviewers for their constructive comments to improve the quality of this paper.

References

1) Al-furaih, I., Aluru, S., Goil, S. and Ranka, S.: Practical Algorithms for Selection on Coarse-Grained Parallel Computers, *IEEE Trans. Parallel and Distributed Systems*, Vol.8, No.8, pp.813–824 (1997).

2) Bader, D.A. and JáJá, J.: Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection, *Proc. 10th International Parallel Processing Symposium*, pp.292–301 (1996).

3) Bäumker, A., Dittrich, W., Heide, F.M. and Rieping, I.: Realistic Parallel Algorithms: Priority Queue Operations and Selection for the BSP Model, *Proc. Second International Euro-Par Conference*, pp.369–376 (1996).

4) Blum, M., Floyd, R., Pratt, V., Rivest, R. and Tarjan, R.: Time Bounds for Selection, *Journal of Commputer and System Sciences*, Vol.7, No.4, pp.448–461 (1973).

5) Dehne, F., Fabri, A. and Rau-Chaplin, A.: Scalable Parallel Computational Geometry for Coarse Grained Multicomputers, *Proc. ACM Symposium on Computational Geometry*, pp.298–307 (1993).

6) Gerbessiotis, A. and Siniolakis, C.: Selection on the Bulk-Synchronous Parallel Model with Applications to Priority Queues, *Proc. 1996 International Conference on Parallel and Distributed Processing Techniques and Applications* (1996).

7) Gerbessiotis, A. and Valiant, L.: Direct Bluk-Synchronous Paralle Algorithms, *Proc. SWAT*

'92, pp.1–18 (1992).
8) Goodrich, M.T.: Communication-Efficient Parallel Sorting, *Proc. 28th Annual ACM Symposium on Theory of Computing* (1993).
9) Ishimizu, T., Fujiwara, A., Inoue, M., Masuzawa, T. and Fujiwara, H.: Parallel Algorithms for Selection on the BSP Model and the BSP* Model, *Trans. IEICE* (DI), Vol.J82, No.4, pp.533–542 (1999).
10) Juurlink, B.H.H. and Wijshoff, H.A.G.: A Quantitative Comparison of Parallel Computation Models, *Proc. 8th Symposium on Parallel Algorithms and Architechtures*, pp.13–23 (1996).
11) Saukas, E. and Song, S.: A Note on Parallel Selection on Coarse-Grained Multicomputers, *Algorithmica*, Vol.24, pp.371–380 (1999).
12) Valiant, L.G.: A Bridging Model for Parallel Computation, *Comm. ACM*, Vol.33, No.8, pp.103–111 (1990).

(Received August 25, 1999)
(Accepted February 4, 2000)

Akihiro Fujiwara received the B.E. degree in Osaka University in 1993, and received the M.E. and Ph.D. degrees in Nara Institute of Science and Technology (NAIST) in 1995 and 1997, respectively. He is now an associate professor of Kyushu Institute of Technology. His main research interests are parallel algorithms, parallel complexity theory and cluster processing. He is a member of IEEE and IEICE.

Michiko Inoue received her B.E., M.E, and Ph.D. degrees in computer science from Osaka university in 1987, 1989, and 1995 respectively. She worked at Fujitsu Laboratories Ltd. from 1989 to 1991. From 1995, she is an instructor of Graduate School of Information Science, Nara institute of Science and Technology (NAIST). Her research interests include distributed algorithms, parallel algorithms, graph theory and design and test of digital systems. She is a member of IEEE, the Institute of Electronics, Information and Communication Engineers, and Japanese Society for Artificial Intelligence.

Toshimitsu Masuzawa received the B.E., M.E. and D.E. degrees in computer science from Osaka University in 1982, 1984 and 1987. He had worked at Education Center for Information Processing, Osaka University between 1987–1990, and had worked at Faculty of Engineering Science, Osaka University between 1990–1994. He is now an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST). He was also a visiting associate professor of Department of Computer Science, Cornell University between 1993–1994. His research interests include distributed algorithms, parallel algorithms and graph theory. He is a member of ACM, IEEE, EATCS and the Institute of Electronics, Information and Communication Engineers.