プログラミング 18−17
(1998. 3. 23)

CGM モデル及び BSP モデル上で選択及びソートを行う並列アルゴリズム

藤原 暁宏† 石水 隆‡ 井上 美智子‡ 増澤 利光‡ 藤原 秀雄‡

†九州工業大学 情報工学部 電子情報工学科
〒820-8502 福岡県飯塚市川津 680-4

‡奈良先端科学技術大学院大学 情報科学研究科
〒630-0101 奈良県生駒市高山町 8916-5

E-mail: fujiwara@cse.kyutech.ac.jp

あらまし

本稿では, 近年注目されている並列計算モデルである CGM モデル及び BSP モデル上で, 要素数 n の選択及びソートを行う決定性の並列アルゴリズムを提案する. まず最初に, 内部計算時間が $O(\frac{n}{p})$ 時間, 通信ラウンド数が $O(\min(\log p, \log\log n))$ のコスト最適な選択を行う並列アルゴリズムを提案する. 次に内部計算時間が $O(\frac{n}{p}\log p)$ 時間, 定数通信ラウンド数の通信ラウンド数が最適な並列アルゴリズムを提案する. 上記の 2 つのアルゴリズムは, $\frac{n}{p} \geq p^{\epsilon}$ かつ $\epsilon > 0$ を満たすプロセッサ数 p に対して動作する. 最後に, 2 つ目の選択アルゴリズムの拡張として, $\frac{n}{p} \geq p^2$ を満たす p に対して, $O(\frac{n}{p}\log n)$ 時間, 定数通信ラウンド数でソートを行うアルゴリズムを提案する.

キーワード 並列アルゴリズム, 選択問題, ソート, CGM モデル, BSP モデル

Parallel selection algorithms for CGM and BSP with application to sorting

Akihiro Fujiwara†, Takashi Ishimizu‡, Michiko Inoue‡, Toshimitsu Masuzawa‡ and Hideo Fujiwara‡

†Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka 820-8502, JAPAN

‡Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0101 Japan

E-mail: fujiwara@cse.kyutech.ac.jp

Abstract

In this paper, we present two deterministic selection algorithms with application to sorting on the CGM (Coarse Grained Multicomputer) model and the BSP (Bulk-Synchronous Parallel) model. The first selection algorithm runs in $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log\log n))$ communication rounds for $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon > 0$, where n is the number of input elements and p is the number of processors. The second selection algorithm runs in $O(\frac{n}{p}\log p)$ computation time and a constant number of communication rounds for $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon > 0$. Furthermore, we apply the second algorithm to sorting. The obtained sorting algorithm runs in $O(\frac{n}{p}\log n)$ computation time and a constant number of communication rounds for $\frac{n}{p} \geq p^2$.

key words Parallel algorithm, CGM, BSP, selection, sorting

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 case of $k = \lceil \frac{n}{2} \rceil$, the element is called the median.) Since the selection problem is a basic problem and plays important roles 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 were also proposed for the problem mainly on the PRAM model or network dependent models (the mesh model, the hypercube model and so on). Since these models are not suited for recent parallel computers, these algorithms are not efficient on the parallel computers in many cases.

For practical use, some parallel computation models were proposed for the recent parallel computers. The BSP (Bulk-Synchronous Parallel) model, which was proposed by Valiant[10], has received considerable attention among the models. The CGM (Coarse Grained Multicomputer) model, which was proposed by Dehne et al.[5], is essentially the same model as the BSP model except for the following points. In the BSP model, communication issues are abstracted using a two parameters, L and g, which denote the latency of the network and the communication throughput ratio, respectively. On the other hand, communication costs are evaluated by the number of communication rounds on the CGM model.

In this paper, we consider selection algorithms for these models. For the BSP model, some randomized selection algorithms were proposed. Gerbessiotis et al.[6] proposed a randomized algorithm which runs in $O(\frac{n}{p} + T_{ppf}(p))$ time with high probability where $T_{ppf}(p)$ is time required for 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 for $B \leq \sqrt{\frac{n}{p}}$ with high probability.[1] In addition, some deterministic and randomized selection algorithms[1, 2] were also proposed on BSP like models. These two algorithms are experimented on real parallel machines.

In this paper, we propose two *deterministic* selection algorithms and its application to sorting. According to a definition of the CGM model, complexities of the algorithms are measured by two complexities, *computation time* and *number of communication rounds*[2]. 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 in $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log\log n))$ communication rounds. The algorithm achieves cost optimality with respect to computation time. In the algorithm, we assume $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon > 0$. Notice that this assumption holds for almost all real parallel computers. In the second selection algorithm, we aim to make the number of communication rounds optimal. Using Goodrich's sorting algorithm[8], we can solve the selection problem in $O(\frac{n}{p}\log p)$ computation time and a constant number of communication rounds in the case of $n = p^{\epsilon}$ and $\epsilon > 0$. Our second selection algorithm runs in the same complexity for $\frac{n}{p} \geq p^2$. By combining the Goodrich's sorting algorithm and our algorithm, we can solve the selection problem in the above complexity for $\frac{n}{p} \geq p^{\epsilon}$ and $\epsilon > 0$. Since the number of processors on parallel machines is usually fixed, the second algorithm may be faster than the first algorithm in some situations. Furthermore, we present a sorting algorithm by modifying the second selection algorithm. The algorithm sort n elements in $O(\frac{n}{p}\log n)$ computation time and a constant number of communication rounds for $\frac{n}{p} \geq p^2$. Although the number of processors p is restricted to no more than $n^{\frac{1}{3}}$, the algorithm is simple and is faster than other cost optimal sorting algorithms in some situation.

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

2 Preliminaries

2.1 Models

The *rank* of an element in a totally ordered set is the number of elements smaller than or equal to the element. The selection problem is a problem to find 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 the CGM model and the BSP model, 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 the algorithm.

[1] Their algorithm assumes BSP* model, which is an extended model of the BSP model.

[2] We can characterize the algorithm on the BSP model by these two complexities and BSP parameters. We describe the method in Section 2.

Both the CGM model and the BSP model consist of three parts: a set of processor modules, communication network for module-to-module communication and a synchronizer which synchronizes all or a subset of processors in barrier style. In this paper, we assume 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 communication round. Therefore data received in a communication round cannot be used in the local computation round in the same superstep. After all processors complete their supersteps, the synchronizer makes all processors start the next superstep.

Complexities of algorithms for the CGM model are measured by two complexities *computation time* and *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 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 superstep i, and let $comm_i$ be the maximum number of data sent or received by one processor among all communication rounds in superstep i. We define that the computation time T_{comp} is a sum of $comp_i$ and $comm_i$ for all supersteps, that is,

$$T_{comp} = \sum_{i=1}^{S}(comp_i + comm_i).^3$$

We also assume that each processor can send $O(\frac{n}{p})$ elements and receive $O(\frac{n}{p})$ elements in each superstep. Although "packing requirement"[4] is assumed in some papers for CGM algorithms, we do not use the assumption in this paper because the assumption increases power of the model.

On the BSP model, two parameters, g and L, are used to denote communication costs. The parameter g denotes the ratio of computation and communication throughput, and L denotes the minimal time to perform a synchronization. The running time of an algorithm on the BSP model is easily measured by these two parameters and complexities on 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 for the CGM model, respectively.

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

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

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 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 which is partitioned into p subsets $A_0, A_1, \ldots, A_{p-1}$. We assume that each processor P_i stores subset A_i and $u_i = |A_i|$. The 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 using the prefix sums as follows.

(1) Each processor P_i computes u_i.

(2) Compute prefix sums of u_i $(0 \leq i \leq p-1)$ by all processors. Let PS_i be the result for 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 processor $P_{\lceil (PS_i - u_i + j + 1)/\frac{u}{p} \rceil - 1}$.

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 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 on the p-processor CGM model and BSP model where $T_{ppf}^{comp}(p)$ and $T_{ppf}^{comm}(p)$ are computation time and the number of communication rounds of the prefix sums, respectively.*

3 Cost optimal selection algorithm

3.1 Basic idea

Our first selection algorithm is based on a well-known strategy, "median of medians". The strategy is proposed as a sequential algorithm by Blum[4], and utilized in some parallel algorithms[1, 2]. We also use the median, whose rank is $\lceil \frac{m}{2} \rceil$ among m elements, as a pivot to split elements.

The overview of our algorithm is as follows. 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 a following phase until $m_s \leq \max(\frac{n}{p}, \frac{n}{\log n})$. ($m_i$ denotes the number of remaining elements in 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 is described in the following subsection.) Let MM be the median of the medians.

(3) Broadcast MM to all processors.

(4) On each processor P_i, split elements on the processor into two subsets L_i and U_i. Subset L_i contains elements which are smaller than MM, subset U_i contains elements which are larger than MM.

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

- discard elements contained in $U_i \cup \{MM\}$ and set $k_{s+1} = k_s$. (If $k_s \leq SUM_L$)
- output MM and end algorithm (If $k_s = SUM_L + 1$)
- 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 all processors and find the k_sth smallest element. Otherwise, gather all elements on one processor and execute the optimal sequential selection algorithm on the processor.

The key point of this strategy is to ensure $m_{s+1} \leq \frac{1}{\delta} m_s$ 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 (which is used in (3)), the prefix sums (which is used (5)) and the pivot computation in (2). The computation time $O(\frac{n}{p} + \min(\log p, \log\log n))$ is $O(\frac{n}{p})$ 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 from the following reasons. If $\frac{n}{\log n}$ elements remain, we use Goodrich's sorting algorithm[8]. The algorithm sort 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 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 the result in the same complexity obviously by the optimal sequential algorithm[4].

Consequently, we can perform the selection algorithm in $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log\log n))$ communication rounds if three operations, which are the broadcast, the prefix sums and the 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 first describe details of the broadcast and the prefix sums, and finally describe details of the pivot computation.

3.2 Broadcast and prefix sums

In this subsection, we show the broadcast and the 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 which satisfies the following conditions.

- Each non-leaf node has d children exactly.

- All leaves are at level $\lceil \frac{\log p}{\log d} \rceil$ where p is the number of processors.

Using the d-ary tree, we can perform the broadcast and the prefix sums in $O(d \times \frac{\log p}{\log d})$ computation time and $O(\frac{\log p}{\log d})$ communication rounds[9]. To perform these operation with the complexity described above, we set $d = \lceil \frac{n}{p \log n} \rceil$, and prove $\frac{\log p}{\log d} = O(1)$.

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

Therefore we can perform the broadcast and the 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) 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 computation of MM

Set $g = 1, l_g = p$ and repeat the following phase until $l_g = 1$.
(The l_g denotes the number of remaining medians in a phase g.)
After all phases complete, set MM to the remained median.

Step 1: Gather the medians on $\lfloor \frac{l_g}{d} \rfloor$ processors so that the number of elements on the processors differs 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) Processor $P_{\lfloor \frac{l_g}{d} \rfloor + 1}$, which stores the $d-1$ medians or less, sends gathered medians to $P_0, P_1, \ldots, P_{\lfloor \frac{l_g}{d} \rfloor}$ evenly.

Step 2: On each processor which stores the gathered medians, find the median of medians on the processor by the optimal 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 all processors and set $g = g + 1$ on each processor.

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

The remaining work is to prove that $m_{s+1} \leq \frac{1}{\delta} m_s$ is ensured for a constant $\delta > 1$ in every phase s of the algorithm by the obtained MM. (Remember 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 probe that $\sum_{i=0}^{p-1} |L_i| \leq (1 - \frac{1}{2^{r+1}}) m_s$ holds in case of $m_s = \sum_{i=0}^{p-1} |L_i|$, where r is a constant which denotes the number of communication rounds of the computation of the median of medians.

(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 d-ary tree such that $d = \lceil \frac{n}{p \log n} \rceil$.)

Let EU_g be the number of the remaining medians which is not less than MM at the beginning of a phase g of the pivot computation. From the definition of the median, $EU_r = \lceil \frac{l_r}{2} \rceil$ holds. Since $EU_g = EU_{g+1} \times \lceil \frac{d}{2} \rceil$,

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

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

$$(\sum_{i=0}^{p-1} |U_i|) + 1 \quad \geq \quad EU_1 \times \left\lceil \frac{\frac{m_s}{p}}{2} \right\rceil$$

$$= \frac{1}{2^{r+1}} m_s$$

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

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

4 Algorithm with constant communication rounds

4.1 Selection algorithm

We show a second selection algorithm which runs in $O(\frac{n}{p} \log p)$ computation time and a constant number of communication rounds for $\frac{n}{p} \geq p^2$. However we can solve the selection problem in the same complexity for $\frac{n}{p} > p^\epsilon$ and $\epsilon > 0$ by combining with the Goodrich's sorting algorithm[8].

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 discovered, we discard input elements which are not between them. Since the 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 $O(p \times \lceil \frac{n}{p^2} \rceil) = O(\frac{n}{p})$ computation time and a constant number of communication rounds.

In the following, we describe an overview of the algorithm. In the description, we find the kth element.

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 which are smaller than pv_{j-1} on the processor P_i.) Discard elements which 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 the optimal sequential selection algorithm.

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: First subset is a set of elements which is smaller than the lower pivot, second subset is a set of elements which is larger than the upper pivot, and third subset is a set of remaining elements. We compute the two pivots recursively for the first subset and the second subset.

In Step 2, first 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 store 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 the number of sorted sequence and the size of each sequence are both p, we can sort in $O(p^2 \log p) = O(\frac{n}{p} \log p)$ time on each processor.

In Step 3, first we compute ranks of the pivots in PV for elements on each processor. To compute the ranks, we execute the following steps on processor P_i. First we decide a pair of neighboring pivots for each elements on the processor such that the element is between them, by 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 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 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 $r_0 = \sum_{i=0}^{p-1} r_0^i$, $r_1 = \sum_{i=0}^{p-1} r_1^i$, $\cdots$, $r_{p^2-1} = \sum_{i=0}^{p-1} r_{p^2-1}^i$. 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 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 broadcasted so that 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 communications 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 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 problem in $O(\frac{n}{p}\log p)$ computation time and a 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 $\frac{m}{p} = p^\alpha$ and $\alpha > 0$. Therefore we obtain the following theorem by combining our selection algorithm and the Goodrich's sorting algorithm.

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

4.2 Application to sorting

In the following, we present a sorting algorithm, which is an extension of the second selection algorithm. We assume that 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 the 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, the 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 processor P_0 receives elements smaller than t_0, 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 selection algorithm with modifying of Step 4 and Step 5 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 Step 2 and Step 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 which 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, ..., 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
ranks of $pv_{x_0-1}, pv_{x_1-1}, \ldots, pv_{x_{p-2}-1}$ 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 which 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 broadcasted so that every processor stores a copy of T.

In Step 4 and Step 5, the broadcast, the prefix sums, 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 of the sorting algorithm by sorting elements on each processor, computing 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 be performed in the same complexity obviously.

In consequence, we obtain the following theorem.

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

5 Conclusions

In this paper, we proposed two selection algorithms and its application to sorting for the CGM model and the BSP model. The first selection algorithm runs in $O(\frac{n}{p})$ computation time and $O(\min(\log p, \log \log n))$ communication rounds, and the second selection algorithm runs in $O(\frac{n}{p} \log p)$ computation time and a constant number of communication rounds. Furthermore, we presented a sorting algorithm which runs in $O(\frac{n}{p} \log n)$ computation time and constant communication rounds for $\frac{n}{p} \geq p^2$.

In the near future, we implement these algorithms on a cluster of PCs, and verify the advantage of our algorithms.

References

[1] I. Al-furaih, S. Aluru, S. Goil, and S. Ranka. Practical algorithms for selection on coarse-grained parallel computers. In *Proc. 10th International Parallel Processing Symposium*, pages 309–313, 1996.

[2] D. A. Bader and J. JáJá. Practical parallel algorithms for dynamic data redistribution, median finding and selection. In *Proc. 10th International Parallel Processing Symposium*, pages 292–301, 1996.

[3] A. Bäumker, W. Dittrich, F. Meyer auf der Heide, and I. Rieping. Realistic parallel algorithms: Priority queue operations and selection for the BSP model. In *Proc. Second International Euro-Par Conference*, pages 369–376, 1996.

[4] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Commputer and System Sciences*, 7(4):448–461, 1972.

[5] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. In *Proc. ACM Symposium on Computational Geometry*, pages 298–307, 1993.

[6] A.V. Gerbessiotis and C.J. Siniolakis. Selection on the Bulk-Synchronous Parallel model with applications to priority queues. In *Proc. 1996 International Conference on Parallel and Distributed Processing Techniques and Applications*, April 1996.

[7] A.V. Gerbessiotis and L.G. Valiant. Direct bluk-synchoronous paralle algorithms. In *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, pages 1–18, 1992.

[8] M. T. Goodrich. Communication-efficient parallel sorting. In *Proc. 28th annual ACM Symposium on Theory of Computing*, 1993.

[9] B. H. H. Juurlink and H. A. G. Wijshoff. A quantitative comparison of parallel computation models. In *Proc. 8th Symposium on Parallel Algorithms and Architechtures*, pages 13–23, 1996.

[10] L. G. Valiant. A bridging model for parallel computation. *Communication of the ACM*, 33(8):103–111, 1990.