A Note on Parallel Selection on Coarse Grained Multicomputers*

E. L. G. Saukas and S. W. Song

University of São Paulo
Institute of Mathematics and Statistics
Department of Computer Science
São Paulo, SP 05508-900 Brazil

Abstract. Consider the selection problem of determining the kth smallest element of a set of n elements. Under the CGM (coarse-grained multicomputer) model with p processors and $O(n/p)$ local memory, we present a deterministic parallel algorithm for the selection problem that requires $O(\log p)$ communication rounds. Besides requiring a low number of communication rounds, the algorithm also attempts to minimize the total amount of data transmitted in each round (only $O(p)$ except in the last round). In addition to showing theoretical complexities, we present very promising experimental results obtained on a parallel machine that show almost linear speedup, indicating the efficiency and scalability of the proposed algorithm.

Key-words: Coarse grained multicomputer, parallel algorithm, selection problem

* Supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) Proc. No. 98/06138-2, CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) Proc. No. 523778/96-1 and Commission of the European Communities through Project ITDC-207-82167.

1 Introduction

Current applications in parallel machines are restricted to trivially parallelizable problems with low communication requirements. In real machines communication time is often much greater than computation time. Therefore for non-trivial problems many theoretically efficient parallel algorithms for PRAM or fine-grained network models often give disappointing speedups when implemented on real machines.

The CGM (Coarse Grained Multicomputer) model was proposed [9, 10, 11] to be an adequate model of parallelism sufficiently close to existing parallel machines. It is a simple model and nevertheless intends to give a reasonable prediction of performance when parallel algorithms on this model are implemented. CGM algorithms are expected to have theoretical complexity analyses close to actual times observed in real implementations.

Under the CGM model we design a deterministic parallel algorithm for the selection problem to determine the kth smallest element in a set of n elements. This algorithm requires $O(\log p)$ communication rounds, where p is the number of processors. One crucial aspect in the design of this algorithm is the special effort to reduce the amount of data transmitted in each communication round. This aspect (not emphasized in the CGM model) considered in this work turns out to be crucial in the experimental results. The total amount of data transmitted is only $O(p)$ in each communication round except in the last one. Experimental results run on a Parsytec PowerXplorer distributed memory parallel machine show an almost linear speedup for sufficiently large values of n. This speedup is obtained by comparing the parallel times with that of an optimized sequential algorithm. Furthermore, given the fact that this particular machine presents a considerable communication latency and low communication bandwidth, the almost linear speedup is a very significant result. To our knowledge, this is the best deterministic CGM algorithm known in the literature for the selection problem.

In section 2 we present the CGM model. The selection problem is presented in section 3. Section 4 contains some concluding remarks. The proposed algorithm has been implemented on a real parallel machine. At the end of the algorithm presentation we give and discuss experimental results. Time and speedup curves of the proposed algorithm are given at the end of this paper.

2 The Coarse Grained Multicomputer (CGM)

The CGM (Coarse Grained Multicomputer) model [9, 10, 11] is similar to Valiant's BSP (Bulk-Synchronous Parallel) model [16]. However it 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 considerably greater than $O(1)$. All processors are connected by a router that can deliver messages in a point to point fashion. A CGM algorithm consists of an alternating sequence of *computation rounds* and *communication rounds* separated by barrier synchronizations.

A computation round is equivalent to a computation superstep in the BSP model. In this computation round, we usually use the best possible sequential algorithm in each processor to process its data locally.

A communication round consists of a single h-relation with $h \leq n/p$, that is, each processor can exchange a total of $O(n/p)$ data with other processors in one communication round. (The selection algorithm proposed in this paper is designed to require the transmission of only $O(p)$ total amount of data in each communication round except in the last one.)

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. Many algorithms for graph and geometric problems [3, 9, 10, 11] require only a constant or $O(\log p)$ rounds. As p is usually small compared to n the resulting CGM algorithm is expected to be efficient when implemented in practice. Observe that PRAM algorithms frequently are designed for $p = O(n)$ and each processor receives a small amount of input data. The CGM model considers a more realistic case of $p << n$.

3 Selection problem

Consider a set X of n elements. For simplicity we assume the elements are distinct. (The proposed algorithms, however, also work without this assumption.) 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. The selection problem can be solved by a sequential deterministic algorithm in linear time [2].

This problem has been extensively studied for the PRAM model [4, 5, 6, 7, 12, 14]. There exist, however, very few works on this problem for distributed memory models. In [15] the selection problem is solved by a probabilistic CGM algorithm using a total of O($\log \log k + \log p$) communication rounds, where p is the number of processors, but this algorithm is based on a strong assumption of uniform random distribution of the elements among the processors. A deterministic parallel algorithm [1] using the block-distributed memory model takes $O(\log(n/p^2))$ communication rounds. Another recent randomized algorithm takes an expected constant number of communication rounds to find the median of n elements [13].

Using the CGM paradigm, we present a new parallel deterministic algorithm that solves the selection problem in $O(\log p)$ communication rounds. This new algorithm is based on successive partitioning of the input data 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.

The following partitioning strategy is used. Choose an element M in set A with the following property: $\frac{n}{4} \leq rank(M, A) \leq \frac{3n}{4}$. Partition A into three subsets A_l, A_e, A_g, respectively with elements less than, equal to, and greater than M. We have $|A_l| \leq \frac{3n}{4}$ and $|A_g| \leq \frac{3n}{4}$. Based on the value of k relative to the sizes of the three subsets A_l, A_e, A_g, we either find the kth element, or continue the selection with a smaller subset. If k corresponds to the first partition A_l, then we continue the selection using subset A_l and discard the rest (see Fig. 1); 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 subset 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.

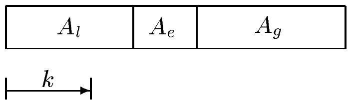

Fig. 1. Position of k relative to the sizes of A_l, A_e, and A_g

It suffices to show how to find such an element M with the above desired property. At each round the elements are partitioned into p blocks and medians of each block are obtained. We show that the weighted median [8] of the medians of the blocks has such a property. Thus, after each round the number of elements is reduced by at least 1/4.

The median of the medians of the blocks also has the desired property [2]. It has been used in some previous algorithms [1, 17]. This would however require equal block sizes at each round. Thus redistribution of data among processor would be needed at each round. This is actually what previous algorithms do to have equal block sizes. In our algorithm, in addition to attempt to minimize the number of communication rounds, a special effort is made to reduce the amount of data transmitted in each communication round. By using the weighted median of the blocks' medians to do the partitioning, redistribution of data is not needed. With this we get a substantial saving in communication time that more than compensates the computation time due to unbalanced local processing.

The weighted median is defined as follows [8]. Given p distinct elements $m_1, m_2, \ldots, m_p$ with corresponding positive weights $w_1, w_2, \ldots, w_p$ such that $\sum_{1 \leq i \leq p} w_i = 1$, the weighted median is the element m_k that satisfies

$\sum_{i, m_i < m_k} w_i \leq \frac{1}{2}$ and $\sum_{i, m_i > m_k} w_i \leq \frac{1}{2}$.

Let n_i be the number of remaining elements in processor i at each round and $N = \sum_{1 \leq i \leq p} n_i$. We consider m_i as the median of the remaining elements in processor i. The weights w_i are given by $w_i = n_i/N$. Therefore using the weighted median we not only consider the values of the medians in each block, but also the amount of remaining elements in each block. This will allow the subsequent rounds to deal with different number of elements in each processor and nevertheless guarantees the reduction by at least 1/4 of the total number of remaining elements at each round.

We now present Algorithm 1 which is the basic algorithm for selection. In this algorithm, c is an adjustable arbitrary parameter that we can take as a constant.

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

Input: Set 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 $N \leq n/(cp)$

(2.1) Each processor i computes the median m_i of its n_i elements

(2.2) Each processor i sends m_i and n_i to processor 1

(2.3) Processor 1 computes the weighted median M

(2.4) Processor 1 broadcasts M to all other processors

(2.5) Each processor i computes l_i, e_i, g_i, respectively the numbers of its local elements less than, equal to, or greater than M

(2.6) Each processor i sends l_i, e_i, g_i to processor 1

(2.7) Processor 1 computes $L = \sum_{1 \leq i \leq p} l_i$, $E = \sum_{1 \leq i \leq p} e_i$, $G = \sum_{1 \leq i \leq p} g_i$, respectively the total numbers of elements less than, equal to, or greater than M

(2.8) Processor 1 broadcasts L, E, G to all other processors

(2.9) One of the following:

if $L < k \leq L + E$ then return solution M and stop

if $k \leq L$ then each processor i discards all but those elements less than M and set $N := L$

if $k > L + E$ then each processor i discards all but those elements greater than M and set $N := G$ and $k := k - (L + E)$

(3) All the remaining N elements are sent to processor 1

(4) Processor 1 solves the remaining problem sequentially

— End of Algorithm —

Theorem 1. *After each round (step 2), the number of elements is reduced by at least 1/4.*

Proof. At the beginning of each round, the total number of remaining elements is N. By the definition of weighted median M,

$$\sum_{i,m_i<M} w_i \leq \frac{1}{2} \Rightarrow \frac{\sum_{i,m_i<M} n_i}{N} \leq \frac{1}{2} \Rightarrow \sum_{i,m_i<M} n_i \leq \frac{N}{2}$$

$$\sum_{i,m_i>M} w_i \leq \frac{1}{2} \Rightarrow \frac{\sum_{i,m_i>M} n_i}{N} \leq \frac{1}{2} \Rightarrow \sum_{i,m_i>M} n_i \leq \frac{N}{2}$$

Since $N = \sum_i n_i$, then $\sum_{i,m_i\leq M} n_i \geq \frac{N}{2}$ and $\sum_{i,m_i\geq M} n_i \geq \frac{N}{2}$. For each local median m_i, the number of local elements a_j such that $a_j \leq m_i$ is at least $n_i/2$, by the definition of median. Hence the weighted median M is greater or equal than at least $\sum_{i,m_i\leq M} \frac{n_i}{2} \geq \frac{N}{4}$ elements. Analogously, M is less or equal than at least $\sum_{i,m_i\geq M} \frac{n_i}{2} \geq \frac{N}{4}$ elements. Therefore the total number of remaining elements that are strictly less than M is $L \leq (3N/4)$, and that are strictly greater than M is $G \leq (3N/4)$. Since at the next round only L or G elements remain, after each round the total number of remaining elements is reduced by at least $1/4$. □

Theorem 2. *The selection problem of the kth smallest element of a set of n elements can be solved on a CGM with p processors and $O(\frac{n}{p})$ local memory per processor using $O(\log p)$ communication rounds and $O(\frac{n}{p})$ local computation per round, if $n > p^2 \log p$.*

Proof. Theorem 1 can be used to determine the number of iterations of the main loop of the algorithm. Taking c as a constant, N can be reduced from n to $n/(cp) = O(n/p)$ in at most $\log_{4/3}(cp) = O(\log p)$ communication rounds.

The local processing at each round consists of computing the local median (in linear $O(n/p)$ time), the weighted median of p medians (in time $O(p \log p)$), and partitioning of local elements (in linear $O(n/p)$ time). At the final round (step 4) the local processing involves the solution of a sequential selection problem with $N \leq n/(cp)$ elements (in linear $O(n/(cp))$ time). So the local computation time is $O(n/p)$ per round, for $n > p^2 \log p$. □

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 (corresponding to step 3), 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.

In the following we discuss very encouraging experimental results of this algorithm implemented on a distributed memory parallel machine. The parallel machine utilized for this experiment is the Parsytec PowerXplorer 16/32 Parallel Computing System, with 16 nodes under a 2-D topology. Each node consists of one PowerPC601 application processor and one T805 communication processor, with 32 MBytes of main memory. The total main memory is therefore 512 Mbytes. To ensure portability, the implementation is done in PVM (Parallel Virtual Machine), a standard communication interface.

The proposed algorithm is deterministic, so it should always work efficiently (regardless of how the input elements are distributed initially among the processors). A probabilistic algorithm as [15] does not guarantee that it will always work efficiently.

To show this, we consider the cases of input elements distributed at random among the processors and also distributed in the worst possible way (regarding parallelization). The worst distribution is an ordered set, when processor 1 contains the smallest subset of elements, processor 2 the next smallest subset, and so on with the last processor holding the largest elements. k is chosen to be n. We refer to this as the *worst input distribution* in the sense that it is very unfavorable for the parallel algorithm (because it causes the worst "load balancing" and requires a great amount of communication at the last round) but very favorable for the sequential algorithm. So it is the "most difficult" case to obtain parallel times smaller than sequential times.

The results are given in the following tables. (See also Fig. 2 and Fig. 3 at the end.) Each experiment is run 20 times. We show the average time of the 20 experiments. They are given in units of clock ticks of the machine (1 clock tick $= 10^{-6}$ seconds).

The sequential time was obtained with an optimized deterministic sequential algorithm run in a single processor (not the parallel algorithm run in one processor).

Algorithm 1 - Uniform input distribution, $k = \lfloor n/3 \rfloor + 1$ and $c = 100$
Average times in 20 executions (and number of rounds between parenthesis):

n	$p=1$	$p=2$	$p=4$	$p=8$	$p=16$
65536	346626 (0)	368494 (8)	191942 (9)	112763 (10)	99103 (11)
131072	700070 (0)	743135 (8)	376584 (9)	201463 (10)	142898 (11)
262144	1401059 (0)	1477383 (8)	741230 (9)	389717 (10)	231555 (11)
524288	2814936 (0)	3013741 (8)	1484357 (9)	754734 (10)	420821 (11)
1048576	5623402 (0)	5969708 (8)	3020690 (9)	1522502 (10)	783559 (11)

Algorithm 1 - Uniform input distribution, $k = \lfloor n/3 \rfloor + 1$ and $c = 100$
Speedup using the average times in 20 executions:

n	$p = 2$	$p = 4$	$p = 8$	$p = 16$
65536	0.9407	1.8059	3.0739	3.4976
131072	0.9420	1.8590	3.4749	4.8991
262144	0.9483	1.8902	3.5951	6.0507
524288	0.9340	1.8964	3.7297	6.6892
1048576	0.9420	1.8616	3.6935	7.1767

Observe the nearly linear speedup for sufficiently large values of n. Given that the sequential algorithm is very efficient and has linear complexity, and that this particular machine presents a considerable communication latency and low communication bandwidth, the almost linear speedup of the proposed parallel algorithm is very significant and indicates the efficiency in practice and scalability of this algorithm.

Algorithm 1 - Worst input distribution, $k = n$ and $c = 100$
Average times in 20 executions (and number of rounds between parenthesis):

n	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
65536	140273 (0)	257946 (8)	178125 (9)	126605 (10)	120916 (11)
131072	280901 (0)	514701 (8)	350514 (9)	234881 (10)	180372 (11)
262144	563144 (0)	1030400 (8)	695963 (9)	451382 (10)	310637 (11)
524288	1129618 (0)	2066309 (8)	1389332 (9)	885263 (10)	571514 (11)
1048576	2259523 (0)	3864623 (8)	2781234 (9)	1756322 (10)	1093314 (11)

Algorithm 1 - Worst input distribution, $k = n$ and $c = 100$
Speedup using the average times in 20 executions:

n	$p = 2$	$p = 4$	$p = 8$	$p = 16$
65536	0.5438	0.7875	1.1080	1.1601
131072	0.5458	0.8014	1.1959	1.5573
262144	0.5465	0.8092	1.2476	1.8129
524288	0.5467	0.8131	1.2760	1.9765
1048576	0.5847	0.8124	1.2865	2.0667

Regarding the experimental results, we observe the following. For a fixed p, the larger is the number of elements n the better the speedup. The reason is that the communication overhead at each except the last round does not depend on n (it depends only on p and c). Therefore the larger the value of n the smaller is the influence of the communication on the result. For $n >> p$ the speedup grows almost linearly in p. This behavior is observed even in the worst input distribution case, which is very unfavorable to parallelization. Even so, we have a practically linear speedup relative to p (for sufficiently large n).

4 Conclusion

Though PRAM and fine-grained network algorithms abound for the selection problem, there exist few works on the CGM model. Using the CGM (coarse-grained multicomputer) model we present a deterministic parallel algorithm for the selection problem.

The deterministic algorithm for selection requires $O(\log p)$ communication rounds. The design of this algorithm takes into account both the reduction on the number of communication rounds and the total amount of data transmitted in each round. This aspect turns out to be crucial in the experimental results with almost linear speedup. To our knowledge, this is the best deterministic CGM algorithm known in the literature for this problem.

In addition to showing theoretical complexities, we have implemented the proposed algorithm on a real parallel machine. The experimental results show almost linear speedup, indicating the efficiency and scalability of this algorithm.

We conclude 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.

Acknowledgments

We wish to thank the referees for their comments.

References

1. J. A. Bader and J. JáJá. Practical parallel algorithms for dynamic data redistribution, median finding, and selection. Proc. 10th International Parallel Processing Symposium (1996) 292–301.
2. M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest and R. E. Tarjan. Bounds for selection. Journal of Computer and System Sciences **7** (1973) 448–461.
3. E. Caceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S. W. Song. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. Proceedings ICALP'97 - 24th International Colloquium on Automata, Languages, and Programming, P. Degano and A. Marchetti-Spaccamela (editors). Lecture Notes in Computer Science 1256 (1997) 390–400.
4. S. Chaudhuri, T. Hagerup and R. Raman. Approximate and exact deterministic parallel selection. Proc. 18th International Symposium on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science **711** (1993) 352–361.
5. R. J. Cole. Parallel merge sort. SIAM Journal on Computing **17** (1988) 770–785.
6. R. J. Cole. An optimally efficient selection algorithm. Information Processing Letters **26** (1988) 295–299.
7. R. J. Cole, and U. Vishkin. Deterministic coin tossing and accelerated cascades: micro and macro techniques for designing parallel algorithms. Proc. 18th Annual ACM Symposium on the Theory of Computing (1986) 206–219.

8. T. H. Cormen, C. E. Leiserson and R. L. Rivest. Introduction to Algorithms, MIT Press (1990).
9. F. Dehne, A. Fabri and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputers. Proc. ACM 9th Annual Computational Geometry (1993) 298–307.
10. F. Dehne, A. Fabri and C. Kenyon. Scalable and Architecture Independent parallel geometric algorithms with high probability optimal time. Proc. 6th IEEE Symposium on Parallel and Distributed Processing (1994) 586–593.
11. F. Dehne, X. Deng, P. Dymond, A. Fabri and A. A. Kokhar. A randomized parallel 3D convex hull algorithm for coarse grained multicomputers. Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA'95) (1995) 27–33.
12. P. F. Dietz and R. Raman. Very fast optimal parallel algorithms for heap construction. Proc. 6th IEEE Symposium on Parallel and Distributed Processing (1994), pp. 1 - 22.
13. A. V. Gerbessiotis and C. J. Siniolakis. Deterministic sorting and randomized median finding on the BSP model. *Proc. ACM Symposium on Parallel Algorithms and Architectures* (1996) 223–232.
14. J. JáJá. An Introduction to Parallel Algorithms. Addison-Wesley (1992).
15. N. Santoro, S. B. Sidney and S. J. Sidney. A distributed selection algorithm and its expected communication complexity. Theoretical Computer Science **100** (1992) 185–204.
16. L. G. Valiant. General purpose parallel architectures. Handbook of Theoretical Computer Science, Edited by J. van Leeuwen, MIT Press/Elsevier (1990) 943–972.
17. U. Vishkin. An optimal parallel algorithm for selection. Advances in Computing Research. JAI Press Inc. Greenwich, CT (1987), pp. 79 - 85.

This article was processed using the LaTeX macro package with LLNCS style

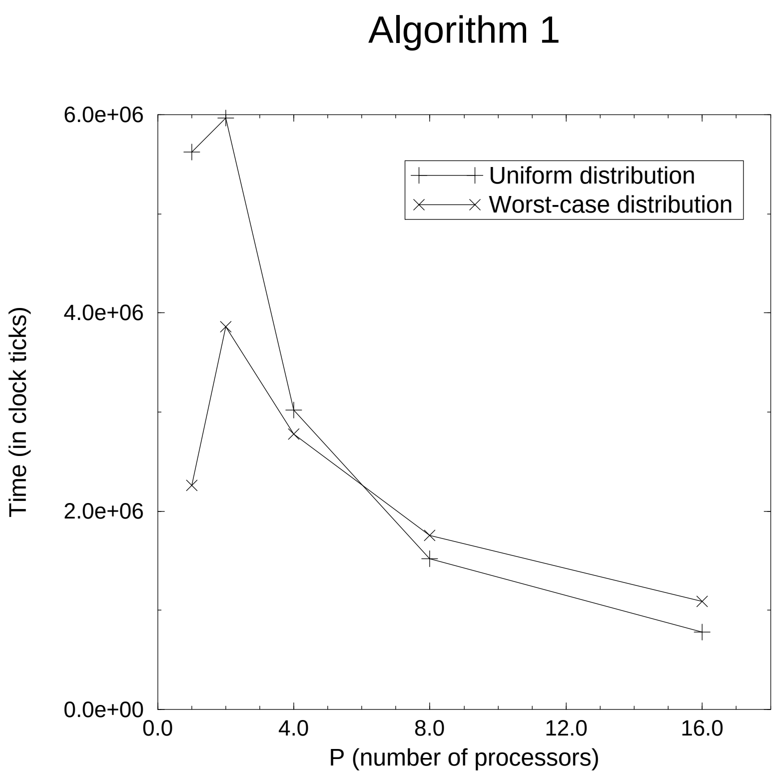

Fig. 2. Time of the selection algorithm for $n = 1048576$

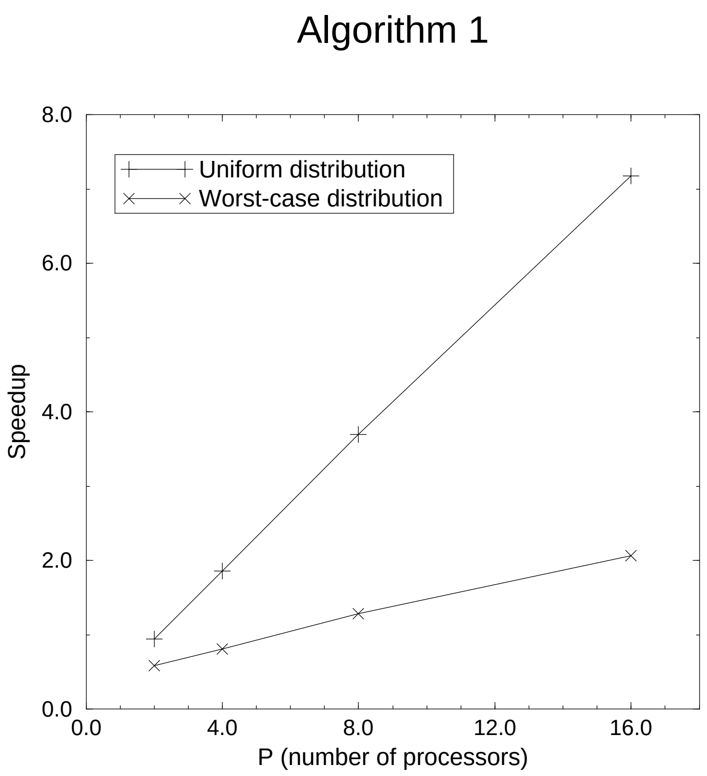

Fig. 3. Speedup of the selection algorithm for $n = 1048576$