A Study on Sorting in A Computer Using The Binary Multi-level Multi-access Protocol

Chang-Duk Jung, PhD
Professor, Korea University
jcd1234@korea.ac.kr

Abstract

The sorting algorithms have been developed to take advantage of distributed computers. But the speedup of parallel sorting algorithms decrease rapidly with increased number of processors due to parallel processing overhead such as context switching time and inter-processor communication cost. In this paper, we propose a parallel sorting method which provides linear speedup of an optimal serial algorithm for a system with a large number of processors. This algorithm may even provide superlinear speedup for a practical system. The algorithm takes advantage of an interconnection network properties and its protocol.

***Key Words** : sorting, network protocol, superlinear speedup*

1 Introduction

1.1 The Sorting of computer

sorting is one of the most time-consuming operations which appear frequently in many applications [1, 2]. The problem of sorting is that , unless elements of be sorted have some know properties, the optimal time complexity of any sorting algorithm for which the only operations permitted on keys are comparisons and interchanges is O(n log n), where n is the number of elements to be sorted. Thus, sorting time increases rapidly with n.to be sorted. Thus, sorting time increases rapidly with n.

The simplest way to implement parallel *sorting is to divide the set into* n equal size subsets and to distribute them to each processor. after each processor completes the sorting of its subset (local sorting), the n sorted subsets may be merged together (sorting). The main difficulty with this *divide – and – conquer* sorting strategy is the need for and effective merging operation. Since most merging operations require many rounds of message exchange per element,

researchers have assumed that the cost of parallel sorting os dominated by its communication costs. Thus, they have concentrated on devising algorithms requiring low communication costs, i .e., less rounds of message exchange [3, 5, 6]. Mikkilinent and Su [7] evaluated the performance of sorting algorithms for common-bus local networks. The result showed that for most algorithms, the execution time of sorting actually increases with the number of processors in the network. Even for the best algorithm, the execution time of the algorithm decreases initially but, after the network size reaches a certain limit, the decrease in execution time of the local sorting step is outweighed by the increase in execution time of the merging process [7].

One of the theoretical debates in the area of parallel computer is whether superlinear speedup of an efficient serial algorithm is possible of not [10, 8]. Contradictory results have been published based on varying assumptions. If the serial computing model ignores context switching time, one can conclude that superlinear speedup is impossible [10]. If the parallel computing model ignores interprocessor communications overhead, one may conclude that superlinear speedup is possible[8]. However, for a practical parallel computer, the context switching time and the communication overhead cannot be ignored. moreover, even if we ignore the context switching time and the communication overhead, it is still possible to achieve superlinear speedup if we can take advantage of extra hardware in the parallel computer.

2.1 Parallel Computer

A parallel computer is a set of N processor and *a communicartion subnet.* Most parallel computing models only deal with N processors and ignore the communication subnet or treat it as only overhead. However, the communication subnet represents extra hardware which has substantial computational power for communication and routing messages. Thus, a parallel computer could achieve superlinear speedup if it could take advantage of the computational power of its communication subnet. There are communication protocols which provide the sorting of messages as a by-product of the communication protocol [12]. Rothauser and Wild [13] have proposed a protocol called MLMA (Multi-Level Multi-Access) to resolve contention in a common transmission medium. The original MLMA was designed to resolve the priority of processors in a decentralized manner, but the algorithm can be applied to the data instead of the processor number. As suggested in [13], the algorithm has been modified for a parallel computer in which the propagation delay is extremely small. The algorithm is analogous to the parallel depth first search of the binary tree representation of messages. This Binary Multi-Level Multi-Access (BMLMA) communication protocol also provides the sorting of messages as a by- product of the protocol. A similar protocol called CAN(Control Area Network) has been proposed for a real-time system[4]

This paper presents a parallel sorting method which can provide superlinear speedup by taking advantage of the communication subnet's sorting capability. The parallel computing model does not make any unrealistic assumptions such as ignoring the context switching time and the communication overhead, In Section 2, We detail the BMLMA communication protocol. In section 3, the parallel sorting algorithm is presented. In section 4, a computer simulation result is presented which confirmed the superlinear speedup of an optimal serial sorting algorithm. Conclusions are drawn in section 5.

2 BMLMA Protocol

The basic principle of the Binary Multi-Level Multi-Access (BMLMA) communication protocol is to organize the

cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bus	0	0	0	1	0	0	1	1	0	1	0	0	0	1	0	1
				*				*				*				*
Contending	1	1	1	2	1	1	1	1	3	3	3	3	3.	3	3	3
Processors	2	2	2		3	3			4	4	4	4				
	3	3			4	4										
	4	4														

Processors	Data Value
1	3(0011)
2	1(0001)
3	5(0101)
4	4(0100)

* The underlined processor indicates the processor which has successfully sent data

figure 1: An example of the BMLMA protocol

processors as an n-ary tree (multi-level) and to assign the access priority accordingly (multi-access). In BMLMA, the processors are contenting for the global channel with the data at the bit level. Thus, before proceeding to a protocol statement, we need the following assumptions.

2.1. Processors are perfectly synchronized with the channel clock (time slot). This can be achieved either with a global clock of in a distributed manner. At the Allied-Signal Aerospace Technology Center, we have successfully built a distributed bit synchronizing circuit operating up to 10 M Hz.

2.2. If all processors sent the same value, i .e., either logical 1 or 0, at the same time slot, the channel stays in the same value. If some processors send logical 1 and the others send 0 at the same time slot, the channel stays in the higher priority value. This can be physically implemented using pull-up resistors and open collector buffers. The null condition would be the high state when the transistor is off, and ground state would be the high state when the transistor is on. In this scheme, if we assign 0 for the low state (ground) and 1 for the high state (null), the value 0 will have high priority.

2.3. A processor can send a bit of data and listen data and listen back to it, overlapping with the values of other processors, within a time slot (one channel lock cycle). This assumption limits the channel speed. In general, the propagation delay must be less than one third of a time slot. For example, if the clock speed is 10 M Hz, the maximum interprocessor distance is 10 m. This is not a major problem for a parallel computer since the processors are closely clustered together in a single cabinet.

2.4. Processors can detect the status (busy or idle) of the channel. This can be effectively done using the *bit stuffing* technique similar to those implemented in packet switch networks [14]

2.5. processors can detect the end of a message. This can be physically implemented either by using a fixed length message or by using a special flag indication the end of a message.

The basin Operation of the BMLMA protocol is as follows. If a processor wants to send a message, it transmits one bit at a time as soon as it senses that the channel is idle. To avoid conflicts, a processor always listens to the channel right after transmitting a bit of data. If it

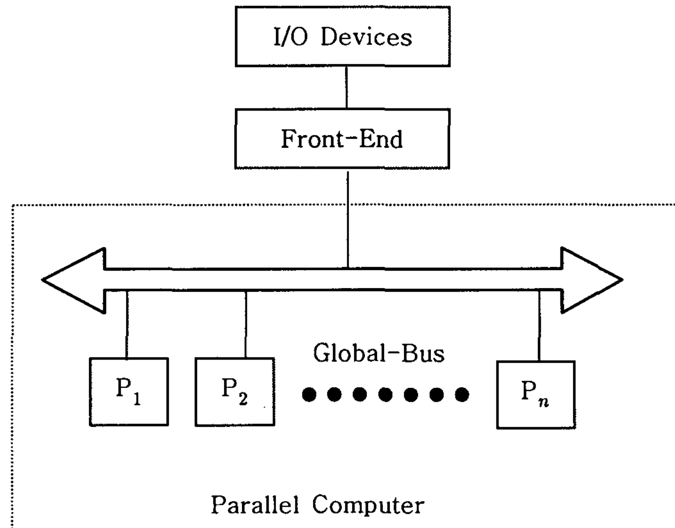

Figure 2: A parallel computer model

detects that its 1 bit has been overwritten with a 0 bit, it aborts its transmission until the end of the winner's transmission. At the end of the winner's transmission, the aborted processor immediately transmits the 4 processors sending the values 1, 3, 4, 5. As shown in Figure 1, the messages are transmitted in an ascending order.

It is easy to show that the algorithm is non-blocking. Since a processor is allowed to attempt to transmit data only when it detects that the channel is idle, there can be two groups of processors ready to send data: those in the current bucket and those in the next bucket. Processors in the current bucket are the processors which originally attempted to transmit data and aborted the transmission. Since there can be only a finite number of processors in the current bucket, they are guaranteed to transmit data in bounded time. Processor which become ready to transmit data during the transmission of the current bucket become the next bucket. At the end of the transmission of the current bucket and *an idle period,* the next bucket become a current bucket.

3 Sorting Algorithm

The definition of speedup, S_n, in evaluation a parallel algorithm is

$$S_n = \frac{time\,(best\ serial\ algorithm)}{time\,(parallel\ algorithm)}. \quad (1)$$

Unfortunately, the best serial algorithm is rarely known. However, the time complexity Of the optimal serial sorting algorithm using only comparison and interchange operations is known to be O $(n \log n)$ where n is the number of elements to be sorted. Thus, sorting is one of a few algorithms which may be used for proving that superlinear speedup is possible.

In general, a parallel computer is used in conjunction with a front-end computer as shown in Figure 2. The parallel computer in this paper is based on a loosely-coupled

parallel computer where multiple processors with their own local memory are inter-

connected by a communication network (global-bus). The front-end computer loads the program and the data into the parallel computer. After the parallel computer completes the processing, the results are loaded back to the front-end computer. Thus, our parallel sorting model assumes that the unsorted data is stored

in the front-end computer and the sorted data (result) will be loaded back into the front-end computer. Then a parallel sorting algorithm may consist of four phases: distribution, local sorting, global sorting, and output.

In the distribution phase, the unsorted data in the front-end computer is divided into n equal size segments. A segment is then assigned to each of the n processors of the parallel computer. Each processor sorts its segment independently of the other processors in the local sorting phase. After the last processor finishes its local sorting,
the n distributed sorted segments are sorted (merged) into a single global order in to the front-end computer. Some of these phases many be combined or overlapped.
For example, our parallel sorting algorithm combined the global sorting phase and the output phase.

Our algorithm is based on divide-and-conquer strategy. The parallel computer consists of n processors and global bus whose the communication protocol is based on BMLMA. However, to achieve global sorting, a processor which has successfully transmitted an element immediately contends for the global bus with its next data element instead of becoming a next bucket as in the original protocol. We call this communication method *block transmission mode*[9]. It is easy to see that the front-end computer will receive globally sorted data because the smallest data element remaining in the processors will always be transmitted to the global bus. Thus, the algorithm combines the global sorting and the output phases. sorting is initiated by the front-end computer but dividing the unsorted data into n equal size data segments and sending them to n processors. Since each processor receives its data segment one at a time from the front-end computer, there is no contention problem for the global bus. Each processor locally sorts its data segment concurrently, immediately after receiving its data segment. Thus, initial data loading and some of the local sorting are pipelined. Each processor reports the completion of its local sorting to the front-end computer. After all processor have completed their local sorting to the front-end computer requests that the locally sorted data segments from all processors be sent to the front-end computer. Since the communication protocol performs global sorting as well as the transmission of data segments from all processors to the front-end computer, the front-end computer will receive globally sorted data

We can summarize the algorithm as follows:

Parallel Sorting Algorithm.

1. The front-end computer divides the m data elements into n equal size data segments (m_i : i = 1 *to* n) Assign m_i to processor i.

2. Processors perform local sorting concurrently as soon as they received the data segment, and inform the front-end computer of the completion of local sorting.

3. After all processors have completed their local sorting, the front-end computer requests the transmission of locally sorted segments. Processors transmit locally sorted segments using the BMLMA block node protocol.

The time complexity of the algorithm can be calculated as follows. Let t_c be the time to transmit an element of data. Since the total number elements of the data is m, the total time to load the data to the parallel computer, T_1, is

$$T_1, = m \cdot t_c.$$

The time required for local sorting for processor i , L_i, is

$$L_i = t_i \cdot \frac{m}{n} \cdot \log(\frac{m}{n})$$

Where t_i is a coefficient which depends on the speed of the processors and the sorting algorithm employed. Even with the same processor and algorithm, t_i varies with the data being sorted. However, for a large volume of data, it is expected that t_i converge to the average value (t_s). Since global sorting cannot begin until all processors have completed local sorting, the time required for local sorting, T_2, is

$$T_2 = MAX(t_i : i = 1 \quad to \quad n),$$

or for a large volume of data,

$$T_2 = t_s \cdot \frac{m}{n} \cdot \log\left(\frac{m}{n}\right).$$

where t_s is a constant dependent upon the speed of the processor and the sorting algorithm employed.

The time required for the output procedure, T_3, is identical to that of the initial loading(T_1). Thus, the total time required for parallel sorting, T_p, with n processors is

$$T_p = T_1 + T_2 + T_3 = 2 \cdot m \cdot t_c + t_s \cdot \frac{m}{n} \cdot \log\left(\frac{m}{n}\right). \quad (2)$$

T_1 and T_3 are serial parts of the algorithm which cannot be parallelized. The time required for a serial sorting algorithm using a single processor, T_s, is

$$T_s = t_s \cdot m \cdot \log(m) \quad (3)$$

Using Equation (1), the speedup of the parallel sorting algorithm is

$$S_n = \frac{T_s}{T_p}.$$

Form Equation (2) and (3), we have

$$S_n = n \cdot \left(\frac{t_s \cdot \log(m)}{t_s \cdot \log(m) + (2 \cdot n \cdot t_c - t_s \cdot \log(n)}\right). \quad (4)$$

S_n becomes greater than n when

$$2 \cdot n \cdot t_c - t_s \cdot \log(n) < 0.$$

Or

$$\frac{t_c}{t_s} < \frac{\log \sqrt{n}}{2n}. \quad (5)$$

For a 16 processor system, we can achieve superlinear speedup when $\frac{t_c}{t_s} < \frac{1}{16}$. For a 1000 processor system, the relationship is $\frac{t_c}{t_s} < \frac{1}{400}$.

In a practical system, $t_c$is much smaller than t_s.For example, for a typical system operating at 100 MHz channel speed with an element consisting of a 32 bit key field and a 32 bit index field, t_c is 0.64 μs, regardless of the size of data. The simulation results showed that t_s of a computer used is 8.1 μs using an internal sorting algorithm,

Quicksort. Thus, it is possible to achieve superlinear speedup of a serial sorting algorithm for up to 32 processors even with an optimal internal sorting algorithm with this processor. However, t_s is expected to be substantially higher for external sorting algorithms due to slow I/O operations.

4 Results

The parallel sorting algorithm has been simulated. The simulation assumes the channel speed is 100 MHz and the element of data consists of a 32 bit key field and a 32 bit index field. An internal sorting algorithm, Quicksort, has been written. The program first creates an internal data table by generating 32 bit random numbers and them sorts them using the Quicksort algorithm in [11]. The program only calculates the CPU time required to sort the table. The elapsed time may be the actual sorting time observed by the user and is substantially

No of items	CPU time (seconds)	t_s (μ seconds)
10,000	0.075	7.5
100,000	0.8	7.6
200,000	2.15	7.7
500,000	7.3	7.8
1,000,000	13.5	8.0

Table 1: Simulation results of t_s.

No of processor	total sorting time (seconds)	speedup
1	13.54	1
2	6.11	2.21
5	2.3	6.23
10	0.99	13.64
50	0.17	82.26
100	0.08	177.45

Table 2: The speedup of the parallel sorting algorithm

larger than the CPU time; thus, the elapsed time can provide better speedup than the CPU time (see Equation (5)). However, the CPU time is used to calculate the speedup of the parallel sorting algorithm in this simulation because the elapsed time depends too much on the configuration of a simulation computer such as memory size.

We have calculated t_s from the simulation results by assuming that the sorting time is $t_s \cdot n \cdot \log(n)$. As shown in Table 1, t_s is relatively constant (7.5 to 8.0 μs) for the wide range of data sizes. However, t_s still increases with the size of the data set. In other words, the average computing time for Quicksort is very close to $O(n \log n)$,
but it is a little worse than $O(n \log n)$. although this may be insignificant for analyzing the sorting algorithm theoretically, the simulation showed that its effect for a practical sorting algorithm is very significant. For example, using Equation (5) and $t_s = 8.0 \mu s$, it is possible to achieve superlinear speedup only up t 32 processors. However, simulation results showed that it is possible to achieve superlinear speedup with more than 100 processors as shown in Table 2. We believe that this is the first known simulation result of superlinear speedup of any algorithm. There is a sorting algorithm which can *theoretically* achieve linear speedup for a small number of processors. Baudet and Stevenson proposed a parallel sorting based on the generalized odd-even transposition. [15] They have shown that theoretically, the asymptotic speed-up ratio of the parallel algorithm over the optimal sequential

algorithm is the number of processors when the number of processors is smaller in order of magnitude than log(*number of elements*). However, the speed-up ratio decreases rapidly with the increase in the number of processors.

5 Conclusions

We have presented a parallel sorting algorithm which takes advantage of the sorting capability of the BMLMA communication protocol. The sorting algorithm is based on the divide-and-conquer strategy. However, locally sorted segments ore merged together using the global bus.

A parallel computer is a set of n processors and a communication subnet; thus, it is possible to achieve superlinear speedup of an optimal serial algorithm by taking advantage of the processing capability of the communication subnet. We have shown that the new parallel sorting algorithm can achieve superlinear speedup. Computer simulation has confirmed the analytical results.

For brevity, we have presented only the BMLMA communication protocol but there is another protocol called CITO [12] which provides not only sorting of data but also data compression; thus, it has the potential to provide even better speedup.

References

[1] Knuth, D. E., "The Art of Computer Programming: Sorting and Searching" , Vol 3, 2nd ed, 1998, chap. 5. Addison-Wesley.

[2] E. E. Lmd, "introduction - Sorting", IEEE Trans. Comput., VOl. C-31, No. 4, Apr. 1985, pp. 293-295

[3] Yijie Han, "Deterministic sorting in O(nlog log n) time and linear space", Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, May 19-21, 2002, Montreal, Quebec, Canada

[4] Thomas Nolte, etc, "Using bit-stuffing distributed in CAN analysis", IEEE Real-Time Embedded Systems, Dec 3, 2001

[5] D. Rotem, N. santoro, and J. B. Sidney, "Distributed Sorting", IEEE Trans. Comput., Vol. C-34, NO. 4, Apr. 1985, pp. 372-376

[6] S. Zaks, "Optimal Distributed Algorithms for Sorting and Ranking", IEEE Trans. Comput., Vol. C-34, NO. 4, Apr. 1985, pp. 376-383

[7] Krishna P. Mikkilinent and Stanley Y.W. Su, "An Evaluation of Sorting Algorithms for Common-Bus Local Networks", J. Parallel and Distributed Computing 5 1988, pp59-81

[8] D. Parkinson, "Parallel efficiency can be greater than unity", Parallel Computer. 3, 1986, pp261-261

[9] You-Keun Park, "The CITO Block Transmitter", Allied-signal ATC, Invention Disclosure 450-87-011, 1987

[10] V. Faber, O.M. Lubeck and A.B. white,Jr., "Superlinear speedup of an efficient sequential algorithm is not possible", Parallel Comput. 3, 1986, pp259-260

[11] Ellis Horowitz and Sartaj Sahni, "Fundamentals of Data structures", Computer Science Perss, 1982

[12] Simon Y. Berkovich and Colleen Roe Wilson, "A Computer Communication Technique Using content-Induce Transaction Overlap", ACM Trans. On Comput., Vol. 2, No. 1, February 1984, pp. 60-77.

[13] E. H. Rothauser and D. Wild, "MLMA - A Collision-Free Multi-Access Method", Proc. to 1977 IFIP Congress, 1977, pp. 431-436

[14] S. Berkovich, et all, "Distributed Associative Processor" submitted to 1989 ACM Computer Science Conference, Louisville, kentucky, February, 1989

[15] G. Baudet and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers", IEEE Trans. On Comput., Vol. C-27, No. 1, jan. 1978, pp. 84-87