An Efficient Implementation of PSRS algorithm on AP1000

Nai-jie GU [+], Guo-liang CHEN [+], Hidehiko TANAKA [++]

[+] Department of computer science and technology
University of science and technology of china, Hefei, Anhui, 230026
E-mail: njgu@cs.ustc.edu.cn, glchen@cs.ustc.edu.cn

[++] University of Tokyo, Department of electrical engineering
7-3-1 Hongo, Tokyo, Japan, 113
E-mail: tanaka@mtl.t.u-tokyo.ac.jp

Abstract: In this paper we will discuss the implementation details of PSRS[1] algorithm on AP1000 parallel computer; give out an all-to-all communication scheme, whose time complexity reaches the lower bound for all-to-all operation on 2-D torus; introduce a multi-way merge algorithm by means of minimum-heap; and present some experiment results to show the difference between various implementation strategies, the good load balance and the small communication overhead of PSRS algorithm.

Key Words: Parallel sorting, scalability, all-to-all communication,.

1. Introduction

With the advent of parallel processing, parallel sorting has become an important area for algorithm research. Many papers have discussed the task of sorting on parallel computers[1-3]. PSRS[1](parallel sorting by regular sample) is an asymptotically optimal (when $n \geq p^3$) sorting algorithm suitable for MIMD multiprocessor system. We have implemented it on a AP1000 parallel computer of 64 processors using various implementation strategies. In this paper we will describe a theoretical optimal all-to-all communication scheme for torus, and give an improved multi-way merge algorithm, discuss the implementation details and the performance of PSRS algorithm on AP1000. Our experiment result shows that PSRS algorithm has a good loading balance, very small communication overhead, and good scalability.

We will describe the architecture of AP1000 in section 2, introduce the general aspect about the PSRS in section 3, discuss the implementation details in section 4, and analyse the performance and give the experiment results in section 5.

2. AP1000 Parallel Computer

AP1000 parallel machine is a distributed memory MIMD system with 64 independent SPARC processors that are called cells. Each cell has a 128 Kbyte direct-mapped copyback cache, 16 Mbyte of memory, and a Weitek floating-point unit (FPU) of theoretical peak speed 8.33 MFLOP (single precision) and 5.67 MFLOP (double precision). The topology of the AP1000 is a torus, with hardware support for wormhole routing. The cells are connected by three independent networks. The T-net provides a theoretical bandwidth of 25 Mbyte/sec between any two cells; in practice, about 6 Mbyte/sec is attainable by user programs. The B-net supports row and column broadcasts, so a broadcast usually takes about the same time as a comparable cell to cell transfer. The S-net supports status checks and barrier synchronizations, and the delay of the synchronization is 1.6 microseconds.

3. PSRS Algorithm

Let X be the data set to be sorted on the p-processor MIMD multiprocessor, and n is the size of X. For simplicity in the design of algorithm, assume $p^2 | n$, and $x_i \neq x_j$(when $i \neq j$). In start, all elements are uniformly distributed on p processors, every processor has $u = n/p$ elements. Denote $PE_0, PE_1, PE_2, \cdots, PE_{p-1}$ as the p processors. The PSRS algorithm can be described as follows[1]:

1) Every processor sorts its own $u = n/p$ elements using the optimal sequential sorting algorithm, and gets a sorted list.
2) Every processor evenly spaced selects $p-1$ sample elements from its sorted list, and sends the $p-1$ sample elements to processor PE_0.
3) After receiving $p(p-1)$ elements, PE_0 merges these p sorted segments, and then selects $y_1, y_2, \cdots, y_{p-1}$ as $p-1$ pivots that y_j locate at the position of $j \cdot (p-1)$ (where $j = 1,2,\cdots, p-1$).
4) PE_0 broadcasts the chosen $p-1$ pivots $y_1, y_2, \cdots, y_{p-1}$ to all other $p-1$ processors.

5) After receiving $p-1$ pivots, every processor merges these pivots with its own sorted list. These pivots partition the list into p sublists. Denote w_i^j as the $(j+1)$th sublist of processor PE_i, where $i=0,1,\cdots,p-1, j=0,1,\cdots,p-1$.
6) Every processor sends its $(j+1)$th sublist $(j=0,1,2,\cdots,p-1)$ to processor PE_j.
7) After receiving all the $(j+1)$th sublists of p lists, processor PE_j performs a p-way merge, sorts these p sublists. After this, the original X is sorted.

If $n \geq p^3$, the complexity of PSRS is $O((n/p)\cdot\log n)$ which is optimal. In the later of this paper we assume $n \geq p^3$ is satisfied. The following results are proved in [1]:

$$\sum_{j=0}^{p-1}\left|w_i^j\right| = u, \quad \sum_{i=0}^{p-1}\left|w_i^j\right| < 2u \qquad (1)$$

That means: initially the number of elements stored in any processor is u, and after sorting the number of elements stored in any processor is less than $2u$.

4. Implementation of PSRS algorithm

Now we discuss the implementation details of above algorithm.

A. Serial sorting

The aim of the serial sorting is to order the elements in each cell in minimum time. For this task, we use the quicksort. The main reason is that, quicksort has a good performance on average, and is easy to program. We also make some changes to improve the performance of quicksort algorithm, we call the new algorithm as sample quicksort.

In sample quicksort, we first evenly spaced select some elements from the list, sorting these elements using quicksort, and then using these ordered elements to split the initial list into several sublists. Continue doing in this way, until the sublist is small enough, then using quicksort.

Table 1: Time used by two sorting algorithms.

	Number of data per-processor				
	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
Quicksort	1.08152	2.36878	5.25889	10.9853	22.8830
S- quicksort	1.06164	2.27849	4.83238	10.4968	21.8194

Our experiment result (Table 1) shows that for large size list or large number of processors, sample quicksort is quicker than original quicksort.

B. Data broadcasting and Data collection

We use the B-net for data broadcast. The delay of data broadcast is very small compare to other communication operations.

Though the time used for data collection is small enough, we have made some efforts to try to improve the data collection scheme. One method is to use multi-level scheme, another method is to use the binary tree method. Since the AP1000 has a very small communication delay, and the number of data collected from each cell is not large enough, above two methods can not improve the behavior of data collection, so we just let cell sends the data to a specified cell directly.

Table 2: Time used for broadcast and collection

P	4	8	16	32	64
broadcast	0.000686	0.000710	0.000718	0.000748	0.000869
collection	0.000950	0.001500	0.002726	0.006079	0.015174

C. Split the ordered list using pivots

To split the ordered list into sublists using $p-1$ pivots, we chose the binary search method. Since the time complexity for merge two ordered lists with length of u and p is $\Omega(u+p)$, the time complexity for p times binary search is $O(p\cdot\log u)$, when $n \geq p^3$, $O(p\cdot\log u) < \Omega(u+p)$. So the behavior of binary search method is better than the behavior of merge method.

Table 3: Time used for split ordered list.

	Number of data per-processor				
	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
$p=4$	0.00012	0.00014	0.00016	0.00016	0.00017
$p=8$	0.00021	0.00028	0.00030	0.00033	0.00034
$p=16$	0.00052	0.00058	0.00068	0.00074	0.00078
$p=32$	0.00091	0.00113	0.00128	0.00138	0.00146
$p=64$	0.00177	0.00220	0.00245	0.00261	0.00281

D. All-to-all communication

We have implemented several all-to-all communication schemes, the following scheme is the quickest. We will analyse the complexity of this communication scheme in the next section. Where *Ncelx* and *Ncely* are the number of cells in X direction and Y direction respectively, we assume *Ncelx*=*Ncely*=$\sqrt{p}$; x and y is the index of the cell in X and Y directions.

All-to-all communication scheme:
For (i=0; i<Ncelx; i++)
For (j=0; j<Ncely; j++)
PE(x,y) sends its (j · Ncelx + i)th sub-list to
PE((x-i)%Ncelx, (y-j)%Ncely);
PE(x,y) receives message from
PE((x+i)%Ncelx, (y+j)%Ncely);
Endfor
Endfor

Table 4: Time used for all-to-all communication

	Number of data per-processor				
	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
$p=4$	0.07374	0.14600	0.28894	0.57930	1.15746
$p=8$	0.07505	0.14667	0.28693	0.56781	1.13472
$p=16$	0.08017	0.15096	0.29176	0.56996	1.12208
$p=32$	0.09084	0.16613	0.31366	0.60587	1.19223
$p=64$	0.11220	0.18990	0.34318	0.64698	1.25527

D. p-way merge

We have implemented two p-way merge algorithms. The first, we perform the p-way merge by means of 2-way merge. The second method is using the p element minimum heap to perform the p-way merge, every time remove the smallest element from the heap, and add a new element into the heap. These two methods have the same theoretical time complexity, both complexities are $O(\log p \cdot u)$. But, the experiment result shows that there are some difference. For large number of cells or large number of elements, the second method is better than the first one.

Table 5: Time used by two merge algorithms.

		Number of data per processor		
		2^{15}	2^{17}	2^{19}
$p=8$	2-way	0.569336	2.365860	9.485399
	heap	0.595140	2.370780	9.455994
$p=16$	2-way	0.769993	3.156132	12.69423
	heap	0.769637	3.027281	11.99837
$p=32$	2-way	0.988803	3.953025	15.92748
	heap	0.975425	3.785856	14.89588
$p=64$	2-way	1.218697	4.881176	19.32312
	heap	1.207572	4.722405	18.28858

5. Time complexity

The theoretical complexity of PSRS algorithm includes four parts: serial sorting, split the ordered list, merging, and communication. Now we analyse these four partial complexities.

A. Serial sorting

Since initially every cell has n/p data, in every case the time used for sorting n/p elements by quicksort is $O(n.p \cdot \log(n/p))$.

B. Split the ordered list

We use $p-1$ pivots to split the ordered array into p sublists by means of binary search. It is well known that the time complexity for binary search in an n/p elements list is $\log(n/p)$. So the time used for split the array is $(p-1)\cdot\log(n/p)$.

C. Merging

We perform the p-way merge by means of p elements minimum heap. The number of elements in each cell (when the total sorting is finished) is no more than $2n/p$ (equation 1). So the time used for p-way merge is $\Theta(\log p \cdot n/p)$.

D. Communication

There are three communication steps: broadcast, collection, and all-to-all communication. Since there is hardware support for data broadcast on AP1000, the time used for broadcast $p-1$ elements is very small, it can be omit.

While doing data collection, every cell sends $p-1$ elements to a specified cell, the time used for this operation is $O(p^2)$.

Since AP1000 support wormhole routing. From our all-to-all communication scheme, we know that in any outer loop step, all the messages traveled along any one links line (all the links of same processors line) come from at most two different processor lines, using the same method[1] used for equation (1), we can prove that the total length of the messages traveled along any links line in one outer loop step is limited by $2\cdot 2n/p$. Since AP1000 is a 2-D torus, the time used for convert these messages along links line in any one outer loop step is limited by $\sqrt{p}\cdot t_s + 2n/p\cdot t_w$, the time used for convert messages along column links in any one outer loop step is also limited by $\sqrt{p}\cdot t_s + 2n/p\cdot t_w$. The time used for all-to-all communication is $O(p\cdot t_s + \sqrt{p}\cdot 2n/p\cdot t_w)$.

Now let us consider the general all-to-all communication problem, i.e., each processor sends an L words message to all the other processors, the length of every message is the same, but to different

destinations, the messages are different.

The sum of the length of the path, that each message comes from one source processor will traveled on AP1000, is $2\cdot\sqrt{p}\cdot p/4$. The total distance all the messages traveled is $p\cdot p^{3/2}/2$. Since there are $2p$ links in a 2-D torus, at least: $\Omega(p\cdot t_s + L\cdot p^{3/2}\cdot t_w)$ time is needed for all-to-all communication. If the message length L is n/p^2, the lower bound is $\Omega(p\cdot t_s + n/p\cdot p^{1/2}\cdot t_w)$.

In PSRS algorithm, the message length of all-to-all communication is not identity, the average length of the messages is n/p^2. So the lower bound of the time used for the all-to-all communication in PSRS algorithm on AP1000 is at least $\Omega(p\cdot t_s + n/p\cdot p^{1/2}\cdot t_w)$.

Above analysis shows that our all-to-all communication scheme is optimal, its time complexity reaches the lower bound within a constant.

6. The scalability of PSRS algorithm

The scalability of parallel algorithm is defined as its capability of effectively utilizing an increasing number of processors. In this section, we using the iso-efficiency analysis method[3] analyse the scalability of PSRS algorithm on AP1000.

From above section we know that the major item of the communication complexity is $O(\sqrt{p}\cdot n/p)$. So the overhead for communication is $O(n\cdot\sqrt{p})$. The serial component is caused by step (3) of PSRS. The complexity of this step is $O(p^2\cdot\log p)$. The major item of overhead for serial component, synchronization, idle e.t., is $O(p^3\cdot\log p)$. When $n\geq p^3$, the major item of total overhead is $O(n\cdot\sqrt{p})$. The computation size for sorting is $O(n\cdot\log n)$. The iso-efficiency function is $\Theta(\sqrt{p}\cdot 2^{\sqrt{p}})$ [2].

7. Experiment results

In section 4, we discussed the implementation strategy from the view of algorithm. Table 6 shows the experiment result of PSRS algorithm on AP1000 before optimizations. We try to use some optimization methods to improve the performance. It was found that these optimizations played an important role in the speed of the program, improved the overall speed more than 50%. Table 7 shows the improved result. Table 8 shows the largest partial times for sorting 32M 32-bites integers on AP1000 with 64 processors. It should be noted that the sum of these largest partial times will greater than the largest total time. .

Table 6: Time used by PSRS before optimization.

	Number of data per-processor				
	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
$p=4$	1.57355	3.40536	7.12131	14.7219	30.3936
$p=8$	1.73758	3.66925	7.98234	15.9110	32.5513
$p=16$	1.91958	4.00256	8.16223	17.1106	34.9210
$p=32$	2.17532	4.44660	8.94646	18.6202	37.9426
$p=64$	2.52960	5.03218	10.0292	20.5383	41.4944

Table 7:Time used by PSRS after optimization.

	Number of data per-processor				
	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
$p=4$	0.70086	1.56833	3.31336	6.95760	14.4226
$p=8$	0.74827	1.66000	3.56987	7.38421	15.1942
$p=16$	0.81790	1.77072	3.76843	7.76780	16.1847
$p=32$	0.90274	1.89268	3.99784	8.19154	17.0079
$p=64$	1.03621	2.10554	4.42186	8.92742	18.2019

Table 8: (Time unit: second)
Largest partial times for sort 32M data, $p=64$.

Serial sorting	12.399955
Merge	4.470601
Communication	1.266779
Idle, *etc.*	1.549202
Total time	18.201884

Acknowledgments

We are grateful to Mr. Ichiro IDE and all the others in professor Tanaka's lab for supporting our research.

References:

[1] Shi Hanmao, Schaeffer J., Parallel Sorting by Regular Sampling, Parallel and Distributed comp., 14(4):361.

[2] Gu Naijie, Chen Guoliang, Zhang qun, The scalability of PSRS algorithm on Mesh, Science in China(A), supp.1995, 38(9):44.

[3] Singh V., Kumar V., Agha G., *et al.*, Efficient algorithms for parallel sorting on mesh multicomputers, J. Parallel Programming,1991, 20(2):95.