Efficient and Scalable Parallel Algorithm for Sorting Multisets on Multi-core Systems

Cheng Zhong, Zeng-Yan Qu, Feng Yang, Meng-Xiao Yin, Xia Li
School of Computer and Electronics and Information, Guangxi University, Nanning, China
E-mail: chzhong@gxu.edu.cn, quzengyan@163.com, yf@gxu.edu.cn, ymx@gxu.edu.cn, lixiaxia@163.com

***Abstract*—By distributing adaptively the data blocks to the processing cores to balance their computation loads and applying the strategy of "the extremum of the extremums" to select the data with the same keys, a cache-efficient and thread-level parallel algorithm** for s**orting Multisets on the multi-core computers is proposed. For the sorting Multisets problem, an aperiodic multi-round data distribution model is presented, which the first round scheduling assigns data blocks into the slave multi-core nodes according to the given distribution order and the other rounds scheduling will distribute data blocks into the slave multi-core nodes by first request first distribution strategy. The scheduling technique can ensure that each slave node can receive the next required data block before it finishes sorting the current data block in its own main memory. A hybrid thread-level and process-level parallel algorithm for sorting Multisets is presented on the heterogeneous cluster with multi-core nodes which have different amount of processing cores, different computation and communication capabilities and distinct size of main memory. The experimental results on the single multi-core computer and the heterogeneous cluster with multi-core computers show that the presented parallel sorting Multisets algorithms are efficient and they obtain good speedup and scalability.**

***Index Terms*— Multisets Sorting, Parallel algorithms, Multi-core computers, Heterogeneous clusters, Multi- level cache, Shared L2 cache, Thread-level parallelism, Selection, Aperiodic multi-round distribution**

I. INTRODUCTION

Some classical parallel sorting algorithms on various computing models were introduced in [1]. Recently, many researchers pay more and more attention to parallel sorting algorithms with GPU processors and multi- core processors. Purcell et al [2] developed a bitonic parallel sorting algorithm by the advantages of high computation-intensive and high bandwidth for GPU processors. Algorithm GPUTerasort [3] executes sort operations by SIMD instructions to obtain data parallelism, and it can lower memory latency and improve cache access efficiency by overlapped pointer and fragment processor memory accesses. Greb and Zachmann [4] applied a bitonic tree to rearrange data to reduce the number of comparing data, and presented a GPU-Abisort algorithm. Sintorn and Assarsson [5] proposed a hybrid parallel GPU-Sorting algorithm, which achieves higher speedup.

Cell machine is a kind of processors adopting heterogeneous multi-core architecture frame, it has a master processing core Power PC and eight SIMD processing cores. Gedik et al [6] proposed a so-called Cellsort three-level parallel merge sort algorithm on the Cell BE machine, the first level of which is an optimized SIMD bitonic sort algorithm, the second level is an optimized in-core bitonic merge algorithm and the third level is responsible of sorting a large amount of the data. Since algorithm Cellsort wanted to access the data in main memory several times, its obtained speedup is small. Keller and Kessler [7] improved algorithm Cellsort and presented a method to merge the data stored on the continued level nodes of the merge tree such that the merged sequences were not written to the main memory, but they were directly outputted to SIMD processing elements for next merge. Ramprasad and Baruah [8] designed an optimized radix sort algorithm which can ensure the load balance among the processing cores on the Cell Broadband Engine. Sharma et al [9] implemented six parallel algorithms for sorting integers on Cell processors and analyzed their speedups. Inoue et al [10] designed a so-called AASort parallel sorting algorithm on the multi-core SIMD processors, which can utilize fully SIMD instructions and multi-core thread-level parallelism, and eliminate the nonalignment memory access. Because algorithm AASort must access main memory several times in the merge sorting stage, it obtained small speedup. Chhugani and Macy [11] implemented efficiently a sorting algorithm on multi- core SIMD CPU architecture, which can reduce cache access latency by multi-core characteristics, increase computational density by vectored SIMD technique, balance the loads among multi-core processors in use of data decomposition, and eliminate the restriction of bandwidth by using multi-way merge. Li and Zhong et al [12] presented a multi-round scheduling technique for divisible loads on the heterogeneous cluster systems with multi-core computers. Zhong and Qu et al [13] proposed an aperiodic multi-round data distribution strategy and a parallel Multisets sorting algorithm on the heterogeneous clusters with multi-core computers.

According to the property of input Multisets sequence

Manuscript received December 26, 2010; revised February 20, 2011; accepted March 28, 2011.
Corresponding author: chzhong@gxu.edu.cn

doi:10.4304/jcp.7.1.30-41

and multi-core architectures, this paper extends our work [13] to study further the Multisets sorting problem on the multi-core systems, propose the efficient and scalable parallel sorting Multisets techniques on single multi-core computer and the heterogeneous cluster systems with multi-core computers respectively. The remainder of this paper is organized as follows. By considering impact of multi-level caches, different number of processing cores and parallel threads, section 2 first presents a cache-efficient and thread-level parallel algorithm for sorting Multisets on single multi-core computer, and then proposes a novel aperiodic multi-round data distribution model by applying the divisible loads principle [14] and design a hybrid process-level and thread-level parallel sorting Multisets algorithm on the heterogeneous clusters with multi-core nodes. Section 3 evaluates the execution time, speedup and scalability of the presented algorithms on single multi-core computer and the heterogeneous clusters with multi-core computers, respectively. Section 4 concludes the paperand gives further research direction.

II. PARALLEL SORTING MULTISETS ON MULTI-CORE SYSTEMS

Multisets is a special input data sequence and Sorting Multisets [15,16] is one kind of the data sorting issues.

Definition 1 Multisets is a special data sequence of size n, which includes only k distinct keys, $0<k<<n$.
For example, let *nl_s* be a set which is consists of people's age, *nl_s* is one kind of the dataset Multisets.

Definition 2 Sorting Multisets is to sort the data sequence for Multisets.

A. Cache-Efficient Thread-Level Parallel Sorting Multisets on Single Multi-core Computer

Definition 3 The so-called extremums are defined as the maximum and minimum in a maximal-minimal element sequence which is composed of all the maximal and minimal elements in the given several data sequences.

By applying the two-tiered mode, we design a cache-efficient and thread-level parallel sorting Multisets algorithm, short for algorithm CETLPS- Multisets, on single multi-core computer. The first tier of algorithm CETLPS-Multisets is a local parallel sorting on the shared L2 cache, it sorts the data subsequence whose size is proportional to the usage size of the shared L2 cache. The second tier is an efficient thread-level parallel algorithm to merge the sorted subsequences in order to achieve better communication performance among the processing cores. To reduce the time to exchange data between main memory and the shared L2 cache on the multi-core computer and decrease the time to merge the sorted subsequences, algorithm CETLPS-Multisets selects the data with the same keys by picking up recursively the extremums in the subsequences and assigns dynamically these data with the same keys to the processing cores to balance their computing loads, and it also applies the data blocking technique to minimize the time to access the main memory.

We assume that the Multisets sequence to be sorted has n data $X[1\sim n]$,the key of $X[i]$ is $X[i].key$, $i=1\sim n$.The distinct k keys in $X[1\sim n]$ are represented as K_1, K_2, …, and K_k respectively, where $0<k<<n$. The number of the data with key K_r is $se[K_r]$, where $0<se[K_r]\le n$, $r=1\sim k$, and $\sum_{r=1}^{k} se[K_r] = n$. Suppose that the multi-core computer has p processing cores, and P_j is the j-th processing core, $j=1\sim p$; the usage space of shared L2 cache can store C data, and a data block in the shared L2 cache contains M elements, where $M < C$. The input sequence for Multisets with length n is partitioned into n/M data blocks.

The memory bandwidth and cache design can influence significantly the performance of multi-core parallel algorithms. Therefore, we must minimize the time to access the main memory to design an cache-efficient and thread-level parallel sorting Multisets algorithm on multi-core computer. We divide the input data sequence for Multisets into several data blocks, which each block contains M elements. In the local sorting stage, we sort each block by the strategy of "the extremum of the extremums" and "selection", find the number of the data whose keys are equal to the maximal and minimal keys in each block respectively, and obtain the values of corresponding to the maximal and minimal keys. We merge in parallel the data with same keys by multiple threads to reduce the size of the data to be merged in the main memory and the number of exchanging data between the main memory and shared L2 cache in the merge stage.

The cache-efficient and thread-level parallel sorting Multisets algorithm on single multi-core computer, short for CETLPS-Multisets, is described as follows.

Algorithm CETLPS-Multisets
Begin
(1) $i=1$.
(2) The i-th data block with M elements from the main memory is read into the shared L2 cache. The data block in shared L2 cache are divided into p subsequences and they are distributed to p processing cores, which each processing core receives one subsequence. Each processing core sorts its subsequence and then writes the sorted subsequence to its local L1 cache.
(3) Processing core P_j executes the optimal selection algorithm using multiple threads to find the maximum max_j and minimum min_j from the keys of the sorted subsequence in its local L1 cache, and P_j writes max_j and min_j to its local L1 cache, $j=1\sim p$.
(4) One processing core executes the optimal selection algorithm using multiple threads to find maximum MAX from $\{max_1,max_2,\ldots, max_p\}$ and minimum MIN from $\{min_1,min_2,\ldots,min_p\}$, and then writes MAX and MIN to the shared L2 cache.
(5) Processing core P_j partitions the sorted subsequence in its local L1 cache into three parts $XL[j]$, $XM[j]$ and $XE[j]$,where the key of each element in $XL[j]$ is equal to MIN, the key of each element in $XM[j]$ is equal to MAX, and the key of each element in $XE[j]$ is not equal to MIN and not equal to MAX, $j=1\sim p$. MIN is

represented by *v_min*[*i*], the number of the data with key *MIN* in all the subsequences on *p* processing cores is denoted by *c_min*[*i*]. Similarly, *MAX* is represented by *v_max*[*i*], the number of the data with key *MAX* in all the subsequences on *p* processing cores is denoted by *c_max*[*i*]. *v_min*[*i*], *c_min*[*i*],*v_max*[*i*] and *c_max*[*i*] are written to the shared L2 cache, *j*=1~*p*.

(6) *p* processing cores combine *XE*[1], *XE*[2],…, and *XE*[*p*] in their local L1 caches into a sequence *XE*, and *XE* is stored on the shared L2 cache.

(7) *i*=*i*+1. if $i \leqslant n/M$ then goto step (2).

// The following is to merge data in shared L2 cache

(8) for *i*=1 to $\lceil k/2 \rceil n/M$ do

do steps (8.1),(8.2),(8.3) and (8.4) in parallel

(8.1) *v*[*i*]←*v_min*[*i*];

(8.2) *c*[*i*]←*c_min*[*i*];

(8.3) $v[2n\lceil k/2 \rceil / M - i + 1] \leftarrow v\_max[i]$;

(8.4) $c[2n\lceil k/2 \rceil / M - i + 1] \leftarrow c\_max[i]$;

endfor

// The following is to merge data in the main memory

(9) for *i*=1 to $2n\lceil k/2 \rceil / M$ do

for *j*=1 to *c*[*i*] do

if *i*=1 then *s*←0 else *s*←*s*+*c*[*i*-1];

X[*s*+*j*].*key*←*v*[*i*];

endfor

endfor

End.

For each iteratiive execution in the local parallel sorting stage, only one block is read into the shared L2 cache, the blocks are not overlapped each other. Therefore, any one element for Multisets is read from and written to the main memory only one time. On the other hand, we can obtaine the data with same keys in a data block and their amount in the local paralle sorting stage. In the parallel merge stage, according to the obtained data with same keys and their amount, the size of the data to be merged can be greatly decreased and the time to exchange data between the main memory and the shared L2 cache can be remarkably reduced. Hence, we can merge efficiently the data by parallel miltiple threads. In other words, CETLPS-Multisets is a cache-efficient thread-level parallel algorithm for sorting Multisets.

B. Fast and Scalable Parallel Sorting Multisets on Heterogeneous Clusters with Multi-core Nodes

We suppose that the heterogeneous multi-core cluster consists of one master node with multi-core processor and *d* slave nodes with multi-core processors, and its logical topology is star structure. Master node D_0 is responsible for receiving *n* input data for Multisets including only *k* distinct keys and distributing these data to *d* slave nodes, *k*<<*n*. Master node D_0 communicates with other slave nodes in serial transfer mode. In other words, master multi-core node is allowed to send data to only one slave multi-core node at the same moment, and only one slave node is permitted to return the sorted subsequence to the master node at the same time. We also assume that both of master node and slave nodes can execute simultaneously computation and communication. That is to say, master node can merge its received sorted subsequences while it sends data block to slave nodes or receives the sorted subsequence from slave nodes, and each slave node can sort its received data while it receives the data from master node or returns the sorted subsequence to master node.

For convenience, let L_i denote a communication link from master node D_0 to slave node D_i, $nLat_i$ represent the required time to start one communication in link L_i. We suppose that the wanted time to transfer a datum is $1/B_i$ and the required time to transfer N_i data is $nLat_i+N_i/B_i$ in link L_i, where B_i denotes the transfer rate and $cLat_i$ denotes the required time that slave node D_i starts a computation, the required average time that slave node D_i sorts a datum is $1/S$ and the required time that slave node D_i sorts N_i data is $cLat_i+N_i/S_i$, where S_i represents the computation rate, *i*=1~*d*. *BUF* represents the total capacity of the main memories on *d* slave nodes, size *n* of the input data for Multisets is not greater than *BUF*, and size N_i of the data distributed to slave node D_i is not greater than capacity buf_i of the main memory on slave node D_i, *i*=1~*d*.

Let C_{max} be the execution time to sort completely the input sequence for Multisets of size *n* on the heterogeneous clusters with multi-core nodes.

B.1. Single-round Data Distribution Model and Parallel Sorting for Multisets

Figure 1 describes the procedure that master node distributes data blocks for Multisets to the slave nodes using single-round distribution mode and the slave nodes sort their received data blocks in parallel on the heterogeneous clusters with multi-core nodes.

From Figure 1 we can see that for the large-scale input sequence for Multisets, if master node distributes data blocks to slave nodes in use of single-round data distribution mode, there will be two limitations. The first one is that the size of the data distributed to the salve node may be more than its storage capability. The second one is that the amount of data scheduled to the salve nodes must be increasing each time, and the computation and communication execution can not beimplemented in parallel. In other words, the required communication time will be increased. Therefore, we must study to present a more efficient data distribution strategy to sort in parallel the input Multisets sequence.

B.2. Aperiodic Multi-round Data Distribution Model and Parallel Sorting for Multisets

To design an efficient and scalable parallel algorithm for sorting Multisets on the heterogeneous clusters with multi-core machines, the issues to be solved are to how to partition properly the input data into several data blocks, how to ensure that each salve node can carry out sufficiently its computational capability and master node can communicate efficiently with slave nodes, and how to insure that each multi-core node can overlap computation and communication operation.

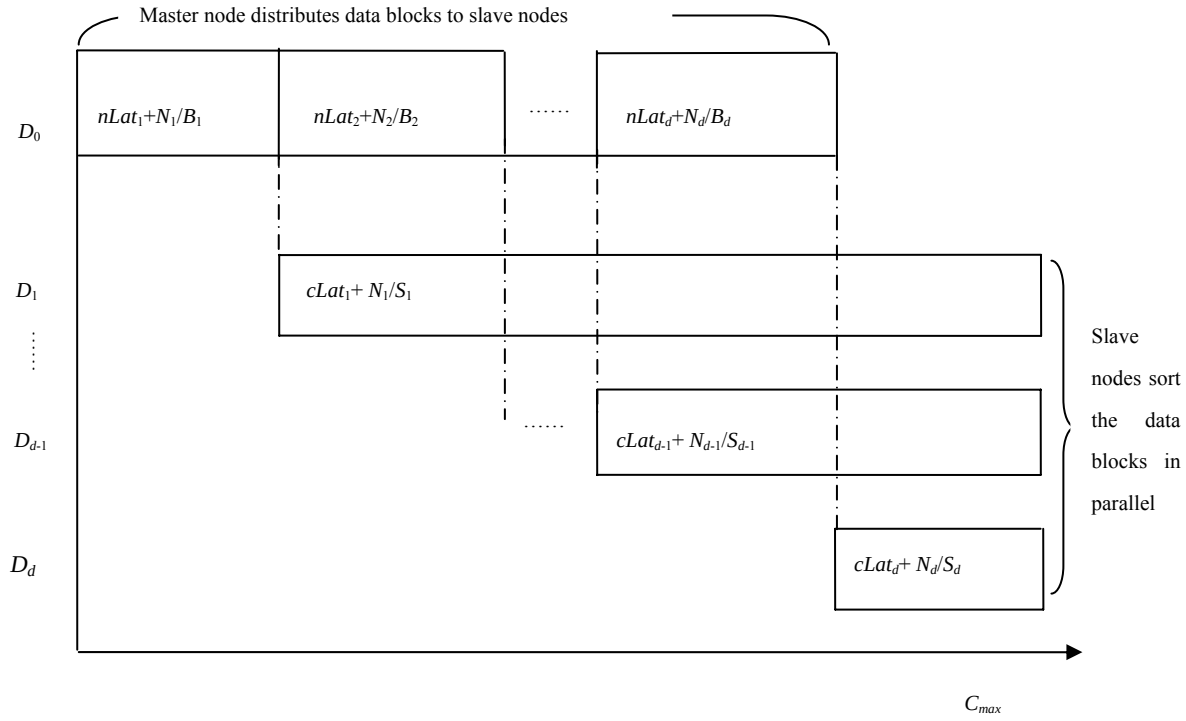

Figure 1 Procedure of distributing data with single-round and parallel sorting Multisets on heterogeneous multi-core clusters

Secondly, each salve node on the heterogeneous multi-core cluster has different size of main memory space. For the large-scale input data for Multisets, if master node distributes all the input data into the slave nodes in single round scheduling mode, the amount of the data to be assigned to each slave node may exceed its usable memory capacity, and the size of the data scheduled to each slave node will be remarkably increased. At this time, it is difficult to overlap computation and communication operations, and the communication overhead will be enhanced.

Finally, the input sequence for Multisets may be not uniform distribution. That is to say, some salve nodes will receive the data sequence for Multisets with fewer distinct keys, and the others will receive the data sequence for Multisets with relatively more different keys. Clearly, for the input sequence for Multisets with the same size, the required time to sort the data sequence with fewer distinct keys is much less than the required time to sort the data sequence with more different keys. If master node distributes the input data for Multisets to the slave nodes in each round scheduling mode according to their computational capability only, different slave node may require distinct amount of time to sort completely its received data sequence. At this time, some slave nodes are busy and the others will be free.

To solve the above problems, we present an aperiodic multi-round data distribution strategy to sort the input sequence for Multisets. Its goal is to ensure that all the slave nodes are always busy and each slave node can sort completely its received data sequence at the same time. The presented aperiodic multi-round data distribution strategy must guarantee that each slave node can receive the next data block before it has sorting the current data block, each slave node immediately starts to sort the next data block when it has sorted the current data block, and there is no free waiting time in the parallel sorting process. The free waiting will only appear when no slave nodes request master node to send data block and master node merges completely the sorted subsequences returned from the slave nodes.

Definition 4 The so-called First Request First Distribution (short for FRFD) strategy is that master node is responsible for storing the input data sequence and distributing the data to the slave multi-core nodes on the heterogeneous cluster in multi-round scheduling mode, and if over a salve node require the next data blocks, then master node will distribute the data blocks to the slave node with the first request.

Now we present an aperiodic multi-round data distribution model on the heterogeneous clusters with multi-core nodes:

$$\sum_{i=1}^{d} N_i = n \tag{1}$$

$$N_i \geq 0, 1 \leq i \leq d \tag{2}$$

$$cLat_i + N_i / S_i + \sum_{j=1}^{i} nLat_j + N_j / B_j \leq C_{\max}, \quad 1 \leq i \leq d \tag{3}$$

$$N_i \leq buf_i, 1 \leq i \leq d \tag{4}$$

$$\sum_{i=1}^{d}(nLat_i + bk_i / B_i) \le tim \quad (5)$$

$$bk_i < buf_{\min} / 2, i = 1 \sim d \quad (6)$$

$$\sum_{i=1}^{d} bk_i < n \quad (7)$$

where *tim* represents the required execution time that the slave node with the strongest computational ability sorts the subsequence of size bk_i, i=1~d, and $buf_{\min}$ denotes the capacity of main memory on the slave node with the smallest storage capability. The value of bk_i will be determined by the experiment, i=1~d. We set up bk_i=buf_i /j^l and can obtain the appropriate value of bk_i to meet the conditions (5), conditions (6) and conditions (7) by experimental testing, where j is a positive integer and j>2, l is also a positive integer and l≥1, i=1~d.

The parallel sorting Multisets algorithm using the aperiodic multi-round data distribution strategy on the heterogeneous clusters with multi-core nodes, short for PSAMRD-Multisets, is described as follows:

Algorithm 2 PSAMRD-Multisets

Begin

(1) According to the given order of nodes D_1, D_2,..., and D_d, master node D_0 distributes two data blocks to slave node D_i, which the size of each block is bk_i, i=1~d.

(2) Slave node D_i immediately sorts the first received data block of size bk_i by algorithm CETLPS-Multisets while it has received two data blocks, i=1~d.

(3) After the first received data block has been sorted, slave node D_i immediately begins to sort the second received data block and retruns the sorted data subsequence of size bk'_{ji} to master node D_0, and slave node D_i simuitaneously requests master node D_0 to distribute the next data block of size bk_i to it, i=1~d. Since slave node D_i returns only the data with same key in the sorted subsequence of size bk_i and the amount of these data to master node D_0, we can know that bk'_{ji}<<bk_i, i=1~d.

(4) If more than one slave node requests master node D_0 to distribute the next data block, master node D_0 sends the next data block of size bk_i to the corresponding slave node by the First Request First Distribution strategy. Repeat this step until all the data blocks for Multisets have been distributed and sorted.

(5) Master node D_0 merges the sorted subsequences returned from d slave nodes to a new sorted longer sequence while it distributes the data blocks to the slave nodes in each round scheduling. Hence, node D_0 can almost merge completely the sorted subsequences to a whole sorted sequence of length n when d slave nodes have sorting their received data blocks.

End.

Figure 2 shows that master node D0 distributes the data blocks to the slave nodes and the slave nodes sort the received data blocks and return the sorted subsequences to master node on heterogeneous cluster with one master multi-core node and three slave multi-core nodes by using the aperiodic multi-round scheduling and first request first distribution strategy.

For the presented algorithm PSAMRD-Multisets, we emphasizes sufficiently utilization of both heterogeneity of the multi-core clusters and the multi- level cache and multiple threading techniques. Algorithm PSAMRD-Multisets has the following advanteges. Firstly, it considers the usable amount of main memory capacity on each slave node and distributes the data block with appropriate size to each slave node. Secondly, it aims at the property of sorting Multisets, it ensures that each slave node can receive the next data block befor it has sorting the current data block and the communication links on the hetergeneous multi-core cluster are alway busy, and it can improve remarkably the performance of parallel sorting Multisets. Thirdly, it applys first request first distrbution strategy to assign the data blocks to the slave nodes, and it can eliminate the restriction of the bandwidth of main commuincation link. Fourthly, it applies the aperiodic multi-round scheduling strategy to distribute data blocks to the slave nodes, and it sorts quickly the data blocks on the single multi-core node by executing algorithm CETLPS-Multisets and can overlap computation and communication operation. Finally, we notice that the input sequence for Multisets of length n has only distinct k keys, where k<<n, and since each slave node returns only the sorted data with same key and the amount of these data to master node while it has sorting its received data subsequence, the amount of the data returned to master node is very small and the required communication overhead can be ignored.

III. EXPERIMENT

We first test the cache-efficient and thread-level parallel sorting Multisets algorithm on single multi-core computer, then evaluate the parallel sorting Multisets algorithm using aperiodic multi-round distribution strategy on the heterogeneous clusters with multi-core computers and analyze its scalability.

A. The experiments on single multi-core computer

The multi-core computer has Intel Core(TM) 2 Quad Processor, the running operating system is Linux kernel 12.0, the programming language is C and parallel communication library is OpenMP. Each element of input sequence for Multisets is from {0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,…,748.5,749,749.5,750}, and k=1501. We select a half of size of the shared L2 cache on the Intel Core(TM) 2 Quad Processor as the size of a data block.

We test the execution time and speedup of algorithm CETLPS-Multisets with different number of the processing cores and distinct number of parallel threads, and evaluate the performance of algorithm CETLPS-Multisets with blocking and non-blocking data technique, respectively.

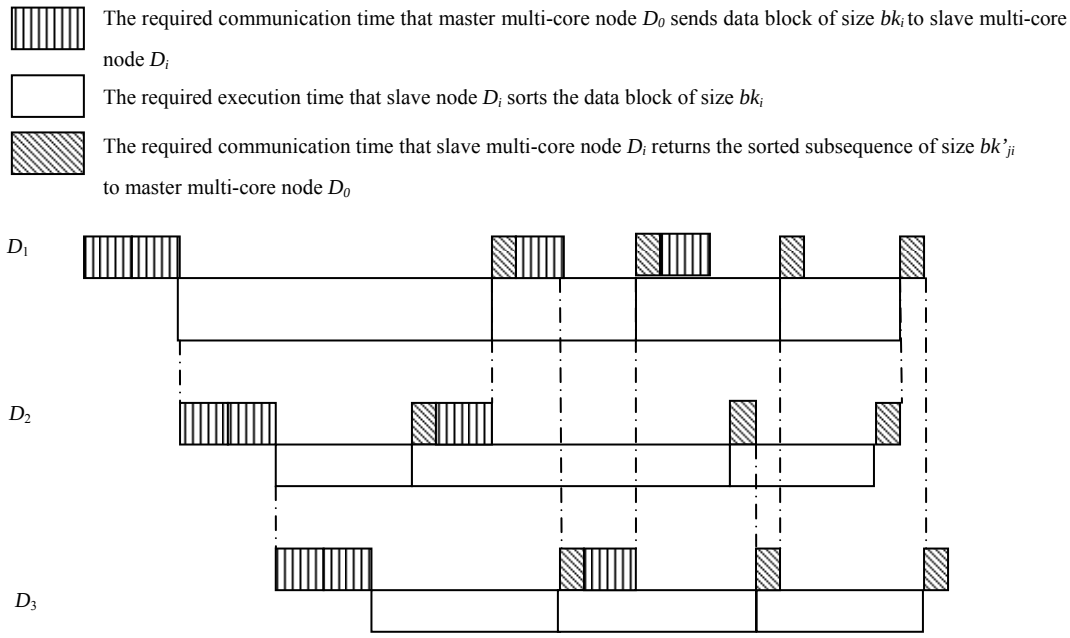

Figure 2. Procedure that master node distributes the data to slave nodes, slave nodes sort the received data and return the sorted subsequences to master node on heterogeneous multi-core clusters

By switching dynamically the processing cores on the multi-core computer, we test the performance of algorithm CETLPS-Multisets.

Figure 3 shows the execution time of algorithm CETLPS-Multisets using four processing cores and different number of parallel threads respectively.

From Figure 3, we can see that for the input data of different size, when algorithm CETLPS-Multisets is executed by 4 processing cores, its required time is gradually decreased with the number of threads varying from 4 to 8, and the required time is gradually increased with the number of threads varying from 8 to 20. In other words, if algorithm CETLPS-Multisets is executed by 4 processing cores and 8 threads, its required time is the smallest.

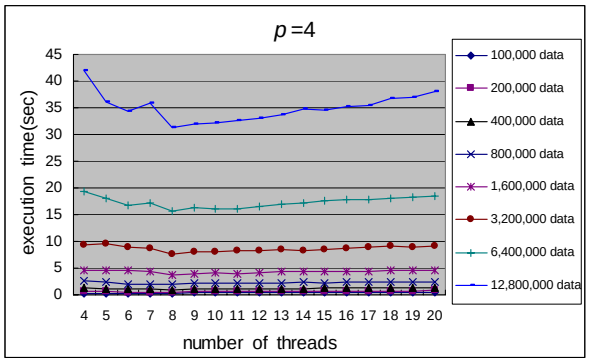

Figure 3 Execution time of algorithm CETLPS-Multisets using 4 processing cores and multiple threads

Figure 4 gives the execution time of algorithm CETLPS-Multisets using three processing cores and different number of threads respectively.

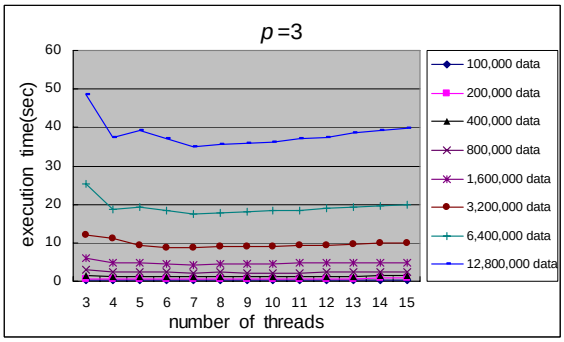

Figure 4 Execution time of algorithm CETLPS-Multisets using three processing cores and multiple threads

Figure 4 shows that for the input data with different size, when algorithm CETLPS-Multisets is executed by 3 processing cores, its execution time is gradually decreased with the number of threads varying from 3 to 7, and the execution time is gradually increased with the number of threads varying from 7 to 15. Hence, if algorithm CETLPS-Multisets is executed by 3 processing cores and 7 threads, its execution time is the smallest.

Figure 5 displays the execution time of algorithm CETLPS-Multisets using two processing cores and multiple threads.

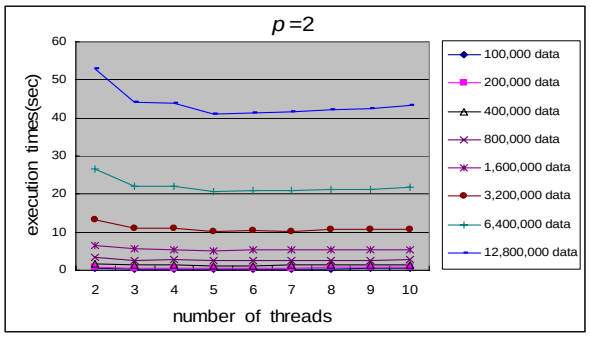

Figure 5 Execution time of algorithm CETLPS-Multisets using 2 processing cores and multiple threads

According to Figure 5, for the input data with different size, when algorithm CETLPS-Multisets is executed by 2 processing cores, its execution time is gradually decreased with the number of threads varying from 2 to 5, and the execution time is gradually increased with the number of threads varying from 5 to 10. It represents that its execution time is the smallest if algorithm CETLPS-Multisets is executed by two processing cores and five threads.

When the size of input data is fixed, Figure 6 shows the speedup of algorithm CETLPS-Multisets using four processing cores and multiple threads.

As we can see from Figure 6 that, when the size of input data is 12,800,000, if algorithm CETLPS- Multisets is executed by 4 processing cores and 8 threads, its speedup is the largest, and if algorithm CETLPS-Multisets is executed by 4 processing cores and 4 threads, its speedup is the smallest.

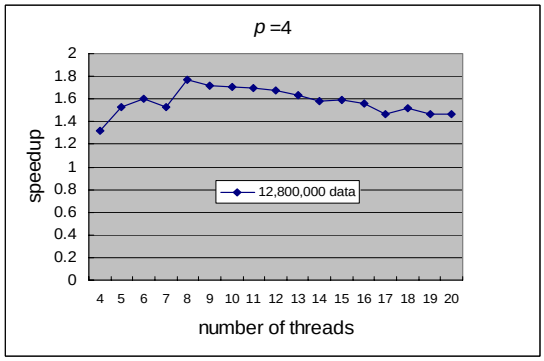

Figure 6 Speedup of algorithm CETLPS-Multisets using 4 processing cores and multiple threads

For the size of input data is 12,800,000, Figure 7 illustrates its speedup when algorithm CETLPS-Multisets is executed by three processing cores and multiple threads.

As we have shown in Figure 7, when the size of input data is 12,800,000, if algorithm CETLPS- Multisets is executed by three processing cores and seven threads, it obtains the largest speedup, and if algorithm CETLPS-Multisets is executed by three processing cores and three threads, it achieves the smallest speedup.

For the size of the input data is 12,800,000, when algorithm CETLPS-Multisets is executed by two processing cores and multiple threads, Figure 8 gives its obtained speedup.

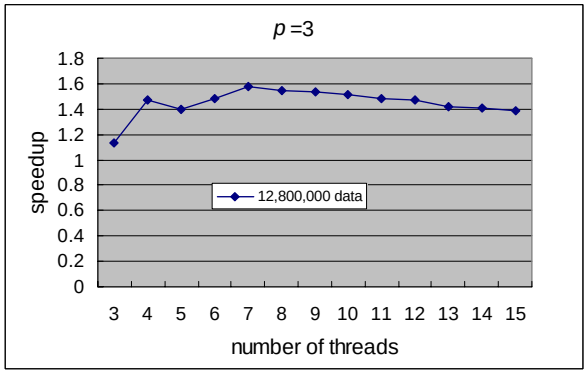

Figure 7 Speedup of algorithm CETLPS-Multisets using 3 processing cores and multiple threads

From Figure 8, we can see that when the size of input data is 12,800,000, its speedup is the smallest if algorithm CETLPS-Multisets is executed by two processing cores and two threads, and its speedup is the largest if algorithm CETLPS-Multisets is executed by two processing cores and five threads.

Figure 9 shows the execution time of algorithm CETLPS-Multisets using the threads of the optimal number and multiple processing cores.

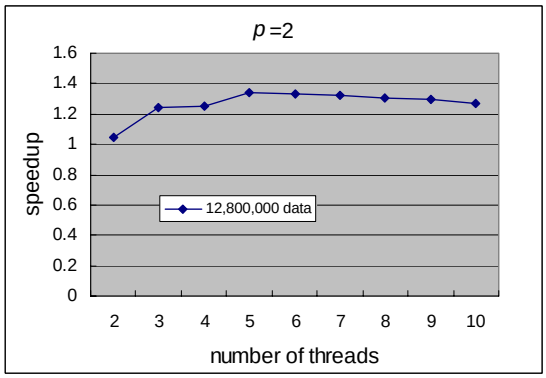

Figure 8 Speedup of algorithm CETLPS-Multisets using two processing cores and multiple threads

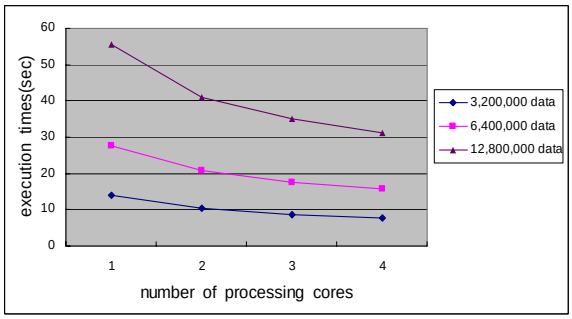

Figure 9 Execution time of algorithm CETLPS-Multisets using multiple processing cores and threads of optimal number

From Figure 9 we can see that, for the input data of different size, if algorithm CETLPS-Multisets is executed by threads with the optimal number, its required time is gradually decreased with the increase of used processing cores. Furthermore, if algorithm CETLPS-Multisets is executed by more processing cores, the descending trend of its execution time is gradually slow. This is due to the fact that the proportion of the communications among the

processing cores is large at this time, at the meanwhile, several processing cores may want to access some shared resources and spend some additional time.

Figure 10 gives the speedup of algorithm CETLPS-Multisets when it is executed by multiple processing cores and the threads of optimal number.

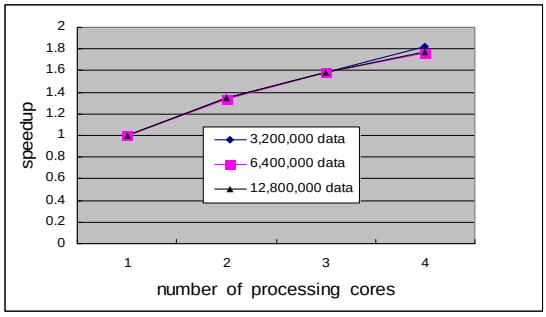

Figure 10 Speedup of algorithm CETLPS-Multisets using multiple processing cores and threads with optimal number

From Figure 10, we can see that the speedup of algorithm CETLPS-Multisets is gradually ascended with increase of used processing cores. However, when algorithm CETLPS-Multisets is executed by using more processing cores, there are more and more communications on the processing cores and the ascending trend of speedup is relatively slow.

When algorithm CETLPS-Multisets with data blocking and non-blocking is executed by 1, 2, 3 and 4 processing cores, respectively, Figure 11 shows their execution time. When algorithm CETLPS-Multisets is implemented by blocking data technique, the size of each data block in main memory is designated by the algorithm and then the data block is read into the shared L2 cache. When algorithm CETLPS-Multisets is implemented by non-blocking data technique, the size of each data block in main memory is designated by the computer system and then the data block is read into the shared L2 cache.

As shown in Figure 11, the required time of executing algorithm CETLPS-Multisets with blocking data is obviously less than that of the one with non-blocking data, and the blocking data technique is more suitable for parallel sorting the large-scale input sequence for Multisets.

For the input dataset with various distribution, when the size of input data is fixed n=12,800,000, Figure 12 gives the execution time of algorithm CETLPS- Multisets using the threads with optimal number and multiple processing cores.

From Figure 12, we can see that when algorithm CETLPS-Multisets is executed by the threads of optimal number, its required time is gradually decreased with the increase of used processing cores. Furthermore, the required time to sort the almost sorted input dataset is the smallest, and the execution time to sort the random input dataset is the most.

For the input sequence with various distribution, Figure 13 illustrates the execution time of algorithm CETLPS-Multisets using four processing cores and the threads of optimal number respectively.

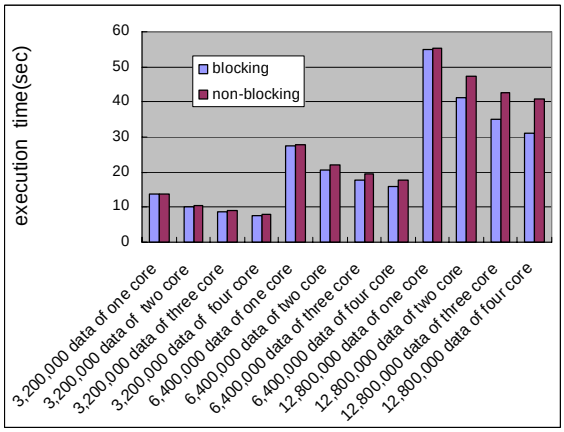

Figure 11 Execution time of algorithm CETLPS-Multisets with data blocking and non-blocking

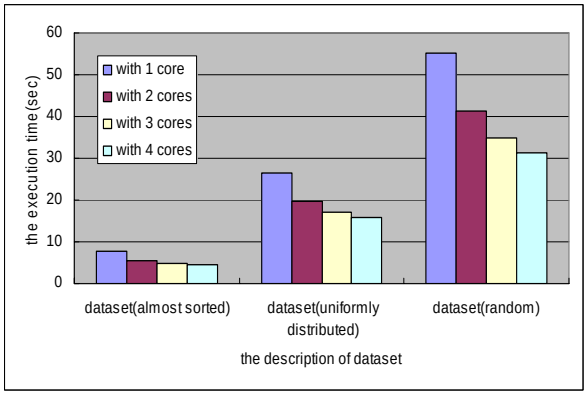

Figure 12 Execution time of CETLPS-Multisets with the input data of various distribution using multiple processing cores

We can see from Figure 13 that for the input data set with different size, when algorithm CETLPS- Multisets is executed, its required time to sort the almost sorted dataset is the smallest, and its required time to sort the random dataset is the most.

Figure 14 shows the curve of equivalent efficiency function of algorithm CETLPS-Multisets with input data of different size.

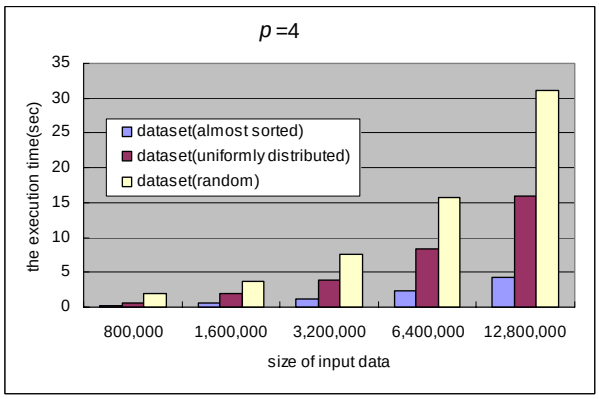

Figure 13 Execution time of algorithm CETLPS-Multisets for the input data with various distribution using 4 processing cores

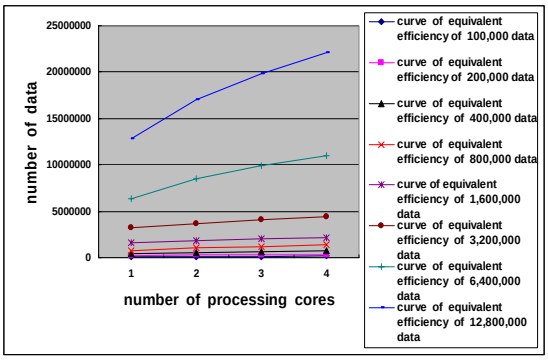

Figure 14 Equivalent efficiency function curve of algorithm CETLPS-Multisets with input data of different size

From Figure 14, we can see that the increase of the required workload is sub-linearly proportional to the increase of used processing cores to maintain the equivalent efficiency of algorithm CETLPS-Multisets for the input data of arbitrary size. Hence, algorithm CETLPS-Multisets has good scalability.

B. The experiments on the heterogeneous cluster with multi-core computers

The experimental platform is a heterogeneous cluster, which consists of the multi-core computers connected via a 100Mbps Ethernet network. The operating system is Red Hat Linux 12.0. The programming language and development environment are C, MPI and OpenMP respectively. The configuration of the heterogeneous multi-core cluster is listed in Table I. The input sequence for Multisets used in the experiment are also from the set of {0, 0.5, 1, 1.5, 2, 2.5, …, 99.5, 100, 100.5,…, 700.5,…,749.5, 750}, where k=1501.

A multi-core computer with the main memory of size 512MB and L2 cache of size 2MB is used as the master node and the other multi-core computers are treated as the slave nodes in the experiment. We test each communication link between master node and every slave node twice and obtain the relationship between the amount of data to be transferred and the required communication time, we achieve the values of B_i and $nLat_i$ by solving the corresponding equations, i=1~d. By executing twice algorithm CETLPS-Multisets on each multi-core node, we get the relationship between the amount of data to be sorted and the required computation time, and obtain the values of S_i and $cLat_i$ by solving the corresponding equations, i=1~d.

TABLE I.
CONFIGURATION OF COMPUTING NODES ON HETEROGENEOUS MULTI-CORE CLUSTER

Type of Nodes	CPU	Size of Main Memory	Size of Shared L2 Cache
INTEL	Intel quad-core	2GB	12MB
IBM	AMD dual-core	1GB	1MB
HP	Intel single-core	512MB	2MB
LEGEND	Intel dual-core	2GB	4MB

We evaluate the execution performance of algorithm PSAMRD-Multisets. Figure 15 first shows its execution time with the input data of different size and multiple multi-core computers.

From Figure 15 we can see that with the increase of the amount of multi-core nodes, the required execution time of algorithm PSAMRD-Multisets is remarkably decreased. We also see that the decrease of the execution time of algorithm PSAMRD-Multisets is slow down when more and more multi-core nodes is used. The reason is that the required communication overhead and the overhead to access shared cache and other resources will be increased when more and more multi-core nodes are used.

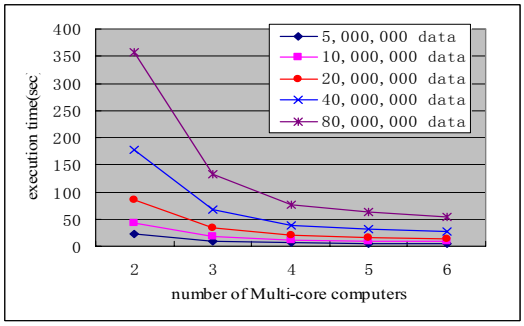

Figure 15. Execution time of algorithm PSAMRD-Multisets with input data of different size and multiple multi-core nodes

For the input sequence for Multisets of different size, the obtained speedup of algorithm PSAMRD-Multisets is given in Figure 16.

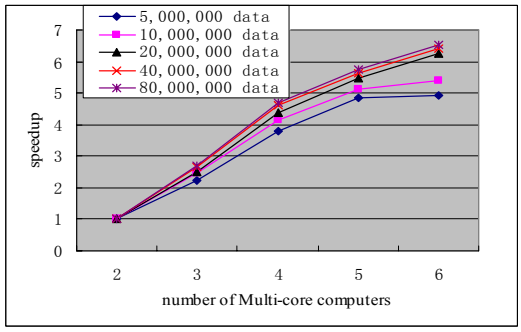

Figure 16. Speedup of algorithm PSAMRD-Multisets with input data of different size

Figure 16 shows that for the fixed number of multi-core nodes, the speedup of algorithm PSAMRD-Multisets is enhanced when the size of the input data sequence is increased. Besides, it shows that the increase of the speedup is slow down when more and more multi-core nodes are used.

When the amount of used multi-core nodes is d=6, Figure 17 displays speedup of algorithm PSAMRD-Multisets when the size of input data sequence is increased progressively.

We can see from Figure 17 that with increase of the size of the input data, the speedup of algorithm PSAMRD-Multisets is progressively enhanced, and the increase trend of the speedup is slowly to be stable. This result shows that algorithm PSAMRD-Multisets is very efficient for sorting the large-scale input data sequence for Multisets.

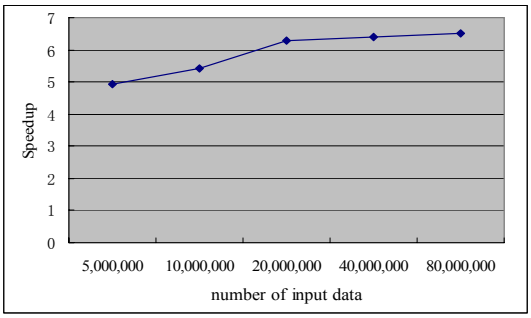

Figure 17. Speedup of algorithm PSAMRD-Multisets with input data of different size and 6 multi-core nodes

For the different combination with the amount of multi-core nodes and the number of processing cores, when the number of input data sequence for Multisets is increased, the required execution time of algorithm PSAMRD-Multisets is shown in Figure 18.

Figure 18 indicates that when the size of the input data sequence is the same, the required execution time of algorithm PSAMRD-Multisets is decreased with increase of the amount of multi-core nodes; when the size of the input data sequence is the same and the amount of used multi-core nodes is also the same, the more the nodes used, the shorter the required execution time of algorithm PSAMRD-Multisets. Figure 18 also shows that algorithm PSAMRD-Multisets can utilize sufficiently the hardware characteristics and software technology of multi-core architectures.

The scalability of the presented parallel sorting algorithm PSAMRD-Multisets can be analyzed by the equivalent efficiency functions.The curve of the equivalent efficiency function of algorithm PSAMRD-Multisets is given in Figure 19.

Figure 19 shows that for arbitrary size *n* of input data sequence, the increase of size of the required workload is sub-linearly proportional to the increase of used multi-core computers (nodes) in order to maintain the equivalent efficiency of algorithm PSAMRD-Multisets. Therefore, algorithm PSAMRD-Multisets has good scalability and it can utilize the computational ability of the increasing multi-core nodes.

IV. CONCLUSIONS

In this paper, we propose the cache-efficient, thread-level parallel and scalable algorithms for sorting Multisets on single multi-core computer and the heterogeneous cluster with multi-core computers, respectively. The presented algorithms sufficiently consider the distribution property of input dataset Multisets and the heterogeneity of multi-core clusters, utilize the function of multi-level cache and parallel multi-threading technique. The presented algorithms implement sorting Multisets using thread-level parallel selection and merge techniques. To efficiently sorting Multisets on heterogeneous multi-core clusters, this paper presents a single-round data distribution model and an aperiodic multi-round data distribution model. The presented aperiodic multi-round data distribution strategy can guarantee no free waiting time in the parallel sorting. On the one hand, the next works is study to design the cache-efficient, fast and scalable thread-level parallel sorting algorithm for more general sequence on the heterogeneous multi-core clusters. On the other hand, we will also study to design the efficient and scalable parallel sorting algorithms on the heterogeneous cluster systems with multi-core general computers and GPU computers.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China under grant NO. 60963001, Project of Outstanding Innovation Teams Construction Plans at Guangxi University, and Program to Sponsor Teams for Innovation in the Construction of Talent Highlands in Guangxi Institutions of Higher Learning.

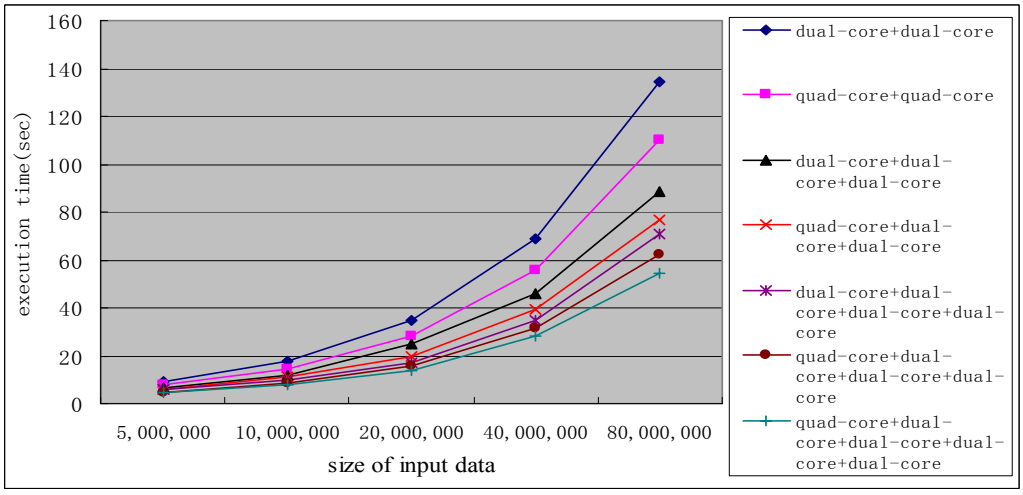

Figure 18. Execution time of algorithm PSAMRD-Multisets with various combination of amount of multi-core nodes and processing cores when the size of input data sequence is increased

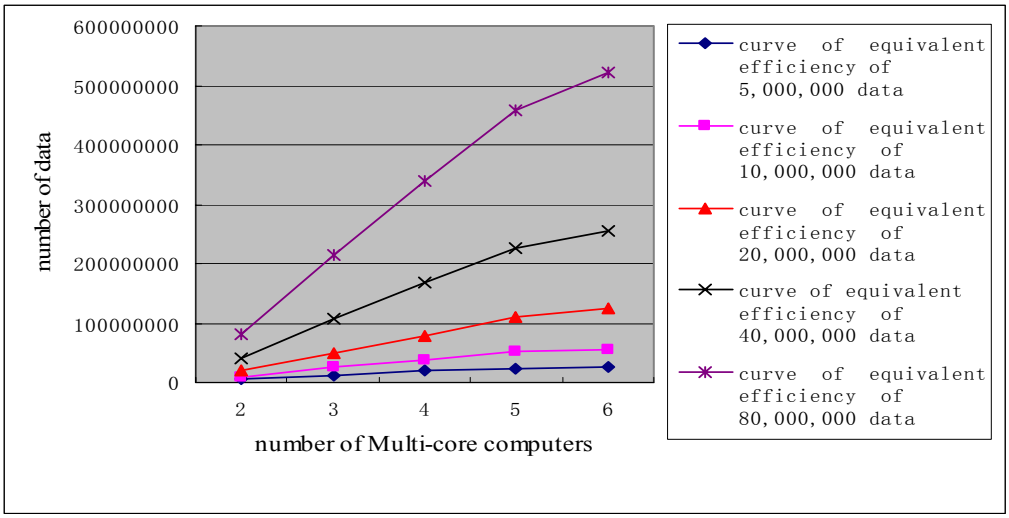

Figure 19. Curve of the equivalent efficiency function of algorithm PSAMRD-Multisets

REFERENCES

[1] Chen Guoliang, Design and Analysis of Parallel Algorithms, Third edition, Higher Education Press, Beijing,2009.

[2] T. J. Purcell, C.Donner, M.Cammarano, et al, "Photon mapping on programmable graphics hardware", Proc. of 2003 ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware, ACM Press, 2003,pp.41-50.

[3] N. Govindaraju,J. Gray,R. Kumar,et al, "Gputerasort: High performance graphics coprocessor sorting for large database management", Proc. of 2006 ACM SIGMOD /PODS, Chicago, USA, ACM Press, 2006, pp.325-336.

[4] A. Greb,G. Zachmann, "GPU-ABiSort: efficient parallel sorting on stream architectures", Proc. of 2006 Parallel and Distributed Processing Symposium(IPDPS'06), Rhodes Island, Greece, 2006, pp. 25-29.

[5] E. Sintorn,U. Assarsson, "Fast Parallel GPU-Sorting Using a Hybrid Algorithm", Journal of Parallel and Distributed Computing, Vol.68,No.10, 2008, pp.1381 -1388.

[6] B. Gedik, R. R. Bordawekar, and P. S. Yu, "CellSort: High Performance Sorting on the Cell Processor", Proc. of VLDB '07, 2007, pp. 1286-1297.

[7] J. Keller, C. W. Kessler, "Optimized Pipelined Parallel Merge Sort on the Cell BE", Proc. of 14th International European Conference on Parallel and Distributed Computing (Euro-Par), Highly Parallel Processing on a Chip (HPPC),2008,pp.26-29.

[8] N. Ramprasad, Pallav Kumar Baruah, " Radix Sort on the Cell Broadband Engine", Proc. of . 2007 Intl. Conf. on High Performance Computing (HiPC), Posters, 2007.

[9] D. Sharma,V. Thapar,R. Ammar, "Efficient sorting algorithms for the cell broadband engine", Proc. of 2008 IEEE Symposium on Computers and Communications, 2008, pp.736-741.

[10] H. Inoue,T. Moriyama,H. Komatsu, et al, "AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors", Proc. of PACT'07, 2007, pp.189-198.

[11] J. Chhugani, W. Macy, and V. W. Lee, et al, "Efficient Implementation of Sorting on MultiCore SIMD CPU Architecture", Proceedings of the VLDB Endowment, Vol.1, No.2, 2008, pp.1313-1324.

[12] Li Xia, Zhong Cheng, Qu Zeng-Yan, "Multi-round Scheduling for Divisible Loads on the Heterogeneous Cluster Systems of Multi-core Computers", Proc. of the 2nd International Conference on Interaction Science: Information Technology, Culture and Human, ACM Press, Vol.2, 2009,pp.846-852.

[13]Cheng Zhong, Zeng-yan Qu, Feng Yang, Meng-xiao Yin, "Parallel Multisets Sorting Using Aperiodic Multi-round Distribution Strategy on Heterogeneous Multi-core Clusters", Proc. of 3rd International on Parallel Architectures, Algorithms and Programming, IEEE Computer Society Press, Dec.2010, pp.247-254.

[14] Y. Yang, Henri Casanova, "RUMR: Robust Scheduling for Divisible Workloads", Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society Press, 2003,pp. 114-123.

[15] S. Rajasekaran, "An efficient parallel algorithm for sorting multisets", Information Processing Letters, Vol.67, No.3,1998, pp.141-143.

[16] Zhong Cheng, Chen Guo-Liang, "An Efficient Parallel Sorting Algorithm for Multisets", Journal of Computer Research and Development, Vol.40,No.2,2003,pp.335-361.

Cheng Zhong received a Ph.D. degree in department of computer science and technology at University of Science and Technology of China in July 2003. He is working as a professor of computer science in the school of computer and electronics and information at Guangxi University in China. He is a senior member of China Computer Federation (CCF), a standing member of CCF on Theoretical Computer Science Committee and Open systems Committee and PC Co-Chairs and PC member for several international and national conferences in computer science and technology. He has published over 60 papers in the journals and conferences and 7 textbooks. His current main research interests include parallel and distributed computing, network security, highly -trusted software, and Bioinformatics.

Zeng-yan Qu received a M.Sc degree in the school of computer and electronics and information at Guangxi University in China in July 2010. Her research interest is parallel and distributed computing.

Feng Yang received a M.Sc degree in school of computer and electronics and information at Guangxi University in China in July 2005. He is working as a lecturer in the school of

computer and electronics and information at Guangxi University. His research interests include parallel computing, network security, and Bioinformatics.

Meng-Xiao Yin received a M.Sc degree in school of Mathematics and information at Guangxi University in China in July 2005. She is working as a lecturer in the school of computer and electronics and information at Guangxi University. Her research interests include parallel computing and graph theory.

Xia Li received a M.Sc degree in the school of computer and electronics and information at Guangxi University in China in July 2010. Her research interest is parallel and distributed computing.