ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.11, November 2007, pp.2683−2690
DOI: 10.1360/jos182683

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

PRAM 和 LARPBS 模型上有向序列翻转距离并行算法*

沈一飞[1,2+], 陈国良[1,2], 张强锋[1,2]

[1](中国科学技术大学 计算机科学技术系,安徽 合肥 230027)

[2](国家高性能计算中心(合肥),安徽 合肥 230027)

Parallel Algorithms for Reversal Distance of Permutations on PRAM and LARPBS

SHEN Yi-Fei[1,2+], CHEN Guo-Liang[1,2], ZHANG Qiang-Feng[1,2]

[1](Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

[2](National High Performance Computing Center at Hefei, Hefei 230027, China)

+ Corresponding author: Phn: +86-551-3601548, Fax: +86-551-3601013, E-mail: shenyf@ustc.edu

Shen YF, Chen GL, Zhang QF. Parallel algorithms for reversal distance of permutations on PRAM and LARPBS. *Journal of Software*, 2007,18(11):2683−2690. http://www.jos.org.cn/1000-9825/18/2683.htm

Abstract: This paper presents two parallel algorithms to compute reversal distance of two signed permutations on different models. These two algorithms are based on Hannenhalli and Pevzner's theory and composed of three key steps: Construct break point graph, compute the number of cycles in break point graph and compute the number of hurdles in break point graph. The first algorithm runs in $O(\log^2 n)$ time using $O(n^2)$ processors in SIMD-CREW model. The second one can solve the problem in $O(\log n)$ bus cycles by using $O(n^3)$ processors on the linear array with a reconfigurable pipelined bus system (LARPBS).

Key words: parallel algorithm; optical bus parallel model; reversal distance; genome rearrangements; sequence comparison; CREW-PRAM model

摘 要: 分别在两种重要并行计算模型中给出计算有向基因组排列的反转距离新的并行算法.基于Hannenhalli和Pevzner 理论,分 3 个主要部分设计并行算法:构建断点图、计算断点图中圈数、计算断点图中障碍的数目.在CREW-PRAM 模型上,算法使用 $O(n^2)$处理器,时间复杂度为 $O(\log^2 n)$;在基于流水光总线的可重构线性阵列系统(linear array with a reconfigurable pipelined bus system, LARPBS)模型上,算法使用 $O(n^3)$处理器,计算时间复杂度为 $O(\log n)$.

关键词: 并行算法;光总线并行模型;反转距离;基因组重排;序列比较;CREW-PRAM 模型

中图法分类号: TP301 **文献标识码**: A

1 Introduction

Computing reversal distance of two signed permutations has gained increasing attention over the last decade

* Supported by the Key Project of the National Natural Science Foundation of China under Grant No.60533020 (国家自然科学基金重点项目)

Received 2006-01-17; Accepted 2006-06-27

with the study of genome rearrangements in computational molecular biology. It plays an important role in checking DNA sequences similarity at the level of genome and finding evolutionary relationship between species.

A signed permutation is a permutation $\pi=\{\pi_1,\pi_2,\ldots,\pi_n\}$ on the set of integers $\{1,2,\ldots,n\}$ whose each element has a sign of plus or minus. A *reversal* $\rho(i,j)$ on permutation π transforms π to

$$\pi'=\pi\cdot\rho(i,j)=\{\pi_1,\ldots,\pi_{i-1},-\pi_j,-\pi_{j-1},\ldots,-\pi_{i+1},-\pi_i,-\pi_{j+1},\ldots,\pi_n\} \quad (1)$$

The *reversal distance* of two permutations is the minimum number of reversals needed to transform one to another permutation. The problem of computing reversal distance of permutation π is to find the minimum number of reversals needed to transform π to identical permutation $\{1,2,\ldots,n\}$.

In 1995, Hannenhalli and Pevzner[1,2] built a basic theory about how a signed permutation is sorted by reversals and gave the first polynomial-time algorithm to solve the problem of sorting a signed permutation by reversals, which runs in $O(n^2)$ time when restricted to distance computation. In 1996, a $O(n^2\alpha(n))$ reversal sorting algorithm was given by Berman and Hannenhalli[3], where $\alpha(n)$ is the Ackerman's function, it also provided the distance as a byproduct. Bader, Moret and Yan[4] showed how to compute reversal distance in the linear time. More recently, Bergeron[5] presented another $O(n)$ time algorithm for problem of reversal distance.

In this paper we present two parallel algorithms for computing reversal distance of a signed permutation of n elements that are based on Hannenhalli and Pevzner's theory and composed of three steps. The first algorithm consists three parts and runs in $O(\log^2 n)$ time using $O(n^2)$ processors in SIMD-CREW model[9]. The second one can solve the problem in $O(\log n)$ bus cycles by using $O(n^3)$ processors on the Linear Array with a Reconfigurable Pipelined Bus System (LARPBS) which has been investigated in Ref.[6] for designing fast algorithms from different domains. To our best knowledge, this is the best time complexity parallel algorithm.

2 Preliminary Definitions

In this section we introduce the basic background for our algorithms. The exposition follows closely the Hannenhalli and Pevzner's theory[1,2].

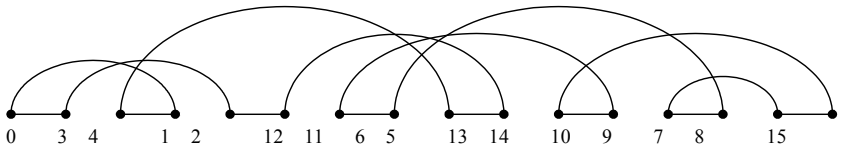
2.1 Basic definitions

Given a signed permutation π of $\{1,2,\ldots,n\}$, we transform it into an unsigned permutation π' of $\{1,2,\ldots,2n-1, 2n\}$ by replacing each positive element x in π by $2x-1$ and $2x$, and each negative element x by $2x$ and $2x-1$, then extend permutation to the set $\{0,1,\ldots,2n,2n+1\}$ by setting $\pi_0=0$ and $\pi_{2n+1}=2n+1$. We represent an extended unsigned permutation with a *breakpoint graph* of the permutation. The breakpoint graph has $2n+2$ vertices; for each i, $1\le i\le n$, we join vertices π_{2i} and π_{2i+1} by a black edge and vertices whose values are $2i$ and $2i+1$ by a gray edge. Notice that a gray edge (π_k,π_l) is oriented if the sum of $k+l$ is even, otherwise is unoriented. The resulting breakpoint graph consists of disjoint cycles in which edges alternate colors. A cycle in breakpoint graph is oriented if it has an oriented gray edge and unoriented otherwise.

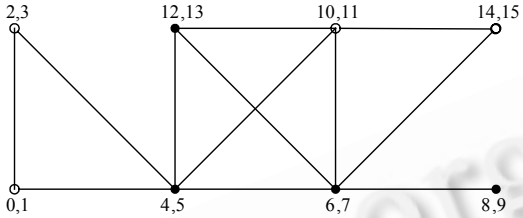
Gray edges (π_i,π_j) and (π_k,π_l) are overlapping whenever the two intervals $[i,j]$ and $[k,l]$ overlap, but neither contains the other. The overlap graph of a permutation π, denoted by $OV(\pi)$, is the overlap graph of the gray edges of $B(\pi)$. In other words, the node set of $OV(\pi)$ is the set of gray edges in $B(\pi)$, and two nodes are connected by an arc if two gray edges overlap. We shall identify a node in $OV(\pi)$ with the edge it represents and with its interval in the representation. Thus, the endpoints of a gray edge are actually the endpoints of the interval representing the corresponding node in $OV(\pi)$. A connected component of $OV(\pi)$ that contains an oriented edge is called an oriented component, otherwise, it is called an unoriented component.

Let M be an unoriented connected component in $OV(\pi)$. Let $E(M)$ be the set of end points of the edges in M. An unoriented component of π is a *hurdle* if the elements belonged to $E(M)$ occur consecutively, in the other words, the

elements belonged to $E(M)$ must be an interval from i to $(i+j)$ or from i to $(i+j)$mod$n$, whose set of elements is $(i,\ldots,i+j)$ but not in a single two-edged circle components. A hurdle is a *simple hurdle* if when one deletes it from $OV(\pi)$ no other unoriented component becomes a hurdle, otherwise is a *super hurdle*. Permutation π is called *fortress* if it has an odd number of hurdles and all these hurdles are super hurdles (Fig.1).



(a) The breakpoint graph $B(\pi)$ for the permutation π=(2,1,−6,−3,7,−5,4)



(b) The overlap graph $OV(\pi)$. Black vertices correspond to oriented edges

Fig.1

Lemma 1. For a signed permutation π of order n, $d(\pi)=n-c(\pi)+h(\pi)+f(\pi)$, where $c(\pi)$, $h(\pi)$ and $f(\pi)$ are the numbers of cycles, hurdles and, fortress in the breakpoint graph of the permutation π.

2.2 The LARPBS model

The LARPBS model connects its processors by an optical bus that uses optical waveguide instead of electrical bus to transfer messages among processors. The advantages of using optical waveguide are high propagation speed, unidirectional propagation and predictable propagation delay per unit length. The last two properties enable synchronized concurrent accesses of an optical bus in a pipelined fashion.

LARPBS can be partitioned into $i\geq 2$ independent subarrays, such those subarrays can be operated as regular linear arrays with pipelined optical bus systems, and all subarrays can be used independently for different computations without interference[6,7,9](Fig.2).

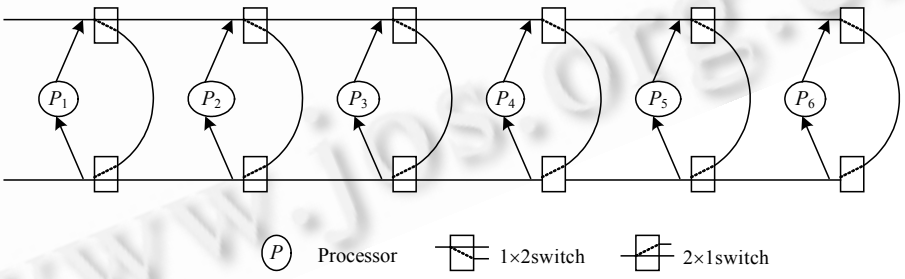


Fig.2 (Source [7]) The LARPBS model of size 6 with two subarrays

The measure of computational complexity on a LARPBS is the number of bus cycles used for the computation and the amount of time spent by the processors for local computations. A bus cycle is the time needed for end-to-end message transmission over a bus and assumed to take only $O(1)$ time.

The following basic communication, data movement, and global operations on the LARPBS are used in this

paper. The reader is referred to [6,7,9] for the implementation details of these operations.

Lemma 2[6,7,9]. One-to-One communication, broadcasting, multicasting and multiple multicasting, all can be done in $O(1)$ bus cycles on the LARPBS model.

Lemma 3[6,7,9]. For a LARPBS with n processors and n binary values v_i, $0\le i\le n-1$, the binary prefix sum requires the computation of $psum_i=v_0+v_1+\ldots+v_{i-1}$, for all $0\le i\le n-1$. It can be done in $O(1)$bus cycles on the LARPBS model.

Lemma 4[9]. Sorting n numbers can be performed in $O(1)$ bus cycles on the LARPBS model with $O(n^2)$ processors.

3 Algorithms

In this section, the complete algorithms for computing reversal distance are presented.

The Complete Algorithm Framework.

Input: Signed permutation π;

Output: Reversal distance of permutation π.

Step 1. Construct break point graph $B(\pi)$ of permutation π (Algorithm 1, Fig.3);

Step 2. Compute the number $c(\pi)$ of cycles in $B(\pi)$ (Algorithm 2, Fig.4);

Step 3. Construct overlap graph $OV(\pi)$ of permutation π based on break point graph $B(\pi)$ built in Step1; find all components in $B(\pi)$ and compute the number $h(\pi)$ of hurdles in $B(\pi)$;

Step 4. Determine whether breakpoint graph $B(\pi)$ is a fortress $f(\pi)$ and compute the reversal distance of π, $d(\pi)=n-c(\pi)+h(\pi)+f(\pi)$.

Algorithm 1. *Construct breakpoint graph*
Input: *Permutation*;
Output: *Breakpoint graph of permutation*.

```
1   for each i: 1≤i≤n pardo
2     if π_i>0 then
3       a_{2i-1}←2π_i−1, a_{2i}←2π_i
4     else
5       a_{2i-1}←2|π_i|, a_{2i}←2|π_i|−1
6     endif
7   endfor
8   a_0←0, a_{2n-1}←2n+1
9   for each i: 1≤i≤n pardo
10    connect (a_{2i},a_{2i+1}) with a black edge
11  endfor
12  for each i: 1≤i≤n pardo
13    b_{a_{2i}} ←2i,  b_{a_{2i-1}} ←2i−1
14  endfor
15  for each i: 1≤i≤n pardo
16    connect ( a_{b_{2i}} , a_{b_{2i-1}} ) with a gray edge
17  endfor
```

Fig.3 The framework of Algorithm 1: *construct breakpoint graph*

Algorithm 1 constructs break point graph $B(\pi)$ of permutation π in parallel. The part of lines (1~8) in Algorithm 1 transforms permutation π into an unsigned permutation according to the precious definition. For each i: $1\le i\le n$, we join vertices a_{2i} and a_{2i+1} by a black edge (9~11). We also need join vertices whose values are $2i$ and $2i+1$ by a gray edge. Assume we need to connect $a_x=2i$ and $a_y=2i+1$, from lines (12~14) we can get $b_{2i}=x$ and $b_{2i+1}=y$ and connect $(a_{b_{2i}},a_{b_{2i+1}})$ with a gray edge.

Algorithm 2 finds all cycles in the breakpoint graph $B(\pi)$ in parallel. In Algorithm 2, a cycle is labeled with the minimum vertex index of that cycle, and for each vertex π_i, we set $c(i)$ with the label of its cycle. Lines (1~3) are to initialize $c(i)$ with the vertex index; for the part of lines (4~16) in Algorithm 2, we use the point jumping method that is sufficed to show that, after kth iteration, for each vertex i, $c(i)$ must be the minimum vertex index within 2^k distance. It is easy to see that in each cycle, there is only one vertex whose index value($c(i)$) remains unchanged after the iteration part. We set $csum(i)$=1 if $c(i)=i$, otherwise $csum(i)$=0, then parallelly compute the sum of $csum(i)$.

By parallelly detecting every pair of edges ($O(n^2)$ pairs), the overlap graph $OV(\pi)$ can be built in $O(1)$ time using $O(n^2)$ processors. The method to find components used by Algorithm 3 is similar to Algorithm 2. Because each node in $OV(\pi)$ has at most $O(n)$ adjacent nodes, there remain some differences: line 7 and Algorithm 3 need to find the minimum number in $O(n)$ numbers; to avoid writing confliction, we specify an array in each node. For any node j, if $d(j)<c(c(j))$: $c(j)=i$, we set $w_i(j)=d(j)$ at node i (lines 9~16)(Fig.5).

Algorithm 2. Find all cycles in breakpoint graph.
Input: Breakpoint graph;
Output: All cycles in breakpoint graph.

```
1  for each i: 1≤i≤2n+1 pardo
2    c(i)←i
3  endfor
4  repeat
5    for each i: 1≤i≤2n+1 pardo
6      d(i)←min{c(j)|j are vertex adjacent to i}
7    endfor
8    for each i: 1≤i≤2n+1 pardo
9      if d(i)<c(c(i)) then
10       c(c(i))←d(i)
11     endif
12   endfor
13   for each i: 1≤i≤2n+1 pardo
14     c(i)←c(c(i))
15   endfor
16 until all c(i)=d(i)
17 for each i: 1≤i≤2n+1 pardo
18   csum(i)←0
19   if c(i)=i then
20     csum(i)←1
21   endif
22 endfor
23 parallel computing sum of csum(i): 1≤i≤2n+1
```

Fig.4 The framework of Algorithm 2: Find all cycles in breakpoint graph

Algorithm 3. Find all components of permutation.
Input: Breakpoint graph;
Output: All components in breakpoint graph.

```
1  for each node i: 1≤i≤n in OV(π) pardo
2    c(i)←min{b_i,e_i|b_i and e_i are the endpoints of gray edge i}
3  endfor
4  a_0←0, a_{2n-1}←2n+1
5  repeat
6    for each node i: 1≤i≤n in OV(π) pardo
7      d(i)←min{c(j)|j are node adjacent to i}
8    endfor
9    for each node i: 1≤i≤n in OV(π) pardo
10     if d(i)<c(c(i)) then
11       w_{c(i)}(i)←d(i)
12     endif
13   endfor
14   for each node i: 1≤i≤n in OV(π) pardo
15     c(i)←min{w_i(j)|j: 1≤j≤n}
16   endfor
17   for each node i: 1≤i≤n in OV(π) pardo
18     c(i)←c(c(i))
19   endfor
20 until all c(i)=d(i)
```

Fig.5 The framework of Algorithm 3: Find all components of permutation

3.1 Parallel algorithms on SIMD-CREW model

Theorem 1. Break point graph $B(\pi)$ of permutation π of $\{1,\ldots,n\}$ can be constructed in a constant time using n processors on SIMD-CREW model.

Proof: In all parts of Algorithms 1, processors don't read and write the same memory places. It is easy to get that Algorithm 1 can run in a constant time using n processors on SIMD-CREW model. □

Theorem 2. Finding all cycles in BP Graph takes $O(\log n)$ time using $O(n)$ processors on SIMD-CREW model.

Proof: The parts of lines (1-3) and (17-22) in Algorithm 2 both use $O(1)$ time; for the part of lines (4-16), after the kth iteration, for each vertex i, $c(i)$ will be the minimum vertex index within 2^k distance, and that is to say, after $O(\log l)$ iterations, each vertex of graph will be covered and each iteration uses a constant time. (l is the max

length of cycles, $l \le n$.) The base case is trivial: parallel computing sum of n numbers runs $O(\log n)$ time with n processors on SIMD-CREW model. Algorithm 2 uses $O(\log n+\log l)$ (l at most is n) time, so the time complexity of Algorithm 2 is $O(\log n)$. □

Theorem 3. Finding components in permutation π takes $O(\log^2 n)$ time using $O(n^2)$ processors on SIMD-CREW model.

Proof: As mentioned before, Line 1 in Algorithm 3 uses $O(1)$ time; Like Algorithms 2, after at most $O(\log n)$ iterations (lines 2~20), we can get the final result. The parts of lines (6~8) and (14~16) in iteration of Algorithm 3 need $O(\log n)$ time with n processors to compute the minimum number. The entire step uses $O(\log^2 n)$ time with $O(n^2)$ processors. □

Lemma 5. Whether a component of $OV(\pi)$ is a hurdle can be determined in $O(\log n)$ time using $O(n)$ processors on SIMD-CREW model.

Proof: According to the definition of hurdle, we need two steps to test a component: first, checking whether the component is unoriented component just needs to know whether all edges are unoriented; second, checking if there is no other component between start and end of it except for two-edged simple cycles. Both steps can complete in a constant time with n processors in SIMD-CREW model. We gather all data in $O(\log n)$ time. □

Theorem 4. The number of hurdles can be counted in $O(\log n)$ time with $O(n^2)$ processors on SIMD-CREW model.

Proof: There are at most $O(n)$ components in the permutation π. We can prove that all components can be determined in constant time using $O(n^2)$ processors as mentioned in Lemma 5. If a component is hurdle, we set $h(i)=1$, otherwise $h(i)=0$. Parallel computing sum of $h(i)$ takes $O(\log n)$ time with n processors. □

Theorem 5. Whether the permutation is fortress can be determined in $O(\log^2 n)$ time with $O(n^2)$ processors on SIMD-CREW model.

Proof: We delete all hurdles, and then compute the number of hurdles in the remaining graph. If the number of hurdles in the remaining graph is equal to the number before deleting and the number is odd, $f(\pi)=1$. That can be completed in $O(\log^2 n)$ time with n^2 processors on SIMD-CREW model. □

Theorem 6. Computing reversal distance of permutation π takes $O(\log^2 n)$ time with $O(n^2)$ processors on SIMD-CREW model.

Proof: The steps of the algorithm at most take $O(\log^2 n)$ time with $O(n^2)$ processors on SIMD-CREW model, so the entire algorithm runs in the worst-case $O(\log^2 n)$ time on SIMD-CREW model. □

3.2 Parallel algorithms on LARPBS

Theorem 7. Break point graph $B(\pi)$ of permutation π of $\{1,\ldots,n\}$ can be constructed in constant bus cycles using $O(n)$ processors on LARPBS.

Proof: rocessors can exchange data by using one-to-one communication that is a constant time operation, so Algorithm 1 also can run in constant bus cycles on a LARPBS of n processors. □

Theorem 8. Finding all cycles in BP Graph runs $O(\log n)$ bus cycles on a LARPBS of $O(n)$ processors.

Proof: he parts of lines (5~7) and lines (13~15) of Algorithm 2 both need to use multiple multicasting that takes $O(1)$ time (Lemma 2), and the operation of computing sum (line 23) only needs $O(1)$ time (Lemma 3) which is different from SIMD-CREW model. The iteration part also needs $O(\log n)$ bus cycles, then Algorithm 2 runs $O(\log n)$ bus cycles on a LARPBS of $O(n)$ processors. □

Theorem 9. Finding components in permutation π performs $O(\log n)$ time using LARPBS of $O(n^3)$ processors.

Proof: The parts of lines (5~19) of Algorithm 3 need to use multiple multicasting that just needs $O(1)$ time as mentioned in Lemma 2, and the parts of lines (6~8) and (14~16) use $O(1)$ time with $O(n^3)$ processors (Lemma 4)

that is different from SIMD-CREW model. The iteration part also needs $O(\log n)$ bus cycles, so Algorithm 3 runs $O(\log n)$ bus cycles on a LARPBS of $O(n^3)$ processors. □

Lemma 6. Whether a component of $OV(\pi)$ is a hurdle can be determined in constant bus cycles using $O(n)$ processors on LARPBS.

Proof: According to the definition of hurdle, we need two steps to test a component: first, checking whether the component is unoriented component just needs to know whether all edges are unoriented; second, checking if there is no other component between start and end of it except for two-edged simple cycles. Both steps can complete in constant time with n processors in LARPBS. But in LARPBS, we can gather all data in constant time. □

Theorem 10. The number of hurdles can be counted in constant bus cycles with $O(n^2)$ processors on LARPBS.

Proof: There are $O(n)$ components in the permutation π. All components can be determined in $O(1)$ time using $O(n^2)$ processors as mentioned in Lemma 6. If a component is a hurdle, we set $h(i)$=1, otherwise $h(i)$=0. Parallelly computing the sum of $h(i)$ can run $O(1)$ time with $O(n^2)$ processors, as mentioned in Lemma 3. □

Theorem 11. Whether the permutation is fortress can be determined in constant bus cycles on a LARPBS of $O(n^3)$ processors.

Proof: The proof is the same as Theorem 5. □

Theorem 12. Computing reversal distance of permutation π performs $O(\log n)$ time on a LARPBS of $O(n^3)$ processors.

Proof: Because the steps of the algorithms at most take $O(\log n)$ time as mentioned before, so the entire algorithm runs in the worst-case $O(\log n)$ time on a LARPBS of $O(n^3)$ processors. □

4 Conclusions

Two parallel algorithms for computing the reversal distance between two signed permutations have been presented in this paper. The time complexities of the algorithms are $O(\log^2 n)$ on SIMD-CREW model and $O(\log n)$ on LARPBS model. Because certain operations such as computing prefix sum and sorting a smaller set of data can be done in constant time on the LARPBS model, we are able to take advantage of them and to make the algorithm faster than PRAM. We believe many other algorithms can also take advantage of the high communication bandwidth on the LARPBS model. In addition, there are some problems concerning in this paper for the future work, such as, whether the time complexity and computational cost of the algorithm can be further improved. We expect to see more results published in this area in the future.

References:

[1] Hannenhalli S, Pevzner PA. Transforming men into mice (polynomial algorithm for genomic distance problem). In: Proc. of the 36th Annual Symp. on Foundations of Computer Science (FOCS'95). IEEE Press, 1995. 581−592.

[2] Hannenhalli S, Pevzner P. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. Journal of the ACM, 1999,46(1):1−27.

[3] Berman P, Hannenhalli S. Fast sorting by reversal. In: Proc. of the 7th Annual Combinatorial Pattern Matching Symp. (CPM'96). LNCS 1075, Springer-Verlag, 1996. 168−185.

[4] Bader DA, Moret BM, Yan M. A linear-time algorithm for computing inversion distance between signed permutation. Journal of Computer Biol., 2001,8:483−491.

[5] Bergeron A, Mixtacki J, Stoye J. Reversal distance without hurdles and fortresses. In: Proc. of the 15th Annual Combinatorial Pattern Matching Symp. (CPM 2004). LNCS 3109, Springer-Verlag, 2004. 388−399.

[6] Pan Y, Hamdi M. Quicksort on a linear array with a reconfigurable pipelined bus system. In: Proc. of the IEEE Int'l Symp. of Parallel Architectures, Algorithms, and Networks. IEEE Press, 1996. 313−319.

[7] Han YJ, Pan Y, Shen H. Sublogarithmic deterministic selection on arrays with a reconfigurable optical bus. IEEE Trans. on Computers, 2002,51(6):702−706.

[8] Chen GL. Parallel Algorithms: Design and Analysis. Beijing: Higher Education Press, 2002 (in Chinese).

[9] Zhong C, Chen GL. Parallel algorithms for approximate string matching on PRAM and LARPBS. Journal of Software, 2004,15(2): 159−169 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/159.htm

[10] Luan JF. Research on Problem of Genome Distance in Computational Biology [Ph.D. Thesis] Ji'nan: Shandong University, 2003.

附中文参考文献:

[8] 陈国良.并行算法——设计与分析.北京:高等教育出版社,1999.

[9] 钟诚,陈国良.PRAM 和 LARPBS 模型上的近似串匹配并行算法.软件学报,2004,15(2):159−169. http://www.jos.org.cn/1000-9825/15/159.htm

[10] 栾峻峰.计算生物学中有关基因组距离问题研究[博士学位论文].济南:山东大学,2003.

SHEN Yi-Fei was born in 1977. He received the B.S. degree in computer science (CS) from University of Science and Technology of China (USTC) in 2000. Now he is a Ph.D. candidate in CS at USTC. His current research areas are bioinformatics, combinatorial optimization and parallel computing.

CHEN Guo-Liang was born in 1938. He is a fellow of Chinese Academy of Sciences, a professor in the Department of CS at USTC and a CCF senior member. His research areas are parallel computing, computer architecture and combinatorial optimization.

ZHANG Qiang-Feng was born in 1979. He is a Ph.D. candidate in CS at USTC. His research areas are bioinformatics and combinatorial optimization.