A Parallel Computation for Shift-Add Algorithm

Baba, Kensuke
Faculty of Information Science and Electrical Engineering, Kyushu University and System LSI Research Center, Kyushu University

E, Hanmei
Graduate School of Information Science and Electrical Engineering, Kyushu University

Yu, Yunqing
Computing and Communications Center, Kyushu University

https://hdl.handle.net/2324/4289

出版情報：DOI Technical Report. 230, 2007-05-08. Department of Informatics, Kyushu University
バージョン：
権利関係：

DOI-TR-230

DOI Technical Report

A Parallel Computation for Shift-Add Algorithm

by

K. Baba, H. E, and Y. Yu

May 8, 2007

Department of Informatics
Kyushu University
Fukuoka 819-0395, Japan

Email: baba@i.kyushu-u.ac.jp   Phone: +81-92-802-3787

A Parallel Computation for Shift-Add Algorithm

Kensuke Baba[*†] Hanmei E[‡] Yunqing Yu[§]

Abstract

The approximate string matching is useful in a wide area of applications such as biology. A practically significant speedup for solving this problem is obtained by representing strings as bit sequences and computing the comparisons of plural characters simultaneously by bit operations. In this method, a practical run-time depends on the word size of a computer. In this paper, as another parameter of the performance of a computer, the number of processors is considered. An efficient algorithm based on the previous speedup method is modified into a parallel algorithm. In the concrete, an $O(mn \log m/w)$ algorithm for a problem of approximate string matching is modified to an $O(mn/w)$ algorithm for a computer with m processors.

1 Introduction

The problem of string matching [3, 4] is to find all occurrences of a string (called a "pattern") in another string (called a "text"). The approximate string matching is defined as the string matching with some errors allowed. The approximate string matching is more useful in a wide area of applications, and its most general form (for example, the problem of weighted edit distance [9] and its extension [8]) is the essence of some interesting systems [7] for homology search in biology.

One of the most active areas for string processing is bit-parallelism [6]. The main idea of this approach is to represent strings as numbers (or bit sequences) and perform plural comparisons of characters simultaneously by arithmetic (or bit operations). Therefore, a practical run-time depends on the performance of a computer, and this idea can be found essentially in the Rabin-Karp algorithm [2]. As for the approximate string matching, we consider the match-count problem [5] in this paper. For this problem, a simple and efficient method based on bit-parallelism is introduced by Baeza-Yate and Gonnet [1], and it is called the "Shift-Add" method. While a naive algorithm based on character comparison requires $O(mn)$ comparisons for input strings of lengths m and n, an algorithm based on the Shift-Add method requires

[*]Faculty of Information Science and Electrical Engineering, Kyushu University
[†]System LSI Research Center, Kyushu University
[‡]Graduate School of Information Science and Electrical Engineering, Kyushu University
[§]Computing and Communications Center, Kyushu University

$O(mn \log m/w)$ bit operations, for the word size w of a computer. In this sense, the speedup by this approach depends on the performance of a computer.

In this paper, as another parameter of the performance of a computer, we consider the number of processors (or cores) of a computer. A simple method to solve the match-count problem for a computer with plural cores is to part a text or a pattern. However, in algorithms on the method, a text or a pattern have to be given completely before the computation, and hence the method cannot be applied to inputs as a streaming data. Then, we consider modifying the Shift-Add algorithm which processes a text on line into a parallel algorithm. The main idea of the modification is to convert each character in input strings into a single bit character rather than a bit sequence. Then, a straightforward parallelism with respect to the characters can be applied. As the result, we have an $O(m^2 n/w)$ algorithm which is constructed from m distinct processes. Therefore, if we consider an ideal computer with a k-core processor for $k \geq m$, the computing time of the algorithm is bounded by $O(mn/w)$.

2 Preliminaries

2.1 The Match-count Problem

Let Σ be a finite set of characters. For an integer $n > 0$, Σ^n denotes the set of the strings of length n over Σ. For a string s, $|s|$ denotes the length of s and s_i denotes the ith element of s for $1 \leq i \leq |s|$. The string $s_i s_{i+1} \cdots s_j$ is a *substring* of s, denoted by $s[i:j]$. In particular, it is called a *suffix* if $j = |s|$.

The *score vector* $C(t,p)$ between a text string $t \in \Sigma^n$ and a pattern string $p \in \Sigma^m$ (we assume $m < n$) is the vector whose ith element c_i is the number of matches between the substring $t[i : i+m-1]$ of the text and the pattern p for $1 \leq i \leq n-m+1$. Let δ be a function from $\Sigma \times \Sigma$ to $\{0,1\}$ such that, for $a, b \in \Sigma$, $\delta(a,b)$ is 1 if $a = b$, and 0 otherwise. Then, the ith element is

$$c_i = \sum_{j=1}^{m} \delta(t_{i+j-1}, p_j) \tag{1}$$

for $1 \leq i \leq n-m+1$. The *match-count problem* is to compute the score vector between two given strings.

2.2 A Computational Model

In the strict sense, the time complexity of an algorithm should be considered for a computer, therefore it is not correct that a performance of a computer is used as a parameter for the notation such as $O(mn/w)$ in the previous section. This problem can be solved straightforwardly, for example, by considering an abstract computer which has the following operations as a computer we use.

In this paper, we consider a computational model with a parameter w such that the following operations are computed respectively in an unit time:

- k sift-left operations to $u \in \{0,1\}^w$ for $k \leq w$,
- k sift-right operations to $u \in \{0,1\}^w$ for $k \leq w$,
- an and operation to $u, v \in \{0,1\}^w$,
- an add operation to $u, v \in \{0,1\}^w$,
- a comparison of $u, v \in \{0,1\}^w$,
- a reference with a parameter $u \in \{0,1\}^w$.

The previous operations are defined as the following functions.

- The *shift-left* operation and the *shift-right* operation are respectively the functions $S_l, S_r : \{0,1\}^w \to \{0,1\}^w$ such that $S_l(u) = u_2 u_3 \cdots u_w 0$ and $S_r(u) = 0 u_1 u_2 \cdots u_{w-1}$. The k shift-left (-right) operation is denoted by S_l^k (S_r^k).
- The *and* operation is the function $A : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$ such that $A(u,v) = w_1 w_2 \cdots w_m$ and, for $1 \leq i \leq w$, w_i is 1 if and only if $u_i = 1$ and $v_i = 1$.
- The *add* operation is the function $P : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$ such that $P(u,v)$ is the suffix of length w of the sum of binary digits u and v.
- The *comparison* is the function $M : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}$ such that $M(u,v)$ is 1 if and only if for $1 \leq i \leq w$ $u_i = v_i$.
- The *reference* is the function $R : \{0,1\}^w \to \{0,1\}^w$.

3 Standard Algorithms

3.1 Comparison-based Method

A naive method for the match-count problem is to compare the $m \times n$ pairs of characters in two given strings, that is, to compute the matrix $D(t,p)$ whose (i,j)-element $D_{i,j}$ is $\delta(p_i, t_j)$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. Then, the element c_k of the score vector is $\sum_{i=1}^{m} D_{i,i+k-1}$ for $1 \leq k \leq n-m+1$. For example, the score vector between two strings `acbabbaccb` and `abbac` is obtained by the 5×10 matrix in the following table. The list of $D_{i,i+k-1}$ for $1 \leq k \leq n-m+1$ is on a diagonal line.

	a	c	b	a	b	b	a	c	c	b	
a	1	0	0	1	0	0	1	0	0	0	
b	0	0	1	0	1	1	0	0	0	1	
b	0	0	1	0	1	1	0	0	0	1	
a	1	0	0	1	0	0	1	0	0	0	
c	0	1	0	0	0	0	0	1	1	0	
c_i						3	1	1	5	2	0

Therefore, if any character in the given strings is represented in b place for $b \leq w$ and $\log{(m+1)} \leq w$, the score vector $C(t,p)$ is obtained by $m \times (n-m+1)$ comparisons and $(m-1) \times (n-m+1)$ add operations. Therefore, the time complexity is $O(mn)$.

3.2 Bit-operation-based Method

Baeza-Yate and Gonnet introduced a simple and efficient method for some problems of string matching. The main idea of the method is to represent each state of the search as a bit sequence, and compute plural states simultaneously by some logical operations. In this method, plural elements of $D(t,p)$ are considered as a single bit sequence, then the sum with respect to a diagonal line is computed by shift operations and add operations.

$D(t,p)$ is computed by preparing the vectors $E(a)$ of bit characters for $a \in \Sigma$ whose ith element is $\delta(p_i, a)$ for $1 \leq i \leq m$. Since any element of $C(t,p)$ is at most m, each element of $D(t,p)$ is represented in $\lceil \log{(m+1)} \rceil$ place. For example, the $E(a)$'s in case where $p = \texttt{abbac}$ are given by the following table, where $x \in \Sigma$ is not appear in p.

	a	b	b	a	c
$E(\texttt{a})$	001	000	000	001	000
$E(\texttt{b})$	000	001	001	000	000
$E(\texttt{c})$	000	000	000	000	001
$E(x)$	000	000	000	000	000

Since we have only to consider at most $m+n$ characters in Σ and the number of the distinct characters in p is at most m, computing the $E(a)$'s requires $O(n+m^2)$ time.

The score vector is computed from the $E(a)$'s as Fig. 1. In this method, we can compute simultaneously at most $\lfloor w / \lceil \log{(m+1)} \rceil \rfloor$ elements of $D(t,p)$. Therefore, the time complexity is bounded by $O(mn \log m / w)$. The outline of an algorithm based on this method is shown in Fig. 2. A single step of the computation of D in the last **for**-sentence requires $\lceil m \log{(m+1)}/w \rceil$.

							c_{i-4}
$D := 0$	000	000	000	000	000		
$E := E(\mathtt{a})$	001	000	000	001	000		
$D := P(D, E)$	001	000	000	001	000	$\rightarrow$	000
$D := S_r^3(D)$	000	001	000	000	001		
$E := E(\mathtt{c})$	000	000	000	000	001		
$D := P(D, E)$	000	001	000	000	010	$\rightarrow$	010
$\vdots$			$\vdots$				$\vdots$

Figure 1: A computation in an algorithm based on the bit-operation-based method.

Procedure Shift-Add
Input: $t = t_1 t_2 \cdots t_n$, $p = p_1 p_2 \cdots p_m$
Output: $C(t, p) = (c_1, c_2, \ldots, c_{n-m+1})$

$b := \lceil \log{(m+1)} \rceil$;

for $1 \le i \le n$ **do** $E[t_i] := 0$;
for $1 \le i \le m$ **do** {
    $E[p_i] := 0$;
    for $1 \le j \le m$ **do** $E[p_i] := P(S_l^b(E[p_i]), M(p_i, p_j))$;
}

$D := 0$;
for $1 \le i \le n$ **do** {
    $D := P(S_r^b(D), E[t_i])$;
    $c_{i-m+1} := A(D, 1^b)$;
}

Figure 2: The Shift-Add algorithm for the match-count problem, where 1^b is the string constructed by b unities.

4 Parallel Algorithm

4.1 A Simple Parallel Computation

The most straightforward methods of parallel computation for the match-count problem is to part t or p into substrings. Intuitively, in this method, using k computers (processors, or cores) yields k times speedup.

Clearly, from $t[i:j]$ and p, we obtain from the ith element to the $(j-m+1)$th element of $C(t,p)$. Therefore, by parting t into k substrings with overlaps of length $m-1$ and combining the results, $C(t,p)$ is obtained by k distinct computations. If we part p, the following is clear in general. Let c_i^p be the ith element of the score vector $C(t,p)$ and c_i^q the ith element of $C(t,q)$. Then, the ith element of $C(t,pq)$ is

$c_i^p + c_{m+i}^q$, where m is the length of p. Therefore, we can also expect straightforward speedup except for the overhead.

4.2 Modification of the Shift-Add Algorithm

The parallel computation in the previous subsection is simple and efficient, however algorithms on the method require some overheads. Moreover, we cannot part t if t is not given completely before the computation. The Shift-Add algorithm allows that t is a streaming data. Then, we consider modifying the algorithm to a parallel algorithm with preserving the previous property.

The essential idea of this modification is to convert t and p into bit sequences with respect to each character in Σ. In this method, using k computers yields $\lfloor \log k \rfloor$ times speedup. We consider the function $f : \Sigma^n \times \Sigma \to \{0,1\}^n$ for $n > 0$ such that $f(s,a) = \delta(s_1,a)\delta(s_2,a)\cdots\delta(s_n,a)$. By Eq. 1, clearly we have $c_i = \sum_{j=1}^{m}\sum_{a\in\Sigma} f(t_{i+j-1},a)\cdot f(p_j,a)$ for $1 \le i \le n-m+1$. Moreover, we can avoid some computations, since $f(t_{i+j-1},a)\cdot f(p_j,a) = 0$ for any a which is not appear in p. Let Σ_p be the set of the characters in p. Then, we have the following equation. The ith element of the score vector between t and p is

$$c_i = \sum_{a\in\Sigma_p}\sum_{j=1}^{m} f(t_{i+j-1},a)\cdot f(p_j,a) \tag{2}$$

for $1 \le i \le n-m+1$.

Let R be the function from $\{0,1\}^w$ to $\{1,2,\ldots,w\}$ such that $R(u)$ is the number of 1 in u. Then, the outline of the modified algorithm is shown in Fig. 3.

Theorem 1 *The Parallel Shift-Add algorithm solves the match-count problem for $t \in \Sigma^n$ and $p \in \Sigma^m$ in $O(m^2n/w)$ time, where w is a parameter for the computer we use.*

By Eq. 2, it is clear that the Parallel Shift-Add algorithm solves the match-count problem. Since the length of ft and fp is m, each computation of ft, fp, and c_i requires m/w time. The cardinality of Σ_p is at most m. Therefore, the time complexity is $O(n-m+1)+O(m)\times(O(m\times m/w)+O((n-m+1)\times m/w)) = O(m^2n/w)$. Since this algorithm can be operated simultaneously by m computers, the computation time is expected to be bounded by $O(mn/w)$.

5 Conclusion

We modified the Shift-Add algorithm which processes a text on line into a parallel algorithm. The main idea of the modification is to convert each character in input strings into a single bit character. Then, a straightforward parallelism with respect to the characters can be applied. As the result, we obtained an $O(m^2n/w)$ algorithm

Procedure Parallel Shift-Add
Input: $t = t_1 t_2 \cdots t_n$, $p = p_1 p_2 \cdots p_m$
Output: $C(t, p) = (c_1, c_2, \ldots, c_{n-m+1})$

for $1 \leq i \leq n - m + 1$ **do** $c_i := 0$;
for $a \in \Sigma_p$ **do** {
    $ft := 0$, $fp := 0$;
    for $1 \leq i \leq m$ **do** {
        $ft := P(S_l(ft), M(t_i, a))$;
        $fp := P(S_l(fp), M(p_i, a))$;
    }

    for $1 \leq i \leq n - m + 1$ **do** {
        $c_i := P(c_i, R(A(ft, fp)))$;
        $ft := P(S_l(ft), M(t_{m+i}, a))$;
    }
}

Figure 3: The Parallel Shift-Add algorithm for the match-count problem.

which is constructed from m distinct processes. If we consider an ideal computer with a k-core processor for $k \geq m$, the computing time of the algorithm is bounded by $O(mn/w)$.

Acknowledgment

This work has been supported by the Grant-in-Aid for Scientific Research No. 17700020 of the Ministry of Education, Culture, Sports, Science and Technology (MEXT) from 2005 to 2007.

References

[1] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, 1992.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms, Second Edition*. MIT Press, 2001.

[3] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[4] M. Crochemore and W. Rytter. *Jewels of Stringology*. World Scientific, 2003.

[5] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[6] G. Navarro. A guided tuor to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[7] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proc. Natl. Acad. Sci. USA*, volume 85, pages 2444–2448, April 1988.

[8] T. F. Smith and M. S. Waterman. Identification of common molecular subseqences. *J. Mol. Biol.*, 147:195–197, 1981.

[9] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.