텍스트의 핑거프린트를 이용한 순위다중패턴매칭 알고리즘 병렬 구현

박소민, 김영호, 심정섭†
인하대학교 컴퓨터공학과
smpark95@inha.edu, yhkim8505@gmail.com, jssim@inha.ac.kr

A Parallel Implementation of the Order-Preserving Multiple Pattern Matching Algorithm using Fingerprints of Texts

Somin Park*, Youngho Kim*, Jeong Seop Sim*
Department of Computer Engineering, Inha University

요 약

순위다중패턴매칭문제는 길이가 n인 텍스트 T와 패턴들의 집합 $P' = \{P_1, P_2, \ldots, P_k\}$가 주어졌을 때, P'에 속하는 패턴들과 상대적인 순위가 일치하는 T의 모든 부분문자열들의 위치를 찾는 문제이다. P'에서 가장 짧은 패턴의 길이가 m, 가장 긴 패턴의 길이를 $\overline{m}$, 모든 패턴들의 길이의 합을 M, q개의 연속된 문자들을 q-그램이라 할 때, 기존에 텍스트의 핑거프린트를 이용하여 순위다중패턴매칭문제를 $O(q! + nq\log q + M\log\overline{m} + nM)$ 시간에 해결하는 알고리즘이 제시되었다. 본 논문에서는 텍스트의 핑거프린트를 활용하여 $O(\max(q!, M, n))$개의 스레드를 이용하여 순위다중패턴매칭문제를 평균적으로 $O(\overline{m} + q\log q + n/q!)$ 시간에 해결하는 병렬 구현 방법을 제시한다. 실험 결과, $n = 1{,}000{,}000$, $k = 1{,}000$, $m = 5$, $q = 3$일 때, 본 논문에서 제시하는 병렬 구현 방법은 기존의 순차 알고리즘보다 약 19.8배 빠르게 수행되었다.

1. 서론

두 문자열 x, $y(|x| = |y|)$의 같은 위치에 있는 문자의 상대적인 순위가 모두 같으면 x, y는 순위동형이라 한다. 예를 들어, $x = (10, 15, 13)$, $y = (3, 20, 17)$일 때, x와 y는 각 문자의 순위가 $(1, 3, 2)$로 같으므로 순위동형이다. 순위패턴매칭문제는 텍스트 T와 패턴 P가 주어졌을 때, P와 순위동형인 T의 모든 부분문자열들의 위치를 찾는 문제이다. 이는 주가지수, 음악 멜로디 등 시계열 데이터 분석에 활용될 수 있다[1].

순위다중패턴매칭문제는 텍스트 $T(|T| = n)$와 패턴들의 집합 $P' = \{P_1, P_2, \ldots, P_k\}$가 주어졌을 때, 패턴 $P_a(1 \le a \le k)$와 순위동형인 T의 모든 부분문자열들의 위치를 찾는 문제이다. P'에서 가장 짧은 패턴의 길이를 m, 가장 긴 패턴의 길이를 $\overline{m}$, 모든 패턴들의 길이의 합을 M, q개의 연속된 문자들을 q-그램이라 하자. [1]에서는 Aho-Corasick 오토마톤[2]을 이용한 $O((M+n)\log\overline{m})$-시간 알고리즘을 제시하였다. [3]에서는 Wu-Manber 알고리즘[4]을 이용한 알고리즘 1과 Karp-Rabin 알고리즘[5]을 이용한 알고리즘 2를 제시하였다. 알고리즘 1은 평균적으로 $O((n/m)\log M)$ 시간에 탐색을 수행한다. 알고리즘 2는 M이 m에 대한 다항식일 때 평균적으로 $O(n)$ 시간에 탐색을 수행한다. [6]에서는 T의 핑거프린트를 이용하여 전처리단계를 $O(q! + nq\log q)$ 시간, 탐색단계를 $O(M\log\overline{m} + nM)$ 시간에 수행하는 알고리즘을 제시하였다.

GPU의 성능이 개선되면서 GPU를 이용한 순위다중패턴매칭문제에 관한 연구가 진행되고 있다. [7]에서는 Z-함수를 이용하여 $O(k(n+\overline{m}))$개의 스레드를 사용하여 $O(n+\overline{m})$ 시간에 순위다중패턴매칭문제를 해결하는 병렬계산 방법을 제시하였다. [8]에서는 [1]에서 제시된 순위다중패턴매칭 알고리즘에 대해 $O(n/\overline{m})$개

* 이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2017R1E1A1A03070867).
* 이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (2020-0-01389, 인공지능융합연구센터지원(인하대학교)).
† 교신저자

의 스레드를 사용하여 $O(\overline{m}\log\overline{m})$ 시간에 수행하는 병렬 구현 방법을 제시하였다.

본 논문에서는 [6]에서 제시된 순위다중패턴매칭알고리즘을 병렬적으로 수행하는 방법을 제시한다. 이는 전처리단계를 $O(\max(q!, n))$개의 스레드를 이용하여 평균적으로 $O(q\log q + n/q!)$ 시간, 탐색단계를 $O(M)$ 개의 스레드를 이용하여 평균적으로 $O(\overline{m} + q\log q + n/q!)$ 시간에 수행한다. 무작위로 생성한 문자열에 대해 실험한 결과, $n = 1{,}000{,}000$, $k = 1{,}000$, $m = 5$, $q = 3$일 때, 본 논문에서 제시하는 병렬 구현 방법은 기존의 순차 알고리즘보다 약 19.8배 빠르게 수행되었다.

본 논문의 구성은 다음과 같다. 2장에서는 용어 정의와 관련 연구를 소개하고, 3장에서는 텍스트의 핑거프린트를 이용한 순위다중패턴매칭 병렬 구현 방법을 설명한다. 4장에서는 실험을 통해 [6]에서 제시한 알고리즘과 본 논문에서 제시한 알고리즘의 수행시간을 비교한다.

2. 관련 연구

2.1 순위동형과 핑거프린트

패턴집합 $P' = \{P_1, P_2, \ldots, P_k\}$에서 가장 짧은 패턴의 길이를 m, 가장 긴 패턴의 길이를 $\overline{m}$, 모든 패턴들의 길이의 합을 M으로 표기한다. 편의상, 문자열은 서로 다른 문자들로 구성된다고 가정한다. 길이가 m인 두 문자열 x, y에 대해 $x[i] < x[j] \Leftrightarrow y[i] < y[j]$ $(0 \le i, j < m)$이면 x와 y는 순위동형이고, $x \approx y$로 표기한다[1,9].

문자열의 순위관계는 접두사표현과 최근접이웃표현을 이용하여 표현한다. x의 접두사표현에 사용되는 접두사테이블 μ_x는 다음과 같이 정의된다[1].

$$\mu_x[i] = |\{j : x[j] < x[i], 0 \le j < i\}|$$

즉, $\mu_x[i]$는 $x[0..i-1]$에서 $x[i]$보다 작은 문자의 개수이다. 만약 $\mu_x = \mu_y$이면 $x \approx y$이다[1].

x의 최근접이웃표현은 위치테이블 $LMax_x$, $LMin_x$으로 나타내고 다음과 같이 정의한다[1,9].

$$LMax_x[i] = j \text{ if } x[j] = \max\{x[k] : k \in [0, i-1], x[k] < x[i]\}$$

$$LMin_x[i] = j \text{ if } x[j] = \min\{x[k] : k \in [0, i-1], x[k] > x[i]\}$$

즉, $LMax_x[i]$는 $x[0..i-1]$에서 $x[i]$보다 작은 문자 중 가장 큰 문자의 위치 $j(0 \le j < i)$를, $LMin_x[i]$는 $x[0..i-1]$에서 $x[i]$보다 큰 문자 중 가장 작은 문자의 위치 j를 저장한다. 그러한 j가 없다면 -1을 저장한다.

표 1. x의 접두사테이블과 위치테이블

i	0	1	2	3	4
x	11	10	7	4	9
μ_x	0	0	0	0	2
$LMax_x$	-1	-1	-1	-1	2
$LMin_x$	-1	0	1	2	1

다. $y[LMax_x[i]] < y[i] < y[LMin_x[i]]$ $(1 \le i < m)$이면, $x \approx y$이다[1,9]. 예를 들어, 표 1은 $x = (11, 10, 7, 4, 9)$일 때, x의 접두사테이블과 위치테이블을 보여준다. 접두사테이블과 위치테이블은 순위통계트리를 이용하여 $O(m\log m)$ 시간에 계산할 수 있다[1].

[10]에서는 Horspool 알고리즘[11]을 순위패턴매칭 문제에 적용하기 위해, q-그램과 이를 범위 [0, $q!-1$] 내의 유일한 정수로 변환시키는 핑거프린트를 활용하였다. q-그램 x에 대한 핑거프린트는 다음과 같이 계산한다.

$$f(x) = \sum_{k=0}^{q-1} \mu_x[k] \times k!$$

예를 들어, q-그램 $x = (1, 5, 8)$이라면, x의 핑거프린트 $f(x) = (0 \times 0!) + (1 \times 1!) + (2 \times 2!) = 5$이다.

2.2 텍스트의 핑거프린트를 이용한 순위다중패턴매칭 알고리즘

[6]에서 제시된 알고리즘은 전처리단계와 탐색단계로 구성된다. 전처리단계에서는 T의 부분문자열들의 q-그램의 핑거프린트를 계산하여 T의 핑거프린트테이블을 생성한다. 탐색단계에서는 패턴 $P_a(1 \le a \le k)$와 q-그램의 핑거프린트가 일치하는 T의 부분문자열만 순위동형을 검증하여 순위다중패턴매칭문제를 해결한다.

전처리단계에서 크기가 $O(q! + n)$인 핑거프린트테이블 FT는 연결리스트들의 배열로 구성된 자료구조이다. FT의 인덱스 범위는 [0, $q!-1$]이다. FT의 생성 알고리즘은 $n-m+1$개의 스텝으로 구성된다. 즉, T의 오른쪽부터 왼쪽으로 진행하면서, 길이가 m인 T의 각 부분문자열에 대해 가장 오른쪽에 위치한 q-그램의 핑거프린트 α를 계산하고 $FT[\alpha]$의 연결리스트의 끝에 T의 부분문자열의 시작위치를 삽입한다. 즉, $FT[\alpha]$ $(0 \le \alpha \le q!-1)$에는 q-그램의 핑거프린트가 동일한 T의 부분문자열들의 시작위치들이 내림차순으로 정렬된 형태로 저장된다. 그림 1은 텍스트 $T = (10, 5, 21, 24, 30, 35)$, $m = 5$, $q = 3$에 대한 핑거

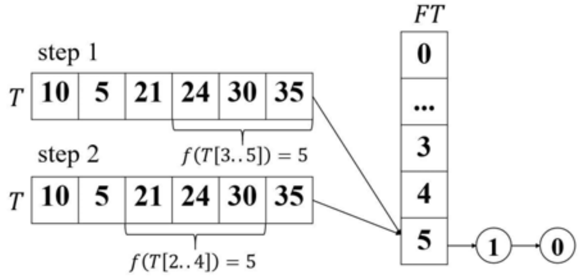

그림 1. 핑거프린트테이블 생성과정$(m=5, q=3)$

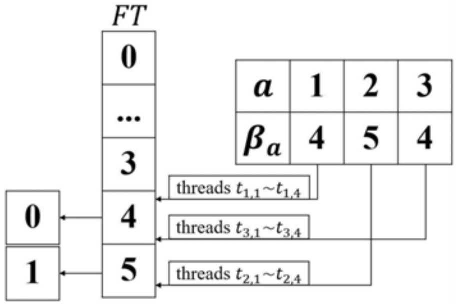

그림 2. 핑거프린트테이블을 이용한 순위다중패턴매칭 알고리즘 병렬 구현$(m=5, q=3)$

프린트테이블 FT를 보여준다.

FT를 초기화하는데 $O(q!)$ 시간, 모든 T의 부분문자열의 q-그램의 핑거프린트를 계산하고 FT에 삽입하는데 $O(nq\log q)$ 시간이 소요된다. 따라서 전처리단계는 $O(q!+nq\log q)$ 시간에 수행된다.

탐색단계는 k개의 패턴에 대해 순차적으로 수행한다. 순위동형 검증횟수를 줄이기 위해 $P_a(1 \le a \le k)$의 1, 2차 q-그램인 $P_a[m-q, m-1]$, $P_a[m-q-1, m-2]$의 핑거프린트, P_a의 위치테이블을 계산한 후, FT와 위치테이블을 이용하여 P_a와 q-그램의 핑거프린트가 일치하는 T의 부분문자열을 모두 탐색한다.

모든 패턴들의 위치테이블 계산, 1, 2차 q-그램의 핑거프린트 계산, 순위동형 검증에 각각 $O(M\log\overline{m})$, $O(kq\log q)$, $O(nM)$ 시간이 소요된다. 따라서 탐색단계는 $O(M\log\overline{m}+nM)$ 시간에 수행된다.

3. 텍스트의 핑거프린트를 이용한 순위다중패턴매칭 알고리즘 병렬 구현

본 논문에서는 [6]에서 제시한 알고리즘을 병렬적으로 수행하는 방법을 제시한다. 전처리단계에서는 T의 핑거프린트테이블 FT를 병렬적으로 생성한다. 탐색단계에서는 모든 패턴들의 q-그램의 핑거프린트와 위치테이블을 병렬 계산하고, FT를 활용하여 패턴들과 순위동형인 T의 부분문자열들을 병렬적으로 탐색한다.

전처리단계에서는 먼저 $O(q!)$개의 스레드를 이용하여 상수시간에 FT를 초기화한다. 이후 $O(n)$개의 스레드를 이용하여 FT를 병렬적으로 생성한다. 구체적으로, 각 스레드 $t(0 \le t < n-m+1)$가 $T[t..t+m-1]$의 가장 오른쪽에 위치한 q-그램의 핑거프린트 α_t $(0 \le \alpha_t < q!)$를 계산하고, $FT[\alpha_t]$의 연결리스트의 끝에 $T[t..t+m-1]$의 시작위치 t를 순차적으로 삽입한다. 모든 α_t는 $O(q\log q)$ 시간에 계산할 수 있다. 최악의 경우, q-그램의 핑거프린트 α_t가 모두 동일하여 FT가 하나의 연결리스트로만 구성될 수 있다. 이때 FT는 $O(q\log q+n)$ 시간에 생성된다. 만약 각 α_t가 발생할 확률이 모두 동일하다고 가정하면, FT는 평균적으로 $O(q\log q+n/q!)$ 시간에 생성할 수 있다. 따라서 전처리단계는 $O(\max(q!, n))$개의 스레드를 이용하여 평균적으로 $O(q\log q+n/q!)$ 시간에 수행된다.

탐색단계에서는 $O(M)$개의 스레드를 이용하며, $P'=\{P_1, P_2, \ldots, P_k\}$의 위치테이블은 다음과 같이 계산한다. 스레드 $t_{a,j}(1 \le a \le k, 0 \le j < |P_a|)$는 먼저 $LMax_{P_a}[j]$, $LMin_{P_a}[j]$를 -1로 초기화하고, $P_a[0..j-1]$을 스캔하여 $LMax_{P_a}[j]$, $LMin_{P_a}[j]$를 병렬적으로 $O(\overline{m})$ 시간에 계산한다. $P_a[m-q, m-1]$ $(1 \le a \le k)$의 핑거프린트 β_a도 병렬적으로 $O(q\log q)$ 시간에 계산한다. 이후 스레드 $t_{a,j}$는 각 P_a와 q-그램의 핑거프린트가 일치하는 T의 부분문자열을 $FT[\beta_a]$에서 순차적으로 탐색한다. $FT[\beta_a]$에 저장된 노드가 i인 경우, P_a와 $T[i..i+|P_a|-1]$의 순위동형 여부는 $O(1)$ 시간에 검증할 수 있다. 만약 순위동형이면, 위치 i를 출력한다. 예를 들어, $q=3$, $T=(30, 25, 5, 3, 9, 20)$, $P_1=(11,10,7,4,9)$, $P_2=(1,2,4,6,8)$, $P_3=(10,20,9,5,15)$이 주어졌을 때, $f(P_1[2..4])=4$, $f(P_2[2..4])=5$, $f(P_3[2..4])=4$이다 (그림 2 참조). P_1, P_3은 $T[0..4]$와 순위동형을 검증하고, P_2는 $T[1..5]$와 순위동형을 검증한다. $P_1 \approx T[0..4]$이므로 위치 0을 출력한다.

최악의 경우, P_a와 순위동형인 T의 부분문자열의 탐색은 $O(n)$ 시간이 소요된다. 만약 각 $FT[\beta_a]$에 저장된 연결리스트의 크기가 모두 균등하다고 가정하면, P_a와 순위동형인 T의 부분문자열의 탐색은 평균적으로 $O(n/q!)$ 시간에 수행할 수 있다. 따라서 탐색단계는 평균적으로 $O(\overline{m}+q\log q+n/q!)$ 시간에 수행된다.

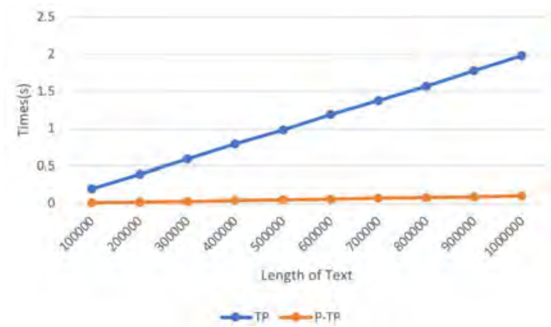

그림 3. $k=1{,}000$, $m=5$, $q=3$일 때, n에 따른 TP와 $P-TP$의 수행시간

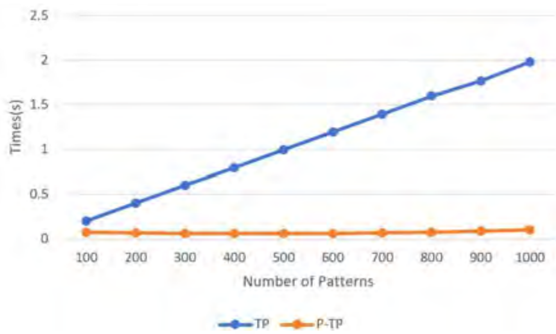

그림 4. $n=1{,}000{,}000$, $m=5$, $q=3$일 때, k에 따른 TP와 $P-TP$의 수행시간

그러므로 본 논문에서 제시하는 병렬 구현 방법은 $O(\max(q!, n, M))$개의 스레드를 이용하여 평균적으로 $O(\overline{m}+q\log q+n/q!)$ 시간에 순위다중패턴매칭문제를 해결할 수 있다.

4. 실험 결과

실험 환경은 다음과 같다. OS는 Windows 10(64bit), CPU는 AMD Ryzen 9 3950X, RAM은 64GB, GPU는 GeForce RTX 2080Ti, 개발 툴은 Visual Studio 2019, CUDA SDK 11.0, 개발 언어는 C++, CUDA이다. 텍스트 T와 패턴집합 P'은 범위 $[1, 2^{30}]$인 정수로 무작위로 생성하였다. 편의상, 패턴들의 길이는 모두 동일하게 실험하였다. n은 100,000부터 100,000씩 증가하여 1,000,000까지, 패턴의 개수 k는 100부터 100씩 증가하여 1,000까지, 패턴의 길이 m은 5, q는 3에 대해 실험하였다. 편의상 [6]에서 제시된 알고리즘을 TP, 본 논문에서 제시한 알고리즘을 $P-TP$로 표기한다.

그림 3은 $k=1{,}000$, $m=5$, $q=3$일 때, n에 따른 TP와 $P-TP$의 수행시간을 보여준다. 그림 4는 $n=1{,}000{,}000$, $m=5$, $q=3$일 때, k에 따른 TP와 $P-TP$의 수행시간을 보여준다. n이나 k가 증가할수록 TP의 수행시간은 선형적으로 증가하였으나, $P-TP$의 수행시간은 변화가 미비했다. $n=1{,}000{,}000$, $k=1{,}000$, $m=5$, $q=3$일 때, TP와 $P-TP$의 수행시간은 각각 약 1.98초, 0.1초로, $P-TP$는 TP보다 약 19.8배 빠르게 수행되었다.

참고문헌

[1] J. Kim, P. Eades, R. Fleischer, S. H. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, and T. Tokuyama, “Order-preserving matching,” Theoretical Computer Science, Vol. 525, pp. 68-79, 2014.

[2] A.V. Aho and M.J. Corasick, “Efficient string matching: An aid to bibliographic search,” Communications of the ACM, Vol. 18, No. 6, pp. 333-340, 1975.

[3] M. Han, M. Kang, S. Cho, G. Gu, J.S. Sim, K. Park, “Fast Multiple Order-Preserving Matching Algorithms,” IWOCA, LNCS 9538, pp. 248-259, 2015.

[4] U. Manber, S. Wu, "A fast algorithm for multi-pattern searching", Tech. Report TR-94-17 CS Dept. University of Arizona, 1994.

[5] R. M. Karp, M. O. Rabin, “Efficient randomized pattern-matching algorithms,” IBM journal of research and development, Vol. 31, No. 2, pp. 249-260, 1987.

[6] 유광모, 심정섭, “텍스트의 핑거프린트를 이용한 순위다중패턴매칭 알고리즘,” 한국정보과학회 학술발표논문집, pp. 1214-1216, 2018.

[7] 신유건, 김영호, 심정섭, “Z-함수를 이용한 순위패턴매칭과 순위다중패턴매칭 병렬계산,” 정보과학회논문지, Vol. 45, No. 8, pp. 778-785, 2018.

[8] 박소민, 김영호, 심정섭, “Aho-Corasick 오토마타를 이용한 순위다중패턴매칭 알고리즘 병렬화,” 한국정보과학회 학술발표논문집, pp. 1253-1255, 2020.

[9] M. Kubica, T. Kulczynski, J. Radoszewski, W. Rytter, T. Walen, “A leaner time algorithm for consecutive permutation pattern matching”, Information Processing Letters, Vol. 113, pp. 430-433, 2013.

[10] S. Cho, J. Na, K. Park, J. S. Sim, “A fast algorithm for order-preserving pattern matching”, Information Processing Letters, Vol. 115, pp. 397-402, 2015.

[11] R.N. Horspool, “Practical Fast Searching in Strings”, Software Practice Exper. 10, 501-506, 1980.