해싱기반 순위다중패턴매칭 알고리즘의 검색단계 병렬 계산

박진혁○, 김영호, 심정섭†
인하대학교 컴퓨터공학과
jhpark1015@inha.edu, yhkim8505@gmail.com, jssim@inha.ac.kr

Parallel Computation of the Searching Step of the Hashing-Based Order-Preserving Multiple Pattern Matching Algorithm

Jinhyeok Park○, Youngho Kim, Jeong Seop Sim†
Department of Computer Engineering, Inha University

요 약

순위다중패턴매칭문제는 텍스트 $T(|T|=n)$와 k개의 패턴들로 구성된 패턴집합 $\hat{P}$이 주어질 때, $\hat{P}$에 속한 패턴들과 상대적인 순위가 같은 T의 모든 부분문자열을 찾는 문제이다. q-그램의 길이를 q, $\hat{P}$에 속한 패턴들의 총 길이의 합은 M, 가장 짧은 패턴의 길이는 m, 가장 긴 패턴의 길이 $\overline{m}$라 할 때, 기존에 이를 해결하는 공간 효율적인 알고리즘이 제시되었으며, 이 알고리즘의 검색단계는 $O((q\log q+M)n)$ 시간에 수행된다. 본 논문에서는 순위다중패턴매칭문제를 위해 기존의 공간효율적인 알고리즘의 검색단계를 k개의 스레드를 사용하여 $O((q\log q+\overline{m})n)$ 시간에 병렬적으로 계산하는 방법을 제시한다. 무작위로 생성한 문자열에 대해 검색단계를 실험한 결과, 본 논문에서 제시하는 병렬알고리즘은 기존의 순차알고리즘보다 $m=10, n=100{,}000, k=10{,}000$일 때 약 6.2배 빠르게 수행되었다.

1. 서 론

순위다중패턴매칭문제는 $T(|T|=n)$와 패턴집합 $\hat{P}=\{P_1,P_2,\cdots,P_k\}$가 주어졌을 때, $P_b(1\le b\le k)$와 순위동형인 T의 모든 부분문자열들을 찾는 문제이다. 이때 두 문자열의 순위동형(order-isomorphism)은 길이가 같고 서로 같은 위치에 있는 문자들의 순위가 동일한 것을 의미한다. 예를 들어, 길이가 같은 두 문자열 $x=(4,7,13,2,9)$, $y=(3,5,14,1,6)$는 문자들의 순위가 동일하게 $(2,3,5,1,4)$이므로 순위동형이다. 이 문제는 주가 변동 분석, 음악멜로디 분석 등의 시계열 데이터 분석에 활용될 수 있다[1].

순위다중패턴매칭문제를 해결하는 다양한 알고리즘이 연구되고 있다. $\hat{P}$에 속한 패턴들의 길이의 합을 M, 가장 짧은 패턴의 길이를 m, 가장 긴 패턴의 길이를 $\overline{m}$라 할 때, [1]에서는 Aho-Corasick 알고리즘[2]을 이용하여 문제를 $O(n\log M)$ 시간에 해결하는 알고리즘을 제시하였다. [3]에서는 Wu-Manber 알고리즘[4]을 이용하여 패턴집합에 대한 이동테이블, 해시테이블, 위치테이블을 $O(q!+k+M)$ 공간을 이용하여 $O(q!+kmq\log q+M\log\overline{m})$ 시간에 생성하고, 검색단계를 평균적으로 $O((n/m)\log M)$ 시간에 수행하는 알고리즘 1과 (q는 q-그램의 길이) Karp-Rabin 알고리즘[5]을 이용하여 M이 m에 대한 다항식일 때 검색단계를 평균적으로 $O(n)$ 시간에 수행하는 알고리즘 2를 제시하였다. 한편, [3]의 알고리즘 1의 검색단계는 최악수행시간이 $O((q\log q+M)n)$이다. [6]에서는 [3]의 알고리즘 1의 사용 공간을 개선하기 위해 이동테이블과 해시테이블 대신 핑거프린트테이블을 사용한 알고리즘을 제시하였다. 즉, 핑거프린트테이블과 위치테이블을 $O(k+M)$ 공간을 이용하여 $O(kq\log q+M\log\overline{m})$ 시간에 생성하고 검색단계를 $O((q\log q+M)n)$ 시간에 수행한다.

GPU의 성능이 향상됨에 따라 GPU를 이용한 병렬화 연구가 진행되고 있다. [7]에서는 순위다중패턴매칭을 위한 이동테이블을 $O(mq\log q)$ 시간에 병렬적으로 계산하는 방법을 제시하였다. [8]에서는 순위다중패턴매칭문제를 $O(k(n+\overline{m}))$개의 스레드를 이용하여 $O(n+\overline{m})$ 시간에 해결하는 병렬 계산 방법을 제시하였다. [9]에서는 사각망 순열패턴매칭문제를 $O(mn)$개의 스레드를 사용하여 $O(n)$ 시간에 해결하는 병렬알고리즘을 제시하였다.

본 논문에서는 [6]의 검색단계를 k개의 스레드를 이용하여 $O((q\log q+\overline{m})n)$ 시간에 병렬적으로 계산하는 방법을 제시한다. 무작위로 생성한 문자열에 대해 검색단계를 실험한 결과, $m=10, n=100{,}000, k=10{,}000$일 때 본 논문에서 제시하는 병렬알고리즘은 [6]의 순차알고리즘보다 약 6.2배 빠르게 수행하였다. 단, $k=1{,}000$일 때 병렬알고리즘이 순차알고리즘보다 느리게 수행되었다.

본 논문의 구성은 다음과 같다. 2장에서 용어를 정의하고 관련 연구를 설명한다. 3장에서는 본 논문에서 제시하는 해싱기반 순위다중패턴매칭 알고리즘의 검색단계를 병렬적으로 계산하는 알고리즘을 설명한다. 4장에서는 알고리즘의 검색단계에 대한 실험을 통해 [6]에서 제시한 알고리즘과 본 논문에서 제시하는 병렬알고리즘의 수행시간을 비교한다.

2. 관련 연구

2.1 문자열의 순위표현과 순위동형

패턴집합 $\hat{P}=\{P_1, P_2, \ldots, P_k\}$로 나타내며 모든 패턴들의 길이의 합을 M, 가장 짧은 패턴의 길이를 m, 가장 긴 패턴의 길이를 $\overline{m}$로 표기한다. $\hat{P}'=\{P_1', P_2', \cdots, P_k'\}$은 $\hat{P}$의 모든 패턴들에 대해 길이 m의 접두사들로 구성된 패턴집합을 나타낸다. 문자열은 서로 다른 문자들로 구성된다고 가정한다.

* 이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2017R1E1A1A03070867)
* 이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 한국연구재단 포스트게놈다부처유전체사업의 지원을 받아 수행된 연구임(NRF-2014M3C9A3064706)
† 교신저자

문자열의 순위를 나타내는 방법으로 접두사표현(prefix representation)이 있다. 문자열 x의 접두사표현을 나타내는 접두사테이블 $\mu(x)$는 다음과 같이 정의된다.

$$\mu(x)[i] = |j : x[j] < x[i] \quad \forall 1 \le j < i|$$

즉, $\mu(x)[i]$는 $x[1..i-1]$의 문자들 중 $x[i]$보다 작은 문자들의 개수를 저장한다. 예를 들어, $x=(4,7,13,2,9)$일 때 $\mu(x)=(0,1,2,0,3)$이다. 두 문자열 x, y가 순위동형이면 $\mu(x)=\mu(y)$이다.

한편, 문자열의 접두사표현에서 하나의 문자에 대한 순위는 항상 0이기 때문에, 순위다중패턴매칭문제에 Boyer-Moore-Horspool 알고리즘[10]의 오문자규칙을 바로 적용할 수 없다. 이를 위해 [3,11]에서는 길이 q의 문자열을 하나의 단위로 취급하는 q-그램(q-gram)과 q-그램 x를 범위 $[1,q!]$내의 유일한 정수로 변환하는 핑거프린트(fingerprint) 함수를 이용하여 Boyer-Moore-Horspool 알고리즘의 오문자규칙을 적용한다. q-그램 x의 핑거프린트 $f(x)$는 다음과 같이 정의한다.

$$f(x) = \sum_{k=1}^{q} [\mu(x)[k] \cdot (k-1)!] + 1$$

예를 들어, $x=(8,10,6)$일 때, $f(x)=2$이다. $f(x)$는 순위통계트리 (order-statistics tree)를 이용하여 $O(q\log q)$ 시간에 계산할 수 있다[3,11].

2.2 공간효율적인 순위다중패턴매칭 알고리즘

[6]의 순위다중패턴매칭 알고리즘은 전처리단계와 검색단계로 구성된다. 전처리단계에서는 T의 부분문자열과 순위동형일 수 있는 후보패턴들을 찾기 위해 $P_b'(1 \le b \le k)$의 가장 오른쪽 q-그램의 핑거프린트를 저장하는 핑거프린트테이블(fingerprint table) $FP[b](1 \le b \le k)$와 순위동형 검증을 위한 $\hat{P}$의 위치테이블 POS를 생성한다. 검색단계에서는 FP와 POS를 이용하여 $P_b(1 \le b \le k)$와 순위동형인 T의 모든 부분문자열들을 탐색한다.

구체적으로, 전처리단계에서 $FP[b]=f(P_b'[m-q+1..m])$ $(1 \le b \le k)$로 계산한다. 따라서 FP는 $O(k)$ 공간을 사용하여 $O(kq\log q)$ 시간에 계산할 수 있다. POS는 다음과 같이 계산한다. P_b의 순위를 저장한 배열을 $\pi(P_b)$라 하자. $\pi(P_b)$는 P_b를 정렬하여 $O(|P_b|\log|P_b|)$ 시간에 계산할 수 있다[6,12]. $\pi(P_b)[i]=j$라 할 때, $POS[b][j]=i$이다. 따라서 $\pi(P_b)$가 주어졌을때 $POS[b]$는 $O(|P_b|)$ 시간에 계산될 수 있다. POS는 모든 패턴들에 대해 계산하므로 $O(M)$ 공간을 이용하여 $O(M\log\overline{m})$ 시간에 계산할 수 있다. 따라서 전처리단계는 총 $O(kq\log q+M\log\overline{m})$ 시간에 계산되고, $O(k+M)$ 공간을 사용한다.

검색단계는 $n-m+1$개의 스텝으로 구성된다. 스텝 i $(m \le i \le n)$에서는 T의 각 위치 i에서 $T[i-q+1..i]$의 핑거프린트를 계산하고, 이 값을 모든 $FP[b](1 \le b \le k)$와 비교한다. 만약 일치하는 $FP[b]$가 존재하면, P_b가 T의 부분문자열과 순위동형일 가능성이 있는 후보패턴이다. 따라서 $T[i-m+1..i-m+|P_b|]$와 P_b의 순위동형 여부는 $T[i-m+POS[b][j]] < T[i-m+POS[b][j+1]](1 \le j \le |P_b|)$으로 검증한다[12]. 만약 순위동형이면 i와 b를 출력한다. 이후 순위동형 여부과 관계없이, i를 1 증가시켜 위 과정을 반복한다. 최악의 경우, 위치 i마다 모든 패턴과 순위동형을 검증하므로 $O(q\log q+M)$ 시간이 요구된다. 따라서 검색단계는 총 $O((q\log q+M)n)$ 시간에 수행된다.

3. 해싱기반 순위다중패턴매칭 알고리즘의 검색단계 병렬 계산

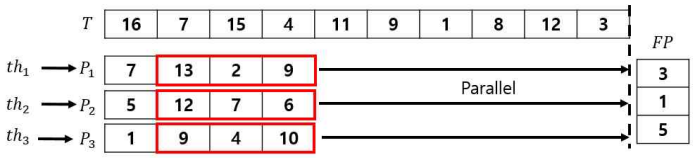

그림 1. $q=3$일 때, 3개의 패턴에 대한 검색단계 병렬 계산

Algorithm 1 Parallel_Computation_Of_Searching

Input: T, $\hat{P}=\{P_1,P_2,\cdots,P_k\}$, FP, and POS
Output: All the positions i of the substrings of T which are order-isomorphic to $P_b(1 \le b \le k)$

```
1 parallel for b ← 1 to k do
2   for i_b ← m to n do
3     if FP[b] = f(T[i_b−q+1..i_b]) then
4       if P_b ≈ T[i_b−m+1..i_b−m+|P_b|] then
5         print pattern P_b occurs at position i_b
```

본 논문에서 제시하는 알고리즘은 전처리단계와 검색단계로 구성된다. 전처리단계는 [6]의 알고리즘과 동일하게 핑거프린트테이블 $FP[b](1 \le b \le k)$와 위치테이블 POS를 생성한다. 검색단계는 다음과 같이 수행한다. T, $\hat{P}$, FP, POS가 주어졌을 때, k개의 스레드 집합 $\widehat{Th}=\{th_1,th_2,\cdots,th_k\}$을 사용하여 하나의 스레드가 하나의 패턴에 대해 검색단계를 병렬적으로 수행한다(Algorithm 1 및 그림 1 참조). 즉, 각 스레드 $th_b(1 \le b \le k)$가 P_b에 대해 순위패턴매칭을 수행한다(Algorithm 1의 1번째 줄). 스레드 th_b에 대한 T의 위치를 $i_b(m \le i_b \le n)$라 하자. 스레드 th_b는 먼저 $f(T[i_b-q+1..i_b])$를 계산한다. 만약 $f(T[i_b-q+1..i_b])$와 $FP[b]$가 같다면 $POS[b]$를 이용하여 P_b와 $T[i_b-m+1..i_b-m+|P_b|]$가 순위동형인지 검증한다. 만약 순위동형이면, i와 b를 출력한다(Algorithm 1의 2번째 줄부터 5번째 줄). 이후 순위동형 여부와 관계없이, i를 1 증가시켜 위 과정을 반복한다.

검색단계의 복잡도는 다음과 같다. 최악의 경우, 모든 패턴들이 위치 $i_b(m \le i_b \le n)$마다 순위동형을 검증할 수 있으므로 $O(q\log q+\overline{m})$ 시간이 소비된다. 따라서 검색단계는 총 $O((q\log q+\overline{m})n)$ 시간에 수행된다.

4. 실험 결과

실험환경은 다음과 같다. CPU는 AMD Ryzen 9 3950X, RAM은 64GB, 그래픽 카드는 NVIDIA GeForce RTX 2080 Ti, 개발툴은 Visual Studio 2017, 프로그래밍 언어는 C++와 CUDA를 사용하였다. 텍스트 T와 패턴집합 $\hat{P}$는 rand()를 이용하여 무작위로 생성하였다. $q=5$, k는 1,000에서 10,000까지 1,000씩, m은 6에서 15까지 1씩, n은 10,000에서 100,000까지 10,000씩 증가시켜가며 실험을 진행하였다. [6]에서 제시된 순차알고리즘을 FA, 본 논문에서 제시하는 병렬알고리즘을 FPA로 표기한다. FPA의 스레드블록의 수와 블록당 스레드 수는 각각 $(k+127)/128$, 128개로 설정하였다. 알고리즘의 수행시간은 cudaMemcpy()의 시간을 포함하고 매개변수별로 100회 반복 실험한 검색단계의 평균 시간이고, 소수점 셋째 자리에서 반올림하였다.

그림 2는 $m=10$, $n=100{,}000$일 때, k에 따른 검색단계의 수행시간을 보여준다. 시간복잡도대로 순차알고리즘은 k가 증가

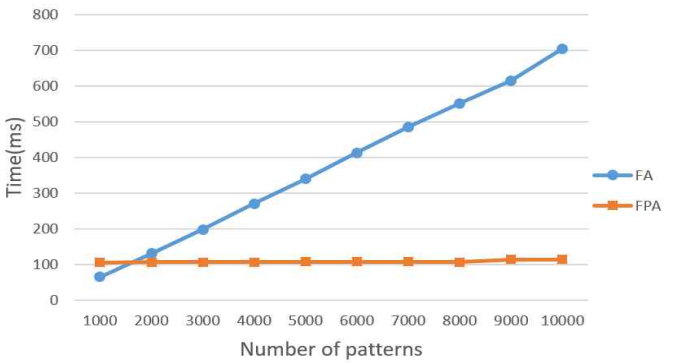

그림 2. $m=10$, $n=100{,}000$일 때, k에 따른 FA와 FPA의 검색단계 수행시간

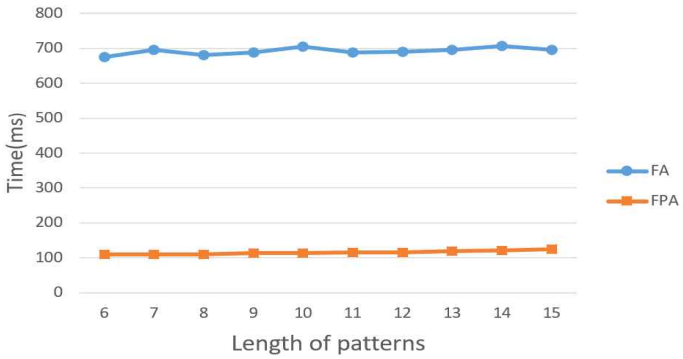

그림 3. $k=10{,}000$, $n=100{,}000$일 때, m에 따른 FA와 FPA의 검색단계 수행시간

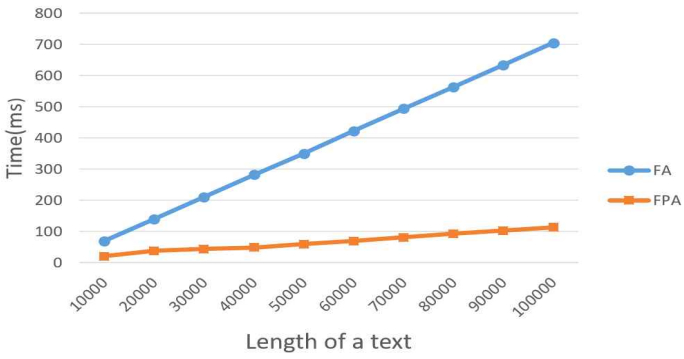

그림 4. $m=10$, $k=10{,}000$일 때, n에 따른 FA와 FPA의 검색단계 수행시간

함에 따라 수행시간이 선형적으로 증가한 반면, 병렬알고리즘은 k의 변화에 거의 영향을 받지 않았다. $k=10{,}000$일 때, FA는 705.23ms, FPA는 113.79ms로 FPA가 FA보다 약 6.2배 빠르게 수행되었다. 한편, $k=1{,}000$일 때 FPA는 FA보다 느리게 수행되었다.

그림 3은 $k=10{,}000$, $n=100{,}000$일 때, m에 따른 검색단계의 수행시간을 보여준다. $m \ll n$이기 때문에, 두 알고리즘의 수행시간은 m의 변화에 크게 영향을 받지 않았다. 평균적으로 FA가 약 716.56ms, FPA가 약 133.44ms로 FPA가 FA보다 약 5.37배 빠르게 수행되었다.

그림 4는 $m=10$, $k=10{,}000$일 때, n에 따른 검색단계의 수행시간을 보여준다. 두 알고리즘의 수행시간은 n이 증가함에 따라 선형적으로 증가하였다. $n=100{,}000$일 때 두 알고리즘의 수행시간은 그림 2에서 설명한 바와 같다.

참고문헌

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

[2] A. V. Aho, 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*, Vol. 9538, pp. 248-259, Oct. 2015.

[4] S. Wu, U. Manber, "A Fast Algorithm For Multi-Pattern Searching," *Technical Report TR 94-17*, University of Arizona at Tuscon, 1994.

[5] R. M. Karp, M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM J. Res. Dev.* Vol. 31, No. 2, pp. 249-260, 1987.

[6] J. Park, Y.H. Kim, J.S. Sim, "A Space-Efficient Hashing-Based Algorithm for Order-Preserving Multiple Pattern Matching Problem," *KIISE Transactions on Computing Practices*, Vol. 24, No. 8, pp. 399-404, 2018. (in Korean)

[7] J. Park, Y. H. Kim, S. Kwan, J.S. Sim, "Parellel Computation of the Shift Table of a Hashing-Based Algorithm for the Order-Preserving Multiple Pattern Matching," *KIPS Spring Conference 2017*, Vol. 24, No. 1, pp. 36-39, 2017 (in Korean)

[8] Y. Shin, Y. H. Kim, J. S. Sim, "Parallel Computation of Z-Function for Order-Preserving Pattern Matching and Order-Preserving Multiple Pattern Matching," *Journal of KIISE*, Vol. 45, No. 8, pp. 778-785, 2018 (in Korean)

[9] J. Choi, Y. H. Kim, J. C. Na, J. S. Sim, "Parallel Algorithm for the Boxed-Mesh Permutation Pattern Matching Problem," *Journal of KIISE*, Vol. 46, No. 4, pp. 299-307, 2019 (in Korean)

[10] R. N. Horspool, "Practical fast searching in strings," *Software: practice & experience*, Vol. 10, No. 6, pp. 501-506, 1980.

[11] S. Cho, J. C. Na, K. Park, J. S. Sim, "A fast algorithm for order-preserving pattern matching," *Information Processing Letters*, Vol. 115, No. 2, pp. 397-402, 2015.

[12] T. Chhabra, J. Tarhio, "A filtration method for order-preserving matching," *Information Processing Letters*, Vol. 116, No. 2, pp. 71-74, Feb. 2016.