문자열의 최대 palindrome을 구하기 위한 상수시간 RMESH 알고리즘

우진운*

요약

문자열 연산이 계산 생물학 분야에 응용되면서 효율적인 문자열 연산을 위한 다양한 자료구조와 알고리즘이 연구되고 있다. 최대 palindrome을 구하는 문제는 주어진 문자열에서 좌우 대칭인 최대 부분문자열을 찾는 연산이다. 이 연산은 패턴 매칭, 유사도 측정 등의 문자열 처리 분야에서 중요하게 사용되고 있다. 본 논문에서는 RMESH(Reconfigurable MESH) 구조에서 3-차원 $n\times n\times n$ 프로세서를 사용하여 문자열의 최대 palindrome을 구하는 알고리즘을 제안하며, 이 알고리즘은 $O(1)$ 시간 복잡도를 갖는다. 상수시간 알고리즘은 대량의 데이터를 처리하는 계산 생물학과 같은 응용 분야에서 유용하게 사용된다.

Constant Time RMESH Algorithm for Finding Maximal Palindromes of String

Jin-Woon Woo*

ABSTRACT

Since string operations were applied to computational biology area, various data structures and algorithms for computing efficient string operations have been studied. The maximal palindrome problem is an operation to find the maximal symmetric substrings in a string. This operation is importantly used in the string processing area such as pattern matching and likelihood measurement. In this paper, we present an algorithm to compute the maximal palindromes of the given string using three-dimensional $n\times n\times n$ processors on RMESH(Reconfigurable MESH). The algorithm has $O(1)$ constant time complexity, which is usefully used in the massive data processing areas like the computational biology.

Key Words : string, maximal palindrome, edit distance, constant time, RMESH

* 단국대학교 컴퓨터학부
· 제1저자(First Author) : 우진운 · 교신저자(Correspondent Author) : 우진운
· 접수일(2010년 2월 11일), 수정일(1차 : 2010년 3월 15일), 게재확정일(2010년 3월 18일)

Ⅰ. 서 론

최근 문자열 연산들이 계산 생물학 분야에 응용되면서 효율적인 문자열 연산을 위한 자료구조와 알고리즘들이 연구되고 있다. 이러한 문자열 연산에는 문자열 패턴 매칭(string pattern matching), 접미사-접두사 매칭(suffix-prefix matching), 최장 공통 부분문자열(longest common substring) 찾기, 최대 반복자(maximal repeat) 찾기 등이 있다.

최대 palindrome은 주어진 문자열에서 좌우 대칭인 최대 부분문자열을 의미한다[1]. 최대 palindrome인 부분문자열을 찾는 문제는 패턴 매칭, 유사도 측정 등의 문자열 처리 분야에서 중요하게 사용될 뿐만 아니라, DNA 순서 간 일치성 여부를 찾아내는 등의 생물정보학 분야에서도 아주 중요하다[1,2].

최대 palindrome을 구하는 단일 프로세서 알고리즘은 주로 동적 프로그래밍을 이용하거나 접미사 트리를 이용한다. 먼저, 동적 프로그래밍은 주어진 문자열과 그 문자열의 역 문자열의 편집 거리(edit distance) 계산을 통한 방식으로 최대 palindrome을 계산한다[3,4]. 이는 문자열의 길이를 각각 n 일 때 O(n2) 시간복잡도를 가진다. 접미사 트리를 이용하는 방법은 선형 시간복잡도를 갖지만 많은 공간을 사용하는 단점이 있다.

문자열 연산과 관련된 병렬 알고리즘이 많이 연구되어 있다[5,6,7,8,9,10]. 대부분 CRCW-PRAM 모델에서 동작하는 알고리즘들로서 접미사 트리를 생성하여 문자열 매칭과 최장 반복 부분문자열을 O(log n) 시간에 수행하거나, 상수 시간에 문자열 매칭을 수행하는 알고리즘을 제안한다[10].

그러나, RMESH 모델이 제안된 이래, 영상처리 분야에서는 많은 RMESH 알고리즘들이 발표되었으나, 문자열 처리 분야에는 활발하게 발표되지 못하고 있다. Lee와 Ercal이 RMESH 모델에서 상수시간 문자열 매칭 알고리즘을 제안하였고, Datta와 Subbiah는 RMESH 모델에서 문자열 처리를 위한 상수시간 알고리즘들을 제안하였다.

본 논문은 RMESH 모델에서 최대 palindrome를 찾는 상수 시간 알고리즘들을 제안한다. 최대 palindrome 알고리즘은 동적 프로그래밍 기법을 사용하며 RMESH 모델의 각 프록세서의 스위치를 효율적으로 작동시킴으로써 상수 시간에 동작한다.

본 논문과 [3]에서는 두 문자열을 비교하기 위해 동적 프로그래밍 기법을 사용하고 이를 버스로 연결하는 점에서는 유사하다. 본 논문과 [3]은 3-차원 RMESH를 사용하여 최장 공통 부분문자열과 최대 반복자를 상수시간에 구하는 알고리즘을 제안하나, 이 문제를 해결하는 과정에서 [3]은 최대값을 구하는 일반적인 RMESH 알고리즘을 사용하나, 본 논문은 연결된 버스에서 직접 최대 palindrome을 구하기 위한 효율적인 기본 연산들을 사용한다.

Ⅱ. RMESH 구조

RMESH는 Reconfigurable MESH의 약어로서 기존의 메쉬(mesh) 구조에 동적으로 재구성 가능한 버스 시스템을 결합한 구조로서 Miller, Prasanna-Kumar, Reisis, Stout에 의하여 제안되었으며[11], 구조적인 장점 때문에 다양한 분야에서 연구되었고 효율적인 알고리즘들이 개발되었다[12,13].

2.1 2-차원 RMESH[5]

크기가 $n \times n$인 2-차원 RMESH의 기본 구조는 메쉬이며 프로세서들 사이의 통신을 위하여 브로드캐스트 버스(broadcast bus)가 존재한다. 예를 들어, 그림 1은 4×4 RMESH 구조를 보여준다. 프로세서들을 식별하기 위해 각 프로세서에게 PE (i, j) 를 부여한다. 이때 $0 \leq i, j < n$, i는 행의 인덱스이고, j는 열의 인덱스이다.

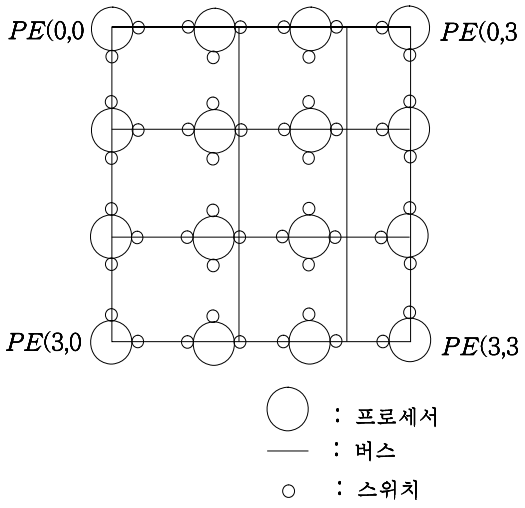


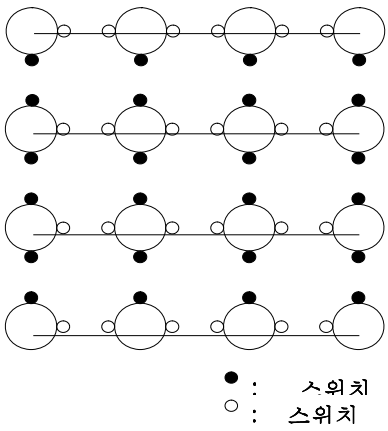
그림 1. 4×4 RMESH 구조
Fig. 1. RMESH structure

브로드캐스트 버스상의 통신 제어를 위하여 버스 스위치가 있다. 버스 스위치들은 각 프로세서의 상, 하, 좌, 우에 하나씩 존재하는데, 이를 각각 N(north), S(south), W(west), E(east)라 한다.

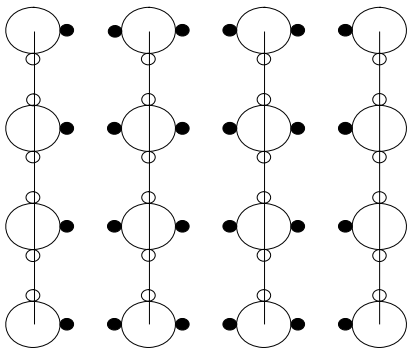
버스 스위치는 각 프로세서의 소프트웨어에 의하여 $O(1)$ 시간에 조작되며, 스위치의 개폐 여부에 따라 브로드캐스트 버스를 다수의 서브버스(subbus)들로 재구성이 가능하다. 예를 들어, 각 프로세서가 자신의 S와 N 스위치를 끊고 E와 W 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 그림 2(a)와 같은 행 버스(row bus)라 하고, 자신의 E와 W 스위치를 끊고 S와 N 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 그림 2(b)와 같은 열 버스(column bus)라 한다.

또한 각 프로세서가 외부적으로 S와 N 스위치, E와 W 스위치를 연결하고, 내부적으로 N과 E 스위치, S와 W 스위치를 연결하게 되면 여러 개의 서브버스가 형성되며 이를 그림 2(c)와 같은 대각선 버스(diagonal bus)라 한다.

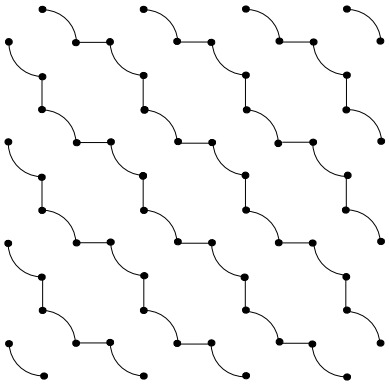
두 개의 프로세서들은 충돌이 없는 한 공통된 하나의 특정 스위치를 동시에 개폐할 수 있다. 버스상에는 특정 시간에 단 하나의 프로세서만이 데이터를 실을 수 있으며, 서브버스 위에 실린 데이터는 단위 시간에 그 버스에 연결된 모든 프로세서에게 전달될 수 있다. 만약 한 프로세서가 서브버스상에 있는 모든 프로세서에게 레지스터(register) X의 값을 브로드캐스트하려면 broadcast(X) 명령을 사용하고, 브로드캐스트 버스의 내용을 읽어 레지스터 R에 저장하려면 R := content(broadcast bus) 명령을 사용한다. 따라서 데이터 브로드캐스트는 $O(1)$ 시간에 수행된다.



(a) 행 버스
(a) row bus



(b) 열 버스
(b) column bus



(c) 대각선 버스
(c) diagonal bus

그림 2. 행, 열, 대각선 서브버스
Fig. 2. row, column, and diagonal bus

2.2 3-차원 RMESH

2-차원 RMESH를 확장하여 3-차원 RMESH를 구성할 수 있다. 3-차원 RMESH에서는 각 프로세서에게 PE (l, i, j)를 부여한다. 이때 $0 \le l, i, j < n$, l은 각 프로세서가 위치한 계층(layer)이고, i와 j는 계층 l에서의 행과 열의 인덱스이다.

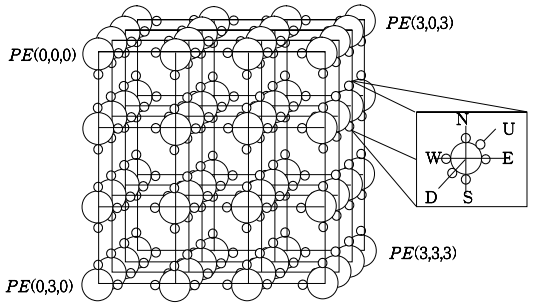


그림 3. 4×4×4 RMESH 구조
Fig. 3. 4×4×4 RMESH structure

예를 들어, 그림 3은 4×4×4 RMESH를 보여준다. 버스 스위치들은 기본적으로 2-차원 RMESH와 같이 N, S, W, E 스위치가 존재하며, 추가적으로 각 프로세서마다 계층을 연결하는 U(up)와 D(down) 스위치가 존재한다. 그리고 모든 프로세서의 N, S, W, E 스위치를 끊고 U와 D 스위치를 연결하면 여러 개의 서브버스가 형성되는데, 이를 UD 버스라 한다.

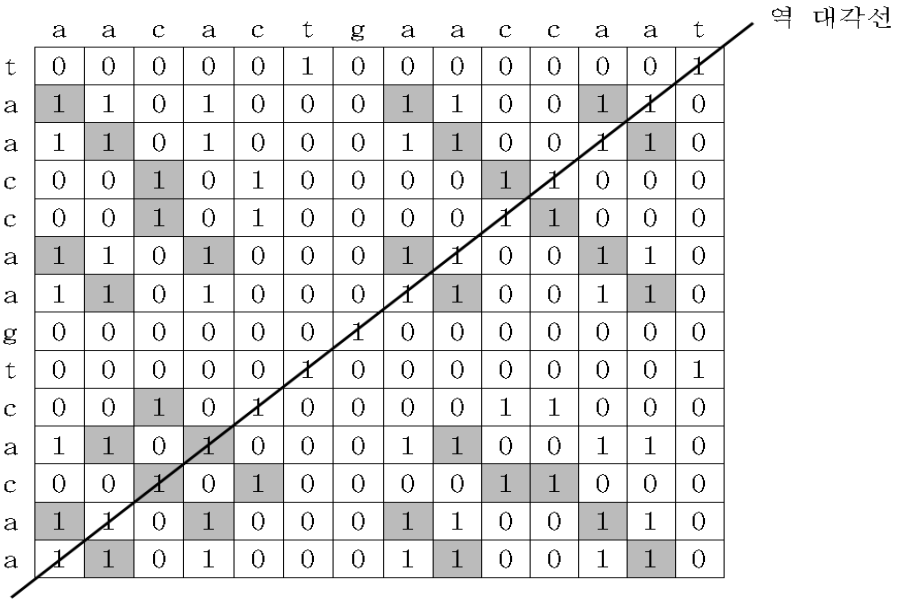
III. 최대 palindrome 구하기

최대 palindrome은 주어진 문자열에서 좌우 대칭인 최대 부분문자열을 의미한다. 예를 들어 문자열 “aacactgaaccaat"에서 좌우 대칭인 최대 부분문자열들을 밑줄로 표시하면 다음과 같다. “aacactgaaccaat", “aacactgaaccaat", “aacactgaaccaat", “aacactgaaccaat", “aacactgaaccaat", “aacactgaaccaat". 이때 다섯 번째 부분문자열 ”aaccaa"에서 “acca"와 ”cc"도 좌우 대칭인 부분문자열에 해당하지만 “aaccaa"에 모두 포함되므로 ”aaccaa"가 최대 palindrome에 해당된다.

RMESH 구조에서 문자열의 최대 palindrome을 구하기 위해서는 동적 프로그래밍 기법을 적절히 이용한다. 동적 프로그래밍은 두 문자열의 편집 거리(edit distance) 계산을 통한 방식으로 최대 palindrome을 계산한다[4,14]. 이차원 테이블에서 각 테이블 원소의 행과 열에 해당하는 문자가 일치하면 1, 그렇지 않으면 0으로 표시한 후에 대각선 방향으로 연속된 1의 개수가 편집 거리에 해당한다.

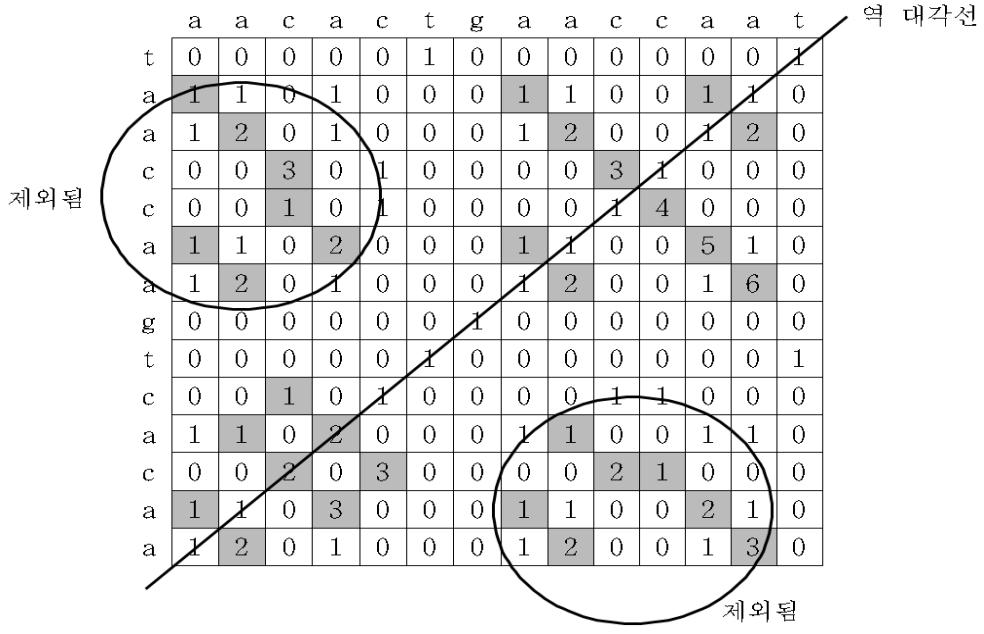
주어진 문자열과 그 문자열의 역 문자열의 편집 거리를 계산하고, 편집 거리가 2 이상이 되는 부분문자열들이 최대 palindrome 후보들이 된다. 최대 palindrome 후보들 중에서 역 대각선을 기준으로 대칭인 부분문자열만이 최대 palindrome 이 되므로 이 부분문자열들만을 남기고 나머지들을 제외시키면, 최종 남은 부분문자열들이 최대 palindrome이 된다.

위에서 사용한 문자열 "aacactgaaccaat" 을 이용하여 예를 들어 보자.



	a	a	c	a	c	t	g	a	a	c	c	a	a	t
t	0	0	0	0	0	1	0	0	0	0	0	0	0	1
a	1	1	0	1	0	0	0	1	1	0	0	1	1	0
a	1	1	0	1	0	0	0	1	1	0	0	1	1	0
c	0	0	1	0	1	0	0	0	0	1	1	0	0	0
c	0	0	1	0	1	0	0	0	0	1	1	0	0	0
a	1	1	0	1	0	0	0	1	1	0	0	1	1	0
a	1	1	0	1	0	0	0	1	1	0	0	1	1	0
g	0	0	0	0	0	0	1	0	0	0	0	0	0	0
t	0	0	0	0	0	1	0	0	0	0	0	0	0	1
c	0	0	1	0	1	0	0	0	0	1	1	0	0	0
a	1	1	0	1	0	0	0	1	1	0	0	1	1	0
c	0	0	1	0	1	0	0	0	0	1	1	0	0	0
a	1	1	0	1	0	0	0	1	1	0	0	1	1	0
a	1	1	0	1	0	0	0	1	1	0	0	1	1	0

(a) 행과 열 문자의 일치성을 0과 1로 표시
(a) 0 and 1 representing the equality of row and column character



	a	a	c	a	c	t	g	a	a	c	c	a	a	t
t	0	0	0	0	0	1	0	0	0	0	0	0	0	1
a	1	1	0	1	0	0	0	1	1	0	0	1	1	0
a	1	2	0	1	0	0	0	1	2	0	0	1	2	0
c	0	0	3	0	1	0	0	0	0	3	1	0	0	0
c	0	0	1	0	1	0	0	0	0	1	4	0	0	0
a	1	1	0	2	0	0	0	1	1	0	0	5	1	0
a	1	2	0	1	0	0	0	1	2	0	0	1	6	0
g	0	0	0	0	0	0	1	0	0	0	0	0	0	0
t	0	0	0	0	0	1	0	0	0	0	0	0	0	1
c	0	0	1	0	1	0	0	0	0	1	1	0	0	0
a	1	1	0	2	0	0	0	1	1	0	0	1	1	0
c	0	0	2	0	3	0	0	0	0	2	1	0	0	0
a	1	1	0	3	0	0	0	1	1	0	0	2	1	0
a	1	2	0	1	0	0	0	1	2	0	0	1	3	0

(b) 편집 거리 (b) edit distance
그림 4. 최대 palindrome을 구하는 예
Fig. 4. an example for computing the maximal palindrome

문자열 "aacactgaaccaat"과 역 문자열 "taaccaagtcacaa" 에 대해 테이블의 행과 열에 해당하는 문자끼리 비교하여 일치하면 1, 일치하지 않으면 0으로 표시하면 그림 4(a)와 같으며 이를 바탕으로 편집 거리를 구하면 그림 4(b)와 같게 된다.

그림 4(b)에서 거리가 2 이상인 부분 문자열들은 최대 반복자의 후보들이다. 이때 역 대각선을 경계로 대칭을 이루는 부분 문자열들만이 최대 palindrome이 되므로, 그림 4(b)에서 "aa", "aca", "cac", "aa", "aaccaa", "aa" 가 이에 해당한다. 후보들 중 이들을 제외한 나머지를 제외시키면 남은 부분문자열들이 최대 palindrome 들이 된다.

주어진 문자열의 길이가 n 일 때, $n \times n \times n$ 3차원 RMESH 구조에서 최대 palindrome을 구하는 알고리즘을 요약하면 알고리즘 1과 같이 9 단계로 구성된다.

초기에 주어진 문자열은 $n \times n \times n$ RMESH의 계층 0에 속하는 프로세서들의 첫 번째 행에 저장되어 있다고 가정한다. 즉, 계층 0의 프로세서 $PE(0,0,j)$ $(0 \le j \le n-1)$는 주어진 문자열의 한 문자씩을 가진다.

최대 palindrome를 계산하는 RMESH 알고리즘이 수행되는 과정을 단계별로 살펴보자.

[단계 0]에서는 계층 0의 첫 번째 행의 프로세서 $PE(0,0,j)$ $(0 \le j \le n-1)$가 자신의 문자를 계층 0의 첫 번째 열의 프로세서 $PE(0,i,0)$ $(0 \le i \le n-1)$에게 역순으로 전달한다. 이 과정을 위해 $PE(0,0,j)$ $(0 \le j \le n-1)$는 열 버스를 이용하여 브로드캐스트한 후, 역 대각선 상의 프로세서 $PE(0,i,j)$ $(i+j=n-1)$만이 전달된 문자를 행 버스를 이용하여 브로드캐스트한다.

[단계 0]: 첫 번째 행의 프로세서에 저장된 문자열을 역순으로 첫 번째 열의 프로세서에 저장한다.

[단계 1]: 계층 0의 첫 번째 행에 속하는 프로세서 $PE(0,0,j)$ $(0 \le j \le n-1)$는 자신의 문자를 열 버스를 이용하여 브로드캐스트하고, 계층 0의 첫 번째 열에 속하는 프로세서 $PE(0,i,0)$ $(0 \le i \le n-1)$는 자신의 문자를 행 버스를 이용하여 브로드캐스트한다.

[단계 2]: 계층 0의 모든 프로세서 $PE(0,i,j)$는 행과 열 버스로 전달받은 두 문자를 서로 비교하여 같으면 대각선 버스를 형성하고, 같지 않으면 대각선 버스를 차단한다.

[단계 3]: 대각선 버스를 이용하여 프로세서들을 세그먼트로 분리하며, 세그먼트의 첫 프로세서는 Start 레지스터를 1로, 마지막 프로세서는 End 레지스터를 1로 설정한다.

[단계 4]: Start = 1 인 프로세서는 자신의 프로세서 id 값을 대각선 버스를 통하여 브로드캐스트한다.

[단계 5]: End = 1 인 프로세서는 전달받은 프로세서의 id 값을 이용하여 편집 거리인 Distance 를 계산하여, (i, j, Distance) 값을 저장한다. 이때 i 는 전달받은 프로세서의 행 번호이고, j 는 열 번호이다.

[단계 6]: Distance > 1 인 (i, j, Distance) 만을 남기고 나머지는 제거한다.

[단계 7]: End = 1 인 프로세서는 자신의 행 번호와 열 번호를 이용하여 역 대각선에 대해 대각선인가를 확인한다.

[단계 8]: <i, Distance, j> 크기의 오름차순으로 정렬한다.

알고리즘 1. 최대 palindrome들을 계산하는 RMESH 알고리즘
Algorithm 1. RMESH algorithm for computing the maximal palindrome

이때 $PE(0,i,0)$ $(0 \le i \le n-1)$ 만이 문자를

이때 $PE(0,i,0)$ $(0 \le i \le n-1)$ 만이 문자를

레지스터에 저장한다. 이 단계는 두 번의 브로드캐스트 만을 수행하므로 $O(1)$ 시간에 수행 가능하다.

[단계 1]에서 계층 0의 프로세서 $PE(0,0,j)$ $(0 \le j \le n-1)$는 자신의 문자를 열 버스를 이용하여 브로드캐스트하고, 계층 0의 프로세서 $PE(0,i,0)$ $(0 \le i \le n-1)$는 자신의 문자를 행 버스를 이용하여 브로드캐스트한다. 따라서 이 단계는 두 번의 브로드캐스트 만을 수행하므로 $O(1)$ 시간에 수행 가능하다.

[단계 2]에서 계층 0의 모든 프로세서는 행과 열 버스를 통해 전달받은 두 문자를 비교하여 같으면 대각선 버스를 형성하고, 같지 않으면 대각선 버스를 차단한다. 이 결과로 계층 0의 프로세서들이 대각선 방향으로 세그먼트를 형성하게 된다. 이 과정은 동적프로그래밍 기법에서 설명된 바와 같이 두 문자가 같으면 1, 같지 않으면 0으로 나타내는 것과 같다. 이 단계는 단순히 한 번의 문자 비교만 하게 되므로 $O(1)$ 시간에 수행 가능하다.

[단계 3]에서는 대각선 방향으로 세그먼트를 형성하게 되며, 대각선 버스에서 왼쪽이 차단되고 오른쪽이 연결된 프로세서는 Start 레지스터에 1을 저장하고, 그렇지 않으면 0을 저장한다. 여기서 Start 레지스터가 1인 프로세서는 하나의 세그먼트 시작을 의미한다. 왼쪽은 연결되었으나 오른쪽이 차단된 프로세서는 End 레지스터에 1을 저장하고, 그렇지 않으면 0을 저장한다. 여기서 End 레지스터가 1인 프로세서는 세그먼트의 마지막에 해당한다. 이 단계는 대각선 버스의 인접한 프로세서만을 확인하므로 $O(1)$ 시간에 수행 가능하다.

[단계 4]에서 Start = 1 인 프로세서는 세그먼트 내의 프로세서들에게 자신의 프로세서 id 값을 대각선 버스를 통하여 브로드캐스트 하는데, 프로세서 id 는 프로세서의 행과 열 번호로 나타낸다. 즉, 프로세서 $PE(0,i,j)$의 경우, (i, j)를 브로드캐스트한다. 이 과정은 세그먼트를 나타내는 대각선 버스 상에서 한 번의 브로드캐스트만 일어나므로 $O(1)$ 시간에 수행된다.

[단계 5]에서 End = 1 인 프로세서는 전달받은 프로세서의 id 를 이용하여 (Distance, i, j) 값을 저장한다. 여기서 Distance는 편집 거리에 해당하며 자신의 열 번호에서 전달받은 프로세서의 id 의 j 값을 뺀 값이며, i와 j는 전달받은 프로세서 id 값에 해당한다. 즉, 프로세서 $PE(0,i',j')$가 (i, j) 값을 전달받았다면, Distance = j'-j+1 로 계산되고, (Distance, i, j) 값을 저장한다. End = 0 인 프로세서는 (0, 0, 0)의 값을 저장한다. 이 단계에서는 세그먼트의 마지막 프로세서가 (Distance, i, j) 필드를 계산하는 시간만 소요하므로 걸리는 시간은 $O(1)$이다.

[단계 6]에서 프로세서는 단계 5에서 계산된 (i, j, Distance)을 가지고 있다면 Distance를 확인하여 Distance > 1 이 아닌 경우에는 (i, j, Distance) 을 제거한다. 이 단계는 단순히 Distance 의 값만을 비교하므로 $O(1)$ 시간에 수행 가능하다.

[단계 7]에서는 End = 1 인 프로세서가 (i, j, Distance)의 i 와 j 값과 자신의 행 번호 i'과 열 번호 j'을 이용하여 역 대각선에 대해 대칭인가를 확인한다. 즉 역 대각선의 식 $y = -x + (n-1)$ (n은 문자열의 길이)에 대칭이기 위해 $(i', j') = (n-1-j,\ n-1-i)$ 의 관계식을 만족해야 한다. 이 단계는 단순히 i', j' 와 i, j 를 서로 비교하므로 $O(1)$ 시간에 수행 가능하다.

[단계 8]은 (i, Distance, j)의 i 와 Distance 값을 기준으로 하여 오름차순으로 정렬한다. Nigam과 Sahni[15]는 rotate sort 알고리즘을 3-차원 $n \times n \times n$ RMESH에 적용하여 n^2 개의 데이터를 $O(1)$ 시간에 정렬할 수 있는 알고리즘을 제안하였는데, 이 알고

리즘에서 초기의 n^2 개의 데이터는 계층 0에 속하는 PE $(0,i,j)$ $(0 \le i,j < n)$에 존재하며, 정렬된 결과는 계층 0에 속하는 프로세서에 row-major 순서로 하나씩 저장된다. 즉, PE $(0,0,0)$, PE $(0,0,1)$, $\cdots$, PE $(0,1,0)$, PE $(0,1,1)$, $\cdots$, PE $(0,n-1,n-1)$의 순서로 저장된다. 이 단계에서 이 정렬 알고리즘을 이용할 때 (i, Distance, j) 값들을 $O(1)$ 시간에 정렬할 수 있다. 이때, 계층 0의 프로세서에 저장된 (i, Distance, j)은 j 번째 문자에서부터 Distance 길이의 부분 문자열이 최대 palindrome임을 나타낸다.

예를 들어, 그림 4의 문자열에 대해 알고리즘1의 단계8까지 적용되면 (1,6,7), (1,2,11), (5,2,7), (9,3,2), (10,3,1), (12,2,0)과 같은 결과가 생성되며, 이것은 각각 최대 palindrome인 "aacactgaaccaat", "aacactgaaccaat", "aacactgaaccaat", "aacactgaaccaat", "aacactgaaccaat", "aacactgaaccaat"에 해당한다.

지금까지 알고리즘 1의 각 단계가 모두 $O(1)$ 시간에 수행될 수 있음을 설명하였으며, 정리 1과 같이 요약할 수 있다.

정리 1 : 주어진 문자열의 길이가 n 일 때, $n \times n \times n$ 3차원 RMESH 구조에서 최대 palindrome에 해당하는 부분문자열들을 구하는 알고리즘은 $O(1)$ 시간에 수행된다.

Ⅵ. 결론

최대 palindrome을 구하는 문제는 주어진 문자열에서 좌우 대칭인 최대 부분문자열을 찾는 연산이다. 이러한 문자열 연산들은 패턴 매칭, 유사도 측정 등의 문자열 처리 분야에서 중요하게 사용되고 있다.

본 논문에서는 3-차원 $n \times n \times n$ RMESH 구조에서 문자열의 최대 palindrome을 계산하는 상수 시간 알고리즘을 제안하였다. 이 알고리즘은 3-차원 RMESH 구조에서 구조적인 특성을 사용함으로써 $O(1)$ 상수 시간 복잡도를 가진다.

최대 palindrome 문제는 패턴 매칭, 유사도 측정 등의 문자열 처리 분야에서 중요하게 사용될 뿐만 아니라, DNA 순서 간 일치성 여부를 찾아내는 등의 생물정보학 분야에서도 아주 중요하다. 이런 응용 분야들은 일반적으로 대량의 데이터를 처리해야 하므로 본 논문에서 제안하는 상수 시간 알고리즘이 효율적으로 사용될 수 있을 것으로 기대된다.

참고문헌

[1] Dan. Gusfield, Algorithms on Strings, Trees, and Sequences, Computer Science and Computational Biology, CamBridge University Press, 1997.

[2] S. Needleman, C. Wunsch, "*A general method applicable to the search for similarities in the amino acid sequence of two proteins*", J. Mol. BioInformatics, 48(3), 443-453, 1970.

[3] A. Datta, S. Subbiah, "*Constant Time Algorithms for String Processing on the Reconfigurable Mesh*", The 4th Australasian conference on Parallel and real-time systems, pp. 226-237, 1997.

[4] W. Masek and M. Paterson, "*A fast algorithm computing string edit distances*", J. of Computer and System Sciences, 20(1), pp13-31, 1980.

[5] 한선미, 우진운, "문자열의 최장 공통 부분문자열과 최대 반복자를 구하기 위한 상수시간 RMESH 알고리즘", *정보처리학회 논문지* A, 제16-A권, 제5호, pp. 319-326, 2009.

[6] A. Apostolico, C. Iliopoulos, G. Landau, BSchieber, and U. Vishkin, "*Parallel Construction of a Suffix Tree with Application*", Algorithmica, vol. 3, pp. 347-365, 1988.

[7] G. Landau, U. Vishkin, "*Fast parallel and serial approximate string matching*", J. of Algorithms, vol. 10, pp. 157-169, 1989.

[8] H. Lee, F. Ercal, "*RMESH Algorithms for Parallel String Matching*", International Symposium on Parallel Architectures, Algorithms and Networks, pp. 223-226, 1997.

[9] Y. Jiang, A. Wright, "*O(k) parallel algorithms for approximate string matching*", Neural, Parallel and Scientific Computations, vol. 1, pp. 443-452, 1993.

[10] Z. Galil, "*A Constant Time Optimal Parallel String Matching Algorithm*", J. of ACM, vol 42, pp. 908-918, 1995.

[11] R. Miller, V. Prasanna-Kumar, D. Reisis, and Q. Stout, "P*arallel Computation on Reconfigurable Meshes*", IEEE Transactions on Computers, Vol.42, No.6, pp.678-692, 1993.

[12] J. Jang, H. Park, and V. Prasanna, "*A Fast Algorithm for Computing Histogram on a Reconfigurable Mesh*", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.17, No.2, pp.97-106, 1995.

[13] 김경훈, 우진운, "RMESH 구조에서 unaligned 선형사진트리의 alignment를 위한 상수시간 알고리즘", *정보과학회논문지*, 제31권 1,2호, pp. 10-18, 2004.

[14] R. A. Wagner, Michael J. Fischer, "*The String-to-String Correction Problem*", Journal of the ACM (JACM), Volume 21 , Issue 1, pp. 168 - 173, 1974.

[15] M. Nigam and S. Sahni, "*Sorting* n *Numbers On* $n \times n$ *Reconfigurable Meshes With Buses*", Proceedings 7th International Parallel Processing Symposium, pp.174-181, 1993.

[16] 김석훈 외, "가중퍼지시스템의 신뢰도 분석을 위한 사다리꼴 모호집합의 사용", *한국지식정보기술학회 논문지*, 제4권 제3호, pp.43-52, 2009.

[17] 이상구 외, "OpenMP를 이용한 새로운 COG 방식의 고속 병렬 퍼지 제어기", *한국지식정보기술학회 논문지*, 제5권 제1호, pp.77-84, 2010.

감사의 글

이 연구는 2009년도 단국대학교 대학연구비의 지원으로 연구되었음.

우진운 (Jin-Woon Woo)

1980년 서울대학교 수학교육과 (학사)

1989년 미국 University of Minnesota 전산학과 (박사)

1989년 ~ 현재 단국대학교 컴퓨터학부 교수

※ 관심분야 : 분산 및 병렬처리, 병렬알고리즘, 자료구조