ISSN(Print): 2287-5891

http://dx.doi.org/10.3745/KTCCS.2013.2.2.067

Parallel Computation For The Edit Distance Based On The Four-Russians' Algorithm

Young Ho Kim† · Ju-Hui Jeong† · Dae Woong Kang† · Jeong Seop Sim††

ABSTRACT

Approximate string matching problems have been studied in diverse fields. Recently, fast approximate string matching algorithms are being used to reduce the time and costs for the next generation sequencing. To measure the amounts of errors between two strings, we use a distance function such as the edit distance. Given two strings $X(|X|=m)$ and $Y(|Y|=n)$ over an alphabet Σ, the edit distance between X and Y is the minimum number of edit operations to convert X into Y. The edit distance between X and Y can be computed using the well-known dynamic programming technique in $O(mn)$ time and space. The edit distance also can be computed using the Four-Russians' algorithm whose preprocessing step runs in $O((3|\Sigma|)^{2t}t^2)$ time and $O((3|\Sigma|)^{2t}t)$ space and the computation step runs in $O(mn/t)$ time and $O(mn)$ space where t represents the size of the block. In this paper, we present a parallelized version of the computation step of the Four-Russians' algorithm. Our algorithm computes the edit distance between X and Y in $O(m+n)$ time using m/t threads. Then we implemented both the sequential version and our parallelized version of the Four-Russians' algorithm using CUDA to compare the execution times. When $t=1$ and $t=2$, our algorithm runs about 10 times and 3 times faster than the sequential algorithm, respectively.

Keywords : Approximate String Matching, Edit Distance, The Four-Russians' Algorithm, CUDA

4-러시안 알고리즘 기반의 편집거리 병렬계산

김 영 호† · 정 주 희† · 강 대 웅† · 심 정 섭††

요 약

근사문자열매칭 문제는 다양한 분야에서 연구되어 왔다. 최근에는 차세대염기서열분석의 비용과 시간을 줄이기 위해 빠른 근사문자열매칭 알고리즘들이 이용되고 있다. 근사문자열매칭은 문자열들의 오차를 측정하기 위해 편집거리와 같은 거리함수를 이용한다. 알파벳 Σ에 대한 길이가 각각 m, n인 두 문자열 X와 Y의 편집거리는 X를 Y로 변환하기 위해 필요한 최소 편집연산의 수로 정의된다. 두 문자열의 편집거리는 잘 알려진 동적프로그래밍을 이용하여 $O(mn)$ 시간과 공간에 계산할 수 있으며, 4-러시안 알고리즘을 이용해서도 계산할 수 있다. 4-러시안 알고리즘은 블록 크기를 t라 할 때, 전처리 단계에서 $O((3|\Sigma|)^{2t}t^2)$ 시간과 $O((3|\Sigma|)^{2t}t)$ 공간이 필요하며, 계산 단계에서 $O(mn/t)$ 시간과 $O(mn)$ 공간을 이용하여 편집거리를 계산하는 알고리즘이다. 본 논문에서는 4-러시안 알고리즘의 계산 단계를 병렬화하고 실험을 통해 CPU 기반의 순차적 알고리즘과 CUDA로 구현한 GPU 기반의 병렬 알고리즘의 수행시간을 비교한다. 본 논문에서 제시하는 4-러시안 알고리즘의 계산단계는 m/t개의 쓰레드를 사용하여 $O(m+n)$ 시간에 편집거리를 계산한다. GPU 기반의 알고리즘이 CPU 기반의 알고리즘 보다 $t=1$일 때 약 10배 빠르고, $t=2$일 때 약 3배 빠른 결과를 보였다.

키워드 : 근사문자열매칭, 편집거리, 4-러시안 알고리즘, CUDA

※ 이 논문은 지식경제부 및 한국산업기술평가관리원의 IT산업원천 기술개발 산업의 일환으로 수행하였음[10038768, 유전체 분석용 슈퍼컴퓨팅 시스템 개발].
※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.2012R1A2A2A01014892).
※ This work was supported by the Industrial Strategic technology development program (10041971, Development of Power Efficient High-Performance Multimedia Contents Service Technology using Context-Adapting Distributed Transcoding) funded by the Ministry of Knowledge Economy(MKE, Korea).
※ 이 논문은 인하대학교의 지원에 의하여 연구되었음.

† 준 회 원 : 인하대학교 컴퓨터정보공학과 석사과정
†† 종신회원 : 인하대학교 컴퓨터정보공학부 부교수
논문접수 : 2012년 9월 26일
수 정 일 : 1차 2012년 11월 26일, 2차 2012년 12월 20일
심사완료 : 2012년 12월 21일
* Corresponding Author : Jeong Seop Sim(jssim@inha.ac.kr)

1. 서 론

문자열들의 오차를 어느 정도 허용하는 근사문자열매칭 문제는 생물정보학, 검색엔진, 컴퓨터보안 등 많은 분야에서 활발히 연구되고 있다[1,2,3,4,5]. 최근에는 차세대염기서열분석(next-generation sequencing)에서 레퍼런스 매핑(reference mapping)의 비용과 시간을 줄이기 위해 빠른 근사문자열매칭 알고리즘들이 이용되고 있다[6,7,8,9,10].

근사문자열매칭은 문자열들의 오차를 판별하는 척도로써 거리함수를 이용한다. 알파벳 Σ에 대해 길이가 각각 m, n인 두 문자열 X와 Y의 거리 $\delta(X, Y)$는 X를 Y로 변환하기 위해 필요한 최소 비용으로 정의된다. 잘 알려진 거리함수로는 해밍거리(Hamming distance), 편집거리(edit distance), 가중편집거리(weighted edit distance) 등이 있다[11]. 두 문자열 X와 Y의 편집거리는 X를 Y로 변환하기 위해 필요한 최소 편집연산(edit operation)의 수로 정의된다. 이때 편집연산은 삽입(insertion)연산, 삭제(deletion)연산, 교체(change)연산으로 구성된다. 해밍거리는 $n = m$일 때, X를 Y로 변환하기 위해 필요한 최소 교체연산의 수이다. 편집연산에 대한 비용을 일반화한 비용행렬(penalty matrix)이 있는데, 가중편집거리는 비용행렬을 이용하여 X를 Y로 변환하기 위해 필요한 최소 비용으로 정의된다.

두 문자열의 편집거리는 잘 알려진 동적프로그래밍(dynamic programming)을 이용하여 $O(mn)$ 시간과 공간에 계산할 수 있으며[12], 4-러시안 알고리즘을 이용해서도 계산할 수 있다[13,14]. 4-러시안 알고리즘은 블록의 너비와 높이를 t라 할 때, 전처리 단계에서 $O((3|\Sigma|)^{2t}t^2)$ 시간과 $O((3|\Sigma|)^{2t}t)$ 공간이 필요하며, 계산 단계에서 $O(mn/t)$ 시간과 $O(mn)$ 공간을 이용하여 편집거리를 계산하는 알고리즘이다. 앞으로 편의상 t를 블록의 크기라 하겠다.

한편, 최근 GPU(graphic processing unit)의 성능이 향상되면서 GPU를 활용한 문자열 관련 연구가 진행되고 있다[10,15,16,17,18,19]. [10]에서는 GPU를 이용하여 [6]의 레퍼런스 매핑 속도를 향상시켰다. [15,16]에서는 CPU를 이용하여 접미사트리(suffix tree)를 생성하고 GPU를 이용하여 레퍼런스 매핑 속도를 향상시켰다. [17]에서는 같은 문제를 접미사배열(suffix array)을 이용하여 해결하였다. [18]에서는 최장공통비상위문자열(longest common non-superstring)의 그래프 모델을 CUDA로 생성하여 수행속도를 향상시켰다. [19]에서는 Smith-Waterman 알고리즘을 GPU 기반으로 구현하여 성능을 향상시켰다.

본 논문에서는 [13]에서 제시된 편집거리를 계산하는 4-러시안 알고리즘의 계산 단계를 병렬화하고 실험을 통해 CPU 기반의 순차적 알고리즘과 CUDA로 구현한 GPU 기반의 병렬 알고리즘의 수행시간을 비교한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 알고리즘을 위한 용어 정의와 관련 연구의 결과들을 제시한다. 3장에서는 4-러시안 알고리즘으로 편집거리를 계산하는 병렬 알고리즘을 설명한다. 4장에서는 각각 CPU와 GPU로 구현한 4-러시안 알고리즘의 성능비교를 하고, 5장에서 결론과 향후 연구 방향을 제시한다.

2. 관련 연구

2.1 용어정리

문자열(string)은 알파벳 Σ에 존재하는 0개 이상의 문자들의 나열이다. 공백문자열은 λ로 나타낸다. Σ에 대한 문자열들의 집합을 Σ^*로 표기하고, 길이가 m인 문자열들의 집합은 Σ^m으로 표기한다. 문자열 S의 길이는 $|S|$로 표현한다.

$S[i](1 \le i \le |S|)$는 i번째 문자를 나타낸다. 문자열 A가 $S[i]S[i+1]...S[j](1 \le i \le j \le |S|)$라면, $S[i..j]$로 표기하고 A를 S의 부분문자열(substring)이라 한다. 예를 들어, bcd는 $abcde$의 부분문자열이다.

2.2 D-테이블

	λ	a	b	c	a	c	d	c	a	c
λ	0	1	2	3	4	5	6	7	8	9
a	1	0	1	2	3	4	5	6	7	8
b	2	1	0	1	2	3	4	5	6	7
a	3	2	1	1	1	2	3	4	5	6
b	4	3	2	2	2	2	3	4	5	6
c	5	4	3	2	3	2	3	3	4	5
a	6	5	4	3	2	3	3	4	3	4

Fig. 1. D-table for $X = ababca$ and $Y = abcacdcac$

Σ에 대해 길이가 각각 m, n인 두 문자열 X, Y가 주어졌을 때, 잘 알려진 동적프로그래밍을 이용하여 편집거리를 계산할 수 있다. 이때 계산된 $(m+1)\times(n+1)$ 크기의 테이블을 D-테이블이라고 하자(Fig. 1 참조). D-테이블의 $D[i,j]$는 $X[1..i]$와 $Y[1..j]$의 편집거리를 저장한다. 따라서 $D[m,n]$이 두 문자열 X, Y의 편집거리이다.

D-테이블을 계산하는 방법은 다음과 같다. 먼저 첫 번째 행과 첫 번째 열은 다음 식(1)과 같이 초기화한다.

$$\begin{cases} D[0,0] = 0 \\ D[i,0] = i \ (1 \le i \le m) \\ D[0,j] = j \ (1 \le j \le n) \end{cases} \quad (1)$$

이후, 각 $D[i,j]$는 다음 식(2)에 의해 계산된다.

$$D[i,j] = \min\begin{cases} D[i,j-1] & +1 \\ D[i-1,j] & +1 \\ D[i-1,j-1] & +\delta(X[i], Y[j]) \end{cases} \quad (2)$$

$\delta(X[i], Y[j])$는 $X[i] = Y[j]$이면 0, $X[i] \neq Y[j]$이면 1이 된다.

각 $D[i,j]$는 식(2)를 이용하여 $O(1)$시간에 계산할 수 있고, 테이블의 크기가 $(m+1)\times(n+1)$이므로 $O(mn)$ 시간과 공간을 이용하여 D-테이블을 계산할 수 있다. Fig. 1은 $X = ababca$, $Y = abcacdcac$에 대한 D-테이블이다.

2.3 4-러시안 알고리즘

4-러시안 알고리즘의 기본 아이디어는 D-테이블을 일정 크기의 블록단위로 나누어 블록의 테두리 부분만 계산하여 편집거리를 계산하는 것이다. 이를 위해 조합 가능한 모든 블록에 대한 거리를 미리 계산하고 필요한 부분만 룩업테이블(lookup table)에 저장하는 전처리 단계와, 룩업테이블을 이용하여 블록단위로 편집거리를 계산하는 계산 단계로 나눌 수 있다.

앞서 설명한 D-테이블에서 크기가 $(t+1)\times(t+1)$인 블록을 t-블록이라 하자. 특히 왼쪽 상단이 (i,j)인 t-블록을 (i,j,t) 블록이라 하겠다[13]. 이때 i, j는 t의 배수이다. 예를 들어 Fig. 2를 보면 t = 3일 때, (0,0,3) 블록, (0,3,3) 블록, (0,6,3) 블록, (3,0,3) 블록, (3,3,3) 블록, (3,6,3) 블록이 있다.

	λ	*a*	*b*	*c*	*a*	*c*	*d*	*c*	*a*	*c*
λ	0	1	2	3	4	5	6	7	8	9
a	1			2			5			8
b	2			1			4			7
a	3	2	1	1	1	2	3	4	5	6
b	4			2			3			6
c	5			2			3			5
a	6	5	4	3	2	3	3	4	3	4

Fig. 2. Computing the edit distances between $X = ababca$ and $Y = abcacdcac$ using the Four-Russians' algorithm, when t = 3

각 (i,j,t) 블록의 편집거리는 다섯 개의 변수 A, B, K, C, E에 의해 결정된다(Fig. 3 참조). A는 각 t-블록의 첫 번째 열을 나타내는 벡터이고, B는 첫 번째 행을 나타내는 벡터이다. 즉, A = $[D[i,j], D[i+1,j], ..., D[i+t,j]]$, B = $[D[i,j], D[i,j+1], ..., D[i,j+t]]$가 된다. K는 가장 위쪽의 블록들($i=0$)과 가장 왼쪽의 블록들($j=0$)의 경우 D-테이블의 정의에 의해 각각 j와 i이고, 그렇지 않으면 $(i-t,j-t,t)$ 블록에서 생성된 편집거리인 $D[i,j]$이다. C와 E는 각각 해당 블록에 대응하는 X와 Y의 부분문자열인 $X[i..i+t]$와 $Y[j..j+t]$이다.

4-러시안 알고리즘은 A, B, K, C, E가 주어졌을 때, Fig. 3과 같이 A', B', K'을 계산할 수 있다. A'과 B'은 각각 t-블록의 마지막 열과 마지막 행을 나타내는 벡터이며, K'은 t-블록의 편집거리이다. 즉, A' = $[D[i,j+t], D[i+1,j+t], ..., D[i+t,j+t]]$, B' = $[D[i+t,j], D[i+t,j+1], ..., D[i+t,j+t]]$, K' = $D[i+t,j+t]$이다. 예를 들어 Fig. 2의 (3,3,3) 블록은 A = [1,2,2,3], B = [1,1,2,3], K = 1, C = "$abca$", E = "$cacd$"이다. 계산 결과인 A' = [3,3,3,3], B' = [3,2,3,3], K' = 3이다.

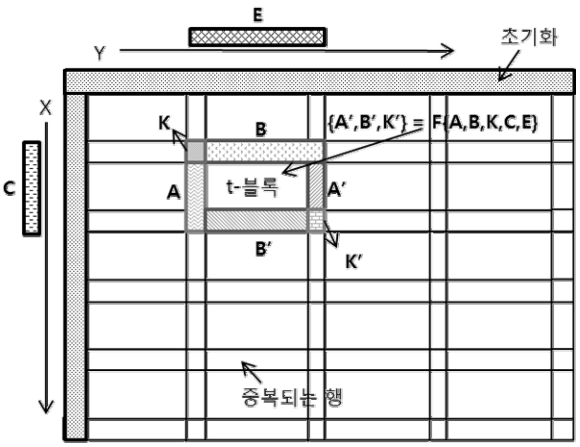

Fig. 3. Computation using preprocessed lookup table[14]

1) 전처리 단계

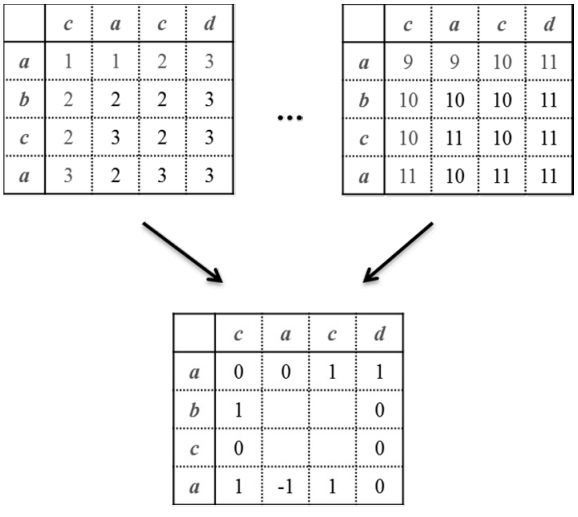

Fig. 4. Converting edit distances

전처리 단계에서는 t-블록에서 가능한 모든 조합에 대해 편집거리를 미리 계산하고 룩업테이블을 생성한다. 즉, A, B, C, E에 대한 모든 조합의 t-블록의 편집거리를 계산하여 A', B'을 룩업테이블에 저장한다.

t-블록의 가능한 조합의 수를 결정하기 위해 D-테이블의 임의의 행 또는 열에서 이전의 값과 차이가 최대 1이라는 속성을 이용하여 t-블록의 A, B, A', B'의 값을

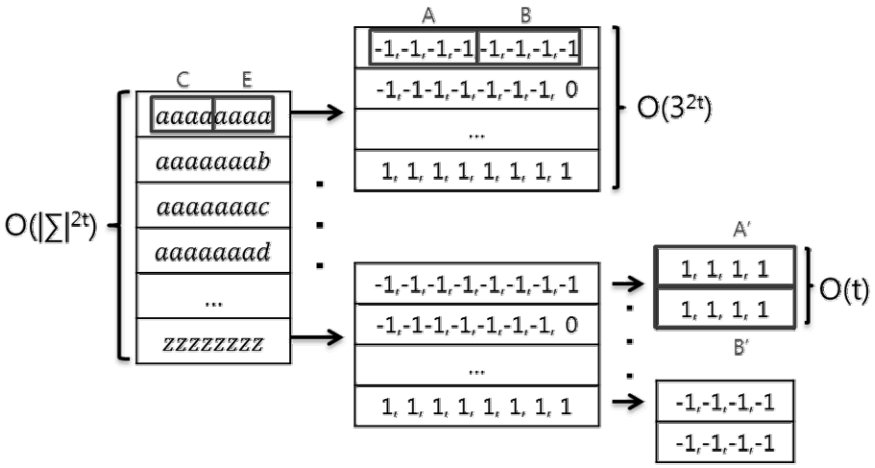

Fig. 5. Construction of a lookup table ($\Sigma = \{a, b, ..., z\}$, $t = 4$)

{-1,0,1}의 조합으로 변환할 수 있다[14]. 예를 들어 Fig. 2의 (3,3,3)블록은 A = [0,1,0,1], B = [0,0,1,1], A' = [0,0,0,0], B' = [0,-1,1,0]으로 변환할 수 있다. 이와 같은 변환은 C와 E만 동일하다면, Fig. 4와 같이 같은 차이 값으로 표현할 수 있는 많은 t-블록을 하나의 t-블록으로 표현할 수 있다. 이후 K를 이용하여 A, B, A', B'을 복원할 수 있다.

룩업테이블을 생성하는 방법은 다음과 같다. A, B, C, E에 대한 모든 조합의 t-블록의 편집거리를 계산한다. 이후 Fig. 5와 같이 각 조합에 대해 각각 길이가 (t+1)인 {-1,0,1}로 구성된 A'과 B'을 저장하고 있는 룩업테이블을 생성한다. 이것은 계산 단계에서 $O(1)$ 시간에 해당 조합인 t-블록의 A'과 B'을 찾게 해준다.

{-1,0,1}에 대한 길이가 (t+1) 벡터의 가능한 모든 조합의 수는 3^{t+1}개이다. 그리고 Σ에 대한 길이 (t+1)인 문자열의 가능한 모든 조합의 수는 $|\Sigma|^{t+1}$개이다. 그러므로 룩업테이블을 생성하는데 $O((3|\Sigma|)^{2t}t^2)$ 시간이 필요하고, 각 조합에 대해 A'과 B'만 저장하기 때문에 $O((3|\sum|)^{2t}t)$ 공간이 필요하다.

2) 계산 단계

먼저 계산 단계에서는 D-테이블의 첫 번째 행과 열은 초기화 되어야 한다. 이는 식(1)의 D-테이블을 초기화하는 과정과 같다.

이후 각 블록의 A, B, K, C, E를 입력받아 전처리 단계에서 생성한 룩업테이블을 이용하여 A', B', K'을 계산한다. 룩업테이블에서 해당 조합의 정보를 찾는데 $O(1)$ 시간이 필요하고, 변환된 A', B', K' 값을 복원하기 위해서 $O(t)$ 시간이 필요하다. A', B', K'은 인접한 블록의 편집거리를 계산할 때, 다시 입력으로 사용된다. 이러한 방법으로 각 t-블록의 마지막 행과 열을 계산하여 편집거리를 계산할 수 있다. 그러므로 D-테이블의 편집거리는 4-러시안 알고리즘을 이용하여 $O(mn/t)$에 계산되며, 이론적으로 $t = \Theta(\log m)$일 때, $O(mn/\log m)$에 계산할 수 있다. 하지만, 메모리의 한계 때문에 실험은 t = 1과 t = 2에 대해 수행한다.

3. 4-러시안 알고리즘의 병렬계산

3.1 t-블록 계산의 병렬화 방법

[20]에서는 D-테이블을 이용한 편집거리 계산 알고리즘이나 Smith-Waterman 알고리즘을 병렬화 할 때, 데이터 종속성 때문에 테이블의 각 셀을 오른쪽 위에서 왼쪽 아래로 향하는 대각선(이후 간단히 대각선으로 부르기로 한다.) 방향으로 계산하는 방법을 제시하였다. 예를 들어 D-테이블을 이용하여 편집거리를 계산할 때, $D[i,j]$를 계산하기 위해서는 식 (2)와 같이 $D[i-1,j]$, $D[i,j-1]$, $D[i-1,j-1]$이 먼저 계산되어야 한다. 따라서 순차 알고리즘에서와 같이 수평 방향 또는 수직 방향으로 D-테이블을 계산하는 것은 병렬 알고리즘에서는 적절하지 않다. 이러한 데이터 종속성을 피하기 위해 [20]에서는 대각선 방향의 각 셀을 병렬적으로 계산하는 방법을 제시하였다.

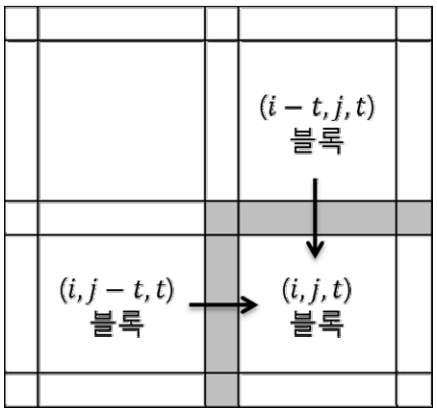

Fig. 6. Data dependency of the Four-Russians' algorithm

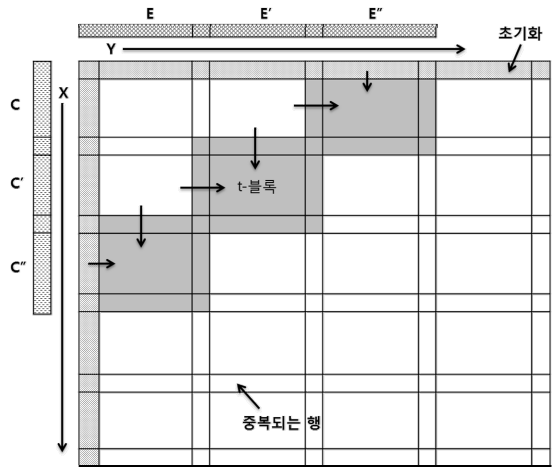

Fig. 7. Parallelization of the Four-Russians' algorithm

이와 유사하게 4-러시안 알고리즘도 t-블록 간에 종속성이 존재한다. Fig. 6과 같이 (i,j,t) 블록의 계산은 $(i-t,j,t)$ 블록과 $(i,j-t,t)$ 블록이 먼저 계산되어야 (i,j,t) 블록의 첫 번째 행과 첫 번째 열을 알 수 있다. 그러므로 본 논문에서는 [20]과 같이 대각선 방향으로 4-러시안 알고리즘을 적용한다.

Fig. 7과 같이 대각선에 있는 t-블록들은 이전 대각선에 있는 t-블록이 계산되어 있다면, 병렬적으로 계산할 수 있다. 예를 들어, Fig. 2에서 (0,0,3) 블록이 계산되어 있다고 가정하면, 0번 쓰레드가 (0,3,3) 블록을, 1번 쓰레드가 (3,0,3) 블록을 병렬적으로 계산할 수 있다.

3.2 계산 단계 병렬화

두 문자열 X와 Y의 길이를 각각 m, $n(m \le n)$이라 하자. 제시하는 알고리즘에서는 각 t-블록을 하나의 쓰레드가 계산한다. Fig. 8은 D-테이블을 나타내며 각 셀은 하나의 t-블록을 나타내고 있다. 각 블록의 번호는 할당된 쓰레드 번호를 의미한다. D-테이블에서 대각선에 있는 t-블록들은 D-테이블의 정의에 의해 독립적으로 계산될 수 있기 때문에, 쓰레드 사이에 동기화가 적절히 이루어진다면 Fig. 8과 같이 쓰레드를 할당하여 계산을 수행할 수 있다. 즉, 본 논문에서 제시하는 병렬 알고리즘은 최대 m/t개의 쓰레드를 사용하여 한 스텝에 대각선에 있는 t-블록들의 편집거리를 병렬적으로 계산한다.

각 스텝 k에 따라 본 알고리즘은 다음과 같이 총 세 가지 경우로 나누어 수행된다.

(1) $1 \le k \le m/t$: 이전 스텝보다 쓰레드를 하나씩 추가로 할당하여 동일한 대각선에 있는 k개의 t-블록의 편집거리를 계산한다.
(2) $m/t+1 \le k \le n/t$: m/t개의 쓰레드를 이용하여 동일한 대각선에 있는 m/t개의 t-블록의 편집거리를 계산한다.
(3) $n/t+1 \le k \le (m+n)/t-1$: 이전 스텝보다 쓰레드를 하나씩 줄이면서 동일한 대각선에 있는 $(m+n)/t-k$개의 t-블록의 편집거리를 계산한다.

(1), (2), (3)에서 알 수 있듯이 본 알고리즘은 총 $O((m+n)/t)$ 스텝에 계산단계를 수행한다. 각 스텝에서 t-블록의 계산은 룩업테이블에서 A'과 B'을 읽고 복원하는데 $O(t)$ 시간이 필요하기 때문에, 본 논문에서 제시하는 병렬 알고리즘은 $O(m+n)$에 수행된다.

4. 실험 결과 및 분석

실험 환경은 Table 1과 같다. Σ는 DNA 알파벳을 사용하였고, 두 문자열의 길이는 동일하게 실험하였다. 시간 함

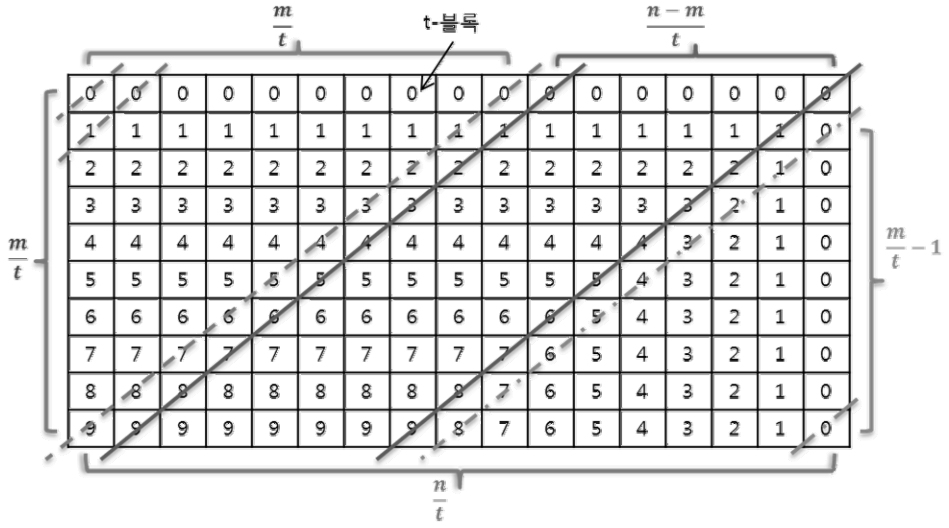

Fig. 8. Parallel computation for the edit distance using the Four-Russians' algorithm

수는 [6]에서 제공하는 SOAP2 프로그램의 함수를 이용하였다. 그리고 실험은 전처리 단계와 계산 단계로 구분하여 실험하였다. 전처리 단계는 t에 대해 룩업테이블을 생성하는 수행시간을 실험하였고, 계산 단계는 두 문자열의 길이를 1,000부터 10,000까지 1,000씩 증가시키면서 4-러시안 알고리즘으로 편집거리를 계산하는 시간을 실험하였다. 그리고 CPU 코드와 GPU 코드의 수행시간을 비교하였다. GPU로 구현한 4-러시안의 수행시간에는 CUDA의 특성상 필요한 호스트 메모리(또는 메인 메모리)와 디바이스 메모리(또는 GPU 메모리) 사이의 데이터를 복사하는 cudaMemcpy() 함수의 수행시간을 포함하였다.

Table 1. Experimental environments

품목	제원
OS	Linux fedora 13 64bit
Kernel	2.6.34.8-68.fc13.x86-64
CPU	Intel i7 970
Main memory	6 GB
GPU	NVIDIA Tesla c2070 (6 GB)
CUDA version	CUDA SDK 3.2
Thread block 수	4096
Block 당 thread 수	256
$\|\Sigma\|$	4

디바이스 메모리는 룩업테이블과 D-테이블의 크기로 인해 전역메모리(global memory)만 사용하였고, 합동 메모리 접근(coalesced memory access)은 4-러시안 알고리즘 특성상 각 쓰레드가 블록 단위로 계산하기 때문에 이용하지 못하였다.

앞으로 GPU 기반의 4-러시안 프로그램을 G4R이라 하고, CPU 기반의 4-러시안 프로그램을 C4R이라 하자.

4.1 전처리 단계의 수행 결과

Table 2. The running time according to block size t

t	시간 (sec)
1	0.000027
2	0.007369
3	2.274071

Table 2는 블록 크기 t가 1에서 3일 때, 전처리 단계의 수행시간이다. t의 크기에 따라 급격하게 수행시간이 증가하였으며 $t \geq 4$일 때는 메모리 부족으로 결과를 측정하지 못했다.

4.2 계산 단계의 수행 결과

계산 단계의 C4R와 G4R의 성능비교는 GPU 메모리 문제로 t가 1, 2일 경우를 각각 실험하였다.

(1) 실험 1: $t=1$인 경우. (Fig. 9 참조)

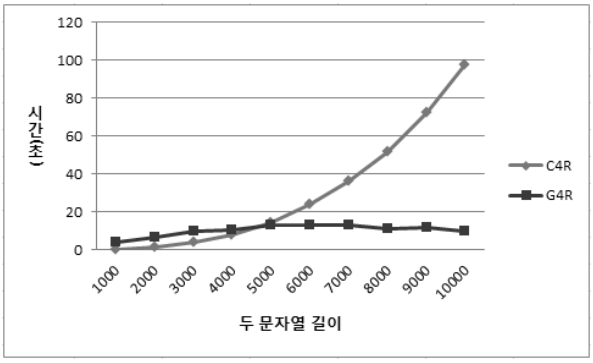

Fig. 9. Performance comparisons between C4R and G4R ($t=1,\ 1{,}000 \leq m, n \leq 10{,}000$)

실험 1에서는 $t=1$일 때, 두 문자열의 길이를 1,000 단위로 증가시키면서 실험하였다. C4R은 수행시간이 선형적으로 증가하는 것을 보이는 반면 G4R은 거의 일정한 수행시간을 보였다. 두 문자열의 길이가 각각 10,000 일 때, G4R의 수행시간은 약 9.78초이고, C4R은 약 97.42초로 약 10배 빠른 실험결과를 보였다.

(2) 실험 2: $t=2$인 경우. (Fig. 10 참조)

실험 2에서는 $t=2$일 때, 실험 1과 동일한 조건으로 실험한 결과를 나타내고 있다. 실험 1의 결과와 비교할 때 G4R의 수행시간은 거의 동일하였지만, C4R의 수행시간이 약 1/3로 감소하였다. 그래도 여전히 입력 크기에 따라 C4R의 수행시간은 선형적으로 증가하였고, G4R의 수행시간은 거의 일정하였다. 두 문자열의 길이가 각각 10,000일 때, G4R은 약 9.98초에 수행되었고, C4R은 약 28.39초에 수행되었다. 즉, G4R이 C4R보다 약 3배 빠른 결과를 보였다.

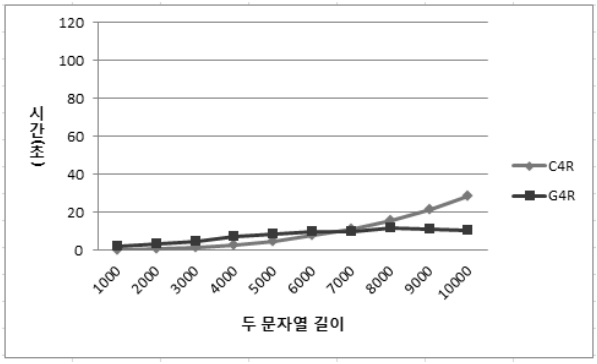

Fig. 10. Performance comparisons between C4R and G4R ($t=2,\ 1{,}000 \leq m, n \leq 10{,}000$)

5. 결 론

본 논문에서는 4-러시안 알고리즘의 계산 단계를 병렬화하여 CUDA로 구현하였다. 블록의 크기 t는 시간과 공간의 트레이드오프(tradeoff) 관계를 나타낸다. 즉, t가 커질수록 전처리 단계에서 많은 시간과 공간이 필요하지만, 계산 단계에서는 빠르게 수행된다. 한편, 4-러시안 알고리즘 특성상, 블록 크기 t에 따라 수행시간 및 필요공간이 급격하게 증가하기 때문에 다양한 블록 크기에 대해 실험하지 못했다. 특히 t = 3일 때, C4R은 정상적으로 수행이 됐으나, G4R은 매우 긴 수행시간으로 인해 결과를 얻지 못했다. t = 4일 때는 C4R에서도 메모리 부족으로 결과를 얻지 못했다. 그러나 작은 t에 대해 G4R은 빠른 수행시간을 보였다. 이론상으로도 m/t개의 쓰레드를 사용했을 때, 계산 단계의 시간복잡도가 $O(mn/t)$에서 $O(m+n)$으로 개선되는 것을 보였다.

향후 전처리 단계의 병렬화에 대한 연구 및 공간복잡도 감소에 대한 연구가 필요할 것이라 판단된다.

참 고 문 헌

[1] A. V. Aho and M. J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search", Communications of the ACM, Vol.18, No.6, 1975.

[2] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer", in Proc.IEEE Symp. Res. Security Privacy, pp.202-212, 1994.

[3] S. B. Needleman and C. D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins", J. Mol. Biol., Vol.48, No.3, pp.443-453, 1970.

[4] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences", J. Mol. Biol. 147, pp.195-197, 1981.

[5] L. L. Cheng, D. W. Cheung, and S. M. Yiu, "Approximate String Matching in DNA Sequences", Proceedings of the Eighth International Conference on Database Systems for Advanced Applications, pp.303-310, 2003.

[6] R. Li, C. Yu, Y. Li, T. W. Lam, S. M. Yui, K. Kristiansen and J. Wang, "SOAP2: an improved ultrafast tool for short read alignment", Bioinformatics, Vol.25, No.15, pp.1966-1967, 2009.

[7] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform", Bioinformatics, Vol.25, No.14, pp.1754-1760, 2009.

[8] B. Langmead, C. Trapnell, M. Pop and S. L Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome", Genome Biology. Vol.10, R25, 2009.

[9] S. Bao, R. Jiang, W. K. Kwan, B. B. Wang, X. Ma and Y. Q. Song, "Evaluation of next-generation sequencing software in mapping and assembly", Journal of Human Genetics, Vol.56, pp.406-414, 2011.

[10] C. M. Liu, T. Wong, E. Wu, R. Luo, S. M. Yiu, Y. Li, B. Wang, C. Yu, X. Chu, K. Zhao, R. Li, T. W. Lam, "SOAP3: ultra-fast GPU-based parallel alignment tool for short reads", Bioinformatics, Vol.28, No.6, pp.878-879, 2012.

[11] D. Gusfield, Algorithms on Strings, Trees, and Sequences, pp.302-307, Cambridge university press, 1997.

[12] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals", Sov. Phys. Dokl, Vol.10, pp.707-710, 1966.

[13] W. J. Masek and M. S. Paterson, "A Faster Algorithm Computing String Edit Distance", Journal of computer and system science, Vol.20, No.1, pp.18-31, 1980.

[14] V. Kundeti and S. Rajasekaran, "Extending the Four Russian Algorithm to Compute the Edit Script in Linear Space", ICCS 2008, Part I, LNCS 5101, pp.893-902, 2008.

[15] M. C. Schatz and C. Trapnell, "Fast Exact String Matching on the GPU", Technical report of Center for Bioinformatics and Computational Biology.

[16] C. Trapnell and M. C. Schatz, "Optimizing data intensive GPGPU computations for DNA sequence alignment", Parallel Computing, Vol.35, No.8-9, pp.429 - 440, 2009.

[17] A. Gharaibeh and M. Ripeanu. "Size Matters: Space/Time Tradeoffs to Improve GPGPU Applications Performance", In IEEE/ACM Supercomputing (SC 2010), pp.1-12, 2010.

[18] Hyun Chul Yoon, Jeong Seop Sim, "Parallel Construction for the Graph Model of the Longest Common Non-superstring using CUDA", KIISE Journal, System and Theory, Vol.39, No.3, pp.202-208, 2012.

[19] L. Ligowski, W. Rudnicki, "An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases", in Parallel & Distributed Processing (IPDPS). IEEE Int. Symp., pp.1-8, 2009.

[20] R. Hughey, "Parallel hardware for sequence comparison and alignment", Comput. Applic. Biosci. Vol.12, pp.473-479, 1996.

김 영 호

e-mail : yhkim8505@gmail.com

2011년 인하대학교 컴퓨터정보공학부 (학사)

2011년~현 재 인하대학교 컴퓨터정보공학과 석사과정

관심분야 : String Algorithm, Parallel Algorithm

정 주 희

e-mail : jngjuhe@gmail.com
2011년 인하대학교 컴퓨터정보공학부(학사)
2011년~현 재 인하대학교 컴퓨터정보 공학과 석사과정
관심분야 : String Algorithm, Bioinformatics

강 대 웅

e-mail : kdw8219@gmail.com
2012년 인하대학교 컴퓨터정보공학부(학사)
2012년~현 재 인하대학교 컴퓨터정보 공학과 석사과정
관심분야 : String Algorithm, Parallel Algorithm

심 정 섭

e-mail : jssim@inha.ac.kr
1995년 서울대학교 컴퓨터공학과(학사)
1997년 서울대학교 컴퓨터공학과(석사)
2002년 서울대학교 컴퓨터공학과 (Ph.D., 박사)
2002년 3월~2004년 8월 한국전자통신 연구원 바이오정보연구팀 선임연구원
2004년 9월~현 재 인하대학교 컴퓨터정보공학부 부교수
관심분야 : Theory of Computation, Algorithm, Bioinformatics