KIPS Tr. Comp. and Comm. Sys.
Vol.4, No.7 pp.213~218 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2015.4.7.213

Parallel Computation for Extended Edit Distances Using the Shared Memory on GPU

Youngho Kim† · Joong Chae Na†† · Jeong Seop Sim†††

ABSTRACT

Given two strings X and Y ($|X|=m$, $|Y|=n$) over an alphabet Σ, the extended edit distance between X and Y can be computed using dynamic programming in $O(mn)$ time and space. Recently, a parallel algorithm that takes $O(m+n)$ time and $O(mn)$ space using m threads to compute the extended edit distance between X and Y was presented. In this paper, we present an improved parallel algorithm using the shared memory on GPU. The experimental results show that our parallel algorithm runs about 19~25 times faster than the previous parallel algorithm.

Keywords : Approximate String Matching, CUDA, Extended Edit Distance, Shared Memory

GPU의 공유메모리를 활용한 확장편집거리 병렬계산

김 영 호† · 나 중 채†† · 심 정 섭†††

요 약

알파벳 Σ로 구성된 길이가 각각 m, n인 두 문자열 X, Y가 주어졌을 때, X, Y의 확장편집거리는 동적프로그래밍을 이용하여 $O(mn)$ 시간과 공간을 계산할 수 있다. 최근 m개의 쓰레드를 이용하여 $O(m+n)$ 시간과 $O(mn)$ 공간을 사용하여 X, Y의 확장편집거리를 계산하는 병렬알고리즘이 제시되었다. 본 논문에서는 GPU의 공유메모리를 활용하여 수행시간을 개선한 병렬알고리즘을 제시한다. 실험 결과, 개선된 병렬알고리즘이 기존의 병렬알고리즘보다 약 19~25배 이상 빠른 수행시간을 보였다.

키워드 : 근사문자열매칭, CUDA, 확장편집거리, 공유메모리

1. 서 론

문자열의 오차를 허용하는 근사문자열매칭(approximate string matching) 알고리즘은 검색엔진[1], 컴퓨터보안[2, 3], 생물정보학[4, 5] 등 다양한 분야에서 연구되고 있다. 근사문자열매칭은 거리함수를 이용하여 문자열의 오차를 측정한다. 알파벳 Σ의 문자들로 구성된 길이가 각각 m, $n(m \leq n)$인 두 문자열 X와 Y의 거리 $\delta(X, Y)$는 X를 Y로 변환하기 위해 필요한 최소 비용으로 정의된다. 대표적인 거리함수로는 해밍거리(Hamming distance), 편집거리(edit distance), 확장편집거리(extended edit distance) 등이 있다. 두 문자열 X와 Y에 대한 해밍거리는 두 문자열의 길이가 같을 때 X를 Y로 변환하기 위해 필요한 최소 교체(change)연산의 수이다. 두 문자열의 편집거리는 X를 Y로 변환하기 위해 필요한 최소 편집연산의 수이다. 이때 편집연산은 삽입(insertion), 삭제(deletion), 교체연산으로 구성된다. 한편, 기존의 편집연산에서 두 문자의 위치를 변경하는 교환(swap)연산을 추가한 확장편집연산이 있다. X와 Y의 확장편집거리는 X를 Y로 변환하기 위해 필요한 최소 확장편집연산의 수이다.

두 문자열의 확장편집거리 문제는 NP-완전(NP-complete)이며, 제한된 경우에 다항시간에 해결될 수 있다[6]. [7]에서는 특정 조건을 만족할 때 확장편집거리를 동적프로그래밍을 이용하여 $O(mn)$ 시간과 공간을 이용하여 계산하는 알고리즘을 제시하였다. 같은 조건으로 [8]에서 t가 X와 Y의 확장편집거리라고 할 때, 동적프로그래밍 테이블을 이용하여 $O(t \min(m,n))$ 시간과 $O(t^2)$ 공간에 확장편집거리를 계산하는 알고리즘을 제시하였다.

최근 GPU의 성능이 향상되어, 문자열매칭 분야에서도 GPU를 활용한 병렬화 연구가 활발해지고 있다[5, 9, 10]. [5]에서는 염기서열데이터베이스를 빠르게 탐색하기 위해 GPU에 최적화된 Smith-Waterman 알고리즘을 구현하여 CPU의 수행시간보다 약 16배 향상시켰다. [9]에서는 Aho-Corasick 오토마타를 이용한 그래프 모델을 GPU 기반으로 병렬적으로 생성하여 최

※ 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2014R1A1A1004901, 2014R1A2A1A11050337).
※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발 사업의 일환으로 수행하였음. [B0101-15-0104, 유전체 분석용 슈퍼컴퓨팅 시스템 개발]
† 종신회원 : 인하대학교 컴퓨터정보공학과 박사과정
†† 종신회원 : 세종대학교 컴퓨터공학과 부교수
††† 종신회원 : 인하대학교 컴퓨터정보공학과 교수
Manuscript Received : February 16, 2015
First Revision : April 21, 2015
Accepted : May 7, 2015
* Corresponding Author : Jeong Seop Sim(jssim@inha.ac.kr)

장공통비상위문자열(longest common non-superstring) 문제를 해결하였다. [10]에서는 [7, 8]의 특정조건에서 GPU를 이용하여 확장편집거리 문제를 $O(m+n)$ 시간과 $O(mn)$ 공간을 이용하여 해결하는 병렬알고리즘을 제시하였다. [10]에서는 X와 Y의 확장편집거리를 계산하기 위해 X에 존재하는 문자들의 정보를 미리 계산하는 전처리단계와 동적프로그래밍 테이블을 계산하는 계산단계를 각각 병렬적으로 계산하였다. 또한 [10]에서는 GPU의 블록(또는 쓰레드 블록)을 하나만 사용하고 전역메모리(global memory)만을 이용하여, 두 문자열의 길이가 1,000일 때 병렬알고리즘이 순차알고리즘보다 약 12배 빠르게 수행됨을 보였다.

본 논문에서는 확장편집거리를 계산하기 위한 전처리단계와 계산단계에 대해 CUDA의 특성을 고려하여, 여러 개의 블록과 공유메모리(shared memory)를 활용한 효율적인 병렬알고리즘을 제시한다. 다중서열배치(multiple sequence alignment)와 같이 여러 개의 문자열 쌍에 대해 확장편집거리를 계산할 때, 각 문자열 쌍은 독립적으로 계산될 수 있기 때문에, 블록을 하나씩 할당하여 동시에 계산한다. 또한 동적프로그래밍 테이블의 일부를 블록의 공유메모리에도 저장함으로써 병렬알고리즘의 수행시간을 개선한다. 실험 결과, 개선된 병렬알고리즘이 기존의 병렬알고리즘보다 약 10~14배 빠르게 수행되었다. 한편, GPU에서 계산된 확장편집거리들을 메인메모리에 복사하지 않고 바로 콘솔로 출력하면, 기존의 병렬알고리즘보다 약 19~25배 이상 빠르게 수행되었다.

본 논문의 구성은 다음과 같다. 2절에서는 본 논문의 선행연구에 대해 설명하고, 3절에서는 본 논문에서 제시하는 알고리즘에 대해 설명한다. 4절에서는 기존의 순차알고리즘과 병렬알고리즘, 본 논문에서 제시하는 병렬알고리즘의 수행시간을 비교한다. 5절에서는 결론을 제시한다.

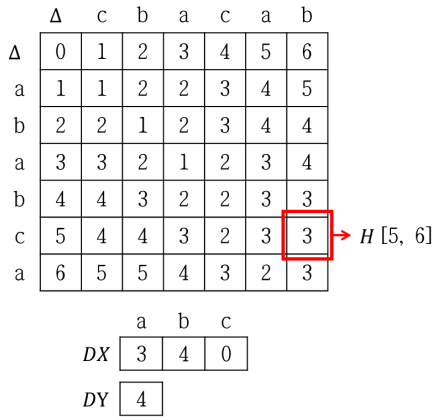

Fig. 1. H-table for $X=ababca$ and $Y=cbacab$ and DX, DY when Computing $H[5,6]$ over $\Sigma=\{a,b,c\}$ [10]

2. 관련 연구

알파벳 Σ의 문자들로 구성된 길이 m인 문자열들의 집합을 Σ^m으로 나타낸다. 문자열 $X(\in\Sigma^m)$의 i번째 문자$(1\le i\le m)$는 $X[i]$로 표기한다. X의 i번째부터 j번째$(1\le i\le j\le m)$까지의 부분문자열인 $X[i]X[i+1]...X[j]$는 $X[i..j]$로 표기하고, X의 길이는 $|X|$로 표현한다. Σ의 크기는 $|\Sigma|$로 표기한다. $X[0]$ 또는 $X[i..(i-1)]$은 공백문자 Δ를 나타낸다고 가정한다. 크기가 m인 배열 A에서 $A[i]$부터 $A[j]$까지 연속된 부분배열은 $A[i:j]$ $(0\le i\le j<m)$로 표현하고, 크기가 $m\times n$인 2차원 배열 B의 i행, j열의 원소는 $B[i,j](0\le i<m,\ 0\le j<n)$로 표현한다.

2.1 확장편집거리 계산 방법

두 문자열 $X(\in\Sigma^m)$와 $Y(\in\Sigma^n)$의 확장편집거리는 X를 Y로 변환하기 위해 필요한 최소 확장편집연산의 수이며, 잘 알려진 동적프로그래밍을 이용하여 계산할 수 있다[7, 8]. 이때 확장편집연산은 삽입, 삭제, 교체, 교환연산으로 구성되며, 계산할 때 사용하는 $(m+1)\times(n+1)$ 크기의 동적프로그래밍 테이블을 H-테이블이라 하자. H-테이블의 $H[i,j]$는 $X[1..i]$와 $Y[1..j]$의 확장편집거리를 저장한다. 따라서 $H[m,n]$이 두 문자열 X와 Y의 확장편집거리이다.

H-테이블을 계산하는 방법은 다음과 같다. 먼저 첫 번째 열과 행에 대해 $H[i,0]=i(0\le i\le m)$, $H[0,j]=j(1\le j\le n)$로 초기화한다. 이후 각 $H[i,j]$는 Equation (1)로 계산된다.

$$H[i,j]=\min\begin{cases}H[i,j-1]+1,\\ H[i-1,j]+1,\\ H[i-1,j-1]+\delta(X[i],Y[j]),\\ s(i,j)\end{cases}\qquad(1)$$

단, $s(i,j)=H[p_{ij}-1,q_{ij}-1]+(i-p_{ij}-1)+(j-q_{ij}-1)+1$

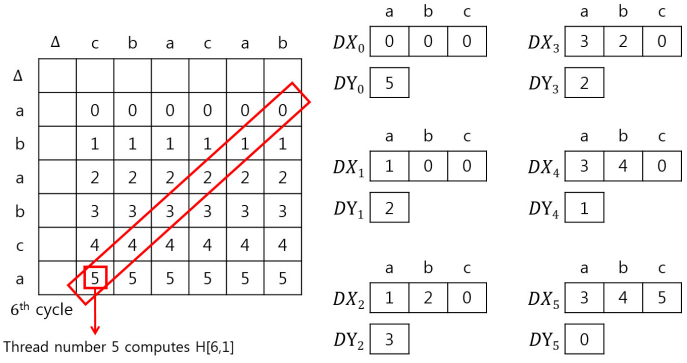

Fig. 2. Thread Allocation for Computing H-table in Parallel when $X=ababca$ and $Y=cbacab$ and DX_t, DY_t at the 6th Cycle [10]

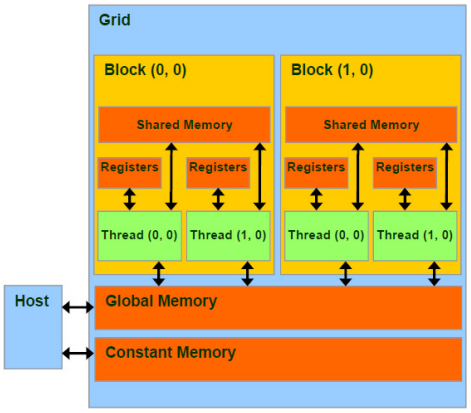

Fig. 3. CUDA Memory Model [11]

$\delta(X[i], Y[j])$는 $X[i]=Y[j]$이면 0, 그렇지 않으면 1이다. 이때 $s(i,j)$는 교환연산을 의미한다. $s(i,j)$에 나타나는 p_{ij}와 q_{ij}는 $H[i,j]$에서 교환연산이 발생할 수 있는 위치를 나타낸다. 즉, p_{ij}는 $Y[j]$가 $X[1..(i-1)]$ 상에 나타나는 마지막 위치를 나타내고, q_{ij}는 $X[i]$가 $Y[1..(j-1)]$ 상에 나타나는 마지막 위치를 나타낸다. 예를 들어, Fig. 1에서 $H[5,6]$을 계산할 때, $Y[6]$인 b가 $X[1..4]$ 상에서 마지막에 나타나는 위치는 4이므로 $p_{5,6}=4$이고, $X[5]$인 c가 $Y[1..5]$ 상에서 마지막에 나타나는 위치가 4이므로 $q_{5,6}=4$이다.

H-테이블을 행 우선으로 계산한다고 가정하자. $X[i]$는 한 행을 계산할 때까지 고정되지만, $Y[j]$는 매번 변경된다. 따라서 p_{ij} 정보를 저장할 $|\Sigma|$ 크기의 배열 DX와 q_{ij}를 저장할 변수 DY가 필요하다. $|\Sigma|$ 가 상수일 때 DX와 DY를 이용하면, $s(i,j)$도 $O(1)$ 시간에 계산할 수 있다.

각 $H[i,j]$는 Equation (1)을 이용하여 $O(1)$ 시간에 계산할 수 있고, 테이블의 크기가 $(m+1)\times(n+1)$이므로 $O(mn)$ 시간과 공간을 이용하여 H-테이블을 계산할 수 있다. Fig. 1은 X="$ababca$"와 Y="$cbacab$"에 대한 H-테이블과 $H[5,6]$을 계산할 때의 DX, DY를 보여준다.

2.2 기존의 확장편집거리 병렬계산 방법

Equation (1)에서 알 수 있듯이, $H[i,j]$를 계산하기 위해서는 $H[i,j-1]$, $H[i-1,j-1]$, $H[i-1,j]$, $H[p_{ij}-1,q_{ij}-1](1\le p_{ij}<i, 1\le q_{ij}<j)$이 먼저 계산되어야 한다. [10]에서는 이러한 데이터 종속성 문제를 해결하기 위해, Fig. 2와 같이 오른쪽 위에서 왼쪽 아래로 향하는 대각선(이하 대각선)을 우선으로 병렬적으로 계산하였다. 한 대각선이 동시에 계산되는 단계를 한 사이클이라 하자. Fig. 2의 각 셀의 번호는 각 사이클에 배치되는 쓰레드의 번호를 나타낸다. 각 쓰레드는 하나의 행을 처리한다. 병렬계산을 위해 각 쓰레드는 따로 DX와 DY를 관리한다. 쓰레드 t가 관리하는 DX와 DY를 각각 DX_t, DY_t라 하자. Fig. 2는 6번째 사이클일 때의 DX_t와 DY_t를 보여준다.

전처리단계는 각 행에 대한 DX 값을 미리 계산한다. 순차알고리즘에서는 순차적으로 계산하여 DX가 적절히 갱신되지만, 병렬알고리즘에서는 쓰레드 t가 $(t+1)$행을 바로 계산해야 하기 때문에 이러한 전처리단계가 필요하다.

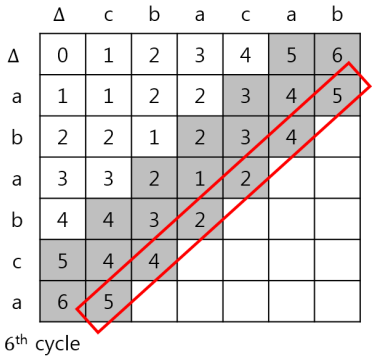

Fig. 4. A Part of H-table Stored in the Shared Memory when Computing Elements in Parallel at the 6th Cycle (shaded elements)

[10]에서는 전처리단계 계산을 위한 순차계산법과 병렬계산법을 소개하였다. 순차계산법은 CPU를 이용하여 계산하며, DX_0은 모두 0으로 할당하고, DX_t를 계산할 때 앞서 계산된 DX_{t-1}를 복사한 뒤, DX_t의 $X[t]$에 대한 정보만 갱신한다. 병렬계산법은 GPU를 이용하며, 각 쓰레드 $t(0\le t<m)$가 DX_t를 모두 0으로 초기화하고 $X[1]$부터 $X[t]$까지 각 문자를 확인하면서 위치정보를 갱신한다. 위의 두 방법은 $|\Sigma|$ 가 상수일 때, $O(m)$ 시간에 계산할 수 있다.

계산단계는 전처리단계에서 계산된 DX_t를 이용하여 Fig. 2와 같이 대각선 단위로 동시에 계산한다. 따라서 한 사이클은 $O(1)$ 시간에 계산할 수 있고, 총 $m+n-1$개의 사이클을 계산하기 때문에 기존의 병렬알고리즘은 $O(m+n)$ 시간과 $O(mn)$ 공간을 이용하여 수행된다.

3. 개선된 확장편집거리 병렬계산

3.1 CUDA의 특성

본 논문에서는 수행시간을 개선하기 위해 NVIDIA의 CUDA(compute unified device architecture)를 사용하였다. 본 논문에서 사용한 CUDA의 메모리 아키텍처는 Fig. 3과 같이 전역메모리, 상수메모리(constant memory), 텍스처메모리(texture memory), 공유메모리, 레지스터(register)로 구성되어있다. 전역메모리, 상수메모리, 텍스처메모리는 하나씩만 존재하며 모든 쓰레드가 접근할 수 있다. 공유메모리는 각 블록마다 존재하며, 블록에서 사용 가능한 쓰레드들만 접근할 수 있다. 공유메모리는 L1 캐시와 온 칩으로 되어있어 쓰레드들이 빠르게 접근할 수 있다. 본 논문에서 사용하는 NVIDIA Geforce GTX 660 모델은 Kepler 아키텍처로 약 2GB의 전역메모리, 64KB의 상수메모리, 블록당 48KB의 공유메모리를 사용할 수 있다. 블록당 이용 가능한 레지스터 수는 65,536개이며, 쓰레드당 최대 255개의 레지스터에 접근할 수 있다. 공유메모리는 메모리의 크기는 작지만, 빠

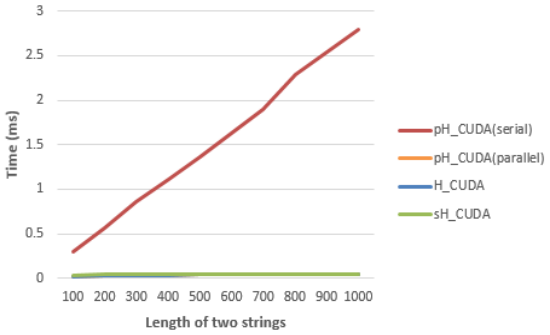

Fig. 5. Comparison of Running Times of the Preprocessing Steps for 100 pairs of Random Strings ($|\Sigma|=20,\ 100 \le m=n \le 1{,}000$)

Table 1. Running Times of Algorithms in Fig. 4 when $m=n=100$ and $m=n=1{,}000$ (ms)

	$m=n=100$	$m=n=1{,}000$
pH_CUDA(serial)	0.344	2.8
pH_CUDA(parallel)	0.018	0.046
H_CUDA	0.017	0.045
sH_CUDA	0.028	0.047

른 접근이 가능하기 때문에, 이를 활용하면 알고리즘의 수행시간을 개선할 수 있다.

CUDA의 Kepler 아키텍처는 8개의 SMX(streaming multiprocessor)로 구성되어 있으며, 각 SMX는 4개의 워프스케줄러(warp scheduler)를 포함한다. 각 워프스케줄러는 32개의 쓰레드들로 구성된 그룹인 워프(warp) 단위로 스케줄링을 하는데 이를 합병메모리접근(coalesced memory access)이라 한다. 합병메모리접근은 한 명령어를 수행할 때 워프단위로 연속된 메모리 주소에 접근하기 때문에 이를 통해 수행시간을 개선할 수 있다.

3.2 공유메모리를 활용한 확장편집거리 병렬계산 알고리즘

본 논문에서는 [10]의 전처리단계와 계산단계의 수행시간을 개선한다. 전처리단계는 [10]에서 소개한 병렬계산법을 공유메모리와 후향탐색(backward search) 방법을 이용하여 개선한다. 공유메모리에 문자열 X와, 각 DX_t를 할당하고, 후향탐색 방법을 다음과 같이 적용한다. 먼저 각 쓰레드 t $(0 \le t < m)$는 DX_t의 모든 원소를 모두 0으로 초기화한다. 이후 $X[t]$부터 $X[1]$까지 탐색하면서 각 문자의 위치정보가 0이면 위치정보를 갱신한다. 이때 갱신된 횟수가 $|\Sigma|$ 이면, 해당 쓰레드는 전처리단계를 종료한다. [10]의 방법은 모든 쓰레드가 $X[1]$부터 동시에 접근하여, 병목현상이 발생하지만, 이 방법은 각 쓰레드 t가 $X[t]$부터 접근하기 때문에, 병목현상이 발생하지 않는다. 전처리단계의 시간복잡도는 최악의 경우 [10]과 같이 $O(m)$이다.

계산단계에서는 CUDA의 여러 개의 블록과 공유메모리를 이용한다. [10]에서는 하나의 블록만 이용했지만, 본 논문에서는

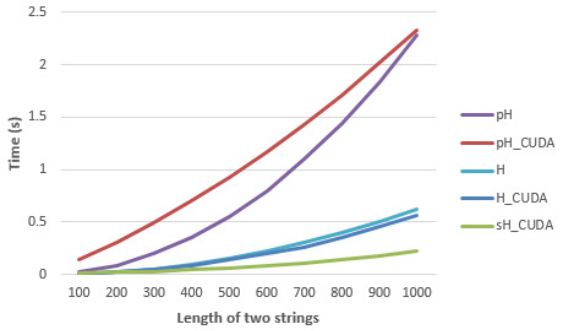

Fig. 6. Comparison of running times of all steps for 100 pairs of random strings ($|\Sigma|=20,\ 100 \le m=n \le 1{,}000$)

Table 2. Running times of parallel algorithms in Fig. 5 when $m=n=100$ and $m=n=1{,}000$ (s)

	$m=n=100$	$m=n=1{,}000$
pH_CUDA	0.1436	2.331
H_CUDA	0.0059	0.567
sH_CUDA	0.0102	0.221

여러 개의 블록을 이용하여 문자열 쌍마다 하나의 블록을 할당하여 각 문자열 쌍에 대해 확장편집거리를 동시에 계산한다. 공유메모리는 두 문자열 X와 Y, 각 DX_t, H-테이블의 일부를 저장하는 데 이용한다. 앞서 말한 바와 같이, $H[i,j]$를 계산하기 위해서는 $H[i,j-1]$, $H[i-1,j-1]$, $H[i-1,j]$, $H[p_{ij}-1,q_{ij}-1]$가 필요하다. 하지만, CUDA의 Kepler 아키텍처에서는 공유메모리를 최대 48KB를 사용할 수 있기 때문에, H-테이블 전체를 공유메모리에 적재할 수 없다. 따라서 본 논문에서는 Fig. 4와 같이 $3(m+1)$ 크기의 배열을 할당하여 $H[i,j-1]$, $H[i-1,j-1]$, $H[i-1,j]$에 대한 정보를 공유메모리에서도 관리하고, $H[p_{ij}-1, q_{ij}-1]$를 참조하기 위해 전체 H-테이블을 전역메모리에서 관리한다. 이를 통해, H-테이블에 대한 전역메모리의 참조횟수를 5번에서 2번으로 줄일 수 있으며, 대각선의 셀들에 저장된 확장편집거리를 연속된 메모리에 저장할 수 있으므로 합병메모리 접근이 가능하다. 계산단계는 [10]과 같이 $O(m+n)$ 시간과 $O(mn)$ 공간을 이용하여 수행된다.

4. 실험 결과

실험환경은 다음과 같다. CPU는 Intel Core i7-3820, RAM은 32GB, GPU는 NVIDIA Geforce GTX 660, CUDA SDK 버전은 5.5, OS는 Windows 8(64bit)을 사용하였다. 실험 데이터의 $|\Sigma|=20$이며 무작위로 길이를 100부터 1,000까지 100씩 증가시키며 생성한 같은 길이의 문자열 100쌍을 이용하였다. $|\Sigma|=20$ 일 때, 두 문자열에 대해 약 2KB, DX_t에 대해 약 40KB, H-테이블의 일부를 저장하는데 약 6KB로 거의 공유메모리의 최대 크기를 사용하고 있기 때문에 문자열의 길이는 1,000까지 설정하였다. 측정된 시간은 문자

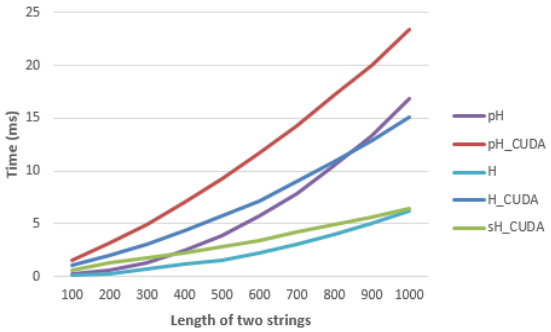

Fig. 7. Comparison of Running Times of all Steps for a pair of Random Strings ($|\Sigma|=20,\ 100 \le m=n \le 1{,}000$)

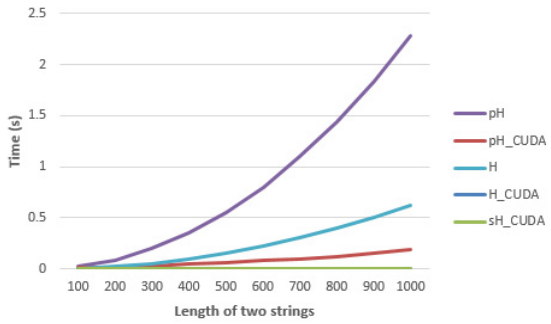

Fig. 8. Comparison of Running Times of all Steps for 100 pairs of Random Strings ($|\Sigma|=20,\ 100 \le m=n \le 1{,}000$)

Table 3. Running Times of Parallel Algorithms in Fig. 6 when $m=n=100$ and $m=n=1{,}000$ (ms)

	$m=n=100$	$m=n=1{,}000$
pH_CUDA	1.6	23.43
H_CUDA	1.14	15.05
sH_CUDA	0.67	6.49

Table 4. Running Times of Parallel Algorithms in Fig. 7 when $m=n=100$ and $m=n=1{,}000$ (s)

	$m=n=100$	$m=n=1{,}000$
pH_CUDA	0.0112	0.1846
H_CUDA	0.0006	0.005
sH_CUDA	0.0004	0.0075

열 100쌍의 확장편집거리를 100번 계산한 시간의 평균이며, 디바이스(GPU) 메모리와 메인메모리 사이에 데이터를 복사하는 cudaMemcpy() 함수의 수행시간을 포함하였다. [10]에서 구현한 순차알고리즘(pH)과 병렬알고리즘(pH_CUDA), 본 논문에서 제시하는 순차알고리즘(H), 전역메모리만 사용한 병렬알고리즘(H_CUDA)과 공유메모리까지 사용한 병렬알고리즘(sH_CUDA)에 대해 수행시간을 비교하였다. 이때 pH와 pH_CUDA는 [10]의 프로그램을 사용하였다.

1) 실험 1 : 전처리단계 성능비교

Fig. 5는 무작위로 생성된 문자열 100쌍에 대한 전처리단계의 수행시간이다. pH_CUDA(serial)는 순차계산법, pH_CUDA(parallel)는 병렬계산법을 각각 나타낸다. 전처리된 결과를 디바이스로 복사하는 cudaMemcpy() 시간은 포함하지 않았다. sH_CUDA에서 공유메모리는 X와 각 DX_t를 저장하는데 이용하였다. Table 1은 두 문자열의 길이가 100과 1,000일 때, 각 알고리즘의 수행시간을 보여준다. Table 1을 보면 알 수 있듯이, pH_CUDA(parallel), H_CUDA, 그리고 sH_CUDA는 수행시간이 거의 차이가 나지 않는다(이로 인해 Fig. 5에서 세 개의 선이 겹쳐 보인다). H_CUDA와 sH_CUDA는 pH_CUDA(serial)보다 16~63배 정도 빠르게 수행되었다. 한편, [10]의 결과와 달리 pH_CUDA(parallel)가 pH_CUDA(serial)보다 빠르게 수행되었으며, H_CUDA, sH_CUDA와 유사한 속도를 보였다. 이는 각 DX_t를 계산할 때, X가 캐시에 저장되어 빠르게 계산되었기 때문으로 판단된다.

2) 실험 2 : 공유메모리 활용

Fig. 6은 무작위로 생성된 문자열 100쌍의 확장편집거리를 계산하는 데 소비되는 총 수행시간을 보여준다. CUDA로 구현된 병렬알고리즘들에서 계산된 확장편집거리 값들에 대해 디바이스메모리에서 메인메모리로 cudaMemcpy()를 수행한 시간을 포함하였다. 앞서 설명한 바와 같이, sH_CUDA에서 공유메모리는 X와 Y, 각 DX_t, H-테이블의 일부를 저장하는데 이용하였다. Table 2는 두 문자열의 길이가 100과 1,000일 때, Fig. 6의 각 병렬알고리즘들의 수행시간을 보여준다. sH_CUDA는 pH_CUDA보다 약 10~14배 빠르게 수행되었다. 한편, 두 문자열의 길이가 1,000일 때, sH_CUDA는 전역메모리의 접근 횟수를 줄여 H_CUDA보다 약 2.6배 빠르게 수행되었다.

3) 실험 3 : 멀티블록 활용

Fig. 7은 멀티블록을 활용한 성능 향상을 확인하기 위해, 무작위로 생성된 문자열 한 쌍에 대해 실험한 결과를 보여준다. (앞서 언급한 바와 같이 [10]에서는 한 개의 블록을 이용하였다.) 실험 2와의 비교를 통해 멀티블록으로 인한 성능 향상 정도를 파악할 수 있을 것이다. Table 3은 두 문자열의 길이가 100과 1,000일 때, Fig. 7의 병렬알고리즘들의 실제 수행시간을 보여준다. H_CUDA는 pH_CUDA보다 약 1.4~1.6배 빠르게 수행되었고, sH_CUDA는 pH_CUDA보다 약 2.4~3.6배 빠르게 수행되었다. 두 문자열의 길이가 1,000일 때 실험 2와 비교하면 다음과 같다. 실험 2에서 pH_CUDA의 수행시간은 실험 3의 수행시간보다 문자열 쌍의 개수에 비례하여 약 100배 느린 수행시간을 보인다. 한편, 실험 2에서 sH_CUDA의 수행시간은 실험 3에서의 수행시간에서 문자열 쌍의 개수에 비례한 0.649초가 아닌, 0.221초로 약 2.9배 향상되었다. 이는 문자열 쌍의 개수만큼 멀티블록을 이용하여 얻은 성능 향상으로 볼 수 있다.

4) 실험 4 : 계산된 확장편집거리를 printf()로 처리한 성능 비교

Fig. 8은 실험 2와 동일한 환경에서 [10]과 같이, 계산된

확장편집거리들을 cudaMemcpy()를 수행하여 메인메모리로 복사하는 대신 printf()를 이용하여 출력한 실험 결과를 보여준다. Table 4는 두 문자열의 길이가 100과 1,000일 때, Fig. 8의 병렬알고리즘들의 실제 수행시간을 보여준다. Table 4를 보면 알 수 있듯이, H_CUDA와 sH_CUDA의 수행시간은 거의 차이가 나지 않는다(이로 인해 Fig. 8에서 두 개의 선이 겹쳐 보인다). [10]의 결과와 유사하게 pH_CUDA는 pH보다 약 12배 빠르게 수행되었다. H_CUDA와 sH_CUDA는 pH_CUDA보다 약 19~25배 이상 빠르게 수행되었다. 한편, 두 문자열의 길이가 1,000일 때, sH_CUDA는 H_CUDA보다 약 1.5배 느린 수행시간을 보였으며, 이때 공유메모리의 할당과 초기화에 대한 시간이 약 0.00071초의 수행시간을 보였다. sH_CUDA는 실험 2와 비교했을 때, 약 29배 빠르게 수행되었다.

5. 결 론

본 논문은 GPU의 특성을 고려하여 기존의 확장편집거리를 계산하는 병렬알고리즘보다 효율적인 병렬알고리즘을 제시하였다. 실험을 통해 멀티블록과 공유메모리를 사용함에 따라 향상되는 수행시간을 분석하였다. 본 논문에서 제시한 병렬알고리즘은 생물정보학 분야에서 긴 레퍼런스 서열에 상대적으로 짧은 리드 서열을 배치하는 레퍼런스 매핑이나 컴퓨터보안 분야에서 악성패킷을 검출하기 위한 침입탐지시스템 연구에 적용될 수 있을 것이다. 향후 본 논문에서 연구한 문자단위의 교환연산뿐만 아니라, 여러 문자를 한꺼번에 교환하는 연산에 대한 연구도 필요할 것으로 판단된다.

References

[1] R. Baeza-Yates, G. Navarro, "Fast approximate string matching in a dictionary," *Proc. South Am. Symp. String Proc. Information Retrieval (SPIR '98)*, pp.14-22, 1998.

[2] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer," *Proc. of the IEEE Symposium on Research in Security and Privacy*, pp.202-212, 1994.

[3] M. Roesch, "Snort-Lightweight Intrusion Detection for Networks," in *Proceedings of the 13th Conference on Systems Administration(LISA '99)*, pp.229-238, 1999.

[4] T. F. Smith, M. S. Waterman, "Identification of Common Molecular Subsequences," *Journal of molecular biology*, Vol.147, No.1, pp.195-197, 1981.

[5] W. Liu, B. Schmidt, G. Voss, A. Schroder, and W. Muller-Wittig, "Bio-Sequence Database Scanning on a GPU," *20th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006) (HICOMB Workshop)*, Rhode Island, Greece, 2006.

[6] R. A. Wagner, "On the complexity of the extended string-to-string correction problem," *Proc. of seventh annual ACM symposium on Theory of computing*, pp.218-223, 1975.

[7] R. Lowrance, R. A. Wagner, "An extension of the string-to-string correction problem," *Journal of the ACM*, Vol.22, No.2, pp.177-183, 1975.

[8] D. K. Kim, J. S. Lee, K. Park, and Y. Cho, "Efficient algorithms for approximate string matching with swaps," *Journal of complexity*, Vol.15, No.1, pp.128-147, 1999.

[9] H. C. Yoon, J. S. Sim, "Parallel Construction for the Graph Model of the Longest Common Non-superstring using CUDA," *Journal of the KIISE*, Vol.39, No.3, pp.202-208, 2012.

[10] D. W. Kang, Y. Kim, and J. S. Sim, "Parallel Computation for Extended Edit Distances Including Swap Operations," *Journal of KIISE: Computer Systems and Theory*, Vol.41, No.4, pp.175-181, 2014.

[11] D. Kirk, W. W. Hwu, Lectures 5 and 6: Memory model and Locality [Internet], https://courses.engr.illinois.edu/ece408/Lectures.html.

김 영 호

e-mail : yhkim8505@gmail.com

2011년 인하대학교 컴퓨터정보공학부(학사)

2013년 인하대학교 컴퓨터정보공학과(석사)

2013년~현 재 인하대학교 컴퓨터정보공학과 박사과정

관심분야 : String Algorithm, Parallel Algorithm

나 중 채

e-mail : jcna@sejong.ac.kr

1998년 서울대학교 컴퓨터공학과(학사)

2000년 서울대학교 컴퓨터공학과(석사)

2005년 서울대학교 전기컴퓨터공학부 (박사)

2006년 헬싱키대학교 박사후 연구원

2007년 건국대학교 연구교수

2008년~현 재 세종대학교 컴퓨터공학과 부교수

관심분야 : 알고리즘, 바이오정보공학, 컴퓨터이론, 스트링매칭

심 정 섭

e-mail : jssim@inha.ac.kr

1995년 서울대학교 컴퓨터공학과(학사)

1997년 서울대학교 컴퓨터공학과(석사)

2002년 서울대학교 컴퓨터공학과(박사)

2002년~2004년 한국전자통신연구원 바이오정보연구팀 선임연구원

2004년~현 재 인하대학교 컴퓨터정보공학과 교수

관심분야 : Theory of Computation, Algorithm, Bioinformatics