ISSN 1975-8359(Print) / ISSN 2287-4364(Online)
The Transactions of the Korean Institute of Electrical Engineers Vol.67, No.5, pp. 656~662, 2018
http://doi.org/10.5370/KIEE.2018.67.5.656

CPU-GPU 메모리 계층을 고려한 고처리율 병렬 KMP 알고리즘

High Throughput Parallel KMP Algorithm Considering CPU-GPU Memory Hierarchy

박 소 은*· 김 대 희*· 이 명 호**· 박 능 수†
(Soeun Park · Daehee Kim · Myungho Lee · Neungsoo Park)

Abstract - Pattern matching algorithm is widely used in many application fields such as bio-informatics, intrusion detection, etc. Among many string matching algorithms, KMP (Knuth-Morris-Pratt) algorithm is commonly used because of its fast execution time when using large texts. However, the processing speed of KMP algorithm is also limited when the text size increases significantly. In this paper, we propose a high throughput parallel KMP algorithm considering CPU-GPU memory hierarchy based on OpenCL in GPGPU (General Purpose computing on Graphic Processing Unit). We focus on the optimization for the allocation of work-times and work-groups, the local memory copy of the pattern data and the failure table, and the overlapping of the data transfer with the string matching operations. The experimental results show that the execution time of the optimized parallel KMP algorithm is about 3.6 times faster than that of the non-optimized parallel KMP algorithm.

Key Words : KMP algorithm, GPGPU, OpenCL

1. 서 론

패턴 매칭 알고리즘은 일반 텍스트에서 일치하는 단어를 검색하는 응용에서부터 네트워크를 통한 침입을 감지하는 시스템이나 바이오 인포매틱스 분야의 유전자 분석에서 일치하는 유전자를 검색하는 응용에 이르기까지 그 활용이 다양하다[1, 2]. 현재 사용되고 있는 다양한 종류의 패턴 매칭 알고리즘 중에서도 Knuth-Morris-Pratt이 개발한 KMP 알고리즘은 패턴과 텍스트의 길이가 증가함에 따라 시간 증가가 큰 기존의 brute-force 알고리즘과는 달리 큰 텍스트와 패턴에 대해서도 상대적으로 실행시간이 빠르다는 점에서 많은 분야에서 사용되고 있다. 하지만 기존 패턴 매칭 알고리즘보다 빠른 KMP 알고리즘의 경우에도 유전자 매칭의 경우와 같이 검색할 대상의 크기가 현저히 큰 경우에는 실행시간이 많이 증가하는 단점이 있다[3].

KMP 패턴 매칭 알고리즘의 속도 저하 문제를 해결하기 위하여 GPGPU(General Purpose computing on Graphic Processing Unit)를 위한 CUDA(Compute Unified Device Architecture)나 FPGA(Field Programmable Gate Array)를 적용한 연구가 현재까지 많이 진행되고 있다[4, 5, 6].

본 논문에서는 KMP 알고리즘의 성능을 GPGPU를 이용한 병렬처리와 최적화 기법 적용을 통해 개선시키고자 한다. 본 논문에서는 GPGPU 플랫폼 중 OpenCL (Open Computing Language)을 기반으로 연구를 진행하였다. OpenCL기반 GPGPU 병렬 프로그램의 최적화를 위해 실험에 사용된 GPU의 특성에 맞는 워크-그룹 및 워크-아이템 할당 최적화, 패턴 매칭에 사용되는 패턴과 실패 테이블의 로컬 메모리 복사 그리고 Host와 GPU 사이의 데이터 전송을 패턴 매칭 알고리즘과 오버래핑하는 기법을 적용하여 성능을 향상시켰다. 실험 결과 최적화를 적용하고 난 후 실행시간이 최대 3.6배 향상되었다.

본 논문의 구성 순서는 다음과 같다. 2장에서는 KMP 알고리즘에 대하여 설명을 하고, 3장에서는 GPGPU 병렬 프로그램의 병렬화 기법 및 성능 최적화에 대해 다룬다. 4장은 실험 결과 및 분석 내용을 설명하고 마지막으로 5장에서는 결론과 향후 연구를 소개한다.

2. KMP 알고리즘

KMP 알고리즘은 Knuth-Morris-Pratt이 개발한 패턴 매칭 알고리즘으로 그림 1과 같이 패턴 매칭 과정에 단어의 접두사와 접미사의 개념을 이용하였다. 먼저 검색할 패턴을 전 처리하여 패턴 매칭 과정에서 사용할 실패 테이블(Failure Table)을 생성한다. 각 패턴의 맨 앞부터 시작하여 패턴의 끝나는 지점까지 이동하면서 해당 위치에서 접두사와 접미사가 일치하는 개수를 찾아 실패 테이블에 저장한다. 생성된 실패 테이블의 길이는 패턴의 길이와 동일하다. 그 다음, 패턴과 실패 테이블 그리고 검색할 텍스트를 입력 값으로 하여 패턴 매칭을 진행한다. 텍스트의 처음부터 패턴이 일치하는지의 여부를 검색하는데 텍스트가 일치하는 경우에는 일반적인 패턴 매칭 방법으로 패턴의 끝까지 일치

† Corresponding Author : Dept. of Computer Science and Engineering, Konkuk University, Korea.
E-mail: neungsoo@konkuk.ac.kr
* Dept. of Computer Science and Engineering, Konkuk University, Korea.
** Dept. of Computer Engineering, Myongji University, Korea.
Received : March 13, 2018; Accepted : April 1, 2018

하는지를 확인하지만, 만약 일치하지 않는다면 바로 다음 텍스트의 위치로 이동하여 패턴을 처음부터 검사하는 것이 아니라 미리 생성한 실패 테이블을 참조하여 위치를 건너 뛰어 패턴을 검사한다.

예를 들어, 그림 1(a)와 같이 다섯 번째 인덱스에서 일치하지 않는 것을 찾았다면 해당 위치의 바로 전 인덱스에 해당하는 즉, 아래 예시에서의 네 번째 인덱스 위치에 실패 테이블 값을 사용하여 건너 뛸 값을 얻는다. 네 번째 인덱스 위치의 실패 테이블 값이 2이기 때문에 그림 1(b)처럼 패턴의 0, 1번째 인덱스는 따로 검사하지 않고 뛰어넘은 다음 두 번째 인덱스부터 탐색을 다시 시작한다. 이 과정을 통해 일치하지 않는 것을 찾은 경우 바로 다음 인덱스로 이동하여 전체 텍스트를 검사해야하는 방식과는 다르게 전체 텍스트를 하나씩 검사하지 않고도 패턴을 찾아낼 수 있다. 텍스트의 길이가 T, 패턴의 길이를 P라고 한다면 KMP 패턴 매칭 알고리즘의 전체 시간 복잡도는 $O(T+P)$이 된다. 따라서 텍스트나 패턴의 길이가 증가함에 따라 시간 복잡도가 $O(TP)$으로 증가하는 기존 brute-force 방식 패턴 매칭 알고리즘과 달리 KMP 알고리즘의 처리 속도는 상대적으로 빠르다[7].

KMP 알고리즘을 사용하여 패턴 매칭을 하는데 걸리는 시간을 현저하게 줄였다 하더라도 KMP 알고리즘에서는 패턴의 크기보다 텍스트의 크기가 훨씬 큰 것이 일반적이기 때문에 텍스트의 크기는 성능에 큰 영향을 준다. 따라서 처리해야하는 텍스트의

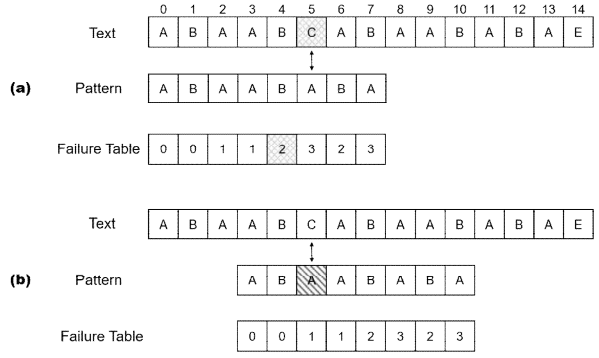

그림 1 KMP 알고리즘 예시
Fig. 1 An example of KMP algorithm

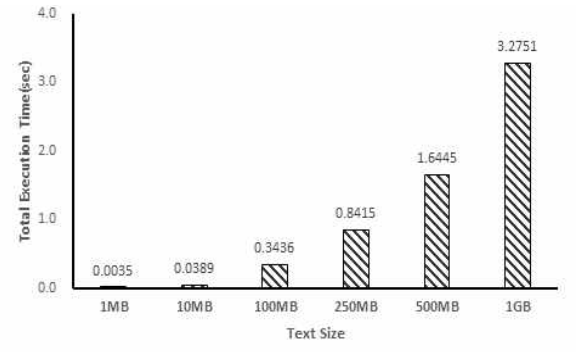

그림 2 각 텍스트 사이즈별 전체 실행시간
Fig. 2 The total execution time of various text sizes

크기가 커질수록 일치하는 패턴을 찾는 시간은 그림 2와 같이 크게 증가하게 된다. 이를 해결하기 위한 방안으로 본 논문에서는 GPGPU를 이용한 병렬화를 통해 KMP 알고리즘을 가속화하고자 한다.

3. GPGPU를 통한 KMP 알고리즘 병렬화

3.1 워크-그룹 및 워크-아이템 할당 최적화

GPU에서 병렬 KMP 알고리즘을 구현하기 위해서는 주어진 텍스트를 분할하여 탐색하는 데이터 병렬처리 기법을 활용한다. 텍스트는 작은 서브-텍스트 단위로 분할되어 각 워크-아이템에 할당이 되고 각 워크-아이템에서 주어진 패턴이 존재하는지를 검색하게 된다. 이 때 텍스트 분할 과정에서 서브-텍스트들 사이에 주어진 패턴이 존재할 경우 중간에 잘리는 경우가 있어 이를 고려하여 처리를 해야 한다[8]. 주어진 텍스트의 마지막 $P-1$ 길이는 새로운 패턴의 탐색을 하지 않고 이전에 시작한 패턴이 일치하여 종료가 되는지 만을 확인하게 된다. 따라서 분할 영역에 $P-1$ 만큼의 길이를 중복하여 추가할당을 하고 이 부분은 이전 패턴의 종료만을 확인하도록 하여 분할 과정에서 발생할 수 있는 패턴의 잘림을 방지할 수가 있다. 이렇게 할당된 워크-아이

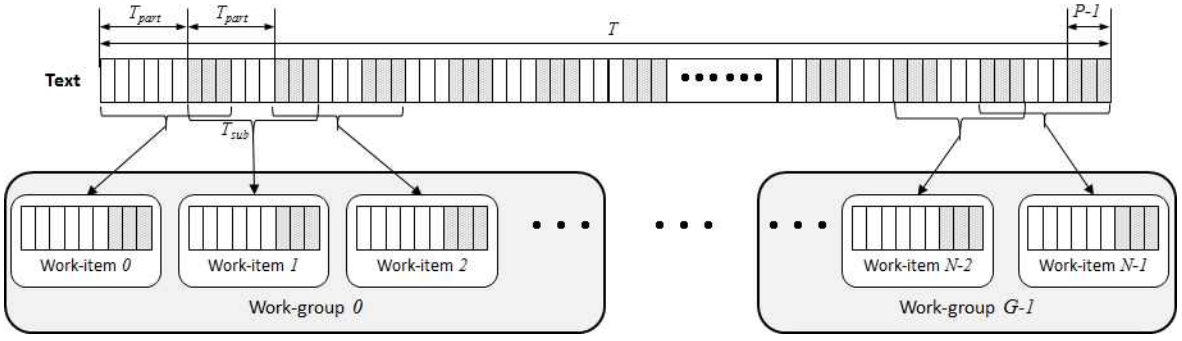

그림 3 워크-아이템 및 워크-그룹 할당
Fig. 3 Work-item and work-group allocation

템은 워크-그룹 단위로 묶여 GPU내의 계산 유닛(Computing Unit)에 할당을 하게 된다. 전체 텍스트에 워크-그룹 및 워크-아이템을 할당하는 방법은 다음 그림 3과 같다.

KMP 알고리즘의 OpenCL 처리 성능을 최적화하기 위하여, 워크-아이템과 워크-그룹을 GPU내에 프로세싱-엘리멘트(Processing Element: PE)와 계산-유닛에 효율적으로 할당하여야 한다. GPU는 동시에 실행이 가능한 여러 개의 계산-유닛으로 구성이 되어 있으며 각 계산-유닛은 다수의 프로세싱-엘리먼트들로 이루어져 있다. 하나의 워크-아이템은 GPU내에 프로세싱-엘리먼트에 할당이 되어 처리가 된다. 이 때, 프로세싱-엘레먼트에 할당된 하나의 워크-아이템은 각각 하나의 스레드와 같다. 이러한 워크-아이템을 워크-그룹으로 묶어 계산-유닛에 할당을 하게 된다. 이 때, 전체 워크-그룹의 개수를 총 계산-유닛의 배수로 할당하지 않으면 유휴 계산-유닛이 생겨 실행 시 최적으로 동작할 수 없다. 하나의 워크-그룹은 다수의 워크-아이템으로 이루어지므로 워크-아이템은 계산-유닛의 배수로 워크-그룹의 개수가 이루어지도록 묶여서 처리되어야 한다. 하나의 워크-그룹에 할당 가능한 워크-아이템의 개수는 계산 유닛 당 PE의 개수보다 작거나 같아야 한다. 따라서 총 워크-아이템과 워크-그룹은 다음과 같이 할당이 되게 된다.

$$W_{group} \propto CU_{gpu} \tag{1}$$
$$W_{item} \propto W_{item/CU} \times CU_{gpu} \; where \; W_{item/CU} \le PE_{CU}$$

여기서, W_{item}는 총 워크-아이템의 개수, W_{group}은 총 워크-그룹의 개수, $W_{item/CU}$는 계산-유닛 당 할당 된 워크-아이템의 개수, PE_{CU}는 각 계산-유닛 당 PE의 개수, CU_{gpu}는 GPU내의 총 계산-유닛의 개수를 의미한다. 따라서 최적의 성능을 내기 위해서는 워크-아이템 개수를 GPU내의 총 PE의 개수($PE_{gpu} = CU_{gpu} \times PE_{CU}$)로 할당하거나 그 배수로 할당함으로써 모든 PE가 같은 작업량을 처리하여 최적으로 동작할 수 있게 만든다[9]. 또한 이때 각 워크-아이템 당 할당되어 검색이 되는 서브-텍스트의 크기는 다음과 같다.

$$T_{part} = \left\lceil \frac{T-P+1}{W_{item}} \right\rceil \tag{2}$$
$$T_{sub} = T_{part} + P - 1$$

여기서, T는 주어진 텍스트의 길이, P는 검색하는 패턴의 길이, 그리고 T_{sub}는 각 워크-아이템에 할당되는 서브-텍스트의 길이를 의미한다. 총 텍스트에서 주어진 패턴이 시작할 수 있는 문자의 개수는 $T-P+1$ 이고 이를 워크-아이템에 동등하게 분배(T_{part})를 하여야 한다. 분배된 텍스트에 마지막 잘린 부분의 검사를 위하여 $P-1$ 길이만큼을 중복 할당하여 검사를 하게 된다. 이때 하나의 워크-아이템이 처리하게 되는 서브 텍스트의 범위 i는 다음과 같다.

$$W_{id} \times T_{part} \le i \le (W_{id}+1) \times T_{part} + (P-1) \tag{3}$$

여기서, W_{id}는 각 워크-아이템의 아이디를 의미하며 그 값은 $0 \le W_{id} \le W_{item} - 1$이다. 따라서 OpenCL 기반으로 처리할 경우의 시간 복잡도는 다음과 같이 개선된다.

$$O(\frac{T-P+1}{PE_{gpu}} + P - 1) \approx O(\frac{T}{PE_{gpu}} + P) \tag{4}$$

3.2 CPU-GPU 메모리 계층 구조를 고려한 성능최적화

OpenCL을 이용하여 성능을 최적화하기 위하여서는 CPU-GPU 간의 메모리 계층구조를 이해하여 이를 최적화 하여한다. 일반적으로 OpenCL과 같은 GPGPU 컴퓨팅은 GPU 내의 글로벌 메모리를 통해 CPU와 데이터를 주고받으며 데이터를 처리한다. 따라서 그림 4와 같이 GPU 내의 메모리 계층구조에 따른 최적화와 CPU-GPU 메모리 간의 이루어지는 데이터 통신의 최적화를 고려하여 전체 성능을 향상시켜야 한다.

GPU 메모리 계층 구조에 따르면 GPU 내에 글로벌 메모리에 접근하는 것은 로컬 메모리와 프라이빗 메모리의 접근 속도보다 약 10배가 느리다[9]. KMP 알고리즘은 패턴과 테스트 간에 반복적인 매칭을 실행하고 실패 때 마다 실패 테이블을 접근해야 한다. 이 과정에서 글로벌 메모리에 계속 접근하게 되고 이는 성능 저하의 원인이 된다. 그러므로 성능을 최적화하기 위해서 워크-아이템들이 글로벌 메모리에 접근하는 횟수를 줄여야 한다. 이를 위해 본 논문에서는 KMP 알고리즘 연산 과정에서 반복적으로 접근이 되는 패턴 그리고 실패 테이블을 로컬 메모리에 복사하여 처리한다. 각 워크-그룹 내 워크-아이템들은 워크-그룹별로 사용가능한 로컬 메모리를 통하여 데이터를 공유한다. 패턴 매칭이 진행되는 동안 로컬 메모리에 있는 패턴과 실패 테이블에 접근하여 글로벌 메모리에 있는 텍스트와 일치하는지의 여부를 검사한다. 그리고 일치하면 글로벌 메모리에 접근하여 전체 일치

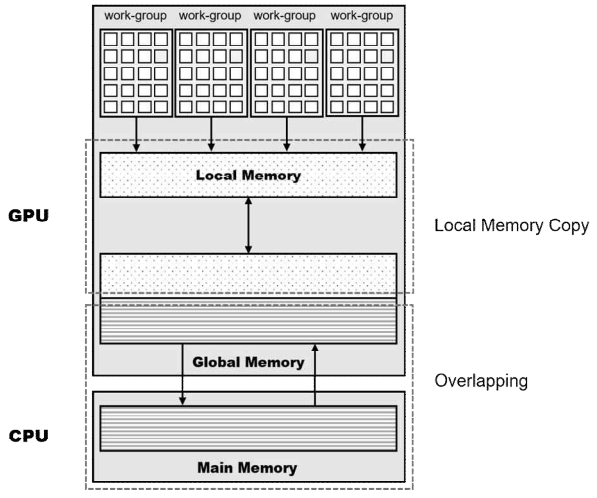

그림 4 CPU-GPU 메모리 계층구조에서의 성능 최적화

Fig. 4 Performance optimization in CPU-GPU memory hierarchy

횟수를 카운팅하는 변수에 값을 써준다. 이를 통해 전체 글로벌 메모리 접근 횟수를 줄이고 성능을 향상시켰다.

KMP 알고리즘에서는 패턴에 비하여 텍스트의 크기는 크다고 할 수 있다. GPU 기반의 병렬 KMP 알고리즘의 전체 실행시간은 매칭을 하는 텍스트가 GPU에 전달되고 연산을 진행한 다음 결과 값을 넘겨받는 시간을 포함한다. 이에 따른 전체 성능은 다음과 같다.

$$Ex.\ Time = \frac{T}{\tau_d} + EX_{gpu} + \frac{R}{\tau_d} \tag{5}$$

여기서, T는 전체 텍스트의 크기, R은 반환되는 결과의 크기, τ_d는 CPU-GPU 간의 전송 속도(PCIE 속도), 그리고 EX_{gpu}는 GPU 실행시간을 의미한다. 따라서 모든 텍스트를 GPU에 전달하여 패턴 매칭을 실행하고 결과를 반환 받을 경우 텍스트의 크기가 커지게 되면 CPU-GPU 사이의 통신 시간이 커지게 되어 전체 성능에 영향을 주게 된다. 따라서 GPU 계산에 의하여 단축되는 성능 향상을 극대화하기 위해서는 CPU-GPU 간에 발생하는 전송 오버헤드를 최소화하여야 한다. 이러한 CPU-GPU간의 데이터 전송 오버헤드는 필연적으로 발생하는 것으로 이를 원천적으로 제거하거나 줄일 수는 없고 이를 숨기는 방안을 제안하고자 한다. 본 연구에서는 이를 위하여 GPU에 전달되는 텍스트를 분할하여 전송을 하고 각각 분할된 텍스트에 대한 패턴 매칭을 시행하는 동안 다음 분할될 텍스트를 전달하는 방법으로 CPU-GPU 사이에 계산과 전송을 중첩시키는 방법을 제안한다. 다음 그림 5는 텍스트를 네 개로 분할하여 처리할 경우 CPU-GPU간에 계산-전송 중첩이 일어나는 과정을 보인다. 한 예로 첫 번째 데이터를 전송하고 패턴 매칭 실행을 하는 동안 두 번째 데이터를 전송한다. 또 두 번째 데이터를 가지고 연산이 진행되는 동안 세 번째 데이터를 전송하고 동시에 첫 번째 데이터를 가지고 연산이 진행된 결과 값을 반환받는 과정을 진행한다. CPU-GPU 전송 중첩을 통한 전체 성능은 다음과 같이 표현할 수 있다.

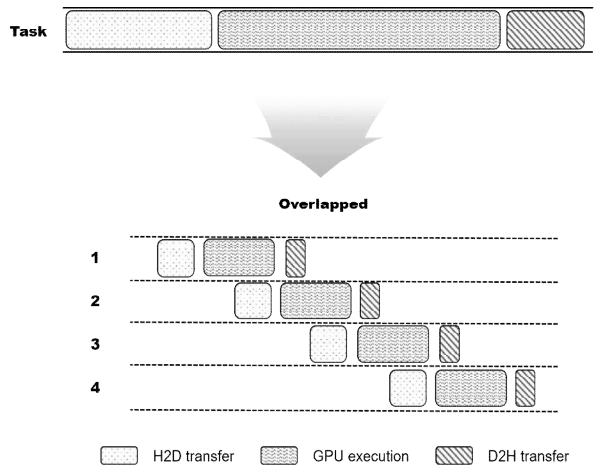

그림 5 네 번의 오버래핑을 통한 처리 과정
Fig. 5 Overlapping process of four computing streams

$$Ex.\ Time_{overlap} = \frac{T/N_s}{\tau_d} + EX_{sgpu} \times N_s + \frac{R/N_s}{\tau_d} \tag{6}$$

여기서, N_s는 텍스트가 분할된 스트리밍의 수이고 EX_{sgpu}는 각 텍스트 스트리밍이 GPU에서 매칭 연산이 실행되는 시간을 의미한다. 이와 같이 실행할 경우 전체 패턴 매칭 연산 시간은 텍스트의 크기에 따라 중가하게 되지만 데이터를 전송하고 반환받는 시간은 패턴 매칭 연산시간과 중첩이 되어 숨기므로 전체 실행시간을 줄일 수 있다. 제시한 호스트 CPU에서 GPU의 글로벌 메모리에 데이터를 분할 전송하여 처리하는 방법은 텍스트의 크기가 커서 GPU 글로벌 메모리의 크기를 초과하는 경우에도 바로 적용을 할 수가 있다. 이 경우 전체 텍스트에 대한 패턴 매칭을 하기 위해서는 글로벌 메모리를 넘지 않는 크기나 성능 최적을 위한 텍스트 크기로 분할하여 제시한 중첩처리 방법으로 처리할 수가 있다.

4. 실험 및 결과

GPGPU기반 KMP 알고리즘의 병렬화는 GPGPU의 프레임워크 중 하나인 OpenCL을 통해 구현하였다. 본 실험을 진행한 환경은 다음 표 1과 같다.

KMP 알고리즘의 성능을 측정하기 위해서 각각 크기가 다른 텍스트와 길이가 8인 패턴 한 개를 사용하였다. 실행 결과는 각각의 텍스트를 입력 값으로 받아 텍스트를 패턴을 이용하여 검색한 뒤 주어진 패턴이 일치하는 개수를 반환한다. 성능의 측정을 위해 GPGPU기반 병렬 프로그램에 각 최적화 기법을 적용하여 실행시간을 측정하고 평균 내어 비교하였다. 실행시간은 데이터를 전송하고 연산을 진행하고 그 결과 값을 반환하는 과정을 포함한다. 본 실험은 각각 워크-그룹 및 워크-아이템 최적화, 로컬 메모리 복사, 오버래핑을 적용하여 진행하였다.

실험은 그림 6와 같이 각각 1MB, 10MB, 100MB, 250MB의 텍스트에 대해 최적화를 적용하기 전, GPGPU기반 병렬 프로그램에 워크-그룹, 워크-아이템 최적화를 적용한 후, 패턴과 실패 테이블을 로컬 메모리 복사하여 실행한 것과 오버래핑을 두 번 적용한 실행 시간을 측정하여 비교하였다. 최적화를 적용하기 전에는 나눠진 텍스트의 길이가 패턴의 길이보다 작아지지 않는 길이로 나눠지는 최대 워크-아이템과 워크-그룹 개수로 할당하였

표 1 실험 환경
Table 1 The experimental environment

CPU	Intel Core i7-4790	
GPU	Geforce GTX 750	
	OpenCL Version	OpenCL 1.2
	Global Memory Size	1GB
	Shared Memory Size	49152 Bytes
	Number of PEs	512
	Number of CUs	4

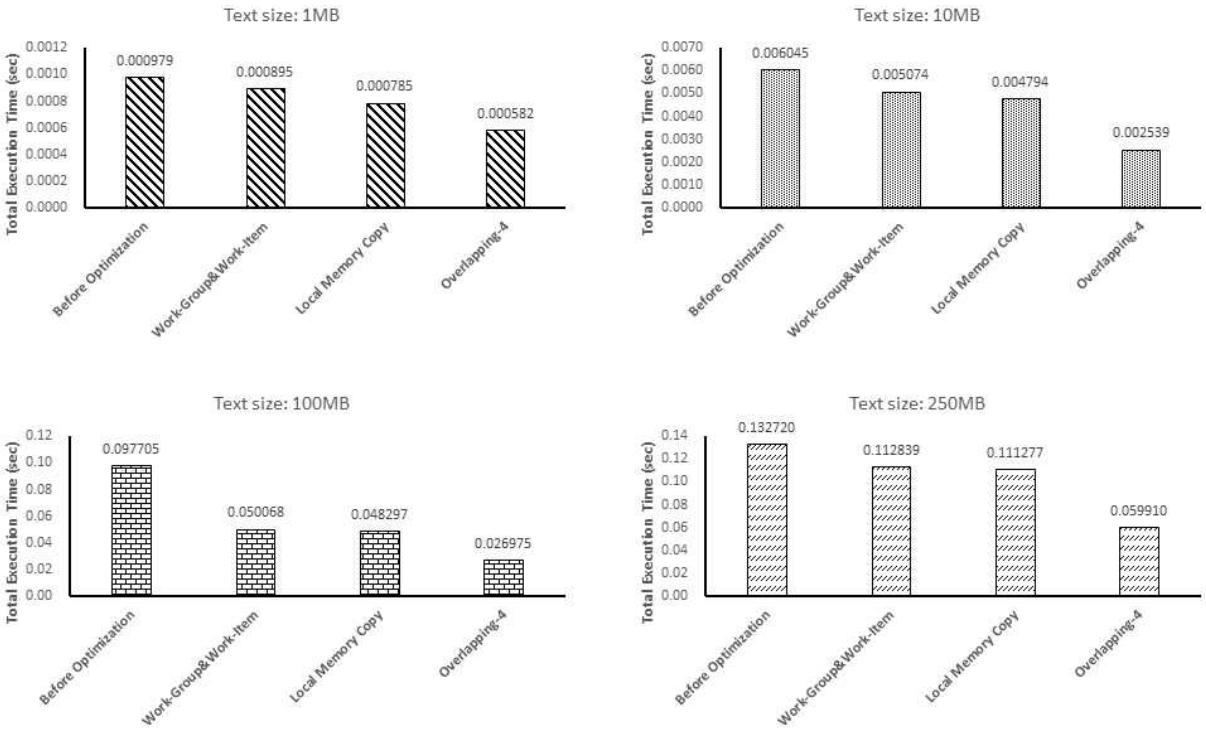

그림 6 각 텍스트 사이즈별 실행시간 비교

Fig. 6 The comparison of the total execution time for various text sizes

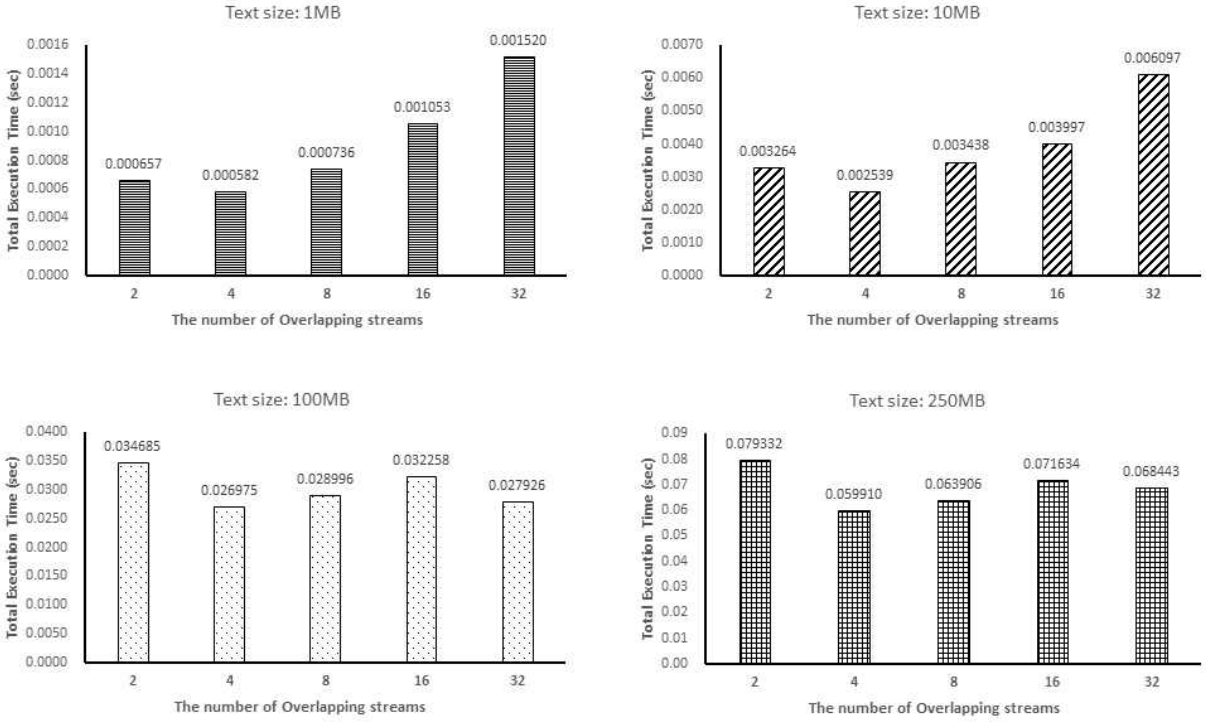

그림 7 다양한 텍스트 크기별 오버래핑 횟수 변화에 따른 총 실행시간

Fig. 7 Total execution time of various overlapping number according to the various text sizes

다. 이 때, 최대로 할당된 워크-아이템의 개수는 1천만 개, 워크-그룹의 개수는 1천 개다. 그 다음에는 워크-아이템 및 워크-그룹 최적화를 적용한 결과와 실패 테이블과 패턴을 로컬 메모리 복사한 결과 그리고 오버래핑을 네 번 적용한 결과를 비교하였다. 실험 결과를 분석해 보면 최적화를 적용하기 전보다 워크-그룹 및 워크-아이템 최적화를 진행한 실행 시간이 최대 1.95배 향상되었다. 워크-아이템과 워크-그룹의 개수를 최대로 할당하더라도 프로세싱-엘레멘트와 계산-유닛의 배수가 되지 않으면 한 번에 동작하지 못하고 유휴자원이 생기므로 성능이 오히려 하락하기 때문이다.

워크-그룹 및 워크-아이템 최적화에 로컬 메모리 복사를 적용한 결과는 크게 차이가 나지 않았는데, 복사를 한 패턴과 테이블은 고정된 작은 길이기 때문에 크게 향상되지 않은 것으로 보인다. 마지막으로 오버래핑을 네 번 적용한 결과는 최적화를 적용하기 전보다 성능이 최대 3.6배 향상되었다.

또한, 그림 7과 같이 텍스트의 크기가 증가됨에 따라 최적의 오버래핑 횟수를 관찰하기 위해 각각 오버래핑 횟수에 따라 텍스트의 크기를 다르게 하여 성능을 측정하였다. 텍스트의 크기는 각각 1MB, 10MB, 100MB, 250MB를 사용하였다. 본 실험에서 사용한 데이터에 오버래핑을 적용한 결과 1MB와 10MB의 텍스트에서는 네 번의 오버래핑을 적용한 실험이 가장 좋은 성능을 보였다. 이는 KMP 패턴 매칭 알고리즘은 데이터의 전송과 반환 시간보다 패턴 매칭을 진행하는 과정에서 소요되는 시간이 압도적으로 크기 때문에 오버래핑을 많이 하면 데이터의 전송과 반환에서 줄어드는 시간보다 오버래핑에 의해서 발생하는 오버헤드가 더 커지는 결과로 보인다. 100MB와 250MB의 텍스트에서도 네 번의 오버래핑을 적용한 결과가 가장 좋은 성능을 보였다. 하지만 1MB와 10MB 텍스트 결과와는 다르게 32번의 오버래핑을 적용한 결과는 다시 실행시간이 줄어드는 것을 볼 수 있었다. 이는 큰 텍스트를 작게 나눠서 전송하고 처리하게 되면 그만큼 한 번 실행하는 시간은 줄어들면서 실행시간 사이에 데이터의 전송과 반환이 일어나서 오버래핑에 의해 발생하는 오버헤드보다 성능의 개선이 더 큰 결과로 볼 수 있다.

5. 결론 및 향후연구

KMP 알고리즘은 전체 텍스트를 전부 검색해야 하는 기존 brute-force 알고리즘에 비해 속도가 빠르다는 장점이 존재하여 많은 분야에서 사용되고 있다. 하지만 KMP 알고리즘의 경우에도 텍스트의 크기가 크게 증가하면 실행시간이 증가하게 된다. 본 연구는 KMP 패턴 매칭 알고리즘을 GPGPU기반 병렬 프로그램으로 구현하여 성능을 개선시켰다. KMP 패턴 매칭 알고리즘은 전체 텍스트를 분할하여 나눠진 각 텍스트를 검색한 다음 최종적으로 전체의 텍스트에 대해 일치하는 패턴의 개수를 반환한다. 분할된 텍스트는 각각의 독립된 작은 텍스트로 하나의 스레드는 작은 단위의 텍스트를 병렬로 처리하게 된다. 본 논문에서는 실험에 사용한 GPU의 특성에 맞게 KMP 알고리즘에 워크-그룹 및 워크-아이템 최적화, 로컬 메모리 복사, 오버래핑을 각각 적용하여 속도를 향상시켰다. 워크-그룹 및 워크-아이템의 개수는 GPU 내의 계산-유닛과 프로세싱-엘레먼트의 배수로 할당하여 유휴자원이 생기지 않고 동작하도록 최적화 하였다. 또한, 패턴 매칭에 사용되는 실패 테이블과 패턴을 로컬 메모리에 복사하여 글로벌 메모리의 접근 횟수를 줄여 성능을 개선시켰다. 마지막으로 텍스트의 크기가 커짐에 따라 한 번에 모든 텍스트를 GPU에 전송하여 처리하려고 할 때 CPU-GPU간 통신 시간이 증가하는 문제점을 해결하기 위하여 오버래핑을 적용하였다. 텍스트를 오버래핑하는 횟수만큼 분할한 다음 각각 분할된 텍스트에 대한 패턴 매칭을 시행하는 동안 다음 분할될 텍스트를 전달하는 방법으로 CPU-GPU 사이에 계산과 전송을 중첩시켜 성능을 개선시켰다. 실험 결과 GPGPU기반 병렬 프로그램에 최적화를 적용하기 전보다 각각의 최적화를 적용한 것이 최대 3.6배 향상되었음을 확인하였다.

향후 글로벌 메모리의 접근 횟수를 줄이기 위해 텍스트를 읽어오는 단위를 늘리고 레지스터 사용을 증가시키기 위한 최적화를 적용하여 성능을 추가적으로 향상시킬 계획이다.

감사의 글

이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No.2016R1D1A1B03935576)과 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No.2017R1D1A1B03033128)임.

References

[1] S. Rajesh, S. Prathima and D. Reddy, "Unusual Pattern Detection in DNA Database Using KMP Algorithm", International Journal of Computer Applications, vol. 1, no. 22, pp. 1-7, 2010.

[2] B. Raju and B. Srinivas, "Network Intrusion Detection System Using KMP Pattern Matching Algorithm", International Journal of Computer Science and Telecommunications, vol. 3, no. 1, pp. 33-36, 2012.

[3] J. Kim, E. Kim and K. Park, "Fast Matching Method for DNA Sequences", Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, vol. 4614, pp. 271-281, 2007.

[4] C. S. Kouzinopoulos and K. G. Margaritis, "String Matching on a Multicore GPU Using CUDA," in Proc. of the 13th Panhellenic Conference on Informatics (PCI'09), 2009, pp. 14-18.

[5] R. P. S. Sidhu, A. Mei, V. K. Prasanna, "String Matching on Multicontext FPGAs using Self-Reconfiguration", in Proc. 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, 1999, pp. 217-226.

[6] X. Bellekens, I. Andonovic, R. Atkinson, C. Renfrew, T.

Kirkham, "Investigation of GPU-based Pattern Matching", in The 14th Annual Post Graduate Symposium on the Convergence of Telecommunications Networking and Broadcasting (PGNet2013), 2013.
[7] D. Knuth, J. Morris, Jr. and V. Pratt, "Fast Pattern Matching in Strings", SIAM Journal on Computing, vol. 6, no. 2, pp. 323-350, 1977.
[8] A. Rasool and N. Khare, "Parallelization of KMP String Matching Algorithm on Different SIMD Architectures: Multi-Core and GPGPU's", International Journal of Computer Applications, vol. 49, no. 11, pp. 26-28, 2012.
[9] AMD Corporation. "OpenCL Programming Guide", AMD Documentation, AMD Corporation, 2013.

저 자 소 개

박 소 은 (So-eun Park)
2016년 건국대학교 컴퓨터공학과(학사)
2016년~현재 건국대학교 컴퓨터공학과(석사과정).
관심분야: OpenCL, GPGPU 등

김 대 희 (Dae-hee Kim)
2015년 건국대학교 컴퓨터공학과(학사)
2017년 건국대학교 컴퓨터공학과(석사)
2017년~현재 건국대학교 컴퓨터공학과 (박사과정).
관심분야: GPGPU, OpenCL, HPC 등

이 명 호 (Myung-ho Lee)
1986년 서울대학교 계산통계학과(학사)
1988년 미국 University of Southern California 전산과학과학과(석사)
1999년 미국 University of Southern California 컴퓨터공학과(박사)
1989년~1999년 Research Assistant, 미국 University of Southern California, Dept. of EE-Systems
1999년~2005년 Staff Engineer, 미국 Sun Microsystems, Inc.
2004년~현재 명지대학교 컴퓨터공학과 교수
관심분야: 컴퓨터구조, 고성능 컴퓨팅, 최적화 컴파일러 등

박 능 수 (Neung-soo Park)
1991년 연세대학교 전기공학과(학사)
1993년 연세대학교 대학원 전기공학과(석사)
2002년 미국 University of Southern California 전기공학과(공학박사)
2002년~2003년 삼성전자 책임연구원
2003년~현재 건국대학교 컴퓨터공학과 교수
관심분야: 컴퓨터구조, 임베디드 시스템, 병렬 시스템, GPGPU 컴퓨팅, HPC, 빅-데이터 처리, 멀티미디어 컴퓨팅 등