효율적인 데이터 중복제거를 위한 GPGPU 병렬 라빈 핑거프린팅

마정현, 박세진, 박찬익

포항공과대학교 컴퓨터공학과 (경북 포항시 남구 효자동 산 31번지)
{doitnow0415, baksejin, cipark}@postech.ac.kr

Parallel Rabin Fingerprinting on GPGPU for Efficient Data Deduplication

Jeonghyeon Ma, Sejin Park, Chanik Park
Department of Computer Science and Engineering, POSTECH, Pohang, South Korea

요 약

데이터 중복 제거를 수행하기 위한 여러 단계 중 청킹에 사용되는 라빈 핑거프린트 값을 구하는 단계가 가장 큰 오버헤드를 차지한다. 따라서, 본 논문에서는 효율적인 데이터 중복 제거를 위한 병렬 라빈 핑거프린트 방법을 제안한다. 또한 효율적인 라빈 핑거프린팅의 병렬화를 위해 세 가지 이슈를 고려한다. 첫 번째로 병렬처리를 위해 입력 데이터 스트림을 일정한 크기의 데이터 섹션으로 분할 할 때, 데이터 섹션의 경계선에 있는 데이터들에 대해서도 라빈 핑거프린팅을 수행하기 위한 고려, 두 번째로 라빈 핑거프린팅 연산 특징을 효율적으로 이용하기 위한 고려, 마지막으로 순차 방식으로 청크 경계선을 구했을 때와 비교하여 병렬 방식으로 청크 경계선을 구했을 때, 변경 될 수 있는 청크 경계선에 대한 고려를 한다.

GPGPU를 이용한 병렬 라빈 핑거프린트 방식은 CPU를 이용한 순차 라빈 핑거프린트 방식에 비해 약 16배 성능향상을 보였고, CPU를 이용한 병렬 라빈 핑거프린트 방식에 비해서도 약 5.3배 성능향상을 보였다. 이러한 라빈 핑거프린팅 연산 처리량의 증가는 데이터 중복 제거 기법의 전체적인 성능향상을 가져올 수 있다.

1. 서 론

데이터가 급격하게 늘어남에 따라 스토리지 시스템에서는 많은 데이터를 효율적으로 처리하기 위해 데이터 중복 제거 기법을 사용한다. 데이터 중복제거 기법은 데이터 집합의 중복성을 검출하여 제거함으로써 공간의 활용성을 높이고, 비용을 절약하기 위한 목적으로 사용되고 있다. 이러한 데이터의 중복성을 검사하는 기본 단위를 청크라고 부르고, 데이터의 집합으로부터 청크들을 찾는 과정을 청킹이라고 부른다. 청킹은 크게 청크를 구하는 방법에 따라 두 가지로 나뉘는데, 첫 번째 방법은 고정된 크기로 데이터를 나누어 청크를 구하는 방식이고 [1] (SC, Static Chunking), 두 번째 방법은 데이터의 내용에 따라 가변적인 크기로 데이터를 나누어 청크를 구하는 방식이다 [2] (CDC, Content-Defined Chunking). 일반적으로 CDC가 SC보다 더 많은 공간을 절약할 수 있다고 알려져 있지만, CDC 방식에서는 청크를 구하기 위해 모든 데이터의 내용을 스캔해야하기 때문에 많은 연산을 필요로 하며 시간을 소비한다. CDC에서는 데이터의 내용을 스캔하는 방법으로 주로 라빈 핑거프린트 알고리즘을 [3] 사용하며, 이는 데이터 중복제거 기법의 과정 중 가장 큰 연산 시간을 차지하는 작업이다 [4]. 따라서, CDC 기반 데이터 중복제거 기법에서는 라빈 핑거프린트를 구하는 과정(라빈 핑거프린팅)이 주된 병목구간으로 여겨진다.

이 논문에서는 이 병목구간을 제거하기 위해 CDC에서 라빈 핑거프린팅을 더 빠르게 수행할 수 있는 GPGPU를 이용한 병렬 라빈 핑거프린팅을 제안한다. 또한, 효율적인 라빈 핑거프린팅의 병렬화를 위해 여러 가지 사안들을 고려한다.

2. 기초지식 및 관련연구

라빈 핑거프린트는 데이터 중복 제거 기법 중 CDC 방식에서 청크를 구하기 위해 사용되는 값으로 다항식을 이용하여 계산되는 해쉬 값이다. 다항식은 n개의 유한 필드로 한정되어 있고, n개 필드들의 합을 통해 라빈 핑거프린트를 구하게 된다. 수식 1은 라빈 핑거프린트를 구하는 다항식을 나타내는데, m_0부터 m_{n-1}까지는 n개의 연속적인 데이터를 의미한다. 이는 핑거프린트를 구하는 크기가 n인 슬라이딩 윈도우의 단위가 된다. $f(x)$는 x에 대한 n-1차 다항식을 통해 계산된 라빈 핑거프린트를 의미한다.

$$f(x) = m_0 + m_1 x + \ldots + m_{n-1} x^{n-1} \quad \text{(수식 1)}$$

일련의 순차적인 데이터 집합에 대해 라빈 핑거프린트를 연속적으로 구할 때, 다항식을 이용한 라빈 핑거프린트 계산 알고리즘의 특성을 이용하여 라빈 핑거프린트들을 쉽게 구할 수 있다. 예를 들어, m_0부터 m_{n-1}까지의 데이터에 대한 라빈 핑거프린트는 m_1부터 m_n까지의 데이터에 대한 라빈 핑거프린트를 구하는데 이용 될 수 있고, m_1부터 m_n까지의 데이터에 대한 라빈 핑거프린트는 m_2부터 m_{n+1}까지의 데이터에 대한 라빈 핑거프린트를 구하는데 이용될 수 있다. 즉, 이러한 라빈 핑거프린트의 특성을 이용하면, 수식 1에 표현된 모든 다항식 연산을

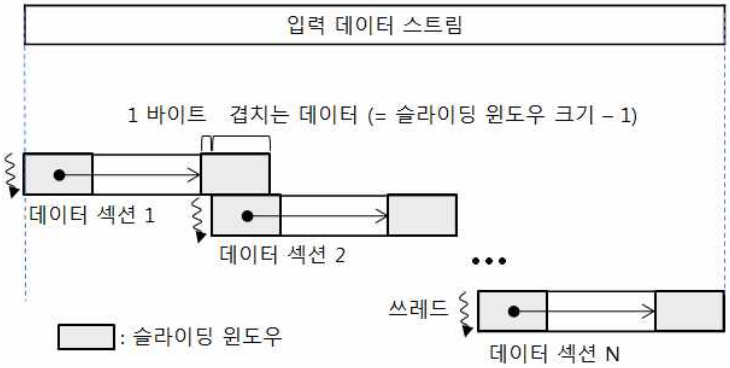
수행하여 라빈 핑거프린트를 구할 필요 없이 이전 라빈 핑거프린트에 간단한 추가 연산을 통해 다음 라빈 핑거프린트를 구할 수 있다.

3. 라빈 핑거프린팅 병렬화

효율적인 라빈 핑거프린팅의 병렬화를 위해 세 가지를 고려한다. 1) 입력 데이터 스트림을 일정한 크기의 데이터 섹션으로 분할할 때 두 데이터 섹션의 경계에 위치한 데이터들에 대해서도 라빈 핑거프린트를 구하기 위한 고려, 2) 이전 슬라이딩 윈도우의 핑거프린트 값을 다음 슬라이딩 윈도우의 핑거프린팅에 사용할 수 있는 라빈 핑거프린팅의 특징을 효율적인 이용하기 위한 고려, 3) 순차 방식의 라빈 핑거프린팅을 통해 구한 청크 경계선과 비교했을 때, 변경될 수 있는 청크 경계선을 변경 없이 각 데이터 섹션들을 병합하기 위한 고려

3.1. 입력 데이터 스트림의 분할

데이터 스트림을 데이터 섹션들로 분할 시 입력 데이터 스트림에 대해 데이터 섹션의 경계선에 상관없이 연속적으로 핑거프린트 값을 구하기 위해선 인접한 데이터 섹션들은 (그림 1)에 나타난 것과 같이 겹쳐지는 데이터 부분을 갖는 상태로 분할되어야 한다.

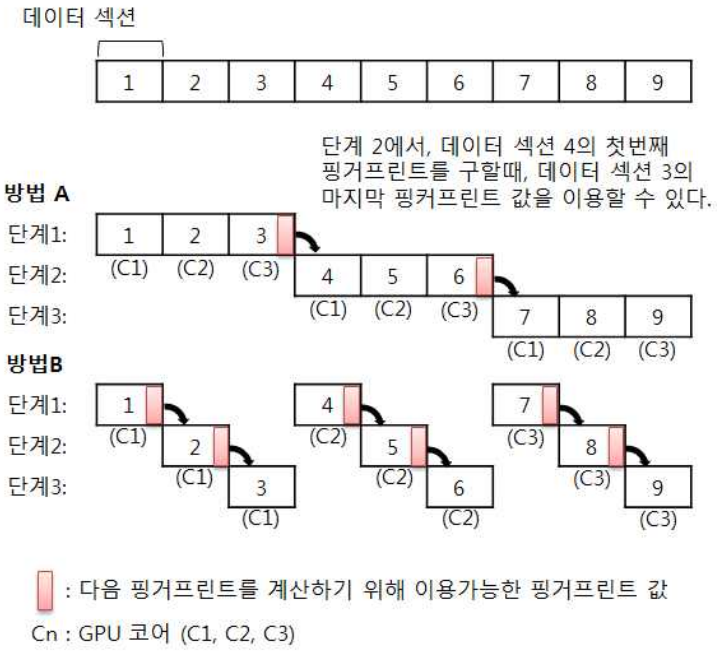


(그림 1) 입력 데이터 스트림의 각 쓰레드에서 수행될 데이터 섹션으로의 분할

추가적인 이슈로, 데이터 스트림을 데이터 섹션들로 분할할 때, 연산량과 병렬화정도의 트레이드오프가 존재한다. 만일 분할된 데이터 섹션의 크기가 작다면, 병렬화정도는 증가하지만 연산량도 증가하게 되는데 이는 각 데이터 섹션의 첫 번째 슬라이딩 윈도우의 라빈 핑거프린팅이 전체 다항식을 계산해야하는 많은 연산을 요구하는 작업이기 때문이다. 이와 반대로, 분할된 데이터 섹션의 크기가 크다면, 앞선 슬라이딩 윈도우의 핑거프린트 값을 이용할 수 있는 후속 슬라이딩 윈도우가 상대적으로 많아지기 때문에, 병렬화정도는 감소하지만 연산량도 역시 감소한다. 따라서, 입력 데이터 스트림의 분할은 GPU 코어의 개수와 같은 GPU 특성에 따라 주의 깊게 이루어져야한다.

3.2. 라빈 핑거프린팅 특징의 효율적인 이용

라빈 핑거프린팅은 이전 슬라이딩 윈도우의 핑거프린트가 다음 핑거프린트를 구하는 연산에 사용될 수 있다는 특징을 가지고 있다. 이러한 라빈 핑거프린팅의 특징을 효율적으로 이용하기 위해선 최대한 많은 수의 데이터 섹션의 마지막 슬라이딩 윈도우 핑거프린트 값이 다음 데이터 섹션으로 전달 되어야한다. 이를 위해선 데이터 섹션의 연산 순서를 앞선 데이터 섹션의 마지막 핑거프린트 값을 이용할 수 있도록 조절하는 방법이 필요하다. (그림 2)는 데이터 섹션의 연산 순서에 따라 재사용할 수 있는 핑거프린트 값의 개수가 달라짐을 보여준다. 예시에선 방법 B가 방법 A보다 4개의 핑거프린트 값을 더 재사용할 수 있다. 이러한 핑거프린트의 재사용성은 라빈 핑거프린팅 수행 성능에 영향을 미치는 요소가 된다.



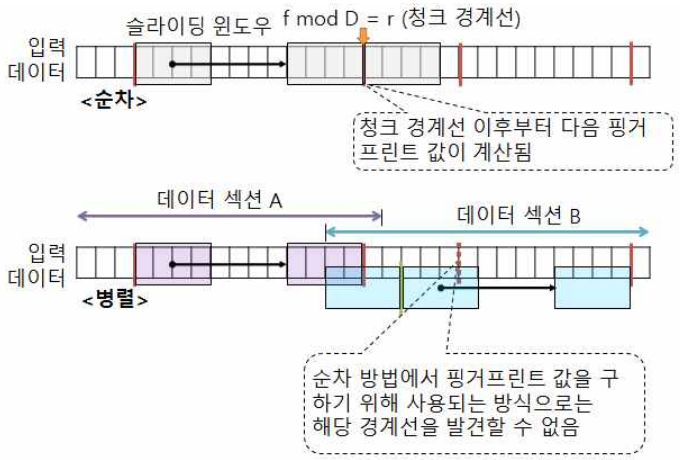
(그림 2) 두 가지 접근방식의 재사용가능한 핑거프린트 개수의 비교

3.3. 각 데이터 섹션 청크 경계선들의 병합

만약 특정 청크 경계선이 발견되면, 다음 핑거프린트 값을 구하는 연산은 발견된 청크 경계선 이후의 슬라이딩 윈도우부터 수행되기 때문에 라빈 핑거프린팅에 병렬화를 적용할 때, (그림 3)에서와 같이 청크 경계선이 순차적인 라빈 핑거프린팅의 결과와 다르게 변경 될 수 있다. 이러한 청크 경계선의 변경 없이 병합하기 위해선, 발견된 청크 경계선에 상관없이, 라빈 핑거프린팅을 청크 경계선 이후로 건너뛰지 않고 연속적으로 모든 슬라이딩 윈도우의 핑거프린트 값을 구해야 한다.

이러한 모든 데이터에 대해 핑거프린트 값을 구하는 작업은 기존 순차 방법에서 쓰이던 방식에 비해 더 많은 연산량으로 인해 더 많은 시간이 소요될 것으로 보이나

사실은 그렇지 않다. 그 이유는 GPGPU 수행의 특징인 와프 분기(Warp divergence) [5] 때문이다. 와프(Warp)란 동일한 코드를 수행하는 쓰레드들의 그룹으로써 보통 32개로 이루어지고, 와프 분기란 와프 내의 쓰레드들이 각 코드 내의 분기문의 결과에 상관없이 모든 분기문에 해당하는 내용을 동일하게 수행하는 것을 말한다. 따라서, 와프내의 특정 쓰레드에서 청크 경계선을 발견함으로 인해 청크 경계선 이후 슬라이딩 윈도우의 핑거프린팅을 통해 얻는 이득은 없을뿐더러, 모든 쓰레드가 추가된 분기문을 모두 수행해야하기 때문에 오히려 핑거프린팅의 효율성을 떨어뜨린다.



(그림 3) 병렬화를 라빈 핑거프린트에 적용함으로 인해 변경될 수 있는 청크 경계선

4. 실 험

본 논문에서 사용한 실험환경은 CPU: Intel Xeon E5-2630, 2.3GHz (12 코어), RAM: 48GB, GPU: Nvidia Quadro 2000을 사용하였다.

표 1은 CPU를 이용한 순차, 병렬 라빈 핑거프린팅과 GPGPU를 이용한 병렬 라빈 핑거프린팅의 처리량 비교 결과이다. CPU 병렬 라빈 핑거프린팅의 경우, 실험에 사용한 CPU 코어 개수와 동일한 12개의 쓰레드를 생성하여 수행한 결과이다. 결과를 통해 알 수 있듯이, GPGPU를 이용한 병렬 라빈 핑거프린팅이 순차 방식에 비해 약 16배의 성능향상을 보일 뿐만 아니라, CPU를 이용한 병렬 라빈 핑거프린팅에 비해서도 약 5.3배의 성능향상을 보인다.

(표 1) CPU를 이용한 순차, 병렬 라빈 핑거프린팅과 GPGPU를 이용한 병렬 라빈 핑거프린팅의 처리량 비교

	CPU (순차)	CPU (병렬)	GPGPU (병렬)
처리량	186MB/s	560MB/s	3004MB/s

5. 결 론

본 논문에서는 GPGPU를 이용한 병렬 라빈 핑거프린팅 방식을 제안하였고, 효율적인 라빈 핑거프린트 연산의 병렬화를 위해 세 가지 이슈를 고려하였다. 실험결과 병렬 라빈 핑거프린팅 방식이 순차 방식에 비해 처리량이 크게 향상 되었고, 이는 데이터 중복 제거 기법 중 가장 큰 병목 부분인 데이터 집합들의 라빈 핑거프린팅 부분을 개선할 수 있음을 의미한다.

본 논문에서는 데이터 중복 제거 기법 중 라빈 핑거프린팅에 대한 병렬화를 다뤘지만, 데이터 중복 제거 기법의 추가적인 성능 향상을 위해서는 데이터 중복 제거 기법의 다른 과정에 대해서도 병렬화 및 동시 수행이 이루어져야한다. 전체 데이터 중복제거 기법에 대해 병렬화 및 동시 수행에 관한 논문으로 P-dedupe [6]이 있지만, 데이터 중복 제거 과정 중 가장 큰 오버헤드를 차지하는 청킹을 효율적으로 병렬화하기 위한 구체적인 방법을 제시하지 않았다.

Acknowledgement

본 연구는 2013년도 미래창조과학부 산업원천기술개발사업의 지원을 받아 수행 중인 디바이스 소셜리티를 이용한 무설정 방식 이종사간 상호연동 기술 개발(10041801) 과제의 일환으로 수행하였음.

참고문헌

[1] Quinlan, Sean, and Sean Dorward. “Venti: A New Approach to Archival Storage.“ FAST. Vol. 2. 2002.

[2] Muthitacharoen, Athicha, Benjie Chen, and David Mazieres. “A low-bandwidth network file system.“ ACM SIGOPS Operating Systems Review. Vol. 35. No. 5. ACM, 2001.

[3] R. M. Karp and M. O. Rabin. “Efficient randomized pattern-matching algorithms.” IBM Journal of Research and Development, 31(2):249-260, 1987

[4] Won, Youjip, et al. “Efficient index lookup for De-duplication backup system.“ In proc of Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. (MASCOTS ’ 08)

[5] http://www.nvidia.com/docs/IO/116711/sc11-perf-optimization.pdf

[6] Xia, Wen, et al. “P-Dedupe: Exploiting Parallelism in Data Deduplication System.“ Networking, Architecture and Storage (NAS), 2012 IEEE 7th International Conference on. IEEE, 2012.