http://dx.doi.org/10.3745/KIPSTC.2012.19C.4.247

# 네트워크 코딩의 병렬처리 성능비교

최 성 민[†] · 박 준 상[††] · 안 상 현[†††]


## 요 약

네트워크 코딩(Network Coding)은 통신망의 성능 향상에 도움을 줄 수 있으나 이의 소프트웨어적 구현은 부호화/복호화 단계에서 매우 큰 지연시간을 유발할 수 있어 이를 줄일 수 있는 병렬화된 구현이 필수적이라 할 수 있다. 본 논문에서는 랜덤 리니어 네트워크 코딩(Random Linear Network Coding)과 랜덤 리니어 네트워크 코딩의 단점을 보완하고자 최근 제안된 파이프라인 네트워크 코딩(Pipeline Network Coding)의 병렬처리 성능을 비교한다. 또한, 네트워크 코딩의 CPU에서의 병렬처리 기법과 GPGPU(General Purpose Graphics Processing Units)에서의 병렬처리 기법을 비교하여 네트워크 코딩의 사용 시 그 파라미터에 따라 적절한 병렬처리 기법을 선택할 필요성이 있음을 보여준다.


**키워드 : 네트워크 코딩, 병렬처리, GPGPU**

# Comparison of Parallelized Network Coding Performance

Seong-Min Choi[†] · Joon-Sang Park[††] · Sanghyun Ahn[†††]


## ABSTRACT

Network coding has been shown to improve various performance metrics in network systems. However, if network coding is implemented as software a huge time delay may be incurred at encoding/decoding stage so it is imperative for network coding to be parallelized to reduce time delay when encoding/decoding. In this paper, we compare the performance of parallelized decoders for random linear network coding (RLC) and pipeline network coding (PNC), a recent development in order to alleviate problems of RLC. We also compare multi-threaded algorithms on multi-core CPUs and massively parallelized algorithms on GPGPU for PNC/RLC.


**Keywords : Network Coding, Parallel Algorithm, GPGPU**

## 1. 서 론

네트워크 코딩(Network Coding)[1]이란 일반적인 통신망 구조와는 달리 중간경유노드(또는 라우터)에서 서로 다른 패킷들을 혼합하는 기법을 통칭한다. 네트워크 코딩은 유선 네트워크에서의 멀티캐스트 시 정보 전송량(throughput)의 향상을 가져다 줄 뿐 아니라 무선 네트워크, Peer-to-Peer(P2P)등 다양한 시스템에서 여러 성능지표 향상에 도움이 된다는 사실이 밝혀졌다. 예를 들면, 네트워크 코딩을 BitTorrent 방식의 P2P 시스템에 적용하였을 경우 파일의 다운로드 시간을 크게 줄일 수 있다[2]. 그러나 이런 성능향상을 가져오는 네트워크 코딩 기법에도 몇 가지 문제점들이 존재하는 데 그 중 하나는 네트워크 코딩을 소프트웨어적으로 구현할 경우 부호화, 복호화 과정에서 발생하는 비용 증가문제이다. 가장 일반적으로 사용되는 네트워크 코딩 기법의 하나인 랜덤 리니어 네트워크 코딩(Random Linear Network Coding, RLC)[3]을 소프트웨어적으로 구현할 경우 복호화 과정에서 지연시간이 증가한다. 즉, 부호화된 패킷을 수신하여 원본 패킷을 복원해 내는 과정이 많은 시간을 필요로 하게 된다. 이러한 문제를 해결하기 위해서 최근 RLC의 복호화 과정을 위한 병렬처리 알고리즘들이 제안되었다. 본 논문에서는 RLC의 병렬처리 알고리즘과 함께 최근 RLC의 단점을 보완하기 위해서 제안된 파이프라인 네트워크 코딩(Pipeline Network Coding, PNC)[4]의 병렬처리 알고리즘을 다룬다. 기존의 네트워크 코딩 병렬처리 연구들[5,6,7]은 모두 RLC에 초점이 맞추어져 있는데 반하여 본 논문에서는 PNC의 병렬처리 성능과 그의 RLC에의 비교를 다룬다. 또한 PNC의 다중-코어(multi-core) CPU에서의 성능과 GPGPU(General Purpose Graphics Processing Units)에서의 성능을 비교하고 이 과정을 통해서 PNC의 사

※ 이 논문은 2010년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(20100005334, 20100027410).
† 준 회 원 : 홍익대학교 컴퓨터공학과 학사
†† 종신회원 : 홍익대학교 컴퓨터공학과 조교수
††† 종신회원 : 서울시립대학교 컴퓨터과학부 정교수
논문접수 : 2012년 6월 13일
심사완료 : 2012년 7월 3일
* Corresponding Author : Joon-Sang Park(jsp@hongik.ac.kr)

용 시 시스템 성능의 최적화를 위해서는 시스템 환경과 코딩 파라미터에 따라 적절한 병렬처리 기법의 선택이 필요함을 보여준다.

본 논문의 구성은 먼저 2장에서는 RLC과 PNC의 병렬처리 기법을 소개하고 3장에서는 실험 결과와 그에 대해 분석을 제시하고 마지막 4장에서는 결론을 도출한다.

## 2. 네트워크 코딩의 병렬 구현

### 2.1 네트워크 코딩

네트워크 코딩 기법을 이용한 데이터 전송을 위해서 먼저 전송할 데이터 원본을 가지고 있는 소스 노드에서는 전송할 데이터를 동일한 크기의 블록 $p_1, p_2, p_3 \ldots$ 으로 분할한다. (여기서 첨자는 양의 정수 내의 유일하고 연속적인 시퀀스 번호들을 나타냄.) 이러한 블록들은 다시 제너레이션 단위로 논리적으로 분할된다. 특정 제너레이션은 인접한 시퀀스 번호를 갖는 블록의 집합이다. 동일한 제너레이션에 속한 블록들의 어떤 선형조합을 만드는 것이 네트워크 코딩에서의 부호화 작업이다. 먼저 RLC의 경우 다음과 같은 수식으로 부호화작업을 표현한다.

$$c = \sum_{k=1}^{n} e_k p_k \tag{1}$$

(식 1)에서 $n$은 제너레이션 크기이고, 각 $e_k$는 어떤 유한 필드 F에서 임의적으로 선택한 값이다. 부호화된 패킷 $c$의 헤더에는 인코딩 벡터 $e = [e_1, \ldots, e_n]$가 포함되어 $n$개이상의 서로 다른 $c$가 만들어져 전송된다.

PNC는 RLC에서의 부호화/복호화 지연 시간을 감축시키는 것을 목표로 하는 방식이다. 기존의 RLC방식에서는 일정 개수 이상의 패킷이 모여야 부호화/복호화가 가능하다. 즉, 다수의 패킷이 모여 하나의 제너레이션을 이루고 이 제너레이션이 부호화/복호화의 단위가 되기 때문에 하나의 제너레이션에 속하는 모든 패킷이 모여야 부호화/복호화가 가능하다. 그러나 PNC에서는 제너레이션의 모든 패킷이 발생되거나 수신되기를 기다리지 않고 하나의 패킷의 도착과 동시에 부호화/복호화가 이루어 질 수 있어 지연시간을 크게 줄일 수 있다. 응용 계층에서 블록 $p_s$가 생성되어 라우팅 계층으로 전달되었을 때, 라우팅 계층은 이전에 저장된 $p_s$와 같은 제너레이션 식별자를 갖는 블록들을 사용하여 하나의 부호화된 패킷 $c_{(g,s)}$을 생성한다. 부호화된 패킷 $c_{(g,s)}$는 제너레이션 $g$에 존재하는 블록들의 선형 조합이다. 이 선형조합은 다음 식으로 나타낼 수 있다.

$$c_{(g,s)} = \sum_{k=1}^{s-g+1} e_k p_{k+g-1} \tag{2}$$

(식 2)에서 각 $e_k$는 임의의 값이고 $g$는 제너레이션 식별자 $s$는 블록의 시퀀스 번호를 나타낸다. 즉, $s$는 패킷 $c_{(g,s)}$을 생성하기 위해 사용한 블록의 시퀀스 번호 중에서 가장 큰 것을 나타낸다. PNC의 경우 라우팅 계층에서는 부호화된 패킷을 생성하기 위해서 하나의 제너레이션에 속하는 모든 블록이 축적되기를 기다리지 않고 매번 새로운 블록이 도착할 때 마다 그때까지 축적된 같은 제너레이션에 속하는 블록을 결합한다. 예를 들어, $p_1$이 라우팅 계층에 전달되었다면 부호화된 패킷 $c_{(1,1)} = e_1 p_1$이 전송된다. 그 뒤에, $p_2$가 전달되면, $c_{(1,2)} = e_1 p_1 + e_2 p_2$가 전송된다. 두 가지 네트워크 코딩은 모두 부호화 과정을 행렬 곱 연산으로 표현할 수 있다는 면에서는 동일하나 PNC의 부호화는 (식 3)과 같이 하삼각 행렬과의 곱으로 표현할 수 있는 반면 RLC의 경우 (식 3)의 부호화행렬 $[e_k]$에서 0인 원소가 존재하지 않는다. $p_k$는 원본 블록 $c_k$는 부호화된 패킷을 지칭한다.

네트워크 코딩의 복호화 과정은 부호화된 패킷에 부호화 행렬 $[e_k]$의 역행렬을 구하여 곱하는 과정이다. 기본적으로 이 과정에 가우스-조던 소거법을 사용한다. RLC에서는 복호화를 위해 하나의 제너레이션 크기에 해당하는 만큼의 부호화된 패킷을 전송받아야 한다. 하나의 제너레이션을 이루는 부호화된 패킷을 모두 전송받으면 이를 이용하여 부호화 행렬 $[e_k]$의 역행렬을 구하고, 이 역행렬을 패킷의 부호화된 데이터와 행렬곱 연산을 하여 원본 데이터를 복원 할 수 있다. PNC 역시 기본적으로 방법은 같다. 하지만 그 과정을 간소화 할 수 있고, 전체 부호화된 패킷이 전송되지 않더라도 일부 패킷을 복호화하여 복원해 낼 수 있다. PNC이 이러한 특징을 갖는 것은 (식 3)으로 알 수 있다. 앞서 설명된 부호화 과정을 통해 생성된 부호화된 패킷을 성공적으로 전송받으면 목적지에서는 다음과 같은 하삼각행렬을 생성할 수 있다. 위의 행렬식은 k가 제너레이션의 크기가 되지 않더라도 항상 해를 구할 수 있다. 즉, 부호화된 패킷이 전송된 순서대로 도착한다면 이전에 도착했던 패킷을 이용하여 위와 같은 식을 만들어 원본 프레임을 복원해 낼 수 있다. 예를 들어, $c_{(1,2)}$를 수신하면, $c_{(1,2)} = e_1 p_1 + e_2 p_2$이므로 (이전에 수신한) $p_1$을 이용하여 식의 풀이가 가능하고 $p_2$를 복원해 낼 수 있게 된다.

$$\begin{bmatrix} c_1 \\ \vdots \\ c_2 \end{bmatrix} = \begin{bmatrix} e_1^{(1)} & 0 & \cdots & 0 \\ e_1^{(2)} & e_2^{(2)} & & \\ \vdots & & \ddots & 0 \\ e_1^{(k)} & e_2^{(k)} & \cdots & e_k^{(k)} \end{bmatrix} \begin{bmatrix} p_1 \\ \vdots \\ p_k \end{bmatrix} \tag{3}$$

PNC의 행렬식 표현
Encoding matrix of PNC

### 2.2 병렬화된 복호화 과정

#### 2.2.1 다중코어 CPU에서의 병렬화

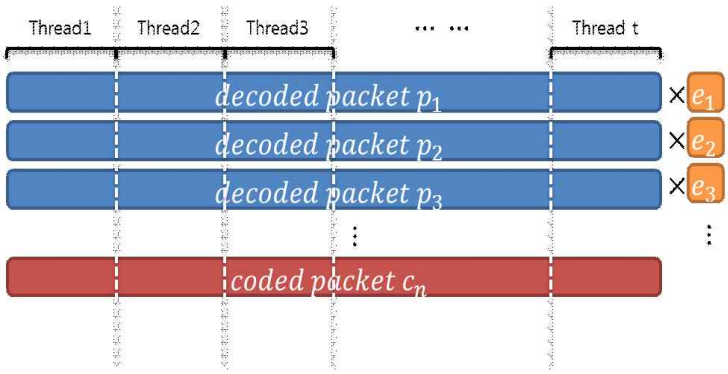



그림 1. 병렬화 기법
Fig. 1. Parallelization Technigue

2개 이상의 코어(Core)를 탑재한 최신 다중코어 CPU에서의 병렬처리 방법은 다수의 스레드(thread)를 활용하는 다중 스레드 프로그램 방법이 일반적이다. RLC과 PNC 모두 앞서 설명하였듯이 복호화과 과정은 기본적으로 같다. 복호화 과정은 가우스-조던 소거법을 이용하여 진행된다. PNC에서는 가우스 소거법만을 사용하여 복호화가 가능하므로 병렬화 방법은 크게 다르지 않다. 두 가지 코딩 방법 모두 그림 1과 같이 영역을 나누어 복호화를 진행한다. 가우스/가우스-조던 소거법은 행연산의 조합이다. 모든 행연산은 그림 1과 같이 영역을 수직으로 나누고 각각의 행에 적절한 값을 곱하고 빼는 과정으로 이루어진다. 수직적으로 나누어진 행간에는 독립성이 보장이 되므로 다수의 스레드를 이용하여 병렬로 처리될 수 있다.

2.2.2 GPGPU를 이용한 병렬화

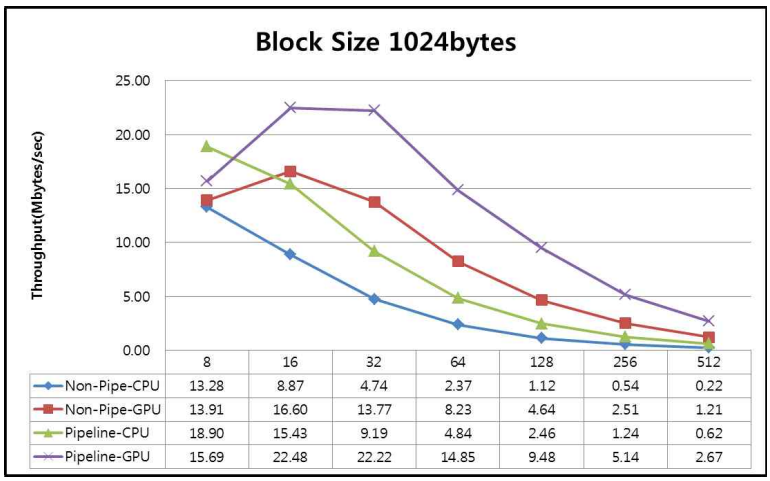
현대의 GPU는 실시간 고해상도 3D 그래픽 처리를 위한 강력한 계산 능력을 가진 하드웨어의 필요성으로 인해 고도로 병렬화되어 많은 수의 코어를 갖도록 발전하였다. 예를 들면, NVIDIA의 GeForce GTX 460 모델은 총 336개의 코어로 구성된다. GPGPU에서의 병렬화를 구현하기 위해서는 먼저 데이터를 균일한 크기로 나눠서 스레드블록(또는 워크그룹)에 분배하고 각각의 스레드블록이 개별적으로 부호화행렬의 복사복을 갖도록 한다. 그림 1과 유사하게 데이터를 일정한 크기로 나누고 이를 스레드블록에게 나누어 준다. 스레드블록은 다수의 GPU스레드를 묶는 단위인데 각각의 스레드블록에 할당된 데이터는 다시 개별 스레드에게 분배되어 처리된다. 결국 아주 많은 수의 스레드를 생성함으로서 동시에 많은 수의 연산이 병렬적으로 이루어지도록 한다. CPU와 다른 점이 있다면 CPU에서는 전체 스레드 간에 동기화가 가능하여 하나의 부호화행렬의 공유가 가능한 반면 GPU에서는 스레드블록 단위로 데이터가 나눠지면 스레드블록 사이에는 동기화가 어려워 개개의 스레드블록이 부호화행렬 복사본을 보유하게 되어 데이터 중복이 발생할 수 있다.

## 3. 실험 및 성능 분석

RLC와 PNC의 성능 그리고 두 가지 병렬화 기법의 성능을 비교 평가하기위하여 다음과 같은 서로 두 가지의 다른 시스템, Intel-i7 960 3.2GHz quad-core CPU와 GeForce GTX460 675MHz GPU가 장착된 고성능 시스템과 Intel Atom D525 1.8GHz CPU와 NVIDIA ION 450MHz GPU를 갖춘 저성능 시스템에서 각각 실험을 수행하였다. 또한 CUDA Toolkit 3.2[8]과 MS Visual Studio 2010 컴파일러를 이용하였다. 모든 그래프의 X축은 제너레이션의 크기를 나타내고 Y축은 처리량을 나타낸다. 처리량(Mbytes/sec 또는 Kbytes/sec)은 복원(복호화)된 데이터의 총량을 복원에 소비한 시간으로 나눈 것이다.

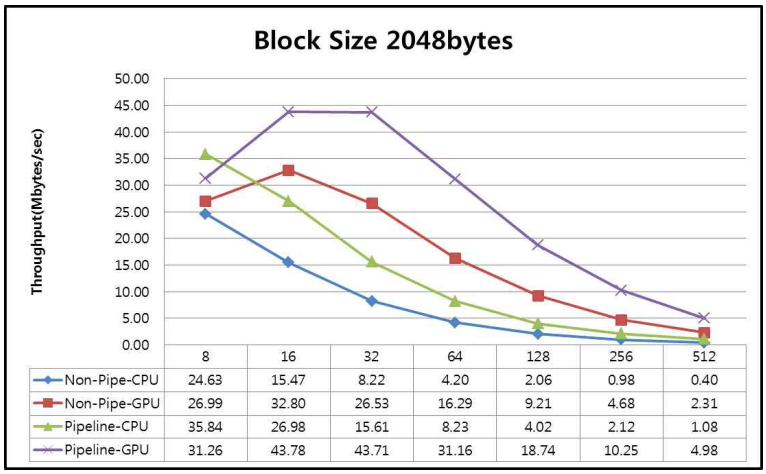
3.1 i7/GTX460에서의 실험 결과

그림 3~5는 제너레이션의 크기를 8부터 512까지 변화시키면, 블록 크기 1024, 2048, 4096, 8192 bytes에 대해 실험한 결과를 보여준다. 그림 2에서와 같이 1024B 블록, 제너레이션 크기 8에서는 RLC의 경우 CPU와 GPU의 성능 차이가 거의 없고, PNC에서는 CPU를 사용한 경우가 약 1.2배 높은 성능을 처리량을 보였다. 그러나 제너레이션의 크기가 커짐에 따라 GPU가 더 높은 처리량을 보였다. 그 차이는 제너레이션 크기가 512인 경우 가장 컸는데 RLC의 경우 약 5.5배, PNC에서는 6.4배 높은 처리량을 보인다. 그림 3과 그림 4에서 볼 수 있듯이 2048B와 4096B 블록에서도 제너레이션 크기가 512일 때, 가장 큰 성능 차이를 보이고, GPU를 사용한 경우가 높은 처리량을 보였다. 또한 모든 경우에서 제너레이션 크기가 16일 때 가장 높은 처리량을 기록했다. 8192B 블록, 제너레이션 크기 16인 경우 RLC가 109.14MB/sec, PNC가 157.17MB/sec의 처리량을 GPU를 통해 얻을 수 있었다. 그림 5에서와 같이 8192B 블록인 경우에서는 GPU의 처리량이 전 구간에 높다. 가장 큰 성능 차이는 제너레이션 크기 512에서 GPU가 CPU에 비하여 RLC의 경우 12배, PNC의 경우는 17배 높은 처리량을 갖는다.



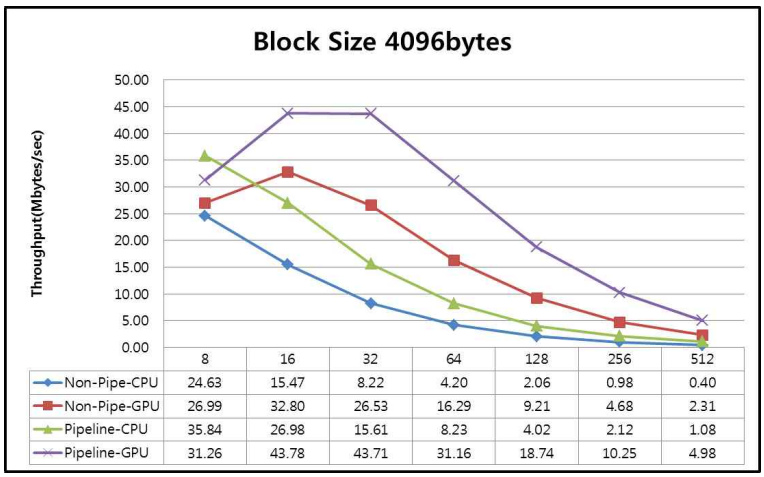

| | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|
| Non-Pipe-CPU | 13.28 | 8.87 | 4.74 | 2.37 | 1.12 | 0.54 | 0.22 |
| Non-Pipe-GPU | 13.91 | 16.60 | 13.77 | 8.23 | 4.64 | 2.51 | 1.21 |
| Pipeline-CPU | 18.90 | 15.43 | 9.19 | 4.84 | 2.46 | 1.24 | 0.62 |
| Pipeline-GPU | 15.69 | 22.48 | 22.22 | 14.85 | 9.48 | 5.14 | 2.67 |

그림 2. 블록 크기 1024bytes
Fig. 2. Block size of 1024bytes



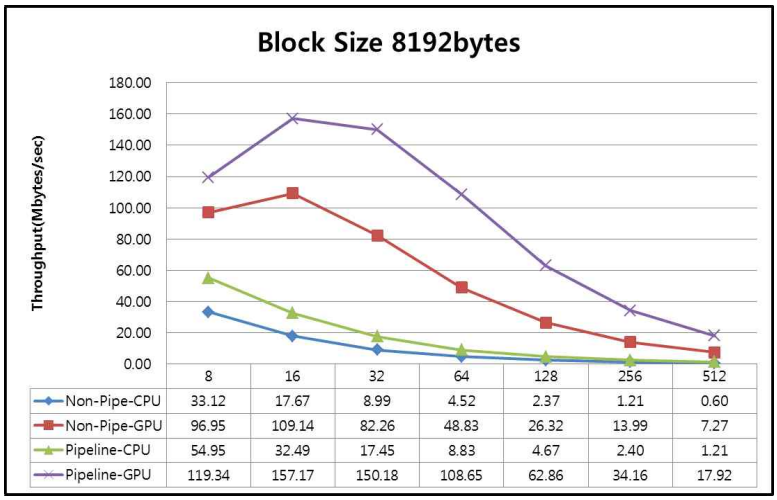

| | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|
| Non-Pipe-CPU | 24.63 | 15.47 | 8.22 | 4.20 | 2.06 | 0.98 | 0.40 |
| Non-Pipe-GPU | 26.99 | 32.80 | 26.53 | 16.29 | 9.21 | 4.68 | 2.31 |
| Pipeline-CPU | 35.84 | 26.98 | 15.61 | 8.23 | 4.02 | 2.12 | 1.08 |
| Pipeline-GPU | 31.26 | 43.78 | 43.71 | 31.16 | 18.74 | 10.25 | 4.98 |

그림 3. 블록 크기 2048bytes
Fig. 3. Block size of 2048bytes



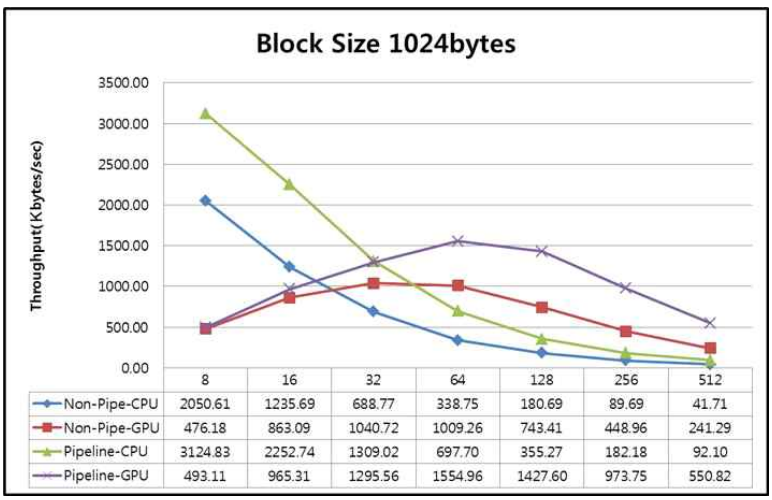

| | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|
| Non-Pipe-CPU | 24.63 | 15.47 | 8.22 | 4.20 | 2.06 | 0.98 | 0.40 |
| Non-Pipe-GPU | 26.99 | 32.80 | 26.53 | 16.29 | 9.21 | 4.68 | 2.31 |
| Pipeline-CPU | 35.84 | 26.98 | 15.61 | 8.23 | 4.02 | 2.12 | 1.08 |
| Pipeline-GPU | 31.26 | 43.78 | 43.71 | 31.16 | 18.74 | 10.25 | 4.98 |

그림 4. 블록 크기 4096bytes
Fig. 4. Block size of 4096bytes




| | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|
| Non-Pipe-CPU | 33.12 | 17.67 | 8.99 | 4.52 | 2.37 | 1.21 | 0.60 |
| Non-Pipe-GPU | 96.95 | 109.14 | 82.26 | 48.83 | 26.32 | 13.99 | 7.27 |
| Pipeline-CPU | 54.95 | 32.49 | 17.45 | 8.83 | 4.67 | 2.40 | 1.21 |
| Pipeline-GPU | 119.34 | 157.17 | 150.18 | 108.65 | 62.86 | 34.16 | 17.92 |

그림 5. 블록 크기 8192bytes
Fig. 5. Block size of 8192bytes




| | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|
| Non-Pipe-CPU | 2050.61 | 1235.69 | 688.77 | 338.75 | 180.69 | 89.69 | 41.71 |
| Non-Pipe-GPU | 476.18 | 863.09 | 1040.72 | 1009.26 | 743.41 | 448.96 | 241.29 |
| Pipeline-CPU | 3124.83 | 2252.74 | 1309.02 | 697.70 | 355.27 | 182.18 | 92.10 |
| Pipeline-GPU | 493.11 | 965.31 | 1295.56 | 1554.96 | 1427.60 | 973.75 | 550.82 |

그림 6. 낮은 성능의 처리장치
Fig. 6. Results on Atom/Ion system

또한 모든 블록크기에 대해서 GPU의 경우 제너레이션의 크기가 8인 경우보다 16인 경우가 처리량이 높았는데 이는 GPU에서의 작업을 위하여 CPU 메모리에 존재하는 데이터를 GPU의 메모리로 옮기는데 필요한 고정비용이 높기 때문이다. 상대적으로 작은 양의 데이터를 전송할 때에는 데이터가 두 배로 증가하더라도 전송시간이 두 배로 증가하지 않고 따라서 처리량이 증가한다.

### 3.2 Atom/ION에서의 실험결과

그림 6은 저성능 시스템에서 1024B의 블록 크기와 8부터 512까지의 제너레이션에 대해 성능을 측정한 결과를 보여준다. PNC의 경우 CPU가 제너레이션 크기 8부터 16까지는 큰 차이로 GPU에 비하여 높은 성능을 보인다. 32에서는 근소한 차이로 GPU의 성능이 조금 낮았다. 가장 큰 차이를 보인 제너레이션 크기 8에서는 CPU의 성능이 RLC는 433%, PNC에서는 633% 높았다. 그 이후 구간에서는 GPU의 성능이 높았다. 제너레이션의 크기가 점점 커질수록 RLC과 PNC의 성능 차이가 커진다. 제너레이션의 크기가 512일 때 PNC의 성능이 RLC의 2배 이상이다.

## 4. 결 론

본 논문에서는 RLC와 PNC에 대하여 CPU와 GPGPU를 이용하여 병렬 처리하는 방법을 비교 평가하였다. 성능 평가 결과 RLC보다 PNC가 높은 처리량을 보여주었다. 이는 PNC의 연산량이 RLC보다 적기 때문이다. CPU와 GPGPU의 성능을 비교해 보면, 전체적으로 GPGPU의 성능이 월등하였지만, CPU의 처리량이 높은 일부 구간이 존재하였다.

성능이 낮은 CPU/GPU 조합의 시스템에서는 GPGPU를 사용할 때 보다 CPU를 사용할 때 성능이 월등히 높은 경우도 존재하였다. 즉, 작은 크기의 블록과 제너레이션을 사용하는 작은 크기의 파일을 주로 주고받는 네트워크 환경에서는 CPU를 사용하여 복호화를 하는 것이 더 효율적이라는 것을 확인하였다. GPGPU를 사용하여 복호화를 하기위해서는 호스트 메모리에 존재하는 데이터를 GPGPU의 메모리로 옮겨와야 하는 추가 비용이 존재한다. 이러한 이유로 낮은 성능의 GPGPU의 경우 메모리 대역폭이 낮아 오버헤드가 더 커져 CPU보다 큰 차이로 낮은 성능을 보일 수 있다. 앞에서 설명한 일부 구간을 제외하면 GPGPU의 성능이 월등히 높았다. 블록의 크기가 증가할수록 그 차이도 증가했다. 이와 같은 결과를 통해서 네트워크 코딩 시 제너레이션의 크기나 블록의 크기와 같은 파라미터에 따라서 병렬처리 기법을 선택할 필요성이 있음을 확인하였다.

## 참 고 문 헌


[1] R. Ahlswede, N. Cai, S. Li, R. Yeung, Network Information Flow. IEEE Transactions on Information Theory. 46(4), pp.1204-1216, 2000.

[2] C. Gkantsidis, P. Rodriguez, Network Coding for Large Scale Content Distribution, Proc. IEEE Infocom. pp.2235-2245, 2005.

[3] P. Chou, Y. Wu, K. Jain, Practical Network Coding, Proc. Allerton Conference. 2003.

[4] C.-C. Chen, C. Chen, S. Oh, J. - S. Park, M. Gerla, and M. Y. Sanadidi, ComboCoding: Combined intra-/inter-flow network coding for TCP over disruptive MANETs, Journal of Advanced Research, 2(3): 241-252, 2011.

[5] H. Shojania, B. Li, and X. Wang, Nuclei: Graphics accelerated Many-core Network Coding, Proc. of IEEE INFOCOM 2009.

[6] K. Park, J.-S. Park, W. Ro. On Improving Parallelized Network Coding with Dynamic Partitioning. IEEE Trans. Parallel and Distributed Systems, 21(11), Nov., 2010.

[7] P. Vingelmann, P. Zanaty, F.H.P.Fitzek, and H. Charaf. Implementation of random linear network coding on opengl-enabled graphics cards. Proc. European Wireless 2009.

[8] NVIDIA Corporation. NVIDIA CUDA: Programming Guide, Version 4.0, Mar., 2011.



**최 성 민**

e-mail : choi12349@hanmail.net

2012년 홍익대학교 컴퓨터공학과(학사)

관심분야 : 네트워크, 병렬처리



**박 준 상**

e-mail : jsp@hongik.ac.kr

2006년 University of California, Los Angeles(전산학박사)

2007년~현 재 홍익대학교 컴퓨터공학과 조교수

관심분야 : 유무선 통신 및 통신망



**안 상 현**

e-mail : ahn@uos.ac.kr

1986년 서울대학교 컴퓨터공학과(학사)

1988년 서울대학교 컴퓨터공학과(석사)

1989년 University of Minnesota 컴퓨터학과(박사)

1994년~1998년 세종대학교 컴퓨터학과 조교수

1998년~현 재 서울시립대학교 컴퓨터과학부 정교수

관심분야 : 애드혹 네트워크, 센서 네트워크, 홈 네트워크, 이동통신, 라우팅 프로토콜, 인터넷