논문 2016-11-37

OpenCL을 활용한 CPU와 GPU 에서의 CMMB LDPC 복호기 병렬화

(Parallel LDPC Decoder for CMMB on CPU and GPU Using OpenCL)

박 주 열, 홍 정 현, 정 기 석*
(Joo-Yul Park, Jung-Hyun Hong, Ki-Seok Chung)

Abstract : Recently, Open Computing Language (OpenCL) has been proposed to provide a framework that supports heterogeneous computing platforms. By using an OpenCL framework, digital communication systems can support various protocols in a unified computing environment to achieve both high portability and high performance. This article introduces a parallel software decoder of Low Density Parity Check (LDPC) codes for China Multimedia Mobile Broadcasting (CMMB) on a heterogeneous platform. Each step of LDPC decoding has different parallelization characteristics. In this paper, steps suitable for task-level parallelization are executed on the CPU, and steps suitable for data-level parallelization are processed by the GPU. To improve the performance of the proposed OpenCL kernels for LDPC decoding operations, explicit thread scheduling, loop-unrolling, and effective data transfer techniques are applied. The proposed LDPC decoder achieves high performance by using heterogeneous multi-core processors on a unified computing framework.

Keywords : Parallel processing, OpenCL, CMMB, Low density parity check (LDPC) codes

Ⅰ. 서 론

최신 모바일 기기들은 단일 코어 프로세서의 클럭 속도 증가를 통한 성능 향상의 한계를 극복하고자 다양한 멀티 코어 프로세서를 채용하고 있다 [1]. 하나의 시스템에 다양한 하드웨어 가속장치들이 탑재되면서, 각 제조사별 상이한 개발 환경과 하드웨어 장치별 특성을 반영하는 것이 필요하게 되었다. 따라서 표준화된 컴퓨팅 환경을 기반으로 시스템의 성능을 개선하기 위해서 Khronos 그룹에 의해 병렬처리 소프트웨어 인터페이스 표준인 Open Computing Language (OpenCL)가 제안 되었다. OpenCL을 통해 어플리케이션을 작성 할 경우, 동일한 프로그래밍 언어로 작성된 소프트웨어를 다양한 연산장치에 이식 후 구동할 수 있다 [2].

OpenCL 의 높은 수준의 이식성을 통해 이종 멀티코어 시스템의 활용률을 높일 수 있지만, 각 연산장치의 하드웨어 특성을 고려하지 않고 병렬화를 진행 할 경우 성능 향상을 보장할 수 없다. 따라서 본 논문은 OpenCL 프레임워크를 활용해 대표적인 이종 연산장치인 central processing unit (CPU)와 graphics processing unit (GPU)의 특성을 고려해 Low Density Parity Check (LDPC) 복호기를 병렬화 하는 방법을 제안한다.

LDPC는 높은 부호정정 성능을 가진 에러 정정 코드로, 메시지 교환을 통해 에러를 정정하는 알고리즘을 반복적으로 연산한다 [3]. 복호화 과정의 연산량은 패리티 검사 행렬인 H-matrix의 크기에 영향을 받는다 [4]. 따라서 행렬 H 의 크기가 급격히 증가하는 최신 표준의 고속 데이터 전송의 안정성을 확보하기 위해 LDPC 복호기를 병렬처리 하는

*Corresponding Author (kchung@hanyang.ac.kr)
Received: June 24 2016, Revised: Oct. 4 2016, Accepted: Oct. 4 2016.
J.Y. Park, J,H. Hong, K.S. Chung: Hanyang University
※ 이 논문은2015년도 정부 (교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2015R1D1A1A09061079)

ⓒ IEMEK J. Embed. Sys. Appl. 2016 Dec. 11(6) 325-334
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2016.11.6.325

연구가 활발히 진행되고 있다 [5-8]. 기존 연구들을 통해 LDPC 복호기의 성능 향상을 얻을 수 있었지만, 제안된 방법들은 특정 연산장치만을 위한 병렬화 모델을 제안해 이종 컴퓨팅 시스템의 하드웨어 자원을 충분히 활용하지 못했다. 또한 각 LDPC 복호 단계에 적합한 병렬처리 단위를 충분히 반영하지 못한 한계가 있다.

본 논문에서는 OpenCL을 활용해 태스크 단위 병렬화가 적합한 단계는 CPU에서 병렬처리 하고, 데이터 단위 병렬화가 적합한 단계는 GPU에 병렬처리 한다. 또한 작성한 LDPC 복호 OpenCL 커널이 병렬처리되는 타겟 연산장치의 하드웨어 특성을 반영한 스레드 스케줄링 및 데이터 전송 기법을 적용했다. 이종 연산장치를 위한 통합된 연산 프레임워크를 활용한 결과, 기존 China Multimedia Mobile Broadcasting (CMMB) 표준 LDPC 복호기의 C언어로 구현된 디자인 대비 최대 13.64배 및 기존 병렬 복호 디자인 대비 최대 1.29배의 성능 향상을 얻었다.

본 논문은 다음과 같은 구성으로 이루어져 있다. Ⅱ장에서는 OpenCL 및 LDPC 복호화 과정을 설명하고, CMMB 부호의 특성을 설명한다. Ⅲ장에서는 본 논문에서 제안하는 이종 연산장치를 위한 LDPC 복호기 병렬화 방법에 대해 설명한다. 그리고 Ⅳ장에서 실험 환경 및 실험 결과에 대해 서술하였으며 논문의 결론과 향후 계획으로 마무리 하였다.

Ⅱ. OpenCL 및 LDPC 복호화 알고리즘

1. OpenCL 표준 명세

OpenCL은 이종 연산장치를 위한 표준화 된 산업 표준 연산 프레임워크다. OpenCL 프레임 워크는 소프트웨어 개발을 지원하기 위해 C기반의 프로그래밍 언어인 OpenCL C, 라이브러리 및 application programming interfaces (API) 들로 구성되어 있다. OpenCL 표준은 다음과 같이 플랫폼, 실행, 및 메모리 모델로 정의할 수 있다.

1.1 플랫폼 모델

OpenCL 플랫폼 모델은 그림 1과 같이 호스트인 CPU와 하나 이상의 OpenCL Compute Device (CD)로 구성된다. 각 CD는 또한 하나 이상의 Compute Unit (CU)로 구성되며 각 CU는 OpenCL C 함수인 커널이 병렬처리 되는 Processing Element (PE)로 구성된다 [2].

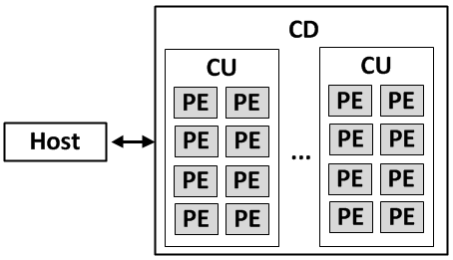

그림 1. OpenCL 플랫폼 모델
Fig. 1 OpenCL platform model

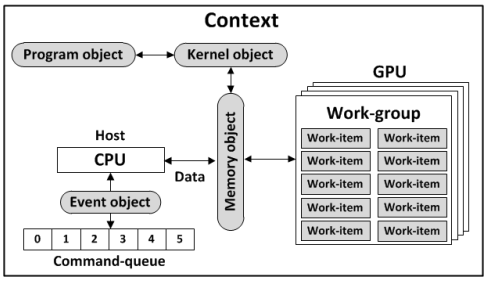

그림 2. OpenCL 실행 모델
Fig. 2 OpenCL execution model

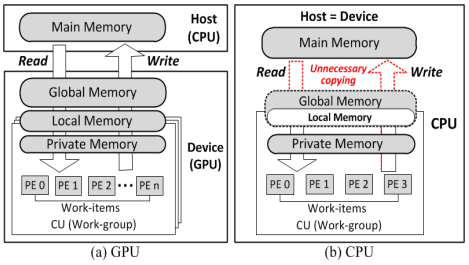

그림 3. OpenCL 메모리 모델
Fig. 3 OpenCL memory model

1.2 실행 모델

실행 모델은 각 PE에서 커널이 어떻게 실행 될지를 정의한다. 호스트를 통해 커널이 연산장치에 할당되고 난 후, PE에서 실행되는 각 커널 실행 인스턴스를 Work-item이라고 정의한다. 각 Work-item 들은 하나의 그룹으로 정의되어 그림 2에서와 같이 CU에 할당된다. OpenCL의 실행은 호스트가 디바이스별 Command-queue에 Read, Write, Copy, Run 명령어를 할당 해 데이터를 연산장치에서 병렬 처리하고, 처리 결과를 읽어오는 과정으로 진행된다. 처리되는 데이터들은 Memory 오브젝트에 저장된다 [2].

1.3 메모리 모델

커널을 병렬 처리하는 Work-item 들은 다양한 연산장치들의 실제 메모리 구조를 신경쓰지 않고도 추상화된 OpenCL 메모리 모델을 사용할 수 있다. OpenCL은 전역 메모리 (Global Memory), 지역 메모리 (Local Memory), 사유 메모리 (Private Memory)의 계층 구조로 메모리 공간을 모델링한다 [2]. 그림 3은 GPU와 CPU의 OpenCL의 메모리 모델을 나타낸다. 그림 3.(a)에서와 같이 CPU와 메모리가 물리적으로 분리된 GPU에서는 메인 메모리에서 전역 메모리로 데이터를 명시적으로 읽고 쓰는 과정이 필요하다. 하지만 그림 3.(b) 에서와 같이 호스트이자 CD인 CPU에서는 메인 메모리와 전역 메모리가 동일한 오브젝트로 정의되기 때문에 명시적인 데이터 전송이 불필요하다 [9]. 또한 CPU와 달리 GPU는 소프트웨어로 컨트롤 가능한 스크래치 패드를 활용해 동일한 Work-group 내부의 Work-item들이 공유할 수 있는 지역 메모리를 사용할 수 있다 [10]. OpenCL 프레임워크를 통해 높은 이식성을 얻을 수 있지만, 이처럼 각 디바이스별 특성을 고려하지 않는다면 어플리케이션의 성능 향상에 어려움이 있다. 따라서 본 논문에서는 다음 단에서 LDPC 복호기의 병렬화 특성을 분석하고, 이종 연산장치인 CPU와 GPU의 하드웨어 자원을 충분히 활용한 병렬화 기법을 제안한다.

2. LDPC 복호화 알고리즘

현재 이동통신 시스템과 차세대 방송 표준에서 오류 정정 부호로 주목받고 있는 LDPC 부호는 1962년에 Gallager에 의해 제안된 block code의 일종이다. 당시의 기술력으로는 구현이 불가능하여 잊혀졌다가 1993년 터보부호의 발견과 더불어 반복 복호 방식에 대한 많은 연구가 이루어 졌으며, 1995년에 Mackay와 Neal에 의해 구현 가능한 방법과 함께 재조명 되었다 [3].

LDPC 코드는 Digital Video Broadcasting Satellite 2nd Generation (DVB-S2) 표준이나 802.11n등 다양한 표준에서 적용되었으며 이를 효율적으로 구현하기 위한 많은 연구들이 진행 되고 있다. 본 논문에서는 4세대 이동 방송 표준의 하나인 CMMB표준에 사용되는 LDPC 부호 복호기를 이종 플랫폼에서 병렬화 하는 방법을 제안한다. CMMB의 패리티 검사 행렬 H는 높은 에러 정정 능력을 갖고 있으며, 부호화율 1/2과 3/4를 선택적으로 적용한다. CMMB 표준의 LDPC 부호 특성은 표 1과 같다 [11].

표 1. CMMB 표준의 LDPC 부호 특성
Table 1. CMMB LDPC coding configuration

Code Rate R	Information K	Length N	Row weight W_c	Column weight W_b
1/2	4,608 bits	9,216 bits	6	3
3/4	6,912 bits	9,216 bits	12	3

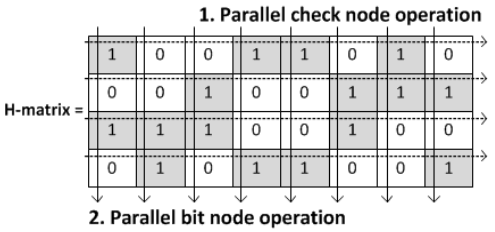

그림 4. LDPC 병렬 복호기
Fig. 4 Parallel LDPC decoder

H 행렬의 각 행과 열은 패리티 체크 코드와 심볼을 나타내며, 각각 체크노드와 비트노드로 부른다. LDPC 복호연산은 크게 4가지로 나누어 볼 수 있다. 초기화 (initialization , INIT), 체크 노드 업데이트 연산 (check node update, CNU), 비트 노드 업데이트 연산 (bit node update, BNU) 그리고 패리티 검사 연산 (parity check, PC)이다. 복호된 값에 에러가 있다고 판별될 경우, 사전에 정의된 횟수만큼 log-likelihood ratio (LLR) 값을 활용해 메시지 전송이 반복된다.

이때 각 체크 또는 비트 노드 연산 시 각각의 행으로 서로의 상관관계가 존재하지 않으므로 동시에 연산을 진행해도 상관이 없다. 그림 4를 예로 들어 설명하면, 주어진 H 행렬의 크기가 (8, 4) 이며. 행렬의 크기에 따라 체크 노드 연산 시 4개의 쓰레드를 생성하여 각 체크 노드 연산을 동시에 진행 할 수 있으며. 비트 노드 연산 시에는 쓰레드를 8개 생성하여 각 열 마다 연산을 동시에 진행 할 수 있다. 이러한 방법이 가능한 이유는 LDPC 복호 알고리즘의 각 연산 마다 메모리 참조에 대한 의존성이 없기 때문이다.

3. LDPC 복호기 병렬화

LDPC 복호 알고리즘은 H 행렬의 사이즈의 크기 증가에 따른 연산량 증가로 소프트웨어로 구현 시에는 원하는 성능을 만족하기 어렵고, 하드웨어 구현 시에도 하나의 표준에서 다양한 부호화율을

지원해야하기 때문에 구현에 대한 어려움이 있다. 최근 멀티코어의 보급증가와 GPU와 같은 하드웨어 가속장치들의 급속한 성능 개선으로 LDPC와 같이 복잡도가 높은 통신 알고리즘을 소프트웨어로 병렬처리 하는 연구가 활발하게 진행 되고 있다. 발표된 논문들은 LDPC 복호 알고리즘을 분석하고 GPU를 활용한 병렬화를 통해 복호 시간을 획기적으로 줄일 수 있음을 보여 주었다 [5-8].

하지만 지금까지 제안된 소프트웨어 복호기들은 CPU와 GPU 각각의 소프트웨어 개발환경에서 병렬화 되어 시스템의 하드웨어 자원을 충분히 활용하지 못하고 있다. 따라서 본 논문에서는 OpenCL을 활용해 CPU와 GPU에서 LDPC 복호 알고리즘을 병렬화하는 방법에 대해 제안한다. OpenCL 프레임워크를 활용해 이종 연산장치들의 특성을 고려한 병렬처리로 하드웨어 자원을 효율적으로 관리하고, LDPC 복호를 소프트웨어로 구현하여 H 행렬의 종류에 따라 다양한 부호화율을 지원하면서 표준에서 요구하는 성능을 만족 할 수 있도록 한다.

Ⅲ. 이종 플랫폼을 위한 LDPC 복호기 병렬화 기법

본 논문에서는 그림 5와 같이 LDPC 복호 과정의 특성을 고려한 OpenCL 커널을 디자인하고, CPU 및 GPU에서 병렬처리 하는 소프트웨어 복호기를 제안한다. CPU와 GPU 커널의 성능 향상 기법을 통해 전체 복호 과정의 성능향상을 얻을 수 있었다.

1. CPU 커널 병렬화 및 성능향상 기법

CPU에서는 먼저 H 행렬 내부의 LLR 값들의 위치를 1차원 배열로 구성해 LDPC 복호기의 병렬화를 돕는 과정을 Task 단위로 병렬 처리한다. 먼저 체크노드를 위한 LLR 값의 위치 L_{addr}는 수식 (1)과 같이 체크노드에 대응되는 비트노드의 위치 x, 비트노드의 순서 y 및 비트노드의 차수 W_B를 이용해 1차원으로 나타낼 수 있다.

$$L_{addr} = x \times W_B + y \quad (0 \le y \le W_B - 1) \qquad (1)$$

마찬가지로 각 비트노드를 위한 LLR값의 1차원 주소 배열 Z_{addr}은 다음과 같이 체크노드의 차수 W_C를 활용해 수식 (2)와 같이 나타낼 수 있다.

$$Z_{addr} = x \times W_C + y \quad (0 \le y \le W_C - 1) \qquad (2)$$

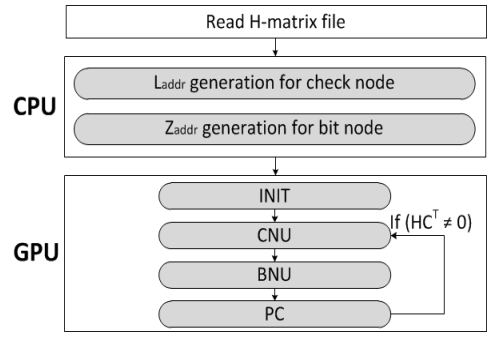

그림 5. 제안하는 이종 컴퓨팅 플랫폼을 위한 LDPC 병렬 복호기

Fig. 5 Proposed parallel LDPC decoder for heterogeneous platform

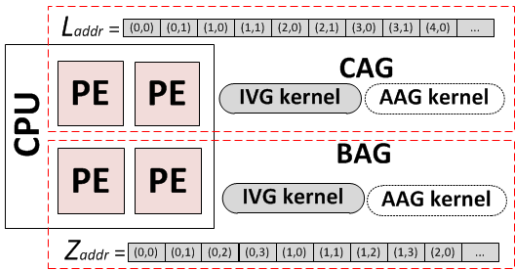

그림 6. 체크노드와 비트노드 1차원 주소생성 과정의 태스크 단위 병렬화

Fig. 6 Task level parallelization for the check node address generation (CAG) and bit node address generation (BAG) kernels

위의 수식으로 계산된 각 비트노드와 체크노드를 위한 위치 배열을 사용할 경우 LLR 값을 사용하는 INIT, CNU, BNU, PC과정의 메모리 접근 횟수가 효과적으로 줄어든다 [11]. 이때 주어진 H 행렬의 LLR값을 읽어 대응하는 노드의 위치와 순서를 고려해 1차원 배열을 생성하는 체크노드 주소생성 (check node address generation, CAG) 과 비트노드 주소생성 (bit node address generation, BAG) 과정은 메모리 접근에 대한 의존성이 없기 때문에 동시에 처리가 가능하다. 따라서 본 논문에서는 그림 6 과 같이 태스크 단위의 병렬 처리에 적합한 CPU에서 CAG와 BAG를 병렬처리 하는 방법을 제안한다.

제안하는 복호기의 CPU 커널은 멀티코어 CPU의 각 코어가 PE로 정의되고, PE에서 독립적인 스레드로 CAG와 BAG 커널의 Work-item을 태스크

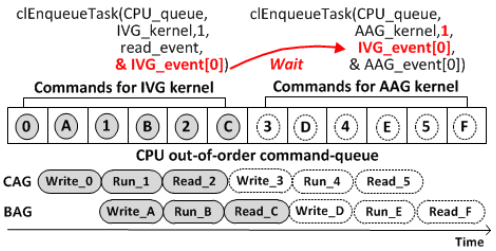

그림 7. 이벤트 오브젝트를 활용한 CPU 커널의 동기화

Fig. 7 Synchronization between proposed CPU kernels using event objects.

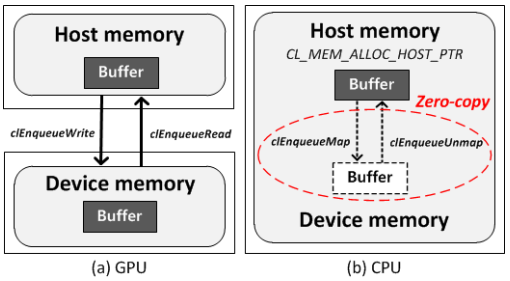

그림 8. CPU 커널을 위한 무복사 기법

Fig. 8 Zero-copy technique for the CPU kernels

단위로 병렬 처리한다. 이때 CPU의 경우 비 순차 command-queue가 지원되기 때문에, 입력되는 명령어의 순서와 상관없이 CPU 코어에 할당된 작업이 끝나면 할당된 작업을 바로 실행해 태스크 단위 병렬처리를 구현하는 것이 가능하다.

이때 비 순차적으로 실행되는 커널들 사이에서도 H 행렬에서 y값을 읽어 인덱스 값을 결정 하는 커널 (index value generation, IVG) 실행 후, 인덱스 값을 기반으로 배열을 생성하는 커널이 (address array generation , AAG) 실행되야 한다. 즉 IVG 커널과 AAG 커널 사이에 동기화가 요구되는데, 이는 그림 7에서와 같이 RUN 커맨드를 실행하는 API에 OpenCL 이벤트 객체를 등록해 의존성을 명시해 구현할 수 있다. AAG 커널은 비 순차 실행 중에도 IVG 커널의 수행이 완료된 후 등록된 이벤트 객체가 반환될 때까지 수행을 대기하게 된다.

이종 컴퓨팅 환경에서 호스트 프로그램은 타겟 연산장치의 메모리에 처리할 데이터를 쓰고, 커널이 처리한 결과를 읽어온다. OpenCL 프레임워크에서 이 과정은 일반적으로 clEnqueueRead 와 clEnqueueWrite application programming interface (API) 로 이루어진다. GPU와 같은 하드웨어 가속장치에서 이러한 데이터 전송과정은 peripheral component interconnect-express (PCI-e)와 같은 외부 연결을 거쳐 수백 사이클 이상의 처리시간 지연이 발생한다. GPU는 수백 개의 멀티 스레드를 스케줄링해서 최대한 많은 데이터를 가져오는 방법으로 이러한 시간지연을 감춘다 [9].

하지만 CAG와 BAG 과정은 CPU에서 태스크 단위로 병렬처리 되기 때문에, 호스트와 타겟 연산장치가 동일하다. 따라서 동일한 메모리에 위치한 데이터를 명시적으로 읽고 쓸 경우, 불필요한 데이터 이동으로 인한 추가적인 오버헤드가 발생한다 [9]. OpenCL에서는 clEnqueueMapBuffer API를 이용해 호스트 주소공간에 위치한 메모리 객체의 포인터를 얻을 수 있고, 호스트와 연산장치 모두 데이터를 읽고 쓸 수 있다 [10]. 따라서 본 논문에서는 그림 8과 같이 커널이 호스트 메모리에 위치한 OpenCL 버퍼 객체에 직접 접근하는 무복사 기법 (Zero-copy)을 통해 CPU 커널의 데이터 전송 효율성을 높였다. 제안하는 기법을 적용한 CPU 커널의 성능 향상은 IV장에 비교되어 있다.

2. GPU 커널 병렬화 및 성능향상 기법

GPU에서 병렬 처리되는 LDPC 복호커널들은 앞에 정의된 것과 같이 INIT, CNU, BNU, PC의 네 단계로 이루어 진다. INIT 단계는 복사된 LLR값들의 위치가 담긴 Zaddr을 활용해 초기화가 이루어진다. CNU에서는 메모리에서 읽어 온 체크노드의 차수를 통해 전달된 메시지의 결정 변수들을 최신화해서 Laddr이 지정한 주소에 저장한다. BNU단계에서는 비트노드에 유사한 연산을 수행하고 Zaddr이 지정한 주소에 결정 변수들을 최신화 한다. 마지막으로 PC 단계에서 결과값이 0인지 검사하는 과정을 통해 복호값의 이상여부를 판별한다 [11].

이러한 과정은 II장에서 살펴본 바와 같이 데이터 단위로 병렬 처리가 가능하기 때문에 GPU의 SIMD 연산기를 충분히 활용해 병렬 처리 하는 것이 유리하다. 일반적으로, GPU의 SIMD 연산기들은 넓은 연산 대역폭을 얻기 위해 very long instruction word (VLIW) 프로그래밍을 활용한다. 이때 VLIW의 스케줄링은 주로 컴파일러에 의해 이루어지므로, OpenCL 컴파일러를 통해 병렬처리 가능한 명시적인 스케줄링이 이루어지도록 GPU 커널의 반복적인 루프를 언롤링 하는 것이 중요하다.

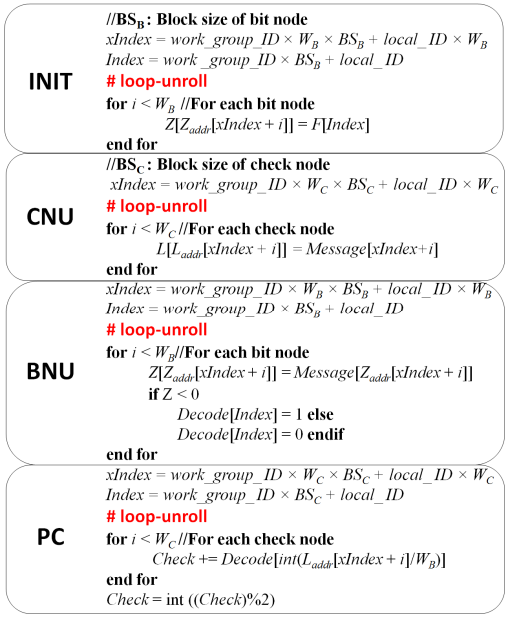

그림 9. GPU 커널을 위한 루프 언롤링
Fig. 9 Loop-unrolling for proposed GPU kernels

그림 9와 같이 루프 언롤링을 통해 컴파일러가 VLIW 슬롯에 명령어를 효율적으로 패킹하는 것이 가능하며, 동시에 수백 개의 스레드를 생성해 병렬처리 하는 GPU의 PE를 효율적으로 활용하는 것이 가능하다 [12].

또한 GPU는 CU가 공유하는 데이터들을 저장하고 Work-group 내부에서 공유할 수 있도록 하는 지역 메모리를 가지고 있다. 지역 메모리를 사용할 경우, 병렬 처리되는 수백개의 Work-item들이 전역 메모리를 접근하는 횟수를 효율적으로 줄일 수 있다 [10]. 제안하는 GPU 커널들에서는 모든 Work-item들이 CPU 커널에서 생성된 Zaddr 과 Laddr 을 참조만 할 뿐 새로운 주소값을 계산하지 않는다. 따라서 본 논문에서는 그림 10과 같이 데이터의 종속성이 없는 1차원 주소 행렬들을 지역 메모리에 저장하고, barrier 함수를 통해 동기화를 맞추는 방법을 통해 데이터 전송 효율을 높였다. 제안하는 기법을 통해 얻은 LDPC 복호기의 성능향상은 다음 장에 자세히 설명되어 있다.

Ⅳ. 실험 환경 및 결과

1. 실험환경

본 논문은 CMMB 표준을 지원하는 LDPC 부호

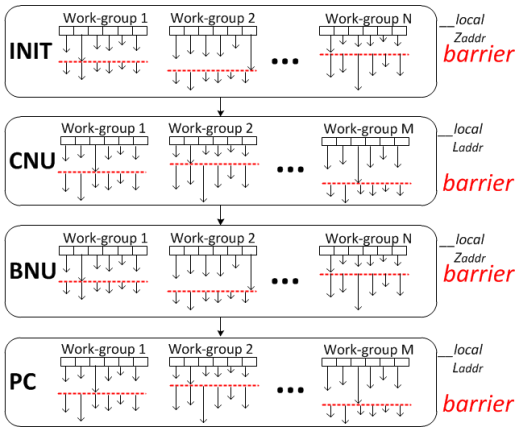

그림 10. GPU 커널을 위한 지역메모리 사용과 동기화 기법
Fig. 10 Local memory allocation and synchronization for proposed GPU kernels

복호기를 OpenCL 프레임워크를 활용해 CPU와 GPU로 구성된 이종 컴퓨팅 환경에서 병렬처리 하는 기법을 제안한다. CMMB 표준은 9216bits 길이의 부호어 길이를 가지며 1/2과 3/4의 두 가지 부호율을 지원한다. 제안하는 LDPC 소프트웨어 복호기는 x86기반 CPU와 AMD 사의 GPU의 컴파일 환경을 지원하는 OpenCL 1.2 표준의 소프트웨어 개발 키트 (software development kit, SDK) 버전 2.9를 활용해 병렬화 했다 [12]. 호스트 프로그램은 마이크로 소프트 Visual Studio 2010 컴파일러를 통해 컴파일 되었다. 호스트 CPU는 Windows 7 Service pack 1 운영체제가 탑재된 인텔의 쿼드코어 i7-3770K이며 타겟 GPU는 AMD의 Radeon HD7800이다.

OpenCL 프레임워크에서 타겟 CPU는 4개의 PE로 구성된 하나의 CU로 정의되며, GPU는 64개의 PE로 구성된 14개의 CU로 정의된다. 본 논문에서는 타겟 이종 플랫폼에서 signal-to-noise (SNR) 값을 변화시켜 가며 제안하는 LDPC 복호기의 성능을 비교했다. 자세한 실험 결과는 다음과 같다.

2. CPU 커널의 성능 향상 비교

본 논문에서는 태스크 단위 병렬화에 적합하고 호스트와 타겟 연산장치가 동일한 CPU의 특성을 반영해 LDPC 복호과정을 위한 LLR 값의 1차원 주소 행렬 생성과정을 OpenCL을 활용해 병렬화 했다. 제안하는 CPU 커널의 성능향상을 측정하기 위해 일반적인 프로그래밍 언어인 C기반 디자인 1과

표 2. 1차원 LLR 주소 행렬 생성 평균 시간
Table 2. Average one-dimensional address generation processing time

SNR	Design 1 (C)	Design 2 (OpenMP)	OpenCL Design (CPU Kernel)		
			Basic Design	Task Parallel	No Copy
Decoding Time Per Frame (CMMB 1/2, Frame: 100,000, ms)					
1	2.535	1.533	2.991	1.609	1.266
1.5	2.413	1.423	3.101	1.810	1.314
2	2.441	1.413	2.895	1.910	1.320
2.5	2.341	1.513	2.901	2.013	1.412
3	2.512	1.559	3.145	2.162	1.512
3.5	2.612	1.412	2.789	1.914	1.390
4	2.531	1.583	2.892	1.729	1.451
Decoding Time Per Frame (CMMB 3/4, Frame: 100,000, ms)					
2.5	4.245	2.522	4.890	2.570	2.006
3	4.124	2.513	4.941	2.513	2.011
3.5	4.351	2.623	4.712	2.894	2.123
4	4.120	2.413	4.541	3.011	2.089
4.5	3.989	2.513	4.824	2.414	2.127
5	4.089	2.642	4.612	2.982	2,102
5.5	3.913	2.551	4.514	2.513	2.114

컴파일러 지시어 기반 병렬처리 표준인 OpenMP [13] 기반 디자인 2와 성능을 비교했다. 타겟 CPU가 쿼드코어기 때문에 OpenMP API를 통해 4개의 스레드로 병렬처리 했다. 표 2에 자세한 실험 결과가 나타나 있다.

실험 결과 CPU에서 OpenCL 디자인을 구동할 경우, 추가적인 OpenCL 오브젝트를 생성하고 관리하는 오버헤드로 인해 병렬화 이전인 디자인 1보다도 복호 시간이 오래 걸리는 것을 알 수 있었다. 하지만 태스크 단위의 병렬처리를 적용할 경우, 성능 차이가 좁혀졌고 무 복사 기법을 통해 불필요한 데이터 전송 오버헤드를 줄이자 OpenMP 디자인보다 높은 성능을 보이는 것을 확인 할 수 있었다. 따라서 OpenCL을 통해 CPU에서 구동 가능한 높은 이식성의 소프트웨어 어플리케이션을 작성할 수 있지만, 성능 향상을 위해 CPU의 하드웨어 특성을 고려하는 것이 중요하다는 것을 알 수 있다.

3. GPU 커널의 성능 향상 비교

Ⅱ장에서 설명한 것과 같이, OpenCL 프레임워크을 활용해 작성한 커널을 병렬처리 하기위해 동시에 활성화 되는 스레드의 수를 Work-item이라고 명시한다. 스레드 블록의 크기를 Work-item들로 구성한 Work-group이라고 하며, 프로그래머는 GPU의 하드웨어 구조를 고려한 그룹 크기를 지정해야 메모리 접근 지연 시간을 효율적으로 감출 수 있다.

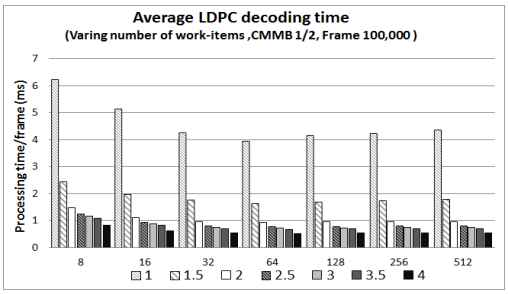

그림 11. Work-item 수의 변화에 따른 GPU 커널의 평균 LDPC 복호 시간
Fig. 11 Average processing time of the GPU kernels with varying numbers of work-items

그림 11은 Work-group의 크기를 변화시켜가며 1/2부호율의 GPU 커널에서 10만 프레임을 처리할 동안 평균 복호 시간을 측정한 결과이다. 이 결과를 통해 타겟 GPU의 PE가 64개이기 때문에 각 커널이 64개의 Work-item으로 병렬 처리될 때 가장 높은 성능을 보이는 것을 알 수 있었다 [12]. 제안하는 소프트웨어 기반 LDPC 복호기는 커널을 다시 프로그래밍 하지 않더라도 OpenCL 프레임워크를 활용해 타겟 연산장치가 지원하는 최대의 하드웨어 스레드를 활용 할 수 있다는 이식성이 가장 큰 장점이다.

표 3은 제안하는 OpenCL GPU 커널들의 성능을 비교한 결과이다. 모든 GPU 커널들은 64개의 Work-item으로 병렬처리 되며, 타겟 GPU의 128bit 넓이의 SIMD 연산기를 최대한 활용하기 위해 float4 벡터 데이터 타입을 사용한다. Ⅲ절에서 설명한 것처럼 GPU의 VLIW 슬롯을 충분히 활용하기 위한 루프 언롤링과 지역 메모리를 사용하는 성능향상 기법을 적용한 결과, C로 작성된 INIT, CNU, BNU 및 PC 복호 과정보다 최대 17.6배, 성능향상기법 적용 전의 OpenCL 디자인보다 최대 2.13배의 복호 성능 향상을 얻을 수 있었다.

4. 전체 LDPC 복호기의 성능향상 비교

표 4는 C 디자인과 OpenMP로 작성된 CPU 타겟의 LDPC 복호기 (가) 과 (나), 기존 이종 컴퓨팅 환경 타겟의 LDPC 복호기 (다) 및 본 논문에서 제안하는 성능향상 기법을 적용한 OpenCL LDPC 복호기 (라)의 성능을 비교하고 있다. 2011년도에 제안된 (다)의 경우, CPU와 GPU를 모두 활용하고 있지만 각각 OpenMP와 Compute Unified Device Architecture (CUDA) 라는 독립적인 컴퓨팅 프레임

표 3. 프레임 당 LDPC 복호 평균 처리 시간
Table 3. Average LDPC decoding time per frame

SNR	Design 1 (C)	Design 2 (OpenMP)	OpenCL Design (GPU Kernel,64 Thread)	
			Basic Design	Proposed Design
Decoding Time Per Frame (CMMB 1/2, Frame: 100,000, ms)				
1	29.39	7.34	3.95	1.81
1.5	12.19	4.92	1.64	0.81
2	7.86	2.41	0.93	0.54
2.5	5.77	1.44	0.78	0.44
3	3.07	0.98	0.72	0.30
3.5	2.35	0.70	0.67	0.26
4	1.73	0.61	0.52	0.23
Decoding Time Per Frame (CMMB 3/4, Frame: 100,000, ms)				
2.5	102.06	28.92	12.43	5.79
3	32.82	11.35	4.32	1.64
3.5	28.15	9.63	3.81	1.27
4	18.68	8.73	3.22	1.01
4.5	15.72	5.82	2.15	0.88
5	12.04	4.25	1.71	0.64
5.5	10.51	3.74	1.62	0.54

표 4. 전체 LDPC 복호 처리 시간 비교
Table 4. Processing time of LDPC decoding

	CPU		CPU+ GPU	
	C (A)	OpenMP (B)	OpenMP + CUDA(C)	OpenCL (D)
Decoding Time (CMMB 1/2, Frame 100,000, SNR: 1, sec)				
Time	319.25	104.73	37.20	30.76
Speedup	10.38	3.40	1.21	-
Decoding Time (CMMB 3/4, Frame 100,000, SNR: 2.5, sec)				
Time	1063.1	314.42	100.9	77.96
Speedup	13.64	4.03	1.29	-

워크를 이용해 복호기를 병렬처리 했다 [11, 14]. 실험 결과 통합된 컴퓨팅 프레임워크로 CPU와 GPU의 하드웨어 자원을 충분히 활용한 OpenCL LDPC 복호기가 C디자인 대비 최대 13.64배 빨랐으며, 가장 성능이 우수한 것을 확인 할 수 있었다.

V. 결 론

OpenCL 프레임워크를 활용해 병렬처리된 디지털 신호처리 및 통신 어플리케이션들은 다양한 하드웨어 가속장치들을 활용해 높은 성능과 이식성을 얻을 수 있다. 본 논문은 CMMB표준의 LDPC 복호기를 CPU와 GPU로 구성된 이종 플랫폼의 하드웨어 자원을 충분히 활용하는 병렬화 기법을 제안한다. 일반적으로 CPU는 태스크 단위의 병렬화에 적합하며 GPU는 데이터 단위의 병렬화에 적합한 하드웨어 구조를 가지고 있다. 본 논문에서는 테스크 단위 병렬화가 가능한 1차원 주소행렬 생성 커널을 CPU에서 병렬처리하고, 데이터 단위 병렬화가 가능한 LDPC 복호 커널을 GPU에서 병렬처리 했다. 또한 효율적인 데이터 전송을 위해 CPU 커널에는 무복사 기법을 적용했고, GPU 커널은 지역메모리를 활용하도록 작성했다. 그 결과 제안하는 LDPC 복호기는 C 디자인 대비 최대 13.64배, 기존 병렬 복호기 대비 최대 1.29배의 성능 향상을 얻을 수 있었다. 앞으로 전역 메모리 공간이 통합된 모바일 환경과 field-programmable gate array (FPGA)를 포함한 이종 컴퓨팅 환경에서 OpenCL을 활용한 LDPC 복호기의 성능 향상 연구를 진행 할 예정이다.

References

[1] Y.H. Park, C.H. Kim, J.M. Kim, “Implementation and performance evaluation of the faddev-leverrier algorithm using GPGPU,” IEMEK J. Embed. Sys. Appl., No. 8, Vol. 3, 2013 (in Korean).

[2] Khronos OpenCL Working Group, “The OpenCL specification version 1.2,” Document Revision 19, 2012.

[3] R.G. Gallager, “Low-density parity check codes,” IEEE IRE Transactions on Information Theory, Vol. 8, No. 1, pp. 21-28, 1962.

[4] S.M. Choi, B.H. Moon, J.T. Ryu, S.H. Park, “Performance analysis on error correction scheme for wireless sensor network over node-to-node interference,” IEMEK J. Embed. Sys. Appl., Vol. 2, No. 1, 2006 (in Korean).

[5] S. Wang, S. Cheng, Q. Wu, “A parallel decoding algorithm of LDPC codes using CUDA,” Proceedings of Asilomar Conference on Signals, Systems and Computers, pp. 171-175, 2008.

[6] G. Falcão, S. Leonel, S. Vitor, “Massive parallel LDPC decoding on GPU,” Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, pp. 83-90, 2008.

[7] H.W. Ji, J.H. Cho, W.Y. Sung, "Memory access optimized implementation of cyclic and quasi-cyclic LDPC codes on a GPGPU," Journal of Signal Processing Systems, Vol. 64, No. 1, pp. 149-159, 2011.
[8] G. Falcão, V. Silva, L. Sousa, "How GPUs can outperform ASICs for fast LDPC decoding," Proceedings of the 23rd international conference on Supercomputing, pp. 390-399, 2009.
[9] J. Shen, J. Fang, H. Sips, A. L. Varbanescu, "Performance traps in OpenCL for CPUs," Proceedings of IEEE 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing, pp. 38-45, 2013.
[10] B. R. Gaster, L. Howes, D. R. Kaeli, P. Mistry, D. Schadd, "Heterogeneous computing with OpenCL: Revised OpenCL 1.2 Edition," Morgan Kaufmann, 2012.
[11] J.Y. Park, K.S. Chung, "Parallel LDPC decoding using CUDA and OpenMP," EURASIP Journal on Wireless Communications and Networking, Vol. 2011, No. 1, pp. 1-8, 2011.
[12] Advanced Micro Devices, "AMD accelerates parallel processing OpenCL programming guide," 2013.
[13] D. Leonardo, R. Menon, "OpenMP: an industry standard API for shared-memory programming," IEEE Computational Science and Engineering, Vol. 5, No.1, pp. 46-55, 1998.
[14] Nvidia, "Compute unified device architecture programming guide Version 2.0," 2008.

Joo-Yul Park (박 주 열)

He received his B.S. in Electronic Engineering from Ajou University, Suwon, Korea, in 2004, and his Ph.D. in Electronics, Computer, and Communication from Hanyang University, Seoul, Korea, in 2013. He was an engineer at LG Electronics Corp. in Seoul from 2005 to 2007. Since 2013, he has been an Adjunct Professor at Hanyang University, Seoul, Korea. His research interests include DSP design, channel coding.

Email: jooyul.park@gmail.com

Jung-Hyun Hong (홍 정 현)

He received his B.S. degree in Media Communication Engineering from Hanyang University, Seoul, Korea, in 2011. Since 2011, he has undertaken a unified M.S. and Ph.D. course at Hanyang University, Seoul, Korea. His research interests include software parallelization, heterogeneous computing.

Email: ongstar304@naver.com

Ki-Seok Chung (정 기 석)

He received his B.E. degree in Computer Engineering from Seoul National University, Seoul, Korea, in 1989 and his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1998. He was a Senior R&D Engineer at Synopsys, Inc. in Mountain View, CA, from 1998 to 2000 and was a Staff Engineer at Intel Corp. in Santa Clara, CA, from 2000 to 2001. He also worked as an Assistant Professor at Hongik University, Seoul, Korea, from 2001 to 2004. Currently, he is a Professor at Hanyang University, Seoul, Korea. His research interests include low-power embedded system design, multi-core architecture, image processing, reconfigurable processor and DSP design, SoC-platform-based verification, and system software for MPSoC.

Email: kchung@hanyang.ac.kr