논문 2010-47SD-11-7

멀티코어 시스템을 위한 멀티스레드 H.264/AVC 병렬 디코더

(Multi-Threaded Parallel H.264/AVC Decoder for Multi-Core Systems)

김 원 진*, 조 걸**, 정 기 석***

(Won-Jin Kim, Keol Cho, and Ki-Seok Chung)

요 약

고해상도의 동영상 서비스가 보편화 되면서 동영상을 빠르게 처리를 위한 연구가 활발히 이루어지고 있다. 멀티코어 프로세서의 사용이 증가하고 멀티코어 시스템에서 H.264/AVC 디코더를 구현하기 위하여 다양한 병렬화 방법이 제안되고 있다. 하지만 H.264/AVC 디코더를 병렬화 하는 경우, 각 스레드에서 처리하는 데이터의 처리 시간 차이로 인하여 지속적으로 스레드의 동기를 확인해야 하는데, 이는 병렬화를 통한 디코더의 성능 향상의 걸림돌이 된다. 이러한 병렬화 과정에서 발생하는 문제점을 해결하기 위해 우리가 제안하는 Multi -Threaded Parallelization(MTP) 방법은 프레임을 매크로 블록 묶음으로 나누어 병렬화 한다. 그리고 병렬화 과정에서 스레드를 처리하는 방법을 개선하고, 메모리를 재사용함으로써 디코더의 성능을 향상 시켰다. 본 논문에서는 FFmpeg H.264/AVC 디코더를 인텔 쿼드 코어 기반의 멀티코어 시스템에서 멀티 스레드로 구현하여 실험이 진행되었다. 그 결과, MTP 방법을 적용하여 병렬화 방법 적용하지 않은 H.264/AVC 디코더와 비교하여 최대 53%의 성능 향상을 보였으며, 2Dwave 병렬화 방법의 메모리 사용량에 비해 HD 영상에서 65%, FHD 영상에서 81%의 메모리 사용량을 줄 일 수 있었다.

Abstract

Wide deployment of high resolution video services leads to active studies on high speed video processing. Especially, prevalent employment of multi-core systems accelerates researches on high resolution video processing based on parallelization of multimedia software. In this paper, we propose a novel parallel H.264/AVC decoding scheme on a multi-core platform. Parallel H.264/AVC decoding is challenging not only because parallelization may incur significant synchronization overhead but also because software may have complicated dependencies. To overcome such issues, we propose a novel approach called Multi-Threaded Parallelization(MTP). In MTP, to reduce synchronization overhead, a separate thread is allocated to each stage in the pipeline. In addition, an efficient memory reuse technique is used to reduce the memory requirement. To verify the effectiveness of the proposed approach, we parallelized FFmpeg H.264/AVC decoder with the proposed technique using OpenMP, and carried out experiments on an Intel Quad-Core platform. The proposed design performs better than FFmpeg H.264/AVC decoder before the parallelization by 53%. We also reduced the amount of memory usage by 65% and 81% for a high-definition(HD) and a full high-definition(FHD) video, respectively compared with that of popular existing method called 2Dwave.

Keywords : H.264/AVC, parallel processing, decoder, frame partitioning

* 학생회원, ** 학생회원, 한양대학교 전자컴퓨터공학과
(Department of Electronics, Computer & Communication Engineering, Hanyang University)
*** 정회원, 한양대학교 융합전자공학부
(Department of Electronics Engineering, Hanyang University)
※ 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2010-0024164). 그리고 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (NIPA-2010-C1090-1031-0009)
접수일자: 2010년7월7일, 수정완료일: 2010년10월25일

Ⅰ. 서 론

디지털TV가 보편화 되고 사용자들은 고해상도의 영상 서비스를 요구하고 있다. 이로 인하여 고해상도 동영상 서비스를 위한 동영상 압축 처리에 대한 연구가 활발히 진행되고 있다. 대표적으로 H.264/AVC는 현존하는 가장 압축률이 우수한 성능의 비디오 부호화 표준으로, 디지털 방송, 멀티미디어 플레이어, 화상회의 등 멀티미디어 서비스 분야에서 많이 사용되고 있고 다양한 연구가 이루어지고 있다. H.264/AVC는 높은 압축률을 지원하기 위하여 기존의 비디오 코덱에 비교하여 복잡도가 높은 알고리즘을 사용하고 있다. 이러한 H.264/AVC 코덱을 처리하기 위해서는 고성능의 프로세서 사용이 필요하다. 기존의 싱글 코어 기반의 프로세서는 클록 속도를 올려서 성능을 향상시키기 때문에 프로세서의 발열 및 소비전력의 증가로 인하여 성능 향상의 한계에 직면 하였다. 이러한 문제를 해결하기 위하여 프로세서 코어 수를 늘리고 병렬 처리를 이용하는 멀티 코어 시스템의 연구가 지속적으로 이루어지고 있다. 그러나 기존의 싱글 코어 프로세서 시스템에서 사용한 프로그래밍 모델은 멀티 코어 시스템의 성능을 충분히 활용할 수 없다. 따라서 프로그램을 멀티 코어 시스템에서 효과적으로 실행하기 위하여 병렬프로그래밍으로 바꾸어야 한다. 그리하여 본 논문에서는 고해상도에서 H.264/AVC 디코더의 성능을 개선하기 위한 효과적인 병렬화 방법을 연구한다.

대표적인 H.264/AVC 디코더 병렬화 방법으로 태스크 레벨 병렬화 방법과 데이터 레벨 병렬화 방법이 있다. 태스크 레벨 병렬화 방법은 하나의 작업을 여러 개의 작업으로 나누어 각 스레드에서 각각을 수행하는 방법이다. 그리고 데이터 레벨 병렬화 방법은 H.264/AVC 데이터를 병렬화가 가능하도록 나누어 여러 스레드에서 동시에 처리하는 방법이다. 그런데 태스크 레벨 병렬화 방법은 H.264/AVC 디코더의 기능 별로 처리 시간 다르기 때문에 성능 향상에 어려움이 있다. 그리고 데이터 레벨 병렬화 기법은 H.264/AVC의 데이터 의존성을 지키면서 병렬화를 진행해야 한다. 이러한 문제점을 해결하기 위하여 새로운 H.264/AVC 디코더 병렬화 방법이 필요하다.

본 논문은 고해상도 동영상을 고속으로 처리하기 위한 H.264/AVC 디코더 병렬화 방법을 연구 한다. 우리가 제안하는 Multi-Threaded Parallelization(MTP) 방법은 병렬화 과정에서 발생하는 문제점을 해결하여 H.264/AVC 디코더 성능을 향상 하였다. 본 논문은 다음과 같은 순서로 이루어져 있다. Ⅱ장에서는 H.264/AVC 디코더에 대하여 간단하게 살펴보고, 기존의 H.264/AVC 디코더의 병렬화 방법에 대하여 살펴본다. 그리고 Ⅲ장에서는 본 논문에서 제안하는 MTP 방법에 대하여 설명하였다. Ⅳ장에서 실험 환경 및 실험 결과에 대하여 서술하였고 마지막으로 논문의 결론과 향후 계획으로 마무리 하였다.

Ⅱ. 관련 연구

1. H.264/AVC 디코더

H.264 혹은 MPEG-4 AVC는 ISO/IEC와 ITU가 함께 만든 비디오 압축 표준으로 기존에 사용되는 비디오 압축 표준보다 더 높은 압축률 가지며, 네트워크를 통한 스트리밍에 적합한 특성을 가지고 있어서 다양한 멀티미디어 응용 분야에 많이 쓰이고 있다. H.264/AVC 표준 및 사용되는 알고리즘에 대한 자세한 사항은 ITU의 H.264 표준[1], ISO MPEG-4/AVC 표준[2] 문서와 H.264/AVC을 분석한 논문[3~4]에 잘 기술되어 있다. H.264/AVC 디코더는 Entropy Decoding(ED), Inverse Transformation(IT), Inverse Quantization(IQ), Intra Prediction(IP), Motion Compensation(MC), Deblocking Filter(DF)로 구성 된다. 간단하게 기능별로 살펴보면 아래와 같다.

가. Entropy Decoding(ED)

H.264/AVC에서 비트 스트림은 Network Adaptation Layer(NAL) 단위로 입력되어 Entropy Decoding(ED)에서 계수(Coefficient)를 생성 한다. H.264/AVC에서는 문맥(Context)에 따라 효율적인 엔트로피 코딩이 가능한 형태의 Context-Adaptive Variable Length Coding (CAVLC)과 Context-Adaptive Binary Arithmetic Coding(CABAC) 두 가지의 엔트로피 디코딩을 지원한다. 베이스라인 프로파일은 CAVLC, 메인 프로파일과 하이 프로파일은 CABAC를 사용한다.

나. Inverse Quantization /Inverse Transformation (IQ/IT)

엔트로피 디코딩 작업 후 생성되는 계수(Coefficient)들은 IQ/IT 과정 후에 residual data로 만들어진다.

다. Intra Prediction(IP),
Motion Compensation(MC)

IP 과정은 공간적 중복성을 이용하여 화면 내 예측을 하고 MC는 시간적 중복성을 사용하여 화면간의 예측을 수행한다. 그리고 매크로 블록의 타입에 따라서 IP를 적용하거나, MC를 적용한다. IP, MC 후 만들어지는 블록은 IQ/IT 후 에 만들어지는 residual data와 합쳐진다.

라 Deblocking Filter(DF)

디코딩 되어진 영상은 블록간의 경계가 뚜렷이 나타나는 블록킹 현상이 나타난다. 이를 없애기 위해서 블록 경계를 부드럽게 하는 DF를 적용한다. H.264/AVC의 DF는 가중치 값을 제어하면서 적용하는 적응형(Adaptive) 방법을 사용 한다

2. H.264/AVC 병렬화 디코더 관련연구

H.264/AVC는 그림 1과 같은 데이터 구조로 이루어져 있다. H.264/AVC의 비디오 시퀀스는 Group Of Pictures(GOP)로 구성된다. 그리고 각 GOP는 여러 개의 픽쳐(Picture) 또는 프레임(Frame)으로 구성된다. 그리고 하나의 픽쳐는 하나의 슬라이스(Slice) 또는 여러 개의 슬라이스들로 구성된다. 그리고 슬라이스는 여러 개의 매크로 블록(Macroblock)으로 구성되며, 매크로 블록은 휘도 블록(Luma block)과 색차 블록(Chroma block)으로 이루어진다.

이러한 H.264/AVC 데이터 단위에 따라서 다양한 H.264/AVC 디코더 병렬화 방법[5~8]이 연구 되었다. 논문[9~10]에서는 H.264/AVC 병렬화 디코더에서 데이터를 스케줄링 하는 방법을 제안하였다. 그리고 논문[11]에서

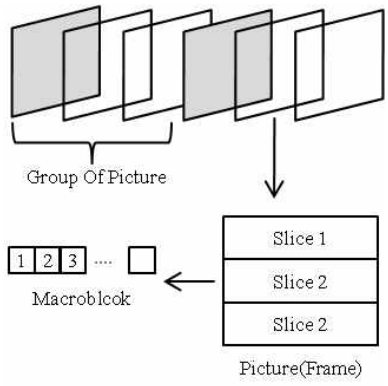


그림 1. H.264/AVC 데이터 구조
Fig. 1. H.264/AVC video frames.

는 ED을 처리하면서 이전 프레임의 DF를 처리 하는 병렬화 방법을 제안하고 있다. 대표적인 H.264/AVC 디코더 병렬화하는 방법인 태스크 레벨 병렬화 방법과 데이터 레벨 병렬화 방법을 좀 더 상세하게 살펴보면 다음과 같다.

가. 태스크 레벨 병렬화 방법

태스크 레벨 병렬화 방법은 H.264/AVC 디코더 태스크를 각 기능별로 나누어 스레드에 할당하여 실행함으로써 성능을 높이는 병렬화 방법이다[8]. 일반적인 병렬화 방법인 파이프라인 병렬화 방법과 같다. 그림 2는 태스크 레벨 병렬화인 매크로 블록 단위로 파이프라인 병렬화 방법을 보여준다. 이 방법의 주요 제한점은 H.264/AVC 디코딩 과정에서 각 기능별로 데이터를 처리하는 시간이 다르다는 것이다.

H.264/AVC 디코딩 과정에서 각 기능별로 매크로 블록을 처리 시간이 다른 이유는 매크로블록 타입에 따라서 처리 시간이 다르기 때문이다. H.264/AVC는 세 가지 타입의 매크로블록이 있다. intra-coded 매크로블록은 공간적 중복성을 이용하여 인코딩 된 블록이다. inter-coded 매크로블록은 시간적 중복성을 이용하여 인코딩 된 블록이다. skipped 매크로블록은 참조 프레

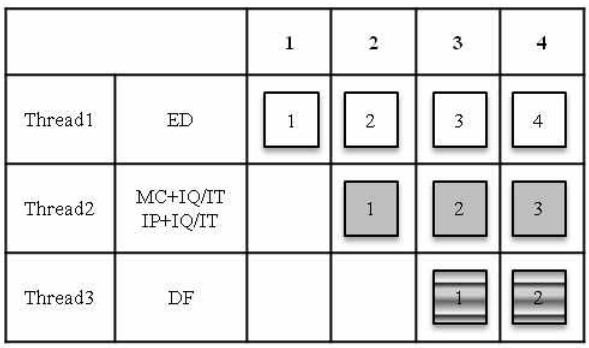


그림 2. 파이프라인 병렬화 방법
Fig. 2. Pipelined parallel processing.

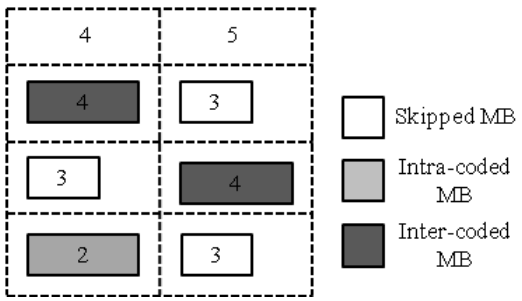


그림 3. 매크로 블록 처리 시간
Fig. 3. Processing time of macroblocks.

임의 매크로 블록과 차이가 극히 적은 블록으로 디코딩 과정에서 참조 프레임의 매크로 블록을 그대로 사용한다. 그러므로 skipped 매크로블록은 ED, IQ/IT 과정이 필요 없다. 따라서 그림 3에서 매크로 블록 3이 skipped 매크로블록 타입이면 처리 시간이 짧다. 그리고 매크로 블록 4와 2가 intra-coded 매크로블록 또는 inter-coded 매크로블록이면 처리 시간 길어진다. 그러므로 매크로 블록의 타입에 따라 처리 속도의 차이가 발생한다. 그리고 파이프라인의 한 단계의 처리 시간은 가장 시간이 오래 걸리는 매크로 블록의 처리 시간이 된다. 그래서 전체적인 처리 시간이 증가 하게 된다.

나. 데이터 레벨 병렬화 방법

데이터 레벨 병렬화 방법은 H.264/AVC 데이터를 병렬화가 가능하게 나누어 처리하는 방법이다. H.264/AVC 데이터 단위에 따라 프레임 단위, 슬라이스 단위, 매크로 블록 단위 병렬화 방법으로 나누어진다. H.264/AVC에서 프레임은 공간적 중복성을 이용한 I 프레임, 시간적 중복성을 이용한 P 프레임, 양방향으로 참조 프레임을 사용하는 B 프레임이 있다. 프레임 단위 병렬화는 P 프레임이나 B 프레임 경우 참조 프레임에 대한 의존성이 발생 한다. 이러한 의존성으로 인하여 병렬화 성능 향상에 한계가 발생한다. 그리고 슬라이스 단위 병렬화의 경우 슬라이스 사이의 데이터 의존성은 없지만 한 프레임을 여러 개의 슬라이스 나누어 인코딩 된 영상만 슬라이스 단위로 디코딩 병렬화가 가능하다. 이러한 이유로 최근 매크로블록 단위의 H.264/AVC 병렬 디코더에 대한 다양한 연구가 이루어지고 있다.[9~12] 매크로 블록 단위 데이터 레벨 병렬화 방법은 동시에 처리 할 수 있는 매크로 블록을 각 스레드에 할당하여 병렬화 한다. 그런데 H.264/AVC에서는 그림 4와 같이 매크로 블록 사이의 의존성이 발생 한다. 예를 들어 그림 4에서 “Current MB”의 IP 처리를 위해서 1,2,3,4 매크로 블록이 먼저 IP 처리 되어야 한다.

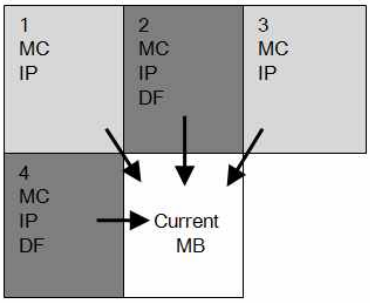


그림 4. 매크로 블록의 데이터 의존성
Fig. 4. Spatial data dependencies for a macroblock.

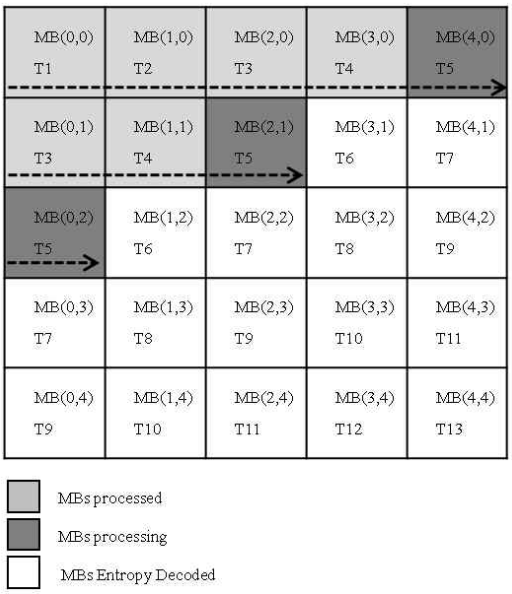


그림 5. 데이터 레벨 병렬화 방법
Fig. 5. Example of data-level parallelization.

2Dwave 병렬화 방법은 대표적인 데이터 레벨 병렬화 방법 중 하나이다. 2Dwave 병렬화 방법은 그림 5에서 각 화살표에 스레드를 할당하여 화살표를 따라서 매크로 블록을 처리한다. 즉 화살표 방향으로 수평적으로 스레드를 할당하여 병렬화 한다. 예를 들어 그림 5에서 MB(4,0), MB(2,1), MB(0,2)는 데이터 의존성을 지키면서 5번째 시간에서 동시에 처리된다. 그리고 이러한 2Dwave 병렬화 방법을 확장하여 3Dwave 방법[13]이 제안 되었다.

그림 6에서와 같이 데이터 레벨 병렬화를 진행하기 위해서는 ED를 선행 처리 한다. 그리고 ED를 처리 한 후 MC+IT/IQ IP+IT/IQ를 병렬화 처리하고 DF의 병렬화가 이루어진다. ED 과정을 선행 처리 하는 이유는

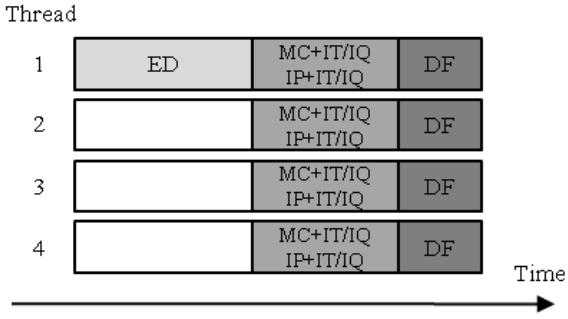


그림 6. 데이터 레벨 병렬화에서 스레드 처리
Fig. 6. Thread allocation in data-level parallelization.

ED 과정은 병렬화가 되지 않기 때문이다. 그리고 병렬화 진행하기 위하여 ED를 선행 처리하면 매크로 블록들이 생성되고 이를 메모리에 저장하게 된다. FHD (1920X1080)의 영상의 경우 프레임 당 8160개의 매크로 블록이 생성되고 이를 메모리에 저장한다. 그러므로 고해상도의 영상의 경우 많은 메모리가 필요하고 이러한 메모리 사용량의 증가는 병렬화 하는데 있어 큰 제한 요소가 된다.

우리는 다양한 H.264/AVC 디코더 병렬화 방법을 살펴보았다. 하지만 병렬화를 통한 성능 향상에는 많은 오버헤드가 있음을 알 수 있다. 우리는 이러한 병렬화 과정에서 발생하는 오버헤드를 해결하기 위하여 새로운 H.264/AVC 디코딩 병렬화 방법을 제안한다. 우리가 제안하는 H.264/AVC 디코딩 병렬화 방법은 Multi-Threaded Parallelization(MTP) 방법이다. 프레임을 매크로 블록 묶음으로 나누어 병렬화하고 독립적으로 스레드를 생성하여 처리하였다. 우리는 이러한 병렬화 방법을 사용하여 H.264/AVC 디코더 병렬화 과정에서 발생하는 스레드 사이의 동기화 오버헤드를 줄일 수 있다. 그리고 메모리를 재사용하는 병렬화 방법을 적용하여 메모리 증가로 인한 오버헤드를 줄었다. 우리는 MTP 방법을 적용한 결과, 병렬화 방법 적용 전의 H.264/AVC 디코더에 비해 최대 53%의 성능이 향상됨을 확인할 수 있었다. 그리고 2Dwave 병렬화 방법과 메모리 사용량을 비교하여 HD 영상에서 65%, FHD 영상에서 81%의 메모리 사용량을 각각 줄 일 수 있었다.

Ⅲ. Multi-Threaded Parallelization

본 논문은 고해상도 영상에서 H.264/AVC 디코더를 빠르게 처리하기 위한 병렬화 방법 제안 한다. 우리는 새로운 H.264/AVC 디코더 병렬화 방법인 Multi-Threaded Parallelization(MTP) 방법을 제안한다. MTP 방법은 프레임을 분할하여 처리 하는 과정과 스레드를 처리하는 과정, 마지막으로 메모리를 재사용하기 위한 과정으로 이루어진다.

MTP 방법은 태스크 레벨 병렬화 방법인 파이프라인 방식 병렬화 방법과 유사하다. 파이프라인 병렬화 방법은 H.264/AVC 디코더를 기능별로 나누고 스레드를 할당하여 병렬화를 처리 한다. H.264/AVC 디코더는 기능적으로 앞서 언급한 바와 같이 ED, IQ/IT, IP, MC, 그리고 DF로 이루어진다. 그리고 매크로 블록의 타입에 따라서 MC 또는 IP가 처리 된다. 일반적으로 IQ/IT는 MC, IP와 동시에 처리 된다. MC, IP에서 IQ/IT를 동시에 처리한 이유는 IQ/IT부분의 처리 속도가 빠르고 H.264/AVC 디코더에서 IQ/IT를 처리하는 비율 낮기 때문이다. 그리고 H.264/AVC 참조 소프트웨어인 JM과 오픈 소스 프로젝트로 개발 중인 ffmpeg H.264/AVC 디코더에서도 IQ/IT를 MC, IP와 동시에 처리 한다. 이러한 이유로 우리는 ED, MC+IQ/IT IP+IQ/IT, DF 부분으로 나누었다. 우리는 나누어진 기능을 스레드에 할당하여 파이프라인 병렬화를 진행한다. 파이프라인 병렬화 방법은 매크로 블록 단위로 파이프라인을 처리하고, 파이프라인 단계마다 매크로 블록 단위로 스레드의 동기화를 확인 한다. 그러나 이러한 스레드의 동기화에서 발생하는 오버헤드는 병렬화를 통한 성능의 걸림돌이 된다. 그리고 해상도가 높아지면 프레임 당 처리하는 매크로블록의 개수가 증가하고, 따라서 스레드의 동기화 횟수도 늘어난다. 이러한 스레드의 동기화로 인한 문제를 해결하기 위하여 프레임을 분할하여 병렬화 하는 방법을 적용 하였다.

1. 매크로 블록 묶음 단위 파이프라인

본 논문에서 제안하는 MTP 방법은 프레임을 매크로 블록 묶음으로 나누어 파이프라인으로 병렬화 처리한다. 그림 7은 프레임 분할을 간단하게 보여준다. 이러한 병렬화 방법은 한 사이클 동안에 여러 명령어를 실행시키는 슈퍼 스칼라, VLIW 방법과 유사하다[14~15]. 파이프라인 단계에서 매크로 블록 묶음 단위로 병렬화하면 매크로 블록 단위의 파이프라인 병렬화에 비교하여 스레드 동기화 횟수를 줄여 오버헤드를 줄 일 수 있다. 우리는 이러한 방법을 사용하여 오버헤드를 줄일

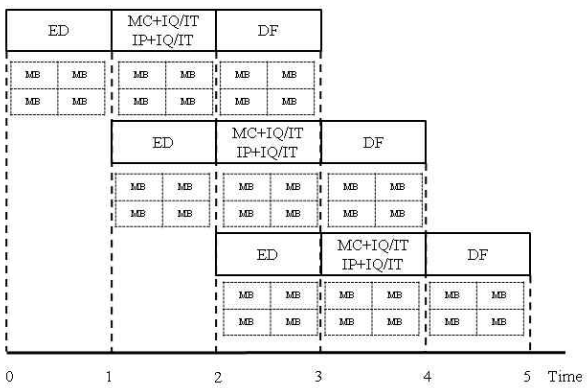


그림 7. 매크로블록 묶음 단위 파이프라인
Fig. 7. Pipelined execution of partitioned frames.

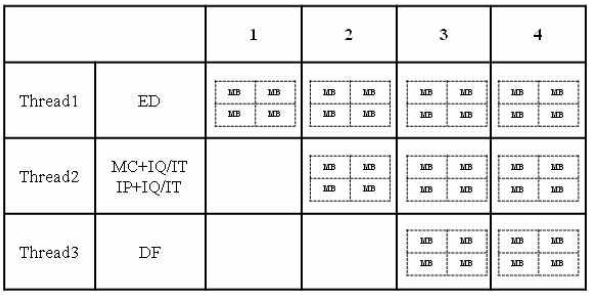


그림 8. 매크로블록 묶음 단위 파이프
Fig. 8. Thread assignment for pipelined execution.

수 있었지만 여전히 파이프라인 단계 사이의 동기화가 존재한다.

파이프라인 병렬화 방법을 자세히 살펴보면 그림 8과 같이 ED에 스레드 1을 할당하고, MC+IQ/IT IP+IQ/IT에 스레드 2를 할당하고 DF를 스레드 3을 할당 한다. 이러한 방법으로 스레드를 할당하여 처리 하면 파이프라인 단계 사이마다 스레드 간의 동기를 확인해야 한다. 그림 8에서 4번째 파이프라인 단계를 진행하기 위해서는 3번째 파이프라인 단계의 각 스레드 처리 완료를 확인해야 한다. 그러므로 파이프라인 단계 사이에서 스레드간의 동기화 문제가 여전히 존재한다.

2. 스레드 생성 및 처리

우리는 앞에서 언급한 문제점을 해결하기 위하여 파이프라인 단계 마다 독립적으로 스레드를 생성하였다. 그리고 파이프라인 단계에서 생성된 스레드는 매크로블록 묶음을 처리하고 종료 한다. 그림 9는 MTP 방법에서 스레드 처리를 보여 준다. 그림 9의 3번째 파이프라인 단계에서는 thread3-1, thread3-2, thread3-3 을 생성하고 생성된 스레드는 매크로블록 묶음을 처리하고 종료 한다. 동일하게 4번째 파이프라인 단계에서 thread4-1, thread4-2, thread4-3을 생성하고, 생성된 스레드는 매크로블록 묶음을 처리하고 종료한다.

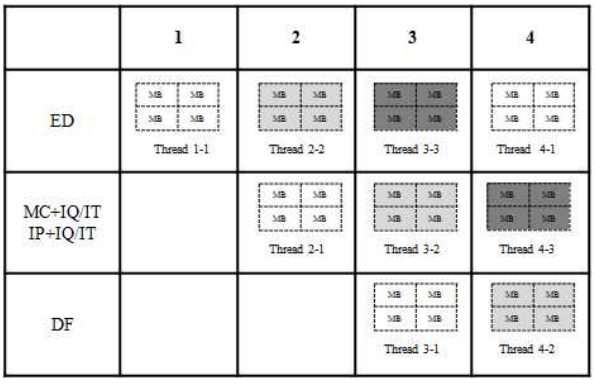


그림 9. MTP 방법에서 스레드 처리
Fig. 9. Thread assignment in MTP.

파이프라인 단계에서 독립적으로 스레드가 생성하고 종료하기 때문에 파이프라인 단계 사이의 스레드 동기화를 줄 일 수 있다. 그러므로 우리는 H.264/AVC 디코더를 병렬화 함에 따라 발생하는 동기화의 오버헤드를 대부분 해결 할 수 있었다. 그런데 우리가 제안하는 MTP 방법은 파이프라인 단계 마다 스레드를 생성하기 때문에 스레드 생성으로 인한 오버 헤드가 발생한다. 스레드 생성으로 인한 오버헤드를 고려하기 위해서는 프레임의 분할하는 크기가 중요하다. 프레임을 작게 분할하면 매크로 블록 묶음의 수가 늘어나고 파이프라인 단계가 많아져서 생성하는 스레드의 수가 증가한다. 반대로 프레임을 크게 분할하면 매크로 블록의 묶음의 수가 적어지고 파이프라인 단계가 적어져서 생성 하는 스레드 줄어든다. 이처럼 매크로 블록 묶음의 크기는 따라서 병렬화의 성능에 많은 영향을 준다. 그래서 우리는 실험을 통하여 영상의 해상도 별로 최적의 결과를 내는 프레임을 분할 단위를 선택하였다.

표 1은 다양하게 프레임을 분할하여 MTP 방법의 성능을 평가한 결과를 보여 준다. 프레임을 분할하는 크기에 따라 묶음 당 매크로 블록 개수(the number of

표 1. 프레임 분할 크기에 따른 성능 변화
Table 1. Performance evaluation for various partition sizes.

	FHD, 1920X1088, 8160 MB (frame/μs)			
N_{MB}/G x N_{ps}	rush_hour	blue_sky	Pedestrian_area	sunflower
240 X 34	10524	10025	11947	9129
480 X 17	9236	8477	10875	8822
960 X 8	11443	11965	12891	9979

	HD, 1280X720, 3600 MB, frame/μs			
N_{MB}/G x N_{ps}	park_run	mobcal	Stockholm	shields
240 X 15	7665	6016	5134	5409
400 X 9	6994	5458	4802	4950
600 X 6	7915	6272	5148	5645

macroblocks per group, N_{MB}/G)가 달라진다. 그리고 묶음 당 매크로 블록 개수에 따라서 파이프라인 구간인 병렬화 구간(the number of pipeline steps, N_{ps})이 결정된다.

FHD, 1920X1080 영상의 경우 매크로 블록 묶음과 병렬화 구간을 240X30, 480X17, 960X8 로 설정 하여 실험 하였다. 표 1에서 FHD, 1920X1080 영상에서는 480X17로 설정 하였을 때 성능이 최적으로 나타났다. 그리고 HD, 1280X720의 영상의 경우 매크로 블록 묶음과 병렬화 구간을 240X15, 400X9, 600X6로 설정 하여 실험 하였다. 표 1에서 HD,1280X720 영상에서는 400X9로 설정 하였을 때 성능이 최적으로 나타났다. 자세한 실험 환경과 실험 결과는 IV장에 서술 하였다.

3. 메모리 재활용

고해상도의 영상으로 갈수록 해상도가 높아지기 때문에 프레임 당 매크로 블록의 개수가 급격하게 높아진다. H.264/AVC 디코더의 병렬화 과정에서는 ED를 처리 후 매크로 블록이 생성되며 이를 메모리에 저장해야 한다. 그러므로 해상도가 높아질수록 프레임 당 매크로 블록이 증가하고 매크로 블록을 저장하기 위하여 많은 메모리가 필요하다. 이러한 메모리 증가로 인하여 지연시간이 발생한다. 우리는 메모리 증가로 발생하는 문제를 해결하기 위하여 메모리를 재사용하는 방법을 사용하였다. 프레임을 분할한 매크로 블록 묶음을 저장하는데 필요한 크기의 메모리의 블록을 MMPG(Memory of Macroblocks Per Group) 라고 정의 한다. 그리고 파이프라인 단계에서 독립된 스레드는 MMPG를 이용하여 병렬화 한다. 그림 9에서 첫 번째 단계에서 thread1-1은 ED처리 후 생성된 매크로 블록을 MMPG 1에 저장한다. 그리고 두 번째 단계에서 thread2-1은 MC+IQ/IT IP+IQ/IT에서 MMPG 1을 사용하고 3번째 단계에서 thread3-1은 DF에서 MMPG 1을 사용하여 H.264/AVC 디코딩 한다. 세 번째 단계 thread3-3에서 MMPG 1을 모두 사용하였기 때문에 네 번째 단계 thread4-1에서 MMPG 1을 다시 사용 할 수 있다. 이러한 방법으로 메모리를 재사용 할 수 있다.

표 2는 병렬화 방법 적용 전 H.264/AVC 디코더와 2Dwave 병렬화 방법, MTP 방법에서 메모리 사용량을 비교 하였다. 메모리 사용량은 FFmpeg H.264/AVC 디코더의 H264Context 데이터 구조의 크기를 측정하였다. H.264/AVC 디코딩하기 위하여 H264Context 데이터 구조는 영상 정보를 가지는 Sequence Parameter Set (SPS)와 프레임의 정보를 가지는 Picture Parameter Set (PPS)와 매크로 블록의 정보를 가지고 있다. 그리고 Entropy Decoding(ED)에서 생성되는 매크로 블록을 저장한다. 병렬화 방법 적용 전 H264Context 데이터 구조에서 하나의 매크로 블록 크기인 1,888 byte 크기의 메모리를 사용하고 전체 H264Context 데이터구조는 174,176 byte 크기의 메모리를 사용한다.

표 2. 메모리 사용량 비교
Table 2. Comparison on memory requirements.

	HD, 1280X720, 3600 MB, (byte)		
	Basic	2D wave	MTP
used MB	1	3,600	400x3
memory of MB	1,888x1=	1,888x3,600=	1,888x400x3=
	1,888	6,796,800	2,265,600
memory for decoding	174,176	6,970,976	2,439,776

	FHD, 1920X1088, 8160 MB, (byte)		
	Basic	2D wave	MTP
used MB	1	8,160	480x3
memory of MB	1,888x1=	1,888x8,160=	1888x480x3=
	1,888	15,406,080	2,718,720
memory for decoding	174,176	15,580,256	2,892,896

그런데 데이터 레벨 병렬화 방법인 2Dwave 병렬화는 ED가 선행 처리 되고 병렬화가 진행 된다. 그래서 2Dwave 방법으로 병렬화 하기 위해서 프레임 당 매크로 블록 개수만큼의 메모리가 필요하다. 2Dwave 병렬화 방법에서 HD, 1280X720 영상에서는 6,970,976 byte의 메모리가 필요하고 FHD, 1920X1088 의 영상에서는 15,580,256 byte 메모리가 필요하다. 우리가 제안 하는 MTP 방법은 MMPG를 이용하여 메모리를 재활용하기 때문에 메모리 사용량을 줄일 수 있었다. MTP 방법은 3개의 MMPG를 사용하고 HD 영상에서는 400개의 매크로블록 단위로 매크로 블록 묶음 사용하므로 2,439,776 byte 크기의 메모리가 필요하다. 그리고 FHD 영상에서는 480개의 매크로블록 단위로 매크로 블록 묶음 사용하므로 2,892,896 byte 크기의 메모리가 필요하다. 2D wave 병렬화 방법과 비교 하면 MTP 방법은 HD영상에서 65%, FHD 영상에서 81%의 메모리 사용량을 줄 일 수 있었다.

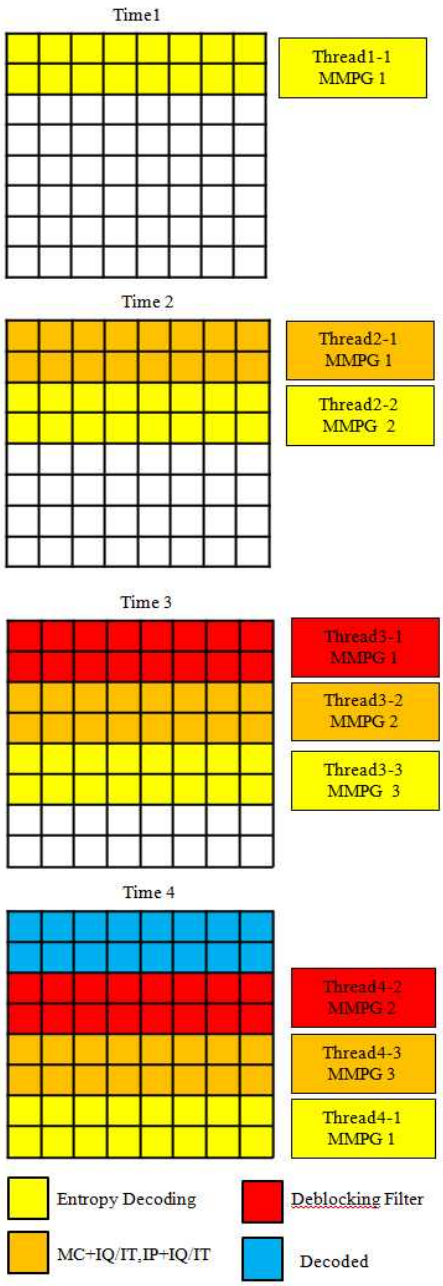


그림 10. MTP 방법에서 메모리 재활용
Fig. 10. Utilization of memory in MTP.

그림 10은 전체적인 MTP 방법을 나타내고 있다. MTP 방법은 그림 10의 time 1 에서 thread1-1은 ED를 수행하여 MMPG 1에 매크로 블록 생성한다. time 2에서 thread2-1은 MMPG 1의 매크로 블록을 이용하여 MC+IQ/IT, IP+IQ/IT 처리한다. 그리고 thread2-2에서는 ED를 수행하여 MMPG 2에서 매크로 블록을 생성한다. time 3에서 thread3-1은 MMPG 1에 저장된 매크로 블록을 사용하여 DF를 수행 한다. 그리고 thread3-2에서는 MMPG 2에 저장된 매크로 블록을 이용하여 MC+ IQ/IT, IP+ IQ/IT 처리한다. 그리고 thread3-3에서는 ED를 수행하여 MMPG 3에 매크로 블록을 생성한다. time 3에서 thread 3-1이 끝나면 디코딩이 완료된 영상을 얻을 수 있다. 그리고 디코딩이 완료되었기 때문에 thread3-1에서 사용된 MMPG 1은 time 4에서 thread 4-1에서 다시 사용된다. 이러한 과정으로 MTP 방법을 수행하여 프레임이 끝날 때까지 병렬화 한다.

Ⅳ. 실험 환경 및 결과

1. 실험 환경

본 논문에서는 오픈 소스 프로젝트로 개발 중인 FFmpeg H.264/AVC 디코더를 사용하여 병렬화 방법을 적용 하여 실험 하였다. 현재 많은 시스템에서 FFmpeg H.264/AVC 디코더를 사용하고 있다. 그리고 FFmpeg H.264/AVC 디코더는 FFplay를 사용하여 영상 재생이 가능하다. 실험을 위해 사용한 영상은 JM-v16을 사용하여 인코딩 하였다. 인코딩 환경은 JM-v16 에서 제공하는 H.264/AVC baseline profile를 기반으로 하였다. 그리고 운영체제는 리눅스 ubuntu 9.10, 커널 버전 2.6.31에서 개발 하였고 인텔 쿼드 코어 i5 프로세서를 사용 하였다. 컴파일러는 gcc v4.4.1을 이용하였고 병렬화 방법은 OpenMP[16]를 사용하였다.

OpenMP는 공유메모리(Shared-memory)구조에서 다중 스레드(multi-thread) 병렬 프로그램을 작성하기 위한 응용 프로그램 인터페이스(API)이다. MPI가 분산 메모리 병렬처리의 표준인 것처럼 OpenMP API는 공유 메모리 병렬 컴퓨팅의 사실상 표준이며 현재 많은 컴파일러에서 OpenMP표준을 지원하고 있다. 사용자는 병렬화가 필요한 부분에 적절한 OpenMP 지시어를 삽입하고 컴파일러는 삽입된 지시어를 참고하여 다중 스레드 코드를 생성한다. 그리므로 OpenMP를 사용하면 쉽게 병렬 프로그래밍 할 수 있다.

4-2. 실험 결과

우리는 인텔 쿼드 코어 기반의 멀티 코어 시스템에서 병렬화 방법 적용 전 H.264/AVC 디코더와 병렬화 방법을 적용한 H.264/AVC 디코더를 비교 하였다. 표 3과 4는 H.264/AVC 디코더 병렬화 방법을 적용 하였을 때 프레임을 처리하는 최소 시간을 나타낸다.

표 3, 4에서 적용한 병렬화 방법은 다음과 같다. 태스크 레벨 병렬화 방법인 파이프라인 병렬화 방법[8]을 실

표 3. FHD 영상에서 H.264/AVC 디코딩 병렬화를 적용 후 프레임 당 최소 시간

Table 3. Results of minimum execution times per frame of parallelization of H.264/AVC decoding in FHD.

FHD 1920X1088	Basic (μs)	Pipeline[8] (μs)	2Dwave[5] (μs)	MTP (μs)
rush_hour	14746	11873	9998	6881
blue_sky	13694	11018	9558	6702
pedestrain _area	13498	12126	9604	8412
sunflower	11034	9481	8507	6680

FHD 1920X1088	Pipeline[8](%)	2Dwave[5](%)	MTP(%)
rush_hour	19%	32%	53%
blue_sky	20%	30%	51%
pedestrain_area	10%	29%	38%
sunflower	14%	23%	39%

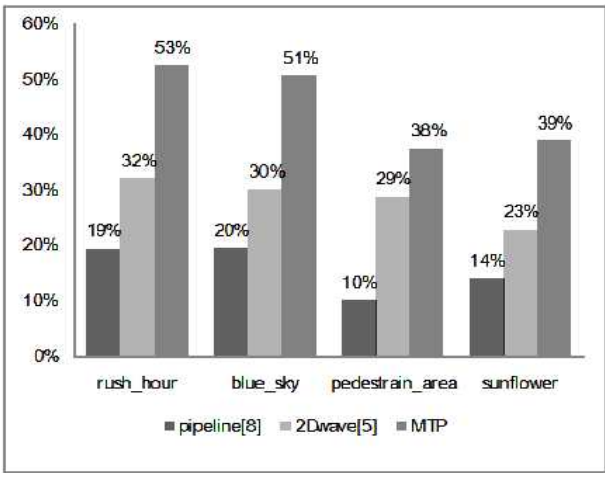


그림 11. FHD 영상에서 H.264/AVC 디코딩 병렬화를 적용 후 프레임 당 최소 시간 그래프

Fig. 11. Minimum execution times per frame of parallelization of H.264/AVC decoding in graphic form in FHD.

험 하였다. 그리고 데이터 레벨 병렬화 방법인 2Dwave 병렬화 방법[8]을 실험 하였다. 마지막으로 본 논문에서 제안한 MTP 방법을 실험하였다. 표 3, 4에서 파이프라인 병렬화 방법은 12-32%, 2Dwave 병렬화 방법은 23-32% 성능 향상을 보였다. 우리가 제안한 MTP 방법은 38-53% 성능 향상을 보였다. 그림 11, 12는 표 3, 4의 결과를 그래프로 나타내었다. 그림 11, 12에서와 같이 본 논문에서 제안하는 MTP 방법이 다른 병렬화 방법에 비교하여 더 효과적임을 알 수 있다.

표 4. HD 영상에서 H.264/AVC 디코딩 병렬화를 적용 후 프레임 당 최소 시간

Table 4. Results of minimum execution times per frame of parallelization of H.264/AVC decoding in HD.

HD 1280X720	Basic (μs)	Pipeline[8] (μs)	2Dwave[5] (μs)	MTP (μs)
park_run	10687	7813	7441	5551
mobcal	3882	2655	2678	2160
stockholm	6640	4996	4563	3701
shields	5707	5046	4160	3515

HD 1280X720	Pipeline[8](%)	2Dwave[5](%)	MTP(%)
park_run	27%	30%	48%
mobcal	32%	31%	44%
stockholm	25%	31%	44%
shields	12%	27%	38%

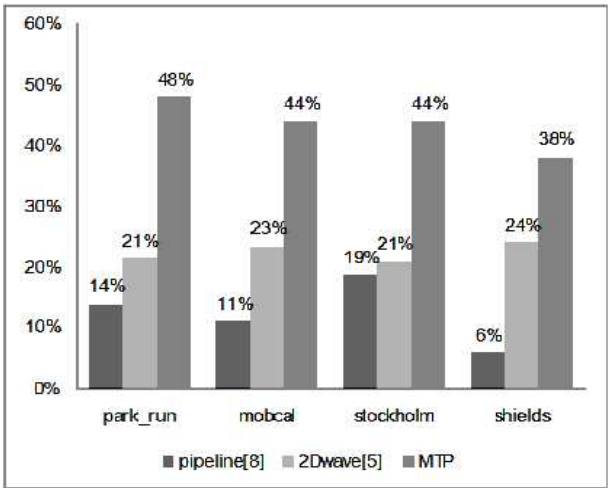


그림 12. FHD 영상에서 H.264/AVC 디코딩 병렬화를 적용 후 프레임 당 최소 시간 그래프

Fig. 12. Minimum execution times per frame of parallelization of H.264/AVC decoding in graphic form in HD.

V. 결 론

본 논문에서 고해상도에서 H.264/AVC 디코더를 빠르게 처리하기 위하여 새로운 H.264/AVC 디코더 병렬화 방법인 MTP 방법을 제안 하였다. MTP 방법은 기존의 H.264/AVC 디코더를 병렬화 과정에서 발생하는 문제점을 해결하여 효과적으로 H.264/AVC 디코더를 병렬화 하였다. MTP 방법은 프레임을 매크로 블록 묶음으로 나누고, 파이프 단계 별로 독립적으로 스레드를

생성, 종료하여 성능을 향상 하였다. 또한 MTP 방법에서는 메모리를 재사용 하여 메모리를 효과적으로 사용하였다. 우리는 MTP 방법을 인텔 쿼드 코어 시스템에서 실험 하였고 다른 병렬화 방법과 성능을 비교하였다. 향 후 MTP 방법을 확장하여 다양한 멀티 코어 시스템에서 적용하여 하려고 한다.

참 고 문 헌

[1] ITU-T Recommendation H.264, SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS Infrastructure of audiovisual services- Coding of moving video, May 2003.

[2] ISO, Information Technology-Coding of Audio-Visual Objects, Part10－Advanced Video Coding, ISO/IEC 14496-10.

[3] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra, Senior Member, “Overview of the H.264/AVC Video Coding Standard”, IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 560-576, July 2003

[4] Michael Horowitz, Anthony Joch, Faouzi Kossentini, and Antti Hallapuro, “H.264/AVC Baseline Profile Decoder Complexity Analysis,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 704-716 July 2003.

[5] E. van der Tol, E. Jaspers, and R.Gelderblom, “Mapping of H.264 decoding on a multiprocessor architecture,” Image and Video Communications and Processing, pp.707-718, May, 2003.

[6] A. Rodriguez, A. Gonzalez, and M. P. Malumbres, “Hierarchical parallelization of an h.264/avc video encoder,” in Proc. Int. Symp. on Parallel Computing in Electrical Engineering, 2006, pp. 363 - 368.

[7] M. Roitzsch, “Slice-Balancing H.264 Video Encoding for Improved Scalability of Multicore Decoding,” in Work-in-Progress Proceedings of the 27th IEEE, 2006

[8] Klaus Schömann, Markus Fauster, Oliver Lampl, Laszlo Böszörmenyi, “An Evaluation of Parallelization Concepts for Baseline-Prole Compliant H.264/AVC Decoders,” in Lecture Notes in Computer Science. Euro-Par 2007 Parallel Processing, August 2007.

[9] J. Chong, N. R. Satish, B. Catanzaro, K. Ravindran, and K.Keutzer,“Effcient parallelization of h.264 decoding with macro block level scheduling,” in 2007 IEEE International Conference on Multimedia and Expo, July 2007.

[10] J. Hoogerbrugge and A. Terechko, “A Multithreaded Multicore System for Embedded Media Processing,” Transactions on High-Performance Embedded Architectures and Compilers, vol. 3, no. 2, pp.168～187, June 2008.

[11] Kosuke Nishihara, Atsushi Hatabu, Tatsuji Moriyoshi, “Parallelization of H.264 video decoder for embedded multicore processor,” In Proceedings of ICME'2008. pp.329～332

[12] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hoogerbrugge, M. Alvarez, and A. Rammirez, “Parallel H.264 Decoding on an Embedded Multicore Processor,” in Proceedings of the 4th International Conference on High Performance and Embedded Architectures and Compilers -HIPEAC, Jan 2009.

[13] Subbarao Palacharla, Norman P. Jouppi and James E.Smith. “Complexity-E ective Superscalar Processors,” In 24th International Symposium on Computer Architecture, pp. 206-218, June 1997.

[14] M.S.Lam, “Software Pipelining: An Effective Scheduling Technique for VLIW Machines,” in Proc. of the SIGPLAN'88 Conference on PLDI, pages 318-328, Atlanta, GA, June 1988.

[15] Chunhua Liao, Zhenying Liu, Lei Huang, and Barbara Chapman. “Evaluating OpenMP on Chip MultiThreading Platforms,” In First international workshop on OpenMP, Eugene, Oregon USA, June 2005.

[16] 조한욱, 조송현, 송용호, “멀티코어 프로세서에서의 H.264/AVC 디코더를 위한 데이터 레벨 병렬화 성능 예측 및 분석,” 전자공학회논문지, 제46권, 제8호, 102～116쪽, 2009년 8월.

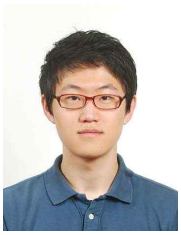
[17] 심동규, 남정학, “고속 비디오 처리를 위한 병렬화 기술,” 전자공학회논문지, 제36권 제4호, (통권 제299호), 83～90쪽, 2009년 4월.

저 자 소 개

김 원 진(학생회원)

2002년 한양대학교 주전공 기계공학과, 부전공 전기전자컴퓨터공학과.
2004년 세나소프트 기술연구소 연구원.
2006년 (주)엠게임 개발팀 연구원.
2008년 한양대학교 전자컴퓨터통신공학과 석사.
2010년 현재 한양대학교 전자컴퓨터통신공학과 박사과정.

<주관심분야 : H.264/AVC, 멀티코어,임베디드 시스템, VLSI 및 SoC 설계>



조 걸(학생회원)

2009년 한양대학교 미디어통신공학과 졸업.
2010년 현재 한양대학교 전자컴퓨터통신공학과 석박사통합과정.

<주관심분야 : 임베디드 하드웨어, 저전력 기법, 멀티코어>

정 기 석(정회원)-교신저자

1989년 서울대학교 컴퓨터공학과 학사.
1998년 Univ. of Illinois at Urbana-Champaign, Computer Science 박사.
1998년 Univ. of Illinois at Urbana-Champaign, 강의 전담 교수.
2000년 Synopsys, Inc. Sr. R&D Engineer
2001년 Intel Corp. Staff Engineer.
2001년 홍익대학교 컴퓨터 공학과 조교수.
2004년 한양대학교 정보통신대학 조교수.
2010년 현재 한양대학교 융합전자공학부 부교수.

<주관심분야 : 임베디드 시스템, VLSI 및 SoC 설계>