실시간 HEVC 인코더 구현을 위한 병렬화 기법에 관한 연구

*안용조, *황태진, **이동규, **김상민, **오승준, *심동규
*광운대학교 컴퓨터공학과
**광운대학교 전자공학과
{madein1st, legendary6}@kw.ac.kr, {dongkyu, kimsangmin}@media.kw.ac.kr,
{sjoh, dgsim}@kw.ac.kr

Study of parallelization methods for real-time HEVC encoder implementation

*Yongjo Ahn, *Taejin Hwang, **Dongkyu Lee, **Sangmin Kim, **Seoung-Jun Oh, *Dong-gyu Sim

*Dept. of Computer Engineering, Kwangwoon University

**Dept. of Electronic Engineering, Kwangwoon University

요 약

ITU-T VCEG 과 ISO/IEC MPEG 이 공동으로 구성한 JCT-VC (Joint Collaborative Team on Video Coding)이 표준화를 진행 중인 HEVC (High Efficiency Video Coding)은 H.264/AVC 대비 약 2 배의 압축효율을 갖는다. 하지만, 계층적 구조를 갖는 가변크기 블록의 사용과 재귀적 부호화 구조에 따른 인코더의 복잡도 증가는 개선해야 할 문제점으로 지적되고 있다. 본 논문에서는 현재 표준화가 진행 중인 HEVC 인코더의 실시간 구현을 위한 SIMD 명령어를 이용한 data-level 병렬화 기법, CPU 및 GPU 를 이용한 multi-threading 기법과 같은 다양한 병렬화 기법을 소개한다. 또한, 이러한 병렬화 기법들을 HEVC 인코더에 적용하기 위해 적합한 연산 및 기능 모듈에 대하여 소개한다. 본 연구를 통하여 HM (HEVC reference model)에 적용한 결과 832×480 영상의 경우 20-30fps 의 부호화 속도를 나타냈으며, 1920×1080 영상의 경우 5-10fps 의 부호화 속도를 나타내었다.

1. 서론

최근 Full-HD (Full High Definition) 및 UHD (Ultra High Definition)와 같은 고품질 비디오 서비스에 대한 수요가 증가함에 따라, ISO/IEC MPEG (Moving Picture Experts Group)과 ITU-T VCEG (Video Coding Experts Group)에서는 JCT-VC (Joint Collaborative Team on Video Coding)를 결성하여 H.264/AVC 의 압축효율을 2 배 이상 향상시키는 것을 목표로 새로운 비디오 압축 표준인 HEVC (High Efficiency Video Coding)에 대한 표준화를 진행하고 있다 [1][2]. HEVC 는 종래의 비디오 압축 표준과 마찬가지로 블록 기반의 하이브리드 코딩에 기반하고 있으나, 추가적인 신규 기술들을 적용하여 H.264/AVC 대비 40% 이상 압축 효율을 향상시키는 것으로 알려져 있다 [3][4]. HEVC 는 기존의 비디오 압축 표준인 H.264/AVC, MPEG-2, 4 에서 사용된 매크로 블록의 개념에 CU (Coding Unit), PU (Prediction Unit), TU (Transform Unit)의 세분화된 부호화 단위를 적용하여 압축 효율을 향상시켰다. 현재, 64×64 블록 크기부터, 32×32, 16×16, 8×8 에 이르는 블록 크기의 CU 단위 부호화를 수행한다. 또한, 화면 내 예측 방향성을 증가시키고, 움직임 예측 결정 기법의 향상, 움직임 벡터 병합 및 다양한 인-루프 필터를 적용하여 압축 효율을 향상시켰다.

다양한 부호화 크기, 향상된 신규 기술들의 적용에 따라 높은 부호화 성능 향상이 있으나, 이로 인하여 부호화 모드 결정에 따른 연산 복잡도 또한 크게 증가 하였다. 다양한 부호화 블록 크기와 세분화된 부호화 단위에 따른 모드 결정이 재귀적으로 이루어지고 있다. 64×64 부터 8×8 에 이르는 블록에 대한 최적의 부호화 모드 결정을 위한 움직임 탐색과 율-왜곡 최적화에 사용되는 다양한 cost 연산 횟수의 증가로 인한 부호화 복잡도 증가는 HEVC 의 개선해야 할 문제점으로 지적되고 있다. 본 논문에서는 이러한 높은 복잡도를 갖는 HEVC 인코더의 실시간성 확보를 위하여 SIMD 명령어를 이용한 data-level 병렬화 기법, CPU 및 GPU 를 이용한 Multi-threading 기법과 같은 다양한 병렬화 기법을 소개하고 이를 적용하기 적합한 연산 및 모듈에 대하여 소개한다.

본 논문의 구성은 다음과 같다. 2 절에서는 본 연구에서 HEVC 인코더에 적용한 병렬화 기법에 대하여 간략하게 소개하고, 각 병렬화 기법들을 적용할 수 있는 HEVC 인코더의

연산 및 모듈에 대하여 소개하고 적용 방법에 대하여 소개한다. 이후, 3 절에서는 다양한 크기의 실험 영상을 이용하여 병렬화 기법을 적용한 HEVC 인코더의 속도를 실험을 통해서 확인한다. 마지막으로 4 절에서는 HEVC 인코더의 실시간성 확보를 위한 앞으로의 연구 방향을 살펴보고 본 논문에 대한 결론을 맺는다.

2. HEVC 인코더를 위한 병렬화 기법

본 절에서는 실시간 HEVC 인코더 구현을 위한 다양한 병렬화 기법에 대하여 소개하고, 이를 적용하기 용이한 연산 및 기능 모듈에 대하여 소개한다. 먼저, SIMD 를 이용한 data-level 병렬화에 대한 소개 및 적용에 대하여 소개하고, OpenMP 를 이용한 CPU 상에서의 multi-threading 기법에 대하여 소개한다. 마지막으로 CUDA 를 이용한 GPU 상에서의 multi-threading 기법과 이에 대한 적용 연산 및 기능 모듈에 대하여 살펴본다.

가. SIMD 를 이용한 data-level 병렬화

SIMD 명령어는 서로 다른 data 에 대하여 동일한 연산을 반복적으로 수행하는데 적합한 병렬화 기법이다. HEVC 인코더에서는 율-왜곡 값을 계산하기 위한 cost 함수, 변환 및 역-변환 모듈, 움직임 정밀도를 높이기 위한 화소 보간 필터, 및 루프필터 등 다양한 연산 및 기능 모듈에 SIMD 명령어를 사용한 병렬화를 적용할 수 있다. 전술한 다양한 연산 및 기능 모듈은 SIMD 명령어를 이용한 병렬처리가 유용한 필터연산 혹은 서로 다른 data 에 대한 동일 연산을 반복적으로 수행됨을 알 수 있다. 특히, HEVC 인코더의 경우 quad-tree 구조의 부호화 단위와 재귀적인 mode 결정과정에 따라 cost 함수, 변환 및 역-변환, 보간 필터의 복잡도가 높게 나타난다 [5]. 본 연구에서는 전술한 연산 및 기능 모듈에 대한 병렬화를 위하여 비디오 압축 응용에서 널리 사용되고 있는 Intel SIMD 명령어 중 SSE2 와 SSE3 를 사용하여 구현하였다.

HEVC 인코더에서는 최적의 율-왜곡 값을 계산하기 위하여 SAD, SATD, SSE 의 세 가지 cost 함수를 사용한다. 이러한 cost 함수들은 HEVC 참조 소프트웨어인 HM (HEVC reference model) 인코더 전체 복잡도 중 약 30%의 높은 복잡도 비율을 차지한다 [5]. 본 연구에서는 4×4 에서 64×64 까지의 SAD, SSE 연산과 4×4, 8×8 SATD 연산을 SIMD 명령어를 사용하여 구현하였다. SAD 연산의 경우, PSADBW, PACKUSWB, PADDD 명령어를, SATD 연산의 경우, PUNPCKLQDQ, PUNPCKHWD, PUNPCKLWD, PADDW, PSUBW, PABSW 를 사용하여 구현하였다. SSE 연산의 경우, PSUBW, PUNPCKLBW, PUNPCKHBW, PMADDWD, PADD 를 사용하여 구현하였다.

Cost 함수 이외에 SIMD 명령어를 적용하기 적합한 연산으로 변환 및 역-변환을 꼽을 수 있다. HM 인코더에서 변환 및 역-변환은 partial butterfly 와 행렬 곱의 두 가지 방법으로 구현되어 있다. 두 가지 방법 중 SIMD 명령어를 사용하여 구현하는 경우, 행렬 곱 방법이 효율적이며, 본 연구에서는 PMADDWD, PADDD, PUNPCKLDQ, PUNPCKHDQ, PUNPCKLQDQ, PUNPCKHQDQ, PSRAD 를 사용하여 구현하였다.

이외에도 HEVC 에서 SIMD 명령어를 사용한 병렬화가 적합한 연산으로는 대표적인 필터연산인 보간 필터와 디블록킹 필터를 꼽을 수 있다. 보간 필터의 경우, HM 인코더 중 30%를 차지하며, 단일 연산으로는 가장 높은 복잡도를 차지한다 [5]. 또한, 디블록킹 필터의 경우 HM 인코더에서 낮은 복잡도를 차지하지만, 추후, 인코더에 고속화가 적용될 경우 복잡도 비율이 증가하게 된다. 이와 같은 필터연산의 경우, PMULLW, PADDW, PSUBW, PSRAW, PUNPCKLQDQ, PUNPCKHQDQ 를 사용하여 구현하였다.

나. OpenMP 를 이용한 CPU multi-threading 기법

OpenMP 는 CPU 환경의 다중 프로세싱을 지원하는 API 로 fork-join 모델에 기반하여, 다양한 응용의 병렬처리에 사용되고 있다. 본 연구에서는 CPU multi-threading 을 위하여 OpenMP 를 사용하였다. 본 연구에서 CPU multi-threading 기법을 이용한 data-level 병렬화를 HEVC 인코더에 적용하였다. 비디오 압축 표준을 위한 data-level 병렬화 방법은 크게 coding unit-level, slice-level (혹은, tile-level), frame-level 로 나눌 수 있으며, 병렬화 성능, 부호화 성능, 코어의 확장성 등의 여러 요인들에 대한 고려가 필요하다. Coding unit-level 및 frame-level 병렬화의 경우, 인접한 블록 혹은 프레임에 대한 의존성이 높기 때문에 병렬처리에 따른 부호화 성능 저하를 최소화 할 수 있으나, 높은 속도 향상은 기대하기 어렵다. 반면, slice-level (혹은, tile-level) 병렬화는 병렬화하는 slice (혹은, tile)의 개수에 따라 부호화 성능에 영향을 받는다. 하지만, 본 연구에서는 실시간 부호화에 초점을 맞추었기 때문에, 병렬화에 따른 높은 속도 향상을 얻을 수 있는 slice-level 병렬화 방법을 선택하였다. 또한, 병렬 처리를 위하여 HEVC 에서 지원하는 두 가지 slice 분할 방식, slice 별 최대 byte 수 지정 방식 및 slice 별 coding unit 개수 할당 방식, 중 slice 별 coding unit 개수 할당 방식을 사용하였다.

OpenMP 를 사용하여 slice 를 처리하기 위해 slice 인코더 인스턴스를 slice 개수에 맞춰 생성하고, slice 부호화에 앞서 fork 를 사용하여 slice 별 thread 을 할당한다. 각 thread 는 앞서 생성한 slice 인스턴스 중 자신의 thread ID 에 해당하는 slice 인코더를 동작시킨다. 이때, 각 slice 인코더 인스턴스들은 독립적인 메모리를 사용하여 부호화를 수행한다. 독립적으로 부호화를 마친 thread 는 디블록킹 필터를 수행하기 전, 모든 thread 의 부호화가 완료될 때 join 을 수행한다. 그림 1 은 OpenMP 를 이용한 CPU multi-threading 기법을 사용하여 HEVC 인코더에 slice 병렬화를 수행한 결과이며, slice 개수에 따른 속도 향상을 나타낸다. Slice 개수를 2, 4, 8, 16, 32 로 증가시켰으며, 최대 6 배의 속도 향상 결과를 나타내었다. 그림 1 은 quad-core (hyper-threading 시 최대 8 thread) CPU 환경에서 측정한 결과이며, CPU 에서 지원하는 thread 의 개수가 증가할수록 slice 병렬화에 의한 속도는 더욱 향상 될 것으로 기대된다. 또한, 병렬화의 속도 개선을 위한 알고리듬 [6]을 통하여 추가적인 속도 향상을 얻을 수 있다.

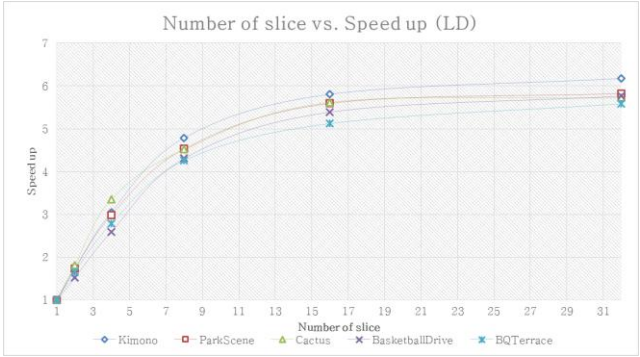

그림 1. HEVC 인코더의 slice 병렬화에 따른 slice

개수별 속도 향상

다. CUDA 를 이용한 GPU multi-threading 기법

CUDA 는 엔비디아에서 개발한 GPU 상에서 병렬처리 알고리듬을 수행하는 플랫폼 및 프로그래밍 언어이다 [7]. CPU multi-threading 기법에 비해 GPU multi-threading 기법은 보다 많은 thread 를 사용할 수 있다는 장점이 존재하지만, 호스트 (CPU)와 디바이스 (GPU) 간의 메모리 전송에 대한 오버헤드가 발생한다. 본 연구에서는 CUDA 를 이용하여 GPU 상에서 움직임 예측을 picture 단위로 병렬처리 하도록 구현하였다. 본 연구에서는 CTU 크기를 32×32 로 고정하고, 움직임 검색 영역은 32 를 사용하였다. CUDA 를 이용한 GPU multi-threading 기법의 움직임 예측을 위하여 현재 부호화하는 picture 와 참조 picture 를 호스트에서 디바이스로 전송하고, thread 블록별로 하나의 CTU 를 처리하도록 구현하였다. 하나의 thread 블록은 512 개의 thread 로 구성되며, 각 thread 별 8×8 블록의 SAD 를 구하여 CPR (Concurrent Parallel Reduction)을 사용하여 8×8 블록부터 32×32 블록까지 각 블록 크기에 해당하는 최소의 SAD 를 계산하였다 [8]. CPR 을 사용하여 계산된 모든 블록 크기별 최소의 SAD 를 갖는 움직임 벡터는 디바이스에서 호스트로 전송되며, CPU 에서는 움직임 예측 수행대신 GPU 에서 전송된 움직임 벡터를 이용하여 부호화를 수행하므로 부호화 속도 향상을 얻을 수 있었다.

3. 실험 결과

본 절에서는 다양한 병렬화 기법을 적용한 HEVC 인코더의 부호화 속도를 측정하고 이에 대하여 분석한다. HEVC 인코더의 부호화 속도 측정을 위하여, HEVC 참조 소프트웨어인 HM (HEVC reference model) 10.0 을 기반으로 C 변환을 수행한 소프트웨어에 SIMD 명령어, OpenMP 기반 CPU multi-threading, CUDA 기반 GPU multi-threading 의 병렬화 기법들을 적용하였다. 본 연구에서 제안하는 병렬화 기법들을 적용한 부호화 속도를 측정하기 위하여 Intel Core™ i7-3960X 3.30GHz CPU, NVIDIA GeForce GTX 660 Ti GPU, 40GB 메모리, Intel C++ 64-bit compiler XE 13.0 을 사용하였으며, Window 8 64-bit OS 환경에서 실험을 진행하였다. 실험 조건은 다음과 같다.

(a) Profile: HEVC Main profile

(b) Encoding structure: Low-delay P

(c) QP value: 22, 27, 32, 37

(d) Test sequence: HEVC common test sequence (ClassB and ClassC)

표 1. 제안하는 병렬화 기법들을 적용한 HEVC 인코더의 부호화 속도 측정 결과

Class	Sequence	Frame	QP	FPS
B	Kimono	240	22	5.74
			27	7.25
			32	8.38
			37	9.40
	ParkScene	240	22	5.51
			27	7.52
			32	8.87
			37	10.03
C	Cactus	500	22	5.19
			27	7.70
			32	9.09
			37	10.09
	BasketballDrive	500	22	4.80
			27	6.71
			32	8.09
			37	9.18
	BQTerrace	600	22	4.14
			27	7.68
			32	9.60
			37	10.62
	BasketballDrill	500	22	14.86
			27	19.07
			32	23.60
			37	28.12
	BQMall	600	22	14.81
			27	19.88
			32	24.91
			37	29.20
	PartyScene	500	22	11.09
			27	16.46
			32	22.03
			37	27.60
	RaceHorses	300	22	10.48
			27	14.60
			32	19.46
			37	24.49

표 1 은 제안하는 병렬화 기법들을 적용한 HEVC 인코더의 부호화 속도를 측정한 결과이다. 연산 및 기능 모듈관점에서는 SIMD 명령어를 사용하여 cost 함수, 변환 및 역-변환, 보간 및 디블록킹 필터를 구현하였으며, CUDA 를 이용한 움직임 예측을 구현하였다. 또한, 부호화 구조 관점에서는 OpenMP 를 이용한 slice 병렬화를 수행하였다. 표 1 에서 나타나는 바와 같이, 1920×1080 의 경우 QP 에 따라 5-10 fps 의 결과를 나타내었으며, 832×480 의 경우 QP 에 따라 10-30 fps 의 부호화 속도를 나타낸다.

4. 결론

본 논문에서는 실시간 HEVC 인코더 구현을 위하여 다양한 병렬화 기법에 대한 소개하고, 이를 적용하기 용이한 연산 및 기능 모듈을 소개하였다. 서로 다른 data 에 대한 동일한 연산을 반복적으로 수행하는 cost 함수, 변환 및 역-변환, 보간 필터, 차분 영상 생성 및 디블록킹 필터에 SIMD 명령어를 사용하여 data-level 병렬화를 수행하였다. 또한, OpenMP 를 이용한 CPU multi-threading 기법을 사용하여 slice 단위 병렬화를 수행하였으며, CUDA 를 이용한 GPU multi-threading 기법을 사용하여 움직임 예측 및 최적의 움직임 벡터 계산에 대한 병렬화를 수행하였다. 전술한 다양한 병렬화 기법을 적용하여 HEVC 인코더의 부호화 속도를 832×480 영상의 경우 10-30fps, 1920×1080 영상의 경우 5-10fps 로 향상시킬 수 있었다. 현재 832×480 영상의 경우

실시간 부호화가 가능하지만, 2K, 4K 영상의 실시간 부호화가 가능한 HEVC 인코더 개발을 위해 추가적인 소프트웨어 최적화 및 병렬화 연구를 진행할 계획이다.

감사의 글

본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천기술개발사업(정보통신)의 일환으로 수행하였음. [10039199, 인지품질 기반 스케일러블 3D 비디오 코덱 핵심 기술 연구]

참고문헌

[1] G. J. Sullivan, J. - R. Ohm, " Recent Developments in Standardization of High Efficiency Video Coding (HEVC)," SPIE Applications of Digital Image Proc. XXXIII, Proc. SPIE, Vol. 7798, pp. 7798-30, Aug. 2010.

[2] JCT-VC, " Report of Subjective Test Results of Responses to the Joint call for Proposals (CfP) on Video Coding Technology for High Efficiency Video Coding (HEVC)," Document JCTVC-A204, Dresden, DE, Apr. 2010.

[3] E. Ohwovoriole, Y. Andreopoulos, " Rate-distortion performance of contemporary video codecs: Comparision of Google/WebM VP8, AVC/H.264, and HEVC TMuC," LENS Symp., London, Sep. 2010.

[4] F. De Simone, L. Goldmann, J. - S. Lee, T. Ebrahimi, " Performance analysis of VP8 image and video compression based on subjective evaluations," SPIE Applications of Digital Image Proc. XXXIV, Aug. 2011.

[5] 안용조, 황태진, 유성은, 한우진, 심동규, " HEVC 부호화기 소프트웨어의 통계적 특성 및 복잡도 분석," 방송공학회 논문지, 제 17 권 6 호, pp. 1091-1105, 2012 년 11 월.

[6] 안용조, 유성은, 심동규, " HEVC 부호화기의 부하균형 기법을 이용한 슬라이스 병렬화," 2012 년도 제 25 회 신호처리합동학술대회, 2012 년 9 월.

[7] NVDIA, " CUDA C Programming Guide," document PG-02829-001_v5.0, pp3-4, Oct. 2012.

[8] D. Lee and S. Oh, " Variable block size motion estimation implementation on compute unified device architecture (CUDA)," IEEE International Conference on Consumer Electronics, pp. 635-636, Jan. 2013.