특집

비디오 코덱 복잡도 추정 기술

최명호 (건국대학교)

I. 서 론

최근 제정된 H.264/AVC 표준은 기존 H.263, MPEG-4 심플 프로파일에 비해 2배 이상의 압축 효율을 제공하나 급격히 높아진 복잡도로 인하여 코덱 구현이 어렵다.

더욱이 Full HD, UD (Ultra High Definition) 등 고해상도/고화질의 영상에 대한 요구가 늘어남으로써 이를 지원하게 될 차세대 비디오 코덱들의 복잡도는 더욱 높아질 전망이다. 이에 반해 비디오 코덱의 기반이 되는 프로세서의 단일 작업 성능은 점차 한계에 도달하고 있어 고성능 고화질 코덱을 구현하기에는 많은 어려움이 있다. 더불어, 일반적으로 재구성된 영상을 기반으로 움직임 추정을 수행하고 압축하는 비디오 코덱은 이전에 처리된 데이터를 재사용하는 구조로써 낮은 병렬성을 갖게 된다. 이러한 낮은 병렬성은 다중 코어 프로세서를 통한 고성능 코덱의 구현이 쉽지 않게 하고 있다. 따라서 구현이 용이하며 고화질/고해상도를 지원하는 코덱을 구현하기 위해서는 저-복잡도의 코덱 또는 높은 병렬성을 갖는 코덱의 개발은 필수적이라 할 수 있다. 이러한 코덱의 개발을 위해, 코덱 알고리즘의 복잡도를 정확히 측정, 평가할 수 있는 방법이 필요하다. 하지만 아직까지 코덱의 복잡도와 그에 대한 측정 방법 및 수단에 대한 기준이 명확히 정의 되지 못한 상황이다. 따라서 코덱의 복잡도를 정확히 파악 할 수 있는 방법에 대한 관심이 집중되고 있다.

본 고에서는 최근 관심이 고조되고 있는 비디오 코덱의 복잡도와 복잡도 측정 방법에 대하여 알아보고자 한다. 본 고의 구성은 제2장에서는 복잡도의 정의와 하드웨어 플랫폼과 복잡도와의 상관관계에 대하여 언급하고 제3장은 다양한 복잡도 측정 방법을 알아본다. 마지막 제4장에서는 결론과 향후 전망에 대하여 살펴본다.

II. 코덱 복잡도 정의

1. 코덱 복잡도 정의의 다양성

코덱 복잡도는 처리시간, 저장 공간, 메모리 사용량, 구현 비용 등 다양한 측면을 갖는다. 코덱 복잡도가 이렇게 다양한 측면을 갖는 이유는 코덱을 사용하는 분야의 목적과 구현 방법에 따른 설계자의 관심분야가 다르기 때문이다. 다양

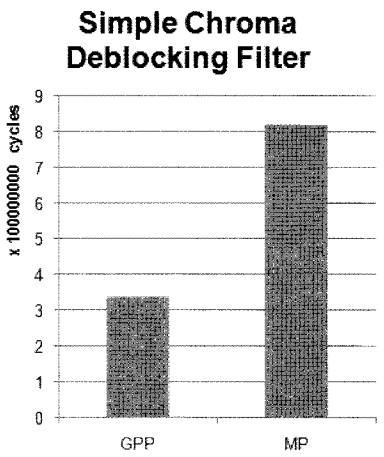
한 복잡도의 측면들 중 본 고에서는 비디오 코덱 복잡도를 코덱 연산 처리 속도로 정의하려 한다. 이러한 복잡도 정의는 직관적이고 간단하여 많은 기존의 연구들에서 사용되어져 왔다.

일반적으로 코덱 연산 처리속도는 단순한 연산량 (예, 더하기 수, 곱셈의 수 등) 계산을 통하여 측정된다. 하지만 연산량 계산을 통한 복잡도 측정 방법은 실행시 고려해야하는 데이터 종속성 및 메모리 접근 시간 등 하드웨어 요소에 따른 지연 시간을 포함하지 않기 때문에 현실적인 실행 시간을 나타낼 수 없다. 따라서 정확한 처리 속도 및 구현 비용을 반영하는 비디오 코덱 복잡도를 측정하기 위해서는 연산 하드웨어 플랫폼을 고려한 실행 기반 시간적 복잡도 측정 방법이 필요하다. 다음 장에서는 이러한 실행 기반 시간적 복잡도 측정 시 하드웨어 플랫폼이 코덱 복잡도에 미치는 영향에 대하여 알아본다.

2. 연산 하드웨어 플랫폼과 복잡도 상관관계

실행 기반 시간적 코덱 복잡도 측정 방법은 코덱 실행시간을 복잡도로 정의하는 것으로써 이는 코덱이 실행되는 하드웨어 플랫폼에 따라 많은 영향을 받게 된다. <그림 1>은 실행 기반 시간적 복잡도 측정 툴인 코덱 시뮬레이터[1]를 이용하여 두 가지 다른 하드웨어 플랫폼에서의 H.264 심플 크로마 디블럭킹 필터(Simple Chroma Deblocking Filter[2]) 알고리즘의 복잡도를 측정한 것이다. 이 경우 프로세서 클럭 속도에 따른 복잡도 왜곡을 피하기 위하여 클럭 사이클 수로 실행시간(복잡도)을 비교하였다.

<그림 1>에서 보는 것과 같이 전체 실행 시간이 두 하드웨어 플랫폼에서 큰 차이가 있음을 알 수 있다. 이는 하드웨어 플랫폼의 다른 구조가 실



Parameter	Micro Architecture (General Processor)
Issue Scheduling	Out-of-Order
fetch/decode/commit width	4/4/4 (4-way Superscalar Pipeline)
IntALU/IntMUL/FpALU/FpMUL/Mem	4/1/1/1/2
I-cache L1	16KB
D-cache L1	16KB
I/D-cache L2	512KB

Parameter	Micro Architecture (Mobile Processor)
Issue Scheduling	In-Order
fetch/decode/commit width	1/1/1 (1-way Simple Pipeline)
IntALU/IntMUL/FpALU/FpMUL/Mem	1/1/1/1/1
I-cache L1	32KB
D-cache L1	32KB
I/D-cache L2	None

〈그림 1〉 코덱 시뮬레이터를 이용한 코덱 알고리즘 복잡도 분석

행시간에 미치는 영향 때문인데 이렇게 영향을 미치는 하드웨어 플랫폼 주요 요소로는 명령어 스케줄러, 연산 파이프라인의 수, 캐시 메모리 구조, 캐시 메모리의 크기 등이 있다.

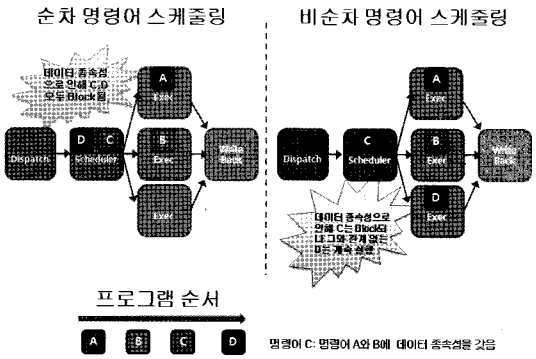
가. 명령어 스케줄러

명령어 스케줄러는 명령어를 연산 유닛에 넘겨주는 하드웨어 요소로써, 연산 유닛이 이용 가능한 경우, 데이터 종속성의 존재 여부를 판단하여, 데이터 종속성이 발생했을 때 모든 명령의 수행을 중지하는 순차 명령어 스케줄러[3]와 데이터 종속성이 있더라도 발생된 데이터 종속성과

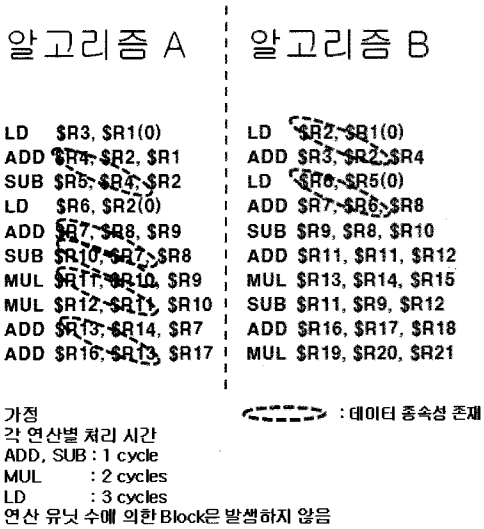
관련이 없는 명령어를 먼저 수행하는 비순차 명령어 스케줄러[3]가 있다.

아래의 <그림 2>는 각 명령어 스케줄러의 처리 방식의 차이점을 보여준다. 명령어 C는 명령어 A와 B에 데이터 종속성을 갖고 있기 때문에 순차/비순차 명령어 스케줄러에서 모두 수행이 되지 못한다. 순차 명령어 스케줄러는 명령어 C에 의해 명령어 D의 수행도 같이 멈추게 되나, 비순차 명령어 스케줄러는 명령어 종속성과 관련 없는 명령어 D를 수행할 수 있다.

<그림 3>는 명령어 스케줄러가 코덱 복잡도에 미치는 영향을 나타낸 예이다. 동일한 기능을 수행 하는 두 가지의 코덱 알고리즘 A, B가 동일한 10개의 명령어로 구성되어 있다. 알고리즘 A의 데이터 종속성은 알고리즘 B의 데이터 종속성보다 많다. 두 알고리즘을 순차 명령어 스케줄러 환경에서 실행했을 경우, 동일한 시간이 소비된다. 하지만 이 두 알고리즘을 비순차 스케줄러 환경에서 수행하면, 데이터 종속성이 많은 알고리즘 A의 수행시간이 더 길게 측정된다. 이는 데이터 종속성으로 인하여 명령어의 수행이 멈춰지는 경우가 많이 발생하기 때문이다. 반면 알고리즘 B의 경우 데이터 종속성이 적고 병렬성이 높



<그림 2> 순차 명령어 스케줄링과 비순차 명령어 스케줄링의 예



<그림 3> 데이터 종속성의 예

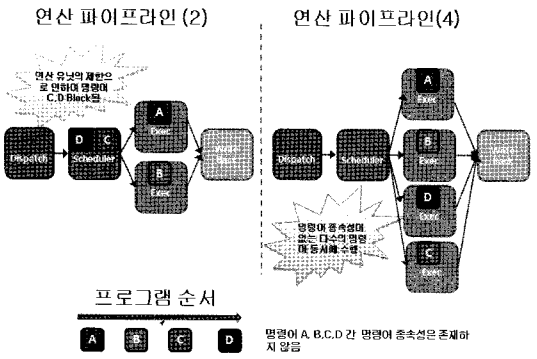
기 때문에 대부분의 명령어 멈추지 않고 지속적으로 수행이 된다.

이와 같이 명령어 스케줄러는 데이터 종속성에 의한 지연 시간을 완화하는 주요 하드웨어 요소로써 코덱 실행 시간에 큰 영향을 미친다.

나. 연산 파이프라인의 수

연산 파이프라인은 직접적으로 연산을 실행하는 부분으로써 연산 파이프라인의 수는 동시에 수행할 수 있는 연산의 수를 나타낸다. 데이터 종속성에 의한 지연이 발생하지 않을 경우, N개의 연산 파이프라인을 갖춘 하드웨어는 1개의 연산 파이프라인을 갖춘 하드웨어 보다 N배 빠른 연산을 수행할 수 있다.

<그림 4>와 같이 만약 데이터 종속성이 존재하지 않는 알고리즘 A를 2개의 연산 파이프라인을 갖춘 하드웨어에서 실행했을 때의 실행시간은 4개의 연산 파이프라인을 갖춘 하드웨어에서 실행했을 때 실행시간의 2배가 된다.



〈그림 4〉 연산 파이프라인의 수에 따른 연산 속도의 차이

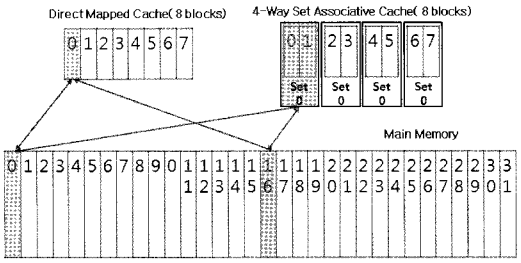
보다 많은 연산 유닛을 갖춘 하드웨어는 다수의 연산을 동시에 처리함으로써 더욱 빠르게 코덱 알고리즘을 수행할 수 있다. 이와 같이 연산 파이프라인의 수는 코덱 실행 시간에 큰 영향을 미친다.

다. 캐시 메모리 구조

캐시 메모리는 메인 메모리와 프로세서 사이에 존재하는 작은 크기의 메모리로 고속, 고비용의 메모리이다. 일반적으로 프로그램은 자주 사용하는 데이터를 지속적으로 사용하거나 또는 현재 접근한 데이터와 가까이 있는 데이터를 사용하는 경향이 높다. 캐시 메모리는 이러한 데이터를 저장하여 빠르게 프로세서에 제공함으로써 프로세서가 데이터에 접근하는 시간을 줄인다. 하지만 이러한 캐시 메모리도 내부구조에 따라 성능에 큰 차이를 보인다.

<그림 5>는 메인 메모리와 캐시 메모리와의 구조를 간략화 시켜 나타낸 것이다. 메인 메모리블록은 32개이고 캐시는 8개의 블록을 갖고 있다. 이 가정 하에서 캐시 메모리의 구조가 코덱의 복잡도에 어떠한 영향을 미치는지 살펴보도록 하겠다.

일반적으로 비디오 코덱은 메크로블록이라는



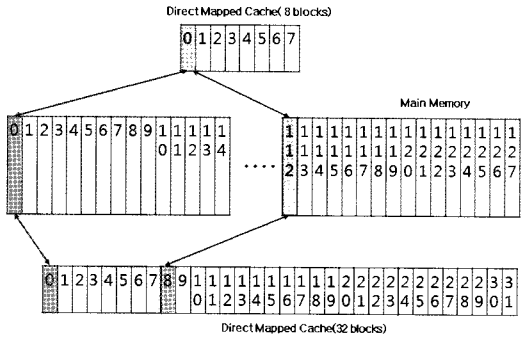
〈그림 5〉 Direct Mapped Cache와 4-way Set Associative Cache

일정 크기의 블록을 기반으로 압축을 수행한다. 최근 제정된 H.264의 경우 16x16크기를 사용하나 현재 가정된 하드웨어의 상황은 설명을 용이하도록 하기 위해 매우 작은 메인 메모리와 캐시를 이용하므로 메크로블록의 크기도 4x4로 가정한다. 재구성된 참조 영상에서 하나의 메크로블록에 접근하기 위해서는 16바이트 단위로 데이터에 접근하게 된다. 따라서 두 개의 메크로블록을 번갈아 접근하는 경우, Direct Mapped 캐시의 경우 캐시 블록 0번의 내용이 계속해서 바뀌며 캐시 미스를 발생시킨다. 반면 4-way set associative 캐시의 경우 초기 캐시 미스를 제외하면 캐시 미스가 발생하지 않는다. 하나의 셋에 2개의 캐시 블록이 존재하기 때문이다. 캐시 미스에 의한 지연 시간이 수백 사이클이라는 점을 감안하면 캐시 구조는 코덱의 실행시간에 큰 영향을 미친다.

라. 캐시 메모리 크기

큰 캐시 메모리는 하나의 캐시 블록 당 부합하는 메인 메모리 블록의 수를 줄임으로써 더욱 낮은 교체율을 갖게 되는 요인이 되며 이로 인하여 캐시의 적중률을 높일 수 있다.

<그림 6>는 캐시 크기에 따른 메인메모리와의



〈그림 6〉 캐시 크기에 따른 메인 메모리와의 연관성

연관성을 나타낸 그림이다. 메인 메모리는 128개의 블록으로 구성되어 있고 Direct Mapped 캐시 구조를 갖는 캐시 2개가 존재한다. 이를 설명하기 위해 "캐시 메모리 구조"에서 언급한 4x4 메크로블록 크기의 가정을 동일하게 이용하도록 하겠다. 4x4 매크로블록의 데이터에 접근하기 위해서는 16바이트 단위로 메모리에 접근하게 된다. 따라서 두 개의 메크로블록을 번갈아 접근하는 경우 8개의 캐시 블록을 갖는 캐시는 0번 블록이 계속 교체되면서 캐시 미스를 발생시키는 반면 32개의 캐시 블록을 갖는 캐시는 초기 캐시 미스를 제외하면 캐시 미스가 발생하지 않는다. 캐시 미스에 의한 지연 시간 수백 사이클이라는 점을 감안하면 캐시의 크기는 코덱의 실행 시간에 큰 영향을 미친다.

이상에서 살펴본 바와 같이 코덱 복잡도(실행 시간)는 실행되는 하드웨어 플랫폼에 따라 다르게 측정되기 때문에 복잡도를 일반화하기 어렵다. 따라서 코덱의 복잡도를 합리적이고 보편화시킬 수 있는 측정 방법 및 수단이 필요하다. 다음 장에서는 지금까지 제안되어져 왔던 복잡도 측정 방법들에는 어떠한 것들이 있는지 알아본다.

III. 다양한 복잡도 측정방법

1. 연산 리소스 및 처리시간 카운팅 방식에 따른 복잡도 측정

정확한 복잡도를 측정하기 위해서는 복잡도에 직접적으로 영향을 미치는 하드웨어에 대한 고려가 필요하다. 연산 리소스 및 처리 시간 카운팅에 의한 복잡도 측정 방법[5]은 비디오 코덱이 동작하는 하드웨어의 연산 유닛을 고려하여, 코덱 알고리즘이 필요로 하는 모든 연산을 처리하는데 소비되는 최소 사이클을 이론적으로 구함으로써 복잡도를 측정한다. 자세한 복잡도 측정 과정 다음과 같다.

STEP 1) 함수에서 사용되는 연산의 수를 세어 함수별 연산수 테이블을 작성한다.

〈표 1〉 함수별 연산수 테이블[5]

	Add8	Add16	Mult8	Mult16	MAC8	Mac16	branch	Shift	Load	Store	AND	OR
4x4 Inverse Transform + Reconstruction	0	96	0	0	0	0	0	32	48	16	0	0
4x4 Null Inv Transform(Reconstruction)	0	0	0	0	0	0	0	0	16	16	0	0
4x4 Luma DC Transform	0	64	0	0	0	0	0	0	16	0	0	0
2x2 Chroma DC Transform	0	16	0	0	0	0	0	0	8	0	0	0
Chroma Interpolation (One Chroma Sample)	0	4	0	4	0	0	0	1	4	0	0	0

〈표 2〉 하드웨어별 연산 유닛 테이블[5]

	Add8	Add16	Add32	Mult8	Mult16	MAC8	MAC16	Branch	Shift	Load	Store	AND	OR
Trimedia	1,3	1,3	1,2,3,4,5	2,3	2,3	2,3	2,3	2,3,4	1,2	4,5	4,5	1,2,3,4,5	1,2,3,4,5
Pentium III	1,2	1,2	1,2	N/A	1	N/A	1	2	2	3	4	1,2	1,2
Ti C64x	1,2	1,2,5,6,7,8	1,2,5,6,7,8	3,4	3,4	N/A	N/A	5,6	3,4,5,6	7,8	7,8	1,2,5,6,7,8	1,2,5,6,7,8
Equator BSP-15	2,4	2,4	1,2,3,4	2,4	2,4	2,4	2,4	1,3	2,4	1,3	1,3	1,2,3,4	1,2,3,4

STEP 2) 비디오 코덱 알고리즘이 동작하는 하드웨어의 연산 유닛 수를 나타내는 테이블을 이용하여, 해당 함수에 포함되어 있는 모든 연산을 수행하는데 필요한 최소 사이클을 계산한다.

STEP 3) 함수의 초당 호출 횟수를 측정하여 함수 호출 빈도 테이블을 작성, STEP 2에서 얻어진 계산값과 함수호출 빈도 테이블의 값을 곱하여 최종 복잡도 측정한다.

이 복잡도 측정 방법은 하드웨어 연산 유닛 에 대한 고려가 포함되어 있으나 그 외 다수의 하드웨어 가정을 포함하고 있다.

- 무한개의 레지스터

〈표 3〉 함수 호출 빈도 테이블[5]

	Mobile, QP = 21
Inverse Quantization	
DC Luma Coefficients	33.3
AC Luma Coefficients	67915.7
DC Chroma Coefficients	3631.5
AC Chroma Coefficients	7443.1
Transform and Reconstruction	
4x4 Luma Transform	28266.6
4x4 Chroma Transform	13899.1
4x4 Luma DC Transform	7.2
2x2 Chroma DC Transform	6972.0

- 캐시 미스에 의한 지연시간 "0"
- 모든 연산의 연산 시간은 "1" 사이클
- 루프 또는 컨트롤 오버헤드는 존재하지 않음

→ 이상적 하드웨어 환경

연산 리소스 및 처리 시간 카운팅에 의한 복잡도 측정 방법은 이와 같이 이상적인 하드웨어 환경을 가정함으로써 데이터 종속성, 캐시 미스, 컨트롤 오버헤드, 연산 유닛의 연산 수행 시간등, 하드웨어에 의한 지연시간을 반영하지 못한다. 2장에서 언급한 바와 같이 각각의 하드웨어 요소는 실제 코덱의 수행시간에 큰 영향을 미친다는 점을 고려했을 때, 이 복잡도 측정 방법은 하드웨어 요소에 의한 지연 시간을 반영하지 못하므로 정확한 코덱의 복잡도 측정 방법이라 할 수 없다.

<표 4>는 펜티엄3에서 인텔 VTune 프로파일링 프로그램을 이용하여 측정한 실제 복잡도와 이론적으로 계산된 복잡도와의 비를 보여주는 표이다. 이 표에서 나타나듯이 실제 측정된 복잡도와 이론적으로 계산된 복잡도는 다양한 크기의 비를 갖고 있다(1.45~6.96배). 이는 이론적 복잡도 측정으로는 코덱 복잡도를 정확히 측정할 수 없음을 나타낸다. 위 결과는 연산 리소스 및 처리시간 카운팅 방식에 따른 복잡도 측정 방법이 이상적 하드웨어 환경(무한개의 레지스터, 캐시 미스에 의한 지연시간 "0" 등)을 가정함으

〈표 4〉 펜티엄3에서 측정된 실제 복잡도와 계산된 복잡도 비[5]

	Container (Qcif, QP=15)	Foreman (Qcif, QP=24)	Silent (Qcif, QP=22)	Paris (Cif,QP =17)	Foreman (Cif,QP= 28)	Mobile (Cif,QP= 21)
Loop Filtering	2.57	2.36	2.71	2.85	2.58	2.84
Reconstr uction	3.90	3.79	3.94	4.57	4.05	4.05
Luma Interpola tion	2.25	1.45	1.87	2.14	1.62	1.72
Chroma Interpola tion	6.38	4.94	6.12	6.96	5.42	5.81

로써 발생한 차이이며, 이 복잡도 측정 방법의 한계를 잘 보여준다. 또한 이 복잡도 측정 방법은 하드웨어에 익숙하지 못한 비디오 전문가들이 사용하기 어렵다는 단점이 있다.

2. 복잡도 요소 정의에 따른 복잡도 측정

복잡도 요소 정의에 따른 복잡도 측정 방법[6]은 연산 복잡도, 데이터 캐시 크기, 메모리 접근 대역폭, 저장 복잡도등 복잡도의 다양한 요소를 정의하고 각각의 측정 방법을 제시한다. 이와 같이 복잡도의 다양한 요소가 중요한 이유는 코덱의 구현 방법에 따라 각 복잡도의 중요성이 다르기 때문이다. 복잡도 요소 정의에 따른 복잡도 측정 방법은 이러한 요소 중, 코덱 구현에 가장 큰 영향을 미치는 연산 복잡도와 메모리 접근 대역폭을 이용하여 복잡도를 측정한다. 완벽하고 정확한 복잡도의 측정을 위해서는 단 한 대의 컴퓨터만으로 모든 복잡도 측정을 수행해야하나 현실적으로 그러한 조건을 만드는 것은 불가능하다. 따라서 이 복잡도 측정 방법은 비슷한 연산 능력을 갖고 최대한 외부적 요인을 제거한 복잡도 측정 환경을 아래와 같이 제시하였다.

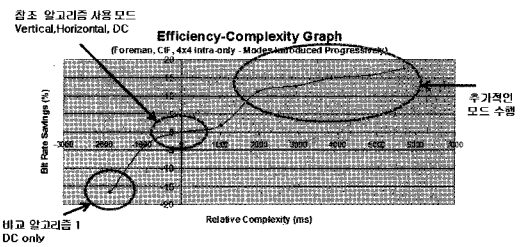
- 특정 컴파일러 사용(ex: GCC 4.1)
- 동일한 코덱 설정파일 사용
- 실험용 컴퓨터의 성능 범위 정의
- 실험 중 파일 및 입출력 I/O 시간 배제
- 백그라운드 프로그램 제거(백신, 방화벽등)

이와 같은 복잡도 측정 환경 하에서 연산 복잡도와 메모리 접근 대역폭을 측정하여 기존의 코덱 압축 성능 평가(BD-rate[7])와 결합, 비디오 코덱의 압축 성능과 코덱 복잡도와의 통합 분석 방법을 제공한다.

<그림 7>의 예는 인트라4x4 코딩에서 사용가능한 모드의 수를 변화시켜 도식화된 코덱 압축 성능/연산복잡도 측정의 예이다. 표의 x축은 수행시간에 기반한 연산 복잡도를 나타내며 y축은 코덱 압축 성능을 나타낸다. 참조 알고리즘은 Vertical, Horizontal, DC 3개의 인트라 4x4 모드를 사용한 경우이고, 비교 알고리즘 1은 DC 모드만 사용한 경우이다. 비교 알고리즘 1은 참조 알고리즘 대비, 연산 복잡도가 크게 줄었음을 볼 수 있으며 더불어 압축 성능 또한 크게 떨어졌음을 볼 수 있다. 이에 반해 추가적인 모드를 수행한 경우, 압축 성능이 조금 향상하지만 연산 복잡도가 크게 상승함을 볼 수 있다.

복잡도 요소 정의에 따른 복잡도 측정 방법은 코덱 복잡도의 다양한 요소를 정의하고 측정 방

〈그림 7〉코덱 압축 성능/연산복잡도 측정의 예[6]



법을 제공한다. 또한 코덱의 실행 시간에 기반하여 복잡도를 측정하므로써 연산 복잡도를 코덱 복잡도에 큰 영향을 미치는 하드웨어 플랫폼의 영향을 반영한다.

하지만 이와 같은 장점에도 불구하고 복잡도 요소 정의에 따른 복잡도 측정 방법은 몇 가지 한계를 갖고 있다.

첫째, 연산 복잡도를 코덱 알고리즘의 실행시간으로 정의함으로써 연산 복잡도뿐만 아니라 메모리 연산에 의한 복잡도가 포함된다. 따라서 연산복잡도의 직관적 정의와는 다른 복잡도를 측정하게 된다.

둘째, 연산 복잡도는 실행시간으로 정의하고 메모리 복잡도는 메모리 접근 대역폭으로 정의함으로써 두 복잡도의 측정 단위가 다르게 된다. 따라서 두 복잡도 간의 크기를 비교 분석하기가 쉽지 않다.

셋째, 항상 동일한 실험 조건을 구현하기 어렵다. 제시된 복잡도 측정 조건을 갖추더라도 운영체제와 백그라운드 프로그램의 영향으로 실행시 다른 결과가 도출 된다.

이 방법은 다양한 복잡도를 정의하고 코덱 구현에 큰 영향을 미치는 요소를 선정하여 복잡도를 측정하였다. 하지만 원래 목적하였던 정의와는 다른 연산 복잡도 측정 결과, 복잡도간의 단위 차이, 불완전한 외부요인 영향들로 인하여 정확하며 일반화된 복잡도 측정하는데 한계를 갖고 있다.

3. 코덱 시뮬레이터에 따른 복잡도 측정

앞서 3.2절의 문제점으로는 잘못된 복잡도 정의, 각 복잡도의 단위 차에 의한 복잡도 비교의 어려움, 불완전한 외부 요인 영향 등이 있다. 이와 같은 문제점들을 해결하기 위하여 코덱 시뮬레이터 기반 복잡도 측정 방법[1]이 제안 되었다. 코덱 시뮬레이터 기반 복잡도 측정 방법은 이전의 잘못된 복잡도 정의 문제를 해결하기 위하여 연산 복잡도, 메모리 복잡도, 전체 복잡도를 아래와 같이 새롭게 정의하였다.

- 새로운 정의 복잡도 정의
  - 연산 복잡도 : 연산만을 처리하는데 소비되는 시간
  - 메모리 복잡도 : 메모리 명령만을 처리하는데 소비되는 시간
  - 전체 복잡도 : 연산 복잡도+메모리복잡도

이렇게 정의되어진 복잡도를 측정하기 다음과 같은 하드웨어 설정을 가정한다.

- 연산 복잡도 측정
  - 코덱 알고리즘에서 연산에 의한 복잡도만를 측정하기 위해, 무제한의 메모리 대역폭을 설정하여 메모리 대역폭 부족으로 인한 지연 발생 방지
  - 메모리 처리시간을 '0'으로 설정하여 메모리 연산에 발생되는 지연 방지
- 메모리 복잡도 측정
  - 메모리 연산에 의한 메모리 복잡도만를 측정하기 위해 연산에 필요한 처리시간을 '0'으로 설정하여 연산에 소비되는 시간을 제거

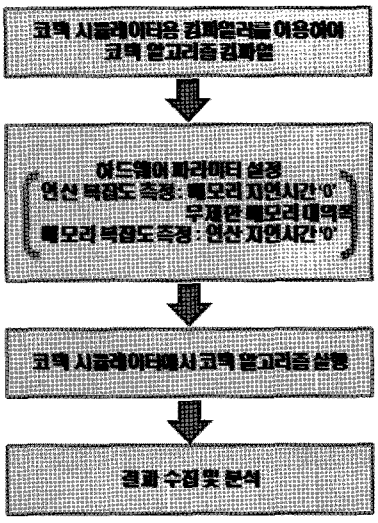
이러한 정의는 연산에 의한 연산 복잡도와 메모리 연산에 의한 메모리 복잡도를 완전히 분리,

측정할 수 있게 하여 실질적 코덱 구현에 필요한 정보를 보다 명확하게 나타낼 수 있게 하여 준다. 그리고 모든 복잡도 측정 단위를 시간으로 하여 각 복잡도간 비교, 평가가 용이하게 하여준다.

위의 하드웨어 설정과 운영체제나 백그라운드 프로그램 등의 외부 영향을 최소화하기 위하여 코덱 시뮬레이터 기반 복잡도 측정 방법[1]은 프로세서 시뮬레이터를 사용하여 복잡도를 측정한다.

코덱 시뮬레이터는 코덱 알고리즘 실행 중 발생하는 모든 하드웨어 이벤트(캐시 미스, 연산 지연시간 및 메모리 지연시간 설정등)를 복잡도 측정에 반영 할 수 있으며 내부 타이머를 따로 사용함으로써 운영체제 및 백그라운드 프로그램 등 외부 요인에 의한 영향을 최소화 할 수 있다.

코덱 시뮬레이터에 따른 복잡도 측정 방법은 다양한 복잡도를 새롭게 정의함으로써 보다 일반적이고 합리적인 복잡도의 측정, 비교를 가능하게 하였다. 또한 코덱 시뮬레이터를 이용하여 외부 요인 제거, 하드웨어 이벤트 반영을 통한 일반적이고 정확한 복잡도의 측정이 가능한 장점을 갖고 있다. 그리고 이 방법의 복잡도 측정 과정은 일반 컴퓨터에서 코덱을 실행하는 과정과 동일하여 하드웨어에 익숙하지 못한 비디오 전문가들이 사용하기 쉽다는 장점이 있다. <그림 8>은 코덱 시뮬레이터를 이용한 복잡도 측정 과정을 보여준다.



〈그림 8〉 코덱 시뮬레이터를 이용한 복잡도 측정 과정

위에서 언급한 것과 같이 코덱 시뮬레이터에 따른 복잡도 측정 방법은 특정 하드웨어 플랫폼 하에서의 복잡도를 측정하기에는 적합하나 다양한 하드웨어 플랫폼을 동시에 고려하기에는 한계가 존재한다. 이를 해결하기 위한 방법을 다음 4절에서 알아본다.

4. 다양한 플랫폼을 고려한 코덱 복잡도 측정

실행기반의 비디오 코덱의 복잡도 측정은 2장에서 알아본바와 같이 하드웨어 플랫폼의 영향을 많이 받게 된다. 따라서 보다 일반적이고 합리적인 복잡도 측정을 위해서는 다양한 하드웨어 플랫폼을 고려한 복잡도 측정 방법 필요하다. 이러한 복잡도 측정의 방법으로는 복잡도 가중치 평균기반의 복잡도 스코어링 방법이 있을 수 있다.

이 방법은 다양한 하드웨어 플랫폼하에서 복잡도를 측정, 측정된 복잡도에 가중치를 적용하여 복잡도 평균을 구하고 이를 바탕으로 복잡도 점수를 평가하는 방법이다. 이때 화질 평가에서와 같이 공통 실험 조건 (Common Experiment Condition[8])과 같은 공통 복잡도 측정 조건을 정의한다면 다양한 하드웨어 플랫폼을 고려한 복잡도를 보다 합리적인 조건으로 측정할 수 있을 것이다. 이를 위하여 다수 하드웨어 플랫폼들을 몇몇 플랫폼클래스로 나누고 (예, GPP

(General Purpose Processor), DSP(Digital Signal Processor), MP(Mobile Processor)) 이를 이용하여 보편타당한 복잡도 측정을 유도할 수 있다.

이와 같이 가중치 평균을 이용한 스코어링 방법은 다양한 하드웨어에서 측정된 복잡도를 기반으로 코덱 알고리즘의 보편화되고 일반화된 복잡도를 측정 할 수 있다.

Ⅳ. 결론 및 향후 복잡도 측정 기술

최근 급격히 높아진 코덱 복잡도로 인하여 코덱 구현의 어려움 증대되고 있다. 이에 MPEG, VCEG와 같은 표준화 기관에서는 차기 표준 코덱 알고리즘 선정 시 복잡도를 고려하여 선정할 예정이다. 따라서 알고리즘의 복잡도를 정확히 측정하는 기준 및 방법에 대한 연구가 활발히 진행 중이다. 본 고에서는 다양한 복잡도 측정 방법을 알아보았다. 이러한 연구의 하나으로써, 코덱 시뮬레이터에 따른 복잡도 측정 방법은 실행 시간 기반 복잡도 측정 방법으로써, 다양한 복잡도를 새롭게 정의하고 복잡도간 비교를 용이하게 만들었다. 또한 운영체제 및 백그라운드 프로그램에 의한 외적 영향을 제거하고 하드웨어요소에 의한 지연 시간을 반영하여 정확하고 다양한 복잡도를 측정하는 수단 및 방법을 제공한다.

이러한 장점에도 불구하고 코덱 시뮬레이터를 이용한 측정 방법은 복잡도 측정 수단으로 코덱 시뮬레이터를 사용하기 때문에 복잡도 측정에 많은 시간이 소모될 수 있다. 또한 점차 보편화되고 있는 멀티코어 프로세서 기반 복잡도 측정도 아직은 지원을 하지 못하고 있다.

따라서 보다 정확하며 합리적인 복잡도를 측정하기 위해서는 기존 코덱 시뮬레이터의 최적화를 통한 고속 복잡도 측정 툴의 개발과 멀티프로세서로 구현된 코덱의 복잡도를 측정할 수 있는 코덱 시뮬레이터의 개발이 이루어져야 할 것이다.

참고문헌

[1] Yungho Choi, Sungho Seo, "A Complexity Measurement Using a Processor Simulator," VCEG-AI28, Jan., 2008.

[2] Gisle Bjontegaard, Arild Fuldseth, "Simplified Chroma Deblocking Filter," VCEG-W10, July., 2004.

[3] John L.Hennessy, David A.Patterson, "Computer Architecture : A Quantitative Approach," 3th edition. Morgan Kaufmann, pp.172-288.

[4] John L.Hennessy, David A.Patterson, "Computer Origanization & Design : The Hareware/Software Interface," 2th edition Morgan Kaufmann, pp.538-564.

[5] Micheal Horowitz, Anthony Joch, "JVT/H.26L Decoder Complexity Analysis," JVT-C152, May., 2002.

[6] Micheal Horowitz, "Toward Useful complexity Evaluation Methods," VCEG-AG19, Oct., 2007.

[7] Gisle Bjontegaard, "Calculation of average PSNR differences between RD-curves," VCEG-M33, May., 2001.

[8] T.K.Tan, G.Sullivan, T.Wedi, "Recommanded Simulation Common Condition for Coding Efficiency Experiments Revision 1," VCEG-AE10r1, Feb., 2007.

저자소개

최 영 호

1991년 연세대학교 전자공학과 학사
1995년 Univ. of Southern California, MS
2001년 Univ. of Southern California, Ph.D
2001년 ~ 2001년 Compaq Corp, Alpha EV8 Senior Engineer
2001년 ~ 2004년 Intel Corp, 차세대 Itanium Senior Engineer
2004년 ~ 2008년 건국대학교 조교수
2008년 ~ 현재 건국대학교 부교수

주관심 분야 : Image/video processing architecture
Processor architecture
Network architecture