MPEG 상에서의 움직임 벡터 추출 병렬화

°이양수, 성순용*, 김영호
부산대학교 전자계산학과, *부산외국어대학교 컴퓨터공학과

A Parallelization of Motion Estimation in MPEG

°Yang-soo Lee, Soon-yong Seong*, Young-ho Kim
Department of Computer Science, Pusan National Univ.
*Department of Computer Engineering, Pusan Univ. of Foreign Studies

요 약

영상 압축의 표준인 MPEG은 대표적 비대칭 코딩방법으로서 인코딩 시간이 디코딩 시간보다 훨씬 많은 시간을 소비한다. MPEG 인코딩 과정은 크게 DCT, 양자화, 움직임 벡터 추출, 가변장 부호화로 구성된다 이중에서 DCT와 움직임 벡터 추출 정도가 많은 계산 량을 가지므로 병렬화의 관심이 되고 있다 본 논문에서는 움직임 벡터의 추출에 관해서 병렬화 하는 기법을 제시한다 병렬화의 단위는 fine-grained이며, 통신 오버헤드를 보다 줄일 수 있는 기법을 제시하였다 최소한의 초기 데이터 할당으로 계산을 시작하여, 계산을 위한 초기화 과정을 줄여 속도를 증대시킨다 통신비용 즉, 메시지 전달 수 및 메시지 전달 홉(hop) 수를 비교하고, 기존의 기법에 대해 한 프레임에 대한 움직임 벡터 추출 시간을 요소로 할 때 보다 나은 결과를 나타냄을 보였다

1. 소 개

오늘날 영상 압축은 전자 도서관, VOD(Video On Demand), HDTV(High Definition Television)와 같은 많은 디지털 비디오 응용에서 중요한 부분을 차지하고 있다 처리할 영상 데이터의 양이 방대하기 때문에 압축 기법은 필수 요소로서 등장하게 되었다 MPEG 표준은 비디오 압축에 있어서, 중간 정도의 화질과 압축 비트율을 지원하고, 오디오 압축을 지원하는 표준으로 많은 관련 시스템들이 소개되고 있다 비대칭 코딩이기 때문에 기존에 나온 상업용 MPEG 인코더는 속도 면에서의 성능 향상을 위해 특별한 하드웨어를 사용하는 것이 일반적이다 예를 들어 C-Cube의 실시간 MPEG-1 비디오 인코더는 8개의 비디오 프로세서 칩을 사용하고, Optibase와 Optivision의 실시간 인코더도 기본적으로 C-Cube의 비디오 프로세서 칩을 사용하고 있다. 단지 프로세서의 성능 향상으로 순수한 소프트웨어만으로 인코더를 구현하는 시도가 또한 제시되고 있으나, 소프트웨어 만으로의 구현은 속도 면에서의 성능 저하를 초래하고, 이것을 극복하기 위한 방법으로 병렬화 방법이 나오게 되었다. 예를 들어 스탠포드에서 DEC 5000상에서 구현한 병렬 인코더[1]나 필립스사가 POOMA machine[2]에서 구현한 병렬 인코더등(각 노드별로 MC68020 CPU, MC68881 FPU, 16MB memory로 구성)은 가격, 성능 면에서 효율적이였다 본 논문의 목적은 네트웍으로 연결된 여러 대의 PC상에서 소프트웨어 병렬 MPEG 인코더를 구현할 때, 가장 시간 소비가 많은 프레임간 예측의 움직임 벡터를 추출하는 계산을 병렬화 시키는 것이나 이 때, 속도 향상을 위해 비디오 데이터의 적절한 할당에 의해 네트웍 상의 통신 량을 줄여 나간다 본 논문의 2장에서 MPEG에 대한 기본적인 소개를 다루고, 3장에서 병렬 MPEG 인코더를 구현하는 것과 관련된 연구를 소개하고, 4장에서 프레임간 예측의 움직임 벡터 추출의 병렬화와 관련된 데이터 할당 기법 및 병렬화 기법 소개의 간단한 성능 평가를 하고 5장에서 결론을 맺는다.

2. 움직임 벡터 추출

Motion estimation은 비디오 시퀀스 상에 움직이는 물체가 있을 때, 참조 프레임에서 현재 프레임으로 움직인 물체의 변위를 측정하는 것을 말한다. 기본적으로 motion estimation은 두 개의 다른 접근 방법[6]이 있다. 우선 pel recursive algorithm(PRA)는 인접한 픽셀들 간의 상관관계가 높을 점우, 이웃한 픽셀들로부터 각 픽셀에 대한 변위를 순환적으로 측정하는 방법이다 이것은 움직임 보상 코딩 방법에서 더 나은 성능을 발휘하나, 구현하기가 힘든 문제가 있다. 따라서 이 방법보다는 성능은 좀 떨어지나, 구현이 용이한 block matching algorithm(BMA) 방법을 많이 사용한다. BMA는 이전 프레임의 검색 영역 내에서 현재 프레임의 매크로 블록에 가장 적합한 블록을 찾는 방법이다 상대적으로 구현이 용이하고, 계산 량이 적은 BMA 방법이 MPEG에서도 채택되어 사용되고 있다

일반적으로 BMA는 현재 프레임의 코딩될 매크로 블록과 참조 프레임의 매크로 블록들 사이의 부정확성 정도를 나타내는 비용 함수를 최소화시키는 것에 의해서 움직임 벡터를 얻는 방법을 사용한다 M을 현재 프레임 Xc에서의 코딩될 매크로 블록이라고 두고, v를 참조 프레임 Xr에서 위치상 일치하는 매크로 블록에서의 변위라고 할 때, 최적의 움직임 벡터 v*는 다음의 수식에 의해서 얻어진다. 이 때 D는 비용 함수를 나타내고, V는 검색영역을 나타낸다.

$$v^* = \min_{v \in V} \sum_{x \in M} D[X_c(x) - X_r(x+v)], \quad v \in V \quad \cdots \text{식 1}$$

비용 함수로는 다음과 같이 주어진 Minimum Mean Absolute Difference(MMAD)를 사용한다.

$$MMAD = \min_{i,j} \left(\frac{1}{mn} \sum_{k} \sum_{l} |X_c(k,l) - X_r(k+i, l+j)| \right) \quad \cdots \text{식 2}$$

(k,l)은 현재 프레임의 m*n의 위치를 나타내고, 참조 프레임의 검색 영역에서의 이동 변위는 스캐닝 방향에서 i 픽셀, j 라인으로 표시한다 motion estimation에서 가장 시간이 많이 소비되는 것은 주어진 검색 영역에서 가장 적합한 매크로 블록을 찾는 것이다 제안된 여러 기법들 사이에도 속도와 압축률 상에 현저한 차이를 보이고 있다. P,

B 프레임을 위해 제안된 검색 기법은 exhaustive, subsample, two-level, logarithmic 등이 있다[7]

3. 관련 연구

MPEG의 병렬화는 가장 시간 소비가 많은 motion estimation 계산을 병렬화 하기 위한 방법으로 접근되었다. 이전에 제시된 방법은 크게 세 종류로 분류된다

① Spatial Parallel algorithm[8]

기본적으로 병렬화의 단위가 매크로 블록 단위이다. 하나의 프레임에 대해서 각 노드들에 매크로 블록을 할당하고, 매크로 블록별로 motion estimation을 수행해 나간다

② Temporal Parallel algorithm[8]

병렬화의 단위가 프레임 단위이다. 비디오 시퀀스를 압축될 비디오 프레임과 필요한 참조 프레임을 포함한 섹션들로 나눈다 이 섹션들은 개별적 파일로 병렬 파일 시스템에 저장된다 각 노드에 한번에 한 섹션씩 할당을 하고, 압축을 한다 압축된 데이터는 병렬 파일 시스템에 저장하고, 각 노드에는 다른 섹션을 할당한다. 주어진 시간에 단지 이는 갯수의 노드들만이 병렬 파일 시스템을 읽고, 쓸 수 있고, 나머지 노드들은 큐에 대기하고, FCFS(First-Come First-Serve) 정책에 따라 병렬 파일 시스템을 억세스 한다 섹션별로 압축된 데이터는 개별적으로 파일에 저장된다

③ Spatial-Temporal Parallel algorithm[8]

기본적으로 temporal parallel algorithm을 기반으로 한다. 각 비디오 섹션들의 압축은 여러 노드들로 구성된 그룹별로 수행되고, 그룹 내에서 다시 노드별로 spatial parallel algorithm이 수행된다. 모든 노드들은 n개의 노드로 구성된 그룹으로 나뉘고, n개의 노드 중에서 m ($m \le n$) 개의 노드가 계산에 사용되고, 나머지 노드들은 특별한 목적을 위해 남겨진다. 그룹 내에서 한 노드가 비디오 섹션을 읽고, 같은 그룹내의 다른 계산 노드들에 데이터를 보낸다 비디오 섹션에서 각 프레임들은 m개의 슬라이스들로 공간적으로 나뉘고, 각 계산 노드들에 의해 병렬 압축된다 압축된 결과는 그룹의 한 노드로 보내지고, 그 노드는 한 프레임에 대한 압축 결과를 조립, 저장한다.

4. 프레임간 예측 및 움직임 벡터 추출의 병렬화

MPEG의 압축 연산은 크게 DCT 변환, 움직임 벡터 측정, 양자화, 가변 길이 코딩으로 구성된다. 따라서, DCT 변환, 움직임 벡터 측정, 양자화, 가변 길이 코딩이 병렬화의 대상이 된다. 병렬화의 대상이 되는 연산들 중에서 상대적으로 시간 소비가 많은 움직임 벡터 측정에 대해 본 논문은 방법을 제시하였다

4.1 시스템 구성 및 데이터 할당

전체 시스템의 구성은 관련 연구에서 언급되었던 spatial-temporal 방법을 사용한다 즉, 기본 큐브가 하나의 그룹을 이루게 되고, 중앙 노드가 그룹 내의 입, 출력을 담당하는 노드가 된다 비디오 데이터를 HBCC 토폴로지 상의 가장 상위 레벨 중앙 노드에서 입력받아 하위 레벨에 있는 중앙 노드에 프레임 단위로 할당한다 그리고, 각 기본 큐브의 주변 노드에 매크로 블록 단위로 데이터를 할당하고, 실질적인 계산을 수행한다 기본 큐브 상에서 매크로 블록을 주변 노드에 할당하는 방법은 8개의 주변 노드에 대해서 동등하게 이루어진다 한 프레임을 16*16의 매크로 블록단위로 나누고, 프레임의 왼쪽에서 오른쪽 순으로, 위쪽에서 아래 순으로 8개의 주변 노드에 매핑 시킨다. 한 프레임이 M*N개의 매크로 블록으로 구성될 때, 주변 노드 ρ (단,0 $\le \rho \le$7) 에 할당되는 매크로 블록의 수 $N_{\rho m}$는 다음 수식과 같다.

$$N_{\rho m} = \lfloor \frac{M*N}{8} \rfloor + c, \quad \left(\begin{array}{l} \rho \le (M*N) \ \% \ 8 - 1, c = 1 \\ \rho > (M*N) \ \% \ 8 - 1, c = 0 \end{array} \right) \quad \text{식 3}$$

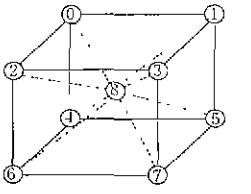
4.2 움직임 벡터 추출의 병렬화 방법

실질적인 움직임 벡터 추출의 계산에 앞서서, 본 논문이 제시하는 방법에서는 우선 멀티캐스팅 그룹을 설정하여야 한다 움직임 벡터 추출을 위한 검색 영역을 찾는 과정에서는 많은 데이터의 중복성이 발생한다 즉, 한 매크로 블록에 대해서 여러 노드에서 참조가 발생하게 된다 따라서 많은 통신이 필요하게 되고, 전체 인코딩 속도를 저하시킬 수 있다 따라서 중복되는 데이터에 대해서는 멀티캐스팅 방법을 사용하여, 한번의 통신으로 효율을 증대시켜야 한다 [그림 1]은 한 프레임에 대해서 기본큐브의 매핑 결과를 보여주고, [그림 2]는 프레임 상의 각 매크로 블록의 주소를 나타낸다

움직인 벡터 추출 기법 중 본 논문의 병렬화 대상은 exhaustive 기법이다. 우선 검색 영역을 코딩될 매크로 블록의 주소 상에서 수평, 수직적으로 ±16으로 정한다 다음과 같은 순서로 병렬화가 수행된다.

0	1	2	3	4	5
6	7	0	1	2	3
4	5	6	7	0	1
2	3	4	5	6	7
0	1	2	3	4	5
6	7	0	1	2	3

참조 프레임 상의데이터 할당



HBCC 기본큐브상의 노드 번호

[그림 1] 참조 프레임에 대한 기본 큐브상의 매핑

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

[그림 2] 매크로블록주소

	0,1	0,2		0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
	3,1	3,2		3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

[그림 3] 움직임 벡터 추출을 위한 계산 순서

① 현재 프레임 상의 코딩될 매크로 블록 데이터 할당

앞장에서 언급한 방식으로 주변 노드에 현재 프레임의 매크로 블록을 할당한다

② 전송될 검색 영역에 대한 멀티캐스팅 그룹 설정

한 프레임이 M*N개의 매크로 블록으로 구성될 때, 기본 큐브의 주변 노드의 수를 P라 하고, p를 노드의 번호라고 하고, 현재 매크로 블록의 주소를 (i, j)라고 하자 이 때, 구해야 할 그룹내 노드 번호는 (i, j)를 기준으로 8방향이다. 여기서 상, 하, 좌, 우로의 노드 번호를 구하면, 나머지 대각선 방향으로는 쉽게 구해진다.

일반적으로 멀티캐스팅그룹을 구하는 수식은 다음과 같다.

i) (i, j-1)를 할당받은 노드 번호: G(p)=(P + p - N) % P

ii) (i, j+1)를 할당받은 노드 번호: G(p)=(p+N) % P

iii) (i-1, j)를 할당받은 노드 번호: G(p)=(P + p - 1) % P

iv) (i+1, j)를 할당받은 노드 번호 G(p)=(p - 1) % P

vi) (i-1, j-1)를 할당받은 노드 번호: i) - 1

vii) (i+1, j-1)를 할당받은 노드 번호 i) + 1

viii) (i-1, j+1)를 할당받은 노드 번호 ii) - 1

ix) (i+1, j+1)를 할당받은 노드 번호: n) + 1

③ 참조 프레임 상의 매크로 블록 주소가 (0,0)인 것부터 멀티캐스팅 움직임 벡터 추출을 위해서 현재 프레임의 매크로 블록의 할당, 참조 프레임 상의 검색 영역의 할당이 선행적으로 이루어져야 한다. 본 논문에서는 가장 최소의 데이터만을 할당하고, 검색 영역을 멀티캐스팅을 통해서 할당하면서 전체적인 데이터 전달 시간과 계산 시간을 오버랩핑시켜 통신 오버헤더를 감소하고자 한다. 초기에 참조 프레임에서 매크로 블록의 주소가 (0,0)인 매크로 블록을 중앙 노드에 의해

서 멀티캐스팅 한다. 첫 매크로 블록이 도착하면 그룹 내에 포함된 주변 노드에서는 계산이 진행된다 기본적으로 프레임의 첫 번째 행의 멀티캐스팅이 끝나면 모든 주변 노드가 계산을 시작하게 된다

3) 멀티캐스팅 그룹 내의 관련 노드들부터 움직임 벡터 추출을 위한 계산 시작.

코딩될 매크로 블록들의 참조 프레임 내의 검색 영역에서 시작 계산 위치는 다음과 같다.

i)매크로 블록 주소가 (0, 0)인 경우 (0, 0)부터 계산시작

ii)매크로 블록 주소가 (0, y)인 경우: (0, y-1)부터 계산시작

iii)매크로 블록 주소가 (x, 0)인 경우: (x-1, 0)부터 계산시작

iv)매크로 블록 주소가 (x,y)인 경우: (x-1,y-1)부터 계산시작

검색 영역내의 이동은 기본적으로 왼쪽에서 오른쪽, 위에서 아래 순이고, 한 픽셀씩 이동하여 계산한다 계산은 매크로 블록단위로 이루어지고, 비용 함수는 MAD(Mean Absolute Difference)를 사용한다. 비용 함수 값이 가장 작은 것을 선택하여, 현재 프레임의 주소와 대응되는 참조 프레임의 주소 값으로부터의 벡터 값을 추출한다

4.3 성능 평가

4.3.1 HBCC와 하이퍼큐브 토폴로지

움직임 벡터 추출을 위해서는 검색 영역의 전송 시 많은량의 메시지 교환이 발생하며 병렬화의 효율은 주로 통신량에 따라 결정된다. 따라서 본 논문에서는 메시지 수의 홉 수를 성능 평가 요소로 정하고, 다음과 같이 통신비용을 정의하여 성능을 평가한다.

통신비용 = 메시지 수 * 전체 홉 수

HBCC 토폴로지와 하이퍼큐브는 데이터의 할당 시, 전체 발생되는 메시지 수나 메시지의 전송 경로상의 홉 수에서 다른 모습을 보인다 HBCC의 경우는 중앙 노드를 중심으로 모든 노드에 이르는 거리가 1로 동일하다 3차원 HBCC의 경우는 8개의 주변 노드가 사용되므로, 전체 통신비용은 8이 메시지 수가된다. D를 차원이라 두고, M을 기본 큐브당 메시지 수라고 했을 때, D차원 HBCC상에서의 전체 통신비용 C는 다음과 같이 나타난다

$$C = (D-2) * M \quad \text{식 4}$$

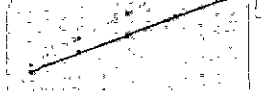
3차원 하이퍼큐브의 경우는 8개의 노드 가운데 하나의 노드가 데이터의 할당을 담당하게 되고, 각 노드로의 거리는 최대 3이된다 노드0을 데이터 할당을 담당하는 노드라고 했을 때, 노드 0에서 각 노드로의 거리는 다음과 같다

노드1=1, 노드2=1, 노드3=2, 노드4=1, 노드5=2, 노드6=2, 노드7=3

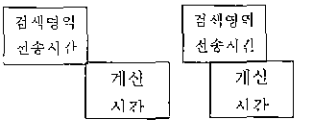
각 노드로 발생되는 메시지 수가 m1, m2, m3, m4, m5, m6, m7이라고 할 때, D 차원 하이퍼큐브 상에서의 전체 통신비용 C는 다음과 같이 나타난다

$$C = (D-2) * (m1+m2+m3*2+m4+m5*2+m6*2+m7*3) \quad \text{… 식 5}$$

[그림 4]는 차원에 따른 HBCC와 하이퍼큐브 상에서의 전체 통신비용을 보여주고 있다 차원이 증가할수록 HBCC가 하이퍼큐브에 비해 통신비용이 더 효율적인 것을 보여주고 있다



[그림 4] 전체 통신비용



[그림 5] 순차적 방법, 오버랩핑 방법

4.3.2 알고리즘 성능 비교

이전 연구[8]에서의 병렬화 방법은 크게 두 가지 과정으로 나누어진다. 첫 번째 과정에서는 움직임 벡터를 추출하기 위해서 검색 영역을 모두 전송하는 과정으로 이후에 움직임 벡터 추출을 위한 계산 시, 통신 오버헤드를 없앨 수 있다 그 다음 과정은 움직임 벡터를 각 매크로블록 단위로 병렬적으로 계산하는 과정이다

[그림 5]의 순차적 방법은 이전 연구의 방식이고, 오버랩핑 방법은 본 논문의 방식이다. 우선 한 매크로 블록에 대해서 걸리는 전송 시간을 T_{macro}라고 하고, 매크로 블록 단위로 움직임 벡터를 추출하기 위해 계산에 걸리는 시간을 C_{macro}라고 하자

[그림 2]상의 6*6 프레임을 할당하여 계산한다고 할 때, 전체 검색 영역을 전송하기 위해 필요한 전송 시간은 $256T_{macro}$ 시간이다 그리고, 전체 움직임 벡터 추출을 위한 계산 시간은 $25600C_{macro}$ 시간이다. 이전 연구의 순차적 방법에서 전체 계산에 수행되는 시간은 전송 시간 $256T_{macro}$ + 계산 시간 $25600C_{macro}$ 이다. 하지만 본 논문에서 제시하는 방법은 [그림 6]상의 첫 번째 행을 멀티 캐스팅 한 뒤, 바로 계산을 수행하므로 전체 계산에 수행되는 시간은 $6T_{macro}$ 전송 시간 + $25600C_{macro}$ 계산 시간이다 M*N 프레임 상에서 걸리는 전체 전송 시간 T_{total}과 전체 계산 시간 C_{total}는 다음과 같다

$$T_{total} = (9MN - 6M - 6N - 68)T_{macro} \quad \text{… 식 6}$$

$$C_{total} = 1024(MN - M - N + 1)C_{macro} \quad \text{… 식 7}$$

따라서, 본 논문의 방법은 기존 연구보다 T_{total}와 NT_{macro}의 카만큼 속도를 향상시킨다

5. 결론 및 향후 계획

본 논문에서는 fine-grained 단위로 움직임 벡터 추출의 과정을 병렬화 시키는 기법을 제시하였다 검색 영역 상의 이동시 발생되는 통신 오버헤드를 줄이기 위해 계산 시간과 통신 시간을 오버랩핑 시켰고, 처음 계산을 위해 최소한의 데이터를 멀티캐스팅을 통해 효율적으로 전송하는 방법을 보였다. 이 알고리즘의 적용에 적합한 성능을 보이는 HBCC 구조를 선택했고, 간단한 통신비용의 비교를 통해 일반적인 다른 구조들 보다 좋은 성능을 보여주는 것을 보였다 앞으로 본 논문에서 제시한 방법을 PC들로 구성된 HBCC 토폴로지 상에 구현하여, 실제적인 성능을 평가해 보고, 분석해 볼 계획이다.

참고문헌

[1] PVRG-MPEG Codec 1.0, Andy C. Hung, Feb. 25, 1993

[2] Frans Systermans and Jan van der Meer, "CD-I Full-Motion Video Encoding on a Parallel Computer," Communications of the ACM, Vol. 34 No. 4, April 1991

[3] 육정화, "계층적 체심 입방 구조의 상호연결망 설계," 부산대학교 석사 학위 논문, 1992

[4] K. Jack, "Video demystified a handbook for the digital engineer," HighText Publications, Inc., California, 1993

[5] K. R. Rao and J. J. Hwan, "Techniques and standards for image, video and audio coding," Prentice Hall PTR, New Jersey, 1996

[6] H.G. Musmann, P. Pirsch, and H.-J. Grallert, "Advances in picture coding," Proceedings of the IEEE Vol.73, No.4, pp.523-548, April 1985

[7] K. L. Gong and L. A. Rowe, "Parallel MPEG-1 Video Encoding," Computer Science Division - Department of EECS, University of California at Berkeley, May 1994 Master's Thesis, Issued as Technical Report 811

[8] S. M. Akramullah, I. Ahmad, and M. Liou, "A data-parallel approach for real-time MPEG-2 video encoding," Journal of Parallel and Distributed Computing, Vol.30, No.2, pp.129-146, November 1995