워크스테이션 클러스트 환경에서 병렬 BMA의 구현 및 성능 분석

김종렬, 나현태, 김정선, 문영식
한양대학교 전자계산학과
E-mail : {jrkim, htna, jskim, ysmoon}@cse.hanyang.ac.kr

Performance Evaluation of Parallel BMA on Networked Cluster of Workstations

Jong Ryul Kim, Hyun Tae Na, Jung Sun Kim, Young Shik Moon
Department of Computer Science and Engineering
Hanyang University

요약

본 논문에서는 동영상에서 움직임 벡터를 찾는 방법 중의 하나인 BMA(Block Matching Algorithm)를 워크스테이션 클러스터(cluster of workstations) 환경하에서 구현하고 이에 대한 성능 분석 모델을 제시한다. 동영상에서 움직임 벡터를 찾는 BMA는 영상처리 및 컴퓨터 비전 분야에서 널리 사용되는 방법으로 병렬화를 통해 처리 속도를 단축시킬 수 있는 알고리즘이다. 그러나 워크스테이션 클러스트 환경하에서는 데이터의 분할 및 각 노드간의 통신 방법에 따라서 전체적인 성능에 많은 영향을 미친다. 따라서 본 논문에서는 최적의 데이터 분할 및 각 노드간의 통신을 최소화하는 병렬 BMA를 설계·구현한다. 또한 데이터의 분할 및 각 노드간의 통신을 고려한 성능 모델을 제시하여 프로세서의 증가 및 데이터의 분배에 따른 성능을 예측하고, 실험 결과를 통하여 제시한 모델의 타당성을 입증한다.

1. 서 론

일반적으로 연속 영상(비디오 데이터)은 탁월한 정보 전달 능력을 가지고 있다. 그러나 비디오 정보는 막대한 양의 데이터를 처리해야 한다는 문제점 때문에 대중화에 어려움이 많았다. 하지만 최근에는 네트워크 기술의 발달과 디지털 비디오 압축 기술, 컴퓨터의 성능 향상으로 인해 다양한 형태의 디지털 비디오 서비스가 가능하게 되었다. 막대한 양의 비디오 데이터를 실시간 처리 및 다양한 응용분야에서 사용하기 위해서는 데이터 압축과 병렬처리 과정을 요구한다. 이미 이러한 실시간 영상처리 및 압축을 위해 소프트웨어를 기반으로한 다양한 형태의 병렬처리가 시도되었다.[1, 2]

비디오 데이터에서 움직임 벡터를 찾는 방법으로는 BMA(Block Matching Algorithm)와 광류(optical flow)를 계산하는 방법 등이 있다[3]. 이러한 방법들은 영상처리 및 컴퓨터 비전 분야에서 널리 사용되고 있는 알고리즘이다. 그 중 BMA 방법은 블록 단위의 데이터 처리 특징 때문에 병렬화의 과정이 쉽고 확장성이 뛰어나 병렬 알고리즘으로 구현하기에 용이하다. 또한 BMA 방법은 이미 표준화된 비디오 압축 기술인 MPEG(Moving Picture Experts Group)-1, MPEG-2, H.261에서 움직임을 찾는 방법으로 채택되었고, 하드웨어적인 구현이 용이하다는 장점도 있다[4]. 따라서 본 논문에서는 병렬구현 시 처리 속도를 극대화 할 수 있는 BMA 방법을 워크스테이션 클러스터 환경하에서 구현한다.

병렬 구현시 각 프로세서의 계산 능력에 따라서 데이터 분할을 최적화 하고 노드간의 통신을 최소화하기 위해 데이터를 중첩시켜 분배하며, 다양한 형태로 데이터를 분배 및 프로세서에 할당을 한다. 또한 구현된 병렬 BMA에서의 데이터의 분할, 분배 방법에 따른 성능 모델을 제시하고 성능을 분석한다.

2. BMA(Block Matching Algorithm)의 개념

블록 기반 움직임 추정(motion estimation)과 보정(compensation)은 영상처리 및 컴퓨터 비전 분야에서 가장 널리 사용되고 있는 방법 중의 하나이다. 움직임이란 그림 1과 같이 현재의 영상내에 있는 임의의 블록이 연속된 다음의 영상에서 움직인 정도를 벡터 형태로 나타낸 것을 말한다. 이러한 BMA 방법을 구현하기 위해서는 matching criteria, search strategy, 블록의 크기와 search 범위등이 중요하게 고려되어야 한다[3].

Matching criteria란 현재 영상에서의 임의의 블록과 다음 영상에서 임의의 블록간의 유사도를 구하는 척도를 말한다. 흔히 사용되는 방법으로는 MSE(Mean Square Error), maximum cross-correlation등을 사용한다. Search strategy란 현재 영상에서의 임의의 블록과 다음 영상에서

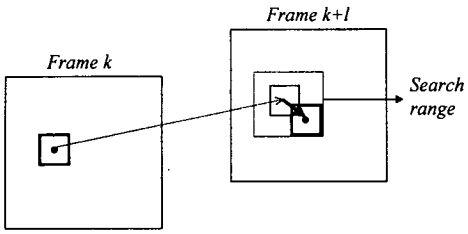

그림 1. BMA의 개념

의 임의의 블록을 찾기 위한 방법이다. 일반적으로 모든 블록을 비교하면 정확성은 높지만 시간상의 이유로 three-step search, cross search등을 사용한다. 본 논문에서는 모든 블록에 대하여 움직임을 찾는 full search를 사용하고, 검색 범위는 ±4로 구현한다.

3. MPI를 사용한 BMA의 병렬 구현

BMA를 병렬적으로 구현하기 위해서는 전체적인 communication 모델과 데이터의 분할 및 분배 방법을 결정해야 한다. 구현된 병렬 BMA 방법에서는 one-to-all personalized communication을 사용하고, 데이터 분할은 프로세서의 성능에 따라 분할되는 데이터의 크기를 다르게 결정한다. 또한 움직임을 찾기 위한 통신을 최소화하기 위해 데이터 분배시 분할되는 데이터의 경계면을 중첩시킨다. 각 프로세서에 데이터를 분배 및 할당시키는 방법으로는 1-D slice형태, 2-D mesh형태, hypercube형태로 구현한다.

3.1 데이터의 분배 및 프로세서 할당

- 1-D slice 형태

1-D slice 형태의 데이터 분배 및 프로세서 할당은 그림 2와 같이 하나의 프로세서가 나머지 프로세서의 성능에 따라서 영상을 slice 형태로 분할하여 각각의 프로세서에 순차적으로 분배한다.

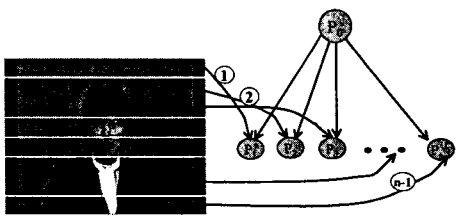

그림 2. 1-D slice 형태의 데이터 분배

- 2-D mesh 형태

2-D mesh 형태의 데이터 분배는 그림 3과 같이 먼저 수평방향으로 프로세서의 성능에 따라서 데이터를 분할하여 할당한 후, 동시에 수직방향으로 데이터를 각각의 프로세서에 분배한다.

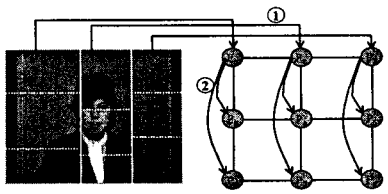

그림 3. 2-mesh 형태의 데이터 분배

- Hypercube 형태

Hypercube 형태인 경우는 그림 4와 같이 먼저 인접 프로세서로 데이터를 분배하고 다시 2개의 프로세서가 동시에 인접 프로세서로 데이터를 분배한 후, 최종적으로 4개의 프로세서가 인접 프로세서로 동시에 데이터를 분배하는 방법이다.

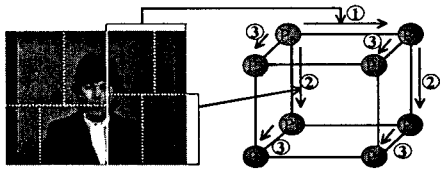

그림 4. hypercube형태의 데이터 분배

3.2 병렬 BMA 성능분석 모델

본 논문에서는 데이터의 분배 및 프로세의 할당 방법에 따라 병렬 BMA의 성능 분석 모델을 제시한다. 먼저 각각의 분석 모델에서 사용될 용어(terminology)는 다음과 같다.

t_b : 2개의 프로세서 사이에서 1-byte의 데이터 전송시간
t_s : 데이터 전송을 위한 setup time
p : 프로세서 전체의 수
H, V : 한 프레임의 가로, 세로크기
s_o : 중첩될 데이터 길이
B : 블록 크기
T_{send} : 데이터를 각 프로세서로 분배하는데 걸리는 시간
T_{recv} : 각 프로세서로부터 결과를 수집하는데 걸리는 시간
T_{comp} : 각 프로세서에서 BMA를 수행하는데 걸리는 시간
T_{total} : 한 프레임의 데이터를 처리하는데 걸리는 시간
T_i : 프로세서 i만을 사용하여 한블록을 처리할 때의 시간
U_i, : 프로세서 i의 계산 능력
U_i' : 하나의 그룹에서 프로세서 i의 상대적 계산 능력

하나의 프레임 데이터를 처리하기 위한 전체 시간은 각 모델에 관계없이 식 (1)과 같다.

$$T^n = \max\left[T^{n-1}, \sum_{k=1}^{n} T_{send}^k + T_{comp}^n\right] + T_{recv}^n \qquad (1)$$

또한 각 프로세서의 능력에 따라서 데이터를 분배하기 위해, 각 프로세서의 능력은 식 (2)와 같이 정규화하여 결정한다.

$$U_i = \frac{\frac{1}{T_i}}{\sum_{j=1}^{p}\frac{1}{T_j}}, U_i' = \frac{\frac{1}{T_{i\times\sqrt{p}}}}{\sum_{j=1}^{\sqrt{p}}\frac{1}{T_{j\times\sqrt{p}}}} \qquad (2)$$

반면 각 모델에 따른 데이터 분배 및 결과 수집 시간은 데이터 분배 및 프로세서 할당 방법에 따라서 다르다. 따라서 각각의 경우에 대하여 데이터 분배 및 결과 수집 시간(T_{send}, T_{recv})을 계산할 필요가 있다.

1-D slice에서의 T_{send}와 T_{recv}는 전체의 데이터를 각 프로세서의 성능에 따라 분배하므로 식 (3)과 같이 계산할 수 있다.

$$T_{send} = \sum_{i=2}^{p}\left(t_s + t_b\left(U_i \times HV + 2s_o \times V\right)\right),\quad T_{recv} = \sum_{i=2}^{p}\left(t_s + t_b\left(U_i \times \frac{VH}{B^2}\right)\right) \quad (3)$$

2-D mesh에서의 T_{send}와 T_{recv}는 수직 방향에 있는 프로세서 성능의 합에 의해서 데이터가 분배 된 후, 각 프로세의 성능에 따라 데이터가 분배되므로 식 (4)와 같이 계산할 수 있다.

$$T_{send} = 2t_s(\sqrt{p}-1) + t_b\left(\sum_{i=2}^{\sqrt{p}}\left(\sum_{j=1}^{\sqrt{p}-1}U_{i+\sqrt{p}\times j}\times HV + 2s_o\times V\right) + \sum_{i=2}^{\sqrt{p}}\left(\sum_{j=1}^{\sqrt{p}-1}U_{\sqrt{p}\times j}\times H + 2s_o\right)\left(U_i\times V + 2s_o\right)\right)$$

$$T_{recv} = 2t_s(\sqrt{p}-1) + t_b\left(\left(\sum_{i=2}^{\sqrt{p}}\left(\sum_{j=1}^{\sqrt{p}-1}U_{i+\sqrt{p}\times j}\times HV\right) + \sum_{i=2}^{\sqrt{p}}\left(\sum_{j=1}^{\sqrt{p}-1}U_{\sqrt{p}\times j}\times H\right)\left(U_i\times V\right)\right)/B^2\right) \quad (4)$$

Hypercube형태인 경우 T_{send}와 T_{recv}는 차원에 따라서 반복적으로 프로세서의 능력이 결정되고 이에 따라서 데이터의 크기가 결정되므로 식 (5)와 같이 계산된다.

$$T_{send} = t_s \log p + t_b\left(\sum_{i=1}^{\log p/2}\left(V\prod_{j=1}^{i}\alpha(j) + 2s_o\right)\left(H\prod_{j=1}^{i}\beta(j) + 2s_o\right)\right)$$

$$\alpha(i) = \begin{cases}\dfrac{A(i-1)-A(i)}{A(i-1)}, & \text{if } i \text{ is even}\\ 1, & \text{otherwise}\end{cases}$$

$$\beta(i) = \begin{cases}\dfrac{A(i-1)-A(i)}{A(i-1)}, & \text{if } i \text{ is odd}\\ 1, & \text{otherwise}\end{cases}$$

$$A(i) = \sum_{j=1}^{p/2^i}U_j$$

$$T_{recv} = t_s \log p + t_b\left(\sum_{i=1}^{\log p/2}\left(V\prod_{j=1}^{i}\alpha(j)\right)\left(H\prod_{j=1}^{i}\beta(j)\right)/B^2\right) \quad (5)$$

5. 실험 결과

- 실험 환경

실험한 플랫폼으로 NOW(Network of Workstations)을 사용하였다. 구축된 환경은 Sun Ultra-1 4대, Sun Ultra-5 2대, Sun Ultra-10 1대, Sparcstation 2대로 구성되어 있다. 네트워크는 10-Mb/s switched Ethernet으로 사용되었다. Communication 환경으로는 오하이오 대학에서 개발한 LAM(Local Area Multicomputer) 버전 6.1을 사용하였다. 사용된 LAM은 MPI-1을 기본으로 구현되었다.

실험에 사용된 데스트 시퀀스는 Qcif 포맷의 car-phone, miss america 시퀀스 등이다. 각 프레임의 크기는 174×144이다.

- 실험 결과

프로세서의 계산능력에 따라서 1-D slice, 2-D mesh, hypercube 형태로 데이터를 분배한 경우 한 프레임의 데이터에 대하여 병렬 BMA방법을 통해 움직임을 찾는 시간은 프로세서의 개수에 따라 표 1과 같다.

표 1 프로세서의 개수에 따른 수행시간(측정값/이론값)

데이터 분배방법 \ 프로세서의 개수	1	2	4	8
1-D slice	4.91/4.91	1.632/1.630	0.961/0.958	0.594/0.590
2-D mesh	4.91/4.91	1.614/1.611	0.922/0.919	0.564/0.561
Hypercube	4.91/4.91	1.631/1.629	0.914/0.912	0.560/0.559

- 성능향상(speedup)

프로세서의 개수가 증가됨에 따른 성능 향상은 식 (6)과 같이 계산된다. 구현된 병렬 BMA에서 프로세서의 개수가 증가됨에 따른 성능 향상은 그림 5와 같다.

$$sp = \frac{T_s}{T_m} \quad (6)$$

식 (6)에서 T_s는 단일 프로세서를 사용하여 계산한 시간이고, T_m은 m개의 프로세서를 사용하여 계산한 시간이다.

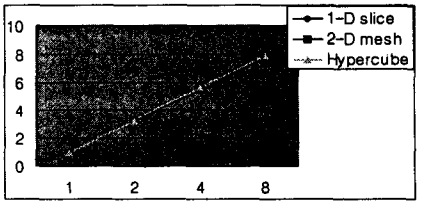

그림 5. 프로세서의 개수에 따른 성능향상

6. 결론

본 논문에서는 BMA방법을 NOW(Network Of Workstations) 환경하에서 병렬적으로 구현하고 성능 모델을 제시하였다. 병렬 알고리즘을 구현시 중요하게 고려해야할 데이터의 분할 및 분배, 프로세서 할당등을 다양한 모델로 구현하였고, 그에 따른 성능 분석을 위한 모델을 제안하였다.

구현된 병렬 BMA 방법은 2-D mesh, Hypercube형태의 데이터 분배 및 프로세서 할당을 고려하였기 때문에 NOW 플랫폼뿐만 아니라, 실제의 병렬처리 시스템(예를들면, Intel Paragon XP/S, Intel iPSC/860 Hypercube등)에서의 구현 및 성능분석에도 사용될 수 있다.

참고 문헌

1. K. L. Gong and L. A. Rowe, "Parallel MPEG-1 video encoding," Proc. Picture Coding Symp., Sept. 1994.
2. A. C. P. Loui, A. T. Ogielski, and M. L. Liou, "A parallel implementation of the H.261 video coding algorithm," Proc. IEEE Workshop on VSCP, Sept. 1992.
3. A. M. Tekalp, Digital Video Processing, Prentice Hall, 1995.
4. S. Eckart and C. Fogg, "ISO/IEC MPEG-2 software video codec," Proc. SPIE, Feb. 1995.