특집

고속 비디오 처리를 위한 병렬화 기술

심동규·남정학 (광운대학교)

I. 서 론

최근 멀티미디어 단말기의 보편화와 함께 사용자들은 과거 저해상도 영상에서 이제는 고해상도 영상에 대한 서비스를 요구하고 있다. 이에 멀티미디어 단말기를 위한 고화질 영상 서비스에 대한 연구가 활발히 진행되고 있다. 영상 해상도의 증가에 따라 늘어난 연산량을 처리하기 위해서는 고성능의 프로세서 사용이 필요하다. 그러나, 현재의 기술에서 프로세서의 클럭 속도를 올리고 고집적을 통한 성능의 향상은 한계에 이르렀다. 이 문제를 해결하고자 최근에 개발된 플랫폼은 시스템의 클럭 속도를 낮추는 대신 프로세서의 수를 늘려 성능 향상을 이루는 멀티 코어 또는 멀티프로세서의 형태로 변하고 있다. 멀티코어 환경에서 응용 프로그램에 대한 속도를 개선하기 위해서는 이전의 많은 응용프로그램들을 멀티코어 환경에 적합하도록 다시 작성돼야 하며, 이는 많은 시간과 노력이 필요할 지도 모른다. 그러나, 우리는 다음의 이유에 의해 멀티 코어로의 전환이 필요하다.

먼저, 멀티코어를 사용하게 되면 전력의 소모량을 줄일 수 있다. 일반적으로 전력은 프로세서의 클럭 속도에 비례하여 증가한다. 그러나 모바일 멀티미디어 단말기와 같은 제한된 환경에서는 전력의 제한 때문에 클럭 속도를 계속해서 증가 시킬 수 없다. 반면, 듀얼코어의 경우에는 동일한 전력 소모량으로 약 70% 정도의 성능 향상을 얻을 수 있게 된다.

멀티코어 사용의 또 다른 이유는 명령어 단위의 병렬 처리 방법이 더 이상 클럭 속도의 증가폭을 따라가지 못하기 때문이다. 초기에 개발된 프로세서에서는 명령어 단위 병렬 처리 방법으로 SIMD (single instruction multiple data), VLIW (very long instruction word), 파이프라인 (pipeline), 슈퍼 스칼라 (super-scalar) 등과 같은 방법들을 사용하여 성능을 향상시킬 수 있었다. 그러나 최근의 응용 프로그램들은 매우 복잡하여 단순히 명령어 단위의 병렬 처리를 적용하기 쉽지 않으며, 프로세서의 클럭은 명령어 단위의 병렬 처리로 극복할 수 있는 한계에 이르렀다.

마지막으로 메모리의 클럭 속도는 프로세서의 클록 속도에 비하여 매우 느리게 발전되고 있다. 이는 프로세서의 클록 속도가 발전하면 발전 할수록 메모리와의 성능차이는 더욱 커지게 된다.

멀티 코어를 사용하여 클럭 속도를 낮춘다면, 이러한 성능 악화의 상황을 막을 수 있다.

본 고는 멀티코어 환경에서 비디오 코덱의 병렬 처리를 위한 방법과 병렬 처리 구현 시 고려사항에 대하여 설명하고자 한다. 점점 더 고화질의 영상을 원하는 사용자들에게 고품질의 멀티미디어 서비스를 위해서 비디오 코덱에 대한 병렬 처리는 필요할 것이다.

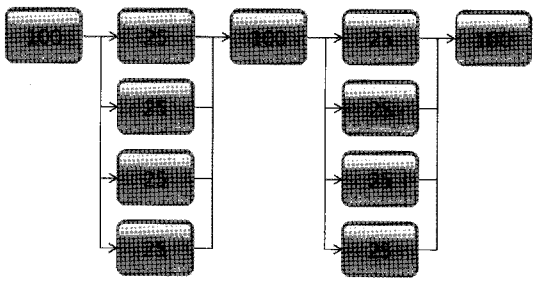
II. 일반적인 병렬 처리 방법

일반적인 컴퓨터 시스템에서 멀티 코어의 사용에 의한 프로그램 처리 속도 향상은 코어 개수에 비례하여 증가할 것이다. 멀티 코어와 전체적인 성능 향상에 대해 설명해주는 대표적인 두 가지 이론이 있다.

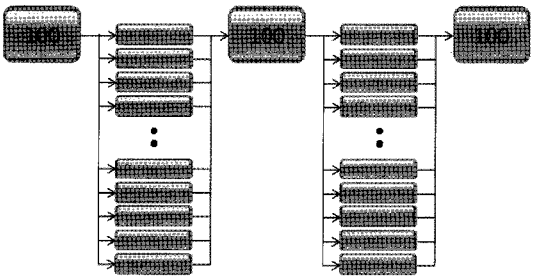
첫 번째로, 암달 (Gene Amdahl)의 법칙이 있다. 이 법칙은 프로그램의 전체 성능은 순차적으로 실행되어야 하는 부분에 의해서 좌우된다는 이론이다. 예를 들어, 다섯 부분으로 구성된 프로그램이 있다고 가정해보자. 프로그램의 각 부분을 처리하는데 100 단위의 시간씩 필요하며, 프로그램의 두 번째와 네 번째 부분에 대해서는 병렬 처리가 가능하다고 한다. 만약, 전체 프로그램을 단일 코어에서 수행할 경우 500 단위의 시간이 소요된다. 이 프로그램에 대하여 <그림 1>과 같이 네 개의 코어를 가진 환경에서 실행할 경우에는 두 번째와 네 번째 부분이 병렬 처리될 수 있다. 각 코어에서는 25 단위의 시간 동안 일을 나누어 처리하여 부분적으로 수행 시간을 1/4로 줄일 수 있다. 전체 프로그램에 대한 수행 시간은 350 단위의 시간이 걸리며, 이는 전체 성능의 30% 속도 향상을 보인다.

유사한 방법으로 <그림 2>와 같이 코어 수가 무수히 많은 경우를 생각해 보자. 코어가 아무리 많아도 전체 프로그램의 성능은 병렬 처리가 되지 않는 첫 번째, 세 번째, 다섯 번째 부분의 성능을 더한 300 단위 시간으로 수렴하게 된다. 즉, 병렬 처리에 따른 성능 향상은 순차적으로 실행되는 부분에 의해서 결정되며, 이 시스템의 성능 향상은 최대 40% 까지로 제한된다.

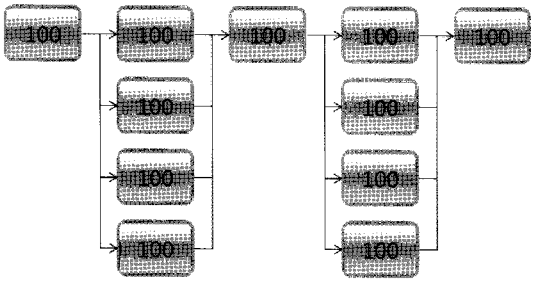
암달의 법칙은 멀티 코어 환경에서 컴퓨터 시스템 성능 향상에 한계가 있다는 것을 의미한다. 이와 달리, 구스타프손 (Gustafson)은 성능 향상에 대해 다른 접근을 사용하였다. 그는 현재 컴퓨터 시스템은 점차 멀티 프로그램 형태로 동작되며 처리해야 하는 데이터량도 기하급수적으로 증가한다고 가정하였다. 그리고 멀티 코어를 이용한 병렬 처리의 성능 향상은 같은 시간 내에 처리되는 데이터량을 증가된다면 전체적인 시스템의



〈그림 1〉 코어 네 개인 경우, 암달의 법칙에 의한 병렬 처리 성능



〈그림 2〉 암달의 법칙에 따른 병렬 처리 성능 한계



〈그림 3〉 구스타프손 법칙에 따른 병렬 처리 성능 향상

성능이 향상될 것이라는 이론을 제시하였다. 즉, 처리할 데이터의 양이 충분히 많다면, 프로그램 내의 순차적인 부분들의 영향력이 약화되기 때문에, 코어의 수가 증가되어도 얼마든지 병렬처리 속도를 향상 시킬 수 있다는 것이다.

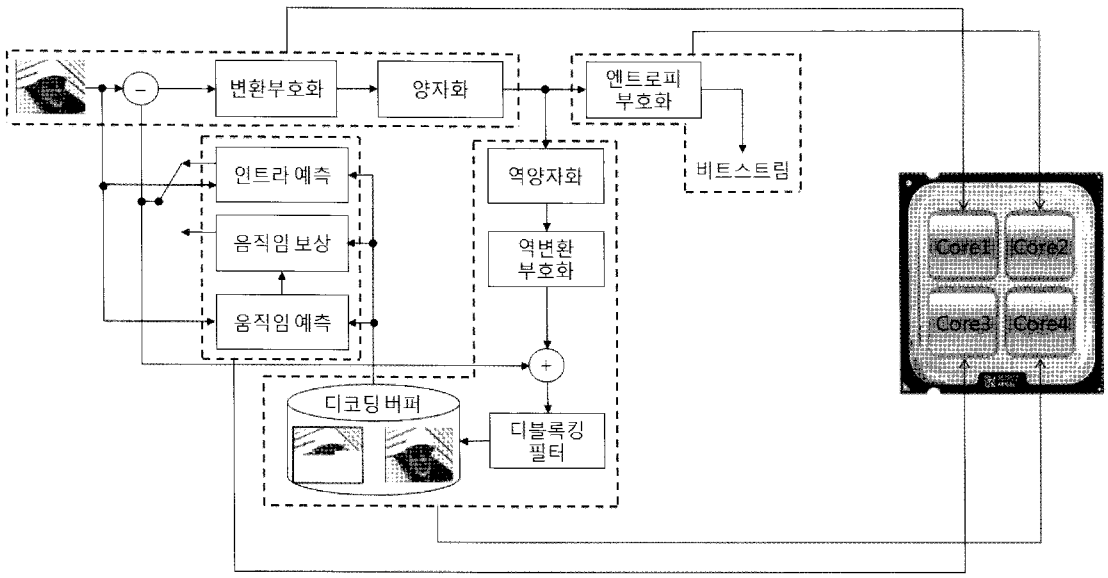
예를 들어, <그림 3>과 같이 코어 네 개를 사용하여 병렬 처리를 수행하는 경우, 500 단위의 시간 동안 1100 단위에 처리할 일을 양을 수행할 수 있다. 전체적으로 처리 속도는 2.2배 향상된다. 구스타프손 이론에 따른 가용 코어 수에 따른 성능 향상률은 다음 식 (1)과 같다.

$$\text{속도 향상}(N) = N + S \times (1 - N) \qquad (1)$$

N : 가용 코어의 개수
S : 순차 처리하는 비율

식 (1)에서 시스템의 성능 향상은 가용 코어의 개수와 비례하여 증가함을 보여준다. 따라서 코어의 개수를 무한이 증가시키면 이론적으로 데이터 처리 능력 또한 무한대로 증가하게 될 것이다.

예를 들어, 네 개의 코어를 가진 시스템에 비디오 코덱의 병렬 처리를 구현했다고 생각해보자. 결과적으로 네 개의 코어가 각기 다른 작업을 병렬적으로 처리하여 단일 코어에 비하여 속도 향상을 얻을 수 있을 것이다. 그러나 우리가 기대했었던 네 개의 코어를 사용하여 네 배의 속도 향상은 얻을 수 없다. 이는 각 코어들 사이에 공유 메모리에 대한 읽기와 쓰기 작업을 요청할 때 발생하는 지연과 캐시 메모리의 일관성을 유지



〈그림 3-1〉 H.264/AVC 코덱에 대한 태스크 단위의 병렬 처리 방법

하기 위한 동기화 신호 요구에 따른 병렬 처리 오버헤드가 발생하기 때문이다. 또한, 비디오 코덱을 병렬로 수행함에 있어서 알고리즘의 특성에 따른 추가적인 동기화 신호가 필요할 수도 있다. 따라서 현실적으로는 코어의 수에 따른 이상적인 성능을 얻기는 쉽지 않다.

멀티 코어 기반의 대표적인 병렬 처리 방법으로는 태스크 단위의 병렬 처리와 데이터 단위의 병렬 처리 방법이 사용될 수 있다.

태스크 단위의 병렬 처리는 하나의 작업에 대하여 여러 개의 하위 작업으로 분할한다. 그리고 분할된 하위 작업을 각 코어에서 나누어서 수행하여 전체적인 처리 효율을 높이는 방법이다. 마치 공장의 컨베이어벨트 시스템에서 여러 사람이 하나의 완성된 제품을 만들기 위하여 일을 분업화하는 과정과 유사하다. 완성된 제품이 시스템에서 처리해야 하는 데이터에 해당하고 공장의 작업자들이 멀티 코어의 코어에 해당된다. 전통적인 병렬 처리 기법 중에 하나인 명령어 단위의 파이프라인 방법과도 유사한 개념이다. 여기서 각 코어는 서로 다른 기능을 병렬적으로 수행하게 된다.

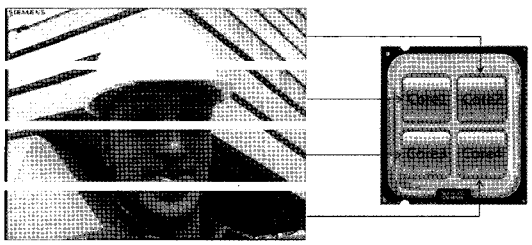
다른 한 가지 방법은 데이터 단위의 병렬 처리 방법이다. 이 방법은 동일한 작업에 대하여, 여러 코어에서 데이터를 나누어 처리하는 형태로 시스템 성능을 높이고자 하는 것이다. 예를 들어, 주어진 시간에 더 많은 양의 제품을 생산하기 위해서 공장을 여러 개 가동하는 것이라 할 수 있다. 여기서 공장이 코어에 해당되며, 각 코어는 동일한 기능을 병렬적으로 수행하게 되는 것이다.

III. 비디오 코덱의 병렬 처리 방법

최근 몇 년 동안 비디오 코덱의 병렬 처리에 대한 많은 연구가 진행되고 있다. 본 고에서는 다양한 비디오 코덱 표준 중에 가장 최신의 비디오 코덱인 H.264/AVC의 병렬 처리 방법에 대하여 소개한다. H.264/AVC 코덱의 병렬 처리 방법도 앞에서 소개한 태스크 단위의 병렬 처리와 데이터 단위의 병렬 처리 방법을 기초로 연구되고 있다. <그림 3-1>은 태스크 단위의 코덱 병렬 처리 방법을 보여준다. H.264/AVC에는 크게 인트라 예측, 움직임 예측, 움직임 보상, 변환 부호화, 양자화, 엔트로피 부호화, 디블록킹 필터, 그리고, 인코더 내에서 복호화를 위한 역변환 부호화 및 역양자화 등의 기능 블록들이 존재한다. 이와 같은 다양한 블록들에 대하여 <그림 3-1>과 같이, 시스템 코어의 개수와 동일한 개수로 기능 블록들을 그룹화 한다. 각 코어에서는 분할된 그룹에 포함된 기능에 대해서 병렬적으로 처리하게 된다.

<그림 4>는 데이터 단위의 코덱 병렬화 방법을 보여준다. 영상을 특정한 개수 또는 코어 개수와 동일하게 분할하고, 각 분할된 영상을 각 코어에서 독립적으로 부호화를 수행하면 된다. 각 코어는 모두 H.264/AVC 부호화기 기능을 수행하게 된다.

H.264/AVC 코덱에 대한 태스크 단위의 병렬 처리 방법과 데이터 단위의 병렬 처리 방법에 대한 특징을 살펴보자.



〈그림 4〉 H.264/AVC 코덱에 대한 데이터 단위의 병렬 처리 방법

먼저, 두 병렬 처리 방법에서 요구되는 데이터 대역폭을 비교해 보자. 태스크 단위의 병렬 처리는 특정 코어에서 수행된 작업에 대한 결과 데이터를 다른 코어에 넘겨주어야 한다. 예를 들어, 영상 부호화 시 사용했던 예측 값, 변환 부호화 계수 값, 양자화된 계수 값 등과 같은 데이터를 한 코어에서 다른 코어로 전달해 주어야 한다. 따라서 태스크 단위의 병렬 처리에서는 높은 데이터 대역폭이 요구된다. 상대적으로 데이터 단위의 병렬 처리는 각 코어가 영상의 서로 다른 부분을 독립적으로 부호화하기 때문에 코어간의 데이터 통신이 거의 요구되지 않는다. 이는 시스템 내부에서 낮은 대역폭을 가지는 장점이 있다.

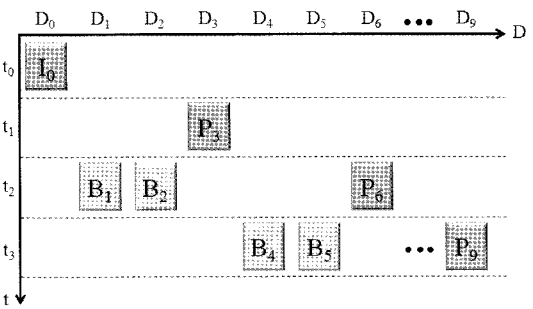
다음으로 작업 분배 (load balancing) 문제를 생각해 보자. 멀티 코어 환경에서 최적의 성능을 얻기 위해서는 모든 코어가 작업을 균등하게 나누어서 수행해야 한다. 즉, 작업을 먼저 끝낸 코어가 작업이 아직 끝나지 않는 다른 코어의 작업을 기다리는 일이 최소화되도록 코어간의 적절한 작업 분배가 필요하다. 그런데 H.264/AVC 부호화기의 계산량은 움직임 예측이 약 80% 이상을 차지하고 있다. 이것은 태스크 단위의 병렬 처리에서 움직임 예측을 담당하고 있는 코어에 많은 작업이 집중되어 있음을 의미한다. 반면, 데이터 단위의 병렬 처리는 각 코어 간의 연산 시간 차이가 크지 않다. 분할된 영상의 특징에 따라 일부 계산량의 차이가 있을 수 있으나, 평균적으로는 각 코어마다 작업이 유사하게 분배될 것이다.

마지막으로 코어 개수에 따른 병렬 처리의 확장성에 대하여 알아보자. 현재 시스템의 코어 수가 N개일 때, 태스크 단위의 병렬 처리를 구현하기 위해서는 코덱의 내부 기능 블록들을 계산량에 따라 N개로 균등하게 나누면 된다. 이 때, 코어 하나가 새롭게 추가되면, 기능 블록을 N+1개로 나누어지도록 코덱 전체를 재설계해야하는 문제점이 있다. 하지만 데이터 단위의 병렬 처리에서는 코어가 새롭게 추가되는 경우, 간단히 영상 분할만 추가하여 구현 할 수 있다.

앞서 살펴본 바와 같이, 데이터 단위의 병렬 처리는 태스크 단위의 병렬 처리에 비하여 낮은 대역폭을 사용하며, 작업 분배가 효율적이고, 코어에 대한 확장성이 뛰어난 장점이 있다. 실제로 최근의 코덱 병렬 처리는 대부분 데이터 단위의 병렬 처리 기반으로 활발히 연구되고 있다.

지금부터는 H.264/AVC의 데이터 단위 병렬 처리 방법에 대하여 살펴보자. H.264/AVC는 부호화를 처리하는 데이터 단위에 따라 프레임, 슬라이스, 매크로블록 단위로 세분화되어 나눠질 수 있다.

먼저, 프레임 단위의 병렬 처리는 B 프레임을 이용하여 간단히 구현 가능하다. IBBP 구조로 부호화를 수행할 경우, B 프레임은 다음 프레임을 부호화 할 때 참조 영상으로 사용되지 않기 때문에 인접해 있는 P 또는 B 프레임들과 동시에 부호화가 가능하다. <그림 5>는 프레임 단위의 병렬 처리에 대한 예를 보여준다. D는 디스플레이 순서이며, t는 부호화되는 시간을 나타낸다. I, B, P는 각각 프레임 종류를 나타낸다. 그림에서 같은 시간에 있는 프레임들은 병렬 처리가 가능하다.

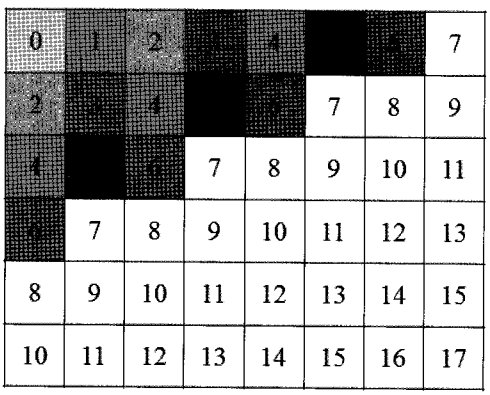


〈그림 5〉 프레임 단위의 병렬 처리 방법

슬라이스 단위의 병렬 처리는 H.264/AVC의 부호화 단위인 슬라이스를 이용하여 구현 할 수 있다. 앞서 소개한 <그림 4>와 같이, 한 프레임의 영상을 일정 수의 슬라이스로 분할하고, 각 분할된 슬라이스에 대하여 각 코어에서 병렬 처리하면 된다. 마지막으로 매크로블록 단위의 병렬 처리 방법은 한 프레임에서 동시에 부호화가 가능한 매크로블록에 대하여 병렬 처리를 구현 할 수 있다. 각 매크로블록에 대하여 부호화 순서를 정의하고, 각 코어에서 이를 나누어서 수행하면 된다. <그림 6>은 매크로블록 단위의 병렬 처리 방법을 보여준다. 현재 부호화 하려는 매크로블록에 대하여 왼쪽 매크로블록과 위쪽 매크로블록이 이미 부호화되었다면 그 매크로블록은 부호화가 가능하다. <그림 6>에서 매크로블록에 있는 번호는 부호화 순서를 나타낸다.

이제까지 프레임, 슬라이스, 매크로블록 단위에 대한 병렬 처리 구현 방법에 대하여 살펴보았다. 각각의 방법에 대하여 부호화 성능, 병렬 처리의 효율, 지연, 확장성 등의 관점에서 성능을 비교하여 보자.

일반적으로 병렬 처리의 효율을 높이기 위해서는 공유 메모리의 접근과 코어간의 동기화 신호를 최소화해야 한다. 프레임 단위의 병렬 처리는 각 코어에서 별도의 메모리를 사용하며, 동기화 신호도 프레임 단위로 사용하기 때문에 다른 방법에 비하여 병렬 처리 효율이 가장 높다. 반면, 매크로블록 단위의 병렬 처리는 코어들 간의 빈번한 동기화가 필요하기 때문에 상대적으로 병렬 처리 효율이 떨어진다.



〈그림 6〉 매크로블록 단위의 병렬 처리 방법

비디오 코덱에서는 병렬 처리 효율뿐만 아니라 부호화 효율도 고려해야 한다. 프레임 단위의 병렬 처리에 따른 부호화 효율은 직관적으로 말하기 쉽지 않다. 프레임 단위의 병렬 처리를 위하여 일반적인 부호화 구조인 IPPP 구조에서 IBBP 구조로 변경하기 때문이다. 즉, 다른 부호화 조건을 가지는 환경임으로 부호화 효율을 객관적으로 비교할 수 없다. 일반적으로는 B 프레임이 P 프레임보다 부호화 효율이 높지만, 그 만큼 계산량은 증가하게 된다. 이는 병렬 처리를 위하여 계산량을 증가시키는 결과를 초래하는 잠재적인 문제점을 안고 있다. 슬라이스 단위의 병렬 처리 방법은 한 프레임을 슬라이스 없이 부호화 하는 방법에 비하여 부호화 성능 저하가 발생한다. H.264/ AVC 슬라이스 부호화는 슬라이스 경계에서 주변 블록에 대한 부호화 정보를 사용하지 않기 때문에 부호화 성능이 떨어지게 되는 것이다. 마지막으로 매크로블록 단위의 병렬 처리 부호화 성능은 주변 매크로블록에 대한 부호화 정보를 모두 사용하는 구조를 사용하기 때문에 부호화 성능이 거의 변하지 않는다.

다음으로 코어 개수의 증가에 따른 확장성에 대하여 알아보자. 프레임 단위의 병렬 처리에서는 B 프레임의 수를 조절하여 확장성을 고려할 수 있다. 하지만 B 프레임의 수는 영상의 지연에 영향을 주기 때문에 실시간 응용 프로그램에서는 B 프레임의 수와 원하는 병렬 처리 성능에 대

한 상관관계를 잘 파악해야 한다. 슬라이스 단위에서의 병렬 처리 확장은 한 프레임에서 더 많은 슬라이스 분할을 통하여 달성할 수 있다. 그리고 매크로블록 단위에서는 별 다른 처리 없이 코어 수에 맞게 가능한 매크로블록들을 각 코어에 나누어주는 작업 분배만 해주면 된다. 슬라이스나 매크로블록 단위의 처리는 한 프레임 내에서 병렬 처리를 수행하기 때문에 지연은 발생되지 않는다.

마지막으로 작업 분배에 대해서 이야기 해보자. 작업 분배는 각 코어간의 수행 시간을 공평하게 하자는 것이다. 병렬 처리 성능과는 반대로 작업 분배는 한 번에 처리하는 데이터의 단위가 적을수록 좋은 성능을 보인다. 프레임이나 슬라이스 단위의 데이터 분할은 영상의 복잡도에 따라서 코어간의 처리 속도 불균형이 발생될 수 있다. 반면, 매크로블록 단위는 각 코어에서 분할되어 처리되는 데이터의 단위가 작기 때문에 작업 분배가 원활히 이루어지며 코어간의 처리 속도가 비슷하게 나타날 것이다.

VI. 결 론

본 고에서는 최근 대두되고 있는 멀티 코어 환경에서 비디오 코덱을 병렬 처리하는 방법에 대하여 살펴보았다. 일반적인 병렬 처리 프로그래밍에서 최적의 성능을 얻기 위해서는 코어간의 균등한 작업 분배, 코어 개수, 플랫폼의 변화에 따른 효과적인 처리가 중요한 이슈가 된다. 또한, 메모리의 사용이나 코어간의 데이터 통신에 대한 제약을 고려해야 할 것이다. 하지만, 비디오 코덱에서는 단순히 병렬 처리 성능뿐만 아니라 압축 효율, 지연과 같은 비디오 코덱이 가지는 특성들도 함께 고려되어야 한다. 만약 독자가 멀티 코어 환경에서 병렬 코덱을 개발할 때, 본 고에서 소개한 여러 가지 요인들을 통합적으로 고려한다면 만족할만한 병렬 처리 코덱을 얻을 수 있을 것이라 생각된다.

참고문헌

[1] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264 /AVC video coding standard," IEEE Trans. on Circuits and Systems for Video Technology, Vol.13, No.7, pp.560-576, 2007.

[2] 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.

[3] Cor Meenderinck, Arnaldo Azevedo, Mauricio Alvarez, Ben Juurlink, and Alex Ramirez, "Parallel scalability of H.264," Proceedings of the 1st Workshop on Programmability Issues for Multi-Core Computers, 2008.

[4] Y.-K. Chen, T. X, S. Ge, and G. M, "Towards efficient multi-level threading of h.264 encoder on intel hyper-threading architectures," in 18th Int. Parallel and Distributed Processing Symposium, p.63, Apr., 2004.

[5] Zhao, Z. and Ping Liang, "A highly efficient parallel algorithm for H.264 video encoder," Proc. of the 2006 IEEE International Conference on Acoustics,

Speech and Signal Processing, Vol.5, 14-19, pp.489-492, May, 2006.

[6] Jonghan Park and Soonhoi Ha, "Performance analysis of parallel execution of H.264 encoder on the cell processor," in Proc. International Fifth Workshop on Embedded Systems for Real Time Multimedia, pp.4-5, Oct., 2007.

[7] Yang Shu-Sian, Wang Sung-Wen, Chen Hong-ming, and Wu Ja Ling, "A parallelism encoding framework for the temporal scalability of H.264/AVC scalable extension," Ninth IEEE International Symposium on Multimedia Workshop 2007, pp.397-400, Dec., 2007.

[8] Jike Chong, Nadathur Satish, Bryan Catanzaro, Kaushik Ravindran, and Kurt Keutzer, "Efficient parallelization of H.264 decoding with macro block level scheduling," IEEE International Conference on Multimedia and EXPO, pp.1874-1877, July, 2007.

[9] Roitzsch, M., "Slice balancing H.264 video encoding for improved scalability of multicore decoding," In Proceedings of the 7th ACM & IEEE International Conference on Embedded Software, pp.269-278, 2007.

[10] A. Rodriguez, A. Gonzalez, and M. P. Malumbres, "Hierarchical parallelization of an H.264/AVC video encoder," International Symposium on Parallel Computing in Electrical Engineering, pp.363-368, Sept., 2006.

저자소개

심 동 규

1993년 2월 서강대학교 전자공학 학사
1995년 2월 서강대학교 전자공학 석사
1999년 2월 서강대학교 전자공학 박사
1999년 3월 ~ 2000년 9월 현대전자 선임연구원
2000년 9월 ~ 2002년 3월 바로비젼 선임연구원
2002년 4월 ~ 2005년 2월 Univ. of Washington
2005년 3월 ~ 현재 광운대학교 부교수

주관심 분야 : 영상압축, 영상신호처리, 컴퓨터비젼

남 정 학

2006년 2월 광운대학교 컴퓨터공학 학사
2008년 2월 광운대학교 컴퓨터공학 석사
2009년 ~ 현재 광운대학교 컴퓨터공학 박사과정

주관심 분야 : 영상압축, 멀티프로세서