네트워크 프로세서를 위한 다중 쓰레드 스케줄링

임 강 빈[†]·박 준 구[††]·정 기 현[†††]·최 경 희[††††]

요 약

본 논문은 다중 프로세서(Multiprocessor) 기반 다중 쓰레드(Multithreaded) 구조의 네트워크 프로세서를 이용한 패킷 처리 시스템에서 패킷을 보다 고속으로 처리하기 위한 쓰레드 스케줄링 기법을 제안한다. 이를 위하여 스케줄링과 관련한 인자를 실험을 통하여 얻고, 패킷 내용 및 다중 쓰레드 아키텍처를 표현하는 인자를 포함하도록 설계하였다. 시뮬레이터를 이용한 실험을 통하여 제안된 스케줄링 기법이 제공하는 처리율 및 부하 분산 정도가 다른 스케줄링 기법과 비교하여 효율적임을 증명하였다.

Multi-thread Scheduling for the Network Processor

Kangbin Yim[†]·Junku Park[††]·Gihyun Jung[†††]·Kyunghee Choi[††††]

ABSTRACT

In this paper, we propose a thread scheduling algorithm for faster packet processing on the network processors with multithreaded multiprocessor architecture. To implement the proposed algorithm, we derived several basic parameters related to the thread scheduling and included a new parameter representing the packet contents and the multithreaded architecture. Through the empirical study using a simulator, we proved the proposed scheduling algorithm provides better throughput and load balancing compared to the general thread scheduling algorithm.

키워드 : 네트워크 프로세서(Network Processor), 다중 프로세서(Multi-processor), 다중 쓰레드(Multithreaded), 스케줄링(Scheduling), 쓰레드 스케줄링(Thread Scheduling)

1. 서 론

최근 인터넷 망의 고속화와 함께 사용자에게 다양한 서비스를 제공하기 위한 새로운 네트워크 프로토콜 및 어플리케이션들이 급속히 증가하고 있으며 이에 대응하여 많은 네트워크 장비들이 고속화되고 적응화될 것이 요구되고 있다. 네트워크 장비들이 고속화되기 위하여는 ASIC 기반의 하드웨어에 의한 트래픽 처리가 필수적이나 이는 다양하게 변화하는 프로토콜 환경에서는 적응성의 결여라는 문제가 따른다. 그러므로, 대개의 시스템은 프로그램의 변경이 용이한 범용 프로세서를 이용하고 있다. 그러나, 기가비트 네트워크의 도래와 함께 범용 프로세서를 이용한 트래픽 처리는 한계를 드러냈다. 따라서, 새로운 서비스에 대한 유연성을 지원하면서 고속 패킷 처리가 가능한 네트워크 프로세서라 불리는 특화 프로세서에 대한 연구가 진행되고 있다[1-3, 10].

※ 본 논문은 과기부 국가지정연구실사업 및 정통부 IT분야 해외교수초빙 국제공동연구사업의 지원으로 연구되었음.
† 정 회 원 : 순천향대학교 정보보호학과 교수
†† 준 회 원 : 디지털스트림테크놀로지 연구원
††† 정 회 원 : 아주대학교 전자공학부 교수
†††† 정 회 원 : 아주대학교 정보 및 컴퓨터공학부 교수
논문접수 : 2004년 1월 14일, 심사완료 : 2004년 4월 22일

일반적으로 네트워크 프로세서는 인터넷 망에서의 패킷 처리 기능 중 최하위 부분은 하드웨어적으로 구현하고, 나머지는 패킷 처리 프로세서가 담당하도록 구성되어 있다. 패킷 처리 프로세서는 고속의 RISC 프로세서로서, 프로그램을 새로이 작성하거나 변경하여 새로운 서비스를 지원할 수 있다[10]. 또한, 고속 패킷 처리를 위하여 하나의 네트워크 프로세서에서 다수의 패킷 처리 프로세서를 지원하며, 각 패킷 처리 프로세서는 하드웨어 수준의 문맥 교환(Context Switching)이 가능한 다중 쓰레드 구조를 가지고 있다. 하나의 네트워크 프로세서는 패킷 처리 프로세서의 수와 각 패킷 처리 프로세서 당 쓰레드 수의 곱에 해당하는 독립적인 쓰레드를 지원한다. 이러한 네트워크 프로세서에서 패킷 처리와 같은 서비스를 설계할 경우, 각 쓰레드의 기능을 결정하여 배치하는 방법에 따라 그 처리율이 달라지며, 이는 입력되는 패킷을 처리하기 위한 쓰레드를 선택하기 위한 쓰레드 스케줄링 방안에 매우 의존적이다.

다중 프로세서 환경에서 작업을 각 프로세서에 전달하여 효율적으로 처리하도록 하는 스케줄링 방안에 대한 연구는 다각도로 진행되어 왔다. 병렬 Loop Scheduling[5]는 Loop가 있는 작업을 다수의 프로세서에 분산하는 방안이다. 이

러한 스케줄링 방안은 정적 스케줄링 기법과 동적 스케줄링 기법으로 구분되는데, 정적 스케줄링 기법은 오버헤드가 매우 적다는 장점을 가지나 부하 분산의 실현이 어렵고, 동적 스케줄링 기법은 오버헤드는 커질 수 있으나 부하 분산의 실현이 가능하다. 따라서, 오버헤드에 따른 지연과 부하 분산에 따른 이득을 고려한 동적 스케줄링 방안들이 제안되어 왔다. 이러한 동적 스케줄링 방안은 경합을 최소화하기 위하여, 한번에 많은 양의 작업을 할당하는 방안이 제안되었고[8, 9], 지역성을 최대화하는 방안이 연구되었다[5]. MSS[7]는 이러한 동적 Loop 스케줄링을 구현함에 있어서, 각 프로세서의 구조를 Multithreaded Architecture로 확장하여 프로세서 당 스케줄러를 다시 두는 구조로 메모리 대기 시간을 최소화 하고자 하였다.

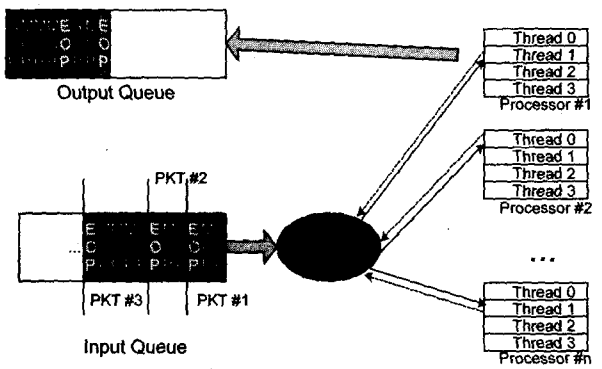
캐시가 있는 네트워크 프로세서에서 캐시에 대한 지역성을 고려한 패킷 스케줄링 방안[4]은 캐시의 지역성에 의한 성능을 기대한 방안이다. 이러한 방안은 캐시가 있는 네트워크 프로세서에서 패킷을 적은 오버헤드로 스케줄링 할 수 있는 장점을 가진다. Pfairness[6]에서는 각 패킷 또는 세션에 대한 처리 시간의 균등함을 목적으로 하는 스케줄링 방안을 제안하였다. 이는 GPS(Generalized Processor Sharing)에 근접한 패킷 스케줄링 방안을 제안하는 것이 목적이며 최대의 처리율보다는 분류된 패킷들에 대하여 균등한 처리시간을 분배하는 것이 목적이다.

상기의 방안 이외에도 많은 문헌에서 다중 프로세서를 위한 스케줄링에 대한 꾸준한 연구를 찾을 수 있으나 대부분 일반적인 작업의 경우만을 고려한 것으로 고속 네트워크 프로세서를 위한 패킷 처리의 특성을 고려하고 있지 않다. 특히, 패킷 처리와 관련하여 네트워크 프로세서에서 일반화되어 있는 다중 쓰레드 구조를 효율적으로 이용하기 위한 스케줄링 방안은 전무한 실정이다. 따라서, 본 논문에서는 네트워크 프로세서가 가지는 다중 프로세서 기반의 다중 쓰레드 구조의 특징과 패킷 처리라는 작업의 특성을 고려하여 트래픽 처리율을 최대화하기 위한 쓰레드 스케줄링 방안을 제안하고자 한다.

제안한 방안은 네트워크 프로세서에서 패킷을 처리하기 위한 부하를 균등하게 분산하고, 각각의 프로세서 내에서는 쓰레드의 상태를 최대한 고려하여 높은 처리율을 제공한다. 제안한 방안의 성능을 검증하기 위한 플랫폼으로서 네트워크 프로세서의 일반적 특성을 고루 갖추고 저가 시스템 구현이 가능한 인텔의 IXP1200을 선택하였으며 실험은 IXP1200을 이용한 실제의 네트워크 시스템 및 시뮬레이터 상에서 동시에 이루어졌다.

본 논문의 구성은 다음과 같다. 제2장에서는 네트워크 프로세서와 같은 다중 쓰레드를 지원하는 시스템에서 패킷 처리 문제와 관련한 고려 사항을 논한다. 제3장에서는 본 논문에서 제안하는 네트워크 프로세서에서의 다중 쓰레드 스케줄링 방안에 대해 기술하고, 제4장에서는 제안한 스케줄러의 구현 및 실험을 통한 성능 평가와 그 결과를 분석하며 제5장에서 결론으로 논문을 마친다.

2. 네트워크 프로세서에서의 패킷 처리



(그림 1) 다중 쓰레드에 의한 패킷 처리

다중 프로세서 기반 다중 쓰레드 구조의 네트워크 프로세서에서 사용되는 패킷 처리 소프트웨어는 그 물리적 구조에 적합한 형태로 설계되어야 한다. 단일 프로세서 환경에서의 소프트웨어를 다중 프로세서 환경에 그대로 적용할 경우 성능 개선 정도가 매우 낮은 것으로 나타났으며[2], 특히 네트워크 프로세서에서는 처리 시간 제한, 프로세스간 통신, 프로세서 레지스터 제한 등의 문제를 신중히 고려해야 한다. 한편, 다중 쓰레드 구조에서 패킷의 처리율이 최대가 되기 위하여는 부하가 각 쓰레드에 고루 분산되어야 하며 이는 각 쓰레드의 부하가 스케줄러를 위한 인자가 되어야 함을 의미하므로 부하를 효과적으로 표현하는 것이 매우 중요하다. 각 쓰레드가 패킷을 처리할 때 가지는 부하는 패킷의 크기와 종류에 의존적이므로, 네트워크 프로세서에서의 부하를 표현함에 있어서는 이를 반드시 고려하여야 한다.

네트워크로부터 입력되는 패킷이 다중 프로세서 기반의 다중 쓰레드 구조를 가지는 네트워크 프로세서에서 처리되는 동안의 패킷의 흐름을 간략히 그리면 (그림 1)과 같다. 패킷은 물리 계층의 장치에 의해 입력 큐에 입력되는데 이 때 입력되는 패킷의 크기는 데이터링크 계층의 프로토콜 종류 및 그 데이터그램의 크기에 따라 달라지며 프로세서 내의 각 쓰레드에 의하여 처리되는 기본 단위와는 큰 관련이 없다. 다만, 일반적인 네트워크 프로세서에서 쓰레드에 의하여 처리되는 데이터의 기본 단위는 데이터링크 계층의 데이터그램 크기보다는 매우 작고 고정된 크기의 구조로 되어 있다. 이러한 처리의 기본 단위를 서로 다른 네트워크 프로세서들이 다양한 이름으로 부르고 있으나 본 논문에서

는 MAC 패킷이라 부르기로 한다.

데이터링크 계층 프로토콜에 의하여 정의되는 하나의 패킷은 여러 개의 MAC 패킷으로 구성될 수 있고, 이들은 패킷에서의 위치에 따라 다음의 세 가지로 분류될 수 있다. 첫째로, SOP(Start Of Packet)는 패킷의 가장 처음의 위치에 있는 MAC 패킷이다. 이는 일반적으로 패킷의 헤더를 포함하게 되므로 대부분의 패킷 관련 처리가 이 부분을 상대로 수행된다. 따라서, SOP에 해당하는 MAC 패킷을 처리하는 쓰레드는 여타 부분을 처리하는 쓰레드에 비하여 더 많은 부하를 가지게 된다. 둘째로, MOP(Middle Of Packet)는 중간에 위치하는 MAC 패킷이다. 이는 내용 기반의 검색 등과 같은 특수한 경우를 제외하고는 단순히 패킷을 버퍼에 저장하는 기능만 수행하게 되므로 가장 적은 부하를 가지게 된다. 마지막으로, EOP(End Of Packet)는 패킷의 마지막에 위치하는 MAC 패킷이며, 패킷에 대한 결정 사항, 즉 소거 또는 전달 등을 적용하게 된다. 예를 들어, 패킷에 대한 폐기 및 전달 여부를 SOP에서 결정한다면, 이를 실제로 폐기 및 전달하는 곳은 EOP에서 수행하게 된다. 그러므로, EOP는 MOP에 비해서 많은 부하를 가지지만, SOP에 비해서는 적은 부하를 가지게 된다. 이와 같이, MAC 패킷에 대한 처리 부하가 패킷에서의 MAC 패킷의 위치에 따라 변화하므로 이를 스케줄링을 위한 부하의 정도로 표현할 수 있다[13].

그림에서의 스케줄러는 입력 큐에 존재하는 패킷의 일부 즉, 각각의 MAC 패킷에 대한 처리 작업을 위하여 주어진 스케줄링 전략에 의거하여 하나의 쓰레드를 선택한다. 선택된 쓰레드는 자신이 담당하는 MAC 패킷이 속한 패킷에 대하여 경로 결정이나 패킷 여과 등과 같은 작업을 처리함에 있어서 일조하며 여러 쓰레드를 거쳐 처리된 패킷은 다시 출력 큐로 전달되고, 이는 다시 출력 인터페이스를 통하여 망에 전송된다. 이러한 과정에서 쓰레드 스케줄링을 효율적으로 수행하기 위하여 각 쓰레드들은 자신이 처리하는 MAC 패킷의 종류 등과 같은 필요한 정보를 스케줄러에게 전달하고, 이 정보를 기반으로 스케줄러는 최대 실행 시간을 최소화할 수 있는 쓰레드를 선택한다.

3. 고속 패킷 처리를 위한 새로운 쓰레드 스케줄링 방안

다중 프로세서 기반의 다중 쓰레드 환경에서 효율적인 패킷 처리를 위한 스케줄러를 설계함에 있어 고려하여야 할 사항에는 다음의 것들이 있다. 첫째로, 다중 프로세서 환경에서 최대의 처리율을 얻기 위해서는 각 프로세서의 프로세서 활용률이 최대가 되어야 하므로 각 프로세서들의 연산량이 동일한 네트워크 프로세서에서는 균등한 부하 분산이 실현되는 스케줄링 방안이 요구된다. 둘째로, 동일한 자원에 대한 여러 프로세서들의 경합에 의하여 프로세서가 멈추거나 다른 쓰레드로 교체되어 발생하는 지연이 고려되어야 한다. 다수의 프로세서 또는 쓰레드의 경합에 의하여 다른 쓰레드로 교체될 경우 하드웨어 수준의 쓰레드 스케줄링이 지원되더라도 파이프라인이 깨지고 모든 쓰레드가 기다리는 상태가 될 수 있으므로 이를 최소화하는 방안이 필요하다. 따라서, 프로세서 간 경합에 의한 지연이 최소화되는 스케줄링 방안이 요구된다. 셋째로, 다중 쓰레드 환경이므로 같은 프로세서 내의 서로 다른 쓰레드들은 각각의 부하량이 서로에게 미치는 영향을 고려하여야 한다. 즉, 작업을 쓰레드에 전달할 경우 해당 프로세서의 다른 쓰레드가 가지는 부하를 모두 고려하여 전달하여야 한다. 그리고, 스케줄링을 하는 쓰레드의 경우 다른 쓰레드의 부하에 의해 스케줄링 결정이 늦어져 처리율이 떨어지지 않도록 고려되어야 한다. 따라서, 다중 쓰레드 환경에서 서로 간의 영향을 인지하는 스케줄링 방안이 요구된다.

상기한 바와 같이 다중의 쓰레드 중에서 다음 패킷을 처리할 쓰레드를 결정하기 위한 인자로서 각 쓰레드의 상태 및 부하를 사용할 수 있다. 이 때, 쓰레드의 현재 상태 및 부하를 상세히 표현할수록 스케줄러가 이를 이용하여 정확한 결정을 할 수 있으나, 추가적인 오버헤드가 발생한다. 따라서, 상태 및 부하의 표현과 전달을 효과적으로 수행할 수 있는 적절한 방안이 요구된다. 본 논문에서는 새로운 패킷을 처리하기 위한 최적의 쓰레드를 선택하는 과정에서 각 쓰레드의 상태 및 부하 정보를 얻는 방법에 따라 정적 스케줄링(Static Scheduling), 최소 부하 스케줄링(Minimum Load Scheduling), 최소 부하 예상 스케줄링(Predictive Minimum Load Scheduling)의 3가지 방안을 고려하였다.

고려되는 스케줄링 방안들은 패킷이 입력된 순서대로 처리하는 First-Come-First-Serve(FCFS) 방식을 사용하는 것으로 가정하였으며, 이 중 정적 스케줄링 방안은 스케줄링 오버헤드가 최소가 되지만, 부하 분산의 편차는 최대가 될 수 있기 때문에 제안되는 스케줄링 방안과 비교를 위해 사용된다. 그리고, 제안하는 방안들은 최소의 부하량을 가지는 쓰레드에게 다음 MAC 패킷을 처리하도록 전달하는 방법을 사용하며, 상세한 내용은 다음과 같다.

- 정적 스케줄링(Static Scheduling) : 가장 기본적인 스케줄링 방안으로 MAC 패킷을 처리하는 쓰레드의 순서가 미리 정해져 변경되지 않는다. 따라서, 스케줄링에 대한 오버헤드가 없다는 장점을 가진다. 그러나, 다른 쓰레드들이 유휴상태에 있더라도, 다음 순서의 쓰레드가 처리 중에 있는 경우 전체적인 처리가 늦어지며, 한 프로세서에 부하가 집중되는 경우에도 처리가 늦어진다. 이러한 정적 스케줄링 방안은 MAC 패킷에 대한 부하가 균일하

거나, MAC 패킷 처리 시간이 다음 순서가 돌아오는 시간보다 적은 경우 유효한 방안이 될 수 있으나, MAC 패킷에 대한 부하는 균일하지 않으며, 복잡한 서비스의 경우 긴 헤더 처리 시간이 필요하다.

- 최소 부하 스케줄링(Minimum Load Scheduling) : MAC 패킷을 처리하는 쓰레드 중에서 최소의 부하를 가지는 쓰레드에게 전달하는 방안을 최소 부하 스케줄링이라 정의하였다. 각 쓰레드의 부하량에 대한 정보는 각 쓰레드의 유휴 여부와 처리되고 있는 MAC 패킷의 속성으로 얻는다. 각 쓰레드의 부하량으로 쓰레드들이 속한 프로세서의 부하량을 얻으며, 이렇게 얻은 각 프로세서의 부하량으로 가장 적은 부하량을 가진 프로세서를 선택한다. 그 다음, 선택한 프로세서에서 유휴이거나, 최소의 부하량을 가진 쓰레드가 다음 MAC 패킷을 처리하도록 전달한다. 즉, 최소 부하량의 프로세서를 선택하고, 선택한 프로세서에서 최소 부하량의 쓰레드를 선택하여 MAC 패킷을 전달한다. 이러한 방법으로 최소 부하의 쓰레드를 선택하도록 하여 부하를 균등하게 분산할 수 있으며, 최대의 성능을 얻을 수 있다. 이 방안의 단점은 현재 쓰레드가 속한 프로세서와 동일한 프로세서 내의 쓰레드에게 다음 작업이 전달되는 점이다. 이렇게 동일 프로세서 내의 쓰레드에게 전달되면, 같은 작업이 1개 프로세서 내에서 중첩되어 발생할 확률이 증가하여 오버헤드가 커진다. 스케줄러의 정의는 다음과 같으며, 기호는 <표 1>에 정의하고 있다. 이 때, 각 쓰레드의 부하량 $L_u(u)$는 앞서 설명한 바와 같이 각 쓰레드에서 현재 처리되고 있는 MAC 패킷의 속성을 통하여 얻는다.

$$S(t, \Pi_{at}) = u_j^i \ where$$

$$u_j^i = argmin\{L_u(u_m) : u_m \in \pi_i\}$$

$$\pi_i = \begin{cases} argmin\{L_\pi(\pi_n) : \pi_n \subset \Pi\} & when \ \Pi_{at} = \{\emptyset\} \\ argmin\{L_\pi(\pi_n) : \pi_n \subset \Pi_{at}\} & when \ \Pi_{at} \neq \{\emptyset\} \end{cases}$$

$$L_\pi(\pi_i) = \sum_{j=0}^{N} L_u(u_j^i)$$

- 최소 부하 예상 스케줄링(Predictive Minimum Load Scheduling) : 다음 작업을 최소 부하를 가지는 쓰레드에게 전달하되, 동일한 프로세서 내의 쓰레드에게 전달하게 될 기회를 최소화하는 방안을 최소 부하 예상 스케줄링이라 정의하였다. 이 방안은 기본적으로 최소 부하 스케줄링 방안과 유사하지만, 프로세서들의 부하량이 일정 값 이내로 계산되었을 경우, 같은 프로세서 내의 쓰레드에게 전달되지 않도록 한다. 즉, 현재 사용되는 프로세서와 같은 프로세서로 할당되는 오버헤드를 부하량으로 포함하여 벌칙을 주었을 때, 최소 부하가 되는 프로세서에 할당하는 방안이다. 이는 같은 프로세서에서 유사한 작업이 비슷한 시기에 중첩되어 발생하는 오버헤드를 방지하기 위함이다. 이 오버헤드에 대한 부하 상수 는 구현에 따라 달라지며, 실험을 통하여 MOP(Middle Of Packet)를 처리하는 부하로 선택하였다. 최소 부하 예상 스케줄링 방안은 다음과 같이 정의되었으며, 최소 부하 스케줄링과 비교하여 프로세서 당 부하량 $L_u(\pi_i)$에 대한 내용만이 다르다.

$$S(t, \Pi_{at}) = u_j^i \ where$$

$$u_j^i = argmin\{L_u(u_m) : u_m \in \pi_i\}$$

$$\pi_i = \begin{cases} argmin\{L_\pi(\pi_n) : \pi_n \subset \Pi\} & when \ \Pi_{at} = \{\emptyset\} \\ argmin\{L_\pi(\pi_n) : \pi_n \subset \Pi\} & when \ \Pi_{at} = \{\emptyset\} \end{cases}$$

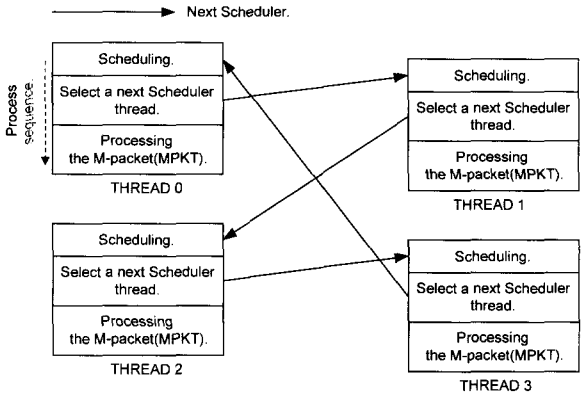
$$L_\pi(\pi_i) = \sum_{j=0}^{N} L_u(u_j^i) + d(\pi_i) \times \alpha$$

〈표 1〉 쓰레드 스케줄러 인자

구 분	기 호	설 명
프로세서 π	Π	전체 프로세서 집합. ($\bigcup_{i=0}^{N} \pi_i \in \Pi$)
	Π_{at}	시간 t에서 유휴 쓰레드를 1개 이상 가지는 프로세서의 집합.
	π_i	i번째 프로세서.
	$L_\pi(\pi_i)$	i번째 프로세서의 부하량.
	$d(\pi_i)$	직전에 스케줄러를 실행한 프로세서가 π_i 인지의 여부.
쓰레드 u	u_j^i	i번째 프로세서에 속하는 j번째 쓰레드. ($\bigcup_{j=0}^{N} u_j^i \in \pi_i$)
	$L_u(u_j^i)$	i번째 프로세서의 j번째 쓰레드 u_j^i의 부하량.
스케줄러	α	동일 프로세서에 할당될 때 발생하는 오버헤드의 부하량.

이렇게 스케줄링 전략의 성능을 최대로 하기 위해서는 인자의 효율적인 전달에 의한 방안뿐만 아니라, 최대의 성능을 낼 수 있는 프로세서 및 쓰레드에서 스케줄러가 실행되어야 한다. 그러나, 앞서 설명한 바와 같이 별도의 쓰레드 또는 프로세서에 스케줄러를 할당하는 것은 비용이 크므로, 스케줄러는 하나의 쓰레드에서 패킷을 처리하는 기능과 스케줄러가 함께 수행되는 자가 스케줄러(Self Scheduler)로 설계하여야 한다. (그림 2)에 이러한 자가 스케줄링에 대한 예제를 나타내었다. 그림에서 각 쓰레드에서 스케줄링 기능, 다음 스케줄러를 선택하는 기능, MAC 패킷을 처리하는 기능으로 분리하여, 스케줄러 기능과 MAC 패킷 처리 기능이 함께 수행되는 내용을 볼 수 있다. 스케줄링 기능을 수행한 뒤, 다음 스케줄러를 선택하면 이를 받은 쓰레드가 다시 스케줄링을 실행하는 순으로 수행된다. 이 때, 스케줄러는 가장 빨리 처리될 수 있는 쓰레드에서 처리되

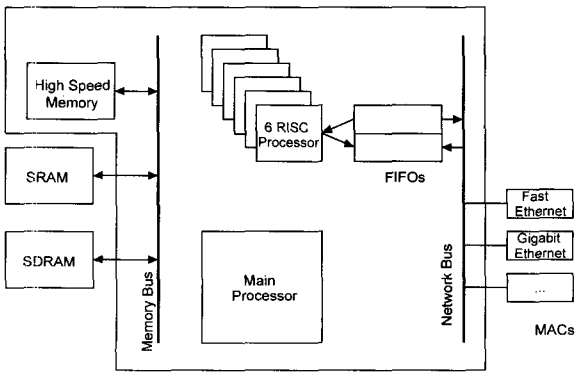
어야 하므로, 다음 MAC 패킷을 처리할 쓰레드를 선택하는 조건과 동일하다. 따라서, 스케줄러가 선택하는 쓰레드가 다음 스케줄러가 실행될 쓰레드와 동일하므로, 그림과 같이 세 가지 기능이 순차적으로 배치된다. 본 논문에서 고려되거나 제안되는 방안들은 모두 이와 같은 자가 스케줄링의 형태로 배치하는 것으로 가정한다.



(그림 2) 패킷 처리 쓰레드에서의 자가 스케줄링

4. 스케줄러의 구현 및 성능 평가

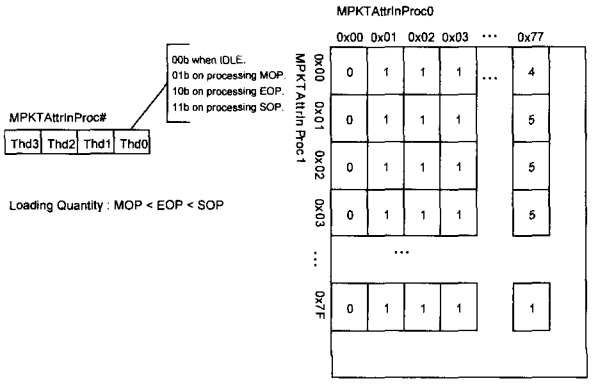
제안한 방안은 이미 언급된 네트워크 프로세서의 특징을 고루 갖춘 네트워크 프로세서인 IXP1200[13] 위에서 구현되었으며 그 구조를 (그림 3)에 나타내었다. IXP1200은 전체적인 관리를 위한 주 프로세서를 가지며 이를 통하여 시스템의 메모리 버스가 제공된다. 또한 공유 메모리를 통하여 서로 연결된 여섯 개의 칩 레벨 RISC 다중 프로세서를 가지고 있으며 각각의 프로세서는 네 개의 하드웨어 기반 쓰레드로 이루어져 있다. 그리고 시스템의 메모리 버스와 구분되어 별개의 네트워크 버스를 제공하여 통신을 위한 인터페이스를 고려하고 있다. 본 논문에서는 각 기가 비트 인터페이스에 대해서 두 개의 프로세서가 패킷을 처리하도록 설계하였다.



(그림 3) 네트워크 프로세서의 구조 예제

최소 부하 스케줄링 방안은 앞서 설명한 바와 같이 네트워크 프로세서의 하드웨어적인 특징을 고려한 설계가 필요하다. 첫째, 프로세서 및 쓰레드간의 경합은 부하를 유지하는 레지스터와 같은 자원에 접근할 때 발생하며, 이 때 접근하는 경합의 수위, 즉 경합하는 쓰레드의 수를 일정 수준으로 유지하는 것이 필요하다. 이는 자가 스케줄링 방안을 사용함으로써, 일정 시점에 접근할 수 있는 쓰레드는 프로세서 당 1개로 제한하였다. 둘째, 빈번하게 사용되는 자료구조 중 순서에 상관없이 동시에 접근하여도 무관한 자료구조는 별도의 메모리로 분리시켜 유지하도록 설계하는 것이 요구된다. 이는 쓰레드의 상태는 내부 메모리에 쓰레드의 부하는 전역 레지스터에 유지하여 분리하였다. 마지막으로, 스케줄링 코드들은 패킷 처리의 임계 경로(critical path)에 있으므로, 스케줄링 결정에 따른 오버헤드의 최소화가 요구된다. 따라서, 계산에 따른 내용은 미리 계산된 테이블을 사용함으로써 오버헤드를 최소화하였다.

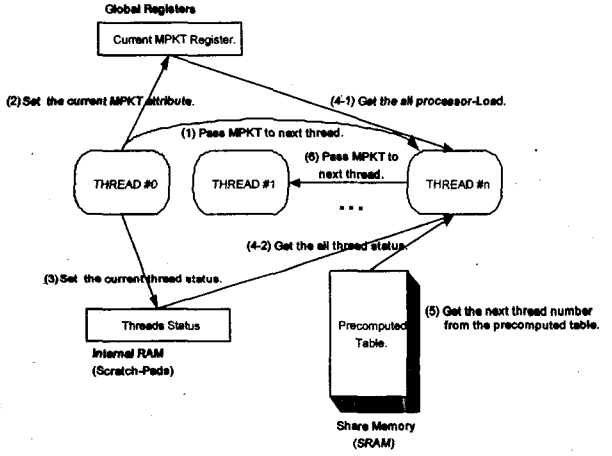
미리 계산된 테이블은 제안될 스케줄링 방안의 수식에 의해 쓰레드의 번호를 직접 반환하도록 구성되었으며, 그 내용은 (그림 4)와 같다. 그림은 2개의 프로세서를 기준으로 프로세서 당 4개의 쓰레드가 유지되는 환경이며, MAC 패킷 별 부하량은 SOP, EOP, MOP 순이라 가정한 예제로 표시하였다. 계산된 테이블의 동작은 다음과 같다. 우선, 현재 처리되고 있는 MAC 패킷의 속성에 따라 부하량을 얻기 위한 자료구조(MPKTAttrInProc#)를 각 프로세서마다 유지하고, 이 값을 계산된 테이블의 인덱스 값으로 얻어낸다. 테이블의 각 필드에는 인덱스 값으로 얻어진 부하량으로 최소의 부하를 가지는 쓰레드 번호를 유지한다. 따라서, MAC 패킷 속성을 인덱스로 하여 읽은 값이 다음 MAC 패킷을 처리할 쓰레드가 된다. 그림에서와 같이 모두 IDLE인 경우 0번의 쓰레드에게 전달하고, 0번 쓰레드가 MAC패킷을 처리하는 경우 1번에 전달한다. 그리고, 0×77 및 0×7F의 값을 가지는 경우 0번째 프로세서의 부하가 적으므로,



(그림 4) 미리 계산된 스케줄링 테이블 예제

0번째 프로세서의 쓰레드 중 부하가 가장 적은 1번 쓰레드에게 전달한다. 최소 부하 예상 스케줄링의 경우에는 이전 처리된 프로세서를 파악하여야 하므로, 각 프로세서마다 접근하는 테이블을 별도로 유지하여 부하량 계산시 오버헤드를 고려하여 테이블을 작성한다.

최소 부하 스케줄링 방안의 전체 구조는 (그림 5)와 같다. 예제에서는 0번 쓰레드, n번 쓰레드, 1번 쓰레드의 순서로 MAC 패킷 처리 순서를 가진다. 0번 쓰레드에서 n번 쓰레드로 MAC 패킷 처리 순서를 결정하여 전달한다.(1) 그리고, 현재 처리하는 MAC 패킷의 속성을 전역 레지스터에 표시하고(2), 현재 쓰레드의 상태를 내부 메모리에 표시한다.(3) MAC 패킷 처리를 받은 n번 쓰레드는 전역 레지스터와 내부 메모리로부터 프로세서 및 쓰레드의 상태와 처리 중인 MAC 패킷의 속성을 동시에 가져온다.(4-1)(4-2) 이를 이용하여 미리 계산된 테이블에서 다음 쓰레드 번호를 얻고(5), 다음 쓰레드에게 MAC 패킷을 처리하도록 신호를 전달한다.(6)



(그림 5) 최소 부하 스케줄링 방안의 구조

모든 실험은 인텔 사의 네트워크 프로세서의 응용 프로그램으로 제공되는 시뮬레이터를 사용하였다[14]. 스케줄러의 인자로 사용되는 프로세서 및 쓰레드의 부하는 MAC 패킷의 처리 구현에 의존적이며, 사용하는 연산의 종류와 개수에 비례한다. 따라서, 정확한 부하 계산을 위하여 구현된 코드의 연산의 종류와 대기시간 및 MAC 패킷 별 연산의 수를 <표 2>에서 구하고, 이를 이용하여 <표 3>과 같이 MAC 패킷 종류 별 부하를 얻었다. 패킷에 대한 실제 처리는 분류 및 여과를 위한 코드가 구현되어 사용되었다[11]. 코드에 포함되는 연산의 수와 실제 실행되는 연산의 수는 같지 않으므로, 분기 명령이나 조건 명령을 고려하여 측정하였다.

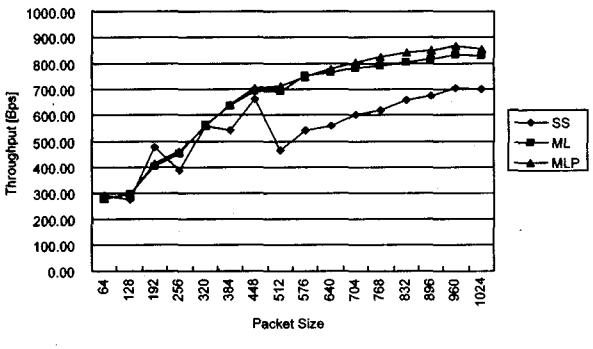
〈표 2〉 연산의 종류 및 수

MAC 패킷 종류	SRAM #	Scratch #	SDRAM to uEng	RFIFO - SDRAM	CSR	RFIFO to uEng	내부 연산#
SOP & ~EOP	7	4	1	1	2	6	254
SOP & EOP	10	5	1	1	3	6	283
~SOP & EOP	5	1	0	1	3	0	92
~SOP & ~EOP	2	0	0	1	2	0	90
Latency [ns]	36	22	21	60	20	22	1

〈표 3〉 MAC 패킷 종류 별 부하

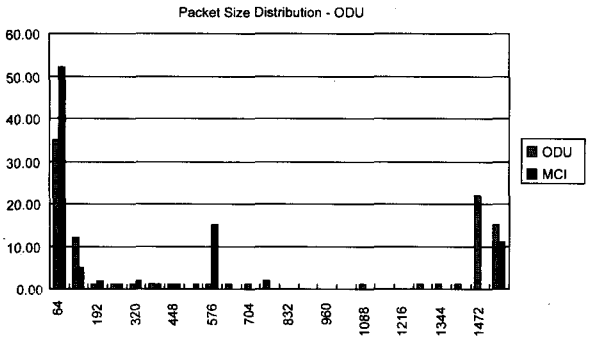
MAC 패킷 종류	부 하
SOP & ~EOP	593
SOP & EOP	743
~SOP & EOP	322
~SOP & ~EOP	172

성능의 측정은 패킷 크기 별 성능 측정과 실제의 망에서 얻은 패킷을 이용한 성능 측정을 수행하였다. 패킷 크기 별 성능 측정은 일정한 크기의 패킷만이 생성되는 환경에서의 처리율을 측정한 것으로 결과는 (그림 6)과 같다. 정적 스케줄링(SS)에 비하여 최소 부하 스케줄링(ML) 및 최소 부하 예상(MLP) 방안이 256바이트 이상의 환경에서 나은 성능을 보이고 있다. 이는 헤더 처리가 많은 256바이트 이하의 패킷에서는 부하의 분산이 정적 스케줄링에서도 제대로 수행되지만, 256바이트 이상에서는 그렇지 않은 것을 의미한다. 그리고, 정적 스케줄링은 프로세서의 수가 2개이므로, 패킷의 크기가 2MAC 패킷 크기의 배수인 경우 헤더 처리가 한 프로세서로 집중되기 때문에 낮은 성능을 보이고 있다.



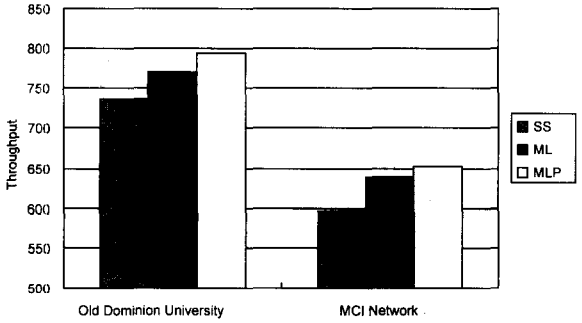
(그림 6) 패킷 크기 별 성능

실제 망의 패킷은 MCI Network 및 Old Dominion University의 Backbone 망에서 1997년 6월 25일 4시에 1분간 얻은 패킷을 이용하여 실험하였다[12]. 해당 패킷의 크기 분포는 (그림 7)과 같다. 64바이트 패킷이 가장 많이 분포하였으나, ODU의 경우에는 1500바이트가 많이 나타났고, MCI의 경우에는 576바이트 및 1500바이트가 많이 나타났다.



(그림 7) 패킷의 크기 분포

MCI Networks Backbone에서 얻은 패킷 통계를 이용하여 패킷을 생성한 환경에서 얻은 성능 및 부하 균등 분산 편차는 (그림 8)과 같다. 이를 통하여, 정적 스케줄링 방안에 비하여 최소 부하 스케줄링 방안과 최소 부하 예상 스케줄링 방안은 약 50Mbps 정도의 성능 개선을 보이는 것을 알 수 있다.



(그림 8) 스케줄링 방안에 따른 성능

5. 결 론

본 논문에서는 다중 프로세서 기반 다중 쓰레드 구조의 네트워크 프로세서를 이용한 패킷 처리 과정에서, 패킷의 내용물에 의한 처리 부하를 효과적으로 분산하여 최대의 성능을 얻기 위한 스케줄링 방안을 제안하였다. 타당성 검증을 위하여 제안한 방안을 네트워크 프로세서 시뮬레이터에서 구현하여 실험하였으며, 그 결과 제안된 최소 부하 스케줄링 방안이 기존 방안에 비하여 나은 성능을 나타내는 것을 증명하였다. 실험에서 1024 바이트 이상의 패킷에서는 기가 비트의 성능을 만족하였으며, 실제 망에서 얻은 패킷으로도 600MBps의 성능을 보였다.

참 고 문 헌

[1] A. Campbell, H. De Meet, M. Kounavis, K. Miki, J. Vicente, and D. Villela, "A Survey of Programmable Networks," ACM Computer Communications Review, April, 1999.

[2] P. Crowley, M. E. Fiuczynski, J.-L. Baer and B. N. Bershad, "Characterizing processor architectures for programmable network interfaces," Proceedings of the International Conference on Supercomputing, 2000.

[3] Niraj Shah, Kurt Keutzer, "Network Processors : Origin of Species," Proceedings of ISCIS XVII, The Seventeenth International Symposium on Computer and Information Sciences, October, 2002.

[4] T. Wolf and M. A. Franklin, "Locality-aware predictive scheduling for network processors," Proc. of IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS), Tucson, AZ, Nov., 2001.

[5] D. J. Lilja, "Exploiting the Parallelism Available in Loops," IEEE Computer, Vol.27, No.2, pp.13-26, 1994.

[6] S. Baruah, J. Gehrke and C. G. Plaxton, Fast scheduling of periodic tasks on mutiple resources, Proc. Of the 9th International Parallel Processing Symposium, pp.280-288, Apr., 1995.

[7] K. P. Hung, N. H. C. Yung, and Y. S. Cheung. Multithreaded self-scheduling : Application of multithreading on loop scheduling for distributed shared memory multiprocessor, IEEE International Conference on Algorithms and Architectures for Parallel Processing, Brisbane, Australia, April, 1995.

[8] C. Polychronopoulos and D. Kuck, "Guided Self-Scheduling : Practical Scheduling Scheme for Parallel Supercomputers," IEEE Transactions on Computers, Vol.C-36, No. 12, pp.1425-1439, December, 1987.

[9] Ten H. Tzen and Lionel M. Ni, "Trapezoid Self-Scheduling : Practical Scheduling Scheme for Parallel Supercomputers," IEEE Transactions on Parallel and Distributed Systems, Vol.4, No.1, pp.87-98, January, 1993.

[10] Nie, X., Gazsi, L., Engel, F., Fettweis, G. "A New Network Processor Architecture for High-Speed Communications," Proc. of IEEE Workshop on Signal Processing Systems, Taipei/Taiwan, Oct., 1999.

[11] V. Srinivasan and G. Varghese and S. Suri and M. Waldvogel, "Fast and Scalable Layer four Switching," Proceedings of ACM SIGCOMM, pp.191-202, 1998.

[12] National Laboratory for Applied Network Research, http://www.nlanr.net

[13] Intel IXP1200 Network Processor Family, Hardware Reference Manual, Intel Corp. December, 2001.

[14] Intel IXP1200 Network Processor Family, Development Tools user's Guide, Intel, December, 2001.

임 강 빈

e-mail : yim@sch.ac.kr

1992년 아주대학교 전자공학과(학사)

1994년 아주대학교 전자공학과(석사)

2001년 아주대학교 전자공학과(박사)

1999년~2000년 (미)아리조나 주립대 객원 연구원

2001년~2002년 아주대학교 정보통신대학 대우조교수

2003년~현재 순천향대학교 정보보호학과 전임강사

관심분야 : 네트워크 보안, 실시간 운영체제, 임베디드 시스템, 멀티미디어 시스템 등

박 준 구

e-mail : hwatk@yahoo.co.kr

2002년 아주대학교 전자공학과(학사)

2004년 아주대학교 전자공학과(석사)

2004년~현재 디지털스트림테크놀로지 연구원

관심분야 : 실시간 운영체제, 임베디드 시스템, 네트워크 시스템 등

정 기 현

e-mail : khchung@ajou.ac.kr

1984년 서강대학교 전자공학과(학사)

1988년 Univ. of Illinois, EECS(석사)

1990년 Univ. of Purdue, 전기전자공학부 (박사)

1991년~1992년 현대반도체 연구소

1993년~현재 아주대학교 전자공학부 교수

관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등

최 경 희

e-mail : khchoi@ajou.ac.kr

1976년 서울대학교 사범대학 수학교육과 (학사)

1979년 프랑스 그랑데꼴 ENSEIHT, 정보공학 및 응용수학(석사)

1982년 프랑스 Univ. of Paul Sabatier (박사)

1991년~1991년 프랑스 렌느 IRISA 연구소 교환 교수

1982년~현재 아주대학교 정보 및 컴퓨터공학부 교수

관심분야 : 운영체제, 분산처리, 실시간 시스템 등