SMT 서버를 위한 효율적인 스케쥴링 알고리즘

이정훈[O] 김진석
서울시립대학교 컴퓨터과학부
{jhlee94, jskim}@venus.uos.ac.kr

Effective Scheduling Algorithm for SMT Server

Jung-Hoon Lee[0] and Jin Suk Kim
School of Computer Science, University of Seoul

요 약

최근 많은 프로세서 제작업체들이 프로세서의 효율을 높이기 위한 방법으로 독립적인 쓰레드들을 한 프로세서 사이클에 동시에 실행시킬 수 있는 SMT 기술을 구현하고 있으며 그 예의 하나가 하이퍼쓰레딩이다. 물리 프로세서 안에 여러 개의 논리 프로세서를 가질 수 있는 하이퍼쓰레딩 기술은 기존의 여러개의 독립적인 프로세서들을 갖춘 멀티 프로세싱 환경과는 차이가 있으며, 하이퍼쓰레딩 환경에 알맞은 특정한 작업 할당방법이 필요하다. 따라서, 본 논문에서는 하이퍼쓰레딩 기술에 적합한 스케쥴링 알고리즘을 제안하고 그 성능을 다양한 방법으로 측정해 봄으로서 하이퍼쓰레딩 시스템을 올바르게 인식하고 적절하게 관리하여 효율적인 성능을 기대할 수 있게 되었다.

1. 서 론

프로세서의 자원 낭비를 줄이고 효율을 높이기 위해 1995년 Dean Tullsen에 의해 제안된 SMT(Simultaneous MultiThreading)[1] 기술은 최근 하이퍼쓰레딩(Hyper Threading)[2] 이라는 이름으로 실제 프로세서에 구현되었다. 하이퍼쓰레딩 기술은 한 개의 물리 프로세서가 두 개의 논리 프로세서를 가질 수 있게 함으로써 서로 다른 쓰레드를 동시에 처리할 수 있도록 설계되었다[3]. 하이퍼쓰레딩 기술은 각 논리 프로세서들이 하나의 물리 프로세서 안에서 Execution engine, 캐쉬, 시스템 버스 인터페이스 등과 같은 실행 자원들을 공유한다[4]. 이같은 하이퍼쓰레딩 시스템의 특징은 독립적인 프로세서 자원들을 갖춘 기존의 일반적인 멀티 프로세서 환경과는 차이가 있으며, 효율적인 자원 활용과 프로세서 성능 향상을 위하여 스케쥴링 정책을 다른 방식으로 접근해야 할 필요가 있다. 따라서 본 논문에서는 하이퍼쓰레딩 시스템에 적합한 스케쥴링 알고리즘을 제시하고 이를 리눅스 운영체제에 적용함으로써 하이퍼쓰레딩 기술을 지원하는 시스템을 올바르게 인식하고 논리 프로세서들을 적절하게 관리하여 효율적인 성능을 기대할 수 있게 되었다.

2. 하이퍼쓰레딩 시스템의 특징

캐쉬, 시스템버스, Execution Engine 등을 독립적으로 보유하고 있는 SMP 시스템과 이를 공유하는 하이퍼쓰레딩 시스템은 기본적으로 프로세서 설계 단계에서부터 큰 차이가 존재하며 이와같은 구조상의 차이는 응용 단계의 시스템 성능에도 다양한 영향을 미치게 된다. 따라서 본 논문에서 제안한 스케쥴링 알고리즘을 설명하기 전에 우선 하이퍼쓰레딩 기술이 응용 단계에서 어떤 특징을 가지고 있는지 살펴보고 왜 현재의 운영체제가 기존의 일반 SMP 환경과 다르게 스케쥴링 정책을 취해야 하는지 살펴볼 필요가 있다. 이를 위해서 본 논문에서는 하이퍼쓰레딩 기술을 지원하는 듀얼 인텔 Xeon 2.4GHz 프로세서와 리눅스 커널 2.4.17, 인텔 C++ 컴파일러, 그리고 OpenMP 버전 2.0을 이용하여 실험하였으며, 실험결과를 분석하기 위해서 인텔 VTune 분석기를 사용하였다. 각각의 실험은 파이값을 계산하는 루틴을 OpenMP를 이용하여 병렬화시킨 코드를 수행시켜 완료된 결과를 분석하는 방법으로 수행되었으며, 각 쓰레드들이 수행하는 루틴은 각각 정수 연산만으로 이루어진 동질적인 코드와 정수 연산과 부동소수점 연산의 혼합으로 이루어진 이질적인 코드로 실험하여 결과를 비교하였다.

```
int compute_pi_by_Thread() {
        int i;
        int x, pi, sum=0, step;
        int myid;
        printf ("Entering compute_pi 1.");
        step = 1/MAX;
        myid = omp_get_thread_num();
        for (i=1; i < MAX; i++) {
                x = (i-1)*step;
                sum = sum + 4/(1+x*x); }
        pi = sum*step;
        printf("%d, %d thread finished.", pi, myid);
        return 0;
}
```

표 1. 쓰레드 수행 루틴

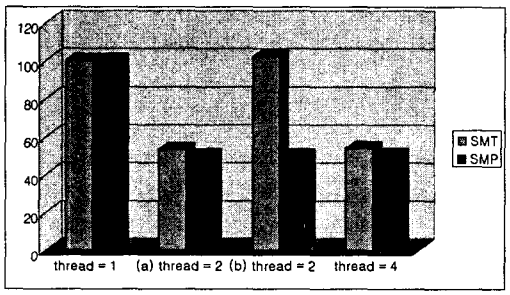

그림 1. 쓰레드 루틴 수행 결과

<그림 1>은 <표 1>의 쓰레드 코드를 수행시킨 결과를 쓰레드 개수의 변화에 따라 초 단위로 나타낸 것이다. 하이퍼쓰레딩(SMT) 시스템을 기존의 SMP 시스템과 비교하기 위하여 같은 시스템에서 각각 하이퍼쓰레딩 기능을 켜거나 끄는 방법으로 실험을 거듭하여 결과값을 추출하는 방법을 이용하였다. 쓰레드 개수가 한 개이거나 네 개일경우 SMT와 SMP가 유사한 성능을 나타내었다. 프로세서 개수가 4개로 보이는 하이퍼쓰레딩 시스템이 프로세서 2개로 보이는 SMP 시스템과 비교하여 그다지 큰 성능향상을 이루지 못하는 이유는 단순 for 루프로 이루어진 쓰레드 루틴의 특성상 극도로 효율적인 분기예측과 캐쉬 활용률, 그리고 상대적으로 소수인 Pipeline Stalling 에 기인한다. 위의 실험에서 특이한 점은 쓰레드 개수가 두 개일 경우 하이퍼쓰레딩 시스템에서 상이한 두 개의 결과(a, b)가 반복하여 나타나는 점이다. 이를 좀더 구체적으로 살펴보기 위하여 VTune 분석기를 이용하여 분석한 결과는 다음과 같다.

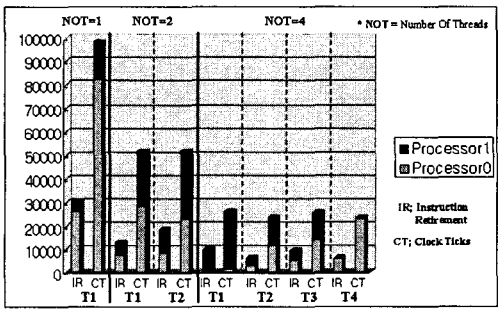

그림 2. SMP 시스템에서 수행결과 분석

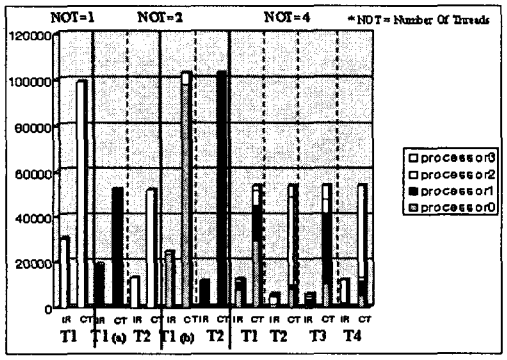

그림 3. 하이퍼쓰레딩 시스템에서 수행결과 분석

<그림 2>는 <그림 1>의 실험을 SMP에서 수행하였을때의 결과이며, <그림 3>은 같은 루틴을 하이퍼쓰레딩 시스템에서 수행하였을때의 결과이다. IR은 Instruction Retired 개수, CT는 Clock Tick을 의미하며, NOT는 Number of Thread, 그리고 T1, T2, T3, T4는 각각의 쓰레드들을 나타낸다. <그림 2>에서 SMP 의 경우 2개의 프로세서로 적절하게 스케쥴링 하여 수행하고 있음을 관찰할수 있으나, <그림 3>의 하이퍼쓰레딩 시스템의 경우 NOT=2 일 때 4개의 논리 프로세서 자원중에서 때때로 잘못된 자원할당 정책으로 연산을 수행시키고 있음을 알 수 있다. 즉, 프로세서 0번과 1번이 실제로 같은 물리 프로세서 안에 포함되어 있는 논리 프로세서이며, 프로세서 2번과 3번도 같은 물리 프로세서의 논리 프로세서로 구성되어 있는 하이퍼쓰레딩 시스템에서 <그림 3>의 (a)의 경우 2개의 쓰레드가 각각 물리적으로 독립적인 프로세서 1번과 3번으로 할당되어 <그림 1>의 (a) 결과처럼 정상적인 결과를 나타내는데 반해서 <그림 3>의 (b)의 경우 2개의 쓰레드가 각각 같은 물리 프로세서에 속해있는 두 개의 종속적인 논리 프로세서 0번과 1번에 각각 할당됨으로서 쓰레드간 연산을 위해 공유된 프로세서 자원경쟁이 심해지고 결과적으로 <그림 1>의 (b)와 같은 나쁜 결과를 나타내는 것이다. 이는 기존의 운영체제 스케쥴러 단계에서 하이퍼쓰레딩 시스템의 특수성을 인지하지 못하고 기존의 일반적인 멀티 프로세서 환경과 같은 방법으로 스케쥴링 정책을 취함으로서 발생되는 현상이다. 이와같은 단점을 극복하기 위하여 하이퍼쓰레딩 기술에 적합한 자원할당 정책이 필요하다.

3. 관련 연구

하이퍼쓰레딩 시스템을 인식하고 프로세서 자원들을 적절하게 관리하기 위하여 리눅스 커널 2.4에서는 128-byte lock alignment, Spin-wait loop optimization, Non-execution based delay loops 기능등 여러 가지 다양한 기능을 추가하였으나 실제적으로 위의 실험에서 보여지는 단점들을 해결하였다고 보기 어렵다. 이를 해결하기 위하여 Molnar 는 리눅스 커널 2.5.32에서 다음과 같은 기능을 지원하는 방법을 제안하였다[5].

1. HT-aware passive load-balancing.
2. "Active" load-balancing.
3. HT-aware task pickup.
4. HT-aware affinity.
5. HT-aware wakeup.

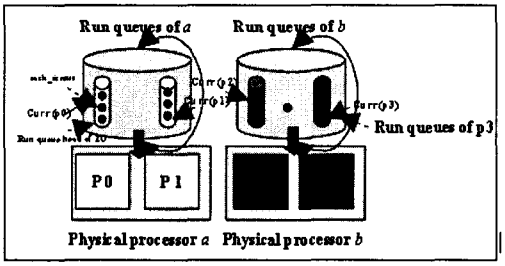

그림 4. Molnar 가 제안한 스케쥴링 기법

<그림 4>는 Molnar 가 제안한 스케쥴링 방법을 도식화한 모습으로서 2개의 논리 프로세서로 구성된 물리 프로세서에 공통적인 작업큐를 생성하고 하위 단계에서 각각의 논리 프로세서에 해당하는 작업큐를 구성하는 방법으로 이루어져 있다. Molnar가 제안한 방식은 하이퍼쓰레딩 시스템을 정확하게 인식하고 관리하며 위의 실험에서 보여지는 하이퍼쓰레딩 시스템의 특수한 성질들을 적절하게 대처할 수 있다는 장점이 있으나 지나치게 복잡하고 확장성이 낮으며, 일반적인 SMP 시스템이나 Uni-processor 환경에 적용할수 없다는 단점을 가지고 있다. 또한 논리 프로세서간의 관계를 인식시키기 위하여 생성된 공통큐로 인하여 잠재적으로 다른 여러 가지 부작용을 가지고 있다는 단점도 가지고 있다. 따라서 본 논문에서는 이와같이 복잡한 설계구조와 구현보다는 상대적으로 단순한 정책으로 하이퍼쓰레딩 시스템에 좀더 적합한 자원관리를 수행할 수 있는

스케쥴링 알고리즘을 구상하였다.

4. HT-Scheduler 알고리즘

하이퍼쓰레딩 시스템에서 논리 프로세서에 할당되는 각각의 작업들은 가능하면 서로 다른 독립적인 프로세서에 할당되는 것이 효율적이라는 사실을 주지해볼 때 적응형 스케쥴링 알고리즘은 기본적으로 리눅스 스케쥴러에서 작업을 할당할 때 서로 다른 물리 프로세서에 포함되어 있는 논리 프로세서를 감지해 내고 이와같은 사실을 기반으로 하여 가능하면 서로 다른 물리 프로세서에 작업을 할당시키는 방법을 핵심으로 한다. 물리 프로세서와 논리 프로세서와의 상관관계는 <그림 5>와 같이 APIC (Advanced Programmable Interrupt Controller) ID 로 구별할 수 있으며[6], 이 정보를 기초로 하여 각각의 물리 프로세서에 포함되어 있는 논리 프로세서 개수와 관계 정보를 정의할수 있다[7].

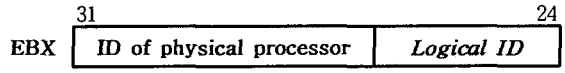

그림 5. 하이퍼쓰레딩 프로세서의 APIC ID 구조

HT-스케쥴링 알고리즘은 이와같은 정보를 토대로 하여 물리 프로세서에 속해있는 논리프로세서의 개수를 판독해내고, 들어오는 작업에 대하여 물리 프로세서에 우선 할당하는 접근법을 취하게 된다. n개의 물리 프로세서와 각각의 물리 프로세서에 m개의 논리 프로세서를 가지는 시스템에서 T_i 태스크를 할당한다고 가정했을 때, 이를 수식으로 표현하면 <그림 6>과 같다.

Algorithm HT − scheduler

Input : $T_i,\ i \geq 0$
Output : $P_j(C_k),\ 0 \leq j \leq n-1$ and $0 \leq k \leq m-1$
1. *find* $\alpha = i \bmod n$
2. *find* $\beta = (i \ div\ n) \bmod m$
3. *return* $P_\alpha(C_\beta)$

그림 6. HT-스케쥴링 알고리즘

또한 이미 특정 논리 프로세서에서 실행된 작업을 다시 재개시킬 때 같은 물리 프로세서안에 포함된 다른 모든 논리 프로세서들에게도 같은 가중치를 부여시킴으로서 공유 자원의 효율성을 극대화시킨다. 이와같은 방법은 하이퍼쓰레딩 시스템에서 논리 프로세서와 물리 프로세서와의 상관관계를 인식하면서 위의 실험에서 보여졌던 특성들을 감안한 스케쥴링을 가능케 함으로서 하이퍼쓰레딩 시스템의 효율성을 높이게 된다. 또한 상대적으로 간단한 설계로 인해 확장성과 이식성이 높으며 기존의 멀티 프로세서 환경에도 적용할수 있다는 장점을 가지고 있다.

5. HT-스케쥴링 알고리즘 성능 분석

본 논문에서 제안한 HT-스케쥴링 알고리즘의 성능을 분석하기 위해 하이퍼쓰레딩 기능을 지원하는 듀얼 인텔 Xeon 2.4GHz 프로세서와 리눅스 커널 2.5.3, 그리고 dbench 벤치마킹 툴킷[8]을 사용하였다. dbench 벤치마킹 툴킷은 네트워크 상에서 다수의 파일 접근을 허용하는 파일서버 성능 측정에 주로 사용되는 툴킷으로서 많은 량의 부하를 생성하고 프로세서 연산을 수행시키며, 동시에 접근하거나 생성하는 파일의 개수가 대단히 높기 때문에 하이퍼쓰레딩 시스템 성능측정에 적합한 것으로 알려져있다.

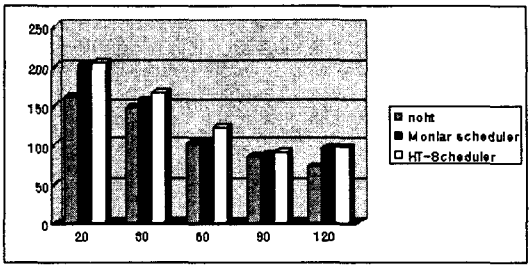

그림 7. HT-스케쥴러의 성능분석 결과

<그림 7>은 하이퍼쓰레딩을 감안하지 않은 것과, Molnar 가 제안한 스케쥴러의 성능, 그리고 본 논문에서 제안한 스케쥴러의 성능을 비교 분석한 것이다. 가로축은 dbench 의 파라미터 값이며, 파라미터 값이 증가할수록 접근하는 클라이언트의 수, 즉 워크로드가 증가하게 된다. 세로축은 throughput을 MB/sec 단위로 나타낸 것으로서 높을수록 좋은 성능을 의미한다. 대부분의 경우에서 HT-Scheduler 가 우수한 성능을 나타냄을 알수 있다.

6. 결 론

본 논문에서는 하이퍼쓰레딩 시스템에서 기존의 멀티 프로세서 환경과의 차이성을 극복하기 위하여 HT-Scheduler 알고리즘을 제안하였으며 실험을 통해 HT-Scheduler 알고리즘이 우수한 성능을 나타냄을 보임으로서 하이퍼쓰레딩 기술을 지원하는 시스템을 올바르게 인식하고 관리하여 효율적인 성능을 기대할 수 있게 되었다. 향후에는 쓰레드 이질성에 따른 하이퍼쓰레딩 시스템의 효율성을 분석하고 이를 감안한 좀더 효과적인 스케쥴링 알고리즘과 실제 응용단계에서 여러 가지 스케쥴링 요구를 충족시킬수 있는 방안에 대한 연구가 진행되어야 할 것이다.

참고문헌

[1] D. Tullsen. S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proc of the 22nd Annual International Symposium on Computer Architecture,* June, 1995.

[2] Intel Corporation., "Introduction To Hyper-Threading Technology," Document number 250008-002, 2001.

[3] Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufaty, J. Alan Miller, Michael Upton, "Hyper-Threading Technology Architecture and Microarchitecture," *Intel Technology Journal,* Q1, 2002.

[4] Intel corporation, "White Paper: Hyper-Threading Technology on the Intel Xeon Processor Family for Servers," 2002.

[5] http://lwn.net/Articles/8553/

[6] Intel corporation, "IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture," 2001.

[7] Intel corporation, "Detecting Support for Hyper-Threading Technology Enabled Processors," Dec, 2001.

[8] http://samba.org/ftp/unpacked/dbench/README