공학석사학위논문

범용 CMT 시스템에서 에너지 효율을 향상시키기 위한 에너지 인지 로드 밸런싱 메커니즘

An Energy Aware Load Balancing Mechanism
to Improve Energy Efficiency
on The General Purpose CMT Systems

2012 년 8 월

서울대학교 대학원
지능형융합시스템학과
이 종 환

범용 CMT 시스템에서
에너지 효율을 향상시키기 위한
에너지 인지 로드 밸런싱 메커니즘

An Energy Aware Load Balancing Mechanism
to Improve Energy Efficiency
on The General Purpose CMT Systems

지도교수 홍성수

이 논문을 공학석사학위논문으로 제출함

2012년 4월

서울대학교 대학원
지능형융합시스템학과
이 종 환

이종환의 석사학위논문을 인준함

2012년 5월

위 원 장 박재홍 (인)
부 위 원 장 홍성수 (인)
위 원 안정호 (인)

초 록

현재 대부분의 범용 프로세서는 CMP(Chip Multi-Processor) 구조로 되어있다. 최근에는 여기에 최소의 비용으로 추가적으로 성능을 향상시킬 수 있는 SMT(Simultaneous Multi-Threading) 기술을 빠르게 결합시키고 있다. 이렇게 CMP 구조에 SMT 기술을 도입한 CMT(Chip Multi-Threading) 구조는 범용 프로세서 시장의 50% 이상을 차지하고 있다. 범용 프로세서 시장의 90% 점유율을 차지하고 있는 Intel사의 경우, 최근 2년간 생산한 CPU의 64%가 SMT 기술의 일종인 Hyper Threading Technology를 탑재하고 있는 CMT 프로세서이다. 또한 시장점유율 2위를 차지하고 있는 AMD사의 최신 불도저 아키텍처에서는 Intel사와는 다른 방식의 SMT 기술을 적극 도입하고 있다. 또한 스마트폰용 모바일 AP 기술의 발전이 급속도로 한계에 다다름에 따라 모바일 프로세서 시장에서의 SMT 기술 도입은 시간문제일 뿐이다.

SMT 기술을 탑재한 범용 프로세서가 널리 퍼짐에 따라 이 기술을 효과적으로 사용할 수 있는 방안도 많이 연구되고 있다. 그러나 대부분의 SMT와 관련된 연구는 SMT 성능을 최대한 활용하기 위한 방안에 초점을 맞추고 있으며 에너지 효율을 중심으로 한 연구는 드물다. 에너지 효율을 중심으로 한 연구들도 대부분 ST(Single Thread)와 SMT의 에너지 효율 비교, 혹은 CMP와 SMT의 에너지 효율을 비교하는데 집중하고 있다.

본 논문에서는 프로세서의 성능을 향상시키기 위한 방법이나 어떤 시스템이 에너지 효율이 더 높은지를 비교하기 보다는 어떻게 하면 CMT 구조에서 SMT 기술을 적극 활용하여 에너지 소비를 줄일 수 있는 지에 대해 알아본다.

SMT와 CMP 기술 모두를 사용할 수 있는 CMT 구조에서는

multi-threading을 수행하기 위해 SMT나 CMP를 선택적으로 사용할 수 있다. 일반적으로 두 개의 thread를 CMP 구조를 이용하여 두 개의 코어에서 동시에 실행할 때 가장 높은 throughput을 얻을 수 있다. 반대로 두 개의 thread를 SMT 구조를 이용하여 한 개의 코어에서 두 개의 HT로 나누어 동시에 실행할 경우 가장 높은 에너지 효율을 얻을 수 있다.

CMT 구조에 존재하는 성능-에너지 트레이드오프에서 많은 OS들이 성능을 중시해왔다. 반대로 에너지 소비를 줄이기 위한 방법은 상대적으로 중요하게 다뤄지지 않았다. 본 논문에서는 SMT를 사용할 경우 특정한 상황에서 기존의 OS들이 체감성능 저하 없이 에너지 소비를 낮출 수 있는 기회를 놓치고 있음을 보이며, 이 문제를 해결하기 위해 에너지 인지 로드 밸런싱 메커니즘을 제안한다. 또한 제안된 메커니즘을 Linux kernel에 구현하여 light load condition일 경우 체감성능 저하 없이 약 10%의 에너지 소비를 줄일 수 있음을 보인다.

주요어: 로드 밸런싱, 에너지 인지, SMT, CMP, CMT
학 번: 2010-23958

목 차

표목차

그림목차

제 1 장 서론

1.1 연구배경

1990년대부터 2000년대 초까지 마이크로프로세서의 성능은 프로세서의 clock frequency에 달려있었다. 마이크로프로세서 개발회사들은 그림 1에서처럼 서로 더 높은 clock frequency를 가진 프로세서를 만들어내기 위해 노력했다[11]. 하지만 이런 마이크로프로세서 개발회사들 사이의 clock frequency 경쟁은 2000년대 초에 한계에 다다랐다. 개발회사들은 그림 2의 clock frequency 상승에 따라 소비전력이 급증하는 power wall 문제를 쉽사리 해결하지 못하였다. 여기에 CPU와 memory 동작 속도 차이에 의해 발생하는 memory wall 문제와 ILP(instruction level parallelism) 개선을 힘들게 만드는 ILP wall 문제들은 프로세서의 성능을 증가시키기 위해 TLP(thread level parallelism)라는 다른 방식의 접근법을 생각하게 만들었다[13].

이 때 많이 연구된 방법이 SMT(simultaneous multi-threading)와 CMP(chip multi-processor) 기술이다. SMT와 CMP는 여러 개의 thread를 동시에 처리할 수 있게 하는 TLP를 구현했다는 점에서는 유사점을 보이지만, 구조적인 면에서는 큰 차이를 보인다.

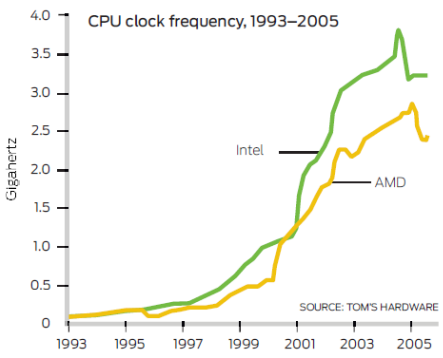


그림 1 CPU clock 속도 추세[11]

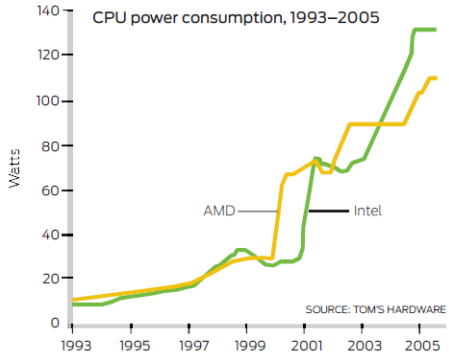


그림 2 CPU 소비전력 추세[11]

그림 3 a)에서 보듯이 일반적인 SMT 프로세서는 2개의 HT(hardware thread)와 슈퍼스칼라구조로 된 1개의 processor execution resource라는 연산자원으로 구현 되어있다. 각각의 HT는 1개의 context를 처리할 수 있어 SMT 구조는 프로세서의 연산 자원을 사용하는 두 개의 작업을 동시에 실행할 수 있다. 마이크로프로세서의 슈퍼스칼라 구조는 여러 개의 execution unit들로 이루어져 있는데 1개의 thread만으로는 이 execution unit들을 100% 사용할 수 없다. 그림 4의 왼쪽 그림은 슈퍼스칼라구조로 thread를 실행하지 않는 execution unit들이 하얀색 사각형으로 존재한다. 그림 4의 오른쪽 그림은 SMT 구조로 두개의 thread를 동시에 실행함으로써 execution unit들의 사용률이 높아지는 것을 알 수 있다[12].

인텔의 SMT 기술인 hyper threading을 적용한 마이크로아키텍처인 Nehalem 에서는 SMT 기술의 구현을 register state와 return stack buffer(RSB)를 추가하는 것으로 쉽게 할 수 있다[14]. 이렇게 추가되는 부분은 패키지 크기 대비 아주 작은 크기를 차지하기 때문에 적은 비용으로 구현이 가능하다. 반면 이러한 비용 대비 SMT 기술은 일반적으로 더 큰 성능 향상을 기대할 수 있으며 구조에 따라서는 1개의 코어에 2개 이상의 HT를 가지고 있기도 한다.

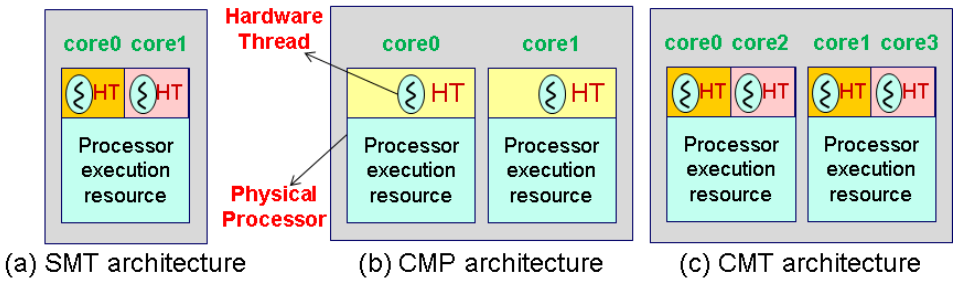


그림 3 SMT / CMP / CMT 의 구조

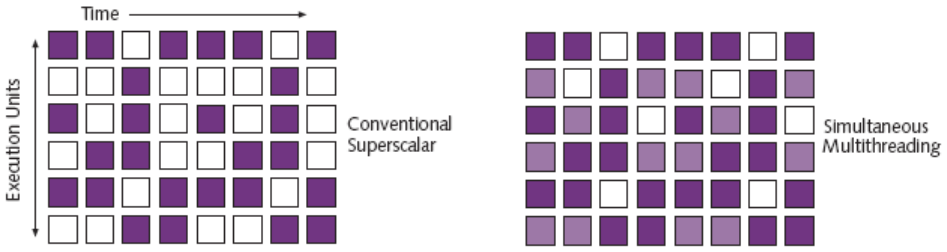


그림 4 슈퍼스칼라와 SMT 구조에서의 Execution unit 사용[12]

그림 3 (a)는 SMT 구조를 보여주고 있다. 한 개의 코어에 HT 두 개를 동시에 실행하여 두 개의 논리 코어를 구성하고 있는 모습이다. 그림 3 (b)는 CMP 구조이다. SMT는 하나의 코어에 HT를 여러 개 사용하는데 반해 CMP는 하나의 HT를 가진 코어를 여러 개 사용한다. 그림 3 (b)는 한 개의 HT 수행할 수 있는 코어 두 개로 구성된 듀얼 코어 CMP 구조이며 코어의 개수가 늘어남에 비례하여 패키지 사이즈가 커지므로 상당한 추가비용이 필요함을 알 수 있다. 그러나 하나의 execution resource를 두고 경쟁적으로 사용하는 SMT 구조에 비해 CMP 구조는 경쟁 없이 동시에 두 개의 thread를 병렬적으로 처리할 수 있어 더 큰 성능향상을 기대할 수 있다.

그림 3 (c)는 SMT와 CMP 기술을 선택적 혹은 동시에 사용할 수 있는 CMT 프로세서이다. CMP 구조에 SMT 기술을 추가함으로서 최소한의 비용으로 최대의 성능을 낼 수 있도록 설계되었다.

이러한 CMT 구조를 가진 프로세서는 현재 범용 프로세서 시장의 과반수를 차지하고 있다. 범용 프로세서 시장의 90% 점유율을 차지하고 있는 Intel사의 경우에는 최근 2년간 생산한 CPU의 64%가 SMT 기술의 일종인 Hyper Threading Technology를 탑재하고 있는 CMT 이다. 또한 시장점유율 2위를 차지하고 있는 AMD사의 최신 불도저 아키텍처에서는 Intel사와는 다른 방식의 SMT 기술을 적극 도입하고 있다. 또한 스마트폰용 모바일 AP 기술의 발전이 급속도로 한계에 다다름에 따라 모바일 프로세서 시장에서의 SMT 기술 도입은 시간문제일 뿐이다.

1.2 관련연구

CMT 구조를 가진 범용 프로세서가 널리 퍼짐에 따라 이 기술들과 관련된 연구가 활발히 이루어지고 있다.

연구의 시작 단계에서는 SMT와 CMP의 성능적 우위를 비교하는 연구가 주를 이뤘다. Dean M. Tullsen et al.은 기존의 슈퍼스칼라 구조를 갖는 프로세서의 한계를 보이고 SMT 기술이 이를 극복할 수 있는 해결책임을 증명하였다[1]. 또한 자원 사용 효율 측면에서는 SMT 구조가 CMP 구조보다 우월함을 보였다. Lance Hammond et al.은 TLP가 적용되는 태스크에서 CMP의 성능이 SMT 보다 우수함을 밝혔다[2].

2000년대 초에는 SMT와 CMP의 에너지 효율에 관심을 갖기 시작했다. Ruchira Sasanka et al.은 멀티미디어 워크로드에서는 일반적으로 CMP 구조가 SMT 구조 보다 에너지 효율이 더 좋으나 아키텍처 복잡도에 따

라 SMT 구조가 더 좋을 수도 있음을 확인하였다[3]. Yingmin Li et al.은 L2 cache miss rate에 따라 SMT와 CMP의 에너지 효율이 달라진다는 결론을 내렸다[4].

SMT와 CMP의 구조에 대한 연구가 진행이 된 후에 SMT 시스템을 위한 태스크 스케줄링에 대한 관심도 높아졌다. Allan Snavely et al.은 hardware thread 보다 task thread 수가 많을 경우 sampling hardware performance counters를 이용하여 태스크를 co-scheduling 하여 성능을 높이는 symbiotic jobscheduling을 제안하였다[5]. Sujay Parekh et al.은 SMT 시스템의 성능을 향상시키기 위해 개별적인 thread 자원 요구사항을 분석하고 예측하는 스케줄링 알고리즘을 만들었다[6]. Matthew Devuyst et al.은 core간의 load를 균등하게 분배하지 않고 연관이 있는 thread끼리 묶어 스케줄링 하는 방법을 제시하였다[7]. Petar Radojkovic et al.은 core의 개수가 많은 상황에서 네트워크 어플리케이션을 실행할 때 시스템 성능을 향상시킬 수 있는 방안에 대해 연구하였다[8].

이상에서 보았듯이 SMT 기술 도입 초기의 연구들은 SMT와 CMP의 특성 비교 및 평가에 치중하였다. SMT 구조가 슈퍼스칼라 구조에 유용함을 보이고 상황에 따른 SMT와 CMP의 성능 및 에너지 효율을 비교하는 연구들이 대다수였다[1]~[4]. SMT와 CMP의 특성에 관련된 연구가 진행된 시점에서야 SMT의 특성을 활용할 수 있는 방안에 대한 것이 나오기 시작하였다. Thread의 자원 요구 사항을 분석하여 자원 경쟁을 줄여 SMT의 성능을 개선시키려는 연구나 연관이 있는 작업을 그룹으로 스케줄링하여 효율을 높이려는 연구들이 이루어졌다[5]~[7]. 그러나 이러한 SMT 특성 활용 방안에 관련된 연구는 SMT 구조의 우수성을 입증하고 성능을 향상시키기 위한 방법에 편중되어 있으며, 에너지 효율을 높이기 위한 방안은 찾아보기 힘들다.

1.3 연구내용 및 논문의 구조

앞서 살펴본 바와 같이 범용 마이크로프로세서 시장에서 SMT 기술이 급속히 확산되고 있다. 그러나 이러한 SMT 기술의 활용에 대한 연구는 많지 않아 기존의 OS들이 SMT 기술을 적절히 사용하지 못하는 환경을 조성하고 있다. 이러한 SMT 기술의 부적절한 사용은 불필요한 시스템 자원 낭비일 뿐만 아니라 불필요한 에너지 소비를 일으킨다.

본 논문에서는 SMT의 높은 에너지 효율성을 적극 활용하기 위한 에너지 인지 로드 밸런싱 메커니즘을 제안한다. 범용 CMT 시스템에서 core 간의 로드 밸런싱을 적절히 수행함으로서 light load condition일 경우 컴퓨터 사용자의 체감성능을 저하시키지 않고 에너지 소비를 10% 절감할 수 있음을 보인다.

본 논문의 나머지는 다음과 같이 구성된다.

2장에서는 기존의 SMT 기술에 관한 연구결과들을 바탕으로 SMT와 CMP의 구조적 특징 및 차이점을 알아본다. 또 Linux의 로드 밸런싱 메커니즘을 분석하여 기존의 Linux 로드 밸런싱 메커니즘의 비효율적인 에너지 사용 문제를 밝힌다. 3장에서는 제시된 문제를 해결하기 위해 체감성능을 정의하고 이를 바탕으로 한 에너지 인지 로드 밸런싱 메커니즘을 제안한다. 4장에서는 제안된 메커니즘을 Linux 스케줄러 상에 구현하고 제안하는 메커니즘이 기존 메커니즘 대비 light load condition에서 10% 에너지 절감 효과가 있음을 실험을 통해 검증한다. 마지막으로 5장에서는 본 논문의 결론을 내린다.

제 2 장 문제 제시

이 장에서는 SMT와 CMP 구조의 성능과 에너지 효율과 관련된 기존의 연구결과들을 살펴보고, 최신 SMT 구조의 프로세서를 대상으로 한 실험을 통해 기존 연구결과와 다른 점을 도출하여 현재 Linux의 로드 밸런싱 메커니즘의 문제점을 도출해 낸다.

2.1 기존 연구결과: SMT vs. CMP

일반적으로 SMT와 CMP의 성능을 비교하기 위하여 ST(single thread) 구조의 결과를 기준으로 평가한다. 그림 5와 6은 SMT와 CMP의 성능과 에너지 효율을 비교한 결과이다[4]. 그림 5의 경우에는 L2 cache miss rate가 낮은 workloads 일 때의 결과이며, 그림 6은 L2 cache miss rate가 높을 때 결과이다.

그림 5의 IPC(Instructions per cycle, 성능지표)를 보면 L2 cache miss rate가 낮은 workloads 에서는 CMP가 SMT 보다 IPC값이 크게 나와 성능이 월등하게 높음을 알 수 있다. 또한 CMP는 성능이 SMT 보다 높음에도 불구하고 에너지의 소모는 오히려 SMT 보다 적게 나와 에너지 효율이 CMP가 SMT 보다 좋음을 알 수 있다. 한편 SMT와 CMP 모두 ST 구조에 비해서 에너지 소비는 많았지만 에너지 효율은 좋았다.

그림 6의 IPC 값을 보면 L2 cache miss rate가 높은 workloads 에서는 SMT가 CMP 보다 높은 성능을 나타냈다. 에너지 소비 측면에서는 SMT가 CMP 보다 낮아 L2 cache miss rate가 낮은 workloads 인 경우와 반대되는 결과가 나왔다. 또한 SMT와 CMP 모두 ST 보다는 많은 에너지를

소비했다.

이를 바탕으로 보았을 때 L2 cache miss rate가 높을 때는 성능과 에너지 소비 및 효율 측면에서 SMT가 유리하고 낮을 때는 CMP가 유리함을 알 수 있으며, 에너지 소모는 ST에 비해 나빠지는 것을 확인할 수 있다.

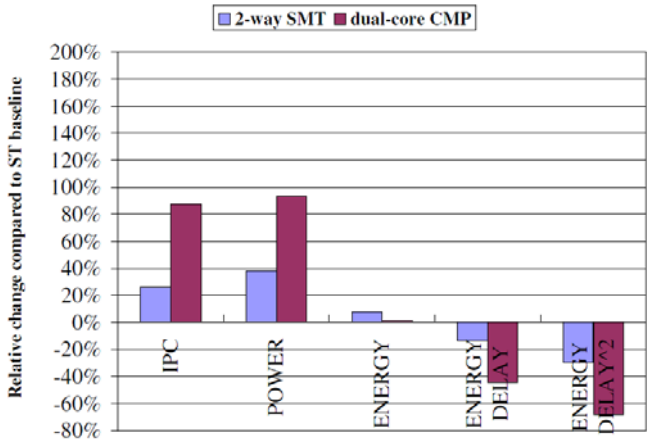


그림 5 낮은 L2 cache miss workloads 일 때 SMT vs. CMP의 성능과 에너지 효율 비교[4]

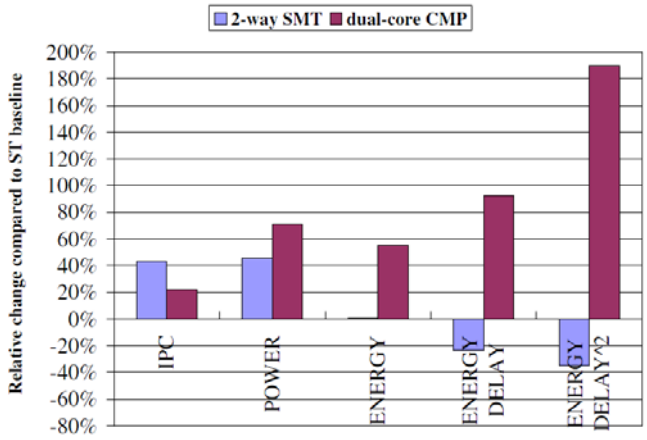


그림 6 높은 L2 cache miss workloads 일 때 SMT vs. CMP의 성능과 에너지 효율 비교[4]

멀티미디어 workload에서 SMT와 CMP 를 비교한 다른 연구결과를 보면 하드웨어 복잡도에 따라 에너지효율이 달라짐을 알 수 있다[3]. 그림 7에서 하드웨어 복잡도가 낮을 때는 SMT의 에너지효율이 좋다가 복잡도가 증가함에 따라 효율이 나빠짐을 알 수 있다.

또한 벤치마크 특성에 따른 에너지효율을 비교한 한 연구결과를 보면 벤치마크 특성에 따라 SMT와 CMP의 우위가 바뀜을 알 수 있다.[9]

이를 종합하면 L2 cache miss rate과 하드웨어 복잡도 그리고 벤치마크 특성에 따라 SMT와 CMP의 에너지 효율이 달라진다고 결론을 내릴 수 있다. 이러한 불확실성 문제는 과거 SMT를 사용하여 적극적으로 에너지를 줄이려는 시도를 제한하는 요인이었다.

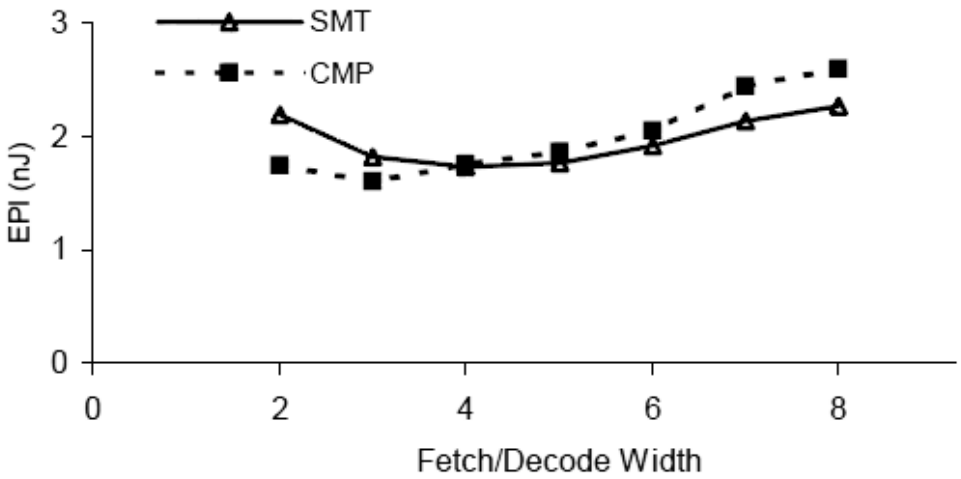


그림 7 하드웨어 복잡도에 따른 SMT와 CMP 에너지 효율 비교[3]

2.2 SMT 구현 기술의 발전

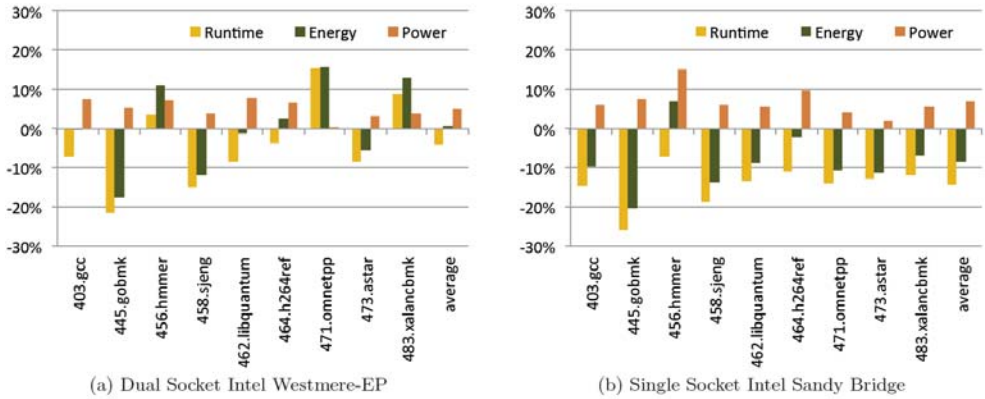
기존의 SMT와 관련된 연구들은 1990년대 중반부터 2000년대 초에 집중되어 있으며, 이때는 SMT 기술이 상업적으로 막 도입되던 시기였다. 즉 SMT 기술의 구현이 안정적이지 않은 시기였다. 그에 따라 2.1장에 서술한 바와 같이 상황에 따른 성능과 에너지 효율이 크게 변하는 문제를 안고 있었다.

Intel의 SMT 기술인 Hyper Threading에서 최근 이러한 문제점을 해결하기 위한 노력들을 볼 수 있다. Intel의 경우 SMT 기술을 최초로 도입한 Netburst 아키텍처 대비 Nehalem 아키텍처에서는 메모리 대역폭과 캐시 용량을 늘리는 등 구조적으로 SMT의 문제점을 개선하려고 하였다. 그러나 여전히 vectorization play에서는 성능상의 이득이 없는 문제를 가지고 있었다[14].

Robert Schöne et al.의 최근 연구는 최신 SMT 기술이 적용된 Intel사의 Sandy bridge 아키텍처에서 기존의 문제들이 많은 부분 해결되었음을 보였다[10]. 그림 8은 Intel Nehalem 아키텍처가 적용된 Xeon X5670 프로세서와 최신 Intel Sandy bridge 아키텍처가 적용된 Core i7 2600 프로세서의 에너지 효율 비교 자료이다. 그림 8은 총 열 개의 벤치마크 테스트 결과인데 (a)와 (b)가 상당히 다른 결과를 나타내고 있다.

그림 8 (a)의 경우에는 벤치마크 종류에 따라 에너지 소비가 나빠질 때도 있고 좋아질 때도 있다. 그러나 그림 8 (b)의 최신 아키텍처에서는 대부분의 경우 에너지 소비가 줄어듦을 볼 수 있다.

그러나 [10]의 결과는 SMT와 CMP를 구분하지 않고 테스트한 결과로 SMT와 CMP 중 어떤 기술이 발전하여 문제점이 개선되었는지 명확하지 않다는 한계를 지닌다.



(a) Dual Socket Intel Westmere-EP (b) Single Socket Intel Sandy Bridge

그림 8 Intel 아키텍처에 따른 에너지 효율 비교[10]

2.3 최신 SMT 아키텍처의 실험 결과

2.2의 결과를 명확히 하기 위하여 다음과 같은 시스템으로 에너지 효율을 측정하는 실험을 하였다.

프로세서	Intel Core i7 2600
Core clock	3.4GHZ
TDP	95W
Codename	Sandy bridge
OS	Ubuntu 12.04 LTS, Linux kernel 3.2.0
Measurement	ZM-PCM1

표 1 대상 시스템

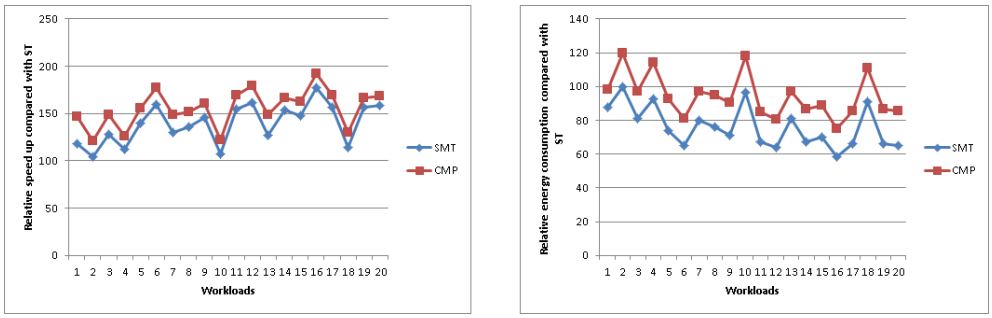


그림 9 Core i7 2600의 성능 및 에너지 소비 측정 결과

실험은 다양한 연산을 하는 2 thread 벤치 프로그램 20종류를 같은 코어의 다른 hardware thread를 이용하여 실행함으로서 SMT의 성능과 에너지 소비를 측정하였고, 같은 프로그램들을 다른 코어에서 각각 실행함으로서 CMP의 성능과 에너지 소비를 측정하였다.

그림 9의 왼쪽은 ST 구조 대비 증가되는 성능을 백분율로 나타낸 것이다. 항상 CMP 구조가 SMT 구조보다 성능이 높으며, CMP나 SMT 둘 다 workloads에 따라 증가되는 성능의 편차가 심함을 알 수 있다. 오른쪽 그림은 workloads에 따른 에너지 소비를 나타내는 도표로서 SMT 구조가 CMP 구조보다 에너지 소비가 적음을 알 수 있다.

위의 내용을 종합하면 시스템의 성능은 CMP > SMT > ST 구조 순이며, 에너지 효율은 SMT > CMP > ST 구조 순임을 알 수 있다. 따라서 에너지 소비를 줄이기 위해서는 CMP보다는 SMT 구조를 적극 사용해야 한다.

2.4 Linux의 로드 밸런싱 메커니즘과 문제점

이상에서 살펴본 바와 같이 기존에는 SMT vs. CMP의 에너지 효율을 하나로 결론 내리기 쉽지 않았다. 때문에 현재 OS들의 로드 밸런싱 메커니즘은 최대의 성능을 내는데 초점을 맞추고 있다. 그러나 이것은 CMP 기술보다 에너지 효율이 높은 최신 SMT 기술을 사용하는 프로세서에 적합하지 않다. 이 장에서는 Linux의 로드 밸런싱 메커니즘을 분석하여 기존 메커니즘의 문제점을 도출해 낸다.

Linux는 NUMA, Core, HT 총 세 가지의 스케줄링 도메인을 갖는다. 로드 밸런싱은 스케줄링 도메인 안에서 로드를 분배하는 과정으로 일반적인 시스템에서는 그림 10과 같이 core 도메인과 HT 도메인의 로드 밸런싱을 수행한다. HT 도메인 간에는 같은 L2 cache를 공유하기 때문에 로드 밸런싱을 통한 오버헤드가 적다. 반면에 core 도메인 간에는 L2 cache를 공유하지 않기 때문에 상대적으로 오버헤드가 크다. 따라서 HT 도메인 사이에는 매 1~2ms 간격으로 자주 로드 밸런싱을 수행하며 core 도메인 사이에는 HT 도메인보다 드물게 로드 밸런싱을 수행한다. 이는 그림 11처럼 태스크가 ST에서 SMT로, 다시 SMT에서 CMT 구조로 분배되거나 혹은 ST에서 CMP 그리고 CMT 구조로 분배되는 것으로 나타낼 수 있다. 두 경우에서 보듯이 Linux의 로드 밸런싱 목표는 가능한 많은 코어를 사용하는 것이다. 즉 리눅스는 SMT와 CMP 구조 모두를 적극 활용하려고 한다. 그렇기 때문에 SMT 하나만 사용하는 것보다 에너지 효율이 떨어지는 문제가 발생하게 된다.

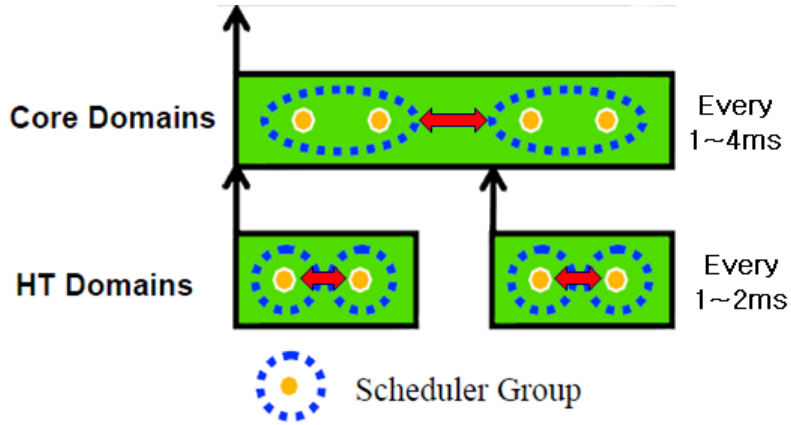


그림 10 Linux의 로드 밸런싱 메커니즘

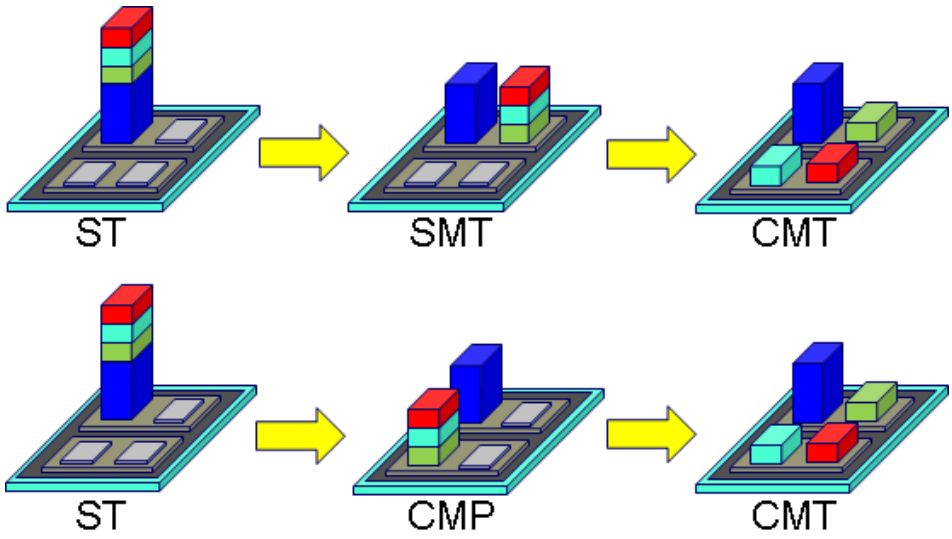


그림 11 Linux의 일반적인 로드 밸런싱 순서

제 3 장 에너지 인지 로드 밸런싱 메커니즘

2장에서 Linux에서의 로드 밸런싱은 시스템의 성능을 높이기 위함이며, 이 때문에 SMT 구조를 적극 활용하는 것보다 에너지 효율이 떨어짐을 보였다. 물리적으로 이러한 성능-에너지 트레이드오프는 극복할 수 없는 것처럼 보인다. 이 장에서는 사용자들이 실제로 느끼는 성능인 체감성능이란 지표를 정의하고 light load condition에서는 체감성능의 저하 없이 에너지 효율을 높일 수 있음을 보이며, 이를 바탕으로 에너지 인지 로드 밸런싱 메커니즘을 제안한다.

3.1 체감성능

컴퓨터 사용자들은 보통 태스크를 수행하는 시스템의 성능 보다 태스크를 완료하는 시간으로 시스템을 평가한다. 반면 많은 태스크들이 CPU 자원을 100% 이용하기가 쉽지 않으며 대부분 50% 이하의 이용률을 보인다[15]. 이러한 프로세서 이용률의 공백을 활용하면 태스크를 완료하는 시간을 지연시키지 않고 에너지 효율을 높일 수 있다.

이를 위해 태스크 완료시간을 기준으로 성능을 평가하는 체감성능(Sensible Performance, SP)이라는 지표를 도입한다.

$$Sensible\,Performance = \frac{Instructions\,of\,task}{Duration} \quad \cdots\cdots\cdots\cdots ①$$

$$Normalized\,Sensible\,Performance = \frac{Duration\,on\,ST}{Duration\,on\,the\,target\,system} \quad \cdots ②$$

식 ①처럼 체감성능은 태스크를 완료하는데 걸리는 시간(Duration) 동안 실행한 Instruction의 수라고 정의한다. 그림 12에서 보듯이 execution time은 태스크 완료시간 중 실제로 태스크가 실행되는 시간을 의미하고 duration은 태스크 실행이 시작된 시간과 태스크 완료 시간의 차이를 의미한다. 따라서 SP는 duration의 역수에 비례하고 performance는 execution time의 역수에 비례하게 된다. 실제 체감성능은 식 ②처럼 ST 구조에서 실행된 값을 기본으로 정규화한 수치로 표현한다.

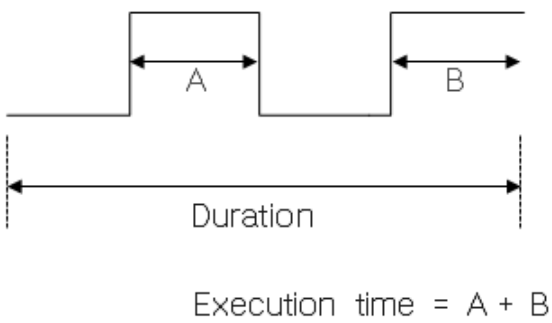


그림 12 CPU 이용률이 50%인 태스크에서 duration과 execution time

일반적으로 마이크로아키텍처의 성능은 IPC(Instructions per cycle)로 나타낸다. 대부분의 벤치마킹 프로그램은 CPU bound 태스크를 수행하는 동안 IPC 값을 측정한다. 그러나 사용자들에 의해 발생되는 많은 태스크들은 CPU bound 태스크가 아니며 대부분 50% 이하의 CPU 이용률을 보인다[15]. 이는 IPC 상의 성능과 사용자들이 느끼는 성능과의 괴리를 발생시킨다.

그림 13에서의 task set은 CPU를 100% 이용할 수 있는 두 개의 태스크들로 구성되어 있다. 이 태스크 A와 B를 한 core의 한 HT에서 실행을 하게 되면 직렬적으로 실행을 해야 하기 때문에 실행시간이 두 배로 길

어진다. 반면에 한 core의 두 개의 HT로 나뉘어서 실행는 SMT의 경우 성능상의 이득이 25%가 있다면 실행시간이 20%로 줄어든다. CMP의 경우에는 두 개의 task thread를 두 개의 코어에 각각 나누어 실행하기 때문에 이상적인 경우에 ST 실행시간의 반밖에 걸리지 않는다. 이처럼 CPU 이용률이 100%인 경우는 task execution time과 duration이 일치하기 때문에 SP와 performance는 동일한 값을 갖게 된다.

반면에 그림 14처럼 CPU를 50% 밖에 이용할 수 없는 특성을 갖는 task set을 실행할 경우에 다른 결과를 얻을 수 있다. ST의 경우 CPU 이용률이 50%이기 때문에 태스크를 한 개 실행하나 두 개 실행하나 같은 duration을 갖는다. 마찬가지로 SMT와 CMP에서도 태스크를 실행하는 방법은 다르지만 동일한 duration을 갖는다. 이를 바탕으로 체감성능을 계산하면 ST와 SMT 그리고 CMP는 모두 동일한 값을 가진다. 반면 태스크를 수행하는데 CPU가 사용한 cycle 수는 변화가 없으므로 성능상의 변화가 없기 때문에 사용자들은 IPC 성능과 체감성능의 괴리를 느끼게 된다.

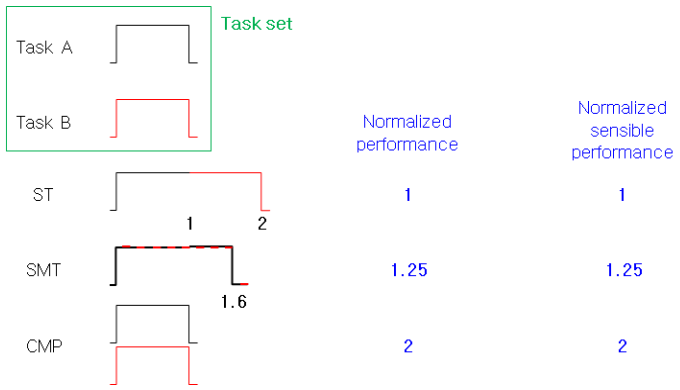


그림 13 CPU 이용률이 100%인 태스크에서 performance와 SP의 관계

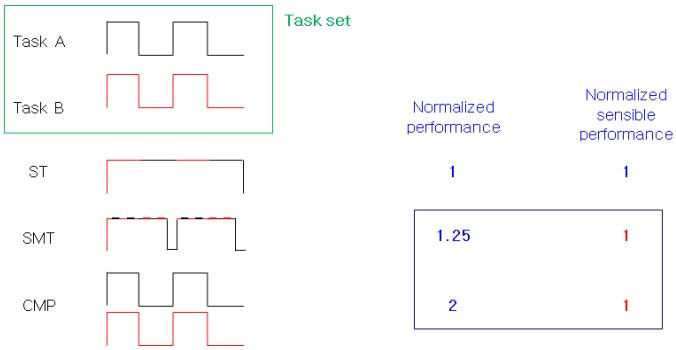


그림 14 CPU 이용률이 50%인 태스크에서 performance와 SP의 관계

위와 같은 결과는 light load condition(코어의 CPU 이용률이 50% 이하인 경우)에서 에너지 효율이 가장 높은 SMT 구조를 적극 활용할 경우 에너지 소비를 줄일 수 있다는 것을 알려준다.

다음 표 2는 CPU 이용률 40%를 갖는 두 태스크를 ST, SMT와 CMP 구조에서 실행하였을 때 소비하는 에너지를 측정한 결과이다. ST의 경우 CPU 이용률이 매우 높아 평균 소비전력 또한 높음을 알 수 있고 SMT와 CMP는 구조의 특성상 평균 소비전력이 낮게 나옴을 알 수 있다. 수행시간은 ST와 CMP가 동일하였으며, SMT의 경우 두 개의 thread의 경쟁으로 인한 오버헤드 때문에 다소 시간이 지연되는 것을 알 수 있다. 이를 통하여 총 소비한 에너지를 구하면 SMT의 에너지 소비가 가장 적은 것을 확인할 수 있다.

구 분	평균 소비전력	수행시간	에너지
ST	24 W	21.12 sec	517.44 J
SMT	16 W	22.56 sec	360.96 J
CMP	20 W	21.35 sec	427 J

표 2 Core i7의 성능 및 에너지 소비 측정 결과

그림 15는 태스크의 CPU 이용률에 따른 체감성능의 변화를 나타낸 그래프이다. 그림 14에서 보인 데로 태스크의 CPU 이용률이 50%에 가까워지면서 SMT와 CMP의 체감성능 차이가 7% 이내로 작아지고 40% 이하인 경우 5% 이내로 차이가 거의 없음을 알 수 있다. 반면 그림 16에서 태스크의 CPU 이용률이 50% 이하일 경우 SMT의 에너지 효율이 CMP인 경우 보다 평균 15% 정도 좋음을 알 수 있다. 따라서 CPU 이용률이 50% 이하인 태스크의 경우 CMP 구조로 실행하는 것 보다 SMT 구조로 실행하는 것이 에너지 소비를 줄일 수 있는 더 나은 방법이다.

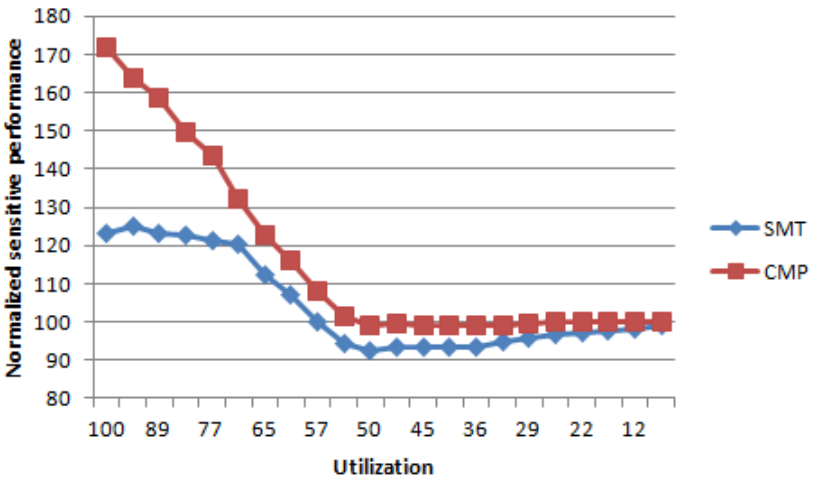


그림 15 태스크의 CPU 이용률에 따른 체감성능의 변화

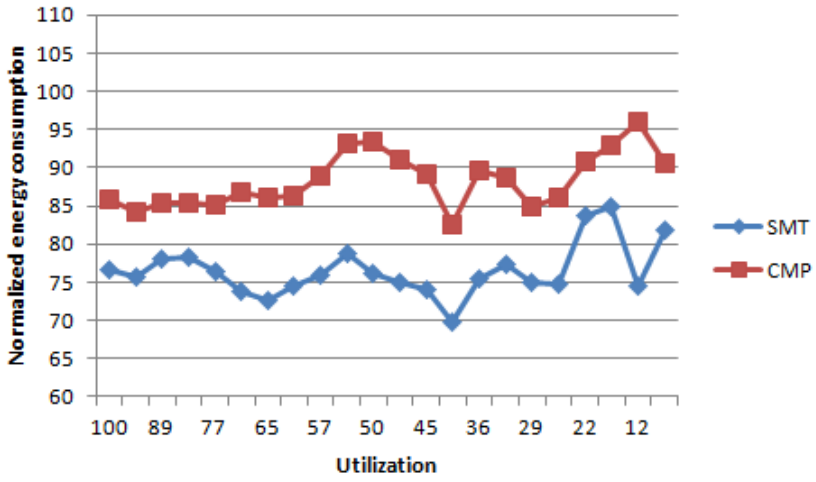


그림 16 태스크의 CPU 이용률에 따른 에너지 소비의 변화

3.2 에너지 인지 로드 밸런싱 메커니즘

3.1장에서는 CPU 이용률이 50% 이하인 태스크의 경우 SMT 구조로 실행하는 것이 CMP 구조로 실행하는 것 보다 에너지 효율이 더 높음을 보였다. 이러한 것을 활용하여 이 장에서는 CMT 구조에서 에너지 소비를 줄이기 위한 에너지 인지 로드 밸런싱을 메커니즘을 제안한다. 에너지 인지 로드 밸런싱을 하기 위한 요구조건은 다음과 같다.

① Core 도메인의 로드 밸런싱을 할 때 light load condition 일 경우에는 로드 밸런싱을 하지 않는다.

② CPU 이용률을 가장 작게 차지하는 thread의 경우 SMT를 최대한 활용할 수 있는 core가 있는 경우 그 core로 태스크를 이동한다. 최대한 활용할 수 있는 조건은 다음과 같다.

- 태스크를 옮길 physical core의 평균 이용률이 옮겼을 경우 50%가 넘지 않을 것.
- 태스크를 옮겼을 때 같은 core 도메인의 태스크들의 load가 균형적인 상황일 것.

이를 바탕으로 만든 알고리즘은 다음과 같다.

At every core domain load balancing time

find the busiest core

if(Average utilization of a physical core > 50%) ・・・・・①

do core domain load balancing

else ・・・・・②

find the most idle core which has one task in core domain

If(When the suitable task moves to the second idle core,

utilization of the core satisfies light load condition)

move the task to the second idle core

①, ② 부분이 에너지 인지 로드 밸런싱 메커니즘에 추가된 부분이다.

제 4 장 실험 결과

4.1 시스템 구현

설계한 로드 밸런싱 메커니즘은 Linux 3.2.0 kernel 상에 구현하였고 인텔 최신 마이크로아키텍처인 Sandy bridge 프로세서를 사용하는 시스템에서 실험을 하였다. 구체적인 실험환경은 다음과 같다.

구 분	Desktop
Processor name	Intel Core i7 2600 @3.4 GHz
Micro architecture	Sandy bridge, 32nm
L2 cache	256KB × 4
L3 cache	8MB
The number of thread	2 hardware threads of each core
Physical core	4 cores
TDP	95W
Measurement	ZM-PCM1 (CPU power consumption meter)
OS	Linux(Ubuntu 12.04 LTS) Kernel ver. 3.2.0

표 3 실험 환경

4.2 실험 시나리오

사용자의 체감성능을 측정하기 위해 사용자들이 실생활에서 자주 사용하는 workloads를 기준으로 실험을 하였다.

Workloads : webpage loading
동영상 및 MP3 재생
파일 압축
사진 loading

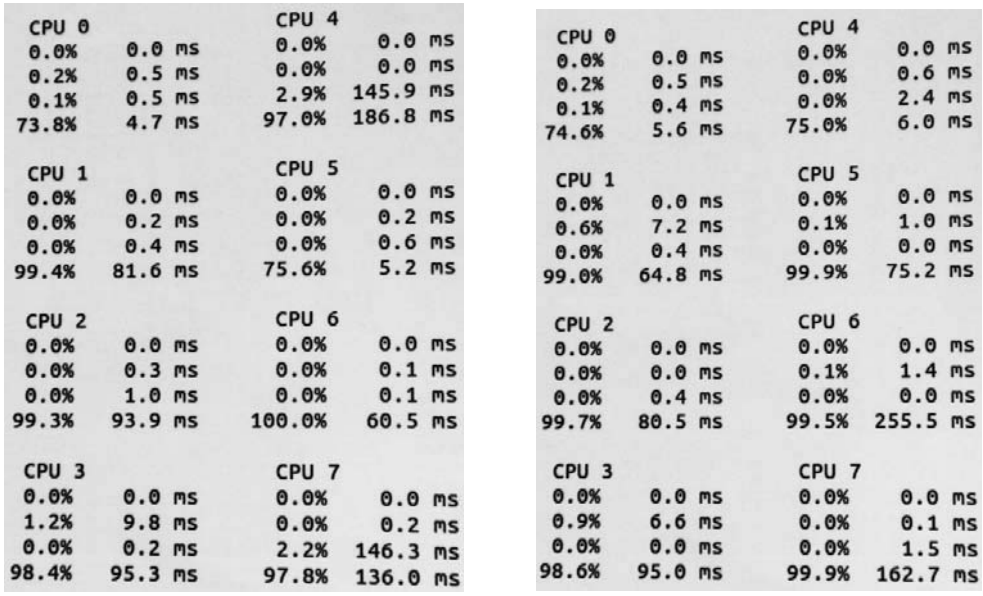
에너지 인지 로드 밸런싱 메커니즘을 적용하기 전 / 후에 각각의 워크로드에 따른 총 소비되는 에너지를 측정하였다. Webpage loading은 2개의 웹브라우저를 실행하고 webpage를 지속적으로 읽어 들이는 방식으로 테스트 하였다. 동영상 및 MP3 재생은 두 개의 파일을 동시 재생하였고 파일압축은 두 개의 파일을 동시에 압축하였으며 사진 loading은 여러 개의 파일을 동시에 실행하는 방법으로 실험을 하였다.

4.3 실험 결과

위와 같이 실험을 할 경우 메커니즘 수정 전에는 각 태스크가 물리적으로 다른 core에 각각 할당될 것이며, 수정 후 에너지 인지 로드 밸런싱 메커니즘이 정상적으로 동작한다면 태스크가 물리적으로 같은 코어 내의 다른 논리 코어에 할당이 되어야 한다.

그림 17은 에너지 인지 로드 밸런싱 메커니즘 적용 전 / 후 CPU idle time을 powertop 프로그램으로 측정한 자료로서 CPU 이용률은 idle time

을 100%에서 차감하여 계산할 수 있다. CPU0와 4, CPU1과 5, 2와 6, 3과 7이 각각 물리적으로 동일한 코어내의 논리 코어이다. 적용 전에는 CPU0가 26.2%, CPU5가 24.4%로 CMP 구조를 정상적으로 사용하고 있다. 적용 후에는 CPU0가 25.4%, CPU4가 24.4%로 light load condition에서 태스크들을 SMT 구조로 실행하는 것을 확인할 수 있다.



CPU 0		CPU 4	
0.0%	0.0 ms	0.0%	0.0 ms
0.2%	0.5 ms	0.0%	0.0 ms
0.1%	0.5 ms	2.9%	145.9 ms
73.8%	4.7 ms	97.0%	186.8 ms
CPU 1		**CPU 5**	
0.0%	0.0 ms	0.0%	0.0 ms
0.0%	0.2 ms	0.0%	0.2 ms
0.0%	0.4 ms	0.0%	0.6 ms
99.4%	81.6 ms	75.6%	5.2 ms
CPU 2		**CPU 6**	
0.0%	0.0 ms	0.0%	0.0 ms
0.0%	0.3 ms	0.0%	0.1 ms
0.0%	1.0 ms	0.0%	0.1 ms
99.3%	93.9 ms	100.0%	60.5 ms
CPU 3		**CPU 7**	
0.0%	0.0 ms	0.0%	0.0 ms
1.2%	9.8 ms	0.0%	0.2 ms
0.0%	0.2 ms	2.2%	146.3 ms
98.4%	95.3 ms	97.8%	136.0 ms

(a)적용 전

CPU 0		CPU 4	
0.0%	0.0 ms	0.0%	0.0 ms
0.2%	0.5 ms	0.0%	0.6 ms
0.1%	0.4 ms	0.0%	2.4 ms
74.6%	5.6 ms	75.0%	6.0 ms
CPU 1		**CPU 5**	
0.0%	0.0 ms	0.0%	0.0 ms
0.6%	7.2 ms	0.1%	1.0 ms
0.0%	0.4 ms	0.0%	0.0 ms
99.0%	64.8 ms	99.9%	75.2 ms
CPU 2		**CPU 6**	
0.0%	0.0 ms	0.0%	0.0 ms
0.0%	0.0 ms	0.1%	1.4 ms
0.0%	0.4 ms	0.0%	0.0 ms
99.7%	80.5 ms	99.5%	255.5 ms
CPU 3		**CPU 7**	
0.0%	0.0 ms	0.0%	0.0 ms
0.9%	6.6 ms	0.0%	0.1 ms
0.0%	0.0 ms	0.0%	1.5 ms
98.6%	95.0 ms	99.9%	162.7 ms

(b)적용 후

그림 17 에너지 인지 로드 밸런싱 메커니즘 적용 전 / 후 CPU idle time 측정 data

그림 18은 Webpage loading test시 CPU의 소비전력을 측정한 그래프이다. 가로축은 테스트 진행시간을 나타내며 세로축은 전력을 나타낸다. 에너지는 전력과 시간의 곱으로 표현되므로 그래프의 아래 면적이 테스트 시 소비된 에너지를 나타낸다. 그림에서 보듯이 적용전보다 적용후의 평균적인 전력이 낮게 나타남을 알 수 있고 적용 전 / 후 똑같은 실행시

간을 갖기 때문에 적용 후의 에너지 소모가 작음을 알 수 있다.

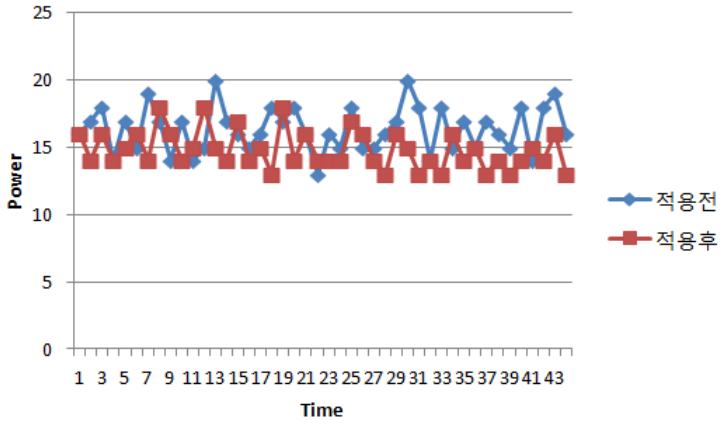


그림 18 Webpage loading test 시 CPU 소비 전력의 변화

그림 19와 20은 동영상 및 MP3 재생 시 CPU의 소비전력을 측정한 그래프이다. 동영상과 MP3 재생 테스트에서도 마찬가지로 적용 후에 평균 소비전력이 줄어들었다.

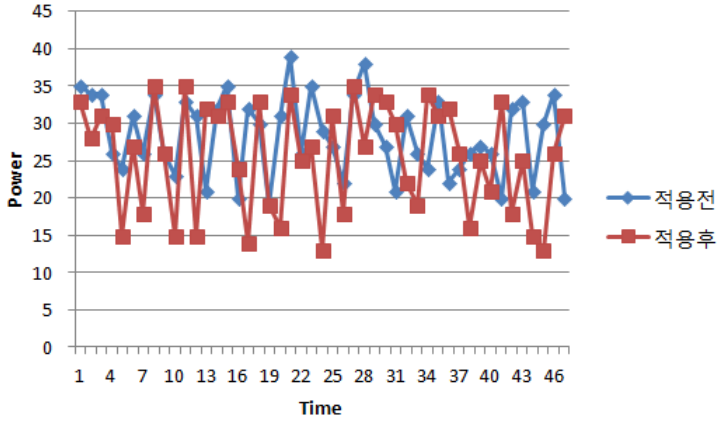


그림 19 동영상 재생 시 CPU 소비 전력의 변화

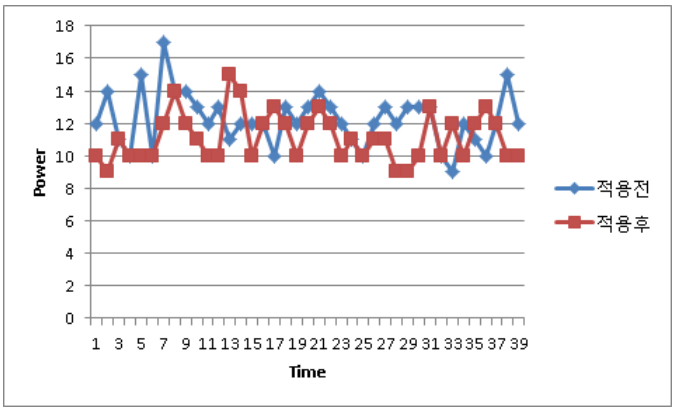


그림 20 MP3 재생 시 CPU 소비 전력의 변화

그림 21은 파일 압축 시 CPU의 소비전력을 측정한 그래프이다. 그림 21은 앞의 그림 18과 19 그리고 20과는 다르게 적용 전 / 후의 소비전력 차이가 없음을 알 수 있다. 이는 파일 압축의 경우 CPU 자원을 많이 사용하는 태스크로서 CPU 이용률이 50% 이상이기 때문에 에너지 인지 로드 밸런싱 메커니즘이 동작하지 않기 때문이다.

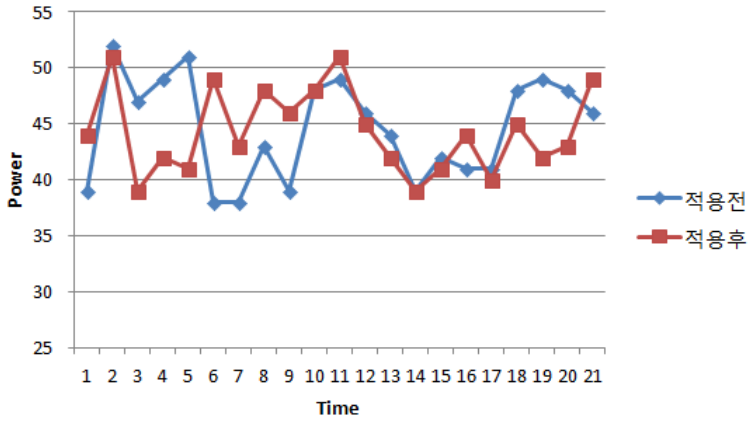


그림 21 파일 압축 시 CPU 소비 전력의 변화

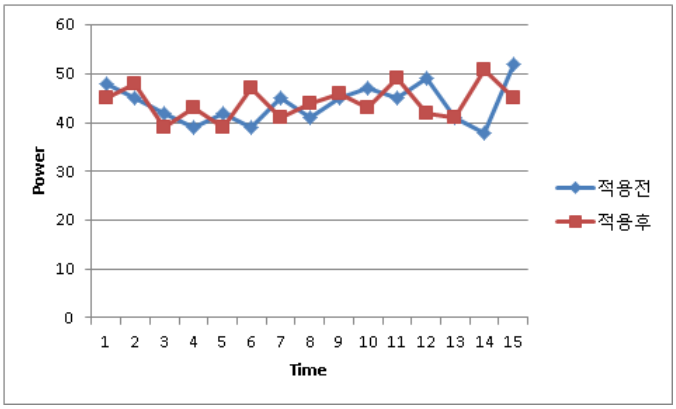


그림 22 대용량 사진파일 loading 시 CPU 소비 전력의 변화

그림 22는 대용량 사진파일 loading 시 소비전력을 측정한 그래프이다. 파일압축 테스트와 마찬가지로 소비전력의 차이가 발생하지 않는다.

위의 결과 정리하면 표 4와 같은 결과를 얻을 수 있다. 이를 바탕으로 webpage loading이나 동영상 및 MP3 재생 등 CPU 사용률이 많지 않은 light load condition이 해당되는 태스크에서는 에너지 인지 로드 밸런싱 메커니즘을 적용할 경우 약 10%의 CPU 소비 에너지를 줄일 수 있음을 알 수 있다.

구 분	적용 전	적용 후	변화율
Webpage loading	721 J	652 J	- 9.6 %
동영상 재생	1334 J	1204 J	- 9.8 %
MP3 재생	477 J	435 J	- 8.8 %
파일 압축	937 J	932 J	- 0.5 %
사진 loading	658 J	663 J	0.8 %

표 4 각 workloads에 따른 메커니즘 적용 전 / 후 에너지 변화

제 5 장 결론

범용 프로세서 시장에서 CMT 구조를 가진 프로세서가 급속히 보급되고 있다. 그에 따라 SMT와 CMP에 관련된 연구들이 1990년대와 2000년대 초에 걸쳐 활발히 진행됐었다. 그러나 대부분의 연구들은 SMT와 CMP의 성능 및 에너지 효율에 대한 비교였으며 이를 활용하는 몇몇 논문들도 SMT 구조에서의 성능 개선에 집중하였고 SMT의 우수한 에너지 효율을 이용하기 위한 시도는 드물었다. 게다가 2000년대 초는 SMT 기술을 상업적으로 도입한 초기였기 때문에 시스템과 workload에 따라 에너지 효율이 크게 달라져 SMT의 에너지 효율성을 하나로 결론 내리기가 쉽지 않았다.

그러나 Intel사의 최신 Sandy bridge 아키텍처에서는 이렇게 혼선을 일으키던 결과가 하나로 일치함을 보여 SMT의 높은 에너지 효율성을 사용하기 적합한 시기가 도래하였다. 그러나 SMT 구조는 일반적으로 CMP 구조에 비해 에너지 효율성은 좋지만 throughput이 떨어져 단순히 SMT 기술을 사용할 경우 성능의 저하를 피할 수 없다.

본 논문은 벤치마킹 프로그램에서 SMT와 CMP 구조의 우위를 비교하는 것 보다 사용자들이 실제 사용하는 응용에서의 체감 성능과 에너지 효율성을 분석하였다는데 의의가 있다. 본 논문에서 제시한 에너지 인지 로드 밸런싱 메커니즘의 가장 큰 특징은 사용자들이 체감성능의 저하를 느끼지 못하는 light load condition에서 CMP 구조 대신 SMT 구조로 응용을 실행하도록 CPU의 로드를 분산시키는 것이다. 이를 위해 사용자들의 체감성능을 나타내는 지표인 SP를 정의하고 CPU 이용률에 따른 SP 변화를 확인하였다. 이를 통해 태스크의 CPU 이용률이 50% 이하인 경우 SP의 저하가 7% 이내로 나타나고 40% 이하인 경우 5% 이내임을 보였

다. 또한 제안하는 에너지 인지 로드 밸런싱 메커니즘을 Linux 시스템에 구현하여 사용자들이 자주 쓰는 세 가지 workload에서 체감성능의 저하 없이 SMT가 CMP 구조 대비 10%의 에너지 절감 효과가 있음을 검증하였다.

참고문헌

[1] Dean M. Tullsen, Susan J. Eggers and Henry M. Levy, “Simultaneous Multithreading: Maximizing On-Chip Parallelism”, International Symposium on Computer Architecture, 1995.

[2] Lance Hammond, Basem A. Nayfeh and Kunle Olukotun, “A Single-Chip Multiprocessor”, Computer of IEEE, 1997.

[3] Ruchira Sasanka, Sarita V. Adve, Yen-Kuang Chen and Eric Debes, “The Energy Efficiency of CMP vs. SMT for Multimedia Workloads”, Intel Technical Report 130581, 2003.

[4] Yingmin Li, David Brooks, Zhigang Hu and Keving Skadron, “Performance, Energy, and Thermal Considerations for SMT and CMP Architectures”, Proceedings of the 11th Int’l Symposium on High-Performance Computer Architecture, 2005.

[5] Allan Snavely and Dean M. Tullsen, “Symbiotic Jobscheduling for a Simultaneous Multithreading Processor”, Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, 2000.

[6] Sujay Parekh, Susan Eggers, Henry Levy and Jack Lo, “Thread-Sensitive Scheduling for SMT Processors”, Technical report, University of Washington, 2000.

[7] Matthew Devuyst, Rakesh Kumar, and Dean M. Tullsen, “Exploiting Unbalanced Thread Scheduling for Energy and Performance on a CMP of SMT Processors CMP of SMT Processors”, International Parallel and Distributed Processing Symposium, 2006.

[8] Petar Radojkovic, Vladimir Cakarevic, Miquel Moretto, Javier Verdu, Alex Pajuelo, Francisco J. Cazorla, Mario Nemirovsky and Mateo Valero “Optimal Task Assignment in Multithreaded Processors: A Statistical Approach”, Architectural Support for Programming Languages and Operating Systems, 2012.

[9] Venkatesan Packirisamy, Yangchun Luo, Wei-Lung Hung, Antonia Zhai, Pen-Chung Yew and Tin-Fook Ngai, “Efficiency of Thread-Level Speculation in SMT and CMP Architectures - Performance, Power and Thermal Perspective”, University of Minnesota - Computer Science and Engineering Technical Report, 2008.

[10] Robert Schöne, Daniel Hackenberg and Daniel Molka, “Simultaneous Multithreading on x86_64 Systems: An Energy Efficiency Evaluation”, 4th Workshop on Power-Aware Computing and Systems (HotPower), 2011.

[11] Tom’s guide team, “The Mother of All Charts 2005/2006”, http://www.tomshardware.com/reviews/mother-cpu-charts-2005,1175.html, 2005.

[12] Kevin Krewell, “INTEL EMBRACES MULTITHREADING”, Microprocessor Report of www.MDRonline.com, 2001.

[13] John L. Manferdelli et al, “Challenges and Opportunities in Many-Core Computing, Vol. 96, No. 5, Proceedings of the IEEE, 2008.

[14] Subhash Saini, Haoqiang Jin, Robert Hood, David Barker, Piyush Mehrotra, Rupak Biswas, “The Impact of Hyper-Threading on Processor Resource Utilization in Production Applications”, 18th International Conference on High Performance Computing, 2011.

[15] Bruno Abrahao and Alex Zhang, “Characterizing Application workloads on CPU Utilization or Utility Computing” HP Technical reports, 2004.

Abstract

Those days almost all general purpose processors are a CMP(Chip Multi-Processor) architecture. Recently SMT(Simultaneous Multi-Threading) and CMP technology are rapidly consolidated to improve performance with minimum cost. Usually the architecture name is called CMT(Chip Multi-Threading). CMT processor are occupying over 50% market share of general purpose processors. 64% of processors produced by Intel which are occupying over 90% market share of general purpose processors are CMT architecture with using Hyper Threading Technology as SMT technology. Also AMD are producing Bulldozer architecture processors with SMT which are different from Intel's. And I expected that we will use SMT architecture in mobile AP to overcome performance limitation.

There are many research to study SMT technology as long as widely distributed CMT. But almost all research of SMT related focus on improving performance. It is rare for energy efficient to study. Energy efficiency centered research mainly concentrates relation of ST, SMT and CMP.

This paper addresses a method to reduce energy consumption on CMT architecture instead of improving performance and architecture comparison.

In CMT architecture which selectively uses both SMT and CMP, we can select whether using CMP or SMT. Normally we can get the best throughput in case of two threads which execute in separated

core(CMP). In contrast, when two thread are executed separated HT on the same core, we can get lower throughput than CMP. But energy consumption of SMT is better than CMP.

Many OSs select to use performance centered configuration between performance and energy efficiency. And OSs didn't care about energy efficiency for SMT architecture. This paper shows that legacy OSs are missing a chance to reduce energy consumption without sensible performance decrease. I suggest an energy aware load balancing mechanism to solve the problem. Also I implemented the suggested algorithm on Linux. Then I verified when we use the solution in case of light load condition, we can reduce about 10% energy consumption without sensible performance decrease.

Keywords: load balancing, energy aware, SMT, CMP, CMT

Student Number: 2010-23958