운영체제 레벨의 DFS에 기반하는 온도를 고려한 스케줄링

정성우
고려대학교 정보통신대학 컴퓨터·통신학부
swchung@korea.ac.kr

KISS Korea Computer Congress 2006

Sung Woo Chung
Division of Computer Science and Engineering, Korea University

요 약

프로세서의 온도를 낮추기 위한 컴퓨터 과학적 접근법으로는 가변전압주파수조절(DVFS), 파이프라인에서 더 이상 명령어를 수행하지 못하게 하는 방법(pipeline throttling) 등이 있다. 하지만, 이러한 해결책은 대부분 소수의 온도 센서가 내장되어 있어 이를 기반으로 온도를 제어하였다. 본 논문에서는 실제 Pentium 4에 기반한 시스템을 통하여, 운영체제 레벨의 가변주파수방법(DFS)을 이용한 스케줄링이 여러 개의 온도센서를 사용하여 국지화된 뜨거운 부분(localized hotspot)을 얼마나 효율적으로 온도를 제어할 수 있는지를 보여준다.

1. 서 론

프로세서에서 온도가 임계 온도를 넘어서면 프로세서의 정확한 동작을 보장할 수 없다. 그러므로 프로세서에서 전력소모를 줄이는 것뿐만 아니라, 온도를 제어하는 것이 중요한 문제이다. 온도를 제어하기 위해서 현재의 온도가 얼마인지를 알아야 하므로, 온도를 측정하는 센서가 필수적이다. Pentium 4에서는 2개의 온도 센서가 내장되었고 [1], 앞으로 출시될 Intel의 차세대 프로세서에는 온도센서가 10개 이상 탑재될 것으로 기대된다. 공정 기술이 미세화됨에 따라서 단위 면적당 전력 소모량이 늘어나고, 그로 인하여 뜨거워지는 국지적인 지역이 있을 가능성이 늘어나게 된다. 그러므로 프로세서 칩 전체에서 온도를 효율적으로 제어하기 위해서는 이러한 여러 개의 온도 센서가 필요하다.

온도가 임계온도를 넘지 않게 하기 위한 기계공학적 해결책으로는 외부 팬(fan)을 동작시켜 칩의 온도를 낮추는 방법, 방열판의 면적을 넓히는 방법 등이 있다. 이미 Intel의 프로세서들은 큰 방열판과 냉각팬을 필수적으로 내장하고 있으며, 심지어 PowerMac G5의 Quad 프로세서는 엄청난 냉각 펌프를 동반하고 있다[2]. 이러한 기계공학적인 해결책에도 불구하고, Intel의 새로운 프로세서에 대한 프로젝트가 온도 문제를 해결하지 못하여 취소되기도 하였다[4]. 즉, 기계공학적 해결책 이외에 다른 접근법이 필요하게 된 것이다. 컴퓨터 과학적 접근법으로는 가변전압주파수조절(DVFS : Dynamic Voltage Frequency Scaling)을 사용하여 시간 당 수행 명령어를 줄이는 방법과 파이프라인에서 더 이상 명령어를 수행하지 못하게 하는 방법(pipeline throttling) 등이 제시되었다[5]. 하지만, 온도를 낮추는 컴퓨터 과학적인 해결책은 대부분 하나의 온도 센서에 의존해왔다. 여러 개의 온도 센서를 이용한 방법들은 대부분 명령어 단위의 스케줄링을 사용해왔다. 예를 들어 명령어를 10개 수행한 후 임계 온도(threshold temperature)를 넘기 때문에 낮은 전압(낮은 주파수)으로 DFS를 수행한 후, 다른 명령어 10개를 수행한 후에 다시 원래의 전압으로 돌아오는 루프를 계속해서 수행한다고 가정하자. 이 때, 명령어 단위의 스케줄링에서는 계속해서 10개의 명령어 마다 전압을 변화시키게 된다. 이것은 매우 효율적으로 보이지만, 온도 센서에 대한 샘플링 횟수가 보통 10ms인 것을 감안하면 이러한 세밀한 스케줄링이 얼마나 무의미한 것인지를 알 수 있다. 또한 이러한 세밀한 스케줄링으로 인하여 추가적으로 소모되는 전력과 그에 따른 온도 상승은 결코 무시될 수 없다. 이를 해결하기 위하여 본 논문에서는 운영체제 레벨에서 온도를 제어하는 방법을 제시한다.

2. 운영체제 레벨의 온도를 고려한 스케줄링

앞에서 언급했듯이, 기존의 칩에 부착된 온도계의 샘플링 단위가 10ms이고 또한 운영체제에서 스케줄링을 하는 granularity가 10ms(linux의 경우) 이므로 10ms 단위로 온도를 샘플링하여 스케줄링을 하는 방법이 적절하다. 그 온도를 고려한 스케줄링이 의미있다고 말하기 위해서는 스케줄링 단위인 10ms 내에서 큰 온도 변화가 없어야 한다.

수치적으로는 10ms에 33 도 단위로 온도가 상승할 수 있으나[3], 이는 악의적으로 특정 부분의 온도를 집중적으로 올렸을 때에 해당된다. [6]에 따르면 각각 Spec 벤치마크와 멀티미디어 응용(mp3, mpeg 등)에 있어서 10ms 사이에서 일어나는 온도 변화는 매우 제한적이다. 그러므

로 우리는 10ms 이내에서 큰 온도 변화가 없다는 것으로 생각할 수 있다. 만약 이 가정이 적용되지 않는 응용 프로그램이 있더라도, 운영체제 레벨의 온도를 고려한 스케줄링으로 인하여 명령어 레벨의 스케줄링에서 온도를 고려해야 하는 부담을 덜어준다는 것은 자명하다.

본 연구의 목적은 운영 체제 레벨의 스케줄링이 온도 제어에 유용하다는 것을 보이는 것이다. 이를 위해 DVS를 이용한 스케줄링의 효율성을 보이려고 한다. 이를 위해서 임의의 임계 온도를 설정하고 실제 온도가 임계 온도에 도달했을 경우, 클럭 주파수를 80%로 설정하여 온도를 내리고 임의의 임계 온도에 도달하지 않았을 경우, 5%씩 클럭 주파수를 올리도록 한다. 이 때 전압 또한 내려서 온도를 줄이는 이중 효과를 볼 수 있다. 하지만, 본 연구에서는 전압을 강하하여 온도를 낮추는 효과는 제외하고 주파수를 강하시켜서 온도를 낮추는 효과에만 집중하기로 한다.

3. 스케줄링의 효율성 평가

본 연구에서는 하이퍼쓰레딩 (hyperthreading) 기법을 적용한 Pentium 4 프로세서 시스템을 기반으로 한다. Pentium 4 프로세서 시스템에서 직접 온도 센서를 읽으면서 실험하기 위해서는 프로세서의 주파수를 자유자재로 변형할 수 있어야 한다. 하지만, 그것이 자유롭지 못하므로 Pentium 4 프로세서가 제공하는 각각의 로직 블록에 대한 접근 회수를 얻는다. 그 접근 회수와 Pentium 4 프로세서에 대한 플루어플랜(floorplan)을 HotSpot[7]에 입력하여 각각의 로직 블록에 대한 온도를 구한다. 모든 로직 블록의 온도를 모두 고려하는 것은 너무 복잡하므로, 대부분의 응용 프로그램에서 로직 블록 중에서 가장 높은 온도를 보이는 정수 레지스터 파일을 온도를 제어해야 할 하나의 로직 블록으로 가정하였다.

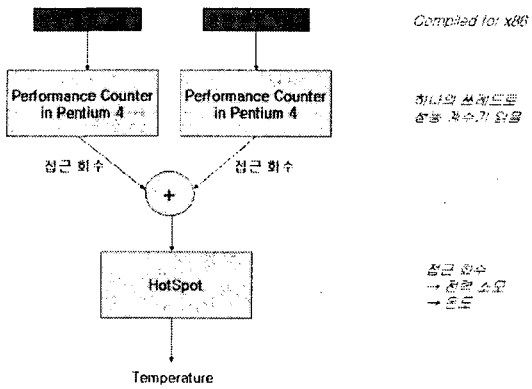

그림 1. 시뮬레이션 방법

Spec 벤치마크의 한 응용 프로그램을 수행했을 때, 온도의 변화가 단조로워서, 응용 프로그램 두 개를 동시에 수행하는 것을 시뮬레이션하였다. 이를 위해서 그림 1에 나타난 것처럼, 각각의 응용 프로그램을 Pentium 4 프로세서에서 수행하고, 각각에 대해서 로직 블록에 대한 접근 회수를 성능 계수기로부터 읽어서 파일에 기록해둔다. 사용한 프로세서가 하이퍼스레딩 기법을 적용하므로, 하나의 쓰레드(thread)는 응용 프로그램을 수행하면서 다른 하나의 쓰레드는 성능계수기를 읽는 기능을 수행하는 것이 큰 오버헤드는 되지 않는다. 각각의 응용 프로그램에 대해서 모든 로직 블록에 대한 접근 회수를 더하여 총 접근 회수를 구한다. 단지 온도를 관찰하려는 정수 레지스터 파일만이 아닌 모든 로직 블록에 대해서 접근 회수를 받아들이는 것에 주의한다. 정수 레지스터 파일의 온도는 그 자체에 대한 접근 회수 뿐만 아니라, 주변 로직 블록에 의해서도 영향을 받으므로 프로세서 전체의 로직 블록에 대한 접근 회수가 필요하다. 모든 로직 블록에 더한 접근 회수를 HotSpot에 입력하여 소모되는 전력과 온도를 구한다. 이렇게 함으로써, 다수 개의 응용 프로그램을 수행하는 시뮬레이션을 할 수 있다.

DFS를 적용하는 것을 평가하기 위해서, 실제 프로세서의 주파수를 조절하는 것은 힘들 뿐만 아니라 원하는 주파수로 조정하는 것이 불가능한 경우도 있다. 프로세서의 주파수를 낮추는 효과를 내기 위해서, 프로세서의 주파수를 낮추는 대신 정수 모든 로직 블록에 대한 접근 회수를 줄인다. 예를 들어, 프로세서의 주파수를 20% 떨어뜨린다면, 접근 회수를 20% 줄인다. 다음 시간 슬롯(time slot)에 대해서 온도를 구할 때는 이전에 수행되지 못한 20%를 포함하고, 해당 시간 슬롯의 60%를 수행하도록 하여 정확한 시뮬레이션이 되도록 하였다. HotSpot에서 사용되는 프로세서의 패키징 변수는 이전의 연구[7]에서 사용된 것과 같은 것을 사용하였다.

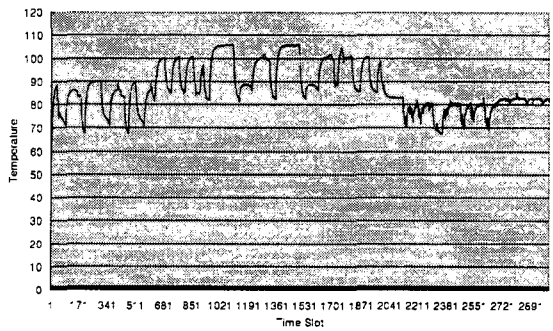

그림 2. DFS가 적용되지 않은 경우의 온도 변화

그림 2는 DFS가 적용되지 않았을 때의 온도 변화를 나타낸다. 실제 프로세서에서는 온도 제어를 하므로 온도가 이렇게 높을 수가 없지만, 인위적으로 높이기 위해서 앞에서 설명한 각각의 프로그램을 수행한 후 각각의 접근 회수를 더해서 온도를 구하였다. 보통 81.5도 정도를 임계 온도로 정하는 기준에서 벗어나는 시간이 많을 뿐 아니라, 최고 온도가 106도까지 다다른다.

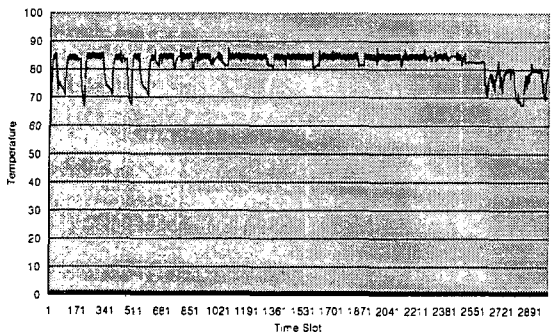

그림 3. DFS가 적용된 경우의 온도 변화
(임계 온도는 85도)

그림 3은 임계 온도가 85도이고, DFS가 적용되었을 때의 온도 변화를 나타낸다. 그림 2의 결과와 비교할 때, 임계 온도를 넘어선 시간이 확연히 줄어들었다. 수행 시간은 늘었지만, 이는 임계 온도를 넘어서지 않기 위한 불가피한 선택이다. DFS를 적용함으로써 큰 로드(load)가 일부의 시간 슬롯에 몰리는 것을 막아서 다수의 시간 슬롯으로 퍼지도록 한다. 주파수를 변경하는 것에 대한 성능 상의 오버헤드가 있지만, 각각의 시간 슬롯이 10 ms인 것을 감안하면 그 오버헤드는 무시될 정도로 작다.

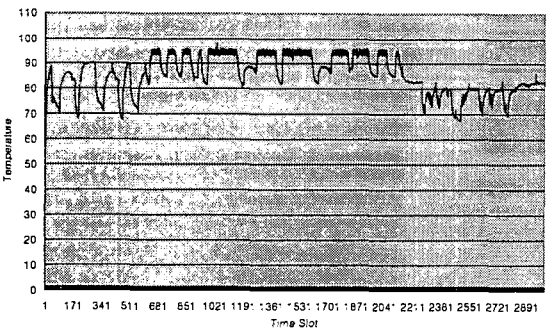

그림 4. DFS가 적용될 경우의 온도 변화
(임계 온도는 95도)

그림 4는 임계 온도가 95도이고, DFS가 적용되었을 때의 온도 변화를 보여준다. 그림 3에 비하여 임계 온도가 10도 높아졌다. 실제 온도가 임계 온도를 넘어선 시간은 DFS가 적용되지 않았을 경우와 비교했을 때, 현저하게 줄어들었다. 임계 온도가 10도 높아짐에 따라서 수행시간 증가는 급격하게 줄어들었다.

4. 결론과 향후 연구 방향

본 연구를 통해서 운영체제 레벨의 온도를 고려한 스케줄링, 특히 DFS가 얼마나 유용하게 이용될 수 있는지를 보였다. 이를 이용하여 임계온도를 넘어서지 않도록 할 수 있으며, 작업의 로드(load)를 시간 슬롯에 최대한 공평하게 분할함으로써 성능 저하를 최소화하였다.

명령어 단위의 온도를 고려한 스케줄링에 대해서는 이미 연구가 되었다. 하지만, 이미 제조된 프로세서 칩에서 과다한 온도로 인하여 문제가 생길 때, 그 프로세서 칩을 사용하기 위해서 운영체제 레벨의 스케줄링은 효과적인 해결책이 될 수 있을 것이다.

본 연구의 결과에서 고려하지 못한 부분이 있다. 본 연구에서는 클럭 주파수를 내림으로써 온도를 낮추는 효과만을 보았다. 하지만, 실제로는 클럭 주파수를 내리면 전압도 내릴 수 있다. 사용 전력은 전압의 제곱에 비례하므로 제안하는 방법을 사용하면 본 연구에서 제시된 것보다 더 빨리 온도를 낮출 수 있을 것이다. 곧 이를 고려한 추후 연구가 곧 뒤따를 것이다.

운영체제 레벨에서 온도를 고려한 스케줄링을 하는 것은 최신 연구분야이다. 실제 Pentium 4에 기반한 시스템에서의 성능계수기(performance counter)에서 얻은 활동 요소(activity factor)와 실제 온도를 회귀분석을 이용하여 연결시킨 이전 연구[8]와 본 연구를 총망라하여, 성능 계수기와 회귀분석으로 얻은 상수를 이용하여 온도를 예측하고 이를 실제 스케줄링에 이용하는 것이 얼마나 효율적인지 제시할 것이다. 또한 Spec 벤치마크 이외에도 mp3나 mpeg 등의 응용 프로그램에서도 효율적인지를 평가해 보는 것도 의미 있는 연구일 것이다. 온도에 대한 연구를 프로세서 코어뿐만 아니라 SOC(System On Chip)으로 확대해 보는 것 또한 흥미로운 연구 분야가 될 것이다.

참고문헌

[1] Intel Corporation. Thermal Zone Information. Available in http://support.intel.com/support/motherboards/desktop/sb/CS-12552.htm

[2] Apple Computer. Quad G5 2.5Ghz Processors. Available in http://homepage.mac.com/thunderaudio PhotoAlbum11.html.

[3] Kevin Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayana, and D. Tarjan. Temperature-Aware Microarchitecture. In Proceedings of the 30th International Symposium on Computer Architecture (ISCA'03), June 2003.

[4] VAR business, Intel Clears Up Post-Tejas Confusion, Available in http://www.varbusiness.com/sections/news/breakingnews.jhtml?articleId=18842588

[5] David Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In Proceedings of HPCA'01, Jan. 2001.

[6] J. Srinivasan and S. V. Adve. Predictive Dynamic Thermal Management for Multimedia Applications. In Proceedings of International Conference on Supercomputing (ICS'03), June 2003.

[7] Kevin Skadron, M. Stan, K. Sankaranarayana, W. Huang, S. Velusamy, and D. Tarjan. Temperature-Aware Microarchitecture: Modeling and Implementation. ACM Transaction on Architectureand Code Optimization. Vol. 1, No. 1, March 2004, pp. 94-125.

[8] Sung Woo Chung and K. Skadron. Using On-Chip Event Counters for High-Resolution, Real-Time Temperature Measurements. Proceedings of ITHERM'06, June 2006.