韓國 컴퓨터情報學會 論文誌
第14卷 第1號, 2009. 1.

2009-14-1-4-1

실시간 온도 감시를 위한 시뮬레이션 도구의 구현

최 진 항*, 이 종 성*, 공 준 호*, 정 성 우**

Implementation of a Simulation Tool for Monitoring Runtime Thermal Behavior

Jinhang Choi *, Jong Sung Lee *, Joonho Kong *, Sung Woo Chung **

요 약

아키텍처 유닛 단위의 프로세서 온도 시뮬레이션은 신뢰성 있는 프로세서 개발이 중요해진 오늘날에 반드시 필요한 실험이다. 프로세서 공정이 미세화하고 회로 집적이 고밀도화하면서 기존의 냉각 기법으로 효과적인 해결이 어려운 열섬(hotspot) 현상이 발생하고 있기 때문이다. 그러나 지금까지 제안되었거나 개발되어있는 온도 시뮬레이션 도구들은 시뮬레이션 시간이 너무 오래 걸리거나 정밀도가 떨어지는 등의 제약으로 인하여 실제 시스템을 모델링하기에 부족한 점이 있었다. 본 논문에서는 성능계수기를 이용한 실시간 온도 추적 도구의 정밀도를 높이는 방법을 제시하고, 이를 구현하는 것을 목표로 한다. 그 결과, 동적 전압 및 주파수 조절(Dynamic Voltage and Frequency Scaling, DVFS)과 같은 온도 제어 기술을 실제 프로세서에 적용시켰을 때 일어나는 온도 변화를 실시간으로 추적할 수 있는 기반환경이 조성되었다.

Abstract

There are excessively hot units of a microprocessor in today's nano-scale process technology, which are called hotspots. Hotspots' heat dissipation is not perfectly conquered by mechanical cooling techniques such as heatsink, heat spreader, and fans; Hence, an architecture-level temperature simulation of microprocessors is evident experiment so that designers can make reliable chips in high temperature environments. However, conventional thermal simulators cannot be used in temperature evaluation of real machine, since they are too slow, or too coarse-grained to estimate overall system models. This paper proposes methodology of monitoring accurate runtime temperature with Hotspot[4], and introduces its implementation. With this tool, it is available to track runtime thermal behavior of a microprocessor at architecture-level. Therefore, Dynamic Thermal Management such as Dynamic Voltage and Frequency Scaling technique can be verified in the real system.

▸ Keyword : 컴퓨터 구조(Computer Architecture), 온도 시뮬레이션(Temperature Simulation), 핫스팟 모델(Hotspot compact model), 성능 계수기(performance counter)

• 제1저자 : 최진항 교신저자 : 정성우
• 투고일 : 2008. 10. 1, 심사일 : 2008. 11. 4, 게재확정일 : 2008. 12. 20.
* 고려대학교 정보통신대학 석사과정 ** 고려대학교 정보통신대학 컴퓨터·전파통신공학과 조교수
※ 이 연구에 참여한 연구자는 '2단계 BK21사업'과 '2006년 정부 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥 재단(KRF-2006-331-D00452)'의 지원비를 받았음.

Ⅰ. 서 론

오늘날의 상용 마이크로프로세서는 프로세서가 소모하는 에너지를 효율적으로 관리하기 위해 동적 전압/주파수 조절(Dynamic Voltage and Frequency Scaling, DVFS) [1, 2]이나 저 전력 스케줄링 [3] 같은 전력 제어 기술을 지원한다. 프로세서의 성능을 일시적으로 낮추며 전력 소모를 줄이는 것으로 대표되는 위 방식은, 고성능 파이프라인 구조에서부터 저 전력기반 임베디드 시스템까지 마이크로프로세서 설계 전반에 걸쳐 하나의 중요한 요소로 자리 잡았다. 그런데, 최근 위 전력 제어 기술이 프로세서의 발열 문제를 해결하기 위한 대안으로 연구되기 시작하였다 [4, 5]. 프로세서의 공정 미세화와 고밀도 집적화가 야기한 높은 전력 밀도를 전력 제어 기술로 완화하여, 프로세서의 온도를 낮추는 방식으로 접근한 것이다.

전력 제어 기술을 온도 제어 기법으로 사용하게 되면서, 프로세서의 온도가 실제 어떤 양상으로 나타나는지, 그리고 온도에 따라 전력 소모가 어떻게 변하는지 알아야 할 필요성이 대두되었다. 하나의 방안으로, K.—J. Lee와 K. Skadron [6]은 성능 계수기(performance counter)를 이용한 Hotspot 모델을 제안하였다. 이 모델은 실제 시스템 환경에서 기능 유닛(Functional Unit) 단위의 온도를 실시간으로 계산하는 인터페이스를 제공한다. 그러나 온도를 시뮬레이션 하는데 필요한 시스템 부담이 온도 값에 포함되기 때문에, 위 모델을 이용하여 온도 상승에 종속적인 온도 제어 기술에 대한 검증을 수행하기에는 무리가 있다. 다시 말해, 온도 시뮬레이션이 더 정확해야 한다.

본 논문에서는 Hotspot 모델에 의한 시스템 부담을 감추며 프로세서 온도 시뮬레이션을 수행하기 위한 방법론을 제시하고, 방법론의 유효성을 실제 소프트웨어 도구의 구현으로 확인한다. 그래서 작업 부하(workload)가 프로세서의 온도에 끼치는 영향을 실제 시스템 환경에 가깝게 측정하고, 더불어 온도 제어 기술이 프로세서의 온도 및 전력 소모, 그리고 성능에 끼친 결과를 신뢰도가 높은 수준에서 정량적으로 얻을 수 있는 기반을 만들고자 한다.

본 논문의 구성은 다음과 같다. 2 장에서는 프로세서의 온도를 측정하기 위해 사용된 기존 연구의 특징을 알아보고, 제안하는 시뮬레이션 도구의 구성에 사용된 연구 내용을 소개한다. 이어서, 3 장에서는 본 논문에서 구현한 온도 시뮬레이션 도구의 구조 및 인터페이스를 소개하고, 이 도구의 유효성을 검증하기 위한 실험 내용을 4, 5 장에 걸쳐 기술하였다. 4 장은 실험 환경, 5 장은 실험 결과의 평가에 대한 내용이다. 마지막으로, 6 장에서는 본 연구의 의의와 향후 과제에 대하여 서술하며 본 논문을 마무리한다.

Ⅱ. 관련 연구

K. Skadron et al.이 제안한 Hotspot 모델 [5]은 반도체 칩의 온도를 프로세서의 아키텍처 수준에서 시뮬레이션 하는 소프트웨어 인터페이스이다. Hotspot은 프로세서가 소모하는 전력 값을 입력 값으로 하여, 플로어플랜(floorplan)에 따른 기능 유닛들의 배치와 특성(저항, 도체 성질)에 따라 각 유닛의 온도를 출력한다. 이 방식은 온도 계산을 위해 따로 특별한 하드웨어를 필요로 하지 않는다는 장점이 있으나, 온도 센서를 이용하여 프로세서의 온도를 측정하는 것과 비교했을 때 상대적으로 느리다는 단점을 가진다. 이런 문제를 극복하고자 K.—J. Lee와 K. Skadron이 제안한 모델이 성능계수기를 이용한 Hotspot [6]이다. 이 모델은 성능계수기를 이용하여 얻은 사건 정보(performance counter event)로 각 기능 유닛의 전력 소모량을 계산한 뒤, 전력 값을 Hotspot 에 입력하여 프로세서의 온도로 변환한다. 이런 구조는 Hotspot 모델이 시스템에서 프로세서의 상세 온도 정보를 실시간으로 습득하는 것을 가능하게 한다. 다시 말하면, 기존 Hotspot 모델은 프로세서의 온도 정보를 얻기 위해 고정된 시스템 환경과 주어진 전력 데이터 집합을 입력으로 하는 오프라인(offline) 시뮬레이션만 가능하지만, K.—J. Lee 와 K. Skadron의 Hotspot모델은 시스템 환경과 작업 부하에 동적으로 변화를 주는 것이 가능하며, 이런 변화가 일어났을 때 프로세서의 온도에 나타나는 영향을 즉시 확인할 수 있는 실시간 시뮬레이션이 가능한 것이다. 이러한 실시간성의 확보는 동적으로 일어나는 프로세서의 변화를 추적할 수 있다는 점에서 아주 중요하다. 하지만 위 Hotspot 모델은 큰 문제점을 안고 있다. Hotspot의 온도 계산이 프로세서에서 수행되는 작업 부하에 포함되는 것이 바로 그것이다. K.—J. Lee 와 K. Skadron은 Hotspot의 온도 계산을 포함하는 시스템 부담(system overhead)이 사건 정보로 누적되면서, 온도 시뮬레이션 결과가 4~8 ℃이상 상승하게 된다고 보고하였다 [6]. 만약 이 차이가 프로세서에서 가장 뜨겁다고 알려진 정수 레지스터 파일(Integer Register File)과 같은 유닛에서 발생한다면, 해당 시뮬레이션 결과를 기준으로 설계된 온도 제어 기술은 상대적으로 많은 온도 제어를 수행하게 되며 결과적으로 프로세서의 성능을 저하시키게 된다. 이 오차를 줄

이기 위해서는 온도 계산이 이루어지는 동안 프로세서에서 사건 정보를 수집하는 것을 중단하여, 온도 계산에 의한 시스템 부담을 시뮬레이션에서 감출 필요가 있다.

한편, 프로세서의 성능계수기를 이용하려면 해당 프로세서에 존재하는 특정 명령어(x86 아키텍처의 rdmsr과 같은 명령어)를 사용해야만 한다. 그런데, 이 명령어는 각각의 명령어 집합 구조(Instruction Set Architecture)에 종속적이기 때문에 프로세서 종류마다 성능계수기를 사용하기 위한 명령어를 다시 조사하는 작업이 필요하다. 게다가, 하나의 프로세서 내에서 얻어낼 수 있는 사건 정보는 100여 가지 이상이다. 다행히, 프로세서의 종류에 따라 사건 정보를 선택하고 성능계수기의 값을 얻어내는 perfmon2 [7]라는 인터페이스가 리눅스 기반 공개 소스 프로젝트로 진행되고 있다. 그래서 본 논문에서는 perfmon2가 지원하는 프로세서에 대해 범용 적으로 적용 가능한 Hotspot 시뮬레이션 도구를 작성하여 사용하는 것을 목표로 한다.

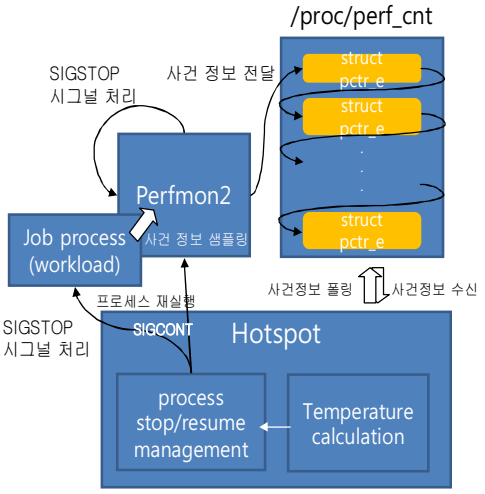

그림 1. 시뮬레이터의 기본 구조 및 인터페이스
Fig 1. Simulator Base Architecture/Interface

III. 구현

온도 시뮬레이션에서 실시간 Hotspot모델의 오차를 줄이는 가장 간단한 방법은 Hotspot모델의 온도 계산이 수행되는 동안 프로세서의 사건 정보 감시(monitoring performance-counter event)를 중단하여, 프로세서에서 온도 계산을 하는 동안 누적되는 사건 정보를 다음 샘플에서 제외하는 것이다. 이 때 중요한 것은 프로세서의 감시가 중단되면 프로세서 위에서 수행되고 있던 작업도 정지하는 것이다. 그렇지 않으면, Hotspot 모델의 온도 계산 시간만큼 해당 작업이 진행되면서 프로세스의 사건 정보가 분실되는 문제가 발생한다. 우리는 위 정책을 기반으로 시뮬레이션 도구를 구성하였다.

1. 시뮬레이션 도구의 기본 구조

시뮬레이터 도구는 크게 네 부분으로 구성된다.

(1) 성능계수기로부터 읽어낸 사건 정보를 임시 저장 공간으로 전달하는 perfmon2
(2) 읽어 들인 사건 정보를 Hotspot 모델에 전달하기 위한 임시 저장 공간 (proc file)
(3) 프로세스의 작업을 중지 및 계속하는 작업관리자
(4) 저장 공간으로부터 읽어 들인 사건 정보로 프로세서의 온도를 계산하는 Hotspot 모델

Perfmon2는 프로세서의 성능계수기로부터 사건 정보를 수집하는 역할을 한다. 사용자에 의하여 정의된 간격마다 누적된 사건 정보를 하나의 샘플로 구성한 다음, proc에 존재하는 임시 저장 공간에 기록한다. 한 번의 샘플 기록이 끝난 perfmon2는 일시 중지 상태에 들어가며, 작업 중지/실행 관리자로부터 재실행 신호가 오기까지 프로세서 감시를 중단하고 대기한다.

임시 저장 공간은 perfmon2로부터 전달 받은 사건 정보 샘플을 Hotspot모델에 전달하는 역할을 담당한다. 이 공간은 proc 파일 으로 구현되어 있으며, 이 파일을 관리하는 시스템은 커널에 연결하는 모듈로 구현되어 쉽게 생성 및 수정이 가능하다.

작업 중지/실행 관리자는 perfmon2의 프로세서 사건 정보 감시 실행, 프로세서 위에서 수행중인 작업에 대한 중지 및 지속, 마지막으로 Hotspot 모델의 온도 계산 수행을 관리한다. 수행 단계는 다음과 같다. 먼저 임시 저장 공간에 해당하는 proc파일로부터 사건 정보 샘플을 얻을 수 있을 때까지 폴링(polling)방식으로 읽기를 요청한다. 이후 사건 정보를 얻게 되면 프로세서 위에서 진행 중인 프로세스에 작업 중지 신호를 준다. 그리고 Hotspot 모델이 온도 계산을 수행한 뒤에는 차례대로 perfmon2와 작업 프로세스에 실행 신호를 준 뒤 다시 폴링 모드로 돌아간다.

Hotspot 모델은 프로세서의 플로어플랜에 따른 기능 유닛

당 전력 소모를 기준으로 온도를 계산한다. 계산된 온도는 메모리에 임시 저장되었다가 시뮬레이션 종료 후 시간 순서대로 파일에 기록된다. Hotspot 모델은 작업 중지/실행 관리자와 연결되어 하나의 프로세스로 작업을 수행하게 되고, perfmon2와는 서로 다른 프로세스로서 작업을 분화하게 된다.

한편, 이 시뮬레이션 도구에서 중요하게 고려되는 것 중 하나는 시뮬레이션을 수행하기 위해 필요한 커널과 관련된 수정을 최소화하여 타 시스템에 대한 이식성을 높이는 것이다. 그래서 임시 저장 공간을 제외한 각 기능은 모두 유저영역에서 수행되도록 구현하였다.

2. 인터페이스

본 시뮬레이션 도구의 인터페이스는 Hotspot-sched [8]의 골격을 차용한다. Hotspot-sched에서는 커널로부터 전달되는 성능계수기의 데이터 샘플을 proc파일의 임시 저장 공간에 리스트형식으로 쌓은 다음 Hotspot 모델이 가져가는 형식을 취하였다. 이를 응용하여, 본 시뮬레이션 도구의 경우 Hotspot-sched와 같은 형식의 proc파일에 읽기와 쓰기를 비동기적으로 처리할 수 있도록 모듈에서 함수를 제공한다. 한편, 작업 관리자로부터 각 프로세스에 전달되는 중지 및 실행 신호는 SIGSTOP과 SIGCONT시그널을 이용한다. 운영체제 위에서 해당 시그널이 전달되어 처리되는 시간은 1 밀리초(ms)보다 작다. 그런데, 이 때 운영체제의 작업 전환(context switching)이 샘플링 시간에 포함되면서 사용자가 원하는 시간보다 작은 기간 동안 수행된 작업 프로세스의 사건 정보가 Hotspot 모델에 전달될 수 있다. 최악의 경우 시뮬레이션을 원하는 프로세스가 아닌 다른 프로세스의 사건 정보가 Hotspot 모델에 전달되는 결과로 이어질 수 있으므로, 근사치 보정을 위해 정의된 샘플링 시간미만으로 누적되는 사건 정보는 버린다. 이 정책이 시뮬레이션에 주는 영향은 유효성 검증 부분에서 다시 논의한다.

Ⅳ. 실험 환경

구현된 온도 시뮬레이션 도구의 성능 및 유효성 평가를 위하여, 주파수가 2.1GHz인 인텔 Core2Duo 프로세서 [9]를 기반으로 리눅스 커널 2.6.25를 사용하는 페도라 코어(Fedora Core) 8을 시스템 환경으로 채택하였다. 실험에는 SPEC CPU2000 벤치마크[10] 중 하나인 gcc를 사용하였다. Hotspot에 적용해야 하는 프로세서의 플로어플랜은 Gabriel H. Loh [11]의 모델을 참조하여 작성하였다.

표 1. SPEC 벤치마크 gcc의 명세
Table 1. Description of SPEC gcc

구분	설명
GCC	C언어 최적화 컴파일러
바이너리 형식	ELF 32-bit, x86
메모리 점유 공간	13.95MB
입력 데이터	REF 데이터 200.i
출력 데이터	어셈블리 코드 파일
실행시간	26.4s
시스템 부하	300ms

Ⅴ. 성능 평가 및 유효성 검증

〈그림 2〉와 〈그림 3〉은 벤치마크 gcc에 대한 온도 시뮬레이션을 각각 오프라인과 실시간으로 추적한 결과를 시간의 흐름에 따라 기록한 것이다. 오프라인으로 수행한 온도 시뮬레이션은 Core2Duo 프로세서의 사건 정보를 perfmon2를 이용하여 10 밀리초 단위로 샘플링 하여 파일로 기록한 다음, 해당 사건 정보 파일을 한 번에 기존 Hotspot 모델에 입력하여 프로세서의 온도를 출력한 결과이고, 실시간 시뮬레이션은 3장에서 구현한 시뮬레이션 도구를 이용하여 10 밀리초 단위로 만들어낸 샘플마다 온도 변화를 계산한 뒤 기록한 결과이다.

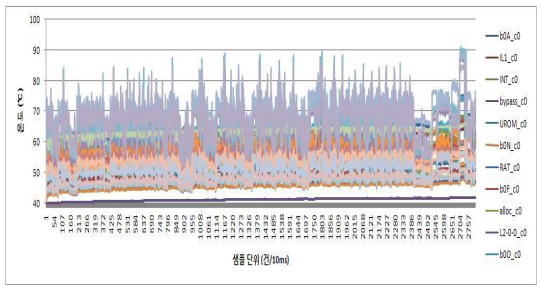

그림 2. gcc의 오프라인 온도 시뮬레이션 결과
Fig 2. Offline Temperature Simulation Result of gcc

그림 3. gcc의 실시간 온도 시뮬레이션 결과
Fig 3. Run-time Temperature Simulation Result of gcc

위 실험에서 가장 먼저 비교 가능한 수치는 샘플의 개수에 따른 작업 프로세스의 실행시간이다. 오프라인으로 진행된 온도 시뮬레이션에서 프로세스의 실행시간에 영향을 미치는 것은 perfmon2를 이용하여 프로세스의 사건 정보를 수집하는 부분이 유일하다. 프로세스가 실행되는 10 밀리 초 마다 하나의 샘플이 발생하는 것을 고려했을 때, 2805 개의 사건 정보 샘플은 약 28.1 초 동안 프로세스가 실행되었음을 의미한다. 온도 시뮬레이션을 수행하지 않는 gcc 벤치마크의 순수 실행시간은 평균 26.4 초, 운영체제를 포함한 시스템의 부하가 끼친 실행시간 지연은 약 300 밀리 초이다. 정리하면, perfmon2를 이용하여 해당 프로세스의 작업 정지 및 재실행을 수행하며 10 밀리 초당 사건 샘플을 모으는 기능이 6.4 %의 추가 사건 정보를 일으켰음을 의미한다. 한편, 실시간 온도 시뮬레이션의 사건 정보 샘플은 2722 개로, 오프라인 실험 대비 3 %의 보정이 이루어졌다. 3 장에서도 언급했듯이, 실시간 온도 시뮬레이션은 10 밀리 초 이하의 샘플링 결과가 포함된 사건 정보가 Hotspot으로 전달되면 해당 정보를 버리게 된다. 정의된 샘플링 간격 이하의 샘플 데이터가 들어오면 해당 프로세스에서 발생한 사건정보가 아니라고 간주하는 이 방법은 실제 작업 프로세스 대비 약 97 %의 정밀도로 사건 정보를 샘플링 하여 데이터를 정리하는 결과를 보여준다. 이에 따른 시뮬레이션 결과는 오프라인과 실시간 온도 시뮬레이션의 패턴이 일치하는 것으로 이어진다.

표 2는 온도 시뮬레이션이 끝난 뒤의 데이터를 대상으로 각 유닛의 최고 온도 값과 최저 온도 값을 검색한 뒤, 온도 패턴이 가지는 분산이 같은지를 F-검증 통계 분석 모델(F-test statistic model) [12]을 이용하여 비교한 결과이다. F-검증 모델은 주어진 표본 집단에 대한 검정의 유의 확률(significance probability) 결과가 0.05 이하일 경우 두 표본 집단이 통계적으로 같은 분산(deviation)을 가진다고 판단한다. 표 2 의 분산 비교 결과에서 모든 유닛의 온도 패턴은 0.05 이하의 유의 확률 값을 보이며, 이는 곧 오프라인과 실시간 온도 시뮬레이션

표 2. Core2 Duo 프로세서의 각 유닛에 대한 온도 시뮬레이션 결과
Table 2. Temperature Simulation Result for each functional unit in Core2Duo processor

기능 유닛	오프라인 온도 시뮬레이션(℃)		실시간 온도 시뮬레이션(℃)		분산 비교 결과
	최고 온도	최저 온도	최고 온도	최저 온도	
B0A_c0	50.09	41.5	78.24	46.09	0
IL1_c0	64.23	50.98	70.93	5.68	1E-86
INT_c0	75.85	44.91	81.84	45.29	0.0368
Bypass_c0	57.85	44.2	63.48	44.47	3.363E-39
UROM_c0	48.08	40.81	55.66	43.72	7.89E-180
b0N_c0	68.37	44.64	72.77	45.7	4.3E-07
RAT_c0	75.19	59.58	78.9	60.98	5E-14
b0F_c0	52.95	42.65	57.4	42.55	4E-44
Alloc_c0	68.58	53.71	72.64	55.24	2.4E-19
L2-0-0_c0	42	40.01	42.84	40	1.3E-81
b0O_c0	59.44	44.44	65.62	46.05	2.7E-34
AGU_c0	65	44.01	67.16	45.06	0.00012
b0I_c0	48.65	41.23	72.02	45.05	0
b0E_c0	51.88	42.69	56.67	43.13	2E-55
LSQ_c0	60.55	43.15	63.85	44.54	7E-09
b0H_c0	49.93	41.43	79.63	46.49	0
b0S_c0	60.11	43.51	69.92	45.88	8E-71
Cmplx_c0	47.88	40.62	53.78	42.48	4E-132
b0D_c0	49.91	41.73	68.7	46.92	0
b0L_c0	67.12	46.31	70.73	46.22	5E-07
BP_c0	57.31	43.26	62.18	43.79	2E-16
b0G_c0	49.25	41.49	56.38	44.48	4E-163
B0M_c0	74.67	49.12	78.55	49.74	9.9E-05
DL1_c0	65.35	44.67	67.88	44.42	2E-06
SIMD_c0	51.5	42.14	54.18	42.64	1.18E-37
b0J_c0	62.52	44.39	66.63	43.37	5E-12
B0R_c0	51.53	42.39	54.34	42.64	9E-38
ROB_c0	89.14	45.63	94.15	47.19	0.00379
Commit_c0	90.96	46.23	95.09	46.93	0.0355951
B0C_c0	55.58	45.26	67.31	52.26	5E-275
RS_c0	90.38	50.97	95.08	48.39	0.02
B0Q_c0	50.51	42.17	57.32	43.14	1E-118
IFQ_c0	65.82	51.33	69.71	55.68	1E-45
Decode_c0	84.95	45.45	88.89	44.29	0.0171781
b0P_c0	51.17	41.59	65.01	44.72	0
B0K_c0	60.93	44.09	64.97	44.92	7E-14
BTB_c0	56.83	43.13	65.82	44.24	5E-44
B0B_c0	51.31	41.79	56.73	43.52	3E-56

결과가 같은 패턴을 가진다고 판정할 수 있음을 의미한다. 한편, 오프라인 시뮬레이션과 비교했을 때 실시간 온도 시뮬레이션의 최고/최저 온도 값이 최저 0.5 ℃에서 최대 28 ℃까지 전체적으로 상승했음을 알 수 있는데, 이것이 바로 프로세서에서 발생한 동적인 시스템 환경 변화가 실시간 온도 시뮬레이션에 적용되어 나타난 사례라고 할 수 있다. 본 논문에서는 공간의 부족으로 〈그림 2〉와 〈그림 3〉에 모든 유닛의 온도 추세를 같이 표현하였으나, 각 기능 유닛 별로 온도 그래프를 살펴보면 유닛의 최고 온도 혹은 최저 온도가 짧은 구간에서 급격하게 상승 또는 하강하는 변화 속에서 발생하는 것이 확인된다. (이 상황에서도 여전히 각 기능 유닛의 온도 패턴은 오프라인 실험과 같은 경향을 보인다.)

한편, 실시간 온도 시뮬레이션으로 gcc 벤치마크를 수행했을 때 gcc가 프로세서 위에서 작업을 수행한 시간은 28.34 초이며, 온도 시뮬레이션을 포함한 총 실험 시간은 10 분 3 초가 소모되었다. 펜티엄 4 프로세서 위에서 사이클 정확(cycle-accurate) 시뮬레이터인 심플스칼라로 gcc벤치마크를 1초 수행하는 것에 대한 온도 및 전력 소모 시뮬레이션을 수행하려면 약 3시간이 걸린다는 것을 상기했을 때 (사이클 단위로 시뮬레이션을 수행하는 심플스칼라는 초당 약 150,000개의 명령어를 처리한다. [13]), 정밀도를 유지하면서 벤치마크의 온도 및 전력 소모 시뮬레이션을 빠르게 수행할 수 있는 실시간 시뮬레이션의 장점은 확연하다.

VI. 결론 및 향후 연구

본 논문은 Hotspot 모델과 성능계수기에서 프로세서의 사건 정보를 읽어오는 공개 소스 소프트웨어인 perfmon2를 연결하여 실시간으로 프로세서의 온도 정보를 얻어내는 방법론을 설명하고, 이에 대한 실제 구현을 제시하였다. 본 논문의 온도 시뮬레이션 도구는 Hotspot이 수행하는 온도 계산이 시스템에 끼치는 영향을 시뮬레이션으로부터 제거하여, 주어진 작업 부하가 가지는 순수한 온도 패턴을 정량적으로 파악한다. 따라서 우리는 온도에 민감한 고성능 프로세서 및 저 전력 임베디드 프로세서가 실제 환경에서 어떻게 변화하는지를 효과적으로 상세하게 확인할 수 있다.

하지만, 상기 온도 시뮬레이션 도구는 실제 시스템의 제약에 따라 샘플링 간격을 아주 세밀하게 할 수 없음에 유의해야 한다. 현재 시스템인 Core2Duo 프로세서의 경우 자유롭게 사용 가능한 두 개의 성능계수기에서 11개의 사건 정보를 10 밀리 초마다 읽어오도록 구성되었는데, 이보다 짧은 간격으로 사건정보를 읽어오게 될 경우 작업 프로세스보다 시뮬레이터 및 운영체제가 프로세서에 위치하는 비율이 높아지게 되어 정확한 사건 정보를 얻어올 수 없게 된다. 그렇기 때문에, 온도 시뮬레이션을 위한 사건 정보 샘플링 간격을 최소 10 밀리 초로 유지하는 것이 좋다.

본 논문에서 제안하는 온도 시뮬레이션 도구를 이용하면 컴퓨터 시스템의 공랭식 냉각 정책 수립이나 새로운 온도 제어 기술 개발 및 전력 제어 기술 검증에 큰 도움이 될 것이다. 향후, 공정 미세화에 따라 전체 전력 소모에서 높은 비율을 차지하게 된 누설 전력(leakage power)이 칩 온도로부터 받는 영향을 분석하여, 온도 제어를 통해 에너지를 최적화하는 기법을 연구하는데 본 논문의 시뮬레이션 도구가 사용될 계획이다. 이 밖에, 프로세서의 주변 온도(ambient temperature)와 프로세서의 온도간의 상관관계를 분석하여 온도 제어 정책을 개선하는 등의 추가 연구도 예상하고 있다.

참고문헌

[1] B. Lin, A. Mallik, P. A. Dinda, G. Memik, and R. P. Dick, *"Power reduction through measurement and modeling of users and CPUs: summary,"* In Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 363-364, 2007.

[2] 이관형, 강진구, 김재진, *"이동형 통신 시스템에서 프로세서에 대한 최소 전력 소모를 위한 주파수 선택 알고리즘 연구,"* 한국컴퓨터정보학회 제 38차 하계학술발표논문집, 제 16권, 제 1호, 25-31쪽, 2008년 6월.

[3] 최지영, 박남서, 안도희, *"다중 공급 전압을 이용한 저전력 스케쥴링 및 할당 알고리듬,"* 한국컴퓨터정보학회 논문지, 제 7권, 제 2호, 79-86쪽, 2002년 5월

[4] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas, *"A framework for dynamic energy efficiency and temperature management,"* In Proceedings of the 33rd annual ACM/IEEE internationnal symposium on Microarchitecture, pp. 202-213, 2000.

[5] K. Skadron, K. Sankaranarayanan, S. Velusamy, D. Tarjan, M. R. Stan, and W. Huang, *"Temperature-Aware Microarchitecture: Modeling and Implementation,"* ACM Transactions on Architecture and Code Optimization, Vol. 1, No. 1, pp.94-125, 2004.

[6] K.-J. Lee and K. Skadron, *"Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors,"* In Proceedings of the 19th IEEE international Parallel and Distributed Processing Symposium -Workshop 1-, pp. 232.1, 2005.

[7] S. Eranian, the Perfmon2 project, *http://perfmon2.sourceforge.net*.

[8] S.-W. Kim, H.-C. Park, S. W. Chung, and C. Yoo, *"HotSpot-Sched: HotSpot Extension for Easy Evaluation of Various Scheduling Policies"* In Technical Report: KU-DCCE-SMRL-2007-001, 2007.

[9] Intel(R) CoreTM2 Duo Processor E8000 and E7000 Series Datasheet, *http://download.intel.com/design/processor/datashts/318732.pdf*

[10] SPEC2000 benchmark suite, *http://www.spec.org*

[11] G. H. Loh, *"A modular 3d processor for flexible product design and technology migration,"* In Proceedings of the 2008 conference on Computing frontiers, pp. 159-170, 2008.

[12] George W. Snedecor and William G. Cochran, *"Statistical Methods,"* The Iowa State University Press, Ames, Iowa, pp. 223~224, 1989.

[13] D. Burger and T. M. Austin, *"The SimpleScalar ToolSet, Version 2.0,"* in University of Wisconsin-Madison. *http://www.simplescalar.com*, 1997

저 자 소 개

최 진 항

2008년 2월: 고려대학교 정보통신대학 컴퓨터과학 학사

2008년 3월~현재: 고려대학교 컴퓨터 통신공학부 석사과정

관심 분야: 컴퓨터 구조, 고성능 컴퓨터, 실시간 데이터 처리

이 종 성

2007년 2월: 고려대학교 정보통신대학 컴퓨터과학 학사

2007년 3월~현재: 고려대학교 컴퓨터 통신공학부 석사과정

관심 분야: 컴퓨터 구조, 시스템 소프트웨어, Solid-State Disk

공 준 호

2007년 8월: 고려대학교 정보통신대학 컴퓨터과학 학사

2007년 9월~현재: 고려대학교 컴퓨터 통신공학부 석사과정

관심 분야: 컴퓨터 구조, 고성능 컴퓨터, 임베디드 프로세서

정 성 우

2003년 2월: 서울대학교 전기 컴퓨터공학 박사

2003년 1월~2005년 2월.: 삼성전자 반도체총괄, Senior engineer

2005년 3월~2006년 1월: 방문 연구원, Department of Computer Science, University of Virginia

2006년 3월~현재: 고려대학교 컴퓨터 통신공학부 조교수

관심 분야: 컴퓨터 구조, 프로세서 온도 관리, 플래쉬 메모리, System on Chip