하드웨어 성능 카운터에 기반한 커널 수준의 동적 태스크 프로파일링 모듈의 설계 및 구현

송욱○ 송지석 김지홍
서울대학교 컴퓨터공학부
{answer03, danielsong, jihong}@davinci.snu.ac.kr

Design and Implementation of a Kernel-Level Run-Time Task Profiling Module Using Hardware Performance Counters

Wook Song○ Jiseok Song Jihong Kim
School of Computer Science and Engineering, Seoul National University

요 약

하드웨어 성능 카운터를 기반으로 한 에너지 예측 모델은 많은 수의 성능 정보 이벤트들의 조합을 요구한다. 그러나 그 수에 비해 대부분의 임베디드 프로세서에서 지원하는 하드웨어 성능 카운터는 상당히 적은 수로 제한되어 있으므로, 태스크 전력 프로파일에 이러한 모델을 직접 활용하기 어려운 실정이다. 따라서 적은 수의 하드웨어 카운터만으로도 다 수의 성능 정보를 유추할 수 있는 하드웨어 성능 카운터 멀티플렉싱 기법의 적용이 필요하다. 본 논문에서는 기존의 리눅스 커널에 샘플링 엔진과 문맥 교환 모듈을 추가하여 하드웨어 성능 카운터 멀티플렉싱과 커널 수준에서의 태스크 별 전력 프로파일링을 구현하였다. 또한 각 샘플링 주기의 변화가 수행 시간의 증가에 끼치는 영향 정도와 그에 따른 프로파일의 정확도를 평가하였다. 특히 SPLASH-2 4개의 벤치마크에 실험을 통해 성능 오버헤드와 오차율 모두 1% 이내의 샘플링 주기를 확인하였다.

1. 서 론

멀티프로세서 시스템은 물리적, 기술적 제약으로 인해 성능 향상에 한계에 부딪힌 단일프로세서에 대한 대안으로 제안된 것으로 복수 개의 프로세서를 하나의 칩에 집약한 시스템이다. 이미 인텔과 AMD와 같은 주요 칩 벤더들이 출시한 여러 제품군이 개인용 컴퓨터와 서버에 광범위하게 채택되어 활용되는 실정이며, 또한 최근에는 멀티미디어 처리, 게임 등과 같이 높은 성능 요구 사항을 갖는 모바일 응용의 증가와 여러 가지 기능들이 단일 기기에 통합되는 추세로 인하여 임베디드 시스템에서도 멀티프로세서를 채택하는 비율이 높아지고 있다.

임베디드 시스템은 대부분 베터리를 전원으로 사용하고 있기 때문에, 한정된 자원인 베터리의 효율적인 사용 역시 성능의 향상 못지않게 매우 중요하다. 따라서 임베디드 시스템의 에너지를 효율적으로 사용하는 것에 대한 관심이 매우 크며, 계속해서 증가하고 있다.

에너지의 최적화를 통한 효율적인 사용을 위해서는 에너지를 정확하게 측정하는 것이 선결되어야 한다. 하지만 시스템에서 소모된 에너지를 효과적으로 측정하기 위해서는 많은 비용이 소모되는 여러 측정 기기가 필요하고 [1], 때로는 시스템의 하드웨어 제약으로 인하여 측정 자체가 불가한 경우도 있다. 이러한 측정에 있어서 존재하는 문제 때문에 측정하고자 하는 응용의 성능 정보를 활용하여 그 응용이 소모한 에너지 정보를 유추하는 방법 및 모델들이 제안되었다. 유추에 사용되는 성능 정보들은 프로세서에서 제공하는 하드웨어 성능 카운터(Hardware Performance Counter, 이하 HPC)를 통해 얻을 수 있는 것들로 프로세서에 따라 측정할 수 있는 이벤트의 종류와 개수가 다르다. 프로세서에서 여러 개의 성능 이벤트를 제공하더라도 한 개의 HPC가 한 번에 측정할 수 있는 이벤트는 하나뿐이므로, 프로세서에서 제공하는 HPC 레지스터의 개수만큼만 동시에 측정할 수 있다. 이러한 점은 한 번에 많은 수의 HPC 레지스터를 제공하는 개인용 컴퓨터나 서버용 프로세서보다 적은 수의 레지스터만을 제공하는 대부분의 임베디드 프로세서에서 더 큰 제약으로 작용한다. 예를 들어 응용의 캐시 미스 비율이나 앞에서 언급한 소모한 에너지 정보 등을 얻기 위해서는 여러 개의 HPC 이벤트의 조합이 필요한데 프로세서에서 제공하는 HPC 레지스터 수보다 많은 수의 이벤트를 측정하기 위해서는 같은 응용을 여러 번 수행하여 측정해야 하는 문제가 발생할 수 있다. 가령 단순히 한 응용의 시작부터 끝까지의 성능 정보를 얻기 위해서라면 제한된 HPC 레지스터의 수를 극복하기 위해 여러 번 측정하는 수고를 감수할 수 있다. 그러나 런타임에 각 응용의 시시각각 변하는 성능 정보를 프로파일

링 하기 위해서는 이러한 방법을 사용할 수 없다. 따라서 적은 수의 HPC를 활용하여 더 많은 수의 이벤트 정보를 얻을 수 있는 멀티플렉싱 기법이 필요하다.

근래의 주요 운영체제들은 멀티프로세서를 지원하는 커널을 채택하고 있다. 이러한 커널들은 각각의 코어를 완전히 독립된 캐시와 메모리를 공유하는 하나의 프로세서의 관점에서 태스크를 할당하고 자원을 관리한다. 단일프로세서와 달리 멀티프로세서에서는 각 프로세서의 효율적인 사용을 위해서 각 태스크를 어떤 프로세서에 할당할 것인지를 결정하는 스케쥴러에 대한 새로운 고찰이 필요하다. 따라서 스케쥴러에 의해 할당될 각 태스크의 특성을 동적으로 프로파일링하는 것이 필요하고 이렇게 프로파일링 된 정보를 스케쥴러에서 적극 활용하는 것이 성능 및 에너지 최적화에 중요하다.

본 논문에서는 멀티프로세서에서 이미 제안된 전력 모델을 활용하여 커널 수준의 태스크 전력프로파일링을 구현하였다. 기존의 전력 모델을 적용하기 위해 필요한 각 이벤트들을 동적으로 수집하는 과정에서 커널 수준의 멀티 플렉싱을 제안하였고, 멀티플랙싱의 각 샘플링 주기에 따른 오버헤드와 정확도를 SPLASH-2 프로그램 4개에 대하여 평가하였다. 향후 이러한 커널 수준의 태스크 프로파일링을 활용하는 스케쥴러를 연구할 계획이다.

이후의 논문 구성은 다음과 같다. 2장에서는 관련 연구를 소개하며, 3장에서는 커널 수준의 동적 태스크 프로파일링 기법의 구현을 설명한다. 4장에서는 구현된 기법을 평가하여 본 후, 5장에서 결론과 향후 연구에 관하여 정리할 것이다.

2. 관련 연구

현재까지 제안된 성능 및 에너지 정보 프로파일링 기법에는 크게 정적인 분석과 동적인 분석 두 가지로 나눌 수 있다. 정적인 분석은 주로 프로그램의 성능이나 에너지 병목 지점에 관한 분석의 여지, 즉 최적화 힌트를 사용자에게 제공하는 일종의 도구의 형태로 사용되었고, 동적인 분석은 사용자를 대상으로 한 것이 아닌 커널 내부에서 태스크나 특정 코어의 성능 또는 에너지 정보를 프로파일링하여 효율적인 스케쥴링이나 코어간의 쓰레드 이동을 지원하기 위해 사용되었다.

PAPI [2,3]는 다양한 프로세서의 HPC를 제어할 수 있는 간단한 인터페이스를 제공한다. 라운드-로빙으로 100밀리 초의 주기마다 이벤트를 교체하는 방법으로 멀티플렉싱을 구현하였으며, 멀티쓰레디드 프로그램의 측정을 지원한다. 많은 정적인 분석 기법이 PAPI와 동일한 형태의 멀티플렉싱을 활용하며 멀티쓰레디드 프로그램의 지원 여부의 차이 정도만 있다. 커널 수준에서 직접 HPC를 제어하는 것이 아닌 사용자 응용에서 사용 가능한 인터페이스를 제공하는 형태이므로 멀티플렉싱을 사용할 경우 정확성과 오버헤드 측면에서 문제의 여지가 있다.

Azimi 등 [4]은 이러한 오버헤드를 감소시키기 위해 커널 레벨에서만 데이터를 수집하는 기법을 제안하였다. 멀티플렉싱을 지원하며 라운드-로빙 방식으로 이벤트를 선택하지 않고 랜덤 방식으로 이벤트를 선택하여 100마이크로 초마다 한 번씩 교체하는 방식을 선택하였다. 그러나 여전히 서버급의 프로세서를 대상으로 한 것이기 때문에 임베디드 시스템에 적용하기에는 너무 복잡하여 무리가 있다.

Merkel 등 [5]은 멀티프로세서 시스템에서 특정 프로세서의 과열로 발생하는 성능 저하, 프로그램 실행 멈춤 등의 현상을 크게 줄일 수 있는 에너지 고려 스케쥴러를 제안하였다. HPC 성능 정보를 이용한 에너지 모델과 열 모델을 이용하여 각 태스크 별 프로파일을 구성하고 이를 스케쥴러에서 활용하는 방식은 본 논문의 확장 연구로 생각하고 있는 방향이다. 그러나 에너지와 열 모델을 이루는 각각의 HPC 이벤트들을 모두 동시에 측정 가능한 프로세서만을 생각하고 있으므로 상대적으로 적은 수의 HPC 레지스터만을 제공하는 임베디드 시스템에는 적합하지 않다.

본 논문에서는 [6]에서 제안한 멀티프로세서용 에너지 예측 모델을 이용하여 각 태스크의 전력 프로파일을 구성하였다. 위의 에너지 예측 모델은 ARM11 MPCore [7] 프로세서에서 제공하는 다양한 HPC 이벤트 중 소모 에너지에 영향을 주는 것을 5가지(Instructions, L1 Data Cache Accesses, L2 Cache Accesses, Data Dependency Stalls, Coherence Transactions)로 추렴한 후, 선형 회기 분석을 통해 모델을 구성하였다.

3. 커널 수준의 동적 태스크 프로파일링 구현

3.1 시스템 설계 개관

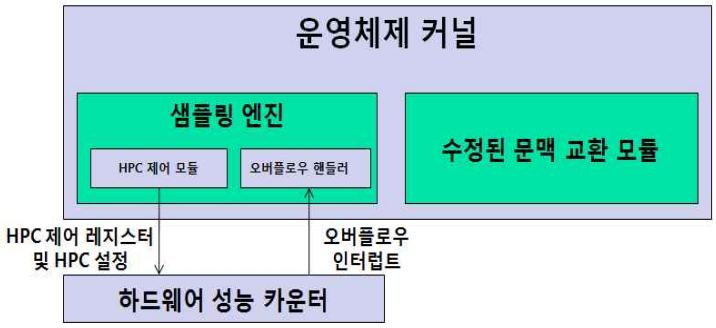

그림 1. 커널 수준의 동적 태스크 프로파일링 시스템의 블록다이어그램.

그림 1은 커널 수준의 동적 태스크 프로파일링을 위해 수정된 리눅스 커널의 블록 다이어그램이다. 본 시스템은 크게 샘플링 엔진과 수정된 문맥 교환 모듈(context_switch 매크로)로 구성되어 있다. 샘플링 엔진은 타겟 프로세서의 HPC 제어 레지스터를 사용 용도에 맞게 설정하고 HPC를 제어하는 역할을 담당하고 있으며, 수정된 문맥 교환 모듈은 멀티쓰레딩을 효율적으로 지원하기 위하여 각 태스크 별 프로파일 데이터를 관리하는 역할을 담당하고 있다.

3.2 HPC 멀티플렉싱 구현

대부분의 프로세서에서 제공하는 HPC 레지스터의 수는 측정할 수 있는 하드웨어 성능 이벤트 수에 비하여 적다. IBM PowerPC 64-bit 프로세서의 경우 수십 개의 측정 가능한 하드웨어 성능 이벤트를 제공하는 것에 반해 각 프로세서에서 제공하는 HPC 레지스터의 수는 4개에서 8개로 제한되어 있으므로 동시에 다양한 성능 정보를 실시간 관찰하는 것에는 무리가 있다. 임베디드 시스템의 경우 이러한 제약이 더욱 크게 작용한다. 본 논문에서 주요 대상으로 하는 ARM MPCore는 4개의 코어를 탑재하고 1MB의 공유 2차 캐시를 갖추어 임베디드 시스템으로써는 상당히 고성능의 시스템임에도 불구하고 각 코어 당 2개의 HPC 레지스터와 한 개의 싸이클 카운터(이하 CCNT)만을 제공하고 있다. 측정 가능한 하드웨어 성능 이벤트가 15개임을 감안하면 상당히 적은 수의 카운터가 제공됨을 알 수 있다.

표 1. MPCore 전력 예측 모델에서 사용되는 성능 정보와 각 성능 정보에 필요한 이벤트의 개수.

성능 정보	설명	필요한 하드웨어 성능 이벤트의 개수
Instructions (Inst.)	수행된 명령어의 개수	1
DL1 Cache Accesses (DL1A)	L1 데이터 캐시 접근 빈도 수	2
L2 Cache Accesses (L2A)	L2 캐시 접근 빈도 수	2
Data Dependency Stalls (Stall)	파이프라인의 데이터 의존성에 의한 스톨 된 싸이클	-
Coherence Transactions (CT)	각각의 코어의 개별 캐시의 일관성 유지를 위한 통신의 빈도 수	3

앞에서 언급한 바와 같이 이전 연구에서 제안된 ARM MPCore의 전력 예측 모델은 총 5가지의 성능 정보를 선택하였다. 표1은 에너지 모델에서 선택한 각 성능 정보가 어떠한 것이며 몇 개의 하드웨어 성능 이벤트의 조합으로 구성되어 있는지 나타낸 것이다. 표에서 알 수 있듯이 총 7가지의 이벤트(중복 1개)가 필요하며 2개의 HPC 레지스터만이 제공되는 MPCore 시스템에서 동적으로 태스크 전력 프로파일을 구성하기 위해서는 HPC 멀티플렉싱의 구현이 필수적이다.

HPC 멀티플렉싱은 특정 시간 구간 T 동안의 하드웨어 성능 이벤트의 측정 값이 T 구간 내의 상대적으로 짧은 시간 t 동안 실제 HPC 레지스터에 의해 측정된 값으로 대표될 수 있다는 것을 이용한다. 예를 들어 하나의 HPC 레지스터를 이용하여 n 개의 하드웨어 성능 이벤트를 측정하고자 한다면, t 주기마다 한번씩 n개의 이벤트를 번갈아 할당하며 각 T 구간을 측정한 후, 합한 값에 n을 곱하는 방법으로 전체 시간 동안의 각각 하드웨어 성능 이벤트 값을 추정한다. 그림2는 시간 기반 HPC 멀티플렉싱의 예를 나타낸 것이다. 총 4개의 하드웨어 성능 이벤트를 한 개의 HPC 레지스터로 측정하고자 할 때, 각각의 t 구간 동안 총 4회 성능 정보 이벤트를 HPC에 할당함으로써 이벤트를 교체하므로 T는 t의 4배인 값이 된다. 이 상황에서 첫 번째 성능 이벤트에 대한 측정값이 t_1, t_2 각각 V_1, V_2 라고 할 때, 전체 구간 $T_1 + T_2$에서의 첫 번째 성능 이벤트의 값은 $(v_1 + v_2) \times 4$로 추정할 수 있다. 그러나 이러한 기법을 사용하기 위해서는 각 t 동안 실제 측정되는 값 V가 최소한 T만큼은 크지 않은 변동을 보인다는 가정이 매우 중요하다. 따라서 성능에 큰 해를 끼치지 않으면서도 충분히 짧은 t를 선택하는 것이 높은 정확도에 절대적인 영향을 끼친다.

샘플링 엔진은 커널 내부에 존재하며 HPC 멀티플렉싱의 역할을 담당하는 모듈이다. 32비트의 CCNT가 오버플로우 될 때마다 인터럽트를 발생시킨다는 것을 이용하여 짧은 시간 t의 주기마다 한번씩 HPC에 할당된 하드웨어 성능 이벤트를 교체하는 메커니즘을 사용한다. 예를들어 0xFFFF0000를 CCNT에 할당한다면 기존의 0xFFFFFFFF마다 발생하던 오버플로우를 절반의 시간만에 발생하는 것으로 바꿀 수 있다. 리눅스에서 제공하는 타이머와 시그널을 이용하는 방법도 있지만, 오버플로우 인터럽트가 발생한 시점과 실제 처리되는 시점의 차를 최소화하기 위하여 이러한 방식을 선택하였다. 또한 MPCore와 같은 멀티코어프로세서의 경우 HPC가 각 코어마다 존재한다. (MPCore의 경우 각 코어마다 2개) 따라서 각 코어의 HPC에서 발생한 오버플로우 인터럽트는 그 코어에서 처리되도록 고정하는 것이 필요하다.

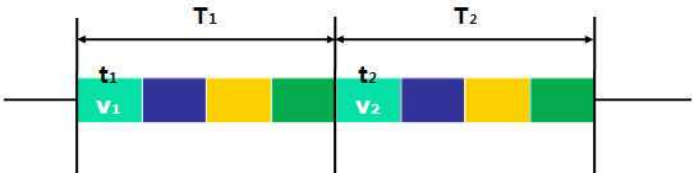

그림 2. 시간 기반의 HPC 멀티플렉싱 예제.

리눅스는 여러 개의 태스크가 스케쥴러에 의하여 CPU를 할당받는 멀티태스킹 운영체제이다. 따라서 각 태스크가 CPU를 할당받는 동안 측정된 이벤트 값은 그 태스크의 프로파일로 저장될 수 있도록 관리되어야 하며, 샘플링 주기와 그 주기 동안 HPC에 할당할 성능 정보 이벤트 역시 태스크 별로 관리되어야 한다. 이러한 역할은 수정된 문맥 교환 모듈이 담당하며 이것을 바탕으로 샘플링 엔진이 태스크 별 HPC 멀티플렉싱을 할 수 있다.

4. 실험 및 평가

본 장에서는 이전 연구에서 제안된 ARM11 MPCore 에너지 예측 모델을 기반하여 구현된 태스크 별 커널 수

준 프로파일링의 샘플링 주기에 따른 오버헤드와 전력 프로파일의 정확성을 4개의 SPLASH-2 [8] 벤치마크에 대하여 실험하고 평가하였다.

4.1 실험 환경

ARM11 MPCore는 4개의 코어가 하나의 다이에 집적된 칩 멀티프로세서이다. 온-칩 개별 1차 캐시와 공유 2차 캐시, 오프-칩 메모리로 구성되어 있으며, 모든 코어는 각각 2개씩의 HPC 레지스터와 1개의 싸이클 카운터를 포함하고 있다. 운영체제는 SMP를 지원하는 ARM용 Linux Kernel 버전 2.6.17을 사용하여 커널 수준의 HPC 프로파일링을 구현하였다. 언급한 실험 환경을 표2에 정리하였다.

표 2. ARM11 MPCore의 구성 및 구현에 사용된 커널.

프로세서	• 4개의 ARM11 프로세서 • 각각 210MHz의 클럭 • 2개의 HPC와 1개의 CCNT가 각 코어에 포함
메모리	• 32KB의 명령어 및 데이터 개별 1차 캐시 • 1MB의 공유 2차 캐시 • 오프-칩 메모리 • Snoop Control Unit
운영체제 커널	• linux-2.6.17-arm1-smp28

4.2 샘플링 주기에 따른 오버헤드

샘플링 주기는 HPC 멀티플렉싱을 사용하여 구성한 태스크의 프로파일 정확도에 가장 직접적인 영향을 끼치는 요소이다. 짧은 주기를 선택할수록 성능 이벤트의 역동적인 변화를 탐지할 수 있지만, 그만큼 많은 오버플로우 인터럽트 핸들러를 호출해야 하므로 태스크의 전체적인 성능에 악영향을 끼칠 수 있다. 이러한 악영향은 다시 성능 이벤트 값을 증가시키는 주변 효과를 일으키므로 대상 장치에 맞는 주기를 선택해야 한다.

그림3은 샘플링 주기의 변화가 태스크의 전체 수행 시간에 끼치는 영향을 나타낸 그래프이다. 10ms, 5ms, 1ms, 0.5ms, 0.1ms의 총 5개 주기에 대하여 샘플링 엔진으로 인한 수행 시간 증가를 백분율로 표시하였으며, 0.1ms의 경우 수행 시간 대비 평균 27.75%, 최대 44.01%의 정도로 성능에 큰 악영향을 끼치는 것으로 확인되어 그래프에 표시하지 않았다. 이후의 정확도 실험에서도 0.1ms 주기는 사용할 수 없는 주기로 판단하고 실험에서 제외하였다. 그래프에 따르면 전체적으로 샘플링 주기가 짧아질수록 성능에 나쁜 영향을 끼치는 것을 확인할 수 있다. GEOMEAN은 SPLASH-2 4개의 벤치마크에 대하여 각각 주기에 따른 성능 증가 비율의 기하평균을 나타낸 것으로 10ms와 5ms가 1% 미만(각각 0.13%, 0.36%), 1ms가 1.93%, 0.5ms가 4.50%의 수행 시간 증가를 보였다. 운영체제가 관리하는 모든 태스크에 대한 프로파일링 오버헤드를 나타내는 것이므로 정확도와 성능, 두 가지 모두를 만족시키는 샘플링 주기를 선택하는 것이 중요하다.

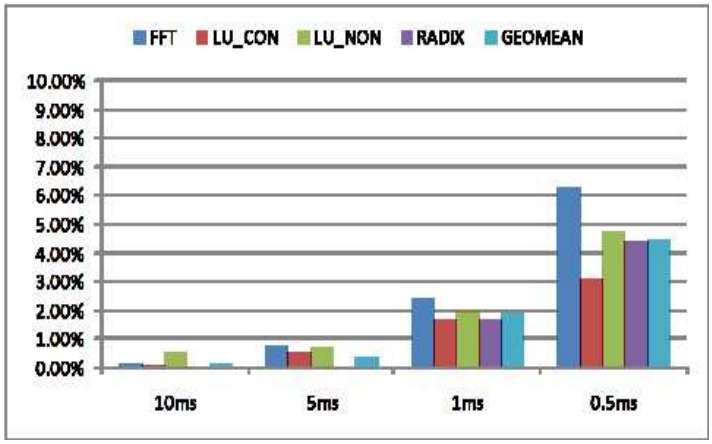

그림 3. 샘플링 주기의 변화에 따른 수행 시간 증가 비율.

4.3 커널 수준 태스크 전력 프로파일링의 정확도

표 3. 각 샘플링 주기에 따라 발생하는 오차의 백분율.

벤치마크 (주기 ms)	Inst (%)	DL1A (%)	L2A (%)	Stall (%)	Cohe (%)
FFT(10)	0.50	0.12	2.13	0.62	0.23
FFT(5)	0.68	0.25	5.13	1.45	0.41
FFT(1)	2.83	1.62	7.65	1.37	1.12
FFT(0.5)	2.89	2.18	11.25	3.42	1.43
LU_CON(10)	0.19	0.14	4.33	1.97	0.27
LU_CON(5)	0.19	0.22	4.71	0.97	0.34
LU_CON(1)	1.44	1.94	9.37	0.77	1.25
LU_CON(0.5)	3.55	1.69	12.27	1.19	1.94
LU_NON(10)	0.34	0.27	0.85	2.33	0.31
LU_NON(5)	0.36	0.39	0.27	0.13	0.40
LU_NON(1)	1.85	1.17	0.77	0.65	1.44
LU_NON(0.5)	3.03	2.09	1.27	1.00	2.04
RADIX(10)	0.51	0.35	6.39	1.80	0.40
RADIX(5)	0.23	0.55	3.63	0.76	2.49
RADIX(1)	3.20	3.38	15.00	4.03	5.09
RADIX(0.5)	3.20	3.38	15.00	4.03	5.09

샘플링 주기는 HPC 멀티플렉싱을 통해 얻는 성능 정보 값의 정확도에 상당히 큰 영향을 끼치는 요소이다. 너무 짧은 주기를 사용하면 각 성능 정보 값의 변동에 빠르게 대응할 수 있지만, 그만큼 많은 인터럽트 오버플로우 핸들러가 호출되므로 성능에 악영향을 끼칠 수 있다. 반대로 너무 긴 주기는 적은 오버헤드를 갖지만, 그만큼 샘플링 횟수가 적으므로 정확하지 않은 태스크 프로파일을 구성할 가능성이 있다.

표3은 SPLASH-2 4개의 벤치마크에 대하여 각 샘플링 주기에 따른 정확도의 차이가 어떠한지 나타낸 것이

다. 많은 주기에 대한 실험을 수행하였지만 지면의 제한으로 인해 10ms부터 0.5ms 까지 4개의 의미있는 결과만 표시하였다. 참조한 전력 모델은 위 표에서 표시한 5개의 성능 정보를 필요로 하고, 이 성능 정보를 위해서는 총 7개의 HPC 성능 정보 이벤트가 필요하다. 따라서 위 표는 샘플링 엔진을 사용하지 않고 이벤트를 직접 매 실험마다 교환하면서 각 벤치마크를 4번씩 수행하여 얻은 결과와 샘플링 엔진을 사용하여 얻은 결과가 얼마나 차이가 있는지 나타낸 백분율 값으로 클수록 정확하지 않음을 나타낸다. 표에 의하면 거의 모든 벤치마크가 5ms보다 짧은 샘플링 주기에 대해서는 오히려 큰 오차율을 보인다. 전체적으로 10ms가 가장 적은 오차율을 보이고 있는 가운데 L2A의 경우에는 벤치마크에 따라 가장 작은 오차율을 갖는 주기가 10ms인 경우와 5ms인 경우로 나뉘고 있다. 이것은 L2A (공유 2차 캐시 접근 회수 : IL1 캐시 미스 + DL2 캐시 미스)와 같이 구간에 따라 변동이 심할 여지가 있는 정보인 경우, 시간에 따른 변동이 심한 벤치마크에 대해서는 5ms가 더 적은 오차를 보이지만, 그렇지 않은 경우 오히려 10ms보다 짧은 주기는 성능에 끼치는 악영향으로 인한 부과 효과로 더 큰 오차가 발생하기 때문이다. 그림 4는 각 주기의 오차율에 대한 기하 평균을 그래프로 나타낸 것이다. 10ms, 5ms의 두 주기 모두 1% 미만의 오차율을 보이고 있으며, LU_NON을 제외하고는 모두 10ms 주기가 가장 낮은 오차율을 보이고 있다. 리눅스의 가장 낮은 고정 우선순위 태스크의 기본 타임 퀀텀이 5ms인 것을 감안하면 5ms의 주기인 경우 4번의 프로세서 할당을 받아야 전력을 구성하는 모든 성능 정보의 멀티플렉싱이 가능한 반면에 10ms의 경우 2배인 8번의 프로세서 할당이 필요하다. 따라서 오차율이 크지 않다면 되도록 짧은 샘플링 주기를 선택하는 것이 태스크 프로파일에 더 유리하다.

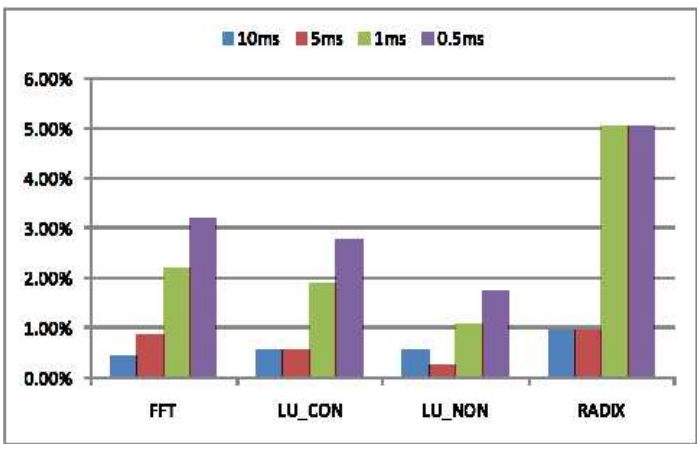

그림 4. 샘플링 주기에 따른 오차율의 기하 평균 그래프.

5. 결론 및 향후 연구

본 논문에서는 이전 연구에서 제안된 하드웨어 성능 카운터 기반의 에너지 예측 모델을 활용한 커널 수준의 태스크 별 전력 프로파일링을 구현하고 평가하였다. 성능 카운터 기반의 예측 모델은 여러 개의 성능 정보의 조합으로 이루어져 있으므로 런타임 전력 프로파일을 위해서는 동시에 여러 개의 성능 이벤트의 관찰이 필요하다. 그러나 대부분의 임베디드 시스템에서 제공되는 HPC 레지스터 수는 에너지 예측 모델이 요구하는 수에 비해 매우 적으므로 이러한 제약을 극복하기 위해 커널 수준의 HPC 멀티플렉싱을 구현하였고, 각 샘플링 주기에 따른 오버헤드와 정확도를 SPLASH-2 벤치마크에 대하여 평가하였다.

향후 전력뿐만 아니라 태스크의 특성을 나타낼 수 있는 다양한 프로파일을 탐색하고 그것을 활용하는 멀티프로세서 스케쥴러를 고안하는 방향으로 연구를 확장할 계획이다. 특히 이러한 커널 기반의 태스크 전력 프로파일링 정보를 스케줄러에 포함시켜 에너지 소모를 고려한 스케줄링 기법으로 확장될 수 있다.

감사의 글

이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다. 이 논문은 2008년도 두뇌한국21 사업에 의해 지원되었으며, 2008년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구입니다.(No. R0A-2007-000-20116-0)

참고 문헌

[1] W. Baek, Y. Kim, and J. Kim, ePRO:A Tool for Energy and Performance Profiling for Embedded Applications, in Proceeding of ISOCC, 2004.

[2] P. J. Mucci, S. Browne, C. Deane, and G. Ho, PAPI:A Portable Interface to Hardware Performance Counters. In Department of Defence HPCMP Users Group Conference, 1999.

[3] Performance data standard API. http://icl.cs.edu/projects/papi/, 2000.

[4] R. Azimi, M. Stumm, and R. W. Wisniewski, Online Performance Analysis by Statistical Sampling of Microprocessor Performance Counters, Proceedings of Supercomputing, 2005.

[5] A. Merkel and F. Bellosa, Balancing Power Consumption in Multiprocessor Systems, In Proceeding of Eurosys, 2006.

[6] 최원일, 김현희, 김지홍, 하드웨어 성능 카운터를 이용한 임베디드 멀티프로세서 환경에서의 에너지 모델링 예측 기법, 한국정보과학회 추계학술대회, 2008.

[7] ARM11 MPCore. http://www.arm.com/products/MPCoreMultiprocessor.html

[8] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, The SPLASH-2 Programs: Characterization and Methodological Considerations, In Proceeding of DAC, 2000.