[특집:기술해설_시스템 소프트웨어 기술 연구] 모바일 단말에서 태스크 상태 정보를 이용한 효율적 CPU 전력관리 방법

저자
(Authors)

김도훈, 박찬익

출처
(Source)

텔레콤 21(2), 2005.12, 10-18 (9 pages)

발행처
(Publisher)

대한전자공학회
THE INSTITUTE OF ELECTRONICS ENGINEERS OF KOREA

URL

http://www.dbpia.co.kr/Article/NODE00684483

APA Style

김도훈, 박찬익 (2005). [특집:기술해설_시스템 소프트웨어 기술 연구] 모바일 단말에서 태스크 상태 정보를 이용한 효율적 CPU 전력관리 방법. 텔레콤, 21(2), 10-18.

모바일 단말에서 태스크 상태 정보를 이용한 효율적 CPU 전력관리 방법

김도훈, 박찬익
포항공과대학교 시스템소프트웨어 연구실

I. 서론

휴대폰, 노트북, MP3 플레이어, PMP(Portable Media Player) 등의 모바일 단말에서 전력은 시스템 가용시간을 결정하는 중요한 자원이다. 모바일 단말은 이동성을 중시하므로 배터리를 통해 전력을 공급받는다. 그러나 배터리 용량 기술은 무어의 법칙이 적용되는 반도체 기술과는 달리 발전 속도가 매우 느린 실정이다. 따라서 모바일 단말의 가용시간을 증가시키기 위해서는 효율적인 전력관리 기법이 요구된다.

모바일 단말의 전력소모에서 큰 비중을 차지하는 CPU는 장치특성을 고려한 전력관리 방법이 필요하다. 일반적으로 모바일 단말의 장치는 장치 사용율이 일정 수준이하로 감소하면 전력절감을 위해 장치의 전력상태를 여러 단계의 sleep 상태로 변경할 수 있다.[1] 그러나 CPU는 I/O장치와는 달리 소프트웨어를 수행하기 위해 항상 필요한 장치이므로, sleep 상태로 변경되는 경우는 드물다. 따라서 CPU는 장치 사용율에 따라 CPU의 성능을 조절하여 전력절감 효과를 기대할 수 있는 DVS(Dynamic Voltage Scaling) 기능을 제공하고 있다[2,3]. DVS는 CPU를 구성하는 CMOS 회로의 동작원리-CPU 전력소모량은 CMOS에 인가된 전력의 제곱에 비례-에 기반하고 있으므로, 성능조절을 통해 얻을 수 있는 전력절감 효과가 크다. 만약 소프트웨어의 CPU 사용율을 미리 알 수 있다면, 소프트웨어의 수행시간에 영향을 주지 않고도 높은 전력절감 효과를 기대할 수 있게 된다. 따라서 일반적으로 CPU 전력관리는 소프트웨어의 CPU 사용율 패턴에 따라 DVS을 스케줄링하는 것을 의미하게 된다.

CPU 전력관리 기법은 DVS 스케줄링이 결정되는 시점에 따라 정적 DVS 스케줄링과 동적 DVS 스케줄링으로 구분된다.[4] 정적 DVS 스케줄링은 소프트웨어를 컴파일하는 과정에서 CPU 사용율을 분석하여 DVS 스케줄링 코드를 첨가하는 방법이다. 이 방법은 소프트웨어의 CPU 사용율을 미리 예측할 수 있는 경성 실시간 소프트웨어에서 사용된다. 그러나 일반적으로 소프트웨어의 CPU 사용율은 입력 데이터 혹은 시스템 상황에 따라 변화할 수 있다. 따라서 구동중인 소프트웨어의 CPU 사용율 변화를 실시간으로 분석하여 현재 상황에 적합한 DVS로 스케줄링하는 방법이 필요하며, 이 기법을 동적 DVS 스케줄링으로 정의한다. 동적 DVS 스케줄링은 스케줄링이 발생하는 위치에 따라 태스크내(Intra-task) DVS 스케줄링 기법과 태스크간 (Inter-task) DVS 스케줄링 기법으로 구분된다. 태스크내 DVS 스케줄링 기법은 컴파일러 혹은 운영체제 수준에서 태스크의 수행시간 정보를 추출한 후 이를 바탕으로 태스크가 수행되는 동안 매 클록마다 적절한 CPU 성능으로 조절한다.[5] 그러나 이 방법은 소프트웨어의 코드에 수정을 가하거나 태스크의 수행시간 정보를 수집하는데 오랜 시간이 걸리는 단점이 있다. 또한 최악의 경우 매 클록마다 DVS 스케줄링이 발생하여 오버헤드가 커질 수 있다는 단점이 있다. 태스크간 DVS 스케줄링 기법은 태스크를 수행하는 매 시점에서 DVS가 결정되는 기법이다. 태스크간 DVS 스케줄링 기법은 DVS 스케

줄링을 위해 필요한 태스크의 정보에 따라 세 가지로 구분된다. 첫 번째 방법은 실시간 태스크의 정보를 이용하는 방법이다.[6] 실시간 태스크의 수행시간은 일반적으로 최악수행시간(Worst-Case Execution Time)보다 짧은 것으로 알려져 있다. 따라서 이때 발생하는 슬랙(Slack)을 이용하여 DVS 스케줄링을 하는 방법이다. 그러나 이 방법은 실시간 태스크와 같은 데드라인 정보가 존재하는 시스템에서만 사용가능하므로 다양한 시스템으로의 적용가능성이 떨어진다. 두 번째 방법은 주기적으로 수행되는 시간구간을 두어, 매 구간마다 CPU 사용율 정보를 수집한 후 이를 기반으로 다음 구간의 CPU 사용율을 예측하여 DVS 스케줄링을 하는 기법이다.[7] 이 기법은 구간기반 DVS 스케줄링이라고도 불린다. 구간기반 DVS 스케줄링은 앞서 소개된 기법들과는 달리 각 시간구간에서 측정된 데이터를 기반으로 수행되므로 수행 복잡도와 오버헤드가 낮은 것으로 알려져 있다. 따라서 비교적 시스템 성능이 낮은 모바일 단말에서 사용하기에 적합한 CPU 전력관리 기법이다. 그러나 기존 구간기반 DVS 스케줄링은 시스템에서 구동중인 각 태스크의 특징을 고려하지 않으므로 전력소모 감소에 한계가 있다. 따라서 본 논문는 모바일 단말에서 수행 복잡도와 오버헤드가 적은 전력관리 기법이 필요하다는 점에서 구간기반 DVS 스케줄링 기법을 고찰하였다.

본 논문은 기존 구간기반 DVS 스케줄링 기법의 효율성을 높이기 위해 태스크당 시간구간을 두어 태스크 단위로 DVS 스케줄링을 하는 기법을 제안한다. 일반적으로 각 태스크는 CPU 자원을 사용하는 사용패턴이 있는 것으로 알려져 있다. 대표적인 예로 미디어 플레이어는 주기적인 디스크 읽기와 읽어 들인 데이터를 디코딩하는 수행패턴을 보인다. 따라서 이러한 특성을 분석하면 각 태스크에 적합한 CPU 사용율을 계산할 수 있다. 그러나 기존의 구간기반 DVS 스케줄링은 이를 고려하지 않고, 특정 구간동안 모든 태스크들에 대해 동일한 DVS를 적용함으로써 효율성이 떨어질 수 있다. 본 논문은 제안된 DVS 스케줄링 알고리즘의 효용성을 검증하기 위해 Linux 운영체제에 구현하였으며, DVS 스케줄링을 사용하지 않았을 경우에 비해 49%-61%의 전력절감효과를 보였다. 또한 기존의 구간기반 DVS 스케줄링 기법에 비해 최대 5%의 전력절감효과가 향상되었다.

본 논문은 다음과 같은 순서로 구성되었다. 2장에서는 본 논문에서 제안하고 있는 CPU 전력관리 알고리즘에 대해 서술한다. 제안된 알고리즘의 구현과 실험결과는 3장에서 서술하고, 마지막으로 4장에서 결론과 향후 계획에 대해 서술한다.

II. CPU 전력관리 알고리즘

1. 태스크의 CPU 사용율 측정기법

가. 태스크별 시간구간 추출

기존 구간기반 DVS 스케줄링 기법은 시스템 전체에 오직 하나의 시간구간만을 두어 CPU 사용율을 측정하므로 각 태스크의 특성이 고려되지 않았다. 또한 시간구간은 10ms-50ms가 적합한 것으로 알려져 있으나[8], 시스템 상황에 따라 적합한 시간구간을 동적으로 변환하는 것은 부가정보없이는 불가능한 문제이다. 따라서 본 논문은 각 태스크의 수행패턴을 분석하여, 각 태스크에 적합한 시간구간을 찾는 방법을 제안한다.

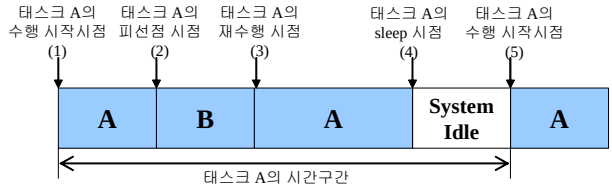


그림 1 태스크의 수행패턴 예

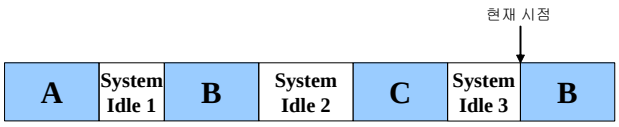
태스크의 수행 패턴은 그림1과 같이 표현할 수 있다. 태스크 A는 (1)시점에서 CPU를 할당받아 수행을 시작한다. 그리고 (2)시점에서 높은 우선순위를 가진 태스크 B에 의해 피선점된다. (3)시점에서 태스크 B는 모든 수행을 끝내고 CPU 사용권한을 태스크 A로 이양한다. 그리고 태스크 A는 CPU를 할당받아 수행을 다시 시작한다. (4)시점에서 태스크 A는 모든 수행을 마치고 사용자로부터 이벤트를 기다리는 sleep 상태로 전환된다. 그리고 이 시점부터 시스템에는 수행중인 태스크가 존재하지 않으므로 전체 시스템은 idle 상태로 전환된다. (5)시점에서 태스크 A는 사용자로부터

이벤트를 수신하여 다시 수행을 재개한다. 그림1의 태스크 수행패턴을 분석하면 sleep 이벤트를 중심으로 CPU 사용시간을 측정할 수 있음을 알 수 있다. 그리고 태스크의 sleep 이벤트들 사이에는 전체 시스템에 대한 idle 시간도 존재함을 알 수 있다. 이러한 idle 시간을 활용하면, 태스크의 CPU 사용율을 떨어뜨려 사용전력의 감소효과를 기대할 수 있다. 따라서 그림1의 시점(1)과 시점(5)사이에서 수행된 CPU 시간과 발생한 idle 시간은 태스크 A의 수행특성을 보이는 시간구간으로 해석할 수 있다. 각 태스크는 시간구간을 확정하기 위해 태스크가 sleep 후 다시 CPU를 할당받기 전까지 발생한 idle 시간을 자신의 시간구간 후보값으로 유지한다. 이를 다음과 같이 나타낼 수 있다.

태스크 시간구간(I) = sleep전 CPU 사용시간(C)
+ sleep후 시스템 idle 시간(D)

나. system idle 시간 분배

각 태스크는 CPU 사용율을 낮추어 전력소모를 최소화하기 위해 idle 시간을 자신의 시간구간 후보값으로 편입한다. 따라서 DVS 스케줄링시 idle 시간을 어느 태스크의 시간구간으로 확정하느냐에 따라 CPU의 전력소모량이 달라질 수 있다.



Accumulated idle for each task

A = idle 1 + idle 2 + idle 3
B = idle 2 + idle 3
C = idle 3

그림 2 각 태스크의 idle 편입 예

그림2는 각 태스크가 시스템에서 발생한 idle 시간을 자신의 idle 후보값으로 가지고 있는 예를 나타낸 것이다. 태스크 A는 가장 먼저 sleep 상태로 전환되었으므로, 현재 시점까지 발생한 모든 idle을 자신의 시간구간에 포함할 수 있다. 태스크 B는 A 다음에 sleep 상태로 전환되었으며, 현재시점까지 발생한 idle 중 idle2와 idle3를 자신의 시간구간에 포함할 수 있다. 마찬가지 방법으로 태스크 C는 idle3을 자신의 시간구간에 포함할 수 있다. 그리고 현재시점에서 태스크 B가 CPU를 할당받아 재수행하게 되었을 때, idle2와 idle3를 모두 자신의 시간구간으로 확정할 수 있다. 그러나 만약 idle2와 idle3를 모두 자신의 시간구간으로 확정하더라도 CPU 전력소모량에 영향을 주지 않는다면, 이를 태스크 A, C로 이양하는 것이 전체 시스템 수준의 전력소모를 감소할 수 있는 방법이 된다. 그러나 최적의 idle 분배 방법은 복잡한 계산이 필요한 방법이므로 모바일 단말에서는 부적합할 수 있다. 우선 본 논문은 이 문제를 향후 계획으로 남긴다. 대신 본 논문에서는 가장 먼저 수행되는 태스크가 자신의 후보값으로 가지고 있는 idle 시간을 시간구간으로 확정하는 것으로 설정하였다.

다. CPU 사용율 계산 및 추정

시간구간동안 측정된 태스크의 CPU 사용율은 다음과 같이 나타낼 수 있다.

$$\text{태스크의 CPU 사용율(U)} = \frac{C}{I}$$

시간구간동안의 CPU 사용율 계산은 태스크가 sleep 상태에서 CPU를 다시 할당받아 수행되는 시점, 즉 그림1의 (5)시점에서 발생한다. 그리고 (5)시점에서 향후 시간구간동안 태스크 A의 CPU 사용율을 추정하여 이에 적합한 CPU 성능으로 DVS 스케줄링을 하게 된다. 다음 시간구간에 대한 CPU 사용율을 계산하는 방법은 K. Govil[7]에 소개한 PAST와 AGED_AVERAGE 기법을 사용하여 계산한다. 지난 시간구간동안 측정된 CPU 사용율을 U_{last}, 계산을 통해 추정된 CPU 사용율을 U^*라 할 때, PAST와 AGED_AVERAGE 값은 다음과 같은 수식으로 계산된다.

$$PAST(U^*) = U_{last}$$

$$AGED(U^*) = \frac{(k-1)\cdot U^* + U_{last}}{k}$$

AGED_AVERAGE에서 상수 k는 최근에 측정된 값이 향후 추정될 값의 결정에 얼마나 영향을 줄 것인지를 의미한다. 본 논문에서는 실험에서 k의 값을 4로 설정하여 진행되었다.

1. 태스크 T_i가 CPU를 할당받았을 때,
   if(T_i가 sleep상태 후 CPU를 할당 받았는가 or // 새로운 시간구간이 시작되는 것인지 검사
   T_i가 CPU를 연속해서 50ms이상 사용하고 있는가) // 오랜 시간동안 CPU를 사용할 경우 응답시간 향상을 위해 정보들을 재계산하게 한다
   {
   T_i가 가지고 있는 idle 후보값 D를 다음과 같은 과정을 통해 자신의 시간구간으로 확정한다.
   for(모든 태스크들에 대해) {
   T_i보다 먼저 sleep 상태로 전환된 태스크들의 idle D_k를 다음과 같이 변경한다.
   $$D_k = D_k - D$$
   T_i보다 늦게 sleep 상태로 전환된 태스크들의 idle D_k를 다음과 같이 변경한다.
   $$D_k = 0$$
   }
   다음 수식에 의해 시간구간과 CPU 사용율을 계산한다.
   $$I = C + D,\ U = \frac{C}{I}$$
   다음구간에 대한 CPU 사용율(U^*)을 추정기법(PAST, AGED_AVERAGE)을 이용하여 계산한다.
   다음구간에 대한 태스크의 CPU 성능(CPU_{freq})을 다음과 같이 계산한 후 해당 성능으로 CPU를 설정한다.
   $$CPU_{freq} = U^* \times Maximum\ CPU\ frequency$$
   } else // 즉, 선점 등으로 인해 잠시 CPU를 양보했던 태스크가 다시 CPU를 할당받아 수행하는 경우
   현재 태스크를 위해 계산되어 있는 CPU_{freq}로 CPU 성능을 재설정한다.

2. 전체 시스템이 idle 상태로 변했을 때,
   for(모든 태스크들에 대해) {
   각 태스크의 idle 후보값 D_k를 idle 시간만큼 증가시킨다.
   }

그림 3 본 논문의 제안 알고리즘

2. CPU 전력관리 알고리즘

그림3은 각 태스크를 위한 시간구간, CPU 사용률 추정 등의 알고리즘을 기술하였다. 우선 태스크 T_i가 CPU 스케줄러에 의해 CPU를 할당받았을 때 sleep 상태 후 받은 것인지 검사하게 된다. 만약 그렇다면, T_i는 새로운 시간구간으로 시작됨을 알 수 있다. 따라서 이때 시간구간을 계산한 후 이를 기반으로 지난 구간동안의 CPU 사용율을 계산한다. 그 후 다음 시간구간의 CPU 사용율을 PAST 혹은 AGED_AVERAGE을 이용하여 계산한다. 그리고 마지막으로 CPU 성능을 조절한다. 만약 태스크 T_i가 CPU를 할당받았을 때, sleep 상태를 거치지 않았다면 아직 현재의 시간구간이 끝나지 않았음을 의미한다. 따라서 지난 시간구간에 결정된 CPU 성능으로 다시 복구하여 수행하도록 한다. 그리고 한 가지 더 부연하자면, 현재 수행중인 태스크가 CPU를 연속하여 50ms 이상 할당받아 수행중이라면, 시간구간 및 CPU 사용율을 재계산하도록 하였다. 그 이유는 이미 D. Grunwald[8]에서 밝혔듯이, 낮은 CPU 성능으로 설정된 대화형 태스크(Interactive Task)의 경우 50ms보다 응답시간이 늦어지게 되면 문제가 발생할 수 있음을 명시하고 있다. 따라서 이러한 문제를 미연에 방지하기 위해 연속해서 50ms 이상 수행할 경우 시간구간을 현재 시점으로 재설정하고, CPU 사용율 등을 재계산한 후 해당 CPU 성능으로 조절하게 한다.

III. 시스템 구현 및 실험

1. CPU 전력관리 알고리즘 구현

본 논문은 제안된 DVS 알고리즘과 기존 구간기반 DVS 알고리즘[7]을 Linux 커널 2.6에 구현하였다. 우선 태스크의 정보를 추출하기 위해 그림4와 같은 데이터 구조가 커널 내부에 추가 되었다.

```
Task Information Tracking Table(TIIT) {
   sleep_time;
   sleep;
   cpu_freq;
   run_time;
   idle_time;
   util_history;
}
```

그림 4 태스크 정보 추출을 위해 추가된 데이터 구조

sleep_time은 태스크가 sleep 상태로 전환된 시간을 의미한다. sleep은 태스크가 sleep 상태로 전환된 적이 있음을 나타낸다. cpu_freq는 제안된 DVS 알고리즘에 의해 계산된 CPU 성능을 나타낸다. run_time은 태스크의 시간구간동안 수행된 CPU 사용시간, 즉, 클록수를 나타낸다. idle_time은 태스크의 지난 시간구간동안 sleep 후 발생한 system idle 클록의 수를 나타낸다. util_history는 PAST, AGED_AVERAGE 기법을 통해 CPU 사용율을 계산하기 위해 과거에 계산된 값을 저장한다.

본 논문은 제안된 알고리즘을 구현하기 위해 Linux 커널의 update_process_times()를 수정하였다. Linux는 PIT(Programmable Inteval Timer)를 이용하여 매 10ms 혹은 설정에 따라 타이머 인터럽트를 처리한다. 타이머 인터럽트를 처리하는 과정에서 현재 수행중인 태스크의 CPU 사용시간 정보를 갱신하게 되고, 이 과정 중에 update_process_times()를 호출하여 수행한다. 본 논문은 제안된 DVS 스케줄링 알고리즘을 update_process_times()함수를 수행하는 첫 라인에 삽입하여 기존 Linux 커널의 수정을 최소화하였다. 또한 현재 수행중인 태스크의 CPU 사용시간을 갱신하기 위해 update_pro cess_times()에 TITT의 CPU 사용시간을 증가하는 코드를 삽입하였다. 마지막으로 시스템 전체에 대한 idle은 update_process_times() 수행시 Linux 커널의 데이터 구조 중 현재 실행중인 태스크를 나타내는 current 구조가 idle 태스크를 가르킬 경우 이를 idle 시간으로 판단하였으며, TITT의 idle_time 값을 갱신하도록 하였다. 그리고 기존 구간기반 DVS알고리즘은 전체 시스템에 오직 한 가지의 시간구간만을 유지하므로, K.Y. Mun[9]와 같이 커널 내부에서 제공하는 동적 타이머 함수를 이용하여 구현하였다. 이때 기존 구간기반 DVS 알고리즘의 구간은 Grunwald[8]에서 제안한 대로 50ms로 설정하였다.

2. 실험

가. 실험환경

실험을 위해 하드웨어 환경으로써 소니 VAIO-C1VJ 노트북을 사용하였다. VAIO-C1VJ 노트북은 DVS 기능을 탑재한 Transmeta사의 크루소 CPU TM5600을 내장하고 있다. TM5600은 300MHz~600MHz의 CPU 성능을 제공하고 있으며, 각 성능 별 전력소모량은 그림5와 같다.

지원 CPU 성능	전압
300 MHz	1.3 V
400 MHz	1.35 V
500 MHz	1.4 V
600 MHz	1.6 V

그림 5 TM5600의 지원 CPU 성능 및 전력소모량

운영체제는 Rehat Linux 7.3을 설치하였으며, 커널은 갱신패치를 통해 2.6으로 설정하였다. 그리고 실험동안 타이머 인터럽트는 10ms마다 발생하도록 하였다. 따라서 각 태스크들의 수행시간은 10ms 단위로 결정된다.

실험을 위해 사용된 작업부하는 세 가지가 사용되었다. 우선 I/O 작업이 많은 태스크 환경에서 성능을 측정하기 위해 파일 I/O 성능 측정 도구인 IOzone v3.2을 사용하였다. IOzone은 100MB 크기의 파일에 대해 512KB단위의 I/O를 수행하

도록 설정되었다. 다음으로 일반적으로 많이 사용되는 멀티미어용 태스크에 대한 실험을 진행하기 위해 Linux 설치시 기본으로 제공되는 Mplayer v0.9를 사용하였다. Mplayer는 모든 실험에 대해 동일하게 동작하도록 하게 위해 프레임 속도를 초당 17개로 설정하였다. 다음으로 여러 개의 태스크가 수행되는 환경을 실험하기 위해 IOzone과 Mplayer를 동시에 수행하여 실험이 진행되었다.

작업부하 종류	수행시간(초)	CPU idle 비율(%)	CPU busy 비율(%)
IOzone	92.6	86.1	14.9
Mplayer	81	58.9	41.1
Mixed (IOzone+ Mplayer)	100	54.1	45.9

그림 6 실험에 사용된 작업부하의 특성

모든 실험은 신뢰성을 위해 4번씩 수행한 후 평균값으로 나타냈다. 우선 각 작업부하의 특성을 살펴보기 위해 CPU 전력관리 알고리즘이 없는 상태에서 그림6과 같은 실험값이 측정되었다.

나. 실험결과

그림7은 각 알고리즘들을 수행했을 때, 그림6의 항목인 CPU idle과 busy 비율을 나타낸 것이다. 수행시간은 그림6과 거의 동일하므로 그림7에서는 생략되었다. 이 실험에서 busy 항목은 CPU 성능과 상관없이 각 작업부하 태스크에 의해 CPU가 사용된 비율을 의미한다. 실험값을 살펴보면 그림7의 CPU busy 비율이 그림6에 비해 현저하게 증가된 모습을 볼 수 있다. 이는 CPU 전력관리 알고리즘에 의해 CPU의 사용율이 증가되었음을 의미한다.

구분		기존 DVS 알고리즘		본 논문의 제안 알고리즘	
사용된 CPU 사용율 추정기법	작업부하	CPU idle	CPU busy	CPU idle	CPU busy
PAST	IOzone	83.7	16.3	83.7	16.3
	Mplayer	40.8	59.2	38	62
	Mixed	34.2	65.8	31.9	68.1
AGED_ AVERAGE	IOzone	83.6	16.4	83.5	16.5
	Mplayer	41	59	40.2	59.8
	Mixed	34.5	65.5	33.3	66.7

그림 7 각 CPU 전력관리 알고리즘을 수행했을 때 측정된 CPU idle과 busy 비율(단위: %)

구분		기존 DVS 알고리즘				본 논문의 제안 알고리즘			
사용된 CPU 사용율 추정기법	작업부하	300	400	500	600	300	400	500	600
PAST	IOzone	86.3	2	1.3	10.4	88.6	0.1	0.1	11.2
	Mplayer	31.8	43.9	17.9	6.4	70.1	8.8	5	16.1
	Mixed	24.5	31.5	20.8	23.2	55.7	7	4.3	33
AGED_ AVERAGE	IOzone	87	1.6	1.4	10	88.2	0.4	1.4	10
	Mplayer	30.6	45.5	17.6	6.3	60.2	19.9	4.5	15.4
	Mixed	21.4	33.1	22.2	23.3	45.1	17.3	7	30.6

그림 8 각 CPU 전력관리 알고리즘을 수행했을 때 측정된 각 CPU 성능별 수행 비율(단위: %)

그림8은 각 알고리즘을 수행했을 때 선택된 CPU 성능의 비율을 나타낸 것이다. 본 논문의 알고리즘은 기존 DVS 알고리즘에 비해 300MHz와 600MHz의 수행 비율이 높은 것으로 나타난다. 그 이유는 다음과 같다. 기존 DVS 알고리즘으로 수행했을 때 400MHz와 500MHz로 선택되었던 수행패턴들은 sleep 상태였던 태스크가 다시 CPU를 할당받아 수행되는 이벤트를 포함하고 있었다. 그리고 이러한 이벤트는 항상 디스크 I/O로 인해 긴 시간의 idle 시간을 포함하고 있는 것으로 나타났다. 따라서 본 논문의 알고리즘은 이러한 수행패턴을 sleep 이벤트를 기점으로 다시 여러 개의 시간구간으로 나누게 되므로 긴 idle 시간을 포함한 시간구간은 300MHz의 CPU 성능을 설정되므로 선택횟수가 증가한다. 그리고 본 논문의 알고리즘은 특정 태스크가 연속해서 50ms 이상의 긴 CPU 사용시간을 가질 경우, 시간구간 및 CPU 사용율을 재계산하므로, 600Mhz의 비율이 높아지게 된다. 이로부터 기존 DVS 알고리즘보다 높은 응답시간을 예상할 수 있다.

그림9는 각 알고리즘으로 실험을 진행했을 때 얻어진 CPU 전력소모 절감율을 나타낸 것이다. 우선 각 알고리즘은 47%~61%의 높은 전력절감율을 보이고 있다. 그 이유는 작업부하의 수행패턴에 있다. 그림6에서 알 수 있듯이 각 작업부하는 많은 idle 시간이 존재하므로, CPU 전력관리 알고리즘에서 이를 활용하여 CPU의 성능을 낮게 설정하였기 때문에 전력절감효과가 높게 나타났다. 그리고 본 논문에서 제안한 알고리즘은 기존 DVS 알고리즘보다 최고 5% 전력절감 효과가 발생했음을 알 수 있다. 그러나 본 논문에서 제안한 알고리즘은 다른 환경에서 수행할 경우 더욱 높은 전력절감 효과가 발생할 것으로 기대된다. 그 이유는 그림8에서 알 수 있듯이 실험에서 사용된 작업부하가 본 논문의 알고리즘으로 하여금 높은 CPU 성능을 선택하도록 하였기 때문이다. 따라서 다른 수행패턴을 보이는 작업부하를 사용할 경우 더 높은 전력절감효과가 기대된다. 다른 작업부하를 통한 실험은 향후 계획으로 남긴다.

결론 및 향후 계획

본 논문은 기존 DVS 스케줄링 알고리즘과는 달리 태스크의 특성을 고려한 알고리즘을 제안하고 있다. 이를 위해 태스크의 수행 패턴을 분석하여 시간구간을 추출하는 기법을 제안하였다. 또한 CPU 전력소모량을 줄이기 위해 system idle 시간을 각 태스크에 배분하는 기법을 제안하였다. 본 논문의 알고리즘은 CPU 전력관리를 사용하지 않았을 경우 보다 49%~61%의 전력절감 효과를 보였다. 또한 기존 DVS 알고리즘과 비교하였을 때, 최고 5%의 전력절감 효과가 나타났음을 보였다.

본 논문에서 제안하고 있는 알고리즘은 아직 많은 연구가 필요하다. 특히 태스크의 수행 패턴 분석과 system idle 시간 배분 기법은 전력절감 효과에 큰 영향을 미치는 부분이므로 이에 대한 연구가 현재 진행되고 있다. 또한 본 논문의 알고리즘을 검증하기 위해 보다 많은 작업부하에서 실험이 진행되고 있다.

구분		기존 DVS 알고리즘	본 논문의 제안 알고리즘	
사용된 CPU 사용율 추정기법	작업부하	전력절감비율	전력절감비율	기존 DVS 알고리즘 비교
PAST	IOzone	61	61.5	1.2
	Mplayer	50.1	53.4	5
	Mixed	48.1	49.5	2.8
AGED_AVERAGE	IOzone	61.8	61.8	-0.1
	Mplayer	50.8	52.4	3.3
	Mixed	47.5	49.4	3.6

그림 9 전력절감효과 비교표(단위: %)

이 논문은 대학 IT연구센터 육성지원사업의 연구결과로써 HY-SDR연구센터의 연구비 지원으로 수행되었습니다.

참 고 문 헌

[1] APM(Advanced Power Management), ACPI(Advanced Configuration & Power Interface), http://www.acpi.info/
[2] Transmeta Crusoe LongRun Tchnology, http://www.transmeta.com
[3] XScale Dynamic Voltage Management, http://www.intel.com/design/intelxscale/
[4] J. M. Rabaey and M. Pedram, Low Power Design Methodlogies, Norwell, MA:Kluwe r, 1996
[5] A. Azevedo and I. Issenin, Profile-based dynamic voltage scheduling using progra m checkpoints, Proceedings of design automation and test in Europe, March, 2002
[6] T. Okuma and T. Ishihara and H.Yasuura, Real-time task scheduling for a variable voltage processor, Proceedings of the international symposium on system synthesis, November, 1999
[7] K. Govil and E. Chan and H. Wasserman, Comparing algorithms for dynamic speed-setting of a low-power CPU, Proceedings of the first international conference on mobile computing and networking, November, 1995
[8] D. Grunwald, P. Levis, K. Farkas, C. B. Morrey III, and M. Neufeld. Polices for Dynamic Clock Scheduling, Proceedings of the First International Conference on Mobile Computing and Networking, November, 1995
[9] K.Y. Mun, D.W, Kim, D.G. Kim and C.I. Park, dDVS: An Efficient Dynamic Voltage Scaling Algorithm on the Differential of CPU Utilization, Proceedings of Ninth Asia-Pacific Computer Systems Architecture Conference (ACSAC2004), September 2004

筆 者 紹 介

김 도 훈

1998년 충남대학교 컴퓨터공학과 학사
2001년 포항공과대학교 컴퓨터공학과 석사
2001년~ 현재 : 포항공과대학교 컴퓨터공학과 박사과정

박 찬 익

1983년 서울대학교 전자공학과 학사
1985년 한국과학기술원 전자공학(컴퓨터공학전공) 석사
1988년 한국과학기술원 전자공학(컴퓨터공학전공) 박사
1989년~현재 포항공과대학교 컴퓨터공학과 교수