한국산학기술학회논문지
Vol. 7, No. 5, pp. 866-874, 2006

# 비주기 태스크를 위한 동적 가변 전압 스케줄링

권기덕[1*], 정준모[2], 권상홍[3]

# A Dynamic Voltage Scaling Algorithm for Aperiodic Tasks

**Ki-Duk Kwon[1*], Jun-Mo Jung[2] and Sang-Hong Kwon[3]**

**요 약** 본 논문은 비주기 태스크에 대한 저전력 스케줄링을 달성하기 위한 새로운 동적 전압 조절(DVS) 알고리즘을 제안한다. 비주기 태스크는 주기(period)가 없고 발생시간(release time)과 최악실행시간(WCET) 예측이 불가능하기 때문에 기존의 DVS 알고리즘으로 스케줄링 할 수 없으므로 전력소모가 많이 발생하는 단점이 있다. 본 논문에서는 일정한 크기의 주기와 최악수행시간을 갖는 주기적인 가상태스크를 정의하고, 발생한 비주기 태스크를 가상태스크에 할당하여 이미 존재하는 주기 태스크들과 함께 DVS 스케줄링을 수행하는 알고리즘을 제안한다. 가상태스크의 주기와 최악수행 시간은 이미 존재하는 주기태스크들과 가상태스크를 모두 포함한 태스크 활용률을 계산하여, 그 값이 1에 가장 근접하는 값으로 설정한다. 제안하는 알고리즘은 기존의 주기 태스크에 대한 DVS 알고리즘보다 11%의 전력 감소 효과가 있음을 시뮬레이션을 통해 확인하였다.

**Abstract** This paper proposes a new Dynamic Voltage Scaling(DVS) algorithm to achieve low-power schedulingof aperiodic hard real-time tasks. Aperiodic tasks schedulingcannot be applied to the conventional DVS algorithm and result in consuming energy more than periodic tasks because they have no period, non predictable worst case execution time, and release time. In this paper, we defined Virtual Periodic Task Set(VTS) which has constant period and worst case execution time, and released aperiodic tasks are assigned to this VTS. The period and worst case execution time of the virtual task can be obtained by calculating task utilization rate of both periodic and aperiodic tasks. The proposed DVS algorithm scales the frequency of both periodic and aperiodic tasks in VTS. Simulation results show that the energy consumption of the proposed algorithm is reduced by 11% over the conventional DVS algorithm for only periodic task.

**Key words:** Dynamic Voltage Scaling, Aperiodic Tasks, RM, EDF, Virtual Task Set

## 1. 서론

대부분의 실시간 시스템들은 제한된 전력 공급 장치를 사용하는 휴대용 내장형 시스템이기 때문에 전력을 효율적으로 사용하는 저전력 구현 기법에 대한 연구가 중요한 문제로 대두되고 있다. 프로세서들의 성능이 높아짐에 따라 소비되는 전력량도 증가하여 실시간 시스템이 소비하는 전력의 많은 부분을 프로세서가 차지한다. 따라서 프로세서의 공급 전압을 낮추어 프로세서에서 소비되는 전력량을 줄이는 동적 전압 조절 기법(DVS)이 실시간 시스템의 저전력 기법으로 많이 연구되고 있다.

요즘 출시되는 대부분의 프로세서들은 여러 전압레벨에서 수행될 수 있는 기능을 탑재하고 있다. 예를 들어, ARM7D 프로세서는 3.3V일 때 20MHz, 5V일 때 33MHz로 MHz로 동작한다.[4] TM5400과 Crusoe 프로세서[10]는 1.1V에서 1.65V까지 16레벨을 사용할 수 있다. 인텔의 StrongARM SA-2는 150MHz와 600MHz에서 동작할 수 있다. 이때 소비되는 전력은 각각 40mW와 500mW이다.[1] 1/4배로 속도를 줄이면 1/12배로 소비전력이 감소함을 알 수 있다. Speedstep 기능을 지닌 Pentium-III 프로세서는 650MHz에서 22W를 소비하지만 500MHz에서는 불과 9W밖에 소모하지 않는다.[8]

최근 몇 년 동안 DVS 에 대해 많은 연구가 진행되고

[1]와세다대학교 컴퓨터정보학과
[2]군산대학교 전자정보공학과
[3]한국폴리텍 구미대학 컴퓨터정보학과
*교신저자: 권기덕(pugara@dcl.info.waseda.ac.jp)

있다.[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 그 결과 실시간 시스템의 저전력 구현을 위한 여러 가지 DVS 알고리즘들이 발표되었다. [7]은 프로세서의 속도를 동적으로 변화시켜 실시간 스케줄링하는 기법을 최초로 제안하였다. [3]은 EDF 스케줄링 알고리즘[11]에 기반한 DVS를 전개하였으며, 비선점형 스케줄러에 기반한 전압 조절 방안[4], 고정 우선순위 스케줄러에 대한 DVS[5] 등이 연구되었다. RM 스케줄러[12]에 대해서도 전압을 조절하는 방안이 연구되고 있다.[6] 또한 Ishihara 와 Yasuura는 실시간 시스템의 전력 소모량을 최소화시키기 위해서는 대부분의 경우 2개의 공급 전압이면 충분하다는 이론을 발표했다.[2]

일반적으로 CMOS로 구성된 회로의 소비 전력은 식 (1)과 같이 회로에 공급되는 전압의 제곱에 비례한다. 여기서 $C_L$, $V_{DD}$, $f$는 각각 출력 커패시턴스, 공급 전압, 클럭 주파수를 의미한다. 또한, 지연 시간은 식 (2)와 같이 나타낼 수 있는데, 여기서 $V_T$는 문턱 전압을 의미하며, $k$는 출력 게이트 크기와 출력 커패시턴스에 의해 결정되는 상수값 이다. 식 (1)에서 알 수 있듯이 공급 전압을 낮추면 전력 소모량이 줄어든다. 대부분의 프로세서들이 CMOS 회로로 구성되어 있기 때문에 공급 전압을 낮추어서 소비 전력을 줄일 수 있다는 것이 DVS의 기본 개념이다. 하지만 식 (2)에서와 같이 공급전압을 낮추면 회로의 지연시간이 길어진다. 결국 클럭 주파수는 지연시간에 반비례($f \propto 1/t_d$)하므로 공급전압을 낮추면 클럭 주파수도 느려져서 회로의 동작 속도가 느려진다. 따라서 실시간 시스템의 전력 소모량을 줄이기 위해 태스크의 실행 주파수를 낮추면 실행 시간이 늘어난다.[8] 이는 곧 태스크의 응답시간이 길어져서 시간적은 요구사항을 만족하지 못할 가능성이 증가하게 된다는 것을 의미한다.

$$P_{CMOS} = C_L V_{DD}^2 f \qquad \text{식. (1)}$$

$$t_d = k\frac{V_{DD}}{(V_{DD}-V_T)^2} \qquad \text{식. (2)}$$

주기적인 태스크에 대한 DVS 알고리즘은 태스크의 주기와 실행시간을 이용하여 시간적인 요구사항들을 충족시키는 범위 내에서 주파수를 낮추어 태스크를 스케줄링한다. 하지만 비주기적인 태스크는 주기가 없기 때문에 시간적인 요구사항을 만족시키면서 주파수를 조절하기가 어려운 단점을 가지고 있다. 지금까지 주기적인 태스크 모델에 대한 DVS 알고리즘들은 많이 연구되고 있지만 비주기 태스크를 고려한 효과적인 알고리즘이 없다.[11, 12, 13]

본 논문에서는 비주기 태스크에 대해 시간적인 요구사항을 만족하면서 실행 주파수를 동적으로 조절하는 DVS 알고리즘(DVS-AT)을 제안한다. 우선 비주기 태스크에 주기를 부여하여 주기적인 태스크로 변환시키는 가상태스크집합(VTS)을 정의하고, VTS를 통해서 주기적인 태스크집합으로 편입된 비주기 태스크의 주파수를 낮추어 전력소비를 줄이는 스케줄링 알고리즘을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 DVS 알고리즘에 대한 소개와 본 논문에서 사용할 태스크 모델에 대해서 언급 하고, 3장에서 본 논문에서 제한하는 DVS 알고리즘을 설명한 후, 제안한 알고리즘에 대한 시뮬레이션을 통해 알고리즘의 성능 입증을 4장에서 수행한다. 마지막으로 5장에서 제안한 알고리즘을 요약하고 향후 과제를 제시하고 끝맺는다.

## 2. 기존의 DVS Algorithms

### 2.1 주기적인 태스크

본 논문에서는 주기적인 태스크들의 집합을 T로 표시하며, 집합 T에 속하는 태스크는 주기($P_i$), 상대적인 마감시간($D_i$), 최악수행시간($C_i$) 등의 속성을 지니고 있다. 여기서 최악수행시간(WCET)은 최악의 경우 실행시간을 의미하며, 최대의 동작주파수로 계산한 값으로 한다. 또한 태스크의 주기와 상대적인 마감시간은 동일하다고 가정한다. 어떤 태스크가 마감시간 이전에 수행을 완료하면 *마감시간을 만족한다* 라고 하고, 마감시간이 지나서야 종료되면 *마감시간을 어겼다*라고 한다. 마감시간을 반드시 만족해야 하는 태스크를 *경성 실시간 태스크* 라고 하고, 경우에 따라서 어겨도 되는 태스크는 *연성 실시간 태스크*라고 한다. 본 논문에서의 주기 태스크들은 경성 실시간 태스크라고 가정한다. n개의 태스크로 구성된 태스크 집합 $T$는 $T = \{ T_1, T_2, \cdots, T_n \}$ 로 표현되며, 각각 태스크의 실제 수행시간(AET)은 Ai로 표시한다. 실제 수행시간은 최악수행시간을 초과하지 않는다. 각각의 태스크는 실행 중에 보다 높은 우선순위를 지닌 태스크에 의해 선점당할 수 있다.

### 2.2 비주기적인 태스크

본 논문에서는 비주기적인 태스크들의 집합을 $J$로 표시하며, 집합 $J$에 속하는 태스크는 평균 발생 간격(λ), 평균 실행 시간(μ) 등의 속성을 지니고 있다. 본 논문에서의 비주기 태스크들은 연성 실시간 태스크라고 가정한다. n

개의 태스크로 구성된 태스크 집합 $J$는 $J = \{J_1, J_2, \cdots, J_n\}$로 표현되며, 각각 태스크의 실제 수행시간은 주기적인 태스크와 마찬가지 $A_i$로 표시한다.

### 2.3 Processor Model

프로세서는 동적으로 공급 전압을 변경할 수 있으며, 동작 주파수는 공급 전압에 비례한다. 공급 전압은 미리 정해진 몇 개의 이산적인 값들로만 변경 될 수 있다. 또한 본 논문에서는 프로세서의 최대 동작 주파수를 1로 가정하고 나머지 주파수들은 최대 주파수에 대한 상대적인 값으로 표현한다. 공급 전압을 변경하는데 소요되는 시간은 극히 적기 때문에[22] 공급 전압 변경에 필요한 시간은 무시한다.

### 2.4 주기적인 태스크를 위한 동적 가변 전압

실시간 시스템에서 주기적인 태스크는 주기 $P_i$ 간격으로 발생하며, 마감시간 이전에 수행이 완료될 수 있도록 실시간 스케줄러에 의해 스케줄링 되어야 한다.[33] 이때 태스크 집합내의 모든 태스크가 마감시간을 지킬 수 있는지를 테스트하는 것을 *schedulability test* 라고 하고, 이 테스트를 통과하면 해당 태스크 집합은 *schedulable* 하다라고 한다. 대부분의 DVS 알고리즘들은 *Rate Monotonic*(RM)[41, 41]과 *Earliest-Deadline-First*(EDF)[42, 43] 스케줄러에 기반하고 있다. RM은 정적인 우선순위 스케줄러로서, 주기가 짧은 태스크에 높은 우선순위를 책정하여 짧은 주기의 태스크를 항상 먼저 수행시킨다. EDF는 동적인 우선순위 스케줄러로서, 마감시간이 가까운 태스크에 높은 우선순위를 동적으로 할당한다.

EDF 스케줄링 알고리즘에서 *schedulability test*를 통과하기 위한 필요충분 조건은 식 (3)과 같이 최악 실행 시간에 대한 태스크 활용률의 전체 합이 1보다 작으면 된다.[18] 여기서 태스크 활용률은 실행시간을 해당 태스크의 주기로 나눈 값을 말하며 해당 태스크가 전체 시스템에서 차지하는 시간적인 비율을 의미한다.

$$U = \sum_{i=1}^{n} (C_i / T_i) \le 1 \qquad \text{식. (3)}$$

실시간 태스크의 소비 전력을 줄이기 위해, 동작 주파수를 $\alpha(0 < \alpha \le 1)$ 배 만큼 낮추어서 태스크를 실행시키면 $1/\alpha$배 만큼 수행시간이 늘어나기 때문에 마감시간을 지키지 못할 경우가 생기게 된다. 동작 주파수를 $\alpha$배 만큼 낮추었을 때는 U가 $\alpha$보다 작아야 schedulable 하다고 할 수 있다. 즉, $C_1 / P_1 + C_2 / P_2 + \cdots + C_n / P_n \le \alpha$를 만족해야 한다. 따라서 동작 주파수는 *schedulability test*를 만족하는 주파수들 중에서 제일 낮은 것을 선택하여 최대한 전력을 줄이게 된다.

실시간 시스템에서 태스크들의 실제 수행시간(AET)은 최악수행시간보다 대부분 짧다. 따라서 이러한 경우, 짧게 끝난 태스크에 의해 $U$의 값이 작아진다. 특정 태스크의 수행이 완료된 후, 그 다음 태스크에 대해서는 변경된 $U$ 값에 따라 동작 주파수를 더욱 낮출 수 있게 된다.

**표 1.** 임의의 태스크 예

| Task | Period | WCET | AET | | |
|---|---|---|---|---|---|
| | | | 1 | 2 | 3 |
| 1 | 8ms | 3ms | 2ms | 1ms | 2ms |
| 2 | 12ms | 2ms | 1ms | 2ms | 2ms |

그림 (1)은 표 (1)과 같은 태스크 집합을 EDF에 기반한 DVS 알고리즘에 의해 스케줄링 한 결과를 나타낸 것이다. 프로세서가 사용 가능한 주파수는 0.5, 0.75, 1.0 이라고 가정한다. 표 (1)에 표시한 시간들은 최대의 주파수로 프로세서가 동작할 경우의 값 들이다. 실제 수행시간은 해당 태스크가 수행이 완료되어야만 알 수 있기 때문에 태스크의 수행이 완료된 후 태스크 활용률을 다시 계산하고 그에 따라 주파수도 적절히 재 조정됨을 볼 수 있다.

### 2.5 비주기 태스크의 연구

비주기 태스크는 발생시간이 정해져 있지 않고(즉, 주기가 없음), 마감시간도 지정되지 않는 경우가 많다. 따라서 비주기 태스크에 대한 실시간 스케줄링은 태스크의 응답시간을 줄이는 것이 중요하다. 가상의 마감시간을 설정하여 비주기 태스크의 응답시간을 조절하는 Total

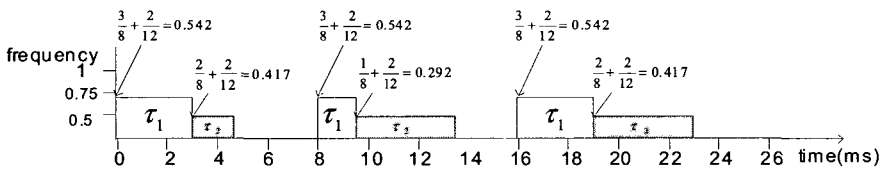



**그림 1.** EDF 알고리즘과 DVS 알고리즘을 사용한 예

Bandwidth Server(TBS) 알고리즘[4]등 비주기 태스크를 실시간 스케줄링하는 여러 방식들이 제안되었다. 이러한 비주기 태스크에 대해 동작 주파수를 조절하려면 태스크 활용률을 계산해야 하는데, 비주기 태스크는 주기가 없기 때문에 정확한 태스크 활용률을 얻기가 용이하지 않다. 따라서 비주기 태스크를 포함한 태스크 집합에 대한 DVS 알고리즘을 적용하기가 상당히 어렵다.

## 3. 제안하는 동적 가변 전압 스케줄링

### 3.1 Virtual Task Set(VTS)

태스크의 주파수를 낮추었을 경우, 마감시간을 어기는 태스크가 존재하는 지를 확인하려면 *schedulability test* 를 검사해 보아야 하는데, 이를 위해서는 반드시 태스크의 주기와 최악실행시간을 미리 알아야 한다. 하지만 비주기 태스크는 WCET는 있지만 주기가 존재하지 않는다. 본 논문에서는 비주기 태스크에 대해서도 DVS알고리즘을 적용할 수 있게 하기 위해, 다음과 같이 정의된 가상의 주기와 WCET를 지닌 가상 태스크 집합(VTS)을 생성한다.

가상 태스크 집합의 정의

$VTS = \{ T_i \mid P_i < P_{min},\ U < 1\}$

여기서 $P_{min}$ 은 주기적인 태스크들의 주기 중에서 가장 짧은 주기를 의미한다. 그런 다음, 비주기 태스크를 VTS에 할당하여 주기 태스크로 편입시킨다.

우선, 비주기 태스크의 응답시간을 고려하여 주기적인 태스크들의 주기 중 가장 짧은 것보다 작게 가상 태스크의 주기를 설정한다. 설정된 값이 작으면 작을수록 비주기 태스크의 응답시간이 짧아지지만 그만큼 문맥교환(context switch) 회수도 증가하여 시스템의 오버헤드가 커진다. 따라서 비주기 태스크의 응답시간 요구사항을 만족하는 범위 내에서 가장 큰 값을 가상의 주기로 선택한다.

일단 주기가 결정되면 *schedulability test*를 통해 schedulable 한 시스템이 되도록 적절히 가상 태스크의 WCET을 설정한다. 즉, 식 (4)와 같은 수정된 *schedulability test*를 만족하는 최대의 실행시간을 구한다.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_a}{T_a} \le 1 \qquad \text{식. (4)}$$

여기서 $T_a$는 가상 태스크의 주기를 의미하며, $C_a$는 가상 태스크의 WCET이다. 예를 들어, 표 (1)과 같은 주기적인 태스크 집합의 경우, $T_a$는 8ms보다 작은 값 중에서 비주기 태스크의 응답시간을 고려하여 7ms로 선택하였을 경우, $C_a$는 3ms가 된다. (실제로는 3.2ms 가 되지만 편의상 3ms 로 설정하였음)

이렇게 가상의 주기와 WCET를 지닌 가상 태스크를 구비해 놓은 후, 실제 비주기 태스크가 발생하면, 해당 태스크를 가상태스크에 할당하여 스케줄링 한다. 이때 발생한 비주기 태스크의 WCET 만큼 해당 가상 태스크의 WCET을 조정한다. 즉, 실제 발생한 비주기 태스크의 WCET이 가상 태스크의 WCET보다 작으면, 가상 태스크의 WCET을 비주기 태스크의 WCET로 줄인다. 만약 더 크다면 가상 태스크의 WCET 만큼만 할당하고 나머지는 그 다음 주기로 이월한다. 따라서 가상 태스크의 WCET을 시간 용량(capacity time)이라고 하겠다.

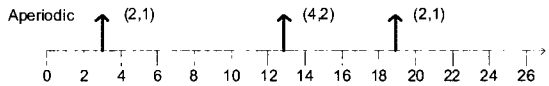



**그림 2.** 비주기 적으로 발생하는 태스크들

그림 (3)은 그림 (2)처럼 비주기 태스크가 발생할 경우, 제안한 VTS를 이용하여 발생한 비주기 태스크가 실시간으로 스케줄링 되는 모습을 나타낸 것이다. 이때 가상태스크의 주기와 시간 용량은 각각 7ms 와 3ms로 설정하겠다. 그림 (2)에서 화살표는 비주기 태스크가 발생하는 시점을 의미하며 괄호 안의 숫자는 각각 발생한 태스크의 WCET와 실제수행시간을 의미한다. 그림 (3)의 점선 영역은 가상 태스크의 시간 용량을 의미하며, 채워진 영역은 실제 설정된 WCET을 의미한다. 두번째 발생한 비주기 태스크는 WCET이 가상 태스크의 시간 용량을 초과하여 다음 주기로 이월 됨을 볼 수 있다.

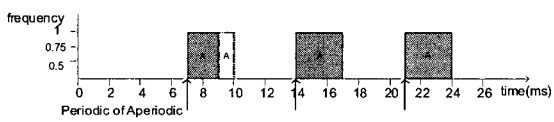



**그림 3.** 비주기 태스크의 실시간 스케줄링

그림 (4)는 표 1의 태스크 집합에 그림 (2)의 비주기 태스크를 가정하였을 경우 태스크들이 실시간 스케줄링 되는 모습을 나타낸 것이다. 이때 가상태스크의 주기와 시간 용량은 그림 (3)과 마찬가지로 각각 7ms 와 3ms로 설정한다.

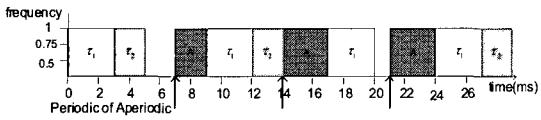



**그림 4.** 주기태스크와 비주기 태스크의 스케줄링

### 3.2 동적 가변 전압 알고리즘과 VTS

앞 절에서 정의한 VTS를 이용하면 비주기 태스크도 주기와 WCET를 가지게 됨으로 DVS 알고리즘을 적용할 수 있다. 구체적인 알고리즘을 살펴보면 다음과 같다. 우선, 가상 태스크의 $C_a$를 0으로 초기화한다. 그런 다음, 식 (5)의 *schedulability test*를 만족하는 가장 작은 α값을 구한 후, 이 α값보다 큰 가장 작은 주파수를 선택하여 첫 번째 태스크를 수행시킨다. 첫 번째 태스크가 종료되면 식 (5)를 이용하여 α를 재계산하여 주파수를 변경하는 작업을 이후 태스크에 대해서 반복 수행한다.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_a}{T_a} \le \alpha \quad \text{식. (5)}$$

그러다가 비주기 태스크가 발생하면 가상 태스크의 시간 용량까지 $C_a$을 할당한 후, α값과 주파수를 설정하는 과정을 반복한다. 제안한 알고리즘을 정리하면 그림 (5)와 같다. 비주기적인 태스크가 VTS에 의해 주기 태스크로 편입되었기 때문에 DVS알고리즘은 기존의 주기적인 태스크와 유사하다. 단지 가상 태스크의 WCET이 비주기 태스크의 발생 상황에 따라 변화한다는 것이 차이점이다.

```
while(true) {
    if T_i is released
        if T_i is periodic task
            set U_i = C_i / P_i
        else if T_i is virtual task from aperiodic task
            set U_i = C_a / P_i
            // C_a is actually allocated time of T_i
        set _ frequency(i)
    else if T_i completes
        set U_i = A_i / P_i
        // A_i is the actual execution time of T_i
}
set _ frequency(i)
    use lowest frequency f_i ∈ {f_1, L , f_n | f_1 < L < f_n}
    such that Σ_{i=0}^{m} U_i ≤ f_i / f_n
```

**그림 5.** 제안하는 DVS 스케줄링 알고리즘

그림 (6)는 그림 (4)의 경우에 대해 제안한 DVS 알고리즘에 따라 태스크들이 스케줄링 되는 모습을 나타낸 것이다. 그림에서 알 수 있듯이 태스크가 종료되거나 시작될 때 마다 태스크 활용률을 매번 다시 계산한다. 그림 안의 수치는 해당 지점에서의 전체 태스크 활용률의 합을 나타낸 것이다.

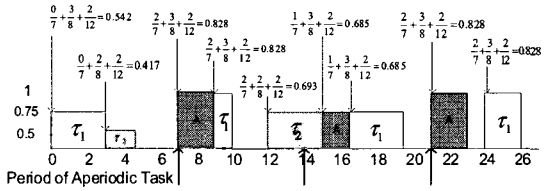



**그림 6.** 제안하는 DVS 알고리즘을 적용한 스케줄링

## 4. 실험 결과

제안된 DVS 알고리즘이 기존의 주기적인 태스크에 대한 알고리즘에 비해 얼마 만큼의 에너지를 소모하는지에 대해 여러 가지 실험을 수행하였다. 시뮬레이션 프로그램은 C/C++ 언어를 사용하여 개발하였으며, 시뮬레이션 시 프로세서는software-controlled halt 기능을 가지고 있어서 유휴시간(idle time)상의 에너지 소모는 없는 것으로 가정하였다. 또한, 시뮬레이션 동안 비주기 태스크는 그림 (7)과 같이 난수적으로 발생을 시켰다. 즉, 발생 시점과 WCET 둘 다 예측 불가능한 상황을 가정하였다.

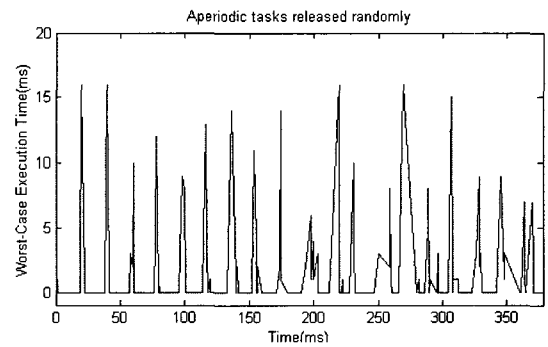



**그림 7.** 비주기 태스크의 생성

그림 9는 표준 PC 메인보드[1]에 10개의 실시간 태스크를 실행시켰을 경우의 에너지 소모 정도를 비교한 것이다. 이때 실제수행시간 대 최악수행시간 비율은 0.35로 가정하였으며, 이후 특별한 언급이 없을 경우 이 값을 계속 유지하겠다. 표준 PC 메인보드의 사용가능 주파수와

그 때의 공급전압은 표 (2)와 같다. 모든 주파수들은 최고 주파수에 대한 상대적인 값으로 표시하였다. DVS를 적용하지 않았을 경우의 에너지 소모량을 1로 설정하였으며, 실제 실행시간을 미리 정확히 예측한 상태에서 DVS 스케줄링 한 경우의 에너지 소모량을 계산하여 이론적인 에너지 소모 하한선으로 표시하였다. 제안한 알고리즘과 비주기 태스크에 대해 DVS를 적용하지 못하는 기존 알고리즘을 비교해 보았다. 그림 6에서 보듯이 제안한 방식이 기존 DVS 보다 에너지 소모가 적으며, 특히 태스크 활용률이 50% 부근일 때 가장 효율이 좋음을 알 수 있다.

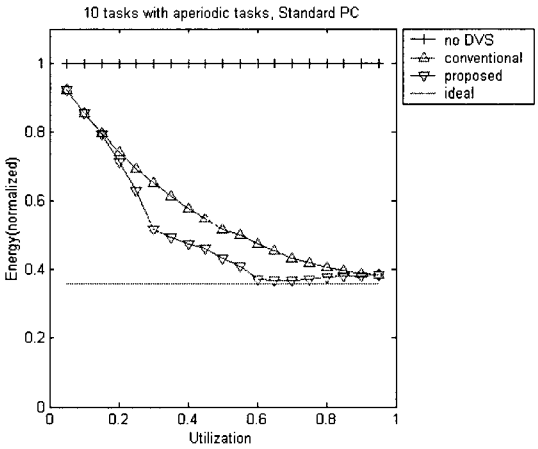



**그림 8.** AMD-K6에서의 에너지 소모량 비교

**표 2.** 표준 PC 메인보드의 사용가능 주파수/공급전압

| 주파수 | 0.5 | 0.75 | 1.0 |
|---|---|---|---|
| 공급전압 | 3V | 4V | 5V |

또한, 태스크의 개수, MPU의 종류, 최악수행시간 대 실제수행시간 비율 등이 에너지 소비량에 어떤 영향을 미치는지를 살펴보기 위해 여러 가지 실험을 수행하였다. 실험 결과는 다음과 같다.

### 4.1 태스크의 개수에 따른 전력 소모량 비교

우선, 태스크의 개수를 달리 하였을 경우 어떤 현상이 발생하는지 살펴보았다. 태스크의 개수가 5개일 때와 15개일 때, 에너지 소모가 어떤 변화를 일으키는지 살펴보면, 그림 7과 같이 별다른 차이가 존재하지 않는다. 따라서, 태스크의 개수의 차이는 에너지 소비에 별다른 영향을 미치지 않으므로 이후 모든 시뮬레이션에서는 태스크의 개수를 10개로 고정시키겠다.

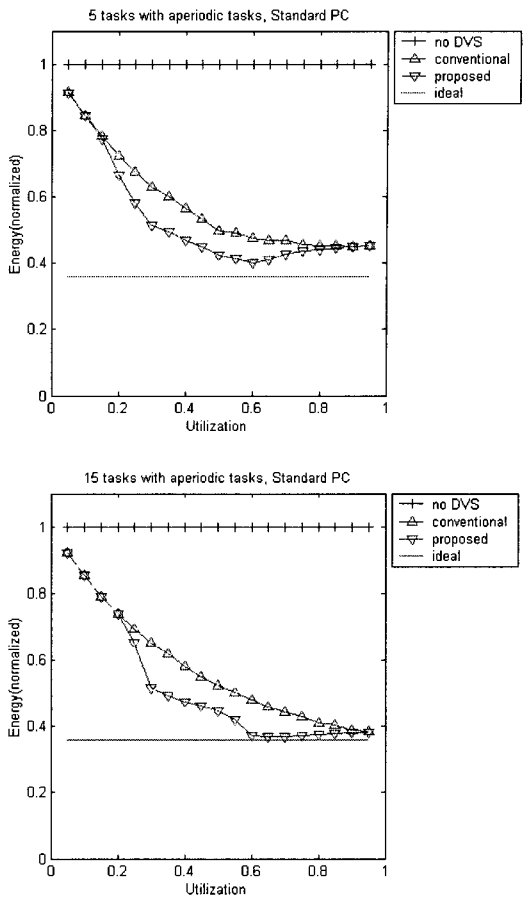
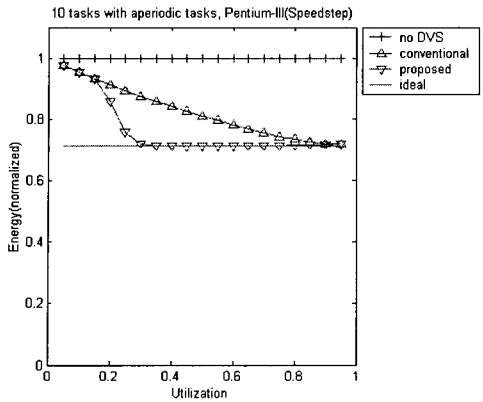
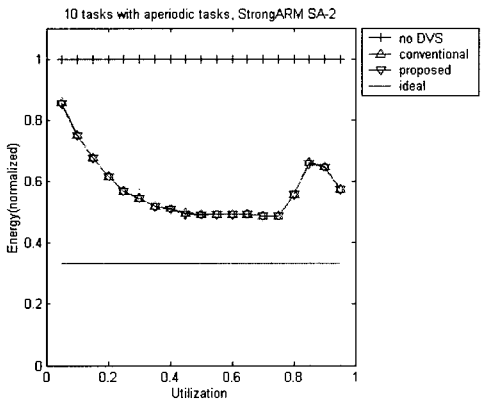



**그림 9.** 5, 15개 태스크의 에너지 소모량 비교

### 4.2 여러 MPU에 대한 전력 소모량 비교

이번에는 MPU에 따라 에너지 소모량이 어떤 분포를 보이는지 살펴보았다. Speedstep 기능을 지닌 Pentium-III, 표준 PC의 메인보드, PowerNow! 기능을 지닌 AMD-K6 프로세서를 사용하여 시뮬레이션 하였다. 표3, 4에 각각의 MPU의 사용 가능 주파수와 공급전압은 표 3, 4와 같다. Pentium-III와 AMD K-6 프로세서는 앞서 살펴본 바와 비슷하게 제안한 알고리즘이 기존의 방식보다 에너지 소모량이 적으며, 태스크 활용률이 50% 부근에서 가장 효율이 좋음을 할 수 있다. 하지만, StrongARM 은 다른 실험과 달리 제안한 알고리즘이 별다른 이점을 보이지 않는다. 이는 StrongARM의 주파수 차이가 너무 커 대부분의 경우 최대 주파수로 스케줄링 되기 때문이다.



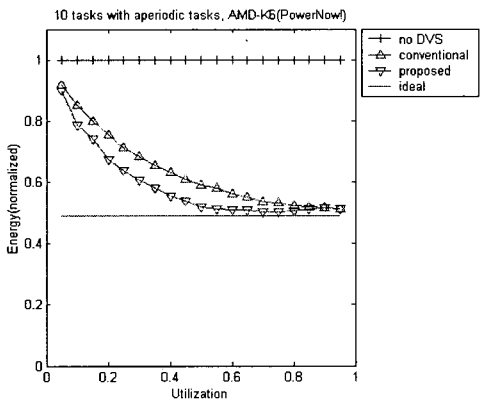




**그림 10.** 여러 가지 MPU에 대한 에너지 소모량 비교

### 4.3 AET 대 WCET 비율에 대한 전력 소모량 비교

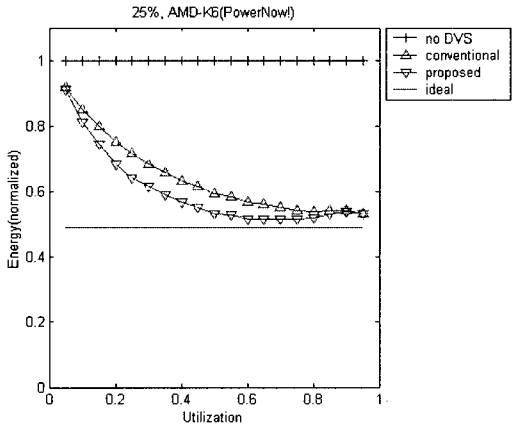
실제수행시간 대 최악수행시간의 비율이 다를 경우 에너지 소모량이 어떤 영향을 받는지 살펴보았다. 각각 25%와 50%의 비율에 대해 시뮬레이션을 수행하였다. 그림 11에서 보듯이 비율이 작을수록 에너지 절감 효과는 많이 나타남을 알 수 있다. 이는 실제 수행시간이 작으면, 그만큼 다음 태스크의 주파수를 낮출 수 있는 여지가 커지기 때문이다.

**표 3.** Pentium-III, StrongARM SA-2의 주파수/공급전압

| 구분 | Pentium-III | | StrongARM SA-2 | |
|---|---|---|---|---|
| 주파수 | 0.77 | 1.0 | 0.25 | 1.0 |
| 전압 | 1.35V | 1.6V | 0.75V | 1.3V |

**표 4.** AMD-K6 프로세서(PowerNow!)의 주파수/공급전압

| 주파수 | 0.36 | 0.55 | 0.64 | 0.73 |
|---|---|---|---|---|
| 전압 | 1.4V | 1.5V | 1.6V | 1.7V |
| 주파수 | 0.82 | 0.91 | 1.0 | |
| 전압 | 1.8V | 1.9V | 2.0V | |



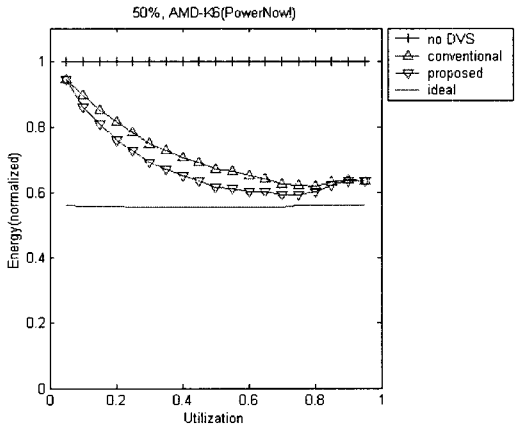




**그림 11.** 25%와 50%의 실제 수행시간 대 최악수행시간 비율에 대한 에너지 소모량 비교

## 5. 결론

본 논문에서는 가상태스크(VTS)를 이용하여 비주기 태스크에 대해서도 보다 효율적으로 주파수를 낮추어 전력소비를 절감하는 새로운 DVS 알고리즘을 제안하였다. 가상의 태스크집합인 VTS를 정의하여 난수적으로 발생하는 비주기 태스크를 주기 태스크로 변환함으로써 비주기 태스크에 대해서는 DVS 알고리즘을 적용할 수 없었던 문제점을 해결하였다. 기존의 DVS 알고리즘에 비해서 에너지 소비가 11% 정도 절감되는 효과를 보임을 시뮬레이션을 통해 입증하였다.

그러나 비주기 태스크를 고려할 때, 발생한 주기에서 바로 실행을 하지 않기 때문에 비주기 태스크를 고려한 동적 전압 가변 스케줄링에서 비주기 태스크들의 평균 반응시간을 더 향상시키면서 전압 소모를 감소시킬 수 있는 연구가 필요하다.

## 참고문헌


[01] Moon Joo Park, "Feasibility Analysis of Scheduling Periodic and Aperiodic Real-Time Tasks," master dissertation, Seoul National University, 2002.

[02] B. Sprunt, L. Sha and J. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems.," The Journal of Real-Time Systems, No.1, pp.27-60, 1989

[03] R I. Davis, K.W. Tindell and A. Burns, "Scheduling slack time in fixed priority preemptive systems," in Proceedings of the Real-Time Systems Symposium, pp.160-171, Dec. 1993

[04] Jae Hong Shim, "An Expanded Real-Time Scheduler Model for Supporting Aperiodic Task Servers," Korea Information Processing Society, 2001. 3.

[05] Kang G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operation System," SOSP, 2001.

[06] Chan Ho Pee, "A Dynamic Voltage Scaling for Shared Resource," The Institute of Electronics Engineers of Korea, 2003. 6.

[07] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a hard Real-Time Environment," Journal of the ACM, Vol.20, No.1, pp.40-61, 1973.

[08] HOUSSINE CHETTO, MARYLINE CHETTO, "Some Results of the Earliest Deadline Scheduling Algorithm," IEEE, 1989.

[09] LUI SHA, RAGUNATHAN RAJKUMAR, JOHN P. LEHOCZKY, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE, 1990.

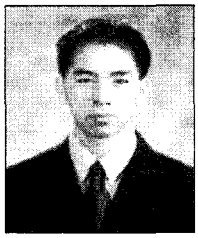
[10] Ala' Qadi, Steve Goddard, Shane Farritor, "A Dynamic Voltage Scaling Algorithm for Sporadic Tasks," IEEE Real-Time Systems Symposium, December 2003.

[11] N. Audslcy, A. Burns, M. Richardso, and A. Welligs, "Hard real-time scheduling: The deadline-monotonic approach,' in Proc.IEEE Workshop on Real-Time Operating Systems and Software, May 1991, pp.133-137.

[12] Duk Joo Lim, "Soft Aperiodic Real-Time Task Scheduling Algorithm Supporting Maxinum Slack Time," The Institute of Electronics Engineers of Korea, 1999년.



**권 기 덕(Ki-Duk Kwon)** [준회원]



- 2005 년 8월 : 한양대학교 미디어통신공학과 (공학석사)
- 2006년 4월 ~ 현재 : Waseda University 정보네트워크공학과 (박사과정 재학중)

<관심분야>
L4 micro-kernel, Device Driver, CPU Scheduling

**정 준 모(Jun-Mo Jung)** [종신회원]

- 1987년 2월 : 한양대학교 전자공학과 (공학석사)
- 2004년 2월 : 한양대학교 전자공학과 (공학박사)
- 1989년 2월 ~ 1996년 3월 : 삼성전자 ASIC 센터
- 1996년 4월 ~ 2004년 2월 : 김포대학 전자정보계열 교수
- 2004년 3월 ~ 2005년 3월 : 한양사이버대학교 컴퓨터공학과 조교수
- 2005년 4월 ~ 현재 : 군산대학교 전자정보공학부 조교수

<관심분야>
VLSI Design, SoC Design, SoC Test & Verification, Test Scheduling

**권 상 홍(Sang-Hong Kwon ) [정회원]**

- 1998년 2월 : 경북대학교 전산공학과 (공학석사)
- 1997년 1월 ~ 현재 : 한국폴리텍 구미대학 컴퓨터정보과 (교수)

<관심분야>
User-Level Network, 소프트웨어 신뢰성