# 동적 전력 관리 기법의 실시간 태스크 스케줄 가능 검사 영향 분석

유시환, 유혁
고려대학교 컴퓨터학과
{shyoo, hxy}@os.korea.ac.kr

# Impact of Dynamic Voltage Scaling on Real-time Schedulability Analysis

Seehwan Yoo, Chuck Yoo
Korea University

**요 약**

동적 전력 관리 기법은 임베디드 시스템과 같은 저전력성이 요구되는 시스템에서 널리 활용되고 있다. 동적 전력 관리 기법은 처리율과 소비전력 간의 상관 관계를 통해, 프로세서의 전압과 주기를 조절하여 소비 전력당 처리율을 높이는 기법이다. 이러한 동적 전압 관리 기법이 실시간 특성이 필요한 임베디드 시스템에 적용되는 경우, 실시간 스케줄러에 큰 영향을 끼치게 된다. 실시간 스케줄러에서는 주어진 임계 시간 이내에 작업의 수행을 마치기 위하여, 스케줄 가능성 테스트를 수행하여 적합한 작업들만을 실행하도록 한다. 하지만, 인터럽트 처리 등으로 인한 선점 가능성은 스케줄 가능성에 대한 분석을 복잡하게 만들고 있다. 본 논문에서는 인터럽트 처리를 고려한 실시간 스케줄링 분석 연구를 기반으로 하여, 동적 전력 관리가 추가된 경우의 영향을 분석하도록 한다. 동적 전력 관리로 인한 실시간 처리 요구 사항의 증가와 실제 적용 가능한 사례를 보인다.

## 1. 서 론

최근의 임베디드 장치에서는 높은 동작 클럭을 가진 프로세서들을 사용하고 있다. 최근의 고사양 임베디드 프로세서의 소비 전력 문제는 큰 문제로 지적되고 있으며 이를 해결하기 위한 다양한 접근 방식이 연구되고 있다.

동적 전력 관리 기법은 처리율과 소비전력 간의 상관 관계를 통해 잔여전력이 부족한 경우, 처리율 저하를 최소화 하면서 소비 전력을 낮추는 기법이다[1-6]. 프로세서의 전압과 동작 클럭의 주파수는 소비전력과 직접적으로 관련이 있다. 즉, 사용되는 소비전력은 수식 1에서 제시하는 바와 같이 동작 주파수와 전압의 제곱, 커패시턴스와 비례한다.

$$P \propto c \cdot V^2 \cdot f$$

**수식 1 소비전력과 전압, 동작 주파수의 관계식**

이 때, 소비전력량은 시간과 소비전력의 곱이며, 실행 시간은 동작 주파수에 반비례하므로, 소비에너지인 소비전력량은 커패시턴스와 전압의 제곱에 비례한다. 실제적인 동적전력 관리로 인한 성능상의 소득은 동작 주파수를 낮춤으로써 얻을 수 있는 전압 강하에서 발생한다. 일반적으로 에너지와 동작 주파수 사이에는 다음 수식 2와 같은 관계식이 성립한다.

$$E \propto \left( V_{th} + \frac{f}{2k} + \sqrt{\frac{V_{th} f}{k} + \left(\frac{f}{2k}\right)^2} \right)^2$$

**수식 2 에너지(소비전력량)과 동작 주파수 사이의 관계식**

소비전력량에 따른 처리율의 변화는 동작 주파수에 따라 반비례하므로, 동적 전력 관리 기법을 통해 단위 소비전력당 처리율을 유지하면서 소비 전력 절감 효과를 얻을 수 있다.

이러한 동적 전력 관리 기법은 실시간 시스템에서 곧바로 적용되기 힘들다. 동적 전력 관리 기법은 소비전력당 처리율은 높여주지만, 실시간 시스템에서는 임계 시간을 맞추어야 하는 요구사항이 존재하기 때문이다. 즉, 임계 시간 이내에 주어진 작업을 모두 처리하기 위해서는 처리율이 너무 낮은 수준으로 떨어져서는 안되기 때문이다. 하지만 실시간 시스템은 특성 상, 처리량보다는 임계 시간 내의 작업 완료가 더욱 중요하다. 단순히 처리량의 임계치를 올리는 것보다 작업 완료 시점에 적절히 스케줄링이 가능한지를 분석하는 작업이 중요하다고 할 수 있다. 본 논문에서는 인터럽트와 같이 선점 처리가 필요한 실시간 스케줄러의 분석 연구를 토대로 하여, 동적 전력 관리가 실시간 태스크 스케줄링에 미치는 영향을 분석한다.

## 2. 관련 연구

실시간 시스템은 모든 실시간 태스크들을 임계시간 안에 완료할 수 있도록 스케줄링 한다. 이를 구현하는 태스크 스케줄링 정책은 크게 고정 우선 순위 기반의 Rate-Monotonic 방식과 동적 우선 순위 기반의 EDF(Earliest Deadline First) 방식이 있다. EDF 방식의 스케줄러는 프로세서 유휴 사용량을 떨어뜨리지 않으면서도 임계 시간 요구사항을 만족할 수 있는 최적의 스케줄러임이 밝혀져 있다.

실시간 시스템의 태스크 스케줄링은 태스크의 스케줄 가능성 검사에서부터 시작된다. 각각의 실시간 태스크들은 수행 시간과 우선 순위, 임계 처리 시간과 동작 주기를 가진다. 태스크 스케줄 가능성의 검사를 위한 분석 방법은 많은 논문을 통해 밝혀져 있다. 본 논문은 그 중에서 네트워크 장치 드라이버의 효과를 고려한 실시간 스케줄 가능성 검사에 대한 연구[7]를 기반으로 진행되었다.

네트워크 장치들은 일반적으로 매우 많은 인터럽트를 매우 빠른 시간 안에 처리한다. 따라서, 이를 위한 인터럽트 처리기의 구조와 실행 지연 등의 기법이 연구되었다. 리눅스와 같은 운영체제에서는 빠른 인터럽트 처리를 위해 인터럽트 처리를 하드웨어 인터럽트 요청 처리와 소프트 인터럽트 처리로 분리하였다. 하드웨어 인터럽트 요청 처리는 기존의 운영체제가 가지고 있는 인터럽트 처리 루틴이다. 하지만, 간단한 장치 의존적인 역할만을 처리하고, 복잡한 나머지 처리를 운영체제의 스케줄링 정책에 따라 처리할 수 있도록 소프트웨어 인터럽트 형태로 보관해 둔다. 커널 스케줄러는 적절한 시점에 소프트 인터럽트들을 꺼내어 처리하도록 한다. 이러한 기법은 인터럽트 처리가 길어짐에 따라 발생하는 인터럽트 소실을 막기 위한 기법이다. 인터럽트 처리 루틴 내에서는 동일한 인터럽트가 마스킹되어 발생하지 않기 때문에, 인터럽트 처리 루틴을 최대한 짧게 구성하는 것이 중요하다.

최근의 논문에서는 이러한 네트워크 장치가 실시간 스케줄러의 스케줄 가능성 분석에 어떠한 영향을 끼치는지 밝혀져 있다. 작업 J의 주어진 시간 구간 [a,b)에서의 실제 실행 시간을 $demand_J(a,b)$ 라고 하자. $r_k$를 작업 k의 릴리스 시간, $d_k$를 임계시간, $e_k$를 수행시간이라고 할 때, 실시간 태스크 $J_k$의 수행을 위해 필요한 프로세서 요구조건은 다음 수식3과 같이 정의할 수 있다.

$$e_k + \sum_{\{i<k\}} \text{demand}_{J_i}(r_k, r_k + d_k) \le d_k$$

**수식 3 최소 수행시간 요구조건**

즉, k보다 우선순위가 높은 작업들이 실행을 모두 마치고 자신의 실행도 임계시간 이내에 마쳐야 실행이 가능하다. 이 때, 실행하는 작업이 정확히 주기적이지 않고, 태스크들의 릴리스 시간 간의 최소 간격이 주어지는 경우를 산발성(sporadic) 태스크라고 한다. 이 경우, 시간 간격 Δ 동안 발생하는 모든 작업들 $\tau_i$ 를 처리하기 위해 필요한 프로세서 요구량을 수식4와 같이 정의할 수 있다.

$$\text{demand}_{\tau_i}^{\max}(\Delta) \stackrel{\text{def}}{=} \max_{s \in S, t>0} \left( \sum_{\{J \in S\}} \text{demand}_J(t-\Delta, t) \right)$$

**수식 4 산발형 태스크의 최대 수행 요구 시간**

이 때, 모든 작업 $\tau_k$의 수행을 임계시간 $d_k$ 이내에 완료하기 위해서는 다음과 같은 요구 조건이 필요하다.

$$e_k + \sum_{i=1}^{k-1} \text{demand}_{\tau_i}^{\max}(d_k) \le d_k$$

**수식 5 산발형 태스크의 수행시간 완료 조건**

또한, 고정 우선순위 환경에서 인터벌 Δ 구간 내의 최대 프로세서 요구량은 작업의 수행 시간과 최대 실행 가능 회수에 의해 제한되므로, 다음 관계식을 얻는다.

$$\text{demand}_{\tau_i}^{\max}(\Delta) \le \left\lceil \frac{\Delta}{p_i} \right\rceil e_i$$

**수식 6 고정 우선 순위에서의 최대 수행 요구 시간의 한계값**

따라서, 수식 5와 6의 관계를 통해 다음 수식 7의 널리 알려진 고정 우선순위 환경의 스케줄 가능성 조건을 이끌어낼 수 있다. 다음 조건이 만족되면, 항상 작업 $\tau_i$는 임계 시간 이내에 완료되도록 스케줄링 할 수 있다.

$$e_k + \sum_{i=1}^{k-1} \left\lceil \frac{d_k}{p_i} \right\rceil e_i \le d_k$$

**수식 7 주기,산발형 실시간 작업의 스케줄 가능 조건**

이 때, 선점 가능한 높은 우선 순위의 작업을 고려해 보면, 실제 Δ 구간 내에서 실행 가능한 횟수는 수식6에서 주어진 최대 수행 요구 시간의 한계값보다 실제로 큼을 알 수 있다. 즉, 시간 인터벌 구간 내의 마지막 작업의 실행 시간이 해당 수행 시간에 맞도록 수식 6을 수정한 수정 프로세서 요구량 한계값(refined demand bound)을 다음과 같이 다시 쓸 수 있다.

$$\text{demand}_{\tau_i}^{\max}(\Delta) \le je_i + \min(e_i, \Delta - jp_i)$$

**수식 8 수정 프로세서 요구량 한계값**

이 때, j는 $j \stackrel{\text{def}}{=} \left\lfloor \frac{\Delta}{p_i} \right\rfloor$와 같이 정의할 수 있다.

$\Delta = jp_i + e_i$인 각 지점에서 수식8의 한계량은 다음과 같이 표현할 수 있다.

$$\left(\frac{\Delta - e_i}{p_i} + 1\right) e_i = u_i(\Delta + p_i - e_i)$$

**수식 9 최대 수행 요구량의 수렴 값**

따라서, 수식 8은 다음과 같이 고쳐 쓸 수 있다.

$$\mathrm{demand}_{\tau_i}^{\max}(\Delta) \le \min(\Delta,\ u_i(\Delta + p_i - e_i))$$

**수식 10 최대 수행 요구량 관계식2**

위의 정의는 앞서 정의한 수식5의 수행시간 완료 조건식을 따라 다음과 같은 수행 요구비율(load) 정의(수식 11)에 의해 수식 12와 같이 바꿀 수 있다.

$$\mathrm{load}_{\tau_i}(t-\Delta, t) \stackrel{\text{def}}{=} \frac{\mathrm{demand}_{\tau_i}(t-\Delta, t)}{\Delta},$$

$$\mathrm{load}_{\tau_i}^{\max}(\Delta) \stackrel{\text{def}}{=} \frac{\mathrm{demand}_{\tau_i}^{\max}(\Delta)}{\Delta}.$$

**수식 11 수행 요구 비율 정의**

$$\frac{e_k}{d_k} + \sum_{i=1}^{k-1} \mathrm{load}_{\tau_i}^{\max}(d_k) \le 1$$

**수식 12 수행 요구 정의에 따른 수행 시간 완료 조건식**

즉, 임의의 시간 인터벌 $d_k$ 구간에서 우선 순위가 높은 모든 작업의 프로세서 활용율을 더한 값과 해당 작업의 프로세서 활용율을 더한 값이 1을 넘지 않는 경우에, 항상 실시간 임계 시간 이내에 스케줄 가능함을 보이고 있다.

수식 11의 수행 요구 비율 정의에 따라 수식8의 수정 최대 허용 가능 프로세서 요구량 한계값을 다시 쓰면 다음 수식13을 얻게 된다.

$$\mathrm{load}_{\tau_i}^{\max}(\Delta) \le \frac{je_i + \min(e_i,\ \Delta - jp_i)}{\Delta}$$

**수식 13 최대 프로세서 수행 요구 비율**

이 때, j는 수식6에서와 같이 $j \stackrel{\text{def}}{=} \left\lfloor \frac{\Delta}{p_i} \right\rfloor$ 로 정의할 수 있다. 그리고, 수식10의 최대 수행 요구량 관계식2를 수행 요구 비율에 따라 고쳐쓰기 위해 양 변을 $\Delta$로 나누면, 다음 수식14와 같은 하이퍼볼릭 수행 요구 비율 임계값(hyperbolic load bound)을 얻는다.

$$\mathrm{load}_{\tau_i}^{\max}(\Delta) \le \min\left(1, u_i\left(1 + \frac{p_i - e_i}{\Delta}\right)\right)$$

**수식 14 하이퍼볼릭 수행 요구비율 임계값**

이렇게 얻은 하이퍼볼릭 수행 요구 비율에 따라 리눅스 스케줄러를 수정하여 실시간 요구사항에 매우 근접한 (예측 가능한) 성능을 얻을 수 있음을 보였다.

## 3. 동적 전력 관리와 수행 시간의 상관 관계

동적 전력 관리는 크게 두 가지 기법으로 구성된다. 프로세서의 동작 주파수를 변경하는 기법은 프로세서가 사용하는 클럭을 조절하여, 낮은 동작 속도를 가져오게 된다. 이 때, 실행 시간이 직접적으로 영향을 받게 된다. 즉, 프로세서의 클럭 주파수와 실행 시간은 정비례 관계에 있다. 대개 클럭 주파수는 클럭 사이클을 가지고 조절하므로, 미리 정의된 몇 개의 클럭 주파수로 변경이 가능하다.

클럭 주파수만을 조절하는 경우에는 소비 에너지에 변함이 없다. 주파수를 감소하면, 소비 전력은 감소하지만, 실제 수행 시간이 비례하여 증가하므로, 실제적인 처리 에너지 이득이 없다.

하지만 동작 전압을 주파수에 따라 조절함으로써 실제적인 에너지 이득을 얻게 된다. 낮은 주파수의 클럭을 사용하는 경우, 사용 전압을 낮출 수 있으며 소비 전력은 전압의 제곱에 비례하므로, 수행 시간을 늘리더라도 에너지 이득을 얻게 된다.

프로세서 클럭이 높게 유지되는 경우, 전압 강하는 신호처리 상의 난점으로 인해 동작이 불안정해진다. 따라서, 프로세서의 주파수가 높은 경우에는 전압 강하가 불가능하다.

프로세서 주파수의 변경은 전압 강하보다는 손쉬운 메커니즘이다. 주파수를 변경하는 것은 단순히 클럭 속도를 조절하면 되지만, 프로세서 내부의 전압 강하를 위해서는 안정화를 위한 추가적인 시간이 필요하다.

프로세서의 동적 전력 관리 메커니즘을 다음과 같은 상태 다이어그램 단계로 구분하여 생각해 볼 수 있다.

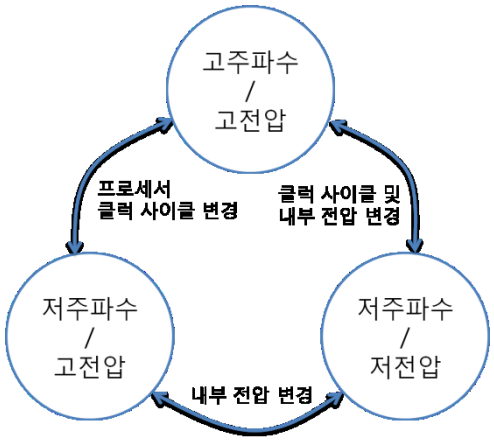


**그림 1 동적 전력 관리 기법 상태 다이어그램**

프로세서 클럭 사이클의 변화에 따른 수행 시간의 증가는 다음과 같이 정의할 수 있다.

$$\mathrm{DiffTime}\left(\frac{f_2}{f_1}\right) = \frac{f_1}{f_2}$$

**수식 15 클럭 사이클 변화에 따른 수행 시간의 증가량**

수행시간의 변화로 인한 태스크 $\tau_i$의 수행 시간 $e_k$의 변화는 다음과 같이 기술할 수 있다.

$$\mathrm{DVFSe}_k = \mathrm{DiffTime}\left(\frac{f_2}{f_1}\right) \cdot e_k$$

**수식 16 동적 전력 관리에 따른 실시간 태스크의 수행 시간**

앞서 정의하였던 demand 함수는 동적 전력 관리 기법에 따라 다음과 같이 정의된다.

$$\mathrm{DVFSdemand_j}(a,b) = \mathrm{DiffTime}\left(\frac{f_2}{f_1}\right)\cdot \mathrm{demand_j}(a,b)$$

**수식 17 동적 전력 관리에 따른 실시간 태스크의 프로세서 요구 시간 함수**

그리고, 임의의 시간 인터벌 Δ에 대한 프로세서 요구량 함수는 다음과 같이 재정의 해야 한다.

$$\mathrm{DVFSdemand}_{\tau_i}^{\max}(\Delta)$$
$$\stackrel{\text{def}}{=} \max_{s\in S,t>0}\left(\sum_{\{j\in S\}}\mathrm{DVFSdemand_j}(t-\Delta,t)\right)$$
$$= \mathrm{DiffTime}\left(\frac{f_2}{f_1}\right)\cdot \max_{s\in S,t>0}\left(\sum_{\{j\in S\}}\mathrm{demand_j}(t-\Delta,t)\right)$$

**수식 18 임의의 인터벌 구간 Δ에 대한 최대 프로세서 요구량 함수**

이에 따른 수행시간 완료 조건은 다음과 같이 변경된다.

$$\mathrm{DVFSe_k} + \sum_{i=1}^{k-1}\mathrm{DVFSdemand}_{\tau_i}^{\max}(d_k) \le d_k$$

**수식 19 동적 전력관리를 사용한 경우의 수행시간 완료 조건**

수식 19는 다음과 같이 변경이 가능하다.

$$\mathrm{DiffTime}\left(\frac{f_2}{f_1}\right)\cdot\left\{e_k + \sum_{i=1}^{k-1}\mathrm{demand}_{\tau_i}^{\max}(d_k)\right\} \le d_k ,$$
$$e_k + \sum_{i=1}^{k-1}\mathrm{demand}_{\tau_i}^{\max}(d_k) \le d_k \cdot \frac{1}{\mathrm{DiffTime}\left(\frac{f_2}{f_1}\right)} ,$$
$$e_k + \sum_{i=1}^{k-1}\mathrm{demand}_{\tau_i}^{\max}(d_k) \le d_k \cdot \left(\frac{f_2}{f_1}\right).$$

**수식 20 동적 전력 관리 사용시 수행 시간 완료 조건식 2**

이 때, $f_2 < f_1$ 이므로, $\frac{f_2}{f_1} < 1$ 이다. 따라서, 실행 시간이 길어진 만큼 임계시간 조건이 더 엄격하다. 이를 수정한 주기형 및 산발형 실시간 작업의 스케줄 가능 조건은 다음과 같다.

$$\mathrm{DVFSe_k} + \sum_{i=1}^{k-1}\left\lceil\frac{d_k}{p_i}\right\rceil\cdot \mathrm{DVFSe_i} \le d_k$$

**수식 21 동적 전력 관리 사용시 스케줄 가능 조건**

이 역시 수식 20에서와 마찬가지로, 실행 시간에 따라 스케줄 가능 조건이 더욱 까다로워짐을 알 수 있다.

$$\mathrm{DiffTime}\left(\frac{f_2}{f_1}\right)\cdot\left\{e_k + \sum_{i=1}^{k-1}\left\lceil\frac{d_k}{p_i}\right\rceil\cdot e_i\right\} \le d_k ,$$
$$\left\{e_k + \sum_{i=1}^{k-1}\left\lceil\frac{d_k}{p_i}\right\rceil\cdot e_i\right\} \le d_k\cdot\left(\frac{f_2}{f_1}\right)$$

**수식 22 동적 전력 관리 사용 시 스케줄 가능 조건**

개선된 요구 조건의 모델에서 역시 동일한 문제가 발생한다. 동적 전력관리 모델 사용 시 수행 요구 비율과 최대 프로세서 수행 요구비율을 다음과 같이 정의할 수 있다.

$$\mathrm{DVFSload}_{\tau_i}(t-\Delta,t) \stackrel{\text{def}}{=} \frac{\mathrm{DVFSdemand}_{\tau_i}(t-\Delta,t)}{\Delta},$$
$$\mathrm{DVFSload}_{\tau_i}^{\max}(\Delta) \stackrel{\text{def}}{=} \frac{\mathrm{DVFSload}_{\tau_i}^{\max}(\Delta)}{\Delta}$$

**수식 23 동적 전력 관리 사용시 수행 요구 비율 및 최대 프로세서 요구 비율 정의**

수행 요구 정의에 따른 수행 시간 완료 조건식은 다음과 같이 재정의 가능하다.

$$\frac{\mathrm{DVFSe_k}}{d_k} + \sum_{i=1}^{k-1}\mathrm{DVFSload}_{\tau_i}^{\max}(d_k) \le 1 ,$$
$$\frac{e_k}{d_k} + \sum_{i=1}^{k-1}\mathrm{load}_{\tau_i}^{\max}(d_k) \le \frac{f_2}{f_1} .$$

**수식 24 동적 전력 관리 사용시 수행시간 완료 조건식**

마찬가지로, 재정의된 수행 요구비율 정의에 따라 최대 허용 가능 프로세서 요구량의 한계값을 다시 쓰면 다음 수식 25를 얻는다.

$$\mathrm{DVFSload}_{\tau_i}^{\max}(\Delta) \le \frac{j\cdot \mathrm{DVFSe_i} + \min(\mathrm{DVFSe_i}, \Delta - jp_i)}{\Delta}.$$

**수식 25 DVFS를 사용한 경우의 최대 프로세서 요구량 한계값**

그리고 Δ값이 $jp_i + \mathrm{DVFSe_i}$ 로 점근함에 따라 수렴하는 하이퍼볼릭 수행 요구 비율 임계값은 다음과 같다.

$$\left(\frac{\Delta - \mathrm{DVFSe_i}}{p_i} + 1\right)\cdot \mathrm{DVFSe_i} = u'_i(\Delta + p_i - \mathrm{DVFSe_i})$$
$$\mathrm{DVFSload}_{\tau_i}^{\max}(\Delta) \le \min\left(1, u_i'\cdot\left(1 + \frac{p_i - \mathrm{DVFSe_i}}{\Delta}\right)\right),$$
$$\mathrm{DiffTime}\left(\frac{f_2}{f_1}\right)\mathrm{load}_{\tau_i}^{\max}(\Delta)$$
$$\le \min\left(1, u_i'\cdot\left(1 + \frac{p_i - \mathrm{DiffTime}\left(\frac{f_2}{f_1}\right)e_i}{\Delta}\right)\right),$$

$$\mathrm{load}_{\tau_i}^{\max}(\Delta) \le \left(\frac{f_2}{f_1}\right) \cdot \min\left(1, u_i' \cdot \left(1 + \frac{p_i - \left(\frac{f_2}{f_1}\right) e_i}{\Delta}\right)\right),$$

**수식 26 DVFS 사용시 하이퍼볼릭 수행 요구 비율 임계값**

위와 같은 모델은 단순히 수행 시간의 길이가 길어짐에 따라 수행시간이 길어지며, 수행 요구 비율이 커짐을 보이고 있다.

## 4. 동적 전력 관리의 효율적 활용

실시간 작업의 스케줄 가능성 검사는 동적 전력 관리 기법의 도입에 따라 매우 엄격해 진다. 이는 프로세서의 동작 주파수에 따라 실행 시간이 영향을 받기 때문이다. 따라서, 높은 실시간성을 필요로 하는 작업에 대해서 동적 전력 관리 기능을 사용하지 않도록 함으로써, 스케줄 가능성 검사를 완화 시킬 수 있다.

실시간 태스크의 스케줄 가능성 검사는 태스크 실행 시점에 해당 태스크보다 우선 순위가 높은 태스크들을 대상으로 실행하게 된다. 따라서, 실시간 태스크들과 인터럽트 처리기에 한해 높은 주파수 클럭을 제공하는 경우, 스케줄 가능한 수행 요구 비율 임계치를 낮추지 않으면서도 스케줄 가능성을 보장할 수 있다.

이러한 기법은 스케줄 가능성을 보장하는 대신, 시스템의 전체 처리량과 소비전력당 처리량의 변화를 가져온다. 전체 시스템의 프로세서 활용율을 1을 기준으로 볼 때, 이전 시스템에서 실시간 처리를 최대한 활용할 수 있는 기준치는 수식 14와 같다. 이 때, 비 실시간 작업-항상 우선 순위가 낮아 언제든 선점이 가능한 작업-이 나머지 프로세서를 점유하고 있다고 가정하자. 즉, 비실시간 작업의 프로세서 최대 수행율($U_{nonRT}$)을 수식27과 같이 가정할 때, 처리량의 증감을 계산할 수 있다.

$$U_{nonRT} = 1 - \min\left(1, u_i\left(1 + \frac{p_i - e_i}{\Delta}\right)\right)$$

**수식 27 비 실시간 작업의 최대 수행율**

$$\Delta\mathrm{Throughput} = \frac{\mathrm{Throughput}_{\mathrm{with\ DVFS}}}{\mathrm{Throughput}_{\mathrm{without\ DVFS}}} = \frac{\mathrm{Time}_{\mathrm{without\ DVFS}}}{\mathrm{Time}_{\mathrm{withDVFS}}}.$$

**수식 28 동적 전력 관리 사용으로 인한 처리량 변화**

$\mathrm{Time}_{\mathrm{without}}$DVFS는 DVFS를 사용하지 않은 경우의 실시간/비실시간 작업을 모두 처리할 때 필요한 시간이며, $\mathrm{Time}_{\mathrm{with}}$DVFS는 비 실시간 작업에 대해 동적 전력 관리를 사용한 경우 이전과 동일한 작업량을 처리하기 위해 필요한 시간이다.

DVFS를 사용함으로 인해 증가한 수행시간을 수식15와 같이 정의할 때, 다음과 같은 처리량 증감 값을 얻게 된다.

$$\frac{\mathrm{Time}_{\mathrm{without\ DVFS}}}{\mathrm{Time}_{\mathrm{with\ DVFS}}} = \frac{U_{nonRT} \times \mathrm{Diff}\left(\frac{f_1}{f_1}\right) + U_{RT} \times \mathrm{Diff}\left(\frac{f_1}{f_1}\right)}{U_{nonRT} \times \mathrm{Diff}\left(\frac{f_2}{f_1}\right) + U_{RT} \times \mathrm{Diff}\left(\frac{f_1}{f_1}\right)} = \frac{U_{nonRT} + U_{RT}}{U_{nonRT} \times \left(\frac{f_1}{f_2}\right) + U_{RT}}.$$

**수식 29 선별적 동적 전력관리 기법 사용시 처리량 저하**

또한, 이로 인한 소비전력량의 변화는 수식2를 통하여 계산할 수 있다.

$$\frac{E_{\mathrm{with\ DVFS}}}{E_{\mathrm{without\ DVFS}}} = \frac{U_{nonRT} \times E' + U_{RT} \times E}{U_{nonRT} \times E + U_{RT} \times E},$$

$$E' \approx \left(\frac{f_2}{f_1}\right) \cdot E.$$

**수식 30 DVFS 사용시 소비에너지 변화량**

수식29와 30을 통해 소비전력당 처리량의 변화를 다음과 같이 계산할 수 있다.

$$\frac{\frac{\mathrm{Throughput}_{\mathrm{with\ DVFS}}}{E_{\mathrm{with\ DVFS}}}}{\frac{\mathrm{Throughput}_{\mathrm{without\ DVFS}}}{E_{\mathrm{without\ DVFS}}}} = \frac{E_{\mathrm{without\ DVFS}} \cdot \mathrm{Time}_{\mathrm{without\ DVFS}}}{E_{\mathrm{with\ DVFS}} \cdot \mathrm{Time}_{\mathrm{with\ DVFS}}} \approx \frac{(U_{nonRT} \times E + U_{RT} \times E) \cdot (U_{nonRT} + U_{RT})}{\left(U_{nonRT} \times \left(\frac{f_2}{f_1}\right) \cdot E + U_{RT} \times E\right) \cdot \left(U_{nonRT} \times \left(\frac{f_1}{f_2}\right) + U_{RT}\right)}$$

**수식 31 선별적 동적 전력 관리 기법 사용시 소비전력당 처리량의 변화**

.

하지만, 위와 같은 선별적 전력 관리 기법도 충분하지 않다. 전압 강하가 일어난 상태에서는 주파수를 바로 상승할 수 없으며, 전압을 충분히 끌어올린 후에 주파수 조절을 통해 처리량을 확보할 수 있기 때문이다. 또한 프로세서 내부 전압을 변경하는 동안 안정화를 위해 시간이 필요하기 때문에, 이러한 오버헤드 역시 고려하여야 한다.

즉, 잦은 전압 변경이 발생하는 경우에는 전압 변경으로 인한 오버헤드가 크게 발생하며, 이는 실시간 스케줄 가능성 검사에 영향을 끼치게 된다.

전압의 변경으로 인해 발생하는 오버헤드가 전압 강하량과 무관하게 일정한 값을 가진다면, 우리는 전압 강하로 인한 오버헤드를 고려한 모델을 생각해 볼 수 있다.

전압 강하로 인한 오버헤드를 DO 라고 가정하면, 수행되는 실시간 태스크의 수의 배수만큼 오버헤드가

추가되므로(강하/상승), 수식8은 다음과 같이 수정된다.

$$\mathrm{demand}_{\tau_i}^{\max}(\Delta) \le \mathrm{j}\cdot(\mathrm{e_i}+2\mathrm{DO})+\min(\mathrm{e_i},\Delta-\mathrm{jp_i})+2\mathrm{DO}$$

**수식 32 전압변경 오버헤드를 고려한 프로세서 요구량 한계값**

따라서, 최대 프로세서 수행 요구 비율(수식13)과 하이퍼볼릭 수행 요구비율 임계값(수식14)는 각각 다음과 같이 변경된다.

$$\mathrm{load}_{\tau_i}^{\max}(\Delta) \le \frac{\mathrm{je_i}+\min(e_i,\Delta-jp_i)+(\mathrm{j}+1)\cdot 2\mathrm{DO}}{\Delta}$$

$$\mathrm{load}_{\tau_i}^{\max}(\Delta) \le \min\left(1, 2\mathrm{DO}\cdot \mathrm{u_i}\left(1+\frac{\mathrm{p_i}-\mathrm{e_i}}{\Delta}\right)\right)$$

**수식 33 전압변경 오버헤드를 고려한 최대 프로세서 수행 요구 비율과 하이퍼볼릭 수행 요구비율 임계값**

전압 변경으로 인한 오버헤드를 고려한 경우, 비실시간 작업의 프로세서 최대 수행율은 수식33과 같이 변경된다.

$$\mathrm{U_{nonRT}} = 1-\min\left(1, 2DO\cdot u_i\left(1+\frac{p_i-e_i}{\Delta}\right)\right)$$

**수식 34 전압변경 오버헤드를 고려한 경우 비실시간 작업의 최대 수행율**

전압 변경으로 인한 처리량의 변화와 소비전력의 변화는 수식 29, 30, 31의 $\mathrm{U_{nonRT}}$를 변경하여 사용할 수 있다.

## 4. 향후 연구

인터럽트가 빈번하게 발생하는 경우나, 실시간 태스크의 처리량이 많아 주파수 변경이 자주 발생하는 경우에는 실시간 처리를 위해, 전압을 상승시켜 둘 수 있다. 이 경우, 전압 변화로 인한 오버헤드를 감소시킬 수 있다. 향후 연구에서는 이와 같은 전압 조절을 위한 보다 효율적인 기법에 대한 연구가 필요하다. 또한, 제시된 분석 내용에 대한 검증 과정이 필요하다. 현재까지의 작업은 분석에 그치고 있기 때문에, 실제 실시간 스케줄러에 적용하여 스케줄 가능성의 영향 및 처리량과 에너지 소비량 변화 등에 대한 세심한 평가가 필요하다.

## 4. 결 론

본 논문에서는 실시간 태스크 스케줄러의 스케줄 가능성 검사가 동적 전력 관리 기법을 도입함으로써 발생하는 영향을 분석하였다. 동적 전력 관리 기법은 프로세서의 전압과 주파수를 조절하여 소비전력당 처리율을 높이는 기술이다. 하지만, 프로세서 클럭이 낮아짐으로 인해, 처리량이 떨어지는 문제가 있다. 즉, 프로그램의 수행 시간이 주파수와 반비례하여 증가하게 되며, 이러한 영향은 실시간 시스템에 큰 영향을 미친다.

본 논문에서는 실시간 태스크 스케줄러의 스케줄 가능성 연구를 기반으로 하여, 동적 전력 관리가 사용된 경우의 최대 허용 프로세서 수행 요구 비율의 임계값을 계산하였다. 또한, 이를 확장하여 선별적으로 동적 전원 관리가 적용된 경우에 대하여 분석을 시도하였다. 실시간 태스크와 인터럽트 처리기에 대해서 최고 클럭을 제공한 경우, 스케줄 가능성 검사와 최대 프로세서 요구량 한계값은 변화가 없다. 하지만, 처리량의 감소와 소비 에너지, 소비전력당 처리량의 변화가 발생하게 된다. 또한 전압 변경으로 인한 오버헤드를 고려한 경우, 최대 프로세서 수행 요구 비율과 하이퍼볼릭 수행 요구 비율 임계값의 변화량을 유도하였다. 또한, 비실시간 작업의 최대 수행율의 변화로 인한 처리량, 소비 에너지, 소비전력당 처리량의 변화를 계산하였다

## 참고문헌


1. J. Rubin Lorch, *Operating Systems Technologies for Reducing Processor Energy Consumption*, http://research.microsoft.com/~lorch/papers/thesis.pdf, 2001.

2. A. Zuquim, A. Loureiro, C Coelho. Jr., M. Vieira, L. Vieira, A. Vieira, A. Fernandes, D. Silva Jr, J. Mata, J. Nacif, H. Carvalho, *Efficient Power Management in Real-Time Embedded Systems*, In proceedings of the IEEE Conference on Emerging Technologies and Factory Automation, pp. 16-19 September 2003.

3. G. Konduri, J. Goodman, A. Chandrakasan, *Energy Efficient Software Through Dynamic Voltage Scheduling* In proceedings of the IEEE ISCAS, 358-361, May 1999.

4. A. Vahdat, A. Lebeck, and C. S. Ellis, *Every joule is precious: the case for revisiting operating system design for energy efficiency* In proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the Pc: New Challenges For the Operating System pp. 31-36, 2000.

5. A. Acquaviva, L. Benini, and B. Riccó, *Energy characterization of embedded real-time operating systems* ACM SIGARCH Computer Architecture News vol. 29, No. 5 (Dec. 2001), pp. 13-18, 2001.

6. P. Pillai and K. G. Shin. *Real-time dynamic voltage scaling for lowpower embedded operating systems.* In proceedings of Symposium on Operating Systems Principles, pp. 89-102, 2001.

7. M. Lewandowski, M. Stanovich, T. P. Baker, K. Gopalan and A. A. Wang *Modeling device driver effects in real-time schedulability analysis: Study of a network driver* In proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium, pp 57-68, 2007.