Journal of The Korea Society of Computer and Information
Vol. 19, No. 12, December 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.12.001

# 주기적 실시간 작업들의 전력 소모 감소를 위한 병렬 수행을 활용한 다중코어 DVFS 스케줄링 기법

박 수 희*

# Multicore DVFS Scheduling Scheme Using Parallel Processing for Reducing Power Consumption of Periodic Real-time Tasks

Suehee Pak*

## 요 약

본 논문에서는 다중코어 프로세서 상에서 주기적 실시간 작업들의 데드라인을 만족하면서 전력 소모량을 최소화하도록 DVFS 기법과 전원 소등 기법을 모두 사용하는 스케줄링 방법을 제안하였다. 제안된 스케줄링 방법은 프로세싱 코어들이 단일 시점에 같은 속도로 동작하는 연관형 프로세서 모델에 적합하도록 설계되었고, 기존 연구에서 해결하지 못한 프로세싱 코어들의 부하불균등 현상을 병렬 수행을 작업들에 적용하여 해소함으로써 전력 소모량을 줄였다. 또한 작업들의 전체 계산량을 고려하여 일부 프로세싱 코어들만을 활성화하여 사용하고 나머지 코어들의 전원은 소등하여 전력 소모량을 줄였다. 전체 프로세싱 코어들 중에서 활성화될 프로세싱 코어들의 개수는 수학적 분석을 통하여 결정되었다. 성능 평가 실험에서 제안된 방법은 기존 방법보다 전력 소모량을 최대 77%까지 감소시킴을 보였다.

▸ Keywords : 실시간 작업, 스케줄링, 병렬 수행, 다중코어 프로세서, 전력 효율적 설계

## Abstract

This paper proposes a scheduling scheme that enhances power consumption efficiency of periodic real-time tasks using DVFS and power-shut-down mechanisms while meeting their deadlines on multicore processors. The proposed scheme is suitable for dependent multicore processors in which processing cores have an identical speed at an instant, and resolves the load unbalance of processing cores by

•제1저자 : 박수희
•투고일 : 2014. 09. 11, 심사일 : 2014. 10. 6, 게재확정일 : 2014. 10. 18.
* 동덕여자대학교 컴퓨터학과 (Dept. of Computer Science, Dongduk Women's University)
※ 이 연구는 2013년도 동덕여자대학교 연구년제도에 의하여 수행된 것임.

exploiting parallel processing because the load unbalance causes inefficient power consumption in previous methods. Also the scheme activates a part of processing cores and turns off the power of unused cores. The number of activated processing cores is determined through mathematical analysis. Evaluation experiments show that the proposed scheme saves up to 77% power consumption of the previous method.

▸ Keywords : real-time task, scheduling, parallel processing, multicore processor, power-efficient design

## I. 서 론

배터리 전원에 의지하여 동작하는 무선 컴퓨팅 장치는 짧은 배터리 잔여 수명으로 인해서 많은 불편을 야기하기 때문에 무선 컴퓨팅 장치의 전력 소모 효율성을 높이는 것은 매우 중요한 문제이다. 본 논문에서는 다중코어 프로세서를 장착한 컴퓨팅 장치가 주기적 실시간 작업들을 수행하면서 소진하는 전력량을 개선시키는 스케줄링 기법을 다룬다.

프로세서의 전력 소모 효율성을 개선시키기 위한 스케줄링 설계에는 두 가지 접근 방법이 있다. 첫 번째 접근 방법은 사용하지 않는 프로세싱 코어의 전원을 소등시켜서 전력 소모량을 줄이는 전원 소등 기법[1]이다. 두 번째 접근 방법은 프로세서의 연산 부하가 적을 때에 낮은 클락 주파수(clock frequency)를 적용하여 프로세싱 코어들의 동작 속도를 줄여서 전력 소모량을 줄이는 DVFS(dynamic voltage and frequency scaling) 기법[2]이다. 프로세서 전력 소모 효율성을 개선시키기 위한 기존 스케줄링 연구들의 대부분은 전원 소등 기법만을 활용하거나[1] 또는 DVFS 기법만을 활용하였다[2-7]. 최근의 일부 스케줄링 연구[8,9,10]에서만 두 가지 기법을 동시에 고려하였지만, 프로세싱 코어들이 단일 시점에 다른 속도 동작을 허용하는 독립형 프로세싱 코어 모델에 적합한 방법이거나 병렬 수행을 고려하지 않아서 전력 소모 감소 효율이 낮은 방법이다. 본 논문에서 제안된 방법은 전원 소등 기법, DVFS 기법 그리고 병렬 수행을 모두 활용하여 전력 소모 감소 효율을 극대화하였고, 또한 프로세싱 코어들이 단일 시점에 같은 속도로 동작하는 연관형 프로세싱 코어 모델에 적합하도록 설계되었다.

제안된 스케줄링 방법은 다중코어 프로세서 상에서 주어진 주기적 실시간 작업들의 데드라인을 항상 만족하면서 프로세싱 코어들이 소모하는 전력량을 최소화하도록 설계되었다. 연관형 다중코어 프로세서 환경에서 전력 소모 효율을 떨어뜨리는 부하불균형을 해소하기 위해서, 높은 연산 부하를 가진 작업에 다수의 코어들을 배치하여 실행시키는 병렬 수행을 적용하여 부하불균형을 완화시킨다. 병렬 수행을 적용할 때 유발되는 오버헤드(overhead)로 인해서 병렬 수행이 적용된 작업을 실행하는 프로세싱 코어들의 전체 계산량은 증가되고, 이로 인해서 전력 소모량이 증가될 수 있다. 따라서 제안된 방법에서는 병렬 수행의 오버헤드로 인한 전력 소모량 증가 부작용과 부하불균등 해소로 인한 전력 소모량 감소 효과를 동시에 분석하여 병렬 수행에 할당될 프로세싱 코어 개수를 결정한다. 또한 작업들의 전체 부하량을 고려하여 일부 프로세싱 코어들만을 활성화하여 사용하고 나머지 코어들의 전원은 소등하여 전력 소모량을 줄였다. 전체 프로세싱 코어들 중에서 활성화될 코어들의 개수는 수학적 분석을 통하여 결정된다.

제안된 방법의 성능을 평가하기 위해서, 병렬 수행을 활용하지 않으면서 일부 프로세싱 코어들만을 활성화하여 전력 소모량을 줄이는 기존 방법[8]과 전력 소모량을 비교하였다. 성능 평가 실험 결과에서 제안된 방법이 기존 방법보다 전력 소모량을 최대 77%까지 감소시킴을 확인하였다.

## II. 관련 연구

실시간 작업들을 실행하는 프로세서의 전력 소모량을 줄이기 위해서 처음에는 전원 소등 기법[1]이 사용되었다. 이 기법에서는 실시간 작업을 수행하고 남는 유휴시간(idle time) 동안에 전원을 소등하여 전력 소모량을 줄이는 방법으로, 실시간 작업들의 데드라인을 만족하면서 소등 기간을 최대화하는 것이 저전력 설계의 주요 관점이다. 전원 소등 기법보다 전력 감소 효율을 향상시키고자 DVFS 기법[2]이 제안되었

다. 전원 소등 기법이 프로세서의 고정된 속도만을 고려한다면, DVFS는 프로세서의 속도를 동적으로 조절하여 전력 소모량을 줄인다. 프로세서의 전력 소모량이 일반적으로 프로세서의 동작 속도의 세제곱에 비례하므로, 실시간 작업을 수행하고 남는 유휴시간 동안 전원을 소등하는 것보다 유휴시간이 발생하지 않도록 낮은 동작 속도를 적용하여 실시간 작업을 수행하는 것이 전력 소모 효율을 대폭 향상시킨다.

최근 들어 프로세서 설계 기술이 발전하면서 하나의 프로세서 칩 내에 다수의 프로세싱 코어들을 내장하는 다중코어 프로세서가 보편적으로 사용되면서, 다중코어 프로세서 상에서 실시간 작업들의 데드라인을 만족시키면서 전력 소모량을 줄이는 많은 스케줄링 방법들[2-6]이 연구되었다. 일반적인 환경에서는 DVFS 기법의 전력 소모 효율이 전원 소등 기법보다 좋기 때문에, 다중코어 프로세서 상에서 설계된 대부분의 스케줄링 방법들은 전체 프로세싱 코어들에게 DVFS 기법을 적용하여 전력 소모량을 줄이도록 설계되었다.

최근의 일부 스케줄링 연구[8,9,10]에서 일부 프로세싱 코어들에게 전원 소등 기법을 적용하여 전원을 소등시키고 나머지 활성화된 코어들에게만 DVFS 기법을 적용하여 전력 소모 효율을 극대화시키는 접근 방법이 고려되었다. 그 이유는 프로세싱 코어의 정적 누수 전력 소모(static leakage power consumption) 때문에, 연산 부하가 매우 낮은 두 개의 프로세싱 코어들이 소모하는 전력량보다 한 개의 프로세싱 코어를 소등시키고 나머지 한 개의 코어만을 이용하여 합친 연산 부하를 실행하는 것이 전력 소모량이 낮다는 사실이 확인되었기 때문이다. 그러나 기존 연구 [8]에서는 연산 부하가 상대적으로 높은 작업들로 인해서 프로세싱 코어들 사이에 심각한 부하불균형이 발생하는 경우에 전력 소모 효율이 떨어지게 되는 단점이 있다. 연산 부하가 상대적으로 높은 작업에 병렬 수행을 적용한다면, 프로세싱 코어들의 연산 부하를 균등화시킬 수 있다. 그리고 다른 연구 [9]과 [10]에서는 프로세싱 코어들이 단일 시점에 다른 속도 동작을 허용하는 독립형 프로세싱 코어 모델에 적합하도록 설계되어 프로세싱 코어들이 단일 시점에 동일한 속도로 동작하는 연관형 프로세싱 코어 모델에서는 전력 소모 효율이 떨어지게 되는 단점이 있다. 기존 연구 [10]에서 연산 부하가 높은 작업들에게 병렬 수행을 적용하여 전력 소모 효율을 개선시켰지만, 독립형 프로세싱 코어 모델에 적합하도록 병렬 수행 여부를 결정하였다.

병렬 수행을 고려한 스케줄링은 최근의 기존 연구 [12]에서도 다루어 졌지만, 이 연구에서는 실행 순서에 선후 관계 제약이 존재하는 부분작업들로 구성된 단일 병렬 작업의 전력 소모를 줄이는 문제를 다루었다. 또한 기존 연구 [13]에서는 클라우드 서버에 적합하도록 실시간 작업 요구를 가상머신 모델로 전환하여 전력 소모 효율을 최적화하는 연구를 진행하였고, 기존 연구 [14]에서는 DVFS의 동작 속도 변환에 필요한 오버헤드를 고려한 전력 소모 최소화 문제를 정형적으로 다루었다.

본 연구에서 제시된 방법은 연관형 다중코어 프로세서 환경에서 병렬 수행의 오버헤드로 인한 전력 소모량 증가 부작용과 부하불균등 해소로 인한 전력 소모량 감소 효과를 동시에 고려하여 전력 소모 효율을 극대화하도록 다수의 실시간 작업들에 대한 스케줄링을 다루었다. 프로세싱 코어들에게 다른 속도 동작을 허용하는 독립형 다중코어 프로세서는 하드웨어 구현이 복잡하고 구현 비용도 증가하기 때문에 대부분의 상용 다중코어 프로세서 칩에는 연관형 구조가 적용된다.

## III. 시스템 환경

다중코어 프로세서 칩에서 사용 가능한 동종의 프로세싱 코어들의 전체 개수를 $M$으로 표시한다. 전원이 소등된 프로세싱 코어들의 전력 소모량은 무시할만한 수준이므로 전력 소모량이 없는 것으로 간주된다[1,9]. 활성화된 프로세싱 코어들의 동작 속도는 시간에 따라 변경할 수 있지만, 단일 순간에는 활성화된 프로세싱 코어들의 동작 속도는 항상 동일하다. 최대 코어 속도를 $S_{\max}$으로 표기하고, 표현의 간결성을 위해 $S_{\max} = 1.0$이 되도록 최대 코어 속도를 정규화한다. 그러면 최대 속도보다 감속된 코어 속도 $S$는 $0 \le S < S_{\max}$의 관계식을 만족한다. 프로세싱 코어의 전력 소모량은 코어 속도의 세제곱에 비례하며[2-6], 프로세싱 코어가 유휴시간에 소모하는 누수전력을 $l_p$로 나타낸다. 프로세싱 코어가 $S$ 속도로 동작할 때의 단일 시간당 소모 전력량을 $F(S)$의 함수로 표기하고 다음과 같이 계산된다.

$$F(S) = \alpha \cdot S^3 + l_p \quad (1)$$

여기서 $\alpha$는 프로세서 하드웨어 특성 상수이다.

서로 독립적으로 수행되는 주기적 실시간 작업들의 전체 개수를 $N$으로 표시하며, 이 작업들을 $T_1, ..., T_N$로 나타낸다. 주기적 작업들의 다음번 도착 주기가 실행을 완료해야 하는 데드라인이 되며, 각 작업 $T_n$의 데드라인을 $D_n$로 나타낸다. 또한 각 작업 $T_n$이 데드라인까지 수행을 완료해야 하는 계산량을 $C_n$로 나타낸다. 각 작업들은 다른 데드라인 $D_n$과

다른 계산량 $C_n$을 가질 수 있다.

각 작업 $T_n$의 실행에 할당된 코어의 개수를 $m_n$로 표시하며, $m_n$개의 코어들을 할당하여 병렬 수행을 적용할 때의 속도 향상을 코어 개수 $1 \le m_n \le M$에 대해서 벡터 $P_n[m_n]$로 나타낸다. 이때 $P_n[1]=1$ 이며, $m_x < m_y$이면 $P_n[m_x] \le P_n[m_y]$ 이다. 그리고 병렬 수행에 필요한 추가 오버헤드[11]로 인해서 $m_n \ge 2$에 대해서 $P_n[m_n] < m_n$ 이다. 모든 작업들의 $P_n[m_n]$ 값은 사전에 주어진다고 가정한다.

$m_n$개의 코어들을 이용하여 $T_n$을 실행할 때, 주어진 계산량 $C_n$을 데드라인 $D_n$에 실행 완료하기 위해서 $m_n$개의 코어들이 최대 속도 $S_{\max}$에 수행해야만 하는 단위 시간당 최소 계산량을 '작업 부하'로 정의하고 $U_n(m_n)$으로 표기하면 $U_n(m_n)$은 다음과 같이 계산된다.

$$U_n(m_n) = m_n \bullet \frac{C_n}{P_n[m_n] \bullet D_n}$$

그리고 $m_n$개의 코어들을 이용하여 $T_n$을 실행할 때, 단일 프로세싱 코어가 최대 속도 $S_{\max}$에 수행해야만 하는 단위 시간당 최소 계산량을 '계산 부하'로 정의하고 $L_n(m_n)$으로 표기하면, $L_n(m_n)$은 다음과 같이 계산된다.

$$L_n(m_n) = \frac{U_n(m_n)}{m_n} = \frac{C_n}{P_n[m_n] \bullet D_n}$$

$U_n(m_n)$과 $L_n(m_n)$의 값은 고정된 값을 가지지 않고, $m_n$과 $P_n[m_n]$의 값이 변화되면 더불어 같이 변동되는 특성을 가진다. 그리고 $m_n=1$이면, $U_n(1)=L_n(1)=\frac{C_n}{D_n}$ 이다.

본 논문에서는 한번 배치된 프로세싱 코어에서만 계속해서 실행되고, 동적 작업 이동(배치가 완료된 작업을 수행 중에 다른 프로세싱 코어로 이동하여 수행하는 과정)은 지원하지 않는 환경을 고려한다. 하나의 프로세싱 코어에 여러 개의 실시간 작업들이 배치되었을 때, 배치된 작업들의 계산 부하 합을 '코어 부하'로 정의하고 $CL$로 표기하여 사용한다. 하나의 프로세싱 코어에 $T_1, ..., T_k$ 작업들이 배치되었을 때, $CL$은 다음과 같이 계산된다.

$$CL = \sum_{i=1}^{k} L_i(m_i) = \sum_{i=1}^{k} \frac{C_i}{P_i[m_i] \bullet D_i}$$

그리고 하나의 프로세싱 코어에게 배치된 실시간 작업들의 데드라인을 모두 만족하는 최소 코어 속도를 '코어 최적 속도'라 정의하고 $s_{opt}$로 표기한다. 코어 최적 속도 $s_{opt}$는 단위 시간당 수행하는 계산량을 나타내고, $s_{opt} = CL$이다. 프로세싱 코어는 $s_{opt}$의 코어 속도를 적용하여 배치된 실시간 작업들중에서 데드라인이 가까운 작업부터 먼저 실행하면 배치된 실시간 작업들의 데드라인을 모두 만족시킬 수 있음이 이미 입증되었다[3-6]. $s_{opt}$의 속도를 적용할 때의 프로세싱 코어의 전력 소모량은 수식 (1)에 의해서 $\alpha \bullet (s_{opt})^3 + l_p$이다.

## IV. 제안된 스케쥴링 기법

### 1. 작업 배치 이후의 전력 소모량 계산

본 논문에서 고려하는 시스템 환경은 III장에서 언급하였듯이, 활성화된 코어들의 속도는 다른 시점에는 다른 속도를 적용할 수 있어도 같은 시점에는 동일한 속도가 프로세싱 코어들에게 적용된다. 따라서 프로세싱 코어들에게 배치된 모든 작업들의 데드라인을 만족시키는 최소 코어 속도는, 프로세싱 코어들의 최적 코어 속도 $s_{opt} = CL$ 값들 중에서 최대값이다. $M$개의 프로세싱 코어들 중에서 일부 $\beta$개의 프로세싱 코어만을 사용하여 주어진 $N$개의 작업들을 배치한 이후의 프로세싱 코어들의 코어 부하 값이 $CL_1, ..., CL_\beta$일 때, $\beta$개의 프로세싱 코어들에 동일하게 적용될 최적 코어 속도들의 최대값을 $s_{opt}^{\max}$로 표기하고, 다음과 같이 계산된다.

$$s_{opt}^{\max} = \max(CL_1, ..., CL_\beta)$$

또한 $\beta$개의 활성화된 프로세싱 코어들과 $(M-\beta)$개의 전원이 소등된 프로세싱 코어들로 구성된 프로세서 전체의 단위 시간당 전력 소모량은 다음과 같이 결정된다.

$$\beta \bullet \{\alpha \bullet (s_{opt}^{\max})^3 + l_p\} = \beta \bullet \{\alpha \bullet (\max(CL_1, ..., CL_\beta))^3 + l_p\} \quad (2)$$

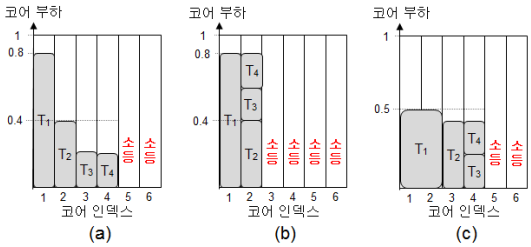


그림 1. 제안된 기법의 동작 예
Fig. 1. Working example of the proposed scheme

제안된 방법은 수식 (2)의 값을 최소화시키는 $m_n$의 값들과 $\beta$ 값을 찾는다. 그림 1은 제안된 방법을 적용하여 전력 소모량을 감소시키는 예제이다. 그림 1(a)에서 $\beta=4$이고 $m_1=m_2=m_3=m_4=1$이며 $s_{opt}^{\max}=0.8$이다. 그림 1(b)에서는 활성화된 코어를 줄여서 프로세싱 코어들의 부하가 균등화되도록 스케줄한 경우로 $\beta=2$이고 $s_{opt}^{\max}=0.8$이다. 그림 1(b)는 그림 1(a)에 비해서 누수전력 $l_p$ 값의 감소로 인해서 수식 (2)의 값이 줄어든다. 그림 1(c)는 병렬 수행을 적용하여 $s_{opt}^{\max}$의 값을 줄이도록 스케줄한 경우로 $\beta=4$이고 $s_{opt}^{\max}=0.5$이다. 그림 1(c)는 그림 1(b)에 비해서 사용된 프로세싱 코어 개수는 2배로 늘었지만 세제곱의 영향을 미치는 코어 속도 $s_{opt}^{\max}$가 5/8만큼 줄어들어서, $l_p$ 값의 영향이 $\alpha$ 값의 영향보다 상대적으로 적은 경우에 오히려 수식 (2)의 값이 감소될 수도 있다. 병렬 수행을 적용하기 이전인 그림 1(a)와 그림 1(b)에서 4개 작업들의 전체 작업 부하 $\sum U(m)$은 1.6이고, 병렬 수행을 적용한 이후인 그림 1(c)에서 $\sum U(m)$은 1.8로 증가하였다. 그림 1의 예제에서 병렬 수행의 오버헤드로 인해서 전체 작업 부하 합은 오히려 증가하지만 병렬 수행을 적용하여 부하 불균등이 완화되어 $s_{opt}^{\max}$ 값이 줄어들게 되면 전력 소모량이 감소될 수 있음을 보여주고 있다.

주어진 작업들을 프로세싱 코어들에게 배치한 이후의 전력 소모량이 위의 수식 (2)과 같이 결정된다면, 남겨진 문제들은 $m_n$의 값들과 $\beta$ 값을 결정하는 문제와 $N$개의 작업들을 $\beta$개의 프로세싱 코어들에게 배치시키는 문제이다. 남겨진 문제들은 해결하는 기법들은 다음 2절과 3절에서 다룬다.

## 2. 활성화될 코어 개수 결정

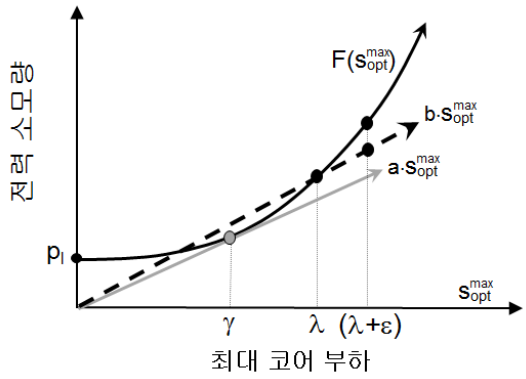


그림 2. 활성화 코어 개수 결정 과정
Fig. 2. Decision process of the number of activated cores

기존 연구[8]에서 프로세싱 코어들의 부하 불균등을 고려하지 않고 활성화될 코어 개수 $\delta$를 다음과 같이 결정하였다. 그림 2와 같이 코어 속도 $s_{opt}^{\max}$를 입력으로 하는 전력 소모량 함수 $F(s_{opt}^{\max})=\{\alpha\cdot(s_{opt}^{\max})^3+l_p\}$에 대해서, 정점을 지나는 일차 직선 함수 $a\cdot s_{opt}^{\max}$와 단일 지점 $s_{opt}^{\max}=\gamma$에서만 접하도록 $a$의 값과 $\gamma$의 값을 찾는다. 그리고 전체 작업 부하 합 $\sum U(1)$을 $\gamma$로 나눈 값을 $\delta(=\frac{\sum U(1)}{\gamma})$라고 했을 때, $\delta$개의 활성화된 코어들이 모두 $s_{opt}^{\max}=\gamma$의 코어 부하를 가지는 경우가 전력 소모량이 최소화됨을 보였다. 그러나 부하 불균등으로 인해서 $s_{opt}^{\max}>\gamma$라면 기존 연구에서 밝혀진 사실을 적용할 수 없게 된다.

따라서 본 논문에서는 부하 불균등으로 인해서 $s_{opt}^{\max}>\gamma$인 경우에 대해서, 프로세싱 코어들의 부하 불균등을 고려하여 수식 (2)를 최소화시키는 활성화될 코어 개수 $\beta$ 값을 찾는다. 부하 불균등을 유발시키는 경우의 $s_{opt}^{\max}$는 다음과 같이 구할 수 있다.

$$s_{opt}^{\max}=\max\{\gamma,L_1(m_1),...,L_N(m_N)\} \quad (3)$$

전체 작업 부하 합 $\sum U$과 $s_{opt}^{\max}$가 주어지는 경우에 대해서, 다음의 정리 1에서 전체 프로세서의 전력 소모량을 최소화시키도록 활성화될 코어 개수 $\beta$ 값을 결정하는 과정을 설명한다. 단 정리 1에서 $\beta$는 정수가 아닌 유리수이다.

**정리(theorem) 1**: 전체 작업 부하 합 $\sum U$과 $s_{opt}^{\max}$가 주어지면, 전체 작업 부하 $\sum U$을 $\beta = \frac{\sum U}{s_{opt}^{\max}}$개의 프로세싱 코어들에 균등 분배하는 것이 전체 전력 소모량을 최소화한다.

증명: 활성화될 코어 개수가 고정된 경우에는 전체 작업 부하를 주어진 코어들에게 균등 분배하는 것이 전력 소모량이 최소화됨이 이미 입증되었다[5]. 먼저 $\beta$개의 코어들에게 $\sum U$을 균등 분배하여 전력 소모량을 최소화한 경우와 $\beta$ 보다 많은 $(\beta+\omega)$개의 코어들에 $\sum U$을 분배한 경우와 비교한다. $\beta$개의 프로세싱 코어들에게 $\sum U$을 균등 분배하면, 이 경우의 전체 전력 소모량은 $\beta \bullet F(s_{opt}^{\max})$가 된다. $(\beta+\omega)$개의 코어들에 $\sum U$을 분배하여도 $s_{opt}^{\max}$가 변동되지 않는다면 이 경우의 전체 전력 소모량은 $(\beta+\omega) \bullet F(s_{opt}^{\max})$가 된다. 양의 수 $\omega$에 대해서, $(\beta+\omega) \bullet F(s_{opt}^{\max}) > \beta \bullet F(s_{opt}^{\max})$ 이다. 즉 $(\beta+\omega)$개의 코어들에 $\sum U$을 분배하여도 최대 코어 속도가 $s_{opt}^{\max} = \frac{\sum U}{\beta}$의 값을 가진다면, $\beta$개의 프로세싱 코어들에게 $\sum U$을 균등 분배하여 각 코어가 $s_{opt}^{\max} = \frac{\sum U}{\beta}$의 코어 속도를 가지는 경우보다 항상 전력 소모량이 크다.

다음으로 $\beta$ 보다 적은 $(\beta-\omega)$개의 코어들에 $\sum U$을 균등하게 분배하여 전력 소모량을 최소화한 경우와 비교한다. $\beta$ 개의 프로세싱 코어들에게 $\sum U$을 균등 분배하여 각 프로세싱 코어가 $CL = \frac{\sum U}{\beta} = \lambda$의 부하 값을 가진다면, 이 경우에 $s_{opt}^{\max} = \lambda$이고 $\beta > (\beta-\omega)$ 이므로 $\gamma < \lambda$ 이다. 그리고 그림 2에서 보여주듯이 정점을 지나는 일차 직선 함수 $b \bullet s_{opt}^{\max}$와 $F(s_{opt}^{\max})$ 함수가 $s_{opt}^{\max} = \lambda$ 지점에서 만나도록 $b$의 값을 지정한다. 그러면 $\beta$ 개의 프로세싱 코어들에게 $\sum U$을 균등 분배한 경우의 전체 전력 소모량은 $\beta \bullet F(\lambda) = \beta \bullet b \bullet \lambda$가 된다.

$(\beta-\omega)$개의 코어들에게 $\sum U$을 균등 분배하면, 각 프로세싱 코어는 $CL = \frac{\sum U}{\beta-\omega} = \lambda+\epsilon$의 값을 가진다. 그리고 $\sum U = (\beta-\omega) \bullet (\lambda+\epsilon) = \beta \bullet \lambda$ 이므로 $(\beta-\omega) \bullet b \bullet (\lambda+\epsilon) = \beta \bullet b \bullet \lambda = \beta \bullet F(\lambda)$이다. 그림 2에서 보여주듯이 $F(\lambda+\epsilon) > b \bullet (\lambda+\epsilon)$ 이므로, $(\beta-\omega) \bullet F(\lambda+\epsilon) > \beta \bullet F(\lambda)$ $= (\beta-\omega) \bullet b \bullet (\lambda+\epsilon)$ 이다. 이 수식은 $(\beta-\omega)$개의 코어들에게 $\sum U$을 완전 균등하게 분배하는 경우의 전력소모량 $(\beta-\omega) \bullet F(\lambda+\epsilon)$이, $\beta$개의 코어들에게 $\sum U$을 완전 균등하게 분배하는 경우의 전력소모량 $\beta \bullet F(\lambda)$ 보다 항상 큼을 의미한다.

위의 논리들에 근거하여 전체 작업들을 $\beta$개의 프로세싱 코어들에 균등 분배하여 각 코어가 $s_{opt}^{\max} = \frac{\sum U}{\beta}$의 부하를 가지는 경우가 전체 전력 소모량을 최소화한다. ▣

위에서 도출된 유리수 $\beta$을 이용하여 제안된 방법은 실제로 활성화될 프로세싱 코어 개수인 자연수를 다음과 같이 결정한다. $\beta$가 $\beta \leq M$인 자연수이라면 $\beta$ 값이 활성화될 코어 개수가 되고, $\beta$가 자연수가 아니고 $k < \beta < (k+1)$인 소수라면 $\beta$에 근접된 두 개의 연속된 자연수 $k$과 $(k+1)$ 중에서 하나를 선택하여 사용한다. $k$개의 프로세싱 코어들 사용할 때의 최소 전력 소모량과 $(k+1)$개의 프로세싱 코어들을 사용할 때의 최소 전력 소모량를 비교하여, 더 적은 전력 소모량 값을 가지는 프로세싱 코어 개수를 $\beta$에 적용한다.

## 3. 병렬 수행 및 작업 배치 결정

본 절에서는 각 작업 $T_n$의 실행에 할당될 코어 개수 $m_n$를 결정한다. 모든 $m_n$은 $m_n = 1$로 초기화되며, 부하 불균등을 유발하는 작업에 대해서만 $m_n$의 값을 증가시켜서 $s_{opt}^{\max}$를 감소시킨다. 부하 불균등을 유발시키는지 여부는 각 작업 $T_n$에 대해서 아래 수식 (4)을 검사하면 알 수 있다.

$$L_n(m_n) > \frac{\sum_{i=1}^{N} U_i(m_i)}{\beta} \quad (4)$$

수식 (4)를 만족하는 작업들에 대해서 가장 큰 $L_n(m_n)$ 값을 가지는 작업의 $m_n$ 값을 증가시킨다. 그 이유는 가장 큰 $L_n(m_n)$ 값을 감소시켜야만 $s_{opt}^{\max}$가 감소되고, 이외의 $L_j(m_j)$ 값을 감소시켜도 $s_{opt}^{\max}$는 감소되지 않는다. 가장 큰 $L_n(m_n)$ 값을 가지는 작업의 $m_n$ 값을 증가시키면, $s_{opt}^{\max}$ 값이 감소되지만 또한 병렬 수행 오버헤드 증가로 인해서 $U_n(m_n)$ 값이 증가하게 되므로 $\sum U$도 증가된다. 변경된 $s_{opt}^{\max}$ 값과 변경된 $\sum U$ 값을 기반으로 정리 1에 근거하여 변경된 활성화 코어 개수 $\beta'$를 결정하고, $\beta'$개의 활성화된 코어들의 최소 전력 소모량을 계산한다. 만약 $m_n$ 증가 이후

에 프로세서의 최소 전력 소모량이 오히려 증가된다면, $m_n$ 증가를 취소하고 기존의 $m_n$ 값들과 $\beta$ 값을 최종 값으로 결정한다. 다시 말해서, 가장 큰 $L_n(m_n)$ 값을 가지는 작업에 대해서 $m_n$ 값을 증가시키는 과정을 프로세서의 전체 전력 소모량이 더 이상 감소되지 않을 때까지 반복한다.

모든 $m_n$ 값들과 $\beta$ 값이 결정되면 마지막으로 주어진 $N$개의 작업들을 $\beta$개의 프로세싱 코어들에게 어떻게 배치할 것인지를 결정하여야 한다. 고정된 개수의 프로세싱 코어들에게 주어진 작업들의 부하를 최대한 균등하게 배치하는 문제는 NP-hard 복잡도를 가진다는 것은 이미 입증된 사실이다[5,6]. 최적의 해법을 찾기 위해서는 모든 경우들에 대해서 전수 검사를 해야 하지만, 전수 검사는 NP-hard에 해당하는 많은 시간을 요구한다. 따라서 본 논문에서는 전수검사 대신 저전력 작업배치 문제에 가장 우수한 성능을 보이는 WFD 휴리스틱(Worst-Fit Decreasing heuristic) 기법[5]을 이용하여 빠른 시간내에 $N$개의 작업들을 $\beta$개의 프로세싱 코어들에게 배치한다. WFD 기법은 주어진 작업들을 계산 부하 $L_n(m_n)$ 값을 기준으로 내림차순으로 정렬하여 계산량 부하 $L_n(m_n)$ 값이 큰 작업부터 배치하고, 각각의 작업들을 배치할 때 $\beta$개의 프로세싱 코어들 중에서 코어 부하 $CL$ 값이 가장 작은 $m_n$개의 코어들에게 해당 작업을 배치하는 방법이다.

아래의 의사 코드(pseudo code)는 본 논문에서 제안된 스케줄링 기법의 전체 동작 과정을 보여주고 있다.

---

**단계 1.** 부하 불균등을 고려하지 않고 활성화될 코어 개수와 전력 소모량을 계산한다.

**1.1:** 각 $m_n$을 $m_n = 1$로 초기화하고, 활성화될 코어 개수 $\beta = \frac{\sum U(1)}{\gamma}$를 계산한다.

**1.2:** 수식 (3)을 이용하여 $s_{opt}^{\max}$ 값을 계산하고, $\beta$개의 프로세싱 코어들의 최소 전력 소모량을 계산한다.

**단계 2.** 부하 불균등을 고려하여 활성화될 코어 개수 $\beta$와 각 $m_n$ 값을 결정한다.

**2.1:** 수식 (4)를 만족하는 부하 불균등을 유발하는 작업들 중에서 가장 큰 $L_n(m_n)$ 값을 가지는 작업의 $m_n$ 값을 증가시켜서 변경된 $s_{opt}^{\max}$와 $\sum U(m)$를 계산한다.

**2.2:** 변경된 $s_{opt}^{\max}$와 $\sum U(m)$ 값을 기반으로 변경된 코어 개수 $\beta$를 계산하고, $\beta$개의 프로세싱 코어들의 최소 전력 소모량을 계산한다.

**2.3:** $m_n$ 값 증가 이후의 전력 소모량과 $m_n$ 값 증가 이전의 전력 소모량을 비교하여, 전력 소모량이 감소되는 경우에만 $m_n$ 값을 증가시키고 전력 소모량이 감소되지 않으면 기존의 $m_n$ 값들과 $\beta$ 값을 최종 값으로 고정시킨다.

**2.4:** 2.1, 2.2 그리고 2.3 단계를 프로세서의 전력 소모량이 더 이상 감소되지 않을 때까지 반복하여, $m_n$ 값들과 $\beta$ 값을 결정한다.

**단계 3.** 결정된 $m_n$ 값들과 $\beta$ 값을 기반으로 최종 스케줄을 결정한다.

**3.1:** WFD 휴리스틱에 기반하여 $N$개의 작업들을 $\beta$개의 프로세싱 코어들에게 배치한다.

**3.2:** 최대 코어 부하 $CL$ 값을 기반으로 $\beta$개의 코어들에게 동일하게 적용할 최적 코어 속도 $s_{opt}^{\max}$를 결정한다.

**3.3:** 각 코어들은 배치된 작업들중에서 데드라인이 가까운 작업들부터 순차적으로 실행한다.

---

위에서 제시된 코드의 전체 계산 복잡도는 $O(N^2 \bullet M)$이다. 단계 1.1과 단계 1.2의 계산 복잡도는 각각 $O(N)$이다. 단계 2.1의 계산 복잡도는 $O(N)$이고, 단계 2.2의 계산 복잡도는 $O(1)$이며, 단계 2.3의 계산 복잡도는 $O(1)$이다. 그리고 단계 2.3은 최대 $N \bullet M$번 이내에서 반복한다. 단계 3.1의 계산 복잡도는 $O(N \bullet M)$이고, 단계 3.2의 계산 복잡도는 $O(N)$이며, 단계 3.3의 계산 복잡도는 계산 복잡도는 $O(N \bullet \ln N \bullet M)$이다.

## V. 성능 평가

성능 평가를 위하여 제안된 기법과 기존 방법[8]의 전력 소모량을 비교하였다. 기존 방법에서는 프로세싱 코어들의 부하 불균등 현상과 병렬 수행을 고려하지 않고 코어 속도와 활성화될 코어 개수를 결정하였다. 평가 지표로는 '기존 방법이 소모하는 단위 시간당 전력량' 대비 '제안된 방법이 소모하는 단위 시간당 전력량' 비율을 '상대적 전력 소모량'이라고 정의하고 이를 성능 지표로 사용하였다.

성능 평가는 시뮬레이션 실험을 통해서 이루어졌고, Window 7 운영체제 환경에서 MATLAB 툴을 사용하여 가

상의 주기적 실시간 작업과 가상의 멀티코어 프로세서 환경을 구축하였다. 실험에서 32개의 주기적 실시간 작업들을 사용하고, 각 작업들은 다른 작업 부하 $U_n(1)$을 가지도록 생성되었다. 작업 부하 값들은 정규 분포를 따르도록 인위적으로 생성되었다. 평가 신뢰도를 높이기 위하여 32개의 작업들로 구성된 집합을 10만번 생성하여 평균 결과 값을 실험 결과로 표시하였다. 사용 가능한 최대 프로세싱 코어 개수를 32개로 설정하였다. 상대적 전력 소모량을 평가 지표로 사용하면 코어 속도의 실제 값은 성능에 영향을 미치지 않으므로, 최대 코어 속도 $S_{\max}$는 1.0의 값을 가지도록 설정하였다. $0 \le S \le S_{\max}$의 관계식을 만족하는 코어 속도 $S$의 단위 시간당 전력 소모량은 실제로 널리 사용되는 인텔 XScale 프로세서의 DVFS 기법 데이터에 최소제곱근회기분석(least-square curve fitting) 방법을 적용하여, 단일 시간당 소모 전력량을 나타내는 $F(S) = \alpha \cdot S^3 + l_p$ 함수의 상수 $\alpha$와 $l_p$를 도출하였다[6]. 도출된 $\alpha$는 $1.55 \times 10^{-6}$이고, $l_p$는 60 mW이다.

병렬 수행을 적용하면 일반적으로 할당된 코어 개수에 비례하여 계산 부하 $L(m)$이 줄어들지만, 병렬 수행의 속도 향상 $P[m] = \frac{U(1)}{L[m]}$은 작업의 부속 모듈 분할 특성에 따라 달라진다[11]. 다양한 작업들의 병렬 수행 효율성을 평가하기 위해, $m$개의 코어들에서의 병렬 수행 속도 향상 모델들을 다음과 같이 정의하였다.

- 선형 속도증가: $P[m] = m$
- 반선형 속도증가: $P[m] = 0.5 \cdot (m-1) + 1$
- 제곱근 속도증가: $P[m] = \sqrt{m}$

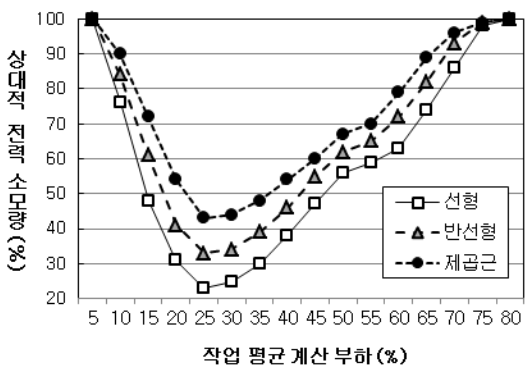


그림 3. 평균 작업 부하에 대한 상대적 전력 소모량
Fig. 3. Relative power consumption again average task workload

32개 작업들의 평균 작업 부하를 $\frac{\sum U(1)}{N} \times 100$로 정의할 때, 그림 3은 평균 작업 부하에 대한 제안된 방법의 상대적 전력 소모량을 보여주고 있다. 평균 작업 부하 값이 25%일 때 최저 상대적 전력 소모량을 보인다. 그리고 평균 작업 부하가 20% 이하이면 높은 평균 작업 부하에 대해서 낮은 상대적 전력 소모량을 보이고, 평균 작업 부하가 30% 이상이면 높은 평균 작업 부하에 대해서 높은 상대적 전력 소모량을 보인다. 평균 작업 부하 값이 5% 이거나 80% 이상 되면 상대적 전력 소모량이 100%가 된다. 10%와 75% 사이의 평균 작업 부하 값들에 대해서 선형 속도 증가 모델이 가장 낮은 상대적 전력 소모량을 보이고, 반선형 속도 증가 모델이 다음으로 낮은 전력 소모량을 보이며, 제곱근 속도 증가 모델이 가장 높은 전력 소모량을 보인다. 평균 작업 부하 값이 25%일 때, 선형 속도 증가 모델의 상대적 전력 소모량은 약 23%이고 반선형 속도 증가 모델은 약 33%이며 제곱근 속도 증가 모델은 약 43%이다. 그림 3의 실험 결과에서 제안된 방법은 기존 방법의 전력 소모량을 최대 77%까지 감소시키고, 또한 다양한 병렬 수행 속도 향상 모델에 대해서 우수한 전력 소모 감소량이 나타남을 확인할 수 있다.

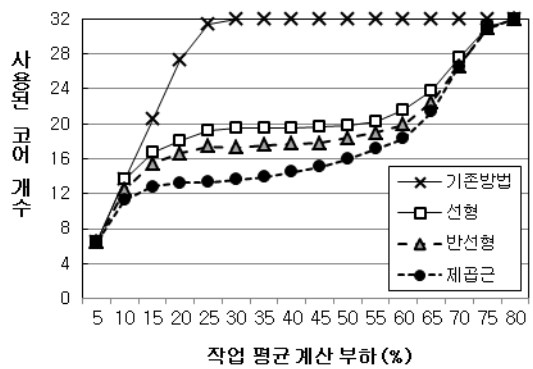


그림 4. 활성화되어 사용된 프로세싱 코어 개수
Fig. 4. Number of activated processing cores

그림 4는 주어진 32개의 프로세싱 코어들 중에서 기존 방법과 제안된 방법에서 활성화되어 사용된 코어 개수를 보여주고 있다. 기존 방법과 제안된 방법 모두에서 작업들의 평균 작업 부하 값이 낮을수록 사용된 코어 개수는 줄어들었다. 그리고 10%와 75% 사이의 평균 작업 부하 값들에 대해서 제안된 방법이 기존 방법보다 사용된 코어 개수가 작았다. 또한 동일한 평균 작업 부하 값에 대해서 선형 속도 증가 모델이 상대적으로 가장 많은 코어들을 활성화하여 사용하고, 반선형

속도 증가 모델이 다음으로 많은 코어들을 사용하며, 제곱근 속도 증가 모델이 가장 적은 코어들을 사용하였다.

그림 3과 그림 4의 결과에서 선형 속도 증가 모델이 다른 속도 증가 모델과 비교하여 더 많은 수의 코어들을 사용하며 또한 상대적 전력 소모량이 낮다는 사실을 알 수 있고, 이는 병렬 수행을 상대적으로 더 많이 활용하였다는 것을 의미한다. 그리고 제곱근 속도 증가 모델이 다른 속도 증가 모델과 비교하여 상대적으로 병렬 수행을 적게 활용하였다.

제안된 기법은 스케줄러가 사전에 각 실시간 작업의 계산량과 병렬 수행에 따른 속도 향상 값을 알고 있어야 하지만, 실제 시스템에서는 작업의 계산량과 병렬 수행의 속도 향상 값은 일정하지 않고 변동되므로 최악의 값을 사전에 측정해서 채용해야만 한다[6,10]. 또한 실제 시스템 환경에서는 DVFS가 코어 속도를 동적으로 변환하는 과정에서 시간 지연과 추가 전력 소모량이 발생하므로 이 오버헤드를 고려한 스케줄링 교정 절차[14]가 추가로 필요하다.

## VI. 결 론

제안된 스케줄링 방법은 DVFS 기법과 전원 소등 기법을 모두 활용하여 프로세싱 코어들이 단일 시점에 같은 속도로 동작하는 연관형 멀티코어 프로세서의 전력 소모량을 최소화하도록 설계되었다. 제안된 방법에서는 프로세싱 코어들의 부하불균등을 유발하는 최대 부하 작업에 병렬 수행을 적용하여 부하불균등을 해소시킴으로써 전력 소모량을 줄였다. 병렬 수행의 오버헤드로 인한 전력 소모량 증가 부작용과 부하불균등 해소로 인한 전력 소모량 감소 효과를 동시에 고려하여 병렬 수행에 할당될 프로세싱 코어 개수를 결정하였다. 또한 기존 방법에서는 전원 소등 기법이 적용되는 활성화 코어 개수를 프로세싱 코어들의 부하불균등을 고려하지 않고 결정하였다면, 제안된 방법에서는 활성화 코어 개수를 프로세싱 코어의 최대 부하를 기반으로 수학적 분석을 통해서 도출하여 전력 소모량을 최소화시켰다. 성능 평가 실험에서 제안된 스케줄링 방법이 기존 저전력 스케줄링 방법의 소모 전력을 최대 77% 까지 감소시킴을 확인하였다.

## 참고문헌


[1] D.L. Benini, A. Bogliolo, and G. Micheli, ``A survey of design techniques for system-level dynamic power management,'' IEEE Trans. VLSI Syst., vol. 8, no. 3, pp. 299-316, 2000.

[2] J.R. Lorch and A.J. Smith, ``Improving dynamic voltage scaling algorithms with PACE,'' Performance Evaluation Review, vol. 29, pp. 50-61, 2001.

[3] C. Yang, J. Chen, and T. Kuo, ``An approximation algorithm for energy-efficient scheduling on a chip multiprocessor,'' Design, Automation and Test in Europe Conference, pp. 468-473, 2005

[4] J.-J. Chen and T.-W. Kuo, ``Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics,'' Int'l Conf. Parallel Processing, pp. 13-20, 2005.

[5] H. Aydin and Q. Yang, ``Energy-aware partitioning for multiprocessor real-time systems,'' Int'l Parallel Distributed Processing Symp., p. 113.2, 2003.

[6] C. Xian, Y.-H. Lu, and Z. Li, ``Dynamic voltage scaling for multitasking real-time systems with uncertain execution time,'' IEEE Trans. Computer Aided Design and Integrity Circuits Systems, vo. 27, no. 8, pp. 1467-1478, 2008.

[7] J. Choi, N. Park, and D. Ahn, ``A lower power scheduling and allocation for multiple supply voltage,'' Journal of the Korea Society of Computer and Information, vol. 7, no. 2, pp. 79-86, 2002.

[8] W.Y. Lee, ``Power-efficient scheduling of Periodic Real-time Tasks on Lightly Loaded Multicore Processors'' Journal of the Korea Society of Computer and Information, vol. 17, no. 8, pp. 11-19, 2012.

[9] E. Seo, J. Jeong, S. Park, and J. Lee, ``Energy efficient scheduling of real-time tasks on multicore processors,'' IEEE Trans. Parallel Distrib. Syst., vol. 19, no. 11, pp. 1540-1552, 2008.

[10] W. Lee, ``Energy-saving DVFS scheduling of multiple periodic real-time tasks on multi-core processors,'' IEEE/ACM Symp. Distributed Simulation and Real Time Applications, pp.

216-223, 2009.

〔11〕 D.L. Eager, J. Zahorjan and E.D. Lozowska, ``Speedup versus efficiency in parallel systems,'' IEEE Trans. Computers, vol. 38, no. 3, pp. 408-423, 1989.

〔12〕 L. Wang, S.U. Khan, D. Chen, J. Kolodziej, R. Ranjan, C. Xu, and A. Zomaya, ``Energy-aware parallel task scheduling in a cluster,'' Future Generation Computer Systems, vol. 29, pp. 1661-1670, 2013.

〔13〕 K.H. Kim, A. Beloglazov and R. Buyya, ``Power-aware provisioning of virtual machines for real-time cloud services,'' Concurrency and Computation: Practice and Experience, vol. 23, no. 13, pp. 1491-1505, 2011.

〔14〕 W.-Y. Shieh and C.-C. Pong, ``Energy and transition-aware runtime task scheduling for multicore processors,'' Journal of Parallel and Distributed Computing, vol. 73, no. 9, pp. 1225-1238, 2013.

## 저 자 소 개

박 수 희

1989: 서울대학교 계산통계학과 학사.

1991: Univ. of California, San Diego 컴퓨터공학과 공학석사.

1994: Univ. of California, San Diego 컴퓨터공학과 공학박사.

현 재: 동덕여자대학교 컴퓨터학과 교수

관심분야: 소프트웨어공학, 실시간 시스템

Email : pak@dongduk.ac.kr