Journal of The Korea Society of Computer and Information
Vol. 17 No. 8 , August 2012

2012-17-8-1-2

저부하 멀티코어 프로세서에서 주기적 실시간 작업들의 저전력 스케줄링

이 완 연*

Power-efficient Scheduling of Periodic Real-time Tasks on Lightly Loaded Multicore Processors

Wan Yeon Lee*

요 약

본 논문에서는 작업 개수보다 프로세싱 코어 개수가 많은 저부하 멀티코어 프로세서에 적합한 실시간 작업용 저전력 스케줄링 기법을 제안하였다. 제시된 기법을 시스템상에 존재하는 모든 프로세싱 코어들을 사용하지 않고, 주어진 작업들의 전체 계산량을 고려하여 일부의 프로세싱 코어들만을 사용하고 나머지 사용하지 않는 코어들의 전원을 소등하여 전력소모량을 줄였다. 또한 휴리스틱 기법을 사용하여 주어진 작업들을 프로세싱 코어들에게 빠르게 배치하였다. 마지막을 각각의 프로세싱 코어는 배치된 작업들의 데드라인 모두 만족하면서 전력소모량을 최소화하도록 프로세싱 코어에 적용되는 최적의 클락 주파수를 선택하여 사용하였다. 제시된 스케줄링 기법과 기존의 프로세싱 코어들을 최대한 많이 사용하는 방법을 비교하는 실험에서, 제시된 기법이 기존 방법의 전력소모량을 최대 78%까지 감소시킴을 확인하였다.

▸ Keywords : 저전력 설계, 멀티코어 프로세서, 작업 스케줄링, 실시간 시스템

Abstract

In this paper, we propose a power-efficient scheduling scheme for lightly loaded multicore processors which contain more processing cores than running tasks. The proposed scheme activates a portion of available cores and inactivates the other unused cores in order to save power consumption. The tasks are assigned to the activated cores based on a heuristic mechanism for fast task assignment. Each activated core executes its assigned tasks with the optimal clock frequency which minimizes the power consumption of the tasks while meeting their deadlines. Evaluation

* 동덕여자대학교 컴퓨터학과 (Dept. of Computer Science, Dongduk Women's University)
• 투고일 : 2011. 12. 29, 심사일 : 2012. 1. 28, 게재확정일 : 2012. 2. 9.

shows that the proposed scheme saves up to 78% power consumption of the previous method which activates as many processing cores as possible for the execution of the given tasks.

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

이 논문은 2011년도 동덕여자대학교 연구비 지원에 의해서 수행된 것임.

I. 서 론

최근 들어 프로세서 설계 기술이 발전함에 따라, 하나의 칩 내에 여러 개의 프로세싱 코어(processing core)들을 포함하는 멀티코어 프로세서 사용이 점진적으로 증가하고 있다. 또한 앞으로 멀티코어 프로세서 내의 프로세싱 코어 개수는 프로세서 설계 기술에 비례하여 계속 증가할 것으로 예상된다[1]. 프로세서의 전력소모량을 줄이는 문제는 중요한 문제이다. 특히 배터리의 전원에 의지하여 동작하는 무선 컴퓨터 시스템에서는 프로세서가 소모하는 전력량을 줄이는 것은 매우 중요하다. 프로세서의 전력소모량을 효율적으로 관리하기 위한 고전적인 방법은 작업을 수행하지 않는 유휴 시간(idle time) 동안 프로세서의 전원을 끄는 power-down-when-idle 기법[2]이 있다. 프로세서의 전력소모량을 더 효율적으로 관리하기 위해서 최근 들어 DVFS(dynamic voltage and frequency scaling) 기법[3,4]이 많이 사용되고 있다. DVFS 기법은 프로세서의 작업량에 따라 프로세서에 제공되는 전압 값을 변화시켜서 프로세서의 클락 주파수(clock frequency) 속도와 전력소모량을 동적으로 조절하는 기법이다.

DVFS 기법을 활용하여 프로세서의 전력소모량을 줄이는 작업 스케쥴링 연구들이 많이 진행되고 있다. 본 논문에서는 DVFS 기법을 제공하는 멀티코어 프로세서 상에서 실시간 작업들의 데드라인을 만족하면서 전력소모량을 최소화는 스케쥴링 문제를 다루고자 한다. 기존의 실시간 작업 스케쥴링 연구들[5-10]은 대부분 프로세싱 코어 개수보다 수행 작업들의 개수가 많은 환경에서 전력소모량을 줄이는 방법을 다루었다. 그러나 프로세싱 코어 개수보다 수행 작업 개수가 적은 환경에서 전력소모량을 줄이는 문제는 거의 다루어지지 않았다. 그 이유는 기존의 프로세서 칩은 소수의 프로세싱 코어 개수를 가지고 있어서, 수행 작업 개수보다 프로세싱 코어 개수가 많은 시스템은 실제 환경에서 구현된 적이 없었기 때문이다. 그러나 앞으로 멀티코어 프로세서 칩이 포함하는 프로세싱 코어 개수가 점진적으로 증가하면서, 수행 작업 개수보다 프로세싱 코어 개수가 많은 상용 시스템이 출현할 것으로 예상된다. 본 논문에서는 수행 작업 개수보다 프로세싱 코어 개수가 많은 멀티코어 프세서를 “저부하 멀티코어 프로세서”라고 명한다. 그리고 본 논문에서는 저부하 멀티코어 프로세서상에서 DVFS 기법을 활용하여 주어진 실시간 작업들의 데드라인을 만족하면서 전력소모량을 최소화하는 스케쥴링 문제를 다루고자 한다.

본 논문에서 제시된 스케쥴링 기법은 전력소모량을 줄이기 위해서, 모든 프로세싱 코어들을 사용하지 않고 주어진 작업들의 전체 계산량을 고려하여 일부의 프로세싱 코어들만을 사용하고 나머지 사용하지 않는 코어들의 전원은 소등한다. 제시된 기법에서는 작업량이 매우 낮은 프로세싱 코어 두 개가 소모하는 전력량이, 하나의 프로세싱 코어가 합친 작업량을 수행하면서 소모하는 전력량보다 크다는 사실을 활용하였다. 프로세싱 코어의 유휴시간에 소모하는 전력량이 0에 근접하다면, 각각의 코어들이 작업량을 최대한 분배하여 실행하는 것이 전력소모량을 줄일수 있다. 그러나 프로세싱 코어의 유휴시간에도 누수 전력이 발생하여, 유휴시간에 소모하는 전력량이 무시할 수 없는 수준이다. 따라서 많지 않은 작업량을 다수의 프로세싱 코어들에 분배하여 실행하는 경우보다, 하나의 프로세싱 코어가 작업량을 전부 실행하고 나머지 사용하지 않는 프로세싱 코어의 전원을 차단하는 것이 오히려 전체 프로세서의 전력소모량을 줄인다.

제시된 기법에서는 실제 사용할 최적의 프로세싱 코어 개수를 작업들의 전체 계산량을 기반으로 수학적 분석을 통해서 도출하였다. 각각의 실시간 작업들을 전담하여 수행할 프로세싱 코어를 빠르게 결정하기 위해서, WDF(Worst-Fit Decreasing) 휴리스틱 기법을 사용하여 작업들을 배치하였다. 각각의 프로세싱 코어는 배치된 작업들의 데드라인 모두 만족하면서 전력소모량을 최소화하도록 프로세싱 코어에 적용되는 최적의 클락 주파수를 선택하여 적용하고, 배치된 작업들의 수행순서는 EDF(Earliest Deadline First) 룰에 따라 가장 빠른 데드라인 가진 작업부터 실행한다.

저부하 멀티코어 프로세서에서 실시간 작업들의 데드라인을 만족하면서 전력소모량을 최소화하는 스케쥴링 문제는 NP-hard의 복잡도를 가진다. 본 논문에서 제시된 스케쥴링 기법은 다항식의 계산 복잡도로 동작하면서 최소 전력소모량에 근접하는 준최적 해를 찾도록 설계되었다. 시뮬레이션 실험에서 하나의 작업을 하나의 프로세싱 코어에서 실행하는 기

존의 스케쥴링 기법과 제시된 스케쥴링 기법을 비교하여, 제시된 기법이 전력소모량을 최대 78%까지 줄이는 것을 확인하였다.

본 논문의 나머지 부분은 다음과 같이 구성된다. Ⅱ 장에서는 제시된 스케쥴링 기법을 기존의 스케쥴링 기법들과 비교하여 차별성을 설명한다. Ⅲ 장에서는 제시된 스케쥴링 기법의 동작 과정을 상세히 설명한다. Ⅳ 장에서는 제시된 스케쥴링 기법과 기존의 스케쥴링 기법의 성능 비교 실험 결과를 다룬다. 마지막으로 Ⅴ 장에서는 본 논문의 내용을 요약하고 정리한다.

Ⅱ. 기존 관련 연구

실시간 작업의 데드라인을 만족하면서 전력소모량을 줄이는 고전적인 power-down-when-idle 기법[2]에서는, 실시간 작업을 수행하고 남는 유휴시간 동안 전원을 차단하여 전력소모량을 줄였다. 이러한 방법에서는 프로세서가 항상 정해진 클락 주파수로 동작하는 환경에서 전원을 차단하는 시간을 최대화하면서 오버헤드를 최소화하는 것이 주된 연구 관심사였다. 최근 들어서 고전적인 기법 대신에 DVFS 기법[3,4]에 기반한 저전력 스케쥴링 방법들이 연구되고 있다. DVFS 기법은 프로세서 속도인 클락 주파수가 프로세서에 제공되는 전압 값에 비례하고, 또한 프로세서 전력소모량이 프로세서에 제공되는 전압 값의 다항식에 반비례하는 점을 활용하는 방법이다. 프로세서 전력소모량이 전압 값의 영향을 많이 받기 때문에, 프로세서 유휴시간 동안 전원을 차단하는 것보다 낮은 전압 값을 적용하면, 프로세서 속도인 클락 주파수가 느려져서 유휴시간은 줄어들지만 전력소모량을 훨씬 더 줄일 수 있다.

멀티코어 프로세서(multicore processor) 또는 멀티프로세서(multiprocessor) 상에서 실시간 작업들의 데드라인을 만족시키면서 전력소모량을 줄이기 위한 많은 스케쥴링 방법들이 제안되었다. 그러나 대부분의 기존 연구들[5-9]에서는 프로세싱 코어 개수(또는 프로세서 개수)가 작업들 개수보다 많은 저부하 상태는 고려하지 않고, 프로세싱 코어 개수가 작업들 개수보다 훨씬 적은 고부하 상태만을 다루었다. 따라서 본 논문에서 고려하는 저부하 멀티코어 프로세서상에서 전력소모량을 줄이기 위해 활용률이 낮은 프로세싱 코어의 작업량을 다른 프로세싱 코어에 넘겨주고 사용하지 않는 프로세싱 코어의 전원을 소등하는 방법은 전혀 고려되지 않았다.

소수의 기존 연구[10]에서만 고부하 멀티코어 프로세서가 실행 중에 일시적으로 활용률이 줄어드는 경우를 동적으로 검사하여, 동적 작업 이동(dynamic task migration)을 이용하여 활용률이 낮은 프로세싱 코어의 전원을 소등시켰다. 이 기존 연구에서 다루는 동적 스케쥴링 기법은, 실행 중에 오버헤드를 최소화하면서 스케쥴을 동적으로 빠르게 변경해야 하므로 전력소모량을 충분하게 감소시키지 못하였다. 반면 본 논문에서 제시된 정적 스케쥴링 기법은, 저부하 멀티코어 프로세서에 적합하도록 충분한 검사와 분석을 통해서 동적 작업 이동을 이용하지 않으면서 전력소모량을 최소화하는 정적 스케쥴링 문제를 다루었다. 기존 연구[11]에서는 저부하 멀티코어 프로세서 상에서 하나의 작업을 적정한 수의 프로세싱 코어들에게 병렬로 수행시키고, 사용하지 않는 프로세싱 코어들의 전원을 소등하여 전력소모량을 최소화하는 스케쥴링 문제를 다루었다. 그러나 이 방법은 하나의 실시간 작업만을 위해서 고안된 방법이며, 본 논문에서 제시된 기법은 여러 개의 실시간 작업들의 전력소모량을 최소화하는 스케쥴링 문제를 다루었다.

본 논문의 예비 연구[12]에서는 제시된 기법과 유사한 방법을 활용하는 스케쥴링 기법을 다루었다. 예비 연구에서는 프로세싱 코어들에게 각각 다른 클락 주파수를 동시에 제공할 수 있는 시스템 환경에 적합한 스케쥴링 기법을 설계하였다면, 본 논문의 확장 연구에서는 같은 시간에는 항상 같은 클락 주파수만을 프로세싱 코어들에게 제공하는 시스템 환경에 적합하도록 스케쥴링 기법을 개선하였다. 프로세싱 코어들에게 각각 다른 클락 주파수를 제공할 수 있는 멀티코어 프로세서는 하드웨어 구현이 복잡하고 구현 비용도 증가하기 때문에 프로세싱 코어들에게 같은 시간에 동일한 클락 주파수만을 제공하는 멀티코어 프로세서를 상용 시스템에서 선호한다.

Ⅲ. 제안된 스케쥴링 기법

1. 시스템 모델 및 수식 정의

멀티코어 프로세서에서 사용 가능한 프로세싱 코어 전체 개수를 N으로 나타낸다. 멀티코어 프로세서는 DVFS 기법을 제공하고, 사용되지 않는 프로세싱 코어들의 전원은 별도로 차단하여 전력소모량을 줄인다[2]. DVFS 기법에서 사용 가능한 클락 주파수들은 K개로 한정되어 있다[4]. 사용 가능한 K개의 주파수들을 오름차순으로 정렬하여 $f_1, ..., f_K$으로 나타낸다. 프로세싱 코어들에게 적용되는 클락 주파수는 프로세싱 코어가 단위시간당 처리하는 계산량(실행에 소요되는 클락 사이클 개수)을 나타내고, 클락 주파수가 높아질수록 프로세싱 코어의 실행속도가 높아지고 또한 프로세싱 코어가 소모하는

전력량도 증가한다. 각 클락 주파수 f_k가 단위 시간당 소모하는 전력량을 p_k로 나타내면, $f_i < f_j$ 이면 $p_i < p_j$이다. 프로세싱 코어들에게 적용되는 클락 주파수는 언제든지 변경할 수 있고, 같은 순간에는 프로세싱 코어들에게는 같은 클락 주파수만 제공된다. 프로세싱 코어가 유휴시간에 소모하는 누수전력을 p_0로 나타내고, 유휴시간의 누수전력에 대칭하는 가상의 클락 주파수 f_0를 새롭게 가정하여 사용한다. 유휴시간에는 아무런 작업도 수행하지 않으므로, $f_0 = 0$의 의미를 가진다. 즉, $p_0 < p_1$ 이면서 $f_0 < f_1$이다.

서로 독립적으로 수행되는 주기적 실시간 작업들의 전체 개수를 M으로 나타내고, 이들 M개의 작업들을 $T_1, ..., T_M$로 표시한다. 주기적 실시간 작업들의 도착 주기가 실행을 완료해야 하는 데드라인이 되며, 각각의 작업 T_m의 데드라인을 D_m로 나타낸다. 또한 각각의 작업 T_m이 데드라인까지 수행을 완료해야하는 계산량(수행에 필요한 클락 사이클 개수)을 C_m로 나타낸다. 작업들은 다른 데드라인 D_m과 다른 계산량 C_m을 가진다.

2. 스케쥴링 문제 정형화

제안된 기법에서는 동적 작업 이동을 지원하지 않고, 한번 배치된 프로세싱 코어에서만 계속해서 실행된다. 주어진 작업들의 데드라인을 모두 만족하면서 최소의 전력량만을 소모하는 스케쥴을 '최적 스케쥴'이라 명한다. 또한 작업 T_m의 주어진 계산량 C_m을 데드라인 D_m에 실행완료하기 위해서 단위 시간당 수행해야하는 계산량(클락 사이클 개수)을 '계산량 부하'로 정의하고 L_m로 표기하여 사용한다. 각각의 작업 T_m의 계산량 부하 L_m를 수식으로 표현하면

$$L_m = \frac{C_m}{D_m}$$

이다. 하나의 프로세싱 코어에 여러 개의 실시간 작업들이 배치되었을 때, 배치된 작업들의 계산량 부하 합을 '코어 부하'로 정의하고 CL로 표기하여 사용한다. 하나의 프로세싱 코어에 $T_1, ..., T_x$ 작업들이 배치되었을 때, CL의 수식적 표현은

$$CL = \Sigma_{i=1}^{x} L_i = \Sigma_{i=1}^{x} \frac{C_i}{D_i}$$

이다.

그리고 하나의 프로세싱 코어에게 배치된 실시간 작업들의 데드라인을 모두 만족하는 최소의 클락 주파수를 '코어 최적 주파수'로 정의하고 f_{opt}로 표기한다. 코어 최적 주파수 f_{opt}는 단위 시간당 수행하는 계산량(클락 사이클 개수)을 나타내고, 하나의 프로세싱 코어에 $T_1, ..., T_x$ 작업들이 배치되었을 때의 f_{opt}는 CL과 동일한 값을 가진다($f_{opt} = CL$). 프로세싱 코어는 배치된 작업들중에서 데드라인이 가까운 작업들부터 먼저 실행하고 f_{opt}의 클락 주파수를 사용하면, 배치된 실시간 작업들의 데드라인을 모두 만족하면서 그 프로세싱 코어의 전력소모량를 최소화시킨다.

$f_{opt} \in \{f_1, ..., f_K\}$이면, 일치된 클락 주파수 $f_{opt} = f_i$를 프로세싱 코어에 적용한다. 만약 $f_{opt} \notin \{f_1, ..., f_K\}$이고 $f_i < f_{opt} < f_{i+1}$이면, f_{opt}에 인접한 두 개의 클락 주파수 f_i와 f_{i+1}를 혼합 사용한다. 혼합 사용의 평균값이 f_{opt}가 되도록 하면, 마치 하나의 클락 주파수 f_{opt}를 사용하는 것과 동일한 평균 전력소모량과 실행 속도가 구현된다[11].

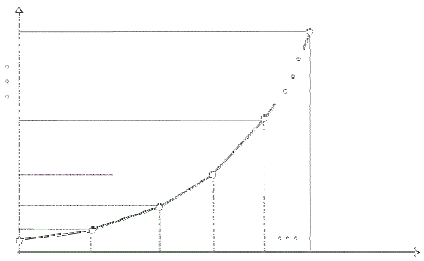

그림 1. 코어 최적 주파수의 단위 시간당 전력소모량
Fig. 1. Power Consumption of Core Optimal Frequency

코어 최적 주파수 f_{opt}는 앞에서도 언급하였듯이 코어 부하 CL로부터 도출된다. 그림 1은 결정된 코어 최적 주파수 f_{opt}를 사용할 때, 하나의 프로세싱 코어가 단위 시간당 소모하는 평균 전력소모량을 보여주고 있다. 코어 최적 주파수가 소모하는 단위 시간당 전력량을 함수 $P(f_{opt})$로 표현한다. $f_{opt} \in \{f_1, ..., f_K\}$이고 $f_{opt} = f_i$인 경우에는

$$P(f_{opt}) = P(f_i) = p_i$$

이다. 또한 $f_{opt} \notin \{f_1, ..., f_K\}$이고 $f_i < f_{opt} < f_{i+1}$인 경우에는, 선형보간법에 근거하여 최적 주파수에 인접한 두 개의 연속된 클락 주파수 f_i와 f_{i+1}의 선형적 중간 위치 지

점 $f_{opt} = f_i + \alpha \cdot (f_{i+1} - f_i)$에 대응하는 두 개의 연속된 단위 시간당 전력소모량 p_i와 p_{i+1}의 선형적 중간 위치 지점의 전력소모량인

$$P(f_{opt}) = P(f_i + \alpha \cdot (f_{i+1} - f_i)) = p_i + \alpha \cdot (p_{i+1} - p_i)$$

의 값을 가진다. 함수 $P(f_{opt})$에 관련된 자세한 내용은 선행연구[11]에서 확인할 수 있다. 이때 유의할 점은 시스템 상에서 사용 가능한 K개의 클락 주파수들과 이들의 단위 시간당 전력소모량 관계를 그림 1과 같은 함수로 표현했을 때, 항상 오목 증가 함수 관계를 따른다는 점이다. 다시 말해서 오목 증가 함수 관계를 위배하는 클락 주파수는 초기 시스템 설정에서 사용 가능한 K개의 클락 주파수 집합에서는 배제된다. 배제 이유는 제외된 클락 주파수는 전력소모량을 최소화하는 최적 스케쥴에서는 항상 사용되지 않는다는 사실은 선행연구[11]에서 증명되었기 때문에, 스케쥴링 고려 대상에 포함하지 않는다.

위에서 언급한 것처럼 작업들이 프로세싱 코어들에게 배치된 이후에, 배치된 작업들의 계산량 부하 합인 CL 값을 이용하면 배치된 작업들의 데드라인을 만족하는 최소 주파수 값인 f_{opt}와 단위 시간당 최소 전력소모량 $P(f_{opt})$은 이미 결정되어 있고 쉽게 구할 수 있다. 본 논문에서 고려하는 시스템 환경은 Ⅲ장 1절에서 언급하였듯이, 클락 주파수는 다른 시점에는 다른 클락 주파수 값을 적용할 수 있어도 같은 시점에는 동일한 클락 주파수가 프로세싱 코어들에게는 적용된다.

따라서 프로세싱 코어들에게 배치된 모든 작업들의 데드라인을 만족시키는 최소 클락 주파수는, 프로세싱 코어들의 최적 주파수 $f_{opt} = CL$ 값들 중에서 최대 최적 주파수이다. N개의 코어들 중에서 일부 n개의 코어만을 사용하여 주어진 M개의 작업들을 배치한 이후의 프로세싱 코어들의 코어 부하 값을 $CL_1, ..., CL_n$로 표기한다. 이때 n개의 코어들에 동일하게 적용되는 최대 최적 클락 주파수는 다음과 같다.

$$f_{opt}^{\max} = \max(CL_1, ..., CL_n)$$

또한 프로세서 전체의 단위 시간당 전력소모량은 다음과 같이 결정된다.

$$n \cdot P(f_{opt}^{\max}) = n \cdot P(\max(CL_1, ..., CL_n)) \quad (1)$$

주어진 작업들이 프로세싱 코어들에게 배치된 이후에 전체 프로세서가 소모하는 최소 전력소모량이 수식 (1)과 같이 이미 결정되어 있다면, 실시간 작업들의 전력소모량에 영향을 미치는 남겨진 문제는 주어진 M개의 작업들을 N개의 프로세싱 코어들에게 어떻게 배치할 것인지를 결정하는 문제이다. 남겨진 작업배치 문제를 해결하기 위해서 제안된 작업배치 기법은 다음 3절에서 다룬다.

3. 제시된 작업배치(task assignment) 기법

프로세서 전체가 소모하는 전력량을 최소화하기 위해서는, 수식 (1)의 값이 최소화되도록 주어진 M개의 작업들을 N개의 프로세싱 코어들에게 배치하여야 한다. 그러나 수식 (1)이 최소화되도록 작업들을 코어들에게 배치하는 문제는 NP-hard 문제에 속한다. 그 이유는 본 절에서 다루는 작업배치 문제의 일부 특수 경우에 해당하는 하위 문제인, 고정된 개수의 프로세싱 코어들을 모두 사용하여 전력소모량을 최소화하는 문제가 NP-hard 임이 이미 알려져 있기 때문이다[6].

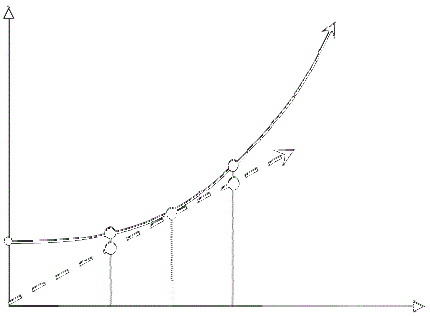

그림 2. 최적의 코어 개수 n_{opt} 결정 과정
Fig. 2. Determination of the Optimal Number n_{opt} of Cores

제시된 배치 기법에서는 먼저 N개의 프로세싱 코어들 중에서 실제로 사용할 프로세싱 코어 개수 η을 다음과 같이 결정한다. 먼저 그림 2와 같이 주어진 $P(f_{opt})$ 함수를 이용하여, 정점을 지나는 일차 직선 함수 $\beta \cdot f_{opt}$가 $P(f_{opt}) \geq \beta \cdot f_{opt}$ 관계를 만족하면서 접하도록 β 값을 찾는다. 또한 $P(f_{opt}) = \beta \cdot f_{opt}$ 관계를 만족하는 f_{opt} 지점의 값을 X라고 표기하고, 전체 작업들의 계산량 부하 합 $\sum L$을 X로 나눈 값을 $n_{opt} = \frac{\sum L}{X}$라고 표기한다. 아래의 정리 1에 근거하여 전체의 작업들을 n_{opt}개의 프로세싱 코어들에게 완전하게 균등 분배하는 경우가, 전체 프로세서의 전력소모량을 최소화하는 경우이다. 단 정리 1에서 n_{opt}는 정수

가 아닌 유리수이다.

- 정리(theorem) 1: 전체의 작업들을 n_{opt}개의 프로세싱 코어들에 완전하게 균등 분배하는 경우가 전체 프로세서의 전력소모량을 최소화하는 경우이다.
- 증명: 작업 배치에 사용할 프로세싱 코어의 개수가 n개로 결정되었다고 가정하면, 전체 전력소모량을 최소화하도록 작업들을 배치하는 경우는 작업들의 전체 계산량 부하 합 $\sum L$을 완전 균등하게 분배하여 각각의 프로세싱 코어들이 $CL = \frac{\sum L}{n}$의 값을 가지는 경우이다. 그 이유는 그림 1과 같이 $P(f_{opt})$가 오목 증가 함수이므로,

$$2 \cdot P(\frac{\sum L}{n}) \le P(\frac{\sum L}{n} - \epsilon) + P(\frac{\sum L}{n} + \epsilon)$$

이다. 위의 수식이 의미하는 것은 코어 부하 값을 균등하게 분배하는 경우의 전력소모량 $2 \cdot P(\frac{\sum L}{n})$이, 코어 부하 값을 불균등하게 분배하는 경우의 전력소모량 $(P(\frac{\sum L}{n} - \epsilon) + P(\frac{\sum L}{n} + \epsilon))$보다 항상 작다는 사실이다. 따라서 전체 작업들의 계산 부하 합 $\sum L$을 완전 균등하게 분배하는 경우가, 정해진 n개의 프로세싱 코어들에게 전력소모량을 최소화하도록 주어진 작업들을 배치하는 경우이다.

다음으로 n_{opt}개의 프로세싱 코어들에게 전체 계산량 부하 합 $\sum L$을 완전 균등하게 분배하여 전력소모량을 최소화하는 경우와, n_{opt}보다 많은 $(n_{opt} + \delta)$개의 코어들에게 전체 계산량 부하 합 $\sum L$을 완전 균등하게 분배하여 전력소모량을 최소화하는 경우를 비교한다. n_{opt}개의 프로세싱 코어들에게 $\sum L$을 완전 균등하게 분배하면, 그림 2와 같이 각 프로세싱 코어는 $CL = X = \frac{\sum L}{n_{opt}}$의 값을 가진다. 이 경우의 전체 단위 시간당 전력소모량은 $n_{opt} \cdot P(X)$가 된다. $(n_{opt} + \delta)$개의 코어들에게 $\sum L$을 완전 균등하게 분배하면, 각 프로세싱 코어는 $\sum L = (n_{opt} + \delta) \cdot (X - \epsilon) = n_{opt} \cdot X$ 일때 $CL = \frac{\sum L}{n_{opt} + \delta} = X - \epsilon$의 값을 가진다. 또한 그림 2가 보여주듯이

$$P(X - \epsilon) \cdot (n_{opt} + \delta) > \beta \cdot (X - \epsilon) \cdot (n_{opt} + \delta)$$

이고,

$$\beta \cdot (X - \epsilon) \cdot (n_{opt} + \delta) = \beta \cdot X \cdot n_{opt} = P(X) \cdot n_{opt}$$

이다. 따라서 $P(X - \epsilon) \cdot (n_{opt} + \delta) > P(X) \cdot n_{opt}$이다. 이 수식은 n_{opt}보다 많은 $(n_{opt} + \delta)$개의 코어들에게 $\sum L$을 완전 균등하게 분배하는 경우의 전력소모량 $P(X - \epsilon) \cdot (n_{opt} + \delta)$이, n_{opt}개의 코어들에게 $\sum L$을 완전 균등하게 분배하는 경우의 전력소모량 $P(X) \cdot n_{opt}$ 보다 항상 큼을 의미한다.

마지막으로 n_{opt}개의 프로세싱 코어들에게 $\sum L$을 완전 균등하게 분배하여 전력소모량을 최소화하는 경우와, n_{opt}보다 적은 $(n_{opt} - \delta)$개의 코어들에게 $\sum L$을 완전 균등하게 분배하여 전력소모량을 최소화하는 경우를 비교한다. 비슷한 논리에 의해서 $(n_{opt} - \delta)$개의 코어들에게 $\sum L$을 완전 균등하게 분배하면, 각 프로세싱 코어는 $CL = \frac{\sum L}{n_{opt} - \delta} = X + \epsilon$의 값을 가진다. 이때 $\sum L = X \cdot n_{opt} = (X + \epsilon) \cdot (n_{opt} - \delta)$이며, 그림 2가 보여주듯이

$$P(X + \epsilon) \cdot (n_{opt} - \delta) > \beta \cdot (X + \epsilon) \cdot (n_{opt} - \delta)$$

이고,

$$\beta \cdot (X + \epsilon) \cdot (n_{opt} - \delta) = \beta \cdot X \cdot n_{opt} = P(X) \cdot n_{opt}$$

이다. 즉 $P(X + \epsilon) \cdot (n_{opt} - \delta) > P(X) \cdot n_{opt}$이며, 이 수식은 n_{opt}보다 적은 $(n_{opt} - \delta)$개의 코어들에게 $\sum L$을 완전 균등하게 분배하는 경우의 전력소모량 $P(X + \epsilon) \cdot (n_{opt} - \delta)$이, n_{opt}개의 코어들에게 $\sum L$을 완전 균등하게 분배하는 경우의 전력소모량 $P(X) \cdot n_{opt}$ 보다 항상 큼을 의미한다.

위의 논리들에 근거하여 전체 M개의 작업들을 n_{opt}개의 프로세싱 코어들에 완전하게 균등 분배하는 경우가 전체 프로세서의 전력소모량을 최소화하는 경우이다.

제시된 배치 기법은 위에서 도출된 유리수 n_{opt} 값을 이용하여 실제로 사용할 코어 개수를 나타내는 자연수 η를 다음과 같이 결정한다. 먼저 n_{opt} 값이 $n_{opt} \le N$인 정수이거나 $n_{opt} > N$이면, n_{opt} 값을 η에 그대로 적용한다. 그리고 n_{opt} 값이 정수가 아니고 $n < n_{opt} < (n+1)$이라면, n_{opt} 값에 근접된 두 개의 연속된 정수 n과 $(n+1)$ 중에서 하나를 선

택하여 η에 적용한다. n개의 프로세싱 코어들을 사용할 경우의 최소 전력소모량과 $(n+1)$개의 프로세싱 코어들을 사용할 경우의 최소 전력소모량을 비교하여, 더 적은 값을 가지는 프로세싱 코어 개수를 η에 적용한다. η개의 프로세싱 코어들의 전력소모량이 n_{opt}개의 프로세싱 코어들의 전력소모량에 가장 근사치 값을 가지므로, 자연수의 프로세싱 코어들 중에서 η개의 프로세싱 코어들을 사용할 때 최저 전력소모량을 가진다.

다음으로는 주어진 M개의 작업들을 η개의 프로세싱 코어들에게 어떻게 배치할 것인지를 결정하여야 한다. 앞에서 언급하였듯이, 고정된 개수의 프로세싱 코어들에게 주어진 작업들을 최대한 균등하게 배치하는 문제는 NP-hard 문제이다[6]. 최적의 해법을 찾기 위해서는 모든 경우들에 대해서 전수 검사를 해야하고, 전수 검사는 많은 시간을 요구한다. 따라서 본 논문에서는 전수검사 대신 휴리스틱(heuristic) 기법을 이용하여 빠른 시간내에 준최적해를 찾는 검사 방법을 이용한다.

1단계: 전체 계산량 부하 합 $\sum L$을 이용하여 N개의 코어들 중에서 실제로 사용할 코어 개수 η을 결정한다.
2단계: 주어진 M개의 작업들을 계산량 부하 값에 따라 내림차순으로 정렬한다.
3단계: M개의 작업들을 정렬된 순서에 따라 하나씩 η개 프로세싱 코어들 중에서 최소 코어 부하 CL 값을 가진 코어에 배치하는 과정을 반복한다.
4단계: 프로세싱 코어들 중에서 가장 큰 코어 부하 CL 값을 이용하여 η개 프로세싱 코어들에게 공통으로 적용할 최적 코어 주파수 $f_{opt}^{\max}$를 결정한다.
5단계: η개 프로세싱 코어들은 최적 코어 주파수 $f_{opt}^{\max}$를 사용하여 각각의 프로세싱 코어들에게 배치된 작업들을 데드라인이 가까운 작업들부터 순차적으로 실행한다.

그림 3. 제시된 스케줄링 기법의 의사 코드
Fig. 3. Pseudo Code of Proposed Scheduling Scheme

많이 사용하는 휴리스틱 기법들 중에서 WFD(Worst-Fit Decreasing heuristic) 기법이 저전력 작업배치 문제에 가장 우수한 성능을 보임이 이미 밝혀졌다[6]. 따라서 본 논문에서는 WFD 기법에 근거하여 M개의 작업들을 η개의 프로세싱 코어들에게 배치한다. WFD 기법은 주어진 작업들을 계산량 부하 값을 기준으로 내림차순으로 정렬하여 계산량 부하 L 값이 큰 작업부터 배치하고, 각각의 작업들을 배치할 때 η개의 프로세싱 코어들 중에서 코어 부하 CL 값이 가장 작은 코어에게 해당 작업을 배치하는 방법이다. 그림 3은 제시된 스케줄링 기법의 전체 의사 코드(pseudo code)를 보여주고 있다.

제시된 스케줄링 기법은 $O(M \cdot \log M + M \cdot N + N \cdot K)$의 계산 복잡도(computational complexity)를 가진다. 1단계의 X 값을 찾는 계산 복잡도는 $O(K)$이고, η 값을 찾는 계산 복잡도는 $O(M)$이다. 2단계의 작업 정렬 복잡도는 $O(M \cdot \log M)$이다. 3단계의 M개의 작업들을 η개의 프로세싱 코어들에 배치하는 연산의 복잡도는 $O(M \cdot N)$이다. 4단계의 $f_{opt}^{\max}$ 값을 찾는 계산 복잡도는 $O(N \cdot K)$이다. 5단계는 각 코어에서 병렬로 수행되므로 배치된 작업들의 수행 순서를 결정하는 연산의 계산 복잡도는 $O(M \cdot \log M)$이다.

Ⅳ. 성능 평가

성능 평가를 위하여 제시된 기법과 기존 방법[6]의 전력소모량을 비교하였다. 기존 방법에서는 시스템 상의 N개 프로세싱 코어들을 모두 사용하고 사용하지 않는 코어들의 전원을 끄지 않았지만, 본 실험에서는 최대한 엄격한 조건에서 성능 비교가 되도록 기존 방법을 변경하여 사용하지 않는 프로세싱 코어의 전원을 끄도록 수정하였다. 또한 기존 방법은 DVFS 기법이 무한개으로 연속된 클락 주파수들을 사용할 수 있다는 비현실적 DVFS 기법을 가정하였지만, 본 실험에서는 유한개의 이산적 클락 주파수 중에서 연속된 두 개의 클락 주파수를 혼합 사용하여 최적 코어 주파수 $f_{opt}^{\max}$ 값을 생성하도록 기존 방법을 변경하였다. 평가 지표로는 '기존 방법이 소모하는 단위 시간당 전력량' 대비 '제시된 방법이 소모하는 단위 시간당 전력량' 비율을 '상대적 전력소모량'이라고 정의하고, 이를 제시된 방법의 성능 지표로 사용하였다.

표 1. 프로세서 모델
Table 1. Processor Model

k	0	1	2	3	4	5
f_k (MHz)	0	150	400	600	800	1000
p_k (mW)	40	80	170	400	900	1600

실용적인 DVFS 프로세서 실험을 위해서, 실제로 널리 사용되는 인텔 XScale 프로세서[4]의 DVFS 기법 데이터를 사용하였다. 표 1은 인텔 XScale 프로세서에서 제공하는 5개의 클락 주파수 값들과 단위 시간당 전력소모량 값을 보여주고 있다. p_0는 프로세서의 유휴 상태에서의 누수 전력을 나타내고, f_0는 Ⅲ장 1절에서 설명하였듯이 가상적으로 정의하

여 추가한 유휴 상태에서의 클락 주파수 값이다. 6개의 주파수 값들의 단위 시간당 전력소모량 함수 $P(f_{opt})$는 그림 1과 같은 오목 증가 함수이다. 또한 실제로 사용할 코어 개수 η을 결정하는데 사용되는 그림 2의 X 값은 f_2 = 400(사이클/초)이다.

모의실험에는 32개의 주기적 실시간 작업들을 사용하고, 각 작업들은 다른 계산량 부하 값을 가지도록 생성되었다. 작업들의 계산량 부하 값들은 정규 분포 또는 지수 분포를 따르도록 인위적으로 생성하였다. 평가 신뢰도를 높이기 위하여 32개의 작업들을 가지는 집합을 십만번 생성하여 평균 값을 실험 결과 지표로 사용하였다.

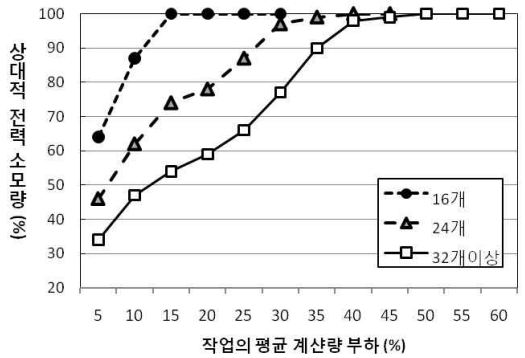

그림 4. 정규 분포상에서 상대적 전력소모량
Fig. 4. Normalized Power Consumption with Normal Distribution

전체 프로세싱 코어 개수 N의 값과 작업들의 평균 계산량 부하 비율의 변화에 따른 제시된 방법의 상대적 전력소모량을 측정하였다. 작업의 '계산량 부하 비율'은 최대 계산량 부하 f_5=1000(사이클/초) 대비 작업의 계산량 부하 L의 비율을 나타낸다. 그림 4는 작업들의 계산량 부하가 정규 분포를 이용하여 생성되었을 때의 결과를 보여주고 있다. 작업들의 평균 계산량 부하 값이 작을수록, 제시된 방법이 기존 방법보다 전력소모량을 더 많이 줄인다. 또한 전체 프로세싱 코어 개수 N의 값이 커질수록 전력소모량을 더 많이 줄임을 확인할 수 있다. $N \geq 32$이고 작업의 평균 계산량 부하가 5%일 때, 상대적 전력소모량은 약 34%이다. 주어진 작업의 개수가 32개이므로, N의 값이 32개 이상이어도 기존 방법과 제시된 방법은 32개를 초과하는 코어들을 사용하지 않는다. 작업 개수보다 전체 코어 개수가 많은 저부하 멀티코어 프로세서에 적합하도록 제시된 방법이 설계되었지만, 작업들의 평균 계산량 부하 값이 작으면 작업 개수보다 코어 개수가 적은 환경에서도 전력소모량을 감소시키는 것을 확인할 수 있다.

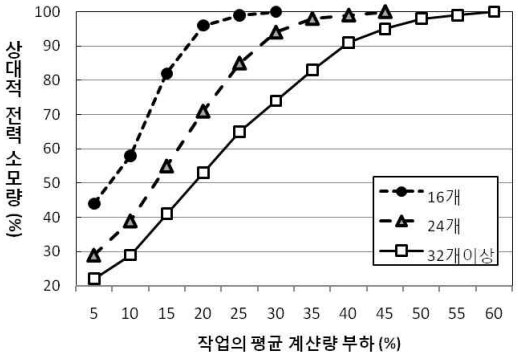

그림 5. 지수 분포상에서 상대적 전력소모량
Fig. 5. Normalized Power Consumption with Exponential Distribution

그림 5는 작업들의 계산량 부하 값들이 지수 분포를 따를 때의 상대적 전력소모량을 보여주고 있다. 그림 4와 유사하게 작업들의 평균 계산량 부하 값이 작을수록 그리고 N의 값이 커질수록 전력소모량을 더 많이 줄임을 확인할 수 있다. $N \geq 32$이고 작업의 평균 계산량 부하가 5%일 때, 상대적 전력소모량은 약 22%이다.

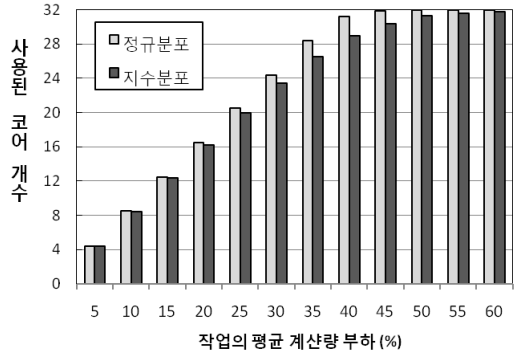

그림 6. 32개 이상의 코어들 중에서 실제 사용된 개수
Fig. 6. Number of Used Cores amongst 32 Cores

그림 6은 $N \geq 32$일 때, 제시된 기법이 실제로 사용하는 프로세싱 코어 개수를 나타내고 있다. 기존 방법은 $N \geq 32$일 때 실제로 사용하는 프로세싱 코어 개수는 항상 작업 개수와 동일한 32개이다. 그림 6에서 보여주듯이 제시된 기법은 작업들의 평균 계산량 부하 값이 작을수록 적은 개수의 프로세싱 코어들만을 사용한다. 제시된 기법은 적은 수의 코어들을 사용할수록 낮은 상대적 전력소모량을 가진다.

v. 결론

본 논문에서는 저부하 멀티코어 프로세서에 적합한 실시간 작업용 저전력 스케줄링 기법을 제안하였다. 제시된 기법은 시스템상의 모든 프로세싱 코어들을 사용하지 않고, 주어진 작업들의 전체 계산량을 고려하여 일부의 프로세싱 코어들만을 사용하고 나머지 사용하지 않는 코어들의 전원을 소등하여 전력소모량을 줄였다. 또한 본 논문에서는 주어진 작업들의 전체 계산량 값을 기준으로 실제 사용할 최적의 프로세싱 코어 개수를 수학적 분석으로 도출하는 방법을 제시하였다. 제시된 스케줄링 기법과 기존의 프로세싱 코어들을 모두 사용하는 방법을 비교하는 실험에서 최대 78%까지 전력소모량을 감소시킴을 확인하였다.

향후 후속 연구에서는 작업들의 계산량이 사전에 고정되어 있지 않고 불확실한 경우에 전력소모량을 확률적으로 최소화하는 스케줄링 기법을 연구할 예정이다.

참고문헌

[1] Semiconductor Industry Association (SIA), International Technology Roadmap for Semiconductors: 2005 Edition, http://www.itrs.net.

[2] 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.

[3] 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.

[4] R. Xu, C. Xi, R. Melhem, and D. Moss, ''Practical PACE for embedded systems,'' ACM Int'l Conf. Embedded Software, 2005, pp. 54-63.

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

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

[7] D. Zhu, R. Melhem, and B. Childers, ''Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems,'' IEEE Trans. Parallel Distrib. Syst., vol. 14, no. 7, pp. 686-700, 2003.

[8] A. Andrei, P. Eles, Z. Peng, M. T. Schmitz, and B. Hashimi, ''Energy optimization of multiprocessor systems on chip by voltage selection,'' IEEE Trans. VLSI Syst., vol. 15, no. 3, pp. 262-275, 2007.

[9] H. Kim, H. Hong, H.-S. Kim, J.-H. Ahn, and S. Kang, ''Total energy minimization of real-time tasks in an on-chip multiprocessor using dynamic voltage scaling efficiency metric,'' IEEE Trans. CAD IC Syst., vol. 27, no. 11, pp. 2088-2092, 2008.

[10] 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.

[11] H. Pack, J. Yeo and W. Lee, ''Energy-efficient multi-core scheduling for real-time video processing,'' Journal of the Korea Society of Computer and Information, vol. 16, no. 6, pp. 11-20, 2011.

[12] 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, 2009, pp. 216-223.

저 자 소 개

이 완 연

2000: POSTECH 공학박사
2000-2003: LG전자 선임연구원
2003-2011: 한림대학교 컴퓨터공학과
현재: 동덕여자대학교 컴퓨터학과 부교수
관심분야: 내장형 컴퓨터, 시스템소프트웨어, 모바일 컴퓨팅