Vol.4, No.6 pp.177~184 pISSN: 2287-5891 http://dx.doi.org/10.3745/KTCCS.2015.4.6.177

Minimum-Power Scheduling of Real-Time Parallel Tasks based on Load Balancing for Frequency-Sharing Multicore Processors

Wan Yeon Lee†

ABSTRACT

This paper proposes a minimum-power scheduling scheme of real-time parallel tasks while meeting deadlines of the real-time tasks on DVFS-enabled multicore processors. The proposed scheme first finds a floating number of processing cores to each task so that the computation load of all processing cores would be equalized. Next the scheme translates the found floating number of cores into a natural number of cores while maintaining the computation load of all cores unchanged, and allocates the translated natural number of cores to the execution of each task. The scheme is designed to minimize the power consumption of the frequency-sharing multicore processor operating with the same processing speed at an instant time. Evaluation shows that the scheme saves up to 38% power consumption of the previous method.

Keywords : Multicore Processor, Real-Time Task, Power-Efficient Scheduling, DVFS

주파수 공유형 멀티코어 프로세서를 위한 부하균등화에 기반한 실시간 병렬 작업들의 최소 전력 스케줄링

이 완 연†

요 약

본 논문에서는 DVFS 기반의 멀티코어 프로세서상에서 실시간 병렬 작업들의 마감시한을 만족하면서 전력 소모량을 최소화시키는 스케줄링 기법을 제안하였다. 제안된 기법에서는 먼저 모든 프로세싱 코어들의 계산부하가 동일해지도록 각 작업에게 할당될 프로세싱 코어들의 실수 개수를 찾는다. 그리고 프로세싱 코어들의 계산부하가 동일하도록 유지하면서 찾은 실수 개수의 프로세싱 코어들을 자연수 개수의 프로세싱 코어들로 변환시켜 각 작업들의 실행에 할당한다. 제안된 방법은 단일 시점에 동일한 속도로 동작하는 주파수 공유형 멀티코어 프로세서의 전력 소모량을 최소화하도록 설계되었다. 성능 평가 실험에서 제안된 기법이 기존 방법의 전력 소모량을 최대 38%까지 감소시킴을 확인하였다.

키워드 : 멀티코어 프로세서, 실시간 작업, 저전력 스케줄링, DVFS

1. 서 론

배터리의 한정된 전원에 의존하여 동작하는 무선 컴퓨팅 기기들을 위해서 많은 저전력 프로세서 스케줄링 기법들이 연구되어왔다. 프로세서의 전력 소모량을 줄이는 방법으로 초창기에는 작업을 수행하지 않는 유휴 시간(idle time) 동안 프로세서의 전원을 차단하는 power-down-when-idle 기법[1]이 활용되었다. 최근 들어 프로세서의 전력 소모 효율성을 개선하기 위해서 DVFS(Dynamic Voltage and Frequency Scaling) 기법[2, 3]이 활용되고 있는데, DVFS 기법은 프로세서의 부하에 따라 프로세서에 입력되는 클락 주파수(clock frequency)를 변화시켜서 프로세서의 연산 속도와 전력 소모량을 동적으로 조절하는 기법이다. 클락 주파수를 감소시키면 프로세서의 연산 속도가 감소되지만 더불어 전력 소모량도 같이 줄어든다. 유휴 시간 동안 프로세서의 전원을 차단하는 power-down-when-idle 기법보다 클락 주파수값을 조절하는 DVFS 기법이 실시간 작업의 전력 소모량 효율성을 개선시키는 데 유리하다는 사실이 밝혀졌다[2, 3].

DVFS 기법을 제공하는 멀티코어 프로세서는 단일 순간에 코어들이 동일한 속도로 동작하는 주파수 공유형 멀티코어 프로세서와 단일 순간에 코어들이 상이한 속도로 동작할 수 있는 주파수 분리형 멀티코어 프로세서로 나뉜다. 본 논

※ 이 논문은 2014년도 동덕여자대학교 학술연구비 지원에 의해서 수행됨.
† 정 회 원 : 동덕여자대학교 컴퓨터학과 부교수
Manuscript Received : December 19, 2014
First Revision : February 23, 2015
Second Revision : March 16, 2015
Accepted : March 16, 2015
* Corresponding Author : Wan Yeon Lee(wanlee@dongduk.ac.kr)

문에서는 주파수 공유형 멀티코어 프로세서상에서 병렬 수행과 부하균등화를 활용하여 실시간 작업들의 마감시한을 만족하면서 모든 프로세싱 코어들의 전력 소모량을 최소화시키는 스케줄링 기법을 제안하였다. 주파수 공유형 멀티코어 프로세서에서는 최대 계산부하를 가진 프로세싱 코어에 의해서 전체 전력 소모량이 결정된다. 따라서 프로세싱 코어들의 계산부하들을 완전 균등화하여 코어들의 최대 계산부하를 낮춤으로써 전체 프로세싱 코어들의 전력 소모량을 최소화하였다. 또한 각 작업의 실행에 병렬 수행을 적용하면 병렬 수행 오버헤드로 인해서 병렬 수행에 할당된 코어 개수에 비례하여 연산량도 같이 증가한다. 따라서 제안된 기법은 작업들의 전체 연산량을 최소화하면서 모든 코어들의 계산부하를 균등화시켰다.

제안된 기법은 먼저 전체 작업 연산량을 최소로 유지하면서 모든 프로세싱 코어들의 계산부하가 균등화되도록 각 작업에 할당될 프로세싱 코어들의 실수 개수를 찾는다. 그리고 프로세싱 코어들의 계산부하를 동일하게 유지하면서 찾아진 실수 개수의 프로세싱 코어들을 자연수 개수로 변환시켜 각 작업에 할당한다. 마지막으로 프로세싱 코어들의 동일한 계산부하를 만족시키는 최소의 연산 속도를 모든 프로세싱 코어들에게 적용시키고, EDF(Earliest Deadline First) 규칙에 따라 작업들의 실행 순서를 결정한다. 본 논문에서 제안한 방법은 주파수 공유형 멀티코어 프로세서 모델에 적합하도록 DVFS 기법, 병렬 수행, 그리고 부하균등화를 모두 활용하여 전력 소모 감소 효율이 극대화되도록 설계되었다. 성능 평가 실험을 통하여 프로세서 구동을 위한 최소 기반 전력을 제외하고 기존 스케줄링 기법이 소모하는 전력량을 제안된 기법은 최대 38%까지 감소시킴을 확인하였다.

멀티코어 프로세서상에서 실시간 작업들의 전력 소모량을 줄이기 위해서 DVFS 기법을 활용한 스케줄링 연구들이 많이 진행되어왔다. 그러나 대부분의 기존 스케줄링 기법들[4-8]은 병렬 수행을 고려하지 않고, 단일 작업은 하나의 프로세싱 코어가 수행한다는 전제하에 설계되었다. 단일 작업에 다수의 코어들을 할당하여 동시에 수행하는 병렬 수행을 적용하면, 작업의 수행 시간을 단축시킬 수 있고 이로 인해서 프로세싱 코어들에 낮은 연산 속도를 적용하여도 마감시한을 만족시킬 수 있게 된다. 전력 소모량이 연산 속도의 약 세제곱에 비례하므로, 병렬 수행에 할당된 프로세싱 코어들의 개수가 증가하여도 전체 프로세싱 코어들이 소모하는 전력량은 대폭 감소된다[9].

최근 들어 병렬 수행을 고려한 저전력 스케줄링 기법들[9-11]이 다루어졌지만, 주파수 공유형 멀티코어 프로세서에 적합한 다수의 실시간 작업들을 위한 저전력 스케줄링 기법은 다루어지지 않았다. 기존 연구[9]에서는 단일 병렬 실시간 작업을 위한 저전력 스케줄링 기법만을 다루었고, 기존 연구[10]에서도 실행 순서에 선후 관계 제약이 존재하는 부분작업들로 구성된 단일 병렬 작업의 전력 소모를 줄이는 문제만을 다루었다. 기존 연구[11]에서는 다수의 실시간 병렬 작업들을 위한 저전력 스케줄링 문제를 다루었지만, 프로세싱 코어들이 단일 시점에 다른 속도로 동작할 수 있는 주파수 분리형 멀티코어 프로세서 모델에 적합하도록 설계되어 주파수 공유형 멀티코어 프로세서 모델에서는 전력 소모 효율이 떨어지는 단점이 있다. 기존 방법[11]에서 일부 코어에 계산부하가 편중되는 부하 불균형이 발생하면, 주파수 분리형 멀티코어에서는 계산부하가 편중된 코어들의 전력 소모량만 증가하지만 주파수 공유형 멀티코어에서는 전체 코어들의 전력 소모량이 모두 증가하게 된다. 제시된 방법은 부하균등화를 적용하여 코어들의 최대 계산부하를 최소화시켜 전체 프로세서의 전력 소모량을 최소화시켰다. 프로세싱 코어들에게 단일 시점에 다른 속도 동작을 허용하는 주파수 분리형 멀티코어 프로세서는 하드웨어 구현이 복잡하고 구현 비용도 증가하기 때문에 대부분의 상용 멀티코어 프로세서 칩에는 주파수 공유형 구조가 적용된다[14].

본 논문에서 제시된 기법은 병렬 수행에 할당된 코어들의 개수를 실행 중에 변경할 수 있는 실시간 멀티미디어 작업 모델을 대상으로 설계되었고, 병렬 수행에 할당된 코어들의 개수를 실행 중에 변경하는 부하균등화를 적용하면 전력 소모량을 최소화시키는 스케줄을 다항식 복잡도(polynomial complexity)를 가지는 알고리즘을 통해서 찾을 수 있음을 보인다. 반면 병렬 수행에 할당된 코어들의 개수를 실행 중에 변경할 수 없는 작업 모델을 고려한 기존 연구들[9-11]에서는, 코어들의 계산부하를 완전 균등화하는 것이 불가능하고 또한 코어들의 최대 부하값이 최소화되는 스케줄을 찾는 문제가 NP-hard의 복잡도를 가짐이 입증되었다[11].

본 논문의 나머지 부분은 다음과 같이 구성된다. 2절에서는 제안된 기법에 적용된 DVFS 기반의 멀티코어 프로세서 모델과 실시간 병렬 작업 모델에 대해서 설명한다. 3절에서는 제시된 스케줄링 기법의 동작 과정을 상세히 설명하고, 4절에서는 제시된 기법과 기존 스케줄링 기법의 성능 비교 실험을 다룬다. 마지막으로 5절에서는 본 논문의 내용을 정리한다.

2. 시스템 환경

2.1 DVFS 기반 멀티코어 프로세서 모델

멀티코어 프로세서 내의 전체 프로세싱 코어 개수를 N으로 나타낸다. DVFS 기반 멀티코어 프로세서에서 프로세싱 코어들에게 적용되는 클락 주파수는 단위시간당 처리하는 계산량(클락 사이클 개수)을 의미하고, 따라서 클락 주파수가 증가하면 코어의 연산 속도가 증가하며, 코어가 소모하는 전력량도 같이 증가한다. 주파수 공유형 멀티코어 프로세서에서 프로세싱 코어들의 연산 속도는 시간에 따라 변경할 수 있지만, 단일 순간에는 프로세싱 코어들의 동작 속도는 항상 동일하다[12]. 최대 코어 속도를 $S_{\max}$로 표기하고, 표현의 간결성을 위해 $S_{\max}=1.0$이 되도록 최대 코어 속도를 정규화한다. 그러면 최대 속도보다 감속된 코어 속도 S는 $0 \le S < S_{\max}$의 관계식을 만족한다. 프로세싱 코어의 전

력 소모량은 코어 속도의 약 세제곱에 비례하므로[2-4], 프로세싱 코어가 S 속도로 동작할 때의 단일 시간당 소모 전력량을 $F(S)$의 함수로 표기하고 다음과 같이 계산된다.

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

여기서 α는 프로세서 하드웨어 특성 상수이고, P_l는 프로세서 구동을 위한 최소 기반 전력의 단위 시간당 소모량을 나타낸다. P_l는 코어의 전원을 차단하는 경우에만 절약되는 전력 소모량으로, 모든 코어들을 사용하는 DVFS 기법에서는 전력 소모 절약 대상에서 제외된다. 성능 평가 실험에서 비교 대상 기존 방법과 제시된 기법의 P_l값은 동일하고 프로세서 특성에 의해서 결정되므로, P_l은 제외하고 $\alpha \cdot S^3$의 전력 소모량만을 비교하였다.

2.2 실시간 병렬 작업 모델

서로 독립적으로 수행되는 주기적 실시간 작업들의 개수를 M으로 나타내고, 이들 M개의 작업들을 $T_1, ..., T_M$으로 표시한다. 주기적 실시간 작업들의 도착 주기가 실행을 완료해야 하는 마감시한이 되며, 각 T_m의 마감시한을 D_m으로 나타낸다. 또한 각 T_m이 마감시한까지 수행을 완료해야 하는 계산량을 C_m으로 나타낸다. 각 작업들은 다른 마감시한 D_m과 다른 계산량 C_m을 가질 수 있다. 각 T_m을 단일 코어가 수행할 때의 실행 시간과 비교하여 n_m개의 코어들을 T_m의 실행에 할당하여 병렬 수행을 적용할 때의 실행 시간 향상 비율을 '병렬 수행 속도 증가'라 명하고 $P_m[n_m]$으로 나타낸다. 이때 $P_m[1]=1$이며, $n_x < n_y$이면 $P_m[n_x] \le P_m[n_y]$이다. 그리고 병렬 수행에 필요한 추가 연산 오버헤드(부분작업 초기 분배, 부분 작업 수행 결과 최종 수집 등)[13]로 인해서 $n_m \ge 2$에 대해서 $P_m[n_m] \le n_m$이다. 병렬 작업의 $P_m[n_m]$값은 프로파일링 기법에 의해서 계측될 수 있으므로[11], 모든 작업들의 $P_m[n_m]$값은 사전에 주어진다고 가정한다.

n_m개의 코어들을 이용하여 T_m을 실행할 때, 주어진 계산량 C_m을 마감시한 D_m에 실행 완료하기 위해서 단일 코어가 최대 속도 $S_{\max}$에 수행해야만 하는 단위 시간당 최소 계산량을 '계산부하'로 정의하고 $U_m(n_m)$으로 표기하면, $U_m(n_m)$은 다음과 같이 계산된다.

$$U_m(n_m) = C_m / (P_m[n_m] \cdot D_m)$$

그리고 n_m개의 코어들을 이용하여 T_m을 실행할 때, n_m개의 코어들이 수행해야 하는 전체 작업량을 '연산량'으로 정의하고 $W_m(n_m)$으로 표기하면, $W_m(n_m)$은 다음과 같이 계산된다.

$$W_m(n_m) = n_m \cdot U_m(n_m) = n_m \cdot \frac{C_m}{P_m[n_m] \cdot D_m}$$

$U_m(n_m)$과 $W_m(n_m)$의 값은 고정되지 않고, n_m과 $P_m[n_m]$이 변화되면 더불어 같이 변동되는 특성을 가진다. 그리고 $n_m = 1$이면, $U_m(1) = W_m(1) = C_m / D_m$이다.

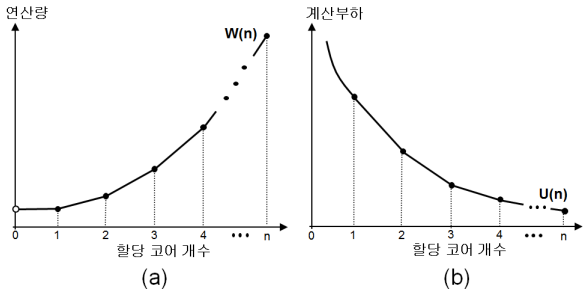

Fig. 1. Workload and Utilization Functions against the Number of Allocated Cores

Fig. 1은 할당된 코어들의 개수 n_m에 대한 연산량(workload) 함수 $W_m(n_m)$과 계산부하(utilization) 함수 $U_m(n_m)$를 보여주고 있다. Fig. 1(a)에서 보여주듯이 병렬 수행 오버헤드로 인해서 작업 수행에 할당된 코어의 개수 n_m이 증가할수록 $W_m(n_m)$값의 증가 비율은 커진다. 반면 Fig. 1(b)에서 보여주듯이 작업 수행에 할당된 코어의 개수 n_m이 증가할수록 $U_m(n_m)$값의 감소 비율이 줄어든다.

본 논문에서는 병렬 수행에 할당된 코어들의 개수를 실행 중에 변경할 수 있는 작업 모델을 고려한다. 대용량의 입력 영역을 계산하는 이미지 처리, Molecular Dynamics 시뮬레이션, Cholesky Factorization, Operational Oceanography와 같은 실시간 멀티미디어 작업들[9, 15-17]은 계산 영역을 분할하여 동시에 수행 가능한 여러 개의 부분작업들로 분리할 수 있고, 계산 영역의 크기를 조절함으로써 부분작업들의 개수를 임의로 변화시킬 수 있다. 그리고 일부의 부분작업들만 상대적으로 적은 계산량을 배당하고 나머지 부분작업들에게는 상대적으로 많은 계산량을 배정하는 불균등 작업량 배정 방법이나, 실행 중에 어떤 코어에 할당된 일부 계산 영역에 대한 인덱스 정보를 다른 코어에게 전달하여 대신 수행하도록 하는 동적 부하균등화 기법[16]이나 동적 작업 이동 기법[8, 17]을 통해서 병렬 수행에 할당된 코어들의 개수를 실행 중에 변경시킬 수 있다. 실행 중에 작업에 할당된 코어들의 개수를 변경하기 위해 발생하는 오버헤드는 병렬 수행 속도 증가 $P_m[n_m]$에 포함된다고 가정한다.

실행 중에 작업에 할당된 코어들의 개수를 변경할 수 있는 환경에서 자연수 이외의 실수 코어 개수 η에 대해서 연산량 $W_m(\eta)$과 계산부하 $U_m(\eta)$을 다음과 같이 도출할 수 있다. 자연수 n에 대해서 $n < \eta < (n+1)$이라면, $(n+1-\eta)$의 시간 비율 동안에는 n개의 코어들이 작업을 수행하고 이후에 $(\eta - n)$의 시간 비율 동안에는 $(n+1)$개 코어들이 수행을 연계하여 작업 실행을 완료하는 경우이다. 즉 $W_m(\eta)$은 $W_m(n)$과 $W_m(n+1)$ 사이의 선형 함수를 따르고, $U_m(\eta)$은 $U_m(n)$과 $U_m(n+1)$ 사이의 선형 함수를 따른다. $W_m(0)$의

값이 정의되지 않은 $0<\eta<1$에 대해서는, 1개 이하의 코어에서는 작업의 연산량이 변하지 않으므로 $W_m(\eta)=W_m(1)$이다. 그리고 $U_m(0)$의 값이 정의되지 않은 $0<\eta<1$에 대해서는, η의 시간 비율 이내에 1개의 코어가 작업 실행을 완료해야 하므로 $U_m(\eta)=U_m(1)/\eta$이다.

3. 제안된 스케줄링 기법

3.1 실수 코어 개수 기반 최소 전력 스케줄링

기존 연구[7]에서 고정된 연산량을 가진 작업들에 대해서 프로세싱 코어들의 계산부하가 균등화되도록 분배하여 모든 코어들에게 동일한 연산 속도를 적용하는 것이 전체 코어들의 전력 소모량이 최소화됨을 입증하였다. 본 논문에서 다루는 문제에서는 Fig. 1(a)에서 보여주듯이 작업 연산량이 할당된 코어 개수에 따라서 변동된다. 이러한 환경에서 전체 코어들의 전력 소모량을 최소화하기 위해서는 모든 작업들의 전체 작업 연산량을 최소화하면서 모든 코어들의 계산부하가 균등하도록 작업들을 배치해야 한다.

실수 코어 개수 η에 대해서 각 작업의 연산량 $W_m(\eta_m)$값이 Fig. 1(a)와 같이 오목 증가함수를 따르는 환경에서는, N개의 코어들을 M개의 작업들에게 적절하게 분배하여 M개의 작업들의 계산부하 $U_m(\eta_m)$값을 일치시키면 N개의 코어들의 전체 전력 소모량이 최소화된다. Fig. 2(a)에서 보여주듯이 M개의 작업들이 동일한 계산부하 $U_m(\eta_m)$값을 가지는 경우가 코어들의 계산부하의 최댓값을 u^*보다 크지 않게 유지하면서 전체 작업들의 연산량을 최소화시킨다. Fig. 2(a)에서 할당된 코어 개수보다 적은 수를 할당하면 작업 연산량을 줄일 수는 있으나, 이 작업의 계산부하가 u^*을 초과하게 된다. 그러면 이 작업의 계산부하로 인해서 모든 코어들의 최대 계산부하가 u^*보다 커지게 되므로 N개의 코어들의 전체 전력 소모량은 오히려 증가된다. 그리고 Fig. 2(a)에서 할당된 코어 개수보다 많은 수를 할당하면, 1을 초과하는 코어 개수에 대해서는 작업 연산량이 증가하게 된다. 하나의 작업이라도 연산량이 증가하게 되면, 전체 연산량이 $u^* \cdot \Sigma\eta_m$을 초과하게 되어 모든 코어들의 최대 계산부하값이 u^*보다 커져야 되므로 전체 전력 소모량이 증가된다. 따라서 모든 작업들이 동일한 계산부하값을 가지도록 코어들을 분배 할당하면, 분배된 코어들의 전체 전력 소모량이 최소화된다. Fig. 2(a)는 $\Sigma\eta_m$개의 코어들만을 분배하여 사용하는 경우에 $\Sigma\eta_m$개의 코어들의 전력 소모량을 최소화시킨다.

N개의 코어들의 전체 전력 소모량을 최소화시키기 위해서는, Fig. 2(c)에서 보여주듯이 $\Sigma\eta_m=N$이 되도록 모든 작업들의 계산부하 $U_m(\eta_m)$이 u^{opt}의 동일한 값을 가지도록 N개의 코어들을 M개의 작업들에 분배해야 한다. 모든 작업들의 단위 계산부하 $U_m(\eta_m)$이 u^{opt}로 동일하면서 $\Sigma\eta_m=N$이 되도록 각 η_m을 찾는 과정은 다음과 같다. 먼저 모든 η_m의 값을 1로 초기화하고, 모든 작업들의 계산부하들 중에서 최대 계산부하 $u^{\max}=\max U_1(1),\ldots,U_M(1)$를 $U_m(\eta)$의 역함수에 적용하여 $U_m^{-1}(u^{\max})=\eta_m$이 되도록 각 η_m의 값을 계산한다. 그리고 계산된 각 η_m의 값에 대해서 $\Sigma\eta_m<N$이라면, $u^{\max}$의 값을 줄여서 $\Sigma\eta_m \geq N$의 조건을 만족할 때까지 각 η_m의 값을 계산하는 과정을 반복한다. $u^{\max}$의 값을 줄일 때, 각 작업들의 연산량 함수 $W_m(\eta_m)$가 모두 선형 증가 관계를 따르도록 $u^{\max}$의 값을 단계적으로 감소시킨다. Fig. 1(a)에서 보여주듯이, η_m의 인접한 자연수까지는 항상 $W_m(\eta_m)$ 함수가 선형 증가 관계를 만족한다. η_m보다 큰 인접한 자연수를 $I(\eta_m)$으로 표현하면, $u^{\max}$의 감소된 값은 $u^{\max}=\max U_1(I(\eta_1)),\ldots,U_M(I(\eta_M))$가 되도록 결정된다. Fig. 2는 제안된 기법의 4개의 작업들을 8개의 코어 상에서 스케줄링하는 주요 과정을 보여주고 있다. Fig. 2(a)의 예에서 $\eta_1=3.2$, $\eta_2=1.0$, $\eta_3=0.7$, $\eta_4=2.1$일 때, $I(\eta_1)=4.0$, $I(\eta_2)=2.0$, $I(\eta_3)=1.0$, $I(\eta_4)=3.0$이다. Fig. 2(b)의 예에서 감소된 $u^{\max}$의 값은 $U_1(I(\eta_1))=\max U_1(I(\eta_1)),\ldots,U_4(I(\eta_4))$로 결정되었고, 결정된 $u^{\max}$를 $U_m(\eta)$의 역함수에 적용하여 $U_m^{-1}(u^{\max})=\eta_m$이 되도록 각 η_m의 값을 다시 계산한다. 계산된 η_m의 값들이 $\Sigma\eta_m \geq N$의 조건을 만족하면 $u^{\max}$의 값을 단계적으로 감소시키는 과정은 정지한다.

$\Sigma\eta_m \geq N$의 조건을 만족하면, 모든 작업들의 계산부하가 동일하면서 $\Sigma\eta_m=N$이 되는 $u^{\max}=u^{opt}$의 값을 다음과 같이 찾는다. 먼저 $\Sigma\eta_m \geq N$의 조건을 만족하기 직전의 $u^{\max}$의 값을 이용하여, 도출한 $\Sigma\eta_m=N_a$값과 $\Sigma W_m(\eta_m)=W_a$의 값을

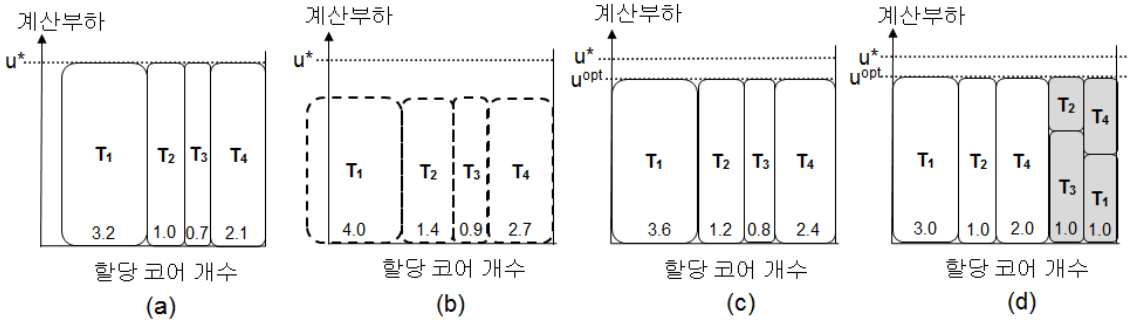

Fig. 2. Working Example of the Proposed Scheme

계산한다. 그리고 $\Sigma\eta_m \geq N$의 조건을 만족한 직후의 $u^{\max}$의 값을 이용하여 도출한 $\Sigma\eta_m = N_b$값과 $\Sigma W_m(\eta_m) = W_b$의 값을 계산한다. $u^{\max}$의 값을 감소시킬 때, 모든 $W_m(\eta_m)$값이 η_m의 값에 대해서 항상 선형 증가 관계를 따르도록 선택되었으므로, 다음의 수식을 이용하여 $\Sigma\eta_m = N$의 상황을 계산할 수 있다.

$$\frac{W_b - W_a}{N_b - N_a} = \frac{W_b - u^{opt} \cdot N}{N_b - N} \tag{2}$$

즉 $u^{opt} = \frac{W_b}{N} - \frac{(W_b - W_a) \cdot (N_b - N)}{(N_b - N_a) \cdot N}$이다. 도출된 u^{opt}의 값을 기준으로 $U_m^{-1}(u^{opt}) = \eta_m$이 되도록 각 η_m의 값을 계산한다. Fig. 2(a)는 $\Sigma\eta_m \geq N$의 조건을 만족하기 직전의 상황이고, Fig. 2(b)는 $\Sigma\eta_m \geq N$의 조건을 만족한 직후의 상황이며, Fig. 2(c)는 Equaition (2)을 이용하여 도출된 $\Sigma\eta_m = N$인 상황을 보여주고 있다.

3.2 자연수 코어 개수 기반 최소 전력 스케줄링

실제 환경에서는 실수 코어 개수들을 작업에게 할당할 수 없으므로, 앞 절에서 도출된 실수 코어 개수 η_m을 자연수 코어 개수로 변환하는 과정이 필요하다. 먼저 각 η_m의 정수 부분인 $\lfloor \eta_m \rfloor$개의 코어들을 작업 T_m에 할당한다. 그러면 $(N - \Sigma\lfloor \eta_m \rfloor)$개의 코어들이 남게 되는데, 이 코어들은 각 η_m의 소수 부분인 $(\eta_m - \lfloor \eta_m \rfloor)$을 수행하도록 할당한다. Fig. 2(d)는 각 η_m의 정수 부분인 $\lfloor \eta_m \rfloor$개의 코어들을 작업 T_m에 할당하고, 각 η_m의 소수 부분인 $(\eta_m - \lfloor \eta_m \rfloor)$을 $(N - \Sigma\lfloor \eta_m \rfloor)$개의 코어들에 할당한 예를 보여주고 있다.

$\Sigma\lfloor \eta_m \rfloor$개의 코어들은 단일 작업만을 실행하지만, $(N - \Sigma\lfloor \eta_m \rfloor)$개의 코어들은 2개 이상의 작업들을 실행하게 된다. 하나의 코어가 다수의 주기적 실시간 작업을 수행하는 경우에는 마감시한이 가까운 작업부터 순차적으로 실행하면 모든 작업들의 마감시한이 만족됨이 입증되었으므로 [7], 2개 이상의 작업들에 대한 실행 순서는 가장 빠른 마감시한을 먼저 실행하는 EDF(Earliest Deadline First) 규칙을 따라 결정된다.

Fig. 2(d)의 예에서 T_3와 T_2를 순차적으로 실행하는 코어는, 80%의 시간 비율 동안에 T_3를 실행하고 20%의 시간 비율 동안에 T_2를 실행한다. 결과적으로 T_2는 80%의 시간 비율 동안에는 1개의 코어가 실행하고, 이후에 동적 작업 이동을 통해서 20%의 시간 비율 동안에는 2개의 코어들이 병렬 수행한다. 마찬가지로 T_1은 60%의 시간 비율 동안에는 4개의 코어들이 실행하고 40%의 시간 비율 동안에는 3개의 코어들이 병렬 수행하며, T_4는 60%의 시간 비율 동안에는 2개의 코어들이 실행하고 40%의 시간 비율 동안에는 3개의 코어들이 병렬 수행한다.

실수 코어 개수를 이용한 스케줄이나 자연수 기반으로 변환한 이후의 스케줄에서 각 프로세싱 코어들의 누적 계산부하는 u^{opt}로 동일하다. 각 프로세싱 코어의 누적 계산부하 값은 배치된 작업들의 마감시한을 모두 만족하기 위해서 단위 시간당 수행하는 최소 계산량을 나타내고, 이는 배치된 작업들의 마감시한을 만족시키는 최소 연산 속도를 의미한다. 따라서 모든 프로세싱 코어들의 동작 속도를 $S = u^{opt}$로 설정하면, 전체 프로세싱 코어들의 전력 소모량은 Equaition (1)을 기반으로 $(\alpha \cdot (u^{opt})^3 + P_l) \cdot N$이 된다.

단계 1. 최대 계산부하 $u^{\max} = \max U_1(1), \ldots, U_M(1)$의 초기값을 계산하고, 계산된 $u^{\max}$를 $U_m(\eta)$의 역함수에 적용하여 $U_m^{-1}(u^{\max}) = \eta_m$인 각 η_m의 초기값을 찾는다. 초기 η_m값들의 합 $\Sigma\eta_m$이 코어 개수 N보다 큰 경우, 마감시간을 만족하는 스케줄링의 불가를 통지하고 동작을 끝낸다.

단계 2. $\Sigma\eta_m < N$이라면 $\Sigma\eta_m \geq N$의 조건을 만족할 때까지 2.1과 2.2의 부속 과정을 반복한다.

2.1 : $\Sigma\eta_m \geq N$의 조건을 만족하기 직전의 $\Sigma\eta_m$값과 $\Sigma W_m(\eta_m)$ 값을 찾기 위해서, 두 개의 변수 N_a와 W_a에 $N_a = \Sigma\eta_m$, $W_a = \Sigma W_m(\eta_m)$가 되도록 저장한다.

2.2 : 모든 연산량 함수 $W_m(\eta_m)$가 선형 증가 관계를 만족하는 범위에서 감소된 $u^{\max}$의 값을 $u^{\max} = \max U_1(I(\eta_1)), \ldots, U_M(I(\eta_M))$ 수식을 이용하여 계산한다. 그리고 계산된 $u^{\max}$을 $U_m(\eta)$의 역함수에 적용하여 $U_m^{-1}(u^{\max}) = \eta_m$이 되도록 각 η_m를 수정한다.

단계 3. 균등화된 계산부하로 $\Sigma\eta_m = N$의 조건을 만족하는 η_m값들을 부속과정 3.1과 3.2를 통해서 찾는다.

3.1 : $\Sigma\eta_m \geq N$의 조건을 만족한 직후의 $\Sigma\eta_m$ 값과 $\Sigma W_m(\eta_m)$ 값을 두 개의 변수 N_b와 W_b에 $N_b = \Sigma\eta_m$, $W_b = \Sigma W_m(\eta_m)$가 되도록 저장한다.

3.2 : N_a, W_a, N_b, W_b 변수들을 Equaition (2)에 적용하여 u^{opt}값을 계산한다. 그리고 계산된 u^{opt}을 $U_m(\eta)$의 역함수에 적용하여 $U_m^{-1}(u^{opt}) = \eta_m$이 되도록 각 η_m값을 수정한다.

단계 4. 자연수 코어 개수를 사용하는 최종 스케줄을 부속과정 4.1과 4.2를 통해서 결정한다.

4.1 : 각 η_m의 정수 부분인 $\lfloor \eta_m \rfloor$개의 코어들을 작업 T_m의 전담 실행에 할당한다. 그리고 각 η_m의 소수 부분 $(\eta_m - \lfloor \eta_m \rfloor)$을 남겨진 $(N - \Sigma\lfloor \eta_m \rfloor)$개의 코어들에 할당하며, 이때 마감시한이 빠른 작업부터 순차적으로 선택하여 계산부하가 가장 낮은 코어에게 할당한다.

4.2 : 모든 프로세싱 코어들의 동작 속도를 $S = u^{opt}$로 설정하고, 2개 이상의 작업들을 실행하는 $(N - \Sigma\lfloor \eta_m \rfloor)$개의 코어들은 마감시한이 가까운 작업부터 순차적으로 실행한다.

Fig. 3. Pseudo Code of the Proposed Scheme

Fig. 3은 제안된 스케줄링 기법의 전체 동작 과정을 보여주는 의사 코드(pseudo code)이다. 여기서 $I(\eta)$는 실수 η보다 큰 인접한 자연수를 간결하게 나타낸다. Fig. 2(a)는 Fig. 3의 단계 2 진행 과정을 보여주고 있고, 단계 2가 완료되면 Fig. 2(b)의 결과가 도출된다. Fig. 3의 단계 3 과정이 완료

되면 Fig. 2(c)의 결과가 도출되고, Fig. 3의 단계 4과정이 완료되면 Fig. 2(d)의 결과가 도출된다.

Fig. 3에서 제시된 코드의 전체 계산 복잡도는 $O(N \cdot \ln N \cdot M^2)$이다. 최대 N개의 변곡점을 가진 $U(\eta)$의 역함수 $U^{-1}(u^{\max})$에 대한 계산 복잡도는 이등분할 검색(bisectional search)을 이용하면 $O(\ln N)$이다. 즉 단계 1과 단계 2.2, 단계 3.2의 계산 복잡도는 $O(\ln N \cdot M)$이다. 단계 2.1의 계산 복잡도는 $O(1)$이며, 단계 2는 최대 $N \cdot M$번 반복된다. 단계 3.1의 계산 복잡도는 $O(1)$이고, 단계 4.1의 계산 복잡도는 $O(M \cdot \ln M + N \cdot M)$이며, 단계 4.2의 계산 복잡도는 $O(M \cdot \ln M)$이다.

4. 성능 평가

성능 평가를 위하여 제안된 기법과 기존의 병렬 수행을 적용한 스케줄링 방법[11]의 전력 소모량을 비교하였다. 기존 방법에서는 병렬 수행을 활용하여 전력 소모량을 줄였지만, 부하균등화를 적용하지 않아서 일부 코어들에 계산부하가 편중되는 부하 불균등 현상이 발생하면 모든 코어들에게 동시에 적용되는 연산 속도가 증가되어 전력 소모 감소 효과가 떨어진다. 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.

모의 성능 평가 실험에서 32개의 주기적 실시간 작업들을 사용하고, 각 작업들은 다른 계산부하 $U(1)$값을 가지도록 생성되었다. 작업들의 계산부하값들은 정규 분포 또는 지수 분포를 따르도록 인위적으로 생성하였고, 모든 작업들의 평균 작업부하를 $\frac{\sum_{m=1}^{M} U_m(1)}{N} \times 100$으로 정의하여 사용하였다. 평가 신뢰도를 높이기 위해서 32개의 작업들을 가지는 집합을 십만 번 생성하여 평균값을 실험 결과 지표로 사용하였다. 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 S의 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 $F(S) = (\alpha \cdot S^3 + P_l)$ 함수의 상수 $\alpha \simeq 1.55 \times 10^{-6}$를 도출하였다[6]. 2.1절에서 언급하였듯이, 기존 방법과 제시된 기법의 P_l값은 항상 동일하므로 성능 평가에서 P_l은 제외하였다.

병렬 수행을 적용하면 일반적으로 할당된 코어 개수 n에 비례하여 계산부하 $U(n)$이 줄어들지만, 병렬 수행의 속도 증가 $P[n]$은 작업의 서브모듈 분할 특성에 따라 달라진다[13]. 다양한 작업들의 병렬 수행 효율성을 평가하기 위해, n개의 코어들에서의 병렬 수행 속도 증가 모델들을 다음과 같이 3가지 모델을 설정하여 사용하였다. 먼저 '선형 속도 증가' 모델은 $P[n] = n$으로 정의하고, 병렬 수행을 통해서 얻을 수 있는 최대 성능을 대표하였다. 그리고 '반선형 속도 증가' 모델은 $P[n] = (0.5 \cdot (n-1) + 1)$로 정의하고, 병렬 수행의 중간 성능을 대표하였다. 마지막으로 '제곱근 속도 증가' 모델은 $P[n] = \sqrt{n}$으로 정의하고, 병렬 수행의 낮은 성능을 대표하였다. 본 논문에서 고려하는 멀티미디어 작업들은 선형 속도 증가 모델에 근접하는 병렬 수행 속도 증가를 보이지만[13, 15], 병렬 수행 속도 증가가 상대적으로 떨어지는 작업들에게 제안된 기법을 적용할 때의 전력 소모 감소 효과를 확인하기 위해서 반선형 속도 증가 모델과 제곱근 속도 증가 모델을 성능 평가에 적용하였다.

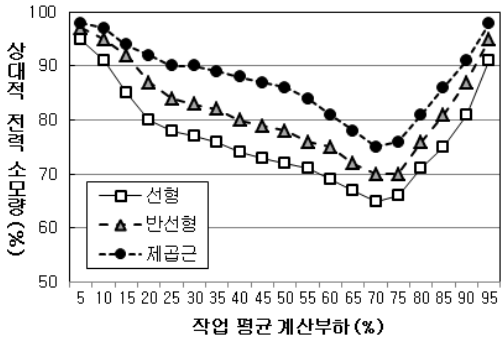

Fig. 4. Relative Power Consumption against Normal Distribution

Fig. 4는 작업들의 $U(1)$값이 정규분포를 따르도록 생성되었을 때, 평균 작업부하에 대한 제안된 방법의 상대적 전력 소모량을 보여주고 있다. 평균 작업부하값이 70%일 때 최저 상대적 전력 소모량을 보인다. 그리고 평균 작업부하 5%와 70% 사이에서는 평균 작업부하가 증가할수록 상대적 전력 소모량은 감소하고, 평균 작업부하 70%와 95% 사이에서는 평균 작업부하가 증가할수록 상대적 전력 소모량은 증가한다. 제안된 기법은 기존 방법의 최대 계산부하값을 부하균등화를 적용하여 줄임으로써 전체 코어들의 전력 소모량을 감소시킨다. 따라서 제안된 기법의 상대적 전력 소모량이 낮다는 것은, 기존 방법에서 부하 불균등으로 인한 일부 코어들의 계산부하 편중 현상이 크다는 것을 의미한다.

모든 평균 작업부하값들에 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 작업부하값이 70%일 때, 선형 속도 증가 모델의 상대적 전력 소모량은 약 65%이고 반선형 속도 증가 모델은 약 70%이며 제곱근 속도 증가 모델은 약 76%이다. Fig. 4의 실험 결과에서 제안된 방법은 기존 방법의 전력 소모량을 최대 35%까지 감소시키고, 또한 다양한 병렬 수행 속도 증가 모델에 대해서 명확한 전력 소모 감소 효과가 있음을 보여주고 있다. 동일한 부하 불균등 상황에 대해서도 병렬 수행 속도

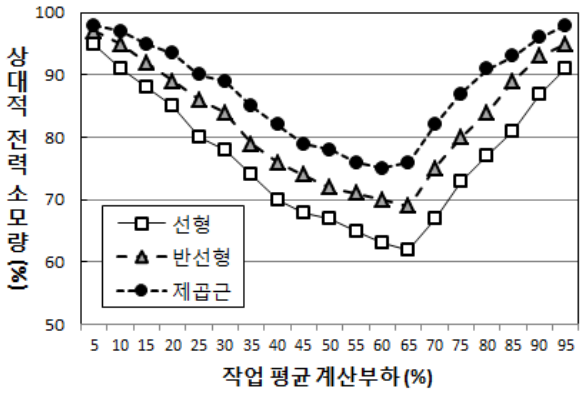

Fig. 5. Relative Power Consumption against Exponential Distribution

증가값이 클수록 부하균등화를 통한 최대 계산부하의 감소폭이 증가하고, 따라서 상대적 전력 소모량의 감소폭이 커진다.

Fig. 5는 작업들의 $U(1)$값이 지수 분포를 따를 때의 상대적 전력 소모량을 보여주고 있다. Fig. 3의 결과와 유사하게 우수한 병렬 수행 속도 모델에 대해서 낮은 상대적 전력 소모량 값을 보인다. 그리고 평균 작업부하 60% 이하에서는 평균 작업부하가 증가할수록 상대적 전력 소모량은 감소하고, 평균 작업부하 65% 이상에서는 평균 작업부하가 증가할수록 상대적 전력 소모량은 증가한다. 선형 속도 증가 모델과 반선형 속도 증가 모델은 평균 작업부하값이 65%일 때 상대적 전력 소모량의 최저값으로 62%와 69%를 각각 보이고, 제곱근 속도 증가 모델은 평균 작업부하값이 60%일 때 상대적 전력 소모량의 최저값으로 75%을 보인다. 즉 선형 증가 모델에 대해서 프로세서 구동을 위한 최소 기반 전력을 제외하고 기존 방법이 소모하는 전력량을 제안된 기법이 최대 38%까지 감소시킨다.

5. 결 론

본 논문에서는 주파수 공유형 멀티코어 프로세서상에서 실시간 작업들의 마감시한을 만족하면서 전력 소모량을 최소화시키는 스케줄링 기법을 제안하였다. 제안된 기법은 병렬 수행과 부하균등화를 적용하여 모든 코어들에게 동일하게 적용되는 연산 속도를 최소화시켰다. 병렬 수행을 적용하면 각 작업을 수행하는 프로세싱코어들의 개수에 따라서 작업량이 변동되므로, 제안된 기법은 모든 작업들의 전체 연산량을 최소화화면서 모든 프로세싱 코어들의 최대 계산부하가 최소화되도록 설계되었다. 성능 평가 실험에서 제안된 기법이 기존의 병렬 수행만을 적용하고 부하균등화를 적용하지 않는 방법보다 전력 소모량을 최대 38%까지 감소시킴을 확인하였다.

References

[1] 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, A. J. Smith, “PACE: a new approach to dynamic voltage scaling,” *IEEE Trans. Computers*, Vol.53, No.7, pp. 856-869, 2004.

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

[4] C. Yang, J. Chen, and T. Kuo, “An approximation algorithm for energy-efficient scheduling on a chip multiprocessor,” in Design, Automation and Test in Europe Conf., pp.468-473, 2005.

[5] 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.

[6] C. Xian, Y. Lu, and Z. Li, “Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time,” in Design Automation Conf., pp.664-669, 2007.

[7] H. Aydin, Q. Yang, “Energy-aware partitioning for multiprocessor real-time systems,” in Int’l Parallel Distributed Processing Symp., pp.113.2, 2003.

[8] E. Seo, J. Jeong, S. Park, and J. Lee, “Energy efficient scheduling of real-time tasks on multicore processors,” *IEEE Trans. Parallel Distributed Syst.*, Vol.19, No.1, pp.1540-1552, 2008.

[9] W. Lee, H. Kim, and H. Lee, “Minimum-energy semi-static scheduling of a real-time video stream on DVFS-enabled multi-core processors,” *IEICE Trans. Information and Systems*, Vol.E94-D, No.12, pp.2389-2392, 2011.

[10] 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.

[11] W. Lee, “Energy-efficient scheduling of periodic real-time tasks on lightly loaded multi-core processors,” *IEEE Trans. Parallel Distributed Syst.*, Vol.23, No.3, pp.530-537, 2012.

[12] K. Lee, “Energy-efficient fault-tolerant scheduling based on duplicated executions for real-time tasks on multicore processors,” *Journal of the Korea Society of Computer and Information*, Vol.19, No.5, pp.1-10, 2014.

[13] 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.

[14] E. Talpes, D. Marculescu, “Toward a Multiple Clock/Voltage Island Design Style for Power-Aware Processors,” *IEEE Trans. Very Large Scale Integration Systems*, Vol.13, No.5, pp.591-603, 2005.

[15] E. K. Burke, M. Dror, and J. B. Orlin, “Scheduling malleable tasks with interdependent processing rates: comments and observations,” *Discrete Applied Mathematics*, Vol.156, pp. 620-626, 2008.
[16] Z. Lan, V. E. Taylor, and G. Bryan, “A novel dynamic load balancing scheme for parallel systems,” J*ournal of Parallel and Distributed Computing*, Vol.62, pp.1763-1781, 2002.
[17] S. Chakravorty, C. L. Mendes, and L. V. Kale, “Proactive fault tolerance in MPI applications via task migration,” in Int’l Conf. High Performance Computing, pp.485-496, 2006.

이 완 연
e-mail : wanlee@dongduk.ac.kr
1994년 포항공과대학교 컴퓨터공학과(학사)
2000년 포항공과대학교 컴퓨터공학과(박사)
2011년~현 재 동덕여자대학교 컴퓨터학과 부교수
관심분야 : 임베디드 시스템, 시스템 소프트웨어