Journal of Korea Multimedia Society Vol. 13, No. 1, January 2010(pp. 30-37)

태스크 동기화가 필요한 임베디드 실시간 시스템에서 시간-효율적인 전압 스케쥴링 알고리즘

이재동[†], 김정종[††]

요 약

최근 많은 임베디드 실시간 시스템에 동적 전압 조절(Dynamic Voltage Scaling: DVS)을 지원하는 프로세서를 사용하고 있다. 이런 시스템의 설계 및 동작의 최적화를 위한 중요한 요소 중 하나가 전력(power)이다. 동적 전압 조절을 지원하는 프로세서의 슬로우다운을 이용함으로서 많은 소비 전력을 절약할 수 있다. 본 논문에서는 태스크의 동기화가 필요한 임베디드 실시간 시스템에서 효율적인 전력 소비를 위해 태스크들의 슬로우다운 값을 구하는 기존 알고리즘을 시간복잡도 측면에서 개선하였다. 시간복잡도가 $O(n^2)$인 기존 알고리즘을 수학적인 분석 및 시뮬레이션을 통하여 그 성질을 파악하고, 그 성질을 이용하여 기존 알고리즘과 같은 성능을 가지는 시간복잡도가 $O(nlogn)$ 및 $O(n)$인 개선된 알고리즘들을 제안하였다.

Time-Efficient Voltage Scheduling Algorithms for Embedded Real-Time Systems with Task Synchronization

Jae Dong Lee[†], Jung Jong Kim[††]

ABSTRACT

Many embedded real-time systems have adopted processors supported with dynamic voltage scaling(DVS) recently. Power is one of the important metrics for optimization in the design and operation of embedded real-time systems. We can save considerable energy by using slowdown of processor supported with DVS. In this paper, we improved the previous algorithm at a point of view of time complexity to calculate task slowdown factors for an efficient energy consumption in embedded real-time systems with task synchronization. We grasped the properties of the previous algorithm having $O(n^2)$ time complexity through mathematical analysis and simulation. Using its properties we proposed the improved algorithms with $O(nlogn)$ and $O(n)$ time complexity which have the same performance as the previous algorithm has.

Key words: Dynamic Voltage Scaling(동적 전압 조절), Embedded Real-Time System(임베디드 실시간 시스템), Task Synchronization(태스크 동기화), SlowDown(슬로우다운), Voltage Scheduling Algorithm(전압 스케쥴링 알고리즘)

1. 서 론

임베디드 시스템들이 발전하고 그 사용 범위가 넓어짐에 따라 실시간 성질을 많이 요구하게 되었다. 즉, 임베디드 시스템의 운영체제로 실시간 운영체제를 많이 사용하고 있다. 이런 시스템을 임베디드 실

※ 교신저자(Corresponding Author) : 이재동, 주소 : 경남 마산시 월영동 449번지, 전화 : 055)249-2214, FAX : 055) 248-2554, E-mail : jdlee@kyungnam.ac.kr
접수일 : 2009년 6월 25일, 수정일 : 2009년 9월 13일
완료일 : 2009년 9월 13일

[†] 정회원, 경남대학교 컴퓨터공학부
[††] 정회원, 경남대학교 컴퓨터공학부 교수
(E-mail : jjkim@kyungnam.ac.kr)
※ 이 연구결과물은 2009학년도 경남대학교 학술연구장려금 지원에 의한 것임.

시간 시스템이라 한다. 전력(power)은 임베디드 실시간 시스템의 설계 및 동작의 최적화를 위한 중요한 요소 중 하나이다. 임베디드 실시간 시스템에서 전력 소비를 줄이는 두 가지 주 방법으로 셧다운(shutdown)과 슬로우다운(slowdown)이 있다. 소비되는 전력은 전압의 제곱에 비례하므로 클럭 속도와 전압을 조절하는 슬로우다운이 전력소비를 줄이는데 더욱 효율적인 것으로 알려져 있다. 프로세서의 클럭 속도와 전압을 조절하면 전력소비는 줄일 수 있지만 작업(job)의 수행시간은 길어진다. 임베디드 실시간 시스템에서는 태스크의 마감시간을 유지하면서 전력소비를 최소화시키려고 한다. 전력소비를 줄이는 것과 마감시간을 만족시키는 것은 서로 상충되므로 전력소비를 최소화하기 위해 전력과 시간을 잘 고려해야 한다.

본 논문에서는 태스크의 슬로우다운 값의 계산을 통해 시스템 레벨의 전력관리에 초점을 맞춘다. 슬로우다운 값(slowdown factor)은 실행시간에 프로세서의 속도를 결정하는 정규화된 클럭속도이다. 예를 들어, 슬로우다운 값이 1이면 최대의 클럭 속도를 나타내고, 0.8이면 최대 클록 속도의 80%의 속도를, 0이면 정지 상태를 나타낸다. 슬로우다운 값(이후로 '슬로우다운 값'과 '슬로우다운'을 같은 의미로 사용함)의 계산은 태스크의 특성을 고려하여 오프라인(offline)으로 계산되는 정적 슬로우다운(static slowdown)과 태스크의 실행 중에 계산되는 동적 슬로우다운(dynamic slowdown)으로 분류할 수 있다. 또한, 프로세서에서 지원되는 클럭 속도 조절 방법에 따라 연속 슬로우다운과 이산 슬로우다운으로 분류할 수 있다. 연속 슬로우다운은 클럭 속도를 임의의 속도로 조절할 수 있는 프로세서를 위해 사용할 수 있는 반면, 클럭 속도를 몇 개의 레벨에 대해서만 지원하는 프로세서에서는 이산 슬로우다운을 사용한다. 본 연구는 태스크들이 공유자원을 사용하는, 즉, 동기화(synchronization)가 필요한 태스크 모델에서 동적 및 정적 슬로우다운에 모두 활용할 수 있는 연속 슬로우다운의 계산에 초점을 맞춘다. 연속 슬로우다운은 간단하게 이산 슬로우다운으로 수정할 수 있다. 본 논문에서 고려하는 실시간 시스템에서는 태스크들이 주기적이고 마감시간을 가지며, 태스크들은 EDF(Earliest Deadline First) 스케쥴링 정책[1,2]에 기반하여 단일 프로세서 시스템에 스케쥴링 된다. 또한, 태스크들은 공유자원을 상호 배제적으로 사용하기 위해 동기화가 필요하다.

전력을 효율적으로 사용하기 위한 많은 연구가 진행되어 왔다[3]. 대부분의 기존 연구들은 독립된 태스크 집합에 대하여 전력을 고려한 스케쥴링에 관한 것이다. 비록 동적 전압 조절(dynamic voltage scaling: DVS)를 이용한 많은 연구가 있었지만, 태스크의 동기화가 존재하는 모델에 대해서는 거의 연구되지 않았다. 대부분의 실시간 응용은 시스템의 공유자원을 사용하고 있다. 공유자원의 상호배제적 사용은 우선순위 전도(priority inversion) 문제를 일으킨다. 만약 낮은 우선순위 작업이 하나의 자원을 사용하고 있다면 그 자원을 요구하는 높은 순위의 작업은 블록(block)되어 마감시간을 놓칠 수 있다. 태스크의 동기화가 있는 경우의 스케쥴링은 NP-hard이며[4-6], 그 경우의 실행가능성 테스트(feasibility test)에 대해 연구 되었다[7,8]. 실행가능성 테스트에 기초한 일정 슬로우다운 값의 계산 방법이 제시 되었으며[9-11], 비슷한 연구로 Zhang과 Chanson은 비선점 섹션(section)을 가진 태스크의 슬로우다운을 계산하는 방법을 제시하고, DS(Dual-Speed)알고리즘을 제안하였다[12]. 이 알고리즘은 태스크 내의 비선점 섹션만을 허용하므로 태스크 동기화를 가진 스케쥴링에 적용할 수 없다. Chen 등은 클럭속도 록킹과 PCP의 확장을 통해 전력 소비를 최소화하는 태스크 동기화 방법을 제안했다[13]. 제시한 알고리즘은 고정 우선순위 태스크들의 집합에 활용할 수 있다. Jejurikar와 Gupta는 동기화 제약조건 하에서 개개의 태스크에 대한 슬로우다운 값을 계산하는 알고리즘을 제시하였다[11]. 이 알고리즘은 EDF와 RM 스케쥴링 정책 둘 다에 사용될 수 있으며, 어떠한 자원 엑세스 정책(PCP[8], DPCP[14], SRP[9])과도 같이 사용할 수 있다. EDF를 위해 제시된 알고리즘의 시간복잡도는 $O(n^2)$ 이다. Jejurikar와 Gupta가 제시한 알고리즘을 개선하여 같은 시간복잡도를 가지면서 더 좋은 성능을 가지는 알고리즘이 제시되었다[18].

본 논문에서는 Jejurikar와 Gupta가 제시한 알고리즘(이후로 '기존 알고리즘'이라 함)을 시간-효율성(즉, 시간복잡도) 측면에서의 개선을 위하여 분석하였다. 수학적 분석 및 시뮬레이션을 통하여 기존 알고리즘의 성질을 제시하고, 이 성질을 이용하여 기존 알고리즘과 같은 성능을 갖는 시간복잡도가 $O(n\log n)$ 및

$O(n)$인 개선된 2개의 알고리즘을 제안하였다. 개선된 알고리즘들은 정적 슬로우다운의 계산에도 효율적으로 사용할 수 있을 뿐만 아니라, 특히 동적 슬로우다운 계산에 효율적이다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 시스템 모델 및 기존 알고리즘을 기술하고, 3장에서는 기존 알고리즘을 분석한 결과를 제시하고, 4장에서는 시간복잡도 측면에서 개선된 2개의 알고리즘을 제시하고, 5장에서는 결론과 향후연구에 대한 방향을 제시한다.

2. 시스템 모델 및 슬로우다운 계산 알고리즘

이 장에서는 시스템 모델과 기존 슬로우다운 계산 알고리즘을 기술한다.

2.1 시스템 모델

n개의 주기적 실시간 태스크의 집합 $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$이다. 태스크 τ_i는 (T_i, D_i, C_i, B_i)로 나타낸다. T_i는 태스크의 주기를, $D_i(=T_i)$는 상대 마감시간을, C_i는 프로세서가 최대속도로 수행할 때의 최악의 수행시간(WCET : Worst Case Execution Time)을 나타낸다. 태스크의 각 호출(invocation)을 작업(job)이라 하고, 태스크 τ_i의 k번째 호출을 $\tau_{i,k}$라 둔다. 시스템은 태스크들에 의해 상호 배제적으로 사용되는 공유자원들을 가진다. 독점적 자원 사용을 위한 동기화 방법으로 세마포어를 사용한다고 가정한다. 모든 태스크들은 공유 자원들에 대한 접근이 보호되는 범위 내에서 선점가능하다. 태스크에게 공유자원에 대한 접근이 허락되었을 때 그 태스크는 임계구역(critical section)에서 수행되고 있다고 한다. 태스크의 임계구역들은 적절히 내포된(properly nested) 것으로 가정한다[8]. 즉, 두 개의 임계구역이 서로 겹칠 때 하나의 임계구역이 다른 임계구역을 완전히 포함하고 있다. 태스크 τ_i의 k번째 임계구역을 $z_{i,k}$로 나타낸다. 만약, 한 태스크가 낮은 우선순위 태스크의 공유 자원 반납(release)을 기다려야 한다면 그 태스크는 블록 되었다고 한다. 자원을 가지고 있는 태스크를 블록킹 태스크(blocking task)라고 하고, 태스크가 블록 되어질 시간을 태스크 블록킹 시간(task blocking time)이라 한다. 각 태스크에 대한 공유자원의 접근과 각 임계구역의 WCET는 미리 주어진다. 주어진 태스크의 정보와 자원접근 프로토콜을 사용하여 태스크의 최대 블록킹 시간을 계산할 수 있다. B_i를 태스크 τ_i에 대한 최대 블록킹 시간이라 둔다.

태스크들은 가변 전압(가변 클럭속도)를 지원하는 단일 프로세서에 스케쥴링 된다. 프로세서를 슬로우다운 함으로서 전력을 줄일 수 있으며, 슬로우다운 값은 정규화된 클럭속도이다. 즉, 슬로우다운 값은 프로세서의 최대 속도에 대한 스케쥴된 속도의 비율이다. 본 논문에서는 시스템의 여유 부분(slack)을 더 잘 이용하기 위해 모든 태스크에 동일한 슬로우다운을 할당하는 방법 대신, 태스크마다 다른 슬로우다운 값을 할당한다. 한 태스크의 모든 작업들은 동일한 슬로우다운(그 태스크의 슬로우다운)을 적용한다. 우리의 목적은 마감시간을 만족시키면서 전력 소비를 최소화 하는 것이다. 프로세서 속도를 변화시키는데 필요한 시간과 전력은 태스크의 시간과 전력에 비해 아주 작다. 전압 변화를 위한 오버헤드는 구문 교환(context switch) 오버헤드와 비슷하게 태스크의 실행시간에 포함시킬 수 있다. 본 시스템의 스케쥴링 정책은 EDF로 하고, 자원접근정책은 DPCP [14] 또는 SRP[9]로 한다.

2.2 실행가능성 테스트

2.1절에서 언급한 시스템 모델에서 태스크 집합의 실행가능성 테스트(feasibility test)는 아래 정리와 같다[9,14].

정리 1. $\Gamma=\{\tau_1, \tau_2, \ldots \tau_n\}$을 시스템에 있는 태스크들의 집합이라 하고, 이 태스크들은 상대 마감시간의 오름차순으로 정렬되어 있다고 하자. 이때, <식 1>이 성립하면 태스크 집합은 실행가능하다.

$$\forall i,\ i=1,2,\ldots,n,\ \frac{B_i}{D_i}+\sum_{k=1}^{i}\frac{C_k}{D_k}\le 1 \qquad \text{(식 1)}$$

2.3 가변 속도 프로세서

Intel strong ARM 프로세서[15], Intel XScale[16], Transmeta Crusoe[17] 등은 가변 전압 및 가변 클럭속도를 지원한다. 전압과 클럭속도는 일대일 대응 관계에 있으므로 전압을 변화시켜(즉, 클럭속도를 변화시켜) 프로세서의 속도를 변화시킬 수 있다[6]. 따라서, 프로세서의 슬로우다운은 클럭속도(즉, 전압)

을 변화시키는 것이다. 프로세서의 최대 클럭속도를 $f_{\max}$으로 둘 때, 슬로우다운 값은 최대클럭 속도에 대한 태스크(τ_i)에 할당된 클럭속도(f_i)의 비로 나타낸다. 즉, 태스크(τ_i)의 슬로우다운 η_i 는 $f_i/f_{\max}$ 로 나타낼 수 있다.

2.4 슬로우다운 계산을 위한 기존 알고리즘

실행가능성 분석을 기반으로 2.1절에서 정의한 태스크 집합에 대한 슬로우다운 계산은 그림 1과 같다[11]. 여기서 구한 η_i 는 태스크 τ_i 의 슬로우다운 값이다. $\eta_i (1 \le i \le n)$가 1보다 큰 태스크가 존재하면 이 태스크 집합은 실행가능하지 않다. 실행가능하면, 이렇게 생성된 슬로우다운 값은 각 태스크가 실행될 때의 프로세서의 클럭속도를 결정한다. 또한, 태스크의 수행 시 스케쥴링 정책은 EDF로 하고 자원 접근 정책은 DPCP 또는 SRP를 이용하며, 클럭속도 상속(frequency inheritance)[11]을 한다. 클럭속도 상속이란 아래와 같이 태스크의 수행 중 슬로우다운이 상속되도록 하는 정책이다. 여기서, η는 현재 프로세서의 슬로우다운 값이다.

1) rule 0 : $\eta = 0$, idle 한 경우.
2) rule 1 : $\eta = \eta_i$, τ_i가 실행 중이고 어떤 다른 태스크도 블록시키고 있지 않을 때.
3) rule 2 : $\eta = (\eta_i, \max_j(\eta_j))$, τ_i가 태스크들을 블록하고 있을 때(τ_j는 블록된 태스크들을 나타냄).

$\{\tau_1, \tau_2, \ldots \tau_n\}$이 상대 마감시간에 대해 증가 순(nondecreasing order)으로 정렬되었다고 가정함.

1) q =1
2) while($q \le n$)do
3) for($i = q$; $i \le n$; i++) do
4) $\eta_i = \dfrac{\frac{B_i}{D_i} + \sum_{q \le p \le i} \frac{C_p}{D_p}}{1 - \sum_{1 \le r < q} \frac{1}{\eta_r} \frac{C_r}{D_r}}$
5) endfor
6) $\eta_m = \max_{i=q}^{n}(\eta_i)$
7) for($i = q$; $i \le m$; i++)do
8) $\eta_i = \eta_m$
9) endfor
10) $q = m + 1$
11) endwhile

그림 1. 슬로우다운 계산 알고리즘

기존 알고리즘의 시간복잡도는 $O(n^2)$이다[11].

3. 기존 슬로우다운 계산 알고리즘의 분석

본 장에서는 시간복잡도 측면에서 더 효율적인 알고리즘을 제안하기 위해 기존 알고리즘의 성질을 분석한다.

3.1 기존 알고리즘의 성질

태스크 집합 $T = \{\tau_1, \tau_2, .., \tau_n\}$은 상대 마감시간의 오름차순으로 정렬되어 있다고 가정한다.

정리 2. 태스크 집합 $T = \{\tau_1, \tau_2, .., \tau_n\}$이 실행가능하면 그림 1의 알고리즘으로 구한 슬로우다운은 아래 조건을 만족한다.

$\forall i,\ i = 1, 2, \ldots, n$에 대하여 $0 < \eta_i \le 1$

(증명) 슬로우다운의 정의에 따라 모든 태스크들이 실행되기 위해서는 각 태스크의 슬로우다운 값 η_i는 0보다 크고 1보다 작거나 같아야 한다. ■

정리 3. 태스크 집합 $T = \{\tau_1, \tau_2, .., \tau_n\}$이 실행가능하면 아래 조건이 성립한다.

$\forall i,\ i = 1, 2, \ldots, n$에 대하여 $\left(1 - \sum_{1 \le r < i} \frac{1}{\eta_r} \frac{C_r}{D_r}\right) > 0$

(증명) 정리1의 실행가능 조건에 의하여 모든 $i = 1, \ldots, n$에 대하여 <식 2>가 만족된다.

$$\frac{B_i}{D_i} + \sum_{r=1}^{i} \frac{C_r}{Dr} \le 1 \qquad \text{(식 2)}$$

정리2와 <식 2>로부터, 모든 i에 대하여

$$\frac{B_i}{D_i} \le 1 - \sum_{r=1}^{i} \frac{C_r}{D_r} \le 1 - \sum_{r=1}^{i} \frac{1}{\eta_r} \frac{C_r}{D_r} < 1 - \sum_{1 \le r < i} \frac{1}{\eta_r} \frac{C_r}{D_r}$$

따라서, $(1 - \sum_{1 \le r < i} \frac{1}{\eta_r} \frac{C_r}{D_r}) > 0$ ■

기존 알고리즘을 자세히 고찰해 보자. 임의의 q에 대하여, 3번~6번 라인에서 $\eta_q, \eta_{q+1}, \ldots, \eta_n$을 구한 후, 6번~9번 라인에서 $\eta_q, \eta_{q+1}, \ldots, \eta_n$중에서 최대값 η_m을 구하여 $\eta_q, \eta_{q+1}, \ldots, \eta_m$의 값을 η_m으로 설정한다. 즉,

$\tau_q, \tau_{q+1}, \ldots, \tau_m$의 슬로우 다운은 η_m이 된다. 그리고 q를 다시 m+1로 재설정 한 다음(이때의 q를 q'로 표시한다), $\eta_{q'}$부터 η_m까지를 다시 계산한다. q'을 설정한 후, 다시 계산된 η_i들을 η_i'으로 표시한다.

정리 4. $\forall i,j \geq q'$에 대하여 $\eta_i > \eta_j$이면 $\eta_i' > \eta_j'$이다.

(증명) $i>j$라고 가정해도 일반성을 잃지 않으므로 $i>j$로 둔다.

$\eta_i > \eta_j$이므로 $\eta_i - \eta_j = \dfrac{\left(\frac{B_i}{D_i} - \frac{B_j}{D_j} + \sum_{j<k\leq i} \frac{C_k}{D_k}\right)}{\left(1 - \sum_{1\leq r<q} \frac{1}{\eta_r}\frac{C_r}{D_r}\right)} > 0$ 이다.

따라서, <식 3>이 성립된다.

$$\left(\frac{B_i}{D_i} - \frac{B_j}{D_j} + \sum_{j<k\leq i} \frac{C_k}{D_k}\right) > 0 \qquad \text{(식 3)}$$

따라서, $\eta_i' - \eta_j' = \dfrac{\left(\frac{B_i}{D_i} - \frac{B_j}{D_j} + \sum_{j<k\leq i} \frac{C_k}{Dk}\right)}{\left(1 - \sum_{1\leq r<q'} \frac{1}{\eta_r}\frac{C_r}{D_r}\right)}$ (식 4)

정리3과 <식 3>에 의해 <식 4>는 0보다 크다. 따라서, $\eta_i' > \eta_j'$이다. ■

정의 1. 기존 알고리즘에서 2번~11번 라인의 while루프를 k번 반복한다고 가정하고, 각 반복에서의 m 값을 $q_1, q_2, \ldots, q_k$라 둔다.

이때, $q_1, q_2, \ldots, q_k$는 아래 조건을 만족한다.

$$q_1 < q_2 < q_3 \ldots < q_k$$

따름 정리 1. $q=q_s$ (s≥1)로 놓고 구한 η_i를 η_i^s라 두자. 이때, $\forall i,j \geq q_s$에 대하여 $\eta_i^{s-1} > \eta_j^{s-1}$이면 $\eta_i^s > \eta_j^s$이다.

(증명) 정리4의 증명과 같은 방법으로 간단히 증명된다. ■

따름 정리 2. q=1로 두고 구한 η_i를 $\eta_1^0, \eta_2^0, \eta_3^0, \ldots, \eta_m^0$이라 두자. 이때, 아래의 조건이 성립한다.

$$\eta_{q_1}^0 > \eta_{q_2}^0 > \eta_{q_3}^0 > \ldots > \eta_{q_k}^0$$

(증명) q=1일 때, η_{q_1}은 알고리즘에 의해 $\eta_1^0, \eta_2^0, \ldots, \eta_n^0$ 중 최대값이다. $q=q_1$일 때 다시 구한 값을 $\eta_{q_1+1}^1, \eta_{q_1+2}^1, \ldots, \eta_n^1$이라 두자. 이들 중 최대값의 위치($q_2$)는 정리4에 의해 $\eta_{q_1+1}^0, \eta_{q_1+2}^0, \ldots, \eta_n^0$ 중에서 최대값의 위치와 같다. 따라서, $\eta_{q_1}^0 > \eta_{q_2}^0$이다. 따름정리1을 이용하여 같은 방법으로, $\eta_{q_1}^0 > \eta_{q_2}^0 > \ldots > \eta_{q_k}^0$이 성립함을 보일 수 있다. ■

3.2 시뮬레이션을 이용한 기존 알고리즘의 성질

$q_1, q_2, \ldots, q_k$는 정의1에서 정의한 것과 같다. 이때, k의 값을 추정하기 위해 시뮬레이션을 수행한다. 가능한 다양한 태스크 집합에 대하여 분석하기 위해, 아래와 같은 세 종류의 태스크 집합에 대하여 각각 시뮬레이션 한다. n은 태스크의 수로 둔다.

첫 번째 태스크 집합 $\{\tau_1, \tau_2, \ldots, \tau_n\}$의 D_i, C_i, B_i는 다음과 같이 설정한다. 먼저, 전체 태스크의 $\frac{1}{3}$의 D_i는 $[100\times n, 300\times n]$구간의 균등분포(uniform distribution)를 따르며, C_i는 [10, 300]구간의 균등분포를 따른다. 전체 태스크의 $\frac{1}{3}$의 D_i는 $[50\times n, 200\times n]$구간의 균등분포를, C_i는 [10, 100] 구간의 균등분포를 따른다. 나머지 태스크들의 D_i는 $[9\times n, 20\times n]$의 균등분포를, C_i는 [10, 20]의 균등분포를 따른다. B_i는 $C_i \times CSperc$로 하고, $CSperc$를 10%, 20%, 30%, 40%에 대하여 각각 시뮬레이션 한다.

두 번째 태스크 집합의 D_i 및 C_i는 첫 번째 태스크 집합과 동일하고, B_i는 $C_i \times CSperc$로 두고, $CSperc$는 [0~40%] 구간의 균등분포로 한다.

세 번째 태스크 집합의 D_i는 $[10\times n, 5\times n]$의 균등분포, C_i는 [10, 200]의 균등분포로 하며, B_i는 $C_i \times CSperc$로 두고, $CSperc$는 [0~40%]의 균등분포로 한다.

세 개의 태스크 집합 각각에 대하여 n(태스크의 개수)을 10, 20, 40, 80, 100로 변화시키면서 시뮬레이션 한다. 모든 경우에 대하여 100000번의 반복을 하여 k의 최대값을 구하였다. 그 결과는 표 1과 같다.

표 1. 시뮬레이션 결과(k의 최대값)

태스크의 수	첫 번째 태스크 집합				두 번째 태스크 집합	세 번째 태스크 집합
	CSperc =10%	20%	30%	40%		
10	2	3	3	3	3	3
20	2	2	3	3	3	3
40	2	2	3	3	3	3
80	2	2	3	3	3	3
100	2	2	3	3	3	3

4. 슬로우다운 계산 알고리즘의 개선

3장에서 기술한 기존 알고리즘의 분석 결과를 바탕으로 2개의 개선 알고리즘을 제시한다. 기존 알고리즘과 똑같은 슬로우다운을 구하는 시간복잡도가 $O(nlogn)$인 개선 알고리즘1과 기존 알고리즘의 슬로우다운과 거의 같은 근사치를 구하는 시간복잡도가 $O(n)$인 개선 알고리즘2를 제시한다.

4.1 개선 알고리즘1

3.1절의 따름정리 1에 의해 q=1에서 $\eta_1,\eta_2,...,\eta_n$을 구하고, $\eta_1,\eta_2,...,\eta_n$에서 $q_1,q_2,...,q_k$를 구한 다음, 이것을 이용하여 각 태스크의 슬로우다운을 구할 수 있다.

개선된 알고리즘은 그림 2와 같다. 알고리즘의 2)번에서는 q=1일 때 $\eta_1,\eta_2,...,\eta_n$을 계산하고, 3)번에서 안정된(stable) 정렬 알고리즘을 사용하여 정렬한다. 정렬된 결과를 $\eta_{i_1},\eta_{i_2},\eta_{i_3},...,\eta_{i_n}$이라 두고, 비교의 끝을 나타내기 위해 $\eta_{n+1}=-\infty$을 추가 하였다. 4)번에서 $q_1,q_2,...,q_k$를 계산하고, 이를 이용하여 5)번에서 각 태스크의 슬로우다운 값 η_i를 구한다. 이 알고리즘에서 구한 슬로우다운 값은 3.1절에서 기술한 성질에 의하여 기존 알고리즘이 구한 값과 동일하다.

이 알고리즘의 시간 복잡도를 분석하면, 먼저 알고리즘의 1)번은 $\Theta(1)$, 2)번은 $\Theta(n)$, 3)번은 $O(nlogn)$, 4)번은 $O(n)$, 5)번은 <식 5>를 제외한 나머지는 $\Theta(n)$이다. <식 5>의 계산을 k번 반복하는 데 걸리는 시간은 다음과 같다. 분모의 계산을 k번 반복하는데 걸리는 시간은 $\sum_{1\le r<q_{k-1}+1}\frac{1}{\eta_r}\frac{C_r}{D_r}$의 계산에 비례하는 시간이 걸리고, 분자의 계산을 k번 반복하는데 걸리는 시간은 $\sum_{1\le p\le n}\frac{C_p}{D_p}$에 비례하는 시간이 필요하므로 <식 5>를 k번 반복하는 데 걸리는 시간은 $\Theta(n)$이다. 따라서, 개선 알고리즘1의 시간복잡도는 $O(nlogn)$이다. 기존 알고리즘의 시간복잡도 $O(n^2)$보다 빠른 시간복잡도를 가진다.

4.2 개선 알고리즘2

3.2절의 시뮬레이션 결과, $q_1,q_2,...,q_k$에서 k가 최대 3이면 기존 알고리즘과 같은 슬로우다운을 구할 수 있음을 보았다. 따라서, 개선 알고리즘1의 3), 4)을 개선 알고리즘2(그림 3)의 3)과 같이 수정하였다. 이 개선 알고리즘2의 시간복잡도는 $O(kn)$ 즉, $O(n)$의 시간복잡도를 갖는다.

$\{\tau_1,\tau_2,..,\tau_n\}$이 상대 마감 시간에 대해 증가 순으로 정렬되었다고 가정.

1) q = 1;
2) for($i=1;i\le n;i$++)do
$$\eta_i=\frac{B_i}{D_i}+\sum_{q\le p<i}\frac{C_p}{D_p}$$
endfor
3) $\{\eta_1,\eta_2,...,\eta_n\}$을 내림차순으로 정렬(안정된 정렬 알고리즘 사용)
정렬결과를 $\{\eta_{i_1},\eta_{i_2},...,\eta_{i_n}\}$이라 하자. $\eta_{n+1}=-\infty$
4) j = 1;
while($\eta_{i_j}\equiv\eta_{i_{j+1}}$)
j = j + 1;
endwhile
$q_1=i_j$; $k=1$;
while($q_k<n$)
j = j + 1;
while($q_k>i_j$) /* q_k보다 작은 첨자를 가진 η_i들은 의미가 없음 */
j = j +1;
endwhile
while($\eta_{i_j}\equiv\eta_{i_{j+1}}$) /* 다음의 q_k를 찾음 */
j = j + 1;
endwhile
$k=k+1$; $q_k=i_j$;
endwhile
5) q = 1;
for($j=1;j\le k;j$++)
$m=q_j$;
$$\eta_m=\frac{\left(\frac{B_m}{D_m}+\sum_{q\le p\le m}\frac{C_p}{D_p}\right)}{\left(1-\sum_{1\le r<q}\frac{1}{\eta_r}\frac{C_r}{D_r}\right)}$$
(식 5)
for($l=q;l\le m;l$++)
$\eta_l=\eta_m$;
endfor
$q=m+1$
endfor

그림 2. 개선 알고리즘1

5. 결 론

전력은 임베디드 실시간 시스템의 설계 및 동작의 최적화를 위한 중요한 요소 중 하나이다. 임베디드 실시간 시스템에서는 태스크의 마감시간을 유지하면서 전력소비를 최소화시키려고 한다. 전력소비를

$\{\tau_1,\tau_2,..,\tau_n\}$이 상대 마감 시간에 대해 증가 순으로 정렬되었다고 가정.

1) $q = 1;$
2) for($i=1;i \le n;i++$)do
$$\eta_i = \frac{B_i}{D_i} + \sum_{q \le p < i} \frac{C_p}{D_p}$$
endfor
3) $j = 1;\ q = 1;\ k = 3;$
while($q \le n$ and $j \le k$)
$\eta_m = \max_{i=q}^{n}(\eta_i);$
$q_j = m;$
$j = j+1;$
$q = m+1;$
endwhile
$k = j - 1;$
4) $q = 1;$
for($j=1;j \le k;j++$)
$m = q_j;$
$$\eta_m = \frac{\left(\frac{B_m}{D_m} + \sum_{q \le p \le m} \frac{C_p}{D_p}\right)}{\left(1 - \sum_{1 \le r < q} \frac{1}{\eta_r}\frac{C_r}{D_r}\right)}$$
for($l=q;l \le m;l++$)
$\eta_l = \eta_m;$
endfor
$q = m+1$
endfor

그림 3. 개선 알고리즘2

줄이는 것과 마감시간을 만족시키는 것은 서로 상충되므로 전력소비를 최소화하기 위해 전력과 시간을 잘 고려해야 한다. 전력을 효율적으로 사용하기 위한 기존 연구들은 독립된 태스크 집합에 대하여 전력을 고려한 스케쥴링에 관한 것이다. 비록 동적 전압 조절를 이용한 많은 연구가 있었지만, 태스크의 동기화가 존재하는 모델에 대해서는 거의 연구되지 않았다.

본 논문에서는 동기화 제약조건이 있는 태스크들의 슬로우다운을 계산하는 Jejurikar와 Gupta가 제시한 알고리즘을 시간-효율성(즉, 시간복잡도) 측면에서의 개선을 위하여 분석하였다. 수학적 분석 및 시뮬레이션을 통하여 기존 알고리즘의 성질을 제시하고, 이 성질을 이용하여 기존 알고리즘과 같은 성능을 갖는 시간복잡도가 $O(nlogn)$ 및 $O(n)$인 개선된 2개의 알고리즘을 제안하였다. 개선된 알고리즘들은 정적 슬로우다운의 계산에도 효율적으로 사용할 수 있을 뿐만 아니라, 특히 동적 슬로우다운 계산에 효율적이다.

향후 연구과제로 슬로우다운을 구하는 알고리즘들을 실시간 운영체제에 구현하는 연구, 슬로우다운을 구하는 최적의 알고리즘 개발에 관한 연구, 그리고 다중프로세서 시스템으로의 확장 등이 필요하다.

참 고 문 헌

[1] J. W. S. Liu, *Real-Time Systems,* Upper Saddle River, NJ: PrenticeHall, 2000.

[2] G. C. Buttazzo, *Hard Real-Time Computing Systems,* Boston, MA: Kluwer, 1995.

[3] J. Chen and C. Kuo, "Energy-Efficient Scheduling for Real-time Systems on Dynamic Voltage Scaling(DVS) Platforms," in the 13th IEEE International Conf. on Embedded and Real-Time Computing Systems and Applications, 2007.

[4] A. K. Mok, "Fundamental Design Problems of Distributed Systems for Hard Real-Time Environment," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., 1983.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of Np-Completeness,* San Francisco, CA: Freeman, 1979.

[6] J. A. Stankovic, M. Spuri, M. D. Natale, and G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Trans. on Computer,* Vol.28, No.6, pp. 16-25, 1994.

[7] T. P. Baker, "Stack-Based Scheduling of Real-Time Processes," *J. Real-Time Syst.,* Vol.3 No.1, pp. 67-99, 1991.

[8] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. on Compututer,* Vol.39, No.9, pp. 1175-1185, 1990.

[9] R. Jejurikar and R. Gupta, "Energy Aware Task Scheduling with Task Synchronization

for Embedded Real Time Systems," in Proc. Int. Conf. Compilers, Architecture and Synthesis Embedded Systems, pp. 164-169, 2002.

[10] R. Jejurikar and R. Gupta, "Energy Aware EDF Scheduling with Task Synchronization for Embedded Real Time Operating Systems," in Workshop Compilers and Operating System Low Power, pp. 7.1-7.6, 2002.

[11] R. Jejurikar and R. Gupta, "Energy-Aware Task Scheduling With Task Synchronization for Embedded Real-Time Systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol.25, No.6, pp 1024-1037, 2006.

[12] F. Zhang and S. T. Chanson, "Processor Voltage Scheduling for Real-Time Tasks with Non-preemptible Sections," in Proc. IEEE Real-Time Systems Symposium, pp. 235-245, 2002.

[13] Y. Chen, C. Yang, and T. Kuo, "FL-PCP: Frequency locking for Energy-Efficient Real-Time Task Synchronization," the 13th IEEE International Conf. on Embedded and Real-Time Computing Systems and Applications, 2007.

[14] M. Chen and K. Lin, "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems," *Real Time Systems Journal*, Vol.2, No.1, pp. 325-346, 1990.

[15] Intel StrongARM Processor, Intel Inc., http://www.intel.com/design/strong/specupdt/278259.htm.

[16] Intel XScale Processor, Intel *Inc.*, http://developer.intel.com/design/intelxscale/xscale_datasheet4.htm

[17] Transmeta Crusoe Processor, *Transmeta Inc.*, http://www.transmeta.com/crusoe/specs/html

[18] 이재동, 허정연,"태스크 동기화가 필요한 임베디드 실시간 시스템에 대한 효율적인 전압 스케쥴링," 정보과학회논문지:시스템및이론, 제35권, 제 5・5호, pp. 273-283, 2008.

이 재 동

1983년 서울대학교 계산통계학과 이학사
1985년 서울대학교 전산과학전공 이학석사
1995년 서울대학교 전산과학전공 이학박사
1986년~현재 경남대학교 컴퓨터공학부 교수

관심분야 : 실시간 시스템, 멀티미디어 시스템, 임베디드 시스템

김 정 종

1977년 중앙대학교 전자계산학과 이학사
1981년 중앙대학교 전자계산학전공 이학석사
1988년 중앙대학교 전자계산학전공 이학박사
1981년~현재 경남대학교 컴퓨터공학부 교수

관심분야 : 소프트웨어공학, 객체지향방법론, 컴퓨터교육, 인터넷응용