태스크 동기화가 필요한 임베디드 실시간 시스템에 대한 효율적인 전압 스케쥴링

(An Efficient Voltage Scheduling for Embedded Real-Time Systems with Task Synchronization)

이 재 동 † 허 정 연 †

(Jae-Dong Lee) (Jung Youn Hur)

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 이용하므로서 많은 소비 전력을 절약할 수 있다. 본 논문에서는 태스크의 동기화가 필요한 임베디드 실시간 시스템에서 효율적인 전력 소비를 위해 태스크들의 슬로우다운 값을 구하는 휴리스틱 알고리즘들을 제안한다. 기존 알고리즘에서는 상대 마감시간이 작은 태스크의 슬로우다운 값은 상대 마감시간이 크거나 같은 태스크의 슬로우다운 값보다 크거나 같아야 한다는 제약조건을 가지고 있다. 본 논문에서는 이 제약조건을 완화하여 기존 알고리즘과 같은 시간복잡도를 가지면서 전력을 더 작게 소비하는 휴리스틱 알고리즘들을 제시한다. 실험을 통해 소비 전력 면에서 효율적임을 보였다.

키워드 : 동적 전압 조절, 임베디드 실시간 시스템, 태스크 동기화, 슬로우다운

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 propose heuristic algorithms to calculate task slowdown factors for an efficient energy consumption in embedded real-time systems with task synchronization. The previous algorithm has a following constraint : given the tasks are ordered in a nondecreasing order of their relative deadline, the task slowdown factors computed are in a nonincreasing order. In this paper, we relax the constraint and propose heuristic algorithms which have the same time complexity that previous algorithm has and can save more energy. Experimental results show that the proposed algorithms are energy efficient.

Key words : DVS(dynamic voltage scaling), embedded real-time systems, task synchronization, slowdown

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 실시간 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

· 이 연구결과물은 2008학년도 경남대학교 학술연구장려금 지원에 의한 것임

† 종신회원 : 경남대학교 컴퓨터공학부 교수
jdlee@kyungnam.ac.kr
hurj@kyungnam.ac.kr

논문접수 : 2007년 11월 15일
심사완료 : 2008년 2월 19일

정보과학회논문지: 시스템 및 이론 제35권 제6호(2008.6)

전력소비를 줄이는 데 더욱 효율적인 것으로 알려져 있다. 프로세서의 클럭 속도와 전압을 조절하면 전력소비는 줄일 수 있지만 작업(job)의 수행시간은 길어진다. 실시간 임베디드 시스템에서는 태스크의 마감시간을 유지하면서 전력소비를 최소화시키려고 한다. 전력소비를 줄이는 것과 마감시간을 만족시키는 것은 서로 상충 되므로 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 값의 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 정적 이산 슬로우다운의 계산에 초점을 맞춘다. 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. 초기연구 중 Yao 등은 도착시간과 마감시간이 주어진 작업(job)들을 연속적으로 전압을 조절할 수 있는 프로세서에 스케쥴링하는 최적의 오프라인 알고리즘을 제시했다[4]. 제시한 알고리즘의 분석 및 정확성은 EDF 스케쥴러에 기반을 둔다. Bansal 등은 Yao 등이 제시한 알고리즘을 개선하기 위해 더욱 철저한 분석을 하였고[5], Kwon과 Kim은 이산 전압 레벨(discrete voltage level)의 경우에 최적의 슬로우다운 값을 계산하도록 확장했다[6]. Li와 Yao는 [4]에서 제시한 알고리즘을 사용하지 않고 최소의 전력을 사용하는 스케쥴을 생성하는 효율적인 알고리즘을 제시하였다[7]. Quan과 Hu는 고정우선순위 스케쥴링 하에서 같은 문제에 대하여 여러 방법을 제시하였다[8,9]. Yun과 Kim은 이것이 NP-hard 임을 보이고, 효율적인 다항식 시간 알고리즘을 제시하였다[10]. Shin 등은 독립된 주기적 태스크 집합에 대해 슬로우 다운 값을 계산했다. 여기서 RM(Rate Monotonic) 분석에 기초하여 일정 슬로우다운(constant slowdown : 모든 태스크에 똑같이 사용되는 슬로우다운 값)을 계산했다[11]. Gruian은 더 많은 반복을 수행하므로써 각각의 태스크 타입에 대해 더 좋은 슬로우다운 값을 얻을 수 있음을 보였다[12]. EDF 스케쥴링 정책 하에서 프로세서의 이용률을 최대로 하는 일정 슬로우다운은 실행가능함(feasible)을 보였다[13]. 다른 전력소비 특성을 가지는 태스크들에 대한 최적 슬로우다운 값의 계산 방법을 Aydin 등이 제시했다[14]. 태스크의 최악의 수행시간에 비해 일찍 끝나므로 생기는 슬랙(slack)을 재사용하는 동적 슬로우다운 태크닉들이 제시되었다[13,15-17].

비록 동적 전압 조절(dynamic voltage scaling: DVS)이 많이 연구되었지만, 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[18-20], 그 경우의 실행가능성 테스트(feasibility test)에 대해 연구되었다[21,22]. 실행가능성 테스트에 기초한 일정 슬로우다운 값의 계산 방법이 제시되었으며[23-25], 비슷한 연구로 Zhang과 Chanson은 비선점 섹션(section)을 가진 태스크의 슬로우다운을 계산하는 방법을 제시하고, DS(Dual-Speed)알고리즘을 제안하였다[26]. 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[27]. 제시한 알고리즘은 고정 우선순위 태스크들의 집합에 활용할 수 있다. Jejurikar와 Gupta는 동기화 제약조건 하에서 개개의 태스크에 대한 슬로우다운 값을 계산하는 알고리즘을 제시하였다[25]. 이 알고리즘은 EDF와 RM 스케쥴링 정책 둘 다에 사용될 수 있으며, 어떠한 자원 엑세스 정책(PCP[22], DPCP[28], SRP[21])과도 같이 사용할 수 있다. 그러나, 이 알고리즘은 연속 슬로우다운 값을 계산하여 이산 슬로우다운으로 변환한다. 프로세서가 연속 슬로우다운을 지원하면 최적의 전력 사용을 보장할 수 있지만, 이산 슬로우 다운을 지원

하는 프로세서에서는 최적의 전력 사용을 보장할 수 없다. 현재 제공되는 대부분의 프로세서는 전압레벨(즉, 클럭속도의 레벨)이 이산(discrete)이다. 이런 이산레벨을 가지는 프로세서에 이 알고리즘을 사용한다면 최적의 전력 사용을 보장할 수 없다. 본 논문에서는 이산 레벨을 가지는 프로세서에 이 알고리즘을 사용했을 때의 문제점을 예를 들어 보이고, 알고리즘을 수정하여 전력을 더욱 효율적으로 사용할 수 있는 슬로우다운 값을 구하는 휴리스틱 알고리즘 들을 제시한다.

본 논문의 구성은 다음과 같이 구성되어 있다. 2장에서는 시스템 모델 및 기존 알고리즘을 기술하고, 기존 알고리즘의 문제점을 예를 들어 제시한다. 3장에서는 소비 전력를 줄이기 위한 휴리스틱 알고리즘들을 제시하며, 4장에서는 제시한 휴리스틱 알고리즘들의 성능을 비교 분석하고, 5장에서는 결론과 향후연구에 대한 방향을 제시한다.

2. 시스템 모델 및 슬로우다운 계산 알고리즘

이 장에서는 시스템 모델과 기존 슬로우다운 계산 알고리즘을 기술하고, 기존 알고리즘의 문제점을 예를 들어 제시한다.

2.1 시스템 모델

n개의 주기적 실시간 태스크의 집합 $I = \{\tau_1, \tau_2, ..., \tau_n\}$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 최대 프로세서 속도로 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[22]. 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 우선순위의 태스크가 공유 자원을 반납(release)하기를 기다려야 한다면 그 태스크는 블록 되었다고 한다. 자원을 가지고 있는 태스크를 블록킹 태스크(blocking task)라고 한다. 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. 이 논문에서는 시스템의 여유 부분(slack)을 더 잘 이용하기 위해 모든 태스크에 동일한 슬로우다운을 할당하는 것이 아닌 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 변화시키는 데 필요한 시간과 전력은 태스크의 시간과 전력에 비해 아주 작다. 전압 변화를 위한 오버헤드는 구문교환(*context switch*) 오버헤드와 비슷하게 태스크의 실행시간에 포함시킬 수 있다. 본 시스템의 스케쥴링 정책은 EDF로 하고, 자원접근정책은 DPCP[28] 또는 SRP[21]로 한다.

2.2 실행가능성 테스트

2.1절에서 언급한 시스템 모델에서 태스크 집합의 실행가능성 테스트(feasibility test)는 아래 정리와 같다[21,28].

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, 아래 식이 성립하면 태스크 집합은 실행가능하다.

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

2.3 가변 속도 프로세서

Intel strong ARM 프로세서[29], Intel XScale[30], Transmeta Crusoe[31] 등은 가변 전압 및 가변 클럭속도를 지원한다. 전압과 클럭속도는 일대일 대응 관계에 있으므로 프로세서의 속도는 전압을 변화시키므로(즉, 클럭속도를 변화시키므로) 이룰 수 있다[10,20]. 따라서, 프로세서의 슬로우다운은 클럭속도(즉, 전압)을 변화시키는 것이다. 프로세서는 이산전압(즉, 이산 클럭속도)를 지원한다고 가정하고, 최대 클럭속도를 $f_{\max}$으로 둔다. 슬로우다운 값은 최대클럭 속도에 대한 클럭속도의 비로 나타낸다. 프로세서가 지원하는 클럭속도의 레벨을 L개로 두면 프로세서가 지원 하는 클럭속도는 $f_{\min}, f_2, f_3, \ldots, f_{\max}$이다. 따라서 태스크가 가질 수 있는 슬로우다운 값은 $f_{\min}/f_{\max}, f_2/f_{\max}, f_3/f_{\max}, \ldots, 1$이다. 여

기서, 알고리즘의 기술을 위해 슬로우다운의 값을 아래와 같이 정의한다.

정의 1. 프로세서가 지원하는(즉, 태스크가 가질 수 있는) 슬로우다운 값은 $s_1 = f_{\min}/f_{\max}, s_L = 1,$ $s_i = \frac{f_i}{f_{\max}}, (2 \le i \le L-1)$로 둔다.

2.4 슬로우다운 계산을 위한 기존 알고리즘

실행가능성 분석을 기반으로 2.1절에서 정의한 태스크 집합에 대한 연속 슬로우다운 계산은 그림 1과 같다[25]. 여기서 구한 η_i^c는 태스크 τ_i의 연속 슬로우다운 값이다. 이산 슬로우다운을 지원하는 프로세서를 위해 각 태스크의 슬로우다운을 아래와 같이 수정한다.

i = *1, 2, …, n* 에 대하여,

η_i = 0, $\eta_i^c = 0$ 이면

= $s_{j-1} < \eta_i^c \le s_j$을 만족하는 s_j, j = *2, 3, …, L,*

$\eta_i^c \neq 0$ 이면

$\eta_i (1 \le i \le n)$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)[25]을 한다. 클럭속도 상속이란 아래와 같이 태스크의 실행 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 실행중이고 어떤 다른 태스크도 블록하고 있지 않을 때
3) rule 2 : $\eta = (\eta_i, \max_j(\eta_j))$, τ_i가 태스크들을 블록하고 있을 때(τ_j는 블록된 태스크들을 나타냄)

2.5 기존 알고리즘의 문제점

기존 알고리즘의 문제점을 예를 들어 제시한다. 4개의 태스크로 구성된 I = $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ 의 주어진 정보는 표 1과 같다. 프로세서가 지원하는 슬로우다운 레벨은 s_1 = 0, s_2 = 0.1, s_3 = 0.2, …, s_{11} = 1.0으로 가정한다. 이 태스크들의 슬로우다운을 2.4절의 기존 알고리즘으로 구한 값과 다른 슬로우다운 값을 표 2에 제시한다. 기존 알고리즘으로 구한 슬로우다운으로 태스크 집합을 수행할 때 그 태스크 집합은 실행가능하다[25]. 다른 슬로우다운으로 수행되는 태스크 집합에 대해 식 (2)가 성립한다. 이 태스크 집합이 클럭속도 상속을 사용하여 수행된다면 3.1절의 정리 2에 의하여 실행가능하다. 표 2에서 보듯이 기존 알고리즘의 슬로우다운 보다 작거나 같은 다른 실행가능한 슬로우다운이 존재한다. 따라서, 전력

표 1 4개의 태스크에 대한 정보

	D_i	C_i	B_i
τ_1	40	10	4
τ_2	40	15	8
τ_3	80	10	4
τ_4	80	10	0

표 2 실행가능한 슬로우다운

	기존 알고리즘	다른 슬로우다운
τ_1	1.0	0.9
τ_2	1.0	1.0
τ_3	0.9	0.8
τ_4	0.9	0.8

1) $\{\tau_1, \tau_2, \ldots \tau_n\}$이 상대 마감시간에 대해 증가 순(nondecreasing order)으로 정렬되었다고 가정
2) q =1
3) while($q \le n$)do
4) for(i = q; $i \le n$; i++) do
5) $$\eta_i^c = \frac{\frac{B_i}{D_i} + \sum_{q \le p \le i} \frac{C_p}{D_p}}{1 - \sum_{1 \le r < q} \frac{1}{\eta_r^c} \frac{C_r}{D_r}}$$
6) endfor
7) $\eta_m^c = \max_{i=q}^{n}(\eta_i^c)$
8) for(i = q; $i \le m$; i++)do
9) $\eta_i^c = \eta_m^c$
10) endfor
11) q = m + 1
12) endwhile

그림 1 연속 슬로우다운 계산 알고리즘

소비량 역시 더 적은 슬로우다운이 존재한다.

$$\forall i = 1, \ldots, 4 \text{에 대하여} \quad \frac{1}{\eta_i}\frac{B_i}{D_i} + \sum_{k=1}^{i} \frac{1}{\eta_k}\frac{C_k}{D_k} \le 1 \qquad (2)$$

3. 휴리스틱 알고리즘

본 장에서는 기존 알고리즘을 수정하여 전력소비량을 줄일 수 있는 휴리스틱 알고리즘을 제시한다.

3.1 기본 방향

2.5절에서 언급한 기존 알고리즘의 문제점은 슬로우다운 값이 아래의 조건 1을 만족해야한다는 점에서 기인한다.

조건 1. $1 \ge \eta_1 \ge \eta_2 \ge \eta_3 \cdots \ge \eta_n$

이 조건을 완화함으로써 더욱 전력소비량을 줄일 수 있다. 아래 정리 2는 이 조건을 완화하기 위한 것이다.

정리 2. 태스크 $\tau_i = (T_i, D_i, C_i, B_i)$인 태스크 집합 I

$= \{\tau_1, \tau_2, \ldots \tau_n\}$을 슬로우다운 $\eta_1, \eta_2, \ldots, \eta_n$으로 클럭속도 상속을 하면서 수행한다고 하자. 아래 식 (3)이 만족되면 주어진 슬로우다운으로 수행된 태스크 집합은 실행가능하다.

$$\forall i = 1, \ldots, n\text{에 대하여} \quad \frac{1}{\eta_i}\frac{B_i}{D_i} + \sum_{k=1}^{i}\frac{1}{\eta_k}\frac{C_k}{D_k} \le 1$$

$$\text{단, } \eta_i \le 1 \tag{3}$$

증명. 태스크 집합 Γ이 슬로우다운 $\eta_1, \eta_2, \ldots, \eta_n$으로 클럭속도 상속을 하면서 수행되는 태스크 모델은 클럭속도를 최대(즉, 슬로우다운이 1)로 하여 수행되는 태스크 집합 $\Gamma' = \{\tau_1', \tau_2', \ldots, \tau_n'\}$로 리모델링 할 수 있다. 여기서, $\tau_i' = (T_i', D_i', C_i', B_i')$로 둔다. 이때, $T_i' = T_i, D_i' = D_i$, $C_i' = \frac{1}{\eta_i}C_i$이다. τ_i를 블록킹하는 태스크들을 $\tau_{i1}, \tau_{i2}, \ldots, \tau_{im_i}$라 하고, 이 태스크들의 가장 긴 임계영역(critical section)의 실행시간을 $z_{i1}, z_{i2}, \ldots, z_{im_i}$라 하자. 그러면, $B_i' = \max\left\{\frac{1}{\eta_{i1}}z_{i1}, \frac{1}{\eta_{i2}}z_{i2}, \ldots, \frac{1}{\eta_{im_i}}z_{im_i}\right\}$이다. 그러나, 클럭속도 상속을 하고, SRP[21]와 DPCP[28]의 성질에 의해 연결된 블록킹(chained blocking)이 발생하지 않으므로 B_i'은 다음과 같다.

$$B_i' = \max \left\{\frac{1}{\max\{\eta_i, \eta_{i1}\}}z_{i1}, \frac{1}{\max\{\eta_i, \eta_{i2}\}}z_{i2}, \ldots\ldots, \frac{1}{\max\{\eta_i, \eta_{im_i}\}}z_{im_i}\right\}$$

최대값을 $\frac{1}{\max\{\eta_i, \eta_{ij}\}}z_{ij}$라 두자. $\Gamma' = \{\tau_1', \tau_2', \ldots, \tau_n'\}$이 실행가능함을 보이기 위해 아래 식 (4)가 성립함을 보이면 된다.

$$\forall i = 1, 2, \cdots, n\text{에 대하여} \quad \frac{B_i'}{D_i'} + \sum_{k=1}^{i}\frac{C_k'}{D_k'} \le 1 \tag{4}$$

먼저, $\frac{B_i'}{D_i'} = \frac{1}{D_i}\frac{1}{\max\{\eta_i, \eta_{ij}\}}z_{ij}$ 이다. $z_{ij} \le B_i$이고, $\max\{\eta_i, \eta_{ij}\} \ge \eta_i$ 이므로 $\frac{B_i'}{D_i'} \le \frac{1}{D_i}\frac{B_i}{\eta_i} = \frac{1}{\eta_i}\frac{B_i}{D_i}$이다. $i = 1, \cdots, n$에 대하여 $\frac{B_i'}{D_i'} + \sum_{k=1}^{i}\frac{C_k'}{D_k'} \le \frac{1}{\eta_i}\frac{B_i}{D_i} + \sum_{k=1}^{i}\frac{1}{\eta_k}\frac{C_k}{D_k}$이다. 식 (3)에 의하여 $\frac{B_i'}{D_i'} + \sum_{k=1}^{i}\frac{C_k'}{D_k'} \le 1$이다. 따라서, $\Gamma' = \{\tau_1', \tau_2', \ldots, \tau_n'\}$은 실행가능하다. ■

위 정리에 의해 모든 i에 대하여 조건 1은 무시하고 식 (3)이 성립하도록 기존 알고리즘을 수정한다.

3.2 휴리스틱 알고리즘1

기존 알고리즘을 수정하여 휴리스틱 알고리즘1을 그림 2에 제시한다. 이 알고리즘은 기존 알고리즘과 유사

```
1) {τ1, τ2, ....τn}이 상대 마감시간에 대해 증가 순으로
   정렬되어 있다고 가정
2) q = 1
3) while(q ≤ n) do
4)       for(i = q; i ≤ n; i++) do
5)                η = ((Bi/Di + Σ_{q≤p≤i} Cp/Dp)) / (1 − Σ_{1≤r<q} (1/ηr)(Cr/Dr))
6)                ηi = determine_slowdown(η)
7)       endfor
8)       ηm = max_{i=q}^{n}(ηi)
9)       for(i = q; i ≤ m; i++)do
10)               ηi = ηm
11)      endfor
12)      ηq에서 ηm까지의 수정
13)      q = m + 1
14) endwhile

    determine_slowdown(η)
    {
      if(η > 1) not feasible
      if(η == 0) return (0)
      else j = 2, 3, ··· , L에 대하여 s_{j-1} < η ≤ s_j 을
      만족하는 s_j를 return
    }
```

그림 2 휴리스틱 알고리즘1

하나 가장 큰 차이점은 본 알고리즘의 핵심인 12번 라인의 'η_q에서 η_m까지의 수정'이다. 수정 방법을 아래에 기술한다.

이 알고리즘에서 구한 $\eta_q \sim \eta_m$에 대하여 아래 식 (5)가 성립한다.

$$\sum_{1 \le r < q}\frac{1}{\eta_r}\frac{C_r}{D_r} + \frac{1}{\eta_m}\left(\frac{B_m}{D_m} + \frac{C_m}{D_m}\right) + \sum_{q \le p < m}\frac{1}{\eta_p}\frac{C_p}{D_p} \le 1 \tag{5}$$

η_m과 $\eta_p (q \le p \le m-1)$의 값을 더 작은 값으로 수정하여 전력을 절약할 수 있다(2.5절의 예 참조). 다른 태스크에 비해 $\frac{B_i + C_i}{D_i}$가 상대적으로 큰 태스크인 τ_m이 전력 소비량에 더 많은 영향을 줌으로 η_m를 먼저 수정한다. $\alpha = 1 - \sum_{1 \le r \le m}\frac{1}{\eta_r}\frac{C_r}{D_r} - \frac{1}{\eta_m}\frac{B_m}{D_m}$라 두자. α는 식 (5)을 만족 시키면서 η_m를 수정할 수 있는 최대 여유값이다. 따라서, 수정된 값을 t_m이라 두면, 아래 식 (6)을 만족시켜야 한다.

$$\left(\frac{1}{t_m} - \frac{1}{\eta_m}\right)\left(\frac{B_m + C_m}{D_m}\right) \le \alpha \tag{6}$$

식 (6)을 풀면 t_m은 식 (7)을 만족시켜야 한다.

$$t_m \geq \frac{1}{\frac{1}{\eta_m} + \alpha \frac{D_m}{B_m + C_m}} \quad (7)$$

따라서, $t_m \leq s_j$을 만족하는 최소의 s_j를 η_m으로 수정한다.

다음은 $\eta_q, \eta_{q+1}, \cdots, \eta_{m-1}$ 순으로 수정한다. 임의의 태스크 $\tau_k (q \leq k \leq m-1)$에 대한 η_k를 수정하는 방법은 다음과 같다. $\alpha = 1 - \sum_{1 \leq r \leq m-1} \frac{1}{\eta_r} \frac{C_r}{D_r} - \frac{1}{\eta_m} (\frac{B_m + C_m}{D_m})$ 라 두자. α는 식 (5)를 만족시키면서 η_k를 수정할 수 있는 최대 여유 값이다. 수정된 값을 t_k로 두면, $(\frac{1}{t_k} - \frac{1}{\eta_k}) \frac{C_k}{D_k} \leq \alpha$을 만족시켜야 하므로 t_k는 아래 식 (8)을 만족시켜야 한다.

$$t_k \geq \frac{1}{\frac{1}{\eta_k} + \alpha \frac{D_k}{C_k}} \quad (8)$$

따라서, $t_k \leq s_j$을 만족하는 최소의 s_j를 t_k로 선정한다. 이렇게 구한 t_k는 태스크 집합이 실행가능하도록 아래 조건 2를 만족시켜야 한다.

조건 2. $k \leq i \leq m-1$에 대하여 $\frac{1}{\eta_i} \frac{B_i}{D_i} + \sum_{1 \leq r \leq i} \frac{1}{\eta_r} \frac{C_r}{D_r} \leq 1$ 여기서, $\eta_k = t_k$로 둠.

조건 2가 만족되면 $\eta_k = t_k$로 수정하고, 그렇지 않으면 원래 값으로 그대로 둔다.

정리 3. 휴리스틱 알고리즘1로 구한 η_i가 $\eta_i \leq 1$이면, 태스크 집합 $\{\tau_1, \tau_2, \ldots, \tau_n\}$이 $(\eta_1, \eta_2, \ldots, \eta_n)$ 클럭속도로 클럭속도 상속을 하면서 수행하면 이 태스크 집합은 실행가능하다.

증명. 휴리스틱 알고리즘1에서 m으로 선정되는 값들을 $q_0 = 0, q_1, q_2, \ldots, q_k$라고 두자. 임의의 $j(\geq 1)$에 대하여 $\{\tau_1, \ldots, \tau_{q_{j-1}}\}$ 태스크 집합이 $(\eta_1, \eta_2, \ldots, \eta_{q_{j-1}})$로 수행하면 실행가능하다고 가정하자. 그러면 모든 $i = 1, 2, \ldots, q_{j-1}$에 대하여 아래 식 (9)가 성립한다.

$$\frac{1}{\eta_i} \frac{B_i}{D_i} + \sum_{k=1}^{i} \frac{1}{\eta_k} \frac{C_k}{D_k} \leq 1 \quad (9)$$

$a = q_{j-1}, b = q_j$라 두자. $\eta_{a+1}, \ldots, \eta_b$를 구하기 위해 휴리스틱 알고리즘1의 4~7번 라인을 수행하면 식 (10)이 성립한다.

$$a+1 \leq i \leq b\text{에 대하여 } \eta_i \geq \frac{(\frac{B_i}{D_i} + \sum_{a+1 \leq p \leq i} \frac{C_p}{D_p})}{1 - \sum_{1 \leq r \leq a} \frac{1}{\eta_r} \frac{C_r}{D_r}} \quad (10)$$

식 (10)을 정리하면 식 (11)이 성립한다.

$$\frac{1}{\eta_i} \frac{B_i}{D_i} + \sum_{1 \leq r \leq a} \frac{1}{\eta_r} \frac{C_r}{D_r} + \frac{1}{\eta_i} \sum_{a+1 \leq p \leq i} \frac{C_p}{D_p} \leq 1 \quad (11)$$

휴리스틱 알고리즘1의 8~11번 라인에서 $\eta_{a+1} = \eta_{a+2} = \ldots = \eta_b = \max_{i=a+1}^{b}(\eta_i)$로 두었으므로 식 (12)가 성립한다.

$$a+1 \leq i \leq b\text{에 대해서}$$

$$\frac{1}{\eta_i} \frac{B_i}{D_i} + \sum_{1 \leq r \leq a} \frac{1}{\eta_r} \frac{C_r}{D_r} + \sum_{a+1 \leq p \leq i} \frac{1}{\eta_p} \frac{C_p}{D_p} \leq 1 \quad (12)$$

12번 라인에서 $\eta_{a+1} \sim \eta_b$의 수정을 한다. 이때, 항상 식 (12)가 성립하도록 유지한다. 정리 2에 의해 $\{\tau_1, \tau_2, \ldots, \tau_{q_j}\}$태스크 집합을 $(\eta_1, \eta_2, \ldots, \eta_{q_j})$의 클럭속도로 수행하면 실행가능하다. 따라서, 태스크 집합 $\{\tau_1, \tau_2, \ldots, \tau_n\}$이 $(\eta_1, \eta_2, \ldots, \eta_n)$ 클럭속도로 수행하면 이 태스크 집합은 실행가능하다. ■

3.3 휴리스틱 알고리즘2

기본 알고리즘은 휴리스틱 알고리즘1과 같다. 단지 알고리즘1의 12번 라인 'η_q에서 η_m까지의 수정' 부분을 전력 소비량에 영향을 많이 주는 태스크 순서로 수정하였다. 일반적으로, $u_i = \frac{B_i + C_i}{D_i}$가 큰 태스크 일수록 전력 소비량이 크다. 따라서, 'η_q에서 η_m까지의 수정' 부분을 다음과 같이 수정하였다. 먼저, η_m을 3.2절에서 설명한 알고리즘1과 같이 수정한다. 다음으로, u_i가 큰 태스크부터 차례대로 수정한다. 임의의 태스크 $\tau_k (q \leq k \leq m-1)$에 대한 η_k를 수정하는 방법 역시 알고리즘1과 같이 t_k를 구한다. 이렇게 구한 t_k는 태스크 집합이 실행가능하도록 3.2절의 조건 2를 만족시켜야 한다. 조건 2가 만족되면 $\eta_k = t_k$로 수정하고, 그렇지 않으면 원래 값으로 그대로 둔다.

정리 4. 휴리스틱 알고리즘2로 구한 η_i가 ≤ 1이면, 태스크 집합 $\{\tau_1, \tau_2, \ldots, \tau_n\}$이 $(\eta_1, \eta_2, \ldots, \eta_n)$ 클럭속도로 클럭속도 상속을 하면서 수행하면 이 태스크 집합은 실행가능하다.

증명. 정리 3과 유사함.

3.4 시간 복잡도

기존 알고리즘의 시간복잡도는 $O(n^2)$이다[25]. 제안한 알고리즘의 시간복잡도를 제시한다. 먼저 휴리스틱 알고리즘1에서 1번 라인에서 11번 라인까지는 기존 알고리즘처럼 $O(n^2)$으로 처리 가능하다. 12번 라인의 'η_q에서 η_m까지의 수정' 부분의 시간복잡도를 구하기 위해 t를 $m-q+1$로 둔다. η_m의 수정을 위해서는 $O(t)$ 시간 걸린다. 다음으로 $\eta_q, \eta_{q+1}, \cdots, \eta_{m-1}$를 수정하는 데

(t-1)×$O(t)$시간 걸린다. 따라서, 'η_q에서 η_m까지의 수정'을 위해 $O(t^2)$ 걸린다. 그러므로 12번 라인의 전체 시간복잡도는 $O(n^2)$이다. 결론적으로 휴리스틱 알고리즘1의 시간복잡도는 $O(n^2)$이다. 휴리스틱 알고리즘2의 시간복잡도 역시 같은 방법으로 구하면 $O(n^2)$이다.

4. 성능평가

알고리즘의 성능 비교를 위해 시뮬레이션을 이용한다. 소비 전력의 비교를 위해 각 태스크 모델에서 최대의 전력을 필요로 하는 최악의 경우에 대한 전력 소비량만을 측정하여 비교하였다. 최악의 경우는 각 태스크가 최악의 수행시간 동안 수행되며 최대시간 동안 블록되는 경우에 발생한다.

4.1 소비 전력 계산

소비되는 전력의 양 F는 슬로우다운 값 s의 제곱에 비례한다[24]. 본 연구에서는 정확한 전력의 양의 계산보다는 알고리즘의 비교에 초점을 맞추고 있으므로 소비 전력량 F는 아래와 같이 계산한다.

$$P=\sum_{\text{모든태스크}i}(\tau_i\text{의 슬로우다운값})^2 \times(\text{한 하이퍼주기동안의 수행시간})$$

4.2 시스템 모델

먼저, 9개의 태스크로 구성된 시스템 모델을 설정하였다. 각 태스크의 특성은 표 3과 같다.

태스크 τ_i의 최대 블록킹 시간 B_i는 τ_i를 블록킹하는 태스크들의 임계영역 중 가장 긴 임계영역을 가진 태스크 τ_j의 최대 실행시간 ×$CSperc$로 둔다. 예를 들어, B_1은 태스크 τ_3의 최대 임계영역의 값이다. 따라서, C_3 ×$CSperc$로 둔다. B_4와 B_5는 τ_7의 서로 다른 최대 임계영역의 값으로 두어 $B_4=C_7\times CSperc/2$, $B_5=C_7\times CSperc/2$로 둔다. B_6과 B_8도 같은 방법으로 설정한다. 각 태스크들을 블록킹하는 최대 임계영역을 가진 태스크들을 표 4와 같이 설정한다. 예를 들어, τ_1이 최대로 블록킹 되는 시간인 B_1은 τ_3의 최대 임계영역의 수행시간으로 설정하였다. 즉, 앞에서 언급한 것처럼 C_3 ×$CSperc$로 둔다.

위 시스템 모델의 최대 전력소비량은 표 5의 값들의 합으로 구할 수 있다. 여기서, h는 하이퍼 주기, 즉, 태스크들의 모든 주기의 최소공배수이다.

표 4 태스크들을 블록킹하는 최대 임계영역을 가진 태스크들

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9
최대 임계구역을 가진 태스크	τ_3	τ_4	τ_6	τ_7	τ_7	τ_9	τ_8	τ_9	없음

전력소비량 계산 시, B_4와 B_5는 다른 임계영역으로 가정한다. 마찬가지로 B_6과 B_8 역시 다른 임계영역으로 가정한다. 만약, B_4와 B_5가 같은 임계영역이면 위 표의 'τ_7의 최대 임계영역(B_4)의 실행을 위한 전력소비량', 'τ_7의 최대 임계영역(B_5)의 실행을 위한 전력소비량', 'τ_7의 실행을 위한 전력소비량(B_4, B_5제외)'를 아래와 같이 수정해야 한다.

τ_7의 최대 임계영역(B_4)을 위한 전력소비량 :

$$\frac{B_4}{\max\{\eta_4,\eta_5,\eta_7\}}\times(\max\{\eta_4,\eta_5,\eta_7\})^2\times\frac{h}{D_7}$$

τ_7의 최대 임계영역(B_5)을 위한 전력소비량 :

필요 없음

τ_7의 실행을 위한 전력소비량(B_4, B_5제외) :

$$\frac{C_7-B_4}{\eta_7}\times\eta_7^2\times\frac{h}{D_7}$$

마찬가지로 B_6과 B_8이 같은 임계영역이면 위와 비슷하게 수정해야 한다.

4.3 성능비교

위 시스템 모델에서 CSperc를 10%, 20%, 30%, 40%에 대하여 아래 경우에 대하여 시뮬레이션 하였다.

- 최적(optimal) : 모든 가능한 경우에 대하여 조사하여 최적의 해를 구함
- 기존 알고리즘 : Jejurikar 등이 제안한 알고리즘[25]
- HA1 : 휴리스틱 알고리즘1
- HA2 : 휴리스틱 알고리즘2

프로세서가 지원하는 슬로우다운의 값은

L=6(즉, $s_1=0, s_2=0.2, s_3=0.4, \ldots, s_6=1.0$),

L=11(즉,$s_1=0, s_2=0.1, s_3=0.2, \ldots, s_{11}=1.0$)과,

L=21(즉, $s_0=0, s_1=0.05, s_2=0.1, \ldots, s_{20}=0.95, s_{21}=1.0$)

일 때에 대하여 시뮬레이션 한다. 표 6은 4.2절에서 주

표 3 태스크의 특성

태스크	1	2	3	4	5	6	7	8	9
T_i	100	100	300	1500	1500	1700	2500	3000	9000
D_i	100	100	300	1500	1500	1700	2500	3000	9000
C_i	15	15	120	15	28	90	20	15	60

표 5 최대 전력 소비량 계산

	최대 전력 소비량
τ_1의 실행을 위한 전력소비량	$\frac{C_1}{\eta_1}\times\eta_1^2\times\frac{h}{D_1}$
τ_2의 실행을 위한 전력소비량	$\frac{C_2}{\eta_2}\times\eta_2^2\times\frac{h}{D_2}$
τ_3의 최대 임계영역(B_1)의 실행을 위한 전력소비량	$\frac{B_1}{\max\{\eta_1,\eta_3\}}\times(\max\{\eta_1,\eta_3\})^2\times\frac{h}{D_3}$
τ_3의 실행을 위한 전력소비량 (최대 임계영역, 즉, B_1 제외)	$\frac{C_3-B_1}{\eta_3}\times\eta_3^2\times\frac{h}{D_3}$
τ_4의 최대 임계영역(B_2)의 실행을 위한 전력소비량	$\frac{B_2}{\max\{\eta_2,\eta_4\}}\times(\max\{\eta_2,\eta_4\})^2\times\frac{h}{D_4}$
τ_4의 실행을 위한 전력소비량 (최대 임계영역, 즉, B_2 제외)	$\frac{C_4-B_2}{\eta_4}\times\eta_4^2\times\frac{h}{D_4}$
τ_5의 실행을 위한 전력소비량	$\frac{C_5}{\eta_5}\times\eta_5^2\times\frac{h}{D_5}$
τ_6의 최대 임계영역(B_3)의 실행을 위한 전력소비량	$\frac{B_3}{\max\{\eta_3,\eta_6\}}\times(\max\{\eta_3,\eta_6\})^2\times\frac{h}{D_6}$
τ_6의 실행을 위한 전력소비량 (최대 임계영역, 즉, B_3 제외)	$\frac{C_6-B_3}{\eta_6}\times\eta_6^2\times\frac{h}{D_6}$
τ_7의 최대 임계영역(B_4)의 실행을 위한 전력소비량	$\frac{B_4}{\max\{\eta_4,\eta_7\}}\times(\max\{\eta_4,\eta_7\})^2\times\frac{h}{D_7}$
τ_7의 최대 임계영역(B_5)의 실행을 위한 전력소비량	$\frac{B_5}{\max\{\eta_5,\eta_7\}}\times(\max\{\eta_5,\eta_7\})^2\times\frac{h}{D_7}$
τ_7의 실행을 위한 전력소비량(B_4, B_5제외)	$\frac{C_7-B_4-B_5}{\eta_7}\times\eta_7^2\times\frac{h}{D_7}$
τ_8의 최대 임계영역(B_7)의 실행을 위한 전력소비량	$\frac{B_7}{\max\{\eta_7,\eta_8\}}\times(\max\{\eta_7,\eta_8\})^2\times\frac{h}{D_8}$
τ_8의 실행을 위한 전력소비량(B_7 제외)	$\frac{C_8-B_7}{\eta_8}\times\eta_8^2\times\frac{h}{D_8}$
τ_9의 최대 임계영역(B_6)의 실행을 위한 전력소비량	$\frac{B_6}{\max\{\eta_6,\eta_9\}}\times(\max\{\eta_6,\eta_9\})^2\times\frac{h}{D_9}$
τ_9의 최대 임계영역(B_8)의 실행을 위한 전력소비량	$\frac{B_8}{\max\{\eta_8,\eta_9\}}\times(\max\{\eta_8,\eta_9\})^2\times\frac{h}{D_9}$
τ_9의 실행을 위한 전력소비량(B_6, B_8제외)	$\frac{C_9-B_6-B_8}{\eta_9}\times\eta_9^2\times\frac{h}{D_9}$

어진 태스크 모델에 대하여 $L=6$, $CSperc$=0.4일 때의 슬로우다운 값을 나타낸다. 그 결과를 요약하면 아래와 같다.

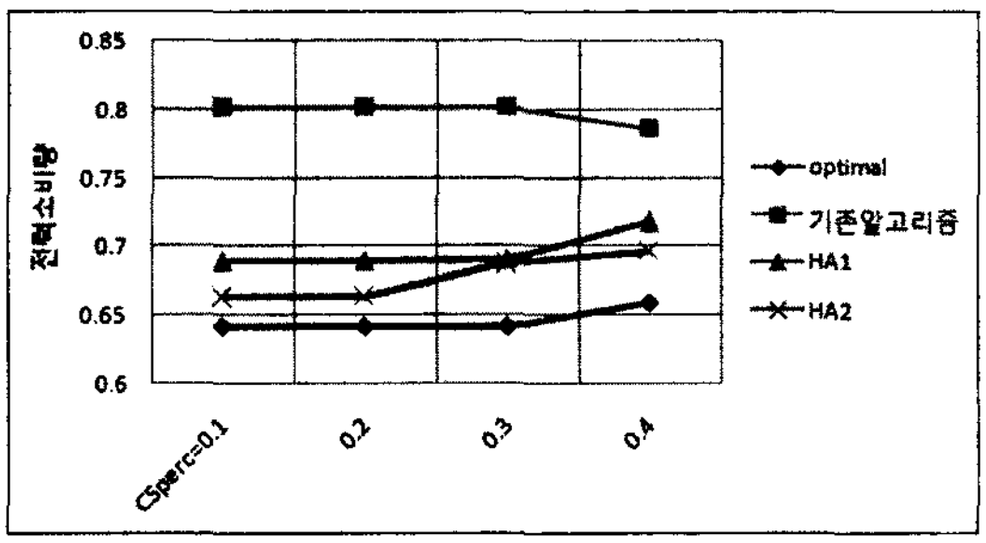
- 기존 알고리즘에 비해 휴리스틱 알고리즘들의 슬로우다운 값이 작음을 알 수 있다.
- 기존 알고리즘의 슬로우다운 값은 태스크의 상대 데드라인이 작을수록 슬로우다운 값이 큰 반면, 최적인 경우와 제안한 휴리스틱 알고리즘들은 상대 데드라인과 무관하다.

그림 3은 성능평가 결과를 보여 주고 있다. 그림에서 나타낸 전력소비량은 아래와 같이 구한 정규화된 전력소비량이다.

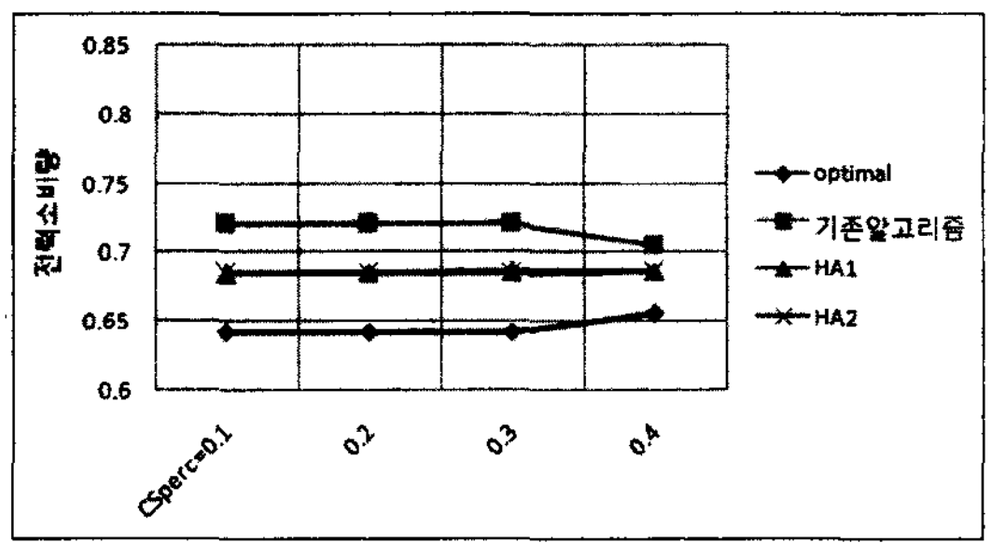
정규화된 전력소비량 = P / (하이퍼 주기 동안 최대의 클럭속도로 수행할 때의 전력소비량)

표 6 태스크들의 슬로우다운 값(L=6, $CSperc$=40%)

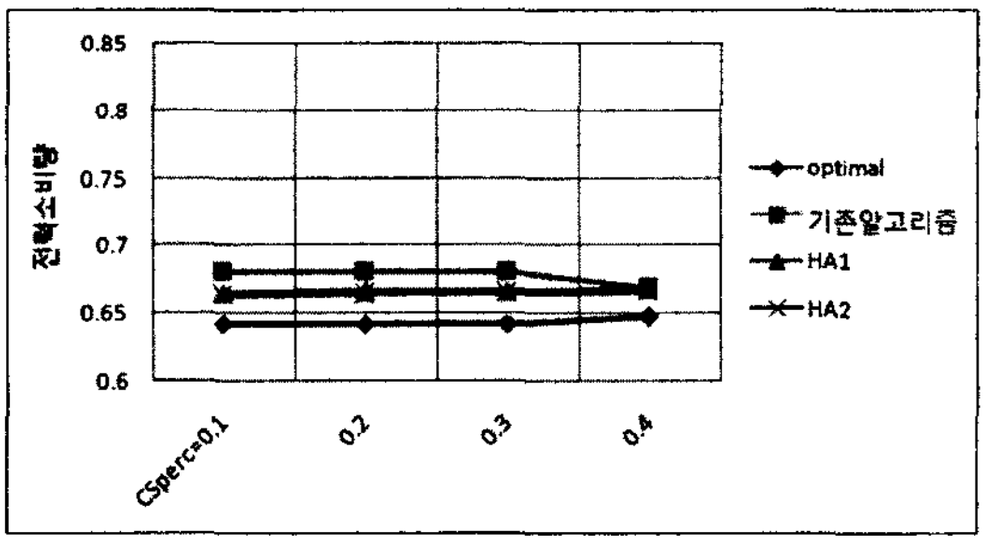
η_i	optimal	기존알고리즘	HA1	HA2
η_1	0.8	1.0	1.0	1.0
η_2	1.0	1.0	0.6	0.8
η_3	0.8	1.0	1.0	0.8
η_4	0.8	0.8	0.2	0.6
η_5	0.6	0.8	0.4	0.6
η_6	0.6	0.8	1.0	0.8
η_7	0.6	0.8	0.8	0.8
η_8	0.6	0.8	0.8	1.0
η_9	0.8	0.8	0.2	0.2



(a) $L = 6$

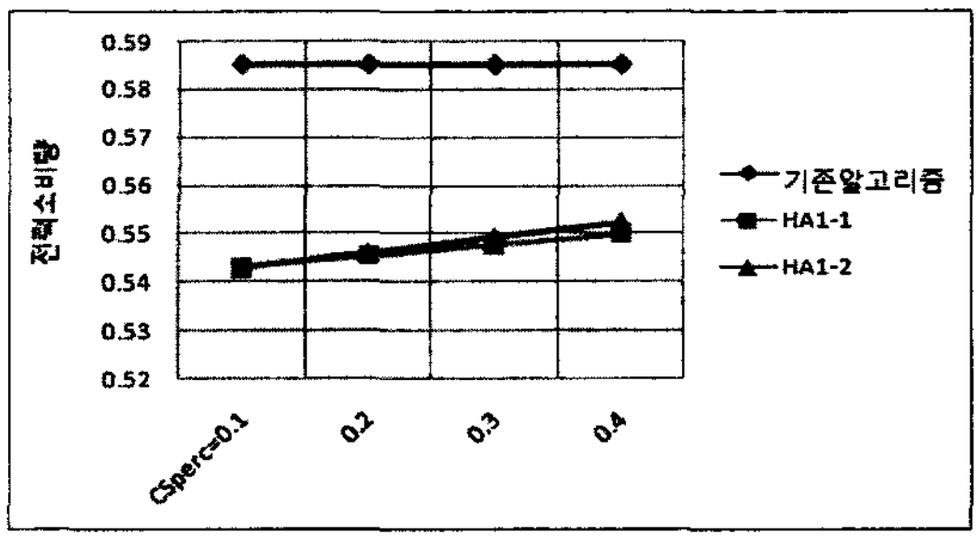


(b) $L = 11$

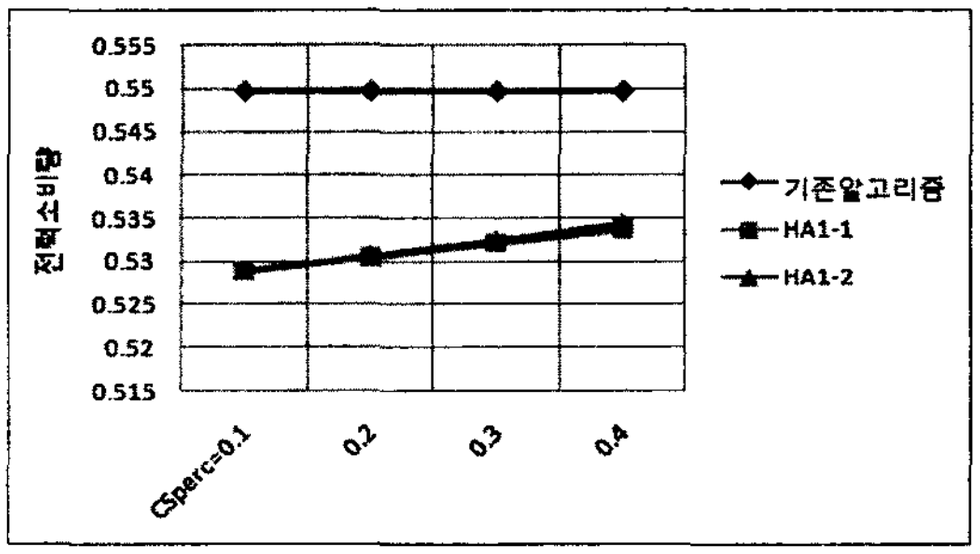


(c) $L = 21$

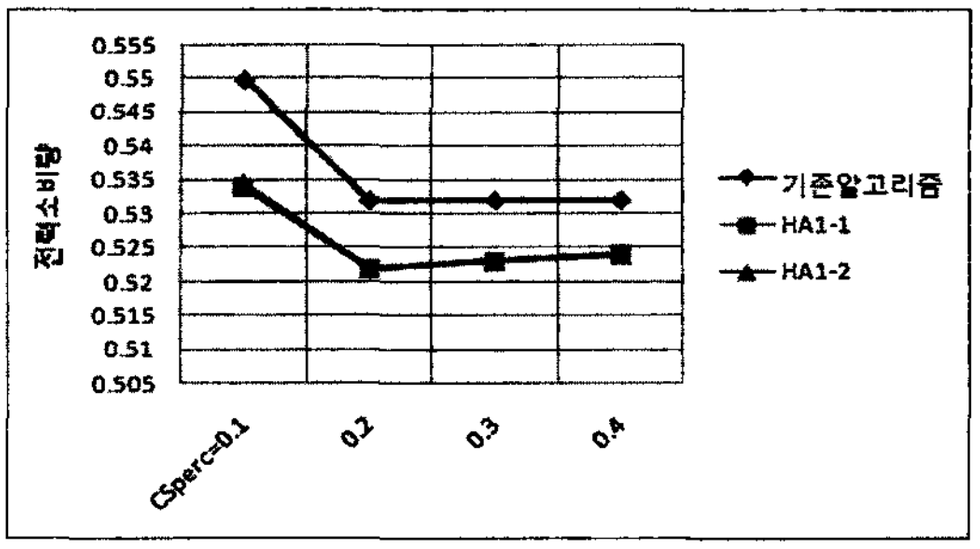
그림 3 성능비교 결과



(a) $L = 6$



(b) $L = 11$



(c) $L = 21$

그림 4 성능분석 결과(일반적인 경우)

성능평가 결과를 요약하면 아래와 같다.

- L이 작은 경우 제안한 알고리즘들이 10% 이상의 전력소비량을 절약함을 알 수 있다. 또한, 전체적으로 제안한 알고리즘들이 기존 알고리즘보다 전력을 작게 소비함을 알 수 있다.
- 휴리스틱 알고리즘1과 알고리즘2는 거의 비슷한 성능을 보였다. 단지, L이 작은 경우, 알고리즘2가 알고리즘1보다 약간 작은 전력을 소비한다.
- 전력소비량은 *CSperc*의 크기에 큰 영향이 없음을 알 수 있다.

기존 알고리즘과 휴리스틱 알고리즘들의 성능 비교를 위해 좀 더 일반화된 시스템 모델로 시뮬레이션 한다. 3개의 태스크의 주기는 [90, 200]의 균등분포(uniform distribution)로 실행시간은 [10, 20]의 균등분포로 하고, 3개의 태스크는 [500, 2000]의 주기와 [10, 100]의 실행시간을, 나머지 3개의 태스크는 [2000, 5000]의 주기와 [10, 500]의 실행시간 분포를 가진다. 각 태스크의 최대 블록킹 시간은 4.2절의 태스크 모델의 표 4와 같은 방법으로 설정한다. 성능 분석 결과는 그림 4와 같다.

분석 결과를 요약하면 아래와 같다.

- 제시한 알고리즘1과 알고리즘2의 성능은 거의 비슷하다.
- 제시한 알고리즘들이 기존 알고리즘에 비해 L=6일 때는 5%, L=11일 때는 2%, L=21일 때는 1% 정도의 전력 소비량의 절약을 가져 온다.

5. 결 론

논문에서는 태스크의 동기화가 필요한 실시간 시스템에서 효율적인 전력 소비를 위해 태스크들의 슬로우다

운 값을 구하는 휴리스틱 알고리즘들을 제안하였다. 태스크들의 스케줄링을 위해서는 EDF 알고리즘을 사용하고 자원 엑세스 정책으로는 SRP와 DPCP를 사용한다. 이런 환경에서의 슬로우다운 값을 구하는 기존 알고리즘에서는 상대 마감시간이 작은 태스크의 슬로우다운 값은 상대 마감시간이 크거나 같은 태스크의 슬로우다운 값보다 크거나 같아야 한다는 제약조건을 가지고 있다. 본 논문에서는 이 제약조건을 완화하여 기존 알고리즘과 같은 시간복잡도를 가지면서 전력을 더 작게 소비하는 휴리스틱 알고리즘들을 제시했다. 실험을 통해 소비 전력 면에서 1~10%의 전력을 절약할 수 있음을 보였다.

향후 연구과제로 제시한 휴리스틱 알고리즘들을 실시간 운영체제에 구현하여 실제 환경에서의 성능 평가가 필요하며, 최적의 해에 근사하는 다항식 시간을 가지는 알고리즘 개발이 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] F. Yao, A. J. Demers, and S. Shenker, *"A Scheduling Model for Reduced CPU Energy,"* in *Proc. IEEE Symposium Foundations Computer Science*, pp. 374-382, 1995.

[5] N. Bansal, T. Kimbrel, and K. Pruhs, *"Dynamic Speed Scaling to Manage Energy and Temperature,"* in *Proc. the Symposium on Foundation of Computer Science*, pp. 520-529, 2004.

[6] W. Kwon and T. Kim. *"Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors,"* in *Proc. Design Automation Conference*, pp. 125-130, 2003.

[7] M. Li and F. Yao, *"An Efficient Algorithm for Computing Optimal Discrete Voltage Schedules," SIAM J. Computer*, Vol.35, No.3, pp 658-671, 2005.

[8] G. Quan and X. Hu, *"Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors,"* in *Proc. Design Automation Conference*, pp. 828-833, 2001.

[9] G. Quan and X. Hu, *"Minimum Energy Fixed-Priority Scheduling for Variable Voltage Processors,"* in *Proc. Design Automation and Test Europe*, pp. 782-787, 2002.

[10] H. Yun and J. Kim, *"On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-Time Systems," ACM Trans. on Embeddded Computing Systems*, Vol.2, No.3, pp. 393-430, 2003.

[11] Y. Shin, K. Choi, and T. Sakurai, *"Power Optimization of Real-Time Embedded Systems on Variable Speed Processors,"* in *Proc. Int. Conf. Computer Aided Design*, pp. 365-368, 2000.

[12] F. Gruian, *"Hard Real-Time Scheduling for Low-Energy using Stochastic Data and DVS Processors,"* in *Proc. Int. Symposium Low Power Electronics and Design*, pp. 46-51, 2001.

[13] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez, *"Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems,"* in *Proc. IEEE Real-Time Systems Symposium*, pp. 95-105, 2001.

[14] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez, *"Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics,"* in *Proc. EuroMicro Conference Real-Time Systems*, pp. 225-232, 2001.

[15] P. Pillai and K. G. Shin, *"Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems,"* in *Proc. 18th Symposium Operating Systems Principles*, pp. 89-102, 2001.

[16] W. Kim, J, Kim, and S. L. Min, *"A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems using Slack Time Analysis,"* in *Proc. Design Automation and Test Europe*, pp. 788-794, 2002.

[17] R. Jejurikar and R. Gupta, *"Dynamic Slack Reclamation with Procrastination Scheduling in Real-Time Embedded Systems,"* in *DAC*, pp 111-116, 2005.

[18] 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.

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

[20] 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.

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

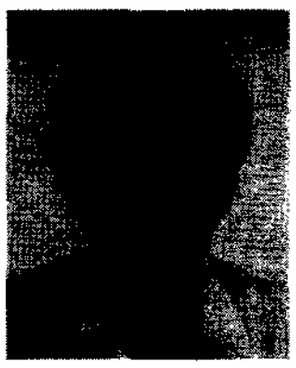
[22] L. Sha, R. Rajkumar, and J. P. Lehoczky, *"Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Trans. on Compututer*, Vol.39, No.9, pp. 1175-1185, 1990.

[23] 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.

[24] 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.

[25] 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.
[26] 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.
[27] 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.
[28] M. Chen and K. Lin, *"Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems," Real Time Systems Journel*, Vol.2, No.1, pp. 325-346, 1990.
[29] Intel StrongARM Processor, *Intel Inc.*, http://www.intel.com/design/strong/specupdt/278259.htm
[30] Intel XScale Processor, Intel *Inc.*, http://developer.intel.com/design/intelxscale/xscale_datasheet4.htm
[31] Transmeta Crusoe Processor, *Transmeta Inc.*, http://www.transmeta.com/crusoe/specs/html



이 재 동

1983년 서울대학교 계산통계학과 이학사 1985년 서울대학교 전산과학전공 이학석사. 1995년 서울대학교 전산과학전공 이학박사. 1986년~현재 경남대학교 컴퓨터공학부 교수. 관심분야는 실시간 시스템, 멀티미디어 시스템, 임베디드 시스템

허 정 연

1970년 경북대학교 물리학과 이학사. 1982년 인하대학교 전자공학과 공학석사. 1990년 영남대학교 전자공학과 공학박사. 1983년~현재 경남대학교 컴퓨터공학부 교수 관심분야는 병렬처리, 컴퓨터비젼, 패턴인식