실행 시간 프로파일을 이용한 저전력 경성 실시간 프로그램용 동적 전압 조절 알고리즘

(A Dynamic Voltage Scaling Algorithm for Low-Energy Hard Real-Time Applications using Execution Time Profile)

신 동 군 † 김 지 홍 ††
(Dongkun Shin) (Jihong Kim)

요 약 태스크내부에서 공급 전압을 조절하는 태스크내 전압 스케쥴링(IntraVS)은 저전력 프로그램을 구현하는 데 효과적인 방법이다. 본 논문에서는 경성 실시간 응용프로그램에서 평균 실행 시간에 대한 정보를 이용하여 전력 소모를 효과적으로 줄이는 새로운 태스크내 전압 스케쥴링 알고리즘을 제시한다. 최악 실행 시간을 사용하여 전압 조절의 결정을 내렸던 기존의 태스크내 전압 스케쥴링과는 달리, 제안된 알고리즘은 평균 실행 시간에 바탕을 두고 실행 속도를 조절함으로써 주어진 시간 제약 조건을 만족시키면서도 기존 방법보다 에너지 효율성을 높일 수 있다. MPEG-4 디코더를 이용한 실험 결과, 제안된 알고리즘은 기존의 태스크내 전압 스케쥴링에 비해서 전력 소모를 최대 34% 감소시켰다.

키워드 : 동적 전압 조절, 저전력, 실시간

Abstract Intra-task voltage scheduling (IntraVS), which adjusts the supply voltage within an individual task boundary, is an effective technique for developing low-power applications.

In this paper, we propose a novel intra-task voltage scheduling algorithm for hard real-time applications based on average-case execution time. Unlike the conventional IntraVS algorithm where voltage scaling decisions are based on the worst-case execution cycles, the proposed algorithm improves the energy efficiency by controlling the execution speed based on average-case execution cycles while meeting the real-time constraints. The experimental results using an MPEG-4 decoder program show that the proposed algorithm reduces the energy consumption by up to 34% over conventional IntraVS algorithm.

Key words : dynamic voltage scaling, low-power, real-time

1. 서 론

정보통신 기술이 발달함에 따라 휴대전화나 개인용 휴대 단말기(PDA), 그리고 동영상 휴대전화와 같은 이동용 시스템(mobile system)의 중요성은 날로 증가하고 있다. 한 번 배터리를 충전했을 때 얼마나 오랫동안 사용할 수 있는지를 나타내는 연속 동작 가능 시간은 상업적인 이동용 시스템에 있어서 가장 중요한 성능 척도의 하나이며, 이동용 시스템의 전력 소모를 줄이는 것은 VLSI 시스템 설계에서 중요한 요소로 부각되고 있다. 특히, VLSI 시스템이 고성능, 고집적화 되어감에 따라 전력 소모는 지수적으로 증가하는 반면에 배터리의 전력 용량은 산술적인 증가에 그치고 있어서, 배터리 자체 개량보다는 VLSI 시스템의 전력 소모를 줄이는 저전력 기법이 중점적으로 연구되는 추세이다. 고성능 마이크로프로세서와 같은 비이동용 시스템에 있어서도 전력 소모는 중요한 설계 목표의 하나가 된다. 높은 전력 소모는 VLSI 내부에서 많은 열을 발생시켜 반도체의 성능 저하나 기능 이상, 심지어는 고장을 일으키기도 하기 때문이다. 이러한 문제점들 때문에 최근 들어 소프트웨어 및 하드웨어 측면에서 다양한 저전력 기법이 요구되고 있다.

일반적인 VLSI 시스템의 전력 소모는 CMOS 회로의

· 본 연구는 한국과학재단 목적기초연구(R01-2001-00360)지원으로 수행되었음.

† 비 회 원 : 서울대학교 전기 컴퓨터 공학부
sdk@davinci.snu.ac.kr
†† 종신회원 : 서울대학교 전기 컴퓨터공학부 교수
jihong@davinci.snu.ac.kr

논문접수 : 2001년 6월 14일
심사완료 : 2002년 9월 24일

동적 전력 소모(dynamic power)가 대부분이며, 이때의 전력 소모 E는 $E \propto C_L \cdot N_{cycle} \cdot V_{DD}^2$ [1]로 주어진다. 여기서 C_L은 CMOS 회로의 부하 커패시턴스(load capacitance), N_{cycle}은 프로그램이 실행된 사이클 수, 그리고 V_{DD}는 공급 전압(supply voltage)을 의미한다. 전력 소모 E가 V_{DD}의 제곱에 비례하기 때문에 공급 전압 V_{DD}를 낮추는 것은 전력 소모를 줄이는 데 매우 효과적인 방법이다. 그러나 공급 전압을 낮추면 클록 속도도 낮아지는 데, 그 이유는 CMOS의 회로 지연 시간 T_D가 $V_{DD}/(V_{DD}-V_T)^{\alpha}$에 비례하기 때문이다. 여기서 V_T는 임계 전압, α는 속도 포화 계수(velocity saturation index)[2]를 나타낸다.

실시간 시스템(real-time system)의 경우, 시스템의 최대 클록 속도는 모든 태스크를 주어진 마감 시간(deadline)이내에 끝낼 수 있는 클록 속도보다 크거나 같아야 실시간 동작을 보장할 수 있다. 이때, 각 태스크의 동작 상태를 살펴가면서 마감 시간 제약 조건을 만족하는 가장 낮은 클록 속도까지 시스템의 클록 속도를 조절해가면서 낮출 수 있으며, 클록 속도에 따른 공급 전압도 함께 낮추어서 전력 소모를 크게 줄일 수 있다. 이것이 동적 전압 조절(dynamic voltage scaling: DVS) 기법의 기본 개념이다. 예를 들어 그림 1(a)와 같이 어떤 태스크가 50MHz의 클록 속도와 5.0V의 공급 전압을 가진 프로세서 상에서 실행된다고 가정하자. 만약 이 태스크가 실행되는데 5×10^5사이클이 걸리고 마감 시간 조건이 25msec이라면, 프로세서는 주어진 태스크를 10msec만에 끝낼 수 있고 태스크의 마감 시간까지 15msec의 유휴 시간(idle time)을 가지게 된다. 그러나 그림 1(b)에서와 같이 클록 속도와 공급 전압이 20MHz와 2.0V로 낮아지면 유휴 시간 없이 주어진 태스크를 마감 시간에 맞추어서 끝낼 수 있고 전력 소모는 $2.0^2/5.0^2$=16%로 낮아지게 된다.

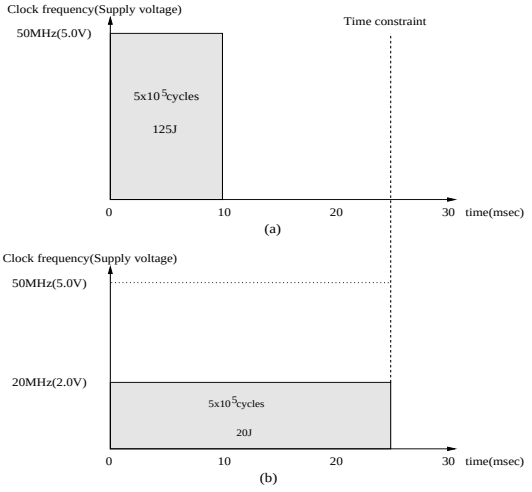

그림 1 전압 조정의 예; (a) 125J의 전력을 소비하며 15msec의 유휴시간을 가지는 경우, (b) 20J의 전력을 소비하며 유휴시간이 없는 경우

1.1 동적 전압 조절(DVS) 기법

동적 전압 조절 기법에서 가장 중요한 부분은 실시간 시스템에서 요구되는 마감시간 제약 조건(deadline constraint)을 만족시키면서 공급 전압을 어떻게 효과적으로 조절하느냐 하는 것인데, 공급 전압 조절을 어느 수준에서 수행하느냐에 따라서 크게 두 종류로 구분된다. *태스크간 전압 스케쥴링*(InterVS)[3,4,5,6]이 각각의 태스크 단위로 공급 전압을 조절하여 하나의 태스크내부에서는 동일한 공급 전압을 사용하는 반면, *태스크내 전압 스케쥴링*(IntraVS)[7,8]은 태스크내부에서 공급 전압을 조절하여, 하나의 태스크내에서도 공급 전압이 여러 번 바뀌게 된다. 최근까지는 태스크간 전압 스케쥴링 기법이 주로 연구되었지만, 실제로 실시간 시스템에 적용되는 과정에서는 몇 가지의 문제점을 보여주고 있다.

태스크간 전압 스케쥴링에서는 OS 내의 태스크 스케쥴러가 태스크의 공급 전압을 결정하기 때문에 전압 스케쥴링을 적용하려면 OS 자체를 수정해야 하며, 각각의 태스크마다 공급 전압이 하나의 고정값으로 결정되기 때문에 단일 태스크 환경에는 적용될 수 없다[7,8]. 소형 내장형 이동 시스템(embedded mobile system)에서는 대부분의 응용프로그램이 단일 태스크 모델에 기반을 두고 있다는 것을 고려할 때, 이 문제는 태스크간 전압 스케쥴링 기법이 실제 내장형 시스템에 적용되는 데 큰 걸림돌로 작용한다. 다중 태스크 환경에서도 만약 하나의 태스크가 전체 실행 시간의 대부분을 차지할 경우에는 태스크간 전압 스케쥴링 기법이 공급 전압을 효과적으로 낮추지 못하는 경우가 발생하기도 한다[8]. 태스크내 전압 스케쥴링은 태스크간 전압 스케쥴링의 이러한 한계점을 극복하기 위해서 제안되었다. 태스크내 전압 스케쥴링은 다른 실행 경로에 따른 실행 시간 변화로 인한 모든 유휴 시간(idle time)을 이용하기 때문에 스케쥴된 프로그램이 실행을 마쳤을 때 전혀 유휴시간이 남지 않게 되고 에너지 효율성도 크게 향상시킨다. 또한 OS가 공급 전압을 조절하지 않기 때문에 기존의 OS를 수정할 필요가 없다는 장점도 가지고 있다.

1.2 연구의 의의

본 논문은 경성 실시간(hard real-time) 응용프로그램

의 평균 실행 시간(Average Case Execution Time: ACET)에 대한 정보를 이용하여 기존의 태스크내 전압 스케쥴링의 에너지 효율성을 크게 높인 새로운 알고리즘을 제시한다. 최악 실행 시간(Worst Case Execution Time: WCET)을 근거로 공급 전압을 조절하는 기존의 태스크내 전압 스케쥴링[8]과 달리, 제시되는 알고리즘은 평균 실행 경로(Average Case Execution Path: ACEP)를 이용하여 실행 속도를 조절한다. 여기에서 평균 실행 경로란 태스크내에서 프로그램 코드가 수행되는 여러 경로중에서 예제 프로그램이 수행될 때 가장 자주 실행되는 경로를 의미한다. 제안된 알고리즘은 프로그램 실행 시에 가장 선택될 확률이 높은 평균 실행 경로의 에너지 감소에 최적화되어 있기 때문에 기존의 알고리즘보다 전체 에너지를 좀 더 효과적으로 감소시킬 수 있다. 기존의 태스크내 전압 스케쥴링은 경성 실시간 시스템의 마감시간 제약 조건을 만족시키기 위해서 프로그램의 남은 부분이 항상 최악 실행 경로(Worst Case Execution Path:WCEP)로 수행된다고 가정하고 공급 전압을 결정하지만, 제안하는 알고리즘은 프로그램의 남은 부분이 평균 실행 경로로 수행된다고 가정하고 전력 소모가 최소가 되도록 공급 전압을 결정하면서도 마감시간 제약 조건을 완벽하게 지킨다는 것이 가장 큰 특징이다. MPEG-4 디코더 프로그램을 사용한 실험 결과, 제안된 알고리즘은 기존의 태스크내 전압 스케쥴링과 비교하여 전력 소모를 최대 34% 감소시킬 수 있었다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 기존의 태스크내 전압 스케쥴링 기법에 대하여 설명하고, 3장에서는 제안되는 새로운 태스크내 전압 스케쥴링에 대하여 자세히 설명한다. 4장에서는 MPEG-4 디코더를 이용한 실험 결과가 논의되며, 5장에서는 결론과 향후 연구에 대한 방향을 제시한다.

2. 기존 태스크내 전압 스케쥴링 알고리즘

경성 실시간 태스크에 있어서 태스크내 전압 스케쥴링 알고리즘의 목표는 시간 제약 조건을 만족시키면서도 전력 소모를 최소화하도록 각각의 기본 블록(basic block)에 적당한 클록 속도를 할당하는 것이다. 기존의 태스크내 전압 스케쥴링에 대해서 살펴보기에 앞서, 본 논문에서는 태스크가 수행될 하드웨어 플랫폼인 가변 전압 프로세서(variable voltage processor)에 대해서 다음과 같은 가정을 한다. (1) 가변 전압 프로세서는 change_f_V(f_{CLK}) 라는 특별한 명령어를 제공한다. 이 명령어는 프로세서의 클록 속도 f_{CLK}와 이에 따른 해당

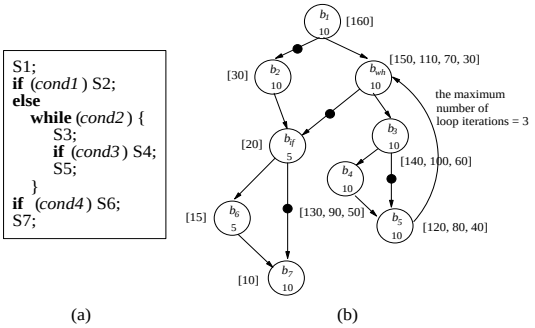

그림 2 예제 프로그램; (a) 예제 실시간 프로그램 P와 (b) CFG G_P

하는 공급 전압 V_{DD}를 조절한다. (2) f_{CLK}와 V_{DD}는 가변 전압 프로세서가 정상적으로 동작할 수 있는 최대값과 최소값 범위 내에서 임의의 연속된 값으로 조절될 수 있다. (3) 가변 전압 프로세서가 클록 속도와 공급 전압을 바꿀 때, 클록과 공급 전압을 바꾸는 데는 일정한 사이클이 소요되며, 이 시간 동안에 가변 전압 프로세서는 실행을 멈춘다.[1)]

그림 2(a)에 나타난 것과 같이 2μsec의 마감시간을 가진 경성 실시간 프로그램 P를 생각해보자. 프로그램 P의 제어 흐름 그래프 (CFG) G_P는 그림 2(b)에 나타나 있다. G_P에서 각각의 노드(node) b_i는 P의 기본 블록을 나타내며 각 에지(edge)는 기본 블록들간의 제어 흐름 의존성(control flow dependency)을 나타낸다. 각 노드 내부에 있는 숫자는 해당 기본 블록 b_i을 실행하는 데 필요한 사이클 수인 $C_{ik}(b_i)$를 나타낸다. b_5에서 b_{wh}로의 백 에지(back edge)는 프로그램 P의 **while** 반복문을 나타낸다.

프로그램에서 **while** 반복문의 최대 반복 횟수가 사용자에 의해서 3으로 주어졌다고 가정할 때, WCET 분석 도구를 사용하여 최악 실행 경로(WCEP)를 찾으면[2)]

1) 이 가정은 클록 속도와 공급 전압을 바꿀 때, 프로세서의 실행 상태를 나타낸다. 트랜스메타(Transmeta)사의 Crusoe와 같은 프로세서의 경우는 전압 변경동안에도 계속 실행할 수 있지만 여기서는 편의상 프로세서가 멈춘다고 가정한다. 본 논문의 연구는 클록과 전압의 변경시 오버헤드에 대한 모델링을 약간 수정하여 두 가지 종류의 프로세서를 모두 지원할 수 있다.

2) 마감시간(deadline)과 같은 엄격한 시간 제약 조건을 가진 경성 실시간 시스템을 개발할 때, 요구되는 시간 제약 조건을 보장하기 위해서 태스크의 최악 실행 시간(WCET)이 미리 계산되는데, 기존에 나와있는 많은 WCET 분석 도구[9]는 사용자가 루프의 최대 반복 횟수 같은 기본적인 값만 입력하면 스스로 경성 실시간 프로그램을 분석하여 최악 실행 경로(WCEP) 및 최악 실행 시간을 안전하고 정확하게 예측해준다.

p_{worst}=(b_1, b_{wh}, b_3, b_4, b_5, b_{wh}, b_3, b_4, b_5, b_{wh}, b_3, b_4, b_5, b_{wh}, b_{if}, b_6, b_7)를 구할 수 있으며, 이때의 실행 사이클인 160 사이클이 최악 실행 사이클 수(Worst Case Execution Cycles: WCEC)가 된다. 만약 목표 프로세서가 최대 80MHz의 클록 주파수로 동작한다면, 프로그램 P는 유휴시간(idle time)을 발생시키지 않으며 실행을 2 μsec에 끝낸다. 여기서 실행시간 대신에 실행 사이클수를 사용했는데, 이것은 가변 전압 프로세서에서는 클록 속도가 조절됨에 따라 실행 시간은 변하지만 실행 사이클 수는 변하지 않기 때문이다.

태스크내 스케쥴링 방법은 서로 다른 실행 경로 사이에는 실행 시간의 차이가 크다는 점에 착안하고 있다. 그림 2(b)에 있는 예제 프로그램을 보면 32개의 서로 다른 경로가 존재한다. 최악 실행 경로 p_{worst}는 160 사이클이 걸리지만 32개중에서 8개의 실행 경로는 80 사이클이 채 걸리지 않는다. 만약 프로그램을 시작할 때 이런 짧은 경로를 실행할 것이라는 것을 알 수 있다면 클록 속도를 낮추어 상당한 양의 에너지를 줄일 수 있다.

속도 조절을 위해서, 태스크내 전압 스케쥴링은 최악 실행 시간에 대한 정적(static) 프로그램 분석 기법을 이용하여 적응적(adaptive) 접근법을 사용한다. 기본 블록 b_i에서의 잔여 최악 실행 사이클수(Remaining Worst Case Execution Cycles: RWEC) $C_{RWEC}(b_i)$를 b_i로부터 출발하는 모든 실행 경로 중에서 가장 긴 경로의 수행 사이클 수로 정의하면, WCET 분석 도구를 약간 수정하여 각각의 기본 블록 b_i에 대해서, *컴파일 시간*에 $C_{RWEC}(b_i)$를 구할 수 있다. 그림 2(b)에서 기호 [ ]는 각 기본 블록의 $C_{RWEC}(b_i)$ 값을 나타낸 것이다. **while** 반복문과 관련 된 기본 블록(즉, b_{wh}, b_3, b_4, b_5)에 대해서는 **while** 반복문이 최대 3번 반복되기 때문에 여러 개의 $C_{RWEC}(b_i)$ 값이 나타나 있다.

계산된 $C_{RWEC}(b_i)$ 값을 이용하면 CFG G_P에서 $C_{RWEC}(b_i)-C_{EC}(b_i)\neq C_{RWEC}(b_j)$인 에지 (b_i, b_j)를 찾을 수 있다. 예를 들어, 그림 2(b)에서 이러한 에지로 (b_1, b_2), (b_{wh}, b_{if}), (b_{if}, b_7)와 (b_3, b_5)의 4개가 존재하며 그림에서 ● 기호로 표시되어 있다. 이 경우에는 에지 (b_i, b_j)라는 실행 경로가 최악 실행 경로가 아니며, 따라서 클록 속도 및 공급 전압을 조절할 수 있는 가능성이 있다는 것을 의미한다. 이렇게 표시된 에지들은 전압 조절 에지 (Voltage Scaling Edge: VSE)의 후보로 선택된다. 만약 에지 (b_i, b_j)가 VSE로선택되면, 프로그램의 제어 흐름이 b_i에서 b_j로 갈 때 클록 속도 및 공급 전압이 바뀐다는 것을 의미한다. 예를 들어, 기본 블록 b_1 이후에 b_2가 실행된다면 남아 있는 작업량이 1/5($C_{RWEC}(b_2)$에 대한 [$C_{RWEC}(b_1)-C_{EC}(b_1)$]의 비율)로 줄었기 때문에 클록 속도는 낮추어 진다. VSE는 B형과 L형의 두 가지 종류로 구분되는 데, B형 VSE는 **if** 문과 같은 조건문내의 CFG 에지(예를 들어, (b_1, b_2), (b_{if}, b_7)와 (b_3, b_5))에 해당하며 L형 VSE는 CFG의 반복문 출구 부분(예를 들어, (b_{wh}, b_{if}))에 해당한다.

선택된 VSE에서 바뀌는 새로운 클록 속도는 앞으로 남아있는 작업량이 얼마나 많이 줄어 들었는가에 따라 결정된다. 예를 들어, 프로그램의 제어 흐름이 VSE (b_i, b_j)에 도착했을 때 클록 속도는 잔여 작업량이 $[C_{RWEC}(b_i)-C_{EC}(b_i)]-C_{RWEC}(b_j)$만큼 줄어들었기 때문에 낮추어 진다. 클록 속도 S로 b_i가 실행된 후에, 클록 속도는 잔여 작업량의 감소를 반영하여 b_j에 대한 새로운 클록 속도는 $S\times\frac{C_{RWEC}(b_j)}{C_{RWEC}(b_i)-C_{EC}(b_i)}$로 된다. 여기서, $\frac{C_{RWEC}(b_j)}{C_{RWEC}(b_i)-C_{EC}(b_i)}$를 에지 $b_i\rightarrow b_j$에 대한 속도 변경 비율 (Speed Update Ratio: SUR)이라고 하며 $SUR(b_i\rightarrow b_j)$로 나타낸다.

그림 3은 그림 2에 있는 예제 프로그램에 대해 태스크내 전압 스케쥴링이 사용되었을 때와 사용되지 않았을 때 어떻게 클록 속도와 전압이 변하는 지를 비교하여 보여주고 있다. 에지 (b_1, b_2)에서 클록 속도는 80 MHz로부터 16 MHz(= 80 MHz $\times\frac{30}{160-10}$)로 바뀐다. 유휴상태에서 전력이 전혀 소모되지 않는다고 하며 $E\propto C_L\cdot N_{cycle}\cdot V_{DD}^2$라고 가정할 때, 프로그램 실행

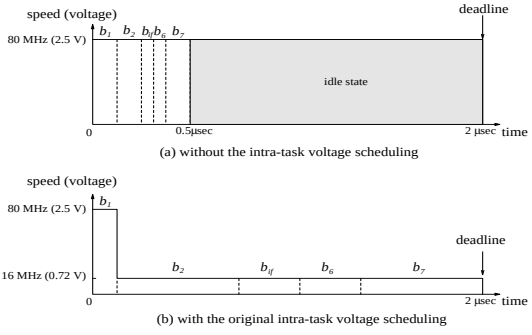

그림 3 태스크내 스케쥴링에 의한 클록 속도와 전압의 변화

이 $p_1=(b_1, b_2, b_{if}, b_6, b_7)$의 경로로 수행될 경우, 태스크내 전압 스케쥴링 알고리즘은 에너지 소모를 69% 줄일 수 있다.

위에서 모든 VSE 후보들이 최종적으로 VSE로 선택되지는 않는다는 사실에 주목하여야 한다. 가변 전압 프로세서에서 클록 속도 및 공급 전압을 변화시키는 데에는 어느 정도의 시간 및 전력이 추가로 소요되고 프로그램 코드 내에도 클록 속도 및 공급 전압을 계산하고 변화시키기 위한 명령어가 추가로 삽입되기 때문에, 클록 속도와 공급 전압을 바꾸었을 때 마감시간 제약 조건이 보장되고, 줄어든 전력 소모가 오버헤드보다 더 클 때만 주어진 VSE 후보가 최종적으로 VSE로 선택된다. 태스크내 전압 스케쥴링에서는 VSE 후보 (b_i, b_j)에서 프로그램의 제어 흐름이 b_i에서 b_j로 바뀔 때 잔여 작업량이 줄어든 정도, 즉 여분 사이클 수(saved cycle)가 주어진 임계치(threshold)보다 커야만 (b_i, b_j)를 최종적으로 VSE로 선택하는데, 이때의 임계치 값은 클록 속도 및 공급 전압을 변화시키는데 걸리는 시간과 전력의 오버헤드와 프로그램 코드 크기의 증가치에 대한 함수로 주어진다.

3. 프로파일을 이용한 태스크내 전압 스케쥴링

3.1 동기

프로파일을 이용한 태스크내 전압 스케쥴링 알고리즘을 설명하기 전에 먼저 2장에서 설명한 기존의 태스크내 전압 스케쥴링 알고리즘을 일반화해 보자. 태스크내 전압 스케쥴링에서는 프로그램의 남은 부분이 어떤 경로로 수행될 것이라고 예측하고 이를 바탕으로 VSE에서 클록 속도를 조절하는데, 이때 예측된 수행 경로를 참조 실행 경로(Reference Execution Path)라고 부르기로 한다. 앞으로 자세하게 설명되겠지만 어떻게 참조 실행 경로를 선택하느냐에 따라서 전력 소모의 감소율이 크게 차이 나게 된다.

일단 참조 실행 경로가 결정되면, 태스크내 스케쥴링은 태스크가 예측된 참조 실행 경로를 갈 것이라고 가정하고 초기 동작 전압과 클록 속도를 결정한다. 분기문 등에 의해서 실제 실행이 예측된 참조 실행 경로를 벗어나게 되면, 클록 속도는 참조 실행 경로와 새롭게 벗어난 실행 경로의 잔여 실행 사이클 수의 차이 값만큼 조절된다. 만약 새 실행 경로가 참조 실행 경로보다 훨씬 긴 경로라면 클록 속도는 마감시간을 지키기 위해서 높아져야 한다. 반면에, 만약 새 실행 경로가 참조 실행 경로보다 실행을 일찍 끝낼 수 있다면 클록 속도는 전력 소모를 줄이기 위해서 낮추어 질 수 있다. 일단 실행이 참조 실행 경로와 다른 경로를 취하게 되면, 벗어난 기본 블록부터 시작하는 새로운 참조 실행 경로가 만들어진다.

태스크내 전압 스케쥴링은 정적 프로그램 분석 기법을 이용하여, 클록 속도가 높여지거나 낮추어져야하는 적절한 프로그램 위치를 찾게 된다. 그리고 클록 속도를 실행 시간에(run-time) 조절하기 위해서 선택된 프로그램 위치에 전압 조절 코드를 삽입한다. 전압 조절 코드가 삽입될 후보 위치는 CFG에서 분기문이나 반복문에 해당하는 분기 에지에 해당한다.

기존의 태스크내 전압 스케쥴링의 경우에는 프로그램의 남은 부분이 어떤 경로로 수행되더라도 마감시간 제약 조건을 지켜야 하기 때문에 잔여 최악 실행 경로(RWEP)를 참조 실행 경로로 선택하게 된다. 본 논문에서는 기존의 태스크내 전압 스케쥴링을 RWEP 스케쥴링이라고 부르기로 한다. RWEP 스케쥴링에서는 클록 속도가 모든 VSE에서 단조적으로 감소한다. 그러나 제안하는 알고리즘에서는 RWEP 스케쥴링과는 다른 방법으로 참조 실행 경로를 선택하기 때문에, 어떻게 참조 실행 경로가 선택되는가에 따라 클록 속도는 어떤 VSE에서는 올라가고 어떤 VSE에서는 내려간다. 그러므로, VSE를 Up-VSE와 Down-VSE로 구분한다. Up-VSE에서는 클록 속도가 증가하며 Down-VSE에서는 클록 속도는 감소한다. 모든 후보 Up-VSE는 최종 VSE로 선택된다. 그렇지 않으면 요구되는 마감시간이 어겨지게 된다.

RWEP 스케쥴링은 마감시간을 지키면서 전력 소모를 줄이지만, 이 방법은 항상 가장 긴 경로가 실행되리라 예상하기 때문에 비관적인 접근법으로 볼 수 있다. 좀 더 낙관적인 접근법은 참조 실행 경로로 평균 실행 경로(ACEP)를 사용하는 것이다. 평균 실행 경로는 가장 실행될 가능성이 높은 경로로 정의되며, 실행에 대한 프로파일 정보를 통해 결정된다.

WCEP 대신에 ACEP를 사용하는 주된 동기는 프로그램 수행 시 자주 발생하는 경우에 대해서 최적화함으로서 전력 소모면에서 좀 더 높은 효율을 얻고자 함이다. 일반적인 프로그램에서는 프로그램 실행 시간의 약 80%는 프로그램 코드 크기에서 단지 20%에 불과한 부분에서 수행되며, 이러한 코드를 핫 패스(hot path)라고 부른다[10]. 태스크내 전압 스케쥴링의 전력 소모를 보다 효율적으로 하기 위해서는 우선적으로 핫 패스 부분에서의 전력 소모를 줄이는 것이 유리하다. 만약에 핫 패스 중에서 하나를 태스크내 전압 스케쥴링을 위한 참조 경로로 사용한다면, 속도 변경 그래프는 핫 패스에

대해 속도의 변경이 거의 없이 평평한 형태를 가진다. [11]에 의하면 속도의 변경이 없는 경우가 같은 작업량에 대해서 전력 소모를 최대로 줄인 전압 스케쥴이 된다. 핫 패스가 아닌 경로에 대해서도, 참조 실행 경로로 핫 패스를 사용하게 되면 WCEP를 참조 실행 경로로 사용할 때보다 더 낮은 클록 속도로 출발할 수 있기 때문에 전력 소모면에서 보다 효율적으로 된다.

핫 패스를 골라내는 방법으로는 여러 가지 방법이 있겠지만, 본 논문에서는 핫 패스의 일반적인 의미를 가장 잘 나타내는 ACEP를 사용한다. 본 논문에서 제안하는 프로파일을 이용한 태스크내 전압 스케쥴링에서는 프로그램의 남은 부분이 ACEP로 수행될 것이라고 예측하기 때문에, 참조 실행 경로로 잔여 평균 실행 경로(Remaining Average-case Execution Path: RAEP)를 사용한다. 따라서 프로파일을 이용한 태스크내 전압 스케쥴링을 본 논문에서는 RAEP 스케쥴링이라고 부르기로 한다.

본 논문에서는 프로그램의 ACEP를 찾기 위한 프로파일 기법으로 프로그램의 제어 흐름 그래프(CFG)상의 에지(edge)의 실행 빈도를 측정하는 방법을 사용했다. 이 방법은 개별 에지의 실행 빈도를 타겟 머신의 시뮬레이터를 이용하여 특정 입력 값들에 대해서 측정한 후, 이 값을 바탕으로 가장 자주 실행되는 경로(ACEP)를 추정한다. 이러한 기법은 핫 패스(hot path)를 찾기 위해서 가장 널리 사용되고 있는 방법이다.

하지만 좀 더 정확한 프로파일링을 위해서는 최근에 제시된 개선된 기법을 사용할 수도 있다. 예를 들어, 최근에 제안된 acyclic 경로에 대해서 직접 프로파일을 하는 기법에서는 오차에 대한 부분을 SPEC95의 train과 ref의 두 가지 입력 데이타를 사용한 실험 결과를 통해서 보여주고 있는 데, train 데이터를 이용해 관찰 된 경로의 41.8-96.0%의 경로가 ref 데이터에서도 발견되었다[12]. 그리고, 이러한 공통의 경로가 전체 실행 경로의 71.9-100.0%를 차지했다. 즉, train 데이터를 이용한 프로파일에서 ref 데이터를 사용할 경우 실행될 경로를 대부분 감지해 냈다는 것이다. 이것은 프로파일 기법의 오차가 크지 않다는 것을 보여주고 있다.

본 논문에서 사용한 프로파일링 기법은 에지 정보만을 이용함으로 경로를 이용하는 개선된 프로파일 기법에 비해 프로파일링 오차가 클 수도 있다. 하지만 본 논문에서는 구현의 간편성을 위해서 에지를 이용한 프로파일 기법을 사용하였으며 이 기법의 정확도는 논문의 핵심 기여도를 설명하는 데는 충분히 유용하다고 생각된다. Ball 과 Larus[12]의 기법을 적용할 경우, 프로파일 오차가 줄어들어 더욱 좋은 결과가 나오리라 예상된다.

프로파일 된 실행 경로의 실행 시간 측정을 위해서 본 논문에서는 Lim 등이 제시한 정적 분석 기법(static analysis)[9]을 사용하였다. RISC와 같이 타겟 아키텍쳐의 구조가 복잡하지 않을 때는 이러한 정적 분석 기법이 어느 정도 효과적이라고 알려져 있다. 물론, 복잡한 구조하에서는 Puschner [13]가 제시한 것과 같이 적절한 입력 데이터를 생성하여 직접 측정하는 방법을 사용할 수도 있다.

그림 4는 b_i로부터 출발하는 모든 경로중에서 잔여 평균 실행 사이클을 나타내는 $C_{RAEP}(b_i)$ 값을 가진 RAEP를 이용한 CFG G_P^{RAEP}를 보여주고 있다. G_P^{RAEP}에서 굵은 선은 두 개의 분기 에지중에서 실행시에 선택될 확률이 높은 것을 의미한다. 그림 4에서 최초의 참조 실행 경로는 ($b_1, b_{wh}, b_3, b_5, b_{wh}, b_3, b_5, b_{wh}, b_{if}, b_7$)가 된다. 이 참조 실행 경로를 통해, $C_{RAEP}(b_i)$가 계산된다. 예를 들어, $C_{RAEP}(b_{if}) = C_{EX}(b_{if}) + C_{RAEP}(b_7)$가 된다. RAEP 스케쥴링에서는 그림에서 ●로 표시된 Down- VSE뿐만 아니라 ○로 표시된 Up-VSE도 있다. 그림 5는 RAEP 스케쥴링에 의해서 어떻게 클록 속도와 공급 전압이 변화하는 지를 보여주고 있다. 클록 속도는 SUR 값 1.5 ($=\frac{15}{15-5}$)를 가진 Up-VSE(b_{if}, b_5)에서 14MHz에서 21MHz로 바뀐다. 그림 3(b)에 나타난 RWEP 스케쥴링의 전력 소모량과 비교하여 RAEP를 이용한 태스크내 스케쥴링은 55%의 전력을 더 줄일 수 있다.

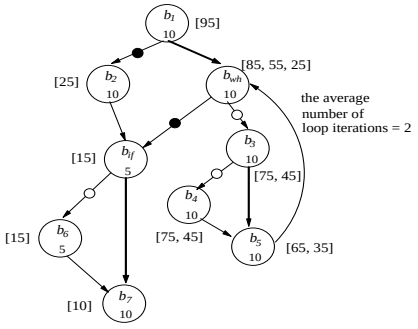

그림 4 RAEP를 이용한 CFG G_P^{RAEP}

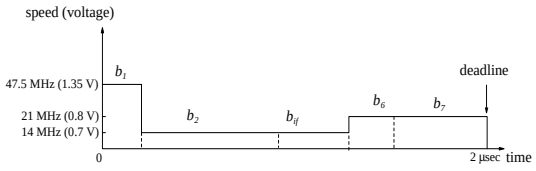

그림 5 RAEP 스케쥴링의 속도와 전압 변화

RAEP 스케쥴링이 RWEP 스케쥴링보다 전력 소모를 감소시키는 데에는 더욱 효과적이지만, 순수하게 RAEP만을 이용한 접근법은 경성 실시간 응용 프로그램의 마감시간 제약 조건을 지키지 못할 수 있다. 그 이유는 이 기법이 모든 실행 경로에 대해서 마감시간 제약 조건을 만족시키지 않기 때문이다. 예를 들어, WCEP와 ACEP가 실행 사이클 수에서 상당히 큰 차이를 보이는 경우를 생각해보자. 프로그램의 실행이 WCEP를 지나가고 있을 경우, 프로그램의 중간에 프로세서의 최대속도를 사용해도 잔여 실행 사이클을 마감시간까지 모두 완료하지 못할 수도 있다. 이러한 문제를 해결하기 위해서 다음 절에서는 RAEC를 사용하면서도 모든 실행 경로에 대해서 마감시간 제약 조건을 보장할 수 있는 새로운 방법을 설명한다.

3.2 참조 경로 수정

본 논문에서 제안하는 알고리즘은 원래의 RAEP 스케쥴링이 마감시간을 어길 수 있다는 문제를 해결하기 위해서 마감시간이 지켜지지 않을 수 있는 상황이 포착되면 참조 실행 경로를 수정한다. 참조 실행 경로가 $p_{ref}=(b_1,\cdots,b_i,b_{i+1},\cdots,b_N)$이며 b_i가 b_{i+1}와 b_{miss}를 자식 노드로 가지는 분기 노드이고 b_i에서의 현재 속도가 S라고 가정하자. 만약 $S\times SUR(b_i\rightarrow b_{miss})$로 주어지는 b_{miss}에서의 클록 속도가 프로세서의 최대 클록 속도인 $MaxS$보다 크다고 하면, 마감시간은 지켜지지 못할 수 있다. 그 이유는 마감시간까지의 남아있는 시간인 T_R은 $T_R=\frac{C_{RAEC}(b_{i+1})}{S}$이고 $MaxS\times T_R<C_{RAEC}(b_{miss})$가 되기 때문이다. 즉, b_{miss}에서부터 시작되는 새로운 참조 실행 경로를 따라 프로그램이 진행할 때 마감시간 이후에 $M=[C_{RAEC}(b_{miss})-MaxS\times T_R]$의 사이클이 추가로 실행되어야 한다. 마감시간이 지켜지지 않는 것을 피하기 위해서는 $k\le i$인 모든 k에 대해서 $C_{RAEC}(b_k)$를 M만큼 늘려주어야 한다. 즉, 참조 경로의 b_i와 b_{i+1}사이에 새로운 가상 기본 블록(virtual basic block)인 b_v를 추가하여 변경한다. $C_{EC}(b_v)$는 M으로 한다. 가상 기본 블록은 단지 속도 할당 시에 마감시간을 지키도록 하기 위해서 사용된 것으로 실행시간에 실제로 실행되지는 않는다.

그림 6은 어떻게 참조 경로의 수정이 이루어지는 지를 보여주고 있다. 원래의 CFG G_P^{RAEP}가 주어졌을 때, ACEP (b_1,b_3,b_4)가 참조 실행 경로로 이용된다. (굵은 에지는 실행시간에 선택될 가능성이 더 높은 에지를 의미한다.) 100MHz가 최대 클록 주파수일 때, 경로 (b_1,b_3,b_5)는 0.5μsec의 마감시간을 지키지 못하게 되는 데, 그 이유는 (b_3,b_5)에서 클록 속도가 120 MHz(60MHz×2)로 올라가야 하기 때문이다. 120 MHz를 사용하지 못하고 최대 클록 주파수인 100 MHz를 사용하게 되면 $\frac{10}{3}$ 사이클[3]이 마감시간으로부터 벗어나기 때문에 그림 5(b)에 보이듯이 b_3와 b_4사이에 가상 블록 b_v를 추가하게 되며, $C_{EC}(b_v)$는 4($\lceil\frac{10}{3}\rceil$)가 된다. 추가된 b_v로 인해 $C_{RAEC}(b_1)$와 $C_{RAEC}(b_3)$가 각각 34와 24로 변경되고 속도 변경 비율도 수정된다. 예를 들어, (b_3,b_5)에서의 SUR은 2에서 1.43($=\frac{20}{14}$)으로 변경된다.

그림 6(c)와 (d)는 각각 경로 (b_1,b_3,b_4)와 (b_1,b_3,b_5)에 대해서 RWEP 스케쥴링과 원래의 RAEP 스케쥴링, 그리고 수정된 참조 실행 경로를 이용하는 RAEP 스케쥴링에서의 클록 속도 변화를 비교한 것이다. 수정된 RAEP를 이용한 스케쥴링은 전체 전력 소모에 가장 큰 영향을 미치는 핫 패스에 대해서 RWEP 스케쥴링보다 전력 소모면에서 효율적이다(그림 6(c)). 이것은 또 원래의 RAEP 스케쥴링과 달리 마감시간 조건도 지킨다(그림 6(d)).

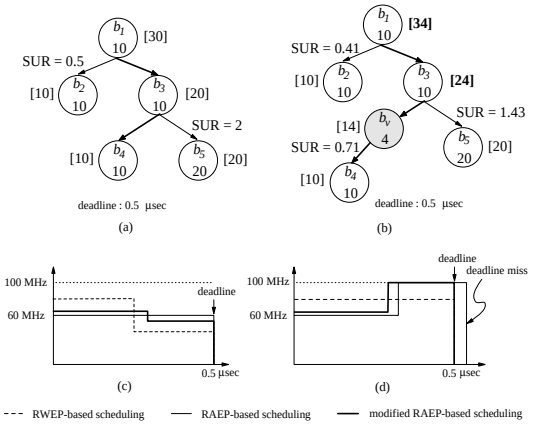

그림 6 수정된 RAEP를 이용한 태스크내 스케쥴링: (a) 원래의 G_P^{RAEP}, (b) 수정된 G_P^{RAEP}, (c)-(d) 경로 (b_1,b_3,b_4)와 (b_1,b_3,b_5)에 대한 세 가지 태스크내 스케쥴링 알고리즘의 속도 변화 그래프

3) 20 cycles 100 MHz $\times\frac{10\text{cycles}}{60\text{MHz}}$

```
Procedure Modify_RP( IN G, IN MaxS, IN D)
  /* Input: CFG G, Max Speed MaxS, and Deadline D */
   repeat
      unsafeVSE := NotFound;
      S := C_WCEC(b0)/D;    /* Calculate the start speed */
      foreach p ∈ G          /* for each path p in the CFG G */
         foreach e ∈ p          /* for each VSE e in the path p */
            /* Change speed with the speed update ratio of the edge e */
            S := S × SUR[e];
             if S > MaxS then
                unsafeVSE := Found;  /* unsafe VSE is found */
                 goto line 17;
             endif
         end foreach
      end foreach
  /* Calculate the number of cycles to be missed from the deadline */
       MissedCycles := Get_MissedCycle(e, S);
  /* Add a virtual block to the reference path of the path p */
      Add_VirtualBlock(p, e, MissedCycles);
  /* Update SURs of all the VSEs preceding e in the path p */
      Update_SUR(p, e);
  until unsafeVSE is NotFound;
end Procedure
```

그림 7 참조 실행 경로 수정 알고리즘

그림 7은 참조 경로 수정 과정을 의사 코드(pseudo code)로 나타낸 것이다. 주어진 CFG *G*에서 프로시저 *Modify_RP*는 *G*의 경로 *p*를 탐색하며 각 VSE *e*에서 속도 변화를 계산한다. 만약에 속도가 프로세서의 최대 가능 속도인 *MaxS*보다 올려져야 하는 문제가 있는 VSE를 찾게되면, 그 VSE 이후에 속도 변화가 없다고 가정하고 *MissedCycles*을 계산한다. 그리고 참조 실행 경로에 가상 블록이 추가되고 선행 VSE의 속도 변경 비율도 수정된다. 이러한 탐색은 문제가 있는 VSE가 발견되지 않을 때까지 반복된다.

3.3 온라인 속도 할당

2장에서 설명한 대로 원래의 태스크내 알고리즘에서는 VSE를 위한 속도 변경 비율이 컴파일 시간에 결정되고 실행시간에 바뀌지 않는다. 이러한 속도 할당 방법을 오프라인(off-line) 속도 할당 방법이라고 부른다. 오프라인 속도 할당 방법에서는 모든 VSE의 후보들이 최종의 VSE로 선택되지는 않기 때문에 선택되지 않은 후보 VSE 때문에 이용되지 않는 사이클이 존재한다. 오프라인 방법은 속도 변경 비율을 결정할 때 *남아있는* 사이클만 이용하므로 이들 이용되지 않은 사이클은 이후의 VSE에 의해서도 이용되지 않아서 유휴시간(idle time)을 발생시킨다.

이러한 유휴시간을 이용하기 위해서, 목표 시스템이 효율적인 실시간 카운터를 제공한다고 가정하고 온라인(on-line) 속도 할당 방법을 제시한다. 온라인 할당 방법에서는 새로운 클록 속도는 잔여 사이클 수를 잔여 시간으로 나눈 값으로 결정한다. 새로운 속도가 실행시의 지나간 시간에 대한 정보를 사용하기 때문에 온라인 할당 방법에서는 속도 할당 비율은 필요하지 않다.

전압 변화 오버헤드가 작고 선택되지 않는 VSE의 개수가 적을 때에는 오프라인 할당 방법이 온라인 할당 방법에 비교해 결과가 상대적으로 많이 뒤떨어지지 않는다. 하지만 전압 변화 오버헤드가 클 때는 선택되지 않은 VSE의 개수가 늘기 때문에 온라인 속도 할당 방법이 오프라인 속도 할당 방법보다 더 효율적이다.

4. 실험 결과

본 연구에서는 제안된 태스크내 전압 스케쥴링 알고리즘이 원래의 태스크내 스케쥴링 알고리즘에 비해 얼마나 나은지를 비교하기 위해 이미 개발된 자동 전압 조절기 도구(Autimatic Voltage Scaler:AVS)를 확장하였다. AVS는 DVS를 고려하지 않고 제작된 원래의 프로그램 P와 이 프로그램의 마감시간 제약 조건을 입력으로 받아들여 DVS를 사용하는 저전력 프로그램 P_{DVS}를 만든다. 변환된 프로그램 P_{DVS}는 가변 전압 프로세서에서 속도와 전압을 조정하기 위한 모든 작업을 하는 전압 조절 코드를 포함하고 있다. 확장된 AVS는 프로그램을 RWEP 스케쥴링이나 RAEP 스케쥴링을 사용하여 프로그램을 변환하며, 또 온라인과 오프라인의 속도 할당 알고리즘을 모두 지원한다.

본 논문에서는 제안된 태스크내 전압 스케쥴링 알고리즘의 전력 감소 효과를 측정하기 위해서 MPEG-4 디코더를 사용하였고 에너지 시뮬레이터(simulator)를 이용하여 전력을 측정하였다. 시스템이 유휴상태에 있을 때에는 DVS 시스템이나 DVS를 사용하지 않는 시스템이나 모두 전력 차단(power-down) 모드로 바뀌며 전력 차단 모드에서는 전력 소모가 0이라고 가정했다. 주어진 클록 주파수에 대한 공급전압은 $f_{CLK}=1/T_D \propto (V_{DD}-V_T)^{\alpha}/V_{DD}$ [2]로부터 구해지는데, 이때 V_{DD}, V_T, α는 각각 2.5V, 0.5V, 1.3으로 가정했다. RAEP 스케쥴링을 위해서 MPEG-4 비디오 디코더의 CFG에 들어갈 분기 에지의 선택될 확률과 반복문의 평균 실행 회수는 프로파일 정보를 통해 계산되었다. 프로파일링을 통해서 이러한 정보를 구해내지 못하는 에지에 대해서는 0.5의 확률을 할당했다. 클록과 전압의 변경에 소요되는 시간은 일반적인 DC-DC 변환기의 슬루율(slew rate)인 1.0V/200μsec의 값으로 가정했다.

그림 8은 여유 비율값(slack factor)을 변화시켜가며 정규화된 시작 속도가 어떻게 변하는 지를 보여주고 있다. 여유비율은 $\frac{\text{deadline}-WCET}{\text{deadline}}$로 정의되며, 프로세서가 WCET 이후에 유휴한(idle) 상태가 되는 시간의

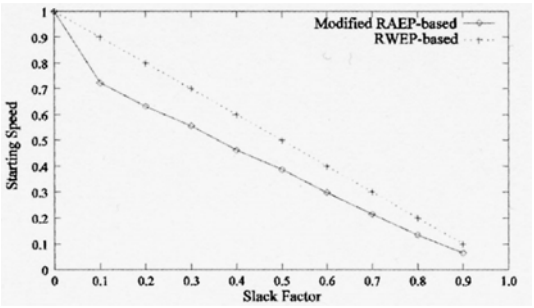

그림 8 RWEP 스케쥴링과 RAEP 스케쥴링의 여유 비율 변화에 따른 정규화된 시작속도 변화

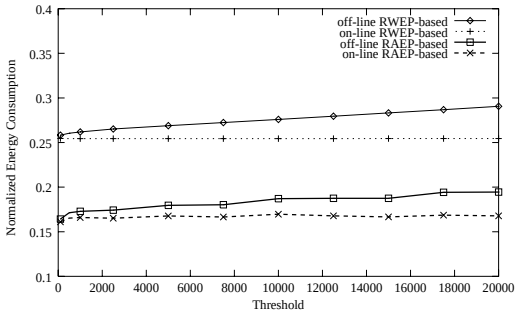

그림 10 온라인과 오프라인 속도 할당 방법의 임계값 변화에 따른 정규화된 전력 소모값

비율을 나타낸다. 3.2절에서 설명된 과정에 의해서 수정된 ACEP에 대한 실행 시간이 MPEG-4 디코더에 있어서 WCET보다 35%까지 작음을 알 수 있다. 이것은 프로세서가 처음 프로그램을 수행할 때 RWEP 스케쥴링이 요구하는 속도보다 35% 더 느린 속도로 시작할 수 있으며, 그 결과 전력 소모를 더 많이 줄일 수 있다는 것을 의미한다.

그림 9는 여유 비율을 변화시켜가며 두 가지 태스크내 스케쥴링 알고리즘의 전력 소모를 비교한 것이다. 모든 결과는 DVS를 사용하지 않는 시스템에서 실행되는 원래 프로그램의 전력 소모값에 대해서 정규화 되어있다. 실험을 위해서 VSE 선택을 위한 임계값은 1,000사이클을 사용했다. MPEG-4 디코더에 대해서, RAEP 스케쥴링은 RWEP 스케쥴링에 비해 34%까지의 전력 소모를 감소시킨다.

여기서 여유 비율이 0인 경우(즉, 마감시간과 WCET가 같은 경우)에도 RWEP 스케쥴링과 RAEP 스케쥴링의 전력 소모 사이에 많은 차이가 있다는 것을 알 수 있다. 이것은 시작 속도는 RWEP 스케쥴링에서와 같은

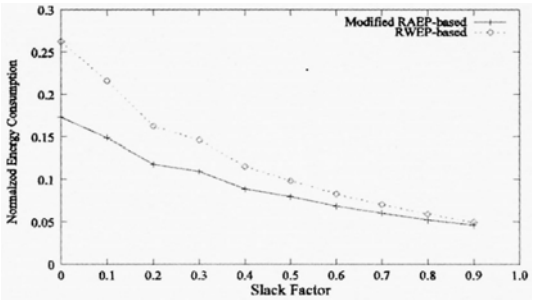

그림 9 RWEP 스케쥴링과 RAEP 스케쥴링의 여유 비율 변화에 따른 정규화된 전력 소모값

속도로 결정되지만 RAEP를 이용하여 속도가 결정된 부분을 이용할 수 있는 경로가 프로그램 중간에 많이 있기 때문이다. 즉, 시간 제약 조건을 만족시키기 위해서 가상 블록이 추가되어 시작 속도는 RWEP를 이용한 방법과 같아지지만 가상 블록 이후의 일부 경로는 ACEP를 이용한 속도 결정을 활용할 수 있게 된다. 여유 비율이 증가할수록 전력 소모의 차이는 감소하는 데, 이것은 두 가지 태스크내 전압 스케쥴링 알고리즘이 모두 낮은 전압을 사용하기 때문이다. 전력 소모는 V_{DD}^2에 비례하므로 낮은 전압을 사용할수록 전력 소모에 더 작은 차이를 발생시키는 것이다.

그림 10은 VSE 선택을 위한 임계값을 바꾸어 가며 어떻게 전력 소모가 변하는 지를 보여주고 있다. 여유 비율은 0으로 가정했다. 온라인 속도 할당법은 추가로 40 사이클의 오버헤드를 반영했다. 임계값이 작을 때는 온라인과 오프라인 속도 할당법 사이에 전력 소모에 있어 큰 차이가 없으나 임계값이 커질수록 선택되지 않는 VSE들이 많아져 전력 소모 차이가 10%까지 생기는 것을 볼 수 있다.

5. 결 론

본 논문에서는 RAEP 정보를 바탕으로 한 새로운 태스크내 전압 스케쥴링 알고리즘을 제시했다. 제안된 알고리즘은 프로그램이 실행될 때 평균 실행 경로, 즉 핫 패스가 최악 실행 경로(WCEP)보다 실행될 가능성이 많으며 이러한 핫 패스에 대해서 전력 소모를 최적화하는 것이 좋다는 사실에 착안하였다. 제안된 알고리즘의 주된 기여는 마감시간을 보장하면서도 각 실행 경로의 확률을 이용하여 원래의 태스크내 전압 스케쥴링 알고리즘을 개선시켰다는 점이다. MPEG-4 비디오 디코더를 이용한 실험 결과에서 RAEP 스케쥴링이 RWEP 스

케쥴링보다 전력 소모를 34%까지 감소시키는 것을 관찰할 수 있었다. 본 연구에서는 또 태스크내 전압 스케쥴링 알고리즘에서 선택되지 않은 VSE로 인해 발생하는 유휴시간을 완전히 이용할 수 있도록 온라인 속도 할당 방법을 제안하였으며, 전압 변화를 위한 오버헤드가 클 때에는 온라인 속도 할당 방법을 사용함으로써 전력 소모를 10%까지 감소시킬 수 있었다.

감사의 글

이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터신기술공동연구소에 감사 드립니다.

참 고 문 헌

[1] T. Burd and R. Broderson. Processor design for portable systems. Journal of VLSI Signal Processing, vol. 13, no. 2, pp. 203-222, 1996.

[2] T. Sakurai and A. Newton. Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas. IEEE Journal of Solid State Circuits, vol. 25, no. 2, pp. 584-594, 1990.

[3] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In Proc. of the 36th Annual Symposium on Foundations of Computer Science, pp. 374-382, 1995.

[4] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processor. In Proc. of the 19th IEEE Real-Time Systems Symposium, pp. 178-187, 1998.

[5] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In Proc. of the 36th Design Automation Conference, pp. 134-139, 1999.

[6] Y. Lee and C. M. Krishna. Voltage-clock scaling for low energy consumption in real-time embedded systems. In Proc. of the 6th International Conference on Real-Time Computing Systems and Applications, pp. 272-279, 1999.

[7] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In Proc. of the 37th Design Automation Conference, pp. 806-809, 2000.

[8] D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. IEEE Design and Test of Computers, vol. 18, no. 2, pp. 20-30, 2001.

[9] S.-S. Lim, Y. H. Bae, G. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Kim. An accurate worst case timing analysis for RISC processors. IEEE Transactions on Software Engineering, vol. 21, no. 7, pp. 593-604, 1995.

[10] T. Ball and J. R. Larus. Using paths to measure, explain, and enhance program behavior. IEEE Computer, vol. 33, no. 7, pp. 57-65, 2000.

[11] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In Proc. of International Symposium On Low Power Electronics and Design, pp. 197-202, 1998.

[12] T. Ball and J. R. Larus. Efficient path profiling. In Proc. of International Symposium on Microarchitecture, pp. 46-57, 1996.

[13] P. Puschner and R. Nossal. Testing the results of static worst-case execution-time analysis. In Proc. of the 20th IEEE Real-Time Systems Symposium, pp. 134-143, 1998.

신 동 군

1994년 서울대학교 계산통계학과 학사. 2000년 서울대학교 전산과학과 석사. 현재 서울대학교 전기 컴퓨터 공학부 박사과정. 관심분야는 내장형 시스템, 저전력 시스템, 실시간 시스템, 컴퓨터 구조

김 지 홍

1986년 서울대학교 계산통계학과 학사. 1988년 University of Washington 컴퓨터과학과 석사. 1995년 University of Washington 컴퓨터과학 및 공학과 박사. 1995년 ~ 1997년 미국 Texas Instruments사 선임연구원. 1997년 ~ 2001년 서울대학교 전기 컴퓨터공학부 조교수. 2002년 ~ 현재 서울대학교 전기 컴퓨터공학부 부교수.