내장 실시간 시스템에서의 성능 재조정

박정근○ 유민수 홍성수
E-mail: {jkpark, msryu, sshong}@redwood.snu.ac.kr
서울대학교 전기공학부

Performance Re-engineering of Embedded Real-Time Systems

Jungkeun Park Minsoo Ryu Seongsoo Hong
School of Electrical Engineering, Seoul National University, Seoul 151-742, Korea

요 약

본 논문에서는 내장 실시간 시스템(real-time embeddedd system)에서의 성능 재조정(re-engineering) 문제를 다룬다. 성능 재조정 문제는 하드웨어와 소프트웨어가 이미 구현된 상태에서 새로운 성능조건이 요구될 때 이를 만족시키도록 시스템을 수정하는 문제이다. 본 논문에서의 성능 요구조건은 시간당 처리량(throughput)이나 입출력 지연시간(input-to-output latency)등이 고려된다. 제안된 방법은 병목점 분석(bottleneck analysis)과 비선형 최적화를 이용한다. 이를 위해 프로세스 네트워크(process network)로 표현된 시스템 디자인과 태스크 그래프, 태스크 할당(allocation)과 스케줄링, 그리고 새로운 성능조건인 실시간 처리량을 입력으로 사용한다.

제안된 방법은 두 단계로 구성된다. 첫째, 프로세스 네트워크에서 프로세스의 지연시간을 계산하여 병목이 되는 프로세스를 찾아낸다. 둘째, 프로세싱 요소의 성능 개선율(performance scaling factor)을 변수로 하여 주어진 성능을 만족시키기 위한 시스템 제약조건을 유도한다. 이를 사용하여 하드웨어의 업그레이드 비용을 최소화 하도록 제약조건을 풀고 각 프로세싱 요소(processing element)에 필요한 성능 개선율을 구한다. 제안된 방법은 기존에 구성된 소프트웨어의 구조를 고치지 않기 때문에 재조정의 시간을 줄일 수 있게 한다.

1. 서론

시스템이 복잡해 지면서 내장 실시간 시스템의 디자인, 분석, 디버깅을 위한 소프트웨어에 대한 요구가 점차 높아지고 있다. 또한, 이를 위해 많은 설계 기법[3, 4]과 소프트웨어[2]가 개발되었다. 이들은 대부분 내장 실시간 시스템의 초기 개발단계에 초점을 두고 있다. 그러나, 실제 산업계에서는 많은 시간이 이미 개발된 시스템의 성능 재조정에 투자되고 있으며 많은 제품들이 기존 제품의 일부 성능을 개선하여 생산되고 있다.

이러한 재조정 과정에서는 내장 시스템 디자인의 세세한 부분을 개발자들이 직접 찾아서 수정해야 하는 어려움이 있다. 또한, 이 과정은 가격과 성능, 개발시간을 모두 고려해야 하기 때문에 더욱 어렵다. 새로운 성능조건을 만족시키는 가장 쉬운 방법은 시스템의 모든 요소들의 성능을 요구되는 전체 성능 만큼 향상시키는 것이다. 그러나 이러한 방법을 사용하게 되면 최종 시스템의 가격이 너무 높아지기 때문에 실제적으로는 사용할 수 없다. 따라서, 개발자들은 비용을 최소화 하기 위해서 성능 향상에 병목이 되는 부분을 찾아서 해당 부분의 성능만을 향상 시키는 방법을 사용할 수 밖에 없게 된다.

보통 내장 실시간 시스템은 여러 마이크로 프로세서와 ASIC 칩, 각종 전기기계 장치들로 구성되기 때문에 재조정 단계에서는 이종 분산 다중 프로세서 하드웨어 플랫폼(heterogeneous distributed multiprocessor hardware platform)을 다루게 된다. 이와 함께 성능 재조정은 다음과 같은 특징을 지닌다. (1) 대상 시스템의 소프트웨어나 펌웨어들은 이미 개발되어 다양한 테스트를 마친 상태이다. (2) 대상 시스템의 태스크의 할당이나 스케줄링이 이미 결정된 상태이다. 본 논문에서는 이러한 특징을 고려하여 성능 재조정을 위한 체계적인 방법론을 제안한다. 그리고, 이를 이용하여 효과적으로 시스템의 병목을 찾아서 제거할 수 있도록 한다.

제안된 방법은 두 단계로 구성된다. 첫째 단계는 프로세스 네트워크로 주어지는 내장 시스템의 전체 디자인과 태스크 그래프, 태스크 할당 및 스케줄링, 그리고 새로운 성능조건인 시간당 처리량을 입력으로 한다. 이 단계에서는 프로세스의 지연시간을 계산하여 병목이 되는 프로세스를 찾아낸다. 둘째 단계에서는 병목이 되는 프로세스의 태스크 그래프를 분석하여 교체되어야 할 프로세싱 요소를 결정한다. 이 과정에서는 태스크 그래프와 스케줄링에 의해 결정되는 선행 제약조건(precedence constraints)을 이용하여 선형 제약조건(linear constraints)을 유도한다. 그리고, 선형 제약조건으로부터 하드웨어 교체비용을 최소화 할 수 있는 최적화된 해를 구한다. 제안된 방법의 전체 과정은 그림 1에 나타나 있다. 그림에서 볼 수 있듯이 최종 결과는 교체되어야 할 프로세싱 요소들의 필요한 성능 개선율이다.

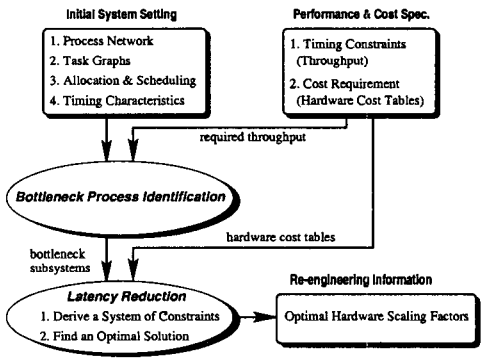

그림 1: 제안된 방법의 전체 과정

1.1 관련연구

본 논문에서 제시된 문제와 관련하여 여러 연구가 수행되었다. [2]에서는 AFTER라 불리는 CASE도구가 제안 되었다. 이 도구는 구현된 시스템의 실시간 스케줄링을 분석할 수 있도록 한다. AFTER는 내장 시스템의 타이밍 데이터를 기록하고 분석한 후 시스템에서의 명세(specification)와 비교하여 가능한 문제점

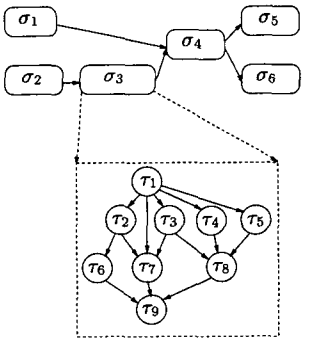

그림 2: 프로세스 네트워크와 태스크 그래프

을 제시한다. 개발자는 이를 이용하여 시스템의 타이밍을 수정하고 그 효과를 관찰할 수 있다. 이는 개발된 시스템을 수정한다는데 있어서 본 논문의 목적과 일치한다. 그러나 AFTER는 개발자가 수정한 시스템의 성능을 모니터링하는데 중점을 두고 있다. 반면, 본 논문에서는 병목 부분을 찾아내고 비선형 최적화 기법을 사용하여 이를 제거하는데 중점을 두고 있다. 또한, 본 논문에서 제안된 방법은 이미 구성된 태스크 코드를 수정하지 않도록 하고 있다.

[1]에서는 분산 플랫폼에서 수행되는 태스크 그래프의 입출력 지연시간을 줄일 수 있는 알고리즘을 제시하였다. 이 알고리즘에서는 시스템에서의 가장 긴 지연시간을 가지는 경로(path)를 줄이기 위해 해당하는 태스크를 복제(duplicate)하고 이를 다른 프로세서에 할당한다. 그러나 이 알고리즘은 하나의 경로만을 고려하는 문제점이 있다. 반면, 본 논문에서 제시된 방법은 시스템의 다른 경로도 모두 고려하여 하나의 경로의 길이를 지나치게 줄여서 다른 경로가 더 길게 되는 문제점을 해결한다.

2. 시스템 모델 및 문제 정의

2.1 시스템 모델

본 논문에서는 계층적인 시스템 모델을 사용한다. 내장 실시간 시스템은 프로세스 네트워크(PN)과 태스크 그래프(TG)로 표현된다. 프로세스 네트워크에서는 동시에 수행되는 여러 프로세스들이 단방향의 FIFO 채널을 통하여 통신한다. 프로세스 네트워크는 그래프 $G(\mathcal{P}, E)$로 나타내며 각각은 다음과 같다.

- $\mathcal{P}$는 $\mathcal{P} = \{\sigma_1, \ldots, \sigma_m\}$로 표현되는 프로세스의 집합이다. 하나의 프로세스는 여러개의 입력 스트림과 출력 스트림을 매핑시켜 준다.
- $E \subseteq \mathcal{P} \times \mathcal{P}$는 방향성을 가지는 간선(edge)의 집합이다. $\sigma_i \rightarrow \sigma_j$에서 σ_i는 σ_j에 선행한다. 각 프로세스는 모든 선행 프로세스가 수행을 종료한 후 시작되며 프로세싱하는 동안 시간을 소모하여 입출력 지연시간을 발생시킨다.

하나의 프로세스는 태스크 그래프로 표현된다. 태스크 그래프 $G(V, E')$는 다음과 같다.

- V는 $V = \{\tau_1, \ldots, \tau_n\}$로 표현되는 태스크 집합이다.
- $E' \subseteq V \times V$는 방향성을 가지는 간선의 집합이다. $\tau_i \rightarrow \tau_j$에서 τ_i는 τ_j에 선행한다. 프로세스의 E와 같은 성질을 지닌다.

그림 2는 프로세스 네트워크와 프로세스 σ_3의 태스크 그래프를 보여준다.

내장 시스템은 프로세싱 요소 $PE = \{P_1, \ldots, P_k\}$의 집합으로 표현된다. $\mathcal{S}_j$는 P_j의 성능 개선율을 나타낸다. $\mathcal{S}_j = 1$일 때 현재 성능을 나타내며 $\mathcal{S}_j < 1$일 때 더 빠른 성능의 PE를 나타낸다. $\mathcal{S}_j$는 $\mathcal{S}_j^0, \mathcal{S}_j^1, \ldots, \mathcal{S}_j^{n_j}$의 이산적인 값만을 가질 수 있다. 각 성능 향상율 $\mathcal{S}_j$에 따라 성능 향상 비용 $c_j(\mathcal{S}_j)$가 정의되며 전체 하드웨어 업그레이드 비용은 $\sum_{P_j \in PE} c_j(\mathcal{S}_j)$로 주어진다.

태스크 할당 A는 각 태스크로 부터 PE로의 매핑이며 태스크 τ_i는 할당된 PE $A(\tau_i)$에서 최악 수행시간(WCET) e_i를 가진다. 태스크가 최악 수행시간을 가지는 것과 마찬가지로 프로세스는 최대 입출력 지연시간을 가진다. 또한, 모든 프로세스는 동일한 주기를 가지고 수행된다. 이 주기는 시스템의 시간당 처리량을 결정한다. 이에 반해 태스크는 서로 다른 주기를 가질 수 있다.

2.2 문제 정의

본 논문에서는 병목이 되는 프로세스를 찾아내어 이를 효율적으로 제거하는 방법을 제안한다. 제안된 방법은 기존에 이미 설계된 소프트웨어의 구조를 변경시키지 않도록 하기 위해서 병목이 되는 프로세싱 요소만을 교체하는 방법을 사용한다.

제안된 방법은 다음과 같은 입력을 가진다.

(1) 시스템의 PN과 TG

(2) 태스크 할당과 스케줄링

(3) 새로이 요구되는 시간당 처리량(throughput) 또는 주기

(4) 각 프로세싱 요소들의 성능 개선에 따른 비용표

제안된 방법은 다음과 같은 두 단계로 구성된다.

단계 1 프로세스의 지연시간과 새로 주어진 주기를 비교하여 PN에서 병목이 되는 프로세스를 찾는다.

단계 2 단계 1에서의 병목 프로세스에 대해 교체되어야 할 프로세싱 요소들을 결정하기 위하여 3장에서 설명할 지연시간 감소 알고리즘(latency reduction algorithm)을 수행한다.

두번째 단계에서는 태스크 그래프 및 스케줄링에 의해 결정되는 선행 제약조건을 이용하여 선형 제약조건을 유도하고 이를 푼다. 목적함수는 $\sum_{P_j \in PE} c_j$로 주어지는 총 하드웨어 비용을 최소화하는 것이다.

3. 성능 재조정 과정

3.1 단계 1: 병목 프로세스 검색 단계

내장 시스템이 주어졌을 때 새로이 요구되는 시간당 처리량에 의해 결정되는 주기조건을 $\mathcal{L}$이라 하면, 프로세스 σ_j의 입출력 지연시간 $\mathcal{L}_j$가 $\mathcal{L}$보다 작을 때 σ_j는 병목 프로세스가 된다. 이를 위해서는, 프로세스의 최대 입출력 지연시간을 정확하게 구해야 한다. 그러나, [4]에서와 같이 일반적인 모델에서는 이를 구하기가 매우 어렵다. 따라서, 본 논문에서는 다음의 가정을 한다.

(1) 태스크 할당과 스케줄링이 모두 디자인 단계에서 정적으로 수행되고 이후 변하지 않는다.

(2) 태스크 스케줄링은 비선점형(non-preemptive)이다.

(3) 하나의 프로세스내의 모든 태스크의 주기는 같다. 만약 주기가 다른 태스크가 있다면 LCM(least commmon multiple)만큼 펼쳐서 주기가 같은 태스크들로 재구성할 수 있다.

위와같은 가정은 본 논문에서의 시스템 모델을 크게 제약시키지는 않는다. 왜냐하면 내장 시스템의 구조나 수행은 본 과정에서 이미 고정되어 있기 때문이다. (1)과 (2)는 태스크의 수행순서가 한 프로세싱 요소에서 고정되어 있도록 한다. 따라서, 프로세스의 최대 입출력 지연시간은 선행관계를 고려하여 태스크들의 최악 수행시간의 합으로 구할 수 있다.

마지막으로 다음의 가정이 필요하다.

(4) 프로세스들은 서로 독립적이다. 프로세스들은 프로세싱 요소들을 공유하지 않는다.

둘 이상의 프로세스가 프로세싱 요소를 공유하게 되면 프로세스 간의 태스크들이 서로 섞여서 수행되므로 고정된 스케줄을 얻기가 힘들다.

3.2 단계 2: 지연시간 감소 단계

3.2.1 단계 2.1: 제약조건 유도

우선 태스크의 할당과 스케줄링에 의하여 결정되는 선행 제약조건을 이용하여 선형 제약조건들을 유도한다. 이를 위해 주어

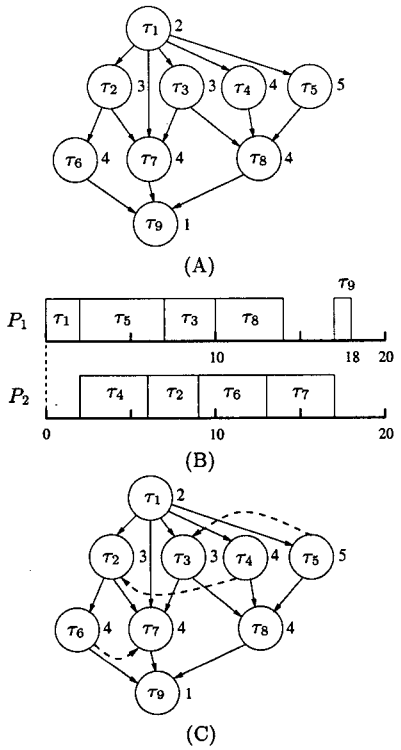

그림 3: (A) 태스크 그래프 (B) 태스크 스케줄 (C) 변형된 태스크 그래프

진 태스크 그래프를 변형한다. 만약 태스크 τ_i와 τ_j에 대해 τ_i가 τ_j보다 먼저 스케줄되면서 둘 사이에 선행관계가 없을 때 부가적인 선행관계 $\tau_i \rightarrow \tau_j$를 주어진 그래프에 추가 시킨다. 그림 3은 주어진 태스크 그래프와 그 스케줄 그리고 변형된 태스크 그래프의 예를 보여준다. 그림 3 (A)에서 태스크의 최악 수행시간은 각 노드의 옆에 표시되어 있다. 그림 3 (B)는 그림 3 (A)의 태스크 그래프가 두 프로세싱 요소 P_1과 P_2에 스케줄된 결과이다. 그림 3 (A)에서 태스크 τ_4와 τ_2 사이에는 선행관계가 없지만 P_2 상에서 τ_4가 τ_2보다 먼저 스케줄되었으므로 그림 3 (C)의 변형된 태스크 그래프에서는 $\tau_4 \rightarrow \tau_2$를 점선으로 표시하였다.

$\mathcal{L}$을 새로 제시된 시간당 처리량이라 하고, $s(\tau_i)$와 $f(\tau_i)$를 각각 τ_i의 시작 시간과 종료시간이라 하고, $\mathcal{S}(\tau_i)$를 τ_i가 할당된 PE의 성능 개선율이라 할 때, 변형된 태스크 그래프 상의 마지막 태스크 τ_i에 대하여 다음과 같은 제약조건을 구할 수 있다.

$$f(\tau_i) \leq \mathcal{L} \iff \max_{\tau_j \in Pred(\tau_i)} \{f(\tau_j)\} + e_i \mathcal{S}(\tau_i) \leq \mathcal{L} \qquad (1)$$

식 1에서 $Pred(\tau_i)$는 변형된 태스크 그래프상에서 τ_i에 바로 선행하는 태스크들의 집합이다. 식 1은 τ_i가 모든 선행 태스크의 종료 후에 수행될 수 있으며 τ_i의 종료시간은 τ_i가 할당된 프로세싱 요소의 성능 개선율에 의해 결정된다는 것을 의미한다. 만약 $\mathcal{S}(\tau_i) < 1$ 이면 τ_i는 현재 하드웨어 플랫폼에서 보다 더 일찍 수행을 마칠 수 있다.

식 1에서 $\tau_j \in Pred(\tau_i)$인 τ_j에 대하여 $f(\tau_j)$를 $s(\tau_j)+e_j\mathcal{S}(\tau_j)$로 그래프의 맨 아래서부터 반복적으로 치환하게 되면 식 1에서 시작시간과 종료시간을 없앨 수 있다. e_i는 상수이므로 위의 과정을 거치면 $\mathcal{S}(\tau)$만을 변수로 가지는 선형 제약조건을 구할 수 있다.

3.2.2 단계 2.2: 제약조건 풀기

선형 제약 조건을 구한 후에 총 하드웨어 비용 $C(\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k) = \sum_{P_j \in PE} c_j(\mathcal{S}_j)$을 최소화하는 각 프로세싱 요소의 성능 개선율의 벡터 $\langle \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k \rangle$를 구한다. 그러나, 목적함수의 $c_j(\mathcal{S}_j)$가 이산적인 값을 가지기 때문에 이는 선형 프로그래밍의 확장된 형태가 된다. 위의 문제는 심플렉스법과 같은 방법을 사용하여 근사해를 빨리 구할 수도 있지만 성능 재조정은 오프라인(off-line) 과정이므로 정확한 해를 구하는 것이 중요하다.

가장 단순한 방법은 모든 해 공간을 계산하여 최적의 해를 찾는 것이지만 이는 찾아야 할 해 공간이 너무 크게 된다. 본 논문에서는 탐색 공간의 분할을 이용한 알고리즘을 제시한다.

알고리즘 1 성능 개선율의 최적벡터 찾기

```
let C^min = ∞
for i = 0 to n_1 + n_2 + ... + n_k
    let C_i^min = ∞
    for each candidate γ = ⟨S_1^{i_1}, S_2^{i_2}, ..., S_k^{i_k}⟩
    where i_1 + i_2 + ... + i_k = i
        if C(γ) < C_i^min
            γ_i^min = γ
            C_i^min = C(γ_i^min)
        if C(γ) < C^min and γ satisfies the derived constraints
            γ^min = γ
            C^min = C(γ^min)
    if C_i^min ≥ C^min
        break
if C^min ≠ ∞
    γ^min is an optimal solution vector
    and C^min is its hardware cost
else
    no feasible solution
```

알고리즘 1은 k개의 PE가 있을 때, 성능 개선율 벡터를 $\gamma = \langle \mathcal{S}_1^{i_1}, \mathcal{S}_2^{i_2}, \ldots, \mathcal{S}_k^{i_k} \rangle$로 나타내고 이들을 $\Gamma_i = \{\gamma | i_1 + i_2 + \ldots + i_k = i\}$를 만족하는 집합들로 나눈다. 그리고 각 집합 Γ_i에서의 최소 비용 C_i^{min}을 구하고 $C_i^{min} \leq C_{i+1}^{min}$인 성질을 이용하여 효율적으로 해를 구한다.

4. 결론

본 논문은 내장 실시간 시스템에서의 성능 재조정 문제를 정의하고 병목 프로세스 분석과 비선형 최적화에 기반한 방법을 제안하였다. 제안된 방법은 프로세스 네트워크와 태스크 그래프로 주어지는 시스템 디자인과 태스크 할당 및 스케줄링, 그리고 새로 주어지는 실시간 처리량을 입력으로 하여 각 프로세싱 요소들의 성능 개선율을 출력으로 낸다. 제안된 방법은 기존에 구성된 소프트웨어의 구조를 고치지 않기 때문에 시스템을 처음부터 다시 디자인하지 않고도 새로운 요구조건을 만족시키도록 할 수 있다.

참고 문헌

[1] I. Ahmad and Y.-K. Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):872–892, September 1998.

[2] G. Arora and D. Stewart. A tool to assist in fine-tuning and debugging embedded real-time systems. In *ACM Workshop on Languages, Compilers and Tools for Embedded Systems*, pages 73–92, June 1998.

[3] M. Ryu, S. Hong, and M. Saksena. Streamlining real-time controller design: From performance specifications to end-to-end timing constraints. In *Proceedings of Real-Time Applications and Technology Symposium*, pages 192–203, June 1997.

[4] T.-Y. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(11):1125–1136, November 1998.