내장형 실시간 시스템의 성능 개선을 위한 리엔지니어링 기법

(Performance Reengineering of Embedded Real-Time Systems)

홍 성 수 †

(Seongsoo Hong)

요 약 본 논문에서는 내장형 실시간 시스템의 성능 개선을 위한 리엔지니어링(performance re-engineering) 기법을 제시한다. 시스템 리엔지니어링은 구현이 완료된 시스템에서 새로운 성능 요구사항을 만족시키기 위한 일련의 작업이라 할 수 있다. 일반적으로 실시간 시스템의 성능은 실시간 처리량(real-time throughput)과 입출력 시간 지연(input-to-output latency) 등으로 기술할 수 있으며, 새로운 성능 요구사항은 이와 같은 파라미터를 통해 기술된다.

본 연구의 리엔지니어링 기법은 두 단계로 구성된다. 첫째, 시스템을 프로세스 네트워크의 형태로 파악한 후, 프로세스의 수행시간을 분석하여 병목(bottleneck)이 되는 프로세스를 찾아낸다. 둘째, 병목 프로세스의 수행시간을 개선할 수 있도록 프로세싱 요소의 성능비례계수(performance scaling factor)를 구한다. 성능비례계수는 성능 개선을 비율로 나타낸 것으로서 리엔지니어링 비용을 최소화하도록 그 값을 구한다. 따라서 유도된 성능비례계수에 따라 하드웨어 장치를 업그레이드하면 하드웨어 비용을 최적화할 수 있다. 이러한 방법을 사용하면 소프트웨어를 수정할 필요가 없으며, 리엔지니어링 비용 및 시간을 단축할 수 있다.

키워드 : 실시간 시스템, 시스템 리엔지니어링, 실시간 성능, 병목 프로세스 분석, 비선형 최적화 기법

Abstract This paper formulates a problem of embedded real-time system re-engineering, and presents its solution approach. Embedded system re-engineering is defined as a development task of meeting performance requirements newly imposed on a system after its hardware and software have been fully implemented. The performance requirements may include a real-time throughput and an input-to-output latency. The proposed solution approach is based on a bottleneck analysis and nonlinear optimization. The inputs to the approach include a system design specified with a process network and a set of task graphs, task allocation and scheduling, and a new real-time throughput requirement specified as a system's period constraint.

The solution approach works in two steps. In the first step, it determines bottleneck processes in the process network via estimation of process latencies. In the second step, it derives a system of constraints with performance scaling factors of processing elements being variables. It then solves the constraints for the performance scaling factors with an objective of minimizing the total hardware cost of the resultant system. These scaling factors suggest the minimal cost hardware upgrade to meet the new performance requirement. Since this approach does not modify carefully designed software structures, it helps reduce the re-engineering cycle.

Key words : real-time system, system re-engineering, real-time performance, bottleneck process analysis, nonlinear optimization

1. 서 론

최근 컴퓨터 시스템이 고도로 복잡해지면서 내장형 실시간 시스템의 설계 및 디버깅 등에 대한 연구가 활발히 진행되어 왔다. 프로세서 및 주변기기들의 성능과 종류가 급격히 늘어남에 따라 유용한 설계 기법과 이를

† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
sshong@redwood.snu.ac.kr
논문접수 : 2001년 2월 19일
심사완료 : 2003년 3월 7일

활용하는 설계 도구의 필요성이 증대되었기 때문이다. 이에 따라 다양한 설계 기법[1, 2, 3]과 소프트웨어[4, 5]가 개발되었으며, 실시간 운영체제에 기반한 상용 개발 도구들이 널리 사용되고 있다[6, 7]. 하지만 이러한 결과물들은 시스템 개발에만 초점을 두고 있으며, 개발이 완료된 이후의 리엔지니어링은 간과하고 있다. 상당수의 시스템이 리엔지니어링을 통해 성능이 개선된다는 점에서 체계적인 리엔지니어링 기법이 요구되는 현실이다. 한 예로, 분당 25장을 복사할 수 있는 25 cpm(copies per minute) 디지털 복사기는 리엔지니어링을 통해 30 cpm 제품으로 출시된다. 이는 새롭게 개발된 제품이 아니라, 단지 리엔지니어링을 통해 업그레이드된 시스템이라는 점을 인식해야 한다.

리엔지니어링을 위한 방법론이나 도구(tool)의 부재로 인하여, 현재까지의 리엔지니어링은 개발자의 경험과 반복적인 튜닝(tuning)에만 의존하고 있다. 리엔지니어링의 목표는 비용을 최소화하는 한편 최상의 성능을 확보하는 것이다. 하지만 개발자의 경험적인 방법만으로는 이러한 목표를 달성하기 어려우며, 통상적으로 시스템의 성능을 일률적으로 개선하는 방법이 널리 사용되어 왔다. 이는 불필요하게 리엔지니어링의 비용을 증가시키는 문제를 안고있어, 성능 개선의 병목(bottleneck) 부분만을 해결하는 효율적 기법이 필요한 실정이다.

본 연구에서 제시하는 리엔지니어링 기법에서는 소프트웨어는 수정하지 않고 하드웨어만을 개선하여 성능을 향상시키고자 한다. 일반적으로 내장형 실시간 시스템은 다양하고 이질적인 하드웨어 장치를 포함하며[3], 이들은 각종 소프트웨어에 의해 구동된다. 시스템의 하드웨어는 마이크로프로세서와 ASIC 칩, 각종 전기 및 기계적 장치 등을 포함하며, 소프트웨어로는 펌웨어 및 응용 소프트웨어 등을 포함한다. 순수하게 하드웨어 대체만으로 리엔지니어링을 수행하려면 다음과 같은 전제가 요구된다: (1) 시스템의 소프트웨어나 펌웨어들은 검증된 상태이어야 하며, (2) 태스크의 할당 및 스케줄링이 이미 완료되어야 한다.

이러한 가정 하에서 본 연구에서 제시하는 리엔지니어링 과정은 2단계로 구성된다. 첫째 단계에서는, 프로세스 네트워크 및 태스크 그래프를 이용하여 시스템을 기술한 후, 병목 프로세스를 파악한다. 둘째 단계에서는, 병목이 되는 프로세스를 분석하여 성능 개선의 대상이 될 프로세싱 요소(processing element, CPU 또는 모터와 같은 전기/기계 장치)와 성능비례계수(performance scaling factor)를 결정한다. 그림 1은 제안된 방법의 전체적인 과정을 보여주고 있다. 그림 1에서 볼 수 있듯

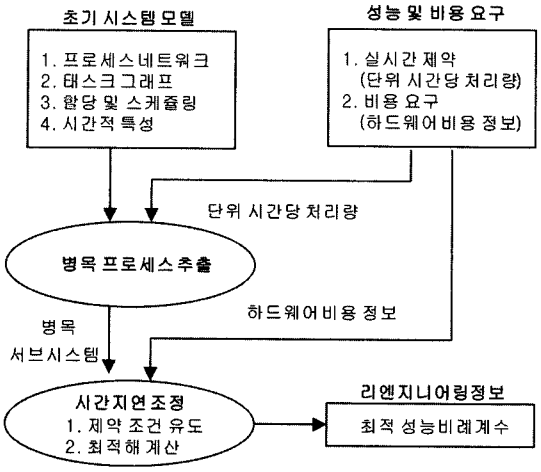

그림 1 시스템 리엔지니어링 과정

이, 리엔지니어링 문제는 초기 시스템 모델과 성능/비용 요구사항을 통해 정의된다. 이로부터 병목 프로세스를 찾은 다음, 성능 개선을 비율로 나타내는 성능비례계수 값을 구한다. 마지막으로 얻어진 값에 따라 프로세싱 요소를 교체하여 리엔지니어링을 완료한다.

2. 관련연구

본 연구와 유사한 것으로 [4]의 AFTER라 일컬어지는 CASE(computer aided software engineering) 도구가 있다. 이는 구현된 시스템의 타이밍 데이터와 성능요구사항(performance specification)을 비교하여 문제점을 발견한다. 개발자는 이를 이용하여 시스템의 타이밍 계수를 수정하고 그 효과를 관찰할 수 있다. 이는 개발된 시스템을 수정한다는 점에서 본 논문의 리엔지니어링과 비교될 수 있다. 그러나 AFTER는 시스템 성능을 모니터링하는데 중점을 두는 반면, 본 논문에서는 병목 부분을 찾아낼 뿐만 아니라 이를 최적화하는 것에 중점을 두고 있다. 또 다른 차이점으로서, 본 논문에서 제안된 방법은 이미 구현된 소프트웨어를 수정하지 않는 것을 전제하고 있다.

한편, [8]에서는 태스크 그래프 상에서의 입출력 시간지연(input-to-output latency)을 줄이는 알고리즘을 제시하였다. 이 알고리즘에서는 시스템에서 가장 긴 시간지연을 가지는 임계 경로(critical path)를 대상으로 복제된 태스크를 다른 프로세서에 할당한다. 이는 프로세스의 시간지연을 단축시키는데 효과적인 것으로 알려져 있다. 하지만 [8]의 방법은 임계 경로만을 고려하여 최

적해를 찾지 못할 뿐만 아니라 시스템 소프트웨어를 수정해야한다는 단점을 가진다. 본 논문에서는 시간지연을 줄이기 위해 최적해를 찾으며, 이를 통해 불필요한 비용의 요인을 제거한다.

3. 시스템 모델 및 문제 도출

3.1 시스템 모델

본 논문에서는 계층적인 모델[9]에 기반하여 실시간 시스템을 프로세스 네트워크(process network)로 표현하며, 임의의 프로세스를 다시 태스크 그래프(task graph)로 표현한다. 즉, 주어진 시스템 설계로부터 물리적인 구성요소를 프로세싱 요소로 간주하며, 각 프로세싱 요소가 수행하는 행위를 프로세스로 파악한다. 각 프로세스는 다시 세분화된 태스크들의 그래프로 표현할 수 있다. 이러한 표현 기법은 칸 네트워크[10]의 일종으로서 다중 프로세스가 동시에 수행되며 상호간에 통신하는 모델을 잘 표현하는 장점이 있다. 프로세스들은 단방향의 FIFO(first in first out) 채널을 통해 통신하며, 쓰기 행위는 비중단(non-blocking) 모델을 따르고 읽기 행위는 중단(blocking) 모델을 따른다[9]. 프로세스 네트워크는 그래프 모델로 나타낼 수 있는데, 본 논문에서는 다음과 같이 정의한다.

(1) $P=\{\sigma_1,\ldots,\sigma_m\}$는 프로세스의 집합이다. 하나의 프로세스는 여러개의 입력 데이터를 받아 출력 데이터를 생성한다.

(2) $E \subseteq P \times P$ 는 프로세스간 통신 채널의 집합이다. 프로세스 σ_i에서 프로세스 σ_j로의 통신 채널은 $\sigma_i \rightarrow \sigma_j$으로 표현되며, σ_j는 σ_i가 보내준 데이터를 입력으로 하여 변환된 데이터를 출력한다. 이 때, σ_j는 수행시간이 필요하며 이는 입출력 시간 지연의 요인이 된다.

그림 2는 프로세스 네트워크의 예를 보여주며, 임의의 프로세스는 다시 태스크 그래프로 구체적으로 표현할 수 있다. 태스크 그래프는 G(V, E')로 표시하며 다음과 같이 정의한다.

(1) $V=\{\tau_1,\ldots,\tau_n\}$는 태스크 집합이다.

(2) $E' \subseteq V \times V$는 방향성을 가지는 간선의 집합이다. $\tau_i \rightarrow \tau_j$에서 τ_i는 τ_j에 선행한다. 프로세스간의 통신과 유사하게 데이터 흐름의 방향을 나타낸다.

실시간 시스템의 물리적인 구성은 프로세싱 요소 $PE=\{P_1,\ldots,P_k\}$의 집합으로 표현할 수 있다. S_j는 프로세싱 요소 P_j의 성능비례계수를 나타낸다고 하자. $S_j=1.0$은 현재 성능을 나타내며, $S_j<1.0$는 더 빠른 성능(처리속도)을 나타낸다. S_j는 S_j^0, S_j^1, ..., $S_j^{n_j}$의 이산적인 값만을 가진다고 가정하며, 성능비례계수 S_j에 따른 비용함수 $c_j(S_j)$를 정의한다. 이에 따라 성능 개선에 요구되는 전체 비용은 $\sum_{P_j \in PE} c_j(S_j)$로 나타낼 수 있다.

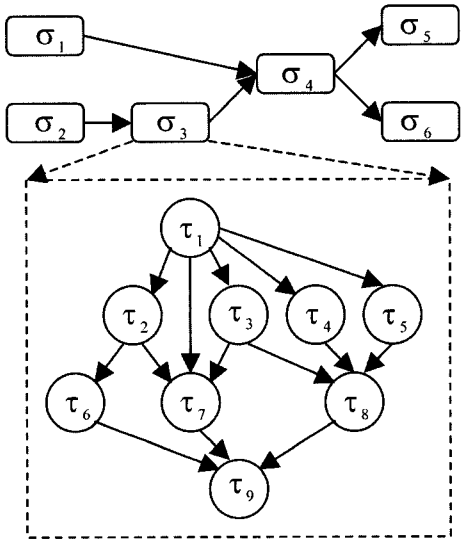

그림 2 프로세스 네트워크와 태스크 그래프

태스크 할당은 $A : V \mapsto PE$로 나타내어 태스크 V는 프로세싱 요소 PE에서 수행됨을 의미한다. 태스크 τ_i가 프로세싱 요소 PE에서 수행될 때는 최대 수행시간(worst case execution time) e_i를 가진다. 태스크가 최대 수행시간을 가짐에 따라, 상위의 프로세스 역시 최대 입출력 시간지연(maximum input-to-output latency)을 가진다. 모든 프로세스는 동일한 시스템 주기로

표 1 프로세스 네트워크와 태스크 그래프에 사용된 기호

기 호	설 명
σ_i	프로세스
τ_i	태스크
P_j	프로세싱 요소
S_j	P_j의 성능비례계수
S_j^i	S_j의 i^{th}번째 값
$c_j(S_j)$	P_j의 S_j에 대한 비용
$A(\ )$	태스크 할당 함수
e_i	$A(\tau_i)$상에서 τ_i의 최대수행시간

수행되며, 그 주기값은 시스템의 시간당 처리량을 결정한다. 반면에 태스크는 서로 다른 주기를 가질 수 있다.

3.2 사례 연구 - 디지털 복사기 모델

사례연구로서 디지털 복사기 시스템을 프로세스 네트워크와 태스크 그래프로 기술한다. 디지털 복사기는 다양한 전기 및 기계 부품으로 구성되고 실시간 처리 제약을 가지므로 전형적인 내장형 실시간 시스템이라 할 수 있다. 디지털 복사기의 주요 구성요소로서 스캐너, 레이저 빔 프린터, OPC(organic photo conductive) 드럼, 급지기, 전동 벨트 등이 있다. 복사는 다음과 같이 6 단계의 프로세스들로 이루어진다.

- 급지(feed in): 전동벨트를 통해 복사지를 주입한다.
- 노출(exposing): 복사 원본을 램프에 노출시키고 반사된 이미지를 디지털 이미지로 메모리에 저장한다.
- 이미징(imaging): 저장된 이미지를 이용하여 OPC 드럼을 충전한다.
- 인화(developing): 인화한다.
- 출지(feed-out): 복사본을 출력한다.
- 세정(clean-up): OPC 드럼을 세정한다.

그림 3은 디지털 복사기의 프로세스 네트워크로서 그림 2와 동일한 시스템이다. 그림 3의 이미징 프로세스는 그림 2의 σ_3에 해당하며 σ_3의 태스크 그래프가 그림 2에 나타나 있다. 디지털 복사기 프로세스의 시간적 특성은 표 2와 같다.

표 2에서 볼 수 있듯이, 이미징 프로세스가 1.8초의 가장 큰 시간지연을 가진다. 프로세스 네트워크로 표현되는 시스템은 파이프라인(pipeline) 형태로 수행되므로 가장 큰 시간지연 값에 의해 시스템의 동작 주기가 결정된다. 위의 디지털 복사기에서 복사 주기는 가장 큰 값인 1.8초로 결정되며 이는 33.3 cpm에 해당한다. 따라서 디지털 복사기의 병목 프로세스는 이미징 프로세스임을 알 수 있다.

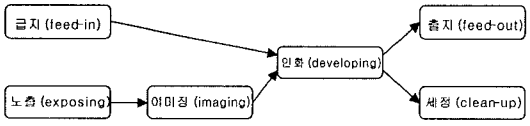

그림 3 디지털 복사기의 프로세스 네트워크

표 2 디지털 복사기 프로세스의 시간적 특성

프로세스	시간지연	주기
급 지	0.3	1.8
노 출	0.5	1.8
이 미 징	1.8	1.8
인 화	0.7	1.8
출 지	0.3	1.8
세 정	0.2	1.8

3.3 문제 정의 및 접근 방법

본 연구의 목적은 병목 프로세스만을 찾아 이를 개선하는 것으로서 리엔지니어링의 비용을 최소화하는데 있다. 소프트웨어는 수정하지 않으며, 단지 병목이 되는 프로세싱 요소만을 교체하는 방법을 사용한다. 성능 리엔지니어링을 명확히 정의하기 위해서 다음과 같은 사항이 필요하다.

(1) 시스템의 프로세스 네트워크와 태스크 그래프
(2) 태스크 할당과 스케줄링
(3) 실시간 처리량으로 표현되는 새로운 성능 요구사항
(4) 각 프로세싱 요소들의 성능 개선에 따른 비용 함수

위와 같은 사항에 의해 리엔지니어링 문제가 정의되며, 이를 풀기 위해 본 연구에서 제시하는 방법은 아래와 같이 구성된다.

단계 1-프로세스의 시간지연과 성능 요구 사항을 비교하여 PN(process network)에서 병목이 되는 프로세스를 찾는다.

단계 2-병목 프로세스의 시간지연을 조정하기 위해 교체되어야 할 프로세싱 요소 및 성능비례계수를 결정한다.

두번째 단계에서는 태스크 스케줄링에 따라 선형제약식들을 유도하고, 이로부터 성능비례계수를 결정한다. 이 때 목적함수는 $\min(\sum_{p_i \in p_E} c_i(S_i))$로 표현되며, 이는 리엔지니어링 비용을 최소화해야 함을 의미한다.

4. 성능 리엔지니어링

4.1 단계 1 - 병목 프로세스 분석

실시간 처리량을 개선하기 위해 요구되는 시간지연을 L이라 하자. 프로세스 σ_i의 입출력 시간지연 L_i가 L보다 클 때 σ_i는 병목 프로세스가 된다. 따라서 병목 프로세스를 찾아내기 위해서는 프로세스의 최대 입출력 시간지연을 정확하게 계산해야 한다. 프로세스의 최대 입출력 시간지연은 태스크 그래프 상에서 임계경로(critical path)를 따라 태스크들의 최악수행시간을 합하여 구할 수 있다. 그러나, [3]에서 언급한 바와 같이 이는 NP-hard 문제로서 일반화된 모델에서는 이를 구하기가 매우 어렵다. 따라서, 본 논문에서는 다음과 같은 가정을 세운다.

(A-1) 태스크 할당과 스케줄링이 정적(static)이다.

즉, 수행시 태스크의 전이 (migration) 및 스케쥴링의 순서가 변하지 않는다.

(A-2) 태스크 스케쥴링은 비선점형(non-preemptive)이다.

(A-3) 동일한 프로세스내의 모든 태스크는 같은 주기를 가진다. 만약 주기가 다른 태스크가 존재하면, 언롤(unroll)하여 주기가 같은 태스크로 변환한다 (주기가 작은 태스크를 여러 개의 주기가 큰 태스크로 복제하면 동일한 결과를 얻을 수 있다.)

(A-1)과 (A-2)는 임의의 프로세싱 요소에서의 태스크의 수행순서를 고정되도록 한다. 따라서, 프로세스의 최대 입출력 시간지연은 선행관계(precedence)를 고려하여 태스크들의 최악 수행시간의 합으로 구할 수 있다. 이는 현실적으로 무리가 없는 가정이다. 일반적으로 내장형 실시간 시스템은 태스크 할당 및 스케쥴링 우선순위가 고정되도록 설계되기 때문이다[5, 11]. 마지막으로 다음의 가정을 세운다.

(A-4) 프로세스들은 서로 독립적이다. 즉, 프로세스들은 프로세싱 요소들을 공유하지 않는다.

둘 이상의 프로세스가 프로세싱 요소를 공유하게 되면 태스크 스케쥴링이 고정되지 않으므로 프로세스의 입출력 시간지연이 변동적이다. 본 연구에서는 프로세싱 요소의 공유를 배제한다. 위의 4가지 가정 하에서 프로세스의 최대 입출력 시간지연은 태스크 그래프상의 최장 경로로 결정된다.

4.2 단계 2 - 시간지연 조정

4.2.1 제약조건 유도

태스크의 할당과 스케쥴링에 의하여 결정되는 선행관계를 이용하여 선형제약식을 유도한다. 이 때 선행관계를 명확히 파악하기 위하여 주어진 태스크 그래프를 변형한다. 만약 태스크 τ_i와 τ_j에 대해 τ_i가 τ_j보다 먼저 스케쥴되면 부가적인 선행관계 $\tau_i \rightarrow \tau_j$가 존재한다. 그림 4와 그림 5는 디지털 복사기 프로세스 σ_3의 태스크 그래프와 태스크 스케쥴을 보여준다. 그림 4에서 각 태스크의 최악 수행시간은 각 노드의 옆에 표시되어 있다.

그림 5는 그림 4의 태스크 그래프가 두 프로세싱 요소 P_1과 P_2에 의해 스케쥴된 결과를 보여준다. 그림 4에서 태스크 τ_4와 τ_2의 사이에는 선행관계가 없지만 P_2상에서 τ_4가 τ_2보다 먼저 수행된다. 이렇게 스케쥴링에 의해 성립되는 선행관계 $\tau_4 \rightarrow \tau_2$를 그림 6의 변형된 태스크 그래프에서 점선으로 표시하였다.

L을 새롭게 요구되는 최대 입출력 시간지연이라 하자. $s(\tau_i)$와 $f(\tau_i)$를 각각 τ_i의 시작 시간과 종료시간이라 하고, $S(\tau_i)$를 τ_i가 할당된 PE의 성능비례계수라 하면, 변형된 태스크 그래프 상의 마지막 태스크 τ_i에 대하여 다음과 같은 제약식을 얻을 수 있다.

$$f(\tau_i) \le L \Leftrightarrow \max_{\tau_j \in Pred(\tau_i)}\{f(\tau_j)\} + e_i S(\tau_i) \le L \quad (1)$$

식 (1)에서 $Pred(\tau_i)$는 τ_i에 선행하는 태스크들의 집합이다. 따라서 식 (1)은 τ_i가 모든 선행 태스크의 종료 후에 수행될 수 있으며, τ_i의 실제 수행시간은 τ_i가 할

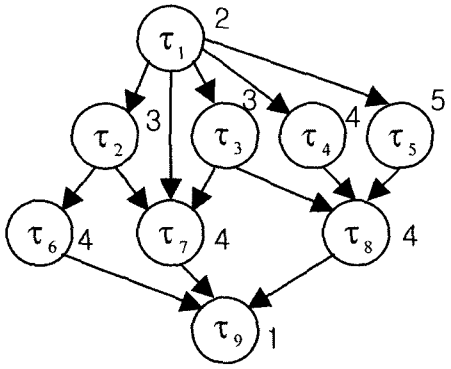

그림 4 디지털 복사기 프로세스 σ_3의 태스크 그래프

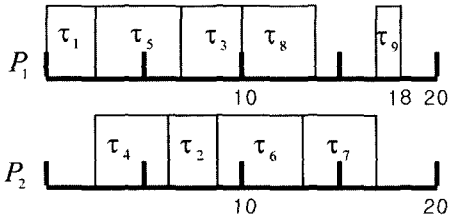

그림 5 프로세스 σ_3의 태스크 스케쥴

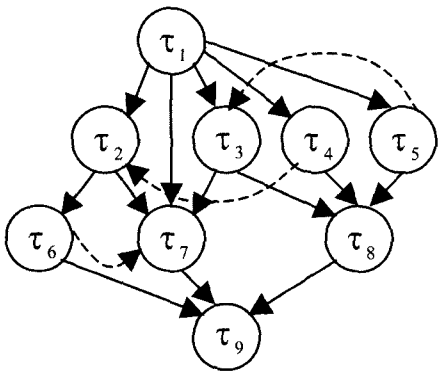

그림 6 프로세스 σ_3의 변형된 태스크 그래프

당된 프로세싱 요소의 성능비례계수에 의해 결정된다는 것을 의미한다. 만약 $S(\tau_i) < 1.0$ 이면 τ_i가 현재 보다 수행 시간이 작아짐을 의미한다.

식 (1)에서 $f(\tau_j)$를 $s(\tau_j) + e_jS(\tau_j)$로 회귀적으로 치환하면 시작시간 $s(\cdot)$와 종료시간 $f(\cdot)$를 소거할 수 있다. e_i는 상수이므로 식 (1)은 결국 $S(\tau)$를 변수로 하는 일차의 선형 제약식이 된다.

표 3 선형제약식에 사용되는 기호 및 설명

기 호	설 명
$s(\tau_i)$	τ_i의 시작 시간
$f(\tau_i)$	τ_i의 종료시간
$S(\tau_i)$	$A(\tau_i)$의 성능비례상수
$Pred(\tau_i)$	τ_i의 선행 태스크 집합

4.2.2 사례 연구 : 디지털 복사기의 선형제약식 유도

앞서 제시된 디지털 복사기의 새로운 성능조건으로서 복사 주기가 15로 되어야 한다고 가정하자. 따라서, 병목 프로세스 σ_3의 최대 입출력 시간지연을 15보다 작게 만들면 새로운 성능조건을 만족시킬 수 있다.

그림 6은 변형된 태스크 그래프를 보여주며, 점선은 부가적인 선행관계를 나타낸다. 그림 6의 태스크 그래프는 그림 5와 같이 프로세싱 요소 P_1과 P_2에 의해 수행되며, 각각의 성능비례계수는 S_1과 S_2라 하자. 이 때 태스크 τ_9에 대해서 선형제약식을 표현하면 다음과 같다.

$$\max\{f(\tau_6), f(\tau_7), f(\tau_8)\} + e_9S_1 \le 15 \quad (2)$$

$S_1 = S(\tau_9)$이고 추가된 선행조건 $\tau_6 \rightarrow \tau_7$에 의해 $f(\tau_6) < f(\tau_7)$이므로, 식 (2)는 $\max\{f(\tau_7), f(\tau_8)\} + e_9S_1 \le 15$로 변형할 수 있다. max{} 연산자를 소거하면 다음과 같다.

$$f(\tau_7) + e_9S_1 \le 15 \text{ and } f(\tau_8) + e_9S_1 \le 15 \quad (3)$$

위 식을 회귀적으로 대입하여 풀면 다음과 같은 부등식을 얻는다.

$$(e_1 + e_9)S_1 + (e_4 + e_2 + e_6 + e_7)S_2 \le 15$$
$$(e_1 + e_5 + e_3 + e_9)S_1 + e_7S_2 \le 15$$
$$(e_1 + e_5 + e_3 + e_8 + e_9)S_1 \le 15$$
$$(e_1 + e_8 + e_9)S_1 + e_4S_2 \le 15 \quad (4)$$

그림 4에 따라 변수 e_i를 치환하면 다음과 같다.

$$3S_1 + 15S_2 \le 15, \quad 11S_1 + 4S_2 \le 15,$$
$$15S_1 \le 15, \quad 7S_1 + 4S_2 \le 15$$

그림 7은 S_1과 S_2의 해공간을 이차원 평면상에 도시한 것이다. 흐린 영역이 위의 선형제약식을 만족하는 해의 영역이다.

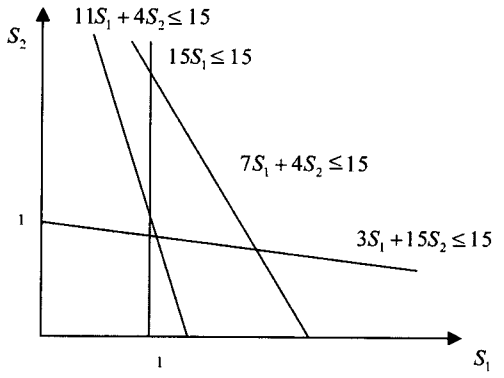

그림 7 성능비례계수의 해공간

4.2.3 성능비례계수의 도출

선형제약식을 유도한 후에 이를 만족하는 최선의 성능비례계수 벡터 $\langle S_1, S_2, \ldots, S_k \rangle$를 구한다. 이 때 하드웨어 비용 $C(S_1, S_2, \ldots, S_k) = \sum_{P_j \in PE} c_j(S_j)$의 최소화를 목표로 한다.이 때 $c_j(S_j)$는 S_j에 대해 감소하며 이산적인 값을 가진다. 성능비례계수가 커질수록(성능향상이 적을 수록) 비용이 감소하고, 대체할 수 있는 하드웨어의 선택이 유한하기 때문이다. 이러한 형태의 문제는 정수 선형 프로그래밍(integer linear programming) 문제의 일종이다. 일반적으로 선형 프로그래밍 문제에서는 심플렉스(simplex)와 같은 방법을 사용하여 최적해를 구할 수 있는 반면, 정수 프로그래밍 문제에서는 심플렉스와 같은 기법은 최적해를 구하지 못하는 것으로 알려져 있다. 선형 프로그래밍 기법을 사용한다면, 실수 공간에서 찾은 최적해의 소숫점 이하를 버리고 정수 공간의 근사해를 구할 수 있다. 하지만 리엔지니어링은 오프라인 작업이므로 시간이 걸리더라도 최적의 해를 구하는 것이 의미를 가진다.

본 논문에서는 다차원 공간의 분할을 통해 최적해를 찾는 알고리즘을 제시한다. 탐색 공간(search space)을 성능비례계수의 인덱스 값을 이용하여 분할한다. k개의 프로세싱 요소들에 대해 임의의 해를 $\gamma = \langle S_1^{i_1}, S_2^{i_2}, \ldots, S_k^{i_k} \rangle$이라 하자. 인덱스 i_j는 S_j의 감소순으로 주어진다고 하자. 표 4의 예를 들면, $S_1^{i_1}$의 인덱스 i_1은 {1,2,3,4}를 취하며 그에 따른 성능비례계수는

$\{S_1^1=1, S_1^2=0.8, S_1^3=0.5, S_1^4=0.3\}$와 같다. 이러한 인덱스를 이용하여 전체 해공간을 분할할 수 있으며, 분할된 각 부분집합은 $\Gamma_i = \{\gamma | i_1 + i_2 + \ldots + i_k = i\}$라 하자. 해 γ의 비용을 $C(\gamma)$라 하고 Γ_i에서의 최소 비용값을 $C_i^{\min}$라 하자. $c_j(S_j^k)$는 k에 대해 증가하므로 다음과 같은 성질이 성립한다.

$$C_i^{\min} \leq C_{i+1}^{\min} \quad (5)$$

부등식 (5)의 증명은 $i > 1$일 때 임의의 해 $\gamma_\alpha \in \Gamma_i$에 대해 $C(\gamma_\beta) \leq C(\gamma_\alpha)$를 만족하는 $\gamma_\beta \in \Gamma_{i-1}$가 존재함을 보이면 된다. $c_j(S_j^k)$가 감소함수이므로 γ_β는 γ_α의 인덱스 값을 1만큼 줄이면 구할 수 있다. 이에 따라 $\gamma_\beta \in \Gamma_{i-1}$가 존재함을 알 수 있다. 그림 8은 이러한 성질을 이용하여 최적해를 찾는 과정을 보여주고 있다. 그림 8에서 C는 $\langle S_1^1, S_2^2\rangle$의 비용벡터를 나타낸다. $C \geq C_3^{\min}$과 $C_3^{\min} \geq C_2^{\min}$이 성립하므로 $C \geq C_2^{\min}$이 성립한다. 따라서 Γ_{i+1}의 임의의 해는 $C_i^{\min}$보다 같거나 큰 비용값을 가짐을 알 수 있다.

알고리즘의 두 번째 단계는 i를 증가시켜가면서 Γ_i를 검사하는 과정이다. $C^{\min}$을 현재까지 탐색된 해들의 최소 비용값이라 하자. 알고리즘은 $C^{\min} \leq C_i^{\min}$이 될 때까지 해를 찾는다. 앞에서 보인 바와 같이 Γ_{i+1}의 해들은 $C_i^{\min}$보다 같거나 큰 비용값을 가지기 때문이다.

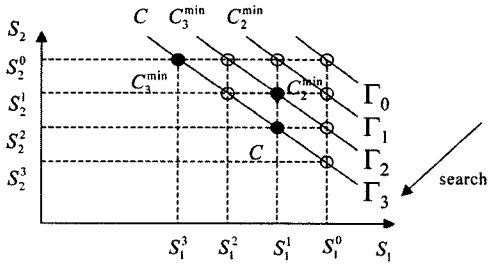

그림 8 공간 분할을 이용하여 최적해를 찾는 알고리즘

위의 과정을 의사코드(pseudo code)로 표현하면 아래와 같다. $C^{\min}$의 초기값을 ∞로 정한 후 $C^{\min}$이 더 이상 감소하지 않을 때까지 검색을 계속한다.

성능비례계수 탐색 알고리즘

```
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
```

4.2.4 사례 연구: 디지털 복사기의 성능비례계수 도출

표 4와 표 5는 성능비례계수에 따른 P_1과 P_2의 비용을 보여준다. 이를 이용하여 제안된 알고리즘이 최적해를 찾는 과정은 다음과 같다.

$i = 0$: $\Gamma_0 = \{\langle 1, 1\rangle\}$
$C_0^{\min} = 0$ $C^{\min} = \infty$ no feasible solution

$i = 1$: $\Gamma_0 = \{\langle 0.8, 1\rangle, \langle 0.8, 0.9\rangle\}$
$C_1^{\min} = 0$ $C^{\min} = \infty$ no feasible solution

$i = 2$: $\Gamma_0 = \{\langle 0.5, 1\rangle, \langle 0.8, 0.9\rangle, \langle 1, 0.7\rangle\}$
$C_2^{\min} = 50$ $C^{\min} = 50$ $\gamma^{\min} = \langle 0.8, 0.9\rangle$

Γ_0과 Γ_1에는 유도된 제약을 만족하는 해가 없으므로 $i = 0$과 $i = 1$일 때 $C^{\min} = \infty$이 된다. $C_2^{\min} = C^{\min} = 50$이므로 알고리즘의 탐색은 $i = 2$일 때 종료한다. 따라서 P_1과 P_2는 각각 20%와 10%의 성능 개선이 필요하게 된다. 이는 성능 개선 비용을 총 50만큼 요구한다.

표 4 프로세싱 요소 P_1의 비용

성능비례계수	1	0.8	0.5	0.3
비용	0	20	50	100

표 5 프로세싱 요소 P_2의 비용

성능비례계수	1	0.8	0.5	0.3
비용	0	20	50	100

5. 결 론

본 논문에서는 내장형 실시간 시스템의 리엔지니어링 문제를 정의하고, 병목 프로세스 분석과 최적의 성능비례계수를 찾는 알고리즘을 제안하였다. 제안된 방법에서는 프로세스 네트워크와 태스크 그래프로서 시스템을 기술하고, 주어진 태스크 스케줄링과 새로운 실시간 처리량으로부터 최적화된 성능비례계수를 찾아낸다.

본 연구 결과는 두 가지 측면에서 큰 의의를 가진다고 할 수 있다. 첫째, 시스템 리엔지니어링 문제를 명확히 도출하였다. 과거에는 시스템 리엔지니어링이 단지 개발의 연장선상에서 다루어져 왔으나, 본 연구에서는 이를 개발로부터 분리하여 새로운 문제로 정의하였다. 둘째, 리엔지니어링을 효과적으로 수행할 수 있는 이론적인 방법론을 제공하였다. 경험적인 방법에만 의존하던 리엔지니어링을 체계화하여 리엔지니어링 비용을 최소화하도록 하였다.

향후에는 본 논문에서 세워졌던 가정들을 완화하고자 한다. 특히 프로세스간의 공유를 배제하였는데, 앞으로는 공유를 허용한 모델에서의 리엔지니어링 기법을 연구하고자 한다. 마지막으로, 제시된 결과를 자동화된 리엔지니어링 도구로 구현하여 유용성을 검증하고자 한다.

참 고 문 헌

[1] S. Nadjm-Tehrani and J.-E. Stromberg. "Proving dynamic properties in an aerospace applications," In Proceedings of IEEE Real-Time Systems Symposium, pages 2-10, December 1995.

[2] 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 91-99, June 1997.

[3] 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.

[4] G.Arona 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.

[5] N. Kim, M. Ryu, S. Hong, and H. Shin. "Experimental assessment of the period calibration method: A case study," The Journal of Real-Time Systems, Vol. 17, No. 1, pp. 41-64, July, 1999.

[6] Microtec Research Inc. "VRTX32/86 User's Guide," Microtec Research Inc., May 1991.

[7] Wind River Systems. "The next generation of embedded development tools," A Wind River Systems White Paper, 1998.

[8] 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.

[9] E. Lee and T. Parks. "Dataflow process networks," IEEE Proceedings. 83(5):773-801, May 1995.

[10] G. Kahn. "The semantics of simple language for parallel processing," In the IFIP Congress 74, 1974.

[11] C. Locke. "Software architectures for hard-real-time applications: cyclic executives vs. fixed priority executives," The Journal of Real-Time Systems, 4(1):37-53, 1992.

홍 성 수

1982~1986 서울대학교 컴퓨터공학과 (B.S.). 1986~1988 서울대학교 컴퓨터공학과(M.S.). 1988~1989 한국전자통신연구소(연구원). 1989~1994 University of Maryland, Department of Computer Science (Ph.D.). 1994~1995 University of Maryland, Department of Computer Science (Faculty Research Associate). 1995~1995 Silicon Graphics Inc. (Member of Technical Staff). 1995~1997 서울대학교 전기공학부 전임강사. 1997~2001 서울대학교 전기공학부 조교수. 2001~현재 서울대학교 전기컴퓨터공학부 부교수