주기 조정과 커널 자동 생성을 통한 다중 루프 시스템의 구현

Synthesizing Multi-Loop Control Systems with Period Adjustment and Kernel Compilation

홍 성 수, 최 종 호, 박 홍 성
(Seongsoo Hong, Chong-Ho Choi, Hong Seong Park)

Abstract : This paper presents a semi-automatic methodology to synthesize executable digital controller software in a multi-loop control system. A digital controller is described by a task graph and end-to-end timing requirements. A task graph denotes the software structure of the controller, and the end-to-end requirements establish timing relationships between external inputs and outputs. Our approach translates the end-to-end requirements into a set of task attributes such as task periods and deadlines using nonlinear optimization techniques. Such attributes are essential for control engineers to implement control programs and schedule them in a control system with limited resources. In current engineering practice, human programmers manually derive those attributes in an ad hoc manner: they often resort to radical over-sampling to safely guarantee the given timing requirements, and thus render the resultant system poorly utilized. After task-specific attributes are derived, the tasks are scheduled on a single CPU and the compiled kernel is synthesized. We illustrate this process with a non-trivial servo motor control system.

Keywords : multi-loop control system, embedded digital controller, compiled kernel, control theory, real-time scheduling, task graph, nonlinear optimization, and integer programming

I. 서론

마이크로 프로세서 기술의 발달로 인하여 많은 제어 시스템에서 사용되던 아날로그 제어기가 빠른 속도로 내장 디지털 제어기(embedded digital controller)로 대체되고 있다. 디지털 제어기의 응용 분야들이 다양해지고 제어 시스템들이 복잡해지면서, 복잡한 제어 기능들을 수행하기 위하여 높은 계산 능력을 갖는 프로세서들을 필요로 하게 되었다. 근래에는 고성능 마이크로 프로세서를 장착한 내장 디지털 제어기가 널리 사용되고 있다.

내장 디지털 제어기가 점점 복잡해지면서 설계 상에서 두 가지 심각한 문제가 발생한다. 첫째는 시스템의 시간 제약들을 보장하기가 어렵다는 점이며, 둘째는 제어기의 여러 활동들을 스케쥴링 하는 것이 어렵다는 점이다. 첫번째의 문제는 실시간 스케쥴링 이론의 제약 때문에 발생한다. 실시간 시스템은 시스템의 외부 입력과 출력들의 시간 관계를 제한하는 양극단 시간 제약(end-to-end timing constraints) 만을 갖는 것이 보통이다. 예를 들어, 시스템의 센서를 읽어 10ms 이내에 출력장치(actuator)를 구동해야 한다는 것이나, 두 개의 센서로부터의 입력이 1ms 이내로 동기되어야 한다는 것이다. 하지만, 실시간 스케쥴링 알고리즘들은 태스크의 시간 정보들인 주기나 종료 시한(deadline)들과 같은 것에 따라 스케쥴링을 하기 때문에 양극단 시간 제약으로 표현된 시간 제약을 직접적으로 보장해 줄 수 없다.

현재의 실시간 시스템에서는 관행적으로 두 가지 방법에 의해 태스크의 시간 특성들을 도출한다. 이것들을 설명하기 위하여, 다음과 같은 시간 제약과 속성들을 갖는 세 개의 제어 루프를 생각해 보자.

접수일자 : 1996. 10. 2., 수정완료 : 1997. 3. 4.
홍성수, 최종호 : 서울대 전기공학부, ERC-ACI, ASRI
박홍성 : 강원대 제어계측공학과

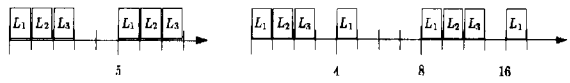
루프	루프 처리 주기	제어기 실행 지연
L_1	5ms	1ms
L_2	8ms	1ms
L_3	15ms	1ms

두번째 열의 루프 처리 주기(loop processing period)는 각 제어 루프의 실행 주기의 최대 값을 표현하며, 마지막의 제어기 실행 지연(controller execution delay)은 각 제어 태스크의 실행 시간을 표현한다. 예를 들어, 루프 L_1의 제어 태스크는 1ms를 소요하며, 이 루프는 최소한 5ms에 한 번씩 수행되어야 한다. 여기서는 세 개의 제어 루프들이 서로 간의 의존 관계를 갖는 세 개의 독립적인 태스크들을 포함한다고 가정한다.

시스템에 부과된 시간 제약 조건들과 태스크들 사이의 상호 의존(dependence) 제약 조건들을 만족시키기 위하여, 두 가지 간단한 휴리스틱(heuristic)을 사용할 수 있다. 하나는 세 개의 제어 루프를 가장 작은 루프 처리 주기로 묶는 것이다. 위의 예에서 L_2와 L_3의 루프 처리 주기를 5ms로 줄이는 것이다. 세 개의 모든 태스크들이 동일한 주기를 공유하기 때문에 그림 1의 (A)와 같이 일렬로 5ms의 주기로 수행될 수 있다.

전체 제어 태스크들이 선택된 루프 처리 주기 내에 수행될 수 없다면 좀 더 개선해야 할 필요가 있다. 이런 경우 두번째 방법이 이용될 수 있다. 각 제어 루프에 대하여, 루프의 제어 주기 보다 크지 않은 2의 제곱 수 중 가장 큰 것을 찾는 것이다. 위의 예에서, $\{2^2, 2^3, 2^3\}$이 그러한 수들이 된다. 각각 세 개의 제어 루프들의 새로운 제어 주기가 되며, 각 제어 태스크는 8ms의 시간 기간 내에 정적인 슬롯 번호를 할당받게 된다. 시간 슬롯의 크기가 1ms라고 하고 0에서 7까지로 번호가 지정된다고 하면 L_1은 0과 4 슬롯을,

L_2는 슬롯 1을, L_3은 슬롯 2를 각각 할당받게 된다. 그림 1의 (B)는 이러한 슬롯 할당을 보이고 있다. 이 방법을 *pinwheel* 스케쥴링이라고 부른다 [7].



(a) (b)

그림 1. (a) 간단한 스케쥴링, (b) pinwheel 스케쥴링.
Fig. 1. (a) A simple scheduling; and (b) pinwheel scheduling.

이러한 방법들은 간단하고 구현하기 쉽지만, 만족스러운 결과를 주지 못한다. 이 방법들을 사용하면, 루프 처리 주기가 불필요하게 작아지는 경향이 있어 제어 루프들의 실행 빈도가 증가하게 되어, CPU의 대역폭(bandwidth)을 불필요하게 낭비하게 된다. 그림 1(b)에서 L_3의 루프 처리 주기는 최대값의 약 1/2로 줄어들어 태스크의 이용률이 거의 두 배가 된다. 더구나 제어 루프들 사이에 복잡한 데이터 의존성이 존재하는 경우, 이 pinwheel 스케쥴링 기법을 적용하기가 힘들다. 데이터 의존성은 제어 루프들간의 선행 제약(precedence constraints)을 만들어 내기 때문에 제어 루프들이 서로 다른 주기를 갖게 되면 선행 제약을 만족시키기 힘들어 진다. 결과적으로 시스템이 복잡해질수록 위와 같은 주먹구구의 방법에 의한 설계는 실패할 가능성이 커지게 된다.

최근 Gerber팀은 [6]에서 본 논문의 기초가 되는 설계 방법론을 제시하였다. 이 설계 방법에서는 태스크 시간 속성들을 반자동적으로 유도한다. 이것은 응용의 태스크 그래프와 태스크들의 입력과 출력에 대한 양극단 시간 제약을 받아들여, 태스크 의존적인 속성들을 자유 변수(free variables)로 갖는 비선형 부등식(non linear inequalities)으로 변환을 한 후, 응용의 CPU 요구를 줄이면서 이 부등식들을 푸는 방법이다.

하지만, [6]에서의 설계 방법론에서는 태스크의 속성들만을 유도하고 태스크 스케쥴링에 대한 문제를 특별히 언급하지 않고 있다. 더구나 이것은 가상적인 실시간 시스템에 적용하기 위한 것이었으므로 로봇이나 CNC 공작 기계와 [12, 8] 같은 내장 제어 응용들이 갖는 구현상의 제약을 표현하는데 부족하다. 이 방법에서는 옵셋이라는 태스크의 특별한 속성을 불필요하게 도입함으로써 태스크의 스케쥴링을 어렵게 만든다. 실제로 옵셋을 갖는 태스크들을 최적으로 스케쥴링하는 알고리즘은 아직 존재하지 않고 있다[13]. 그래서 본 논문에서는 내장 실시간 응용(embedded real-time application)들의 구현에 적합하도록 이전의 방법을 확장한다. 그리고 태스크의 속성들을 유도하는 새로운 알고리즘(algorithm)을 제시한다. Gerber등[6]이 제시된 알고리즘이 단순한 탐색에 의존하여 실제 응용에 사용되기 부적합하였지만, 새로운 알고리즘은 branch-and-bound 휴리스틱에 기반을 두어 빠른 탐색을 할 수 있다.

이러한 확장의 결과로 내장 디지털 제어기의 개발의 효율성을 증가시키는 설계 도구를 만들 수 있다. 이 설계 도구의 입력으로는 태스크 그래프(task graph)와 양극단 시간 제약이 있다. 출력은 제어 알고리즘을 구현하는 응용 프로그램 코드와 태스크 스케쥴링과 I/O 명령들을 구현하는 커널 프로그램을 포함하는 컴파일된 커널(compiled kernel)이다. 이 커널은 내장 디지털 제어기에 다운로드된다. 그림 2는 설계 도구의 상호 동작을 그림으로 표현한 것이다. 이것은 주기 할당기(period assigner)와 커널 합성기(kernel synthesizer)의 두 가지로 구성된다. 주기 할당기는 태스크 그래프와 양극단 시간 제약을 입력으로 받아 주기와 종료 시한과 같은 태스크 시간 속성을 유도한다. 이것은 제약 유도기(constraint deriver)와 비선형 최적기(nonlinear optimizer)로 구성되며, 제약 유도기는 양극단 시간 제약을 주기와 종료 시한에 대한 비선형 제약 집합으로 변환을 한다. 이때, 변환된 비선형 제약 집합은 유도된 시간 제약들이 원래의 양극단 시간 제약을 만족시킬 수 있어야 한다. 제약 유도기는 또한 양극단 시간 제약의 부분 집합이 쉽게 만족될 수 있도록 태스크 그래프를 재구성한다. 다음으로 비선형 최적기는 시스템의 CPU 이용률을 최소화하는 방향으로 유도된 제약식의 해(solution)를 찾는다. 그림 2에서 볼 수 있듯이 비선형 최적기가 첫번째 시도에서 유용한 해를 찾지 못한다면 주기를 할당하는 방법은 반복 작업이 된다. 이 때, 반복의 횟수를 줄이기 위해 시스템 설계자의 개입이 필요할 수도 있다. 이러한 방법으로 주기 할당기에 의하여 태스크의 속성이 유도되면, 커널 합성기는 태스크와 커널의 코드를 설계자가 제공한 코드를 이용하여 태스크 통신과 합성 커널의 스케쥴링 방법을 만든다. 이러한 과정을 거치면 실행 가능한 프로그램인 합성 커널이 만들어진다.

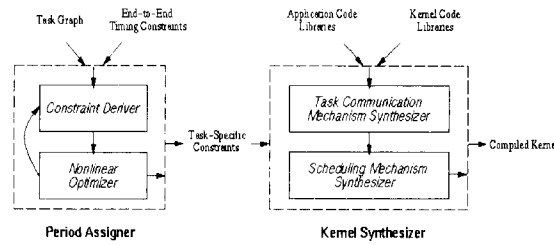


그림 2. 도구의 내부 동작.
Fig. 2. Inter-workings of the tool set.

본 논문은 다음과 같이 구성된다. 2절에서는 본 논문의 접근 방법을 설명하기 위한 제어 응용을 소개하고, 태스크 그래프와 양극단 시간 제약을 정의한다. 3절에서는 비선형 정수 계획법(nonlinear integer programming)을 이용하여 태스크의 시간 제약을 유도하는 과정을 제시한다. 5절에서 결론과 앞으로의 할 일에 대하여 기술한다.

II. 제어 시스템 모델

폐쇄 루프 제어 시스템(closed-loop control system)에서의 내장 디지털 제어기는 센서들과 제어 프로그램들 그리고 출력 장치(actuator)들로 구성이 된다. 제어 프로그램들은 입력 센서들로부터 샘플된 데이터에 반응하는 출력장치들을 구동하기 위하여 주기적으로 수행된다. 시스템이 많은 제어 변수들을 필요로 하고 고도의 안전성과 견고성을 요구하게 됨에 따라, 연관된 제어 프로그램들은 더욱 복잡해지게 된다. 그 결과로 내장 디지털 제어기는 상호 연관된 요소들을 갖는 다중 제어 루프들로 구성된다. 그림 3은 각각 두 개의 종속된 제어 프로그램을 갖는 다중 루프 제어 시스템을 보이고 있다.

이들 제어 프로그램들은 복잡한 제어 알고리즘들을 구현하기 때문에, 자연적으로 복잡한 내부 구조와 상호 의존 관계를 갖게 된다. 본 절에서는 제어 프로그램의 구조를 나타내는 도구로서 사용할 태스크 그래프(task graph)를 정의한 후, 내장 디지털 제어기의 시간적 행동을 표현하는데 사용할 수 있는 여러 종류의 양극단 시간 제약을 설명한다.

2.1 제어 시스템의 예

그림 3은 폐쇄 루프 서보 제어 시스템의 블럭도(block diagram)를 보이고 있다. 이 시스템은 2차원 평면 위를 움직이는 기계 장치를 제어하며, 두 개의 서보 모터에 의하여 X, Y축으로 움직인다. 이것은 CNC(Computerized Numerical Controller) 공작 기계에서 자주 접할 수 있는 전형적인 서보 제어 시스템이다. 이 시스템에서 두 개의 서보 제어 루프는 X, Y축을 각각 제어한다.

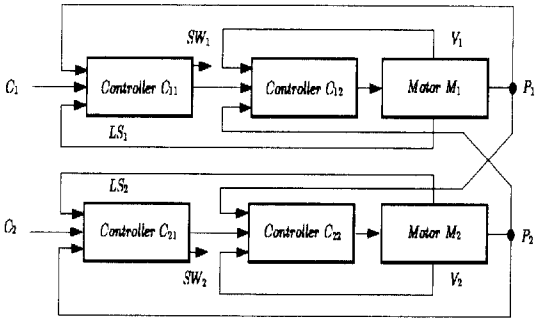


그림 3. 다중 루프 제어 시스템의 블록도.
Fig. 3. A multi-loop feedback control system.

제어 루프 1은 5개의 센서에서 입력을 받고, 두 개의 출력 장치를 구동시킨다. 입력은 다음과 같다.

1) C_1은 운동 명령을 나타낸다. 이런 명령은 주 메모리에 보관되고, 제어기 C_{11}이 주기적으로 메모리에서 읽는다.
2) V_1은 서보 모터의 각속도(angular velocity)를 나타낸다.
3) P_1은 X축 위치를 나타낸다.
4) P_2는 Y축 위치를 나타낸다. 제어 루프는 두 개의 서보 모터의 움직임을 조정하기 위해 P_2에서 데이터를 읽는다.
5) LS_1은 한계 센서(limit sensor)로부터의 입력을 나타낸다. 한계 센서는 장치가 끝까지 움직였을 때 신호를 발생시킨다. 기계적 손상을 피하기 위하여, 제어기는 이 신호에 빨리 반응을 해야만 한다.

출력은 다음과 같다.

1) M_1은 서보 모터를 의미한다.
2) SW_1은 LS_1로 부터 발생한 신호를 감지하면 장치의 주 전원을 차단하는 차단 스위치이다.

제어 루프 2는 동일한 I/O 연결을 갖는다. 본 논문에서는 이 서보 제어 시스템을 이용하여 우리의 접근 방법을 설명하도록 한다.

2.2 태스크 그래프

그림 4는 제어 예제를 표현하는 태스크 그래프를 보이고 있다. 태스크 그래프에서 원은 주기적인 태스크를 표현하며, 사각형은 두 태스크 사이의 통신 버퍼를 의미한다. 태스크는 고정된 주기로 반복하여 수행되는 단순한 코드의 모임으로 표현될 수 있으며, 태스크가 수행될 때 태스크는 입력 버퍼에서 데이터를 읽고 출력 버퍼에 데이터를 쓴다. 프로그래머는 구현 단계에서 자신의 취향이나 사용하는 라이브러리에 따라 태스크들과 태스크들 사이의 상호연결을 결정한다. 그래서 태스크들과 태스크들 사이의 데이터 버퍼를 이용하여 제어 프로그램을 완벽하게 표현할 수 있다.

태스크 그래프의 위와 아래에 굵은 선으로 표현된 사각형은 외부 입력과 출력을 나타낸다. 입력으로는 센서와 메모리에 저장되어 있는 동작 명령(motion commands)들이 있고, 출력으로는 출력 구동기(actuator)와 스위치가 있다.

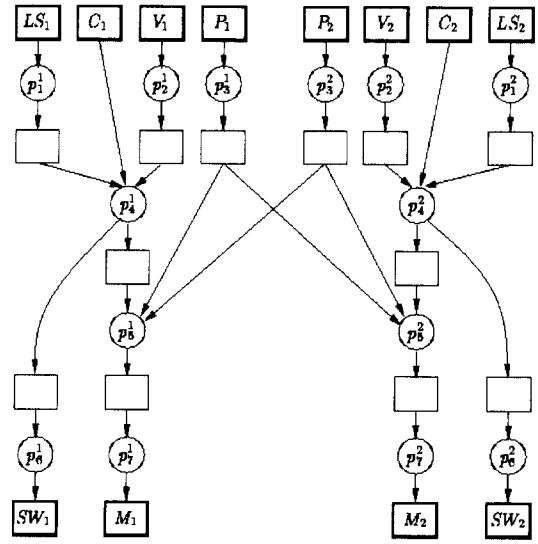


그림 4. 내장 실시간 제어기의 태스크 그래프.
Fig. 4. The task graph of the embedded real-time controller.

그림 3의 블럭도를 이용하여 연관된 코드 모듈들을 연결하여 태스크 그래프를 만들 수 있다. 두 개의 주 제어 프로그램 C_{11}과 C_{12}가 태스크 그래프에서 p_4^1과 p_5^1로 표현된다. 이 태스크들은 해당하는 제어 알고리즘들을 구현하고 있으며, 태스크 p_1^1, p_2^1, p_3^1들은 센서 LS_1, V_1, P_1에서 입력을 받아들인다. 비슷하게 태스크 p_6^1, p_7^1은 SW_1과 M_1에 출력하는 작업을 한다. 제어 루프 2는 1과 동일한 구조를 갖고 있다. (태스크 그래프에서 윗첨자 1과 2는 그림 3의 두 개의 제어 루프를 나타낸다.)

시간적인 관점에서 볼 때, 태스크는 입력 버퍼에서 출력 버퍼로의 제어기의 수행 지연을 표현한다. 이 때 통신 버퍼는 전송중인 데이터를 단지 저장만 하는 수동적인 개체(passive entity)이다. 그래서 데이터를 버퍼에 쓰거나 버퍼에서 읽을 때 발생하는 지연은 태스크의 수행 지연에 포함된다. 데이터 버퍼와는 달리, 센서가 연결된 하드웨어에 의하여 처리된다고 가정하게 되면, 센서를 읽는 작업 역시 일정 정도의 지연이 발생한다. 하지만 본 논문에서는 센서와 출력 구동기들을 데이터 버퍼와 동일하게 수동적 개체로 간주하기로 한다.

태스크 그래프를 다음과 같이 방향성 그래프 $G(V, E)$로 표현한다.

- $V = P \cup B$, 이 때 $P = \{p_1, \ldots, p_n\}$는 태스크의 집합이고, $B = \{b_1, \ldots, b_m\}$는 데이터 버퍼의 집합이다.
- $E \subset (P \times B) \cup (B \times P)$은 태스크와 버퍼 사이의 방향성 간선(directed edge)의 집합이다. $p_i \rightarrow b_j$는 태스크 p_i의 버퍼 b_j에 대한 쓰기 접근을, $b_j \rightarrow p_i$ 는 p_i의 b_j에 대한 읽기 접근을 의미한다. 무결성 문제(consistency problem)[1]를 회피하기 위해, 본 논문에서는 하나의 버퍼에 여러 개의 입력 간선을 허용하지 않는다. 여러 개의 입력 간선들을 갖는 버퍼는 여러 개의 동일한 것

1) 공유하는 데이터 버퍼에 대해 두 개의 쓰기 접근이 동시에 발생하게 되면, 무결성이 보장되지 않는 상태에 놓일 수 있게 된다. 이것은 어떤 쓰기 접근이 뒤에 발생하는 것인지 예측할 수 없기 때문이다.

으로 복사되어, 각각 하나의 간선으로 연결된다.

양극단 시간 제약이 제어 루프의 수행 행태를 제한하므로, 제어 루프를 정의해야 한다. 제어 루프는 센서들과 출력 구동기들 사이의 관계로서 정의된다. 센서 $S_1,...,S_k$를 사용하는 제어기가 출력 구동기 $A_1,...,A_l$을 갱신하는 경우를 가정해 보자. 이 경우 연관된 제어 루프는 "$L(S_1,...,S_k \mid A_1,...,A_l)$"로 표현된다. 제어 루프와 연결된 센서와 구동기 사이의 경로를 따라감으로써, 태스크 그래프로부터 제어 루프의 제어 프로그램을 얻을 수 있다.

2.3 양극단 시간 제약

제어 시스템은 많은 시간 제약을 가질 수 있지만, 이러한 제약들은 다음과 같이 세 가지로 분류될 수 있다.

1) *동기 제약.* 여러 센서에서 샘플된 데이터가 하나의 출력 장치를 구동하는데 사용될 때, 동기 제약은 읽혀지는 샘플들간의 시간을 제약한다. 예를 들어, 제어 루프가 세 개의 입력 센서 S_1, S_2, S_3에서 샘플된 데이터를 사용하여 출력 장치 A를 구동시키고, 이 때 샘플된 시간이 t_1, t_2, t_3이라고 하자. "$Sync(S_1, S_2, S_3 \mid A)$"로 표현된 동기 제약은

$$\max(|t_1-t_2|,|t_1-t_3|,|t_2-t_3|) \le Sync(S_1, S_2, S_3 \mid A)$$

을 의미한다.

2) *최대 허용 유효 시간(MAVT: Maximum allowable validity time).* 이 제약은 센서에서 출력 구동기까지의 양극단 전송 지연을 제한한다. 센서 S에서 샘플된 데이터가 출력 장치 A를 구동하는데 사용된다고 하면, 최대 허용 유효 시간("$M(S \mid A)$"로 표현)은 S에서 A로의 전송 지연이 $M(S \mid A)$보다 커서는 안된다는 것을 나타낸다. 이 제약은 샘플이 지정된 시간 이내에 반드시 사용되어야 한다는 것을 보장한다.

3) *최대 루프 처리 주기(MLPP: Maximum loop processing period).* 이 제약은 제어기가 피제어기(plant)의 상태를 파악하여 유효한 제어를 하기 위하여, 주어진 시간 간격당 제어 루프의 최소 수행 수를 제한한다. 제어 루프가 센서 $S_1,...,S_k$에서 샘플을 받아 출력 장치 $A_1,...,A_l$를 구동할 때, 최대 루프 처리 주기는

$$P(S_1,...,S_k \mid A_1,...,A_l)$$

이 된다.

위의 시간 제약에 추가적으로, 미리 주어지는 중요한 시간 제약이 있다.

- *최대 태스크 수행 지연.* 주어진 태스크 p_i의 최대 태스크 수행 지연은 태스크가 여러 수행 경로 중 가장 긴 경로를 따라 수행하였을 때 걸리는 시간으로 정의되고, 이것은 e_i로 표현된다. 제어 시스템에서의 모든 태스크의 수행 시간은 컴파일된 커널 합성 단계 이전에 미리 알려져 있어야만 한다. 태스크의 수행시간을 예측하는 것에 대한 방법들이 이미 [1, 9]에서 제시되어 있으므로 본 논문에서는 이 문제를 직접 다루지 않기로 한다.

태스크의 수행 시간이 때에 따라 변하기 때문에, 내장 디지털 제어기가 예측가능하고 안전하게 동작하도록 보장하기 위하여 태스크 수행시간의 최대 값을 이용해야만 한다. 그림 4의 제어기 예제에 부과된 모든 시간 제약과 속성들이 표 1에 요약되어 있다.

표 1. 제어기의 양극단 시간 제약.

Table 1. End-to-end timing constraints of the controller.

동기 제약	$Sync(P_1, V_1, P_2 \mid M_1) = 7,$ $Sync(P_2, V_2, P_1 \mid M_2) = 6$
최대 허용 유효 시간	$M(LS_1 \mid SW_1) = 150,$ $M(LS_1, P_1, V_1, P_2 \mid M_1) = 350$ $M(LS_2 \mid SW_2) = 150,$ $M(LS_2, P_2, V_2, P_1 \mid M_2) = 400$
최대 루프 처리 주기	$P(LS_1, C_1, P_1 \mid SW_1) = 200,$ $P(LS_1, C_1, P_1, V_1, P_2 \mid M_1) = 400$ $P(LS_2, C_2, P_2 \mid SW_2) = 150,$ $P(LS_2, C_2, P_2, V_2, P_1 \mid M_2) = 500$
최대 태스크 수행 지연	$e_1^1=5,\ e_2^1=5,\ e_3^1=5$ $e_4^1=45,\ e_5^1=20,\ e_6^1=8,\ e_7^1=5$ $e_1^2=5,\ e_2^2=5,\ e_3^2=5,$ $e_4^2=35,\ e_5^2=15,\ e_6^2=7,\ e_7^2=5$

2.4 태스크 시간 속성

태스크 그래프에서의 각 태스크는 고정된 주기로 수행을 반복한다. 그림 5는 주기적 태스크 p_i의 두 개의 수행을 그림으로 표현한 것이다. 주기적 태스크는 주기와 종료시한의 두 가지 시간 속성을 갖는다. 주기와 종료시한은 각각 T_i, D_i로 표현한다. 종료시한은 태스크의 수행이 가장 늦게 끝날 수 있는 시간을 말하며, 주기의 시작 시간에서부터 측정된다.

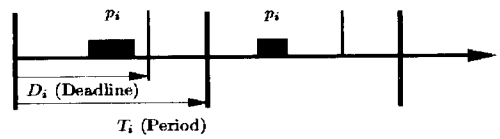


그림 5. 주기적 태스크 pi의 수행 행태와 시간 제약.

Fig. 5. Execution of periodic task pi and its timing attributes.

[6]에서처럼 본 논문에서는 태스크의 주기에 대한 중요한 가정을 한다. 태스크 p_i가 버퍼 b에 데이터를 쓰고, 태스크 p_j가 b에서 데이터를 읽는 상황을 가정해 보자. 이 때, p_i를 생산자 태스크, p_j를 소비자 태스크라고 부른다. 주어진 생산자/소비자 태스크의 쌍 (p_i, p_j)에 대하여, 주기 T_j가 주기 T_i의 정수 배인 관계가 성립하도록 할 필요가 있다. 이것을 주기의 조화성(harmonicity requirement)이라 하고, "$T_i \mid T_j$"로 표현한다. 이 조화 주기를 이용하면 태스크간의 통신 문제와 태스크 스케쥴링 문제를 간단하게 해결할 수 있다. 논문의 뒤에서 명확하게 설명하도록 한다.

2.5 문제의 기술

n개의 태스크 집합 $\{p_1, p_2, ..., p_n\}$을 포함하는 태스크 그래프와 입력과 출력 사이의 양극단 시간제약에 대하여, 본 논문의 문제는 다음과 같이 표현될 수 있다.

(P1) 유도된 제약이 원래의 양극단 시간제약을 만족하도록, 태스크의 시간 속성인 T_i, D_i에 대한 비선형 제약식을 유도한다.

(P2) 태스크의 스케쥴 가능성이 최대가 되도록 유도된 비선형 제약식을 푼다. 이 때 다음의 목적 함수를 이용한다.

$$\min\left(U = \sum_{i=1}^{n} \frac{e_i}{T_i}\right) \quad \text{and} \quad \max(D_i)\ \forall i,\ 1 \le i \le n$$

(이 목적 함수에 대하여 3절에서 설명한다.)

(P3) 위에서 계산된 T_i와 D_i를 이용하여 유용한 태스크 스케쥴을 얻는다.

(P4) 위의 태스크 스케쥴을 이용하여 수행 가능한 컴파일된 커널을 구성한다.

Ⅲ. 태스크 시간 속성 할당

본 절에서는 두 개의 부문제 **P1**과 **P2**에 대하여 설명한다. 주어진 양극단 시간 제약에서 제약의 종류에 따라 서로 다른 전략을 사용할 수 있다. 먼저 동기 제약에 대해서는 그래프 변환 방법을 이용한다. 이 경우 주어진 태스크 그래프를 동기 제약이 쉽게 만족될 수 있도록 새로운 것으로 변환한다. 다른 두 가지 종류의 제약에 대해서는 정수 계획법(integer programming)을 이용하여 태스크의 시간 속성에 대한 중간 제약을 유도하고 해를 구한다.

먼저 그래프 변환 규칙과 잘 정의된 제약 유도 규칙을 제시한다. 그리고 효율적인 방법으로 유도된 제약식에서 유용한 해를 찾는 알고리즘을 제시한다.

3.1 중간 제약의 유도

그래프 변환 규칙과 이전 절에서 정의된 세 가지 종류의 양극단 제약에 대하여 제약을 유도하는 규칙에 대하여 설명한다.

3.1.1 동기 제약

동기 제약은 구동기를 구동시키기 위해 사용되는 동기 입력들간의 시간차를 제한한다. 예를 들어, 그림 4의 제어 시스템에서 제어기는 M_2를 구동시키기 위하여 세 개의 입력 센서 P_2, V_2, P_1에서 $6ms$내에 읽어야 하며, 이 입력들은 동기되어야 한다. 전형적인 제어 시스템에서 이러한 동기 제약은 다른 종류의 제약들에 비하여 매우 엄격할 수 있기 때문에, 특별한 처리가 필요하다.

특별히 전체 제어 시스템이 동일한 비율로 수행되지 않는다면, 엄격한 동기 제약은 과샘플(over-sampled)된 시스템을 만들게 된다. 그림 4에서, 세 개의 독립된 태스크 p_3^2, p_2^2, p_3^1은 각각 P_2, V_2, P_1에서 입력을 받는다. 주기에 제약이 없기 때문에 주어진 동기 제약을 보장하기 위해 다음의 제약이 만족되도록 한다.

$$\max\left(|T_2^2+T_3^2|, |T_2^2+T_3^1|, |T_3^2+T_3^1|\right) \le 6ms$$

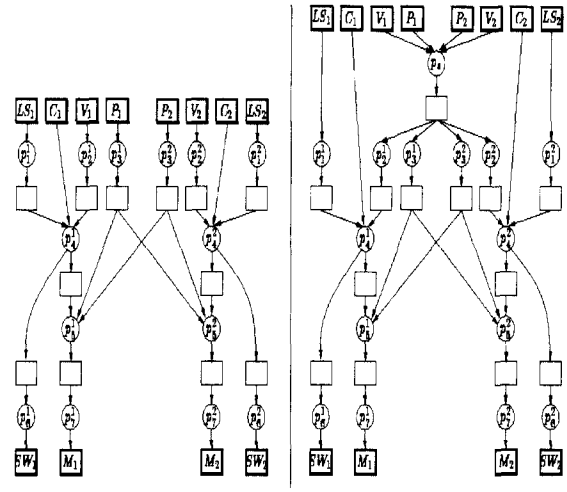
시스템이 과사용되지 않도록 하면서 엄격한 동기 제약을 보장하기 위하여 [6]에서 제시된 간단한 그래프 변환 방법을 사용할 수 있다. 그래프 변환 규칙은 다음과 같다.

동기 입력들을 읽기 위한 샘플링 태스크들 각각 대하여 새로운 태스크들 생성하여 원래의 샘플링 태스크들과 입력 센서들 사이에 놓고, 모든 동기 입력들을 한 번에 읽는다. 이것을 샘플링 서버라고 한다. 만약 두 샘플링 서버가 동일한 센서에서 입력을 받는다면, 두 개를 합하여 하나로 만든다.

예제: 위의 규칙을 그림 4의 태스크 그래프에 적용하면 다음과 같다.

1) P_1, V_1, P_2에 대하여 샘플링 서버 p_s^1을 생성한다.
2) P_2, V_2, P_1에 대하여 샘플링 서버 p_s^2를 생성한다.
3) 두 서버 p_s^1, p_s^2가 모두 P_1, P_2에서 입력을 받으므로, 하나의 샘플링 서버 p_s로 합한다.
4) p_s를 V_1, P_1, P_2, V_2와 태스크 $p_2^1, p_3^1, p_3^2, p_2^2$ 사이에 위치시킨다.

그림 6은 그림 4의 태스크 그래프와 위의 과정에 따라 변환된 태스크 그래프를 보이고 있다.



(a) (b)

그림 6. (a) 원래의 태스크 그래프, (b) 동기 제약을 이용하여 변환된 그래프.

Fig. 6. (a) The original task graph, (b) the new one transformed for synchronization constraints.

샘플링 서버가 비선점적(nonpreemptive)으로 수행을 하는 경우, 다음의 제약은 간단히 p_s를 통한 동기 제약을 보장한다.

$$end(p_s) - begin(p_s) \le 6ms$$

여기서 "$begin(p_s)$"와 "$end(p_s)$"는 p_s의 시작과 끝 시간을 나타낸다. p_s가 단지 센서로부터 데이터를 읽고 데이터 버퍼에 쓰는 일을 하기 때문에, 아주 작은 시간의 지연만이 발생한다. (위의 예에서 e_s는 $2ms$라고 가정한다).

3.1.2 최대 허용 유효 시간

최대 허용 유효 시간(MAVT)은 센서에서 출력까지의 전송 지연 시간을 제한한다. 이것은 센서에서 샘플된 입력이 출력장치를 구동하기 위하여 실제로 사용될 때까지 신선함(freshness)을 유지하기 위해서이다. 제어 시스템에 부과된 MAVT는 잘 정의된 규칙들에 의하여 태스크 시간 속성에 대한 두 가지 종류의 제약으로 변환될 수 있다. 규칙을 제시하기 전에 그림 6(b)의 태스크 그래프에서 LS_1과 SW_1사이에 태스크 p_1^1, p_4^1, p_6^1와 표 1의 MAVT "$M(LS_1|SW_1)$"을 살펴보자. 태스크들을 태스크 그래프로부터 분리시켜 그림 7(a)에 다시 표현하였다. 그러면 변환 규칙은 다음과 같다.

1) 2절에서와 같이, 생산자/소비자 쌍을 이루는 태스크들의 주기는 조화적이어야 한다.

$$T_1^1 \,|\, T_4^1, \quad T_4^1 \,|\, T_6^1$$

2) MAVT는 출력 태스크의 종료시한을 제한한다.

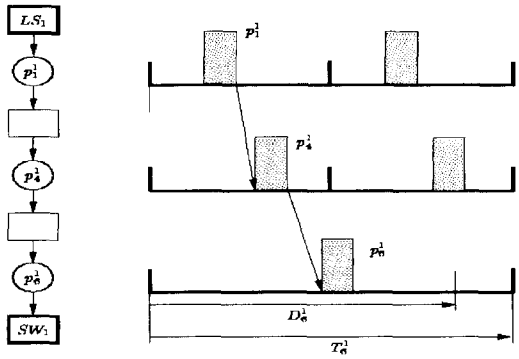
$$D_6^1 \le M(LS_1|SW_1) = 150$$

표 2는 위의 규칙을 이용하여 제어기 예제에서 유도한 부등식을 보이고 있다.

3.1.3 최대 루프 처리 주기

제어 루프의 최대 루프 처리 주기(MLPP)는 루프의 출력 태스크들의 주기와 종료시한의 상한을 부여한다. 이것은 모든 루프 처리 주기 내에 출력 장치를 구동시키기 위해서이다. 위의 제어기 예제에서 네 개의 제어 루프들이 각각 하

나의 출력 태스크를 갖고 있으며, 상한 요구 사항은 표 3의 제약으로 표현된다.



(a) (b)

그림 7. (a) 태스크 순차, (b) 실제로 수행되는 행태의 예.

Fig. 7. (a) The task sequence, (b) one of its execution scenarios.

표 2. MAVT에서 유도된 중간 제약.

Table 2. Intermediate constraints derived from MAVT.

주기 조화성	$T_s \mid T_2^1$, $T_s \mid T_3^1$, $T_s \mid T_2^2$, $T_s \mid T_3^2$, $T_1^1 \mid T_4^1$, $T_2^1 \mid T_4^1$, $T_1^2 \mid T_4^2$, $T_2^2 \mid T_4^2$, $T_4^1 \mid T_5^1$, $T_3^1 \mid T_5^1$, $T_4^2 \mid T_5^2$, $T_3^2 \mid T_5^2$, $T_3^1 \mid T_5^2$, $T_4^1 \mid T_6^1$, $T_5^1 \mid T_7^1$, $T_4^2 \mid T_6^2$, $T_5^2 \mid T_7^2$
MAVT	$D_6^1 \le 150, D_7^1 \le 350, D_6^2 \le 150, D_7^2 \le 400$

표 3. 최대 루프 처리 주기에서 유도한 중간 제약들.

Table 3. Intermediate constraints derived from maximum loop processing periods.

제어 루프 $(LS_1, C_1, P_1 \mid SW_1)$	$T_6^1 \le 200, \quad D_6^1 \le T_6^1$
제어 루프 $(LS_1, C_1, P_1, V_1, P_2 \mid M_1)$	$T_7^1 \le 400, \quad D_7^1 \le T_7^1$
제어 루프 $(LS_1, C_1, P_1 \mid SW_1)$	$T_6^2 \le 150, \quad D_6^2 \le T_6^2$
제어 루프 $(LS_2, C_2, P_2, V_2, P_1 \mid M_2)$	$T_7^2 \le 500, \quad D_7^2 \le T_7^2$

이들 조건과 함께 스케쥴 가능성을 위한 필요 조건들로 필요한 자명한 시간 제약들이 있다. 그림 7(a)에서 수행 가능한 상황 중 하나를 그림 7(b)로 표현한 것이다. 일련의 태스크들 p_1^1, p_4^1, p_6^1이 원하는 대로 수행되는 것을 보장하기 위하여, 이 태스크들이 D_6^1내에 수행을 끝마칠 수 있도록 충분한 시간이 있어야만 한다. 그렇지 않으면 시스템은 스케쥴 불가능하게 된다. 이것으로부터 다음의 제약을 얻을 수 있다.

$$e_1^1 + e_4^1 + e_6^1 \le D_6^1$$

비슷하게 두 개의 태스크 열(sequence) "p_1^1"과 "$p_1^1 \rightarrow p_4^1$" 들도 마지막 태스크의 주기 내에 수행을 끝마쳐야만 한다. 여기에서 다음의 제약을 얻을 수 있다.

$$e_1^1 \le T_1^1$$
$$e_1^1 + e_4^1 \le T_4^1$$

표 4는 제어기 예제에서 네 개의 제어 루프에서 유도된 이러한 제약을 요약한 것이다.

표 4. 스케쥴 가능성에 의한 중간 제약.

Table 4. Intermediate constraints due to schedulability.

제어 루프 $(LS_1, C_1, P_1 \mid SW_1)$	$e_s + e_1^1 + e_2^1 + e_4^1 + e_6^1 \le D_6^1$ $e_s + e_1^1 + e_2^1 + e_4^1 \le T_4^1$ $e_s + e_2^1 \le T_2^1$ $e_1^1 \le T_1^1$ $e_s \le T_s$
제어 루프 $(LS_1, C_1, P_1, V_1, P_2 \mid M_1)$	$e_s + e_1^1 + e_2^1 + e_4^1 + e_3^1 + e_5^1 + e_7 \le D_7^1$ $e_s + e_1^1 + e_2^1 + e_4^1 + e_3^1 + e_5^1 \le T_5^1$ $e_s + e_3^1 \le T_3^1$
제어 루프 $(LS_1, C_1, P_1 \mid SW_1)$	$e_s + e_1^2 + e_2^2 + e_4^2 + e_6^2 \le D_6^2$ $e_s + e_1^2 + e_2^2 + e_4^2 \le T_4^2$ $e_s + e_2^2 \le T_2^2$ $e_1^2 \le T_1^2$
제어 루프 $(LS_2, C_2, P_2, V_2, P_1 \mid M_2)$	$e_s + e_1^2 + e_2^2 + e_4^2 + e_3^1 + e_3^s + e_5^2 + e_7^2 \le D_7^2$ $e_s + e_1^2 + e_2^2 + e_4^2 + e_3^1 + e_3^2 + e_5^2 \le T_5^2$ $e_s + e_3^2 \le T_3^2$

지금까지 유도한 모든 부등식들은 원하는 태스크의 시간 속성을 얻기 위해 풀어야하는 제약 시스템을 형성한다. 이 부등식을 만족하는 해를 구하기 위하여 태스크 수행 시간을 이 부등식에 대입하여 상수로 표현함으로써 유도된 제약식들을 단순화시킬 수 있다. 이렇게 단순화된 전체 제약식이 표 5에 나타나있다.

표 5. 중간 제약 유도 후의 전체 제약식.

Table 5. The whole constraints after intermediate constraint derivation.

주기 조화성	$T_s \mid T_2^1$, $T_s \mid T_3^1$, $T_s \mid T_2^2$, $T_s \mid T_3^2$, $T_1^1 \mid T_4^1$, $T_2^1 \mid T_4^1$, $T_1^2 \mid T_4^2$, $T_2^2 \mid T_4^2$, $T_4^1 \mid T_5^1$, $T_3^1 \mid T_5^1$, $T_4^2 \mid T_5^2$, $T_3^2 \mid T_5^2$, $T_3^1 \mid T_5^2$, $T_4^1 \mid T_6^1$, $T_5^1 \mid T_7^1$, $T_4^2 \mid T_6^2$, $T_5^2 \mid T_7^2$
MAVT	$D_6^1 \le 150$ $D_7^1 \le 350$ $D_6^2 \le 150$ $D_7^2 \le 400$
MLPP	$T_6^1 \le 200, \quad D_6^1 \le T_6^1$ $T_7^1 \le 400, \quad D_7^1 \le T_7^1$ $T_6^2 \le 150, \quad D_6^2 \le T_6^2$ $T_7^2 \le 500, \quad D_7^2 \le T_7^2$
스케쥴 가능성	$65 \le D_6^1, \ 52 \le T_4^1, \ 7 \le T_2^1, \ 5 \le T_1^1, \ 2 \le T_s$ $87 \le D_7^1, \ 82 \le T_5^1, \ 7 \le T_3^1$ $54 \le D_6^2, \ 47 \le T_4^2, \ 7 \le T_2^2, \ 5 \le T_1^2$ $77 \le D_7^2, \ 72 \le T_5^2, \ 7 \le T_3^2$

3.2 제약식 풀기

중간 제을약 유도하고 나면 주기와 종료시한을 갖는 태스크 집합이 주어진다. 이 제약을 풀 때의 목표는 모든 제약들이 만족되고, 전체 프로세서 이용률 ($U = \sum_i e_i / T_i$)이 최소화되도록 태스크의 시간 속성들을 결정하는 것이다. 이런 목적 함수를 이용하는 이유는 현 단계에서 가능한 제한된 정보를 이용하여, 스케쥴 가능한 태스크 집합을 얻을 수 있

는 가능성을 최대화시킬 수 있기 때문이다 [10].

이 제약식들의 풀이 과정에는, 조화 주기 제약과 프로세서 이용률을 최소화하는 목적함수 때문에 비선형 정수 계획(nonlinear integer programming) 문제가 포함되게 된다. 이 비선형성 때문에 해를 찾기 위해 휴리스틱을 이용해야 한다. 여기서는 변수 분류를 통하여 이 문제를 해결하도록 한다. 특별히 변수들을 주기 변수와 종료시한 변수의 두 종류로 분류한다. 그 후, 각 변수에 대하여 서로 다른 휴리스틱을 적용하여 하나씩 변수 값을 계산한다. 주어진 제약식 C에서 변수 소거법 [4]를 통하여 다른 부류의 변수들을 제거할 수 있다. 결과적으로 주어진 제약식 C를 풀기 위해 다음과 같이 세 단계를 거친다.

단계 1 C에서 종료시한 변수들을 제거하여, 제약 시스템 C'을 만든다. (본 논문의 예에서는 표 5과 표 6이 각각 C와 C'을 나타낸다.)

단계 2 제약식 C'을 만족하도록 주기 T_i, $1 \le i \le n$에 대하여 프로세서 이용률 $u = \sum_i \frac{e_i}{T_i}$를 최소화한다.

단계 3 모든 주기값 T_i를 원래의 제약 시스템 C에 대입하여 C''를 얻는다. 제약 C''를 만족하도록 하면서 종료시한 D_i의 최대 값을 구한다. (표 7이 C''에 해당한다.)

이와 유사한 방법은 논문 [6]에서 제시된바 있으나, 본 논문에서는 제약 C'과 C''을 최적화하기 위해 새로운 탐색 알고리즘을 제시한다. [6]에서 제시된 알고리즘은 단순한 탐색에 의존하고, 너무 많은 중간 제약 해를 얻을 가능성이 있다. 이것은 탐색 과정을 지연시키고 결과적으로 실제 제어 응용들을 위해 부적합한 알고리즘이 될 수 있다. 하지만 본 논문의 알고리즘은 효율적인 "branch-and-bound" 휴리스틱을 이용하기 때문에 빨리 해를 찾을 수 있다.

3.2.1 주기 할당

유도된 제약식에서 종료시한 변수들을 제거하기 위해 부록 A에 설명된 푸리에 (Fourier) 변수 소거법을 사용한다. 표 6은 이 변수 소거법을 이용하여 표 5의 제약식에서 종료시한 변수들을 제거한 결과를 나타낸다. 이들 주기에 대한 제약들을 두 가지로 분류할 수 있다.

- *주기 조화성*: 이것은 모든 생산자/소비자 쌍 (p_i, p_j)에 대한 제약이며, $T_i | T_j$가 된다.
- *범위 제약*: 이것은 태스크들의 주기에 대한 상한과 하한을 표현한다.

표 6. 종료시한 변수 제거후의 중간 제약.
Table 6. The intermediate constraints after deadline variable elimination.

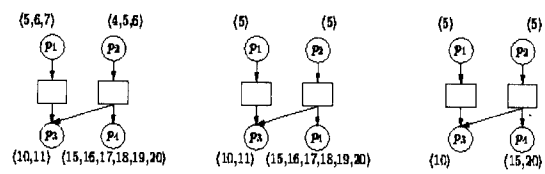
주기 조화성	$T_3 \mid T_2^1$, $T_3 \mid T_3^1$, $T_3 \mid T_2^2$, $T_3 \mid T_3^2$, $T_1^1 \mid T_4^1$, $T_2^1 \mid T_4^1$, $T_1^2 \mid T_4^2$, $T_2^2 \mid T_4^2$, $T_4^1 \mid T_5^1$, $T_3^1 \mid T_5^1$, $T_4^2 \mid T_5^2$, $T_3^2 \mid T_5^2$, $T_3^1 \mid T_5^2$, $T_4^1 \mid T_6^1$, $T_5^1 \mid T_7^1$, $T_4^2 \mid T_6^2$, $T_5^2 \mid T_7^2$
범위 제약	$5 \le T_1^1$, $5 \le T_1^2$ $7 \le T_2^1$, $7 \le T_2^2$ $7 \le T_3^1$, $7 \le T_3^2$ $52 \le T_4^1$, $47 \le T_4^2$ $82 \le T_5^1$, $72 \le T_5^2$ $65 \le T_6^1 \le 200$, $87 \le T_7^1 \le 400$ $54 \le T_6^2 \le 150$, $77 \le T_7^2 \le 500$

본 절에서는 적절한 크기의 태스크 그래프에 대하여 잘 동작하는 그래프에 기초한 간단한 방법을 제시한다. 이 방법의 주된 아이디어는 가능한한 문제의 크기를 줄자는 것이다. 이렇게 줄어든 문제를 "branch-and-bound" 알고리즘을 이용하여 푼다.

임의의 태스크 p_i에 대하여, I_i가 T_i에 주어진 범위를 만족하는 모든 변수의 집합이라고 하자. 본 알고리즘은 두 단계에 따라 주기 할당 문제를 해결한다. 먼저 주기의 조화성을 이용하여 각 I_i의 크기를 줄인다. 이를 위해, 주어진 태스크 그래프를 앞뒤로 여러 번 순회한다. 그리고 유용한(feasible) 해를 찾기 위해 branch-and-bound 알고리즘을 수행한다.

예제 : 네 개의 태스크 p_1, p_2, p_3, p_4로 이루어진 그림 8(a)의 간단한 태스크 그래프를 예로 들자. 노드 옆에 표기된 숫자는 각 태스크들의 I_i값을 의미한다. 후방 순회시(backward traversal) 알고리즘은 후속자(successor)가 없는 노드에서부터 시작한다. p_3에서부터 시작한다고 하자. 주기의 조화성 때문에, p_3에 선행하는 노드들은 주기 T_3의 약수를 주기로 가져야한다. 그래서 집합 I_3의 어떤 원소의 약수도 아닌 6과 7을 I_1에서 삭제한다. 마찬가지 이유로 I_2에서 4와 6을 제거한다. 다음에 노드 p_4를 방문한다. I_2의 모든 원소들이 I_4에 있는 몇몇 원소들을 나눌 수 있는 값이므로, 알고리즘은 더 이상 I_2를 줄일 수 없다. 그림 8(b)에 중간 결과를 표현하였다.

전방 순회 때는 후속하는 노드가 없는 노드에서부터 출발한다. 이 때, 후방 순회에서 줄인 중간 결과인 그림 8(B)를 이용한다. 노드 p_2에서 시작한다고 하자. 앞에서와 비슷하게, 주기의 조화성 때문에, p_2의 후속 노드들은 T_2의 배수가 되는 값들만을 주기로 취할 수 있다. 그래서 알고리즘은 I_3에서 11을 제거하고, I_4에서 16, 17, 18, 19를 제거한다. 그림 8(c)는 이 순방향 순환의 결과를 보이고 있으며, 더 이상 I_i의 원소의 개수를 줄일 수 없다. 그래서 알고리즘은 끝나게 된다.



(a) (b) (c)

그림 8. 해공간 줄이기, (a) 원래의 태스크 그래프, (b) 후방 순회후, (c) 전방 순회후.
Fig. 8. Reducing the solution space (a) The original task graph, (b) after a backward traversal; and (c) after a forward traversal.

이렇게 해 공간이 줄어들면, 알고리즘은 다음과 같이 모든 가능한 주기 값들을 나열한다.

	T_1	T_2	T_3	T_4
해 1	5	5	10	15
해 2	5	5	10	20

알고리즘은 이들 해중 가장 작은 프로세서 이용률을 갖는 해를 선택한다. 경험적으로 이 탐색 알고리즘은 "미리 설정된" 값과 계산된 프로세서 이용률을 비교하여, 상당량의 해공간을 줄일 수 있다. 이후에 여기에 대하여 언급하도록 한다.

알고리즘의 전체적인 구조는 그림 10에 도시되어 있다. 후방 화살표가 의미하듯이 해공간을 줄이는 것은 반복적인 작업이다. 이제 알고리즘의 각 부분에 대하여 자세히 설명하도록 한다. 첫째, 새로운 태스크 그래프를 구성하여, 각 제약 $T_i|T_j$에 대하여 간선 $\tau_i \rightarrow \tau_j$을 만든다. 이 그래프는 태스크 그래프에서 버퍼 노드를 제거하고, 각 생산자/소비자 쌍에 대하여 간선을 만들어줌으로써 간단히 얻을 수 있다. 이것을 조화 그래프(harmonicity graph)라고 한다. 그림 9는 그림 6(b)의 태스크 그래프에서 변환된 조화 그래프이다. 해 공간 축소 작업은 이 조화 그래프에 전방/후방 순회를 반복적으로 적용함으로써 얻을 수 있다. 알고리즘이 후방 순회 과정에서 노드 p_i를 방문하고 있다고 가정하자. p_i 바로 이전의 모든 노드 p_j에 대하여, I_j의 원소 a가 I_i의 어떤 원소의 약수도 아니면 I_j에서 a를 제거한다.

비슷하게, 전방 탐색과정에서 노드 p_i를 방문하고 있다고 하자. 노드 p_i 바로 뒤에 위치하는 모든 노드 p_j에 대하여, I_j의 원소 a가 I_i의 어떤 원소의 배수도 아니면, a를 I_j에서 제거한다. 이 과정을 모든 I_i에서 원소를 제거할 수 없을 때까지, 전/후방 순회를 반복한다. 여기서 주의할 것은 알고리즘은 후방 순회를 전방 순회보다 먼저 처리한다는 것이다. 이것은 오직 출력 태스크들만이 주기의 상한값을 갖고 있기 때문이다. 그래서 전방 순회를 먼저 하게 되면 알고리즘은 무한히 반복을 하게 되어, 수행을 끝마칠 수 없다.

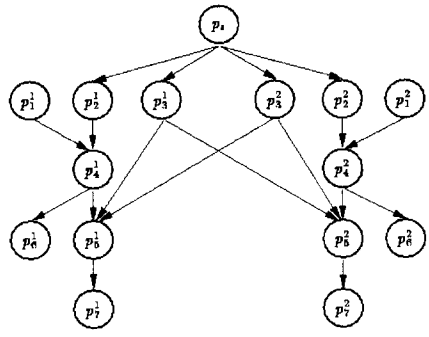


그림 9. 조화 그래프.
Fig. 9. The harmonicity graph.

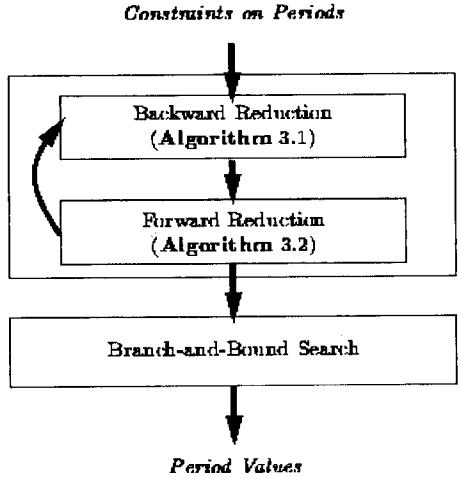


그림 10. 주기 할당 알고리즘의 구성.
Fig. 10. The structure of the period assignment algorithm.

후방 순회 알고리즘은 다음과 같다.

알고리즘 3.1 각 노드에서 후방으로 조화 그래프를 순회하면서 해공간을 줄여나간다.

단계 1 : 노드 p_i에 후속하는 모든 노드들이 이미 방문되거나 후속하는 노드들이 없으면, 노드 p_i를 방문한다.

단계 2 : p_i 바로 이전에 위치하는 모든 노드 p_j들에 대하여 다음을 만족하는 새로운 I_j를 결정한다.

$$I_j := I_j \cap \{a \in I_j \mid b \in I_i \ \textbf{and} \ a|b\}$$

단계 3 : 조화 그래프에 있는 모든 노드들을 방문했으면 순회를 멈춘다. 그렇지 않으면 **단계 1**로 간다.

전방 순회 알고리즘은 다음과 같다.

알고리즘 3.2 *각 노드에서 전방으로 조화 그래프를 순회하면서 해공간을 줄여나간다.*

단계 1 : 노드 p_i에 선행하는 모든 노드들이 이미 방문되거나 선행하는 노드들이 없으면, 노드 p_i를 방문한다.

단계 2 : p_i 바로 다음에 위치하는 모든 노드 p_j에 대하여 다음을 만족하는 새로운 I_j를 결정한다.

$$I_j := I_j \cap \{a \in I_j \mid b \in I_i \ \textbf{and} \ b|a\}$$

단계 3 : 조화 그래프에 있는 모든 노드들을 방문했으면 순회를 멈춘다. 그렇지 않으면 **단계 1**로 간다.

해공간이 줄어들면, 가능한 주기들의 집합을 결정하기 위해 "branch-and-bound" 탐색 알고리즘을 적용한다. 하지만 이 알고리즘을 이용하여 가장 좋은 해(즉, 프로세서 이용률을 가장 낮게 만드는 해)를 찾으려고 하지는 않는다. 대신 가능하면 빨리 "수용가능한(acceptable)" 해를 얻는 것이 목적이다. 이를 위해 알고리즘에 "cutoff" 이용률을 명시할 수 있도록 한다. 이것을 "u_{cutoff}"라고 한다. 이 u_{cutoff}를 이용하여 본 탐색 알고리즘은 주어진 제약을 만족하면서 u_{cutoff}보다 크지 않은 이용률을 갖는 첫 번째 해를 찾고자 한다. 만일 탐색 알고리즘이 주어진 시간 안에 해를 찾지 못하면, 사용자는 좀 더 큰 u_{cutoff}를 지정하여 알고리즘을 다시 수행할 수 있다.

예제 : 그림 9의 조화 그래프 예를 이용하여 탐색 알고리즘을 수행시켜보자. 처음 수행시 u_{cutoff} 값으로 1.0을 사용하였다. 이 때, 알고리즘은 프로세서 이용률 0.91의 수용가능한 해를 찾아내었다. 두 번째로 0.9를 u_{cutoff}로 사용하였을 때는 프로세서 이용률이 0.89로 좀 더 좋은 결과를 얻었다. 다음은 이 때의 결과이다.

T_s	T_1^1	T_2^1	T_3^1	T_4^1	T_5^1	T_6^1	T_7^1	T_1^2	T_2^2	T_3^2	T_4^2	T_5^2	T_6^2	T_7^2
65	195	195	390	195	390	195	390	130	130	390	130	390	130	390

u_{cutoff} = 0.89를 이용하여 알고리즘을 수행시키면, 수용가능한 해를 찾을 수 없게 된다.

3.2.2 종료시한 변수 할당

주기가 결정되면, 결정된 주기 값들을 표 6의 제약식들에 넣어 표 8에서와 같이 새로운 선형 제약식의 집합을 만들 수 있다. 이 단계에서의 목적 함수 역시 태스크들의 스케쥴 가능성을 최대화하는 것이다.

태스크 p_i에 대하여, 주기 T_i에서 시간 간격 $[0, D_i]$는 각 태스크가 수행될 때의 수행 윈도우가 된다. 이 윈도우의 크기가 클수록 p_i가 스케쥴될 수 있는 가능성이 증가하게 된다. 그래서 목적함수는 다음과 같다.

표 7. 종료시한에 대한 선형 제약.
Table 7. The linear constraints on deadlines.

종료시한에 대한 범위 제약	$65 \le D_6^1 \le 150$ $85 \le D_7^1 \le 350$ $54 \le D_6^2 \le 130$ $77 \le D_7^2 \le 390$

- D_1의 값을 최대화한다.

n개의 종료시한 변수들을 갖는 선형 제약들의 집합에 대하여, $n-1$개의 변수를 제거하기 위하여 변수 소거법을 사용한다. 소거된 변수 집합 $\{D_2, D_3, \ldots, D_n\}$이 있다고 하자. D_1에 대한 목적함수를 최대로 만들기 위하여 간단하게 D_1의 가장 큰 값을 선택한다. D_1이 결정되면 값을 다시 원래의 식에 대입하고, $n-1$개의 변수를 갖는 식을 만든다. 이러한 과정을 변수가 남지 않을 때까지 반복한다. 이렇게 얻어진 종료시한의 결과는 다음과 같다.

D_6^1	D_7^1	D_6^2	D_7^2
150	350	130	390

3.3 컴파일된 커널 합성

주기 할당기(period assigner)를 통해 태스크의 시간 정보들을 유도한 뒤에는, 커널 합성기(kernel synthesizer)를 이용하여 응용 프로그램 코드와 커널 루틴들을 하나로 묶어 컴파일된 커널을 생성하여야 한다. 커널 합성기는 그림 2에서와 같이 변환된 태스크 그래프와 태스크의 시간 정보들을 입력으로 받는다. 컴파일된 커널에서는 이들 정보를 이용하여 유한 메모리를 갖는 통신 데이터 버퍼를 구현하고, 태스크 스케쥴링 기법을 구현한다. 이 두 가지를 구현하기 위하여 시스템에 존재하는 태스크들의 반복적 수행 특성을 고려할 수 있다. 예를 들어, 태스크 간의 통신 버퍼는 환형 큐(circular queue)로 구현을 할 수 있으며 [6], 태스크 스케쥴링 방식으로는 순환 커널(cyclic executive)이라고 알려진 기법을 [5, 11] 이용할 수 있다.

Ⅳ. 결론

본 논문에서는 다중 루프 제어 시스템에서 컴파일된 커널을 만들어내는 반자동화된 방법을 제시하였다. 다중 루프 제어 시스템은 태스크 그래프와 양극단 시간 제약으로 표현되며, 태스크 그래프는 제어기의 소프트웨어 구조를 표현한다. 이것은 제어 시스템 개요도(schematic diagram)에서 쉽게 얻을 수 있다. 또한 양극단 시간 제약은 외부 입력과 출력 사이에 존재하는 시간 제약을 표현한다.

전체적인 설계 과정은 다음의 순서를 따른다. 먼저 태스크의 시간 속성에 대한 비선형 제약식을 유도한다. 두번째로 두 가지 서로 다른 부류의 변수에 대하여 특화된 휴리스틱을 이용한 비선형 최적화 기법을 이용하여 제약식을 푼다. 세번째로 유도된 태스크 시간 정보에 따라 태스크들을 스케쥴링한다. 마지막으로 컴파일된 커널을 생성한다.

본 논문에서는 위의 네 가지 단계에서의 해법을 제시하였으며, 전체 과정을 컴퓨터 도구로 구현할 수 있다. 도구의 많은 부분이 자동화될 수 있으며 제어 기술자들이 실제 시스템이나 시범 (prototype) 시스템을 빠른 시간 내에 개발할 수 있을 것으로 기대한다. 현재 도구의 중심이 되는 부분인 비선형 최적기를 구현하였으며 계속해서 커널 합성기를 위한 GUI(Graphical User Interface)와 커널 라이브러리를 구현하고 있다.

본 연구를 확장하기 위하여 몇 가지 연구 방향이 있다. 현재는 단일 프로세서로 구성된 제어 시스템을 가정하고 있다. 이것을 분산 환경에서 사용할 수 있도록 확장을 계획하고 있다. 분산 환경에서는 많은 자원들을 동시에 고려해야 하기 때문에 아주 복잡한 문제를 야기한다. 또한, 설계 도구가 처리할 수 있는 양극단 시간 제약의 종류를 다양화시키려 한다. 이것은 현재 사용하고 있는 양극단 시간 제약들만으로 제어 시스템에서의 모든 시간 제약을 표현하기 힘들기 때문이다.

마지막으로 본 논문에서 기술된 기술들을 CNC 제어기 개발에 적용하여 제안된 방법론의 실용화 가능성을 검증하고 있다.

참고문헌

[1] R. Arnold, F. Mueller, and D. Whalley. "Bounding worst-case instruction cache performance," *Proceedings of IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 172-181, December, 1994.

[2] G. Brassard and P. Bratley, *Algorithmics: Theory and Practice*, chapter 6. Prentice-Hall, Inc., 1988

[3] S. Cavalieri, A. Stefano, and O. Mirabell. "Pre-run-time scheduling to reduce schedule length in the FieldBus environment," *IEEE Trans. Software Engineering*, vol. 21, no. 11, 865-880, November, 1995.

[4] G. Dantzig and B. Eaves. "Fourier-Motzkin Elimination and its Dual," *Journal of Combinatorial Theory* (A), vol. 14, pp. 288-297, 1973.

[5] M. Factor, D. Gelernter, C. Kolb, P. Miller, and D. Sittig, "Real-Time data fusion in the intensive care unit," *IEEE Computer*, pp. 45-53, November, 1991.

[6] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing end-to-end timing constraints by calibrating intermediate processes," *Proceedings of IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 192-203, December, 1994.

[7] C. Han and K. Kin, "Scheduling distance-constrained real-time tasks," *Proceedings of IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 300-308, December, 1992.

[8] Y. Koren, *Computer Control of Manufacturing Systems*, McGraw-Hill, Inc., 1983.

[9] S. Lim, Y. Bae, G. Jang, B. Rhee, S. Min, C. Park, H. Shin, K. Park, and C. Kim, "An accurate worst case timing analysis for RISC processors," *Proceedings of Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 97-108, December, 1994.

[10] C. Liu and J. Layland, "Scheduling algorithm for multiprogramming in a hard real-time environment," *Journal of ACM*, vol. 20, no. 1, pp. 46-61, January, 1973.

[11] C. Locke, "Software architecture for hard real-time applications: Cyclic executives vs. fixed priority

executives," *The Journal of Real-Time Systems*, vol. 4, no. 1, pp. 37-53, March, 1992.

[12] C. Shaffer and G. Herb, "A real-time robot arm collision avoidance system," *IEEE Trans. Robotics and Automation*, vol. 8, no. 2, pp. 149-160, April, 1992.

[13] J. Xu and D. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations," *IEEE Trans. Software Engineering*, vol. 16, no. 3, pp. 360-369, March, 1990.

부록

푸리에 변수 소거법(Fourier Variable Elimation)

푸리에 변수 소거법[4]은 선형 제약을 갖는 시스템에서 동작하며, n차원의 polytope(제약식으로 기술된다)에서 낮은 차원의 영역으로 투영(projection)시키는 것으로 볼 수 있다. 이것을 예를 통해 살펴보기로 하자. 본 논문의 예제의 태스크 그래프에서 변수 D_7^1에 대한 제약을 생각해보면 다음과 같다.

$$87 \le D_7^1, \quad D_7^1 \le T_7^1$$

만약 다음의 식이 만족된다면 해를 구할 수 있다.

$$87 \le T_7^1$$

이 새로운 제약으로 단순히 각 상한에 대하여 각각의 하한값으로 대치함으로써 D_7^1을 소거할 수 있다.

홍 성 수
제어 · 자동화 · 시스템공학 논문지 제 2권 제 3호 참조.

박 홍 성
제어 · 자동화 · 시스템공학 논문지 제 2권 제 3호 참조.

최 종 호
제어 · 자동화 · 시스템공학 논문지 제 2권 제 4호 참조.