정적 분석을 이용하여 시간 제약 조건을 해결한 실시간 언어의 설계 및 구현

(Design and Implementation of Real-Time Language Satisfying Timing Constraints using the Results of Static Analysis)

이 준 동 † 백 정 현 †† 원 유 헌 †††
(JoonDong Lee) (JungHyun Baek) (YooHun Won)

요 약 실시간 프로그램은 다양한 응용분야에 중요하게 이용되고 있는데, 기존의 일반 언어는 시간적인 개념을 고려하여 설계하지 않았으므로 실시간 응용에 부적합하며, 이를 해결하기 위한 실시간 언어는 시간 개념을 표현하기 위하여 많은 문법이 추가되어 기존 프로그래머에 익숙치 못한 결점이 있다.

본 연구에서는 기존의 C언어에 익숙한 프로그래머들이 저항감 없이 실시간 프로그래밍을 할 수 있는 언어를 설계하고 구현한다. 이 구현에서는 시간 트리를 이용함으로써 원시 언어와 목적 언어의 연결이 가능하며, 정적 분석을 이용한 결과를 코드 생성에 이용함으로써 외부적인 타이머 없이 시간적인 사건의 처리가 가능하다.

Abstract The real-time programs have played an important role in many application fields. But both the common language and the existing real-time programs have their limitations for the real-time applications. The common languages are not suitable to the real-time applications because they don't have time concepts. Also, programmers are not accustomed to the existing real-time programs because they have many additive syntaxes for the time concepts.

In this study, a new language is designed and implemented for the skillful C language programmers. In this implementation, the source languages are linked to the object languages using the timing tree and it is possible to process the successive occurrences without an external timer using the results of static analysis to generate the codes.

1. 서 론

실시간 시스템은 시스템의 동작이 논리적으로 맞아야 함은 물론, 그 결과가 산출된 시간도 시스템 성공의 중요한 요인이 된다. 실시간 시스템은 내장 시스템(embedded systems)과 같은 의미로 사용되기도 하는데, 내장 시스템이란 컴퓨터 시스템이 커다란 시스템의 일부로서 사용되는 시스템을 의미한다.

내장 시스템은 제어를 담당하는 부 시스템과 제어를 받는 개체들로 나눌 수 있다. 부 시스템은 환경으로부터 입력을 받아 제어 프로그램을 실행시키고, 환경의 상태를 변화시키기 위하여 개체들에게 명령을 보낸다[2].

이와 같이 실시간 시스템이 올바르게 동작하기 위해서는 제어를 담당하는 부 시스템이 논리적으로 맞아야 함은 물론, 시간적인 오류도 없어야 한다.

현재 많이 사용중인 대부분의 언어가 시간 분석을 고려하여 설계하지 않았기 때문에 이들 언어로 실시간 제어를 구현할 경우에는 많은 시행착오를 거쳐 수동적인 형태로 프로그램을 하여야 한다.

실시간 연구 분야에서는 이와 같은 문제점을 해결하기 위하여 Real-Time Euclid, DPS, MPL, RTC, RTC++, FLEX, TCEL, RTL/MCS와 같은 많은 언어

· 이 연구는 학술진흥재단의 자유공모과제 (과제번호 : 972030106)의 지원으로 이루어진 것임.

† 종신회원 : 원주대학 전산정보처리 교수
jlcc@sky.wonju.ac.kr
†† 정 회 원 : 우송공업대학 컴퓨터정보계열 교수
jhbaek@woosongtech.ac.kr
††† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
won@cs.hongik.ac.kr

논문접수 : 2000년 2월 15일
심사완료 : 2000년 10월 2일

가 연구되었지만 문법이 생소하여 실제 산업계에서는 많이 사용되고 있지 않다[3,4,5,6,7,8].

본 논문에서는 현재 가장 많이 사용하는 C언어에 시간 제약 사항을 기술하는 방법을 고안한 언어를 설계하고 이를 구현하였다.

이 방법을 사용하면 기존의 C 프로그래머가 새로운 언어를 숙지하는 번거로움 없이 간단하게 시간 제약 사항을 기술할 수 있으며, 일반 컴파일러와도 호환이 가능한 프로그램의 작성이 가능하다.

또한, 언어의 구현에는 시간 트리와 정적 분석을 이용하였다. 시간 트리를 이용함으로써 원시 언어와 목적 언어 사이의 매핑이 가능하며, 정적 분석 방법을 코드 생성에 이용함으로써 시간 제약 사항에 위배되는 문법을 검출할 수 있고, 외부의 타이머 없이 시간에 따른 사건의 처리가 가능하다.

2. 관련 연구

실시간 프로그래밍 언어를 사용하여 실시간 제어를 구현하는 방법 중에 하나는 시간 제약 조건을 표현할 수 있도록 일반 목적의 언어를 확장하는 것이다. 이 방법은 언어의 기본적인 형태는 변화시키지 않고 시간과 연관된 구성자만을 추가한다[2].

이와 같이 추가된 구성자는 크게 두 가지 종류로 나누어 생각할 수 있는데, 한 종류는 프로그램의 실행에 영향을 미치는 시간 제한 구조이고, 다른 종류는 실행에 영향을 미치는 것은 아니지만, 프로그램 실행에 정보를 제공할 수 있는 주석이다.

본 관련 연구에서는 기존의 실시간 언어에서 나타난 시간 제한 구조와 일반 언어의 실행 시간을 분석하기 위한 정적 분석 방법에 관하여 살펴본다.

2.1 실시간 언어의 시간 제한 구조

실시간 응용의 범위가 늘어나자 시간제한 표현의 필요성에 자극 받아 많은 실시간 언어들이 발표되었다. 이러한 언어들은 언어의 구성자안에서 시간 제한을 기술하는 것을 허용하였다.

표 1 Real-Time Euclid의 반복문

```
loop noLongerThan compileTimeExpn: Time- outReason
   [invariant booleanExpn]
   declarationandstatement
end loop

for [decreasing] [id]:compileTimeExpn..com- pileTimeExpn
   declarationandstatement
   [invariant booleanExpn]
end for
```

Real-Time Euclid는 모든 언어 구성자들이 시간과 공간면에서 제한되어진 형태를 가졌으며, 동적 자료 구조와 재귀적 호출은 허용되지 않았다. 병행 프로그래밍은 프로세스를 통하여 이루어졌다[4].

Real-Time Euclid에서 사용된 반복 구조는 표 1과 같은 형태를 가진다.

FLEX 시스템[5]은 유연성 있는 실시간 시스템 설계를 위한 프로그래밍 언어, 컴파일러, 분석 도구를 포함한다. FLEX 언어는 C++언어를 기초로 하여 제한 블록(Constraints blocks)과 성능 다형성, 시간 분석을 위한 주석을 지원한다. FLEX시스템에서 지원하는 제한 블록은 표 2와 같은 형태를 가진다.

표 2 FLEX언어의 제한구조

```
[lable:] (constraints: constraints:...)
[->{exception-statements)]
{statement...}
```

TCEL(Time-Constrained Evevt Language) [3]은 시간 제한 구조를 갖는 사건 기반의 실시간 언어로써 사건 사이의 시간 제약 조건을 컴파일러가 자동적으로 처리해준다. TCEL에서의 시간 제약 조건은 표 3과 같이 do문과 every문으로 표현된다.

표 3 TCEL언어의 제한구조

```
do
   <reference block>
[start after tmin][start before tmax1][finish within tmax2]
   <constraint block>
every p [while <condition>]
[start after tmin][start before tmax1][finish within tmax2]
   <constraint block>
```

CHaRTS[2]는 C언어를 기반으로 설계되었으며, 시간 블록과 반복 실행 블록, 시간 변수, 시간 수식과 같은 구문구조가 도입되었다. 그중 시간 블록은 사용자 정의 변수이거나 문장들의 집합으로 기존 실시간 시스템에서 타스크에 해당하는 역할을 한다.

이들 언어들은 모두 시간 제한을 기술하기 위하여 수준 높은 구조들을 사용하였다. 이 들 언어들의 주요 개념은 프로그래밍 문장들의 실행 순서에 시간적인 영역(timing scope)이 존재하며, 시간제한은 시간적 영역의 시작과 끝을 기술하는 것이다. 또한, 시간적 영역의 내포는 허용하지만, 겹침은 허용하지 않는다.

2.2 정적 분석

정적 분석은 프로그램의 실행시간 예측 기법이다. 프로그램의 실행시간은 시간 제한 문장들의 타당성과 스케줄 가능성 분석에 이용하므로 아주 중요하다. 프로그램의 실행시간은 최적 실행시간(Best-Case Execution time)과 최악 실행시간(Worst-Case Exe- cution time)으로 나눌 수 있으며, 이를 그림으로 표현하면 그림 1과 같다.

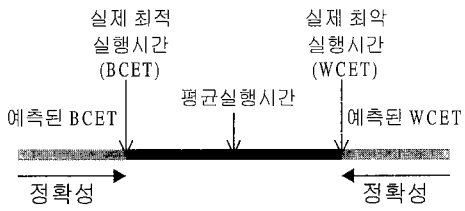


그림 1 측정된 실행시간의 가변성

프로그램의 실행 시간은 측정할 때마다 일정한 값이 아니며, 가장 길게 측정된 실행 시간도 실제 최악 실행시간을 만족하는지 알 수 없기 때문에 측정된 실행시간보다는 정적 분석에 의한 실행시간 예측 기법을 사용한다. 정적 분석시 고려 사항은 다음과 같다.

2.2.1 구조에 종속적인 부분

이 부분은 컴퓨터의 구조와 밀접한 관계를 가지는 부분으로 주로 저급 언어 단계의 분석(low level analysis)으로 알려져 있다. 이에는 중앙처리 장치의 주기, 메모리 접근시간, 자원의 사용 방법, 캐쉬와 파이프라인 등이 속한다.

특히, 캐쉬와 파이프라인의 사용은 최악실행시간을 과대 평가하는 중요한 요인으로 90년 대 이후에 많은 연구가 있었다. 캐쉬와 파이프라인이 최악실행시간에 미치는 영향과 이들의 과대 평가를 해결하기 위한 방법은 다음의 논문들에서 연구되어졌다[9,10,11,12].

2.2.2 구조에 독립적인 부분

구조에 독립적인 부분은 언어의 제약 사항과 프로그램이 실행되는 경로에 연관된 부분으로 고급 언어 단계의 분석(high level analysis)으로 알려져 있다.

기존의 언어에 대하여 정적 분석을 행하기 위해서는 GOTO 문의 사용을 제한하고, 프로그램 단위는 하나의 입구와 출구만을 소유해야 한다. 또한, 재귀적 호출의 사용을 제한하고, 모든 루프는 한계를 명시해야 한다.

현재 정적 분석의 연구 방향의 하나는 이러한 제약 조건들의 완화에 있으나 아직 실용화되지는 못하였고, 주로 주석을 이용하여 프로그래머의 도움으로 이러한 문제를 해결한다.

프로그램의 실행 경로는 컴파일러의 고전적인 문제인 자료 흐름분석과 제어 흐름 분석 등과 밀접한 관계를 가지는데 구문의 의미를 고려한 방법과 그렇지 않은 방법으로 나눌 수 있다.

제어의 흐름만을 고려한 대표적인 방법으로 ILP(Integer Linear Programming) [13]가 있으며, 구문의 의미를 고려한 방법으로 기호 실행(symbolic execution) 방법[14]과 요약 해석(Abstract Interpretation) 방법[15,16]이 있다.

특히, 요약 해석(abstract interpretation)의 이용은 아직은 현실화 되어있지는 않지만, 실시간 언어의 제약 사항을 줄여, 실시간 응용의 범위를 확대할 것으로 기대된다.

이 밖에 실시간 언어의 사용에서 고려 사항 중 하나가 사용자 인터페이스 문제이다. 사용자는 고급 언어에서 시간 제약 사항을 표현하기를 원하고 실제 실행시간은 저급 언어에서 다루어지므로 이들 사이의 매핑이 중요한 요인이 된다[17][18].

3. 실시간 언어

3.1 시간 제한 구문

2장의 관련 연구에서 이미 기존의 많은 실시간 제어 언어들을 살펴보았다. 이러한 언어들이 널리 사용되고 있지 않은 이유는 아직 실시간 응용들의 요구사항을 모두 충족시키지 못하는 것도 한 요인이지만, 기존의 언어 사용자들이 새로운 언어를 배워서 이용해야 한다는 문제점도 있다. 한가지 해결책은 기존의 언어에 약간의 수정으로 실시간 응용을 충족시키는 것이다. 기존의 C언어에 약간의 시간 제한 구문을 추가하여 Timing-C언어를 정의한다.

Timing-C의 설계 목적은 C프로그램의 친숙성을 유지하며, 최악실행시간을 분석 가능하게 하는 것이다. 기존 C의 사용자들이 쉽게 이용하며, 최악 실행시간 계산이 가능하게 하려면, 가능하면 C의 구문의 변화 없이 제약조건을 첨가하는 것이 유리하다.

Timing-C는 C의 부분집합으로 GOTO 문이 없고, 재귀적 호출과 동적 기억장소 할당이 금지된다. 또한, 프로그램 단위는 하나의 입구와 출구만을 가지며, 반복문은 반복횟수가 제한된다. 반복문을 제한하는 방법은 언어의 수정을 요구한다. Timing-C에서는 변화를 최소화하기 위하여, 프로그램의 각 영역의 시작에서 주석과

비슷한 형태로 반복횟수를 명시할 수 있게 하였다. 반복횟수를 제공하는 형태는 아래와 같다.

- 반복문의 최대 반복횟수 →

/** Count(Max_Count) **/

※ Max_Count : 최대 반복 횟수

반복횟수가 사용된 예는 그림 2와 같다.

```
1)  main()
2)  {
3)    int  i, t_count, e_count, sum;
4)    while(i <100)  { /** count(100) **/
5)    if(i % 2) {
6)       t_count = t_count+1;
7)       sum = sum+i;
8)     }
9)    else
10)      e_count = e_count+1;
11)   i = i+1;
12)  }
13)  }
```

그림 1 예 제

4)번 줄은 while 문장의 최대 반복 수가 100번임을 의미한다. 그림 2의 예제에서와 같이 제약조건은 1줄로 나타나고, 이 때 이 제약조건의 범위는 while 문장의 범위를 의미한다. 즉, 4)~12)번 줄까지의 영역을 의미한다. 이와 같이 같은 Count()명령이라도 나타나는 위치에 따라 다른 영역 범위를 갖게 되며, 새로운 영역을 만들고 싶은 경우에는 {}(중괄호)를 이용하여 간단하게 생성할 수 있다.

Time() 명령도 영역 범위는 Count()와 같으나 의미는 다르다. Count()명령은 실제 실행과는 관련이 없이 최악실행시간에만 영향을 주지만, Time()명령은 실제 코드 생성에도 영향을 주며, 정확한 시간에 사건의 발생을 가능하게 한다.

- 시간제한 구문 →

/** Time(Max_Time) **/

※ Max_Time : 최대 실행 시간

Max_Time에는 시간의 단위를 명시하여 주어야 하는데 본 논문에서는 사이클의 수를 의미한다. 이는 8051프로세서가 12MHz의 클록을 사용할 경우 각 인스트럭션의 실행시간이 머신 사이클에 의하여 1 또는 2 μs로 한정되므로 시간과 같은 의미로 사용 가능하다.

3.2 실시간 언어의 구현

실시간 언어는 SUN Ultra-5에서 GNUC를 사용하여 개발하였으며, 목적코드로써 8051마이크로 프로세서의 코드를 생성한다. 목적 프로세서로써 8051 프로세서를 선택한 이유는 8051프로세서가 현재 내장용 프로세서로써 전자 자동 제어분야에서 많이 사용하고 있을 뿐만 아니라, 캐쉬와 파이프라인을 사용하지 않으므로, 구조에 독립적인 분석이 가능하기 때문이다.

실시간 컴파일러는 그림 3과 같이 구성된다.

그림 3에서 보듯이 실행시간 분석기와 연관된 부분을 제거하면 일반적인 컴파일러의 구조와 같다. 주석 형태로 제공된 시간 제한 구문은 구문 트리에 반영되어진다. 구문 트리에는 원시코드에서의 라인 번호와 시간 제한 구문들, 그리고 나중에 실행시간 분석기의 결과를 표시한다. 구문 트리가 시간에 관한 정보를 가지고 있게 되면 이를 시간 트리라고 부른다. 본 논문에서는 구문 트리라는 용어와 시간 트리라는 용어를 같은 것으로 간주한다.

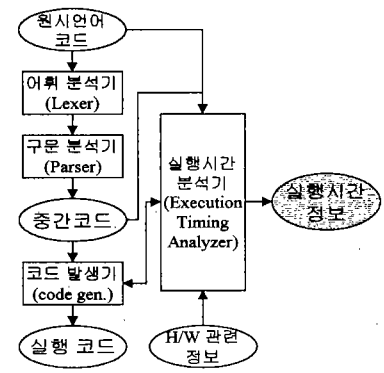


그림 3 컴파일러의 구조

실행시간 분석기는 컴파일러가 생성하는 여러 가지 정보를 이용하여 최악실행시간을 계산한다. 실행시간 분석기가 필요로 하는 정보에는 시간 트리, 제어 흐름 그래프, 어셈블리 명령어, 그리고 사용자의 제약 조건 등이 있다. H/W 관련 정보로는 프로세서 의존적인 정보가 제공된다. 본 Timing-C의 구현에는 8051프로세서를 이용하였으므로 프로세서 의존적인 정보로는 매뉴얼 상의 어셈블리 명령어의 실행시간만이 테이블로 제공된다.

4. 실행시간 분석기

실행시간 분석기는 프로그램 일부의 실행 시간의 예측과 프로그램의 시간적인 연산이 올바르게 동작하도록 하는 일을 담당한다. 이를 위하여 원시 언어로부터 시간 제한 구문을 입력 받아 테이블에 저장하고, 이를 중간 코드인 시간 트리에 적용한다.

시간 트리를 사용시의 장점은 원시 언어와 목적 언어의 연결이 가능하다는 점이다. 일반적인 컴파일러의 사용시 목적언어는 최적화에 의하여 코드가 이동하게 되는데, 이 경우 목적 언어로 원시 언어를 판별하기가 어렵게 된다. 실시간 응용에 있어서 프로그래머는 원시 언어 상에서 작업하기가 쉽고, 시간 도구는 목적 언어 상에서 시간을 계산하므로 이의 연결을 쉽게하는 시간 트리의 사용은 인터페이스 구현시 장점이 있다.

실시간 컴파일러의 구문분석기는 원시 언어를 해석하여 트리 형태의 중간 코드를 만든다. 코드 발생기는 이 트리를 순행하며, 코드를 생성하는데, 루트 노드(root node)에서 리프 노드(leaf node)로 가면서 코드를 생성하고, 다시 리프 노드에서 루트 노드로 올라오면서 프로그램의 실행시간을 계산하게 된다. 코드의 생성이 각 노드에서 발생하기 때문에 리프 노드에 도달하기까지는 생성되는 코드를 알 수 없고, 이를 알기 전까지는 프로그램의 실행시간을 계산할 수 없기 때문이다.

실행 시간의 분석은 각 노드에서 이루어진다. 각 노드는 자신이 발생시키는 코드를 알 수 있으며, 코드를 발생시킬 때마다 exec_cycle()함수를 호출하여, 명령어의 실행 주기를 실행 주기 테이블로부터 찾아오게 된다. 찾아온 값은 각 노드에서 누적되며, 노드의 처리가 끝난 후에 부모 노드로 반환하게 된다. 그러므로, 각 노드에서는 자신의 실행 주기 값(ccycle)과 자식의 실행 주기 값(scycle)을 가지며, 부모 노드에 반환할 때에는 두 개의 값을 더하여 반환하게 된다. 각 노드에서 사용되는 실행시간 계산의 규칙은 그림 4에 나와 있다.

Construct	Worst case execution cycle
Alternative node	scycle(alternative) = total_cycle(condition) + max(total_cycle(true), total_cycle(else)) ccycle(alternative) = Σexec_cycle(alternative)
while(cond) node /**Count(β)**/	scycle(while) = total_cycle(cond) + β*(total_cycle(body) + total_cycle(cond)) ccycle(while) = Σexec_cycle(while)
while(cond) node /**Time(α)**/	scycle(while) = α - ccycle(while) ccycle(while) = Σexec_cycle(while)
Other node t	scycle(t) = Σtotal_cycle(t's children) ccycle(t) = Σexec_cycle(t's primitive)

* total_cycle(a) = scycle(a)+ccycle(a)

그림 4 실행시간 계산 공식

앞의 예제에서 반복문에 관한 시간 트리는 그림 5에 나와 있다. 각 노드에는 두 개의 작은 사각형이 있는데 왼쪽은 서브 트리의 주기 값(scycle)을 표시한 것이고, 오른쪽은 자체 노드의 주기 값(ccycle)을 표현한 것이다.

if문의 계산도 트리의 순행 중에 이루어지게 된다. if문은 if문과 iftail문으로 구성되는데 최악 실행 시간을 계산하기 위하여 둘 중에 큰 값이 부모 노드인 IF노드로 반환하게 된다. 그림 5에서 IF노드의 왼쪽 값이 48인 이유는 (expression)과 statement노드의 값이 더해진 결과이다. 이 때 iftail의 값은 statement 노드의 값보다 작으므로 무시되어졌다.

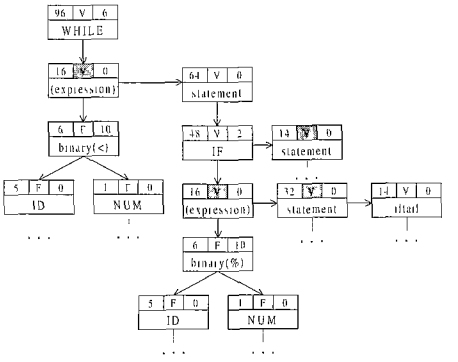


그림 4 시간 트리

Count(Max_Count) 명령은 시간 트리에 각 구성자들의 영역이 이미 표현되어졌기 때문에 단지 구성자들의 값을 Max_Count와 곱하여 표현하면 된다. 예제의

표 4 실행시간 분석 결과

```
MAIN ( )
{
 WHILE  (  i   <  100)
                ...expression... scycle =16 ,ccycle = 0
  {
    IF  (  i   %  2)
                ...expression... scycle =16 ,ccycle = 0
   { t_count  =  t_count  +  1   ;
                ...statement... scycle = 14, ccycle = 0
     sum   =   sum  +  i ;
               ...statement... scycle = 18, ccycle = 0
   }            ...statement... scycle = 32, ccycle = 0
   ELSE
     e_count  =  e_count  +  1 ;
              ...statement... scycle = 14, ccycle = 0
              ...else_cycle = 14 , if_true_cycle= 32
             ...if statement...scycle = 48, ccycle = 2
    i  =  i  +  1;
            ...statement...scycle = 14, ccycle = 0
   }          ...statement...scycle = 64, ccycle = 0
while ccycle = 402, scycle = 8016.
}
program scycle = 8418 program ccycle = 2
total cycle = 8420
```

경우 while문장의 내부가 expression과 두 개의 문장으로 구성되어졌으므로, (16 X 101) +((50 + 14) X 100))을 계산하여 WHILE 노드의 처리비용인 402(4 X 100 + 2)를 더하여 준 값이 WHILE 노드의 부모에게로 반

환하게 된다. 표 4에 그림 2예제의 시간 분석 결과를 표현하였다.

Time(Max_Time) 명령이 있을 경우, 실행 시간 분석기는 명령이 지정한 시간 안에 종료되는 것을 보장하여야 한다. 본 실시간 컴파일러에서는 외부의 시간적인 정보와 인터럽트를 이용하지 않고 정적인 시간 분석만을 이용하여 이를 해결하였다.

실행시간 분석기는 시간 트리 안의 각 노드를 F와 V로 분류하고, V로 표시된 노드에 일정한 코드를 삽입하여 줌으로써 Time()명령이 일정한 시간 안에 종료하는 것을 보장한다.

이 때 F 플래그는 시간 예측이 가능한 노드로써, 자식 노드가 모두 F이고, 노드 자신이 F인 경우에 F의 값을 갖는다. V 플래그는 노드 자신이 V이거나 자식 중에 하나라도 V가 있을 경우, V의 값을 갖는다. 선택문과 반복문 같이 동적으로 실행을 결정할 수 있는 경우에 기본적인 노드의 값이 V를 가지며, 정적으로 결정이 가능한 경우에는 F를 갖게 된다.

그림 5에서 음영 처리된 V플래그는 원래는 F플래그의 성격을 갖지만 코드 삽입을 최소화할 목적으로 자식 노드의 값을 대표하여 V값으로 변화시킨다.

V 플래그인 경우 삽입되는 코드의 내용은 다음과 같다.

```
<초기 삽입코드>
 move Max_cycle to register;
// time() 범위의 시작부분에 나타남

<일반 삽입코드>
if(register < θ) jump end_label
 register = register - θ;
// θ : 현재 노드의 실행시간
        (or 현재 노드까지의 실행시간)
         + 삽입 코드의 overhead

<종료 삽입코드>
 delay register;
```

5. 시간 분석 결과

그림 6에는 여러 가지 테스트 프로그램의 예측된 실행 시간과 실제 실행 시간을 비교하였다.

1, 2번 실험 예제는 반복문과 연산자들을 테스트하기 위한 프로그램이었고, 3, 4번 실험 예제는 반복문안에 선택구조를 포함하고 있는 예제이다. 모든 예제에서 같은 범위에 Count()와 Time()의 명령을 사용하였고, Count()는 최대치로 Time()의 값은 미리 분석기를 실행시켜 예측되어진 값으로 주었다.

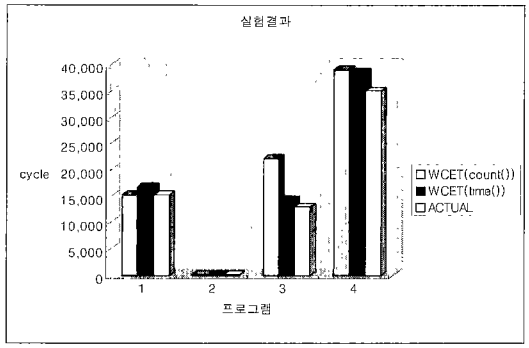


그림 6 실험 결과

1번과 2번 예제에서는 Count()함수를 사용한 최악 실행 시간과 실제 실행 시간이 같아, 만족스러운 결과가 나왔으나, 3번과 4번 예제에서는 너무 과대 평가되어 있음을 볼 수 있다. 이는 최악 실행 시간의 계산에 있어서 선택문의 경우 참의 경우와 거짓의 경우가 서로 최악 시간이 다르고, 양쪽이 어떤 확률로 선택되어지는데 반하여, 최악 실행 시간의 계산에 있어서는 무조건 최대치로 평가하여 과대 평가되어졌다. 또한, 이러한 선택문이 반복문에 내포되어졌을 경우 더욱 증폭되어 나타난다.

이와 같은 문제를 해결하기 위하여 Puschner는 [19]에서 "marker"를 도입하여 프로그래머가 시간 분석기에 더욱 많은 정보를 입력하여 주게 하였지만, 너무 프로그래머 의존적이 되어 신뢰성이 떨어지는 문제가 있다.

본 연구에서는 Time()명령으로 이러한 문제를 해결할 수 있다. Time()명령은 최악 실행 시간 계산에 영향을 줄 뿐 아니라 코드 발생시 정적 시간 분석 결과를 이용하여 효율적인 코드를 발생시키도록 한다.

선택의 내포를 알 수 없는 반복문에 Count()명령보다 Time()명령을 줌으로써 과대 평가를 없앨 수 있으며, 이 때 시간을 알 수 없는 경우, 미리 분석기를 실행시켜 보면, Time()으로 줄 수 있는 값을 예측할 수 있다.

실행 결과에서 Time()명령을 사용한 경우 예측된 실행 시간이 실제 실행시간보다 약 10%정도 과대 평가되었음을 볼 수 있다. 그러나 이것은 새로운 코드가 삽입되어 발생되어진 수치로 실제 코드상에 주어진 시간 제한 조건을 만족함을 알 수 있었다. 또한, Count()의 경우 예제에서 최대 70%정도 과대 평가가 되어진 것에 반하여 Time()의 경우 10~20% 정도의 과대평가가 있었음을 알 수 있다.

그러므로, 최악 실행시간과 최선 실행시간의 차이가 큰 경우 Time()명령을 사용하여 과대 평가의 폭을 줄일

수 있다.

6. 결론 및 추후 연구과제

실시간 시스템은 프로그램의 논리적인 오류뿐만 아니라 시간적인 오류도 치명적이 될 수 있다.

본 논문에서는 실시간 프로그램에는 적합하지 않지만 현재 가장 많이 사용하고 있는 C언어에 시간 제한 구문을 첨가하여 실시간 응용 프로그램에 적합한 언어를 설계 및 구현하였다.

본 논문에서 구현한 언어를 이용하면, 새로운 언어를 익힌다는 거부감이 없이 기존에 반복적인 실험에 의하여 구하였던, 프로그램의 시간적인 논리를 쉽게 구현할 수 있다. 또한, 코드 발생시 정적 분석을 이용함으로써, 외부의 시계나 인터럽트의 도움없이 코드를 생성할 수 있다는 장점을 가지고 있다.

본 연구의 추후 연구과제로는 병행성의 추가이다. 실시간 응용에서는 병행성이 많이 사용되어지나 C언어에서 이를 지원하지 않는 관계로 첫 계획에서는 병행성이 제거되어졌었다. 가장 작고 단순하며, C언어의 구문을 많이 파괴하지 않는 선에서 병행성을 위한 구조가 추가되어져야 할 것이다.

참 고 문 헌

[1] A. Burns and A.J, Wellings, Real-time Systems and their Programming Languages, Addison-Wesley, 1990.

[2] T.Chung, H.Dietz, Language construction and transformation for Hard real-time system, Second ACM SIGPLAN Workshop, 1995

[3] S. Hong and R. Gerber. Compiling real-time programs into schedulable code. In Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation. ACM Press, June 1993. SIGPLAN Notices, 28(6):166-176.

[4] E. Kligerman and A. D. Stoyenko. Real-time Euclid: A language for reliable real-time systems. IEEE Transactions on Software Engineering, 12:941--949, September 1986.

[5] K. J. Lin and S. Natarajan. Expressing and Maintaining Timing Constraints in FLEX. In Real-Time Systems Symposium, December 1988.

[6] V. Wolfe, S. Davidson, and I. Lee. RTC: Language support for real- time concurrency. In Proceedings of the 12th IEEE Real-time Systems Symposium, pages 43--52, San Antonio, Texas, December 1991.

[7] V. Nirkhe, S. Tripathi, and A. Agrawala. Language Support for the Maruti Real-Time System. In Real-Time Systems Symposium, December 1990.

[8] 백정현, 원유헌, 프로그램형 자동화기기를 위한 실시간 메카니즘제어 언어의 구현기법, 대한전자공학회 논문지, 제 34권 11호, 1997.

[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 Proceeding of 15th Real-Time Systems Symposium, 1994.

[10] M.G.Harmon, T.P.Baker, and D.B.Whalley. A Retargetable Technique for Predicting Execution Time of Code Segments. REAL-TIME SYSTEMS, 7(2):152 -182, 1994.

[11] F.Mueller, Static cache Simulation and Its Applications, PhD Dissertation, Florida State Univ., 1994

[12] Yau-Tsun Steven Li, S.Malik,A.Wolfe Cache Modeling for Real-Time Software:Beyond Direct Mapped Inctruction Caches, Princeton Univ., 1995.

[13] P.Puschner and A.Schedl Computing Maximum Task Execution Times - A Graph-based Approach. 1991.

[14] P. Altenbernd, CHaRy : The C-LAB Hard Real-Time System to Support Mechatronical Design, International Conference and Workshop on Engineering of Computer Based System(ECBS), Monterey, 1997.

[15] F. Bourdoncle, Effective chaotic iteration strategies with widening, Procedings of the International Conference on Formal Methods in Programming and Their Applications, 1993.

[16] P. Cousot, Semantic foundations of program analysis, in Muchnick and Jones Eds., Program Flow Analysis, Theory and Applications, Prentice-Hall, 1981.

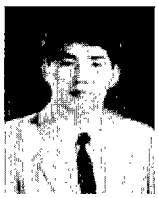
[17] Jakob Engblom, Worst-Case Execution Time Analysis for Optimized Code, Uppsala Univ. 1997

[18] L.Ko,C.Healy,E.Ratliff, R.Arnold, D. Whalley, M.G.Harmon Supporting the specification and analysis of timing constraints. In proc. of the IEEE Real-Time Tech. and App. Symposium, June 1996.

[19] P. Puschner and C. Koza. Calculating the Maximum Execution Times of Real-Time Programs. The Journal of Real-Time Systems, 1(2):159--176, September 1989.

이 준 동

1990년 홍익대학교 전자계산학과 학사. 1993년 홍익대학교 대학원 전자계산학과 석사. 1993년 9월 ~ 현재 홍익대학교 대학원 전자계산학과 박사과정. 1997년 3월 ~ 현재 원주대학 전산정보처리 조교수. 관심분야는 프로그래밍언어, 실시간시스템.



백 정 현

1985년 홍익대학교 전자계산학과 학사. 1987년 홍익대학 학원 전자계산학과 석사. 1998년 홍익대학교 대학원 전자계산학과 박사. 1987년 3월 ~ 1992년 8월 현대중공업(주) 중전기연구소 연구원. 1992년 9월 ~ 현재 우송공업대학 컴퓨터정보계열 부교수. 관심분야는 프로그래밍언어, 실시간시스템, 전자상거래 보안 및 마케팅기법.

원 유 헌

1972년 성균관대학교 수학과 이학사. 1975년 한국과학원 전자계산학과 이학석사. 1985년 고려대학교 수학과 전산전공 이학박사. 1975년 ~ 1976년 한국과학기술연구소 연구원. 1986년 ~ 1987년 R.P.I. 교환교수. 1976년 ~ 현재 홍익대학교 컴퓨터공학과 교수. 관심분야는 프로그래밍 언어론, 실시간 프로그래밍, 멀티미디어 시스템.