9

TimeBounder: 실시간 시스템 자동 분석기

㈜포멀웍스 ▌김태효
한국과학기술원 ▌방호정 · 차성덕*

개 요

최장수행시간(Worst-Case Execution Time: WCET)은 실시간 시스템 분석의 기본 정보로 작업 스케줄 분석 등 다양한 분석에 사용된다. 기존의 최장수행시간 분석은 주로 측정에 기반한 수작업으로 이루어진 바, 시간과 비용이 많이 소요될 뿐 아니라 결과의 안전성을 보장하지 못하는 단점이 있었다. 이와 같은 단점을 보완하기 위하여 최근 정적 분석기법을 이용한 최장수행시간 분석 기법에 대한 연구들이 활발하게 진행되고 있다. 정적 최장수행시간 분석 기법들은 결과의 안전성을 보장할 뿐 아니라, 자동화가 가능하여 분석 시간과 노력을 크게 절감할 수 있는 장점이 있다.

TimeBounder는 C언어 프로그램을 대상으로 하는 정적 최장수행시간 자동 분석 도구이다. TimeBounder는 최장수행시간을 분석하는 과정에서 파생되는 다양한 프로그램 흐름 정보와 분석 결과를 효과적으로 시각화하여 개발자로 하여금 프로그램에 대한 이해를 높이고, 디버깅 및 최적화 과정에 쓰일 수 있도록 한다. 이와 같은 정보들과 내부적인 분석 기법들은 테스트 데이터 자동 생성 등 다양한 분석에 활용 가능하다. 본 논문에서는 TimeBounder 실시간 시스템 자동 분석 도구를 소개하고 사용되는 기법들과 활용방안을 설명한다.

1. 서 론

최근 내장형 시스템의 활용이 폭발적으로 증가하고 있다. 이는 시스템 기능이 복잡해지고 변화에 대한 높은 적응력이 요구됨에 따라, 하드웨어 중심의 기존 시스템들이 점차 소프트웨어에 기반한 내장형 시스템으로 옮겨가고 있기 때문이다. 내장형 시스템은 통상 실시간 시스템의 특성을 지닌다. 실시간 시스템이란 어떤 기능을 수행함에 있어 수행 결과의 정확성뿐 아니라, 시간적 요구사항의 만족 여부도 중요한 시스템이다. 따라서 내장형 시스템에서는 기능적 요구사항에 대한 검증뿐 아니라, 비기능적 요구사항 특히 시간 제약에 대한 검증이 필수적이다.

시간제약의 오류는 디버깅이나 테스팅을 통하여 발견하기 매우 어려운 특성을 가진다. 왜냐하면 오류 상황을 발견하더라도 이를 재연하기가 힘들고 그 원인을 파악하기 힘들기 때문이다. 따라서 시간제약 오류는 개발 후기 또는 개발 완료 후에 발견되는 경우가 많으며, 이를 수정하는데 많은 비용이 소요됨은 물론 제품의 신뢰도를 하락시키는 주요 원인이 된다.

또한 시간적 요구사항의 분석은 내장형 시스템의 성능을 높이고 한정된 자원을 최대한 활용하기 위해서도 필수적이다. 내장형 시스템의 경우 매우 제한된 환경에서 동작하기 때문에 구동 소프트웨어가 최적화되어야 한다. 이 최적화 과정을 보다 효율적으로 엄밀히 하기 위해서는 소프트웨어 수행시간에 대한 분석이 이루어져야 한다.

최장수행시간이란 특정 작업 혹은 프로그램이 선점이 없는 상태에서 그 수행에 소요되는 시간의 최댓값으로 시간제약 관련 분석의 기본 정보이다. 따라서 최장수행시간을 예측함에 있어서 다음과 같은 특성이 지켜져야 한다. i) 예측 값은 실제 최장수행시간보다 작아서는 안되고(안전성), ii) 가능한 한 실제 값에 가까워야한다(정확성). 최장 수행시간 분석의 목표는 안전한 최장수행시간 예측치를 가능한 정확하게 구하는 것이다.

최장수행시간 분석 기법은 크게 측정 기반의 분석과 정적 분석으로 나눌 수 있다. 측정 기반의 분석은 실제 입력 데이터를 구하고 이를 실제로 수행하여 측정된 소요 시간의 최댓값을 구하는 반면, 정적 분석은 프로그램 소스 코드 자체를 정적으로 분석함으로써 최장수행시간을 예측한다. 측정 기반의 분석이 시간과 비용이 많이 소요되고 결과의 안전성을 보장할 수 없는 단점이 있는 반면, 정적 분석은 항상 안전한 결과를 산출하나 정확도가 떨어지는 단점이 있다.

* 종신회원

본 논문에서는 자동 실시간 시스템 분석도구인 TimeBounder를 간략히 소개한다. TimeBounder는 정적 분석을 기반으로 하는 실시간 시스템 자동 분석기로, 소스 코드를 정적으로 분석하여 프로그램 흐름 정보를 추출하고 이를 토대로 수행 가능한 경로 및 최장수행시간을 자동으로 예측한다. TimeBounder는 소스 코드 분석부터 문서화까지 모든 검증 및 확인 단계를 지원하며, 최장수행시간 분석 이외의 분석을 모듈로 지원함으로써 다양한 분석 도구로 활용이 가능하도록 설계되었다. 또한 분석과정 중에 추출되는 소스코드에 대한 정보와 분석 결과물을 효율적으로 시각화한 사용자 인터페이스를 제공한다. 이를 통하여 개발자가 직관적으로 소스코드를 이해하고 분석 결과를 디버깅 및 최적화에 활용할 수 있도록 지원한다.

이 논문은 다음과 같이 구성되어 있다. 2장에서는 최장수행시간 분석 기법에 대해 설명하고, 3장에서는 TimeBounder의 구조와 기능에 대해 설명한다. 그리고 4장에서는 간단한 실험 결과를 살펴보고 5장에서 결론 및 향후 개선 방향을 설명한다.

2. 배경지식

2.1 최장수행시간

주어진 작업들을 한정된 시간 내에 수행해야 하는 실시간 시스템의 경우 수행 작업들의 스케줄링 분석이 필수적이다. 이러한 작업 스케줄링을 위해서는 각 단위 작업들의 최장수행시간에 대한 정보가 미리 산출되어야 한다.

최장수행시간 분석은 특정 프로그램이 수행하는데 소요되는 시간의 최댓값을 예측하는 기술이다. 이때 예측치는 안전하면서도 정확해야 한다. 왜냐하면 안전하지 못한 예측 값은 스케줄링 실패를 가져와 시스템을 불안정하게 하고, 부정확한 예측치의 경우 내장형 시스템의 제한적인 자원을 심각히 낭비하게 한다.

그림 1은 최장수행시간 예측의 특성을 보여준다. 수평선은 시간의 흐름이고 가운데 수직선은 실제 최장수행시간이다. 보통의 경우 최장수행시간을 정확히 예측할 수 없으므로, 그 예측 값은 수평선 위의 원으로 나타나게 된다. 수직선 왼쪽의 원들은 실제 최장수행시간보다 작은 예측 값들이므로 안전하지 않다. 이와 같은 값들은 측정 기반 예측 방법에서 고려하지 못한 최장수행시간 경로가 있을 경우 발생한다. 정적 기반 예측은 항상 실제 값보다 큰 예측치를 구하므로 안전성이 보장되지만, 자원의 낭비를 줄이고 그 성능을 최대화하기 위해서는 최대한 실제값에 가까운 예측치를 산출하는 것이 중요하다.

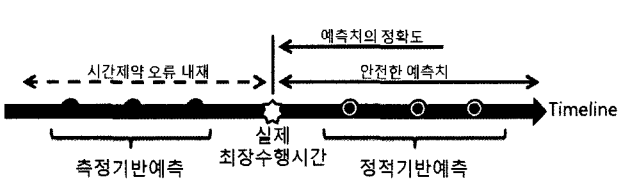

그림 1 최장수행시간 예측의 특성

2.2 최장수행시간 분석 기법

최장수행시간 분석은 소스 혹은 바이너리 코드와 해당 프로그램이 동작할 하드웨어 특성을 입력 받아, 프로그램의 최장수행시간을 예측한다. 최장수행시간 분석기법은 크게 측정 기반의 분석과 정적 분석으로 나뉜다.

2.2.1 측정 기반의 최장수행시간 분석

그림 2는 측정 기반 최장수행시간 분석의 일반적인 흐름을 보여준다. 이 기법에서는 개발자가 자신의 지식과 경험에 근거해 비교적 시간이 많이 소요될 것으로 예상되는 후보 경로들을 선택하고(경로 추출), 각각의 경로에 대한 입력 값을 준비한다(테스트 데이터 생성). 이러한 입력들을 해당 프로그램에 시뮬레이터 혹은 대상 하드웨어에서 직접 수행하여 그 수행시간을 측정한다(실행 및 측정). 측정된 수행시간들 중 최댓값을 예측치로 산정한다(최장수행시간 추출).

하지만 프로그램의 제어 구조가 복잡한 경우에는 후보 경로를 선정하는 것과 그 경로에 대한 입력 값을 추출하는 것은 매우 많은 노력과 시간이 소요되고 그 과정에서 오류 가망성이 존재하게 된다. 일례로 항공우주연구원에서 개발된 다목적 실용위성 아리랑 2호의 명령어 처리 모듈[2]의 경우, 측정 기반의 기법으로 그 최장수행시간을 분석한 바 있으며, 이를 완료하는 데 약 6 man-month가 소요되었다.

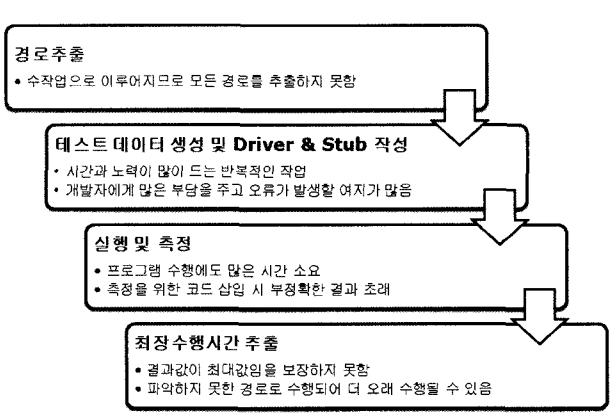

그림 2 측정 기반 최장수행시간 예측의 단계 및 문제점

2) 해당 모듈은 C로 구현되었으며, 모두 17개 함수로 약 4,100줄 정도의 크기이다.

또한 많은 경우 모든 경로를 고려할 수 없기 때문에 측정 기반의 기법으로 도출한 최장수행시간의 예측치의 경우 그 안전성을 보장할 수 없다. 안전성을 보완하기 위하여 분석 결과에 일정 비율의 시간을 덧붙여 최장수행시간 예측 값으로 사용하기도 하지만, 안전성 문제를 완전히 해결하지는 못한다.

2.2.2 정적 최장수행시간 분석

정적 최장수행시간 분석 기법들은 프로그램을 실행하는 대신 소스 혹은 목적 코드(objective code)를 정적으로 분석함으로써 최장 수행시간을 예측한다. 이러한 기법들은 보통 프로그램 경로를 분석하여 실현 가능한 경로들을 파악하고(프로그램 경로 분석), 각 기본 블록(basic block)의 수행시간을 캐시, 파이프라인 등 하드웨어 특성을 고려하여 계산한 후(하위 수준 분석), 이러한 정보를 이용해 최장수행시간 예측치를 계산하게 된다. 대표적인 정적 최장수행시간을 계산 방법에는 트리 기반 기법(tree-based technique)[1,2], 경로 기반 기법(path-based technique)[3], Implicit Path Enumeration Technique(IPET)[4,5] 등이 있다. 다음의 표 1은 정적 최장수행시간 분석 단계를 간략히 정리한 것이다.

정적 최장수행시간 분석의 장점은 (1) 자동화가 가능하여 시간과 비용을 절감할 수 있을 뿐 아니라 수작업에서 오는 오류를 배제할 수 있으며 (2) 개발자 및 분석자가 반복 작업에서 탈피하여 실제 개발에만 집중할 수 있게 하며 (3) 하드웨어 완성 전에도 검증이 가능하므로 하드웨어와 소프트웨어를 동시에 개발할 수 있는 점이다. 또한 (4) 분석 시 프로그램을 실제 수행할 필요가 없으므로 실행에 소요되는 시간을 절감할 수 있다.

그러나 정적 분석 기법은 수행 불가능한 경로를 정적인 방법으로 모두 파악하는 것이 불가능하기 때문에 일반적으로 예측치를 과다 계상되는 경향이 있다. 정적 분석에 있어 결과의 정확도는 추출되는 프로그램 흐름 정보의 양과 직접적인 관계가 있으므로 가능한 많은 흐름 정보를 추출해야 하지만, 그럴수록 더 많은 계산 비용이 소모되게 된다. 따라서 정적 분석의 경우 이러한 정확성과 비용 사이의 trade-off 관계를 어떻게 조화시키느냐가 관건이 된다.

표 1 정적 최장수행시간 분석의 단계

분석 단계	수행되는 작업
프로그램 경로 분석 (Program flow analysis)	프로그램의 기본 블록 사이의 제어 흐름과 데이터 흐름을 분석하여 프로그램 수행 정보를 추출
하위 수준 분석 (Low-level analysis)	대상 하드웨어 특성을 감안하여 기본 블록들이 수행되는데 필요한 시간(시간 정보)을 계산
계산 (Calculation)	흐름 정보와 시간 정보를 토대로 최장수행시간을 예측

3. TimeBounder

그림 3은 TimeBounder의 간략한 구조를 보여준다. 프로그램은 사용되는 프로세서의 종류와 컴파일러에 따라서 그 수행시간이 영향을 받게 된다. TimeBounder는 다양한 환경에서의 최장수행시간을 예측하기 위하여 여러 개의 프로세서와 컴파일러 지원 모듈을 plug-in 형식으로 제공한다. 또한 최장수행시간 분석뿐 아니라 다른 분석들도 쉽게 추가할 수 있도록 여러 가지 분석 모듈도 plug-in 형식으로 제공한다. 분석의 결과는 다양한 방식으로 산출되는데, 최장수행시간 분석의 경우 최장수행시간 또는 최장수행시간 경로를 생성하여 보여주게 되며, 분석 결과는 HTML 문서로 자동 생성되어 개발자 및 분석자가 추가적인 노력없이 문서화의 과정을 끝낼 수 있게 한다.

TimeBounder는 C 프로그램의 최장수행시간을 예측하는 기능 이외에 다양한 분석 기능을 지원한다. 표 2는 이러한 분석 중 대표적인 것을 정리한 것이다.

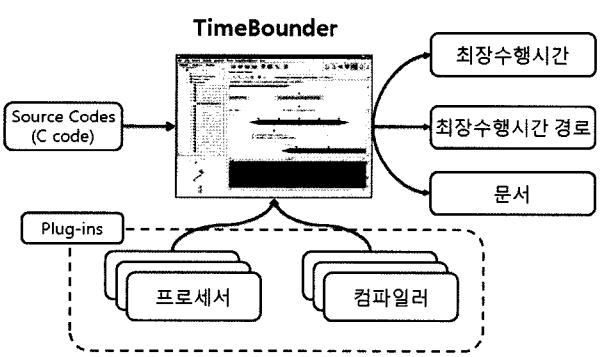

그림 3 TimeBounder의 구조

표 2 TimeBounder의 분석 기능들

분석기능	활용
소스코드 기본 분석	제어흐름 및 데이터 흐름 파악 개발중인 소스 코드 분석 레가시 코드 이해 및 분석 문서화
최장수행시간 분석	실시간 시스템 검증 작업 스케줄링 분석 성능 최적화
가능값 분석	수행 불가능 경로 추출 테스트 데이터 생성 시뮬레이션 및 자동화

3.1 소스코드 기본 분석

TimeBounder는 소스 코드의 기본 정보인 제어 흐름 및 데이터 흐름을 분석하고 이를 시각화 및 문서화 한다. TimeBounder는 주어진 C 소스 코드를 분석하여 제어흐름도(그림 4)와 scope overview graph(그림 5) 그리고 호출 그래프 등을 생성하여 보여준다. 이때 소스코드와 해당 그래프 간의 추적성을 항상 유지하여 개발자가 디버깅 및 최적화를 수행하는데 활용할 수 있게 한다. Scope overview graph는 호출 그래프를 확장한 것으로 함수간의 호출 정보와 더불어 함수 내부의 간략한 구조를 함께 보여준다.

3.2 최장수행시간 분석

TimeBounder는 C 프로그램 소스 코드와 하드웨어 특성을 입력 받아 각 모듈의 최장수행시간을 계산한다. 최장수행시간 분석의 결과는 다양한 방법으로 시각화되어 사용자에게 제공된다. 우선 TimeBounder는 최장수행시간의 분석 결과를 다양한 형태로 보고서를 생성한다. 그림 6은 최장수행시간에서 각각의 함수가 소모하는 시간을 파이 그래프로 보여주는 화면의 예이다. 이와 같은 보고서들을 토대로 개발자는 어떠한 함수를 최적화해야 하는 지 파악할 수 있다.

그림 7에서는 CFG에 시간이 많이 소요된 기본 블록을 색으로 강조하여 보여주고 있다. 따라서 사용자는 어떤 함수가 차지하는 수행시간의 비율 뿐 아니라 함수 내부에서도 기본 블록이 소요한 수행 시간을 한 눈에 알아볼 수 있다. 또한 최장수행시간이 소요된 경로를 추출하여 시뮬레이션 할 수 있는 기능을 제공함으로써 디버깅 및 최적화를 더 용이하게 한다.

3.3 가능값 분석

가능값이란 프로그램의 각각의 위치에서 변수들이 가질 수 있는 값의 범위를 나타내며, 프로그램의 각 위치에서 변수들의 가질 수 있는 값의 범위를 파악

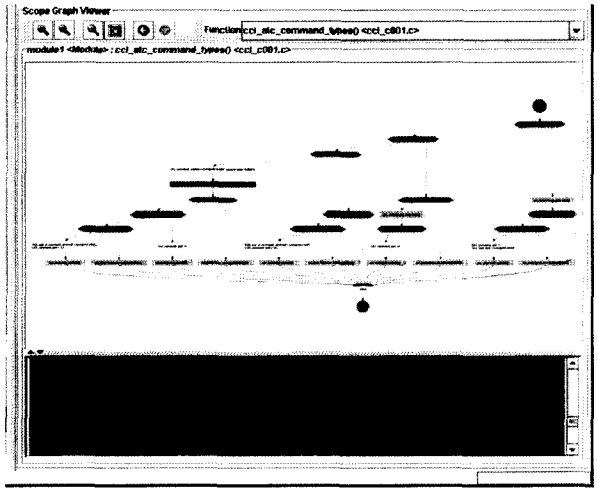

그림 4 Control Flow Graph

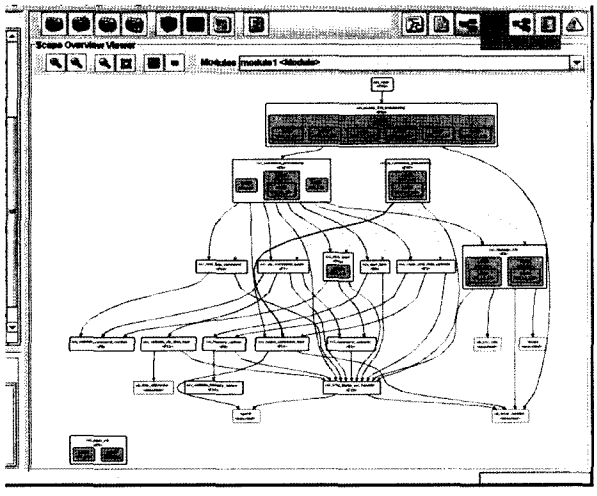

그림 5 Scope Overview Graph

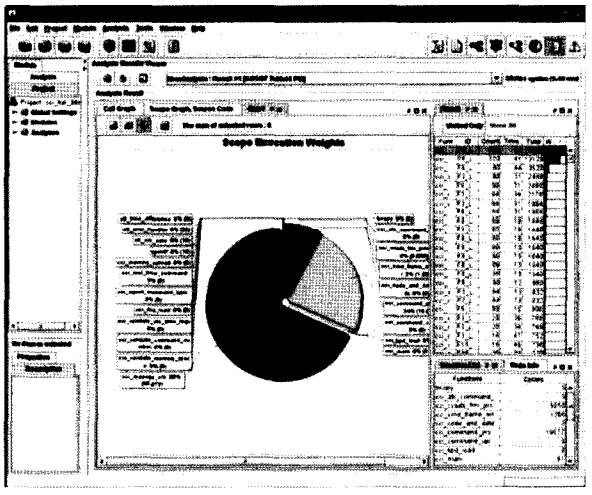

그림 6 Pie Graph of WCET Result

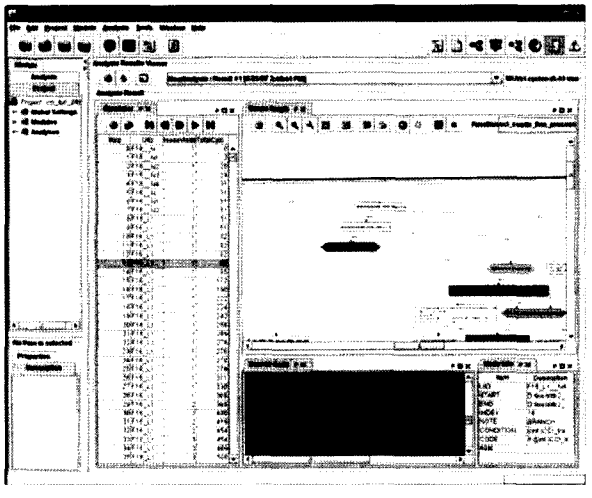

그림 7 Simulator for WCET Result

하면 실행 불가능한 경로도 추출할 수 있다. 또한 이러한 값들은 array out of bound를 분석하거나 자동화된 테스트 데이터 생성에 활용할 수 있다. 이를 위해 TimeBounder는 프로그램의 의미를 수의 도메인(numerical domain)에 대하여 요약해석(abstract interpretation)을 수행한다. 현재 TimeBounder는 변수의 구간 범위(interval range)를 그 결과물로 생성한다.

그림 8의 예제에서 함수 foo는 b를 입력으로 받아 그 부호에 따라 a 값을 증가 혹은 감소시킨 후 이를 반환한다. 함수 foo의 첫 줄의 구문이 실행된 후에 a의 가능값은 3이나 b의 값은 정해지지 않는다. 반면 셋째 줄의 구문이 실행된 후는 a의 값은 4가 되는 반면 b는 양의 값을 가지게 된다. 이처럼 모든 프로그램 위치에서 각 변수가 가질 수 있는 값의 범위를 분석하는 것이 가능값 분석이다. 그림 9는 가능값 분석의 결과를 보여주고 있다. 특정 기본 블록을 선택하면 그 블록에서의 각각의 변수가 가질 수 있는 값을 우측 하단의 표로 보여준다.

그림 10은 가능값 분석 중 발견된 수행이 불가능한 경로를 추출한 화면이다. 그림 중앙의 그래프는 수행 불가능 경로에서 한 번 이상 수행된 기본 블록을 색으로 표시한 것이며, 우측의 표는 수행 불가능 경로를 기본 블록의 나열로 보여주고 있다.

```
void foo(int b) {
  int a=3;          { a:[3~3], b:[-∞~+∞] }
  if(b>0) {         { a:[3~3], b:[1~+∞] }
    a++;            { a:[4~4], b:[1~+∞] }
  } else {          { a:[3~3], b:[-∞~0] }
    a--;            { a:[2~2], b:[-∞~0] }
  return a;         { a:[2~4], b:[-∞~+∞] }
}
```

그림 8 가능값 분석의 예제

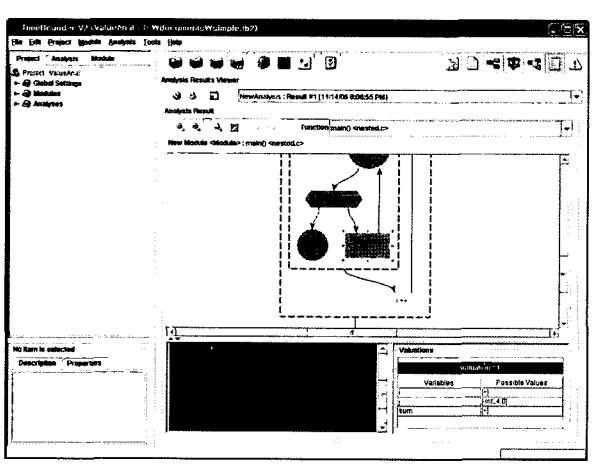

그림 9 가능값 분석의 결과

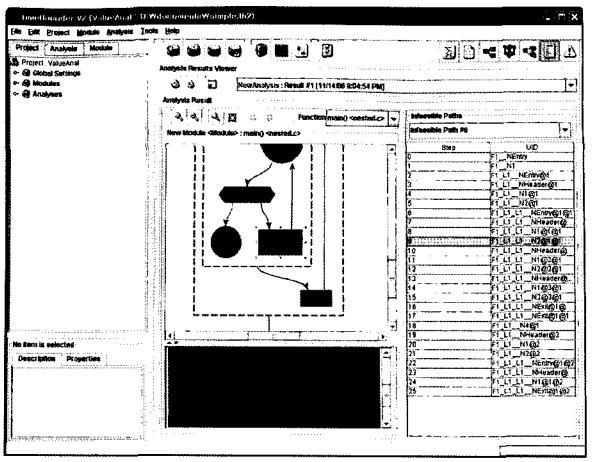

그림 10 수행 불가능한 경로 분석의 결과

3.4 TimeBounder 최장수행분석의 내부 구조

그림 11은 TimeBounder의 최장수행분석 모듈의 흐름을 개괄적으로 보여준다. 다양한 프로세서를 지원하기 위하여 TimeBounder는 프로세서에 대한 명세를 그 입력으로 받아서 수행시간 분석기를 구성한다. 이 수행시간 분석기는 캐시와 파이프라인과 같이 수행시간에 영향을 미치는 것들을 고려한 측정기이다. 이 분석기를 이용하여 소스코드의 각 기본 블록의 수행시간을 정적으로 추출한다.

TimeBounder는 최장수행분석의 정확도를 높이기 위하여 다양한 프로그램 흐름분석을 수행한다. 이때 함께 추출된 수행 불가능 경로들은 이후 최장수행경로의 수행 가능 진위를 판별하는데 사용되게 된다. TimeBounder는 정적 계산 방법으로 IPET 기법[4,5]을 사용한다. 이때 프로그램의 제어 및 데이터 흐름은 모두 flow fact라는 선형식으로 나타내게 되는데 이에 추가적으로 수행 불가능 경로를 제외하기 위한 선형식을 추가로 생성하여 더 엄밀한 예측치를 구하게 된다[6].

4. 실험 결과

TimeBounder의 효용성을 알아보기 위하여 다목적

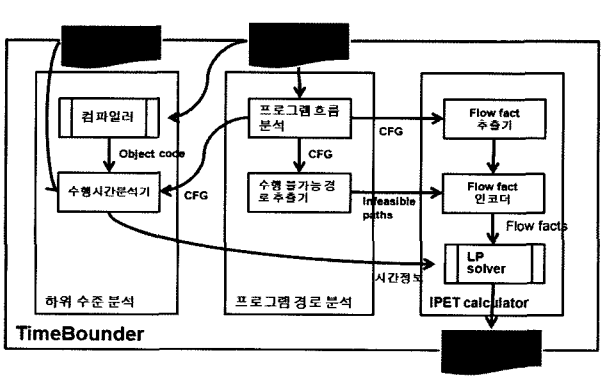

그림 11 TimeBounder 최장수행분석의 흐름

표 3 다목적 실용위성 명령어 처리모듈의 최장수행분석 결과

File	LoC	측정기반분석 (CPU cycle)	TimeBounder (CPU cycle)	B/A
MAIN.C	178	315,480	357,019	113.2%
C001.C	205	12,988	16,824	129.5%
C002.C	227	12,903	14,392	111.5%
C003.C	516	313,339	325,717	104.0%
C004.C	141	9,551	14,061	147.2%
C005.C	169	212	222	104.7%
C006.C	163	9,835	143,66	146.1%
C007.C	295	1,231	1,284	104.3%
C008.C	646	312,123	308,820	98.9%
C009.C	250	12,704	14,061	110.7%
C010.C	172	12,957	14,359	110.8%
C011.C	94	9,385	13,922	148.3%
C012.C	243	9,952	14,679	147.5%
C013.C	196	10,347	16,440	158.9%
C014.C	73	169	171	101.2%
C015.C	89	11,486	13,921	121.2%
C016.C	718	315,340	354,011	112.3%
	4,197	1,054,522	1,137,250	122.3%

실용위성 2호의 명령어 처리 모듈을 대상으로 실험을 실시하였다. 해당 위성의 명령어 처리 모듈의 경우 C 프로그램으로 구현되어 있으며, 모두 17개의 주요 함수로 구성되어 있고 총 크기는 약 4200줄 정도이다. 이 모듈은 항공우주연구원에서 위성을 개발할 당시에 측정 기반의 최장수행시간 분석을 수행한 바 있어 그때의 결과값을 비교값으로 사용하였다.

표 3은 측정 기반의 분석 결과값과 TimeBounder로 예측한 결과값을 비교한 것이다. 해당 CPU cycle은 Intel 386 임베디드 프로세서 상에서의 수행 사이클을 의미한다. 전체적으로 TimeBounder로 예측한 값이 최장수행시간을 122% 과다 계상하고 있는 것을 볼 수 있다. 이와 같은 결과는 위성 소프트웨어의 경우 안전성을 위하여 측정치에 추가적인 비율만큼의 시간을 더하여 최장수행시간의 예측치로 사용하는 것을 고려할 때 큰 자원의 낭비 없이 사용할 수 있는 예측치라 할 수 있다. 실제 측정 기반 예측을 위하여 약 6 man-month가 소요된 것에 비하여 TimeBounder는 Intel Core 2 Duo 2.0 GHz CPU 환경에서 5분 이내에 분석을 완료할 수 있었다.

5. 결론 및 향후 개선 방향

본 논문에서는 실시간 시스템 분석도구인 TimeBounder를 간략히 소개하였다. 현재 TimeBounder는 다음 버전을 개발 중에 있고 새로운 버전에서는 다음과 같이 그 기능을 확장하고자 한다. 우선적으로 최장수행경로에 대한 테스트 데이터를 자동 생성하고자 한다. 이때 추출된 테스트 데이터는 분석자가 실제 하드웨어가 완성된 후 그 수행시간을 측정하는 데 활용이 가능하고, 예측치와의 비교를 통하여 안전성을 더 높일 수 있게 한다. 다음으로 정적 분석 기법의 정확도를 높이기 위한 연구 개발이 이루어지고 있으며, 메모리 누수 검사와 같은 추가 분석 모듈을 개발 중에 있다.

감사의 글

본 연구는 방위사업청 및 국방과학연구소의 지원과, 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 정보통신부의 유비쿼터스 컴퓨팅 및 네트워크 원천기반기술 개발사업의 지원에 의한 것임.

참고문헌

[1] C. Y. Park and A. C. Shaw. Experiments with a program timing tool based on a source-level timing schema. In 11th IEEE Real-Time Systems Symposium (RTSS' '90), 1990.

[2] Gustav Pospischil, Peter Puschner, Alexander Vrchoticky, and Ralph Zainlinger. Developing real-time tasks with predictable timing. IEEE Software, 9(5), 1992.

[3] C. Healy, R. Arnold, F. Muller, D. Whalley, and M. Harmon. Bounding pipeline and instruction cache performance. IEEE Transactions on Computers, 48(1), 1999.

[4] Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In Proceedings of the 32nd ACM/IEEE Design Automation Conference, 1995.

[5] P. Puschner and A. Schedl. Computing maximum task execution times with linear programming techniques. Technical report, Technische University at Wien, Institut fur Technische Informatik, 1995.

[6] Tai Hyo Kim, A static analysis technique for improving accuracy of worst case execution time estimation, Ph.D. Dissertation, Korea Advanced Institute of Science and Technology, 2007.

김 태 효

1998 KAIST 전자전산학과 전산학전공 학사
2000 KAIST 전자전산학과 전산학전공 석사
2007 KAIST 전자전산학과 전산학전공 박사
2007～현재 (주) 포멀웍스 대표이사
관심분야 : 정형 검증, 프로그램 정적 분석
E-mail : taihyo.kim@formalworks.com

방 호 정

1996 서울대학교 경제학과 학사
2003 KAIST 전자전산학과 전산학전공 석사
2003～현재 KAIST 전자전산학과 전산학전공 박사과정
관심분야 : 소프트웨어공학, 정형기법
E-mail : hjbang@dependable.kaist.ac.kr

차 성 덕

1983 University of California, Irvine 전산학 학사
1986 University of California, Irvine 전산학 석사
1991 University of California, Irvine 전산학 박사
1994～현재 KAIST 전자전산학과 전산학전공 교수
관심분야 : 정형기법 및 명세, 정보보호, 침입탐지
E-mail : cha@dependable.kaist.ac.kr