Time bounder : WCET 분석 도구

박수현, 방호정, 김태효, 차성덕, 이종인
한국과학기술원 전자전산학과 전산학전공, 한국항공우주연구원
{suhyun, hjbang, taihyo, cha}@salmosa.kaist.ac.kr, jilee@kari.re.kr

Time bounder : WCET Analysis tool

Suhyun Park, Hojung Bang, Taihyo Kim, Sungdeok Cha, Jongin lee
Dept. of EECS, Div. Of CS, Korea Advanced Institute of Science and Technology

요 약

인공위성과 원자력발전소와 같은 안전필수시스템의 경우, 각 작업이 주기 안에 완료되어야 하는 실시간적 특성을 가진다. 시스템이 이와 같은 요구사항을 만족하는가를 판단하기 위해서는 프로그램의 최장수행시간을 분석하는 것이 필수적이다. 본 논문에서는 프로그램을 직접 실행하지 않고 최장수행시간을 분석하기 위해 구현한 Time bounder 도구를 소개하며, 다목적실용위성 2호에 탑재되는 프로그램을 대상으로 수행한 실험 결과를 분석한다.

1. 서론

최장수행시간(Worst Case Execution Time: WCET)의 분석이란 프로그램이 실제 시스템에 탑재되기 전에, 프로그램을 실행하는데 소요되는 최장 시간을 예측하는 것을 의미한다. 분석의 결과인 WCET 예측치는 단위 작업을 최소 주기 내에 완료하고 제한된 자원을 효율적으로 사용해야 하는 실시간 임베디드 시스템에서 각 작업을 스케쥴링하기 위한 필수적인 정보로 사용된다. 그러나 시스템의 모든 가능한 행위를 실행하는 것은 불가능하기 때문에 단순히 프로그램을 실행하는 것만으로 안전한 WCET를 얻을 수 없다. 따라서 이론적으로 가능한 프로그램의 모든 행위를 고려하여 안전하면서 실제 WCET에 근접한 WCET를 분석하기 위한 연구가 진행되고 있다. 본 논문에서는 프로그램을 실행하지 않고 WCET를 예측하는 Time bounder 도구를 구현하고, 다목적 실용위성 위성 2호에 탑재되는 코드를 대상으로 실험을 수행한 결과를 분석한다. 실험 결과, Time bounder는 기존에 수동으로 WCET를 분석하기 위해 소요한 시간과 노력을 줄일 수 있었으며, 사용자가 프로그램의 흐름에 대한 정보를 충분히 입력해 준다면 실제 WCET에 근접한 예측치를 도출하였다.

2. Time bounder의 WCET 분석 과정

Time bounder는 프로그램의 WCET를 분석하기 위하여 그림 1과 같은 과정을 거친다.

Program Flow Analysis는 C 소스코드를 입력받아 이를 flow graph의 형태로 변환한다. 생성된 flow graph에서 구조적으로 불가능한 경로를 제외하고 WCET를 분석하기 위하여, Time bounder는 flow graph로부터 구조적인 제약 조건을 자동으로 생성한다. 이외에도 사용자는 프로그램의 흐름에 대한 제약조건을 추가로 입력할 수 있다. 이는 Time bounder가 어셈블리 코드가 아닌 소스코드에 대한 flow graph를 생성하는 이유로서, 사용자로 하여금 flow graph에서 특정 소스코드가 대응되는 노드를 확인하고, 프로그램에 대한 제약조건을 노드들에 대한 수식으로서 기술할 수 있도록 한다. 예를 들어, 그림 3은 그림 2의 예제 프로그램에 대한 flow graph이다. 사용자는 while loop이 12번 이상 반복되는 경로를 제약하기 위하여, while 조건문에 대응하는 노드 N4_F1L1가 최대 11번까지만 반복된다는 정보를 *N4_F1L1 <= 11* 과 같이 수식으로서 입력할 수 있다.[2] 이러한 프로그램의 제약조건이 많이 추가될수록, 실행 불가능한 경로를 제외할 수 있으므로 실제 WCET에 근접한 예측치를 구할 수 있다.

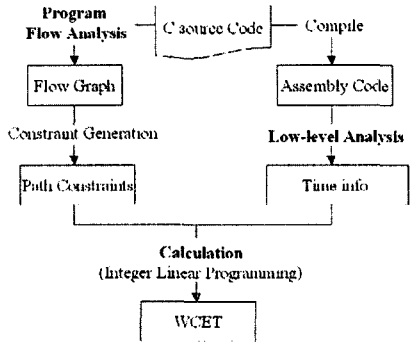

그림 1 WCET 분석 과정

Low Level Analysis는 주어진 하드웨어 환경을 가정하여 flow graph에서 각 노드를 실행하는데 소요되는 시간을 계산한다. 이를 위해 Time bounder는 소스코드를 컴파일하여 어셈블리 코드를 생성한 후, flow graph에서 각 노드의 소스코드에 해당하는 어셈블리 코드를 매핑한다. 현재 컴파일러는 GCC 3.x와 Microsoft Visual C++ 1.52 버전을 지원하며, Intel 80386 프로세서를 가정하여 실행 시간을 계산하였다. 특히 프로세서가 cache 및 pipeline을 지원할 경우, 실행 시간을 단축시키므로 이러한 하드웨어 특징은 이 단계에서 고려해야 한다. 그림 3은 flow graph에서 특정 노드를 선택한 화면으로, 그 노드의 실행 시간과 어셈블리 코드를 확인할 수 있다.

```
while(i ... = 10){
    j = i;
    while (a[j] ... a[j-1]) { ... }
  i++;
}
```

그림 2 예제 프로그램

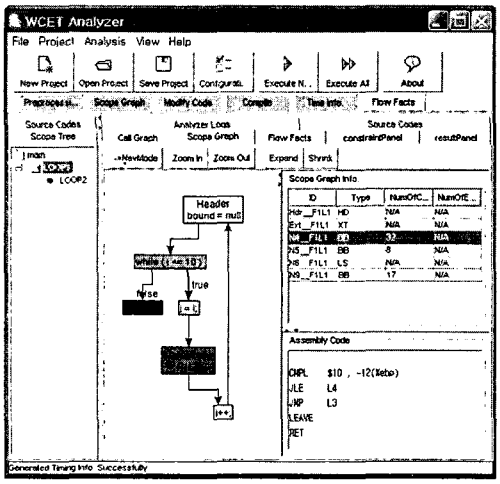

그림 3 예제 프로그램의 Flow graph

Calculation 단계에서는 이전 단계에서 생성한 경로에 대한 제약조건과 각 노드의 실행시간에 대한 정보를 가지고 WCET 예측치를 구한다. 이를 위해 Time bounder는 Implicit Path Enumeration(IPE) 기법을 채택하였다. IPE 기법이란 경로를 명시적으로 다루지 않고, 경로 내의 노드들의 수행횟수만을 다루는 것을 의미한다. 따라서 Time bounder에서는 longest path search algorithm으로 flow graph에서 명시적인 최장 경로를 찾는 대신, 프로그램의 실행시간을 최대로 만드는 각 노드의 실행 횟수만을 구한다. 식 1은 프로그램의 최장실행시간에 대한 식으로서, 각 노드의 실행 횟수(x_i)와 실행 시간(t_i)을 곱하여 모두 더한 값을 최대로 만드는 값이다. t_i의 값은 이미 Low-level analysis 단계에서 구했으므로, 변수 x_i의 값을 구함으로써 WCET를 획득할 수 있다. 단 변수 x_i의 값은 WCET를 최대로 만들면서, 동시에 프로그램의 경로에 대한 제약조건을 만족해야 한다.[3]

$$WCET = MAX(\sum_{i=1}^{N} t_i x_i)$$

식 1 WCET의 식

Time bounder는 IPE 기법에 기반하여 WCET를 분석하므로 최장실행시간을 도출하는 각 노드의 실행횟수는 계산하지만, 명시적인 최장 경로는 도출하지 않는다는 한계를 가진다. 반면 경로의 순서 정보를 무시함으로써, 큰 규모의 프로그램도 분석할 수 있다는 장점이 있다. 그림 4는 예제 프로그램의 WCET의 분석결과로서, 프로그램을 실행하는데 최악의 경우 8928 CPU cycles이 소요되며 각 노드의 실행 횟수를 확인할 수 있다.

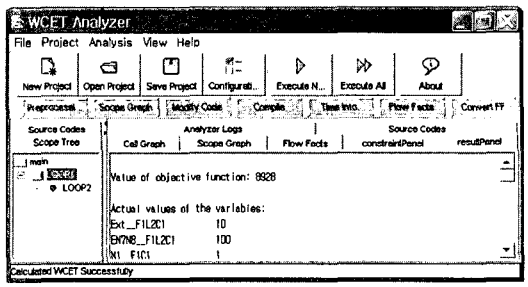

그림 4 예제 프로그램의 WCET 분석 결과

3. 실험

본 도구의 효율성을 보이기 위하여, 다목적 실용위성 2호에 탑재되는 소프트웨어의 일부인 CCI 모듈을 대상으로 WCET를 분석하는 실험을 수행하였다. CCI 모듈은 인공위성의 명령어를 처리하는 모듈로서, 총 18개의 함수로 구현되어 있고 소스 코드의 길이는 주석을 포함하여 3981 줄이다. 그림 5는 CCI 모듈에 포함된 함수들의 호출 관계를 보여주는 call graph로서, 분석의 대상이 되는 함수가 소스코드가 제공되지 않은 함수를 호출할 경우, Timebounder는 사용자로 하여금 수동으로 실행 시간을 입력하도록 한다.

실험 결과는 항공우주연구원(KARI)에서 CCI 모듈의 개발자들이 수동으로 산출한 WCET 값과 비교하였다. 프로그램을 가장 잘 파악하고 있는 CCI 모듈의 개발자들은 최장 경로를 예상한 후, 이 경로를 지나는 테스트 데이

터를 생성하고, 이를 시뮬레이션의 입력으로 이용하여 프로그램을 모의 실행함으로써 WCET를 산출하였다.

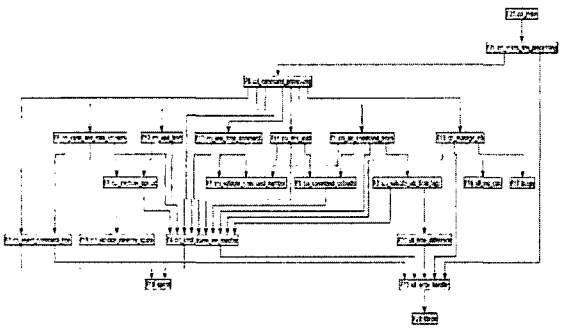

그림 5 CCI 모듈의 call graph

본 실험에서는 WCET의 분석 대상이 되는 함수의 소스 코드만을 입력하고, 대상 함수가 호출하는 함수들의 실행시간은 KARI에서 수동으로 산출한 값으로 대입하였다. 이는 하위 함수의 실행시간을 동일하게 함으로써, 각 대상 함수의 결과 값과 KARI에서 수동으로 산출한 결과 값을 비교할 수 있도록 한다. 표 1은 실험 결과로서 KARI 열은 항공우주연구원에서 수동으로 산출한 WCET 예측치이며, Time bounder 열은 본 도구에서 분석한 WCET 예측치이다. 실험 결과, 1번과 14 ~ 16번 함수를 제외하고는 KARI의 예측치의 120% ~ 332% 정도의 비율로 비교적 근접한 WCET를 도출함을 알 수 있다.

106배의 차이를 보이는 cci_command_processing 함수(1번)는 인공위성 명령어를 처리하기 위하여 준비하는 함수로서, loop을 포함하며 loop 내부에서 cci_manage_cib 함수를 호출한다. 본 실험에서는 이론적으로 loop이 최대 84회 반복된다고 가정하였으나, 개발자가 수동으로 WCET를 분석할 때는 이 함수가 단 한번 실행된다고 가정하였다. 따라서 cci_command_processing 함수의 loop 내부에서 호출하는 cci_manage_cib 함수가 1회가 아니라 84회 반복하게 되어, WCET 예측치가 과다 계상되었다. cci_command_processing 함수의 loop bound를 KARI와 동일하게 1회로 가정한 결과, Time bounder의 예측치는 KARI의 결과값의 130%인 363862 cycle로 근사한 값을 도출하였다. 즉 사용자가 프로그램의 경로에 대한 제약조건을 충분히 입력한다면, Timebounder는 실제 WCET에 근접한 예측치를 도출함을 알 수 있다.

14번~15번 함수의 경우, Time bounder가 KARI에서 산출한 결과 값보다 작은 예측치를 도출하는데, 이는 Time bounder와 KARI에서 실행시간을 계산하는 방식이 다르기 때문이다. KARI에서는 code simulation을 수행함으로써 CPU cycle의 수를 산술적으로 계산하는 반면, Time bounder는 CPU simulation을 수행함으로써 instruction prefetch와 같은 pipeline의 특성을 고려하기 때문에 KARI의 예측치보다 작은 값이 도출되었다.

표 1 실험 결과

No.	Module name	File Name	CCI PDA esimation (CPU cycle)		
			KARI	Time bounder	Percent
1	cci_command_processing	CCI_C003.C	281,770	30,104,200	10684%
2	cci_kpd_load	CCI_C006.C	9,554	31,708	332%
3	cci_ccsds_frm_processing	CCI_C016.C	283,391	775,723	274%
4	cci_code_and_data_uploads	CCI_C002.C	12,715	31,639	249%
5	cci_atc_command_types	CCI_C001.C	12,789	28,122	220%
6	cci_manage_cib	CCI_C008.C	280,932	495,264	176%
7	cci_validate_atc_time_tags	CCI_C013.C	9,895	16,669	168%
8	cci_command_uploads	CCI_C004.C	9,425	15,069	160%
9	cci_rtcs_load	CCI_C012.C	9,705	15,500	160%
10	cci_main	CCI_MAIN.C	283,485	352,470	124%
11	cci_real_time_command	CCI_C010.C	12,765	15,340	120%
12	cci_validate_command_number	CCI_C005.C	170	204	120%
13	cci_memory_upload	CCI_C009.C	12,578	15,069	120%
14	cci_validate_memory_space	CCI_C014.C	125	116	93%
15	cci_reject_command_type	CCI_C011.C	9,311	8,560	92%
16	cci_cmd_frame_err_handler	CCI_C015.C	11,424	8,452	74%

4. 결론

Time bounder는 프로그램의 WCET를 정적으로 분석하기 위해 구현한 도구로서, 인공위성에 탑재되는 프로그램을 대상으로 실험을 수행한 결과 기존에 수동으로 분석한 예측치에 근접한 값을 도출하며, WCET 분석에 소요한 시간과 노력을 줄일 수 있었다. 기존의 WCET 분석에서는 소프트웨어 명세와 제반 환경을 고려하여 가능성이 희박한 부분을 제외한 입력 데이터를 추출하여 실험한 반면, 본 실험에서는 소스 코드만을 분석하여 이론적인 최장 수행시간을 산출하기 때문에, 특정 함수에서는 기존 예측치를 크게 상회하는 값을 도출할 수도 있다. 이는 사용자가 loop의 최대반복횟수[4]나 프로그램의 흐름에 대한 추가적인 정보를 충분히 입력함으로써 극복할 수 있다. 향후에는 실제 WCET에 근접한 예측치를 도출하기 위하여 프로그램으로부터 자동으로 loop bound 및 경로에 대한 제약조건을 추출하기 위한 연구를 수행할 것이다.

참고문헌

[1] J. Engblom, A. Ermedahl, M. Sjoedin, J. Gubstafsson and H. Hansson, "Worst-case execution-time analysis for embedded real-time systems", *Journal of Software Tools for Technology Transfer*, 2001

[2] J. Engblom and A. Ermedahl, "Modeling Complex Flows for Worst-Case Execution Time Analysis", *IEEE Real-Time Systems Symposium* (RTSS'00) 394-405, 2000

[3] Yau-Tsun Steven Li and Sharad Malik, "Performance Analysis of Embedded Software Using Implicit Path Enumeration", *Workshop on Languages, Compilers, Tools for Real-Time Systems*, 88-98, 1995

[4] C. Healy, M. Sjodin, V. Rustagi and D. Whalley, "Bounding Loop iterations for Timing Analysis", *In Proc. 4th IEEE Real-Time Technology and Application Symposium*, 1998