韓國컴퓨터情報學會 論文誌
第15卷 第4號, 2010. 4.

2010-15-4-1-5

소프트웨어 감시 기법을 활용한 정적 실행시간 분석의 신뢰성 향상

김 윤 관*, 김 태 완**, 장 천 현***

Improvement of Reliability of Static Execution Time Analysis Using Software Monitoring Technique

Yun-Kwan Kim*, Tae-Wan Kim**, Chun-Hyon Chang***

요 약

시간적 정확성을 필요로 하는 시스템은 신뢰성을 위하여 실행시간에 관한 정확한 설계와 검증이 필요하다. 따라서 실행시간의 분석을 위한 개발 지원 도구가 필요하고 이를 위한 많은 연구가 진행되고 있다. 이러한 개발 지원 도구의 분석 방법은 정적 분석 방법과 측정 기반 분석 방법의 두 가지로 구분된다. 먼저 정적 분석은 짧은 시간에 분석이 가능하지만, 다양한 하드웨어의 존재로 인해 I/O 정보 예측이 어려워 분석 결과의 신뢰성이 떨어진다. 두 번째로 측정 기반 분석은 실제 결과에 근접한 분석이 가능하지만, 사용하기 어렵고 분석에 걸리는 시간이 길다. 이러한 분석 방법의 문제를 해결하기 위하여 본 논문에서는 정적 분석 과정에 소프트웨어 감시 방안을 적용한 방법을 제안한다. 제안하는 분석 방안은 정적 분석을 통해 감시가 필요한 대상을 자동으로 결정하고 감시 결과를 통해 과대 예측을 줄일 수 있다. 따라서 감시에 대한 어려움과 시간의 부하를 줄이고 정적 분석의 가장 큰 문제점인 신뢰성을 향상시킬 수 있다.

Abstract

A system which needs timely accuracy has to design and to verify correctly about execution-time for reliability. Accordingly, it is necessary for timing analysis tools, and much previous research worked. In timing analysis tool, there are two methods. One is a static analysis, and the other is a measurement based analysis. A static analysis is able to spend time less than a measurement based analysis method, but has low reliability of analysis result caused by hard to estimate time of I/O caused by various hardware. A measurement based analysis can be close analysis to real result, but it is hard to adapt to actual application, and spend a lot of time to get result of analysis. As such, this paper present a software monitoring architecture to supply reliability of static analysis process. In a presented architecture, it can select target as needed measurement through static analysis, and reuse result of measurement exist. Therefore, The architecture can reduce overload of time and performance for measurement, and improve the reliability which is the worst problem of static analysis.

▸ Keyword : 실행시간 분석(execution time analysis), 정적 분석(static analysis), 소프트웨어 감시(software monitoring)

• 제1저자 : 김윤관 교신저자 : 김태완
• 투고일 : 2010. 03. 08, 심사일 : 2009. 03. 10, 게재확정일 : 2010. 03. 31.
* 건국대학교 컴퓨터공학부 박사과정 ** 명지대학교 차세대전력기술연구센터 연구교수 *** 건국대학교 컴퓨터공학부 교수
※ 이 논문은 2008 학년도 건국대학교의 지원에 의하여 연구되었음

Ⅰ. 서 론

실시간 임베디드 시스템과 같이 시간적 정확성을 필요로 하는 시스템은 개발 과정에서 실행시간에 관련된 정확한 설계와 검증이 필요하다[1]. 특히 임베디드 시스템의 사용 확대에 따라 제한된 자원에서의 시간적 정확성 확보는 더욱 중요시 되고 있다. 이러한 시간적 정확성은 특정 작업이 정해진 수행마감 시간을 지키는 것으로 이루어지기 때문에 이를 위하여 특정 작업의 실행시간 분석이 필요하다. 따라서 실행시간 분석을 위한 여러 연구가 진행되었고 시스템의 개발 기간을 단축하고 질적 향상을 위하여 개발 지원 도구가 필요하다[2, 3, 4]. 이러한 개발 지원 도구가 작업의 실행시간을 분석하는 방법은 크게 두 가지로 구분할 수 있다. 첫 번째로, 정적 분석 방법은 분석에 걸리는 시간이 짧고 단순한 경로를 예측하거나 구조 해석이 쉽다. 하지만, 사용자 및 네트워크를 통한 입출력 정보의 예측이 불가능하고 다양한 하드웨어 벤더들이 존재함에 따라 모든 하드웨어의 지원이 어려워 분석이 정확하지 않고 과대 예측이 이루어져 신뢰성이 떨어지는 문제점이 있다. 여기서 과대 예측이란 실제 실행시간보다 지나치게 크게 실행시간을 예측하는 것을 말한다. 두 번째로, 측정 기반 분석은 실제 수행을 통한 측정 결과를 기준으로 분석을 수행하기 때문에 실제 결과에 근접한 분석이 가능하지만, 분석을 위한 별도의 코드 작성을 필요로 하기 때문에 실제 응용에 적용하기 어렵고, 수행을 통해 결과를 얻기까지 오래 걸리는 문제점이 있다[5].

따라서 본 논문에서는 정적 실행시간 분석 과정의 신뢰성을 보완하고 측정에 걸리는 부하를 해결하기 위하여 소프트웨어 감시 기법을 적용한 분석 방안을 제안한다. 제안하는 방안은 먼저 정적 분석을 통해 분석이 불가능하거나 어려운 부분을 결정하고, 결정한 부분은 수행을 통한 감시로 측정한다. 마지막으로 측정한 결과와 정적 분석 결과를 결합하여 최종 분석 결과를 도출한다. 이는 정적 분석을 통해 감시할 대상의 범위를 제한함으로써 감시를 위한 작업과 시간을 줄일 수 있다. 또한, 분석 과정에서 분석이 불가능한 부분에 대처할 수 있게 하고 정적 분석의 가장 큰 문제점인 실행시간의 과대 예측으로 인한 낮은 신뢰성을 올릴 수 있다. 따라서 과대 예측으로 인한 오차를 보완함으로써 정적 실행시간 분석의 신뢰성을 향상 시킬 수 있고, 이를 개발 지원 도구에 적용하여 그 활용성을 높이며 개발 부하와 기간의 단축에 기여할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 정적 실행시간 분석 기법에 대하여 소개하고 소프트웨어 감시 기법에 대하여 알아본다. 3장에서는 소프트웨어 감시 기법을 적용한 분석 기법을 소개하고 이를 적용한 결과를 사례 연구를 통해 알아본다. 마지막으로 결론과 향후 연구 방향을 제시한다.

Ⅱ. 관련 연구

1. 정적 분석 도구

정적 분석은 프로그램을 수행하지 않은 상태에서 여러 가지 필요한 정보를 얻는 방법을 의미한다. 이 중에서 프로그램을 수행하지 않은 상태에서 소스코드, 또는 실행파일을 분석하여 실행시간을 예측하는 것을 정적 실행시간 분석이라고 한다. 정적 실행시간 분석은 시간적 정확성을 중요시 하는 실시간 시스템 분야에서 주로 연구되었고 이 중에서 응용 프로그램의 소스코드를 기반으로 실행시간을 분석하는 Program segment Time Execution Time Bound Analyzer (PTETBA)는 개발자의 코드를 분석하여 반복횟수와 실행시간을 정적으로 분석하는 도구이다[6]. 프로그램의 실행 시간은 소스코드가 가지는 반복문의 반복횟수나 경로에 따라 크게 달라질 수 있기 때문에 PTEBA는 최악 경로 분석을 위한 경로 기반 분석 기법[7]과 실행시간 분석을 위한 트리 기반 분석 기법을 결합하여 실행시간을 분석하고 정확한 반복횟수의 분석을 위해 제어변수정보테이블을 사용한다. 그림 1은 PTETBA의 분석을 수행하는 구조를 나타낸다.

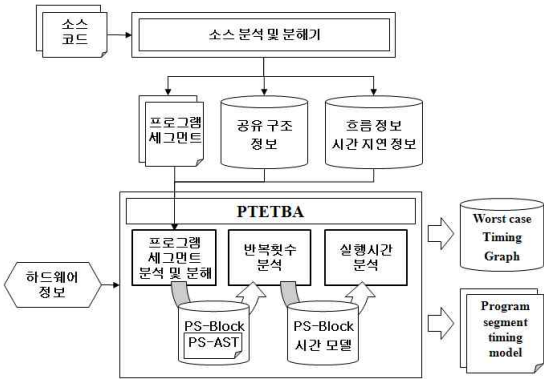

그림 1. PTETBA 구조
Fig. 1. PTETBA Architecture

그림 1에서 소스코드 분석 및 분해기는 원시소스코드를 분석하여 실행시간 분석 과정의 기본 정보를 생성하고[8], PTETBA는 소스코드 분석 및 분해기에서 분석된 정보와 하드웨어 정보를 분석하여 프로그램 수행 전에 실시간성 및 응

답 시간 보장 정보를 파악할 수 있는 정적 분석을 수행한다.

PTETBA의 소스코드 분석 및 분해 과정은 소스코드를 시간 계산을 위한 최소 단위로 분해하고 PS-AST로 구성한다. 최소 단위로 분해된 PS-AST의 각 노드는 상향식 실행시간 계산을 위하여 프로세서 모델에 따라 고유의 실행시간에 대한 값을 가진다. 생성된 트리는 선택문과 반복문을 기준으로 경로 기반 분석을 위한 PS-Block을 생성하게 된다. PTETBA는 다익스트라 알고리즘을 사용하여 경로기반 분석을 수행한다. 즉, 프로그램을 순차적으로 검색하면서 자신의 밑에 있는 PS-Block들 중에서 가장 실행시간을 많이 걸리는 블록을 선택하는 과정을 반복하여 프로그램의 시작점부터 끝점까지의 최악 실행 경로를 찾아낸다.

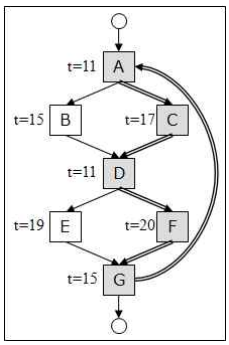

그림 2. PS-Block의 실행 경로 분석
Fig. 2. Execution Flow Analysis of PS-Block

그림 2에서 하나의 박스는 PS-Block을 나타내고 이중선과 회색으로 나타난 부분은 분석한 최악실행경로를 나타낸다. 블록의 분기가 발생할 시에 가장 실행시간이 많은 부분을 선택하기 때문에 실제 실행 경로와는 다를 수 있다. 다음으로 반복횟수 분석 과정은 앞에서 생성된 트리 구조와 블록들을 기반으로 반복횟수와 실행 경로를 분석하고 마지막 실행시간 분석 과정에서는 분석된 실행 경로에 따라 원시 소스의 실행시간을 계산하고 반복횟수를 적용해 최악 실행시간 그래프를 도출한다. 블록의 실행시간은 CPU의 명세에서 각 명령어의 클록 틱의 합으로 계산하고 의존성을 고려하여 추가되거나 감소되는 지연 시간을 분석한다.

이러한 정적 분석 기법들을 사용한 예측은 실제 실행 결과와 많은 차이가 발생한다. 차이를 유발시키는 요인으로는 파이프라인, 캐시 등이 있고 이를 사용하는 대부분의 프로세서에 대해서는 예측성을 떨어뜨린다. 따라서 이를 막기 위한 연구가 진행되고 있지만 프로세서에 대한 정보가 부족하고, 빠른 프로세서 구조의 변화로 대응이 어려운 상황이다. 또한, 프로그램 외부에서 주어지는 입력은 예측이 어렵고 프로그램의 경로나 반복횟수를 변화시킬 수 있기 때문에 사용자 입력에 대한 적절한 대응이 필요하다.

2. Bound-T 최악 실행시간 분석도구

Bound-T는 경성 실시간 소프트웨어 개발을 위한 고수준 지원 도구를 목표로 핀란드의 Tidorum에서 개발하였다. Bound-T의 주 기능은 정적 분석을 통해 프로그램이나 부 프로그램의 최악실행시간을 계산하는 것으로 자동화된 반복횟수 분석, 정적 분석을 통한 경로 분석 정보를 제공한다. 이러한 Bound-T는 서로 다른 실행 시나리오에 따라 평균 응답시간과 CPU로드를 예측하여 성능 검증에 따른 비용을 감소시킬 수 있다. 그림 3는 Bound-T의 입출력 구조를 나타낸다.

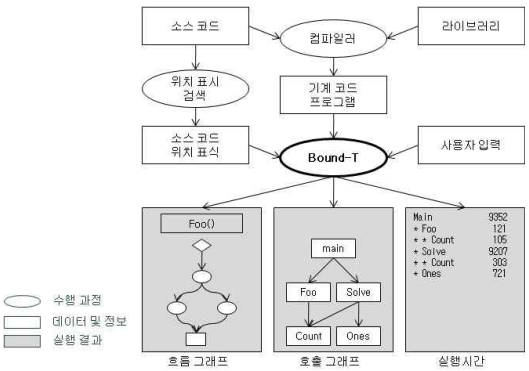

그림 3. Bound-T의 구조
Fig. 3. Bound-T Architecture

Bound-T의 구조를 살펴보면, Bound-T는 목표 플랫폼에 따라 크로스 컴파일 된 기계 코드 프로그램을 대상으로 분석을 수행하며 결과로서 흐름 그래프와 호출 그래프, 실행시간을 출력한다. 호출 그래프는 프로그램을 구성하는 부 프로그램들 사이의 관계를 표현하고, 흐름 그래프는 부 프로그램의 시작점부터 종료시점까지 분기와 반복이 일어나는 순서를 표현한다. 실행시간은 호출 그래프에 나타난 부 프로그램 단위의 최악실행시간 분석 결과이다. 분석의 순서는 결과에 따라 구분되는데, 먼저 부 프로그램의 명세에 따라 호출 그래프를 작성하고 부 부프로그램 단위로 흐름 그래프를 작성한다. 다음으로 흐름 그래프를 통해 루프의 반복횟수를 분석하고 실행경로를 찾는다. 마지막으로 IPET방법에 따라 최악실행시간을 계산한다[9]. 현재 Bound-T는 Intel 8051이나 SPARC V7, V8등의 임베디드 프로세서를 대상으로 개발되었으며 현재 ARM7 버전의 배포를 앞두고 있다[10].

본 논문에서는 IPET 분석 방법을 사용하는 사례로서 PTETBA와 함께 개선된 방식의 성능을 비교한다.

3. 소프트웨어 감시 기법

소프트웨어 감시 기술이란 응용프로그램 수행 중에 프로그램의 이벤트, 변수 등에 대한 정보를 수집하여 프로그램이 동작 상태를 점검하고 문제발생 시에 문제의 원인을 분석하기 위한 기술이다[11].

소프트웨어 감시 기술은 크게 하드웨어 기반 기술과 소프트웨어 기반 기술로 구분된다. 먼저 하드웨어 기반 기술은 실행 중에 적은 간섭으로 수행 가능하지만 전용의 장비가 필요하고 저수준의 정보만을 얻을 수 있다. 이에 반해 소프트웨어 기반 방식은 실행에 부하를 준다는 문제점이 있지만 소스나 라이브러리, 컴파일러를 통해 센서를 설치하기 때문에 비용적인 측면은 물론 이식성과 유연성, 편의성에서 하드웨어 방식보다 낫다. 이러한 소프트웨어 기반 방식은 동기식과 비동기식으로 구분된다.

동기식은 코드에 직접 감시를 위한 코드를 설치하는 방식이며 비동기식은 감시 대상으로부터의 이벤트를 감시하는 외부 프로세스를 사용하는 방식이다. 비동기식은 이벤트와 같은 추상적인 수준의 정보만을 얻을 수 있지만, 동기식은 코드의 설치를 통한 실행으로 대상에 대한 명확한 감시가 가능하다.

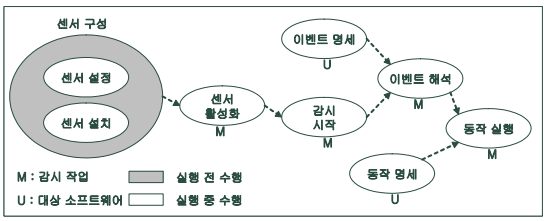

그림 4. 감시 작업의 흐름
Fig. 4. Monitoring Process

그림 4는 비동기 감시 작업의 일반적인 흐름을 표현한 것이다. 감시 작업은 크게 실행 전 준비 단계와 실행 중 수행 단계로 나누어진다. 실행 전 준비 단계에서는 감시를 위한 대상을 결정하고 감시를 위한 센서를 설치한다. 설치하는 센서는 감시 대상의 데이터 타입이나 크기 등을 결정하는 센서 설정을 수행해야 한다. 또한 준비 단계는 필요에 따라 해석을 위한 이벤트나 동작의 종류와 감시 방법에 대한 명세를 필요로 한다. 다음으로 실행 중 단계에서는 실행과 함께 감시 센서를 활성화시키고 감시를 수행하게 된다. 여기서 이벤트는 명세를 통해 정해진 특별한 정보가 나타난 경우를 의미하고 이러한 이벤트가 발생한 경우 정해진 동작이 명세 되었을 때 이를 실행하게 된다.

동기식 방식은 주로 코드 인라인 기법을 사용한다. 코드 인라인 기법이란 데이터 도출을 위한 센서를 소스레벨에서 삽입하고 실행시간에 삽입된 센서를 통하여 데이터를 도출하는 기법을 말한다[12]. 이는 실행 전에 센서 구성 단계에서 감시 대상에 따라 센서의 종류 및 위치를 결정하고 삽입해야 하기 때문에 사용이 어려운 문제점이 있다.

이러한 소프트웨어 감시 기법은 개발 언어나 개발 플랫폼, 타겟 플랫폼 등 개발 환경에 밀접한 관련이 있기 때문에 구현이 어려우며, 센서 구성 단계에서 걸리는 시간과 함께 센서 삽입으로 인한 소프트웨어의 성능 저하는 소프트웨어 감시 기법을 개발 도구에 활용하는데 가장 큰 문제점으로 지적된다.

III. 본 론

1. 정적 분석의 소프트웨어 감시 기법 적용

1.1 개선된 정적 분석 전체 구조

본 논문에서는 정적 분석의 문제점으로 지적되는 과대 예측으로 인한 신뢰성 저하를 해결하기 위해 소프트웨어 감시 기법을 적용한다. 정적 분석 기법은 원시 소스 코드를 기반으로 분석을 수행하기 때문에 소프트웨어 감시 기법에서 소스 코드의 정보를 이용할 수 있는 동기식 감시 기법을 응용하기에 적합하다. 이러한 소프트웨어 감시를 위한 절차가 적용된 제안하는 분석 방안은 세 가지 작업으로 분석을 수행한다. 첫 번째는 정적 분석을 통해 감시 대상을 결정하는 작업이고, 두 번째는 결정한 감시 대상에 맞는 센서를 설치하고 감시를 수행하는 작업이다. 마지막으로 정적 실행시간 분석을 수행하고 감시 결과를 적용하는 작업을 수행한다.

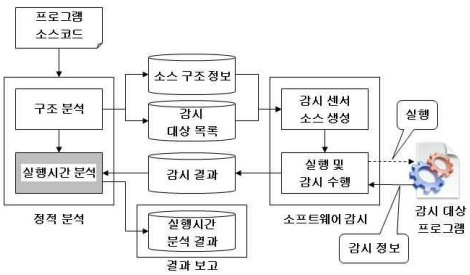

그림 5. 소프트웨어 감시 기법을 적용한 분석 구조
Fig. 5. Analysis Architecture Appling Software Monitoring

그림 5는 제안하는 방안의 전체 구조를 보여준다. 전체 작업은 크게 정적 분석과 소프트웨어 감시로 나뉘고, 정적 분석은 소프트웨어 감시의 전에 수행하는 구조 분석과 후에 수행하는 실행시간 분석으로 구분된다. 여기서 구조 분석은 기존 시스템의 소스 분석 및 분해기에 해당하고, 실행시간 분석은 PTETBA에 해당한다. 먼저 구조 분석은 입력된 소스의 흐름과 제어변수 분석을 수행하여 소스 구조 정보를 작성하고 감시 대상을 결정한다. 두 번째의 소프트웨어 감시 단계는 결정된 감시 대상 목록을 기준으로 감시 센서를 삽입하고 감시 센서 소스를 생성한다. 그리고 그 소스를 수행하여 감시 대상의 데이터를 수집한다. 마지막 실행시간 분석 단계는 정적 분석을 통해 흐름 정보와 제어변수 정보, 그리고 감시를 통해 얻은 정보를 결합하여 실행시간을 분석한다. 여기서 소프트웨어 감시 단계를 통해 얻은 정보는 실행시간 분석 단계에서 최신 정보로 유지되어 감시를 수행하지 않는 상태에서도 사용되고 정적 분석 결과의 오차를 줄이는 역할을 수행한다.

1.2 구조 분석을 통한 감시 대상의 제한과 결정

제안하는 방안에서 소프트웨어 감시 기법은 적용하기 위한 절차와 과정이 어렵고 오래 걸리는 문제점이 있어 그대로 적용하기 어렵다. 이러한 문제를 해결하기 위하여 감시 범위를 제한하고 대상을 미리 결정하도록 한다. 감시 대상의 결정은 구조 분석을 통해 자동화하여 감시를 위한 작업을 최소화한다. 그림 6은 구조 분석을 통한 감시 대상 결정 과정을 나타낸다.

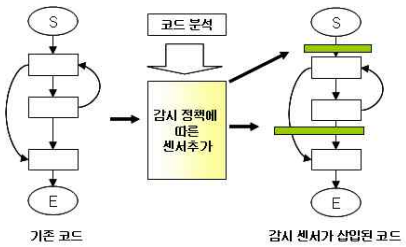

그림 6. 감시 대상 결정을 위한 구조 분석
Fig. 6. Structure Analysis to Determine Monitoring Target

그림 6에서 구조 분석은 코드 분석을 수행하고 감시 대상 목록에 센서를 추가한다. 코드 분석 과정에서는 소스 코드를 분석하여 감시가 반드시 필요한 감시 대상을 결정한다. 여기서 감시 대상은 감시 정책에 따라 결정되는데, 감시 정책은 파이프라인과 캐쉬와 같은 실행시간 영향 요소들의 영향이 큰 코드나 예측이 불가능한 제어변수들을 구분하는 방법을 규정한다. 감시 정책은 아래와 같은 순서와 규칙에 따라 감시 대상을 결정한다.

a. 값의 예측이 불가능한 제어 변수
b. 내부에 조건이나 반복이 없는 조건문 세그먼트 블록
c. 반복문 안에서 실행횟수가 가장 많은 세그먼트 블록

이러한 감시 정책에 따라서 감시 대상이 결정되면, 소프트웨어 감시 기법 중에서 코드 인라인 기법을 사용하기 위한 센서를 삽입한다. 삽입되는 센서는 그 역할에 따라 실행시간의 측정을 위한 시간 측정 센서와 반복횟수와 실행 경로의 측정을 위한 제어변수 센서의 두 가지로 구분된다.

첫 번째로 제어변수 센서의 경우, 반복문이나 조건문에서 조건의 예측이 가능한 세그먼트는 측정 대상에서 제외한다. 다음으로 예측이 불가능한 세그먼트는 반복문이나 조건문이 수행되기 전에 조건을 결정하는 제어변수에 대한 센서 코드를 삽입하여 반복횟수나 실행 경로를 측정한다. 측정된 제어변수의 값은 먼저 프로그램의 수행 경로를 통계적으로 분석하는데 사용되어 최악 경로의 실행 확률을 계산할 수 있다. 또한 반복횟수의 최대 및 최소값의 범위를 결정하여 실시간 작업의 데드라인 또는 주기의 위반 여부 확인을 도울 수 있다.

두 번째로 시간 측정 센서는 실행시간 영향 요소에 대한 실행시간 측정을 위해 사용한다. 측정 대상은 실행시간에 가장 큰 영향을 주는 반복문과 조건문을 대상으로 하고, 정적 분석을 통해 반복문과 조건문을 기준으로 나눈 코드 세그먼트에서 기존에 측정된 부분을 제외하고 새롭게 갱신된 코드 중에서 측정 대상을 선택한다. 선택된 코드 세그먼트의 전후에 실행 시간 측정을 위한 센서 코드를 삽입하여 프로그램의 실행을 통해 수행 중의 실행시간을 측정하여 저장한다.

코드 세그먼트로 분할되어 측정되는 시간은 블록 단위의 실행시간만을 나타내기 때문에, 실제 수행에서는 CPU의 파이프라인이나 캐쉬에 대한 영향으로 세그먼트 단위로 측정된 실행시간과 실제 실행시간 사이에서 오차가 발생한다. 이러한 오차를 영향 요소 값이라 하고, 이를 최소한으로 줄이기 위해서 여러 세그먼트가 결합된 그룹 단위의 실행시간 측정을 수행한다. 각 그룹은 조건문 또는 반복문에 의한 경로의 변화에 따라서 하나 이상의 세그먼트로 구성한다. 먼저 조건문 그룹은 분기 조건 세그먼트와 분기 세그먼트로 구성하고 반복문 그룹은 분기 조건 세그먼트와 반복 세그먼트, 그리고 반복 후 세그먼트로 구성한다. 이와 같이 감시 정책에 따른 감시 대상은 그를 포함한 그룹 단위로 실행시간을 측정하여 영향 요소로 인한 실행시간의 오차를 줄인다.

1.3 분석 결과와 측정 결과의 결합

이상의 두 종류의 센서를 통해 얻은 감시 결과는 실행시간 분석에 사용되며 그 상세 구조는 그림 7과 같다.

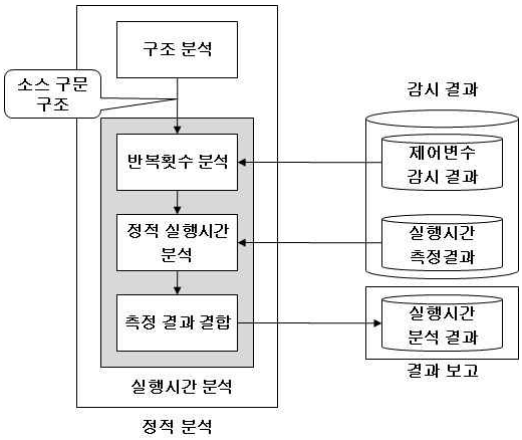

그림 7. 감시 결과 적용 실행시간 분석 구조
Fig. 7. Execution Time Analysis Using Monitoring

실행시간 분석은 구조 분석 결과인 소스 구문 구조를 바탕으로 반복횟수 분석과 정적 실행시간 분석, 측정 결과 결합의 세 단계로 이루어진다. 먼저 반복횟수 분석 단계에서는 소스로부터 얻은 반복횟수 예측 결과에 제어변수 감시 결과로부터 얻은 알 수 없는 제어변수의 값을 사용하여 모든 반복문의 반복횟수를 계산한다. 다음으로 정적 실행시간 분석을 수행하고 마지막 측정 결과 결합 과정에서 분석 결과를 측정된 블록 단위의 실행시간으로 대체하여 반복횟수와 결합한 전체 실행시간을 계산한다. 측정 결과의 대체는 실행시간에 가장 큰 영향을 주는 블록에 대해서 실행시간 영향요소가 적용된 실행시간과 함께 정적 분석을 통한 반복문의 최대 실행 횟수를 적용하여 분석 결과의 오차를 줄이도록 한다.

이러한 측정 과정은 실행 과정에서 측정을 수행하기 위한 부하를 발생시킨다. 따라서 측정 횟수를 줄이기 위하여 측정을 통해 얻은 반복횟수와 실행시간은 해당 루틴이 변하지 않는 조건에서 이후의 정적 분석과정에 재활용 되어 반복해서 측정을 수행하지 않도록 한다.

2. 감시 기법을 적용한 실행시간 분석 과정

개선된 실행시간 분석은 먼저 정적 분석을 수행하고 제어흐름을 도식화한 소스구조 정보를 생성한 후, 소프트웨어 감시를 위한 센서를 삽입하여 감시 소스를 생성한다. 이 감시 소스를 컴파일하고 수행하여 얻은 측정 결과는 정적실행시간 분석 결과와 결합되어 실행시간 및 제어 흐름 분석을 수행할 수 있도록 한다. 본문에서는 이러한 일련의 과정을 예를 들어 설명한다. 예시로 사용하는 소스는 WCET 측정을 위한 예제 중 하나로 거품정렬을 수행하는 bosrt.c이다.

그림 8은 분석 대상 소스와 코드 분석을 통한 제어 흐름과 구조를 나타낸 그림이다. BubbleSort 함수를 호출하는 부분은 사용자 입력을 대기하는 코드로 인해 실험 대상에서 제외하였다.

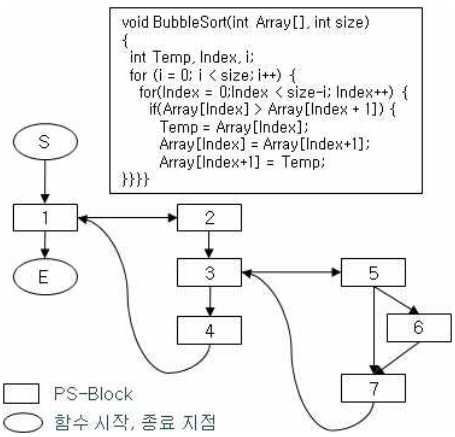

그림 8. 거품정렬의 제어 흐름과 소스 구조 정보
Fig. 8. Control-Flow-Graph and Source Structure of Bubble Sort Code

그림 8의 PS-Block은 각각 소스 정보를 가지고 있는데, 1번 블록은 거품정렬의 바깥쪽 반복문 블록을 의미하고 3번 블록은 안쪽 반복문 블록을 나타낸다. 6번 블록은 실제 데이터의 교환이 이루어지는 코드 블록이다. 도식화된 제어흐름과 구조는 소스의 실행 시 반복되는 흐름이나 선택문에 의해 발생한 분기에 따른 PS-Block 단위로 구성되고, 측정을 위한 센서 삽입의 기준점이 된다. 또한 반복횟수 및 실행시간 계산을 위한 기반 정보로 사용된다.

다음으로 감시 센서를 삽입하기 위한 감시 대상을 결정하고 감시 소스를 생성한다. 그림 8의 소스 구조 정보에서 결정되는 감시 대상은 반복문과 선택문 내부에 존재하여 실행시간에 가장 큰 영향을 미치는 블록으로서 6번 코드 블록의 실행시간과 중첩된 반복문의 반복횟수를 결정하는 제어변수인 size 변수이다.

결정된 감시 대상에 따라 목적에 맞는 센서를 소스 구조 정보에 포함시켜 감시 소스를 생성한다. 그림 9는 이러한 과정에 따라 감시 대상을 측정하기 위한 센서가 삽입된 구조와 생성된 소스를 보여준다.

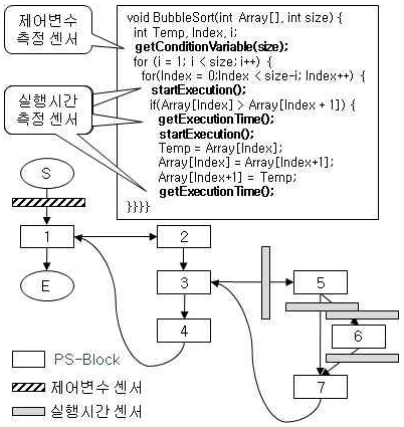

그림 9. 센서 삽입 후의 소스 및 구조 정보
Fig. 9. Structure After Sensor Insert

그림 9의 예제에서 감시 대상은 제어변수인 size와 반복문의 최하위 블록인 5번과 6번 블록의 실행시간이다. 제어변수는 반복문이 시작되기 전에 결정되기 때문에 측정을 위해 제어변수 측정 센서를 반복문 시작점인 1번 블록 앞에 삽입한다. 실행시간은 블록의 시작 시간과 종료 시간의 측정이 필요하기 때문에 하나의 블록당 두 개의 센서가 추가된다. 5번 블록은 제어문의 조건을 검사하는 블록이고 6번 블록은 조건을 만족하는 경우에 실행되는 부분에 해당하는 블록이다.

감시 대상을 결정하고 감시를 위한 소스가 생성되면, 실제 수행을 통해 감시 결과를 얻고 센서가 포함되지 않은 소스를 대상으로 수행한 정적실행시간분석 결과를 대체한다. 그림 10은 정적실행시간분석을 수행한 결과와 측정된 결과를 결합하고 전체 실행시간을 계산하는 과정을 나타낸다.

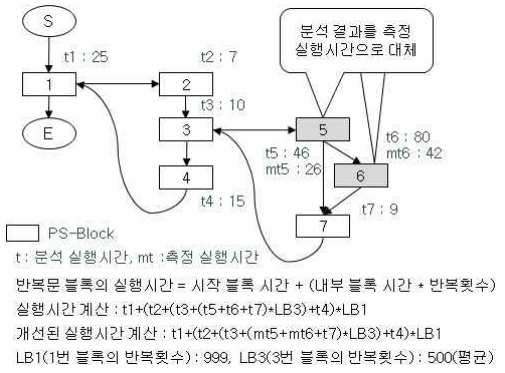

그림 10. 실행시간 분석 및 계산 결과
Fig. 10. Execution Time Analyzing Result in CFG

그림 10에서 t는 각 블록의 정적실행시간 분석 결과이고 m은 각 블록의 측정된 실행시간 결과이다. 두 시간의 단위는 CPU 클럭 cycle을 단위로 한다. 측정결과의 적용은 먼저 반복횟수에 적용되는데 LB1은 예제에서 size의 감시 결과 99회이고 LB2는 중첩 반복문의 반복횟수 계산 방법에 따라 50회로 결정된다. 5번과 6번 블록의 측정 결과는 기존의 실행시간 계산 방법에서 t5와 t6에 해당하는 값을 측정된 결과인 mt5와 mt7로 바꾸어 계산한다. 이에 따라 개선된 계산 방법에 따른 실행시간은 602,420로 나타난다.

3. 실험 및 결과 분석

앞에서 설명한 예제와 함께 개선된 실행시간 분석의 신뢰성을 확인하기 위하여 추가적인 실험을 수행하였다. 실험은 정적 분석만을 사용하는 방식과 실제 동작을 통한 측정결과, 그리고 감시 구조를 추가한 개선된 방식의 분석된 실행시간을 비교하였다. 정적 분석은 경로 기반과 트리 기반 분석 방식을 사용하는 PTETBA와 함께 IPET 방식을 사용하는 Bound-T를 사용한다. 측정 결과는 정확한 최악실행시간이라고 할 수 없지만 과대 예측의 크기를 비교하기 위해 추가하였다. 실험 대상은 앞서 예로 사용한 bsort.c와 함께 추가로 3개의 벤치마크 소스를 대상으로 한다[13].

실험 환경은 ARM7 칩을 사용한 AT91SAM7S256 보드를 사용하였으며 부트로더에 직접 프로그램을 삽입하는 방식을 사용한다[14]. 여기서 Bound-T는 사용자 입력에 대해서 반복횟수를 분석하지 못하기 때문에 1회의 반복횟수를 적용한다. 또한 정적 분석을 수행하는 Bound-T와 PTETBA는 printf와 같은 동적 호출 함수의 실행시간을 분석하지 못한다. 따라서 실험에 앞서 동일한 조건을 얻기 위해 unknown 제어변수와 printf와 같은 동적 호출 함수를 제거하였다. 실험 결과는 Bound-T와 PTETBA, 그리고 감시 기법을 적용한 개선된 PTETBA의 순서로 표 1에 정리하였다.

표 1. 코드 분석 및 감시 결과
Table 1. Result of Code Analysis and Monitoring

프로그램	설명	기존 방법			개선된 PTETBA
		Bound-T	PTETBA	측정결과	
bsort.c	거품 정렬 프로그램	943,900	671,458	351,214	602,420
fdct.c	빠른 이산 코사인 변환	11,140	17,811	1,602	3,204
matmult.c	20x20 행렬 곱셈	737,313	1,180,028	563,343	653,304
fibcall.c	단순 피보나치 계산(30)	1,241	1,413	129	258

(단위 : cycles)

타겟 보드는 전력 소모에 따라 30~55Mhz의 속도를 갖기 때문에 실험 결과는 CPU 클럭 틱을 단위로 표시하였다. 표 1의 결과에서 bsort.c의 경우 Bound-T의 결과가 가장 크게 나타나는데, 이는 반복횟수를 분석할 때 방법의 차이로 인해 발생한다. Bound-T는 basic block단위의 실행 횟수를 계산하여 최대의 실행횟수를 반복횟수로 사용하지만, PTETBA는 제어변수테이블을 사용하여 반복횟수가 변하는 중첩 반복문의 경우 평균값 계산을 통해 좀 더 정확한 반복횟수를 계산한다. fdct.c등 다른 예제는 중첩되지 않는 반복문과 고정된 배열을 사용하였으며 실행시간 결과는 PTETBA > Bound-T > 개선된 PTETBA의 순서로 나타났다.

이상의 실험 결과에서 알 수 있듯이 감시 기법을 적용하기 전의 PTETBA는 Bound-T의 결과와 비교했을 때 중첩 반복문의 분석에서 더 나은 성능을 보였지만 더욱 많은 과대 예측이 나타났다. 하지만 감시 결과를 적용한 후의 실행시간 분석 결과는 다른 두 정적 분석 결과보다 더 적게 나타났다. 이로써 정적 분석의 과대 추정의 범위를 감시 기법을 적용하여 실제 실행시간에 가깝게 과대 추정을 감소시켰음을 알 수 있다.

또한 실험 내용에서는 나타나지 않았지만, 감시 기법을 적용함으로 얻은 또 다른 특징은 비교를 위해 포함시키지 않은 반복문의 unknown 제어변수의 측정과 printf와 같은 동적 호출 함수의 실행시간 측정을 통해 분석이 가능하다는 것이다. 이는 정적 분석만으로 불가능한 정보를 측정을 통해 알아내고 이를 정적 분석에 활용함으로 더욱 다양한 프로그램을 대상으로 분석을 수행할 수 있도록 한다.

IV. 결 론

시간적 정확성을 필요로 하는 시스템은 동작에 대한 신뢰성을 보장하기 위하여 실행시간에 관련된 정확한 설계와 검증이 필요하다. 이를 위한 방법론으로써 정적 실행시간 분석 기법은 빠른 응답을 기대할 수 있지만 하드웨어 플랫폼에 대한 정보 부족으로 신뢰성이 떨어지고, 측정 기반 실행시간 분석은 실행 환경에 대한 정확한 결과를 얻을 수 있지만 실행까지 걸리는 시간이 길다는 문제가 있다. 따라서 본 논문에서는 정적 분석 결과의 신뢰성 향상을 위하여 하드웨어 플랫폼에 최적화할 수 있도록 소프트웨어 감시 기법을 적용한 정적 분석 기법을 제안하였다. 제안한 분석 기법은 코드의 분석 정보와 감시 정책을 활용하여 감시를 위한 대상 결정을 자동화하고 감시 결과를 코드 분석에 활용하여 실행시간의 과대 예측을 감소시키며 반복횟수와 함께 하드웨어 플랫폼에 최적화된 동적 호출 함수의 실행시간을 알아내 정적 분석 기준 시간의 신뢰도를 향상시킬 수 있다. 이를 통하여 실시간 소프트웨어 개발 지원을 위한 실행 경로 및 실행시간 분석에서 정확한 분석 결과를 개발자에게 제공할 수 있으며, 결과적으로 분석 도구의 활용성을 향상시켜 개발 기간 단축 및 비용 절감에 기여할 수 있다.

향후 연구 과제로는 제안한 감시 결과가 적용된 정적 분석 결과를 이해하기 쉽고 효과적으로 개발자에게 전달하며 이용할 수 있도록 하는 방안을 연구하고 이를 사용하는 도구를 개발하며 임베디드 및 분산 환경에 적용시킬 예정이다.

참고문헌

[1] P.Puschner, Ch. Koza, "*Calculating the maximum, execution time of real-time programs*," Real-Time Systems, Vol. 1, No. 2, pp.159-176, Sep. 1989.

[2] 김태완, 장천현, 김문회, "*TMO 네트워크로 구성된 분산 실시간 시스템을 위한 실시간성 분석기 설계*," ITRC forum 2004.

[3] K.H. (Kane) Kim, "Timeliness Assurance via Hybrid Approaches during Design of Distributed Embedded Computing Systems," WORDS'03F, pp.307-313, Oct. 2003

[4] K.H(Kane) Kim, Lynn Choi, Moon Hae Kim, "*I*ssues in Realization of an Execution Time Analyzer for Distributed Real-Time Objects," ASSET'00, pp.171, Mar. 2000.

[5] 김윤관, 신원, 김태완, 정천현, "PS - Block 구조를 사용한 PS-Block Timing Model의 설계 및 구현," 정보처리학회논문지D, 제13-D권, 제3호, 399-404쪽 2006년 6월.

[6] 신원, 김태완, 장천현, "정적 실행시간 분석기의 기반 구조," 한국 소프트웨어공학회 학술대회논문집, 제8권, 제1호 115-123쪽, 2006년.

[7] Jakob Engblom, Andreas Ermedahl, Friedhelm Stappert, "Comparing Different Worst-Case Execution Time Analysis Methods," RTSSWIP'00, Nov. 2000.

[8] Yun kwan kim, Won Shin, Tae wan Kim, Chun Hyon Chang, "Organizing Information for Execution Time Analysis in Real-Time Embedded Systems," SERP'07 pp.710-714, June 2007.

[9] Y-T. S. Li, S. Malik, "Performance Analysis of Embedded Software Using Implicit Path Enumeration," DAC'95, pp. 456-461, Jun. 1995.
[10] Bound-T *http://www.tidorum.fi/bound-t/*
[11] B.A. Schroeder, "On-line Monitoring: A Tutorial," IEEE Computer, Vol.28, No. 6, pp.72~78, Jun. 1995.
[12] Ulfar Erlingsson , Fred B. Schneider, "The inlined reference monitor approach to security policy enforcement," Doctoral Thesis, 2004.
[13] WCET project *http://www.mrtc.mdh.se/*
[14] AT91SAM7S256 *http://www.atmel.com/*

저 자 소 개

김 윤 관

2007 : 건국대학교 컴퓨터공학과(공학석사)
2007-현재 : 건국대학교 컴퓨터공학과 박사과정
관심분야 : 컴파일러, 실시간 프로그래밍, 임베디드 시스템

김 태 완

1996 : 건국대학교 전자계산학과(공학석사)
2007 : 건국대학교 컴퓨터공학과(공학박사)
1996-2001 : 현대중공업 기전연구소 연구원
2004-2007 : 건국대학교 컴퓨터공학부 강의교수
2007-현재 : 명지대학교 전기공학과 차세대전력기술연구센터 연구교수
관심분야 : 프로그래밍 언어, 실시간 프로그래밍, 자동화 소프트웨어, 산업기기 가시진단 제어 시스템

장 천 현

1977 : 서울대학교 계산통계(학사)
1979 : KAIST 전산학(석사)
1985 : KAIST 전산학(박사)
1985-현재 : 건국대학교 컴퓨터공학과 정교수
관심분야 : 프로그래밍 언어, 컴파일러, 실시간 시스템