ARM용 내장형 소프트웨어의 정적인 수행시간 분석 도구

(Static Timing Analysis Tool for ARM-based Embedded Software)

황 요 섭 † 안 성 용 † 심 재 홍 †† 이 정 아 †††
(Yo-Seop Hwang) (Seong-Yong Ahn) (Jea-Hong Shim) (Jeong-A Lee)

요 약 내장형 시스템에서 응용 프로그램을 구동시킬 때는 일련의 태스크들의 집합을 수행하여야 한다. 이러한 태스크들은 특정 하드웨어로 구현 될 수도 있고, 특정 프로세서에서 구동되는 소프트웨어로 구현될 수도 있다. 내장형 시스템에서 응용 프로그램을 구동시키기 위하여 하드웨어/소프트웨어의 자원 선택 및 작업 분할이 필요하게 되고 이때 하드웨어 및 소프트웨어의 성능 예측이 이용된다. 하드웨어 성능 예측과 달리 소프트웨어 성능 예측은 구동 환경과 밀접한 관계가 있으며, 하드웨어 소프트웨어 통합 설계를 위하여 최적 및 최악의 수행 시간 경계를 예측하는 것은 중요한 문제이다. 수행 시간 경계의 엄격한 예측은 저 비용의 프로세서를 사용할 수 있게 하며, 시스템 비용을 낮추는데 도움을 준다. 본 논문에서는 ARM용 내장형 시스템을 고려하여, loop문의 반복 횟수 경계 값과 프로그램의 추가적인 경로 호출 정보를 이용하여, 수행 시간의 경계를 최대한 실제 값에 접근하도록 예측하는 도구를 개발하였다. 개발된 도구는 현재 i960과 m68k 아키텍처를 지원하는 "Cinderella"라는 시간 분석 도구를 기본 도구로 활용하고 있다. ARM 프로세서를 지원하기 위하여 제어흐름과 디버깅 정보를 추출할 수 있는 ARM ELF 목적 파일 모듈을 추가하고, ARM 명령어 집합을 처리할 수 있는 모듈을 기존 도구에 추가하였다. 여러 가지 벤치마크 프로그램을 대상으로 실시한 실험 결과, 임의의 입력 데이타를 이용하고 수행 횟수를 고려한 ARMulator의 수행 시간이 구현된 도구에서의 정적인 수행 시간 예측 경계 값으로 들어오는 것을 확인할 수 있었다.

키워드 : 내장형 시스템, 최악 수행 시간, 시간분석, ARM 프로세서, ELF 목적파일

Abstract Embedded systems have a set of tasks to execute. These tasks can be implemented either on application specific hardware or as software running on a specific processor. The design of an embedded system involves the selection of hardware software resources, partition of tasks into hardware and software, and performance evaluation. An accurate estimation of execution time for extreme cases (best and worst case) is important for hardware/software codesign. A tighter estimation of the execution time bound may allow the use of a slower processor to execute the code and may help lower the system cost. In this paper, we consider an ARM-based embedded system and developed a tool to estimate the tight boundary of execution time of a task with loop bounds and any additional program path information. The tool we developed is based on an exiting timing analysis tool named "Cinderella" which currently supports i960 and m68k architectures. We add a module to handle ARM ELF object file, which extracts control flow and debugging information, and a module to handle ARM instruction set so that the new tool can support ARM processor. We validate the tool by comparing the estimated bound of execution time with the run-time execution time measured by ARMulator for a selected bechmark programs.

Key words : Embedded System, Worst Case Execution Time, Timing Analysis, Advanced RISC Machines Processor, Executable and Linkable Format Object File

· 이 논문은 2002년도 조선대학교 학술연구비의 지원을 받아 연구되었음
† 비 회 원 : 조선대학교 컴퓨터공학부
bluewind@stmail.chosun.ac.kr
dis@chosun.ac.kr
†† 비 회 원 : 조선대학교 인터넷소프트웨어공학부 교수
jhshim@chosun.ac.kr
††† 종신회원 : 조선대학교 컴퓨터공학부 교수
jalee@chosun.ac.kr
논문접수 : 2003년 12월 17일
심사완료 : 2004년 10월 8일

1. 서 론

내장형 시스템은 특정한 용도를 위해 구동되는 시스템으로 기존의 PC 시스템과는 달리 시스템의 기능들 대부분이 주문형 칩이나 특화된 목적의 ASIC(Application Specific Integrated Circuits) 또는 프로그램이 가능한 FPGA(Field Programmable Gate Array) 등으로 하드웨어에 구현된다. 또한 하드웨어의 빠른 처리능력보다는 특별한 기능을 주어진 시간 제약 안에 수행하기 위한 용도로 사용범위가 제한되어 있다. 이처럼 대부분의 내장형 시스템은 하드웨어 구성이 고정되어 있고 소프트웨어 역시 그 하드웨어에 내장된 상태로 공장에서 출하되어 사용자가 시스템에 접근할 수 없도록 되어있다. 따라서 내장형 시스템은 PC처럼 사용자가 운영체제를 바꾼다거나 재구성 할 수 없으며, 응용 프로그램 즉, 소프트웨어 역시 하드웨어에 내장된 채로 사용될 수 있기 때문에, 내장형 소프트웨어는 내장형 시스템을 구성하는 하나의 구성 요소라고 할 수 있다.

내장형 시스템을 개발하기 위해서는 하드웨어부터 소프트웨어까지 전반적인 사항을 고려할 필요가 있다. 즉 어떤 프로세서를 선택할 것인가, 하드웨어와 소프트웨어에서 수행될 작업을 어떻게 분할할 것인가, 프로그램 코드와 논리 소자들을 어떻게 합성할 것인가, 또는 시스템의 성능대 비용을 어떻게 조절할 것인가 등을 고려하여야 한다. 또한, 내장형 시스템을 개발하고자 할 경우 하드웨어의 논리회로보다는 프로세서에서 구동되는 소프트웨어의 프로그램 코드가 더 중요한 요소가 된다. 이것은 두 가지 이유 때문인데, 첫째는 반도체 회사의 제작라인의 설비비용이 증가하기 때문이고, 둘째는 시장에 내놓을 시기를 줄이는 것뿐만 아니라 작업 계획을 조정할 수 있어야 하기 때문이다. 현재의 추세는 새로운 하드웨어의 개발은 많은 시간과 높은 비용을 요구하므로 소프트웨어를 기반으로 한 내장형 시스템 개발이 더욱 활발해지고 있다. 이처럼 내장형 시스템을 개발하는데 있어 내장형 소프트웨어는 중요한 구성 요소가 되며, 소프트웨어의 성능을 평가하는 기본 척도가 된다. 특히 시스템에서 사용되는 소프트웨어는 똑같은 기능을 수행하는 소프트웨어라 하더라도 어떻게 프로그램 했느냐에 따라 코드 크기 및 성능 면에서 많은 차이가 발생할 수 있다. 즉 내장형 소프트웨어의 성능을 분석하기 위한 측정 및 예측 도구가 필요하다.

내장형 소프트웨어의 성능을 분석하기 위해서는 소프트웨어 구성 요소들에 대하여 최적 및 최악의 수행 시간 경계를 최대한 실제 값에 접근하도록 예측할 필요가 있다. 그러나 이를 위한 분석 기술들은 복잡한 프로그램을 분석하기가 쉽지 않다는 점과 캐쉬 메모리나 파이프라인을 지원하는 프로세서의 마이크로 아키텍처 모델링이 필요하다는 점, 그리고 새로운 하드웨어 플랫폼에 쉽게 적용할 수 없다는 등의 한계를 가지고 있다. 이러한 한계들은 소프트웨어의 성능 예측 시 오류를 초래할 수 있으며, 시스템을 설계할 때 불필요한 비용의 증가를 유도할 수도 있다. 소프트웨어의 정확한 성능 예측은 시스템을 개발할 때 목표 시스템의 성능과 비용을 조절하는 척도 역할을 제공할 수 있다.

본 논문은 ARM용 내장형 소프트웨어의 수행 시간을 분석하기 위하여 Princeton University에서 개발된 실시간 내장형 소프트웨어 분석도구인 "Cinderella"의 기본 모듈을 활용한다[1]. 내장형 시스템은 그것을 제공하는 벤더들이 많고 하드웨어 제품도 다양해서 기존의 성능분석 도구는 어느 한가지 제품에 제한을 가지고 있다. 그러나 이 도구는 다른 프로세서 하드웨어 모델을 쉽게 재구성 할 수 있는 장점이 있어서 새로운 제품들을 추가할 수 있는 확장성이 용이하다. 뿐만 아니라, 복잡한 프로그램을 분석하기 위해 사용자가 직접 조건들을 추가할 수도 있다. 또한 프로세서를 모델링하여 캐쉬 메모리나 파이프라인을 지원할 수도 있다. 현재 이 도구는 COFF (Common Object File Format) 이진 실행 파일을 지원하며 i960과 m68k 프로세서를 지원한다. 하지만 현재 널리 사용되는 대표적인 이진 실행 파일 포맷인 ELF (Executable and Linkable Format)는 지원되지 않는다.

본 논문에서는 현재 내장형 시스템에서 가장 널리 사용되는 ARM 프로세서를 지원할 수 있도록 기존의 분석도구를 확장하였다. ARM 머신을 지원하기 위해 ELF 목적 파일, ARM 명령어 집합 그리고 ARM 머신을 위한 모듈을 설계하고 구현하였다. 이전의 연구에서는 COFF 포맷의 실행 파일만 지원하였으나, 본 연구에서는 ELF 파일 포맷을 지원함으로써 최신의 실행 파일 포맷을 지원함은 물론, 성능분석을 위해 과거 파일 포맷으로 재 컴파일 해야 하는 부담을 제거하였다. 또한 ARM의 명령어 사이클 분석과 어셈블리 코드 분석을 통해 머신 기반 모듈로 우선적으로 ARM 프로세서를 지원하였으며, 이를 통해 ARM용 소프트웨어에 대한 성능분석을 가능하게 하였다. 이는 향후 다양한 프로세서에 대한 머신 기반 모듈을 추가적으로 지원할 수 있는 기본 틀을 제공한다. 현재 ARM용 소프트웨어의 성능을 정적으로 분석할 수 있는 툴이 없는데 비해, 본 연구의 도구는 이를 가능하게 한다. 즉, 사용자가 직접 파라미터를 입력하여 소프트웨어의 성능을 보다 정밀히 분석할 수 있도록 하였다. 구현된 도구를 이용하여 여러 가지 벤치마크 프로그램의 성능을 분석한 결과, 프로그램의 최적 및 최악의 양 극단적인 경우의 수행 시간을 예

측할 수 있었으며, 이는 실제 수행 시간에 근접한 결과 값이었다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해서 간략히 기술하고, 3장에서는 ARM을 지원하기 위한 확장된 구조를 제안한다. 4장에서는 실험에서 사용된 ARM기반 내장형 소프트웨어의 성능 예측 방법에 대해 살펴본다. 5장에서는 실험 결과를 분석하고 타당성을 검증하며 마지막 6장에서는 결론 및 향후 연구 과제를 제시한다.

2. 관련 연구

2.1 내장형 소프트웨어의 최악 수행 시간 측정 방법

지난 몇 년 동안 내장형 소프트웨어의 최악 경우 수행 시간(Worst Case Execution Time : WCET) 분석을 위한 여러 가지 방법들이 연구되어 왔다. 대부분의 분석 기술들은 프로그램의 흐름 분석이나 마이크로 아키텍처 모델링에 초점을 맞추고 문제를 해결하고자 하였다. 프로그램의 WCET을 정확히 결정한다는 것은 일반적으로 어려운 문제라고 할 수 있다. Kligerman[2]과 Puschner[3]는 이러한 문제를 해결할 수 있도록 다음과 같은 조건들을 제시하였다. 첫째, 재귀 호출 함수가 없어야 하고 둘째, 동적 배열과 포인터 같은 동적 구조도 없어야 하며, 셋째, 모든 반복 구문은 제한된 반복 구문을 가져야 한다는 것이다. 이러한 제약사항들은 특별한 언어 구조나 외부 주석 메카니즘을 필요로 한다. 이들을 위해 Real-Time Euclid[2]와 MARS-C[3]로 표현되는 언어를 개발하였다. 이러한 방법의 단점은 새로운 프로그래밍 언어와 함께 이 언어를 습득하기 위한 고비용이 요구된다는 것이다. 반면에 Mok[4]과 Park[5]은 프로그램으로부터 반복 경계와 경로 정보 같은 주석 정보 파일을 이용한다. 이 분석 도구들은 프로그램의 주석 파일과 실행 코드를 읽어들여 계산함으로써 WCET을 예측한다. 이러한 방법은 프로그램 언어를 선택할 필요가 없으며, 단지 최소의 추가 프로그램 도구만을 필요로 한다.

좀더 엄격한 WCET을 예측하기 위해 논리적으로 실행 불가능한 프로그램의 경로를 제거할 필요가 있다. 프로그램의 데이타 흐름과 심볼릭(symbolic) 기술로부터 자동으로 추론하는 기술들은 엄격한 WCET을 구하기 어렵다. 이에 비해 프로그래머는 엄격한 WCET을 위해 경로 정보의 범위를 직접적으로 제공할 수 있다. 기존의 연구[3,4]들은 반복 경계와 프로그램 구문(program statement)의 최대 수행 횟수만을 제공한다는 제약을 가지고 있다. [5]는 경로 주석과 프로그램 구문의 상호 작용, 예를 들어, 상호 배제를 고려한다. 이러한 방법은 프로그램 경로를 보다 효과적으로 기술 할 수 있고 최악 경우(worst case)의 프로그램 경로를 평가할 수 있다. 결과적으로 정확한 WCET을 예측하기 위해 사용자 주석의 범위와 엄격한 분석이 필요하다.

Li[1]는 WCET을 예측하기 위하여 프로그램 흐름 분석과 마이크로 아키텍처 모델링[6-9]을 고려하였다. 또한 프로그램의 명시적인 경로 나열을 사용하지 않는다. 이 방법은 프로그램의 흐름 정보에 있는 기본 블록을 산술식과 제약조건으로 표현한다. 제약조건들을 만족하기 위해서 ILP(Integer Linear Programming)를 이용하며 아래의 공식에 의해서 WCET을 예측한다[10].

$$\sum \forall (x_{BasicBlk} * t_{BasicBlk} - t_{effect})$$

$x_{BasicBlk}$은 기본 블록을 위한 반복 횟수이며, $t_{BasicBlk}$은 각 블록의 실행시간이며, t_{effect}는 블록들 사이의 파이프라인 효과로 얻어지는 시간을 의미한다. 본 논문에서는 ILP를 이용하여 WCET을 구하는 기존의 성능 분석 도구를 확장하여 ARM 기반의 내장형 소프트웨어를 분석할 수 있도록 하였다.

2.2 실시간 내장형 소프트웨어 성능 분석 도구

Cinderella[1]는 ILP 기반의 내장형 소프트웨어 성능 분석 도구이다. 이것은 특정 프로세서에서 프로그램의 수행 시간을 예측할 때 실제 수행 시간에 최대한 근접한 영역(최소값, 최대값)을 찾을 수 있게 한다. 프로그램 수행 시간의 최소값과 최대값을 T_{min}과 T_{max}라고 했을 때 프로그램의 실제 수행 시간 영역(actual bound)은 [T_{min}, T_{max}]사이에 존재한다. 그리고 프로그램 수행 시간을 예측할 수 있는 최적 조건(best case)과 최악조건(worst case)을 t_{min}, t_{max}라고 했을 때, 프로그램의 예측 수행 시간 영역(estimated bound)은 그림 1과 같이 [t_{min}, t_{max}] 사이이다. 그림 1에서 볼 수 있는 것처럼 실제 수행 시간은 예측 수행 시간 영역에 반드시 들어가야 한다.

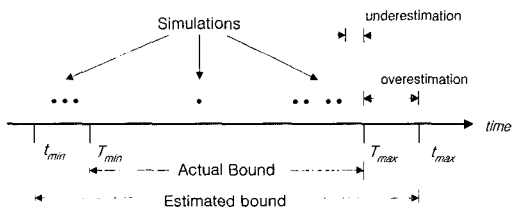


그림 1 실제 수행 시간 영역과 예측 수행 시간 영역

예측 수행 시간 영역을 결정하기 위해 즉, WCET를 구하는 문제를 해결하기 위해서 이 도구는 두 가지를 고려한다. 첫째는 프로그램 흐름을 분석하는 것이고, 둘째는 마이크로 아키텍처를 모델링하는 것이다. 프로그램 흐름 분석은 그림 2와 같이 구조적 제약(structural constraints)과 기능적 제약(functionality constraints)

이라고 정의되는 두 가지 형태의 선형 제약(linear constraints)을 이용하여 프로그램 흐름을 표현한다. 그림 2에서 각각의 노드와 지시선에 부여된 레이블 중 b_i는 기본 블록(basic block)을 의미하고 d_i는 지시선의 흐름빈도, x_i는 기본 블록의 수행빈도(execution count)를 각각 나타낸다. 구조적 제약은 각각의 노드들에 대한 흐름 제약을 기반으로 구성되는 CFG(Control Flow Graph)로부터 자동적으로 추출되어질 수 있다. 예를 들면, 그림 2에서 기본 블록의 수행 빈도 x_1은 제어가 해당 노드로 진입한 횟수 d_1과 같고, 이것은 또한 제어가 그 노드에서 벗어난 횟수 d_2와 같다. 즉 $x_1 = d_1 = d_2$라는 의미이다. 기능적 제약은 데이타 흐름(data flow) 분석을 수행하거나, 사용자가 직접 제약사항을 정의함으로써 얻을 수 있다.

기능적 제약은 다음과 같이 두 가지 형태의 논리적인 흐름(logical flow) 정보를 제공해 준다. 첫째, 프로그램 내에 존재하는 순환 영역(loop bounds)을 결정한다. 예를 들어, 그림 2의 소스 코드를 보면, 매번 while 루프에 들어갈 때마다 루프 내의 코드는 최소 0번, 최대 10번 수행되도록 되어 있다. 이러한 정보는 $0x_1 \le x_3 \le 10x_1$과 같이 표현할 수 있다. 둘째, 분석결과를 좀더 정확하게 산출할 수 있는 또 다른 경로 정보를 정의한다. 예를 들어, 그림 2에서 기본 블록 b_4와 b_5의 상관관계를 관찰해보면, else 문장은 루프안에서 최대 1번만 수행되도록 되어 있다. 이러한 정보는 $(x_4 = 0 \& x_5 = 1) | (x_4 = 1 \& x_5 = 0)$ 같이 표현할 수 있다.

임의의 프로그램의 기본 블록 b_i의 수행 시간이 상수 c_i라 하고, x_i가 해당 기본 블록의 실행 빈도라고 한다면, 이 프로그램의 전체 수행 시간은 '$\sum c_i x_i$'와 같이 표현될 수 있다. 앞에서 기술한 바와 같이 x_i와 관련된 모든 제약사항은 정수형 선형 공식들(Integer Linear Formulas)로 표현되기 때문에, 앞에서 추출된 제약들을 기반으로 ILP 기술을 이용하여 전체 수행 시간을 계산하는 함수(cost function)를 최대화함으로써 WCET 문제를 해결할 수 있다.

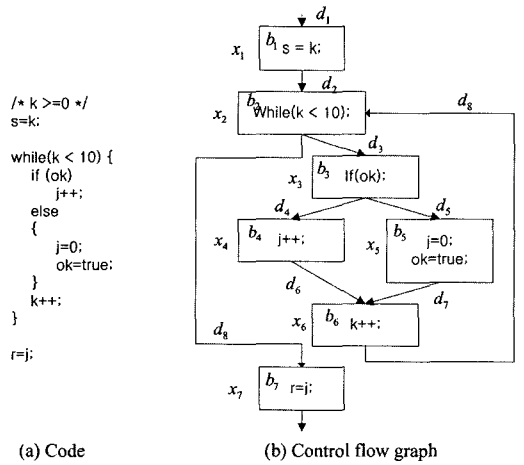


그림 2 C 소스코드와 CFG

최근의 마이크로 컴퓨터 구조에서 명령어 수행 시간은 피 연산자 값(operand value)과 시스템 상태(machine state)와 같은 요소들에 의해 영향을 많이 받는다. 마이크로 아키텍처 모델링은 프로그램이 수행 중에 동적으로 발생하는 이와 같은 영향을 고려하기 위해서 사용된다. 예를 들어, 알려져 있는 명령어들을 순차적으로 처리할 때 파이프라인 구조를 사용할 수 있다. 이러한 구조는 기본 블록의 실행 시간에 영향을 미친다. 추가적으로 캐쉬메모리의 상태도 마찬가지로 중요한 부분이라고 할 수 있다. "Cinderella"는 파이프라인 구조와 캐쉬 메모리의 상태 모델링을 마이크로 아키텍처 모델링에 추가하여 수행시간 분석의 정확도를 높였다.

3. ARM을 지원하기 위한 수행시간 분석 도구의 설계 및 구현

현재 내장형 시스템에 가장 많이 사용되는 ARM 프로세서는 저렴한 가격과 저 전력 소모란 장점을 가지고 있다. 특히, 모바일이나 정보 가전 애플리케이션에 최적화된 프로세서로서 이동 전화기 시장의 내장형 시스템에 가장 널리 사용되고 있다. 하지만 기존의 소프트웨어 성능 분석 도구는 ARM 프로세서를 지원하지 않는다. 본 논문에서 기반으로 하는 "Cinderella"는 프로세서에 종속적인 모듈과 프로세서 종류에 무관한 기본 모듈로 구성되어 있어서 확장성이 용이하다. 크게 프로세서에 독립적인 Cinderella 코어 부분, ILP Solver 부분, GUI (Graphical User Interface)부분과 프로세서 종류에 따른 목적 파일(Object file) 모듈 부분, 명령어 집합(Instruction set) 모듈 부분, Machine 모듈 부분 등으로 구성되어 있다. 본 논문에서는 ARM 기반의 내장형 소프트웨어의 수행 시간 분석이 가능하도록 그림 3과 같이 ARM-ELF 목적 파일 모듈, ARM7TDMI 명령어 집합 처리 모듈 그리고 하드웨어의 실행 시간을 계산할 수 있는 시간 속성들의 모델로 된 ARM-Machine 핸들러 모듈을 설계하고 구현하였다. ARM-Machine 모듈은 실제 하드웨어 보드를 모델링하기 위한 모듈로써 cpu의 특성과 버스 구조 같은 부분들을 모델링 하는 부분이다. 그러나, pipeline이나 cache에 대한 모델링은 되어 있지 않으며, 버스에 대한 구조도 모델링 되어 있지

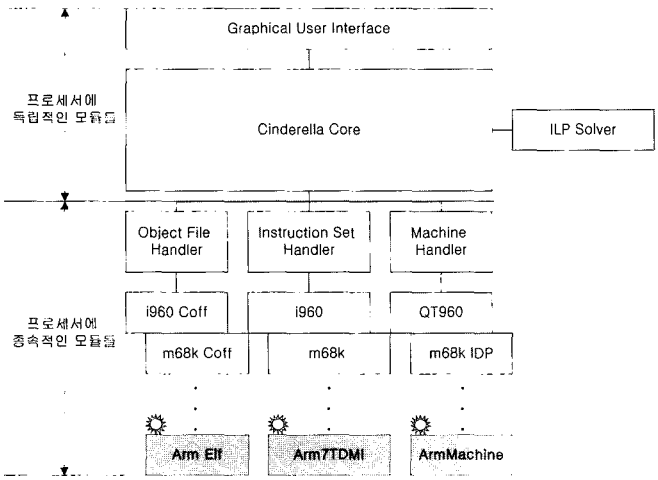


그림 3 ARM용 내장형 소프트웨어 수행 시간 분석을 위한 전체구조

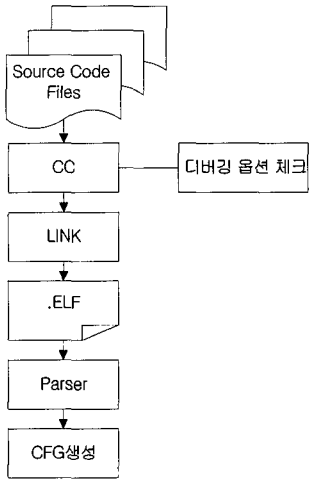


그림 4 전처리 과정

않고 다만 cpu의 명령어 사이클을 계산하기 위해 명령어 수행 사이클 테이블이 구현되어 있다.

3.1 ARM-ELF 목적 파일 처리 모듈

ARM 기반의 내장형 소프트웨어의 성능을 측정하기 위해서 처음 단계는 소스 코드 파일과 목적 코드를 포함하는 ELF 파일로부터 흐름 정보를 추출하는 것이다. 기존의 도구는 Unix 이진 실행 파일인 COFF를 사용하며, 구현되어 있는 모듈이 i960과 m68k를 위한 i960 COFF 이진 파일과 m68k COFF 이진 파일만 지원한다. 현재 ARM은 ELF 파일을 지원하기 때문에 성능 분석을 위해서는 ELF 파일을 해석할 수 있어야 한다. 또한 소스 프로그램을 컴파일 할 때 디버깅 옵션을 체크하여 그림 4와 같이 전처리 과정을 거쳐야 한다.

ELF는 DWARF(Debugging With Attribute Record Format) 라고 하는 디버깅 포맷을 가지고 있다[11]. 이 파일 포맷은 TIS(Tool Interface Standards)에서 디버깅에 관계된 표준을 제시한 것으로, DWARF는 컴파일러, 어셈블러, 링크 에디터에 의한 심볼들과 소스레벨 디버깅을 위해 필요한 정보 포맷을 제공한다. 디버깅 정보 포맷은 특정 컴파일러나 디버거들에 의존적이지 않는 표준을 따른다. DWARF의 목적은 이전 호환성을 유지하면서 다른 언어로 쉽게 확장할 수 있는 형태로, 어떤 디버거든 소스 프로그램의 정확한 모양을 전달할 수 있도록 정보를 제공한다. DWARF DIE(Debugging Information Entries)는 ELF파일에서 .debug 섹션을 형성한다. 고정된 크기의 작은 정보들 대신에 DWARF DIE들은 각각 임의의 길이를 갖는 복잡한 속성들을 포함하고 있으며, 영역별로 프로그램 데이타의 트리 구조로 쓰여져 있다. DWARF 정보들은 소스 파일의 라인 번호나 지역 심볼들의 정보를 가지고 있으며, 이 같은 디버깅 정보(debugging info)는 CFG를 생성하기 위해 필요한 정보를 제공한다.

CFG는 분기(조건 분기, 무조건 분기)명령과 함수 호출, 함수 반환과 같은 프로그램의 제어 흐름을 생성하기 위한 그래프이다. CFG를 생성하기 위해서는 프로그램 명령어가 어디에 존재하는지를 알아야 한다. 디버깅 섹션에서는 여러 가지 정보를 제공한다. 소스 프로그램의 각 함수들의 이름과 또 라인번호가 어떻게 되는지, 메모리 상에 로드되는 가상주소, 파일에서의 위치(offset), 크기 등 CFG를 생성하기 위한 모든 정보를 가지고 있다. 이와 관련된 부분은 .debug섹션들에서 추출할 수 있는데 .debug_info 섹션은 컴파일 유닛들에 대한 내용을 표현하는 섹션으로 offset/name의 형태로 정보들을 표현한다. 소스파일의 이름, 컴파일 디렉토리, 라인번호, 프로그램의 시작 주소와 끝나는 주소 등을 정보로 가지고 있다. 그림 5의 .debug_abbrev섹션은 .debug_info의 축약되어 있는 내용을 구체적으로 알기 위해 필요한 섹션이다.

3.2 ARM7TDMI 명령어 집합 처리 모듈

전처리 과정을 통하여 얻은 정보들 즉, 디버깅 정보를 이용하여 목적 파일에서 함수 이름들과 이 함수들을 조작할 수 있는 물리적인 주소를 가져올 수 있다. 다음 단계는 ELF 이진 파일을 분석하여 ARM 명령어들을 파싱할 수 있는 그림 6과 같은 ARM 명령어 처리 모듈이 필요하다. ARM 명령어를 파싱함에 있어서 가장 중요한 것은 제어흐름을 파악할 수 있는 분기 명령어들을 구분하는 일이다. ARM 명령어 처리 모듈은 ARM 명령어 포맷 중 Data processing, Single Data Transfer,

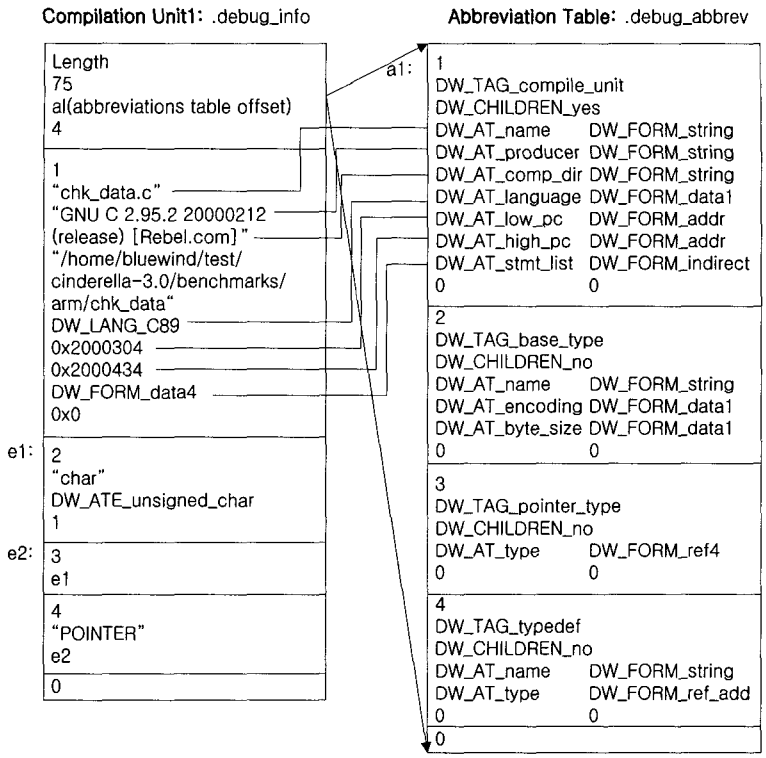


그림 5 DWARF의 .debug_info와 .debug_abbrev섹션의 구조

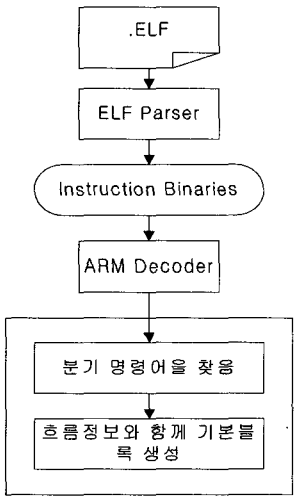


그림 6 ARM 명령어 처리 모듈

Block Data Transfer, Branch & Branch with Link 명령어들을 분석한다.

이중 Branch & Branch with Link 와 같은 명령어 포맷에서 무조건 분기와 조건 분기, 함수 호출, 함수 반환과 같은 제어흐름 명령어들의 주소 정보를 이용하여 CFG의 연결정보들로 표시한다. 나머지 명령어들은 CFG의 기본 블록들 속에 포함되는 것들이다. 그림 7은 C 소스 코드로부터 CFG를 생성하는 모습을 보여주고 있다. 그림 7(a)는 C 소스코드와 이를 ARM으로 컴파일 했을 때의 어셈블러 코드를 보여주고 있다. 그림 7(a)에서 보는 바와 같이 프로그램의 제어 흐름은 어셈블러 코드에서 분기 명령어로 표현되는 것을 알 수 있다. 그림 7(b)는 C 소스 코드에서 CFG를 생성하는 모습을 보여준다. 그림 7에서 보는 바와 같이 CFG를 생성하기 위해서 ARM 명령어 처리 모듈은 명령어를 파싱할때 어셈블러 코드에서 조건 분기(blt, bge, bne, etc.)와 무조건 분기(b) 명령어를 찾는다. CFG는 분기 명령어가 나오기 전까지의 명령어를 기본 블록으로 구분한다. 분기하는 주소는 연결 정보로 사용하여 프로그램의 제어흐름을 표현한다. 각 기본 블록의 수행 시간은 ARM 어셈블러 명령어들의 수행 사이클 테이블을 참고하여 계산한다.

4. 실험에서 사용된 ARM기반 내장형 소프트웨어의 성능 예측 방법

각 기본 블록들의 수행 시간을 3장에서 설계하고 구

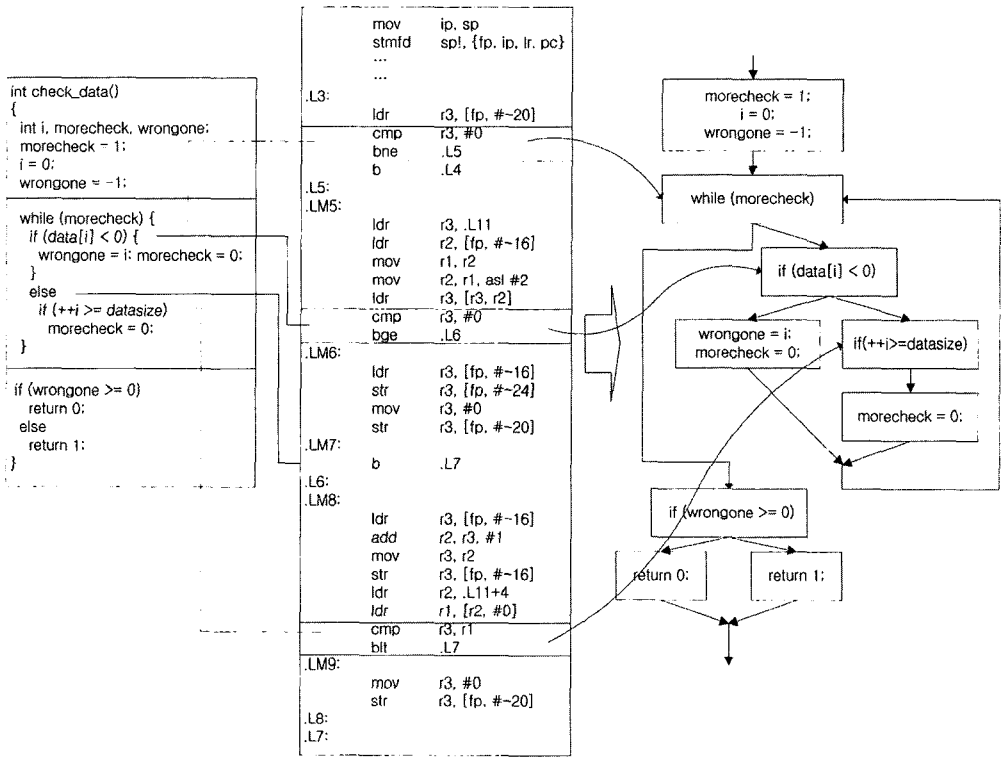
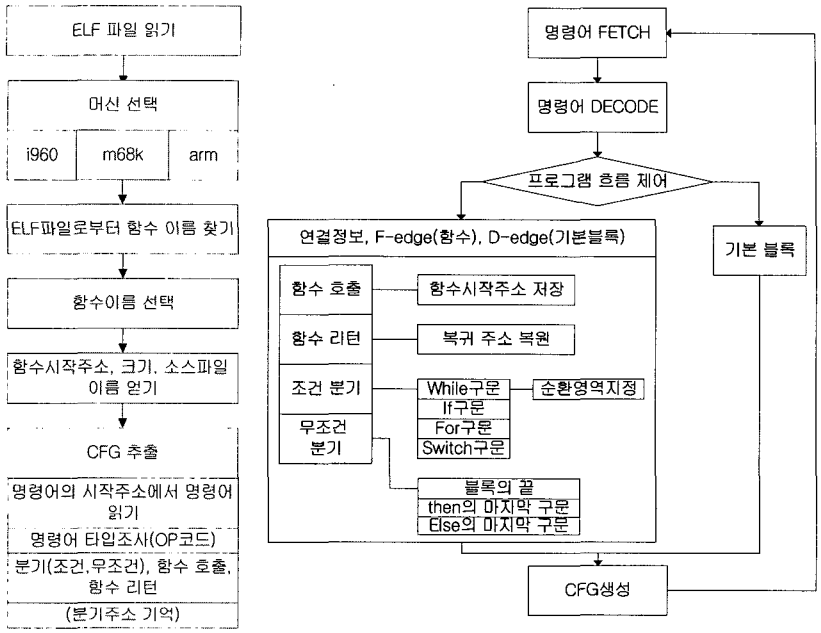


그림 7 C 소스 코드로부터 CFG의 생성

현된 모듈에서 계산하고, 프로그램의 연결정보에 의해 나열된 각 기본 블록들의 수행 시간을 축적함으로써 내장형 소프트웨어의 성능을 측정할 수 있다. 소프트웨어의 성능을 측정할 때, 고려할 사항으로는 프로그램의 연결정보이다. 또한 프로그램의 제어흐름 연결 정보 중 중요한 것이 반복 구간이 존재할 때이다. 이 같은 반복 구간은 프로그램 코드 상에서 몇 번 반복할 것인지 결정되는 경우도 있지만 특별한 조건이나 재귀 호출로 인하여 결정할 수 없는 경우도 있다. 이런 경우 사용자가 직접 최소값과 최대값을 입력함으로써 최적 및 최악 경우의 수행 시간을 구한다.

그림 8은 ARM 기반 내장형 소프트웨어의 성능분석을 위한 흐름도와 ARM 명령어 처리 모듈에서 CFG를 추출하는 알고리즘을 표현한 것이다. 그림 8(a)는 소프트웨어의 성능을 분석하기 위한 프로그램의 절차를 표현한 흐름도이다. 처음 ELF 실행 파일을 읽을 때 파일 포맷과 함수 이름들에 관한 정보를 읽어온다. 또한 하드웨어 플랫폼을 선택할 때 ARM을 지원하기 위한 명령어 집합 구조를 결정하고 하드웨어 모듈을 검색하게 된다. 하드웨어 모듈은 각 명령어에 대한 수행 시간과 마이크로 아키텍처에 관한 구조를 가지고 있다. 수행 시간을 측정하기 위한 함수 이름을 선택하면 CFG를 생성하며 for구문이나 while구문과 같은 반복 구간이 존재하는지를 찾게 된다. 만약 반복 구간이 있다면 사용자가 반복 구간의 횟수를 지정해줌으로써 수행 시간을 계산할 수 있도록 한다. 각 기본 블록들의 시간은 명령어 집합과 머신을 참고한다. 또한 사용자의 입력을 통해 여러 가지 제약 조건들을 생성한다. 각 기본 블록과 제약조건은 ILP 문제로 변환되어 프로그램의 예측 수행 시간 즉 WCET을 구하게 된다. 그림 8(b)는 ARM 명령어들을 파싱해서 CFG를 추출하는 알고리즘을 표현한 것이다. CFG를 추출하기 위해서, 이진 스트림을 읽은 후 명령어를 페치(fetch)하게 된다. 그런 연후 명령어를 디코딩(decoding)하는데, 명령어가 프로그램의 제어 흐름 명령어인지 아닌지에 따라 기본 블록과 연결정보로 표현된다. 연결정보들은 F-edge라는 함수의 연결정보와 D-edge라는 기본 블록의 연결정보로 나눌 수 있다. F-edge는 프로그램에서 함수의 흐름 정보를 표시하기 위한 것으로 함수 호출, 함수 리턴에 관계된 주소들을 나타낸다. D-edge는 각 기본 블록들의 연결을 표시하기 위한 것으로 while, if, for 그리고 switch 같은 조건 분기 명령어들과 무조건 분기 같은 명령어들의 주소들을



(a) 전체흐름도 (b) CFG 추출 알고리즘

그림 8 ARM 기반의 소프트웨어 성능분석을 위한 전체흐름도와 CFG추출 알고리즘

나타낸다. 이처럼 함수 호출과 분기 구문이 발생하기 이전까지의 명령어들을 기본 블록으로 표시하고 함수 호출, 함수 리턴, 조건 분기와 무조건 분기 명령어들은 연결정보로 표시함으로써 CFG를 생성한다.

생성된 CFG로부터 2장에서 설명한 구조적 제약 조건들을 추출할 수 있다. 이 같은 정보들은 소프트웨어의 수행 시간을 분석하기 위하여 정수형 선형 공식들로 변환되어 ILP Solver의 입력 변수로 들어간다. 마찬가지로, 2장에서 설명한 사용자가 입력하는 기능적 제약 조건들도 ILP Solver의 입력 변수이다. 수행 시간 분석기인 ILP Solver는 앞에서 설명한 입력 변수들을 통하여 소프트웨어의 전체 수행 시간을 계산한다.

5. 실 험

실험에 앞서 개발 환경에 관하여 알아보면 기존의 성능 분석 도구는 UNIX 환경에 적합하게 작성되어 있으며, Tcl/Tk를 이용하여 사용자 인터페이스를 구성하였다. 또한 이 도구는 i960 프로세서에서 구동되는 인텔 i960 프로그램과 M68000 프로세서에서 구동되는 모토롤라 m68k 프로그램을 지원한다. 본 논문에서 구현된 성능 분석 도구는 Visual C++ 6.0 환경에서 작성되었으며, 윈도우즈 환경의 사용자 인터페이스로 구성되었다. 또한 이 도구는 새로운 하드웨어 플랫폼인 ARM을 지원할 수 있도록 구현되었으며, ARM7TDMI 프로세서에서 구동되는 ARM 프로그램을 지원한다. 3장에서 살펴본 바와 같이 새로운 하드웨어 플랫폼을 지원하기 위해 하드웨어에 의존적인 여러 가지 모듈을 구현하였다. 즉, 목적 파일 모듈에서는 ELF를, 명령어 집합 모듈은 ARM7TDMI 프로세서의 명령어를 지원하도록 구현하였다.

5.1 실험 환경-벤치마크 프로그램

벤치 마크 프로그램들은 리눅스 환경에서 ARM 컴파일러 툴을 이용하여 컴파일 하였으며, 디버깅 정보를 함께 포함시키기 위해 컴파일 옵션으로 -ggdb를 사용하였다. 표 1은 벤치 마크 프로그램으로 사용된 프로그램들의 기능과 라인 수, ARM 컴파일러를 통하여 컴파일한 파일의 크기를 보여준다. 각각의 프로그램들은 기존의 성능 분석 도구에서 사용된 벤치마크 프로그램으로 기존의 여러 연구들에서 사용된 샘플 프로그램들이다. 여러 샘플 프로그램들 중, check_data는 [5]에서, sort는

표 1 벤치 마크 프로그램의 라인수와 컴파일된 파일의 크기

프로그램	설 명	Lines	Bytes
check_data	배열에 들어있는 요소들이 음수인지 체크하는 프로그램	102	13K
circle	원 그리는 루틴	200	21K
piksort	10개의 삽입정렬	103	13K
sort	500개의 버블정렬	123	14K

[8]에서, circle은 [12]에서 사용하는 벤치마크 프로그램들이다. 또한 각각의 프로그램들은 while이나 for 구문과 같은 반복 구간을 하나 또는 둘을 가지고 있다.

5.2 실험 결과

벤치마크 프로그램의 수행 시간을 구현된 도구와 비교하기 위해 ARM Developer Suite(ADS) 버전 1.2에서 ARMulator를 이용하여 각 함수들의 실행시간을 측정한 수행 사이클을 수집하였다. ARMulator는 명령어 집합 시뮬레이터(Instruction Set Simulator : ISS)로 다양한 ARM 프로세서의 명령어 집합과 아키텍처를 시뮬레이션 하여 수행 사이클을 계산 할 수 있다. 또한, ARMulator는 ARM 기반의 소프트웨어를 벤치마크 할 수 있으며 명령어 정확성(instruction-accurate)을 가지고 있다[13]. 물론 ARMulator는 100% 사이클 정확성을 가지지는 않지만 시스템을 설계하는데 있어서 소프트웨어의 개발에 적합하다고 볼 수 있다. 그러나 하드웨어의 모델에서는 반드시 100% 정확성을 만족해야 한다.

각각의 벤치마크 프로그램들은 반복 구간이 하나 이상 존재한다. 반복 구간에서는 반복 횟수를 최소 몇 번에서 최대 몇 번까지 반복할 것인지를 사용자가 직접 입력한다. 특히 정렬 프로그램들은 이중 for 구문으로 되어 있어서 두 번의 반복 구간을 갖는다. ARMulator에서 벤치마크 프로그램들을 실제 run-time시에 실행했을 때, 각 프로그램의 서브 함수들의 반복 구간 진입횟수를 계산한 결과 표 2에서와 같이 check_data 프로그램의 서브 함수인 check_data()는 10번, circle 프로그램의 서브 함수인 circle()은 0번이었다. 다중 반복 구문을 가지는 정렬 프로그램들(piksort, sort)의 경우에는 outer loop에서는 piksort 프로그램의 서브 함수인 piksort()가 9번, sort 프로그램의 서브 함수인 sort()는 250번이었다. 그리고 inner loop에서는 둘 다 반복횟수가 다양하였다.

표 2 ARMulator에서 run-time시 반복 구간 진입 횟수 결과

프로그램의 서브함수	반복구간	
	outer loop	inner loop
check_data()	10	
circle()	0	
piksort()	9	다양
sort()	250	다양

표 3은 ARM사에서 제공하는 ADS의 디버거에 내장되어 있는 수행 시간 측정도구인 ARMulator를 이용하여 각 함수들의 수행 사이클을 측정한 것과 본 논문을 통해 구현된 성능 분석 도구에서 각 함수들의 반복 구간의 반복 횟수를 입력하여 예측된 수행 결과 값을 보여준다. 실제 ARMulator에서 수행한 값은 프로그램 코드를 한단계 한단계 어셈블리 레벨에서 수행하여 사이클을 계산한 결과 값이다. 표 3에서 보는 바와 같이 check_data() 함수의 반복 구간을 최소 1에서 최대 10을 입력한 결과, 예측 수행 결과 값이 45사이클에서 495사이클이 수행됨을 보여준다. 실제 ARMulator에서 한 경우 반복 구간은 10으로 실행되었고 이 때의 수행사이클은 467사이클이었다. 나머지 프로그램도 이와 마찬가지로, 반복 구간의 반복 횟수를 사용자가 직접 입력함으로써 실제 소프트웨어의 수행 사이클(최소 사이클, 최대 사이클)을 계산한 결과, ARMulator에서 측정한 실제 수행 시간이 본 논문에서 구현한 정적인 수행시간 분석 도구의 반복 구간에 따른 값의 변화로 예측된 경계 값 안에 들어오는 것을 확인할 수 있었다.

ARMulator에서 얻어진 run-time시 반복 구간의 변화에 따른 수행 시간을 실제 수행 시간 영역(actual bound)이라고 간주했을 때, 실제 수행 시간 영역은 우리가 구현한 프로그램에서 즉, 정적으로 분석하고 예측되는 예측 수행 시간 영역(estimated bound)의 경계 안에 들어오는 것을 알 수 있다.

6. 결론 및 향후 연구과제

본 논문에서는 소프트웨어 성능 분석 도구인 "Cinderella"를 현재 가장 널리 사용되고 있는 내장형 프로세서인 ARM 머신을 지원 할 수 있도록 확장 설계하였다. 확장된 성능 분석 도구는 프로그램의 최적 및 최악 경우의 수행 시간을 정적으로(compile-time) 측정할 수 있다. 이 도구는 정적인 수행 시간을 측정할 수 있기 때문에 실제 run-time시의 수행 시간을 측정하는 것보다 훨씬 빨리 프로그램의 수행 시간을 측정할 수 있다는 장점이 있다. 또한 설계자가 시스템을 설계하는데 있어

표 3 ARMulator와 구현된 도구를 이용한 수행시간의 비교

프로그램의 서브함수	ARMulator (cycle)	반복구간(loop)		예측값(cycle)	
		outer	inner	lower	upper
check_data()	467	1~10		45	495
circle()	344	0~7		343	4219
piksort()	1509	9~9	0~9	503	4255
sort()	7560369	1~250	1~250	33	8168265

서 일련의 태스크들을 하드웨어와 소프트웨어로 분할할 때, 어떻게 분할할 것인가에 대해서도 도움을 줄 수 있다.

구현된 도구는 ARM용 소프트웨어의 성능을 측정하기 위하여 실제로 ELF 파일정보 뿐만 아니라 DWARF라는 디버깅 정보를 분석하고 적용하였으며, ARM 이진 실행 파일을 읽어 들여 실제로 ELF파일에서 CFG를 생성하며, 이를 위해 이진 파일을 파싱할 때, 각 명령어의 제어 흐름에 따라 기본 블록과 연결정보를 사용하여 표시한다. 또한 실제 사용자가 직접 정보를 제공함으로써 소프트웨어의 정확한 성능을 측정할 수 있게 한다. 각 기본 블록들은 ARM 명령어의 수행 사이클 테이블을 참고하여 계산된다. 프로그램의 여러 가지 제약 사항들은 정수형 선형 공식들로 변환되고 ILP 기술을 이용하여 전체 수행 시간을 계산하는 함수를 최대화함으로써 최적 및 최악 경우의 수행 시간의 정적인 분석을 가능하게 한다. 여러 가지 벤치마크 프로그램을 구현된 성능분석 도구와 ARMulator를 이용하여 비교한 결과, ARMulator로 측정된 실제 수행 시간이 구현된 성능분석 도구에서의 정적으로 하는 예측 수행 시간의 양 경계 값 안으로 들어오는 것을 확인할 수 있었다.

본 논문에서 설계된 ARM 머신을 지원하는 정적인 수행 시간 분석 도구는 ARM용 내장형 소프트웨어의 성능을 예측 및 측정할 수 있도록 한다. 즉 ARM 기반의 시스템 제약 사항들을 만족하는 하드웨어 및 소프트웨어 구성을 시스템 제작 전에 파악함으로써, 전체적인 내장형 시스템의 설계 시간 및 비용을 현저하게 줄여줄 것으로 기대된다. 향후 연구 과제로는 특정 ARM 보드를 선택하여 실제 하드웨어 플랫폼에서 적용하는 세부적인 마이크로 아키텍처 즉, 파이프라인 구조나 캐쉬 메모리 구조 등을 모델링 하고자 한다. 제안된 연구 과제는 실제 하드웨어에서 수행되는 시간을 모델링 함으로써 더욱더 정확한 성능을 예측할 수 있을 것으로 예상된다.

참 고 문 헌

[1] Y. S. Li and S. Malik, "Performance analysis of Real-Time Embedded Software," Kluwer Academic Publishers, 1999.

[2] E. Kligerman and A. D. Stoyenko, "Real-time Euclid: A language for reliable real-time systems," IEEE Transactions on Software Engineering, vol. SE-12, no. 9, pp. 941-949, September 1986.

[3] P. Puschner and Ch. Koza, "Calculating the maximum execution time of real-time programs," The Journal of Real-Time Systems, vol. 1, no. 2, pp. 160-176, September 1989.

[4] A. K. Mok, P. Amerasinghe, M. Chen, and K. Tantisirivat, "Evaluating tight execution time bounds of programs by annotations," in Proceedings of the 6th IEEE Workshop on Real-Time Operating Systems and Software, pp. 74-80, May 1989.

[5] C. Y. Park, "Predicting Deterministic Execution Times of Real-Time Programs," PhD thesis, University of Washington, August 1992.

[6] J. C. Liu and H. J. Lee, "Deterministic upper-bounds of the worst-case execution times of cached programs," in Proceedings of the 15th IEEE Real-Time Systems Symposium, pp. 182-191, December 1994.

[7] S. S. Lim, Y. H. Bae, G. T. Jang, B. D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Kim, "An accurate worst case timing analysis technique for RISC processors," in Proceedings of the 15th IEEE Real-Time Systems Symposium, pp. 97-108, December 1994.

[8] R. Arnold, F. Mueller, D. Whalley, and M. Harmon, "Bounding worst-case instruction cache performance," in Proceedings of the 15th IEEE Real-Time Systems Symposium, pp. 172-181, December 1994.

[9] J. Rawat, "Static analysis of cache performance for real-time programming, Master's thesis," Iowa State University of Science and Technology, November 1993.

[10] P. Puschner and A. Schedl. "Computing Maximum Task Execution Times with Linear Programming Techniques," Technical report, Technische Universitt, Institut fr Technische Informatik, Wien, Apr. 1995.

[11] TIS Committee, Debugging Information Format (DWARF) Specification Ver 2.0, May 1995.

[12] Rajesh Kumar Gupta, "Co-synthesis of Hardware and Software for Digital Embedded systems," PhD thesis, Stanford University, April 1991.

[13] http://www.arm.com, AXD and armsd Debuggers Guide(ARM DUI 0066) & ARM Developer Suite Debug Target Guide(ARM DUI 0058D).

황 요 섭

2000년 조선대학교 전자계산학과(이학사). 2002년~현재 조선대학교 전자계산학과 석사과정 재학중. 관심분야는 내장형 시스템, 유비쿼터스 컴퓨팅, 모바일 컴퓨팅

안 성 용

1996년 조선대학교 전자계산학과(이학사). 1998년 조선대학교 전자계산학과(이학석사). 1999년~현재 조선대학교 전자계산학과 박사과정 재학중. 관심분야는 재구성 가능한 시스템, 하드웨어 소프트웨어 통합설계, 내장형 시스템

심 재 홍

1987년 서울대학교 자연과학대학 전산과학과(이학사). 1989년 아주대학교 공과대학 컴퓨터공학과(공학석사). 2001년 아주대학교 정보통신대학 컴퓨터공학과(공학박사). 1987년~1994년 서울시스템(주) 공학연구소. 1999년~2000년 University of Arizona 객원연구원. 2001년 3월~2001년 9월 아주대학교 정보통신전문대학원 BK21 전임연구원. 2001년 10월~현재 조선대학교 인터넷소프트웨어공학부 조교수. 관심분야는 운영체제, 실시간 및 멀티미디어 시스템, 내장형 시스템

이 정 아

1982년 서울대학교 공과대학 컴퓨터공학과(공학사). 1985년 미국 인디애나 주립대학교 컴퓨터학과(공학석사). 1990년 미국 가주주립대학(UCLA) 컴퓨터공학과(공학박사). 1990년~1995년 Assistant Professor, University of Houston USA. 2000년 8월~2002년 2월 Stanford University 교환교수. 1995년~현재 조선대학교 컴퓨터공학부 교수. 관심분야는 재구성 가능한 시스템, 컴퓨터 연산, 컴퓨터 시스템