XScale 프로세서 기반의 임베디드 소프트웨어를 위한 최악실행시간 분석도구의 구현

박 현 희[†] · 최 명 수[††] · 양 승 민[†††] · 최 용 훈[††††] · 임 형 택[†††††]

요 약

신뢰성 있는 내장 실시간 시스템을 구축하기 위해서는 프로그램의 스케줄링 가능성 여부를 검증해야 한다. 스케줄링 가능성 분석을 위해서는 프로그램의 최악실행시간 정보가 필수적인 요소이다. 최악실행시간 분석은 두 단계로 나뉜다. 첫 번째 단계에서는 프로그램 언어 구문상의 흐름을 분석하고, 두 번째 단계에서는 수행되는 흐름 경로상의 하드웨어적인 요소를 고려하여 수행시간을 분석한다.

본 논문에서는 XScale 프로세서를 대상으로 하는 최악실행시간 통합 분석 도구인 WATER(WCET Analysis Tool for Embedded Real-time system)를 설계하고 구현한다. 상위 수준의 흐름 분석기와 하위 수준의 실행시간 분석기로 이루어진 WATER의 구조를 소개하고 소프트웨어의 실제 측정과 WATER의 분석 결과를 비교한다.

키워드 : XScale 프로세서, 내장 실시간 시스템, 최악실행시간, 최악실행시간 분석기

Implementation of Worst Case Execution Time Analysis Tool For Embedded Software based on XScale Processor

Park Hyeon Hui[†] · Choi Myeong Su[††] · Yang Seung Min[†††] · Choi Yong Hoon[††††] · Lim Hyung Taek[†††††]

ABSTRACT

Schedulability analysis is necessary to build reliable embedded real-time systems. For schedulability analysis, worst-case execution time(WCET) analysis that computes upper bounds of the execution times of tasks, is required indispensably. WCET analysis is done in two phases. The first phase is high-level analysis that analyzes control flow and finds longest paths of the program. The second phase is low-level analysis that computes execution cycles of basic blocks taking into account the hardware architecture.

In this thesis, we design and implement integrated WCET analysis tools. We develop the WCET analysis tools for XScale-based system called WATER(WCET Analysis Tool for Embedded Real-time system). WATER consist of high-level flow analyzer and low-level execution time analyzer. Also, We compare real measurement for execution of program with analysis result calculated by WATER.

Key Words : XScale Processor, Embedded Real-time System, Worst Case Execution Time, WCET Analysis Tool

1. 서 론

내장 시스템 중에서 논리적 정확성뿐만 아니라 마감 시간 내에 수행되어야 하는 시간적 정확성까지 포함되어야 하는 특성을 갖는 시스템을 내장 실시간 시스템(embedded real-time system)이라고 한다. 제한된 자원을 가진 내장 실시간 시스템에서 신뢰성 있는 시스템을 구축하기 위해서는 실시간 스케줄링 연구와 그 스케줄링 정책이 적합한지 판단하는 스케줄링 분석(schedulability analysis)이 필요하다. 스케줄링 분석을 위해 각 태스크들에 대한 최악실행시간(Worst-Case Execution Time, WCET) 분석은 중요하다. 최악실행시간 분석이란 태스크가 실행될 수 있는 여러 경로 중에서 실행시간이 가장 긴 경우를 찾는 것이다.

최악실행시간을 분석하는 여러 기법과 도구들이 존재하나, 이러한 기법과 도구들은 프로그램 코드에 대한 최악실행 흐름을 분석하는 상위 수준의 최악실행시간 연구이거나, 하드웨어 최악실행시간에 영향을 끼치는 요소들만을 고려한 하위 수준의 연구들이 대부분이다. 그러나 정밀한 최악실행시간을 분석하기 위해서는 상위 수준과 하위 수준 모두를 고려할 수 있는 분석도구가 필요하다.

※ 본 연구는 한국전자통신연구원 위탁연구과제 지원으로 수행되었습니다. (과제번호: 3010-2005-0065)
† 준 회 원 : 숭실대학교 대학원 컴퓨터학과 박사과정
†† 정 회 원 : 숭실대학교 대학원 컴퓨터학과 석사과정
††† 정 회 원 : 숭실대학교 컴퓨터학부 교수 겸 (주)엠스톤 대표이사
†††† 정 회 원 : 한국전자통신연구원 S/W개발도구연구팀 연구원
††††† 정 회 원 : 한국전자통신연구원 S/W개발도구연구팀 선임연구원
논문접수 : 2005년 4월 8일, 심사완료 : 2005년 8월 9일

본 논문에서는 상위 수준의 프로그램 코드에 대한 최악실행 흐름분석과 더불어 하위 수준의 하드웨어 영향까지 분석할 수 있는 최악실행시간 분석도구인 WATER(WCET Analysis Tool for Embedded Real-time system)를 설계하고 구현한다. WATER는 프로그램의 흐름을 분석하는 상위 수준의 흐름 분석기와, 하드웨어적인 요소(캐시, 파이프라인 등)를 고려하여 실행 사이클을 계산하는 기능을 가지는 하위 수준의 실행시간 분석기로 구성된다. 흐름 분석기는 고 수준 언어로 작성된 프로그램의 제어 흐름을 분석하며 여기서 생성된 기본블록 정보와 흐름 정보를 실행시간 분석기에 입력으로 제공한다.

WATER는 기존의 상위 수준 흐름 분석과 하위 수준의 하드웨어 요소를 고려한 분석기법들을 적용할 수 있는 확장성을 지니고 있으며, 각 분석기는 독립적으로도 동작할 수 있는 구조를 지니고 있는 것이 특징이다.

WATER는 내장 시스템용에서 많이 쓰이는 XScale 기반의 PXA255 프로세서를 사용하는 내장 실시간 시스템을 대상으로 하며, WATER의 사용자 인터페이스는 통합 개발환경인 이클립스의 플러그인 형태로 제공한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존에 연구된 최악실행시간 분석 방법에 대해 알아보고, XScale 프로세서의 구조를 소개한다. 3장에서는 WATER의 구조와 흐름 분석기, 실행시간 분석기의 구현 메커니즘을 소개한다. 4장에서는 WATER의 측정 평가에 대해 언급하고 5장에서는 결론 및 향후 연구방향을 제시한다.

2. 관련 연구

2.1 최악실행시간 분석 및 도구

최악실행시간 분석은 주어진 프로그램의 코드에 대하여 예상되는 실행시간의 상위 한계 값을 계산하는 것이며, 코드의 실행시간은 프로세서가 코드를 실행하는데 걸리는 시간으로써 정의한다[1].

최악실행시간 분석을 정의하는데 있어서 몇 가지 고려할 사항이 있다. 첫째, 코드의 실행시간은 사용자의 입력으로 받을 수 있는 파라미터 값에 따라 달라질 수 있다. 둘째, 최악실행시간 분석은 프로세서가 코드를 실제 실행하는 동안을 평가한다. 최악실행시간 결과 값은 프로세스의 선점과 블로킹, 그 외의 인터럽트에 의해 지연되는 시간은 포함하지 않는다. 마지막으로 최악실행시간 분석은 하드웨어 의존적이다. 최악실행시간 분석은 분석하고자 대상이 되는 하드웨어의 특징을 고려해야 한다.

최악실행시간 분석 기법들은 Real-Time Euclid[2, 3]와 Timing Schema[5, 6, 7, 8], TAL(Timing Analysis Language)[9, 10, 11], Timing Tree[12, 13, 14, 15]등이 있다. 또한 최악실행시간 분석에서 ILP(integer linear programming)기법을 사용한 연구도 존재한다[16, 17].

이러한 최악실행시간 분석 기법들은 상위 수준 언어의 흐름 분석에 대한 연구 또는 하위 수준의 하드웨어와 관련되어 있는 최악실행시간에 영향을 끼치는 요소들에 대한 연구로 나뉘어 있다. 즉, 상위 수준의 흐름 분석에 대한 연구에서는 하위 수준의 하드웨어의 특성에 의한 지연시간에 대한 영향을 고려하지 않고 있으며, 하위 수준의 하드웨어에 의존적인 실행시간 연구들은 코드 실행흐름을 전체적으로 파악하지 못하는 단점이 존재한다.

2.2 XScale 프로세서

XScale 프로세서는 INTEL사의 StrongARM을 대체하는 ARM5TE 코어에 기반을 두는 32비트 프로세서이다[18]. XScale 프로세서는 PXA255의 경우에는 200Mhz에서 400Mhz의 스피드로 동작하며, 가장 최근에 나온 PXA27x는 최대 624Mhz로 동작이 가능하다. 이 논문에서는 현재 가장 널리 사용되고 있는 PXA255를 대상으로 한다.

2.2.1 캐시

최악실행시간 분석에 있어서 캐시는 큰 영향을 미친다. 캐시 적중(hit) 또는 적중 실패(miss)에 따라서 수십에서 수백 사이클까지 차이가 발생할 수 있다. XScale 프로세서는 각각 32kbyte의 명령어와 데이터 캐시를 가지고 있으며, 32-set 연관 캐시(associative cache)이다.

2.2.2 파이프라인

XScale 프로세서는 슈퍼파이프라인(superpipeline) 구조를 가지고 있다. XScale의 파이프라인 구조는 주 실행 파이프라인(main execution pipeline)과 메모리 파이프라인(memory pipeline), MAC 파이프라인(Multiply ACcumulate pipeline) 세부분으로 구성된다.

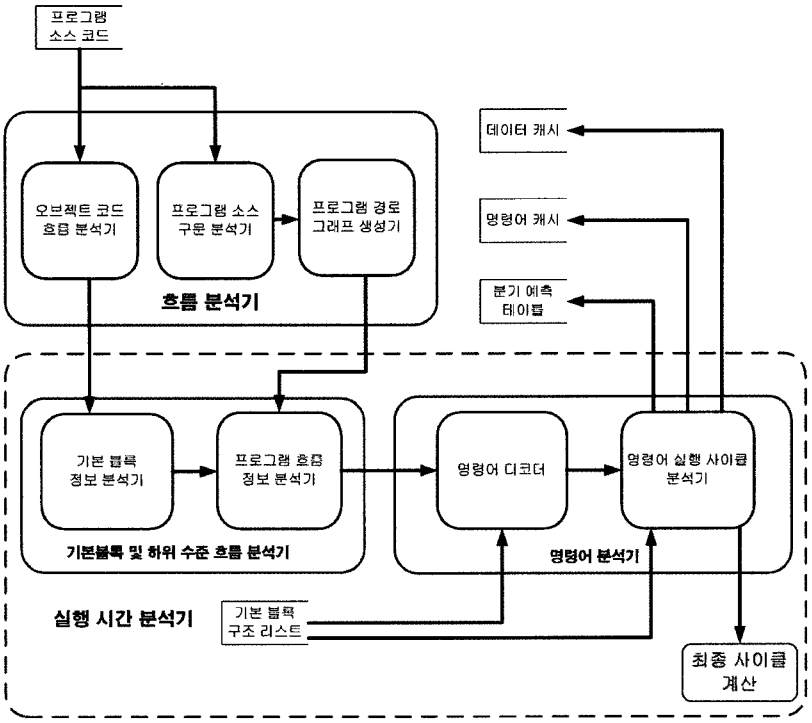
주 실행 파이프라인은 add, sub, mov와 같은 산술 및 기본 명령을 처리하고, 메모리 파이프라인은 ldr, str와 같은 메모리 접근 명령을 처리한다. MAC 파이프라인은 곱셈 명령에 대한 명령을 처리한다.

2.2.3 분기 예측

분기 명령어가 나왔을 경우, 파이프라인으로 가져올 명령어를 판단하는 것을 분기 예측이라 한다. 이 판단 기준은 분기 예측 버퍼(branch target buffer)의 결과에 따른다. XScale의 분기 예측 버퍼는 분기 명령어의 주소와 분기 대상 주소, 히스토리 비트를 저장할 수 있는 128개의 직접사상(direct-mapped) 캐시이다.

3. WATER(WCET Analysis Tool for Embedded Real-time system)

WATER는 최악실행시간 분석 시에 다음과 같은 전제를 둔다. 첫째, 내장 리눅스 시스템에서 널리 사용되는 GNU C/C++ 3.4 버전의 컴파일러에 의해 생성된 프로그램을 대상으로 하고, 컴파일러의 소스코드는 수정하지 않는다. 둘째, XScale 프로세서를 대상으로 분석하되, XScale의 캐시와 파



(그림 1) WATER의 전체 구성도

이프라인, 분기예측과 같은 프로세서 특성을 고려하나 캐시 분석의 경우, 캐시 락과 같은 특정 조건에 대해서는 고려하지 않는다. 셋째, 정수 유닛이 두개 있어서 out of order execution으로 인한 timing anomaly[19] 현상이 발생하는 PowerPC와는 달리, XScale은 각 기능을 가지는 유닛이 하나씩 존재하므로 이런 현상이 나타나지 않는다. 따라서 WATER에서는 timing anomaly 현상을 고려하지 않는다. 넷째, 흐름 분석의 반복 횟수 측정은 반복 횟수가 상수 값으로 명시된 것만을 고려하며, 변수 값의 변화로 분기 횟수가 가변적인 것은 고려하지 않는다.

3.1 구성

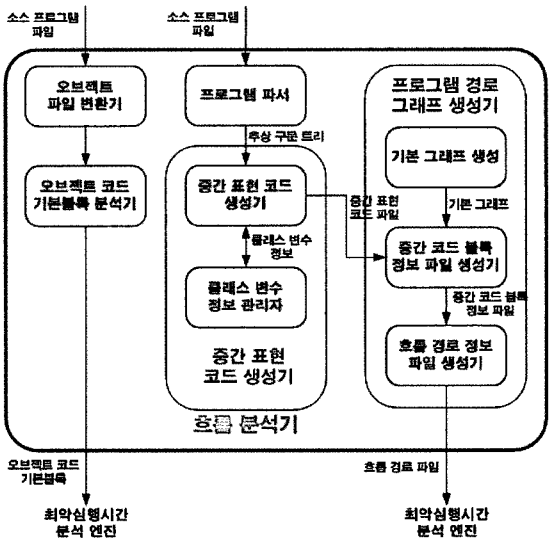
WATER는 XScale 프로세서를 대상으로 하며 스케줄링 분석기와 연동이 가능하다. 또한 C++로 작성된 소스코드를 분석하여 기본블록의 흐름 그래프를 생성한다. 그리고 이 정보를 바탕으로 최악실행시간을 계산하는 최악실행시간 분석 도구이다. 다음 (그림 1)은 WATER의 전체 구성도이다.

WATER는 크게 두 부분으로, 흐름 분석기(경로 흐름 변환)와 실행시간 분석기(기본블록 및 하위 수준 흐름 분석과 명령어 분석)로 구성된다. 흐름 분석기는 소스 프로그램으로부터 구문 트리를 생성하고, 이 트리를 기본으로 상위 수준 기본블록을 생성하며, 이를 기반으로 프로그램 경로 그래프를 생성한다. 또한 소스 코드를 오브젝트 코드로 변환 후 하위 수준 기본블록으로 나눈다. 이후, 프로그램 경로 그래프와 하위 수준 기본블록을 실행시간 분석기로 전달한다.

실행시간 분석기는 흐름 분석기가 생성한 기본블록과 흐름 정보를 입력으로 받은 후 명령어를 분석한다. 명령어 분석 시 계산되는 최악실행시간은 하드웨어 특성(캐시, 파이프라인, 분기 예측)을 고려한다. 분석결과로서 표시되는 최악실행시간의 단위는 XScale 프로세서 실행 사이클의 개수이다.

3.2 흐름 분석기

흐름 분석기는 프로그램 소스 코드와 코드의 바이너리 또는 오브젝트 파일을 입력으로 받아 오브젝트 코드 기본블록 파일과 기본블록간의 흐름 정보 파일을 생성한다. (그림 2)는 흐름 분석기의 구성을 보여준다.



(그림 2) 흐름 분석기 전체 구성도

흐름 분석기는 크게 세 부분으로 오브젝트 코드 흐름 분석기와 프로그램 코드 흐름 분석기, 프로그램 경로 그래프 생성기로 나타낼 수 있다. 오브젝트 코드 흐름 분석기는 프로그램의 바이너리 또는 오브젝트 파일로부터 오브젝트 코드 기본블록 파일을 생성한다. 프로그램 코드 흐름 분석기는 프로그램으로부터 중간 표현 코드 파일을 생성한다. 프로그램 경로 그래프 생성기에서는 중간 표현 코드 파일을 입력받아 흐름 경로 파일을 생성한다.

오브젝트 코드 흐름 분석기 내부는 오브젝트 파일 변환기와 오브젝트 코드 기본블록 분석기로 구성되어 있다. 오브젝트 파일 변환기는 XScale용 GNU objdump 유틸리티를 사용하여, 오브젝트 정보파일을 생성한다. 오브젝트 코드 기본블록 분석기는 오브젝트 정보파일에서 오브젝트 코드 단위의 기본블록을 추출한다. 오브젝트 코드 기본블록을 나누는 기준은 (그림 3)과 같다.

프로그램 코드 분석기 내부는 프로그램 파서와 중간 표현 코드 생성기, 클래스 변수 정보 관리자로 구성되어 있다. 프로그램 파서는 프로그램 소스 코드를 입력으로 하여 추상 구문 트리를 생성한다. 중간 표현 코드 생성기는 생성된 추상 구문 트리를 이용하여 중간 표현 파일을 생성한다. 중간 표현 파일은 3-주소 코드(three-address code) 형태를 사용한다[20]. <표 1>은 3-주소 코드의 형식을 보여주고 있다. 클래스 변수 정보 관리자는 중간 표현 파일에서 나타나는 클래스나 변수에 대한 정보들을 관리한다.

(그림 4)는 추상 구문 트리로부터 중간 표현 코드를 생성하는 예를 나타낸다.

- 함수의 시작은 기본블록의 시작이 된다.
- 분기문의 다음 명령어는 기본블록의 시작이 된다.
- 함수 호출의 다음 명령어는 기본블록의 시작이 된다.
- 분기문에 의해 분기되는 명령어는 기본블록의 시작이 된다.

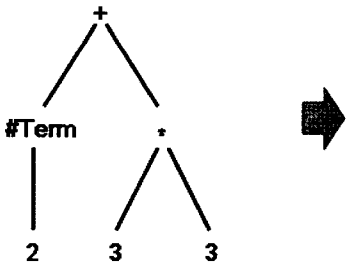
(그림 3) 기본블록을 나누는 기준

〈표 1〉 3-주소 코드의 형식

표기	의미
x = y *op* z	할당문 (*op* : 이진 산술 또는 논리 연산자)
x = *op* y	*op* : 단항연산자 (마이너스(-), 논리부정(!), 쉬프트 연산자)
x = y	복사 문장
goto L	무조건 분기
if x *relop* y goto L	조건 분기(*relop* : 논리 연산자)
call func	함수 호출
call class:method	클래스 메서드 호출

프로그램 경로 그래프 생성기 내부는 기본 그래프 생성기와 중간 코드 블록 파일 생성기, 흐름 경로 정보 파일 생성기로 구성되어 있다. 기본 그래프 생성기는 시작 노드와 끝 노드를 가진 기본 그래프를 생성한다. 중간 코드 블록 정보 파일 생성기는 중간 표현 코드 파일로부터 기본블록을 추출하여 기본 그래프에 연결시킨다. 중간 표현 코드 파일에서 기본블록을 나누는 기준은 오브젝트 코드에서 기본블록을 추출하는 기준과 같은 (그림 3)을 따른다. 흐름 경로 정보 파일 생성기는 기본블록의 정보로부터 경로 흐름 그래프를 생성하고, 분기문과 반복문을 고려하여 인접 노드 간에 명령어 수가 많은 노드들을 따라 흐름 경로 파일을 생성한다. 반복문은 반복의 기준이 되는 상수 값을 반복 횟수로 결정한다. (그림 5)는 중간 표현 코드 파일로부터 흐름 경로 파일을 생성하는 예를 보여준다.

기본블록에 대한 정확한 흐름 정보가 되기 위해서는 오브젝트 코드 기본블록과 중간 표현 코드 파일로부터 추출한 상위 수준 기본블록이 정확히 일치되어야 한다. 이를 위해 WATER는 3-주소코드 형태인 중간 표현 파일을 사용하며, 중간 표현 파일의 생성 시에 제어 흐름에 대한 규칙과 실제 컴파일러가 오브젝트 파일의 생성 시에 사용하는 제어 흐름의 규칙을 일치시킨다. 상위 수준 기본블록으로부터 생성한

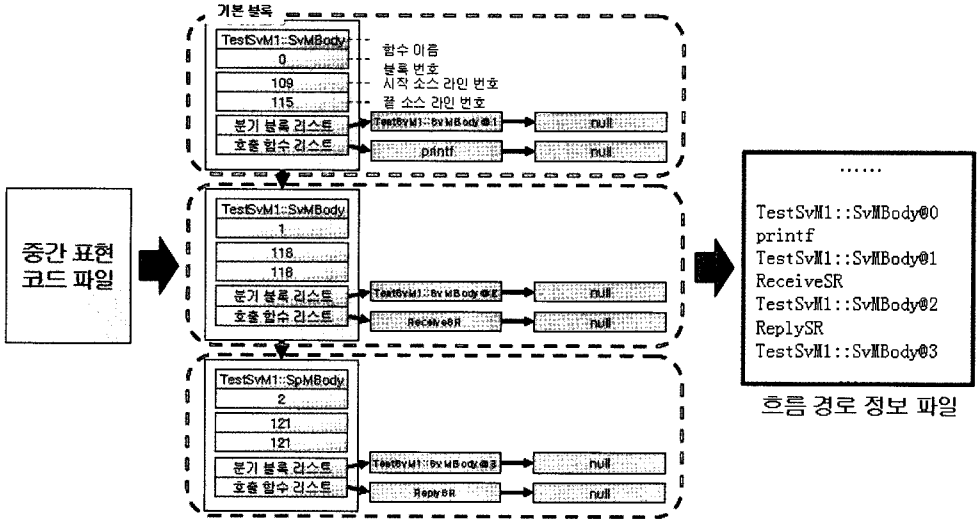


추상 구문 트리

```
19 $    TestSpM1::TestSpM1()   //19 46
19 $    {
24 $       gate_1 = gate1;
25 $       ODSS1 = odss1;
26 $       call build_regist_info_ODSS
27 $       call build_regist_info_SpM_name
31 $       _t3 = 1000*1000;
31 $       from *= _t3;
32 $       _t4 = 1000*1000;
32 $       until *= _t4;
41 $       call build_regist_info_AAC
46 $       call RegisterSpM
46 $       if _t6 != FAIL goto L10;
46 $    L9 :
46 $       call printf
46 $       goto L8;
46 $    L10 :
46 $       call printf
46 $    L8 :
46 $
19 $    }
```

중간 표현 코드

(그림 4) 중간 표현 코드 생성 예



(그림 5) 흐름 경로 정보 파일 생성 예

흐름 정보는 기본블록의 수행 순서를 나타낸다. 하위 수준 분석 시에 흐름 정보에 따라 오브젝트 코드 기본블록으로 수행시간을 측정하기 때문이다. 흐름 분석기로부터 생성된 오브젝트 코드 기본블록 파일과 흐름 정보 파일은 하위 수준 분석기의 입력으로 제공하여 수행 사이클을 분석한다.

3.3 실행시간 분석기

실행시간 분석기는 기본블록 정보 파일과 프로그램 흐름 정보 파일을 입력으로 받는다. 기본블록 정보 파일은 오브젝트 덤프 유틸리티로부터 생성된 덤프 파일에서 기본블록을 추출하고, 흐름 정보는 사용자가 제공하는 기본블록의 흐름 정보 파일로부터 받는다. 실행시간 분석기는 명령어 디코더와 흐름 분석 리더, 명령어 분석기로 구성되어 있다.

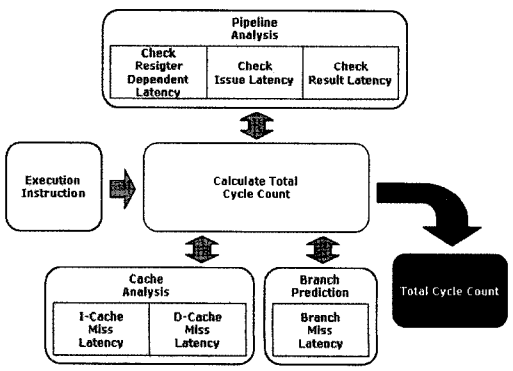
명령어 디코더는 캐시나 파이프라인, 분기 예측에서 명령어에 따른 분석을 적용하기 위해서 수행하며, 명령어의 타입과 피연산자(operand)를 분류한다. 분류의 기준은 ARM 계열 명령어 분류 기준에 따른다. 흐름 분석 리더는 흐름 정보 파일로부터 명령어 분석 단계에서 수행될 기본블록의 순차적인 정보를 읽는다. 명령어 분석기은 실제 명령어 사이클을 분석하는 단계로, 명령어/데이터 캐시와 파이프라인, 분기 예측을 고려하여 전체 사이클을 계산한다.

명령어 분석기에서는 캐시 분석을 위해 명령어와 데이터 캐시 시뮬레이션 방법을 사용한다. 이 방법은 기본블록의 명령어를 수행하면서 발생하는 캐시 적중과 캐시 미스를 측정하고, 캐시 미스가 발생 시에 메모리로 접근하는데 걸리는 패널티 사이클을 부여한다. 또한 명령어 분석기는 파이프라인 분석을 위해 명령어 디코더에서 분류한 명령어의 종류에 따라 수행하는데 걸리는 최소 사이클의 개수와 레지스터 의존성에 의해 발생할 수 있는 지연 사이클의 존재 여부를 판단, 명령어 사이클의 총 개수를 계산한다.

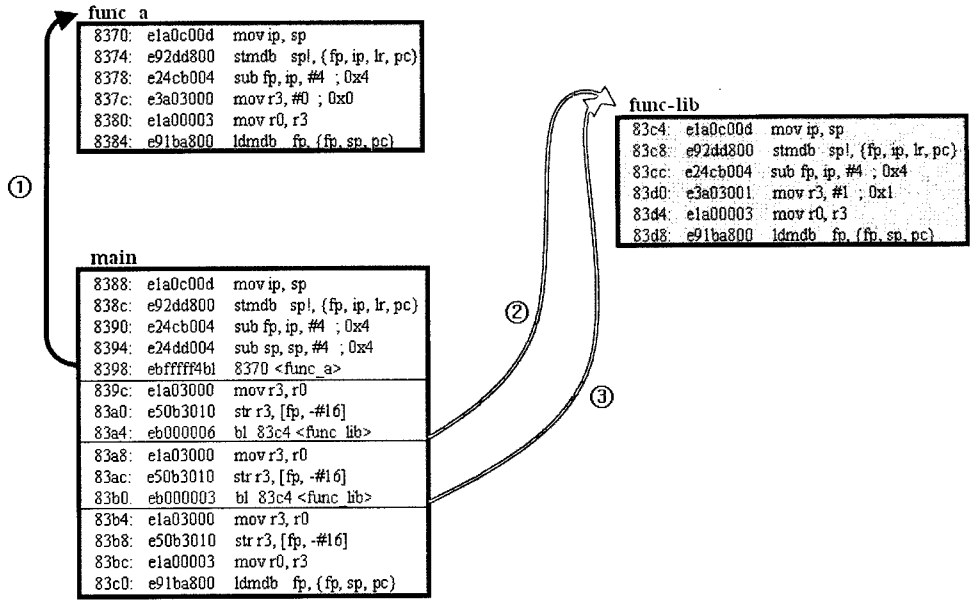
명령어 분석기는 분기 명령어가 발생했을 때를 위한 분기 예측 시, 분기 예측 시뮬레이터를 이용하여 분기 주소를 결정하고 그 주소를 다음 실행하는 기본블록의 첫 명령어 주소와 비교한다. 이 두 값이 같을 경우 분기 예측 성공이 되고, 다를 경우 분기 예측 실패가 된다. 분기 예측 실패가 발생하였을 경우, 패널티 사이클을 적용한다. 이 패널티 사이클은 기존에 실행된 명령어를 파이프라인에서 비우는데 걸리는 시간이다. (그림 6)은 실행시간 분석기의 수행과정을 나타낸다.

실행 시간 분석기는 흐름 분석기로부터 받은 기본블록 정보와 흐름 정보를 이용하여 제어 흐름을 파악하고 제어 흐름 내에 존재하는 명령어를 분석하기 위해 캐시와 파이프라인, 분기 예측 분석을 수행한다. 이를 통해 나온 결과 값이 XScale 프로세서의 사이클 개수로 표시되는, WATER가 분석한 최악 실행시간이다.

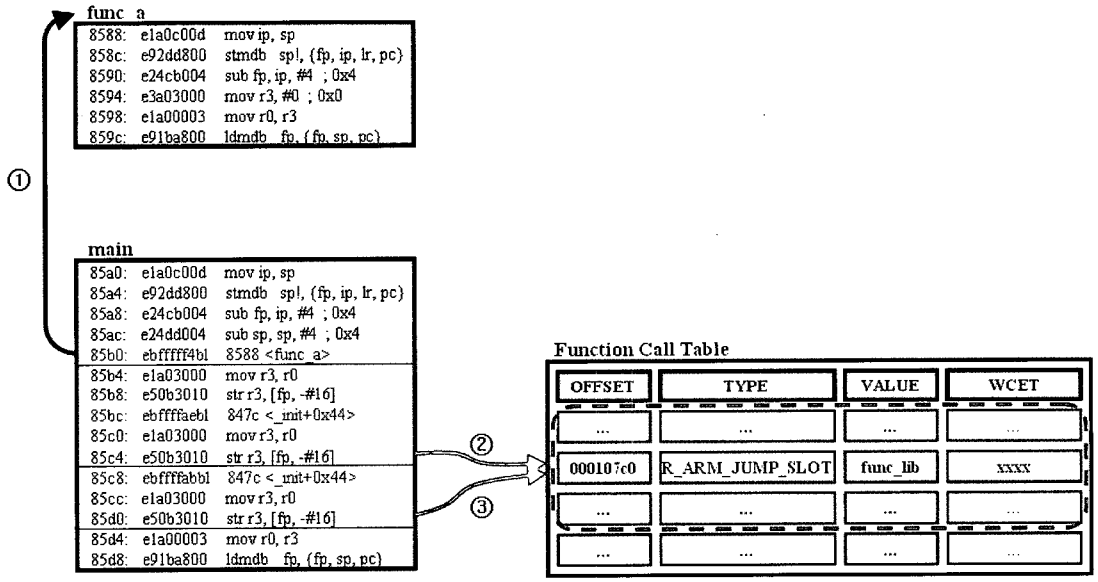
실행시간 분석기에서 분석 시에 제어문에 의한 분기가 아닌 함수 호출에 의한 분기의 경우, 일반적인 제어문의 분기와 동일하게 처리한다. 또한 프로그램 내에서 정의한 함수 호출이 아닌 라이브러리 함수가 호출되는 경우에도 실행시간 분석기에서 고려한다. WATER는 이런 종류의 함수를 분석할 때, 정적 방식뿐만 아니라 동적 방식의 라이브러리 함수까지도 고려한다.



(그림 6) 실행시간 분석기 수행 과정



(그림 7) 정적 라이브러리 호출



(그림 8) 동적 라이브러리 함수 호출 분석

3.3.1 정적 라이브러리 함수 호출시의 실행시간 분석

(그림 7)은 컴파일 시에 정적 라이브러리로 구성한 그림이다. ①은 코드 내부에 정의된 함수가 호출되는 경우이고, ②와 ③은 라이브러리 함수를 호출한 경우이다. 정적 라이브러리로 컴파일을 하면 생성된 바이너리 파일 안에 라이브러리의 함수 루틴이 포함되어 하나의 오브젝트로 생성된다.

실행시간 분석기는 프로그램의 오브젝트 코드로부터 기본블록 정보를 추출한다. 함수 호출 시에 기본블록 정보에서 함수를 찾아 분석하기 때문에 호출되는 함수의 코드 영역이 기본블록 정보 내에 존재해야 한다. 그러나 프로그램이 동적 라이브러리의 함수를 호출하는 경우에는 프로그램 내에 라이브러리 함수의 코드 루틴이 포함되지 않고, 실행 중에 라이브러리 함수 호출 시에 메모리에 적재하여 실행한다. 이런 이유로 분석 시 기본블록 정보 내에 동적 라이브러리 함수 루틴이 포함되지 않는다. 따라서 프로그램 코드 내에서 동적 라이브러리 함수가 호출되는 경우는 추가적인 분석이 이루어진다.

3.3.2 동적 라이브러리 함수 호출시의 분석

이러한 동적 라이브러리 함수 호출에 대한 추가적 분석을 위해 함수 호출 테이블을 사용한다. 이는 프로그램이 수행하기 전에 Object Relocation Table로부터 호출하는 동적 라이브러리 함수들을 찾아 라이브러리에서 이 함수들에 대한 최악실행시간 분석을 하는 방법이다. 분석된 결과 값을 함

수 호출 테이블에 저장한 후 프로그램 분석 시에 라이브러리 함수 호출 시에 테이블 만을 참조하여 그 함수에 대한 최악실행시간 결과 값을 반영하는 방법이다. (그림 8)은 라이브러리를 동적 라이브러리로 구성하여 컴파일하고, 함수 호출 테이블을 이용하는 방법을 표현한 것이다.

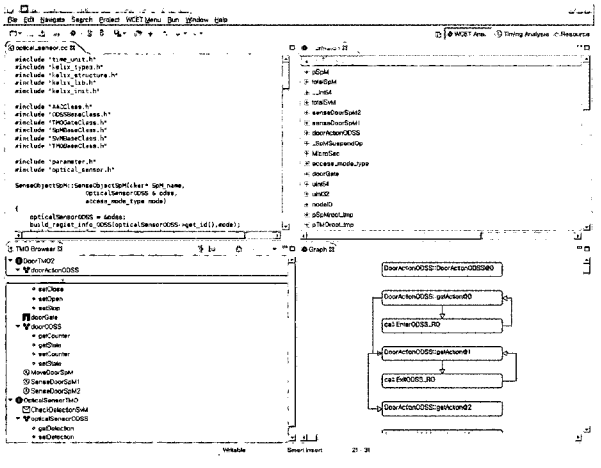
WATER에서 동적 라이브러리 함수를 분석하기 위해 프로그램이 수행되기 전에 수행될 동적 라이브러리 함수 정보를 미리 추출하고, 이에 대한 최악실행시간 값을 계산하여 테이블을 구성한다. 프로그램의 최악실행시간 분석단계에서는 이 테이블에 나타난 함수들의 최악실행시간 정보를 참조하여 분석을 수행한다.

4. 동작 환경 및 측정 평가

4.1 동작 환경

WATER는 INTEL XScale 내장 프로세서를 대상으로 하고, XScale용 GNU C/C++ 크로스 컴파일러에 의해 생성된 프로그램에 대하여 분석하며, INTEL x86 호환기종에서 동작한다. WATER는 다양한 플랫폼 상에서의 동작을 지원하기 위해서 JVM(java virtual machine) 상에서 동작하는 통합개발환경인 eclipse[21]의 plug-in으로 제공된다.

상위 수준의 흐름 분석기는 플러그인 내에 존재하기 위해 Java로, 하위 실행 시간 분석기의 구현은 GNU C 컴파일러를 사용한다. 흐름 분석기와 실행시간 분석기간의 연동을 위해, JNI(Java Native Interface)를 사용한다. (그림 9)는 WATER의 실행화면이다.



(그림 9) WATER의 실행화면

4.2 측정 평가 방법 및 제한

WATER의 측정을 평가하기 위한 테스트 프로그램은 동적 함수 호출 테이블을 사용하지 않는 정적 함수 호출을 호출하는 경우와 동적 함수 호출 테이블을 사용하는 동적 함수를 호출하는 경우로 나누어 분석하며, 동일한 테스트 프로그램의 코드에 대하여 최악실행시간 분석 도구로 분석한 결과와 XScale의 모니터링 Coprocessor를 사용하여 측정한 결과와의 비교한다.

```
void matrix(int array[20][20])
{

   for i = 0 to 20
      for j = 0 to 20
         array[i][j] += 2;

}

int main()
{
   int a[20][20];

   for i = 0 to 20
      for j = 0 to 20
         a[i][j] = j;

   matrix(a);
}
```

```
void quicksort(int *array)
{
   int p, tmp;

   for i = 0 to 20 {
      p = i;
      for j = i+1 to 20
         if (array[j] < array[p])
            p = j;

      tmp = array[i];
      array[i] = array[p];
      array[p] = tmp;
   }
}

int main()
{
   int a[20]

   for i = 0 to 20
      a[20-i-1] = i;

   quicksort(a);
}
```

(그림 10) matrix와 quicksort 코드

테스트 프로그램을 임의의 대상으로 하기에는 시스템 함수 호출이 있을 경우와 스케줄러에 의해 영향을 미치는 경우 등 여러 가지 분석하기 어려운 면이 많다. 프로그램 내의 시스템 호출은 사용자 영역에서 커널 영역으로 진입이 발생하므로, 커널 영역 루틴에 의한 분석의 어려움이 존재한다. 프로그램 수행 중에 스케줄러에 의해 태스크 전환이 발생하였을 경우 다른 태스크에 의한 코드 수행 결과가 수행시간 측정에 반영된다. 성능 평가를 위한 테스트 코드는 실행시간 측정을 확실하게 결정할 수 있는 matrix와 quicksort의 코드를 대상으로 수행한다.

XScale의 모니터링을 사용한 측정 방법은 XScale용 리눅스 커널 버전 2.4.18의 커널모듈의 형태로 테스트 프로그램 수행 코드를 삽입하여 측정한다. (그림 10)은 테스트를 위한 matrix와 quicksort를 보여준다.

4.3 평가 분석

<표 2>는 WATER에 의한 최악실행시간 분석 결과이며, <표 3>은 XScale 모니터링에 의한 프로그램 수행시간 값이다.

〈표 2〉 최악실행시간 분석 결과

프로그램 코드	최악실행시간 분석 결과 값 (Clock Cycle)
matrix	47280
matrix (동적 함수 사용)	48257
quicksort	21682
quicksort (동적 함수 사용)	21179

〈표 3〉 XScale 성능 모니터링 Coprocessor 측정 결과

프로그램	실행시간(Clock Cycle)	평균 실행시간(Clock Cycle)
matrix	41092	41038
	41164	
	40980	
	41068	
	40884	
quicksort	10595	10563
	10683	
	10539	
	10531	
	10467	

<표 4>는 그램 코드에 대한 분석 결과와 XScale 성능 모니터링에 의한 프로그램 실행 측정 결과에 대한 과대 측정 비율이다.

〈표 4〉 WATER의 과대측정 비율

프로그램	최악실행시간 분석기 분석 결과	XScale 측정 결과	과대측정비율 (%)
matrix (정적 호출)	47280	41038	15.21%
matrix (동적 호출)	48257		17.59%
quicksort (정적 호출)	21682	10563	105.26%
quicksort (동적 호출)	21179		100.50%

$$\text{과대측정비율} = \left(\frac{\text{분석기측정전체주기수}}{\text{실제측정전체주기수}} - 1\right) \times 100$$

<표 4>에 의하면, matrix의 과대측정비율은 15-17%에 반해, quicksort는 100-105%를 보여주고 있다. (그림 20)의 matrix의 경우, 중첩 반복 문에서 $n \times n = n^2$번 반복하는데, WATER 에서도 또한 n^2번의 횟수로 분석하기 때문에 근사한 값을 보여준다. 그러나 quicksort의 경우,$n+(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$번 반복하는데, WATER에서는 $n \times n = n^2$번을 반복한 것으로 인지한다. 이 때문에 과대측정비율이 약 2배의 결과로 나타나게 된 것이다. 이 부분은 코드 상에서 반복 횟수가 상수가 아닌 변수에 의해 결정되므로, 정확한 측정이 어렵다. 따라서 이 부분을 개선하기 위해서는 변수에 의해 결정되는 반복문의 경우, 사용자 입력에 의해 결정하는 부분이 필요하다.

5. 결론 및 향후 연구 과제

신뢰성 있는 내장 실시간 시스템 환경을 구축하기 위해서는 시스템에 대한 스케줄링 분석이 필수적이며 이를 위해 시스템 내에서 동작하는 태스크의 최악실행시간 분석 도구가 필요하다. 기존에는 대학 및 연구소를 중심으로 최악실행시간 분석 연구가 많이 이루어져 왔다. 이 연구들을 기반으로 구현된 대부분의 최악실행시간 분석 도구들은 Power PC나 Pentium 등 일부 프로세서를 대상으로 하고 있지만 XScale 프로세서를 대상으로 하고 있는 도구는 드물고 캐시와 파이프라인, 분기 예측 등 여러 하드웨어 영향을 고려한 분석 도구 또한 많지 않다.

WATER는 기존의 실행흐름 분석기법과 특정 하드웨어에 대한 최악실행시간 분석 기법 모두를 적용할 수 있는 확장성을 지니고 있으며, 각 분석기들은 독립적으로 동작이 가능한 구조를 가지고 있다.

향후 연구 과제로서 사용자 입력에 의해 결정하는 부분에 대한 기능을 추가하는 작업이 진행 중이다. 또한 제어 흐름 그래프에서 ILP를 사용하는 Timing Tree와 같은 신뢰성이 있는 분석 방법을 적용하고, 실행시간 분석기내에서도 캐시나 파이프라인의 영향에 대한 효율적인 기법을 도입하여 과대측정비율을 줄이도록 개선할 필요가 있다. 또한 변수 분석을 통해서 사용자가 결정해야 할 것인지 아닌지를 판단하며, 사용자가 변수의 값을 결정하면 그에 맞는 측정값이 재계산되도록 한다. 현재는 응용프로그램의 최악실행시간만을 분석하지만, 앞으로는 응용프로그램뿐만 아니라 커널코드의 분석도 필요하다.

참 고 문 헌

[1] P. Puschener and A. Burns, "A review of worst-case execution-time analysis, Real-Time Systems," Guest Editorial, 18(2-3):115- 128, May, 2000.

[2] Kligerman, E. and A. Stoyenkok, "Real-Time Euclid: A Language for Reliable Real-Time Systems." IEEE Transactions on Software Engineering SE-12(9), pp.941-949, 1986.

[3] Stoyenko, A., "A Real-Time Language With A Schedulability Analyzer," Computer Systems Research Institute, University of Toronto, Canada, Dissertation, 1997.

[4] Stoyenko, A., V. Hamacher, and R. Holt, "Analyzing Hard-Real-Time Programs for Guaranteed Schedulability," IEEE Transactions on Software Engineering SE-17(8), pp.737-750, 1991.

[5] Shaw, A. C., "Reasoning About Time in Higher-Level Language Software," IEEE Transactions on Software Engineering SE-15(7), pp.875-889, 1989.

[6] Park C. Y. and A. C., "Experiments with a Program Timing Tool Based on Source-Level Timing Schema," IEEE Computer 24(5), pp.48-57, 1991.

[7] Park C. Y., "Predicting Deterministic Execution Time of Real-Time Programs," Ph.D. thesis, Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA, 1992.

[8] Park C. Y., "Predicting Program Execution Times by Analyzing Static and Dynamic Program Paths," Real-Time Systems 5(1), pp.31-62, 1993.

[9] Mok, A. K., P. Amerasinghe, M. Chen, and K. Tantisirivat, "Evaluating Tight Execution Time Bounds of Programs by Annotations," In: Proc. 6th IEEE Workshop on Real-Time Operating Systems and Software, Pittsburgh, PA, USA, pp. 74-80, 1989.

[10] Amerasinghe, P., "A Universal Hardware Simulator," Dept. of Computer Sciences, University of Texas, Austin, TX, USA: Undergraduate Honors Thesis, 1985.

[11] Chen, M., "A Timing Analysis Language - (TAL)," Dept. of Computer Sciences, University of Texas, Austin, TX, USA: Programmer's Manual.

[12] Puschner, P. and C. Koza, "Calculating the Maximum Execution Time of Real-Time Programs," Real-Time Systems 1(2), pp.159-176, 1989.

[13] Puschner, P. and A. Schedl, "A Tool for the Computation of Worst Case Task Execution Times," In: Proc. Euromicro Workshop on Real-Time Systems, Oulu, Finland, pp. 224-229, 1993.

[14] Pospischil, G., P. Puschner, A. Vrchoticky, and R. Zainlinger, "Developing Real-Time Tasks with Predictable Timing," IEEE Software 9(5), 35-44.

[15] Vrchoticky, A., "Compilation Support for Fine-Grained Execution Time Analysis," In: Proceedings of the ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems, Orlando FL, 1994.

[16] Puschener, P. and A. Schedl., "Computing Maximum Task Execution Times - A Graph-Based Approach," Real-Time Systems 13(1), pp. 67-91, 1997.

[17] Puschner, P., "A Tool for High-Level language Analysis of Worst-Case Execution Times," In: Proc. Euromicro Workshop on Real-Time Systems, Berlin, Germany, pp. 130-137, 1998a.

[18] Intel, "XScale Microarchitecture for the PXA255 Processor User's Manual," March, 2003.

[19] Thomas Lundqvist, "A WCET Analysis Method for Pipelined Microprocessors with Cache Memories," Ph.D. thesis, Chalmers University of Technology, Goteborg, Sweden, 2002.

[20] Alfred V. aho, Ravi Sethi, Jeffrey D. Ullman, "Compilers - Principles, Techniques and Tools," Ch.8, pp.463-512, Addison-Wesley, 1988

[21] Eclipse project official site, http://www.eclipse.org.

박 현 희

e-mail : darklight@realtime.ssu.ac.kr
2000년 숭실대학교 컴퓨터학과(학사)
2003년 숭실대학교 컴퓨터학과(석사)
2004년~현재 숭실대학교 대학원 컴퓨터학과 박사과정
관심분야 : 실시간 시스템, 내장 시스템, 리눅스 커널, 운영체제

최 명 수

e-mail : mosu@realtime.ssu.ac.kr
2004년 숭실대학교 컴퓨터학과(학사)
2004년~현재 숭실대학교 대학원 컴퓨터학과 석사과정
관심분야 : 실시간 시스템, 운영체제, 최악 실행시간 분석

양 승 민

e-mail : yang@computing.ssu.ac.kr
1978년 서울대학교 공과대학 전자공학과 (학사)
1978년~1981년 삼성전자㈜ 연구원
1983년 미국 Univ. of South Florida 전산학(석사)
1986년 미국Univ. of South Florida 전산학(박사)
1987년 미국Univ. of South Florida 조교수
1988년~1993년 미국 Univ. of Texas at Arlington 조교수
1993년~현재 숭실대학교 컴퓨터학부 교수 겸 (주)엠스톤 대표이사
관심분야 : Real-Time System, Operating System, Fault-Tolerant System

최 용 훈

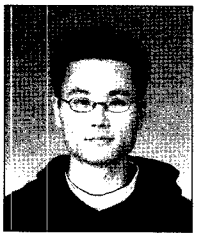
e-mail : yonghoon@etri.re.kr

2002년 고려대학교 컴퓨터학(학사)

2003년 카네기멜론대학교 소프트웨어공학(석사)

2004년~현재 한국전자통신연구원 임베디드 S/W연구단 S/W개발도구연구팀 연구원

관심분야 : 소프트웨어 아키텍쳐, 실시간 임베디드 시스템, 사용자 경험 연구



임 형 택

e-mail : htlim@etri.re.kr

1994년 숭실대학교 전자계산학과(학사)

1996년 숭실대학교 전자계산학과(석사)

2001년 숭실대학교 컴퓨터학과(박사)

2001~현재 한국전자통신연구원 임베디드 S/W연구단 S/W개발도구연구팀 선임연구원

관심분야 : 실시간 임베디드 시스템, 운영체제, 임베디드 시스템 개발 도구