실행정보를 적용한 최악실행시간 분석도구

문인철[0] 박현희 양승민
숭실대학교 컴퓨터학과
moonpfe@realtime.ssu.ac.kr, darklight@realtime.ssu.ac.kr, yang@computing.ssu.ac.kr

Worse Case Execution Time Analysis Tool Using The Run-Time Information

InChul Moon[0], HyunHee Park, SeungMin Yang
Department of Computing Graduate School Soongsil University

요 약

내장 실시간 시스템은 논리적 정확성과 시간적 정확성을 모두 만족해야 하는 내장 시스템이다. 시스템의 시간적 정확성을 위해서는 해당 시스템에서 동작하는 태스크들의 스케줄링 가능성을 검사해야 한다. 스케줄링 가능성을 분석하기 위해서는 태스크의 실행 시간 분석이 선행 되어야 한다. 하지만 태스크의 실행 시간은 실행 시점에 따라 가변적이기 때문에 태스크의 정확한 실행 시간을 알아내기는 힘들다. 따라서 가능한 모든 경우를 고려하여 해당 태스크를 구성하는 코드 경로 중 최악의 경로일 경우의 실행 시간인 최악 실행 시간을 이용한다. 기존의 정적 최악 실행 시간 분석을 하는 도구의 경우 동적인 상황의 정보 부재로 인해 최악 실행 시간의 과대 측정 비율이 높다는 문제점이 있다.

본 논문에서는 정적 최악 실행 시간 분석 시 과대 측정 비율을 줄이기 위해 대상 기기에 실행 정보를 적용한 RunInfo(WCET analysis tool using the Run-Time Information) 분석 도구를 설계하고 구현한다. 실행 정보를 정의하고, RunInfo 분석 도구의 구조에 대해 설명한다. 그리고 실행 정보를 적용할 때, 고려할 점에 대해 알아본다. 성능 평가를 위해 RunInfo 분석 도구의 과대 측정 비율을 기존의 분석도구와 비교한다.

1. 서 론

내장 시스템 중 논리적 정확성(logical correctness) 뿐만 아니라 시간적 정확성(temporal correctness)도 만족해야 하는 시스템을 내장 실시간 시스템(embedded real-time system)이라 한다. 제한된 자원을 가진 내장 실시간 시스템에서 신뢰성 있는 시스템을 구축하기 위해서는 실시간 스케줄링에 대한 연구와 그 스케줄링 정책이 적합한지를 판단하는 스케줄링 가능성 분석(schedulability)이 필요하다. 스케줄링 가능성이란 모든 태스크들의 마감시간(deadline)을 만족하는 스케줄링이 가능한가를 나타낸다.

스케줄링 가능성을 알기 위해서는 태스크의 실행 시간 분석이 필요하다. 하지만 태스크의 실행 시간은 실행 시점과 실행 환경에 따라 가변적이기 때문에 태스크의 정확한 실행 시간을 알아내기는 힘들다 따라서 가능한 모든 경우를 고려하여 해당 태스크를 구성하는 코드 경로 중 최악의 경로일 경우의 실행 시간을 이용한다 이 시간을 최악 실행 시간(worst case execution time)이라고 한다. 신뢰성 있는 내장 실시간 시스템을 구축하기 위해서는 다른 무엇보다 최악 실행 시간 분석이 선행 되어야 한다.

그 중 정적 최악 실행 시간 분석 도구는 해당 프로그램을 실행하지 않고 최악 실행 시간을 측정한다 모든 시스템의 입력과 프로그램과 하드웨어 사이의 상호 작용을 고려함으로써 프로그램을 실행하지 않고 최악 실행 시간을 측정할 수 있다 하지만 정적 분석 도구는 동적인 상황의 정보 부재로 인해 최악 실행 시간을 지나치게 과대 측정(overestimation)하여 프로세서의 자원을 낭비하는 문제점이 있다.

본 논문에서는 정적 최악 실행 시간 분석 시 과대 측정 비율을 줄이기 위해 대상 기기의 실행 정보를 적용한 RunInfo(WCET analysis tool using the Run-Time Information) 분석 도구를 설계하고 구현한다 실행 정보(run-time information)는 대상 기기의 하드웨어 상태 정보와 태스크의 실행 시 초기 상태 정보를 통합한 정보이다. 대상 기기로부터 수집한 실행 정보를 바탕으로 시스템의 동적인 상황에 대한 정보를 일정 부분 반영함으로써 과대 측정 비율을 줄인다 RunInfo 분석 도구는 내장 시스템에 많이 사용하는 인텔 XScale 코어를 분석 대상으로 하고, 운영체제는 리눅스로 한다 RunInfo 분석 도구의 사용자 인터페이스는 통합 개발 환경인 이클립스의 플러그인 형태로 제공한다

본 논문의 구성은 다음과 같다 2장에서는 정적 최악 실행 시간 분석에 대해 소개하고 기존의 정적 최악 실행 시간 분석 도구에 대해 알아본다 그리고 본 논문의 분석 대상이 되는 인텔 XScale 코어에 대해 알아본다 3장에서는 RunInfo 분석 도구의 구조와 실행 정보를 분석 도구에 적용하기 위한 방법을 알아본다 4장에서는 과대 측정 비율을 통해 도구의 성능을 평가하고 5장에서는 결론 및 향후 연구 방향을 제시한다

2. 관련 연구

2.1 최악 실행 시간 분석

최악 실행 시간 분석은 코드의 실행 시간 중 가장 최악의 경우 실행 시간의 상한선을 구하는 것이다. 코드의 실행 시간은 프로세서가 코드를 실행하는데 걸리는 시간으로 정의한다[1]. 실제 최악 실행 시간과 분석 도구를 통해 예측한 최악 실행 시간 사이의 차이를 유발시키는 요소로는 성능 향상을 위해 사용하는 파이프라인, 캐시 등이 있다. 이러한 요소를 사용하는 프로세서는 예측성이 떨어지고, 기존의 분석 기법을 적용하면 실제 실행 시간과 큰 차이를 보인다. 이를 위해 Zhang[2]은 파이프라인을 모델링하여 실행 시간을 분석하였고, Mueller[3, 4]는 정적 캐시 시뮬레이션 기법을 제안하여 캐시를 분석하였다. 실행 시간의 차이를 유발하는 다른 요소로는 최악 실행 시간 분석 시 불가능한 경로(infeasible path)[5, 6, 7]에 의한 영향이 있다. 실제 실행 상황에서 발행 할 수 없는 실행 경로의 실행 시간이 분석 도구의 최악 실행 시간에 영향을 줌으로써 실제 최악 실행 시간과 차이가 발생한다. 이를 해결하기 위한 연구로는 Park의 동적 경로 분석 기법[8]과 Puschner 등이 제안한 마커-스코프 기법[9] 등이 있다.

2.2 정적 최악 실행 시간 분석 도구

정적 최악 실행 시간 분석은 프로그램 흐름 분석과 하드웨어의 구조를 모델링하여 최악 실행 시간을 구하는 것이다. 정적 최악 실행 시간 분석은 프로그램의 루프 횟수나 분기 방향등을 판단하는 흐름 분석과 파이프라인, 캐시, 분기 예측기 등을 적용하기 위한 하드웨어 모델링, 그리고 흐름 분석과 하드웨어 모델링을 바탕으로 최악 실행 시간을 결정하는 실행 시간 계산으로 나눌 수 있다. 정적 최악 실행 시간 분석도구에는 본 연구실에서 개발한 WATER[10]나 WATER를 확장한 Kernel-Analyzer[11]가 존재하고 외국의 경우 HEPTANE[12]이나 Cinderella[13]등이 존재한다.

2.3 인텔 XScale 코어

인텔 XScale 코어는 ARM V5TE 를 기반으로 하는 마이크로 프로세서이다. 고성능의 저전력을 위해 고안 되었으며, 휴대 단말 장치, 네트워크 장치, 저장 장치, 원격 접근 서버 등의 내장 시스템에서 많이 사용하고 있다[14]. 본 논문에서는 200 ~ 400Mh 의 스피드로 동작하는 PXA255를 대상으로 한다.

인텔 XScale 코어를 대상으로 최악 실행 시간을 분석하기 위해 고려해야 하는 것으로는 명령어 캐시, 데이터 캐시, 분기 예측기, 메모리 관리 장치, 파이프 라인 등이 있다. 그리고 최악 실행 시간 측정과 실행 정보 추출을 위해서 코프로세서 장치를 알아 볼 필요가 있다.

XScale의 명령어 캐시는 주기억 장치로부터 다수의 명령어를 캐시에 저장해 두어, 명령어를 패치(Fetch)하는 명령어의 수를 줄임으로써 시스템의 성능을 향상시킨다. XScale 프로세서의 명령어 캐시는 32K 혹은 16K 바이트 크기이고, 32-way 집합 연관 캐시(set associative cache)이다.

분기 예측기는 프로그램 실행에 흐름의 변경과 관련된 손해를 줄이기 위해 동적으로 분기를 예측하는 장치이다. XScale 코어의 파이프라인은 하나의 주기(cycle) 동안 여러개의 명령어를 동시에 수행할 수 있는 슈퍼파이프라인(Superpipeline) 구조이다. 최악 실행 시간 분석에 있어서 직접적으로 연관된 레지스터로는 시스템 제어 레지스터(CP15) 레지스터와 성능 모니터링 레지스터(CP14)가 있다. CP15 레지스터를 통해 메모리 관리 장치, 캐시, 버퍼 등의 정보를 알 수 있고, CP14 레지스터를 통해 프로그램의 수행 시간을 알 수 있다.

3. 실행 정보를 적용한 최악 실행 분석기

3.1 실행 정보를 적용한 최악 실행 시간 분석기 (RunInfo)

본 논문에서는 정적 최악 실행 시간 분석 시 동적인 사항에 대한 정보의 부재로 인해 실행 시간을 과대 측정하는 문제를 해결하기 위해 실행 정보 수집기를 제안한다. 실행 정보 수집기는 태스크가 실행을 시작할 시점에 필요한 정보를 대상 기기로부터 수집하여 분석기에 정보를 제공하는 역할을 한다.

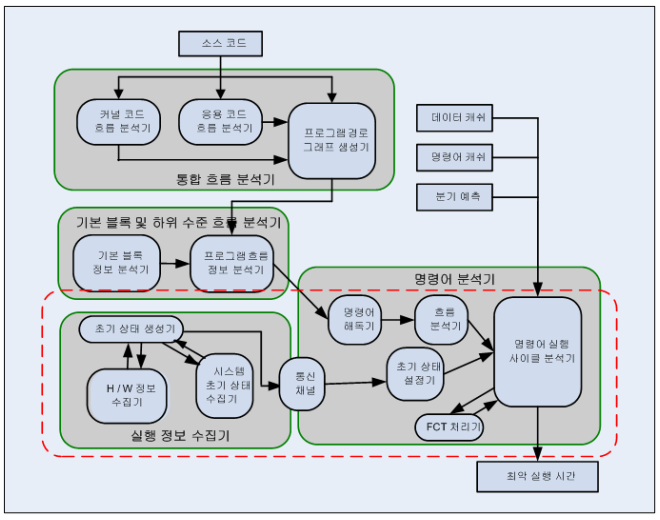

그림 1 RunInfo 분석 도구 구조

그림 1은 RunInfo 분석 도구의 전체 구조를 나타낸다. 기존의 최악 실행 시간 분석 도구인 Kernel-Analyzer에 실행 정보 수집기를 추가하고, 명령어 분석기를 확장한다. 실행 정보 수집기는 하드웨어 정보 수집기, 시스템 초기 상태 수집기, 초기 상태 생성기로 구성되고, 명령어 분석기에는 기존 명령어 분석기에 실행 정보를 적용하기 위해 초기 상태 설정기를 추가한다.

RunInfo 분석 도구는 크게 세 부분, 흐름 분석기(통합 흐름 분석기, 기본 블록 및 하위 흐름 분석기), 실행 정보 수집기, 명령어 분석기로 구성된다. 흐름 분석기는 태스크의 소스 코드로부터 상위 수준 흐름 그래프를 생성하고, 태스크의 오브젝트 파일로부터 하위 수준 흐름 그래프를 생성한다. 이 두 흐름 그래프를 동기화하여 명령

어 분석기로 전달한다. 실행 정보 수집기는 대상 기기로부터 태스크와 관련된 실행 정보를 생성하고, 이 정보를 명령어 분석기로 전달한다. 명령어 분석기는 태스크의 실행 시간 분석 시 흐름 분석기로부터 받은 흐름 그래프와 실행 정보 수집기로부터 받은 실행 정보를 적용하고 하드웨어 특성(캐시, 파이프라인, 분기 예측)을 고려하여 실행 시간을 계산한다.

3.2 실행 정보 수집기

실행 정보 수집기는 대상 기기로부터 실행 정보를 수집하여 명령어 분석기로 정보를 전달하는 역할을 수행한다. 실행 정보는 CPU, 캐시, 분기 예측기, MMU 등 대상 기기의 하드웨어 상태 정보와 태스크가 실행되기 직전의 레지스터 값, 캐시 잠금 정보, 현재 구동 중인 태스크의 수 등의 정보를 초기 상태 정보를 합한 정보이다. 정적 분석 시 동적인 상황에 대한 정보 부재로 과대 측정하는 문제는 이 실행 정보를 분석기에 적용함으로써 해결 할 수 있다.

그림 2는 실행 정보 수집기의 세부 구조이다. 실행 정보 수집기는 크게 세 부분으로 구성되어 있다. 태스크의 실행 정보를 생성하는 초기 상태 생성기, 하드웨어의 전반적인 정보를 수집하고 분석하는 하드웨어 정보 수집기, 태스크가 실행되기 직전의 레지스터 정보 등을 수집하고 분석하는 시스템 초기 상태 수집기가 있다.

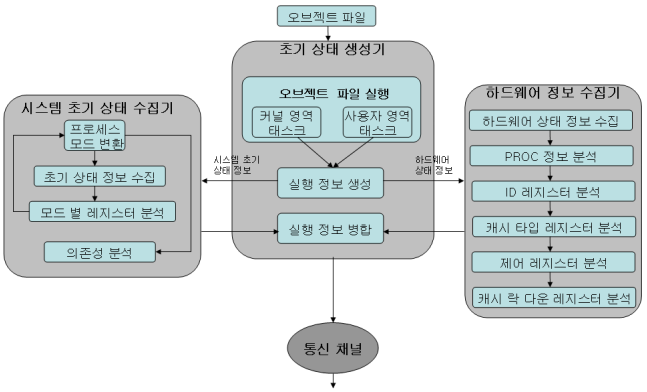

그림 2 실행 정보 수집기

실행 정보 수집기에서 태스크의 실행 정보를 수집하는 과정은 다음과 같다. 사용자가 분석을 원하는 태스크의 오브젝트 파일을 초기 상태 생성기에 입력한다. 초기 상태 생성기에서는 커널 모드에서 동작하는 태스크일 경우와 사용자 영역에서 동작하는 태스크로 나누어 파일을 실행한다. 프로그램이 실행 도중 분석을 원하는 태스크가 호출될 경우 실행되기 직전에 실행 정보를 생성한다. 실행 정보 중 하드웨어 상태 관련 정보는 하드웨어 정보 수집기에서 처리하고, 시스템 초기 상태 관련 정보는 시스템 초기 상태 수집기에서 처리한다. 하드웨어 정보 수집기에서는 PROC 파일 시스템 정보 분석, ID 레지스터 분석, 캐시 타입 레지스터 분석, 제어 레지스터 분석, 캐시 락 다운 레지스터 분석 등을 수행하여 실행 정보를 분석기에서 처리할 수 있는 데이터로 변환한다. 시스템 초기 상태 수집기에서는 각 프로세서 모드 별 레지스터 값을 수집하기 위해 프로세스 모드를 변환한다. 모든 프로세서 모드의 레지스터 값을 수집한 후 시스템 초기 상태 수집기와 분석을 원하는 태스크 사이의 의존성 문제를 처리한다. 하드웨어 정보 수집기와 시스템 초기 상태 수집기에서 수집하고 분석한 실행 정보를 초기 상태 생성기에서 병합하여 통신 채널을 통해 명령어 분석기로 전송한다.

3.2.1 하드웨어 정보 수집기

하드웨어 정보 수집기는 태스크가 실행되기 직전의 CPU, MMU, 명령어 캐시, 데이터 캐시, 캐시 잠금, 분기 예측기, 쓰기 버퍼 등의 하드웨어 상태 정보를 수집하는 역할을 한다. 최악 실행 시간 분석기는 이 정보를 통해서 대상 기기의 하드웨어 상태 정보를 재조정하여 보다 정확한 최악 실행 시간을 분석할 수 있다. 그림 3은 하드웨어 상태 정보의 자료 구조이다.

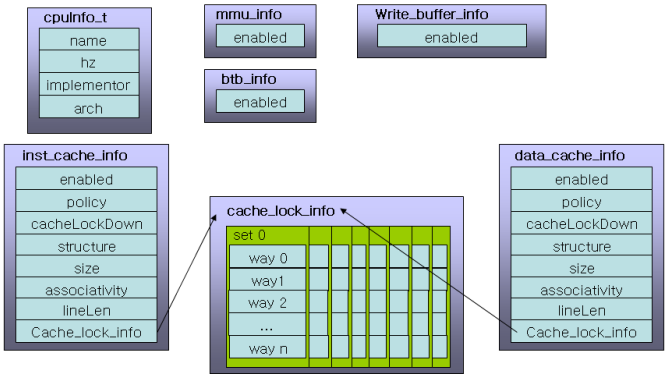

그림 3 하드웨어 상태 정보

하드웨어 상태 정보는 RunInfo 분석기의 실행 시간을 나타내는 사이클의 단위를 위한 CPU의 클럭(Clock) 정보, ARM의 기본 골격이 되는 구조를 알기 위한 아키텍쳐 버전 정보, 메모리 관리 유닛을 사용하는지를 나타내는 MMU 정보, 분기 명령어의 분기 방향을 예측함으로써 파이프라인을 유지시켜주는 분기 예측기 정보, 주 메모리에 데이터를 저장하는 명령어로 인한 성능 감소를 해결해 주는 쓰기 버퍼의 사용 여부 정보, CPU와 주 메모리 사이의 속도차를 완화시켜주는 명령어 캐시와 데이터 캐시의 구조, 크기, 라인 단위, 잠금 정보 등으로 구성된다. 하드웨어 상태 정보는 인텔 XScale의 CP15 레지스터(ID레지스터, 캐시 타입 레지스터, 제어 레지스터, 캐시 락 다운 레지스터)와 리눅스의 PROC 파일 시스템을 분석하여 얻는다.

3.2.2 코프로세서 15 레지스터(CP15) 관련 자료 구조

코프로세서 15 레지스터는 시스템 제어 레지스터로 인텔 XScale 코어의 CPU, MMU, 분기 예측기 등의 하드웨어를 제어하는데 사용한다. CP15 의 ID 레지스터, 캐시 타입 레지스터, 제어 레지스터, 캐시 락 다운 레지스

터를 통해 최악 실행 시간 분석기에 필요한 정보를 얻을 수 있다.

ID 레지스터는 CP15 레지스터로 ARM 프로세서를 식별할 수 있는 정보(아키텍쳐 버전과 제조사 등)가 들어있다. 분석기에서 ID 레지스터의 정보의 관리를 쉽게 하기 위해, ID 레지스터의 자료형은 value 와 field 로 구성된 union 타입으로 정의한다.

캐시 타입 레지스터(Cache Tyegister)는 CP15 의 0번 레지스터로 명령어 캐시와 데이터 캐시에 대한 정보가 들어있다. 이 레지스터를 통해 명령어 캐시와 데이터 캐시의 구조, 크기, 셋 연관성(associativity), 라인 단위 등의 정보를 알 수 있다. 분석기에서 캐시 타입 레지스터의 관리를 편하게 하기 위해 union 타입으로 정의한다.

제어 레지스터(Control Register)는 CP15 의 1번 레지스터이다. 이 레지스터를 통해 MMU, 분기 예측기, 캐시 등의 정보를 알 수 있다

캐시 락 다운 레지스터(Cache Lock Down Register)는 CP15 의 9번 레지스터로 캐시의 잠금을 설정할 수 있다. 캐시 락 다운 레지스터의 자료 구조는(그림 13)과 같다.

3.2.3 시스템 초기 상태 수집기

시스템 초기 상태 수집기는 태스크가 실행하기 직전의 각 모드별 레지스터 값, 프로세서 모드, 현재 구동 중인 태스크의 수 등의 정보를 수집한다. 최악 실행 시간 분석기는 이 정보를 통해 태스크를 분석할 때 초기 상태 값을 알 수 있다.

3.2.4 초기 상태 생성기

초기 상태 생성기는 대상 기기에서 최악 실행 시간 분석을 원하는 태스크를 구동하여 분석에 필요한 실행 정보를 생성한다. 그림 8은 사용자 태스크와 커널 태스크가 실행 정보를 생성하고 수집하는 함수를 요청하는 과정이다.

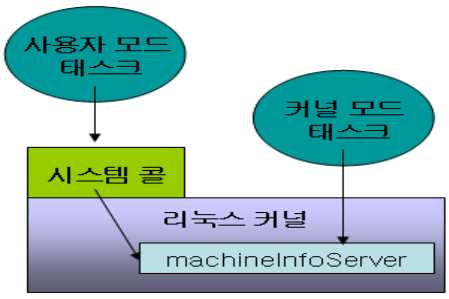

그림 8 실행 정보 함수 호출

실행 정보를 생성하고 수집하기 위해서는 해당 자원에 접근할 권한을 가지고 있어야 한다. 실행 정보를 생성하고 수집하는 모듈은 machineInfoServer 로 리눅스 커널에 포함시키고, 커널 심볼 테이블에 등록하여, 이 함수를 다른 태스크에서 호출 할 수 있게 한다. 커널 모드에서 동작하는 태스크의 경우는 machineInfoServer 를 바로 호출 할 수 있다. 하지만 사용자 모드에서 동작하는 태스크의 경우는 커널 함수를 호출 할 수 없기 때문에 machineInfoServer를 호출 할 수 있는 machineInfoServer_Call 함수를 시스템 콜로 등록한다. 사용자 모드 태스크는 machineInfoServer_Call 시스템 콜을 통해 실행 정보를 생성하고 수집한다.

초기 상태 생성기로부터 수집한 정보는 그림9의 왼쪽과 같고, 이 정보를 분석한 정보는 오른쪽과 같다.

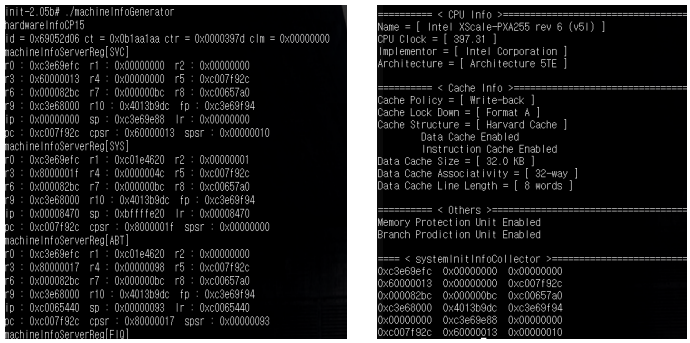

그림 9 실행 정보와 실행 정보 분석 예

4. 성능 평가

4.1 동작 환경

RunInfo 분석기는 인텔 XScale 코어를 대상으로 하고, XScale 을 위한 GNU C/C++ 크로스 컴파일러를 사용한다. 다양한 플랫폼을 지원하기 위해 통합 개발 환경인 eclipse[15] 의 플러그인으로 제공한다.

RunInfo 분석기는 인텔 x86 호환 기종에서 동작하고, 실행 정보를 수집하는 실행 정보 수집기 모듈은 인텔 XScale 코어를 사용하는 장비에서 동작한다. RunInfo 분석기의 실행 결과 화면으로 코드, CFG, 실행 정보 창 등으로 구성된다.

4.2 측정 방법 및 제한

성능 평가의 기준이 되는 태스크의 최악 실행 시간 값(사이클) 측정은 XScale 코어의 성능 모니터링 관련 코프로세서인 CP14 레지스터의 PMNC(Performance Monitor Control Register), CCNT(Clock Counter Register), PMN0(Performance Count Register 0), PMN1 (Performance Count Register 1) 를 이용하여 측정한다. CP14 레지스터를 통해 측정 한 값 중 가장 큰 값을 최악 실행 시간으로 간주한다.

성능 평가의 기준이 되는 과대 측정 비율은 다음과 같다. 이는 분석기에서 측정한 최악 실행 시간 값이 실제 최악 실행 시간에 얼마나 근접했는지를 나타내는 척도가 된다.

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

4.3 측정 결과 및 분석

표 1은 분석할 태스크의 종류와 태스크의 최악 실행 시간을 측정한 값이다. 커널 태스크의 경우 가능한 실제 커널 코드에 존재하는 태스크를 대상으로 한다. 분석에 사용된 태스크의 수는 커널 태스크 6개, 사용자 태스크 6개이다. 실측값은 대상 기기에서 태스크를 호출하여 측정한 태스크의 최대 실행 시간이다. 실행 시간의 단위는 사이클이다.

표 1 태스크의 종류와 최악 실행 시간

	태스크	실측값(사이클)
커널	(a) tqueue_bh	9370222
	(b) immediate_bh	11866228
	(c) do_fork	336339
	(d) show_state	104011786
	(e) schedule	12907
	(f) kerneltimer_periodic_led	3090
사용자	(g) LinearSearch	4999
	(h) init_CS2	657826
	(i) LedControl	748231
	(j) removeSpacesAndCaps	42007
	(k) periodicHandler	81665
	(l) SignalRestore	19997

표 2는 표 1의 태스크를 WATER, Kernel-Analyzer, RunInfo 분석기를 통해 측정한 최악 실행 시간 값과 이들 분석기의 과대 측정 비율을 나타낸다. WATER 분석기의 경우 커널 코드는 분석할 수 없기 때문에 WATER의 최악 실행 시간 측정값과 과대 측정 비율 값을 생략한다. 표 2에서 Kernel, RunInfo 는 각각 Kernel-Analyzer, RunInfo 분석기를 의미한다.

<표 2> 분석기 별 최악 실행 시간과 과대 측정 비율

T		WATER		Kernel		RunInfo	
		WCET	과대 측정	WCET	과대 측정	WCET	과대 측정
커널	(a)			107391	-99%	107311	-99%
	(b)			102640	-99%	1142470	-90%
	(c)			406180	21%	386019	15%
	(d)			130002122	25%	125002009	20%
	(e)			18829	46%	15251	18%
	(f)			4073	32%	3903	26%
사용자	(g)	7987	60%	6272	25%	6332	27%
	(h)	709883	8%	759207	15%	659037	0.2%
	(i)	1079738	44%	1030822	38%	1028822	38%
	(j)	60010	43%	54221	29%	52833	26%
	(k)	85602	5%	86132	5%	85962	5%
	(l)	25157	26%	22992	15%	22822	14%

최악 실행 시간 분석의 목표는 정확한 최악 실행 시간의 계산이 아니라 과대 측정 범위를 가능한 작게 하고 과소 측정 하지 않아야 한다는 것이다. 표 2에서 커널 태스크 queue_bh(a) 와 immediate_bh(b)의 경우 과대 측정 비율이 음수로 나온다. 이는 두 태스크의 경우 분석기에서 과소 측정을 한 것이다. 두 태스크는 리눅스에서 하반부 작업을 처리한다. 하반부 작업은 스케줄러에서 여분의 시간에 작업을 처리하기 때문에 태스크의 코드 실행 시간 이외의 지연이 발생한다. 이 지연으로 인해 분석기에서 과소 측정을 한 것이다. 본 논문에서는 하반부 작업을 수행하는 태스크를 제외하고 분석을 한다. 그림 11은 표 2에서 커널 태스크의 과대 측정 비율을 도표로 나타낸 것이다. 커널 태스크의 경우 Kernel-Analyzer 분석기는 평균 31% 정도의 과대 측정을 하고, RunInfo 분석기는 평균 20% 정도의 과대 측정을 한다. RunInfo 분석기의 경우 실행 정보를 적용함으로써 Kernel-Analyzer 분석기보다 11% 정도의 과대 측정 비율이 감소한다. 태스크의 실행 시간에 크게 영향을 주는 요소로는 루프 횟수, 분기 방향, 캐시 적중률을 들 수 있다. RunInfo 분석기의 경우 실행 정보 적용으로 캐시 미스율에 대한 분석이 Kernel-Analyzer 보다 정확하기 때문에 과대 측정 비율이 크게 감소한다. 대체적으로 태스크의 기본 블록 수가 많고, 분기와 루프가 많을수록 과대 측정 비율의 감소량이 증가한다.

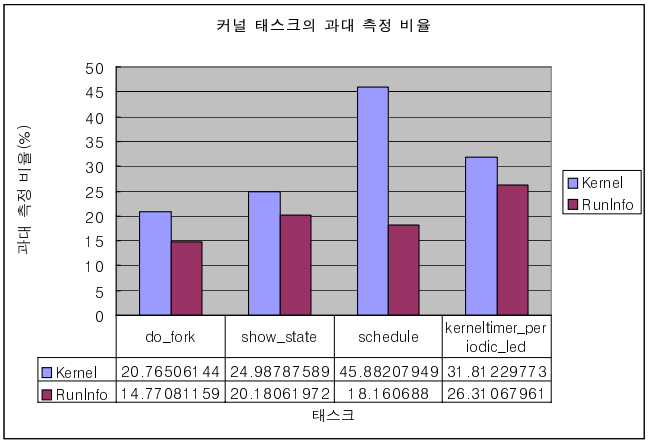

그림 11 커널 태스크의 과대 측정 비율

그림 12는 표 2에서 사용자 태스크의 과대 측정 비율을 도표로 나타낸 것이다. WATER, Kernel -Analyzer, RunInfo 분석기는 각각 평균 31%, 25%, 19% 의 과대 측정을 한다. Kernel-Analyzer 와 RunInfo 분석기는 힌트 정보 적용으로 WATER 분석기 보다 10% 이상 적다. RunInfo 분석기는 실행 정보를 적용하여 Kernel-Analyzer 분석기보다 평균 3% 정도의 과대 측정 비율이 감소한다. 사용자 태스크의 경우 커널 태스크에 비해 RunInfo 분석기의 과대 측정 비율 감소량이 적은 이유는 사용자 태스크의 경우 커널 자원을 사용하기 위해 여분의 작업이 필요하고, 이에 의해 지연이 발생하기 때문이다. 지연에 의한 실행 시간 값이 측정할 태스크의 코드 실행 시간 값에 비해 상대적으로 크기 때문에 과대 측정 비율 감소량이 적다.

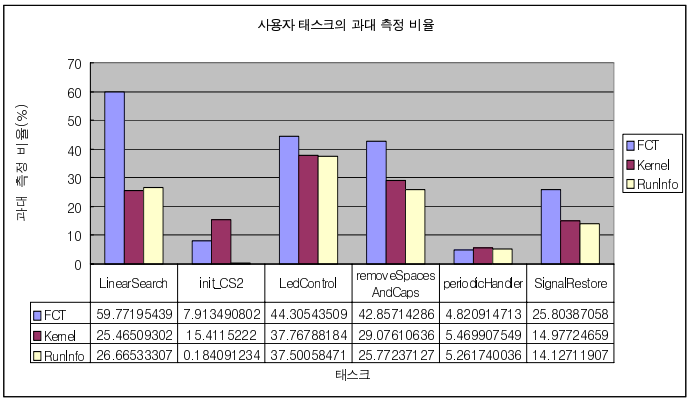

그림 12 사용자 태스크의 과대 측정 비율

표 3은 분석기 별 과대 측정 비율을 요약한 것이다. 실행 정보를 적용한 RunInfo 분석기는 기존 분석기 보다 평균 3 ~ 11% 정도의 과대 측정 비율이 감소한다. 메모리 접근 명령어가 많은 태스크의 경우 캐시 미스율에 대한 분석이 보다 정확해 지기 때문에 최대11% 까지 차이가 난다.

표 3 분석기 별 과대 측정 비율

분석기 / 태스크	WATER	Kernel-Analyzer	RunInfo
커널 태스크		31%	20%
사용자 태스크	31%	21%	18%
총 태스크	31%	25%	19%

5. 결론 및 향후 계획

내장 실시간 시스템의 시간적 정확성을 만족하기 위해서는 태스크들의 스케줄링 가능성이 분석이 필요하다. 태스크의 스케줄링 가능성 분석을 위해 선행 되어야 하는 과제는 태스크의 최악 실행 시간 분석이다. 정적 분석을 기반으로 최악 실행 시간을 분석하는 기존의 도구는 동적인 상황에 대한 정보 부재로 최악 실행 시간을 크게 과대 측정하여 시스템의 자원을 낭비하는 문제점이 있다.

본 논문에서는 정적 최악 실행 시간 분석기의 과대 측정 비율을 줄이기 위해 실행 정보를 제안한다. 실행 정보를 대상 기기로부터 생성하고, 분석기에 적용 시 고려사항을 기술한다. 본 논문에서 제안한 실행 정보를 분석기에 적용함으로써 기존 분석기보다 평균3 ~ 11%의 과대 측정 비율이 감소한다.

향후 연구 과제로서 리눅스 커널의 하반부 작업을 처리하는 태스크의 경우 지연에 대한 처리 방법이 필요하다. 그리고 현재 시스템에서 동작하는 태스크들 간의 의존성 분석과 태스크의 불가능 경로(Infeasible Path) 분석을 통해 보다 정확한 최악 실행 시간 분석이 필요하다.

참고문헌

[1] Peter P. Puschner, Alan Burns, "A Review of Worst-Case Exec- ution-Time Analysis", Real-Time Systems, IEEE Transaction on Software Engineering SE-12(9), pp. 941-949, 1986

[2] N. Zhang, A. Burns, and M. Nicholson, "Pipelined Processors and Worst-Case Execution Times", Real-Time Systems, Vol. 5, No. 4, pp.319-343, October 1993.

[3] F. Mueller, "Static Cache Simulation and its Applications", PhD thesis, Florida State University, 1994.

[4] F. Mueller and D. B. Whalley, "Efficient On-the-fly Analysis of Program Behavior and Static Cache Simulation", In Proceedings of the 11th Real-Time Systems Symposium, pp. 72-81, December 1990.

[5] David Hedley, Michael A. Hennell, "The Causes and Effects of Infeasible Paths in Computer Programs", ICSE 1985: 259-267

[6] Derek Yates, N. Malevris, "Reducing the Effects of Infeasible Paths in Branch Testing", Symposium on Testing, Analysis, and Verification 1989: 48-54

[7] Apostolos A. Kountouris, "Safe and efficient elimination of infea- sible execution paths in WCET estimation", RTCSA 1996: 187-194

[8] C. Y. Park, "Prediction Program Execution Times by Analyzing Static and Dynamic Program Paths", Real-Time Systems, Vol. 5, No. 1, pp. 31-62, 1993.

[9] G. Pospischil, P. Puschner, A. Vrchoticky, and R. Zainlinger, "Developing Real-Time Tasks with Predictable Timing", IEEE Software, Vol. 9, No. 5, pp. 35-44, September 1992.

[10] Jeon. H. J., "동적 함수 호출 예측 방법을 이용한 최악 실행 시간 분석기 설계 및 구현, 숭실대학교 석사학위 논문, Dec. 2004.

[11] Choi, M. S., "XScale 최악실행시간 분석 도구를 위한 커널 코드 흐름 분석기의 설계 및 구현 숭실대학교 석사학위 논문, Dec. 2005.

[12] A. Colin and I. Puaut. "A modular and retargetable framework for tree-based wcet analysis", In Proceedings of the 13th Euromicro Conference on Real-Time Systems, pages 37-44, Delft, Netherlands, June 2001.

[13] Yau-Tsun Steven Li, Sharad Malik, Andrew Wolfe, "A Retarget- able Environment for Performance Analysis of Real-Time Software. Euro-Par 1997: 1308-1315

[14] Intel, "XScale Core Developer's Manual", January, 2004.1

[15] Elclipse project official site, http://www.eclipse.org