□ 기술해설 □

실시간 시스템을 위한 최악 실행시간 분석 기법

서울대학교 임성수* · 민상렬**

● 목 차 ●

1. 서 론

실시간 시스템을 구성할 때에는 스케쥴 가능성 분석(schedulability analysis)에 이용하기 위한 각 태스크의 최악 실행시간(worst case execution time : WCET) 예측이 필요하다. 이 때 예측된 결과가 실제 최악 실행시간보다 작은 경우에는 그 결과를 이용한 스케쥴 가능성 분석 결과가 올바르지 않게 되고, 실제 최악 실행시간보다 지나치게 큰 경우에는 프로세서 자원의 낭비를 가져온다. 따라서 예측된 결과는 최소한 실제 최악 실행시간보다 커야하는 안전성(safeness)을 필수적으로 갖추어야 하며, 실제 최악 실행시간에 근접해야 하는 정확성(accuracy)도 아울러 갖추어야 한다. 최근까지 이러한 안전성과 정확성을 갖춘 최악 실행시간 분석 기법에 관한 여러 연구가 진행되었다.

초기의 연구들은 프로그램을 실행하는 프로세서가 순차적으로 한 번에 하나의 명령어만을 실행한다고 가정하고 있다. 그러나 최근의 프로세서들은 대부분 성능 향상을 위해서 캐쉬와 파이프라인 구조를 갖추고 있다. 캐쉬는 메모리 참조의 시간적(temporal), 공간적(spatial) 지역성(locality)을 활용하여 메모리 참조 시간을 줄이기 위한 구조이다. 파이프라인 실행구조는 한 명령어를 여러 단계로 나누어 실행하는 것으로서 각 실행단계에서는 서로 다른 명령어가 동시에 실행 가능하기 때문에 프로세서의 실행 성능을 높인다. 이러한 프로세서에 기존의 분석 기법들을 그대로 적용하면 실제 실행시간과 큰 차이를 보이게 된다. 따라서 파이프라인 실행구조와 캐쉬를 갖춘 프로세서를 위한 최악 실행시간 분석 기법들이 최근에 여러 연구팀에서 제안되었다. 본 논문에서는 최근에 제안된 최악 실행시간 분석 기법들의 주요 특징을 소개하고 이들을 비교한다.

본 논문의 구성은 다음과 같다. 2 장에서는 프로그램의 최악 실행시간을 분석하고자 할 때 가질 수 있는 분석의 방향에 대한 관점을 소개하고 프로그램의 경로상의 제약을 활용한 기법들을 소개한다. 3 장에서는 최근의 여러 연구에서 보다 정확한 최악 실행시간 분석 결과를 얻기 위해 제안된 프로세서의 파이프라인 실행이나 캐쉬에 대한 모델링 기법들을 소개한다. 4 장에서는 여러 연구 팀에서 제안한 기법들 중 분석기를 갖추고 있고 이 분석기를 이용한 결과를 제시한 바 있는 기법들을 선택하여 비교한다. 4 장에서 다루는 기법들은 Park과 Shaw의 타이밍 스키마 기법 [9, 10, 14], 비엔나 대학(Technical University of Vienna)에서 제안한 Mars 시스템의 MAXT 기법[11,

*학생회원
**종신회원

12], 플로리다 주립 대학(Florida State University)의 실시간 시스템 연구 그룹에서 제안한 분석 기법[1, 7, 8], 프린스턴 대학(Princeton University)에서 제안한 기법[4], 그리고 서울대학교 실시간 시스템 연구 그룹에서 제안한 확장된 타이밍 스키마 기법[2, 6] 등이다. 5 장에서는 최악 실행시간 분석 기법의 비교 결과를 정리하고 결론을 맺는다.

2. 프로그램의 최악 실행시간 분석

프로그램의 최악 실행시간을 분석할 때에는 그 프로그램의 구문 트리 정보를 이용하거나 제어 흐름 정보를 이용할 수 있다. 구문 트리 정보를 이용하는 경우는 하나의 프로그램을 작은 단위로 나누어서 가장 작은 단위로부터 전체 프로그램에까지 거슬러 올라가면서 분석을 행하는 계층적인 분석 기법이고, 제어 흐름도에 기반한 분석 기법은 프로그램의 시작 시점부터 제어의 흐름을 따라가면서 분석을 행하는 기법이다. 구문 트리 정보를 이용하는 경우에는 프로그램을 작성할 때 프로그래머가 사용한 고수준의 프로그래밍 언어의 구조와 연관을 시킬 수도 있다. 따라서 프로그램 작성자의 관점에서 선호되는 기법이다. 이밖에 프로그램의 구조를 정수 선형식으로 표현하여 이 선형식을 만족하는 값을 구함으로써 최악 실행시간을 얻는 방법[4]도 있다.

프로그램의 구문 구조에 근거하여 계층적으로 실행시간을 분석하는 연구로는 Park과 Shaw의 타이밍 스키마 기법, 비엔나 대학의 Puschner 등이 제안한 MAXT 기법, 그리고 서울대학교 실시간 시스템 연구 그룹의 Lim 등이 제안한 확장된 타이밍 스키마 기법 등이 있다. 이들 연구에서는 모두 프로그램 구문마다의 분석식을 정의하여 실행시간 분석에 이용하고 있다. 타이밍 스키마와 MAXT 연구에서의 분석식에서는 각 프로그램 구문마다 최악 실행시간 범위를 구하고 프로그램 구문 사이에 단순한 덧셈 연산을 적용한다. 한편, 확장된 타이밍 스키마에서는 프로그램 구문마다 WCTA (Worst Case Timing Abstraction) 라고 하는 실행시간 정보를 파이프라인 실행과 캐쉬 동작 행태를 반영하여 구성하고, WCTA 사이에는 덧셈 연산 대신 접속 연산을 정의하여 사용한다. 확장된 타이밍 스키마에서는 여러개의 경로가 포함된 구문의 경우 어느 경로가 최악 실행 경로에 포함될 지 결정할 수 없는 경우 각 경로의 WCTA를 함께 유지한다[5, 6].

비엔나 대학의 Puschner와 Koza는 C 프로그램의 형식에 실행시간 분석에 이용하기 위한 정보를 제공할 수 있는 요소를 추가하여 MARS-C[11] 라고 하는 언어를 제안하였다.

플로리다 주립 대학의 Healy 등의 연구와 프린스턴 대학의 Li 등의 연구에서는 제어 흐름도에 기반하여 프로그램의 실행시간을 분석한다. 이들 연구에서는 대상 프로그램을 제어 흐름도로 표현하고, 기본블럭으로 나뉘어진 어셈블리 프로그램과 제어 흐름도를 연관시켜 분석에 이용한다. Li 등의 연구에서는 제어 흐름도로 표현된 프로그램을 정수 선형 계획법(integer linear programming)을 사용하여 다시 표현하고 프로그램의 실행을 제약하는 정보를 표현한 제약식을 이용하여 이 정수 선형식을 만족하는 값을 구함으로써 최악 실행시간을 구한다.

실제 최악 실행시간과 예측된 최악 실행시간 사이의 차이를 유발하는 중요한 요인으로 불가능한 경로(infeasible path)에 의한 영향을 들 수 있다. 다시 말하면 실제 실행 상황에서 발생할 수 없는 실행경로의 실행시간이 최악 실행시간 예측에 포함되는 것이다. 이러한 요인으로 인한 과예측은 정적인 분석 과정에서 실행경로간 제약을 고려하지 않기 때문에 발생한다. 실행 경로간 제약을 고려하여 최악 실행시간을 분석하고자 한 연구로는 Park의 동적 경로 분석(dynamic path analysis)[10]과 비엔나 대학의 Puschner 등이 제안한 마커-스코프(marker-scope)[11, 12] 방법이 있다.

Park의 동적 경로 분석 기법은 프로그래머가 프로그램의 실행 경로에 관해 제공한 정보를 기반으로 하여 프로그램 내의 불가능한 경로를 제거한다. 여기서는 사용자가 자신이 작성한 프로그램의 알고리즘과 논리적 흐름에 관한 충분한 지식을 갖고 있으므로 프로그램의 실제 실행 행태에 관한 정보를 제공할 수 있다

고 가정하고 있다.

Puschner 등은 실제 실행시간과 예측된 수행시간과의 차이를 줄이기 위해 마커와 스코프라는 새로운 언어구조를 정의하고 있다. 스코프는 프로그램 코드의 일부분이고, 마커는 스코프 내에 위치하는 표시기로서 표시된 부분의 명령어가 최대 몇 번 수행되는 지를 나타낸다. 특히, 마커는 순환문 내의 여러 경로의 수행 횟수의 제약 조건을 표현하는데 유용하게 이용될 수 있으므로 순환문 구조를 하나의 스코프로 설정하여 사용하는 경우가 많다.

3. 최악 실행시간 분석을 위한 프로세서의 모델링

실제 실행시간에 근접하는 최악 실행시간 분석 결과를 얻기 위해서는 분석할 프로그램이 실행될 하드웨어 환경을 파악하고, 이를 분석 과정에 반영해야 한다. 최근의 RISC 프로세서들은 대부분 파이프라인 실행 구조와 캐쉬를 갖추고 있다. 그러나 Park과 Shaw의 타이밍 스키마 연구나 비엔나 대학의 MAXT 기법에서는 파이프라인 실행과 캐쉬에 의한 영향을 전혀 고려하지 않았다. 따라서 이들 기법들을 그대로 RISC 프로세서의 최악 실행시간 분석에 이용하면 분석 결과는 실제 최악 실행시간과 큰 차이를 보이게 된다. 최근 제안된 플로리다 대학의 Healy 등의 연구, 프린스턴 대학의 Li 등의 연구나 서울대학교 실시간 시스템 연구 그룹의 Lim 등의 확장된 타이밍 스키마 연구에서는 파이프라인 실행구조와 캐쉬의 실행 행태를 최악 실행시간 분석에 반영하였다. 3 장에서는 이들 기법들에서 프로세서의 파이프라인 실행 행태와 캐쉬의 동작을 모델링하기 위하여 사용한 기법들을 소개한다.

3.1 파이프라인 실행의 모델링

파이프라인 실행구조를 갖춘 프로세서는 하나의 명령어를 여러 단계로 나누어서 실행한다. 또한 각 실행 단계에서는 동시에 각각 다른 명령어를 실행할 수 있어서 여러개의 명령어가 중첩되어 실행되는 효과를 보이게 된다. 이러한 파이프라인 실행 구조를 갖춘 프로세서의 최악 실행 행태를 분석하면서 기존의 각 명령어를 순차적으로 하나씩 실행하는 프로세서의 실행시간을 분석하기 위한 기법을 그대로 적용하면, 분석 결과는 실제 실행시간과 큰 차이를 보이게 된다. 따라서 최근의 최악 실행시간 분석 기법들에서는 파이프라인 실행 행태를 반영하였다.

최초로 파이프라인 실행 행태를 반영하여 프로그램의 최악 실행시간을 분석하고자 한 연구는 Zhang 등의 연구[15]가 있다. 이 연구에서는 Intel 80C188 프로세서의 두 단계 파이프라인 실행을 모델링하여 명령어의 중첩된 실행 행태를 최악 실행시간 분석에 반영하였다. 그러나 이 연구에서 모델로 삼은 Intel 80C188 프로세서는 두 단계만으로 이루어진 지극히 간단한 파이프라인 실행 구조를 갖추고 있어서 이 모델을 일반적인 파이프라인 실행 구조에 적용하기는 어렵다.

서울대학교 실시간 시스템 연구 그룹의 Lim 등이 제안한 확장된 타이밍 스키마[5, 6]에서는 자원예약표를 이용하여 파이프라인 실행을 모델링하였다. 자원예약표는 명령어가 파이프라인 실행되는 모습을 파악하기 위해 제안된 기법으로 실행시간의 흐름에 따른 각 실행자원의 이용 상태를 표의 형태로 나타낸다.

파이프라인 실행 구조를 갖춘 프로세서에서는 앞의 명령어들의 실행에 의해 결정된 파이프라인 상태에 따라 현재의 명령어들이 보일 실행 행태가 결정된다. 다시 말하면 현재 실행될 명령어들의 실행 행태는 과거 상태에 의존적이다. 따라서 파이프라인 실행 구조를 갖춘 프로세서에 대하여 프로그램의 최악 실행시간을 계층적으로 분석할 때에는 이러한 특성을 반영하여야 한다. 또한, 계층적으로 프로그램의 실행시간을 분석할 때에는 선행하여 실행될 명령어들이 결정되지 않은 상태에서 어떤 실행블럭[1]의 실행 행태를 예측하여야 한다. 확장된 타이밍 스키마에서는 우선 각 기본블럭에 대하

1) 실행블럭은 프로그램 내의 명령어들로 이루어진 분석의 단위로서 재귀적으로 정의된다. 즉, 가장 작은 실행블럭은 기본블럭이고, 기본블럭들의 모임인 하나의 프로그램 구문, 더 나아가 프로그램 구문들의 모임인 확장된 프로그램 구문이 실행블럭이 될 수도 있다.

여 파이프라인의 상태가 초기 상태라고 가정하고 자원예약표를 구성하고 기본블럭들간에 접속 연산을 정의하여 이를 적용해가면서 점차로 분석의 범위를 확대한다.

플로리다 대학의 Healy 등은 Lim 등의 연구에서 사용한 자원예약표와 같은 형태의 파이프라인 실행도(pipeline diagram)[1]를 사용하여 파이프라인 실행을 모델링하였다.

프린스턴 대학의 Li 등은 프로그램을 정수 선형 계획법을 사용해 표현하기 위해 각 기본블럭의 최악 실행시간을 상수로 결정하여 사용하였다. Li 등의 연구와 Zhang 등의 연구에서는 기본블럭 사이의 파이프라인 실행으로 인한 중첩 실행을 반영하지 않았기 때문에 분석 대상 프로그램의 특성에 따라서는 분석 결과와 실제 실행시간 사이에 큰 차이가 발생할 수 있다.

3.2 캐쉬 동작의 모델링

캐쉬는 프로세서의 실행 속도와 메모리 접근 속도와의 차이를 극복하기 위해 개발된 것으로 캐쉬 접근 성공시에는 메모리 접근으로 인한 실행 지연이 필요치 않게 되어 프로세서의 실행 성능을 높이게 된다. 일반적으로 캐쉬를 갖춘 프로세서의 경우 캐쉬 접근의 성공, 실패 여부를 프로그램의 실행 전에 완전히 파악하기가 힘들기 때문에 전체 프로그램의 실행시간 예측을 어렵다고 간주되어 실시간 시스템에 이용되기 어려웠다. 그러나 최근에 캐쉬의 동작 행태을 정적으로 파악하여 실행시간 분석에 반영하고자 한 기법들이 제안되었다. 이러한 연구로는 Mueller 등의 연구[7,8]와 Lim 등의 연구[5, 6], 그리고 Li 등의 연구[4]가 있다.

Mueller 등은 정적 캐쉬 시뮬레이션 기법을 제안하여 이를 최악 실행시간 분석에 이용하였다. 정적 캐쉬 시뮬레이션 기법에서는 프로그램의 각 명령어의 메모리 접근 주소를 바탕으로 하여 각 명령어의 캐쉬 접근을 다음 네가지 범주로 분류하여 실행시간 분석시 이를 이용한다.

① 항상 캐쉬 접근 실패하는 경우
② 항상 캐쉬 접근 성공하는 경우
③ 최초의 캐쉬 접근만 실패하는 경우
④ 최초의 캐쉬 접근만 성공하는 경우

정적 캐쉬 시뮬레이션은 프로그램의 제어 흐름도를 기반으로 하여 수행된다. 프로그램의 시작 기본블럭에서부터 시작하여 제어 흐름도를 따라 분석을 진행하면서 각 기본블럭마다 input_state와 output_state라고 하는 캐쉬의 상태를 나타내는 자료구조를 구성한다. 구성된 input_state와 output_state에 따라 각 명령어의 캐쉬 접근이 위에 열거한 네가지 중 하나의 범주로 결정된다. 정적 캐쉬 시뮬레이션 기법은 직접 연관(direct mapped)된 명령어 캐쉬만을 대상으로 하고 있다.

Lim 등이 제안한 확장된 타이밍 스키마에서는 계층적인 최악 실행시간 분석에 이용하기 위하여 각 기본블럭마다 캐쉬의 동작 행태에 따른 실행시간 정보를 구성한다. 계층적인 분석시에는 프로그램의 시작 시점에서부터의 캐쉬의 상태가 반영되지 않고 분석하고자 하는 실행블럭 자체만 대상으로 할 때의 캐쉬의 상태를 고려할 수 밖에 없다. 이 때에는 선행하는 실행블럭이 결정되지 않은 상태에서 캐쉬 동작과 관련된 정보를 구성한 후 선행 실행블럭이 결정되면 이의 캐쉬 동작에 의한 영향을 반영하여 이전에 구성한 실행시간 정보를 재구성해야 한다. 확장된 타이밍 스키마에서는 각 실행블럭에 대하여 실행블럭의 시작 시점에서 캐쉬의 상태가 완전히 초기 상태라고 가정했을 때, 그 실행블럭의 마지막 명령어의 실행 시점까지 캐쉬의 상태를 추적하여 최초 참조 정보(first reference)와 최종 참조 정보(last reference)라는 정보를 유지한다. 최초 참조 정보에는 각 캐쉬블럭에 처음으로 적재될 메모리 블럭의 주소가 유지되며, 최종 참조 정보에는 해당 실행블럭의 실행이 끝났을 때 각 캐쉬블럭에 남아 있을 메모리 블럭의 주소가 유지된다. 최초 참조 정보는 인접하여 앞에 실행되는 실행블럭의 캐쉬 동작에 의한 영향을 반영하기 위하여 유지되며, 최종 참조 정보는 해당 실행블럭이 인접하여 뒤에 실행되는 실행블럭에 미칠 수 있는 영향을 반영하기 위하여 유지된다. 확장된 타이밍 스키마에서는 접속 연산시에 연산에 참여하는 앞 실행블럭의 최종 참조 정보와 뒤 실행블럭의 최초 참조 정보의 내용이 비

교되고 이때 밝혀지는 두 실행블럭 사이의 캐쉬접근 성공, 실패 횟수에 따라 실행시간 정보가 재구성된다.

프린스턴 대학의 Li 등은 선형 계획법에 의해 최악 실행시간을 분석하는 기법을 제안하였다. 이 기법에서는 캐쉬 동작의 영향을 반영하기 위하여 기본블럭을 l-block이라고 하는 더 작은 단위로 나누었다. 각 기본블럭 내에서 같은 캐쉬 블럭에 사상되는 명령어 군을 하나의 l-block이라고 한다. 각 l-block들 사이의 캐쉬 접근 충돌 가능성을 조사하여 이를 반영한 정수 선형식을 새로 정의하여 프로그램의 최악 실행시간을 분석하였다.

최악 실행시간 분석을 행함에 있어서 데이타 캐쉬의 영향을 고려하게 되면 명령어 캐쉬의 경우와는 달리 다음과 같은 문제점들을 고려해야만 한다.

① 각 명령어의 주소는 프로그램의 시작 주소만 주어지면 계산이 가능하지만, 데이타 참조 주소는 그렇지 않은 경우도 있다. 참조되는 데이타가 전역적인 것이 아니고 어떤 프로시쥬어 내에서 사용되는 지역적인 변수에 의한 경우일 때에는 그 프로시쥬어가 호출된 프로그램 수행 위치까지도 고려해야 한다. 지역 변수는 전역 데이타 영역에 저장되지 않고 해당 프로시쥬어가 호출될 때마다 스택상에 만들어지는 활성 레코드 영역에 저장되기 때문에 참조 주소가 매 호출에 따라 달라지는 것이다.

② 명령어 참조에 대한 모든 주소는 정적으로 결정될 수 있지만, 데이타 참조에 대한 주소는 그렇지 않은 경우가 있다. 예를 들면, 순환문(loop) 내에서 참조되는 배열 변수의 주소와 포인터 변수의 주소 등이다.

데이타 캐쉬에 의한 영향을 고려한 연구는 아이오와 주립대학(Iowa State University)의 Rawat[13]에 의해서 처음으로 수행되었다. Rawat은 이 연구에서 데이타 변수의 활동 영역(live range)의 개념을 이용하여 정적으로 데이타 캐쉬 접근 실패의 최대 갯수를 계산해 내는 기법을 제안하였다. 즉, 변수의 활동 영역이 다른 변수의 활동 영역과 충돌하지 않는다면 그 변수에 대한 참조에 의해서는 데이타 캐쉬 접근 실패가 발생하지 않는다는 성질을 이용하여, 서로 다른 활동 영역간 충돌의 총 횟수를 계산, 전체 프로그램 수행 중 발생 가능한 데이타 캐쉬 접근 실패 갯수의 최대치를 구한다. 이 기법은 다음과 같은 몇가지 문제점을 안고 있다. 첫째, 이 기법에서는 최악 실행 경로를 고려하지 않기 때문에 데이타 캐쉬 접근 실패 갯수에 대한 최대치가 과예측될 수 있다. 둘째, 이 기법에서는 순환문 내에서 참조되는 배열 데이타에 대해서 별다른 분석 방법을 제시하지 않고 있다. 따라서 수학 계산이 많은 프로그램의 경우 데이타 캐쉬 접근 실패 횟수에 대한 예측이 지나치게 크게 나올 수 있다.

한편, 서울대학교 실시간 시스템 연구 그룹에서 발표한 확장된 타이밍 스키마[5, 6, 2]에서는 데이타 참조에 대해서 명령어 참조의 경우와 마찬가지로 각 실행블럭마다 최초 참조 정보와 최종 참조 정보를 구성한다. 이들 정보의 구성시 지역 변수의 주소를 파악하기 위해서 프로시쥬어의 호출 시나리오를 표현한 호출 그래프(call graph)를 이용한다. 반면, 컴파일 단계에서 데이타 참조 주소를 결국 알아내지 못하는 경우, 즉 순환문 내의 배열 데이타 참조나 포인터 변수에 의한 데이타 참조 등의 경우에 대해서는 2개의 데이타 캐쉬 참조 실패를 부과하는 정책을 취하고 있다. 여기서 한 개는 자기 자신이 캐쉬 접근 실패일 것이라 가정하여 부과한 것이고 다른 한 개는 자신에 의해서 캐쉬에서 쫓겨났을지도 모르는 유용한 데이타(useful data)에 대한 것이다. 이 기법을 따르면 배열에 대한 참조가 많은 프로그램을 분석할 경우에는 최악 실행 시간이 과예측되는 문제가 있다. 같은 연구 그룹의 Kim 등의 연구[3]에서는 확장 타이밍 스키마 기법이 안고 있는 위와 같은 문제점을 극복하기 위하여 데이타 캐쉬 분석 부분에 대하여 다른 추가적인 기법을 사용한다. 이 연구에서는 순환문 내의 각 배열 참조에 대해서 참조 영역(reference region)을 정의하는데, 참조 영역은 그 참조 명령어에 의해서 생성되는 모든 참조 주소들의 집합이 된다. 이 참조 영역과 피전홀(Pigeonhole) 원리를 이용하여 데이타 참조 분석의 정확성을 높인다. 하지만, 이 기법에서도 포인터 변수에 대한 분석은 고려하지 않고 있다.

표 1 최악 실행시간 분석 기법들의 비교

	타이밍스키마	확장된 타이밍스키마	플로리다 주립대학	프린스턴대학	비엔나대학
파이프라인분석	×	○	○	○	×
명령어캐쉬분석	×	○	○	○	×
데이타캐쉬분석	×	○	×	×	×
분석을 위해 필요한 정보	구문 트리	구문 트리	제어 흐름도	제어 흐름도	구문 트리
타이밍 분석을 위한 새로운 언어 구조	×	×	×	×	○
동적 경로 분석	○	×	×	○	○
검증 방법	실측	실측	시뮬레이션	실측	×
검증 대상 프로세서	Motorola MC68010	MIPS R3000/R3010	MicroSPARC I	Intel i960KB	

4. 최악 실행시간 분석 기법의 비교

여기서는 지금까지 소개한 최악 실행시간 분석 기법들의 특징들을 비교한다. 비교 대상이 되는 기법들은 모두 이를 구현한 실행시간 분석기가 개발되어 있는 것들이다. 실행시간 분석기를 통해 분석한 최악 실행시간이 실제 최악 실행시간과 얼마만큼의 차이를 보이는지 조사하기 위해서는 검증 환경이 필요하다. 실제 프로세서가 장착된 환경에서 실제 프로그램을 실행하고 이를 측정한 결과와 분석 결과를 비교하는 것이 가장 타당한 검증 방법이라고 할 수 있다. 그러나 이러한 실측 환경 구현이 어려울 때에는 시뮬레이션과 같은 소프트웨어적인 방법을 사용하기도 한다. 표 1은 이상 설명한 여러 요소에 대한 비교 결과이다.

5. 결 론

프로그램의 분석 방법은 제어 흐름도에 기반한 방법과 프로그램 구문 트리에 기반한 방법의 두 방향이 존재하고, 각기 장단점을 가지고 있다. 본 논문에서는 최근까지 발표된 최악 실행시간 분석 기법들 중에서 제어흐름도에 기반한 방법으로 플로리다 대학과 프린스턴 대학의 기법, 그리고 프로그램의 구문 트리에 기반한 방법으로 Park과 Shaw의 타이밍 스키마, 서울대학교의 확장된 타이밍 스키마, 그리고 비엔나 대학의 MAXT 기법을 소개하고 이들을 비교하였다.

최근의 최악 실행시간 분석 기법의 흐름은 프로세서의 발전된 구조를 분석에 반영하고자 하는 것이다. 최근의 기법들에서는 파이프라인 실행 구조나 캐쉬와 같이 기존에는 실행시간 분석시 고려되지 않았던 성능 향상 구조들의 실행 행태를 파악하고 이를 분석에 반영하고 있다.

본 논문에서는 실행시간 분석기를 구현하여 분석 기법의 실제적인 적용을 가능하게 한 분석 기법들을 선택하여 이들의 특징을 프로그램을 분석하는 방향과 프로세서의 파이프라인 실행과 캐쉬 동작을 반영하는 기법에 기준을 두어 설명하고, 이를 통해 최악 실행시간 분석 기법의 현재의 발전 단계를 파악하였다.

참고문헌

[1] C. A. Healy, D. B. Whalley, and M. G. Harmon, "Integrating the Timing Analysis of Pipelining and Instruction Caching," *In Proceedings of the 16th Real-Time Systems Symposium*, pp. 288-297, December 1995.

[2] Y. Hur, Y. H. Bae, S.-S. Lim, S.-K. Kim, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, and C. S. Kim, "Worst Case Timing Analysis of RISC Processors : R3000/R3010 Case Study," *In Proceedings of the 16th Real-Time Systems Symposium*, pp. 308-321, 1995.

[3] S.-K. Kim, S. L. Min, and R. Ha, "Efficient Worst Case Timing Analysis of Data Caching," In *Proceedings of the 2nd Real-Time Technology and Applications Symposium*, pp. 230-240, June 1996.

[4] Y. S. Li, S. Malik, and A. Wolfe, "Efficient Microarchitecture Modeling and Path Analysis for Real-Time Software," In *Proceedings of the 16th Real-Time Systems Symposium*, pp. 298-307, December 1995.

[5] 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 for RISC Processors," In *Proceedings of the 14th Real-Time Systems Symposium*, pp. 97-108, December 1994.

[6] 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 for RISC Processors," *IEEE Transactions on Software Engineering*, Vol. 21, No. 7, pp. 593-604, July 1995.

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

[8] F. Mueller and D. B. Whalley, "Efficient On-the-fly Analysis of Program Behavior and Static Cache Simulation," In *Proceedings of Static Analysis Symposium*, pp. 101-115, September 1994.

[9] C. Y. Park and A. C. Shaw, "Experiments With A Program Timing Tool Based On Source-Level Timing Schema," In *Proceedings of the 11th Real-Time Systems Symposium*, pp. 72-81, December 1990.

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

[11] P. Puschner and C. Koza, "Calculating the Maximum Execution Time of Real-Time Programs," *Real-Time Systems*, Vol. 1, No. 2, pp. 159-176, September 1989.

[12] 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.

[13] J. Rawat, *Static Analysis of Cache Performance for Real-Time Programming*, Master's thesis, Iowa State University, 1993.

[14] A. C. Shaw, "Reasoning About Time in Higher-Level Language Software," *IEEE Transactions On Software Engineering*, Vol. 15, No. 7, pp. 875-889, July 1989.

[15] 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.

임 성 수

1993 서울대학교 공과대학 컴퓨터공학과(공학사)
1995 서울대학교 대학원 컴퓨터공학과(공학석사)
1995~현재 서울대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 컴퓨터 구조, 실시간 시스템

민 상 렬

1983 서울대학교 컴퓨터공학과 졸업
1985 서울대학교 컴퓨터공학과 석사
1989 워싱턴 주립대학 전산학 박사
1989~90 IBM T.J. Watson Research Center객원 연구원.
1990~92 부산대학교 컴퓨터공학과 조교수
1992~현재 서울대학교 컴퓨터공학과 조교수
관심분야 : Computer Architecture, Parallel Processing, Computer Performance Evaluation 임