다중 이슈 프로세서를 위한 최악 실행시간 분석 기법

(A Worst Case Execution Timing Analysis Technique for Multiple-Issue Processors)

임 성 수 † 한 정 희 †† 김 지 홍 ††† 민 상 렬 †††

(Sung-Soo Lim) (Jung Hee Han) (Jihong Kim) (Sang Lyul Min)

요 약 본 논문에서는 한 번에 여러 개의 명령어를 이슈할 수 있는 다중 이슈 프로세서(in-order, multiple-issue processors)에 대해 최악 실행시간을 분석하는 기법을 제시한다. 명령어들의 이슈 행태를 분석하기 위해서 명령어들 사이의 의존성 관계를 표현하는 IDG(Instruction Dependence Graph)라고 하는 자료구조를 사용한다. 이 자료구조로부터 각 명령어들의 이슈간 거리 범위를 구하고, 프로그램의 계층적인 분석 과정에서 점차로 더 정확한 이슈간 거리 범위로 갱신한다. 프로그램의 최악 실행시간은 최종적으로 얻어진 프로그램 전체에 대한 IDG를 분석하여 얻은 명령어들의 이슈간 거리 범위로부터 계산한다. 제안하는 기법을 구현한 시간 분석기를 사용하여 실험한 결과, 논문에서 사용한 다중 이슈 프로세서 모델에 대해서 정확하게 다중 이슈 행태를 분석할 수 있었다.

Abstract In this paper, we present a worst case timing analysis technique for in-order, multiple-issue processors. In the proposed technique, timing information for each program construct is represented by a directed acyclic graph (DAG) that encodes dependences among nodes (i.e., instructions). From this information, we calculate for each pair of instructions distance bounds between their issue times. The worst case execution time of a program is calculated by progressively refining the distance bounds through hierarchical analysis over the program's syntax tree. Our experimental results show that the proposed technique can predict the worst case execution times for in-order, multiple-issue processors as accurately as for simpler RISC processors.

1. 서 론

실시간 시스템에서 스케쥴 가능성 분석을 할 때에는 각 태스크의 최악실행시간(WCET: Worst Case Execution Time)을 사전에 파악해야 한다. 분석된 WCET는 실제 WCET보다는 항상 커야 하는 안전성과 함께, 실제 WCET와의 차이가 가능하면 작아야 하는 정확성을 아울러 갖추어야 한다. 실제 WCET보다 작은 WCET 분석 결과를 사용하여 시스템을 구성하는 경우 예기치 않게 태스크가 종료시한(deadline)을 어기게 됨으로써 심각한 오류를 초래하게 되며, 실제 WCET와의 차이가 지나치게 큰 WCET 분석 결과를 사용하는 경우에는 시스템의 자원을 비효율적으로 사용하게 된다. 따라서, 보다 정확한 WCET를 분석하고자 하는 연구가 최근 여러 연구 그룹에서 진행되어 왔다. 정확한 WCET를 분석하려면 캐쉬, 파이프라인 실행 등 분석 대상 프로세서의 하드웨어 요소에 의한 시간적인 특성을 고려하여

실행시간 분석에 반영하여야 한다. 최근 제안된 WCET 분석 기법들 중 파이프라인 실행을 분석하고자 한 연구로는 Zhang, Burns, Nicholson [18], Harmon, Baker, Whalley [6], Narasimhan, Nilsen [13], Arnold 등 [1], Healy 등 [8], Lim 등 [11], 그리고 Ottosson, Sjödin [14]의 연구가 있다. 이들 연구에서는 한 사이클에 하나의 명령어를 이슈하는 프로세서에 대해서만 분석하였기 때문에 이들 연구 결과를 한번에 여러 개의 명령어를 이슈하는 다

· 본 연구는 한국과학재단 (97-01-02-05-01-3)의 지원을 받았습니다.

† 비 회 원 : 서울대학교 컴퓨터공학과
sslim@archi.snu.ac.kr
†† 비 회 원 : 미시간주립대학교 전기컴퓨터공학과
jungheeh@eecs.umich.edu
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
jihong@davinci.snu.ac.kr
symin@dandelion.snu.ac.kr

논문접수 : 1999년 10월 26일
심사완료 : 2000년 7월 10일

중 이슈 프로세서에는 적용하기 힘들다. 본 논문에서는 확장된 타이밍 스키마(ETS: Extended Timing Schema) [11]에 기반한 다중 이슈 프로세서를 위한 WCET 분석 기법을 제안한다. 다중 이슈 프로세서에서 WCET 분석을 하기 위한 핵심적인 문제는 함께 이슈될 수 있는 명령어들을 어떻게 파악할 수 있는가이다. 이런 문제의 해결을 위해서 본 논문에서는 프로그램의 명령어들 사이의 관계를 IDG (Instruction Dependence Graph)로 표현하는 방법을 사용하였다. IDG를 사용하여 하나의 프로그램에 속해 있는 명령어들 사이의 의존성(dependences) 관계를 표현한 후, 이를 이용하여 명령어들 사이의 이슈간 거리 범위(distance bounds)를 계산한다. 명령어들 사이의 이슈간 거리 범위는 초기에는 각 기본 블럭에 대해서 인접하는 다른 기본 블럭들 내의 명령어들을 고려하지 않고 계산하고, ETS의 계층적인 분석 과정에서 인접하는 다른 블럭들의 명령어들이 알려지면 이들의 실행 행태를 반영하여 차차 보다 정확한 정보로 갱신한다. 분석의 복잡도를 줄이기 위해서 본 논문에서는 이슈간 거리가 고정되는 노드들을 하나의 노드로 묶어서 분석에 이용하는 방법을 사용한다. 이는 각 IDG에서 유지해야 하는 노드의 개수를 줄임으로써 분석의 복잡도를 줄인다.

본 논문에서는 제안하는 기법을 간단한 다중 이슈 프로세서 모델에 적용하였다. 사용한 다중 이슈 프로세서 모델에서는 명령어가 프로그램에 나타난 순서대로 이슈(in-order issue)된다고 가정한다. 또한, 캐쉬나 분기 예측(branch prediction) 등에 대한 고려는 하지 않고 다중 이슈 동작에 대한 분석에 초점을 맞춘다.

제안하는 기법의 검증을 위해서 기법을 구현한 시간 분석기를 구현하였으며 이 시간 분석기에 의한 분석 결과를 본 논문에서 사용하는 프로세서 모델의 시뮬레이터의 실행 결과와 비교하였다. 실험 결과를 통해 본 논문에서 제안하는 기법이 다중 이슈 동작을 정확하게 분석할 수 있으며, 본 논문에서 사용한 프로세서 모델에 대해 실제 실행 결과에 근접하는 WCET 분석 결과를 얻을 수 있음을 보인다.

이 논문의 구성은 다음과 같다. 2 장에서는 WCET 분석에 관한 관련 연구를 살펴 보며, 3 장에서는 본 논문에서 제안하는 기법의 기반이 되는 ETS에 대해 설명하고, 4 장에서는 사용한 다중 이슈 프로세서 모델을 보인다. 5 장에서는 명령어의 이슈간 거리 범위를 구하는 절차를 설명하고 이를 통해 어떻게 함께 이슈될 수 있는 명령어를 파악할 수 있는 지 설명한다. 6 장에서는 명령어의 이슈간 거리 범위를 구하는 절차를 ETS에 통합하는 방법을 기술하고, 7 장에 실험 결과를 보인다. 8 장에서 본 논문의 결론을 기술한다.

2. 관련연구

실시간 시스템 연구 분야 중 프로그램의 WCET 분석은 최근 중요한 연구 분야로 간주되어 왔다. 초기의 연구들은 주로 프로그램의 시간 분석을 위한 이론적인 틀을 확립하는 데에 초점을 맞추고 있다. Shaw는 타이밍 스키마(timing schema)라고 하는 WCET 분석 틀을 제안하였다 [16]. 타이밍 스키마에서는 고급 언어로 씌어져 있는 프로그램의 다양한 구문 구조에 대한 분석식을 제공한다. 예를 들면, 순차 구문 $S: S_1; S_2$에 대한 분석식은 $T(S) = T(S_1) + T(S_2)$ 로 주어진다. 여기서, $T(S)$, $T(S_1), T(S_2)$는 각각 S, S_1, S_2의 실행시간 범위를 나타낸다. 이런 식으로 각각의 구문 구조에는 서로 다른 WCET 분석식이 정의된다. 프로그램의 WCET는 이 분석식들을 프로그램의 구문 트리(syntax tree)에 계층적으로 적용해 가면서 구하게 된다. Puschner와 Koza [15], Mok [12] 역시 비슷한 형태의 WCET 분석식을 제안하였다. 위의 연구들의 공통적인 문제는 최근 프로세서에서 일반적으로 채용하고 있는 파이프라인이나 캐쉬 등의 구조에 대해 고려하지 않고 있다는 것이다. 위의 기법들에서는 이들 구조가 미치는 시간적인 영향을 비관적으로 가정해야 하기 때문에 실제 실행시간과의 차이가 큰 분석 결과를 출력하게 된다. 따라서, 최근 제안되는 기법들에서는 파이프라인, 캐쉬 등의 구조를 고려하여 WCET 분석을 시도하였다. 이러한 연구의 예로는 Zhang, Burns, Nicholson [18], Lim 등 [11], Healy 등 [7], Li, Malik, Wolfe [10], Ottosson과 Sjödin [14], Ferdinand, Martin, Wilhelm [4], Theiling과 Ferdinand [17] 등의 연구가 있다.

Zhang, Burns, Nicholson은 80C188 프로세서의 명령어 실행과 명령어 선인출(prefetch)의 중첩 실행을 분석하였다. Lim 등은 파이프라인과 캐쉬 구조에 의한 영향을 고려하여 타이밍 스키마를 확장한 확장된 타이밍 스키마(ETS)를 제안하였다. ETS의 자세한 설명은 다음 장에 이어진다. Healy 등은 파이프라인 실행과 캐쉬 접근 실패로 인한 지연 시간의 중첩을 고려한 WCET 분석 기법을 제안하였다. Li, Malik, Wolfe는 WCET 분석을 정수 선형 계획법(ILP: integer linear programming)으로 풀었다. 이 기법에서는 사용자가 제공하는 프로그램의 실행 경로에 관한 정보를 사용하여 보다 정확한 WCET 분석 결과를 얻을 수 있는 방안을 제공한다. Ferdinand, Martin, Wilhelm은 추상적 해석(abstract interpretation) 기법을

사용하여 프로그램의 WCET를 분석하는 기법을 제안하였다. Theiling, Ferdinand는 Li 등이 제안한 ILP를 이용한 분석 기법과 추상적 해석을 이용한 기법을 절충한 WCET 분석 기법을 제안하였다. 이상의 모든 기법들은 한 사이클당 하나의 명령어만을 이슈할 수 있는 프로세서를 가정하고 있기 때문에 이 기법들을 그대로 다중 이슈 프로세서에 적용할 수 없다. 대표적인 다중 이슈 프로세서인 수퍼스칼라 프로세서를 위한 분석 기법으로는 Harcourt, Mauney, Cook의 연구 [5]와 Choi, Lee, Kang의 연구 [2]가 있다. 이들 기법들은 주어진 명령어 순차에 대한 실행시간 분석에 초점을 맞추고 있으며 일반적인 프로그램에 대한 시간 분석은 고려하지 않는다.

3. 확장된 타이밍 스키마(ETS)

본 장에서는 본 논문에서 제안하는 기법의 기반이 되는 ETS에 대해 설명한다. ETS는 기존의 계층적인 프로그램의 실행시간 분석 기법인 타이밍 스키마를 확장한 것으로 파이프라인, 캐쉬 등의 프로세서 구조들에 의한 영향을 고려하였다. ETS에서는 파이프라인, 캐쉬 등을 고려하기 위해 기존 타이밍 스키마의 다음 두가지 요소를 수정한다. 첫째, 각 프로그램 실행블럭(이는 기본블럭, 혹은 여러 기본 블록의 집합이 될 수도 있다.)에 대한 실행 시간 정보에 파이프라인, 캐쉬 등의 실행 형태 정보를 수록한다. 둘째, 각 구문 구조에 대한 분석식을 새로 정의한 실행 시간 정보를 이용하여 수정한다. 새로 정의되는 분석식에는 접속(concatemation)과 절지(pruning)연산이 새로 정의되어 이용된다.

ETS에서 새로 정의하는 실행시간 정보를 WCTA (Worst Case Timing Abstract)라고 부른다. 이 정보에는 계층적인 실행시간 분석을 위해 필요한 자료 구조를 유지한다. 각 프로그램 구문 구조 별로 유지되는 WCTA에는 해당 구문 구조에 속해있는 모든 실행 경로에 대한 정보를 유지한다. WCTA 내에 들어 있는 각 실행 경로에 대한 정보를 각각 PA (Path Abstract)라고 부른다. 예를 들면, 조건문(if 문)이 있는 구문의 경우에는 then 경로와 else 경로에 대한 실행시간 정보(PA)를 일단 함께 유지한다. 이들 경로의 정보 중에서 어떤 경우에도 최악 실행경로에 속할 가능성이 없는 경로에 대한 정보는 절지 연산을 통해 삭제한다. 이러한 절지연산이 가능한 시점은 인접한 프로그램 구문들에 대한 실행시간 정보가 파악되고, 이들 인접 프로그램 구문에 의한 영향으로 현재 분석중인 프로그램 구문의 실행 행태가 어떻게 변할지 알려지는 시점이다. 따라서, PA 정보는 계층적인 분석 과정에서 점차로 더 정확한 정보로 갱신될 수 있는 형태로 정의되어야 한다. [11]에서는 파이프라인 분석을 위해 자원예약표(reservation table)라고 하는 자료 구조를 PA 내에 포함시켰다. 이 자료 구조는 인접한 프로그램 구문들 사이의 파이프라인 실행으로 인한 중첩 실행 행태를 반영할 수 있다.

ETS에서 분석식에 사용하기 위해 새로 정의한 두 개의 연산은 접속 연산과 절지 연산이다. 이 중 접속 연산은 두 개의 PA를 접속하여 하나의 PA로 만들기 위한 연산으로 두 개의 프로그램 구문이 연달아 실행되는 상황을 모델링하기 위한 것이다. 파이프라인 실행을 고려하면 두 개의 프로그램 구문이 연달아 실행되는 경우의 실행시간이 각각 프로그램 구문의 실행시간의 단순 합산이 되지 않게 되는데, 접속 연산은 이를 반영하여 정의된다. 절지 연산은 하나의 WCTA에 속해있는 여러 개의 PA들 중에서 어떤 경우에도 최악 실행 경로에 속할 가능성이 없는 경로를 가려 내어 해당 PA를 WCTA에서 삭제한다. 표 1은 ETS의 각 구문 구조에 따른 실행시간 분석식을 보인다.

표 1 ETS의 실행시간 분석식

Statement	*Timing Formulas*
$S: S_1; S_2$	$W(S) = W(S_1) \oplus_p W(S_2)$
S: if (exp) then S_1 else S_2	$W(S) = (W(\exp) \oplus_p W(S_1)) \cup (W(\exp) \oplus_p W(S_2))$
S: while (exp) S_1	$W(S) = (\oplus_p^N {}_1 (W(\exp) \oplus_p W(S_1))) \oplus_p W(\exp)$
S: f(exp $_1$, ..., exp $_n$)	$W(S) = W(\exp_1) \oplus_p \ldots \oplus_p W(\exp_n) \oplus_p W(f())$

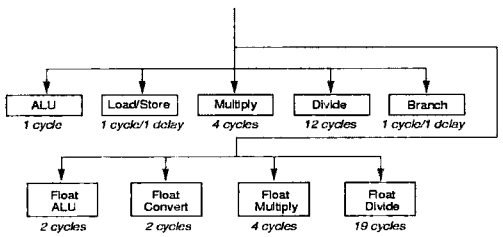


그림 1 다중 이슈 프로세서 모델

요약하면, ETS를 특정 프로세서에 대해 적용하려면 다음 세 단계의 과정이 필요한다. 첫째, PA 구조를 다시 정의해야 하고, 둘째, 접속 연산을 다시 정의하며, 마지막으로 절지 연산을 다시 정의한다. 이상의 세 단계가 완료되

면 대상 프로세서에서 실행되는 프로그램의 WCET를 이미 정의되어 있는 분석식을 이용하여 분석할 수 있게 된다.

4. 다중 이슈 프로세서 모델과 프로그램의 표현

4.1 다중 이슈 프로세서 모델

그림 1은 본 논문에서 사용하는 다중 이슈 프로세서 모델을 보인다. 이 프로세서 모델은 9 개의 실행 유닛(functional unit)을 갖추고 있으며 명령어 집합의 MIPS R3000/R3010의 명령어 집합을 사용한다. 그림에는 각 실행유닛에서 명령어가 실행될 때의 소요시간(latency)를 함께 보인다. 프로세서 모델과 관련하여 다음과 같이 가정한다.

* 명령어들은 프로그램에 나타난 순서대로 이슈된다(in-order issue). 그러나 명령어들의 실행은 프로그램에 나타난 순서와 관계없이 종료될 수 있다(out-of-order completion).
* 분기 명령에 대해서 지연 분기 기법(delayed branch)을 가정한다. 여기서 지연 슬롯(delay slot)은 한 사이클로 가정한다. 메모리 로드 명령의 경우에도 한 사이클의 지연시간을 갖는다고 가정한다.
* 모든 메모리 참조는 캐쉬 접근 성공(cache hit)한다고 가정한다. 이 가정은 캐쉬 접근 실패로 인한 실행시간의 변화를 본 논문의 고려 대상에서 제외하기 위함이다.
* 프로세서는 최대 k 개의 명령어를 함께 이슈할 수 있는데, k는 전체 실행 유닛의 개수보다 작거나 같아야 한다. (참고로, 본 논문에서 k는 9보다 작거나 같아야 한다).

4.2 프로그램의 표현

수퍼스칼라 프로세서에서 명령어들의 다중 이슈 행태를 분석하기 위해서는 명령어들 사이의 의존성을 파악해야 한다. 한 명령어의 이슈는 이 명령어가 사용하는 모든 자원들이 사용 가능해질 경우 이루어지기 때문에 필요한 자원이 선행하는 다른 명령어에 의해 점유되어 있으면 그 명령어는 이슈될 수 없다. 명령어들 사이의 의존성 [9]에는 구조적인 의존성(structural dependence), 데이타 의존성(data dependence), 그리고 제어 의존성(control dependence) 등이 있다. 이들 의존성을 표현하기 위해서 본 논문에서는 IDG (Instruction Dependence Graph)를 사용한다. 이 IDG에는 명령어들이 노드로, 명령어들 사이의 의존성이 간선으로, 해당 의존성으로 인한 지연시간(latency)이 간선의 무게로 표현된다. 이 간선의 무게는 해당 노드들이 나타내는 명령어들의 이슈 사이에 경과해야 하는 지연시간을 의미한다. 이 지연시간은 해당 명령어들이 실행할 때 필요한 프로세서 내의 실행 유닛(functional unit)을 점유하는 시간으로 결정된다.

기존 ETS의 PA에서 사용한 자원예약표(reservation table) 대신 IDG를 사용하는 이유를 설명하기 위해 그림 2에 자원예약표의 예를 보인다. 그림에서 각 행은 파이프라인 실행 단계를 나타내며 각 열은 실행 사이클을 나타낸다. x 표시된 부분은 해당 실행 사이클에 사용되는 파이프라인 실행단계를 나타낸다. 그림에 나타나 있듯이 자원예약표가 구성될 때에는 명령어들의 실행 행태가 고정되는 것으로 가정한다. 이렇게 명령어의 실행 행태를 고정하게 되면, 인접한 프로그램 구문의 실행 행태에 따라 실행시간 변화가 큰 다중 이슈 프로세서에 대해서는 정확한 분석을 행하기 어렵게 된다. 따라서, 본 논문에서는 IDG를 정의하여 PA로 사용한다.

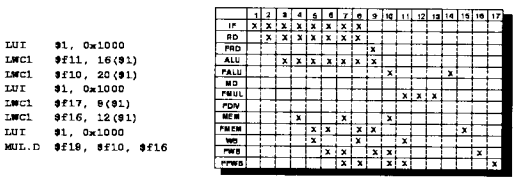


그림 2 자원예약표의 예

그림 3에 IDG의 예를 보인다. 그림 3의 IDG에 나타나 있듯이 제어 의존성을 제외한 명령어들 사이의 의존성들이 간선으로 표현된다. 제어 의존성을 제외하는 이유는 본 논문에서 가정하는 수퍼스칼라 프로세서 모델에서는 지연된 분기(delayed branch)를 가정하여 분기 명령어들을 특별히 고려할 필요가 없기 때문이다. 일반적으로 고려되는 의존성들 외에 본 논문에서는 순서 의존성(order dependence)이라고 하는 새로운 의존성 관계를 도입하였다. 이는 본 연구에서 사용하는 프로세서 모델이 순서적으로 명령어들을 이슈한다는 가정을 하고 있기 때문이다. 이러한 가정은 비순서적인 이슈(out-of-order issue)를 고려하는 데 부과되는 분석의 복잡성을 피하고, 일단 보다 간단한 모델의 분석에서 시작하여 이를 확장하기 위함이다. 순서 의존성으로 인한 지연시간은 0으로 가정한다. 그림 3(a)는 MIPS R3000/R3010 명령어 집합을 사용한 명령어 순차의 예를 보이고, 그림 3(b)는 그림 3(a)의 명령어들 사이의 의존성을 모두 하나의 그래프에 표현한 것이다. 그림 3(c)는 그림 3(b)의 의존성들 중에서 고려하지 않아도 되는 의존성들(redundant dependences)을 제외한 의존성들만을 표현한 IDG이다. 고려하지 않아도 되는 의존성들은 다른 의존성들의 지연시간에 의해서 그 영향이 실제로 나타나지 않는 의존성들로서 각 노드의 쌍에는 하나의 의존성만을 제외한 나머지 의존성들은 모두 고려하

지 않아도 되는 의존성이 된다. IDG에서 고려해야 하는 의존성 및 고려하지 않아도 되는 의존성을 이론적으로 정리하면 다음과 같다:

두 노드 v와 w 사이의 의존성 e=(v, w)는 다음의 조건을 만족하면 고려하지 않아도 된다.

* 같은 노드 v와 w 사이에 또다른 의존성 e'=(v, w)가 있고 이 의존성의 간선의 무게가 의존성 e의 무게보다 크거나 같은 경우, 혹은

*다음 조건을 만족하는 의존성 집합, $S=\{e_0, e_1, \ldots, e_p\}$ 이 존재할 때이다.

(i) $e_0=(v, i_0), e_p=(i_{p-1}, w)$, 이고

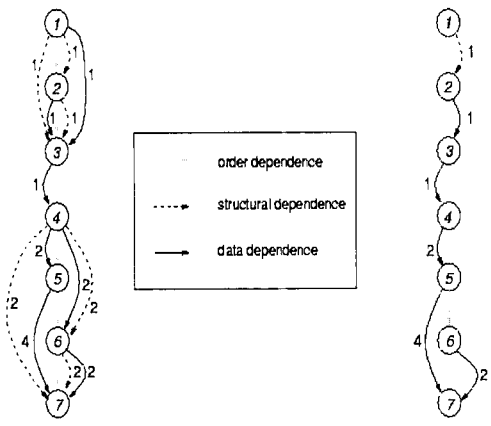
$e_j=(i_{j-1}, i_j)(1\le j<p)$, 그리고

(ii) $\sum_{i=0}^{p} l_{e_i} \ge l_e$.

이상 설명한 IDG가 표현하는 정보는 해당 IDG의 노드들에 대응되는 명령어들 사이의 의존성 관계로서 이들 명령어들의 이슈 행태를 파악하고 명령어들이 구성하는 프로그램의 최악 실행시간을 구하는 데 기초가 된다.

```
1  sll   $25, $12, 0x2       ; $25 <- $12 << 2
2  lui   $1, 0x1000          ; $1  <- 0x1000
3  addu  $1, $1, $25         ; $1  <- $1 + $25
4  lwc1  $f18, 96($1)        ; $f18 <- mem[$1 + 96]
5  mul.s $f15, $f18, $f18    ; $f16 <- $f18 * $f18
6  add.s $f18, $f18, $f18    ; $f18 <- $f18 + $f18
7  add.s $f18, $f18, $f16    ; $f18 <- $f18 + $f16
```

(a)



(b) (c)

그림 3 IDG의 예: (a) 기본블럭 예, (b) 모든 의존성이 표현된 그래프, (c) 고려하지 않아도 되는 의존성이 제거된 IDG

어떤 프로그램 구문에 대한 IDG를 대상으로 명령어들 사이(노드들 사이)의 이슈 행태를 분석할 때에는 인접한 다른 프로그램 구문에 대한 IDG 정보를 알지 못한 상태이기 때문에, 최악의 시나리오를 가정하여 이슈 행태를 분석한다. 인접한 프로그램 구문의 정보를 알게 되면 보다 정확한 정보로 갱신하게 된다. 이러한 명령어들 사이의 이슈 행태를 결정짓는 요소는 명령어들 사이의 의존성이고, 인접한 프로그램 구문에 대한 정보가 알려지면 밝혀지지 않았던 추가의 명령어들 사이의 의존성이 밝혀지게 되기 때문에 보다 정확하게 이슈 행태를 분석할 수 있다. 본 논문에서는 IDG 내의 노드들의 이슈 행태 분석을 위해 노드들을 두 부류로 나눈다. 첫째 부류는 의존성이 결정된 노드들(resolved nodes)이고, 두번째 부류는 의존성이 결정되지 않은 노드들(unresolved nodes)이다.

의존성이 결정되었다고 하는 것은 해당 노드에 대응되는 명령어가 사용하는 모든 자원(여기서는 레지스터, 프로세서의 실행 유닛을 통칭하여 자원이라고 부른다.)을 같은 IDG 내에서 먼저 사용하는 노드들의 존재가 밝혀져 있다는 말이다. 즉, 해당 노드가 사용하는 모든 자원을 이미 같은 IDG 내의 다른 노드가 사용하고 있기 때문에 그 노드에 대응되는 명령어가 언제 이슈될 수 있는 지는 같은 자원을 먼저 사용하는 다른 노드들의 이슈 행태를 파악할 경우 알 수 있다는 것이다. 의존성이 결정되지 않았다고 하는 것은 해당 노드에 대응되는 명령어가 사용하는 자원 중 일부는 같은 IDG 내에서 그 노드에 의해 처음 사용되는 것으로서 이 노드의 이슈 행태는 아직 밝혀지지 않은 선행하는 프로그램 구문의 다른 노드의 이슈 행태가 파악되지 않으면 정확하게 알아낼 수 없다는 말이다. 따라서 인접 프로그램 구문의 실행 정보가 밝혀지지 않은 상태에서는 의존성이 결정되지 않은 노드들에 대해서는 최악의 상황을 가정하여 가능한 가장 큰 지연시간 뒤에 이슈가 가능하다고 가정한다. 이렇게 가정된 의존성이 결정되지 않은 노드들의 이슈 행태는 인접 프로그램 구문 내의 다른 노드(명령어)와의 의존성이 밝혀지면 갱신된다. 의존성이 결정된 노드와 의존성이 결정되지 않은 노드는 다음의 정의에 따라 파악한다.

정의 1

어떤 노드 내의 명령어가 소스 레지스터로 특정 레지스터를 사용하면 해당 노드가 이 레지스터를 use한다고 말한다. use(i)는 노드 i가 use하는 모든 레지스터의 집합을 뜻한다.

정의 2

어떤 노드 내의 명령어가 목적 레지스터로 특정 레지스터를 사용하면 해당 노드가 이 레지스터를 define한다고 말한다. def(i)는 노드 i가 define하는 모든 레지스터의 집합을 뜻한다.

정의 3

어떤 노드 내의 명령어가 실행시 특정 실행 유닛을 점유하면 해당 노드가 이 실행 유닛을 occupy한다고 말한다. occupy(i)는 노드 i가 occupy하는 모든 실행 유닛의 집합을 뜻한다.

정의 4

어떤 노드 i에 대한 use(i), def(i)에 속하는 모든 레지스터가 같은 IDG 내의 다른 노드에 의해 define되어 있고, occupy(i)에 속하는 모든 실행 유닛이 역시 같은 IDG 내의 다른 노드에 의해 occupy되어 있을 때, 노드 i를 의존성이 결정된 노드(resolved node)라고 말한다. 그렇지 않을 경우 노드 i를 의존성이 결정되지 않은 노드(unresolved node)라고 말하고, 노드 i는 결정되지 않은 의존성(unresolved dependences)을 가지고 있다고 말한다.

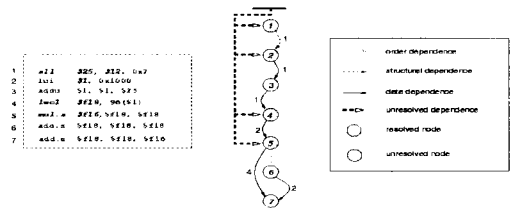


그림 4 의존성이 결정된/결정되지 않은 노드들

그림 4는 그림 3에 제시된 IDG를 의존성이 결정된 노드와 의존성이 결정되지 않은 노드를 구분하여 다시 보이고 있다. 그림 4에서 노드 1의 경우 명령어 sll이 사용하는 실행유닛 ALU와 레지스터 $25 및 $12를 이 IDG 내에서 처음 사용하고 있으므로 이들 자원에 의해 의존성이 결정되지 않은 노드(unresolved node)로 분류된다. 마찬가지로 노드 2, 노드 4, 그리고 노드 5의 경우에도 의존성이 결정되지 않은 노드로 분류된다.

5. 명령어의 이슈간 거리 범위

이 장에서는 IDG를 사용하여 명령어 순차의 실행시간을 예측하고, 함께 이슈될 수 있는 명령어를 파악하는 방법을 기술한다. 먼저, IDG 표현이 주어졌을 때, 각 명령어들의 이슈 사이의 거리 범위(distance bounds)를 구하는 방법을 설명하고, 이어서 이들 거리 범위를 이용하여 함께 이슈될 수 있는 명령어들을 판별하는 방법을 설명한다. 아울러, 하나의 IDG를 유지해야 하는 비용을 줄이기 위해 여러 개의 노드를 하나로 합쳐서 노드의 수를 줄일 수 있는 방법을 설명한다.

5.1 용어 정의

본 논문에서는 명령어간 이슈 거리의 분석 기법을 위해 다음의 용어들을 정의하여 사용한다.

* 본 논문에서는 레지스터와 실행 유닛을 통칭하여 자원(resource)이라고 부른다. 프로세서의 자원들의 집합을 나타내기 위한 집합 $R=\{r_1, r_2, \ldots, r_{N_R}\}$을 정의하였으며, 이때 N_R은 프로세서 내의 레지스터의 수와 실행 유닛의 수를 합한 것이다. (본 논문에서 사용하는 프로세서 모델의 경우 N_R은 64 개의 레지스터와 9 개의 실행 유닛의 수를 합한 73 개가 된다).
* 각 자원에는 최대 지연시간 z_i 값이 대응된다. 이 값은 명령어가 실행시 자원 r_i를 사용하는 최대 시간을 의미한다. z_i 값들 중에서 가장 큰 값을 $z_{\max}$라고 한다.
* 의존성이 결정되지 않은 노드 i에 대해 unresolved(i)는 노드 i가 사용하는 모든 자원들(레지스터, 실행 유닛) 중에서 다른 노드들과의 의존성이 밝혀지지 않은 자원들의 집합을 말한다.
* 각 노드 i에 대해 min_latency_node(i)는 노드 i 내의 첫번째 명령어의 이슈부터 마지막 명령어의 이슈까지의 시간을 구하는 함수이고, max_latency_node(i)는 노드 i 내의 첫 번째 명령어의 이슈부터 노드 i 내의 모든 명령어들의 실행이 끝날 때까지의 시간을 구하는 함수이다. 이 두 함수로부터 얻어지는 값들은 노드 i와 후속 노드 사이의 의존성을 나타내는 간선의 무게를 결정하기 위해 사용되는 것으로서, min_latency_node(i)는 노드 i와 후속 노드 사이의 의존성을 나타내는 간선의 최소 무게를 결정한다, max_latency_node(i)는 노드 i와 후속 노드 사이의 의존성을 나타내는 간선의 최대 무게를 결정한다. 또한 max_latency_unresolved(i)는 노드 i의 의존성이 결정되지 않은 자원들 중에서 지연시간이 가장 큰 값을 취한다. 이는 노드 i와 선행하는 노드 사이의 의존성을 나타내는 간선의 최대 무게를 결정하기 위해 쓰인다.

5.2 이슈간 거리 범위의 계산

명령어 이슈간 거리는 명령어들(노드들) 사이의 의존성 관계및 그 지연시간, 그리고 노드의 의존성이 결정되었는지의 여부에 따라 최소값과 최대값이 결정된다. 명령어 이슈간 거리를 결정하는 알고리즘을 설명하기 위한 간단한 예제를 그림 5에 보인다. 그림 5에 제시되어 있는 IDG에

는 모두 세 개의 노드가 있고, 노드 1과 2는 의존성이 결정되지 않은 노드들이다. 따라서, 노드 1과 노드 2가 언제 이슈 가능한 지는 선행하는 IDG의 노드들의 이슈 행태가 결정되지 않는 한 정확하게 알 수는 없다. 그림 5에서는 노드 1과 노드 2, 노드 2와 노드 3, 그리고 노드 1과 노드 3의 이슈간 거리를 계산한다. 먼저 노드 1과 노드 2의 이슈간 최소, 최대 거리를 계산하기 위해 노드 1과 노드 2 사이의 의존성 관계를 조사해 보면, 그림 5에 나타나 있듯이 지연시간이 0인 순서 의존성만 존재한다. 따라서, 노드 1과 노드 2의 이슈간 최소 거리는 0임을 알 수 있다. 다시

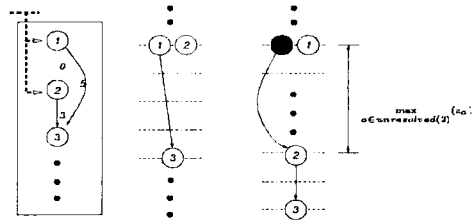


그림 5 명령어들의 이슈간 거리 범위 계산 (a) IDG의 예, (b) 노드 1과 2가 동시에 이슈되는 경우, (c) 노드 2가 최대 지연시간을 갖는 경우

말하면, 이 두 노드의 이슈간 최소 거리는 두 노드가 함께 이슈되는 상황에서 결정된다 (그림 5(b)).

한편, 두 노드 간 최대 거리는 노드 2가 노드 1로부터 가장 큰 지연시간을 두고 이슈되는 상황에서 결정된다. 이러한 상황은 그림 5(c)에 나타나 있다. 이 그림에서 검은색으로 나타나 있는 노드는 노드 1과 함께 이슈되었다고 가정하는 가상의 노드이다. 이 검은색 노드는 또한 노드 2에 대해 가능한 최대의 지연 시간을 두고 의존성을 가지고 있다고 가정한다. 이러한 가정은 노드 2로 하여금 노드 1로부터 최대의 지연시간을 두고 이슈되도록 하는 가상의 상황을 만들기 위한 것이다. 이러한 가상의 의존성을 가지도록 하려면 검은색 노드가 노드 2의 의존성이 결정되지 않은 자원들(unresolved(2)) 중에서 지연시간이 가장 큰 자원을 사용한다고 가정하고, 따라서 검은색 노드로부터 노드 2까지의 지연시간은 식 $\max_{a \in unresolved(2)}(z_a)$에 의해 계산된다. 여기서 a는 노드 2의 의존성이 결정되지 않은 자원 중 하나를 가리킨다.

다음으로 노드 2와 노드 3의 이슈간 거리를 계산한다. 노드 2와 노드 3의 이슈간 거리를 계산할 때 고려해야 하는 이미 밝혀진 지연시간은 노드 1과 노드 2, 노드 2와 노드 3, 노드 1과 노드 3 사이의 간선의 무게인 $l_{1,2}, l_{2,3}, l_{1,3}$ 이고 아울러 전 단계에서 계산한 노드 1과 노드 2의 이슈간 최소 최대 거리를 사용한다. 우선 노드 2와 노드 3의 이슈간 최소 거리로 생각할 수 있는 값은 노드 2와 노드 3 사이의 지연시간인 3이다. 그러나, 다른 요인들에 의해 이 두 노드의 이슈간 최소 거리는 이보다 큰 값을 가질 수 있다. 예를 들어 노드 1과 노드 2의 이슈간 최대 거리가 2보다 작을 경우, 노드 1과 노드 3 사이에는 의존성에 의해 지연시간 5가 지켜져야 하기 때문에, 노드 2와 노드 3 사이의 거리는 최소한 3보다 큰 값을 가지게 된다. 따라서, 노드 2와 노드 3의 이슈간 최소 거리는 (1) $l_{2,3}$ 값과 (2) $l_{1,3}$에서 노드 1과 노드 2의 이슈간 최대 거리를 뺀 값 중에서 큰 값을 취한다. 마찬가지로 노드 2와 노드 3의 이슈간 최대 거리는 (1) $l_{2,3}$ 값과 (2) $l_{1,3}$에서 노드 1과 노드 2의 이슈간 최소 거리를 뺀 값 중에서 큰 값을 취한다. 이상의 설명에서 나타나 있듯이 노드 2와 노드 3 사이의 거리를 분석하는데 노드 1과 노드 2의 이슈간 최소 및 최대 거리 값이 사용된다. 이 값들은 사전에 계산되어져야 한다. 마지막으로, 노드 1과 노드 3의 이슈간 최소, 최대 거리를 계산한다. 이 두 노드간의 거리는 최소한 $l_{1,3}$ (= 5) 값 이상이다. 그러나, 두 노드의 이슈간 최소 거리는 이 값보다 커질 수 있다. 예를 들어 노드 1과 노드 2 사이의 거리가 항상 2보다 크다고 하면, 노드 1과 노드 3 사이의 거리 또한 5보다 큰 값을 가지게 된다. 따라서, 노드 1과 노드 3의 이슈간 최소 거리는 (1) $l_{1,3}$ (= 5) 값과 (2) 노드 1과 노드 2의 이슈간 최소 거리에 $l_{2,3}$ 값을 더한 값 중에서 더 큰 값을 취한다. 마찬가지로, 노드 1과 노드 3의 이슈간 최대 거리는 (1) $l_{1,3}$ 값과 (2) 노드 1과 노드 2의 이슈간 최대 거리에 $l_{2,3}$ 값을 더한 값 중에서 더 큰 값을 취한다.

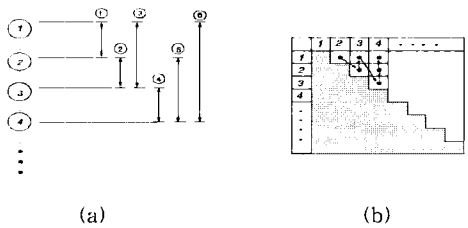


그림 6 이슈간 거리를 계산하는 순서. (a) 이슈간 거리를 계산하는 순서, (b) 이슈간 거리 계산 순서의 테이블 형태 표현

이상의 설명에서 나타나 있듯이 노드들의 이슈간 거리

를 계산할 때에는 미리 계산되어져 있어야 하는 다른 노드들의 이슈간 거리가 있다. 이러한 조건을 만족시키기 위해서 본 연구에서는 그림 6에 나타나 있는 순서대로 노드들의 이슈간 거리를 구하는 방법을 사용하였다. 그림 6(b)는 노드들의 이슈간 거리를 구하는 순서를 테이블의 형태로 보이고 있다. 이 테이블에서 행과 열의 번호는 각각 노드들의 번호를 가리킨다.

이상의 간단한 예를 통한 설명을 일반화하여 설명하면 다음과 같다. 먼저 $D_{i,j}^{\min}$와 $D_{i,j}^{\max}$를 각각 노드 i, j의 이슈간 최소, 최대 거리라고 하자. 그림 6에 나타나 있는 순서를 따르자면, 노드들의 이슈간 거리를 계산하는 순서는 [$D_{1,2}^{\min}$, $D_{1,2}^{\max}$], [$D_{2,3}^{\min}$, $D_{2,3}^{\max}$], [$D_{1,3}^{\min}$, $D_{1,3}^{\max}$], [$D_{3,4}^{\min}$, $D_{3,4}^{\max}$], [$D_{2,4}^{\min}$, $D_{2,4}^{\max}$], [$D_{1,4}^{\min}$, $D_{1,4}^{\max}$], ...이어야 한다. 어떤 노드 i와 노드 j의 이슈간 최소, 최대 거리([$D_{i,j}^{\min}$, $D_{i,j}^{\max}$])를 구할 때에는 노드 j와 의존성 관계에 있는 모든 노드들을 고려해야 한다. 계산 과정에 대한 이해를 돕기 위해서 노드 j와 의존성 관계에 있는 노드들을 그림 7과같이 세 부류로 나눈다. 첫째는 노드 i보다 먼저 나타나는 노드들이고 (즉, 그림 7(b)의 노드들), 둘째는 노드 i (그림 7(c)), 그리고 노드 i보다 후에 나타나는 노드들 (그림 7(d)의 노드들)이다.

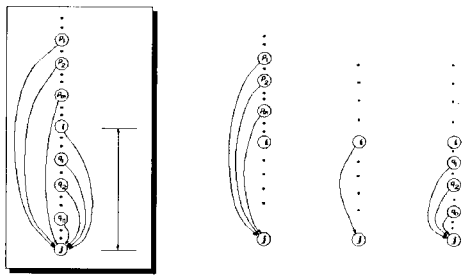


그림 7 노드 j와 의존성 관계에 있는 노드들의 세 부류 (a) IDG 예, (b) 노드 i를 선행하는 노드들, (c) 노드 i, (d) 노드 i와 노드 j 사이에 있는 노드들

먼저 $D_{i,j}^{\min}$을 구하기 위해서 각 부류에서의 후보 값을 계산하고 이들 값 중에서 가장 큰 값을 $D_{i,j}^{\min}$로 취한다. 각 부류에서 후보 값을 계산하는 방법은 다음과 같다. 첫째 부류에서는 각 $l_{p_x,j}$와 $D_{p_x,i}^{\max}$의 차이 값을 구한다. (이때, x는 1에서 m사이의 값이다.) $l_{p_x,j}$ 값은 IDG에 나타나 있고 $D_{p_x,i}^{\max}$ 값은 그림 6에 나타나 있는 계산 순서에 의해 이미 계산되어져 있는 값이다. 이 차이 값들 중에서 가장 큰 값을 첫번째 부류의 후보 값으로 취한다.

두번째 부류에서는 노드 i에서 노드 j의 지연시간인 $l_{i,j}$를 후보 값으로 취한다. 세번째 부류에서는 $D_{i,q_y}^{\min}$ 값과 $l_{q_y,j}$ 값의 합을 구한다. (이때, y는 1에서 n까지이다.) 이 합한 값들 중에서 가장 큰 값을 후보 값으로 취한다. 한편, 노드 i와 노드 j의 이슈간 최대 거리인 $D_{i,j}^{\max}$ 값을 구할 때에는 각 노드의 지연시간, 즉, 간선의 무게 값을 특별하게 처리할 필요가 있다. 그 이유는 이 값은 노드 j가 가능한 최대의 지연시간이 지난 뒤에 이슈가 되는 상황을 가정해야 하기 때문이다. 이러한 상황을 만들기 위해서 노드 1에서부터 의존성이 결정되지 않은 다른 노드(unresolved nodes) 사이에 가상의 의존성이 있다고 가정하고, 이 의존성은 노드 1과 각 의존성이 결정되지 않은 노드 사이에 가능한 최대의 지연시간을 일으킨다고 가정한다. 이러한 가상의 상황을 위해 노드 1에서부터 각 의존성이 결정되지 않은 노드 i 사이에 가상의 간선을 도입하고 (이를 max edge라고 부른다.) 이 간선의 무게는 max_latency_unresolved(i)라고 가정한다. 이 간선에 의해 의존성이 결정되지 않은 노드들이 노드 1로부터 최대의 지연시간 뒤에 이슈되는 상황을 만들어 낸다.

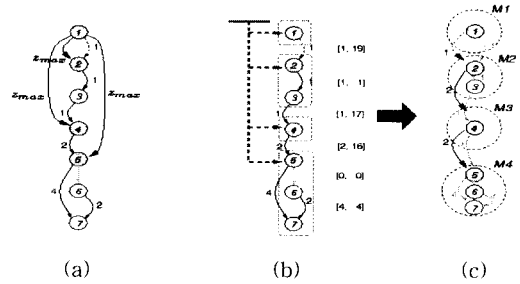


그림 8 $D_{i,j}^{\max}$ 계산을 위해 수정된 IDG 및 합병 연산 (a) 가상의 간선(max edges)이 도입된 IDG, (b) 합병 가능한 노드들, (c) 합병 연산 후의 IDG

그림 8은 이러한 가상의 간선을 도입한 IDG의 예를 보인다. 그림 8(a)는 그림 4의 IDG에 가상의 간선을 추가한 새로운 IDG를 보인다. 새로운 간선의 무게가 모두 $z_{\max}$인 이유는 이 노드들의 의존성이 결정되지 않은 자원들이 모두 레지스터들이었기 때문이다. 위의 설명대로 가상의 간선을 도입한 IDG에 대해서 $D_{i,j}^{\max}$를 $D_{i,j}^{\min}$을 구할 때와 비슷한 방법으로 구한다. 먼저 각 부류에서 후보 값을 계산하고, 이 후보 값들 중에서 가장 큰 값을 $D_{i,j}^{\max}$로 취한

다. 첫번째 부류에 대해서는 각 $l_{p..j}$ 값과 $D^{min}_{p..i}$ 값 사이의 차이들 중에서 가장 큰 값을 후보값으로 취한다. (이때 x의 값은 1에서 m까지이다.)

두번째 부류에서는 $l_{i,j}$를 후보값으로 취한다. 마지막으로 세번째 부류에서는 각 $D^{max}_{i,q}$ 값과 $l_{q..j}$ 값의 합을 구하여 이 합한 값들 중에서 가장 큰 값을 후보 값으로 취한다. (이때 y의 값은 1에서 n까지이다.) 지금까지 설명한 노드 i와 노드 j의 이슈간 최소, 최대 거리를 구하는 식은 다음과 같이 정리할 수 있다.

$$D^{min}_{i,j} = \max(\max_{1\le x\le m}(l_{p..j} - D^{max}_{p..i}), l_{i,j}, \max_{1<y<n}(D^{min}_{i,q} + l_{q..j}))$$

$$D^{max}_{i,j} = \max(\max_{1\le x\le m}(l_{p..j} - D^{min}_{p..i}), l_{i,j}, \max_{1\le y\le n}(D^{max}_{i,q} + l_{q..j}))$$

이상 설명한 노드들의 이슈간 최소, 최대 거리는 인접한 프로그램 구문의 IDG가 알려지고 새로운 노드들 간의 의존성 관계가 밝혀지면 다시 위에 설명한 계산식을 통해 이슈간 최소, 최대 거리를 갱신하게 되고, 이 갱신된 거리 범위는 보다 정확한 거리 범위가 된다.

5.3 다중 이슈 행태 분석 및 노드 합병

앞 절에서 설명한 식에 의해 노드들 사이의 이슈간 최소, 최대 거리가 결정되면 이를 토대로 명령어들의 이슈 행태를 분석할 수 있다. 직관적으로 알 수 있듯이, 어떤 노드들 간의 이슈간 최소, 최대 거리가 모두 0일 경우 이 두 노드는 함께 이슈되는 것을 보장하게 된다. 그러나, 무조건 [0, 0]의 이슈간 거리 범위를 가지고 있는 노드들이 함께 이슈된다고 말할 수 없다. 그 이유는 프로세서들마다 함께 이슈될 수 있는 명령어의 수가 제한되어 있기 때문이다. 따라서, [0, 0]의 이슈간 거리 범위가 여러 개 이어서 나타나는 경우 프로세서의 다중 이슈 능력을 고려하여 특별하게 처리해야 한다. 예를 들어 어떤 프로세서가 최대 k개의 명령어들을 함께 이슈할 수 있다고 하자. 또한, 연달아 나타나는 노드들 p, p+1, . . ., p+q이 모두 [0, 0]의 이슈간 거리 범위를 가지고 있다고 하자. 이때, 프로세서의 다중 이슈 능력이 이 모든 노드들을 함께 이슈할 수 있는 지 조사하기 위해 먼저 q+1이 k보다 작은 지 알아보아야 한다. 만약, q+1이 k보다 작은 경우에는 이 모든 노드들이 함께 이슈될 수 있다. 그리고, 이들 노드들에 대해 더 이상의 특별한 처리를 할 필요가 없게 된다. 그러나, q+1이 k보다 큰 경우에는 모든 노드들이 함께 이슈될 수는 없고 처음 k 개의 노드들만 함께 이슈되고, 다음 노드들은 다음 이슈 가능한 사이클에 이슈된다. 이들 노드들이 모두 함께 이슈될 수 있다고 분석되어지는 것을 막기 위해서 IDG내에 노드 p+k-1과 p+k 사이에 인위적으로 간선을 추가한다. 이 간선의 무게를 1로 정함으로써 처음 k 개의 노드만 함께 이슈되도록 하고 다음 노드들은 적어도 한 사이클이 지난 후에 이슈되도록 만든다.

한편, 하나의 IDG에 속해있는 노드들의 수는 분석이 진행되어 인접한 프로그램 구문의 IDG와 합쳐짐에 따라 점차 크게 증가하게 된다. 따라서, IDG 내에 유지되는 노드의 수를 줄이는 방법이 필요하고, 이 방법은 결국 ETS를 적용하기 위해 구성하는 PA 정보 내에 포함되어야 하는정보의 크기를 줄일 수 있게 된다. IDG 내의 노드의 수를 줄이기 위한 방법으로 본 연구에서는 노드 합병 알고리즘을 개발하였다. 이 노드 합병은 노드들 사이의 거리 범위가 더 이상 변화할 가능성이 없는 경우 해당 노드들을 합쳐서 하나의 노드로 유지하는 것이다. 다시 말하면, 앞에서 설명한 식에 의해 구한 노드간의 거리 범위의 최소 거리와 최대 거리 값이 같다면, 해당 두 노드 사이의 거리는 어떠한 경우에도 더 이상 변화할 가능성이 없다는 것이 보장된다. 따라서, 본 연구에서는 인접한 두 노드 사이의 이슈간 최소, 최대 거리의 값이 같은 경우 두 인접한 노드를 합하여 하나의 노드로 유지한다. 이 때 합쳐진 노드와 다른 노드들 사이의 의존성 관계 및 지연시간 정보는 적절하게 갱신된다. 그림 8(b)와 그림 8(c)에 노드 합병 과정을 보인다. 예를 들면, 노드 1과 노드 2는 이슈간 거리 범위가 [1, 1]이기 때문에 합병이 가능하다. 또한, 노드 5와 노드 6, 그리고 노드 6과 노드 7도 각각의 이슈간 거리 범위가 [0, 0]과 [4, 4]이기 때문에 합병이 가능하다.

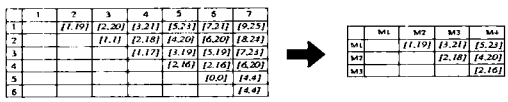


그림 9 이슈간 거리 범위의 테이블 표현 (a) 원래 IDG의 이슈간 거리 범위, (b) 합병 연산 후 이슈간 거리 범위

그림 8(c)는 이렇게 합병된 노드로 구성된 IDG를 보인다. 본 연구에서는 이러한 노드의 합병을 이미 계산되어진 이슈간 최소, 최대 거리 정보를 담고 있는 그림 9의 테이블을 사용하여 수행할 수 있는 알고리즘을 개발하여 사용하였다. 그림 9(a)는 노드 합병이 수행되기 전의 노드들의 이슈간 거리 정보를 담고 있는 테이블이고, 그림 9(b)는 노드 합병이 수행된 이후의 노드들의 이슈간 거리 정보를 담고 있다. 이러한 노드들의 이슈간 거리 정보를 담고 있는 테이블을 본 연구에서는 bounds table이라고 부른다.

6. ETS의 확장

본 연구에서는 이상 설명한 IDG의 노드들의 이슈간 최소, 최대 거리의 계산 및 이슈 행태의 분석, 그리고 노드들의 합병 방법을 사용하여 결정된 특정 프로그램 구문에 대한 실행시간 정보를 ETS를 사용한 WCET 분석에서 활용하기 위해서 각 프로그램 구문에 따라 구성하는 PA를 새로 정의하고, 이에 따라 접속, 절지 연산을 재정의하였다.

6.1 PA 자료 구조

그림 10에 본 연구에서 제시한 다중 이슈 프로세서 모델에 대한 PA 자료구조를 보인다. 이 PA 자료구조에는 다

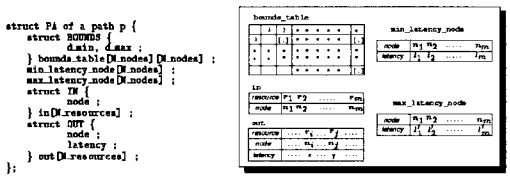


그림 10 PA 자료구조

섯 개의 테이블이 포함된다. PA 자료구조에 포함되는 정보를 나열해 보면, 우선 앞 절에서 설명한 방법에 의해 계산된 노드들의 이슈간 최소, 최대 거리 정보를 담고 있는 bounds_table이 포함된다. 그리고, 해당 IDG 내에서 사용하는 자원에 대한 정보를 인접하여 선행하는 프로그램 구문으로부터 영향받을 수 있는 정보와 인접하여 뒤에 따르는 프로그램 구문에 영향을 줄 수 있는 정보로 나누어 각각 in 정보와 out 정보에 포함시킨다. in 정보는 각 자원에 대해서 그 자원을 IDG 내에서 처음으로 사용하는 노드 번호를 유지하고 있는 테이블로서, 선행하는 프로그램 구문의 노드들의 자원 활용 행태에 따라 이들 노드들이 이슈되는 시간이 결정된다. out 정보는 각 자원을 마지막으로 사용하는 노드의 번호와, 만약 같은 자원을 다른 뒤에 따르는 노드가 사용하고자 할 때 유지되어야 하는 지연시간 정보를 포함한다. 이는 인접하여 뒤에 따르는 프로그램 구문의 노드들이 같은 자원을 사용하고자 할 때 유지되어야 하는 지연시간을 유지함으로써, 접속 연산시 새로 확인되는 의존성을 처리하기 위함이다. 나머지 두 테이블은 min_latency_node와 max_latency_node로서 모두 여러 개의 노드의 병합에 의해 생성된 노드 내의 명령어들 사이의 거리를 유지하기 위해서 PA에 포함시킨다. 이 테이블의 값들은 5.1 장에 정의되어 있는 min_latency_node(i)와 max_latency_node(i) 값들을 사용한다. 이렇게 결정된 PA 자료구조의 테이블들을 활용하면 하나의 IDG에 대한 PA가 주어졌을 때 해당 프로그램 구문의 최악 실행시간을 구할 수 있게 된다. 구하는 식은 다음과 같다.

$$T_{worst} = \max_{0 \le i \le m}(D_{1,i}^{\max} + \max\_latency\_node(i))$$

여기서 m은 해당 IDG 내에 속해있는 모든 노드들의 수이다. 이 식이 나타내는 것은, 하나의 IDG에 대해서 첫번째 노드를 제외한 모든 노드가 최악의 지연시간을 두고 이슈되는 상황을 가정하고, 첫번째 노드의 이슈에서부터 IDG 내의 모든 노드의 실행이 끝나는 순간까지를 계산하는 것이다.

6.2 접속 연산

ETS를 본 논문에서 가정하는 다중 이슈 프로세서 모델에 적용하기 위해 두번째로 재정의해야 하는 것은 위에서 정의한 PA 자료구조를 고려한 접속 연산이다. 접속 연산은 두 개의 인접한 PA 정보를 연결하여 새로운 PA 정보를 만들어내는 연산으로서 두 개의 프로그램 구문이 연달아 실행되는 상황을 모델링한다. PA 정보에는 bounds_table, in, out, min_latency_node, 그리고 max_latency_node 등의 다섯개의 테이블이 포함되므로 이 테이블 각각에 대해 새로운 정보를 구성하는 연산을 접속 연산 내에 포함시켜야 한다.

접속 연산은 크게 세 부분으로 나누어져 있다. 우선 두 개의 bounds_table 내용(w_1, w_2에 대해 각각 $\text{bounds\_table}_{w_1}$, $\text{bounds\_table}_{w_2}$)을 조사하여 새로운 PA w_3의 $\text{bounds_table}_{w_3}$을 채운다. 이 때 기존 bounds_table들의 내용을 그대로 옮기는 부분이 있고, 기존 PA들의 in 정보와 out 정보를 조사하여 기존 PA내의 노드들 사이에 새로 발견되는 의존성이 있는 지 파악한 후 이를 반영하여 채워지는 부분이 있다. 새로이 발견되는 의존성이 있는 경우는 w_1의 out과 w_2의 in내에 공유되는 자원이 있는 경우이다. 이 때에는 이 자원을 사용하는 각각의 PA 내의 노드들 사이에 새로운 간선과 함께 이 자원의 지연시간(latency)이 간선의 무게로 추가된다. 이렇게 새로 발견되는 간선을 고려하여 노드들의 이슈간 거리 범위를 구하기 위한 알고리즘을 수행하면 새로 발견되는 의존성을 고려하여 bounds_table을 구성할 수 있다.

접속 연산의 두번째 단계는 w_3의 min_latency_node와 max_latency_node의 내용을 채우는 것이다. 이 테이블들은 이전 단계에서 합병 연산이 일어나지 않은 경우 다음과 같이 채워진다.

(1) w_1에 있던 노드들에 대해서,

$w_3.\text{min\_latency\_node}[i](\text{max\_latency\_node}[i]) =$
$w_1.\text{min\_latency\_node}[i](w_1.\text{max\_latency\_node}[i])$,
여기서 $1 \le i \le N_{w_1}$, 그리고

(2) w_2에 있던 노드들에 대해서,

w_3. min_latency_node[$N_{w_3}+1$] (max_latency_node[$N_{w_3}+1$]) =
w_2. min_latency_node[i] (w_2. max_latency_node[i])

, 이때 $1 \le i \le N_{w_2}$ 이다.

만약 이전 단계에서 합병연산이 일어나 몇 개의 노드가 합쳐졌다면, 이 두 테이블에 대해서는 노드들의 이슈간 거리 범위를 구하는 알고리즘을 다시 적용하여 이슈간 거리 범위를 구한다.

접속 연산의 마지막 단계는 w_3의 in와 out의 내용을 채우는 것이다. 먼저, in 테이블의 경우, w_1에 내용이 있는 자원들에 대해서는 w_1의 내용으로 채우고 그렇지 않은 자원들에 대해서는 w_2의 내용으로 채운다. out 테이블의 경우, w_2에 내용이 있는 자원들에 대해서는 w_2의 내용으로 채우고 그렇지 않은 자원들에 대해서는 w_1의 내용으로 채운다.

6.3 절지 연산

절지 연산은 하나의 WCTA 내에 포함되어 있는 여러 개의 PA 중에서 어떤 경우에도 최악 실행경로에 포함될 수 없는 것들을 찾아내어 WCTA에서 삭제함으로써 이후 분석 과정에 이들 PA가 연산에 이용되는 것을 막아서 분석의 복잡도를 줄이는 역할을 한다. 절지 연산을 수행하기 위해서는 각 PA가 나타내는 프로그램 구문의 최악 실행시간을 두가지 관점에서 구해야 한다. 하나는 최선의 상황에서의 최악 실행시간이고 다른 하나는 최악의 상황에서의 최악 실행시간이다. 이를 각각 $T_{best_worst}(w)$와 $T_{worst_worst}(w)$라고 하고 이들을 구하는 식은 아래와 같다.

$$T_{best\_worst}(w) = D^{\min}_{1,m} + \min\_latency\_node(m),$$

여기서 m은 PA w 내의 IDG 노드의 수이다. 그리고,

$$T_{worst\_worst}(w) = \max_{1 \le i \le m}(D^{\max}_{1,m} + \max\_latency\_node(i))$$

이다. 어떤 PA w의 최악의 상황에서의 실행시간이 다른 어떤 PA w'의 최선의 상황에서의 실행시간보다 작은 경우 PA w는 절대로 최악 실행경로에 포함될 수 없기 때문에 절지 연산에 의해 삭제된다.

6.4 프로그램의 WCET 계산

한편, 하나의 프로그램에 대한 IDG가 결정된 상태에서 이 프로그램의 최악 실행시간(WCET)을 구할 때에는 다음 식을 사용한다. 이때의 가정은 첫번째 노드를 제외한 다른 모든 노드들이 최악의 지연시간을 두고 이슈된다는 것이다.

$$WCET_{program} = \max_{i \in N}(D^{\max}_{1,i} + \max\_latency\_node(i)),$$

이때, N는 프로그램의 최종적인 IDG 내에 포함된 노드의 집합이다.

7. 실험 결과

본 논문에서는 이상 설명한 순서적, 다중 이슈(in-order, multiple issue) 프로세서를 위한 최악 실행시간 분석 기법을 토대로 하여 최악 실행시간 분석기를 구현하여 이를 이용한 분석 결과를 시뮬레이터를 이용한 실제 실행 결과와 비교한다. 시간 분석 환경은 그림 11과 같다. 구현한 최악 실행시간 분석기는 하나의 응용 프로그램에 대해 구문 트리 정보, 호출 그래프 정보, 어셈블리 코드,

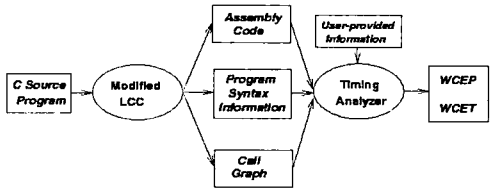


그림 11 시간 분석 환경

그리고 순환문의 순환 횟수 정보 등의 네 개의 정보를 입력으로 받는다. 이들 정보 중 구문 트리 정보, 호출 그래프 정보, 그리고 어셈블리 코드는 본 실험에서 사용하기 위해 수정된 lcc 컴파일러로부터 생성된다. 순환문의 순환 횟수 정보는 사용자로부터 입력된다.

시간 분석기를 통한 분석 결과와 비교하기 위해 사용된 시뮬레이터는 VMW (Visuliation-based Microarchitecture Workbench) [3]라고 하는 시뮬레이터 생성 도구를 통해 생성하였다. VMW는 사용자가 프로세서의 여러 가지 사양을 VMW가 이해하는 기술 언어를 사용하여 기술하면, 해당 프로세서에 대한 시뮬레이터를 생성해 준다. VMW의 기술 언어를 사용하여 본 논문에서 사용하는 프로세서 모델을 기술하였으며 해당 프로세서 모델의 시뮬레이터를 생성하였다.

표 2 벤치마크 프로그램

benchmarks	description
Arrsum	10개 배열을 합산한다.
Fib	피보나치 수열을 계산한다.
MM	두 개의 5 X 5 행열을 곱셈한다.
BS	15 개 배열에 대한 이진 탐색을 수행한다.
InLP	어셈블리 코드의 명령어들을 재배열하여 보다 큰 ILP를 보이도록 작성된 프로그램이다.

표 2는 본 연구에서 구현한 최악 실행시간 분석기로부터의 분석 결과를 검증하기 위하여 사용한 벤치마크 프로그

램들을 보인다. 이들 프로그램들은 수정된 lcc를 통해 MIPS R3000/R3010 프로세서의 명령어로 구성된 어셈블리 코드로 변환되어 최악 실행시간 분석기에 입력된다. 본 논문에서 제안한 기법이 한번에 여러 개의 명령어를 이슈할 수 있는 다중 이슈 프로세서를 대상으로 하는 데 반해, 일반 RISC 프로세서인 MIPS R3000/R3010 프로세서의 어셈블리 코드를 출력하는 컴파일러로부터 생성된 코드들은 평균 1 내지 2 개의 명령어만을 동시에 이슈할 수 있는 낮은 ILP를 보인다. 따라서, 표 2에 나타난 바와 같이 {\em InLP}라고 하는, 이미 컴파일러를 거쳐 출력된 어셈블리 코드의 명령어 순서를 인위적으로 재배열하여 상대적으로 높은 ILP를 보이게 함으로써 제안하는 기법의 효용성을 검증하는 데 활용하였다.

표 3 실험결과

	single issue		double issue		quadruple issue	
	Simulation	Prediction	Simulation	Prediction	Simulation	Prediction
Arrsum	108	108	92	92	92	92
Fib	227	227	190	190	189	189
MM	4142	4142	3553	3553	3552	3552
BS	101	106	81	84	80	83
InLP	3331	3331	2498	2498	2290	2290

이상 설명한 벤치마크 프로그램들의 최악 실행시간 분석 결과와 VMW를 사용하여 생성된 시뮬레이터에서 출력된 결과를 비교한 내용이 표 3에 나타나 있다. Arrsum, Fib, MM, 그리고 InLP 프로그램들에 대해서는 시뮬레이션 결과와 최악 실행시간 분석기로부터의 분석 결과가 동일하였다. 이는 이들 프로그램에는 실행 경로가 오직 하나씩만 존재하기 때문이다. 그러나, BS 프로그램에 대해서는 두 결과가 차이를 보였다. 이는 이 프로그램에는 여러 개의 실행 결과가 존재하고, 이 중에서 실제 실행시에는 불가능한 실행 경로가 분석시 최악 실행경로로 분석되었기 때문이다. 이러한 문제를 불가능한 실행경로(infeasible path) 문제라고 하며 정적으로 최악 실행시간을 분석하는 데 있어서 분석 결과와 실제 실행시간 사이의 차이를 유발하는 중요한 요인중 하나이다. 본 논문의 목적은 다중 이슈가 가능한 수퍼스칼라 프로세서 모델을 세우고 이 모델에 대해 실제 실행시 나타나는 명령어들의 이슈 행태에 근접하도록 명령어들의 이슈 행태를 분석하여 최악 실행시간을 도출하는 데 있기 때문에 불가능한 실행경로(infeasible path) 문제는 다루지 않았다.

8. 결 론

본 논문에서는 ETS를 기반으로 하여 다중 이슈 프로세서의 최악 실행시간을 분석할 수 있는 방법을 제시하였다. 다중 이슈 프로세서의 명령어들의 이슈 행태를 분석하기 위해서 명령어들 사이의 의존성 관계를 표현하는 IDG를 PA 정보로 사용하며, 이 IDG로부터 명령어들 사이의 이슈간 거리 범위를 구한다. 명령어들 사이의 이슈간 거리 범위는 프로그램의 계층적인 분석을 통해 점차로 더 정확한 정보로 갱신되어진다. 명령어들 사이의 이슈간 거리 범위를 이용하여 동시에 이슈될 수 있는 명령어들을 파악할 수 있게 된다. 본 논문에서는 ETS의 접속 연산 및 절지 연산을 IDG를 반영하여 수정하였다.

제안한 기법을 토대로 하여 순서적(in-order), 다중 이슈 프로세서 모델에 대한 WCET를 분석할 수 있는 시간 분석기를 작성하였다. 시간 분석기에서 출력된 분석 결과와 시뮬레이터로부터 얻은 실제 실행 결과를 비교한 실험 결과는 본 논문에서 제안하는 기법이 프로세서의 순서적, 다중 이슈 동작을 정확하게 분석할 수 있음을 보였다.
본 논문에서 제안하는 기법은 프로세서의 다중 이슈 동작에만 초점을 맞추고 있으나 실제 다중 이슈 프로세서에 대한 분석을 하기 위해서는 본 논문에서 고려하지 않은 다른 프로세서 구조에 대한 고려가 있어야 한다. 예를 들면, 수퍼스칼라 프로세서에서 많이 채용하고 있는 동적 명령어 스케줄링, 동적 분기 예측, 캐쉬 등에 대한 고려가 필요하다. 또한, 프로그램의 불가능한 실행경로를 파악하고 이를 WCET 분석시 고려 대상에서 제외할 수 있는 방법의 개발도 분석의 정확도를 높이기 위해서 필요하다.

참 고 문 헌

[1] R. Arnold, F. Mueller, D. Whalley, and M. Harmon. Bounding Worst Case Instruction Cache Performance. In Proceedings of the 15th Real-Time Systems Symposium, pages 172--181, Dec. 1994.

[2] J.-Y. Choi, I. Lee, and I. Kang. Timing Analysis of Superscalar Processor Programs Using ACSR. In Proceedings of the 11th Workshop on Real-Time Operating Systems and Software, pages 63--67, May 1994.

[3] T. A. Diep and J. P. Shen. VMW: A Visualization-Based Microarchitecture Workbench. IEEE Computer, 28(12):57--64, Dec. 1995.

[4] C. Ferdinand, F. Martin, and R. Wilhelm. Applying Compiler Techniques to Cache Behavior Prediction. In Proceedings of 1997 ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems, pages 37--46, June 1997.

[5] E. Harcourt, J. Mauney, and T. Cook. High-Level

Timing Specification of Instruction-Level Parallel Processors. Technical Report TR?3?8, Dept. of Computer Science, North Carlina State University, August 1993.

[6] M. Harmon, T. P. Baker, and D. B. Whalley. A Retargetable Technique for Predicting Execution Time. In Proceedings of the 13th Real-Time Systems Symposium, pages 68--77, Dec. 1992.

[7] C. Healy, R. Arnold, F. Mueller, D. Whalley, and M. Harmon. Bounding Pipeline and Instruction Cache Performance. IEEE Transactions on Computers, 48(1):53--70, Jan. 1999.

[8] 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, pages 288--297, December 1995.

[9] M. Johnson. Superscalar Microprocessor Design. Prentice Hall, Englewood Cliffs, NJ, 1991.

[10] 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, pages 298--307, Dec. 1995.

[11] 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, 21(7):593--604, July 1995.

[12] A. Mok. Evaluating Tight Execution Time Bounds of Programs by Annotations. In Proceedings of the 10th IEEE Workshop on Real-Time Operating Systems and Software, pages 74--80, 1989.

[13] K. Narasimhan and K. Nilsen. Portable Execution Time Analysis for RISC Processors. In Proceedings of the Workshop on Architecture for Real-Time Systems, April 1994.

[14] G. Ottosson and M. Sjödin. Worst-Case Execution Time Analysis for Modern Hardwre Architectures. In Proceedings of 1997 ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems, pages 47--55, June 1997.

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

[16] A. C. Shaw. Reasoning About Time in Higher-Level Language Software. IEEE Transactions on Software Engineering, 15(7):875--889, July 1989.

[17] H. Theiling and C. Ferdinand. Combining Abstract Interpretation and ILP for Microarchi tecture Modelling and Program Path Analysis. In Proceedings of the 19th IEEE Real-Time Systems Symposium, pages 144--153, Dec. 1998.

[18] N. Zhang, A. Burns, and M. Nicholson. Pipelined Processors and Worst-Case Execution Times. Real-Time Systems, 5(4):319--343, Oct. 1993.

임 성 수
1993년 서울대학교 컴퓨터공학과 학사. 1995년 서울대학교 컴퓨터공학과 석사. 현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 컴퓨터구조, 실시간시스템

한 정 희
1995년 서울대학교 컴퓨터공학과 학사. 1997년 서울대학교 컴퓨터공학과 석사. 현재 미시간 주립대학교 전기컴퓨터공학과 박사과정. 관심분야는 컴퓨터구조, 실시간시스템

김 지 홍
1986년 서울대학교 계산통계학과 학사. 1988년 University of Washington 컴퓨터과학과 석사. 1995년 University of Washington 컴퓨터과학 및 공학과 박사 1995년 ~ 1997년 미국 Texas Instruments사 선임연구원. 1997년 ~ 현재 서울대학교 전기 컴퓨터공학부 조교수. 관심분야는 컴퓨터구조, 내장형 시스템, 저전력 시스템, 멀티미디어 시스템

민 상 렬
정보과학회논문지 : 시스템 및 이론
제 27 권 제 3 호 참조