ACSR을 이용한 비순차 슈퍼스칼라 프로세서의 시간 분석*

이기흔, 최진영
고려대학교 컴퓨터학과

Timing Analysis of Out-of-order Superscalar Processor Programs Using ACSR

Ki-Huen Lee, Jin-Young Choi
Department of Computer Science and Engineering, Korea University

요 약

본 논문은 프로세스 알제브라의 하나인 ACSR을 이용하여 파이프라인 비순차 슈퍼스칼라 프로세서의 타이밍 특성과 자원제한을 묘사하기 위한 정형기법을 제시한다. ACSR의 두드러진 특징은 시간, 자원, 우선 순위의 개념이 알제브라에서 직접적으로 제공되어 진다는 것이다. 여기서의 접근 방식은 슈퍼스칼라 프로세서의 레지스터를 ACSR 자원으로, 명령어를 ACSR 프로세스로서 모델링하는 것이다. 결과적으로 얻어지는 ACSR 식에서 각각의 클럭 주기에서 어떻게 명령어가 실행되고 레지스터들이 이용되는지 확인할 수 있으며 이 모델링을 이용해 비순차 슈퍼스칼라 프로세서 구조를 검증하거나 분석하는 것이 가능하다

1. 서론

슈퍼스칼라 프로세서의 병렬성[7, 5, 8]은 단순히 하드웨어 기능 유니트들의 구성으로부터만 얻어지는 것이 아니라 명령어들 사이의 데이터 의존성과 깊은 관련을 가지고 있으며 이것이 타이밍 특성을 분석하데 어려움이 되고 있다. 여기서는 프로세스 알제브라에 기반한 정형 방법을 통해 비순차 슈퍼스칼라 프로세서에서 일련의 명령어로 된 프로그램을 실행하는데 있어 타이밍 특성과 자원 제한을 분석하기 위한 방법을 제시함으로써 ISA(Instruction Set Architecture) 레벨의[3] 기술(記述)을 늘리고자 한다. 그리고 그 프로세스 알제브라로는 ACSR(Algebra for Communication of Shared Resources)을 사용한다. 왜냐하면, 첫째, ACSR에는 자체적으로 시간, 자원, 그리고 우선순위에 대한 개념이 들어 있기 때문이다 ACSR의 시간 개념은 타이밍 분석을 하는데 자연스럽게 적용되고, 우선순위는 몇 가지 가능한 선택이 있을 때 가장 자원의 활용이 높은 선택을 가능케 함[6]으로써 슈퍼스칼라 프로세서의 근본 개념과 일치한다. 둘째로 ACSR은 프로세스 알제브라로서 식으로 표현된 명령어 명세를 검증할 수 있는 증명 기법을 갖추고 있으며 이를 위한 VERSA(Verification & Specification Tool for ACSR)[2] 라는 소프트웨어 도구(tool) 또한 개발되어 있다

이 논문은 앞서 발표되었던 [9],[10]에 기초하여, 비순차 게시에 대한 경우를 추가한 것이다. 앞의 논문에 소개되었던 일련의 작업들은 Harcourt 등의 선구적인 업적으로부터 영감을 얻은 것으로, 그들은 일찍이 명령어 명세를 위해 SCCS라는 프로세스 알제브라를 이용했었다[4] SCCS에는 자원이란 개념이 없었으므로 각 자원을 바이너리 세마포어(binary semaphore)

* 이 연구는 한국 학술 진흥 재단 신진 교수 학술 연구비 지원 과제의 지원을 받음

$$\begin{array}{ll}
\mathbf{add}\ R_i, R_j, R_k & \stackrel{def}{=} \sum_{1\le l\le 6}\sum_{1\le m\le 6}\{R_i, R_{j_l}, Rk_m\}\ \text{Done} \\
\mathbf{mov}\ R_i, R_j & \stackrel{def}{=} \sum_{1\le l\le 6}\{R_i, R_{j_l}\}:\text{Done} \\
\mathbf{load}\ R_i, R_j, \#c & \stackrel{def}{=} \sum_{1\le l\le 6}\{R_i, R_{j_l}\}\ \{R_i\}.\text{Done} \\
\mathbf{store}\ R_i, R_j, \#c & \stackrel{def}{=} \sum_{1\le l\le 6}\sum_{1\le m\le 6}\{R_{j_l}, R_{j_m}\}\ \{R_{i_l}\}:\text{Done}
\end{array}$$

그림 4.1 명령어 모델링

로 나타내야 했고, 우선순위를 표현하기 위해 CCS 를 위해 개발된 우선순위 연산자[1]를 끌어 들여 명세가 매우 복잡했었다.

2. ACSR

ACSR 의 문법(syntax)은 다음과 같다

$$P \stackrel{def}{=} \mathrm{NIL} \mid A:P \mid P+P \mid P\|P \mid [P]_R \mid P[f] \mid P\Delta_b^t(Q,R,S) \mid rec\,X.P \mid X$$

NIL 은 정지된 프로세스이다. $A:P$ 는 자원을 사용하고 한 시간 단위를 소비하는 시간 행동(timed action)이다 $P+P$ 는 두 프로세스 $\sum_{1\le i\le 6}\{R_i, R_{j_i}\}$: Done 의 $rec\,X.\{\ \}$ 합성(choice composition)을 나타내며 $P\|P$ 는 두 프로세스 사이의 병렬 합성(parallel composition)을 나타낸다. $[P]_R$ 은 자원 폐쇄(resource closure)로서 자원의 집합 R에 포함된 모든 자원이 P에 의해 독점된다. $P[f]$에서 f는 자원에 대한 재표지 함수(re-labelng function)로서 $y_1/x_1, y_2/x_2, \cdots, y_n/x_n$ 으로 나타내며 P의 액션에서 자원 x_n을 y_n으로 바꾼다 $P\Delta_b^t(Q,R,S)$에서 t는 액션이 일어날 때마다 1 씩 줄어들고, 0 이 되면 프로세스 P가 R로 진행한다 $rec\,X.P \mid X$ 는 일반적인 재귀 순환이다

여기서 로 정의되는 프로세스 Done 은 모든 프로세스 P에 대해 $\mathrm{Done}\|P = P$ 인 성질을 갖는다.

3. 모델 프로세서 명세

이 장에서는 모델링 될 프로세서에 대한 가정을 서술한다.

워드 크기 모든 명령어는 전부 한 워드(32 비트)로 구성되며 별도의 워드를 필요로 하지 않는다.

레지스터 명령어는 최대 세 개까지 실행될 수 있고, 여섯 개의 레지스터를 사용한다. 레지스터에는 여섯 개의 포트가 있어서 명령어 프로세스가 읽기 접근 시에는 포트를 하나만 얻으면 되지만 쓰기를 할 때는 여섯 개 모두 필요 하다. 이때 레지스터 자원 R_{i_j}는 레지스터 i 번의 j 번 포트를 나타내며 Ri 라고 하면 Ri_1, Ri_2,...Ri_6까지의 모든 포트를 의미한다. 모든 레지스터 자원의 집합 R을 $R = \{Ri \mid 1 \le i \le 6\}$로 정의한다.

재배열 버퍼 모델 프로세서는 명령어를 3 개까지 재배열한다. 가능한 많은 명령어를 실행하는 것이 가장 최적화된 방법이다

4. 명령어 모델링

여기서 모델링 될 명령어들은 다음과 같다

add Ri, Rj Rk	$Ri \leftarrow Rj + Rk$
mov Ri, Rj	$Ri \leftarrow Rj$
load $Ri, Rj, \#c$	$Ri \leftarrow \mathrm{Mem}[Rj + c]$
store $Ri, Rj, \#c$	$\mathrm{Mem}[Rj + c] \leftarrow Ri$

add Ri, Rj, Rk 는 레지스터 Rj 와 Rk 의 내용을 더하여 그 결과를 Ri 에 저장하고 **mov** Ri, Rj 는 Rj 의 내용을 Ri 로 옮긴다 **load** $Ri, Rj, \#c$ 는 레지스터 Ri 에 메모리 주소 $Rj + c$ 의 데이터를 저장한다 **store** $Ri, Rj, \#c$ 는 **load** 와 반대이다. 앞의 두 명령어는 실행하는데 한 주기가 필요한 데 반해, 메모리 관련 명령어는 두 주기가 필요하다.

명령어 모델링은 명령어들을 레지스터 자원을 소모하는 ACSR 프로세스로 변환하는 것이다 그림 4.1 의 변환식에선 모든 우선순위가 1 로 같으므로 우선순위 표현을 생략했다. 최대한의 병렬성을 얻기 위해 명령어 프로세스는 실행되는 시점에서 유효한 모든 레지스터 자원의 조합을 가진다 즉, $\sum_{1\le i\le 6}\{R_i, R_{j_i}\}$: Done 에서 처럼 R_j 의 읽기 쓰기 포트 여섯 개 중 어느 하나라도 유효하다면 사용 가능하도록 한다.

이렇게 해서 변환된 ACSR 프로세스의 집합을 프로그램 명세라고 한다 다음은 그 예이다

예제 4.1 다음의 메모리 위치에 명령어들이 있다.

```
0:   add R1, R1, R1
4:   load R2, R1, #8
8:   mov R4, R2
12:  add R5, R5, R6
16   store R3, R1, #12
```

위 명령어들은 그림 4.1 의 변환식에 따라 다음처럼 변환된다.

$$\begin{array}{ll}
\mathrm{Mem}(0) & \stackrel{def}{=} \{R1\}.\text{Done} \\
\mathrm{Mem}(4) & \stackrel{def}{=} \{R1_1, R2\}:\{R2\}\cdot\text{Done} \\
\mathrm{Mem}(8) & \stackrel{def}{=} \{R2_1, R4\}:\text{Done} \\
\mathrm{Mem}(12) & \stackrel{def}{=} \{R5.R6_1\}:\text{Done} \\
\mathrm{Mem}(16) & \stackrel{def}{=} \{R1_2, R3_1\}:\{R1_2\}:\text{Done}
\end{array}$$

5. 실행 모델링

실행 모델링은 두 개의 프로세스로 나뉘어 진다. 하나는 명령어들을 실행 단계로 넘기는 *Issue*(PC_1, PC_2, $PC3$) 프로세스, 또 하나는 명령어들을 실행하는 Exec(PC_1, PC_2, PC_3)이다. 각각의 매개 변수 PC_1, PC_2, PC_3는 재배열 버퍼의 명령어 주소 인덱스로서 명령어가 저장된 위치를 가리킨다. Exec 프로세스는 모든 레지스터 자원의 집합에 대해 폐쇄되어 있으므로 언

제나 가정 많은 명령어를 실행하는 쪽으로 행동하게 된다.

$$\begin{array}{ll}
\mathrm{Issue}(PC_1, PC_2, PC_3) & \stackrel{def}{=} \{\,\}:(\mathrm{Issue}(PC_1,PC_2.PC_3)+\mathrm{Exec}(PC_1,PC_2.PC_3)) \\
\mathrm{Exec}(PC_1, PC_2, PC_3) & \stackrel{def}{=} \mathrm{Mem}(PC_1)\|\mathrm{Mem}(PC_2)\|\mathrm{Mem}(PC_3)\|\mathrm{Issue}(PC_3+4,PC_3+8,PC_3+12) \\
 & \quad +\mathrm{Mem}(PC_1)\|\mathrm{Mem}(PC_2)\|\mathrm{Issue}(PC_3,PC_3+4.PC_3+8) \\
 & \quad +\mathrm{Mem}(PC_1)\|(\mathrm{Mem}(PC_2)\|\mathrm{Mem}(PC_3))[f]\Delta_1(\mathrm{NIL},\mathrm{Done},\mathrm{NIL})\|\mathrm{Mem}(PC_3) \\
 & \quad \|\mathrm{Issue}(PC_2,PC_3+4,PC_3+8) \\
 & \quad +\mathrm{Mem}(PC_1)\|\mathrm{Issue}(PC_2,PC_3,PC_3+4) \\
 & \quad +\mathrm{Mem}(PC_1) \\
\mathrm{Program} & \stackrel{def}{=} [\mathrm{Issue}(0,4,8)]_R
\end{array}$$

$$f=\{Si_j/\mathrm{R}\imath_j \mid 1\le \imath \le 6, 1\le j\le 6\}$$

그림 5.1 실행 모델링

슈퍼 스칼라 프로세서에서 한 번에 두 명령어가 하나의 레지스터에 대해 읽고 쓰기를 겹쳐서 하는 경우를 데이터 해저드라고 한다. 이러한 데이터 헤지드는 모델링에서 두 명령어 프로세스 사이의 자원 충돌로 나타난다

예제 5.1 첫번째 주기에서 메모리 주소 0, 4 8의 명령어들이 실행 유니트에 게시 된다.

$$\mathrm{Program}=[\{\,\}:\mathrm{Exec}(0,4,8)]_R$$
$$=\left[\{\,\}:\begin{pmatrix}\mathrm{NIL}+\mathrm{NIL}+\mathrm{NIL}+\mathrm{NIL}+\{\mathrm{R1}\}:\\(\mathrm{Issue}(4,8,12)+\mathrm{Exec}(4,8,12))+\{\mathrm{R1}\}\cdot\\\mathrm{Done}\end{pmatrix}\right]_R$$

0 과 4, 4 와 8 이 각각 서로 충돌한다.

$$\mathrm{Program}=\left[\{\,\}\ \{\mathrm{R1}\}:\begin{pmatrix}\mathrm{NIL}+\mathrm{NIL}+\{\mathrm{R1}_1,\mathrm{R2},\mathrm{R5},\mathrm{R6}_1\}:\\\begin{pmatrix}\{\mathrm{R2}\}:\mathrm{Done}\\\|\begin{pmatrix}\mathrm{Issue}(8,16,20)\\+\mathrm{Exec}(8,16.20)\end{pmatrix}\end{pmatrix}\\+\mathrm{NIL}+\cdots\end{pmatrix}\right]_R$$

여전히 4와 8이 충돌하므로 4와 12가 병렬로 실행된다.

6. 결론

이상에서와 같이 ACSR이라는 프로세스 알제브라를 이용함으로써, 까다로운 비순차 슈퍼 스칼라 프로세서의 타이밍 분석을 편리하게 나타낼 수 있다. 앞으로는 좀더 실제적인 프로세서에 근접한 모델링이 가능하도록 기능 유니트나 파이프 라인 단계가 세분화되어야 할 것이다.

7. 참고 문헌

[1] J. Camilleri and G Winskel "CCS with Priority Choice". In *Proc. of IEEE Symposium on Logic in Computer Science*, 1991

[2] D. Clarks, I. Lee, and H Xie "VERSA: A Tool for the Specification and Analysis of Resource-Bound Real-Time Systems" Technical Report MS-CIS-93-77, Dept. of CIS. Univ of Pennsylvania. Sept 1993.

[3] T. Cook, P franzon, E Harcourt, and T. Miller "System-Level Specification of Instruction Sets" In *Proc of the Internatinol Conference on Computer Design*, 1993.

[4] E. Harcourt, J. Mauney, and T Cook "Specification of Instruction-Level Parallelism". In *Proc of the North American Process Algebra Workshop*. 1993.

[5] M Johnson. "Superscalar Microprocessor Design". Prentice-Hall, 1991.

[6] I Lee, P. Brémond-Grégoire, and R Gerber "A Process Algebraic Approach to the Specification and Analysis of Resources-bound Real-time Systems". Technical Report MS-CIS-93-08, Univ of Pennsylvania, January 1993 To appear in *IEEE Proceedings*, Jan 1994

[7] D Patterson and J. Hennesy. "Computer Architecture. A Quantitative Approach" Morgan Kauffman, 1990

[8] R Rau and J Fisher "Instruction-Level Parallel Processing History, Overview, and Perspective". *Journal of Supercomputing*, Jul, 1993.

[9] Jin-Young Choi, Insup Lee, and Inhye Kang "Timing Analysis of Superscalar Processor Programs Using ACSR", *IEEE Real-Time Systems Newsletter*, Volume 10, No 1/2. 1994.

[10] 이기훈, 최진영. "분기 명령어를 포함한 슈퍼스케일러 프로세서의 타이밍 분석", 정보과학회 춘계 학술 대회. 1997.