Empress 프로세서를 위한 DFA 기반 명령어 스케줄링 [1)]

정동하*, 이호균*, 김선욱*, 김관영**
*고려대학교 전자공학과
**에이디칩스㈜
e-mail : djung@korea.ac.kr, hokyoon79@korea.ac.kr, seon@korea.ac.kr, kevinkky@adc.co.kr

DFA based Instruction Scheduling for Empress Processor

Dongha Jung, Hokyoon Lee, Seon Wook Kim, Kwan Young Kim
Dept. of Electrical Engineering, Korea University
Advanced Digital Chips Inc.

요 약

최근 스마트 폰, 타블렛 PC 등 고성능 모바일 기기에 대한 시장수요가 증가함에 따라 임베디드 프로세서에 대한 성능 최적화가 활발히 이루어지고 있다. 고성능 멀티미디어 시스템을 대상으로 설계된 Empress 프로세서는 다수의 기능 유닛을 포함하고 있어 명령어 스케줄링을 통해 성능 향상을 기대할 수 있으나 기존의 컴파일러는 이를 지원하지 않고 있다. 본 연구에서는 GNU C 컴파일러를 이용하여 Empress 프로세서를 위한 DFA 기반의 명령어 스케줄링 최적화를 구현하였다. 그 결과 EEMBC 벤치마크를 이용한 성능 분석에서 실행시간 기준 평균 8%의 향상이 있음을 확인하였다.

1. 서론

IPC(Instruction Per Cycle)는 프로세서가 한 사이클 동안 실행한 명령어 수의 평균을 의미하며 프로세서의 자원이 얼마나 효율적으로 활용되고 있는지를 보여준다. 다수의 기능 유닛(Functional Unit)을 포함한 프로세서의 IPC 를 저해하는 요소는 명령어간의 의존에 의한 지연(Data Hazard)과 기능 유닛의 부족에 의한 지연(Structural Hazard)이며, 이는 명령어들의 순서를 바꾸는 것으로 숨길 수 있다. 명령어 스케줄링 최적화(Instruction Scheduling Optimization)는 컴파일러의 기법을 통해 추가적인 하드웨어 없이 성능향상이 가능하다.

Empress 프로세서의 경우 1-way 이슈(Issue) 프로세서이나 다수개의 기능 유닛을 가지며 각 유닛들마다 다른 사이클이 소요되므로 명령어 스케줄링을 통한 성능향상을 기대할 수 있다. 본 논문에서는 GNU C 컴파일러에 포함된 DFA (Deterministic Finite Automata)기반 명령어 스케줄러를 이용하여 EISC 컴파일러에 명령어 스케줄링을 구현하였고 EEMBC 벤치마크를 이용해 결과 코드의 성능을 측정하였다.

2. DFA 기반 명령어 스케줄링

최적의 명령어 스케줄은 최소한의 실행시간을 의미하며 이를 찾는 것은 NP-hard 문제이다. 상용 컴파일러에서는 실용성을 위해 결과의 최적을 포기하는 대신 합리적인 컴파일 시간이 보장되는 휴리스틱 알고리즘을 사용하고 있다. GNU 컴파일러는 매 사이클에 이슈 할 수 있는 명령어 수를 최대화 하는 것을 목표로 하는 'The First Cycle Multi-pass' 알고리즘을 사용하고 있다[1]. 이 알고리즘은 프로세서의 파이프라인이 가질 수 있는 모든 상태를 오토마타로 표현하고 각 명령어 입력에 의한 전이를 시뮬레이션 하여 이슈 되는 명령어 수를 최대화할 수 있는 명령어 순서를 찾는다.

컴파일러는 프로세서에 포함된 기능 유닛의 수와 각 유닛이 필요로 하는 사이클 수, 그리고 각 명령어가 어떤 기능 유닛을 사용하는 지를 입력 받고 이를 바탕으로 오토마타를 생성한다. 생성된 오토마타는 DFA 나 NFA (Non-deterministic Finite Automata)가 될 수 있는데, DFA 의 경우 가능한 스케줄을 하나씩 순차적으로 시도하지만, NFA 의 경우 시도 가능한 모든 스케줄을 동시에 시도하게 되므로 DFA 보다 느리게 동작한다. GCC 는 모든 NFA 를 DFA 로 바꾸어 사용하여 알고리즘의 속도를 향상시키고 있다[2].

3. Empress 프로세서의 파이프라인 구조

명령어 코드에 임의의 상수 값을 실을 수 있도록 하여 메모리 접근을 줄인 EISC(Extendible Instruction Set Computer)프로세서는 코드의 압축성과 전력효율 면에서 큰 장점을 지니고 있다[3]. 본 연구에서 사용한 EISC 프로세서는 고성능 멀티미디어 처리를 위해 설계된 Empress 프로세서이며 파이프라인은 그림 1 과 같이 ALU, MULTIPLY, 그리고 LD/ST 유닛으로

1) 본 연구는 한국산업기술평가원의 시스템 IC2010 사업(10030565-2010-04)으로 지원되었음을 밝힙니다.

구성되어 있다[4]. 각 유닛의 실행 사이클은 ALU 의 경우 1 사이클, 나머지 두 유닛들은 3 사이클이 소요되며 내부적으로 다시 파이프라인으로 구성되어 연속된 동일 명령어를 처리할 수 있도록 설계되어 있다.

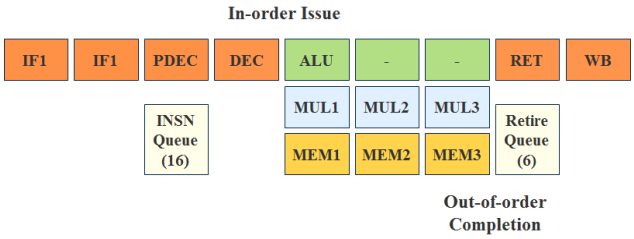

그림 1. Empress 프로세서의 파이프라인 구조

4. Empress 파이프라인의 DFA

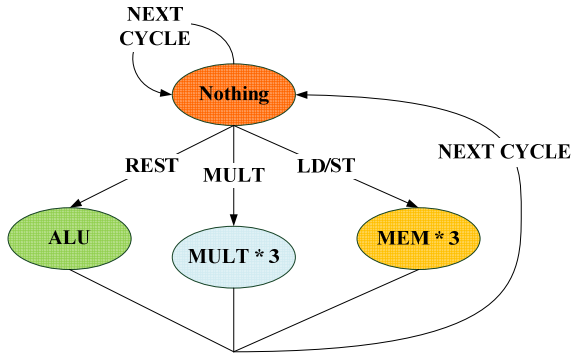

그림 2. Empress 파이프라인의 DFA

파이프라인은 GCC 의 MD (Machine Description) 파일에서 제공하는 구조체들을 사용하여 모델링 할 수 있고, GCC 는 이를 기반으로 DFA 를 생성한다. Empress 컴파일러는 기능 유닛을 표현하는 오토마타 선언과 명령어의 속성 정의를 통해 파이프라인을 모델링 한다. 속성 정의는 해당 명령어가 몇 사이클 동안 어떤 기능 유닛을 사용하는지를 표현하는데 사용된다. 그림 2 는 Empress 컴파일러에서 정의한 파이프라인의 DFA 를 보여주고 있다. MULTIPLY 와 LD/ST 연산은 3 사이클 동한 한 상태에 머물게 되며 나머지 연산은 한 사이클 안에 실행을 완료하게 된다.

GCC 는 최소화를 통해 DFA 의 중복된 상태를 줄인 다음, 각 상태에서의 입력에 따른 전이를 테이블 형태로 생성한다. 테이블은 GCC C 언어의 switch 문을 이용해 구현되며, 'The First Cycle Multi-pass' 알고리즘이 이슈 넓이(Issue Width)를 최대화 하는 명령어를 선택하는 데 사용된다. 알고리즘이 빠르게 동작하기 위해서는 각 명령어가 이슈 될 수 있는지 여부를 따져 명령어 후보 군을 파악해야 하는데 테이블을 이용해 그 여부를 빠르게 파악할 수 있다.

5. 스케줄링 예제

Empress 컴파일러 명령어 스케줄링은 멀티 사이클 명령어와 이에 의존적인 명령어들을 대상으로 적용된다. LOAD 와 MULTIPLY 연산의 경우 3 사이클이 소요되며 둘에 의해 정의되는 값을 사용하는 명령어가 스톨(stall)없이 실행되기 위해선 다른 명령어들을 사이에 위치시켜야 한다. 그림 3 의 예제는 명령어 스케줄링 전후의 어셈블리 코드를 보여주고 있다.

스케줄링 전의 코드에서 마지막 두 명령어는 R9 레지스터에 값을 읽고 다시 저장 하는데 두 명령어 사이의 거리가 1 밖에 되지 않아 STB 명령어가 2 사이클 지연된다. 스케줄러는 R8 값을 계산하는 두 명령어(MOV 와 ADD)를 LDB 와 STB 사이에 위치시켜 로드로 인한 지연을 숨기고 있다.

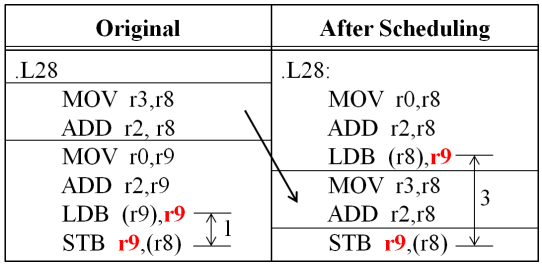

그림 3. 스케줄링 예제

6. 성능 평가 및 분석

명령어 스케줄링 최적화의 성능을 측정하기 위해 최적화가 적용된 코드의 정확도와 IPC(Instruction Per Cycle), 그리고 프로그램 실행 시간을 비교하였다. 정확도 검사에는 GCC 에 포함된 테스트 묶음(Testsuite)을 사용하였으며 모든 코드가 문제없이 컴파일 되고 실행됨을 확인하였다.

IPC 와 실행시간 측정에는 ESCAsim 시뮬레이터를 사용하였다[5]. ESCAsim 시뮬레이터는 Empress 프로세서의 파이프라인을 정확히 모델링하여 파이프라인 위험(Pipeline Hazard)을 모두 감지할 수 있으며 사이클 단위까지 정확한 실행시간 측정이 가능하다. 벤치마크로는 임베디드 시스템의 성능을 측정하는데 많이 사용되는 EEMBC 벤치마크를 사용하였다[6].

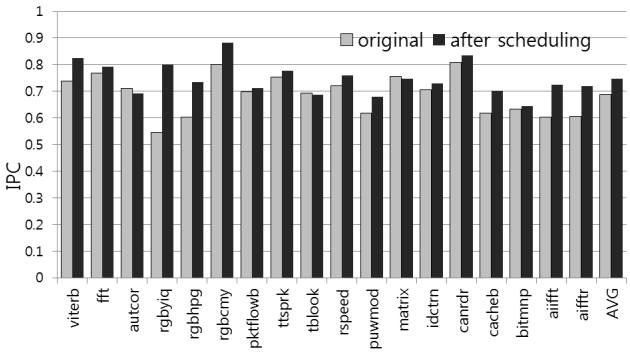

그림 4. 명령어 스케줄링 전후의 IPC 변화

그림 4 은 최적화 적용 전후의 IPC 를 비교한 결과이다. 몇몇(autcor, tblook, matrix)을 제외한 대부분의 벤치마크에서 IPC 가 향상되었고 (평균 7%), 특히 rgbyiq 의 경우 IPC 가 0.54 에서 0.79 로 크게 향상되었다.

명령어 스케줄링은 다른 컴파일러 단계에 영향을 미치며, EISC 컴파일러의 경우 레지스터 할당(Register Allocation)과 핍홀 최적화(Peephole Optimization)가 직접적 영향을 받은 것으로 확인되었다. 그림 4 에서 IPC 가 감소한 autcor 이 첫 번째 경우로 상이한 레지스터 할당으로 인해 추가적인 MOV 명령어가 발생하였다.

EISC 컴파일러는 SP 레지스터의 값을 일반 레지스터로 옮기는 명령어와 이 일반 레지스터에 상수 값을 더하는 명령어가 일렬로 나타나면 핍홀 최적화를 통해 두 명령어를 하나의 LEA 명령어로 대체시킨다. 하지만 명령어 스케줄링을 적용하게 되면 두 명령어가 떨어져 핍홀 최적화가 적용되지 못할 수도 있고 그 결과 명령어 수가 증가할 수도 있다. 따라서 명령어 스케줄링이 실제로 효과가 있는지를 파악하기 위해 실제 실행시간을 측정 및 비교하였다.

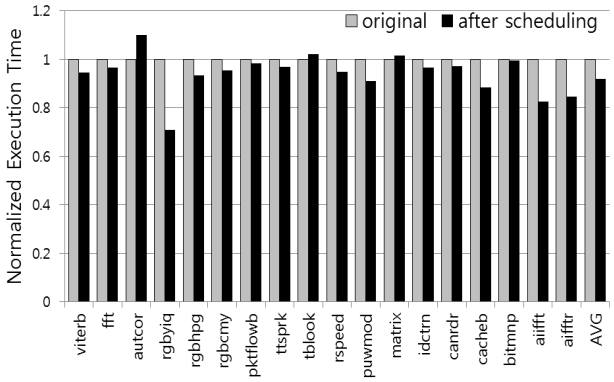

그림 5. 명령어 스케줄링 전후 실행 시간 변화

그림 5 는 EEMBC 벤치마크의 실행 시간을 측정 결과이며, 스케줄링 적용 전의 결과에 표준화 (Normalized)하였다. 위의 결과에서, 최대 30% (rgbyiq)까지 성능향상이 있음을 볼 수 있고, 평균적으로 8%의 실행 시간 단축이 있음을 확인 하였다.

7. 결론

명령어 스케줄링은 한정된 파이프라인 자원을 최적으로 사용할 수 있도록 컴파일러에서 분석을 통해 이루어 진다. 최적의 스케줄을 찾기가 어려운 만큼 실용적이면서도 효과적인 체험적 알고리즘 선택이 중요하며 GNU 컴파일러의 DFA 기반의 파이프라인 모델을 효과적으로 이를 구현하였다. 레지스터 할당 및 핍홀 등 다른 컴파일러 단계들이 영향을 받았지만 사이클 단위 시뮬레이터를 사용한 성능 측정에서 상당한 성능향상이 있음을 확인하였다.

참고문헌

[1] Vladimir N. Makarov, “The finite state automaton based pipeline hazard recognizer and instruction scheduler in GCC”, GCC Developers Summit, pp.135-149, Ottawa, Canada, 2003.

[2] V. Bala and N. Rubin, “Efficient Instruction Scheduling Using Finite State Automata”, International Journal of Parallel Programming, Vol.25, No.2, pp.53-82, 1995.

[3] H. Lee, P. Beckett, and B. Appelbe, “High-Performance Extendable Instruction Set Computing”, Proceedings of 6th ACSAC, pp.89-94, Queensland, Australia, 2001.

[4] ADC Corporation, http://adc.co.kr.

[5] Hyun Gyu Kim and et. al., “AE32000B: A Fully Synthe-sizable 32-Bit Embedded Microprocessor Core,” ETRI Journal, Vol.25, No.5, pp.337-344, 2003.

[6] EEMBC, http://www.eembc.org.