소프트웨어 파이프라이닝에서 레지스터 변경을 통한 성능 개선

조두산
순천대학교 전자공학과
e-mail : dscho@sunchon.ac.kr

Improving Software Pipelining Performance Using a Register Renaming Technique

Doosan Cho
Dept. of Electrical Engineering, Sunchon National University

요 약

멀티미디어 도메인의 응용 프로그램에는 풍부한 병렬성이 내재하기 때문에 VLIW (Very Long Instruction Word) 형식의 신호처리 프로세서가 많이 사용되고 있다. VLIW 프로세서를 구성하는 복수의 연산처리유닛 (processing unit, PU)의 사용률은 컴파일러의 명령어 스케쥴러의 성능에 의하여 결정된다. 명령어들 사이의 병렬성을 분석하여 동시 수행가능한 명령어들을 각기 다른 PU 에서 수행되도록 프로그램 코드를 최적화한다. 하지만 기존의 명령어 스케쥴러는 복잡한 데이터 디펜던스 그래프 (data dependence graph, DDG)를 구성하여 복수의 PU 를 충분히 사용하도록 하지 못하는 문제점을 내재하고 있다. 이는 명령어 스케쥴러가 각 레지스터 사용시간을 별도로 고려하지 않기 때문에 실제로 내재된 데이터 디펜던스 보다 복잡성이 높은 디펜던스 그래프를 구성하게 되어 스케쥴러가 올바르게 최적화된 코드 스케쥴링 결과를 제공하지 못하기 때문이다. 본 연구에서는 레지스터의 라이프타임을 다른 레지스터를 이용하여 적절히 끊어주는 것으로 데이터 디펜던스 복잡도 완화하여 시스템 성능 향상의 가능성을 보이고 있다.

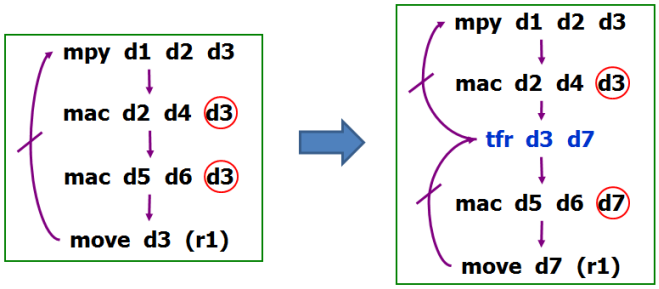
1. 서론

반복 모듈로 스케쥴링 (Iterative Modulo Scheduling) [1] 은 소프트웨어 파이프라이닝을 구현하는 한 방법으로서 어플리케이션의 실행시간을 개선하는데 목적이 있다. 반복 모듈로 스케쥴링을 적용하기 위해서는 우선 입력된 코드를 분석하여 DDG 를 구성하고 디펜던스가 가용하는 범위에서 MII (Minimum Initiation Interval)을 계산한다. MII 는 루프 코드가 올바른 결과를 제공하는 범위에서 최대한의 루프 중첩의 정도를 의미한다. 즉, 더 작은 MII 를 얻을 수 있다면 더 많은 루프 이터레이션 (iteration)을 중첩시킬 수 있어 실행시간을 개선시킬 수 있게 된다 [2].

우리는 몇몇 미디어, 통신용 응용 프로그램을 대상으로 반복모듈로 스케쥴링을 적용하면서 MII 가 실제 내재된 병렬성을 제대로 반영하지 못하는 문제를 파악하였다. 문제는 레지스터 할당기에 의하여 긴 레지스터 라이프타임을 생성함으로 인하여 DDG 의 노드간 에지의 개수가 많아지고 결과적으로 MII 가 가능한 값보다 높게 계산되는 것이다. 이러한 문제를 해결하기 위하여 스케쥴링 전처리단계에서 레지스터 변경 (renaming) 기법을 적용하였다.

2. 문제정의 및 제안된 알고리즘

[1][2]에 기술된 바와 같이 MII 는 리소스의 개수에 의하여 계산되는 ResMII 와 DDG 상의 백에지 (backward edge)에 의하여 생성되는 디펜던스 사이클로 계산되는 RecMII 값을 사용하여 MII = Max(ResMII, RecMII)와 같이 계산된다 [3]. 본 연구에서는 ResMII < RecMII 의 경우만을 고려하기 때문에 MII = RecMII 로 결정된다. 따라서 MII 가 디펜던스 제약조건 하에서 최소 값을 갖도록 하기 위하여 디펜던스 사이클 패스의 길이 (혹은 에지의 지연시간-delay time)을 줄이는 것이 문제의 핵심이 된다.



(그림 1) 레지스터 변경 예제

그림 1 의 왼쪽은 루프 코드에서 DDG 를 나타내고 있다. 레지스터 d3 에 의하여 긴 디펜던스 사이클이 구성됨을 확인할 수 있다. 레지스터 변경을 통한 라이프타임 스플리팅을 그림 1 의 오른쪽에 나타난 바와 같이 적용하면 디펜던스 사이클의 길이를 줄일 수 있게 된다. 그림 1 의 예제의 경우 MII 가 왼쪽 3cycle 에서 오른쪽 2cycle 로 축소되었다. 결과적으로 해당 루프코드가 100 만번 수행된다면 100 만 사이클타임을 개선 시키는 결과를 얻을 수 있게 될 것이다 [4].

문제정의: 라이프타임 스플리팅 적용할 시점을 어떻게 정할 것인가?

디펜던스 사이클 패스에서 어느 시점에서 사이클을 분할 해야 가능 작은 MII 를 얻을 수 있을지를 분석해야하는 것이 본 연구의 핵심이다. 우리는 다음과 같은 비용 함수를 통하여 이 문제를 해결하였다.

Benefit = RecMII – Max(split_ddgCycles)

위의 비용함수에서 split_ddgCycle 은 레지스터 변경에 의하여 분리된 디펜던스 사이클들을 갖는 집합이며 Max 함수에 의하여 최대 RecMII 를 갖는 값이 리턴된다. 리턴된 값과 본래 디펜던스 사이클의 ReCMII 와 차를 통하여 얼마나 MII 를 개선시킬 수 있는지를 Benefit 에 저장한다. 디펜던스 사이클을 따라 Benefit 을 반복하여 계산하고 최대값을 갖는 지점을 솔루션으로 정하여 적용하게 된다. 알고리즘은 그림 2 에 나타난 바와 같다.

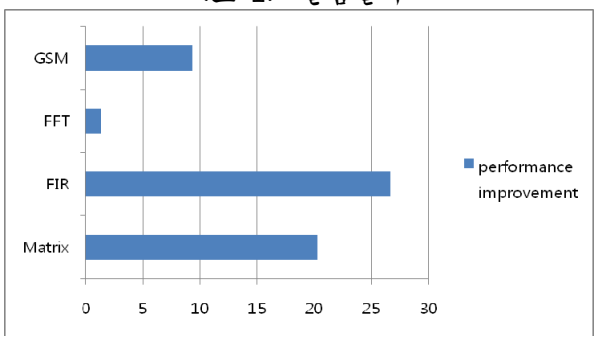
```
Procedure Split_search(unsigned int *DDG)
//DDG: data dependence graph
// build priority queue with dependence cycles
PQ = Build_priority_queue(ddg);
oldBenefit=0;
newBenefit=0;

 While(Nonempty(PQ)){
    Cur_cyclePath = Pop(PQ);

    While(EndOfCycle(Cur_cyclePath)){
      SplitDdgCycleMax=ApplyingSplit(Cur_cyclePath);
      newBenefit = RecMII-RecCalc(SplitDdgCycleMax);

      if(oldBenefit < newBenefit)
        oldBenefit = newBenefit;
      Cur_cyclePath++;
    }
 }

  return oldBenefit; // the best solution
End Procedure Split_search
```

(그림 2) 최적 스플리팅 지점 검색 알고리즘

3. 실험결과

제안하는 알고리즘을 검증하기 위하여 우리는 FFT, Matrix, GSM, FIR 등과 같은 멀티미디어 혹은 통신 관련 벤치마크 코드들을 사용하였다. 실험은 스타코어사의 SC140 VLIW 형식의 프로세서 [5]에서 매뉴얼하게 진행되었다. 표 1 은 적용전/후 결과를 비교하여 % 로 나타낸 것이다.

<표 1> 실험결과



실험 결과는 위 표 1 에 나타나 있다. 평균 14% 정도의 성능 개선 결과를 나타내고 있다. 스케쥴러 알고리즘을 수정하지 않고 스케쥴러의 입력이 되는 루프 커널 코드를 최적화하여 얻은 결과로서 의미가 있다. 본 실험에서 성능이 저하된 경우는 없었으나 PU 가 2 개 정도로 적을 경우 ResMII 의 증가로 인하여 성능이 일부 저하 될 수도 있다.

Acknowledgment
이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(No. 2010-0024529). 본 지식재산권은 지식경제부 및 정보통신산업진흥원의 지원을 받아 수행된 연구결과임 (10-기반, 기술혁신사업).

참고문헌

[1] B. Rau and D. Glaser: Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing, in *Proceedings of the 14 Ann. Microprogramming Workshop*, Nov 1981.

[2] M. Lam: Software pipelining: an effective scheduling technique for VLIW machines, in *Proceedings of the SIGPLAN'88 Conference on Programming Language Design and Implementation*, June, 1988.

[3] E. Stotzer and E. Leiss: Modulo Scheduling for the TMS320C6x VLIW DSP Architecture. in *Proceedings of the SIGPLAN'99 Workshop on Languages, Compilers, and Tools for Embedded Systems*, May 1999.

[4] D. Lavery and W. Hwu: Unrolling-Based Optimizations for Modulo Scheduling, in *Proceedings of the 28th annual international symposium on Microarchitecture*, page 327-337, 1995.

[5] T. Halfhill: Motorola enhances StarCore DSP. In Microprocessor Report, INSTAT/MDR, www.MPRonline.com, Oct 20, 2003.