공학 연구논문집 제22권 제2호 pp. 67~73, 1991. 울산대학교
Journal of Engineering Research Vol. 22, No. 2, pp. 67~73, 1991.
University of Ulsan.

파이프라인형 수퍼컴퓨터에서의 컴파일링 최적화 연구

구 자 록
전자계산학과

〈요 약〉

본 논문은 파이프라인형의 수퍼컴퓨터에서 효율적인 컴파일링을 통한 병렬성을 향상시키기 위한 기법으로 스칼라 컴파일링 기법인 소프트웨어 파이프라이닝(Software Pipelining)과 루프 펼침(Loop Unrolling)기법을 채택하여 성능향상을 꾀하고자 한다. 또한 다중 명령어 발생 기법의 성질과 구조를 고찰한 뒤, 프로그램 상에 존재하는 오버헤드를 제거한다.

A Study on the Compilation Optimizing Techniques for Pipelined Supercomputers

Koo, Ja-Rok
Dept. of Computer Science

〈Abstract〉

This paper studies new compilation techniques for enhancing performance in high-speed superpipelined processors using both software pipelining and loop unrolling. The discussion of the machine parallelism is based on the concepts of superscalar and superpipelined machines. By examining the structure and characteristics of these multiple-instruction-issue schemes, we can isolate potential overhead sources, and propose compiling enhancements that reduce their impact on performance.

I. 개 요

컴퓨터 기술자와 설계를 하는 디자이너들은 성능 향상을 위해 기계에서 다양한 수준의 병렬성을 추구하여왔다. 이 병렬성 구현의 기본은 여러 개의 명령어들을 수행함에 있어서 중첩(Overlap)을 이용하는 것이다. 단일 프로세서에서는, 같은 기능단위(Functional Unit)를 사용하는 여러 개의 명령어를 중첩되게 수행하기 위한 기능단위들의 파이프라이닝과 다른 기능단위들을 사용하는 여러 개의 명령어를 중첩되게 수행하기 위한 다양한 종류의 다중 기능단위들의 파이프라이닝이 있다. 또한 기억장소를 이용한 연산들의 수행을 중첩되게 처리하기 위한 파이프라인형 또는 인터리브드(Interleaved)형의 기억장소 시스템이 있다[13, 15].

기존의 하드웨어가 동시에 여러 개의 명령어를 수행할 수 있다고 할지라도, 실제 명령어의 수행기능은 명령어 발생체계(Instruction Issue Mechanism)에 달려있다. 전통적으로, 프로세서들은 한 클럭 사이클에 최대 1개의 명령어를 발생시키려는 시도를 해왔다. 경우에 따라서는 이러한 사이클 당 한 개의 명령어의 한계를 플린의 한계(Flynn's limit)라고 일컫는다[7]. 데이타 및 콘트롤 종속성은 이러한 명령어 발생체계로 얻을 수 있는 이론상 최고의 성능을 저하시킬 수 있다.

기계어 사이클당 최대 한 개 명령어를 발생시킬 수 있는 시스템에서 얻을 수 있는 성능의 한계성 때문에 더욱 더 많은 관심이 사이클당 여러 개의 명령어를 발생시킬 수 있는 방법에 집중되어 왔다[1, 2, 3]. 사이클당 다중 명령어를 발생시키는 방식이 하위 수준의 병렬성을 구현하는 것으로 볼 수 있어, 최근에 많은 관심을 불러 일으키고 있다. 연구되고 있는 다중 명령어 발생 방식 중의 하나가 수퍼 파이프라인형 기계에 관한 연구이다[1, 2, 3].

대규모 과학 계산용 컴퓨터에서의 컴파일링 기법의 최근 연구들은 주로 벡터자동화 분야에 집중되어 온 것이 사실이다[11, 12]. 그럼에도, 더 고전적인 스칼라 형태의 연산에서 구현할 수 있는 성능이 벡터처리에 의한 성능에 못지 않게 중요하게 인식되고 있다. 이 논문은 간과할 수 있는 문제인 스칼라 컴파일링 기법의 중요성을 수퍼 파이프라인형 기계에의 적용에 관한 연구이다. 고기능의 스칼라 컴파일링 기법을 체계적으로 설명하고자 하는 것이 이논문의 주요 목표이다.

II. 수퍼 파이프라인형 기계

1. 기본 기계

명령어 수준의 병렬성을 구현함으로써 성능 향상을 꾀하고자 하는 이러한 기계의 비교를 위해, 비파이프라인형 기본 기계를 다음과 같이 정의한다:

사이클 당 명령어 발생 수=1
간단한 연산 수행 시간(사이클 단위)=1
최상의 이용에 필요한 명령어 수준의 병렬성=1

한 사이클 수행 시간이란, 하나의 명령어가 다른 명령어를 따를 때, 첫 번째 수행결과가 지체없이 다음 번 명령어에 사용 가능한 것을 의미한다. 그래서, 기본 기계에는 어떠한 연산 수행시간의 중단이나 비수행 연산(예, NOP)이 있을 수 없다. 그림 1.이 기본 기계의 파이프라인을 보여 주고 있다. 비파이프라인형 수행단계로 인해, 한 번에 오직 한 개의 명령어만이 수행단계에 존재할 수 있다. 명령어 패치(Fetch), 해독(Decode), 재기록(Store)과 같은 다른 파이프 단계들은, 그들이 우회하여 지나간다면 명령어 수행시간에 영향을 미치지 않을 것이다. 두 개의 명령어를 발생시키는 시간적인 차이를 발생 수행시간이라 한다. 이 시간은 두 개 명령어의 클래스에 따라 차이가 있다. 이 명령어 클래스란 같은 유

형의 기능 장치에 발생이 이뤄지는 명령어들의 집합을 의미한다. 예를 들면, 레지스터 간의 이동에 필요한 시간은 부동소숫점 연산에 필요한 시간보다 적을 것이다.

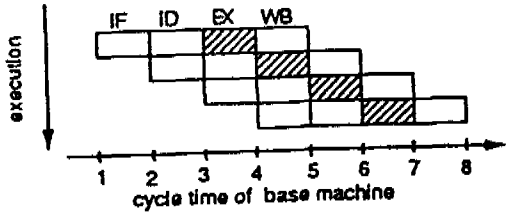

그림 1.

2. 수퍼 파이프라인형 기계

수퍼 파이프라인형 기계는 명령어 수행단계를 파이프라이닝하여 명령어 수준의 병렬성을 구현한다[1, 2, 3, 14, 16]. 각각의 단계는 작은 단위의 파이프라인 세그먼트로 나눠진다. m-정도 수퍼 파이프라인형 기계에서는 사이클 시간이 기본 기계의 1/m이다. 연산 수행시간은 이 기계의 사이클 시간으로 m이다. 하나의 명령어를 기본 기계에서 수행하는 데 하나의 사이클이 필요할 때, 같은 방법의 구현 기법으로 수퍼 파이프라인형 기계에서는 m 사이클이 필요하다. 그림 2는 수퍼 파이프라인형 기계에서 명령어의 수행을 보이고 있다. 이 그림을 보면, 세 번째 명령어가 발생될 시기에는 동시에 세 가지의 연산이 수행되고 있음을 볼 수 있다. 수퍼 파이프라인형 기계를 다음과 같이 정의한다:

사이클 당 명령어 발생 수 = 1
(사이클 시간은 기본 기계의 1/m)
간단한 연산 수행 시간(사이클 단위)=m
최상의 이용에 필요한 명령어 수준의 병렬성=m

수퍼 파이프라인형 기계는 오랜 기간 존재하여 왔다. 사이머 크레이(Cymor Cray)는 수퍼 파이프라인형 기계를 제작한 오랜 경험을 갖고 있다. 예를 들면, CDC 6600과 Cray-1에서는 고정 소숫점 덧셈의 연산에 3 사이클이 필요하다. 6600 기계의 기능 장치들은 파이프라인형이 아니고 중복되어 있기 때문에, 6600은 클래스 충돌을 갖는 기계의 예이다. CDC 7600은 기능 장치들이 파이프라인형으로 이뤄져 있기 때문에, 아마도 현존하는 가장 순수한 수퍼 파이프라인형 기계일 것이다.

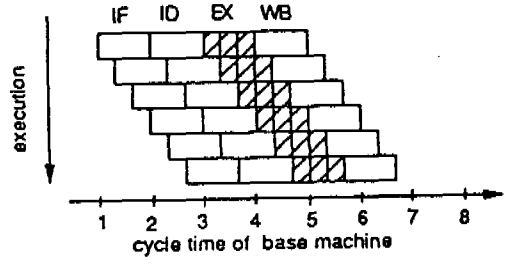

그림 2.

III. 스칼라 컴파일링 기법

1. 루프 펼침(Loop Unrolling)

루프 펼침은 수년 동안 사용되어 왔다. CDC6600, 7600, 그리고 CYBER 200에서, STACKLIB 루틴은 벡터화 할 수 없는 루프 구조를 최적화하기 위한 목적으로 사용되어 왔다. STACKLIB 루틴은 전형적으로 하나의 루 프 반복(Loop Iteration)마다 한 개 이상의 결과를 산출하는 펼침 루프를 갖는다. Trace-scheduling global compaction 기법에서는 루프 펼침이 중요한 역할을 한다[11, 12]. Dongarra와 Hinds는 어느 정도의 성능 향상을 위해 포트란(FORTRAN)으로된 루프를 펼치는 기법을 설명하고 있다. 그들은 수작업으로 루

프의 내용물을 여러 번 복사하여, 배열과 루프의 증분을 조정하도록 제안하고 있다. 여기서는 컴파일러에 의한 자동 스케줄링 관점에서 루프 펼침을 논하고자 한다. 루프 펼침은 코드를 재구성하고 재순서화하는 것으로 이뤄져 있다. 재구성하는 절차는 다음의 단계로 구성된다:

(1) 루프 몸체를 n 번 반복 복사한다.
(2) 복사된 블럭의 마지막 부분을 제외하고는 모든 부분에서 인덱스화된 레지스터 증가 명령어들을 제거한다. 남은 부분의 증분은 n배 한다. 로드 및 스토아(Load and Store) 명령어들의 옵셋(Offset)은 원래 위치한 복사된 블럭에 맞추어 다시 조정한다.
(3) 루프 카운터 증가 명령어들도 마지막 블럭을 제외하고는 모든 복사된 블럭에서 제외시킨다. 남아 있는 것은 n배 한다.
(4) 조건부 분지(Conditional Branch) 명령어들도 마지막 블럭을 제외하고는 모든 복사된 블럭에서 제외시킨다.

루프 펼침의 명확한 두 가지 장점은,

(1) 상당한 수의 분지 명령어를 제거한다. 파이프라인형 프로세서에서는, 이 분지가 명령어 파이프라인에서의 명령어들의 흐름을 부숴버리기 때문에 조건부 분지로 인하여 치러야 할 댓가가 매우 크다.
(2) 인접한 블럭간의 몇몇 명령어들을 제거하여 발생시켜야 할 전체 명령어의 수를 줄일 수 있다.

파이프라인형 프로세서에게 무엇보다 중요한 이익은 기본 블럭의 크기를 증가시킴으로써 더 나은 순서화를 가능하게 한다. 이것은 루프 펼침의 순서화 단계로 유도한다. 루프 펼침의 순서화 단계에서는 CDC6600 FORTRAN 컴파일러에서 구현한 것과 유사한 순서화 알고리즘을 새로운 기본 블럭에 적용하여 최종의 순서화된 코드를 만들어 낸다. 컴파일러의 한 부분으로 구현할 때는, 스케줄러가 로드 및 스토아 명령어들의 이동에서의 제약뿐만 아니라 명령어들 간의 종속성도 처리하여야 한다.

기억장소의 로드 및 스토아 명령어들은 이동에 따른 위험을 없애지 않는 한 본래의 순서를 바꿀 수 없다. 변수와 배열을 살핌으로써, 컴파일러가 다음의 상태 중 하나로 결정할 수 있다:

(1) 연속된 루프 반복에서는 로드 및 스토아는 서로 독립 관계에 있다.
(2) 루프 반복간에 리커런스(Recurrence) 형태의 종속성이 있다. 이러한 형태의 리커런스는 루프 반복 i에서 계산하여 저장한 데이타를 루프 반복 i+1에서 다시 로드하여 사용하는 성질을 가지고 있다. 또한, 리커런스가 배열 요소를 가질 때는 배열간의 정확한 관계를 컴파일 시기에 결정해야 한다.
(3) 위의 (1)과 (2) 어느 쪽에도 해당하지 않는다면, 최악의 상태인, 즉 종속성이 존재하는 리커런스라 할 수 있다. 또한 배열간의 정확한 관계를 결정할 수 없다.

위 (1)의 경우에는, 다른 루프의 반복에 존재하는 로드 및 스토아 명령어들이 서로 재배열 가능하기 때문에, 기억장소의 위험은 없다. 이것은 벡터화 컴파일러의 경우와 비슷하다.

위 (2)의 경우에는, 컴파일러가 리커런스를 검출, 배열 관계를 결정하여 기억장소보다는 레지스터를 통한 연속된 루프 반복간의 데이타 전송을 함으로써 몇 개의 로드 및 스토아 명령어를 제거할 수 있다. 이것은 순서화하기 전의 코드의 변환을 의미한다.

위 (3)의 경우에는, 컴파일러가 로드 및 스토아 명령어 간의 관계를 결정할 수 없기 때문에 로드 및 스토아 명령어를 재배치하거나 펼침 루프에서 제거할 수 없다. 그래서

그러한 루프의 펼침으로 얻을 수 있는 것은 몇 개의 분지와 몇 개의 인덱스를 갖는 명령어들의 제거에 국한될 수 밖에 없다.

2. 소프트웨어 파이프라이닝(Software Pipelining)

소프트웨어 파이프라이닝은 어레이 프로세서(Array Processor, 일명 수평형 아키텍쳐)에서 널리 이용하는 기법이다. 소프트웨어 파이프라이닝의 한 형태를 CDC7600과 CYBER176 컴파일러에서 수 년간 사용하였다.

STACKLIB 루틴의 몇 부분도 소프트웨어 파이프라이닝을 이용하였다. Rau를 비롯한 몇 사람들은 수평형 아키텍쳐에서 효율적인 소프트웨어 파이프라이닝을 수행하는 데 어려움을 설명하고 순서화 작업을 위한 아키텍쳐상의 지원을 제안하고 있다.

VLIW 기계는 근본적으로 트레이스 스케줄링 기법을 구현한 수평형 아키텍쳐이다[14]. 최근에, 몇몇 연구가들은 트레이스 스케줄링의 대신으로 소프트웨어 파이프라이닝 기법을 제안하고 있다[9]. Aiken과 Nicolau는 병렬화 공식을 개발하여 어떠한 변환을 비교하는 데 사용할 수 있음을 보였다. 그 이후, Aiken과 Nicolau는 Doacross 루프를 순서화하는 알고리즘을 제시하였다. Lam은 연속적인 루프의 반복에서 같은 변수가 다른 레지스터에 할당될 수 있는 소프트웨어 파이프라이닝의 확장성을 연구하였다. 이 기법은 루프 펼침과 소프트웨어 파이프라이닝의 복합이라고 할 수 있다.

짧은 클럭 주기로 사이클 당 최대의 명령어 발생을(CRAY-1의 경우 1개)을 내는 시스템에서 순서화 작업의 큰 장애는 기억장소의 거리가 길어지는 것이다. 코드 순서화의 주요 목표는 이렇게 길어진 기억장소와의 거리로 인한 지연된 시간을 줄이는 것이다. 소프트웨어 파이프라이닝에서는 배열을 로드하는 명령어는 그 배열을 필요로 하는 루프반복에 위치 시켜서 그 배열을 사용하는 루프 반복 전에 미리 효과적으로 배치한다. 이러한 개념을 스토아 명령어에도 같이 적용시켜, 저장할 데이타를 산출하는 명령어 다음에 위치시킨다. 따라서 루프를 파이프라이닝하는 방식에는 여러 가지로 구성을 할 수 있다. L은 로드 명령어를, E는 수행명령어를, S는 스토아 명령어를 나타낸다고 할 때, 대표적인 다음 루프를 생각할 수 있다:

```
LOOP : Li
       Ei
       Si
       JC  LOOP
```

JC는 루프의 종결을 판단하는 조건부 분지로 사용된다.

이론적인 파이프라인 순서를 생각할 때는 6 가지의 L, E, S 순서를 만들 수 있지만, S. Weiss와 J. E. Smith의 실험 결과[8, 11, 12]를 따르면, E, S, L과 S, E, L 순서의 소프트웨어 파이프라이닝이 가장 성능이 뛰어나다:

```
      Li                 L1
LOOP  Ei           LOOP: Si-1
      Si                 Ei
      Li+1               Li+1
      JC  LOOP           JC  LOOP
      En                 Sn-1
      Sn                 En
                         Sn
```

3. 제 안

반복 횟수가 적으며(시스템에 따라 다소의 차이가 존재함), 벡터화 및 동시성을 구현할 수 없는 루프에서는 루프 펼침을 통하여 반복시마다 발생하는 루프의 오버헤드를 줄이는 것이 바람직하다. 예를 들어, Alliant FX/2812에서는 컴파일 시간에 루프의 반복 횟수가 알려질 경우 대략 10이하의 루프는 루프 펼침을 권하고 있다[17,19]. Cray-2S에서는 루프의 반복횟수에 따라, 스칼라, 벡터, 그리고 Multitasking을 선택하여 처리 속도를 꾀하고 있다[18]. 물론 이들 두 시스템 모두 사용자 명령어(User Directives) 형식을 이용하여 프로그램 코드에 삽입하여야 한다.

반복 횟수가 크면서, 벡터화 및 동시성을 구현할 수 없는 루프에서는 소프트웨어 파이프라이닝에 의한 스케쥴링 기법을 이용하여, 짧은 클럭 주기로 사이클 당 최대의 명령어 발생율(CRAY-1의 경우 1개)을 내는 시스템에서의 큰 장애인 기어장소의 거리가 길어지는 것을 방지함은 물론 루프 구조로 인한 오버헤드를 최소화 할 수 있을 것이다.

앞으로 이 부분에 관한 시뮬레이션 및 성능 평가를 통하여 시스템 자원(레지스터 및 기억장소의 사용 형태)과의 관계 및 루프의 오버헤드(분지 변수)에 따른 스칼라 컴파일링 기법의 실효성을 연구해야 할 것이다. 또한 Lam이 제시한 연속적인 루프의 반복에서 같은 변수가 다른 레지스터에 할당될 수 있는 소프트웨어 파이프라이닝의 확장성을 이 시스템에서 고려해 볼 수 있을 것이다.

III. 결 론

본 논문은 초고속의 수퍼 파이프라인형 프로세서에서 효율적인 컴파일링을 통한 병렬성을 향상시키기 위한 기법으로 스칼라 컴파일링 기법인 소프트웨어 파이프라이닝(Software Pipelining)과 루프 펼침(Loop Unrolling)기법을 채택하여 성능향상을 꾀하고자 하였다. 다중 명령어 발생 기법의 성질과 구조를 살펴봄으로써, 주어진 프로그램에서의 중요한 오버헤드를 제거할 수 있는 컴파일링 기법을 제안하였다.

IV. 참고 문헌

1. Norman P. Pouppi, "Superscalar vs. Superpipelined Machines", in Computer Architecture News, Vol. 16, No. 3, June 1988, pp. 71-80.
2. Norman P. Pouppi, David W. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines", in Proceedings Third International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS III), April 3-6, 1989, pp. 272-282.
3. NORMAN P. JOUPPI, "The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance", IEEE TRANSACTIONS ON COMPUTERS, VOL. 38, NO. 12, DECEMBER 1989. pp. 1645-1658.
4. Mitsuhisa Sato, Shuichi Ichikawa and Eiichi Goto, "MULTIPLE INSTRUCTION STREAMS IN A PIPELINED PROCESSOR", 1990 Conference on Supercomputers, pp. 182-189.
5. Michael D. Smith, Monica S. Lam, and Mark A. Horowitz, "Boosting Beyond Static Scheduling in a Superscalar Processor", The 17th Annual International Symposium on COMPUTER ARCHITECTURE, May 28-31, 1990.
6. Peter Y. T. Hsu and Edward S. Davidson, "HIGHLY CONCURRENT SCALAR PROCESSING", The 13th Annual International Symposium on COMPUTER ARCHITECTURE, June 2-5, 1986, pp. 386-395.

7. J. E. Smith, "Decoupled Access/Execute Computer Architectures", in ACM Transactions on Computer Systems, Vol. 2, No. 4, November 1984, pp. 289-308.
8. S. Weiss and J. E. Smith, "Instruction Issue Logic in Pipelined Supercomputers", IEEE Trans. Comp., vol. C-33, pp. 1013-1022, Nov. 1984.
9. Steven R. Kunkel and James E. Smith, 'OPTIMAL PIPELINING IN SUPERCOMPUTERS', in the 13th Int. Symp. on Comp. Arch., June, 1986, pp. 404-411.
10. J. E. Smith, G. E. Dermer, B. D. Vanderwarn, S. D. Klinger, C. M. Rozewski, D. L. Fowler, K. R. Scidmore, and J. P. Laudon, "The ZS-1 Central Processor", in Proceedings Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), Oct. 5-8, 1987 pp. 199-204.
11. Shlomo Weiss, James E. Smith, "A STUDY OF SCALAR COMPILATION TECHNIQUES FOR PIPELINED SUPERCOMPUTERS", in Proceedings Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), Oct. 5-8, 1987, pp. 199-204.
12. SHLOMO WEISS and JAMES E. SMITH, "A Study of Scalar Compilation Techniques for Pipelined Supercomputers", ACM Transactions on Mathematical Software, Vol. 16, No. 3, September 1990, pp. 223-245.
13. CONSTANTINE D. POLY-CHRONOULOS, Compiler Optimizations for Enhancing Parallelism and Their Impact on Architecture Design, IEEE TRANSACTIONS ON COMPUTERS, VOL. 37, NO. 8, AUGUST 1988.
14. Robert P. Colwell, Robert P. Nix, John J. O'Donnel, David B. Papworth, Paul K. Rodman, A VLIW Architecture for a Trace Scheduling Compiler, 2nd International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 5-8, 1987.
15. Hwang, K. and Briggs, F. A., Computer Architecture and Parallel Processing, McGraw-Hill, 1984.
16. COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH, JONNL HENNESSY & DAVID PATTER-SON, MORGAN KAUFMANN PUBLISHERS INC. 1990.
17. FX/FORTRAN-2800 Program-mer's Handbook, ALLIANT COMPUTER SYSTEMS CORPORA-TION, JUNE, 1990.
18. 구 자록, "SPICE2G6의 벡터화 및 Multitasking화", UOU Report Vol. 21, No. 1, pp. 65-71, 1990.
19. 구 자록, "Alliant FX/2812 시스템의 병렬처리 구현에 관한 연구", UOU Report Vol. 22, No. 1, 1991.
(to be appeared)