論文
7-3-7

Parafrase Ⅱ를 이용한 병렬 컴파일러 설계
(A Design of Parallel Compiler Using the Parafrase Ⅱ)

송월봉(Worl-Bong Song)[1)]

요 약

본 논문에서는 Parafrase Ⅱ를 이용한 간단한 병렬화 컴파일러를 제안한다. 이것은 중첩루프에서 효율적인 병렬처리를 하기위하여 병렬성을 추출하는 새로운 기법이다. 이를 위해서 Parafrase Ⅱ 병렬화 컴파일러의 원시 프로그램을 분석하였으며 Parafrase Ⅱ를 이용한 새로운 병렬 컴파일러를 구현하였다. 아울러 본 컴파일러는 불변 및 가변 종속 관계를 갖는 모든 형식에 적용이 가능하다.

ABSTRACT

In this paper, a simple parallel compiler using of Parafrase Ⅱis presented. This is a new general method the extracting parallelism in order to parallel processing effectively in nested loop. For this, the source program of Parafrase Ⅱ parallel compiler is analyzed and implemented. Moreover, this method can be applicable where the dependency relation is both uniform and non-uniform in distance.

논문접수 : 2006. 4. 22.
심사완료 : 2006. 5. 19.

1)정회원: 시립인천전문대학 컴퓨터정보과 교수

** 본 논문은 시립인천전문대학의 2005년도 연구지원비에 의한 것임.

1. 서 론

기존의 병렬 컴파일러 중 원시 코드가 공개되었고, 병렬 컴파일러에 대한 기술 축적이 가장 많은 것은 Parafrase Ⅱ[1,7]이다. 따라서 Parafrase Ⅱ 병렬화 컴파일러의 패스별 원시 코드를 분석하고 각 패스에서 다루어져야 할 내용과 패스들과 관련 있는 패스들의 관계를 분석하고자한다.

따라서 Parafrase Ⅱ의 각 패스들 중에서 fixup, callgraph, donest, depend, codegen, flow[8]등에 관심을 갖고 분석하여 새로운 대안을 제시하고자하며, 종속거리가 불변 및 가변 모두의 경우에 컴파일시간에 병렬성 검사를 하고 병렬코드를 생성해내서 실행시간을 단축시킬 수 있는 병렬 컴파일러를 설계한다. 이는 병렬 처리시스템의 실행시간을 단축시킬 수 있는 병렬화 컴파일러를 개발하고 향후 효율적인 병렬화 컴파일러[4,5,7]를 구현하는데 도움이 될 것이다.

2. Parafrase Ⅱ

Parafrase 컴파일러는 실행할 때 -p 옵션을 주면, 패스들이 저장되어 있는 패스 파일을 읽어 실행시킨다. 예를 들어, source.c라는 파일을 병렬화하려 하고, 패스파일이 pass.dat라면 아래와 같이 실행시킨다.

p2fpp -p pass.dat source.c

위의 명령을 실행시, [그림 1]과 같은 순차 프로그램을 [그림 2]와 같은 병렬 프로그램으로 바꾸어 준다. 결과 파일인 병렬 원시 코드는 pass.dat에 있는 codegen 패스의 옵션으로 준다.

[그림 1]에 주어져 있는 문장의 수행순서를 살펴보자. 이 문장들을 병렬 컴퓨터에서 수행시키기 위해서는 첨자 A, B가 사이클을 이루며, 자료 종속성을 갖기 때문에 한꺼번에 수행시킬 수가 없다.

이에 대하여 병렬성을 가장 많이 추출하려고 하는 것이 병렬화 컴파일러가 해야할 일이다.

[그림 1]에서 병렬로 처리할 수 있는 문장의 수를 살펴보자.

OUT(X)을 문장 X에서 좌측값 즉, 결과값이라 하고, IN(X)를 문장 X에서 우측값인 입력값이라 하자. 이때, [그림 1]에서 5번째 문장인 A(i, j) = B(i-3, j-5)를 S5이라 하고, 6번째 문장인 B(i, j) = A(i-2, j-4)를 S6라고 할 때, $OUT(S5)\cap IN(S6)\neq\varnothing$이므로 [그림 1]의 5번째 문장과 6번째 문장은 흐름 종속(flow dependence)을 가진다. <표 1>은 자료종속성[2,3,5]의 종류를 나타낸다.

$OUT(S_1)\cap IN(S_2)\neq\varnothing$	$S_1\ \delta^f\ S_2$	flow dependence
$IN(S_1)\cap OUT(S_2)\neq\varnothing$	$S_1\ \delta^a\ S_2$	anti dependence
$OUT(S_1)\cap OUT(S_2)\neq\varnothing$	$S_1\ \delta^o\ S_2$	output dependence

<표 1> 자료 종속성의 종류
<Table 1> A kind of Data dependence

[그림 1]의 처음 나오는 종속은 20번 반복 후에 생기는 흐름 종속(flow dependence)이며, 20번의 반복 동안 생기는 40개의 문장은 서로 종속성을 가지지 않는다. 따라서 [그림 1]은 한 번에 40개의 문장을 서로 병렬로 처리가 가능하다.

다음으로, Parafrase Ⅱ가 생성한 병렬 코드인 [그림 2]를 살펴보자. [그림 2]에서 Sn을 n번째 문장이라고 할 때, S6의 CDOALL문장은 3~10의 반복구간 동안 문장 S7와 문장 S8를 순차로 처리하므로, 한번에 8개의 문장을 병렬로 처리가 가능하다.

이상에서 볼 수 있듯이, Parafrase Ⅱ에서 생성하는 코드는 최적의 코드가 아니다. 즉, 이러한 점을 개선하여 병렬 컴파일러를 개선할 수 있다.

```
        DIMENSION A(5:20,3:10)
          DIMENSION B(5:20,3:10)
      do 1200 i = 5, 20
         do 1200 j = 3, 10
                        A(i,j) = B(i-3, j-5)
                        B(i,j) = A(i-2, j-4)
1200  continue
      end
```

[그림 1] 순차 프로그램의 예

[Fig. 1] A example of sequential program

```
        IMPLICIT NONE
      REAL a(5:20,3:10)
      REAL b(5:20,3:10)
      INTEGER i, j
         DO 1200 i = 5,20
            CDOALL 1200 j = 3,10
               a(i,j) = b(i - 3,j - 5)
               b(i,j) = a(i - 2,j - 4)
1200     CONTINUE
     END
```

[그림 2] 순차 프로그램의 병렬 코드

[Fig. 2] The parallel code of sequential program

3. Parafrase II 특징

다음으로 Parafrase II의 특징을 살펴보면, Parafrase II는 최초의 병렬 컴파일러인 Parafrase I의 후신이며, 100여 개 이상의 코드 변환 및 병렬화 알고리즘을 제공한다. C나 Fortran[2,6] 프로그램을 입력을 받아 병렬 프로그램으로 출력한다. 또한 각 패스가 독립되어 있어서 사용자가 변환 과정의 순서와 종류를 선택하도록 하고 있다. 이러한 패스의 독립성은 소스 코드를 분석할 때, 분석의 용이성을 기하게 한다.

[그림 3]은 패스가 저장된 파일의 예를 보여준다.

```
fixup
callgraph /dev/null
sumfcn -d0 test.sumfcn
libsum -d0 -imath test.libsum
param_alias test.param_alias
donest -d0 test.donest
depend -d0 test.depend
flow  test.flow
constant test.constant
induction -d000 -v test.induction
dotodoall -d0 test.dotodoall
codegen -d0 test.out
```

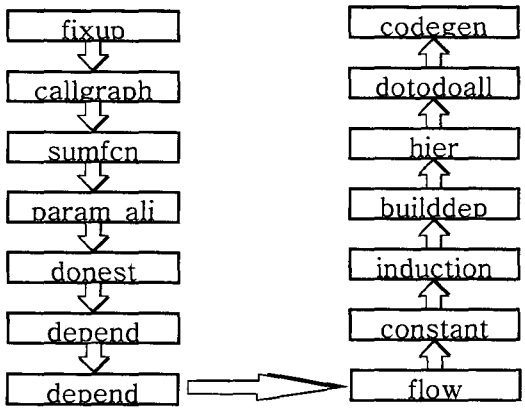
[그림 3] 패스 파일의 예

[Fig. 3] A example of Pass file

4. Parafrase II 분석

본 논문에서는 Parafrase II의 특징 중의 하나인 원시코드가 패스들로 구성되어 있으며, 이런 패스들은 각자 독립성을 지니는 것을 이용하여 패스들을 먼저 분석하고, 해당 패스들이 참조하는 파일들을 파악하고 나서, 각각의 파일들이 참조하는 함수들을 분석하는 방법을 택한다.

[그림 4]는 주요 패스들의 실행 순서를 보여준다.



[그림 4] 주요 패스들의 실행 순서

[Fig. 4] A execution order of major pass

Parafrase Ⅱ는 여러 단계의 패스들을 실행시킴으로써 결과를 출력하게 된다. 패스들은 각각 여러 개의 원시코드 파일들을 포함하고 있다. Parafrase Ⅱ는 36개의 패스들로 구성되어 있으며, 주요 패스들의 각 특징을 살펴보면 다음과 같다.

4.1 fixup

fixup 패스는 구문 분석할 때 작업하기 어려운 것들 즉, 링크드 리스트의 parent pointer, previous pointer같은 정보로 채워져 있다. 이것은 파스 트리 데이터 구조의 완전성을 보장한다. Parent pointer는 statement tree와 expression tree에서 부모를 가리키는 포인터를 지칭하며, previous pointer는 링크드 리스트에서 이전의 원소(element)를 가리키는 포인터을 말한다. 또한 line numbers는 문장의 입력 파일에서의 문장들의 순서와 일치하지 않고, 문장들에서 어휘들의 순서를 반영하여 번호가 정해한다.

fixup 패스는 parafrase Ⅱ에 존재하는 패스들 중에서 반드시 가장 먼저 실행되는 패스이며, 후에 제거된다.

이 패스의 소스가 존재하는 디렉토리는 src/lexers_and_parsers/commom이다.

4.2 callgraph

Callgraph는 각 프로시져간의 호출/피호출 관계를 그래프로 표시한 것으로서 노드는 각 프로시져를 나타내며 노드 사이의 호출표(directed edge)는 프로시져의 호출을 나타낸다.

프로시져간에 함수를 인자로 주고받을 수 없다면 프로그램 상에 나타난 호출문을 그대로 호출 그래프의 화살표로 구성하기만 하면 된다. 그러나 프로시져의 인자(parameter)에 함수값이 바인딩 될 수 있다면 각각의 형식 인자에 바인딩이 가능한 함수값들을 증가시켜 가면서 더 이상 변화가 없을 때까지 반복하는 알고리즘을 수행하여야 한다.

4.3 donest

donest 패스는 각 문장의 중첩 단계(nesting level)를 생성한다. 각 문장에 대하여 T_NEST는 각 문장이 중첩된 DO loop의 목록을 가리키기 위하여 갱신된다. 이 정보는 나중에 loop induction variable을 확인하기 위하여 depend 패스에서 사용된다. 에러 메시지와 출력은 파일로 저장되며, 그 파일의 이름은 각 패스의 argument로 쓰인다.

[그림 1]의 순차 프로그램을 병렬 프로그램으로 바꿀 때, donest 패스에서 나타나는 결과는 [그림 5]와 같다.

```
Donest called on another module
  start is 5
  stop is 20
  step is 1
  start is 3
  stop is 10
  step is 1
```

[그림 5] donest 패스 실행 결과

[Fig. 5] A execution result of donest pass

이 패스가 존재하는 디렉토리는 src/misc/control이다.

4.4 codegen

codegen 패스는 병렬 코드를 파일 또는 표준 출력장치로 출력하는 패스이다. syntax tree에 존재하는 모든 것을 파일 또는 표준 출력장치로 출력한다. 이때, 파일로 출력하고자 하면 옵션을 파일이름을 주면 된다. 즉, 실행시킬 패스들이 저장되어 있는 파일에 'codegen parallel.c'라는 명령으로 parallel.c 옵션을 붙이면 parallel.c의 파일에 병렬 프로그램이 출력된다. 만약, 옵션을 생략할 경우에는 표준 출력장치로 출력을 하게된다.

코드가 Fortran인 경우는 출력할 때 라인 번호를 출력할 수가 있는데, -l 옵션을 붙이면 된다. 그러면 출력할 때, 73-80칼럼에 라인 번호가 추가된다.

이 패스가 존재하는 파일은 src/misc/control이다.

```
Doall called on another module
====   Do loop considered is ====
/* S_LOOP */DO 1200 i = 5,20
/* S_LOOP */DO 1200 j = 3,10
a[i][j] = b[i - 3][j - 5];
b[i][j] = a[i - 2][j - 4];
/* S_F_CONTINUE *//* S_C_CONTINUE */continue;

   1 : To b[i][j] = a[i - 2][j - 4];
         flow Cause a vector  loop is i [ dist is 2
         loop is j [ dist is 4
   2 : To a[i][j] = b[i - 3][j - 5];
         flow Cause b vector  loop is i [ dist is 3
          loop is j [ dist is 5
Loop cannot be made parallel due to dependences 2
1
====   Do loop considered is ====
/* S_LOOP */DO 1200 j = 3,10
a[i][j] = b[i - 3][j - 5];
b[i][j] = a[i - 2][j - 4];
/* S_F_CONTINUE *//* S_C_CONTINUE */continue;

   1 : To b[i][j] = a[i - 2][j - 4];
         flow Cause a vector  loop is i [ dist is 2
         loop is j [ dist is 4
   2 : To a[i][j] = b[i - 3][j - 5];
         flow Cause b vector  loop is i [ dist is 3
       loop is j [ dist is 5
Loop can be made doall
```

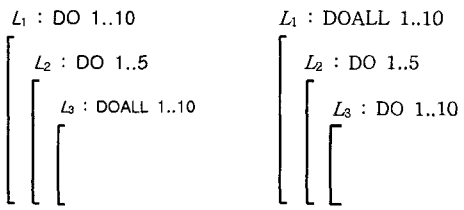
[그림 6] dotodoall 패스 실행 결과

[Fig. 6] A execution result of dotodoall pass

4.5 flow

flow 패스는 프로그램의 flow graph를 생성한다. 만약 -d 옵션을 통해 디버그 플래그가 양수 값을 가지면, flow graph와 그에 상응하는 연결하는 배열과 flow graph의 dominator tree에 관한 정보가 출력된다.

flow 패스와 관련된 주요 스트럭쳐는 두 개가 있다. 이 두 개의 주요 스트럭쳐는 노드의 정보를 담고 있는 flowgraph_t 스트럭쳐와 edge의 정보를 담고 있는 flowedge_t 스트럭쳐이다.



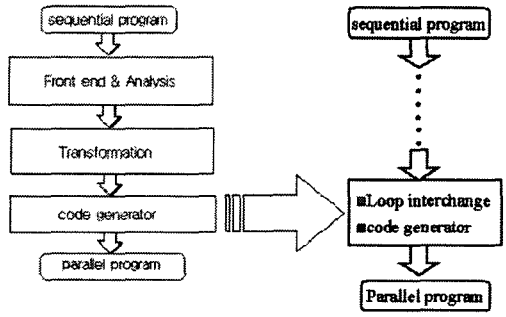
[그림 7] Loop interchange의 예

[Fig. 7] A example of Loop interchange

[그림 7]은 Parafrase II에서 생성된 코드는 좌측처럼 DOALL 문장이 안쪽에 존재하는 것을 반복문 교환 후의 생성된 원시 코드는 우측의 그림처럼 DOALL 문장이 바깥쪽에 존재함을 보여준다.

5. 병렬컴파일러 설계

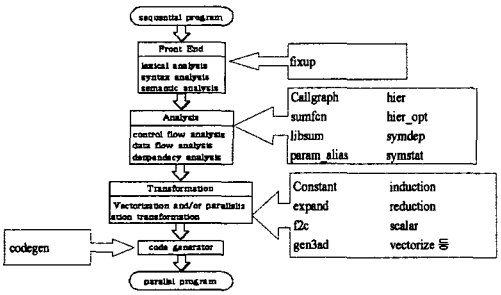
구현하고자 하는 대략적인 구조는 다음 [그림 8]과 같다. 즉, 기존의 code generator에서 loop interchage를 먼저 한 후 코드를 생성하게 하였다.



[그림 8] 병렬 컴파일러의 전체 구성도

[Fig. 8] A organization of parallel compiler

[그림 8]을 이용하여 개선된 병렬 컴파일러의 구조를 살펴보면 [그림 9]와 같다. [그림 9]는 병렬 컴파일러 기본 구성도에 해당되는 패스들을 보여준다. codegen 패스에 interchange를 할 수 있는 모듈이 추가되었다.



[그림 9]개선된 병렬 컴파일러의 구성도

[Fig. 9] A improvement parallel compiler

6. 결 론

본 논문에서는 기존의 Parafrase Ⅱ 병렬 컴파일러를 이용하여 간단한 병렬 컴파일러를 구현하여 보았다. Parafrase Ⅱ의 다른 패스들은 그냥 사용하고, 병렬성을 이용하여 SUIF 병렬 컴파일러에서 사용하는 중첩된 루프에서 병렬 코드를 중첩된 루프의 바깥으로 끌어내는 기법을 추가시켰으며, 이를 위하여 codegen 패스에서 DOALL 문장을 중첩된 루프의 바깥으로 끌어내었다.

이는 병렬 처리시스템의 실행시간을 단축시킬 수 있는 병렬화 컴파일러를 개발하고 향후 효율적인 병렬화 컴파일러를 구현하는데 도움이 될 것이다.

참고문헌

[1] Allen, F., M. Burke, P. Charles, R. Cytron, and J. Ferrante, "An overview of the PTRAN analysis system for multiprocessing," *Journal of parallel and distributed computing*, vol.5, No. 5, Oct. 1988

[2] Allen, J.R. and K. Kennedy, "PFC:A program to convert Fortran to parallel form," Tech. Rept. MASC-TR82-6, Rice University, Houston, Texas, Mar., 1982

[3] Alliant Computer Systems Corp., Alliant FX/Series architecture manual, Acton, MA, Aug., 1986

[4] Ayguade, E., J. Labarta, J. Torres, J. M. Llaberia, and M. Valero, "Parallelism evaluation and partitioning of nested loops for shared memory multiprocessors," Advances in languages and compilers for parallel processing, pp. 220-242, MIT press, 1991

[5] Banerjee, U., Dependence Analysis for super computing, Kluwer Academic Pub., 1988

[6] BBN Advanced Computers Inc., Mach 1000 Fortran compiler references, rev. 1.0 ed, Cambridge, MA, Nov., 1988

[7] Blume, B. et. al., "Polaris : The Next Generation in Parallelizing Compilers," Proceedings of the Workshop on Languages and Compilers for Parallel Computing,

[8] Chen. D.K., and P.C. Yew, "On Effective Execution of Nonuniform DOACROSS Loops," IEEE Trans. on Parallel and Distributed Systems, vol.7, No. 5, pp463-476, 1996

1974년 숭실대 공학사
1982년 한양대 공학석사
1998년 순천향대학교 공학박사(전산학)
1978년 ~ 현재시립인천전문대학 컴퓨터정보과 교수
관심분야 : 병렬처리컴파일러, 알고리즘