論文
3-8-2

루프구조의 병렬화 컴파일러 설계 및 구현
(A Design and Implementation of Parallelizing Compiler in Loop Structure)

송 월 봉*
(Worl-Bong Song)

요 약

본 논문에서는 순차루프를 이용한 간단한 병렬화 컴파일러를 제안한다. 이것은 컴파일 시간에 중첩 병렬 DOALL루프로 바꾸어주는 순차루프의 자동 변환에 관한 절차이다. 이를 위해서, Parafrase II 병렬화 컴파일러의 원시 프로그램을 분석하였으며 중첩루프에서 효율적인 병렬처리를 위한 새로운 병렬성 추출 방법을 구현하였다.

ABSTRACT

In this paper, a simple parallel compiler of a sequential loop is presented. This is a procedure for the automatic conversion of a sequential loop into a nested parallel DOALL loops at compile time. For this, the source program of Parafrase II parallel compiler is analyzed and a new general method the extracting parallelism in order to parallel processing effectively in nested loop is implemented.

1. 서론

지금까지 병렬화 컴파일러의 연구는 메시지 교환 방식의 병렬 프로그램 생성보다는, 공유 메모리 (shared memory) 방식의 병렬 컴퓨터에서 유용한 병렬 루프 생성이나 벡터 연산의 생성을 중심으로 수행되어 왔다. 그 결과 Illinois대학의 Parafrase I, II[6], Rice대학의 PFC[2], IBM의 PTRAN[1]등이 있다.

본 연구에서는 Illinois대학의 Parafrase II 병렬화 컴파일러의 원시 코드를 분석하고, 패스(pass) 별로 수행 순서를 분석하고 이중에 가장 기본이 되고 핵심인 루프에서의 제어흐름분석, 자료흐름분석을 토대로 종속성분석을 수행하고 종속성 그래프를 가지고 병렬 코드로 변환해주는 병렬코드 변환을 수행하는 병렬화 컴파일러를 설계하고 구현하고자 한다.

2. 병렬화 이론

이 장에서는 본 연구에서 다루는 병렬처리 시스템에서의 병렬처리를 다루기 위해 필요한 기법 중 루프의 재구조화 기법을 다루는 데 필요한 기본 개념인 자료 종속성과 병렬화 컴파일러에 대하여 알아본다.

2.1 자료 종속성(Data Dependency)

프로그램의 종속성은 자료(data) 종속성과 제어(control) 종속성으로 구성된다. 제어 종속성은 조건 분기에 의해서 발생되는 종속성으로 자료 종속성과 유사한 방법으로 처리할 수 있기 때문에 본 연구에서는 자료 종속성에 대해서만 알아본다.

* 정회원 : 시립인천전문대학 전자계산과 교수

논문접수 : 2002. 7. 3.
심사완료 : 2002. 7. 29.

※ 본 논문은 시립인천전문대학의 2001년도 연구지원비에 의한 것임.

두 문장 Si와 Sj 사이에서, Si에서 변수 X가 선언(define)되고 Sj에서 X가 사용(use)되며 Si가 Sj 이전에 수행되면 흐름 종속성(SiδSj)이 존재하고 두 문장 Si와 Sj에서, 같은 변수가 선언되고, Si가 Sj 이전에 수행되면 출력 종속성(SiδoSj)이 존재하며 문장 Si에서 X를 사용하고 Sj에서 X가 정의되며 Si가 Sj 이전에 수행되면 반 종속성(SiδaSj)이 존재한다.

이러한 자료 종속성은 병렬성을 추출하고 프로그램 재구조화를 수행할 때 반드시 고려해야 할 중요한 개념이다. 즉, 자료 액세스를 위한 순서를 나타내며 올바른 수행 결과를 얻기 위해 반드시 유지되어야 한다. 효과적이고 정확한 자료 종속성 분석은 루프 구조를 병렬화 하는데 매우 중요하다. 자료 종속성 분석 방법의 기본 원리는 다음과 같다. 먼저 동일한 배열 변수에 대해 두개의 첨자(subscript) 값이 같아지는 루프 변수의 정수 값이 루프 변수의 영역 안에 존재하는 가를 판단한다. 여기서 배열의 차원(dimension)에 따라 연립 방정식을 얻게 되는 데, 이를 종속 방정식(dependence equation)이라 한다. 이 종속 방정식이 루프 변수의 영역 안에서 정수 해를 갖는 다면 종속 관계(dependence relation)가 존재하고 그렇지 않으면 종속 관계가 없는 것이다.

자료 종속성을 분석하는 방법에는 여러 가지 종류가 있다. 그 중에서 가장 간단한 방법은 seperability test[10]이다. 이 방법은 두 문장에서 사용된 동일한 변수에 대해서, 사용된 두개의 첨자 식이 공통적인 루프 변수를 한 개 이하로 표현된 경우에만 적용될 수 있는 방법이다. GCD(Greatest Common Divisor) test[4]는 루프의 영역과는 상관없이 종속 방정식이 정수 해를 가지는 지의 여부를 판단하는 방법이다. 이 외에도 Power test[11], I test[5], λ test[8]방법 등이 자료 종속성 분석을 위해서 사용되기도 한다.

2.2 병렬화 컴파일러

병렬화 컴파일러는 C나 Fortran같은 순차 프로그램을 입력으로 하여 코드 분석 과정을 거쳐 병렬 언어 프로그램이나 메시지 전달 인터페이스(massage passing interface), 병렬 스레드(thread)와 같은 병렬 라이브러리를 호출하는 프로그램을 생성한다. 병렬화 컴파일러는 일반적으로 입력된 순차 프로그램의 어휘 및 구문에 대한 분석들이 전단부에서 점검되며, 이는 기존 컴파일러의 문제와 같다. 전단부를 거친 입력 프로그램들은 중간코드를 출력하며, 출력된 중간코드는 분석부에 입력되어 프로그램의 최적화와 병렬 프로그램의 생성에 필요한 제어 흐름 및 자료 흐름 분석과 종속성 분석(dependence analysis)이 수행된다. 이러한 프로그램 분석은 프로시져 내 분석과 프로시져간 분석으로 구분되며, 주로 프로그램 명령어들 사이에 수행 순서 관계와 프로시져 간의 호출 관계를 분석하는 제어 흐름(control flow)분석, 자료의 정의 및 사용 관계와 프로시져를 호출할 때 자료 전달에 관한 다양한 정보를 분석하는 자료 흐름(data flow)분석 그리고 배열(array) 원소 첨자(index)를 분석하여 루프 내의 배열 요소로 인한 의존성 관계에 대한 분석을 수행한다. 이 결과를 바탕으로 병렬 프로그램 생성을 위한 코드 변환이 수행된다. 코드 변환 작업에서는 주로 벡터 연산 생성(vectorization)과 병렬 루프 생성(parallelization) 등의 다양한 프로시져 변환 작업을 거쳐 목적 기계(target machine)에 적합한 병렬 프로그램을 생성하게 된다. 생성된 병렬 프로그램은 병렬 컴퓨터에서 P개의 프로세서에 의해 동시에 병렬로 스케쥴링이 된다.

이러한 병렬화 컴파일러는 다음과 같은 유용성이 있다.

- 이미 개발되어 사용되는 순차 프로그램을 병렬 컴퓨터에서 신속하게 사용할 수 있다.
- 프로그래머가 병렬 프로그램 대신 순차 프로그램을 작성하도록 함으로서 프로그래머의 부담을 덜 수 있다.
- 병렬 프로그램들은 실행될 병렬 컴퓨터의 구조에 따라 병렬 수행 구조나 고려할 사항이 다르기 때문에 병렬 컴파일러를 사용함으로써 프로그램의 이식성(portability)을 높일 수 있다.

3. 병렬 컴파일러의 분석

본 장에서는 기존의 병렬 컴파일러 중 원시 코드가 공개되었고, 병렬 컴파일러에 대한 기술 축적이 가장 많은 Parafrase Ⅱ를 분석한다.

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 실행 시, [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]의 순차 프로그램에 대한 자료 종속성 및 Parafrase Ⅱ가 생성한 병렬 코드는 [그림 3]과 같다(수행 시 마지막 16개의 문장은 생략).

[그림 1]에서 병렬로 처리할 수 있는 문장의 수를 살펴보자. [그림 3]은 [그림 1]에서 생기는 자료의 종속성을 나타낸 그림으로써 [그림 3]의 첫 문장의 좌측 값인 A(5, 3) 과 21번째 반복에서 생기는 문장의 우측 값인 A(5, 3) 을 살펴보면, 21번째 반복에서 생기는 문장의 A(5, 3)의 값은 첫 번째 문장에서 A(5, 3)의 값이 대치되어야 한다. 그러나 만약 이 두 문장을 병렬로 처리할 때는 21번째 반복에서 생기는 문장이 먼저 수행될 수 있으므로 21번째 반복문에서 생기는 문장의 좌측 값인 B(7, 7)에는 엉뚱한 값이 대입될 수 있다. 그러므로 첫 번째 반복과 21번째 반복에서 생기는 각각의 문장들은 서로 동시에 처리할 수 없다. OUT(X)을 문장 X에서 좌측 값 즉, 결과 값이라 하고, IN(X)를 문장 X에서 우측 값인 입력 값이라 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.

[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 2]는 한 번에 40개의 문장을 서로 병렬로 처리가 가능하다.

다음으로, Parafrase Ⅱ가 생성한 병렬 코드인 [그림 3]을 살펴보자. [그림 3]에서 Sn을 n번째 문장이라고 할 때, S6의 CDOALL문장은 3～10의 반복구간 동안 문장 S7 와 문장 S8를 순차로 처리하므로, 한번에 8개의 문장을 병렬로 처리가 가능하다.

```
      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
```

[그림1] 순차 프로그램의 예

[Fig. 1] The 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] [그림 1] 순차 프로그램의 병렬 코드

[Fig. 2] The parallel code of sequential program

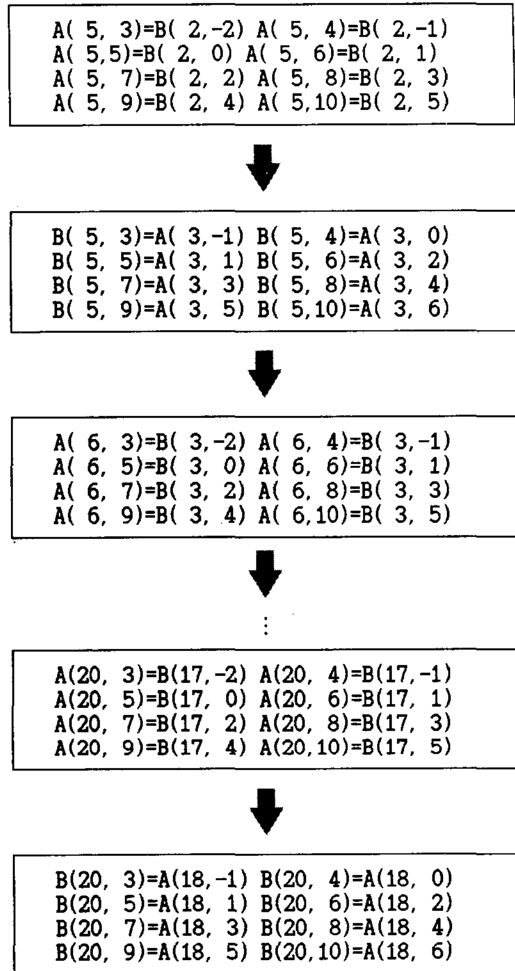
[그림 2]에서는 한 번에 40개의 문장이 병렬 처리가 가능하나, [그림 3]에서 보듯이 ParafraseⅡ에서 생성된 병렬코드는 한 번에 8개의 문장이 한 번에 병렬로 처리가 가능하다.

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

4. 병렬 컴파일러의 설계 및 구현

Parafrase-Ⅱ에서는 기본적으로 병렬화 코드보다 벡터화 코드를 우선시하여 코드를 생성하므로, 중첩된 루프에서 병렬화가 가능한 루프는 기본적으로 안쪽에 위치하여 코드가 생성된다. 만약 Parafrase-Ⅱ를 사용할 때, 병렬 가능한 루프를 바깥쪽에 위치하게 하여 병렬화를 우선시하는 코드를 생성하기 위해서는 loop interchange를 하여야 하며, Parafrase-2에서는 이러한 기능을 유틸리티 형태로 제공된다.

본 장에서는 이러한 기능을 개선하여, 병렬화를 우선시하여 병렬 코드를 생성하는 컴파일러를 구현하였다. 이를 위해서 Parafrase-Ⅱ에서 쓰인 여러 패스들 중, 다른 패스들은 그냥 사용하고 병렬 코드를 생성하는 패스인 codegen 패스에 loop interchange와 연관된 함수를 추가하여 구성하였다. Loop interchange는 초기에 traverse_modules() 함수를 호출한다. 이때 인수로 test_all_for_swap()를 호출하는데, 여기에 연관된 함수는 크게 3가지다. 첫째, ok_to_interchange로써 바깥쪽 반복문(lout)과 안쪽 반복문(lin)이 서로 안전하게 바뀔 수 있는지, 즉 종속성이 없는지를 검사하여 바뀔 수 있으면 TRUE값을 반환하고, 바뀔 수 없으면 FALSE값을 반환한다. 이 함수가 호출하는 함수는 nest_distance, this_nest_level,gen_intrchg_info, free_intrchg이며, 이 함수를 호출하는 함수는 test_all_fro_swap, interchange_ 2_loops이다.



[그림 3] 병렬프로그램의 자료 종속성

[Fig. 3] The data dependence of parallel program

두 번째 함수로는 interchange_2_loop 함수로써 바깥쪽 반복문인 'lout'와 안쪽 반복문인 'lin'을 바꾸는 것을 시도한다. 이 두 반복문은 반드시 완전 중첩 반복문이어야 한다. 만약, 두 반복문사이에 종속성이 없으면 두 반복문은 바뀌고 TRUE값을 반환하고, 그렇지 않으면 FALSE값을 반환한다. 마지막으로 연관된 함수는 do_interchange이며, 이 함수는 종속성을 검사하지 않고, 완전 중첩된 두 반복문을 바꾸어 준다. 이 함수가 호출하는 함수는 in_perfect_nest, rectangular_loop, triangular_loop이며, 이 함수를 호출하는 함수는 interchange_2_loops, test_all_for_swap이다.

```
int call_codegen(p2_module, s)
module_t *p2_module;
char *s;
{
 module_t *modls;
 int lineflag;
 char *getnext();
 int flag;
 char ch;
 char *t;

DB0(5,"BEGIN Loop Interchange\n");
traverse_modules(P2_module,FALSE,NULL,test_all_for_swap);
DB0(5,"END Loop Interchange\n");
 printf("\n");
 if (Opus == F77) {
    printf("GENERATING FORTRAN SOURCE\n");
    Print_IMPLICIT_NONE = 1;
 } else if (Opus == KNRC) {
    printf("GENERATING C SOURCE\n");
 } else {
    yyerror("What language are you using?");
 }
 modls = p2_module;
 lineflag = FALSE;
 getnext(s, &ch, &flag);
 while ((t = getnext(NULL, &ch, &flag)) != NULL){
   if(flag){
     switch(ch){
     case 'l':
       lineflag = TRUE;
       break;
     case 'I':
       if (Opus == F77) Print_IMPLICIT_NONE = atoi(t);
     default:
       break;
     }
   } else {
     break;
   }
 }
 while (modls != NULL) {
    gen_code(p2stdout,modls, lineflag);
    modls = T_NEXT(modls);
 }
 fflush(p2stdout);
 return 0;
}
```

[그림 4] loop interchange가 추가된 codegen 패스의 소스 코드

[Fig. 4] The source code of codegen pass added loop interchange

```
static void test_all_for_swap(f)
function_t    *f;
{
 statement_t *l1, *l2;

 DB1(10,"test_all_for_swap(%x)\n",f);
 fprintf(p2stdout,"testing '%s' for potential loop  interchange\n", S_IDENT(T_FCN(f)));
 for (l1=T_FIRSTDO(f); l1 && T_NEXTDO(l1); ) {
    l2 = T_NEXTDO(l1);
    while (l2 && in_perfect_nest(l1,l2)) {
        if (OK_to_interchange(l1,l2)) {
    fprintf(p2stdout,"swap loops lines %d and %d\n",
                T_LINENO(l1), T_LINENO(l2));
            if (manual && inquire(l1,l2)) {
                if (!do_interchange(l1, l2)) {
                    WARN2("loops %d & %d not swapped",
                T_LINENO(l1), T_LINENO(l2));
                }
            }
        } else {
fprintf(p2stdout,"cannot swap loops lines %d and %d\n",
                T_LINENO(l1), T_LINENO(l2));
        }
        l2 = T_NEXTDO(l2);
    }
    l1 = T_NEXTDO(l1);
 }
}
```

[그림 5] test_all_for_swap() 함수의 원시 코드

[Fig. 5] The source code of test__all__for__swap() function

[그림 4]와 [그림 5]는 loop interchange 시에 주요 코드를 보여준다. [그림 4]에서 traverse_modules()함수를 호출함으로써 loop interchange가 진행된다. 이때 인수로 쓰인 test_all_for_swap()의 원시코드는 [그림 5]와 같다.

[그림 6]은 loop interchange후에 개선되어 나온 병렬코드를 보여준다. 병렬로 처리 가능한 CDOALL 문장이 중첩 루프의 바깥쪽에 위치함을 알 수 있다. 이렇게 병렬 수행이 가능한 루프를 중첩 루프의 바깥쪽에 끌어냄으로써 스케줄링의 용이성과 더 나은 로드 밸런스를 이끌어 낼 수 있다.

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

[그림 6] Loop interchange 후 [그림 2]의 개선된 병렬 코드

[Fig. 6] After the application of Loop interchange, advanced parallel code

5. 결론

본 연구에서는 병렬 컴파일러의 한 종류인 Parafrase Ⅱ를 분석하고 순차루프를 이용한 간단한 병렬화 컴파일러를 제안하였다. Parafrase Ⅱ는 크게 여러 개의 패스로 구성되어 있으며 각각의 패스는 각 파일들을 포함한다. 각 패스의 특성과 그 패스의 파일들이 존재하는 파일과 함수를 분석하였다. 또한 기존의 Parafrase Ⅱ를 갱신하여 간단하게 loop interchange를 자동으로 구현하였다. 이러한 loop interchange는 반복문에서 cedar fortran의 CDOALL문장을 바깥쪽으로 보냄으로서 더 많은 병렬화 효과를 준다.

앞으로 이 알고리즘을 Unimodular 변환과 같은 선형 변환과 혼합하여 더욱 효과적인 실행 결과를 기대하는 문제와 불완전 중첩 루프에의 적용 방법, Benchmark 프로그램에 적용해서 실질적인 문제에 도입하는 문제들을 향후에 실시할 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] Ayguade, E., J. Labarta, J. Torres, J.M. Llaberia, and M. Valero, "Parallelism ealuation and partitioning of nested loops for shared memory multiprocessors," Advances in languages and compilers for parallel processing, pp. 220-242, MIT press, 1991

[4] Banerjee, U., Dependence Analysis for supercomputing, Kluwer Academic Pub., 1988

[5] Ju., J. and V. Chaudhary, "Unique sets oriented partitioning of nested loops with non-uniform dependences," in Proc., Int. Conf. Parallel Processing, Vol III, pp45-52, 1996

[6] Kuck, D.J., E.S. Davidson, D.H. Lawrie, and A.H. Sameh, "Parallel supercomputing Today and the CEDAR approach," *Science*, 231, Feb., 1986

[7] Li, j. and M. Wolfe, "Defining, Analyzing, and Transforming Program Constructs," IEEE Parallel & Distributed Technology, pp32-39, 1994

[8] Lusk, E. and R.A. Overbeek, "Implementation of Monotors with Macro : A programming aid for HEP and other parallel processor," Tech. Rep. ANL-MCS-83-97, Argone National Lab., 1983

[9] Maydan, D.E., J. Hennessy and M.S. Lam, "Effectiveness of Data Dependence Analysis," International Journal of Parallel Programming, Vol. 23, No. 1, pp 63-81, 1995

[10] Tang, P., P.C. Yew, C.Q. Zhu, "Compiler Techniques for Data Synchronization in Nested Parallel Loops," *Proc. of* the ACM *International Conference* on Supercomputing, pp. 176- 181, July, 1990

[11] Tzen, T.H. and L.M. Lionel, "Trapezoid self-scheduling : A practical scheduling for parallel compilers," IEEE Trans. Parallel and Distributed Systems, Vol. 4, No. 1, 1993

송 월 봉

1974년 숭실대 공학사
1982년 한양대 공학석사
1998년 순천향대학교
공학박사(전산학)
1978년～현재
시립인천전문대학
전자계산과 교수
관심분야 : 병렬처리,
컴파일러, 알고리즘