韓國 컴퓨터情報學會 論文誌
第12卷 第5號, 2007. 11.

2007-12-5-1-14

HPF FORALL 구조의 스칼라화(Scalarization)

구 미 순*

Scalarization of HPF FORALL Construct

Misoon Koo*

요 약

스칼라화(Scalarization)는 포트란 90의 array statement나 HPF FORALL 등의 병렬 구조를 동일한 의미의 순차 DO 루프로 변환하는 과정이다. 표준 자료 병렬 언어인 HPF 컴파일러도 HPF로 작성된 프로그램을 메시지 패싱 프리미티브가 삽입된 포트란 77 프로그램으로 변환하고, 병렬 구조인 FORALL을 스칼라화하여 포트란 77의 순차 DO 루프로 변환해야 한다. 본 논문에서는 병렬 구조의 시맨틱을 지닌 다중문장 FORALL 구조를 개선된 성능의 순차 DO 루프로 변환하는 스칼라화 알고리즘을 제안한다. 이를 위해 필요한 종속성 정보를 유지하는 수단으로 관계거리벡터를 정의하여 사용한다. 끝으로 제안된 알고리즘을 적용하여 생성된 코드와 기존 PARADIGM 컴파일러에 의해 생성된 코드의 성능을 비교 평가한다.

Abstract

Scalarization is a process that a parallel construct like an array statement of Fortran 90 or FORALL of HPF is converted into sequential loops that maintain the correct semantics. Most compilers of HPF, recognized as a standard data parallel language, convert a HPF program into a Fortran 77 program inserted message passing primitives. During scalariztion, a parallel construct FORALL should be translated into Fortran 77 DO loops maintaining the semantics of FORALL. In this paper, we propose a scalarization algorithm which converts a FORALL construct into a DO loop with improved performance. For this, we define and use a relation distance vector to keep necessary dependence informations. Then we evaluate execution times of the codes generated by our method and by PARADIGM compiler method for various array sizes.

▸ Keyword : HPF, FORALL 구조, 스칼라화(Scalarization)

• 제1저자 : 구미순
• 접수일 : 2007. 10.30, 심사일 : 2007. 11.12, 심사완료일 : 2007. 11.20.
* 백석문화대학 컴퓨터정보학부 조교수

www.kci.go.kr

Ⅰ. 서 론

분산 메모리 구조의 병렬 처리 시스템에서 프로그래밍의 어려움을 프로그래밍 언어 차원에서 해결하기 위해 제안된 것이 자료 병렬 언어이다. 자료 병렬 언어의 종류는 매우 다양한데, 기존의 순차 프로그래밍 언어에 자료 병렬성을 지원하는 사양이 추가된 형태가 주류를 이룬다. 특히 과학 기술계산 프로그램에 유용하게 사용되던 포트란 언어에 자료 병렬성을 지원하도록 확장된 형태가 가장 일반적이다. 이 가운데 분산 메모리 병렬 시스템에 적합한 자료 병렬 언어의 표준으로 인식되는 언어가 HPF(High Performance Fortran)이다.[1,2,3,4,5]

프로그래머가 작성한 자료 병렬 프로그램을 실제 분산 메모리 컴퓨팅 환경에서 수행 가능하도록 통신 라이브러리를 포함한 병렬 코드로 변환하는 것은 자료 병렬 언어 컴파일러인데, 이들은 대부분 "source-to-source" 형태이다.[1,3,6,7,8,9] 즉, 자료 병렬 언어로 작성된 프로그램을 입력으로 하여 여러 가지 프로그램 분석 후, 분산 메모리 다중 컴퓨터 시스템 구조에서 병렬로 실행되기 위해 데이터 및 연산이 분할되고 프로세서간 통신에 필요한 메시지 패싱 프리미티브가 삽입된 형태의 새로운 소스 프로그램을 생성한다. 이렇게 생성된 새로운 소스 프로그램은 실행될 컴퓨터에 존재하는 포트란 77 컴파일러 등의 순차 컴파일러에 의해 컴파일되고 메시지 패싱 라이브러리와 링크되어 그 컴퓨터 구조에 가장 알맞은 형태의 실행코드가 생성된다.

HPF source-to-source 컴파일러가 HPF로 작성된 프로그램을 병렬 수행 가능한 포트란 77 프로그램으로 변환하기 위해서는 우선 HPF 구문을 포트란 77 구문으로 변환한 뒤, 포트란 77 구문만으로 구성된 프로그램에 대해 병렬화하는 순서로 진행된다. 이처럼 FORALL 구조를 병렬화된 DO 루프로 직접 변환하지 않고 순차 DO 루프로 변환한 뒤 다시 병렬화하는 이유는 병렬화 가능한 순차 DO 루프의 병렬화에 관한 연구가 이미 많이 이루어져 다양한 최적화 방안들이 제시되어 있어 이들을 그대로 적용하기 위함이다.[2,5,8,10,11]

HPF source-to-source 컴파일러의 병렬화 단계의 선행과정으로 HPF의 FORALL을 그 시맨틱이 유지되는 DO 루프로 변환하는 과정을 스칼라화한다고 한다. 이 salarization 과정에서 병렬 FORALL과 순차 DO 루프 사이의 시맨틱 차이로 인해 여러 문제들이 야기된다.

포트란 90의 배열 연산(array operation)이 일반화된 HPF의 FORALL은 단일 문장으로 이루어진 FORALL문(FORALL statement)과 FORALL 구조(FORALL construct)로 나뉜다. FORALL문은 배열 연산과 마찬가지로 LHS(Left-Hand Side)로의 배정이 수행되기 전에 모든 반복에 대해 RHS(Right-Hand Side)가 처리되어야 하는 시맨틱을 갖는다. 다중문장으로 이루어진 FORALL 구조는 FORALL문의 시맨틱이 몸체(body)내 각 문장들에 적용되고, 또 모든 문장에 대해 모든 반복 공간에서 앞 문장의 실행은 뒷 문장의 실행 시작 전에 완료되어야 한다는 시맨틱을 갖는다. 그러므로 동일한 반복 공간에서 몸체내 문장들은 구문적 순서(lexically order)에 따라 순차 실행된다.

컴파일러가 FORALL문을 시맨틱이 유지되는 DO루프로 변환하기 위해 임시배열(temporary arrays)을 도입함으로써 메모리가 추가 사용되는 문제, FORALL구조의 시맨틱을 유지하기 위해 몸체 내 문장들을 다수의 루프로 변환함으로써 루프 오버헤드가 증가되고 그로 인해 프로그램 성능이 저하되는 문제 등이 그것이다. 포트란 90의 배열 연산이나 HPF의 단일문장 FORALL문의 스칼라화 과정에서 발생되는 문제들을 개선하기 위한 몇몇 연구들은 있었으나, 다중문장 FORALL구조의 스칼라화에 관한 연구는 많지 않다.[12,13,14]

본 논문에서는 FORALL 구조를 스칼라화하는 과정에 적용함으로써 개선된 성능의 DO루프로 변환할 수 있는 스칼라화 알고리즘을 제안한다. 기존 컴파일러 최적화 방법 중 루프 반전(loop reversal), 루프 교환(loop interchange), 루프 정렬(loop alignment) 등을 활용한 이 스칼라화 알고리즘을 적용하여 FORALL구조의 시맨틱은 유지하면서도 최소의 임시기억장소 사용, 적은 수의 DO루프 생성 등으로 향상된 성능 결과를 얻을 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 연구와 그들의 문제점에 대해 기술한다. 3장에서는 본 논문에서 사용되는 필요 정보 유지 수단으로 관계거리벡터를 정의하고, 이를 기반으로 스칼라화 알고리즘을 제시한다. 4장에서는 기존 연구 중 PARADIGM 컴파일러의 스칼라화 방법에 의해 생성된 코드와 본 논문에서 제안한 스칼라화 알고리즘에 의해 생성된 코드의 성능 평가 결과에 대해 논한다.

Ⅱ. 기존 연구

2.1 PARADIGM 컴파일러

PARADIGM에서는 단일문장 FORALL문 변환시 복제

www.kci.go.kr

(clone)방법이, 다중문장 FORALL구조 변환시 루프 분할 방법이 사용된다.[7]

복제 방법은 변환된 DO 루프에서 FORALL문의 시맨틱 유지를 보장할 수 없을 경우 임시 배열을 생성하여 DO 루프 이전에 원래 배열을 임시 배열로 복사하여 사용하는 것이다. <그림 2-1>은 복제 방법을 이용한 예이다.

```
FORALL(I=1:100, X(I).EQ.0) X(I) = X(I-1) + X(I+1)
```
(a) FORALL문

```
CALL MEM_COPY(X_CLONE, X, 100)
DO I=1, 100
   IF (X_CLONE(I) .EQ. 0)
      X(I) = X_CLONE(I-1) + X_CLONE(I+1)
   END IF
END DO
```
(b) 변환된 DO 루프

그림 2-1 복제방법에 의해 변환된 예
Fig.2-1 A conversion example using clone method

다중 문장 FORALL 구조를 중첩 DO 루프로 변환할 때, 반복 공간에 관계없이 선행 문장의 수행 결과가 후행 문장에 반영되어야 하는 FORALL구조의 시맨틱을 유지하기 위해 <그림 2-2>와 같이 루프 분할 방법을 적용하여 변환한다.

```
FORALL(J=2:99, I=2:99, A(I,J) .NE. B(I,J))
  C(I,J) = SQRT(A(I,J)*A(J,I))
  B(I,J) = (C(I+1,J+1)+C(I-1,J-1))/(B(I,J)-A(I,J))
END FORALL
```
(a) FORALL구조

```
DO J=2, 99
  DO I=2,99
     IF (A(I,J) .NE. B(I,J)) THEN
       C(I,J) = SQRT(A(I,J)*A(J,I))
     END IF
  END DO
END DO
DO J=2, 99
  DO I=2,99
     IF (A(I,J) .NE. B(I,J)) THEN
       B(I,J)= (C(I+1,J+1)+C(I-1,J-1))/(B(I,J)-A(I,J))
     END IF
  END DO
END DO
```
(b) 변환된 DO 루프

그림 2-2 루프분할에 의해 변환된 예
Fig.2-2 A conversion example using loop distribution

PARADIGM 컴파일러에서 사용되는 복제방법은 FORALL 문의 시맨틱 유지를 위해 도입된 임시 배열로 인해 메모리가 추가로 요구되는 문제가 있다. 다음으로 루프 분할을 적용하여 FORALL구조를 변환하였는데, FORALL구조내 문장 수가 증가할수록 다수의 DO 루프가 생성되어 루프 오버헤드가 증가하여 프로그램의 성능을 저하시키는 문제가 있다.

2.2 포트란 90D/HPF 컴파일러

포트란 90D/HPF 컴파일러에서는 FORALL을 DO루프로 변환할 때 임시 기억장소 이용, 루프 교환, 마스크 삽입 기법 등이 사용된다.[1]

임시 기억장소 이용 방법은 PARADIGM 컴파일러의 복제 방법과 동일하므로 생략한다. 다음으로 루프 교환은 FORALL문이 중첩 DO 루프로 변환될 때, 중첩 루프의 두 레벨을 맞바꾸는 것이다. 포트란은 열 우선 순서(column-major order)로 배열을 저장하므로, 포트란 90D/HPF 컴파일러는 <그림 2-3>과 같이 FORALL triplet을 열 우선으로 정렬한다.

```
FORALL(I=1:N, J=1:N) A(I,J) = B(I,J)
```
(a) FORALL문

```
DO J=1, N
   DO I=1, N
      A(I,J) = B(I,J)
   END DO
END DO
```
(b) 변환된 DO 루프

그림 2-3 루프교환에 의해 변환된 예
Fig.2-3 A conversion example using loop interchange

마지막으로 마스크 삽입 기법은 <그림 2-4> (a)와 같이 Gaussian Elimination 코드에 마스크가 존재하는 FORALL문에서 (b)와 같이 연관 있는 인덱스에 대해서만 마스크가 적용되도록 변환하는 방법이다.

포트란 90D/HPF 컴파일러에서 사용된 변환 방법들 중 루프 교환은 포트란 컴파일러에서 대부분 적용되고 있는 일반적 방법이고, 마스크 삽입 기법은 극히 제한적으로만 적용 가능한 소극적인 최적화 방법이다.

```
FORALL(I=1:N, J=1:NN, INDX(I) .EQ. -1)
  A(I,J) = A(I,J) - FAC(I) * ROW(J)
```
(a) FORALL문

```
DO I=1, N
   IF (INDX(I) .EQ. -1) THEN
```

www.kci.go.kr

```
      DO J=1, NN
         A(I,J) = A(I,J) - FAC(I) * ROW(J)
      END DO
    END IF
  END DO
```

(b) 변환된 DO 루프

그림 2-4 마스크삽입기법에 의해 변환된 예
Fig.2-4 A conversion example using mask insertion

2.3 배열 연산의 스칼라화

포트란 90의 병렬 구조인 배열 연산의 스칼라화에 관한 몇몇 연구들이 있다.[12,13,15]

2.3.1 One-pass 스칼라화

기존 대부분의 포트란 90 컴파일러들은 각 배열 연산에 대해 다음 두 단계로 스칼라화를 수행한다.

1) 그대로 스칼라화(naive scalarization)
2) 변환된 코드의 자료 종속성에 따라 루프 반전, 루프 교환, 입력 선출, 임시배열 도입 등을 적용하여 코드 변환

G. Roth는 위와 같이 두 단계로 스칼라화할 경우, 각 배열 연산 별로 스칼라화한 결과 단일 문장으로 이루어진 다수의 루프들이 생겨나 루프 오버헤드도 커지고 자료 재사용성(reuse)도 떨어지는 문제가 있음을 제기하였다. 그래서 모든 배열 연산들의 첨자에 대해 종속성 분석을 실시하여 임시배열 도입 없이 스칼라화 가능한 문장들을 묶어 하나의 루프로 변환하는 one-pass 스칼라화방법을 제안하였다.[12]

기존 스칼라화 방법에 의해 발생되는 문제들을 해결하기 위해 제안된 이 방법은 여러 배열 연산의 triplet형태의 첨자가 일치되는가에 따라 종속성 여부를 판단하고, 어떤 종속성도 존재하지 않는 경우에만 적용할 수 있는 한정적인 방법이다.

2.3.2 루프 정렬을 이용한 스칼라화

Y. Zhao는 기존 포트란 90 컴파일러들에서 배열 연산 스칼라화시 시맨틱을 유지하기 위해 임시배열을 도입해야 하는 경우에 대해 종속성 정보에 따라 루프 정렬을 적용함으로써 최소 크기의 임시배열만을 도입하여 변환할 수 있는 방안을 제안하였다.[13,15]

〈그림 2-5〉(a)는 시맨틱 유지를 위해 임시배열을 도입해야 하는 배열 연산 예이다. 이를 기존 방법에 의해 스칼라화한 것이 (b)이고, Y. Zhao가 제안한 루프 정렬을 적용시켜 스칼라화한 것이 (c)이다. 그림에서 보는 바와 같이 기존 방법에서는 배열 크기만큼의 임시배열이 필요한데 비해, 루프 정렬을 적용하면 최소 종속 거리에 해당하는 크기만큼의 임시배열만으로 시맨틱이 유지되는 스칼라화가 가능하다.

그러나 이 방안은 하나의 배열 연산만을 대상으로 한다. 여러 배열 연산이 인접해 있는 경우와 유사한 다중 문장 FORALL구조의 스칼라화에는 직접 적용하기 어렵다.

```
A(2:11) = A(1:10) + A(3:12)
```

(a) F90 배열 연산

```
ALLOCATE T(10)
DO I=1, 10
   T(I) = A(I) + A(I+2)
END DO
DO I=1, 10
   A(I+1) = T(I)
END DO
DEALLOCATE T
```

(b) 기존 방법에 의해 변환된 DO 루프

```
ALLOCATE T(2)
T(1) = A(1) + A(3)
DO I=1, 9
  T(2) = A(I+1) + A(I+3)
   A(I+1) = T(1)
   T(1) = T(2)
END DO
A(11) = T(1)
DEALLOCATE T
```

(c) 루프 정렬에 의해 변환된 DO 루프

그림 2-5 배열 연산의 스칼라화 예
Fig.2-5 A scalarization example of array operation

III. 제안 방안

HPF 컴파일러가 다중 문장 FORALL 구조를 스칼라화하여 생성된 DO 루프에서 FORALL 구조의 시맨틱이 그대로 유지되려면, 앞에서 언급한 바와 같이 선행 문장에서 후행 문장으로의 전향 종속성(Forward Data Dependence: 이하 FDD라 함)만이 발생해야 한다. 이러한 이유로 기존의 HPF 컴파일러들은 하나의 FORALL 구조를 다수의 DO 루프로 변환하고 있다.

본 논문에서는 FORALL 구조내 문장 간 종속성에 의해 스칼라화된 DO 루프에서 역향 종속성(Backward Data Dependence: 이하 BDD라 함)이 발생되는 경우에 대해 FORALL 구조의 시맨틱이 유지되면서도 가능한 한 임시배열이 도입되지 않고 적은 수의 루프가 생성되는 알고리즘을 제안하고자 한다. 이를 위해 필요한 데이터 종속성 정보와

www.kci.go.kr

흐름 정보를 모두 표현하는 수단으로 관계거리벡터를 정의하고, 이 관계거리벡터 정보를 기반으로 새로운 스칼라화 알고리즘을 설명한다.

3.1 관계거리벡터

데이터 종속성을 표현하는 수단으로 방향벡터(direction vector)와 거리벡터(distance vector)가 많이 사용된다.[9,11] n-단계 중첩 루프 인덱스에 대해 반복 a내의 어떤 문장으로부터 반복 b내의 문장으로 데이터 종속성이 존재할 때 같은 인덱스에 대한 반복 값의 차이를 거리벡터라 한다. 거리벡터 값이 양이면 방향벡터를 '+' 또는 '<'로, 음이면 '-' 또는 '>'로, 0이면 0 또는 '='로 표현한다. 이와 같이 거리벡터와 방향벡터는 종속성이 존재하는 두 문장 사이에서 '종속하는 문장에서 종속되는 문장으로의 종속방향 관점에서 같은 루프 인덱스 값의 차이'를 표현한다.

프로그램 코드 상에서 종속 관계에 있는 두 문장을 그 순서에 따라 먼저 나오는 문장을 선행 문장(predecessor statement), 뒤에 나오는 문장을 후속 문장(successor statement)라 하자. 방향벡터와 거리벡터로는 종속성이 존재하는 두 문장 중 어느 문장이 선행 문장이고, 어느 문장이 후속 문장인지 알 수 없다.

한편, 관계벡터(relation vector)는 종속성이 존재하는 두 문장 사이에서 '선행문장에서 후속문장으로의 종속 방향 관점에서 대응되는 루프 인덱스 값의 차이' 관계를 '>', '<', '='로 표현한다.[10] 이를 통해 종속 관계에 있는 문장들의 종속성 정보와 흐름 정보는 나타낼 수 있으나, 종속거리 정보는 얻을 수 없다.

그래서 본 논문에서는 프로그램 문장들 사이의 종속거리를 포함한 데이터 종속성 정보와 문장간의 실행 순서인 흐름 정보를 모두 나타낼 수 있도록 관계벡터를 수정한 관계거리벡터(relation distance vector)를 정의하여 사용한다. 관련 용어는 다음과 같이 정의한다.

S_i : FORALL 구조내 i번째 문장
l_{ik} : S_i의 최상위 루프로부터 k번째 루프 인덱스
rd_k : 선행문장 S_i와 후속문장 S_j 사이의 k번째 루프 인덱스 값의 차이(양수, 음수, 0로 표현)

rd_k는 인덱스 값의 차이 관계에 따라 다음 세 경우로 나뉜다.(단, C는 임의의 양의 정수)

(1) rd_k = '+C' : $l_{ik} = m$, $l_{jk} = m - C$이고, '$l_{ik} - l_{jk}$'의 결과가 양수
(2) rd_k = '-C' : $l_{ik} = m$, $l_{jk} = m + C$이고, '$l_{ik} - l_{jk}$'의 결과가 음수
(3) rd_k = '0' : $l_{ik} = l_{jk} = m$이고, '$l_{ik} - l_{jk}$'의 결과가 0

n-단계의 중첩 루프에서, 두 문장 S_i와 S_j 사이에 종속관계가 존재할 때, n개의 루프 인덱스 값들의 차이 관계 집합을 관계거리벡터 RD_{ij}라고 하고, 다음과 같이 나타낸다.

$$RD_{ij} = (rd_1,\ldots,rd_k,\ldots,rd_n), \quad 1 \le k \le n$$

여기서 n은 루프 인덱스의 최대값이고, 인덱스 값의 차이관계를 나타내는 rd_k는 두 문장이 속하는 k번째 루프 인덱스의 관계거리벡터 항목이 된다.

관계거리벡터는 종속성이 존재하는 두 문장 사이에서 '선행문장에서 후속문장으로의 종속방향 관점에서 루프 인덱스 값의 차이'를 표현한다. 따라서 종속성 정보와 제어흐름 정보를 동시에 나타낼 수 있다. 즉, 두 문장 사이의 관계를 관계거리벡터로 유지함으로써 다양한 데이터 종속성의 내용 및 기존의 방향 벡터나 거리벡터로는 나타낼 수 없었던 제어 흐름상의 선후관계(precedence relation)도 표현할 수 있다.

단일 항목을 갖는 관계거리벡터에 대하여 그 값이 양수이면 전향 종속성을, 음수이면 역향 종속성을 나타낸다. 이 관계를 정리하면 다음과 같다.

(1) RD_{ij} = +C이면, S_j가 S_i에 전향 종속(FDD), LCD된다.
(2) RD_{ij} = -C이면, S_i가 S_j에 역향 종속(BDD), LCD된다.
(3) RD_{ij} = 0이면, S_j가 S_i에 전향 종속(FDD), LID된다.

여기서 LCD(Loop Carried Dependence)는 서로 다른 두 반복 내에서, LCD(Loop Independent Dependence)는 동일한 반복 내에서 두 문장 간 데이터 종속관계가 있음을 의미한다.

다중 문장 FORALL 구조 내 문장들에서 참조하는 여러 변수 간에 종속관계가 있을 때, 이들 종속관계를 나타내는 관계거리벡터의 항목 중 음수 값이 적어도 하나 존재하면 스칼라화 후 변환된 루프 내 문장들 사이에 역향 종속성이 발생함을 의미한다.

www.kci.go.kr

3.2 스칼라화 알고리즘

본 논문에서는 선행 연구 논문[16]에서 제안한 최적화기를 좀 더 체계적으로 개선한 스칼라화 알고리즘을 제안한다. [16]에서는 다중 문장 FORALL 구조내 문장들에서 종속성 검사를 통해 구해진 모든 관계거리벡터를 기반으로 다음과 같이 스칼라화하는 최적화기를 제안하였다.

(1) 관계거리벡터 항목 값들 중 음수 항목 값이 없으면, 어떤 역향 종속성도 발생하지 않으므로 FORALL 구조를 그대로 DO 루프로 변환한다.

(2) 관계거리벡터 항목 값들 중 한 개 이상의 음수 항목 값이 존재하면 역향 종속성이 발생하므로

① 음수 항목 값들 중 최소값의 절대값(최대종속거리)을 구한다.

② FORALL 구조내 첫 문장을 기준으로 나머지 문장들을 모든 루프 인덱스에 대해 최대 종속 거리 만큼 루프 정렬을 적용하여 DO 루프로 변환한다.

이 최적화기는 스칼라화시 발생하는 역향 종속성을 없애기 위해 루프 인덱스 구분 없이 최대 종속 거리만큼 루프 정렬을 적용한다. 역향 종속성을 일으키는 루프 인덱스가 아니어도 다른 루프 인덱스에 대한 관계거리벡터 값 중 음수 항목 값이 있으면 모든 루프 인덱스에 대해 루프 정렬이 적용된 DO 루프로 변환하게 된다. 즉, 루프 정렬이 필요치 않은 루프 인덱스에 대해서도 동일하게 루프 정렬이 적용됨으로써 보다 복잡한 형태의 DO 루프가 생성된다. HPF 컴파일러에서 스칼라화 후 병렬화 단계가 진행되는 것을 감안할 때, 불필요하게 루프 정렬이 적용된 코드는 병렬화에 방해가 될 수 있다.

이러한 문제점을 해결하고 보다 최적화된 코드로 변환하기 위해 FORALL 구조에서 발생 가능한 관계거리벡터 경우의 수를 아래와 같이 세분화하고, 각 경우에 보다 적절한 루프 변환을 적용한다.

(1) 모든 루프 인덱스에 대해 관계거리벡터 항목 값이 0과 양수

(2) 모든 루프 인덱스에 대해 관계거리벡터 항목 값이 0과 음수

(3) 한 루프 인덱스에 대해서만 관계거리벡터 항목 값이 0과 양수

(4) 모든 루프 인덱스에 대해 관계거리벡터의 항목 값이 0, 양수, 음수

(1)의 경우, 스칼라화를 방해하는 어떤 역향 종속성도 발생하지 않으므로 그대로 DO 루프로 변환한다. (2)의 경우, 모든 루프 인덱스에 대해 전향 종속성은 발생하지 않고 역향 종속성만 발생하므로 이를 제거하기 위해 루프 반전을 적용한다. (3)의 경우, 전향 종속성만 발생하는 특정 인덱스에 대한 루프를 최외곽 루프로 배치하는 루프 교환을 적용함으로써 다른 루프 인덱스들에서 발생되는 역향 종속성을 없앤다. 끝으로 (4)의 경우, 모든 루프 인덱스에 대해 전향 종속성과 역향 종속성이 함께 발생하므로 루프 정렬을 적용한다.

〈그림 3-1〉, 〈그림 3-2〉, 〈그림 3-3〉은 〈그림 2-2〉 (a)코드에 대해 복잡성을 덜기 위해 마스크를 생략하고 위에서 분류한 (2)~(4) 경우에 해당하는 종속성을 갖도록 수정된 코드를 각각 다른 방법으로 스칼라화한 예이다.

```
FORALL(J=1:98, I=1:98)
  C(I,J) = SQRT(A(I,J)*A(J,I))
  B(I,J) = (C(I+1,J+1)+C(I+2,J+2))/(B(I,J)-A(I,J))
END FORALL
```

(a) (2)의 경우에 해당하는 FORALL구조

```
DO J=98, 1, -1
  DO I=98, 1, -1
    C(I,J)= SQRT(A(I,J)*A(J,I))
    B(I,J)= (C(I+1,J+1)+C(I+2,J+2))/(B(I,J)-A(I,J))
  END DO
END DO
```

(b) 변환된 DO 루프

그림 3-1 루프반전에 의해 변환된 예
Fig.3-1 A conversion example using loop reversal

```
FORALL(J=2:99, I=2:99)
  C(I,J) = SQRT(A(I,J)*A(J,I))
  B(I,J) = (C(I-1,J+1)+C(I-1,J-1))/(B(I,J)-A(I,J))
END FORALL
```

(a) (3)의 경우에 해당하는 FORALL구조

```
DO I=2, 99
  DO J=2, 99
    C(I,J)= SQRT(A(I,J)*A(J,I))
    B(I,J)= (C(I-1,J+1)+C(I-1,J-1))/(B(I,J)-A(I,J))
  END DO
END DO
```

(b) 변환된 DO 루프

그림 3-2 루프교환에 의해 변환된 예
Fig.3-2 A conversion example using loop interchange

```
FORALL(J=2:99, I=2:99)
  C(I,J) = SQRT(A(I,J)*A(J,I))
  B(I,J) = (C(I+1,J+1)+C(I-1,J-1))/(B(I,J)-A(I,J))
END FORALL
```

www.kci.go.kr

(a) (4)의 경우에 해당하는 FORALL구조

```
DO I=2, 99
  C(I,2)=SQRT(A(I,2)*A(2,I))
END DO
DO I=3, 99
  C(2,J)=SQRT(A(2,J)*A(J,2))
  DO J=3, 99
     C(I,J)= SQRT(A(I,J)*A(J,I))
     B(I-1,J-1)=
              (C(I,J)+C(I-2,J-2))/(B(I-1,J-1)-A(I-1,J-1))
  END DO
  B(99,I) = (C(100,J+1)+C(98,J-1))/(B(99,J)-A(99,J))
END DO
DO I=2, 99
  B(I,99) = (C(I+1,100)+C(I-1,98))/(B(I,99)-A(I,99))
END DO
```

(b) 변환된 DO 루프

그림 3-3 루프정렬에 의해 변환된 예
Fig.3-3 A conversion example using loop alignment

〈그림 3-4〉는 본 논문에서 제안하는 FORALL 구조의 스칼라화 알고리즘이다.

```
Algorithm : Scalarize_FORALL()
begin
/* Initialize */
Positive_i = 0                                  // (1≤i≤n)
Negative_i = 0
Max_distance_i = 0

/* Check any negative entry
   for relation distance vectors of each loop index */
for each l_i,
  check C_i^1⋯,C_i^r,⋯,C_i^d
     if there is any positive value then
       Positive_i = 1
     else if there is any negative value then
          Negative_i = 1
          if (Max_distance 〉 C_i^r) then
             Max_distance = C_i^r

if all of Negative_i are zero then              // Case (1)
   naively scalarize
else if all of Positive_i are zero then         // Case (2)
       apply loop reversal
else if any Positive_i is 1 then                // Case (3)
       apply loop interchange
else                                            // Case (4)
       apply loop alignment to each loop
                              with abs(Max_distance_i)
end if
end
```

그림 3-4 스칼라화 알고리즘
Fig.3-4 Scalarization Algorithm

〈그림 3-4〉에서 사용된 용어들은 다음과 같다.

$l_1,\cdots,l_n$: 루프 인덱스의 반복 공간(iteration space)
$S_1,\cdots S_m$: FORALL 구조내 문장
RD : $S_1,\cdots S_m$ 사이에 존재하는 d가지 종속관계에 대한 관계거리벡터 집합
$= \{(C_1^1,\cdots,C_n^1),\cdots,(C_1^r,\cdots,C_n^r),\cdots,(C_1^d,\cdots,C_n^d)\}$, where $(1\le r\le d)$
abs() : 절대값을 구하는 함수

IV. 성능 평가

이 장에서는 본 논문에서 제안한 스칼라화 알고리즘에 의해 생성된 코드와 PARADIGM 컴파일러에 의해 생성된 코드에 대한 비교 성능 평가를 행한다. 4.1절에서 성능 평가 대상 코드와 성능 평가 방법에 대해 기술하고, 4.2절에서 성능 평가 결과에 대해 논한다.

4.1 성능 평가 방법

성능 평가를 위해서는 다중 문장 FORALL 구조이면서 몸체 부분의 문장들 간에 다양한 종속성이 발생하는 코드를 포함한 벤치마크 프로그램이 필요하다. 그러나 survey결과 이에 맞는 적절한 벤치마크가 존재하지 않는 것으로 판단되어 PARADIGM 컴파일러에서 스칼라화 예로 제시한 〈그림 2-2〉 (a) 코드를 성능평가에 사용하였다. 〈그림 2-2〉 (a)에서 복잡성을 덜기 위해 마스크를 생략하고 본 논문에서 분류한 (2)~(4) 경우에 해당하는 종속성을 갖도록 코드를 각각 수정한다. 이렇게 만들어진 FORALL 구조에 대해 PARADIGM 컴파일러 방법으로 변환된 코드와 본 논문의 제안 알고리즘에 따라 생성된 코드 〈그림 3-1〉, 〈그림 3-2〉, 〈그림 3-3〉의 실행 시간을 여러 배열 크기에 대해 측정하였다.

실험 환경은 윈도우 XP가 설치된 펜티엄 4 PC(CPU 2,8GHz)에 Innotek VirtualBox로 가상 머신(기본메모리 512MB, 하드디스크 6GB)을 구축하고 운영체제로 리눅스(우분투 7.04)를 설치하였다. 변환된 코드의 컴파일은 g77 포트란 컴파일러, 실행시간 측정 라이브러리는 dtime()을 사용하였다.

4.2 성능 평가 결과 및 분석

본 논문에서 제안한 스칼라화 알고리즘에서 (2)~(4) 경우의 생성된 코드에 대해 배열 크기를 100×100, 256×256, 512×512 까지 변화시켜가며 4.1절에서 기술한 방법에 의해 실행시간을 측정한 결과가 〈그림 4-1〉이다. 스칼라화 알고리

www.kci.go.kr

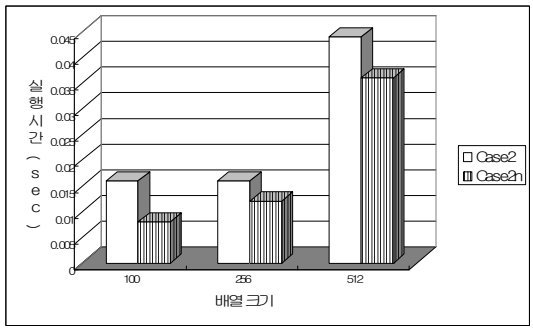
즘에서 분류한 (2)~(4) 경우에 해당하는 각각의 코드에 대해 기존 PARADIGM 컴파일러의 루프 분할방법을 적용하여 변환된 코드의 실행시간 측정결과가 〈그림 4-1〉의 Case2, Case3, Case4이다. 그리고 (2)~(4) 경우에 해당하는 각각의 코드에 대해 본 논문에서 제안한 루프 반전, 루프 교환, 루프 정렬을 각각 적용하여 변환된 코드의 실행시간 측정결과가 Case2n, Case3n, Case4n이다.

〈그림 4-1〉에서 보는 바와 같이 본 논문에서 제안한 방법으로 스칼라화된 코드의 실행시간(Case2n, Case3n, Case4n)이 PARADIGM 컴파일러 방법으로 스칼라화된 코드의 실행시간(Case2, Case3, Case4)에 비해 모든 배열 크기에 대해 단축되었다. 이 같은 결과는 루프 오버헤드와 자료 재사용성에 기인한다고 할 수 있다. 즉, PARADIGM 컴파일러는 FORALL 구조 스칼라화에 루프 분할을 적용하여 별개의 루프들을 생성하므로, 단일 루프에 비해 루프 오버헤드가 증가하게 된다. 루프 내에서 참조되는 배열의 크기가 커질수록 각 루프의 반복 횟수도 증가되므로, 루프 수가 많아지면 프로그램 실행시간은 증가된다. 또한, FORALL 구조를 별개의 루프들로 변환하면 한 개의 루프 내에 있을 때 가능했던 자료 재사용 기회가 줄어들고 그로 인해 캐쉬 효율이 떨어져 성능이 저하된다. 이에 반해 본 논문에서 제안한 스칼라화 알고리즘에 따라 변환된 코드들은 (2)~(4) 경우 각각 적절한 루프 변환 방법들을 다르게 적용하여 모두 단일 루프를 생성하므로 루프 오버헤드가 줄고 자료 재사용 기회가 높아져 실행시간면에서 비교우위를 보인 것이다.

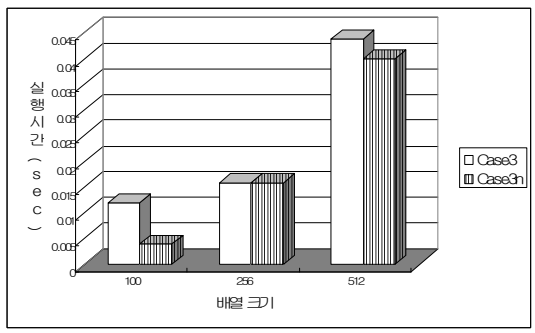
V. 결 론

자료 병렬 언어의 표준인 HPF 컴파일러들은 대부분 source-to-source형태이다. 즉, HPF source-to-source 컴파일러는 HPF 구문으로 작성된 프로그램을 포트란 77 구문으로 변환한 뒤, 포트란 77 구문만으로 구성된 프로그램에 대해 병렬화하는 순서로 진행된다. 그런데 병렬 수행 구조인 FORALL 구조를 순차 DO 루프로 변환하는 스칼라화 과정에서 단순히 FORALL 구조의 시맨틱을 유지하도록 변환하다 보니 임시배열 도입으로 인한 캐쉬 성능 저하, 다수의 루프 생성으로 인한 실행시간 증가 등의 문제들이 발생한다.

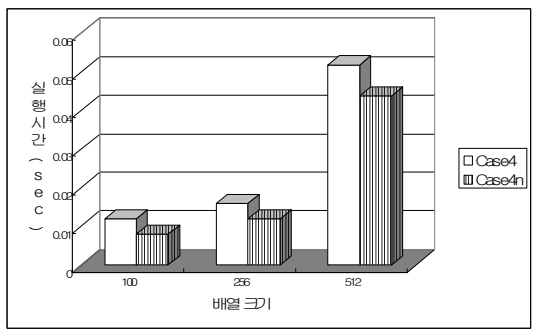
본 논문에서는 이 가운데 다중 문장 FORALL 구조를 순차 DO 루프로 스칼라화할 때 시맨틱이 유지되는 단일 루프를 생성할 수 있도록 경우에 따라 다른 루프 변환 방법을 적용하는 알고리즘을 제안하였다. 이렇게 변환된 코드와 PARADIGM 컴파일러 방법에 의해 변환된 코드의 실행시간을 측정하여 비교 평가한 결과, 루프 오버헤드를 줄이고 자료 재사용 기회를 높여 개선된 성능을 보였다.



(a) Case 2



(b) Case 3



(c) Case 4

그림 4-1 Case (2)~(4)의 실행시간
Fig.4-1 Execution times of (2)~(4) cases

www.kci.go.kr

참고문헌

〔1〕 Bozkus (1995), "Compiling Fortran 90D/HPF for Distributed Memory Computers," PhD Thesis, Syracuse University

〔2〕 Brandes (1994), "Compiling Data Parallel Programs to Message Passing Programs for Massively Parallel MIMD Systems," GMD, St. Augustin, Germany.

〔3〕 Chapman, Mehrotra and H. P. Zima,(1991), "Vienna Fortran-A Fortran Language Extension for Distributed Memory Multiprocessors," Univ. of Vienna , Austria.

〔4〕 High Performance Fortran Forum (1993), "High Performance Fortran Language Specification, Version 1.0," Technical Report CRPC-TR92225, Center for Research on Parallel Computation, Rice University, Houston, Texas.

〔5〕 Polychronopoulos et. al. (1989), "Parafrase-2: An environment for parallelizing, partitioning, synchronizing and scheduling programs on multiprocessors," Proceeding of the 18th Int'l Conference on Parallel Processing, pp.II: 39-48.

〔6〕 Callahan and K. Kennedy (1988) "Compiling Programs for Distributed-Memory Multiprocessors," Journal of Supercomputing 2, pp.151-169.

〔7〕 Hodges (1995), "High Performance Fortran Support for the PARADIGM Compiler," Master's Thesis, University Illinois at Urbana-Champaign.

〔8〕 Tseng (1993), "An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines," Technical Report Rice COMP TR-93-199, Rice University.

〔9〕 Wolfe (1995), "Optimizing Supercompilers for Supercomputers," Addison-Wesley Publishing Company.

〔10〕 Park S. S. (1993), "The Vectorization of Program using Graph Type Intermediate Representation Form," Ph.D Thesis, Korea University.

〔11〕 Zima and Chapman (1990), "Supercompilers for Parallel and Vector Computers," ACM Press.

〔12〕 Roth (2000), "Advanced Scalarization of Array Syntax", Lecture Notes In Computer Science; Vol. 1781 pp.219-231.

〔13〕 Zhao and K. Kennedy (2001), "Scalarizing Fortran90 Array Syntax," Technical report TR01-373, Rice University.

〔14〕 육현규, 구미순, 박성순, 박명순 (1996), "고성능자료 병렬언어 컴파일러에서의 최적화," 병렬처리시스템연구회지, 제7권, 제2호, pp.20-35.

〔15〕 Zhao and K. Kennedy (2005), "Scalarization Using Loop Alignment and Loop Skewing," Journal of Supercomputing 31, pp.5-46.

〔16〕 구미순, 박명순 (1999), "자료 병렬 언어 프로그램의 병렬 구조 변환을 위한 최적화기 설계," 정보처리논문지, 한국정보처리학회, 제6권 제3호, pp.792-803.

저 자 소 개

구 미순

고려대학교 이과대학 수학과 졸업
고려대학교 일반대학원 컴퓨터학과 석사
고려대학교 일반대학원 컴퓨터학과 박사 수료
현재: 백석문화대학 컴퓨터정보학부 조교수

www.kci.go.kr