High Performance Fortran 병렬 프로그래밍 변환기의 구현 및 성능 평가

김 중 권† · 홍 만 표†† · 김 동 규†††

요 약

분산 메모리 병렬 컴퓨터의 성능을 충분히 활용하고 프로그래밍의 난이도와 기종간 프로그램의 호환성을 해결하기 위하여 시스템 독립적이고 쉽게 프로그래밍 할 수 있는 데이터 병렬 언어에 대한 연구가 최근에 활발히 진행되고 있다. 대표적인 데이터 병렬 언어인 HPF 컴파일러는 사용자가 정의한 정보를 이용하여 데이터와 연산을 프로세서에 분할하여 할당하고, 메시지 패싱을 생성하는 기능을 제공함으로써 프로그램 작성자에게 전역 주소 공간을 이용하여 병렬 프로그램을 쉽게 개발할 수 있는 기반을 제공한다. 본 논문에서는 데이터 종속성 분석, 데이터 및 연산 분할과 메시지 패싱 코드 생성의 4 단계를 통하여, HPF 입력 프로그램을 MPI 메시지 패싱 코드가 삽입된 SPMD 프로그램으로 변환하는 HPF 병렬 프로그래밍 언어 변환기인 PPTran을 구현하고 그 성능을 검증한다.

Implementation and Performance Evaluation of Parallel Programming Translator for High Performance Fortran

Joong Kwon Kim† · Man Pyo Hong†† · Dong Kyoo Kim†††

ABSTRACT

Parallel computers are known to be excellent in performance per cost also satisfying scalability and high performance. However, parallel machines have enjoyed limited success because of difficulty in parallel programming and non-portability between parallel machines. Recently, researchers have sought to develop data parallel language that provides machine independent programming systems. Data parallel language such as High Performance Fortran provides a basis to write a parallel program based on a global name space by partitioning data and computation, generating message-passing function. In this paper, we describe the Parallel Programming Translator(PPTran), source-to-source data parallel compiler, generating MPI SPMD parallel program from HPF input program through four phases such as data dependence analysis, partitioning data, partitioning computation, and code generation with explicit message-passing and verify the performance of PPTran

1. 서 론

단일 프로세서의 성능 향상이 그 한계를 노출하면서 병렬 컴퓨팅은 빠른 연산처리를 요구하는 과학자와 기술자들에게 컴퓨터의 연산처리 속도를 지속적으로 증가시킬 수 있는 현재의 유일한 방법으로 인식되고 있다. 병렬 컴퓨터는 벡터형 슈퍼컴퓨터에 비하여 높은 가격 대 성능비를 제공한다. 특히 Intel의 Paragon, Thinking Machines의 CM 5와 CRAY Research Inc.

† 정 회 원 : ETRI 슈퍼컴퓨터센터 책임연구원
†† 종신회원 : 아주대학교 정보 및 컴퓨터공학부 교수
††† 정 회 원 : 아주대학교 정보 및 컴퓨터공학부 교수

논문접수 : 1999년 2월 8일, 심사완료 : 1999년 2월 26일

의 T3D와 같은 분산 메모리 컴퓨터는 매우 빠른 연산처리 속도를 제공하며, 확장성과 이식성이 뛰어난 병렬 컴퓨터이다. 이러한 많은 장점에도 불구하고 병렬 컴퓨터는 프로그래밍의 어려움과 프로그램 작성 시간의 과다로 전문가에 의해서만 제한적으로 사용되어 왔다. 또한, 병렬 컴퓨터의 프로그래밍은 시스템의 구조에 따라 프로그래밍 작성 방법이 다르다. CRAY C90같은 공유 메모리 MIMD(Multiple instruction, multiple data) 시스템은 PCF(Parallel Computing Forum) 포트란에서의 동기화 및 병렬 루프를 이용하여 병렬 프로그램을 작성하고[1], Thinking Machines의 CM-2 같은 SIMD (Single-instruction, multiple data) 시스템은 Fortran90의 병렬 배열구조를 이용하여 프로그래밍 한다[2]. 또한, CRAY T3E, 인텔 Paragon과 같은 분산메모리 MIMD 시스템은 사용자가 직접 메시지 패싱 기능을 제공하는 Fortran 77 프로그램을 작성해야 한다.

병렬 컴퓨터가 일반 사용자가 쉽게 접근하기 위해서는 벡터 슈퍼컴퓨터처럼 프로그래밍이 용이하고 시스템 독립적인 병렬 컴파일러가 필요하다. 병렬 컴파일러에 대한 연구는 80년대 중반부터 기존의 순차 프로그램을 완전 자동화하는데 집중되어 왔으나 기대한 만큼의 성능 향상이 이루어지지 않자, 90년대 초반부터는 사용자가 병렬성 정보를 컴파일러에게 제공하고 컴파일러는 그 정보를 이용하여 데이터와 계산을 각 프로세서에 분할하는 데이터 병렬 언어(Data Parallel Language)에 대한 연구가 주류를 이루어 왔다.

데이터 병렬 언어의 종류는 매우 다양하나, 일반적으로 기존의 순차적 프로그래밍 언어(Sequential Programming Language)에 데이터 병렬성(Data Parallelism)을 첨가한 형태의 것이 일반적이며, 특히 과학기술 계산 프로그램으로 유용하게 사용되어 왔던 포트란 언어에 이러한 데이터 병렬성을 지원하는 형태가 현재의 일반적인 경향이다. 그 결과 각종 포트란 언어에 대한 새로운 사양들이 제시되어 왔는데, 그중 대표적인 것들로 Callahan & Kennedy[3], SUPERB[4], KALI[5], CM -Fortran[6], Fortran D[7,8], ARF[9], Forge90[10], Vienna Fortran[11] 등이 있으며, 함께 컴파일러도 같이 발표되어 있다.

이렇게 다양한 병렬 컴파일러가 나타나게 되자 HPFF(High Performance Fortran Forum)에서는 표준화된 데이터 병렬 언어를 제공하기 위하여 HPF(High Performance Fortran) 사양을 제정하였으며, 현재는 부분 사양인 2.0 버전의 초안(draft)이 발표되어 있다[12, 13]. 따라서, HPF를 구현하는 컴파일러 연구는 아직 전세계적으로 초기 연구 단계인 편이며, HPF의 전체 사양을 구현한 상용화 수준의 시스템이 나오는데는 어느 정도 시간이 필요할 것으로 보인다. 그러나, HPF가 현재의 대규모 병렬 시스템에 적합한 프로그래밍 언어로서 인정되고 있는 지금, HPF 컴파일러의 개발은 전세계적으로 관심이 집중되고 있는 상태이며, 그에 대한 연구 역시 각지에서 활발히 진행되고 있다[14].

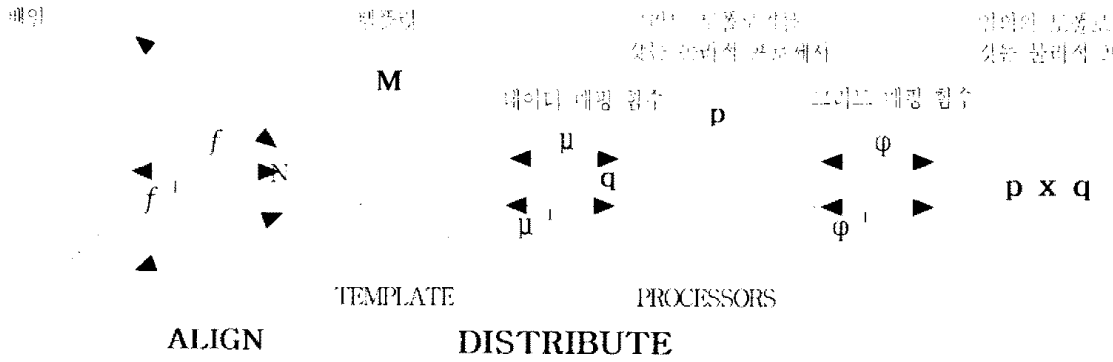
본 논문에서는 HPF로 작성된 프로그램을 Fortran 77 언어로 변환하고 MPI(Message Passing Interface) [15]의 통신 라이브러리를 자동으로 삽입시키는 HPF 컴파일러인 PPTran(Parallel Program Translator)을 구현하고, 병렬컴퓨터인 Cray T3E를 이용하여 그 성능을 검증한다. 이 PPTran은 기존의 PVM(Parallel Virtual Machine) 기반의 PPTran[16,17,18]을 MPI를 기반으로 하여 재구성 한 것으로, 사용자가 제공한 병렬성 정보와 데이터 종속성 정보를 이용, 데이터 병렬 언어의 핵심 기술인 데이터 분할, 연산 분할 및 통신 분석 기술을 구현하여, 성능 확장성과 이식성이 좋고 사용이 용이한 병렬 프로그래밍을 가능하게 해준다.

본 논문의 구성은 다음과 같다. 2장에서는 HPF의 데이터 분할을 위한 주요 기능적 특성을 알아보고, 3장에서 PPTran의 설계 및 구현 내용을 살펴보며, 4장에서 PPTran의 성능 분석 결과를 제시한다. 끝으로 5장에서 결론을 맺는다.

2. High Performance Fortran

HPF는 시스템 독립적인 데이터 병렬 모델을 제공하기 위하여 (그림 1)에서와 같이 두 단계 매핑 과정을 거쳐 배열의 데이터를 프로세서로 분할하는 기능을 정의한다. 첫 번째 단계에서는 프로세서간에 데이터 이동이 최소화하도록 배열들을 논리적 배열 공간인 템플릿에 정렬하며, 두 번째 단계에서는 정렬된 데이터들을 격자 구조의 논리적 프로세서에 분할한다. HPF에서는 이렇게 데이터를 물리적 프로세서로 매핑하는 대신 논리적 프로세서를 대상으로 매핑을 수행하므로 시스템 사이의 이식성을 제공한다[12,19].

HPF는 이 두 단계 병렬성을 위하여 사용자가 명시할 수 있는 TEMPLATE, PROCESSORS, ALIGN, DISTRIBUTE 등의 지시어(directive)를 제공한다. TEM-



(그림 1) HPF의 데이터 매핑 모델
(Fig. 1) Data mapping model For HPF

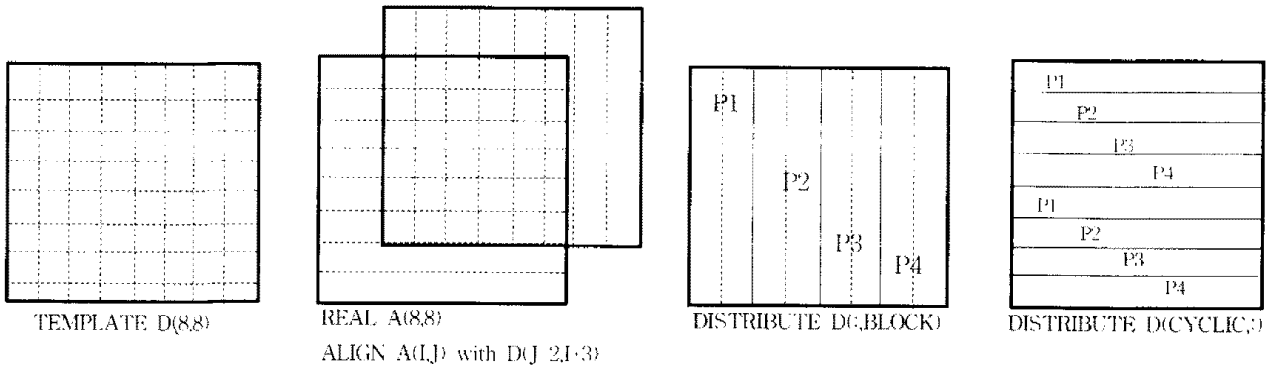
PLATE 지시어는 배열 요소를 위한 추상적 인덱스 정의에 사용되고, PROCESSORS 지시어는 배열 요소가 분할될 논리적 프로세서 격자를 정의하며, ALIGN 지시어는 배열 요소를 템플릿에 매핑해서 정렬하기 위해 사용된다. 배열의 템플릿으로의 정렬은 배열과 템플릿의 첨자에 의해 명시된다. 다음의 예에서 배열 D는 크기가 8×8인 2차원 템플릿으로 선언되어 있고, 배열 A는 열과 행을 치환하여 행은 2만큼 위로, 열은 3만큼 오른쪽으로 이동한 상태로 D와 정렬되어 있으며, (그림 2)의 첫 번째와 두 번째에 그 내용이 도시되어 있다.

```
!HPF$ REAL A(8,8)
!HPF$ TEMPLATE D(8,8)
!HPF$ ALIGN A(I,J) with D(J-2,I+3)
```

DISTRIBUTE 지시어는 BLOCK, CYCLIC, BLOCK_CYCLIC의 속성을 이용하여 배열 요소를 논리적 프로세서 격자에 분할한다. BLOCK은 배열 요소를 PROCESSORS 지시어에 명시되어 있는 프로세서 갯수에 균등하게 나누어 분할하고, CYCLIC은 라운드로빈 방식에 의하여 배열 요소를 분할하며, BLOCK_CYCLIC(k)는 한번에 k개의 배열 요소를 라운드로빈 방식에 의해 프로세서에 분할한다. ":"은 해당 차원의 배열 요소가 프로세서에 분할되지 않고 모든 프로세서에 복제됨을 의미한다.

(그림 2)의 3번째와 4번째에 각각 도시되어 있는 바와 같이 DISTRIBUTE D(:,BLOCK)은 배열의 열을 4개의 프로세서에 분할하고, DISTRIBUTE D(CYCLIC,:)은 배열의 행이 4개의 프로세서에 분할한다.

HPF는 데이터 분할 지원 이외에 프로그램에서 병렬로 수행될 수 있는 부분을 명시하는 FORALL 구문 등을 제공하여 사용자에 대한 병렬화를 지원한다. FORALL은 단문의 형태로서 FORALL 분, 또는 복수개의 문장을 갖는 FORALL 구문으로 사용될 수 있다. FORALL은 배열의 각 요소들에 대한 지정문을 동시에 실행시키고자 할 때 사용하는데, FORALL 다음에 오



(그림 2) TEMPLATE, ALIGN 및 DISTRIBUTE 지시어
(Fig. 2) TEMPLATE, ALIGN, and DISTRIBUTE

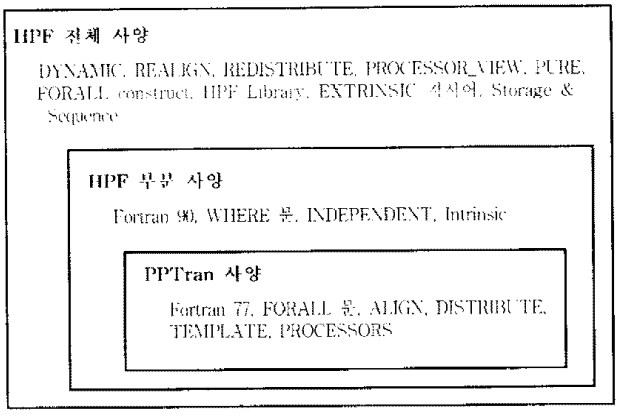
는 표현식에는 연산에 필요한 배열 요소의 식과, 참 및 이용되는 배열 요소간의 간격을 정의하며 다음에 그 사용 예를 보이고 있다.

FORALL (I=1:N,J=1:M) A(I,J)=I+J

FORALL (I=1:N,J=1:M/2) A(I,J)=I*B(J)

3. PPTran

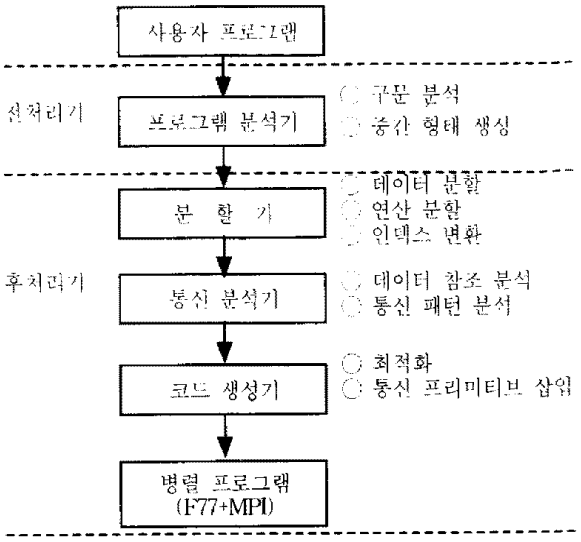
본 논문에서는 HPF로 작성된 데이터 병렬 언어를 입력으로 받아 분산 메모리 구조를 갖는 고성능 컴퓨터에서 수행 가능한 SPMD 프로그램을 생성하는 병렬 컴파일러 PPTran(Parallel Program Translator)을 구현하고 그 성능을 분석한다. PPTran에서의 구현 범위는 (그림 3)과 같다.



(그림 3) PPTran 구현 범위
(Fig. 3) Implementation Scope of PPTran

3.1 PPTran 구성

PPTran은 입력된 HPF 프로그램을 MPI 메시지 패싱 프리미티브가 삽입된 Fortran 77 원시(source) 코드를 생성하며, 이를 위해 (그림 4)와 같이 프로그램 분석, 데이터 및 연산 분할, 통신 분석, 코드 생성의 4가지 단계를 수행한다. 프로그램 분석단계에서는 입력 프로그램이 문법적으로 정확한가를 조사하고, AST (Abstract Syntax Tree)와 Name Table을 생성하며, 데이터 종속성(Data Dependence) 분석도 수행한다[20]. 데이터 및 연산 분할 단계에서는 사용자가 명시한 데이터 분할 지시어를 분석하고[7,8] 병렬문과 배열 요소를 각 프로세서에서 수행될 순차화 코드로 변환 및 분할 작업을 수행한다. 통신 분석단계에서는 각 프로세서에서 비 지역 데이터(non-local data)를 분석하여[21] 통신 프리미티브가 삽입될 위치를 조사하며[7,8], 코드 생성 단계에서는 메시지 패싱 프리미티브가 첨가된 순차 Fortran 77 원시 코드를 생성한다.



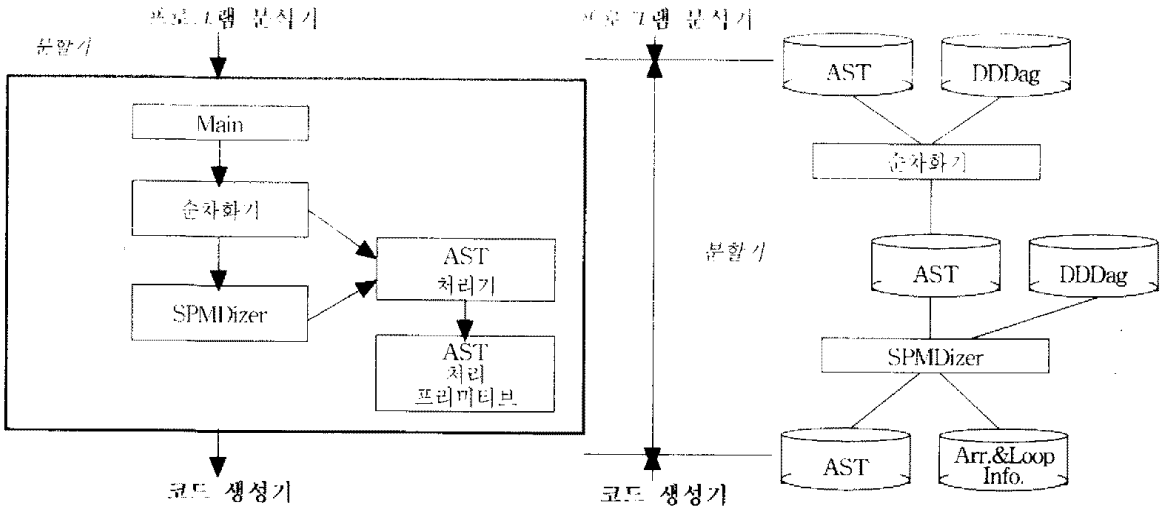
(그림 4) PPTran의 개념적 컴파일 단계
(Fig. 4) Conceptional Compilation Phases of PPTran

3.2 프로그램 분석기(Program Analyzer)

PPTran의 프로그램 분석기는 입력된 HPF 프로그램의 구문이 정확한가를 검사하고, 다음 단계 에서 이용될 중간형태(Intermediate Representation)인 AST, Name Table 및 데이터 종속성 DAG(Directed Acyclic Graph)를 구성한다. AST는 입력 프로그램의 모든 구문들을 축약 파스 트리 형태로 나타내며, Name Table은 일종의 심볼 테이블로 변수의 선언 정보를 담고 있으며 배열의 경우에는 데이터 분할 정보도 포함한다. 데이터 종속성 DAG는 변수간의 데이터 종속성 정보를 저장하는 자료구조로서 종속성의 종류(flow, anti, output), 종속성 방향 벡터, 종속성 거리 등의 정보가 보관된다.

3.3 분할기(Partitioner)

프로그램 분석이 종료되면, PPTran은 코드의 순차화 및 데이터 분할과 연산 분할 정보를 분석, 보관하고 또한, 적절한 메시지 패싱 프리미티브 첨가를 위한 통신 분석 기능을 수행하며, (그림 5)에서와 같이, PPTran의 분할기(Partitioner)가 이 역할을 수행한다. 분할기는 Serializer, SPMDizer, 코드 생성기로 구성된다.



(그림 5) PPTran의 분할기
(Fig. 5) Partitioner of PPTran

3.3.1 순차화기(Serializer)

코드의 순차화란 HPF 프로그램을 그 의미가 유지되는 Fortran 77 구문으로 변환하는 것을 의미 하며, 분할기의 한 부분인 순차화기가 이 기능을 수행한다. 순차화기는 프로그램 분석기에서 생성된 AST를 탐색하여 FORALL 문과 배열 표현에 해당하는 노드를 찾아 Fortran 77의 DO 루프 노드로 치환한다. 그런데, FORALL 문은 FORALL 문 내의 rhs(right hand side) 배열이 모든 인덱스에 대해 루프 이전에 정의된 값이어야 한다는 시맨틱스를 가지기 때문에, lhs(left hand side)와 rhs 배열요소간에 종속성이 존재하는 경우에는 변환된 순차 DO 루프에서는 이 같은 시맨틱스가 유지되지 않는다. 따라서 순차화기는 FORALL 문을 DO 루프로 변환하기 전에 데이터 종속성 DAG의 정보를 이용하여 종속성 존재 여부를 검사하여, 종속성이 존재하면 루프이전에 rhs 배열의 값을 임시 기억장소에 복사(clone)해 두었다가 변환된 DO 루프 내 rhs에서 이 임시 배열을 원래의 배열 대신 참조해야 된다. 임시 배열로의 복사 작업이 요구되는 경우는 임시 배열에 대한 선언 작업도 이루어 져야 한다. 이 같은 선행 작업이 수행된 후 마지막으로 FORALL 문을 대체할 DO 루프를 생성한다. 배열 표현을 DO 루프로 변환하기 위해서는 우선 모든 lhs와 rhs의 각 배열에 대해, 각 배열의 차원의 수만큼의 인덱스 부분이 추가되고 DO 루프 노드가 생성된다. 다음은 FORALL 문과 배열 표현을 DO 루프로 변환하는 예를 보여 주며, X와 Y는 배열 크기가 15로 미리 선언되어 있다.

```
FORALL(I=1:15, X(I)=Y(I))   → do i1=1, 15
                                x(i1)=y(i1)
                              enddo
X=Y+1                       → do i1=1, 15
                                x(i1)=y(i1) + 1
                              enddo
```

3.3.2 SPMDizer

PPTRan에서 SPMDizer는 각 프로세서로 분할될 배열의 요소를 계산하고, 각 프로세서에서 실행될 연산을 분할하며, 또한 프로세서간에 전송될 비 지역 데이터 집합 계산, 메시지 패싱 프리미티브를 첨가할 위치 등을 분석한다.

● 데이터 분할

PPTran에서의 데이터 분할은 Fortran D의 데이터 분할 방법을 이용하여 BLOCK과 CYCLIC 분할 형태를 지원하며, Name Table에 보관된 배열에 대한 정보를 기반으로 하여 이루어진다. PPTran은 SPMD 노드 프로그램을 생성하기 때문에, 모든 프로세서는 동일한 배열 선언을 해야되며, 따라서 모든 프로세서는 연산을 위해 지역 인덱스(local index)를 이용한다. (그림 6)은 배열 A가 4개의 프로세서에 BLOCK 분할 시 생성된 노드 프로그램을 예시한 것으로, A의 4개 프로세서에 대한 전역 인덱스(global Index)가 각각 [1:25], [26:50], [51:75], [76,:100]인 반면에, 지역 인덱스는 모두 프로세서가 [1:25]이다.

```
REAL A(100)                 REAL A(25)
do I = 1, 100               do i = 1, 25
  A(I) = 0.0                  A(i) = 0.0
```

enddo (원래의 프로그램) (Original program)	enddo (SPMD 노드 프로그램) (SPMD node program)

(그림 6) 전역 인덱스에서 지역 인덱스로의 변환
(Fig. 6) Converting global to local indices

일반적으로, BLOCK이나 CYCLIC과 같이 정형화된 형태로 분할되는 경우의 전역 인덱스와 지역 인덱스간의 변환관계는 <표 1>과 같이 구할 수 있으며, I는 전역 인덱스, i는 지역 인덱스, N은 배열 요소의 개수, P는 프로세서 개수, p는 프로세서 번호($0 \le p \le P-1$) 이다.

<표 1> 정규 데이터 분할에 대한 인덱스 매핑
<Table 1> Indices Mapping for Regular Data Decomposition

구분	Global to Local	Local to Global
BLOCK	i = I - p*N/P	I = i + p*N/P
CYCLIC	i = ⌊(I - p - 1)/P⌋ + 1	I = (i-1)*P + p + 1

● 연산 분할

데이터 분할 분석 단계가 끝나면, 프로세서간에 데이터와 연산을 분할한다. 먼저 프로세서간에 배열 요소를 분할한 후에 소유자 연산법칙(Owner computes rule)[3]에 따라 루프 반복(loop iteration)을 분할한다. 먼저, 이 논문에서 사용될 몇 가지 용어를 정의한다. 어느 프로세서에서의 지역 반복 집합(local iteration set)은 그 프로세서가 소유하고 있는 데이터를 참조하는 루프의 반복 집합을 의미하며, 지역 인덱스 집합(local indices set)은 프로세서에서 소유하고 있는 지역 배열 요소의 부분을 의미한다. PPTran에서는 RSDs (Regular Section Descriptors) 혹은 3-tuple 슬라이스(slice) 개념을 이용하여 반복 집합이나 인덱스 집합을 표현한다. RSDs는 $[l_1:u_1:s_1,...]$로 표시되며, l_i, u_i 및 s_i는 i번째 차원의 상한값, 하한값 및 간격을 나타낸다. 중첩 루프나 다차원 배열에서 RSD의 최 좌측 차원은 가장 바깥 루프(outermost loop)나 배열의 최 좌측 차원을 나타내고, 다른 차원은 순서적으로 열거한다. 간격은 명시되어 있지 않으면 1로 가정한다.

연산 분할의 첫 단계로서 지역 인덱스 집합을 결정한다. 이 과정을 (그림 7)의 Jacobi 프로그램으로 예시하면 다음과 같다. 프로세스 개수는 4개로 가정하였으며, 배열 A와 B는 템플릿 D와 똑 같이 정렬되어 있으므로 D와 같은 분할을 갖는다. D의 1차원은 모든 프로세서에 의해 소유되고 2차원은 블록 분할이므로, 배열 A와 B의 지역 인덱스 집합은 모든 프로세서에 있어서 [1:100, 1:25]가 된다.

```
REAL A(100,100), B(100,100)
CHF$ TEMPLATE D(100,100)
CHF$ A, B with D
CHF$ DISTRIBUTE D(:, BLOCK)
        do k = 1, 50
          do j = 2, 99
            do i = 2, 99
S1                A(i, j) = (B(i,j-1) + B(i-1,j) +
                             (B(i+1) + B(i,j+1))/4
            enddo
          enddo
          do j = 2, 99
            do i = 2, 99
S2              B(i,j) = A(i,j)
            enddo
          enddo
        enddo
```

(그림 7) Jacobi 프로그램
(Fig. 7) Jacobi program

BLOCK과 CYCLIC 분할에 대한 일반적인 지역 인덱스 집합을 결정하는 알고리즘은 (그림 8) 및 (그림 9)와 같으며, <표 2>는 이 알고리즘에서 사용되는 변

<표 2> 지역 인덱스 집합 결정 요소
<Table 2> Varibles determining local indices set

변 수 명	내 용	변 수 명	내 용
lower	전역 배열 범위의 하한값	GUpper(tp)	tp가 소유하는 배열 범위의 상한값
upper	전역 배열 범위의 상한값	GStride(tp)	tp가 소유하는 배열 범위의 간격
ArrSize	전역 배열의 크기(upper-lower+1)	BlockSize	지역 배열의 최대 크기($\lceil \frac{ArrSize}{nProcs} \rceil$)
StartProc	배열의 첫번째 요소가 놓이는 프로세서		
LastProc	배열의 마지막 요소가 놓이는 프로세서	NoElements	지역 배열요소 수($\lceil \frac{Arrsize-\lvert Align\_offset \rvert}{nProcs} \rceil$)
tp	프로세서 id		
Align_offset	정렬식의 오프셋	Shortage	배열이 각 프로세서 상에 동일한 크기로 나뉘지 않을 경우의 부족분
GLower(tp)	tp가 소유하는 배열 범위의 하한값		

수단이다. 각 프로세서 tp가 소유하고 있는 지역 인덱스 집합 OwnSet(tp)는 (그림 8)과 (그림 9)로부터 (GLower(tp):GUpper(tp): GStride(tp))가 된다.

```
if (Align_offset > 0)
  if (tp < StartProc)
    GLower(tp)=0
    GUpper(tp)=0
  else
    if (tp = StartProc)
      GLower(tp)=lower
      GUpper(tp)=GLower(tp)+BlockSize-Align_offset
                 %BlockSize-1
    else
      GLower(tp)=tp*BlockSize-Align_offset+lower
      GUpper(tp)=GLower(tp)+BlockSize-1
    endif
  endif
else
  if (tp > LastProc)
    GLower(tp)=0
    GUpper(tp)=0
  else
    if (tp < LastProc)
      GLower(tp)=lower+tp*BlockSize-Align_offset
      GUpper(tp)=GLower(tp)+BlockSize-1
    else
      GLower(tp)=lower+tp*BlockSize-Align_offset
      GUpper(tp)=GLower(tp)+BlockSize-1-Shortage
    endif
  endif
endif
Gstride(tp)=1
```

(그림 8) 지역 인덱스 집합 결정 알고리즘 : BLOCK 분할
(Fig. 8) Algorithm determining local indices set: BLOCK decomposition

```
if (Align_offset > 0)
 if (tp = StartProc)
   GLower(tp)=lower
   GUpper(tp)=GLower(tp)+(NoElements-1)*nProcs
 else
   if (tp < StartProc)
     GLower(tp)=lower+nProcs-Align_offset%nProcs+tp
     GUpper(tp)=GLower(tp)+(NoElements-⌈(Align_offset)/nProcs⌉)
                * nProcs
   else
     GLower(tp)=lower+tp-Align_offset%nProcs
     GUpper(tp)=GLower(tp)+(NoElements-1)*nProcs
   endif
 endif
else
 if (tp = LastProc)
   GLower(tp)=lower-Align_offset+tp
   GUpper(tp)=GLower(tp)+(NoElements-1)*nProcs
 else
   GLower(tp)=lower-Align_offset+tp
   GUpper(tp)=GLower(tp)+(NoElements-2)*nProcs
 endif
endif
Gstride(tp)=nProcs
```

(그림 9) 지역 인덱스 집합 결정 알고리즘 : CYCLIC 분할
(Fig. 9) Algorithm determining local indices set: CYCLIC decomposition

여기서 S_1 에서생성되어 지역 반복 집합은 배열 참조 함수의 역함수를 tp 프로세서의 지역 인덱스 집합에 적용한 후, 그 결과를 루프의 반복 집합과의 교집합으로 구할 수 있다. 어느 프로세서가 소유자 인산 법칙을 적용하여 루프내의 배열 할당문을 실행시킬 때, 실행되어야 할 루프 반복 집합은 그 할당문의 lhs의 지역 반복 집합과 실제로 똑 같다[7].

지역 반복 집합을 구하는 알고리즘을 예시하기 위하여 (그림 7)의 S_1 할당문을 생각해 보자. A의 지역 인덱스 집합은 앞에서 구한 [1:100,1:25]이고, lhs인 A(i,j)의 참자함수의 역함수는 [:,1:25,1:100]이 된다. 여기서 1차원의 ":"는 k루프의 모든 반복에서 A의 지역 요소를 참조하기 때문이다. 이 첨자함수의 역함수를 루프 중첩의 실질적인 반복 집합인 [1:50,2:99,2:99]와의 교집합을 구하면 [1:50,1:25,2:99]이 된다. 각 프로세서의 지역 인덱스로 변환시키면 배열 A에 대해, 나음과 같은 지역 반복 집합을 얻을 수 있다.

프로세서(0) = [1:50,2:25,2:99]
프로세서(1:2) = [1:50,1:25,2:99]
프로세서(3) = [1:50,1:24,2:99]

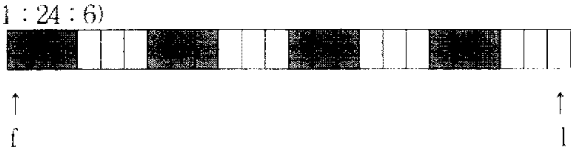
여기서 주의 할 점은 양끝에 존재하는 프로세서(0)와 프로세서(3)은 배열의 2차원에서 차이가 존재한다. 즉, 경계 조건 처리가 필요함을 보여 주고 있다. S2 할당문에 대해서도 위의 방식을 적용해 보면, 같은 결과가 얻어 진다.

● 통신 분석

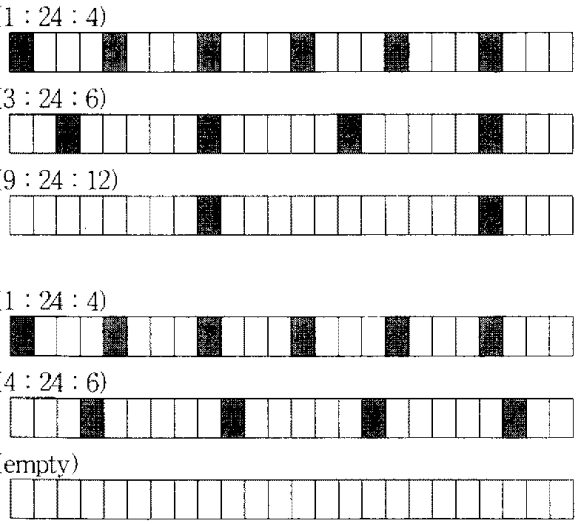
데이터 및 연산 분할 분석이 끝난 후, PPTran은 통신 분석을 수행한다. 통신 분석에서는 루프내의 어느 변수 참조가 비 지역 데이터를 접근하는가를 분석하고 또한, 통신비용을 감소시키기 위한 최적화를 수행한다.

프로세서간에 송수신 되는 배열 요소의 집합은 슬라이스의 교집합 개념을 바탕으로 하여 계산 할 수 있다[21]. 앞에서 정의한 바와 같이, $A(f_a: l_a: s_a)$-$B(f_b: l_b: s_b)$와 같은 배열 할당문에서 A(f: l: s)는 A(f + i*s)처럼 배열 요소를 정의하는 슬라이스이다. 여기서 f, l 및 s는 각각 첫 번째 배열 요소 첨자, 마지막 배열 요소의 첨자 및 한 요소의 첨자와 다음 요소의 첨자간의 간격을 나타내며, i는 $0 \le i \le \lfloor (l-f)/s \rfloor$로 정의되는 값이다. HPF에서 데이터 분할을 위해 BLOCK, CYCLIC 및 BLOCK-CYCLIC의 3가지 방식이 지원되나, BLOCK과 CYCLIC은 BLOCK-CYCLIC의 특별한 방식이므로, 모든 배열은 각 프로세서에 BLOCK-CYCLIC으로만

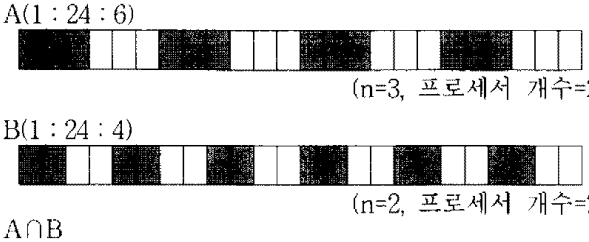
분할된다고 가정한다. BLOCK-CYCLIC(n) 같은 분할 방식에서 s는 다음 예와 같이 블록 크기 n과 프로세서 개수의 곱으로 표현될 수 있다(n=3, 프로세서 개수=2).



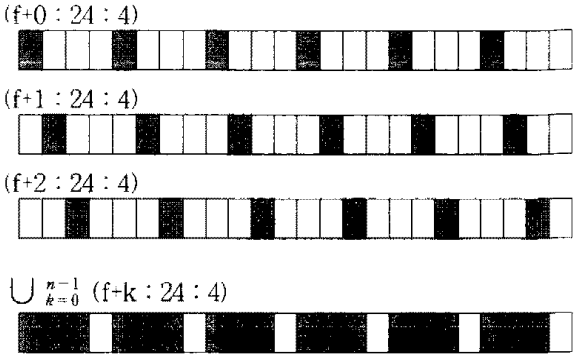
일반적으로 슬라이스의 교집합은 (그림 10)처럼 공집합이나 또 다른 슬라이스이므로, 슬라이스는 교집합에 대하여 닫혀 있으나, BLOCK-CYCLIC에서의 슬라이스는 교집합에 대하여 (그림 11)처럼 닫혀 있지 않다. 따라서 BLOCK-CYCLIC에서의 슬라이스는 그림 12처럼 각각의 슬라이스의 합집합 $\bigcup_{k=0}^{n-1}$ (f + k:l:s)로 대하여 교집합을 구한다. 이렇게 확장된 개념의 슬라이스에서, (f_1:l_1:s_1)와 (f_2:l_2:s_2)의 교집합은 (max(f_1,f_2) : min(l_1,l_2):LCD(s_1,s_2))가 된다. 여기서 LCD는 최소공배수를 의미한다.



(그림 10) 두 슬라이스의 교집합 예
(Fig. 10) Two examples of slice intersection



(그림 11) 두 BLOCK-CYCLIC의 슬라이스 교집합
(Fig. 11) Intersection of two BLOCK-CYCLICs



(그림 12) 각 개별 슬라이스의 합집합으로 나타낸 BLOCK-CYCLIC 분할(n=3)
(Fig. 12) A BLOCK-CYCLIC distribution as a union of disjoint slices(n=3)

이제 확장된 개념의 슬라이스를 이용하여 어느 프로세서가 다른 프로세서로(혹은 부터) 송신(혹은 수신)할 데이터를 계산할 수 있다. 임의의 프로세서가 연산에 필요한 데이터의 보유 유무는 할당문의 lhs항을 기준으로 결정한다. 즉, 자신이 보유하고 있는 lhs항이 i번째 연산에 이용될 경우, i번째 연산 수행에 필요한 rhs항의 데이터가 있는지를 검사하면 된다. 배열 $A(f_1 : l_1 : s_1)$와 $B(f_2 : l_2 : s_2)$가 각 프로세서에 BLOCK-CYCLIC 분할법으로 분배되어진 후, $A(f_a : l_a : s_a) = F(B(f_b : l_b : s_b))$를 병렬로 수행한다고 하자. 임의의 프로세서 D는 연산식을 수행하기 위해 임의의 프로세서 S의 메모리에 있는 배열 B의 데이터를 필요로 할 수 있다. 이를 위해, 송신 프로세서 S는 수신 프로세서 D가 필요로 하는 데이터를 검출하여 이를 D로 송신해야 한다. 배열 A와 B는 BLOCK-CYCLIC 분할이기 때문에 D에서 보유하고 있는 배열 A의 부분 집합과 S에서 보유하고 있는 배열 B의 부분 집합은 각각 (1)과 (2)에 의해 구해진다. 식 (1)과 (2)에서 n_D와 n_S는 프로세서 D와 S에 분배된 블록의 크기이며, f_D, l_D, s_D, f_S, l_S, s_S는 D와 S가 각각 A와 B에 대해 보유하고 있는 배열 요소 집합 즉, 지역 인덱스 집합을 나타내는 3-tuple 슬라이스의 첨자이다.

$$O_{WND}(A) = \bigcup_{j=0}^{n_v-1}(f_D + j : l_D : s_D) \qquad (1)$$

$$O_{WNS}(B) = \bigcup_{i=0}^{n_S-1}(f_s + j : i : s_S) \qquad (2)$$

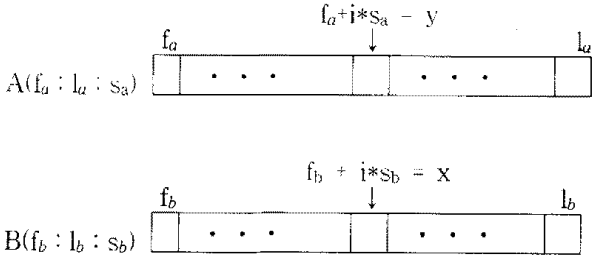
다음에 프로세서 D에서 보유하고 있으면서 프로세서 D의 연산에 이용되는 배열의 집합과 프로세서 S에서 보유하고 있으면서 프로세서 D의 연산에 이용되는 배열의 집합을 구하면 각각 식 (3) 및 (4)와 같다.

$$S_1 = O_{WND}(A) \cap (f_a : l_a : s_a) \qquad (3)$$

$$S_2 = O_{WNS}(B) \cap (f_b : l_b : s_b) \qquad (4)$$

프로세서 S에서 프로세서 D로 송신할 배열 요소의 집합은 다음과 같이 결정한다. i번째 배열 요소인 f_a+i*s_a가 S_1의 집합이고, f_b+i*s_b가 S_2의 집합이라면 프로세서 S는 $B[f_b+i*s_b]$를 프로세서 D로 송신하고 프로세서 D는 그 데이터를 $A[f_a+i*s_a]$에 저장한다. 이를 교집합 개념으로 표현하려면 새로운 함수식이 필요하다. (그림 13)과 같이 배열 B의 i번째 요소(f_b+i*s_b)인 x가 배열 A의 i번째 요소(f_a+i*s_a)인 y에 대응한다고 할 때 x를 구하기 위한 함수식은 (5)와 같다.

$$Map(y) = Map(f_a + i*s_a) = f_b + i*s_b = f_b + ((y - f_a)/s_a)*s_b \qquad (5)$$



(그림 13) x와 y의 관계
(Fig. 13) x and y

따라서 프로세서 S에서, x가 $Map(S_1)$에 포함되고 동시에 S_1에 포함되면 배열 B의 요소 중 B(x)를 프로세서 D로 송신하면 되고, 따라서 프로세서 S에서 프로세서 D로 전송할 배열 B의 요소집합은 식 (6)과 같으며, 프로세서 S로부터 프로세서 D로 전송해야 할 배열 요소의 집합을 결정 하는 알고리즘은 (그림 14)와 같다.

$$SendSet = S_2 \cap Map(S_1) = O_{WNS}(B) \cap Map(O_{WND}(A) \cap (f_a : l_a : s_a)) \qquad (6)$$

프로세서 D는 프로세서 S로부터 송신된 메세지를 수신한 후, 수신 버퍼에 있는 메세지를 연산을 수행하기 위해 배열 B의 요소에 매핑해야 한다. 이를 위해 프로세서 D에서는 프로세서 S로부터 전송된 메세지를 배열 B와 매핑시키기 위해 프로세서 S에서 전송할 배열 요소를 결정하기 위해 수행한 송신측 알고리즘을 다시 수행한 후 식 (5)에 있는 Map 함수의 역함수를 수행하여 수신 버퍼에 있는 데이타를 연산에 사용되는 순서로 매핑 할 수 있다. 수신측 알고리즘은 송신측과 비교해 볼 때, Map 함수에 대한 역함수를 이용하는 것 이외에는 동일하다. 수신측 알고리즘은 (그림 15)와 같다.

```
BEGIN Algorithm
  SendSet = {empty}
  DO j = 0 UPTO nD - 1
    DO i = 0 UPTO nS - 1
      SendSet = SendSet ∪ ((fS+i : lS : sS) ∩ Map((fD+j
                : lD : sD) ∩ (fa : la : sa))
    END DO
  END DO
END Algorithm
```

(그림 14) 송신할 배열의 요소를 결정하는 알고리즘
(Fig. 14) Algorithm for computing the indices of the array elements to send

```
BEGIN Algorithm
  RecvSet = {empty}
  DO i = 0 UPTO nS - 1
    DO j = 0 UPTO nD - 1
      RecvSet = RecvSet ∪ Map-1((fD+j : lD : sD) ∩ (c) ∩
                (fS+i : lS : sS))
    END DO
  END DO
END Algorithm
```

(그림 15) 수신할 배열의 요소를 결정하는 알고리즘
(Fig. 15) Algorithm for computing the indices of the array elements to receive

지금까지의 과정을 예시하기 위하여 (그림 16)과 같이 배열 크기가 각각 12인 A와 B를 프로세서 2개에 BLOCK-CYCLIC 방식으로 분할하고 다음과 같은 프로그램을 수행한다고 하자.

```
do i = 1, 12
   A(i) = B(i)
enddo
```

그러면 프로세서 0가 보유하고 있는 배열 A와 B의 요소는 각각 다음과 같이 구해진다.

$$\bigcup_{k=0}^{2} A(f_D + k : l_D : s_D) = \bigcup_{k=0}^{2} A(1 + k : 12 : 6)$$
$$= \{A(1), A(2), A(3), A(7), A(8), A(9)\}$$

$$\bigcup_{k=0}^{1} B(f_s + k : l_s : s_S) = \bigcup_{k=0}^{1} B(1 + k : 12 : 4)$$
$$= \{B(1), B(2), B(5), B(6), B(9), B(10)\}$$

마찬가지로 프로세서 1이 보유하고 있는 배열 요소는 다음과 같다.

$$\bigcup_{k=0}^{2} A(f_D + k : l_D : s_D) = \bigcup_{k=0}^{2} A(4 + k : 12 : 6)$$
$$= \{A(4), A(5), A(6), A(10), A(11), A(12)\}$$

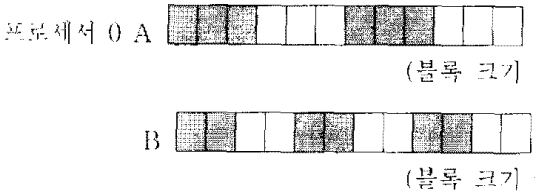
$$\bigcup_{k=0}^{1} B(f_s + k : l_s : s_S) = \bigcup_{k=0}^{1} B(3 + k : 12 : 4)$$
$$= \{B(3), B(4), B(7), B(8), B(11), B(12)\}$$

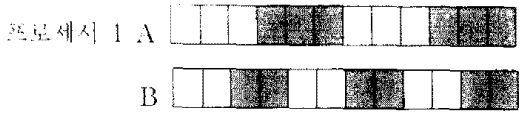
또한, 프로세서 0에 속하면서 계산에 참여한 배열 요소는 {A(1), A(2), A(3), A(7), A(8), A(9)}과 {B(1), B(2), B(5), B(6), B(9), B(10)}이며, 프로세서 1에 속하면서 계산에 참여한 배열 요소는 {A(4), A(5), A(6), A(10), A(11), A(12)}과 {B(3), B(4), B(7), B(8), B(11), B(12)}가 얻어진다. 따라서 프로세서 0에서 프로세서 1로 보내는 B의 배열 요소는 다음과 같이 구해진다.

{프로세서 0에 있는 B의 배열 요소} ∩ {프로세서 1에서의 Map(y)}

{B(1), B(2), B(5), B(6), B(9), B(10)} ∩
{Map(4)=B(4), Map(5)=B(5), Map(6)=B(6),
Map(10)=B(10), Map(11)=B(11), Map(12)=B(12)}=
{B(5), B(6), B(10)}

동일한 방식에 의해 프로세서 1에서 프로세서 0로 보내는 B의 배열 요소는 {B(3), B(7), B(8)}이 된다.





(그림 16) 배열 A와 B의 분할된 형태
(Fig. 16) BLOCK-CYCLIC distribution of A and B arrays

프로세서간에 송신할 배열 요소의 집합을 결정한 후에, PPTran은 통신비용을 감소시키기 위한 최적화를 수행한다. PPTran에서는 메시지 벡터화 기법과 연산·통신 중첩 기법을 적용한다. 메시지 벡터화는 LCD(Loop-carried Data) 종속성의 레벨을 이용하여 통신이 바깥(outer) 루프에서 수행될 수 있는가를 분석하여 여러 개의 적은 메시지들을 하나의 큰 메시지로 묶어서 전송할 수 있게 하기 때문에 통신의 초기 비용과 지연시간(latency)을 감소시킨다. 메시지를 벡터화시키기 위해서는 종속성을 야기하는 시작 부분과 마지막 부분을 포함하는 가장 깊숙한 루프의 레벨을 찾아내야 한다. 즉, 프로세서간 순 데이터 종속성(true Data Dependence)이 발생되는 가장 깊숙한 루프의 헤더가 메시지를 벡터화 하는 위치가 된다. 이를 예시하기 위하여 다시 (그림 7)의 Jacobi 프로그램을 이용하면, 앞의 통신 분석 단계에서 S_1의 B(i,j-1)과 B(i,j+1)이 비지역 데이터를 접근함을 알 수 있으며, B(i,j-1)과 B(i,j+1)는 k루프에서 S_2의 B로부터 프로세서간 순 데이터 종속성이 발생하기 때문에, 이 정보는 저장되었다가 코드 생성 단계에서 메시지 패싱 프리미티브 삽입 시에 이용된다

연산·통신 중첩 기법은 루프내의 배열 참조 중 종속성이 존재하는 배열 접근에 사용된다. 이러한 배열 접근은 루프 이전에 모든 배열원소를 확보하는 벡터화 방법을 사용할 수 없고, 반드시 루프 내에서 데이타가 정의된 후 통신을 수행해야 한다. PPTran에서는 이러한 경우 통신 오버헤드를 줄이기 위해 통신과 연산을 중첩시키는 최적화 기법을 사용한다. 즉 프로세서마다 데이터의 송수신 시간을 분리하여, 다른 프로세서에서 필요한 데이터는 정의된 직후 송신하고, 송신된 자료는 사용되기 직전에 수신함으로써 통신으로 인한 오버헤드를 최소화할 수 있다.

3.4 코드 생성(Code Generator)

컴파일러 부분의 가장 마지막 단계인 코드 생성기는 SPMDizer에서 생성한 AST로부터 Fortran 77 코드

로 구성되는 코드를 생성한다. 생성된 코드는 배열 요소의 인덱스 축소, 경계조건을 고려한 루프 인덱스 축소, 수신 데이터 저장 장소 확보, 및 MPI 메시지 패싱 코드가 삽입된 원시 코드이다. (그림 7)의 Jacobi 프로그램은 코드 생성 단계에서 (그림 17)과 같이 변환된다.

```
      REAL A(100, 25), B(100, 0:26)
      #INCLUDE <MPIF.H>

C     MPI Initialization
      CALL MPI_INIT(IERR)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NPES, IERR)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD, MYPE, IERR)

      do k = 1, 50
      if (MYPE .EQ. 0) lb1 = 2 else lb1 = 1
      if (MYPE .EQ. NPES-1) ub1 = 24 else ub1 = 25

C     SEND
        IF (MYPE .GT. 0) CALL MPI_SEND(B(2:99, 1), 98,
          MPI_REAL, MYPE-1,
     *                11, MPI_COMM_WORLD, IERR)
        IF (MYPE .LT. NPES-1) CALL MPI_SEND(B(2:99, 25),
98, MPI_REAL, MYPE+1,
     *                10, MPI_COMM_WORLD,IERR)
C       RECEIVE
        IF (MYPE .LT. NPES-1) CALL MPI_RECV(B(2:99, 26),
          98, MPI_REAL,
     *                MPI_ANY_SOURCE, 11, MPI_COMM_
                      WORLD, STATUS, IERR)
        IF (MYPE .GT. 0) CALL MPI_RECV(B(2:99, 0), 98,
          MPI_REAL,
     *                MPI_ANY_SOURCE, 10, MPI_COMM_
                      WORLD, STATUS,IERR)

        do j = lb1, ub1
              do i = 2, 99
     S1           A(i, j) = (B(i,j-1) + B(i-1,j) + B(i+1) +
                        B(i,j+1))/4
              enddo
            enddo
        do j = lb1, ub1
              do i = 2, 99
     S2         B(i,j) = A(i,j)
              enddo
        enddo
      enddo
      CALL MPI_FINALIZE(IERR)
      end
```

(그림 17) MPI SPMD 코드로 변환된 Jacobi 프로그램
(Fig. 17) Generated Jacobi

(그림 17)에서 NPES, MYPE는 각각 프로세서 개수, 0부터 MYPE-1의 값을 갖는 프로세서 번호이며 NPES는 4로 가정되어 있다. 배열 A, B의 인덱스는 각각 축소되어 있으며, B의 경우는 데이터를 수신하기 위한 여상 장소로서 0과 26번째 요소가 설정되어 있다. 루프 인덱스는 j루프에서 경계 조건을 고려한 상황에서 축소되어 있다. 통신 분석 단계에서 비 지역 데이터를 접근하는 B(i,j+1)에 대해, 프로세서 MYPE에서 프로세서 MYPE-1로 보내야 할 배열 요소는 [2:99, 1], B(i,j-1)에 대해서는 프로세서 MYPE에서 프로세서 MYPE-1로 보내야 할 배열 요소는 [2:99, 25]로 분석되고, 또한, k루프에서 프로세서간 순 데이터 종속성이 존재하기 때문에 통신 코드는 k루프내의 헤더부분에 첨가되어 있다. 이 통신 코드는 배열 B의 각각 1번째 열과 25번째 열의 98개 요소들을 한꺼번에 송신하는 메시지 벡터화가 적용되어 있다.

4. 성능 분석

이 장에서는 HPF 프로토타입 컴파일러인 PPTran의 성능을 분석한다. 즉, 벤치마크 프로그램을 선정하여 PPTran에 의해 컴파일된 코드와 상용 HPF 컴파일러인 pghpf에 의해 생성된 코드를 Cray T3E상에서 실행하여 그 결과를 비교 평가한다. PPTran은 SPMD 모델의 Fortran 77 부분으로 구성된 노드용 소스코드를 생성하기 때문에 다시 Fortran 77 컴파일러를 이용하여 실행 코드를 생성해야 하는 반면에, pghpf는 SPMD 모델의 노드용 실행 코드를 바로 생성한다[22]. 프로그램을 pghpf를 이용하여 실행시킬 때는 컴파일시 최적화 옵션으로 reduction 등을 병렬화 시키는 것과 연산·통신 중첩 옵션을 명시한다.

Cray T3E 시스템은 128개의 DEC Alpha(900 MFLOPS) 칩으로 구성되어 있고, 프로세서당 128MByte의 메모리를 갖는 고수준 병렬 시스템이다. 운영체제는 UNIX를 기반으로 하고 있으며, 프로그래밍 언어는 Fortran 77, Fortran 90, pghpf 등이 제공되고 메시지 패싱 라이브러리는 PVM-3, MPI, SHMEM이 제공된다.

성능 평가용 프로그램으로는 PARKBENCH의 HPF 컴파일러 벤치마크 프로그램[23]중에서 AA.HPF와 SH.HPF를 이용한다. AA.HPF는 프로세서간 통신이 필요없는 FORALL 문의 수행 능력을 측정할 수 있는 프로그램으로 BLOCK 분할과 CYCLIC 분할의 경우에 대하여 실행시킨다. (그림 18)은 BLOCK 분할의 경우로서, 21라인부터 26라인까지의 FORALL 내 할당문에서 lhs와 rhs의 배열 이름은 동일하나 데이터 종속성이 발생하지 않으며, 배열 PX의 2차원과 템플릿 d가 같은 분할로 정렬되어 있기 때문에 프로세서간 통신이

요구되지 않는다. 이 코드의 수행 결과는 (그림 20)에 도시되어 있는 것처럼, 배열크기가 적은 경우와 프로세서 수가 4와 8인 때의 배열 크기가 100,000인 경우를 제외하고는 PPTran에서 생성된 코드가 pghpf에 의해 생성된 코드보다 우수한 성능을 보인다. 배열 크기가 적은 경우의 PPTran 성능이 저조한 이유는 데이터 분할과 연산 분할 기능이 실행시간 라이브러리로 구현되어 있기 때문에 그에 따른 오버헤드로 분석되고, 배열의 크기가 100,000인 경우의 매우 저조한 성능에 대한 이유는 시스템 구조와 관련이 있을 것으로 판단된다. 그러나 프로세서 수가 4와 8과 같이 적은 경우에만 PPTran에서 생성된 코드의 성능이 나쁘고, 프로세서 수가 증가하면 성능이 좋아지기 때문에 일반적으로 많은 수의 프로세서를 이용하는 병렬 프로그래밍 환경에서 큰 문제는 없는 것으로 생각된다. CYCLIC 분할 경우의 수행결과는 (그림 21)에 보인 것처럼 모든 경우에 PPTran에서 생성된 코드의 성능이 우수하다. SH.HPF는 (그림 19)와 같이 바로 이웃 프로세서간에 shift형 통신을 발생시키는 FORALL 문의 수행 능력을 측정할 수 있는 프로그램이다. 21라인에서 23라인까지의 할당문을 살펴보면 프로세서 0, 1, 2는 바로 이웃하고 있는 프로세서 2, 3, 4로부터 각각 배열 U의 6개의

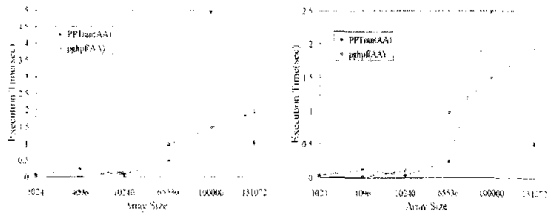
```
1          program AA
2          parameter ( N = 100000 )
3          real PX(13,N), Q
4          real DM22, DM23, DM24, DM25, DM26, DM27, DM28, C0
5          integer i, k, it, nit
6          data nit /23/
7  CHPF$     processors p(4)
8  CHPF$     template d(N)
9  CHPF$     distribute d(block) onto p
10 CHPF$     align PX(*,I) with d(I)
11           DM22 = 1.0
12           DM23 = 2.5
13           DM24 = 0.11
14           DM25 = 0.25
15           DM26 = 2.6
16           DM27 = 0.2
17           DM28 = 10.0
18           C0   = 0.5
19           FORALL(I=1:13,J=1:N) PX(I,J) = I+1.E-07*J
20           DO IT=1,NIT
21                 FORALL ( i = 1:N )
22     &  PX(1,i) = DM28 * PX(13,i) + DM27 * PX(12,i) +
23     &            DM26 * PX(11,i) + DM25 * PX(10,i) +
24     &            DM24 * PX(9,i)  + DM23 * PX(8,i)  +
25     &            DM22 * PX(7,i)  + C0 * (PX(5,i)   +
26     &            PX(6,i))        + PX(3,i)
27           ENDDO
```

(그림 18) AA.HPF 프로그램(BLOCK)
(Fig. 18) AA.HPF program

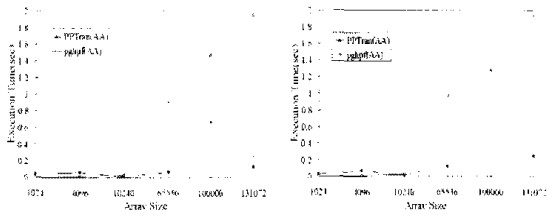
요소들을 가져와야 하며 lhs와 rhs간에 데이터 종속성이 발생하지 않으므로 루프 이전에 벡터화 통신코드를 삽입할 수 있다. 이 코드의 수행 결과는 (그림 22)에 보인 것처럼, 모든 경우에 PPTran에서 생성된 코드가 pghpf에 의해 생성된 코드 보다 우수한 성능을 보이고 있으며, 특히, 프로세서 수가 증가할수록 더욱 좋은 성능을 나타내고 있다. SH.HPF 경우의 성능의 우수함은 LCD 종속성을 바탕으로 한 통신 분석을 통하여 메시지를 벡터화한 점과 연산·통신 중첩 기법 이용이 효과를 본 것으로 판단된다.

```
1          program SH
2          integer N
3          parameter ( N = 1000000)
4          real, dimension(N) :: X, Y, Z, U
5          real Q, R, T
6          integer k
7  CHPF$   processors p(4)
8  CHPF$   template d(18432)
9  CHPF$   distribute d(block) onto p
10 CHPF$   align (:) with d(:) :: X, Y, Z, U
11                 R = 1.5
12                 T = 3.77
13                 U = 7.0
14         X = 8.0
15         Y = 9.0
16         Z = 10.0
17         do it=1,5
18                 forall (k=1:N-6)
19     &   X(k)=     U(k) + R*( Z(k) + R*Y(k)) +
20     &          T*( U(k+3) + R*( U(k+2) + R*U(k+1)) +
21     &          T*( U(k+6) + Q*( U(k+5) + Q*U(k+4))))
22         enddo
```

(그림 19) SH.HPF 프로그램
(Fig. 19) SH.HPF program

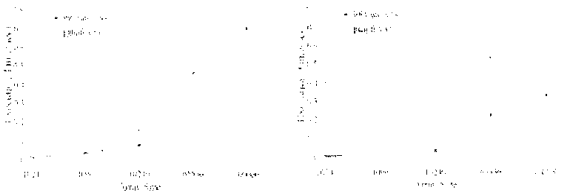


PE = 4 PE = 8

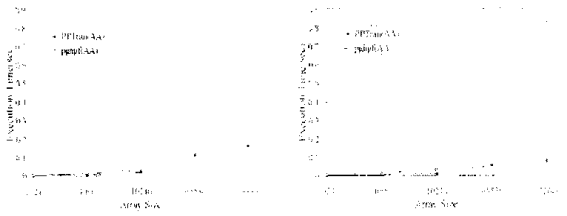


PE = 16 PE = 32

(그림 20) AA.HPF의 수행시간(BLOCK)
(Fig. 20) Execution time of AA.HPF(BLOCK)

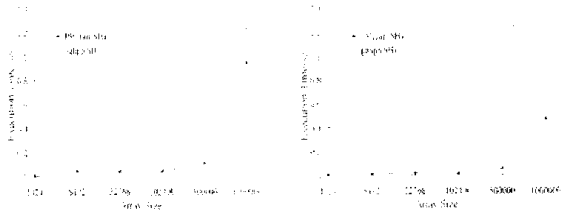


PE = 4 PE = 8

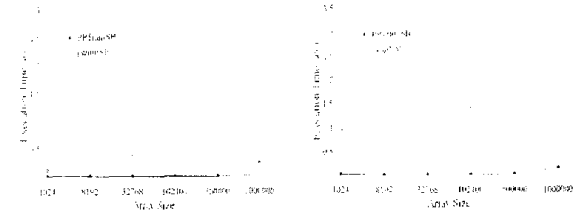


PE = 16 PE = 32

(그림 21) AA.HPF의 수행시간(CYCLIC)
(Fig. 21) Execution time of AA.HPF(CYCLIC)



PE = 4 PE = 8



PE = 16 PE = 32

(그림 22) SH.HPF의 수행시간(BLOCK)
(Fig. 22) Execution time of SH.HPF(BLOCK)

6. 결 론

분산 메모리 컴퓨터는 강세성, 확장성과 높은 성능 등의 많은 장점에도 불구하고 병렬 프로그래밍의 난이도와 기종간 프로그램의 호환성 결여로 기대만큼 성공적으로 활용되지 못하고 있다. 이에, 시스템 독립적이고 쉽게 프로그래밍 할 수 있는 데이터 병렬 언어에 대한 연구가 최근까지 활발히 진행되어 왔다. 대표적인 데이터 병렬 언어인 HPF 컴파일러는 사용자가 정의한 정보를 이용하여 데이터와 연산을 분할하여 프로세서에 할당하고 메시지 패싱 삽입 기능을 제공함으로써, 프로그램 작성자에게 선역 주소 공간을 이용하여 범렬 프로그램을 쉽게 개발할 수 있는 기반을 제공한다.

본 논문에서는 HPF 프로토타입 컴파일러인 PPTran을 구현하였고 성능을 검증하였다. PPTran에서는 데이터 병렬 언어의 핵심 기술이라 할 수 있는 데이터 분할, 연산 분할 및 통신 분석 기술이 구현되어 있다. 특히, 벡터화 통신 기법과 연산 통신 중첩 기법 적용을 통한 최적화를 도모함으로써 보다 성능이 좋은 노드용 SPMD 프로그램을 생성하도록 하였다. 또한, 연산 분할 및 통신 분석 기술을 실행시간 라이브러리로 구현하였기 때문에 PPTran에 의해 생성된 코드의 성능은 상용 HPF 컴파일러인 pghpf 생성 코드의 성능에 비해 PARKBENCH 마크 시험에서 전반적으로 우수함을 보여 주었다. 특히, PPTran 생성 코드는 데이터 양이 많아지고 프로세서 수가 증가할수록 성능이 현저하게 향상됨을 보이고 있어, PPTran의 성능과 확장성이 뛰어남을 나타내고 있다.

향후 HPF subset의 일부분만 구현된 PPTran의 기능을 추가 확장하고, 루프 분할과 통합, 루프 교환, strip mining, 메시지 병합(Message Coalescing), 메시지 통합(Message Aggregation) 등의 병렬 최적화와 통신 최적화 연구를 수행하게 되면 PPTran은 한층 유용한 HPF 컴파일러가 될 수 있을 것으로 생각된다.

참 고 문 헌

[1] Leasure, B., Ed. "PCF Fortran : Language Definition, version 3.1," The Parallel Computing Forum, Champaign, Il, Aug. 1990.

[2] ANSI X3j3/s8.115. Fortran 90. June 1990.

[3] D. Callahan and K. Kennedy, "Compling Programs for Distributed-memory Multiprocessors," Journal of Supercomputing, Oct. 1988.

[4] M. Gerndt, "Updating Distributed Variables in Local Computations, Concurrency : Practice & Experience," Vol.2, No.3, Sept. 1990.

[5] C. Koelbel and P. Mehrotra, "Compiling Global Name Space Parallel Loops for Distributed Execution," IEE Trans. Parallel Distributed System, Vol.2 No.4, Oct. 1991.

[6] Thinking Machines Corp., CM Fortran Reference Manual, Ver. 5.2, Sept. 1989.

[7] S. Hiranandani, Ken Kennedy, and C. Tseng, "Compiling Fortran D," Communications of the ACM, Vol.35, No.8, Rice University, Aug. 1992.

[8] C. Tseng, "An Optimizing Fortran D Compiler for MIMD Distributed Memory Machines," Ph.D. Thesis, Dept. of Computer Science, Rice Univ., 1993.

[9] J. Wu, J. Saltz, S. Hiranandani, and H. Berryman, "Runtime Compilation Methods for Multicomputers," In Proceedings of the 1991 International Conference on Parallel Processing, Aug. 1991.

[10] Applied Parallel Research, "Forge 90 Distributed Memory Parallelizer : User's Guide," 1992.

[11] B. Chapman, P. Mehrotra, and H. Zima, "Programming in Vienna Fortran," Scientific Programming, Vol.1, Fall 1992.

[12] High Performance Fortran Forum, High Performance Fortran Language Specification, Ver 1.1, Technical Report CRPC-TR9225, Center for Research on Parallel Computation, Rice University, Tex., 1994.

[13] http://www.crpc.rice.edu/HPFF/hpf2/index.html

[14] P. J. Hatcher, "The Impact of High Performance Fortran," IEEE Parallel & Distributed Technology, Vol.2, No.3, Fall 1994.

[15] M. Snir, S. Otto, S. Huss-Lederman, D. Walker and J. Dongarra, "MPI : The Complete Reference," The MIT Press

[16] SERI, "Development of Optimizing Tool for Parallel Programming," Research Report, KIST, December 1995.

[17] 문경덕, 김태근, 반난주, 김중권, 유여백, "PVM을 이용한 HPF 병렬 프로그래밍 번역기 구현에 관한 연구", 병렬처리시스템연구회지, Vol.13, No.1, 1995.

[18] Taegeun Kim, Kyeongdeok, Nanjoo Ban, and Jungkwon Kim, "PPTRan : Source to Source Translator for High Performance Fortran," Parallel Algorithm and Applications, Vol.9, 1996.

[19] High Performance Fortran Forum, "HPF-2 Scope of Activities and Motivating Application (Version 0.8)," Nov. 1994.

[20] M. Wolfe, "Tiny A Loop Restructuring Research Tool," Oregon Graduate Institute of Science and Technology, April 1994.

[21] J. M. Stichnoth, D. O'Hallaron, and T. R. Gross, "Generating Communication for Array Statements : Design, Implementation, and Evaluation," Journal of Parallel and Distributed Computing, Vol.21, No.1, 1994.

[22] Portland Group, pghpf USER'S GUIDE.

[23] Tomasz Haupt, Shravanthi G. Reddy and Giriraj Vengurlekar, "Low Level HPF Compiler Benchmark Suite," NPAC Technical Report SCCS-735, Aug. 1995.

김 중 권

e-mail : jkkim@seri.re.kr

1973년 서강대학교 전자공학과(학사

1976년~1981년 KIST 전산센터

1981년~1982년 대통령 경호실

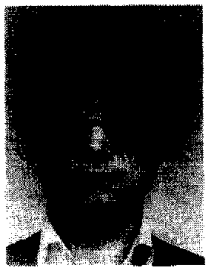
1985년 연세대학교 전자계산학과(석사)

1990년~현재 아주대학교 전자계산학과 박사과정

1982년~1998년 5월 시스템공학연구소 책임연구원

1998년~ 현재 ETRI 책임연구원

관심분야 : 병렬 컴파일러, 이기종 컴퓨팅, 병렬처리, 병렬 알고리즘



홍 만 표

e-mail : mphong@madang.ajou.ac.kr

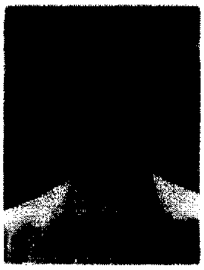
1981년 서울대학교 계산통계학과(학사)

1983년 서울대학교 계산통계학과(석사)

1991년 서울대학교 전산과학과(박사)

1985년~현재 아주대학교 정보 및 컴퓨터공학부 교수

관심분야 : 병렬처리, 광상호연결망, 시스템 성능분석



김 동 규

e mail : dkkim@madang.ajou.ac.kr

1973년 서울대학교 응용수학과 졸업(학사)

1979년 서울대학교 자연과학대학원 전자계산학과 졸업(석사)

1984년 미국 Kansas State University 전자계산학과 졸업(박사)

1973년~1977년 한국과학기술연구소 연구원

1977년~1979년 한국전자기술연구소 선임연구원

1979년~현재 아주대학교 정보 및 컴퓨터공학부 교수

관심분야 : 컴퓨터 네트워크, 정보통신 Security, 통신 프로토콜 엔지니어링