ハイパフォーマンス
コンピューティング 57−18
（1995. 8. 25）

HPFコンパイラにおける並列化手法

石崎 一明　　小松 秀昭
ishizaki@trl.ibm.co.jp

日本アイ・ビー・エム（株）東京基礎研究所
〒242 大和市下鶴間 1623-14

概要

本稿では、分散メモリ並列計算機用の HPF コンパイラにおいて、データ並列性を抽出するためにインデックス空間分割グラフを用いたループ並列化手法を提案する。インデックス空間分割グラフは、インデックス空間を分割するプロセッサと、プロセッサへ分散されたインデックスの範囲を表す。このグラフ上のノード間の関係によって、並列化を行う。このグラフを用いたコード生成方法を述べ、SP 上での性能評価を示す。

A Loop Parallelization Technique for HPF Compilers

Kazuaki Ishizaki,　Hideaki Komatsu
ishizaki@trl.ibm.co.jp

Tokyo Research Laboratory, IBM Japan, Ltd.
1623-14, Shimo-tsuruma, Yamato, Kanagawa, 242 Japan

Abstract

This paper presents a loop parallelization technique to extract data parallelism using a Index Space Partitioning Graph (ISPG). The ISPG shows processors and the loop iteration sub-space allocated to processors. The algorithm parallelizes loop nests using the relationship of nodes in the ISPG. We give how the SPMD code can be generated using the ISPG. We also show performance results on the SP.

1 はじめに

HPF[1]は、データ並列用に配列の配置分割を指示することを可能にしたFortran90の拡張言語である。プログラマによってデータ分割が記述されたプログラムからデータ並列を抽出するためには、配列の分割情報からコンパイラがループのインデックスをプロセッサに分散し、ループを並列化することが必要である。

本稿では、分散メモリ並列計算機の並列化のためにインデックス空間分割グラフ（Index Space Partitioning Graph: 以下 ISPG）を提案し、すでに提案している communication dependence vector[2] と ISPG を用いたループの並列化の手法を示す。また、ISPG を用いた Single-Program Multiple-Data (SPMD) プログラムのコード生成方法を示す。

以下、2章で従来のループ並列化の問題点を述べる。3章では、並列化のために導入する ISPG を定義し、ISPG を用いた並列化方法とループの SPMD コード生成方法を述べる。4章では、具体的な並列化の例を示す。5章では、本稿で提案したアルゴリズムを適用し、サンプルプログラムを実行した結果を示す。6章では関連する研究について述べ、7章で本稿をまとめる。

2 従来のループ並列化の問題点

従来の分散メモリ並列計算機のコンパイラのループ並列化では、通信のベクトル化によってループ内部から通信を除去し、通信なしで並列実行できるループを検出することで行うことが重要である。従来提案されているループ並列化手法[3]では、以下の場合効果的な方法が示されていない。

1. データ依存関係によって内部に通信が残る imperfect ループネスト
2. ループ内の複数の代入文から決定される分散されたループインデックスが異なるループネスト

実アプリケーションでは、これらの2つの imperfect ループが多く見られる。以下、具体例を用いて問題点を説明する。

2.1 内部に通信が存在するループネスト

図 1のプログラムを用いて説明する。以下では、説明を簡単にするために loop distribution 可能な例を用いるが、loop distribution 不可能なループも同様に考えることができる。まず、ループインデックスを各プロセッサに分散する。このための手法として、本稿では owner-computes-rule[4]を適用する。line 9 の左辺の配列 A の分散に基づいて DO K と DO I のループインデックスを、line 13 の左辺の配列 B の分散に基づいて DO J ループのインデックスを分散する。

DO K ループの内部に、配列 A と B に関する true dependence が存在するので、オペランド A(J-1,K) と B(J-1)の通信のベクトル化は DO K ループの前で行うことができない。従って、line 12 の DO J ループの前に、配列 A と B のベクタ通信を生成する。この結果、DO K ループの中に通信が残るので、ループ内から通信を除去する従来の並列化手法では、2 重ループ全体で並列化は不可能である。

```
1:*HPF$ PROCESSORS P(2,2)
2:*HPF$ DISTRIBUTE A(BLOCK,BLOCK) onto P
3:*HPF$ ALIGN B(I) with A(I,*)
4:*HPF$ ALIGN C(J) with A(1,J)
5:*HPF$ INDEPENDENT ,NEW(B)
6:      DO 10 K = 2, 99
7:        C(K)=K
8:        DO 20 I = 1, 99
9:          A(I,K) = I+K*100
10:  20    CONTINUE
11:        B(1) = K
12:        DO 30 J = 2, 99
13:          B(J) = B(J-1) + A(J-1,K)
14:  30    CONTINUE
15:  10  CONTINUE
```

図 1:プログラム例(1)

2.2 インデックス範囲が異なるループネスト

図 2のプログラムでは、line 4 と line 5 の代入文の左辺の配列分散から決まる各プロセッサのループインデックス I が異なる。SPMD プログラムでは、分散されたインデックス範囲はプロセッサ毎に一意に決定されなければならない。ループの内部に複数の代入文が存在し、それぞれの代入文の左辺が実行される範囲が異なるとき、並列化が不可能である。

```
1:*HPF$ PROCESSORS P(2,2)
2:*HPF$ DISTRIBUTE (BLOCK,BLOCK) onto P::A,B
3:      DO 10 J = 2, 99
4:        DO 10 I = 2, 98
5:          A(I,J) = B(I,J)
6:          B(I+1,J)=A(I-1,J)+A(I+2,J)
7:  10  CONTINUE
```

図 2:プログラム例(2)

このようなループの並列化を行うための方法として、配列をアクセスするインデックスを変形する computation alignment[5]、ループを変形する loop alignment[3]が提案されている。

computation alignment はインデックスの変形のための解析が複雑であり、loop alignment は生成されたコードが複雑であるという問題点がそれぞれある。

3 インデックス空間分割グラフ

本章では、インデックス空間分割グラフ(ISPG)を定義しその上での通信について述べる。さらに、ISPG を用いた並列化手法と SPMD コード生成方法を示す。

データ並列性は、以下の2つによって決定される。

1. ループインデックスをプロセッサ集合にマッピングして分散する
2. 分散されたインデックスを並列実行するプロセッサ間の同期として通信が発生しない

この2つを統合して扱うために ISPG を導入し、ISPG のノードが示すインデックス範囲とプロセッサ集合の関係に着目して並列化を行う。

本稿では、以下のように従来の問題を解決する。

1. ループ内に通信が残る場合、発生する通信が並列実行を妨げないことを ISPG 上で検出する。
2. 各代入文から決まる分散されたインデックス範囲から、ISPG を用いて各プロセッサで実行されるインデックスを一意に決定し、正しい実行を保証する。

3.1 ISPG の定義

本節では、ループ並列化に用いるグラフである

ISPG を定義する。ISPG はインデックスの分散を決定する配列から生成され、プロセッサ情報と分散されたインデックス情報を示す。ISPG を以下に定義する。

定義 1:ISPG は以下のノードとアークで構成される。

ノード：代入文を実行するプロセッサ集合 $P=\{p_1,p_2,p_3,\ldots\}$、そのプロセッサへ分散されたループインデックス名とインデックス範囲 L=[lb:ub]を持つ。

アーク：プログラムの DO ループの開始にあたるノードの分割、DO ループの終了にあたるノードの合流を示す。

定義 2:ISPG 上で 2 つのノードが独立とは以下の 2 つの条件を満たすことである。

1. 2 つのノードが持つインデックス名が同じである。
2. 2 つのノードが持つプロセッサ集合を P_1, P_2 としたとき、$P_1 \cap P_2 = \varnothing$ である。

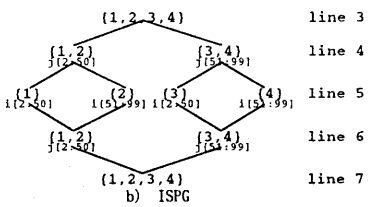
定義 2 は、同一ループインデックスを実行する際に 2 つのノード間で干渉するプロセッサがないことを示す。以下の条件が満たされれば、インデックスに関して並列実行可能である。

条件 1: プロセッサが独立なノードに属するならば、ループインデックスを分散並列実行可能である。

図 3に ISPG の例を示す。

```
1:*HPF$ PROCESSORS P(2,2)
2:*HPF$ DISTRIBUTE A(BLOCK,BLOCK) onto P
3:       DO 10 J = 2, 99
4:         DO 20 I = 2, 99
5:           A(I,J) = A(I+1,J)
6:   20    CONTINUE
7:   10  CONTINUE
```

a) ソースプログラム



b) ISPG

図 3: ISPG の例

図 3のグラフの意味を以下に述べる。グラフのトップでは、プログラムは配列 A を持つプロセッサ集合{1,2,3,4}によって実行されている。line 4 でループインデックス J は、配列 A(I,J)の 2 次元目を持つプロセッサ集合{1,2}と{3,4}へ分散される。よって、ループインデックス J は[2:50],[51:99]に分散される。このとき、2 つのノードは独立である。

line 5 でループインデックス I は、配列 A(I,J)の 1 次元目を持つプロセッサ集合へ分散される。プロセッサ集合は{1,2}は{1}と{2}に、{3,4}は{3},{4}に分割される。また、ループインデックス I は[2:50],[51:99]に分散される。このとき、4 つのノードは独立である。よって、line 5 の代入文はプロセッサ{1},{2},{3},{4}で並列実行可能である。

3.2 communication dependence vector の定義

我々は、communication dependence vector (CDV) を用いてループ内から通信を除去するループ並列化アルゴリズム[2]を提案している。CDV は分散された配列間に存在するデータ依存関係によって発生する通信の有無を示すオペランドが持つベクタである。CDV は、true communication dependence vector (tCDV) $\vec{c}_t$ と anti communication dependence vector (aCDV) $\vec{c}_a$ の 2 つからなり、イタレーション空間上の次元で示す。

$\vec{d}_t$ を true dependence による dependence vector、$\vec{d}_a$ を anti dependence による dependence vector とする。また、$\vec{b}$ はプロセッサ間通信の有無をイタレーション空間上の次元で 1/0 で示したベクタである。

$$\vec{c}_t=(c_{t1},c_{t2},\ldots c_{tn}),\ c_{ti}=\begin{cases}d_{ti}\ (\text{if } b_i=1)\\ 0\ \ (\text{if } b_i=0)\end{cases}$$

$$\vec{c}_a=(c_{a1},c_{a2},\ldots c_{an}),\ c_{ai}=\begin{cases}d_{ai}\ (\text{if } b_i=1)\\ 0\ \ (\text{if } b_i=0)\end{cases}$$

オペランドの tCDV が非零の場合はベクタパイプライン通信を生成し、aCDV が非零の場合はベクタプリフェッチ通信を生成する。tCDV と aCDV がともに非零の場合はベクタ通信を生成できないが、並列化を行うループネストの対象を小さくし、再度並列化を試みる。並列化されたループの前後に、プロセッサ間のデータ依存関係を保つための同期としてのベクタ通信を生成する。

3.3 ISPG における通信

ISPG のノードはプロセッサ集合も示す。通信をそれを行うプロセッサ対で表現することによって、ISPG 上でのノード間の関係として表現する。

ISPG の独立なノードは分散されたインデックスを並列に実行可能なことを表す。独立なノード上のプロセッサ間に同期となる通信が発生するならば、プロセッサ間の並列実行を妨げる。これは、ISPG の重要な性質である並列実行可能なプロセッサ集合の表現をそこなうことになる。

ループが並列実行できるために、条件 1 を満たした上で、ISPG 上の通信は以下の条件を満たさなければならない。

条件 2:ISPG の独立なノードに属するプロセッサ集合が並列実行できるために、通信を行うプロセッサ対は ISPG のノード上のプロセッサ集合内に閉じていなければならない。

図 3の例では、通信がプロセッサ 2 から 1 へ、プロセッサ 4 から 3 へ発生する。よって、それぞれのプロセッサが並列実行可能であるためには、通信を行うプロセッサ対を含むノード（{1,2}と{3,4}、または{1,2,3,4}）まで通信を移動できなければならない。

ISPG は分散されたループインデックスと実行するプロセッサの関係を表すだけである。ISPG 上で移動可能な通信の位置は、データ依存解析の結果を用いて別に保証する必要がある。

CDV はデータ依存によって発生する通信を表す。よって CDV を用いた通信解析を適用して、以下の方法で ISPG 上の通信の移動がデータ依存関係を満たすこ

とを保証する。

ISPG のトップノードから、ループ内のオペランドが持つ CDV の値に従って、以下の手順で外側のループネストから通信解析を行う。

1. aCDV と tCDV がともに零で通信が発生する場合、着目している ISPG のノードに対応するループネストで、ベクタプリフェッチ通信可能である。
2. aCDV が非零の場合、着目している ISPG のノードに対応するループネストで、ベクタプリフェッチ通信可能である。
3. tCDV が非零の場合、着目している ISPG のノードに対応するループネストで、ベクタパイプライン通信可能である。
4. aCDV と tCDV がともに非零の場合、または imperfect ループネスト内に true dependence が存在するとき、着目しているループネストのレベルを 1 つ減らす。新たに CDV を求めて、上記の通信解析を繰り返す。

通信解析の結果、ループネスト内に 1、2、3 によって生成されたベクタ通信が存在する場合でも、通信を行うプロセッサ対が ISPG のノードのプロセッサ集合内に閉じているならば、ベクタ通信を含むループネスト全体で並列実行可能と判断される。

3.4 ISPG を用いた並列化とコード生成

本節では、ループネスト内に複数の代入文が存在する場合の並列化と SPMD コード生成について述べる。

ループネスト内の複数の代入文によってインデックスの分割範囲が決定される場合、代入文毎に ISPG を生成する。このループネストが並列実行可能であるためには、n 個の代入文から生成された ISPG $G_1, G_2, \ldots, G_n$ において、以下のプロセッサ集合に関する条件と、ループインデックスに関する条件が同時に満たされなければならない。

条件 3:

1. 各 ISPG 間でループインデックス i に対応するノードが持つプロセッサ集合 $P_{i1}, P_{i2}, \ldots, P_{in}$ が、全て等しいか、サブセットの関係にある。

$$\forall P_{ix}, P_{iy} : \left(P_{ix} = P_{iy}\right) or \exists P_{ix} : \left(P_{iy} \subseteq P_{ix}\right), \text{ただし} 1 \le x, y \le n$$

2. 各 ISPG 間でループインデックス I に対応するノードで、ループインデックスの実行範囲 $L_{i1}, L_{i2}, \ldots, L_{in}$ が等しい。

$$\forall L_{ix}, L_{iy} : \left(L_{ix} = L_{iy}\right), \text{ただし} 1 \le x, y \le n$$

以下では、それぞれの条件を満たすための並列化手法について述べる。

3.4.1 ループインデックスを実行するプロセッサ集合が異なる場合

1. の条件が満たされない場合、ループの直前で各 ISPG が条件を満たすように代入文の左辺の配列を再分散する。

1. の条件が満たされるとき、対応するノードのプロセッサ集合がすべて等しい場合には何もしない。対応するノードのプロセッサ集合に等しくないものが存在するならば、SPMD コードの生成時に配列を所有するプロセッサ上でのみ実行されるように IF 文を生成する。以後、このために挿入される IF 文をプロセッサガードと呼ぶ。

3.4.2 ループインデックスが異なる場合

2. の条件が満たされる場合、SPMD コードが実行するループインデックス範囲は、ノードが持つ唯一のループインデックス範囲 L_i を用いる。

2. の条件が満たされない場合、各プロセッサが実行する分散されたループインデックス範囲を等しくするために以下の方法のうちの 1 つを選ぶ。

1. ループインデックス $L_{i1}, L_{i2}, \ldots, L_{in}$ のずれ幅が、袖 [6] の大きさより小さい場合はそのまま実行し、袖に値を書き込む。これはイタレーションのオーバラップ分散による owner-computes-rule の緩和になる。オーバラップ分散によって必要になる右辺の配列の通信領域を再計算し、袖の大きさをインデックスのずれ幅分大きくする。
2. ループインデックス L_i として $L_{i1}, L_{i2}, \ldots, L_{in}$ の和をとる。その後、L_i のサブセットとなるインデックスを持つ代入文には、配列を所有するインデックスのみに実行を制限するための IF 文を挿入する。以後、このために挿入される IF 文をインデックスガードと呼ぶ。
3. ループの直前で各 ISPG が 2. の条件を満たすように代入文の左辺の配列の分割を一時的に再分散する。この場合インデックスガードは必要ない。

4 実際の並列化の例

4.1 内部に通信が存在するループネストへの適用例

本節では、図 1の例を用いて並列化の具体例を示す。

図 4は、それぞれ配列 A、B、C から生成された ISPG を示す。配列 A によってループインデックス K と I が、配列 B によってループインデックス J が、配列 C によってループインデックス K が分散される。このとき、3 つの ISPG の間で対応するループインデックスは K のみである。

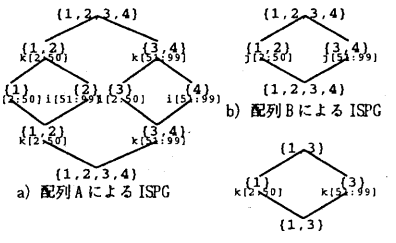


図 4: 図 1のプログラムから生成される ISPG

ループインデックス K に対応するノードが持つプロセッサ集合は、A から決まるプロセッサ集合 {1,2} と

C から決まるプロセッサ集合{1}、同様に{3,4}と{3}は、それぞれサブセットである。よって SPMD コード生成の際に配列 C を左辺に持つ代入文について、プロセッサ 1 と 3 に実行を制限するプロセッサガードを挿入する。

DO K ループのレベルでは、imperfect ループの内部に true dependence が存在するので、3.3節の手順に基づいて、1つ内側のループネストである DO J ループで解析を行う。このとき、line 13 の右辺オペランドでは、配列 A は aCDV、tCDV が零で通信が発生し、配列 B は tCDV が非零である。この結果、DO J ループのレベルでは、A についてベクタプリフェッチ通信を、B についてベクタパイプライン通信を生成する。

上記の通信を行うプロセッサ対は、プロセッサ 2 から 1、プロセッサ 4 から 3 である。図 4の ISPG 上で、この通信はループインデックス K と J を実行するノードが持つプロセッサ集合内に閉じている。従って、2 重ループで並列実行可能である。最終的に生成される SPMD コードを図 5に示す。図中の ownID(C(K))は、C(K)を持つプロセッサ番号を返す関数である。

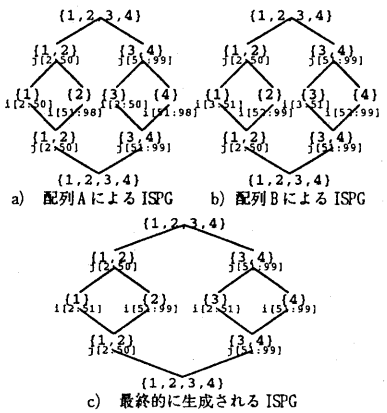
```
DATA myLB.K/  2,  2,  51,  51/
DATA myUB.K/ 50, 50,  99,  99/
DATA myLB.I/  1, 51,   1,  51/
DATA myUB.I/ 50, 99,  50,  99/
DATA myLB.J/  2, 51,   2,  51/
DATA myUB.J/ 50, 99,  50,  99/
DO K = myLB.K(my$P),myUB.K(my$P)
  IF (ownID(C(K))=my$P) THEN
    C(K) = K
  ENDIF
  DO I = myLB.I(my$P),myUB.I(my$P)
    A(I,K) = I+K*100
  END DO
  B(K)=K
  PREFETCH A(myLB.J(my$P)-1, K)
  PRERECV B(myLB.J(my$P)-1)
  DO J = myLB.J(my$P),myUB.J(my$P)
    B(J) = B(J-1) + A(J-1,K)
  END DO
  POSTSEND B(myUB.J(my$P))
END DO
```

図 5: 生成された SPMD プログラム

4.2 インデックスが異なるループネストの適用例

本節では、図 2の例を用いて並列化の具体例を示す。

図 6の配列 A、B から生成された 2 つの ISPG は、ループインデックス J と I の分散を示す。



a) 配列Aによる ISPG　b) 配列Bによる ISPG

c) 最終的に生成される ISPG

図 6: 図 2のプログラムから生成される ISPG

2 つの ISPG でループインデックス J に対応するノードでは、プロセッサ集合{1,2}, {3,4}はそれぞれ等しい。インデックス範囲もそれぞれ[2:50] ,[51:99]と等しい。

ループインデックス I に対応するノードにおいて、プロセッサ集合{1}, {2}, {3}, {4}はそれぞれ等しい。しかし、インデックス範囲が [2:50]と[3:51]、[51:98]と[52:99]と異なっている。この場合、3.4.2 節で述べた 3 つの方法のうちの 1 つを用いて並列化を行う。以下、2.のインデックスガードを適用したと仮定する。

line 5 では通信が発生しない。line 6 では、A(I-1,J)は tCDV が非零で、A(I+2,J) は aCDV が非零である。よって、DO J ループのレベルでベクタプリフェッチ通信とベクタパイプライン通信を生成する。結果として、2 重ループでパイプライン実行可能である。最終的に生成される SPMD コードを図 7に示す。

```
DATA myLB.J/ 2, 2,51,51/
DATA myUB.J/50,50,99,99/
DATA myLB.I/ 2,51, 2,51/
DATA myUB.I/51,99,51,99/
DATA LB.A/ 2,51, 2,51/
DATA UB.A/50,98,50,98/
DATA LB.B/ 3,52, 3,52/
DATA UB.B/51,99,51,99/
PREFETCH A(LB.B(my$P)+2,myLB.J(my$P):myUB.J(my$P))
PRERECV A(LB.B-1(my$P),myLB.J(my$P):myUB.J(my$P))
DO J = myLB.J(my$P), myUB.J(my$P)
  DO I = myLB.I(my$P), myUB.I(my$P)
    IF (LB.A(my$P)<=I and I<=UB.A(my$P)) THEN
      A(I,J) = B(I,J)
    ENDIF
    IF (LB.B(my$P)<=I and I<=UB.B(my$P)) THEN
      B(I+1,J) = A(I-1,J)+A(I+2,J)
    ENDIF
  ENDDO
ENDDO
POSTSEND A(LB.A(my$P),myLB.J(my$P):myUB.J(my$P))
```

図 7: 生成された SPMD プログラム

5 評価

本稿で提案した並列化アルゴリズムを、サンプルプログラムに適用し実行することによって評価する。IBM 社の RISC システム/6000 SP thin ノードの 16 台構成にハイパフォーマンススイッチを接続したシステム上で実行する。プロセッサ台数を変化させたときの実行時間の比を測定することで並列化の効果を示す。今回、実際の並列化は人手で行った。

5.1 内部に通信が存在するループネスト

図 8のプログラムでは、DO 100 と DO 200 のループネストで、2 次元配列 A,B の両次元方向にアクセスが行われている。このループに対して、従来のループ並列化方法で配列の 1 次元目を分割する ("Dist. 1st")、2 次元目を分割する ("Dist. 2nd") ものと、本稿で提案したアルゴリズムを適用することによって両次元で分割することを可能にした ("Dist. both") 2 重ループを並列実行可能にした結果を図 9に示す。本稿のアルゴリズムを適用した結果が、もっとも速度向上率がよいことを示している。

5.2 インデックスが異なるループネスト

図 10のプログラムでは、DO 10 のループネストは loop distribution 不可能である。また、このネスト

内の2つの代入文から決定される分散されたループインデックスは、ループインデックスIについて異なる。

このループに対して本稿で提案したアルゴリズムを用いて並列化を行い、3.4.2節で述べた3つの方法（1.がOverLap Area、2.がIndex Guardが、3.がRemap）を適用して実行し、3つの手法を比較した結果を図11に示す。

```
      PARAMETER(N=2400)
      REAL*8 A(N,N),B(N,N),S(N),T(N),U(N),V(N)
*HPF$ PROCESSORS P(2,2)
*HPF$ DISTRIBUTE (BLOCK,BLOCK) onto P::A,B
*HPF$ ALIGN (I) with A(I,*) :: S,T
*HPF$ ALIGN (I) with A(*,I) :: U,V
*HPF$ INDEPENDENT ,NEW(S,T)
      DO 100 J = 1, N
        DO 10 I = 1, N
   10     S(I) = A(I,J)
        DO 20 I = 1, N
   20     B(I,J) = S(I)*2
        DO 30 I = 1, N-1
   30     T(I) = B(I+1,J)*3
        DO 40 I = 1, N
   40     B(I,J) = S(I)+T(I)
  100 CONTINUE
*HPF$ INDEPENDENT ,NEW(U,V)
      DO 200 I = 1, N
        DO 110 J = 1, N
  110     U(J) = A(I,J)
        DO 120 J = 1, N
  120     B(I,J) = U(J)*2
        DO 130 J = 1, N-1
  130     V(J) = B(I,J+1)*3
        DO 140 J = 1, N
  140     B(I,J) = U(J)+V(J)
  200 CONTINUE
      END
```

図 8: サンプルプログラム

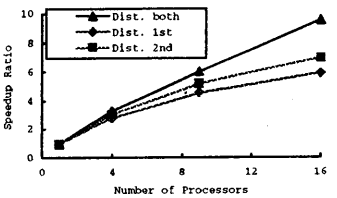


図 9: 図8の評価結果

```
      PARAMETER (N=2400)
      REAL*8 A(N,N), B(N,N)
*HPF$ PROCESSORS P(2,2)
*HPF$ DISTRIBUTE (BLOCK,BLOCK) onto P::A,B
      DO 20 K = 1, 50
        DO 10 J = 2, 99
          DO 10 I = 2, 98
            A(I,J) = B(I,J)
            B(I+1,J)=A(I-1,J)+A(I+2,J)
   10   CONTINUE
   20 CONTINUE
```

図 10: サンプルプログラム

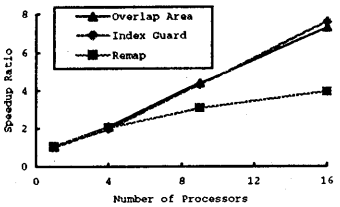


図 11: 図10の評価結果

6 関連研究

分散メモリ並列計算機のための言語において、ループを並列化する研究は数多く行われている。

言語によって、ループの並列化やインデックス分割、プロセッサ間の通信、同期を記述する VPP Fortran [6]が提案されている。この言語仕様は、並列実行するループ内部に通信や同期の記述を許すが、具体的な使用方法は明示していない。本稿では、分散されたインデックス、プロセッサと通信の関係を示す ISPG を用いて、通信を含んだループの並列化方法を示した。

データ分割の記述のみを行い、コンパイラが通信のベクトル化によってループ内部から通信を除去し、ループから並列性の抽出を行うものに Fortran-D[3]がある。本稿では、HPF 指示行による配列の分割方法から ISPG を生成し、従来並列化不可能なループネストを並列化するアルゴリズムを提案した。

袖とインデックスオーバラップに関しては、VPP Fortran と NEC の HPF 拡張[7]で提案されている。これらは言語の拡張として提案されており、コンパイラによる最適化の手段として議論されていない。

複数の代入文から決定される分割されたループインデックスが異なるループを並列化するために、配列をアクセスするインデックスを変形する computation alignment、ループを変形する loop alignment が提案されている。本稿では、ISPG を用いて分散されたループインデックスをそれぞれのプロセッサにおいて一意に決定する単純な手法を提案した。

7 まとめ

本稿では、ISPG を用いた並列化モデルと、ループ並列化方法を提案し、ISPG を用いた SPMD コード生成方法を示した。また、このアルゴリズムを用いて並列化したコードで実際の効果を示した。

8 謝辞

本研究にあたり日頃からご指導、ご助言いただく先進コンパイラグループの中谷登志男氏、郷田修氏、大澤暁氏、菅沼俊夫氏、小笠原武史氏に感謝いたします。

参考文献

[1] "High Performance Fortran Language Specification Version 1.1," High Performance Fortran Forum, 1994

[2] K. Ishizaki and H. Komatsu: "A Loop Parallelization Algorithm for HPF Compilers," 8th Workshop on Language and Compilers for Parallel Computing, pp.12.1-15, 1995

[3] C. W. Tseng : "An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines," PhD thesis, Rice University, CRPC-TR93291, 1993

[4] D. Callahan and K. Keneddy: "Compiling Programs for Distributed-Memory Multiprocessors," Journal of Supercomputing, Vol. 2, pp.141-169, 1988

[5] D. Kulkarni and M. Stumm: "Computational Alignment: A New, Unified Program Transformation for Local and Global Optimization," Technical Report, University of Toronto, CSRI-292, 1994

[6] 岩下、進藤、岡田: "VPP Fortran; 分散メモリ型並列計算機言語", 並列処理シンポジウム JSPP'94, pp. 153-160, 1994

[7] 末広、草野、蒲池、妹尾、田村、左近、渡辺、白戸: "HPF処理系における計算処理マッピング", 並列処理シンポジウム JSPP'95, pp.369-376, 1995