HPFにおけるデータ分散の図式表現と効果的計算分散法

佐 藤 真 琴†

計算分散とは与えられたデータ分散指示文に基づき，プログラム中のループ処理等の計算を各プロセッサに分散させる HPF コンパイラの処理フェーズである．従来の計算分散方法は対象とするプログラムやデータ分散に制限があったり，制限がなくても非効率なコードしか生成できなかったりした．本論文は，(1) 一般のループ制御式を持つ多重ループ，(2) 一般の規則的な HPF データ分散指示文が与えられた多次元配列，および，(3) 互いに異なるループ制御変数による一般の 1 次式を各次元の添字に持つ上記多重ループ中の上記配列への参照，を含むプログラムを対象とし，これを一般的な枠組みを使って表現する手法とこのプログラムに適用可能ないくつかの計算分散手法を提案する．これらの手法により，明示的コード，3 種類のテーブル参照法コード，および，ある特別な種類のプログラムに対する最適コードを得た．複雑な block-cyclic 分散を持つプログラムに提案手法を適用し，日立 SR8000 上で評価を行った．その結果，最適コードは従来の while ループによるテーブル参照コードよりも 10 倍以上高速であること，ならびに，do ループによる 2 つのテーブル参照コードは while ループによるテーブル参照コードよりも高速であることを示した．

Diagram Representation of Data Distribution in HPF and Effective Computation-partitioning Method

Makoto Satoh†

Computation partitioning (CP) is one of the phases of an HPF compiler which distributes a computation such as a loop in a program onto processors based on given data-mapping directives. Some conventional CPs have limitation in target programs or data mappings, and the others generate ineffective codes for general targets. In this paper, we deal with more general programs, including (1) multi-nested loops with general loop-control expressions, (2) multi-dimensional arrays to which general regular HPF data-mapping directives are applied, and (3) references to the arrays in the loops each of whose subscripts is a general linear expression of a loop-control variable that is different from the variables in the other dimensions. We propose a method representing these programs using a general framework and some CP methods applicable to the programs. As a result, we obtain three kinds of table-driven codes and an optimized code under a certain condition. We applied our methods to a program with a complicated block-cyclic distribution and evaluated the resulting codes on a Hitachi SR8000 supercomputer. We found that the optimized codes outperformed a conventional table-driven code with a while loop by more than 10 times and two table-driven codes with do loops ran faster than the conventional code.

1. はじめに

今日，スーパコンピュータは，多くの計算ノードをネットワークで接続した分散メモリ型計算機の構成をとることが多い．このような構成に対するプログラミング手法として，プログラマが MPI 等のメッセージパッシング・ライブラリを用いて並列プログラムを書く並列プログラミングが現在の主流である．並列プログラミングにおいてプログラマはデータ分散，計算分散，およびプロセッサ間通信という並列処理に関わるすべてを記述する．このため，非常に効率の良いプログラムの作成が可能になる一方で，プログラマの負担は重い．応用プログラムはますます複雑化・大規模化しているため，今後，このような並列プログラミングでは，プログラマに過大な負担がかかると予想される．

この課題を解決するために，我々は，プログラマとコンパイラが協調して並列化を行うことが重要であると考え，そのようなアプローチの 1 つとして High Performance Fortran（HPF）[6] に対するコンパイル技術の研究を進めている．HPF を使う場合，プログラマはデータ分散指示文等を逐次プログラムに挿入

† 株式会社日立製作所システム開発研究所
Systems Development Laboratory, Hitachi, Ltd.

し，コンパイラはその指示文を解析して計算分散やプロセッサ間通信の生成を行う，という風に役割を分担する．

HPF の特徴として複雑なデータ分散指示文がある．その 1 つは block-cyclic 分散である．プログラマはこれをループ繰返し間の負荷が不均一なループ中に出現する配列に指定することにより，配列の連続要素参照によるメリットと負荷分散によるメリットとのトレードオフを調整できる．2 つ目は ALIGN 指示文を用いた 2 レベルマッピングである．これは配列をいったんテンプレート（領域を確保しない仮想配列）にアラインし，このテンプレートをデータ分散することにより，元の配列をデータ分散する手法である．この手法により，より柔軟で一般的なデータ分散が可能になる．

一方，このような複雑なデータ分散指示文が指定されたプログラムに対してコンパイラが効率の良いコードを出力するのは非常に困難である．なぜなら ALIGN 指示文におけるアライン添字の係数（またはループストライドや配列添字中のループ制御変数の係数）がテンプレートに対するデータ分散指示文で指定されるブロックサイズの約数にならない場合，配列（またはループ）が均等に分散されなくなり，このことと block-cyclic 分散によって，元の配列要素とデータ分散後の配列要素との対応付けが複雑になるためである．

上記の 2 種類のデータ分散指示文が同時に指定されたプログラムに適用可能な計算分散方法としてテーブル参照法[4] がある．この方法は，線形添字を持つ配列参照を含むループに対して，データ分散後の配列要素の参照パターンは不均等であるがある周期ごとに同じパターンを繰り返すことを利用し，ある小さいループ繰返し範囲における参照パターンを 2 種類のテーブルに格納し，次にそれらのテーブルを用いて全ループ繰返し範囲における配列参照コードを生成する．本方法は上記複雑なプログラムに対して従来，最も高速であったと思われるが，配列参照時に間接参照オーバヘッドがつねにともなうという問題がある．また，ループのストライドが負の場合は扱われていない．

本論文は，上記の 2 つのデータ分散指示文が同時に指定された 1 次元配列を左辺に持つ文を含む 1 重ループに対して計算分散を行う一般的枠組みを提案する．この枠組みはループ制御変数と配列次元が 1 対 1 に対応するような d 次元配列・d 重ループに容易に拡張できる．この枠組みを用いることにより，ループストライドが負の場合にも適用可能で従来より高速なテーブル参照コードを導く．さらに，ある特別な条件が成り立つ場合に最適なコードが出力できることも示す．

表 1 記号
Table 1 Symbols.

A	データ分散される配列
P	論理プロセッサの配置形状を表す配列
T	配列 A がアラインするテンプレート
T_l	テンプレート T のある次元の下限値
b	データ分散におけるブロックサイズ
c	cyclic 分散におけるデータ割付けの繰返し回数
e	テンプレート T のある次元の寸法
$f_a i + f_b$	アライン添字．f_a, f_b：整数，変数，または式
g	T から標準テンプレートへの埋め込み写像
h	標準ループ空間 L^0 からループ空間 L への写像
$i_a i + i_b$	配列添字．i_a, i_b：整数，変数，または式
k	ループ制御変数に配列添字を対応させる写像
l, u, m	ループ下限値，上限値，ストライド
p	プロセッサ数
q	あるプロセッサの番号
n, r, s, v	AXIS_TYPE が NORMAL，REPLICATED，SINGLE，VANISHED となる次元数
$\mathcal{L}$	グローバル添字をローカル添字に変換する配列
ρ	標準テンプレートのローカルアドレス表現
π	$= f_a i_a m$
$\psi(i)$	$A(i)$ がマッピングされるプロセッサ番号

本論文の残りは以下となる．2 章では HPF 指示文を復習し，本論文が扱うプログラムパターンを定義し，3 章ではこのプログラムパターンを見通し良く表現するのに用いる図式を復習する．4 章では配列添字式—ループ制御式—データ分散の間の関係を図式で表現し，5 章ではこれを用いて計算分散公式を導き，6 章では特別な場合に対して最適コードを導く．7 章では本公式における添字参照の擬周期性を示してテーブル参照コードを与える．8 章では本論文で提案したコードを評価し，9 章では関連研究について述べ，10 章で本研究をまとめる．なお，**表 1** に本論文中で用いる記号を示す．

2. HPF 指示文と RDLS(1,1) パターン

本章では，HPF 指示文[6] を復習し，本論文において考察の対象とするプログラムパターンを定義する．

2.1 HPF によるデータ分散と計算分散

図 1 は HPF によるデータ分散と計算分散を説明した図である．HPF ではデータ分散は以下に述べる 2 段階で行われる（2 レベルマッピング）．

第 1 段階で配列 A は，テンプレートと呼ばれる実メモリを確保しない仮想配列 T にアラインされる☆．$A(i)$ が $T(f_a i + f_b)$ ☆☆にアラインすることをプログ

☆ HPF では A をテンプレートでない配列 B にアラインさせることもできる．しかし，結局は（アラインの連鎖により）あるテンプレート T にアラインすることになる．

☆☆ 以降，特別な場合以外は乗算記号を略す．

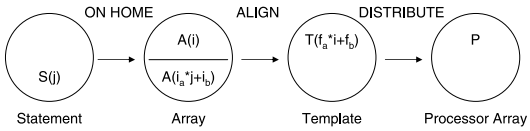

図 1 HPF によるデータ分散と計算分散
Fig. 1 Data mapping and comp. partitioning in HPF.

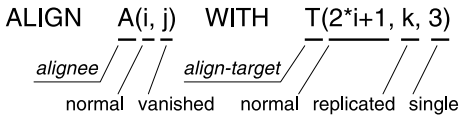

図 2 AXIS_TYPE の例
Fig. 2 Example of AXIS_TYPEs.

ラマは ALIGN 指示文で記述する．第 2 段階でテンプレート T は，プロセッサ形状を表す配列 P にマッピングされる．プログラマは BLOCK(b)，CYCLIC(b) 等の規則的データ分散または GEN_BLOCK 等の不規則データ分散を DISTRIBUTE 指示文で記述する．以上の 2 段階により，配列 A はプロセッサ P にマッピングされる．なお，本論文では規則的データ分散のみを扱う．

次に，計算を実行するプロセッサの決定（計算分散）は owner-computes rule（OCR）または ON HOME 指示文で行う．OCR では代入文 $S(j)$ はその左辺の配列要素 $A(i_a j + i_b)$ がマッピングされるプロセッサで実行される（図 1）．ON HOME 指示文では上記のような配列要素をプログラマが任意に指定できる．

2.2 AXIS_TYPE

図 2 は ALIGN 指示文とその AXIS_TYPE の例である．ALIGN 指示文における ALIGN 節直後の配列は alignee，WITH 節直後のテンプレート（または配列）は align-target と呼ばれる．align-target の各次元は以下の 3 つの AXIS_TYPE に分類される：

- NORMAL：その次元に含まれる変数は，alignee のある次元の添字と一致する．
- REPLICATED：その次元の添字は alignee のどの次元にも現れない．
- SINGLE：その次元の添字は定数である．

我々は 4.2 節で用いるため，alignee の各次元に対して以下の AXIS_TYPE を定義する：

- NORMAL：その次元の添字は align-target のある次元の添字に現れる．
- VANISHED：その次元の添字は align-target のどの次元にも現れない．

2.3 RDLS(1,1) パターン

本節では，本論文において考察の対象とする RDLS(1,1) パターンの定義を述べる．

```
1: !HPF$ PROCESSORS P(p)
2: !HPF$ TEMPLATE T(T_l : T_l + e − 1)
3: !HPF$ DISTRIBUTE T(CYCLIC(b)) ONTO P
4: !HPF$ ALIGN A(i) WITH T(f_a i + f_b)
5:       REAL A(A_l, A_u)
6:       do i = l, u, m
7:         A(i_a i + i_b) = ...
8:       end do
```

図 3 RDLS(1,1) パターンの例
Fig. 3 An example of RDLS(1,1) pattern.

定義 1 RDLS(1,1) パターン
以下の配列 A，ループ L，および代入文 S から構成されるプログラムパターンを RDLS(1,1) パターン（Regular data Distribution of 1-dimensional array and 1-tuple nested loop with Linear Subscript）と呼ぶ：

(1) 一般の規則的 HPF データ分散指示文が指定された 1 次元配列 A，
(2) 一般のループ制御式を持つ一重ループ L，
(3) ループ L の制御変数の線型式を添字とする A の要素を左辺に持つループ L 中の代入文 S． □

図 3 は RDLS(1,1) パターンの例である．条件 (1) は文 1 から 5 で，条件 (2) は文 6 で，条件 (3) は文 7 で指定される．なお，図 3 に現れる変数はコンパイル時に定数である必要はない．

RDLS(1,1) パターンは多くの科学技術計算プログラムの基本となる．なぜなら，d 重ループに含まれる代入文の左辺に現れる d 次元配列において，各次元が異なるループ制御変数を 1 つだけ含むというよく出現するパターンは次元ごとに RDLS(1,1) パターンになっているからである．

3. 図 式

図式とは集合および集合間の写像を見通し良く表現する 1 つの方法[2] である．以下は最も簡単な例である．

$$X \xrightarrow{\phi} Y \xrightarrow{\sigma} Z \quad \text{(exact)} \qquad (*)$$

図式 $(*)$ において，X，Y，Z は集合を，ϕ，σ は写像を表す．本論文では集合として整数全体（$\mathbf{Z}$）の部分集合，または，ある整数 a の剰余集合 $\mathbf{Z}_a = \{\bar{0}, \bar{1}, \ldots, \overline{a-1}\}$ を考える．

図式の特別なものに完全系列がある．図式 $(*)$ 中の ϕ を線形写像，$\mathrm{Im}\,\phi = \phi(X)$，$\mathrm{Ker}\,\phi = \{x \in X | \phi(x) = 0\}$ と定義する．ただし，Y は 0 を含むとする．このとき，もし，$\mathrm{Im}\,\phi = \mathrm{Ker}\,\sigma$ が成立するならば，図式

(∗) は完全系列と呼ばれ，図式の右に (exact) と書く．

X，Y，Z を群，f 等を準同型写像とするとき，完全系列の重要な例として以下がある：

$$0 \xrightarrow{i} X \xrightarrow{\phi} Y \xrightarrow{\sigma} Z \xrightarrow{j} 0 \quad \text{(exact)} \qquad (**)$$

ここで，完全系列の定義より ϕ は単射，σ は全射，$Y/\operatorname{Im}\phi \cong Z$ となる．また，$\sigma \circ \eta = id$ となる線形写像 $\eta : Z \mapsto Y$ があれば Y は直和分解される[2]：

$$Y \cong Y_1 \oplus Y_2. \qquad (***)$$

4. データ分散の図式表現

本章では ALIGN や規則的データ分散に対する DISTRIBUTE を完全系列を含む図式[2] で表現する．

4.1 テンプレートのデータ分散の図式表現

本節では，図 3 における，下限値 T_l と要素数 e を持つテンプレート T のデータ分散を図式で表現する．以降，テンプレートや配列と，それらの添字からなる集合とを同一視する．

4.1.1 標準テンプレートへの埋め込み

テンプレート T の要素数 e が図 3 中のプロセッサ数 p やブロックサイズ b で割りきれない場合，データ分散の取扱いが煩雑になる．そこで，以降では T のかわりにその要素数がこれらで割り切れるようなより大きな添字区間を扱い，T に特有な性質は別に取り扱う．

定義 2 標準テンプレート

$c = \lceil e/pb \rceil$ とする．このとき，$I_{pbc} = [0 : pbc - 1]$ を T に対する標準テンプレートと呼ぶ． □

次に，T のデータ分散は，T と I_{pbc} との関係，および，I_{pbc} のデータ分散の 2 つに分解できることを示す．まず，$pbc \geq e$ が成り立つので，T は標準テンプレート I_{pbc} の部分集合と見なせ，T から I_{pbc} の中へ写像

$$g : T \ni x \mapsto x - T_l \in I_{pbc} \qquad (1)$$

が定義できる．次に，I_{pbc} に対して T と同じ指示文

$$\text{DISTRIBUTE } I_{pbc} \text{ (CYCLIC}(b)\text{) ONTO } P \qquad (2)$$

を指示すると，図 3 による $x \in T$ のマッピングは式 (1) と (2) の合成による x のマッピングと等しくなることが容易に分かる．よって，以降では，テンプレート T のデータ分散のかわりに式 (1) と (2) のペアを扱う．

4.1.2 標準テンプレートの図式表現と区間表現

本項では標準テンプレートの図式表現を考察する．指示文 (2) によって分散されるデータを，同じプロセッサにマッピングされるもの別に標準テンプレート I_{pbc} 上で表現する．すると，b 個の連続要素からなる区間

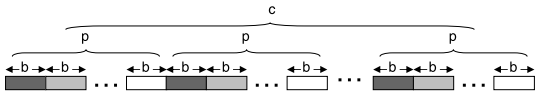

図 4 Block-cyclic 分散
Fig. 4 Block-cyclic distribution.

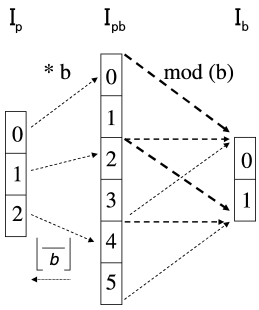

図 5 I_p，I_{pb}，および I_b の間の関係（$p = 3, b = 2$）
Fig. 5 Relationships among I_p, I_{pb} and I_b $(p = 3, b = 2)$.

（**ブロック区間**と呼ぶ）が p 個だけ連続的に並び，これら bp 個の連続要素列が c 個並ぶことになる（**図 4**）．そこで，ブロック区間を $I_b = [0 : b - 1]$，bp 個の連続要素列を $I_{pb} = [0 : bp - 1]$，p 個のプロセッサ列を $I_p = [0 : p - 1]$ 等の区間でモデル化する．以下，我々はこれらの区間とそれらの間の関係を図式で表現する．表現方法はいろいろ考えられるが，我々はブロック区間 I_b 中の添字が I_{pb} において連続と解釈される自然なモデルを選んだ．これは，データの連続性による通信最適化等への応用を考えたためである．

$\mathbf{Z}_b$，$\mathbf{Z}_p$，および $\mathbf{Z}_{pb}$ の間には写像 “mod$b$” と “b 倍” で結ばれた完全系列がある[2]．これのアナロジとして，区間 I_b 等に対して以下の完全系列を考える：

$$0 \to I_p \xrightarrow{*b} I_{pb} \xrightarrow{\bmod b} I_b \to 0 \quad \text{(exact)} \qquad (3)$$

図式 (3) では，“mod$b$” で $I_b = [0 : b - 1]$ に写されるブロック区間 $[0 : b - 1], [b : 2b - 1], \ldots$ は連続要素から構成されるので添字の連続性をモデル化できている．次に，“b 倍” によって I_p の要素 $0, 1, \ldots, p-1$ には I_{pb} の各ブロック区間の先頭要素 $0, b, 2b, \ldots$ が対応する．このことは，逆に，これら先頭要素が I_p の各要素，すなわち，プロセッサ $0, 1, \ldots, p-1$ にマッピングされると解釈できる．さらに，ブロック区間内の他の要素も同じプロセッサに写されると解釈する．すなわち，I_{pb} の元 x は写像 $x \mapsto \lfloor x/b \rfloor$ で計算できるプロセッサ番号にマッピングされる．以上のことは I_{pb} にブロックサイズ b のブロック分散が適用されたことを意味する（**図 5**）．さらに，図式 (3)，$h : I_b \ni x \mapsto x \in I_{pb}$，および，図式 $(***)$ より以下の直和表現が得られる：

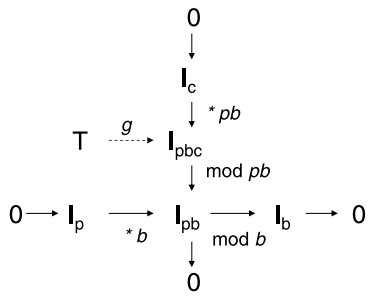

図 6 テンプレートに対するデータ分散の図式表現
Fig. 6 Diagram rep. of data mapping for template.

$$I_{pb} = \bigcup_{q=0}^{p-1} (bq + I_b). \tag{4}$$

ここで，$\bigcup$ は disjoint な和集合を表し，$x + I_b$ を $[x : x + b - 1]$ と定義する．同様にして，以下の図式および直和表現が得られる．図式 (5) は I_{pbc} において I_{pb} が連続的に並ぶこと，すなわち，図式 (3) と合わせると I_{pbc} が cyclic(b) でデータ分散されることを表す．

$$0 \to I_c \xrightarrow{*pb} I_{pbc} \xrightarrow{\text{mod } pb} I_{pb} \to 0 \quad \text{(exact)} \tag{5}$$

$$I_{pbc} = \bigcup_{j=0}^{c-1} \bigcup_{q=0}^{p-1} (pbj + bq + I_b). \tag{6}$$

以降，式 (6) の j の値を**分散周期**と呼ぶ．式 (1) および図式 (3) と (5) を I_{pb} で交差させることによって**図 6** の図式を得る．図中，点線矢印は線形でない写像を表す．また，図 6 および以降では “(exact)” は省略する．

4.2 アラインメントの図式表現

本節では D_A 次元配列 A と D_T 次元テンプレート T からなる以下の ALIGN を図式で表現する．

$$\text{ALIGN } A(\mathbf{I}) \text{ WITH } T(M\mathbf{I} + \mathbf{F} + \mathbf{J}) \tag{7}$$

ここで，$\mathbf{I}$ は D_A 次元ベクトルで各次元は異なるインデックス変数からなる．$\mathbf{J}$ と $\mathbf{F}$ は D_T 次元ベクトルで，$\mathbf{J}$ は REPLICATED な次元にインデックスを持ち，他の次元は 0；$\mathbf{F}$ は NORMAL および SINGLE な次元における定数項を対応する次元に含み，他の次元は 0；$\mathbf{M}$ はアライン添字を表現する (D_T, D_A) 型整数行列とする．これらは図 2 に対して以下となる：

$$\mathbf{I} = \begin{pmatrix} i \\ j \end{pmatrix}, \mathbf{J} = \begin{pmatrix} 0 \\ k \\ 0 \end{pmatrix}, \mathbf{F} = \begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix}, \mathbf{M} = \begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

一方，AXIS_TYPE が NORMAL，REPLICATED，SINGLE，VANISHED となる次元数を，各々，n，r，s，v とすると，以下が成立する：

$$n + v = D_A, \quad n + r + s = D_T.$$

よって，配列添字集合を $\mathbf{Z}$ の部分集合と見なすと，

$$A \subset \mathbf{Z}^{n+v}, \qquad T \subset \mathbf{Z}^{n+r+s}. \tag{8}$$

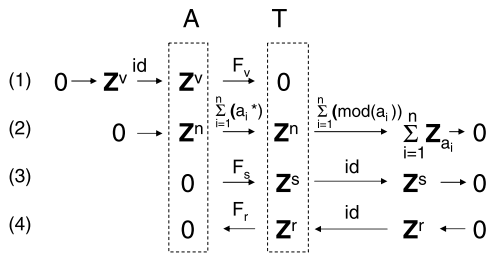

図 7 各 AXIS_TYPE に対する図式表現
Fig. 7 Diagram representation of each AXIS_TYPE.

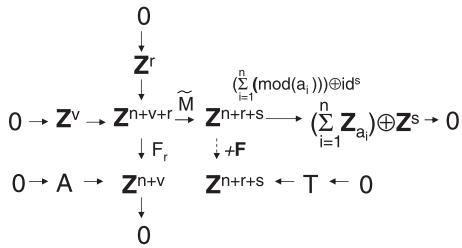

図 8 アラインメントの図式表現
Fig. 8 Diagram representation of alignment.

図 7 は配列 A およびテンプレート T の次元を AXIS_TYPE 別にまとめて得た 4 行の完全系列である．(1) から (4) は各々，VANISHED，NORMAL，SINGLE，および REPLICATED を表現する．式 (8) より，A および T は，図 7 中の左および右の点線枠内の $\mathbf{Z}$ 加群の直和の部分集合として表現される☆．ここで，id は恒等写像，F_v（または F_r）はすべての元を 0 に写す全射を表す．以下，図 7 の各図式を説明する．

(1) テンプレート T の任意の要素に対して $\mathbf{Z}^v$ のすべての元がアラインすることを表現する．

(2) 行列 $\mathbf{M}$ は HPF の規定より各行の非零要素はたかだか 1 つ．すなわち，NORMAL な次元に対し $\mathbf{M}$ は定数倍写像である．それを a_i 倍と表すと，剰余集合 $\mathbf{Z}_{a_i}$ を用いた上記完全系列を得る．

(3) 写像 F_s が単射であることから，テンプレート T で 1 つの定数ベクトルに対応することを表現する．定数は**図 8** で表現される．

☆ 図 7 は式 (7) の項 $\mathbf{F}$ を含んでない．これは図 8 で含める．

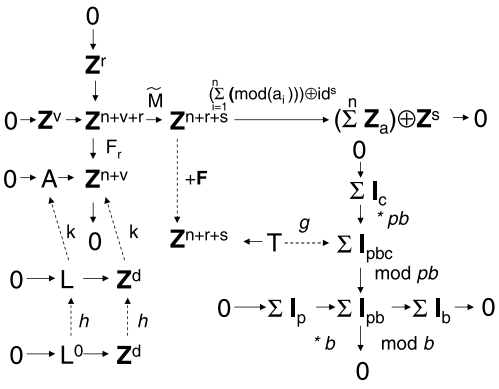

図 9 一般のデータ分散の図式表現
Fig. 9 Diagram representation of general data mapping.

(4) 配列 A の任意の要素に対して $\mathbf{Z}^r$ のすべての元がアラインすることを表現する．すなわち，配列 A の任意の要素がコピーされることを表す．

図 8 の図式は図 7 の完全系列をつないで得られる．すなわち，1 行目は (1)，(2) と (3) から，1 列目は (1) と (4) および，$id : \mathbf{Z}^n \mapsto \mathbf{Z}^n$ から得られる．ここで，$\mathbf{F}$ は指示文 (7) の $\mathbf{F}$ を，$\widetilde{\mathbf{M}}$ は $\mathbf{Z}^r$ に対応する次元に恒等写像が適用されるように $\mathbf{M}$ に列を追加した行列を表す．図 2 に対して $\widetilde{\mathbf{M}}$ は以下となる：

$$\widetilde{\mathbf{M}} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

4.3 一般の場合の図式表現

図 9 は多次元配列 A に対する HPF の規則的データ分散の図式表現である．この図式の左下以外の部分は図 6 中の各区間を次元ごとに直和をとったものと図 8 の図式をつないで得られる．左下部分において，L は d 重ループによるループ繰返し空間 $[l : u : m]^d \subset \mathbf{Z}^d$，$L^0$ は標準ループによるループ繰返し空間 $[0 : U : 1]^d \subset \mathbf{Z}^d$，$h$ は L^0 から L への写像 $h : x \mapsto m * x + l$，k はループ制御変数 i に配列 A の添字を対応させる写像 $k : i \mapsto i_a i + i_b$ を表す．ここで，$[x : y : z]$ は，左からループ下限値，ループ上限値，ストライドを表す．

なお，以降では 1 つのループ制御変数は 1 つの配列次元に現れ，1 つの配列次元には 1 つのループ制御変数が現れる場合のみ扱う．一般の場合は別に報告する．

5. 計算分散公式

本章では計算分散の対象となる AXIS_TYPE は NORMAL と SINGLE のみであることを示し，これらの AXIS_TYPE を持つ RDLS(1,1) パターンに対する計算分散を与える．

配列 A とテンプレート T においてループの計算分散に関連する次元は，ループ制御変数が現れる配列 A の次元，および，その次元がアラインするテンプレート T の次元である．このことから以下の次元は計算分散に無関係なことが分かる．

- AXIS_TYPE が REPLICATED な T の次元：対応する配列 A の次元を持たないため．
- AXIS_TYPE が VANISHED な A の次元：対応するテンプレート T の次元を持たないため．この次元に制御変数を含むループは元のままである．

よって，計算分散の対象となる AXIS_TYPE は NORMAL と SINGLE のみであり，図式を用いて計算分散を考えるには，図 9 を単純化，すなわち，図 9 において $r = v = 0$ とすればよい．このとき，$F_r = \mathrm{id}$ となるので，L^0 と I_{pbc} の間を結ぶ図式は以下となる☆：

$$L^0 \xrightarrow{h} L \xrightarrow{k} A \xrightarrow{f} T \xrightarrow{g} I_{pbc}$$

ここで，$f : i \mapsto f_a i + f_b$ は $\widetilde{\mathbf{M}}$ と $\mathbf{F}$ を合成したもので，アライン添字に対応する．

また，図 3 のプログラムは NORMAL と SINGLE の AXIS_TYPE を表現することを注意しておく．よって，以下では，図 3 のプログラムの計算分散を与える．その前に，以降で用いる記号を定義する．

定義 3 記号
$\psi(i)$: $A(i)$ がマッピングされるプロセッサ番号．
π: $= f_a i_a m$.
$t(j, q, x)$: $= (pbj + bq + x - gfk(l))/\pi$.
$\mathcal{L}(i)$: グローバル添字 i に対するローカル添字． □

ここでローカル配列とは，データ分散によりあるプロセッサに割り付けられる配列要素群を，そのローカルメモリ上に連続して並ぶように圧縮配置させた配列を指し，ローカル添字とはローカル配列における添字を指す．また，以降，ローカル配列の最小添字は元のグローバル配列の最小添字と同じとする．

定理 1 図 3 のループ L の計算分散コードは以下となる．

(A) $i_a f_a = 0$ のとき

```
if (ψ(i_b) = mype)
  do i = l, u, m
    A(i_a i + i_b) = ...
  enddo
```

☆ この図式は完全系列ではない．

endif

(B) $i_a f_a \neq 0$ のとき

$j = A_l$; $q = mype$

L1: do $i = A_l, A_u, 1$ // Get $\mathcal{L}$

if $(\psi(i) = q)$ $\mathcal{L}(i) = j{+}{+}$

enddo

$C_l = \lfloor gfk(l)/pb \rfloor; C_u = \lfloor gfk(u)/pb \rfloor$

L2: do $j = C_l, C_u, sign(\pi) * 1$

$S(j, q)$

L3: do $i = L(j,q), U(j,q), m$

$A(\mathcal{L}(i_a i + i_b)) = \ldots$

enddo; enddo

ただし，$S(j,q)$ は以下のコードである．

if $(\pi > 0)$ $L(j,q) = \max(m\lceil t(j,q,0)\rceil + l, l)$

$U(j,q) = \min(m\lfloor t(j,q,b-1)\rfloor + l, u)$

else $L(j,q) = \min(m\lceil t(j,q,b-1)\rceil + l, l)$

$U(j,q) = \max(m\lfloor t(j,q,0)\rfloor + l, u)$

証明 付録参照. □

6. 最適計算分散コード

本章では，ある配列 A に ALIGN 指示文が指定され，かつ，その align-target が block-cyclic 分散されるという条件の下でも，$\pi|b$，すなわち，b が π で割り切れる場合には最適なコードが生成できることを示す．これを示す準備として，以下を順に説明する．

- プロセッサ q にマッピングされる配列 A の最小要素，
- 配列 A のデータ分散前の添字（グローバル添字）をデータ分散後の添字（ローカル添字）に写す写像，
- データ分散前の配列・ループとデータ分散後の配列・ループとの関係（計算分散の図式表現）．

6.1 各プロセッサでの配列の下限添字

データ分散前の配列 A においてプロセッサ q にマッピングされる部分集合を A_q，データ分散後，プロセッサ q にマッピングされる配列を $\tilde{A}$，A_q の下限添字を A_q^l とする．このとき，A_q^l は以下のように計算できる．

補題 1 $f_a|b$ のとき，A_q^l は**図 11** の関数 GetMin を用いて以下のように計算できる：

$A_q^l = \mathrm{GetMin}(gf, A_l, f_a, 1)$.

証明 **図 10** を用いて説明する．$gf(x) = f_a x + f_b - T_l \in I_{pbc}$ による A_l の像を $A'_l = gf(A_l)$ とする．A'_l がマッピングされるプロセッサ番号を Q とすると，$Q = \lfloor A'_l \bmod pb/b \rfloor$ となる（図 6）．A'_l が含まれるブロック区間 J の先頭から A'_l までの要素数を δ とすると，$\delta = A'_l \bmod b$ であり，A'_l からブロック区間 J の最後の要素までの要素数は $b-1-\delta$ となる．

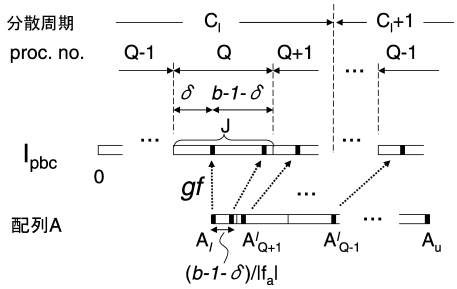

図 10 配列添字の I_{pbc} における像
Fig. 10 Image of array subscripts in I_{pbc}.

integer function GetMin(ϕ, x, y, z)

$Q = \lfloor (\phi(x) \bmod pb)/b \rfloor$

if $(q = Q)$ return (x)

else

$\delta = \phi(x) \bmod b$

if $(y > 0)$ $\delta' = b - 1 - \delta$; $\Delta_q = q - Q$

else $\delta' = \delta$; $\Delta_q = Q - q$

$\Delta_i = \lfloor \delta'/|y| \rfloor + 1 + \{(\Delta_q \bmod p) - 1\} b/|y|$

return$(x + \Delta_i z)$

endif

図 11 最小値取得関数 GetMin
Fig. 11 A function getting minimum value: GetMin.

まず，プロセッサ番号 $q = Q$ なら明らかに $A_q^l = A_l$.

次に，アファイン写像 gf の 1 次係数 $f_a > 0$ とする．このとき，図 10 において，配列添字の増加方向（右）と配列添字の gf による像の標準テンプレート I_{pbc} における増加方向（右）は一致する．したがって，その像がブロック区間 J の最後の要素になるような配列添字は $X = A_l + \lfloor (b-1-\delta)/|f_a| \rfloor$ となる．$f_a|b$ より $f_a < b$. よって，$gf(X+1)$ は隣接するブロック区間に含まれ，$Q+1$ にマッピングされる最初の A の要素となる．したがって，$q = Q+1$ に対して，$A_q^l = X+1$ となる．また，$f_a|b$ より $Q+2$ 以降のプロセッサに対する A_q^l は $X+1$ に $b/|f_a| \in \mathbf{Z}$ を順次加えることにより得られる．一方，$q < Q$ となるプロセッサ番号 q に対しては，図 10 より q が属する分散周期の次の分散周期に最小値を持つので，$q + p\,(> Q)$ をプロセッサ番号と見なすことにより上記と同様の計算が適用できる．

また，$f_a < 0$ の場合は，図 10 における配列添字の増加方向（右）と配列添字の gf による像の増加方向（左）

は逆転する．このとき，その像がブロック区間 J の最後の要素になるような配列添字は $X = A_l + \lfloor \delta/|f_a| \rfloor$ となる．後は上記と同様にして A_q^l の値が得られる．

以上の結果を整理すると，図 11 の関数 GetMin を用いて $A_q^l = \mathrm{GetMin}\ (gf,\ A_l,\ f_a,\ 1)$ が得られる．□

6.2 ローカルアドレス表現

本節では，配列 A のグローバル添字をローカル添字に写す写像を定式化し，次にこれを求める．

定義 4 X（と Y）をグローバルな（ローカルな）インデックスで表現されるオブジェクトとする．このとき，以下を満たす写像 $\phi : \mathbf{Z} \ni X \mapsto Y \in \mathbf{Z}$ を X のローカルアドレス表現と呼ぶ：

(1) ϕ は狭義単調増加：$x < y \Rightarrow \phi(x) < \phi(y)$,

(2) $\phi(X)$ は連続な区間．□

補題 2 ρ を標準テンプレート I_{pbc} にローカルな添字 I_{bc} を対応させる写像，$\Delta_A = (gf)^{-1}\rho(gf)(A_q^l) - A_l$，$\widetilde{f}(x) = f(x + \Delta_A)$，$\widetilde{g} = g$ とする．$f_a | b$ のとき，$\alpha = (\widetilde{g}\widetilde{f})^{-1}\rho(gf)$ は配列 A のローカルアドレス表現であり，$\alpha(A_q^l) = A_l$ となる．

証明 $\omega : x \mapsto x + \Delta_A$ とすると $\widetilde{f} = f \circ \omega$ より $\widetilde{f}^{-1} = \omega^{-1} \circ f^{-1}$．よって，$\alpha(x) = (gf)^{-1}\rho(gf)(x) - \Delta_A$．したがって，$\alpha(A_q^l) = A_l$．次に，$\Delta_A$ は定数なので，$\epsilon = gf$ として，$E(x) = \epsilon^{-1}\rho\epsilon(x)$ がローカル・アドレス表現であることを示せばよい．式 (4) と (6) より，

$$\rho : x = pbj + bq + y \mapsto bj + y$$

となるので，

$$j = \lfloor x/pb \rfloor, \quad y = x \bmod b,$$
$$\rho(x) = \lfloor x/pb \rfloor b + x \bmod b.$$

また，$\epsilon(x)$ を b で割った商は

$$\lfloor \epsilon(x)/b \rfloor = (\epsilon(x) - \epsilon(x) \bmod b)/b.$$

以上の 2 式より，

$$E(x) = \epsilon^{-1}(\lfloor \epsilon(x)/pb \rfloor b + \epsilon(x) \bmod b)$$
$$= \{\lfloor \epsilon(x)/pb \rfloor - \lfloor \epsilon(x)/b \rfloor\} b/f_a + x \in \mathbf{Z}.$$

ϵ と ϵ^{-1} は 1 次式なので単調．$\epsilon(A_q)$ は I_{pbc} の中でプロセッサ q にマッピングされる部分 $I_q = pbJ + bq + I_b$（ただし，J はある区間）に含まれる．ρ をこの I_q に制限したもの

$$\rho|_{I_q} : pbJ + bq + I_b \mapsto bJ + I_b$$

は単調増加．よって，α は単調増加．次に，A_q は以下の形で表現できる：

$$[A_q^l : B] \cup \bigcup_{i=1}^{N-1} [C + pbi/|f_a| : B + pbi/|f_a|]$$
$$\cup [C + pbN/|f_a| : D].$$

ここで，図 10 より $f_a > 0$ のとき

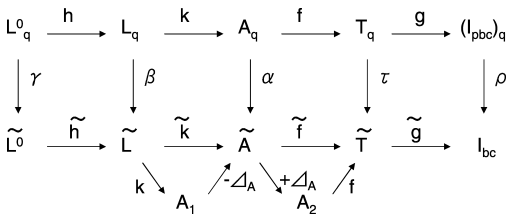

図 12 計算分散の図式表現
Fig. 12 Diagram rep. of computation partitioning.

$$C = A_q^l - \lfloor \delta/f_a \rfloor, \quad B = C + b/f_a - 1,$$
D：A_qに含まれる最大の添字，　δ：補題 1 のδ.

α が狭義単調増加なので，$\alpha(A_q)$ の連続性は，より大きな以下の区間に対して証明すればよい：

$$A'_q = \bigcup_{i=0}^{N} [C + pbi/|f_a| : B + pbi/|f_a|].$$

$$A'_q = [0 : N : 1] * pb/f_a + C + [0 : b/f_a - 1 : 1]$$

とも書けるので

$$gf(A'_q)$$
$$= [0 : N : 1] * pb + gf(C) + [0 : b - f_a : f_a].$$

よって，

$$E(A'_q) = [0 : (N+1)b/f_a - 1 : 1] + E(C).$$

すなわち，$\alpha(A_q)$ は連続領域となる．$f_a < 0$ でも同様．□

6.3 計算分散の図式表現と最適計算分散コード

本節では計算分散の図式表現を示し，最適計算分散コードを導く．

補題 3（グローバル・ローカルアドレス間の関係）$\widetilde{h} = h$，$\widetilde{k} = k - \Delta_A$，$L_q = \{i \in L | k(i) \in \underset{\sim}{A}_q\}$，$L_q^0 = \{i \in L^0 | h(i) \in L_q\}$，$\beta = \widetilde{k^{-1}\alpha k}$，$\gamma = \widetilde{h^{-1}\beta h}$ とする．また，A_q 等の α 等による像を $\widetilde{A}_q$ 等と記す．$\pi | b$ のとき，**図 12** の図式における α，β，γ は各々，A_q，L_q，L_q^0 に対するローカルアドレス表現を与える．

証明 $\widetilde{f}\widetilde{k} = fk$，$\widetilde{f}\widetilde{k}\widetilde{h} = fkh$ を用いれば，補題 2 と同様に証明される．□

以上より最適計算分散コードは次の形で述べられる．

定理 2 $\pi | b$ のとき，図 3 のループ L を計算分散した結果は以下の 1 重ループで与えられる．

$$A_q^l = \mathrm{GetMin}(gf, A_l, f_a, 1)$$
$$\Delta_A = \alpha(A_q^l) - A_l$$
$$l_1 = \mathrm{GetMin}(gfk, l, \pi, m)$$
$$L_c = \{\lfloor gfk(l_1)/pb \rfloor - \lfloor gfk(l_1)/b \rfloor\} b/f_a i_a + l_1$$
$$u' = l + \lfloor |u - l|/|m| \rfloor * m$$

$u_1 = \mathrm{GetMin}(gfk, u', -\pi, -m)$
$U_c = \{\lfloor gfk(u_1)/pb \rfloor - \lfloor gfk(u_1)/b \rfloor\} b/f_a i_a + u_1$
do $i = L_c, U_c, m$
  $A(i_a i + i_b - \Delta_A) = \ldots$
enddo

証明 補題 3 より $\tilde{L}_q^0 = \gamma(L_q^0)$ はローカルなループ空間 $\tilde{L}^0$ における連続区間；すなわち，ストライド 1 の 1 重ループのインデックス空間を表現する．よって，ストライド m の 1 重ループ $L = h(L_q^0)$ の β による像 $\beta(h(L_q^0)) = \tilde{h}(\tilde{L}_q^0)$ もローカルなループ空間 $\tilde{L}$ におけるストライド m の 1 重ループになる．

次に，$\beta(i) = \tilde{i}$ は $\tilde{L}_q$ のループ制御変数を表す．よって，配列添字 $i_a i + i_b$ の α による像 $\alpha(i_a i + i_b)$ は $\alpha(k(i)) = \tilde{k}(\beta(i)) = i_a \tilde{i} + i_b - \Delta_A$ となる．

最後に，ループ下限値 L_c に対しては補題 1 の証明と同様にして $l_1 = \mathrm{GetMin}\ (gfk, l, \pi, m)$ が証明でき，$L_c = \beta(l_1)$ を得る．上限値 U_c に対してもループ制御変数の最終値 u' を用いて L_c と同様に求められる． □

なお，$\pi \nmid b$ だが $\pi | 2^n b$ かつ $2^n | p$ のとき，2^n-way の SMP ノードからなるクラスタ向けに最適な階層並列コードを生成することができる．すなわち，p プロセッサ上の cyclic(b) 分散を $p/2^n$ プロセッサ上の cyclic($2^n b$) 分散に変更して定理 2 のコードを生成し，その生成コードに 2^n 個のスレッドからなる SMP 向けスレッド並列化を適用すればよい．

7. 擬周期性とテーブル参照法コード

本章では，block-cyclic 分散に対する計算分散コードにおける添字参照が，従来[4] よりも小さい周期を持つことを示し，do ループを用いた計算分散コードを導く．以下，$f_a i_a \neq 0$ とする．

7.1 参照添字の擬周期性

本節では図 3 のループ L の添字参照の擬周期性とローカル添字が持つ性質を述べる．

補題 4

$$\lambda = LCM(pb, \pi), \quad \kappa = \lambda/\pi \tag{9}$$

とすると，図 3 のループ L の添字参照において以下の周期性が成立する；すなわち，ループインスタンス $L(i)$ と $L(i + m\kappa)$ は同じプロセッサで実行される：

$$\psi(i_a(i + m\kappa) + i_b) = \psi(i_a i + i_b). \tag{10}$$

証明 $f_a i_a m\kappa \bmod pb = \lambda \bmod pb = 0$ より明らか． □

よって，$[i : i + m\kappa - 1 : m]$ の範囲でプロセッサ q で実行されるループ制御変数を見つければ，プロセッサ q で実行される任意のループ制御変数はその値と $m\kappa$ の定数倍との和で表される．

補題 4 の周期は従来の周期[4] よりも短い．従来，各プロセッサに対して b 個の参照ごとに周期を持つとしていた．このとき，全プロセッサに対する周期の合計は pb となる．一方，補題 4 は，全プロセッサに対して第 1 回目の周期はストライド m を持つループ範囲 $[i : i + m\kappa - 1 : m]$ に含まれることを主張する．このとき，全プロセッサに対する周期の合計は $\kappa = LCM(pb, \pi)/\pi \leq pb$ となる．

また，ローカル添字を表す配列 $\mathcal{L}$ は以下を満たす．

補題 5 I をループ区間 $[l : u : m]$，$l_q \in I$ をループインスタンス $L(l_q)$ がプロセッサ q で実行されるような最初の値，G_q^l を $\mathcal{L}(i_a(l_q + m\kappa) + i_b) - \mathcal{L}(i_a l_q + i_b)$ とする．このとき，ループインスタンス $L(i_q)$ がプロセッサ q で実行されるような任意の i_q に対して，

$$G_q^i = G_q^l.$$

証明 式 (10) より以下が容易に得られる：

$$\begin{aligned} G_q^i &= \mathcal{L}(i_a(i_q + m\kappa) + i_b) \\ &\quad - \mathcal{L}(i_a(l_q + m\kappa) + i_b) \\ &\quad + \mathcal{L}(i_a(l_q + m\kappa) + i_b) - \mathcal{L}(i_a i_q + i_b) \\ &= \mathcal{L}(i_a i_q + i_b) - \mathcal{L}(i_a l_q + i_b) \\ &\quad + \mathcal{L}(i_a(l_q + m\kappa) + i_b) - \mathcal{L}(i_a i_q + i_b) \\ &= G_q^l. \end{aligned}$$ □

よって，任意の i に対してある定数 G_q が存在して，

$$\mathcal{L}(i_a(i + m\kappa) + i_b) = \mathcal{L}(i_a i + i_b) + G_q.$$

となる．我々はこの式を参照添字の**擬周期性**と呼ぶ．

7.2 テーブル参照法コード

前節の結果より以下の定理が成り立つ．

定理 3 図 3 のループ L の計算分散コードは以下となる．

$N_e = 0$
do $i = l, l + m\kappa - 1, m$ // inspector
  if $(\psi(i_a i + i_b) = q) i_x(N_e + +) = \mathcal{L}(i_a i + i_b)$
enddo
do $i = l + m\kappa, l + 2m\kappa - 1, m$ // get G_q
  if $(\psi(i_a i + i_b) = q) g = \mathcal{L}(i_a i + i_b)$; exit
enddo
$G_q = g - i_x(0)$; $N_c = \lfloor (u - l + 1)/m\kappa \rfloor$
do $i = 0, N_c - 1$ // executor
  do $j = 0, N_e - 1$
    $A(i_x(j) + iG_q) = \ldots$
enddo; enddo
j=0
do $i = l + N_c m\kappa, u, m$ // residue loop

表 2 実測用パラメータ
Table 2 Parameters for evaluation.

$p=8,\quad i_a=-2,\quad i_b=7,$	$f_a=-3,\quad f_b=1,$
$l=100{,}000*(-m)*p,$	$u=-m,$
$A_l=i_a l+i_b,$	$A_u=i_a u+i_b,$
$T_l=f_a A_u+f_b,$	$T_u=f_a A_l+f_b.$

表 3 適用された最適化
Table 3 Applied optimizations.

rr	if-conversion, pseudo-vectorization
th	(to inner loop) pseudo-vectorization
t-w	if-conversion
t-d	(to inner loop) 2× loop unrolling
t-2d	(to inner loop) 2× loop unrolling, pseudo-vec.
opt	4× loop unrolling

if $(\psi(i_a i+i_b)=q)A(i_x(j++)+N_c G_q)=\ldots$
enddo

証明 擬周期性と補題 5 より明らか. □

また，$N_e^-=N_e-1$ とし，配列 Γ と Δ を

$$\Gamma(i_x(i))=\begin{cases} i_x(i+1) & \text{if} \quad 0\le i<N_e^- \\ i_x(0) & \text{if} \quad i=N_e^- \end{cases}$$

$$\Delta(i_x(i))=\begin{cases} i_x(i+1)-i_x(i) & \text{if} \quad 0\le i<N_e^- \\ i_x(0)-i_x(i)+G_q & \text{if} \quad i=N_e^- \end{cases}$$

と定義すると，以下が容易に得られる．

系 1 定理 3 の executor および residue ループは以下の従来タイプのテーブル参照法コード[4),10)] になる：

$N_r=0$
do $j=l+N_c m\kappa, u, m$ // count residues
if $(\psi(i_a j+i_b)=q)N_r++$
enddo
$i=i_x(0);\quad \delta=i_x(0);\quad j=0$
while $(j<N_c N_e+N_r)$ // executor
$A(i)=\ldots;\quad j++$
$i+=\Delta(\delta);\quad \delta=\Gamma(\delta)$
endwhile

8. 評　　価

定理 2 のコードを評価するために種々の計算分散法と性能を比較した．対象プログラムは**表 2** に示すパラメータを持つ図 3 のプログラムである．ただし，ブロックサイズ b とループストライド m には定理 2 の前提条件を満たす値（$m=-1,-3,-25$，b は $b/\pi=-1,-5,-20,-80$ となる値）を組み合わせた 12 種類の値を用いた．ここで，表 2 より $\pi=6m$ となる．

適用した計算分散法は，定理 2 の手法（opt），実行時解決法（rr），定理 1 の手法（th），系 1 による従来の while ループを用いたテーブル参照法（t-w），t-w における while ループを do ループに変更したもの（t-d），および，定理 2 で示した 2 重 do ループによるテーブル参照法（t-2d）の 6 種類である．

測定マシンは分散メモリ型並列計算機である日立 SR8000/E0（処理性能は 9.6 GFLOPS/node）である．OS は HI-UX/MPP 03-07，Fortran90 コンパイラは OFORT90 V01-06-/A であり，指定オプションは最速オプション o(ss) と mp(p(0)) である．mp(p(0)) は，SMP クラスタ構成の SR8000 において自動 SMP 並列化の適用を避け，ノード内では 1 プロセッサのみ実行させるために指定した．なお，エラーが発生したため，プログラム th にはループ展開を適用しないオプション loopexpand(0) を追加した．

今回の計算分散の性能比較ではプロセッサ間通信は発生しないので，実測は SR8000 の 1 ノード中の 1 プロセッサを用いて 8 プロセッサ分の計算を行った．具体的には，プロセッサ番号を変数とした SPMD 型の計算部分を作成し，その計算部分をプロセッサ番号が 0 から 7 まで変化するループで囲んだ．ただし，測定間におけるキャッシュの再利用を避けるために，各プロセッサ番号および各パラメータによる実測の直前に巨大配列への代入文を実行し，キャッシュをクリアした．SR8000 のキャッシュは L1（サイズは 128 KB）のみであり，store データは必ずキャッシュに書き込まれるので上記代入文によりキャッシュはクリアされる．

測定区間は，テーブル参照法では executor ループのみ，その他のプログラムでは計算分散したループ以外にループの上下限値を求める計算も含めた．ただし，配列 $\mathcal{L}$ を求めるループは全プログラムで除外した．

表 3 に各プログラムに適用された最適化を示す．擬似ベクトル化は rr，ならびに，th と t-2d の内側ループに適用された．また，t-d および t-2d の内側ループは 2 倍展開され，opt の内側ループは 4 倍展開された．

表 4 および **図 13** は SR8000 上での実測結果である．表中の値は 8 プロセッサ分の実行時間の平均である．なお，すべてのプログラムにおいて各プロセッサで参照された配列要素数は 100,000 個だったので，各プロセッサの負荷は均一であり，各プロセッサの実行時間もほぼ同一であった．表 4 および図 13 より以下が分かる：

表 4 SR8000 上での実測結果 [ms]
Table 4 Evaluation results on SR8000 [ms].

$(m, b/\pi)$	rr	th	t-w	t-d	t-2d	opt
(−1,−1)	1,100	53.7	5.64	3.41	4.82	.47
(−1,−5)	1,103	22.4	5.64	3.43	4.55	.47
(−1,−20)	1,102	8.7	5.64	3.43	1.82	.47
(−1,−80)	1,107	4.5	5.65	3.43	1.25	.47
(−3,−1)	1,103	53.8	5.64	3.43	4.86	.47
(−3,−5)	1,108	28.5	5.64	3.43	4.64	.48
(−3,−20)	1,106	14.3	5.64	3.42	2.16	.47
(−3,−80)	1,110	8.8	5.66	3.43	2.53	.47
(−25,−1)	1,107	53.7	5.64	3.43	4.87	.52
(−25,−5)	1,110	55.0	5.64	3.42	4.65	.52
(−25,−20)	1,107	46.1	5.65	3.44	2.14	.52
(−25,−80)	1,118	36.5	5.69	3.46	2.53	.54

表 5 SR8000 上での実測結果（同程度の最適化）[ms]
Table 5 Evaluation results on SR8000 (equally optimized) [ms].

$(m, b/\pi)$	rr	th	t-w	t-d	t-2d	opt
(−1,−1)	1,119	68.6	5.63	4.04	4.05	.52
(−1,−5)	1,137	24.1	5.64	4.03	4.20	.52
(−1,−20)	1,136	8.5	5.64	4.03	1.80	.52
(−1,−80)	1,140	4.9	5.68	4.05	1.50	.52
(−3,−1)	1,159	81.5	5.62	4.03	4.06	.52
(−3,−5)	1,179	27.7	5.65	4.03	4.30	.52
(−3,−20)	1,176	13.8	5.64	4.04	2.40	.52
(−3,−80)	1,181	9.7	5.69	4.05	2.79	.52
(−25,−1)	1,415	81.5	5.63	4.02	4.06	.55
(−25,−5)	1,428	53.1	5.64	4.03	4.28	.55
(−25,−20)	1,424	44.5	5.66	4.04	2.42	.54
(−25,−80)	1,430	42.3	5.70	4.07	2.77	.55

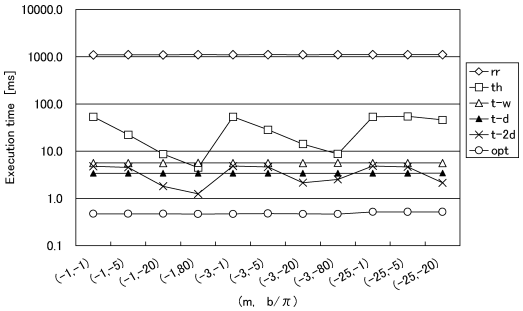

図 13 SR8000 上での実測結果
Fig. 13 Evaluation results on SR8000.

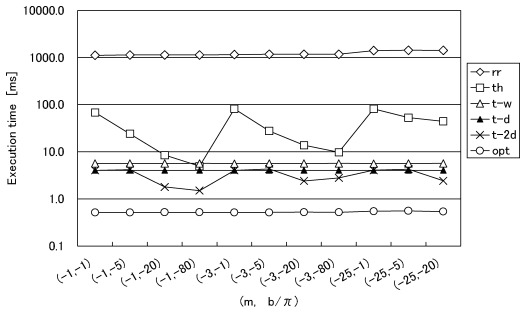

図 14 SR8000 上での実測結果（同程度の最適化）
Fig. 14 Evaluation results on SR8000 (equally optimized).

(1) opt は他のコードと比べて 2 倍以上高速である．特に従来法である t-w に比べて 10 倍以上高速である．これは 1 重ループであり，しかも配列添字が単純であるためと考えられる．

(2) th や t-2d はパラメータによって性能が変化する．これは内側ループをほぼ b/π 回実行するため，その回数が増えるに従い，ループ最適化の効果が現れたためと考えられる．一方，これらを除いた他のプログラムは 1 重ループでループ繰返し回数が不変のため，ストライドやブロックサイズの影響を受けない．

(3) th は b/π が大きな値の場合，従来法 t-w よりも高速になる場合がある．

(4) テーブル参照法では while ループよりも do ループの方が高速である．これは do ループの方がコンパイラで最適化されやすいためと考えられる．

上記実測では，高速になったプログラムにはループ展開等の最適化が適用されたため，純粋にコード生成法の比較になっていなかった．そこで，適用される最適化がほぼ同じになるようにして比較を行った．すなわち，コンパイルオプションを o(ss)，swpl(0)，loop-expand(0)，nopvec，mp(p(0)) とし，これ以外は上記と同じ測定環境を用いて実測した．この結果，どのプログラムに対しても，ソフトウエアパイプライニング，ループ展開，擬似ベクトル化は適用されなかった．

表 5 および **図 14** は SR8000 上での実測結果である．これにより，適用される最適化を同一にしてもほぼ同様な結果が得られることが分かる．

9. 関 連 研 究

ある配列がテンプレートに ALIGN されて block-cyclic 分散される場合，テーブル参照法以外に，その配列参照を含むループの計算分散方法がいくつかある．

実行時解決法は任意のプログラムに適用可能であるが，コードの処理性能は非常に低い[8]．

D system[1] における計算分散法は定数ストライドを持つループに適用可能である．D system はプロセッサ数やブロックサイズがコンパイル時に定数となる場合，オメガテスト[11] を用いてコードを生成する．生成コードの質はオメガテストの解析精度によるため，その実行性能は一般に不明確である．また，block-cyclic 分散に対してオメガテストを使ったコード生成方法は記述されてない．一方，プロセッサ数やブロックサイズがコンパイル時に定数でない場合は Virtual Pro-

cessor 法（VP 法）[5] によるコードを生成する．これは block-cyclic 分散を block 分散と cyclic 分散の 2 つに分割して行うコード生成法である．配列は 2 次元化されループは 2 重ループ化される．block-cyclic 分散に対して VP 法コードはテーブル参照法コードより遅い[10]．

文献 3) は行列を解くことによって計算分散する方法を示した．この方法では block-cyclic 分散に対して配列は 2 次元化された後にデータ分散され，次にデータ分散後の配列中に生じた元の配列には存在しない要素（hole[9]）を除去するために各次元で圧縮されるが，hole は完全には除去されない．圧縮にともない，配列要素の参照順は元のプログラムにおける参照順と異なるものになるため，DOALL ループにしか適用できない．生成ループは 2 次元圧縮配列の各次元をたどるように 2 重ループ化される．評価はないが，2 重ループ化されていることから性能は VP 法程度であると予想される．

文献 9) は ALIGN に対応できるように VP 法を拡張した．配列を 2 次元化し，ループを 2 重ループ化する．

文献 12) は配列をポインタで参照するが，実質，配列は 2 次元化され，ループも 2 重ループ化される．よって，これも VP 法と同等の性能であると予想される．

以上のいずれの研究においても，効率的な 1 重ループに変換した例はない．本論文はある特別な場合に，効率的な 1 重ループが出力できることを示した．なお，我々の出力コードではデータ分散後の配列において要素は連続に配置されるため，hole は生じない．

Block-cyclic 分散には対応しているが，ALIGN に対応していない方法として以下の研究がある．文献 7) はループのストライドが正の場合のみ扱う．文献 10) の方法は参照添字が作る格子に対する 1 つの基底ベクトルを求め，それを使って参照添字を計算する．本方法は文献中のいくつかの実測☆においてテーブル参照法よりも最高で 1.5 倍高速になる．しかし，高速になるための条件は不明確である．

10. おわりに

我々は，我々が RDLS(1,1) パターンと呼ぶ，HPF の一般の規則的データ分散が指示された配列への参照を含む一般の 1 重ループを，図式による一般的枠組みの中で取り扱うことにより以下に示す結果を得た．

- 上記パターンに対する計算分散コードを与えた．
- block-cyclic 分散と ALIGN 指示文が指定された配列への参照を含むループに対して，ある 3 つのパラメータの積がブロックサイズの約数となる場合に最適な計算分散コードを与えた．
- 一般的計算分散コードにおける添字参照において従来より短い周期があることを示し，これより 2 重 do ループ，1 重 do ループ，または 1 重 while ループからなるテーブル参照法コードを導いた．特に，ストライドが負の場合に対してもテーブル参照法コードを与えた．
- 提案した種々の方法を複雑な block-cyclic 分散を持つプログラムに適用して日立 SR8000 上で実測した．その結果，最適計算分散コードが従来，block-cyclic 分散に対して最速であるとされたテーブル参照法コードよりも 10 倍以上高速であること，ならびに，do ループによるテーブル参照法コードが上記従来法コードよりも高速であることを示した．

本論文の結果は，一般のループ制御式を持つ多重ループ，一般の規則的な HPF データ分散指示文が与えられた多次元配列，および，互いに異なるループ制御変数による一般の 1 次式を各次元の添字に持つ多重ループ中の配列への参照，を含むプログラムに拡張可能である．

謝辞 日立製作所ソフトウエア事業部の根岸清氏および人見洋一氏には SR8000 の使用に際してご協力いただきましたことを感謝いたします．

☆ テーブルや基底ベクトルの計算時間は含まない．

参考文献

1) Adve, V. and Mellor-Crummey, J.: Using Integer Sets for Data-Parallel Program Analysis and Optimization, *Proc. PLDI '98*, pp.186–198 (1998).
2) 岩井斉良：ホモロジー代数入門，サイエンス社 (1978).
3) Ancourt, C., Coelho, F., Irigoin, F. and Keryell, R.: A linear algebra framework for static HPF code distribution, *Proc. CPC '93*, Delft, Netherlands, pp.161–172 (1993).
4) Chatterjee, S., Gilbert, J., long, F., Schreiber, R. and Teng, S.-H.: Generating local addresses and communication sets for data-parallel programs, *Proc. PPoPP '93*, pp.149–158 (1993).
5) Gupta, S., Kaushik, S., Huang, C.-H. and Sadayappan, P.: Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines, *J. Parallel and Distributed Computing*, Vol.32, pp.155–172 (1996).
6) High Performance Fortran Forum: High Per-

formance Fortran 2.0 公式マニュアル (1999).
7) Hiranandani, S., Kennedy, K., Crummey, J.M. and Sethi, A.: Compilation Techniques for Block-Cyclic Distributions, *Proc. ICS '94*, Manchester, UK, pp.392–403, ACM (1994).
8) Hiranandani, S., Kennedy, K. and Tseng, C.-W.: Compiling Fortran D for MIMD Distributed-Memory Machine, *CACM*, Vol.35, pp.66–80 (1992).
9) Kaushik, S.D., Huang, C.-H. and Sadayappan, P.: Efficient Index Set Generation for Compiling HPF Array Statemets on Distributed-Memory Machines, *J. Parallel and Distributed Computing*, Vol.38, pp.237–247 (1996).
10) Kennedy, K., Nedelijkovic, N. and Sethi, A.: Efficient Address Generation for Block-Cyclic Distribution, *Proc. ICS '95*, pp.180–184 (1995).
11) Pugh, W.: A practical algorithm for exact array dependence analysis, *CACM*, Vol.35, No.8, pp.102–114 (1992).
12) van Reewijk, K., Denissen, W., Sips, H.J. and Paalvast, E.: An Implementation Framework for HPF Distributed Arrays on Message-Passing Parallel Computer Systems, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.9, pp.897–914 (1996).

付　　録

A.1 定理 1 の証明

まず，$f_a i_a \neq 0$ の場合を証明する．ループ L1 はグローバル添字にプロセッサ q におけるローカル添字を対応させる配列 $\mathcal{L}$ を計算するコードである．後の証明で分かるように，ループ L3 のループ制御変数 i は，プロセッサ q にマッピングされる A の部分配列のみを参照するような範囲を動く．よって，配列 A の添字は単にローカル添字 $\mathcal{L}(i_a i + i_b)$ を使って表現できる．

以下，ループ L2 と L3 に対するループ範囲（上限値，下限値，およびストライドの組）について証明する．プロセッサ q にマッピングされる標準テンプレート I_{pbc} の添字集合は 3.1 節より，

$$\bigcup_{j\in[0:c-1:1]} [pbj + bq : pbj + bq + b - 1 : 1] \quad (11)$$

となる．ここで，$c = \lceil e/pb \rceil$ である．また，

$$\bigcup_{j\in[l:u:m]} X_j \quad (12)$$

は，区間 $X_l, X_{l+m}, \cdots, X_{l+m\lfloor (u-l)/m \rfloor}$ がこの順序に並んでできる区間集合を表す．順序も考慮するのはループ範囲を求めるためである．

次に以下の方針で計算分散後のループ範囲を求める．

(1) 式 (11) より計算分散後のループ範囲を計算．
  (a) 式 (11) に $(gfkh)^{-1}$ を作用させ，その結果と $\mathbf{Z}$ との交わりをとる．
  (b) 上記結果に h を作用させ，その結果と $[l:u:1]$ との交わりをとる．
(2) 式 (11) の j の範囲からループ L2 の範囲を計算．

この方針に関して以下にいくつかの注意をする．

注意 1： 方針 (1)(a) は本来は g^{-1} 等を作用させるたびにその結果と A 等との交わりをとるべきと考えられる．しかし，上記の各関数が 1 対 1 写像なので，以下の補題 6 により，$(gfkh)^{-1}$ を一度に作用させた後に $(L^0 \subset)\mathbf{Z}$ との交わりをとっても同じ結果を得る．

補題 6 $f : X \mapsto Y$，$g : Y \mapsto Z$ を 1 対 1 写像とする．このとき，以下が成り立つ：

$$f^{-1}(g^{-1}(Z) \cap Y) \cap X = (gf)^{-1}(Z) \cap X. \quad (13)$$

証明 f を 1 対 1 写像とするとき，以下が成り立つ：

$$f(V \cap W) = f(V) \cap f(W).$$

式 (13) の両辺の各々に gf を作用させ，この等式を用いることで補題は証明される． □

注意 2： 方針 (1) は，本来，$(gfk)^{-1}$ を作用させ，その結果と $[l:u:m]$ との交わりをとるべき．しかし，交わりの計算が困難なため，$[l:u:m]$ をストライドと上下限値の 2 つに分解して交わりを計算した．すなわち，(a) で $(gfkh)^{-1}$ を作用させ，$\mathbf{Z}$ との交わりをとった後で h を作用させることでストライド m を，(b) で $[l:u:1]$ との交わりをとることで上下限値との交わりをとった．

以下，上記方針に従って証明する．まず，

$$(gfkh)^{-1}(i) = (i + T_l - f_b - f_a i_b - f_a i_a l)/\pi$$

より，方針 (1)(a) に従って式 (11) の区間は以下となる：

$$\bigcup_{j\in[L_c:U_c:sign(\pi)*1]} [L(j) : U(j) : 1]. \quad (14)$$

ここで，$\pi < 0$ なら各区間の上下限は逆になり，区間列も逆順になることから以下を得る．

$$\begin{aligned} &L(j) = \lceil l_j \rceil, U(j) = \lfloor u_j \rfloor, \\ &\pi > 0 \text{ なら } l_j = t(j,q,0), u_j = t(j,q,b-1), \\ &\pi < 0 \text{ なら } l_j = t(j,q,b-1), u_j = t(j,q,0). \end{aligned} \quad (15)$$

ただし，

$$t(j,q,x) = (pbj + bq + x - gfk(l))/\pi.$$

これと，方針 (1)(b) より，定理 1 のループ L3 を得る．

次に，方針 (2) に従って j のループ範囲を計算する．

$$w(x) = gfk(x) = f_a(i_a x + i_b) + f_b - T_l \quad (16)$$

とおくと，

$m > 0$ なら $l \leq u, m < 0$ なら $l \geq u$

であり，$w(x)$ の x の係数は $f_a i_a$ なので

$$\pi = f_a i_a m > 0 \text{ なら } w(l) \leq w(u),$$
$$\pi = f_a i_a m < 0 \text{ なら } w(l) \geq w(u). \quad (17)$$

また，l は標準テンプレート I_{pbc} において $C_l = \lfloor w(l)/pb \rfloor$ 番目の分散周期に，u は $C_u = \lfloor w(u)/pb \rfloor$ 番目の分散周期に含まれる．

ここで，l や u がマッピングされないプロセッサ q に対して，その j の範囲は上記の範囲より狭くなる可能性がある．しかし，範囲の厳密な計算は方針 (1)(b) で行われているので，ここでは少し広い範囲であっても間違いではない．よって，定理 1 のループ L2 を得る．以上により，$f_a i_a \neq 0$ の場合に定理が証明された．

次に，$f_a i_a = 0$ の場合に対して証明する．このとき，$A(i_a i + i_b)$ は $T(f_a(i_a i + i_b) + f_b) = T(f_a i_b + f_b)$ と同じプロセッサにマッピングされるので，図 5 よりそのプロセッサ番号は以下となる．

$$\lfloor (f_a i_b + f_b - T_l) \bmod pb/b \rfloor.$$

よって，$f_a i_a = 0$ の場合に定理が証明された． □

(平成 17 年 1 月 31 日受付)
(平成 17 年 7 月 4 日採録)

佐藤　真琴（正会員）

昭和 34 年生．昭和 59 年京都大学理学部数学科修了．昭和 62 年神戸大学大学院理学研究科数学専攻修士課程修了．同年（株）日立製作所中央研究所入社．現在，同システム開発研究所勤務．汎用コンピュータ，スーパコンピュータ，および，組み込みプロセッサを対象とした最適化コンパイラ，プログラム開発環境，および，アーキテクチャ評価の研究開発に従事．