ハイパフォーマンス
コンピューティング 72－15
（1998. 8. 7）

並列化コンパイラTINPARにおける自動データ分割決定手法

窪田昌史[†] 辰巳尚吾[†††] 五島正裕[††]
森眞一郎[††] 富田眞治[††] 津田孝夫[†]

本稿では，メッセージ交換型並列計算機のための並列化コンパイラ TINPAR における配列変数の自動分割手法について述べる．従来，プログラム実行中にデータ分割を固定する場合 (静的データ分割) については，有効な手法が提案されている．一方，複雑なプログラムでは，プログラム実行中に配列変数の再分割 (動的データ分割) を行なう方が高い性能が得られることもあるが，最適なデータ再分割の位置を解析するには多大な計算量が必要である．そこで我々は，手続き内では静的データ分割，手続き呼び出し時には必要に応じてデータ再分割を行なうようなデータ分割決定手法を採用することで，解析に多大な計算時間を必要としない自動データ分割決定部を実現した．
また，本システムをいくつかのベンチマークプログラムに適用して得られたデータ分割の結果についても報告する．

Automatic Data Distribution Scheme for Parallelizing Compiler TINPAR

ATSUSHI KUBOTA,[†] SHOGO TATSUMI,[†††] MASAHIRO GOSHIMA,[††]
SHIN-ICHIRO MORI,[††] SHINJI TOMITA[††] and TAKAO TSUDA[†]

In this paper, we present the automatic data distribution scheme for TINPAR; a parallelizing compiler for message-passing multiprocessors. Several researchers have proposed systems that are able to produce static data distributions for the entire execution of a program. For complex programs, dynamic data distributions during the parallel execution of the programs may be required to obtain acceptable performance. Inter-procedural analysis of automatic data distribution however, requires much computation at compile-time. We therefore have been implemented a tool for automatic data distributions which determines intra-procedural static distributions and inter-procedural dynamic distributions.
Preliminary results of the selected data distributions as a result of applying our system to several programs are also reported.

1. はじめに

近年，データ並列性を持つプログラムを高速に実行するための手段として，メッセージ交換型並列計算機が普及してきているが，その上で稼働する並列プログラムの作成には，データの各プロセッサへの分割，処理の分割，プロセッサ間通信コードの挿入などの困難が伴う．そこで，データ並列性を持つプログラムを記述するための FORTRAN 言語の拡張として HPF(High Performance Fortran)[1] が提案され，国内外の研究機関やメーカによってコンパイラが開発されている．

しかし，各プロセッサへのデータの分割がプログラムの実行時間に多大な影響を与えるにもかかわらず，HPF ではデータの分割はユーザが指定しなければならず，逐次プログラムを自動的に並列化することはできない．また，複数のアーキテクチャの並列計算機間で HPF のプログラムを移植するにはデータの分割を指定し直す必要があり，並列計算環境間のプログラムの可搬性を保持できない．

そこで我々は，メッセージ交換型並列計算機のための並列化コンパイラ TINPAR[2] を再構築して，データ分割を自動的に解析する自動並列化コンパイラを開発中である[3]．

以下，2 章で，我々の開発中の並列化コンパイラの概要について述べる．3 章で，我々の採用したデータ分割の決定手法について述べる．4 章で，本システムをいくつかのベンチマークプログラムに適用して得られたデータ分割の結果についても報告し，最後に 5 章でまとめとする．

† 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City University
†† 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
††† 三菱電機 (株)
Mitsubishi Electric Corporation

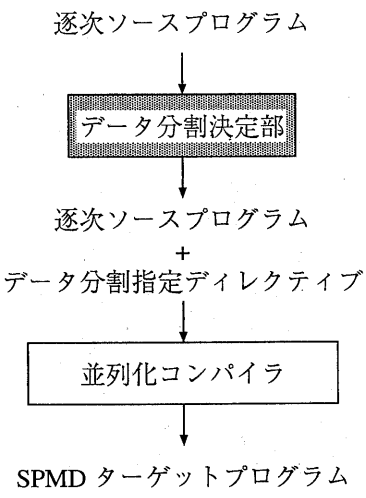

図 1　自動並列化コンパイラ

2. 並列化コンパイラ TINPAR

我々は，自動的にデータ分割を行ない，メッセージ通信のコードの挿入された SPMD(Single Program Multiple Data-stream) コードを生成するコンパイラの実現を目指している．本コンパイラでは，図 1 に示すように，逐次ソースプログラムを入力とし，それを解析して求めたデータ分割ディレクティブを逐次ソースプログラムに付加する．それらを渡された並列化コード生成部には，与えられたディレクティブをもとに Owner Computes Rule☆を適用して SPMD コードを生成する．

従来，プログラム実行中にデータ分割を固定する場合 (静的データ分割) については，Illinois 大学の PARADIGM[4] などで有効な手法が提案されている．

一方，複雑なプログラムでは，プログラム実行中に配列変数の再分割 (動的データ分割) を行なう方が高い性能が得られることもある．Rice 大学の D System[5] では，ループネスト内での最適なデータ分割を求め，隣接ループネスト間での再分割の必要性を解析する手法を提案しているが，この場合は最適なデータ再分割位置を求めるための探索空間が大きくなり，解析には多大な計算量が必要であることが報告されている．

そこで我々は，手続き内では静的データ分割，手続き呼び出し時には必要に応じてデータ再分割を行なうようなデータ分割決定手法を採用することで，解析に多大な計算時間を必要としない自動データ分割決定部を実現した．

我々は，各手続きで，どの配列もデータ分割は 1 種類に限定している．現在は，組み込み関数以外の分割コンパイルは禁止しているので，データ再分割コードはコンパイル時に生成している．組み込み関数は，FORTRAN77 に限った範囲では引数としてスカラ変数，あるいは配列変数の 1 要素しかとらないため，データ再分割の解析対象に含む必要はない．

我々の手法を拡張して，分割コンパイルを許すことにすれば，リンク時に各手続きで引数となっている配列変数のデータ分割情報を集め，必要に応じてデータ再分割のコードを強制的に挿入して再コンパイルすることなどが必要となるであろう．

なお，現在のところ，実装を容易にするため以下のような制限を加えている．

- プロセッサ配列は，1 次元のみとしている．
- COMMON, EQUIVALENCE などの，記憶域結合を扱わない．そのため，手続き間で共通のデータを参照するには必ず引数として渡す必要がある．
- 仮引数として渡されてくる配列の大きさは，明示的に定数で指定する．

3. データ分割決定手法

本章では，まず，手続き内の静的データ分割について述べ，次に手続き間のデータ再分割の解析手法について述べる．

3.1 手続き内データ分割決定

プログラムのある範囲内で，各配列のデータ分割を一意に決定するため，以下のステップで処理を行なう[3]．

(1) 配列次元アラインメントの決定
すべての配列次元の組に対して，それらの次元の分割の有無を同じにすべきかどうかという関係を解析し，それらの関係から配列次元をグループ分けする．このグループを本稿では**配列次元アラインメント**と呼ぶ．

(2) ブロックサイズ比の決定
配列変数の各次元間のブロックサイズ比の制約を求める．

(3) ブロックサイズの決定
上記の各ブロックサイズ比から，実際にブロックサイズを求める．

(4) 分割次元の決定
配列のどの次元をプロセッサ配列へと分割するかを決定する．

これらのステップは，HPF のデータ分割ディレクティブによるデータ分割指定と対応している．ステップ (1) は ALIGN ディレクティブによって，各配列をテンプレート配列にマッピングすることに対応する．このとき，

```
!HPF$ALIGN A(I) ONTO T(2*I)
!HPF$ALIGN B(I) ONTO T(I)
```

☆ 代入文の右辺に現れる演算は，左辺のデータを所有するプロセッサが実行するよう処理をスケジューリングする規則

のように，配列 A と B で分割比を変えることを指定することがステップ (2) のブロックサイズ比の決定に対応する．ステップ (3) と (4) は，DISTRIBUTE ディレクティブで，どの次元をプロセッサ配列に分割するか，BLOCK, CYCLIC, BLOCK-CYCLIC のどの分割が適切か，BLOCK-CYCLIC 分割であれば，ブロックサイズをいくらにするのが適当かを指定することに対応する．

以下，これらの 4 ステップについて，特に重要なステップ (1) と (4) を中心に述べる．詳細については文献 3) を参照されたい．

3.1.1 配列次元アラインメントの決定

本ステップではまず，逐次ソースプログラムに現れる配列変数同士の代入文を解析する．次に，この解析情報から CAG(Component Affinity Graph)[6] と呼ばれるグラフを作成し，それを用いて配列次元アラインメントを決定する．

配列変数同士の代入文の解析において，配列変数の添字式間のアフィン関係 [6] に着目する．ここで，アフィン関数，アフィン関係という用語を以下の意味で用いることにする．

- アフィン関数— $f(i) = a \times i + b$ の形であらわせる関数 f．ただし，a, b は整数であり，$a \neq 0$，i は，ループ変数である．
- アフィン関係— 変数 (上の関数 f では i) が同じアフィン関数どうしを，互いにアフィン関係にあるという．

代入文の左辺に現われる配列変数と右辺に現われる配列変数のそれぞれの次元の添字式の組について，アフィン関係の有無を調べる．添字式の間にアフィン関係があった場合，それらの配列の次元をテンプレート配列の同じ次元にマッピングすると，並列実行時の通信量を減少させることができる．

しかし，必ずしもこのように最適な配列次元アラインメントが求まるわけではなく，配列次元アラインメントに競合が生じる場合もある．図 2 では，1 つ目のループボディの代入文と 2 つ目のループボディの代入文では，それぞれを局所的に見た場合の最適な配列次元アラインメントは異なる．以上の 2 つの例では，配列次元アライメントの 2 つの候補が競合しているが，静的データ分割をする場合は，最終的にどちらかの候補を採用しなければならない．

このように，配列次元アラインメントの候補が競合している場合，メッセージ通信量を最適化するという観点から，プログラム全体として最適な配列次元アラインメントを求めなければならない．このため，配列変数どうしの代入文を解析した情報から CAG を作成し，競合候補から 1 つ候補を選択する問題をグラフを利用して解くことになる．

CAG のノードは配列変数の次元に対応する．ソースプログラムの代入文の解析の結果，異なる配列変数

```
DO j = 0 , n - 1
  DO i = 0 , n - 1
    A(i,j) = F(B(i,j))
  ENDDO
ENDDO
      :
DO j = 0 , m - 1
  DO i = 0 , m - 1
    A(i,j) = F(B(j,i))
  ENDDO
ENDDO
      :
```

図 2 異なる代入文間での競合の例

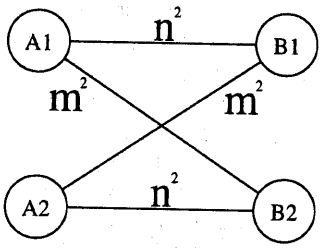

図 3 図 2 に対する CAG

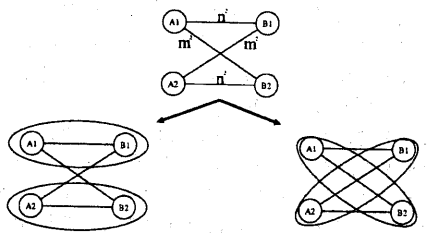

図 4 CAG のノードのグループ分け

の次元間にアフィン関係があった場合，それぞれの次元に対応するノードの間にエッジがつくられる．図 3 に図 2 のプログラムの CAG を示す．

例えば，図 4 の例では，2 つの配列次元アラインメントが考えられる．これらの可能なアラインメントの中から，通信量が最小となるようなものを求めればよい．

ここで，CAG のエッジの重みは，アフィン関係の重要性を示すものであり，それら 2 つの次元を同じグループにすべきであるという情報を表現する．また，CAG において上記の方法に従ってグループ分けをしたとき，異なるグループ間に跨がるエッジは，メッセージ通信の必要性を表し，その重みが通信量を表現するものである．逆に同じグループ内に存在するエッジは，そのアラインメントがエッジの重みの分だけ配列変数

の参照の局所性を反映したものになっているということを表すことになる．

従って，最適な配列次元アラインメントを求めるということは，CAGにおいて，グループ間を結ぶエッジの重みを最小とするようなグループ分けを求めるということに帰着される．

このグループ分けは，NP困難な問題であることが知られているが，次元数が大きい配列から順にグルーピングするヒューリスティックアルゴリズムによって，ほぼ最適に近いグルーピングが求められる[6]ことが知られており，我々もこのヒューリスティックアルゴリズムを採用している．

3.1.2 ブロックサイズ比の決定

本ステップの目的は，並列実行時のプロセッサ間の通信量を最適にするような配列変数間の各次元の分割のブロックサイズについての関係を求めることである．代入文の配列変数の各次元の添字式を解析することにより，2つの次元間のブロックサイズについての制約をブロックサイズの比として求める．

3.1.3 ブロックサイズの決定

本ステップでは，プロセッサ間の負荷バランス，通信量を最適にするようなブロックサイズの決定を行うことを目的とする．今，前ステップで，求まったある配列次元のグループ内の各配列次元のブロックサイズ比を $\alpha_1 : \alpha_2 : \cdots : \alpha_m$ とする．この時，ブロックサイズの決定とは，ある整定数 b_k をテンプレート配列の第 k 次元目に対応づけられた配列次元のグループに対して求め，そのグループに属する配列次元のブロックサイズをそれぞれ $(b_k \times \alpha_1), (b_k \times \alpha_2), \cdots, (b_k \times \alpha_m)$ として求めることである．

この問題に対して，本手法では現在の所，ループの実行回数が外側ループのループ変数によらず一定であるかどうかに関わらず，すべての次元を前ステップで求まったブロックサイズ比を満たす範囲で可能な限り大きなブロックサイズで分割するようにしている．

3.1.4 分割次元の決定

本ステップでは，並列性を最も引き出せるような配列変数の分割次元を求めることを目的とする．

本決定法は，ループ運搬フロー依存情報と，各ループの実行回数から分割次元の決定を行う．

あるループに関してイタレーション間に跨がるフロー依存がある場合，そのフロー依存によって，あるプロセッサの実行が，他のプロセッサの実行をブロックするため，そのループは並列化しない方が良い．逆に，イタレーション間に跨がるフロー依存がない場合は，フロー依存があるループを並列化する場合に比べて，そのループを並列化すると並列性を抽出できるため，そのループを並列化すべきである．

一方，Owner Computes Ruleによって並列化する場合，代入文の左辺に現れる配列変数の，分割される次元の添字式に現れる変数をループ変数とするループのうち，その代入文を囲むループが並列化される．

これらの分割される配列次元と並列化されるループの関係を利用し，並列化すべき(すべきでない)ループを並列化する(しない)ような配列変数の分割次元を求めることが可能である．

以下に具体的な方法を示す．なお，あるループに対して，イタレーション間に跨がるフロー依存の有無によって求められる，そのループの並列実行の可能性を，そのループに対する**ループ分割適性度**，ループ分割適性度から求められる各配列次元を分割すべきであるかどうかを表す値を**配列次元分割適性度**と呼ぶことにする．

(1) ループ分割適性度の計算
まず，各ループに対してループ運搬依存の解析を行なう．次に，フロー依存があるループに対しては，そのループの実行が完了する時間として最悪のケースを想定し，そのループを逐次実行した時の実行時間を反映した値をペナルティとしてつける．ここではその値として，そのループに囲まれるすべてのループネストによってループボディが実行される回数をペナルティとして採用する．すなわち，ループ分割適性度として(ループボディの実行される回数)×(-1)をつける．それ以外のループに対しては0を分割適性度としてつける．

(2) 配列次元分割適性度の計算
ループボディの左辺に現れる配列変数の各次元に対して，その添字式に現れるループ変数のループのループ分割適性度を，その次元の配列次元分割適性度として加える．

(3) 配列次元アラインメントされたグループに対する分割適性度の計算
配列次元アラインメントによってグループ分けされた各グループに対して，そのグループに属する配列次元の配列次元分割適性度の総和を求め，その値をそのグループの分割適性度とする．

上記のグループに対してつけられた分割適性度の中で最大値をとるグループを分割次元として採用する．

配列次元分割適性度によって求める分割次元は，適用可能なプログラムの範囲が広いという特徴を持つ一方，正確な通信量や，演算量などの見積りを行えないため，必ずしも最適なデータ分割を求めるとは限らない．しかし，解析対象のループのうち，ループ運搬依存のないループが存在する場合，そのループを優先的に分割するよう決定する．この決定は妥当なものと考えられる．

なお，配列の分割は，ある程度の要素数をもつ配列に対して行なわなければ並列性を抽出できない．そのため，現在の実装では，要素数が10以下の配列は分割せず，すべてのプロセッサに重複配置するようにしている．

3.2 手続き間データ分割決定

前節で述べた手続き内データ分割をもとに，手続き間のデータ分割の解析を行なう．

呼び出し側の各実引数と呼び出される側の仮引数のうち，配列変数であるものを選ぶ．この配列変数の実引数と仮引数のデータ分割を比較し，異なる場合は呼び出し側でデータ再分割を行なうコードを生成する．

この再分割は，呼び出す直前に，呼び出される側の仮引数のデータ分割へ合わせるためのデータ再分割と，呼び出した直後に呼び出し側の実引数のデータ分割へ戻すためのデータ分割を行なう．

関数呼び出しが DO 文，IF 文，CALL 文中に現れる場合は，一時的なスカラ変数に置き換え，関数呼び出しを含むこれらの文の直前このスカラ変数への代入文を挿入する．この変換により，関数呼び出しにおけるデータ再分割の解析は代入文についてのみ行なえばよいことになる．

4. データ分割決定手法の適用例

本章では，共役勾配法，FFT のプログラムに対し，前章で述べたデータ分割決定手法を適用して得られた結果について述べる．

なお，今回使用したプログラムは，NAS Parallel Benchmarks(NPB)[7] の CG, FT のベンチマークプログラムをもとにした．具体的には，NPB2.3-serial を参考に，2 章で述べたように

- COMMON, EQUIVALENCE などの，記憶域結合を扱わない．そのため，手続き間で共通のデータを参照するには必ず引数として渡す必要がある．
- 仮引数として渡されてくる配列の大きさは，明示的に定数で指定する．

のような条件を満たすように変更を加えたものである．

NPB2.3-serial は，MPI で記述された NPB2.3 の逐次コード版にあたる．以下では，我々のデータ分割決定手法で得られたデータ分割を，NPB2.3 で使われているデータ分割と比べ，その妥当性の判断を行なっている．

4.1 CG

CG では，共役勾配法のプログラムであり，疎行列とベクトルの積がカーネルループとなる．このプログラムでは，手続き間でデータ再分割を行なうようなデータ分割は得られなかった．

主要な配列データの分割を表 1 に示す．

配列 (分割)	説明
A(BLOCK)	疎行列
P(BLOCK)	ベクトル
W(BLOCK)	ベクトル
COLIDX(BLOCK)	疎行列の非零要素を示すインデックス

表 1 CG の主要な配列のデータ分割

```
do j=1,lastrow-firstrow+1
   sum = 0.d0
   do k=rowstr(j),rowstr(j+1)-1
      sum = sum + a(k)*p(colidx(k))
   enddo
   w(j) = sum
enddo
```

図 5 CG のカーネルループ

NPB2.3(MPI 版) conj_grad では，Inspector/Executor[8),9)] の Executor のみ実行するようにしており，Inspector は conj_grad の前に実行して通信パターンを求めている．

我々のデータ分割決定システムでは，このような最適化が行なわれることを前提とした見積もりを行なっていないため，a, p の両方のデータを分割するような解析結果を出力している．

4.2 FT

3 次元 FFT を行なうプログラムである．X 方向，Y 方向，Z 方向のそれぞれの方向で 1 次元 FFT(以下，1D-FFT と呼ぶ) を行なう．この 1D-FFT では，他の 2 つの次元方向については依存がないため，並列実行可能である．例えば，X 方向の 1D-FFT では Y 方向，Z 方向には依存がないため，Y 方向，あるいは Z 方向にデータ分割すれば並列実行可能である．そこで，分割次元を 1 次元とすると，X 方向の 1 次元 FFT のときは，Y 方向にデータ分割，Y 方向，Z 方向の 1D-FFT のときは X 方向にデータ分割することで並列性が抽出できる．データ再分割は，X 方向の 1D-FFT の前後にデータ再分割を行なえばよい．

なお，NPB2.3(MPI 版) では，X, Y 方向の 1D-FFT では Z 方向に分割，Z 方向の 1D-FFT では X 方向に分割している．

いずれも，X, Y, Z, Z, Y, X の順で 1D-FFT を行なうので，この 1 イタレーションで 2 回のデータ再分割が行なわれる．

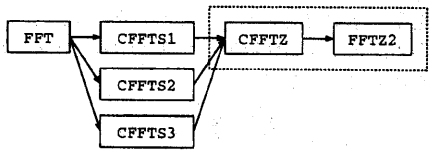

図 6 FFT の手続き呼び出し

X, Y, Z 方向の 1D-FFT は，図 6 に示すように，サブルーチン FFT から呼び出される CFFTS1, CFFTS2, CFFTS3 の 3 つのサブルーチンとして実現されている．これらのサブルーチンからさらに呼び出されるサブルーチンは，各プロセッサ内で逐次的に実行される．

しかし，それら 3 つの手続きから CFFTZ，さらに

FFTZ2 が呼び出されるが，FFTZ2 内にも並列性が存在するため，FFTZ2 を並列実行するようなデータ分割が行なわれてしまう．

現状では，CFFTZ, および FFTZ2 を，サブルーチン CFFTS1, CFFTS2 および CFFTS3 の各サブルーチン内で展開して，強制的に各プロセッサ内で実行させることしかできない．

このようにして，CFFTZ, FFTZ2 を展開して得られたデータ分割を表 6 に示す．

サブルーチン	配列 (分割)	説明
FFT	X1(BLOCK, *, *)	3 次元データ
FFT	X2(BLOCK, *, *)	3 次元データ
CFFTS1	X(*, BLOCK, *)	FFT の X1
CFFTS1	XOUT(*, BLOCK, *)	FFT の X1
CFFTS2	X(BLOCK, *, *)	FFT の X1
CFFTS2	XOUT(BLOCK, *, *)	FFT の X1
CFFTS3	X(BLOCK, *, *)	FFT の X1
CFFTS3	XOUT(BLOCK, *, *)	FFT の X2

表 2 FT の主要な配列のデータ分割

5. おわりに

本稿では，データ並列プログラムに対する，手続き間解析を含む自動データ分割について述べた．

生成したデータ分割ディレクティブ付きプログラムは，コード生成フェーズを改良する予定し，実測する予定である．

また，今後の課題として，実行時間の見積もりにより，FT で指摘したような各プロセッサでローカルに実行される手続きを自動的に判定することなどが挙げられる．

ただし，コンパイル時では，実行時間見積もりの精度を上げることには限界があるため，シミュレーションによる挙動解析や，実機上での実行のプロファイルのデータを用いることになろう．その場合，自動データ分割解析システムは，コンパイラから切り離し，ユーザに対する自動データ分割選択の補助ツールとすることも考えられよう．

謝辞 本研究を進めるにあたり御討論いただいた，広島市立大学津田研究室，京都大学富田研究室の諸氏に感謝いたします．

参考文献

1) High Performance Fortran Forum: High Performance Fortran Language Specification (Ver. 2.0), Technical report, Rice University (1997).
2) 三吉郁夫, 前山浩二, 後藤慎也, 森眞一郎, 中島浩, 富田眞治: メッセージ交換型並列計算機のための並列化コンパイラ TINPAR, 情報処理学会論文誌, Vol. 37, No. 7, pp. 1265–1275 (1996).
3) 辰巳尚吾, 窪田昌史, 五島正裕, 森眞一郎, 中島浩, 富田眞治: 並列化コンパイラ TINPAR における自動データ分割部の実現, 情処研報 96-PRO-8-5, 情報処理学会 (1996). SWoPP'96.
4) Gupta, M. and Banerjee, P.: Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers, *IEEE Trans. Parallel and Distributed Syst.*, Vol. 3, No. 2, pp. 179–193 (1992).
5) Kennedy, K. and Kremer, U.: Automatic Data Layout for High Performance Fortran, *Proc. of Supercomputing* (1995).
6) Li, J. and Chen, M.: Index Domain alignment: Minimizing cost of cross-referencing between distributed arrays, *The 3rd Symposium on the Frontiers of Massively Parallel Computation* (Jaja, J.(ed.)), IEEE Computer Society Press, pp. 424–433 (1990).
7) Bailey, D., Barton, J., Lasinski, T. and Simon, H.: The NAS Parallel Benchmarks, Technical Report NASA Technical Memorandum 103863, NASA Ames Research Center (1993).
8) Koelbel, C. and Mehrotra, P.: Compiling Global Name-Space Parallel Loops for Distributed Execution, *IEEE Trans. Parallel and Distributed Syst.*, Vol. 2, No. 4, pp. 440–451 (1991).
9) 窪田昌史, 三吉郁夫, 大野和彦, 森眞一郎, 中島浩, 富田眞治: 不規則アクセスを伴うループの並列化コンパイル技法–Inspector/Executor アルゴリズムの高速化–, 情報処理学会論文誌, Vol. 35, No. 4, pp. 532–541 (1994).