(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4740561号
(P4740561)
(45)発行日　平成23年8月3日(2011.8.3)　(24)登録日　平成23年5月13日(2011.5.13)

(51)Int.Cl.　F I
G06F 9/45 (2006.01)　G06F 9/44 322G

請求項の数 5 (全 34 頁)

(21)出願番号　特願2004-203063(P2004-203063)
(22)出願日　平成16年7月9日(2004.7.9)
(65)公開番号　特開2006-24088(P2006-24088A)
(43)公開日　平成18年1月26日(2006.1.26)
審査請求日　平成19年7月3日(2007.7.3)

（出願人による申告）平成１５年度、独立行政法人新エネルギー・産業技術総合開発機構、「基盤技術研究促進事業（民間基盤技術研究支援制度）高信頼・低消費電力サーバの研究開発」委託研究、産業再生法第３０条の適用を受ける特許出願

(73)特許権者　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号
(74)代理人　100089118
弁理士　酒井　宏明
(72)発明者　岩下　英俊
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

審査官　坂庭　剛史

最終頁に続く

(54)【発明の名称】トランスレータプログラム、プログラム変換方法およびトランスレータ装置

(57)【特許請求の範囲】
【請求項１】
コンピュータに、並列言語プログラムを解析させ、複数のプロセッサで分散処理される逐次言語プログラムに変換させるトランスレータプログラムであって、
分散に関するパラメタである、分散種別を含む分散パラメタに値を設定するコードを生成する分散パラメタ設定手順と、
複数のプロセッサで共通して使用されるグローバルインデックスとローカルインデックスとを相互に変換する計算式を分散種別毎に記憶する記憶部から、前記分散パラメタ設定手順により生成されたコードにより値が設定される分散パラメタに対応した計算式を参照し、該参照した計算式および当該分散パラメタを用いて、ループで使用されるインデックスの範囲を示すループインデックスおよびループ内の配列で使用されるインデックスを示す配列インデックスについて、分散処理するプロセッサ内でローカル化処理を実行した結果として表されるコードを生成するインデックスローカル化手順と、
をコンピュータに実行させることを特徴とするトランスレータプログラム。
【請求項２】
前記インデックスローカル化手順は、前記ループインデックスが示す初期値および終値を含むループパラメタをローカル化するループパラメタローカル化手順と、ループ内で引用される変数を示すループ変数をグローバルインデックスに変換するループ変数変換手順をコンピュータに実行させることによってループインデックスについて、ローカル化処理を実行した結果として表されるコードを生成することを特徴とする請求項１に記載のトラ

ンスレータプログラム。
【請求項３】
前記並列言語プログラムには、分散種別として不均等ブロック分散および不規則分散を指定可能であることを特徴とする請求項１または２に記載のトランスレータプログラム。
【請求項４】
コンピュータが、並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するプログラム変換方法であって、
分散に関するパラメタである、分散種別を含む分散パラメタに値を設定するコードを生成する分散パラメタ設定工程と、
複数のプロセッサで共通して使用されるグローバルインデックスとローカルインデックスとを相互に変換する計算式を分散種別毎に記憶する記憶部から、前記分散パラメタ設定工程により生成されたコードにより値が設定される分散パラメタに対応した計算式を参照し、該参照した計算式および当該分散パラメタを用いて、ループで使用されるインデックスの範囲を示すループインデックスおよびループ内の配列で使用されるインデックスを示す配列インデックスについて、分散処理するプロセッサ内でローカル化処理を実行した結果として表されるコードを生成するインデックスローカル化工程と、
を含んだことを特徴とするプログラム変換方法。
【請求項５】
並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するトランスレータ装置であって、
分散に関するパラメタである、分散種別を含む分散パラメタに値を設定するコードを生成する分散パラメタ設定手段と、
複数のプロセッサで共通して使用されるグローバルインデックスとローカルインデックスとを相互に変換する計算式を分散種別毎に記憶する記憶部から、前記分散パラメタ設定手段により生成されたコードにより値が設定される分散パラメタに対応した計算式を参照し、該参照した計算式および当該分散パラメタを用いて、ループで使用されるインデックスの範囲を示すループインデックスおよびループ内の配列で使用されるインデックスを示す配列インデックスについて、分散処理するプロセッサ内でローカル化処理を実行した結果として表されるコードを生成するインデックスローカル化手段と、
を備えたことを特徴とするトランスレータ装置。
【発明の詳細な説明】
【技術分野】
【０００１】
この発明は、並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するトランスレータプログラム、プログラム変換方法およびトランスレータ装置に関し、特に、ループインデックスの計算に起因するループの実行効率の低下を防ぐことができるトランスレータプログラム、プログラム変換方法およびトランスレータ装置に関するものである。
【背景技術】
【０００２】
科学技術計算の並列化では、データを複数のプロセッサに分割して処理させるデータの分散処理と、ループを複数のプロセッサに分割して処理させるループの分散処理が基本となる。図２１－１は、データの分散を示す図であり、図２１－２は、ループの分散を示す図である。
【０００３】
図２１－１は、３０個のデータから構成される１次元配列Aを８個、８個、８個および６個のデータから構成される４つのグループに分割し、４つのプロセッサで分散処理する場合を示している。図２１－２は、ループ変数Ｉについての３０回の繰り返し処理を、８回、８回、８回および６回の繰り返し処理に４分割し、４つのプロセッサで分散処理する場合を示している。

【０００４】
このように、科学技術計算では、データおよびループを分散処理することによって、処理の高速化が図られている。なお、データの分割およびループの分割は、インデックスの分割として統一的に扱うことができる。
【０００５】
HPF、OpenMP、VPPFortranなどの並列プログラミング言語では、インデックスの分割を指定することができる。具体的には、VPP Fortranはデータ分割とループ分割を表現する書式を持つ。HPFはデータ分割を表現する書式を持ち、ループ分割は間接的に表現できる。OpenMPはループ分割を表現する書式を持つが、共有メモリ型計算機を対象とするのでデータ分割の表現を持たない。
【０００６】
また、これらの並列プログラミング言語は、複数種類のインデックス分割を提供している。図２２は、インデックス分割による分散の種類を示す図であり、３０要素を４分割する場合を例として示している。同図に示すように、インデックス分割による分散には、ブロック分散、サイクリック分散、ブロック－サイクリック分散、不均等ブロック分散、不規則分散の５種類がある。
【０００７】
それぞれの詳細な意味については、非特許文献１または２に記載されているが、特徴は以下のとおりである。
【０００８】
ブロック分散は、分散次元についてもデータの連続性を残しているので近傍のデータ（配列要素）との相関が大きいアプリケーションに適している。また、計算負荷が計算領域に対して均一でない場合や、並列計算機の能力が不揃いの場合（ヘテロ環境）には、ブロック分散の代りに不均等ブロック分散を使用すれば負荷バランスが調節できる。
【０００９】
また、負荷バランスや計算範囲が実行ループごとに違う場合や実行時まで不明の場合には、サイクリック分散を使えば負荷分散をほぼ均等にできる。しかし、サイクリック分散は近傍のデータとの相関が強い場合に通信を多く発生するので、ブロック分散とサイクリック分散の中間的な方法としてブロック－サイクリック分散がある。
【００１０】
また、空間中を浮遊する粒子に関する計算など、データとプロセッサの対応づけが不規則でテーブルによって管理する必要がある場合には、不規則分散を使用することができる。
【００１１】
図２３－１～図２３－３は、並列プログラミング言語による並列化の例を示す図である。図２３－１は、逐次プログラムを示す図であり、図２３－２は、OpenMPによる並列化を示す図であり、図２３－３は、HPFによる並列化（入力例１）を示す図である。
【００１２】
図２３－２において、４行目のparallel do指示文は、ループdo Iを並列に実行するよう指示している。OpenMPは、データの共有を前提とするので、データの分割配置に関する指示文は持たない。
【００１３】
図２３－３において、２行目と３行目は、変数Aの１次元目方向を均等に４ブロックに分割して４プロセッサに配置することを指示している。６行目のindependent指示文は、ループdo Iが並列実行可能であることを指示している。
【００１４】
それぞれ、ループの並列実行のためには、各プロセッサが実行するループインデックスの範囲を決定する必要がある。OpenMPではループの分割方法は言語仕様によって決められていて、インデックス範囲を機械的に分割する。HPFでは、ループ内でアクセスされる変数（この例ではA）の分割配置となるべく整合性が取れるように、コンパイラが自動的に

分割を決定する。

【００１５】

このように、並列化されたプログラムに対して、コンパイラは複数のプロセッサが実際に並列実行できるコードを生成する。このとき、コンパイラは、データもループもインデックス範囲を分割してプロセッサに割り振るが、データのインデックス付けを変換することで宣言形状をプロセッサ間で同じにし、SPMD(Single Program/Multiple Data)方式で静的なデータ割付を可能にする。

【００１６】

図２４－１は、データおよびループのインデックス分割とプロセッサとの対応を示す図である。同図は、1000×1000の配列a(i,j)に対して、１行～２５０行までのデータが０番のプロセッサP(0)に割り当てられ、２５１行～５００行までのデータが１番のプロセッサP(1)に割り当てられ、５０１行～７５０行までのデータが２番のプロセッサP(2)に割り当てられ、７５１行～１０００行までのデータが３番のプロセッサP(3)に割り当てられることを示している。

【００１７】

図２４－２は、図２３－３に示した並列言語プログラムに対して従来のコンパイラが出力するコードを示す図である。同図において、myIDはプロセッサの番号であり、myIDを用いて各プロセッサの担当インデックス範囲の下限値myLBおよび上限値myUBが計算され、これら下限値および上限値が繰り返し範囲の指定およびデータのアクセスに使用される。

【００１８】

ここで、データa(i,j)をアクセスする場合に、変換前の入力プログラムのインデックスであるグローバルインデックスIから変換後のプログラムのインデックスであるローカルインデックスiへの変換i=gtol(I)=I-myLB+1=I-250*myIDが実行時に必要となる。

【００１９】

この変換gtolは、分散の種類やパラメタ（インデックスの長さ、分割数など）によって異なる関数となり、分散種別によっては複雑な計算式となるため、実行時ライブラリの呼出しによって実現されることも多い。なお、不規則分散の場合には一つの式では表現できず、テーブルの参照となる。

【００２０】

【非特許文献１】High Performance Fortran Forum, "High Performance Language Specification Version 2.0.", 1997

【非特許文献２】富士通, 日立, 日本電気訳、「High Performance Fortran 2.0公式マニュアル」、シュプリンガー・フェアラーク東京、ISBN4-431-70822-7、1999年

【非特許文献３】Japan Association for High Performance Fortran (JAHPF),"HPF/JA Language Specification Version 1.0", November 1999、［平成１６年６月１８日検索］、インターネット＜URL: HYPERLINK http://www.hpfpc.org/jahpf/spec/hpfja-v10-eng.pdf http://www.hpfpc.org/jahpf/spec/hpfja-v10-eng.pdf＞

【非特許文献４】OpenMP Architecture Review Board, "OpenMP Fortran Application Program Interface Version 2.0", November 2000、［平成１６年６月１８日検索］、インターネット＜URL: HYPERLINK http://www.openmp.org/specs/mp-documents/fspec20.pdf http://www.openmp.org/specs/mp-documents/fspec20.pdf＞

【非特許文献５】OpenMP Architecture Review Board, "OpenMP C/C++ Application Program Interface Version 1.0", October 1998、［平成１６年６月１８日検索］、インターネット＜ HYPERLINK http://www.openmp.org/specs/mp-documents/cspec10.pdf http://www.openmp.org/specs/mp-documents/cspec10.pdf＞

【発明の開示】

【発明が解決しようとする課題】

【００２１】

図２４－２に示したコードでは、グローバルインデックスIからローカルインデックスiへの変換gtolは、比較的単純であった。しかし、このような場合でさえも、インデックス

変換のない場合の変数アクセスのコスト（load命令1回分程度）と比較して、インデックス変換のコスト（数演算の実行）はかなり大きいため、その部分の実行時間は数倍に増加する場合がある。

【００２２】

より一般的な場合（ブロック分散以外の分散、ループの初期値・終値が配列の宣言形状と不一致、ループに1以外の増分値、インデックスの大きさがプロセッサ数で割り切れないなど）、ループを並列化した結果はより複雑になり、実行効率が低下するという問題がある。

【００２３】

例えば、図２５－１は、ブロック－サイクリック分散を用いる並列言語プログラムの例を示す図である。また、この例では、ループの上下限値が配列の上下限値と異なる。このとき、グローバルインデックスとローカルインデックスの対応は、図２５－２に示すようになる。

【００２４】

図２５－２において、プロセッサP(0)では、グローバルインデックス「１～５」にローカルインデックス「１～５」が対応し、グローバルインデックス「２１～２５」にローカルインデックス「６～１０」が対応し、グローバルインデックス「９８１～５」にローカルインデックス「２４６～２５０」が対応する。

【００２５】

図２５－３は、図２５－１に示した並列言語プログラムに対して生成される従来のコードを示す図である。同図に示すように、インデックス変換式は、(I-1)/20*5+MOD(I-1,5)+1となる（除算は余りを切り捨てる整数除算）。また、並列ループについて、各プロセッサの担当範囲は５要素を１ブロックとする飛び飛びの範囲となるため、１重のDOループでは表現できず複雑な２重ループとなっている。このような出力コードの複雑化は、実行効率の低下を招きやすい。

【００２６】

図２６－１は、別の分散の例として、不規則分散を用いる並列言語プログラムの例を示す図である。この場合、グローバルインデックスとローカルインデックスの対応は図２６－２に示すようになる。

【００２７】

図２６－３は、図２６－１に示した並列言語プログラムに対して生成される従来のコードを示す図である。このコードでは、インデックス変換は式では表現できず、コンパイラが生成したテーブルGTOLの参照となっている。

【００２８】

また、プロセッサ毎に実行するループインデックスの値は、FORTRANのDO文では表現することができないため、IF文によって実行が必要な反復を選び出している。つまり、全てのプロセッサで全てのループ範囲を実行するという無駄が生じる。この無駄はプロセッサ数が多くなるほど相対的に大きくなる。

【００２９】

この発明は、上述した従来技術による問題点を解消するためになされたものであり、ループインデックスの計算に起因するループの実行効率の低下を防ぐことができるトランスレータプログラム、プログラム変換方法およびトランスレータ装置を提供することを目的とする。

【課題を解決するための手段】

【００３０】

上述した課題を解決し、目的を達成するため、本発明は、並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するトランスレータプログラムであって、分散に関するパラメタである分散パラメタに値を設定するコードを生成する分散パラメタ設定手順と、前記分散パラメタ設定手順により生成されたコードにより値が設定される分散パラメタを用いてループインデックスおよび配列インデックスをロー

カル化するコードを生成するインデックスローカル化手順と、をコンピュータに実行させることを特徴とする。

【００３１】

また、本発明は、並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するプログラム変換方法であって、分散に関するパラメタである分散パラメタに値を設定するコードを生成する分散パラメタ設定工程と、前記分散パラメタ設定工程により生成されたコードにより値が設定される分散パラメタを用いてループインデックスおよび配列インデックスをローカル化するコードを生成するインデックスローカル化工程と、を含んだことを特徴とする。

【００３２】

また、本発明は、並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するトランスレータ装置であって、分散に関するパラメタである分散パラメタに値を設定するコードを生成する分散パラメタ設定手段と、前記分散パラメタ設定手段により生成されたコードにより値が設定される分散パラメタを用いてループインデックスおよび配列インデックスをローカル化するコードを生成するインデックスローカル化手段と、を備えたことを特徴とする。

【００３３】

かかる発明によれば、分散に関するパラメタである分散パラメタに値を設定するコードを生成し、生成したコードにより値が設定される分散パラメタを用いてループインデックスおよび配列インデックスをローカル化するコードを生成するよう構成したので、ループインデックス変換に起因するオーバヘッドを削減することができる。

【発明の効果】

【００３４】

本発明によれば、ループインデックス変換に起因するオーバヘッドを削減するので、ループの実行効率の低下を防ぐことができるという効果を奏する。

【発明を実施するための最良の形態】

【００３５】

以下に添付図面を参照して、この発明に係るトランスレータプログラム、プログラム変換方法およびトランスレータ装置の好適な実施例を詳細に説明する。なお、本実施例では、本発明をHPFを用いて作成された並列言語プログラムに適用した場合を中心に説明する。また、ここでは、数学記号を図２７に示すように定義し、使用する用語の定義は以下のとおりである。

【００３６】

SMP（Symmetric Multiprocessor）

複数CPUが同じメモリ空間を共有する並列実行。データの分散を考える必要がなく、すべてのデータはすべてのCPUから直接アクセス可能である。共有メモリ型計算機は通常SMP型の計算機であると言える。

【００３７】

SPMD（Single Program/Multiple Data）

複数のプロセッサに同じプログラムを与えて、プロセッサIDなどのパラメタの違いによって処理を分担させ、全体として並列実行を行わせる方法。

【００３８】

恒等的な整列

m=1，n=0の整列。

【００３９】

整列

Aの第d次元がBの第e次元に整列するとは、Aの第d次元のインデックスiからBの第e次元のインデックスjに一定の対応付けがあり、対応する配列要素どうしは同じプロセッサに対して分散されることを言う。HPFで定義される整列は、

$$j = m\ i + n \quad (m, n\text{は整数、}m \neq 0)$$

という形であり、ARIGN指示文を使って以下のように表現する。
ALIGN A(i) WITH B(m*i+n)
HPFでは、ARIGN指示文によって、配列データから配列データまたはテンプレートへの整列が表現できる。また、ON指示文によって間接的に、ループから配列データまたはテンプレートへの整列が表現できる。VPP Fortranでは、配列データからテンプレート、ループからテンプレートへの整列が表現できる。OpenMPでは整列の表現はないが、並列ループからテンプレートへ常に恒等的な整列（m=1, n=0の整列）があると思えばよい。

【００４０】

テンプレート

配列要素やループの反復をプロセッサに分散するとき、仲介となる仮想的な配列。HPFの言語仕様（非特許文献１および２参照。）で定義されていて、VPP Fortranのインデックス（非特許文献５参照。）の概念にも近い。テンプレートは、図２２に例示したような分散方法でプロセッサと対応づけられる。変数やループはテンプレートに対して整列することができる。

【００４１】

プロセッサ

並列計算処理を実行する仮想的な単位。プロセッサは任意の次元数の矩形状に配置することができ、プロセッサの総数が並列度となる。分散メモリ型計算機では、各プロセッサは固有のメモリ空間を持つ。共有メモリ型計算機(SMP)では、メモリ空間はプロセッサ間で共有される。プロセッサと物理的なCPUの対応は、言語仕様で定義されることもあるし、コンパイラやオペレーティングシステムが決定することもある。複数のプロセッサの実行が物理的には1つのCPUで実現される場合もある。複数のプロセッサ配列が定義されるとき、その配列要素間の対応もまた、言語仕様で定義されることもあるし、コンパイラやオペレーティングシステムが決定することもある。

【００４２】

ローカルアクセス

プロセッサ間の通信を伴わないデータのアクセス。参照の場合にはアクセスするプロセッサがそのデータを所有していることが条件であり、定義の場合には、それに加えて、複製を持つすべてのプロセッサが同じ値を定義することが条件となる。分散されていない変数（プロセッサに分割配置されず全要素のコピーが配置されている変数）は、分散されたDOループの中では、参照はローカルアクセスだが、定義はローカルアクセスでない。

【００４３】

また、分散種別と共に分散を特徴づけるパラメタには以下のものがある。

Nd：テンプレートの次元dの寸法。言い換えると、分割されるインデックスの長さ。例えば、HPFプログラムで、

```
!HPF$ PROCESSORS Q(4,6)
!HPF$ TEMPLATE U(100,0:29)
!HPF$ DISTRIBUTE U(block,cyclic) ONTO Q
```

とある場合、N1=100、N2=30となる。dを省略し単にNと書くこともある。

【００４４】

Pd：テンプレートの次元dに対応するプロセッサ群の次元の寸法。言い換えると、インデックスの分割数。分散先プロセッサ群が一次元配列の場合には、P1はプロセッサ群を構成するプロセッサの数となる。例えば、上の例で、P1=4、P2=6となる。ここではテンプレートのd次元目はプロセッサ群のd次元目に対応するよう正規化されていると仮定する。dを省略し単にPと書くこともある。

【００４５】

pd：テンプレートの次元dに対応するプロセッサ群の次元の中の自プロセッサの位置（$0 \leqq pd < Pd$）。例えば、上の例で、プロセッサQ(1,3)はp1=1、p2=3となる。pdの値は実行時にライブラリ呼出しなどによって得られる仮想プロセッサ番号から計算できる。dを省略し単にpと書くこともある。

【００４６】
ｗ：ブロック分散またはブロック－サイクリック分散のときのブロック幅。ブロック分散でユーザ指定のない場合には、HPFとOpenMPでは
【数１】

$$w = \lceil N/P \rceil \qquad \cdots(1)$$

と定義される。
【００４７】
Ｗ：不均等ブロック分散のときのブロック幅配列。長さPのベクトル。
W(p) (0 ≦ p < P)はプロセッサpに割り当てられるインデックスの幅となる。W(0) = W(1) = ... = W(P-1) = wであるとき幅wのブロック分散を意味する。
【００４８】
Ｍ：不規則分散のときの正規化されたマッピング配列。長さNdのベクトル。
M(k) (0 ≦ k < N)はテンプレートの各要素に対応するプロセッサの要素番号であり、0 ≦ M(k) < Pとなる。
【００４９】
また、以下は上記のパラメタから誘導されるパラメタである。
Ｂ：不均等ブロック分散のときのグローバルインデックスでの下限値の配列。長さPのベクトル。以下のように定義される。
【数２】

$$B(p) = \sum_{k=0}^{p-1} W(k) \qquad \cdots(2)$$

以降、これらを合せて分散パラメタと呼ぶ。
【００５０】
なお、それぞれの手順の中に数式を作成する処理が数多くあるが、定数評価や数式処理的な技法を使って、出来る限り簡単化した式として生成するか、コード生成後に定数伝播・畳み込みなどの最適化技術によりできる限り簡単化することが望ましい。
【実施例】
【００５１】
まず、本実施例に係るプログラム変換による並列化の例について説明する。図１－１は、本実施例に係るプログラム変換による並列化の例（１）を示す図である。この例は、図２３－３に示した並列言語プログラム（入力例１）を並列化したものである。
【００５２】
同図に示すように、本実施例に係るプログラム変換では、配列データだけでなくループについてもインデックスをローカル化することにより、分散配列のループ内のローカルアクセスでは添字式の変換を全く必要としないコードを生成する。具体的には、配列aにアクセスする場合に、ローカルインデクスiを用いてアクセスするコードを生成する。
【００５３】
図１－２は、本実施例に係るプログラム変換による並列化の例（２）を示す図である。この例は、図２５－１に示した並列言語プログラム（入力例２）を並列化したものである。同図に示すように、ブロック－サイクリック分散を用い、ループの上下限値が配列の上下限値と異なる場合にも、図２５－３に示した従来の並列化のように複雑なインデックス変換式や２重ループを用いることなく、添字式の変換を全く必要としないコードを生成する。
【００５４】
図１－３は、本実施例に係るプログラム変換による並列化の例（３）を示す図である。この例は、図２６－１に示した並列言語プログラム（入力例３）を並列化したものである。同図に示すように、不規則分散を用いる場合にも、図２６－３に示した従来の並列化のようにテーブルGTOLを参照することなく、添字式の変換を全く必要としないコードを生成

する。

【００５５】

このように、本実施例に係るプログラム変換では、配列データだけでなくループについてもインデックスをローカル化することにより、ループインデックスの計算に起因するループの実行効率の低下を防ぐことができる。

【００５６】

次に、本実施例に係るトランスレータプログラムの構成について説明する。図２は、本実施例に係るトランスレータプログラムの構成を示す機能構成図である。同図に示すように、このトランスレータプログラム１００は、並列言語パーサ１１０と、最適化・正規化処理部１２０と、コード生成部１３０とを有する。

【００５７】

並列言語パーサ１１０は、HPFプログラムを入力して構文解析を行うパーサである。この並列言語パーサ１１０は、例えば、図２３－３、図２５－１、図２６－１に示したHPFプログラムを入力する。

【００５８】

最適化・正規化処理部１２０は、並列言語パーサ１１０による構文解析結果に基づいてHPFプログラムの最適化および正規化を行う処理部である。

【００５９】

コード生成部１３０は、最適化・正規化処理部１２０により最適化および正規化が行われたHPFプログラムに対して、複数のプロセッサで分散処理されるFORTRANプログラムを生成する処理部である。

【００６０】

このコード生成部１３０は、ループインデックスをローカル化することによって、実行効率の良いFORTRANプログラムを生成し、生成されたFORTRANプログラムは、FORTRANコンパイラ１０によってコンパイルされ、実行可能ファイルが生成される。なお、FORTRANコンパイラ１０は、MPI(Message Passing Library)などの実行時ライブラリを結合して実行可能ファイルを生成する。

【００６１】

コード生成部１３０は、分散パラメタ設定部１３１と、ループインデックスローカル化部１３２と、配列形状ローカル化部１３３とを有する。

【００６２】

分散パラメタ設定部１３１は、テンプレートTの全ての分散次元dについて、分散パラメタｐを実行時に計算して生成変数に設定する実行コードを生成する処理部である。生成する場所は各手続の入口である。あるいは、主プログラムの入口で外部変数に設定して全手続から見えるようにすることもできる。

【００６３】

なお、他の分散パラメタについては、同様に生成変数を設けて値を設定する実行コードを生成することもできる。ただし、コンパイル時に値が求められるパラメタは、実行コードを生成しない。

【００６４】

また、この分散パラメタ設定部１３１は、不規則分散の場合に、グローバルインデックスからローカルインデックスを得るためのテーブルとして形状(0:N)の１次元配列を宣言し、図３に示す漸化式で定義される値を設定するコードを生成する。ここで、配列名はテンプレートの次元毎に一意の名前とする（ここではgtolとする）。なお、このテーブルの値はプロセッサによって異なる。

【００６５】

また、この分散パラメタ設定部１３１は、不規則分散の場合に、ローカルインデックスからグローバルインデックスを得るためのテーブルとして形状(0: n-1)の領域を確保し、M(k)の値が自プロセッサ番号pと一致するkの値を昇順に並べたものを設定するコードを生成する。ここで、テーブルのサイズnは、n=Nとしてもよいが、少なくともこの設定で溢れ

ないだけの大きさがあれば十分である。また、配列名はテンプレートの次元毎に一意の名前とする（ここではltogとする）。

【００６６】

ループインデックスローカル化部１３２は、ループインデックスをローカル化する処理部であり、ループパラメタローカル化部１３２ａと、ループ変数変換部１３２ｂとを有する。

【００６７】

ループパラメタローカル化部１３２ａは、テンプレートTの分散次元dに整列するループLについて、グローバルインデックスをローカルインデックスに変換するためのコードをループの前に生成する処理部である。

【００６８】

図４－１は、ループパラメタローカル化部１３２aが生成するコードを示す図である。同図に示すように、ループパラメタローカル化部１３２aは、グローバルインデックスの三つ組I1:I2:I3からローカルインデックスの三つ組i1:i2:i3を求めるコードをループの前に生成する。なお、グローバルインデックスの三つ組I1:I2:I3からローカルインデックスの三つ組i1:i2:i3を得る計算式は数学的に求めることができ、その解の一例については後述する。また、i1、i2、またはi3が１つの式の評価だけで計算できる場合、その式で直接ループパラメタを置き換えてもよい。

【００６９】

図４－２は、ループパラメタに増分が存在しない場合に、ループパラメタローカル化部１３２aが生成するコードを示す図である。同図に示すように、ループパラメタローカル化部１３２aは、グローバルインデックスの二つ組I1:I2からローカルインデックスの二つ組i1:i2を求めるコードをループの前に生成する。なお、グローバルインデックスの二つ組I1:I2からローカルインデックスの二つ組i1:i2を得る計算式は数学的に求めることができ、その解の一例についても後述する。また、i1またはi2が１つの式の評価だけで計算できる場合、その式で直接ループパラメタを置き換えてもよい。

【００７０】

図４－３は、不規則分散の場合に、ループパラメタローカル化部１３２ａが生成するコードを示す図である。同図に示すように、ループパラメタローカル化部１３２ａは、分散パラメタ設定部１３１が生成したグローバルインデックスからローカルインデックスを求めるテーブルgtolを用いて、ループ初期値および終値を置き換える。

【００７１】

なお、ループパラメタに増分が存在する場合、ループパラメタのローカル化の直前に、図５に示すようにループを変換することにより、増分を消し去ることができる。

【００７２】

ループ変数変換部１３２ｂは、ループLの中でループ変数の引用を検出し、それぞれグローバルインデックスに変換する式に置き換える処理部である。ただし、そのループ変数がローカルアクセスの添字式として引用されていて、かつ、その次元で配列がテンプレートTの次元dに整列している場合には、この変換は行わない。なお、ローカルインデックスをグローバルインデックスに変換する関数は数学的に求めることができ、その解の一例についても後述する。

【００７３】

配列形状ローカル化部１３３は、テンプレートTに整列する分散配列Aについて、配列インデックスのローカル化を行う処理部である。すなわち、この配列形状ローカル化部１３３は、テンプレートTのすべての分散している次元dに対して、配列Aの次元dに整列している次元について、下限を0に置き換え、上限をlsize_supより1小さい値を表現するコードに置き換える。

【００７４】

lsize_supは、$\max\left[\ lsize_q \mid 0 \leqq q < P\ \right]$ 以上の任意の値であり、メモリ領域の無駄を小さくするためにはできるだけ小さな値であることが望ましい。ここで、$lsize_q$は、プロ

セッサqに分割割当てされるその次元のインデックスの数である。lsize_supは数学的に求めることができ、その解の一例については後述する。

【００７５】

なお、不規則分散の場合には、$lsize_q$は、M(k) = qとなる要素数だが、記憶領域の無駄を厭わないのであれば、lsize_sup=Nとしてもよい。

【００７６】

なお、ここでは、不規則分散の場合に、分散パラメタ設定部１３１が、グローバルインデックスからローカルインデックスを得るためのテーブルおよびローカルインデックスからグローバルインデックスを得るためのテーブルを生成するが、ループパラメタローカル化部１３２ａが各ループに対して行うこともできる。その場合、コードの生成場所は手続の入口ではなく各ループの直前とする。

【００７７】

次に、コード生成部１３０の処理手順について説明する。図６は、コード生成部１３０の処理手順を示す流れ図である。同図に示すように、このコード生成部１３０は、全てのテンプレートTについて、分散パラメタの設定を行い（ステップＳ１００）、ループインデックスのローカル化を行い（ステップＳ２００）、配列インデックスのローカル化を行う（ステップＳ３００）。

【００７８】

また、ステップＳ２００のループインデックスのローカル化では、Tの全ての分散している次元dについて、dに整列する全てのループLについて、ループパラメタのローカル化を行い（ステップＳ２１０）、ループ変数引用の変換を行う（ステップＳ２２０）。

【００７９】

また、ステップＳ３００の配列インデックスのローカル化では、Tに整列する全ての配列Aについて、Tの全ての分散している次元dについて、配列形状のローカル化を行う（ステップＳ３１０）。

【００８０】

このように、このコード生成部１３０が、全てのテンプレートTについて、分散パラメタの設定、ループインデックスのローカル化、配列インデックスのローカル化を行うことによって、実行効率の良いコードを生成することができる。

【００８１】

次に、グローバルインデックスとローカルインデックスの相互変換関数について説明する。図７は、分散Dの属性を示す図である。同図に示すように、分散Dは、分散種別d、プロセッサ数P、配列の寸法N、ブロック幅w、ブロック幅配列W、分散下限値Bおよびマッピング配列Mを有する。

【００８２】

ブロック分散の場合、以下のすべての関数で、自プロセッサ番号がpのときの部分式p wをmyLBに置き換えてもよい。不均等ブロック分散の場合、以下のすべての関数で、自プロセッサ番号がpのときの部分式B(p)をmyLBに置き換えてもよい。ただし、myLBはそれぞれp wおよびB(p)を自プロセッサ番号pについて事前に評価した値を持つ変数とする。

【００８３】

gtolは、分散Dについて、グローバルインデックスIに対応するプロセッサpのローカルインデックスiを得る関数であり、図８に各分散種別dに対応するgtolの定義を示す。

【００８４】

ltogは、分散Dについて、プロセッサpのローカルインデックスiに対応するグローバルインデックスIを得る関数であり、図９に各分散種別dに対応するltogの定義を示す。

【００８５】

gtol2は、(i1,i2) = gtol2(D,I1,I2,p)の形式で用いられ、分散Dについて、グローバルインデックス区間I1:I2に対応する、プロセッサpにおけるローカルインデックス区間i1:i2のi1とi2を得る関数であり、図１０－１～図１０－２に各分散種別dに対応するgtol2の定義を示す。

【００８６】
なお、図１０－１に示したブロック分散の他の実現方法のうち最初の方法は、元の方法に比べて場合分けが多いので生成コードが大きくなり実行時の判断分岐が多くなるが、並列化した後のループの上下限値が -1以上w以下であることが保証されるので、例えば4バイト整数では表現できないような巨大ループを分割して4バイト整数で扱えるようにするなど、スケーラビリティを重視する場合に適する。
【００８７】
gtol3は、(i1,i2,i3) = gtol3(D,I1,I2,I3,p)の形式で用いられ、分散Dについて、グローバルインデックス三つ組I1:I2:I3に対応する、プロセッサpにおけるローカルインデックス三つ組i1:i2:i3のi1とi2とi3を得る関数であり、図１１－１～図１１－３に各分散種別dに対応するgtol3の定義を示す。
【００８８】
なお、サイクリック分散では一般にi1,i2の値を場合分け（IF文）と式による評価（代入文）だけで生成することはできないので、動的に計算するための実行時ライブラリを用意してオブジェクトと結合するか、または、インライン展開してソースに挿入するか、または、FORTRANなどの言語で実行手続をコンパイル時に生成して、オブジェクトと結合する（特願２０００－５９６４５１参照。）。
【００８９】
Lsize_supは、n_sup = lsize_sup(D)の形式で用いられ、分散Dの分散後の大きさのうち、プロセッサ間で最大のものと等しいか、それよりも大きな値で、できるだけ小さな値を得る関数であり、図１２に各分散種別dに対応するLsize_supの定義を示す。
【００９０】
次に、図２３－３に示した入力例１に対するコード生成部１３０の処理について説明する。図１３－１は、図２３－３に示した入力例１に対してコード生成部１３０が入力する内部表現のイメージを示す図である。
【００９１】
図１３－１に示すように、トランスレータプログラム１００は、内部表現に落す際に、ループの分散は配列Aの1次元目に沿って行うという判断を加え、配列の下限値とDOループの初期値は0に正規化している。なお、この正規化は必須の処理ではなく、代りにグローバルインデックス／ローカルインデックスの相互変換式に下限値の考慮を加えて修正してもよい。または、特願２００３－３１６１３７に記載した発明を使うなどして正規化すればよい。
【００９２】
また、前処理により、Aのアクセスは３個所ともローカル、つまりプロセッサ間通信なしでアクセスできることが保証されているとする。なお、ローカルアクセスの検出は特願２００３－３１６１３７に記載した発明などで可能であり、ローカルでないアクセスを並列ループ外へ出す方法は、特許第３２６１２３９号に記載された技術やその他の既存技術が使用できる。また、ここでは、HPFプログラムを入力例としたが、自動並列化コンパイラの解析結果やツールによる生成コードをコード生成部１３０の入力としてもよい。
【００９３】
［ステップＳ１００］
プロセッサ番号pを変数myIDに格納するため、手続の入口に以下のコードを挿入する。この値にはMPIライブラリが返す仮想プロセッサ番号をそのまま使用している。

```
CALL mpi_comm_rank(MPI_COMM_WORLD,myID,...)
```

ブロック分散の他のパラメタはそれぞれ、
【数３】

$$N = 1000, P = 4, w = \lceil N/P \rceil = 250 \qquad \cdots(3)$$

と定数となるので、実行コードとしては生成しない。
【００９４】

〔ステップＳ２００〕
テンプレートに整列するループはdo Iだけである。
［ステップＳ２１０］ループインデックスはI1 = 0, I2 = 999であるので、変換後の初期値i1と終値i2はブロック分散のgtol2関数よりそれぞれ以下のように計算できる。これに対応するコードをループの直前に生成する。

```
i1 = -250*myID      for  myID ≦ 0
   = 0              for  myID ＞ 0
i2 = 249            for  myID ＜ 3
   = 999-250*myID   for  myID ≧ 3
```

［ステップＳ２２０］ループ内でループ変数Iは３個所で引用されているが、いずれも変換を行わない条件に該当する。

【００９５】
〔ステップＳ３００〕
対象変数はAだけで、分散次元は１次元目である。w = 250 であるので、Aの１次元目の形状は0:249とする。

【００９６】
以上の変換を施した結果を、図１３－２に示す。同図で太字で示した範囲が新しく生成または置換された部分である（以下同様）。さらに定数伝播・定数畳み込みなどの既存の最適化を施すことにより、図１３－３のようなコードに簡単化できる。これは図１－１で示した結果と一致する。図２４－２に示した従来のコンパイラが出力するコードと比較して、配列アクセスの添字式が簡単になることが分かる。

【００９７】
次に、図２５－１に示した入力例２に対するコード生成部１３０の処理について説明する。図１４－１は、図２５－１に示した入力例２に対応してコード生成部１３０が入力する内部表現のイメージを示す図である。

【００９８】
〔ステップＳ１００〕
入力例１と同様、myIDにpを設定するコードを生成する。N = 1000, P = 4, w = 5である。

【００９９】
〔ステップＳ２００〕
［ステップＳ２１０］I1 = 1, I2 = 998であるので、変換後の初期値i1と終値i2はブロックーサイクリック分散のgtol2関数よりそれぞれ図１４－２に示すように計算できる。これに対応するコードをループの直前に生成する。pの取り得る範囲が0以上P未満であることから、i1の計算におけるp<0のケースと、i2の計算におけるp>3のケースは省略することができる。
［ステップＳ２２０］変換を行わない。

【０１００】
〔ステップＳ３００〕
【数４】

$$w = \lceil N/(Pw) \rceil = 250 \qquad \cdots(4)$$

であるので、Aの１次元目の形状は0:249とする。

【０１０１】
以上の変換を施した結果を、図１４－３に示す。さらに外側ループ内のIFブロックをループの外に移動する最適化を施せば、図１－２で示した結果と一致する。図２５－３に示した従来のコードではDOループネストが深くなることでループ実行のコストが増加するのに対し、この例ではDOループの深さを変化させない。また、添字式も複雑にならない。

【０１０２】
次に、図２６－１に示した入力例３に対するコード生成部１３０の処理について説明す

る。図１５－１は、図２６－１に示した入力例３に対応してコード生成部１３０が入力する内部表現のイメージを示す図である。

【０１０３】

〔ステップＳ１００〕

入力例１と同様にmyIDにプロセッサ番号pを設定する実行コードを生成する。また、以下のコードにより、実行時にmapArray(0:11)にマッピング配列Mを設定させる。

```
mapArray = MAP-1
または
do I=0,11
  mapArray(I) = MAP(I-1)-1
end do
```

【０１０４】

ここでMAP-1とするのは、プロセッサ配列の下限値を0に正規化するためのシフトである。また、変換テーブルgtol(0:12)とltog(0:11)を宣言し、値を設定するための実行コードを以下のように生成する。

```
      gtol(0) = 0
      i = 0
      do k=0,11
        if (myID.EQ.mapArray(k)) then
          ltog(i) = k
          i = i+1
        end if
        gtol(k+1) = i
      end do
```

【０１０５】

〔ステップＳ２００〕

DO文の初期値Mをgtol(M)に、終値Nをgtol(N+1)-1に、それぞれ置換する。ループ内で変数Iは４個所で引用されていて、うち一個所だけは変換を行わない条件に該当しないので変換する。すなわち、３番目に出現するIだけをltog(I)に置換する。

【０１０６】

〔ステップＳ３００〕

N=12であるので、Aの１次元目の形状は0:11とする。

【０１０７】

以上の変換を施した結果を、図１５－２に示す。これは図１－３で示した結果と一致する。図２６－３で示した従来のコードでは全プロセッサが内側ループの全範囲を実行することになり並列効果が得られにくいのに対し、この例では各プロセッサが担当範囲だけの実行に縮小されている。

【０１０８】

次に、サイクリック分散に対するコード生成部１３０の処理について説明する。図１６－１は、コード生成部１３０が入力する内部表現のイメージを示す図である。

【０１０９】

〔ステップＳ１００〕

入力例１と同様、myIDにpを設定するコードを生成する。N = 24, P = 6である。

【０１１０】

〔ステップＳ２００〕

［ステップＳ２１０］I1 = 4, I2 = 20, I3 = 4であるので、変換後の初期値i1、終値i2、および増分i3はサイクリック分散のgtol3関数より以下のように計算する。

（１）t = GCD(P, |I3|) = GCD(6, 4) = 2をコンパイル時の計算で得る。

（２）図１７を実現するソースコードをコンパイル時にFORTRAN外部関数として生成する。ここでは、関数名をcyclic_fとし、解なしの場合に-1を返すインタフェースとしている

。
（３）cyclic_fを呼び出して戻り値を変数n_cycに代入するコードをDOループの前に生成する。
（４）n_cycの値が-1でない場合に限ってDOループが実行されるように、DOループ全体をIFブロックで囲む。
（５）DOループパラメタをそれぞれ変数k1,k2,k3に置換し、k1,k2,k3の値を求めるコードをDOループの直前に生成する。
［ステップＳ２２０］ループ内で変数kは２個所で引用されていて、うち右辺式の出現は変換を行わない条件に該当しないので変換する。すなわち、式i*j*kのkをltog関数に従い(6*k+myID)に置き換える。

【０１１１】
〔ステップＳ３００〕
対象変数Xの形状について、下限を0、上限を
【数５】

$$lsize\_max-1=\lceil N/P\rceil-1=3 \qquad \cdots(5)$$

に置き換える。
【０１１２】
以上の変換を施した結果を、図１６－２に示す。出力コードは関数cyclic_fのソースコードを含む。別の実現方法として、関数cyclic_fは作り付けのライブラリとして事前に用意してシステムの一部とし、必要に応じて出力コードと結合するように構成してもよい。あるいは、インライン展開によって出力コード中に展開してもよい。また、この例では最大公約数を求める計算をコンパイル時に行うとしたが、関数cyclic_fと同様にソースプログラムとして生成してもよいし、インライン展開してもよいし、ライブラリとして出力コードと結合してもよい。
【０１１３】
次に、多次元に渡る分散に対するコード生成部１３０の処理について説明する。図１８－１は、コード生成部１３０が入力する内部表現のイメージを示す図である。ここでは、配列の下限値は0、配列からテンプレートへは恒等的な整列に正規化された状態から考える。また、前処理によりデータのアクセスはすべてローカルアクセスと保証されているとする。これらの正規化や前処理は、必要に応じて事前に行うことができる。
【０１１４】
入力プログラムは、図１８－２に示すようなデータ分散を表現している。すなわち、配列AとCはプロセッサ群Pに対してブロック分散されている。配列Bはその２次元目がPに対してブロック分散されている。配列Dは２次元構成のプロセッサ群P2に対して２次元ブロック分散されている。配列Rは分散されず、すべてのプロセッサにその複製が配置されている。プロセッサ群PとP2との対応はここでは同図に示すようにする。
【０１１５】
まず、テンプレートT1に対して以下の処理を行う。
［ステップＳ１００］
テンプレートT1に対するプロセッサ番号pを変数myID1に格納するため、手続の入口に以下のコードを挿入する。この値にはMPIライブラリが返す仮想プロセッサ番号をそのまま使用している。

```
CALL mpi_comm_rank(MPI_COMM_WORLD,RANK,...)
myID1=RANK
```

【０１１６】
なお、T1に対する他の分散パラメタはすべて定数であり、N=30、P=4、w=8である。また、自プロセッサの分割下限値を示す式p wすなわち8*myID1の評価回数を削減するため、以下のコードを生成する。

```
myLB1=8*myID1
```

【０１１７】
［ステップＳ２００］
T1に整列するループはL1のみである。
［ステップＳ２１０］I1 = 1, I2 = 28であるので、変換後の初期値i1と終値i2はブロック分散のgtol2関数よりそれぞれ図１８－３に示すように計算できる。この値を実行時に変数ib1,ie1を介してループの初期値と終値に設定できるように、ループL1の直前に実行されるコードを生成する。
［ステップＳ２２０］ループL1内のループ変数iの引用は図１８－１に示したプログラムの１４行に４箇所と１６行に２箇所ある。このうち、１４行の変数Bの第１添字式に出現するiと、１６行の右辺に出現するiは、対象から除外する条件に該当しないので、ブロック分散のltog関数に沿って(i+myLB1)に置換する。
【０１１８】
［ステップＳ３００］
T1に整列する変数はA,B,Cである。そこで、Aの宣言形状に対して、T1に整列している次元の宣言の下限を0、上限をw-1すなわち定数7に変換する。Bの２次元目の宣言形状と、Cの宣言形状に対しても、同様に変換する。
【０１１９】
図１８－４に、ここまでの変換後のイメージを示す。テンプレートT1および変数A,B,Cの整列に関する情報はここで不要になる。
【０１２０】
また、テンプレートT2に対して以下の処理を行う。
［ステップＳ１００］
ユーザが定義する多次元プロセッサ群とプロセッサとの対応付けは、入力プログラムの言語仕様や、処理系の実装方法や、実行時ライブラリの変換仕様による。ここでは図１８－５に示すような対応付けを行うとする。この場合、p1、p2は以下の式で得られる。
【数６】

$$\begin{aligned} p1 &= \mathrm{mod}(rank,2) \\ p2 &= \lfloor rank/2 \rfloor \end{aligned} \qquad \cdots(6)$$

【０１２１】
そして、次元d=1について、プロセッサ番号p1を変数myID2に設定するコードを以下のように生成する。
myID2=MOD(RANK,2)
なお、他の分散パラメタは、N1=30、P1=2、w=5となる。p1 * w = 5*myID2を変数myLB2に設定する。
【０１２２】
また、次元d=2について、プロセッサ番号p2を変数myID3に設定するコードを以下のように生成する。
myID3=RANK/2
なお、他の分散パラメタは、N2=30、P2=2、w=5となる。p2 * w = 5*myID3を変数myLB3に設定する。
【０１２３】
［ステップＳ２００］
まず、T2の1次元目に対して以下の処理を行う。T2の第1次元に整列するループはL2のみである。
［ステップＳ２１０］I1 = 0, I2 = jであるので、gtol2関数を用いてL2の下限ib2および上限ie2は図１８－６に示すように計算できる。myID2は0以上なので、ib2の値は常に定数0となる。ここでは最適化によりそのような簡単化ができたとする。
【０１２４】
［ステップＳ２２０］ループL2内のループ変数iの引用は図１８－１に示したプログラ

ムの２６行に２箇所ある。このうち、非分散配列Rの添字に出現するiは、対象から除外する条件に該当しないので、ltog関数を用いてi+myLB2に置換する。

【０１２５】

また、T2の2次元目に対して以下の処理を行う。T2の第2次元に整列するループはL3のみである。

［ステップＳ２１０］I1 = 1, I2 = 9, I3 = 2であるので、gtol3関数でブロック分散でI3>0のケースを用いてL3の下限ib3、上限ie3、および増分is3は図１８－７に示すように計算できる。この値を実行時に変数ib3,ie3を介してループの初期値と終値に設定できるように、ループL3の直前に実行されるコードを生成する。is3は定数なので直接DO文の増分に設定する。

【０１２６】

［ステップＳ２２０］ループL3内のループ変数jの引用は図１８－１に示したプログラム２６行に２箇所ある。このうち、非分散配列Rの添字に出現するjは、対象から除外する条件に該当しないので、ltog関数を用いてi+myLB3に置換する。

【０１２７】

［ステップＳ３００］

T2に整列している変数はDだけである。まず、Dの1次元目の下限を0、上限をw-1すなわち定数4に変換する。また、Dの2次元目の下限を0、上限をw-1すなわち定数4に変換する。

【０１２８】

以上の手順により、最終的に出力されるコードを図１８－８に示す。

【０１２９】

次に、本実施例に係るトランスレータプログラムを実行するコンピュータシステムについて説明する。図１９は、本実施例に係るトランスレータプログラムを実行するコンピュータシステムを示す図である。

【０１３０】

同図に示すように、このコンピュータシステム２００は、本体部２０１と、本体部２０１からの指示により表示画面２０２ａに情報を表示するディスプレイ２０２と、このコンピュータシステム２００に種々の情報を入力するためのキーボード２０３と、ディスプレイ２０２の表示画面２０２ａ上の任意の位置を指定するマウス２０４と、ＬＡＮ２０６または広域エリアネットワーク（ＷＡＮ）に接続するＬＡＮインタフェースと、公衆回線２０７に接続するモデムとを有する。ここで、ＬＡＮ２０６は、他のコンピュータシステム（ＰＣ）２１１、サーバ２１２、プリンタ２１３などとコンピュータシステム２００とを接続している。

【０１３１】

また、図２０は、図１９に示した本体部２０１の構成を示す機能ブロック図である。同図に示すように、この本体部２０１は、ＣＰＵ２２１と、ＲＡＭ２２２と、ＲＯＭ２２３と、ハードディスクドライブ（ＨＤＤ）２２４と、ＣＤ－ＲＯＭドライブ２２５と、ＦＤドライブ２２６と、Ｉ／Ｏインタフェース２２７と、ＬＡＮインタフェース２２８と、モデム２２９とを有する。

【０１３２】

そして、このコンピュータシステム２００において実行されるトランスレータプログラムは、フロッピィディスク（ＦＤ）２０８、ＣＤ－ＲＯＭ２０９、ＤＶＤディスク、光磁気ディスク、ＩＣカードなどの可搬型記憶媒体に記憶され、これらの記憶媒体から読み出されてコンピュータシステム２００にインストールされる。

【０１３３】

あるいは、このトランスレータプログラムは、ＬＡＮインタフェース２２８を介して接続されたサーバ２１２のデータベース、他のコンピュータシステム（ＰＣ）２１１のデータベースなどに記憶され、これらのデータベースから読み出されてコンピュータシステム２００にインストールされる。

【０１３４】

そして、インストールされたトランスレータプログラムは、ＨＤＤ２２４に記憶され、ＲＡＭ２２２、ＲＯＭ２２３などを利用してＣＰＵ２２１により実行される。

【０１３５】

上述してきたように、本実施例では、分散パラメタ設定部１３１が分散に関するプロセッサ毎のパラメタを設定するコードを生成し、ループインデックスローカル化部１３２がループのインデックスのローカル化を行い、配列形状ローカル化部１３３が分散配列のローカル化を行うこととしたので、添字式でインデックス変換を避けることができ、インデックス変換に起因するオーバーヘッドを削減することができる。

【０１３６】

また、本実施例に係るトランスレータプログラム１００によって生成されるプログラムは、ループ上下限の式が簡単になり、プロセッサ番号に依存しない式や、定数にできることも多い。それにより、上下限値がコンパイル時に確定するため、一般的な最適化が促進される。また、並列ループの外側に逐次ループがある場合でも、ループ交換、ループ合体（loop coalescing）などの最適化が行いやすい。さらに、ループのコストがコンパイル時に見積もりやすい。

【０１３７】

また、本実施例に係るトランスレータプログラム１００によって生成されるプログラムは、並列ループ入口でのコストが非常に小さい。したがって、内側のループで並列化しても性能低下が小さいので、アプリケーションプログラムの自由度が大きくなる。従来は、並列ループは入口でコストが高いので、ループの並列化は最外ループで行うのがよいとされていた。

【０１３８】

また、本実施例に係るトランスレータプログラム１００によって生成されるプログラムは、ループ／配列のインデックスのローカル化により、扱うインデックスの絶対値を小さく抑えられる。したがって、32ビット整数ではインデックスが表現できないような巨大配列や巨大ループでも、並列化によって32ビットでも扱えるようになる。

【０１３９】

また、本実施例に係るトランスレータプログラム１００によって生成されるプログラムでは、配列割付けは連続領域に圧縮できる。したがって、メモリ領域を節約することができる。また、アクセス範囲が局所化されるため、キャッシュ効率が向上する。

【０１４０】

また、本実施例に係るトランスレータプログラム１００によって生成されるプログラムでは、ループはマスクなし一重にできる。すなわち、ブロック－サイクリック分散では従来２重化していたが、ループに増分がなければ効率よく一重化できる。また、増分ありでもマスク付きで一重化できる。不規則分散は、従来マスク付きアクセスでしか実装できなかったが、ループに増分がなければ効率よく一重化できる。

【０１４１】

また、本実施例に係るトランスレータプログラム１００によって生成されるプログラムでは、分散配列の手続間受渡しが自然であり、実引数と仮引数の次元数／形状不整合にも自然な対応が可能である。

【０１４２】

また、本実施例では、FORTRANを基本とする並列プログラミング言語の場合について説明したが、本発明はこれに限定されるものではなく、例えばCなどの他のプログラミング言語を基本とする並列プログラミング言語の場合にも同様に適用することができる。

【０１４３】

（付記１）並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するトランスレータプログラムであって、

分散に関するパラメタである分散パラメタに値を設定するコードを生成する分散パラメタ設定手順と、

前記分散パラメタ設定手順により生成されたコードにより値が設定される分散パラメタ

を用いてループインデックスおよび配列インデックスをローカル化するコードを生成するインデックスローカル化手順と、
をコンピュータに実行させることを特徴とするトランスレータプログラム。
【０１４４】
（付記２）前記インデックスローカル化手順は、ループパラメタをローカル化するループパラメタローカル化手順と、ループ変数をグローバルインデックスに変換するループ変数変換手順をコンピュータに実行させることによってループインデックスをローカル化するコードを生成することを特徴とする付記１に記載のトランスレータプログラム。
【０１４５】
（付記３）前記並列言語プログラムには、分散種別として不均等ブロック分散および不規則分散を指定可能であることを特徴とする付記１または２に記載のトランスレータプログラム。
【０１４６】
（付記４）前記並列言語プログラムには、分散種別としてブロック分散、サイクリック分散およびブロック－サイクリック不均をさらに指定可能であることを特徴とする付記３に記載のトランスレータプログラム。
【０１４７】
（付記５）前記並列言語プログラムは、FORTRANを基本とする並列プログラミング言語を用いて作成されたプログラムであることを特徴とする付記１または２に記載のトランスレータプログラム。
【０１４８】
（付記６）前記分散パラメタ設定手順は、各手続きの入り口に分散パラメタに値を設定するコードを生成することを特徴とする付記１または２に記載のトランスレータプログラム。
【０１４９】
（付記７）前記ループパラメタローカル化手順は、ループの初期値および終値をローカルインデックスに変換することを特徴とする付記２に記載のトランスレータプログラム。
【０１５０】
（付記８）並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するトランスレータプログラムを記録したコンピュータ読み取り可能な記録媒体であって、
分散に関するパラメタである分散パラメタに値を設定するコードを生成する分散パラメタ設定手順と、
前記分散パラメタ設定手順により生成されたコードにより値が設定される分散パラメタを用いてループインデックスおよび配列インデックスをローカル化するコードを生成するインデックスローカル化手順と、
をコンピュータに実行させるトランスレータプログラムを記録したことを特徴とするコンピュータ読み取り可能な記録媒体。
【０１５１】
（付記９）並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するプログラム変換方法であって、
分散に関するパラメタである分散パラメタに値を設定するコードを生成する分散パラメタ設定工程と、
前記分散パラメタ設定工程により生成されたコードにより値が設定される分散パラメタを用いてループインデックスおよび配列インデックスをローカル化するコードを生成するインデックスローカル化工程と、
を含んだことを特徴とするプログラム変換方法。
【０１５２】
（付記１０）並列言語プログラムを解析し、複数のプロセッサで分散処理される逐次言語プログラムに変換するトランスレータ装置であって、

分散に関するパラメタである分散パラメタに値を設定するコードを生成する分散パラメタ設定手段と、
前記分散パラメタ設定手段により生成されたコードにより値が設定される分散パラメタを用いてループインデックスおよび配列インデックスをローカル化するコードを生成するインデックスローカル化手段と、
を備えたことを特徴とするトランスレータ装置。
【産業上の利用可能性】
【０１５３】
以上のように、本発明に係るトランスレータプログラム、プログラム変換方法およびトランスレータ装置は、並列言語プログラムのコンパイラに有用であり、特に、並列言語プログラムが分散メモリ型計算機により分散処理される場合に適している。
【図面の簡単な説明】
【０１５４】
【図１－１】本実施例に係るプログラム変換による並列化の例（１）を示す図である。
【図１－２】本実施例に係るプログラム変換による並列化の例（２）を示す図である。
【図１－３】本実施例に係るプログラム変換による並列化の例（３）を示す図である。
【図２】本実施例に係るトランスレータプログラムの構成を示す機能構成図である。
【図３】グローバルインデックスからローカルインデックスを求める漸化式を示す図である。
【図４－１】ループパラメタローカル化部が生成するコードを示す図である。
【図４－２】ループパラメタに増分が存在しない場合に、ループパラメタローカル化部が生成するコードを示す図である。
【図４－３】不規則分散の場合に、ループパラメタローカル化部が生成するコードを示す図である。
【図５】増分を消し去るループ変換を示す図である。
【図６】コード生成部の処理手順を示す流れ図である。
【図７】分散Dの属性を示す図である。
【図８】gtolの定義を示す図である。
【図９】ltogの定義を示す図である。
【図１０－１】gtol2の定義を示す図（１）である。
【図１０－２】gtol2の定義を示す図（２）である。
【図１１－１】gtol3の定義を示す図（１）である。
【図１１－２】gtol3の定義を示す図（２）である。
【図１１－３】gtol3の定義を示す図（３）である。
【図１２】Lsize_supの定義を示す図である。
【図１３－１】図２３－３に示した入力例１に対してコード生成部が入力する内部表現のイメージを示す図である。
【図１３－２】入力例１の変換後のプログラムを示す図である。
【図１３－３】汎用最適化後のプログラムを示す図である。
【図１４－１】図２５－１に示した入力例２に対してコード生成部が入力する内部表現のイメージを示す図である。
【図１４－２】初期値i1と終値i2を計算する式を示す図である。
【図１４－３】入力例２の変換後のプログラムを示す図である。
【図１５－１】図２６－１に示した入力例３に対してコード生成部が入力する内部表現のイメージを示す図である。
【図１５－２】入力例３の変換後のプログラムを示す図である。
【図１６－１】入力例４に対してコード生成部が入力する内部表現のイメージを示す図である。
【図１６－２】入力例４の変換後のプログラムを示す図である。
【図１７】サイクリック分散の計算式のnを求める処理手順を示すフローチャートである

。
【図１８－１】入力例５に対してコード生成部が入力する内部表現のイメージを示す図である。
【図１８－２】指定されたデータ分散を示す図である。
【図１８－３】ループパラメタの値を示す図である。
【図１８－４】変換の途中結果を示す図である。
【図１８－５】多次元プロセッサ群とプロセッサとの対応付けを示す図である。
【図１８－６】ループパラメタの値を示す図である。
【図１８－７】ループパラメタの値を示す図である。
【図１８－８】入力例５の変換後のプログラムを示す図である。
【図１９】本実施例に係るトランスレータプログラムを実行するコンピュータシステムを示す図である。
【図２０】図１９に示した本体部の構成を示す機能ブロック図である。
【図２１－１】データの分散を示す図である。
【図２１－２】ループの分散を示す図である。
【図２２】インデックス分割による分散の種類を示す図である。
【図２３－１】逐次プログラムを示す図である。
【図２３－２】OpenMPによる並列化を示す図である。
【図２３－３】HPFによる並列化（入力例１）を示す図である。
【図２４－１】データおよびループのインデックス分割とプロセッサとの対応を示す図である。
【図２４－２】図２３－３に示した並列言語プログラムに対して従来のコンパイラが出力するコードを示す図である。
【図２５－１】ブロック－サイクリック分散を用いる並列言語プログラムの例を示す図である。
【図２５－２】グローバルインデックスとローカルインデックスの対応を示す図である。
【図２５－３】図２５－１に示した並列言語プログラムに対して生成される従来のコードを示す図である。
【図２６－１】不規則分散を用いる並列言語プログラムの例を示す図である。
【図２６－２】グローバルインデックスとローカルインデックスの対応を示す図である。
【図２６－３】図２６－１に示した並列言語プログラムに対して生成される従来のコードを示す図である。
【図２７】数学記号の定義を示す図である。
【符号の説明】
【０１５５】
１０　　FORTRANコンパイラ
１００　　トランスレータプログラム
１１０　　並列言語パーサ
１２０　　最適化・正規化処理部
１３０　　コード生成部
１３１　　分散パラメタ設定部
１３２　　ループインデックスローカル化部
１３２ａ　　ループパラメタローカル化部
１３２ｂ　　ループ変数変換部
１３３　　配列形状ローカル化部
２００，２１１　　コンピュータシステム
２０１　　本体部
２０２　　ディスプレイ
２０２ａ　　表示画面
２０３　　キーボード

２０４　マウス
２０６　ＬＡＮ
２０７　公衆回線
２０８　フロッピィディスク
２０９　ＣＤ－ＲＯＭ
２１２　サーバ
２１３　プリンタ
２２１　ＣＰＵ
２２２　ＲＡＭ
２２３　ＲＯＭ
２２４　ハードディスクドライブ
２２５　ＣＤ－ＲＯＭドライブ
２２６　フロッピィディスクドライブ
２２７　Ｉ／Ｏインタフェース
２２８　ＬＡＮインタフェース
２２９　モデム

【図１－１】

本実施例に係るプログラム変換による並列化の例(1)を示す図

```
      real a(0:249,1:1000)

      do j=2,1000
        do i=0,249
          a(i,j)=a(i,j)+a(i,j-1)
        end do
      end do
      end
```

【図１－２】

本実施例に係るプログラム変換による並列化の例(3)を示す図

```
      real a(0:11,0:999)
      integer,parameter ::
     &  MAP(12)=(/2,1,3,4,3,2,4,4,1,2,2,4/)
      integer mapArray(0:11), gtol(0:12), ltog(0:11)
!--- 初期化
      do I=0,11
        mapArray(I)=MAP(I-1)-1
      end do
      gtol(0) = 0
      i = 0
      do k=0,11
        if (myID.EQ.mapArray(k)) then
          ltog(i) = k
          i = i+1
        end if
        gtol(k+1) = i
      end do

!--- 実行
      do j=1,1000
        do i=gtol(M),gtol(N+1)-1
          a(i,j) = a(i,j)+ltog(i)*a(i,j-1)
        end do
      end do
      end
```

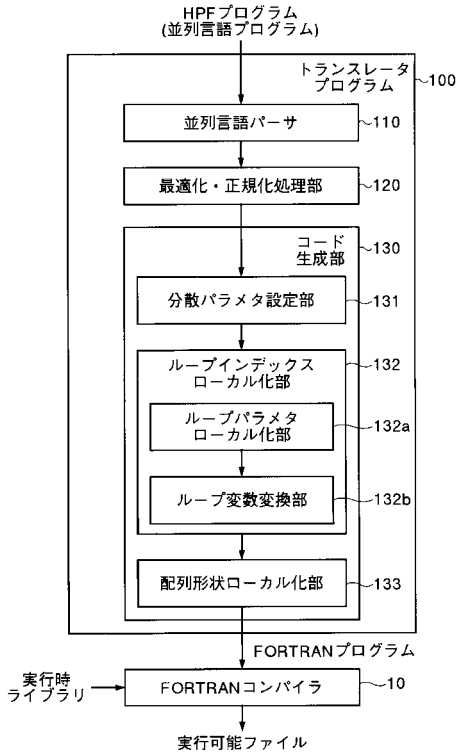
【図１－３】

本実施例に係るプログラム変換による並列化の例(3)を示す図

```
      real a(0:11,0:999)
      integer,parameter ::
     &  MAP(12)=(/2,1,3,4,3,2,4,4,1,2,2,4/)
      integer mapArray(0:11), gtol(0:12), ltog(0:11)
!--- 初期化
      do I=0,11
        mapArray(I)=MAP(I-1)-1
      end do
      gtol(0) = 0
      i = 0
      do k=0,11
        if (myID.EQ.mapArray(k)) then
          ltog(i) = k
          i = i+1
        end if
        gtol(k+1) = i
      end do

!--- 実行
      do j=1,1000
        do i=gtol(M),gtol(N+1)-1
          a(i,j) = a(i,j)+ltog(i)*a(i,j-1)
        end do
      end do
      end
```

【図２】

本実施例に係るトランスレータプログラムの構成を示す機能構成図



【図３】

グローバルインデックスから
ローカルインデックスを求める漸化式を示す図

$$gtol(0) = 0$$

$$gtol(k) = \begin{cases} gtol(k-1)+1 & \text{for } 1 \leqq k \leqq N,\quad M(k-1)=p \\ gtol(k-1) & \text{for } 1 \leqq k \leqq N,\quad M(k-1)\neq p \end{cases}$$

【図４－１】

ループパラメタローカル化部が生成するコードを示す図

変換前	変換後
DO I=I1,I2,I3 ループ本体 ENDDO	i1,i2,i3を求める計算コード DO I=i1,i2,i3 ループ本体 ENDDO

【図４－２】

ループパラメタに増分が存在しない場合に、
ループパラメータローカル化部が生成するコードを示す図

変換前	変換後
DO I=I1,I2 ループ本体 ENDDO	i1,i2を求める計算コード DO I=i1,i2 ループ本体 ENDDO

【図４－３】

不規則分散の場合に、
ループパラメタローカル化部が生成するコードを示す図

変換前	変換後
DO I=I1,I2 ループ本体 ENDDO	DO I=gtol(I1),gtol(I2+1)-1 ループ本体 ENDDO

【図５】

増分を消し去るループ変換を示す図

• 変換前

```
DO I=I1,I2,I3
    ... ! loop body
  ENDDO
```

• 変換後（I3>0のとき）

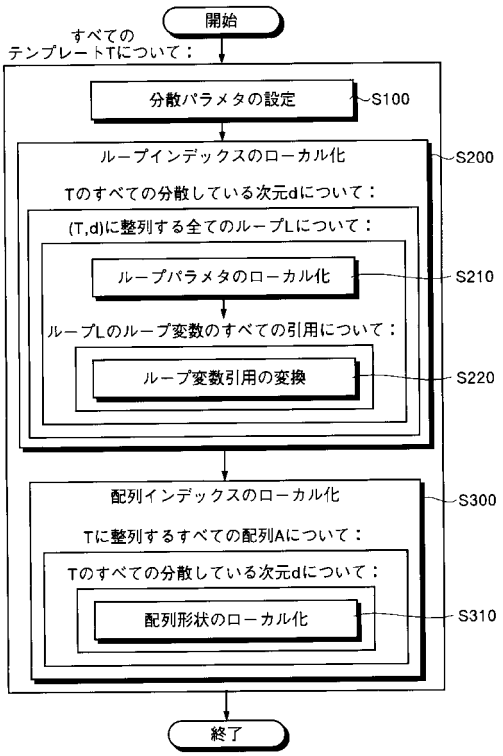
```
DO I=I1,I2
    IF (MOD(I-I1,I3).EQ.0) THEN
      ... ! loop body
    ENDIF
  ENDDO
```

• 変換後（I3<0のとき）

```
DO I=I2,I1
    IF (MOD(I-I1,I3).EQ.0) THEN
      ... ! loop body
    ENDIF
  ENDDO
```

【図６】

コード生成部の処理手順を示す流れ図



【図７】

分散Dの属性を示す図

文字	呼称	意味概略
d	分散種別	ブロック分散,サイクリック分散, 不均等ブロック分散, ブロック-サイクリック分数または不規則分散
P	プロセッサ数	プロセッサ群の着目する次元の寸法
N	配列の寸法	配列の着目する次元の寸法
w	ブロック幅	ブロック分散の幅。 無指定の場合にはw=⌈N/P⌉
W	ブロック幅配列	不均等ブロック分散で、 各プロセッサのブロック幅
B	分散下限値	B(p)=sum{W(p)|0≦q＜p}
M	マッピング配列	不規則分散で、 各インデックスに対応するプロセッサ番号

【図８】

gtolの定義を示す図

分散種別d	i = gtol(D,I)
ブロック分散	I - p w または mod(I, w)
サイクリック分散	⌊I / P⌋
不均等ブロック分散	I - B(p)
ブロック-サイクリック分散	w ⌊I / (P w)⌋ + mod(I, w)

【図９】

ltogの定義を示す図

分散種別d	I = ltog(D,i,p)
ブロック分散	i + p w
サイクリック分散	P i + p
不均等ブロック分散	i + B(p)
ブロック-サイクリック分散	P w ⌊i / w⌋ + p w + mod(i, w)

【図１０－１】

gtol2の定義を示す図(1)

ブロック分散

$$i1 = \begin{cases} I1 - p\,w & \text{for } p \leqq \lfloor I1 / w \rfloor \\ 0 & \text{for } p > \lfloor I1 / w \rfloor \end{cases}$$

$$i2 = \begin{cases} w - 1 & \text{for } p < \lfloor I2 / w \rfloor \\ I2 - p\,w & \text{for } p \geqq \lfloor I2 / w \rfloor \end{cases}$$

ブロック分散の他の実現方法

$$i1 = \begin{cases} w & \text{for } p < \lfloor I1 / w \rfloor \\ I1 - p\,w & \text{for } p = \lfloor I1 / w \rfloor \\ 0 & \text{for } p > \lfloor I1 / w \rfloor \end{cases}$$

$$i2 = \begin{cases} w - 1 & \text{for } p < \lfloor I2 / w \rfloor \\ I2 - p\,w & \text{for } p = \lfloor I2 / w \rfloor \\ -1 & \text{for } p > \lfloor I2 / w \rfloor \end{cases}$$

ブロック分散の他の実現方法
以下のようにループを実行しないという判断を
加えることもできる。
$p < \lfloor I1 / w \rfloor$ または $p > \lfloor I2 / w \rfloor$なら、ループをスキップする。

ブロック分散の他の実現方法
条件式
$p < \lfloor I1 / w \rfloor$　$p = \lfloor I1 / w \rfloor$　$p > \lfloor I1 / w \rfloor$
の代りに、それぞれ
$p\,w < I1$　$p\,w = I1$　$p\,w > I1$
を用いてもよい。不等号≧、≦についても同様である。
I2についても同様である。

【図１０－２】

gtol2の定義を示す図(2)

不均等ブロック分散

$$i1 = \begin{cases} I1 - B(p) & \text{for } B(p) \leqq I1 \\ 0 & \text{for } B(p) > I1 \end{cases}$$

$$i2 = \begin{cases} W(p) - 1 & \text{for } B(p+1) \leqq I2 \\ I2 - B(p) & \text{for } B(p+1) > I2 \end{cases}$$

不均等ブロック分散の他の実現方法

$$i1 = \begin{cases} W(p) & \text{for } B(p) < I1 \\ I1 - B(p) & \text{for } B(p) = I1 \\ 0 & \text{for } B(p) > I1 \end{cases}$$

$$i2 = \begin{cases} W(p) - 1 & \text{for } B(p) < I2 \\ I2 - B(p) & \text{for } B(p) = I2 \\ -1 & \text{for } B(p) > I2 \end{cases}$$

不均等ブロック分散の他の実現方法

以下のようにループを実行しないという判断を
加えることもできる。

B(p) ＜ I1またはB(p) ＞ I2なら、ループをスキップする。

サイクリック分散

$i1 = \lceil (I1 - p) / P \rceil$

$i2 = \lfloor (I2 - p) / P \rfloor$

ブロック-サイクリック分散

$$i1 = \begin{cases} w \left\lfloor \dfrac{I1}{P\,w} \right\rfloor + w & \text{for } p < \mathrm{mod}(\lfloor I1 / w \rfloor, P) \\ w \left\lfloor \dfrac{I1}{P\,w} \right\rfloor + \mathrm{mod}(I1, w) & \text{for } p = \mathrm{mod}(\lfloor I1 / w \rfloor, P) \\ w \left\lfloor \dfrac{I1}{P\,w} \right\rfloor & \text{for } p > \mathrm{mod}(\lfloor I1 / w \rfloor, P) \end{cases}$$

$$i2 = \begin{cases} w \left\lfloor \dfrac{I2}{P\,w} \right\rfloor + w - 1 & \text{for } p < \mathrm{mod}(\lfloor I2 / w \rfloor, P) \\ w \left\lfloor \dfrac{I2}{P\,w} \right\rfloor + \mathrm{mod}(I2, w) & \text{for } p = \mathrm{mod}(\lfloor I2 / w \rfloor, P) \\ w \left\lfloor \dfrac{I2}{P\,w} \right\rfloor - 1 & \text{for } p > \mathrm{mod}(\lfloor I2 / w \rfloor, P) \end{cases}$$

【図１１－１】

gtol3の定義を示す図(1)

ブロック分散

I3 ＞0のとき

$$i1 = \begin{cases} I1 - p\,w & \text{for } p \leqq \lfloor I1 / w \rfloor \\ \mathrm{mod}(I1 - p\,w, I3) & \text{for } p > \lfloor I1 / w \rfloor \end{cases}$$

$$i2 = \begin{cases} w - 1 & \text{for } p < \lfloor I2 / w \rfloor \\ I2 - p\,w & \text{for } p \geqq \lfloor I2 / w \rfloor \end{cases}$$

$i3 = I3$

I3 ＜0のとき

$$i1 = \begin{cases} w - 1 - \mathrm{mod}(p\,w + w - 1 - I1, -I3) & \text{for } p < \lfloor I1 / w \rfloor \\ I1 - p\,w & \text{for } p \geqq \lfloor I1 / w \rfloor \end{cases}$$

$$i2 = \begin{cases} I2 - p\,w & \text{for } p \leqq \lfloor I2 / w \rfloor \\ 0 & \text{for } p > \lfloor I2 / w \rfloor \end{cases}$$

$i3 = I3$

ブロック分散の他の実現方法

I3 ＞0のとき

$$i1 = \begin{cases} w & \text{for } p < \lfloor I1 / w \rfloor \\ I1 - p\,w & \text{for } p = \lfloor I1 / w \rfloor \\ \mathrm{mod}(I1 - p\,w, I3) & \text{for } p > \lfloor I1 / w \rfloor \end{cases}$$

$$i2 = \begin{cases} w - 1 & \text{for } p < \lfloor I2 / w \rfloor \\ I2 - p\,w & \text{for } p = \lfloor I2 / w \rfloor \\ -1 & \text{for } p > \lfloor I2 / w \rfloor \end{cases}$$

$i3 = I3$

I3 ＜0のとき

$$i1 = \begin{cases} p\,w + w - 1 - \mathrm{mod}(p\,w + w - 1 - I1, -I3) & \text{for } p < \lfloor I1 / w \rfloor \\ I1 - p\,w & \text{for } p = \lfloor I1 / w \rfloor \\ -1 & \text{for } p > \lfloor I1 / w \rfloor \end{cases}$$

$$i2 = \begin{cases} w & \text{for } p < \lfloor I2 / w \rfloor \\ I2 - p\,w & \text{for } p = \lfloor I2 / w \rfloor \\ 0 & \text{for } p < \lfloor I2 / w \rfloor \end{cases}$$

$i3 = I3$

ブロック分散の他の実現方法

以下のようにループを実行しないという判断を加えることもできる。

I3 ＞0のとき

$p < \lfloor I1 / w \rfloor$ または $p > \lfloor I2 / w \rfloor$なら、ループをスキップする。

I3 ＜0のとき

$p > \lfloor I1 / w \rfloor$ または $p < \lfloor I2 / w \rfloor$なら、ループをスキップする。

ブロック分散の他の実現方法

条件式

$p < \lfloor I1 / w \rfloor$　$p = \lfloor I1 / w \rfloor$　$p > \lfloor I1 / w \rfloor$

の代りに、それぞれ

$p\,w < I1$　$p\,w = I1$　$p\,w > I1$

を用いてもよい。不等号≧、≦についても同様である。I2についても同様である。

【図１１－２】

gtol3の定義を示す図(2)

不均等ブロック分散

I3 ＞0のとき

$$i1 = \begin{cases} I1 - B(p) & \text{for } B(p) \leqq I1 \\ \mathrm{mod}(I1 - B(p), I3) & \text{for } B(p) > I1 \end{cases}$$

$$i2 = \begin{cases} W(p) - 1 & \text{for } B(p+1) \leqq I2 \\ I2 - B(p) & \text{for } B(p+1) > I2 \end{cases}$$

$i3 = I3$

I3 ＜0のとき

$$i1 = \begin{cases} W(p) - 1 - \mathrm{mod}(B(p+1) - 1 - I1, -I3) & \text{for } B(p+1) \leqq I1 \\ I1 - B(p) & \text{for } B(p+1) > I1 \end{cases}$$

$$i2 = \begin{cases} I2 - B(p) & \text{for } B(p) \leqq I2 \\ 0 & \text{for } B(p) > I2 \end{cases}$$

$i3 = I3$

L(p)はmyLBに置換えてもよい。

不均等ブロック分散の他の実現方法

I3 ＞0のとき

$$i1 = \begin{cases} W(p) & \text{for } B(p) < I1 \\ I1 - B(p) & \text{for } B(p) = I1 \\ \mathrm{mod}(I1 - B(p), I3) & \text{for } B(p) > I1 \end{cases}$$

$$i2 = \begin{cases} W(p) - 1 & \text{for } B(p) < I2 \\ I2 - B(p) & \text{for } B(p) = I2 \\ -1 & \text{for } B(p) > I2 \end{cases}$$

$i3 = I3$

I3 ＜0のとき

$$i1 = \begin{cases} B(p+1) - 1 - \mathrm{mod}(B(p+1) - 1 - I1, -I3) & \text{for } B(p) < I1 \\ I1 - B(p) & \text{for } B(p) = I1 \\ -1 & \text{for } B(p) > I1 \end{cases}$$

$$i2 = \begin{cases} W(p) & \text{for } B(p) < I2 \\ I2 - B(p) & \text{for } B(p) = I2 \\ 0 & \text{for } B(p) > I2 \end{cases}$$

$i3 = I3$

不均等ブロック分散の他の実現方法

以下のようにループを実行しないという判断を加えることもできる。

I3 ＞0のとき

B(p) ＜ I1またはB(p) ＞ I2なら、ループをスキップする。

I3 ＜0のとき

B(p) ＞ I1またはB(p) ＜ I2なら、ループをスキップする。

【図１１－３】

gtol3の定義を示す図(3)

サイクリック分散

I3 ＞0のとき

t = GCD(P, |I3|)と定義し、nを以下の条件を満たす整数とする。

$I1 + n\,I3 \equiv p \pmod{P}$

$0 \leqq n < P / t$

このnを求めるアルゴリズムを図17に示す。

GCDはユークリッドの互除法としてよく知られたアルゴリズムで求めることができる。

$$i1 = \begin{cases} (I1 + n\,I3 - p) / P & \text{for } p \equiv I1 \pmod{t} \text{（nが存在するとき）} \\ \text{正の最大数} & \text{otherwise（nが存在しないとき）} \end{cases}$$

$i2 = \lfloor (I2 - p) / P \rfloor$

$i3 = I3 / t$

nが存在しないとき、i1の値はi2より大きな値であれば任意でよい。

I3 ＜0のとき

同様にnを求める。

$$i1 = \begin{cases} (I1 + n\,I3 - p) / P & \text{for } p \equiv I1 \pmod{t} \text{（nが存在するとき）} \\ -1 & \text{otherwise（nが存在しないとき）} \end{cases}$$

$i2 = \lceil (I2 - p) / P \rceil$

$i3 = I3 / t$

nが存在しないとき、i1の値はi2より小さな値であれば任意でよい。

サイクリック分散の他の実現方法

以下のようにループを実行しないという判断を加えることもできる。

I3 ＞0のとき

nが存在しないなら、ループをスキップする。

I3 ＜0のとき

nが存在しないなら、ループをスキップする。

サイクリック分散の他の実現方法

nが存在する／しないは、$p \equiv I1 \pmod{t}$か否か、すなわち、

$\mathrm{mod}(I1 - p, t) = 0$か否かで判定してもよい。

【図１２】

Lsize_supの定義を示す図

分散種別d	lsize sup(D)
ブロック分散	w
サイクリック分散	⌈N / P⌉
不均等ブロック分散	max{W(q) : 0≦q<P}
ブロック-サイクリック分散	w ⌈N / (P w)⌉

【図１３－１】

図23-3に示した入力例1に対して
コード生成部が入力する内部表現のイメージを示す図

```
      real A(0:999,1000)
!hpf$ processors P(0:3)
!hpf$ distribute A(block,*) onto P

      do J=2,1000
!hpf$   independent, on home(A(I,:))
        do I=0,999
          A(I,J)=A(I,J)+A(I,J-1)
        end do
      end do
      end
```

【図１３－２】

入力例1の変換後のプログラムを示す図

```
      real a(0:249,1:1000)
      CALL mpi_comm_rank(MPI_COMM_WORLD,myID,...)

      do j=2,1000
        if (myID.le.0) then
          i1=-250*myID
        else
          i1=0
        endif
        if (myID.lt.3) then
          i2=249
        else
          i2=999-250*myID
        endif
        do i=i1,i2
          a(i,j)=a(i,j)+a(i,j-1)
        end do
      end do
      end
```

【図１３－３】

汎用最適化実施後のプログラムを示す図

```
      real a(0:249,1:1000)
      CALL mpi_comm_rank(MPI_COMM_WORLD,myID,...)

      do j=2,1000
        do i=0,249
          a(i,j)=a(i,j)+a(i,j-1)
        end do
      end do
      end
```

【図１４－１】

図25-1に示した入力例2に対して
コード生成部が入力する内部表現のイメージを示す図

```
      real A(0:999,1000)
!hpf$ processors P(4)
!hpf$ distribute A(cyclic(5),*) onto P

      do J=2,1000
!hpf$   independent, on home(A(I,:))
        do I=1,998
          A(I,J)=A(I,J)+A(I,J-1)
        end do
      end do
      end
```

【図１４－２】

初期値i1と終値i2を計算する式を示す図

$$i1 = \begin{cases} 5*\lfloor 1/(4*5)\rfloor+5 & = 5 \quad \text{for } p<0 \\ 5*\lfloor 1/(4*5)\rfloor+\mathrm{mod}(1,5) & = 1 \quad \text{for } p=0 \\ 5*\lfloor 1/(4*5)\rfloor & = 0 \quad \text{for } p>0 \end{cases}$$

$$i2 = \begin{cases} 5*\lfloor 998/(4*5)\rfloor+5-1 & = 249 \quad \text{for } p<3 \\ 5*\lfloor 998/(4*5)\rfloor+\mathrm{mod}(998,5) & = 248 \quad \text{for } p=3 \\ 5*\lfloor 998/(4*5)\rfloor-1 & = 244 \quad \text{for } p>3 \end{cases}$$

【図１４－３】

入力例2の変換後のプログラムを示す図

```
      real a(0:249,1:1000)
      CALL mpi_comm_rank(MPI_COMM_WORLD,myID,...)

      do j=2,1000
        if (myID.eq.0) then
          i1=1
        else
          i1=0
        endif
        if (myID.lt.3) then
          i2=249
        else
          i2=248
        endif
        do i=i1,i2
          a(i,j)=a(i,j)+a(i,j-1)
        end do
      end do
      end
```

【図１５－１】

図26-1に示した入力例3に対して
コード生成部が入力する内部表現イメージを示す図

```
      real A(0:11,0:999)
      integer,parameter ::
     &  MAP(12)=(/2,1,3,4,3,2,4,4,1,2,2,4/)
!hpf$ processors P(4)
!hpf$ distribute A(indirect(MAP),*) onto P

      do J=1,1000
!hpf$   independent, on home(A(I,:))
        do I=M,N
          A(I,J)=A(I,J)+A(I,J-1)*I
        end do
      end do
      end
```

【図１５－２】

入力例3の変換後のプログラムを示す図

```
      real a(0:11,0:999)
      integer,parameter ::
     &  MAP(12)=(/2,1,3,4,3,2,4,4,1,2,2,4/)
      integer mapArray(0:11), gtol(0:12), ltog(0:11)
!--- 初期化
      CALL mpi_comm_rank(MPI_COMM_WORLD,myID,...)
      do I=0,11
        mapArray(I)=MAP(I-1)-1
      end do
      gtol(0) = 0
      i = 0
      do k=0,11
        if (myID.EQ.mapArray(k)) then
          ltog(i) = k
          i = i+1
        end if
        gtol(k+1) = i
      end do

!--- 実行
      do j=1,1000
        do i=gtol(M),gtol(N+1)-1
          a(i,j) = a(i,j)+ltog(i)*a(i,j-1)
        end do
      end do
      end
      end
```

【図１６－１】

入力例4に対してコード生成部が入力する内部表現のイメージを示す図

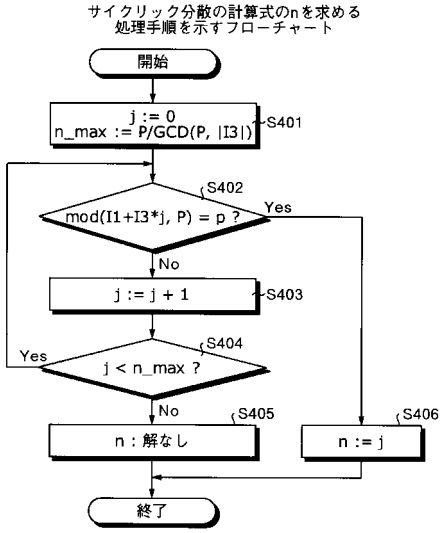
```
      real X(100,100,0:23)
!hpf$ processors Q(0:5)
!hpf$ distribute X(*,*,cyclic) onto Q

!hpf$ independent, on home(X(:,:,k))
      do k=4,20,4
        do j=1,100
          do i=1,100
            X(i,j,k)=i*j*k
          end do
        end do
      end do
      end
```

【図１６－２】

入力例4の変換後のプログラムを示す図

```
      REAL x(1:100,1:100,0:3) ―――――――――――――― ｽﾃｯﾌﾟS300
      INTEGER i,j,k
      INTEGER myID ―――――――――――――――――――― ｽﾃｯﾌﾟS100
      INTEGER k1,k2,k3 ―――――――――――――――――― ｽﾃｯﾌﾟS210
      INTEGER n_cyc ――――――――――――――――――― ｽﾃｯﾌﾟS210
      INTEGER,EXTERNAL:: cyclic_f ―――――――――――― ｽﾃｯﾌﾟS210
      CALL mpi_comm_rank(MPI_COMM_WORLD,myID,...) ―― ｽﾃｯﾌﾟS100
      n_cyc = cyclic_f(4,4,6,myID,2) ―――――――――― ｽﾃｯﾌﾟS210
      IF (n_cyc.NE.-1) THEN ――――――――――――――― ｽﾃｯﾌﾟS210
        k1 = (4+n_cyc*4-myID)/6
        k2 = (26-myID)/6-1          } ｽﾃｯﾌﾟS210
        k3 = 2
        DO k=k1,k2,k3
          DO j=1,100,1
            DO i=1,100,1
              x(i,j,k) = i*j*(6*k+myID) ―――――――― ｽﾃｯﾌﾟS220
            ENDDO
          ENDDO
        ENDDO
      ENDIF
      END

      INTEGER FUNCTION cyclic_f(i1,i3,P,myID,t)
      INTEGER i1,i3,P,myID,t
      INTEGER j,n,n_max
      j = 0
      n = -1
      n_max = P/t
 2000 CONTINUE
      IF (mod(i1+i3*j,P).EQ.myID) THEN
        n = j                       } ｽﾃｯﾌﾟS210
      ELSE
        j = j+1
        IF (j.LT.n_max) GOTO 2000
      ENDIF
      cyclic_f = n
      END FUNCTION
```

【図１７】



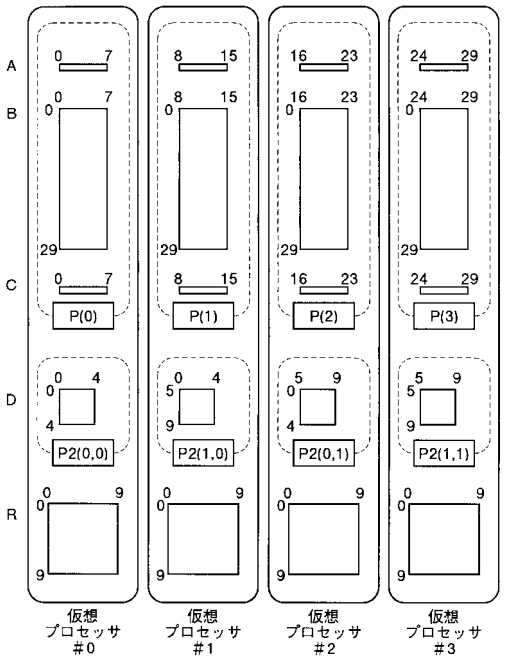
【図１８－１】

入力例5に対してコード生成部が入力する内部表現のイメージを示す図

```
 1 !hpf$ processors P(0:3),P2(0:1,0:1)
 2       real A(0:29),B(0:29,0:29),C(0:29),D(0:9,0:9),R(0:9,0:9)
 3 !hpf$ template T1(0:29),T2(0:29,0:29)
 4 !hpf$ distribute T1(block(8)) onto P
 5 !hpf$ distribute T2(block(5),block(5)) onto P2
 6 !hpf$ align A(:)   with T1(:)
 7 !hpf$ align B(*,:) with T1(:)
 8 !hpf$ align C(:)   with T1(:)
 9 !hpf$ align D(:,:) with T2(:,:)
10
11 !hpf$ independent, new(j)
12       L1: do i=1,28
13 !hpf$   on home(T1(i))
14         A(i)=B(i,i)+C(i)
15         do j=1,10
16           B(j,i)=0.1*i*j
17         end do
18       end do
19
20 !hpf$ independent, new(i)
21       L3: do j=1,9,2
22 !hpf$   on home(T2(:,j))
23 !hpf$   independent
24         L2: do i=0,j
25 !hpf$     on home(T2(i,j))
26           D(i,j)=R(i,j)
27         end do
28       end do
29
30       end
```

【図１８－２】

指定されたデータ分散を示す図



【図１８－３】

ループパラメタの値を示す図

$$i1 = \begin{cases} 1-myLB1 & \text{for } myID1 \leqq 0 \\ 0 & \text{for } myID1 > 0 \end{cases}$$

$$i2 = \begin{cases} 7 & \text{for } myID1 < 3 \\ 28-myLB1 & \text{for } myID1 \geqq 3 \end{cases}$$

【図１８－４】

変換の途中結果を示す図

```
 1 !hpf$ processors P2(0:1,0:1)
 2       real A(0:7),B(0:29,0:7),C(0:7),D(0:9,0:9),R(0:9,0:9)
 3 !hpf$ template T2(0:29,0:29)
 5 !hpf$ distribute T2(block(5),block(5)) onto P2
 9 !hpf$ align D(:,:) with T2(:,:)
         integer myID1,myLB1,ib1,ie1
10
         CALL mpi_comm_rank(MPI_COMM_WORLD,RANK,ierr)
         myID1=RANK
         myLB1=8*myID1

           if (myID1.le.0) then
           ib1=1-myLB1
         else
           ib1=0
         endif
         if (myID1.lt.3) then
           ie1=7
         else
           ie1=28-myLB1
         endif
12       L1: do i=ib1,ie1
14         A(i)=B(i+myLB1,i)+C(i)
15         do j=1,10
16           B(j,i)=0.1*(i+myLB1)*j
17         end do
18       end do
19
20 !hpf$ independent, new(i)
21       L3: do j=1,9,2
22 !hpf$   on home(T2(:,j))
23 !hpf$   independent
24         L2: do i=0,j
25 !hpf$     on home(T2(i,j))
26           D(i,j)=R(i,j)
27         end do
28       end do
29
30       end
```

ステップ'S300
ステップ'S100, ステップ'S210
ステップ' S210
ステップ'S210
ステップ'S220

【図１８－５】

多次元プロセッサ群とプロセッサとの対応付けを示す図

ユーザ定義プロセッサ	プロセッサ番号p1	プロセッサ番号p2	MPIのrank
P2(0,0)	0	0	0
P2(1,0)	1	0	1
P2(0,1)	0	1	2
P2(1,1)	1	1	3

【図１８－６】

ループパラメタの値を示す図

$$ib2 = \begin{cases} 0-myLB2 & \text{for } myID2 \leqq 0 \\ 0 & \text{for } myID2 > 0 \end{cases}$$

$$ie2 = \begin{cases} 4 & \text{for } myID2 < 1 \\ j-myLB2 & \text{for } myID2 \geqq 1 \end{cases}$$

【図１８－７】

ループパラメタの値を示す図

$$ib3 = \begin{cases} 1-myLB3 & \text{for } myID3 \leqq 0 \\ modulo(1-myLB3,2) & \text{for } myID3 > 0 \end{cases}$$

$$ie3 = \begin{cases} 4 & \text{for } myID3 < 1 \\ 9-myLB3 & \text{for } myID3 \geqq 1 \end{cases}$$

$$is3 = 2$$

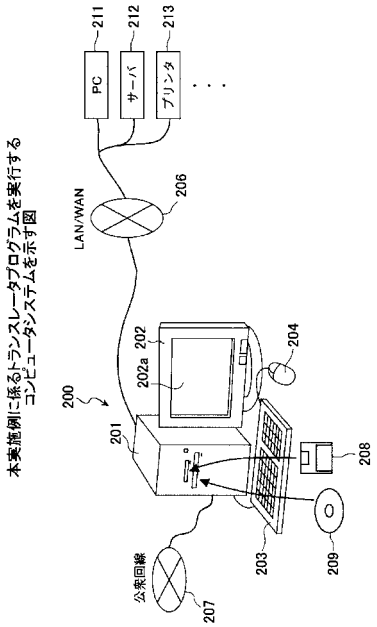
【図１８－８】

入力例5の変換後のプログラムを示す図

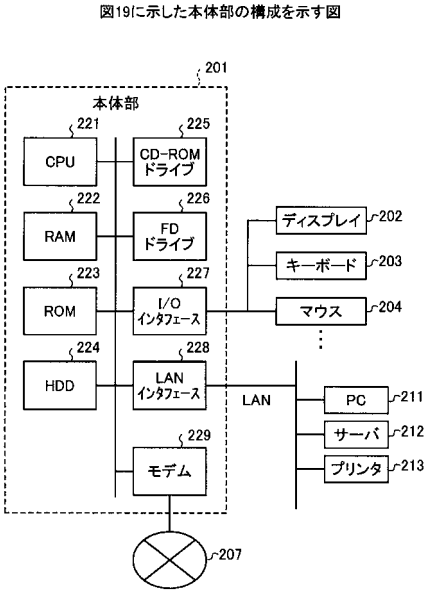
```
 2       real A(0:7),B(0:29,0:7),C(0:7),D(0:4,0:4),R(0:9,0:9)
         integer myID1,myLB1,ib1,ie1
         integer myID2,myLB2, myID3,myLB3
         integer ie2,ib3,ie3
10
         CALL mpi_comm_rank(MPI_COMM_WORLD,RANK,ierr)
         myID1=RANK
         myLB1=8*myID1
         myID2=MOD(RANK,2)
         myID3=RANK/2
         myLB2=5*myID2
         myLB3=5*myID3

         if (myID1.le.0) then ; ib1=1-myLB1
         else                 ; ib1=0
         endif
         if (myID1.lt.3) then ; ie1=7
         else                 ; ie1=28-myLB1
         endif
12       L1: do i=ib1,ie1
14         A(i)=B(i+myLB1,i)+C(i)
15         do j=1,10
16           B(j,i)=0.1*(i+myLB1)*j
17         end do
18       end do
19
         if (myID3.le.0) then
           ib3=1-myLB3
         else
           ib3=MODULO(1-myLB3,2)
         endif
         if (myID3.lt.1) then
           ie3=4
         else
           ie3=9-myLB3
         endif
21       L3: do j=ib3,ie3,2
           if (myID2.lt.1) then
             ie2=4
           else
             ie2=j-myLB2
           endif
24         L2: do i=0,ie2
26           D(i,j)=R(i+myLB2,j+myLB3)
27         end do
28       end do
29
30       end
```

ステップ'S300
ステップ'S100
ステップ'S210
ステップ'S100
L3に対するステップ'S210
L2に対するステップ'S210
L2に対するステップ'S220
L3に対するステップ'S220

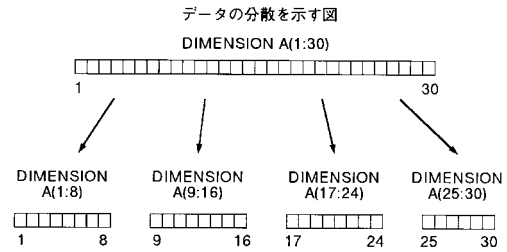
【図１９】



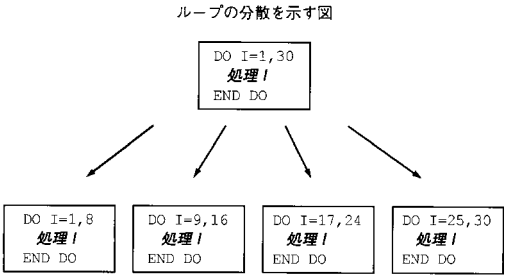
【図２０】



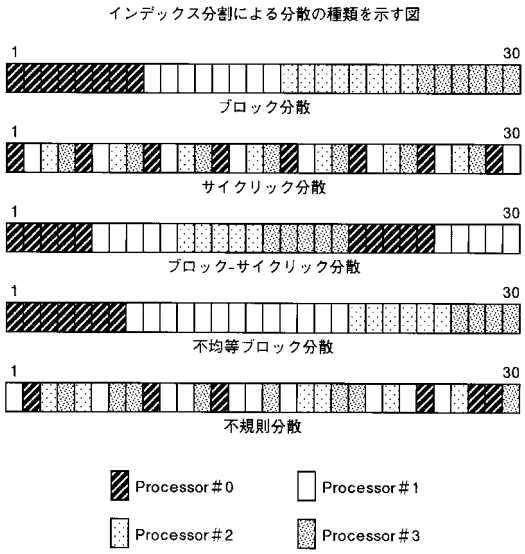
【図２１－１】



【図２１－２】



【図２２】



【図２３－１】

逐次プログラムを示す図

```
      real A(1000,1000)

      do J=2,1000
        do I=1,1000
          A(I,J)=A(I,J)+A(I,J-1)
        end do
      end do
      end
```

【図２３－２】

OpenMPによる並列化を示す図

```
      real A(1000,1000)

      do J=2,1000
!$omp   parallel do
        do I=1,1000
          A(I,J)=A(I,J)+A(I,J-1)
        end do
      end do
      end
```

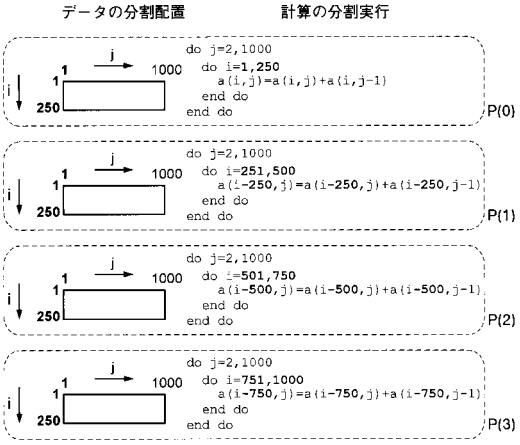
【図２３－３】

HPFによる並列化(入力例1)を示す図

```
      real A(1000,1000)
!hpf$ processors P(0:3)
!hpf$ distribute A(block,*) onto P

      do J=2,1000
!hpf$   independent
        do I=1,1000
          A(I,J)=A(I,J)+A(I,J-1)
        end do
      end do
      end
```

【図２４－１】

データおよびループのインデックス分割とプロセッサとの対応を示す図



【図２４－２】

図23-3に示した並列言語プログラムに対して
従来のコンパイラが出力するコードを示す図

```
real a(1:250,1:1000)   !分割後のサイズで各プロセッサに割付け

myLB=250*myID+1        !プロセッサ毎の担当範囲の下限値
myUB=myLB+249          !同、上限値

do j=1,1000
  do I=myLB,myUB
    a(I-myLB+1,j)=a(I-myLB+1,j)+a(I-myLB+1,j-1)
  end do
end do
end
```

【図２５－１】

ブロック-サイクリック分散を用いる
並列言語プログラムの例(入力例2)を示す図

```
      real A(1:1000,1:1000)
!hpf$ processors P(4)
!hpf$ distribute A(cyclic(5),*) onto P   !幅5のブロック-サイクリック
      do J=1,1000
!hpf$   independent
        do I=2,999                       !配列の形状と一致しない
          A(I,J)=A(I,J)+A(I,J-1)
        end do
      end do
      end
```

【図２５－２】

グローバルインデックスと
ローカルインデックスの対応を示す図

プロセッサ	p	ローカルインデックス																
		1	2	3	4	5	6	7	8	9	10	11	…		246	247	248	249	250
P(0)	0	1	2	3	4	5	21	22	23	24	25	41	…		981	982	983	984	985
P(1)	1	6	7	8	9	10	26	27	28	29	30	46	…		986	987	988	989	990
P(2)	2	11	12	13	14	15	31	32	33	34	35	51	…		991	992	993	994	995
P(3)	3	16	17	18	19	20	36	37	38	39	40	56	…		996	997	998	999	1000

【図２５－３】

図25-1に示した並列言語プログラムに対して
生成された従来のコードを示す図

```
  real a(1:250,1:1000)          !分割後のサイズで圧縮して割付け

  myLB1=5*myID+1                !プロセッサ毎の最初のブロックの下限値
  myUB1=myLB1+980               !プロセッサ毎の最後のブロックの下限値
  myST1=20                      !ブロック間の距離
  myLB2=0
  IF(myID.eq.0) myLB2=1         !ループ初期値のずれを考慮
  myUB2=4
  IF(myID.eq.3) myUB2=3         !ループ終値のずれを考慮

  do j=1,1000
     do I1=myLB1,myUB1,myST1
       do I=I1+myLB2,I1+myUB2
         a((I-1)/20*5+MODULO(I-1,5)+1,j)=
&            a((I-1)/20*5+MODULO(I-1,5)+1,j)+
&            a((I-1)/20*5+MODULO(I-1,5)+1,j-1)
       end do
     end do
  end do
  end
```

【図２６－１】

不規則分散を用いる並列言語プログラムの例(入力例3)を示す図

```
      real A(0:11,0:999)
      integer,parameter ::
     &  MAP(12)=(/2,1,3,4,3,2,4,4,1,2,2,4/)
!hpf$ processors P(4)
!hpf$ distribute A(indirect(MAP),*) onto P   !対応表MAPで
                                             !表される分散
      do J=1,1000
!hpf$   independent
        do I=M,N
          A(I,J)=A(I,J)+I*A(I,J-1)
        end do
      end do
      end
```

【図２６－２】

グローバルインデックスとローカルインデックスの対応を示す図

プロセッサ	p	ローカルインデックス			
		1	2	3	4
P(0)	0	2	9		
P(1)	1	1	6	10	11
P(2)	2	3	5		
P(3)	3	4	7	8	12

【図２６－３】

図26-1に示した並列言語プログラムに対して
生成される従来のコードを示す図

```
      real A(0:11,0:999)
      integer,parameter ::
     &  MAP(1:12) =(/2,1,3,4,3,2,4,4,1,2,2,4/)
      integer GTOL(1:12),icount(4)
!--- 初期化
      do id=1,4
        icount(id)=0
      end do
      do I=1,12
        icount(MAP(I)) = icount(MAP(I))+1
        GTOL(I) = icount(MAP(I))
      end do

!--- 実行
      do j=1,1000
        do I=M,N
          if (myID.eq.MAP(I)) then
            A(GTOL(I),j)=A(GTOL(I),j)+I*A(GTOL(I),j-1)
          endif
        end do
      end do
      end
```

【図２７】

数学記号の定義を示す図

⌈ x ⌉	xより小さくない最小の整数（ceiling関数）
⌊ x ⌋	xより大きくない最大の整数（floor関数）
mod(x, y)	剰余関数。mod(x, y) = x - ⌊ x / y ⌋ y 例えば mod(8,5) = 3、mod(-8,5) = 2
GCD(x, y)	xとyの最小公倍数
｜ x ｜	xの絶対値

フロントページの続き

(56)参考文献 特開平１１－２８２８１４（ＪＰ，Ａ）
特開平０８－２５５０８６（ＪＰ，Ａ）
特開平０８－２２１２７６（ＪＰ，Ａ）
浅岡香枝、平野彰雄、岡部寿男、金澤正憲，ＮＰＢを用いたＨＰＦ／ＪＡ拡張のＶＰＰ上での評価，情報処理学会研究報告，日本，社団法人情報処理学会，２００２年 3月 8日，Ｖｏｌ．２００２，Ｎｏ．２２，ｐｐ．１３３～１３８
左近彰一、田村正典、妹尾義樹、村井 均，ＳＸ－５シリーズのＨＰＦコンパイラ，ＮＥＣ技報，日本，日本電気株式会社，１９９９年１１月２５日，Ｖｏｌ．５２，Ｎｏ．１１，ｐｐ．７１～７４
岩下英俊、進藤達也、岡田 信，ＶＰＰ Ｆｏｒｔｒａｎ：分散メモリ型並列計算機言語，情報処理学会論文誌，日本，社団法人情報処理学会，１９９５年 7月１５日，Ｖｏｌ．３６，Ｎｏ．７，ｐｐ．１５４２～１５５０

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ 9／４５