Fortran マクロデータフロー処理におけるデータローカライゼーション手法

吉田明正† 前田誠司†
尾形航† 笠原博徳†

本論文では，ループ，サブルーチン，基本ブロック等の粗粒度タスクを，実行時にプロセッサにスケジューリングし並列処理を行うマクロデータフロー処理における，データローカライゼーション手法を提案する．マクロデータフロー処理のようにタスクのダイナミックスケジューリングを行う方式では，粗粒度タスク間で共有される変数を集中型共有メモリに配置し，粗粒度タスク間のデータ授受も集中型共有メモリを介して行われるのが一般的であった．しかし，このような方式では共有メモリを介したデータ転送オーバヘッドが大きくなるという問題点がある．この問題点を考慮し，本論文では，実行時に同一プロセッサに割り当てられる粗粒度タスク間でのデータ授受を，ローカルメモリを介して行うことにより，共有メモリを介したデータ転送オーバヘッドを軽減するデータローカライゼーション手法を提案する．本手法では，このデータローカライゼーションを複数ループに渡って行うために新たに提案するループ整合分割手法を用いる．ここで，ループ整合分割手法とは，ループ間に存在する配列データ依存を局所化するように，各ループを整合して分割する手法である．提案手法を用いたコンパイラは，マルチプロセッサシステム OSCAR 上でインプリメントされており，複数アプリケーションプログラムによる OSCAR 上での性能評価の結果から，処理時間が 15%～20% 程度短縮されることが確認された．

A Data-Localization Scheme for Fortran Macro-Dataflow Computation

Akimasa Yoshida,† Seiji Maeda,† Wataru Ogata† and Hironori Kasahara†

This paper proposes a data-localization scheme for macro-dataflow computation which automatically exploits a parallelism among coarse-grain tasks such as loops, subroutines, and basic blocks in a Fortran program. In macro-dataflow computation, data shared by macrotasks are allocated to common memory and transferred among macrotasks via common memory. However, data transfer via common memory sometimes causes large overhead. Considering the problem, this paper proposes a data-localization scheme, by which data are transferred via local memory among macrotasks assigned to the same processor, to minimize data transfer overhead. The proposed scheme employs an aligned loop decomposition method to allow a compiler to localize data among macrotasks like loops. The scheme has been implemented on an actual multiprocessor system OSCAR. Performance evaluation on OSCAR shows that the proposed data-localization scheme can reduce the execution time by 15%-20%.

1. はじめに

従来のマルチプロセッサシステム用 Fortran 並列化コンパイラでは，Doall，Doacross 等のループ並列化（中粒度並列処理）[1),2),14)] のみを用いた自動並列化が行われていた．しかし，このループ並列化コンパイラでは，ループ以外の部分の並列性，例えば，サブルーチン，ループ，基本ブロック間の粗粒度並列性を利用することができないという問題であった．この問題点を解決するために最近では，ループやサブルーチン等の粗粒度並列性を自動的に抽出し，それらの粗粒度タスクを実行時にダイナミックスケジューリングし，並列処理を行うマクロデータフロー処理[10)~13)]が提案されている．

このマクロデータフロー処理のように，ダイナミックスケジューリングを用いて粗粒度タスク（マクロタスク）をプロセッサに実行時に割り当てる方式では，共有データを集中型共有メモリ上に置き，マクロタス

† 早稲田大学理工学部
School of Science and Engineering, Waseda University

ク間のデータ転送は集中型共有メモリを介して行われるのが一般的である．しかし，このような方式では集中型共有メモリを介したデータ転送オーバヘッドが大きくなってしまうという問題が生じる．そこで，マクロタスク間データ転送オーバヘッドを軽減し効率良い並列処理を実現するためには，データ分割・配置を適切に行い各プロセッサ上のローカルメモリを介したマクロタスク間データ授受を可能とすることが重要となる．

例えば，ローカルメモリの有効利用をはかり単一ループの処理の高速化を行う基礎的な研究として，Tu と Padua[3]，Li[4] が，Array Privatization 法とその自動化手法を提案している．この Array Privatization 法では，単一ループ内で使用される作業配列変数を各プロセッサ上のローカルメモリに割り当てることにより，ループの並列化，および集中型共有メモリアクセスによるオーバヘッド軽減を実現する手法である．しかし，この Array Privatization 法は，複数ループ間でのローカルメモリを介したデータ授受に適用できないという問題点がある．

また，分散メモリマシン上でのデータ分割・配置に関しては，High Performance Fortran[5] や Fortran D[6] 等の Fortran 拡張言語を用いて，ユーザがデータ分割・配置を指定する方法が研究されている．しかし，一般ユーザが並列性最大化とプロセッサ間通信最小化を満たすデータ分割・配置を決定することは非常に難しい．そこで，データ分割・配置を自動的に行うために，Li と Chen[7] は，単一ループ内での配列データの参照パターンから，適切な通信ライブラリを決定し，その通信時間を最小化するようにデータ分割・配置する方法を提案している．さらに，Gupta と Banerjee[8] は，プログラム中の各ループで望まれるデータ分割・配置に関する制約とその制約が実行時間に与える影響を求め，プログラム全体で実行時間を最小化するように制約を調整し，データ分割・配置を決定する方法を提案している．また，分散型共有メモリマシンにおいては，Anderson と Lam[9] が，Doall および Doacross ループの並列性を利用しつつ，プロセッサ間通信最小化を目指した，手順とデータの分割手法を提案している．しかし，これらのデータ分割・配置法は，いずれもユーザあるいはコンパイラが静的にデータ割り付けを行う場合にしか適用できないという制約がある．すなわち，マクロデータフロー処理のように実行時に手順とデータを動的に配置する方式には適用できない．

そこで，本論文では，マクロデータフロー処理において複数ループ間でのローカルメモリを介したデータ授受を可能とするために，各ループでの配列データの使用・定義範囲を考慮して手順とデータを分割するループ整合分割法を新たに開発し，さらに，そのループ整合分割後にマクロタスク間データ転送量の多いマクロタスク集合を同一のプロセッサに割り当て，マクロタスク間共有（配列）変数のローカルメモリを介した授受を実現するデータローカライゼーション手法を提案する．本手法を用いたコンパイラは，すでに，マルチプロセッサシステム OSCAR[15] 上にインプリメントされており，本論文では，OSCAR 上で行った性能評価についても述べる．

本論文2章では，マクロデータフロー処理の概要，3章ではマクロデータフロー処理におけるデータローカライゼーション手法について述べる．4章では性能評価に用いたマルチプロセッサシステム OSCAR のアーキテクチャと性能評価の結果について述べる．

2. マクロデータフロー処理

本章では，Fortran プログラムのマクロデータフロー処理[10)~13)]について概説する．

2.1 マクロタスク（MT）生成

本マクロデータフローコンパイレーション手法では，Fortran プログラムをマクロタスクと呼ぶ並列処理単位に分割する．この各マクロタスクは，2.4 で述べるダイナミックスケジューリングルーチンによって，実行時にプロセッサクラスタ（PC）に割り当てられて並列処理される．マクロデータフロー処理では，以下に示す3種類のマスクタスクを定義する[12)]．

- 擬似代入文ブロック
  (Block of Pseudo Assignment statements: BPA)
- 繰り返しブロック (Repetition Block: RB)
- サブルーチンブロック (Subroutine Block: SB)

BPA は基本的には通常の基本ブロックであるが，基本ブロックの処理時間は1回のダイナミックスケジューリングに要する時間に比べ短い場合が多いので，複数の基本ブロックを融合し BPA を生成する．RB は最外側ナチュラルループ[16)]である．また，サブルーチンに関しては，基本的に可能な限りインライン展開を適用するが，コードが長くなり過ぎ効果的にインライン展開できない場合には，そのサブルーチンをマク

ロタスク（SB）として定義する．

しかし，上述のようにRBを定義すると，Doallループが1台のPCに割り当てられるため，全PCを使ってDoallループを実行することができない．そこで，Doallループの場合には，そのループをPC台数あるいはその倍数に分割し，その分割された各部分ループをマクロタスク（RB）と定義することにより，全PC上でのDoall処理を可能としている[12]．

また，3.2で述べるマクロタスク融合法により融合されたマクロタスクは，1つのマクロタスクとして扱われ，1台のPCに割り当てられる[12]．

2.2 マクロフローグラフ（MFG）の生成

コンパイラは次に，BPA，RB，SBなどのマクロタスク（MT）間のコントロールフロー，データフローを解析する．

このMT間のデータフロー解析では，各MTで使用または定義される配列変数のデータ（添字）範囲を考慮して解析を行う．具体的には，まず，MT間のデータフローを解析する前に，MTごとに各配列変数の入力データ（MT内で使用されるデータの中でMT外部から到達するデータ）範囲情報と出力データ（MT内で定義されるデータの中でMT外部に到達するデータ）範囲情報を求める．次に，変数名によるデータフロー解析で，MT間に配列変数のデータ依存が存在する場合，予め求めておいた各MTにおける配列変数の入力または出力データ範囲情報を参照して，配列変数の使用範囲を考慮したデータフロー解析を行う．例えば，変数名によるデータフロー解析でMT_iとMT_j間に配列変数Xに関するフロー依存が存在する場合，MT_iにおける配列変数Xの出力データ範囲とMT_jにおける配列変数Xの入力データ範囲に重なり（共通データ）が存在するかを調べ，共通データが存在すればMT_iとMT_j間に配列変数Xのデータ依存があることになる．

解析されたマクロタスク間のコントロールフロー，データフローは，**図1**に示すようなマクロフローグラフ（MFG）[10),11)]で表現される．本マクロデータフロー処理では，MFG上でループを構成する後方へのエッジ（バックエッジ）はRB内部に含めるようにマクロタスクを生成するため，MFGは無サイクル有向グラフとなる．

2.3 マクロタスクグラフ（MTG）の生成

マクロフローグラフ（MFG）はマクロタスク間のコントロールフローとデータフローを陽に表したものであり，マクロタスク間の並列性を表現していない．マクロタスク間には，コントロール依存とデータ依存が存在するので，本コンパイレーション方法では，コントロール依存とデータ依存を考慮したマクロタスク間並列性を最大限に引き出すために，各マスクタスクの最早実行可能条件解析[10),11)]を用いる．マクロタスク$i(MT_i)$の最早実行可能条件とは，MT_iが最も早い時点で実行可能となるための条件であり，一般形は次のとおりである．

（MT_iがコントロール依存するMT_jがMT_iの方向に分岐）

AND

（（MT_iがデータ依存するすべてのMT_kの実行終了）

OR（MT_kが実行されないことが確定））．

しかし，この最早実行可能条件の定義では冗長な条件項が含まれてしまうため，OSCAR Fortranコンパイラでは，冗長な条件を排除した最早実行可能条件を自動的に求めている[10)]．GirkarとPolychronopoulosは，この最早実行可能条件の研究成果[10)]を利用し，多少変更したアルゴリズムを提案している[17)]．

また，各マクロタスクの最早実行可能条件は，**図2**に示すようなマクロタスクグラフ（MTG）[10),11)]と呼ぶ無サイクル有向グラフで表すことができる．

MTGにおいて各ノードはマクロタスクを表す．点線のエッジは拡張されたコントロール依存を示し，実線のエッジはデータ依存を表す．拡張されたコントロール依存は，通常のコントロール依存だけでなく，データ依存先行マクロタスクが実行されないための条件も表している．MTG中のノード内の小円を起点とするデータ依存エッジは，コントロール依存とデータ

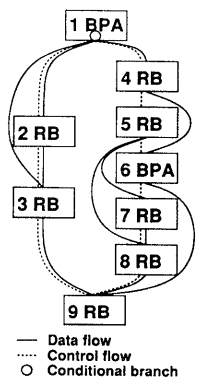


図1 マクロフローグラフ
Fig. 1 Macro-flow graph.

依存の2つを同時に表している．また，MTG 中のエッジを束ねている実線のアークは，そのアークによって束ねられたエッジが互いに AND の関係にあることを示し，点線のアークはそのアークで束ねられたエッジが互いに OR の関係にあることを示す．この MTG においてエッジの向きは下向きと仮定しており，ほとんどの矢印は省略されている．矢印がついているエッジは，もとの MFG 上での分岐方向を表すエッジである．

2.4 ダイナミックスケジューリングルーチンの生成

マクロデータフロー処理では，条件分岐やマクロタスクの実行時間の変動のような実行時不確定性の問題に対処するため，マクロタスクを実行時にプロセッサクラスタ（PC）に割り当てる方式をとる．

このダイナミックスジューリングは，粗粒度タスク（マクロタスク）に対して適用されるため，スケジューリングオーバヘッドは相対的に小さく抑えられる．さらに，本手法では，コンパイラにより生成されたスケジューリングルーチンを使用するため，従来のマルチプロセッサシステムで問題となっていた OS コールやライブラリコールに関連した大きなオーバヘッドを除去できる．

また，OSCAR Fortran コンパイラでは，ダイナミックスケジューリングアルゴリズムとして，Dynamic-CP 法[14]を用いている．

3. データローカライゼーション手法

本章では，集中型共有メモリを介したデータ転送オーバヘッドを軽減するためのデータローカライゼーション手法を提案する．データローカライゼーションとは，同一プロセッサに割り当てられたマクロタスク間では，共有メモリを介さず，ローカルメモリを介してデータ授受を行う手法である．この手法は，次の手順で実現される．

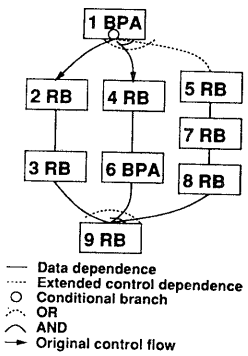


図 2 マクロタスクグラフ
Fig. 2 Macro-task graph.

まず，Doall ループの並列性を利用し，かつ複数ループに渡ってデータローカライゼーションを行うために，ループ整合分割を行う．このループ整合分割では，各ループでの配列データの使用・定義範囲を考慮し，ループ間に存在する配列データ依存を局所化するように，各ループを分割する．

次に，データ転送量の多いマクロタスク間でローカルメモリを介してデータ授受を行うためには，データ転送量の多いマクロタスク集合を同一のプロセッサクラスタ（PC）に割り当てなければならない．これをダイナミックスケジューリングで実現すると，実行時に行うべき作業が増え，ダイナミックスケジューリングのオーバヘッドが大きくなる可能性がある．そこで，本手法では，もし異なる PC に割り当てられたとすると多量のデータ転送を必要とするマクロタスク集合をコンパイル時に融合し，その融合により生成された融合マクロタスクを，実行時にダイナミックスケジューリングにより1台の PC に割り当てる方式をとる．

コンパイラは，マクロタスク融合後，融合 MT 内（融合 MT 内に存在する旧 MT 間）で，ローカルメモリを介したデータ授受を行うマシンコードを生成する．

以下の節では，各手順の詳細を示す．

3.1 ループ整合分割

マクロデータフロー処理では，Doall ループが1台のプロセッサクラスタ（PC）だけで実行されるのを防ぐため，前述のように Doall ループを PC 台数あるいはその倍数に分割し，それらを複数 PC 上で並列実行させる．しかし，他の Doall ループとの間にデータ依存のある Doall ループを各々独立に分割すると，ローカルメモリを介したデータ授受を行うためのマクロタスク融合を適用できなくなる可能性がある．

例えば，**図3**（a）の各 RB（ループ）を3台の PC 用に，イタレーション数を均等に分割（ループ均等分割）すると図3（b）のようになる．図3（b）（上図）で対角の黒い部分は，RB_1 と RB_2 の各インデクス I, J における配列変数Aの要素の定義または使用を表している．また，網掛けで示される四角形の部分は，分割により生成される各部分 RB のインデクス範囲（I, J の値の範囲）および配列変数Aの使用あるいは定義さ

れる要素の範囲を表している．図3（b）（下図）は，上図のMTGである．このMTGにおいて，データ依存エッジ（実線）の太さはMT間で転送されるデータ量を表している．

図3（b）のループ均等分割の場合，$RB_{1\alpha}$ と $RB_{1\beta}$ で定義されるデータのうち，点線により挟まれる範囲（$RB_{1\alpha}$ と $RB_{1\beta}$ の境界付近）のデータは，$RB_{2\alpha}$，$RB_{2\beta}$ の両方で使用される．このとき，データ転送量の多い $RB_{1\alpha}$ と $RB_{2\alpha}$ の組，および $RB_{1\beta}$ と $RB_{2\beta}$ の組等を融合して1PCに割り当てると，これらの融合MTはデータ依存（図3（b）のMTGでクロスした細い実線）を満たすため，融合MT内部で同期が必要となる．しかし，これらの同期は，ダイナミックスケジューリングの割り当て単位がマクロタスク（融合MTを含む）であるため，スケジューラからは認識できない．そこで，スケジューラから認識できない同期が融合MT間で独自に取られるとすると，融合MTのPCへの割り当てによっては，デッドロックを生じる可能性がある．従って，データ転送量の多い $RB_{1\alpha}$ と $RB_{2\alpha}$ の組，および $RB_{1\beta}$ と $RB_{2\beta}$ の組等を融合することができない．すなわち，$RB_{1\alpha}$ と $RB_{2\alpha}$ 間や，$RB_{1\beta}$ と $RB_{2\beta}$ 間等でデータローカライゼーションを行うことが困難となる．

それに対して，提案するループ整合分割法では，図3（a）の各RBを図3（c）のように分割する．この場合，RB_{2a}，RB_{2b} で共通に使用されるデータを定義する RB_1 のインデクス範囲（Iの値の範囲）は，独立した小ループ（図中では RB_{1b}，以後，共通データ定義ループと呼ぶ）に分割されているので，データ転送量の多い RB_{1a} と RB_{2a} の組，および RB_{1c} と RB_{2b} の組等を融合することができ，これらのRB間でデータローカライゼーションが可能となる．

以下に，このループ整合分割の手順を述べる．

3.1.1 ループ整合分割の適用対象とするRBグループの抽出

ループ整合分割を適用するRB集合をRBグループとする．このRBグループは，次の手順で生成される．

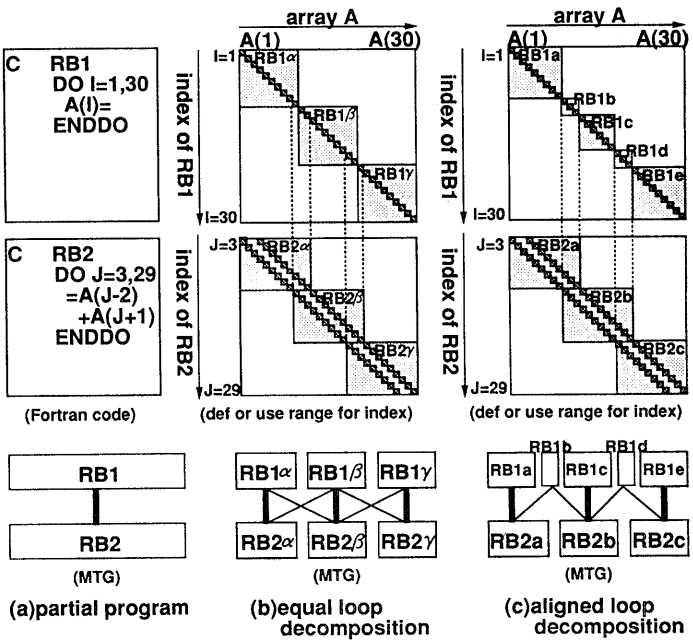


図3 ループ均等分割とループ整合分割
Fig. 3 Equal loop decomposition and aligned loop decomposition.

マクロタスクグラフ（MTG）上で，お互いが唯一の直接後続タスクおよび唯一の直接先行タスクであるような配列変数データ依存が存在するRBの中で，以下の条件を満たすRBグループを抽出する．ただし，MTG上ではトランシティブ☆なデータ依存エッジ[19]が除去されているため，ここで定義されるRBグループ内のRB間では，このトランシティブなデータ依存エッジを付加したMTGに対して解析を行う．また，このRBグループの決定には，RB内で使用される変数の初期化等を行う小規模BPAからのデータ依存エッジは無視してグループ化を行っている．

ここで，RBグループが持つべき条件を示す．

（1） RBグループ内の各RB（最外側ループ）は，Doallループまたはリダクションループ（総和計算ループ）である．
以下の説明においてRBの誘導変数は，RBの最外側ループの誘導変数を意味する．

（2） RBグループ内である配列変数 X に関するデータ依存が存在する RB_i と RB_j では，

☆ タスクグラフ上でタスク T_i からタスク T_j へのエッジと，タスク T_j からタスク T_k へのエッジがある時に，タスク T_i からタスク T_k へのエッジが存在する場合，これをトランシティブエッジという．

RB_i 中の配列変数 X の添字式で誘導変数の使用される次元と，RB_j 中の配列変数 X の添字式で誘導変数の使用される次元とが同じである．この次元を配列変数 X の整合分割次元とする．

なお，各 RB の誘導変数は，整合分割次元の添字式で使用されていれば，整合分割次元以外の次元の添字式で使用されていてもよい．

以後，整合分割次元の添字式を単に添字式と呼ぶ．

（3） RB グループ内の各 RB_i 中の各配列変数 X の添字式が，$p^i_X \times (RB_i$ の誘導変数$) + q^i_X (p^i_X \neq 0)$ で表される1次式である．

なお，p^i_X，q^i_X および以下の議論で使用する記号の意味は**表1**に示されるとおりである．

（4） RB グループ内でデータ依存がある RB_i と RB_j では，データ依存を導くすべての配列変数 X に対して，RB_i での添字式の誘導変数係数と RB_j での添字式の誘導変数係数との比 (p^i_X/p^j_X) が一定である．

上記の条件を満たすために前処理として，各 RB にループインターチェンジ[18]と，各 RB（最外側ループ）のストライドを1にするリストラクチャリングを必要に応じて適用する．

ここで**図4**の RB グループの例を用いて，上記条件を説明する．なお，この図4では，ループ整合分割の説明に不要な変数や定数等を省略している．図4において，RB_1，RB_2 の最外側ループが Doall ループであるとすると条件(1)を満たし，また，RB_1，RB_2 の誘導変数 I_1 と I_2 が，配列変数 A では1次元目，配列変数 B では2次元目という具合に，各配列変数において添字式の同一次元で使用されているので条件(2)も満たしている．次に，各配列変数の添字式は誘導変数の1次式で表されているので条件(3)も満足している．さらに，データ依存のある RB 間で誘導変数係数の比が等しい，すなわち $p^2_{Ause_1}/p^1_{Adef_1} = p^2_{Ause_2}/p^1_{Adef_1} = p^2_{Buse_1}/p^1_{Bdef_1}$ であれば条件(4)を満たすことになり，この RB_1，RB_2 が RB グループとして選ばれる．

以後，各 RB グループごとに，RB グループ内の各 RB（最外側ループ）に対してループ整合分割を行う．

3.1.2 RB グループ内 RB 間でのデータ依存解析

3.1.1 で選び出された RB グループ（m 個の RB で構成されているとする）内の複数 RB 間で，イタレーション間データ依存を解析する．この解析結果は，各 RB_i（RB グループ内で i 番目の RB，$1 \leq i \leq m$）の標準イタレーション変換係数 SIC_i と標準イタレーションデータ依存範囲 $[SIL_i : SIU_i]$（ただし，[下限値：上限値] は下限値から上限値までの範囲を意味する）を用いて表す．

ここで，RB_i $(1 \leq i \leq m-1)$ の SIC_i と $[SIL_i : SIU_i]$

```
C     RB1
      DO I1=LB1,UB1,1
        DO J1=…,…,…
          A(p^1_Adef1 * I1+q^1_Adef1,…)=
          B(…,p^1_Bdef1 * I1+q^1_Bdef1)=
        ENDDO
      ENDDO
C     RB2
      DO I2=LB2,UB2,1
        DO J2=…,…,…
          =A(p^2_Ause1 * I2+q^2_Ause1,…)
            +A(p^2_Ause2 * I2+q^2_Ause2,…)
            +B(…,p^2_Buse1 * I2+q^2_Buse1)
        ENDDO
      ENDDO
```

図4 RB グループの例
Fig. 4 An example of RB group.

表1 記号と意味
Table 1 Denotation.

記号	意味
$p^i_{Xuse_k}[p^i_{Xdef_k}]$	RB_i で配列変数 X を使用［定義］する k 番目の添字式の誘導変数係数
$q^i_{Xuse_k}[q^i_{Xdef_k}]$	RB_i で配列変数 X を使用［定義］する k 番目の添字式の定数項
SIC_i	RB_i の標準イタレーション変換係数 ただし，SIC_i の値は，分数（分母は自然数）として保持する
LCD_{SIC}	RB グループ内のすべての SIC_i の分母の最小公倍数
$SIL_i[SIU_i]$	RB_i の標準イタレーションデータ依存範囲の下限値［上限値］
$LB_i[UB_i]$	RB_i のインデクス範囲の下限値［上限値］
$SLB_i[SUB_i]$	RB_i のインデクス範囲を，標準ループのインデクス範囲に換算した範囲の下限値［上限値］
$GSLB[GSUB]$	グループ標準インデクス範囲の下限値［上限値］
$DGSLB_n[DGSUB_n]$	グループ標準インデクス範囲の分割後の n 番目の部分領域の下限値［上限値］
$DLB^i_n[DUB^i_n]$	RB_i のインデクス範囲の分割後の n 番目の部分領域の下限値［上限値］

は，標準ループ RB_m の S 番目のイタレーションがデータ依存している RB_i のイタレーション集合が，$(S\times SIC_i+SIL_i)$ から $(S\times SIC_i+SIU_i)$ のインデクス範囲内にあることを意味している．すなわち，$(S\times SIC_i+SIL_i)$ と $(S\times SIC_i+SIU_i)$ は，それぞれ，データ依存の及ぶ可能性のあるイタレーション集合のインデクスの下限値と上限値を表している．

以下に，SIC_i，$[SIL_i : SIU_i]$ の求め方を示す．ただし，説明に使用される記号の意味は表1のとおりである．

i）RB グループ内MTGの出口ノード RB_m を標準ループとし，$SIC_m=1$, $[SIL_m : SIU_m]=[0 : 0]$ とする．

ii）RB グループ内で，RB_{m-1} から RB_1 まで順に，標準ループ RB_m に対する各 RB_i $(1\leq i\leq m-1)$ の標準イタレーション変換係数 SIC_i を次の式で求める．

$$SIC_i=SIC_j\times\frac{p^i_{Xud_1}}{p^i_{Xud_1}}.$$

j: RB_i の直接後続データ依存ループ RB_j

X: RB_i と RB_j 間でデータ依存が存在する配列変数

ここで，上式の意味を簡単に説明する．SIC_i の値は，SIC_j に，RB_j と RB_i におけるデータ依存配列変数 X の添字式の誘導変数係数の比の値を掛けることにより求められる．この RB_j と RB_i における誘導変数係数の比の値は，3.1.1 の RB グループ条件(4)によりどの配列変数Xに対しても同じ値をとるため，どの配列変数Xから求めてもよい．

また，上式での配列変数 X の添字式の誘導変数係数 px_{ud} と定数項 qx_{ud}（手順 iii）で使用する）の添字 ud は，use あるいは def を意味する仮の添字とする．例えば，RB_i と RB_j 間に配列変数 X のフロー依存が存在する場合には，p^i_{Xud} と q^i_{Xud} の添字 ud は def を表し，一方，p^j_{Xud} と q^j_{Xud} の添字 ud は use を表すものとする．

なお，RB_i に複数の直接後続データ依存ループ RB_j が存在し，その各 RB_j から求めた RB_i の SIC_i が異なる場合には，対象としている RB グループ全体でループ整合分割を行うことができない．このような場合には，RB グループを RB_1 から RB_i と，RB_{i+1} から RB_m の2つの RB グループに分割し，その各々の RB グループに再びループ整合分割を適用する．

iii）RB グループ内で，RB_{m-1} から RB_1 まで順に，標準ループ RB_m に対する各 RB_i $(1\leq i\leq m-1)$ の標準イタレーションデータ依存範囲 $[SIL_i : SIU_i]$ を次の式で求める．

$$SIL_i=\begin{cases}\min_j\left(\min_D\left(\min_X\left\lfloor\left(\min_k\left(\frac{q^i_{Xud_k}}{p^i_{Xud_1}}\right)-\max_k\left(\frac{q^i_{Xud_k}}{p^i_{Xud_1}}\right)\right)\right.\right.\right. & \\ \left.\left.\left.+SIL_j\times\frac{p^j_{Xud_1}}{p^i_{Xud_1}}\right\rfloor\right)\right) & if(SIC_i>0)\\ \max_j\left(\max_D\left(\max_X\left\lceil\left(\max_k\left(\frac{q^i_{Xud_k}}{p^i_{Xud_1}}\right)-\min_k\left(\frac{q^i_{Xud_k}}{p^i_{Xud_1}}\right)\right)\right.\right.\right. & \\ \left.\left.\left.+SIL_j\times\frac{p^j_{Xud_1}}{p^i_{Xud_1}}\right\rceil\right)\right) & if(SIC_i<0).\end{cases}$$

$$SIU_i=\begin{cases}\max_j\left(\max_D\left(\max_X\left\lceil\left(\max_k\left(\frac{q^i_{Xud_k}}{p^i_{Xud_1}}\right)-\min_k\left(\frac{q^i_{Xud_k}}{p^i_{Xud_1}}\right)\right)\right.\right.\right. & \\ \left.\left.\left.+SIU_j\times\frac{p^j_{Xud_1}}{p^i_{Xud_1}}\right\rceil\right)\right) & if(SIC_i>0)\\ \min_j\left(\min_D\left(\min_X\left\lfloor\left(\min_k\left(\frac{q^i_{Xud_k}}{p^i_{Xud_1}}\right)-\max_k\left(\frac{q^i_{Xud_k}}{p^i_{Xud_1}}\right)\right)\right.\right.\right. & \\ \left.\left.\left.+SIU_j\times\frac{p^j_{Xud_1}}{p^i_{Xud_1}}\right\rfloor\right)\right) & if(SIC_i<0).\end{cases}$$

j: RB_i のすべての直接後続データ依存ループ RB_j

D: RB_i と RB_j 間に存在する各データ依存，すなわち，$D\in$ {フロー依存，逆依存，出力依存}

X: RB_i と RB_j 間で各データ依存 D が存在する配列変数

k: 各ループ中で配列変数 X の使用 $(ud=use)$ [定義 $(ud=def)$] に関する k 番目の添字式

ここで，上式の意味を簡単に説明する．例えば，$SIC_i>0$ かつ RB_i と RB_j 間で配列変数 X のフロー依存だけが存在する場合，SIL_i の導出式における $\min_k(q^i_{Xud_k}/p^i_{Xud_1})-\max_k(q^i_{Xud_k}/p^i_{Xud_1})$ の項は，RB_j の $S\times SIC_j$ 番目のイタレーションが，RB_i にデータ依存する可能性のあるイタレーション集合のインデクス下限値を，$S\times SIC_i$ に対する相対値として表した値である．また，$SIL_j\times(p^j_{Xud_1}/p^i_{Xud_1})$ の項は，SIL_j を RB_i のインデクスに換算した値を表してい

る．

なお，$SIC_i<0$ の場合には，標準ループ RB_m のインデクス K の増加に伴い，イタレーション K がデータ依存している RB_i のインデクスが減少するため，$SIL_i \geq SIU_i$ となる．

上述の複数ループ間のイタレーション間データ依存解析を，**図5**（a）のRBグループ例に適用すると，各 RB_i の SIC_i，$[SIL_i : SIU_i]$ は，図5（a）（右側）に示した値になり，これらを図示すると図5（b）のようになる．この図5（b）は，RB_3 の $K=S$ のイタレーションがデータ依存している各 RB_i $(1\leq i\leq 2)$ のインデクス範囲を網掛けで表している．また，$K=50$ としたときのインデクスを｛ ｝に記す．この場合，図5（b）では，RB_3 の $K=50$ のイタレーションが，RB_2 の $J=102$ のイタレーションと，RB_1 の $I=99$ から103の範囲内のイタレーションにデータ依存していることを表している．

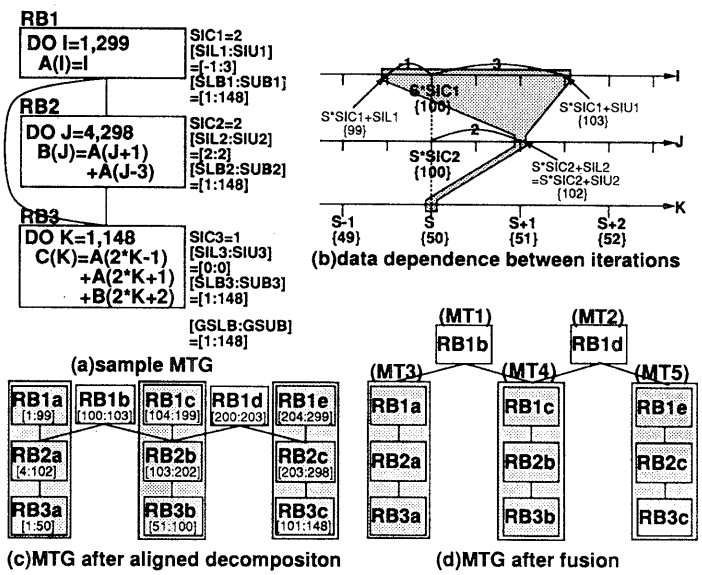


図5 サンプルプログラムのループ整合分割例
Fig. 5 An example of aligned loop decomposition.

3.1.3 グループ標準インデクス範囲計算

コンパイラは次に，RBグループ内のすべての RB_i $(1\leq i\leq m)$ でのデータ使用・定義範囲を標準ループ RB_m のインデクス範囲に換算したグループ標準インデクス範囲 $[GSLB : GSUB]$ を以下のように求める．

ⅰ）RBグループ内の各 RB_i $(1\leq i\leq m)$ のインデクス範囲 $[LB_i : UB_i]$ を標準ループ RB_m のインデクス範囲に換算した標準換算インデクス範囲 $[SLB_i : SUB_i]$ を次の式で求める．

$$SLB_i=\begin{cases}\lfloor (LB_i-SIL_i)/SIC_i \rfloor & if(SIC_i>0)\\ \lfloor (UB_i-SIL_i)/SIC_i \rfloor & if(SIC_i<0).\end{cases}$$

$$SUB_i=\begin{cases}\lceil (UB_i-SIU_i)/SIC_i \rceil & if(SIC_i>0)\\ \lceil (LB_i-SIU_i)/SIC_i \rceil & if(SIC_i<0).\end{cases}$$

ただし，LB_i と UB_i の値は，前処理として行っている定数伝播により，定数になっているものとする．

ⅱ）RBグループ内 RB_i $(1\leq i\leq m)$ の標準換算インデクス範囲 $[SLB_i : SUB_i]$ の最大範囲を表したグループ標準インデクス範囲 $[GSLB : GSUB]$ を次の式で求める．

$$GSLB=\min_i(SLB_i).$$

$$GSUB=\max_i(SUB_i).$$

i：RBグループ内の各 RB_i $(1\leq i\leq m)$

ここで，前述の図5（a）のRBグループのグループ標準インデクス範囲 $[GSLB : GSUB]$ を求めてみる．まず，各 RB_i $(1\leq i\leq 3)$ の $[SLB_i : SUB_i]$ を求めると，図5（a）（右側）に記したように，どのRBにおいても [1 : 148] となる．従って，このRBグループの $[GSLB : GSUB]$ は [1 : 148] となる．

3.1.4 RBグループ内各RBの分割

最後に，各 RB_i $(1\leq i\leq m)$ の分割後のインデクス範囲（分割インデクス範囲 $[DLB_n^i : DUB_n^i]$）を，以下の手順で求める．

ⅰ）グループ標準インデクス範囲 $[GSLB : GSUB]$ をプロセッサクラスタ数（d 個）に分割し，n $(1\leq n\leq d)$ 番目の部分領域の分割グループ標準インデクス範囲 $[DGSLB_n : DGSUB_n]$ を次の式で求める．

$$DGSLB_n=\begin{cases}GSLB & if\ (n=1)\\ DGSUB_{n-1}+1 & if\ (n>1).\end{cases}$$

$$DGSUB_n=\begin{cases}\lfloor (GSLB+\lceil (GSUB-GSLB+1)/d \rceil \times n \\ \quad -1)/LCD_{SIC} \rfloor \times LCD_{SIC} & if\ (n<d)\\ GSUB & if\ (n=d).\end{cases}$$

ただし，この式では，$DGSUB_n$ と $DGSLB_{n+1}-1$ $(1\leq n\leq d-1)$ が，RB グループ内のすべての SIC_i（値は分母が自然数の分数として保持する）の分母の最小公倍数 (LCD_{SIC}) の倍数になるように近似している．これによって，手順ⅱ）で用いる式での，$(DGSLB_n-1)\times SIC_i$ の項および $DGSUB_n\times SIC_i$ の項は整数となる．

ⅱ）各 RB_i $(1\leq i\leq m)$ での分割後の n $(1\leq n\leq d)$ 番目の部分領域の分割インデクス範囲 $[DLB_n^i: DUB_n^i]$ を次の式で求める．

$$DLB_n^i = \begin{cases} LB_i & if\ ((SIC_i>0)\wedge(n=1)) \\ & \quad \vee((SIC_i<0)\wedge(n=d)) \\ (DGSLB_n-1)\times SIC_i+1+SIU_i & \\ & if\ (SIC_i>0)\wedge(n>1) \\ DGSUB_n\times SIC_i+SIL_i & \\ & if\ (SIC_i<0)\wedge(n<d). \end{cases}$$

$$DUB_n^i = \begin{cases} DGSUB_n\times SIC_i+SIL_i & \\ & if\ (SIC_i>0)\wedge(n<d) \\ UB_i & if\ ((SIC_i>0)\wedge(n=d)) \\ & \quad \vee((SIC_i<0)\wedge(n=1)) \\ (DGSLB_n-1)\times SIC_i-1+SIU_i & \\ & if\ (SIC_i<0)\wedge(n>1). \end{cases}$$

ⅲ）各 RB_i $(1\leq i\leq m-1)$ で，$SIL_i\neq SIU_i$ の場合に限り，共通データ定義ループ（例えば，図3（c）の RB_{1b}，RB_{1d}）を生成する．

RB_i の整合分割後の n $(1\leq n\leq d-1)$ 番目の部分領域と $n+1$ 番目の部分領域の間の共通データ定義ループのインデクス範囲は，$SIC>0$ の場合に $[DUB_n^i+1: DLB_{n+1}^i-1]$ となり，一方，$SIC<0$ の場合には $[DUB_{n+1}^i+1: DLB_n^i-1]$ となる．

なお，上述した方法で求めた共通データ定義ループのインデクス範囲は，複数の後続 RB から共通にデータ依存されているインデクス範囲より SIC_i 程度広くなることがある．これは，RB グループ内の各 RB_i の SIC_i がどのような値をとる場合にも，RB グループ内の同一部分領域の RB 集合の融合を可能にし，かつ，コンパイル時の解析を容易にするためである．

ここで，前述の図5（a）の各 RB の分割後のインデクス範囲を求める．この例では，図5（a）（右側）に示した各 RB_i の $[SIL_i: SIU_i]$ と，$[GSLB: GSUB]=[1:148]$ を用いて各 RB_i を分割すると，MTG は図5（c）のようになり，分割された各 RB_i のインデクス範囲は，MTG 上の MT 番号の下に記述された範囲となる．また，図中の網掛け部分では，後述のマクロタスク融合とデータローカライゼーションが行われる．

3.2 マクロタスク融合

前述のように，本手法では，ループ整合分割後にデータ転送量の多いマクロタスク集合を融合し，実行時のダイナミックスケジューリングではこの融合マクロタスクを1つのマクロタスクとして PC に割り付ける方法をとる．このコンパイル時のマクロタスク融合により，データ転送量の多いマクロタスク集合は実行時に低オーバヘッドで同一 PC に割り当てられる．さらに，このマクロタスク融合により全体のマクロタスク数が減るため，ダイナミックスケジューリングの回数が減りそのオーバヘッドも軽減される．

3.2.1 ループ整合分割の適用された RB グループ内でのマクロタスク融合

3.1 で述べたループ整合分割の適用された各 RB グループ内で，同一部分領域の（データ転送量の多い）部分 RB 集合を融合する．これは，ループ整合分割法により，分割された部分 RB 間でデータ依存が局所化されているため実現できる．さらに，ループ整合分割時に，複数の共通データ定義ループが生成される場合には，データ依存のある共通データ定義ループを融合する．

例えば，ループ整合分割された図5（c）の MTG の場合，網掛け部分の MT は融合され，融合後の MTG は図5（d）のようになる．なお，融合後の MT 番号は（ ）に記す．

3.2.2 データ転送を考慮したマクロタスク融合

ループ整合分割の適用された RB グループ内でのマクロタスク融合を行った MTG に対して，従来より提案しているデータ転送とダイナミックスケジューリングオーバヘッドを考慮したマクロタスク融合法[12]を適用する．

3.3 融合 MT 内でのデータローカライゼーション用データ転送コード生成

3.2 で述べた融合手法により生成された融合マクロタスクに，データローカライゼーションを適用する．融合マクロタスク内のループ間データ転送は，ローカルメモリ（LM）を介して行われ，共有メモリ（CM）を介したデータ転送オーバヘッドが軽減される．

3.3.1 データローカライゼーションの有効な配列変数の検出

融合マクロタスク内のループで，定義または使用される各配列変数に対して，データローカライゼーション適用時のデータ転送時間（詳細は後述）および CM を介した転送時間（CM へのロード・ストア時間）を計算する．このとき，データローカライゼーション適用時の転送時間が，CM 経由転送時の転送時間より小さい場合，その配列変数にデータローカライゼーションを適用する．

3.3.2 融合 MT 内で定義されないデータを CM から LM へ転送するコードの生成

データローカライゼーションの適用される配列変数で，使用されるデータが融合マクロタスク内で使用前に LM に定義されない場合，そのデータを融合マクロタスク内のループ実行前に CM から LM に転送する命令を生成する．ここでは，**表 2** に示すデータ転送パターン中の CM→LM 転送を用いる．

例えば，**表 3** の融合 MT の例では，配列変数 A，B，C にデータローカライゼーションが適用される．このとき，表 3（右上）に示される A[101 : 101]，A[201 : 201]，B[101 : 200] のデータは，融合 MT 内で使用前に LM に定義されないので，融合 MT 内のループ実行前に CM から LM に転送するマシンコードを生成する．

3.3.3 ローカルメモリ経由データ転送コードの生成

データローカライゼーションの適用される配列変数は，CM へのロード・ストア命令の代わりに，各プロセッサ上の LM へのロード・ストア命令を生成する．ここでは，表 2 の loadLM，storeLM のデータ転送パターンを用いる．

例えば，表 3（左側）の融合 MT に，データローカライゼーション（LM 経由データ転送）を適用すると，データ転送パターンは表 3（右側）に示したようになり，この時のデータ転送時間は，従来の CM 経由転送時の 3192 クロックから 1332 クロックに短縮さ

表 2 OSCAR 上でのデータ転送パターンとコスト
Table 2 Data transfer patterns and costs on OSCAR.

データ転送パターン	クロック／データ
loadCM, storeCM	4
loadLM, storeLM	1
CM→LM 転送，LM→CM 転送	5(5.25†)

† 16 データ単位のブロックデータ転送時の 1 データ当たりの平均転送時間

表 3 CM 経由転送と LM 経由転送における転送時間の比較
Table 3 A comparison between transfer time via CM and transfer time via LM.

融　合　MT	従来の CM 経由データ転送		LM 経由データ転送（ローカライズ）	
	転送パターン（配列）	クロック	転送パターン（配列）	クロック
			CM→LM(A[101 : 101])	5*1
			CM→LM(A[201 : 201])	5*1
			CM→LM(B[101 : 200])	5.25*96+5*4
ループ 1				
DO I=102, 200	storeCM(A[102 : 200])	4*99	storeLM(A[102 : 200])	1*99
A(I)=B(I)*2	loadCM(B[102 : 200])	4*99	loadLM(B[102 : 200])	1*99
ループ 2				
DO J=101, 200	loadCM(A[102 : 201])	4*100	loadLM(A[102 : 201])	1*100
C(J)=A(J+1)+B(J)	loadCM(B[101 : 200])	4*100	loadLM(B[101 : 200])	1*100
	storeCM(C[101 : 200])	4*100	storeLM(C[101 : 200])	1*100
ループ 3				
DO K=101, 200	loadCM(A[101 : 200])	4*100	loadLM(A[101 : 200])	1*100
=A(K)+B(K)+C(K)	loadCM(B[101 : 200])	4*100	loadLM(B[101 : 200])	1*100
	loadCM(C[101 : 200])	4*100	loadLM(C[101 : 200])	1*100
			(LM→CM(A[102 : 200])†)	(5.25*96+5*3†)
			(LM→CM(C[101 : 200])†)	(5.25*96+5*4†)
計		3192		1332(2375†)

† 融合 MT で定義されたデータが融合 MT の後続 MT で使用される場合

れる．ただし，ローカライズされたデータが融合 MT の後続 MT で使用される場合には，LM から CM へのデータ転送が必要となるため，データローカライゼーション適用時のデータ転送時間は 2375 クロックとなる．

このようなデータローカライゼーションによるデータ転送時間の短縮効果はマシンによって異なるが，CMと LM のアクセス時間の差が大きく，また，高速ブロック転送命令をもっているマルチプロセッサシステムでは，さらに大きな効果が得られると考えられる．

4. OSCAR 上での性能評価

本章では，提案するループ整合分割・融合を伴うデータローカライゼーション手法を複数のプログラムに適用し，OSCAR[15] 上で評価した結果について述べる．

ここで，OSCAR は，ローカルメモリ (LM) と分散型共有メモリを持つ 32 ビット RISC プロセッサ (PE) を，集中型共有メモリ (CM) に 3 本のバスで接続した共有メモリ型マルチプロセッサシステムである．OSCAR 上の各 PE は平等結合となっているが，複数 PE をソフトウェア的にグループ化することにより，マルチプロセッサクラスタシステムとしても利用できる．なお，OSCAR は最大 16 PE を接続できるが現在 7 PE のみが稼働中であるため，以下の性能評価では，各プロセッサクラスタ (PC) を 1 PE 構成とした 6 PC 構成を用いる．

4.1 スプライン補間法プログラムによる性能評価

まず，最初のテストプログラムは，スプライン補間を行うプログラムであり，これは 8 つの Doall ループ，2 つのシーケンシャルループ，2 つの基本ブロックで構成されている．このプログラムを OSCAR 上で実行した結果は**表 4** のようになる．

従来の Doall 処理では，シーケンシャルループを並列処理できないため，6 台の PE を用いても，1 PE のときの 1/3.02 倍というように処理時間をあまり短縮できない．一方，ループ均等分割を伴うマクロデータフロー処理 (MDC) では，シーケンシャルループと他のマクロタスクを，その間にデータ依存がない場合に並列実行できるため，6 PE を用いると 1 PE の処理時間の 1/3.49 倍と Doall 処理より処理時間を短縮することが可能となる．

さらに，本論文で提案するループ整合分割・融合・データローカライゼーションを適用した場合には，集中型共有メモリを介したデータ転送オーバヘッドおよびダイナミックスケジューリングオーバヘッドが軽減されるため，ループ均等分割を用いたマクロデータフロー処理 (MDC) より，3 PE で 17.6%，6 PE で 21.6%処理時間が短縮されることが確かめられた．

4.2 CG 法プログラムによる性能評価

次に，連立 1 次方程式間接求解法である CG (Conjugate Gradient) 法プログラムのメイン収束ループに提案手法を適用する．このメインループは，4 つの Doall ループ，2 つのリダクションループ (総和計算ループ)，5 つの基本ブロックで構成されている．

適用結果は**表 5** に示すように，従来の Doall 処理が 1 PE 上での処理時間を 3 PE で 1/2.86，6 PE で 1/4.75 に短縮し，通常のループ均等分割を伴うマクロデータフロー処理 (MDC) も 3 PE で 1/2.75，6 PE で 1/4.78 に短縮するという Doall 処理とほぼ同様な結果であった．これに対して，ループ整合分割を用いたデータローカライゼーションを伴うマクロデータフロー処理 (MDC) では 3 PE で 1/3.17，6 PE で 1/

表 4 スプライン補間法プログラムでの性能評価
Table 4 An execution result for Spline-Interporation program.

PE 数	実 行 方 式	時間[s]	1PE 比
1	Sequential 処理	5.731	1
3	Doall 処理	2.623	1/2.19
3	MDC (均等分割)	2.317	1/2.47
3	MDC (整合分割・融合・ローカライズ)	1.910	1/3.00
6	Doall 処理	1.897	1/3.02
6	MDC (均等分割)	1.644	1/3.49
6	MDC (整合分割・融合・ローカライズ)	1.288	1/4.45

表 5 CG 法プログラムでの性能評価
Table 5 An execution result for CG method program.

PE 数	実 行 方 式	時間[s]	1PE 比
1	Sequential 処理	4.793	1
3	Doall 処理	1.677	1/2.86
3	MDC (均等分割)	1.742	1/2.75
3	MDC (整合分割・融合・ローカライズ)	1.493	1/3.17
6	Doall 処理	1.009	1/4.75
6	MDC (均等分割)	1.002	1/4.78
6	MDC (整合分割・融合・ローカライズ)	0.805	1/5.95

5.95 というように顕著に処理時間が短縮されることが確かめられた.

なお, 3PE でローカライゼーションを適用したとき 1PE に比べて3倍以上の速度向上が得られているのは, 1PE での処理においては, データサイズが PE 上のローカルメモリサイズより大きいので集中型共有メモリを使用しているが, ローカライゼーション後は一部のデータ授受がローカルメモリ上で行われているためである.

5. おわりに

本論文では, マクロデータフロー処理において, 複数ループ間データ転送にローカルメモリを有効に利用し, データ転送オーバヘッドを軽減するデータローカライゼーション手法を提案した. さらに, このデータローカライゼーションを実現するために, 分割したループ間でのデータ依存を局所化するループ整合分割手法とデータ転送量の多いループ集合を融合するマクロタスク融合手法を提案した.

また, OSCAR 上での性能評価の結果より, 従来の集中型共有メモリ経由転送を用いたマクロデータフロー処理に比べ, 処理時間を 15%～20% 程度短縮できることが確かめられ, ループ整合分割を用いたデータローカライゼーション手法の有効性が確認された.

今後の課題としては, スタティックスケジューリングモードでのマクロデータフロー処理におけるより広範囲に渡ったデータローカライゼーションの実現, データローカライゼーションとデータプレローディング・ポストストアリング技術との統合によるデータ転送オーバヘッドのさらなる軽減などがあげられる.

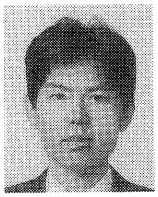
謝辞 本研究の一部は, 文部省科学研究費 (特別研究員奨励費 05-3760, 一般研究(b) 05452354, 一般研究(c) 05680284) により行われた.

参考文献

1) Padua, D. A. and Wolfe, M. J.: Advanced Compiler Optimizations for Super Computers, *Comm. ACM*, Vol. 29, No. 12, pp. 1184-1201 (1986).
2) Banerjee, U.: *Dependence Analysis for Supercomputing*, Kluwer Academic, Pub. (1988).
3) Tu, P. and Padua, D.: Automatic Array Privatization, *6th Annual Workshop on Languages and Compilers for Parallel Computing* (1993).
4) Li, Z.: Array Privatization for Parallel Execution of Loops, *Proc. of the 1992 ACM Int'l Conf. on Supercomputing*, pp. 313-322 (1992).
5) High Performance Fortran Forum: High Performance Fortran Language Specification, DRAFT Ver. 1.0, High Performance Fortran Forum (1993).
6) Hiranandani, S., Kennedy, K., Koelbel, C., Kremer, U. and Tseng, C.-W.: An Overview of the Fortran D Programming System, *Proc. 4th Workshop on Languages and Compilers for Parallel Computing* (1991).
7) Li, J. and Chen, M.: Compiling Communication-Efficient Programs for Massively Parallel Machines, *IEEE Trans. on Parallel and Ditributed System*, Vol. 2, No. 3, pp. 361-376 (1991).
8) Gupta, M. and Banerjee, P.: Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers, *IEEE Trans. on Parallel and Ditributed System*, Vol. 3, No. 2, pp. 179-193 (1992).
9) Anderson, J. M. and Lam, M. S.: Global Optimizations for Parallelism and Locality on Scalable Parallel Machines, *Proc. of the SIGPLAN '93 Conference on Programming Language Design and Implementation*, pp. 112-125 (1993).
10) 本多, 岩田, 笠原: Fortran プログラム粗粒度タスク間の並列性検出手法, 信学論 (D-I), Vol. J 73-D-I, No. 12, pp. 951-960 (1990).
11) Kasahara, H., Honda, H., Mogi, A., Ogura, A., Fujiwara, K. and Narita, S.: Multi-Grain Parallelizing Compilation Scheme for OSCAR, *4th Workshop on Language and Compilers for Parallel Computing* (1991).
12) 笠原, 合田, 吉田, 岡本, 本多: Fortran マクロデータフロー処理のマクロタスク生成手法, 信学論 (D-I), Vol. J 75-D-I, No. 8, pp. 511-525 (1992).
13) 本多, 合田, 岡本, 笠原: Fortran プログラム粗粒度タスクの OSCAR における並列実行方式, 信学論 (D-I), Vol. J 75-D-I, No. 8, pp. 526-535 (1992).
14) 笠原: 並列処理技術, コロナ社 (1991).
15) 笠原, 成田, 橋本: OSCAR のアーキテクチャ, 信学論(D), Vol. J 71-D, No. 8, pp. 1440-1445 (1988).
16) Aho, A. V., Sethi, R. and Ullman, J. D.: *Compilers (Principles, Techniques, and Tools)*, Addison Wesley (1988).
17) Girkar, M. and Polychronopoulos, C. D.: Automatic Extraction of Functional Parallelism from Ordinary Programs, *IEEE Trans. on Parallel and Distributed System*, Vol. 3, No. 2, pp. 166-178 (1992).
18) Wolfe, M.: *Optimizing Super-compilers for*

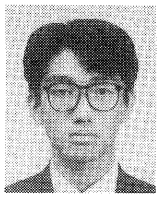
Super-computers, MIT Press (1989).
19) Coffman, E. G.: *Computer and Job-Shop Scheduling Theory*, A Wiley-Interscience Publication (1976).

(平成5年11月15日受付)
(平成6年6月20日採録)



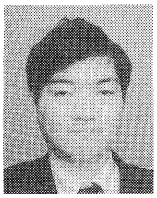
吉田 明正（正会員）

1968年生．1991年早稲田大学理工学部電気工学科卒業．1993年同大大学院修士課程修了．現在，同大大学院博士後期課程在学中．1993年より日本学術振興会特別研究員．並列化コンパイラ，並列処理方式，マルチプロセッサアーキテクチャ等の研究に従事．電子情報通信学会会員．



前田 誠司（正会員）

1969年生．1992年早稲田大学理工学部電気工学科卒業．1994年同大大学院修士課程修了．同年(株)東芝入社．在学中，並列化コンパイラの研究に従事．現在，同社研究開発センターに所属．



尾形 航（正会員）

1967年生．1991年早稲田大学理工学部電気工学科卒業．1993年同大大学院修士課程修了．現在，同大大学院博士後期課程在学中．並列実行方式，近細粒度並列処理手法，計算機アーキテクチャの研究に従事．

笠原 博徳（正会員）

1957年生．1980年早稲田大学理工学部電気工学科卒業．1985年同大大学院博士課程修了．工学博士．1983年～1985年早稲田大学電気工学科助手．1985年カリフォルニア大バークレー短期客員研究員，日本学術振興会特別研究員．1986年早稲田大学電気工学科専任講師，1988年助教授，1991年情報学科助教授，現在に至る．1989年～1990年イリノイ大 Center for Supercomputing R&D 客員研究員．1987年 IFAC World Congress 第1回 Young Author Prize 受賞．主な著書「並列処理技術」（コロナ社）．マルチプロセッサスケジューリング，スーパコンピューティング，並列化コンパイラ等の研究に従事．電子情報通信学会，電気学会，シミュレーション学会，ロボット学会，IEEE，ACM等の会員．