共有メモリマルチプロセッサ上での データローカライゼーション対象マクロタスク決定手法

八木 哲志† 板垣 裕樹† 中野 啓史†
石坂 一久§† 小幡 元樹§† 吉田 明正§‡ 笠原 博徳§†

†早稲田大学 理工学部 電気電子情報工学科
‡東邦大学 理学部 情報科学科
§アドバンスト並列化コンパイラ研究体

概要

本稿では，階層型粗粒度タスク並列処理におけるマクロタスクグラフよりデータローカライゼーションの対象となるマクロタスク集合を，並列性とデータローカリティの両方を考慮し決定する手法を提案する．本手法は自動並列化コンパイラ OSCAR Fortran コンパイラ上に実装されており，対象アーキテクチャは分散キャッシュや分散共有メモリを持つ共有メモリマルチプロセッサマシンである．本データローカライゼーション手法を用いた階層型粗粒度タスク並列処理の性能を 4 プロセッサ SMP ワークステーション Sun Ultra80 上で評価をした結果，Sun Forte version 6 update 1 の自動ループ並列化コンパイラによる実行と比べて，TOMCATV で実行時間が 3.00 倍，SWIM で 4.36 倍の速度向上が得られ，提案手法の有効性が確認された．

A Macrotask selection technique for Data-Localization Scheme on Shared-memory Multi-Processor

Satoshi Yagi† Hiroki Itagaki† Hirofumi Nakano†
Kazuhisa Ishizaka§† Motoki Obata§† Akimasa Yoshida§‡ Hironori Kasahara§†

†Department of Electrical, Electronics and Computer Engineering, Waseda University
‡Department of Information Science, Toho University
§Advanced Parallelizing Compiler Research Group

Abstract

This paper proposes a macrotask selection scheme that determines sets of macrotasks, to which data localization can be applied on a macrotask graph for hierarchical coarse grain parallel task processing in consideration of both of parallelism and data locality. This technique is implemented at OSCAR Fortran multigrain parallelizing Compiler. The target architectures are multiprocessor systems is a machine with distributed cache or distributed shared memory. The performance evaluation shows that hierarchical coarse-grain prallel processing with the data-localization by OSCAR compiler on Sun 4 processor SMP workstation Ultra80 gives up 3.00 times speedup for SPEC95fp TOMCATV and 4.36 times speedup for SWIM comparing with the automatic loop parallelization by Sun Forte version 6 update 1 compiler on 4 processor.

1. はじめに

マルチプロセッサシステム上での自動並列化コンパイラを用いた並列処理では従来よりループ並列化手法[12)]が用いられている．例えば，イリノイ大学の Polaris[3)] やスタンフォード大学の SUIF[2)] などのような先端の並列化コンパイラでは，強力なデータ依存解析手法とループリストラクチャリング手法を組み合わせることでさまざまな形状のループが並列化可能になっている．しかしながら，これらのループ並列化手法では，単一ループのイタレーション間の局所的並列性しか利用することができず，今後大幅な性能向上は困難と言われている．このループ並列化の限界を越え，より大きな並列性を抽出するために，プログラム全域に渡る並列性解析により，ループやサブルーチン等の粗粒度タスクレベルの並列性を利用する階層型粗粒度タスク並列処理が有効である[13)9)]．

この階層型粗粒度タスク並列処理を，近年普及している共有メモリ型マルチプロセッサシステム上で効果的に実現するためには，粗粒度タスク間並列性を最大限に利用し，かつ PE 上の分散共有メモリ，分散キャッシュを有効利用することが必要となる．このようなデータ分散手法に関しては多くの研究が行われており，High Performance Fortran（HPF）のようなユーザ指定によるデータ分散[4)]，ループ内の作業配列をローカル化する Array Privatization 法[10)]，自動データ分散法[5)1)8)] などが提案されている．しかしながら，これらの方式では，ループのイタレーション間のみの並列性利用時のデータローカリティ最適化を前提にしている．

また本論文で扱う，粗粒度タスク並列処理を対象とした自動データ分散に関しては，ループ分割後のタスク垂

直実行によるローカリティ利用[11]，複数の粗粒度タスク間での共有データをローカルメモリ経由で授受するデータローカライゼーション手法[14)7)]が提案されている．しかしながら，従来のデータローカライゼーション手法では，直列型に接続されたループ集合をデータローカライゼーションの単位として扱っていた．それに対して，本稿では，任意形状データ依存マクロタスクグラフ上の広範囲に渡ってのデータローカライゼーション手法を提案する．

本論文の構成は以下のとおりである．第2章では，階層型粗粒度タスク並列処理について概説する．第3章では，データローカライゼーション手法，及び提案するデータローカライゼーション対象マクロタスク決定手法について述べる．第4章では，Sun Ultra80 4プロセッサSMPワークステーション上で行った性能評価の結果について述べる．

2. 階層型粗粒度タスク並列処理

本章では，本手法の対象アーキテクチャと階層的な粗粒度タスク並列処理を実現する階層型マクロデータフロー処理について述べる．

2.1 対象マルチプロセッサアーキテクチャ

階層型マクロデータフロー処理では，分散キャッシュあるいは分散共有メモリを持った共有メモリ型マルチプロセッサシステムを対象とする．

階層型粗粒度タスク並列処理を適用する場合，ソフトウェア的にプロセッサをグループ化して，階層的にプロセッサクラスタを定義する．具体的には，まず，マルチプロセッサシステム全体を第0階層プロセッサクラスタ（PC）と定義し，第0階層PC内のPEをグループ化して第1階層PCを定義する．同様に，第n階層PC内のPEをグループ化して第$(n+1)$階層PCを定義する．

2.2 階層型マクロデータフロー処理

階層型マクロデータフロー処理[13)]では，ループやサブルーチン等の粗粒度タスク（マクロタスク）が，プロセッサクラスタに割り当てられて並列処理される．このとき，プロセッサクラスタに割り当てられたマクロタスク内部では，ループ並列処理，あるいは，階層的に粗粒度並列処理が適用される．

2.2.1 階層型マクロタスク生成

階層型マクロデータフロー処理手法では，まず，プログラム（全体を第0階層マクロタスクとする）を第1階層マクロタスクに分割する．マクロタスク（MT）は，擬似代入文ブロック（BPA），繰り返しブロック（RB），あるいは，サブルーチンブロック（SB）の3種類から構成される[13)]．

次に，第1階層マクロタスク（RBまたはSB）内に複数のサブマクロタスク（サブBPA，サブRB，サブSB）を含んでいる場合には，それらのサブマクロタスクを第2階層マクロタスクとして定義する．

2.2.2 階層型マクロタスクグラフ生成

各階層のマクロタスク生成後，各階層内で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解析し，階層型マクロフローグラフ[13)]を生成する．

次に，コントロール依存とデータ依存を考慮しマクロタスク間並列性を最大限に引き出すために，各マクロタスクの最早実行可能条件を解析する．この最早実行可能条件は，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の並列性を表す．この各マクロタスクの最早実行可能条件は，階層型マクロタスクグラフ（MTG）[13)]で表すことができる．

2.2.3 並列処理用実行コード生成

階層型マクロデータフロー処理は，コンパイル時にマクロタスクをプロセッサクラスタ（PC）に割り当てるスタティックスケジューリング方式と，実行時にマクロタスクをプロセッサクラスタに割り当てるダイナミックスケジューリング方式[13)]があるが，本論文ではダイナミックスケジューリング方式を採用する．またダイナミックスケジューリング用のスケジューリングルーチンはコンパイラによって生成される．また並列処理用のコードはOpenMPで記述された並列コードで出力され，最終的に各環境におけるネイティブコンパイラにより出力コードをコンパイルし実行コードを得る．

3. データローカライゼーション手法

階層型粗粒度タスク並列処理の効率をより向上させるためには，分散共有メモリ，分散キャッシュを有効利用してデータ転送オーバヘッドを軽減することが必要となる．

本データローカライゼーション手法は，データローカライゼーションの対象とするマクロタスクを選択しターゲットループグループ（TLG）を生成(3.1)，TLG内ループ間データ依存解析（3.2），TLG内ループの整合分割（3.3），データ転送量の多いマクロタスクを同一PCに割り当てるダイナミックスケジューリング（3.4）という手順により実現される．

3.1 データローカライゼーション対象MT選択（TLG生成）

従来提案されてきたデータローカライゼーション対象MT生成手法[14)7)]では，対象領域がMTG上で直列データ依存エッジにより接続されているループ集合に限られるという制約があった．

それに対して，本稿で提案するデータローカライゼーション対象MT選択手法であるConsecutive-Adjacent法では並列性とデータローカリティ共に考慮した上で，対象を任意形状のマクロタスクグラフへと拡張した手法である．以下，当手法について述べる．

(1) 標準ループの選定
データローカライゼーションの対象とする階層のマクロタスクグラフMTG_{init}において，最も性能に影響を及ぼすと推測されるマクロタスクであるクリティカルパス（CP）上でマクロタスクのうち最も大きな処理時間(この処理時間には共有メモリアクセス時間を含めて計算している)を持つマクロタスクを標準ループMT_{std}とする．この標準ループは，ローカライゼーションのためのターゲットループグループを生成する際の基準となるループとして使われ，MT_{std}が$TLG_{current}$の初期構成要素となる．ここで標準ループとなりうるマクロタスクの種類は，Doallループ，Reductionループ，ループキャリッドデータ依存(リカレンス)によるSequentialループのいずれかである．

(2) Consecutive-part生成
(1)で選定されたMT_{std}を基準にConsecutive-partを生成する．このConsecutive-partはマクロタスクグラフ上で一直線にデータ依存で結合された整合可能なループ集合である．
まずMT_{std}を$MT_{current}$とし，Consecutive-partの先行部を構成するマクロタスク集合

$Consecutive\_part_{pred}$ の初期構成要素とする．この $MT_{current}$ が直接データ依存している先行マクロタスク集合 MT_{pred} の内，$MT_{current}$ と整合可能（後述）かつ，最も処理時間の大きなマクロタスク $MT_{pred\_max\_cost}$ を $Consecutive\_part_{pred}$ の構成要素とし，$TLG_{current}$ の構成要素とする．次に，追加された $MT_{pred\_max\_cost}$ を新たに $MT_{current}$ とし，同様の手順で新たな $MT_{pred\_max\_cost}$ を $Consecutive\_part_{pred}$ に追加し $TLG_{current}$ の構成要素とする．同手順を $MT_{current}$ に対する直接先行マクロタスクがなくなるまで繰り返す．

次に MT_{std} を再び $MT_{current}$ とし，Consecutive-part の後続部を構成するマクロタスク集合 $Consecutive\_part_{succ}$ の初期構成要素とする．この，$MT_{current}$ に直接データ依存している後続マクロタスク集合 MT_{succ} の内，$MT_{current}$ と整合可能（後述）かつ，最も処理時間の大きなマクロタスク $MT_{succ\_max\_cost}$ を Consecutive-part の後続部 ($Consecutive\_part_{succ}$) に追加し，$TLG_{current}$ の構成要素とする．次に今追加された $MT_{succ\_max\_cost}$ を新たに $MT_{current}$ とし，同様の手順で新たな $MT_{succ\_max\_cost}$ を $Consecutive\_part_{succ}$ に追加し $TLG_{current}$ の構成要素とする．同手順を $MT_{current}$ に対する直接後続マクロタスクがなくなるまで繰り返す．

上記の手順により求められた $Consecutive\_part_{pred}$ と $Consecutive\_part_{succ}$ に含まれるマクロタスクを合わせて $Consecutive\_part$ とする．

ここで，2 つのループが整合可能であるとは，以下のすべての条件を満たすことである．

- 各ループが Doall ループ，Reduction ループ，ループキャリッドデータ依存 (リカレンス) による Sequential ループのいずれかである．
- ループ間に配列変数のデータ依存が存在する．
- ループ間において同一配列の分割次元が一致し，その分割次元の配列添字がループ制御変数の 1 次式で表されている．
- ループ間にデータ依存を導く各配列に対して，配列添字中のループ制御変数係数のループ間での比が一定の場合である．

(3) Adjacent-part 生成

(2)で生成された Consecutive-part($Consecutive\_part$) を基準に Adjacent-part を生成する．この Adjacnet-part は Consecutive-part に直接隣接したマクロタスクにより構成される．

$Consecutive\_part$ を構成する各マクロタスクを MT_{sub_std} とし，$Adjacent\_part_{succ}$ の初期構成要素とする．そして，まだいずれのターゲットループグループにも所属していないマクロタスクのうち各 MT_{sub_std} に対し整合可能である直接後続マクロタスク MT_{succ} を $Adjacent\_part_{succ}$ に追加し，$TLG_{current}$ の構成要素とする (図 1(b) の MT6，Diverge 型)．以上を各 MT_{sub_std} の各 MT_{succ} 毎に繰り返す（各 MT_{sub_std} に対し整合可能である直接後続マクロタスク MT_{succ} の個数だけ $Adjacent\_part_{succ}$ は作られる）．

同様に $Consecutive\_part$ を構成する各マクロタスクを MT_{sub_std} とし，$Adjacent\_part_{pred}$ の初期構成要素とする．そして，まだいずれのターゲットループグループにも所属していないマクロタスクのうち，各 MT_{sub_std} に対し整合可能である直接先行マクロタスク MT_{pred} を $Adjacent\_part_{pred}$ に追加し，$TLG_{current}$ の構成要素とする．(図 1(b) の MT1，Merge 型) 以上を各 MT_{sub_std} の各 MT_{pred} 毎に繰り返す（各 MT_{sub_std} に対し整合可能である直接後続マクロタスク MT_{pred} の個数だけ $Adjacent\_part_{pred}$ は作られる）．

(4) 次 TLG 生成

以上の手順により生成された $TLG_{current}$ に含まれるマクロタスク集合を現在のマクロタスクグラフ $MTG_{current}$ より取り除いた後，(1) に戻り，別の TLG を生成する．ただし標準ループを選択する際は，マクロタスクグラフの初期状態 MTG_{init} における CP 上でまだいずれのターゲットループグループにも選択されていないマクロタスクの中で標準ループとなりうる種類のマクロタスクがあれば，その中で最も大きな処理時間を持つマクロタスクが新たな標準ループ MT_{std} となる．同 Critical path 上に標準ループとなりうる種類のマクロタスクが存在しなければ，Critical path 上以外のマクロタスクの中で最も大きな処理時間を持つマクロタスクを新たな標準ループ MT_{std} とする．

3.2 TLG 内ループ間データ依存解析

本手法では，3.1で生成された各 TLG に対し，各 TLG の標準ループから TLG 内 MT へのイタレーションに関するデータ依存（**Inter-Loop-Dependence**（**ILD**））を求める．ここで，TLG は $MT_i(1 \leq i \leq end)$ により構成されているものとし，MT_{std} を標準ループとする．

MT_i の ILD の解析結果は，

$ILD_{MT_i}^{(MT_{std},K)} = [K * c_i + l_i : K * c_i + u_i]$

のように表現する．本式では，標準ループ MT_{std} の K 番目のイタレーションが，MT_i の $K*c_i+l_i$ 番目〜$K*c_i+u_i$ 番目のイタレーションに，直接的あるいは間接的（他 MT を経由して）にデータ依存していることを表している．図 1(c) は，解析結果の例である．

なお，上記解析前に，直接接続された TLG 内マクロタスク間のイタレーションに関するデータ依存 **Direct-Inter-Loop-Dependence**（**DirILD**）を求めておく．ここで $DirILD_{MT_i}^{(MT_x,K)} = [K * C_i^x + L_i^x : K * C_i^x + U_i^x]$ と表現すると，本式は，MT_x の K 番目のイタレーションが，MT_i の $K * C_i^x + L_i^x$ 番目〜$K * C_i^x + U_i^x$ 番目のイタレーションに直接的にデータ依存していることを表している．

ILD 解析方法

3.1節で生成したさまざまな形状の TLG において，その構成要素である $MT_i(1 \leq i \leq end)$ の ILD を求めるために，従来の PreToPre 法[14)7)] に加えて，新たに PostToPost 法，PostToPre 法，PreToPost 法を導入する．

まず，標準ループの ILD を次のように定義する．

$ILD_{MT_{std}}^{(MT_{std},K)} = [K + 0 : K + 0]$.

次に，標準ループ以外の $MT_i(i \neq std)$ の ILD の値 ($ILD_{MT_i}^{(MT_{std},K)}$) を，下記の手順で求める．この際，解析済の他 MT_x の ILD の値（$ILD_{MT_x}^{(MT_{std},K)} = [K + l_x : K + u_x]$ とする）と，MT_i の MT_x に対する DirILD の値（$DirILD_{MT_i}^{(MT_x,K)} = [K + L_i^x : K + U_i^x]$ とする）を用いる．なお，ここでは理解を容易とするため K の係数は 1 として説明する．

(1) TLG 内 CP 上の $MT_i(std \geq x > i \geq 1)$ の場合

（図 1(b) の MT2 と MT3）
$ILD_{MT_i}^{(MT_{std},K)} = [K + l_x + L_i^x : K + u_x + U_i^x]$.
（**PreToPre 法**）
なお，MT_i にデータ依存する MT_x が複数存在する場合には，各々の MT_x から求めた最大範囲を ILD とする．
例えば，図 1(b) の MT2 の ILD を求める場合，解析済の MT3 の ILD の値 $ILD_{MT_3}^{(MT_{std},K)} = [K + 0 : K + 1]$ と，MT3 から MT2 へ DirILD の値 $DirILD_{MT_2}^{(MT_3,K)} = [K + 0 : K + 1]$ を用いて，$ILD_{MT_2}^{(MT_{std},K)} = [K + 0 + 0 : K + 1 + 1] = [K + 0 : K + 2]$ と求められる．

(2) TLG 内 CP 上の $MT_i(std \leq x < i \leq end)$ の場合
（図 1(b) の MT5）
$ILD_{MT_i}^{(MT_{std},K)} = [K + l_x - U_x^i : K + u_x - L_x^i]$.
（**PostToPost 法**）
なお，MT_i がデータ依存する MT_x が複数存在する場合には，各々の MT_x から求めた最大範囲を ILD とする．

(3) TLG 内 CP 上 MT_x より先行データ依存エッジで接続された MT_i の場合（図 1(b) の MT1）
(i)$1 \leq x \leq std$ の場合
前述の **PreToPre 法**により ILD を求める．
(ii)$std < x \leq end$ の場合
$ILD_{MT_i}^{(MT_{std},K)} = [K + u_x + L_i^x : K + l_x + U_i^x]$.
（**PostToPre 法**）

(4) TLG 内 CP 上 MT_x より後続データ依存エッジで接続された MT_i の場合（図 1(b) の MT6）
(i)$1 \leq x \leq std$ の場合
$ILD_{MT_i}^{(MT_{std},K)} = [K + u_x - U_x^i : K + l_x - L_x^i]$.
（**PreToPost 法**）
(ii)$std < x \leq end$ の場合
前述の **PostToPost 法**により ILD を求める．

3.3 ループ整合分割

マクロタスク（ループ）間でのデータ転送をプロセッサ上の分散共有メモリ，分散キャッシュあるいはローカルメモリ上で効果的に行うためには，各タスクにおけるデータの使用範囲が等しくなるように，複数のループを整合して分割する必要がある．

本ループ整合分割法では，まず，前述の TLG ごとに，データの使用範囲が等しくなるように複数ループを整合分割する．

例えば，図 1(b) のような複数の先行・後続データ依存タスクを持つ TLG（ソースコードは図 1(a) に対応）が MTG から抽出されたとする．この場合，本手法では，6 つの MT（ループ）間におけるイタレーションに関するデータ依存を解析し，図 1(c) のような解析結果を得る．図 1(c) では，最大コストを持つ MT4 を標準（基準）ループとしており，MT1，MT2，MT3 の横軸方向には，MT4 の K 番目のイタレーションがデータ依存しているイタレーション範囲が示されている．また，MT5 と MT6 の横軸方向には，MT4 の K 番目のイタレーションにデータ依存しているイタレーション範囲が示されている．なお，ここでは他 MT を経由した間接的なデータ依存も含まれる．

次に，図 1(c) の解析結果を用いて，各ループが整合分割される．3 分割の場合には図 1(d) のように，各 MT は，3 つの **LR**(Localizable-Region) と 2 つの **CAR**(Commonly-Accessed-Region) に分割される．ここで，標準ループより先行の CAR（CAR_{pre} と呼ぶ）は，後続の複数 LR が共通にデータ依存しているイタレーション集合を表し，標準ループより後続の CAR（CAR_{post} と呼ぶ）は，先行の複数 LR に共通にデータ依存しているイタレーション集合を表している．

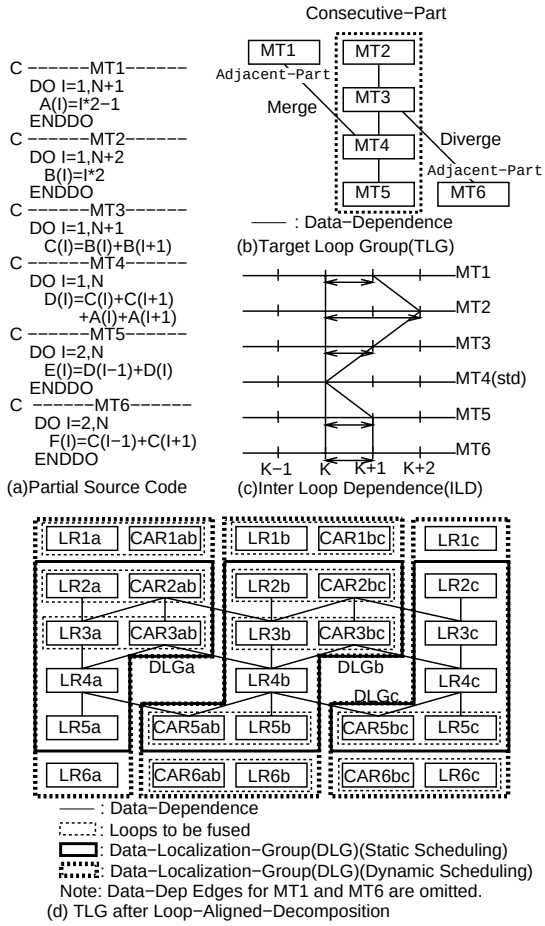

図 1 ループ整合分割の例．

ループ整合分割手順

コンパイラは，各 TLG 内で使用されるデータの範囲を，標準ループ MT_{std} のインデクス範囲で表したグループ変換インデクス範囲（$GCIR$）を求める．その後，$GCIR$ を分散キャッシュあるいは分散共有メモリのサイズを考慮して PC 数の整数倍（n）に均等に分割し，この各分割範囲（分割グループ変換インデクス範囲）を $DGCIR^p$（$1 \leq p \leq n$）と表現する．

例えば，図 1(b) の TLG において N=300，分割数 n=3 とした場合，$GCIR$ は [1:300] であり，$DGCIR^1 = [1 : 100]$，$DGCIR^2 = [101 : 200]$，$DGCIR^3 = [201 : 300]$ となる．

次に，$DGCIR^p$（$1 \leq p \leq n$）と TLG 内のループ間データ依存の解析結果を用いて，各 MT_i（$1 \leq i \leq end$）を分割する．具体的には，部分標準ループ（ループインデクス上下限値が $DGCIR^p$ のもの）がデータ依存する MT_i のイタレーション集合を，LR（Localizable-Region）として生成し，複数の部分標準ループ（ループインデクス上下限値が $DGCIR^p$ と $DGCIR^{p+1}$ のもの）が共通に

データ依存しているMT_iのイタレーション集合を，CAR（Commonly-Accessed-Region）として生成する．なお，CARは標準ループより先行MTの場合は，CAR_{pre}として生成され，標準ループより後続MTの場合は，CAR_{post}として生成される．ここで生成されたCARはLRに比べて処理時間が非常に小さいループであり，実行時スケジューリングオーバーヘッドの軽減のため，スケジューリングの前に，隣接するLR（CAR_{pre}は左のLR，CAR_{post}は右のLR）に融合される．

前述の図1(b)のTLG（N=300，分割数n=3）は，図1(d)のように分割される．分割後の各ループのループインデクス範囲は，表1の通りである．

表1 ループ整合分割後のループインデクス範囲

	LR^a	CAR^{ab}	LR^b	CAR^{bc}	LR^c
MT1	1:100	101:101	102:200	201:201	202:301
MT2	1:100	101:102	103:200	201:202	203:302
MT3	1:100	101:101	102:200	201:201	202:301
MT4	1:100		101:200		201:300
MT5	2:100	101:101	102:200	201:201	202:300
MT6	2:100	101:101	102:200	201:201	202:300

3.4 DLGを考慮したダイナミックスケジューリング

階層型粗粒度タスク並列処理により粗粒度並列処理される各階層において，多量のデータ転送を必要とするMT間で，PE上の分散共有メモリ，分散キャッシュを介したデータ授受を実現するためには，それらのMT集合を同一PC（あるいはPE）に割り当てなければならない．

今回の評価では，MTの割り当てにはパーシャルスタティック割当てを伴うダイナミックスケジューリングを用いた[6]．

3.4.1 データローカライゼーショングループ（DLG）指定

データローカライゼーショングループ（DLG）とは，LRにアクセスするMT集合であり，後述のようにダイナミックスケジューラによって同一PCに割り当てられることが保証される[7]．

またこのダイナミックスケジューリング時は実行時オーバーヘッドを考慮しAdjacent-PartもDLGに含める．図1の例においてダイナミックスケジューリング時のDLGは図1(d)の太波線により囲まれた領域となる．

なお，図1(d)の太実線でかこまれた領域はスタティックスケジューリング時のDLGであり，Adjacent-Partは並列性とデータローカライゼーション効果の双方を考慮したデータ転送ゲイン/CPスケジューリングが適用される[15)16)]．

3.4.2 パーシャルスタティックタスク割り当てを伴うスケジューリング

DLG内の複数のMT間でデータローカライゼーションを実現するためには，前述のようにこれらのMTが実行時に同一のPCにダイナミックスケジューリングされなければならない．このためコンパイラは，あるDLGに属するいずれかのMTがPCに割り当てたられた時点で，そのDLGに属するその他全てのMTに対して，必ず最初に割り当てられたMTと同一のPCに割り当てを行うためのスケジューリングルーチンを生成し，並列化コード中に埋め込む．

4. 性能評価

本章では，データローカライゼーションを伴う階層型粗粒度タスク並列処理を，SMPワークステーションSun Ultra80上で性能評価した結果について述べる．

4.1 評価環境

Sun Ultra80は，450MHzのUltraSPARC IIプロセッサを4台持つマルチプロセッサシステムであり，キャッシュを含めたハードウェア仕様を表2に示す．

表2 Sun Ultra 80仕様

CPU	UltraSPARC II x 4
CPU Clock	450 MHz
L1(instruction)	16KB(2-way set associative)
L1(data)	16KB(direct mapped)
L2(unified)	4MB(direct mapped)
Main Memory	1GB

使用したネイティブコンパイラはSun Forte version 6 update 1のFortranコンパイラ f95であり，使用したコンパイルオプションは表3のとおりである．OSCAR自動並列化の欄はOSCAR Fortranコンパイラを用いて本データローカライゼーション結果を含め粗粒度タスク並列化したOpenMPコードとして出力し，それをForteコンパイラを用いてオブジェクトコードを生成する際に使用したオプションである．

表3 Forte使用オプション

Forte逐次実行	-fast
Forte自動並列化	-fast -parallel -stackvar -reduction
OSCAR自動並列化	-fast -mp=openmp -explicitpar -stackvar

4.2 TOMCATVプログラムによる評価

表4 TOMCATV評価結果

	1PE	2PE	3PE	4PE
Forte実行時間 [s]	120	95	92	93
Oscar実行時間 [s]	125	75	47	31
ForteL2ヒット率	0.91	0.90	0.90	0.89
OscarL2ヒット率	0.92	0.94	0.96	0.98

本節では，SPECfp95ベンチマークのTOMCATVプログラムを用いて，提案するデータローカライゼーション手法の性能評価を行う．TOMCATVは，Vectorized Mesh生成プログラムであり，初期化部分と収束計算ループから構成されている．データサイズはN=513である．

本評価では，Forte用いて自動並列化した場合のUltra80上での実行時間と，提案手法を実装したOSCAR Fortranコンパイラを用いて並列化した結果をOpenMPで出力し，それをForteでコンパイルした場合のUltra80上での実行時間を比較する．

表4にForteによる各プロセッサ数の時の実行時間，OSCARコンパイラを用いた場合の各プロセッサ数の時の実行時間，Forte及びOSCARにおけるL2キャッシュヒット率を示す．

表からわかるように本データローカライゼーション手法を伴う階層型マクロデータフロー処理では，L2キャッシュの有効利用により，4PEでForte 1PEの3.87倍の速度向上率を得ることができ，同じ4PEを使用した場合，Forteと比較して3.00倍の速度向上が得られることが確かめられた．

4.3 SWIMプログラムによる評価

次に，SPECfp95ベンチマークのSWIMプログラム

表 5　SWIM 評価結果

	1PE	2PE	3PE	4PE
Forte 実行時間	104	67	63	61
Oscar 実行時間	105	50	23	14
ForteL2 ヒット率	0.94	0.95	0.95	0.95
OscarL2 ヒット率	0.96	0.97	0.98	0.99

を用いて，提案するデータローカライゼーション手法の性能評価を行う．SWIM は，Shallow Water 方程式の求解プログラムであり，初期化部分と複数のサブルーチンコールを含むメインループから構成されている．データサイズは N1=N2=513 である．

SWIM を Ultra80 上で実行した結果を表 5に示す．表から，Forte 1PE の時 104 秒であった実行時間が，4PE では Forte が 61 秒に，また本データローカライゼーションを伴う階層的粗粒度タスク並列化を実現する OSCAR コンパイラが 14 秒と Forte 1PE の 7.43 倍の速度向上を得ることができることが確かめられた．またこの性能差は，L2 ヒット率が Forte が 95%なのに対し，OSCAR が 99%であることから本手法によるキャッシュヒット率向上が大きな原因となっていることが確かめられた．

5. むすび

本稿では，階層型粗粒度タスク並列処理における，タスク間並列性とデータローカリティの両方を考慮しデータローカライゼーションを実現するマクロタスクを決定する手法を提案した．

本手法を OSCAR マルチグレイン並列化コンパイラ上に実装し，SMP ワークステーション Ultra80 上で性能評価を行った．これにより本データローカライゼーション手法を伴う階層型マクロデータフロー処理では，Sun Forte コンパイラのループ自動並列化と比べ，SPEC95fp TOMCATV で実行時間で 4 プロセッサ使用時で 3.00 倍，SWIM で 4.36 倍の速度向上が得られ，提案手法の有効性が確認された．

なお本研究の一部は，経済産業省 NEDO ミレニアムプロジェクト IT21 アドバンスト並列化コンパイラプロジェクトにより行なわれた．

参考文献

1) A. Agarwal, D. A. Kranz, and V. Natarajan. Automatic partitioning of parallel loops and data arrays for distributed shared-memory multiprocessors. *IEEE Trans. on Parallel and Distributed System*, Vol. 6, No. 9, pp. 943–962, 1995.

2) S. Amarasinghe, J. Anderson, M. Lam, and C. Tseng. The suif compiler for scalable parallel machines. *Proc. of the 7th SIAM conference on parallel processing for scientific computing*, 1995.

3) W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, J. Lee, and D. Padua. Advanced program restructuring for high performance computers with polaris. Technical Report 1473, CSRD, University of Illinois, Urbana-Champaign, 1996.

4) High Performance Fortran Forum. *High Performance Fortran Language Specification Version 2.0*. Jun. 1997.

5) M. Gupta and P. Banerjee. Demonstration of automatic data partitioning techiniques for parallelizing compilers on multicomputers. *IEEE Trans.on Parallel and Ditributed System*, Vol. 3, No. 2, pp. 179–193, 1992.

6) Kazuhisa Ishizaka, Motoki Obata, and Hironori Kasahara. Coarse grain task parallel processing with cache optimization on shared memory multiprocessor. *Proc. of 14th International Workshop on Languages and Compilers for Parallel Computing (LCPC2001)*, Aug. 2001.

7) H. Kasahara and A. Yoshida. A data-localization compilation scheme using partial static task assignment for fortran coarse grain parallel processing. *Journal Of Parallel Computing Special Issue On Languages And Compilers For Parallel Computers*, May 1998.

8) A. W. Lim, G. I. Cheong, and M. S. Lam. An affine partitioning algorithm to maximize parallelism and minimize communication. *International Conference on Supercomputing*, pp. 223–237, 1999.

9) Xavier Martorell, Eduard Ayguade, Nacho Navarro, Julita Corbalan, Marc Gozalez, and Jesus Labarta. Thread fork/join techniques for multi-level parllelism exploitation in numa multiprocessors. *ICS'99 Rhodes Greece*, 1999.

10) P. Tu and D. Padua. Automatic array privatization. *6th Annual Workshop on Languages and Compilers for Parallel Computing*, 1993.

11) S. Vajracharya, S. Karmesin, P. Beckman, J. Crotinger, A. D. Malony, S. Shende, R. R. Oldehoeft, and S. Smith. Smarts: Exploiting temporal locality and parallelism through vertical execution. *International Conference on Supercomputing*, pp. 302–310, 1999.

12) M. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley, 1996.

13) 岡本雅巳, 合田憲人, 宮沢稔, 本多弘樹, 笠原博徳. Oscar マルチグレインコンパイラにおける階層型マクロデータフロー処理手法. 情報処理学会論文誌, Vol. 35, No. 4, pp. 513–521, Apr. 1994.

14) 吉田明正, 前田誠司, 尾形航, 笠原博徳. 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. 情報処理学会論文誌, Vol. 35, No. 9, 1994.

15) 吉田明正, 八木哲志, 笠原博徳. SMP 上でのデータ依存マクロタスクグラフのデータローカライゼーション手法. 情報処理学会研究報告 ARC, No. 141, pp. 29–34, Jan. 2001.

16) 中野啓史, 石坂一久, 小幡元樹, 木村啓二, 笠原博徳. キャッシュ最適化を考慮したマルチプロセッサシステム上での粗粒度タスクスタティックスケジューリング手法. 情報処理学会研究報告 ARC2001-140-12, Aug. 2001.