計算機アーキテクチャ 141－6
（2001．1．26）

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

吉田 明正† 八木 哲志‡ 笠原 博徳§‡

†東邦大学 理学部 情報科学科
‡早稲田大学 理工学部 電気電子情報工学科
§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，データ依存マクロタスクグラフを対象とした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，粗粒度タスク並列処理の対象となる各階層のデータ依存粗粒度タスク集合に対して，ループ整合分割法を用いて処理とデータを分割し，分割後の粗粒度タスク集合において多量のデータ転送を必要とするタスクをデータ転送ゲイン/CPスケジューリング法により同一プロセッサに割り当て，分散キャッシュあるいはローカルメモリを効果的に利用することによりデータ転送オーバヘッドを軽減する．マルチプロセッサシステムSGI Origin2000上で行った性能評価の結果，本データローカライゼーションを用いた階層型タスク粗粒度並列処理は，処理時間を顕著に短縮できることが確認された．

A Data-Localization Scheme for Macrotask-Graph with Data Dependencies on SMP

Akimasa Yoshida† Satoshi Yagi‡ Hironori Kasahara§‡

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

Abstract

This paper proposes a data-localization scheme for macrotask-graph with data dependence edges in hierarchical coarse-grain parallel processing. In this scheme, first, multiple coarse-grain tasks having data dependencies are decomposed by Loop-aligned-decomposition method in each macrotask-graph layer. Next, the compiler assigns macrotasks with strong data dependence to the same processors by Data-transfer-gain/CP scheduling method, so that data transfer overhead is reduced by using caches or local memories. Finally, this paper describes the performance evaluation on a multi-processor system SGI Origin2000. The evaluation shows that hierarchical coarse-grain parallel processing with data-localization can reduce execution time remarkably.

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[1]が用いられている．例えば，イリノイ大学のPolaris[2]やスタンフォード大学のSUIF[3]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並列性しか利用することができないという問題があった．この問題を解決するために，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と考えられる[4][5]．

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（SMP）あるいは分散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，分散キャッシュあるいはローカルメモリ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[6]，ループ内の作業配列をローカル化するArray Privatization法[7]，自動データ分散法[8][9][10]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を利用することを前提にしている．

一方，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 [12][13] 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，データ依存エッジで接続された任意形状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通りとする．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，データ依存マクロタスクグラフを対象としたデータローカライゼーション手法，特にデータ依存マクロタスクグラフを対象としたループ整合分割法と，データ転送ゲイン/CP スケジューリング法について述べる．第 4 章では，SGI Origin2000 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，共有メモリに加え分散キャッシュあるいはローカルメモリを持ったマルチプロセッサシステム，あるいは分散共有メモリを持った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 [4] 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 [4]．

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

マクロタスク生成後，マクロタスク間あるいはサブ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 [4] 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，各マクロタスクの最早実行可能条件を解析する．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最大限に表している．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）[4] 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 [4] があるが，本論文ではデータ依存エッジで接続されたマクロタスクグラフを対象とするため，スタティックスケジューリング方式を採用する．スタティックスケジューリング方式では，コンパイル時にスケジューリングを行い，並列処理用実行コード（本論文では OpenMP で記述された並列コード）を生成する．

3 データ依存 MTG のためのデータローカライゼーション

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，分散キャッシュあるいはローカルメモリ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，3.1節で述べるデータ依存 MTG を対象としたループ整合分割法と 3.2節のデータ転送ゲイン/CP スケジューリング法により実現される．

3.1 データ依存 MTG のためのループ整合分割法

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．開発されているループ整合分割法 [12][13] 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，本稿で提案するループ整合分割法では，適用範囲をデータ依存エッジで接続された任意形状のマクロタスクグラフに拡大し，後述のデータ転送ゲイン/CP スケジューリング法と組み合わせることにより，粗粒度タスク並列性とデータローカリティの両方を有効利用することを可能とする．

3.1.1 ループ整合分割の概念

提案するループ整合分割法では，まず，マクロタスクグラフ（MTG）から抽出されたループ集合，即ちターゲットループグループ（**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，タスク間で最も多いデータ授受を必要とする 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．

3.1.2 データ依存 MTG を対象とした TLG 生成

本節では，データ依存エッジで接続された MTG から，ループ整合分割の単位となるターゲットループグループ（TLG）を生成する方法について述べる．

(1) MTG の CP 上における TLG 生成

MTG の CP（Critical Path，但し各 MT の処理時間には共有メモリアクセス時間を含めて計算する）上のループ集合において，データ入力が最大のループを標準ループとし，$TLG_{current}$ の構成要素とする．次に，CP 上において標準ループに隣接（ループ間に BPA があってもよい）しているループを $TLG_{current}$ へ追加する．但し，$TLG_{current}$ 内ループとの間で整合ができない場合には，$TLG_{current}$ に追加しない．

ここで，整合条件とは，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 次式で表されており，かつ，ループ間にデータ依存を導く各配列

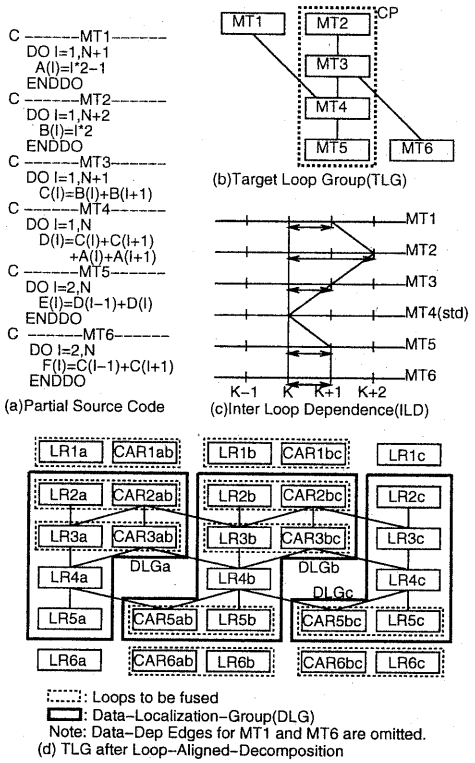

図 1: ループ整合分割の例．

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 [12]．

(2) TLG からデータ依存エッジで接続された MT の TLG 追加

$TLG_{current}$ からデータ依存エッジで直接接続された先行 MT_i を $TLG_{current}$ に追加する．（図 1(b) の MT1，Merge 型）

次に，$TLG_{current}$ からデータ依存エッジで接続された後続 MT_i を $TLG_{current}$ に追加する．（図 1(b) の MT6，Diverge 型）

但し，MT_i と $TLG_{current}$ 間データ転送量が，MT_i と他 MT 間データ転送量に比べて小さい場合，あるいは，MT_i と $TLG_{current}$ 内ループとの間で整合できない場合には，MT_i は $TLG_{current}$ に追加しない．

(3) 現在の MTG から $TLG_{current}$ を取り除いた後，(1) に戻り，別の TLG を生成する．

以上の方法で生成された各 TLG は，MTG の CP 上のループ（RB），CP に直接先行接続されたループ（RB），CP に後続接続されたループ（RB）から構成される．また，TLG を構成するループの種類は，Doall ループ，リダクションループ，ループ制御変数がデータ領域を支配するシーケンシャルループのいずれかである．

3.1.3 TLG 内ループ間データ依存解析

本手法では，前節で生成された各TLG に対して，TLG 内 CP 上でデータ入力が最大の MT を標準ループとし，この標準ループから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，**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

3.1.2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 [12][13] 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 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）

$ILD_{MT_i}^{(MT_{std},K)} = [K + l_x - U_i^x : K + u_x - L_i^x]$.
（**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
$ILD_{MT_i}^{(MT_{std},K)} = [K + u_x - U_i^x : K + l_x - L_i^x]$.
（**PreToPost 法**）

(ii)$std < x \leq end$ の場合
前述の **PostToPost 法**により ILD を求める．

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

3.1.4 TLG 内ループの整合分割

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ローカル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 3.1.1節で述べた通り，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るため，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．

3.2 DLGを考慮したデータ転送ゲイン/CPスケジューリング法

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，分散キャッシュあるいはローカルメモリ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．

本節では，データローカライゼーショングループ（**DLG**）を考慮したデータ転送ゲイン/CPスケジューリング法を提案する．本スケジューリング法では，ループ整合分割により生成されたMT集合に対して，粗粒度タスク並列性を最大限に利用し，かつデータ転送量の多いMT集合を同一のPC（またはPE）にスケジューリングすることが可能となる．

本スケジューリング法におけるプライオリティは，(1)データローカライゼーショングループ（DLG），(2)データ転送ゲイン（DT-Gain），(3)CP（Critical Path）の順となる．以下に詳細を示す．

3.2.1 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，スケジューラが，同一PCに割り当てることを保証するMT集合のことである[13]．DLGはコンパイラにより決定されるが，仮に，粗粒度タスク並列性のある複数MTが同一DLGに指定されると，それらのMTは同一PCに割り当てられるため，DLG内の粗粒度タスク間並列性が失われてしまう．

そこで，本スケジューリング法では，図1(d)の太実線で囲まれた領域（例，$LR_2^a(CAR_2^{ab}$と融合$)\sim LR_5^a$）のように，分割前のTLGにおいてCP上に並んでいた（粗粒度並列性のない）MT集合に対応する，分割後の同一データ領域使用ループをDLGとして指定する．その結果，$LR_1^a(CAR_1^{ab}$と融合）とLR_6^a等は，DLGには含まれず，後述するデータ転送ゲイン/CPスケジューリングが適用される．

3.2.2 データ転送ゲイン/CPスケジューリング

本論文で提案するデータ転送ゲイン（DT-Gain）/CPスケジューリング法では，データ転送ゲイン（キャッシュあるいはローカルメモリに保持されているデータの利用により，データ転送が削減される量）が最も大きいタスクとPE（あるいはPC）の組合せを選択し割り当てを行う．但し，データ転送ゲインが同一のタスクが複数存在する場合には，CP長の大きなタスクを割り当てる．

ここで，図2(a)のタスクグラフを用いて，従来のデータ転送を考慮したスケジューリングアルゴリズムDT/CP法[14]と提案するDT-Gain/CP法を比較する．このタスクグラフにおいて，各タスクの処理時間は100[u.t.]とし，タスク間データ転送時間はデータ依存エッジの横に示す値とする．図2(b)のDT/CP法と図2(c)のDT-Gain/CP法のどちらの場合にも，まず，タスクT1，T2，T3が，PE1，PE2，PE3にそれぞれ割り当てられる．次に，T1，T2，T3の終了時（経過時刻100[u.t.]）に，タスクT4，T5がレディタスクとなり，アイドルのPE1，PE2，PE3への割当てを決定する．その際のデータ転送およびデータ転送ゲインは，表2の通りである．

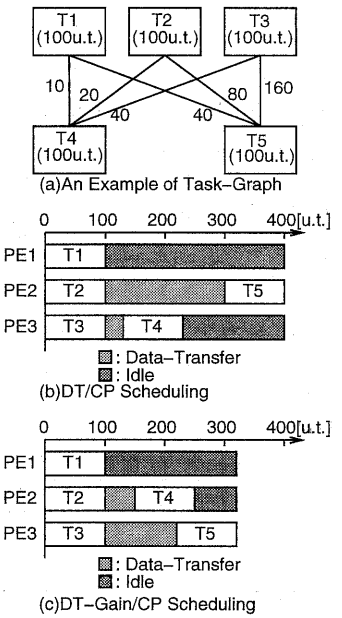

図2: データ転送ゲイン/CPスケジューリングの例．

表2: データ転送量とデータ転送ゲイン．

	DT			DT-Gain		
	PE1	PE2	PE3	PE1	PE2	PE3
T4	60	50	[30]	10	[20]	40
T5	240	[200]	120	40	80	[160]

この時，DT/CP法では，まず，データ転送が最小の割当て，即ちタスクT4のPE3への割当てが行われ，次に，タスクT5のPE2への割当てが行われる．一方，提案するDT-Gain/CP法では，データ転送ゲイン（データ転送の削減量）が最大の割当て，即ちタスクT5のPE3への割当てが行われ，次に，タスクT4のPE2への割当てが行われる．結果として，図2(b)と図2(c)のガントチャートに示すように，DT/CP法では割当て時点で最小のデータ転送のタスクを選んでいるのに対して，DT-Gain/CP法ではデータ転送削減量の大きなタスクを効果的に割当てしており，データ転送が大幅に短縮されることが分かる．

このデータ転送ゲイン/CPスケジューリングは，前述の図1(d)では，$LR_1^a(CAR_1^{ab}$と融合$)\sim LR_1^c$と

LR_6^a～LR_6^c(CAR_6^{bc}と融合) に対して適用される．その際，例えばLR_1^aやLR_6^aは，LR_2^a～LR_5^aのDLGとの間のデータ転送量が多いため，LR_2^a～LR_5^aのDLGと同じPCにスケジューリングされる可能性が大きい．なお，データ転送ゲインを計算する際には，DLG内の未実行MTについてもPCに割当てされているものとし，DLG内の全MTに対するデータ転送を計算する．

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，Origin2000上で性能評価した結果について述べる．

4.1 Origin2000のアーキテクチャ

SGI Origin2000は，195MHz MIPS R10000プロセッサ2台と分散共有メモリ（256MB）を1ノードとし，各ノードがインターコネクションネットワーク（ハイパーキューブ）により接続されたマルチプロセッサシステムである．各PEには，オンチップの命令キャッシュとデータキャッシュ（各32KB，32Bラインサイズ），2次キャッシュ（4MB，128Bラインサイズ，2ウェイセットアソシアティブ）をもっている．Origin2000では，First-Touch Page Allocation Policyにより，16KBのページ単位でページ割当てが行われる．

表 3: Origin2000上での実行時間．

実行方式	PE数	実行時間 [s]
シーケンシャル処理	1	358
SGIの自動並列化	4	181
提案手法	4	108
SGIの自動並列化	8	132
提案手法	8	74

4.2 Tomcatvプログラムによる評価

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．このプログラムは，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．

このプログラムをOrigin 2000上で実行した結果を表3に示す．本評価では，SGIの自動並列化コンパイラを用いて並列化した場合と，提案する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の結果をOpenMPコードとして出力しSGIコンパイラでコンパイルして実行した場合を比較する．

提案した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，キャッシュの有効利用により，4PEで1PEの3.31倍，8PEで4.84倍の速度向上が得られた．また，SGIの自動並列化コンパイラによる実行と比べると，実行時間が4PEで40%，8PEで44%短縮されており，提案手法の有効性が確かめられた．

5 おわりに

本稿では，階層型粗粒度タスク並列処理におけるデータ依存マクロタスクグラフを対象としたデータローカライゼーション手法を提案した．本手法を実現するために，データ依存MTGを対象としたループ整合分割法とデータ転送ゲイン/CPスケジューリング法を新たに開発した．

Origin2000上で行った性能評価により，提案した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，SGIの自動並列化コンパイラによる実行と比べて，実行時間が最大44%短縮されており，提案手法の有効性が確認された．

本研究の一部は，NEDOアドバンスト並列化コンパイラプロジェクト，及び，科学研究費補助金（奨励研究(A)12780243）により行われた．

参考文献

[1] Wolfe,M.: High Performance Compilers for Parallel Computing, *Addison-Wesley Pub.* (1996).

[2] Blume,W., Doallo,R., Eigenmann,R., et al.: Advanced Program Restructuring for High Performance Computers with Polaris, *Technical Report 1473, CSRD, University of Illinois, Urbana-Champaign* (1996).

[3] Amarasinghe,S., Anderson,J., Lam,M. and Tseng, C.-W.: The SUIF Compiler for Scalable Parallel Machines, *Proc. of the seventh SIAM conference on parallel processing for scientific computing* (1995).

[4] 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 (1994).

[5] Martorell,X., Ayguade,E., Navarro,N., et al.: Thread Fork/Join Techniques for Multi-level Parallelism Exploitation in NUMA Multi-processors, *Proceedings of International Conference on Supercomputing*, 1999.

[6] High Performance Fortran Forum: *High Performance Fortran Language Specification Version 2.0* (1997).

[7] Tu,P. and Padua,D.: Automatic Array Privatization, *6th Annual Workshop on Languages and Compilers for Parallel Computing* (1993).

[8] Gupta,M. and Banerjee,P.: Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers, *IEEE Trans. on Parallel and Distributed System*, Vol. 3, No. 2 (1992).

[9] Agarwal,A., Kranz,D.A. and Natarajan,V.: 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).

[10] Lim,A.W., Cheong,G.I. and Lam,M.S.: An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication, *International Conference on Supercomputing*, pp. 228–237 (1999).

[11] Vajracharya,S., Karmesin,S., Beckman,P., et al.: SMARTS: Exploiting Temporal Locality and Parallelism through Vertical Execution, *International Conference on Supercomputing*, pp. 302–310 (1999).

[12] 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).

[13] Kasahara, H., Yoshida, A.: A Data-Localization Compilation Scheme Using Partial-Static Task Assignment, *Journal of Parallel Computing*, Vol. 24, No. 3, pp. 579–596 (1998).

[14] 藤原和典, 白鳥健介, 鈴木真, 笠原博徳: データプレロードおよびポストストアを考慮したマルチプロセッサスケジューリングアルゴリズム, 電子情報通信学会論文誌 D-I, Vol. J75-D-I, No. 8, pp. 495–503 (1992).