社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－HPC－107（22）
2006／8／1

PCクラスタ上での階層統合型粗粒度タスク並列処理

吉　田　明　正†

本稿では，階層統合型実行制御を用いた粗粒度タスク並列処理をPCクラスタ環境でMPIにより実装する手法を提案する．階層統合型粗粒度タスク並列処理では，階層的に定義されたマクロタスクを統一制御し，全プロセッサ上で異なる階層にまたがったマクロタスク間並列性を利用することが可能である．本手法では，集中型ダイナミックスケジューリング方式を採用しており，PCクラスタの1PEをスケジューリング処理用，他PEをマクロタスク実行用としている．また，データ転送オーバヘッドを最小化するために，データローカライゼーション技術を導入する．本稿ではPCクラスタ上で行った性能評価についても述べられており，提案手法の有効性が確認されている．

Layer-Unified Coarse Grain Task Parallel Procesing on a PC cluster

Akimasa Yoshida†

This paper proposes an implementation scheme of the layer-unified coarse grain task parallel processing on a PC cluster by using MPI. In the coarse grain task parallel processing with the layer-unified control technique, the macrotasks defined hierarchically are controlled by a scheduler and it is possible to use parallelism among coarse grain tasks across several layers. In this implementation, one PE inside a PC-cluster is used as the dynamic scheduler and other PEs are used to execute the macrotasks. This paper also proposes the data localization techniques to minimize the data transfer overheads. In the performance evaluations on the PC-cluster, the effectiveness of the proposed scheme has been confirmed.

1. はじめに

マルチプロセッサシステム上での並列処理手法としては，従来よりループレベルの並列化技術[1]が用いられており，さまざまな形状のループが並列処理可能になっている．しかしながら，今後さらなる性能向上を目指すためには，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[2)~5)]を利用する粗粒度タスク並列処理が有効と考えられる．

粗粒度タスク並列処理[2),3)]では，粗粒度タスク間の並列性を並列化コンパイラが自動抽出して階層型マクロタスクグラフを生成し，各階層の粗粒度タスクを，グルーピングしたプロセッサに階層的に割り当て並列処理を行っていた．この場合，対象プログラム中の各階層の粗粒度タスクは，その階層を処理すべきプロセッサグループに割り当てられて実行されるため，十分な台数のプロセッサを確保できない場合には，対象プログラムに内在する全階層の粗粒度タスク間並列性を利用できない可能性がある．

それゆえ，粗粒度タスク並列処理で用いられている階層型マクロタスクグラフ[2),3)]を利用しつつ，対象プログラム中の異なる階層の粗粒度タスクを統一的に取り扱い，異なる階層にまたがった粗粒度タスク間並列性を最大限に利用する階層統合型実行制御手法[6]が提案されている．本稿では，その階層統合型実行制御を伴う粗粒度タスク並列処理を，普及しているPCクラスタにおいて実現する方法を提案する．

本論文の構成は以下の通りとする．第2章では階層統合型実行制御を伴う粗粒度タスク並列処理の概要を述べる．第3章では，階層統合型粗粒度タスク並列処理を実現するためのMPI実装方法について述べる．第4章では，3階層プログラムに対して，階層統合型粗粒度タスク並列処理を実現する並列プログラムをMPIにより実装し，PCクラスタで行った性能評価について述べる．第5章でまとめを述べる．

2. 階層統合型実行制御を伴う粗粒度タスク並列処理

本章では，階層統合型粗粒度タスク並列処理の概要を述べる．

2.1 対象アーキテクチャ

本稿の対象とする階層統合型粗粒度タスク並列処理[6]は，現在までに，共有メモリ型マルチプロセッサシステム（SMP）において，PthreadやOpenMPを

† 東邦大学理学部情報科学科
Department of Information Science, Toho University

用いて実装されている．この場合，プロセッサはグルーピングすることなしに，対象プログラムの全階層の並列性を最大限に利用することが可能である．また，本稿で提案する MPI 実装手法を用いることにより，PC クラスタ上においても実装が可能である．

2.2 粗粒度タスク並列処理のための並列性抽出

粗粒度タスク並列処理[2),3)] は，階層的にループやサブルーチン等の粗粒度タスク間の並列性を抽出し，粗粒度タスク（マクロタスク）をプロセッサあるいはプロセッサグループに割り当て並列処理する方式である．

粗粒度タスク並列処理による実行では，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（MT）に分割する．マクロタスクは，擬似代入文ブロック（基本ブロック），繰り返しブロック（ループ），あるいは，サブルーチンブロックの 3 種類から構成される[2),3)]．次に，第 1 階層マクロタスク内部に複数のサブマクロタスク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．同様に，第 L 階層マクロタスク内部において，第 $(L+1)$ 階層マクロタスクを定義する．

ここで，繰り返しブロックが Doall ループ（あるいはリダクションループ）である場合には，粗粒度並列性を向上させるために，複数の部分 Doall ループに分割し，それらを別々のマクロタスクとして定義する[7)]．この際，従来の階層型実行制御の場合には，当該階層のプロセッサグループ数あるいはその整数倍に分割し，階層統合型実行制御の場合には，全 PE 数あるいはその整数倍に分割することにより，Doall ループの並列性を有効利用できる．

マクロタスク生成後，各階層のマクロタスク間の制御フローとデータ依存を解析し，階層型マクロフローグラフ[2),3)] を生成する．次に，制御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[2),3)] を解析する．最早実行可能条件は，制御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おり，マクロタスクの実行制御に用いられる．例えば，**図 1** の MT5 の最早実行可能条件は，**表 1** に示すように $1 \wedge 2 \wedge 3 \wedge 4$ と求められ，MT5 は MT1〜MT4 の実行終了後に実行可能となることを表している．

各階層の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）[2),3)] によって表すことが可能であり，例えば，表 1 の最早実行可能条件は図 1 の MTG により表現される．

2.3 階層統合型実行制御によるマクロタスクスケジューリング

階層統合型実行制御を伴う粗粒度タスク並列処理では，各階層ごとの最早実行可能条件を解析した後，第 3 章で述べる方法により階層開始マクロタスクを導入し，各種条件を変換する．

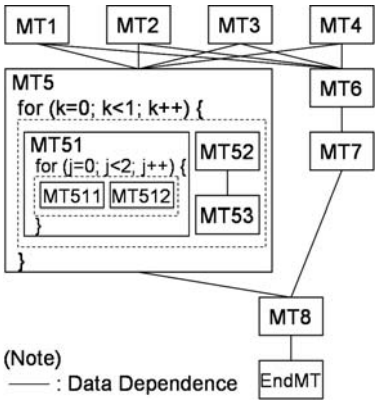

図 1 階層型マクロタスクグラフ（MTG）．

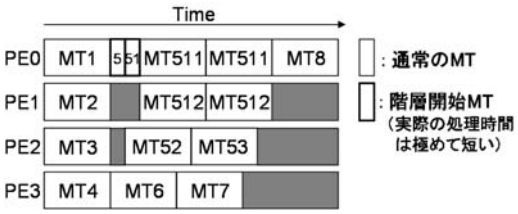

図 2 階層統合型粗粒度タスク並列処理の実行イメージ．

マクロタスクのスケジューリングに関しては，マクロタスク間の条件分岐等の実行時不確定性に対処するために，マクロタスクを実行時にプロセッサに割り当てるダイナミックスケジューリング方式を採用している．

階層統合型実行制御では，全ての階層のマクロタスクが統一的に取り扱われ，それぞれの PE（グルーピングなし）に割り当てられ実行される．例えば，図 1 の階層型マクロタスクグラフの場合，**図 2** に示すように，MT1〜MT8 の第 1 階層マクロタスク，MT5 内部の MT51〜MT53 の第 2 階層マクロタスク，MT51 内部の MT511〜MT512 の第 3 階層マクロタスクを統一的に取り扱い，それらを各プロセッサに割り当てて実行することが可能となる．

ここで，MT5 はその内部の第 2 階層の開始処理のみを行う階層開始マクロタスクとして扱われ，MT51 もその内部の第 3 階層の開始処理のみを行う階層開始マクロタスクとして扱われる．本手法では，第 1 階層から第 3 階層までの並列性（例えば MT511，MT512，MT52，MT6 の間の並列性）が同時に利用されており，実行時間が大幅に短縮されることが分かる．

3. 階層統合型粗粒度タスク並列処理の MPI 実装方法

本章では，階層統合型実行制御を伴う粗粒度タスク並列処理を，PC クラスタ上で MPI により実装する

表 1 最早実行可能条件と終了ステートの階層統合型変換．

MT 番号	最早実行可能条件		終了ステート	
	階層型	階層統合型	階層型	階層統合型
MT1	true		1	
MT2	true		2	
MT3	true		3	
MT4	true		4	
MT5 †	$1 \wedge 2 \wedge 3 \wedge 4$		5	**5S**
MT6	$1 \wedge 2 \wedge 3 \wedge 4$		6	
MT7	6		7	
MT8	$5 \wedge 7$		8	
EndMT9	8		9	
MT51 ††	true	**5S**	51	**51S**
MT52	true	**5S**	52	
MT53	52		53	
CtrlMT54	$51 \wedge 53$		54	
RepMT55	54_{55}		55	
ExitMT56	54_{56}		56	**5**
MT511	true	**51S**	511	
MT512	true	**51S**	512	
CtrlMT513	$511 \wedge 512$		513	
RepMT514	513_{514}		514	
ExitMT515	513_{515}		515	**51**

(注) † 階層統合型の場合，第 2 階層用の階層開始 MT.
†† 階層統合型の場合，第 3 階層用の階層開始 MT.

方法について述べる．

3.1 階層開始マクロタスク

階層統合型粗粒度タスク並列処理では，全階層のマクロタスクを統一的に取り扱うため，第 L 階層マクロタスクを内部に持つ上位の第 $(L-1)$ 階層マクロタスクを，第 L 階層用の階層開始マクロタスクとして取り扱う．この階層開始マクロタスクは，内部の第 L 階層マクロタスクの実行を開始するために使用される．この階層開始マクロタスクの導入により，当該階層のマクロタスクの実行が可能になったことが保証され，全階層のマクロタスクを同時に取り扱うことが可能となる．

例えば，図 1 の MT5 の場合，内部に第 2 階層 MT（MT51，MT52，MT53）を含んでおり，MT5 は第 2 階層用の階層開始マクロタスクとして扱われる．同様に，図 1 の MT51 の場合，内部に第 3 階層 MT（MT511，MT512）を含んでおり，MT51 は第 3 階層用の階層開始マクロタスクとして扱われる．

3.2 最早実行可能条件の階層統合型変換

階層開始マクロタスクの導入により，従来の階層ごとに求めた最早実行可能条件[2),3)] を階層統合型実行制御用に変換する．具体的には，第 L 階層マクロタスクの最早実行可能条件が「true」（即ちその階層が実行可能になればすぐに実行可能）である場合，その条件を「第 L 階層用の階層開始マクロタスク MT_i の終了」に置き換える．ここで，階層開始マクロタスクとしての MT_i の終了ステートは iS と表記し，本来の MT_i の内部全体の終了ステート[2),3)] は従来通り i と表記する．

例えば，図 1 の各 MT の最早実行可能条件[2),3)] は表 1 に示す通りである．第 2 階層の MT51 と MT52 の最早実行可能条件は，従来の階層型実行制御用では「true」であるため，階層統合型実行制御用では「その階層の階層開始マクロタスク MT5 の終了（5S）」と置き換える．同様に，第 3 階層の MT511 と MT512 の最早実行可能条件は，階層型実行制御用では「true」であるため，階層統合型実行制御用では「その階層の階層開始マクロタスク MT51 の終了（51S）」と置き換える．

なお，最早実行可能条件において，i は MT_i の終了，$(i)_j$ は MT_i から MT_j への分岐，i_j は MT_i から MT_j への分岐と MT_i の終了を表している．また，EndMT（終了処理），CtrlMT（当該階層の繰り返し判定処理），RepMT（当該階層の繰り返し更新処理），ExitMT（当該階層の終了処理）は制御用マクロタスクである．

なお，本方式では，階層開始マクロタスクとしての MT_i の実行終了を表す終了ステート iS は，階層開始マクロタスク自身に発行させ，MT_i 内部の第 L 階層の実行終了を表す終了ステート i は，第 L 階層の ExitMT に発行させている．

3.3 階層統合型レディマクロタスクキューとローカルレディマクロタスクキュー

粗粒度タスク並列処理においてダイナミックスケジューリングを適用する場合，各マクロタスクはその最早実行可能条件[2),3)] が満たされた後，レディマクロタスクキューに投入され，プライオリティの高い（Critical-Path（CP）長の大きい）マクロタスクから順にレディマクロタスクキューから取り出されてプロセッサに割り当てられる．

階層統合型実行制御を実現する場合，異なる階層のマクロタスクを統一的に扱うため，全階層のマクロタスクを対象とした階層統合型レディマクロタスクキューを導入する．即ち，各階層のマクロタスクは，その最早実行可能条件が満たされた後，階層統合型レディマクロタスクキューに投入される．

但し，3.6 節で述べるデータローカライゼーションを適用する場合には，新たに各 PE 用のローカルレディマクロタスクキューを導入する．PE 用のローカルレディマクロタスクキューは，その PE で実行することが確定している MT 集合であり，パーシャルスタティックタスク割当てを伴うダイナミックスケジューラにより投入される．実行時には，PE 用のローカルレディマクロタスクキューのマクロタスクを先に実行し，その後，階層統合型レディマクロタスクキューのマクロタスクが実行される．

3.4 ダイナミックスケジューリング方式による並列処理コードの MPI 実装

階層統合型実行制御を伴うダイナミックスケジュー

リングは，SMP でのマルチスレッド実装の場合，集中型ダイナミックスケジューリング，あるいは，分散型ダイナミックスケジューリングにより実現できる[6]．本稿では，PC クラスタを対象とした集中型ダイナミックスケジューリング方式を対象とする．この場合，PC クラスタ上の 1PE がスケジューリング処理を専属で行い，それ以外の PE がマクロタスク実行を行う．

3.4.1 スケジューリング処理用 PE コード

スケジューリング処理用 PE におけるスケジューリングの手順を以下に示す．なお，初期状態において，各 PE 用のローカルレディマクロタスクキューは空であり，階層統合型レディマクロタスクキューには，初期状態でレディとなるマクロタスクを投入しておく．

(i-a) PE_i 用ローカルレディマクロタスクキューに MT_x があり，PE_i がアイドルであれば，PE_i にマクロタスク番号 x を送信（MPI_Send）する．
(i-b) MT_x で参照される共有データ（データローカライゼーションの適用されないもの）を，スケジューリング処理用 PE から，MT_x を実行する PE_i に送信（MPI_Send）する．
(ii-a) 階層統合型レディマクロタスクキューから CP 長の大きい MT_y を取り出し，アイドル状態の PE_j にマクロタスク番号 y を送信（MPI_Send）する．CP 長の値としては，最上位階層 MTG の出口ノードからの絶対 CP 長[6] を用いる．制御用マクロタスクに関しては，処理時間が短いため，スケジューリング処理用 PE で実行する．
(ii-b) MT_y で参照される共有データを，スケジューリング処理用 PE から，MT_y を実行する PE_j に送信（MPI_Send）する．
(iii-a) あるマクロタスク MT_z の実行が PE_k で終了した場合，そのマクロタスクを実行した PE の番号 k を受信（MPI_Recv）する．
(iii-b) MT_z で定義された共有データ（データローカライゼーションの適用されないのもの）を PE_k から受信（MPI_Recv）する．
(iii-c) MT_z の終了ステートを設定する．
(iv) 新たに実行可能になるマクロタスクを階層統合型レディマクロタスクキューに投入する．但し，そのマクロタスクがデータローカライゼーショングループ（3.6 節）に属しており，パーシャルスタティック割当てされる場合には，データローカライゼーショングループの先頭マクロタスクの割り当てられた PE 用のローカルマクロタスクキューに投入する．
(v) EndMT が終了してない間は (i-a) に戻る．EndMT が終了した際には，マクロタスク実行用 PE に終了信号を送信（MPI_Send）する．

3.4.2 マクロタスク実行用 PE コード

マクロタスク実行用 PE（PE_j）におけるマクロタスク実行の手順を以下に示す．

(i) スケジューリング処理用 PE から送信されたマクロタスク番号 y を受信（MPI_Recv）する．
(ii-a) MT_y で参照される共有データ（データローカライゼーションの適用されないもの）を，スケジューリング処理用 PE から受信（MPI_Recv）する．
(ii-b) MT_y の実行コードを実行する．
(ii-c) MT_y の実行を終了した PE の番号 j を，スケジューリング用 PE に送信（MPI_Send）する．
(ii-d) MT_y で定義された共有データ（データローカライゼーションの適用されないもの）を，スケジューリング用 PE に送信（MPI_Send）する．
(iii) EndMT の終了信号を受信（MPI_Recv）していない間は (i) に戻る．

3.5 マクロタスク間データ転送の MPI 実装

階層統合型実行制御を伴う粗粒度タスク並列処理用コードは，SMP 上で実装する場合，マクロタスク間データ転送コードを明示的に生成する必要はないが，MPI 環境ではデータ配置を決定し，マクロタスク間データ転送コードを生成する必要がある．

本手法では，基本的にスケジューリング処理用 PE において，マクロタスク間共有データを管理する方式をとる．このため，マクロタスク間データ転送は，スケジューリング処理用 PE を介して行われることになる．

MT_y をマクロタスク実行用 PE（PC_j）に割り当てる際には，MT_y で参照される共有データを PC_j に送信する．具体的には，3.4.1 節の (i-b) と (ii-b)，3.4.2 節の (ii-a) が対応する．一方，MT_y の実行がマクロタスク実行用 PE（PE_j）で終了した時には，MT_y で定義された共有データを PE_j から受信する．これは，3.4.1 節の (iii-b) と 3.4.2 節の (ii-d) に対応する．この際，複数の配列データを送受信する際には，オーバヘッドを軽減するために，MPI_Pack と MPI_Unpack を用いて一括転送している．

但し，3.6 節のデータローカライゼーションを適用する場合には，特定のマクロタスク間において，PE 上のローカルメモリを介してデータ転送が行われる．この場合，スケジューリング用 PE を介したデータ送受信は行われない．

なお，関連研究としては，PC クラスタ上でマクロデータフロー処理を実現するために，データ到達条件[8] を用いる方法が提案されているが，階層統合型実行制御やデータローカライゼーションは対象とされていない．

3.6 データローカライゼーションによる PE 間データ転送最小化

本手法では，ダイナミックスケジューリング方式を採用しているため，マクロタスク間の共有データに関するデータ転送は，スケジューリング処理用 PE を介して行われている．

このため，**図 4** の評価用プログラムのように多量のマクロタスク間データ転送が必要となる場合には，データ転送オーバヘッドが大きくなる．このような問題を解決するために，本手法では，パーシャルスタティック割当てを用いたデータローカライゼーション手法[7] を導入する．階層統合型粗粒度タスク並列処理では，同一階層のマクロタスク間データ転送のみならず，異なる階層のマクロタスク間データ転送に対しても，できるだけ PE 上のローカルメモリを介して行うことにより，データ転送オーバヘッドを最小化することが必要となる．

本手法では多量のマクロタスク間データ転送を必要とするマクロタスク集合をデータローカライゼーショングループと定義し，それらのマクロタスク集合の間では，PE 上のローカルメモリを介したデータ転送を実現する．その際，データローカライゼーショングループ内のマクロタスク集合は同一 PE に割り当てることが必須であるため，パーシャルスタティック割当てを用いたダイナミックスケジューリングを用いる[7]．

データローカライゼーショングループの決定方法としては，同一階層あるいは異なる階層にまたがったマクロタスク間のデータ転送を解析し，複数のデータ依存先行マクロタスクがある場合や複数のデータ依存後続マクロタスクがある場合には，CP 長の大きいマクロタスクを優先的に選び，それらのマクロタスク集合をデータローカライゼーショングループと定義する．つまり，互いに並列性のあるマクロタスク集合を同一のデータローカライゼーショングループと定義することはない．

例えば，図 4(a) の場合，(MT1, MT711) はデータローカライゼーショングループと定義され，パーシャルスタティック割当てを用いたダイナミックスケジューリングにより同一 PE に割り当てられ，ローカルメモリ経由のデータ転送が行われる．一方，図 4(b) の場合，MT4 で定義されたデータは MT72 と MT8 の両方で参照されるが，CP 長の大きい MT72 が選ばれ，その後続の MT75 もあわせて，データローカライゼーショングループと定義される．この場合，(MT4, MT72, MT75) は，パーシャルスタティック割当てを用いたダイナミックスケジューリングにより同一 PE に割り当てられ，ローカルメモリ経由のデータ転送が行われる．但し，MT8 は，MT72 及び MT75 との間の粗粒度並列性があるたるため，異なるプロセッサに割り当てられる可能性が高く，MT4 と MT8 の間のデータ転送は，スケジューリング処理用 PE を介したデータ転送が行われる．

4. MPI 実装による階層統合型粗粒度タスク並列処理の性能評価

本章では，階層統合型実行制御を伴う粗粒度タスク並列処理を MPI により実装し，PC クラスタ上で性能評価した結果について述べる．

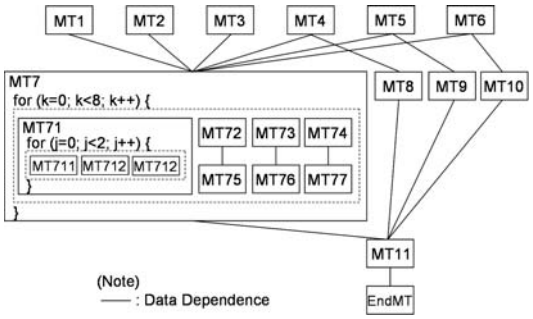

図 3 3 階層の評価用プログラムのマクロタスクグラフ．

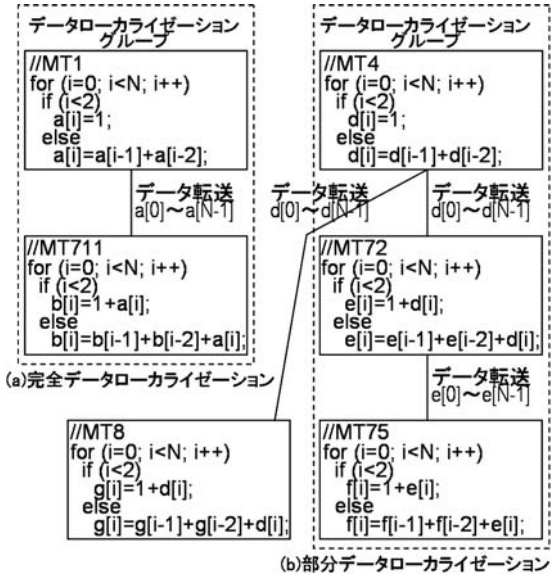

図 4 評価用プログラムにおけるデータローカライゼーショングループの例．

4.1 性能評価環境

性能評価に用いた PC クラスタは，7 台の PC を 1000BASE-T のイーサネットで接続した構成となっている．各 PC の仕様は，PentiumIII : 933MHz，メモリ : 256MB，OS : VineLinux3.2（Kernel-2.4.31），MPI : MPICH2（Version-1.03）となっている．

4.2 PC クラスタ上での性能評価

本節では，階層統合型実行制御を伴う粗粒度タスク並列処理を，**図 3** の 3 階層の評価用プログラム（その一部分を図 4 に示す）に対して適用し，PC クラスタ上で性能評価を行う．ここでは，PC クラスタの 1PE をスケジューリング処理用とし，他 PE をマクロタスク実行用とした並列プログラムを MPI を用いて作成した．この並列プログラムを 4.1 節の PC クラスタ上で実行した結果は，**図 5** と**図 6** の通りである．

図 5 は，各マクロタスクのループ回転数 N=1000

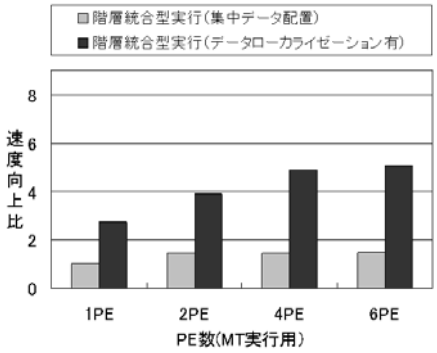

図 5 PC クラスタ上での階層統合型実行（N=1000）.

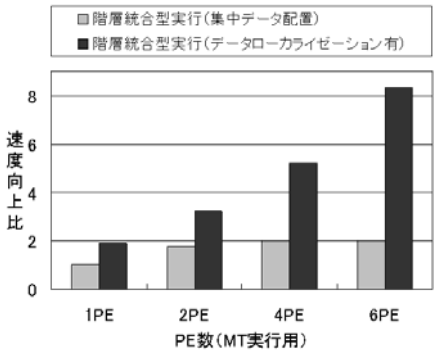

図 6 PC クラスタ上での階層統合型実行（N=10000）.

の場合の実行結果であり，図 4 に示すようにマクロタスク間で転送されるデータ数は N 個となる．ここで，マクロタスク実行に 6PE を用いた場合，1PE（集中データ配置）による実行と比べて，1.47 倍の速度向上となった．これは，マクロタスク間データ転送にスケジューリング処理用 PE を介して行っているため，データ転送オーバヘッドが原因で処理時間があまり短縮されていないと考えられる．

一方，データローカライゼーションを伴う階層統合型粗粒度タスク並列処理により実行を行うと，スケジューリング処理用 PE を介したデータ転送が大幅に軽減されており，6PE において 5.06 倍の速度向上が得られている．ここで，図 3 のプログラムにおけるデータローカライゼーション適用部分は，(MT1, MT711)(図 4(a))，(MT2, MT712)，(MT3, MT713)，(MT4, MT72, MT75)（図 4(b)），(MT5, MT73, MT76)，(MT6, MT74, MT77) となる．

図 6 はデータ数 N=10000 の場合の結果であり，6PE 実行において，データローカライゼーション無で 2.00 倍，データローカライゼーション有で 8.33 倍の速度向上が得られている．以上の結果，データローカライゼーションを伴う階層統合型粗粒度タスク並列処理は，MPI 環境においても効率よく実現できることが確認された．

5. おわりに

本稿では，PC クラスタ上で階層統合型実行制御を伴う粗粒度タスク並列処理の実現する方法を提案した．本手法により，PC クラスタ上でダイナミックスケジューリングを用いた階層統合型粗粒度タスク並列処理を MPI 実装できるようになり，全階層にまたがった粗粒度タスク間並列性を利用することが可能となった．

また，パーシャルスタティック割当てを用いたデータローカライゼーションを階層統合型粗粒度タスク並列処理に導入することにより，データ転送オーバヘッドの軽減も実現している．PC クラスタ上での性能評価の結果からも，十分な速度向上が得られており，提案手法の有効性が確かめられた．

今後の課題としては，ベンチマークプログラムを用いて性能評価を行うことや，粗粒度並列性とデータ転送オーバヘッドのトレードオフを検討することがあげられる．

参 考 文 献

1) M. Wolfe. High performance compilers for parallel computing. *Addison-Wesley Publishing Company*, 1996.
2) 笠原博徳, 小幡元樹, 石坂一久. 共有メモリマルチプロセッサシステム上での粗粒度タスク並列処理. 情報処理学会論文誌, Vol. 42, No. 4, 2001.
3) 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.
4) X. Martorell, E. Ayguade, N. Navarro, et. al. Thread Fork/Join techniques for multi-level parallelism exploitation in NUMA multi-processors. *Proc. of International Conference on Supercomputing*, 1999.
5) C. J. Brownhill, A. Nicolau, S. Novack, and C. D. Polychronopoulos. Achieving multi-level parallellization. *Proc. of ISHPC'97*, Nov. 1997.
6) 吉田明正. 粗粒度タスク並列処理のための階層統合型実行制御手法. 情報処理学会論文誌, Vol. 45, No. 12, pp.2732-2740, 2004.
7) 吉田明正, 越塚健一, 岡本雅巳, 笠原博徳. 階層型粗粒度並列処理における同一階層内ループ間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. 40, No. 5, pp.2054-2063, 1999.
8) 本多弘樹, 上田哲平, 深川保, 弓場敏嗣. 分散メモリシステム上でのマクロデータフロー処理のためのデータ到達条件. 情報処理学会論文誌, Vol. 43, No. SIG 6(HPS 5), 2002.