計算機アーキテクチャ 152－7
（２００３．３．１０）

ソフトウェア分散共有メモリを用いたマクロデータフロー処理

田邊浩志[†]　本多弘樹[†]　弓場敏嗣[†]

マクロタスク間の並列性を実行開始条件で表現したマクロデータフロー処理は，従来より共有メモリシステム上で実現されてきた．このマクロデータフロー処理を分散メモリ上で実現するためには，異なるプロセッサに割り当てられたマクロタスク間でデータ授受をおこなう機能が新たに必要となる．この問題を解決するため，本稿では，ソフトウェア分散共有メモリを用いてマクロデータフロー処理を実現する方式を提案する．提案方式では，コンパイル時に求めたマクロタスク間の依存解析の結果をもとに，ソフトウェア分散共有メモリの一貫性制御機構を利用してデータ授受をおこなう．さらに提案方式を 2 つのページベースソフトウェア分散共有メモリの TreadMarks と JIAJIA を用いて予備的な実装をおこない，その動作検証をおこなった．

Macro-Data-Flow using Software Distributed Shared Memory

Hiroshi Tanabe,[†] Hiroki Honda[†] and Toshitsugu Yuba[†]

Macro-Data-Flow processing using “execution start condition” has been realized on shared memory systems. An implementation of the Macro-Data-Flow processiong on distributed memory systems requires new functions for data transfer between macro-tasks assigned to distinct processors. This paper proposes an implementation method to realize Macro-Data-Flow processing using Software Distributed Shared Memory system. The proposed method utilizes the result of data dependence analysis between macro-tasks at the time of compilation, and maintains coherence between them. This paper discribes the implementation of the proposed method by using two well-known page-based Software Distributed Shared Memory systems, TreadMarks and JIAJIA, and gives its preliminary evalation.

1. はじめに

共有メモリシステム上の並列処理技術として，従来よりループレベルの並列化処理がおこなわれてきた．更なる性能向上のために，ループ並列化に加えて，サブルーチンや基本ブロックといった粗粒度タスク (マクロタスク) レベルの並列性を実行開始条件[1] として表現し，その並列性を利用するマクロデータフロー処理が注目されている[2),3)]．これまでのマクロデータフロー処理は共有メモリシステムを対象として実現されてきたため，マクロタスクで使用する変数の値は共有メモリ上に存在することを前提とすることが可能であった．

一方，近年の並列処理のプラットフォームとして PC クラスタが普及するようになり，このような分散メモリシステム上でのマクロデータフロー処理の実現が望まれている．分散メモリシステム上でマクロデータフロー処理を実現するには，異なるプロセッサに割り当てられたマクロタスク間のデータ授受が問題となる．

この問題を解決するアプローチとしては，ソフトウェア分散共有メモリの仮想的な共有メモリ空間を利用する方法と，ソフトウェア分散共有メモリを用いずに明示的なメッセージ通信をおこなう方法とが考えられる．

後者については，どのマクロタスク間でどの変数に対するデータ授受が必要となるかを実行時に判断する方法が必要となる．これに対し，我々はマクロタスク間でのデータ授受の関係をコンパイル時にデータ到達条件[4] として求め，実行時にこの条件を検査することでデータ授受の送信元・送信先を決定する方法を提案してきた．

前者のソフトウェア分散共有メモリを用いた実現方法では，マクロタスク間でのデータ授受に明示的なメッセージ通信を必要としないものの，その性能が各種ソフトウェア分散共有メモリの一貫性制御方式などに依存する．そのため，適切なソフトウェア分散共有メモリの撰択が必要となる．さらに，Lazy Release Consistency(LRC)[5] などに代表される緩い一貫性モデルでは，同期操作に関連づけられたメモリ参照についてしか一貫性が保証されない．すなわち，このような緩い一貫性モデルでマクロタスク間のデータの一貫性をとるには，一貫性モデルにあわせた適切な同期操

† 電気通信大学 大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications.

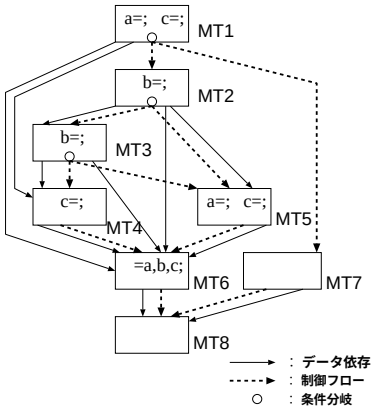

図 1 マクロフローグラフの例

作が必要となる．

本稿では，ソフトウェア分散共有メモリ上でのマクロタスク間のデータ授受を可能とするデータ一貫性制御方式を提案し，ソフトウェア分散共有メモリを用いてマクロデータフロー処理を実現する方式について述べる．さらに，一貫性モデルや一貫性制御方式が異なる２つのページベースソフトウェア分散共有メモリの TreadMarks[6] と JIAJIA[7] を用いてマクロデータフロー処理の予備的な実装をおこない，その動作検証をおこなった．

2. マクロデータフロー処理

2.1 マクロタスク

マクロデータフロー処理では，プログラムのループや基本ブロック，サブルーチンなどの粒度の大きい処理をマクロタスクとし，このマクロタスクをプロセッサへの割り当て単位として並列に処理するものである．コンパイル時にはプログラムをマクロタスクに分割し，マクロタスク間の制御フローとデータ依存を**図 1** に示すようなマクロフローグラフで表現する．

2.2 実行開始条件

実行開始条件[1] とは，あるマクロタスクの実行開始が可能となるための条件で，他のマクロタスクの実行状況を項とした論理式で表現したものである．この論理式は ＜マクロタスク終了＞ と ＜ 分岐方向決定 ＞ の二種類の原子条件，および論理演算子の ∨(論理和) と ∧(論理積) で構成される．マクロタスク MTa の終了条件は，MTa の実行が終了したときに $True$ となる条件である．マクロタスク MTb から MTc への制御フローエッジへの分岐による分岐方向決定条件は，条件分岐の条件式の評価によってこの制御フローエッジへの分岐が確定したときに $True$ となる条件である．

2.3 マクロタスクスケジューリング

マクロデータフロー処理では，実行時にスケジューラが順次マクロタスクをプロセッサに割り当てることで処理を進める．スケジューラの実装には，集中型ダイナミックスケジューリング方式と分散型ダイナミック方式が可能である[2),3),8)]．集中型では特定のプロセッサにスケジューリングコードを実行させるのに対し，分散型では各プロセッサにスケジューリングコードを分散させて実行させる方式である．

3. ソフトウェア分散共有メモリを用いたマクロデータフロー処理

分散メモリシステム上でマクロデータフロー処理を実現するにはマクロタスク間のデータ授受が問題となる．そこで 3.1 節でこの問題について述べ，3.2 節で本稿で対象とするソフトウェア分散共有メモリでの緩い一貫性モデルを説明し，3.3 節でその一貫性モデルを用いたマクロデータフロー処理において，マクロタスク間のデータ授受を可能とするデータ一貫性制御方式を提案する．

3.1 マクロタスク間でのデータ授受

実行開始条件を用いたマクロデータフロー処理では．あるマクロタスク MTi の実行開始条件が成立した時点で，MTi がデータ依存する全てのマクロタスクは「実行が終了している」か「実行されない」ことが確定している．共有メモリシステムにおいては，たとえデータ依存するマクロタスクが異なるプロセッサで実行されていたとしても，MTi で使用するデータは共有メモリ上に存在しているとして，MTi を実行することができる．よって，特にマクロタスク間のデータ授受を明示的におこなう必要がなかった．

一方，分散メモリシステムにおいては，データ依存するマクロタスクが異なるプロセッサで実行される際に，明示的なデータ通信を必要とする場合がある．そのためにはどのマクロタスク間でどの変数に関するデータ授受をおこなうかが明確でなくてはならない．

しかしながら，あるマクロタスク MTi で変数 V の使用 (MTi 外で定義された) があり，V への定義が複数のマクロタスクで行われる際，そのうちどのマクロタスクで定義された V の値を MTi で使用するべきなのかは，一般的に実行時にならなければ決定できない．よって，どのマクロタスク間でどの変数に関する通信をおこなうかという判断は実行時におこなうこととなるが，実行開始条件だけではそのための情報は不十分であり，通信相手と変数を決定することができないという問題がある．

3.2 対象とする一貫性モデル

これまでのソフトウェア分散共有メモリの研究において，一貫性維持操作のためのオーバヘッドを削減するために様々な緩い一貫性モデルが提案されてきた．本稿ではその中で現在もっとも一般的といえる一貫性モデルとして Lazy Release Consistency(LRC)[5] と Scope Consistency(ScC)[9] を対象とする．これらの一貫性モデルでは，ページを一貫性を維持する単位として扱い，あるプロセッサによって更新されたページの内容は即座に他のプロセッサには反映させず，次の同期操作まで遅延させる方式である．

この 2 つの一貫性モデルは更新内容をどこまで保証するのかという点で異なる．LRC ではロックの解放時において，それまでに更新された全ての情報がロックを獲得したプロセッサに対して反映されることを保証する．それに対し，ScC ではロック変数ごとのロックの獲得から解放までを Scope とし，Scope の中で更新された情報が，同一 Scope に入るプロセッサに対して反映されることを保証する．したがって，LRC に比べて ScC はより緩い一貫性モデルである．

各プロセッサで更新されたページは，ロックの解放時に write notice 構造体としてロック変数に記録させる．次にロックを獲得するプロセッサは，write notice の情報をもとに更新されたページ (と更新をおこなったプロセッサ) を検出することができる．

3.3 データー貫性制御

本稿で提案するソフトウェア分散共有メモリを用いたマクロデータフロー処理では，コンパイル時に求めたマクロタスク間のデータ依存解析をもとに，ソフトウェア分散共有メモリのロック操作によるメモリ一貫性維持の枠組を利用することでマクロタスク間のデータ授受とその一貫性の制御をおこなう．

3.3.1 ロック操作を付加したマクロタスクへの変換

コンパイル時にプログラムはマクロタスクに分割され，マクロタスク間のデータ依存と制御フローはマクロフローグラフとして表現される．このマクロフローグラフ内のマクロタスクに対し，以下に示すロック操作を付加する変換をおこなうことでデータの一貫性を制御する．このロック操作によりマクロタスク間では適切なデータ授受がおこなわれる．

(1) マクロタスク MTi に対して，ユニークなロック変数 L_{MTi} を割り当てる．

(2) MTi の先頭に L_{MTi} を獲得するコードを挿入し，MTi の最後には L_{MTi} を解放するコードを挿入する．

この変換で，MTi の実行によって更新されたページに関する情報が write notice として当該ロック変数に記録される．

(3) MTi の実行にさきがけて，MTi がデータ依存するマクロタスク MTj のロック変数 L_{MTj} の獲得と解放をするコードを挿入する．

MTi を実行する前に L_{MTj} を獲得することで L_{MTj} の write notice を受け取ることとなり，MTj で更新されたページの内容を検出できる．このことにより MTi と MTj の間で適切なデータの授受がおこなわれるようになる．

図 2 と**図 3** に，ロック操作を付加したマクロタスクへの変換前と変換後のマクロフローグラフの例をそれぞれ示す．

3.3.2 データ到達条件との相違点

我々はまた，従来の到達する定義[10] の概念を拡張したデータ到達条件[4] を提案し，これを用いてマクロタスク間で明示的な通信をおこなうことによりソフトウェア分散共有メモリを用いずに分散メモリシステム上でのマクロデータフロー処理を実現してきた．ここではデータ到達条件による実行方式と本方式との相違点について述べる．

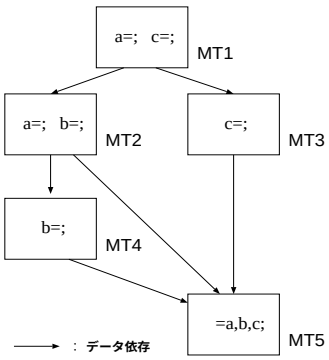

図 2 変換前のマクロフローグラフ

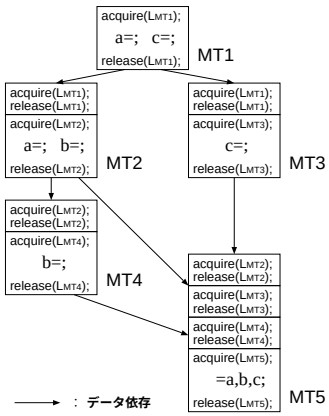

図 3 変換後のマクロフローグラフ

コンパイル時： データ到達条件による実行方式では，各マクロタスクに対して，そのマクロタスクで使用する全ての変数とそのマクロタスクに到達する定義をもつマクロタスクの組に対してデータ到達条件を求める．

一方，本方式ではマクロフローグラフに対して提案するロック操作をおこなうコードを挿入する．

実行時のスケジューラ： データ到達条件による実行方式では，実行開始条件の検査とマクロタスクの割当て・実行指示に加えて，データ到達条件の検査によりデータ転送の必要性を判定し，必要であればプロセッサにその指示を与える．

一方，本方式では実行開始条件の検査とマクロタスクの割当て・実行指示のみをおこなう．データの一貫性制御に必要なデータ転送はソフトウェア分散共有メモリのシステムが行う．

4. 予備的実装と動作検証

ソフトウェア分散共有メモリを用いてマクロデータ

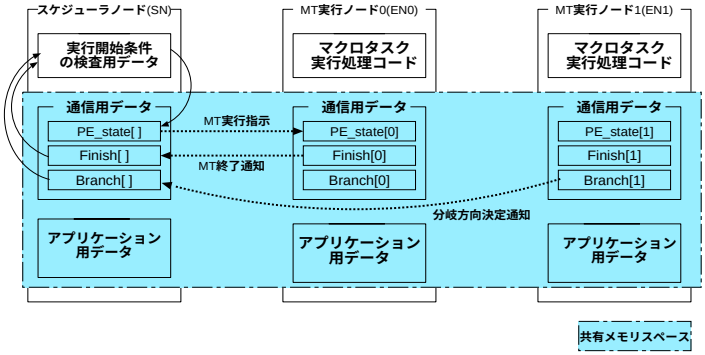

図 4 集中型ダイナミックスケジューリング方式のメモリレイアウト

フロー処理の予備的な実装をおこない，提案するデータ一貫性制御方式の動作検証をした．本章では実装の概要と動作検証の結果について述べる．

4.1 実装の概要

マクロデータフロー処理の実装方式には，集中型ダイナミックスケジューリング方式と分散型ダイナミックスケジューリング方式の実装が可能であるが，本実装においてはデータ到達条件を用いた分散メモリシステム上でのマクロデータフロー処理[4] と同様に集中型ダイナミックスケジューリング方式を採用した．

集中型ダイナミックスケジューリング方式では，スケジューリング用のコードをマクロタスク実行用のコードとは別に生成し，1つのノードがスケジューリングコードを処理し，残りのノードがマクロタスク実行コードを処理する．以後，スケジューリングコードを処理するノードをスケジューラノード (SN)，その他のノードをマクロタスク実行ノード (EN) と呼ぶ．

SN は実行開始条件を検査するとともに，実行開始条件が成立したマクロタスクの EN への割り当てをおこなう．一方，EN は SN から割り当てられたマクロタスクを実行するとともに，実行中のマクロタスクの＜マクロタスク終了＞や＜分岐方向決定＞などの情報を SN に対して通知する．これらの通信用データのために，以下の 3 つの変数がソフトウェア分散共有メモリ上に共有変数として割り当てられる．

(1) `PE_State[ノード番号]`: SN から EN へマクロタスクを割り当てるを通知するのに用いる変数．
(2) `Finish[ノード番号]`: EN に割り当てられたマクロタスクの実行終了を SN に通知するために用いる変数．
(3) `Branch[ノード番号]`: EN で実行中のマクロタスクにおいて，決定した分岐方向を SN に通知するために用いる変数．

SN ではローカル変数として用意された実行開始条件の検査用データと，上記の通信データを用いて以下のスケジューリングコードを処理する．

(1) `Finish[]` と `Branch[]` をポーリングして EN から＜マクロタスク終了＞と＜分岐方向決定＞の通知を受け取り，通知された情報に基づき実行開始条件の更新をおこなう．
(2) 実行開始条件が成立したマクロタスクを `PE_State[]` を介して EN に割り当ての通知をおこなう．

EN ではソフトウェア分散共有メモリ上に割り当てられたアプリケーション用データと，通信用のデータを用いて以下のマクロタスク実行コードを処理する．

(1) `PE_State[]` をポーリングして SN からマクロタスクの割り当て通知を受け取り，通知されたマクロタスクの実行を開始する．ここでのマクロタスクのコードは，3.3 節で述べたロック操作をおこなうように変換されたものである．
(2) 割り当てられたマクロタスクの＜マクロタスク終了＞と＜分岐方向決定＞は，`Finish[]` と `Branch[]` を介して SN に通知する．

ソフトウェア分散共有メモリを用いた集中型ダイナミックスケジューラ方式でのマクロデータフロー処理の概要図を**図 4** に示す．

表 1 システム構成

CPU	PentiumIII 866 MHz
メモリ	1GB
ネットワーク	100BASE-TX
OS	Red Hat Linux 7.1
	Kernel 2.4.10
コンパイラ	egcs 1.1.2

4.2 実装環境

提案方式の動作検証のため，**表 1** に示す PC クラスタ上に，集中型ダイナミックスケジューリング方式によるマクロデータフロー処理を実装した．実装には一貫性モデルや一貫性制御方式が異なる 2 つのページベースソフトウェア分散共有メモリとして， TreadMarks version 1.0.3.3[6] と JIAJIA version 2.1[7] を用いた．

これらのソフトウェア分散共有メモリは，OS のメモリ管理機能を利用しユーザレベルのソフトウェアライブラリとして実現されている．共有メモリの一貫性制御は OS が提供するページ単位で行われ，その各ページにホームノードを割り当てて一貫性制御をおこ

表 2 swim の実行時間 [秒] と対遂次速度向上率

EN 数	TreadMarks		JIAJIA	
	R	PA	R	PA
遂次	47.8(1)	47.8(1)	47.8(1)	47.8(1)
2	267.4(0.18)	29.3(1.63)	378.2(0.13)	41.2(1.16)
4	187.6(0.25)	17.6(2.72)	285.8(0.17)	25.7(1.86)

表 3 swim のメッセージ転送数とデータ転送量

EN 数	メッセージ転送数				データ転送量 [MB]			
	TreadMarks		JIAJIA		TreadMarks		JIAJIA	
	R	PA	R	PA	R	PA	R	PA
2	1,730,008	84,772	517,731	26,118	1880	25.8	2257	30.9
4	3,722,683	118,601	729,791	46,950	2124	42.9	2432	53.1

なうホームベース型と，ホームノードを割り当てないホームレス型の一貫性制御プロトコルがある．

TreadMarks は一貫性モデルに Lazy Release Consistency[5] を適用し，一貫性制御プロトコルはホームレス型のプロトコルを採用している．一方，JIAJIA は一貫性モデルに Scope Consistency[9] を適用し，一貫性制御プロトコルはホームベース型のプロトコルを採用している．

4.3 動作検証

動作検証のためのベンチマークプログラムとして SPEC CFP95 の swim(データサイズ 505x505) を用いた．このプログラムのマクロタスク分割や並列性検出などの並列化と，提案するロック操作の付加は人手によっておこった．

並列化に際しては，主ループの内側の処理をマクロタスクに分割し，マクロデータフロー処理をおこなうようにした．ここでのマクロタスクの主要なものは，ループを分割または融合したものである．また，マクロタスク間のデータ授受は配列変数によるものである．主ループの内側の処理を 4 並列で実行したときに最適となるようなマクロタスク分割をした場合のマクロフローグラフを**図 5** に示す．

図 5 のマクロフローグラフでは．主ループの各イタレーション内でのデータ依存関係のみを示している．実際にはこの他にイタレーション間にまたがるデータ依存が存在している．

もともとの遂次プログラムを実行したときと，並列化したプログラムを EN 数が 2 の場合，および 4 の場合において並列実行したときの実行時間を**表 2** に示す．また，アプリケーション用のデータおよび一貫性制御のためにノード間で転送されたメッセージ数，および転送されたデータの総量を**表 3** に示す．

表中の R は実行開始条件が成立したマクロタスクをアイドルの EN にランダムで割り当てるようにしたときの結果である．また，マクロデータフロー処理を効率よく処理するためには，データ転送量を考慮したマクロタスクの割り当てが必要である[3),11)]．そこで，文献 11) のパーシャルスタティック割当てによるスケジューリング手法を参考にし，本実装においてもデータ共有量の多いマクロタスクをグループ化し，そのグループのマクロタスクを同じ EN に割り当てるようにすることでデータ転送量を削減するようした．これを，マクロタスクを割り当てる EN をあらかじめ決定していることから，便宜上プリアサインとよび，表中では PA で表す．

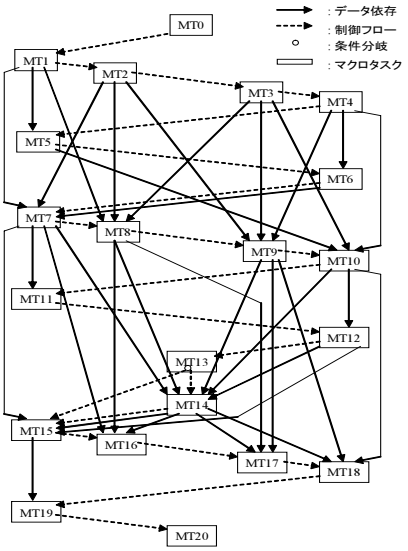

図 5 swim のマクロフローグラフ

図 5 の例では，MT1, MT5, MT7, MT11, MT15, MT19 はデータ共有量の多いマクロタスクなので，これらをグループ化し EN0 に割り当てるようにした．同様に MT2, MT8, MT16 は EN1 に，MT3, MT9, MT17 は EN2 に，MT4, MT6, MT10, MT12, MT18 は EN3 に割り当てるようにした．これ以外のマクロタスクについてはどの EN に割り当てもデータ転送がない，もしくは転送するデータ量に違いがないのでプリアサインはおこなわなかった．

また，JIAJIA のようにホームベース型の一貫性制

御プロトコルでは，ホームノードの割り当てが実行性能に大きく影響する[12] ので，PA においては次のようなホームノードの割り当てをおこないデータ転送量の削減を図った．

生成した主要なマクロタスクは２重ループの外側ループをブロックに分割したものであり，これらの２重ループでの２次元配列へのアクセスは，ループプインデックスと配列の次元とが一致している．プリアサインによる並列実行では２次元配列も外側でブロック分割し，マクロタスクが割り当てられる EN がブロック分割した配列のホームノードとなるように指定した．ランダムなマクロタスク割り当てによる並列実行では，特にホームノードの指定はおこなわず round-robin で割り当てた．

TreadMarks と JIAJIA によるマクロデータフロー処理の実行において，マクロタスクの割り当てがランダムによるものと，プリアサインによるもののいずれの場合もマクロタスク間で正しくデータ授受がおこなわれ，矛盾なく並列実行されていることが確認された．

並列処理効果については，データ転送量を考慮しないでマクロタスクの割り当てをおこなった場合，ノード間で頻繁にデータ転送がおこなわれるため全く並列処理効果がでず，逐次の実行時間よりも遅くなっている．一方，データ転送量を考慮してマクロタスクの割り当てをおこなった場合，EN 数が 4 のときにデータ転送量を考慮しない場合と比べて，メッセージ数が TreadMarks で 97%，JIAJIA で 93%削減された．データ転送量においても，TreadMarks で 98%，JIAJIA で 98%削減された．この結果，実行時間が大幅に短縮され，TreadMarks では 2.72 倍，JIAJIA では 1.86 倍の並列処理効果が得られた．

JIAJIA の実行時間が TreadMarks に比べて長くなっているのは，一貫性制御プロトコルの違いによるロック操作のコストの差や，JIAJIA の場合ではホームノードの割り当てが必ずしも最適ではなかったため TreadMarks に比べてデータ転送量が大きくなったためと考えられる．

5. おわりに

本稿では，ソフトウェア分散共有メモリを用いてマクロデータフロー処理を実現する方式を提案し，ベンチマークプログラムを用いて動作検証をおこなった．本方式はソフトウェア分散共有メモリのロック操作による一貫性維持の枠組を利用することで，データ到達条件による明示的な通信をすることなく，マクロタスク間のデータ授受をおこなうことができる．本方式を TreadMarks と JIAJIA を用いて予備的な実装をおこない動作検証した結果，ノード間のデータ転送量を考慮したマクロタスクの割り当てやホームノードの割り当てをおこなうことで，ある程度の並列処理効果を得ることができた．

データ転送コストの大きい分散メモリシステム上でのマクロデータフロー処理は，データ転送量を削減するためのマクロタスク分割や割り当て方法が性能に大きく影響する．今後の課題として，ソフトウェア分散共有メモリの一貫性制御方式に適応するマクロタスク分割や割り当て手法，ならびにホームノードの配置手法の確立などがあげられる．

謝辞 本研究の一部は文部科学省科学研究費（基盤 C，2，12680336）によって行われた．

参 考 文 献

1) 本多弘樹, 岩田雅彦, 笠原博徳: Fortran プログラム粗粒度タスク間の並列性検出手法, 電子情報通信学会論文誌, Vol. J73-D1, No. 12, pp. 951–960 (1990).
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, pp. 910–920 (2001).
3) 石坂一久, 中野啓史, 八木哲志, 小幡元樹, 笠原博徳: 共有メモリマルチプロセッサ上でのキャッシュ最適化を考慮した粗粒度タスク並列処理, 情報処理学会論文誌, Vol. 43, No. 4, pp. 958–970 (2002).
4) 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:ハイパフォーマンスコンピューティングシステム, Vol. 43, No. SIG6(HPS 5), pp. 45–55 (2002).
5) Keleher, P., Cox, A. L. and Zwaenepoel, W.: Lazy Release Consistency for Software Distributed Shared Memory, *Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA'92)*, pp. 13–21 (1992).
6) Keleher, P., Dwarkadas, S., Cox, A. and Zwaenepoel, W.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, *Proceedings of the Winter 94 Usenix Conference*, pp. 115–131 (1994).
7) Hu, W., Shi, W. and Tang, Z.: JIAJIA: An SVM System Based on A New Cache Coherence Protocol, *Proceedings of the High Performance Computing and Networking (HPCN'99)*, pp. 463–472 (1999).
8) 合田憲人, 岩崎清, 岡本雅巳, 笠原博徳, 成田誠之助: 共有メモリ型マルチプロセッサシステム上での Fortran 粗粒度タスク並列処理の性能評価, 情報処理学会論文誌, Vol. 37, No. 3, pp. 418–429 (1996).
9) Iftode, L., Jaswinder, Singh, P. and Li, K.: Scope Consistency: A Bridge between Release Consistency and Entry Consistency, *Proc. of the 8th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'96)*, pp. 277–287 (1996).
10) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley (1988).
11) 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, 情報処理学会論文誌, Vol. 40, No. 5, pp. 2054–2063 (1999).
12) 佐藤三久, 原田浩, 長谷川篤志, 石川裕: Cluster-enabled OpenMP: ソフトウエア分散共有メモリシステム SCASH 上の OpenMP コンパイラ, 情報処理学会論文誌: ハイパフォーマンスコンピューティングシステム, Vol. 42, No. SIG 9(HPS 3), pp. 158–169 (2001).