Vol. 46 No. SIG 4(ACS 9) 情報処理学会論文誌：コンピューティングシステム Mar. 2005

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[†]

ループ並列処理の限界を超えてさらなる性能向上のために，ループやサブルーチンの粗粒度タスク（マクロタスク）レベル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．マクロデータフロー処理を分散メモリシステム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行う機能が必要となる．これに対し，我々は明示的な通信によってデータを授受するデータ到達条件による実行方式を提案しているが，コンパイル時にデータ参照を正確に解析できないプログラムでは不要なデータ転送をしてしまい，性能低下が問題となる．本稿では，マクロタスク間のデータ授受にソフトウェア分散共有メモリを用い，必要に応じたデータ転送を行う方式を提案する．2 つのページベースソフトウェア分散共有メモリの TreadMarks と JIAJIA を用いて PC クラスタ上で提案方式の実装と性能評価をした．その結果，不規則なデータ参照のプログラムに対して不要なデータ転送を削減でき，データ到達条件による実行方式に比べて最大 25%の性能向上が得られた．

Macro-data-flow Using Software Distributed Shared Memory

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

Macro-dataflow processing, which exploits a parallelism among coarse grain tasks (macro-tasks) such as loops and subroutines, is considered promising to break the performance limits of loop parallelism. To realize macro-dataflow processing on distributed memory systems, “data reaching conditions”, a method to make a sender-receiver pair of a data transfer determined at runtime, has been proposed. However, irregular data accesses induce extra data transfers, which lead to performance deteriorations. This paper proposes an implementation scheme using software distributed shared memory, which enables on-demand data fetching. This paper describes the implementations using two well-accepted page-based Software Distributed Shared Memory systems, TreadMarks and JIAJIA. Evaluation results on a PC cluster show software distributed memory approach is up to 25% faster than the “data reaching conditions”.

1. はじめに

共有メモリシステム上の並列処理方式として，従来よりループレベルの並列処理が用いられており，様々な優れた並列化技術が開発されてきた．さらなる性能向上のための方式として，ループレベルの並列処理に加えて，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）レベルの並列性を利用するマクロデータフロー処理が有効である[1),2)]．マクロデータフロー処理では，コンパイル時にループやサブルーチンなどをマクロタスクとして分割し，それらマクロタスク間の並列性を実行開始条件[3)] で表現する．実行時には実行開始条件を検査しながら，条件が成立したマクロタスクを順次プロセッサに割り当てることで並列実行する．

これまでの共有メモリシステムを対象としたマクロデータフロー処理では，実行開始条件が成立した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ため，特にマクロタスク間でのデータ授受を考慮することなく処理を進めることができた．

一方，分散メモリシステムでは，マクロタスクで使用する変数の値がマクロタスクを実行するプロセッサのメモリ上に存在しているとは限らず，これまでのようには処理を進めることができない．そのため，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でのデータ授受が新たに問題となる．

これを解決する方法として，大きく分けてコンパイラによりデータ授受のためのメッセージ通信コードを生成する方法と，ソフトウェア分散共有メモリ（SDSM）の仮想的な共有メモリ空間を利用する方法

† 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

が考えられる．

前者を実現するには，どのマクロタスク間でどの変数に対するデータ授受すべきかという情報が必要となる．この情報を，実行開始条件を拡張したデータ到達条件[4]としてデータ授受の関係を求め，実行時にこの条件を検査することでメッセージ通信に必要となる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[4]．

他方，後者の SDSM を用いる方法では，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コードを生成する必要がないものの，その性能が各種 SDSM の一貫性維持方式などに依存する．そのため，適切な SDSM の撰択が必要となる．

両者の特徴を比較すると，データ到達条件による通信管理方式は，ページベースの SDSM のように一貫性制御をすべて実行時に行うのではなく，コンパイル時のデータ依存解析によって可能な限り静的に行うようにしている．これにより，冗長な一貫性制御のための通信の除去や通信の集合化といった最適化が可能となる．しかし，不規則なデータ参照のような，実行時になって初めて参照されるデータが確定する（以降，このデータ参照を不確実なデータ参照と呼ぶ）プログラムでは，一貫性維持のための検査が大きなオーバヘッド[5)~7)]となったり，安全なデータ参照を保証するために必要以上のデータ転送をしたりすることがある．

これに対し，ページベースの SDSM による通信管理方式は，参照されるデータを実行時に検出するため，不確実なデータ参照でも不要なデータ転送を削減できることや，データの局所性がある場合には効率良く処理することが可能であると考えられる．しかし，ページ単位での一貫性制御のためのオーバヘッドによって，性能低下を引き起こすことが危惧される．また，Lazy Release Consistency（LRC）[8] などに代表される緩い一貫性モデルでは，一貫性制御のためのオーバヘッドを削減するため，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．そのため，このような緩い一貫性モデルにおいてマクロタスク間でデータの一貫性をとるには，新たに一貫性モデルに合わせた適切な同期操作を考慮しなければならない．

本稿では，SDSM の一貫性モデルに LRC と Scope Consistency（ScC）[9] を対象として，これらの緩い一貫性モデルでマクロタスク間のデータ授受を可能とするデータの一貫性制御手法を提案する．また，提案手法を用いて SDSM によるマクロデータフロー処理を実装し，従来手法のデータ到達条件による実行方式と比較する．これにより，従来手法では不要なデータ転送をしてしまう場合でも，提案手法により不要なデータ転送を削減できることを述べ，その有効性を示す．

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当て単位として並列に実行する．

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．マクロタスクへの分割にあたっては，制御フローグラフが非循環になるように分割する．

2.2 実行開始条件

実行開始条件とは，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[3]．この論理式は < マクロタスク終了 > と < 分岐方向決定 > の 2 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，論理式中では a と表記する．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，論理式中では b - c と表記する．

マクロタスク MTi の実行開始条件は式 (1) のように定義される．

$$\text{(MT}i\text{ の実行確定条件)} \quad \wedge \quad \text{(MT}i\text{ のデータアクセス可能条件)} \qquad (1)$$

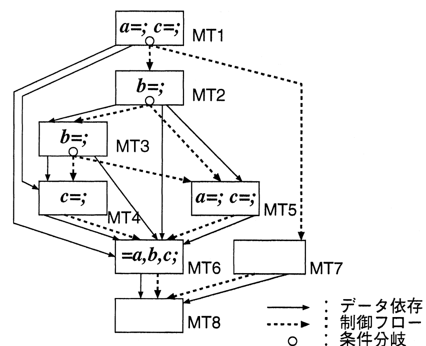


図 1　マクロフローグラフの例
Fig. 1 Example of macro-flow graph.

表 1 図 1 中の MT6 の実行開始条件
Table 1 Execution start condition for MT6 for Fig. 1.

実行確定条件	1-2	
データアクセス可能条件	MT1	1
	MT2	2 ∨ 1-7
	MT3	3 ∨ 1-7 ∨ 2-5
	MT4	4 ∨ 1-7 ∨ 2-5 ∨ 3-5
	MT5	5 ∨ 1-7 ∨ 3-4
実行開始条件	1-2 ∧ 1 ∧ (2 ∨ 1-7) ∧ (3 ∨ 1-7 ∨ 2-5) ∧ (4 ∨ 1-7 ∨ 2-5 ∨ 3-5) ∧ (5 ∨ 1-7 ∨ 3-4)	

式 (1) の MTi の実行確定条件は，MTi を実行することが確定する条件で，MTi が制御依存[10] するマクロタスクから MTi が逆支配するマクロタスクへの分岐による分岐方向決定条件の論理和で構成される．

式 (1) の第 2 項は，MTi で使用する変数へのアクセスが可能となる条件で，MTi がデータ依存するすべてのマクロタスクのそれぞれのマクロタスクに対するデータアクセス可能条件の論理積で構成される．

MTi がデータ依存する MTj に対するデータアクセス可能条件は式 (2) のように定義される．

$$(\text{MT}j\text{ の終了条件}) \quad \vee \quad (\text{MT}j\text{ の非実行確定条件}) \qquad (2)$$

式 (2) の MTj の非実行確定条件は MTj が実行されないことが確定するための条件で，MTj が補制御依存[3] するマクロタスクから MTj へ至らないパスへの分岐による分岐方向決定条件の論理和で構成される．

図 1 のマクロフローグラフ中の MT6 を例にすると，その実行開始条件は**表 1** のようになる．

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スケジューリング方式が可能である[2]．集中型の方式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，分散型の方式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．ここでは集中型の方式を例にとり説明する．

スケジューラは，コンパイル時に実行開始条件をもとに生成されたスケジューリングコードにより，以下の処理を繰り返し実行する[11]．

(1) 実行開始条件の検査：各マクロタスクから通知される <マクロタスク終了>，および < 分岐方向決定 > の情報をもとに実行開始条件を検査し，条件が成立したマクロタスクをレディマクロタスクとする．

(2) 割当てプロセッサの決定：所定の割当て戦略に従って，レディマクロタスク中のどのマクロタスクをどのプロセッサに割り当てるかを決定する．

(3) マクロタスクの実行指示：各プロセッサに実行すべきマクロタスク番号を通知する．

マクロタスクを実行するプロセッサは，以下の処理を繰り返し実行する．

(1) 実行指示の検査：スケジューラからマクロタスクの実行指示があるかを検査する．

(2) マクロタスクの実行：実行指示があった場合には，通知されたマクロタスク番号の実行コードを処理する．

(3) スケジューラへの通知：スケジューラに対して，マクロタスクが終了したら < マクロタスク終了 > を通知し，条件分岐方向が決定したら < 分岐方向決定 > を通知する．

スケジューラからはマクロタスク番号のみが通知されるため，マクロタスクを実行する各プロセッサのコードは，すべてのマクロタスクの実行コードが含まれている．

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

本章では，ソフトウェア分散共有メモリ（SDSM）を用いてマクロデータフロー処理を実現するためのデータ一貫性制御手法を提案する．以下，まず分散メモリシステム上で問題となる，マクロタスク間のデータ授受について述べる．次に提案手法を詳述し，その後，提案手法による方式と SDSM を用いないデータ到達条件による方式を比較する．

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 がデータ依存するすべ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，マクロタスク間のデータ授受を明示的に行う必要がなかった．

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

データ授受を行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 外で定義された値の使用）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データ授受を行うかは実行時に判断することとなるが，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生じる．

3.2 対象とする SDSM の一貫性モデル

これまでの SDSM の研究において様々な一貫性モデルが提案されてきた．本稿では，その中で現在最も一般的な Lazy Release Consistency（LRC）[8] と Scope Consistency（ScC）[9] を対象とする．様々な一貫性モデルの中から LRC と ScC を対象としたのは，性能面で優れている緩い一貫性モデルであることと，多くの SDSM システムが LRC や ScC を実装していることから，そのような既存の SDSM システムを容易に適用できるという利点によるものである．

LRC や ScC での緩い一貫性モデルでは一貫性の維持をページ単位で扱い，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に次の同期操作まで遅延させることで，一貫性維持操作のオーバヘッドを削減している．

この 2 つの一貫性モデルの違いは，更新内容をどこまで保証するのかという点である．LRC ではロックの解放時において，それまでに更新されたすべ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．これに対し，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といえる．

3.3 データの一貫性制御

提案するデータの一貫性制御手法は，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，SDSM 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を維持する．

まず，LRC や ScC での SDSM システムにおけるロック操作による一貫性維持の枠組みについて説明する．SDSM システムでは，ロックの解放時に各プロセッサで更新したページの情報を write notice 構造体としてロック変数に記録する．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行ったプロセッサを検出し，この情報をもとに一貫性維持の処理を行っている．

一方，マクロデータフロー処理ではコンパイル時にプログラムをマクロタスクに分割し，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する．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行うことで，マクロタスク間のデータの一貫性を制御する．

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

(2) MTi の先頭に $L_{\mathrm{MT}i}$ を獲得するコードを挿入し，MTi の最後には $L_{\mathrm{MT}i}$ 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 $L_{\mathrm{MT}j}$ の獲得と解放を行うコードを挿入する．

　これは，まず MTi を実行する前に $L_{\mathrm{MT}j}$ を獲得することで，$L_{\mathrm{MT}j}$ 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する．次に，ロック変数を獲得したままでは他のマクロタスクが同ロック変数を獲得することができなくなるため，当該ロック変数を解放する．このことにより MTi と MTj の間で適切なデータの授受が行われるようになる．

以上のロック操作により，LRC もしくは ScC を実装する SDSM システムを変更することなく，マクロタスク間での適切なデータ授受が実現できる．**図 2** と

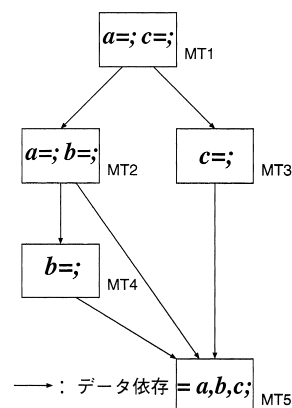


図 2 変換前のマクロフローグラフ
Fig. 2 Original macro-flow graph.

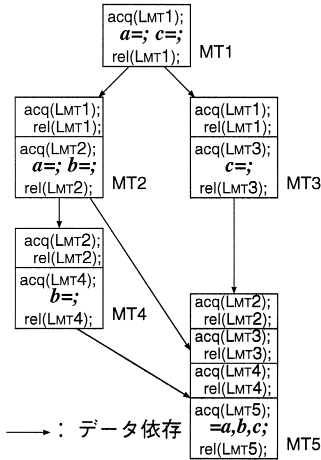


図 3 変換後のマクロフローグラフ
Fig. 3 Converted macro-flow graph.

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示す．

ただし，上記の変換では以下に示す不要なロック操作が含まれている．

(a) マクロタスク MTi にデータ依存する後続のマクロタスクが存在しなければ，(2) の変換で行う MTi の当該ロック変数の獲得と解放は不要となる．

(b) データ依存関係にあるマクロタスク間で，実際にデータ授受を行う必要があるのはフロー依存だけである．よって，(3) の変換においてデータ依存するマクロタスク MTj がフロー依存でない場合，すなわち逆依存か出力依存もしくはその両方の場合，MTj の当該ロック変数の獲得と解放は不要である．

(c) (3) の変換のうち，データ依存するマクロタスク MTj が実行されない場合，MTj ではデータの更新が行われないので MTj の当該ロック変数の獲得と解放は不要である．

不要なロック操作を行ってもデータの一貫性においては問題ないが，実行時のオーバヘッドとなるため，このようなロック操作が不要と判断できるものはコンパイル時に除去する．

3.4 データ到達条件を用いたマクロデータフロー処理との比較

我々はすでに，従来の到達する定義[12]の概念を拡張したデータ到達条件を提案し，これを用いてマクロタスク間で明示的な通信をすることにより，SDSM を用いずに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できることを示している[4]．ここではデータ到達条件について説明し，次に，定義および使用に関する不確実なデータ参照のプログラムに対して，データ到達条件方式では不要なデータ転送を行ってしまう問題を述べ，提案方式によってその問題を緩和できることを示す．

3.4.1 データ到達条件

データ到達条件とは，マクロタスク MTi（の先頭）に到達する[12]変数 V への定義を持つマクロタスクの集合を S_V^i としたとき，MTi（の先頭の文）での V の値が，MT$j \in S_V^i$ で定義した値となることが確定するための条件で，実行開始条件と同様に他のマクロタスクの実行状況を項とした論理式で表現する．

MTi での V に対する MT$j \in S_V^i$ のデータ到達条件は，MTj から MTi へのパス上において，MTj での V への定義を kill する定義を持つすべてのマクロタスクの集合を K としたとき，式 (3) のように定義する．

(MTj の実行確定条件) ∧
(K 中の全マクロタスクの非実行確定の条件) (3)

式 (3) の K 中の全マクロタスクの非実行確定の条件は，MTj から MTi へのパス上で MTj での V への定義を kill する定義を持つマクロタスクは 1 つも実行されないことが確定する条件で，そのようなマクロタスクの非実行確定条件の論理積で構成される．マクロタスク MTk の非実行確定条件は MTk が実行されないことが確定するための条件で，MTk が補制御依存[3]するマクロタスクから MTk へ至らないパスへの分岐による分岐方向決定条件の論理和で構成される．

3.4.2 不確実な定義による kill

前項でのデータ到達条件の説明において，あるマクロタスク MTk で変数 V への定義が行われるとは，MTk 内で必ず V への**確実な定義**[12]が行われることを前提としていた．一方，実際のプログラムでは，(i) MTk 内の文での V への定義自体が**不確実な定義**である場合，(ii) V への定義を実行するか否かがマクロタスク内の条件分岐方向によって実行時に決定される場合，(iii) V が配列変数で，それぞれの要素に対する定義を実行するか否かが実行時に決定される場合がある．上記 (i) から (iii) の場合，V への定義が MTk で kill されるか否かが確定せず，実行開始条件とデータ到達条件を求めることができない．

これに対して，MTk に到達する V への定義の値を MTk の入り口で V へ定義（代入）し直すこととすれば，実行時の状況によらず MTk で確実な定義が行われるようにできる．ただし，この方法は実際には kill されてしまう定義の値によるデータ転送を行ってしま

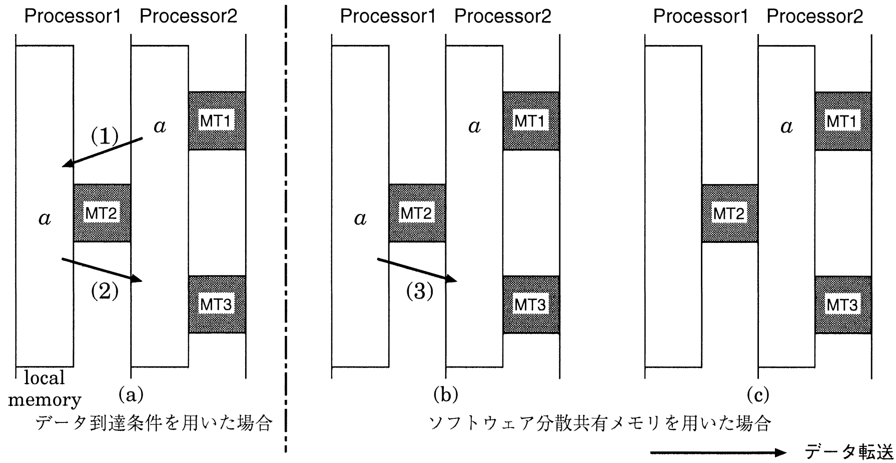


図 5 不確実な定義による kill でのデータ転送の比較
Fig. 5 Comparison of data transfer for ambiguous kill.

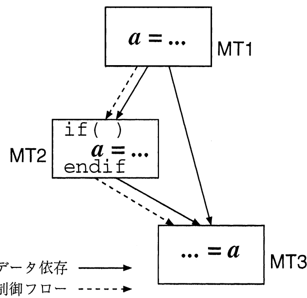


図 4 不確実な定義による kill を含むマクロフローグラフの例
Fig. 4 Macro-flow graph of ambiguous kill.

い，オーバヘッドとなる．

図 4 に不確実な定義による kill を含むマクロフローグラフの一例を示す．この例では，MT2 において変数 a への定義が (ii) の場合となる．この処理を，データ到達条件による実行方式によって 2 プロセッサ（ノード）で実行した場合のデータの流れを**図 5** の (a) に，SDSM を用いた場合のデータの流れを図 5 の (b) と (c) にそれぞれ示す．また，ここではマクロタスク MT1 と MT3 をプロセッサ 2 に，MT2 をプロセッサ 1 に割り当てたものとしている．

データ到達条件による実行の場合，図中の (1) で MT2 の入り口で a の値を定義し直すためのデータ転送を，(2) は MT3 で使用される a の値のためのデータ転送を表している．このとき，MT2 の処理で a への定義が実行されなかった場合には (1) と (2) のデータ転送は不要である．また，MT2 の処理で a への定義が実行された場合でも (1) のデータ転送は不要なものとなる．このように，不確実な定義による kill を含む場合，データ到達条件による通信管理では不要なデータ転送が生じる．

一方，SDSM による実行の場合，図中 (b) では MT2 において a の定義が実行された場合，(c) は a の定義が実行されなかった場合をそれぞれ示している．(b) の MT2 で a への定義が実行された場合，実際にデータ授受が必要となる (3) のデータ転送のみ実行し，データ到達条件での (1) のような kill される値によるデータ転送は実行されない．さらに，SDSM ではメモリに対する Read/Write を実行時に検出するため，(c) の MT2 で a への定義が実行されない場合はデータ到達条件で実行された (1) と (2) の不要なデータ転送はいっさい実行されない．

3.4.3 不確実な使用

データ到達条件による実行方式では，前述の変数への定義に関する不確実性と同様に，使用に関する不確実性も考慮する必要がある．

すなわち，(i') MTk 内の文で使用される V の値に別名がある場合，(ii') V を使用するか否かがマクロタスク内の条件分岐方向によって実行時に決定される場合，(iii') V が配列変数で，それぞれの要素を使用するか否かが実行時に決定される場合，コンパイル時に V の値を特定することや，または V が使用されるかどうかを判断することは困難である．

そこで，上記 (i') から (iii') の場合は MTk 内で使用される V となる可能性のあるすべての変数を，MTk の実行開始前に当該プロセッサに揃えることとする．ただし，揃えた変数がすべて MTk で使用されるとは限らず，使用されない変数に関するデータ転送は不要なものである．

一方，SDSM による実行方式では，マクロタスクの

実行時に SDSM システムによってデータの所在が検知されるため，必要に応じたデータ転送が行われる．これにより，データ到達条件方式での不要なデータ転送を SDSM 方式により削減することが可能となる．

4. 評 価

各ノードが**表 2** に示す構成要素の PC クラスタを用いて，提案する SDSM によるデータの一貫性制御手法を用いた方式と，データ到達条件を用いた方式との性能比較を行った．

評価に用いたベンチマークは，規則的なデータ参照パターンのものとして SPEC fp95 の swim と tomcatv を，不規則なデータ参照パターンが行われるものとして，Nas Parallel Benchmarks（NPB）の Kernel CG を用いた．プログラムのデータサイズは swim を 1009×1009，tomcatv を 513×513，CG を CLASS B とした．

SDSM による実行方式には，一貫性モデルに Lazy Release Consistency を適用し，一貫性維持プロトコルを diff 分散方式によって実装されている TreadMarks version 1.0.3.3[8] と一貫性モデルに Scope Consistency を適用し，一貫性維持プロトコルをホームベース方式によって実装されている JIAJIA version 2.2[13] を用いた．これらの SDSM は，マルチプルライタプロトコル[14] を実装しているため，false sharing の問題を回避できる．なお，これらの SDSM は配布版をそのまま利用し，システムへの変更はいっさい加えていない．一方，データ到達条件による実行方式には，MPICH-SCore 1.2.5 と Pthread を用いた．

本評価で用いた 2 つの SDSM は通信に UDP を，MPICH-SCore は通信に PM/Ethernet および PM/Myrinet を用いている．ただし，今回の実装に用いた SDSM システムは，両者とも Myrinet-2000 に対応していないため，100BASE-TX のみを使用した．そのため，データ到達条件との性能比較には 100BASE-TX の結果を用いる．また，Myrinet-2000 に関するデータは本稿で提案する SDSM による方式を直接反映するものではないが，高速ネットワークを利用した場合の指標として掲載する．

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[4),15)] を用いた．この方式では，スケジューリング用のコードとマクロタスク実行用のコードを別々に生成し，1 つのノードをスケジューラノード（**SN**）としてスケジューリングコードを専門に処理させ，残りのノードをマクロタスク実行ノード（**EN**）としてマクロタスク実行コードを処理させる．

また，マクロデータフロー処理を効率良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当てが重要である[2),16)]．そこで，データ到達条件による実行と SDSM による実行でのマクロタスク割当ては，文献 16) のパーシャルスタティック割当てを参考にして，データ共有量の多いマクロタスクを同一のノードに割り当てる方法[15] で行った．

なお，評価に用いたプログラムのマクロタスク分割や並列性検出，提案するロック操作の付加などの並列化は人手によって行っている．また，ホームベース方式による一貫性維持プロトコルである JIAJIA では，分割したマクロタスクが割り当てられたノードに対して，書き込みの局所性が最適になるようにホームノードの調整を行った．

4.1 swim による評価

swim は差分近似による浅瀬式の解を求めるもので，ループ並列性の高いプログラムである．このプログラムのメインループから呼ばれるサブルーチン CALC1, CALC2，CALC3 をインライン展開したものを使用する．マクロタスクへの分割は基本的に各ループを 1 つのマクロタスクとしたが，処理量の大きい 2 重ループは使用する EN 数で等分になるように外側ループをブロック分割した．また，ループ以外の代入文などについては隣接するループに融合して 1 つのマクロタスクとしている．

図 6 に実行時間を示す．このときの SDSM による

表 2 システム構成
Table 2 System configuration.

CPU	PentiumIII 866 MHz (dual)
メモリ	PC133 SDRAM 1 GB
NIC1	Myrinet-2000
NIC2	100BASE-TX
OS	linux kernel 2.4.21
	SCore 5.6.1
C コンパイラ	egcs-2.91.66

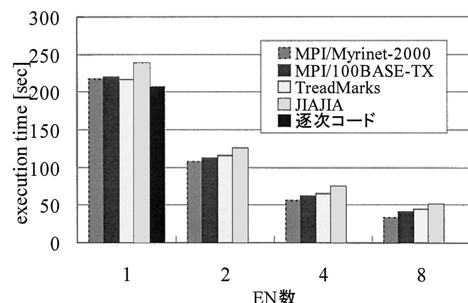


図 6 swim の実行時間
Fig. 6 Execution time of swim.

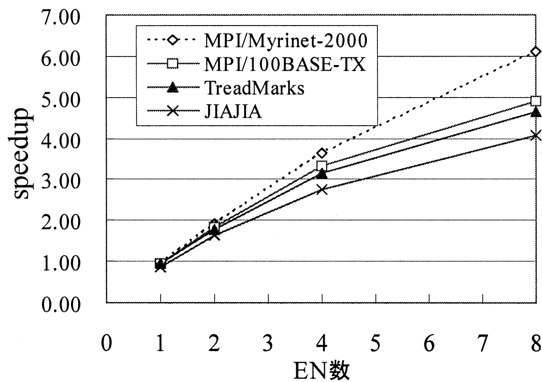


図 7　swim の速度向上率
Fig. 7　Speedup of swim.

方式の実行時間は，3.3 節で示した不要なロック操作を除去したときの結果を示す．また，以降のベンチマークプログラムも不要なロック操作を除去したときの結果を示す．EN 数が 8 のときに TreadMarks での実行時間は 44.7 秒，JIAJIA での実行時間は 50.9 秒であった．これに対し，不要なロック操作を除去しなかった場合には，TreadMarks での実行時間は 44.9 秒，JIAJIA での実行時間は 51.8 秒であった．これにより，不要なロック操作を除去することで，TreadMarks では 0.45%，JIAJIA では 1.74%の実行時間が短縮された．

図 7 に逐次コード実行に対する速度向上率を示す．EN 数が 8 のときの速度向上率は，Myrinet-2000 上のデータ到達条件による実行で 6.11，100BASE-TX 上のデータ到達条件による実行で 4.92，TreadMarks の実行で 4.62，JIAJIA の実行で 4.07 であった．100BASE-TX 上での実行において，TreadMarks を用いた場合にデータ到達条件を用いた実行と同等の性能が得られていることから，マクロデータフロー処理での SDSM の実行時オーバヘッドはさほど大きいものではないと判断できる．一方，JIAJIA の実行時間は他のものに比べて長くなっていたが，これは TreadMarks に比べてロック操作のコストが大きいことが要因としてあげられる．

図 8 に 1 ノードあたりのデータ転送量を示す．データ到達条件でのデータ転送量は MPI で送受信したアプリケーションのデータ量を計測したもので，制御のためのデータ量は含まれていない．swim では規則的な参照パターンのため，データ到達条件によるデータ転送量は TreadMarks や JIAJIA の SDSM による方式よりも少なくなっていた．

4.2　tomcatv による評価

tomcatv は 2 次元のメッシュ生成を行うプログラムであり，swim と同様にループ並列性が高い．このプログラムから生成した主要なマクロタスクは，2 重ループの外側を EN 数で等分にブロック分割したものである．外側ループで並列処理できないループについてはループ交換を行った．

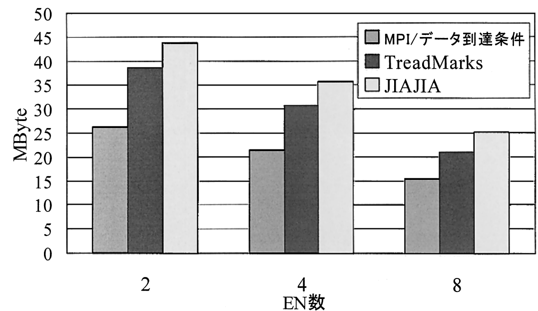


図 8　1 ノードあたりのデータ転送量：swim
Fig. 8　Amount of data transfer on swim (per node).

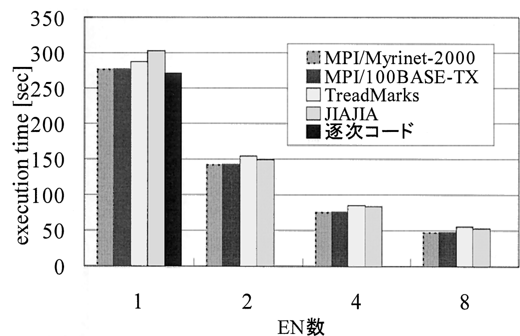


図 9　tomcatv の実行時間
Fig. 9　Execution time of tomcatv.

図 9 に実行時間を示す．EN 数が 8 のときに TreadMarks での実行時間は 55.3 秒，JIAJIA での実行時間は 52.2 秒であった．これに対し，不要なロック操作を除去しなかった場合には，TreadMarks での実行時間は 55.6 秒，JIAJIA での実行時間は 53.5 秒であった．これにより，不要なロック操作を除去することで，TreadMarks では 0.54%，JIAJIA では 2.43%の実行時間が短縮された．

図 10 に逐次コード実行に対する速度向上率を示す．EN 数が 8 のときの速度向上率は，Myrinet-2000 上のデータ到達条件による実行で 6.85，100BASE-TX 上のデータ到達条件による実行で 5.61，TreadMarks による実行で 5.01，JIAJIA による実行で 5.16 であった．

図 11 に 1 ノードあたりのデータ転送量を示す．tomcatv では規則的な参照パターンのため，swim と同様にデータ到達条件によるデータ転送量がどちらの SDSM 方式よりも少なくなっていた．一方 SDSM 方式の TreadMarks と JIAJIA を比較すると，JIAJIA

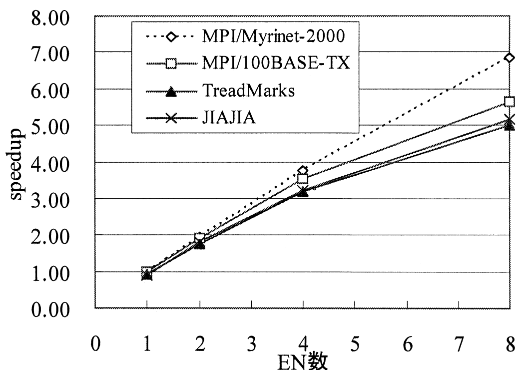


図 10 tomcatv の速度向上率
Fig. 10 Speedup of tomcatv.

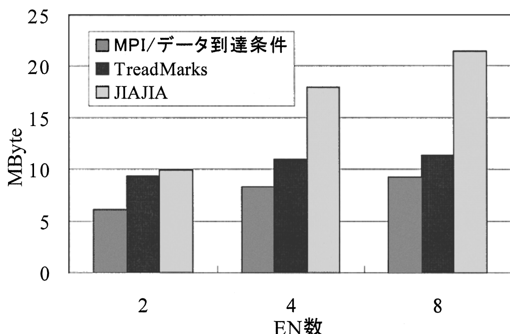


図 11 1 ノードあたりのデータ転送量：tomcatv
Fig. 11 Amount of data transfer on tomcatv (per node).

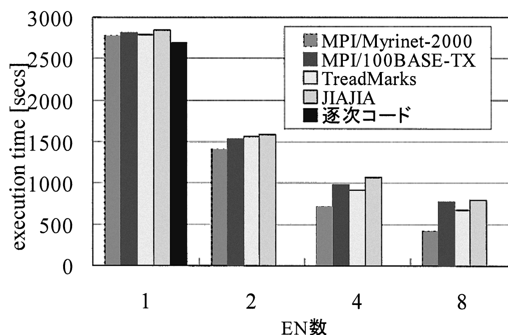


図 12 CG の実行時間
Fig. 12 Execution time of CG.

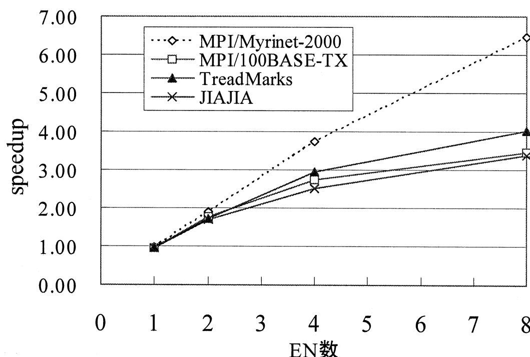


図 13 CG の速度向上率
Fig. 13 Speedup of CG.

ではリダクション演算の処理ごとにデータの更新内容をホームノードに転送するため，TreadMarks に比べてデータ転送量が大きくなっていた．しかし，tomcatv ではリダクション演算以外で他ノードとのデータ転送を行わないため，結果として，データの局所性を活かせるホームベース方式の JIAJIA の方が TreadMarks に比べて実行時間が短くなっていた．

4.3 CG による評価

先の 2 つの規則的な参照のプログラムに対し，不確実なデータ参照が含まれるプログラムの評価として NPB の CG を用いた．CG は正値対称な大規模行列の固有値を Conjugate Gradient（CG）法によって求めるものである．このプログラムのカーネル部分は，以下に示す行列とベクトルの積を求めるもので，この計算に処理時間の大部分が費やされる．

```
for(j=0; j<n; j++){
  for(k=rowstr[j]; k<rowstr[j+1]; k++){
    sum = sum + a[k] * p[colidx[k]];
  }
  q[j] = sum;
}
```

プログラムの並列化に際しては，主ループから呼ばれるサブルーチンをすべてインライン展開し，この主ループをマクロデータフロー処理するようにした．特に，カーネル部分は 2 重ループの外側を EN 数でブロック分割してマクロタスクにした．これらのマクロタスクでは a，p，colidx の各配列への（間接）参照が行われるが，これらの配列参照すべてにおいて，3.4.3 項の (iii') に示した不確実な使用が行われる．

図 12 に実行時間を示す．EN 数が 8 のときに TreadMarks での実行時間は 669 秒，JIAJIA での実行時間は 798 秒であった．これに対し，不要なロック操作を除去しなかった場合には，TreadMarks での実行時間は 671 秒，JIAJIA での実行時間は 808 秒であり，不要なロック操作を除去することで，TreadMarks では 0.30%，JIAJIA では 1.24%の実行時間が短縮された．

図 13 に逐次コード実行に対する速度向上率を示す．EN 数が 8 のときの速度向上率は，Myrinet-2000 上のデータ到達条件による実行で 6.45，100BASE-TX 上のデータ到達条件による実行で 3.47，TreadMarks による実行で 4.01，JIAJIA による実行で 3.38 であった．EN 数が 8 のときの 100BASE-TX 上での実行に

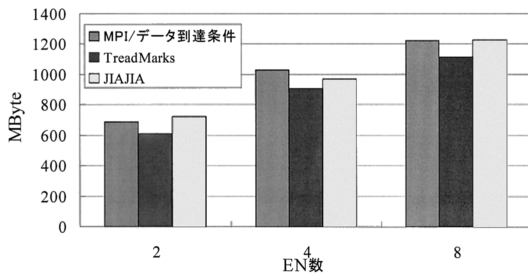


図 14 1 ノードあたりのデータ転送量：CG
Fig. 14 Amount of data transfer on CG (per node).

表 3 CG でのデータサイズ
Table 3 Data size of CG.

配列名	データサイズ (MB)
a	160
colidx	80
p	0.6
z	0.6

おいて，TreadMarks を用いた場合にデータ到達条件を用いた実行に比べて約 14%の性能向上となった．一方，JIAJIA を用いた場合ではデータ到達条件を用いた実行に比べて約 3%の性能低下となった．

図 14 に 1 ノードあたりのデータ転送量を示す．EN 数が 8 のときの 1 ノードあたりのデータ転送量を比較すると，SDSM を用いることで不確実な使用に対して不要なデータ転送が削減され，データ到達条件に比べて TreadMarks で約 9%のデータ転送量が削減された．JIAJIA では不要なデータ転送は削減されたものの，その他の各種一貫性維持の処理でデータ転送が増えてしまい，結果的にデータ到達条件のデータ転送量とほぼ変わらなかった．

CG で不確実な使用が行われる配列とそのサイズは**表 3** のとおりである．この配列のうち SDSM を用いた方式によって不要なデータ転送を削減できたのは配列 a のみであった．ただし，この a と colidx に関しては，初期化の処理以外は参照されるのみであり，データ到達条件を用いた場合でも最初のイタレーションで各 EN に転送されれば，次のイタレーションからは転送されない．

また，p と z に関してはプログラム中で繰り返し代入されるため，イタレーションごとにすべての要素のデータ転送が行われる．しかし．実際に転送されたデータはその後にすべて参照されるため，不要なデータ転送というわけではない．このため，SDSM を用いることで削減できるデータ転送は，最初のイタレーションの a の転送のみとなる．ただし，この a のデータサイズは他のデータに比べて大きいため，削減された不要なデータ転送の効果が現れたと考えられる．

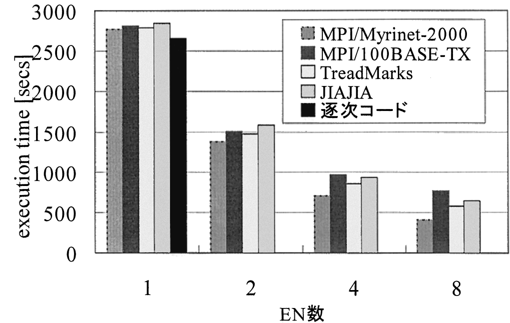


図 15 サンプルプログラムの実行時間
Fig. 15 Execution time of sample program.

4.4 サンプルプログラムによる評価

不確実な定義による kill の処理を，サンプルプログラムを用いて評価した．このサンプルプログラムは前節の CG の一部処理を，故意に不確実な定義による kill が行われるように変更したものである．そのため，プログラムとしての意味は特にない．

CG プログラムからの変更点について説明する．カーネル部分を含む主ループでは，以下のように配列 p に対する代入と参照が繰り返し行われる．このうちカーネル部分は前節に示したものである．

```
for(it=0; it<NITCG; it++){

    カーネル部分のコード

    for(i=0; i<cols; i++){
        p[i] = r[i] + beta * p[i];
    }
}
```

CG の評価では，p への代入を行うループも EN 数でブロック分割してマクロタスクとした．このマクロタスクに対して条件分岐を挿入し，最外側ループのイタレーションごとに p へ代入する処理と，代入しない処理を交互に行うようにした．具体的には，ループ制御変数 `it` が偶数のときは p への代入を行い，奇数のときは代入しないようにした．これにより，このマクロタスクでの p への代入は 3.4.2 項の (ii) の不確実な定義による kill となり，図 4 の例と同じ状況になる．

図 15 に実行時間を示す．EN 数が 8 のときに TreadMarks での実行時間は 576 秒，JIAJIA での実行時間は 641 秒であった．これに対し，不要なロック操作を除去しなかった場合には，TreadMarks での実行時間は 581 秒，JIAJIA での実行時間は 664 秒であり，不要なロック操作を除去することで，TreadMarks では

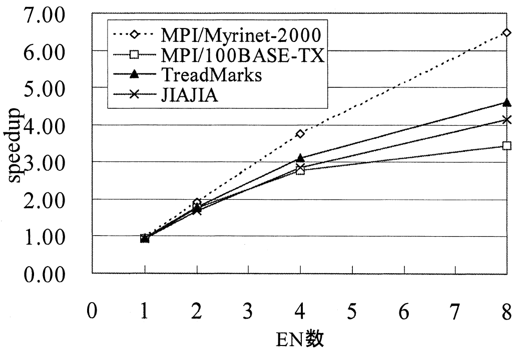


図 16 サンプルプログラムの速度向上率
Fig. 16 Speedup of sample program.

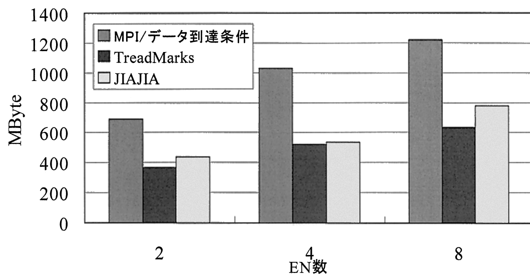


図 17 1 ノードあたりのデータ転送量：サンプルプログラム
Fig. 17 Amount of data transfer on sample program (per node).

0.86%，JIAJIA では 3.46%の実行時間が短縮された．

図 16 に逐次コード実行に対する速度向上率を示す．EN 数が 8 のときの速度向上率は，Myrinet-2000 上のデータ到達条件による実行で 6.49，100BASE-TX 上のデータ到達条件による実行で 3.46，TreadMarks による実行で 4.61，JIAJIA による実行で 4.14 であった．

図 17 に 1 ノードあたりのデータ転送量を示す．EN 数が 8 のときの 1 ノードあたりのデータ転送量を比較すると，実際に配列へ代入されなかった場合は図 5 (c) の状況となり，SDSM を用いることで不確実な定義による kill での不要なデータ転送がなくなった．そのため，データ到達条件に比べて TreadMarks では約 49%，JIAJIA では約 37% のデータ転送量が削減された．その結果，EN 数が 8 のときの 100BASE-TX 上でのデータ到達条件を用いた実行に比べて，TreadMarks を用いた場合に約 25%，JIAJIA を用いた場合に約 16%の性能向上となり，提案手法の有効性を確認できた．

また，このサンプルプログラムでは p への不確実な定義による kill が行われる前に p が参照されているため，このマクロタスクを実行するプロセッサへはデータ到達条件が成立している値が転送されている．よって，図 5 の (1) のような定義をし直すためのデータ転送は実行されない．そのため，もしデータ到達条件が成立する値が存在しない状況ならば，提案手法によって，さらに不要なデータ転送の削減が期待できる．

このサンプルプログラムは作為的な例によるものであったが，たとえば，ピボッティング付きの LU 分解などは，インデックス配列を介して配列の各要素を定義するため，3.4.2 項の (iii) の不確実な定義による kill となる．このようなアプリケーションにおいても提案手法が有効であると考えられる．

5. おわりに

本稿では，ソフトウェア分散共有メモリ（SDSM）を用いてマクロデータフロー処理を実現するためのデータの一貫性制御手法を提案した．提案手法は，マクロタスク間のデータ依存関係と SDSM のロック操作による一貫性維持の枠組みを利用することで，マクロタスク間のデータ授受を行う．また，LRC といった一般的な SDSM の一貫性モデルを対象としているために，分散メモリシステム上でのマクロデータフロー処理に利用できる既存の SDSM システムの適用範囲が広いという特徴がある．

本稿ではまた，SDSM を利用しないデータ到達条件を用いた実行方式との性能比較を行った．データ到達条件を用いた実行方式では静的な依存解析によって明示的なデータ授受を行っているため，十分に依存解析が行える規則的な参照パターンのプログラムでは，SDSM 方式よりも効率的なデータ授受を実現できることが明らかになった．

一方，静的な依存解析ができない不規則な参照パターンのプログラムに対して，データ到達条件を用いた実行方式では上述の最適化が行えず通信管理のためのオーバヘッドが大きくなることや，不確実なデータ参照に対しては必要以上のデータ転送を行ってしまう．このようなプログラムに対し，本稿で提案した SDSM による実行方式を用いることにより，不要なデータ転送が削減され効率的にデータを授受できることが明らかになった．

今後の課題として，未実装であった SDSM の Myrinet-2000 への実装を進め，高速ネットワーク上での評価や，キャンパスグリッド規模のより広域環境での評価があげられる．また，逐次プログラムから提案方式やデータ到達条件を用いた実行方式のコードを生成するコンパイラの開発を進め，これらを用いて様々なプログラムで評価し，分散メモリシステム上でのマクロデータフロー処理の有効性を検証するととも

に，他の並列化手法であるクラスタ上での OpenMP コンパイラ[17] などとの性能比較も必要であると考えている．

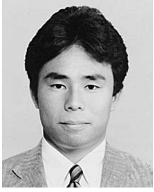
謝辞 本研究の一部は科学研究費（基盤研究（C）16500025）と財団法人大川情報通信基金（助成番号03-13）によるものである．

参 考 文 献

1) 小幡元樹，白子 準，神長浩気，石坂一久，笠原博徳：マルチグレイン並列処理のための階層的並列性制御手法，情報処理学会論文誌，Vol.44, No.4, pp.1044–1055 (2003).
2) 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).
3) 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).
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：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. and Tseng, C.-W.: Enhancing Software DSMs for Compiler-Parallelized Applications, *Proc. 11th Int'l Parallel Processing Symp.* (*IPPS'97*) (1997).
6) 丹羽純平，松本 尚，平木 敬：ソフトウェアDSM 機構を支援する最適化コンパイラ，情報処理学会論文誌，Vol.42, No.4, pp.879–897 (2001).
7) 佐藤茂久，草野和寛，佐藤三久：OpenMP 向けコンパイラ支援ソフトウェア DSM，情報処理学会論文誌，Vol.42, No.4, pp.788–801 (2001).
8) Keleher, P., Dwarkadas, S., Cox, A. and Zwaenepoel, W.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, *Proc. Winter 94 Usenix Conference*, pp.115–131 (1994).
9) Iftode, L., Jaswinder, Singh, P. and Li, K.: Scope Consistency: A Bridge between Release Consistency and Entry Consistency, *Proc. 8th ACM Annual Symp. on Parallel Algorithms and Architectures* (*SPAA'96*), pp.277–287 (1996).
10) Ferrante, J., Ottenstein, K.J. and Warren, J.D.: The program dependence graph and its use in optimization, *ACM Trans. Prog. Lang. Syst.*, Vol.9, No.3, pp.319–340 (1987).
11) 本多弘樹，合田憲人，岡本雅巳，笠原博徳：Fortran プログラム粗粒度タスクの OSCAR における並列実行方式，電子情報通信学会論文誌，Vol.J75-D1, No.8, pp.526–535 (1992).
12) Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers: Principles, Techniques and Tools*, Addison-Wesley (1988).
13) Hu, W., Shi, W. and Tang, Z.: JIAJIA: An SVM System Based on A New Cache Coherence Protocol, *Proc. High Performance Computing and Networking* (*HPCN'99*), pp.463–472 (1999).
14) Amza, C., Cox, A.L., Dwarkadas, S. and Zwaenepoel, W.: Software DSM Protocols that Adapt between Single Writer and Multiple Writer, *Proc. 3rd IEEE Symp. on High-Performance Computer Architecture* (*HPCA-3*), pp.261–271 (1997).
15) 田邊浩志，本多弘樹，弓場敏嗣：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，情報処理学会研究報告，No.27 (ARC-152:HOKKE-2003), pp.37–42 (2003).
16) 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).
17) 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).

(平成 16 年 7 月 23 日受付)
(平成 16 年 11 月 9 日採録)

田邊 浩志（学生会員）
2002 年電気通信大学情報工学科卒業．2004 年同大学大学院情報システム学研究科修士課程修了．現在，同大学院博士後期課程在学中．分散メモリシステムにおける粗粒度並列処理の研究に従事．



本多 弘樹（正会員）

1984年早稲田大学理工学部電気工学科卒業．1991年同大学大学院理工学研究科博士課程修了．1987年より同大学情報科学研究教育センター助手．1991年より山梨大学工学部電子情報工学科専任講師．1992年より同助教授．1997年より電気通信大学大学院情報システム学研究科助教授．並列処理方式，並列化コンパイラ，並列計算機アーキテクチャ，グリッド等の研究に従事．工学博士．電子情報通信学会，IEEE-CS，ACM各会員．平成15年度山下記念研究賞受賞．

弓場 敏嗣（フェロー）

1966年神戸大学大学院工学研究科修士課程修了．（株）野村総合研究所を経て，1967年通商産業省工業技術院電子技術総合研究所（現在，独立行政法人産業技術総合研究所）に入所．以来，計算機のオペレーティングシステム，見出し探索アルゴリズム，データベースマシン，データ駆動型並列計算機等の研究開発に従事．その間，計算機方式研究室長，知能システム部長，情報アーキテクチャ部長等を歴任．1993年より，電気通信大学大学院情報システム学研究科教授．並列処理・分散処理の科学技術一般に興味を持つ．工学博士．電子情報通信学会フェロー，日本ソフトウェア科学会，日本ロボット学会，ACM，IEEE各会員．