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

2004－ARC－157　(18)
2004－HPC－ 97　(18)
2004／3／2

PC クラスタ上でのマクロデータフロー処理の評価

田 邊 浩 志† 本 多 弘 樹† 弓 場 敏 嗣†

PC クラスタなどの分散メモリシステム上での並列処理方式として，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．我々は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方式として，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と，ソフトウェア分散共有メモリによる実行方式を提案してきた．本稿ではそれらを PC クラスタ上に実装し，性能評価をした結果について報告する．

Evaluation of Macro-Data-Flow on a PC cluster

Hiroshi Tanabe ,† Hiroki Honda † and Toshitsugu Yuba †

On distributed memory systems such as PC clusters, macro-dataflow processing, which exploits a parallelism among coarse grain tasks (macro-tasks), is considered promising to break the performance limits of loop parallelism. To realize macro-dataflow processing on distributed memory systems, two methods have been proposed. One is the “data reaching conditions”, a method to make a sender-receiver pair of a data transfer determined at runtime, and the other uses software distributed shared memory systems. This paper gives comparison results of the two methods on a PC cluster.

1. はじめに

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レベルの並列性を利用するマクロデータフロー処理が，共有メモリシステム上で高い実効性能を与えることが報告されている[1),2)]．

一方，近年は PC クラスタが並列処理環境として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．しかし，従来の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のデータ授受が新たに問題となる．

この問題を解決する手法として，我々は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式[3)] と，SDSM を用いずに明示的なメッセージ通信をおこなうデータ到達条件による方式[4)] をそれぞれ提案してきた．

両者の特徴を比較すると，データ到達条件による実行方式は，ページベースの SDSM のように一貫性維持処理を全て実行時におこなうのではなく，データ依存解析によって可能な限り静的におこなうようにしている．これにより，冗長な一貫性維持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パターンがある場合に一貫性維持のための検査が大きなオーバヘッドとなったり[5),6)]，または参照されるデータがコンパイル時にはわからず，実行時になって確定する (以降，不確実なデータ参照と呼ぶ) 場合に，不要なデータ転送をしてしまうことがある．

これに対し，ページベースの SDSM を用いた場合，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．反面，ページ単位での一貫性維持のためのオーバヘッドにより，性能低下が危惧される．

そこで，本稿ではこれらの違いを検証するために，データ到達条件による実行方式と SDSM による実行方式によるマクロデータフロー処理を PC クラスタ上に実装し，性能比較をおこなった．

† 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.

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．

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[7)] 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．

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

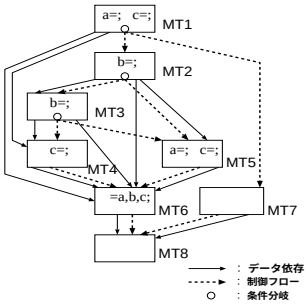

図 1 マクロフローグラフの例

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[1]．集中型では特定のプロセッサがスケジューリングコードを実行するのに対し，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，全てのプロセッサがスケジューリングコードを実行する．

3. 分散メモリシステム上でのマクロデータフロー処理

3.1 マクロタスク間でのデータ授受

分散メモリシステム上でのマクロデータフロー処理においては，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，あるマクロタスクで変数 V 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，実行時に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という仕組みが必要となる．

3.2 データ到達条件による実行方式

データ到達条件[4] とは，マクロタスク MTi(の先頭) に到達する[8] 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 する定義を持つ全てのマクロタスクの集合を K としたとき，以下の式のように定義する．

[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]

[MTj の実行確定条件] は，MTj を実行することが確定するための条件で，MTj が制御依存[9] するマクロタスクから MTj 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．

[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[7] 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．

並列実行中に MTi での V の使用に関するデータ到達条件が成立するのは S_V^i 中のただ 1 つのマクロタスクとなる．これにより，実行時にデータ到達条件を検査することでデータ授受が必要なマクロタスクを特定し，スケジューラからデータ転送を指示することによってマクロタスク間のでのデータ授受を実現する．

3.3 SDSM による実行方式

SDSM による実行方式は，緩い一貫性モデルである Lazy Release Consistency(LRC)[10] と Scope Consistency(ScC)[11] を対象としている．これらの一貫性モデルでは一貫性の維持をページ単位でおこない，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ものである．

SDSM による実行方式では，SDSM のロック変数に付加されるページの更新情報 (write notice) を利用し，マクロタスク間のデータ授受を実現する．そこで，コンパイル時には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．

(1) マクロタスク MTi に対して，ユニークなロック変数 L_{MT_i} を割り当てる．
(2) MTi の先頭に L_{MT_i} を獲得するコードを挿入し，MTi の最後には L_{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 の入り口で，MTi がデータ依存するマクロタスク MTj のロック変数 L_{MT_j} を獲得と解放をおこなうコードを挿入する．

MTi を実行する前に L_{MT_j} を獲得することで L_{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．

3.4 データ到達条件による実行方式と SDSM による実行方式の比較

3.4.1 動 作 概 要

従来の共有メモリシステム上で実行方式とともに，両方式の違いを**表 1** にまとめる．また，両方式ではマクロタスク間のデータ授受とマクロタスクの実行を開始できるタイミングが異なる．この違いを**表 2** にまとめる．

3.4.2 不確実な定義による kill

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 への定義がおこなわれるとは，MTk 内で必ず V への**確実な定義**[8] がおこなわれることを前提と

表 1 実行方式

	コンパイラ	スケジューラ
従来方式 (共有メモリシステム)	・実行開始条件の導出 ・スケジューリングコード生成	・実行開始条件の検査 ・マクロタスク割り当て
SDSM による方式	従来方式 ＋ ・データ一貫性制御のための ロック操作コード生成	従来方式と同様
データ到達条件 による方式	従来方式 ＋ ・データ到達条件の導出 ・データ転送指示コードの生成	従来方式 ＋ ・データ到達条件の検査 ・データ転送の指示

表 2 データ授受とマクロタスクの実行開始のタイミング

	データ授受	マクロタスクの実行開始
SDSM による方式	マクロタスクの実行中 (必要に応じて)	実行開始条件成立後
データ到達条件 による方式	マクロタスクの実行開始前	実行開始条件成立後 ＋ データ到達条件が成立 したデータがローカル メモリ上に揃ってから

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りうる．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，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 へ定義し直すこととすれば，MTk で確実な定義がおこなわれるようにできる．ただし，この方法では実際には kill されてしまう定義による値のデータ転送をおこなってしまい，オーバヘッドとなる．

図 2 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．

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転送の流れを**図 3** の (a) に示す．ここでは，マクロタスク MT1 と MT3 をプロセッサ 2 に割り当て，MT2 をプロセッサ 1 に割り当てたものとする．

図中の (1) は 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た場合には (1) のデータ転送は不要な通信となる，また，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．

一方，SDSM によるデータ転送は図 3 の (b) と (c) になる．(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一切実行されない．

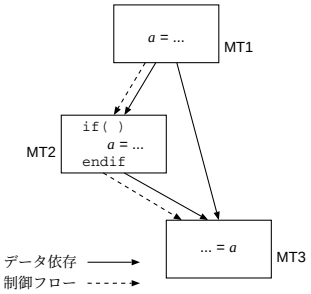

図 2 不確実な定義による kill を含むマクロフローグラフの例

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全ての変数を，MTk の実行開始前に当該プロセッサに揃えることにする．ただし，揃えた変数が全て 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となる．

一方，SDSM による方式では，マクロタスク実行時に 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おこなわれる．これにより，データ到達条件方式ではおこなわれてしまう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

4.1 評 価 環 境

データ到達条件による実行方式と SDSM による実行方式を，**表 3** に示す構成要素を持つ 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[3),4)] で実装した．このスケジューリング方式は，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コードを処理させる．

データ到達条件の実装には MPI と Pthread を用い，SDSM による実装には一貫性モデルに Lazy Re-

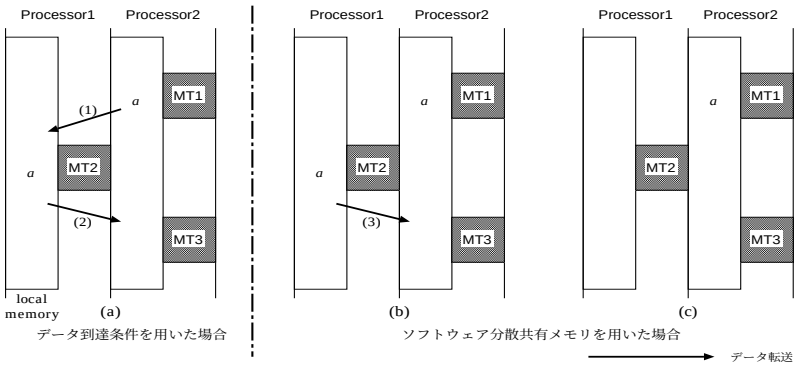

図 3 不確実な定義による kill でのデータ転送の比較

表 3 システム構成

CPU	PentiumIII 866 MHz (×2)
メモリ	1GB
NIC1	Myrinet-2000
NIC2	100BASE-TX
OS	SCore 5.6.1
	Linux kernel 2.4.18
MPI	MPICH-SCore
C コンパイラ	egcs 1.1.2

lease Consistency を適用している TreadMarks version 1.0.3.3[12] と一貫性モデルに Scope Consistency を適用している JIAJIA version 2.1[13] を用いた．ただし，今回の実装に用いた 2 つの SDSM システムは Myrinet-2000 に対応していないため，100BASE-TX のみを使用した．

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のものとして，Nas Parallel Benchmarks(NPB) の 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．

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割り当てが重要である[1),14)]．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り当ては，文献 14) のパーシャルスタティック割り当てを参考にして，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[3)] でおこなった．

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おこなっている．

4.2 swim による評価

swim のメインループから呼ばれるサブルーチンの CALC1，CALC2，CALC3 をインライン展開したプログラムを使用する．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 4 の折れ線グラフに対逐次の速度向上率，棒グラフ

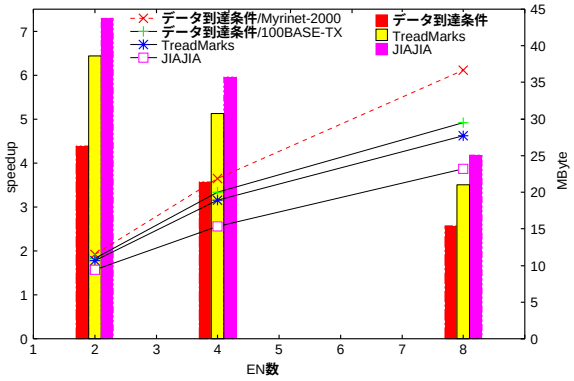

図 4 swim の速度向上率と 1 ノード当たりのデータ転送量

に 1 ノード当たりのデータ転送量を示す．EN 数が 8 の時の速度向上率は，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 3.71 であった．

1 ノード当たりのデータ転送量をみると，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実行方式は，TreadMarks や JIAJIA の SDSM による実行方式よりもデータ転送量が少なく，効率的なデータ授受が実現できていた．

4.3 tomcatv による評価

tomcatv の収束ループの内側をマクロデータフロー処理するようにした．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おこなった．

図 5 に対逐次の速度向上率と 1 ノード当たりのデータ転送量を示す．EN 数が 8 の時の速度向上率は，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 4.35 であった．

1 ノード当たりのデータ転送量をみると，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実行方式でのデータ転送量が SDSM 方式よりも少なくなっている．一方 swim とは異なり，EN

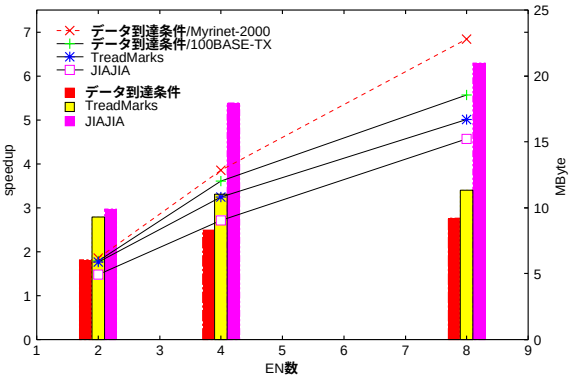

図 5 tomcatv の速度向上率と 1 ノード当たりのデータ転送量

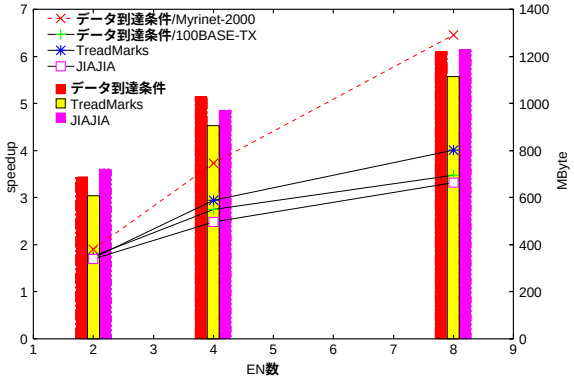

図 6 CG の速度向上率と 1 ノード当たりのデータ転送量

数を増やすことでノード間のデータ転送量が増加している．これは，主ループの 1 イタレーションではリダクション演算のみ他ノードとデータ転送するため，EN 数に比例してデータ転送量が増えたためである．また，JIAJIA 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．

4.4 CG による評価

先の 2 つの規則的なデータ参照のプログラムに対し，不確実な使用が含まれるプログラムの評価として NPB の CG を用いた．このプログラムでは間接参照による不確実な使用がおこなわれる．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全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．

図 6 に対逐次の速度向上率と 1 ノード当たりのデータ転送量を示す．EN 数が 8 の時の速度向上率は，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 3.32 であった．EN 数が 8 の時の 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による実行方式に比べて 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による実行方式に比べて 5%の性能低下となった．

EN 数が 8 の時の 1 ノード当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 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が削減されたものの，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 での不確実な使用がおこなわれる配列は**表 4** のとおりであり，この配列のうち SDSM による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プログラム中で参照されるのみであり，データ到達条件による実行方式の場合でも最初のイタレーションでデータが転送されれば，次のイタレーションからはデータ転送されない．

表 4 不確実な使用がおこなわれる配列とそのデータサイズ

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

また，p と z に関してはプログラム中で繰り返し定義さるため，イタレーション毎に全ての要素のデータ転送がおこなわれる．しかし．実際に転送されたデータはその後に全て参照されるため，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量の効果が表われたと考えられる．

4.5 サンプルプログラムによる評価

サンプルプログラムを用いて，不確実な定義による kill の処理について評価した．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 がおこな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 のプログラムからの変更点について説明する．前節において，不規則参照される配列では配列 p と z がイタレーション毎に繰り返し定義と使用されることを説明した．このうち p への定義をおこなうループに対し，条件分岐によって隔イタレーション毎に実際に定義をする処理と，定義をしない処理を繰り返すようにした．これにより，p への定義は擬似的に図 2 の例のような不確実な定義となる．

この変更したプログラムの対逐次の速度向上率と 1 ノード当たりのデータ転送量を**図 7** に示す．EN 数が 8 の時の速度向上率は，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 4.13 であった．

EN 数が 8 の時の 1 ノード当たりのデータ転送量を比較すると，実際に配列へ定義されなかった場合は図 3(c) の

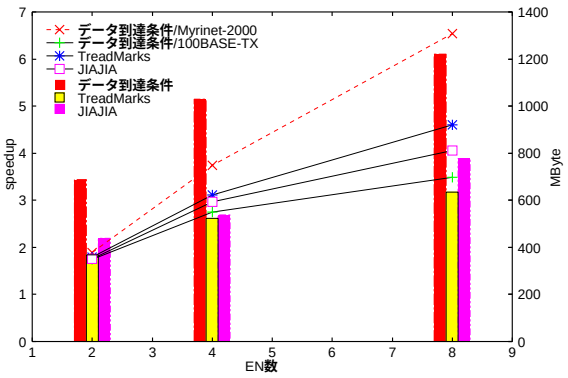

図 7 サンプルプログラムの速度向上率と 1 ノード当たりのデータ転送量

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．このため．データ到達条件に比べて TreadMarks では 49%，JIAJIA では 37% のデータ転送量が削減された．

この結果，EN 数が 8 の時の 100BASE-TX 上でのデータ到達条件による実行方式に比べて，TreadMarks を用いた場合に 25%，JIAJIA を用いた場合に 16%の性能向上となった．

5. おわりに

本稿では，分散メモリシステム上でのマクロデータフロー処理において，SDSM による実現方式と，SDSM を利用せずデータ到達条件による実行方式をそれぞれ PC クラスタ上に実装し，性能比較をした．

データ到達条件による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おこなっているため，十分に依存解析がおこなえる規則的な参照パターンのプログラムでは，冗長な通信の削除や通信の集合化などの最適化が可能となり，SDSM 方式よりも効率的なデータの授受を実現できることがわかった．

一方，静的な依存解析ができない不確実なデータ参照のプログラムに対しては，データ到達条件による実現方式では必要以上にデータ転送を増大させてしまい，性能が低下することがある．これに対し，SDSM による実行方式を用いることで不要なデータ転送を削減でき，効率的なデータ授受を実現できることがわかった．

今後は，より詳細な評価をおこなうことでマクロデータフロー処理に適する SDSM を検討することや，データの一貫性制御手法の改良などが課題である．例えば，tomcatv の評価ではホームベース型の SDSM である JIAJIA がホームレス型の TreadMarks に比べてデータ転送量が増大していた．これに対しては権限委譲プロトコル[15] を用いることでホームベース型の SDSM でも不要なデータ転送を削減できると考えられる．

また，逐次プログラムからデータ到達条件による方式と SDSM による方式のコードを自動的に生成するコンパイラの開発と，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ため，他の並列化手法であるクラスタ上での OpenMP コンパイラ[16] などとの性能比較をおこなっていく予定である．

謝辞 本研究の一部は財団法人大川情報通信基金 (助成番号 03-13) によっておこなわれた．

参 考 文 献

1) Ishizaka, K., Obata, M. and Kasahara, H.: Coarse Grain Task Parallel Processing with Cache Optimization on Shared Memory Multiprocessor, *Proc. of 14th International Workshop on Languages and Compilers for Parallel Computing* (2001).
2) 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).
3) 田邊，本多，弓場: 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).
4) 本多, 上田, 深川, 弓場: 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. of the 11th Int'l Parallel Processing Symp.* (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) 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).
8) Aho, A. V., Sethi, R. and Ullman, J. D.: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley (1988).
9) Ferrante, J., Ottenstein, K.J. and Warren., J.D.: The program dependence graph and its use in optimization, *ACM Transactions on Programming Languages and Systems*, Vol. 9, No. 3, pp. 319–340 (1987).
10) 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*, pp. 13–21 (1992).
11) 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*, pp. 277–287 (1996).
12) Keleher, P., Dwarkadas, S., Cox, A. and Zwaenepoel, W.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, *Proc. of the Winter 94 Usenix Conference*, pp. 115–131 (1994).
13) Hu, W., Shi, W. and Tang, Z.: JIAJIA: An SVM System Based on A New Cache Coherence Protocol, *Proc. of the High Performance Computing and Networking*, pp. 463–472 (1999).
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ーゼーション手法, 情報処理学会論文誌, Vol. 35, No. 9, pp. 1848–1860 (1995).
15) 城田祐介, 吉瀬謙二, 本多弘樹, 弓場敏嗣: ホームベースソフトウェア分散共有メモリ上で Migratory Access を効率良く処理する権限委譲プロトコル, 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. 44, No. SIG1(HPS 6), pp. 103–113 (2003).
16) 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).