細粒度通信機構を持つ並列計算機 EM-X における共有メモリプログラムの効率的実行

坂根広史[†,††] 本多弘樹[††] 弓場敏嗣[††]
児玉祐悦[†] 山口喜教[†,†††]

分散メモリ型並列計算機 EM-X は，グローバルポインタと細粒度通信によるリモートメモリアクセスを用いることにより，システム全体のメモリを分散共有メモリとして扱うことができる．共有メモリプログラムの実行において高速な通信起動とマルチスレッド実行によるレイテンシ隠蔽の効果を検証するため，ベンチマークプログラムを EM-X 上で実行した．マルチスレッドの効果は見られたが，共有メモリアクセスの頻度が高いプログラムでは台数効果が限定的であった．この主な原因はマルチスレッド実行におけるスレッド切替えのオーバヘッドであることが分かった．次に，スレッド切替え抑制の観点から共有データのコピーをローカルメモリに置き，ソフトウェアにより一貫性を制御する機構を実装して改善を試みた．その結果，高頻度の共有アクセスを行うプログラムに対して注意深く最適化を施した場合に，マルチスレッドを併用したリモートメモリアクセスを用いる方式よりも高い性能が得られた．共有メモリアプリケーションを効率的に実行するためには，両方式のトレードオフを考慮し，アプリケーションに応じて適切な選択を行うことが重要であることが分かった．

Efficient Execution Technique of Shared Memory Programs on Distributed Memory Parallel Computer EM-X Using Fine Grain Communication Mechanism

Hirofumi Sakane,[†,††] Hiroki Honda,[††] Toshitsugu Yuba,[††]
Yuetsu Kodama[†] and Yoshinori Yamaguchi[†,†††]

In this paper, we discuss efficient parallel execution of shared memory programs on a physically distributed memory multiprocessor EM-X. The EM-X provides shared memory abstraction with a global address space and a remote memory access mechanism. For this approach, multithreading efficiently hides the latency caused by fine-grain communication, while the thread switching overhead still remains. To reduce the thread switching overhead and exploit locality of shared data, we have implemented a coherent local copy mechanism by software. Performance analyses show that a highly optimized implementation for a frequently shared access program greatly improves the performance, in spite of additional software overhead. We show that the tradeoffs between these two approach provide a basis for the selection of a technique that is more appropriate for efficient executions of various applications on the EM-X.

1. はじめに

本論文は，分散メモリ型並列計算機 EM-X において共有メモリプログラムを効率的に実行させることを目的とし，リモートメモリアクセスを用いる方法と，共有データのローカルコピーを用いる方法を比較する．その結果として，プログラムごとの特性に合った実行方法の選択が高い性能を得るために重要であることを示し，より優れたアーキテクチャの設計指針について議論する．

分散メモリ型アーキテクチャはハードウェア構成のスケーラビリティに優れ，超並列計算機向きである．一方，ソフトウェア開発の観点からは，その扱いやすさから共有メモリが望ましいとされ，これらを両立する分散共有メモリ（DSM）の研究が活発に行われている．ほとんどの DSM システムでは，データアクセスの局所性を有効利用するために共有データのコピーを必要

† 電子技術総合研究所情報アーキテクチャ部
Computer Science Division, Electrotechnical Laboratory
†† 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
††† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

に応じて各プロセッサのローカルメモリに置き，それらの一貫性を維持する方法をとる．専用ハードウェアやコプロセッサにより一貫性を維持するシステム[1)~3)]や，ソフトウェアによって制御するSDSM（Software DSM）[4)~9)]はその実現例である．

一方我々は，将来の超並列計算機に適したプロセッサアーキテクチャを視野に入れ，演算と通信を密に融合させた分散メモリ型並列計算機EM-Xをテストベッドとして，高性能と汎用性を備えた並列システムの研究を行っている[10)]．EM-Xでは明示的なメッセージ通信だけでなく，共有メモリプログラミングが可能である．この共有メモリは，システム全体のメモリを単一のアドレス空間として扱うグローバルアドレスを用いて，分散配置された共有データをリモートメモリアクセスすることにより実現している．この方法ではローカルコピーを作らないため，共有メモリアクセスごとに細粒度通信が発生する．一般に細粒度通信では，通信起動時間と通信レイテンシがオーバヘッドとして問題となるが，EM-Xでは，前者は要素プロセッサ（PE）内に演算パイプラインと直結した通信機構を備えることで大幅な削減に成功し，後者はマルチスレッドで隠蔽可能である[11),12)]．本論文では，このリモートメモリアクセスによって共有メモリを提供する方式をNo Local Copy（NL）と表記する．

最初に，予備評価としてNLに基づいた方法により共有メモリプログラムを並列実行させ，性能を測定する．解析の結果，スレッド切替え時のオーバヘッドが大きく性能を制限し，さらにそのオーバヘッドの主因はコンテクスト待避処理であることを明らかにする．このコンテクスト待避オーバヘッドを削減するアプローチとしては，1) 1回のコンテクスト待避にかかる時間を短縮する方法と，2) リモートメモリアクセス回数を減らしてスレッド切替えを抑制する方法が考えられる．1) はレジスタセット数の増設やレジスタパーティショニング技法の研究[13)]等があるが，それぞれハードウェアコストの問題や実効レジスタ数の減少等の問題がある．本論文では2) に焦点を当て，リモートメモリアクセスを減らすために，一般的なDSMのように共有データのローカルコピーを用いる方法を導入する．目的の共有データがローカルメモリ上にあるならば，スレッドを切り替える必要がない．期待される効果はスレッド切替えオーバヘッドの回避だけでなく，局所性の利用による通信量の削減も図ることができる．EM-Xはキャッシュや MMU 等，コピーの生成や一貫性制御をサポートするハードウェアを持たないため，必要な機能はソフトウェアで実装する．この方式を本論文ではCoherent Local Copy（CL）と表記する☆．CLでは，ソフトウェアによる実装に起因する新たなオーバヘッドが加わるが，最適化を施したうえでNLのオーバヘッドより小さく抑えることができれば性能向上が期待できる．また，CLの実装上の最適化技法とその性能評価を考察することにより，将来のアーキテクチャ設計の指針を得ることができる．

NLとCLの評価結果を比較すると，プログラムによって性能の高い方式が異なることが分かる．これは，共有メモリアクセスの種々のパターンに対する振舞いが両者で異なることを示し，プログラムの性質と性能の関係を調べることにより，プログラムに応じた適切な方式を選択することが可能となる．

本論文の構成は以下のとおりである．2章でEM-Xの概要とNLの動作について述べ，3章でCL機構の実装方法について説明する．4章でベンチマークプログラムと実験方法について説明し，5章に実験結果を示す．6章で考察と課題，および関連研究について述べ，7章で総括を述べる．

2. 並列計算機 EM-X

電子技術総合研究所で開発されたEM-X[10)]は，その要素プロセッサ（PE: EMC-Y）の内部に演算パイプラインと直結した通信機構を持つことにより，データ駆動モデルに基づく命令スレッドの起動や細粒度通信によるリモートメモリアクセスを効率良く実現している．EM-Xの主要な諸元と通信パラメータを**表1**に示す．

EM-Xの並列プログラミング環境としてはEM-C[14)]が用意されている．ライブラリによるメッセージ通信を提供するほか，共有メモリプログラミングやマルチスレッド実行をサポートする．

2.1 リモートメモリアクセス

EM-Xの通信パケットはアドレス部とデータ部の2ワードからなる固定長であり，リモートメモリアクセスやスレッドの起動を担う．アドレス部はシステム全体の任意のメモリを一意に指定できるグローバルなアドレスを保持できるほか，パケットの機能を表すタグを持つ．データ部には通常のデータのほか戻り先アドレス（continuation）を収めることができ，リモートメモリアクセスの単純化と効率化に寄与している．

☆ これは一般のDSMあるいはSDSMと多くの共通点を持つ方式であるが，一方のNLも，distributeされたshared memoryのアクセスを通信命令で起動するという意味ではソフトウェアで制御される広義のDSMといえる．混乱を避けるために，EM-X上の実装を示す用語としてはDSMの代わりにCLを用いる．

表 1 EM-X の主な諸元と通信パラメータ
Table 1 Machine and communication parameters of EM-X.

プロセッサ（PE）台数	80
クロック周波数	16 MHz
メモリ	1 Mwords/PE, 38 bits/word
命令実行サイクル	1 CPI
浮動小数点演算精度	32 bit (single)
ローカルメモリレイテンシ	1 clock
ネットワーク最大スループット	2 clocks/packet = 8 Mwords/sec/port
リモート書込み発行サイクル	1 clock
リモート読出し発行サイクル	2 clocks
自 PE 内リモート読出しレイテンシ☆	9 clocks
PE 間リモート読出しレイテンシ☆☆	20 clocks

リモートメモリアクセスパケットが到着した PE では，実行中の命令を中断することなく，通信処理ハードウェアが直接メモリをアクセスする[15]．

リモート書込み（SYSWR）の発行に要する時間は 1 クロックで，リクエストを出した PE はすぐに次の処理を進めることができる．リモート読出し（SYSRD）の発行には，continuation 生成を含めて 2 クロックかかる．パケットが到着した目的 PE では，データを持ったパケットを continuation に向けて返送する．データが帰ってくるまでの時間は通信レイテンシとして観測される．帰着したデータパケットは，サスペンドしていたスレッドを直接起動する．

2.1.1 共有メモリプログラミング

EM-C コンパイラは，PE-ID とローカルアドレスの組合せによって，システム全体のメモリを単一の共有メモリとして扱えるグローバルポインタをサポートしており，これによって共有メモリプログラムを実行することができる[11),14)]．すべての共有メモリアクセスはコンパイル時にリモートメモリアクセスのためのパケット送出命令に変換され，実際に通信を発生する．この方法では共有データのローカルコピーは用いない（No Local Copy; NL）．

2.1.2 マルチスレッド

EM-C コンパイラはマルチスレッド実行をサポートしており，PE 内で複数のスレッドを切り替えながら実行することによりリモート読出しのレイテンシを隠蔽可能である[11),12)]．ただしスレッド切替えには数クロックの時間がかかり，オーバヘッドとして性能に影響を及ぼす．スレッド切替えオーバヘッドのうち，ハードウェア上のオーバヘッドは 1 クロックであるのに対し，ソフトウェア上のオーバヘッドは一般にそれ以上となる．そのうち特にコンテクスト待避にかかる時間が大きな割合を占める．スレッド切替え時にはコンテクストを保持するレジスタをメモリへ待避する必要があり，スレッド切替え箇所とセーブ・リストアが必要なレジスタをコンパイラが静的に決定し，ストア命令とロード命令を埋め込む．このストアとロードの実行時間は待避レジスタ数が増えるに従い長くなり，スレッド切替えオーバヘッドを増加させてしまう．

スレッド切替えの要因は，関数呼出しと復帰，およびリモート読出しである．共有メモリプログラムでは一般にリモート読出しの回数が多いため，本論文におけるオーバヘッド解析はリモート読出しに焦点を絞って行う．EM-C コンパイラはマルチスレッド実行を前提としているため，スレッドを 1 本しか用意しない場合でも，これらの要因が発生するときには必ずいったんスレッドを中断するコードを生成する．

3. ローカルコピーを利用する共有メモリ

共有メモリアクセスが高頻度に行われるプログラムでは，スレッド切替えオーバヘッドが実行性能に大きな影響を及ぼす．この問題を回避する手法として，共有データのコピーを各プロセッサのローカルメモリに置き，それらの間で一貫性を維持する機構を導入する（Coherent Local Copy; CL）．CL は他の DSM システムと同様に，データアクセスの局所性を利用するものであり，必要なデータがローカルメモリにコピーされていれば通信を行わずにローカルアクセスを行えばよい（**図 1**）．結果として共有アクセスごとのスレッド切替えの必要がなくなり，通信の粗粒度化による効率化や，通信量の削減等のメリットが得られる．

EM-X では仮想記憶機構やアクセス制御のハードウェアサポートがないため，ミスチェックやミス時処理をすべてソフトウェアで実現する必要があるが，これらは新たなオーバヘッドとなる．

3.1 基本構成

今回実装した EM-X の CL 機構は一般的な SDSM と多くの共通点を持つ．ただし，仮想アドレスは EM-X におけるハードウェアサポートがないことから導入せず，その結果，論理共有アドレスは各 PE でそのまま物理ローカルアドレスとなる．このことは次に述べるようにデータサイズに関して厳しい制限をもたらすが，限定された条件のもとで，MMU を用いた仮想記憶システムの近似的な評価手段とすることができる．

各 PE のローカルアドレス空間には全共有空間のコピー領域が割り当てられ，すべての PE が同じ共有ア

☆ 通信相手が自 PE である場合．
☆☆ システム内の平均値．

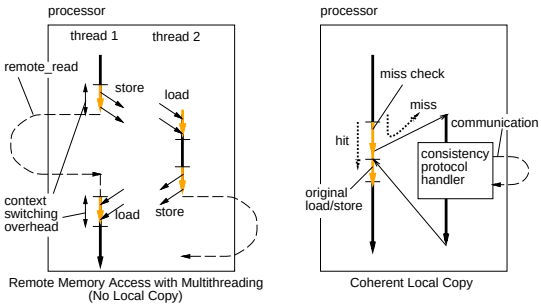

図 1 分散メモリアーキテクチャにおける共有メモリの実現方法：リモートメモリアクセス（NL）vs. コヒーレントコピー（CL）
Fig. 1 Shared memory on distributed memory processor: Remote Memory Access with multithreading (no local copy; NL) vs. coherent local copy (CL).

ドレスを同じローカルアドレス上にマップする．一般の SDSM ではこれが仮想アドレス空間で行われ，実際に使用する共有ブロックのみが物理メモリに置かれるのに対し，本実装では物理アドレス空間を用いるため共有メモリのサイズは物理メモリのサイズに制限される．また，容量性ミスを扱わない実装となっているため，容量性ミスが起きないような小規模のワーキングセットを扱う必要がある．後述の性能評価実験で用いるデータはその条件を満足すると仮定する．これにより，CL のオーバヘッドの議論はミスチェックオーバヘッドと共有ミス☆時のプロトコルオーバヘッドに焦点を絞ることができる．ただし実際に仮想記憶を用いたシステムでは，容量性のミスは起きうるために適切なブロック置換え処理が必要となるが，その手法やオーバヘッドの議論については本論文の範囲を超えており，別の機会に検討する．

SDSM では多様なメモリ・コンシステンシモデルとコンシステンシ維持プロトコルが研究されているが，ここでは単純化のため，Sequencial Consistency モデルとディレクトリベースの無効化プロトコルを用いる．共有メモリ空間は一定サイズのブロックに分割され，すべてのブロックに固有のホーム PE が静的に割り当てられる．ホーム PE は共有空間上のブロックの状態を示すディレクトリを管理する．ディレクトリにはホームに最新コピーがあるかどうかのビット（dirty bit），オーナ（最新コピーを持つ PE）を示す ID，コピー保持者を示すフルマップのベクタを持つ．

3.2 共有ミス判定方法

共有ミスの検出には，多くの SDSM では MMU の page fault を利用するが，Shasta[6]や UDSM[7]のようにソフトウェアで行う方法もある．EM-X では MMU がないため，ソフトウェアでミスチェックを行う必要がある．そのために，該当ロード・ストア命令の前に状態フラグを検査するための命令コードを挿入する．挿入する命令の概要は以下のとおりである．

(1) 該当ワードがあるブロックに対応する状態フラグのポインタを計算
(2) 状態フラグをロードして内容をテスト
(3) ヒットならばオリジナルのメモリアクセス命令を実行
(4) ミスならばミス処理ルーチンへ分岐

今回は，将来コンパイラを改良することを念頭に置いて，C ソースに対する手作業およびプリプロセッサ処理と，コンパイル結果のアセンブリ言語出力に対するポストプロセッサ処理により，半自動的にミスチェックのコードとミス時処理ルーチン呼出し命令コードを挿入した．これらの命令コードは定常的なオーバヘッドの要因となるため，できるだけ少ない命令数となることが望ましいが，主にコンパイル結果のレジスタ割当ての複雑さのために，現状では冗長な命令がいくつか残っている．レジスタ割当てと命令配置に関する最適化はコンパイラの改良が必要であり，そこまでの追求は今回は行わなかった．ただし最内周ループが単純なプログラムについては，コンパイラ実装を念頭に置きながら，手作業によってこれらの最適化を施したプログラムも作成した．

3.3 状態管理方法

共有データの各コピーブロックの状態は，一般的な INVALID，SHARED，EXCLUSIVE の 3 状態とし，各ブロックに対応づけられた状態フラグ領域に保持する．

EMC-Y の演算レジスタとローカルメモリの各ワードには，データ駆動モデルに基づく待ち合わせ処理のためのタグフィールドが含まれている．1 ワードは 38 bit で，6 bit のタグフィールドと 32 bit のデータフィールドからなる．本システムではこのタグフィールドを共有ブロックの状態フラグとして用いる．EMC-Y のロード・ストア命令はこの 38 bit を同時にアクセスでき，タグ操作命令によりタグフィールド内のデータを用いた演算や制御が可能である．タグフィールドは，該当メモリワード上でデータ駆動待ち合わせを行わなければ，汎用目的で使用できる．タグフィールドをブロックの状態フラグとして用いれば，状態フラグテーブルが不要になるだけでなく，処理の効率化も可能である．

状態フラグ領域の実現方法は，ミス判定とミス処理

☆ 有効なローカルコピーがないために起きるミス．

時の状態変更の効率に大きく影響する．その様子を調べるため，次の 3 通り（+最適化 1 通り）の方法を用意する．

State on Table（ST） 各ブロックの状態を表すテーブルを配列で確保する．タグフィールドは用いない．該当エントリへのアクセスのためにはテーブルのベースアドレスとオフセットを所望データのポインタから計算する必要がある．状態フラグのアクセスに時間がかかるが，汎用的な実装である．

State on Block（SB） 各ブロックの先頭ワードのタグフィールドに状態フラグを格納する．状態フラグ位置へのポインタは，マスク演算により所望データのポインタをブロックサイズ単位の配置に合わせるだけで求まる．状態フラグのアクセスおよび状態の変更は比較的高速に行える．

State on Word（SW） ブロック上のすべてのワードのタグフィールドに同一の状態フラグを格納する．状態フラグを検査する位置は所望データのポインタそのものを用いればよく，ポインタの再計算は不要である．ただし，状態を変更する場合はブロック内の状態フラグをすべて変更する必要がある．判定のための状態フラグアクセスは高速に行えるが，状態の変更には時間がかかる．

State on Word – Optimized（SWO） プリ・ポストプロセッサを用いた命令コード挿入から一歩踏み込んで，SW に対してさらに手作業による最適化を施す．これは，現在のコンパイラとの連携が不十分であることにより残っている冗長な命令を取り除くもので，一般的なコンパイラが持つ最適化能力を逸脱しないよう留意して行う．今回は手作業のため，比較的単純なループへの適用に向く．逐次コードに対する命令増加は，ロード命令では+2 命令，ストア命令では+4 命令となる．

3.4 コンシステンシ維持プロトコル処理

共有アクセスのミスを検出した後の処理は，ライブラリとして提供されるプロトコル処理ルーチンが担う．PE 間のリクエスト発行や共有ブロックの転送は，リモート関数呼出しとライブラリのリモートブロック転送ルーチンを用い，メッセージ通信方式で行われる．使用しているコンシステンシモデルならびにプロトコルの実装については，ともに普遍的な方式であることから詳細な説明は省略する．

4. ベンチマークプログラムの実装

本章では，性能評価実験に用いる共有メモリプログラムの実装について，リモートメモリアクセスを用いる方法（NL）とローカルコピーを用いる方法（CL）をそれぞれ説明する．

SPLASH2 ベンチマークセットの中から評価に用いる共有メモリプログラムとして，次の 3 つのプログラムを選んだ．

- LU（contig）：ブロック化した密行列 LU 分解の計算を行う．ブロック内データはメモリ上で連続配置される．
- FFT：1 次元 FFT を，局所計算と行列転置を交互に計 6 ステップ行うことで計算する．
- BARNES：粒子の相互作用の計算を Barnes-Hut の Tree アルゴリズムにより行う．

SPLASH2 のプログラムは，種々のアーキテクチャや処理系へ実装することを考慮して，同期や共有メモリ割当て等の並列プリミティブが PARMACS というマクロで記述されている．また，物理的な分散メモリを持つマシンに実装しやすいように，共有空間のマッピングを実装者に任せている．各オリジナルソースプログラムにはアクセスの局所性を高めるためにマッピングの簡単なガイドラインが示してあり，EM-X での実装もそれに従った．SPLASH2 のメモリモデルは共有領域とローカル領域に分かれている．G_MALLOC マクロで動的に確保される領域が共有領域となり，静的に宣言された領域はローカル領域となる．このことに留意して，明らかなローカル作業用等，共有メモリである必要のないものをローカルアクセスに置き換えたうえで，プログラム上の通信主体の変更や解法アルゴリズムの変更等は行わないこととした．

4.1 逐次実装（SEQ）

SPLASH2 では，逐次プログラムを得るために並列化記述を隠す Null Macro が用意されている．それをオリジナルプログラムに適用した後，浮動小数点演算精度を単精度とする等 EM-X 向けの変更を施したものを，逐次バージョン（SEQ）とする．SEQ は，並列化や通信のためのコードをいっさい含まない．

4.2 NL による共有メモリアクセス

NL による並列プログラムでは，共有アクセスのために，リモートメモリアクセスのパケットを発行する命令が生成される．リモート読出し（共有読出し）の際にはスレッドを中断し，さらにレジスタを待避・復帰する命令が生成される．以下に示すように，単一のスレッドを用いた基本実装（Base; BS）と，レイテンシ隠蔽を目的としたマルチスレッド実装（Multithread; MT）を用意する．

並列プログラムの基本実装（BS） BS では，コンパ

イラにおける指示文の使用を念頭に置いて，次のような手順で実装した．

- G_MALLOC マクロで確保される領域は原則としてすべて共有メモリとして扱う．すなわちその領域のアクセスはすべて通信をともなう．
- ただし G_MALLOC でローカル作業用配列を確保している部分はローカルメモリとして確保する．
- 共有変数へのアクセスを，EM-C のグローバルポインタを用いたアクセスに変更する．
- EM-X 固有の同期プリミティブ等は PARMACS マクロに埋め込む．
- プログラムの初期化から終了時にわたって不変な少数のグローバル定数（基本パラメータ等）を保持する変数は，初期化時に各 PE へ値を放送し，その後は各 PE でローカルアクセスを行う．
- 共有メモリ空間と物理空間のマッピングを決め，グローバルポインタは配列インデクスから求めるか，変換テーブルを参照する．同一 PE 内であることが明らかな場合は通常のポインタ計算を適用する．
- 浮動小数点演算精度はハードウェアを利用するため単精度とする．

マルチスレッド実装（MT） MT は，BS をもとにマルチスレッド化を施したものである．マルチスレッド化はループ分割により行う．すなわち，隣り合うイテレーションを別のスレッドに巡回的に割り当てる．その際，スレッド生成・同期オーバヘッドを抑えるために，スレッドの片寄りが起きない程度になるべく外側のループに適用する．マルチスレッド化可能な部分は，同一 PE 内のスレッドどうしで冗長な処理をしないよう大局的に手作業で判断する．また，必要に応じて作業用変数をスレッドごとに局所化する．

4.3 CL による共有メモリアクセス

CL による並列プログラムも，原則として BS で示した手順で実装される．ただし，共有アクセスはグローバルポインタではなく，共有空間用のポインタを用いる．共有アクセスのたびにコピーの共有状態がチェックされ，ヒットすれば通常のロード・ストア命令が実行される．ミスすればコンシステンシ維持プロトコル処理ルーチンが呼ばれる．

3 章で述べたように，CL の実装は原則としてコンパイルの前後にプリプロセッサとポストプロセッサを適用することにより半自動的に行った．今回作成した CL 実装支援プリプロセッサは，単純なポインタ操作と比較的規則的な配列計算にのみ対応しているため，3 つの SPLASH2 プログラムの中では，LU と FFT についてのみ適用し，FFT は ST/SB/SW の 3 種類，LU はそれに SWO を加えた 4 種類の実装を用いた．BARNES はプログラム構造とデータ構造が複雑で，プリプロセッサでの対応は限界があったため今回 CL は適用していない．今後適用範囲を広げ，より最適化をおし進めるためには，コンパイラの構文解析能力や最適化能力との連携が課題となる．

実装手法としては，さらに，ミス時にスレッドを切り替えるマルチスレッド実行を組み合わせることも考えられるが，今回の評価では CL の効果を切り分けて評価するために，マルチスレッドの併用は実施していない．

4.4 SPLASH2 プログラム

次に，各 SPLASH2 プログラムの実装の詳細について説明する．

LU データサイズは 512×512 とする．

LU ではブロックサイクリック状に PE を使用するため，データ分散もそれに従い BS を実装した．MT については，互いに依存性のない最内ループの外側のループを分割しスレッドに割り当てた．プログラムを注意して見ると，データブロックの更新はそのブロックが割り当てられた PE（ホーム PE）だけが行うように作られており，PE ごとの局所参照性があることが分かる．CL ではこの局所参照性が自然な形で利用される．最内ループが比較的単純であるため，SWO の実装が可能である．

FFT データサイズは 65536 点とする．

FFT では 1 次元データを 2 次元配列として扱う．行ブロック単位で PE 内計算を行うため，データ分散も行ブロックとして BS を実装した．ブロック割当ては非巡回である．6 ステップのうち 3 つはローカル計算ステップであり，部分 FFT 計算とそこで発生するデータアクセスはすべてブロック単位でホーム PE により行われる．通信は残りの 3 ステップの行列転置で発生する．その通信は規則的で，更新はホームが行い参照はホーム以外が行う．MT においては，計算ステップのブロック内処理を行単位でスレッドに割り当て，転置ステップでは転送単位である正方小ブロック内の列単位で割り当てた．CL では，共有アクセスを行うループボディが大きくやや複雑であったため，

SWO は適用できなかった．

BARNES データサイズは 2048 粒子とする．

BARNES は，複数パラメータを持つデータ要素を主に 8 進木構造で動的に管理するため，複雑なデータ構造とアクセスパターンを持つ．主要データ構造のうち，leaf と cell データは各 PE に割り付けられる．body（=particle，粒子データ本体）は計算主体の定まらない共有データであるが，BS および MT では leaf や cell と同様に初期化時に各 PE に均等に割り付けた．計算は主に木の生成，相互作用の計算，座標計算等のステップで進み，その中では相互作用の計算が最も計算量が多い．各データ構造の計算主体はホーム PE 以外に広く分散するため，通信パターンは不規則である．MT では相互作用計算のステップにマルチスレッドを適用した．相互作用計算を開始するときに複数スレッドを生成し，粒子インデクスでスレッドを振り分けた．

5. 実験結果

本章では各ベンチマークプログラムの実行結果を示し，その解析を行う．明記していない実行パラメータは SPLASH2 のデフォルト値を使用した．

5.1 NL による共有アクセス

NL による LU，FFT，BARNES の実行時間を，それぞれ**図 2**〜**図 4** に示す．それぞれの図では SEQ/BS/MT をグループ分けし，BS と MT は PE 数を 1 から 64 まで変化させた場合の結果が示してある．BS と MT の 1PE の場合にはオーバヘッドの内訳が併記してあり，実行時間の左側のバーがオーバヘッドを示す．MT のスレッド数は一律に 1PE あたり 4 とした．これは，現在の EM-X の構成と，今回使用したプログラムにおいて，レイテンシを十分隠蔽できるスレッド数である．

図 2 の LU の測定時間（Measured Time）において，`Exec` は実際の計算時間である．`Barrier Wait` はバリア同期による待ち時間であり，各 PE がバリア待ちに入ってから残りの全 PE がバリアに到達するまでの時間の PE ごとの累計の平均値である．図 3 の FFT の測定時間において，`Calculation` は各 PE の受持ち領域の計算時間，`Transpose` は行列転置の時間である．図 4 の BARNES の測定時間において，`ForceCalc` は相互作用の計算時間，`TreeBuild` は 8 進木の生成時間，`Partition+Rest` は粒子ポインタの分散と座標計算である．なお，BARNES のデフォルトサイズである 16384 particles は 1PE のメモリに収まらなかった

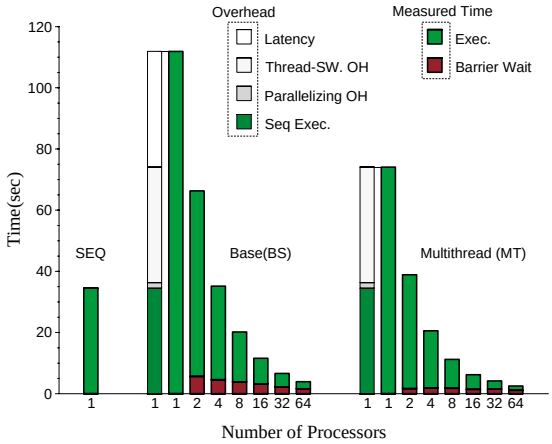

図 2 実行時間（LU-NL：512 × 512 matrix）
Fig. 2 Execution time (LU-NL: 512 × 512 matrix).

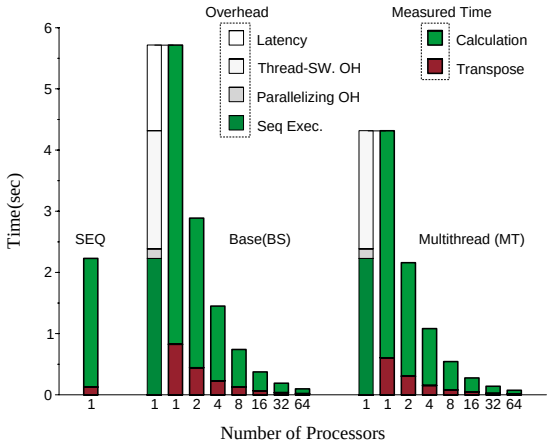

図 3 実行時間（FFT-NL：65536 points）
Fig. 3 Execution time (FFT-NL: 65536 points).

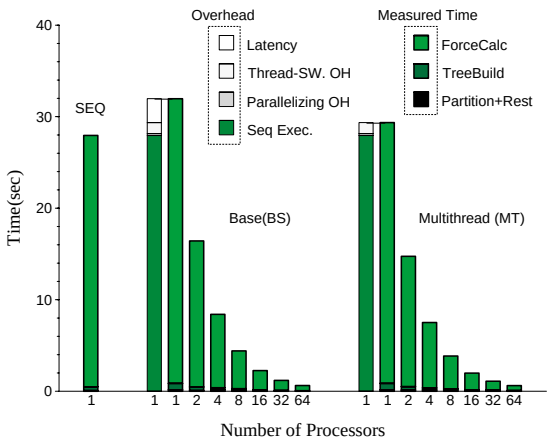

図 4 実行時間（BARNES-NL：2048 particles）
Fig. 4 Execution time (BARNES-NL: 2048 particles).

ため，2048 particles で実行した．

オーバヘッド内訳の意味は次のとおりである．`Seq Exec`．はアプリケーションを実行するのに本質的に必要な演算処理の時間であり，SEQ の実行時間に相当する．`Parallelizing OH` は，プログラムを並列化した結果増加した同期やリモートスレッド生成等にかかる時間である．`Thread-SW. OH` は，共有アクセスごとのスレッド切替え処理の時間である．`Latency` はリモートメモリアクセスにともなう通信レイテンシである．

SEQ の実行時間を 1 とした LU，FFT，BARNES の台数効果を，それぞれ**図 5**〜**図 7** に示す．これらのうち LU と FFT では，比較のため CL の結果もあわせて表示してある．台数効果のグラフで `PRAM` と表記されている曲線は，文献 16) から引用した PRAM モデルにおける性能である．PRAM モデルで得られる性能は，通信オーバヘッドを排除した問題固有の性能上限を与える．BARNES の PRAM 性能については文献 16) の条件が 16384 particles のものであるため，比較を行う場合はその違いに注意する必要がある．

5.2 CL による共有アクセス

CL による LU と FFT の実行時間を，それぞれ**図 8**，**図 9** に示す．各図では，ST/SB/SW/SBO（LU のみ）をグループ分けし，それぞれについて PE 数を 1 から 64 まで変化させて測定した結果を示してある．時間の絶対値を折れ線グラフで示し，実行時間に占めるオーバヘッドの割合を 100%に対する帯グラフで表してある．ブロックサイズはすべての場合で 256 Bytes とした．他のパラメータは NL の場合と同一である．

オーバヘッド内訳を示すグラフの中で，`exec` は有効な計算時間，`check` はミスチェックにかかる時間，`protocol` はミス時のコンシステンシ維持プロトコルの処理時間である．オーバヘッドの算出は，プロトコル処理時間を 0 と仮定したプログラムと，ミスチェック時間を 0 と仮定したプログラムを用いて，実行時間を差引きすることによって得た．これは実行結果は正しくならないが，LU と FFT については実行順序がデータに依存せず保存されるため可能な方法である．

CL の台数効果は，NL とあわせて図 5，図 6 に示してある．CL の場合，それぞれプログラムで最も高い性能が得られた状態フラグ実装方法を選んで表示してある．

5.3 解　　析

本節では，得られた実験結果の解析を行う．

5.3.1 NL 方式の性能

図 2，図 3 によると，LU および FFT の 1PE 時の実行時間が BS，MT とも SEQ より大幅に増加して

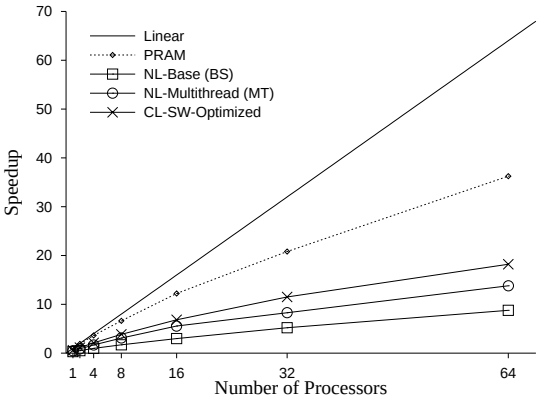

図 5 台数効果（LU-NL，LU-CL：512 × 512 matrix）
Fig. 5 Speedup (LU-NL, LU-CL: 512 × 512 matrix).

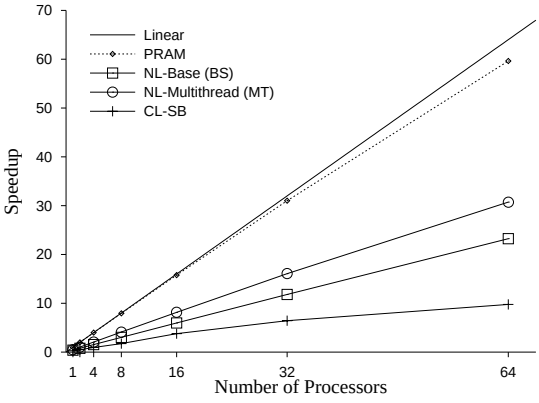

図 6 台数効果（FFT-NL，FFT-CL：65536 points）
Fig. 6 Speedup (FFT-NL, FFT-CL: 65536 points).

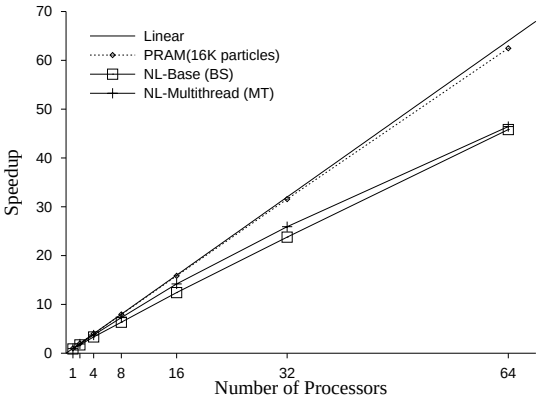

図 7 台数効果（BARNES-NL：2048 particles）
Fig. 7 Speedup (BARNES-NL: 2048 particles).

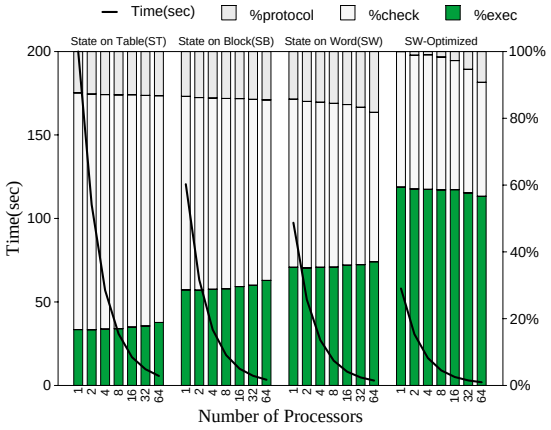

図 8 実行時間とオーバヘッドの内訳（LU-CL：512 × 512 matrix）
Fig. 8 Execution time and overhead breakdown (LU-CL: 512 × 512 matrix).

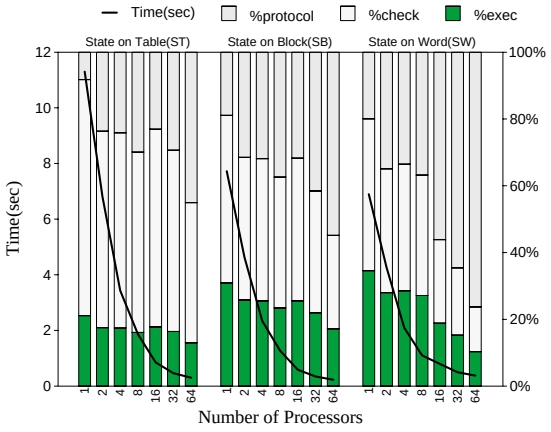

図 9 実行時間とオーバヘッドの内訳（FFT-CL：65536 points）
Fig. 9 Execution time and overhead breakdown (FFT-CL: 65536 points).

いる．BS は細粒度通信のレイテンシとスレッド切替えオーバヘッドの影響が直接現れていることにより，LU では 224%，FFT では 156%の増加となっている．MT はそのうちレイテンシを隠蔽することによって性能が改善されているが，スレッド切替えオーバヘッドが残っており，LU と FFT のオーバヘッドはそれぞれ 115%と 94%となっている．

図 5，図 6 によると，PE 台数を増やしても，逐次実行に対するこれらのオーバヘッドが性能向上を抑制していることが分かる．これらの台数効果は，PRAM が示す上限からスレッド切替えオーバヘッドを差し引いたものを表していると考えられる．

なお LU の PRAM が示す性能上限が高くない理由は，バリアにおける待ち時間である．これは行列計算のブロック化にともなう負荷のアンバランスに起因し，オリジナルのアルゴリズムが本質的に持つ性質である．

図 4 および図 7 の BARNES の結果を見ると，オーバヘッドは比較的小さい．これは，スレッド切替えを引き起こす共有メモリアクセスの頻度が少ないプログラムであるからである．台数効果は大きく，LU や FFT に比べ，PRAM が示す性能に迫っている．MT の 64 台時には SEQ の 46 倍の性能に達した．データサイズが小さくなると一般には負荷バランスが悪化するため，実験の条件は図に引用した PRAM の場合より悪く，実際の理想性能はより近くにあると考えられる．通信頻度が少ないことから MT の効果は小さい．

5.3.2 NL 方式のオーバヘッド

並列実行時の細粒度通信オーバヘッドの正確な解析は，クリティカルパスが特定できないため困難である．そこで，まず 1PE 時のオーバヘッド解析を測定結果を用いて行う．

BS および MT における共有メモリアクセスは，1PE での実行においてもすべて通信を発生する．結果的に自 PE 内の通信となるためそのレイテンシは短く，MT ではマルチスレッドによりレイテンシのほとんどの部分を隠蔽できるとする☆．BS と MT の実行結果の差はこの隠蔽されたレイテンシに相当する．スレッド制御やグローバルポインタ操作等，並列実行制御にかかる時間を別途測定して MT から差し引き，さらに SEQ との差をとると，残るのは共有アクセスに起因するスレッド切替えオーバヘッドである．実際には並列実行制御時間は小さく，MT と SEQ の差のうちスレッド切替えオーバヘッドがほとんどを占める．図 2，図 3 に示したオーバヘッド内訳は上述のようにして得た．

1PE では，MT によるレイテンシ隠蔽効果をスレッド切替えオーバヘッドが相殺しているが，2PE 以上ではリモート読出し 1 回あたりのレイテンシがより長くなるため，それを隠蔽することによる MT の効果が高まることは定性的に明らかである．

さらに，LU では最内周ループが単純であることから命令実行の局所的な解析を行い，1 イテレーションあたりのスレッド切替えオーバヘッドの内訳を調べた．1 イテレーションには 2 回の共有読出しと 1 回の共有書込みが含まれており，スレッドの中断は 2 回発生する．**図 10** に示した解析結果によれば，1 イテレーションあたりのオーバヘッド 13 クロック中 9 クロックがコンテクスト待避・復帰に費やされている．残りはハードウェア上のオーバヘッドと continuation 生

☆ 実際には 2PE 以上でも十分隠蔽できる．

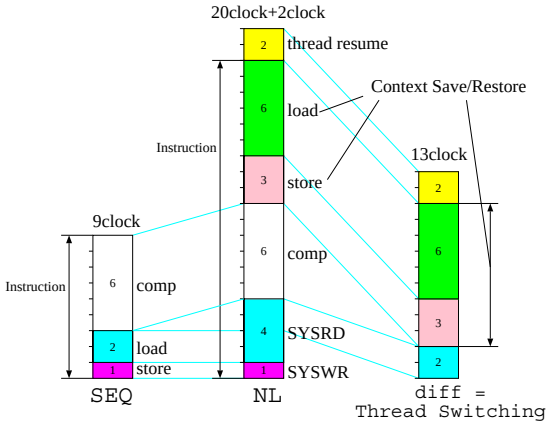

図 10 LU の最内周ループのスレッド切替えオーバヘッド
Fig. 10 Thread switching overhead of the most inner loop of LU with NL.

成である．なお，増加クロックサイクル数とループ実行回数からスレッド切替えオーバヘッドの合計を計算し，1PE における測定結果からの算出値とほぼ一致することを確認している．このようにスレッド切替えオーバヘッドの中では，コンテクスト待避の割合が大きい．FFT のようにループ中で使用される変数の数が増えると，さらに大きくなる．

5.3.3 CL 方式の性能

CL による LU の実行結果において，状態フラグの実装方法を比較すると，共有ブロック上の該当ワードに状態フラグを置いた SW が高い性能を示した．特に最適化を施した SWO は NL の MT より高い性能となり，64PE 時で MT が SEQ の 13.8 倍であるのに対し，SWO は 18.2 倍となり，約 32% の性能向上となった．これは，ソフトウェアオーバヘッドが十分低く抑えられているとともに，ローカルコピーの有効利用が成功していることを示している．LU の最内周ループの定常的なオーバヘッドを比較すると，SWO では，1 イテレーション 9 命令の逐次コードに対して 8 命令（8 clocks）の追加となっており，図 10 における NL のオーバヘッド：13 clocks より小さい．言い換えれば，スレッド切替えオーバヘッド削減効果がソフトウェアオーバヘッド増加分を上回っていることを示している．LU で SW が良い性能を示す理由は，ミスが少ないため，ミス処理に時間がかかってもミスチェックの効率が高い方が有利であるからである．

一方 FFT では，1PE では SW が良いが，64PE 時にはブロックの先頭に状態フラグを置いた SB の方が良い．これは，FFT ではミスと無効化処理が多く，PE が増えるに従い状態フラグの変更の効率が良い方が有利となるからである．ページベース DSM の場合の FFT では，転置ステップにおけるキャッシュ無効化の悪影響が非常に大きいといわれており，今回の結果で NL より大幅に性能が低いのはそれと同じ理由である．

表 2 ミス回数（64PE 時，1PE あたり平均値）
Table 2 Miss Counts (for 64PEs, average number of misses per processor).

	Read-miss	Write-miss	Invalidation	unit
LU-CL	451.5	63.0	63.0	counts
ST	79.4	11.1	11.1	
SB	132.4	18.5	18.5	
SW	155.9	21.8	21.8	counts/sec
SWO	238.5	33.3	33.3	
FFT-CL	752.1	96.0	531.6	counts
ST	2493	318.2	1762	
SB	3299	421.1	2332	counts/sec
SW	1986	253.5	1404	

LU，FFT とも，状態フラグテーブルを用いる ST 方式はミスチェックのオーバヘッドが大きく，性能は良くない．

図 8 と図 9 において，有効な命令実行時間（`exec`）に対する相対的なオーバヘッドの大きさを観察すると，ミスチェック（`check`）の割合は LU と FFT で大きな違いはないが，プロトコル処理部（`protocol`）は LU に比べて FFT が大きいことが分かる．また，FFT では PE 数が増えると，このプロトコルオーバヘッドの割合が増大している．ミスと無効化がこの直接的な原因である．プロトコルオーバヘッドは，リモート関数呼出し，リモートブロック転送，ブロックの状態変更，無効化情報のマルチキャスト等にかかる時間を含む．プロトコル処理のためにメッセージが数回往復するが，この部分のマルチスレッド化は行われていないため，オーバヘッドにはそのレイテンシも含まれる．

5.3.4 CL 方式におけるミス回数

表 2 に LU-CL，FFT-CL の 64PE 実行時のミス回数について，PE あたりの平均値を示す．各プログラムごとに，状態フラグ実装方法間ではミス回数は同じであるが，実行時間の違いから結果的に単位時間あたりのミス回数が状態フラグ実装方法間で異っている．単位時間あたりの回数を求めるのは，プログラム間で比較するためである．

全体的に FFT は無効化頻度（Invalidation），ミス頻度ともに LU より多く，性能低下の理由となることが分かる．一方，LU はミス，無効化とも少なく，局所性とコヒーレント性を利用するシステムで高い性能が得られる性質を持っているといえる．

表 3 通信量（64PE 時，1PE あたり平均ワード数）
Table 3 Traffic (for 64PEs, average number of words per processer).

	PE 内		PE 間	
	count (words)	throughput (words/sec)	count (words)	throughput (words/sec)
LU-NL				
BS	1490 k	379 k	652 k	166 k
MT	1490 k	596 k	652 k	261 k
LU-CL				
ST	——		7.2 k	1.3 k
SB	——		7.2 k	2.1 k
SW	——		7.2 k	2.5 k
SWO	——		7.2 k	3.8 k
FFT-NL				
BS	85.6 k	893 k	6.1 k	63.1 k
MT	85.6 k	1181 k	6.1 k	83.4 k
FFT-CL				
ST	——		48.1 k	160 k
SB	——		48.1 k	211 k
SW	——		48.1 k	127 k
BANES-NL				
BS	8.2 k	13.4 k	78.8 k	129 k
MT	8.2 k	13.6 k	78.8 k	131 k

5.3.5 通 信 量

LU-NL/CL, FFT-NL/CL, BARNES-NL の 64PE 実行時の共有メモリアクセスを計数し，通信量として**表 3** に PE あたりの平均値を示した．共有メモリアクセス回数を各プログラムの実行時間で正規化した throughput で比較する．

NL では，共有メモリアクセスに起因する通信量を PE 内と PE 間に分けて示してある．一方 CL では，ミスに起因する PE 間通信量を示してある．PE 内ではミスに起因するデータ転送は発生しないため表示していない．

NL の PE 内と PE 間の通信量の合計は，スレッド切替え 1 回あたりのオーバヘッドが性能に与える影響の大きさを左右する．BARNES は比較的通信量が少なく，オーバヘッドは大きな問題とならない．

LU，FFT では，ともに PE 内通信が多く，PE ごとの局所性が高いことが分かる．LU-CL はこの局所性を利用することにより，通信量が大幅に削減されていることが分かる．FFT-CL では逆に PE 間通信量が増えているが，これは無効化が多く発生し，結果としてミスが多発することが原因である．

6. 考 察

EM-X のリモートメモリアクセスを用いた実行（NL）では，それぞれのプログラムにおいてプロセッサ数の増加に応じて一定の性能向上が得られている．これは，実験に用いたプログラムのアクセスパターンではネットワークのバンド幅に余裕があり，PE 台数の増加にともなうオーバヘッド増加がほとんどないためである．また，マルチスレッド実行（MT）はマルチスレッド実行しない場合（BS）に比べて一定の性能向上を与えており，レイテンシ削減効果を示している．性能上限からの比較では，BARNES が最も性能が良く，続いて FFT，最も悪いのは LU である．これは，共有アクセスの頻度，すなわちスレッド切替えオーバヘッドが直接影響している．

スレッド切替えオーバヘッドの除去を目的とした CL の効果は，共有アクセス頻度の高いプログラムであるほど有効であると考えられるが，そのほかにアクセスパターンの局所性と無効化頻度も重要である．LU と FFT はともに高頻度アクセスと高い局所性を持つが，LU では 無効化が少ないために CL が有効であったのに対し，FFT では行列転置にともなって頻発する無効化のために CL の性能が低く，NL が適していた．今回，手作業による最適化は LU のみに適用されたが，FFT に適用できたとしても，無効化によるプロトコルオーバヘッドが大きいため性能改善効果は小さいと考えられる．BARNES に対する CL の効果は検証できなかったが，細粒度のコンシステンシ管理を行える Shasta の研究では比較的良好な性能を示しているため，EM-X においても細粒度アクセス制御の実験を行い，局所性の有効利用が性能向上をもたらすかどうかの検証を今後行う必要がある．

ソフトウェアによる CL の実装は，その実装オーバヘッドが大きな問題である．特にミスチェックは定常的なオーバヘッドの要因となるが，今回，EM-X のタグ付きメモリを利用して，部分的に手作業で状態フラグ操作の最適化を行うことによってオーバヘッドの大幅な削減に成功した．今後これをコンパイラに組み込むことにより，汎用性を高めるとともに，冗長なミスチェックの削減等，より進んだ最適化手法の追求が可能となる．ハードウェアの観点からは，タグフィールドの利用が効果をあげたことから，そのようなミスチェックをサポートする機構が有効であると考えられ，より効果的なサポート機構について今後検討したい．

プロトコル処理部の最適化は，ミス頻度の高い場合に特に重要となるが，今回の評価に用いた実装ではいまだに多くの最適化の余地がある．例えばプロトコルメッセージに関しては，メッセージオーバヘッドそのものは EM-X の細粒度通信機構の利用により大きな問題ではないが，マルチスレッド化を施していないため，リクエスト先の処理に時間がかかるとレイテンシを増長させる原因となる．そのほか，今後効率の良い

マルチキャスト方式や，種々のプロトコル等を追求していく必要がある．

本論文の評価では，NL と CL が相補的に並列処理の効果を示した．これは，問題の性質に応じて細粒度アクセスが有効に働く場合と，逆にそのオーバヘッドが問題となって局所性利用の方が有利に働く場合があるからである．後者は一般的な DSM でも示される性質であるが，前者は EM-X の演算と通信を融合したアーキテクチャの効果であると考えられる．しかしながら EM-X におけるこの性質は，メモリアクセスおよび通信が演算と同程度のスループットで動作し，それらのレイテンシが非常に小さいことにも大きく依存している．近年の高クロック周波数を前提とした回路技術を用いた場合には，メモリアクセスと通信性能は相対的に低くなると考えられ，アーキテクチャのパラメータを見直す必要がでてくる．そのため，今後 EM-X のアーキテクチャ研究にもそのような前提を導入し，そのうえで有効な共有メモリアクセス技術の研究を進めていく予定である．

6.1 関 連 研 究

EM-X の CL は仮想記憶を用いない実装であるため，非常に小さな規模の問題でしか性能評価が行えなかった．これに対し，これまで研究されてきた SDSM システムの多くは，仮想記憶機構を利用したものである．IVY[4]や TreadMarks[5]等は，ワークステーションクラスタの各ノードにおいて MMU の page fault でアクセスミスを検出し，ページ単位のコンシステンシ制御を行っている．TreadMarks では LRC（Lazy Release Consistency）モデルを Multiple Writer プロトコルにより実装し，false sharing の影響を低減している．しかしながら，メッセージ通信のオーバヘッドが大きいため性能向上は限定的である．

CASHMERe[17]，Shasta[6]，SCASH[9]は，メモリマップされた通信インタフェースのリモートメモリ書込み機能を利用しており，高い性能を実現している．EM-X のようにリモート読出し機能が利用できるようになれば，より性能が向上することが期待できる．

Shasta[6]や UDSM[7]は，各ノードのローカル仮想記憶空間の中に共有領域を設定しているが，ミスチェックは MMU に頼らず，ソフトウェアで行っている．高度な最適化により，ソフトウェアオーバヘッドを効果的に削減しているうえ，ソフトウェア使用の利点である柔軟性を積極的に利用している．EM-X における CL 方式は，ソフトウェアによるミスチェックという点でこれらに近いが，より進んだ最適化の研究は未着手であり今後の課題である．

EM-X における CL の実装では，メモリのタグフィールドを利用することでミスチェックのオーバヘッドを削減した．このようにデータワードのほかに付加ビットを利用するシステムとして，メモリモジュールの ECC bit を利用した Blizzard-E[18]等がある．タグメモリのアプローチに汎用部品を利用できることを示す例として興味深い．

上にあげた SDSM システムは，すべてローカルコピーの一貫性を維持することによって実現されるものである．それに対し，EM-X は同様の方式である CL だけでなく，リモートメモリアクセスを用いる NL によっても共有メモリプログラムを効率良く実行することができ，効率的実行の可能性を高めている．

このほか，山本ら[19]は，明示的な通信コードの挿入により SPLASH2 の各プログラムをメッセージ通信用に変更している．本論文では行わなかった通信主体の変更によって，多くのプログラムにおいて高い性能を得ているが，不規則アクセスを生じる BARNES では成功していない．

Mowry[13]らは共有メモリモデルでのソフトウェア制御マルチスレッドの性能を報告している．キャッシュミスの際にソフトウェアでスレッドを切り替えることによりレイテンシを隠蔽している．EM-X と同様にコンテクスト待避のための明示的なセーブ・リストアによるオーバヘッドが問題であるとしているが，低コストな削減テクニックとしてレジスタ分割（register partitioning）を試みており興味深い．ただしスレッドあたりのレジスタ数が減るため，性能が向上する例は一部である．また，4 プロセッサまでの評価となっており，より多数のプロセッサでの性能は不明である．

7. ま と め

種々の共有メモリプログラムを分散メモリ型アーキテクチャで効率良く実行することを目的とし，細粒度通信機構を持つ EM-X を用いて細粒度のリモートメモリアクセスを用いる No Local Copy（NL）方式と，ローカルコピーを利用する Coherent Local Copy（CL）方式を，ベンチマークプログラムの実行によって比較した．

EM-X における基本的な共有メモリ実現方式である NL では，マルチスレッドによるレイテンシ隠蔽効果を確認したが，性能向上を大きく阻んでいる原因として，スレッド切替えオーバヘッドが問題であることを示した．スレッド切替え抑制によるオーバヘッド削減と，データアクセス局所性の利用を目的とし，EM-X 上で共有データのローカルコピーを管理する CL 機構をソ

フトウェアにより実装した．SPLASH2 プログラムを用いた評価において，LU では CL が NL を 32%上回る性能を示す一方，FFT では 68%下回った．

EM-X のアーキテクチャでは，NL と CL の間のトレードオフがアクセス頻度，データアクセス局所性，無効化頻度等によって決まる．あらかじめアプリケーションごとにこれらの性質を把握しておくことにより，適切な手法の選択が可能となり，種々のアプリケーションで効率的な実行が可能となることが分かった．

謝辞 本研究を遂行するにあたり，ご指導，ご討論いただいた電子技術総合研究所・大蒔和仁情報アーキテクチャ部長ならびに同僚諸氏，電気通信大学大学院情報システム学研究科並列処理学講座の皆様，有益なコメントをいただいた査読者の方々に感謝いたします．

参 考 文 献

1) 安生，井上，佐藤，工藤，天野，平木：超並列計算機 JUMP-1 における分散共有メモリ管理プロセッサ MBP-light，情報処理学会論文誌，Vol.39, No.6, pp.1632–1643 (1998).
2) Laudon, J. and Lenoski, D.: The SGI Origin: A ccNUMA Highly Scalable Server, *Proc. 24th Int. Symp. on Computer Architecture*, pp.241–251 (1997).
3) 細見，加納，中村，広瀬，中田：並列計算機 Cenju-4 の分散共有メモリ機構，並列処理シンポジウム JSPP'99, pp.15–22 (1999).
4) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, *Proc. Int. Conf. on Parallel Processing*, pp.94–101 (1988).
5) Keleher, P., Cox, A.L., Dwarkadas, S. and Zw-aenepoel, W.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, *Proc. Winter 1994 USENIX Conference* (1994).
6) Scales, D.J., Gharachorloo, K. and Thekkath, C.A.: Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory, *Proc. 7th Int. Conf. on ASPLOS*, pp.174–185 (1996).
7) 丹羽，稲垣，松本，平木：非対称分散共有メモリ上における最適化コンパイル技法の評価，情報処理学会論文誌，Vol.39, No.6, pp.1729–1737 (1998).
8) バルリ，渡辺，坂井，田中：高速通信機構を用いたソフトウェア DSM のパフォーマンス解析，電子情報通信学会技術研究報告，CPSY99-66, Vol.99, No.252, pp.33–41 (1999).
9) 原田，手塚，堀，住元，高橋，石川：Myrinet を用いた分散共有メモリにおけるメモリバリアの実装と評価，並列処理シンポジウム JSPP'99, pp.237–243 (1999).
10) Kodama, Y., Sakane, H., Sato, M., Yamana, H., Sakai, S. and Yamaguchi, Y.: The EM-X Parallel Computer: Architecture and Basic Performance, *Proc. 20th Int. Symp. on Computer Architecture*, pp.14–23 (1995).
11) 佐藤，児玉，坂井，山口：並列計算機 EM-4 の細粒度通信による共有メモリの実現とマルチスレッドによるレーテンシ隠蔽，情報処理学会論文誌，Vol.36, No.7, pp.1669–1679 (1995).
12) Sakane, H., Sato, M., Kodama, Y., Yamana, H., Sakai, S. and Yamaguchi, Y.: Dynamic Characteristics of Multithreaded Execution in the EM-X Multiprocessor, *Proc. Int. Workshop on Computer Performance Measurement and Analysis*, pp.14–22 (1995).
13) Mowry, T.C. and Ramkissoon, S.R.: Software-Cont-rolled Multithreading Using Informing Memory Operations, *Proc. 6th Int. Symp. on High Performance Computer Architecture*, pp.121–132 (2000).
14) 佐藤，児玉，坂井，山口：並列計算機 EM-4 の並列プログラミング言語 EM-C，並列処理シンポジウム JSPP'93, pp.183–190 (1993).
15) 児玉，坂根，佐藤，坂井，山口：高並列計算機 EM-X のリモートメモリ参照機構の評価，情報処理学会論文誌，Vol.36, No.7, pp.1691–1699 (1995).
16) Woo, S.C., Ohara, M., Torrie, E., Singh, J.P. and Gupta, A.: The SPLASH-2 Programs: Characterization and Methodological Considerations, *Proc. 22nd Int. Symp. on Computer Architecture*, pp.24–36 (1995).
17) Stets, R., Dwarkadas, S., Hardavellas, N., Hunt, G., Kontothanassis, L., Parthasarathy, S. and Scott, M.: Cashmere-2L: Software Coherent Shared Memory on a Clustered Remote-Write Network, *Proc. 16th ACM Symp. on Operating Systems Principles* (1997).
18) Schoinas, I., Falsafi, B., Lebeck, A.R., Reinhardt, S.K., Larus, J.R. and Wood, D.A.: Fine-grain access control for distributed shared memory, *Proc. 6th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.297–307 (1994).
19) 山本，宮脇，坂，工藤：共有メモリ向プログラムの通信の解析による最適化，電子情報通信学会技術研究報告，CPSY98-61, Vol.98, No.234, pp.9–14 (1998).

(平成 12 年 5 月 17 日受付)
(平成 12 年 9 月 5 日採録)

坂根　広史（正会員）

1990 年山口大学工学部電子工学科卒業．1992 年電気通信大学大学院博士前期課程電子工学専攻修了．同年通商産業省工業技術院電子技術総合研究所入所．以来，並列計算機のアーキテクチャおよびその性能評価の研究に従事．1998 年より，在職のまま電気通信大学大学院情報システム学研究科博士後期課程に在学．電子情報通信学会，神経回路学会各会員．

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 年同大学大学院博士課程修了．1987 年同大学情報科学教育センター助手．1991 年山梨大学工学部電子情報工学科専任講師．1992 年同助教授．1997 年電気通信大学大学院情報システム学研究科助教授．現在に至る．並列処理方式，並列コンパイラ，マルチプロセッサアーキテクチャ等の研究に従事．工学博士．電子情報通信学会，IEEE，ACM 各会員．

弓場　敏嗣（正会員）

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気試験所（現，電子技術総合研究所）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に従事．その間，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，日本ソフトウェア科学会，日本ロボット学会，ACM，IEEE-CS 各会員．

児玉　祐悦（正会員）

1986 年東京大学工学部計数工学科卒業．1988 年同大学大学院情報工学専門課程修士課程修了．同年電子技術総合研究所入所．現在，同所情報アーキテクチャ部主任研究官．データ駆動計算機，マルチスレッド計算機等の並列計算機システムの研究に従事．特にチップマルチプロセッサにおけるプロセッサアーキテクチャやその並列性制御等に興味あり．情報処理学会奨励賞，情報処理学会論文賞（1990 年度），市村学術賞（1995 年）等受賞．電子情報通信学会，IEEE 各会員．

山口　喜教（正会員）

1972 年東京大学工学部電子工学科卒業．同年電子技術総合研究所入所，計算機方式研究室長等を経て，1999 年筑波大学電子・情報工学系教授（電子技術総合研究所併任），工学博士．高級言語計算機，並列計算機アーキテクチャ，並列実時間システム等の研究に従事．1991 年情報処理学会論文賞，1995 年市村学術賞受賞．著書「データ駆動型並列計算機」（共著）．IEEE Computer Society，ACM，電子情報通信学会各会員．