Local メモリを持つ共有メモリ型並列計算機を効果的に使用するための環境 EULASH

山本 淳二　服部 大　徳吉 隆宏[*]　大和 純一[†]　天野 英晴

慶應義塾大学 理工学部

低速な共有メモリと高速なローカルメモリを持つ並列計算機を効率よく使用する環境である EULASH について述べる。

EULASH はプログラミングモデルの変換を行うプリプロセッサ、軽量なスレッドを基本とするカーネル (EULASH kernel)、変数管理機構を提供するライブラリからなる。

評価の結果、カーネルを導入することにより実行速度の若干の低下が見られたが、プログラムの記述性・移植性は向上した。また、新たなデータ管理機構を用いることで、スヌープキャッシュを持つバス結合型並列計算機 ATTEMPT-0 で数%程度、並列計算機シミュレータにより共有メモリがローカルメモリに比べ 4 倍遅い場合に 10% 程度の速度向上が認められた。

EULASH: An environment for efficient use of multiprocessors with local memory

J. Yamamoto　D. Hattori　T. Tokuyoshi[*]　J. Yamato[†]　H. Amano

Faculty of Science and Technology, Keio University

This paper describes the environment, EULASH which makes the best use of the multiprocessors with a high speed local memory and shared memory with a large latency. It consists of preprocessor to fit programs for the architecture of multiprocessors, the processing system, the EULASH kernel and the library for data management.

From result of empirical evaluations, it appears that the overhead of EULASH kernel is smaller than 5% if the TLB works effectively. Using the new data management, the performance is improve 10% if the access speed of the local memory is four times that of the shared memory.

---

[*] 現在 (株) 東芝

[†] 現在 日本電気 (株)

1 はじめに

現在、商用化され普及しつつあるバス結合型マルチプロセッサは、プロセッサ性能の向上にバスの転送能力が追従できないことから、既にその規模の限界が見え始めている。このため、さらに多数のプロセッサによる大規模な並列処理を行なうために様々な方法が試みられている。DASH[4]、Alewife 等の分散共有メモリ型マルチプロセッサに代表される NUMA (Non-Uniform Memory Access model) は大規模化の有力な候補であるが、分散共有メモリの一貫性を維持するためのハードウェアコストが大きい。

これに対し、複数のプロセッサで共有するメモリを持たず、ローカルメモリを持つプロセッサ同士がメッセージ交換により処理を進める NORA (Non Remote Access Memory model) は、大規模な構成を容易に実現できるが、共有メモリが存在しないため従来から用いられている共有メモリを意識したプログラムを移植することが困難なため用途が制限される。これを回避するためソフトウェアにより仮想的に共有メモリを実現する方法も試みられているが、効率の良い処理は困難である。

一方で、各プロセッサがローカルメモリを持つと共に、スイッチまたはバス等の結合媒体で直接アクセス可能な共有メモリを持つ構成も、大規模マルチプロセッサの有力な候補である。この構成は、NYU Ultracomputer[2] を始めとして古くから検討され実装が試みられているが、一般に普及するに至っていない。これはひとつには、共有メモリとローカルメモリを効率良く運用するための OS が確立していないことに原因があると考えられる。

また、プログラミングモデルとしては、全ての実行主体から全てのデータが同じ方法でアクセスできるモデルと、データによりアクセスの方法を変えるモデルの 2 つがあるが、現在はバス結合型マルチプロセッサが普及の主流であることもあり、前者のモデルに基づいているプログラムが多い。

そこで本研究では、全てのデータが等価にアクセスできるプログラミングモデルとローカルメモリも持つ共有メモリ型並列計算機のアーキテクチャの差を埋める EULASH システムを考案し、本論文ではまず設計について述べ、次に EULASH のカーネル部と新しいデータ管理機構についての実装・結果を示し、システムの評価を行なう。

2 EULASH 概要

本章では前章で述べたプログラミングモデルとターゲットアーキテクチャの差を埋めるシステムである EULASH の設計方針を述べる。

前章で述べたように、プログラムの記述性を第一に考えプログラミングモデルには単一メモリモデル (図 1) を、また、今後の並列計算機の流れを念頭においてターゲットアーキテクチャには共有メモリとローカルメモリの双方を持つシステムを選択した。しかし、単一メモリモデルのプログラムはそのままではこのターゲットアーキテクチャに実装することが出来ない。そこで、図 2の手順を通してプログラミングモデルに基づいたプログラムをターゲットアーキテクチャに最適なプログラムに変更する。

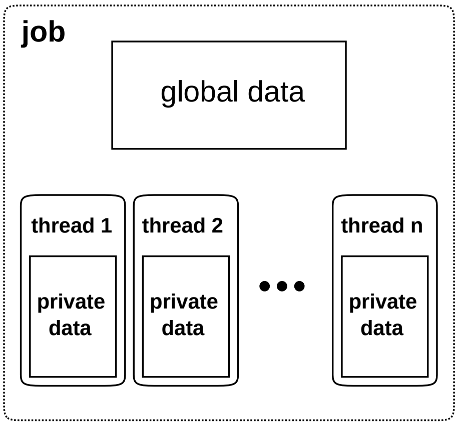

図 1: ソースプログラムのデータ共有関係

まず、コンパイラがソースプログラムを解析し、スレッドのプライベートデータと複数のスレッドで共有されるデータを分離する。また、このときに、データとスレッドの密接さも抽出し、どのスレッドを同じプロセッサに割り当てたほうが良いかを決定する。ここで、コンパイラによって配置場所を判断できないデータが見つかった場合はすべて共有メモリに配置される。しかし、ある一つのスレッドが主にアクセスし、ごくたまに他のスレッドもアクセスするようなデータは、主となるスレッドのプライベートデータとし、必要に応じて共有メモリにコピーしたほうが効率がよい。

しかし、プログラマがこの変数の管理を行うことは、プログラマに対してよけいな負荷を与えることになる。そこで、EULASH システムでは共有状態によって変数の配置場所が変化するデータ型で

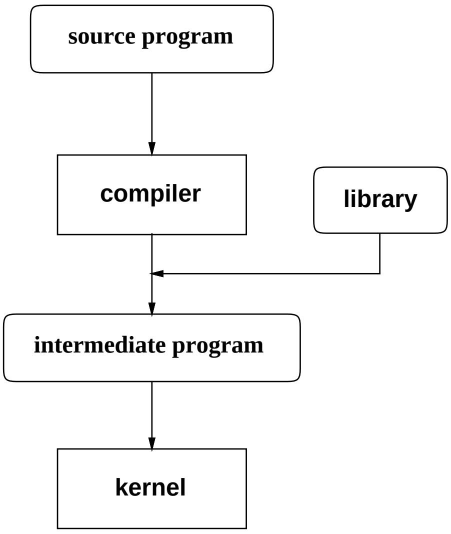

図 2: EULASH システムの流れ

ある mobile 型を提供する。また、EULASH システムではコンパイラが出力したプログラムを最大限に効率よく実行するための OS も提供する。ここで、 OS とは必要最低限の基本的な機能を提供するカーネルと、この機能を組み合わせ高度な機能を提供するライブラリからなる。多くの機能をライブラリ化する事でプログラムに応じて機能を変更、追加することが可能である。

3　カーネル

基本的に並列計算機は計算処理能力の向上を目指しているので、本システムのカーネルではターンアラウンド時間を短くする、すなわち単一ジョブをいかに高速に動作させるかに重点を置いて設計・実装を行い、マルチジョブ・マルチユーザについては考慮しない。また、カーネルでは最低限の処理を行い、ライブラリによって大部分をサポートすることで、ユーザモード・カーネルモードの移行を伴うシステムコールの発行はできるだけ抑えるようにしている。

3.1　メモリ管理

本システムでターゲットとしたマシンはローカルメモリを持つものだが、すべてのプロセッサから参照できる共有メモリと違い、ローカルメモリは容量が限られると考えられる。そのため、処理できる問題の規模がメモリの容量に左右されかねない。そこで、ローカルメモリの仮想化を行うが、そのスワップ先を共有メモリにとることにする。これは、2 次記憶装置がすべてのプロセッサからアクセスできるとは限らないためである。また、共有メモリについても仮想化を行い、スワップエリアは 2 次記憶装置にとることにする。

3.2　ジョブ管理

プログラマは前述のように単一記憶レベルのモデルに沿ってプログラミングを行なう。これはコンパイラによって変換され、カーネルが実行するプログラムには図 3のような新たにプロセスという単位が導入される。

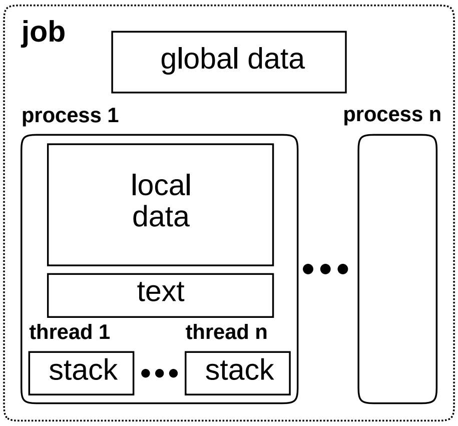

図 3: メモリ空間の共有関係

ここで、プロセスは基本的にはプロセッサと 1 対 1 に対応する単位であり、これが並列実行の単位となる。また、プロセスのマイグレーションはローカルメモリにあるデータの移動を伴うため行なわない。スレッドは並行実行の単位となり、プロセス内でローカルデータを共有する。スレッドのコンテキストスイッチは、プログラムで明示的に指示された場合、もしくはそのスレッドがこれ以上実行を継続できない場合 (ロックの獲得に失敗した場合) に行うことでオーバヘッドを最小限に抑えている。また、ユーザがスケジューラを記述しカーネルに登録することで、ユーザはプログラムに応じたスケジューリングを行うことができる。特に指定しない場合は round-robin が使用される。

本カーネルでは、アプリケーションは単一プロセス・単一スレッドで起動される。その後、プロセス

やスレッドの fork によって別のプロセス・スレッドを起動する。

4 ライブラリ

4.1 データ管理システム

本章では、ローカルメモリと共有メモリの 2 つのメモリシステムを有効に使い分ける方法を提案し、その設計方針について記述する。

4.2 方針

EULASH システムにおいては、主記憶共有型で記述されたプログラム中の変数の配置位置をコンパイラがローカルメモリと共有メモリに振り分ける。しかし、変数の種類によっては共有メモリに置くほどアクセスがなく、かといってローカルメモリに置くほどプロセスにプライベートでもないものがある。従来の方式ではそのようなデータは共有メモリに配置するしかなかったが、このような変数においてもローカルメモリを有効に活用できるように新たなデータ型である mobile 型、write-update 型、write-invalidate 型を提案する。また、これらは次のような特徴を持っている。

- 特定の言語に依存しない。
- ライブラリの形で提供され、システムに手を加えない。
- false sharing を避けるために管理をデータ単位で行う。

以降でこれらの型について説明する。

4.3 mobile 型

共有メモリとローカルメモリを比べた場合、一般的に共有メモリに対するアクセスはローカルメモリに対するそれより高価であるため、変数はできるだけローカルメモリに置くことが望ましい。しかし、2章で述べたようにそれほど共有度が大きくないデータはローカルメモリに配置できるならばそれがよいが、実際には共有されるため従来の方法では共有メモリに配置する必要があった。

ここで、変数の配置位置を動的に変更することができる機構を導入することでローカルメモリの有効活用を目指す機構を提案する。この機構では、初期状態では変数はあるプロセッサのローカルメモリに配置される。その後、他のプロセッサからのアクセスが生じた場合に初めてデータを共有メモリに移動する。このような機構を設けることにより、従来共有メモリに配置されていた変数をローカルメモリに配置することが可能となる。このような機構を mobile データ型と呼ぶ。

4.4 write-update 型および write-invalidate 型

mobile 型ではデータのコピーは存在しない。それは、コピーを認めるとコンシステンシの維持にコストがかかるためである。しかし、メモリアクセスの局所性が高い場合にはコンシステンシ維持によるコスト増加より、データへのアクセス速度の向上によるコスト減少が上回る場合がある。このような場合にデータのコピーを持つ方法について考えてみる。コンシステンシ維持の方法によって 2 つに分けられる。

一つは write-update 型である。これは書き込みを行った場合にはその値をブロードキャストする事ですべてのキャッシングされているデータを更新する。従って、読み出し時にはローカルメモリを参照することで値を得ることができるため、書き込みに比べて読み出しが多い場合に有効である。

もう一つは write-invalidate 型である。これは書き込みを行う際に、他のプロセッサがキャッシングしているデータを無効化するよう指示する。このようにすることで、単一プロセッサでアクセスをしている場合には読み出しだけでなく書き込みもローカルメモリに対して行えばよいため、局所性の高いプログラムに対して有効である。

5 コンパイラ

EULASH でのコンパイラの役割は、2章で述べたようにユーザにより記述された単一メモリモデルのプログラムをカーネルが使用する階層メモリモデルのプログラムへ変換することである。単一メモリモデルでは、スレッドのみが並行処理の単位であるが、階層メモリモデルではプロセスとスレッドの 2 つが単位である。そこで、コンパイラはいくつかのスレッドをひとつのプロセスとしてまとめる。通常、プロセスはプロセッサと一対一に対応

する。プロセス数はコンパイル時にユーザによって与えられる。どのスレッドがどのプロセスに割り当てられるかはスレッド間のデータ依存関係、負荷バランスを基に静的に決定される。

ユーザの記述したプログラムは単一メモリモデルで記述してあるため、データはスレッド内でローカルなデータと共有データに分けられる。ローカルデータはローカルメモリに配置される。1プロセスに複数スレッドが存在する場合、共有データはスレッド間の依存関係によって、プロセス内だけで参照されるデータはローカルデータとして、また、複数プロセスから参照されるデータは mobile 型データとして配置される。mobile 型として配置されたデータはプログラムが1度実行される時にアクセス履歴が保存され、2回目のコンパイル時にそのアクセスパターンに応じて mobile 型、 write-update 型、 write-invalidate 型、 グローバルデータのいずれかに置き換えられる。

6 ATTEMPT-0 への実装

EULASH カーネルは現在、バス結合型並列計算機 ATTEMPT-0[1][8] に実装を行っている。ATTEMPT-0 は図4に示すように最大20枚の PU ボードを Futurebus[3] を用いて接続できる。

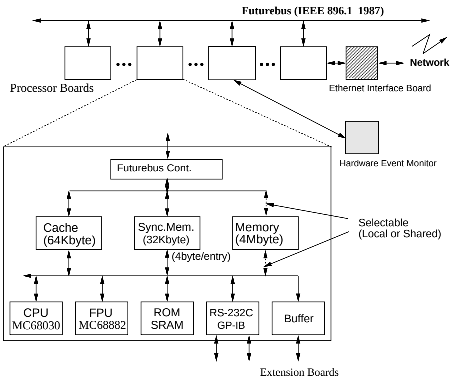

図 4: ATTEMPT-0 の構成

各ボードには MC68030、MC68882 と 4Mbytes のローカルメモリ、64Kbytes のスヌープキャッシュメモリが搭載され、シンクロナイザ[1] と呼ばれる同期機構のための特殊なメモリを持っている。また PU ボード上のコネクタにより、 Ethernet インターフェース、 SCSI インターフェースが接続でき、ハードウェアイベントモニタ等の拡張も行うことができる。

ATTEMPT-0 に特徴的なシンクロナイザはリード、ライトのほかに同期動作として Fetch&Decrement を行え、さらに PU 間割り込みを発生させることができる。さらに、シンクロナイザは一種の分散共有メモリであり、シンクロナイザからのリード動作と0に対する F&D は共有バスを使用せずに行うことができる。

表1にシンクロナイザメモリに対するアクセスタイムを、表2に各メモリに対するアクセスタイムを示す。

表 1: シンクロナイザの動作時間

機能	時間	バス操作
Read(X)	200ns	無し
Write(X,data)	650ns	有り
Fetch&Dec(X)	750ns	有り

表 2: メモリシステムのアクセス時間

アクセスの種類	時間 [ns]	転送 byte 数
ローカルメモリ		
	250	4 bytes
共有メモリ		
Read miss	18200	64 bytes
Read hit	150	4 bytes
Write	750	4 bytes

7 評価

7.1 EULASH カーネルの評価

EULASH カーネルの評価を次の3つのアプリケーションを用いて行った。

jacobi :

N 元一次方程式をヤコビ法を用いて解くプログラム。

サイズ	行列成分
250 元	帯行列

channel :

ニューラルネットワークを用いたアルゴリズムによりチャネル配線問題を解くプログラム[5][6][9]。

問題: Deutch's difficult example

ネット数	レイヤ数	iteration
72	4	100

mp3d :

SPLASH (Stanford Parallel Application for SHared-memory) から、直方体トンネル内での高速希薄流体の障害物、境界、他の粒子との衝突をシミュレートするプログラム。

分子数	ステップ
100	1000

測定には ATTEMPT-0 を使用した。ユーザがすべてを記述し OS 等を使用しないプログラム (original) と EULASH のプロセスを用いて記述したプログラム (EULASH) を使用して測定した。

台数効果は original の 1 PU での実行時間を 1 としたスピードアップ率である。EULASH を用いるとスピードアップ率は低下するが、これはアプリケーションプログラムの記述性、移植性等とのトレードオフであるため許容範囲内だと考えられる。mp3d は他の 2 つにくらべ、original と EULASH の差が大きいが、これは original が ATTEMPT-0 のために特に高速化を施してあるのに対して、EULASH は配布されているソースを特に変更を行なわずに使用したためであると考えられる。

7.2 データ管理システムの評価

ここでは ATTEMPT-0 上での評価結果について述べる。しかし、本研究ではローカルメモリが共有メモリに対し高速であることを前提としているが、ATTEMPT-0 では表 2に示すようにローカルメモリが共有メモリと比較して低速である。特に、共有メモリからの読み込みにおいてキャッシュにヒットした場合は、ローカルメモリの読み込みより高速である。したがって、大幅な性能向上は見込めない。そこで、メモリのアクセススピードと本システムの影響を調べるために、Multiprocessor Instruction Level simuLator MILL[7] を用いての計測も行なった。

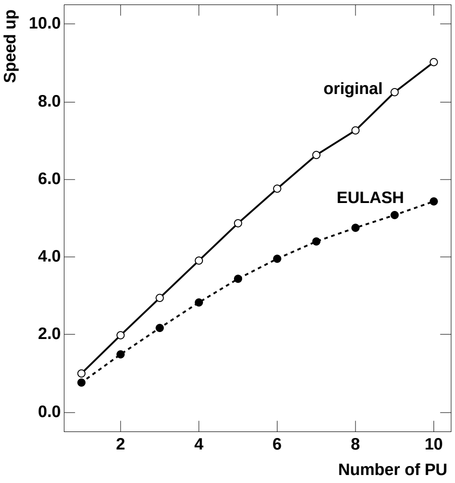

図 5: jacobi の台数効果

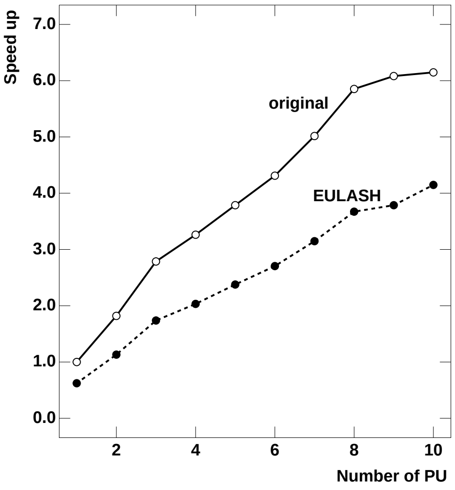

図 6: channel の台数効果

7.2.1 対象とする問題

連立一次方程式 (ヤコビ法)

連立一次方程式をヤコビ法を用いて解く。ヤコビ法は基本的な反復法の一つで、k回目の反復値ベクトル$\boldsymbol{x}^k$を

$$\boldsymbol{x}^k = (\boldsymbol{b} - (\boldsymbol{A} - \boldsymbol{D}) \cdot \boldsymbol{x}^{k-1}) / \boldsymbol{D}$$

によって求める。ここで、 $\boldsymbol{A}$ は係数行列、 $\boldsymbol{D}$ は $\boldsymbol{A}$ の対角成分によって構成される対角行列、 $\boldsymbol{b}$ は方程式の右辺の定数ベクトルである。解の収束判定は、解ベクトルの変化量のノルムで行う。今回のプログラムでは $\boldsymbol{A}$ に mobile 型等の各種データ型を用いている。

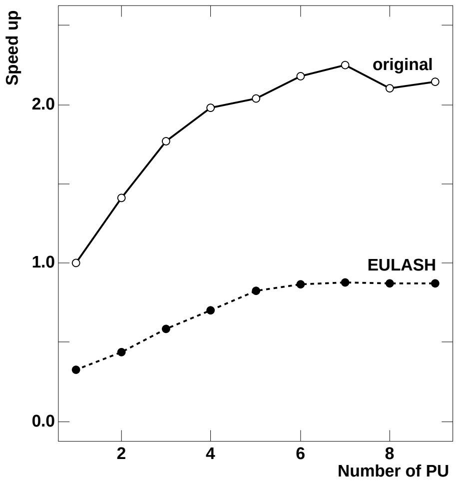

図 7: mp3d の台数効果

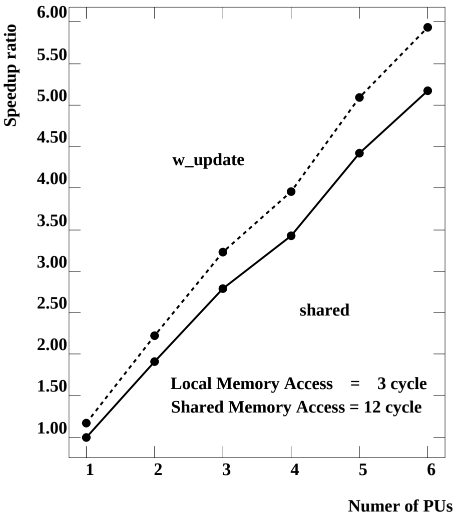

図 9: MILL 上での評価 (ヤコビ法)

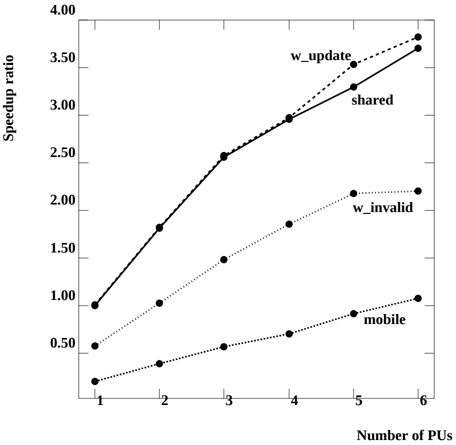

図 8: ATTEMPT-0 上での評価 (ヤコビ法)

7.2.2 ATTEMPT-0 における評価

ATTEMPT-0 上でヤコビ法を実行した結果を図8に示す。これらのグラフでは、mobile 型、write-update 型、write-invalidate 型を用いたプログラムをそれぞれ mobile、w_update、w_invalid と示す。また、データを共有メモリに配置したプログラムを shared で示す。図 8に示すグラフの縦軸は、shared を 1 プロセッサで実行した時を基準とした時のスピードアップ率を表している。

ヤコビ法では共有メモリに置かれるデータは連立一次方程式の係数行列である。これは読み込まれるだけで書き込みは行なわれない。したがって、write-update 型に適していると予想される。ATTEMPT-0 で実行を行なった結果は図 8であるが、予想されたようにわずか 1% 程度ではあるが write-update 型を用いることによって性能の向上が見られた。

今回のヤコビ法では完全に読み込みだけなので、プログラマは write-update 型を用いずともローカルメモリに値を配置することができる。しかし、write-update 型を用いることにより、プログラマはより容易に同様のことを行なうことが出来る。また、完全に読み込みだけでない場合には上のようなプログラマによる高速化は困難であるが、そのような場合にも write-update 型は有効に使うことができると考えられる。

mobile 型と write-invalidate 型では単に共有メモリにデータを配置した場合より低速となった。これは mobile 型も write-invalidate 型も、アクセス時にテーブル検索のためのローカルメモリへのアクセスが余分にかかり、そのコストによるためと考えられる。

7.2.3 MILL における評価

前節で ATTEMPT-0 上で評価を行なった結果、ヤコビ法のプログラムで高速化を行なうことができた。そこで、プログラムについて、メモリのアクセス速度を変化させた場合の影響を調べるため、MILL 上でシミュレーションを行なった結果を図 9に示す。図 9ではローカルメモリのアクセス時間を 3 クロック、共有メモリのキャッシュヒット時のアクセス時間をローカルメモリの 4 倍の時間の 12 クロックに設定した場合の結果である。

その結果 5% ~ 10% の性能向上がみられた。ATTEMPT-0 上では 1% 程度の高速化であったが、アクセス速度の差が開いたためこのような結果が得られたと考えられる。すなわち、ローカルメモリが共有メモリより高速なシステムでは本手法は有効であるといえる。

7.3 まとめ

カーネルでは、実機により評価を行なった。結果は予想されたように、本カーネルを使用することで速度低下が生じたが、これはプログラムの記述性、移植性を考慮すると許容される範囲であると考えられる。

データ管理システムでは、実機とシミュレーションにより評価を行なった。その結果、ローカルメモリの比較的低速な実機でも数%の性能向上が見られ、また、シミュレーションからはローカルメモリが共有メモリより約 4 倍高速な場合において、最大で約 10% の高速化を確認できた。このように、ローカルメモリが共有メモリより高速な場合には、本システムが有効であることが確認された。

8 おわりに

遅い共有メモリと速いローカルメモリの双方を持つ並列計算機において、両方のメモリシステムを効率良く使用することのできるシステムである EULASH を提案した。また、新しいデータ管理機構である mobile 型、write-update 型、write-invalidate 型を提案した。さらに、カーネル、データ管理機構をバス結合型並列計算機 ATTEMPT-0 に実装し評価した。カーネルによって提供されるプロセスを使用したプログラムはプログラマがデータを管理する従来の方法と比較して、 60% 程度の速度だが、記述性、移植性とのトレードオフであり許容範囲内だと考えられる。また、データ管理機構の評価では ATTEMPT-0 が高速なキャッシュを持つにも関わらず、わずかながら速度向上が認められた。シミュレータによる結果では共有メモリがローカルメモリより遅い場合にはさらなる速度向上が得られることが分かり、このデータ管理機構の有用性が認められた。

参考文献

[1] H. Amano, T. Terasawa, and T. Kudoh. Cache with synchronization mechanism. In *IFIP Congress89*, 1989.

[2] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. Mcauliffe, L. Rudolf, and M. Snir. The NYU Ultracomputer - designing an MIMD shared memory parallel computer. *IEEE Trans. on Comput.*, 1983.

[3] IEEE. *IEEE Standard BackPlane Bus Specification for Multiprocessor Architectures: Futurebus.*

[4] D. Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Hennessy, M. Horowita, and M. S. Lam. The Stanford Dash multiprocessor. *Computer*, 1992.

[5] K. Suzuki, H. Amano, and Y. Takefuji. Neural network parallel computing for channel routing algorithms. In *International Conference on Automation, Robotics, and Computer Vision*, 1992.

[6] Y. Takefuji. *NEURAL NETWORK PARALLEL COMPUTING*. KLUWER ACADEMIC PUBLISHERS, 1992.

[7] T. Terasawa and H. Amano. Performance evaluation of the mixed-protocol caches with instruction level multiprocessor simulator. In *IASTED International Conference of MODELLING AND SIMULATION*, 1994.

[8] 鳥居淳, 天野英晴. 並列計算機テストベッド attempt の交信機構の評価. 並列処理シンポジウム JSPP'91 論文集, 1991.

[9] 大和純一, 鈴来響太郎, 天野英晴, 武藤佳恭. ニューラルネットワークを用いた最適化アルゴリズムの並列計算機への実装と評価. 信学技法 CPSY92-3, 1992.