# 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

佐藤三久[†] 児玉祐悦[†]
坂井修一[††] 山口喜教[†]

EM-4は，高速なデータ駆動機構を持つ分散メモリの並列計算機である．データ駆動機構により，プロセッサ間において，高速なスレッド起動，同期，リモートメモリアクセスの機能が提供されている．EM-4の並列プログラミング言語であるEM-Cでは，各プロセッサのローカルメモリをまとめて，グローバルアドレス空間として，データ分散・参照することができる．本稿では，EM-4でのいくつかの典型的な共有メモリプログラムを実行した結果について報告する．効率的な実行をするためにリモートメモリアクセスに伴うレーテンシを，ループなどを並列化し，マルチスレッド化することにより隠蔽した．その結果，スレッド数を調整することにより，比較的よい速度向上率を得ることができた．リモートメモリアクセスのための命令による静的なオーバヘッドと，プロセッサ数が大きくなるにしたがって増えるネットワークの動的なオーバヘッドが明らかになった．EM-4では1つのプロセッサ内のスレッド数を過度に増やすことによって，逆にネットワークの混雑させ，性能が低下するのが見られた．さらに，現在開発中のEM-4の次期マシンEM-Xのアーキテクチャと共有メモリプログラムの効率的な実行をサポートする機構についても述べる．

# Experience with Executing Shared Memory Programs using Fine-Grain Communication and Multithreading in EM-4 Multiprocessor

MITSUHISA SATO,[†] YUETSU KODAMA,[†] SHUICHI SAKAI [††]
and YOSHINORI YAMAGUCHI [†]

In this paper, we present our experience and results obtained from executing shared memory application programs using fine-grain remote memory access communication and multithreading in the EM-4 multiprocessor. The EM-4 is a distributed memory multiprocessor which has a dataflow mechanism. The dataflow mechanism enables a fine-grain communication packet through the network to invoke the thread of control dynamically with very small overhead, and is extended to access remote memory in different processors. The EM-4 parallel programming language EM-C provides the notion of a *global address space* and parallel constructs for exploiting medium-grain parallelism to tolerate several remote operation latencies. We hide the remote memory access latencies with multithreading. The benchmark results show that shared memory applications achieve reasonable speedup with four to eight threads in the EM-4 prototype. We found that aggressive multithreading can negatively affect its network interface and increase the network contention. We also describe the EM-X multiprocessor, the next generation of EM-4.

## 1. はじめに

並列プログラミングにおいて，共有アドレス空間あるいは共有メモリモデルを提供することは重要な課題である．共有メモリモデルにおいては，プログラマはプロセス間で共有するデータを陽にメッセージ通信によって移動，同期する煩わしい操作を避けることができる．ハードウェアにより実現された共有メモリアーキテクチャは多くのプロセッサを結合する場合，複雑になることが多く，分散メモリアーキテクチャはその構成の容易さから多くのプロセッサを持つ場合には有利であるとされている．しかし，分散メモリモデルでのメッセージ通信によるプログラムでは，データの共有をメッセージで行うように送り手，受け取り手を意識して，再構成しなくてはならない．従来のメッセージ通信による分散メモリ型アーキテクチャでは，共有メモリ空間をコンパイラあるいは低レベルの実行時ルーチンによって実現する方法が多く提案されている．

† 電子技術総合研究所
Electrotechnical Laboratory
†† RWCつくば研究センタ
RWC Tukuba Research Center

並列プログラムの通信パターンが静的かつ規則的な場合には，プログラマあるいは，最近のコンパイラ技術により，共有メモリプログラムの共有するデータを大きなメッセージにして通信することで効率的な実行ができる．しかしながら，データの共有が細粒度かつ動的に変化する場合には，細粒度のメッセージ通信が必要となり，大きなオーバヘッドを伴う従来のアーキテクチャでは効率的な実行は困難になる．

本論文では，並列計算機 EM-4 のリモートメモリアクセス機構と高速なスレッドの生成機構を用いて，いくつかの典型的な共有メモリプログラムの実行を行ったので，そのプログラミングと性能について，述べる．

EM-4[9]は，データ駆動機構をもつ分散メモリ型の並列計算機である．データ駆動機構は，プロセッサ間の通信によって，それに対応するスレッドを高速に起動・同期することを可能にしている．プロセッサはネットワークに対してメッセージを直接送りだしたり，受けとったメッセージに対するスレッドの起動をハードウェアの機構としてサポートしている．EM-4 では，この機構はデータフロー実行だけでなく，遠隔のメモリの読み出し・書き込みにも拡張されている．EM-4 は，基本的には分散メモリのマルチプロセッサであり，各プロセッサはローカルなメモリをもち，グローバルな共有メモリはもたない．しかし，メッセージにはシステムで定義されている特定のスレッドを実行する機能があり，これを用いて，他のプロセッサのメモリに対してリモートアクセスすることが可能である．

我々は，共有メモリプログラムを実行するために，細粒度のメッセージ通信を用いて，リモートメモリアクセスを行い，動的なデータ共有を実現した．リモートメモリアクセス時には，通信によるネットワークのレーテンシが生じるが，個々のプロセッサの中でも並列プログラミングを行い，複数のスレッドを用いて，他のスレッドにより，このレーテンシを隠蔽することが可能である．EM-4 のデータ駆動機構は高速なコンテキストスイッチの機構として働くため，マルチスレッド化により効率的なリモートメモリアクセスのレーテンシ隠蔽の効果が期待できる．

EM-4 でのプログラミングには並列プログラミング言語 EM-C[2]を用いた．EM-C は，すべてのプロセッサのローカルメモリをアドレスづけするグローバルアドレス空間を提供している．プログラマはこの空間に共有するデータを分散配置し，グローバルアドレスを用いて，参照することができる．コンパイラはこれらのグローバルなアクセスに対して，リモートメモリのメッセージを用いて参照するコードを生成する．また，言語の構文として**タスクブロック**を導入し，プログラマが medium-grain のタスクを用いて，並列性を記述することができる．これにより，マルチスレッド化し，リモートの操作のレーテンシ隠蔽を行うことができる．

本論文では，典型的な共有メモリ並列プログラムの1つとして，共有メモリアプリケーションベンチマークセットである SPLASH[10]の中から，並列スタンダードセルルータ LocusRoute と粒子シミュレーション MP 3 D を取り上げ，その EM-4 での実行と性能について報告する．例えば，LocusRoute では1つの配列をすべてのプロセス（プロセッサ）が最適解を求めてアクセスを行い，求められた解で更新するが，このようなアルゴリズムではあらかじめどの情報が共有されるかを知ることは困難である．このような情報を共有するには，頻繁な更新情報の交換を行うことになる．これらのアプリケーションはデータの共有は動的であり，従来のメッセージ通信型のアーキテクチャでは実行するのは非常に困難なものである．

まず，2章において，EM-4 のアーキテクチャについて述べ，3章において，EM-4 のプログラミング環境である EM-C について述べる．4章において，上に述べたアプリケーションプログラムの実現とその実行結果を報告し，考察を行う．我々は現在，EM-4 で得られた経験を基に，並列計算機 EM-X を開発している．5章では，今回の評価から得られた知見を基にどのように EM-X を改良するかについて議論する．6章において，関連研究について述べる．

## 2. 並列計算機 EM-4

EM-4 プロトタイプは，80 プロセッサ（PE）から成り，EM-4 の要素プロセッサ EMC-R[14]は，サーキュラオメガネットワークで接続されている．それぞれのプロセッサは，ローカルなメモリを持っており，分散メモリ型のマルチプロセッサである．

EMC-R は，RISC アーキテクチャを採用し，パイプラインは逐次の命令実行とデータ駆動機構のためのネットワークとの通信・同期処理が融合されるように設計されている．ネットワークに対して直接，メッセージを送ったり，メッセージでデスパッチできる．EMC-R は，**図1**に示すようにネットワークのルーテングを行うスイッチングユニット(SU)，メッセージを格納する入力バッファユニット(IBU)，命令を起動する待ち合わせユニット（FMU），命令実行ユニット（EXU）から構成されている．

メッセージは，アドレス部とデータ部からなる2ワ

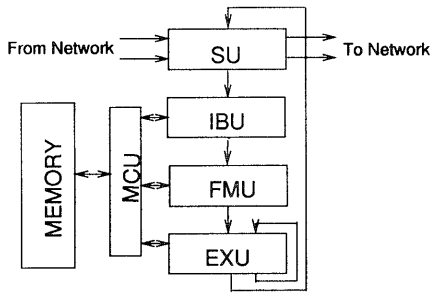



**図1** EMC-R プロセッサ
Fig. 1 The EMC-R processor.

ードの固定長で，これをパケットと呼んでいる．パケットが到着するとデータ駆動機構によって，そのアドレス部で指定されるスレッドがデータ部にある値と共に起動される．そのスレッドが終了すると，次のパケットがキューの中から取り出され，処理される．パケットは，データフロートークンとして解釈することができる．パケットにおいてマッチングの属性を指定すると，他方のデータフロートークンが到着していない場合は，そのパケットは起動するスレッドに対応するメモリに退避され，他方のトークンが到着するとそれらのデータと共にスレッドを起動する．スレッドの終了は，命令フィールドにおいてプログラム中に明示され，スレッドが終了する前に必要なレジスタの値などを実行中のスレッドに対応する関数フレーム（逐次実行の場合のスタックにあたる）に退避しておくようにする．

新たなスレッドを起動するには，関数のためのフレームを割り当て，そのフレームで実行するコードに対して，パケットを送出すればよい．この機構を利用して，スレッドの生成(fork)，リモートプロシージャコールなどの基本操作を行っている．

いくつかの機能を実現するためにパケットの解釈をソフトウェアで定義することができる．パケットにおいて，パケットタイプのフィールドを指定することによって，パケットタイプに対応するスレッドが実行される．この機能を用いて他の PE のメモリの書き込み，読みだしを行うことができる．例えば，リモートメモリの読みだしの場合は，指定されたアドレスを読み，それをリクエスト側にメッセージにして返すというごく簡単なスレッドを目的のプロセッサで実行させることでできる．

EM-4 プロトタイプは，12.5 MHz のクロックで動作しており，メモリ参照命令などを除く大部分の命令は1クロックサイクルで実行される．各プロセッサは，2×2 のネットワークノードになっているが，ネットワークの性能は PE のポート当たり 60.9 Mbytes/sec である．ネットワークに負荷がない状態では，リモートメモリの読みだし，書き込みはそれぞれ 1.85 $\mu$sec，0.40 $\mu$sec（80 プロセッサの平均）で実行できる．

## 3. 並列プログラミング言語 EM-C

我々は，EM-4 のプログラミングのために C 言語の superset である EM-C[2]を開発した．EM-C はリモートメモリアクセスの機能を用いて，プログラマに対して，仮想的な共有メモリを提供している．これは，すべてのプロセッサのローカルメモリをアドレスづけできる空間であり，これをグローバルアドレス空間(global address space)と呼んでいる．プログラマは，この空間上にデータを分散配置し，共有メモリとして直接アクセスすることができる．これによって，従来のメッセージ通信ではプログラムが困難だった共有メモリ並列プログラムを容易に実現することができるようになった．EM-C は以下の特徴を持つ：

**グローバルアドレス空間**—同じ PE で実行されるスレッドは，同じローカルメモリ空間を共有する．プログラム中で宣言されるデータは，各プロセッサ上のローカルメモリの同一アドレスに配置される．EM-C では，データの宣言に対して，global キーワードを付け加えることによって，プログラマはグローバルアドレス空間上に配列などのデータを分散配置し，直接アクセスすることができる．コンパイラは，グローバルメモリのアクセスに対して，リモートメモリアクセスのコードを生成する．

**スレッド基本操作の inline 化**—スレッド基本操作として，スレッドを生成する fork やリモートプロシージャコールなどの基本関数は数命令に inline 展開され，効率的に実行される．

**タスクブロック**—EM-C では，medium-grain の並列性の記述を構文として提供している．

*task-prefix*[forkwith | dowith] (*parameters for task body*) {*task body*... }

forkwith 構文は，一連のコード(これをタスクブロックと呼ぶ）を実行するスレッドを生成する．パラメータリストとして，タスクブロックから参照する変数を指定する．これらの変数は，タスク生成時にコピーされる．forkwith の代わりに dowith を使うことによって，元のスレッドはタスクブロックが終了するまで，待つことになる．*task-prefix* を指定することにより，タスクブロック単位に以下のような制御ができる：

**where(p)** グローバルアドレス p で示されるプロセッサでタスクブロックを実行する．

**iterate(n)** 同じタスクブロックを実行するnスレッドを生成する．マルチスレッドによるループ並列実行に用いる．

**everywhere** すべてのプロセッサで，タスクブロックを実行する．これは，SPMDプログラムの実行に用いる．

生成されたタスクのスレッドは，I-structure，Q-structure[1]，バリア同期，ロックなどの同期機構を使って，他のスレッドと同期・通信することができる．

## 4. 共有メモリプログラムの実行と結果

本章では，SPLASH benchmark set[10]のLocusRouteとMP3Dについて，EM-Cでのプログラミングと EM-4プロトタイプにおいて得られた性能について述べる．元のプログラムをEM-Cを用いて，EM-4で実行できるように書き換えた．まず，各プロセッサから共有されるデータをグローバルアドレス空間に配置し，リモートアクセスするようにした．次に，並列構文を用いて，ループを並列化，個々のプロセッサ内においても複数スレッドで実行し，リモート操作のレーテンシを隠すようにした．なお，これらのプログラムは共有メモリプログラムの1つとして取り上げたもので，プログラム自身のアルゴリズムについて議論するものではないことを断っておく．

### 4.1 LocusRoute

LocusRoute[8]はチャネル法を用いたスタンダードセルルータである．入力データとして，接続するピン・グループ（セルの上下の2つのセットの場合がある）の集合がワイヤのデータとして与えられる．中心となるデータ構造はコスト表と呼ばれる配列である．チャネル内の各配線格子点に対応した要素からなるコスト表を用いて，同じ格子点上に配線されているルートの数をコストとして，なるべく多くのルートが重ならないようにルーテングを行う．結線するピンのペアをminimum spanning treeアルゴリズムで決定し，それぞれのピンのペアに対して，コストが最小になるルートを見つけ，それをコスト表に反映する．このプロセスを数回繰り返すことにより，適当な解を求める．実際のベンチマークプログラムでは，2回繰り返す．

プログラムのアルゴリズムは，3レベルの直交した並列性を持つ：

1. ワイヤレベルの並列性：それぞれのワイヤはそれぞれのプロセッサでルーテングができる．
2. セグメント間の並列性：1つのワイヤにおいて，接続を行うピンの対は並列にルートを捜すことができる．

```
ワイヤーデータを読み込み、PEに分散。
for (2回繰り返す){
 everywhere dowith(){ /* 全PEで実行開始 */
  iterate(ワイヤレベルのスレッド数(#w))
   dowith(){
    一つワイヤをとる；(前回の結果を取り消す)
    どのピンを結ぶかを決定;
    while(すべてのピンペアについて){
       iterate(ルートレベルのスレッド数(#r))
          dowith(){  一つのルートをとり、
              コスト配列にリモートアクセスし、
              コストの計算をおこなう;}
              最小コストのルートを選択;}
        コスト配列に選択されたルートを反映;}
 /* 負荷分散 */
 他のPEのワイヤをコピーして、処理; }}
```

図2 EM-CによるLocusRouteの概略
Fig. 2 Outline of LocusRoute implementation in EM-C.

3. ルーテングの並列性：ピンの対について，最適なルートの探索は並列に行うことができる．

EM-4では，以下のように並列化を行った[2]．

**データの割り当て**—ワイヤの入力データは，round-robinに各プロセッサに割り当てる．このデータはプロセッサ内で局所的にアクセスされる．コスト表は，全プロセッサにインタリーブされるように割り当てた．すべてのプロセッサからリモートメモリアクセスにより，参照・更新される．

**マルチスレッドによる実行**—各プロセッサにおいて，複数のスレッドを用いて，ワイヤの処理を同時に行うことにより，ワイヤレベルの並列性を利用して，リモートメモリのレーテンシを隠蔽する．また，1つのワイヤに関しても，ルートの探索は複数のスレッドを動的に生成して行った．

**負荷分散**—プロセッサに当てられたワイヤをすべてルーテングし終えたら，他のプロセッサのまだルーテングされていないワイヤを移動し，ルーテングし，負荷分散を行う．このワイヤデータのコピーはリモートメモリアクセスを用いて行うことにより，メッセージパッシングを用いるよりも容易に行うことができる．

図2にEM-Cプログラムの概略を示す．すべてのプロセッサでワイヤ処理を起動するために，everywhere構文を用いる．プロセッサ内でコスト配列に対するリモートメモリアクセスのレーテンシを隠蔽するために，iterate構文を用いて，ワイヤ処理とルート探索のループを複数スレッドを動的に生成し，並列ループ実行する．元のプログラムでは多くの大域変数が用いられていたが，並列実行するために局所変数になるように書き換えた．

EM-Cプログラムでは，関数呼び出しあるいはリモートメモリアクセス時にスレッドの切り替えを行うが，各ワイヤの処理において，ピン間の接続を決定するminimum spanning treeの計算ではスレッドの粒

度が大きくなり，ルート探索ではリモートメモリアクセスを多く行うため，粒度が小さくなる．EM-4 ではリモートメモリアクセスの処理も他のスレッド処理と同じようにスケジューリングされるため，たまたま，粒度の大きいスレッドが実行されていた場合，リモートメモリアクセスも長時間待たされてしまうことになり，性能を低下させてしまう恐れがある．そのため，minimum spanning tree の計算内のループにスレッドを中断するコードをプログラム内にいれ，これを避けなくてはならなかった．

### 4.2 MP3D

MP 3 D は，3 次元の粒子シミュレーションプログラムである．プログラムは，空間をセルに分割し，時間ステップごとに各粒子の速度と位置を計算し，同じ空間セルにある粒子との確率的なモデルを用いて衝突をシミュレーションをすることを繰り返す．

元の共有メモリプログラムは，粒子をプロセッサごとに分割し，空間セルを共有メモリ上で共有するものである．EM-C プログラムでは，以下のように並列化した．

**データの割り当て**—元のプログラムと同様に粒子を各プロセッサに静的に割り当てる．空間セルの配列を全プロセッサにインタリーブして割り当てる．空間セルはそれぞれのプロセッサからリモートメモリアクセスで参照される．メインループでは，それぞれの粒子は局所的に参照されるが，衝突が検出された場合，相手の粒子をリモートに参照する．

**マルチスレッドによる実行**—それぞれのプロセッサ内でも，複数のスレッドを用いて，同時に複数の粒子の移動更新の計算を行うことによって，空間セルに対するリモートメモリアクセスのレーテンシを隠蔽する．

時間ステップの終りにおいて，バリア同期でプロセッサの同期を行う．粒子について計算量はほぼ一定であるため，動的な負荷分散は行っていない．衝突回数

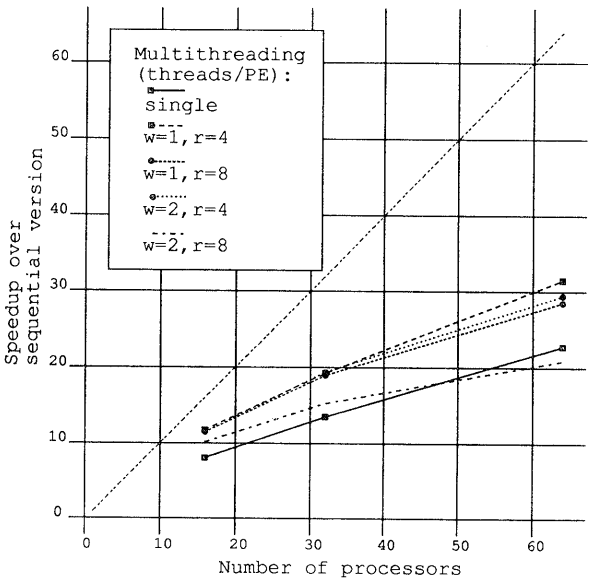



図3 LocusRoute の速度向上率
Fig. 3 Speedup of LocusRoute in EM-4.

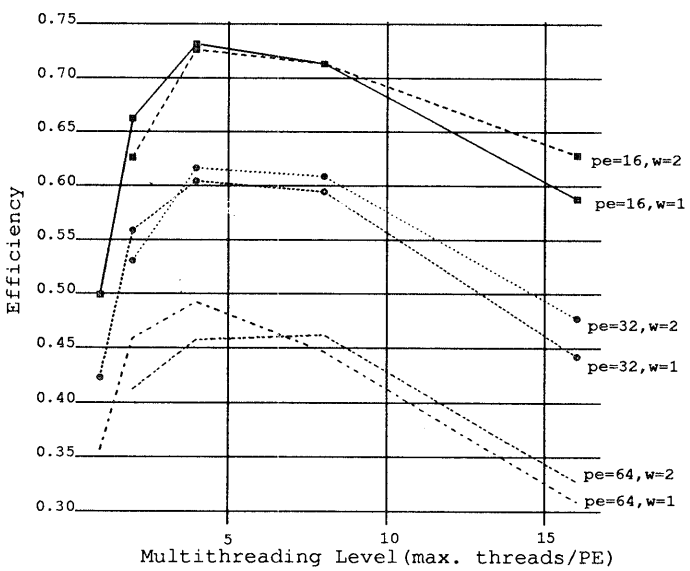



図4 LocusRoute の対逐次効率
Fig. 4 Efficiency of LocusRoute in EM-4.

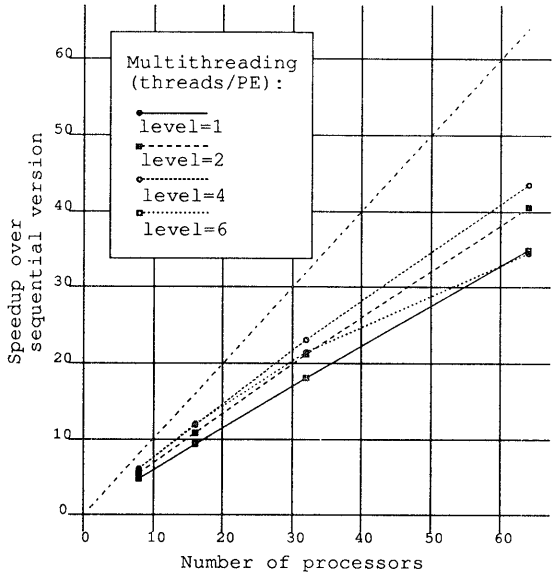



図5 MP 3 D の速度向上率（浮動小数点エミュレーション）
Fig. 5 Speedup of MP 3 D (with floating point emulation).

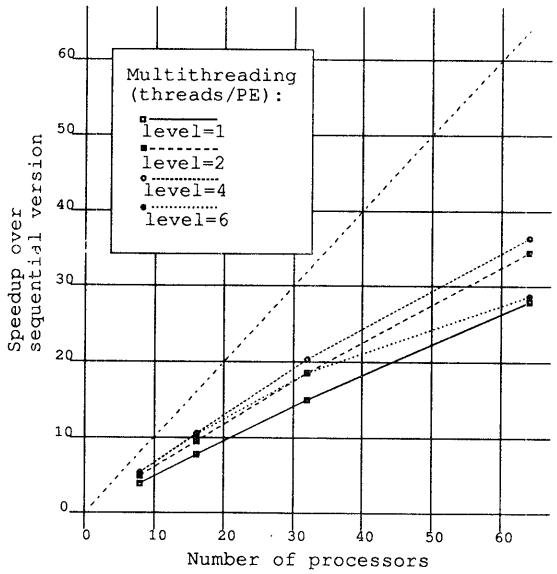



図6 MP 3 D の速度向上率（浮動小数点命令）
Fig. 6 Speedup of MP 3 D (with floating point instruction).

などの統計情報は，バリア同期とともに reduction 計算を行うことにより，計算する．

### 4.3 実行結果

LocusRoute の実行結果として，プロセッサ数を変えた場合の速度向上率を**図3**に示す．LocusRoute での入力データは 3,817 ワイヤ，1,920 グリット，20 チャネルの回路 (Primary 2) である．ちなみに逐次プログラムの実行時間は 251.1 sec である．アドレス計算の都合上，80 PE のうち，64 PE までのプロセッサを用いた．マルチスレッドによる実行の効果を見るために，プロセッサ内のスレッド数 (Multithreading level) を変化させた．**図4**に対逐次効率を示す．ワイヤレベルの並列性のスレッド数 ($w$)，ルーテングレベルの並列性のスレッド数 ($r$) を変化させた．1 つのワイヤについて，ルーテングのスレッドを動的に生成するため，($w=2$, $r=8$) では最大 16 スレッドが実行されていることになる．図のスレッド数はこの最大のスレッド数としている．

MP 3 D は 14×24×17 の空間中に 3000 粒子をおき，10 ステップ実行した．現在，EM-4 には浮動小数点ユニットがないため，実機において，浮動小数点演算はソフトウェアでエミュレーションして得られた結果とハードウェアの浮動小数点ユニットを仮定したクロックレベルシミュレータを用いた結果をそれぞれ**図5**と**図6**に示す．逐次プログラムの実行時間は，それぞれ 6.8 sec，1.3 sec であった．これらについても，1 プロセッサ内の並列ループ実行のスレッド数 (level) を変化させた対逐次効率を**図7, 8**に示す．ソフトウェアによる浮動小数点のエミュレーションは，ハードウェアで行う場合に比べて，演算にもよるが数十倍の時間がかかる．

なお，基準となる 1 プロセッサの実行時間はリモートメモリアクセスを含まない逐次実行のものである．図からわかるように，LocusRoute では 4 から 8 スレッド，MP 3 D では 4 スレッドぐらいと比較的少ないスレッドでの効率が良い．逆に多くのスレッドは，ネットワークを混雑させ，性能を低下させるのが観察された．この理由については，次節において詳しく考察する．プロセッサ数が多くなるにしたがって，よい効率が得られるスレッド数は少なくなる．これはプロセッサが多くなるにつれて，大きなネットワークの容量が必要となり，ネットワークを混雑させるからである．また，2 つのバージョンの MP 3 D を比較して見ると，浮動小数点ハードウェアを持つ場合の方がリモートメモリアクセスなどによるスレッド切替え間の粒度が細かくなるため，少ないスレッドで性能が飽和するのがわかる．

### 4.4 考　　察

実験結果を詳しく解析するために，**図9, 10**にそれぞれのプログラムの Marginal 効率を示す．Marginal 効率とは，少ないプロセッサ台数の時の効率を 1 として，あるプロセッサ台数の時の相対的な効率を表したものである．1-8 の効率は逐次からの効率であるから，並列化のコードによるオーバヘッドによる効率低下を

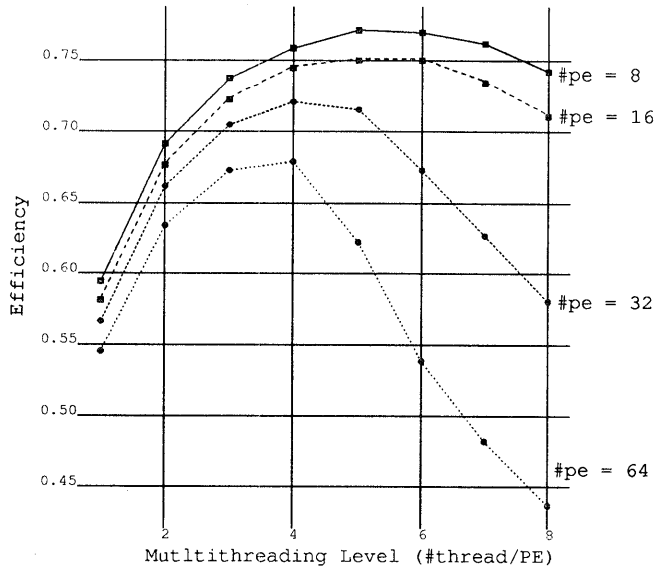



図7 MP 3 D の対逐次効率（浮動小数点エミュレーション）
Fig. 7 Efficiency of MP 3 D (with floating point emulation).

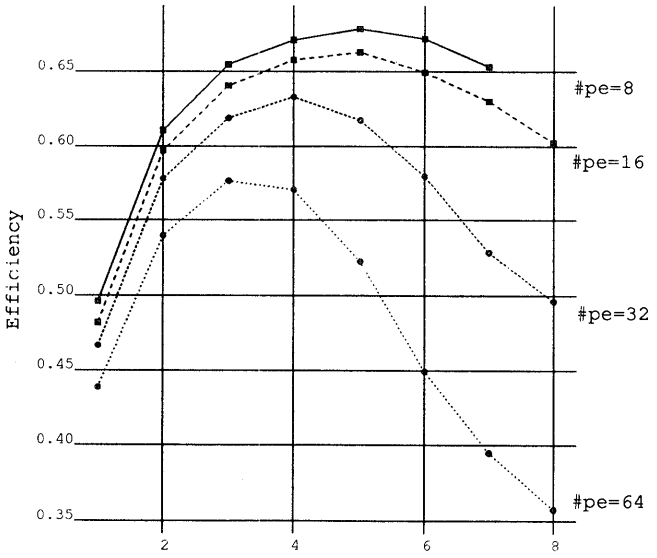



図8 MP 3 D の対逐次効率（浮動小数点命令）
Fig. 8 Efficiency of MP 3 D (with floating point instruction).

考察することができる．逐次版では単なるメモリ参照のコードが，並列コードではグローバルメモリの参照となり，現在の live なレジスタの値をメモリに退避するオーバヘッドも伴う*．また，EM-4 ではリモートメモリアクセスのためにはパケットを生成するために数命令必要となっている．このオーバヘッドは命令セットの工夫により減少させることができる．例えば，J-machine[4]ではこれに類した命令が用意されている．

それ以降の効率は並列コード同士の比較であるから，主にネットワークのオーバヘッド，リモートメモリアクセス自身による低下と見ることができる．スレッド数が適度である場合には，32 プロセッサまでは効

* コンパイルには基本ブロックのみのレジスタ割り当てを行うようにしている．大域的なレジスタ割り当てまで行った場合は逐次プログラムの実行時間は，並列プログラムよりも，減少することが予想される．

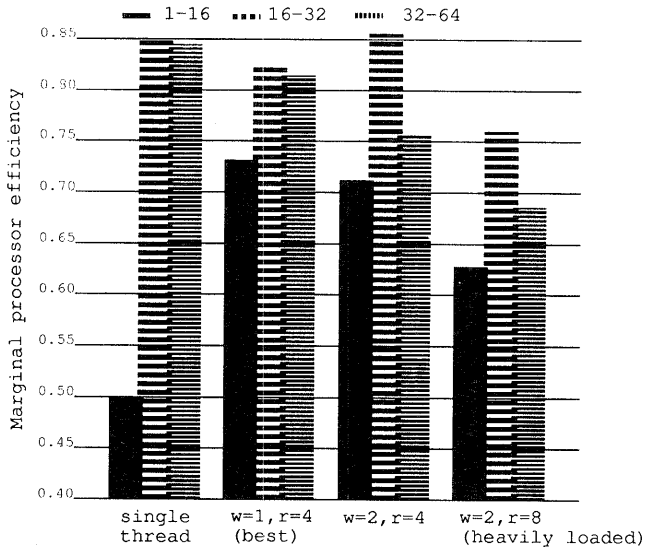



図9 LocusRouteのMarginal効率
Fig. 9 Marginal efficiency of LocusRoute.

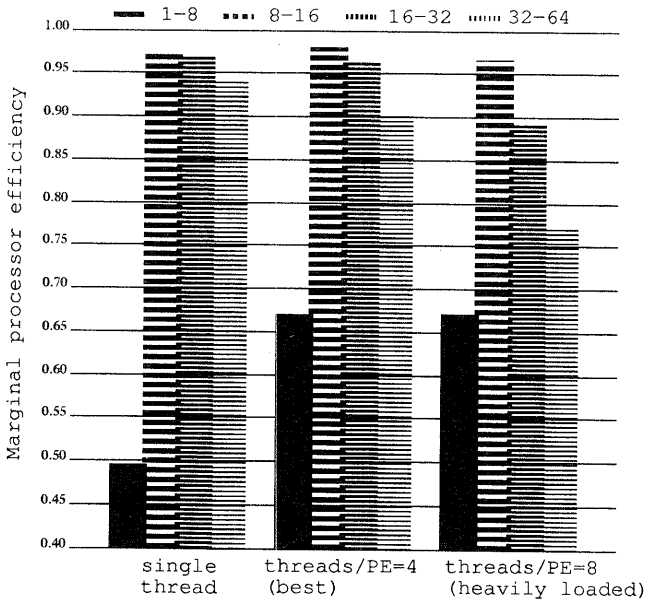



図10 MP 3 Dの Marginal 効率（浮動小数点命令）
Fig. 10 Marginal efficiency of MP 3 D (with floating point instruction).

率が比較的良いことがわかる．しかし，スレッド数が多くなるにしたがって，効率低下の度合が大きくなっているのがわかる．ネットワークの利用率を上げる場合に起こるこのような問題点については，プリフェッチなどのレーテンシ隠蔽の場合にも，指摘されている[11]．

EM-4では，リモートメモリアクセスのパケットは通常パケットと同じく，FIFO順に処理されるため，スレッドの粒度にバラツキがあるとレーテンシが隠蔽できないことが多くなってしまう．また，1つのプロセッサにリクエストが集中すると命令実行中にプロセッサ内のパケットバッファに取り込むことができなくなり，ネットワークの性能を低下させることもわかっている．さらに，この性能低下の原因の1つとして，EM-

4のネットワークインタフェースの問題点が挙げられる．EM-4では，ネットワークから自プロセッサあてのパケットを受けとるIBU (Input Buffer Unit) に16パケット分のバッファがあり，それ以上のパケットを受けとる場合には，メモリ上のパケットバッファに退避される．しかし，プロセッサが実行中の場合はプロセッサの命令実行の方のメモリアクセスが優先されるため，入力されているパケットは退避できず，したがって，ネットワークにパケットが残ってしまい，ネットワークを混雑させる大きな原因になっている．

## 5. 並列計算機 EM-X

我々は現在，EM-4の次世代マシンとして，並列計算機EM-X[6]を開発している．EM-4での経験を踏まえて，以下のような改良を行っている．

- 前にも述べたとおり，EM-4では簡単なリモートメモリアクセスのメッセージが通常のパケットと同等に，FIFO順でスケジューリングされるため，長い実行時間を必要とするスレッドが前にあった場合，リモートメモリ参照が待たされてしまう．EM-Xではそのような事態を避けるため，リモートメモリ参照パケットについては，ネットワークインタフェース部において，命令実行部とは独立に，ハードウェアで直接メモリ参照を行う機能を加えた．これによって，リモートメモリ参照のレーテンシを軽減し，かつ，バラツキを抑えることができる．
- 入力バッファからのパケットをチップ内のバッファからメモリに，退避する場合には，命令実行部からのメモリ参照よりも優先度を高くし，ネットワーク上のパケットを速やかに取り込み，ネットワークの混雑を低減した．
- EM-4では，パケット出力時にプロセッサ上のネットワークノードにパケットが存在する場合には，出力できず，プロセッサのパイプラインがストールする．EM-Xでは，新たに出力側にもバッファ (8パケット分程度) を加えて，命令実行パイプラインがネットワークの影響を受けにくくした．
- パケットに対して，2レベルの優先度を設け，関数起動時のリモートの関数フレームの割り当てなどの操作を優先的に処理することによって，関数の起動を効率化できる．また，負荷分散が必要なアプリケーションに関しては，負荷分散のための情報を優先的に処理することによって，効率的な実行が期待できる．
- リモートアクセスのためのパケット生成のための命令も工夫し，2命令サイクル程度で行うことができる．

これらの改良を行い，共有メモリプログラムをさらに効率的に実行できる見込みを得ている[15]．また，ネットワークインタフェースの入力バッファ，出力バッファに関する最適化は，論文13)において検討している．なお，ネットワークのバンド幅の制限による性能低下は基本的なオーバヘッドとしてのこるが，本論文でとり上げたプログラムについての性能の改良の評価については，まだ開発中のため，残念ながら，行われていない．

## 6. 関連研究

SPLASHベンチマークでの性能は多くの共有メモリアーキテクチャにおいて，評価されている．本論文でとり上げたプログラムはその中でも，いずれも性能的に厳しいものである．例えば，LocusRouteではコスト配列について頻繁に書き換えが起きるため，共有メモリアーキテクチャでは性能は良くなく，例えば，DASH[7]では16プロセッサで10倍程度になっている．

キャッシュをもつ共有メモリアーキテクチャにおいて，マルチスレッドとレーテンシの隠ぺいに関する研究[3),5),12)]が多く行われている．それらで用いられているモデルはキャッシュがミスした時点で，コンテキストの切り替えが起こるものであるが，EM-4ではそのようなキャッシュはなく，すべてのリモートメモリアクセス時にスレッドを切り替える．コヒレントキャッシュを使う従来の共有メモリアーキテクチャと比較すると，本論文でのEM-4の方式はリモートメモリアクセスの場合には必ずメッセージを送出しなくてはならないため，大きなネットワーク容量を必要とする[3]．コヒレントキャッシュを用いることにより，データはローカルに再利用することができることがあり，読みだしのデータが多い場合には効率的な実行が可能である．しかし，頻繁に共有データの書き換えが起こる場合には，キャッシュ更新の情報が飛び交うようになり，性能が低下する．

これまでの研究の多くはトレース駆動シミュレーションでの結果であり，ネットワークなどの要素が考慮されていないものが多い．実際のハードウェアではネットワークのトポロジやバンド幅などの物理的な制限が大きな影響をあたえる．本論文での結果はEM-4プロトタイプの実際のハードウェアを用いて得られたものである．一部の結果はシミュレーションから得たも

のであるが，これも実際のハードウェアを基にしたクロックレベルのシミュレーションである．

## 7. おわりに

EM-4において，リモートメモリアクセス機構とマルチスレッドを用いて，共有メモリプログラムの実行した結果と解析結果について述べた．EM-4のプログラミング言語EM-Cは，リモートメモリアクセスのためのグローバルアドレス空間を提供しており，タスクブロック構文により，medium-grainの並列性を記述することができる．

いくつかの典型的な共有メモリプログラムを取り上げ，マルチスレッドによる実行により，適当な数のスレッドを使ってリモートメモリアクセスのレーテンシを隠蔽できることを確認した．分散メモリでありながら，他の従来の共有メモリプロセッサの実行結果と比べても比較的良い速度向上率を達成しているといえる．性能低下の原因として，リモートメモリアクセスを行う命令自体の静的なオーバヘッドとネットワークの動的なオーバヘッドがあることがわかった．

現在開発中のEM-Xにおいて，前者については，パケット生成命令などの命令セットを工夫することによって，低減している．また，後者については，パケットバッファのオーバフローを命令実行に優先して処理をしたり，出力バッファを設けるなどのネットワークインタフェース部の改良により，改善が期待される．なお，EM-Xは95年には稼働の予定である．

**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，太田情報アーキテクチャ部長ならびに計算機方式研究室の同僚諸氏に感謝いたします．

## 参考文献

1) 佐藤, 児玉, 坂井, 山口：高並列計算機EM-4における分散データ構造を用いたマルチスレッドプログラミング，情報処理学会研究報告，ARC, 92-7, pp. 1-8 (1992).
2) 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, JSPP '93論文集, pp. 183-190 (1993).
3) Boothe, B. and Ranade, A.: Improved Multithreading Techniques for Hiding Communication Latency in Multiprocessors, *Proc. of the 19th Annual International Symposium on Computer Architecture*, pp. 214-223 (May 1992).
4) Dally, W.J. et al.: The J-Machine: A Fine Grain Concurrent Computer, *Proc. of IFIP Congress*, pp. 1147-1153 (1989).
5) Gupta, A., Hennessy, J., Gharachorloo, K., Mowry, T. and Weber, W.-D.: Comparative Evaluation of Latency Reducing and Tolerating Techniques, *Proc. of the 18th Annual International Symposium on Computer Architecture*, pp. 254-263 (1991).
6) Kodama, Y., Koumuara, Y., Sato, M., Sakane, H., Sakai, S. and Yamaguchi, Y.: EMC-Y: Parallel Processing Element Optimizing Communication and Computation, *Proc. of 1993 ACM International Conference on Supercomputing*, pp. 167-174 (1993).
7) Lenoski, D., Laudon, J., Joe, T., Nkahira, D., Stevens, L., Gupta, A. and Hennessy, J.: The DASH Prototype: Implementation and Performance, *Proc. of the 19th Annual International Symposium on Computer Architecture*, pp. 92-103 (May 1992).
8) Rose, J.: The Parallel Decomposition and Implementation of an Integrated Circuit Global Router, *Proc. of PPEARS 88*, pp. 138-145 (1988).
9) Sakai, S., Yamaguchi, Y., Hiraki, K., Kodama, Y. and Yuba, T.: An Architecture of a Dataflow Single Chip Processor, *Proc. of the 16th Annual International Symposium on Computer Architecture*, pp. 46-53 (June 1989).
10) Singh, J.P., Weber, W.D. and Gupta, A.: SPLASH: Stanford Parallel Applications for Shared Memory, *Technical Report CSL-TR-92-505*, Stanford University (1991).
11) Tullsen, D.M. and Eggers, S.J.: Limitations of Cache Prefetching on a Bus-Based Multiprocessor, *Proc. of the 20th Annual International Symposium on Computer Architecture*, pp. 214-223 (May 1993).
12) Weber, W.-D. and Gupta, A.: Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture, *Proc. of the 16th Annual International Symposium on Computer Architecture*, pp. 273-280 (1989).
13) 坂根, 児玉, 佐藤, 山名, 坂井, 山口：並列計算機EM-Xのプロセッサ・ネットワークインタフェースの最適化の検討，情報処理学会研究報告，ARC-104-14, pp. 105-112 (1994).
14) 児玉, 坂井, 山口：データ駆動型シングルチッププロセッサEMC-Rの動作原理と実装，情報処理学会論文誌，Vol. 32, No. 7, pp. 849-858 (1991).
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，JSPP '94論文集, pp. 225-232 (1994).

(平成6年9月28日受付)
(平成7年1月12日採録)

**佐藤 三久**（正会員）

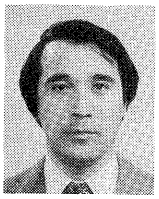
昭和34年生．昭和57年東京大学理学部情報科学科卒業．昭和61年同大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成3年より，通産省電子技術総合研究所勤務．現在，同所情報アーキテクチャ部計算機方式研究室主任研究官．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術等の研究に従事．日本応用数理学会会員．

**児玉 祐悦**（正会員）

昭和37年生．昭和61年東京大学工学部計数工学科卒業．昭和63年同大学院工学系研究科情報工学専門課程修士課程修了．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，データ駆動計算機などの並列計算機システムの研究に従事．特に，プロセッサアーキテクチャ，並列性制御，動的負荷分散などに興味あり．現在，情報アーキテクチャ部計算機方式研究室に所属．

**坂井 修一**（正会員）

昭和33年生．昭和56年東京大学理学部情報科学科卒業．昭和61年同大学院情報工学専門課程修了．工学博士．同年，電子技術総合研究所入所．平成3年4月より1年間米国MIT招聘研究員．平成5年3月よりRWC超並列アーキテクチャ研究室室長，現在に至る．計算機システム一般，特にアーキテクチャ，並列処理，スケジューリング問題などの研究に従事．情報処理学会研究賞（平成元年），同論文賞（平成2年度），元岡記念賞（平成3年），日本IBM科学賞（平成3年）各受賞．



**山口 喜教**（正会員）

1972年東京大学工学部電子工学科卒業．同年通商産業省工業技術院電子技術総合研究所入所．以来，高級言語計算機，データフロー計算機などの研究に従事．現在，情報アーキテクチャ部計算機方式研究室長．工学博士．1991年情報処理学会論文賞．著書「データ駆動型並列計算機」（共著）．電子情報通信学会会員．