細粒度通信機構を持つ並列計算機 EM-X による疎行列計算の性能評価

佐藤三久[†] 児玉祐悦[††] 坂根広史[††]
山名早人[††] 坂井修一[††] 山口喜教[††]

本稿では，疎行列問題の１つとして，疎行列の CG 法を取り上げ，並列計算機 EM-X での細粒度通信を用いた並列プログラミングと性能について報告する．EM-X では，データ駆動機構により細粒度パケットによる通信が効率的に処理されるため，従来のメッセージ通信型プロセッサに比べて，きわめて低レーテンシの通信が可能になっている．細粒度通信機構の効果を評価するため，CG 法の主要な演算である行列ベクトル積の計算について，complete exchange のブロック転送を使う方法と，細粒度通信を用いて必要な要素のみを転送する方法，行列計算中に要素ごとにリモートメモリ読み出しを行う方法について比較した．その結果，プロセッサ数が増えるに従って，EM-X では必要な要素にのみアクセスを行う細粒度通信による方法が有効であることが分かった．特に，プロセッサあたりの問題サイズが小さくなる場合において，有効である．ブロック転送による方法はいっせい転送によりネットワークがネックになり，効率が低下し，マルチスレッドでレーテンシを隠蔽したリモートメモリ読み出しによる方法はネットワークへの負荷は低いが，スレッド切替え等のオーバヘッドによって性能が低下している．

Parallelization and Performance Evaluation of Sparse Matrix Computation in The EM-X Multiprocessor

MITSUHISA SATO,[†] YUETSU KODAMA,[††] HIROFUMI SAKANE,[††]
HAYATO YAMANA,[††] SHUICHI SAKAI[††] and YOSHINORI YAMAGUCHI[††]

In this paper, we describe the parallelization of a sparse matrix computation, CG (Conjugate Gradient method) kernel taken from NAS parallel benchmark suite, for the EM-X multiprocessor. Dataflow mechanism of EM-X supports fine-grain communication very efficiently, which provides low latency communication, and flexible message-passing facility. We compare the performance of sparse matrix vector multiplications by the complete exchange communication, by element-wise remote update and by the element-wise remote read with multithreading. The measurements taken on the EM-X indicates effectiveness of the fine-grain communication which enables element-wise access efficiently. Fine-grain communcation is effective when problem size per PE becomes small in large scale multiprocessor systems. The complete complete exchange version incurs the negative impact due to the limitation of its bandwidth, and the performance of the element-wise remote read version is degraded by the overhead of context-switching for multithreading.

1. はじめに

本稿では，我々が開発中の並列計算機 EM-X[3] での疎行列問題の並列処理，特に CG 法による解法のプログラミングと性能について報告する．

EM-X は，データ駆動機構を持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出したり，受け取ったメッセージに対するスレッドを起動することをハードウェアの機構としてサポートしている．EM-X では，この機構がスレッド実行だけでなく，遠隔のメモリの読み出し・書き込みにも拡張されている．リモートメモリアクセスの通信は，プロセッサの命令実行とは独立にネットワークのインタフェース部のハードウェアで処理され，他のプロセッサのメモリに対して細粒度にアクセスすることが可能である．

† 新情報処理開発機構
Real World Computing Partnership
†† 電子技術総合研究所
Electrotechnical Laboratory

EM-X ではすべての通信は，2 ワードからなるパケットを単位として行われる．EM-X のネットワークは FIFO 性を保証しており，この性質を利用し，データの相手のメモリへの書き込みのパケット，同期のためのパケットにより，メッセージ通信操作を行う．これにより，低レーテンシでかつ，柔軟なメッセージ通信が可能となっている．

EM-X での疎行列問題の 1 つとして，本稿では，NAS Parallel benchmark[2] の CG kernel を取り上げる．CG 法（Conjugate Gradient Method：共役傾斜法）は，対称行列の反復解法の 1 つで，大規模な疎行列によく使われるアルゴリズムである．疎行列の解法には，ICCG 法など適当な前処理を行い効率的に計算を行う方法や解くべき疎行列の性質を利用した解法の工夫があるが，本稿で取り上げる問題はランダムな疎行列に対する基本的な CG 法の kernel 部分である．

疎行列の計算は，従来の分散メモリ並列計算機でも多く行われているが，send/receive のメッセージ通信を基本とするプログラミングモデルでは，プロセッサ内で，局所的に参照できる行列要素を使って，ローカルに計算し，その結果を必要なプロセッサにメッセージとしてブロック転送するプログラミングになる．これまでの並列計算機では，メッセージ転送のレーテンシが大きいので，全要素あるいは必要な要素をパックして 1 つのメッセージとして転送する．

EM-X では直接メモリアクセスによるブロック転送を用いることにより，同様なプログラミングが可能であるが，細粒度な通信が効率的にできるため，ブロック転送で一括して転送するのではなく，あらかじめ必要な要素が分かっていれば，その要素のみを相手プロセッサに書き込むことができる．また，細粒度通信を用いて，計算中に他のプロセッサの必要な値のみをリモートアクセスする方法も考えられる．リモートアクセスされる要素が少なくなる疎な行列の場合に有利になることが期待できる．要素へのリモートメモリ読み出しのレーテンシは，計算をマルチスレッド実行することによって隠蔽できる．

本稿では，ブロック転送と要素ごとのリモートメモリアクセスを使ったいくつかの方法について比較し，性能についての考察を行う．

2 章において，並列計算機 EM-X の概要について述べ，3 章で本稿の問題である疎行列の CG 解法の概要について述べる．4 章では，並列化と EM-X での並列プログラミングについて説明し，5 章において，実行結果を報告する．6 章で実行結果について考察を行い，性能の分析を行う．6 章において，結論を述べる．

2. 並列計算機 EM-X

EM-X は，EM-4[5] のアーキテクチャを基に，高機能・高性能化を目指したマルチプロセッサシステムである．基本構成としては，EM-4 と同様に，ローカルなメモリを持った要素プロセッサがサーキュラオメガネットワークで接続されている．

要素プロセッサ EMC-Y[3] は，RISC アーキテクチャとなっており，プロセッサの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と同期処理が融合されるように設計されている．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ともに起動される．スレッドが終了すると，次のパケットがハードウェアによりサポートされた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，同期機構として使うことができる．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とも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レジスタの値などスレッドの live な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ことができる．

他のプロセッサのメモリへのアクセスは，リモートメモリアクセスパケットのアドレス部にプロセッサ番号とアドレスを指定して，送出することによって行うことができる．リモートメモリアクセスパケットは，プロセッサの命令実行とは独立にネットワークのインタフェース部のハードウェアで処理される．EM-4 では，リモートメモリアクセスは命令を起動することによって行っていたが，その性能の違いについては児玉らの論文[3]で報告されている．

なお，EMC-Y には cache がなく，すべてのローカルメモリのアクセスは 1 マシンサイクルで行われる．開発したプロトタイプシステムは 80 PE で，設計クロックサイクルは 20 MHz である．

我々は，EM-X のプログラミングのために C 言語に並列プログラミングのための機能を拡張した EM-C[1] を開発した．EM-C は全プロセッサのメモリに対するグローバルアドレス空間に対するグローバルポインタ

```
vector r,p,q,z; /* working vector */

/* solve A*x */
real CGSOLVE(matrix A,vector x, real zeta){
  int it; real alpha, beta, rho, rho0;
  r = x; p = x; z = 0.0; rho = x * x;
  for(it=0; it < NITCG; it++){
      q = A*p;
      alpha = rho/(p * q);
      z += alpha * p;
      rho0 = rho;
      r += -alpha * q;
      rho = r*r;
      beta = rho / rho0;
      p += beta*r;
  }
  r = A*z - x;
  zeta = 1.0/x*z;
  x = (1.0/sqrt(z*z))*z;
  return sqrt(r*r);
}
```

図1 CG kernel のプログラム
Fig. 1 Outline of CG kernel.

を提供し，リモートメモリアクセスの記述を容易にしている．また，スレッドの生成，同期等の操作は組込み関数として提供することによって，in-line 展開し効率化を図っている．本稿で用いたプログラムは EM-C とアセンブラで記述されたライブラリで記述した．

3. CG 法のアルゴリズムと疎行列のデータ構造

図1 に，CG kernel のアルゴリズムを示す．CG 法の基本的な演算は，行列とベクトルの乗算，ベクトルの内積である．行列（matrix）とベクトルの結果はベクトル，ベクトルどうしの乗算は内積とし結果はスカラの値になるものとしている．

疎行列の基本的なデータ構造は，非ゼロ要素の値とその要素の行番号を格納する配列 `a` と `col_index`，これらの配列に対して，それぞれの列の要素の先頭へのポインタの配列 `row_start` から成る（行と列は逆でもよい）．**図2** に例を示す．

3.1 疎行列ベクトル積の計算

行列とベクトルの積の計算を並列化する場合，行列は演算中に更新されないため，静的にプロセッサに配置しておくことができるが，行列とベクトル積の演算結果は次回の繰返しに使われるため，次の繰返しまでに，演算結果を次の積の演算に使えるように配置しておかなくてはならない．

send/receive のメッセージ通信によるブロック転送を用いた並列化は，プロセッサへのデータの配置によって，以下の方法が考えられる．行列を `A` とし，被演算ベクトルを `p`，演算結果ベクトルを `q` とする．

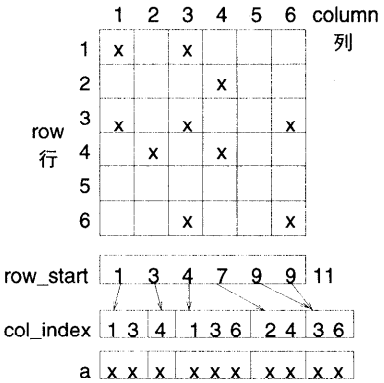


図2 疎行列のデータ構造
Fig. 2 Data structure of sparse matrix.

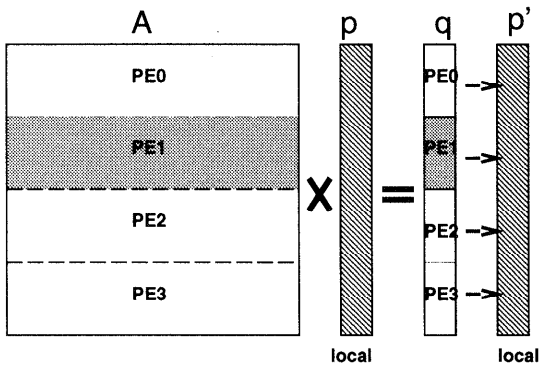


図3 行分割
Fig. 3 Row order mapping.

行分割（row order mapping） — 行列 `A` を行方向に分割する場合は，すべてのプロセッサにベクトル `p` を置き，`q` の担当する行に対応する部分を局所的に計算する．したがって，結果のベクトル `q` は，各プロセッサに分割された形で結果が求まる．だだし，`q` を次回の演算に使う場合は，すべての PE にベクトル `p` がなくてはならないため，各 PE の持っている `q` の部分ベクトルを他のプロセッサに転送しなくてはならない．その概要を**図3** に示す．部分ベクトルの転送では，all-to-all の個別転送が必要となる．

列分割（column order mapping） — 行列 `A` を列方向に分割する場合は，被演算ベクトル `p` は列に対応する部分ベクトルをおき，列に対応する部分的な積を計算する．全体の積を計算するために，各プロセッサにある部分積を集計して，演算結果のベクトル `q` を求める．集計には，行分割と同じく，プロセッサが持っている部分積を対応する `p` の部分ベクトルを持つプロセッサに転送し，集計する方法と，全体のベクトルを加算しながら，転送し，集計する方法がある．後者の場合は，部分積ベクトル全体を転送しなくてはならな

いが，集計する reduction 操作は $O(logN)$ ステップで済む．

以上の分割の他にこれらを組み合わせた2次元のブロック分割をする方法が考えられる．プロセッサ数が増えた場合には，単純に一方向に分割すると転送単位が細かくなり性能が低下してしまう．適当に両方向に組み合わせることによって，転送回数と転送サイズのトレードオフを計ることができる．本稿の EM-X での並列化では，行分割のみを考える．

4. EM-X での並列化

実際，相手プロセッサではすべての要素が参照されるわけではないので，あらかじめ必要となる要素をプロセッサごとに計算しておき，転送のフェーズでは必要な要素のみを書き込むことにより，転送するデータを減らすことができる．従来のメッセージ通信では，メッセージ通信のオーバヘッドが大きいため，これらのデータを個別に送るのではなく，パックして大きいメッセージにして送る方法が一般的であるが，EM-X では細粒度のリモートメモリ書き込みにより，パックせずに個別に転送ができる．

この方法の他に，EM-X では細粒度のリモートメモリアクセス機構を利用して，被演算ベクトルを必要に応じて読み出す方法が考えられる．この方法は，必要なデータのみを転送するため，疎な行列の場合は効率的であることが期待される．リモートメモリ読み出しのレーテンシは，計算のループを複数のスレッドで実行することによって，隠蔽することができる．EM-X に対して，以下の行列ベクトル積の計算の方法を用いた CG kernel の並列プログラムについて検討した．

ブロック転送による方法（blk） — EM-X では，all-to-all のブロック転送をメッセージの send/receive で行うよりも，メモリに直接転送したほうが効率的であるため，それぞれのプロセッサが，相手プロセッサの対応するメモリに書き込みを行い，最後にバリア同期を行い，書き込みを確認することによって，all-to-all の個別転送を行う．ローカルに計算された部分ベクトルを all-to-all のブロック転送を行う行分割行列ベクトル積を用いて，計算を行う．

細粒度更新による方法（upd） — 計算の当初に，アクセスされる要素を調べ，その情報をプロセッサ間で交換し更新する要素の配列を作る．ローカルに計算された部分ベクトルをその配列に従って，細粒度のリモートメモリ書き込みを使って更新する．

リモートメモリ読み出しによる方法（thd-n） — 行列を行分割で各プロセッサに配置し，被演算ベクトルに関しては，必要に応じて，リモートメモリ読み出しを行う．被演算ベクトルと演算結果ベクトルは同じブロック分割にする．なお，ブロック分割ではグローバルアドレスの計算に除算を必要とするため，大きなオーバヘッドをともなう．そのため，事前にプロセッサ番号とオフセットを格納した補助的な配列を用意して，そのオーバヘッドを低減した．さらに，リモートメモリ読み出しのレーテンシを隠蔽するため，n スレッドを行列演算時に動的に生成し計算を行う．

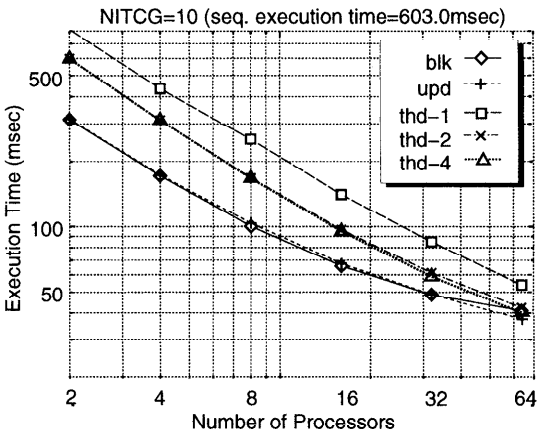


図4 CG kernel の実行時間
Fig. 4 Execution time of CG kernel.

もう1つの基本演算，ベクトルとベクトルの内積に関してはいずれの場合でも，各プロセッサは部分ベクトルを持っているため，まず，部分ベクトルどうしに関して，局所的に部分積を計算し，それを reduction 操作で，全体の内積を計算した．なお，EM-X での reduction 操作の実行時間は，64PE で 7 μsec 程度である．

5. 実行結果

CG kernel の blk，upd，thd-n 各プログラムについて，EM-X プロトタイプ☆で実行時間を測定した．

最適化として，共通部分式の最適化，ループ不変式，ループ帰納変数の最適化，関数内での大域的なレジスタ割当てなどを行った．

図4に，PE 数☆☆が，2，4，8，16，32，64 での実行結果を示す．実行に用いたデータは，NAS Parallel

☆ 96年4月現在，クロックは16 MHz で稼働しているが，本稿では設計クロックの20 MHz として実行時間を測定した．

☆☆ EM-X のサーキュラオメガネットワークは，80 PE システムでは，5×16段（1グループ 5PE が16グループ）で構成されている．台数は，グループ数が増える方向に増やした．たとえば，4PE では1グループ，16PE では，3グループを用いる．

Benchmarkのサンプルとして付属されている疎行列で，次元サイズ1400，非ゼロ要素数78148（スパース率4%，1行の平均要素数は55.8）のランダムな対称行列である．CG kernelの繰返し回数$NITCG$は10とした．逐次の並列化をしていないプログラムの実行時間は6073.0 msecである．なお，EMC-Yは，float型のみをサポートしている．

結果は，部分ベクトルの要素をローカルに計算し転送するblk，updが実行時間が短い．両者，ほぼ同程度の性能であるが，プロセッサ数が増えるに従って，1つのプロセッサから実際に参照される要素が少なくなるため，updのほうが効率が良くなる．実際，用いた疎行列はランダムな疎行列であるため，プロセッサ数が少ない場合にはほとんどの要素が参照され同等となる．転送速度は，転送先を指定する配列をアクセスしなくてはならないupdに比べて，単純なblkの転送の方が効率が良いが，ネットワークのバンド幅が制限されているため，あまり差がなくなっていると思われる．

thd-1は単一スレッドで実行した時間，thd-2，thd-4はそれぞれ，2，4のスレッドで実行した時間を示す．単一スレッドではレーテンシが隠蔽されていないため，大きく実行時間が増えるが，マルチスレッド実行ではそれらが隠蔽されて，実行時間が短縮されている．リモートメモリ読み出しのレーテンシはネットワークの影響がないとするとパケット送出，ネットワーク通過時間，相手メモリの読み出し処理を含めて，二十数クロック程度であるが，行列ベクトル積のループも同じ程度の命令数であるため，2つのスレッドで実行することでほとんど隠蔽できていると考えられる．そのため，thd-2とthd-4を比較して分かるようにスレッドを増やす効果は少ない．

PE数が少ない範囲では，明らかに，マルチスレッドでレーテンシを隠蔽したとしても，リモートメモリ読み出しを使うプログラムでは実行時間が増えている．その理由としては，PE数が少ない場合，多くの行を計算するために同じ被演算ベクトルの要素を重複してリモートメモリ読み出ししていることがあるからである．ブロック転送を使う方法はいわば，被演算ベクトルの要素を必要とするか否かにかかわらず1回だけ，効率的にアクセスする方法である．それに対して，要素ごとのリモートメモリ読み出しは，比較的オーバヘッドが大きく，PE数が少ない場合には，要素を何回も読み出したり，多くの要素の値を必要とすることになり，不利になる．また，リモートメモリ読み出しは，スレッドを中断しなくてはならないため，レジスタ上にある必要な値をメモリに退避するためのオーバヘッドもある．

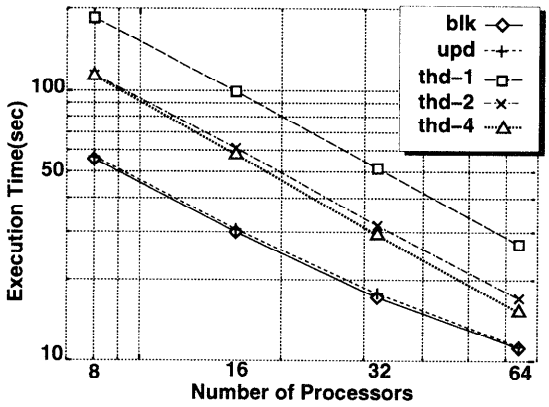


図5 CG kernel（class A）の実行時間
Fig. 5 Execution time of CG kernel (Class A).

PE数が増えるに従って，各プロセッサが計算する要素は少なくなり，必要な値のみを読み出すリモートメモリ読み出しによるthd-4は，すべての値を交換するblkによる方法よりも有利になる．結果として，64PEではthd-4は，blkと同程度の実行時間になった．

図5に，NAS parallel Benchmarkの標準の問題であるクラスAの実行時間について示す．問題のサイズは，14000で非ゼロ要素数は1853104である．このような大きい問題の場合は，64PEでもblkとupdが良い．用いている行列はランダムであり，プロセッサあたりの問題サイズが大きい場合には被演算ベクトルの要素のほとんどが参照されるため，細粒度通信で，選択的に参照するメリットは少ない．

6. 考　　察

6.1 疎行列のスパース率と並列化効率

リモートメモリアクセスを用いる行列ベクトル積の計算の性能は，行列が疎になると，実際にアクセスする要素が限られるため，全要素を効率的に交換するブロック転送よりも効率が良くなることが期待される．

そこで，CG-kernelの実行に用いたベクトルAとそれを間引きして要素数を1/4に減らした行列Bについて，行列ベクトル積を2回行う時間を計測してみた．行列A，Bの特徴を**表1**に示す．それぞれの行列での実行時間を**図6**，**図7**に示す．ブロック転送を用いたblkでは，プロセッサ数が増えるに従って，性能向上が見られなくなる．疎な行列の場合，64PEではリモートメモリ読み出しを行い，マルチスレッドを使ってレーテンシを隠蔽するプログラムの方が実行時間が短くなる．さらに，転送を必要な要素だけにした

表1 実行に用いた行列の特徴
Table 1 Characteristics of used matrices.

行列	次元サイズ	非ゼロ要素数	スパース率	1行あたりの平均要素数
A	1400	78148	4.0%	55.8
B	1400	19537	1.0%	14.0

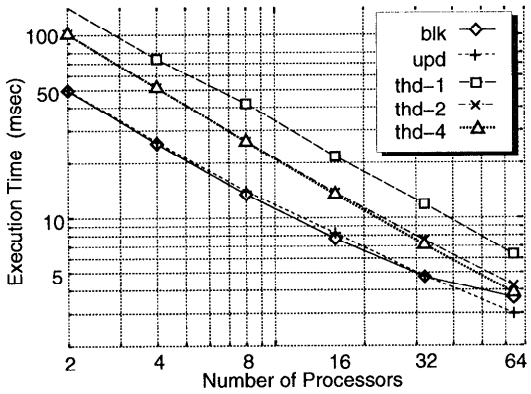


図6 疎行列Aの実行時間
Fig. 6 Execution time for sparse matrix A.

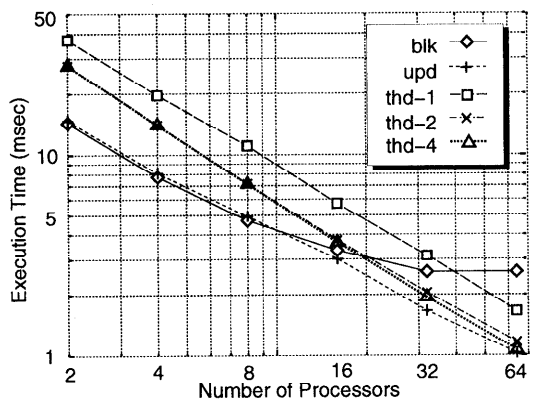


図7 疎行列Bの実行時間
Fig. 7 Execution time for sparse matrix B.

updが最も実行時間が短い.

それぞれの行列での逐次実行時間を基準にした効率を図8, 図9に示す. リモートメモリ読み出しの方法では, プロセッサ数が増えても, 効率がそれぞれ0.5から0.4程度とあまり低下しないのに比べて, ブロック転送を用いた方法では, 急激に性能が低下する. 当然のことながら, 行列のサイズが同じであるため, ブロック転送の時間は一定であり, 行列が疎になると計算に対して, 相対的に転送時間の割合が増えるので, 効率の低下は著しくなる. 必要な要素だけ更新するupdでも, 転送量が多い範囲では, blkと同じ状況になるため, 効率は低下するが, PE数が増え, 行列が

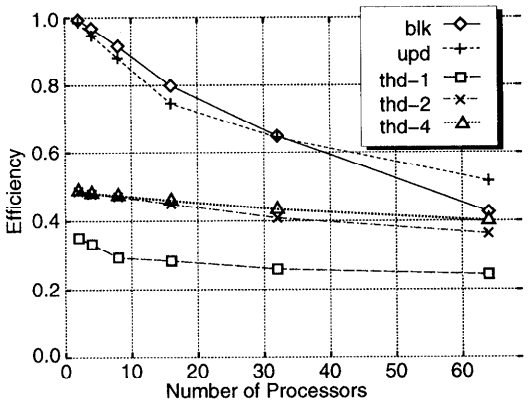


図8 疎行列Aの対逐次効率
Fig. 8 Efficiency for sparse matrix A.

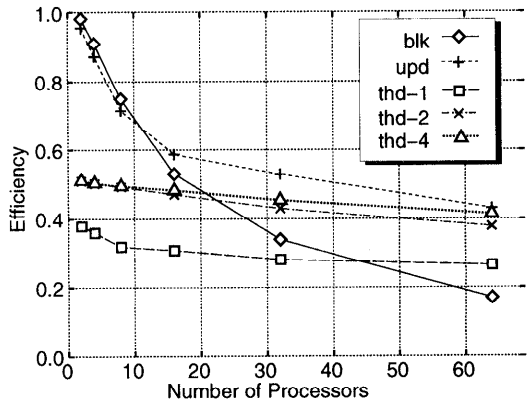


図9 疎行列Bの対逐次効率
Fig. 9 Efficiency for sparse matrix B.

疎になるにつれて, 転送量が少なくなるため, 効率が低下しなくなる.

演算中は被演算ベクトルの値は変わらないため, リモートメモリによる実行を効率的に行う方法として, 一度リモートメモリ読み出しを行った被演算ベクトルの値を局所的に記憶しておくという方法も考えられる. 実際に, この方法で実行してみたところ, 期待どおりプロセッサが少ない場合には, ある程度の効率化ができたが, プロセッサ数が多くなるにつれて, 値があるかのチェックのオーバヘッドのために, 単純にリモートメモリ読み出しを行うプログラムよりも性能が低下するのが観測された. この方法は, 共有メモリアーキテクチャのリモート参照をキャッシュしておく機構をソフトウエアで行ったものに相当するが, 今回の例ではオーバヘッドの方が大きかった. また, 共有メモリアーキテクチャとの関連でいえば, updが良いのはupdateのプロトコルが有効なことを示しているといえよう.

6.2 性能モデル

n を疎行列の次元サイズ，m を非ゼロ要素数とすると，疎行列ベクトル演算の逐次実行時間 T_s は以下の式で表される．

$$T_s = T_e \times m + T_l \times n$$

ここで，T_e は要素ごとの計算時間，T_l はループにともなうオーバヘッドである．並列処理において，プロセッサ数を p としよう．計算される要素はプロセッサに均等に割り当てられているとして，計算の負荷が均等に分散されるとすると，計算結果をブロック転送する方法の実行時間 T_{blk} は，

$$T_{blk}(p) = T_s/p + T_{all}(p, n/p)$$

$T_{all}(p,n)$ は，サイズ n のブロックを all-to-all 転送する時間である．細粒度に更新する方法での実行時間 $T_{upd}(p)$ は，

$$T_{upd}(p) = T_s/p + T_w(p,k)$$

T_w は，細粒度更新する時間であり，各プロセッサで他のプロセッサから参照される要素数 k に依存する．実際，この値は各プロセッサで異なるが均等に参照されると仮定して，ここでは単一の値 k を考える．この k は，転送ブロックサイズ n/p より小さいが，細粒度更新するためには更新する要素を選択する手間がかかるので，データ量あたりの転送時間は T_{all} の方が短い．しかし，次節に述べるように，$T_{all}(p,n)$ は，そのプロセッサ数 p に依存する．

さて，細粒度に要素をリモート読み出しをする場合，非ゼロ要素ごとにリモート読み出しするため，その実行時間 T_{thd} は，

$$T_{thd}(p) = ((T_e + T_r) \times m + T_l \times n)/p$$

T_r は要素のリモートメモリ読み出しに要する時間である．転送時間はマルチスレッドにより完全に隠蔽されるとすると，T_r は，転送のセットアップにかかるオーバヘッドである．このときの実行効率 E_{thd} は，

$$E_{thd}(p) = \frac{T_s}{T_{thd}(p) \times p} = \frac{T_s}{T_s + m \times T_r}$$

となる．この式は，$1/(1+m\times T_r/(T_e\times m+T_l\times n))$ となるが，ループのオーバヘッド T_l が計算時間 T_e に比べて十分に小さいとすると，$E_{thd} = 1/(1+T_r/T_e)$ になる．すなわち，転送時間と計算時間の比によって与えられる．マルチスレッドにより通信のレーテンシが完全に隠蔽されるとしても，リモートメモリ読み出しするためのオーバヘッドはループ変数のメモリへの退避・復帰，転送のためのセットアップなど，10 サイクルほどかかる．計算時間も 10 サイクル程度であり，E_{thd} が，ほぼ 0.5 であることが裏づけられる．この転送のオーバヘッドを減らすためには，レジスタの退避の回数を減らすことが有効であるが，そのためには，ループを unrolling して，複数の要素を同時に読み出すようにすることが考えられる．そこで，2 つの要素を同時に読み出し，EM-X のマッチング機能を使ってデータを待ち合わせるように工夫したところ，実行時間で 15%程度の効果が得られた．この場合，レジスタの退避にかかるコストは半分になり，ループあたりの命令数は 22 から 15 に減らすことができる．だだし，ハードウェアでサポートされるマッチング機能は 2 つのデータの待ち合わせだけであり，これ以上の複数の読み出しではデータの待ち合わせのためにコストが必要になり，効果が少ないことが分かっている．

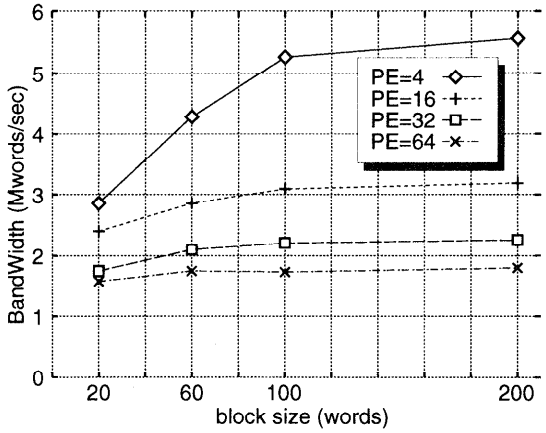


図 10 Complete exchange の性能
Fig. 10 Performance of complete exchange.

6.3 Complete Exchange の性能

complete exchange は各 PE にあるデータをすべての PE に送る all-to-all の通信操作である．ブロック転送を用いるプログラムの性能は，complete exchange の性能に大きく左右される．**図 10** に，complete exchange の性能を示す．これは，全 PE で block size 分のデータを他のプロセッサ全部に書き込んだときの単位時間あたりの転送量を示したものである．PE 台数が少ない場合には，6 M words/sec の転送速度☆ となっているが，PE 台数が多くなるにつれて，著しく転送性能が低下していることが分かる．

complete exchange の操作はすべてのプロセッサがいっせいにネットワークを使うため，ネットワークに対して非常に厳しい要求となる．32 PE 以上で，転送速度がおさえられているのは，ネットワークの容量に対して，非常に多くのパケットが出力されるため，頻

☆ 実際，point-to-point では，ハンドオプティマイズすることによって，ほぼ最大性能の最大 10 Mwords/sec で，転送を行うことができることが分かっている．

繁に衝突しているからである．この実験結果から，ブロック転送を用いる行列ベクトルの演算がプロセッサ数が多くなるにつれて，急激に性能低下をしている理由の1つがcomplete exchangeの性能低下であることが推測される．

今回用いたcomlete exchangeは，転送の相手先の順序は相対的な順番にすることにより，なるべく書き込みは衝突しないように工夫はしてあるものの，EM-Xのオメガネットワークのトポロジは考慮していない．通信をintensiveに使うcomplete exchangeの性能を改善するためには，トポロジに応じた衝突の少ない通信パターンあるいはアルゴリズムを検討する必要がある．

6.4 疎行列の性質と自動並列化の可能性

本稿で取り上げたNAS parallel benchmarkのデータは，ランダムな疎行列であり，行列の性質による最適化はできない．現実の問題に現れる疎行列の場合，非ゼロ要素が対角成分に付近に現れることが多く，このベンチマークのデータに比べて，細粒度通信で選択的にアクセスできることで性能向上することが期待できる．すなわち，EM-Xでは，自PEに対する通信はプロセッサ内のローカルなデータパスで行われ，通信はネットワークを経由しないように設計されている．ブロック分割をしている場合，細粒度通信を用いた方法では対角成分付近に非ゼロ要素を多く持つ行列に対しては多くの参照はローカルで行われ，他のプロセッサにある要素に対する参照は少なくなるため，ネットワークからの影響を受けにくくなり，性能が向上することが期待される．

ブロック転送による更新の方法は，ある程度のバンド行列が仮定できる場合には，その性質を利用して，通信するプロセッサを限るといった最適化が可能である[6]．しかしながら，この最適化を行うためにはプログラムを問題の性質を前提として構成する必要があり，自動並列化で行うのは困難である．

本稿で取り上げた細粒度通信を用いた方法を自動並列化という観点から見た場合，リモート読み出しを用いた方法は共有メモリに対する並列化と同様に自動並列化することが容易である．また，細粒度更新を行う方法（upd）は，Inspector/Executor[4]の技法の1つとして自動並列化を行うことができる．EM-Xでは細粒度通信のオーバヘッドが小さいため，従来の分散メモリ並列計算機で行われているような更新のためのデータ通信を一括して行うpack/unpackは必要ない．

7. 結　論

本稿では，疎行列問題の並列処理としてNAS parallel benchmarkのCG kernelを取り上げて，EM-Xでの並列プログラミングならびに性能について述べた．CG法の主要な演算である行列ベクトル積の計算について，complete exchangeのブロック転送を使う方法と，参照される要素のみを細粒度メモリアクセスにより更新する方法，計算中に要素ごとにリモートメモリ読み出しを行う方法で行い，比較した．その結果，

- EM-Xのデータ駆動機構により，低レーテンシで柔軟な細粒度通信が可能になっている．プロセッサ数が増えるにしたがって，必要なデータのみをアクセスする細粒度通信を活用した方法が効率が良い．特に，細粒度に更新する方法は，EM-Xでは細粒度の通信が低オーバヘッドで行うことができ，通信量を削減できるため効率が良い．
- 更新を行う方法では，転送がいっせいに行われるため，プロセッサ数が増えるに従って，効率が低下する．特に，ブロック転送による方法ではプロセッサが多い場合のネットワークに対する負荷が大きくなり，実行効率を低下させる大きな要因になる．
- 計算中に要素ごとにアクセスする方法では，マルチスレッドでレーテンシを隠蔽することができるが，スレッド切替えのオーバヘッド等が大きい．だだし，通信が計算と交互に行われるため，ネットワークに対する負荷が低く，ネットワークの影響を受けにくい．64PEではブロック転送による方法の性能にほぼ同程度となる．

細粒度通信を用いる方法はプロセッサあたりの問題サイズが小さくなる場合に有効である．細粒度通信により，必要な要素を低オーバヘッドで選択的にアクセスすることができ，また，通信と計算とオーバラップさせることにより，ネットワークに対する負荷を軽減できる．ある特定のサイズの問題を解く場合に大規模な超並列システムではプロセッサあたりの問題サイズが小さくなり，そのスケーラビリティが問題になるが，細粒度の低レーテンシの通信により，性能低下をおさえることができる．

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，太田前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．CC kernelについて，有用な議論の場を提供していただいた電総研，筑波大，お茶の水大のNPB研究グループ，特に板倉氏に感謝します．

参考文献

1) 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，情報処理学会論文誌，Vol.35, No.4, pp.551–560 (1994).
2) Bailey, D., et al.: The NAS Parallel Benchmarks, RNR Technical Report RNR-94-007, NASA Ames Research Center (1994).
3) Kodama, Y., Koumuara, Y., Sato, M., Sakane, H., Sakai, S. and Yamaguchi, Y.: The EM-X Parallel Computer: Architecture and Basic Performance, *Proc. 20th ISCA*, pp.14–23 (1995).
4) Koelbel, C. and Mehrotra, P.: Compiling Global Name-space Parallel Loops for Distributed Execution, *IEEE Trans. Parallel and Distributed Systems*, Vol.2, No.4, pp.440–451 (1991).
5) Sakai, S., Yamaguchi, Y., Hiraki, K., Kodama, Y. and Yuba, T.: An Architecture of a Dataflow Single Chip Processor, *Proc. 16th Annual International Symposium on Computer Architecture*, pp.46–53 (1989).
6) 加納，中田，奥村，大竹，中村，福田，小池：並列マシン Cenju 上の有限要素法による非線形変形解析，JSPP '92 論文集，pp.399–405 (1992).

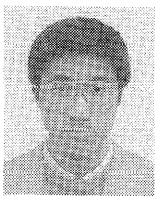
(平成 8 年 9 月 17 日受付)
(平成 9 年 7 月 1 日採録)

佐藤 三久（正会員）

1959 年生．1982 年東京大学理学部情報科学科卒業．1986 年同大学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．1991 年，通産省電子技術総合研究所入所．1996 年より，新情報処理開発機構つくば研究センターに出向．現在，同機構並列分散システムパフォーマンス研究室室長．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．情報処理学会，日本応用数理学会会員．

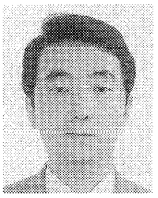
児玉 祐悦（正会員）

1962 年生．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．同年通産省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．データ駆動型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，並列探索問題などに興味あり．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）など受賞．電子情報通信学会，IEEE 各会員．



坂根 広史（正会員）

1966 年生．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，超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．電子情報通信学会，情報処理学会，神経回路学会会員．

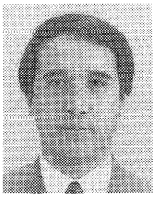


山名 早人（正会員）

1964 年生．1987 年早稲田大学理工学部電子通信学科卒．1989 年同大学大学院修士課程修了．1993 年同大学大学院博士後期課程修了．1989～93 年同大学情報科学研究教育センター助手．工学博士．1993 年電子技術総合研究所勤務．並列処理システム，並列化コンパイラ，WWW 情報検索の研究に従事．ACM，IEICE，IEEE 各会員．

坂井　修一（正会員）

1958年生．1981年東京大学理学部情報科学科卒業．1986年同大学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．1990年同研究所主任研究官．1991年4月より1年間米国MIT招聘研究員．1993年3月より1996年2月までRWC超並列アーキテクチャ研究室室長．1996年10月より筑波大学助教授（電子・情報工学系）．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．情報処理学会論文賞（1990年度），日本IBM科学賞（1991年），市村学術賞（1995年），ICCD Outstanding Paper Award（1995年）など受賞．IEEE，電子情報通信学会会員．



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，並列計算機アーキテクチャ，特にデータフロー計算機や実時間並列処理システムなどの研究に従事．現在，情報アーキテクチャ部・並列アーキテクチャラボ・リーダー，筑波大学連携大学院教授．工学博士．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）．電子情報通信学会，IEEE各会員．