計算機アーキテクチャ 119−41
（1996. 8. 29）

行列演算ベンチマークを用いた並列計算機 EM-X の評価

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

分散メモリ型並列計算機 EM-X の上で LINPACK ベンチマークを並列化して実装し、定型的な粗粒度演算および通信パターンが現れる行列問題における浮動小数点演算能力について評価した。並列化においてはピボット列のブロードキャストアルゴリズムと負荷分散の関係や、ブロードキャスト通信と列消去演算のオーバラップについて検討した。最内周ループの逐次実行部分は、過去に報告した高速コードを、多列同時消去によりレジスタの有効利用を図ってさらに高速化し、理論ピーク性能として1要素演算につき4命令で実行できるコードを作成した。そしてこれら有効な高速化手法と EM-X 固有の特性との関連を調べた。80PE 構成において 1000 次元の LINPACK ベンチマークに対して、354.2 Mflop/s、5000 次元で 601.5 Mflop/s の実測値を得た。

Performance Evaluation for a Matrix Operation Benchmark on EM-X Multiprocessor

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

In this paper, we discuss an implementation of the LINPACK benchmark parallelized on the EM-X multiprocessor and evaluate its performance focusing the floating point operations in which a regular repetitive pattern occurs. It is important to overlap the communication and calculation as much as relationship between the broadcast algorithms and load balancing. Exploiting the potential of a reduction of the number of memory accesses and adopting the multi-column simultaneous elimination technique, we also further accelerated the most inner-loop code we had already reported for optimization on a single processor. We demonstrate that the parallelized LINPACK benchmark on the 80PEs system can achieve 354.2 Mflop/s for a matrix of order 1000 and 601.5 Mflop/s for order 5000.

1. はじめに

EM-X は種々のアプリケーションに対して高速な実行が行えるよう、柔軟かつ強力な並列処理を目的として設計・製作された分散メモリ型並列計算機である。

従来、行列計算に代表される規則的な定型演算処理を効率よく行うために、種々のベクトル型スーパーコンピュータや MPP が開発されてきた。これらの計算機では、メモリ空間上で連続するデータあるいは規則的に配置されたデータに対して、特に演算や通信のスループットを向上させるために演算装置や通信機構に工夫が凝らされ、多大なハードウェア資源を投入している。

EM-X では演算や通信の機構は単純であるが、互いの能力を低下させることなく強く結合させ、しかもきめ細かく働かせることにより柔軟な並列処理が可能となり、規則的な演算パターンを持つ問題から不規則な演算パターンの問題まで広い範囲で効率的な演算処理能力を発揮することを目指している。

本研究では、EM-X が汎用計算機であるための必要条件の一つである規則的演算パターンの高速実行に焦点をあて、規則的で大量の演算と通信が発生するアプリケーションの処理能力指標を示すものとして、LINPACK ベンチマークをとりあげる。

本稿では、2章で EM-X のアーキテクチャとプログラミングについて述べ、3章で LINPACK ベンチマークの概要について述べる。4章では EM-X の特性と高速化技法について議論し、5章でそれらの技法に対する演算性能について評価する。最後に6章で総括と今後の検討課題を述べる。

2. EM-X

2.1 アーキテクチャ

EM-X は各要素プロセッサにローカルメモリを持つ

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

分散メモリ型並列計算機であり、データ駆動モデルの拡張と、通信と演算の融合によって、柔軟性のある並列処理を行うことを主眼として設計されている。現在80台のプロセッサからなるプロトタイプが稼働しており、その性能評価が進められている[1]。

EM-X の要素プロセッサ EMC-Y は CMOS ゲートアレイのチップ上にネットワークインターフェース、マッチング機構、演算実行部を実装しており、1チップと外付けメモリモジュールだけで要素プロセッサとしての機能を実現している。

演算部のパイプラインは2段、単一命令発行であり、供給クロック 20MHz 時に整数演算性能は 20MIPS、浮動小数点演算(単精度)の理論ピーク性能は加減乗算命令の場合 20Mflop/s、積和命令の場合 40Mflop/s である。ロード・ストアをはじめ、ほとんどの命令は1クロックで実行可能である。メインメモリは高速 SRAM を用いており、1プロセッサ当たり 1Mword(32bit+拡張6 bit) 実装されている。

各 EMC-Y はサーキュラオメガ網と呼ばれる直接多段結合網で接続されている。隣接プロセッサとのインターフェースとして2入力、2出力の通信ポートを持ち、そのピーク通信能力はポート当たり 10Mpacket/s である。1パケットは1ワードのデータを転送できるので、演算に利用できるデータの転送能力は 10Mward/s となる。送信処理については、パケット出力命令により1クロックで1パケットを送出できる。受信処理は適切なバッファリングの後、データ駆動モデルに基づくマッチング機構が受け持ち、1個あるいは2個のパケットの受信により、パケットで指定されたロケーションに置かれているスレッドと呼ばれる命令ブロックの実行が開始される。ブロードキャストやバリア同期に関する特別なハードウェアは持っていないので、それらはソフトウェアで実現する必要がある。

リモートメモリアクセスは、リモートプロセッサ上の数命令からなるスレッドを特殊パケットで起動してソフトウェア的におこなう方法と、スレッドを起動せず、ハードウェアで直接メモリアクセスする方法がある。前者は演算部の処理時間を消費するが多機能が実現でき、後者は1ワード単位の read あるいは write がおこなえるだけであるがリモートプロセッサ上の演算部をバイパスできるメリットがある。

2.2 並列プログラミング

EM-X には EM-C コンパイラ、標準ライブラリ、およびスレッドライブラリからなる EM-C 並列プログラミング環境が用意されている。EM-C の言語仕様は C 言語のスーパーセットであり、いくつかの並列構文やグローバル変数等を独自にサポートしている[2]。並列プログラミングは、言語によるサポート機能とスレッドライブラリを組み合わせることにより、共有メモリ的、メッセージパッシング的、それらの混在等多彩なスタイルを選択できる。EM-C は EM-X のもつ細粒度並列処理機能を低オーバヘッドで利用することができるため、高性能な並列プログラムを記述できる。

スレッドライブラリでは、スレッド操作、同期メモリ、バリア同期、ブロードキャスト、メッセージ転送、リモートメモリ処理等がサポートされている。

本研究では C 版のベンチマークプログラムを EM-C によるマルチスレッドプログラミングスタイルに一部書き換えて並列化し、コンパイルしたコードを EM-X 実機上で動作させた。

性能に大きく影響する部分は、EM-C コンパイラが出力したコードをさらにハンドオプティマイズした。

2.3 ブロードキャスト通信

次章で述べる LINPACK ベンチマークの分散メモリ型並列計算機における並列化アルゴリズムでは、1 to all のブロードキャストが必要とされる。EM-X ではブロードキャストのための特別なハードウェアはなく、ソフトウェアとパケットによる通信処理で実現する必要がある。ここでは二つのブロードキャスト手法をとりあげる。

ブロック転送ツリー： これは、直接リモートメモリ書き込みパケット (SYSWR パケット) を用いたブロック転送ツリーによるものである。EM-X でリモートメモリアクセスを行う場合、パケット送出はソフトウェアで行う必要があるが、リモートプロセッサ上の read および write 処理はハードウェアで行うことができる。EM-C から利用できるライブラリとして、10 Mword/s (1word/2clock) のスループットを持つブロック転送関数が用意されている。これによってデータを持つプロセッサから隣接する2つのプロセッサに転送し、以後ツリー状に転送すればよい。データを受信する側のプロセッサ時間を消費しないので、原理的にはオーバヘッドは最小限に抑えられ、スループットも高い。この方式を適切なスケジューリングなしで用いる場合、IDLE 状態のプロセッサが多くなり、完了するまでの全体スループットおよびレイテンシが次に述べる方法に比べると劣るため、スケジューリングの最適化が重要である。

細粒度スレッド (DISTI) によるパイプライン転送： これは、プロセッサ間でデータやプログラムのパイプライン転送を行うための細粒度スレッドを、ブロードキャストするデータのワード毎に起動するものである。EMC-Y では DISTI パケットと呼ばれるパケットを受け取ると、システムが提供する DISTI 処理スレッドが起動される。SYSWR と異なりリモートプロセッサ上の受信処理は他のスレッド実行とオーバラップできない。DISTI スレッドでは、パケットのデータ部にあるデータを次のプロセッサへ再び DISTI パケットで送出するとともに、アドレス部で示されるローカルアドレスに書き込む。これを全てのプロセッサについて繰り返せばブロードキャストが完了する。この DISTI スレッドは1パケットにつき4命令で実行できるので、スループットは 5Mword/s であるが、全プロセッサを通じてパイプライン的動作をするので、全体のスループットは高く、レイテンシも小さく抑えられる。

3. LINPACK ベンチマーク

LINPACK は Dongarra らによって開発された汎用の線形問題求解ソフトウェアパッケージである。その一部がベンチマーク化[3] されており、浮動小数点演算能力を示す指標として広く用いられている。このベンチマークでは、密行列で表された連立一次方程式を部分ピボット選択を用いたガウス法で解く過程の計算時間を測定することにより、単位時間当たりの処理能力を求める。計算過程で用いられる浮動小数点演算回数は問題サイズによって決まるので、計算時間がわかれば Mflop/s 値が得られる。プログラムは大きく分けて LU 分解部と後退代入部で構成される。実際には後退代入が n^2 オーダであるのに対し、LU 分解が n^3 オーダであるので LU 分解部の計算時間が支配的である。

3.1 LU 分解

LINPACK で用いられている LU 分解のアルゴリズムでは最内周ループが列方向に沿っており (KJI 型)、係数行列の列データをメモリ上で連続配置する。そのアルゴリズムはピボット選択とピボット列の処理を含めて図1に示す3重ループで表される。係数行列上での処理を図2に示す。LINPACK ベンチマークを並列化する場合にはアルゴリズムの変更が許されているが、ここでは列指向のアルゴリズムのままで議論を進める。

3.2 並列化

上で述べたアルゴリズムでは明らかに列単位の局所性が高いので、並列化に際して係数行列データのプロセッサへの分散を列サイクリックに行う。中間の j ループは完全な並列性があるのでこのループを各プロセッサに分散し、SPMD 並列処理を行う。最外周ループの各 k について、ピボット処理はその対象列を持つプロセッサでのみ行われる。各列データの消去過程でピボット列データ a[i,k] が必要となるが、ピボット列を持つプロセッサでピボット処理計算をした後に一度各プロセッサへブロードキャストしてしまえば、同一プロセッサ内で持っている各列データの計算については再利用できる。従って最外周ループ毎に、1 to all ブロードキャストと各プロセッサローカルな列消去処理が繰り返されることになる。

```
for (k = 0; k < n-1; k++) {
    for (i = k+1; i < n; i++)
        pivoting for a[i,k];
    for (j = k+1; j < n; j++) {
        /* row elimination */
        for (i = k+1; i < n; i++) {
            a[i,j] -= a[k,j] * a[i,k];
        }
    }
}
```

図1 LU 分解のアルゴリズム。a[i,j] は添字 i に沿うデータがメモリ上で連続することを表す。

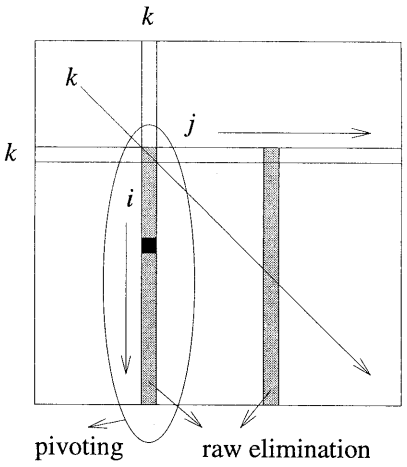

図2 係数行列上の処理

4. EM-X における高速化技法

分散メモリ型並列計算機上での LINPACK ベンチマークの実装と実行については、その計算機のアーキテクチャに適した種々のアルゴリズムが研究されている[5][6][7]。

それらで取り上げられている技法を含め、EM-X 上での高速化について検討する。

4.1 ブロードキャスト通信と演算のオーバラップ化

分散メモリ型並列計算機では、プロセッサ間通信のレイテンシが問題となりそれを隠蔽することが性能低下を防ぐために重要となる。EM-X は細粒度パケットアーキテクチャとそれを活かす通信機構を持っており、細粒度通信は非常に低オーバヘッド・低レイテンシで実現できる。しかしブロードキャストの場合は全てのプロセッサへデータが到達するのにかかる時間が無視できない場合がある。さらに、データ量が多い場合はプロセッサ時間を多く消費するため、ブロードキャストの方法によっては通信と演算のスケジューリングを適切に行わないと負荷の偏りが生じてしまう。

LINPACK におけるブロードキャスト通信は、EM-X にとってプロセッサ間通信のレイテンシ問題とともにプロセッサ処理時間を消費することによる負荷の偏りの問題を引き起こす。LU 分解の演算とブロードキャスト通信を単純に並列化すると、通信と演算の開始時間と終了時間がプロセッサ間で異なるので負荷の均等化を阻害し、全体の処理効率を低下させる。通信処理のためのプロセッサ処理時間は本質的になくすことはできないが、レイテンシや負荷の偏りは演算とブロードキャスト通信のオーバラップによって軽減できる。

図2に示した最外周ループの k 番目のイテレーションにおける k 列目のピボット処理は、前のイテレーショ

ンでその列の消去演算が完了していなければ開始できない。逆に、k+1 列目の消去演算が完了すれば他の列の消去演算に先行して k+1 列目のピボット処理を行うことができる。そして、次の k でのピボット列の計算完了とともにブロードキャストを起動し、さらにそれぞれのプロセッサでピボット列の受信が完了すると同時に列消去を開始するようにして最外周ループをパイプライン処理すれば、レイテンシの隠蔽とともに負荷の偏りが改善される。この場合、各プロセッサで参照中のピボット列に上書きしないよう、オーバラップ分のピボット列領域を別に持つ必要がある。

列指向の LINPACK ベンチマークではピボット列の計算で逐次処理を行う部分があり、並列化効率を阻害する要因となる。しかしながら、オーバラッピングによって最外周ループを複数段多重実行させることにより、その逐次処理部分を隠蔽することができる。

4.2 ブロードキャストアルゴリズムの選択

ブロードキャスト通信はプロセッサ時間を消費するため明らかにオーバヘッドとなり、その性能が直接全体の演算性能に影響する。DISTI のパイプライン転送は高速であり、細粒度から粗粒度まで高い性能を持つ。しかしながら、1 対 1 のプロセッサ間通信だけを見るとブロック転送ライブラリを使ったツリー方式の方がスループットが高い。EM-X では低オーバヘッドのマルチスレッド実行を行うことができるため、ブロードキャストで生じ得る IDLE 時間を埋めるだけの十分な演算スレッドを供給してプロセッサ間でオーバラッピングを行えば、高いスループットを生かせる場合がある。

4.3 多段・多列同時消去

最外周ループをアンローリングすることによる多段同時消去、中間ループをアンローリングすることによる多列同時消去はメモリアクセスを減らすレジスタブロッキングとして有効である。EMC-Y ではロード命令・ストア命令は 1 クロックで実行できるため、メモリアクセスレイテンシの削減や命令スケジューリングの自由度向上の効果は他のパイプライン段数の深い RISC アーキテクチャに比べて小さい。しかしコンパクトな最内周ループに対しては、1 要素あたりのロードストア回数を減らすことの効果は相対的に大きくなる。

我々はすでに文献 4) で最内周ループの高速化について報告した。その本質的な演算部分は、2 個のロード、乗算、加算、結果のストアの 5 命令から構成される。つまり次元数 n が無限大で、ループアンローリングを無限に行うと仮定すると、1 要素あたり 5 clock で演算実行できる。

本稿では多列同時消去をとりあげ、最内周のループをさらに高速化する。図 1の中間ループ (j ループ) に 4 重アンローリングを施し、一つの最内周ループ (i ループ) にまとめて 4 列を同時に消去すると、図 3のようになる。ここで、i ループの a[i,k] は各列共通に参照しているので、データのロードを 1/4 に減らすことができる。この j ループのアンローリングは中間結果保持のためループ展開数に比例してレジスタが必要となるので 4 重にとどめている。ループカーネルは、j ループを m 重、i ループをさらに l 重にアンローリングする場合、$(m*4+2)*l+1$ 個の命令 (= クロック数) で構成できる。評価に用いたプログラムでは、$m=4, l=8$ であり、配列サイズ n が無限大であればその実行クロック数は 32 要素あたり 145 クロック、1 要素あたり 4.53125 クロックである。

```
for (j = k+1; j < n; j+=4) {
    for (i = k+1; i < n; i++) {
        a[i,j] -= a[k,j] * a[i,k];
        a[i,j+1] -= a[k,j+1] * a[i,k];
        a[i,j+2] -= a[k,j+2] * a[i,k];
        a[i,j+3] -= a[k,j+3] * a[i,k];
    }
}
```

図 3 多列同時消去 (実際には余りループの処理が必要)

列間が隣接している必要はないので、列サイクリック分割の場合でも同一プロセッサ内で同様のループ展開が行える。

4.4 ブロック化

データのブロック化の効用として、一般にはキャッシュ利用率向上と行列積演算による演算性能向上が期待される。EM-X ではキャッシュを用いない単純なメモリ階層を採用しているため、後者の可能性を検討する。行列ブロックサイクリック分割を行うとそのブロック内演算で行列積演算が現れ[5]、EMC-Y の浮動小数点積和演算が利用できる。

5. 性能評価

これまで述べてきたような手法を組み合わせて LINPACK ベンチマークプログラムを作成し、EM-X 上で実行させた☆。

以下では、多列同時消去、ブロードキャストおよび負荷分散の効果について実行結果をもとに考察を行う。

5.1 多列同時消去による逐次性能の向上

これまで多列同時消去を行わない最内周ループの逐次ピーク性能は 1 要素分の演算 (2 flop) あたり 5 clock つまり 8 Mflop/s であった。実際には配列サイズやアンローリング多重度を無限大にできないので、ループ処理のオーバヘッドがかかってくる。そのほかに LINPACK ベンチマークとしては後退代入の処理時間も加わる。表 4に $m=1, l=8$ の場合として測定値 (文献 4) に比べてコンパイラ性能とライブラリ性能が向上している) および n $\rightarrow \infty$ 時のピーク値を示す。配列サイズが大きくなるとオーバヘッドが小さくなるので性能が向上する (n 依存性)。なお単一プロセッサ上ではプログラムとシ

☆ EM-X は現在プロセッサボード上のメモリアクセスタイミングを調整中であり、実際には 16MHz のクロックで動作させて計測したが、本稿では設計クロック 20MHz で動作しているものとして換算した値を用いた。

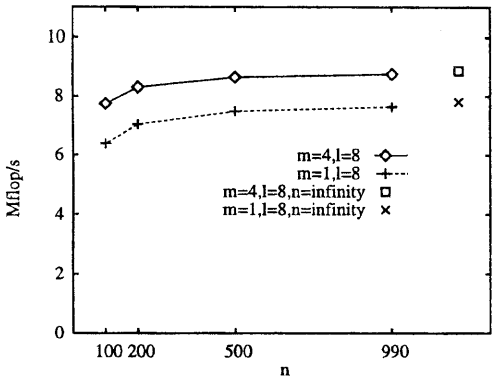

図4　多列同時消去による逐次コードの高速化

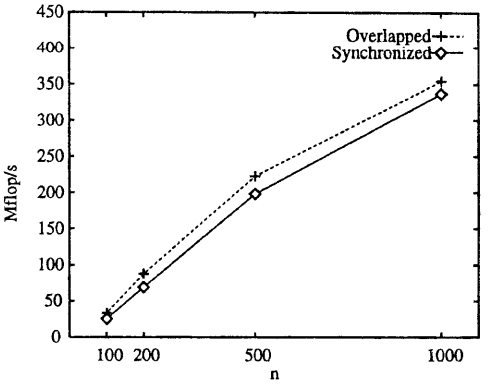

図5　オーバラッピングの効果

ステム領域のため n = 1000 のデータ領域を確保できず、最大 n = 990 としている。

多列同時消去は、プロセッサ内の逐次的列消去演算処理を高速化する。配列サイズとアンローリング多重度を無限大とすれば理想的に 4 clock で 1 要素を計算できるので、ループ部分の理論ピーク性能は 10 Mflop/s となる。実際には 4.3 で述べた $m = 4, l = 8$ に展開したコードを用いて図4の結果を得た。

その場合のピーク性能は約 8.828 Mflop/s、n = 100 の場合の実測値として 7.741 Mflop/s を得た。これは多列同時消去を行わない場合に対してそれぞれ 21.4% および 13.1% の向上率である。

5.2　オーバラッピング

図5に、80台のプロセッサ上で、ピボット列のブロードキャスト通信と列消去演算をオーバラップさせずに実行した結果 (Synchronized) と、オーバラップさせて実行した結果 (Overlapped) を示す。ブロードキャストには DISTI パイプラインを用い、内部二重ループは $m = 4, l = 8$ のコードを用いている (以下同様)。

グラフより、n 依存性が顕著であることがわかる。つまり n が大きくなると演算量が増えるため負荷が分散され性能が向上する。

オーバラッピングすることにより、n = 100 で 32% の性能向上があるが、n = 1000 の場合では 5.1 % の向上にとどまっている。これは n が大きくなると演算量が n^3 オーダで増加するのに対し、通信量は n^2 オーダであるため相対的に通信時間が短くなってオーバラッピングの効果が小さくなるためである。

5.3　ブロードキャスト

図6に、DISTI パイプラインによるブロードキャストを用いた実行結果と、ブロック転送ツリーによる実行結果を比較して示す。両方とも列消去演算とブロードキャスト通信をオーバラップさせ、ピボット列の演算と通信だけで同期をとっている。プロセッサ台数を変えた場合の実行結果も併せて示した。

図では差がわかりにくいが、ほとんどの場合 DISTI

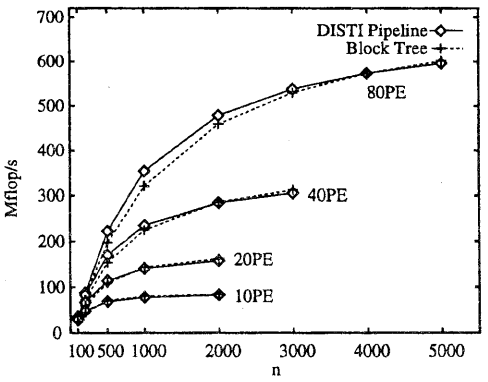

図6　ブロードキャスト方法の比較

パイプラインを用いた方が性能が良い。ブロック転送ツリーでは n が大きい場合、またプロセッサ台数が少ない場合に DISTI より性能が良くなる現象が見られるが、これは一台あたりの演算量と直接関係がある。つまり演算量が多ければ通信にオーバラップして供給される演算が多くなり、ブロック転送ツリー方式での負荷の偏りが補償され、高いプロセッサ間通信のスループットが活かされるからである。80PE、n = 5000 の場合において、DISTI では 595.8 Mflop/s であるのに対して、ブロック転送ツリーの方が性能が良くなり 601.5 Mflops/s を得た。

LINPACK の LU 分解においては、n が大きくなると演算量に対する通信量が相対的に少なくなるため、ブロック転送ツリーの特性が活かされる領域では結局利が少なくなってしまう。しかしながら、より通信ブロックサイズが大きいアプリケーションにおいては有効な方法であると考えられる。

5.4　台数効果

台数効果のグラフを図7に示す。全体の問題サイズが同じままでプロセッサ台数が増えると1プロセッサ当た

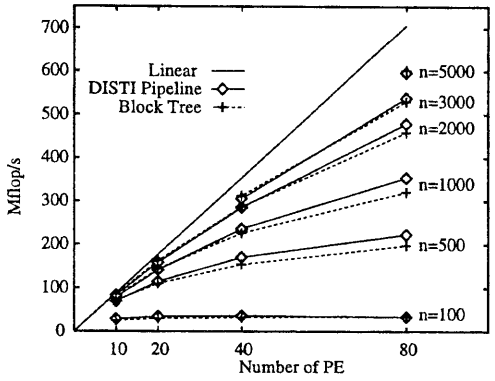

図7 台数効果

りの演算量が少なくなり、ブロードキャスト時間が長くなるため、負荷の偏りが大きくなる。これは台数効果の鈍化を引き起こす。問題サイズを大きくすることにより負荷の偏りは小さくできる。

問題サイズが小さい範囲では、台数効果が現れにくい。特に、n = 100 では性能低下が見られる。これは、負荷の偏り、演算に対する通信の割合、さらに後退代入部の逐次処理部分の影響が全て大きくなってきたことによるものと考えられる。

問題サイズ無限大の逐次性能に対する性能向上効率は、80PE, n = 5000, ブロック転送ツリー型ブロードキャストの場合に約 85% となった。

6. おわりに

EM-X は細粒度から粗粒度に渡って柔軟な並列処理を目指したアーキテクチャをもつ。本稿では比較的粗粒度な問題として LINPACK ベンチマークをとりあげて EM-X の浮動小数点演算能力を評価するとともに、その高速化技法について述べた。LINPACK ベンチマークプログラムの最外周ループ毎に、列消去演算とピボット列ブロードキャスト通信を全プロセッサで同期をとりながら単純に繰り返すと、通信レイテンシや負荷の偏りによる効率低下を招くが、同期を最小限にして演算と通信をオーバラップさせることにより負荷を均等化し並列効率を向上させることができた。

ブロードキャスト通信については DISTI パイプライン転送とブロック転送ツリーを取り上げ比較した。その結果、DISTI パイプラインは 1 プロセッサあたりの演算量が少ない領域で有効に働き、ブロック転送ツリーは配列サイズを大きくして先行演算を十分供給できる場合にその高スループットが活かせることがわかった。EM-X ではこのように、演算量と通信量の割合によって通信方法とスケジューリングを選択することにより、高い性能を維持する柔軟な対応が可能である。

浮動小数点演算能力については、内部二重ループに対してアンローリングを行うことにより多列同時消去を施しその逐次効率を高めることができた。高速化技法としては、さらに最外周ループのアンローリングによる多段同時消去があるが、多列同時消去を適用した時点でレジスタ数が大量に必要となったためと、多段同時消去は複雑なピボット列先行演算が必要である上、今以上の効率向上は期待しにくいため、今回は見送った。しかしながら多段同時消去単独使用の可能性はまだ残されており、さらに注意深く検討してレジスタ資源問題が解決できれば多段多列同時消去によってさらに性能を向上できると考えられる。また、行列ブロックサイクリック分割による積和演算命令の活用の可能性も残されている。多段同時消去と併せて、今後の検討課題である。

さらに、細粒度並列処理に適した EM-X の特徴を活かし、配列の要素毎に演算と通信をパイプライン化することにより負荷分散の促進が期待できる。これにより負荷の不均衡が問題となっている n の小さい範囲での速度向上を図ることを検討中である。

謝辞

本研究を遂行するにあたり御指導、御討論いただいた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) 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.
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, pp.183-190, 1993.
3) Dongarra,J.J.: Performance of Various Computers Using Standard Linear Equations Software, Computer Science Department, University of Tennessee, CS-89-85, July 3, 1996.
4) 坂根, 児玉, 佐藤, 山名, 坂井, 山口: 並列計算機用要素プロセッサ EMC-Y の基本性能評価, 情報処理学会研究報告, 94-ARC-107, pp.65-72, 1994.
5) 上村, 清水, 石畑, 堀江: LINPACK ベンチマークの並列ベクトル処理 – 並列計算機 AP1000 用数値演算アクセラレータによる実現, JSPP'94, pp.185-192, 1994
6) 建部: 分散メモリ型並列計算機による LU 分解, 情報処理学会研究報告, 95-HPC-57, pp.55-60, 1995.
7) 曽根, 服部, 朴, 中村, 中澤: CP-PACS パイロットモデルにおける LINPACK ベンチマークの高速化, 情報処理学会研究報告, 96-ARC-117, pp.83-88, 1996.