Vol. 39 No. 7 情報処理学会論文誌 July 1998

テクニカルノート

分散メモリ型並列計算機による ブロック化 Householder 法の性能評価

片桐孝洋† 金田康正††

この論文では，一般行列を Hessenberg 形に変換するためのブロック化 Householder 法の並列アルゴリズムを論じている．さらにこのアルゴリズムを分散メモリ型並列計算機 AP1000+に実装して，性能を評価した．その結果，データ分割方式としてサイクリック分割方式がブロックサイクリック分割方式に対して有効となることが分かった．

Performance Evaluation of Blocked Householder Algorithm on Distributed Memory Parallel Machine

TAKAHIRO KATAGIRI† and YASUMASA KANADA††

In this paper, we discuss parallel implementations for the reduction of general matrix to Hessenberg form. For implementing our algorithm, the blocked Householder algorithm is used. Its performance was evaluated on the distributed memory message-passing multiprocessor AP1000+. According to the results from our experiments, we found that the cyclic distribution is better than the block-cyclic distribution.

1. はじめに

Householder 法は数ある行列計算手法のうちの最も重要な手法の1つである．たとえば，行列の固有値や固有ベクトルを計算する際に，まずその行列の固有値を変えない変換（相似変換）を Householder 法を用いて行い，それから次に固有値や固有ベクトル計算を行うのが一般的である．

その一方で，Hoseholder 法において主メモリからのデータのロード回数を減らし，できるだけキャッシュ，レジスタなどの高速アクセス可能な記憶を利用するアルゴリズムが知られている．このアルゴリズムは Bischof と Loan ら[1]によって開発されたもので，ブロック化 Householder 法と呼ばれている方法である．

ブロック化 Householder 法を用いた相似変換の並列化に関する初期の重要な並列アルゴリズムは，Dongarra と Geijn の文献 2) に示されている．また同種の研究開発報告として，米国 Oak Ridge 国立研究所の Choi らによる技術報告書[3]に，並列ブロック化アルゴリズムの性能が報告されている．しかしこれらの文献では，本来切り離して考慮すべきと考えられる，逐次処理の性能向上要因であるブロック幅の依存性の議論と，並列処理の性能向上要因であるデータ分割方式との議論がまったくなされていない．

そこで本論文では，このブロック幅の依存性とデータ分割方式とを分離して性能評価を行う．並列ブロック化アルゴリズムの性能評価のために，富士通の高並列計算機 AP1000+に並列ブロック化アルゴリズムを実装した．

2. 並列実行環境と表記法

本論文において，並列計算機は p 個の均質的な PE（Proccssing Element）で構成され，それらは一次元的に $P_0, P_1, \cdots, P_{p-1}$ とラベル付けされているものとする．さらに各 PE はメッセージの放送（同一メッセージを1対多の通信で放送すること）や，各 PE が局所的に所有するデータの要素に対して総和演算を行うような計算（1対1もしくは全対全のなどの通信により実現）が行えるように，ネットワーク網により結

† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Graduate School of Science, University of Tokyo
†† 東京大学大型計算機センター
Computer Centre, University of Tokyo

表1 数学表記法とそれらの説明
Table 1 Mathmatical notations and its illustration.

表記	説明
α, μ	スカラ $\in \Re$
x, y, u	ベクトル $\in \Re^n$
d, e, f	ベクトル $\in \Re^m$
X, Z, U	行列 $\in \Re^{n \times m}$
A	行列 $\in \Re^{n \times n}$
$[A]_{i:j,k:l}$	A の行 $i, \cdots, j$, 列 $k, \cdots, l$ 部分行列
$[A]_{*,k:l}$	A の行すべて, 列 $k, \cdots, l$ 部分行列
$A^{(k)}$	反復 k における行列 A

合されているものとする.

定式化のために, **表1** に示す表記を用いることとする.

3. 並列アルゴリズムの説明

3.1 列ブロックサイクリック分割方式の定義

ここで, 入力行列 A をどのように各PEに分散させるかを定義しておく. いま入力行列の大きさを n, 使用するPEの番号を $0, 1, \cdots, p-1$, 入力行列の列のインデックスを i $(0 \leq i < n)$, ブロック幅を m とする. いま, 列 i を所有しているPE番号を $OwnerPEnum$, 各PE内の局所インデックスを $LocalI$, i 列が示すブロック内番号を $LocalBlkNum$ とする. このとき, 列ブロックサイクリック分割方式とは i を次に示す組, すなわち $\langle OwnerPEnum, LocalI, LocalBlkNum \rangle$ に写像する分割方式である.

$$i \mapsto \langle r \bmod p,\ \lfloor r/p \rfloor \times m + s,\ s \rangle \qquad (1)$$

ただし, $r = \lfloor i/m \rfloor$, $s = i \bmod m$ とおいた.

さらに, ブロック幅 $m = 1$ の特殊な場合を列サイクリック分割方式といい, 次の写像関数で表される.

$$i \mapsto \langle i \bmod p,\ \lfloor i/p \rfloor,\ 0 \rangle \qquad (2)$$

3.2 列ブロックサイクリック分割方式にともなう並列アルゴリズム

列ブロックサイクリック分割方式に基づき, 各PEが所有している行列データ A に対してのみ演算をすることを考える. このときこのデータ分割方式に基づく単純な計算方式では, 行列 U, X およびベクトル d, e, f は全PEが分割せずにすべて所有し, 行列 Z は列ブロックサイクリックに, ベクトル y はブロックサイクリックに分割するのが妥当である. よって, この並列アルゴリズムは**図1** のようになる.

ここで図1中の枠で示すベクトルリダクション演算とは, たとえば, ⟨12⟩ は各PEが所有している異なるベクトルに対して, その要素の総和を求めて, 全PEがその値を所有する処理を意味している. この処理は通信と計算を必要とする. また図中の表記 $H^{(k)}(x) = (u, \alpha)$

```
⟨1⟩  for (iter=1; iter ≤ n-2; iter+=m) {
⟨2⟩    [wA]_{iter:n,1:m} = [A^{(iter)}]_{iter:n,iter:iter+m-1};
⟨3⟩    for (blk=1; blk ≤ m; blk++) {
⟨4⟩      if ([A^{(iter+blk-1)}]_{*,iter+blk-1} を所有)
⟨5⟩        [wA]_{iter+blk-1:n,blk}の放送;
⟨6⟩      else
⟨7⟩        [wA]_{iter+blk-1:n,blk}の受信;
⟨8⟩      H^{(iter+blk-1)}([wA]_{*,blk}) =
&            ([U]_{*,blk}, α_{iter+blk-1});
⟨9⟩      if (blk == 1) {
⟨10⟩       /* 未ブロック化アルゴリズム */
⟨11⟩       [X]_{*,1} = α_{iter+blk-1} A^{(iter)} [U]_{*,1};
⟨12⟩       ベクトルリダクション [X]_{*,1};
⟨13⟩       y = α_{iter+blk-1} A^{(iter)T} [U]_{*,1};
⟨14⟩       μ_{iter+blk-1} = α_{iter+blk-1} [U]^T_{*,1} [X]_{*,1};
⟨15⟩       [Z]_{*,1} = y - μ_{iter+blk-1} [U]_{*,1};
⟨16⟩     } else {
⟨17⟩       /* ブロック化アルゴリズム */
⟨18⟩       d = α_{iter+blk-1} A^{(iter)} [X]^T_{*,1:blk} [U]_{*,blk};
⟨19⟩       e = α_{iter+blk-1} A^{(iter)} [U]^T_{*,1:blk} [U]_{*,blk};
⟨20⟩       f = α_{iter+blk-1} A^{(iter)} [Z]^T_{*,1:blk} [U]_{*,blk};
⟨21⟩       ベクトルリダクション f;
⟨22⟩       [X]_{*,blk} = α_{iter+blk-1} A^{(iter)} [U]_{*,blk}
&              - [U]_{*,1:blk} f - [X]_{*,1:blk} e;
⟨23⟩       ベクトルリダクション [X]_{*,blk};
⟨24⟩       y = α_{iter+blk-1} A^{(iter)T} [U]_{*,blk}
&              - [Z]_{*,1:blk} e - [U]_{*,1:blk} d;
⟨25⟩       μ_{iter+blk-1} = α_{iter+blk-1} [U]^T_{*,blk} [X]_{*,blk};
⟨26⟩       [Z]_{*,blk} = y - μ_{iter+blk-1} [U]_{*,blk};
⟨27⟩     }
⟨28⟩     [wA]_{iter+blk-1:n,blk:m} - =
&            [U]_{iter+blk-1:n,blk} [Z]^T_{iter+blk-1:iter+m-1,blk}
&          + [X]_{iter+blk-1:n,blk} [U]^T_{iter+blk-1:iter+m-1,blk};
⟨29⟩   }
⟨30⟩   [A^{(iter+m-1)}]_{iter:n,iter:iter+m-1} =
&            [wA]_{iter:n,1:m};
⟨31⟩   [A^{(iter+m-1)}]_{iter:n,iter+m:n} - =
&            [U]_{iter:n,1:m} [Z]^T_{iter+m:n,1:m}
&          + [X]_{iter:n,1:m} [U]^T_{iter+m:n,1:m};
⟨32⟩ }
```

図1 列ブロックサイクリック分割方式にともなう並列アルゴリズム
Fig. 1 Parallel blocked algorithm for Hessenberg reduction under the column-wise block cyclic distribution.

とは, 第 k 反復時における k 列ベクトル x から, 変換処理に必要なベクトル u とスカラ α に写像する関数である[4].

この並列アルゴリズムとほぼ同様のアルゴリズムとして, Dongarra と Geijn によるアルゴリズム[2]があるが, 彼らのアルゴリズムは我々が後に述べるデータ分割方式とブロック幅との関係を議論していない.

3.3 問 題 点

3.2節に示す列ブロックサイクリック分割方式に基づくブロック化アルゴリズムの並列化には以下に示す

問題がある．

1) ブロック幅 m を大きくすると並列処理での負荷バランスが悪くなる．
2) ブロック化アルゴリズムを並列化すると，未ブロック化アルゴリズムの並列化版に比べて通信回数と通信量が増す．
3) ブロック化アルゴリズムは未ブロック化アルゴリズムに対して計算量が多くなる．

まず1) の問題の解決法として，データ分割方式は列サイクリック分割方式を採用しつつ，計算はブロック化アルゴリズムに基づいて行うという戦略がある．ところが，本来逐次処理での処理単位であるブロックごとにデータ分割する必然性はない．したがって，このアルゴリズムではブロック内で変換を累積する際にも各反復で通信を必要とするため，ブロック単位でデータを分割することによる不利益は生じないことに注意しておく．またブロック単位でデータ分割しない並列アルゴリズムも，図1の処理で実現可能であることを指摘しておく．

次に 2), 3) の各問題は，計算性能と通信性能のトレードオフであり，最適なブロック幅の決定が困難であることを意味している．これらの理由から，現状では経験的に最適なブロックサイズが決定されている．

4. 性能評価

ここでは256PE の富士通 AP1000+を用いて並列アルゴリズムの性能評価を行った結果を示す．AP1000+の各 PE の理論ピーク性能は 50 MFlops，PE 間は二次元トーラス網で結合されており，その最大転送性能は 25 Mbyte/s である☆．

4.1 逐次アルゴリズムの性能

並列化されたブロック化アルゴリズムの性能評価を妥当なものとするため，逐次処理を十分にチューニングしなくてはならない．ここでは問題サイズを $n = 1000$ に固定し，ブロックサイズ m を 1〜40 まで変化させる場合と，ループアンローリングをしない〜6段までループアンローリングを行う場合とのそれぞれに対し，パラメータを1ずつ変化させて効果を調べた[5]．その結果，ループアンローリングにより十分にチューニングした未ブロック化アルゴリズム（$m = 1$ の場合）の性能に対して，ブロック化アルゴリズムの速度向上は約 1.2 倍であった．一方，チューニングをまったく行わない未ブロック化アルゴリズム（$m = 1$ の場合）に対して，十分にチューニングされたブロック化アルゴリズムは約 2.4 倍もの速度向上が得られることが分かった．

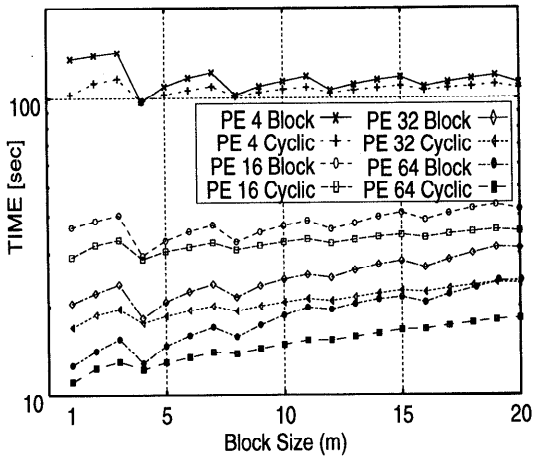

図2 ブロック化 Householder 法の実行時間（$n = 1000$, $p = 4$, 16, 32, 64, **Block**：列ブロックサイクリック分割方式, **Cyclic**：列サイクリック分割方式）

Fig. 2 Execution times for blocked Hessenberg reduction ($n = 1000$, $p = 4$, 16, 32, 64, **Block**: column-wise block cyclic distribution, **Cyclic**: column-wise cyclic distribution).

このことから，十分にチューニングした未ブロック化アルゴリズムに対する性能を評価しないと，ブロック化アルゴリズムの性能が過大評価となる可能性があることが分かる．よって本論文では以降，チューニングした未ブロック化アルゴリズムに対しての性能を評価することにする．

4.2 並列アルゴリズムの性能

アルゴリズムを並列実装する際には，逐次処理と異なる理由でループアンローリングの段数の決定が問題となる．なぜならば，ブロック幅に関するループについてアンローリングの段数を増やすと，ブロック幅もそれに応じて大きくしないとその効果がないからである．しかしブロック幅を増やすと通信時間も増えることはすでに述べたとおりである．よって逐次処理の場合と状況が異なり，最適なループアンローリングの段数が決定し難い．ここではアンローリングを6段まで1ずつ変化させたときの実験結果を考慮し，すべてのブロック幅に対して4段のアンローリングを実装した．

図2 に問題サイズを $n = 1000$ に固定し，PE 台数を 4, 16, 32, 64 と変化させて実行した場合の時間を示す．

図2 から，データ分割を列サイクリックに行う場

☆ 東京大学理学部情報科学科が所有している AP1000+のうち，256台すべてを使用した．また，セルプログラム用のコンパイラとして gcc version2.6.3，オプションとして-O3 -msupersparc -fcaller-saves -fomit-frame-pointer -funroll-loops を指定した．測定日は 1997 年 9 月 16 日である．

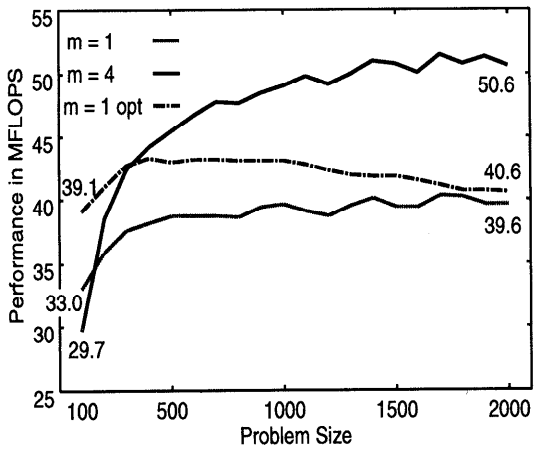

図3 サイクリック分割方式によるブロック化 Householder 法の性能 ($p = 4$)

Fig. 3 Performances under the column-wise cyclic distribution for Hessenberg reduction ($p = 4$).

合，列ブロックサイクリックに行う場合と比較して高速となっていることが分かる．これには2つの理由が考えられる．すなわち，i) アンローリング実装の難しさ，ii) 負荷バランスの悪さ，の2つである．i) の理由は，本実装ではブロック幅 m に関するループでアンローリングしている．たとえば4段アンローリングの場合，ブロック幅 m が4の倍数でないとその効果がない．もちろん所有する列のすべての長さを調べてアンローリングすると，m が小さな領域では列サイクリック分割方式の結果とほぼ一致するはずである．実際図2から分かるように，各PEに所有するデータが十分大きな場合には，$m = 4, 8, 16, 20$ のときに実行時間がほぼ一致している．しかしながら，そのようなアンローリングを効果的に列ブロックサイクリック分割方式に実装するのは困難である．また ii) の理由は，列ブロックサイクリック分割方式の本質的な欠点である．

次に，PEあたりに所有するデータ量が十分大きい場合のブロック化アルゴリズムの効果を調べる．そのためにPE台数 $p = 4$ に固定し，問題サイズ n を100から2000まで変化させて性能を調べた．その結果を**図3**に示す．ここで図3中の $m = 1$ opt とは，ブロック化アルゴリズムのプログラム中から余分なワークエリアを取り除いて最適化したプログラム☆による実行を示している．またMFlops値の計算にあたって，理論演算量として $10/3n^3$ を利用した．

図3から問題サイズが大きくなるに従い，最適化された未ブロック化アルゴリズムでさえも40.6 MFlopsとその性能が低下していくが，ブロック化されたアルゴリズムでは50.6 MFlopsと性能が引き出されている．

☆ ブロック化アルゴリズムにおける $m = 1$ の場合と同様の4段のループアンローリングを実装している．

5. おわりに

分散メモリ型並列計算機AP1000+での実装評価の結果から，PEあたりのデータ量が十分に大きな場合は並列ブロック化アルゴリズムが有効となりうることが分かった．

Householder 法の並列化のためのデータ分割方式は，ブロックサイクリック分割方式である必然性はなく，むしろサイクリック分割方式の方が実装の簡易さと性能との面からみても有効であることも分かった．

今後の課題として，PE間のネットワークアーキテクチャおよびPEの計算方式が異なる分散メモリ型計算機に実装して評価すること，およびこのアルゴリズムに基づく並列固有値ソルバーを実現すること，があげられる．一方，我々がすでに提案した二次元的にサイクリック分割した方式[4]による並列Householder変換ルーチンのブロック化の検討も必要である．

参考文献

1) Bischof, C. and van Loan, C.: The WY Representation for Products of Householder Matrices, *SIAM J. Sci. Stat. Comput.*, Vol.8, No.1, pp.s2–s13 (1987).
2) Dongarra, J.J. and van de Geijn, R.A.: Reduction to Condensed Form for the Eigenvalue Problem on Distributed Memory Architectures, *Parallel Computing*, Vol.18, pp.973–982 (1992).
3) Choi, J., Dongarra, J.J. and Walker, D.W.: The Design of a Parallel Dense Linear Algebra Software Library: Reduction to Hessenberg, Tridiagonal, and Bidiagonal Form, Technical Report ORNL/TM-12472, Oak Ridge National Laboratory (1995).
4) 片桐孝洋，金田康正：分散メモリ型並列計算機による Householder 法の性能評価，情報処理学会研究報告，96-HPC-62, pp.111–116 (1996).
5) 片桐孝洋，金田康正：分散メモリ型並列計算機による固有値計算のためのブロック化 Householder 法の性能評価，情報処理学会研究報告，97-ARC-123, pp.13–18 (1997).

(平成 9 年 9 月 18 日受付)
(平成 10 年 5 月 8 日採録)