超並列処理に向く効果的な並列固有値計算法

片 桐 孝 洋[†,††] 金 田 康 正[†††]

本論文では，密対称行列の全固有値を計算する効率の良い並列アルゴリズムについて述べる．このアルゴリズムは，行列の次元が小さい場合や超並列処理を行う場合に特に効果的である．また 1024 台構成の日立の分散メモリ型並列計算機 SR2201 を用いて性能評価を行った．その性能評価の結果，我々のルーチンは広く用いられている ScaLAPACK の同種ルーチンに比べて，行列の次元が小さいときに約 2〜5 倍高速であることが明らかになった．

An Efficient Implementation of Parallel Eigenvalue Computation for Massively Parallel Processing

Takahiro Katagiri[†,††] and Yasumasa Kanada[†††]

In this paper, we describe an efficient parallel algorithm which can compute all eigenvalues for dense symmetric matrices. We construct this algorithm to establish high efficiency when the dimension of matrices is small or we were in a massively parallel processing environments. We evaluated the algorithm by using 1024 processors of the HITACHI SR2201, which is a distributed memory parallel computer. From the results of the evaluation, we could find that our routine based on our algorithm was 2–5 times as fast as the ScaLAPACK routine which is widely used as a parallel library when the dimension of matrices is small.

1. はじめに

密対称行列の固有値問題を解くためのソルバは多くの研究者によって並列化されてきている[1)〜9)]．しかしながら，超並列処理（Massively Parallel Processing；MPP）向きの並列化についてはあまり考慮や実装評価がされてこなかった．この理由は，(1) 実際に利用可能な MPP マシンがほとんどなかったこと；(2) 効率的な MPP 向きの実装が困難であったこと；に起因するものと考えられる．特に行列の次元が小さく多くのプロセッサを用いて並列処理する場合，十分に性能を引き出すことが困難である．このような状況は MPP の実行環境でよく生じる．

今日 MPP のための並列計算機が構築され始めており，それらは実際に利用可能な数百のプロセッサを有する．このような MPP 環境では，従来のアルゴリズムに基づくソフトウェアでは十分に性能を引き出せないことが予想される．なぜならばこれらの従来ソフトウェアは，たかだか数十のプロセッサ環境しか想定していないからである．

さて密対称行列の固有値を計算するための効率の良いアルゴリズムとして，Householder アルゴリズムが知られている．しかしながら，対称性を利用する従来の Householder アルゴリズム[4),5),7),9)]には，対称性を利用しないアルゴリズムに対して通信時間が増加してしまうという問題がある．もちろん，この従来のアルゴリズムは対称性を利用することで演算量を半分に抑えてはいるが，MPP 環境では通信量の増加について無視することはできない．一方データ参照の局所性を高め，性能を向上させるブロック化を施した三重対角化アルゴリズムの実装も多くなされてきた[4),7),9)]．ところがこれらブロック化アルゴリズムにおいても，対称性利用の議論と同様に通信量が増すという問題があることが知られている[10)]．したがって，対称性を利用する/しない，ブロック化をする/しない，を考慮してアルゴリズムを設計することが MPP 環境では重要である．

そこで我々は，対称性を利用しないうえにブロック

† 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, the University of Tokyo
†† 日本学術振興会特別研究員
Research Fellow of Japan Society for the Promotion of Science
††† 東京大学情報基盤センタースーパーコンピューティング部門
Computer Centre Division, Information Technology Center, the University of Tokyo

化をしない最も通信を減らした三重対角化アルゴリズムが，従来アルゴリズム（対称性を利用，ブロック化を行う）アルゴリズムよりも MPP 環境で有効となる場合があることを実機を用いて示す．さらにこのアルゴリズムを用いることにより，従来の対称性を利用したアルゴリズムに対して行列のサイズが小さいときでも高性能であることを示す．

本論文の構成は以下のとおりである．まず 2 章で，並列計算環境の概要や表記法，およびデータ分散方式についてまとめる．3 章で標準固有値問題のすべての固有値を計算できる並列アルゴリズムを説明し，4 章で日立の分散メモリ型並列計算機 SR2201 を用いて本アルゴリズムの評価を行う．最後に 5 章でまとめを行う．

2. 並列計算環境の概要，表記法，データ分散方式

我々の想定する並列計算機は，均質的な演算素子（PE）で構成されているとする．またそれらの PE は，二次元メッシュ構成 $q \times r = p$（p は PE の数）をとるとともに，各 PE は $P_{myidx,myidy}$，$(myidx = 0,1,\ldots,q-1,\ myidy = 0,1,\ldots,r-1)$ とラベル付けされ，放送や局所的に所有されているデータに対しての加算といったようなリダクション操作が行えるように相互ネットワーク網で接続されているとする．

この論文では，以下に示すよく知られている Householder 法が相似変換で用いられている．

［定理］ ベクトル $x \in \Re^n$ が与えられるとすると，以下に示すベクトル $u \in \Re^n$ とスカラ $\alpha \in \Re$ が存在する：

$$(I-\alpha uu^T)x = (\chi_1,\cdots,\chi_k,\pm\sigma,0\cdots,0)^T,$$
$$\text{ここで } \sigma = ||x_{k+1:n}||_2. \qquad (1)$$

ベクトル $u \equiv (0,\cdots,0,\chi_{k+1}\pm\sigma,\chi_{k+2},\cdots,\chi_n)^T$，とスカラ $\alpha \equiv 1/(\sigma^2+|\chi_{k+1}\sigma|)$ は上記の定理を満たす．また $||u||_2^2 = (\chi_{k+1}\pm\sigma)^2+\chi_{k+2}^2+\cdots+\chi_n^2 = \sigma^2+\sigma^2+2|\chi_{k+1}\sigma| = 2(\sigma^2+|\chi_{k+1}\sigma|) = 2/\alpha$ なので $\alpha u^T u = 2$ である．σ の符号はベクトル u の計算時の桁落ちを防ぐために χ_{k+1} と同符号にとる．ここで変換 $(I-\alpha uu^T)x$ を $H^{(k)}(x)$ と表記する．この変換は要素 $\chi_1,\cdots,\chi_k$ に作用しない．さらに変換 $H^{(k)}(x)$ に必要なベクトルとスカラの組みを (u,α) と表記する．

次に行列のデータ分割方法について述べる．いま Π を行列 A の行方向の添字集合し，Γ を列方向の添字集合とする．ここで $j \in \Pi$，Γ は $1 \le j \le n$ で，n は行列 A の次元である．これらの集合はデータ分散の方式により各 PE で異なる．ここでは，二次元分散方式である (Cyclic, Cyclic) 格子分散方式を以下のように定義する．

$$\Pi = \{myidx+1+(j-1)q\},$$
$$(j=1,2,\cdots,\mathbf{last_c}(myidx+1+q\lfloor n/q\rfloor,\lfloor n/q\rfloor))$$
$$\Gamma = \{myidy+1+(j-1)r\},$$
$$(j=1,2,\cdots,\mathbf{last_c}(myidy+1+r\lfloor n/r\rfloor,\lfloor n/r\rfloor)) \qquad (2)$$

ここで，関数 $\mathbf{last_c}(a,b)$ は

$$\mathbf{last_c}(a,b) = \begin{cases} \text{if} \quad a \le n \quad \text{then} \quad b+1 \\ \text{if} \quad a > n \quad \text{then} \quad b \end{cases} \qquad (3)$$

である．

最後に，**表 1** において並列アルゴリズムを記述するための表記法をまとめる．

表 1 数学的な表記法とその説明
Table 1 Mathematical notation and its explanation.

表記	説明
α，μ，σ	スカラ $\in \Re$.
x，y，u	ベクトル $\in \Re^n$.
χ_i，η_i，υ_i	上記のベクトル x，y，u における i 番目の要素.
x_Π	集合 Π で示された添字から構成した，上記ベクトル x の部分ベクトル.
A	行列 $\in \Re^{n\times n}$.
$A_{i:j,k}$	$i,\cdots,j$ 番目の行と k 番目の列で構成した上記の行列 A の部分ベクトル.
$A_{\Pi,j}$	集合 Π により示された行，および j 列で構成した上記の行列 A の部分行列.
$A^{(k)}$	行列 A の第 k 反復時の行列.

3. 並列アルゴリズムの説明

3.1 全体の処理の概要

ここではすでに処理対象の行列要素は PE 上に分散されているとする．このとき我々の並列アルゴリズムでは，以下に示すよく知られている Householder-bisection 法を用いた 3 つの手順で，標準固有値問題 $Ax_i = \lambda_i x_i$ の全固有値 $\lambda_i \in \Re$ $(i=1,2,\ldots,n)$ を計算する．

手順 1) 並列に密対称行列 A を三重対角行列 T に変換する．
（三重対角化ルーチン）

手順 2) 分散された三重対角行列 T のすべての非ゼロ要素をすべての PE が持つようにする．
（再分散ルーチン）

手順 3) 手順 2) で集まった三重対角行列 T の全固有値を二分法で並列に計算する．
（固有値計算ルーチン）

3.2 三重対角化ルーチン

ここでは $A^{(1)} \equiv A$ から三重対角行列 $A^{(n-2)} \equiv T$ への変換を考える．$H^{(k)} = I - \alpha uu^T$ を行列 A に関する $k+1$ 番目の反復に適用することで，以下に示す式を得ることができる：

$$\begin{aligned} A^{(k+1)} &= H^{(k)} A^{(k)} H^{(k)} \\ &= A^{(k)} - \alpha A^{(k)} uu^T - \alpha uu^T A^{(k)} \\ &\quad + \alpha^2 uu^T A^{(k)} uu^T \\ &= A^{(k)} - xu^T - uy^T + \alpha uu^T xu^T \\ &= A^{(k)} - uy^T + u\mu u^T - xu^T \\ &= A^{(k)} - u(y^T - \mu u^T) - xu^T, \end{aligned} \quad (4)$$

ここで

$$x = \alpha A^{(k)} u,\ y^T = \alpha u^T A^{(k)},\ \mu = \alpha u^T x. \quad (5)$$

いま A は対称なので $x = y$ であり，以下の式を得る．

$$A^{(k+1)} = A^{(k)} - u(x^T - \mu u^T) - xu^T. \quad (6)$$

ここで k 番目の反復では，行列 $A_{k:n,k:n}$ における列ベクトル $A_{k:n,k}$ が必要であることに注意する．この列ベクトル $A_{k:n,k}$ を枢軸ベクトルと呼ぶ．

我々はすでに式 (4)，(6) の変換を用いた並列固有値ソルバ[6),11)]を開発している．**図 1** に (Cyclic, Cyclic) 格子分割方式による並列三重対角化のアルゴリズム[11)]をのせる．図 1 では，⟨2⟩〜⟨12⟩ で枢軸ベクトルの転送，⟨13⟩〜⟨15⟩ で行列–ベクトル積 $x = \alpha A^{(k)} u$，⟨22⟩〜⟨24⟩ で内積計算 $\mu = \alpha u^T x$，そして ⟨25⟩〜⟨29⟩ で行列更新 $A^{(k+1)} = A^{(k)} - u(x^T - \mu u^T) - xu^T$ の各処理を行っている．

図 1 では PE 間での総和演算（リダクション操作）が k に関する各反復ごとに必要であることに注意する．**表 2** に図 1 の三重対角化におけるリダクション操作の通信量をまとめる．ここで議論を簡単にするため，PE グリッドの構成は $r \times q$ $(r \leq q)$，かつ q は r で割り切れることとする．またリダクション操作は二進木通信方式で実現するものとする．

表 2 から三重対角化では，リダクション操作の時間は PE グリッドの構成に影響すること，各反復においてリダクション操作を 5 回する必要があり通信量が比較的多いことが分かる．

図 1 に示す三重対角化は，一次元列サイクリック分割方式を用いた三重対角化に比べて放送とリダクション操作における通信量を $1/\sqrt{p}$ のオーダで減らせることが知られている．同様のアルゴリズムを文献 9)，12)，13) にみることができる．しかしながら，我々の三重対角化は対称性を利用していないことから従来の三重対角化[7),9)]の計算量である $4/3n^3$ に対して，2 倍の計算量である $8/3n^3$ を必要とする．

c　　$P_{myidx,myidy}$ owns row set Π and
c　　　column set Γ of $n \times n$ matrix A.
⟨1⟩ **do** k=1, $n-2$
⟨2⟩ 　**if** ($k \in \Gamma$) **then**
⟨3⟩ 　　Broadcast($A^{(k)}_{\Pi,k}$) to PEs sharing rows Π.
⟨4⟩ 　**else**
⟨5⟩ 　　receive($A^{(k)}_{\Pi,k}$)
⟨6⟩ 　**endif**
⟨7⟩ 　Computation of (u_Π, α).
⟨8⟩ 　**if** (I have diagonal elements of A) **then**
⟨9⟩ 　　Broadcast(u_Π) to PEs sharing columns Γ.
⟨10⟩ **else**
⟨11⟩ 　receive(u_Γ)
⟨12⟩ **endif**
⟨13⟩ **do** j=k, n
⟨14⟩ 　**if** ($j \in \Gamma$) $x_\Pi = x_\Pi + \alpha\ A^{(k)}_{\Pi,j}\ \upsilon_j$ **endif**
⟨15⟩ **enddo**
⟨16⟩ Global summation of x_Π to PEs sharing rows Π.
⟨17⟩ **if** (I have diagonal elements of A) **then**
⟨18⟩ 　Broadcast(x_Π) to PEs sharing columns Γ.
⟨19⟩ **else**
⟨20⟩ 　receive(x_Γ)
⟨21⟩ **endif**
⟨22⟩ **do** j=k, n
⟨23⟩ 　$\mu = \alpha u_\Pi^T x_\Pi$ 　**enddo**
⟨24⟩ Global summation of μ to PEs sharing rows Π.
⟨25⟩ **do** j=k, n
⟨26⟩ 　**do** i=k, n
⟨27⟩ 　　**if** ($i \in \Pi$ **.and.** $j \in \Gamma$) **then**
⟨28⟩ 　　　$A^{(k+1)}_{i,j} = A^{(k)}_{i,j} - \upsilon_i\ (\chi_j^T - \mu\upsilon_j^T) - \chi_i\ \upsilon_j^T$
⟨29⟩ 　　**endif** **enddo** **enddo**
c　　Remove k from active columns and rows.
⟨30⟩ **if** ($k \in \Gamma$) $\Gamma = \Gamma - \{k\}$ **endif**
⟨31⟩ **if** ($k \in \Pi$) $\Pi = \Pi - \{k\}$ **endif**
⟨32⟩ **enddo**

図 1　三重対角化における並列アルゴリズム
（(Cyclic, Cyclic) 格子分割方式）
Fig. 1　Parallel algorithm for the tridiagonalization
(The (Cyclic, Cyclic) grid-wise distribution).

ところがデータ構造とデータ参照パターンが単純なことにより，通信量は従来の三重対角化[7),9)]よりも少ない．この理由は従来アルゴリズムが対称行列用の行列更新を行うために，図 1 ⟨25⟩〜⟨29⟩ の行列更新処理を行列 A の上三角部分しか更新しないように改良した後，以下の 2 通りの実装方法のどちらかを選ばなくてはならないことによる．

(1) **対称行列用データ構造を利用する実装法：**
図 1 ⟨13⟩〜⟨15⟩ の行列–ベクトル積において，行列 A が上三角部分しか保存されていないことから，余分な通信処理と演算カーネルの再変更が必要．

(2) **対称行列用データ構造を利用しない実装法：**
図 1 ⟨25⟩〜⟨29⟩ の行列更新処理を改良したうえで，下三角部分のデータを保存するためのデータ

表 2 三重対角化における第 k 反復で必要なリダクション操作の通信量
Table 2 Communication complexities of reduction operation for the tridiagonalization in the k-th iteration.

行番号	通信回数	1 回あたりの通信量	注釈
⟨7⟩	$\lceil \log_2(r) \rceil$	1	
⟨8⟩〜⟨12⟩	$\lceil \log_2(r) \rceil$	$\lceil (n-k+1)/q \rceil$	$r=q$ なら放送
⟨16⟩	$\lceil \log_2(q) \rceil$	$\lceil (n-k+1)/r \rceil$	
⟨17⟩〜⟨21⟩	$\lceil \log_2(r) \rceil$	$\lceil (n-k+1)/q \rceil$	$r=q$ なら放送
⟨24⟩	$\lceil \log_2(r) \rceil$	1	

表 3 三重対角化における行列–ベクトル積を行うための通信量
Table 3 Communication complexities of matrix-vector product for the tridiagonalization.

実装方式	通信回数	通信量の総和
実装方式 (1)	$4n \lceil \log_2(r) \rceil$	$2n^2 \lceil \log_2(r)/q \rceil$
実装方式 (2)	$n \lceil \log_2(r) \rceil + n\ (p-1)$	$n^2/2 \cdot \lceil \log_2(r)/q \rceil + (p-1)\ (n^3/(3p) + n^2/(2p))$
アルゴリズム図 1	$n \lceil \log_2(r) \rceil$	$n^2/2 \cdot \lceil \log_2(r)/q \rceil$

再分散処理が必要.
さらにブロック化アルゴリズム[7]を用いる場合は，ブロック化のための余分な通信が必要になることにも注意しておく.

表 3 に各実装方式における行列–ベクトル積の通信量をのせる．ここで表 3 の通信量は，通信処理の実装方式により大きく変わる．そこで実装方式 (1) は文献 7) による ScaLAPACK での実装，実装方式 (2) は素朴な実装方式である全対全通信を用いる場合を仮定する.

表 3 から実装方式 (2) は通信量の総和が $O(n^3)$ である．よって他の方法に比べて通信時間が多いことが予想されるのでここでは議論しない.

いま 1 回あたりの浮動少数点演算時間を，図 1 のアルゴリズムにおいては ϕ_1，実装方式 (1) においては ϕ_2 とする．さらに，1 回あたりの通信立ち上げ時間を α，一データの転送時間を β とする．このとき，図 1 のアルゴリズムが実装方式 (1) より高速となる条件は，

$$4n^2/3 \cdot (2\phi_1 - \phi_2) < \log_2(r)(3\alpha + n\beta/q) \quad (7)$$

となる．ここで $n, r, q, \alpha, \beta > 0$ なので，$2\phi_1 - \phi_2 < 0$ のとき，不等式 (7) はつねに成り立つ．したがって，どちらの実装方式が有効かは，図 1 のアルゴリズムの演算性能に大きく依存することが分かる．すなわち図 1 のアルゴリズムにおける演算処理が実装方式 (1) に比べて 2 倍高速であるならば，図 1 のアルゴリズムがつねに高速となる．ここで図 1 のアルゴリズムにおける演算処理の構成が，対称行列用データ構造を利用していないことから単純であることを考えると，実装方式 (1) の演算カーネルより効率良く実装できる可能性があることに注意しておく.

一方問題サイズ n を固定したうえで，演算性能よりも通信性能が十分に悪い（$\phi_1, \phi_2 \ll \alpha, \beta$）として PE 数を十分増加させる（$r, q \to$ 大）と，不等式 (7) が成立しやすくなる．これらの理由から，PE 数が増加するにつれ我々の三重対角化が従来のルーチンに比べて高速になることが期待される.

さらにブロックサイクリック分割（ある幅を持って (Cyclic, Cyclic) 格子分割する分割方式）は n/p が小さい場合に激しい負荷バランスの崩れが生じるために，我々の三重対角化ではこの分割を提供していない．なぜならこのような負荷バランスの崩れは MPP 環境ではよく生じるものと推察され，三重対角化でのブロックサイクリック分割は MPP 環境での並列ルーチンには適さないと考えられるからである[9),10)].

3.3 再分散ルーチン

各 PE 上に三重対角行列全体を所有させるため，(Cyclic, Cyclic) 格子分割された要素をこのルーチンで再分散させる.

3.4 固有値計算ルーチン

このルーチンでは，固有値を計算するための二分法が実装されている．実装方式は文献 14) の`BISECT`ルーチンとほぼ同様である．我々の実装においては，固有値の精度と実行時間に影響する孤立固有値を追い詰める最大反復回数 nbi は 200 としている．しかしながらこのルーチンでは，追い詰めの区間が（マシンイプシロンのような）十分小さな値になると戻るようになっているので，200 回も反復することはない．このルーチンの並列化は，PE ごとに異なった担当範囲を指定することで容易に並列化できる.

4. 性 能 評 価

この章では本論文で示した並列固有値ソルバ（通信ライブラリとして MPI を使用）を日立 SR2201 を用いて評価した結果を記す．なお SR2201 の各 PE の理論ピーク性能は 300 MFlops，PE 間は三次元ハイパクロスバ網で結合されており，その最大転送性能は 300 Mbyte/秒である☆.

☆ 東京大学情報基盤センターが所有している 1024 PE の SR2201 のうち 1024 PE すべてを使用した．またコンパイラとして日立の最適化 FORTRAN90 V02-06-/D，オプションとしては -rdma -W0,'OPT(O(SS))' を指定した．測定日は 1999 年 6 月 22 日から 7 月 6 日である.

表 4 固有値 8000 個を求める場合の実行時間［秒］（$nbi = 200$，解析値との最大相対誤差 0.2493×10^{-7}）．ここで表中の手順 1)〜3) とは，3.1 節の手順を意味している．

Table 4 Calculation time of 8000 eigenvalues in seconds ($nbi = 200$, the maximal relative error with respect to the analytical values is 0.2493×10^{-7}). The notations 1)–3) in the table show the processes in Section 3.1.

(a) 4〜32 PE の場合

PEs (Grid)	4 (2 × 2)	8 (2 × 4)	16 (4 × 4)	32 (4 × 8)
手順 1) （割合%）	1962 (95.1%)	989.5 (94.6%)	490.3 (94.0%)	254.9 (93.8%)
手順 2)	0.002	0.004	0.005	0.006
手順 3) （割合%）	98.57 (4.7%)	55.61 (5.3%)	30.86 (5.9%)	16.79 (6.1%)
総合時間	2061	1045	521.2	271.7
速度向上	1.00	1.97	3.95	7.58

(b) 64〜1024 PE の場合

PEs (Grid)	64 (8 × 8)	128 (8 × 16)	256 (16 × 16)	1024 (32 × 32)
手順 1) （割合%）	119.0 (92.1%)	70.42 (92.7%)	47.90 (93.9%)	63.16 (98.6%)
手順 2)	0.011	0.013	0.025	0.082
手順 3) （割合%）	10.15 (7.8%)	5.469 (7.2%)	3.060 (6.0%)	0.783 (1.2%)
総合時間	129.2	75.90	50.99	64.02
速度向上	15.9	27.1	40.4	32.1

4.1 テスト行列

性能評価とプログラムの動作を確認するため，以下に示す Frank 行列の全固有値を計算した．

$$A_n = (a_{ij}),\ a_{ij} = n - \mathbf{max}(i, j) + 1. \qquad (8)$$

この行列の固有値は解析的に求められ

$$\lambda_k = \frac{1}{2\left(1 - \cos\frac{(2k-1)}{(2n+1)}\pi\right)},\ k = 1, 2, \ldots, n \qquad (9)$$

となることが知られている．

4.2 全固有値計算性能

我々は SR2201 において，問題サイズを 8000 次元に固定し PE 数を 4〜1024 PE まで変化させた場合における実行時間を調べた．**表 4** にこの時間を示す．表 4 から密対称行列の全固有値を計算する場合には，90%以上が三重対角化の時間であることが分かる．すなわち三重対角化の性能が全体の性能に大きく影響を及ぼす．

次に理論計算量を $8/3n^3$ としたときに，PE 数を 4〜1024 と変化させた場合における三重対角化ルーチンの Flops による性能を**図 2** に示す．図 2 から 4 PE での性能は，ピーク性能の 75%で飽和している．一方で 1024 PE での飽和性能は，178 GFlops である．このことから我々の三重対角化ルーチンは，Flops 性能の観点では十分に高性能であるといえる．また図 2 は，PE 数が増加すると効率が低下することも示している．この理由は，PE 数が増加すると全体の実行時間に対する通信時間の割合も増加することによると考えられる．

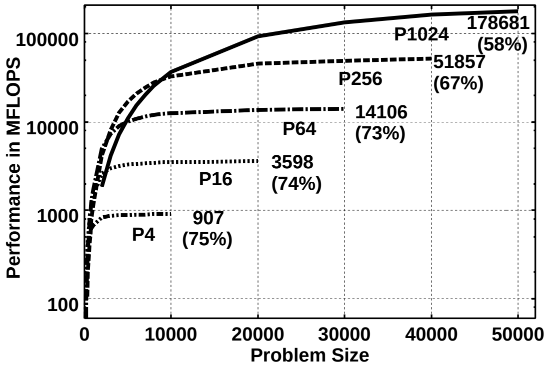


図 2 三重対角化ルーチンの性能（MFlops，括弧中の値は論理ピーク性能に対する効率）

Fig. 2 Performance in parallel tridiagonalization routine. (MFlops. The percentages in parentheses are fractions of the theoretical peak performance).

4.3 SR2201 用 ScaLAPACK との性能比較

表 5 と**表 6** に，ScaLAPACK [4)] の三重対角化ルーチン（以降，SLP TRD）と我々の三重対角化ルーチン（以降，Our TRD）の性能比較の結果をのせる．ここで ScaLAPACK の測定には，日立製作所が SR2201 用にチューニングして提供した PVM 版 ScaLAPACK Version 1.2 [15)] を用いた．ScaLAPACK の演算カーネルである PBLAS は，日立製作所により SR2201 向きに最適化されている．SLP TRD では，対称性を利用したうえでブロック化を行ったアルゴリズムが実装されている[7)]．ブロック化アルゴリズムを利用していることから ScaLAPACK では，ブロックサイズ BL が性能に大きく影響する．**表 7** にそれを示す．

表 7 から分かるように，SR2201 上では BL の幅を 1〜5 と小さくとり負荷バランスを良くしようとすると，性能が十分に発揮できない．この理由は，BL を小さくすると擬似ベクトル化の効果が得られないからと考えられる．一方 Our TRD では，データ分散のブロック幅である BL と，演算カーネルにおけるループ幅が別に設計されているため，このような問題が生じることはない．

文献 15) によると SR2201 では，問題サイズ n が 4000 以下ならば $BL = 60$，問題サイズ n が 4000 より大きいなら $BL = 100$ という指針が示されている．

表 5 三重対角化の性能 I（SR2201）．単位は秒
Table 5 Performance for tridiagonalization I (SR2201). Unit is second.

(a) PE = 4 の場合

Size	SLP TRD (Grid, BL)	Our TRD (Grid)	SLP /Ours
100	0.02 (1×4, 100)	0.056 (2×2)	0.35
200	0.48 (1×4, 100)	0.133 (2×2)	3.6
400	1.73 (1×4, 40)	0.475 (2×2)	3.6
800	6.01 (1×4, 40)	2.454 (2×2)	2.4
1000	9.32 (2×2, 40)	3.785 (2×2)	2.4
2000	41.90 (2×2, 40)	26.937 (2×2)	1.5
4000	231.10 (2×2, 40)	242.010 (2×2)	0.95
8000	1422.69 (2×2, 100)	1962.512 (2×2)	0.72

(b) PE = 8 の場合

Size	SLP TRD (Grid, BL)	Our TRD (Grid)	SLP /Ours
100	0.02 (1×8, 100)	0.125 (2×4)	0.16
200	0.53 (1×8, 100)	0.164 (2×4)	3.2
400	1.77 (1×8, 40)	0.424 (2×4)	4.1
800	5.49 (2×4, 40)	1.659 (2×4)	3.3
1000	7.81 (2×4, 40)	2.476 (2×4)	3.1
2000	29.63 (2×4, 40)	14.838 (2×4)	1.9
4000	142.27 (2×4, 40)	124.205 (2×4)	1.1
8000	815.12 (4×2, 100)	989.504 (2×4)	0.82

(c) PE = 16 の場合

Size	SLP TRD (Grid, BL)	Our TRD (Grid)	SLP /Ours
100	0.03 (1×16, 100)	0.082 (4×4)	0.36
200	0.82 (8×2, 100)	0.195 (4×4)	4.2
400	1.92 (1×16, 40)	0.419 (4×4)	4.5
800	5.48 (4×4, 40)	1.733 (4×4)	3.1
1000	7.53 (2×8, 40)	1.824 (4×4)	4.1
2000	23.00 (4×4, 40)	8.649 (4×4)	2.6
4000	92.21 (4×4, 60)	56.239 (4×4)	1.6
8000	474.49 (4×4, 60)	490.346 (4×4)	0.96

表 6 三重対角化の性能 II（SR2201）．単位は秒
Table 6 Performance for tridiagonalization II (SR2201). Unit is second.

(a) PE = 64 の場合

Size	SLP TRD (Grid, BL)	Our TRD (Grid)	SLP /Ours
100	0.21 (4×16, 100)	0.153 (8×8)	1.3
200	0.98 (1×64, 100)	0.278 (8×8)	3.5
400	2.82 (4×16, 100)	0.638 (8×8)	4.4
800	6.60 (8×8, 40)	1.402 (8×8)	4.7
1000	8.79 (8×8, 40)	1.612 (8×8)	5.4
2000	20.73 (8×8, 40)	5.105 (8×8)	4.0
4000	57.6 (8×8, 40)	19.631 (8×8)	2.9
8000	210.49 (8×8, 60)	119.065 (8×8)	1.7

(b) PE = 128 の場合

Size	SLP TRD (Grid, BL)	Our TRD (Grid)	SLP /Ours
200	1.93 (8×16, 100)	0.462 (8×16)	4.1
400	3.78 (8×16, 60)	0.860 (8×16)	4.3
800	7.68 (8×16, 80)	1.650 (8×16)	4.6
1000	9.74 (8×16, 100)	2.109 (8×16)	4.6
2000	22.17 (8×16, 40)	5.122 (8×16)	4.3
4000	54.13 (8×16, 40)	15.420 (8×16)	3.5
8000	162.01 (8×16, 40)	70.422 (8×16)	2.3
10000	245.60 (8×16, 40)	123.891 (8×16)	1.9

(c) PE = 256 の場合

Size	SLP TRD (Grid, BL)	Our TRD (Grid)	SLP /Ours
400	5.69 (16×16, 60)	1.373 (16×16)	4.1
800	10.17 (16×16, 80)	2.480 (16×16)	4.1
1000	12.89 (16×16, 100)	3.217 (16×16)	4.0
2000	30.12 (16×16, 40)	5.964 (16×16)	5.0
4000	67.29 (16×16, 40)	14.338 (16×16)	4.6
8000	161.76 (16×16, 100)	47.906 (16×16)	3.3
10000	226.11 (16×16, 40)	79.889 (16×16)	2.8
20000	774.10 (16×16, 60)	454.267 (16×16)	1.7

そこでブロックサイズ BL に関して $BL = \{40, 60, 80, 100, 120\}$ の 5 通りをすべて実行し，最も高速であった BL の値と実行時間をのせた．またプロセッサグリッドの構成は文献 15) の指針に従い，なるべく $\sqrt{p} \times \sqrt{p}$ になるようにして実行した．ただし PE 数が少ない場合はその他の構成でも実行して，最も高速になった構成ものせてある．

4.3.1 結　　果

表 5 と表 6 から

(i) 実行時間が 100 秒以下の処理に関して Our TRD は，SLP TRD より 1.4〜5.4 倍高速である（ただし問題サイズ 100 と小さいときは 2〜4 倍遅い）

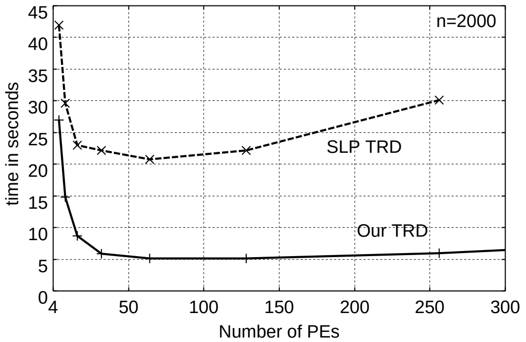
(ii) PE 数が少なく（PE = 4）問題サイズが 4000 以上のとき，SLP TRD は Our TRD に対して 1.3 倍程度高速である

表 7 ブロック幅 BL を変化させた場合における ScaLAPACK の実行時間．単位は秒〔SR2201，$n = 8000$，256 PE（PE グリッド：16×16)〕
Table 7 Execution time of the ScaLAPACK for varying the blocking length of BL in seconds (SR2201, $n = 8000$, 256 PEs (PE grid: 16×16)).

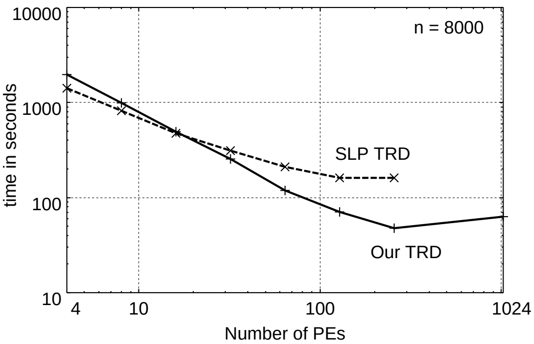
BL	1	2	5	10
SLP TRD	517.37	296.53	201.97	174.02
Speedup	1.00	1.74	2.56	2.97
BL	15	20	25	30
SLP TRD	171.50	156.86	157.28	155.70
Speedup	3.01	3.29	3.28	3.32

ことが分かる．

図 3 は SR2201 での SLP TRD と Our TRD の実行時間を示している．図 3 (a) から問題サイズが 2000 と小さい場合は，Our TRD の方が約 2〜6 倍高速で



(a) $n = 2000$ の場合



(b) $n = 8000$ の場合

図 3　三重対角化における SLP TRD と Our TRD の実行時間（SR2201）

Fig. 3　Execution time between SLP TRD and Our TRD for the tridiagonalizaion (SR2201).

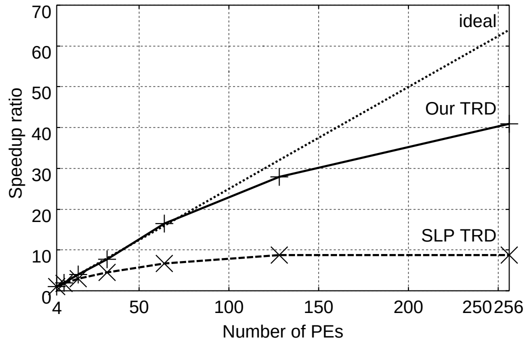


図 4　三重対角化における SLP TRD と Our TRD の台数効果（$n = 8000$，SR2201）

Fig. 4　Speedup ratios between SLP TRD and Our TRD for the tridiagonalization ($n = 8000$, SR2201).

ある．また図 3 (b) から，問題サイズが 8000 と大きくなると PE 数が 4〜16 と少ない場合は SLP TRD が高速である．しかしながら PE 数が増加するにつれ Our TRD の方が高速になる．この性能が逆転する PE 数はおおむね 16 PE である．**図 4** に問題サイズ 8000 の場合の PE = 4 の実行時間に対する台数効果をのせる．問題サイズ 8000 の場合，SLP TRD が 10 倍程度の速度向上しか得られないのに対し Our TRD は 40 倍の速度向上が得られることが図 4 から分かる．ここで SLP TRD は 4 PE においては Our TRD より高速なので，台数効果による評価は正当でないと思える．しかし PE 数が増加するに従い，実行時間でも Our TRD が SLP TRD より高速になる事実から，Our TRD が MPP 環境で十分に高速であることを示している点に注意する．

4.3.2　考　　察

本ソルバは PE あたりのデータ量が小さい問題を解く場合に ScaLAPACK より高速である（図 3 (a)）．このことが生じる理由として

(I) ScaLAPACK の設計方針の誤りによる負荷バランスの劣化

(II) 対称行列用データ圧縮形式による通信量の増加

があげられる．

(I) が生じる理由は，片桐らの指摘[10)]や Hendrickson らの指摘[9)]で述べられている．すなわち並列性能の観点から，**分散メモリ計算機上でのデータ分割のブロック幅とブロック化アルゴリズムのブロック幅を切り離して並列アルゴリズムを設計せよ**ということである[☆]．なぜなら ScaLAPACK ではデータ分割のブロック幅 BL がそのまま BLAS（Basic Linear Algebra Subprograms）2，BLAS 3 の処理単位となっている．それゆえ性能向上のためには，ブロック幅 BL を比較的大きくとる必要がある．このことにより (問題サイズ)/(PE 数) が小さい場合にきわめて負荷バランスが悪くなる．本来逐次計算の性能パラメータであるブロック化アルゴリズムのブロック幅と，データ分散のパラメータであるブロック幅は異なったパラメータであり，完全に区別して議論する必要がある[10)]．性能に関するブロック幅は，データ分散のブロック幅に依存しない．この理由から，Our TRD はデータ分割のブロック幅を 1 に固定した分割方式しか提供しておらず，この負荷バランスの悪化の問題は生じることはない．また Our TRD では演算カーネルにおけるループ長を，データ分散のブロック幅と独立に長くすることができる．

一方 (II) は対称行列用の圧縮形式を利用（SLP TRD）することで，利用しない場合（Our TRD）に比べてデータ構造が複雑なことから通信が増してしま

☆ この主張の一方で寒川[16)]はライブラリ設計の観点において，分散メモリ計算機上でのデータ分割のブロック幅と BLAS 演算のブロック幅を同一にする方が良いとしている．

うことによる．すなわち通信時間が演算時間に対して無視できない状況下では，対称行列の特性を利用しないルーチンの方が高速になるということである．ところが圧縮形式を利用したアルゴリズムは利用しない場合に比べて演算回数が 1/2 になる．三重対角化の計算オーダは $O(n^3)$ なので，問題サイズが大きくなるにつれ総演算時間に占める計算時間の割合が増す．そのため PE あたりの問題サイズが十分大きくなる場合は，圧縮形式版の方が高速となるので圧縮形式を利用した方が良いと推察される．

ところが PE 数が増えるにつれ通信時間が増加するとともに PE あたりの問題サイズも小さくなっていくので，問題サイズも十分に大きくしないと圧縮形式版の利点が得られなくなってくる．このことは表 5 (a), (c) において SLP TRD が Our TRD より高速となる実行時間が約 230 秒（PE = 4，$n = 4000$，0.95 倍）から約 470 秒（PE = 32，$n = 8000$，0.96 倍）にまで増加する事実からも推察される．この増加はある種のアプリケーションではきわめて深刻な事態を引き起こす．たとえば化学計算において，密行列の対角化を 6000 回（もしくはそれ以上）必要とするアプリケーションが存在する[17]．このようなアプリケーションでは，1 回の対角化が 100 秒としても全体の実行時間は約 166 時間（約 7 日）も必要となる．このことは我々の対角化よりも優位になるほど問題サイズを大きくすることは，総合的な計算時間の制約から困難となることを意味している．一方で 100 秒程度の対角化処理において，Our TRD が SLP TRD の約 5 倍程度高速であることを考慮すると，少なくとも 35 日程度の計算が 7 日程度に短縮されることになる．このことは小規模の問題を高速に処理することが重要であることを示している．

以上の (I)〜(II) から PE 数が多い場合，対称行列圧縮形式を用いたライブラリや負荷バランスが悪くなる設計をしているライブラリは，使用目的によっては並列処理の効果がなくなる場合があるといえる．すなわち小さな問題に対して，PE 数が増えても性能が飽和しない特長を持つことが重要である．この点において Our TRD は SLP TRD に対して優れている．

5. おわりに

本論文では密対称行列に対する並列固有値ソルバの実装方式とその性能を述べた．我々のソルバは多くの PE を用いて小さな問題サイズの対角化を行う場合に，ScaLAPACK の同種ルーチンよりも 2〜5 倍程度高速である．本論文で示された並列三重対角化は，MPP 環境を用いる場合や，小さな問題サイズの対角化を多数回行う場合にきわめて有効なツールとなりうる．

また RISC プロセッサを利用する場合，キャッシュを有効利用するブロック化アルゴリズムが有効になることが知られている[7]．しかしながら並列処理においては，このブロック化アルゴリズムはブロック化しない場合に対して通信量が増加する[10]．このことは効果的なプロセッサ内アルゴリズムとプロセッサ間アルゴリズムは異なることを意味する．高性能な並列ライブラリを構築するためにブロック化をする/しない，対称性を利用する/しないアルゴリズムに関して，それらの有効性を理論的に解析する必要がある．この解析とアルゴリズムの実装評価は今後の課題である．

謝辞 日頃討論いただく，東京大学大学院理学系研究科情報科学専攻金田研究室の諸氏に感謝いたします．特に本論文を執筆するにあたり貴重な助言をいただいた，埼玉大学大学院理工学研究科高橋大介博士に感謝いたします．また有益な意見をいただいた，査読者の各位に感謝いたします．なお本研究の一部は，文部省科学研究費特定領域研究（A）「発見科学」（課題番号 10143103），および科学研究費補助金（特別研究員奨励費）の支援により行った．

参考文献

1) Lo, S., Philippe, B. and Sameh, A.: A Multiprocessor Algorithm for the Symmetric Tridiagonal Eigenvalue Problem, *SIAM J. Sci. Stat. Comput.*, Vol.8, No.2, pp.s155–s165 (1987).

2) Chinchalkar, S. and Coleman, T.: Computing Eigenvalues and Eigenvectors of a Dense Real Symmetric Matrix on the NCUBE 6400, Technical Report 91-074, Cornell University, Theory Center (1991).

3) Demmel, J. and Stanley, K.: The Performance of Finding Eigenvalues and Eigenvectors of Dense Symmetric Matrices on Distributed Memory Computers, Technical Report UT-GS-94-254, University of Tennessee, Knoxville (1994).

4) Choi, J., Dhillon, I., Dongarra, J., Ostrouchv, S., Petitet, A., Stanley, K., Walker, D. and Whaley, R.: ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers – Design Issues and Performance, Technical Report 95, LAPACK Working Note (1995).

5) 直野 健，猪貝光祥，山本有作：並列固有値ソルバーの開発と性能評価，並列処理シンポジウム JSPP '96 論文集，pp.9–16 (1996).

6) 片桐孝洋，金田康正：並列固有値ソルバーの実現

とその性能，情報処理学会研究報告，97-HPC-69, pp.49–54 (1997).
7) Stanley, K.S.: Execution Time of Symmetric Eigensolvers, Ph.D Thesis, the University of California at Berkeley (1997).
8) Brent, R., Grosz, L., HarrarII, D.M.H., Kahn, M., Keating, G., Mercer, G., Nielsen, O., Osborne, M. and Zhou, B.: Development of a Mathematical Subroutine Library for Fujitsu Vector Parallel Processors, *Proc. International Conference on Supercomputing '98*, pp.13–20 (1998).
9) Hendrickson, B., Jessup, E. and Smith, C.: Toward an Efficient Parallel Eigensolver for Dense Symmetric Matrices, *SIAM J. Sci. Comput.*, Vol.20, No.3, pp.1132–1154 (1999).
10) 片桐孝洋，金田康正：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，情報処理学会論文誌，Vol.39, No.7, pp.2391–2394 (1998).
11) 片桐孝洋，金田康正：並列固有値ソルバーの実現とその並列性の改良，並列処理シンポジウム JSPP '98 論文集，pp.223–230 (1998).
12) Chang, H., Utku, S., Sakama, M. and Rapp, D.: A Parallel Householder Tridiagonalization Stratagem Using Scattered Square Decomposition, *Parallel Computing*, Vol.6, pp.297–311 (1988).
13) 片桐孝洋，金田康正：分散メモリ型並列計算機に向く Hessenberg 形への変換アルゴリズムとその有効性，情報処理学会論文誌，Vol.39, No.11, pp.3065–3075 (1998).
14) 村田健郎，名取　亮，唐木幸比古：大型数値シミュレーション，pp.157–165, 岩波書店 (1990).
15) 日立製作所：HITACHI SR2201 用 ScaLAPACK，PBLAS ライブラリーのご紹介，東京大学大型計算機センターセンターニュース，Vol.30, No.2, pp.36–58 (1998).
16) 寒川　光：分散メモリ型並列計算機での対称行列のデータ構造と多重スカイライン法への適用，並列処理シンポジウム JSPP '99 論文集，pp.191–198 (1999).
17) 長嶋雲兵：私信 (1999).

(平成 11 年 9 月 1 日受付)
(平成 12 年 2 月 4 日採録)

片桐　孝洋（学生会員）

1973 年生．1994 年豊田工業高等専門学校情報工学科卒業．1996 年京都大学工学部情報工学科卒業．1998 年東京大学大学院理学系研究科情報科学専攻修士課程修了．現在，同大学院博士課程在学中．日本学術振興会特別研究員．並列計算機による行列計算アルゴリズムの研究に従事．大規模固有値問題，並列数値計算に興味を持つ．日本応用数理学会，ACM，SIAM 各学生会員．

金田　康正（正会員）

1949 年生．1973 年東北大学理学部物理第二学科卒業．1978 年東京大学理学系研究科博士課程修了．理学博士．1978 年名古屋大学プラズマ研究所助手．1981 年東京大学大型計算機センター助教授，教授を経て現在東京大学情報基盤センター教授．その間英国ケンブリッジ大学計算機研究所客員研究員，名古屋大学プラズマ研究所客員助教授，核融合科学研究所客員助教授．昭和 58 年度および平成 10 年度情報処理学会論文賞，平成 6 年度情報処理学会 Best Author 賞受賞．著書「π のはなし」（東京図書），共著「アドバンスド・コンピューティング—21 世紀の科学技術基盤」（培風館），編著「Trends in Supercomputing」（World Scientific）．日本応用数理学会，プラズマ・核融合学会，ACM，SIAM 各会員．研究テーマは「大規模数値計算」および「研究の研究」．