

OpenMP を用いた Jacobi-Davidson 法の並列実装とその性能評価

西田 晃[†] 小柳 義夫[†]

大規模疎行列向きの固有値解法である Jacobi-Davidson 法は、従来の Lanczos/Arnoldi 系の解法に比べて高精度な計算が可能であり、また並列化も容易であることから、有力な解法として注目されている。本稿では、Jacobi-Davidson 法において計算量の大部分を占める修正方程式の計算に対して、OpenMP による BLAS レベルでの並列化手法を提案し、その有効性を示すとともに、前処理として並列 Jacobi 法を適用、実装し、その評価結果について報告する。

A Parallel Implementation of the Jacobi-Davidson Method using OpenMP and its Evaluation on Shared Memory Architectures

AKIRA NISHIDA[†] and YOSHIO OYANAGI[†]

The Jacobi-Davidson method is a promising alternative to the Lanczos/Arnoldi approach. In fact, the Lanczos/Arnoldi approach is suitable for computing extreme eigenvalues of general sparse matrices, whereas the Jacobi-Davidson method does not have such restrictions, and it is also highly parallelizable. In this presentation, we discuss the advantages of the BLAS-level parallelizing methodology with OpenMP directives for the correction equation solver, which dominates the computing time of the Jacobi-Davidson method. Also, the parallel Jacobi preconditioner will be implemented and evaluated.

1. はじめに

大規模疎行列の固有値計算アルゴリズムとしては、従来 Lanczos/Arnoldi 系の解法¹⁾を用いるのが一般的であった。比較的小規模な行列においては、全固有値を求める QR 法を用いることができるが、問題の大きさ n に対して $O(n^3)$ の計算量を要するため、この方法では規模の大きな問題を扱うことができない。また、一般に大次元行列の固有値問題では、通常は最大または最小のものから数えて数個までの固有値及び対応する固有ベクトルを求めればよい場合が多い。このため、リスタートを用いた反復 Lanczos/Arnoldi 法は、特に疎行列を扱う場合に最も実際の解法であるといえるが、固有値が近接している場合、正確な計算が難しいことが知られている。Jacobi-Davidson 法^{10),14)}は、量子化学において対角化に用いられることの多い Davidson 法⁵⁾をもとに、Jacobi 法⁷⁾の考え方を用いて構成された解法であるが、比較的条件の悪い場合にも正確に固有値を計算できる^{6),10)}ことから、Lanczos/Arnoldi 法に代わる有力なアルゴリズムとして注目されている。本稿では、Jacobi-Davidson 法において計算量の大部分を占める

修正方程式の計算に対して、OpenMP による BLAS レベルでの並列化手法を提案し、その有効性を示すとともに、前処理として並列 Jacobi 法を適用、実装し、その評価結果について報告する。

2. Jacobi-Davidson 法

2.1 アルゴリズム

Davidson 法では、以下のような手続きで絶対値最大の固有値^{*}を求める。次元 k の部分空間 $\mathcal{K} = \text{span}\{v_1, \dots, v_k\}$ 上で、行列 A の近似固有対、すなわち Ritz 対 (θ_k, u_k) を考える。ここで v_1, \dots, v_k は正規直交基底とする。 u_k を更新するためには \mathcal{K} の次元を拡張する必要があるが、Davidson 法では残差 $r = Au_k - \theta_k u_k$ について修正方程式と呼ばれる以下のような方程式を解く。

$$M_k t = r, \quad M_k = D_A - \theta_k I \quad (1)$$

D_A は A の対角成分である。さらに t を \mathcal{K} と直交化して v_{k+1} を得る。 $V_{k+1} = [v_1, \dots, v_{k+1}]$ と置けば、新しい Ritz 対 (θ_{k+1}, u_{k+1}) は行列

$$H_{k+1} = V_{k+1}^* A V_{k+1} \quad (2)$$

の固有対として計算される。

これに対して、Jacobi-Davidson 法では u_k の直交補空間から更新のための成分を取り出す。以下では u_k は

[†] 東京大学大学院情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, the University of Tokyo

^{*} 以下単に最大固有値と書く。

```

input a starting vector  $v$  and a tolerance  $\epsilon$ ;
compute  $u_1 = v_1 = v / \|v\|_2$ ;
 $w_1 = Av_1$ ,  $\theta = h_{1,1} = w_1^* v_1$ ,  $r = w_1 - \theta v_1$ ;
for  $k = 2, \dots$ 
    solve approximately a  $z \perp u$  from
     $(I - uu^*)(A - \theta I)(I - uu^*)z = -r$ ;
    for  $j = 1, \dots, k - 1$ 
         $z = z - (z^* v_j) v_j$ ;
     $v_k = z / \|z\|_2$ ,  $w_k = Av_k$ ;
    for  $j = 1, \dots, k$ 
         $h_{j,k} = w_k^* v_j$ ;
    compute the rightmost eigenpair  $(\theta, y)$ 
    of the matrix  $H_k$  with  $\|y\| = 1$ ;
    compute the Ritz vector  $u = Vy$ 
    and  $\tilde{u} = Au = Wy$ ;
     $r = \tilde{u} - \theta u$ ;
    stop if  $\|r\|_2 \leq \epsilon$ ;

```

図1 JD法による最大固有値の計算

Fig. 1 Computation of rightmost eigenvalue by Jacobi-Davidson.

正規化されているものと仮定する。固有値問題 $Ax = \lambda x$ を、以下のように u_k の直交補空間 u_k^\perp 上に射影する。行列 A の u_k^\perp への直交射影は

$$A_P = (I - u_k u_k^*) A (I - u_k u_k^*) \quad (3)$$

で表されるが、これは

$$A = A_P + u_k u_k^* A + A u_k u_k^* - \theta_k u_k u_k^* \quad (4)$$

と書き直すことができる。修正ベクトル z は

$$A(z + u_k) = \lambda(z + u_k), \quad z \perp u_k \quad (5)$$

を満たすので、ここに式(4)を代入すれば

$$(A_P - \lambda I)z = -r + (\lambda - \theta_k - u_k^* A z) u_k \quad (6)$$

となる。 $A_P z \perp u_k$, $z \perp u_k$, $r \perp u_k$ より u_k の係数は0でなければならないので、問題は

$$(A_P - \lambda I)z = -r \quad (7)$$

の計算に帰着されることが分かる。実際には λ の値を知ることができないが、式(7)は厳密に解く必要がないため、ここでは代わりに θ_k を用いて

$$(I - u_k u_k^*)(A - \theta_k I)(I - u_k u_k^*)z = -r \quad (8)$$

を解く。得られたベクトルを V_k に対して直交化し、 v_{k+1} とする。 $H_{k+1} = V_{k+1}^* A V_{k+1}$ の最大固有値が次ステップの Ritz 値 θ_{k+1} となる。具体的なアルゴリズムを図1に示す。同様の要領で、減次を用いて複数の固有値を求めることができる。

2.2 前処理

Jacobi-Davidson 法においては、反復法による(8)の計算を効率的に行なう必要がある。そこで以下では Jacobi-Davidson 法の前処理について考える¹¹⁾。式(8)の近似解を \tilde{z} とする。このとき、 $\tilde{z} \perp u_k$ より、

$$(A - \theta_k I)\tilde{z} - \alpha u_k = -r \quad (9)$$

が成り立つので、 $A - \theta_k I$ を M_k で近似すれば、

$$\tilde{z} = -M_k^{-1} r + \alpha M_k^{-1} u_k \quad (10)$$

と表すことができる。この場合にも近似解は u_k と直交する空間に限定されるので、実際には近似演算子として

$$\tilde{M}_k = (I - u_k u_k^*) M_k (I - u_k u_k^*) \quad (11)$$

```

solve  $\tilde{u}$  from  $\tilde{M}_k \tilde{u} = u$ ;
compute  $\tilde{r} \equiv \tilde{M}_k^{-1} r$  as:
    solve  $x$  from  $\tilde{M}_k x = r$ ;
     $\tilde{r} = x - \frac{u^* x}{u^* \tilde{u}} \tilde{u}$ ;
solve approximately  $\tilde{M}_k^{-1} \tilde{A} z = -\tilde{r}$ 
with  $z_0 = 0$ ;

```

図2 左前処理を用いた修正方程式の計算部分

Fig. 2 Computation of correction equation with left preconditioning.

を用いる必要がある。左前処理では、演算子として $\tilde{M}_k^{-1} \tilde{A}$ ($\tilde{A} = (I - u_k u_k^*)(A - \theta_k I)(I - u_k u_k^*)$) を用いる。この場合、反復ベクトル y に対して $\tilde{y} = (A - \theta_k I)y$ と置くと、 $y \perp u_k$ より

$$\tilde{A} y = (I - u_k u_k^*)(A - \theta_k I)(I - u_k u_k^*) y \quad (12)$$

$$= (I - u_k u_k^*)(A - \theta_k I) y \quad (13)$$

$$= (I - u_k u_k^*) \tilde{y} \quad (14)$$

となるので、 $\tilde{M}_k^{-1} \tilde{A} y = \hat{y}$ は

$$\tilde{M}_k \hat{y} = (I - u_k u_k^*) \tilde{y} \quad (15)$$

を解くことによって求めることができる。実際には、 \hat{y} と u_k との直交性から

$$M_k \hat{y} = \tilde{y} - \alpha u_k \quad (16)$$

と書けるので、 $M_k \tilde{y} = \tilde{y}$, $M_k \tilde{u} = u_k$ とすれば

$$\hat{y} = \tilde{y} - \alpha \tilde{u} \quad (17)$$

より \hat{y} の直交条件を用いて

$$\alpha = \frac{u_k^* \tilde{y}}{u_k^* \tilde{u}} \quad (18)$$

を得る。以上をまとめたものを図2に示す。

3. 複数固有値の計算

3.1 JDQR 法

ここでは、Jacobi-Davidson 法において減次を用いて複数の固有値を求める方法について説明する。行列 A の部分 Schur 形を

$$A Q_k = Q_k R_k, \quad k \ll n \quad (19)$$

とする。ここで Q_k は $n \times k$ 正規直交行列、 R_k は $k \times k$ 上三角行列である。このとき R_k の固有対を (x, λ) とすると、 A の固有対は $(Q_k x, \lambda)$ となる。部分 Schur 形は以下のように計算する。

- (1) 正規直交基底 $V_i = [v_1, \dots, v_i]$ に対して、射影行列 $M = V_i^* A V_i$ を求め、QR法により Schur 形 $M = US$, $U^* U = I$ を計算する。 τ に近い固有値を求める場合、 $|s_{i,i} - \tau|$ が昇順に並ぶよう S の列を交換すると、 s_1, s_2, \dots の順に、必要な固有値に近い近似固有ベクトルが並ぶ。この過程で、必要に応じてリスタートを行なう。

- (2) 次に部分空間の拡張を行なう。既に k 次の部分 Schur 形が得られているとすると、

$$A [Q_k, q] = [Q_k, q] \begin{bmatrix} R_k & s \\ & \lambda \end{bmatrix}, \quad (20)$$

$$Q_k^* q = 0 \quad (21)$$

となるベクトル q を定めればよいが、このとき

$$(I - A_k Q_k^*)(A - \lambda I)(I - Q_k Q_k^*) = 0 \quad (22)$$

が成り立つ。 q は

$$\hat{A} = (I - A_k Q_k^*)A(I - Q_k Q_k^*) \quad (23)$$

の固有ベクトルであるので、 \hat{A} に対する Jacobi-Davidson 法により計算する。以上のアルゴリズムを JDQR 法と呼ぶ。

3.2 前処理

$A - \theta I$ に対する前処理行列 M_k が既に得られていると仮定する。 \tilde{Q}_{k-1} を u により拡張したものを \tilde{Q} とおく。 M_k は \tilde{Q} の次元に制限されるので、実際には

$$\tilde{M}_k = (I - \tilde{Q}\tilde{Q}^*)M_k(I - \tilde{Q}\tilde{Q}^*) \quad (24)$$

を用いなければならない。この場合も、以下のように前処理を行なうことができる。

修正方程式の計算において、反復の初期ベクトルは \tilde{Q}_k の空間内にあるので、全反復ベクトルはこの中に含まれる。したがって、 v を反復法から得られるベクトルとすると、この部分空間内でベクトル $z = \tilde{M}_k^{-1}\tilde{A}_k v$ を計算する必要がある。

これは以下のように行なう。 $\tilde{Q}^*v = 0$ より、 $y = (A - \theta I)v$ と置けば

$$\tilde{A}v = (I - \tilde{Q}\tilde{Q}^*)(A - \theta I)(I - \tilde{Q}\tilde{Q}^*)v \quad (25)$$

$$= (I - \tilde{Q}\tilde{Q}^*)y \quad (26)$$

と書ける。左前処理の場合、

$$\tilde{M}_k u = (I - \tilde{Q}\tilde{Q}^*)y \quad (27)$$

を満たす $z \perp \tilde{Q}$ を計算する。 $\tilde{Q}^*z = 0$ より、 z は $M_k z = y - \tilde{Q}\gamma$ 、すなわち $z = M_k^{-1}y - M_k^{-1}\tilde{Q}\gamma$ を満たす。 γ については、 $\tilde{Q}^*z = 0$ より

$$\gamma = (\tilde{Q}^*M_k^{-1}\tilde{Q})^{-1}\tilde{Q}^*M_k^{-1}y \quad (28)$$

から計算できる。

4. 具体的な前処理法の適用

前処理行列 M_k の取り方により、いくつかの方法が考えられる。 $M_k = I$ と取る場合は、

$$\tilde{z} = -r + \alpha u_k = -A u_k + (\theta_k + \alpha)u_k \quad (29)$$

であることから、右辺第1項より Arnoldi 法と同値である。また、 $M_k = A - \theta_k I$ と取る場合は、

$$\tilde{z} = -u_k + \alpha(A - \theta_k I)^{-1}u_k \quad (30)$$

であることから、右辺第2項よりシフト付逆反復法と同値である。 $M_k = \text{diag}(A)$ とする場合もこれに含まれる。

一方、式 (8) を近似的に解く場合は、係数行列 $B = (I - u_k u_k^*)(A - \theta_k I)(I - u_k u_k^*)$ に関して方程式 $B\tilde{z} = -r$ を近似的に解くことになる。実装方式については、QMR 法やブロック ILU 分解による前処理の実装例が報告されているが^{3),9)}、一般行列に対する Jacobi-Davidson 法の前処理としては、

(1) 問題の物理的性質を仮定しないこと

(2) 演算量が小さいこと

(3) 並列性の高い解法であること

の各条件を満たす必要がある。この場合のアルゴリズムは以下のように構成される。

```

solve  $\bar{u}$  from  $M_k \bar{u} = u$ ;
compute  $\tilde{r} \equiv \tilde{M}_k^{-1} r$  as:
    solve  $x$  approximately from  $M_k x = r$ 
    by the preconditioner;
 $\tilde{r} = x - \frac{u^* x}{u^* \bar{u}} \bar{u}$ ;
apply BiCGSTAB to solve  $\tilde{M}_k^{-1} \tilde{A} z = -\tilde{r}$ 
approximately with  $z_0 = 0$  as:
solve  $x'$  from  $M_k x' = (A - \theta_k I)z$ ;
 $y' \equiv \tilde{M}_k^{-1} \tilde{A} z = x' - \frac{u^* x'}{u^* \bar{u}} \bar{u}$ ;

```

図3 前処理を用いた修正方程式の計算部分

Fig. 3 Computation of correction equation with preconditioning.

5. 実装と性能評価

大規模疎行列の反復解法においては、疎な成分を持つ行列とベクトル間の演算が計算量の大部分を占めるため、この演算の効率的な処理が必要不可欠である。これらは Dongarra らによって開発された BLAS (Basic Linear Algebra Subprograms)⁸⁾ を用いて実装することが可能であるが、行列-ベクトル間演算においては、一般に演算量は $O(n^2)$ であり、十分にキャッシュメモリを活用するのは難しい。したがって、疎行列アルゴリズムの並列化を行う場合に、BLAS レベルでの並列化が適切な選択肢であるかどうかを決定するためには、問題の性質について十分な評価が必要である¹²⁾。

BLAS の並列実装に関しては、1) ScaLAPACK⁴⁾ のように、MPI による BLAS と等価な並列化ライブラリを構築するもの、と、2) ATLAS¹³⁾ のように、pthreads により並列化を行うものがあり、1), 2) とともに一部の BLAS ルーチンについては、効率的な並列実装が実現している。ただし、PBLAS は分散メモリアーキテクチャを主な対象としており、また、ATLAS の Level 3 BLAS ルーチンに関する対称型マルチプロセッサ向けの並列化は、本アルゴリズムのような疎行列解法への適用を想定したものではない。

並列演算を実現するソフトウェアを効率的に構築していくためには、ユーザにアーキテクチャの低位構造を意識させることなく高い性能を引き出すことのできる開発環境が求められる。そのような環境を実現するためには、アルゴリズムを設計する段階で明示的にデータを分割する必要のある分散記憶型の並列処理方式には限界があり、単一のアドレス空間を持ち、プロセッサを追加することによりスケラブルに計算資源を拡張することのできる共有メモリ方式を採用することが望ましい。

共有メモリアーキテクチャはメモリアクセス遅延が小

ScaLAPACK Software Hierarchy

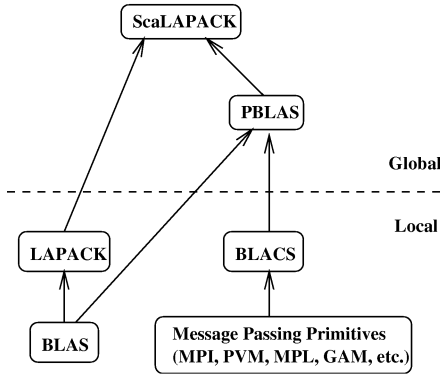


図4 ScaLAPACKのソフトウェア階層
Fig. 4 ScaLAPACK software hierarchy.

さい反面、分散メモリアーキテクチャと比較してキャッシュミスの発生が多く、また、同一のキャッシュラインへの書き込みが発生した場合には性能が低下する場合がある。しかしながら、この点についても留意して実装を行うことにより、高い性能を確保することが可能である。メモリアクセス遅延の小さい共有メモリアーキテクチャ上においては、ループレベルでの並列化によって、Level 1, 2 BLAS 演算についても効率的に性能向上を達成できるものと期待される。

そこで、本アルゴリズムの並列化に当たっては、上記の関数の外側ループを OpenMP API を用いてブロック化し、並列に処理することとした。OpenMP Fortran API の実行モデルでは、プログラムの実行はマスターレッドと呼ばれる単一プロセスとして開始される。マスターレッドは通常のステートメントを逐次実行し、PARALLEL と END PARALLEL 指示文の対で構成される並列構造が現れると、1つ以上のスレッドからなるチームを生成し、チームのメンバのそれぞれについてデータ環境の設定を行なう。並列構造内のステートメントは、チーム内の各スレッドによって並列に実行され、並列構造の終了時点でチーム内のスレッドは同期し、マスターレッドは更新されたデータを用いて計算を続ける。

6. Jacobi-Davidson 法の数値特性

以下では、上記アルゴリズムの実装、及び対称型共有メモリアーキテクチャ上での並列化手法について検討する。プログラムに関しては、Fokkema, van Gijzen⁶⁾ による JDQR 法の実装である JDQZ ルーチンをベースとした。反復解法の記述には Templates²⁾、また線形演算には BLAS, LAPACK を用いている。

評価には、Intel Pentium III Xeon (550MHz, 16KB data, 16KB instruction cache, 512KB L2 cache) を要素とする 4-way SMP (Dell PowerEdge 6300, 450NX chipset, 768MB main memory) 及び Sun Ul-

traSPARC II (250MHz, 16KB data, 16KB instruction cache, 1MB L2 cache) を要素とする 64-way SMP (Ultra Enterprise 10000, 10GB main memory) を用いた OS にはいずれも Solaris 7, またコンパイラにはそれぞれ PGI Fortran 3.2 及び Omni OpenMP Compiler 1.3 を用いた。Omni のバックエンドコンパイラには gcc 2.95.2 を使用した。本章では、予備評価として前者の環境を用いて計算を行った。なお、共有メモリ型アーキテクチャとして、他に cc-NUMA を挙げることはできるが、メモリの局所性について考慮する必要があるため、ここでは扱わない。

解析解を持つ標準固有値問題の例としては、移流拡散差分方程式の係数行列に関する固有値解析を挙げることができる。2次元領域 $D \equiv \{(x, y) : 0 \leq x, y \leq 1\}$ 上での移流拡散問題

$$\nabla u(x, y) + (v_x, v_y) \cdot \text{gradu}(x, y) \quad (31)$$

$$= f, (x, y) \in D \quad (32)$$

$$u(x, y) = 0, (x, y) \in \partial D \quad (33)$$

は、刻み幅 $h = 1/(N + 1)$ の中心差分近似

$$\frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{h^2} \quad (34)$$

$$+ v_x \frac{u_{i+1,j} - u_{i-1,j}}{2h} + v_y \frac{u_{i,j+1} - u_{i,j-1}}{2h} \quad (35)$$

$$= f(i, j), \quad (36)$$

$$u_{i,j} = u(ih, jh), 1 \leq i, j \leq N \quad (37)$$

において、

$$C(v) = \begin{pmatrix} 0 & 1 + \frac{hv}{2} & & O \\ 1 - \frac{hv}{2} & \ddots & \ddots & \\ & \ddots & \ddots & 1 + \frac{hv}{2} \\ O & & & 1 - \frac{hv}{2} & 0 \end{pmatrix}$$

として係数行列

$$A = -4I \otimes I + I \otimes C(v_x) + C(v_y) \otimes I \quad (38)$$

を持つ。 \otimes はクロネッカー積を表す演算子である。この固有対の厳密解は

$$\lambda_{i,j} = -4 + 2\sqrt{1 - \left(\frac{hv_x}{2}\right)^2} \cos(ih\pi) \quad (39)$$

$$+ 2\sqrt{1 - \left(\frac{hv_y}{2}\right)^2} \cos(jh\pi), \quad (40)$$

$$u_{(i,j)} = \begin{pmatrix} \sqrt{\alpha_y} \sin(j\pi h) \\ \sqrt{(\alpha_y)^2} \sin(2j\pi h) \\ \vdots \\ \sqrt{(\alpha_y)^N} \sin(Nj\pi h) \end{pmatrix} \quad (41)$$

$$\otimes \begin{pmatrix} \sqrt{\alpha_x} \sin(i\pi h) \\ \sqrt{(\alpha_x)^2} \sin(2i\pi h) \\ \vdots \\ \sqrt{(\alpha_x)^N} \sin(Ni\pi h) \end{pmatrix}, \quad (42)$$

$$\alpha_x = \frac{1 - \frac{hv_x}{2}}{1 + \frac{hv_x}{2}}, \alpha_y = \frac{1 - \frac{hv_y}{2}}{1 + \frac{hv_y}{2}} \quad (43)$$

で与えられるため, Jacobi-Davidson 法の評価に用いることが可能である. 本稿ではこれを簡略化し, 1次元及び2次元拡散方程式の差分係数行列について評価を行った.

対角要素 2, 副対角要素 -1 の n 次 3 重対角行列 A_1 , 及び $n = N^2$ 次 5 重対角行列

$$A_2 = \begin{pmatrix} T_N & -I & & O \\ -I & \ddots & \ddots & \\ & \ddots & \ddots & -I \\ O & & -I & T_N \end{pmatrix}, \quad (44)$$

$$T_N = \begin{pmatrix} 4 & -1 & & O \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ O & & -1 & 4 \end{pmatrix} \quad (45)$$

の固有値は, それぞれ解析的に $2 - 2 \cos[k\pi/(n+1)]$, $k = 1, \dots, n$, $4 - 2(\cos(k\pi/(N+1)) + \cos(j\pi/(N+1)))$, $j, k = 1, \dots, N$ で与えられる.

ここでは, 残差の許容範囲を 10^{-8} として, A_1, A_2 の最大固有値を計算した. 探索空間の基底数は $10-15$ の範囲とし, 標準 Petrov 空間上で反復ベクトルを生成した. 修正方程式の計算には BiCGSTAB(4) を用いた.

まず, 問題サイズと逐次での計算時間の関係を表 1 に示す.

表 1 Jacobi-Davidson 法による最大固有値の計算時間
Table 1 Time for Computation of rightmost eigenvalue by Jacobi-Davidson.

Size n	Time(s)	
	A_1	A_2
32^2	2	1
64^2	50	12
128^2	4328	60
256^2	—	396

表 2 は 128^2 次の A_1 について計算した場合のプロファイラによる解析結果であるが, BiCGSTAB の計算時間が全体の約 85% を占めている. A_1, A_2 の条件数はそれぞれ

$$\frac{4}{2 - 2 \cos\left(\frac{\pi}{n+1}\right)} \approx \frac{4}{\pi^2} n^2, \quad (46)$$

$$\frac{8}{4 - 4 \cos\left(\frac{\pi}{N+1}\right)} \approx \frac{4}{\pi^2} N^2 \quad (47)$$

で見積もることができるので, A_1 の場合の収束が遅い原因は反復解法の収束の遅れにあると考えられるが, このように, Jacobi-Davidson 法においては修正方程式の計

算の高速化が不可欠であることが分かる.

表 2 $A_1, n = 128^2$ での実行結果
Table 2 Result for A_1 with $n = 128^2$.

Function	Calls	Cost(s)	Cost(%)
jdqz	1	4328	100.00
zcgstabl	990	3659	84.55
zgemv	298096	1064	24.59
zdotc	344829	794	18.35
zaxpy	230778	719	16.63
...			

次に, 256^2 次の A_2 について, 逐次で実行した場合の解析結果を表 3 に示す. ここでは各関数の実行時間を個別に集計している. 最大固有値は 5 個まで求めた.

表 3 $A_2, n = 256^2$ での実行結果
Table 3 Result for A_2 with $n = 256^2$.

Function	Calls	Time(s)	Time(%)
zgemv	10246	319	33.40
zdotc	12615	150	15.77
zaxpy	8163	116	12.14
jdqz	1	109	11.50
jdqzmv	3479	54	5.74
zxpav	4193	54	5.68
dznrm2	5402	52	5.50
amul	3540	47	5.00
bmul	3540	31	3.26
zcgstabl	41	12	1.33
zmgs	369	3	0.32
...			

なお, zgemv は

```
y := alpha*A*x + beta*y,
y := alpha*A'*x + beta*y,
y := alpha*conjg( A' )*x + beta*y,
```

型の Level 2 BLAS, また, zdotc は

```
z := conjg(x)*y,
```

zaxpy, zxpav はそれぞれ

```
y := alpha*x + y,
```

```
y := x + alpha*y
```

型の Level 1 BLAS 演算である.

このように, Jacobi-Davidson 法では Level 1, Level 2 BLAS 演算, 特に zgemv 型の演算が計算量の大部分を占める. ただし, 疎行列を対象としているため, 行列-ベクトル積の演算量はほぼ $\mathcal{O}(n)$ である.

並列化に当たっては, まず, zgemv, zdotc, zaxpy, zxpav の 4 関数の外側ループを OpenMP API を用いてブロック化したルーチンを作成し, 並列に処理した. それぞれの演算のループは静的に各スレッドへ割り当てた. また, ループ内での変数の私有化によりメモリアクセスの競合を最小限に抑えるとともに, 可能な限り陰的なバリア同期を行わないよう指定する. なお, ノルム演算ルーチン dznrm2 では十分な並列化効率が得られな

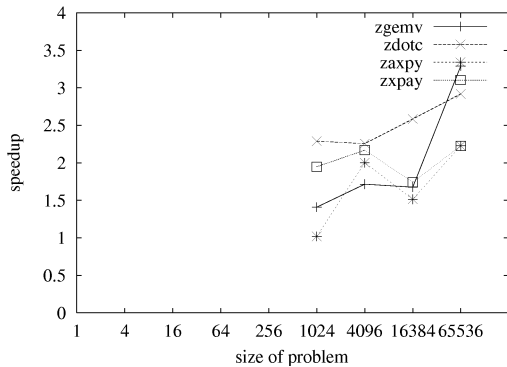


図5 問題サイズと各ルーチンの速度向上率
Fig. 5 Size of problems and speedups.

かったため、ここでは並列化を行っていない。

これらのサブルーチンをそれぞれ連続して複数回実行し、計算時間を測定した。まず、4-way SMP 上での並列化後の実行結果を表 4 に示す。表から、並列化により、各ルーチンについてそれぞれスピードアップが得られていることがわかる。実行時間の大きな部分を占めている Level 2 BLAS の `zgemv` についても、計算のオーダがおおよそ $O(n)$ であるため、上記の並列化により十分高い速度向上率が得られている。また、次元を変えて実行時間をみたものを図 5 に示す。次元が大きくなるにしたがって速度向上率がスレッド数に近づくが、複素乗算を含む `zgemv`, `zdotc` 型の演算で、より高い効果が得られていることが分かる。なお、問題サイズ 16384 での性能低下は、L1 キャッシュが溢れたことによるものと考えられる。

表 4 $n = 256^2$ での並列実行結果
Table 4 Result for $n = 256^2$ in parallel execution.

Function	Calls	Time(s)	Time(%)
jdqz	1	113	19.95
zgemv	9814	96	17.12
zaxpy	7818	54	9.69
zxpvy	3954	51	9.12
zdotc	12212	50	8.93
jdqzmv	3290	50	8.85
dznrm2	5296	45	8.09
amul	3352	42	7.56
bmul	3352	42	7.47
zcgstabl	42	11	2.02
zmgs	376	3	0.56
...			

7. Enterprise 10000 上への並列実装

次に、Sun Enterprise 10000 上での実行結果について述べる。Enterprise 10000 はセンタプレーンにより相互接続された 16 枚のシステムボードから構成され、

ボード 1 枚あたり最大 4 個のプロセッサモジュール、及び 4 バンク、4GB のメモリを搭載可能である。なお本環境には 1 ボードあたり 4 個の 250MHz UltraSPARCII プロセッサ、及び 512MB または 1GB のメモリが搭載されている。各ボードはグローバルデータルータによって相互に接続され、クロスバ総帯域幅は最大 12.8GB/sec である。

Omni では、Solaris プラットフォーム上でスレッドをプロセッサに固定することが可能である。今回の評価ではキャッシュの利用が前提となるため、プロセッサに対して `light weight process` を固定している。

BLAS は本来レジスタ、キャッシュレベルでの最適化をアプリケーションプログラムから分離して行うためのライブラリであり、並列化を低レベルで実現する目的で BLAS を使用するためには、条件分岐の見直しやループ順序の変更など、SMP 上での並列処理に適した効率的な実装が必要となる。

Level 2 BLAS である `zgemv` において、BLAS の reference implementation では、以下のように行列・ベクトル積を

$$y(1:m) = y(1:m) + \alpha M(1:m, j)x(j), \quad (48)$$

$$j = 1, 2, \dots, n \quad (49)$$

として計算している。

```
*
* Form y := alpha*A*x + y.
*
      DO 60, J = 1, N
        IF( X( JX ).NE.ZERO )THEN
          TEMP = ALPHA*X( JX )
          DO 50, I = 1, M
            Y( I )
            $          = Y( I ) + TEMP*A( I, J )
          50 CONTINUE
        END IF
        JX = JX + INCX
      60 CONTINUE
```

この演算はレジスタレベルでの最適化を目的としたものであるが、一般的な Level 2 BLAS 演算においては、ベクトル y と行列 M の行とを

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} + \alpha \begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_k \end{pmatrix} \times x \quad (50)$$

としてブロック化し、以下のように各ブロックでのベクトル演算を並列に処理する必要がある。

```
*
* Form y := alpha*A*x + y,
*
!$OMP PARALLEL PRIVATE(JX, TEMP, TEMP_SUM)
!$OMP+ FIRSTPRIVATE(KX, INCX, ALPHA)
```

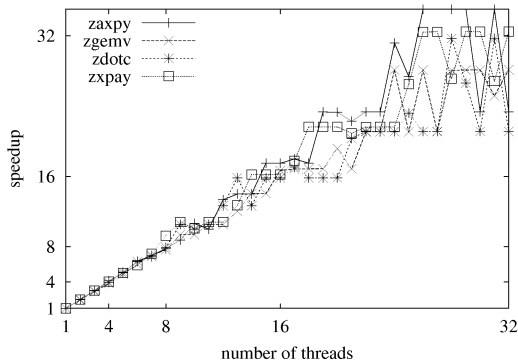


図 6 $n = 256^2$ での各ルーチンの速度向上率
Fig. 6 Speedups of subroutines for $n = 256^2$.

```
!$OMP DO
DO 50, I = 1, M
  TEMP_SUM = (0, 0)
DO 60, J = 1, N
  JX = KX + J * INCX - INCX
  IF( X( JX ).NE.ZERO )THEN
    TEMP = ALPHA*X( JX )
    TEMP_SUM
  $
    = TEMP_SUM + TEMP*A( I, J )
  END IF
60 CONTINUE
  Y( I ) = Y( I ) + TEMP_SUM
50 CONTINUE
!$OMP END DO NOWAIT
!$OMP END PARALLEL
```

図 6に、問題サイズ 256^2 でスレッド数を変化させた時の関数 `zgemv`, `zdotc`, `zaxpy`, `zxpax` のスピードアップを示す。Enterprise 上では、Level 1, Level 2 BLAS とほぼ線形な性能向上が得られている。なおスレッド数が増加するにつれて性能の変動が激しくなっているが、これはスレッドに割り当てられる負荷の不均衡の相対的な増大によるものと考えられる。

8. Jacobi 前処理

前節の例から、Jacobi-Davidson 法において修正方程式の求解に要する計算量が大きいたことが分かるが、適当な前処理を行なうことにより、この計算量を削減することができる。

ここでは行列形式のみを利用する解法として、高い並列性を持つことが知られている Jacobi 反復法を適用することを考える。Jacobi 法では、係数行列は対角優位性を満たせばよく、修正方程式の計算においても高い収束性を示すと予想される。

以下では前述の行列 A_2 を用いて評価を行った。まず、問題サイズ 128^2 での適用結果を図 7 に示す。横軸は Jacobi-Davidson 法の反復回数、縦軸は残差を示す。前処理には、並列化した Jacobi 法を用いた。ここでは、

Jacobi 法の反復回数は総計算時間が最小となる 150 回とした。

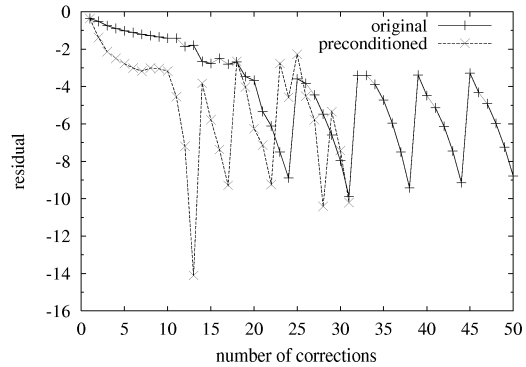


図 7 $n = 128^2$ での Jacobi 前処理の効果
Fig. 7 Performance enhancement with Jacobi preconditioner for $n = 128^2$.

この例では、前処理を用いた場合の反復回数は、前処理を行なわない場合の約 50% であり、計算時間についても約 50% である。解法全体において反復解法の占める割合は大きく、またこの部分の並列化効率、アルゴリズムが複雑であるためそれほど高くない。Jacobi 前処理の並列化効率はほぼ 100% であることから、前処理を行なうことにより、Jacobi-Davidson 法の並列化効率を損なうことなく計算時間を短縮できることが分かる。

次に図 8 に Sun Enterprise 10000 上での速度向上率を示す。ここでは問題サイズは 256^2 とし、最大固有値 1 個を計算した。Jacobi 前処理、係数行列・ベクトル積、`zgemv`, `zdotc`, `zaxpy`, `zxpax` の 4 ルーチンを並列化している。Jacobi 法の反復回数は 20 回とした。

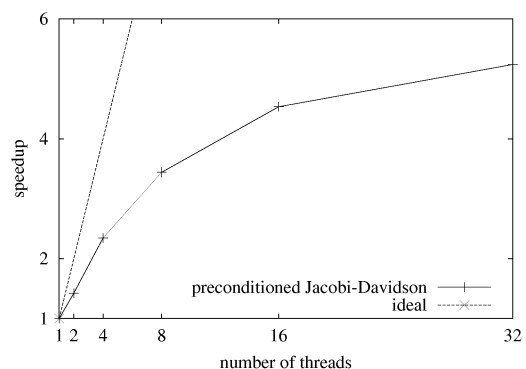


図 8 $n = 256^2$ での Jacobi 前処理付 JD 法の速度向上率
Fig. 8 Speedup of Jacobi preconditioned JD for $n = 256^2$.

並列化された BLAS ルーチンの逐次での実行時間に占める割合は、この場合 76% であるが、実際にはこの他に係数行列 - 反復ベクトル間演算のためのルーチンであ

る `amul`, `bmul` が並列化されており, また `dznrm2` のように十分な並列化効率が得られなかったものも含まれているため, 本稿の趣旨である BLAS レベルでの並列化に関しては, ほぼ完全に実現されていると考えてよい. 図 8 において速度向上率は 5.24 倍となっていることから, 並列化されたコードの割合は約 84% であることが分かるが, これは実際の数値と一致しており, BLAS レベルでの並列化によるオーバーヘッドは無視できる程度であることを示している.

9. まとめと今後の課題

本稿では, Jacobi-Davidson 法において計算量の大部分を占める修正方程式の計算に対して, 共有メモリ型並列計算機向けの解法である Jacobi 前処理を適用, 実装し, その評価結果について報告した. 特に, Jacobi 前処理を用いることにより, Jacobi-Davidson 法の計算時間に関して約 2 倍の性能向上が得られること, また, OpenMP を用いたループ並列化により, 前処理付 Jacobi-Davidson 法において計算量の大部分を占める Level 1, Level 2 BLAS 演算においても, 大規模 SMP 上で効率的に並列実行できることを示した.

本手法は比較的新しい解法であるため, 特性については明らかになっていない点も多い. 今後, 大規模固有値解法に対する有力な解法の一つとして様々な評価を行なっていく必要がある. 疎行列アルゴリズムにおいて必要となる Level 1, 2 BLAS 演算についても, プロセッサレベルでの最適化と合わせて, 今後実験・評価を通じてより効率的な実装方法を明らかにしていきたい.

謝辞 本研究の一部は, 科学研究費補助金奨励研究 (A) 11780194, 13780200, 及び基盤研究 (B) 13480080 によるものである.

参考文献

- 1) Z. BAI, J. DEMMEL, J. DONGARRA, A. RUHE, AND H. VAN DER VORST, eds., *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, 2000.
- 2) R. BARRETT, M. BERRY, T.F. CHAN, J. DEMMEL, J. DONATO, J. DONGARRA, V. ELJKHOUT, R. POZO, C. ROMINE, AND H. VANDER VORST, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, 1994.
- 3) A. BASERMANN, *Parallel Jacobi-Davidson Methods with Iterative Preconditioning for the Solution of Large Sparse Hermitian Eigenproblems*, in Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, CD-ROM, SIAM, Philadelphia, 1999.
- 4) L. S. BLACKFORD, J. CHOI, A. CLEARY, E. D'AZEVEDO, J. DEMMEL, I. DHILLON, J. DONGARRA, S. HAMMARLING, G. HENRY, A. PETITET, K. STANLEY, D. WALKER, AND R.C. WHALEY, *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics, 1997.
- 5) E. R. DAVIDSON, *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real symmetric matrices*, J. Comp. Phys., 17 (1975), pp. 87–94.
- 6) D. R. FOKKEMA, G. L. G. SLEIJPEN, AND H. A. VAN DER VORST, *Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix pencils*, Tech. Rep. 941, Department of Mathematics, Utrecht University, 1996.
- 7) C. G. J. JACOBI, *Über ein leichtes Verfahren, die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen*, Journal für die reine und angewandte Mathematik, (1846), pp. 51–94.
- 8) L. LAWSON, R. J. HANSON, D. KINCAID, AND F. T. KROGH, *Basic Linear Algebra Subprograms for FORTRAN usage*, tech. rep.
- 9) M. NOOL AND A. VAN DER PLOEG, *A parallel Jacobi-Davidson-Type method for solving large generalized eigenvalue problems in magnetohydrodynamics*, SIAM J. Sci. Comput., 22 (2000), pp. 95–112.
- 10) G. L. G. SLEIJPEN AND H. A. VANDER VORST, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.
- 11) G. L. G. SLEIJPEN, H. A. VAN DER VORST, AND E. MEIJERINK, *Efficient expansion of subspaces in the Jacobi-Davidson method for standard and generalized eigenproblems*, Tech. Rep. 1047, Department of Mathematics, Utrecht University, 1998.
- 12) S. TOLEDO, *Improving Memory-System Performance of Sparse Matrix-Vector Multiplication*, IBM Journal of Research and Development, 41 (1997), pp. 711–725.
- 13) B. C. WHALEY AND J. DONGARRA, *Automatically Tuned Linear Algebra Software*, in Proceedings of SC98, 1998.
- 14) 西田晃, 小柳義夫, 大規模固有値問題のための Jacobi-Davidson 法とその特性について, 情報処理学会論文誌: ハイパフォーマンスコンピューティングシステム, 41 (2000), pp. 101–106.