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，様々な研究がなされている．本稿では，OpenMP を用いた共有メモリアーキテクチャ上での並列実装の詳細とその評価結果について報告する．

Parallel Implementation of the Jacobi-Davidson Method and its Evaluation

Akira Nishida[†] and Yoshio Oyanagi[†]

The Jacobi-Davidson method, which computes high quality eigenpairs of general sparse matrices, is one of the promising alternatives to the Lanczos/Arnoldi approach, and are studied and applied to various problems. In this paper, we discuss the parallelizing methodology with OpenMP directives for shared memory architectures, and report its implementation and performance evaluation on a symmetric multiprocessor.

1. はじめに

大規模疎行列の固有値計算においては，従来 Lanczos/Arnoldi 系の解法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，問題サイズ n に対して $\mathcal{O}(n^3)$ 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．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的な解法であるといえるが，固有値間の分離が十分でない場合に，正確に固有値を計算することが難しいことが知られている．

Jacobi-Davidson 法は，このように比較的条件の悪い場合にも正確な固有値を計算できる[6),9)] ことから，Lanczos/Arnoldi 法に代わる有力な固有値解法として注目されている．しかし，アルゴリズムが複雑であることから，現時点では Matlab などの高水準言語を利用した実装が多く，計算量の評価や並列化に関する研究事例はほとんど報告されていない．本研究では，この手法の概要を紹介するとともに，OpenMP[4)] を用いた共有メモリアーキテクチャ上での並列実装を行なったので，その評価結果について報告する．

2. Jacobi-Davidson 法について

Jacobi-Davidson 法は，従来 Davidson 法[5)] として提案された手法に，Jacobi 法[7)] の考え方を用いて改良を加えたものである．

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 $\mathcal{K} = \mathrm{span}\{v_1, ..., v_k\}$ 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 $v_1, ..., v_k$ 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．

$$M_k t = r, \quad M_k = D_A - \theta_k I \tag{1}$$

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．$V_{k+1} = [v_1, ..., v_{k+1}]$ と置けば，新しい Ritz 対 (θ_{k+1}, u_{k+1}) は行列

$$H_{k+1} = V_{k+1}^* A V_{k+1} \tag{2}$$

の固有対として計算されることになるが，このことから，$M_k = I$ の場合に，Davidson 法は Lanczos/Arnoldi 法と同一となることが分かる．

ところが，ここで

$$M_k^{-1} \approx (A - \theta_k I)^{-1} \tag{3}$$

を残差ベクトル r に対する前処理行列と考えれば分かるように，この方法では θ_k に対応する近似固有ベクトル u_k の方向の成分を増幅させる結果となり，特に A が対角行列である場合には，新しい固有ベクトル成分を得ることができない．

したがって，ここでは u_k の直交補空間から更新のための成分を取り出すことを考える．以下では u_k は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

† 東京大学大学院理学系研究科情報科学専攻
Division of Information Science, School of 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 ϵ;
compute $u_1 = v_1 = v/ \parallel v \parallel_2$;
$w_1 = Av_1,\ \theta = h_{1,1} = w_1^* v_1,\ r = w_1 - \theta v_1$;
for $k = 2, ...$
  solve approximately a $z \perp u$ from
    $(I - uu^*)(A - \theta I)(I - uu^*)z = -r$;
  for $j = 1, ..., k-1$
    $z = z - (z^* v_j)v_j$;
  $v_k = z/ \parallel z \parallel_2,\ w_k = Av_k$;
  for $j = 1, ..., k$
    $h_{j,k} = w_k^* v_j$;
  compute the largest eigenpair (θ, y)
  of the matrix H_k with $\parallel y \parallel = 1$;
  compute the Ritz vector $u = Vy$
  and $\tilde{u} = Au = Wy$;
  $r = \tilde{u} - \theta u$;
  stop if $\parallel r \parallel_2 \leq \epsilon$;

図 1　JD 法による最大固有値の計算

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

$$A_P = (I - u_k u_k^*)A(I - u_k u_k^*) \tag{4}$$

で表されるが，これは

$$A = A_P + u_k u_k^* A + A u_k u_k^* - \theta_k u_k u_k^* \tag{5}$$

と書き直すことができる．修正ベクトル z は

$$A(z + u_k) = \lambda(z + u_k), \quad z \perp u_k \tag{6}$$

を満たすので，ここに (5) を代入すれば

$$(A_P - \lambda I)z = -r + (\lambda - \theta_k - u_k^* A z)u_k \tag{7}$$

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

$$(A_P - \lambda I)z = -r \tag{8}$$

の計算に帰着されることが分かる．実際には λ の値を知ることはできないが，(8) 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

$$(I - u_k u_k^*)(A - \theta_k I)(I - u_k u_k^*)z = -r \tag{9}$$

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．

3. 複数固有値

減次を用いて複数の固有値を求めることができる．行列 A の部分 Schur 形を

$$AQ_k = Q_k R_k, \quad k \ll n \tag{10}$$

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 $V_i = [v_1, ..., v_i]$ 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，$s_1, s_2, ...$ 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，記憶容量に応じてリスタートを行なうこともできる．

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

$$A[Q_k,\ q] = [Q_k,\ q] \begin{bmatrix} R_k & s \\ & \lambda \end{bmatrix}, \tag{11}$$

$$Q_k^* q = 0 \tag{12}$$

となるベクトル q を定めればよいが，このとき

$$(I - A_k Q_k^*)(A - \lambda I)(I - Q_k Q_k^*) = 0 \tag{13}$$

が成り立つ．q は

$$A_P = (I - A_k Q_k^*)A(I - Q_k Q_k^*) \tag{14}$$

の固有ベクトルであるので，A_P に対する Jacobi-Davidson 法により計算することができる．以上のアルゴリズムは JDQR 法と呼ばれている．

4. 内部固有値

実際には，上の方法では τ より大きな絶対値を持つ固有値 (内部固有値) を安定に計算することができない．これは，Ritz 値が主にスペクトルの外部にある固有値に関する情報を含んでいるためであるが，Jacobi-Davidson 法では，以下に述べる調和 Ritz 値を用いることで，この問題に対処している．

対称行列 A に関する固有値問題において，Lanczos 過程では，Krylov 部分空間 $V_{i+1} = [v_1, ..., v_{i+1}]$ を導く．Lanczos 過程から

$$AV_i = V_{i+1} T_{i+1,i} \tag{15}$$

と書くことができる．$(i+1) \times i$ 行列のうち，第 $i+1$ 行を除いた部分を $T_{i,i}$ とする．このとき，

$$V_i^* A A V_i = T_{i+1,i}^* V_{i+1}^* V_{i+1} T_{i+1,i} \tag{16}$$

$$= T_{i+1,i}^* T_{i+1,i} \equiv M_i \tag{17}$$

と置くと，M_i は対称正定値であることから，$M_i = U_i^* U_i$ と Cholesky 分解すれば，

$$U_i^{-*} V_i^* A A V_i U_i^{-1} = I \tag{18}$$

を得るので，$AV_i U_i^{-1}$ は直交行列であることが分かる．これは $AK_i(A, v_1)$ についての正規直交基底 を生成する．A^{-1} を $AK_i(A, v_1)$ に制限する射影は

$$U_i^{-*}(AV_i)^* A^{-1} A V_i U_i^{-1} \tag{19}$$

$$= U_i^{-*}(AV_i)^* V_i U_i^{-1} \tag{20}$$

$$= U_i^{-*} T_{i,i}^* U_i^{-1} \tag{21}$$

となるので，A^{-1} の固有ベクトルの近似は，

$$U_i^{-*} T_{i,i} U_i^{-1} t = \theta t \tag{22}$$

または

$$M_i^{-1} T_{i,i} s = \theta s \tag{23}$$

から求められる．実際には

$$T_{i,i}^{-1} M_i s = \theta^{-1} s \equiv \tilde{\theta} s \quad (24)$$

を解く必要があるが, これは

$$T_{i,i} + t_{i+1,i}^2 T_{i,i}^{-1} e_i e_i^* \quad (25)$$

から計算することができる. $\tilde{\theta}$ を A の調和 Ritz 値 (harmonic Ritz value) という.

5. 一般化固有値問題

5.1 アルゴリズム

一般化固有値問題

$$Ax - \lambda Bx = 0 \quad (26)$$

については, Petrov-Galerkin 法によって近似最大固有対を計算する.

探索空間 $\mathrm{span}\{v_1, ..., v_k\}$ 上の近似固有対 (θ, y) は, 試験部分空間 $\mathrm{span}\{w_1, ..., w_k\}$ に対して以下の関係を満たすものとする.

$$AV_k y - \theta B V_k y \perp \{w_1, ..., w_k\} \quad (27)$$

V_k, W_k を前節と同様に定義し, (θ_k, y_k) を k 次元一般化固有値問題

$$W_k^* A V_k y_k - \theta_k W_k^* B V_k y_k = 0 \quad (28)$$

の解とする. このとき, A の固有対は $(\theta_k,\ u_k \equiv V_k y_k)$ で近似される. u_k は Petrov ベクトル, θ_k は Petrov 値と呼ばれる. u が固有ベクトルに収束するにつれて $Au \approx \lambda Bu$ となることから, W_k の取り方としては, AV_k と BV_k の線形結合とするのがよい. 残差を $r = -(A - \theta_k B)u_k$ と定義すると, 探索空間は,

$$(I - q_k q_k^*)(A - \theta_k B)(I - u_k u_k^*) z = -r \quad (29)$$

の解 $z \perp u_k$ により拡張される. ここで Petrov ベクトル u_k, 試験ベクトル q_k とも正規化されているものとする. 一般化固有値問題に対する Jacobi-Davidson 法のアルゴリズムを図 2 にまとめる.

5.2 JDQZ

一般化固有値問題の場合にも, 減次による複数固有値の計算は以下のように行なうことができる.

ここでは, 問題 $(\beta A - \alpha B)q = 0$ に関して, 部分一般化 Shur 形 $AQ_{k-1} = Z_{k-1} S_{k-1}$, $BQ_{k-1} = Z_{k-1} T_k$ が得られていることを仮定する. (31) と同様に, 適当な q, z を用いて, これを

$$A[Q_{k-1},\ q] = [Z_{k-1},\ z] \begin{bmatrix} S_{k-1} & s \\ & \alpha \end{bmatrix}, \quad (30)$$

$$B[Z_{k-1},\ z] = [Q_{k-1},\ q] \begin{bmatrix} T_k & t \\ & \beta \end{bmatrix} \quad (31)$$

に拡張することを考える. この関係から, $u \equiv Z_{k-1}^*(\beta A - \alpha B)q$ とすれば, 一般化 Shur 対 $(q, \langle \alpha, \beta \rangle)$ について

$$Q_{k-1}^* q = 0, \quad (\beta A - \alpha B)q - Z_{k-1} u = 0 \quad (32)$$

を満たすことが分かるので,

$$Q_{k-1}^* q = 0, \quad (33)$$

$$(I - Z_{k-1} Z_{k-1}^*)(\beta A - \alpha B) q = 0 \quad (34)$$

が得られる. 以上から, $(q, \langle \alpha, \beta \rangle)$ について

$$Q_{k-1}^* q = 0, \quad (35)$$

choose starting vectors v and w;
set $V = [v],\ W = [w],\ k = 0$;
for $k = 0, ...$
  compute eigenpairs (y, θ) of the projected eigenproblem
    $W^* A V y - \theta W^* B V y = 0$
  of dimension $k + 1$;
  select a solution y and associated Petrov value θ;
  compute the Petrov vector $u = Vy$ and the residual $r = Au - \theta Bu$;
  stop if u and θ are accurate enough;
  select a w in $\mathrm{span}\{W\}$ and select $\tilde{u} \not\perp u$ and $\tilde{w} \not\perp w$;
  compute an approximate solution $z \perp \tilde{u}$ of the correction equation
    $\left(I - \frac{\tilde{w} w^*}{w^* \tilde{w}}\right)(A - \theta B) z = -r$;
  if k is too large:
    select an $l < k$, select $k \times l$ matrices R_V, R_W and compute
    $V = V R_V$, $W = W R_W$, $k = l$;
  select a $v \in \mathrm{span}\{V, z\} \setminus \mathrm{span}\{V\}$ and $V = [V\ v]$;
  select $\tilde{v} \notin \mathrm{span}\{W\}$ and $W = [W, \tilde{v}]$;

図 2 一般化固有値問題への JD 法の適用

$$(I - Z_{k-1} Z_{k-1}^*)(\beta A - \alpha B)(I - Q_{k-1} Q_{k-1}^*) q = 0 \quad (36)$$

が成り立つので, Shur 対 $(q, \langle \alpha, \beta \rangle)$ は, 減次された行列の対

$$((I - Z_{k-1} Z_{k-1}^*) A (I - Q_{k-1} Q_{k-1}^*),\ (I - Z_{k-1} Z_{k-1}^*) B (I - Q_{k-1} Q_{k-1}^*)) \quad (37)$$

の固有対になっていることが分かる. JDQZ では, 一般化固有値問題に対する Jacobi-Davidson 法を用いてこの固有値問題を解く.

5.3 調和 Petrov 値

τ に近い固有値を求める場合, 調和 Petrov 対 $(\theta, u) = V_k s$ は, W_k を $W_k \equiv (A - \tau B)V_k$ となるよう (または $(A - \tau B)V_k$ と直交するよう) 取れば, (28) の解から計算できる.

6. 実装と性能評価

Jacobi-Davidson 法の特性を評価するため, 本研究では上記アルゴリズムの実装, 及び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した. プログラムに関しては, Fokkema, van Gijzen[6] らによる JDQZ ルーチンをベースとした. 反復解法の記述には Templates[2], また線形演算には BLAS[8], LAPACK[1] を用いているため, 実装に当たってはこれらのライブラリを必要とす

る.

評価に使用した環境は, Intel Pentium III Xeon (550MHz, 512KB L2 cache) を 4 要素搭載した Dell Computer 社製対称型マルチプロセッサ PowerEdge 6300 (450NX chipset, 768MB main memory) 上の Solaris 7 で, コンパイラには Portland Group 社の PGI Parallel Fortran を用いた. PGI Fortran は OpenMP API をサポートしている.

まず, 固有値が既知である実対称行列を用いて計算量に関する評価を行なった. 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

$$A_2 = \begin{pmatrix} T_N & -I & & O \\ -I & \ddots & \ddots & \\ & \ddots & \ddots & -I \\ O & & -I & T_N \end{pmatrix}, \qquad (38)$$

$$T_N = \begin{pmatrix} 4 & -1 & & O \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ O & & -1 & 4 \end{pmatrix} \qquad (39)$$

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

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

表 2 は 256^2 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

$$\frac{4}{2 - 2\cos\left(\frac{\pi}{n+1}\right)} \approx \frac{4}{\pi^2} n^2, \qquad (40)$$

$$\frac{8}{4 - 4\cos\left(\frac{\pi}{N+1}\right)} \approx \frac{4}{\pi^2} N^2 \qquad (41)$$

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

Function	Calls	Cost(s)	Time(%)
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. 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

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
zxpay	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
...			

この結果から, 個々の関数についてみると, 計算時間の大部分を zgemv, zdotc, zaxpy などの BLAS ルーチンが占めていることが分かる. なお, zgemv は

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

zdotc は

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

また zaxpy, zxpay はそれぞれ

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

型の線形演算を行なうための関数である.

7. 並 列 化

このように, Jacobi-Davidson 法では Level 1, Level 2 BLAS 演算が全体の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 $\mathcal{O}(n)$ であると考えてよい.

BLAS レベルでの並列化に関しては PBLAS[3] などの関連研究があるが, PBLAS は MPI 上に実装されており, 共有メモリアーキテクチャを対象としたものではない. また, 対称型マルチプロセッサ上での BLAS 自動最適化[10] や, Level 3 BLAS の並列化についての研究も進められているが, 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.

しかしながら, 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 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, 上記の関数の最内側ループを 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スレッドと呼ばれる単一プロセスとして開始される. マスタス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, マスタースレッドは更新されたデータを用いて計算を続ける.

ここでは, zgemv, zdotc, zaxpy, zxpay の 4 関数について, それぞれの演算の最内側ループを静的に各スレッドへ割り当てた. また可能な限り陰的なバリア同期を行なわないよう指定した. なお, BLAS, LAPACK については PGI の最適化ライブラリを利用し, 必要な関数のみを並列化して置き換えた.

並列化後の実行結果を表 4に示す. 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. 全体の計算時間についても 50% 程度に短縮されており, 高い効果を示している.

表 4 $A_2, n = 256^2$ での並列実行結果

Function	Calls	Time(s)	Time(%)
jdqz	1	113	19.95
zgemv	9814	96	17.12
zaxpy	7818	54	9.69
zxpay	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
...			

図 3にスレッド数を変化させた時のスピードアップを示す. なお, 逐次での実行時間と並列版の 1 スレッドでの実行時簡の間に有意な差は見られず, OpenMP API 自体のオーバーヘッドは無視できる程度であった.

並列化した上記のルーチンについて, 次元を変えて実行時間をみたものが図 4 である. 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ことが分かる.

このように, BLAS レベルでの並列化は, 比較的細粒度であるため行列の次元が小さい場合は大きなスピードアップは得られないが, 大規模な問題に対しては有効な手法であるといえる. 今回は BLAS ライブラリのうち

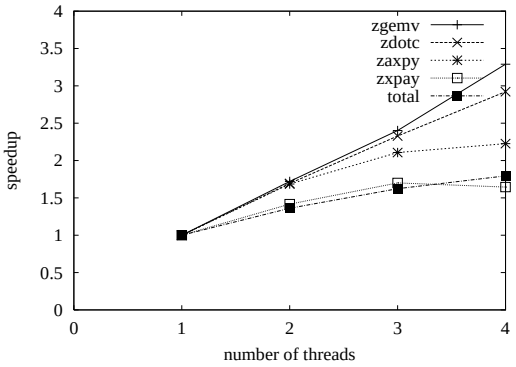

図 3 $A_2, n = 256^2$ での各ルーチンの速度向上率

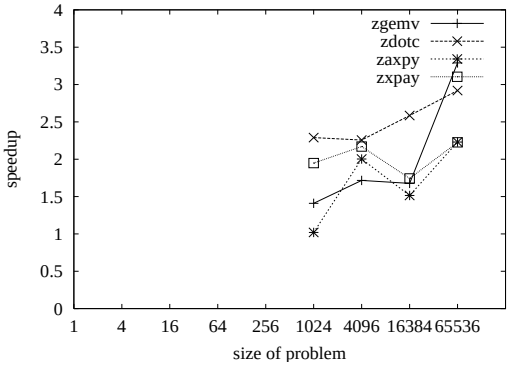

図 4 問題サイズと各ルーチンの速度向上率

計算量の多い部分についてのみ並列化を行なったが, このような修正によって容易に性能を向上させることができる点は特筆すべきである.

8. ま と め

本稿では, アルゴリズムを中心に Jacobi-Davidson 法とその共有メモリアーキテクチャ上での実装方式について検討した. また, OpenMP を用いたループ並列化により, Jacobi-Davidson 法において計算量の大部分を占める Level 1 BLAS 演算においても, 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く, 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. 今後は各解法の特性について, 最適な実装手法を中心に研究を進めていきたい.

参 考 文 献

1) E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied

Mathematics, third ed., 1999.

2) R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Society for Industrial and Applied Mathematics, 1994.

3) J. Choi, J. Dongarra, and D. Walker, *ParallelMatrix Transpose Algorithms on Distributed Concurrent Computers*, Tech. Rep. UT CS-93-215, LAPACK Working Note #65, University of Tennessee, 1993.

4) L. Dagum and R. Menon, *OpenMP: An Industry-Standard API for Shared-Memory Programming*, IEEE Computational Science & Engineering, 5 (1998).

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, *Ueber 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) G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.

10) B. C. Whaley and J. Dongarra, *Automatically Tuned Linear Algebra Software*, in Proceedings of SC98, 1998.