共有メモリ型並列計算機上の行列計算に対する並列化手法の性能評価

舘野 諭司[†1] 重原 孝臣[†2]
長谷川 秀彦[†3] 桧山 澄子[†4]

共有メモリ型並列計算機において線形計算プログラムを並列化する方法としては，全体の処理をOpenMP で並列化する方法，行列・ベクトル演算を扱う BLAS を並列化する方法の 2 つの方法がある．本研究では，HITACHI SR8000 1 ノードを共有メモリ型並列計算機と見なして，連立一次方程式の直接解法および固有値問題を例に，これら 2 種類の並列化手法を採用した線形演算ライブラリの性能を比較し，各並列化手法の効果と問題点を明らかにする．また，BLAS を並列化・チューニングする際，扱う問題によってどのレベルの BLAS を並列化・チューニングすれば性能改善効果が大きいかを述べる．実験の結果より，2 つの並列化手法の優劣は対象とアルゴリズムにより異なること，チューニングの効果が大きい BLAS レベルはアルゴリズムに依存することが明らかになった．

Performance Evaluation of Parallelizing Techniques for Matrix Computations on Shared Memory Parallel Computers

Satoshi Tateno,[†1] Takaomi Shigehara,[†2] Hidehiko Hasegawa[†3] and Sumiko Hiyama[†4]

There are two methods of parallelizing the programs for numerical linear algebra on shared memory parallel computers. One is the method which parallelizes a round sum of process in the main routine by using OpenMP. The other is the method where the BLAS routines for basic operations in linear algebra are highly parallelized. In this paper, we evaluate the performance of two linear algebra libraries with each parallelizing technique on a single node of HITACHI SR8000 as a shared memory parallel computer, and clarify the features of each parallelizing technique. For comparison, direct solution of linear systems and eigenvalue problems are considered. Furthermore, in order to make clear which of BLAS levels should be highly parallelized and tuned for the improvement of the performance for each problem and algorithm, we present an entire data for each problem which exhausts all the combinations of tuned/not-tuned BLAS for each BLAS level. The results for performance evaluation show that the parallelizing technique which should be adopted as well as the BLAS level which should be highly tuned is strongly dependent on the problem and the algorithm for it under consideration.

1. はじめに

一般に共有メモリ型並列計算機では，コンパイラによる自動並列化機能によってユーザプログラムの性能が容易にしかも飛躍的に向上するものと期待されがちである．しかし，プログラムに内在する並列度はアルゴリズムの性質に大きく依存し，単にコンパイラによる自動並列化だけで共有メモリ型並列計算機の性能を十分に引き出せるとは限らない[1)~4)]．また，仮にコンパイラの自動並列化機能が有効であったとしても，適切なチューニングを施さなければ計算機の能力を十分に引き出せるとはいえない．したがって，コンパイラの自動並列化機能を十分に活用しうるアルゴリズムに関する知見や，適切なチューニング技法の開発は，共有メモリ型並列計算機の有効活用のために必要不可欠である．本研究では，LAPACK（Linear Algebra Package）[5),6)] 互換のルーチンを含み，かつ並列化手

†1 東京コンピュータサービス株式会社
Tokyo Computer Service Co., Ltd.
†2 埼玉大学工学部情報システム工学科
Department of Information and Computer Sciences, Saitama University
†3 筑波大学図書館情報学系
Institute of Library and Information Science, University of Tsukuba
†4 埼玉大学理学部数学科
Department of Mathematics, Faculty of Science, Saitama University

法が異なるライブラリの性能を HITACHI SR8000 1 ノード上にて評価することにより，行列計算アルゴリズムにおいて有効となる並列化手法の特徴と適切なチューニングによる性能への影響について詳細な調査を行う．調査の対象となる問題は連立一次方程式の直接解法，対称行列/非対称行列の固有値問題である．これらに用いられるアルゴリズムの一部またはその変形版は一般固有値問題や特異値分解においても多用されるものであり，本研究の実験データや知見はこれらを含む線形問題全般に関わるものである．

ライブラリの使用は，ユーザが自ら処理ルーチンを作成する手間が省けるという点だけでなく，精度の保証・汎用性といった面でも優れている．しかし，ユーザの作成したプログラムと比較した場合，ライブラリ使用により性能を確実に改善できるという保証はない[7),8)]．また，同一機能のライブラリは多数あり，施されているチューニングの差異により各ライブラリの性能は異なる．特に並列計算機上では，並列化手法の違いでも性能が大きく変化すると考えられる．

共有メモリ型並列計算機における代表的な並列化手法としては，次の 2 種類の方法があげられる．1 つはアルゴリズム中で並列性のある部分を OpenMP[9)] 等の並列化指示子を用いて並列化する方法である．もう 1 つは行列・ベクトル演算に用いられる BLAS（Basic Linear Algebra Subprograms）[10)~14)] を並列化する方法である．それぞれの並列化手法を施しているライブラリとして，本研究では前者に The Numerical Algorithm Group 社の NAG Fortran SMP ライブラリ Release 2（以下，*NAG* と称す）[15)]，後者にベンダ提供の HITACHI LAPACK V01-03（以下，*v-LAPACK* と称す），および Netlib で配布されているチューニングの施されていないソースコードを HITACHI Optimizing FORTRAN90 コンパイラの自動並列化機能・最適化機能のみを用いてコンパイルした LAPACK Version 3.0（以下，*n-LAPACK* と称す）を利用する．

本研究では，並列化手法の違いによる性能の変化だけでなく，BLAS のチューニングによる性能の変化，および両者の関係もあわせて検討する．そのため，並列化手法の異なるライブラリの性能評価だけでなく，各ライブラリにおける BLAS 処理の実行時間の割合，BLAS レベルごとにチューニングの程度を変更した場合の性能変化に関しても調査を行い，各並列化手法の特徴と問題点およびチューニングの効果が大きい BLAS レベルを問題ごとに明らかにする．

本稿の構成は以下のとおりである．計算環境および各ライブラリの特徴を 2 章に示す．3 章では，まず実験方法を提示し，次に連立一次方程式，対称行列の固有値問題，非対称行列の固有値問題の順に実験データの提示・解析を行い，各問題における並列化手法の特徴・問題点を明らかにする．最後に 4 章でまとめを行う．

2. 計 算 環 境

2.1 計算機とコンパイラ

数値実験は HITACHI SR8000 1 ノード上で行う．SR8000 1 ノードは演算 CPU 8 台，制御 CPU 1 台から構成され，16GByte のメインメモリを共有するため，1 台の共有メモリ型並列計算機と見なすことができる．各 CPU は PowerPC ベースで，二次キャッシュを利用せずに高い性能を発揮する擬似ベクトル機構[16),17)] を備えている．また，各 CPU の一次キャッシュは命令 64 KByte，データ 128 KByte であるが，並列処理時には一次キャッシュも共有され，容量が CPU 台数倍になる．測定に用いた CPU 台数は 1 台および 8 台である．なお，SR8000 の性能は文献 17) で詳細に評価されている．

測定プログラムは FORTRAN77 の仕様に基づいて記述する．使用したコンパイラは HITACHI Optimizing FORTRAN90 コンパイラである．使用したコンパイラのバージョンとコンパイルオプションを**表 1** に示す．どのプログラムにおいても，コンパイラによる最適化は最高レベルを指定し（-O4, OPT(O(4))），コンパイル時のメモリ制限，変数のスコープは無効にする（-nolimit, -noscope）．OpenMP による並列化を行う *NAG* では OpenMP と自動並列化機能を有効にする（-omp -save -parallel）．*v-LAPACK*，*n-LAPACK* では，並列版（8 CPU）とシングル版（1 CPU）とで付加するオプションが異なる．並列版では，自動並列化機能を有効にし（MP(P(3)) -parallel），利用する CPU 台数を指定する（-procnum=8）．一方，シングル版では自動並列化機能を使用しない（-noparallel）．なお，*v-LAPACK*，*n-LAPACK* においては擬似ベクトル機構を有効にする（-pvfunc=3）．

2.2 比較対象のライブラリ

各ライブラリとも行列・ベクトル演算には BLAS を用いている．チューニングによる BLAS の性能変化を調査するため，Netlib[18)] で配布されている未チューニングの BLAS（以下，n-BLAS と称す）と，SR8000 の性能を十分に発揮できるよう，適切な段数でのループ展開や SR8000 特有の機能を使用するようにソースコードの一部を書き換える等のチューニングが施されたベンダ提供の BLAS（以下，v-BLAS と称す）の 2

表 1 コンパイラとコンパイルオプション
Table 1 Compiler version and compile options.

Compiler	Library		Compile Options
Optimizing FORTRAN90 Compiler V01-05-/A	*NAG*		-64 -nolimit -noscope -omp -save -O4 -parallel +SBTLB
	v-LAPACK *n-LAPACK*	8 CPU	-64 -nolimit -noscope -W0,'OPT(O(4)),MP(P(3))' -procnum=8 -pvfunc=3 -parallel +SBTLB
		1 CPU	-64 -nolimit -noscope -O4 -pvfunc=3 -noparallel +SBTLB

表 2 各ライブラリの並列化手法と使用する BLAS
Table 2 Parallelizing technique and BLAS.

Library		Parallelizing technique	BLAS
NAG	8 CPU	Parallelize outer loops by OpenMP	Nonparallelized
	1 CPU	—	
v-LAPACK	8 CPU	Parallelize core loops by compiler	Parallelized
n-LAPACK	1 CPU	—	Nonparallelized

種類の BLAS を使用する．

LAPACK（*v-LAPACK*，*n-LAPACK*）

並列化は核となる演算や補助的な演算を扱う BLAS ルーチンに対して重点的に行われている．*v-LAPACK* は，BLAS 以外の部分も並列化した並列版と，まったく並列化されていないシングル版の 2 種類に分かれている．*n-LAPACK* もコンパイルオプションの違いによって並列版とシングル版に分かれている．並列処理の場合は，並列版 *v-LAPACK/n-LAPACK* に並列版 BLAS を，逐次処理する場合は，シングル版 *v-LAPACK/n-LAPACK* に並列化されていないシングル版 BLAS をリンクする．

v-LAPACK と *n-LAPACK* の違いは SR8000 の特性に合わせたチューニングの有無である．*v-LAPACK* はベンダによる SR8000 向けのチューニングが施されている．*n-LAPACK* にはチューニングをいっさい施さず，コンパイラによる最適化のみを施す．

NAG Fortran SMP ライブラリ（*NAG*）

BLAS に対してではなく，ルーチン本体の並列性のある部分に対して OpenMP を用いて並列化されている．環境変数に CPU 台数を指定することで使用する CPU 台数を切り替えるため，同一の実行形式で並列処理が行える．また，ルーチン本体が並列化されているため，1 つの BLAS ルーチンが複数の CPU 上で同時に呼び出されることがあるので，並列化されていないシングル版 BLAS を使用し，二重並列化による性能劣化を避ける．

NAG, *v-LAPACK*, *n-LAPACK* における並列化手法と使用する BLAS をまとめた一覧表を**表 2** に示す．

3. 実験結果

本研究では連立一次方程式，対称/非対称行列の固有値問題に対して 2 種類の数値実験を行う．第 1 の実験（以下，実験 1 と称す）では，v-BLAS，n-BLAS を使用した場合の各ライブラリの性能を比較する．第 2 の実験（以下，実験 2 と称す）では，*n-LAPACK* に対して，各 BLAS レベルごとにリンクする BLAS を v-BLAS，n-BLAS に変更し，どのレベルのチューニング・並列化が効果的なのかを調べる．各実験ともルーチンの実行時間で性能を評価する．3.1 節以降に示す実験結果はすべて行列サイズが $n = 5000$ の場合のものであり，結果が正確であることは確認済みである．なお，$n = 3000$ の場合でも本研究の実験結果と同様の傾向がみられた．本研究の実験に用いたプログラムのソースコードは文献 19) にある．

表 3 〜 11 において，実行時間はすべて “分：秒” で表す．また，Speed-Up は 1 CPU の実行時間を 8 CPU の実行時間で割った値である．実験 2 の結果を示した表 5，8，11 では，使用した BLAS を

(Level 1, Level 2, Level 3)

の形式で表す．ここで，“v” は v-BLAS，“n” は n-BLAS を意味する．たとえば，(v,n,n) は Level 1 BLAS に v-BLAS，Level 2，3 BLAS に n-BLAS を用いたことを意味する．上段の値は実行時間，下段の値は全 BLAS レベルに n-BLAS を使用した場合を 1.00 とする実行時間の短縮率である．なお，(v,v,v)，(n,n,n) はそれぞれ表 3，6，9 の v-BLAS，n-BLAS を用いた場合の *n-LAPACK* と同一である．

表 4, 7, 10 は，n-BLAS を用いた逐次処理（1 CPU）において，全体の実行時間に対して各 BLAS レベルの実行時間が占める割合（以下，相対実行時間と称す）

表 3 LU 分解の実行時間（単位は分：秒）．行列サイズは $n = 5000$
Table 3 Real time (minute : second) for LU decomposition. Matrix size is $n = 5000$.

Library	v-BLAS			n-BLAS		
	1 CPU	8 CPU	Speed-Up	1 CPU	8 CPU	Speed-Up
NAG	01:18.40	00:12.01	6.53	04:04.51	00:38.42	6.36
v-LAPACK	01:28.33	00:17.82	4.96	04:16.12	00:44.02	5.77
n-LAPACK	01:54.44	00:22.35	5.12	04:09.71	00:44.06	5.66

表 4 n-BLAS 使用時における BLAS 処理の相対実行時間．行列サイズは $n = 5000$
Table 4 Relative time of each BLAS level in LU decomposition with n-BLAS. 1 CPU. Matrix size is $n = 5000$.

Library	Level 1 BLAS	Level 2 BLAS	Level 3 BLAS	Others
NAG	0.06%	0.04%	96.58%	3.32%
v-LAPACK	0.28%	3.48%	92.09%	4.15%
n-LAPACK	0.14%	0.93%	94.89%	4.04%

を百分率で示した表である．これらの相対実行時間はプロファイラにより得た個々のルーチンの実行時間を BLAS レベルごとに分けて集計して得る．また，表中の Others の欄はライブラリルーチンを含む BLAS 以外の処理に要する相対実行時間を表している．

3.1 連立一次方程式

本節では，部分軸選択をともなう LU 分解による連立一次方程式の直接解法に対する結果について議論する．係数行列は区間 $[0, 1)$ の一様乱数を成分に持つ密行列とする．LAPACK ルーチンは dgetrf，dgetrs を用いる．まず dgetrf を用いて行列を LU 分解し，dgetrs により連立一次方程式の解を求める．しかし，LU 分解後の求解部分の演算量 $[O(n^2)]$ は分解の演算量 $[O(n^3)]$ よりはるかに少なく，dgetrs に要する時間はどの実験結果でも 0.1〜0.5 秒程度と dgetrf の結果よりもはるかに短かった．そのため，本章では dgetrf，すなわち LU 分解の実行時間について議論する．

表 3 に実験 1 の結果を示す．v-BLAS を用いた場合は，1 CPU，8 CPU ともに *NAG*(1 CPU 01:18.40, 8 CPU 00:12.01)，*v-LAPACK*(1 CPU 01:28.33, 8 CPU 00:17.82)，*n-LAPACK*(1 CPU 01:54.44, 8 CPU 00:22.35) の順に高速である．8 CPU の場合は，NAG(00:12.01) が Vendor(00:17.82) より 32%程高速であることから，LU 分解ルーチン本体を並列化する *NAG* の並列化手法は，BLAS を並列化する *v-LAPACK*，*n-LAPACK* の並列化手法よりも優れているように見える．この理由としては，次の 2 点が主な要因であると考えられる．

第 1 の要因は，ブロック化 LU 分解[5),20)] を並列化していることである．ブロック化 LU 分解は，対象となる行列をいくつかのブロックに分割し，要素単位ではなくブロック単位で LU 分解を行う方法である．ブロック化 LU 分解では，核演算が行列–行列演算となるため，通常の LU 分解よりも処理の並列性が高くなる[20)]．*NAG* では，ブロック化 LU 分解を OpenMP で並列化しているため，複数のブロックに対する処理を並列に実行できる．一方 *v-LAPACK*，*n-LAPACK* は，ルーチン本体のブロックを逐次的に処理している．*NAG* の実行時間が *v-LAPACK*，*n-LAPACK* よりも短く，しかも Speed-Up が 6.53 と *v-LAPACK*(4.96)，*n-LAPACK*(5.12) よりも高いことは，ブロックを並列に処理する方法の処理効率が，BLAS を並列化する方法よりも良いことを示している．

第 2 の要因は，シングル版 v-BLAS が十分にチューニングされていることである．n-BLAS を使用した場合の Speed-Up は，*NAG* は 6.36 と低くなり，*v-LAPACK* は 5.77 と高くなる．*v-LAPACK* での 1 CPU の性能低下が 8 CPU よりも大きいことから，*v-LAPACK* の Speed-Up の向上は，シングル版 BLAS のチューニングの効果が大きいことを意味する．*NAG* はブロックの並列処理にシングル版 BLAS を用いているため，シングル版 BLAS のチューニングは *NAG* 8 CPU の性能にも影響を及ぼす．そのため，*v-LAPACK* 8 CPU に対する *NAG* 8 CPU の高速化の割合は，チューニングされていない n-BLAS の場合は 12%程度にとどまり，チューニングされている v-BLAS の場合には約 32%に達する．これらのことから，*NAG* の並列化手法は，シングル版 BLAS が十分チューニングされている場合に高い効果を発揮するといえる．

表 4 に示す BLAS の相対実行時間を見ると，すべてのライブラリにおいて，Level 3 BLAS の相対実行時間が 92%以上と長いことが分かる．これは，ブロック化 LU 分解の核演算が行列–行列演算となり，Level 1,

表 5 レベルごとに BLAS を変えた場合の *n-LAPACK* の実行時間（単位は分：秒）．行列サイズは $n = 5000$
Table 5 Real time of *n-LAPACK* with v-BLAS/n-BLAS for each BLAS level. Matrix size is $n = 5000$.

Routine		BLAS routines of each level : (Level 1, Level 2, Level 3) v = "v-BLAS", n = "n-BLAS"							
		(n, n, n)	(v, n, n)	(n, v, n)	(n, n, v)	(v, v, n)	(v, n, v)	(n, v, v)	(v, v, v)
dgetrf	8 CPU	00:44.06	00:44.07	00:44.09	00:22.34	00:44.08	00:22.34	00:22.35	00:22.35
		1.00	1.00	1.00	1.97	1.00	1.97	1.97	1.97
	1 CPU	04:09.68	04:09.73	04:09.82	01:54.22	04:09.87	01:54.35	01:54.42	01:54.45
		1.00	1.00	1.00	2.19	1.00	2.18	2.18	2.18
	Speed-Up	5.67	5.67	5.67	5.11	5.67	5.12	5.12	5.12

表 6 対称固有値問題の実行時間（単位は分：秒）．行列サイズは $n = 5000$
Table 6 Real time (minute : second) for symmetric eigenvalue/eigenvector problem. Matrix size is $n = 5000$.

Library		v-BLAS			n-BLAS		
		1 CPU	8 CPU	Speed-Up	1 CPU	8 CPU	Speed-Up
NAG	dsyevd	09:03.68	02:19.01	3.91	25:05.55	08:48.34	2.84
v-LAPACK	dsyevr	07:32.82	01:24.37	5.37	25:26.40	08:08.45	3.12
	dsyevd	08:29.55	01:20.83	6.30	29:34.69	08:31.81	3.47
	dsyev	23:11.17	04:41.39	4.94	37:17.13	10:47.42	3.46
n-LAPACK	dsyevr	09:12.92	01:39.86	5.54	25:05.00	07:44.65	3.24
	dsyevd	10:05.30	01:36.21	6.29	29:16.70	08:07.72	3.60
	dsyev	23:02.82	04:47.30	4.81	36:01.30	10:20.92	3.48

2 BLAS よりも Level 3 BLAS を使用する回数が多くなるためである．また，Level 3 BLAS の演算量は $O(n^3)$ と多いため，1 回の演算に要する時間が Level 1, 2 BLAS よりも長いことも相対実行時間の長い理由である．このことをふまえ，**表 5** に示す実験 2 の結果を見ると，相対実行時間の短い Level 1，2 BLAS に v-BLAS を使用しても性能は向上しないが，相対実行時間の長い Level 3 BLAS に v-BLAS を使用する場合には，1 CPU で 2.19 倍，8 CPU で 1.97 倍に性能向上している．これらのことから，ブロック化 LU 分解の性能は Level 3 BLAS の性能に大きく影響されることが分かる．

以上のことより，*NAG* で採用しているルーチン本体を並列化する方法では，全体のアルゴリズムに高い並列性があり，なおかつシングル版 BLAS が十分チューニングされていることが，並列処理時に高い性能を発揮するための条件といえる．特にブロック化 LU 分解の場合には，処理の大部分を占める Level 3 BLAS のチューニングが非常に重要であり，シングル版 BLAS における Level 3 BLAS のチューニングが不十分な場合には，BLAS を並列化する *v-LAPACK*, *n-LAPACK* の並列化手法がルーチン本体を並列化する *NAG* の並列化手法よりも性能が良くなる可能性が高くなる．

3.2 対称固有値問題

本節では，実対称行列の全固有値・固有ベクトルの計算に対する結果について議論する．使用する行列は (i, j) 成分が

$$a_{ij} = n - \max(i, j) + 1, \quad i, j = 1, \cdots, n$$

で表される n 次フランク行列である．LAPACK ルーチンは，Relatively Robust Representation [21] による dsyevr，分割統治法による dsyevd，QR 法による dsyev を用いる．*NAG* では dsyevr，dsyev に相当する LAPACK 互換ルーチンが含まれていないため，dsyevd と同等のルーチンを用いる．どのルーチンにおいても，1) 対称行列の 3 重対角化，2) 3 重対角行列の固有値・固有ベクトルの計算，3) 固有ベクトルの復元（3 重対角行列の固有ベクトルを元の固有ベクトルへ変換）の 3 段階の処理を経て固有値・固有ベクトルが計算される．

表 6 に実験 1 の結果を示す．*v-LAPACK* の結果を見ると，1 CPU では，v-BLAS，n-BLAS ともに dsyevr の性能が高く，v-BLAS を使用した場合は 07:32.82 となる．8 CPU では，v-BLAS を用いた場合には dsyevd(01:20.83)，n-BLAS を用いた場合には dsyevr(08:08.45) の性能が高い．また，Speed-Up が最も高くなるのは dsyevd であり，v-BLAS を使用した場合には 6.30 となる．演算量が多く並列性に欠ける QR 法を採用する dsyev は，v-BLAS を用いた場

表 7 n-BLAS 使用時における BLAS 処理の相対実行時間．行列サイズは $n = 5000$
Table 7 Relative time of each BLAS level in symmetric eigenvalue/eigenvector problem with n-BLAS. 1 CPU. Matrix size is $n = 5000$.

Library		Level 1 BLAS	Level 2 BLAS	Level 3 BLAS	Others
NAG	dsyevd	0.17%	0.29%	84.99%	14.55%
v-LAPACK	dsyevr	0.08%	17.91%	79.28%	2.73%
	dsyevd	0.15%	15.38%	82.67%	1.80%
	dsyev	0.05%	12.69%	40.62%	46.64%
n-LAPACK	dsyevr	0.08%	16.57%	80.59%	2.76%
	dsyevd	0.15%	14.18%	83.60%	2.07%
	dsyev	0.05%	11.66%	42.81%	45.48%

合，1 CPU が 23:11.17，8 CPU が 04:41.39 と，使用する CPU 台数を問わず他のルーチンより倍以上遅くなる．これらの傾向は *n-LAPACK* でも同じである．以上のことから，使用する固有値計算アルゴリズムは性能を左右する最大の要因であるといえる．

8 CPU の結果に注目すると，v-BLAS を用いた *NAG* が 02:19.01 と，同等のアルゴリズムを採用した dsyevd の *n-LAPACK*(01:36.21)，*v-LAPACK*(01:20.83) に劣ることが分かる．n-BLAS を用いた場合も同じ傾向を示しており，dsyevd の *n-LAPACK*(08:07.72)，*v-LAPACK*(08:31.81)，*NAG*(08:48.34) の順に実行時間が短い．コンパイラによる最適化のみを施した *n-LAPACK* が，高度にチューニングされた *NAG* を上回る性能を発揮したことは，並列化手法の違いが性能に大きな影響を与えているといえる．v-BLAS を用いた場合の Speed-Up を比較してみると，*v-LAPACK*，*n-LAPACK* ともに 4.80〜6.30 と *NAG*(3.91) よりも高い値を得ている．このことから，対称固有値問題では，シングル版 BLAS を使用するため BLAS の処理速度が変化しない *NAG* の並列化手法よりも，並列化によって BLAS を高速化する *v-LAPACK*，*n-LAPACK* の並列化手法のほうが効率が良いといえる．

n-BLAS を用いる場合，最速となるルーチンは 1 CPU，8 CPU ともに dsyevr の *n-LAPACK*(1 CPU 25:05.00，8 CPU 07:44.65) である．しかし，これらの実行時間は v-BLAS を用いた場合で最も遅い dsyev の *n-LAPACK*(1 CPU 23:02.82，8 CPU 04:47.30) よりもさらに長い．また，Speed-Up が 2.84〜3.60 と，v-BLAS を用いた場合（3.91〜6.30) よりも低下している．これらのことは，v-BLAS のチューニング・並列化の効果が大きいことを示している．

表 7 に示す各ライブラリにおける BLAS の相対実行時間を見ると，dsyev 以外のルーチンでは，Level 3 BLAS の相対実行時間が 80%から 85%と長い．また，*NAG* 以外のルーチンでは，Level 2 BLAS の相対実行時間が 11%から 15%と連立一次方程式の場合よりも長くなっている．これらのことから，対称固有値問題においては Level 3 BLAS だけでなく，Level 2 BLAS の高速化も性能向上の重要な要因であるといえる．dsyev では，BLAS 以外のルーチンの相対実行時間が 45%前後と長い．これは，BLAS を利用できない逐次性の高い処理がルーチン内部に含まれているためであり，BLAS の高速化だけでは劇的な性能改善が望めないことを意味する．

表 8 は，*n-LAPACK* の dsyevr，dsyevd，dsyev において，処理を 1) 行列の 3 重対角化，2) 3 重対角行列の固有値・固有ベクトルの計算，3) 固有ベクトルの復元の 3 つの段階に分け，それぞれの処理ごとに実験 2 の結果を示したものである．なお，行列の 3 重対角化はどのルーチンでも同一のルーチンを用いて処理しているため，実験結果は同じになる．

3 重対角化の段階において Level 2 BLAS に v-BLAS を使用した場合，1 CPU では 1.45 倍，8 CPU では 4.11 倍性能が改善される．特に，8 CPU 時に Level 2，3 BLAS に v-BLAS を使用した場合，性能は約 8 倍と大幅に向上している．これは，3 重対角化に用いるハウスホルダー変換で Level 2 BLAS が多用されるためである．また，dsyevr，dsyev の 3 重対角行列の固有値・固有ベクトルの計算では，v-BLAS を使用しても実行時間に変化がみられない．このことは，分割統治法のルーチン dsyevd 以外では，BLAS をほとんど利用していないことを意味している．なお，固有ベクトルの復元は 3 重対角化の際に蓄積した変換行列と 3 重対角行列の固有ベクトルを格納した行列の積であり，Level 3 BLAS が処理の大半を占めている．そのため，Level 3 BLAS に v-BLAS を用いると固有ベクトル復元部分の性能が約 3 倍に向上する．

以上のことより，対称固有値問題では，演算量が少なく並列性の高いアルゴリズムの選択と個々の演算の高速化が重要であり，BLAS を並列化する手法が有効であることが分かった．特に，Level 2，3 BLAS のチューニング・並列化は行列の 3 重対角化，固有ベク

表 8 レベルごとに BLAS を変えた場合の *n-LAPACK* の実行時間（単位は分：秒）．行列サイズは $n = 5000$
Table 8 Real time of *n-LAPACK* with v-BLAS/n-BLAS for each BLAS level. Matrix size is $n = 5000$.

Routine		BLAS routines of each level : (Level 1, Level 2, Level 3) v="v-BLAS", n="n-BLAS"							
		(n,n,n)	(v,n,n)	(n,v,n)	(n,n,v)	(v,v,n)	(v,n,v)	(n,v,v)	(v,v,v)
1) Reduce symmetric matrix to tridiagonal form									
dsyevr	8 CPU	04:58.63	04:57.77	01:12.67	04:23.69	01:11.89	04:23.37	00:37.00	00:36.38
		1.00	1.00	4.11	1.13	4.15	1.13	8.07	8.21
	1 CPU	07:33.67	07:33.16	05:12.97	05:51.59	05:12.96	05:51.63	03:30.90	03:30.95
		1.00	1.00	1.45	1.29	1.45	1.29	2.15	2.15
	Speed-Up	1.52	1.52	4.31	1.33	4.35	1.34	5.70	5.79
dsyevd	8 CPU	04:55.78	04:55.97	01:12.00	04:21.30	01:11.26	04:21.04	00:36.73	00:35.99
		1.00	1.00	4.11	1.13	4.15	1.13	8.05	8.22
	1 CPU	07:33.00	07:32.48	05:11.42	05:51.35	05:11.59	05:50.99	03:29.85	03:29.59
		1.00	1.00	1.45	1.29	1.45	1.29	2.15	2.16
	Speed-Up	1.53	1.53	4.33	1.34	4.37	1.34	5.71	5.82
dsyev	8 CPU	04:57.20	04:56.80	01:12.37	04:22.34	01:11.73	04:21.67	00:36.72	00:36.10
		1.00	1.00	4.11	1.13	4.14	1.14	8.09	8.23
	1 CPU	07:33.38	07:33.09	05:12.37	05:52.00	05:12.15	05:51.51	03:30.19	03:30.57
		1.00	1.00	1.45	1.29	1.45	1.29	2.15	2.15
	Speed-Up	1.53	1.53	4.32	1.34	4.35	1.34	5.72	5.83
2) Calculate eigenvalues and eigenvectors of tridiagonal matrix									
dsyevr	8 CPU	00:19.57	00:19.56	00:19.56	00:19.57	00:19.58	00:19.52	00:19.52	00:19.57
		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	1 CPU	00:20.35	00:20.33	00:20.32	00:20.32	00:20.33	00:20.36	00:20.34	00:20.32
		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Speed-Up	1.04	1.04	1.04	1.04	1.04	1.04	1.04	1.04
dsyevd	8 CPU	00:43.59	00:42.72	00:43.61	00:17.23	00:42.72	00:16.38	00:17.23	00:16.36
		1.00	1.02	1.00	2.53	1.02	2.66	2.53	2.66
	1 CPU	04:02.55	04:02.27	04:02.59	01:13.24	04:02.29	01:12.95	01:13.42	01:12.93
		1.00	1.00	1.00	3.31	1.00	3.32	3.30	3.33
	Speed-Up	5.56	5.67	5.56	4.25	5.67	4.45	4.26	4.46
dsyev	8 CPU	03:42.27	03:41.91	03:41.37	03:40.89	03:41.10	03:41.41	03:41.67	03:41.79
		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	1 CPU	16:00.38	15:58.66	15:57.11	15:59.93	15:59.53	15:59.34	15:59.03	15:58.38
		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Speed-Up	4.32	4.33	4.32	4.35	4.34	4.33	4.33	4.32
3) Transform eigenvectors of tridiagonal matrix into eigenvectors of original matrix									
dsyevr	8 CPU	02:27.14	02:27.09	02:25.21	00:45.75	02:25.13	00:45.71	00:43.63	00:43.67
		1.00	1.00	1.01	3.22	1.01	3.22	3.37	3.37
	1 CPU	17:08.03	17:08.46	17:07.90	05:21.84	17:06.89	05:21.67	05:20.13	05:20.13
		1.00	1.00	1.00	3.19	1.00	3.20	3.21	3.21
	Speed-Up	6.97	6.97	7.07	7.03	7.08	7.04	7.34	7.34
dsyevd	8 CPU	02:27.61	02:27.69	02:25.65	00:45.63	02:25.60	00:45.68	00:43.58	00:43.66
		1.00	1.00	1.01	3.23	1.01	3.23	3.39	3.38
	1 CPU	17:35.92	17:35.87	17:34.61	05:22.40	17:34.94	05:22.16	05:20.86	05:20.63
		1.00	1.00	1.00	3.28	1.00	3.28	3.29	3.29
	Speed-Up	7.15	7.15	7.24	7.07	7.24	7.05	7.36	7.34
dsyev	8 CPU	01:41.48	01:41.47	01:37.57	00:36.05	01:37.58	00:34.26	00:30.31	00:30.33
		1.00	1.00	1.04	2.82	1.04	2.96	3.35	3.35
	1 CPU	12:23.21	12:23.01	12:20.00	03:38.04	12:19.92	03:36.98	03:33.98	03:34.01
		1.00	1.00	1.00	3.41	1.00	3.43	3.47	3.47
	Speed-Up	7.32	7.32	7.58	6.04	7.58	6.33	7.06	7.06

トルの復元といった各アルゴリズムで共通となる処理を高速化する際に有効であり，性能改善に欠かせない要因である．

3.3 非対称固有値問題

本節では，非対称行列の固有値問題に対する結果について議論する．固有値はすべて求めるが，固有ベクトルは求めない．実験には

表 9 非対称固有値問題の実行時間（単位は分：秒）．行列サイズは $n = 5000$
Table 9 Real time (minute : second) for non-symmetric eigenvalue problem. Matrix size is $n = 5000$.

Library		v-BLAS			n-BLAS		
		1 CPU	8 CPU	Speed-Up	1 CPU	8 CPU	Speed-Up
NAG	dgehrd	15:33.48	15:35.91	1.00	23:02.10	23:02.07	1.00
	dhseqr	09:01.90	09:03.45	1.00	09:02.62	09:02.61	1.00
	Total	24:35.38	24:39.36	1.00	32:04.72	32:04.68	1.00
v-LAPACK	dgehrd	15:32.25	02:15.39	6.89	23:08.03	10:54.48	2.12
	dhseqr	44:27.74	11:01.06	4.03	45:03.90	10:18.24	4.37
	Total	59:59.99	13:16.45	4.52	68:12.22	21:12.72	3.21
n-LAPACK	dgehrd	15:32.51	02:15.27	6.89	23:04.26	10:54.34	2.12
	dhseqr	31:25.49	07:54.01	3.98	32:51.49	07:13.11	4.55
	Total	46:58.00	10:09.28	4.63	55:55.75	18:07.45	3.09

表 10 n-BLAS 使用時における BLAS 処理の相対実行時間．行列サイズは $n = 5000$
Table 10 Relative time of each BLAS level in non-symmetric eigenvalue problem with n-BLAS. 1 CPU. Matrix size is $n = 5000$.

Library	Level 1 BLAS	Level 2 BLAS	Level 3 BLAS	Others
NAG	0.89%	71.18%	0.00%	27.93%
v-LAPACK	1.04%	33.49%	0.00%	65.47%
n-LAPACK	1.23%	40.83%	0.00%	57.94%

$a_{ij} = \mathrm{mod}(n + j - i, n) + 1, \quad i, j = 1, \cdots, n$
を (i, j) 成分とする巡回行列を用いた．LAPACK ルーチン dgehrd により行列をヘッセンベルグ行列に変換し，dhseqr を用いて QR 法で全固有値を求める．これらのルーチンを利用して固有ベクトルを求めることも可能だが，今回は固有ベクトルを計算しない．なお，*NAG* では非対称行列の固有値問題に関するルーチンは並列化されていない[15]．

表 9 に実験 1 の結果を示す．表 9 中，Total の行は行列の初期化を除くプログラム全体の実行時間を表す．*NAG* は 8 CPU でも 1 CPU とまったく同じ処理を行うため，Speed-Up が 1.00 となる．Total は，v-BLAS を用いた場合，CPU 台数を問わずチューニングされていない *n-LAPACK*(1 CPU 46:58.00，8 CPU 10:09.28) がチューニングされている *v-LAPACK*(1 CPU 46:58.00，8 CPU 10:09.28) を上回ることが分かる．この傾向は n-BLAS を用いた場合でも同じであるため，非対称固有値問題の結果は，チューニングされたライブラリの性能がユーザの作成したプログラムに劣る一例であるといえる[7),8)]．

v-BLAS を用いた場合，*NAG* を除く各ライブラリの dgehrd の実行時間は 1 CPU で 15:32，8 CPU で 02:15 と同程度であり，性能に大きな影響を与えるルーチンは dhseqr であることが分かる．dhseqr は使用する BLAS のチューニングに関係なくほぼ一定の性能になる．また，1 CPU では *NAG* の dhseqr 相当のルーチンが 09:01 前後とはるかに高速であり，次いで *n-LAPACK*(31:25〜32:51)，*v-LAPACK*(44:27〜45:03) の順に性能が良い．8 CPU でも *n-LAPACK*(07:13〜07:54)，*v-LAPACK*(10:18〜11:01) の順序は変化しない．これらのことから，*v-LAPACK* における dhseqr のチューニングは不適切であることが分かる．性能を改善するには，固有値の計算ルーチンである dhseqr の徹底的なチューニング，または代替となるルーチンの開発が必要といえる．

n-BLAS を用いた場合には，dgehrd の Speed-Up が 6.89 から 2.12 に著しく低下する．Speed-Up 低下の原因を**表 10**，**表 11** より追求する．まず，表 10 より，非対称固有値問題では Level 3 BLAS がまったく使用されていないことが分かる（Level 3 BLAS を使用する固有ベクトルの復元は今回の非対称固有値問題の実験には含まれていない）．一方，Level 2 BLAS の相対実行時間は *v-LAPACK* で 33.49%，*n-LAPACK* で 40.83%と長い．このため，n-BLAS 使用時の dgehrd の Speed-Up 低下の主な原因はヘッセンベルグ行列への変換時に多用する Level 2 BLAS のチューニングにあると考えられる．事実，表 11 より，dgehrd の性能は Level 2 BLAS に v-BLAS を用いる場合のみ改善されており，しかも 1 CPU での性能改善率 1.49 倍に比べ，8 CPU では 4.80 倍と大幅に性能改善率が高くなっている．このことから，非対称固有値問題では，Level 2 BLAS のチューニングが dgehrd の性能を左右し，全体の性能に影響を与える最大の要因であるといえる．

表 11 レベルごとに BLAS を変えた場合の *n-LAPACK* の実行時間（単位は分：秒）．行列サイズは $n = 5000$
Table 11 Real time of *n-LAPACK* with v-BLAS/n-BLAS for each BLAS level. Matrix size is $n = 5000$.

Routine		BLAS routines of each level : (Level 1, Level 2, Level 3) v = "v-BLAS" , n = "n-BLAS"							
		(n, n, n)	(v, n, n)	(n, v, n)	(n, n, v)	(v, v, n)	(v, n, v)	(n, v, v)	(v, v, v)
dgehrd	8 CPU	10:53.48	10:52.95	02:16.14	10:53.33	02:15.18	10:53.46	02:15.68	02:15.19
		1.00	1.00	4.80	1.00	4.83	1.00	4.81	4.83
	1 CPU	23:05.09	23:05.05	15:02.16	23:02.56	15:32.47	23:02.81	15:30.92	15:30.44
		1.00	1.00	1.49	1.00	1.49	1.00	1.49	1.49
	Speed-Up	2.11	2.12	6.63	2.11	6.89	2.11	6.86	6.88
dhseqr	8 CPU	07:12.18	07:51.69	06:42.72	07:14.56	07:48.96	07:54.75	06:47.09	07:49.58
		1.00	0.92	1.07	0.99	0.92	0.91	1.06	0.92
	1 CPU	32:47.70	33:55.02	31:41.92	32:45.63	31:16.06	33:52.55	31:41.62	31:35.42
		1.00	0.98	1.03	1.00	1.05	0.98	1.03	1.04
	Speed-Up	4.55	4.31	4.72	4.52	4.00	4.28	4.67	4.04

以上のことから，現時点での非対称固有値問題の解法アルゴリズムでは，ヘッセンベルグ行列に変換する dgehrd で多用される Level 2 BLAS のチューニングと，QR 法で固有値を計算する dhseqr のチューニングによって性能が決定してしまうといえる．また，これらのアルゴリズムには，並列性が乏しいためルーチン本体を並列化することが難しい，BLAS のチューニングの効果が表れない等，性能改善の障壁となる課題が多く含まれている．そのため，非対称固有値問題の性能改善には，既存アルゴリズムの大幅な改良，もしくは新しいアルゴリズムの開発が不可欠である．

4. ま と め

本稿では SR8000 1 ノードを 8 台の CPU を持つ共有メモリ型並列計算機と見なし，BLAS の並列化に頼らずライブラリルーチン本体を OpenMP を用いて並列化する NAG Fortran SMP ライブラリ，BLAS ルーチンを重点的に並列化するベンダ提供の LAPACK，Netlib 上のソースコードをコンパイラの自動並列化機能・最適化機能を用いて作成した LAPACK について性能比較を行い，各並列化手法の特徴と問題点を明らかにした．

数値実験の結果より，LU 分解を用いた連立一次方程式の解法では，アルゴリズムの並列性が高いため，ルーチン本体を OpenMP により並列化する手法の効率が良く，シングル版 BLAS がチューニングされている場合には BLAS を並列化する手法よりも高い性能が得られること，ブロック化 LU 分解を使用しているため，Level 3 BLAS のチューニングが有効となることが明らかとなった．対称行列の固有値問題の結果からは，効率の良いアルゴリズムの選択と BLAS の高速化が性能向上の要であり，Level 2，3 BLAS 両方のチューニングが重要であることが明らかとなった．また，非対称行列の固有値問題の結果からは，並列性が乏しいアルゴリズムのためルーチン本体の並列化が困難であり，現状では BLAS を並列化する手法しか適用できないこと，Level 2 以外の BLAS をチューニングしても効果がないことが分かった．これらの結果より，ルーチン本体を OpenMP 等によって並列化する手法と BLAS を並列化する手法の処理効率は，問題やルーチンのアルゴリズムに依存すると考えられる．また，前者の場合にはアルゴリズムの並列性とシングル版 BLAS の性能，後者の場合には並列版 BLAS の性能により全体の性能が決まるといえる．加えて，多くの線形計算問題で用いられる Level 2，3 BLAS の高速化，現状のアルゴリズムより並列性の高いアルゴリズムの開発も，プログラムの性能を向上させるうえで重要といえる．

本研究で得た結論をふまえると，今後の共有メモリ型並列計算機用の並列ライブラリに期待されることは，本研究で調査した 2 種類の並列化手法をともに備え，状況に応じて並列化手法を適切に使い分けることができるような機能を備えていることであろう．このようなライブラリでは，対象とする計算機の特性や扱う問題の性質を考慮に入れなければならないため，ライブラリの作成・チューニングが従来のライブラリよりも困難になると考えられる．しかし，ATLAS [22] のように最適なパラメータを自動的に決定する自動チューニングの技法を応用すれば，柔軟性の高い並列化手法を採用したライブラリは近いうちに実現できると考えられる．また，逐次的な問題に対しても効率良く処理できる工夫，並列性が高く効率の良いアルゴリズムの開発は，既存の並列版/シングル版ライブラリの性能を改善し，新しい並列化手法を備えたライブラリを完成

させるためにも継続して研究する必要がある．

謝辞 本研究の遂行にあたり，SR8000 上で最新版の LAPACK を利用できるようにしていただき，OpenMP 使用の際に数々の情報を提供していただいた（株）日立製作所 後保範氏，吉村卓弘氏，永山崇氏，NAG Fortran SMP ライブラリの並列化について様々なコメントをいただいた The Numerical Algorithms Group Ltd.，Dr. Stefano Salvini 氏，ならびに NAG Fortran SMP ライブラリの使用を快諾していただいた日本ニューメリカルアルゴリズムグループ株式会社に感謝の意を表する．

参 考 文 献

1) 津田孝雄：並列数値処理の現状，応用数理，Vol.11, No.1, pp.14–32 (2001).
2) Pieper, K.L.: *Parallelizing Compilers: Implementation and Effectiveness*, Ph.D. Thesis, Stanford University (1993). http://suif.stanford.edu/
3) 桧山澄子，長谷川秀彦：並列計算機 CS6400 における数値計算ライブラリの現状と問題点，技術研究報告（東京大学地震研究所），No.1, pp.56–71 (1996).
4) 桧山澄子，鷹野 澄，山中佳子：Fortran コンパイラの自動並列化性能評価，技術研究報告（東京大学地震研究所），No.2, pp.1–7 (1998).
5) Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J.J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D.: *LAPACK Users' Guide (3rd Ed.)*, SIAM (2000).
6) 山本喜一，榊原 進，野寺隆志，長谷川秀彦：これだけは知っておきたい数学ツール，共立出版 (1999).
7) 西村成司，舘野論司，重原孝臣，長谷川秀彦，桧山澄子：共有メモリ型並列計算機における LAPACK の性能評価，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，Vol.43, No.SIG 6(HPS 5), pp.163–171 (2002).
8) 舘野論司，西村成司，重原孝臣，長谷川秀彦，桧山澄子：共有メモリ型並列計算機向け線形演算ライブラリにおける並列化手法の評価，情報処理学会研究報告 2002-HPC-90，pp.7–12 (2002).
9) OpenMP Architecture Review Board: http://www.OpenMP.org/.
10) 小国 力，村田健郎，三好俊郎，Dongarra, J.J., 長谷川秀彦：行列計算ソフトウエア，丸善 (1991).
11) Dongarra, J.J., Duff, I.S., Sorensen, D.C. and van der Vorst, H.A.: *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM (1991).
12) Lawson, C., Hanson, R., Kincaid, D. and Krogh, F.: Basic Linear Algebra Subprograms for FORTRAN Usage, *ACM Trans. Math. Software*, Vol.5, pp.308–325 (1979).
13) Dongarra, J.J., Du Croz, J., Hammarling, S. and Hanson, R.: An Extended Set of FORTRAN Basic Linear Algebra Subprograms, *ACM Trans. Math. Software*, Vol.14, pp.1–17 (1988).
14) Dongarra, J.J., Du Croz, J., Duff, I.S. and Hammarling, S.: A Set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Software*, Vol.16, pp.1–17 (1990).
15) NAG Fortran SMP Library Documentation: http://www.nag.co.uk/numeric/FL/manual/html/FSlibrarymanual.asp.
16) Nishiyama, H., Motokawa, K., Kyushima, I. and Kikuchi, S.: Pseudo-vectorizing Compiler for the SR8000, *Proc. Euro-Par 2000, LNCS 1900*, pp.1023–1027, Springer-Verlag (2000).
17) Brehm, M., Bader, R., Heller, H. and Ebner, R.: Pseudovectorization, SMP, and Messege Passing on the Hitachi SR8000-F1, *Proc. Euro-Par 2000, LNCS 1900*, pp.1351–1361, Springer-Verlag (2000).
18) Netlib: http://www.netlib.org/.
19) プログラムのソースコード：http://www.me.ics.saitama-u.ac.jp/~sigehara/hps/hps.html.
20) Dongarra, J.J., Duff, I.S., Sorensen, D.C. and van der Vorst, H.A.: *Numerical Linear Algebra for High-Performance Computers*, SIAM (1998).
21) Dhillon, I.S.: *A New $O(n^2)$ Algorithm for Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*, Ph.D. Thesis, University of California at Berkeley (1997).
22) Whaley, R.C. and Dongarra, J.J.: Automatically Tuned Linear Algebra Software, *Proc. SC98 Conference*, Orlando, FL, IEEE Publications (1998). http://www.netlib.org/atlas/

(平成 15 年 1 月 30 日受付)
(平成 15 年 5 月 14 日採録)

舘野　諭司

昭和 54 年生．平成 13 年埼玉大学工学部情報システム工学科卒業．平成 15 年同大学大学院理工学研究科情報システム工学専攻修了．同年東京コンピュータサービス株式会社入社．専門分野はハイパフォーマンスコンピューティング，並列数値計算．

重原　孝臣（正会員）

昭和 35 年生．昭和 58 年東京大学理学部物理学科卒業．昭和 63 年同大学大学院理学系研究科物理学専攻修了．同年より東京大学大型計算機センター研究開発部助手，平成 9 年より埼玉大学工学部情報システム工学科講師を経て，現在，同助教授．理学博士．専門分野はハイパフォーマンスコンピューティング，並列数値計算，数理物理，量子カオス．電子情報通信学会，日本物理学会各会員．

長谷川秀彦（正会員）

昭和 33 年生．昭和 55 年筑波大学第 1 学群自然学類卒業．昭和 58 年同大学大学院社会工学研究科中退．図書館情報大学助教授を経て，現在，筑波大学助教授．博士（工学）．PHASE，LA 研究会を運営する．専門分野は線形数値計算．日本応用数理学会，SIAM，ACM 各会員．

桧山　澄子（正会員）

昭和 15 年生．昭和 38 年宇都宮大学教育学部数学科卒業．昭和 40 年東京都立大学大学院理学研究科数学専攻修了．昭和 43 年より東京大学地震研究所所属．平成 13 年同研究所地震予知情報センター停年退官．現在，埼玉大学理学部数学科非常勤講師．博士（工学）．専門分野はハイパフォーマンスコンピューティング，数値計算．日本応用数理学会会員．