

JAERI-Data/Code
97-016



実対称帯行列に対する
一般固有値問題計算ライブラリの並列化

1997年5月

田中靖久

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは、日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは、日本原子力研究所研究情報部研究情報課（〒319-11 茨城県那珂郡東海村）あて、お申し越しください。なお、このほかに財団法人原子力弘済会資料センター（〒319-11 茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.

Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

© Japan Atomic Energy Research Institute, 1997

編集兼発行 日本原子力研究所
印 刷 細原子力資料サービス

実対称帯行列に対する一般固有値問題計算ライブラリの並列化

日本原子力研究所計算科学技術推進センター

田中 靖久

(1997年4月2日受理)

テキサス大開発のブロックランチヨス法による、実対称帯行列に対する標準固有値問題の求解ライブラリ L A S O (Lanczos Algorithm with Selective Orthogonalization) をベースに、実対称帯行列に対する一般固有値問題の並列計算ライブラリを開発した。

ブロックランチヨス法による一般固有値問題の求解ルーチンについて、自由度分割による並列化手法を検討し、その並列化アルゴリズムの性能評価を行った。なお、並列化は分散メモリ型スカラ並列計算機である IBM SP および日立 SR 2201 上で行った。

Parallelization of Mathematical Library
for Generalized Eigenvalue Problem for Real Band Matrices

Yasuhisa TANAKA

Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
Nakameguro, Meguro-ku, Tokyo

(Received April 2, 1997)

This research has focused on a parallelization of the mathematical library for a generalized eigenvalue problem for real band matrices on IBM SP and Hitachi SR2201. The origin of the library is LASO (Lanczos Algorithm with Selective Orthogonalization), which was developed on the basis of Block Lanczos method for standard eigenvalue problem for real band matrices at Texas University.

We adopted D.O.F. (Degree Of Freedom) decomposition method for a parallelization of this library, and evaluated its parallel performance.

Keywords: Generalized Eigenvalue Problem, Parallel Calculation, Block Lanczos Method, LASO, D.O.F. Decomposition Method, IBM SP, Hitachi SR2201

目 次

1.はじめに	1
2.一般固有値問題	2
2.1 方程式	2
2.2 問題	2
2.3 応用	2
2.4 計算手法	3
2.4.1 標準固有値問題に対するランチヨス法	3
2.4.2 一般固有値問題に対するランチヨス法	3
2.4.3 一般固有値問題に対するブロックランチヨス法	5
3.並列化	7
3.1 概説	7
3.2 自由度分割による並列化手法	8
3.2.1 概説	8
3.2.2 並列化手順	9
4.並列化性能評価	14
4.1 使用並列計算機	14
4.2 計算例題	14
4.3 性能評価結果	15
4.3.1 修正コレスキーフ分解	15
4.3.2 前進／後退代入	19
4.3.3 線形方程式の求解	23
4.3.4 EXPANDアルゴリズム	27
4.3.5 プログラム全体	29
5.結言	33
謝辞	33
参考文献	34

Contents

1. Introduction	1
2. Generalized Eigenvalue Problem	2
2.1 Equations	2
2.2 Problem	2
2.3 Application	2
2.4 Numerical Computation Methods	3
2.4.1 Lanczos Method for Standard Eigenvalue Problem	3
2.4.2 Lanczos Method for Generalized Eigenvalue Problem	3
2.4.3 Block Lanczos Method for Generalized Eigenvalue Problem	5
3. Parallelization	7
3.1 Overview	7
3.2 Parallelization Method by Degree of Freedom Decomposition	8
3.2.1 Overview	8
3.2.2 Parallelization Procedure	9
4. Performance Evaluation for Parallelization	14
4.1 Parallel Computers in Employ	14
4.2 Calculation Examples	14
4.3 Results of Performance Evaluation	15
4.3.1 Modified Cholesky Decomposition	15
4.3.2 Forward and Backward Substitution	19
4.3.3 Linear Solver	23
4.3.4 EXPAND Algorithm	27
4.3.5 Total Evaluation	29
5. Conclusions	33
Acknowledgements	33
References	34

1. はじめに

近年における並列計算機の著しい進歩に伴い、並列計算機の利用技術に関する研究が盛んに行われている。また、超並列計算機の普及、及び、利用促進を図るためにいくつかの方策が提唱、実施されているが、数値計算ライブラリの整備が不可欠な要素となる。既存ソフトウェアの超並列計算機への移行、もしくは超並列計算機でのソフトウェアの新規開発において、開発者自らがその都度、数値計算ライブラリを開発することは、大きな負担となるとともに、それ自体の信頼性の検証のために多くの労力を要することとなる。また、今日発展途上である超並列計算機は内部処理方法において機種依存性が強く、数値計算ライブラリも機種毎での開発が必要となっているのが現状である。そこで、超並列計算機の利用者に対し共通基盤として充分信頼の足る、機種依存性を排除した数値計算ライブラリを提供することは有意義である。

世界的に見ると、行列やベクトルの四則演算の並列化ルーチンの整備はほぼ完了している。しかし、構造物の自由振動問題などに応用のある、実対称帶行列に対する一般固有値問題の求解ライブラリの並列化はあまり行われていない。そこで、一般固有値問題の求解ライブラリの並列化を行った。ここでは、テキサス大学で開発されたPDS (Public Domain Software) であるLAS0 (実対称帶行列に対する標準固有値問題のブロックランチヨス法¹⁾による求解ライブラリ) をベースにして開発した。まず、一般固有値問題の求解ルーチンに改良し、次に並列化を行った。

ブロックランチヨス法は、大規模行列に対する一般固有値問題の求解手法として、他の手法に比べて、1プロセッサによる処理速度が速い。並列ライブラリの性能評価としては、並列化による速度向上はあくまでも二次的評価（相対的評価）であり、一次的評価（絶対的評価）である処理速度が重要であるのは言うまでもない。そういう意味でも、1プロセッサによる処理速度が速い、ブロックランチヨス法による並列化求解ルーチンの研究開発を行うことは大変意義深い。また、報告者が知る限り、ブロックランチヨス法による一般固有値問題の並列計算の研究例はない。

通信ライブラリとしては、機種依存性のない汎用的な通信ライブラリであるMPIを用いた。本年度はまずスカラ並列版を開発し、分散メモリ型スカラ並列計算機であるIBM SPおよび日立SR2201上で並列化性能評価を行った。次年度以降に、ベクトル並列版の開発を予定している。

本研究においては、一般固有値問題の求解ルーチンの主要な処理である線形方程式の解法について、自由度分割による並列化手法を検討し、その並列化アルゴリズムの性能評価を行った。

2. 一般固有値問題

本研究において取り扱う一般固有値問題の求解ルーチンの概略を述べる。

2.1 方程式

標準固有値問題および一般固有値問題は (2-1) および (2-2) で定式化される。

$$[A]\{x\} = \lambda\{x\} \quad (2-1)$$

$$[A]\{x\} = \lambda[B]\{x\} \quad (2-2)$$

ここで、 $[A]$ および $[B]$ は実対称帶行列で、ともに非負定値である。 λ は固有値、 $\{x\}$ は固有ベクトルである。

2.2 問題

一般固有値問題については実用上、以下の2タイプの問題が考えられる。

- (1) 求める固有値 λ の範囲の上下限を与え、その範囲内の全ての固有値およびそれに対応する固有ベクトル $\{x\}$ を求める。
- (2) 求める固有値 λ の範囲の下限 σ および求める固有値の個数 N_λ を与え、 σ より大きい固有値を小さい方から（少数の） N_λ 個求める。各固有値に対応する固有ベクトル $\{x\}$ も同時に求める。

本ライブラリにおいては、上記の (2) のタイプの問題を解くことを目的とする。ここで、求める固有値 λ の範囲の下限 σ と求める固有値の個数 N_λ は引数で指定する。

2.3 応用

代表的な応用例である構造物の振動問題においては、 $[A]$ は剛性マトリックス $[K]$ に、 $[B]$ は質量マトリックス $[M]$ に相当する。全体マトリックスを組み立てた後、拘束自由度を消去した $[K]$ と $[M]$ に対して、本ライブラリをCALLすることになる。

2.4 計算手法

2.4.1 標準固有値問題に対するランチョス法

標準固有値問題 (2-1) に対するランチョス法¹⁾ は、以下のように定式化される。

以下の定式化において、小文字で表記した文字のうち、 α_j および β_{j+1} のみをスカラーとし、それ以外をベクトルとする。

(1) 初期化

(a) 初期ベクトルとして、単位ベクトル q_1 を任意にとる。

(b) $\beta_1 = 0$ および $q_0 = 0$ とする。

(2) 反復計算

$j=1, 2, 3, \dots$ に対して、(2-3) で与えられる反復計算を収束するまで行う。

$$r_{j+1} = Aq_j - \alpha_j q_j - \beta_j q_{j+1} \quad (2-3)$$

ここで、 $\alpha_j, \beta_{j+1}, q_{j+1}$ はそれぞれ (2-4) ~ (2-6) により定式化される。 $|r_{j+1}|$ はベクトル r_{j+1} のユークリッドノルムである。

$$\alpha_j = q_j^T A q_j \quad (2-4)$$

$$\beta_{j+1} = |r_{j+1}| \quad (2-5)$$

$$q_{j+1} = \frac{r_{j+1}}{\beta_{j+1}} \quad (2-6)$$

ベクトルの組 $Q_j = (q_1, q_2, \dots, q_j)$ は正規直交マトリックスとなる。

2.4.2 一般固有値問題に対するランチョス法

一般固有値問題に対するランチョス法¹⁾ については、収束性を十分良くするために、一般固有値問題 (2-2) を変形して (2-7) とする。

$$[B][A - \sigma B]^{-1}[B]\{x\} = \theta[B]\{x\} \quad (2-7)$$

ここで、 θ は (2-8) により定義される。

$$\theta = \frac{1}{\lambda - \sigma} \quad (2-8)$$

これにより、2.2節で述べた問題は、(2-7) を満たす固有値 θ と固有ベクトル $\{x\}$ のうち、大きい方から N_λ 個求める問題に帰着される。

一般固有値問題 (2-2) に対するランチョス法は、以下のように定式化される。

(1) 初期化

(a) 初期ベクトル r_1 を任意にとる。

(b) (2-9) により、 r_1 を計算する。

$$r_1 = (A - \sigma B)^{-1} B r_1 \quad (2-9)$$

(c) r_1 を (2-10) により正規化した q_1 を求める。

$$q_1 = \frac{r_1}{(r_1^T B r_1)^{1/2}} \quad (2-10)$$

(d) $\beta_1 = 0$ および $q_0 = 0$ とする。

(2) 反復計算

$j=1, 2, 3, \dots$ に対して、(2-11) の反復計算を収束するまで行う。

$$r_{j+1} = (A - \sigma B)^{-1} B q_j - \alpha_j q_j - \beta_j q_{j+1} \quad (2-11)$$

ここで、 $\alpha_j, \beta_{j+1}, q_{j+1}$ はそれぞれ (2-12) ~ (2-14) により定式化される。

$$\alpha_j = q_j^T B (A - \sigma B)^{-1} B q_j \quad (2-12)$$

$$\beta_{j+1} = (r_{j+1}^T B r_{j+1})^{1/2} \quad (2-13)$$

$$q_{j+1} = \frac{r_{j+1}}{\beta_{j+1}} \quad (2-14)$$

ベクトルの組 $Q_j = (q_1, q_2, \dots, q_j)$ は B 直交性、すなわち (2-15) を満たすものとする。ここで、 δ_{ij} はクロネッカのデルタである。

$$(q_i, Bq_j) = \delta_{ij} \quad (2-15)$$

2.4.3 一般固有値問題に対するブロックランチヨス法

本ライブラリが適用する一般固有値問題に対するブロックランチヨス法¹⁾ は、以下のように定式化される。ここで、自由度数を N 、ブロックサイズを p とする。

(1) 初期化

- (a) p 個の列ベクトルを任意にとり、それらからなる $N \times p$ マトリックス R_1 を初期ブロックベクトルとする。
- (b) (2-16) により、 R_1 を計算する。

$$R_1 = (A - \sigma B)^{-1} B R_1 \quad (2-16)$$

- (c) R_1 は (2-17) により QR 分解する。

$$R_1 = Q_1 C_0 \quad (2-17)$$

- (d) p 次正方上三角行列 $C_1 = 0$ および $Q_0 = 0$ とする。

(2) 反復計算

- (a) $j = 1, 2, 3, \dots$ に対して、(2-18) の反復計算を収束するまで行う。

$$R_{j+1} = (A - \sigma B)^{-1} B Q_j - Q_j D_j - Q_{j+1} C_j^T \quad (2-18)$$

ここで、 D_j は (2-19) を満たす。

$$D_j = Q_j^T B (A - \sigma B)^{-1} B Q_j \quad (2-19)$$

(b) (2-20) により、QR分解を計算する。

$$R_{j+1} = Q_{j+1} C_{j+1} \quad (2-20)$$

ここで、 Q_{j+1} は $N \times p$ マトリックスで、 p 個の列ベクトルは B 直交性を満たすものとする。 C_{j+1} は p 次正方上三角行列とする。

3. 並列化

3.1 概説

2.2節で述べたように、一般固有値問題については実用上、以下の2タイプの問題が考えられる。

- (1) 求める固有値 λ の範囲の上下限を与え、その範囲内の全ての固有値およびそれに対応する固有ベクトル $\{x\}$ を求める。
- (2) 求める固有値 λ の範囲の下限 σ および求める固有値の個数 N_λ を与え、 σ より大きい固有値を小さい方から（少数の） N_λ 個求める。各固有値に対応する固有ベクトル $\{x\}$ も同時に求める。

(1) のタイプの並列化については、指定した範囲を最初から各プロセッサに均等に分割して処理する方法が標準的である。ユーザが固有値の分布をある程度知っている場合には、それに応じて区間の分割を指定することも考えられる。

一方、本ライブラリで対象とする (2) のタイプについては、2.4.3節で述べたように、(3-1) および (3-2) からなる反復計算が主要な処理となる。

$$R_{j+1} = (A - \sigma B)^{-1} B Q_j - Q_j D_j - Q_{j+1} C_j^T \quad (3-1)$$

$$R_{j+1} = Q_{j+1} C_{j+1} \quad (3-2)$$

特に、(3-1) の右辺第一項である (3-3) のタイプの計算が最もコストのかかる処理となる。

$$Y = (A - \sigma B)^{-1} BX \quad (3-3)$$

(3-3) は線形方程式 (3-4) を解くことに帰着される。

$$(A - \sigma B)Y = BX \quad (3-4)$$

ということで、以下の4つの並列化が考えられる。

- (1) 最初から固有値のサーチ範囲、すなわちシフト量 σ を各プロセッサで変化させて並列化させる。
- (2) 自由度 N を各プロセッサで分割して並列化する（自由度分割）。

- (3) ブロックサイズ p を各プロセッサで分割して並列化する（ブロック分割）。
- (4) 自由度分割とブロック分割を組み合わせて並列化する（ブロック自由度分割）。

(1) は求めるべき個数の固有値の分布範囲をあらかじめ知る必要があり、現実的には容易ではない。

(2) の自由度分割は、 N_p 個のプロセッサで並列化する場合に、マトリックスデータを格納するメモリコストが約 $(1/N_p)$ に減少するメリットがある。また、ブロックサイズ p の制約も受けない。

(3) のブロック分割は、ブロックサイズ p をプロセッサ N_p で分割して、線形方程式 (3-4) を解く方法である。メモリコストにおける並列化のメリットはないが、ブロックサイズ p がプロセッサ数 N_p で割り切れる場合にはロードバランスが良好で、特に有効である。

将来的には、自由度数 N 、プロセッサ数 N_p 、ブロックサイズ p およびメモリサイズに応じて、自由度分割とブロック分割を組み合わせる、(4) のブロック自由度分割を本ライブラリの最終目標としていきたい。

本研究では、(2) の自由度分割による並列化を詳細に検討する。

3.2 自由度分割による並列化手法

3.2.1 概説

線形方程式 (3-4) の直接解法の並列化においては、自由度の番号付け (sequence generation) が鍵となるが、最近、多重スカイライン法²⁾が提唱されており、そのアイデアは非常に興味深い。多重スカイライン法は本質的に、二進木 (binary tree) ベースのアルゴリズムであり、各プロセッサ間のロードバランスが良い利点を持ち合わせており、並列化に適している。しかし、正方形領域に対する番号付けのルールおよび並列化アルゴリズムは明確に示されているが、複雑な形状をもつ一般的な領域に対して、番号付けのルールは確立していない。

本研究では、数値計算ライブラリの汎用性を重視するという性格上のこともあり、幾何的数据から定まるsequence generatorを入力とせず、ライブラリの中で番号付けを行う並列化手法を検討した。多重スカイライン法を基本アイデアとしたが、それが多次元分割であるのに対して、本手法は一次元分割であるため、多重スカイライン法の並列化性能における優位性は否めない。しかし、本手法においては、ユーザが幾何的数据からsequence generatorを生成する手間を必要としない簡便性および汎用性を有している。短距離力分子動力学法 (MD) の並列化手法に例えると、多重スカイライン法が領域分割法に、本手法が粒子分割法に相当する。

3.2.2 並列化手順

異なる2つの自由度*i, j*に対して、マトリックス $E = (A - \sigma B)$ の非対角成分 (*i, j*) が非零 (nonzero) であるとき異なる2つの自由度*i*と*j*が連成すると呼び、逆に非対角成分 (*i, j*) が零 (zero) であるとき異なる2つの自由度*i*と*j*が連成しないと呼ぶことにする。

(1) 担当自由度

全自由度数Nを N_p 個のプロセッサに均等に分割し、各プロセッサの担当自由度とする。特に、自由度数Nがプロセッサ数 N_p で割り切れる場合には、プロセッサ1は1~ N/N_p の自由度を、プロセッサ2は $(N/N_p) + 1$ ~ $2N/N_p$ の自由度を担当することになる。

(2) 境界自由度

プロセッサ $P=2 \sim N_p$ については、番号の小さい方から境界自由度を設定する。境界自由度の個数は、境界自由度の両側の自由度（左側はプロセッサ $P-1$ の右端の担当自由度、右側はプロセッサ P の左端の担当自由度）が連成しないという条件を満たす値のうち、最小値を選択する。従って、境界自由度の個数はプロセッサにより可変である。

境界自由度以外の自由度を内部自由度と呼ぶ。

(3) 番号付け

まず、内部自由度について、プロセッサ1から順に、元の番号の小さい順に番号付けを行う。次に、境界自由度について、プロセッサ2から順に、元の番号の小さい順に番号付けを行う。

例えば、図3.1に示すように、プロセッサ数 $N_p=3$ 、 $N=9$ 個の自由度からなるばね系を考える。この場合、隣り同士の自由度しか連成しないため、各プロセッサの境界自由度の個数は1個となる。図3.1に示すように、番号付けを行う。

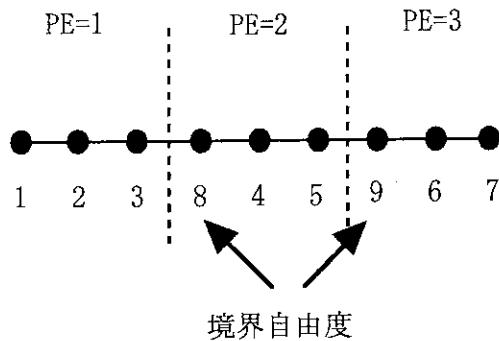


図3.1 番号付け

(4) 修正コレスキーフ分解

ここでも簡単のために、プロセッサ数 $N_p=3$ の例を考える。(3)で述べた番号付けにより、マトリックス $E = (A - \sigma B)$ は図3.2のようになる。

$E_{ib_{i-1}}$ の添字*i*はプロセッサ*i*の内部自由度、添字 b_{i-1} はプロセッサ*i*の境界自由度とする。 E_{11} はプロセッサ1の内部自由度同士の小行列の成分を意味し、 E_{1b_1} はプロセッサ1の内部自由度とプロセッサ2の境界自由度との小行列の成分を意味する。

$$\begin{bmatrix} E_{11} & E_{1b_1} & \\ E_{22} & E_{2b_1} & E_{2b_2} \\ E_{33} & & E_{3b_2} \\ & E_{b_1b_1} & \\ & & E_{b_2b_2} \end{bmatrix}$$

図3.2 番号付けされたマトリックス

なお、プロセッサ2の境界自由度とプロセッサ3の境界自由度との小行列の成分 $E_{b_1b_2}$ は元々零行列であるが(図3.2)、三角分解によりfill-inされ、上三角行列 $U_{b_1b_2}$ は非零となる。このように、隣接プロセッサどうしの境界自由度に関する小行列 $E_{b_{i-1}b_i}$ はfill-inされ、上三角行列 $U_{b_{i-1}b_i}$ は非零となる。

各プロセッサは、以下の手順で並列化処理を行う。

- ・全プロセッサPが、内部自由度同士の小行列 E_{pp} を三角分解する。
- ・ $1 \leq P \leq N_p - 1$ が、内部自由度と右隣のプロセッサP+1の境界自由度との小行列 E_{pb_p} を三角分解する。
- ・ $1 \leq P \leq N_p - 1$ が、内部自由度と右隣のプロセッサP+1の境界自由度との上三角行列 U_{pb_p} および対角行列 D_p をP+1にsendする。
- ・ $2 \leq P \leq N_p$ が、内部自由度と自分の境界自由度との小行列 $E_{pb_{p-1}}$ を三角分解する。
- ・ $2 \leq P \leq N_p$ が、左隣のプロセッサP-1の内部自由度と境界自由度との上三角行列 $U_{p-1b_{p-1}}$ および対角行列 D_{p-1} をP-1からreceiveする。
- ・ $3 \leq P \leq N_p$ が、境界自由度同士の小行列 $E_{b_{p-1}b_{p-1}}$ 、および境界自由度と右隣のプロセッサの境界自由度との小行列 $E_{b_{p-1}b_p}$ を更新し、P=2に全部集める（gather）。
- ・P=2が、全境界自由度同士の小行列 E_{bb} をまとめて三角分解する。

(5) 前進代入

各プロセッサは、以下の手順で並列化処理を行う。

- ・全プロセッサPが、内部自由度の前進代入を行う。
- ・ $1 \leq P \leq N_p - 1$ が、右隣のプロセッサの境界自由度と連成する内部自由度の解をP+1にsendする。
- ・ $2 \leq P \leq N_p$ が、境界自由度と内部自由度との連成分を計算し、境界自由度の右辺を更新する。
- ・ $2 \leq P \leq N_p$ が、境界自由度と左隣のプロセッサの内部自由度との解をP-1からreceiveする。
- ・ $2 \leq P \leq N_p$ が、境界自由度と左隣のプロセッサの内部自由度との連成分を計算し、境界自由度の右辺を更新する。

- ・ $3 \leq P \leq N_p$ の境界自由度の右辺を $P=2$ に全部集める (gather)。
- ・ $P=2$ が、全境界自由度の前進代入を行う。

(6) 後退代入

各プロセッサは、以下の手順で並列化処理を行う。

- ・ $P=2$ が、全境界自由度の後退代入を行う。
- ・ $P=2$ が、 $P=1$ および $3 \leq P \leq N_p$ に、各プロセッサの境界自由度の解および右隣のプロセッサの境界自由度の解を送る (scatter)。
- ・ $2 \leq P \leq N_p$ が、境界自由度と内部自由度との連成分を計算し、内部自由度の右辺を更新する。
- ・ $1 \leq P \leq N_p - 1$ が、内部自由度と右隣のプロセッサの境界自由度との連成分を計算し、内部自由度の右辺を更新する。
- ・全プロセッサ P が、内部自由度の後退代入を行う。

(7) ベクトルデータの共有

線形方程式の求解後、各プロセッサは、自身の担当自由度のベクトルデータのみを所有している。全プロセッサが全ベクトルデータを共有するために、EXPANDアルゴリズム³⁾ を採用する。EXPANDアルゴリズムは、二進木 (binary tree) ベースのアルゴリズムであるため、プロセッサ数が 2 のべき乗の場合に最も有効であるが、本ライブラリではプロセッサ数が 2 のべき乗でない場合にも、EXPANDアルゴリズムベースの通信方法を採用している。

(8) ベクトルの正規直交化

(3-2) の分解処理などに対して、 P 個のベクトルの組 (x_1, x_2, \dots, x_p) の正規

直交化の処理が現われる。 $i=1, 2, \dots, P$ に対して、次に示す(3-5)および(3-6)により、 P 個のベクトルの組 (x_1, x_2, \dots, x_p) の正規直交化を行う。

$$x_i = x_i - \sum_{k=1}^{i-1} (x_i, Bx_k) x_k \quad (3-5)$$

$$x_i = \frac{1}{\sqrt{(x_i, Bx_i)}} x_i \quad (3-6)$$

ベクトルデータの加算・減算・スカラ倍は、各プロセッサごとに担当自由度に対して行われる（自由度分割）。2つのベクトルの内積については、各プロセッサごとに担当自由度の内積の部分和を求め、全プロセッサで総和をとり、共有化する。総和の共有化については、グループ内の通信演算ルーチン (MPI_ALLREDUCE) による方法と、二進木 (binary tree) ベースの1対1通信ルーチン (MPI_SEND, MPI_RECV) を組み合わせた方法とを、通信のサイズにより使い分ける。

4. 並列化性能評価

4.1 使用並列計算機

本研究において使用した並列計算機は、分散メモリー型スカラ並列計算機IBM SPおよび日立 SR2201である。各並列計算機の諸元を表4.1に示す。計算に用いたプロセッサ数Npは、IBM SPがNp=1, 2, 4, 8, 16の5ケースで、日立 SR2201がNp=1, 2, 4, 8, 16, 32の6ケースである。

表4.1 各並列計算機の諸元

並列計算機	単一PEのメモリ サイズ (MB/PE)	単一PEのピーク演算 性能 (MFLOPS/PE)	PE間のピーク通信 性能 (MB/sec)
IBM SP	128	266	40
日立 SR2201	256	300	300

4.2 計算例題

計算例題としては、自由振動問題としては最も基本的な片持はり（図4.1）の自由振動問題を用いる。片持はりをFEM（有限要素法）でモデル化する際に生成される剛性マトリックス $[K]$ および質量マトリックス $[M]$ を、それぞれ $[A]$ および $[B]$ として入力する。質量マトリックス $[M]$ はLumped Massを用いた。

はりの分割数を N_{div} とすると、全体マトリックスから拘束6自由度を消去した、 $N = 6 \times N_{\text{div}}$ 個の自由度からなる方程式を考える。

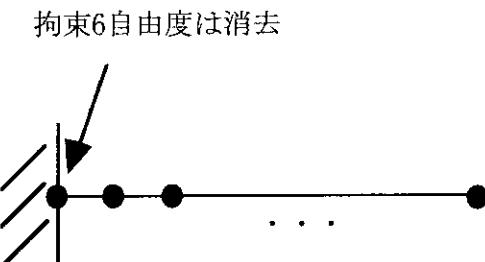


図4.1 片持はり

4.3 性能評価結果

ここでは、自由度数N=9600, 19200, 28800の3ケース（はりの分割数はそれぞれNN=1600, 3200, 4800）、求める固有値 $N_\lambda=10$ 個の場合の結果を報告する。

日立 SR2201は、ループ長が大きくなり、データがキャッシュからあふれると性能が大きく低下するという通常のRISCプロセッサの性能低下を防ぐことを主眼にした擬似ベクトル（pseudo-vectorize）機構を備えている。そこで、擬似ベクトル機構を用いたケース（「SR擬似」と呼ぶ）と、擬似ベクトル機構を用いないケース（「SRスカラ」と呼ぶ）との2ケースについて並列計算を行った。

4.3.1 修正コレスキーフ分解

本アルゴリズムの並列計算時間は、以下の3つに分けられる。

- (1) t_{i-comp} ：内部自由度同士の小行列 E_{ii} の三角分解に要する時間で、プロセッサ数 N_p の増加に伴い、ほぼ線形に減少すべき時間である。
- (2) t_{b-comp} ：(1)以外の演算、すなわち境界自由度に関する演算に要する時間である。
- (3) t_{comm} ：通信に要する時間である。

t_{i-comp} は、 N/N_p にほぼ比例する時間である。 t_{b-comp} は、境界自由度数にほぼ比例するが、 N/N_p にはさほど影響を受けない時間である。従って、速度向上比に利く t_{b-comp}/t_{i-comp} は、自由度数Nの増加に伴い、ほぼ単調に減少する。 t_{comm} はプロセッサ数 N_p および境界自由度数の増加に伴い、単調に増加するが、自由度数Nには無関係である。

各プロセッサ数に対する、修正コレスキーフ分解の並列計算時間および速度向上比を表4.2～4.4および表4.5～4.7に示す。各プロセッサ数に対する、修正コレスキーフ分解の速度向上比のグラフを図4.2～4.4に示す。

SPでは8プロセッサ以下で、SRでは16プロセッサ以下で、プロセッサ数 N_p の増加に伴い、計算時間は単調に減少しているが、それ以上では逆に増加している。通信コスト t_{comm} の増加の方が、 t_{b-comp}/t_{i-comp} の減少に比べて大きくなつたためである。

SP, SRスカラ, SR擬似とも、8プロセッサ以上では、自由度数Nの増加に伴い、速度向上比が単調に増加している。これは、境界自由度数は自由度数Nに無関係に一定であり、自由度数Nの増加に伴い、境界自由度数に対する内部自由度数の比率が増加したためである。より大規模なマトリックスに対しては、より高い速度向上比が期待できる。

各自由度数および各プロセッサ数において、SRスカラとSR擬似の計算時間の差は殆どない。

本アルゴリズムでは、プロセッサ1が境界自由度を所有せず、境界自由度に関する処理に

一切関わらないため、以下の3つの処理を終えた後、idle状態に入る。

- ・内部自由度同士の小行列 E_{ii} を三角分解する。
- ・内部自由度と右隣のプロセッサの境界自由度との小行列 E_{ib_i} を三角分解する。
- ・内部自由度と右隣のプロセッサの境界自由度との上三角行列 U_{ib_i} および対角行列 D_i をプロセッサ2にsendする。

一方、プロセッサ2が全境界自由度の三角分解を行うため、常に最も負荷が大きい。これは一次元分割の宿命とも言える欠点である。全境界自由度の三角分解を現在のプロセッサ2に代わり、プロセッサ1が行う方法も考えられるが、通信量が（プロセッサ2の担当する境界自由度に対応するマトリックスの分だけ）増加するため、得策と言えない。但し、この負荷不均衡を見直すために、全自由度を各プロセッサに均衡に割り振る現行の方法を、プロセッサ1に多く割り振るように変更する余地はあるが、この自由度分割の変更が前進／後退代入やベクトルの正規直交化など他の処理に影響を与えるので、一概に善悪を言えない。

さらに速度向上比を上げるために、プロセッサ1以外のプロセッサで行われる境界自由度関連の処理のチューニングと、プロセッサ数が十分大きくなつた場合のMPIによる通信のチューニングを並行して行うことが必要となる。

表4.2 修正コレスキーフィルタの計算時間 (SP、単位: sec)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	0.051	0.052	0.030	0.022	0.033
19200	0.099	0.101	0.054	0.034	0.036
28800	0.143	0.154	0.079	0.045	0.039

表4.3 修正コレスキーフィルタの計算時間 (SRスカラ、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	0.09	0.09	0.05	0.03	0.03	0.06
19200	0.19	0.19	0.09	0.06	0.05	0.07
28800	0.28	0.28	0.13	0.07	0.05	0.07

表4.4 修正コレスキーフィルタの計算時間 (SR擬似、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	0.10	0.08	0.04	0.03	0.03	0.06
19200	0.20	0.16	0.09	0.05	0.04	0.07
28800	0.30	0.24	0.12	0.07	0.04	0.07

表4.5 修正コレスキーフィルタの速度向上比 (SP)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	1.000	0.981	1.700	2.318	1.545
19200	1.000	0.980	1.833	2.912	2.750
28800	1.000	0.929	1.810	3.178	3.667

表4.6 修正コレスキーフィルタの速度向上比 (SRスカラ)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.000	1.000	1.800	3.000	3.000	1.500
19200	1.000	1.000	2.111	3.167	3.800	2.714
28800	1.000	1.000	2.154	4.000	5.600	4.000

表4.7 修正コレスキーフィルタの速度向上比 (SR擬似)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.000	1.250	2.500	3.333	3.333	1.667
19200	1.000	1.250	2.222	4.000	5.000	2.857
28800	1.000	1.250	2.500	4.286	7.500	4.286

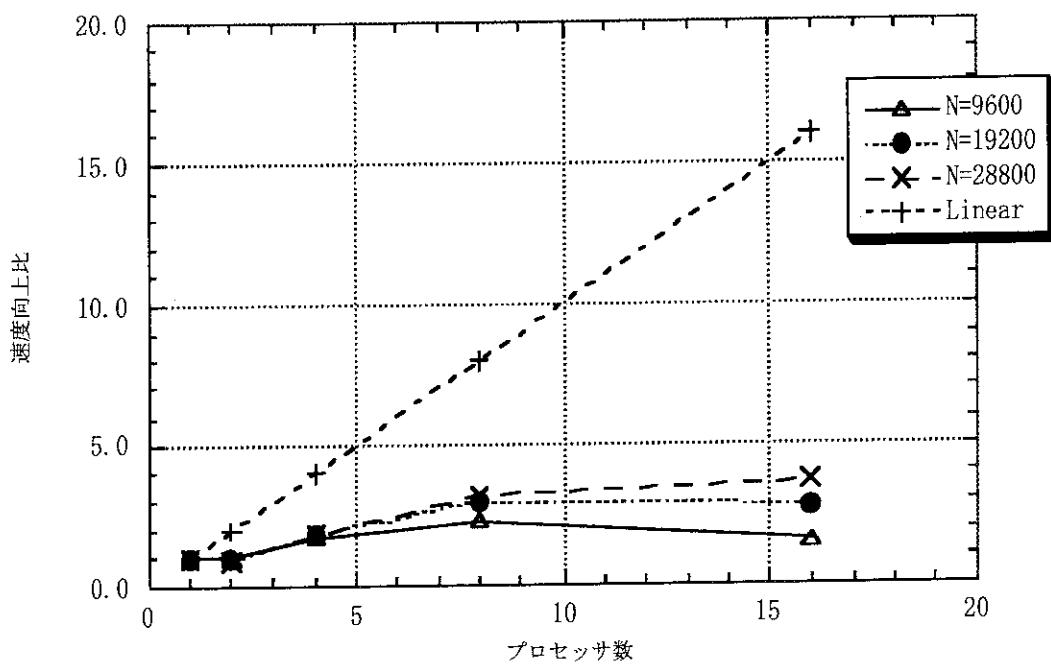


図4.2 修正コレスキーフィルタの速度向上比 (SP)

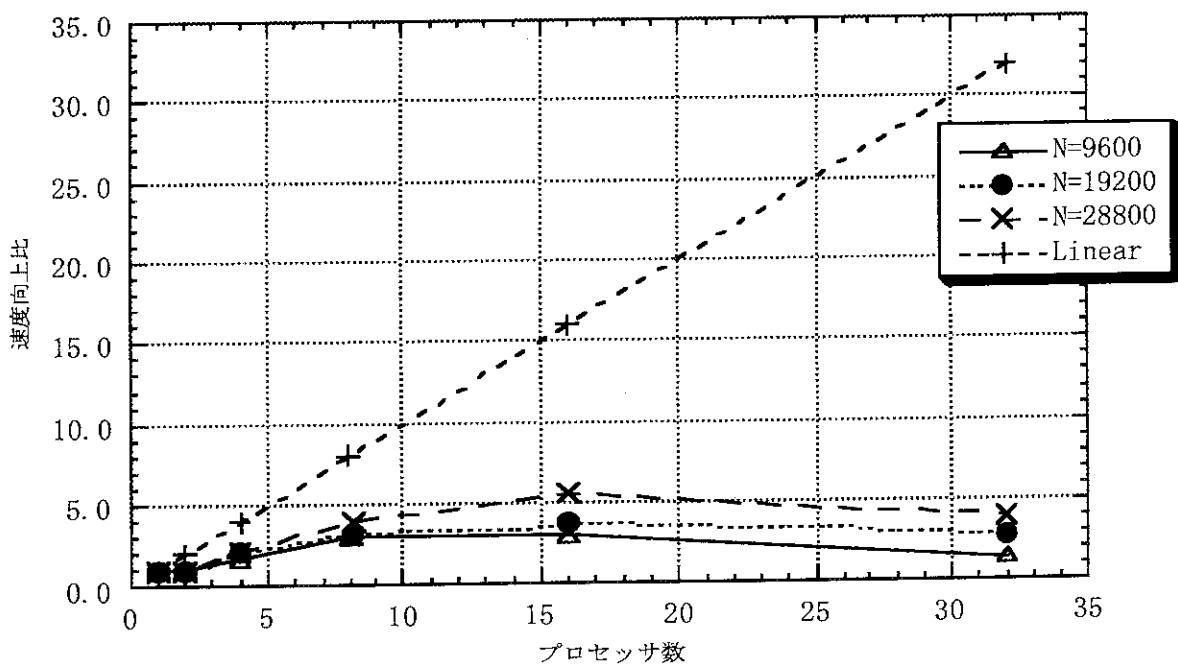


図4.3 修正コレスキーフィルタの速度向上比 (SRスカラ)

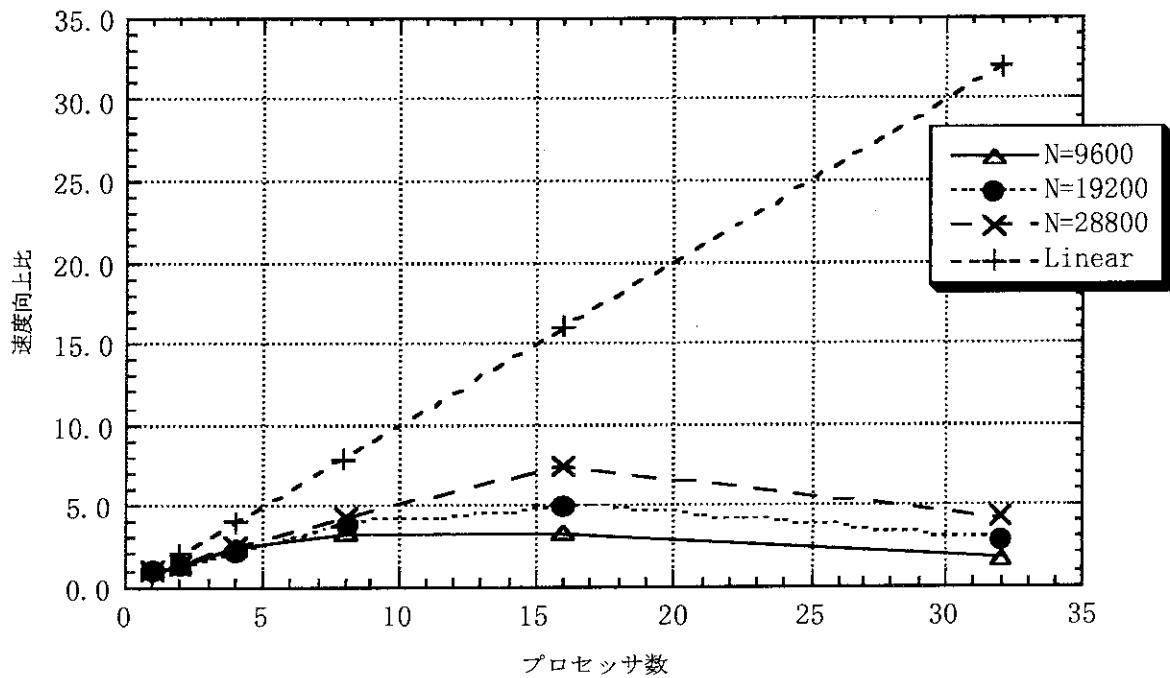


図4.4 修正コレスキーフ分解の速度向上比 (SR擬似)

4.3.2 前進／後退代入

ここでいう前進／後退代入の計算時間とは、ベクトルデータに対する行列 $[B]$ の乗算および前進／後退代入に要する計算時間を意味する。

本アルゴリズムの並列計算時間は、以下の4つに分けられる。

- (1) t_{i-comp} : 内部自由度の前進／後退代入に要する時間で、プロセッサ数 N_p の増加に伴い、ほぼ線形に減少すべき時間である。
- (2) t_{b-comp} : (1) 以外の演算、すなわち境界自由度に関する演算に要する時間である。
- (3) t_{i-comm} : 前進代入後、右隣のプロセッサの境界自由度と連成する内部自由度の解の通信に要する時間である。
- (4) t_{b-comm} : (3) 以外の通信に要する時間である。すなわち、全境界自由度の右辺をプロセッサ2に集め(gather)、プロセッサ2が前進／後退代入により求めた全境界自由度の解を各プロセッサに送信する(scatter)のに要する時間である。

t_{i-comp} は、 N/N_p にほぼ比例する時間である。 t_{b-comp} は、境界自由度数にほぼ比例するが、 N/N_p にはさほど影響を受けない時間である。従って、速度向上比に利く t_{b-comp} / t_{i-comp} は、自

由度数Nの増加に伴い、ほぼ単調に減少する。 t_{i-comm} は隣接プロセッサ間のローカル通信で、しかも通信サイズも小さいため、他の3つの時間に比べて十分小さい。 t_{b-comm} は、ほぼ全プロセッサが参加する通信であるため、プロセッサ数 N_p および境界自由度数の増加に伴い、単調に増加する。但し、自由度数Nには無関係である。

各プロセッサ数に対する、前進／後退代入の並列計算時間を表4.8～4.10に、速度向上比を表4.11～4.13に示す。

各プロセッサ数に対する、前進／後退代入の速度向上比のグラフを図4.5～4.7に示す。

SP, SRスカラ, SR擬似とも、16プロセッサ以下では、プロセッサ数 N_p の増加に伴い、計算時間は減少しているが、SRの32プロセッサでは逆に増加しているケースもある。通信コスト t_{b-comm} の増加の方が、 t_{b-comp}/t_{i-comp} の減少に比べて大きくなつたためである。

SP, SRスカラ, SR擬似とも、8プロセッサ以上では、自由度数Nの増加に伴い、速度向上比が増加している。これは、境界自由度数は自由度数Nに無関係に一定であり、自由度数Nの増加に伴い、境界自由度数に対する内部自由度数の比率が増加したためである。より大規模なマトリックスに対しては、より高い速度向上比が期待できる。

8プロセッサ以下では、各自由度数において、SR擬似の方がSRスカラより約1.3～1.6倍速い。16プロセッサでは、N=9600のケースは両者にさほど差がないが、N=19200, 28800のケースでは、SR擬似の方がSRスカラより約1.3倍速い。32プロセッサになると、計算時間の差は殆どない。本アルゴリズムでは、 N/N_p をループ長と見なせるが、これが十分大きい場合には、擬似ベクトル機構の効果が出ている。擬似ベクトル機構の効果が出るループ長のしきい値を知るには、さらに実験が必要となる。

本並列化アルゴリズムでは、修正コレスキーフィルタと同様に、全境界自由度の前進／後退代入を行うプロセッサ2の負荷が最も大きく、境界自由度の求解を担当しないプロセッサ1の負荷が最も小さい。各プロセッサへの自由度の割り振りの変更については、修正コレスキーフィルタの節で述べたことと同様なことが言える。

さらに速度向上比を上げるために、これも修正コレスキーフィルタと同様に、プロセッサ1以外のプロセッサで行われる境界自由度関連の処理のチューニングと、プロセッサ数が十分大きくなつた場合のMPIによる通信のチューニングを並行して行うことが必要となる。

表4.8 前進／後退代入の計算時間 (SP、単位: sec)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	0.420	0.341	0.177	0.101	0.078
19200	0.946	0.688	0.346	0.184	0.116
28800	1.410	1.035	0.520	0.269	0.158

表4.9 前進／後退代入の計算時間 (SRスカラ、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	0.72	0.57	0.27	0.21	0.18	0.24
19200	1.49	1.34	0.62	0.38	0.31	0.27
28800	2.25	2.08	0.97	0.49	0.35	0.30

表4.10 前進／後退代入の計算時間 (SR擬似、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	0.53	0.37	0.21	0.15	0.19	0.23
19200	1.07	0.85	0.42	0.26	0.24	0.26
28800	1.76	1.42	0.61	0.36	0.27	0.28

表4.11 前進／後退代入の速度向上比 (SP)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	1.000	1.232	2.373	4.158	5.385
19200	1.000	1.375	2.734	5.141	8.155
28800	1.000	1.362	2.712	5.242	8.924

表4.12 前進／後退代入の速度向上比 (SRスカラ)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.000	1.263	2.667	3.429	4.000	3.000
19200	1.000	1.112	2.403	3.921	4.806	5.519
28800	1.000	1.082	2.320	4.592	6.429	7.500

表4.13 前進／後退代入の速度向上比 (SR擬似)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.000	1.432	2.524	3.533	2.789	2.304
19200	1.000	1.259	2.548	4.115	4.458	4.115
28800	1.000	1.239	2.885	4.889	6.519	6.286

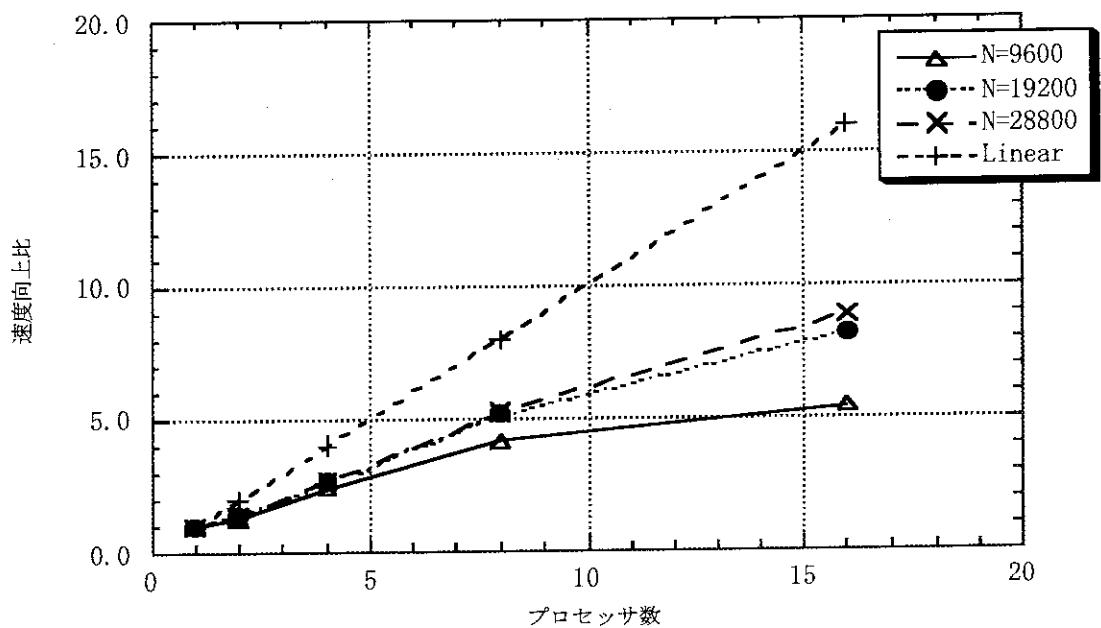


図4.5 前進／後退代入の速度向上比 (SP)

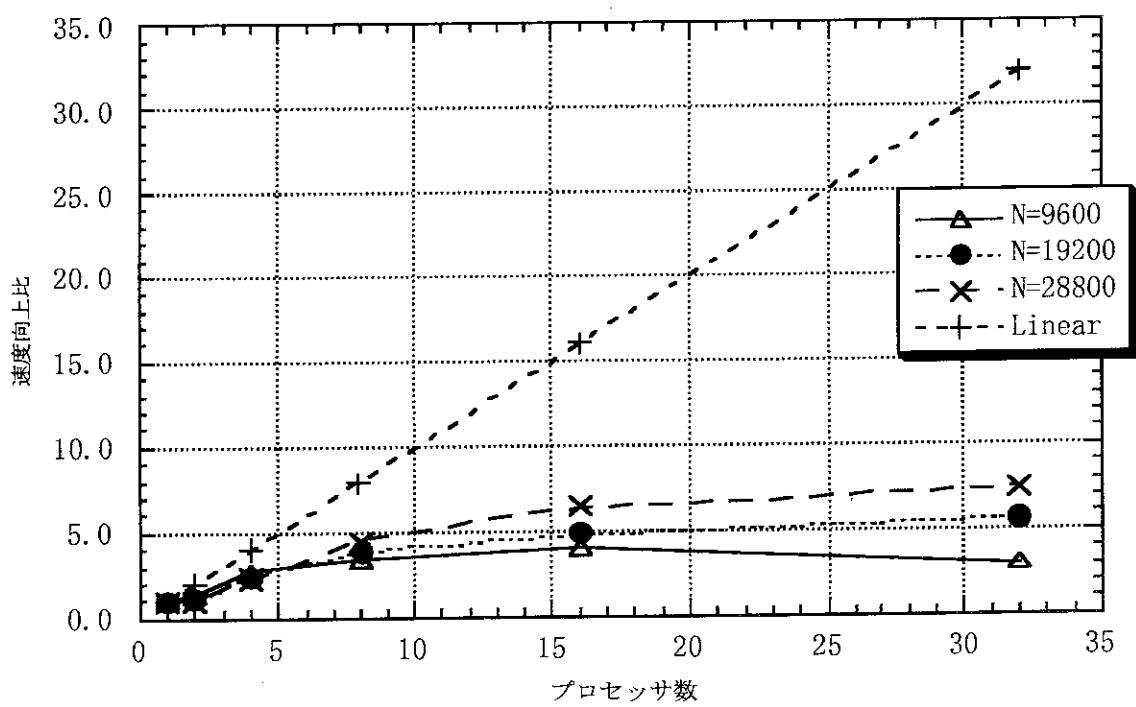


図4.6 前進／後退代入の速度向上比 (SRスカラ)

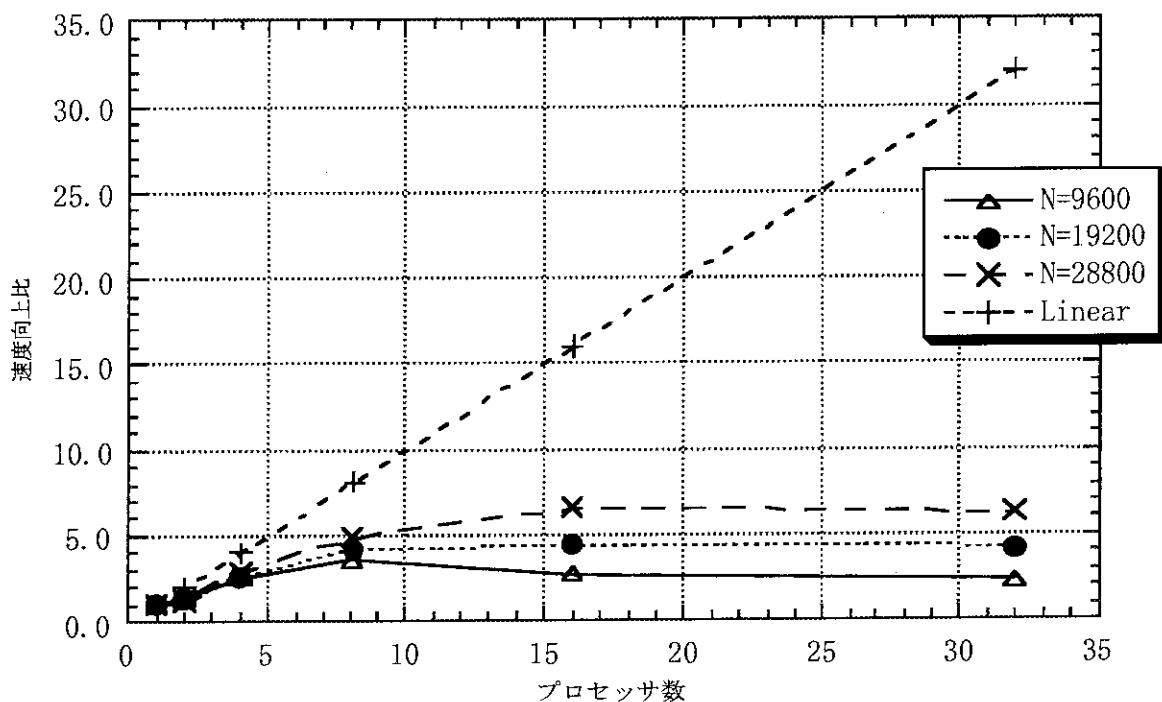


図4.7 前進／後退代入の速度向上比 (SR擬似)

4.3.3 線形方程式の求解

ここでは、修正コレスキーフ分解と前進／後退代入とを合わせた線形方程式の求解ルーチン全体の性能評価を行う。

各プロセッサ数に対する、線形方程式の求解ルーチン全体の並列計算時間を表4.14～4.16に、速度向上比を表4.17～4.19に示す。

各プロセッサ数に対する、線形方程式の求解ルーチン全体の速度向上比のグラフを図4.8～4.10に示す。

SP, SRスカラ, SR擬似とも、16プロセッサ以下では、プロセッサ数 N_p の増加に伴い、計算時間は減少しているが、SRの32プロセッサでは逆に増加しているケースもある。

SP, SRスカラ, SR擬似とも、8プロセッサ以上では、自由度数Nの増加に伴い、速度向上比が増加している。より大規模なマトリックスに対しては、より高い速度向上比が期待できる。

SR擬似とSRスカラとの計算時間の差も、前進／後退代入と同様な傾向を示している。

表4.14 線形方程式の求解ルーチン全体の計算時間 (SP、単位: sec)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	0.471	0.393	0.207	0.123	0.111
19200	1.045	0.789	0.400	0.218	0.152
28800	1.553	1.189	0.599	0.314	0.197

表4.15 線形方程式の求解ルーチン全体の計算時間 (SRスカラ、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	0.81	0.66	0.32	0.24	0.21	0.30
19200	1.68	1.53	0.71	0.44	0.36	0.34
28800	2.53	2.36	1.10	0.56	0.40	0.37

表4.16 線形方程式の求解ルーチン全体の計算時間 (SR擬似、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	0.63	0.45	0.25	0.18	0.22	0.29
19200	1.27	1.01	0.51	0.31	0.28	0.33
28800	2.06	1.66	0.73	0.43	0.31	0.35

表4.17 線形方程式の求解ルーチン全体の速度向上比 (SP)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	1.000	1.198	2.275	3.829	4.243
19200	1.000	1.324	2.613	4.794	6.875
28800	1.000	1.306	2.593	4.946	7.883

表4.18 線形方程式の求解ルーチン全体の速度向上比 (SRスカラ)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.000	1.227	2.531	3.375	3.857	2.700
19200	1.000	1.098	2.366	3.818	4.667	4.941
28800	1.000	1.072	2.300	4.518	6.325	6.838

表4.19 線形方程式の求解ルーチン全体の速度向上比 (SR擬似)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.000	1.400	2.520	3.500	2.864	2.172
19200	1.000	1.257	2.490	4.097	4.536	3.848
28800	1.000	1.241	2.822	4.791	6.645	5.886

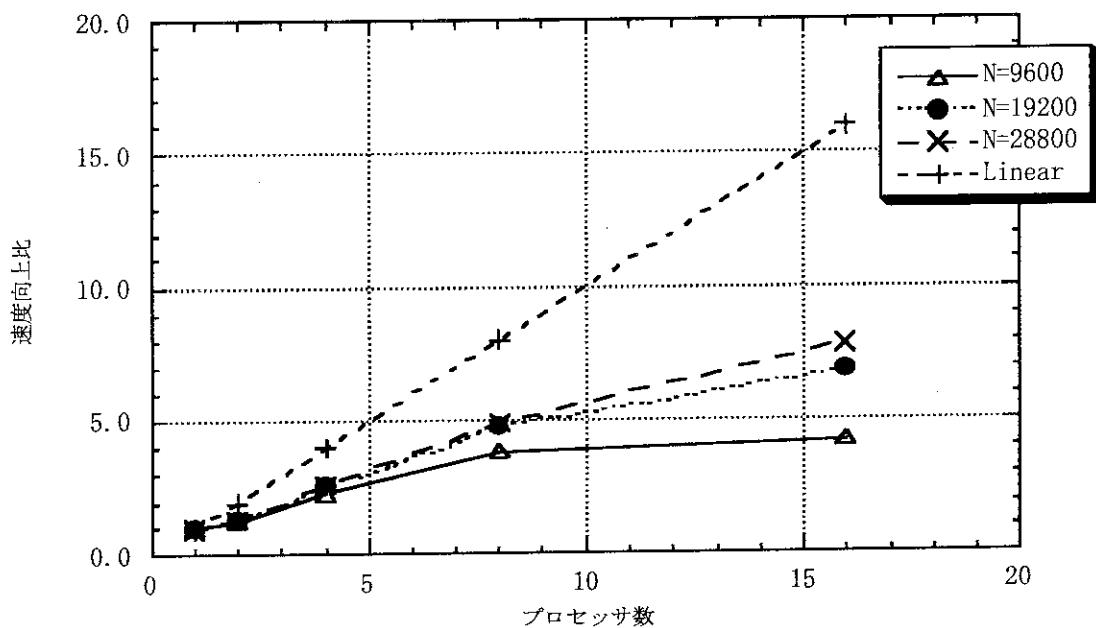


図4.8 線形方程式の求解ルーチン全体の速度向上比 (SP)

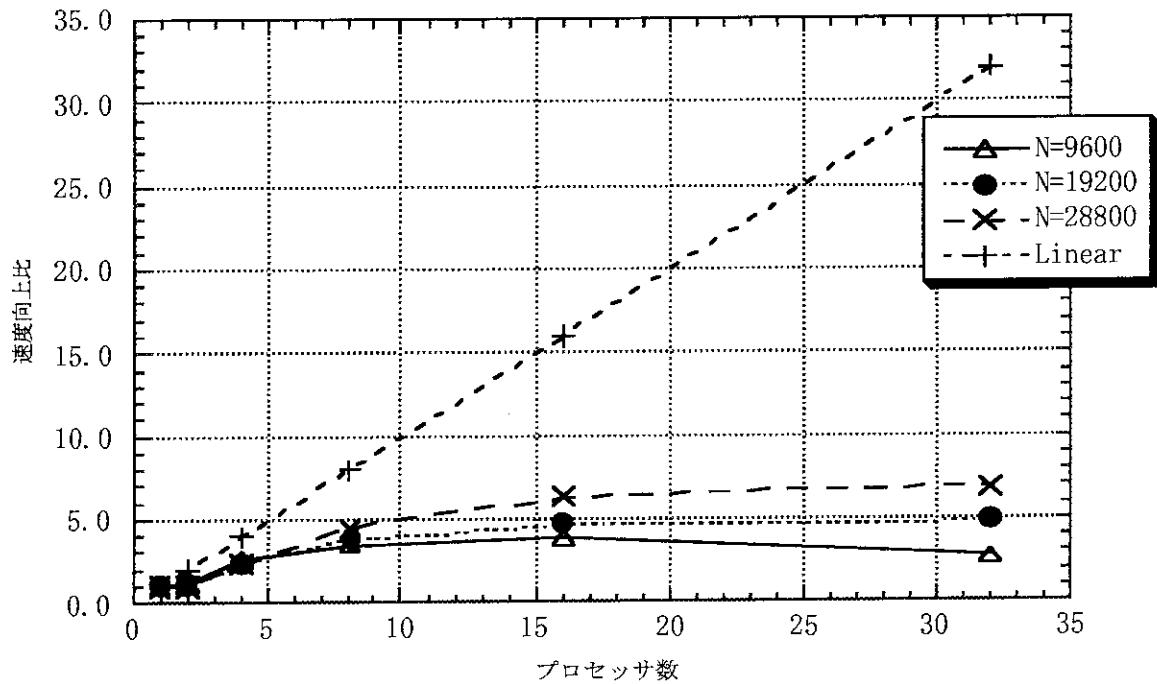


図4.9 線形方程式の求解ルーチン全体の速度向上比 (SRスカラ)

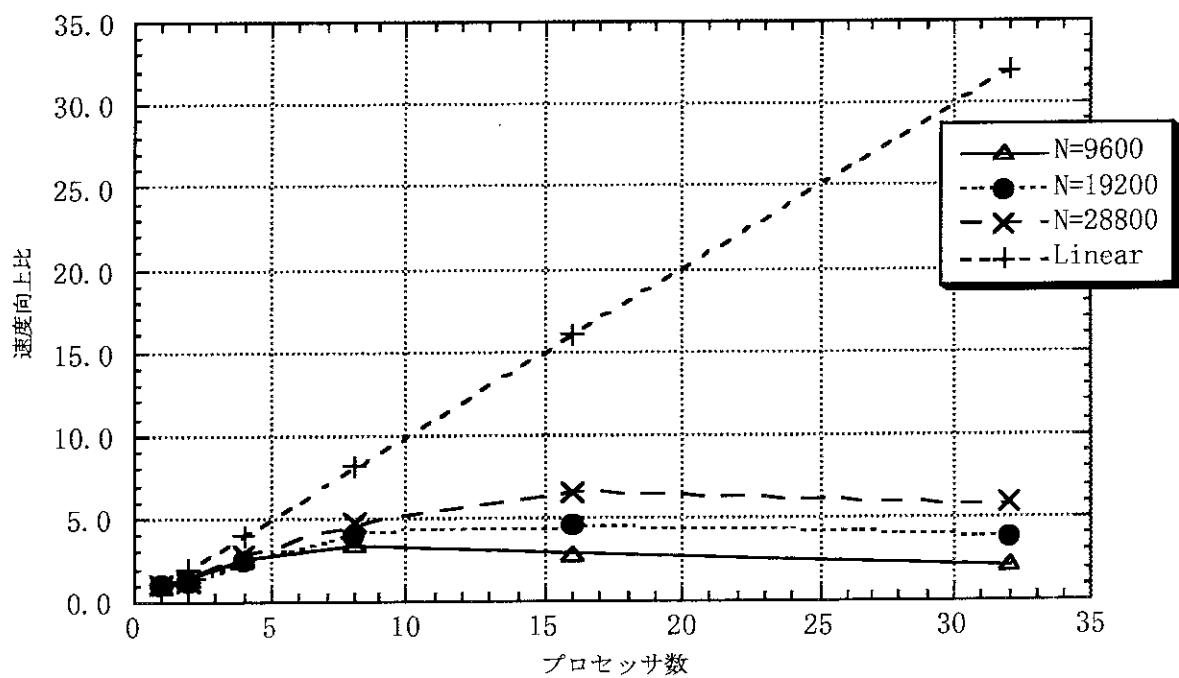


図4.10 線形方程式の求解ルーチン全体の速度向上比 (SR擬似)

4.3.4 EXPANDアルゴリズム

ここでは、ベクトルデータの共有化に用いたEXPANDアルゴリズム³⁾による通信時間の性能評価を行う。

EXPANDアルゴリズムは、二進木 (binary tree) ベースのアルゴリズムであり、全プロセッサが分割データを共有するグローバル通信としては、最適である。通信回数は $\log_2 N_p$ で、各通信 i ($1 \leq i \leq \log_2 N_p$) における通信サイズは $N/2^i$ で、トータルの通信サイズは $(N_p - 1)N/N_p$ である。

各プロセッサ数に対する、アルゴリズムの通信時間を表4.20および4.21に、そのグラフを図4.11および4.12に示す。

2プロセッサ (通信回数1回)においては、両者とも、通信時間は自由度数 N にほぼ比例している。プロセッサ数 N_p の増加に伴い、 N に対する通信時間の増加率は徐々に小さくなっている。これは、通信 i ($1 \leq i \leq \log_2 N_p$) の増加に伴い、通信サイズがより小さくなっていくのに対し、通信回数の増加に伴う通信の立ち上げに要する時間 (N に無関係) の比率が大きくなっていくためである。

表4.20 EXPANDアルゴリズムの通信時間 (SP、単位: sec)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	0.000	0.098	0.161	0.198	0.220
19200	0.000	0.183	0.287	0.350	0.389
28800	0.000	0.268	0.415	0.501	0.557

表4.21 EXPANDアルゴリズムの通信時間 (SR、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	0.00	0.03	0.07	0.13	0.20	0.22
19200	0.00	0.06	0.09	0.15	0.25	0.28
28800	0.00	0.08	0.13	0.19	0.30	0.33

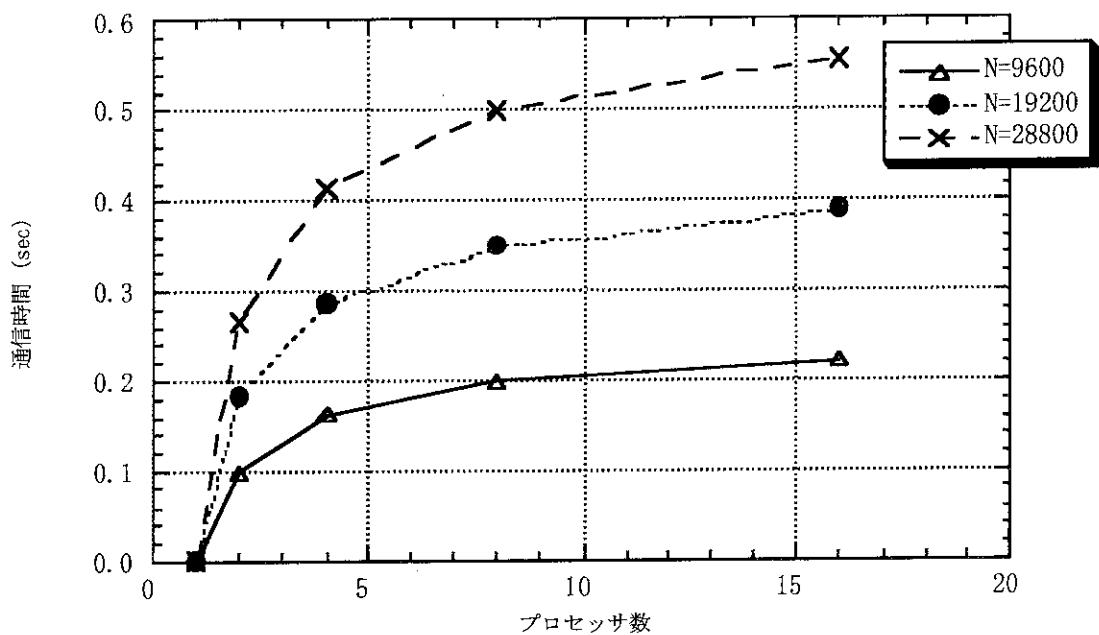


図4.11 EXPANDアルゴリズムの通信時間 (SP)

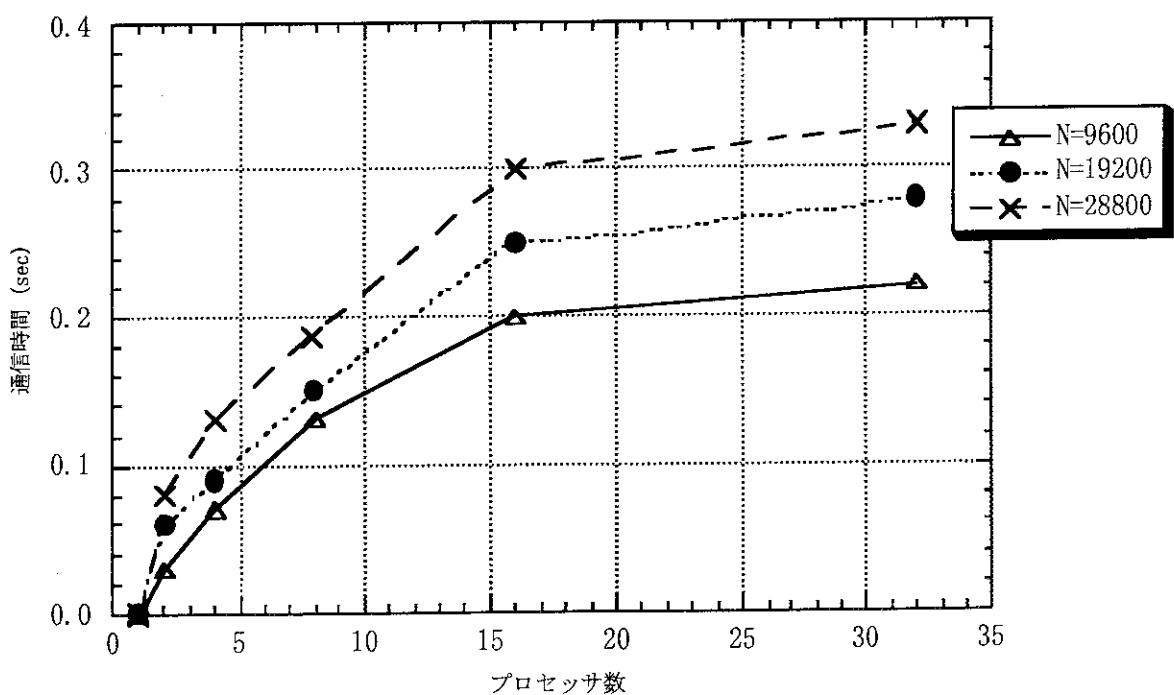


図4.12 EXPANDアルゴリズムの通信時間 (SR)

4.3.5 プログラム全体

ここでは、プログラム全体の性能評価を行う。

各プロセッサ数に対する、プログラム全体の並列計算時間を表4.22～4.24に、速度向上比を表4.25～4.27に示す。

各プロセッサ数に対する、プログラム全体の速度向上比のグラフを図4.13～4.15に示す。

SPでは8プロセッサ以下で、SRでは16プロセッサ以下で、プロセッサ数 N_p の増加に伴い、計算時間は減少しているが、それ以上では逆に増加しているケースもある。SPの16プロセッサにおいて計算時間の急激な増加が見られるのは、内積の部分和の総和に関するグローバル通信コストの急激な増加が原因と考えられる。

SP, SRスカラ, SR擬似とも、全プロセッサに対して、自由度数Nの増加に伴い、速度向上比も増加している。より大規模なマトリックスに対しては、より高い速度向上比が期待できる。

全ケースにおいて、SR擬似の方がSRスカラより速い。本並列化アルゴリズムは自由度分割によるもので、 N/N_p をループ長と見なせるが、SRスカラの計算時間とSR擬似の計算時間の比率、すなわち増加率は、 N/N_p の増加に伴い、ほぼ単調に増加している。

$N=28800$ のケースにおける8プロセッサによる速度向上比は、SPが約2.25倍、SR擬似が約3.6倍に抑えられているが、ブロックランチヨス法自体の絶対的処理速度が他の手法に比べて速く、通信コストが見えやすい上に、EXPANDアルゴリズム、内積の総和の共有化など全プロセッサが参加する通信が多いのが原因である。各メーカーには、ハードウェアとしての通信性能の向上はもとより、MPIの通信性能の（メーカー固有の通信ライブラリ並みの）向上を強く期待する。EXPANDアルゴリズムをサブルーチンとして用意し、並列プログラムの開発者の負担の軽減に努めることも必要である。

表4.22 プログラム全体の計算時間 (SP、単位: sec)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	1.185	0.915	0.753	0.704	1.655
19200	2.403	1.786	1.314	1.162	2.072
28800	3.620	2.680	1.888	1.614	2.592

表4.23 プログラム全体の計算時間 (SRスカラ、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	2.07	1.40	0.82	0.61	0.59	0.63
19200	4.49	3.03	1.65	1.13	0.93	0.92
28800	6.85	4.72	2.48	1.57	1.22	1.18

表4.24 プログラム全体の計算時間 (SR擬似、単位: sec)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.38	1.00	0.64	0.51	0.50	0.58
19200	2.81	1.94	1.18	0.86	0.77	0.79
28800	4.31	3.01	1.65	1.19	0.93	0.98

表4.25 プログラム全体の速度向上比 (SP)

自由度数N	プロセッサ数P				
	1	2	4	8	16
9600	1.000	1.295	1.574	1.683	0.716
19200	1.000	1.345	1.829	2.068	1.160
28800	1.000	1.351	1.917	2.243	1.397

表4.26 プログラム全体の速度向上比 (SRスカラ)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.000	1.479	2.524	3.393	3.508	3.286
19200	1.000	1.482	2.721	3.973	4.828	4.880
28800	1.000	1.451	2.762	4.363	5.615	5.805

表4.27 プログラム全体の速度向上比 (SR擬似)

自由度数N	プロセッサ数P					
	1	2	4	8	16	32
9600	1.000	1.380	2.156	2.706	2.760	2.379
19200	1.000	1.448	2.381	3.267	3.649	3.557
28800	1.000	1.432	2.612	3.622	4.634	4.398

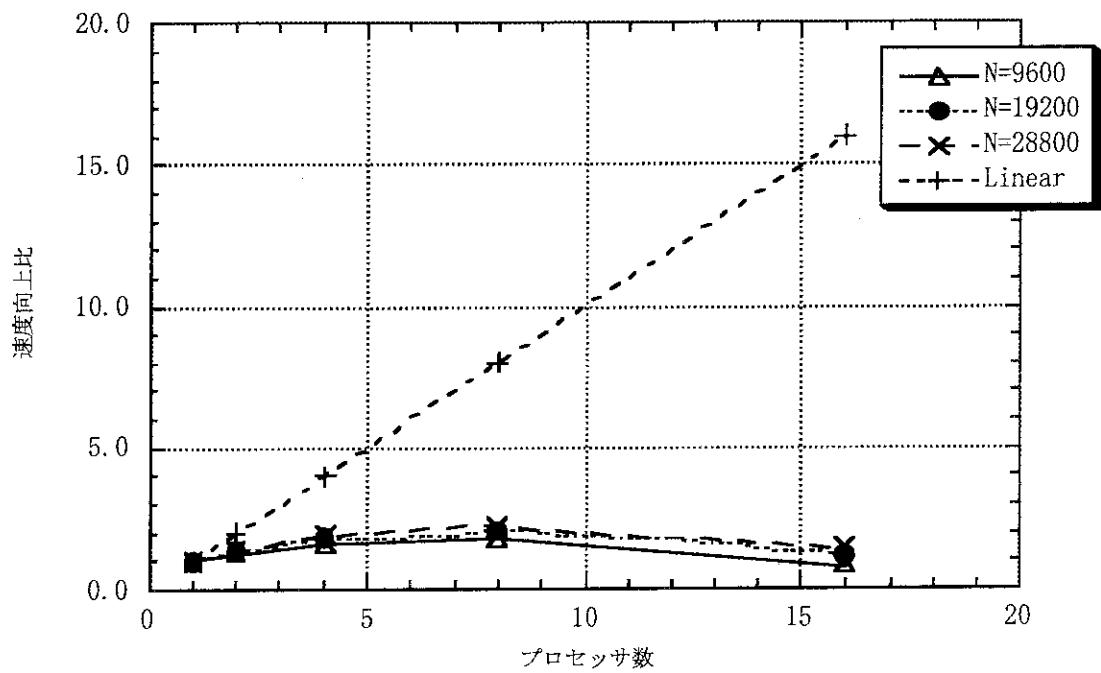


図4.13 プログラム全体の速度向上比 (SP)

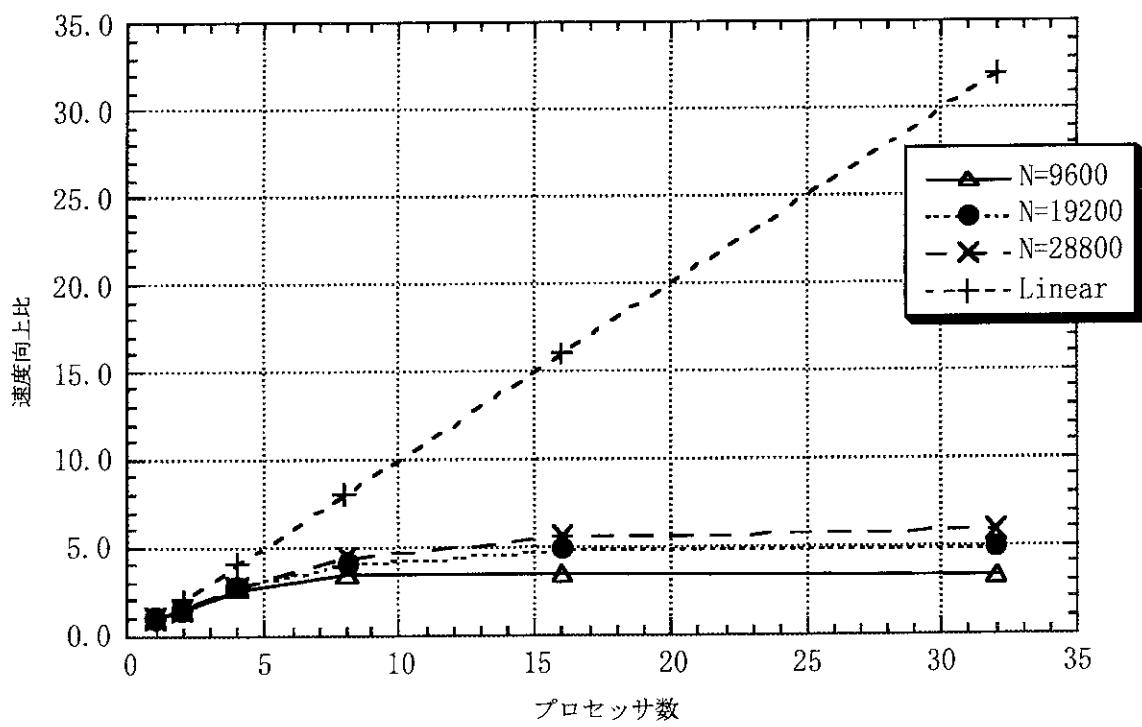


図4.14 プログラム全体の速度向上比 (SRスカラ)

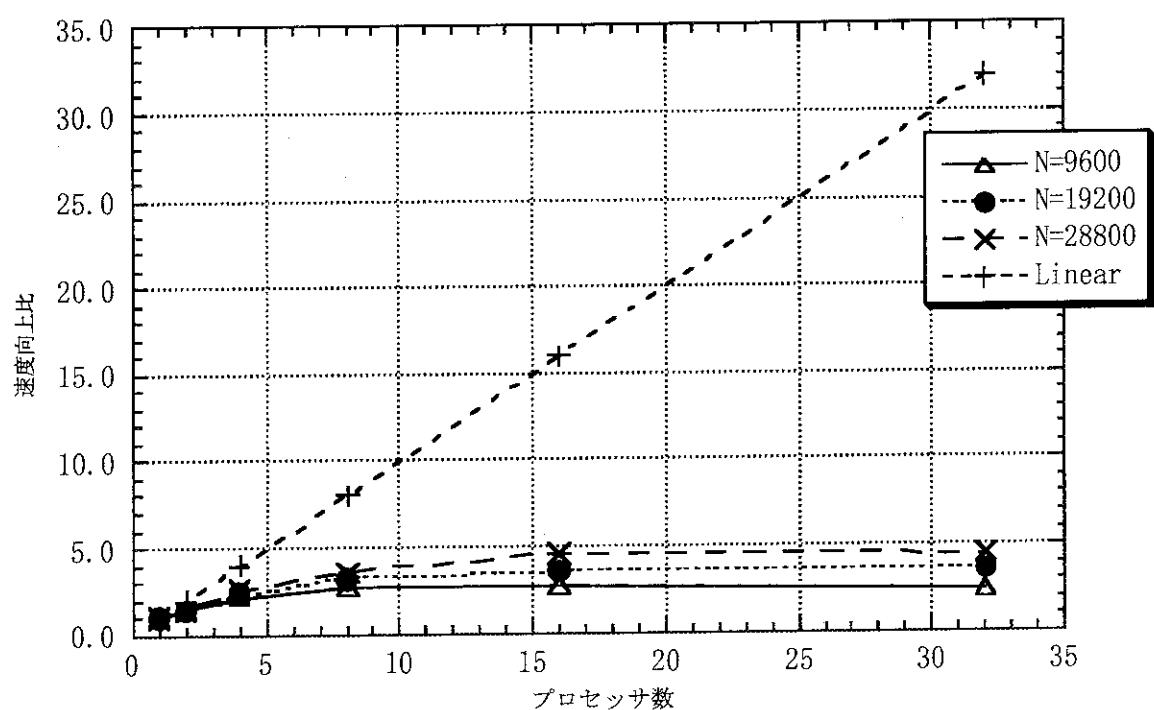


図4.15 プログラム全体の速度向上比 (SR擬似)

5. 結言

本研究では、実対称帶行列に対する一般固有値問題の並列計算ライブラリの研究開発を行った。数値解法としては、他の手法に比べて、大規模問題に対する1プロセッサの処理速度が非常に速いブロックランチヨス法を採用し、自由度分割による並列化手法を検討した。自由度分割については、多重スカイライン法を基本アイデアとして、数値計算ライブラリ用に、ユーザが幾何的データからsequence generatorを生成する手間を必要としない簡便性および汎用性を有している一次元分割による並列化手法を検討した。

プログラム全体、線形方程式の求解ルーチン、およびEXPANDアルゴリズムに対して、その並列化性能評価を、IBM SPおよび日立SR2201上で行った。日立SR2201については、擬似ベクトル機構の効果も評価した。プログラム全体で、各並列計算機において、自由度数Nの増加に対する速度向上比の単調増加性が見られた。

今後は、ブロックサイズに対する計算コスト／メモリコストの関係をサーベイした後、ブロック分割および、ブロック分割と自由度分割とを組み合わせたブロック自由度分割による並列化についても検討し、その並列化性能評価を行う予定である。ベクトル並列化版の開発についても、来年度以降に予定している。

謝辞

まず、日本IBMの寒川光氏には、入手し難い貴重な論文の提供など、大変御世話になつたことに感謝します。日本原子力研究所計算科学技術推進センターの木村俊哉氏および村松一弘氏には、並列計算機の使用方法などに関する報告者の質問に親切に答えていただいたことに感謝します。株式会社三菱総合研究所の落合孝正主任研究員には、計算例題のデータを提供して下さったことに感謝します。最後に、計算科学技術推進センターの薫木英雄グループリーダーおよび横川三津夫氏には、出向期間中終始、報告者の研究活動を支えていただいたことに、特に深く感謝します。

5. 結言

本研究では、実対称帶行列に対する一般固有値問題の並列計算ライブラリの研究開発を行った。数値解法としては、他の手法に比べて、大規模問題に対する1プロセッサの処理速度が非常に速いブロックランチヨス法を採用し、自由度分割による並列化手法を検討した。自由度分割については、多重スカイライン法を基本アイデアとして、数値計算ライブラリ用に、ユーザが幾何的データからsequence generatorを生成する手間を必要としない簡便性および汎用性を有している一次元分割による並列化手法を検討した。

プログラム全体、線形方程式の求解ルーチン、およびEXPANDアルゴリズムに対して、その並列化性能評価を、IBM SPおよび日立SR2201上で行った。日立SR2201については、擬似ベクトル機構の効果も評価した。プログラム全体で、各並列計算機において、自由度数Nの増加に対する速度向上比の単調増加性が見られた。

今後は、ブロックサイズに対する計算コスト／メモリコストの関係をサーベイした後、ブロック分割および、ブロック分割と自由度分割とを組み合わせたブロック自由度分割による並列化についても検討し、その並列化性能評価を行う予定である。ベクトル並列化版の開発についても、来年度以降に予定している。

謝辞

まず、日本IBMの寒川光氏には、入手し難い貴重な論文の提供など、大変御世話になつたことに感謝します。日本原子力研究所計算科学技術推進センターの木村俊哉氏および村松一弘氏には、並列計算機の使用方法などに関する報告者の質問に親切に答えていただいたことに感謝します。株式会社三菱総合研究所の落合孝正主任研究員には、計算例題のデータを提供して下さったことに感謝します。最後に、計算科学技術推進センターの蕪木英雄グループリーダーおよび横川三津夫氏には、出向期間中終始、報告者の研究活動を支えていただいたことに、特に深く感謝します。

参考文献

- 1) MSC社：“MSC/NASTRAN Ver. 68 Numerical Methods User' Guide”.
- 2) 寒川光：「分散メモリ並列計算機のための多重スカイライン法」，日本情報処理学会ハイパフォーマンスコンピューティング 研究報告 No.63 数値計算アルゴリズム分科会（1996）.
- 3) 田中靖久、横川三津夫、蕪木英雄：「短距離力分子動力学法のベクトル並列化手法の性能評価」，JAERI-Data/Code 96-011 (1996) .