数理解析研究所講究録
第1930巻 2015年 51-59

行列Horner法の並列化の実装について

田島 慎一*
SHIN-ICHI TAJIMA
筑波大学 数理物質系
FACULTY OF PURE AND APPLIED SCIENCES, UNIVERSITY OF TSUKUBA

小原 功任†
KATSUYOSHI OHARA
金沢大学 理工研究域
FACULTY OF MATHEMATICS AND PHYSICS, KANAZAWA UNIVERSITY

照井 章‡
AKIRA TERUI
筑波大学 数理物質系
FACULTY OF PURE AND APPLIED SCIENCES, UNIVERSITY OF TSUKUBA

Abstract

本稿では，行列多項式に対するHorner法（行列Horner法）の並列化の実装について述べる．我々は，行列Horner法の主要部分の計算量を見積もった上で，行列積の演算を並列化することにし，2種類の異なる並列化法を試みた．実験の結果，我々が行った実装の範囲では，CPU（中央処理装置）のコア数に合わせて行列を行ブロックに分割し，行列の乗算を並列化することにより，並列化の効果が最も高まったことを示す．

1 はじめに

本稿では，主に整数を要素とする n 次正方行列 A, n 次正方行列 M もしくは n 次列ベクトル $\boldsymbol{v}$, 整係数 m 次1変数多項式 $g(\lambda)$ が与えられたときに，行列 $g(A)M$ もしくは列ベクトル $g(A)\boldsymbol{v}$ を効率的に計算する方法について論ずる．

我々は，これまで，レゾルベントの留数解析に基づいて，与えられた行列に対し，スペクトル分解 [12], 最小消去多項式候補・最小消去多項式やこれらを用いた最小多項式の計算 ([14], [15]), 固有ベクトルの計算 [10] などを効率的に行う算法を提案しているが，多項式 $g(\lambda)$ の変数 λ に行列を代入した評価，すなわち行列多項式の評価は，これらの算法において中心的に用いられる計算の一つであり，行列多項式の評価を効率化することは，算法全体の効率化において重要である．

*tajima@math.tsukuba.ac.jp
†ohara@air.s.kanazawa-u.ac.jp
‡terui@math.tsukuba.ac.jp

我々は，1変数多項式に数値を代入して評価する効率的算法として古くから知られている Horner 法を行列多項式に用いる算法，すなわち行列 Horner 法を扱ってきたが，これまでに，逐次計算の下で行列 Horner 法を拡張し，効率化する算法の提案を行った [13]．我々が提案した算法は，Horner 法をある次数（“分割次数”と呼ぶ）で分割して行列の乗算回数を抑えることにより，算法全体の効率化を図るものである．我々が提案した算法を実装して計算機実験を行った結果，適切な分割次数のもとで，計算時間のみならず，メモリ使用量も減少した．さらに，多倍長整数上の行列 Horner 法のみならず，固定精度（IEEE 倍精度）の浮動小数上の行列 Horner 法でも，算法の効率化による効果が得られた．

しかしながら，行列の成分や多項式の係数が多倍長整数で，かつ行列の次元や多項式の次数が数百次程度に達すると，逐次計算の下では，計算時間の増加が顕著になる．一方で，昨今では，安価に入手可能なパソコン等においても，CPU（中央処理装置）が，マルチ CPU やマルチコアといったように複数の演算器を搭載し，並列計算が可能な計算機環境が普及している．

そこで，本稿では，行列 Horner 法の並列化の実装を試みる．Horner 法 [6, Sec. 4.6.4] は，もともと逐次的な算法として知られているが，計算機が発達する中で，Horner 法を並列化することにより，複数の演算装置を用いて計算を並列化することによる効率化が提案されてきた ([3], [4], [7], [8], [16])．しかしながら，行列 Horner 法は，これまで Horner 法が主な対象としていたように，数値を変数に代入するのではなく，行列を変数に代入するので，我々が目指す並列化は先行研究の対象とはやや異なる．

本稿では，行列どうしの乗算法として古典的な算法（詳細は第 3.1 節を参照）を用いる仮定の下，行列 Horner 法の計算量を見積もった上で，行列積の演算の並列化が最も並列化の効果が期待できると判断し，2種類の異なる並列化法を試みた．並列計算の実装には，数式処理システム Risa/Asir と，小原 [9] による並列計算フレームワーク `oh_p` を用いた．実験の結果，我々が行った実装の範囲では，CPU（中央処理装置）のコア数に合わせて行列を行ブロックに分割し，行列の乗算を並列化することにより，並列化の効果が最も高まったことを示す．

以下，本稿の構成は次の通りである．第 2 章では，行列 Horner 法の効率化を復習する．第 3 章では，並列化の実装箇所と実装方法について検討する．第 4 章では，計算機実験により，いくつかの異なる並列化の実装方法による効果の違いを検証する．

2 行列 Horner 法の効率化

本章では，我々がこれまでに行った，行列 Horner 法の効率化について，その概要を説明する（詳細は我々による別稿 [13] を参照されたい）．

A, M を体 K 上の n 次正方行列，$\boldsymbol{v}$ を K 上 n 次列ベクトルとし，$g(\lambda)$ を K 上 m 次 1 変数多項式とする．我々が取り上げるのは，$g(A)M$ もしくは $g(A)\boldsymbol{v}$ の Horner 法による計算（行列 Horner 法）である．

行列 Horner 法における効率化のアイデアは，Horner 法を分割して計算することにより，行列どうしの乗算回数を抑えることである．一般の場合，n 次正方行列 A, M および m 次多項式 $g(\lambda) = a_m\lambda^m + a_{m-1}\lambda^{m-1} + \cdots + a_1\lambda + a_0$ に対し，$g(A)M$ の計算手順は，$d = 2^b$ 次（ただし $d < m$）ごとに分割した Horner 法により，以下のアルゴリズム 1 にまとめられる．

アルゴリズム 1 (効率化した行列 Horner 法)
[Step 1] あらかじめ $A^{d-1}M, A^{d-2}M, \ldots, AM$ を計算し，これらに M を加えた d 個の行列を用意しておく．

[Step 2] $A^d = A^{2^b} = (\cdots(A^2)^2\cdots)^2$ も，あらかじめ計算して用意しておく．

[Step 3] Horner 法を以下の通り d 次ごとに分割して加える.

$$\begin{aligned} g(A)M = A^d\{\cdots\{A^d(a_mA^rM+\cdots+a_{qd+1}AM+a_{qd}M)\} \\ +(a_{qd-1}A^{d-1}M+\cdots+a_{(q-1)d+1}AM+a_{(q-1)d}M)\} \\ +\cdots \\ +(a_{d-1}A^{d-1}M+\cdots+a_1AM+a_0M), \end{aligned} \tag{1}$$

ここに q および r はそれぞれ m を d で割った商および剰余を表す. ∎

アルゴリズム 1 の各ステップに現われる行列どうしの乗算回数を $T(b,d,m)$ で表すと, Step 1 が $d-1$ 回, Step 2 が b 回, Step 3 が $\lfloor m/d \rfloor$ 回より

$$T(b,d,m) = b+d+\lfloor m/d \rfloor - 1$$

となる. $d=2^b$ を用いると

$$T(b,m) = b+2^b+\lfloor m/2^b \rfloor - 1 \tag{2}$$

と表される.

式 (2) の $T(b,m)$ が最も小さくなるような b の値は, 以下のように見積もることができる. $T(b,m)$ の大きさに影響を与える項は 2^b および $\lfloor m/2^b \rfloor$ である. 相加・相乗平均の関係より, これら 2 つの項が等しいときに $T(b,m)$ が最小値をとる. 実際には, 2^b は自然数である一方, $\lfloor m/2^b \rfloor$ は一般に自然数とは限らないので, 与えられた m に対し, これら 2 つの項が等しくなるような b が常に存在するとは限らないが, 2^b と $\lfloor m/2^b \rfloor$ の差が最も小さくなるような b の値を考えると, 2^b が $\sqrt{m}$ 程度の大きさのときに $T(b,m)$ の大きさが最も小さくなると見積もられる.

3 行列 Horner 法の並列化

3.1 並列化の実装箇所の検討

前章の行列 Horner 法の計算量の見積もりに基づき, 我々はまず, 行列 Horner 法の計算のどの部分を並列化させるかの検討を行う. なお, 行列どうしの乗算法として, 本稿では n 次正方行列に対する時間計算量が $O(n^3)$ [5, Chap. 12] の古典的算法を用いる点に注意する (より効率的な算法については Bürgisser *et al.* [2, Chap. 15], von zur Gathen and Gerhard [5, Chap. 12] 等を参照). また, 以下で行う計算量の見積もりは, K の元(数) の四則演算の回数 [1] の見積もりである点に注意する.

アルゴリズム 1 の各ステップの中で, 本来の Horner 法の計算を行っている部分は Step 3 であるので, まず Step 3 を並列化の対象とする. Step 3 は, さらに以下の部分ステップの計算に分けることができる.

[Step 3a] $j=1,\ldots,q+1$ に対し, R_j を以下の通り計算する.

$$\begin{aligned} R_j &= a_{jd-1}A^{d-1}M+\cdots+a_{(j-1)d+1}AM+a_{(j-1)d}M, \quad j=1,\ldots,q, \\ R_{q+1} &= a_mA^rM+\cdots+a_{qd+1}AM+a_{qd}M. \end{aligned} \tag{3}$$

[Step 3b] 式 (3) を用いて, $g(A)M$ を以下の Horner 法で計算する.

$$g(A)M = A^d\{\cdots\{A^d\{A^dR_{q+1}+R_q\}+R_{q-1}\}+\cdots\}+R_1. \tag{4}$$

ここで，上記の Step 3a, Step 3b の計算量を見積もる．まず，Step 3a の計算量は以下の通り 見積られる．各 R_j の計算において，$A^{d-1}M,\ldots,AM$ は Step 1 であらかじめ計算されて用意されているので，R_j の計算は行列-スカラ積の和の計算であり，$O(dn^2)$, これを $O(q)$ 個計算するので，Step 3a の計算量は $O(qdn^2)$ と見積もられる．前章の議論より，行列積の回数を最小にする q や d の値は $q \simeq d \simeq \sqrt{m}$ であるので，この場合の Step 3a の計算量は $O(mn^2)$ と見積られる．

一方，Step 3b の計算量は以下の通り 見積もられる．A^d は Step 2 であらかじめ計算されて用意されており，本ステップで行われる計算は $A^dR_{j+1}+R_j$ $(j=q,\ldots,1)$ の計算の繰り返しである．この中で計算量を支配するのは行列積 A^dR_{j+1} であり，その計算量は $O(n^3)$ と見積もられる．これを q 回繰り返すので，Step 3b の計算量は $O(qn^3)$ と見積もられる．前章の議論より，行列積の回数を最小にする q や d の値は $q \simeq d \simeq \sqrt{m}$ であるので，この場合の Step 3b の計算量は $O(\sqrt{m}n^3)$ と見積もられる．

以上を比較し，本稿では $\sqrt{m}n^3 > mn^2$, すなわち

$$n > \sqrt{m} \tag{5}$$

を仮定した [1] 上で，Step 3b に現われる行列-行列の乗算を並列化することにした．

3.2 並列化の実装方法と手順

我々は，本稿において，上記の並列化を実装する環境として，我々がこれまで行列 Horner 法の実装を行ってきた数式処理システム Risa/Asir を用いる．さらに Risa/Asir 上の並列計算実装環境として，小原 [9] による並列計算フレームワーク `oh_p` を用いる．`oh_p` は以下の特徴をもつ(詳細は小原 [9] を参照)．

1. OpenXM プロトコル上に Asir (言語) で実装された Asir (言語) 用のソフトウェアパッケージである.(OpenXM は，同一もしくは異なる計算機上のプロセス間で，数式処理をはじめとする数学情報を扱い，情報のやりとりや計算の制御を行うための手順(プロトコル) である.)
2. OpenXM がプロセス単位での並列計算に対応することから，`oh_p` もプロセス単位での並列計算を行う．
3. `oh_p` における並列計算の単位は，1 個の client と l 個の server により構成される．
4. 並列計算の際は，client からジョブの集合の要素を各 server に投入して計算させ，計算結果を client に集める．ジョブの各要素間の依存性にも対応した処理が可能である．

次に，並列化の手順を述べる．A, B を n 次実正方行列とし，A を適当に分割することにより，積 AB の計算を並列化する手順として，本稿では以下の異なる 2 種類の並列化を考える．

[並列化 1] A を行ベクトルに分割する．$A={}^t(\boldsymbol{a}_1,\ldots,\boldsymbol{a}_n), \boldsymbol{a}_j \in \mathbb{R}^n\ (j=1,\ldots,n)$ とし，$AB={}^t({}^tB\boldsymbol{a}_1,\ldots,{}^tB\boldsymbol{a}_n)$ により AB を求める．${}^tB\boldsymbol{a}_j$ の計算を各 server に割り振ることで並列化する．A の列数が server の個数を上回るときは，${}^tB\boldsymbol{a}_1$ から順次(計算が終わって) 空いている server に計算を割り振る．

[並列化 2] A を行ブロックに分割する．n を l で割った商を q, 剰余を r とするとき

$$A={}^t\begin{pmatrix} A_1 & A_2 & \cdots & A_{l-1} & A_l \end{pmatrix},\quad A_j \in \begin{cases} \mathbb{R}^{n\times q} & j=1,\ldots,l-1 \\ \mathbb{R}^{n\times r} & j=l \end{cases}$$

[1] 我々が本算法の適用範囲として想定している多項式は，行列の特性多項式や，それを割り切るような多項式(特性多項式の既約因子の積) であるため，$n \geq m$ をみたすとしてよい．その点において，式 (5) の仮定は妥当なものである．

(すなわち，A を l 行毎の行ブロックに分割し，最後のブロックを A の下から r 行からなるブロックとする) とし，

$$AB = {}^t\left({}^tBA_1 \quad {}^tBA_2 \quad \cdots \quad {}^tBA_{l-1} \quad {}^tBA_l\right)$$

により AB を求める．${}^tBA_j$ の計算を各 server に割り振ることで並列化する．A のブロック分割数が server の個数を上回るときは，${}^tBA_1$ から順次(計算が終わって) 空いている server に計算を割り振る．

4 実験

本章では，前章に述べた行列 Horner 法の並列化の実装による効果を確かめるために行った実験の結果を示す．

各算法の実装は数式処理システム Risa/Asir 上で行い，実験を行った．実験環境は以下の通り：Intel(R) Xeon(R) E5607 (4 cores) ×2 at 2.27 GHz, RAM 64GB, Linux 2.6.32-5-amd64 (SMP).

本章の各実験は，以下の問題に対して行った．

- 行列 A, M は 64 次正方行列 ($n = 64$) とし，各要素は長さ 64 ビットの整数で無作為に与えた．
- 多項式 $g(\lambda)$ の次数は 64 次 ($m = 64$) とし，係数は長さ 64 ビットの整数で無作為に与えた．
- 以上の A, M, $g(\lambda)$ に対し，$g(A)M$ を行列 Horner 法で計算し，計算時間を測定した．
  - Horner 法の分割次数 d は，我々の先行研究の実験結果から，上で与えた問題に対する最適値として 8 に設定した．
  - 行列 A として異なるものを 10 個用意し，$g(A)M$ の計算を，各分割数あたり 1 回ずつ，計 10 回行い，計算時間とメモリ使用量の平均値を計算した．これらの計算において，行列 M は同じものを用いた．

以上の要領で与えた問題に対し，各実験では oh_p において起動する server の個数(すなわち並列度) を 1 (逐次処理), 2, 4, 8 と変化させ，それぞれの場合の計算時間を比較した．

そして，行列どうしの乗算に対し，前章に述べたように異なる手順による並列化を行い，それぞれの場合における計算効率の比較を行った．

4.1 実験 1: 行ベクトル単位の並列化の場合

本実験では，前章の [並列化 1] に従い，行列の乗算を行ベクトル単位で並列化した．

実験結果を表 1 および図 1 に示す．表 1 において，“# Cores (CPU)” は起動した server の個数(用いた CPU のコア数) を表す．“Time (preprocessing)”, “Time (Horner)” および “Time (total)” は，それぞれ，Horner 法の前処理(アルゴリズム 1 の [Step 1] および [Step 2]) の時間，行列 Horner 法(アルゴリズム 1 の [Step 3]) の時間，および両者の合計を秒単位で，小数第 3 位で四捨五入して表したものである．図 1 は表 1 を棒グラフで表したものである．x 軸は計算時間(秒) を表し，y 軸は用いた CPU のコア数を表す．グラフのうち，網かけの部分および塗りつぶしの部分は，それぞれ表 1 の “Time (preprocessing)”, “Time (Horner)” を表す．ここで，“Time (preprocessing)” の部分(棒グラフの斜線部) に対応する算法のステップは並列化を行っていないので，その計算時間は用いた CPU コアの個数に依存せず，ほぼ一定である点に注意する．

実験結果を見ると，用いるCPUのコア数がより大きくなるのに対し，“Time (Horner)”の部分（棒グラフの塗りつぶしの部分）の値がより小さくなることから，並列化の効果がある程度は現われていることがわかる．しかしながら，コア数が1から2，2から4，...と2倍になっても，行列Horner法の計算時間が半分になっているわけではない．すなわち，CPUの並列度に相当する計算の効率化が達成されていないことが見てとれる．

この理由を考察すると，本実装では行列Aを1行ずつ分割してserverに計算を割り振ることから，計算前後のclientとserver間のデータ転送がAの行数分（本実験では各64回）発生することによる通信コストが大きくなっていることが考えられる．

4.2 実験2: 行ブロック単位の並列化の場合

本実験では，並列処理における通信コストの削減を目指し，前章の[並列化2]に従い，行列の乗算を行ブロック単位で並列化した．なお，本実験においても，“Time (preprocessing)”の部分（棒グラフの斜線部）に対応する算法のステップは並列化を行っていないので，その計算時間は用いたCPUコアの個数に依存せず，ほぼ一定である点に注意する．

実験結果を表2および図2に示す．以下，表およびグラフの要領は実験1の場合と同様である．計算時間のうち，“Time (Horner)”で表される行列Horner法の計算時間は，実験1の場合と比較すると，よりCPUのコア数の増加に対して反比例に近い割合で減少している．この結果から，行列の乗算を行ブロック単位で並列化した実装は，並列化の効果がより大きいことがわかる．

4.3 実験3: 実験2に加えて，Horner法の前処理も並列化した場合

本実験では，前節の実験2で行った[並列化2]による[Step 3b] (4)の並列化に加え，アルゴリズム1の[Step 1]および[Step 2]の部分についても[並列化2]による並列化を行った．

実験結果を表3および図3に示す．“Time (Horner)”の部分（棒グラフの塗りつぶしの部分）の計算時間は，実験2の結果とほぼ同様である．これに対し，“Time (preprocessing)”の部分（棒グラフの斜線部）の計算時間が，実験2の結果に比べて減少していることに注意する．特に，CPUの並列度（コア数）が8コアの場合の計算時間は，実験3では実験1の計算時間の約半分まで抑えられている．

5 まとめ

本稿では，行列Horner法の並列化の実装方法について議論し，いくつかの実装方法について，計算機実験により，実際の効率を比較した．

行列Horner法の並列化にあたっては，計算量の見積もりから，行列の乗算を並列化することにし，その結果、計算機実験においてもその効果が確かめられた．

行列の乗算の並列化法においては，行列を1) 1行毎に分割する並列化，2) CPUのコア数に応じた行ブロックに分割する並列化，の2通りの実装を比較した結果，2)の行ブロックに分割する並列化がより効率的であるとの実験結果を得た．並列処理においては，計算の粒度（並列に実行させるタスクの大きさ）を適切に設定することが重要である[11, p. 399]とされるが，本稿の実装においては，より粒度が粗い2)の実装の方が並列化の効果がより大きいといえる．

今後は，本稿で比較した実装も用いながら，我々が行ってきた行列の最小消去多項式や固有ベクトルの算法に応用し，それぞれの算法の効率化を行っていきたいと考えている．

# Cores (CPU)	Time (preprocessing)	Time (Horner)	Time (total)
1	1.34	8.39	9.73
2	1.34	6.50	7.84
4	1.32	3.83	5.15
8	1.28	2.86	4.14

表 1: 実験 1 の計算時間. 詳細は第 4.1 章を参照.

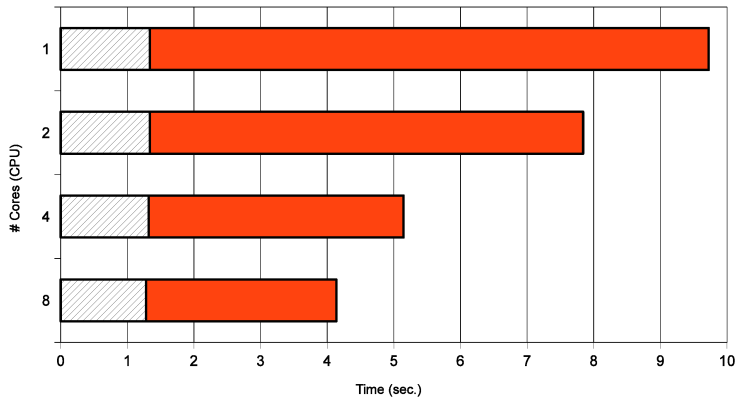

図 1: 表 1 のグラフ. 詳細は第 4.1 章を参照.

# Cores (CPU)	Time (preprocessing)	Time (Horner)	Time (total)
1	1.34	8.31	9.65
2	1.27	4.04	5.31
4	1.38	2.28	3.66
8	1.42	1.43	2.85

表 2: 実験 2 の計算時間. 詳細は第 4.2 章を参照.

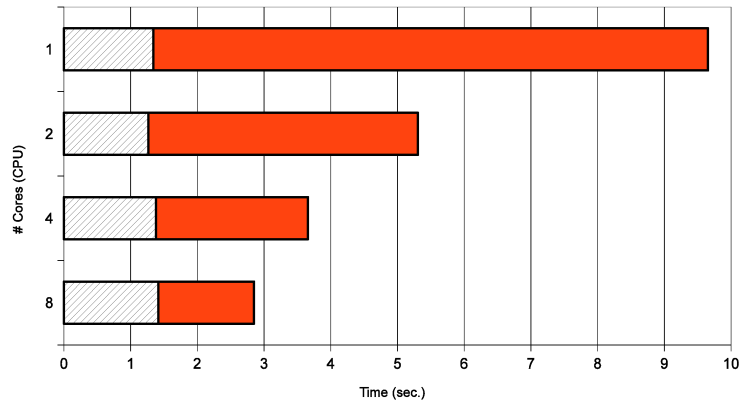

図 2: 表 2 のグラフ. 詳細は第 4.2 章を参照.

# Cores (CPU)	Time (preprocessing)	Time (Horner)	Time (total)
1	1.35	8.33	9.68
2	0.78	3.71	4.49
4	0.56	2.21	2.78
8	0.51	1.49	2.00

表 3: 実験 3 の計算時間. 詳細は第 4.3 章を参照.

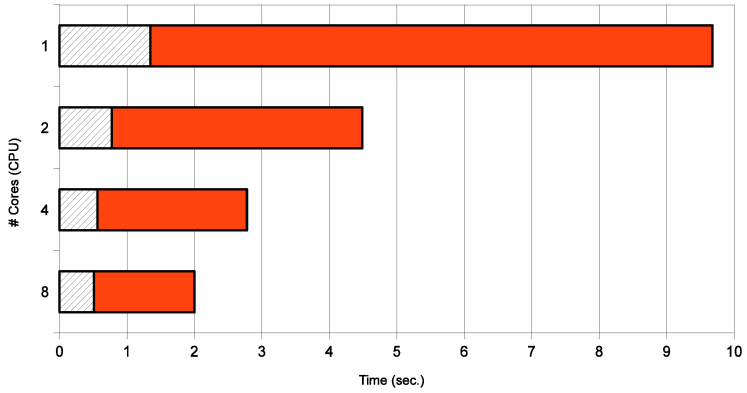

図 3: 表 3 のグラフ. 詳細は第 4.3 章を参照.

参 考 文 献

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1975. Second printing.

[2] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, Vol. 315 of *Grundlehren der Mathematischen Wissenschaften.* Springer-Verlag, Berlin, 1997.

[3] W. S. Dorn. Generalizations of Horner's Rule for Polynomial Evaluation. *IBM Journal of Research and Development*, Vol. 6, No. 2, pp. 239–245, April 1962.

[4] Michael L. Dowling. A Fast Parallel Horner Algorithm. *SIAM Journal on Computing*, Vol. 19, No. 1, pp. 133–142, February 1990.

[5] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, New York, NY, USA, Third edition, 2013.

[6] D. Knuth. *The Art of Computer Programming*, Vol. 2: Seminumerical Algorithms. Addison-Wesley, Third edition, 1998.

[7] Kiyoshi Maruyama. On the Parallel Evaluation of Polynomials. *IEEE Transactions on Computers*, Vol. C-22, No. 1, pp. 2–5, January 1973.

[8] Ian Munro and Michael Paterson. Optimal algorithms for parallel polynomial evaluation. *Journal of Computer and System Sciences*, Vol. 7, No. 2, pp. 189–198, April 1973.

[9] 小原功任. OpenXMを用いた Risa/Asir 並列計算フレームワークの開発. 数式処理, Vol. 18, No. 1, pp. 20–26, 2011.

[10] 照井章, 田島慎一. 行列の最小消去多項式候補を利用した固有ベクトル計算. In *Computer Algebra — Design of Algorithms, Implementations and Applications*, 数理解析研究所講究録, 第 1815 巻, pp. 13–22. 京都大学数理解析研究所, October 2012.

[11] 情報処理学会(編) . 情報処理ハンドブック , 第 3 編: 計算機アーキテクチャ , 第 4 章: 並列計算機. オーム社, 1989.

[12] 田島慎一. 微分作用素を用いたレゾルベントの留数解析と行列のスペクトル分解. In *Computer Algebra — Design of Algorithms, Implementations and Applications*, 数理解析研究所講究録, 第 1814 巻, pp. 17–28. 京都大学数理解析研究所, October 2012.

[13] 田島慎一, 小原功任, 照井章. 行列 Horner 法の拡張と効率化. 数式処理研究の新たな発展, 数理解析研究所講究録. 京都大学数理解析研究所, 投稿中.

[14] 田島慎一, 奈良洸平. 最小消去多項式候補とその応用. In *Computer Algebra — Design of Algorithms, Implementations and Applications*, 数理解析研究所講究録, 第 1815 巻, pp. 1–12. 京都大学数理解析研究所, October 2012.

[15] 田島慎一, 奈良洸平, 小原功任. 行列の最小多項式計算について. In *Computer Algebra — Design of Algorithms, Implementations and Applications*, 数理解析研究所講究録, 第 1814 巻, pp. 1–8. 京都大学数理解析研究所, October 2012.

[16] 丸山清, David J. Kuck. 行列の式の並列計算について. 情報処理, Vol. 15, No. 5, pp. 335–341, 1974.