記　号　処　理 62−3
(1991. 11. 29)

# ベクトル計算機による記号多項式計算

村尾 裕一

東京大学・大型計算機センター
〒113 東京都文京区弥生 2-11-16

murao@cc.u-tokyo.ac.jp

**概要**

数式処理における, 行列式等の多項式計算にベクトル計算機を利用する手法を示し, その広範な活用を提言する. その手法とは, 数式の特別な表現法を用いてベクトル化を行うのではなく, 適切な算法の選択により, 目的とする計算の全体にわたってベクトル処理を導入するものである. その算法とはモジュラー算法と多項式の補間法であり, これらにより, 主要部分における記号式の計算を全て数値による計算に置き換えることが可能となる. かくして, スーパーコンの持てる処理能力と資源が数式処理においても最大限に利用可能となり, 計算規模の大幅な拡大が達成される. また, インプリメントのための単純かつ有用な方法やベクトル化の技法についても言及する.

# Symbolic Polynomial Calculations on Vector Processor

Hirokazu MURAO

Computer Centre, University of Tokyo
Yayoi 2-11-16, Bunkyo-ku Tokyo 113, JAPAN

murao@cc.u-tokyo.ac.jp

**ABSTRACT**

A method to apply vector procesing to symbolic polynomial calculations is described. The key idea for vectorization is the both use of the modular algorithm and the polynomial interpolation algorithm. This combination of algorithms constructs an objective polynomial from its numerical values of modular numbers, and, thus, enables us to perform intermediate calculations only with word-sized integers. Consequently, highly effective vectorization can be introduced into polynomial calculations, and, as its effect, the computable size of formulas are to be largely expanded. Simple but useful techniques for implementation are also presented.

## 1. まえがき

記号処理の典型的な応用例の一つである数式処理において, 行列式の展開を初めとした多項式計算に, 高率かつ効果的にベクトル処理を適用するための方法を示す. 同時に, その方法を実現する上で必要となる, ベクトル化の技法とプログラムを構成するための単純だが有用な手法についても説明する. 本稿では, 現時点でインプリメントまで完了済みの多項式行列式の計算を主たる題材として, 説明を行うが, ここで用いる様々な手法やアイディアは行列式の計算に固有のものではなく, 多項式を求める計算においてはごく一般的なものであること及び, 計算の対象が行列であるが故にベクトル処理が可能となったわけではないという点に注意されたい. 本手法は今後様々な計算に応用されうることが期待され, 題材の行列式の展開はその応用のうちの一つにすぎないのである.

## 2. 記号計算とベクトル処理 — 動機

スーパーコンピュータが, 数値計算の分野において, その可能性を飛躍的に増大させ, 今日大規模な数値的解析のためには必要不可欠な手段となっていることには, 様々な応用例を挙げるまでもなく, 疑う余地がない. ひとくちにスーパーコンと言っても, ベクトル演算装置から超並列まで, 様々なアーキテクチャが存在しているが, 現時点では, 世界中に最も広く普及し, また, ソフトウェア開発の観点から一番安定していると思われるのは, Cray に代表されるベクトル演算の機能を持ったスーパーコンであろう. そのようなスーパーコンでは, giga に及ぶ数値演算処理能力のみならず, その処理能力を活かすために, giga に及ぶメモリを実装し高速な入出力機構を兼ね備えているのが普通である. これらの膨大な計算資源だけに着目すれば, 「遅い」「メモリ喰い」で鳴らしてきた, Lisp に代表される記号処理系にとっては, スーパーコンは極めて魅力的である. その応用例の古典的な代表格である, 数式処理においては尚更である.

数式処理は, その性質上, 科学技術計算の道具として, 数値計算と並び称されることが多いのだが, その計算可能な規模ではひどく見劣りがする. 実際, 数値計算においては, スーパーコンの登場以来, 1000 次元程度の行列を扱うことは当たり前になっているが, 数式処理では, 既存のシステムにおいては, 現在でも 10 次元程度の行列式を展開するのが限界である. これは, 一般には複雑な構造へのポインタにより表現される式の計算操作自体に多大な時間を要すること, 及びそれらの大きさが変化し多くの場合増大して, メモリ不足に陥ることが原因である. 言うまでもなく, このことは数式処理あるいは多くの場合その記述言語である Lisp における「数」にもあてはまる. また, 上述のデータ表現は構造および大きさ共に不均一であることを意味し, ベクトル処理には向かない. ベクトル処理においてはデータの均一性が重要である.

では, 如何なる計算において, また如何様にすれば, ベクトル処理を数式処理に持ち込むことが可能であろうか? 勿論, すぐに気付くように, 多倍長整数や多項式の演算は, 特定の表現形式の仮定の元にベクトル化されうる. しかし多くの場合, これらの微視的な方法では, 目的とする数式計算の全体に比べ, ほんの一部において局所的にベクトル処理が導入されるにすぎず, ベクトル計算機の能力を充分に引き出し活用するまでには到らない. 我々が期待し目的とするところは, ベクトル計算機の計算能力を最大限いかすことにより, 数値計算におけるように, 計算可能な式の規模を飛躍的に増大させることである. この目的は, 次節に述べる, 適切なる算法の選択利用により, ある種の多項式を求める計算において達成される.

## 3. 多項式計算のための基本算法

我々のアイディアの要は, 多項式のための以下の二種類の算法を結合させることである. そのベクトル化への効能は, 途中の主要部分の計算を全て, 機械語長の整数演算で行い得るようにすることである.

### 3.1. モジュラー算法

中国剰余定理 (Chinese remainder theorem) によれば, 「ある範囲内の整数値は, 充分に多くの法による剰余から, 唯一定められうる.」但し, その範囲は全ての法の積により定められる. これを整数係数の多項式に適用すれば, 目的とする多項式は, 充分に多くのそのモジュラー像の多項式から定められる. つまり, ある整数上の多項式を求めるには, 充分に多くのモジュラー像を計算すればよい. 例えば, 与えられた

行列の行列式の多項式を求めるには,行列式の展開を,各要素をそのモジュラー像で置き換えた行列に対して行えば良いことになる.法として一語長におさまる素数のみを選べば,途中の計算では多倍長整数の計算は不要になる.また,複数の法を用いる場合,法ごとの計算は明らかに並列処理が可能である.この並列性は, Knuth のように [6],多倍長整数の剰余数による表現とみなせば,係数内へと局所化でき,ベクトル演算可能である [11].しかも,その演算は極めて単純なベクトル演算となるため,多倍長整数演算のベクトル化よりも効率の良いベクトル化が可能である.本手法は,数係数膨張の著しい,ある種の応用には非常に有効であることが報告されているものの [11, 12],多項式を扱う場合には,依然として局所的であり,ベクトル処理自体は,多項式の構造等の処理操作により,頻繁に妨げられることになる.

## 3.2. 多項式の補間

未知の多項式を,その値から補間法により定めることは,基本的には,考えうる全ての単項式に対する数係数を未定係数として線形方程式系を解くことに他ならない.よって,そのように直接的に解くことが可能であれば,高度にベクトル化されうることは当然である.しかし,この方法は,多大なメモリを要するため,通常用いられず,より簡単化された, Newton や Lagrange による補間法が用いられる.これらの古典的な補間法も,必要となる値の個数から,疎な多変数の多項式に対しては実用性を欠く.そのような場合には,ごく最近開発された方法 [1, 4, 5, 8] が有用である.以下では,その概要を簡単に説明する.

### 3.2.1. 一変数の場合

一般に,求める多項式が一変数の場合には密 (dense) であると仮定して構わない.実際,密でなければ,適当な冪乗を独立な変数で置き換えれば,多変数でしかも疎な多項式を求めればよいから,次節の方法を適用すれば良い.

以下に,未知の多項式 $P(z)$ $(\deg(P(z)) \le d)$ を, Lagrange の方法により, $z = 0, 1, \ldots, d$ における $P(z)$ の値から補間する算法を示す.

$$u_0 = P(0), \quad u_k = \frac{P(k)}{k!} - \sum_{j=0}^{k-1} \frac{u_j}{(k-j)!} \text{ for } k = 1, 2, \ldots, d,$$

$$L^{(0)}(z) = z, \quad L^{(k)}(z) = (z-k)\,L^{(k-1)}(z) \text{ for } k = 1, \ldots, d-1,$$

とすれば,求める多項式は次により与えられる.

$$P(z) = u_0 + u_1\,L^{(0)}(z) + \cdots + u_d\,L^{(d-1)}(z).$$

### 3.2.2. 多変数で疎な場合

$P(x_1, x_2, \ldots, x_n)$ を求める $n$ 変数多項式とし, $P(x_1, x_2, \ldots, x_n) = \sum_{k=1}^{t} c_k {x_1}^{e_{k,1}} {x_2}^{e_{k,2}} \cdots {x_n}^{e_{k,n}}$ と表されるとしよう.当然, $P$ 中に存在する単項式,その個数 $t$,および数係数 $c_k$ は未知である. Ben-Or と Tiwari による算法 [1] では,次の二つの手順により多項式 $P$ を定める.

1. $P$ 中に存在する全ての単項式を定める,

2. 線形方程式系を解くことにより,数係数を定める.

$p_i$ が $i$ 番目の素数を表すとし, $b_k = {p_1}^{e_{k,1}} {p_2}^{e_{k,2}} \cdots {p_n}^{e_{k,n}}$, $a_j = P({p_1}^j, {p_2}^j, \ldots, {p_n}^j) = \sum_{k=1}^{t} c_k {b_k}^j$ とする.前半部では, Berlekamp と Masssey の算法 [2] を $a_0, a_1, \ldots, a_{2t}$ に適用し, $\Lambda(z) = \prod_{i=1}^{t}(1 - b_i z)$ を求めれば, $t$ および全ての指数が求まり,単項式が決定される.後半部では,次の Vandermonde 方程式を解けば良い.

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ b_1 & b_2 & \ldots & b_t \\ \vdots & \vdots & \ddots & \vdots \\ {b_1}^{t-1} & {b_2}^{t-2} & \ldots & {b_t}^{t-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{pmatrix}$$

この線形方程式は,直接解けば勿論ベクトル化されるが,後述する算法により,より効率良くかつベクトル化されて解くことができる.

さて,以上の算法を組み合わせれば,途中の計算(行列式の展開)はすべて剰余数の計算で置き換えられることが判った.換言すれば,多項式を求める計算では,その計算に補間法を適用することが可能であれば,計算全体は,

1. 目的とする多項式の充分に多くの点における整数値の,充分に多くの法による剰余を計算する,

2. それらの値から目的とする多項式を構成する

の二つのフェイズに分割される.そして,手間のかかる計算において,その複雑さの主要部となる前半のフェイズにおいて,データの均一性が確保され,高度なベクトル処理が可能となる.

## 4. パラレルかベクトルか?

本稿では,ベクトル処理のみを想定して議論を展開しているが,モジュラー算法においてもまた上記の算法の組合せにおいても,並列性は自明である.これらの自明な並列性は,単純に並列処理されるべきであろうか,或いはベクトル処理されるべきであろうか? 既に述べたように,その並列性は,多くの場合,容易に局所化されベクトル化されうる類のものである.この局所化は,実は,多項式や行列といった構造体に対するアクセスや処理を一元化するという効果もある.つまり,単純に並列処理した場合に行われるであろう複数のプロセッサにおける同等の処理は,ベクトル処理では取り除くことができ,計算時間に大きな影響はないであろうが,少なくともその無駄を省くことにはなろう.また,メモリの共有が行われない限り,データの通信量という観点からも,ベクトル処理の方が有利であろう.実際,行列のように中間式がより大きな構造を持つ場合,データの通信量が膨大なものとなりうる.次節で述べるインプリメントに際しても,このことが実際に問題となり,プログラムの構成を大きく変更せざるをえなかったことは事実である.以上から,本稿における手法においては,ベクトル処理が圧倒的に有利であると結論付ける.さらには,共有メモリのもとに,大域的には並列処理が,局所的には各プロセッサ毎にベクトル処理が可能な計算機環境が望まれる.

## 5. 実現技法

本節では,インプリメントの技法について説明する.図1は,実際にインプリメント済みの一変数の行列式計算のためのプログラムの構成とデータの流れを示している.詳細は,[9, 10]に詳しい.ここでは,設計方針とそれを実現するためのキーとなるアイディアのみを説明する.

### 設計方針

1. 対象となる多項式計算が明確にふたつのフェイズに分割されたことを活かし,ユーザーインターフェイスとしては既存の数式処理システムを利用し,ベクトル処理のプログラムは Fortran により開発する.ここで,既存の数式処理システムにベクトル処理プログラムを直接組み込むことはしない.

2. 複数のプログラム間の通信手段としては,通常のテキストファイルを用い,そのフォーマットにも特別な形式を仮定せず,利用者が扱い易いように工夫する.

3. ベクトル処理プログラムからは,できる限り数式処理的な処理を排除する.

以上の要求の満たすための主要なアイディアは,「ベクトル処理プログラムをフィルタとして機能させる」ことである.ベクトル処理プログラムは,標準入力中の,フロントエンドの数式処理システムの関数から生成された特定のブロックを,単に,法と多項式対の列に書き換える.そのブロックは,他の数式中に混ざって,あるオペレータ形式の引数として式の一部を構成するため,書き換えられた結果は,そのまま数式処理システムへの入力となって評価され得る.また一般に,多項式の記法は,*Mathematica* において乗算オペレータが不要な以外,多くの数式処理システムに共通である.つまり,ベクトル処理プログラム自体は,一切の変更無しに,色々な数式処理システムと連携させることが可能となっている.

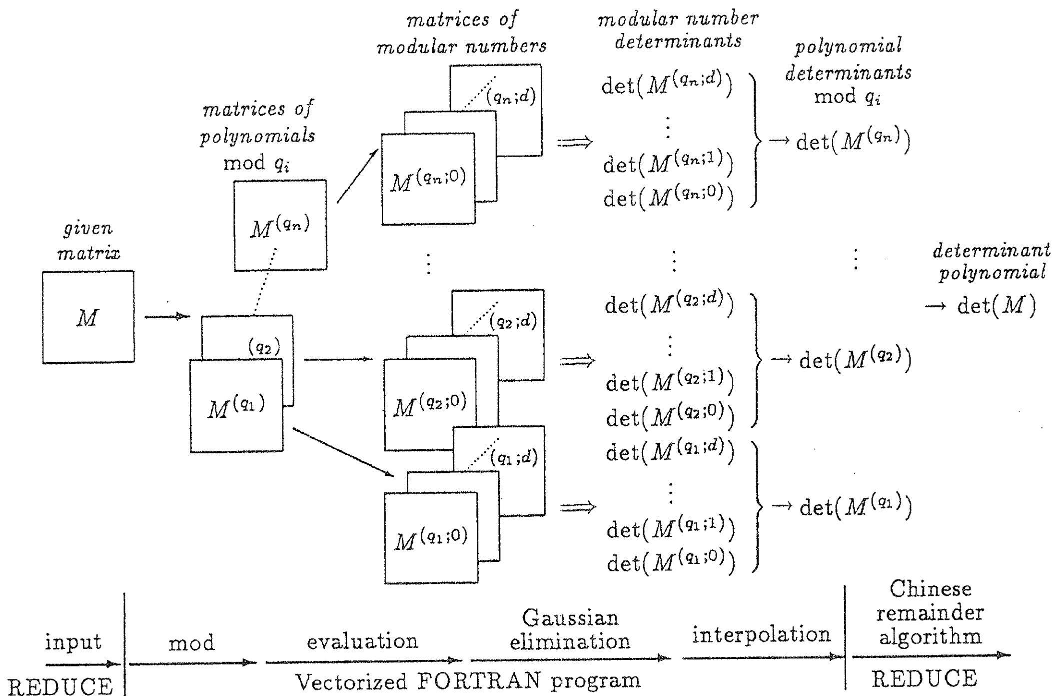


図 1: プログラムの構成とデータの流れ

## 6. ベクトル化技法

### 6.1. 剰余数の表現と四則演算

データ (剰余数) を適切に表現し, その演算をベクトル命令によって処理されるようにすることが, 高度なベクトル処理を導き出すための要点である. 表 1 は, HITAC S-820 上で, ハードウェア命令により

表 1: HITAC S-820 がサポートするデータ型と FORTRAN における宣言

| declaration in FORTRAN | number of bits | |
|---|---|---|
| | exponent | mantissa |
| integer | – | 31 |
| real*4 | 7 | 24 |
| real*8 | 7 | 56 |

直接処理されうるデータの型と FORTRAN における型宣言をまとめたものである (他の国産スーパーコンでも同様である). 法の個数は, 必要とされる (計算されるべき) 多項式像の個数を決めるものゆえ, 計算量の観点からは, できるだけ少なくすることが望ましい. そのためには, 法として用いる素数をできるだけ (勿論, 一語内で) 大きくとることが要求される. しかし, 精度を完全に保って計算を行うには, 次のような表現と演算のみが考慮の対象となりうる. 各表現における長所・短所もまとめると以下のとおりである.

1. 全て整数とする場合: 剰余数を (符合 + 15-bit) で表し, データも演算も integer*4 により行う.
   × ビット幅が狭く, 法が小さい.
2. 剰余数を integer*4 により (符合 + 28-bit) で表し, 演算は real*8 により行う.
   ○ 最大のビット幅 ⇒ 最大の法を用いうる.
   × 型変換 (integer*4 ⇔ real*8) を伴う.

3. 剰余数を (符合 + 28-bit) で表し, データも演算も real*8 により行う.

　○ 最大のビット幅 ⇒ 最大の法を用いうる.

　× メモリ使用量が倍になる.

4. 剰余数を real*4 により (符合 + 24-bit) で表し, 演算は real*8 により行う.

　○ 充分なビット幅 ⇒ 充分に大きい法を用いうる.

　○ 型の変換不要 (S-820 では浮動小数点数の演算は全て real*8 で行われる).

　○ 不要にメモリを喰わない.

以上から, 剰余数の表現と演算は最後に示した方法が最適であり, 法としては $2^{24}$ より小さい素数を選べばよいと結論付けられる.

剰余数の加減乗算は, 通常の演算を行い, その計算結果の値にたいして, 法による剰余計算を適用すれば良い. その剰余を求める計算は, FORTRAN の組込み関数 dmod (総称名では mod) で実現されており, ベクトル化されうる. しかし本関数を用いると, 非常に遅い命令の一つである除算命令を含んだベクトル命令列に翻訳される. しかも除数は, 常に法の素数であるから, 一定である. 以上のことから, 法の逆数をあらかじめ real*8 で計算しておき, mod 演算をその逆数との乗算と簡単な補正で実現してやれば, 高速化されうることが容易に推測される. 実際にこの方法を用いた結果, ベクトル処理の最も効を奏するガウス消去法の実行時間が 20 ~ 40% 減少することが確認された.

法演算における除算は, 除数と被除数の剰余列から商を計算するため, 直接はベクトル化されえない. しかし, 如何なる剰余列の長さも, $fib_i$ を Fibonacci 数として, $fib_n > 2^24$ なる $n$ より必ず小さいことに着目し, 剰余列の終端を検知する処理を加えてやれば, 複数の除数被除数に対してベクトル化することは可能である.

### 6.2. ガウス消去法におけるベクトル化

ガウス消去法においては, いくつかのベクトル化の方向が考えられる. ベクトル処理の効率を上げるには, ベクトル長をできるだけ大きく取ることが望ましい. このためには, 行や列をベクトルとするのではなく, 剰余数を要素とする多くの行列をベクトルを要素とする一つの行列とみなし, 「串刺し」にベクトル化するのが最適である. 何故ならば, 行列の「枚数」は, 法の個数と多項式中に存在するであろう単項式の個数の積により定められ, 一般には少なくとも行列の次元の 2 乗に比例する. このようなベクトル化は及びこの議論は, 行列式展開におけるガウス消去法の場合に限らず, あらゆる応用において成り立つと予想される (ベクトル化のひとつのテクニックである).

### 6.3. 多項式補間におけるベクトル処理

実際のインプリメントでは様々な処理においてベクトル化が可能となるが, その中でもとりわけベクトル化が有効と思われるのは, 補間法における一変数の多項式の演算である. これは冒頭で述べた, 多項式の特別な表現によるベクトル化だが, 密な一変数多項式の演算は高率かつ容易にベクトル化される. 既に具体的な算法を示した一変数の場合の補間では, 計算式自体は再帰的だが, その係数等の演算は単純にベクトル化される.

多変数の補間においても, 数係数を決定する最後の段階における Vandermonde 方程式の解法において, 一変数多項式を扱うことになる. その算法は, [4, 13] に詳しいが, 以下のとおりである. ふたつの一変数多項式を次のように定義する.

$$B(z) = \prod_{k=1}^{t} (z - b_k), \quad D(z) = \sum_{k=0}^{t-1} a_k z^{t-k}.$$

式 $B(z)D(z)/(z-w)$ を $z$ について形式的に展開した式中の $z^t$ の係数を $Q_t(w)$ とすれば, 解の係数 $c_i$ は $Q_t(b_i)/B'(b_i)$ で与えられる. 中間的なこれらの式は, 全て一変数で密な多項式である. よって, ベクト

ル化が有効に行われうる．しかも，一変数の補間の場合に比べて，扱う多項式の次数がはるかに大きくなるため，その効果は顕著であろう．その実際の効能については，[10] 中の簡単なテストの結果を参照されたい．

## 7. テストと評価

紙面の都合上，具体的なテスト例とその測定結果は省略し，その結果の要点のみを列記する．詳細については [10] を参照されたい．

1. これまで実際に計算が行われえなかった，100 次元を越すような行列の行列式の計算も可能となった．
2. 全く同じプログラムをベクトル化せずに実行した結果と比較すると，計算時間の短縮は著しい．このことは，単に Fortran と Lisp という記述言語の違いによって高速化されたのではなく，ベクトル処理が効を奏していることに他ならない．特にベクトル処理の効果はガウス消去法において著しい．
3. 数十次元程度の小さな (!) 行列では，数値行列の生成に最も時間を要する (当然，スカラーのみによる処理においてはガウス消去法に最も時間を要する)．
4. 一変数の場合には，補間法の計算時間は殆んど無視しうる．
5. 一変数の場合には一般に，補間法におけるよりもガウス消去法におけるベクトル化率の方が高い．このことは，我々の予測どおり，特別なデータ構造による単純なベクトル化よりも算法によるベクトル化の方が有効であることを示している．

## 8. 終わりに

結論じみたことは，既に随所に述べてあるので，ここでは今後の課題を述べて結びとする．第一の課題は，ごく最近理論的解明が完了した [8] 多変数多項式のモジュラー算法の実現とベクトル化である．次には，本稿で述べた手法を様々な多項式計算に適用することである．最も単純には長大な式の加減乗算にも適用することである．また行列式展開の拡張として，逆行列の計算も挙げられる．クラメルの公式や [7] によれば，逆行列の計算も多項式演算のみで可能であるし，より多大のメモリを要することから，スーパーコンの利用の効果がより大きく得られるものと期待される．さらに，代数方程式系を Gröbner 基底の計算と組み合わせた一般消去法によって解く解法は，最終的に行列式の計算に帰着される．その算法に我々の計算手法を適用すれば，これまで未解決の問題も解けるであろう．

### 謝辞

ベクトル処理プログラムの開発に際し，ベクトル化を導き出すための相談にのりまた適切な助言をくださった白沢智輝氏 (日立東北ソフトウェア) に感謝します．また本研究は文部省科学技術研究費奨励研究 (A)「スーパーコンピュータと数式処理算法」(課題番号 03780023) の補助のもと行われています．

### 参考文献

[1] Ben-Or, M., and Tiwari, P. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of 20th Symposium on the Theory of Computing* (1988), pp. 301–309.

[2] Blahut, R. E. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, 1985.

[3] Della Dora, J., and Fitch, J., Eds. *Computer Algebra and Parallelism.* Computational Mathematics and Applications. Academic Press, 1989.

[4] Kaltofen, E., and Lakshman, Y. Improved sparse multivariate polynomial interpolation algorithms. In *Symbolic and Algebraic Computation, ISSAC '88* (Rome, Italy, July 4–8 1988), no. 358 in LNCS, Springer-Verlag, pp. 467–474.

[5] Kaltofen, E., Lakshman, Y. N., and Wiley, J.-M. Modular rational sparse multivariate polynomial interpolation. In *Symbolic and Algebraic Computation, ISSAC '90* (Tokyo, Japan, Aug. 20–24 1990), pp. 135–139.

[6] Knuth, D. E. *Seminumerical Algorithms*, 2nd ed., vol. 2 of *The Art of Computer Programming.* Addison-Wesley, 1981.

[7] McClellan, M. T. The exact solution of systems of linear equations with polynomial coeffifients. *J. ACM 20*, 4 (1973), 563–588.

[8] Murao, H. Improved modular Ben-Or and Tiwari algorithm for sparse multivariate polynomial interpolation. (manuscript), Sept. 1991.

[9] Murao, H. Vectorization of symbolic determinant calculation. *SUPERCOMPUTER VIII*, 3 (1991), 36–48.

[10] Murao, H. Vector processing in symbolic determinant expansion on supercomputer. In *Proc. International Symposium on Supercomputing : ISS '91* (Fukuoka, Japan, Nov. 6–8 1991), pp. 145–154.

[11] Neun, W., and Melenk, H. *Implementation of the LISP-arbitrary precision arithmetic for a vector processor.* In Della Dora and Fitch [3], 1989, pp. 75–89.

[12] Villard, G. *Exact parallel solution of linear systems.* In Della Dora and Fitch [3], 1989, pp. 197–205.

[13] Zippel, R. Interpolating polynomials from their values. *J. Symbolic Computation 9*, 3 (1990), 375–403.