数理解析研究所講究録
986巻 1997年 224-233

多項式の高速多点評価法とその並列処理について
— 高速プログラムの開発へ向けて —

東京大学大型計算機センター 村尾裕一 (Hirokazu MURAO)

1. はじめに

Aho らによる著名なアルゴリズムの教科書 [1] の § 8.5 には,

系 2 n 次多項式の n 個の点での値は, $O_A(n\log^2 n)$ 時間で求めることができる.

とある. 本稿では, より一般化した次の問題を考える. 但し, $\mathbf{R}$ は適当な可換環とする.

> **問題:** 多項式 $g(z)\in\mathbf{R}[z]$ 及び 2^s 個の点 $h_k\in\mathbf{R}$, $0\le k<2^s$ が与えられた時, 多項式の値 $g(h_k)\in\mathbf{R}$, $0\le k<2^s$ を同時に求める. $\deg g=n$ とする.

最も単純な方法は, 各点における評価を独立な計算として, 2^s 回だけ Horner 法を適用することだが, その計算量は $2^s n$ の $\mathbf{R}$-演算となり, 計算量的には上述のアルゴリズムに劣る.

この問題は, 例えば離散的フーリエ変換 (DFT: discrete Fourier Transform) や次数別因数分解 (DDF: Distinct Degree Factorization) の計算に現れる. 但し, DFT に関しては, 高速 Fourier 変換法を用いるのが普通である. 一方, DDF に関しては, von zur Gathen と Shoup [7] や Kaltofen と Shoup [9] による最近のアルゴリズム ($GF(q)$ 上の多項式 F の DDF) では, 複数の k に対する $\prod_{i=1}^{l}(\lambda_{kl+i}(z)-z) \bmod F(z)$ 或は $\prod_{i=0}^{l-1}(\lambda_{kl}(z)-\lambda_i(z)) \bmod F(z)$ を分離多項式として用いるが, これらの式は, 予め求めておいた 2 変数多項式 $\prod_{i=0}^{l-1}(Y-\lambda_i(z)) \bmod F(z)$ を $Y=\lambda_{kl}(z)$ において評価して得る. この評価は, 後述のとおり, 複数の k に対し同時に行うことにより計算量を減らすことが可能となる. 本稿では, この DDF への応用を主たる目的として [5], 上の問題を扱うための実用的かつ高速なアルゴリズムを開発する. 既存のアルゴリズムを改良する基本的なアイディアは, 既に [11] に示したが, 本稿では, 並列処理も含めた実用化へ向けての詳細な検討を進める.

2. 既存の高速アルゴリズム

冒頭で触れた Aho らの教科書に記述されたアルゴリズム, 及び, それを元に上記問題の場合に一般化したアルゴリズム [7, Lemma 2.1] の概略を以下に示す.

(CR1) 次の多項式 $\Gamma_{j,k}(z)$ を計算する. 初期値を $\Gamma_{0,k}(z)=z-h_k$, $0\le k<2^s$ とし,

$$\Gamma_{j,k}(z)=\prod_{i=0}^{2^j-1}(z-h_{i+k2^j}) \Leftarrow \Gamma_{j-1,2k}(z)\,\Gamma_{j-1,2k+1}(z),\quad 0\le k<2^{s-j},\ 1\le j\le s.$$

(CR2) $\Phi_{s,0}(z) = g(z) \bmod \Gamma_{s,0}(z)$ とする.

(CR3) $j = s-1, s-2, ..., 0$ に対し，次の剰余計算を繰り返し，

$$\Phi_{j,\kappa}(z) \Leftarrow \Phi_{j+1,k}(z) \bmod \Gamma_{j,\kappa}(z), \quad \kappa = 2k, 2k+1, \ 0 \le k < 2^{s-1-j}, \quad (1)$$

最終的に，$g(h_k) = g(z) \bmod (z-h_k) = g(z) \bmod \Gamma_{0,k}(z) = \Phi_{0,k}(z)$ を得る.

計算量について

以降において，2つの d 次の多項式の積を計算するための計算量を $M(d)$ $(\approx O(d \log d))$ で表す．また，$2d$ 次割る d 次の多項式除算の計算量が $O(M(d))$ であることは既知のとおりである．(CR1) の計算量は，$\deg \Gamma_{j,k} = 2^j$ ゆえ，$\sum_{j=1}^{s} 2^{s-j} O(M(2^{j-1})) \le \sum_{j=1}^{s} O(M(2^{s-1})) = O(sM(2^{s-1}))$ の **R**-演算である．また，(CR3) の計算量は，$\deg \Phi_{j,\kappa} < \deg \Gamma_{j,\kappa} = 2^j$ ゆえ (CR1) の計算量に等しい．(CR2) の計算は $n \ge 2^s$ の場合にのみ必要だが，その計算は，後述のとおり，$\Gamma_{s,0}$ を除数とする除算により被除数式の次数を 2^s ずつ落していくという計算と同等ゆえ，計算量は $O((n/2^s)M(2^s))$ の **R**-演算である.

上のアルゴリズムの (計算量という観点での漸近的な) 高速性は，DFT を用いた多項式乗算 (及び除算) の高速性に依存しており，多項式演算に古典的算法を用いる限り ($M(d) = O(d^2)$)，互いに独立な 2^s 個の点での評価を Horner 法により行う方が実際上は得である．以降の議論では，n 及び 2^s は充分に大きいと仮定し，多項式演算には DFT による漸近的高速アルゴリズムを用いるものとする[1)]．そして，上のアルゴリズム全体の高速性は，評価点を適宜 (2 の冪乗個ずつ) 組み合わせて除算を行うことと，その除算の高速アルゴリズムに基づいている．そこで，いくつの評価点を組み合わせるべきかという疑問が生ずる．次に，このことについて検討する.

(i) 評価する多項式の次数 $(= n) \le$ 点の個数 $(= 2^s)$ の場合:
(CR1) と (CR3) の計算だけを行えば良いが，その計算量は $O(sM(2^s))$ の **R**-演算である．2^s 個の点を 2^k 個 (但し $n \le 2^k < 2^s$) ずつに分けて評価することを考える．この場合の総計算量 $2^{s-k}O(kM(2^k))$ は，一般に $M(d)/d$ は d について単調増加ゆえ，k と共に減少する．よって，2^s 個の点を $m = 2^{\lceil \log_2 n \rceil}$ 個ずつに分け，$\lceil 2^s/m \rceil$ 回だけ (CR1) と (CR3) の計算を繰り返す方が計算量的には得である．つまり，この場合は冒頭の一般化する前の問題に帰着して，総計算量は $O((2^s/n)\, M(n) \log_2 n)$ となり，また，以降においては $n \ge 2^s$ の場合だけを考えれば良いことになる.

(ii) 評価する多項式の次数 $(= n) \ge$ 点の個数 $(= 2^s)$ の場合:
総計算量は $O((n/2^s + s)\, M(2^s))$ の **R**-演算である．2^s 個の点を 2^k $(k < s)$ 個ずつに分けるべきだろうか? 2^k 個ずつに分ける場合と 2^{k-1} 個ずつに分ける場合とを比較すると，次に示す計算部分だけが異なる (§ 3.1. に示す方法を用いるものとする).
(2^k 個ずつの場合) $\cdots$ 2^{s-k} 組の点の組合せについて,

(a) $\Gamma_{k,0}(z)$ 及び $\Delta_{k,0}$ の計算 ..... $2^{s-k}\, O(M(2^{k-1}))$

[1)] 多項式演算の高速アルゴリズムが，次数が実用的な範囲の値であっても実際に高速であることが，Shoup([12] 及び [13]) により検証されている

(b) (CR2) で $g(z)$ の次数を 2^k未満まで落す .... $2^{s-k}\,O((\lceil n/2^k\rceil-1)\,M(2^k))$
(2^{k-1}個ずつの場合) ... 2^{s-k+1} 組の点の組合せについて,

(b') (CR2) で $g(z)$ の次数を 2^k未満まで落す
..... $2^{s-k+1}\,O((\lceil n/2^{k-1}\rceil-2)\,M(2^{k-1}))$

$M(d)$ を上記のとおりとすれば, (b') の計算量は (b) の約 2 倍である. n が $2^s \geq 2^k$ に比べて充分大きい場合には, (a) の計算量は (b) の計算量より小さいので, 2^k 個ずつの場合の方が全体の計算量が小さくなり, 2^s個全部でやった方が得である. そうでない場合には, 最適な k の値を決めることは難しい. しかしながら, この場合には (CR2) での除算の回数が少いこと, 及び, § 3. のとおり, (a) と (b) とでは, 実際の乗算の回数が (a) の方が倍くらい多い一方, (a) における乗算の多項式の次数が (b) におけるものの半分であり計算量が少いことを考慮すれば, $k=s$ と取りさえすれば良いであろう.

3. 改良されたアルゴリズム

3.1. 除算のインライン展開

式 (1) の除算において, 商を $q_{j,\kappa}$ とし

$$\Phi_{j+1,k}(z) = q_{j,\kappa}(z)\,\Gamma_{j,\kappa}(z) + \Phi_{j,\kappa}(z), \tag{2}$$

漸近的高速算法 [2] を具体的に記述する. 各多項式の次数を $d_{j,i} = \deg \Phi_{j,i}$ 及び $e_{j,\kappa} = \deg q_{j,\kappa}$ とおく. この時 $d_{j,\kappa} < 2^j = \deg \Gamma_{j,\kappa}$ 及び $e_{j,\kappa} = d_{j+1,k} - 2^j < 2^j$ である. また, $d_{j+1,\kappa} \geq 2^j$ と仮定する (さもなくば, 除算は不要で$\Phi_{j,\kappa} = \Phi_{j+1,k}$ とすれば良い). ここで, 高次と低次で係数を入れ換えた多項式を表す次の記法を導入する.

(記法) 任意の多項式 f に対し, f^* を $\boxed{f^*(z) \stackrel{\text{def}}{=} z^{\deg f}\, f(z^{-1})}$ と定義する.
例えば $f(z) = f_0 + f_1 z + \cdots + f_d z^d$ に対しては, $f^*(z) = f_d + f_{d-1} + \cdots + f_0 z^d$.

式 (2) で, $z \to z^{-1}$ とした後 $z^{d_{j,i}}$ を掛けた式

$$\left(z^{d_{j+1,k}}\Phi_{j+1,k}(z)\right) = \left(z^{e_{j,\kappa}} q_{j,\kappa}(z)\right)\left(z^{2^j}\Gamma_{j,\kappa}(z)\right) + z^{e_{j,\kappa}+1}\left(z^{2^j-1}\Phi_{j,\kappa}(z)\right)$$

を考えると, ( ) 内は全て多項式なので, 次式が成り立つ.

$$\Phi^*_{j+1,k}(z) = q^*_{j,\kappa}(z)\,\Gamma^*_{j,\kappa}(z) \bmod z^{e_{j,\kappa}+1}.$$

$\Delta_{j,\kappa}$ を $\Gamma^*_{j,\kappa}(z)$ の逆数, 即ち, $\Delta_{j,\kappa}\Gamma^*_{j,\kappa}(z) = 1 \bmod z^{2^j}$ を満たす多項式とすれば, 商 $q_{j,\kappa}(z)$ は

$$q^*_{j,\kappa}(z) = \Delta_{j,\kappa}\ \Phi^*_{j+1,k}(z) \bmod z^{e_{j,\kappa}+1} \tag{3}$$

より得られる. この逆数の多項式 $\Delta_{j,\kappa}$ は, 次の Newton iteration によって求める.

$$\begin{aligned}
&\Delta^{(0)}_{j,\kappa} \Leftarrow 1, \quad \Delta^{(1)}_{j,\kappa} \Leftarrow 2 - \Gamma^*_{j,\kappa}(z) \bmod z^2,\\
&\Delta^{(i)}_{j,\kappa} \Leftarrow \Delta^{(i-1)}_{j,\kappa}\left(2 - \Delta^{(i-1)}_{j,\kappa}\,\Gamma^*_{j,\kappa}(z)\right) \bmod z^{2^i},\ \ i = 2, 3, \ldots
\end{aligned} \tag{4}$$

各 i について$\Delta_{j,\kappa}^{(i)}\,\Gamma_{j,\kappa}^{*}(z) = 1 \bmod z^{2^i}$ が成り立ち，最後に $\Delta_{j,\kappa} = \Delta_{j,\kappa}^{(j)}$ を得る．ここで，$\Gamma_{j,k}^{*}(z) = \Gamma_{j-1,2k}^{*}(z)\Gamma_{j-1,2k+1}^{*}(z)$ ゆえ，$(\Delta_{j-1,2k}\,\Delta_{j-1,2k+1})\,\Gamma_{j,k}^{*}(z) = 1 \bmod z^{2^{j-1}}$ であることに注意すれば，

$$\Delta_{j,k}^{(j-1)} = \Delta_{j-1,2k}\,\Delta_{j-1,2k+1} \bmod z^{2^{j-1}}. \tag{5}$$

よって，上の反復は不要であり，最後の $i = j-1$ の場合の (4) 式だけを計算すれば良い．また，[13] のように，$\Delta_{j,\kappa}^{(i-1)}\Gamma_{j,\kappa}^{*}(z) \bmod z^{2^i} = 1 + z^{2^{i-1}}\delta_{j,\kappa}$ (但し，$\delta_{j,\kappa}$ は $(2^{i-1}-1)$ 次以下の多項式) と書けるので，(4) 式を

$$\Delta_{j,\kappa}^{(i)} = \Delta_{j,\kappa}^{(i-1)} - \Delta_{j,\kappa}^{(i-1)}\left(z^{2^{i-1}}\delta_{j,\kappa}\right) \bmod z^{2^{j-1}} \tag{6}$$

と書き直して，順次，高次の近似項だけを求めていくように計算すれば良い．

(CR2) の $g \bmod \Gamma_{s,0}$ の計算について

von zur Gathen と Shoup [7, Lemma 2.1] は，技巧的な方法を示しているが，上の $\Delta_{j,\kappa}$ を用いると，同等の計算量でより素直な計算法を記述することができる．

一般に，多項式 $P(z)$, $G(z)$ が与えられた時 (但し，$e = \deg P - \deg G > 0$)，$D^{(j)}$ を $G^{*}(z)$ の modz^j での逆数，即ち，$D^{(j)}G^{*}(z) \bmod z^j = 1$ を満たす多項式とし，$q_j^{*}(z) = P^{*}(z)D^{(j)} \bmod z^j$ とすれば，$P^{*}(z) - q_j^{*}(z)G^{*}(z) \bmod z^j = P^{*}(z)(1 - D^{(j)}(z)G^{*}(z)) \bmod z^j = 0$ ゆえ，

$$\deg(P - z^{\deg P - \deg q_j - \deg G}\,q_j G) \le \deg P - j$$

が成り立つ．即ち，P の項の内，最高次から (少くとも) j 個の項だけを消去することができる[2]．(CR2) の $g \bmod \Gamma_{s,0}$ の計算は，g の次数を (2^s-1) 以下に落すことが目的だが，上述のとおり，$\Delta_{s,0}^{(i)}$ を用いて次数を (少くとも) 2^i ずつ減らして行くことが可能である．では，i としてはどのような値を用いるべきだろうか? $i < s$ の場合は，§ 2. の考察のとおりなので考えない．一方，$i > s$ の場合，Newton iteration を続けることにより$\Delta_{j,\kappa}^{(i)}$ を求めることも可能だが，この高次の近似式を用いることは有効だろうか?答えは否である．なぜならば，$g \bmod \Gamma_{s,0}$を$\Delta_{j,\kappa}^{(i)}$を用いて得るまでには，上述の高次項の消去の計算を $(\lceil \deg(g)/2^i \rceil - 1)$ 回だけ繰り返すが，その一回毎の計算量は $O(M(2^i)) > O(2^i)$ なので，全体の計算量は i と共に増加する．よって，$\Delta_{s,0}$ を用いるのが良い．

3.2. Speeded Chinese remaindering algorithm for multipoint evaluation

上で述べたとおり，除算において漸近的な高速性 (乗算と同等の計算量) を導くための計算技法は，実用的な反復公式となった．図 1 に，改良されたアルゴリズム (Speeded Chinese Remaindering: 略称 SCR) を記述する．このアルゴリズムの計算量は，元のアルゴリズムと同等である．但し，このアルゴリズムの高速性は，前述のとおり，高速多項式乗算アルゴリズムの利用を前提としており，次節に示すような，より詳細な検討とアルゴリズムの再構築が必要となる．スペース要量については，元のアルゴリズムと比較すると，$\Delta_{j,k}$ の分だけ余計に必要だが，この分は $\Gamma_{j,k}$ の分と同等ゆえ，オーダーでは同等である ($\sum_{j=1}^{s} 2^{s-j}O(2^j) = O(s2^s)$ だけの **R**-要素)．

[2] 因みに，この方法 (の * の方) は，最近の多倍長整数の GCD 計算アルゴリズム [8], [14] において，除算の代わりに多倍長整数の下位の語を消去して行く方法と同等であり，その多項式版とみなすことができる．

Algorithm **SCR**

入力　：多項式 $g(z) \in \mathbf{R}[z]$ と 2^s個の $h_k \in \mathbf{R}, 0 \le k < 2^s$. 但し, $\deg g \ge 2^s$ とする.

出力　：2^s 個の値 $g(h_k) \in \mathbf{R}, 0 \le k < 2^s$.

(SCR1)　初期値を $\Gamma_{0,k}(z) \Leftarrow z - h_k, 0 \le {}^\forall k < 2^s$ とし, 多項式 $\Gamma_{j,k}(z)$ を次のとおり計算する.
for $j = 1$ **to** s **do** $\Gamma_{j,k}(z) \Leftarrow \Gamma_{j-1,2k}(z)\,\Gamma_{j-1,2k+1}(z), \quad 0 \le {}^\forall k < 2^{s-j}$.

(SCR1')　初期値を$\Delta_{0,k} \Leftarrow 1, 0 \le {}^\forall k < 2^s$ 及び $\Delta_{1,k} \Leftarrow 2 - \Gamma^*_{1,k}(z) \bmod z^2 = 1 + (h_{2k} + h_{2k+1})z$, $0 \le {}^\forall k < 2^{s-1}$ とし, $\Delta_{j,k}$ を次のとおり計算する.
for $j = 2$ **to** s **do**

$$\begin{aligned}\Delta_{j,k} &\Leftarrow \Delta^{(j-1)}_{j,k}(2 - \Delta^{(j-1)}_{j,k}\Gamma^*_{j,k}(z)) \bmod z^{2^j} \\ &= \Delta^{(j-1)}_{j,k} - \Delta^{(j-1)}_{j,k}\left(z^{2^{j-1}}\delta_{j,k}\right) \bmod z^{2^j}, \quad 0 \le {}^\forall k < 2^{s-j},\end{aligned}$$

但し, $\Delta^{(j-1)}_{j,k} \Leftarrow \Delta_{j-1,2k}\Delta_{j-1,2k+1} \bmod z^{2^{j-1}}$ 及び
$\left(z^{2^{j-1}}\delta_{j,k}\right) \Leftarrow \Delta^{(j-1)}_{j,k}\Gamma^*_{j,k} \bmod z^{2^j} - 1$ とする.

(SCR2)　$\Phi_{s,0} \Leftarrow g$;
while $\deg \Phi_{s,0} > 2^{s+1}$ **do** $\left[\begin{array}{l} q^*(z) \Leftarrow \Phi^*_{s,0}(z)\,\Delta_{s,0} \bmod z^{2^s}; \\ \Phi_{s,0} \Leftarrow \Phi_{s,0} - z^{\deg \Phi_{s,0} - \deg q - 2^s}\, q\,\Gamma_{s,0}; \end{array}\right.$

if $\deg \Phi_{s,0} \ge 2^s$ **then** $\left[\begin{array}{l} q^*(z) \Leftarrow \Phi^*_{s,0}(z)\,\Delta_{s,0} \bmod z^{\deg \Phi_{s,0} - 2^s + 1}; \\ \Phi_{s,0} \Leftarrow \Phi_{s,0} - q\,\Gamma_{s,0}; \end{array}\right.$

(SCR3)　**for** $j = s - 1$ **downto** 0 **do**
$0 \le {}^\forall k < 2^{s-j-1}$ 及び $\kappa = 2k, 2k+1$ に対し

$$\left[\begin{array}{ll} q^*_{j,\kappa}(z) & \Leftarrow \Phi^*_{j+1,k}(z)\,\Delta_{j,\kappa} \bmod z^{\deg \Phi_{j+1,k} - 2^j + 1}; \\ \Phi_{j,\kappa} & \Leftarrow \Phi_{j+1,k} - q_{j,k}\,\Gamma_{j,\kappa}; \end{array}\right.$$

$\Phi_{0,k} = g(h_k), 0 \le {}^\forall k < 2^s$ が求める値である.

Fig. 1.　改良されたアルゴリズム

4. DFT による高速多項式乗算アルゴリズムの利用

今日 (扱うような規模の計算) では, 高速なソフトウェアを開発する場合 (少くとも, ある特定の目的の計算では), 多項式の乗算に高速アルゴリズムを用いることは不可欠である ([12], [3], [13], [6]). 本節では, 前節のアルゴリズムにおいて, DFT による高速多項式乗算アルゴリズムを用いる場合の詳細について考察する. 多項式の係数域 $\mathbf{R}$ としては, p を素数として, 有限体 $\mathbf{Z}_p$ の場合と, 我々の主目的である DDF で必要な $\mathbf{Z}_p[x]/(f)$ (但し, f は $\mathbf{Z}_p$ 上の多項式で $\deg f \gg 1$ とする) の場合を考える[3]. 特に, 後者の場合, 2 変数の多項式として演算を行う必要があるが, 係数の $\mathbf{R}$-演算が高価であるので, g の次数が低い場合でも, z に関して変換を施し $\mathbf{R}$-演算 (乗算) の回数を減らすことは有効であろう. また, プログラムの高速性を求める場合, 多倍長演算をできるだけ避けるための努力も必要だが, $p \gg 1$ の場合の扱い等についても言及する.

[3] より一般的な場合でも, Cantor と Kaltofen の方法 [4] 等の利用により, 同様の扱いが可能と思われるが, ここでは触れない.

DFT による多項式乗算アルゴリズム (の計算量 $M(d) = O(d \log d)$) の主要な部分は (逆) 変換である．このため，前節のアルゴリズムにおけるように，途中の計算を変換した値によって計算することが可能な場合には，アルゴリズム全体での変換の回数を減らす努力が必要になる．その場合，DFT による乗算というのは，基本的に evaluation & interpolation (により，結果の多項式の係数を決定する方法) ゆえ，多項式中の何次の項の係数を決定するかを詳細に見極める必要があり (例えば，$\Gamma_{j,k}$ の主係数が 1 であることは既知)，これにより何次の変換が必要かも決まってくる．また，変換結果においては，$\bmod z^i$ の演算や，多項式の次数を定めることが困難であることにも注意する必要がある．

以降において，次の表記を用いる．先ず，通常の式と区別するために，変換結果の値には $\overline{w}$ のように上線を付し，DFT 変換値の (要素毎の) 乗算を $\otimes$ で表す．

$\mathrm{DFT}(P, a : b, n)$ ：多項式 P の a 次 $\sim (b-1)$ 次の係数の列に対し，n 次の離散的フーリエ変換を施した結果の値の列．

$\mathrm{DFT}^{-1}(V, a : b, n)$ n 個の離散的フーリエ変換の値の列 V に逆変換を施して得られる a 番目 $\sim (b-1)$ 番目の値の列．

(1') $\Delta_{j,k}$ の計算について．$\Delta_{j,k}$ の次数は $\leq 2^j - 1$ ゆえ，(SCR1') における全ての右辺の (乗算の) 結果は，2^j 次の多項式 $\Gamma^*_{j,k}$ をそのまま用いたとしても，高々 2^j 項であり，各 j の段階では 2^j 次の変換を用いれば良い．$\bmod z^{2^j}$ による高次の項の除去は，逆変換により多項式を求める際に，$0 \sim (2^j - 1)$ 次の項だけを復元するという方法を採れば良い．また，項数が増えるが，中間式の計算における $\bmod z^{2^j}$ を除去した次のような計算法も可能である．

$\Delta^{(j-1)'}_{j,k} \Leftarrow \Delta_{j-1,2k}\Delta_{j-1,2k+1}$.
$\Delta^{(j-1)'}_{j,k}\Gamma^*_{j,k} = 1 \bmod z^{2^{j-1}}$ ゆえ，$\left(z^{2^{j-1}}\delta'_{j,k}\right) \Leftarrow \Delta^{(j-1)'}_{j,k}\Gamma^*_{j,k} - 1$ とし，
$\Delta'_{j,k} \Leftarrow \Delta^{(j-1)'}_{j,k} - \Delta^{(j-1)'}_{j,k}\left(z^{2^{j-1}}\delta'_{j,k}\right)$ とすれば，$\Delta'_{j,k}\Gamma^*_{j,k} = 1 + \left(z^{2^{j-1}}\delta'_{j,k}\right)^2$ が成り立つ．よって，$\Delta_{j,k} = \Delta'_{j,k} \bmod z^{2^j}$.

ここで，$\Delta^{(j-1)'}_{j,k}$ の次数は $\leq 2^j - 2$ であり，また，$\left(z^{2^{j-1}}\delta'_{j,k}\right)$ については，2^{j-1} 次以上の項のみから成り，次数が $\leq (2^{j+1} - 2)$ であることに注意．よって，補正項の全体は，2^{j-1} 次以上の項のみから成り，次数は $\leq (2^{j+1} + 2^j - 4)$ である．即ち，$2^{j+2}(\geq 2^{j+1} + 2^{j-1} - 3 =$ 補正項全体の最大項数) 次の変換を用いて計算すれば，中間式の逆変換を求める必要は無くなる．真の補正項だけは，$2^{j-1} \sim (2^j - 1)$ 次の項のみを逆変換により求める．即ち，前提条件である $\deg \Delta_{j,k} \leq 2^j - 1$ を満たすためには，2^{j+2} 次の DFT(逆) 変換を 3 回だけ計算する必要がある ($\Delta_{j-1,\kappa}$ の変換と $\Delta_{j,k}$ を求める逆変換)．一方，(SCR1') に示したアルゴリズムでは，Δ や δ に関する 2^j 次の (逆) 変換 を 7 回だけ計算する必要があるが，n 次の DFT 変換の計算量は $O(n \log_2 n)$ なので，このアルゴリズムの方が，DFT の変換の回数が多いとは言え，全体の計算量は少くて

済む.

(1'-1) $\overline{\Delta_{j-1,\kappa}} \Leftarrow \mathrm{DFT}(\Delta_{j-1,\kappa}, 0:2^{j-1}, 2^j)$ for $\kappa = 2k, 2k+1$

(1'-2) $\overline{w} \Leftarrow \overline{\Delta_{j-1,2k}} \otimes \overline{\Delta_{j-1,2k+1}}$

(1'-3) $x \Leftarrow \mathrm{DFT}^{-1}(\overline{w}, 0:2^{j-1}, 2^j)$ $(= \Delta_{j,k}^{(j-1)})$

(1'-4) $\overline{x} \Leftarrow \mathrm{DFT}(x, 0:2^{j-1}, 2^j)$

(1'-5) $\overline{w} \Leftarrow \overline{x} \otimes \overline{\Gamma_{j,k}^*} - \overline{1}$ $\overline{\Gamma_{j,k}^*}$ 及び $\overline{1}$は 2^j 次の DFT 値列

(1'-6) $w \Leftarrow \mathrm{DFT}^{-1}(\overline{w}, 2^{j-1}:2^j, 2^j)$ $(= z^{2^{j-1}}\delta_{j,k})$

(1'-7) $\overline{w} \Leftarrow \overline{x} \otimes \mathrm{DFT}(w, 2^{j-1}:2^j, 2^j)$ $(= \Delta_{j,k}^{(j-1)}(z^{2^{j-1}}\delta_{j,k})$ の DFT)

(1'-8) $w \Leftarrow \mathrm{DFT}^{-1}(\overline{w}, 2^{j-1}:2^j, 2^j)$

(1'-9) $\Delta_{j,k} \Leftarrow x - w$

(2) (SCR2) では，g の次数を 2^s の倍数であると仮定し(必要ならば，係数を 0 とする高次の項を補う)，g^* の低次の方の項から順々に 2^s項だけ消去していくことにより，次数を考慮する必要がなくなると同時に，g^* の変換値のみを用いて計算することが可能となる．$2^s n > \deg g \geq 2^s(n-1)$ とし，$g^*(z) = \sum_{i=0}^{n} \phi_i z^{2^s i}$ と書き，$\overline{\psi}$ 及び $\overline{\phi_i}$ を各々 z^{2^s} 及び ϕ_i の 2^{s+1}次の DFT とする ($\overline{\psi}$ は $(1,-1,1,-1,...)$)．$i = 0,1,...,n-1$ の順に

$$\begin{aligned} q^* &\Leftarrow \mathrm{DFT}^{-1}(\overline{\phi_i} \otimes \overline{\Delta_{s,0}}, 0:2^s, 2^{s+1}) \\ \overline{q^*} &\Leftarrow \mathrm{DFT}(q^*, 0:2^s, 2^{s+1}) \\ \overline{\phi_{i+1}} &\Leftarrow \overline{\phi_{i+1}} + (\overline{\phi_i} - \overline{q^*} \otimes \overline{\Gamma_{s,0}^*}) \otimes \overline{\psi} \end{aligned}$$

を計算すれば，最終的に $\overline{\phi_n}$ として，$\Phi_{s,0} = g \bmod \Gamma_{j,\kappa}$ を (2^s-1) 次の多項式と見做した時の $\Phi_{s,0}^*(z)$ の 2^{s+1} 次の DFT を得る．但し，上の計算では，$\overline{q^*}$ 及び$\overline{\Gamma_{s,0}^*}$ として 2^{s+1} 次の変換を用いていることに注意．上の最後の式の () 内は，$\phi_i^* z^{2^s}$ を $\Gamma_{s,0}$ で除した余りを計算していることに他ならず，高々 2^s 項からなる．よって，次のような計算も可能である．

$$\begin{aligned} \overline{q^*} &\Leftarrow \mathrm{DFT}(q^*, 0:2^s, 2^s) \\ \phi_{i+1} &\Leftarrow \phi_{i+1} + \mathrm{DFT}^{-1}(\mathrm{DFT}(\phi_i, 0:2^s, 2^s) - \overline{q^*} \otimes \overline{\Gamma_{s,0}^*}, 0:2^s, 2^s) \end{aligned}$$

計算量は同等である．

(3) (SCR3) でも，前項と同一の方法を適用する．即ち，$\Phi_{j,\kappa}$ 及び $q_{j,\kappa}$ の次数は，共に (2^j-1) と見做した上で，次の計算を行う．

(3-1) $\Phi_{j+1,k}^* \Rightarrow \phi_{j+1,k}^{(l)} + z^{2^j}\phi_{j+1,k}^{(h)}$ と，低次と高次の項に分ける

(3-2) $\overline{w} \Leftarrow \mathrm{DFT}(\phi_{j+1,k}^{(l)}, 0:2^j, 2^{j+1})$

(3-3) $q \Leftarrow \mathrm{DFT}^{-1}(\overline{w} \otimes \overline{\Delta_{j,\kappa}}, 0:2^j, 2^{j+1})$

(3-4) $\overline{q} \Leftarrow \mathrm{DFT}(q, 0:2^j, 2^j)$

(3-5) $\overline{w} \Leftarrow \overline{w} - \overline{q} \otimes \overline{\Gamma_{j,\kappa}^*}$ 右辺の $\overline{w}$ は 2^j次の変換の分だけを使用

(3-6) $\Phi_{j,\kappa}^* \Leftarrow \phi_{j+1,k}^{(h)} + \mathrm{DFT}^{-1}(\overline{w}, 0:2^j, 2^j)$

この場合も，全体の計算を 2^{j+2} 次の DFT を用いて計算すれば ($\deg \Phi_{j+1,k}^* < 2^{j+1}$ に注意)，$\Phi_{j,k}^*$に関して，DFT 値のみを用いて計算することが可能となり変換の回数を減らすことが可能だが，q^* に関する (逆) 変換が必要なため，計算量的には劣る．

(SCR2) と (SCR3) において上のような計算法を用いれば，多項式 $\Gamma_{j,k}$ については，$\Gamma_{j,k}^*$ の DFT のみが必要であり，その結果，本アルゴリズム全体においても同様であることがわかる．

係数域 $\mathbf{R}$ での演算に関し，注意が必要である．先ず，係数域 $\mathbf{R}$ が $\mathbf{Z}_p[x]/(f)$ の場合，DFT 値も多項式となるが，乗算 $\otimes$ の度毎に $\bmod f$ の計算を行う必要がある．この場合，DFT 値に対して行えば充分である．次に，係数域の基礎体 $\mathbf{Z}_p$ の p が多倍長の場合，多倍長整数の計算を減らすことがしばしば高速化につながる．それには，Shoup [13] のように，p を法とする演算を整数上での演算と見做し，途中の計算は，その整数演算の結果の値を中国剰余定理により復元するのに充分な個数の一語長の素数 p_i を法として行う．この場合，乗算 $\otimes$ の度毎に，逆変換を施し多項式表現を得た上で，整数上での演算結果 (とその結果の $\bmod p$) との対応をとる必要があることに注意．また，上の議論では，2^t を $(p-1)$ の因子として，$t > s$ (即ち，1 の原始 2^{s+1} 乗根が存在) を仮定しているが，そうでない場合には，2^s個の点を 2^{t-1}ずつに分割して上記の方法を用いるか，或は，上述の多倍長の場合の扱いをすればよい．

5. 並列処理についての検討

(SCR1), (SCR1') 及び (SCR3) の各ステップで計算する $\Gamma_{j,k}$, $\Delta_{j,k}$, $\Phi_{j,k}$ は，それぞれに，j 段において 2^{s-j}個の node からなる二分木を構成する．図 2 は，$\Gamma_{j,k}$ についての二分木

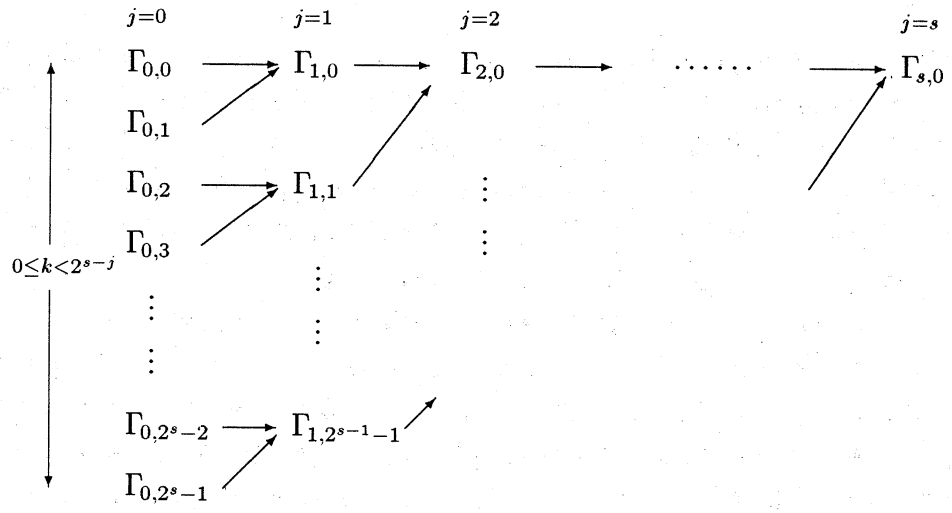

Fig. 2.

を示したものだが，矢印の右側の式は左側の (2 つの) 式から計算される (依存する) ことを示す (つまり，計算は左から右へ行なわれる．bottom-up 型)．$\Delta_{j,k}$ についての二分木も全く同じ形になる．一方，$\Phi_{j,k}$ についての二分木は，矢印の向きが逆になり，計算は右から左に行なわれることになる (top-down 型)．いずれの場合も，j に関しては依存関係があるため逐次的に計算を進める必要があるが，j の各段においては，$0 \leq {}^{\forall}k < 2^{s-j-1}$ 個の計算は独立であり，同時に (並列で) 計算実行可能である (並列度は最大 2^s)．通信について検討する．

(仮定) $M(n)$ だけの $\mathbf{R}$-演算は，少なくとも，n 個の $\mathbf{R}$ 要素の通信のコスト程度である．

- bottom-up 型の場合: (SCR1) と (SCR1')
  添字 $_{j,k}$の 2^{s-j}個の式の計算は，全て異なるプロセッサ上で行なうものとする．添字 $_{j+1,k}$の式 (矢印の右側の式) の計算を $_{j,2k}$ の式が存在するプロセッサ上で実行するならば，対となっている斜めの矢印の左側 (添字$_{j,2k+1}$ の) 式のデータ転送が必要となる．各二分木においては，この通信の latency は免れ得ない．ところが，(SCR1) と (SCR2) を融合してしまい，$\Gamma_{j+1,k}$ 及び $\Delta_{j+1,k}$ の計算を (同一プロセッサ上で) 交互に実行することにより，上の仮定の基では，この latency の隠蔽 (latency hiding) が可能である．
- top-down 型の場合: (SCR3)
  計算処理は，$\Phi_{j+1,k} \to \Phi_{j,\kappa}$, $\kappa = 2k, 2k+1$ と 2 つずつに枝分かれしていくが，一方 ($\kappa = 2k$) は同一プロセッサ上で，もう一方 ($\kappa = 2k+1$) は，($\Phi_{j+1,k}$の) データ転送の後，別の ($\Gamma_{j,\kappa}$と$\Delta_{j,\kappa}$ が存在する) プロセッサ上で実行するものとする．この場合，$\Phi_{0,k}$ に到るまでには，k を 2 進表示した時の 1 の個数分だけの回数の通信が必要で，各段階 j ($j = s, ..., 1$) では 2^j 個の $\mathbf{R}$ 要素分の通信が行なわれる．通信のデータ待ちにより全プロセッサが idle 状態になってしまうことは無いとは言え，全体の計算時間へのこの分の latency の影響は免れられない．これを避けるには，同一の計算を複数のプロセッサ上で行うことを許し，$\Phi_{s,0}$ を最初に broadcast した後，各プロセッサで必要となる式の計算を各自で行えば良い．

(SCR2) の $\Phi_{s,0}(z) = g(z) \bmod \Gamma_{s,0}(z)$ の計算については，§ 3.1. の方法は逐次的なため，von zur Gathen と Shoup の方法 [7, Lemma 2.1] の並列化を検討すべきかもしれない．つまり，$g(z)$ を折り畳んだ各部分式の計算を，適宜別のプロセッサに振り分けるような方法を考える．例えば，$\left(z^{2^s}\right)^i$, $0 \le i \le t$ の冪乗計算を上のような二分木で計算する方法が考えられる．この場合，逐次性は $\log_2(t+1)$ まで圧縮することができ，最大の並列度は $O((t+1)/2)$ 程度になる．各計算は $O(M(2^s))$ だけの $\mathbf{R}$-演算．しかし，2^s 次毎に折り畳むのが良いのか，或は，より高い次数で折り畳むのが良いのかは難しい問題である..

以上のような並列処理を行うとして，一般に 2^P台のプロセッサを用いた場合の並列計算量を求めることは，様々な場合分けが必要となり，なかなかの難問である．さらに，実際の計算となると，多項式乗算の算法の切替え等も含めて考慮する必要があり，一般的かつ実効的な方法を決めるのは極めて難しい．独立計算である Horner 法を，別々のプロセッサでやった方が速いということも充分にありえるであろう．実際，$P \ge s$ の場合には，(SCR1), (SCR1') 及び (SCR3) に関する限り，点一個当たりの$\otimes$ の回数は $O(2^s)$ となるため，Horner 方の方が有利である．

6. まとめ

多項式の多点評価のための漸近的高速アルゴリズムにおいて，高速除算アルゴリズムのインライン展開をした結果，単純な数学的事実に基づいたオプティマイズ (Newton iteration の除去) が可能であることを示し，技巧的ではない実用的なアルゴリズムを示した．同様にして，与えられた多項式の次数を，Chinese remaindering の技巧を施しうるところまで落とすための処理については，除算アルゴリズムに隠された，共通式の再計算を除去する方法を示している．但し，ここで示したアルゴリズムの計算量は，元のアルゴリズムと同

等である．また，多項式の乗除算にフーリエ変換による高速アルゴリズムを用いることは必須だが，アルゴリズム全体における具体的な方法もまとめた．さらに，並列処理についても，通信コストも考慮に入れた方法を示した．いずれにしても，これらの有効性を主張することは，実際に検証を行ってからとすべきであろう．

謝　辞

本研究の実施にあたり，一部，平成7及び8年度文部省科学研究費補助金一般研究(C)/基盤研究(C)「数式処理算法のベクトル処理と並列処理及びその分散協調に関する研究」(課題番号 07680337) の補助を受けました．

参 考 文 献

[1] Aho, A. V., Hopcroft, J. E., and Ullman, J. D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] Borodin, A. and Munro, I.: *The Computational Complexity of Algebraic and Numeric Problems*, Theory of Computation Series, American Elsevier Publishing Company, Inc., 1975.

[3] Burke-Perline, T.: The parallel computation of $f(x)^{\frac{(p-1)}{2}} \bmod h(x)$ using *Sugarbush 1.1*, *Proceedings of PASCO '94* (Hong, H., ed.), Linz, Austria, 1994, 74–83.

[4] Cantor, D. G. and Kaltofen, E.: On fast multiplication of polynomials over arbitrary algebras, *Acta Informatica*, **28**(7), 1991, 693–701.

[5] Fujise, T. and Murao, H.: Parallel distinct degree factorization algorithm, Lakshman [10], 18–25.

[6] von zur Gathen, J. and Gerhard, J.: Arithmetic and factorization of polynomialz over $\mathbf{F}_2$, Lakshman [10], 1–9.

[7] von zur Gathen, J. and Shoup, V.: Computing Frobenius maps and factoring polynomials, *Computational Complexity*, **2**, 1992, 187–224.

[8] Jebelean, T.: An algorithm for exact division, *Journal of Symbolic Computation*, **15**(2), 1993, 169–180.

[9] Kaltofen, E. and Shoup, V.: Subquadratic-time factoring of polynomials over finite fields, *Proc. 27th Annual ACM Symposium on the Theory of Computing*, Las Vegas, Nevada, 1995, 398–406.

[10] Lakshman, Y. N., ed.: *Proceedings of ISSAC '96*, ETH Zurich, Swithzerland, 1996.

[11] 村尾: Speeded Chinese remaindering, 数式処理, **5**(1), 1996. (第5回日本数式処理学会大会(岩手大学，1995年6月)報告.

[12] Shoup, V.: Factoring polynomials over finite fields: asymptotic complexity vs. reality, *Proc. International IMACS Symposium on Symbolic Computation, New Trends and Development*, Lille, France, 1993, 124–129.

[13] Shoup, V.: A new polynomial factorization algorithms and its implementation, *Journal of Symbolic Computation*, **20**, 1995, 363–397.

[14] Weber, K.: The accelerated integer GCD algorithm, *ACM Transactions on Mathematical Software*, **21**(1), 1995, 111–122.