# ネットワーク型並列計算機上の数式処理*

2P－2

野村祐司，野田松太郎†
愛媛大学工学部情報工学科‡

## 1. はじめに

数式処理の並列計算に関する研究は広く進められている．一般的な方法は，因数分解など法演算を基本としたものに対する並列化である [1]．これは，複数の法を用いた計算を同時に実行することによって計算効率を上げ，アルゴリズムに大きな変更を加えることなく並列化を行うものである．数式処理の基本アルゴリズムの一つである，多項式の最大公約式 (GCD) の計算についても同様の手法が用いられる．

本稿では並列化をさらに進めるため，アルゴリズムの細部を変更する立場から GCD 計算の並列化について考察する．そして，並列計算機上で具体的なシステムを構築し，アルゴリズムの効率や問題点について検討する．

## 2. GCD の並列計算

モニックである 1 変数の有理係数多項式を考える．2 つの多項式

$$\begin{aligned} u(x) &= \textstyle\sum_{i=0}^{l} u_i x^i, \\ v(x) &= \textstyle\sum_{i=0}^{m} v_i x^i \end{aligned} \qquad (1)$$

に対して，拡張された Euclid 互除法を用いると，次の関係が成立する．

$$u(x)\,s(x) + v(x)\,t(x) = g(x). \qquad (2)$$

今 $g(x)$ の次数を $k$ とする．この時，(2) は次のように行列表現できる．

$$\begin{pmatrix} u_l & & & v_m & & \\ u_{l-1} & u_l & & v_{m-1} & v_m & \\ \vdots & \ddots & \ddots & \vdots & \ddots & \ddots \\ \vdots & & u_l & \vdots & & v_m \\ u_0 & \ddots & u_{l-1} & v_0 & \ddots & v_{m-1} \\ & & \vdots & & & \vdots \\ & u_0 \cdots & u_k & & v_0 \cdots & v_k \end{pmatrix} \begin{pmatrix} s_{m-k-1} \\ \vdots \\ s_1 \\ s_0 \\ t_{l-k-1} \\ \vdots \\ t_1 \\ t_0 \end{pmatrix} = \begin{pmatrix} g_{l+m-k-1} \\ \vdots \\ g_{k+1} \\ g_k \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \qquad (3)$$

$u(x), v(x)$ の係数行列は $l+m-2k$ 次正方行列である．ここで，その行列式を $R(k)$ とする．$R(k)$ は部分終結式と呼ばる．

(3) から $s(x), t(x)$ を求め，(2) に代入すると $g(x)$ が得られる．$u(x), v(x)$ の最大公約多項式 (GCD) $g(x)$ の次数は，$0 \le k \le \min(l, m)$ の範囲で，(3) に解が存在する最小の $k$ である．

文献 [2] は，2 分探索によって GCD $g(x)$ の次数 $k$ を求める方法を示している．この場合，(3) の解が存在するか否かによって探索領域を変える．解が存在するか否かは行列の正則性を調べる，すなわち $u(x), v(x)$ の部分終結式 $R(k)$ を求めることになる．[2] では，この部分終結式の計算を並列化している．

## 4. n 分探索を用いた GCD の並列計算

ここで，並列計算による n 分探索を用いた GCD の次数決定の方法を提案する．[2] では部分終結式の計算を並列化しているが，本稿では，探索領域を n 分割し，各分割点での複数の部分終結式を並列に計算する方法について考察する．

n 分探索を適用した GCD の次数を求めるアルゴリズムは

入力：$u(x), v(x)$
出力：$u(x), v(x)$ の GCD の次数 $k$
方法：
1. $A = 0$
   $B = \min(\deg(u(x)), \deg(v(x)))$
2. 探索領域 $[A, B]$ の $n$ 分割点で
   $u(x), v(x)$ の部分終結式を計算する．
3. 部分終結式が 0 でない最小の分
   割点を $k$ とする．
4. $A = k - (B - A - (n-2))/n$
   $B = k - 1$
   $A > B$ なら終了，
   $A \le B$ なら 2,3,4 を繰り返す．

となる．このアルゴリズムの詳細を次に示す．

$B-A-(n-2)$ が $n$ の倍数の場合

$s = (B-A-(n-2))/n$ とすると，分割点は

*Symbolic GCD algorithm on A Network Model Paralell Computer
†Yuji Nomura and Matu-Tarow Noda
‡Department of Computer Science, Ehime University

$$\begin{aligned} A_1 &= A+s, \\ A_2 &= A_1+s+1, \\ &\cdots \\ A_{n-1} &= A_{n-2}+s+1 \end{aligned} \tag{4}$$

である. 但し, 便宜上 $A_0 = A-1$ とする.

計算する部分終結式は $R(A_1), R(A_2), \cdots, R(A_{n-1})$ である. 部分終結式の性質から $R(A_{i-1})=0$, $R(A_i)\neq 0$ なら,

$$\begin{aligned} R(A_1) = R(A_2) = \cdots = R(A_{i-1}) &= 0, \\ R(A_i) = R(A_{i+1}) = \cdots = R(A_{n-1}) &\neq 0 \end{aligned} \tag{5}$$

となる. この時 $k=A_i$ であり, $A=A_{i-1}+1, B=A_i-1$ と取り直し, 探索領域を変更する. $A>B$ ならその時の $k$ が GCD の次数である.

$B-A-(n-1)$ が $n$ の倍数でない場合

この場合, 探索領域を完全に等分割することはできないが, 均等に近く分割にすることにより, 上と同様の等式を得ることができる.

### 3. 並列計算機上での実行結果

2. で述べたアルゴリズムを実際に具体化する. 簡単のため, ネットワーク型並列計算機であるトランスピュータを用いてシステムを構築する. そして, 分割点を増加すると計算効率が良くなるであろうことを確かめるため, 2 分探索 (n=2) と 4 分探索 (n=4) による GCD 計算の速度比較を行う.

次の問題を考える.

$$\begin{aligned} u(x) &= x^6+x^4+x^3+2x^2+x+2, \\ v(x) &= x^8+x^6+2x^2+2, \\ g(x) &= x^2+1. \end{aligned}$$

すなわち, $l=6, m=8, k=2$ である. アルゴリズム実行の結果, GCD を得るまでの計算時間として

2 分探索: 75.97 msec
4 分探索: 64.77 msec

を得る. これは期待通りの結果である.

次に, $l, m, k$ を変更し, いくつかの例でアルゴリズムの実行速度を見る (トランスピュータ).

| 次数 | | | 時間 (msec) | |
|---|---|---|---|---|
| l | m | k | 2 分探索 | 4 分探索 |
| 6 | 7 | 0 | 41.92 | 36.29 |
| 7 | 10 | 2 | 79.49 | 68.42 |
| 7 | 8 | 5 | 11.14 | 33.86 |
| 7 | 8 | 6 | 9.86 | 31.04 |

明らかに, $k=5,6$ など GCD の次数が大きい場合, 分割点を増加すると効率が低下する結果を得る. これには次の要因が考えられる. n 分探索を並列的に処理するには,

> 各分割点での計算量が等しい

あるいは,

> 目的の分割点での処理が終了した段階で, 他の分割点での処理を直ちに終了させることができる

といった条件が必要である. すなわち, $R(A_1)$ の次数が $R(A_i), i\geq 2$ よりも大きく, アルゴリズムの実行速度は, 計算量の大きい $R(A_1)$ 等, 高次の部分終結式の計算速度に依存している.

しかし, 実際には $R(A_1)$ など, 高次の部分終結式を常に計算する必要は無く, 低次の部分終結式 ($R(A_{n-1})$ など) の計算で目的を達する場合もある. この場合には, 他の部分終結式の計算を停止させ, 次のステップに移行する操作が必要であるが, 現在使用しているトランスピュータ上のアルゴリズムでは不可能である.

なお, 分割点の取り方を不均等分割に変えることも考えられるが, 十分な検討は行っていない.

### 4. まとめ

本稿では n 分探索による GCD のアルゴリズムについて考察し, トランスピュータを用いてアルゴリズムの具体化を行った. 低い次数の GCD に対してはここで述べたアルゴリズムが有効であることがわかる. しかし, 使用したトランスピュータの制限もあり, 高い次数の GCD に対しては並列処理の十分な効果を見ることはできなかった.

早急に行うべき課題は, いくつかの問題点を改善するとともに, 他の多くの GCD 計算のアルゴリズムとの比較検討を行うことである.

### 参考文献

[1] 野呂正行, 竹島卓: 並列処理による数式処理の試み, 数式処理通信 Vol.6 No.2 第 22 号, 1-6, 1989

[2] Joseph JáJá: An Introduction to Parallel Algorithms, Addison-Wesley Publishing Company, 415-419, 1992