数理解析研究所講究録
941 巻 1996 年 123-129

15.

モジュラー計算の擬似並列実行

佐渡　貴宏　(筑波大学)
秋川　友宏　(筑波大学)
佐々木　建昭 (筑波大学)

15.1 はじめに

本稿では、モジュラー計算として、r 個の互いに素な整数 $m_1,\ldots,m_r$ を法とする多項式の有理演算を扱うものとする。$m_1,\ldots,m_r$ は、中国剰余定理の法となるものでもよいし、$p_1,\ldots,p_r$ を互いに異なる素数として $m_1=p_1^k,\ldots,m_r=p_r^k$ なる形で $k=1\rightarrow 2\rightarrow\cdots$ と変えていく Hensel 構成の法でもよい。

モジュラー算法は、Hensel 構成を利用するものは GCD 計算や因数分解などに不可欠であるし [Sas1]、中国剰余定理を利用するものは Gröbner 基底 [Buch85] の計算などで極めて有用であることが示されている [Sas93]。

本稿で扱う型のモジュラー算法は並列計算に非常に適している。法 $m_1,\ldots,m_r$ に対する計算は互いに独立なので、r 台の並列に稼働する計算機で同時に実行できるからである。しかも、これらの計算は数係数の違いを除けばほとんど同じゆえ、r 個の計算はほぼ同期して終了する。したがって、これらの計算に関する限り、並列化による速度向上率は r となり、並列化は非常に有効である。

とは言うものの、並列計算機が十分に普及していない現状では、通常のユーザにとって並列計算を実行するのは容易ではない。そこで我々は“擬似並列化”なる方式を提案する。この方式によれば、多項式の内部表現を若干拡張するだけで、上記のモジュラー算法を逐次計算機上で実行してもかなりの速度向上が達成できる。実際、我々はこの方式を数式処理システム GAL でテストした結果、$r=10$ の場合、加減算で 4.4 ～ 8 倍、乗除算で 1.2 ～ 3 倍の速度向上が達成できることを確認した。

15.2 モジュラー計算の擬似並列化

簡単のため、1変数多項式で我々のアイデアを説明する。$\mathbf{Z}$ 上で本来計算されるべき多項式を $F(x) = a_n x^n + \cdots + a_0 x^0$ とし、それを整数 m_i を法として計算すれば $F_i(x)$ となるとする $(i = 1, \ldots, r)$。

$$F_i(x) = a_{i,n} x^n + a_{i,n-1} x^n + \cdots + a_{i,0} x^0 \qquad (i = 1, \ldots, r)$$

モジュラー演算は係数部に対してのみ行われることから、F_i の数係数は法 m_i ごとに異なるが、F_i の多項式としての構造は同じであり、加減乗算に関しては全く同じ構造の多項式を答えとして与える。除算においても、除多項式の主係数 a_n が $a_n \not\equiv 0 \pmod{m_i}$ である限り、計算は全く同じになる。以下では、$a_n \equiv 0 \pmod{m_i}$ となる場合、m_i をアンラッキーな法ということにする。

さて、$F_1(x), \ldots, F_r(x)$ の多項式としての構造が全く同じなので、これらを以下のように“多重係数”の一つの多項式にまとめることができる。

$$F_{1,\ldots,r}(x) = [a_{1,n}, \ldots, a_{r,n}] x^n + [a_{1,n-1}, \ldots, a_{r,n-1}] x^{n-1} + \cdots + [a_{1,0}, \ldots, a_{r,0}] x^0$$

こうすれば、1回の多項式演算の実行で、r 個の法に対する計算が同時に行えることになる。しかもそれは、既存の数式処理システムの多項式の内部表現を少し拡張するだけで、従来の逐次計算機上で実行できる。我々はこれをモジュラー計算の**擬似並列化**と名付ける。

アンラッキーな法の処理は以下のように行う。たとえば、法 m_2 がアンラッキーな場合、

$$F_{1,\ldots,r}(x) = [a_{1,n}, 0, a_{3,n}, \ldots, a_{r,n}] x^n + \cdots$$

となる。この場合でも、加減乗算はかまわず実行してよい。除算の場合には、m_2 に対する計算は実行せず、結果式のすべての多重係数の第2要素を nil とでもしておき、m_2 に対する計算は以後は無意味であることを明示しておく。こうしておけば、たまたまアンラッキーな法に遭遇しても、残りの $r-1$ 個の法に対する計算はそのまま実行できる。非常に稀な場合として、$m_1, \ldots, m_r$ がすべてアンラッキーな場合、$F_{1,\ldots,r}(x)$ の主係数が $[0, \ldots, 0]$ となって除かれ、表面上 $[a_{1,n-1}, \ldots, a_{r,n-1}]$ が主係数のように見えることがある。モジュラー計算では、最後に得られた答の正当性を確認しなければならないので、上記の非常に稀な場合はこの正当性のチェックに引っかかることになろう。

15.3 数式処理システムGALでのテスト結果

我々は、擬似並列化の効果を確認するため、GAL の多変数多項式演算ルーチンを若干修正してテストを行った。現在の GAL のルーチンは、数係数として各種の型を許しており、若干重いので、数係

数部では整数のモジュラー演算のみを行うパッケージを二つ作成した。

Package1: 係数は通常の整数で、整数 m を法とする計算を行う

Package2: 係数は多重係数で、整数の組 $[m_1,\ldots,m_r]$ を法とする計算を行う

モジュラー演算は、具体的には remainder$[x, m_i]$ で行った (これが最も速いため)。数係数以外の多項式の構造に関しては、GAL の内部表現をそのまま用いた。すなわち、**Package1** では GAL の係数部演算のうち、たとえば加算プロシジャplus(a,b) は次のプロシジャで置き換えた。

```
procedure mplus(a, b, m) == remainder(a + b, m);
```

同様に、$as = [a_1,\ldots,a_r], bs = [b_1,\ldots,b_r]$ を整数のリスト、$ms = [m_1,\ldots,m_r]$ を法のリストとするとき、**Package2** ではたとえば係数部の加算は次のプロシジャで実行する。

```
procedure mcplus(as, bs, ms) ==
          construct and return a list
         (remainder(a1 + b1, m1),...,remainder(ar + br, mr));
```

GAL の多項式表現は、変数として不定元や超越数のほか、代数的数や代数関数も含むかなり一般的なものであり、その演算は幾分重い。そのため、上記のパッケージによるデータは、擬似並列化の効果が最大限に出るようになっていることを注意しておく。

テストとして、$r = 5, 10, 15, 20, 25, 30$ に対して、密な (dense) 1 変数多項式での加減乗除算 (表 1) と密な 4 変数多項式で加減乗除算 (表 2)、及び疎な (sparse) 4 変数多項式の加減乗除算 (表 3) を行った。例として、テストには dense な 1 変数多項式の乗算には、

$$(17x + 41)^6 \times (85x + 12)^4$$

sparse な 4 変数多項式の減算には、

＼法の数	5	10	15	20	25	30
並列加算	5260	7659	9860	12140	14450	16690
逐次加算	24980	50780	74930	100340	125870	155760
並列減算	6190	8960	12000	14790	17730	20750
逐次減算	25560	51610	76830	101880	128750	156290
並列乗算	9560	17630	25780	34300	42070	50100
逐次乗算	15460	30880	46350	61680	77230	92770
並列除算	6760	12160	18060	26000	34040	39140
逐次除算	12690	24290	36650	49720	62410	75720

表 1　dense な 1 変数多項式演算の実行時間 (msec)

＼法の数→	5	10	15	20	25	30
並列加算	2240	3180	4170	5040	5980	6910
逐次加算	7030	14010	21010	28060	35710	42050
並列減算	2760	4030	5370	6470	7850	9040
逐次減算	7890	15760	23660	31550	39450	47320
並列乗算	1260	2270	3260	4270	5290	6340
逐次乗算	2060	4120	6160	8220	10300	12340
並列除算	1210	2230	3350	4450	5470	6590
逐次除算	1690	3300	5030	6720	8370	10090

表 2　dense な 4 変数多項式演算の実行時間 (msec)

＼法の数	5	10	15	20	25	30
並列加算	3350	3630	3750	3990	4240	4470
GC	700	330	660	660	330	670
逐次加算	15280	30430	46150	61220	75820	91460
GC	1980	4690	6960	9270	11270	13580
並列減算	3530	4020	4530	5190	5730	6190
GC	660	1000	990	1000	1330	1670
逐次減算	14650	29130	43670	58900	72870	87970
GC	1990	4670	6970	9270	11250	13590
並列乗算	880	1370	1840	2340	2790	3350
GC	330	0	330	670	330	660
逐次乗算	2110	4220	6330	8490	10570	12690
GC	330	1000	1320	1670	2330	2640
並列除算	6820	12380	18150	23850	29430	35260
GC	680	1330	1690	2040	2720	3410
逐次除算	7470	14870	22320	29750	37260	44620
GC	1010	1650	3000	4000	4150	5990

表 3　sparse な 4 変数多項式演算の実行時間 (msec)

$$(17x^{10}+24x^5-41y^{10}-29y^5+18)-(85v^{10}-14v^5+62w^{10}+53w^5-12)$$

といったような多項式演算を用いた。これらの計算を多数回行い、その全計算時間を計測した。

15.4 テスト結果に対する考察

多項式の四則演算をコンピュータで行うときの全操作を2つに分割する：

操作 C: 多項式の数係数に関する演算と操作 (0 判定など)

操作 S: 多項式の構造に関する操作 (構造を調べる、新たに作るなど)

多項式の四則演算をコンピュータで行ったとき、操作 C と S に要する計算時間をそれぞれ $T_{\mathrm{C}}, T_{\mathrm{S}}$ とする。すると、r 個の法に対する計算を逐次的に実行したときの全計算時間 $T_{\mathrm{seq}}(r)$ は次式で与えられる。

$$T_{\mathrm{seq}}(r) = r\cdot(T_{\mathrm{C}}+T_{\mathrm{S}}) \tag{1}$$

一方、同じ計算を擬似並列実行した場合の全計算時間 $T_{\mathrm{qp}}(r)$ は

$$T_{\mathrm{qp}}(r) = r(1+\alpha)T_{\mathrm{C}}+T_{\mathrm{S}} \tag{2}$$

と評価してよかろう。ここで、α は数値で、係数部をベクトル化することによる係数部操作の手順の増加をあらわす因子である。これらより、擬似並列化の速度向上率として次式を得る。

$$\frac{T_{\mathrm{seq}}(r)}{T_{\mathrm{qp}}(r)} = \frac{T_{\mathrm{C}}+T_{\mathrm{S}}}{(1+\alpha)T_{\mathrm{C}}+T_{\mathrm{S}}/r} \tag{3}$$

$$\simeq \frac{T_{\mathrm{C}}+T_{\mathrm{S}}}{(1+\alpha)T_{\mathrm{C}}} \qquad \text{if } r\gg 1. \tag{4}$$

すなわち、$T_{\mathrm{S}} \gg T_{\mathrm{C}}$ であれば、擬似並列化による効果が非常に大きくなることがわかる。表1、2を見ると、加減算に関しては、1変数多項式であれ多変数多項式であれ、擬似並列化の効果が大きく出ていることがわかる。しかしながら、乗除算に関しては、効果は2倍程度で、大きくはない。(4) 式によると、T_{S} が T_{C} に比べてそれほど大きくはないのではないかと思われる。

この予想を裏付けるため、15.3節に与えた2つのプロシジャ`mplus` と`mcplus`, および乗算に対する同様の2つのプロシジャ`mtimes` と`mctimes` の実行時間を計測した（ただし、`mplus` と`mtimes` に対しては r 個の法 $m_1,\ldots,m_r$ に対してすべて実行したものの和とする)。`mplus` あるいは`mtimes` の実行時間が (1) 式の T_{C} に対応し、`mcplus` あるいは`mctimes` の実行時間が (2) 式の $(1+\alpha)T_{\mathrm{C}}$ に対応する。表4は計測結果を示している。この表より、乗算における (2) 式の $(1+\alpha)$ に対応する値は加算における $(1+\alpha)$ に対応する値に対して 1.2 倍ほどであることがわかる（加算といえども`remainder` で剰余をとっていることに注意)。したがって、もしも (1) 式あるいは (2) 式の T_{S} が乗算と加算に対

	加算	乗算
逐次実行	1300	1660
擬似並列	1270	1980

表 4　$r = 5$ における係数演算プロシジャの実行時間 (msec)

して同じなら、擬似並列化による乗算の速度向上率は加算のそれの 4/5 程度になるはずである。ところが、表 1, 2, 3のデータは、擬似並列化による乗算の速度向上率はこれよりはるかに低いことを示している。このことは、乗算の場合、T_C に比べて T_S の値が小さいと結論せざるを得ないが、これは GAL の多項式乗算プロシジャのアルゴリズムに起因するものであると考えられる[1]。したがって、多項式乗算ルーチン内で一般的な加算ルーチンを用いた数式処理システムであれば、擬似並列化の効果が乗除算において加減算の 4/5 程度になるであろう。

また、密な多項式と疎な多項式に対するデータを比較すると、疎な多項式に対して擬似並列化の効果がより大きく出ているが、これは当然である。なぜなら、多項式演算における最も重要な操作は同類項の同定であるが、これには疎な多項式の方がより多くの手順を要するからである。しかしながら、擬似並列化の効果は疎な多項式にしても大きくは向上しなかった。これも前述の理由によるものであろう。

最後に、擬似並列化の目に見えない効果として、同じ計算を r 回逐次実行するよりも内部構造を小さくできる分メモリ消費が少なくてすみ、GC(ガーベッジ・コレクション) 時間が小さくなる事を指摘したい。表 3には、演算時間に加えて GC 時間も示されているが、それによると擬似並列化により GC 時間が $r = 20$ で 1/2〜1/10 以下に減少していることが分かる。

[1]GAL における多項式乗算は以下のように行われている。乗じる多項式を $S = s_1 + \cdots + s_m, T = t_1, \cdots, t_n$ とする。ただし、s_i と t_j は主変数に関する項である。まず、$P = s_1T = s_1t_1 + \cdots + s_1t_n$ に対応する内部表現を作る。次に、$P_2 = s_2T = s_2t1 + \cdots + s_2t_n$ に対応する内部表現を作り、P に足し込むのであるが、その際、P の内部表現に体するリストを壊しながら (具体的に言うと Lisp 関数 rplaca と rplacd を用いて) この操作を実行する。以下、同様に $P_i = s_it_1 + \cdots + s_it_n$ $(i = 3, \ldots, m)$ を P に足し込むのである。リストを壊しながら P に足し込むこの操作は、通常の加算操作よりもより効率的である。乗算は、i) 各項の積 s_it_j を作る操作、ii) s_it_j を P に足し込む操作からなるが、同類項の同定に時間がかかるので操作 ii) が計算時間の大半を占める。したがって、上述の方法による加算は mn 個の項を通常の多項式加算で加えるよりもより効率的であると言える。

参考文献

[Buch65] B.Buchberger, *An Algorithm for Finding a Basis for the Risidue Class Ring of a Zero-dimensional Polynomial Ideal*, (German.) Ph.D Thesis, Math. Inst., Univ. of Insbruck,Austria, 1965.

[Buch85] B.Buchberger, *Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, in Multidimensional System Theory*, N.K.Bose,ed., D.Reidel Publ. Comp., 1985. pp.184-232.

[Sas1] 佐々木建昭, 今井浩, 浅野孝夫, 杉原厚吉 著. 計算代数と計算幾何. 岩波講座応用数学 5 [方法 9]. 岩波書店, 1993.

[S&T89A] T.Sasaki and T.Takeshima, *A Modular Method for Gröbner-basis Construction over* **Q** *and Solving System of Algebraic Equations*, Journal of Infomation Processing, Vol. 12, No. 4, 1989.

[S&T89B] 佐々木建昭, 竹島卓, グレブナー基底の並列計算と連立代数方程式, 情報処理学会論文誌, 第 30 巻, 1555-1561, 1989.

[H&S95] 秋川友宏, 佐々木建昭, Gröbner 基底のモジュラ構成法とその擬似並列化 (その 1), 1995.

[S&S95] 佐渡貴宏, 佐々木建昭, Gröbner 基底のモジュラ構成法とその擬似並列化 (その 2), 1995.