素数判定プログラムGMP-ECPPの高速化手法の検討

指導教員 : 市川 周一　　学籍番号 : 093716 熊谷 大地

1　はじめに

現在，広く普及している RSA 暗号は，桁数が大きい合成数の素因数分解問題が困難であることを安全性の根拠としている．RSA 暗号では，巨大な二つの素数 p, q の積 $n = pq$ を鍵として用いる．一般に素数を直接生成することは難しいので，乱数を発生させ，それが素数か否かの判定を行い，鍵に用いる素数を探すという方法を取っている．

暗号通信は電子商取引や電子署名の普及に伴い日常生活に身近なものとなっており，そのため素数判定法は情報通信において重要な研究分野である．

そして，セキュリティ技術の社会的重要性が高まっている昨今，より効率的なアルゴリズムが求められている．

本研究は，楕円曲線法（Elliptic Curve Primality Proof: ECPP）をオープンソース実装した素数判定プログラム GMP-ECPP について，性能評価を行い実行速度を向上させることを目的とする．

2　GMP-ECPP [1]

GMP-ECPP で用いられている素数判定アルゴリズム ECPP は，楕円曲線上の点の演算規則を利用した素数判定アルゴリズムである．

整数 a, b に対して素数 p を法とする楕円曲線 $E(p)$ とは，$Y^2 \equiv X^3 + aX + b \bmod p$ の解 $(x \bmod p, y \bmod p)$ の全体に仮想的な一点 $\mathcal{O}$ を付け加えた集合に，ある独自の演算を定義することで得られる [2].

整数 N を法とする楕円曲線 $E(N)$ の位数を $m = \#'E(N)$ とする．ここで位数とは，群 G における元の数を意味する．a, b をランダムに動かしながら m を計算し，$m = 2\times$ “ 大きな未確定素数 ”の形になっているか否かを判定するという処理を繰り返し行う（Goldwasser-Kilian の判定方法）この判定方法は，遅いため実用的ではないので，先に m を指定し，それから $m = \#'E(N)$ となる楕円曲線を構成するというアプローチを取るアルゴリズムが Atkin-Morain テストである．

GMP-ECPP では，Henri Cohen の著書にあるアルゴリズム 9.2.4（Atkin-Morain テスト）[3] を実装している．

また，GMP-ECPP は巨大な桁数の数値での演算を行うために，多倍長演算ライブラリである GMP を利用する．本研究では，GMP のインストール方法を検討し，最も高い性能が出せる状態で測定を行った．

3　プロファイリング

プログラムの性能を改善するためには，実行速度のボトルネックになっている箇所を発見し，修正する必要がある．そこで，ソフトウェアの実行時間を関数単位で解析するツールである gprof を用いて，GMP-ECPP に 80 桁の素数を入力した時の実行時間を解析した（図 1）．図 1 に，各関数が実行時間全体に対して占める割合を示す．ここに示した関数の中でも特に，全体の約 80%を占める multiply，addition_2，addition_1 の三つの関数について，実装を改良した．

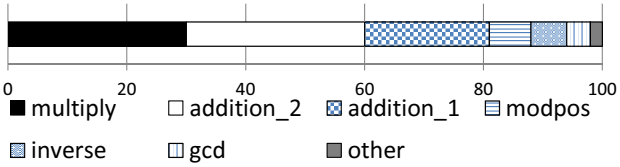

図 1: プロファイリング結果

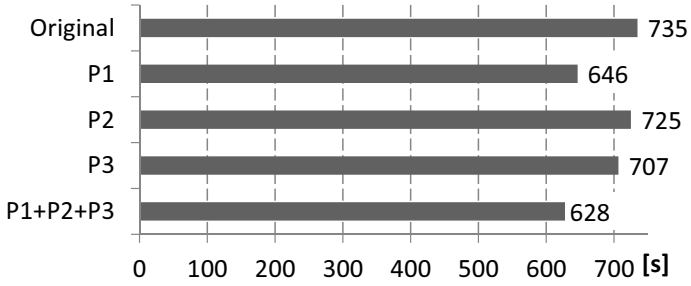

図 2: 実行時間の比較

4　改良方法，改良結果

改良内容は大別すると次の 3 つに分けられる．

P1：ローカル変数の生成に関するオーバーヘッドをなくすために，グローバル変数を使用した．関数内で一時的に使用するだけの変数は毎回生成するのではなく，グローバル変数を使い回すことで，高速化を図った．この改良を施すにあたって，cflow という関数の呼び出し階層を解析するツールを用いて，変数のスコープを調べ，ローカル変数からグローバル変数に変更しても，動作に影響が出ないことを確認した．

P2：プログラム内で無駄な関数呼び出しを行っていた部分を改良した．

```
m=modpos(modpos(B.y-A.y,n)*inverse(modpos(B.x-A.x,n),n),n);
```

この部分を次のように改良した．

```
m=modpos((B.y-A.y)*inverse((B.x-A.x),n),n);
```

modpos は，第二引数を法とする第一引数の剰余を返す関数である．第二引数が同じである場合，modpos の計算結果を再び modpos に渡すのは無駄なので削除した．

P3：関数呼び出しにかかるオーバーヘッドをなくすために，関数の処理内容を直接書くこととした．具体的には，addition_2，addition_1 の処理内容を multiply に埋め込んだ．

元のプログラムと，これらの改良を施したプログラムとの実行時間を 3 回ずつ測定し，それらを比較した結果が図 2 である．

測定したのはプロファイリング時と同様に，80 桁の素数を入力した時の実行時間である．元のプログラムに対して，3 つの改良方法を組み合わせたプログラムである **P1**+**P2**+**P3** の実行時間を比較すると，約 15%の高速化に成功したことが分かる．

5　おわりに

本研究では GMP-ECPP の実装部分を改良し，高速化に成功した．

GMP-ECPP は現在も進行中のプロジェクトであり，プログラムの開発と保守が継続して行われている．プログラムのバージョンアップ毎に大きな改変が行われているため，新たな改良手法が必要とされることが想定される．

今後は Atkin-Morain テストのアルゴリズムを理解し，改良を加えていくことで，さらなる性能向上が達成できると考えられる．

参考文献

[1] GMP-ECPP project: GMP-ECPP. `http://gmp-ecpp.sourceforge.net/`.

[2] 木田祐司, 牧野潔夫: 素数判定アルゴリズム, 情報処理, Vol. 34, No. 2, pp. 150–156 (1993).

[3] Cohen, H.: *A Course in Computational Algebraic Number Theory*, Graduate Texts in Mathematics (1993).