素因数分解プログラム GGNFS の性能測定

指導教官：市川周一　　学籍番号：053710 川辺 裕喜

1 背景と研究目的

認証や電子署名には公開鍵暗号が広く使われており, RSA 暗号が主に用いられている．RSA 暗号の安全性の根拠は巨大数の因数分解問題の困難さである．因数分解の高速化は RSA 暗号の安全性の予測において重要な意味を持つ．

素因数分解には試し割り法, モンテカルロ法, 楕円曲線法, 二次篩法など多数のアルゴリズムが存在する．その中で一般数体篩 (ふるい) 法 (GNFS)[1] は 100 桁以上の数を分解するには, 現在知られる最速のアルゴリズムである．2005 年には 200 桁の数 (rsa200) が一般数体篩法で分解された [2]．

オープンソースの GNFS ソフトウェア GGNFS の性能測定と改良を行うことは, RSA 暗号の安全性を検討する上で有意義である．

2 GGNFS の実装

GGNFS[3] とは, 一般数体篩法を実装したオープンソースのソフトウェアである．おおよそ以下の 6 つの処理に分かれている．(() 内は対応するプログラム名)

多項式選択 (`polyselect`) あらかじめ多項式のデータを与えることでこの処理は省略可能．今回はその方式を取った．

Factor base **構築** (`makefb`)

格子篩 (`gnfs-lasieve4I12e`) 分解対象の桁数に応じて適切なプログラムを選択するが, 今回はこのプログラムのみ使用した．

関係処理 (`matbuild`, `procrels`)

線形代数処理 (`matsolve`)

最終処理 (`sqrt`)

この中で, 篩処理が最も処理時間がかかることが理論的にも経験的にも知られている．また, 篩—関係処理—線形代数処理の一連の処理と最終処理は繰り返し実行される．

3 処理時間の測定

測定は, Athlon XP 3200+ (2200MHz), 主記憶 1GB の PC 上で, RedHat Linux 9, gcc 4.2.2 を用いて行った．測定対象は GGNFS の tests ディレクトリから 3 つを選んだ (表 1)．

表 1　測定対象 (ただし, 数は概数)

保存フォルダ	数	適用アルゴリズム
non-monic	6.34×10^{67}	特殊数体篩法
snfs_small	1.06×10^{37}	特殊数体篩法
tst250	3.68×10^{75}	一般数体篩法

3.1 各プログラムの実行時間

第 2 節で示した各プログラムの実行時間 (user 時間) を調べた．結果を表 2 に示す．いずれの場合も篩処理を行う `gnfs-lasieve4I12e` が最も処理時間が長い．

3.2 篩処理部分のプロファイル

さらに, 篩処理 `gnfs-lasieve4I12e` の中のどの部分が最も実行時間が長いか調べた．tst250 に対する測定結果の一部を表 3 に示す．他の 2 つの数でも共通して実行時間が長いのは `lasieve`, `lasched`, `medsched` の 3 つであった．

3.3 高速化オプションの効果

GGNFS を Make する際にはコンパイルオプション以外の, 独自の configuration オプションが用意されている．

x86 **のアセンブリ言語を用いた高速化**　モンゴメリ乗算などがアセンブリ言語で書かれている．

Block Lanczos **アルゴリズムの高速化**　アセンブリ言語を用いることによる高性能化が図られている．

しかし, アセンブリ言語は C 言語に比べ互換性, 可読性に欠ける．コンパイラの進歩によりアセンブリ言語によるプログラムと C 言語によるプログラムとの処理速度の差は小さくなっている．アセンブリ言語の使用によりどの程度の性能向上があるのか, 調べた結果を表 4 に示す．ただし, 表の各構成は以下の通りである．

x86_common-mmx　モンゴメリ乗算などの部分, Block Lanczos アルゴリズム両方にアセンブリ言語を使用

common-mmx　Block Lanczos アルゴリズムのみアセンブリ言語を使用

x86_common-no-mmx　モンゴメリ乗算などの部分のみアセンブリ言語を使用

common-no-mmx　アセンブリ言語を使用しない

x86 の場合は最大 13% 実行時間が短縮された．Block Lanczos アルゴリズムに対してはほとんど実行時間の短縮は見られなかった．

4 今後の課題

今回は性能の測定で終わってしまったが, 処理性能の改善を試みたい．まず考えられるのは, 処理時間の長い `gnfs-lasieve4I12e` の性能改善である．

表 2　各プログラムの測定結果 (単位 [秒])

プログラム名	non-monic	snfs_small	tst250
makefb	0.15	0.72	1.24
gnfs-lasieve4I12e	372.88	193.87	2505.94
procrels	64.41	68.37	167.47
matbuild	4.99	61.47	41.03
matsolve	1.47	8.82	50.62
sqrt	108.65	122.66	66.62
合計	552.55	455.91	2832.92

表 3　tst250 の測定結果 (6 回の実行の合計)

関数名	実行時間 (s)	実行時間に占める割合 (%)
lasieve	1015.95	43.5
lasched	501.28	21.5
medsched	397.94	17.0
その他	420.07	18.0
合計	2335.24	100

表 4　高速化オプションに関する測定結果 (単位 [秒])

構成	non-monic	snfs_small	tst250
x86_common-mmx	552.55	455.91	2832.92
common-mmx	605.00	506.49	3276.65
x86_common-no-mmx	552.79	455.91	2843.63
common-no-mmx	605.06	505.53	3200.96

参考文献

[1] A. K. Lenstra and H. W. Lenstra, Jr., editors, The development of number field sieve, LNM 1554, Springer-Verlag, 1993.

[2] T. Kleinjung, rsa200, http://www.loria.fr/~zimmerma/records/rsa200

[3] C. Monico, GGNFS - A Number Field Sieve implementation, http://sourceforge.net/projects/ggnfs/