素因数分解
Prime Factorization

1012022　今井啄人　Takuto Imai

1　プロジェクトの目的

本プロジェクトの目的はできるだけ大きい素因数を見つけることである．現在公開鍵暗号において RSA 暗号がよく使われている．RSA 暗号とは桁数の大きな素因数の分解の困難性を利用した暗号方式である．RSA 暗号の安全性は素因数分解の困難性によって評価できる．その素因数分解の最も優れた解法の一つに楕円曲線法 (ECM) がある．私たちは楕円曲線法による素因数分解プログラムの高速化とアルゴリズムの改良を目指す．また, 楕円曲線法を用いて大きい素因数を見つけることを目的とする ECMNET というコンペティションがある [1]．現在記録されている素因数よりも大きい素因数を見つけることで ECMNET のランキングに誰でも名前を載せることができる．私たちはランクインしている素因数を参考にし, より大きな素因数を発見したい．

2　課題

- 一つの数を計算するのにかかる時間の短縮①
- 巨大な合成数を素因数分解する際にかかる CPU への大きな負荷の削減②

この 2 つの課題を克服すべくグループをプログラミング班と理論班の 2 つのグループに分けて課題解決に勤めた．

3　活動内容

3.1　前期

本プロジェクトでは, 楕円曲線法での素因数分解を実装し, ECMNET に挑戦することが目標である．前期には楕円曲線法のアルゴリズムを理解するため, メンバー全員で基礎学習を行った．担当教員の白勢先生に作成していただいたカリキュラムに沿って計算問題や証明問題に取り込んだ．実際に学んだ内容は大きく分けて以下の通りである．

- 群, 環, 体
- ラグランジュの定理
- Z/nZ
- 楕円曲線

次に, 英語で記載されたサイトである ECMNET を利用するにあたり, 私達は ECMNET の和訳を行った．これにより私達は ECMNET の内容を理解するだけではなく, 楕円曲線の歴史, 楕円曲線法の優秀さ, ランキングに掲載されるための素因数の桁数などを学ぶことが出来た．私達はランキングに名前を掲載させることができる素因数の桁数を把握し, 素因数分解を行う上で ECMNET を参考にすることが出来る様になった．

以上が, 前期の主な活動である．

3.2　理論班

理論班の活動目的は楕円曲線法の計算量を削減することである．計算料を削減することによってプログラムの高速化につながる．まずはじめに, 様々な計算手法を学び計算量の削減に一番適切なものはなんなのかを探した．逆元の計算は乗算に比べて約 12 倍もの時間的なコストがかかる．よって如何に逆元計算の回数を減らすか, ということに重点を置いて活動した．次に, 楕円曲線法にシステム上用いる手法について, 数学的な理解を深めるため論文の輪読に勤めた．輪読する際に使用した論文は”「楕円曲線加算公式の改良」永井善孝 (公立はこだて未来大)・伊豆哲也 (富士通研)・白勢政明 (公立はこだて未来大)20130523”である [3]．論文を読み進める上で我々は射影座標系に重点を置き, 勉強を進めることした．射影座標系とは 2 次元の直交座標に対し, 3 つの変数を用いて表現する手法である．射影座標系では座標のデータを分子と分母に分けて別々に保持しているので逆元計算を行わない．よって射影座標系を用いることにより, 計算量を約 1 割削減することができた．さらなる高速化を目指すために, より計算量を削減する手法はないのかと試行錯誤し, 加算公式の改良を試みた．座標の分母の値を変えるなどしてみたがどれも高速化には至らなかった．

3.3 プログラミング班

3.3.1 目的

プログラミング班の目的は, 前期に作成した楕円曲線法プログラムを元に, プログラムをより高速に実行できるよう改良することである.

3.3.2 GMP 勉強会

GMP の演算はすべて GMP が提供している関数を用いるため, 楕円曲線法を行うプログラム開発のためには GMP での演算になれる必要があった. まず初めに, 全員で GMP のインストールを行った. インストールする際, ライブラリのインストールを行うのは初めてと言う人が多かったため, インストール方法の確認も行った. その後, どのように演算を行うのかを知るため, GMP のホームページに掲載されている API をプログラミング班全員で読んだ. API を読んだ後に, 加算や乗算, 剰余算などの基本的な演算を用いた計算プログラムを書き, GMP での演算の練習を行った.

3.3.3 プログラム仕様書

プログラムの仕様を決定するため, プログラム仕様書を作成した. 関数の型や引数, 構造体, マクロといった情報が掲載されている. 誰でも参照できるように, google drive 上に作成した.

3.3.4 コーディング規約

複数人でのコーディングを円滑に行うためにコーディング規約を定めた. プログラムの命名規則やコーディングスタイルなどを定めることで, 可読性が向上した.

3.3.5 テスト

目的

テストではコーディングした関数が正しく動作しているかの確認をテスト用プログラムを用いて行った. また, 前期に作成したプロトタイプの関数と比較して高速化が出来ているか確認するためのテスト用プログラムも作成した. テストした関数は点の加算を行う normal_add(), double_add(), improved_normal_add(), jacobian_double() と, これらの関数を用いた scalar() である.

速度比較

scalar() を直交座標系から射影座標系に変えたことで速度にどのぐらい変化があったかを見るためのプログラム scalar_time を作成した. この scalar_time は scalar() のテストに使用したテスト用プログラムを改良して作った. 射影座標系と直交座標系での scalar() の計算時間を time. h の関数 clock() を使用して計測した. この結果, 射影座標系での scalar() の方が直交座標系での scalar() よりも最大２０%高速化していることが分かった.

3.3.6 実装

並列実行を行わず, 逐次実行するプログラムを製作していた段階では gcc でコンパイルしていた. しかし, mic アーキテクチャ (Xeon Phi のアーキテクチャ) 向けにコンパイルするには, インテルが開発したコンパイラである icc を用いる必要がある. icc は基本的に gcc と同様のオプションを利用することができる. ライブラリの指定は"-l"オプション, ライブラリの場所の指定には"-L"オプションを指定する. また, mic 向けにコンパイルするオプションである"-mmic"オプションと OpenMP を利用するための"-openmp"オプションを用いた. icc にこれらのオプションを付加することで, Xeon Phi 上で並列実行可能なプログラムをコンパイルすることが出来た. また, OpenMP を利用するにあたって参考文献を参照した [2].

3.3.7 性能評価

楕円曲線法は因数の大きさに依存する因数分解法であるため, 2 つの素数をかけ合わせた合成数を因数分解することで, 性能を評価することができる. 36 桁の素数とそれ以上の素数をかけ合わせた合成数を因数分解を行った結果, 36 桁の素数を発見することが出来た. この 36 桁の素数を発見するのにかかった時間は 1 時間程度であるため, より長時間プログラムを稼働することで, より大きい素因数を見つけることが期待できる.

3.3.8 まとめ

ECM プログラムを高速に実行するために, 大きく 2 つのことをした. まず一つ目に, 射影座標系での計算をモジュール化した. 二つ目に, Xeon Phi 上での並列処理を可能にした. この二つによって, 処理時間が最大 20% 高速化することができた. 時間の都合上, 実装までにいたらなかった手法があったが, プログラミング班の目的は概ね達成できたと言えるだろう.

3.4 発表

3.4.1 中間

中間発表には Prezi というソフトでのプレゼンとホワイトボードを用いた簡単な因数分解の実演を行った. 本プロジェクトでは聴講者に飽きさせない発表を目指した. Prezi は多様なアニメーションや視覚効果があるため採用した. Prezi でのプレゼンでは数学用語を実例を交えながら解説し, ホワイトボードを用いた楕円曲線法での素因数分解の実演では小さな合成数である 15 を実際の手順通りに因数分解した. 実際の発表では, 発表場所が 3 階モールで声が通りにくく, 聞き取りづらいものになってしまった. また, 発表練習の時間があまり取れず原稿を読みながらの発表になってしまった.　発表後アンケートを集計した結果, 評価は発表技術, 発表内容共に約 7 割だった. 聴講者に飽きさせないことを意識した Prezi とホワイトボードを用いた発表は概ね好評だったが, 一部では視線の移動が激しい, という意見もあった.

3.4.2 最終

最終発表では, 前期中間発表アンケートでの, 視線の移動が激しい, という意見を考慮し出来るだけ簡潔明瞭な発表を目指した. なのでシンプルなスライド一つでの, 出来るだけ専門的な内容の無い, 専門知識を持たない人でも分かり易い発表を目指した. その際どうしても解説しなければならないような用語に関しても, 大学のカリキュラム内での内容であれば, 説明はしない, という風に取り決めた. 前期での反省を活かし, 大きな声で聞き取りやすい発表にはできたが, また原稿を過度に見ながらの発表となってしまった.　発表後アンケートを集計した結果, 発表技術は約 6 割 6 分, 発表内容は約 7 割だった. 原稿を過度に見ている, 用語がわからない, などの意見が目立った.

4 成果

4.1 理論班

掛け算の計算コストを M とし, 直交座標系と射影座標系の計算コストを比べると, 直交座標系での計算コストは掛け算の計算量の約 14.8 倍, 射影座標系での計算コストは約 13.6 倍になった. よって, 計算コストを 1 割削減することができた.

4.2 プログラミング班

プログラミング班では前期に作成したプロトタイプの改良を行っていた. 主に点の座標系とプログラムの並列化, (k!)P の計算のところが変更されている. プロトタイプでの点は直交座標系で表されていたが, 改良されたプログラムでは射影座標系で点を表している. これにより, 逆元計算を減らし計算コストを削減することが出来る. 実際にプログラミングをして実行した結果では最大 20% の高速化が見られている. さらに, 並列処理も実装した. このために, マルチスレッド並列プログラミングのための API である OpenMP を使用した. プログラムを並列化することで素因数を発見する確率を向上することができる. 楕円曲線法は楕円曲線を決定すると, 独立してアルゴリズムを実行できる. そのため, funecm では異なる楕円曲線を各スレッドへわたして並列実行している. (k!)P の計算では, k!で合成数の累乗計算まで行っているため無駄が多かった. そのため, 素数の累乗のみを計算するアルゴリズムを funecm では利用している. また, プログラムの使用方法が誰にでも分かるようにマニュアルの作成も行った. マニュアルは動画とドキュメントの二つがある. 動画は電源の繋げ方から, プログラムの立ち上げ方, プログラムの終了の仕方まで解説している. ドキュメントでは, 動画より詳細に内容を解説している. Xeon Phi の概要から, プログラムへのコマンド, オプションの使い方を説明している. 加えて, 前期に ECMNET を訳して分かったことを書いた.

5 まとめ

5.1 前期活動結果

前期は主に, 楕円曲線法のアルゴリズムを理解するための知識が不足していたので, 基礎学習を行った. そして, 楕円曲線法での素因数分解プログラムのプロトタイプを作成し, 21 桁の素因数を確認することが出来た.

5.2 前期の問題点と後期への課題

前期に作成した ECM プログラムでは, ECMNET のランキングに掲載されているような巨大な素因数を見つけることは難しいと判断した. プログラムを高速化するためには, アルゴリズム, プログラムの両方で改良する必要があった. 後期は, 楕円曲線法 (ECM) をより早くするための理論を模索する理論班と, それを実装し並列化するプログラミング班に分かれて活動することにした.

5.3 後期活動結果

後期は, 前期作成した ECM プログラムを改良するために, 理論班とプログラミング班に分かれ活動した.
理論班は楕円曲線法の計算量を削減することを目的に活動した. 具体的には直交座標系での計算量より小さくなる手法を探した. 論文を輪読し射影座標系やヤコビアン座標系を学んだ. それらの計算手法を手計算し検証した結果, 射影座標系を用いることによって加算の計算量が減らせることがわかった. 成果として楕円曲線法の加算の時間的なコストを約 1 割削減することができた.
プログラミング班は, 理論班からもらった射影座標系を使った計算をプログラミングした. 加えて, Xeon Phi 上で並列処理を行うために OpenMP を使用した. 報告書執筆段階では 36 桁の素因数を発見することが出来た.

5.4 今年度の成果

素因数分解プロジェクトは, 今回が初年度であった. 一年間の活動を通じて, 我々は以下のことを達成した.
まず一つ目に, 前期に基礎学習を通じて楕円曲線法のアルゴリズムを理解することが出来た. 二つ目に, 楕円曲線法による素因数分解プログラムを作成した. 三つ目に, 楕円曲線法のアルゴリズムの改良を行った. 四つ目に, ECM プログラムを Xeon Phi 上で並列実行を可能にした. 本プロジェクトの目標は, 楕円曲線法プログラムの高速化と, ECMNET のランキングへの挑戦であった. 前者は, 射影写像系を採用し, Xeon Phi 上での並列処理を行うことで, 最大 20% 高速化することが出来た. ECMNET のランキングに名前を載せるには, 執筆時点で 64 桁以上の素因数を発見する必要があるが, 36 桁までの素因数しか発見できなかった.

5.5 反省点

本項では, 素因数分解プロジェクトの反省点を述べていく. まず一つ目に, プログラムを実行する時間が十分ではなかったことが挙げられる. これは, プログラムを完成させる予定が 2 週間遅れてしまったことが原因である. プログラミング班全員がグループでプログラミングを行った経験がなかったために, どのように作業を分担し進めたらいいか分からず, コーディングが滞ってしまうことがあった. 加えて, Xeon Phi 上で並列処理を行うために OpenMP を組み込んだ際, プログラムが正しく動作せず, その調整に時間がかかってしまった. その結果, プログラムが期待通りに動くようになったのは 12 月に入ってからであった. 一般に, 大きな素因数を発見するには数日間プログラムを動かし続ける必要がある. 二つ目に, 楕円曲線法のアルゴリズムを改良する際に, 参考にした文献の量が足りなかった. 一つの文献の内容を確認し, その手法が正しいことを証明するのに時間がかかってしまった. 三つ目に, 時間の都合上, プログラムに実装できなかった手法がいくつかあった. 例えば, ヤコビ座標系を使った楕円曲線上での計算, 高速フーリエ変換, 誕生日攻撃をつかったアルゴリズム等がある. 実装することが出来れば, 更にプログラムを高速化することができたかもしれない.

6 展望

我々素因数分解プロジェクトが来期よりよい結果を出すためには因数分解プログラム「funecm」の改良をする必要がある. 理論班は射影座標系を用いたが, 他の手法を試みて最善の手法を模索し, プログラミング班の手助けをする. プログラミング班は実装できなかったヤコビアン座標系の 2 倍算と高速フーリエ変換のプログラミング・実装を早い段階から行う. そして, Xeon Phi でのプログラム実行時間を多く取り, 巨大な素因数を見つけ, ECMNET のランクインを目指す.

参考文献

[1] ECMNET
`http://www.loria.fr/~zimmerma/records/ecmnet.html`
（最終アクセス 2015 年 1 月 7 日）

[2] ジム・ジェファース/ジェームス・レインダース, ”インテル Xeon Phi コプロッセサー ハイパフォーマンス・プログラミング”, 株式会社 カットシステム, 2014

[3] 永井善孝・ 伊豆哲也・ 白勢政明, ”楕円曲線の加算公式の改良”, 信学技報, vol. 113, no. 53, pp. 39-46, 2014