並列 Gauss Sieve アルゴリズムを用いた 128 次元イデアル格子の最短ベクトル問題の求解

石黒 司† 清本 晋作† 三宅 優† 高木 剛‡

† 株式会社 KDDI 研究所
356-8502 埼玉県ふじみ野市大原 2-1-15
tsukasa, kiyomoto, miyake@kddilabs.jp

‡ 九州大学マス・フォア・インダストリ研究所
819-0395 福岡市西区元岡 744
takagi@imi.kyushu-u.ac.jp

あらまし 格子暗号の安全性は最短ベクトル問題 (SVP) の困難性に基づいている．2010 年に Micciancio 等によって高速な SVP の求解アルゴリズムである GaussSieve アルゴリズムが提案された．著者等は 2012 年に並列 Gauss Sieve アルゴリズムを提案し，従来よりも効率的な並列化を実現している．本稿では並列 Gauss Sieve アルゴリズムに Schneider によって提案されたイデアル格子の性質を利用した高速化手法を適用し，イデアル格子チャレンジの最高記録である 128 次元のイデアル格子上の最短ベクトル問題を 84 台の計算機を用いて約 30,000CPU 時間で求解した．また Gauss Sieve アルゴリズムを高速化する新たな条件 Trinomial lattice を定義し，Gauss Sieve アルゴリズムの約 25 倍の高速化を達成した．

Solving the SVP in the Ideal Lattice of 128 dimensions using Parallel Gauss Sieve Algorithm

Tsukasa Ishiguro† Shinsaku Kiyomoto† Yutaka Miyake† Tsuyoshi Takagi

†KDDI R&D Laboratories Inc.
2-1-15, Ohara, Fujimino, Saitama, 356-8502, Japan
tsukasa, kiyomoto, miyake@kddilabs.jp

‡Institute of Mathematics for Industry, Kyushu University
744, Motooka, Nishi-ku, Fukuoka, 819-0395, Japan
takagi@imi.kyushu-u.ac.jp

Abstract Security of lattice-based cryptosystems is based on the hardness of the Shortest Vector Problem (SVP) in lattices. Micciancio et al. proposed a GaussSieve algorithm for solving the SVP in 2010. The authors proposed parallel Gauss Sieve Alorithm for of Gauss Sieve algrothm without excessive overhead even in a large-scale parallel computation. In this paper, we apply it for speed-up technique proposed by Schneider, which is used rotation structure of ideal lattices. Moreover, we found a new type of lattices suited for speed-up of Gauss Sieve algorithm. Finally, we have succeeded in solving the SVP over a 128-dimensional ideal lattice generated using about 30,000 CPU hours.

1 背景

量子コンピュータを用いても効率的に解読できない暗号系の一つとして格子暗号が注目されている．格子暗号は格子の最短ベクトル問題 (SVP) の困難性を安全性の根拠とする暗号方式である．1996 年に Ajtai によって SVP が NP 困難問題であることが証明され，格子暗号のアイデアが示された [1]．それ以降，格子を利用した完全準同型暗号 [9]，多重線形写像 [8] など様々な方式が提案されている．

格子暗号の安全性の根拠となる SVP に対して様々な求解アルゴリズムが提案されている [13, 24, 26, 7, 14]．その中の一つに篩アルゴリズムがある．篩アルゴリズムは，最短ベクトル問題を求解する確率的アルゴリズムである．2001 年に Ajtai 等によって AKS Sieve[2] が提案され，それ以降，より計算量を削減したアルゴリズムが提案されている [17, 5, 4]．2010 年には Micciancio 等によって Gauss Sieve アルゴリズムが提案されている [15]．一般に篩アルゴリズムの時間・空間計算量は $2^{O(n)}$ であり，Gauss Sieve アルゴリズムの時間計算量は $2^{0.52n}$，空間計算

量は $2^{0.21n}$ となることが実験により確かめられている．Milde らによって Gauss Sieve アルゴリズムの並列実装手法が提案されている [16] が，並列実装は小さい並列度では効果的であるものの，並列スレッド数が数十以上での実行では効率性が不十分であった．そこで著者らは 2012 年に並列 Gauss Sieve アルゴリズムを提案し，並列度を上げても効率性の落ちないアルゴリズムを提案している [11]．これによって 2 台の GPU，1 基の CPU を用いてシングルスレッドの Gauss Sieve アルゴリズムに比べて 60 倍の高速処理が可能となることを示した．しかし，複数台の計算機を用いた大規模な並列計算における並列 Gauss Sieve アルゴリズムの効果は明らかではなかった．

イデアル格子という特殊な格子を用いると暗号処理を高速化でき，鍵サイズを大幅に削減できるため，格子暗号の構成に広く用いられている [10, 8]．イデアル格子の最短ベクトル問題を求解するアルゴリズムとして Schneider らによって Ideal Gauss Sieve が提案されている [20]．このアルゴリズムではイデアル格子の性質を利用し，同じノルムのベクトルを少ない計算量で生成することで処理時間時間を高速化している．しかし，Gauss Sieve アルゴリズム高速化できる条件には制限があり，それ以外の条件でも高速化が可能かどうかは明らかではなかった．

本稿の成果　本稿では並列 Gauss Sieve アルゴリズムに Ideal Gauss Sieve で用いられている高速化手法を適用しイデアル格子の最短ベクトル問題を求解する並列 Ideal Gauss Sieve アルゴリズムを提案する．また，Schneider らの手法 [20] を拡張し，Gauss Sieve アルゴリズムを高速化する新しい条件 Trinomial lattice を定義する．Schneider らの手法 [20] では格子の次元 n が 2 冪または $n+1$ が素数の場合のみ Gauus Sieve を高速化できたが，Trinomial lattice の条件 $n = 2 \cdot 3^m, m > 0$ または $n = 2^s 3^t, s > 1, t > 0$ となる場合に Gauss Sieve を数倍高速化できることを示した．また，MPI を用いて複数台の計算機による並列処理を実装し，並列 Gauss Sieve アルゴリズムが大きな並列度でも有効であることを実証した．これらの結果，イデアル格子チャレンジ [18] で公開されており，従来解かれていなかった 128 次元のイデアル格子の最短ベクトル問題を 84 台の計算機で約 30,000CPU 時間を用いて求解したことを報告する．

2　定義

本章では格子に関する数学的定義を与える．

$B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \in \mathbb{R}^{n \times m}$ を n 個の一次独立なベクトルの集合とする．この時，基底ベクトルの整数係数線形結合全体

$$\mathcal{L}(B) = \mathcal{L}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) = \left\{ \sum_{1 \le i \le n} x_i \boldsymbol{b}_i, x_i \in \mathbb{Z} \right\}$$

を格子とし，B を格子基底と呼ぶ．$n = m$ の時，格子 $\mathcal{L}(B)$ を full-rank と呼ぶ．本稿では簡単のため，full-rank かつ $\boldsymbol{b}_i \in \mathbb{Z}^n$ とし，n を格子の次元とする．

ベクトル $\mathbf{v} = (v_1, \ldots, v_n)$ のユークリッドノルムを，$||v|| = (\sum_{1 \le i \le n} v_i^2)^{1/2}$ とする．格子 $\mathcal{L}(\mathbf{B})$ 上での最短ベクトルのノルムを $\lambda_1(\mathcal{L}(\mathbf{B}))$ と表す．ベクトル $\mathbf{v} = (v_1, \ldots, v_n), \boldsymbol{u} = (u_1, \ldots, u_n)$ の内積を，$\langle \mathbf{v}, \boldsymbol{u} \rangle = \sum_{1 \le i \le n} u_i v_i$ とする．

$g(x)$ を $\mathbb{Z}$ 上の n 次モニック多項式とする．I を環 $\mathbb{Z}[x]/(g(x))$ とし，I の元を $\mathbf{v}(x) = \sum_{0 \le i < n} v_i x^i \in \mathbb{Z}[x]/(g(x))$ と表す．この元 $\mathbf{v}(x)$ をベクトル $\mathbf{v} = (v_0, \ldots, v_{n-1}) \in \mathbb{Z}^n$ と同一視する．イデアル I は環 $\mathbb{Z}[x]/(g(x))$ の加法部分群となり，集合 $\{\mathbf{v} = (v_0, \ldots, v_{n-1}) \in \mathbb{Z}^n | \mathbf{v}(x) = \sum_{0 \le i < n} v_i x^i \in I\}$ は格子となる．この I で生成される格子をイデアル格子と呼ぶ．イデアル格子の性質から，ベクトル $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ の多項式表現を $\mathbf{v}(x)$ とすると，$\mathbf{rot}(\mathbf{v}) = x\mathbf{v}(x) \bmod \boldsymbol{g}(x)$ もイデアル格子 $\mathcal{L}(\mathbf{B})$ のベクトルとなる．この演算 **rot** をベクトルのローテーションと呼ぶ．i 回のローテーションの繰り返しを $\mathbf{rot}^i(\mathbf{v}) = \mathbf{rot}(\cdots \mathbf{rot}(\mathbf{rot}(\mathbf{v})) \cdots)$ と表し，$\mathbf{rot}^0(\mathbf{v}) = \mathbf{v}$ とする．

格子の最短ベクトル問題は以下のように定義する．

Definition 1 (最短ベクトル問題 (SVP)) 格子 $\mathcal{L}(\mathbf{B})$ が与えられた時，ユークリッドノルムが $\lambda_1(\mathcal{L}(\mathbf{B}))$ となるベクトル $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ を求める問題を最短ベクトル問題 *(SVP)* と呼ぶ．

Gaussian heuristic により，格子 $\mathcal{L}(\mathbf{B})$ 上の最短ベクトルのノルムは $\lambda_1(\mathcal{L}(\mathbf{B})) = (1/\sqrt{\pi})\Gamma(\frac{n}{2}+1)^{\frac{1}{n}} \cdot \det(\mathcal{L}(\mathbf{B}))^{\frac{1}{n}}$ と見積もることができる [20]．ここで，$\Gamma(x)$ はガンマ関数を表す．

Gauss Sieve アルゴリズムは，格子上のベクトルに関する 2 つの性質 (Gauss-reduced, Pairwise-reduced) が重要であるため，以下に定義を示す．

Definition 2 (Gauss-reduced) 格子上の *2* つの異なるベクトル $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{L}(\mathbf{B})$ が $||\boldsymbol{a} \pm \boldsymbol{b}|| \ge \max(||\boldsymbol{a}||, ||\boldsymbol{b}||)$ を満たすとき，$\mathbf{a}, \mathbf{b}$ は *Gauss-reduced* であるという．

Reduce アルゴリズムを繰り返し用いて 2 つのベクトルを Gauss-redeced なベクトルの組みへ変換することができる．Reduce アルゴリズムを 1 に示す．入力するベクトルに順番があること ($Reduce(\mathbf{a}, \mathbf{b}) \neq Reduce(\mathbf{b}, \mathbf{a})$) に注意されたい．もし入力ベクトル

Algorithm 1 Reduce [15]

Input: Vectors $\boldsymbol{p}_1, \boldsymbol{p}_2$ in lattice $\mathcal{L}(\mathbf{B})$
Output: Vector $\boldsymbol{p}_1$ in lattice $\mathcal{L}(\mathbf{B})$
1: **if** $|2 \cdot \langle \boldsymbol{p}_1, \boldsymbol{p}_2 \rangle| > ||p_2||^2$ **then**
2: $\quad \boldsymbol{p}_1 \leftarrow \boldsymbol{p}_1 - \left\lfloor \frac{\langle \boldsymbol{p}_1, \boldsymbol{p}_2 \rangle}{\langle \boldsymbol{p}_2, \boldsymbol{p}_2 \rangle} \right\rceil \cdot \boldsymbol{p}_2$
3: **return** $\boldsymbol{p}_1$

$\mathbf{a}, \mathbf{b}$ が線形従属の場合，必ず原点ベクトルが出力される．このことを衝突と呼ぶ．

また，ベクトルの集合 A 内の相異なるペア $\{\mathbf{a}_i, \mathbf{a}_j\} \in A$ が Gauss-reduced であるとき，集合 A を Pairwise-reduced という．Pairwise-reduced の定義を以下に示す．

Definition 3 (Pairwise-reduced) A を格子 $\mathcal{L}(B)$ 上のベクトルの集合とする．$m(\in \mathbb{N})$ 個の格子ベクトルの集合 $A = \{\boldsymbol{a_1}, \ldots, \boldsymbol{a_m}\} \subset \mathcal{L}(\mathbf{B})$ が全ての組み合わせ $(\boldsymbol{a}_i, \boldsymbol{a}_j)$, *(ただし，$i \neq j$)* について *Gauss-reduced* であるとき，集合 A は *Pairwise-reduced* であるという．

3 関連研究

本章では Gauss Sieve アルゴリズム [15] について説明し，その改良版である並列 Gauss Sieve アルゴリズム [11]，Ideal Gauss Sieve アルゴリズム [22] について説明する．

Gauss Sieve アルゴリズム [15] GaussSieve アルゴリズムは，Micciancio 等によって 2009 年に提案された SVP を解くアルゴリズムである [15](以下，GS アルゴリズムと呼ぶ)．GS アルゴリズムの処理時間の評価は文献 [20] を参照されたい．

GS アルゴリズムは，Pairwise-reduced となるベクトルの集合 L を生成し，新しいベクトル $\mathbf{v}$ を追加していく．この $\mathbf{v}$ を追加する際に，L の元 $\boldsymbol{\ell}_i \in L$ と $\mathbf{v}$ の全ての組み合わせ $(\boldsymbol{\ell}_i, \mathbf{v})$ が Gauss-reduced かどうかを判定し，Gauss-reduced である場合のみ L に $\mathbf{v}$ を追加する．この処理によって L が Pairwise-reduced であることが保証される．

GS アルゴリズムは，2 章で示した Reduce アルゴリズムが衝突を返す回数を終了条件とし，衝突回数が一定以上 $(\alpha|L| + \beta$ 以上) の場合にリスト L 内で最も短いベクトルが格子 $\mathcal{L}(\mathbf{B})$ の最短ベクトルとして出力される．

並列 Gauss Sieve アルゴリズム [11] 著者らは 2012 年に並列 Gauss Sieve アルゴリズム (以下，PGS アルゴリズム) を提案している．PGS アルゴリズムはリスト L を Pairwise-reduced に保つため L が肥大化せず，並列度を上げても効率性が低下しない．

アルゴリズムの概要を示す．まず，新たなベクトル $\mathbf{v}_i$ を r 個生成する (サンプルベクトル V と呼ぶ)．GS アルゴリズムと同様に，スタックがある場合にはスタックから選択し，スタックが空の場合には新たに生成する．この r 個のベクトル $\mathbf{v}_i$ と，リスト L の全てのベクトル $\boldsymbol{\ell}_j \in L$ の組 $(\mathbf{v}_i, \boldsymbol{\ell}_j)$ が Gauss-reduced となるように Reduce 関数を適用する．そのため，最初に $\mathbf{v}_i \leftarrow \mathit{Reduce}(\mathbf{v}_i, \boldsymbol{\ell}_j)$ を計算し，全ての $\mathbf{v}_i$ を更新する．次に $\mathbf{v}_i \leftarrow \mathit{Reduce}(\mathbf{v}_i, \mathbf{v}_j), i \neq j$ を計算する．最後に $\boldsymbol{\ell}_i \leftarrow \mathit{Reduce}(\boldsymbol{\ell}_j, \mathbf{v}_i)$ を計算する．各ステップで変更されたベクトル $\mathbf{v}_i, \boldsymbol{\ell}_j$ はスタック S に移動し，変更されなかったベクトルによってリスト L を更新する．

上記の処理を GaussSieve と同様に，衝突がある一定以上発生するまで繰り返す．このアルゴリズムでは，リスト L は必ず Pairwise-reduced となり，並列度を上げても肥大化することはない．あるベクトルを reduce によって更新する際に，他のベクトルの更新を並列実行できるため，最大で r 並列化が可能となる．しかし，実装上は一つのスレッドで複数のサンプルベクトルを処理することでオーバーヘッドを削減できる [11]．従って，スレッド数を t とすると一つのスレッド当たりのサンプルベクトル数は $\lfloor r/t \rfloor$ となる．

Ideal Gauss Sieve アルゴリズム [22] Schneider らはイデアル格子の性質を利用し Gauss Sieve アルゴリズムを高速化する手法を提案している [22]．このアルゴリズムを Ideal Gauss Sieve と呼ぶ．イデアル格子で次元 n の $\mathcal{L}(\mathbf{B})$ を生成する多項式を $g(x)$ とすると，$\mathbf{rot}(\mathbf{v}) = x\mathbf{v}(x) \bmod \boldsymbol{g}(x)$ もイデアル格子 $\mathcal{L}(\mathbf{B})$ のベクトルとなる．そのため $\boldsymbol{g}(x)$ の選び方によって剰余演算が簡単な処理になり，ベクトル $\mathbf{v}$ をローテションすることにより新しいベクトル $\mathbf{rot}(\mathbf{v})$ を高速に生成できる．

n が 2 冪のとき，円分多項式 $\boldsymbol{g}(x) = x^n + 1$ で生成される格子を *Anti-cyclic lattice* と呼ぶ．この時，ベクトル $\mathbf{v}$ のローテーションは $\mathbf{rot}(\mathbf{v}) = (-v_{n-1}, v_0, \ldots, v_{n-2})$ となる．この Anti-cyclic lattice の場合，ローテーション後のベクトルのノルム $|\mathbf{rot}(\mathbf{v})|$ と元のベクトルのノルム $|\mathbf{v}|$ は変化しない．そのため，Gauss Sieve アルゴリズム中のリスト L 内のベクトル $\mathbf{v}$ に対して $\mathbf{rot}^i(\mathbf{v}), i = 1, 2, ..., n-1$ を少ないコストで計算することによって Gauss Sieve アルゴリズム高速化することができる．Schneider らはこの Ideal Gauss Sieve アルゴリズムを用いて 64 次元のイデアル格子の SVP を Gauss Sieve を約 15 倍高速化できることを示している [22]．

4 提案方式

本章では並列 Gauss Sieve アルゴリズムにイデアル格子の性質を用いた高速化手法を適用したアルゴリズムを提案する．更に，Gauss Sieve アルゴリズムを高速化する新たな条件 Trinomial lattice について提案する．

4.1 Parallel Ideal Gauss Sieve

本章では PGS アルゴリズムにイデアル格子の性質を利用した高速化手法を適用した Parallel Ideal Gauss Sieve を提案する．このアルゴリズムを Alg.2 に示す．

3 章で示したように，$\boldsymbol{g}(x)$ の選び方によってイデアル格子 $\mathcal{L}(\mathbf{B})$ のベクトルを $\mathbf{v}$ に対して $\mathbf{rot}^i(\mathbf{v}), i = 1, \ldots, u$ を高速に求めることができる．このローテーション数 u は格子の次元 n 以下の整数とする．GS アルゴリズムで用いる Reduce 関数 (Alg.1) ではノルム $||\boldsymbol{p}_2||$ と内積 $\langle \boldsymbol{p}_1, \boldsymbol{p}_2 \rangle$ の大小関係によって第一引数のベクトル $\boldsymbol{p}_1$ が更新するか否かが決定する．従って，なるべく多くのベクトルで内積とノルムを比較し，$\boldsymbol{p}_1$ が更新される確率を上げることによって最短ベクトルの発見を早めることができる．以下にその手順を示す．

PGS アルゴリズムではサンプルベクトル $\mathbf{w}_i \in V$ とベクトル $\boldsymbol{\ell}_j \in L$ に対して $\mathbf{w}_i \leftarrow \mathit{Reduce}(\mathbf{w}_i, \ell_j)$ を実行する．PIGS アルゴリズムでは，さらに $\mathit{Reduce}(\mathbf{w}_i, \mathbf{rot}^k(\boldsymbol{\ell}_j))$ を実行する．ここで，$i = k, \ldots, u-1$ とする．PGS アルゴリズムでは $\mathbf{w}_i$ に対して $|L|$ 個の元を用いて Reduce 関数を実行していたが，PIGS アルゴリズムでは $\mathbf{w}_i \in L$ に対して $u|L|$ 個の元を用いて Reduce 関数を実行することになる．

次に，PGS アルゴリズムと同様に $\mathbf{w}_i \leftarrow \mathit{Reduce}(\mathbf{w}_i, \mathbf{rot}^i(\mathbf{w}_j))$ を実行する．この処理によって $\mathbf{w}_i \in V$ に対して $u|V|$ 個の元を用いて $\mathbf{w}_i$ に Reduce 関数を適用することができる．更に，ベクトル $\mathbf{w}_i \in L, \mathbf{v}_j \in V$ に対して $\mathbf{w}_i \leftarrow \mathit{Reduce}(\mathbf{w}_i, \mathbf{rot}^i(\mathbf{v}_j))$ を実行する．これによって $\mathbf{w}_i \in L$ に対して $u|V|$ 個の元を用いて Reduce 関数を実行することになる．

以上の処理により，PGS アルゴリズムに比べてると，一つのベクトルに対して u 倍の回数の Reduce 関数を実行してベクトルのノルムを小さくすることができる．この結果，Schneider らの IGS アルゴリズム [22] と同様に通常の GS アルゴリズムに比べて少ないループで最短ベクトルを求めることが可能となる．

ローテーション回数 u はイデアル格子の種類によって異なる．Anti-cyclic lattice の場合には，何度ローテーションを行なってもノルムが変わらないため $u = n$ とするのが適している．一般のイデアル格子でのローテーションの場合には，多項式剰余を行うと係数が膨張するため，ノルムも大きくなってしまう．一方，4.2 章で示す Trinomial lattice の場合には $u = 6$ を選択することによって高速化することが可能となる．提案アルゴリズムの詳細な動作について文献 [12] を参照されたい．

Algorithm 2 Proposed Parallel Ideal Gauss Sieve

Input: lattice basis $\mathbf{B}$, the number of sample vectors $r \in \mathbb{N}$,the number of rotation $u \in \mathbb{Z}$, $\alpha, \beta \in \mathbb{R}$
Output: a shortest vector $\mathbf{v}$ in $\mathcal{L}(\mathbf{B})$

$L \leftarrow \{\}, V \leftarrow \{\}, S \leftarrow \{\}, K \leftarrow 0$
while $K < \alpha|L| + \beta$ **do**
  if $|S| \neq 0$ **then**
    $t \leftarrow \min(r, |S|)$
    for $j = 1, \ldots, t$ **do**
      Pop from Stack S to $\mathbf{v}_j$
  if $|S| < r$ **then**
    for $j = |S| + 1, \ldots, r$ **do**
      Generate a new vector $\mathbf{v}_j$
  $V' \leftarrow \{\}, V'' \leftarrow \{\}, L' \leftarrow \{\}$
  $V \leftarrow \{\mathbf{v}_1, ..., \mathbf{v}_r\}, L = \{\boldsymbol{\ell}_1, ..., \boldsymbol{\ell}_m\}$
  for $i = 1, \ldots, r$ **do**
    $\mathbf{w}_i \leftarrow \mathbf{v}_i$
    for $j = 1, \ldots, m$ **do**
      for $k = 0, \ldots, u-1$ **do**
        $\mathbf{w}_i \leftarrow \mathrm{Reduce}(\mathbf{w}_i, \mathbf{rot}^k(\boldsymbol{\ell}_j))$
    if $||\mathbf{w}_i|| = 0$ **then**
      $K++$
    else if $\mathbf{w}_i \neq \mathbf{v}_i$ **then**
      $S \leftarrow S \cup \{\mathbf{w}_i\}$
    else
      $V' \leftarrow V' \cup \{\mathbf{w}_i\}$
  $V' = \{\mathbf{v}_1, ..., \mathbf{v}_{r'}\}$
  for $i = 1, \ldots, r'$ **do**
    $\mathbf{w}_i \leftarrow \mathbf{v}_i$
    for $j = 1, \ldots, r'$ **do**
      if $i \neq j$ **then**
        for $k = 0, \ldots, u-1$ **do**
          $\mathbf{w}_i \leftarrow \mathrm{Reduce}(\mathbf{w}_i, \mathbf{rot}^k(\mathbf{v}_j))$
    if $||\mathbf{w}_i|| = 0$ **then**
      $K++$
    else if $\mathbf{w}_i \neq \mathbf{v}_i$ **then**
      $S \leftarrow S \cup \{\mathbf{w}_i\}$
    else
      $V'' \leftarrow V'' \cup \{\mathbf{w}_i\}$
  $V'' = \{\mathbf{v}_1, ..., \mathbf{v}_{r''}\}$
  for $i = 1, \ldots, m$ **do**
    $\mathbf{w}_i \leftarrow \boldsymbol{\ell}_i$
    for $j = 1, \ldots, r''$ **do**
      for $k = 0, \ldots, u-1$ **do**
        $\mathbf{w}_i \leftarrow \mathrm{Reduce}(\mathbf{w}_i, \mathbf{rot}^k(\mathbf{v}_j))$
    if $||\mathbf{w}_i|| = 0$ **then**
      $K++$
    else if $\mathbf{w}_i \neq \boldsymbol{\ell}_i$ **then**
      $S \leftarrow S \cup \{\mathbf{w}_i\}$
    else
      $L' \leftarrow L' \cup \{\mathbf{w}_i\}$
  $L' = \{\boldsymbol{\ell}_1, ..., \boldsymbol{\ell}_{m'}\}$
  $L \leftarrow L' \cup V''$
return a shortest vector in $\mathcal{L}(\mathbf{B})$

4.2 Trinomial Lattice

本稿では既約 3 項式で定義されるイデアル格子を Trinomial lattice と呼ぶ．すなわち，$\boldsymbol{g}(x) = x^n + ax^k + 1, 1 < k < n, a \in \{\pm 1\}$ かつ既約となる $\boldsymbol{g}(x)$ によって生成される n 次元イデアル格子を Trinomial lattice と定義する．

円分多項式から $\boldsymbol{g}(x)$ を選択する場合，以下の 2 条件のいずれか満足する場合に Troinomial lattice となる．

条件 1　$n/2$ が 3 のべき乗，すなわち $n = 2 \cdot 3^m, m > 0$ となる場合，円分多項式は $\boldsymbol{g}(x) = x^n + x^{n/2} + 1$ となる．この時ベクトル $\mathbf{v}$ のローテーションは $\mathbf{rot}(\mathbf{v}) = (-v_{n-1}, v_0, \ldots, v_{\frac{n}{2}-2}, v_{\frac{n}{2}-1} - v_{n-1}, v_{\frac{n}{2}}, \ldots, v_{n-2})$ となる．

条件 2　n が 2 のべき乗と 3 のべき乗の積の場合，すなわち $n = 2^s 3^t, s > 1, t > 0$ となる場合，円分多項式は $\boldsymbol{g}(x) = x^n - x^{n/2} + 1$ となる．この時ベクトル $\mathbf{v}$ のローテーションは $\mathbf{rot}(\mathbf{v}) = (-v_{n-1}, v_0, \ldots, v_{\frac{n}{2}-2}, v_{\frac{n}{2}-1} + v_{n-1}, v_{\frac{n}{2}}, \ldots, v_{n-2})$ となる．

$n = 2 \cdot 3^m, m > 0$ または $n = 2 \cdot 3^m, m > 0$ となる場合に円分多項式が 3 項式になることは Gallot によって示されている [6]．Trinomial lattice の場合，ローテーションの処理自体は全体の回転処理と中間項の差分 (または和) を 1 回計算するだけなので Anti-cyclic lattice に比べてもほぼ同様の処理量となる．

本稿では Trinomial lattice を用いた以下の 2 つの高速化手法を提案する．

ベクトルの逆ローテーション　Anti-cylcic lattice の場合，ローテーションした後でもベクトルのノルムが変わらいため Gauss Sieve アルゴリズムを高速化できることを 3 章で示した．本稿ではローテーションと逆方向となる逆ローテーションによる高速化手法を提案する．格子 $\mathcal{L}(\mathbf{B})$ のベクトルを $\mathbf{v}$ とする．このとき，$\mathbf{v}$ の逆ローテーションを $\mathbf{rot}^{-1}(\mathbf{v}) = x^{-1}\mathbf{v} \bmod \boldsymbol{g}(x)$ と定義する．また，i 回の逆ローテーションの繰り返しを $\mathbf{rot}^{-i}(\mathbf{v}) = \mathbf{rot}^{-1}(\cdots \mathbf{rot}^{-1}(\mathbf{rot}^{-1}(\mathbf{v}))\cdots)$ と表す．

Trinomial lattice の場合，$\boldsymbol{g}(x)$ を法とした x の逆元 x^{-1} は，

$$x^{-1} = \begin{cases} -x^{n-1} + x^{\frac{n}{2}-1} & (\text{条件 1 のとき}) \\ -x^{n-1} - x^{\frac{n}{2}-1} & (\text{条件 2 のとき}) \end{cases}$$

となる．

条件 1 の Trinomial lattice のベクトル $\mathbf{v} = (v_0, \ldots, v_{n-1})$ に対して逆ローテーションは，$\mathbf{rot}^{-1}(\mathbf{v}) = (-v_{n-1}, v_0, \ldots, v_{n/2-2}, v_{n/2-1} + v_{n-1}, v_{n/2}, \ldots, v_{n-2})$ となり，元のローテーションとほぼ同様の計算量で計算できる．これは条件 2 の場合も同様である．この逆ローテーションを用いると，Alg.2 のステップ 16 の $\mathbf{w}_i \leftarrow \mathit{Reduce}(\mathbf{w}_i, \mathbf{rot}^k(\boldsymbol{\ell}_j))$ に加えて $\mathbf{w}_i \leftarrow \mathit{Reduce}(\mathbf{w}_i, \mathbf{rot}^{-k}(\boldsymbol{\ell}_j))$ も処理することができる．従って $\mathbf{w}_i$ に対して $2u|L|$ 回 Reduce 関数を適用することができるため，$\mathbf{w}_i$ が更新される確率が高まり，PIGS アルゴリズムを高速化することが可能となる．ステップ 29，41 も同様に第二引数を逆ローテーションすることで，ベクトル一つあたりの Reduce 回数を 2 倍にすることができる．

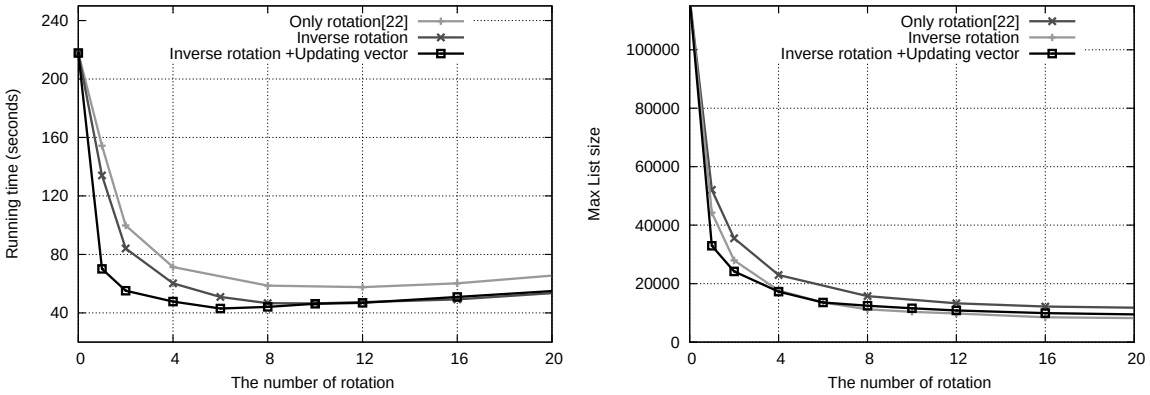
ベクトルアップデート　Trinomial lattice の場合，ローテーションしただけで，より短いベクトルが生成できることがある．そのため，Gauss Sieve アルゴリズム内のリスト L の元 $\boldsymbol{\ell}$ に対してローテーションを計算し，より短いベクトルが生成できた場合には元のベクトルを更新することによって，Gauss Sieve アルゴリズムを高速化することが可能となる．

ここでは条件 1 の Trinomial lattice の場合について説明する．Trinomial lattice のベクトルを $\mathbf{v} = (v_0, \ldots, v_{n-1})$ とする．このときローテーション，逆ローテーションを行ったベクトルと元のベクトルとのノルムの差分は以下のようになる．

$$\begin{aligned} ||\mathbf{rot}(\mathbf{v})|| - ||\mathbf{v}|| &= (v_{n-1})^2 + 2v_{\frac{n}{2}-1}v_{n-1}, \\ ||\mathbf{rot}^{-1}(\mathbf{v})|| - ||\mathbf{v}|| &= (v_0)^2 + 2v_{\frac{n}{2}}v_0. \end{aligned}$$

上式から，ベクトル内の 2 要素の符号・大小関係によって，より短いベクトルが生成できることがわかる．ローテーションしたベクトルのノルムの差分 $(v_{n-1})^2 + 2v_{\frac{n}{2}-1}v_{n-1}$ が負になる必要十分条件は，$v_{n-1}, v_{n/2-1}$ が異符号かつ $|v_{n-1}| < 2|v_{n/2-1}|$ となる．$v_{n-1}, v_{n/2-1}$ がそれぞれ独立かつ同じ確率分布から選ばれると仮定すると，$|v_{n-1}| < 2|v_{n/2-1}|$ となる確率は 1/2 以上である．そのため 1/4 以上の確率で $(v_{n-1})^2 + 2v_{\frac{n}{2}-1}v_{n-1}$ が負になり，ローテーション後のベクトルのノルム $||\mathbf{rot}(\mathbf{v})|| < ||\mathbf{v}||$ となる．この確率は $||\mathbf{rot}^{-1}(\mathbf{v})||$ についても同様である．

この性質を用いると，Gauss Sieve アルゴリズム内の集合 V, L 内のベクトルを小さいベクトルに変換することができる．Alg.2 のステップ 49 で，$L \leftarrow L' \cup V''$ でリスト L の更新を行った後，L 内の全てのベクトル $\boldsymbol{\ell} \in L$ に対して $\mathbf{rot}^k(\boldsymbol{\ell})$,$\mathbf{rot}^{-k}(\boldsymbol{\ell})$ を計算し，その中で最もノルムの小さいベクトルを見つける．もし小さいベクトルが見つかって更新されたベクトルは L からスタック S に移動する．これによって Pairwise-reduced の性質を変えずに Gauss Sieve アルゴリズムを高速化することができる．



(a) 処理時間　　(b) リスト L の最大サイズ

図 1: PIGS を用いた 72 次元の Trinomial latticec の SVP の処理時間とリスト L の最大サイズ

一方，ローテーションしたベクトルは 1/2 以上の確率でノルムが大きくなるため，ローテーションを繰り返すと元のベクトルのノルムより大きくなっていく．従って，ローテーション，逆ローテーション数 u は次元の数よりも低く設定する必要がある．そこで，5.3 章で実際の計算機実験によって最適なローテーションの回数を算出した．

5　実験結果

本章では提案アルゴリズムを実装し，いくつかの次元について SVP を求解した結果について説明する．

本実験では提案アルゴリズムを Amazon EC2 上の計算機 (cc1.8xlarge インスタンス) を用いた．この計算機は Intel Xeon E5-2670(2.6Ghz) という CPU が 2 基搭載されており，計 16 コアで並列計算できる．本実験では Voulgaris によって公開されている gsieve ライブラリ [28] を基に C++言語を用いて実装を行った．コンパイラは g++4.1.2 を用い，並列化のためのライブラリとして OpenMP，OpenMPI を用いた．

5.1　実装

本章では提案アルゴリズムを複数の計算機で並列実装手法，SIMD 命令を用いた高速化手法について説明する．

本実験では，Voulgaris によって公開されている gsieve ライブラリ [28] をベースに提案アルゴリズムの実装を行った．このライブラリには Gauss Sieve[15] が実装されているため，OpenMP を用いて 1 台の計算機内での並列処理を実装し，OpenMPI を用いてネットワーク上の並列計算を実装した．

更に予備実験によって以下の事実を確かめた．

- 出現するベクトルのすべての要素は 2^{16} 未満
- 提案アルゴリズムの処理時間はベクトルの内積計算 (Alg.1 のステップ 1) が支配的

これらの結果から Reduce 関数を SIMD 命令を用いて実装し高速化を行った．

実験に用いた CPU は Intel Xeon E5-2670 であるため，SSE4.2 をサポートしている．この命令セットを用いると，16 ビットの整数値を 128 ビットの SSE レジスタに 8 個詰め込み，8 個の整数に対して加減乗除などの整数演算を 1 命令で実行することが可能となる．特に内積計算 (Alg.1 のステップ 1)，ベクトルの定数倍によるベクトル減算処理 (Alg.1 のステップ 2) はベクトルの要素ごとに独立しているため 8 要素同時処理が可能となる．要素全てが 2^{16} 未満になるような 128 次元の格子上のベクトルをランダムに生成し Reduce 関数を 2^{28} 回実行した結果，SIMD 命令を用いない実装では 1,328msec，SIMD 命令を用いると 232msec となり約 5.7 倍高速化された．

5.2　パラメータの最適化

本章ではパラメータの最適化について述べる．

PGS アルゴリズムでは，スレッド数 t とサンプルベクトルの数 r を指定することができる．本実験で用いた計算機の物理コア数は 16 であり，ハイパースレッディングに対応しているためスレッド数 t は計算機の台数 $\times 32$ とした．この $t = 32$ で最も高速になる r は数千程度になることが実験によって確かめられている [11]．従って，本実験ではスレッド数は，計算機の台数 $\times 8,192$ とした．

表 1: Parallel Ideal Gauss Sieve を用いた 80,90,128 次元の処理時間

	次元	CPU 時間	インスタンス数	スレッド数 t	サンプルベクトル数 r	タイプ
SVPC[23]	80	0.9	1	32	8,192	*Random lattice*
	96	200	4	128	32,768	*Random lattice*
ILC[18]	80	0.9	1	32	8,192	*Ideal lattice*
	96	8	1	32	8,192	*Trinomial lattice*
	128	29,994	84	2,688	688,128	*Anti-cyclic lattice*

次に，Trinomial lattice の場合の最適なローテーション数について述べる．4 章で示したように，Trinomial lattice の場合にはローテーションすると高い確率でノルムが増大するため，次元の数よりも少ない回数にする必要がある．最適なローテーション数を見積もるために SVP を求解する処理時間を回転数ごとに測定した．イデアル格子チャレンジ [18] に掲載されている次元 72 のイデアル格子を入力として用い，事前計算として BKZ(ブロックサイズ 30) を適用している．この結果を図 1 に示した．

この結果からローテーション数を増やすとリストサイズの最大値が減少していくが，減少度合いはローテーション数 8 程度から急激に穏やかになり，ローテション処理の処理時間が増えるため，全体としては遅くなっていく．そのため，ローテション数は 6 程度が最適であり，ローテーションしない場合 (217 秒) に比べて約 5.1 倍高速になった．従って，本稿では Trinomial lattice の求解にはローテーション数 $u = 6$ とした．

5.3 SVP の求解実験

本稿で提案したアルゴリズムを用いていくつかの SVP の求解を行った結果について説明する．本実験では SVP チャレンジ [23], イデアル格子チャレンジ [18] で公開されている問題を入力として用いた．これらの問題の詳細な設定は文献 [19] に記述されている．本実験では事前計算として NTL ライブラリの [27] の BKZ(ブロックサイズ 30) を適用し，簡約された基底を提案アルゴリズムの入力とた．BKZ の処理時間は提案アルゴリズムに比べて十分小さいため，処理時間には含めていない．表 1 に求解を行った問題とアルゴリズムのパラメータを示す．

SVP チャレンジに対しては 80 次元，96 次元について求解を行った．それぞれの問題は “svpchallengedim80seed0.txt”,“svpchallengedim90seed0.txt” として SVP チャレンジのサイト [23] に公開されている．80 次元の求解は 1 台の計算機 (32 スレッド, 8,192 サンプルベクトル) を用いて約 1 時間で求解を完了した．文献 [11] では同じ CPU1 基を用いて，同問題の求解に約 12 時間程度かかっている．そのため，本稿で提案した高速化手法によって約 13 倍の高速化を達成し，CPU1 基あたり約 6.5 倍の高速化を達成したことになる．次元 90 の SVP についても求解を行い，4 台の計算機を用いて約 50 時間で求解を完了した．そのため約 200CPU 時間で求解したことになる．

イデアル格子チャレンジ [18] については 80 次元, 96 次元，128 次元について求解を行った．イデアル格子チャレンジでは，次元 n に対する円分多項式を選択することができるため，80 次元は通常のイデアル格子 ($g(x) = x^{80} + x^{78} - x^{70} - x^{68} + x^{60} - x^{56} - x^{50} + x^{46} + x^{40} + x^{34} - x^{30} - x^{24} + x^{20} - x^{12} - x^{10} + x^2 + 1$, “ideallatticedim80index220seed0.txt”)，96 次元は Trinomial lattice($\boldsymbol{g}(x) = x^{96} - x^{48} + 1$, “ideallatticedim 96index288seed0.txt”)，128 次元は Anti-cyclic lattice($\boldsymbol{g}(x) = x^{128} + 1$, “ideallatticedim128index256seed0.txt”) を選択した．80 次元のイデアル格子の SVP の求解は約 0.9 時間となり，通常の格子の SVP とほぼ同等の計算時間となった． 96 次元のイデアル格子では，4 章で提案した Trinomial lattice の高速化手法を適用し求解を行った．その結果，処理時間は約 8 時間となり通常の SVP の 96 次元に比べて約 25 倍の高速化を達成した．

128 次元のイデアル格子上の最短ベクトル問題の求解には 84 台の計算機を用いた．1 台あたり 32 スレッド, 8,192 サンプルベクトルとなり，合計 2,688 スレッド，688,122 サンプルベクトルとした．この実験の結果，約 16 日で求解を完了し，総 CPU 時間は 29, 994 時間となった．得られたベクトルのノルムは 2,959 となり，$n = 128$ の $(1.05/\sqrt{\pi})\Gamma(\frac{n}{2}+1)^{\frac{1}{n}} \cdot \det(\mathcal{L}(\mathbf{B}))^{\frac{1}{n}}$ よりも十分小さい値が得られた．

Gauss Sieve アルゴリズムの計算量見積りは次元 n に対して $2^{0.52n}$ となる [15]．提案アルゴリズムも同様の計算量と仮定すると，128 次元の SVP の処理時間の見積りは，96 次元の SVP の処理時間を用いて

$$200 \times 2^{0.52\Delta n} \approx 2.0 \times 10^7 [\text{時間}]$$

と見積もることができる．この見積りから，128 次元の Anti-cyclic lattice の SVP の求解は random lattice に比べて約 600 倍高速であると見積もることができる．

6 まとめ

本稿では Gauss Sieve アルゴリズムを改良しイデアル格子の最短ベクトル問題を求解する Parallel Ideal Gauss Sieve アルゴリズムを提案した．提案アルゴリズムを用いて，イデアル格子チャレンジに世界記録となる 128 次元のイデアル格子の最短ベクトル問題を 84 台の計算機で約 30,000CPU 時間を用いて求解した，Sieve 系アルゴリズムの漸近計算量は他の SVP 求解アルゴリズムよりも小さいため，今後解析できる次元が大きくなるにつれて提案アルゴリズムの優位性が更に高まっていく可能性がある．また，Gauss Sieve アルゴリズムを高速化する新しいイデアル格子 Trinomial lattice を発見した．これによって Gauss Sieve アルゴリズムを高速化できる次元を従来よりも拡張できることを示した．

参考文献

[1] M. Ajtai and C. Dwork. A Public-key Cryptosystem with Worst-case/average-case Equivalence. In *Proc of the 29th Annual ACM Symposium on Theory of Computing*, STOC'97, pages 284–293. ACM, 1997.

[2] M. Ajtai, R. Kumar, and D. Sivakumar. A Sieve Algorithm for the Shortest Lattice Vector Problem. In *Proc of the 33th Annual ACM Symposium on Theory of Computing*, STOC'01, pages 601–610. ACM, 2001.

[3] Amazon. Amazon Elastic Compute Cloud. Available at `http://aws.amazon.com/jp/ec2/`.

[4] V. Arvind and P. S. Joglekar. Some Sieving Algorithms for Lattice Problems. In *Proc of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'08, volume 2 of *LIPIcs*, pages 25–36. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2008.

[5] J. Blömer and S. Naewe. Sampling Methods for Shortest Vectors, Closest Vectors and Successive Minima. *Journal of Theoretical Computer Science*, volume 410, issue 18, pages 1648–1665, 2009.

[6] Y. Gallot. Cyclotomic Polynomials and Prime Numbers, 2000. Available at Gallot's homepage `http://yves.gallot.pagesperso-orange.fr/papers/cyclotomic.pdf`.

[7] N. Gama, P. Nguyen, and O. Regev. Lattice Enumeration Using Extreme Pruning. In *Proc of the 29th Annual International Conference on Theory and Application of Cryptographic Techniques*, Eurocrypt'10, volume 6110 of *LNCS*, pages 257–278. Springer, 2010.

[8] S. Garg, C. Gentry, and S. Halevi. Candidate Multilinear Maps from Ideal Lattices. Cryptology ePrint Archive, Report 2012/610, 2012.

[9] C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proc of the 41st Annual ACM Symposium on Theory of Computing*, STOC'09, pages 169–178. ACM, 2009.

[10] J. Hoffstein, J. Pipher, and J. Silverman. NTRU: A Ring-based Public Key Cryptosystem. In *Algorithmic Number Theory*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998.

[11] 石黒, 清本, 三宅, 高木. 格子の最短ベクトル問題に対する並列 GaussSieve アルゴリズム. 信学技報, vol.112, no.305, ISEC2012-71, pp.93–99, 2012.

[12] T. Ishiguro, S. Kiyomoto, Y. Miyake and T. Takagi. Parallel Gauss Sieve Algorithm: Solving the SVP in the Ideal Lattice of 128 dimensions. Cryptology ePrint Archive, Report 2013/388, 2013.

[13] A. Lenstra, H. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Journal of Mathematische Annalen*, volume 261, issue 4, pages 515–534, 1982.

[14] D. Micciancio and P. Voulgaris. A Deterministic Single Exponential Time Algorithm for Most Lattice Problems Based on Voronoi Cell Computations. In *Proc of the 42nd ACM Symposium on Theory of Computing*, STOC'10, pages 351–358. ACM, 2010.

[15] D. Micciancio and P. Voulgaris. Faster Exponential Time Algorithms for the Shortest Vector Problem. In *Proc of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'10, volume 65, pages 1468–1480. SIAM, 2010.

[16] B. Milde and M. Schneider. A Parallel Implementation of GaussSieve for the Shortest Vector Problem in Lattices. In *Proc of the 11th International Conference on Parallel Computing Technologies*, volume 6873 of *LNCS*, pages 452–458. Springer, 2011.

[17] P. Q. Nguyen and T. Vidick. Sieve Algorithms for the Shortest Vector Problem Are Practical. *Journal of Mathematical Cryptology*, volume 2, pages 181–207, 2008.

[18] T. Plantard and M. Schneider. Ideal Lattice Challenge. `http://www.latticechallenge.org/ideallattice-challenge/`.

[19] T. Plantard and M. Schneider. Creating a Challenge for Ideal Lattices. Cryptology ePrint Archive, Report 2013/039, 2013.

[20] M. Schneider. Analysis of Gauss-Sieve for Solving the Shortest Vector Problem in Lattices. In *Proc of the 5th International Workshop of Algorithms and Computation*, WALCOM'11, volume 6552 of *LNCS*, pages 89–97. Springer, 2011.

[21] M. Schneider. *Computing Shortest Lattice Vectors on Special Hardware*. PhD thesis, Technische Universität Darmstadt, 2011.

[22] M. Schneider. Sieving for Shortest Vectors in Ideal Lattices. Cryptology ePrint Archive, Report 2011/458, 2011.

[23] M. Schneider and N. Gama. SVP Challenge. `http://www.latticechallenge.org/svp-challenge/`.

[24] C.-P. Schnorr. A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. *Journal of Theoretical Computer Science*, volume 53, issue 2-3, pages 201–224, 1987.

[25] C.-P. Schnorr. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. *Journal of Mathematical programming*, pages 181–191. Springer, 1993.

[26] C.-P. Schnorr and H. H. Horner. Attacking the Chor-Rivest Cryptosystem by Improved Lattice Reduction. In *Proc of the 14th annual international conference on Theory and application of cryptographic techniques*, Eurocrypt'95, pages 1–12. Springer, 1995.

[27] V. Shoup. Number Theory Library (NTL) for C++. Available at Shoup's homepage `http://shoup.net/ntl`.

[28] P. Voulgaris. Gauss Sieve beta 0.1, 2010. Available at Voulgaris' homepage at the University of California, San Diego `http://cseweb.ucsd.edu/~pvoulgar/impl.html`.