

異種クラスタ環境における Block Lanczos アルゴリズムの並列化

小須田 昭太† 渋谷 進‡ 黒澤 馨‡

† 茨城大学大学院理工学研究科情報工学専攻
‡ 茨城大学工学部情報工学科

要 旨

公開鍵暗号の安全性の根拠は、素因数分解問題の困難性に基いている。高速な素因数分解アルゴリズムである数体ふるい法において、大規模な線形方程式を解く必要があり、Block Lanczos 法が有効である。暗号解読者は、様々な環境を用いる可能性があり、複数の計算機を用いる場合、異なる性能の計算機であることが十分考えられる。このため、異種クラスタ環境での暗号の性能評価は非常に重要である。本研究では、異種クラスタ環境において、MPI を用いた異種性を考慮した動的な行列の分割による Block Lanczos 法の並列化手法について提案する。処理時間の計測を行ったところ、異種性を考慮しない場合に比べて、十分高速に処理できることが確認できた。

A Parallel Implementation of Block Lanczos Algorithm in Heterogeneous Cluster Environment

Shota Kosuda† Susumu Shibusawa‡ Kaoru Kurosawa‡

† Graduate School of Science and Engineering, Ibaraki University
‡ Department of Computer and Information Sciences, Ibaraki University

Abstract

The integrity of public key cryptosystems are based on the difficulty of integer factorization. When using the number field sieve which is one of high-speed integer factorization algorithms, we need to solve large-scale linear equations. Block Lanczos method is efficient for this purpose. The adversary who tries to decode a cipher may use a variety of computer environments such as a heterogeneous cluster environment which consists of different performance computers. Therefore, the performance evaluation in such an environment is very important. In this research, we propose a parallel technique of Block Lanczos method using the dynamic matrix division in heterogeneous cluster environment. We implement the parallel technique and measure the processing time. The result shows that our method is superior to methods that do not consider heterogeneous cases.

1 はじめに

近年、素因数分解問題は、公開鍵暗号に用いられていることから、情報工学の分野においても注目されるようになってきた。素因数分解問題の困難性は、ある整数を素因数分解するのに費やされる処理時間を計測することで評価することができる。現在、最も高速な素因数分解アルゴリズムは一般数体ふるい法であると考えられている。一般数体ふるい法で

は、 $GF(2)$ を係数とする線形方程式を解く必要がある。この線形代数部分において扱う行列は、巨大な疎行列であるため、Montgomery の Block Lanczos 法 [1] が有効であることが知られている。このため、Block Lanczos 法の評価を行うことは、素因数分解問題の困難性の評価につながり、非常に重要であると言える。

本研究では、Block Lanczos 法の処理の高速化を行い、性能評価を行うことを目的とする。同様の研

究として [2, 3, 4, 5] において, Block Lanczos 法の並列実装を行っているが, 計算機環境として, 同じ性能の計算機 (CPU) を用いた環境を想定している。しかし, 暗号解読者は様々な計算機環境を用いる可能性があり, 複数の計算機を用いる場合, 同じ性能の計算機であるとは限らない。同じ性能の計算機における性能評価だけではなく, 異なる性能の計算機における性能評価を行うことも必要である。

そこで, 本研究では, 異種性を考慮した効率的な実装方法について提案する。本手法は, 異種性を考慮し, 各計算機の計算速度の評価を行い, 動的に大規模な行列やベクトルの分割数を決定する。さらに, 同種のクラスタ環境でも, より高速に処理できるように, 通信量の減少による, より効率的な実装方法を提案する。

性能評価として, 異種クラスタ環境において, MPI (Message Passing Interface) を用いて並列実装し, 処理時間の計測を行う。また, 同様の研究 [3] で用いられている方法と比較する。

本論文は, 以下のような構成である。2 節において, 逐次処理における Block Lanczos 法のアルゴリズムについて述べる。3 節では, 関連研究による並列手法を紹介する。4 節では, 異種性を考慮した並列実装について提案する。5 節では, 実験し, 性能評価を行う。最後に, 6 節として, 本研究の考察を行い, 今後の課題について述べる。

2 研究背景

2.1 記号, 用語の定義

本論文で用いる記号及び用語を定義する。

- N : ブロック長 (実質的には, コンピュータのワードサイズの倍数)。
- B : $GF(2)$ 上の非常に疎な $n_1 \times n_2$ 行列 ($1 \ll n_1 < n_2$ とする)。
- A : n_2 次対称行列 $B^T B$ 。 T は転置行列を表す。
- I_N : N 次単位行列。
- Large Matrix: B のような非常に疎で大規模な行列。
- Small Matrix: N 次正方行列。
- Vector: n_1 または n_2 次ベクトルを N 個並べた $n_1 \times N$ または $n_2 \times N$ 行列, あるいはその転置行列。
- nonzero rate: Large Matrix において列あたりに存在する “1” の割合。

Input: B, Y

Output: X, V_m

01. Initialize: $W_{-2}^{inv} = W_{-1}^{inv} = 0$,
 $V_{-2} = V_{-1} = 0$,
 $Cond_{-1} = 0, K_{-1} = 0$,
 $SS_{-1}^T = I_N, X = 0$
02. $V_0 = B^T * (B * Y)$
03. $BV_0 = B * V_0$
04. $(V^T B^T)_0 = (BV_0)^T$
05. $Cond_0 = (V^T B^T)_0 * BV_0$
06. $i = 0$
07. **while** $Cond_i \neq 0$ **do**
08. $AV_i = B^T * BV_i$
09. $V_i^T A^2 V_i = (AV_i)^T * AV_i$
10. W_i^{inv}, SS_i^T
11. $X = X + V_i * (W_i^{inv} * (V_i^T * V_0))$
12. $K_i = V_i^T A^2 V_i * SS_i^T + Cond_i$
13. $D_{i+1} = I_N - W_i^{inv} * K_i$
14. $E_{i+1} = -W_{i-1}^{inv} * (Cond_i * SS_i^T)$
15. $F_{i+1} = -W_{i-2}^{inv} * (I_N - Cond_{i-1} * W_{i-1}^{inv} * K_{i-1} * SS_{i-1}^T)$
16. $V_{i+1} = AV_i * SS_i^T + V_i * D_{i+1} + V_{i-1} * E_{i+1} + V_{i-2} * F_{i+1}$
17. $BV_{i+1} = B * V_{i+1}$
18. $(V^T B^T)_{i+1} = (BV_{i+1})^T$
19. $Cond_{i+1} = (V^T B^T)_{i+1} * BV_{i+1}$
20. $i = i + 1$
21. **end while**
22. $V_m = V_i$
23. **return** X and V_m .

図 1: Block Lanczos アルゴリズムの擬似コード

- weight: Large Matrix 内に存在する “1” の個数。

2.2 Block Lanczos 法

Block Lanczos 法は, n_2 次対称行列 A と $n_2 \times N$ 次の初期 Vector Y を与えたときに, $AX = Y$ を満たす $n_2 \times N$ 次の Vector X を求めるアルゴリズムである [1]。図 1 に, このアルゴリズム [2, 3] を示す。図 1 において, V_i は Vector であり, W_i^{inv} , SS_i^T , D_i , E_i , F_i , $Cond_i$, K_i は Small Matrix である。詳細については省略する。図 1 から分かるように, Block Lanczos 法は複数の行列積を行うことにより, 処理していく。

3 関連研究

この節では, 本研究の実装方法の比較対象であり, 同様の研究である田中等の並列実装方法 [3] を簡単

に紹介する。田中等の研究では、同種のクラスタ環境を用いている。

3.1 並列化の対象

図1において、step 1~6は、初期設定のstepであり、主要な部分は、step 8~20のループ内の部分である。その中でも計算量が多いのは、step 8, 9, 11, 16, 17, 19の

Large Matrix $\times$ Vector $\Rightarrow$ Vector
 Vector $\times$ Vector $\Rightarrow$ Small Matrix
 Vector $\times$ Small Matrix $\Rightarrow$ Vector

を計算するstepである。ループ内のその他の部分は、Small Matrix 同士の計算となるため、あまり計算量は多くないと考えられる。よって、並列化の対象をstep 8, 9, 11, 16, 17, 19としている。

3.2 並列化手法

並列化を図るアプローチは、大きな行列やベクトルを分割するというものである。想定している計算機環境は、1台の管理ノードと複数台の計算ノードからなるクラスタシステムである。管理ノードと計算ノード間の通信は、一対一通信と、多数のノードに対して同様の情報を同時に送信する一斉同期通信とを状況に応じて使い分けている。

4 提案手法

本研究での並列実装方法は、3節の田中等の並列実装方法を基にしている。本手法では、ループ内の並列化だけでなく、初期設定のstepにおいても計算量が多いと考えられるstepがあるため、step 2, 3, 5についても並列化を行う。また、田中等の研究と同様に、計算機環境は1台の管理ノードと複数台の計算ノードからなるクラスタシステムを想定している。

まず、いくつかの改良点について述べた後、異種性を考慮した並列実装方法について述べる。以降では計算ノード j が保持しているデータ及び計算結果を、右下に添え字 j をつけることで表現する。例えば、 $(B)_j$ は、ノード j が保持している分割されたLarge Matrix B を表す。

4.1 step 17の並列化

Large Matrix $\times$ Vector の演算であり、田中等の並列化手法では、計算ノードはLarge Matrix B の行番号で判断し、計算ノードのCPU数を p とす

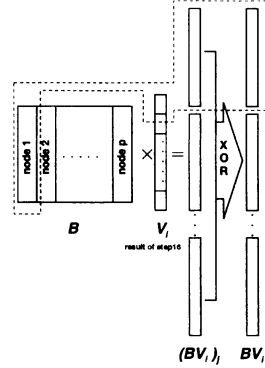


図 2: step 17 の並列化手法

ると、 p 分割した分割 Large Matrix $(B)_j$ を用いて演算を行っている。本研究では、行方向に対してではなく、列方向に対して分割を行った分割 Large Matrix $(B^T)_j$ を用いる。step 17 の時点では、この分割 Large Matrix $(B^T)_j$ は、既に送受信が済んでいるため、計算ノードは各自のメモリから読むこととなる。Vector V_i については、本手法では、step 16 の各計算ノードの演算結果である分割 Vector $(V_i)_j$ を用いる。分割 Vector $(V_i)_j$ についても、計算ノードは各自のメモリから読むこととなる。

また、各計算ノードからの結果を XOR することで step 全体の演算結果を計算する。この処理は、全ノードによるオールレデュースを使用して行う。これは、この step の演算結果である Vector BV_i を step 8 の演算で使用するの、Vector BV_i を各計算ノードが保持しておいた方が良いためと、オールレデュースによる XOR 操作の並列処理による効率化を期待しているためである。

この方法を用いることにより、分割 Large Matrix $(B)_j$ に関する通信やいくつかの step で必要とされる Vector に関する通信も省略できる。これらの通信処理の減少より、高速化が期待できる。さらに、分割 Large Matrix $(B)_j$ を扱う必要がなくなること等により、記憶領域の減少にもつながる。図2にstep 17の演算過程を示し、以下に、計算ノードと管理ノードにおけるstep 17の演算を示す。

- 計算ノード
 ノード j ($j = 1, 2, \dots, p$) :
 $(BV_i)_j = (B^T)_j \times (V_i)_j$,
 $BV_i = (BV_i)_1 \oplus (BV_i)_2 \oplus \dots \oplus (BV_i)_p$
- 管理ノード
 $BV_i = (BV_i)_1 \oplus (BV_i)_2 \oplus \dots \oplus (BV_i)_p$

4.2 step 11の効率化

田中等の並列化手法では、管理ノードが Vector X を保持している。step 11において、各計算ノードは演算結果を管理ノードに送信し、管理ノードは、各計算ノードからの結果を順に Vector の配列に格納し、全体の演算結果を取得している。管理ノードは、この演算結果に前のループでの Vector X を加えることによって、そのループでの新しい Vector X を得る。ここで、Vector X は、図 1 から分かるように、他の step での処理では必要とされない。

本手法では、ループ中で各計算ノードにおける分割数が変化しないことから、この step では各計算ノードは、演算結果の送信を行わず、各自のメモリに分割 Vector $(X)_j$ を保持する方法をとる。そして、ループから抜けたとき、つまり全 step の最後に、各計算ノードは、分割 Vector $(X)_j$ を管理ノードに送信し、管理ノードは、各計算ノードからの結果を順に Vector の配列に格納し、Vector X を取得する。実際に、興味のある大きさの巨大疎行列においては、ループ数は大きなものとなり、ループ中の数多くの通信処理が最後の一回のみに減少することにより、高速化が期待できる。

4.3 step の順序

各 step の計算を行うため、管理ノードが計算ノードに必要なデータを送信した後は、計算ノードは演算を行うが、管理ノードは行わない。よって、計算ノードが演算を行っている間、管理ノードは他の step の演算を行うようにする。ここでの他の step とは、管理ノードのみが計算を行わなければならない step を意味する。図 3 にまとめて、本手法における管理ノード及び計算ノードの計算処理、通信処理の流れを示す。図 3 において、縦の矢印は各ノード内の処理の流れを表し、管理ノードと計算ノードの間の横の矢印は、管理ノードと各計算ノードの間の通信を表している。また、通信を表している矢印の向きは、通信の方向を表す。

4.4 異種性を考慮した実装

今まで述べてきたように、並列化の方法は、Large Matrix 及び Vector を行方向あるいは列方向に対して分割し、計算ノードが分割 Large Matrix 及び分割 Vector を処理するという方法である。具体的に、Large Matrix 及び Vector のサイズに現れる n_1 及び n_2 を分割することによって処理をする。これらの分割の仕方は、これまで計算ノードの CPU 数 p で割って、 n_1/p または n_2/p となるように分割するというものであった。このような分割の仕方は、計

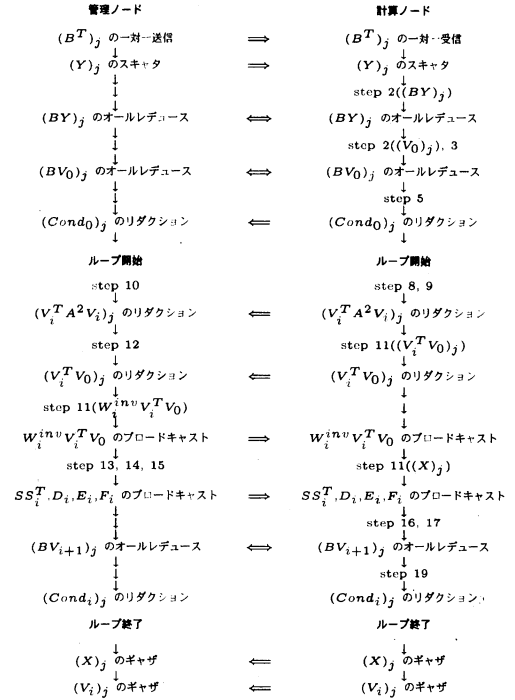


図 3: 本手法の処理の流れ

算ノードの計算速度が等しいときは適切であるが、異種クラスタ環境のように各計算ノードで計算速度が異なるような場合は、計算速度が速い計算ノードは、計算速度が遅いノードが処理を終了するのを待つことになり、効率が良くない。このため、本手法では、計算ノードに均等に分割するのではなく、異種性を考慮し、計算ノードの計算速度に応じて各計算ノードが担当する n_1 及び n_2 の分割のサイズを変更する方法をとる。以下に、分割のサイズを決定する方法を述べる。

まず、 n_2 については、step 2 の $B^T \times BY$ の Large Matrix $\times$ Vector の演算を使用して分割のサイズを決定する。各計算ノードでは、今までと同様に $(n_2/p) \times n_1$ のサイズの分割 Large Matrix $(B^T)_j$ と Vector BY を用いて計算する。この際、各計算ノードは、この計算処理に費やした時間を計測する。この時間を $(Time)_j$ とする。この $(Time)_j$ を使用して、各計算ノードの計算速度を推定するための値を求める。この値を、 $(AbilityValue)_j$ とし、式 (1) のように求める。ここで、 $(n_2)_j = n_2/p$ とする。

$$(AbilityValue)_j = \left\lfloor \frac{(n_2)_j}{(Time)_j} \right\rfloor \quad (1)$$

また、 $(Time)_j$ の単位は、動作環境によるが、本研究では、 $10^2 \mu\text{sec}$ としている。そして、各計算ノード

ドは, $(AbilityValue)_j$ についてオールギャザを使用して通信し, 全ノードの $(AbilityValue)_j$ を取得する. その後, 式 (2) を用いて, 新しい n_2 の分割サイズ $new(n_2)_j$ を求める.

$$new(n_2)_j = n_2 \times \frac{(AbilityValue)_j}{\sum_{k=1}^p (AbilityValue)_k} \quad (2)$$

以上のように, n_2 の分割サイズが変更するため, ギャザを用いて step 2 の各計算ノードの結果 $(V_0)_j$ を管理ノードに集め, 管理ノードは演算結果 V_0 を取得しておく. そして, n_2 の分割サイズが変更された後, 具体的には, step 3 においてスキヤタを用いて分配しなおす. また, n_2 の分割サイズが変更された直後に, 管理ノードは新しい分割 Large Matrix $(B^T)_j$ を再送信する.

次に, n_1 の分割サイズの変更について述べる. n_2 の場合とほとんど同様であるが, step 5 の Vector $\times$ Vector の演算を使用して分割のサイズを決定する. 各計算ノードでは, 今までと同様に $(n_1/p) \times N$ および $N \times (n_1/p)$ のサイズの分割 Vector を用いて計算する. この際, 各計算ノードは, この計算処理に費やした時間を計測する. n_1 の場合は, この時間を $(Time)_j$ とし, 式 (3) のようにして $(AbilityValue)_j$ を求める. ここで, $(n_1)_j = n_1/p$ とする.

$$(AbilityValue)_j = \left[\frac{(n_1)_j}{(Time)_j} \right] \quad (3)$$

また, $(Time)_j$ の単位は, n_2 の場合と同様である. そして, 各計算ノードは, $(AbilityValue)_j$ についてオールギャザを使用して通信し, 全ノードの $(AbilityValue)_j$ を取得する. その後, 式 (4) のようにして, 新しい n_1 の分割サイズ $new(n_1)_j$ を求める.

$$new(n_1)_j = n_1 \times \frac{(AbilityValue)_j}{\sum_{k=1}^p (AbilityValue)_k} \quad (4)$$

n_1 の場合は, n_1 の分割サイズの変更処理の他には, 新たに加わる処理はない.

このように n_1 及び n_2 の分割サイズを動的に変更することで, 均等に分割した場合に比べて, 各計算ノードの step の処理時間のバラツキが小さくなり, 処理全体の高速化につながることが期待できる. また, 本手法の分割サイズの変更は, 初期設定 step において行うため, 全ての処理時間に対して小さな時間で変更が行える. さらに, ループ中では分割サイズは変更しないため, 今までの節で述べてきたように, 各 step における処理データを再利用できるというメリットも損なわないで処理を進めることができる.

表 1: 計算機環境

Part	CPU	Memory	OS
管理ノード	Pentium2 400MHz	192MB	FreeBSD 4.8
計算ノード	Duron 750MHz	384MB	FreeBSD 4.10
計算ノード	Pentium4 3.20GHz	1GB	FreeBSD 5.3
計算ノード	Crusoe 1GHz	256MB	FreeBSD 4.10
計算ノード	Celeron 496MHz	64MB	FreeBSD 5.3

Network: Fast Ethernet, MPI: LAM-6.5.9

5 実験

本研究では, 性能評価を行うために, 異種クラスタ環境において, 計算機環境を変更せず, 巨大疎行列のサイズ等を変化させ, 田中等の並列化手法と本手法の処理時間の計測を行う. 本手法では, 異種性を考慮する場合と考慮しない場合の 2 種類の計測を行う. また, その他に同種のクラスタ環境において, 計算機 30 台程度を使用し, 実験を行っているが, ここでは省略する.

以降の実験で使用した実装方法において, ブロック長 N は 64 とし, Large Matrix は, 巨大疎行列であるため, 行列内の “1” の位置 (座標) を線形配列で保持する構成をとっている [3]. 行列の生成方法としては, 田中等の研究と同様に, 擬似乱数生成アルゴリズム Mersenne Twister [6] を利用した. Large Matrix B は, nonzero rate にあうように “1” の座標をランダムに定めるようにした. また, Vector Y も同様である.

5.1 実験環境

PC クラスタによる実験環境を表 1 に示す. MPI のライブラリとして LAM を使用した. また, Memory は各ノードが搭載しているメインメモリ容量を表している.

5.2 実験結果と考察

表 2 に実験の対象となる Large Matrix B を示す. まず, 田中等の並列化手法, 本手法の異種性を考慮した方法, 本手法の異種性を考慮しない場合に対して, それぞれループを 10 回実行した場合の 1 ループの平均処理時間を求めた.

また, 全処理時間については, 1 ループの平均処理時間から見積もりを行った. 見積もり方法は, ループによって処理時間が大きく変化しないこと, ループ以外の処理時間は, 全ループの処理時間に比べて非常に小さいことから, 1 ループの平均処理時間とループ回数の積により求めた. ループの回数については, Large Matrix B の rank を m としたとき, およそ m/N で見積もることができ, m は n_1 で押さ

表 2: 実験対象の Large Matrix

	matrix size	weight	nonzero rate
対象行列 1	300K × 330K	0.99M	1×10^{-5}
対象行列 2	400K × 440K	1.76M	1×10^{-5}
対象行列 3	500K × 550K	2.75M	1×10^{-5}
対象行列 4	600K × 660K	3.96M	1×10^{-5}
対象行列 5	700K × 770K	5.39M	1×10^{-5}
対象行列 6	800K × 880K	7.04M	1×10^{-5}
対象行列 7	900K × 990K	8.91M	1×10^{-5}
対象行列 8	1000K × 1100K	11.00M	1×10^{-5}

表 3: 全処理時間の見積もり (h)

	提案手法	提案手法 (異種性の考慮なし)	田中等の手法
対象行列 1	3.99	5.13	8.37
対象行列 2	7.10	9.34	14.58
対象行列 3	11.29	14.85	24.39
対象行列 4	16.38	21.57	39.05
対象行列 5	26.18	41.08	69.32
対象行列 6	39.64	72.53	119.56
対象行列 7	45.36	121.68	152.98
対象行列 8	58.65	199.54	211.39

えられるため、 n_1/N をループ回数とする [4]。表 3 に全処理時間の見積もりを示し、それをグラフ化したものを図 4 に示す。

実際に、表 2 の対象行列 1 については、全処理時間を計測した。この結果を表 4 に示す。表 4 から全処理時間の見積もりに、あまり大きな誤差はないと考えることができる。

以上の実験結果より、表 1 の計算機環境では、田中等の並列化手法に比べて、本手法（異種性の考慮なし）の方が高速に処理できることがわかる。また、異種性を考慮した実装方法では、さらに高速に処理できることがわかる。

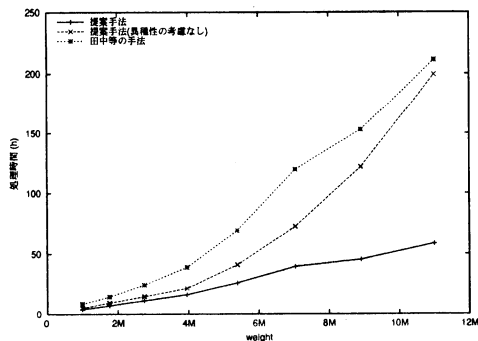


図 4: 全処理時間の見積もり

表 4: 全処理時間 (h)

	提案手法	提案手法 (異種性の考慮なし)	田中等の手法
対象行列 1	3.95	5.01	8.07

6 おわりに

本研究では、Block Lanczos 法の効率的な並列処理のためにいくつかの実装方法を提案してきた。それは、主に通信処理の減少による高速化を期待した、行列やベクトルの分割による方法である。これらの方法は、実験結果から今までの方法に比べて、確かに処理時間を短くすることができ、高速化できることが確認できた。

さらに、異なる計算速度の計算ノードを使用した場合の異種クラスタ環境でも高速に処理できるために、異種性を考慮した動的な行列やベクトルの分割方法も提案した。

そして、実験結果から少なくとも実験で用いたような、ノードの性能がバラバラで、台数が比較的少ないような場合では、異種性を考慮した提案分割手法は、より高速に処理できることが確認できた。これにより、異種性を考慮した動的な分割方法は、異種クラスタ環境にとって、重要であると考えられることができる。

今後の課題としては、より高性能な計算機を用いたクラスタでの性能評価、そして、より効果的な異種性を考慮した動的な分割方法の検討等である。

参考文献

- [1] Montgomery, P. L.: A Block Lanczos algorithm for finding dependencies over GF(2), *Proc. of Eurocrypt95*, LNCS 921, pp.106–120 (1995).
- [2] Flesch, I.: A New Parallel Approach to the Block Lanczos Algorithm for Finding Nullspaces over GF(2), Master's thesis, Utrecht University (2002).
- [3] 田中寛晃, 小池正修, 四方順司, 松本勉: Block Lanczos 法の並列処理による一実装, *Proc. of SCIS2004*, pp.281–286 (2004).
- [4] 下山武司, 植田広樹, 青木和麻呂: GF(2) 上巨大疎行列に対する並列計算法, *Proc. of SCIS2004*, pp.275–280 (2004).
- [5] Yang, L. T. and Brent, R. P.: The Parallel Improved Lanczos Method for Integer Factorization over Finite Fields for Public Key Cryptosystems, *Proc. of ICPPW'01*, pp.106–111 (2001).
- [6] 松本真: Mersenne Twister Home Page. <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html>.