Hybrid MPI/OpenMP による網羅率100% のレインボーテーブル生成の高速化

安藤 公希[1,a)]　桑原 寛明[2]　上原哲太郎[3]　國枝 義敏[3]

概要：パスワードクラックに用いられる辞書は，与えられたハッシュ関数により，任意の平文とそのハッシュ値のペアを予め生成し，網羅的にすべて記録する．これに対し，レインボーテーブルは，ハッシュ値から平文を生成する還元関数を導入し，複数ペアを接続したチェインを生成した上で，先頭の平文と末尾のハッシュ値のみを登録することでデータサイズを圧縮する．還元関数は，通常異なるハッシュ値から同一の平文が還元される可能性があり，圧縮率が悪化する．こうした衝突を排除するには，膨大な計算量を要するため，本稿では平文が英小文字 2〜5 文字それぞれの場合について，MPI と OpenMP の並列化により高速化した結果を示す．

キーワード：CSS2016，レインボーテーブル，Hybrid MPI/OpenMP，並列処理，パスワードクラック

Acceleration of Generating Full Coverage Rainbow Tables using Hybrid MPI/OpenMP

Koki Ando[1,a)]　Hiroaki Kuwabara[2]　Tetsutaro Uehara[3]　Yoshitoshi Kunieda[3]

Abstract: A dictionary for password cracking stores the pairs of the passwords and their corresponding hash values. On the other hand, a rainbow table is a compressed dictionary consisting of chains that are connected pairs of a password and its hash value. Only the first password and the last hash value of each chain are stored in a rainbow table since each chain can be restored. A reduction function generating some plain text from a hash value is needed for connecting pairs. However, different hash values are reduced to a same plain text. These collisions should be detected and eliminated since collisions have a bad effect on the compression ratio. In this paper, we propose a parallelized process of rainbow table generation with distributed collision detection by Hybrid MPI/OpenMP. We apply our proposed method to generate rainbow tables containing all passwords composed of 2, 3, 4 and 5 lowercase letters for showing the effect of speeding up.

Keywords: CSS2016，rainbow tables，Hybrid MPI/OpenMP，parallel processing，password cracking

1. はじめに

情報システムにおける個人認証には，一般にID(identification:識別子) とパスワードが用いられ，パスワードは ID を使用する人物が本人であることを証明する．以下，本稿ではパスワードを用いる認証技術をパスワード認証と呼ぶ．パスワード認証において，パスワードが他者に知られるとなりすましをされる危険性がある．そのため，一般的にパスワードはハッシュ関数を用いてハッシュ化され、ハッシュ値が保存される．ハッシュ化される前のパスワードを割り出すことをパスワードクラックといい，攻撃者はパスワードクラックによりハッシュ値から他人のパスワードを不正に取得する．

[1] 立命館大学大学院 情報理工学研究科
Graduate School of Information Science and Engineering, Ritsumeikan University
[2] 南山大学 情報センター
Center for Information and Communication Technology, Nanzan University
[3] 立命館大学 情報理工学部
Department of Computer Science, Ritsumeikan University
[a)] is0098ef@ed.ritsumei.ac.jp

パスワードクラックには，総当たり攻撃や辞書攻撃がある．総当たり攻撃は，パスワードとして使用されるであろうあらゆる単語の組合せを都度生成しながらパスワードを割り出す手法である．総当たり攻撃は，パスワードの文字長や文字種が増加すれば，パスワードを割り出すための候補が増加するため，膨大な計算時間が必要である．辞書攻撃はパスワードとして使用されるであろう単語を事前に登録して作成されたデータテーブルである辞書を用いてパスワードを割り出す手法である．辞書攻撃は，効率的なクラックが可能であるが，辞書に登録されていないパスワードは割り出すことができない．辞書をあらゆるパスワードに対応させるためには，あらゆるパスワードを登録すればよいが，パスワードの候補となりうる単語の文字長と文字種が増加すれば事前に登録しなければならないパスワードも増加するため，膨大な記憶容量が必要となる．このような理由のため，総当たり攻撃，辞書攻撃ともに計算量，記憶容量の観点からパスワード認証の現実的な脅威とはなりにくい．

一方で計算量，記憶量の観点から脅威となりうるパスワードクラックにレインボークラックがある．レインボークラックは，レインボーテーブルを用いてパスワードをクラックする手法である．レインボーテーブルは，辞書攻撃に使用される辞書を圧縮したデータテーブルであり，パスワードとハッシュ値で構成される．レインボーテーブルにより，本来の辞書の格納に必要な記憶容量を削減でき，かつ効率的なクラックが可能である．そのため，レインボークラックは計算量的にも記憶量的にも現実的なパスワードクラックを可能にする．

本研究では，レインボークラックがパスワード認証の脅威となりうること，および危険性がどの程度現実的となっているかを示すため，網羅率 100 %かつ重複のないレインボーテーブル生成の高速化を目標とする．従来手法 [1] では，MPI によるレインボーテーブル生成の高速化が実現されていたが，MPI 通信処理のオーバヘッドの問題があった．本研究では，この問題を解決すべく並列化の手法を見直し，通信処理のオーバヘッドを抑制する．具体的には，MPI および OpenMP による Hybrid MPI/OpenMP 並列処理によりレインボーテーブル生成を高速化する．レインボーテーブルは，パスワードである平文（以降，平文と表記）とそのハッシュ値のペアを接続することで生成されるが，ペアとペアの接続のためにハッシュ値から適当な平文を還元する必要がある．しかし，異なるハッシュ値から同一の平文が還元される衝突が高い確率で発生し，還元のたびに衝突判定を繰り返す必要がある．逐次処理では膨大な生成時間が必要となるため，並列処理によって負荷を分散し，レインボーテーブル生成を高速化する．

レインボーテーブルの生成例として，ネット掲示板などで利用されているパスワードを対象とするレインボーテーブルを生成する．本研究で対象とするパスワードは，英小文字 2〜5 文字のそれぞれのパスワードであり，それらに対応する網羅率 100% かつ重複のないレインボーテーブルを生成する．

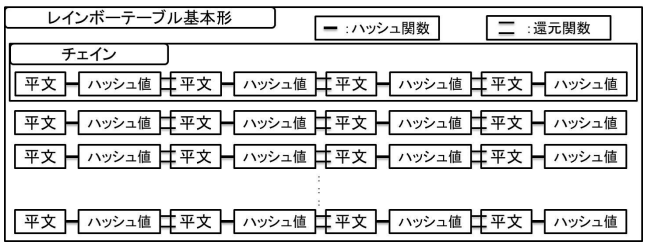

図 1　レインボーテーブルの構造
Fig. 1　The data structure of a rainbow table

本稿の構成は以下の通りである．2 章でレインボーテーブルとレインボーテーブルを使用したパスワードクラックについて説明し，3 章で MPI および OpenMP による Hybrid MPI/OpenMP 並列環境について記す．4 章でレインボーテーブル生成手法および，高速化手法について述べ，5 で提案手法の実装について記し，6 章で提案手法の評価，および考察を述べる．7 章で関連研究について述べ，8 章で本稿をまとめる．

2. レインボーテーブルとパスワードクラック

2.1 レインボーテーブル

レインボーテーブルは辞書攻撃に使用する辞書を圧縮した特殊なデータテーブルであり，平文とハッシュ値によって構成されるチェインの集合である．チェインは，平文とハッシュ値のペアを複数接続したものであり，ペアを接続することをチェイン化という．チェイン化には，ハッシュ関数，および，還元関数を使用する．

図 1 にレインボーテーブルの構造を示す．複数のチェインの集合体が，一つのレインボーテーブル基本形（以降，基本形と表記）を構成する．このとき，レインボーテーブルに含まれる平文はすべて異なっていることが望ましい．

基本形には，従来の辞書と同様にすべての平文とハッシュ値が含まれており，格納する平文の数が大きくなると必要な記憶容量が膨大になるため，圧縮して容量を小さくする．基本形の圧縮には，圧縮されたチェインであるレインボーセットを用いる．実際にパスワードクラックで使用するレインボーテーブルは，基本形を構成する一つ一つのチェインを圧縮したレインボーセットの集合である．以降，圧縮した基本形を単にレインボーテーブルと呼ぶ．

2.1.1 レインボーセット

レインボーテーブルを構成するチェインを圧縮する仕組みがレインボーセット（RainbowSet；以降，RS と表記）である．一つの RS は，一つのチェインの先頭の平文（FirstWord；以降，FW と表記）と末尾のハッシュ値

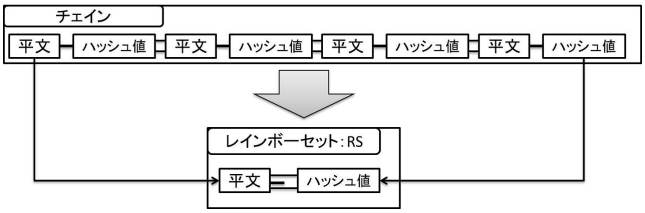

図 2　チェインとレインボーセット
Fig. 2　Rainbow Set(RS): compressed form of a chain

（LastHash；以降，LH と表記）の二つの文字列で構成される．チェインと RS の関係を図 2 に示す．一つのチェインから FW と LH を取りだし，一つの RS を構成する．本稿では，LH のもととなる平文を最後の平文（LastWord；以降，LW と表記）と呼ぶ．本稿の提案手法では処理の効率化のために一時的に FW と LW のペアも生成するが，FW と LW のペアも単に RS と呼ぶ．

チェイン生成に使用したハッシュ関数と還元関数を用いることにより，RS から元のチェインを復元できる．RS とチェインの違いは，FW と LH の間にある平文とハッシュ値を，必要となるたびに計算するか，一度生成した値を保持するかである．

2.1.2　ハッシュ関数

平文からハッシュ値を生成する関数を一方向性暗号学的ハッシュ関数（以降，ハッシュ関数と表記）と呼ぶ．ハッシュ関数は任意長の平文の入力に対し固定長ビット列のハッシュ値を出力する関数である．ハッシュ関数により平文がハッシュ化されハッシュ値となる．

2.1.3　還元関数

還元関数は，ハッシュ値から適当な平文を還元する関数である．ハッシュ値に対応する元の平文である必要はなく，異なるハッシュ値から同一の平文が還元される可能性がある．

2.1.4　レインボーテーブルの網羅率

レインボーテーブルの網羅率とは，生成したレインボーテーブルにおいて，対象とする平文をどれだけ網羅しているかの割合である．レインボーテーブルの網羅率は高い方が良く，より多くの平文のパスワードクラックが可能となる．本研究では，英小文字 2〜5 文字それぞれの場合における網羅率 100% レインボーテーブルを生成しており，対象とする平文すべてをクラック可能である．

2.1.5　レインボーテーブルの圧縮率

レインボーテーブルの圧縮率とは，辞書攻撃に使用される辞書に格納されている要素の個数とレインボーテーブルに格納されている要素の個数との比率である．圧縮率は高い方がもとの辞書を圧縮できたことになり，レインボーテーブルの容量も小さくなる．しかし，レインボーテーブルの圧縮率が非常に高い場合，クラック時におけるチェイン復元による計算処理の負荷が大きくなり，結果，パスワードクラックに要する時間が総当たり攻撃に要する時間に近くなっていくため，圧縮率を調整し，辞書攻撃に要する時間に近くする必要がある．

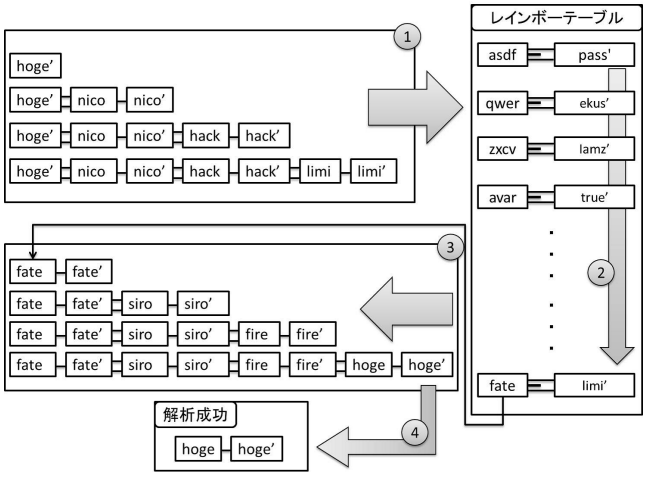

図 3　レインボークラック
Fig. 3　The process of a rainbow crack using a rainbow table

2.2　レインボーテーブルを使用したパスワードクラック

2.2.1　想定する攻撃状況

個人がパスワード認証を利用するとき，本人であることを証明するためにパスワードを入力する．入力されたパスワードはハッシュ関数によりハッシュ値に変換され，すでに登録されているハッシュ値と比較して個人認証をする．

本研究では，攻撃者がすでに ID とクラック対象のハッシュ値を取得している状況を想定する．攻撃者の目的は，取得したハッシュ値からもとのパスワードを導き出すこと，そして，ID とパスワードを不正に使用して個人になりすますことである．

2.2.2　レインボークラック

レインボーテーブルを用いて目的のハッシュ値に対応する平文を求めることをレインボークラックと呼ぶ．レインボークラックには，対象のハッシュ値，事前生成済みのレインボーテーブル，レインボーテーブル生成時に使用したハッシュ関数，および，還元関数を必要とする．

レインボークラックの例を図 3 に示す．ここではクラック対象のハッシュ値を hoge' とし，取得したいパスワードを hoge とする．① では，ハッシュ値に還元関数とハッシュ関数を順に適用して新たなハッシュ値を生成し，② でハッシュ値と LH（pass'，ekus'，lamz'，true'，…，limi'）を比較する．ハッシュ値と LH が一致しなかった場合は① の処理に移る．一致した場合（① の limi' と② の limi' が一致）は③ の処理に戻り，さらに一段チェインを延ばす．③ では LH（② における limi'）に対応する FW（②，③ における fate）からチェインを復元して対象のハッシュ値を探索する．対象のハッシュ値に対応する平文が目的のパスワードであり，④ でレインボークラックの成功となる．

3. Hybrid MPI/OpenMP

Hybrid MPI/OpenMP とは，プロセス間の処理を分散メモリ型の MPI で並列処理し，プロセス内の処理を共有メモリ型の OpenMP で並列処理することをいう．

3.1 MPI（Message Passing Interface）

MPI は，並列コンピューティングのための規格であり，C，C++，Fortran などの言語で使用できるライブラリとして実装されている [2] ．MPI は，SPMD(Single Program, Multiple Data streams) モデルとして並列処理を行う．このモデルでは，各プロセスは同じプログラムを実行するが，各プロセスはプロセスごとで異なるランク (rank) に基づく分岐によって異なるステートメントを実行する．

3.2 OpenMP（Open Multi-Processing)

OpenMP は，共有メモリ型並列コンピューティングのための規格である [3] ．OpenMP は，C/C++，Fortran などで利用でき，プラグマ・ディレクティブと呼ばれる命令文を用いて並列処理の記述ができ，スレッド並列処理を実現する．

4. レインボーテーブル生成の高速化

4.1 レインボーテーブルの生成手法

4.1.1 チェインの生成

チェイン生成時に，チェイン化済みの平文が還元されたとき，衝突が発生したと判断する．衝突した平文から生成される平文とハッシュ値のチェインは常に同一であるので，2 回以上生成することは無駄である．そのため，衝突した場合はチェインの生成を打ち切り，チェイン化されていない平文から新たなチェイン生成を開始する．本研究では，網羅率 100%の平文の重複のないレインボーテーブルを生成する．生成が完了したレインボーテーブルは，平文の衝突が一つも含まれておらず，無駄のないテーブルとなる．以下のようにして網羅率 100% のレインボーテーブルを構成する各チェインを生成する．

(1) チェイン化していない平文からチェイン生成を開始する．
(2) チェイン化済みの平文と同一の平文が還元された場合，チェイン生成を終了し（1）へ
(3) すべての平文のチェイン化が完了したとき，レインボーテーブルの生成を終了する．

パスワードの文字長や文字種といった条件を満たす平文とハッシュ値のペアがすべてチェイン化されるまでチェイン生成を続ける．最終的に生成されたすべてのチェインをまとめて一つのレインボーテーブルとする．

生成されたチェインについて，残りのチェインを生成するさいの衝突判定のためにチェインの FW と LW で構成される RS をメモリに格納し，レインボークラックに使用するレインボーテーブルとして FW と LH で構成される RS をファイルに書き出す．RS を記録した後，RS 生成に利用したチェインを破棄し，次の RS となるチェインを生成する．

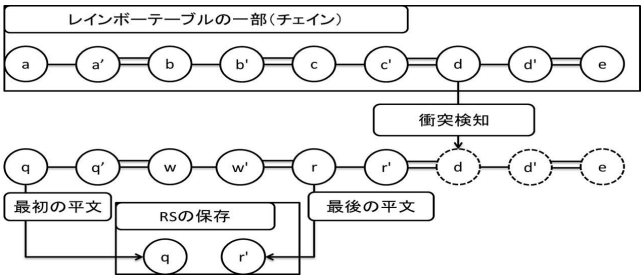

図 4　重複判定の例

Fig. 4　The process on collision between a reduced password and LW

4.1.2 LW 比較による衝突判定

還元された平文の衝突判定を効率的に行うために，LW 比較による衝突判定（以降，LW 判定と表記）を行う．具体的には，還元された平文と生成済みの各 RS に含まれる LW を比較することで衝突判定する．衝突した場合，生成中のチェインの FW から衝突した平文までの間で衝突している可能性があるため，衝突した LW に対応する FW からチェインを復元しながら二度目の衝突判定（以降，重複判定と表記）を行う．生成済みのチェインには衝突が発生していないため，還元された平文から衝突が発生していないかを探索するだけでよい．LW 判定により，生成済みのチェインすべてを探索しなくても衝突判定を実現できる．

図 4 に重複判定の例を示す．最後に還元された平文（図 4 の下の e）と RS の LW（図 4 の上の e）が衝突しているため，LW に対応した FW（図 4 の a）からチェインを復元しながらチェイン化中のチェインの FW（図 4 の q）から重複判定を行う．e より手前で還元された平文（図 4 の d）で衝突が起きているため，その直前の平文（図 4 の r）でチェイン化を終了する．生成されたチェインの FW（図 4 の q）と LW（図 4 の r）を RS として記憶する．

LW 判定の特徴として，チェインの圧縮と復元による時間と空間のトレードオフが挙げられる．これは衝突判定のためにチェインではなく RS を保存しているからである．生成するレインボーテーブルに格納すべき平文の数が膨大であるとき，チェインの保存は，RS の保存に比べ多くのメモリを使用する．RS での保存は，メモリ使用量を抑えられるが，チェインを復元するための計算が必要である．

4.2 Hybrid MPI/OpenMP による高速化

本研究では，レインボーテーブル生成を MPI および OpenMP を用いて並列処理することで高速化する．MPI ではプロセス間の並列処理を，OpenMP ではプロセス内の

並列処理をおこなう．

4.2.1 MPI による並列処理

MPI を用いて，同一のレインボーテーブルを各プロセスで分担して生成する．MPI による並列処理では，生成済みの RS を各プロセスで分散して保持することにより，還元された平文と生成済みの RS との LW 判定にかかる負荷をプロセスごとで分散し高速化する．具体的には，以下のようにして生成する．

(1) チェインの生成
　各プロセスで同一の平文から同一のチェイン生成を開始する．

(2) LW 判定
　還元された平文を各プロセスが保持する生成済みの RS と LW 判定する．

(3) 衝突結果の共有
　各プロセスでの LW 判定による平文の衝突結果をすべてのプロセスで共有する．いずれのプロセスでも衝突していない場合は，チェイン化を続け一段チェインを延ばし（2）へ，そうでない場合は（4）へ．

(4) RS の格納
　FW と衝突する直前の平文で構成される RS をいずれかのプロセスに保存する．このとき，各プロセスが持つ RS の数が均一になるよう保存するプロセスを決定する．チェイン化していない平文がある場合は（1）へ，そうでない場合は（5）へ．

(5) レインボーテーブル生成の完了
　各プロセスで分担して保持している RS を一つにまとめ，レインボーテーブルの生成を完了する．

（4）で生成済みの RS を各プロセスで分散して保持する．これにより，（2）において還元された平文と各プロセスに割り当てられた生成済みのレインボーテーブルとの LW 判定を並列処理できる．

従来手法の課題であった通信処理の回数を削減するため，提案手法では，通信を（3）における衝突結果の共有のみで行っている．

提案手法では，（1）において同時に複数本のチェイン生成が可能である．チェインを同時に複数本生成することを本稿では複数同時生成と呼ぶ．複数同時生成により，衝突結果をまとめて通信することで同期通信の回数を削減し高速化する．

本研究では，提案手法を LastHashBoundedCollectiveCommunication（以降，LHBC2 と表記）と呼ぶ．

4.2.2 MPI 通信における通信回数の削減

LW 判定の結果をプロセス間で共有するさい，生成中の各チェインについての衝突結果を個別に通信するのではなく一つにまとめて通信する．図 5 に個別に通信する場合とまとめて通信する場合の例を示す．個別に通信する場合，衝突結果を一つ一つ通信して同期処理をするため，通信回数は合計で 12 回となる．一方，まとめて通信する場合，各チェインに対する衝突結果をまとめて通信することができるため，通信回数は 3 回に抑えられる．

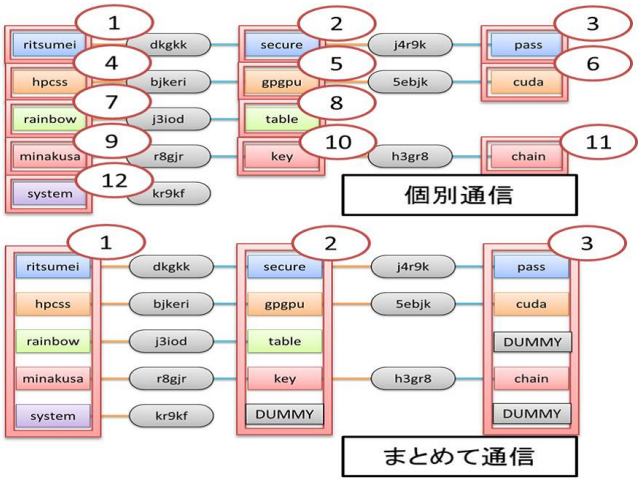

図 5　個別通信とまとめて通信
Fig. 5　Putting together in one communication process

4.2.3 OpenMP による並列化

MPI による各プロセスでのレインボーテーブル生成において，複数のチェインを同時に生成するためチェイン生成を OpenMP により並列化する．OepnMP による並列処理は，以下に示すように二つに分けられる．

(1) チェインの生成の並列化

(2) LW 判定の並列化

（1）では，複数のチェインにおける平文からハッシュ値を生成する処理，および，ハッシュ値から他の平文を還元する処理を並列化する．

（2）では，複数のチェインにおいて還元された平文の衝突判定を並列化する．複数の還元された平文について LW 判定を並列処理する．本稿では，この OpenMP の処理を+omp と表記する．

4.3 グルーピングによる LW 判定の高速化

LW 判定は，生成済みのチェインの数が増加するにつれて処理が遅くなる．そこで，生成済みの RS をグルーピングし，LW 判定の対象となる RS の数を削減することで高速化する．具体的には，RS に含まれる LW の先頭から n 文字をインデックスとし，同じインデックスの LW を含む RS を同一のグループにまとめる．

本稿では，グルーピングされた生成済みの RS を LastTable と呼び，この処理を+LT と表記する．LastTable により，還元された平文との LW 判定に使用する LW を絞り込むことができる．LastTable を導入した LW 判定では，グルーピングによって衝突しないことが明らかな衝突判定を削減することで LW 判定の高速化が可能である．

4.4 チェイン分割

生成したチェインを適当な長さに分割して各プロセスで保持することにより，各プロセスでの衝突判定処理における負荷を分散させる．チェインの長さによって LW 判定に要する処理量が変わるため，各プロセスが保持するチェインの長さはできるだけ均等な方がよい．

5. 提案手法の実装

5.1 チェイン生成処理

5.1.1 ハッシュ関数

今回の実装ではハッシュ関数に Unix の crypt(3) を使用する．Unix の crypt(3) のアルゴリズムは，DES（Data Encryption Standard）を使用している．

5.1.2 還元関数

今回の実装では，以下の処理を行う還元関数を独自に実装した．

(1) 引数に還元対象のハッシュ値および，対象とする平文の文字長を取る．

(2) ハッシュ値の先頭一文字目から還元する平文の文字長分一文字ずつ取得する．

(3) 取得した一文字 [0-9./]，[A-Z]，[a-z] をそれぞれ数値 [0-11]，[12-37]，[38-63] に変換後，0x3f との XOR をとり X とする．取得した文字の先頭からの位置（先頭は 0 とする）を pos で表し，還元する文字列の変数を RW で表す．

(4) X の値により次の三つの処理のいずれかを実行する．

(a) 0〜11：
pos が偶数のとき，$X = (2 \times X) + 65$ を実行し，奇数のとき，処理 $X = (2 \times X + 1) + 65$ を実行する．

(b) 12〜37：
処理 $X = X + 95$ を実行する．

(c) 38〜63：
処理 $X = X + 59$ を実行する．

(5) RW に変換が完了した X を順次代入する．すべての代入が完了したとき，RW を還元された平文として返す．

5.2 衝突判定処理

5.2.1 LW 判定

LW 判定では，還元された平文と RS に含まれる LW との衝突判定の結果（以降，更新された LW）を保持する．還元された平文が衝突しているとき，重複判定を行い，衝突する直前の未衝突の平文を更新された LW とする．衝突していないときは，制御用のダミーの平文を更新された LW とする（図 5 参照）．本研究では，対象とする平文が [a-z] で構成される文字列のため，[a-z] 以外の文字をダミーの平文に使用する．今回の実装では，ダミーの平文として [#] で構成された文字列を使用する．例として英小文字 4 文字を対象とするレインボーテーブルを生成する場合，ダミーの平文は“####”である．

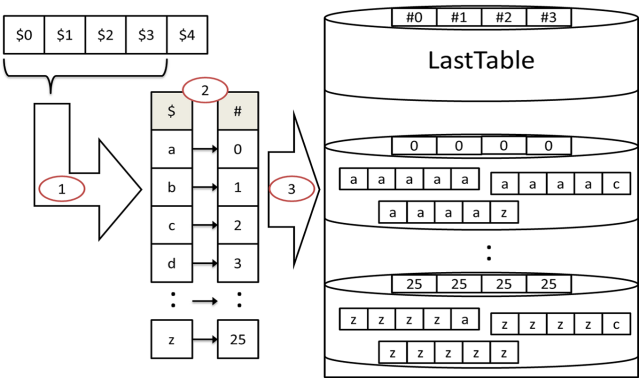

図 6 LastTable
Fig. 6 LastTable

表 1 ranka 環境
Table 1 Specification of the PC cluster used for evaluation

OS	CentOS 7.0
CPU	intel core i7 6700K 4.00GHz
MEM	32GB
コア数	4 コア
スレッド数	8 スレッド

5.2.2 LastTable

LW の先頭 4 文字をインデックスに用いる LastTable の実装を図 6 に示す．図 6 において，$0〜$4 はそれぞれ平文の 1〜5 文字目を表す．例えば，平文が“abcde”であった場合，$0=a，$1=b，$2=c，$3=d，$4=e である．図 6 中の① で$0〜$3 を得たのち，[a-z] をそれぞれ対応する数値の [0-25] に変換する．ここでは$0 の文字は“a”であるため，#0 に“0”を格納する．同様に$1〜$3 のそれぞれの値を変換し#1〜#3 にそれぞれ格納する（図 6 中の②）．#0〜#3 の数値でインデックスを割り当てグルーピングする．図 6 中の③ では，“aaaaa”，“aaaac”，“aaaaz”が，“0,0,0,0”のグループに属し，“zzzza”，“zzzzc”，“zzzzz”が，“25,25,25,25”のグループに属している．

6. 評価と考察

6.1 評価環境

提案手法を PC クラスタを用いて評価した．PC クラスタとは，ネットワークで接続された複数の計算機を 1 台の計算機として扱うことである．使用したクラスタを ranka クラスタ（以降，ranka と表記）と呼ぶ．ranka は表 1 の環境の PC が 4 台で構成される．

6.2 レインボーテーブル生成に必要なパラメータ

レインボーテーブルを生成するために，パラメータを事前に設定する必要がある．LHBC2+omp を実行するため

に必要なパラメータは 5 つあり，MPI 並列処理させるプロセスの数，OpenMP によるプロセス内並列させるスレッドの数，対象となる英小文字の平文の文字長，複数同時生成用のチェインの本数，生成が完了したチェインを分割するための分割値をそれぞれ指定する．今回の評価では，分割値を 5 に設定する．

6.3 レインボーテーブルの生成時間

6.3.1 LHBC2+LT+omp による計測

LHBC2+LT+omp による英小文字 2〜5 文字のそれぞれの平文に対応するレインボーテーブルの生成時間を図 7，図 8，図 9，図 10 に示す．pro はプロセス数，th はスレッド数であり，それぞれ MPI 処理，OpenMP 処理の並列数である．プロセス数とスレッド数の積が 32 であるのは，ranka 環境が 4 台で構成されるクラスタで 1 台最大 8 スレッドであるため，4 台 ×8 スレッドで最大 32 となる．

2〜5 文字のそれぞれの場合において，4 プロセス 8 スレッドの Hybrid MPI/OpenMP 並列処理での 100 チェイン同時生成が最も速い．通信処理を最小限に抑えたプロセス数による MPI と，使用可能なスレッドを最大限まで活用した OpenMP による Hybrid MPI/OpenMP 並列処理の効果である．

実験結果から，文字長の長い方が並列処理による高速化の効果が大きくなった．これは生成する平文の数が増加し，生成するレインボーテーブルに必要なチェインの数が増え，チェイン生成の並列化の効果が大きくなるからである．したがって，文字長を長くすること，文字種を増やしたときの対象とする平文の総数が多くなる場合のレインボーテーブル生成の高速化が期待できる．

6.3.2 従来手法と提案手法の比較

従来手法による生成時間を図 11，図 12，図 13 にそれぞれ示す．従来手法と提案手法では，提案手法によるレインボーテーブル生成が速い結果となった．これは，従来手法における通信処理のオーバヘッドが提案手法では削減され，OpenMP によるチェイン生成の高速化がされているためである．

7. 関連研究

Sykes らは，MPI を用いて MS-Windows のハッシュ値に対するレインボーテーブルを生成している [4]．レインボーテーブルの網羅率は 99.9% であり，生成に 6 日要している．Tabata らは，GPU を用いてレインボーテーブルを生成している [5]．本研究で用いる還元関数は一種類であるが，[5] では複数の還元関数を使用して衝突の発生を抑えている．

8. おわりに

本研究では，Hybrid MPI/OpenMP 並列処理による網羅率 100% のレインボーテーブル生成の高速化手法を提案し，英小文字 2〜5 文字のパスワードをすべて含むレインボーテーブルの生成に必要な時間を実際に測定した．

レインボーテーブルの生成を MPI および OpenMP により並列化する手法を述べた．並列化手法として MPI 並列処理では，生成済みの RS を各プロセスで分散させ保持す

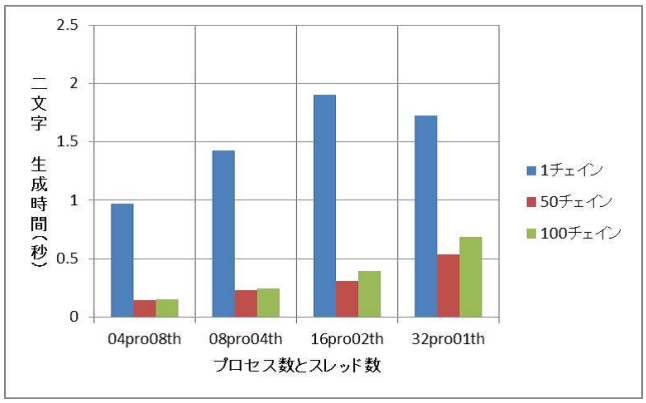

図 7 2 文字生成：LHBC2+LT+omp
Fig. 7 Generation time of rainbow table of 2 alphabet: LHBC2+LT+omp

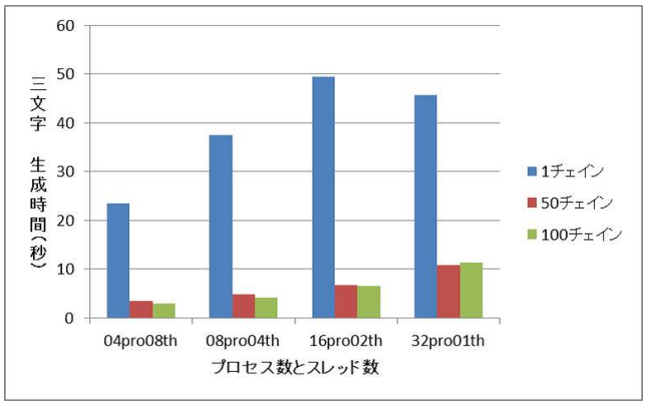

図 8 3 文字生成：LHBC2+LT+omp
Fig. 8 Generation time of rainbow table of 3 alphabet: LHBC2+LT+omp

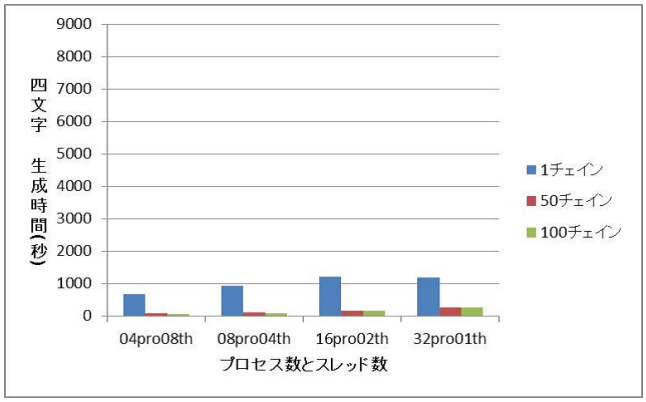

図 9 4 文字生成：LHBC2+LT+omp
Fig. 9 Generation time of rainbow table of 4 alphabet: LHBC2+LT+omp

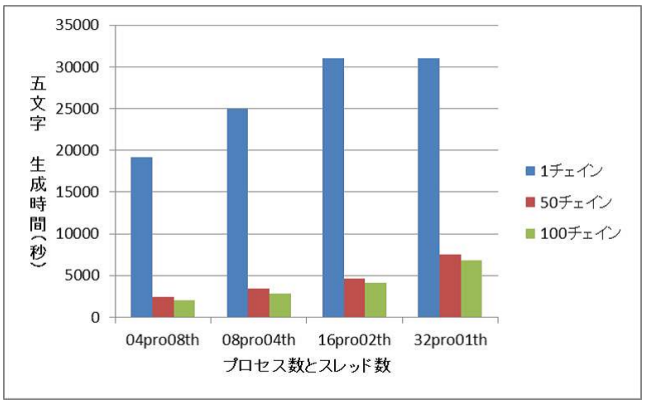

図 10　5 文字生成：LHBC2+LT+omp

Fig. 10　Generation time of rainbow table of 5 alphabet: LHBC2+LT+omp

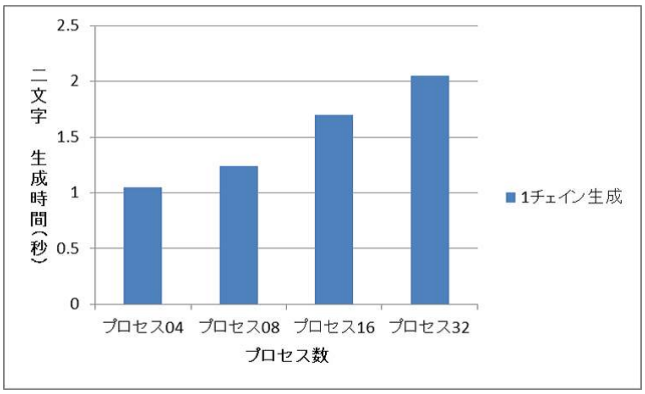

図 11　2 文字生成：LHB

Fig. 11　Generation time of rainbow table of 2 alphabet: LHB

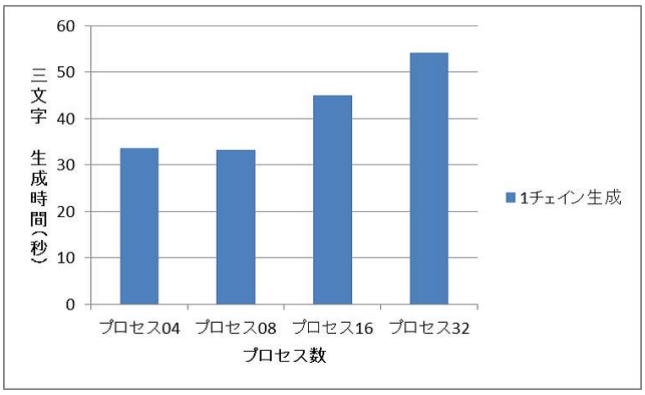

図 12　3 文字生成：LHB

Fig. 12　Generation time of rainbow table of 3 alphabet: LHB

ることで，衝突判定処理の高速化を実現し，OpenMP 並列処理では，複数のチェイン生成および衝突判定における処理をスレッド並列処理させることで高速化を実現した．加えて，高速化する技術として，LW 判定による衝突判定処理における無駄な判定の削減，LastTable を導入した LW 判定による LW を絞ることでの判定の高速化，およびチェイン分割による負荷分散の技術を導入した．

提案手法の効果を示すため，英小文字 2〜5 文字それぞれの平文に対応するレインボーテーブル生成の評価実験をおこなった．実験結果から，文字長を長くしたとき，文字種を増やしたときの対象とする平文の総数が多くなる場合におけるレインボーテーブル生成の高速化が期待できる．

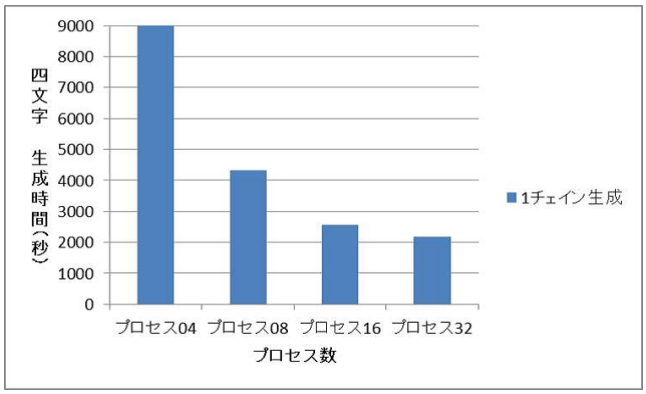

図 13　4 文字生成：LHB

Fig. 13　Generation time of rainbow table of 4 alphabet: LHB

今後の課題として，衝突における平文の重複削除の問題と複数同時生成の同期処理の効率化が挙げられる．本稿の手法では，LW 判定によって衝突を検知した場合、他の衝突が無いか二度目の重複判定をしている．この重複判定が生成を遅くしているため，LW 判定のみ行い高速化する．このとき，衝突が含まれる可能性があるが，生成速度を優先する．現在の提案手法では，チェインを静的に割り当てて複数同時生成を行い同期処理を行っている．複数同時生成しているすべてのチェインの生成が完了したとき，完了したチェインを RS として登録後，新たなチェインの複数同時生成を始める．この操作を，生成が完了したチェインから順次 RS として登録し，新たなチェインの生成を開始することにより，さらなる高速化が期待できる．

謝辞　本研究の一部は JSPS 科研費 15K00112 および 2016 年度南山大学パッヘ研究奨励金 I-A-2 の助成による．

参考文献

[1] 安藤公希，桑原寛明，上原哲太郎，國枝義敏：MPI 並列処理によるレインボーテーブル生成の高速化，コンピュータセキュリティシンポジウム 2015 論文集，Vol. 2015, No. 3, pp. 1335–1342 (2015).

[2] ウイリアムグロップ，ラジーブタークル，ユーイングラスク，畑崎隆雄 (訳)：実践 MPI - 2—メッセージパッシング・インタフェースの上級者向け機能，ピアソンエデュケーション (2002).

[3] 菅原清文：C/C++プログラマーのための OpenMP 並列プログラミング，カットシステム，東京, Japan，第 2 版 edition (2012).

[4] Sykes, E. R. and Skoczen, W.: An improved parallel implementation of RainbowCrack using MPI, *Journal of Computational Science*, Vol. 5, No. 3, pp. 536–541 (2014).

[5] Tabata, Y., Iwai, K., Tanaka, H. and Kurokawa, T.: Improved GPU Implementation of RainbowCrack, *Third International Symposium on Computing and Networking, CANDAR 2015, Sapporo, Hokkaido, Japan, December 8-11, 2015*, pp. 616–618 (2015).