ハイパフォーマンス
コンピューティング　93－25
（２００３．３．１２）

MMX 命令を利用したビット並列化塩基配列照合アルゴリズム

岸本　誠†　金子　敬一†

† 東京農工大学工学部

概要　ゲノム配列歩行は，生物研究におけるゲノムを使った解析手法のひとつである．ゲノム配列歩行で必要となるDNA 配列の近似的な照合に，文字列照合アルゴリズムである Wu らのアルゴリズムを利用することが提案されている．われわれは新規に，DNA 配列照合において記号の数が少ないことを利用したアルゴリズムを提案する．提案アルゴリズムは，64 ビットレジスタを利用できることや適した論理演算命令があることにより，MMX 命令を利用して効率的に実装できる．プログラムに実装し，従来のアルゴリズムを用いた同等の照合能力を持つプログラムとで実行に要する時間を測定し性能比較をおこなった．比較の結果，約 20 %の高速化を確認した．

A Matching Algorithm for Base Sequences with Bit-Parallel Approach Using MMX Instructions

Makoto KISHIMOTO† and Keiichi KANEKO†

† Faculty of Technology, Tokyo University of Agriculture and Technology

Abstract　Sequence walking is a method for genome analysis. Finding chains of base sequences needs an approximate matching algorithm. Wu and Manber have proposed such a text matching algorithm. In this paper, we propose a new algorithm that takes advantage of the small alphabet set of DNA. The algorithm can implement effectively with MMX, because it has 64-bit registers and useful logical operations. We have implemented our algorithm and compared execution time with an implementation of the algorithm by Wu and Manber. As a result, our algorithm has turned out to be faster by 20 percents than the algorithm by Wu and Manber.

1．はじめに

ゲノム配列歩行は，ゲノムの断片の配列にもとづき，接続する断片を探索し遺伝子全体の配列を決定する，という研究手法である．

三浦らのゲノム配列歩行システム [1] では，照合アルゴリズムに，Wu らによるアルゴリズムである Shift-And 法 [3] を利用している．われわれは，新規に DNA 配列の照合に適したビット並列化アルゴリズムを考案した．このアルゴリズムは，近年のパーソナルコンピュータ用マイクロプロセッサに用意されているマルチメディア向け SIMD 型拡張命令を用いた実装に適している．パーソナルコンピュータ上に，実際に照合をおこなうプログラムを実装し，従来のアルゴリズムを用いた照合プログラムとの性能比較をおこない高速化を確認する．

以下では，まず，配列照合と従来アルゴリズムについて説明し，次にわれわれの考案した新規アルゴリズムについて説明する．その後，実装とその評価について報告する．最後に結論を述べる．

2．従来アルゴリズム

まず，配列照合の詳細と，その照合のための従来のアルゴリズムについて説明する．

2.1　DNA 配列照合

ゲノム配列歩行においては，遺伝子の中のある断片の配列を，データベース中のほかの断片の配列と照合して，その断片に接続する断片を探し出すことが必要となる．

DNA を作る 4 種類のヌクレオチドは，A・G・C・T の 4 種類の記号であらわされる．よって，DNA の配列は，この 4 種類の記号を並べた文字列であらわされる．

塩基配列も文字列で表現可能であることから，基本的には文字列の照合と同じ方法によって照合が可能である．しかしながら，DNA の配列照合は，次のような点が通常の文字列の場合とは異なる．

- 部分的に 1 文字ぶん読み取り不可能である場合があり，その位置の記号は「不明」となる
- 完全に一致しないことがあるので，近似的な照合が必要である

- 記号集合 Σ の要素数 $|\Sigma|$ が 4 で，自然言語の文字のそれに比べ小さい
- 探索すべき列が，必ずある程度以上に長い

以上のような理由から，文字列検索のために一般に使われる KMP 法 [4] 及び BM 法 [5] ならびに Karp-Rabin 法 [6] は，DNA の配列照合には適していない．また，不一致部分が記号列中の任意のところに出現可能であることから，正規表現のためのアルゴリズムも適用できない．

特に，文字列に対して近似的な照合をおこなうアルゴリズムは，Wu らのアルゴリズム [3] 以外は知られていない．

2.2 編集距離

ある 2 つの記号列が，どれだけ似ているかを示すためのものとして，2 つの記号列の間に「編集距離」を定義する．

- 2 つの記号列が，完全に一致するならばそれらの間の編集距離は 0
- 片方の記号列中のある 1 文字を取り去ることで，編集距離を d にできるならばもとの 2 つの記号列間の編集距離は $d+1$
- 片方の記号列中のどこかに，ある 1 文字を付け加えることで，編集距離を d にできるならばもとの 2 つの記号列間の編集距離は $d+1$
- 片方の記号列中のある 1 文字を他の 1 文字に換えることで，編集距離を d にできるならばもとの 2 つの記号列間の編集距離は $d+1$
- 2 つの記号列間の編集距離を 0 にするために複数の編集方法がある場合は，より短くなるほうの編集距離とする

文字列検索における用語に従い，以下では，探す側の記号列を「パターン」，一致する部分列を探される側を「テキスト」と呼ぶ．

配列歩行において必要とされるアルゴリズムは，テキスト中から，パターンとの編集距離がある距離以下である部分列を見つけることができる照合アルゴリズムである．Wu らによるアルゴリズム [3] は，この目的にかなうものである．

2.3 Wu らによるアルゴリズム

ここでは，具体例を用いて Wu らによるアルゴリズムを簡単に説明する．詳細については Wu らによる論文 [3] を参照されたい．

記号集合を {'a', 'b', 'c', 'd', 'e'} とし，パターンを "aecbc" とする．このとき，以下のようにビット列を定める．

```
Ma = 01111
Mb = 11101
Mc = 11010
Md = 11111
Me = 10111
```

ビット列中の '0' が，パターン中における各記号の出現に対応していることに注意する．次にパターン "aecbc" における各記号の出現順にしたがい，"Ma, Me, Mc, Mb, Mc" の順にずらしながら並べたようすを示す．

```
   Ma = 01111
  Me = 10111
 Mc = 11010
Mb = 11101
Mc = 11010
```

ビット列中の 0 がある特定の列に並ぶことが見てとれる．

テキストの記号列中に出現する記号に対応するビット列を，出現位置に対応させながら重ね合わせて論理和を取ったとする．すると，パターンに一致する部分列が現れたところでは，対応するビットが 0 のまま残り，それ以外では 1 に設定される．

計算機のアルゴリズムとしては，この手続きを，レジスタのシフトとそのレジスタへの論理和演算として実現する．以下，このレジスタを A とする．

次に，処理するべきテキストを "aeaecbc" として考えてみる．

まず，レジスタ A の全ビットを 1 で初期化する．シフトによって空くビットには 0 が入る．以後，シフトと，テキストの各記号に対応するビット列との論理和をくりかえす． A の値の変化は以下のようになる．

```
A = 11111 （初期状態）
A = 01111 （シフト）
A = 01111 （Ma と論理和）
A = 00111 （シフト）
A = 10111 （Me と論理和）
A = 01011 （シフト）
A = 01111 （Ma と論理和）
A = 00111 （シフト）
A = 10111 （Me と論理和）
A = 01011 （シフト）
A = 11011 （Mc と論理和）
A = 01101 （シフト）
A = 11101 （Mb と論理和）
A = 01110 （シフト）
A = 11110 （Mc と論理和）
```

このように，パターンに一致する記号列があると，0 のビットが最右位置まで残ることがわかる．

テキストの記号 x に対する操作を式で表現すると以下のようになる．

$$\mathrm{A}' = (\mathrm{A} >> 1) | \mathrm{M}x$$

さらに，このレジスタ A をあらためて A_0 とおき，以下のように拡張することで，編集距離 1,2,3,... に対応する照合結果を保持するレジスタ群 $\mathrm{A}_1, \mathrm{A}_2, \mathrm{A}_3, \ldots$ を定義することができる．A_d において，0 のビットは，パターンとの編集距離が d 以下の部分列の存在を示している．

それぞれの編集操作に対して，対応する式を示す．論理和演

算で 1 になってしまっているビットのうちから，編集操作に応じて照合する可能性のある位置に対応するビットを論理積演算で 0 に戻す処理をおこなう，というのが基本的な考え方である．

以下の式中，A_d は，テキストに出現する記号に対応した論理和演算をする前の値で，A'_d は論理和演算をした後の値であることに注意する．

- 置換

$$A'_{d+1} \;\&= \; A_d >> 1$$

- 削除（テキスト中のある文字を削除すると一致する）

$$A'_{d+1} \;\&= \; A'_d >> 1$$

- 挿入（テキスト中に一文字適当に追加すると一致する）

$$A'_{d+1} \;\&= \; A_d$$

以上の処理は，ひとつにまとめることができる．まとめた式は以下のようになる．

$$A'_{d+1} \;=\; ((A_{d+1} >> 1)|Mx)\&((A'_d\&A_d) >> 1)\&A_d$$

2.4 問題点と改良案

Wu らによるアルゴリズムは，記号 1 個ぶんの情報を 1 ビットにして処理してしまうことで効率をあげている．DNA 配列照合においては，もともとの情報が 2 ビットであり，例えば一般の文字列の 8 ビットの場合に比べて，この利点が小さくなる．

この問題を解決するべく，われわれは，テキストのビット列をそのまま利用するアルゴリズムを考案した．以下でそのアルゴリズムを説明する．

3. 提案アルゴリズム

3.1 厳密な照合

記号集合を {0,1,2,3} とし，入力テキストを “0123” とする．記号 1 個は 2 ビットであらわされる．入力は，上位ビットのビット列と下位ビットのビット列で与えられるものとする．それぞれ H = “0011” と L = “0101” とする．

ここで，パターンの上位ビットと下位ビットの組合せ $(0,0),(0,1),(1,0),(1,1)$ に対して，それぞれ `H|L`, `H|~L`, `~H|L`, `~H|~L` という演算の組合せを考える．

実際にテキスト “0123” に対してこれを計算すると以下のようになる．

```
 H |  L = 0111
 H | ~L = 1011
~H |  L = 1101
~H | ~L = 1110
```

このように，論理否定演算と論理和の組合せにより，Wu らのアルゴリズムにおいて論理和演算によるマスクに利用したビット列に似たものが得られることがわかる．ただし，Wu らのアルゴリズムにおいてはパターンに対応しており，こちらはテキストの記号列に対応していることが基本的な違いである．

ここで，パターンを “0123” とする．

次のように入力テキストを左シフトしながら論理和を繰り返し，0 のまま残ったビットがあれば，パターンを見つけたことを意味することがわかる．

```
A = 0000
A |= ( H |  L)
H <<= 1, L <<= 1
A |= ( H | ~L)
H <<= 1, L <<= 1
A |= (~H |  L)
H <<= 1, L <<= 1
A |= (~H | ~L)
```

繰り返される操作を式にすると次のようになる．

$$A' \;|= \; \mathrm{mask}$$

ただし mask は前述のように，パターンの各ビット対に対応して H と L に論理否定を組合せて論理和をとったビット列である．

3.2 編集距離

3.1節で示したアルゴリズムを，編集距離に対応するように拡張する．

基本的な考え方は，Wu らのアルゴリズムの場合と同様である．レジスタ A をあらためて A_0 とおき，編集距離 1,2,3,... に対応するレジスタ群 $A_1, A_2, A_3, \ldots$ を導入し，論理和演算で 1 になってしまっているビットのうちから，編集操作に応じて照合する可能性のある位置に対応するビットを論理積演算で 0 に戻す処理をおこなう．

それぞれの編集操作に対する処理を表す式を示す．

- 置換

$$A'_{d+1} \;\&= \; A_d$$

- 削除（テキスト中のある文字を削除すると一致する）

$$A'_{d+1} \;\&= \; A_d << 1$$

- 挿入（テキスト中に一文字適当に追加すると一致する）

$$A'_{d+1} \;\&= \; (A_d >> 1)|\mathrm{mask}$$

これらの操作を全てまとめると，以下の式となる．

$$A'_{d+1} \;=\; ((A_{d+1}\&(A_d >> 1))|\mathrm{mask})\&(A_d << 1)\&A_d$$

ここで，シフト操作があるために，レジスタ全体を常に有効に使って処理ができないという問題点がある．編集距離 d までの照合を行う場合は，レジスタの幅から $2d$ ビットを引いた分を並列に処理できる．しかし，われわれの実装では，アラインメントを考慮して 64 ビットレジスタのうち，端を除いた 32 ビットを使って正しい処理になるようにしている．

4. 実　　装

評価のため，このアルゴリズムを実装した．以下では実装について述べる．

4.1 MMX

実装には，パーソナルコンピュータを使用した．近年のパーソナルコンピュータのプロセッサには，映像や音声などの処理を高速に実行するためのSIMD型拡張が備えられるようになっている．われわれは，Intel製プロセッサのMMXを提案アルゴリズムの実装に利用した．われわれが利用した機能について簡単に説明する．

- 64ビットレジスタ

MMXレジスタと呼ばれる64ビットレジスタを8個使うことができる．それぞれにはmm0 〜 mm7という名前がついている．MMX命令では，これらのどのレジスタも同等に利用できる．

従来の浮動小数点レジスタを転用しているため，MMXは浮動小数点関係の命令と同時に利用することができない．MMX命令を使用した場合，“EMMS”命令で終了をプロセッサに明示しなければならない．終了させずに浮動小数点の命令を実行しようとした場合，例外が発生する．

また，汎用レジスタでの演算におけるフラグレジスタに相当するものは，MMXには存在しない．

- 転送命令

MMXレジスタ間およびレジスタ・メモリ間の64ビット転送命令がある．また，MMXレジスタの下位側32ビットと汎用32ビットレジスタおよびメモリ間の32ビット転送命令がある．

32ビット転送でMMXレジスタが書き込まれる側の場合，上位側のビットにはゼロ拡張によって0が書き込まれる．

- 64ビットシフト命令

64ビットシフトをおこなう命令である．左シフト，右算術シフト，右論理シフトがある．MMX命令としては例外的に，オペランドに即値を使うことができる．回転命令はない．

- 64ビット論理演算命令

論理積（AND），論理和（OR），排他的論理和（XOR）の各論理演算命令がある．命令形式に1オペランド形式が存在しないため，論理否定（NOT）は無い．

他に，ANDNという命令があり，$r_1 = \tilde{}r_1 \& r_2$という型の演算ができる．

4.2 実行時のコード生成

提案アルゴリズムでは，照合するパターンに応じて論理演算のコードを変える必要がある．そこで，パターンに応じてアセンブリ言語のソースコードを生成してアセンブルしたのち，OSの動的ローディングを利用して照合を行うルーチンを呼び出している．

そのため，メインルーチンは，まずパターンからコードを生成するルーチンを呼び出し，次にアセンブラを起動して今生成したコードをアセンブルし，最後に動的ロードと呼び出しを行う．

OSにPC-Unix（FreeBSD）を利用したため，標準で用意されている動的ローディングによってこの処理は容易に実現できた．

4.3 実装の詳細

MMX命令の64ビットシフト命令と論理演算命令を利用して，提案アルゴリズムを実装した．

8本のレジスタのうち，入力を置くために2本，一時作業用に2本のレジスタを使うこととし，残りの4本を照合の途中経過の累加用にわりあてることとした．このことから，照合プログラムが受け入れる編集距離は3となる．

長さ32文字で誤りを3個とすると，約0.1のエラーを許容するものとなる．また，入力をランダムであると仮定した場合に長さ29（$32-3=29$）のパターンが偶然一致する可能性は，$1/4^{29}$　1.4×10^{-17}である．このことから，長さに関しては十分だが，エラーの許容範囲は狭いかもしれない．実際のアプリケーションでの利用における能力の評価については今後の課題である．

論理演算命令に$r_1 = \tilde{}r_1 \& r_2$という演算をおこなうANDN命令があることを利用するため，前節の説明とは双対的に論理を反転させた型に変換して実装した．

パターンの上位ビットと下位ビットのそれぞれの組合せに対する演算は以下のようになる．

```
(0, 0) : L = L & H
(0, 1) : L = ~L & H
(1, 0) : H = ~H & L
(1, 1) : L = ~(L & H)
```

このようにすると，すべて1命令ないし2命令で処理することができる．この手法では，パターンによって結果を得るレジスタが変わる．しかしながら，コードを動的に生成するので問題にはならない．

以下，実装をおおまかな擬似コードで示す．実際のコードはプログラムでパターンに対応したものが生成される．

- 初期設定

mm0のビット55 〜ビット0に入力のLを先頭から56ビット，mm1のビット55 〜ビット0に入力のHを先頭から56ビットセットする．ビット順は，MSB側に入力の先頭が対応する．

mm4 〜 mm7のビット55 〜ビット0を1に，他を0にセットする．

- 繰り返し（前半）

次の手順を，シフト幅16 〜 1で繰り返す

- mm0をmm2に，mm1をmm3にコピーする
- mm2とmm3をシフト幅だけ右シフトする
- 前述の論理演算をおこなう．結果の入っているレジスタをmaskとする
- 以下を$n=7, 6, 5$で繰り返す
  - mm$(n-1)$を右に1ビットシフト
  - mm$n$にmm$(n-1)$との論理和をとる

＊ mm$n$ に mask との論理積をとる
＊ mm$(n-1)$ を左に 2 ビットシフト
＊ mm$n$ に mm$(n-1)$ との論理和をとる
＊ mm$(n-1)$ を右に 1 ビットシフト
＊ mm$n$ に mm$(n-1)$ との論理和をとる
− mm4 に mask との論理積をとる

- データの追加

mm0 と mm1 を 16 ビット左シフトし，LSB 側 16 ビットにそれぞれの次の入力を論理和を使って書き込む．

- 繰り返し（後半）

「繰り返し（前半）」とまったく同じ処理をおこなう．

- 照合結果の判別

mm7 のビット 39 からビット 8 に 1 のビットがあるようであれば，一致を発見したので中断する．

- mm4 〜 mm7 の更新

mm4 〜 mm7 を左に 32 ビットシフトし，論理和を使って右側 32 ビットを 1 にする．

- データの追加

mm0 と mm1 を 16 ビット左シフトし，LSB 側 16 ビットにそれぞれの次の入力を論理和を使って書き込む．

- 「繰り返し（前半）」に戻る

以上のような手続きにより，照合アルゴリズムを実装した．

プログラムが実行中に生成するアセンブリ言語コードの例として一部を以下に示す．これは，パターンの 1 文字目が (H, L) = (0, 1) に対応するコードである．参考のためにコメントの記述を加えてある．

```
movq mm2, mm0
movq mm3, mm1
psrlq mm2, 15
psrlq mm3, 15
pandn mm3, mm2  ;  H := ~H & L

psrlq mm6, 1  ; A3 の処理
por mm7, mm6
pand mm7, mm3
psllq mm6, 2
por mm7, mm6
psrlq mm6, 1
por mm7, mm6

psrlq mm5, 1  ; A2
por mm6, mm5
pand mm6, mm3
psllq mm5, 2
por mm6, mm5
psrlq mm5, 1
por mm6, mm5

psrlq mm4, 1  ; A1
por mm5, mm4
pand mm5, mm3
psllq mm4, 2
por mm5, mm4
psrlq mm4, 1
por mm5, mm4

pand mm4, mm3  ; A0
```

表 1　測定に使用したコンピュータの仕様

CPU 品名	Intel Celeron Processor
CPU クロック周波数	400MHz
システムクロック周波数	66MHz
メモリ容量	128Mbyte
OS	FreeBSD 3.5.1

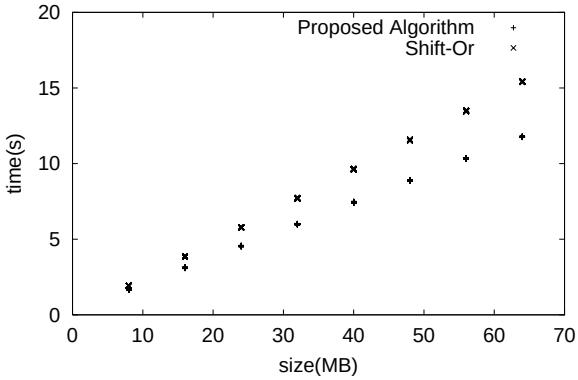

図 1　性能比較結果

5. 評　　価

5.1 測　　定

まず，提案アルゴリズムの実装と，Wu らのアルゴリズムによる同等の照合能力の実装を作成した．次に，疑似乱数列を書き出した異なるサイズのファイルを用意し，これらの各々のファイル中のデータに対して照合にかかる時間を測定した．測定に利用した計算機環境を表 1に示す．

5.2 結　　果

結果のグラフを図 1に示す．

実行時間には，提案アルゴリズムが必要としているコード生成ならびにアセンブルに要する時間も含んでいる．提案アルゴリズムの実装のほうが，すべてのデータサイズにおいて 20 % 程度高速であるという結果が得られた．

5.3 考　　察

提案アルゴリズムによる照合が高速である理由は，レジスタだけで全てを処理できていることと，繰り返しを完全に展開していることが主であると考える．現状では，編集距離のある照合を可能にするために，レジスタ全体をうまく使うことができていない．レジスタ全体をうまく使うことができれば，さらに高速化できるはずである．

6. おわりに

DNA配列照合アルゴリズムについて，新しいアルゴリズムを提案した．提案アルゴリズムは，入力データを直接内部処理用に利用していることとレジスタ間演算で処理が可能であることから，高速な実装が可能である．

確認のため，パーソナルコンピュータを利用した環境に提案アルゴリズムを実装し，同等の照合能力をもつ従来アルゴリズムの実装と実行時間を比較し，20 %程度の高速化を確認した．

より大きな編集距離にも対応することができるようにすること，レジスタ全体を使って64ビットずつ並列に処理することができるようになることが今後の課題である．

文　献

[1] 三浦 輝久，高瀬 俊郎，石田　亨，“ゲノム解析のための配列歩行システム”，電子情報通信学会論文誌，Vol.J84-D-I, No.1, pp. 108–115, Jan. 2001.

[2] Ricardo A. Baeza-Yates and Goston H. Gonnet, “A New Approach to Text Searching”, CACM, Vol.35, No.10, pp. 74–82, Oct. 1992.

[3] Sun Wu and Udi Manber, “Fast Text Searching Allowing Errors”, CACM, Vol.35, No.10, pp. 83–91, Oct. 1992.

[4] Donald E. Knuth, James H. Morris and Vaughan R. Pratt, “Fast Pattern Matching in Strings”, SIAM Journal of Computing, Vol.6, No.2, pp. 323–350, Jun. 1977.

[5] Robert S. Boyer and J. Strother Moore, “A Fast String Searching Algorithm”, CACM, Vol.20, No.10, pp. 762–772, Oct. 1977.

[6] Richard M. Karp and Michael O. Rabin, “Efficient Randomized Pattern-matching Algorithms”, IBM Journal of Research and Development, Vol.31, No.2, pp. 249–260, Mar. 1987.

[7] Intel Corp., “Intel Architecture MMX Technology Developer’s Manual”, Intel Corp, 1997.

[8] 竹田 正幸，篠原 歩，“圧縮されたテキスト上のパターン照合 — データ圧縮とパターン照合の新展開 —”，情報処理，Vol.43, No.7, pp. 763–769, Jul. 2002.

[9] Masayuki Takeda, Yusuke Shibata, Tetsuya Matsumoto, Takuya Kida, Ayumi Shinohara, Shuichi Fukamachi, Takeshi Shinohara and Setsuo Arikawa, “Speeding up String Pattern Matching by Text Compression: The Dawn of a New Era”, 情報処理学会論文誌，Vol.42, No.3, pp. 370–384, Mar. 2001.

[10] 外村 元伸，“ブロックソート圧縮データの検索法”，情報処理学会研究報告，Vol. AL-76-2, pp. 9–16, Jan. 2001.

[11] Takuya Kida, Masayuki Takeda, Ayumi Shinohara and Setsuo Arikawa, “Shift-And Approach to Pattern Matching in LZW Compressed Text” Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching, LNCS 1645, Springer-Verlag, pp. 1–13, 1999.