# Cell/B.E. 向け編集距離計算の高速化手法

田原 慎也
東京工業大学 大学院理工学研究科

Cell Challenge 2009 の規定課題において，文字列の編集距離を高速に求めるプログラムの開発に取り組んだ．本プログラミングコンテストでは，Cell Broadband Engine を対象としている．このマルチコア・プロセッサ特有のアーキテクチャは高い理論ピーク性能を誇るが，性能を十分に引き出すためにはそのアーキテクチャを熟知した上で様々なテクニックを駆使しなければならない．本開発では，文字列編集距離を求めるアルゴリズムとしてビット並列化した動的計画法を用いている．また，高速化のために，効率的なブロック化やローカルストアの有効活用，SPE 間の同期最適化，DMA ダブルバッファリングなどを実装した．

## 1 はじめに

Cell Broadband Engine(以下 Cell) は，1 つの汎用的なプロセッサコアである PPE と 8 つのマルチメディア処理に適したシンプルなプロセッサコアである SPE を持つ，ヘテロジニアスなマルチコア・プロセッサである．その特徴的なアーキテクチャは，非常に高い理論ピーク性能を誇るが，性能を十分に引き出すことは難しい．今回，Cell Speed Challenge 2009 の規定課題部門において，Cell 向けに文字列編集距離を高速に求めるプログラムの開発を行った．次節より，実装したアルゴリズムや Cell の性能を引き出すための高速化手法などを述べる．

## 2 求解アルゴリズム

今回の規定課題は，文字列編集距離計算の高速化である．文字列編集距離とは，２つの文字列が与えられたとき，それらの近さを表す１つの尺度である．具体的には，与えられた一方の文字列に対して，挿入，削除，置換の３種類の文字列操作を行って，もう一方の文字列に変形するのに必要な最短回数のことをいう．

編集距離は一般に動的計画法に用いて求められる．２つの文字列 $s$，$t$ の長さをそれぞれ $n$，$m$ とすると動的計画法により計算時間 $O(nm)$ で求められることが知られている．今回開発したプログラムは，動的計画法をビット並列化により高速化したアルゴリズムを用いた．このアルゴリズムは，ビット長 $w$ の値同士のビット演算が定数時間で可能な場合に計算時間 $O(\lceil n/w \rceil m)$ で求めることが可能である．SPE の場合，レジスタ長が 128 ビットなので $w = 128$ となる．このビット並列化アルゴリズムの詳細については，文献 [4, 5] を参考にされたい．

## 3 高速化手法

### 3.1 計算領域のブロック化

SPE 間の並列処理のために予め計算領域をブロック分割し，各 SPE に静的に割り当てた．具体的には，２つの文字列の長さに比例した $n$ 行 $m$ 列の計算領域に対して，128 行ずつに区切り各 SPE に割り当てることにした．そして，各 SPE は 128 列を単位として 128×128 のブロック領域ごとに計算することで並列処理の効率化をはかった．

ここで，ブロック間には依存関係が生じることに注意する必要がある．全てのブロックは，左と上のブロックの計算結果を用いて処理を行うため，それらのブロックの処理が完了するまで待機しなければならない．左のブロックについては，同じ SPE が処理するため，最左ブロックから最右ブロックに向かって処理を進めていけば問題ないが，上のブロックについては，別の SPE が処理するため，処理を完了したこととその計算結果を受け取る必要がある．つまり，各 SPE は最左ブロックから順に処理を進め，処理対象の上のブロックの計算が終わっていなければ待機し，終わっていればその結果を用いて計算をし，完了したら計算結果を下のブロックを担当する SPE に渡すことを繰り返す．これを右下のブロックまで行えば文字列編集距離を求めることができる．

### 3.2 ブロック化の改良

上記のブロック化により，各 SPE の並列処理を効率的に行うことができるようになった．さらに，SPE 単体での処理効率をはかるためにブロック領域の分割における改良を行った．ブロックの計算過程で，アルゴリズムで利用する 128 ビットのビットベクタ間の依存関係によりストールが生じてしまう．このストールを防ぐ為に，128×4

| 問題 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| n | 92,416 | 8,064 | 130,816 | 130,944 | 130,816 | 131,072 | 131,072 | 92,672 | 16,384 | 130,816 |
| m | 92,416 | 1,048,320 | 130,816 | 130,944 | 130,816 | 131,072 | 131,072 | 185,344 | 1,048,320 | 130,816 |
| 時間 (msec) | 81.84 | 77.55 | 16.81 | 145.82 | 142.91 | 148.98 | 149.18 | 144.45 | 17.78 | 68.16 |
| 順位 | 4 | 4 | 3 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |

表 1: 決勝ラウンド結果

行 128 列のブロックにして，依存関係のない計算を増やすことで解消することができる．

### 3.3 ビットベクタのテーブル参照

アルゴリズムにおいてマッチベクタ $PM$ と呼ばれるビットベクタを用いた計算が行われる．この $PM$ は，1 つの列に対して，その列に対応する文字と各行の文字との一致をビットで表したものである．例えば，ある列の文字 $e$ に対して，行に対応する文字列が $weight$ であれば，マッチベクタ $PM$ は 010000 となる．問題の設定上，文字の種類は 256 通りしかないため，予め 256 通りのマッチベクタをテーブルとして保存しておけば，文字列長のぶんだけのマッチベクタ生成する過程を省略できる．必要なときに文字に対応するインデックスを用いてテーブルからマッチベクタを取り出せばよい．このテーブルは，高速に取り出すことができるようにメインメモリ上ではなく，各 SPE のローカルストアに保存しておく．

### 3.4 SPE 間の同期の最適化

SPE 間の同期には，PPE やメインメモリを介さずに行うことで通信回数の削減，同期時間の短縮をはかった．必要な同期用の変数や計算結果を格納するバッファは，SPE のローカルストアに用意しておく．つまり，あるブロックを計算した SPE は，その結果を用いる SPE のローカルストアの適切な場所へ直接 DMA 転送を行うことで同期をシンプルかつ効率的に実装した．

### 3.5 DMA ダブルバッファリング

SPE は，SPU という演算処理を担当する機構と MFC という転送処理を担当する機構があり，それぞれ独立して動作することができる．これを利用して，複数のバッファを用意し，演算処理と DMA 転送処理を並行して行うことで転送にかかる時間を隠蔽することができる．これをマルチバッファリングといい，用意するバッファが 2 つの場合を特にダブルバッファリングという．今回，ダブルバッファリングを用いて転送時間の隠蔽をはかったが，もともと演算処理が支配的であったため大幅な改善は見られなかった．

## 4 結果

予選では，128 並列で処理できるビット並列化を用いずに SIMD 演算を用いた 4 並列でのアルゴリズムを実装した．実装したアルゴリズムは，最大問題サイズで 1.5 秒で文字列編集距離を求めることができた．本選では，ビット並列化したアルゴリズムにより最大問題サイズで 0.15 秒と予選と比較して約 10 倍の高速化となった．本選結果の詳細を図 1 に示す．

## 5 まとめ

今回の文字列編集距離を求める問題は，アルゴリズムの選択が重要であった．ビット並列化したアルゴリズムで 128 並列に処理を進めることで，大幅な高速化をはかることができた．

## 参考文献


[1] Sony Computer Entertainment Inc. Cell Broadband Engine$^{TM}$ アーキテクチャ Ver 1.01, 2006.

[2] Sony Computer Entertainment Inc. SPU C/C++ 言語拡張 Version 2.3, 2006.

[3] Gene Myers. A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming. Journal of the ACM, 46(3):395-415, 1999.

[4] Heikki Hyyrö. A Bit-Vector Algorithm for Computing Levenshtein and Damerau Edit Distances. 2002.