Cell BE を用いた文字列の編集距離計算の高速化

須藤 郁弥　篠原 歩
東北大学 大学院情報科学研究科

Cell Challenge 2009[1] の規定課題では，文字列の編集距離計算の高速化が競われた．我々は，動的計画法に対するビット並列アルゴリズムを基に高速化を行った．プログラムは，PPE で DP テーブルをブロックに分割して SPE に割り当て，SPE で割り当てられたブロックの計算を行うように構成した．また，SPE プログラムに対して求解アルゴリズムにおける 128 ビット加算の回避，ループアンローリング，even 命令の置き換えなどの高速化をし，さらに PPE で計算が不要であるブロックの検出を行うようにした．結果として長さ 2^{17} の 2 つの文字列に対して実行時間が 143msec. となった．これは実行委員会によるサンプルプログラムの約 300 倍の速度にあたる．また，編集距離が小さい場合は不要な計算の検出によりさらに実行時間を短縮できた．

1　はじめに

Cell Broadband Engine(以下 Cell BE) を対象としたマルチコアプログラミングコンテスト Cell Challenge 2009[1] の規定課題では文字列の編集距離計算の高速化が競われた．

文字列の編集距離は 2 つの文字列の近さを表す指標として用いられ，一方の文字列に挿入，削除，置換の 3 種類の操作を加えてもう一方の文字列を得るための操作の最小回数として定義される．コンテストでは制約として，与えられる 2 つの文字列の長さはいずれも 128 の倍数かつ $2^{20} - 128$ 以下であり，その積は 2^{34} 以下であると定められていた．

本稿では，コンテスト向けに開発し第 2 位を受賞したプログラムについて述べる．

2　プログラム概要

本節では採用した求解アルゴリズムおよび，PPE および SPE の役割について説明する．

2.1　求解アルゴリズム

与えられる 2 つの文字列 s，t の長さをそれぞれ n，m とする．一般に，編集距離計算に対しては動的計画法を用いた計算時間 $O(nm)$ のアルゴリズムが知られているが，我々のプログラムでは高速化のためワード長 w に対し計算時間が $O(n\lceil m/w \rceil)$ となるビット並列アルゴリズム [2, 3] を採用した．

編集距離計算に用いる DP テーブル D に対し，i 行 j 列の値を $D[i,j]$ と表す．D において，

$$\begin{aligned} &| D[i,j] - D[i,j-1] | \leq 1 \\ &| D[i,j] - D[i-1,j] | \leq 1 \\ &0 \leq D[i,j] - D[i-1,j-1] \leq 1 \end{aligned} \tag{1}$$

となることを利用し，ビットベクタ $D0_j$，HP_j，HN_j，VP_j，VN_j を次のように定める．

$$\begin{aligned} D0_j[i] = 1 \quad &\text{iff} \quad D[i,j] = D[i-1,j-1] \\ HP_j[i] = 1 \quad &\text{iff} \quad D[i,j] - D[i,j-1] = 1 \\ HN_j[i] = 1 \quad &\text{iff} \quad D[i,j] - D[i,j-1] = -1 \\ VP_j[i] = 1 \quad &\text{iff} \quad D[i,j] - D[i-1,j] = 1 \\ VN_j[i] = 1 \quad &\text{iff} \quad D[i,j] - D[i-1,j] = -1 \end{aligned}$$

また，各文字 λ に対し，ビットベクタ PM_λ を次のように定める．

$$PM_\lambda[i] = 1 \ \text{ iff } \ t[i] = \lambda$$

アルゴリズムでは，第 $(j-1)$ 列のビットベクタから以下の関係式により第 j 列のビットベクタを求める．

$$\begin{aligned} D0_j &= (((PM_{s[j]} \& VP_{j-1}) + VP_{j-1}) \hat{} VP_{j-1}) \\ &\quad | PM_{s[j]} | VN_{j-1} \\ HP_j &= VN_{j-1} | \tilde{} (D0_j | VP_{j-1}) \\ HN_j &= D0_j \& VP_{j-1} \\ VP_j &= (HN_j << 1) | \tilde{} (D0_j | ((HP_j << 1) | 1)) \\ VN_j &= D0_j \& ((HP_j << 1) | 1) \end{aligned}$$

関係式を繰り返し用いることにより，s と t の編集距離 ed は次式により求められる．

$$ed = n + \sum_i VP_n[i] - \sum_i VN_n[i] \tag{2}$$

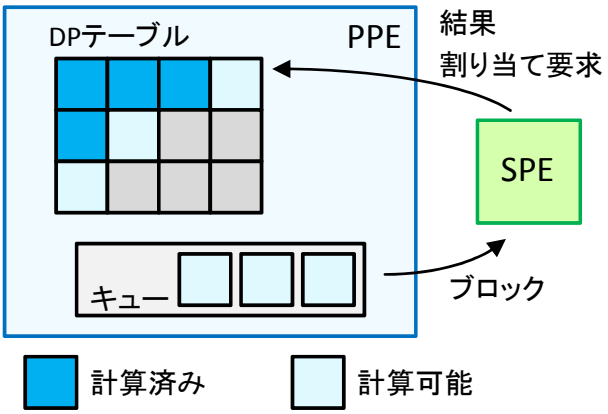

図 1: PPE によるブロックの割り当て.

2.2 PPE プログラム

我々のプログラムでは，D をサイズ $(128 \cdot 10) \times 128$ のブロックに分割して計算を行った．各ブロックはその上側および左側のブロックの計算が終了していれば計算可能となるため，PPE ではキューを用いて計算可能なブロックを管理し SPE への割り当てを行った (図 1)．また，4 節に述べる計算が不要なブロックの判定および，式 (2) による編集距離の計算も行った．なお，ブロックのサイズは，サイズ 128×128 のブロックを行方向に 10 個並べた分に相当し，ブロックサイズを大きくすることで PPE-SPE 間通信の回数および PPE の処理量を削減している．

2.3 SPE プログラム

ビットベクタ gHP_i，gHN_i を次のように定義する．

$$gHP_i[j] = HP_j[i]$$
$$gHN_i[j] = HN_j[i]$$

gHP_i および gHN_i は，DP テーブルの行方向の値の変化を行方向に連続して持つビットベクタである．

SPE が PPE から受け取るブロックの最上段が DP テーブルの第 x 行，最左列が第 y 列であるとする．SPE は，PPE からブロックの情報としてその位置，最左列の状態を表すビットベクタ $VP_y[x+1...x+128]$，$VN_y[x+1...x+128]$，最上段の状態を表すビットベクタ $gHP_x[y+1...y+1280]$，$gHN_x[y+1...y+1280]$ を受け取り，受け取った情報を基に編集距離計算を行う．結果として最右列に関して $VP_{y+1280}[x+1...x+128]$，$VN_{y+1280}[x+1...x+128]$，最下段に関して $gHP_{x+128}[y+1...y+1280]$，$gHN_{x+128}[y+1...y+1280]$ を返す．

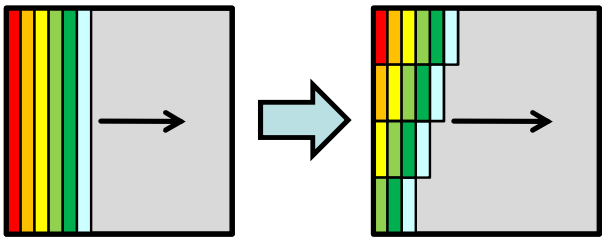

図 2: ビットベクタの計算．128 ビットで同じ列を管理 (左) せずに，32 ビットごとに管理する列を 1 列ずつずらし (右)，遅延の解消に成功した．

3 SPE プログラムの高速化

本節では，SPE プログラムに対して行った高速化について述べる．チューニングにおいては [6, 7] を参考にした．

3.1 128 ビット加算の回避

ビットベクタ $D0_j$ の計算には加算が用いられており，128 ビットに同じ列の情報を保持した場合は 128 ビット整数の加算が必要となる．しかし，Cell BE では 128 ビット加算は命令として用意されておらず，32 ビット加算を用いて実装する必要がある．このとき，128 ビット加算部分において依存関係のある命令が多く並ぶため，この部分で依存遅延が発生した．そのため，我々のプログラムでは 32 ビットごとに管理する列を 1 列ずつずらす (図 2) ことで 128 ビット加算を回避し，加算部分を SIMD 命令として用意されている 32 ビット加算 $\times$ 4 により行った．この変更によってブロックの両端における処理が増えるが，128 ビット加算による依存遅延が解消できたために 1.2 倍程度の高速化に成功した．

3.2 データ転送

ブロックの計算に必要な情報はその位置，ビットベクタ VP，VN，gHP，gHN，および対応する部分文字列であるが，このうち位置情報とビットベクタはサイズが小さいため，PPE-SPE 通信に用いる構造体にその値を詰め込み，データ転送の回数が少なくなるようにした．また，アルゴリズムでは最初に前処理として各文字 λ に対して PM_λ を計算するが，この計算には一方の文字列しか用いないため，もう一方の文字列は前処理の間に取得するようにし，データ転送時間を隠蔽した．

3.3 パイプライン最適化

まず，ループアンローリングにより 1 度の反復で 32 列を処理するようにした．また，アルゴリズムはビット演算が大半であるため，加算や要素単位のビット演算を含む even 命令が多くなるが，それらを可能な限り odd 命令で代用することにより，2 命令同時発行が多くなるようにした．具体的には，シャッフル命令により OR 命令やビットシフト，ビット選択を置き換え，さらに最下段の gHP，gHN の計算については，16 列ごとに処理することでシャッフル命令，ギャザー命令などの odd 命令のみで実現した．

4 計算不要なブロックの検出

本節では，PPE で行った計算不要なブロックの検出およびその実装について述べる．

4.1 編集距離の上界を用いた検出

$ed \leq u$ かつ $m \leq n$ であるとする．このとき，次式を満たす i，j に対してのみ $D[i,j]$ を求めれば良いことが知られている [4, 5].

$$-\left\lfloor \frac{u-n+m}{2} \right\rfloor \leq j-i \leq \left\lfloor \frac{u+n-m}{2} \right\rfloor \tag{3}$$

式 (3) をみたさない領域を通る経路のコストは必ず u を超えるため，この領域の値は解に影響しない．我々のプログラムでは u として，2 つの文字列の接頭辞 m 文字のハミング距離と文字列長の差 $n-m$ の和を用いた．これは，先頭から文字を比較して一致しない位置のみ置換を行い，s の残り $(n-m)$ 文字を削除するという戦略で文字列操作を行った場合の操作回数に相当する．

例えば，$n=m$ の場合を考えると，式 (3) は，$-\lfloor u/2 \rfloor \leq j-i \leq \lfloor u/2 \rfloor$ となり，最悪時 $(u=n)$ でも 25 % 程度の計算を削減できる (図 3).

4.2 ブロック周囲の値を用いた検出

ブロックの最上段が単調増加，最左列が単調減少である場合は，テーブル D の性質 (1) から文字列によらず結果は最下段が単調増加，最右列が単調減少となるため，計算不要である．このようなブロックは編集距離が小さい場合に多く出現し，多くの計算の削減が期待できる．

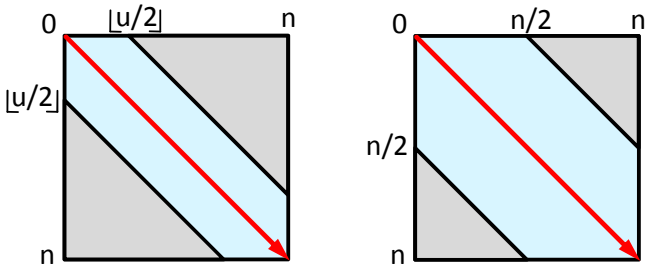

図 3: $n=m$ における計算不要な領域．u は図の赤矢印の経路をたどったときのコストに相当し，グレーで示した部分は計算不要となる．右図はこの場合における最悪時.

4.3 実装

以上の 2 種類の性質から計算が不要と判定されたブロックについては，受け取った最左列の情報を最右列のものとして，最上段の情報を最下段のものとしてそのまま返して良い．このとき，DP テーブルの最上段，最左列は定義から単調増加で初期化されているため，4.1 節の条件で計算不要となる領域の値は十分大きいものとして認識される．

4.2 節の条件判定は，ブロックの最上段，最左列の値すべてを見るのではなく，ブロックの列方向のサイズである 128 行ごとに値が単調増加となる範囲を覚えることで行った．例えば，第 0 行については初期条件から全体で単調増加であるため，最左列が単調減少となるブロックが現れると，その左のブロックはすべて 4.2 節の条件を満たすブロックとなる．よって，このブロックが現れた列以降で第 128 行の値は単調増加となるため同様の議論が成り立つ．図 4 に例を示す．なお，この条件で判定を行うと実際には 4.2 の条件をみたすブロックに対し計算が行われる場合があるが，そのようなブロックは全体に対してごく少数であり，条件判定は値すべてを見るよりも高速であるため，全体として高速となる．

なお，この実装において編集距離は計算不要なブロックの判定を行わない場合と同様に式 (2) を用いて計算できる．

5 結果

コンテストにおける問題サイズの制限 $nm \leq 2^{34}$ において，実行委員会によるサンプルプログラムでは，最大サイズの求解に 43 sec. 必要であったの対し，我々のプログラムでは最大サイズで $n=m$ の場合に 143 msec. で答

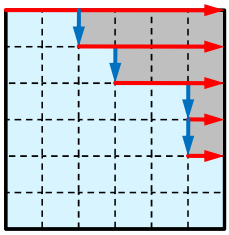

図 4: 4.2 節による計算不要なブロックの検出．図中で赤矢印で示した部分は値が単調増加，青矢印で示した部分は値が単調減少を示しており，灰色で示したブロックは計算不要と判定される．

えを出力できるようになり，300 倍程度の高速化に成功した．

コンテスト本選では 10 問の問題によりプログラムの評価が行われた．結果は表 1 の通りであった．問題 2, 8, 9 は s が t の部分列であり，この場合は編集距離が $m - n$ となる．我々のプログラムでは，前処理として部分列判定を行っていたため，この 3 問では部分列判定のみを行って答えを出力した．また，ほとんどの問題で 2 位以内の実行時間を記録しており，我々が行った高速化が有効であったことが示された．1 位を獲得した問題 4, 10 では，u が答えに対して非常に良い見積もりとなり，高速に答えを出力することができた．

現状で我々のプログラムで確認されている問題点として，実行ごとに実行時間が 50 msec. 程度遅くなる場合があることが挙げられ，本選では問題 5, 7 において同現象が発生した．原因としては，PPE-SPE 間で同期を取る際に DMA 転送を用いてスピンロックを行っていたことが挙げられる．Cell BE では DMA のコンジェスチョンが起こると発行された DMA 命令がしばらく実行されない場合があり，このために動作が不安定になったと考えられる [8].

6 まとめ

Cell Challenge 2009 では編集距離計算の高速化に取り組み，ビット並列アルゴリズムを基に，128 ビット加算の回避や計算不要なブロックの検出を行った．結果としてサンプルプログラムの 300 倍の実行速度を達成し，コンテストでは計算不要なブロックの検出が特に効力を発揮し，高い評価が得られた．また，PPE-SPE 間の通信方式の改良によりさらなる高速化が見込まれる．

表 1: 本選結果

問題	n	m	ed	時間	順位
1	92,416	92,416	86,379	79.6	2 位
2	8,064	1,048,320	1,040,256	2.6	1 位
3	130,816	130,816	1	28.5	5 位
4	130,944	130,944	61,276	92.4	1 位
5	130,816	130,816	130,816	193.3	4 位
6	131,072	131,072	2	81.7	2 位
7	131,072	131,072	256	130.9	2 位
8	92,672	185,344	92,672	0.9	1 位
9	16,384	1,048,320	1,031,936	2.7	2 位
10	130,816	130,816	6	29.7	1 位

時間は msec.

参考文献

[1] Cell Challenge 2009 実行委員会. Cell Challenge 2009 http://www.hpcc.jp/sacsis/2009/cell/

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

[3] H. Hyyrö. Explaining and Extending the Bit-parallel Approximate String Matching Algorithm of Myers. Technical Report A-2001-10, University of Tampere, Finland, 2001.

[4] E. Ukkonen. Finding approximate patterns in strings. *Information and Control*, 64:100-118, 1985.

[5] H. Hyyrö. A Bit-Vector Algorithm for Computing Levenshtein and Damerau Edit Distances. *Nordic Journal of Computing*, 10:1-11, 2003.

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

[7] IBM. IBM Assembly Visualizer for Cell Broadband Engine, 2007.

[8] 桜庭　俊, 吉田　悠一. Cell プロセッサを用いた編集距離計算の高速化. 先進的計算基盤システムシンポジウム SACSIS 2009 論文集, 97, 2009.