Cell を用いた編集距離計算の高速化手法

奥田 遼介　高橋 大樹　鈴木 貴樹
一関工業高等専門学校

Cell Challenge 2009 規定課題部門の課題である文字列の編集距離を求めるプログラムの高速化について報告する．高速化はコンテストで配布された動的計画法を利用したサンプルプログラム [1] に改良を加えることで行った．

1　開発の背景

今回の問題は,2 つの文字列を入力してその編集距離を求めるものであった．編集距離問題とは, ある文字列に 1 文字追加,1 文字削除, 1 文字置換を行う事で別の文字列に編集する作業の最小回数を求める問題である [2]．この問題には亜種が多いが, 今回の問題では 3 つの編集のコストが全て 1 であるものとした問題が出題された．

この問題のアルゴリズムとしては動的計画法が知られている．しかし, この動的計画法は計算途中のデータ間の依存が強く, そのまま適用した場合 Cell Broadband Engine(以下 Cell) のパイプラインがストールしてしまう．この問題の解決が高速化に重要だという認識の基でプログラムの改良を行った．

2　設計

編集距離の動的計画法 (Dynamic Programming) による解法は, 与えられた 2 つの文字列の長さを n_1,n_2 とすると,$O(n_1 n_2)$ で解ける．これは,$n_1 \times n_2$ の 2 次元領域において, 任意の要素 (i,j) を順次決定するもので, (n_1,n_2) に決定された整数が解となる．さて,(i,j) を決定するには,$(i-1,j)$,$(i,j-1)$,$(i-1,j-1)$ が決定している必要がある．この制約により,(i,j) と $(i+1,j)$ や,(i,j) と $(i,j+1)$ を同時に求めることは不可能であることが分かるだろう．

Cell は SIMD によって 32bit Integer4 つに対して同時に演算可能である．つまり, 高速化を行うためには同時演算を行ったときに依存関係が発生しないようにデータ構造を変更する必要がある．今回は, 依存関係がない $(i+3,j)$,$(i+2,j+1)$,$(i+1,j+2)$,$(i,j+3)$ のような斜め 4 マスを演算の基本単位として利用した．また,Cell に搭載されている Synergistic Processor Element （以下 SPE）を 7 基使用可能だったので, これにも斜め方向に依存関係が発生しないことを利用し, 演算領域の割り当てを行った．

以上のデータ構造に関する変更の他に, 次のような点を考慮した．

- 条件分岐の排除
- ループの展開
- 命令発行の充填
- 通信回数の削減
- コンパイル後のプログラムサイズ

3　実装

3.1　データ構造

Cell の SPE に搭載された LocalStore （以下 LS）は 128bit のアライメントに沿ったデータのロードストアのみがアトミックに行える [3]．アライメントに沿わないデータの場合には何回かのロードストアによって操作を実行する．つまり,128bit を一単位としない操作が増えると演算時間が増加することになる．従って,32bit Intege4 つを演算の基本単位とした．ただし,8bit char 型で提供される文字列は, shuffle 演算を用いて Integer に拡張することで演算のサイズを統一している．

SPE は 0 番から 6 番の 7 基が用意されている．そこで,i 番の SPE は 2 次元領域の横方向インデックスが $128i \bmod 7$ から $(128(i-1)-1) \bmod 7$ の短冊形領域の計算を担当する（図 1）．つまり,i 番の SPE の挙動は次の通りとなる．

1. $(i-1) \bmod 7$ の演算結果を受け取る．
2. 担当領域を縦方向に 128 文字分の演算処理する．
3. 演算結果を $(i+1) \bmod 7$ 番の SPE に非同期転送する．

3.2 条件分岐の排除

SPE は静的分岐予測機構しか持たず, またパイプラインが深いため分岐ミスを起こした場合のコストが大きい. その代わりに, 比較演算を行いマスクを生成する命令とマスクに基づき選択を行う select 命令が存在する [6]. 今回は,(i,j) の決定する際 $(i-1,j)$,$(i,j-1)$,$(i-1,j-1)$ の値比較を必要とするが, 上記の命令を組み合わせることによって, 条件分岐を行わずに済む.

3.3 ループの展開

分岐予測の排除を行うに当たって, 小さなループの継続判定は大きなコストとなる. 今回はマクロを使うことによって, ループの展開を行っている. また, ループの展開はレジスタ割り付けやメモリアクセスのコードの依存関係が最適化されるため, 条件判定を単純に削除するよりもパフォーマンスが向上する. しかし, 生成されるコードが膨大となる場合もあり, サイズが 256KB しかない SPE の LS を圧迫する可能性がある. そのためループの展開前に, 十分にループ内の命令数を減らす必要がある. 今回は, ループ中に必要になる演算が少なくなるように前処理を施す, データの並べる向きを工夫する, などを行った.

3.4 命令発行の充填

上述のように, 斜め方向に要素を決定することにより依存関係は減る. しかし SPE での 1 演算の実行には最低 2cycle かかる. この制限の中で単位時間あたりの命令実行数を増やすために, 斜め 4 マス ×2 列分の演算を同時に行うようにした（図 2）. これにより, 実行ユニットの中の 1 つの命令充填率がほぼ 100%となった. 命令の依存関係チェックには Assembly Visualizer を利用した [4].

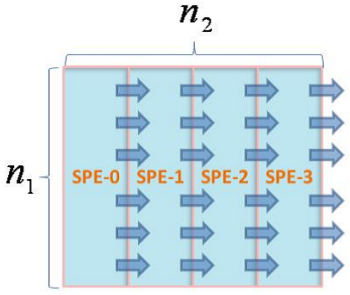

図 1: SPE の動作

3.5 通信回数の削減

3.5.1 LS へのキャッシュ

Cell にはアクセスが高速な LS 領域が 256KB 存在する. 今回の入力として与えられる文字列は,$n_1 n_2 \leq 2^{34}$ という条件を満たし, どちらかの文字列は 128KB 以下のサイズになる. よって,128KB 以下の文字列を LS に最初に転送することで,256KB の LS 領域に一方の文字列をキャッシュし, メモリアクセスを極力減らした.

3.5.2 SPE 間通信

SPE は演算ユニットである Synergistic Processor Unit（以下 SPU）, SPE 専属のメモリである LS, 外部との通信を行う Memory Flow Controller (以下 MFC) によって構成されている [5]. MFC による通信は,LS-メインメモリとの通信のほかにも,LS-LS 通信を行うことが可能である. LS-LS 通信を行うことで, メインメモリを介さずにデータ転送や同期動作を行うことができる. これにより, メインメモリにアクセスが集中することがなくなり, 高速な動作が期待できる. ところで,SPE 間通信を行う場合には, 相手の LS 上のアドレスを知ることが必須となる. 今回は, すべての SPE が同一の実行イメージを利用する形をとったために, 自分の LS アドレスから相手のアドレスが推測可能となっている.

3.5.3 入力サイズに応じた動作変更

以上の工夫を行った場合でも SPE 間の同期をとる必要はなくなるわけではない. この同期をとることによって, 入力される文字列長が短い場合にパフォーマンスの悪化が発生する. この対策として, どちらの文字列も 16KB 以下のサイズの場合は 1 SPE による処理を行うことにした. 16KB というサイズは, 今回制作したプログラムにおいて 256KB の LS に演算の一時データをすべて載せることが可能となるデータサイズである.

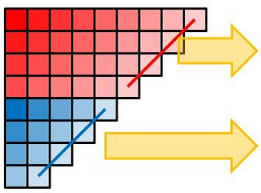

図 2: 斜め 4 マス ×2 列分の演算

3.6 プログラムサイズ

今回コンパイルオプションとして-std=gnu99 を利用した. このオプションは GNU 拡張と C99 を兼ね備えたコーディングを利用可能とするものである. オプション指定によってマクロの拡張, 変数宣言位置の自由化,restrict 指定による明示的な引数の依存関係指定が可能となった.

つまり,1 つの関数で書くことや, 全てをマクロで書くことができれば出力されるプログラムを容易に予想可能になる. しかし, ソースコードは複雑になり保守が不可能となる. 今回は上記の表記方法の他に static inline の利用による関数の完全展開, 配列変数のスタックへの配置によって, 関数を使ってのコーディングを行いながら出力されるプログラムサイズや変数領域のサイズを調整した. 特に, 今回は 2 系統の実装を載せているため, スタックへの変数の確保は大きな効果となった.

4 評価結果

表 1 は各サイズに対する実行結果を示したものである.

全体的な傾向として, 問題サイズが大きくなるほど 1 マスにかかる時間が減少している. これはプログラムの SPE への転送や, 文字列キャッシュの読み込みのような固定的なオーバーヘッドによるものと考えられる. No11,12 は No1,2 と同一のデータを 7 SPE で実行したときのスコアである. 入力サイズが小さい場合には, 固定的なオーバーヘッドを下げると同時に, 通信を行わない方が有利であることがわかる. No9 のスコアは縦方向の文字列長が 128 文字であることから同期待ちが頻繁に発生して実行時間が長くなっていると考えられる.

5 まとめと今後の課題

データ構造の改良, 命令の 2 列同時発行,SPE への文字列キャッシュ,SPE 間通信を用いることでサンプルプログラムを高速化することが出来た. しかし, 隣接部分の差に注目するという考えに至らなかったため上位チームに水をあけられる結果となった. また通信部分のプロファイリングを行っていないためチューニングが不十分であったことも考えられる.

参考文献

[1] Cell Challenge 2009.
http://www.hpcc.jp/sacsis/2009/cell/.

[2] レーベンシュタイン距離 - Wikipedia
http://ja.wikipedia.org/wiki/レーベンシュタイン距離

[3] 4.4 ベクタデータのアラインメント - PS3 Linux Information Site / Cell/B.E. のパワーを体験しよう
http://cell.fixstars.com/ps3linux/index.php/4.4

[4] alphaWorks : IBM Assembly Visualizer for Cell Broadband Engine : Overview
http://www.alphaworks.ibm.com/tech/asmvis

表 1: 実行結果

No	$n_1[byte]$	$n_2[byte]$	サイズ	時間 $[sec]$	1 マスの処理時間 $[pSec]$	備考
1	5k	5k	25M	0.030	1200	1SPE
2	10k	10k	100M	0.094	940	1SPE
3	20k	20k	400M	0.179	448	
4	50k	50k	2.5G	0.333	133	
5	60k	60k	3.6G	0.435	121	
6	65k	65k	4.2G	0.559	132	
7	72k	72k	5.2G	0.636	123	
8	128k	128k	16G	1.594	97	
9	128	1M	128M	0.245	1914	
10	10k	1M	10G	1.080	108	
11	5k	5k	25M	0.127	5060	7SPE
12	10k	10k	100M	0.129	1295	7SPE

[5] Sony Computer Entertainment Inc.
Cell BroadBand Engine^{TM} アーキテクチャ Version 1.0, 2005.

[6] Sony Computer Entertainment Inc. SPU C/C++ 言語拡張 Version 2.1, 2005.