Cell/B.E. における SIMD 論理関数を利用した 編集距離計算の実装法

里城 晴紀　吉瀬 謙二　小長谷 明彦
東京工業大学 大学院情報理工学研究科

Cell Challenge 2009 の規定課題である編集距離計算プログラムの設計と実装について報告する．編集距離計算は隣接する要素間の差分行列を用いた動的計画法にアルゴリズムを変形することにより，SIMD 命令をより効率良く使用することが可能となる．特に，距離情報はビット情報で表現すると，編集距離の特徴から隣り合う要素間の差は 3 通りしかなく，わずか 2 ビットで表せる．このとき行列要素計算の入力ビットパターンと出力ビットパターンの組み合わせは 18 通りとなる．そこで計算式を論理関数で表し，SIMD 論理演算を利用して実装した．SIMD 論理演算を用いることで必要な計算ステップ数は増加するが，128 倍の並列計算が可能となり，トータル性能において整数演算（16 倍) の実装を凌駕した．さらに，不要計算部分の枝刈りならびに SPE 間通信の最適化などを行うことにより，最終的に，規定課題のサンプルプログラムの約 300 倍の性能を達成した．

1　はじめに

Cell Challenge 2009 規定課題部門に用いた編集距離計算プログラムの設計と実装について報告する．Cell/B.E. における並列化は，大きく分けて，SPE を用いたスレッドレベルの並列化と SIMD 命令を用いた命令レベルの並列化がある．編集距離計算のような大量データ処理を高速化するためには領域分割を活用したスレッドレベル並列処理，計算不要な領域の枝刈りと共に，各スレッド計算において，いかに SIMD 命令を有効に活用するかが高性能を実現するための鍵となる．SPU では 128 ビットのレジスタが利用可能であり，各情報を独立した 1 ビットで表現できればSPU の SIMD 論理演算命令を用いることにより，128 倍の並列性を引き出すことが可能となる．ただし，実際の性能向上は論理演算にまで計算を分解したときの処理ステップ増加数とのトレードオフとなる．本稿では，論理関数として与えられた計算アルゴリズムに対して Quine-McCluskey 法 [1] を適用することにより，実現に必要な SIMD 演算命令列を効率よく生成できることを示す．

2　設計と実装

提出したプログラムの設計と実装を述べる．主な高速化手法は論理関数化，枝刈り，通信の最適化である．提出時の高速化比はツールキットのサンプルプログラムのおよそ 300 倍で，内訳は論理関数化で 100 倍，枝刈りで 2 倍，通信の最適化で 1.5 倍である．

2.1　編集距離の差分表現

SIMD 演算を使って計算するときは，できる限り小さいデータ型を使用することで並列処理数が増加し高速化できる．編集距離の計算においては計算に使用する行列の要素間の差が -1，0，$+1$ の 3 通りになることを利用して，すべての行列要素を 2 ビットのデータで表現できる．

編集距離は行列を用いた動的計画法によるアルゴリズムで計算する．文字列 a, b を入力とするとまず，$|a|+1$ 行 $|b|+1$ 列の行列を用意し，0 行目と 0 列目をそれぞれ列番号，行番号で初期化する．残りの行列要素は次の式で計算する．

$$c_{ij} = \begin{cases} 0 & (a_i = b_j) \\ 1 & (\text{otherwise}) \end{cases}$$
$$d_{ij} = \min(d_{i-1,j}+1, d_{i,j-1}+1, d_{i-1,j-1}+c_{ij}) \quad (1)$$

ここで $i>0$，$j>0$ であり，a_i は文字列 a の i 番目の文字，b_j は文字列 b の j 番目の文字，d_{ij} は行列の（i,j）要素である．図 1 に 2 つの文字列 weight と write の距離を求めたときの行列を示す．右下の行列要素の値が編集距離である．

動的計画法によるアルゴリズムを実装するときは式 1 をそのまま用いると，編集距離の大きさに応じて使用するデータ型も大きくしなければならなく，SIMD 演算による並列性が減少する．しかし，次に示す差分表現による式 2,3 を用いるとすべての要素を -1，0，$+1$ の 3 通りで表現でき，必要なデータはすべて 2 ビットで抑えられる．

$$l_{ij} = \min(1, l_{i,j-1} - t_{i,j-1} + 1, -t_{i,j-1} + c_{ij}) \quad (2)$$
$$t_{ij} = \min(1, t_{i-1,j} - l_{i-1,j} + 1, -l_{i-1,j} + c_{ij}) \quad (3)$$

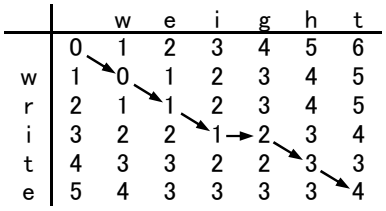

図 1: 動的計画法による編集距離計算．矢印は最短パス．

表 1: データ型と並列度，ステップ数，高速化率の関係．

データ型	bit	char	int
並列度	128	16	4
ステップ数	26	7	7
高速化率	8.62	4	1

ここで l_{ij}, t_{ij} は行列要素の差で d_{ij} を用いて次のように表わされる．

$$l_{ij} = d_{i+1,j} - d_{ij} \quad (4)$$

$$t_{ij} = d_{i,j+1} - d_{ij} \quad (5)$$

編集距離を出力するときは左上の要素から右下の要素に至るまでの任意の経路の差分を足し合わせることで編集距離を求められる．

データが小さくなったことで，SIMD 命令における並列処理数の増加や，データ転送時間の短縮が見込める．このとき，使用するデータ型によって計算の並列度と必要ステップ数にはトレードオフの関係が発生する．整数型を使用すると強力な整数演算により必要ステップ数は少なくなるが，サポートするデータ型は少なくとも 8 ビット以上のため並列性は高くならない．逆にビット単位の論理演算を使用すると簡単な演算しかできないために必要ステップ数は多くなるが，並列性は非常に高くなる．表 1 にデータ型と並列度，必要ステップ数の関係を示す．編集距離の計算においては，ビット単位の論理演算で実装すると必要ステップ数の増加を補って余りある並列度により高速計算が可能である．論理演算による実装法は次の論理関数化の節で述べる．

2.2 論理関数化

編集距離の計算を論理演算で実装するための手順を述べる．この手順は確実に最適な命令列を得られるとは限らないが自動化可能であり，他の計算プログラムにも適用可能な汎用的手法である．

表 2: 差に対応するビット表現．

差	s	0	$+1$	-1
ビット表現	$s_1 s_0$	00	01	$1*$

表 3: 組み合わせ表．$*$ はドントケアである．

l_1	l_0	t_1	t_0	c	d_1	d_0	r_1	r_0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0	1
0	0	0	1	0	0	0	1	$*$
0	0	0	1	1	0	1	0	0
0	0	1	$*$	$*$	0	0	0	1
0	1	0	0	0	1	$*$	0	0
0	1	0	0	1	0	0	0	1
0	1	0	1	0	1	$*$	1	$*$
0	1	0	1	1	0	0	0	0
0	1	1	$*$	$*$	1	$*$	0	1
1	$*$	0	0	$*$	0	1	0	0
1	$*$	0	1	$*$	0	1	1	$*$
1	$*$	1	$*$	$*$	0	1	0	1

計算を論理演算に分解する方法は論理回路の設計と同様の方法でできる．まず，入力すべての組み合わせに対して出力を求める．入力は差が 2 つとコストであり，組み合わせは 18 通りになる．差を -1，0，$+1$ で表したままでは 1 ビット単位の論理演算に分解することはできない．そこで差はすべて 1 ビットのデータ 2 つで表すことにする．差とビット表現の対応を表 2 に示す．表 2 に従って差を変換すると表 3 に示すような論理関数の組み合わせ表が完成する．表の l は左側の差，t は上側の差，c は置換コスト，d は下側の差，r は右側の差である．

次に組み合わせ表の通りに動作する関数を論理演算により合成する．今回は Quine-McCluskey 法 [1] により合成し，次の式を得た．

$$\begin{aligned} x &= \bar{t_1} c \\ y &= \bar{l_1} c \\ d_1 &= \bar{l_1} l_0 \bar{x} \\ d_0 &= l_1 \vee \bar{l_0} x \\ r_1 &= \bar{t_1} t_0 \bar{y} \\ r_0 &= t_1 \vee \bar{t_0} y \end{aligned}$$

SPU 命令セットには 2 項論理演算がすべて含まれている [2] ので論理関数の実現に必要な命令数は 10 である．なお，Quine-McCluskey 法は加法標準形での最簡形を求めるものであるから，SPU 上で最適な命令列が得られるとは限らないが，今回の編集距離計算の場合は整数演算を使う場合より速くなる良い結果となった．

1 回の行列要素の計算に必要な命令数は 52 で，even 命令と odd 命令が共に 26 ずつであるから 2 命令同時発行により 26 サイクルで計算できる．合計命令数が論理演算命令数に比べて非常に多くなったのは文字を比較しビット表現に変換するために多くの命令が必要だったからである．論理演算で関数を合成したことで並列性が高くなったため，128 行分のループアンローリングのみでパイプラインハザードの完全な除去に成功した．

2.3 枝刈り

行列要素の中には計算不要なものや自動的に値が定まるものがある．これらを発見し計算を省略することで高速化可能である．

計算不要な要素は最短パスに含まれないことが確実なものである．最短パスに含まれない要素は計算した行列要素から伸ばす最長見込みと最短見込みの距離を互いに比較することで発見できる．最長見込みは未計算の部分を全文字不一致として，最短見込みは未計算の部分を全文字一致としたときの右下の行列要素の値である．ある要素 x の最短見込み距離が別の要素 y の最長見込み距離よりも長いとき，x を通るパスはいかなる場合にも y を通るパスよりも距離が長い．よって x を通るパスはそれ以降計算不要である．また，隣り合う要素の距離の差は 1 以内であるから x の外側の要素の最短見込み距離も x のそれ以上になり計算不要となる．この枝刈りにより，ほとんどの問題において 40〜50 %の領域の計算を省略できる．

自動的に値が定まる要素とは，値域の中で最小の値を取った要素とその外側の要素である．自動的に値が定まる理由は行列の 0 行目，0 列目の初期化と同じ理由による．すなわち，その要素計算で参照する文字までで文字列のどちらかがもう一方の部分文字列であることが明らかとなったことで残りの文字はすべて削除するかすべて挿入するしかない．よって，計算するまでもなく外側の要素の値は 1 ずつ増加する．この枝刈りは特に文字列が似ているときに効果を発揮し，ほぼ 100 %の領域の計算を省略できることもある．

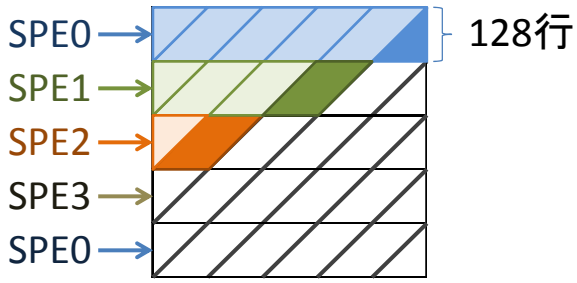

図 2: wave front 処理による並列化．

編集距離は差分表現になっているため，枝刈りのためには差を足し合わせて行列要素の値を求める必要がある．要素の値を求めるたびに左上の要素から差を足し合わせるのは非効率的であるから，実装上は 128 行 128 列ごとにその要素の値を計算し記憶することにした．枝刈りの判定に使う距離はこれらの要素からのみ求めている．すべての要素について枝刈りの判定を行うとすると差分表現から距離を求める計算が無視できないほどになるためである．

各 SPE は割り当てられた行の中に枝刈り対象の部分があれば計算を飛ばす．枝刈りされる領域は必ず行列の外側に存在するため，行ごとに計算する領域の左右の境界を設定することで実装できる．計算の入力として枝刈りした部分を参照するときは，枝刈りの理由を問わずすべて差分を+1 とみなす．こうすることで自動的に値が定まった部分に関しては正しい入力となり，最短パスに入らないために枝刈りされた部分であれば編集距離の特徴により無視される．

2.4 並列化

動的計画法による編集距離の計算においては wave front 処理による並列化が有効である（図 2）．各 SPE には 128 行ずつ順番に領域を割り当てる．SPE は 128 × 128 のブロック単位で計算し，終わるたびに境界要素のデータをメモリに書き込み，隣の SPE にシグナル通知レジスタ [3] を利用してシグナルを送る．このときのブロックは同時実行できる要素が斜め方向に並ぶので，それに合わせて変形させた．

各 SPE の担当する領域はあらかじめ固定される静的なスケジューリングである．この方法を採用した理由は，担当領域が固定されることで計算に必要とするデータの位置が事前に決まるため，ダブルバッファリングなどの通信

の最適化が容易となるからである．また，データの再利用するタイミングも固定のため，ローカルストアにデータを保持することでメモリ使用量を減少させる効果もある．

2.5 通信の最適化

計算の実行時間が減少すると相対的に通信にかかる時間の割合が大きくなる．SPE プログラム最適化の最終段階においては通信時間の隠蔽，短縮が性能向上に大きく貢献する．編集距離計算プログラムにおいては幸いなことに通信時間によって性能が大きく減少することはない．行列要素を差で表現したことで 1 回の SPE 間通信で送る行列要素のデータ量はわずか 256 ビットに収まるからである．そのため，通信時間を短縮する方法は採用せず，隠蔽するダブルバッファリングのみを実装した．

行列データの転送はメモリを介さずにそれぞれの SPE のローカルストア内で行うと通信の高速化が期待できる．しかし，容量の少ないローカルストア上で実装するのは多少の困難が伴う．そのため今回は簡単に実装できるメモリを介したデータ転送を行った．SPE 間の直接転送での実装より遅くなるが，計算時間と比べて通信時間は非常に短いのでダブルバッファリングによって完全に隠蔽できた．

枝刈りの情報はローカルストア間で直接転送を行った．枝刈りには直前に計算した行列要素のデータが必要なためダブルバッファリングの実装が困難であるから，最も遅延の短くなるローカルストア間の直接転送を実装した．

2.6 PPE による前処理

2 つの文字列の前後の部分が完全に一致するような問題は枝刈りの条件分岐の都合上実装が難しい．そこで，この部分に限り PPE にあらかじめ計算させることにした．128 文字ずつ両方の文字列が一致しているかどうかを前後から調べ，一致しなくなるまで両方の文字列から一致した文字を取り除く．取り除いた部分は距離が 0 なので除去後はそのまま SPE に問題を解かせればよい．文字の除去にかかる計算時間はほとんど無いに等しく，特定の問題においては劇的な計算性能の向上が見られた．

3 結果

決勝問題 10 問に対する上位 3 チームの実行時間を図 3 に示す．自明でない問題と簡単に近似解を得られない問題 1,6,7 にて 1 位を記録した．枝刈りなしで問題を解くと問題 1,2 は 120 ミリ秒程度，その他の問題は 200 ミリ秒程度の実行時間になるので，枝刈りによって半分以上の計算を省略できている．

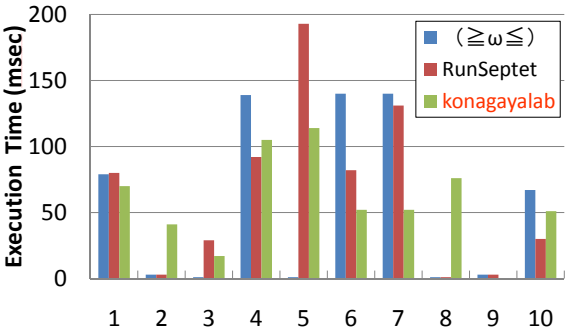

図 3: 決勝問題に対する上位 3 チームの実行時間．

4 まとめ

単純な計算を論理関数で表現し，128 並列の SIMD 論理演算を利用することで編集距離の計算を高速化した．本手法は汎用的な高速化技法であり，様々な計算アルゴリズムを論理演算で実装し Cell/B.E. 上で効率的に実行することが期待できる．

参考文献

[1] McCluskey, E. J.: Minimization of Boolean Functions. Bell Syst. Tech. J. , Vol. 35, No. 5, pp. 1417–1444(1956).

[2] Sony Computer Entertainment Inc. Synergistic Processor Unit 命令セット・アーキテクチャ Version 1.2, 2007

[3] Sony Computer Entertainment Inc. Cell BroadBand Engine^{TM} アーキテクチャ Version 1.01, 2006.