Multicore Programming Contest
Cell Challenge 2009
ツールキット解説書
ver.0.1対応版

Cell Challenge 2009
実行委員会

先進的計算基盤システムシンポジウムSACSIS2009併設企画
Multicore Programming Contest
Cell Challenge 2009

これはなに？

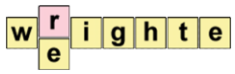
- Cell Challenge 2009規定課題に取り組む際のベースとなるツールキットの解説です
- 実際に公開されるツールキットは仕様が変更される場合があります

規定課題
「文字列の編集距離計算」

- 2つの文字列の近さを計る方法
  - かな漢字変換エンジンや，DNAの相同性検索などに利用される
- 2つの文字列の 編集距離
  - 片方の文字列から，もう一方の文字列を得るための操作回数の最小値
  - 使用可能な操作は以下の3種類
    - 削除：1つの文字を取り除く
    - 挿入：1つの文字を新たに付け加える
    - 置換：1つの文字を別の文字で置き換える
  - 参考になるURI「wikipedia:レーベンシュタイン距離」
    http://ja.wikipedia.org/wiki/%E3%83%AC%E3%83%BC%E3%83%99%E3%83%B3%E3%82%B7%E3%83%A5%E3%82%BF%E3%82%A4%E3%83%B3%E8%B7%9D%E9%9B%A2

編集距離の操作例

- 「weight」と「write」の編集距離
- 以下の操作で「weight」から「write」になる
  - ➢ weight
  1. weighte (挿入：e)
  2. wrighte (置換：e → r)
  3. wrihte (削除：g)
  4. write (削除：h)



- 3回以下の操作では「weight」を「write」できない
- よって編集距離は「4」
- 削除，挿入，置換は文字列の中のどの位置で行ってもよい

編集距離計算のアルゴリズム(概要)

- 「動的計画法(Dynamic Programming)」が有名

  文字列1　weight
  文字列2　write

- 操作回数を表を作って求める

編集距離計算のアルゴリズム(手順)

このスライドには
アニメーションが設定されています

		1	2	3	4	5	N=6
	0	w	e	i	g	h	t
1	w	0	1	1	1	1	1
2	r	1	1	2	2	2	2
3	i	1	2	1	2	3	3
4	t	1	2	2	2	3	3
5	e	1	2	3	3	3	4
M							

アニメーションでは列ごとに計算しているが，「上，左上，左のセル値」が決定しているセルならば計算手順は任意

1. 最左列，最上行のセルは1～N，1～Mの値と仮定
2. 上，左上，左のセル値と，当該の行と列の文字列を使い，以下の方法で各セルの値を計算する
   - A) 文字列が同一ならば左上セルの値
   - B) 異なるならば左上セルの値+1(置換)
   - C) 左セルの値+1(挿入)
   - D) 上セルの値+1(削除)
   - E) (A)-(E)の最小値をセルの値に設定
3. テーブルの左上から値を求める
4. 右下端のセル値が最終的な編集距離となる

アルゴリズムのブロック化

このスライドには
アニメーションが設定されています

	0	w	e	i	g	h	t
0							
w		0	1	1	1	1	1
r		1	1	2	2	2	2
i		1	2	1	2	3	3
t		1	2	2	2	3	3
e		1	2	3	3	3	4

Block1 Block3
Block2 Block4

- 編集距離計算はブロック化して部分ブロックごとに計算を行う
- 以下のデータがあれば，各ブロックの計算が可能
  - 文字列
  - 最左列，最上列の値
- ブロック計算の入力は以下
  - 部分文字列(各問題文字列中の該当部分)
  - 最左列，最上列のセル値
- 出力は以下
  - 最右列，最下行のセル値
- ツールキットでは，各ブロックの最右列，最下行の値をメモリのユーザ領域に格納，適宜読み出して使用する

ツールキット ver0.1

- 2つのテキストファイルを入力すると，それらの中の文字列を読み込んで編集距離を求める
- 制約：各文字列の文字数は128の倍数
  - いろいろなサイズの例題ファイル付き
    (file1, file2, … , file12)
- getrndstr.c を使用すると，任意長のランダム文字列を生成できる

```
$ gcc -O3 -o getrndstr getrndstr.c
$ ./getrndstr 128 13 > file9999
```

乱数種13で生成される128文字の文字列を格納したファイルfile9999を生成する

実行方法(1/2)

- Makefileの「USERNAME」を各ユーザ名に編集

```
USERNAME = USERNAME
IDIR     = /export/home/$(USERNAME)/toolkit0.1
EXE_FILE = /export/home/$(USERNAME)/toolkit0.1/main
OBJS     = main spe1
```

- コンパイル

```
$ make
/opt/cell/toolchain/bin/ppu-gcc -O0 -Wall -m32 -lspe2 main_ppe.c -o main lib.c
/opt/cell/toolchain/bin/spu-gcc -O3 -Wall  main_spe.c spe1.c -o spe1
```

実行方法(2/2)

```
[       @dev toolkit0.1]$ make run4
cellexec -t 30 /export/home/yoshimi/toolkit0.1/main  /export/home/yoshimi/toolki
t0.1/file9 /export/home/yoshimi/toolkit0.1/file10
execute command: /export/home/yoshimi/toolkit0.1/main /export/home/yoshimi/toolk
it0.1/file9 /export/home/yoshimi/toolkit0.1/file10 timeout 30
strnum(a)= 128
strnum(b)= 256
str1 : 128 : blk 1
str2 : 256 : blk 2
[SPE_] : 224
[PPE_] : 224
eclock  : 0.00609088
dec_cnt : 0.00140768
SUCCESSFUL.

execute time:       2083452 usec
```

実行バイナリ，第1ファイル，第2ファイル
第1ファイル，第2ファイルの文字列長
各文字列のブロックの数
SPE,PPEそれぞれの計算結果
SPE,PPEそれぞれの計算時間
(後期バージョンで修正されます)

巨大な文字列長を対象にした場合，計算時間が非常に長くなることがあります

Regulation : メモリ領域の割当

- PPEからSPEに渡される変数について(先頭アドレス)
- 各文字列が配置されるメインメモリ上の領域
  - buf1, buf2　それぞれ20MB
- ユーザが自由に使用できるユーザ領域
  - buf3　　20MB
- 距離計算結果格納領域(buf4[0]に回答を格納すること)
  - buf4　　128B
- 各文字列の長さnum1, num2

参加者は，buf1～buf4までのメモリを自由に使用できます

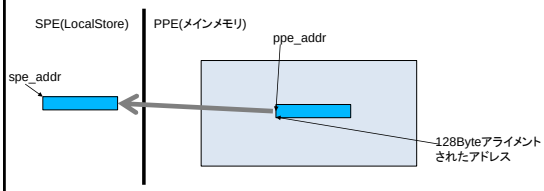
ツールキットver0.1の計算手順

- SPE1個のみ用いて計算する(PPEのプログラムは未変更)
- 各文字列に対して，128文字を1ブロックとして計算を行う(各文字はchar型)
  - ブロックの計算前に，以下をSPEのローカルストアに読み込む
    - ブロックの再左上セルの値(tlm : top-left-most)
    - ブロックの最左列(vbuf)，最上行(hbuf)を読み込む
    - 部分文字列
  - 各SPEが処理する表は，1ブロックあたり128x128x4B=64KB
  - 計算後，表の最右列(vbuf)，最下行(hbuf)を書き出す
  - 回答(最終ブロック最右下セルの値)をメインメモリ上のbuf4[0]に書き込む

プログラムコードにもコメントがあります

補足)DMA転送用サブルーチン

- 転送用の命令
  - dmaget, dmaput : ツールキットで使用されるDMA読み書き関数
  - dmaget((void*))spe_addr, ppe_addr, X);
    メインメモリ領域のアドレスppe_addrを先頭としてXByteを読み出し，LocalStoreのアドレスspe_addrから始まる領域に格納する．dmaputは逆方向．



おわり
皆様のご参加をお待ちしております