

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
w	0	1	2	3	4	5
r	1	1	2	3	4	5
i	2	2	1	2	3	4
t	3	3	2	2	3	3
e	4	4	3	3	3	4



- 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
OBJ_S     = main spel
```

- 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 spel.c -o spel
```

実行方法(2/2)

```
[ @dev toolkit0.1 ]$ make run4
cellexec -t 30 /export/home/yoshimi/toolkit0.1/main /export/home/yoshimi/toolkit0.1/file9 /export/home/yoshimi/toolkit0.1/file10
execute command: /export/home/yoshimi/toolkit0.1/main /export/home/yoshimi/toolkit0.1/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 : 0.006090088
descnt : 0.00140768
SUCCESSFUL.
execute time: 2083452 usec
```

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

ツールキットver0.5の計算手順

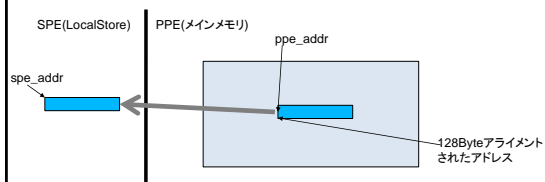
- ツールキットver0.1に加え, SPEの処理を開始した後に, 参加者がPPEで処理できるようコードを修正 (ppe.cのppe_user()関数で処理が可能)
- 参加者が変更可能(不可能)なファイルの明確化
 - 修正不可なファイルはmain_ppe.c, define.hのみ

規定課題への取り組み方

- 規定課題では, SPEのプログラム(spe1.c)を実装する
 - 以下のプログラムを実装してください
 - 編集距離を計算するPPE, SPEプログラム(ppe.c, spe1.cなど)
 - Makefile (SPEごとに別のパイナリを実行したい場合は修正が必要です)
 - 以下のファイルは変更が許可されません
 - PPEプログラム(main_ppe.c, define.h)

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する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