HW 2: Two-way Min-cut Partitioning

105062600 Yi-Cheng, Chao 05:06, April 9, 2017

Homework Concepts

使用 C++實作完成 Fiduccia-Mattheyses Partition Algorithm。

Compile & Execute

How to compile:

Go to src/ directory and type "make main" command

[vlsipda14@ic29 src] make main

How to execute:

Go to src/ directory and type the following command

[vlsipda14@ic29 src] ./main –c <cell_file> -n <net_file> -o <output_file>

Option :

-d: show the detail summary

-h: show the usage of executable binary

-t : show the time report

-s: engage pin number sort for initial partition

```
[vlsipda14@ic29 src]$ ./main -h
Usage: ./main -c <cells_file_name> -n <nets_file_name> -o <output_file_name>
        -d: Show detail message [Option]
        -h: Show the command usage [Option]
        -t: Show the time report [Option]
        -s: Engage pins number sort for initial partition [Option]
```

Time Measurement

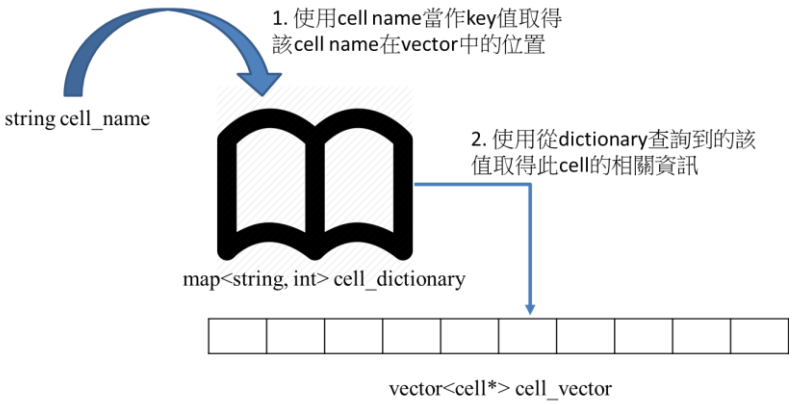
指令使用<sys/time.h> library 的 gettimeofday 函數來取得時間，精度可達微秒。

	p2-1	p2-2	p2-3
I/O time (sec)	0.0033	0.0486	0.9872
Computing time (sec)	0.0087	0.2067	77.5923
Execution time (sec)	0.0120	0.2553	78.5795

主要的 bottleneck 都在 FM algorithm 的 computation，I/O time 的讀寫檔案上並沒有花費太多時間，由於演算法的執行上沒有到很精煉導致複雜度不優，因此在測資比較大的情況下執行時間就急遽膨脹，推測對於 p2-4 隱藏測資可能會需要到半小時以上。

Design Concept

Nets 與 Cells 資料儲存使用兩階架構，第一階叫做 dictionary，使用 map<string, int>以該 net/cell name 為 key 值索引該 net/cell 在第二階 vector 中的位置，存取 cell/net 步驟如下圖所示：



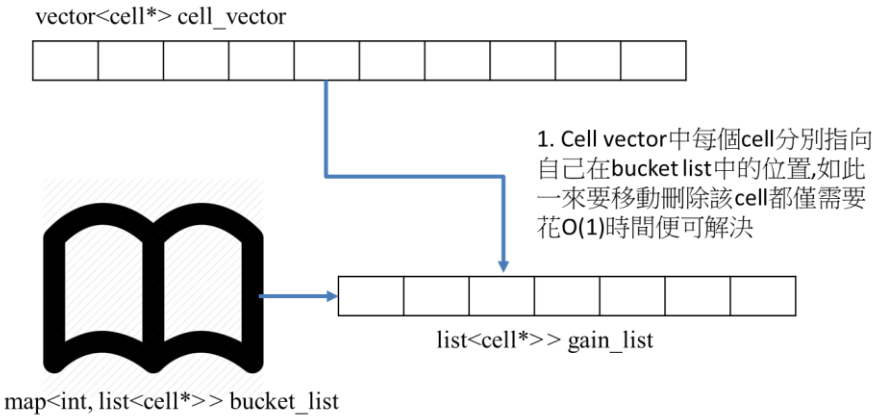
Vector 內儲存該 cell/net 的一切資訊，cell/net 中的成員如下:

```
struct cell{
    string name;//cell name
    int size;//cell size
    int gain;//cell gain
    int pin;//cell pin
    vector<int> net_list;
    bool state;//0 for free; 1 for lock;
    bool partition;//0 for partition A; 1 for partition B
    list<cell*>::iterator ptr;//point to the position in bucket list
};

struct net{
    string name;//net name
    int A, B;//cell num in A and B respectively
    vector<int> cell_list;

};
```

而 Bucket list 則使用 map<int, list<cell*> >資料結構，key 值範圍為+max pin ~ -max pin，並且在建立 bucket list 同時任何 cell 都會有一個 list<cell*>::iterator 型別指標指向自己在該 bucket list 中的位置。



此外使用一個叫做 partition_looper 的 function 來控制是否要做下一次 iteration 的 FM partition，在我的 FM partition function 會回傳 largest partial sum，若 largest partial sum 為 0 則跳出迴圈，進行輸出寫檔。

Bucket list 中的相關動作以及 initial partition 的分法會於 Q/A 時再詳盡分析。

Experiment Analysis & Question/Answer

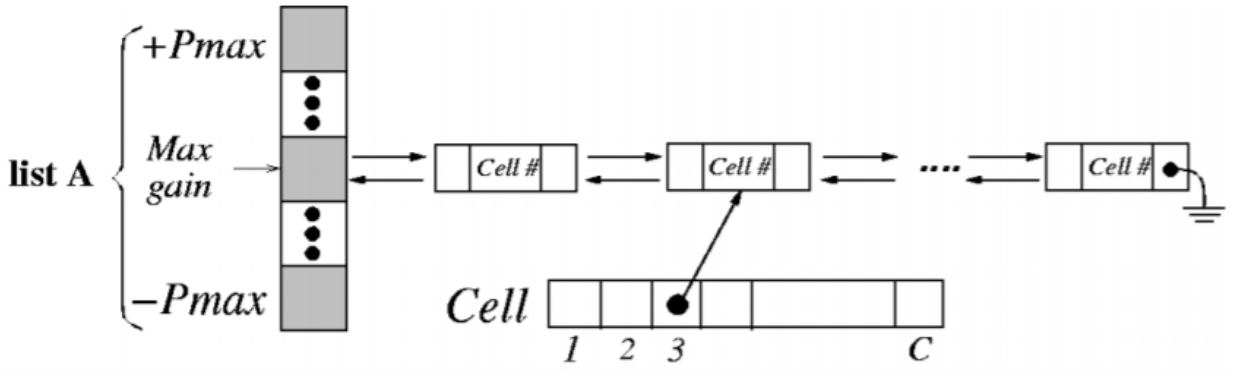
Q1: Where is the difference between your algorithm and FM Algorithm described in class? Are they exactly the same?

資料結構的使用上以及 tie breaker rule 可能與原作者在使用上有所不同，但主要的演算法都是根據上課投影片所進行實作，比較特別處為在 Update gain 時會我先在所有動作執行前使用 array 儲存與該 cell 相連之其他 cell 的 gain 值，最後所有 Update gain 結束在根據此 gain 值索引 bucket list 來移動其他 free cell 到新 gain list。

Q2: Did you implement the bucket list data structure? If so, is it exactly the same as that described in the slide? How many are they?

我使用兩個Bucket List分別代表partition A與partition B。

講義Bucket List示意圖如下：



Bucket List資料結構使用map<int, list<cell*> >，key值為gain，data值為串接每個cell的list結構，STL list本身為double link list，在cell vector中記錄原本的每個cell，並且cell vector中的每個cell皆有list<cell*>iterator指向自己在bucket list中的位置，然而講義從圖上理解為2D-double link list結構。

與講義Operation時間複雜度比較:

	In Slides	My program
Find a cell with max gain	O(1)	O(plog(p))

p = #max pin;

從max pin開始，如果list.empty() == true則往下找，若不為空則輸出第一個cell。

map中找出p為O(log(p))，最多全部map找過一輪為O(p)，輸出list中第一個cell為O(1)。

	In Slides	My program
Remove cell from structure	O(1)	O(log(p))

p = #max pin;

根據gain值索引出map中哪條list為O(log(p))，直接從ptr取得list中位置後erase()為O(1)。

	In Slides	My program
Insert cell into structure	O(1)	O(log(p))

p = #max pin;

根據gain值索引出map中哪條list為O(log(p))，push_back into list為O(1)，從cell vector中的的該cell指標指向該list中的cell為O(1)。

	In Slides	My program
Update g(i) to g(i)+Δ	O(1)	O(nc)

c = #cell list size for one net; n = #net list size for one cell

對於搬動一個cell，從該cell連接的每條net開始去更動該條net相連接cell gain，對於任何一個cell完整update完需要全部走過一輪，因此為O(nc)。

	In Slides	My program
Update max gain pointer	O(1)	X

我使用find a cell with max gain取代該動作，因此沒有update max gain pointer的動作。

Q3: How did you find the maximum partial sum and restore the result?

以下用LPS(Largest Partial Sum)代表maximum partial sum。

對於每次FM algorithm執行的iteration，Update gain動作若有發生則會push進一個stack容器，若與原partition sum相比為大於或等於則儲存該lps_iteration並更新maximum partial sum，最後看執行完的iteration和lps_iteration相差多少，代表要pop多少次cell來進行還原partition。(Ex: 執行10次，LPS發生在第5次，則要pop out出5個cell來進行partition的還原)

Q4: Please compare your results with the top 3 students' results from last year and show your advantage either in runtime or in solution quality. Are your results better than them?

	p2-1		p2-2		p2-3	
	Final cut size	Run time(sec)	Final cut size	Run time(sec)	Final cut size	Run time(sec)
1st	6	0.01	221	0.08	1630	2.95
2nd	9	0.01	210	0.04	4339	1.99
3rd	6	0.14	453	0.17	1676	1.85
My program	6	0.01	184	0.26	2788	78.6

在小測資的條件下我的程式都有跑出不錯的結果，在p2-2甚至比去年前3名都還能得到更好的cutsize，然而隨著測資上升我的執行時間會急遽上升，推測可能是和演算法的執行上有關(2nd也在p2-4 case下的執行時間也急遽上升)，然而去年的前三程式無論執行時間或品質都能維持高水準實在令人讚嘆。

Q5: What else did you do to enhance your solution quality or to speed up your program?

原本打算使用pin number作為排序，使pin多的盡量在其中一側，而pin少的盡量在另一側，然後使用list來pop_front給partition A，pop_back給partition B，但意外中發現如果依照資料輸入順序前後直接分群的話，居然可以一開始就達到不錯的cut size，因為FM algorithm是heuristic也和一開始的資料分群有關，因此有了不錯的結果。

	p2-1			p2-2			p2-3		
	Initial cut size	Final cut size	FM iteration	Initial cut size	Final cut size	FM iteration	Initial cut size	Final cut size	FM iteration
A	228	18	5	3121	258	8	55378	5459	14
B	91	6	3	773	184	3	30935	2788	10

A: initial partition with pin number sort; B: initial partition without pin number sort;

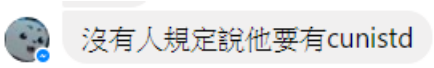
由於結果好的沒有理由，我推測是和使用ruby產生測資時cell file輸出排序方式有關，可能剛好猜對資料產生後打亂的方式，使得分群的結果可以很接近最佳解。

Q6: What have you learned from this homework? What problem(s) have you encountered in this homework?

這次的作業使用到了一堆資料結構，在複習資料結構的特性同時也很慶幸自己是c++ programer，有一堆強大的容器支援完成我想做的動作，如果全部都要自己從建立class開始架構起，可能便要花更多天來完成。

在這次的作業中很常遇到segmentation fault，因此在debug期間常常還是依賴printf大法來操作，若能有更詳盡的規劃來寫程式而不是try and error的直接一步一步構築起可能完成的會更輕鬆吧！

此外，居然只有<unistd.h>而沒有<cunistd>也是長了知識了!



Homework Review

這次作業使用到了很多資料結構像是 stack、list、map、vector 等等，熟悉更多 STL 與更多資料結構都將更能協助作業的完成以及 performance 的增加，希望自己對於資料結構能在更透徹了解，在 EDA 領域中資料結構的選擇往往大大的影響了時間複雜度，以及在演算法的運行上能透過更聰明的方式撰寫，如同去年前三名同學一樣能有不錯的結果，最後使用輕鬆一點的圖勉勵自己。