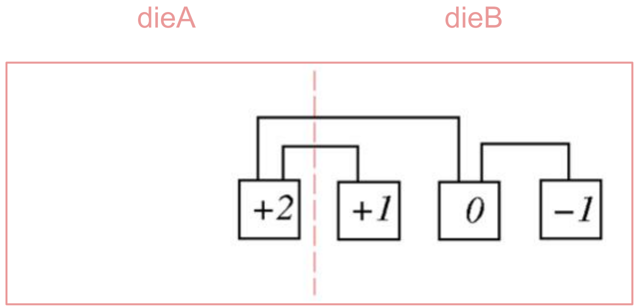
Parallel Programming

Final Project :
Two-way Min-cut Partitioning

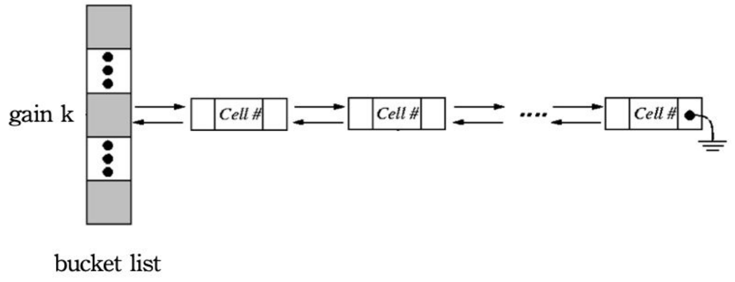
Two-way Min-cut Partitioning

Let C be a set of cells and N be a set of nets. Each net connects a subset of cells.

The two-way min-cut partitioning problem is to partition the cell set into two disjoint groups A and B, where each group of cells is put in a different die. The cost of a two-way partitioning is measured by the cut size(metric), which is the number of nets having cells in both groups.



Fiduccia-Mattheyses Algorithm



1. Select the cell with the highest gain value as the base cell.

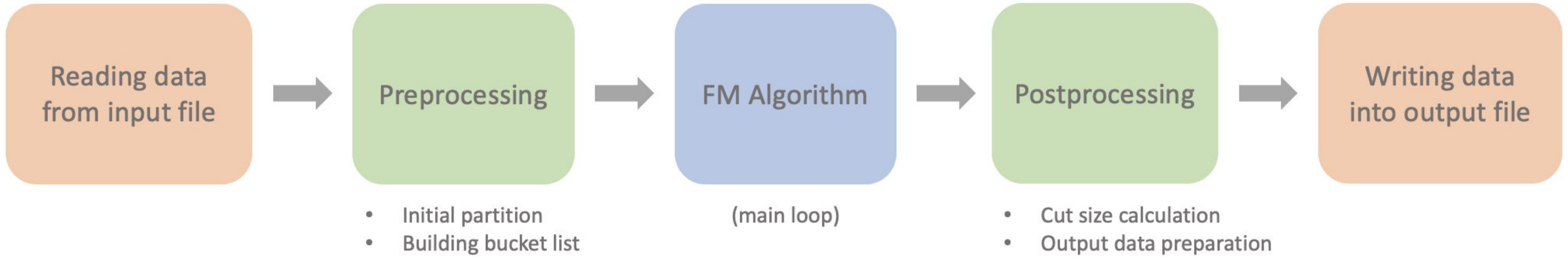
   If moving it satisfies the balance condition, then move and lock the cell.

   Otherwise, find the next base cell.

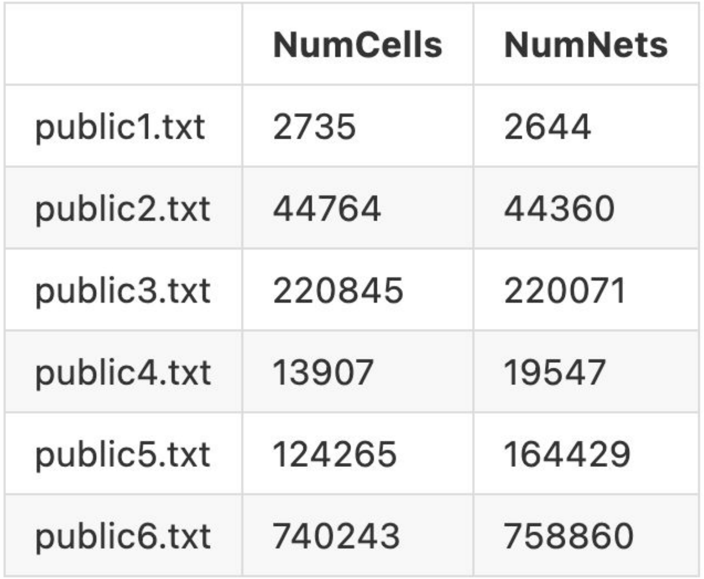
2. Repeat step 1 until there are no more cells that can serve as the base cell.
3. Calculate the maximum partial sum based on the gain values of the base cells.
4. If the maximum partial sum ≤ 0, then terminate.

   Otherwise, revert to the state at the time of the maximum partial sum and go back to step 1.

Process for solving Two-way Min-cut Partitioning



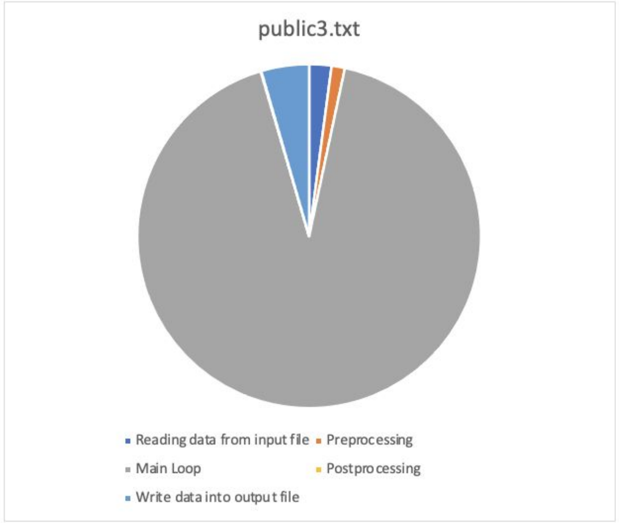
Execution results of the sequential version code

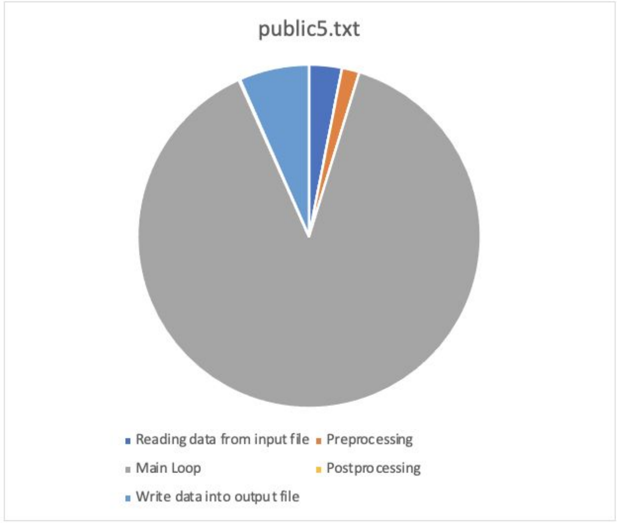


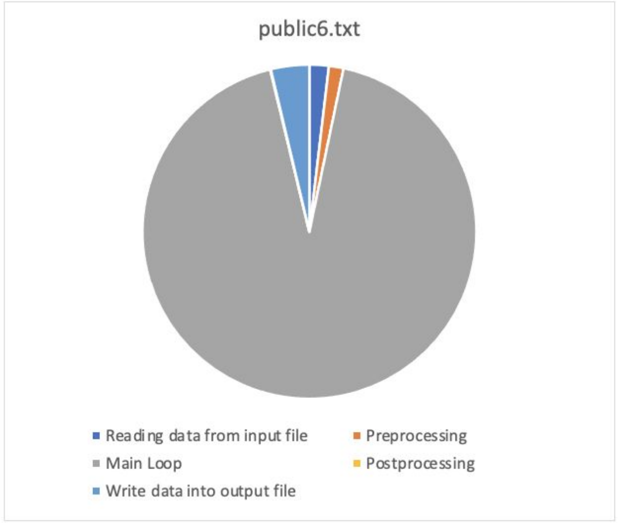
	NumCells	NumNets
public1.txt	2735	2644
public2.txt	44764	44360
public3.txt	220845	220071
public4.txt	13907	19547
public5.txt	124265	164429
public6.txt	740243	758860

Testcase	Cut size(before)	Time(before)
public1	204	0.047678 S
public2	3677	2.87102 S
public3	7103	74.5056 S
public4	1581	1.6877 S
public5	1667	23.8878 S
public6	10278	254.121 S

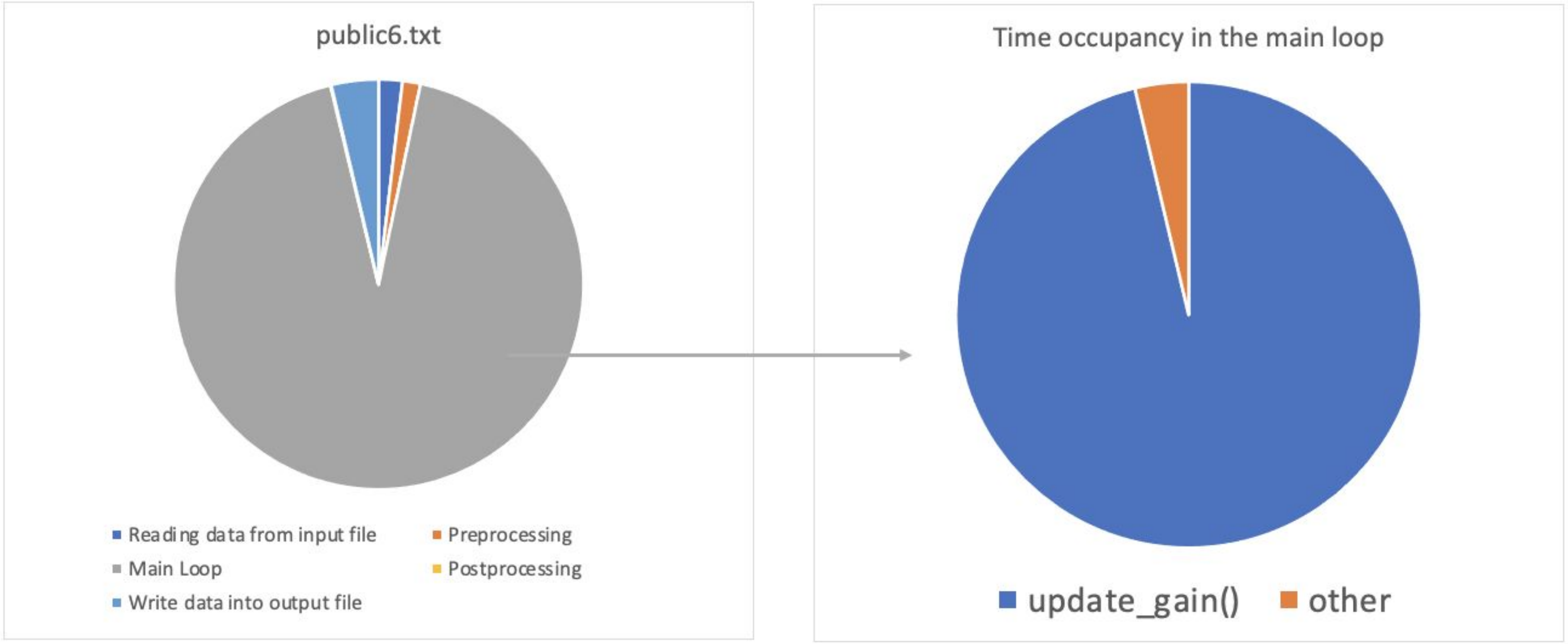
Time distribution of program execution







Bottleneck : update_gain() in main loop



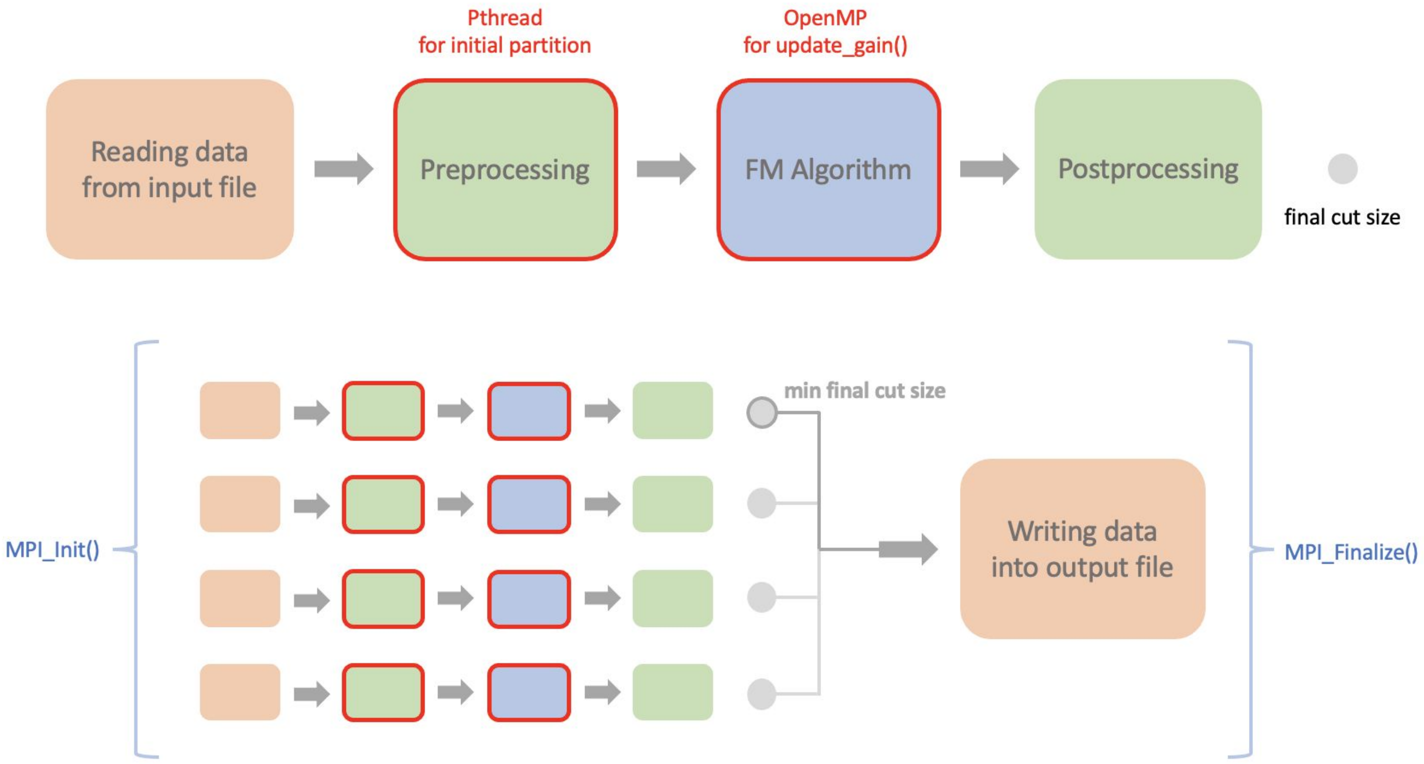
Directions for parallelization

- Due to the significant impact of the initial partitioning on the final cut size result, we use Pthread for parallel computation of multiple initial cut sizes and choose the configuration of the thread with the smallest initial cut size to execute the FM algorithm.

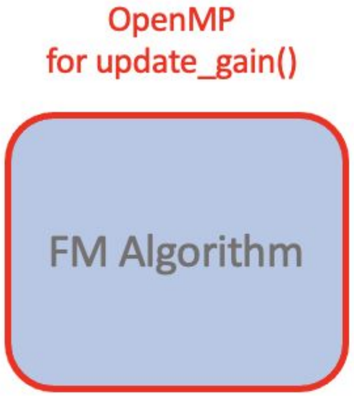
- The process of update_gain() is parallelized using OpenMP to reduce execution time.

- By using MPI, multiple processes run the entire code in parallel, each calculating the final cut size independently, and only the smallest final cut size is written into the output file.

Directions for parallelization



OpenMP optimization



What is Fn() & Tn() & Gn() ?

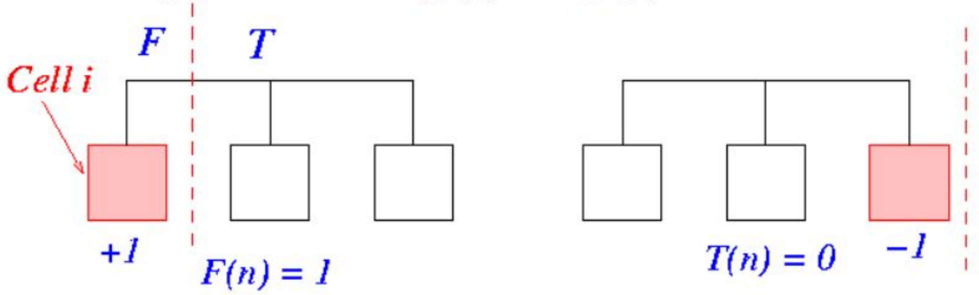
Fn(c,n) = # of cell in F

Tn(c,n) = # of cell in T

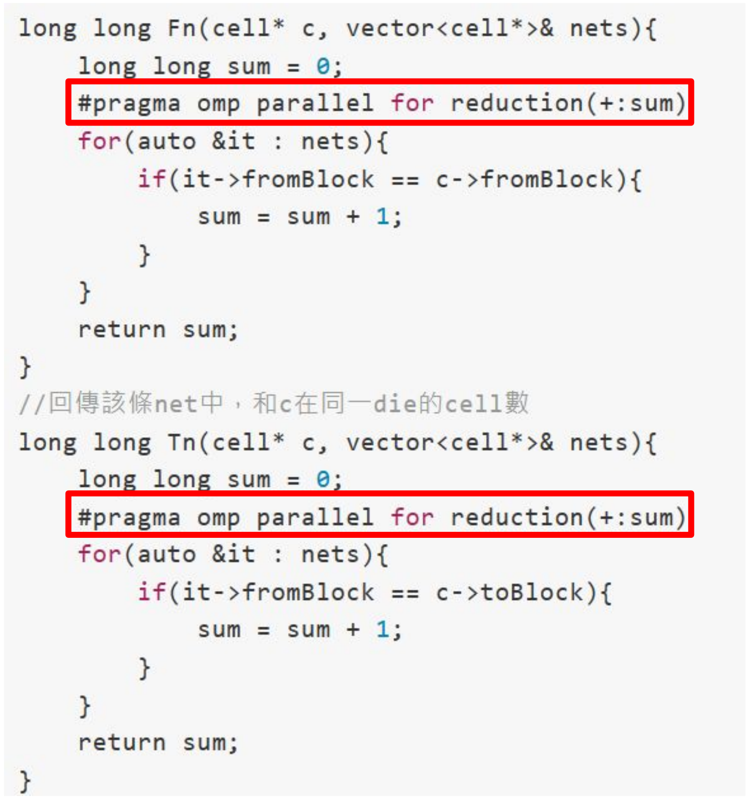
Gn(c,n) = gain of cell c

Initialization of all cell gains requires $O(P)$ time:

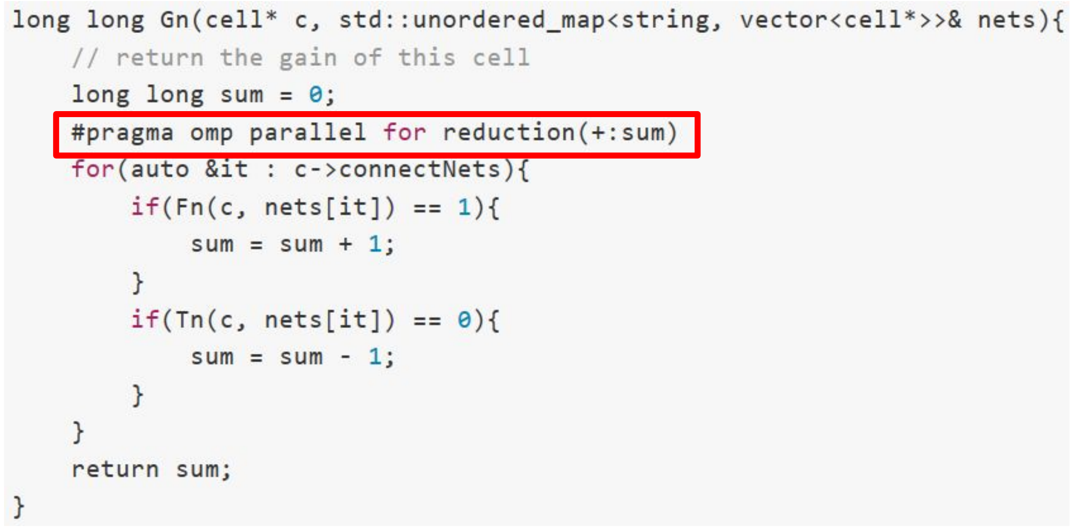
```
$g(i) \leftarrow 0$;
$\boxed{F \leftarrow \text{the "from block" of cell } i; \quad T \leftarrow \text{the " to block" of cell } i;}$
for each net $n$ on Cell $i$ do
    if $F(n) = 1$ then $g(i) \leftarrow g(i) + 1$;
    if $T(n) = 0$ then $g(i) \leftarrow g(i) - 1$;
```



OpenMP for Fn() & Tn() & Gn() - optimize



```cpp
long long Fn(cell* c, vector<cell*>& nets){
    long long sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for(auto &it : nets){
        if(it->fromBlock == c->fromBlock){
            sum = sum + 1;
        }
    }
    return sum;
}
//回傳該條net中，和c在同一die的cell數
long long Tn(cell* c, vector<cell*>& nets){
    long long sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for(auto &it : nets){
        if(it->fromBlock == c->toBlock){
            sum = sum + 1;
        }
    }
    return sum;
}
```



```cpp
long long Gn(cell* c, std::unordered_map<string, vector<cell*>>& nets){
    // return the gain of this cell
    long long sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for(auto &it : c->connectNets){
        if(Fn(c, nets[it]) == 1){
            sum = sum + 1;
        }
        if(Tn(c, nets[it]) == 0){
            sum = sum - 1;
        }
    }
    return sum;
}
```

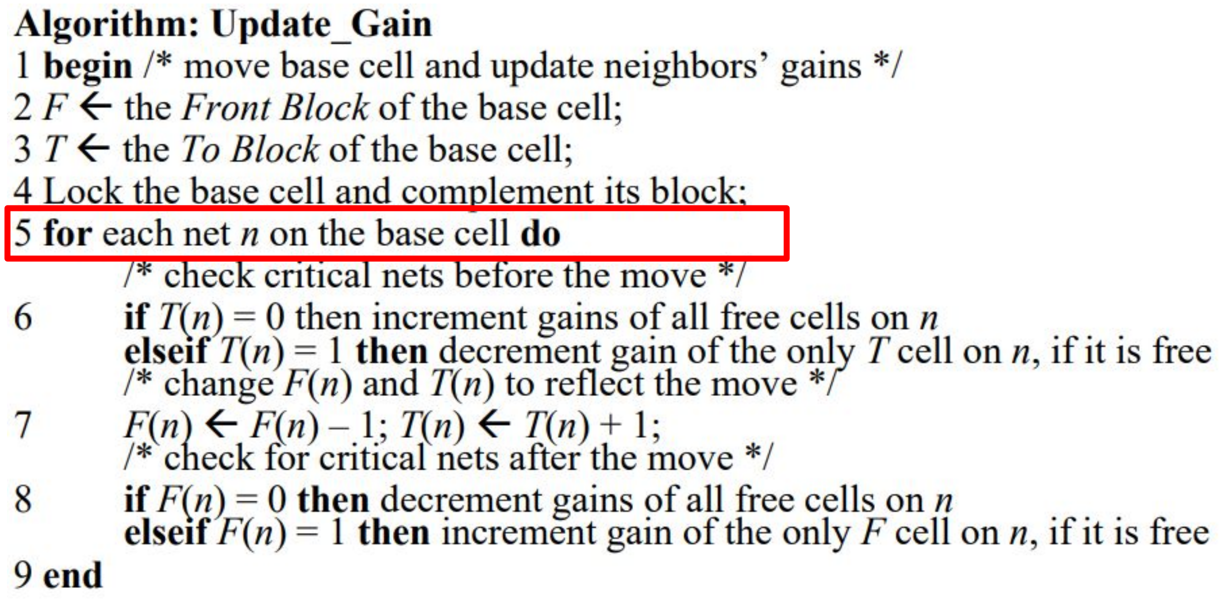
OpenMP for Fn() & Tn() & Gn() - result

設定 4 cores：srun -n1 -c4 ../bin/main ../testcase/public[i].txt ../output/public[i].out

Testcase	Cut size(before)	Time(before)	Cut size(after)	Time(after)
public1	204	0.047678 S	200	0.0924522 S
public2	3677	2.87102 S	3682	3.50781 S
public3	7103	74.5056 S	6910	55.741 S
public4	1581	1.6877 S	1791	1.80591 S
public5	1667	23.8878 S	1518	20.6604 S
public6	10278	254.121 S	11815	163.169 S

What is update_gain() ?

移動 base cell 後，需要更新和 base cell 在同一 net 上的其他 cell 的 gain 值



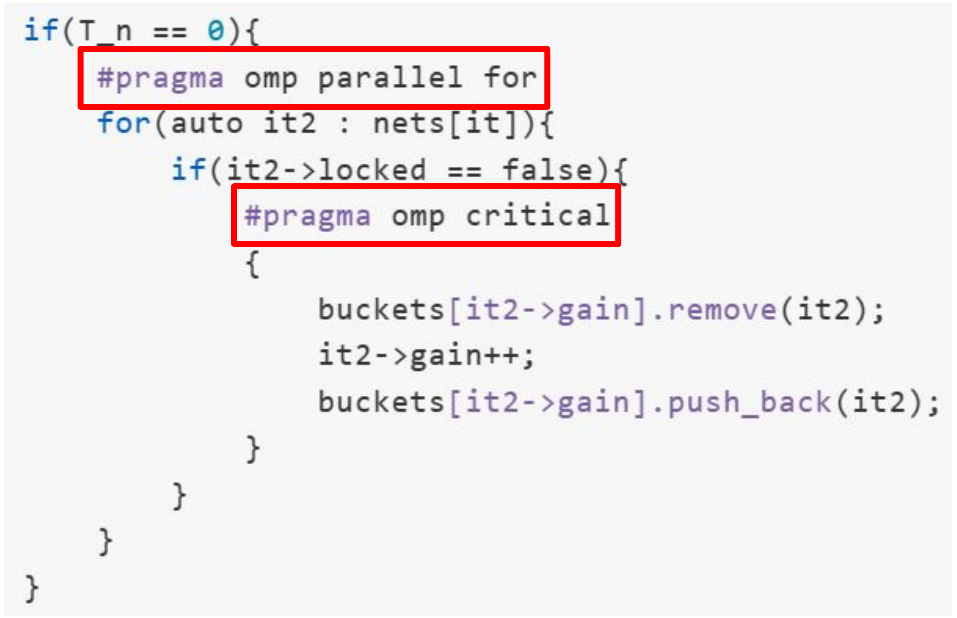
Algorithm: Update_Gain

```
1 begin /* move base cell and update neighbors' gains */
2 $F \leftarrow$ the Front Block of the base cell;
3 $T \leftarrow$ the To Block of the base cell;
4 Lock the base cell and complement its block;
5 for each net $n$ on the base cell do
        /* check critical nets before the move */
6       if $T(n) = 0$ then increment gains of all free cells on $n$
        elseif $T(n) = 1$ then decrement gain of the only $T$ cell on $n$, if it is free
        /* change $F(n)$ and $T(n)$ to reflect the move */
7       $F(n) \leftarrow F(n) - 1$; $T(n) \leftarrow T(n) + 1$;
        /* check for critical nets after the move */
8       if $F(n) = 0$ then decrement gains of all free cells on $n$
        elseif $F(n) = 1$ then increment gain of the only $F$ cell on $n$, if it is free
9 end
```

OpenMP for update_gain() - 針對和 base cell 相連的同一個 net 上的不同 cell

對 List 增加或刪除時，會修改容器的大小，需利用 **omp critical**

否則會造成 segmentation fault (list 的指標錯誤)



```
if(T_n == 0){
    #pragma omp parallel for
    for(auto it2 : nets[it]){
        if(it2->locked == false){
            #pragma omp critical
            {
                buckets[it2->gain].remove(it2);
                it2->gain++;
                buckets[it2->gain].push_back(it2);
            }
        }
    }
}
```

OpenMP for update_gain() - result

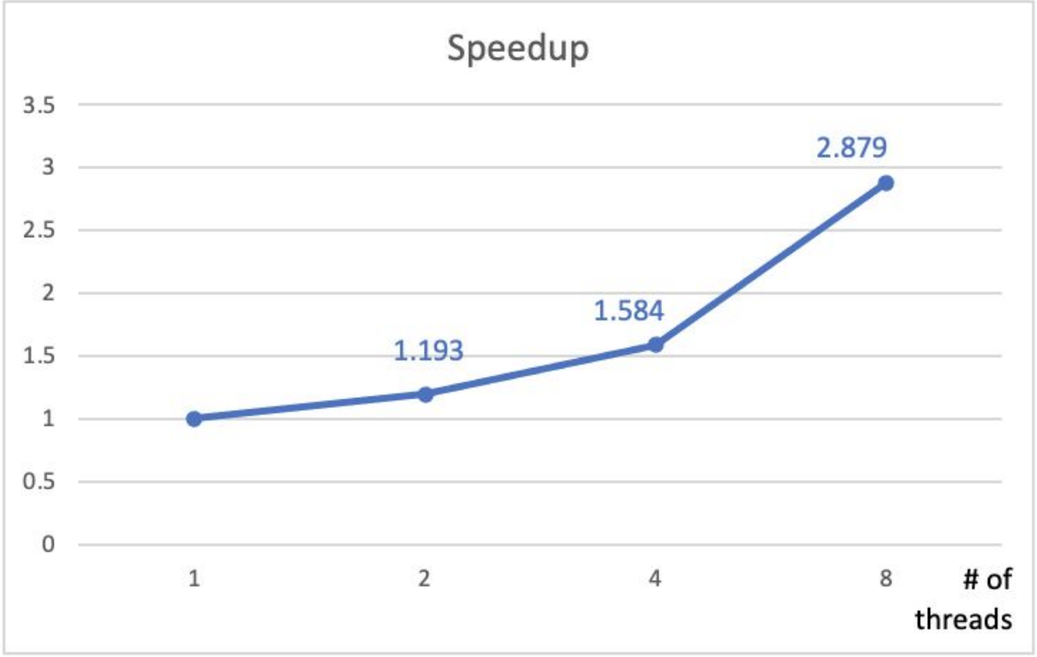
設定 4 cores：srun -n1 -c4 ../bin/main ../testcase/public[i].txt ../output/public[i].out

Testcase	Cut size(before)	Time(before)	Cut size(after)	Time(after)
public1	200	0.0924522 S	204	0.0814006 S
public2	3682	3.50781 S	3676	3.48651 S
public3	6910	55.741 S	7097	59.1856 S
public4	1791	1.80591 S	1504	2.02676 S
public5	1518	20.6604 S	1667	21.3453 S
public6	11815	163.169 S	10386	124.423 S

OpenMP for update_gain() - result

public6.txt

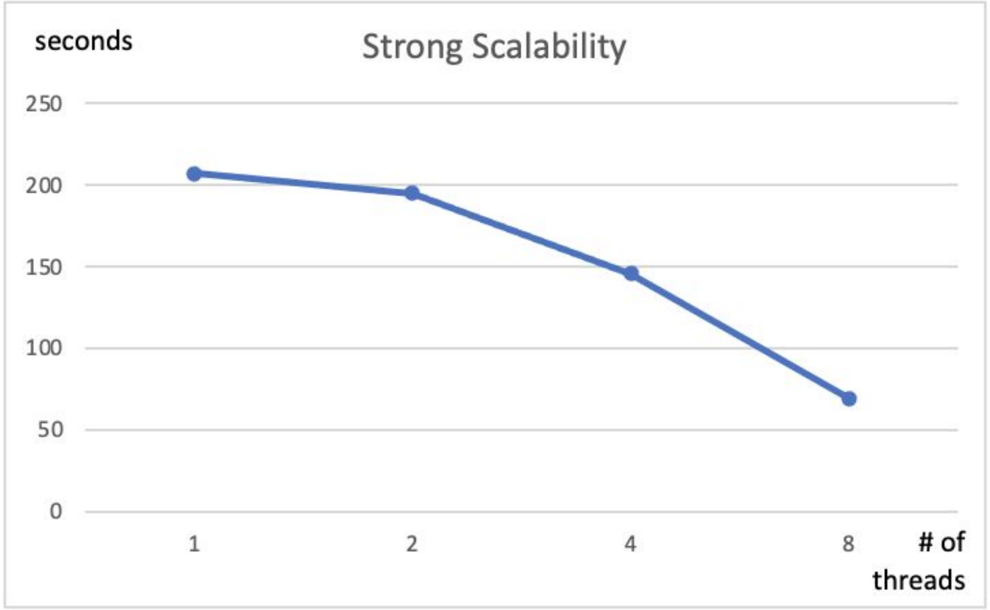
# of threads	Run Time (s)
1	228.508
2	213.006
4	160.474
8	88.268



OpenMP for update_gain() - result

public6.txt

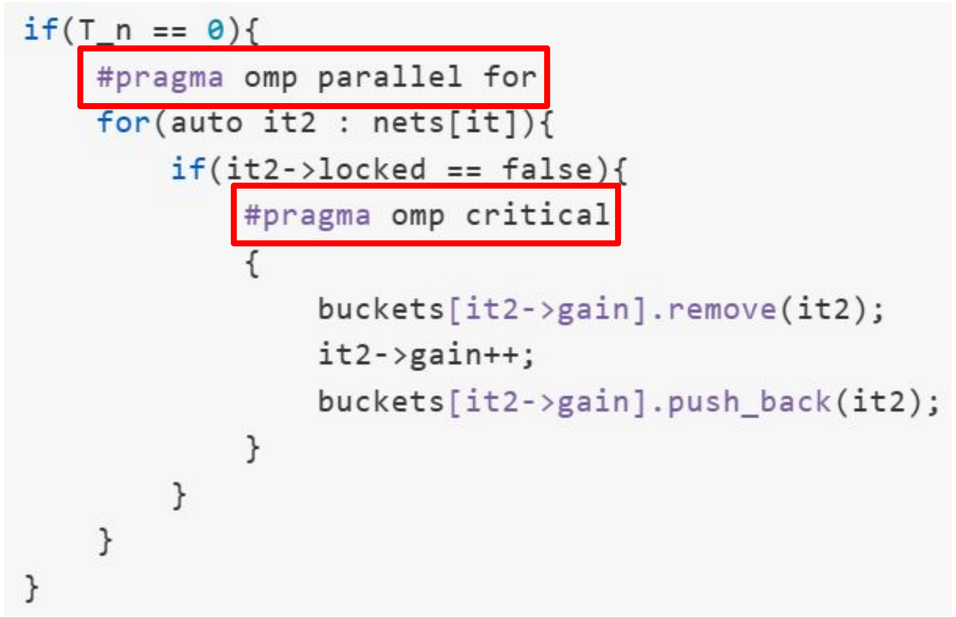
# of threads	Main Loop Time (s)
1	207.109
2	194.924
4	145.463
8	69.304



OpenMP for update_gain

Case 1: 和 base cell 相連的同一個 net 上的不同 cell

對 List 增加或刪除時，會修改容器的大小，需利用 **omp critical** 否則會造成 segmentation fault (list 的指標錯誤)



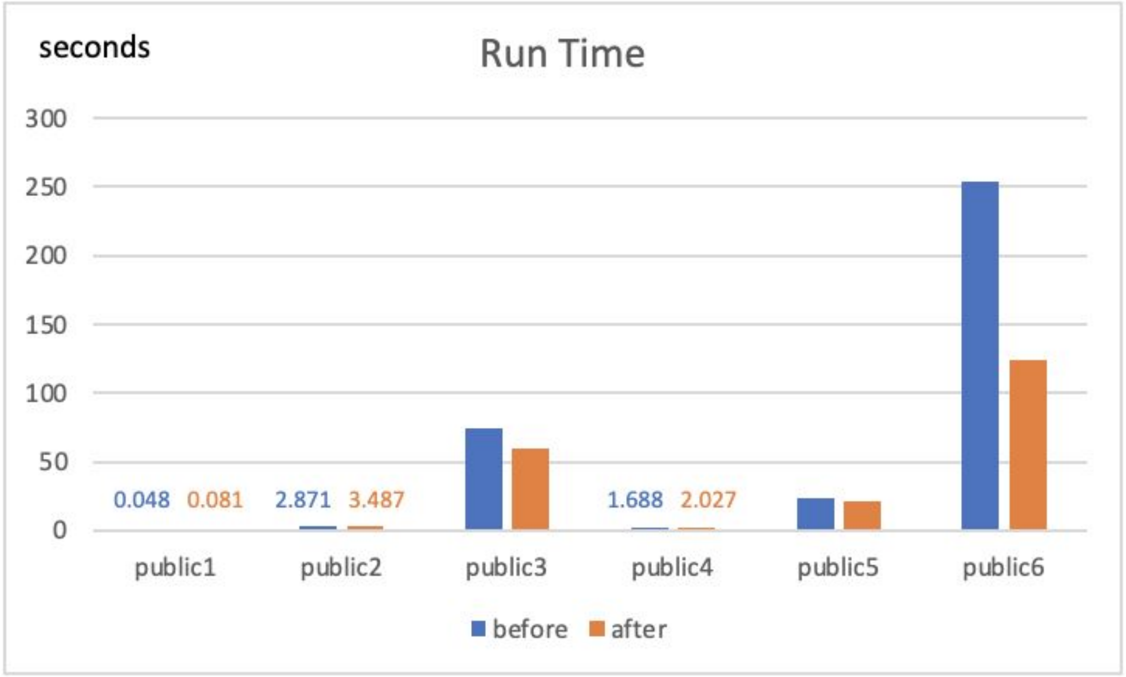
```
if(T_n == 0){
    #pragma omp parallel for
    for(auto it2 : nets[it]){
        if(it2->locked == false){
            #pragma omp critical
            {
                buckets[it2->gain].remove(it2);
                it2->gain++;
                buckets[it2->gain].push_back(it2);
            }
        }
    }
}
```

OpenMP for update_gain case 1 result

設定 4 proc：srun -n1 -c4 ../bin/main ../testcase/public[i].txt ../output/public[i].out

Testcase	Cut size(before)	Time(before)	Cut size(after)	Time(after)
public1	200	0.0924522 S	204	0.0814006 S
public2	3682	3.50781 S	3676	3.48651 S
public3	6910	55.741 S	7097	59.1856 S
public4	1791	1.80591 S	1504	2.02676 S
public5	1518	20.6604 S	1667	21.3453 S
public6	11815	163.169 S	10386	124.423 S

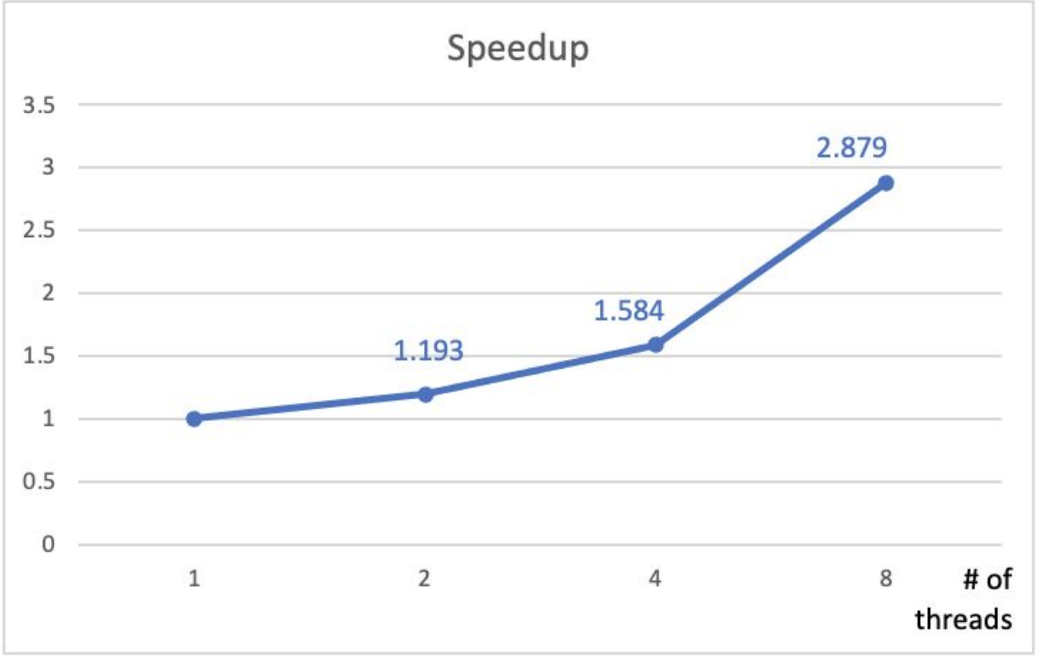
OpenMP for update_gain case 1 result



OpenMP for update_gain case 1 result

public6.txt

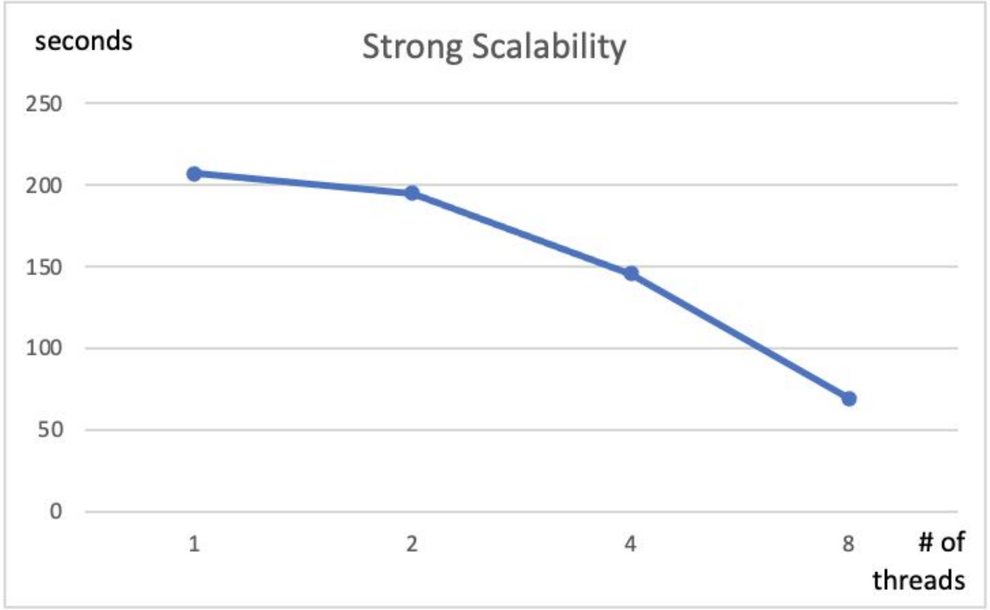
# of threads	Run Time (s)
1	228.508
2	213.006
4	160.474
8	88.268



OpenMP for update_gain case 1 result

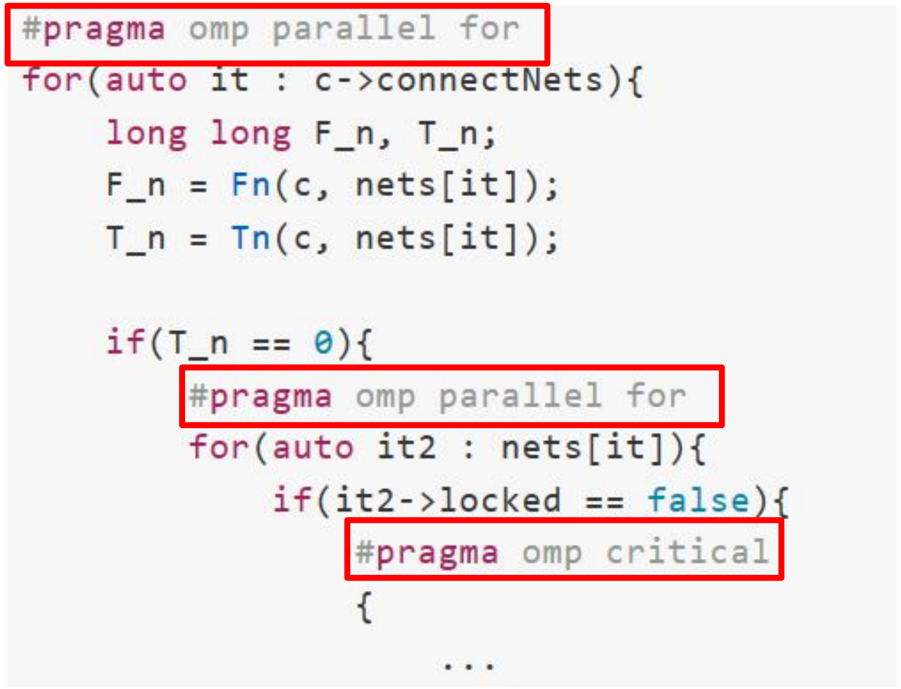
public6.txt

# of threads	Main Loop Time (s)
1	207.109
2	194.924
4	145.463
8	69.304



OpenMP for update_gain

Case 2: 和 base cell 相連的不同 net 上的不同 cell



```
#pragma omp parallel for
for(auto it : c->connectNets){
    long long F_n, T_n;
    F_n = Fn(c, nets[it]);
    T_n = Tn(c, nets[it]);

    if(T_n == 0){
        #pragma omp parallel for
        for(auto it2 : nets[it]){
            if(it2->locked == false){
                #pragma omp critical
                {
                    ...
```

OpenMP for update_gain case 2 result

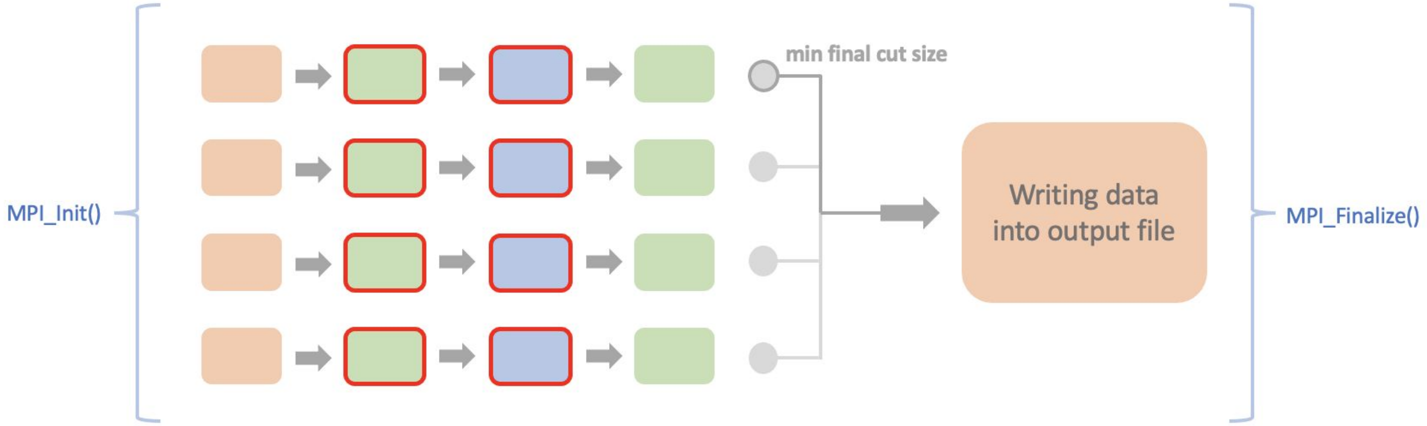
設定 4 proc：srun -n1 -c4 ../bin/main ../testcase/public[i].txt ../output/public[i].out

Testcase	Cut size(before)	Time(before)	Cut size(after)	Time(after)
public1	204	0.0814006 S	200	0.162376 S
public2	3676	3.48651 S	3683	3.50781 S
public3	7097	59.1856 S	6908	108.133 S
public4	1504	2.02676 S	1831	2.64965 S
public5	1667	21.3453 S	1686	20.2047 S
public6	10386	124.423 S	11815	153.281 S

Conclusion

1. 因為在 critical region 裡面的程式碼，無論何時都只會有 1 個 thread 在執行
2. 所以會加速的只有當 locked == true 的情況
3. 如果 locked == false，threads 一樣會 sequential 執行
4. Case 2 的情況，會有更多人要排隊，造成 public6 的時間不降反增

MPI optimization



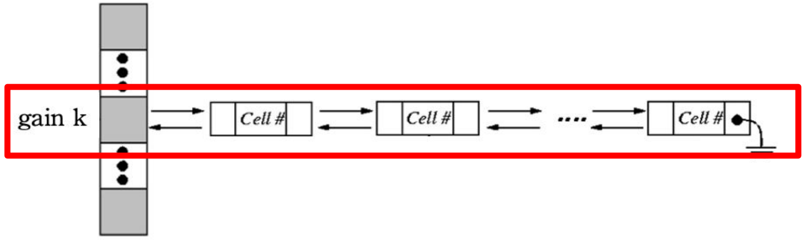
MPI optimization

FM Algorithm step 1.

Select the cell with the highest gain value as the base cell.

因為 bucket[max_gain] 是一個 list，選擇 cell 的順序會影響到最後的結果。

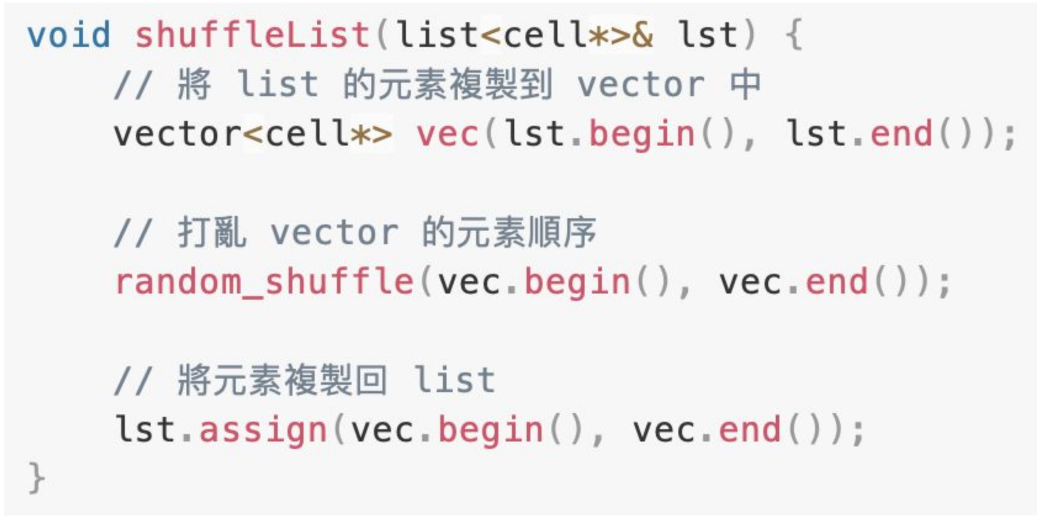
因此我們根據每個 process's rank 設定不同的 seed，shuffle 每次 bucket[max_gain] 中的 cells 次序



bucket list

MPI optimization

因為List不支援shuffle，先複製到vector再shuffle

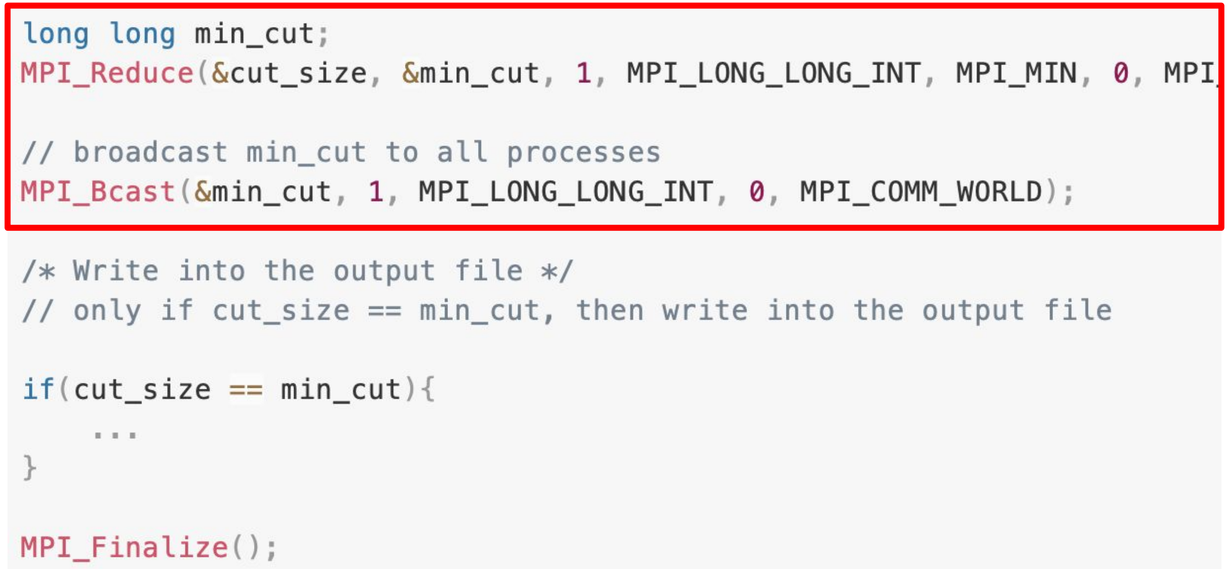


```
void shuffleList(list<cell*>& lst) {
    // 將 list 的元素複製到 vector 中
    vector<cell*> vec(lst.begin(), lst.end());

    // 打亂 vector 的元素順序
    random_shuffle(vec.begin(), vec.end());

    // 將元素複製回 list
    lst.assign(vec.begin(), vec.end());
}
```

MPI optimization

最後，rank 0 process 取得最小的 cut size，並將該值

Broadcasts 到每個 process，由計算出最小 cut size 的 process 負責寫入



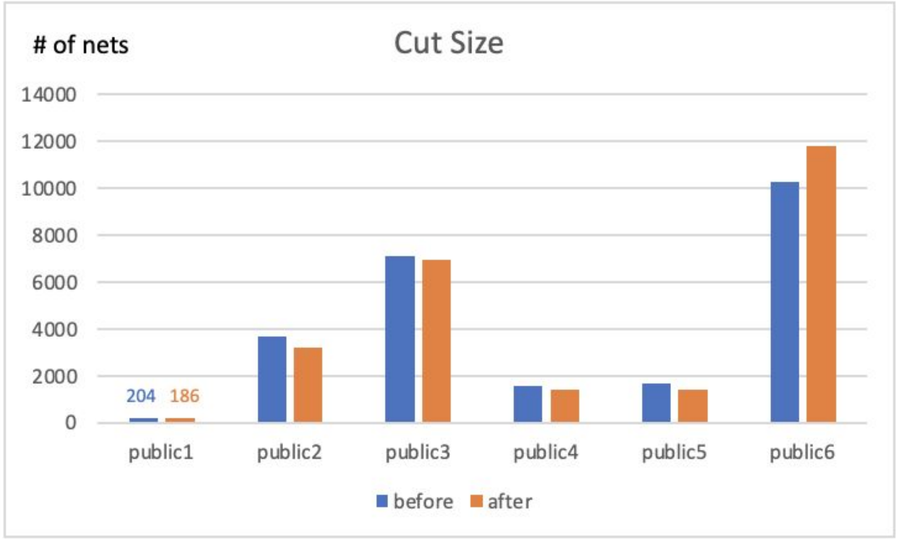
```
long long min_cut;
MPI_Reduce(&cut_size, &min_cut, 1, MPI_LONG_LONG_INT, MPI_MIN, 0, MPI

// broadcast min_cut to all processes
MPI_Bcast(&min_cut, 1, MPI_LONG_LONG_INT, 0, MPI_COMM_WORLD);

/* Write into the output file */
// only if cut_size == min_cut, then write into the output file

if(cut_size == min_cut){
    ...
}

MPI_Finalize();
```

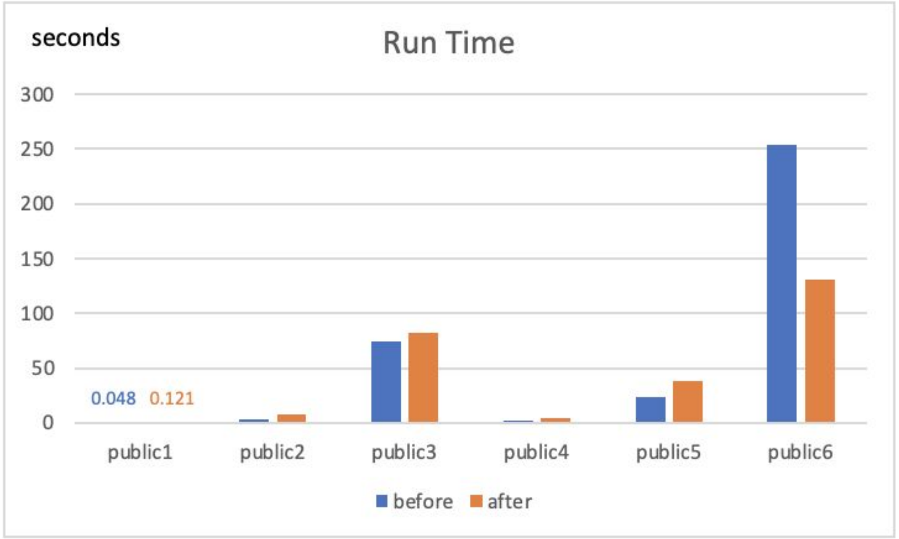
MPI + OMP result

設定 4 tasks, 2 nodes：srun -N2 -n4 -c4 ../bin/main ../testcase/public[i].txt ../output/public[i].out

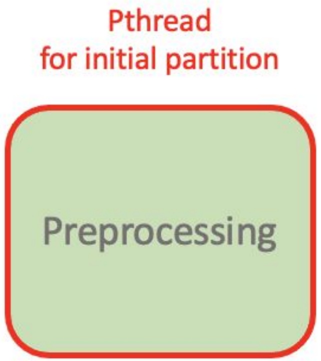
Testcase	Cut size(before)	Time(before)	Cut size(after)	Time(after)
public1	204	0.047678 S	186	0.12082 S
public2	3677	2.87102 S	3185	7.73793 S
public3	7103	74.5056 S	6941	82.0015 S
public4	1581	1.6877 S	1390	4.91123 S
public5	1667	23.8878 S	1428	37.8968 S
public6	10278	254.121 S	11819	130.414 S

MPI + OMP result





Pthread optimization



Pthread optimization

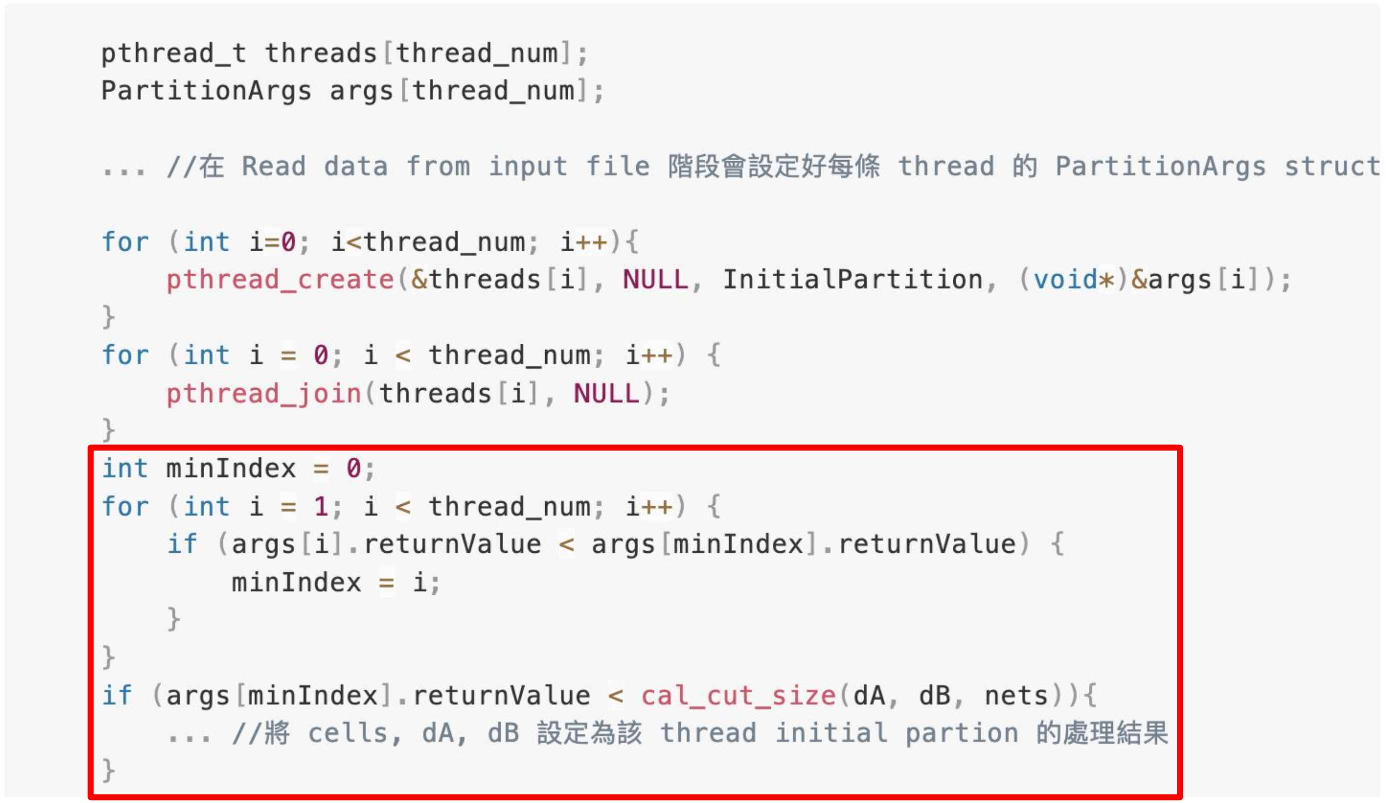
平行使用不同的 seed 對 cellNames 作亂數分配，讓每條 thread 計算出各自的 initial cut size，最終選擇 initial cut size 最小的 thread，以它處理後的配置作為 initial partition

Pthread

```cpp
struct PartitionArgs {
    int id;
    vector<string> cellNames;
    unordered_map<string, cell> cells;
    unordered_map<string, std::vector<cell*>> nets;
    vector<cell*> netCells;
    die dA;
    die dB;
    int returnValue;
};
```

```cpp
void* InitialPartition(void* arg){
    PartitionArgs* args = (PartitionArgs*)arg;
    int tid = args->id;
    unsigned seed = (unsigned) tid+5;
    std::default_random_engine engine(seed);
    std::shuffle(args->cellNames.begin(), args->cellNames.end(), engine);

    ... //根據 shuffle 後的 cellNames 去作 initial partition

    args->returnValue =  cal_cut_size(args->dA, args->dB, args->nets);
    return nullptr;
}
```

Pthread



```
pthread_t threads[thread_num];
PartitionArgs args[thread_num];

... //在 Read data from input file 階段會設定好每條 thread 的 PartitionArgs struct

for (int i=0; i<thread_num; i++){
    pthread_create(&threads[i], NULL, InitialPartition, (void*)&args[i]);
}
for (int i = 0; i < thread_num; i++) {
    pthread_join(threads[i], NULL);
}
int minIndex = 0;
for (int i = 1; i < thread_num; i++) {
    if (args[i].returnValue < args[minIndex].returnValue) {
        minIndex = i;
    }
}
if (args[minIndex].returnValue < cal_cut_size(dA, dB, nets)){
    ... //將 cells, dA, dB 設定為該 thread initial partion 的處理結果
}
```

Pthread optimization

但測試的結果顯示，threads 計算出的 initial cut size 皆遠高於原本的 initial cut

public5.txt

Original initial cut size: 10606

Initial cut size calculated by threads: 75826, 84092, 66888, 65017

推測是跟 input data 的設計有關，原本的 cellNames 中的 cells 順序就是很好的組合

Conclusion

1. 透過 OpenMP，成功讓 run time 降為原來的一半 (public6: 254.121 -> 124.423)
2. 透過 MPI，成功讓大部分 testcase 的 cut size 降低。

Testcase	Cut size(before)	Time(before)	Cut size(after)	Time(after)
public1	204	0.047678 S	186	0.12082 S
public2	3677	2.87102 S	3185	7.73793 S
public3	7103	74.5056 S	6941	82.0015 S
public4	1581	1.6877 S	1390	4.91123 S
public5	1667	23.8878 S	1428	37.8968 S
public6	10278	254.121 S	11819	130.414 S