SME470 Fundamentals of VLSI/FPGA Physical Design Automation Assignment1 -- FM Partitioning

Name: 王竞千（Wang Jingqian）
SID: 12111806

Part 1 – Background

随着集成电路尺寸越来越小，其设计复杂度也越来越高，对布局等物理设计技术的要求也随之增高。为了解决复杂度的问题，常用的策略是分治，即把电路划分为多个子电路，并尽量减少模块之间的连接数。同时还要考虑设计约束的限制，如最大分区大小和最大路径延迟等。虽然电路划分问题是个 NP-hard 问题，但是仍然有算法可以在牺牲设计精确性的前提下大大降低时间复杂度，如 Kernighan–Lin (KL)算法和 Fiduccia–Mattheyses (FM)算法。相比于 KL 算法，FM 算法每次循环只移动 1 个节点，更适应两个分区大小不同的情况；应用超图模型适应更复杂的电路结构；选取移动节点的时间复杂度也比 KL 算法大大优化。[1]

Part 2 – Code & analysis

实现 FM 算法，首先要实现超图(hypergraph)结构。超图由结点(node,代表电路元件 CELL)和边(net)组成，所以要实现一个 CELL 类和一个 NET 类。每个 CELL 要知道与它相连的 NET，而每个 NET 也需要知道它连接的 CELL，但是连接的数量是不固定的。虽然可以自己写动态数组，但是利用 CPP 标准库的容器 vector 和 map 会更加方便。Vector 使用动态内存分配，在堆上申请空间，FILO，内存管理的安全性更高；map 则建立一种 key-value 的对应，方便访问一个 ID 对应节点的成员变量。Multimap 是一种更普适的 map，一个 key 可以对应不同的 value。向 map 插入一个重复的 key 会被拒绝，而 multimap 会新建相应的 key-value 对。Map 和 multimap 的另一个特性是自动排序（基于红黑树），默认升序，调用.end()即可找到 gain 最大的 CELL。

CELL 类:

虽然每次移动都需要重新计算相邻 CELL 的 FS(c)和 TE(c)来确定新的 gain，但是 FS 和 TE 都只是中间变量，只需把 gain 设置为成员变量即可。

```
class CELL {
    public:
    size_t cell_id;
    size_t area;//physical area of the cell
    bool locked;//true for cells locked for the rest of the pass
    int gain;//gain is the change in cutset if this cell were to swap
partitions
    size_t partition;//which partition this cell is in: 0 or 1
    size_t best_partition;//used to remember this cell's partition in
the best solution found so far
    multimap <int, CELL*> :: iterator gain_itr;//an iterator that points
to this cell's location in the gain buckets
    vector <NET*> net_list;//pointers to all nets this cell is on
};
```

Net 类:

相比 CELL 类，net 类的结构非常简单，只需要记录每个 NET 在两个分区链接多少 CELL、这些 CELL 是否可以移动。

```
class NET {
    public:
    size_t net_id;
    size_t num_pins;//total number of pins on this net
    size_t cost;
    bool has_locked[2];//a net is "dead" if no legal cell swaps remain
that could change the cutstate; this occurs when there is a locked or
fixed cell in both partitions on this net
    size_t partition_count[2];//count of how many cells in each
partition on this net
    vector <CELL*> cell_list;//pointers to all cells on this net
};
```

简要介绍一下算法流程：先读入 CELL 和 NET 的信息，写进 Map 里，再随机把 CELL 分进满足大小比例的两组，记录初始 cutcost、计算初始 gain。然后进入循环，不断移动两侧的 CELL，当没有 CELL 可动时，回溯最佳结果并输出。

```
int main(){
    readInput();
    initialPartition();
    min_cut_size = cut_size; //initialize min_cut_size
    computeGains();
    FMPartitionPass();
    printResult();
    return 0;
}
```

源代码选择移动 CELL 的策略与课本有差异。这位 UT Dallas 的 PhD 认为每次轮流从两个分区取移动点，可以快速把权重大的 NET 归到一侧，效率更高。此外，他没有每次遇到更优结果就保存，而是设定了 positive_gains_exhausted，在“拐点”保存最优结果。这个技巧我没太看明白，反正结果大概是对了……

```
//if there is nothing left in one of the gain_buckets, use the other
bucket
    if(gain_bucket[0].size() == 0) //if no cells in bucket 0
        from_partition = 1;
    else if(gain_bucket[1].size() == 0) //if no cells in bucket 1
        from_partition = 0;
    else from_partition = base_partition;

    base_partition = !base_partition; //pick from alternating partitions
to make large nets dead faster
```

本算法的重点之一是在 CELL 移动后更新其他 CELL 的 gain。由于移动的 CELL 之后就锁定了，不需要更新它的 gain 了，只需要更新来侧和对侧的其他自由 CELL，可以分为 4 类来讨论。观察与移动 CELL 相连的一个 NET：在移动前，如果对侧没有 CELL，则移动后所有 CELL 原本的 TE 需要被除去，全体+1；如果对侧有 1 个 CELL，来侧的其他 CELL 本来没有 FS 和 TE，不

需要变，对侧的那个 CELL 原本的 FS 被除去，对侧-1；如果对侧有 2 个以上的 CELL，则全体都没有 gain 变化，不需要讨论；同理，在移动后，如果来侧没有 CELL 剩余，则对侧所有 CELL 获得 TE，全体-1；如果来侧剩余 1 个 CELL，对侧所有 CELL 不变，来侧剩下的 CELL 获得 FS，来侧+1；如果来侧剩余 2 个以上，也是全体 gain 不变，不需要讨论。伪代码如下：

```
if(to == 0){
    for(k = 0; k < size; k++) {
        if(!cell[k].locked){
            cell[k].gain += net[i].cost;
        }
    }
} else if (to == 1) {
    for(k = 0; k < size; k++) {
        if(!cell[k].locked && base_part != cell[k].part){
            cell[k].gain -= net[i].cost;
            break;
        }
    }
}
from--;
to++;
if(from == 0) {
    for(k = 0; k < size; k++) {
        if(!cell[k].locked){
            cell[k].gain-=net[i].cost;
        }
    }
} else if (from == 1) {
    for(k = 0; k < size; k++) {
        if(!cell[k].locked && base_part == cell[k].part) {
            cell[k].gain += net[i].cost;
            break;
        }
    }
}
```

Part 3 – Result & Verification

运行环境：gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

语法标准：C++17

Test case1:

编译命令：g++ fm_trial.cpp

运行命令：./a.out

```
wangjq@LAPTOP-CL1EKPAQ:/mnt/d/cpp$ ./a.out
Please enter the number of nodes:4
Please enter each of the 4 nodes with its id and the node area:
0 1
1 1
2 1
3 1
Please enter the number of edges:3
Please enter each of the 3 edges with the number of connected nodes and their node ids, followed by the edge cost:
2 0 1 1
2 1 2 3
2 2 3 1
Please enter the percentage of the ratio factor:50
The Initial Cutcost:3
****************************************
The result is:
id:0 group:0 gain:1
id:1 group:1 gain:-2
id:2 group:1 gain:-4
id:3 group:1 gain:-1
The final Cutcost:1
```

Test case2:

```
wangjq@LAPTOP-CL1EKPAQ:/mnt/d/cpp$ ./a.out
Please enter the number of nodes:4
Please enter each of the 4 nodes with its id and the node area:
0 1
1 4
2 2
3 1
Please enter the number of edges:3
Please enter each of the 3 edges with the number of connected nodes and their node ids, followed by the edge cost:
3 0 1 2 5
3 0 2 3 3
3 0 1 3 4
Please enter the percentage of the ratio factor:50
The Initial Cutcost:12
****************************************
The result is:
id:0 group:1 gain:-5
id:1 group:1 gain:-5
id:2 group:1 gain:-5
id:3 group:0 gain:7
The final Cutcost:7
```

上面的结果是我挑出来的，实际上第一个样例有可能最终 cutcost 是 2，第二个最终 cutcost 是 0。第二个结果可以解释，因为最大 CELL 面积刚好为总面积的一半，所以允许所有 CELL 都被归到同一组；第一个结果应该是 saveBestSolution()保存了错误的最佳结果，估计跟 positive_gains_exhausted 有关。为什么是这样，我也懒得分析了（我是条卑微的考研狗）

Part 4 – References

[1]Kahng, Andrew & Lienig, Jens & Markov, Igor & Hu, Jin. (2022). VLSI Physical Design: From Graph Partitioning to Timing Closure. 10.1007/978-3-030-96415-3.

[2]C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," 19th Design Automation Conference, Las Vegas, NV, USA, 1982, pp. 175-181

[3] https://github.com/searsm8/FM_Partitioning/tree/master searsm8/FM_Partitioning: Program which implements Fiduccia–Mattheyses partitioning algorithm (github.com) （UT Dallas 大佬的源代码）