**第一章 结果**

以下图1-1至图1-5是case0和case1的两个实例运行结果。用case图形总面积除以打包后矩形的面积得到面积占用率.Case0面积占用率：0.8273. Case1面积占用率0.7957.

|  |  |  |
| --- | --- | --- |
| 图1-1 Case0面积占用率优化过程 | 图1-2 Case0每次优化耗时 | 图1-3 Case0打包结果 |

|  |  |  |
| --- | --- | --- |
| 图1-3 Case1面积占用率优化过程 | 图1-4 Case0每次优化耗时 | 图1-5 Case0打包结果 |

所有case的运行结果如表1-1所示.在参数未优化完毕的情况下,所有case均优化两代,在linux服务器下使用32个CPU核运行.下表中case0和case1的运行结果与上面不同是因为这是两次独立的测试.每次测试的结果会有一定的偏差.

表1-1所有数据集的运行结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| case | 数量 | 总面积 | 长 | 宽 | 占用率 | 运行时间s |
| 0 | 7 | 435 | 219 | 239 | 83.11% | 0.10 |
| 1 | 35 | 1929 | 469 | 519 | 79.25% | 1.70 |
| 2 | 38 | 2243 | 499 | 566 | 79.42% | 2.16 |
| 3 | 85 | 4997 | 708 | 839 | 84.12% | 11.62 |
| 4 | 182 | 8828 | 937 | 1127 | 83.60% | 72.46 |
| 5 | 301 | 16959 | 1288 | 1559 | 84.46% | 240.05 |
| 6 | 165 | 27578 | 1647 | 1989 | 84.18% | 50.61 |
| 7 | 77 | 4630 | 691 | 808 | 82.93% | 9.89 |
| 8 | 65 | 7575 | 877 | 1038 | 83.21% | 6.65 |
| 9 | 159 | 9812 | 979 | 1186 | 84.51% | 47.97 |
| 10 | 68 | 3606 | 628 | 718 | 79.97% | 7.00 |

**第二章 主要技术路线及实现方式**

**2.1 放置策略**

**2.1.1 NO-FIT POLYGON**

根据文献[1],采用计算no-fit polygon(简写为NFP)的方式来计算图形间是否有重合部分.如图所示计算三角形B相对于多边形A的外接多边形,只需要将A保持和B外接,然后绕A滑行一周,B的参考点轨迹形成的多边形就是所求的NFPAB.如果B的参考点落入NFPAB范围内,那么B必然和A重叠.如果B的参考点落在NFPAB边界上,A和B刚好外接.如果B的参考点落NFPAB在边界外,A和B是相离的关系.这种方式将两个图形间是否重叠问题转化为点是否在图形内部的问题,极大简化了重叠检测过程.一个多边形放置时,它的参考点不应该落入此多边形与其他多边形的NFP范围内,否者多边形之间就会产生重叠.

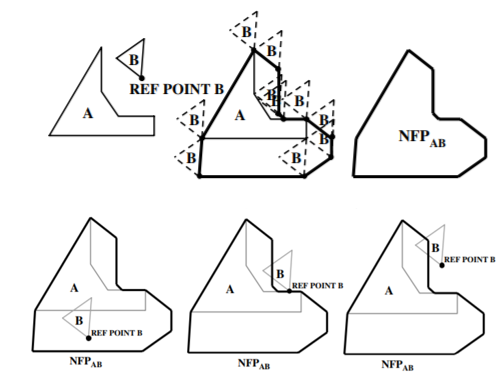


图2-1 no-fit polygon示意图[1]

类似的,我们可以构建内部no-fit polygon,如图2-2所示,B位于A的内部,将A与B保持外接,然后B绕A滑行一周, B的参考点轨迹形成的多边形就是所求的内部NFPAB. 如果B的参考点落入NFPAB范围内,那么B必然在A内部.没有超出A的范围.如果B的参考点落在NFPAB边界上,A和B刚好外接.如果B的参考点落NFPAB在边界外,B就超出了A的边界.一个多边形放置时,它的参考点都应该落入此多边形与容器的内部NFP内,否则多边形就会超出容器的边界.

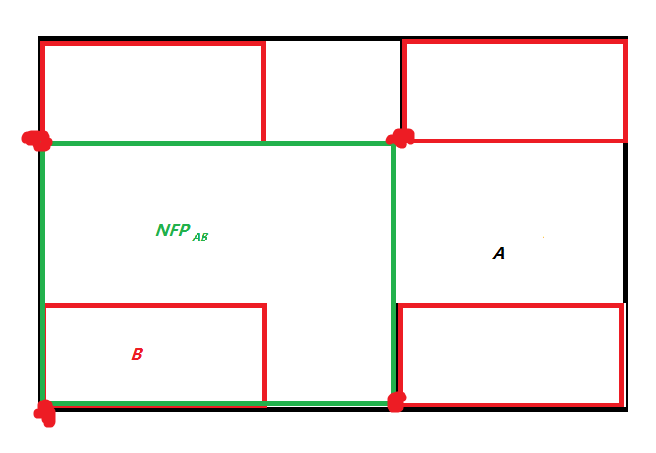


图2-2 内部no-fit polygon示意图

**2.1.2 下左位置放置策略**

根据文献[5] ,我们采用下左位置放置(Bottom-left position)策略.每次放入一个一个多边形,总是选择y轴坐标最小的可行位置放置.如果同时有多个位置满足需要,则选择最左侧的那个点.这样能够使得最终的布局有最小的高度.

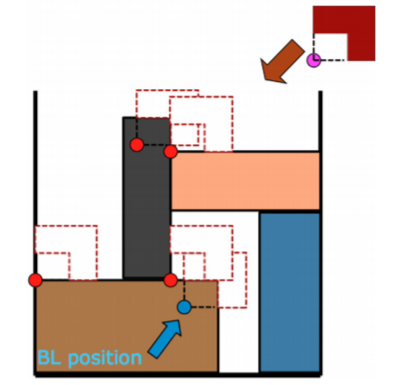


图2-3 下左位置示意图[5]

**2.1.3 放置过程**

按照下左位置放置策略,首先将第一块多边形P1放入与容器的内部NFP内,第n块多边形Pn放置时,先计算Pn与已放置多边形的NFP的并NFP\_AND(Pn参考点落入NFP\_AND内部会与已放置多边形产生重叠),其次计算Pn与容器的内部NFP\_BIN(Pn参考点落在NFP\_BIN外部会超出容器的边界),最后计算NFP\_BIN与NFP\_AND的差得到NFP\_FEASBILE, Pn的参考点应该落入NFP\_FEASBILE内,寻找下左点进行放置.一共需要计算O(nlogn)个NFP完成一次打包过程.

**2.2 优化策略**

我们采用了遗传算法对布局进行优化,搜索出最优解.

遗传算法是模拟自然的优胜劣汰的进化，采用多次迭代的数学计算模型，利用适应度寻找最理想的染色体的过程。主要由适应度函数，染色体编码，选择，交叉和变异五个部分组成。基本遗传算法是由七个元素组成

GA = (M, F, S, C, Mc, Pc, Pm)

式中：M群体大小, F个体适应度评价函数, S选择操作算子, C交叉操作算子, Mc变异操作算子, Pc交叉概率, Pm变异概率.

**2.2.1 染色体编码方式**

一个个体就是一个布局.个体含有1个染色体,染色体上有两个基因placement基因和rotation基因.

placement基因是一个列表,每项存放着一个多边形的坐标信息.placement的顺序就是多边形放置的顺序.



图2-4 placement基因示意图

rotation基因是一个列表,每项对应着placement对应顺序的多边形旋转的角度.



图2-5 rotation基因示意图

**2.2.2 适应度评价函数**

放置完所有多边形后得到适应度fitness,计算方式如下:

fitness = FITNESSSCALE\*Height/binArea

式子中FITNESSSCALE是为了将fitness的值变大而乘的常数. Height是放置后得到的最小高度. binArea是容器的面积.

**2.2.3 选择操作算子**

带精英策略,保存fitness最低的个体不变动,直接保存到下一代.交配的父母从所有的个体中按权重随机选择,排名在前的个体被选择的概率大.先选父个体,再选母个体,选母个体的时候排除已选的父个体.

**2.2.4 交叉操作算子**

一对父母个体产生两个子个体.在基因长度范围内随机产生一个断点,切掉断点后的基因.例如父个体基因经过切割后得到gen1,在母个体基因中找到被切除的形状和角度,添加到gen1中.这就是子个体1.同理母基因也这样产生子个体2

**2.2.5 变异操作算子**

交配产生的子个体经过变异才进入新代种群.变异算子有:交换个体placement基因顺序,随机改变rotation基因数值.

**2.3 实现方式**

程序采用python3.7编写.使用并行结构加快计算过程.使用MPI高性能信息传递库进行各进程间的信息传递.主进程主要运行种群进化过程,子进程主要对个体进行评估得到适应度函数值.使用到的第三方库有以下几个:

1. mpi4py 以MPI为基础进行构建，进而允许python的数据结构在多进程中传递.

2. pyclipper 主要用于多边形坐标偏移和多边形切割问题，在本队代码中用于确定多边形满足不重叠且不超过规定边界的约束条件.

3. polygon 在二维情况中处理多边形，简化对多边形进行的相关操作.

更具体的实现方式见附录设计文档.

**第三章 参考文献**

1. Burke, E. K. , Hellier, R. S. R. , Kendall, G. , & Whitwell, G. . (2007). Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research,* *179*(1), 27-49.

2. Chen, D. , Liu, J. , Fu, Y. , & Shang, M. . (2010). An efficient heuristic algorithm for arbitrary shaped rectilinear block packing problem. Computers & Operations Research, 37(6), 1068-1074. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0305054809002299>

3. Fujiyoshi, K. , Kodama, C. , & Ikeda, A. . (2007). A fast algorithm for rectilinear block packing based on selected sequence-pair. Integration the Vlsi Journal, 40(3), 274-284. Retrieved from <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1354021>

4. Hu, Y., Hashimoto, H., Imahori, S., & Yagiura, M. . (2012). Heuristic Algorithms for Rectilinear Block Packing (New Trends in Algorithms and Theory of Computation). Retrieved from <http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/1799-24.pdf>

5. Hu, Y. , Hashimoto, H. , Imahori, S. , & Yagiura, M. . (2015). Efficient implementations of construction heuristics for the rectilinear block packing problem. Computers & Operations Research, 53(jan.), 206-222. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0305054814001798>

6. Hu, Y. , Hashimoto, H. , Imahori, S. , Uno, T. , & Yagiura, M. . (2016). A partition-based heuristic algorithm for the rectilinear block packing problem. Journal of the Operations Research Society of Japan, 59(1), 110-129. Retrieved from <https://www.jstage.jst.go.jp/article/jorsj/59/1/59_110/_article>

7. Matsushita, K. , Hu, Y. , Hashimoto, H. , Imahori, S. , & Yagiura, M. . (2016). A new solution representation for the rectilinear block packing problem. 2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). IEEE. Retrieved from <https://ieeexplore.ieee.org/document/7797842>

8. Matsushita, K. , Hu, Y. , Hashimoto, H. , Imahori, S. , & Yagiura, M. . (2018). Exact algorithms for the rectilinear block packing problem. Journal of Advanced Mechanical Design Systems & Manufacturing, 12. Retrieved from <https://www.jstage.jst.go.jp/article/jamdsm/12/3/12_2018jamdsm0074/_pdf/-char/en>

9. Matayoshi, M. . (2011). Two dimensional rectilinear polygon packing using genetic algorithm with a hierarchical chromosome. IEEE International Conference on Systems. IEEE. Retrieved from <https://ieeexplore.ieee.org/document/6083798>

10. The University of Melbourne. (2017, October 12). Rectilinear packing without rotation [Video]. Coursera. <https://www.coursera.org/lecture/advanced-modeling/2-4-2-rectilinear-packing-without-rotation-9KEwj>

**附录**

**设计文档**

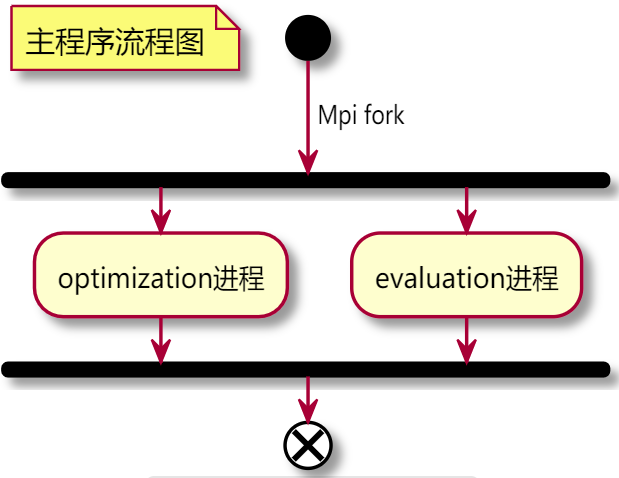
**一、背景介绍**

随着信息科技的迅猛发展，集成电路(IC)在各方面应用方面的普及度也在不断增加，其衡量尺度主要有五个指标：集成度，特征尺寸，晶片直径，芯片面积和封装。其中集成度用于衡量集成电路的规模，为了芯片能够在尽可能小的面积的情况下尽可能大的提高集成度，集成电路版图的拼接成为提高集成电路设计水平的有效手段。此外，优化版图拼接，减少芯片面积能够有效提高每次流片时每个晶圆上的晶片数，进而提高每次流片的合格芯片数以实现成本降低和成品数的提高。在提高芯片集成度的同时，智能拼接算法在其它例如管理工程中，对于流通货物装载问题的求解，尽可能高的提高容器装载率进而提高企业竞争力等方面依旧有着较为重要的作用。因此设计一个好的智能版图拼接算法刻不容缓。

**二、设计思路**

**2.1 软件架构设计**

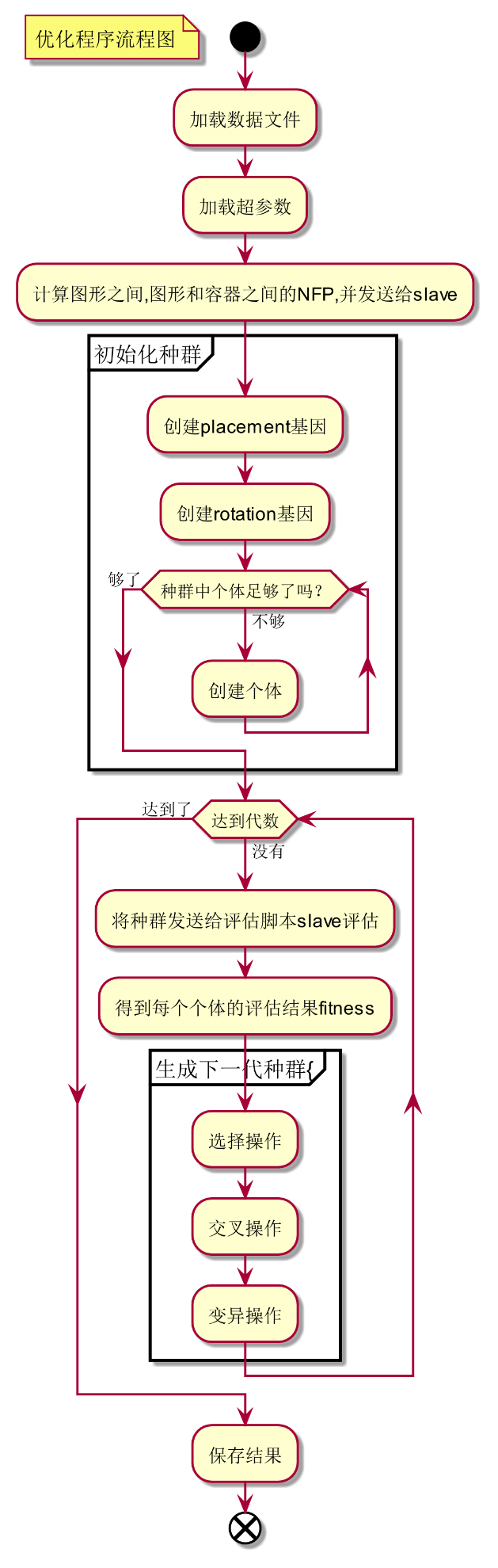
如附图2-1即为软件采用的并行架构.使用MPI高性能信息传递库进行各进程间的信息传递.主进程主要运行种群进化过程,子进程主要对个体进行评估得到适应度函数值.此外子进程还可用于NFP的计算.



附图2-1 软件架构图

**2.1.1 优化脚本设计**

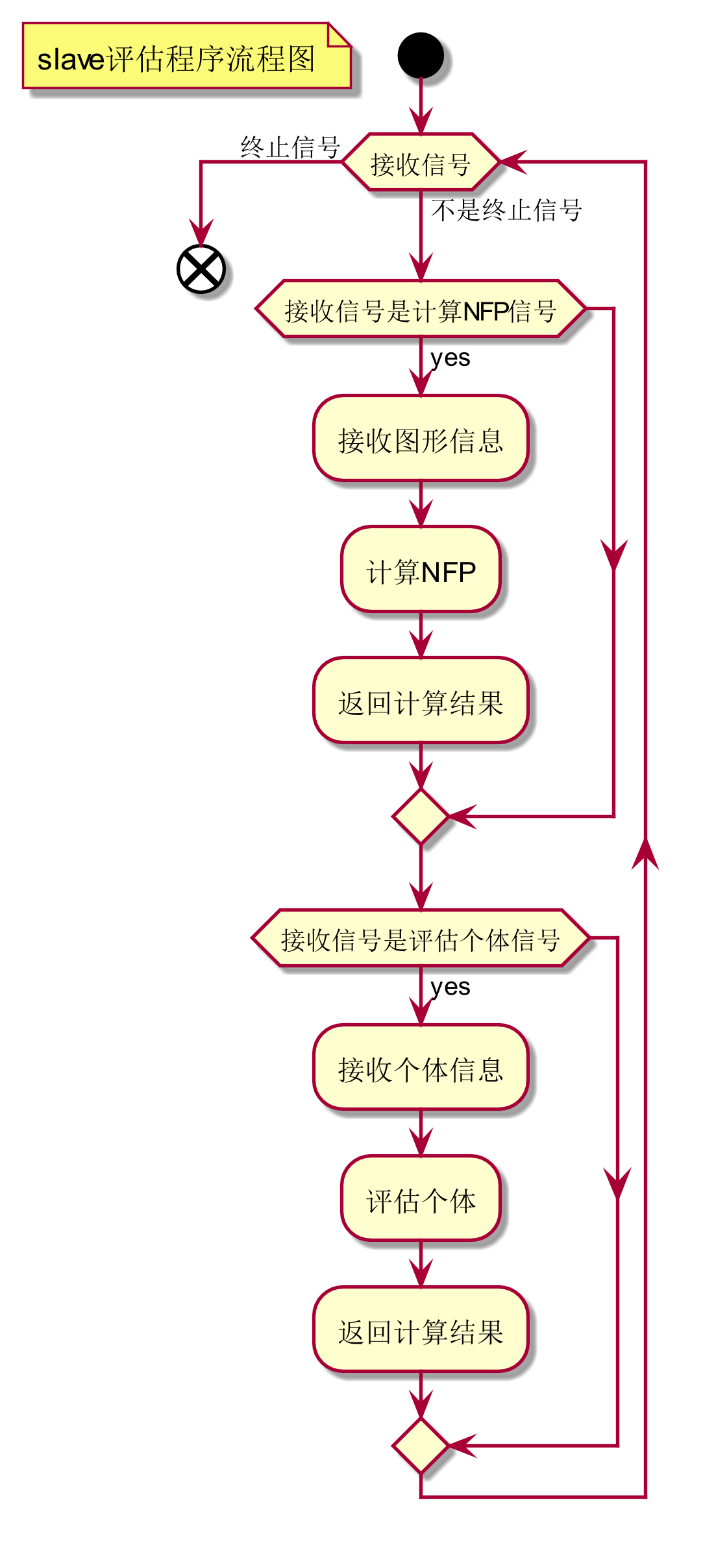
如附图2-2即为优化脚本的流程图,优化脚本主要运行种群进化过程.



附图2-2 优化脚本流程图

**2.1.2 评估脚本设计**

如附图2-3即为评估脚本的流程图,评估脚本可用于NFP的计算以及种群个体的评估.



附图2-3 评估脚本流程图

**2.1.2 主要模块设计**

1.Nester模块控制整个打包的过程

2.genetic\_algorithm模块是遗传算法部分.包含mutate(变异算子),random\_weighted\_individual(选择算子)等部分

3.PlacementWorker模块主要对个体进行评估,得到适应度函数