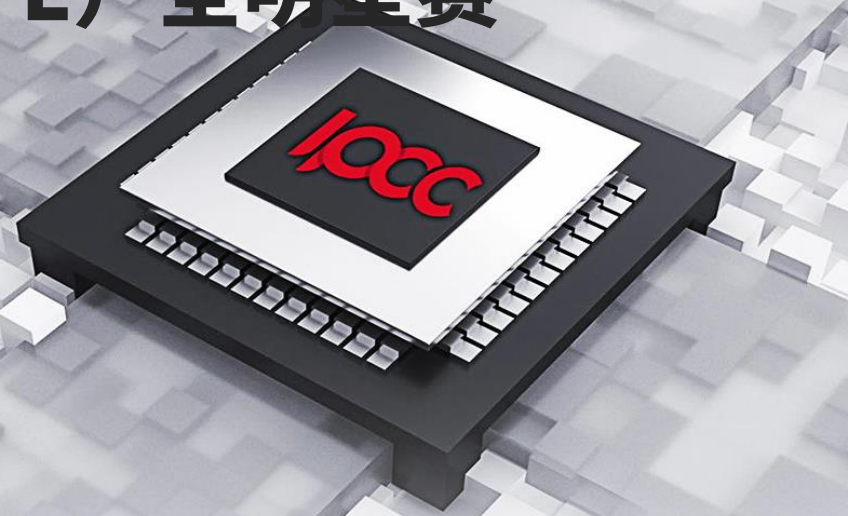
Association for Computing Machinery

ACM中国 国际并行计算挑战赛

AMD

全国大学生高性能计算超级联赛（HPC-PL）全明星赛

暨第三届ACM中国-国际并行计算挑战赛开幕赛



汇报人：罗智宇　2022年5月22日

目录 CONTENTS

赛题简介

康威生命游戏(Conway's Game of Life)，又叫做康威生命棋，是康威于1970年发明的一个数学游戏。它规定了一个网格状的世界内每一个细胞的生存规则：

每个细胞有两种状态——存活或死亡，在网格状的世界中，每个细胞与以自身为中心的周围八格细胞产生互动；

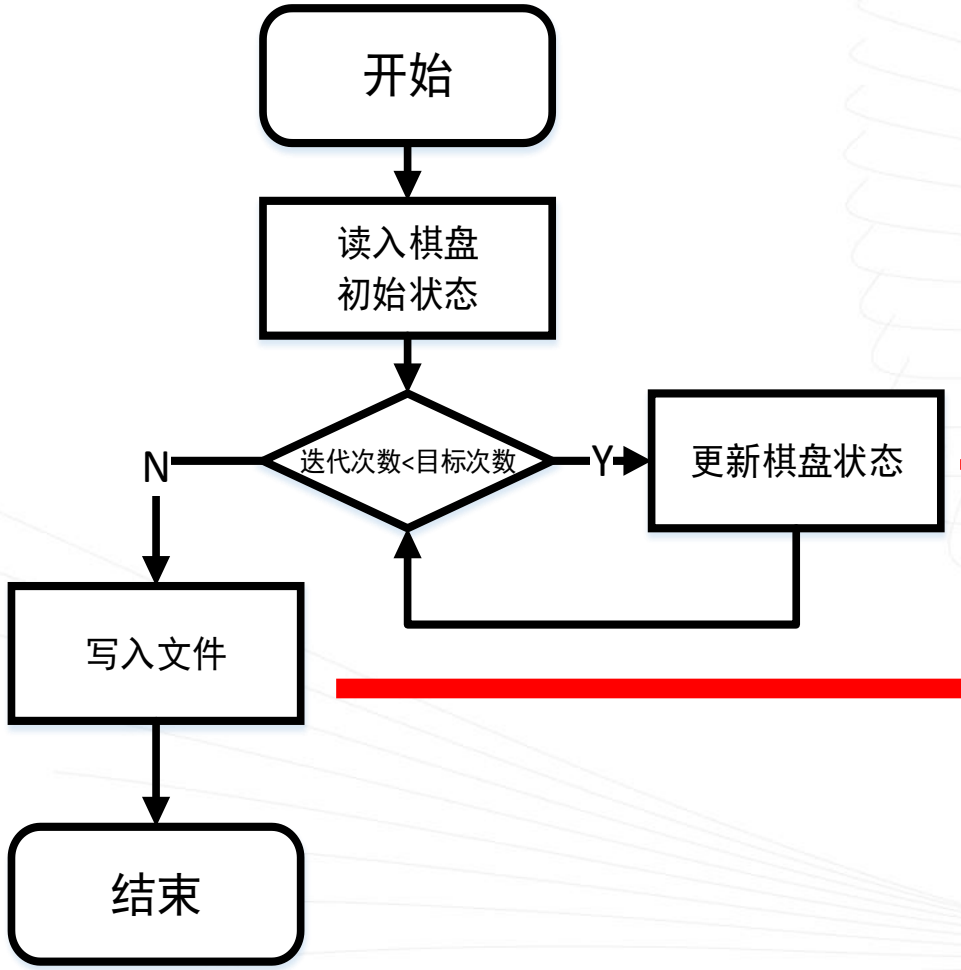
- **生命数量稀少时无法生存**：
  当前细胞为存活状态时，当周围的存活细胞低于2个时（不包含2个），该细胞变成死亡状态；
- **生存环境过于拥挤时无法生存**：
  当前细胞为存活状态时，当周围有2个或3个存活细胞时，该细胞保持原样。
  当前细胞为存活状态时，当周围有超过3个存活细胞时，该细胞变成死亡状态。
- **繁殖**：
  当前细胞为死亡状态时，当周围有3个存活细胞时，该细胞变成存活状态。

应用程序运行的硬件环境和软件环境

本程序采用组委会提供的参赛平台

- CPU型号： AMD EPYC 7452 32-Core Processor
- 节点信息：单节点，双路，64核心，251GB
- 网络： 56GB Infiniband高速网
- 操作系统： CentOS Linux release 7.9.2009
- 编译器：icpc 2021.5.0 /gcc version 4.8.5
- MPI：Intel mpi 2021.5

应用程序的代码结构



- 编译选项优化
- 并行优化 → 进程间通信优化
- 访存优化
- IO优化

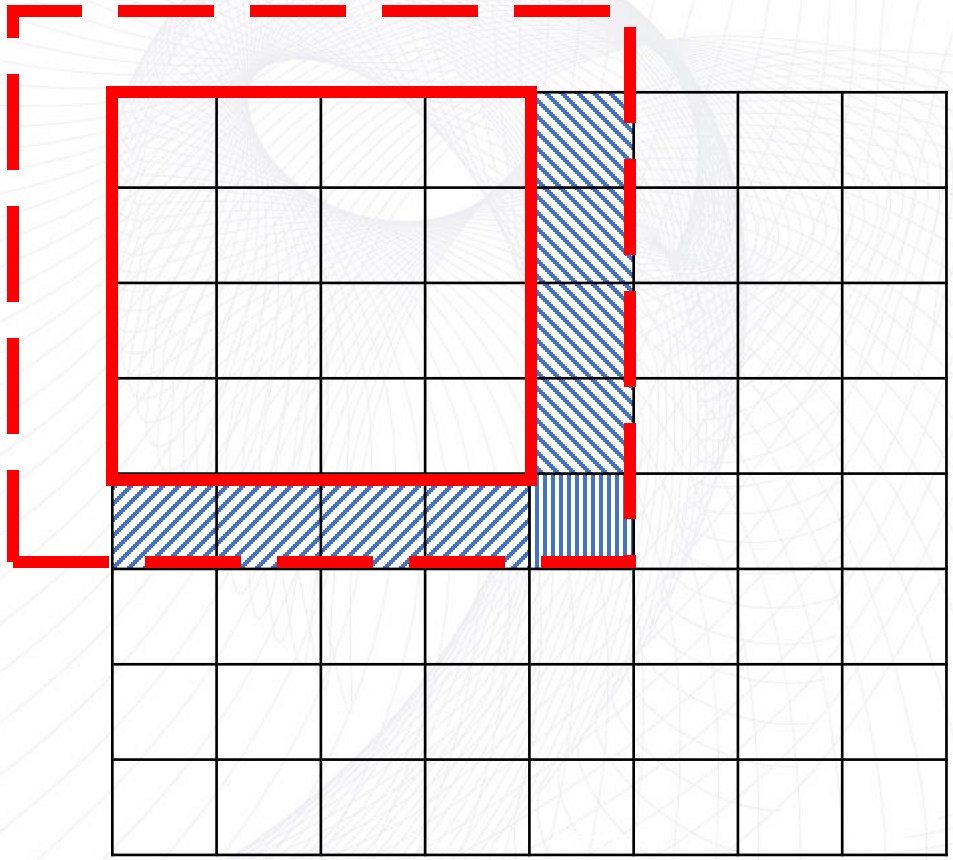
1.编译选项优化

使用“mpiicpc -O3 -march=core-avx2 ./main.cpp -fma -ipo -o ../run/lifegame”对代码进行编译

2.并行优化-使用128进程并行运行更新过程

计算较为简单，如何减少通信开销，提高并行效率？

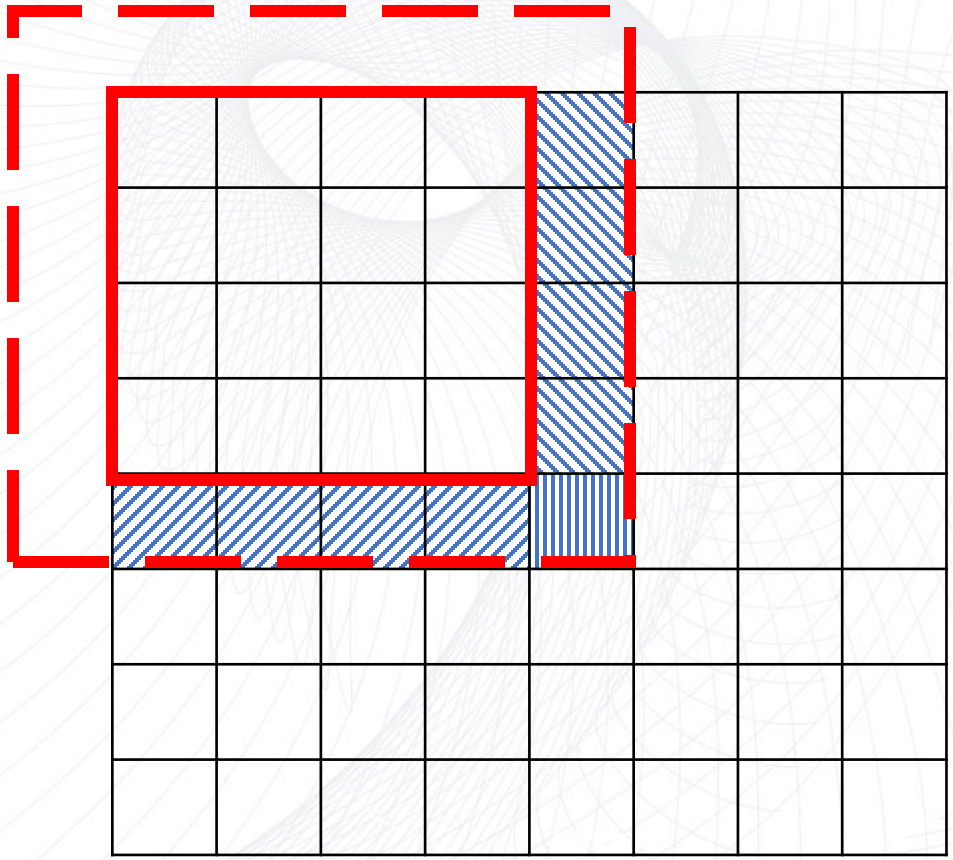
- 使用非阻塞通信，将计算和通信重叠
- 手动将MPI进程与处理器核绑定
- 使一次通信发送尽可能多的数据，减少通信次数
  - 先上下通信，再左右通信
  - 使用MPI_Type_vector打包左右边界数据



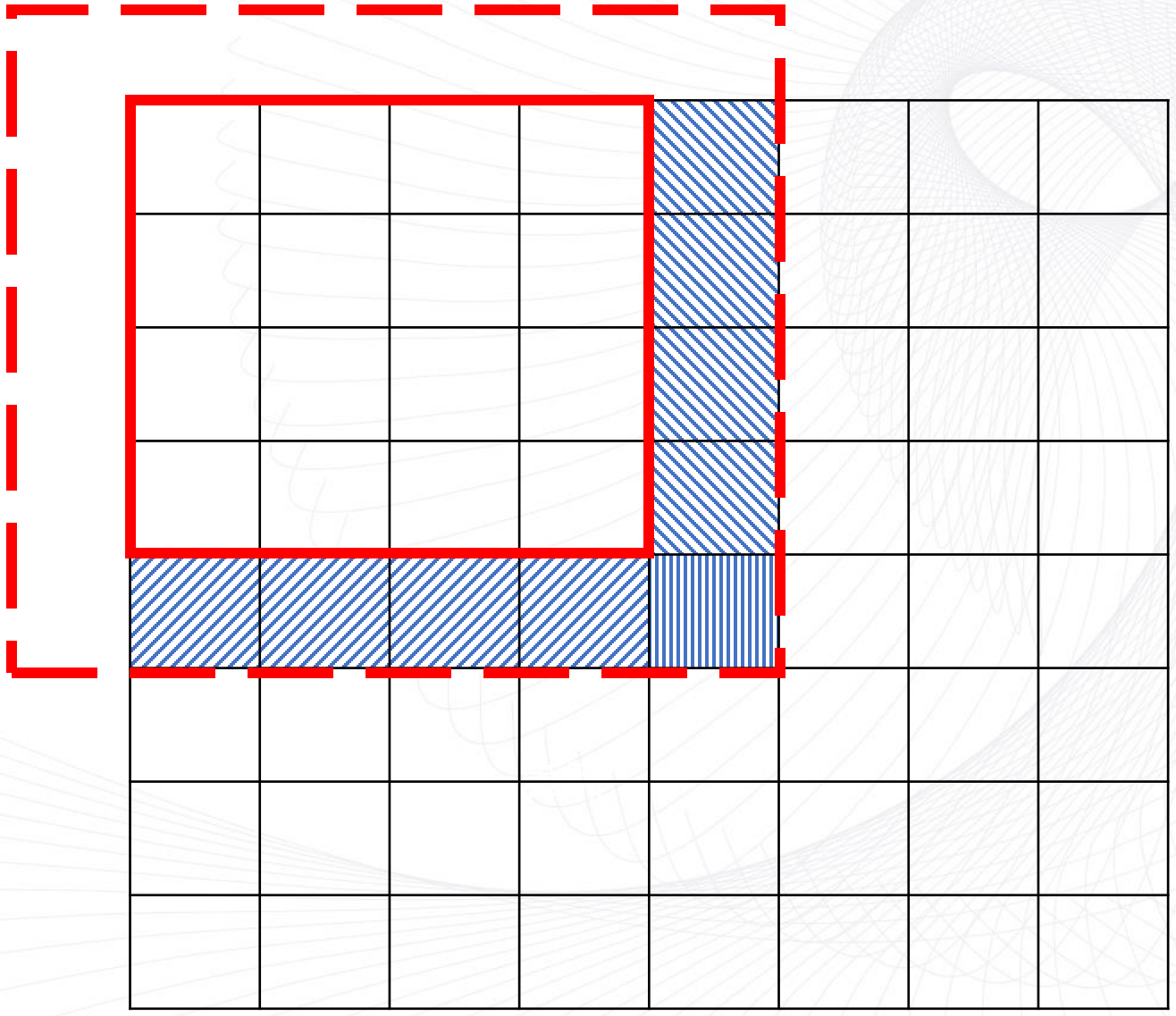
2.并行优化-使用128进程并行运行更新过程

计算较为简单，如何减少通信开销，提高并行效率？

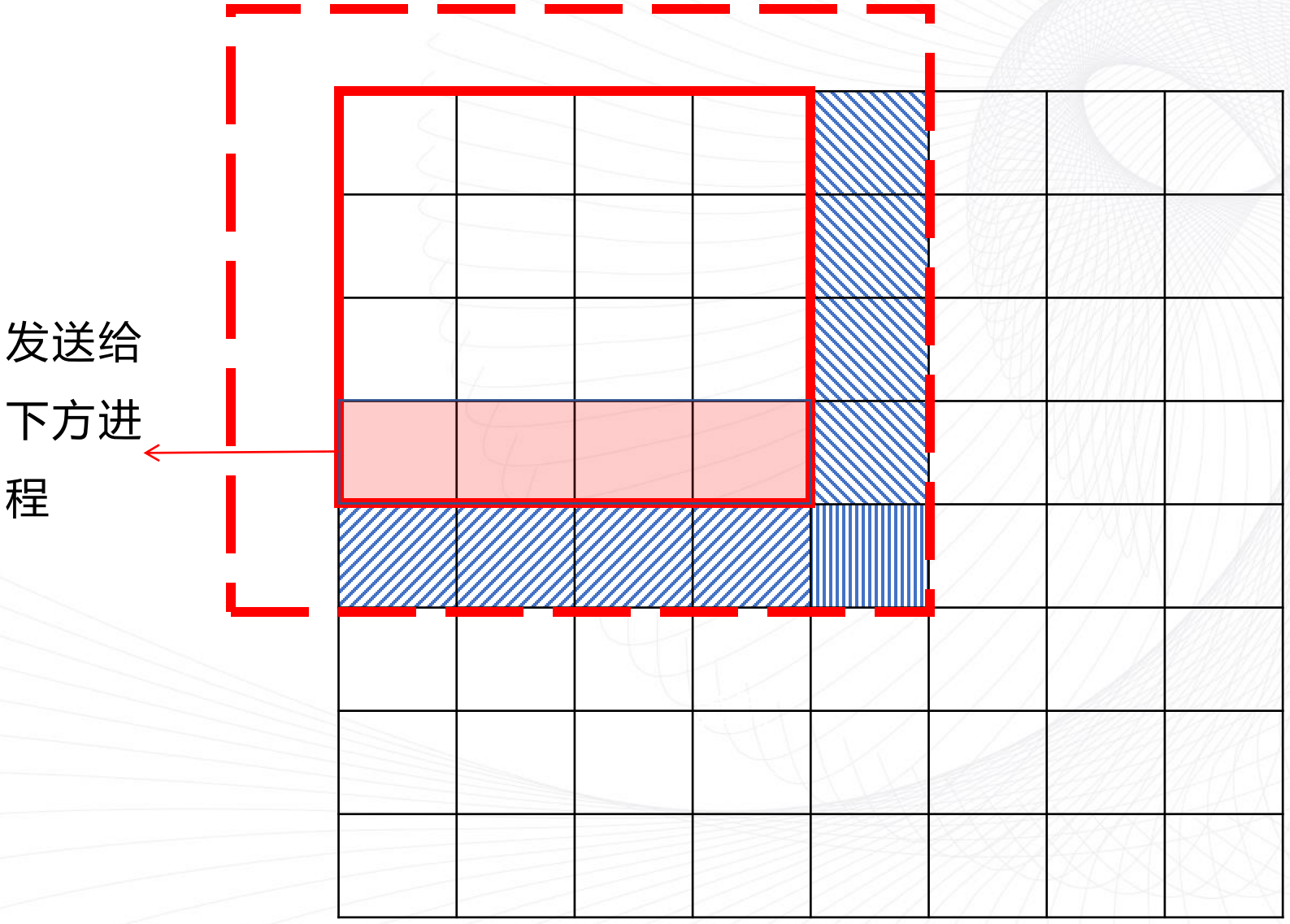
- 使用非阻塞通信，将计算和通信重叠
- 手动将MPI进程与处理器核绑定
- 使一次通信发送尽可能多的数据，减少通信次数
  - 先上下通信，再左右通信
  - 使用MPI_Type_vector打包左右边界数据



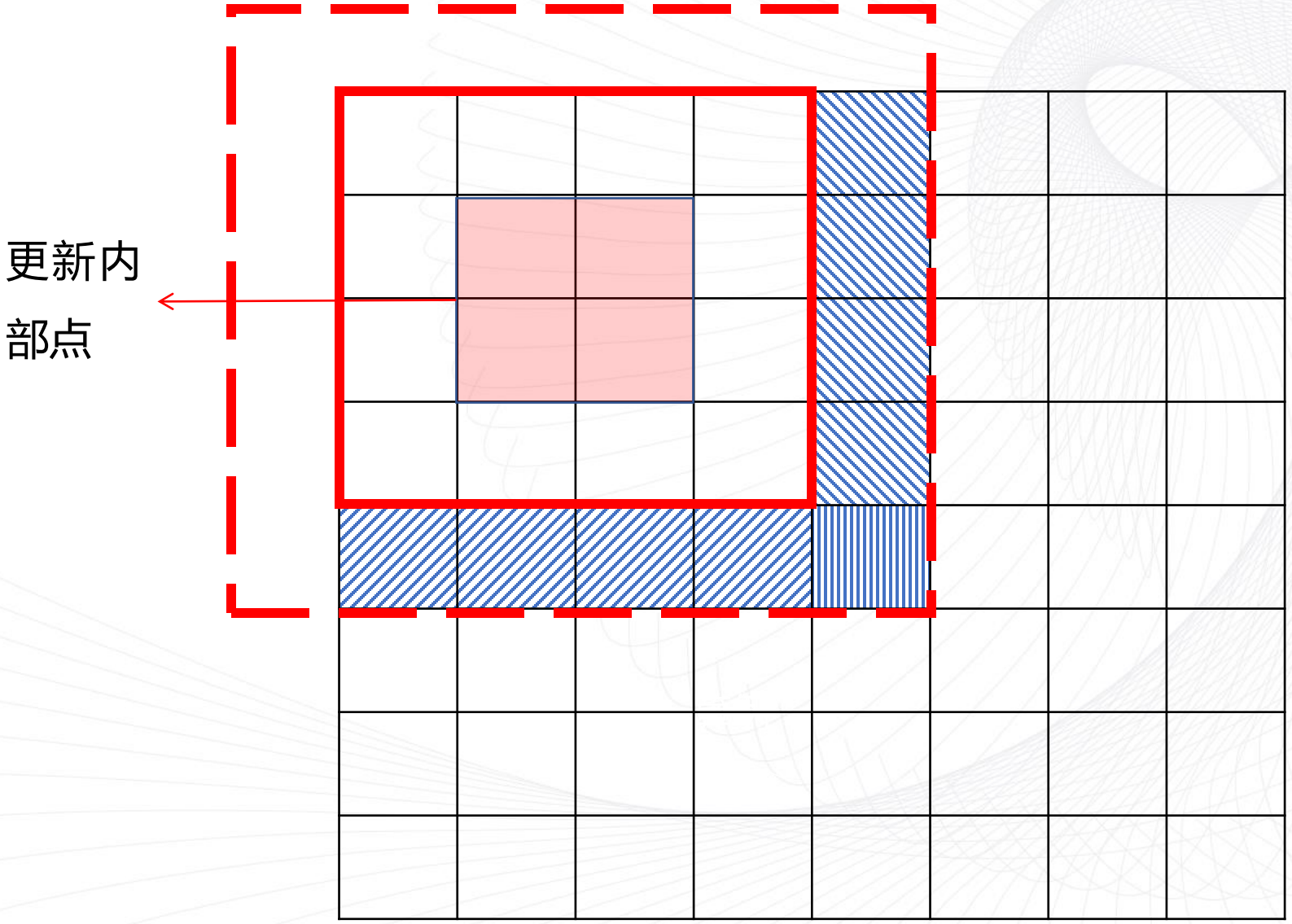
2.并行优化-使用128进程并行运行更新过程



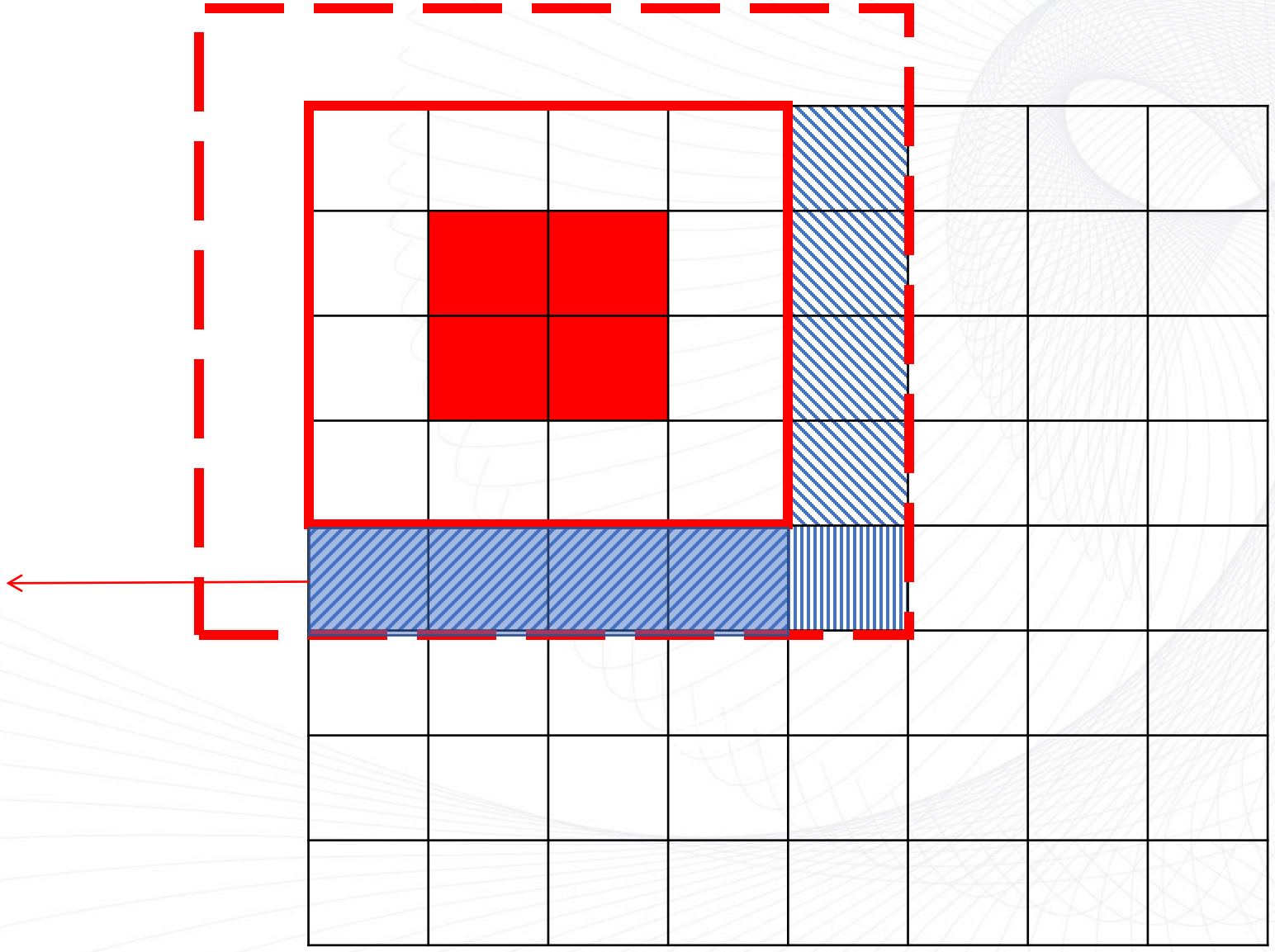
2.并行优化-使用128进程并行运行更新过程



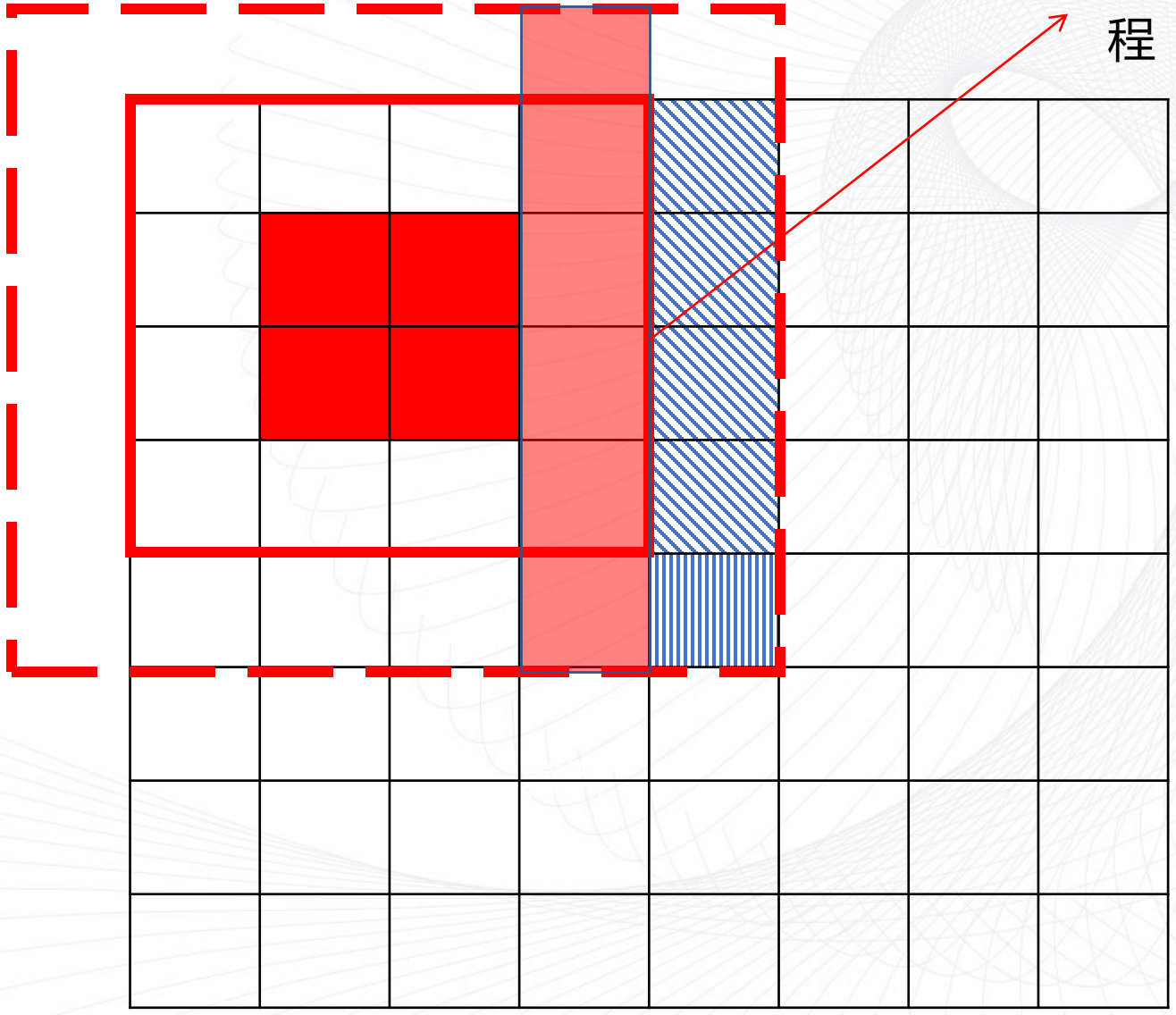
并行优化-使用128进程并行运行更新过程



2.并行优化-使用128进程并行运行更新过程

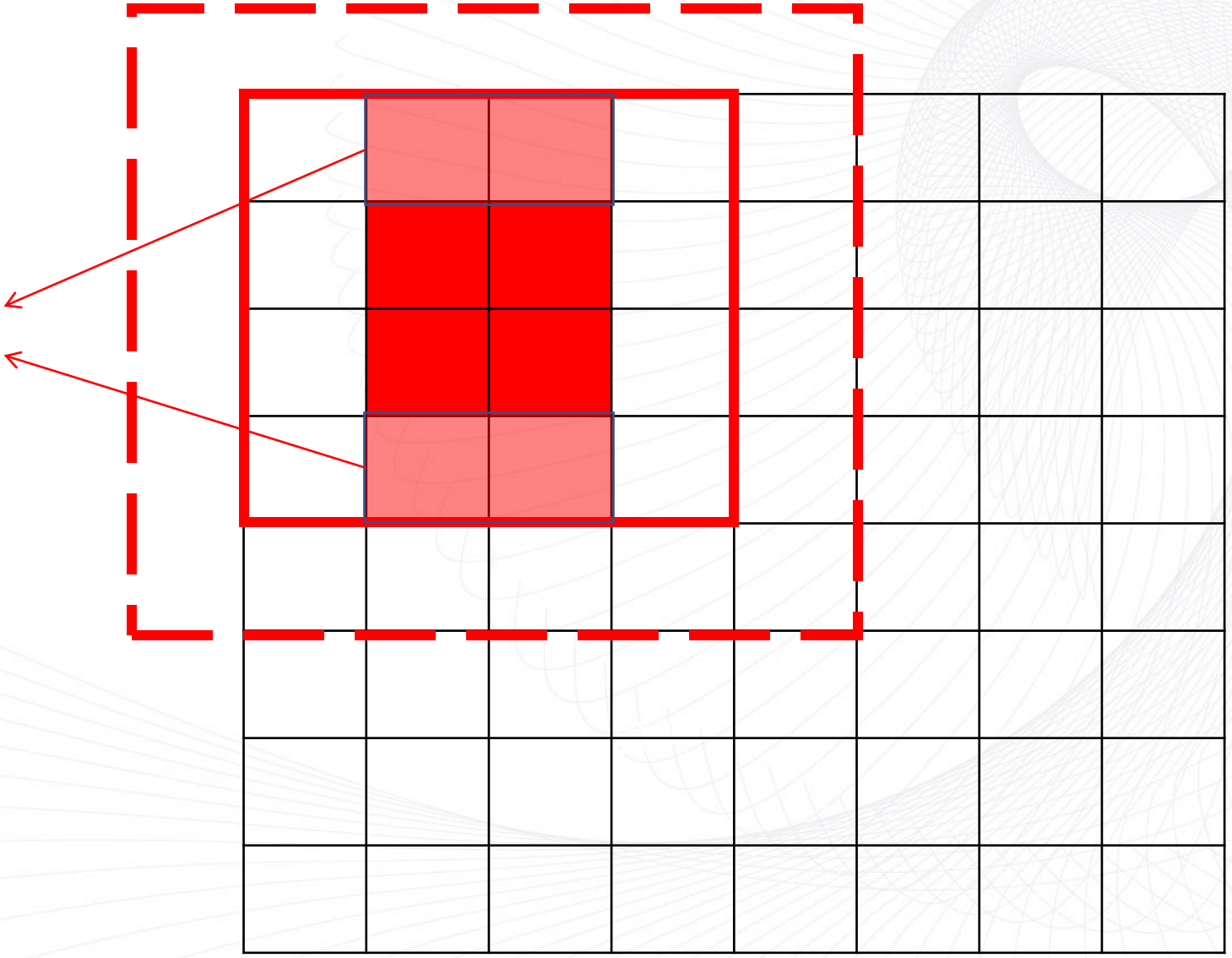


2.并行优化-使用128进程并行运行更新过程

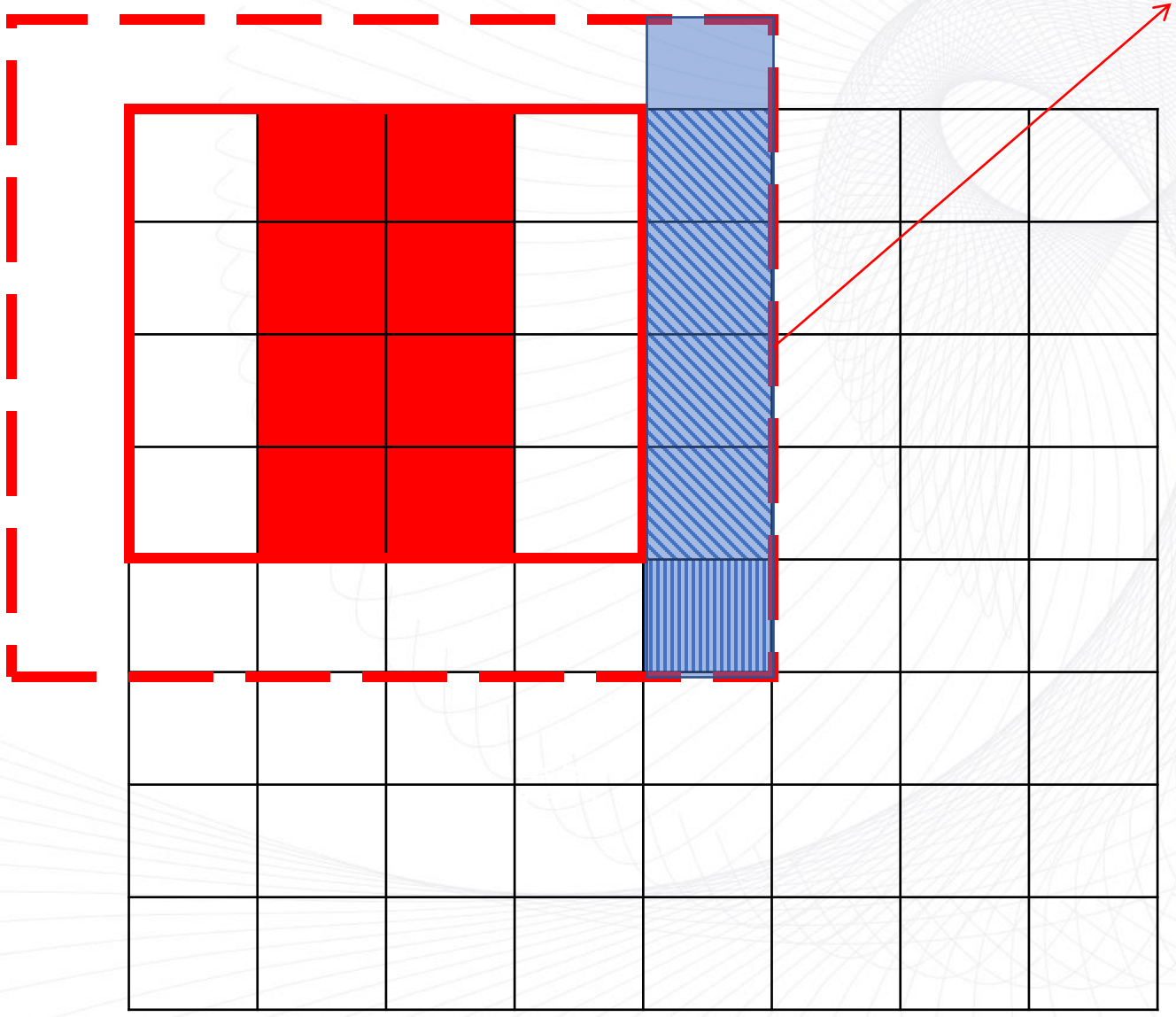


2.并行优化-使用128进程并行运行更新过程

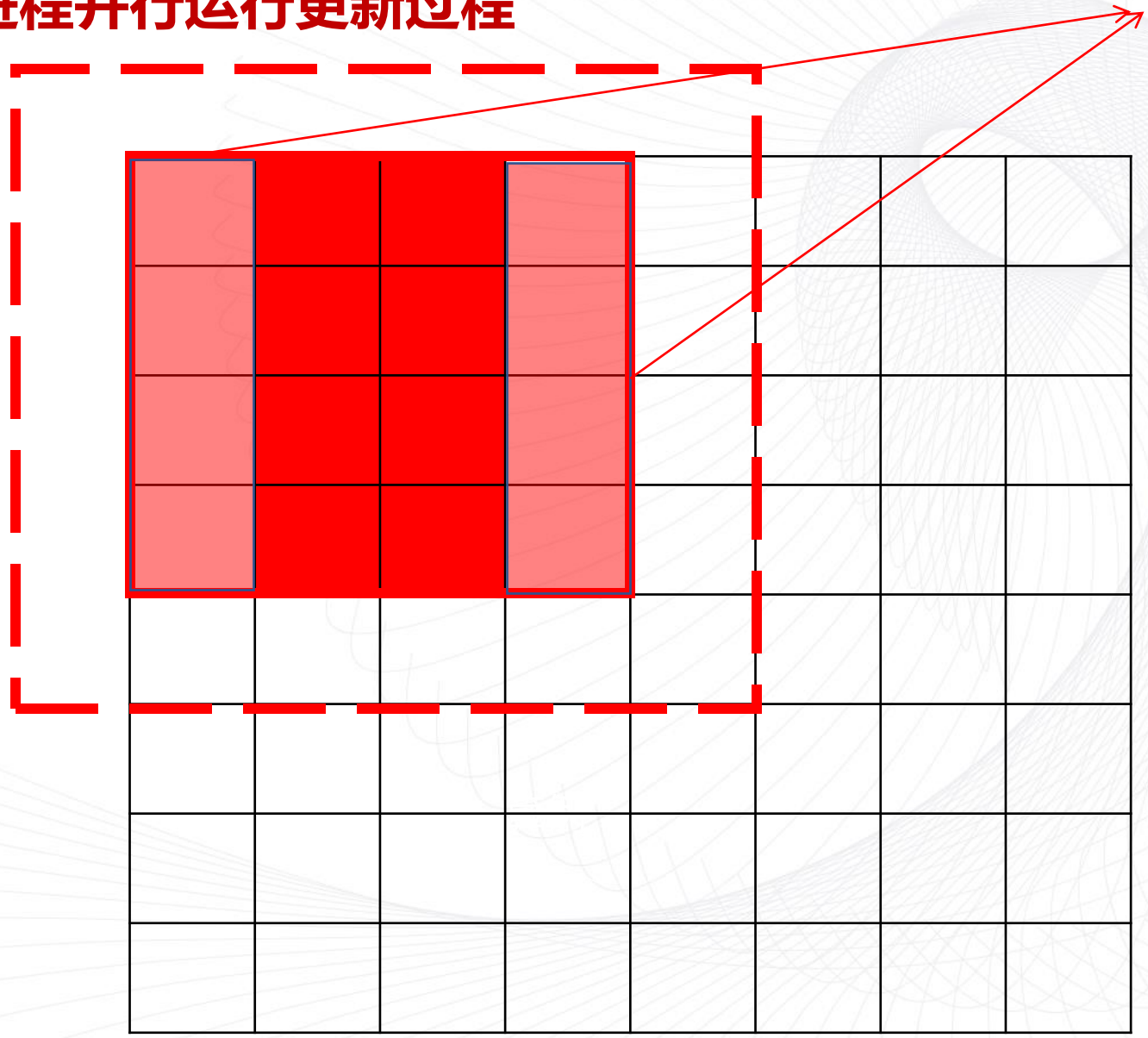
更新上下边界点



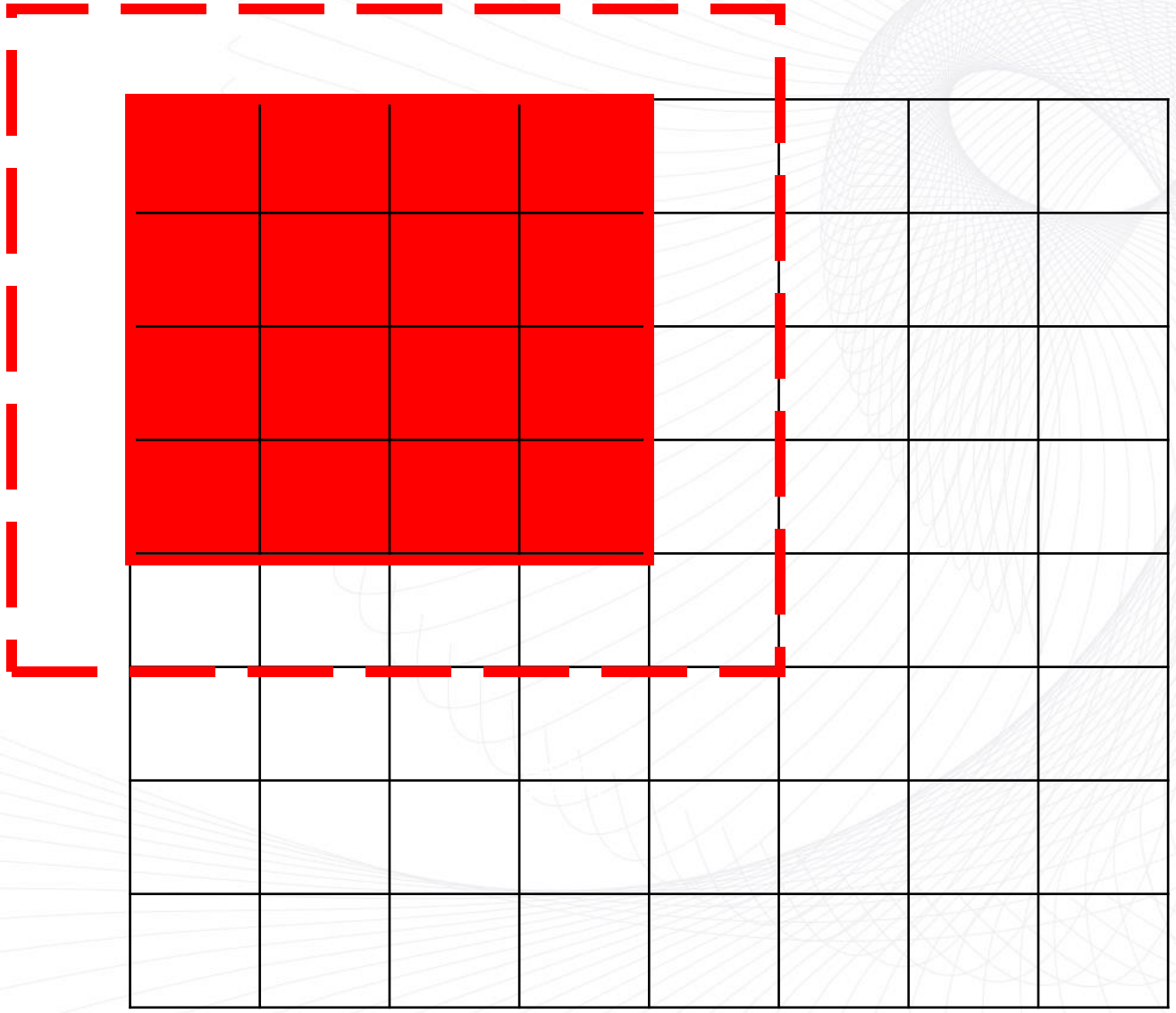
2.并行优化-使用128进程并行运行更新过程



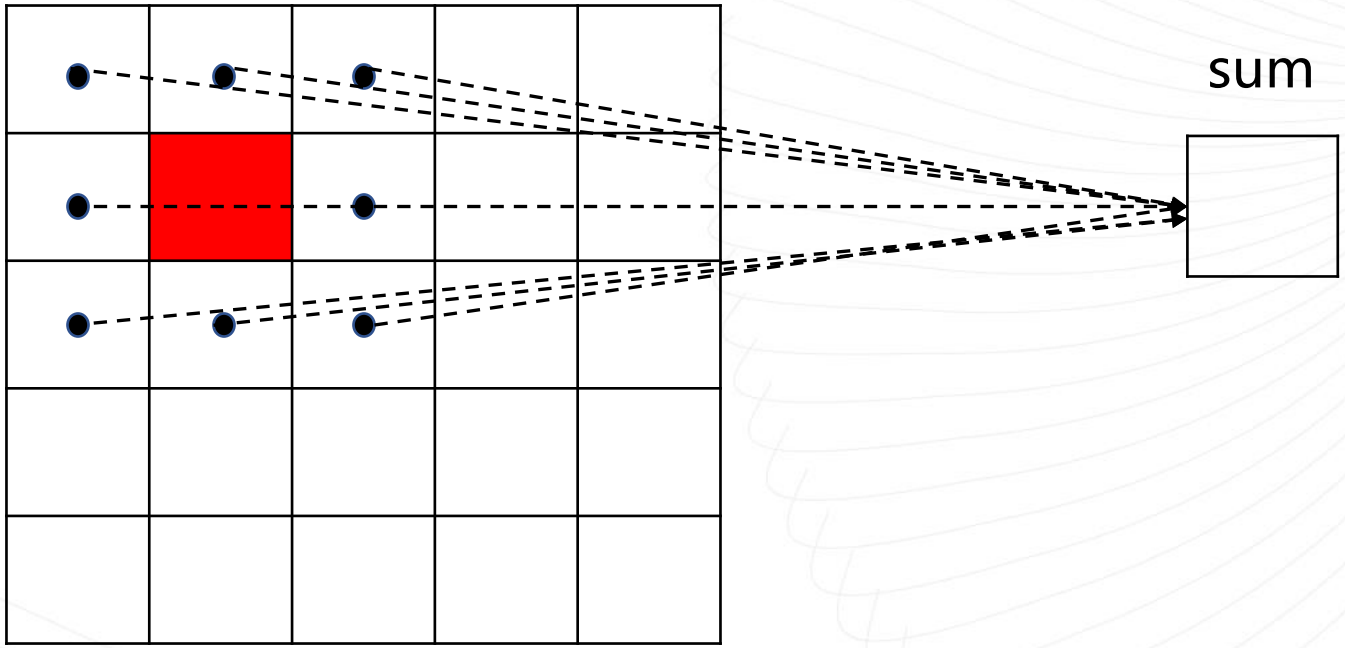
2.并行优化-使用128进程并行运行更新过程



2.并行优化-使用128进程并行运行更新过程



3.访存优化-改变更新时的循环次序

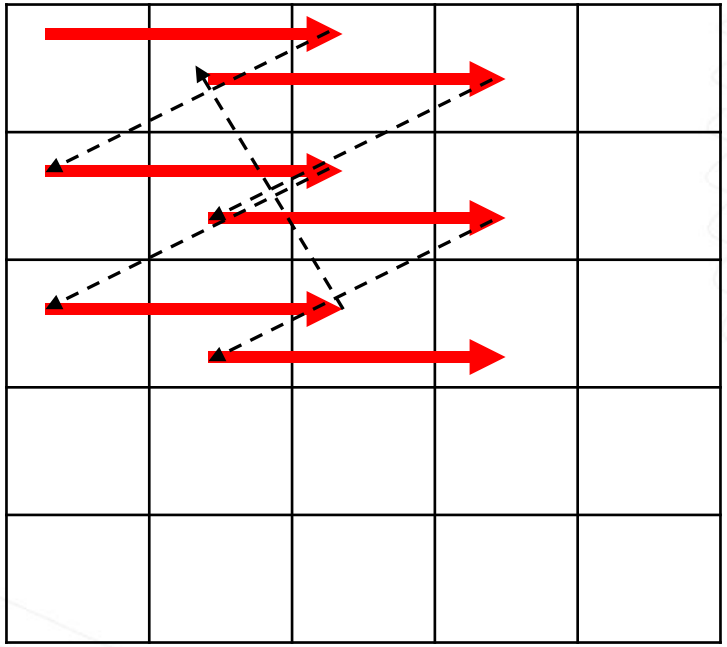


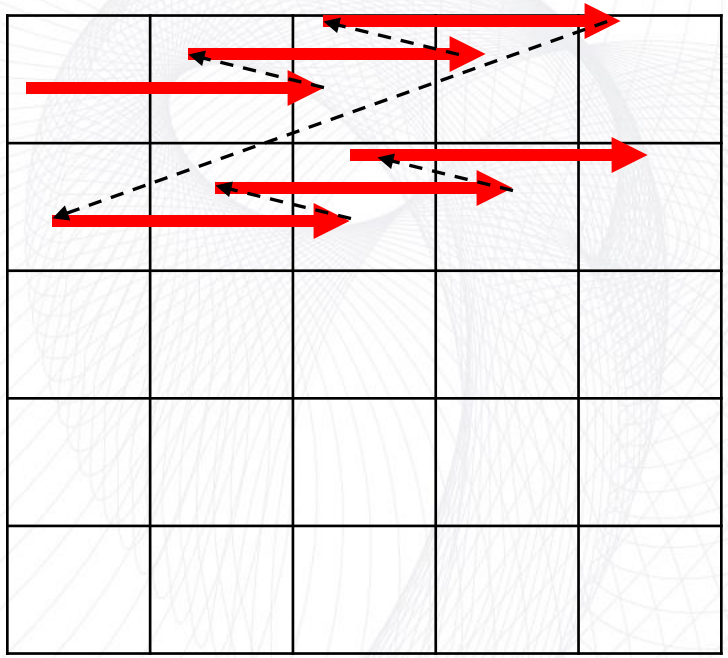
local_field

更新棋盘上1个点（计算sum）需要其周边的8个点的值，这是典型的 Stencil 计算，适当选取遍历次序对访存效率至关重要。

- 访存连续
- 重复利用元素

3.访存优化-改变更新时的循环次序





优化前：x→y→i→j

优化后：x→i→y→j

优化后的循环顺序，在访存的连续性和元素的重复利用上均优于原来的循环顺序

4.IO优化->使用MPI分布式数组进行MPI并行读写

```
void write_file(MPI_Comm COMM2d,int my_rows, int my_cols, int *start_indices, char *filepath, char **local_field,int right)
{
    MPI_File outfh;
    MPI_Datatype filetype,memtype;
    MPI_Status status;
    int gsizes[2] = {A, A+1};
    int lsizes[2] = {my_rows, my_cols};
    if(right==MPI_PROC_NULL){
        lsizes[1] += 1;
        for (int i = 0; i < my_rows + 2;++i){
            local_field[i][my_cols + 1] = '\n';
        }
    }

    MPI_Type_create_subarray( 2 , gsizes , lsizes , start_indices , MPI_ORDER_C , MPI_CHAR , &filetype);
    MPI_Type_commit(&filetype);
    MPI_File_open( COMM2d , filepath , MPI_MODE_CREATE|MPI_MODE_WRONLY , MPI_INFO_NULL , &outfh);
    MPI_File_set_view( outfh , 0 , MPI_CHAR , filetype , "native" , MPI_INFO_NULL);

    int memsizes[2] = {0, 0};
    memsizes[0] = my_rows + 2;
    memsizes[1] = my_cols + 2;
    start_indices[0]=start_indices[1]=1;
    MPI_Type_create_subarray( 2 , memsizes , lsizes , start_indices , MPI_ORDER_C , MPI_CHAR , &memtype);
    MPI_Type_commit(&memtype);
    MPI_File_write_all( outfh , &(local_field[0][0]) , 1 , memtype , &status);
    MPI_File_close(&outfh);
}
```

写入前的预处理

4.IO优化->使用MPI分布式数组进行MPI并行读写

```
void write_file(MPI_Comm COMM2d,int my_rows, int my_cols, int *start_indices, char *filepath, char **local_field,int right)
{
    MPI_File outfh;
    MPI_Datatype filetype,memtype;
    MPI_Status status;
    int gsizes[2] = {A, A+1};
    int lsizes[2] = {my_rows, my_cols};
    if(right==MPI_PROC_NULL){
        lsizes[1] += 1;
        for (int i = 0; i < my_rows + 2;++i){
            local_field[i][my_cols + 1] = '\n';
        }
    }

    MPI_Type_create_subarray( 2 , gsizes , lsizes , start_indices , MPI_ORDER_C , MPI_CHAR , &filetype);
    MPI_Type_commit(&filetype);
    MPI_File_open( COMM2d , filepath , MPI_MODE_CREATE|MPI_MODE_WRONLY , MPI_INFO_NULL , &outfh);
    MPI_File_set_view( outfh , 0 , MPI_CHAR , filetype , "native" , MPI_INFO_NULL);

    int memsizes[2] = {0, 0};
    memsizes[0] = my_rows + 2;
    memsizes[1] = my_cols + 2;
    start_indices[0]=start_indices[1]=1;
    MPI_Type_create_subarray( 2 , memsizes , lsizes , start_indices , MPI_ORDER_C , MPI_CHAR , &memtype);
    MPI_Type_commit(&memtype);
    MPI_File_write_all( outfh , &(local_field[0][0]) , 1 , memtype , &status);
    MPI_File_close(&outfh);
}
```

定义分布式数组文件类型，打开文件

4.IO优化->使用MPI分布式数组进行MPI并行读写

```
void write_file(MPI_Comm COMM2d,int my_rows, int my_cols, int *start_indices, char *filepath, char **local_field,int right)
{
    MPI_File outfh;
    MPI_Datatype filetype,memtype;
    MPI_Status status;
    int gsizes[2] = {A, A+1};
    int lsizes[2] = {my_rows, my_cols};
    if(right==MPI_PROC_NULL){
        lsizes[1] += 1;
        for (int i = 0; i < my_rows + 2;++i){
            local_field[i][my_cols + 1] = '\n';
        }
    }

    MPI_Type_create_subarray( 2 , gsizes , lsizes , start_indices , MPI_ORDER_C , MPI_CHAR , &filetype);
    MPI_Type_commit(&filetype);
    MPI_File_open( COMM2d , filepath , MPI_MODE_CREATE|MPI_MODE_WRONLY , MPI_INFO_NULL , &outfh);
    MPI_File_set_view( outfh , 0 , MPI_CHAR , filetype , "native" , MPI_INFO_NULL);

    int memsizes[2] = {0, 0};
    memsizes[0] = my_rows + 2;
    memsizes[1] = my_cols + 2;
    start_indices[0]=start_indices[1]=1;
    MPI_Type_create_subarray( 2 , memsizes , lsizes , start_indices , MPI_ORDER_C , MPI_CHAR , &memtype);
    MPI_Type_commit(&memtype);
    MPI_File_write_all( outfh , &(local_field[0][0]) , 1 , memtype , &status);
    MPI_File_close(&outfh);
}
```

只写入local_field中本进程负责更新的部分，不包括ghost点。

4.其他优化

- 使用_mm_malloc 和_mm_free替代malloc和free，在分配内存时按照字节对齐。

```
char **local_field = (char **)_mm_malloc((my_rows+2) * sizeof(char *),ALIGNMENT);
int local_field_size = (my_rows+2) * (my_cols+2);
char *local_field_data = (char *)_mm_malloc(local_field_size * sizeof(char),ALIGNMENT);
```

- 重新梳理更新规则，减少条件分支

```
for (int y = 2; y < my_cols;++y){
    if (local_matrix[x][y] == '1') {
            next_local_matrix[x][y] = '0'+(sum[y] == 2 || sum[y] == 3);
    } else {
            next_local_matrix[x][y] = '0'+(sum[y] == 3);

    }
}
```

程序运行结果-test_pattern,迭代1000次

优化前	112259.568ms
优化后	117.301ms~228.368ms
加速比	957~492

```
[sca3213@ln121%bscc-a5 original]$ tail -n 1 ./slurm-1758105.out&&grep -o 1 ./test_output |wc -l
112259.568000 ms
68231
```

```
Submitted batch job 1758089
the time is
117.301ms
68231
```

```
Submitted batch job 1758155
the time is
228.368ms
68231
```

感谢观看

THANKS FOR WATCHING