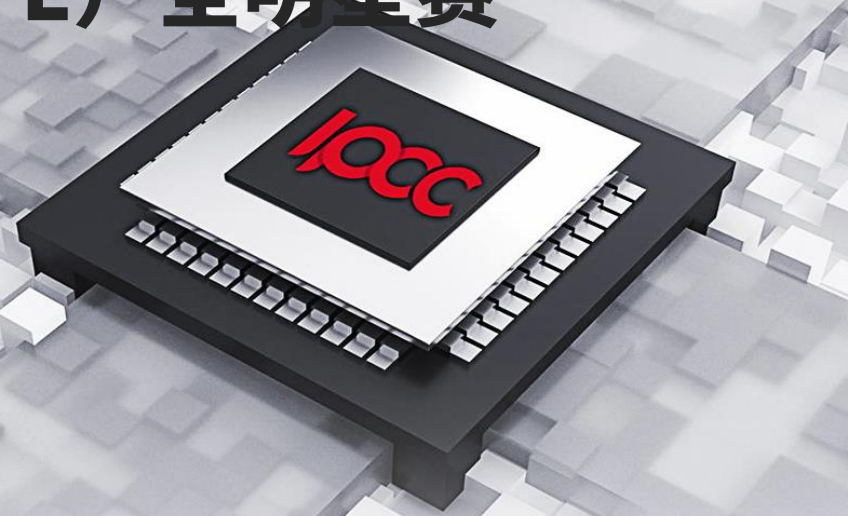
Association for Computing Machinery

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：黄瀚　2022年5月22日

目录 CONTENTS

01. 个人简介
02. 应用程序运行的硬件环境和软件环境
03. 应用程序的代码结构
04. 优化方法
05. 程序运行结果
06. 一些思考

个人简介

黄瀚

中山大学超算队成员

ASC22 & ISC22 参赛队员

应用程序运行的硬件环境和软件环境

硬件环境：AMD EPYC 7452 32-Core Processor，有超线程

软件环境：使用modulefile中的gcc/10.2.0

应用程序运行的代码结构

我们在这里使用的代码结构如下所示：

- main.cpp: 整个康威生命游戏的主函数
- main.hpp: 用于定义使用的串行和并行康威生命游戏模拟方法
- seq_conway.cpp: 实现串行康威生命游戏模拟方法
- seq_conway.hpp: 存放串行时对于每行进行处理的API，方便其它函数调用
- omp_conway.cpp: 实现OpenMP并行的模拟方法
- utils.hpp: 存放一些工具函数，比如交换指针的函数

优化方法 0 编译参数优化

```
[sca3801@ln121%bscc-a5 myconway]$ g++ *.cpp -o Conway  -std=c++11  -fopenmp -O3 -funroll-loops -funswitch-loops
```

1. -O3：开启-O3优化，比如说开启了一些自动向量化之类的
2. -funroll-loops：优化器自己去找最优的循环展开因子
3. -fopenmp：使用openmp多线程优化

优化方法 1 访存优化

1. 使用char类型来存储地图：在给定的baseline代码中，原本用于储存地图的是int类型，但是这个地图里面的值只可能为0或1，所以用char类型更合适

```
char* read_in_world(int width, int height)
{
    int size = width * height;
    char* world = (char*)aligned_alloc(64, size);
    for(int i=0;i<size;i++)
    {
        world[i]=0;
    }
    FILE* file = fopen("test_pattern", "r");
    char line[256];
    int x = -1, y = -1;
```

2. 使用双buffer，在每一次迭代结束之后不用memcpy，直接交换两个指针就好，节省时间

```
if(i==0)
    memcpy(grid, buf, size);
swap_ptr((void**)&grid, (void**)&buf);
```

3. 不再使用dx和dy这两个数组，直接手搓出来8个邻居

优化方法 2 减少分支

1. 计算周围8邻居的时候，直接加起来

```
char cell = grid[i_north + x_west] + grid[i_north] +
            grid[i_north + x_east] + grid[i_row + x_west] +
            grid[i_row + x_east] + grid[i_south + x_west] +
            grid[i_south] + grid[i_south + x_east];
```

2. 判断当前的cell后一个迭代的值的时候，使用真值式求出来

```
cell = (cell == 3) | ((cell == 2) & grid[idx]);
buf[idx] = cell;
```

优化方法 3 多线程优化

1. 划分任务：把当前的任务按照行划分成若干块，每一个线程负责执行这一块任务的所有操作

```
int rows_per_thread = (height - 2  + threads - 1) / threads;
int cells_per_thread = rows_per_thread * width;
if (cells_per_thread < cache_line_size) {
    rows_per_thread = (cache_line_size + width - 1) / width;
}

threads = (height + rows_per_thread - 1) / rows_per_thread;

    #pragma omp parallel num_threads(threads) default(none) \
    shared(width, height, gens, rows_per_thread, threads) firstprivate(grid, buf)
    {
        int tid = omp_get_thread_num();
        int y_start = tid * rows_per_thread;
        int y_end = y_start + rows_per_thread;
```

2. 执行完这些操作之后，使用Barrier同步计算结果

```
if (tid == 0) {
    for (int i = 0; i < gens; i++) {
        for (int y = 1; y < y_end; y++) {
            int y_north = y - 1;
            int y_south = y + 1;
            cpu_seq_row(grid, buf, width, y, y_north, y_south);
        }

        swap_ptr((void**)&grid, (void**)&buf);
        #pragma omp barrier
    }
}
else if (tid == threads - 1) {
    for (int i = 0; i < gens; i++) {
        for (int y = y_start; y < y_end && y < height - 1; y++) {
            int y_north = y - 1;
            int y_south = y + 1;
            cpu_seq_row(grid, buf, width, y, y_north, y_south);
        }
        swap_ptr((void**)&grid, (void**)&buf);
        #pragma omp barrier
    }
}
else {
    for (int i = 0; i < gens; i++) {
        for (int y = y_start; y < y_end && y < height-1; y++) {
            int y_north = y - 1;
            int y_south = y + 1;
            cpu_seq_row(grid, buf, width, y, y_north, y_south);
        }
        swap_ptr((void**)&grid, (void**)&buf);
        #pragma omp barrier
    }
}
```

程序运行结果

- Baseline：

```
109064.966000 ms
[sca3801@ln121%bscc-a5 pre2]$
```

- 优化后的最好结果：（加速比约为627倍）

```
[sca3801@ln121%bscc-a5 myconway]$ cat run1.log
run time is 174.894000ms
```

- 调整OMP_PROC_BIND=true之后的结果（加速比约为3408倍）

```
[sca3801@ln121%bscc-a5 myconway]$ cat run1.log
run time is 32.289000ms
```

- 验证正确性：

```
[sca3801@ln121%bscc-a5 myconway]$ diff test_output output_CHAR
[sca3801@ln121%bscc-a5 myconway]$
```

一些思考

- 还可以加上向量化去优化
- 优化的最厉害的就是Hashlife
- 之前看到过四叉树的实现，但是代码量太大，无法短时间内复现
- 一个参考实现：https://github.com/cmqten/accelerated-game-of-life

感谢观看

THANKS FOR WATCHING