Клеточный автомат Игра «Жизнь»

ЛАБОРАТОРНАЯ РАБОТА

http://www.kimrt.ru
Онлайн-курс
Суперкомпьютерные технологии в задачах моделирования

Игра «Жизнь» (Conway's Game of Life) — клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 [1]. До этого физик Джон фон Нейман предложил нечто похожее. Во времена космической гонки он думал, как колонизировать Марс. Поверхность красной планеты состоит из оксида железа. Фон Нейман предложил создать роботов, которые бы высадились на поверхность планеты и начали расщеплять материал у себя под ногами на кислород и железо. Кислород бы заполнил атмосферу для будущих колонизаторов, а из железа роботы бы делали свои собственные копии. Модель фон Неймана сложная, каждый робот в ней имеет до 20 состояний. Предсказать поведение такой системы невероятно сложно. Конвей упростил модель фон Неймана до следующих правил. Представьте себе двумерную вселенную, состоящую из клеток. Некоторые клетки закрашены — тогда мы говорим, что в них есть жизнь. У каждой клетки есть 8 клеток-соседей [2]. Введём некоторые **правила**:

1. если у живой клетки меньше 2 соседей, она умирает от одиночества;
2. если у живой клетки 2 или 3 соседа, она продолжает жить;
3. если у клетки более 3 соседей, она умирает от перенаселения;
4. если у неживой клетки ровно 3 соседа, происходит размножение и клетка становится живой.

Тем не менее, из таких простых правил всё равно рождаются очень сложные, непредсказуемые структуры. Почти никогда нельзя предсказать, вымрет ли система или стабилизируется и за сколько шагов это произойдёт. Эта простая модель очень помогла развитию таких наук, как математика и программирование, и даже биология с химией. Другое обстоятельство, которым «Жизнь» необычна, состоит в том, что игрок вмешивается в процесс игры всего только раз: задавая начальные условия. После чего игровое поле начинает буквально жить само по себе, подчиняясь только озвученным выше правилам. Никто, конечно, не запрещает вам вмешаться в игру и по ходу её, добавив или удалив фишки, но интересней всего оказывается простое наблюдение [3]! Один из возможных вариантов реализации программы представлен в листинге 1 [4]. Такой подход программирования является удобным в плане чтения кода и

понимания алгоритма. Но для распараллеливания может оказаться неудачным вариантом. Множественный вызов функций внутри цикла сильно замедляет работу программы, особенно в тех случаях, когда речь идет о миллиардах итераций. Можно конечно попробовать добавить *inline* перед такими функциями, но это лишь рекомендация для компилятора, которая срабатывает не во всех случаях. Также этот код можно доработать, сделав случайную генерацию начального состояния матрицы произвольного размера, определяемого пользователем. Состояние решетки на каждом временном шаге можно записывать в файлы. Для больших матриц вместо вывода в консоль логичнее сделать визуализацию. Причем она может быть двух видов: отрисовка в процессе расчета или визуализация отдельной программой на основе файлов с результатами расчета. Если важна скорость расчета, то лучше просто сохранять данные в файл и в дальнейшем при необходимости их визуализировать. Оптимальнее записывать данные в файл не на каждом временном шаге, а накапливать их для большого числа шагов, сократив количество записей, так как это дорогостоящая в плане времени операция.

Листинг 1. Реализация игры жизнь на C++

```
#include <iostream>
#include <vector>

// ----------------------------------------------------------

using Row = std::vector<int>;
using Cells = std::vector<Row>;

// ----------------------------------------------------------

Cells board = {
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
      {0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
      {0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
      {0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
      {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1},
};

int numRows = 10;
int numCols = 20;

// ----------------------------------------------------------
```

```
int getNeighbor(int row, int col, Cells& board) {
     // use modulus to get wrapping effect at board edges
     return board.at((row + numRows) % numRows).at((col + numCols)
% numCols);
}

int getCount(int row, int col, Cells& board) {
     int count = 0;
     std::vector<int> deltas{ -1, 0, 1 };
     for (int dc : deltas) {
          for (int dr : deltas) {
               if (dr || dc) {
                    count += getNeighbor(row + dr, col + dc,
board);
               }
          }
     }
     return count;
}



void showCell(int cell) {
     std::cout << (cell ? "*" : " ");
}

void showRow(const Row& row) {
     std::cout << "|";
     for (int cell : row) { showCell(cell); }
     std::cout << "|\n";
}

void showCells(Cells board) {
     for (const Row& row : board) { showRow(row); }
}

int tick(Cells& board, int row, int col) {
     int count = getCount(row, col, board);
     bool birth = !board.at(row).at(col) && count == 3;
     bool survive = board.at(row).at(col) && (count == 2 || count
== 3);
     return birth || survive;
}

void updateCells(Cells& board) {
     Cells original = board;
     for (int row = 0; row < numRows; row++) {
          for (int col = 0; col < numCols; col++) {
               board.at(row).at(col) = tick(original, row, col);
          }
     }
}
```

```
int main() {
     for (int gen = 0; gen < 20; gen++) {
          std::cout << "\ngeneration " << gen << ":\n";
          showCells(board);
          updateCells(board);
     }
}
```

Реализация программы без пользовательских функций представлена в листинге 2. Такой вариант кода удобнее распараллеливать. Но у него также есть недостатки. Во-первых, используются статические массивы. Чтобы пользователь мог задавать размеры решетки, следует использовать динамические массивы или вектора. Во-вторых, для хранения информации об игровом поле используется двухмерный массив. Соответсвенно, перебор всех элементов массива осуществляется через вложенные циклы. Используя математические трюки, можно хранить информацию о двумерном или трехмерном игровом поле в одномерном массиве. Это позволит в дальнейшем более эффективно распараллелить программу, к примеру, с использованием библиотеки OpenMP простым добавлением *#pragma* для распараллеливания цикла. Такой подход эффективно работает на большом числе итераций. Другие варианты, в том числе и на других языках описаны в [5].

Листинг 2. Реализация игры жизнь на C++ без пользовательских функций

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <random>
#define SIZEX 50
#define SIZEY 1000

int main ()
{
    setlocale(LC_ALL,"Russian");
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(1, 10000);
    int tmp[SIZEX][SIZEY], world[SIZEX][SIZEY];
    int x, y, n=0;//координаты и количество соседей
    FILE * f;
    f = fopen("test.txt","wt");

     //начальная генерация
    for (y=0;y<=SIZEY-1;y++)
          for (x=0;x<=SIZEX-1;x++){
```

```
            int num = dis(gen);
            world [x][y] = num % 2;
        }

    int d, i;
    printf("Количество поколений: ");
    scanf("%i", &d);
    for (i=1;i<=d;i++)
    {
        system("cls");
        for (y=1;y<=SIZEY-2;y++)
          for (x=1;x<=SIZEX-2;x++)
              tmp[x][y] = world[x][y];
        for (y=1;y<=SIZEY-2;y++)
        {
            for (x=1;x<=SIZEX-2;x++)
            {
                if (tmp[x][y] == 0)
                {
                    if (tmp[x-1][y-1] == 1) n++;
                    if (tmp[x][y-1] == 1) n++;
                    if (tmp[x+1][y-1] == 1) n++;
                    if (tmp[x+1][y] == 1) n++;
                    if (tmp[x+1][y+1] == 1) n++;
                    if (tmp[x][y+1] == 1) n++;
                    if (tmp[x-1][y+1] == 1) n++;
                    if (tmp[x-1][y] == 1) n++;
                    if (n == 3) world[x][y] = 1;
                    n = 0;
                } else
                {
                    if (tmp[x-1][y-1] == 1) n++;
                    if (tmp[x][y-1] == 1) n++;
                    if (tmp[x+1][y-1] == 1) n++;
                    if (tmp[x+1][y] == 1) n++;
                    if (tmp[x+1][y+1] == 1) n++;
                    if (tmp[x][y+1] == 1) n++;
                    if (tmp[x-1][y+1] == 1) n++;
                    if (tmp[x-1][y] == 1) n++;
                    if ((n == 2) || (n == 3))
                    {
                        world [x][y] = 1;
                        n = 0;
                    } else
                    {
                        world [x][y] = 0;
                        n = 0;
                    }
                }
            }
        }

    }
```

```
        for (y=0;y<=SIZEY-1;y++) {
            for (x=0;x<=SIZEX-1;x++)
                 fprintf( f, "%d ", world[x][y] );
            fprintf( f, "\n");
        }

    fclose(f);
    system("pause");
    return 0;
}
```

Задание. Напишите оптимальный код, который в дальнейшем можно будет быстро и эффективно распараллелить, для двумерной игры жизнь. Используйте периодические граничные условия. У пользователя должна быть возможность задавать размеры решетки. Реализуйте хранение информации об игровом поле в одномерном динамическом массиве или векторе. Разбейте временные интервалы на периоды, через которые будет выполняться запись данных в файлы (в один файл записывается информация об игровом поле не за один временной шаг, а за период из *N* временных шагов). Протестируйте программу и определите, как меняется время расчета для разных размеров решеток. Для каждого вычислительного эксперимента делайте усреднение временного показателя на основе пяти испытаний. Постройте графики или диаграммы, например, в Excel, напишите краткий отчет о проделанной работе с выводами.

http://www.kimrt.ru/index/course_stm/0-24

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. https://neerc.ifmo.ru/wiki/index.php?title=%D0%98%D0%B3%D1%80%D0%B0_%C2%AB%D0%96%D0%B8%D0%B7%D0%BD%D1%8C%C2%BB
2. https://pikabu.ru/story/programmiruem_zhizn_6389491
3. http://knoppix.ru/sentinel/021017.html
4. http://rosettacode.org/wiki/Conway%27s_Game_of_Life#Simple_Without_Classes
5. http://rosettacode.org/wiki/Conway%27s_Game_of_Life