НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Курсовий проект

із дисципліни «Математичне моделювання»

на тему

«Cell evolution modeling»

Виконав:	Керівник:
студент групи КМ-81	
Чебан Р.С.	Норкін Б. В.

Стек технологій

Програма була написана на мові C++, скомпільована та запущена на операційній системі Windows. В програмі було використано бібліотеку для мультипроцисенгу <omp.h>.

Теоретичні відомості

Алгоритм пульсації на прикладі: розглянемо задачу паралельного обчислення значення функції від мультимножини значень, записаних у вершинах орієнтованого сильно-зв'язкового графа.

Обчислення виконується автоматами, які перебувають в вершинах графа. Автомат має локальну інформацію про графа: він «знає» тільки про дугах, що виходять з вершини, в якій він знаходиться, але «не знає», куди (в які вершини) ці дуги ведуть. Автомати обмінюються повідомленнями,

переданими по дугам графа, які грають роль каналів передачі повідомлень.

Обчислення ініціюється повідомленням, що приходять ззовні в автомат виділеної початкової вершини графа. Цей же автомат в кінці роботи посилає зовні обчислене значення функції. Для вирішення цього завдання пропонуються два алгоритму. Перший алгоритм виконує дослідження графа, метою якого є розмітка графа за допомогою зміни станів автоматів в вершинах. Така розмітка використовується другим алгоритмом, який і виробляє обчислення значення тієї чи іншої функції. Це обчислення

засноване на алгоритмі пульсації: спочатку від автомата початкової вершини по всьому графу поширюються повідомлення-питання, які повинні досягти кожної вершини, а потім від кожної вершини «у зворотний бік» до початкової вершини рухаються повідомлення-відповіді. Алгоритм пульсації, по суті, обчислює агрегатні функції, для яких значення функції від об'єднання мультимножин обчислюється за

значеннями функції від цих мультимножин. Однак показано, що будь-яка функція F (x) має агрегатний розширення, тобто може бути обчислена як H (G (x)), де G агрегатна функція. Зауважимо, що розмітка графа не

залежить від тієї функції, яка буде обчислюватися. Це означає, що розмітка графа виконується один раз, після чого може багаторазово використовуватися для обчислення різних функцій.

Оскільки автомати в вершинах графа працюють паралельно, як розмітка графа, так і обчислення функції виконуються паралельно.

Це перша особливість роботи.

Друга особливість - обчислення виконуються на динамічно мінливому графі: його дуги можуть зникати, з'являтися або міняти свої кінцеві вершини.

На зміни графа накладаються такі мінімальні обмеження, які дозволяють вирішувати цю задачу за обмежений час. Наводиться оцінка часу роботи обох пропонованих алгоритмів.

Інтерфейс OpenMP задуманий як стандарт для програмування на масштабованих SMP-системах (SSMP, ccNUMA, etc.) в моделі загальної пам'яті (shared memory model). У стандарт OpenMP входять специфікації набору директив компілятора, процедур і змінних середовища.

Розробкою стандарту займається організація OpenMP ARB (ARchitecture Board), до якої увійшли представники найбільших компаній - розробників SMP-архітектур і програмного забезпечення. Специфікації для мов Fortran і C / C ++ з'явилися відповідно в жовтні 1997 року і жовтні 1998 року. Відкрито список розсилки для публічного обговорення OpenMP (omp@openmp.org).

До появи OpenMP не було відповідного стандарту для ефективного програмування на SMP-системах.

POSIX-інтерфейс для організації ниток (Pthreads) підтримується широко (практично на всіх UNIX-системах), проте з багатьох причин не підходить для практичного

паралельного програмування:

OpenMP можна розглядати як високорівневу надбудову над Pthreads (або аналогічними бібліотеками ниток).

Багато постачальників SMP-архітектур (Sun, HP, SGI) в своїх компіляторах

підтримують спецдіректіви для розпаралелювання циклів. Однак ці набори директив, як правило, 1) вельми обмежені; 2) несумісні між собою.

В результаті чого розробникам доводиться распараллеливать додаток окремо для кожної платформи. OpenMP є багато в чому узагальненням і розширенням згаданих наборів директив.

Опис проекту

Проект складається з 2 файлів: cell.cpp і covid.cpp. Програма на C++ моделює поле 10x10 (для зручності) і випадковим чином заповнює поле звичайною клітиною або клітиною коронавірусу. Результат кожної епохи виводиться на екран, з адопомогою цього виводу можна чітко порівняти результати епох, побачити нові клітини, відстежити померлі клітини і тд. Підключення бібліотеки <omp.h> аргументується зменшенням кількості часу виконання.

Для клітини:

- Жива клітина, навколо якої <2 живих клітин, вмирає від самотності
- Жива клітина, навколо якої є 2 або 3 живих клітин, виживає.
- Жива клітина, навколо якої> 3 живих клітин, вмирає від перенаселення.

Для клітини коронавірусу:

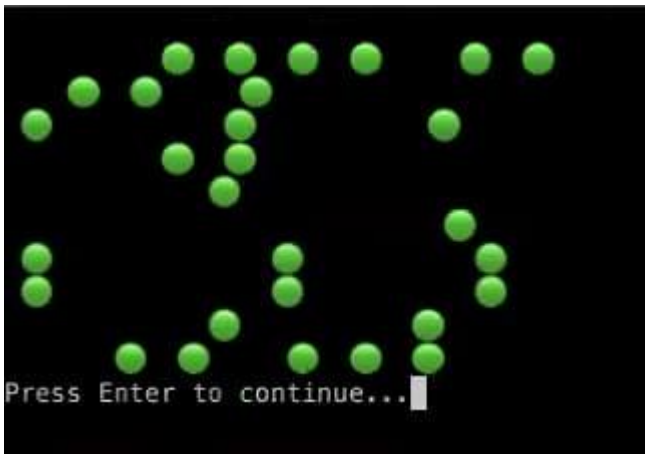
- Жива клітина, навколо якої <2 живих клітин, вмирає від самотності
- Жива клітина, навколо якої є 2 або більше живих клітин, виживає.

Скріншоти виконання програми

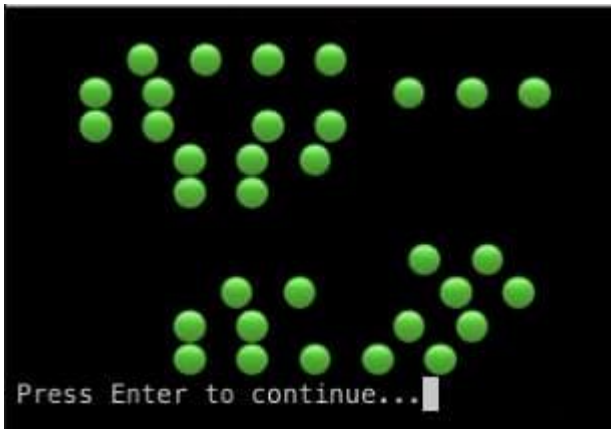
Файл cell.cpp

Епоха - 1

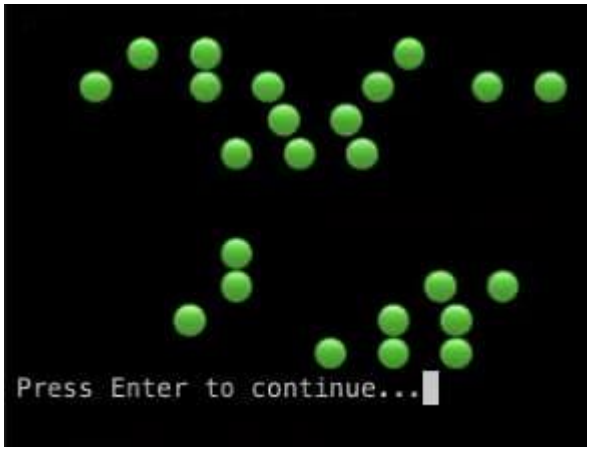
Епоха - 2



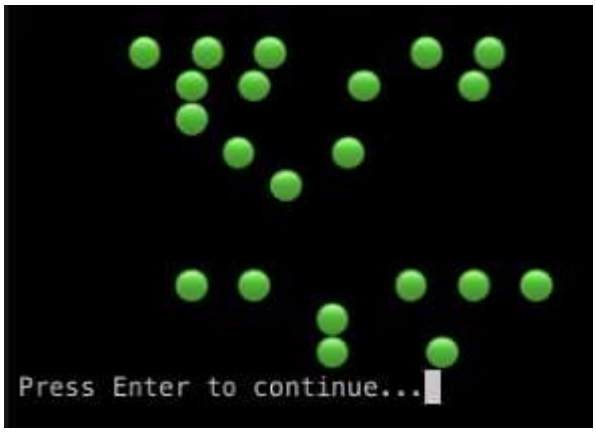
Епоха - 3



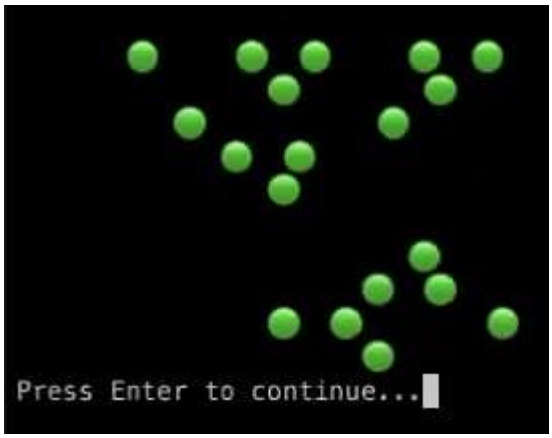
Епоха - 4



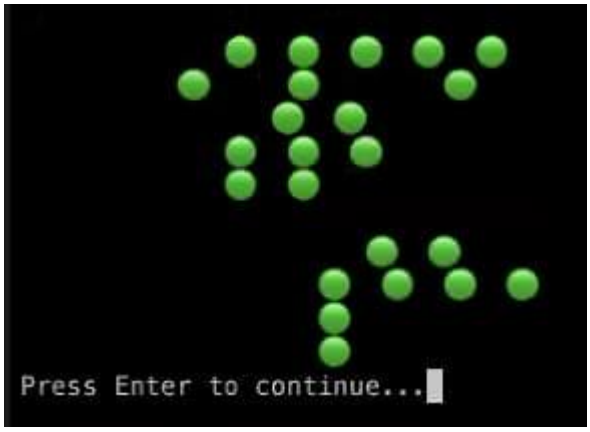
Епоха - 5



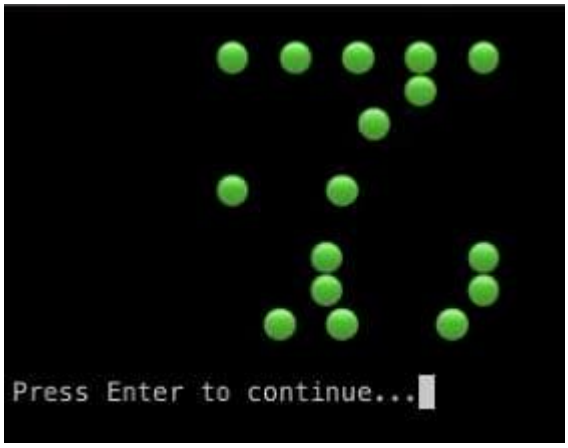
Епоха - 6



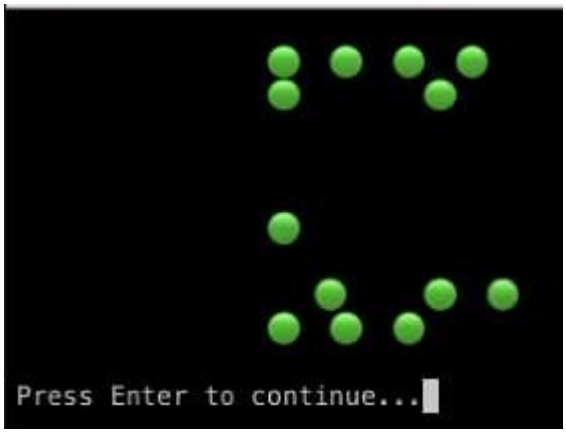
Епоха - 7



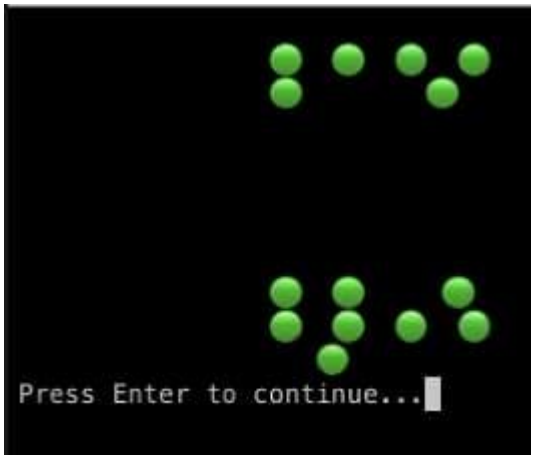
Епоха - 8



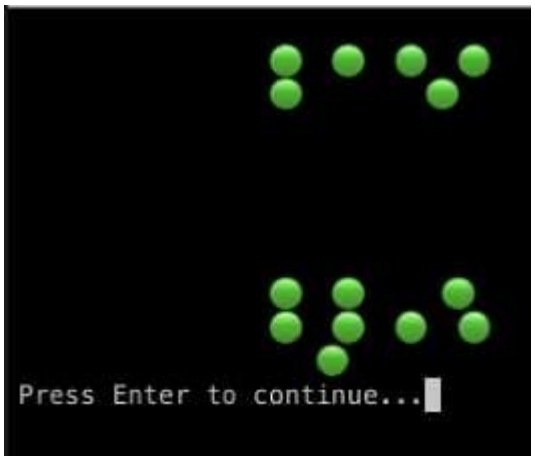
Епоха - 9



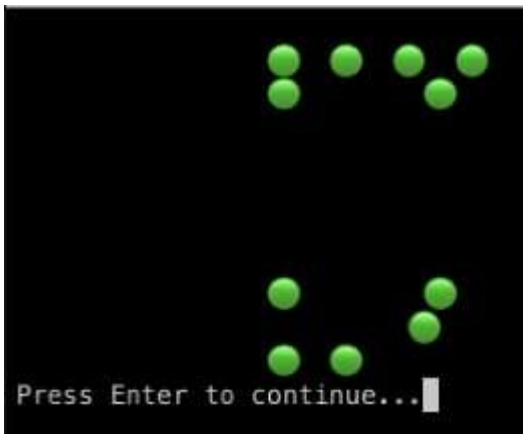
Епоха - 10



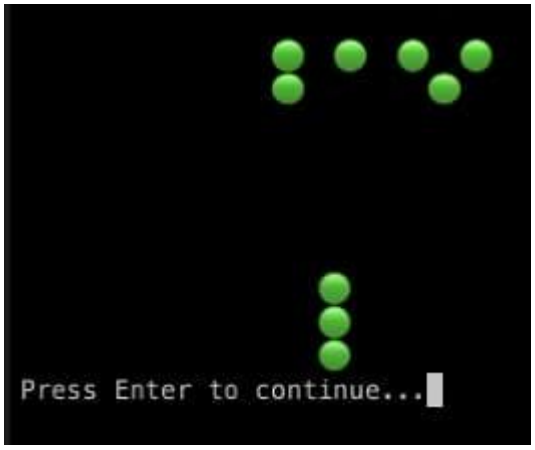
Епоха - 11



Епоха - 12

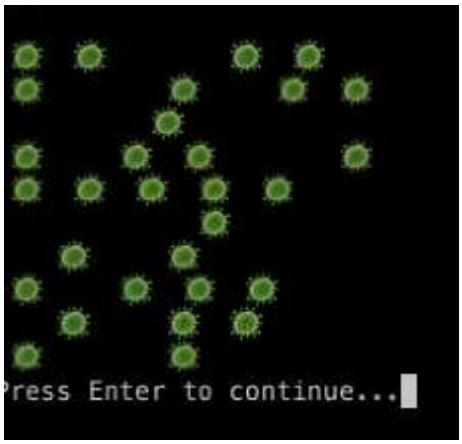


Епоха - 13

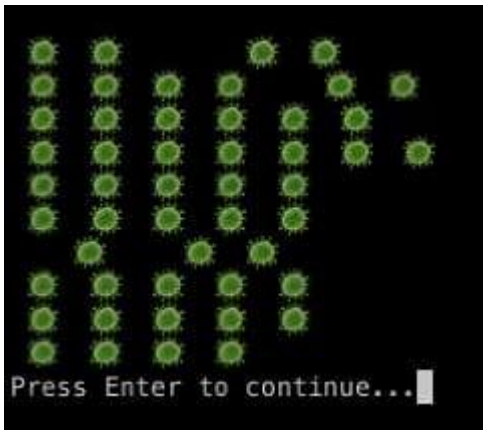


Файл covid.cpp

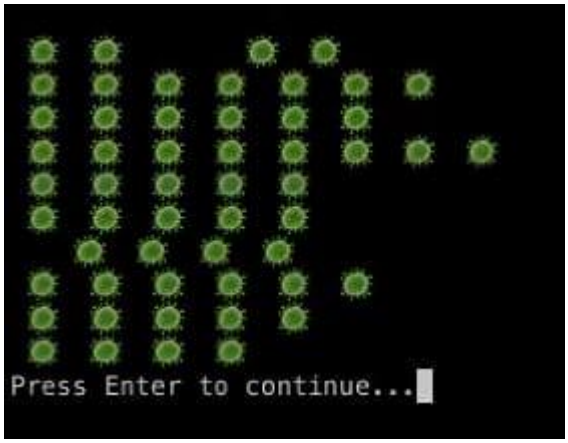
Епоха - 1



Епоха - 2

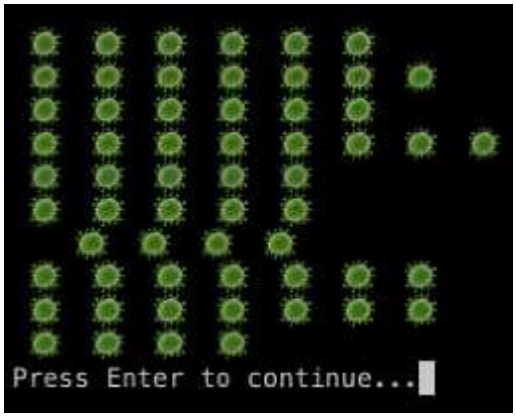


Епоха - 3



Епоха - 4

Епоха - 5



Висновок

Було успішно змодельовано процеси еволюції і смерті звичайних клітин та клітин коронавірусу. При збільшенні розмірів поля моделювання різниця в часі виконання коду послідовно та паралельно буде суттєво збільшуватися, що доводить перевагу використання мультипроцесингу та OpenMP.

Заміри часу (час паралельного виконання програми - час послідовного виконання програми) в залежності від розміру поля:

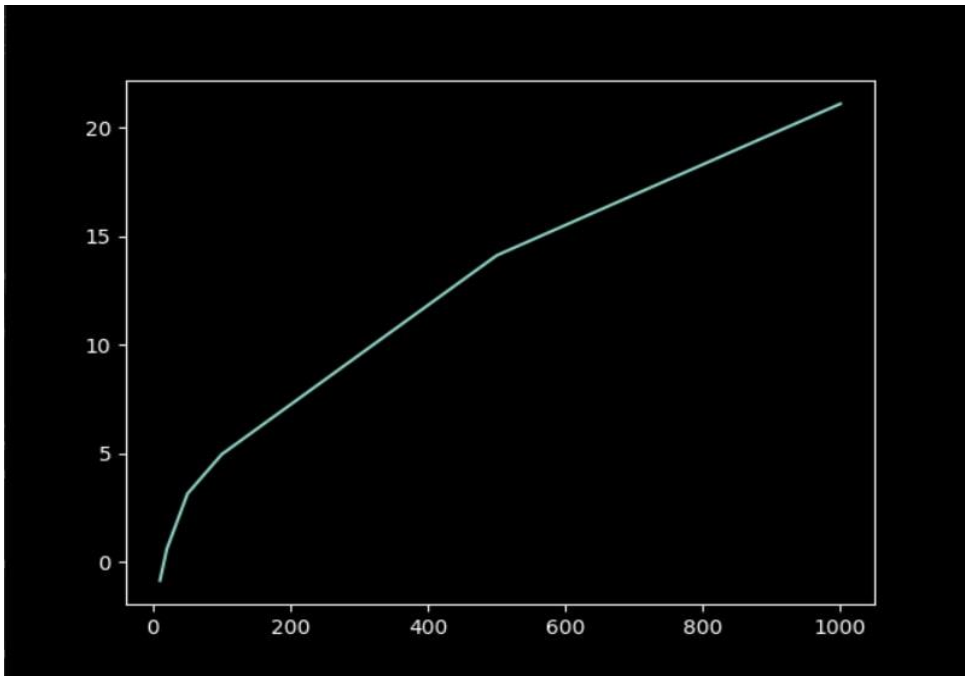
```
Поле розміром 10, різниця в часі = -0.87129847
```

```
Поле розміром 20, різниця в часі = 0.58453399
```

```
Поле розміром 100, різниця в часі = 4.954386091
```

```
Поле розміром 1000, різниця в часі = 21.1154892
```

Графік залежності різниці в часі від розміру поля:



Код програми

```
#include <omp.h>
#include <iostream>
#include <time.h>
#include <stdlib.h>
using namespace std;
const int n = 10, m = 10;



int main()
{
    omp_set_num_threads(4);
    int arr[n][m];
    int new_arr[n][m];
    int i, j, i1, i2, j1, j2, cells;

    srand(time(0));
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            arr[i][j] = rand() % 2;
            new_arr[i][j] = arr[i][j];
        }
    }

    do
    {
        cout << endl;
        for (int i = 0; i < n; i++)
```

```
        {
            for (int j = 0; j < m; j++)
            {
                arr[i][j] = new_arr[i][j];
                if (arr[i][j] == 0)
                {
                    cout << "   ";
                }
                if (arr[i][j] == 1)
                {
                    cout << " 1 ";
                }
            }

            cout << endl;
        }

        system("pause");
        system("cls");
```

```
#pragma omp parallel for private(i, j, i1, i2, j1, j2, cells)
shared(arr)
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < m; j++)
            {
                i1 = i - 1;
                if (i1 < 0)
                    i1 = 0;
                j1 = j - 1;
                if (j1 < 0)
```

```
    j1 = 0;
i2 = i + 1;
if (i2 >= n)
    i2 = n - 1;
j2 = j + 1;
if (j2 >= m)
    j2 = m - 1;



if (arr[i][j] == 0)
{
    cells = 0;

    for (int i3 = i1; i3 <= i2; i3++)
    {

        for (int j3 = j1; j3 <= j2; j3++)
        {
            if (arr[i3][j3] == 1)
            {
                cells++;
            }
        }
    }

    if (cells == 3)
    {
        new_arr[i][j] = 1;
    }
}

if (arr[i][j] == 1)
{
```

```
                    cells = -1;
                    for (int i3 = i1; i3 <= i2; i3++)
                    {
                        for (int j3 = j1; j3 <= j2; j3++)
                        {
                            if (arr[i3][j3] == 1)
                            {
                                cells++;
                            }
                        }
                    }
                    if (cells < 2)
                    {
                        new_arr[i][j] = 0;
                    }
                    if (cells > 3)
                    {
                        new_arr[i][j] = 0;
                    }
                }
            }
        }

    } while (1);

    return 0;
}
```