

Atividade de Programação 1

Rainbow Game of Life - PThreads e OpenMP

Grupos de até 3 alun*s.

Forma de entrega: Link para projeto no GitHub (ou bitbucket ou gitlab) contendo 2 arquivos pedidos: códigos-fonte em C/C++ para as versões PThreads e *OpenMP*.

Deve-se enviar também um **Link para vídeo na plataforma Google Drive institucional ou Youtube de até 7 minutos impreterivelmente**, demonstrando o funcionamento dos códigos desenvolvidos, e informando o desempenho obtido ao se usar múltiplas *threads*.

Formulário google para submissão:

<https://forms.gle/cpi5bYa58XzRiqKc8>

O Jogo da Vida [1], criado por John H. Conway, utiliza um autômato celular para simular gerações sucessivas de uma sociedade de organismos vivos. É composto por um tabuleiro bi-dimensional, infinito em qualquer direção, de células quadradas idênticas. Cada célula tem exatamente oito células vizinhas (todas as células que compartilham, com a célula original, uma aresta ou um vértice). Cada célula está em um de dois estados: viva ou morta (correspondentes aos valores 1 ou 0). Uma geração da sociedade é representada pelo conjunto dos estados das células do tabuleiro. Sociedades evoluem de uma geração para a próxima aplicando simultaneamente, a todas as células do tabuleiro, regras que estabelecem o próximo estado de cada célula. As regras são:

- A. Células vivas com menos de 2 (dois) vizinhas vivas morrem por abandono;
- B. Cada célula viva com 2 (dois) ou 3 (três) vizinhos deve permanecer viva para a próxima geração;
- C. Cada célula viva com 4 (quatro) ou mais vizinhos morre por superpopulação;
- D. Cada célula morta com exatamente 3 (três) vizinhos deve se tornar viva.

Resumidamente, as 4 regras podem ser descritas como:

- 1) Qualquer célula viva com 2 (dois) ou 3 (três) vizinhos deve sobreviver;
- 2) Qualquer célula morta com 3 (três) vizinhos torna-se viva;
- 3) Qualquer outro caso, células vivas devem morrer e células já mortas devem continuar mortas.

Alternativamente ao Jogo do Vida existe uma versão do mesmo conhecida por *Rainbow Game of Life* [2], que tem uma única diferença: as novas células que tornam-se vivas deverão ter como valor a média aritmética dos valores na vizinhança imediata. Dessa forma, o tipo de dados das matrizes correspondentes devem ser do tipo FLOAT.

Programa duas versões concorrentes do *Rainbow Game of Life*, em linguagem C/C++ utilizando PThreads e OpenMP, sobre um tabuleiro finito, $N \times N$ com bordas infinitas, ou seja, a fronteira esquerda liga-se com a fronteira direita e a fronteira superior liga-se com a fronteira inferior.

Admita que a posição $(0,0)$ identifica a célula no canto superior esquerdo do tabuleiro e que a posição $(N-1,N-1)$ identifica a célula no canto inferior direito.

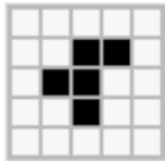
Considere que uma célula viva tem seu valor maior que zero.

Estruture seu programa da forma abaixo:

1. Aloque dinamicamente a(s) matriz(es) necessária(s) (de números do tipo float) para representar duas gerações do tabuleiro com tamanho $N \times N$. Sugestão: use *grid* e *newgrid* para a geração atual e futura, respectivamente.
2. Inicie a geração inicial do tabuleiro (*array*) a partir da posição $(1,1)$ com uma figura de um *Glider*, onde as células vazias correspondem ao valor zero $(0,0)$ e as células preenchidas correspondem ao valor um $(1,0)$, conforme a figura abaixo:



e também com a figura de um R-pentomino (<https://www.conwaylife.com/wiki/R-pentomino>) a partir da linha 10 e coluna 30, da seguinte forma:



Pode-se usar o código a seguir para estas definições iniciais:

```
//GLIDER
int lin = 1, col = 1;
grid[lin][col+1] = 1.0;
grid[lin+1][col+2] = 1.0;
grid[lin+2][col] = 1.0;
grid[lin+2][col+1] = 1.0;
grid[lin+2][col+2] = 1.0;

//R-pentomino
lin =10; col = 30;
grid[lin][col+1] = 1.0;
grid[lin][col+2] = 1.0;
grid[lin+1][col] = 1.0;
grid[lin+1][col+1] = 1.0;
grid[lin+2][col+1] = 1.0;
```

3. Crie uma função/método que retorne a quantidade de vizinhos vivos de cada célula na posição i,j :

```
int getNeighbors(int** grid, int i, int j) { ... }
```

4. Crie um laço de repetição para executar um determinado número máximo de iterações do jogo, ou seja, determine a quantidade de gerações sucessivas do tabuleiro que devem ser geradas.
5. Crie um procedimento (ou trecho de código) que, ao finalizar todas as iterações/gerações, compute a quantidade de células vivas.

A entrega consiste nos códigos-fonte na forma de um link para o *github*, devidamente organizado pelo grupo com arquivos identificando cada problema e os participantes do grupo, os nomes dos participantes **devem** estar devidamente identificados no código-fonte.

No vídeo deve-se exibir as medidas de desempenho (tempo de processamento) para a versão serial e versões concorrentes em *C/C++* (com *PThreads* e *OpenMP*), variando a quantidade de *threads* em 1, 2, 4 e 8 (mesmo que o computador usado para teste não tenha essa quantidade de núcleos). Deve-se considerar ainda um tabuleiro quadrado de dimensões *2048*2048* e um total de *2000* gerações sucessivas do tabuleiro. No vídeo deve ser informado a configuração do computador utilizado nos testes, citando claramente a identificação da CPU e a quantidade de núcleos (*cores*) físicos disponíveis (não confundir com capacidade de núcleos virtuais em *hyperthreading* em CPUs INTEL), além da quantidade de memória principal.

Para demonstrar o funcionamento deve-se demonstrar no vídeo as 5 primeiras gerações, exibindo uma submatriz da grade original de tamanho (50x50), começando na posição (0,0).

Demonstre o resultado de forma a ser possível identificar graficamente a magnitude dos valores dentro das células, como, por exemplo, através de escala de cinza ou com padrões de caracteres.

A quantidade de células vivas esperadas para as gerações iniciais e também para a última geração, sob as condições citadas, são as seguintes:

**** Rainbow Game of Life**

Condição inicial: 10

Geração 1: 11

Geração 2: 12

Geração 3: 14

Geração 4: 13

...

Última geração (2000 iterações): 51 células vivas

A medida de desempenho deve contemplar o tempo total de execução no Sistema Operacional (medido com *time*) e o tempo de execução apenas do trecho que envolve o laço que computa as gerações sucessivas (instrumentar o código para retornar o tempo decorrido apenas desse trecho).

Durante o desenvolvimento pode-se utilizar tabuleiros e um número menor de gerações, entretanto, a análise de desempenho demonstrada no vídeo deve contar com a configuração de tabuleiro e de *threads* já definidos, ou seja:

Tamanho do tabuleiro = $2048 * 2048$

Quantidade de gerações = 2000

Quantidade de *threads* = 1, 2, 4 e 8

Os resultados podem ser demonstrados em tabelas ou gráficos.

Demonstre também que as versões concorrentes chegam exatamente no mesmo valor de células vivas que a versão serial, ao final da execução.

OBS: Os **plágios não serão tolerados** e sofrerão penalidades. Os códigos-fonte enviados estão sujeitos a análise de plágio através do software MOSS (<http://theory.stanford.edu/~aiken/moss/>). A checagem de similaridade baseia-se não apenas em uma simples diferenças entre arquivos texto, mas em critérios mais avançados que podem ser melhor entendidos aqui:

<http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf>

[1] – Martin Gardner, “Mathematical Games – The fantastic combinations of John Conway’s new solitaire game life”, Scientific American 223, Oct. 1970, pp 120-123.

http://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis_projekt/proj_game_life/ConwayScientificAmerican.htm

[2] - Games of Life in Colour: <https://people.kth.se/~gunnarj/LIFE/WLIF/wlcframes.html>