Atividade de Programação 1
Jogo da Vida - *PThreads/OpenMP/JavaThreads*

Grupos de até 4 alunos(as).

Forma de entrega: Arquivo compactado contendo quatro arquivos: um relatório em PDF, um código-fonte em *C/C++* para a versão *PThread*, código-fonte em *C/C++* para a versão *OpenMP*, e um código-fonte em *Java*.
Obs: Não utilizar caracteres acentuados em nomes de arquivos e pastas.

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.

Programe três (3) versões concorrentes deste código, em linguagem *C/C++* utilizando *PThreads*,em linguagem *C/C++* utilizando *OpenMP*, e outra em *Java* com *JavaThreads*, que implementem o Jogo da Vida sobre um tabuleiro finito, NxN 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 *(0,0)* identifica a célula no canto superior esquerdo do tabuleiro e que *(N-1,N-1)* identifica a célula no canto inferior direito.

Estruture seu programa da forma abaixo:

1. Aloque dinamicamente a(s) matriz(es) necessária(s) (de números inteiros) para representar duas gerações do tabuleiro com tamanho *N*N*. Sugestão: use *grid* e *newgrid* para a geração atual e futura, respectivamente.
2. Inicie cada posição da geração inicial do tabuleiro (*array*) pseudo-aleatoriamente da seguinte forma:

   Em *C/C++:*

```
#define SRAND_VALUE 1985

srand(SRAND_VALUE);

for(i = 0; i<dim; i++) { //laço sobre as células do tabuleiro sem contar com um eventual halo

  for(j = 0; j<dim; j++) {

     grid[i][j] = rand() % 2;

  }
```

Em Java:

```
Random gerador = new Random(1985);



for(i = 0; i<dim; i++) { //laço sobre as células do tabuleiro sem contar com um eventual halo

   for(j = 0; j<dim; j++) {

      grid[i][j] = gerador.nextInt(2147483647) % 2;

  }

}
```

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 da vida, 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, some todas as posições da última geração do tabuleiro e retorne a quantidade de células vivas.

No relatório deve-se constar medidas de desempenho (tempo de processamento) para a versão serial e versões concorrentes em *C/C++ (PThreads e OpenMP)* e *Java*, variando a quantidade de *threads* em 1, 2, 4 e 8 (mesmo que a máquina testada 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.

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

Condição inicial: 2096241

Geração 1: 1146561

Geração 2: 1063629

Geração 3: 1052114

Geração 4: 1000392

...

Última geração (2000 iterações): 146951 células vivas

A medida de desempenho deve contemplar o tempo total de execução (medido com time) e o tempo de execução apenas do trecho que envolve o laço que computa as gerações sucessivas.

Durante o desenvolvimento pode-se utilizar tabuleiros e quantidade de gerações menores. Mas a análise de desempenho do relatório deve contar com os valores definidos acima, 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 ao final da execução.

No relatório deve-se especificar a máquina em que o código foi executado, citando a identificação do processador, a quantidade de núcleos reais e a quantidade de memória principal disponível.

OBS: as submissões de códigos estão sujeitas 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 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_gamelife/ConwayScientificAmerican.htm