Trabalho 1 – *Game of Life* Concorrente

INE5410 – Programação Concorrente – UFSC

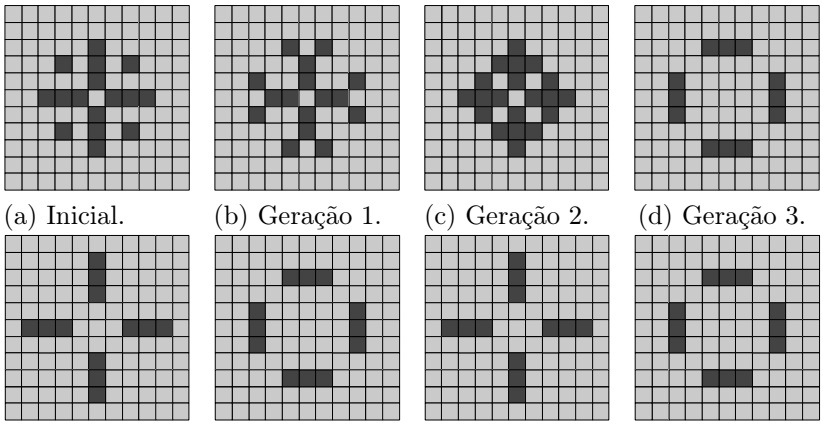
1 Introdução

O Jogo da Vida (*The Game of Life* - GoL) é um autômato celular inventado pelo matemático John Conway da Universidade de Cambridge (`https://playgameoflife.com`). Ele consiste em uma coleção de células que, baseadas em algumas poucas regras matemáticas, podem viver, morrer ou se multiplicar. Dependendo das condições iniciais, as células formam vários padrões durante todo o curso do jogo.

A versão do jogo adotada nesse trabalho possui um tabuleiro quadrado (matriz ou *array* 2D) onde as células são atualizadas (modificadas) em cada geração (*timestep*) de acordo com as seguintes regras:

- Uma célula viva que possui **menos de dois vizinhos vivos** morre de solidão;
- Uma célula viva que possui **dois ou três vizinhos vivos** continua no mesmo estado para a próxima geração;
- Uma célula viva que possui **mais de três vizinhos vivos** morre de superpopulação;
- Uma célula morta que possui **exatamente três vizinhos vivos** se torna uma célula viva.

A Figura 1 mostra um exemplo de uma simulação de 7 gerações do GoL em um tabuleiro de tamanho 11x11. Células vazias/mortas e vivas são representadas pelas cores cinza claro e escuro, respectivamente. A Figura 1a mostra a situação inicial do tabuleiro. As Figuras 1b–1h mostram o resultado em cada uma das 7 gerações.



(a) Inicial. (b) Geração 1. (c) Geração 2. (d) Geração 3.

(e) Geração 4. (f) Geração 5. (g) Geração 6. (h) Geração 7.

Figura 1: Exemplo de uma simulação de 7 gerações do *Game of Life* em um tabuleiro de tamanho 11x11.

2 Definição do Trabalho

O trabalho da disciplina de Programação Concorrente consiste em desenvolver uma versão paralela do GoL utilizando a biblioteca **POSIX threads**. As threads deverão dividir o trabalho da computação de forma a acelerar a execução da aplicação.

Você deverá utilizar como base a versão sequencial do GoL implementada em C disponível no Moodle. Nessa versão, os parâmetros de entrada são descritos em um arquivo texto. A primeira linha informa o tamanho do tabuleiro e o número de gerações (ambos inteiros) separados por um espaço em branco. As linhas seguintes representam as células do tabuleiro inicial, onde um espaço em branco representa uma célula vazia/morta e um "x" representa uma célula

viva. As linhas do arquivo são separadas por quebras de linhas (*line breaks*). Como saída, o programa escreve na tela o último tabuleiro, ou seja, após a execução de todas as gerações.

Na versão paralela, você deverá incluir um parâmetro de entrada para a execução do programa via linha de comando (`argc, argv`) correspondente ao número de threads que deverão ser utilizadas na computação.

3 Compilação, Execução e Debug

Para compilar, execute `make` no diretório principal do projeto. Para remover o binário, execute o comando `make clean` a partir do diretório principal do projeto.

Após a compilação, você deverá executar o programa da seguinte maneira (exemplo de execução com o arquivo de entrada chamado "input-little.in" armazenado no diretório principal do projeto):

```
./gol input-little.in
```

Adicionalmente, você poderá combinar a saída do GoL com a ferramenta `tr` para gerar uma saída que permitirá uma melhor visualização do resultado. A ideia é utilizar o `tr` para substituir os espaços em branco por "_". Para isso, utilize o seguinte comando:

```
./gol input-little.in | tr ' ' '_'
```

Por padrão, o programa somente irá imprimir na tela o último tabuleiro. Caso você deseje imprimir o tabuleiro ao final de cada geração você precisará remover o comentário (`#`) da seguinte linha do arquivo `Makefile`: `-DDEBUG`.

Para medir o desempenho da sua solução paralela utilize a ferramenta `time` do Linux. Por exemplo, a seguinte linha irá medir o tempo de execução do GoL utilizando como entrada o arquivo "input-big.in". Você deverá observar o tempo de execução na linha indicada por "real".

```
time cat input-big.in | ./gol
real    3m1.761s
user    3m1.692s
sys     0m0.160s
```

Sempre que for medir o tempo evite fazer chamadas à função `printf()`. Para isso, deixe `#-DDEBUG` e `#-DRESULT` no `Makefile` e recompile o programa.

4 Grupos, Avaliação e Entrega

O trabalho deverá ser realizado **obrigatoriamente** em grupos de **3 alunos**. Os alunos serão responsáveis por formar os grupos com auxilio da ferramenta "**Escolha de Grupos - Trabalho 1 (T1)**" disponível no Moodle.

Pelo menos um dos integrantes de cada grupo deverá submeter um arquivo contendo o código fonte em C contendo a solução do trabalho através do Moodle no prazo estipulado. **Trabalhos não entregues no prazo receberão nota zero.**

Após a data limite para entrega, os alunos deverão apresentar o trabalho ao professor assim como mostrar sua solução em funcionamento. As apresentações serão feitas durante as aulas conforme o cronograma da disciplina disponível no Moodle.

O professor irá avaliar não somente a corretude mas também o desempenho e a clareza da solução. Durante a apresentação, o professor irá avaliar o **conhecimento individual dos alunos sobre os conteúdos teóricos e práticos vistos em aula e sobre a solução adotada no trabalho**. A nota atribuída à cada aluno i no trabalho ($NotaTrabalho_i$) será calculada da seguinte forma, onde A_i é a nota referente à apresentação do aluno i e S é a nota atribuída à solução do trabalho:

$$NotaTrabalho_i = \frac{A_i * S}{10} \tag{1}$$

Como indicado pela fórmula mostrada acima, **a nota atribuída à solução adotada será ponderada pelo desempenho do aluno durante a apresentação do trabalho**. Por exemplo, se o professor atribuir nota 10 para a solução adotada pelo grupo mas o aluno receber nota 5 pela apresentação – devido ao desconhecimento dos conteúdos teóricos, práticos e/ou da solução do trabalho – a sua nota final do trabalho será 5. A ausência no dia da apresentação ou recusa de realização da apresentação do trabalho implicará em nota zero na apresentação, fazendo com que a nota atribuída ao aluno também seja zero.