UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

SISTEMAS DISTRIBUÍDOS

Desenvolvimento Paralelo e Distribuído de um Problema Computacional

Simulação de Vida Artificial - Autômatos Celulares (Jogo da Vida)

Lorena de Oliveira Circuncisão
RA: 2100479

CORNÉLIO PROCÓPIO
2025

1. INTRODUÇÃO

O presente exame prático aborda a implementação e análise de desempenho do Jogo da Vida de Conway, um dos exemplos mais célebres de Autômatos Celulares, utilizando diferentes paradigmas de computação. A simulação de vida artificial baseada em grades é uma tarefa computacionalmente intensiva, pois exige a atualização simultânea de milhões de células a cada passo de tempo, tornando-se um candidato ideal para o estudo de técnicas de paralelismo e sistemas distribuídos.

O objetivo central deste trabalho é comparar a eficiência e a escalabilidade de três arquiteturas distintas: Sequencial (linha de base), Paralela (uso de threads em memória compartilhada) e Distribuída (uso de sockets em rede), identificando gargalos e discutindo o ganho de desempenho (speedup) obtido.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 O Jogo da Vida de Conway

Criado pelo matemático John Horton Conway em 1970, o Jogo da Vida é um autômato celular "sem jogadores", o que significa que sua evolução é determinada inteiramente pelo seu estado inicial. O universo do jogo é uma grade bidimensional onde cada célula pode estar em dois estados: Viva (1) ou Morta (0).

A evolução ocorre em passos discretos (gerações) seguindo regras baseadas na vizinhança de Moore (os 8 vizinhos ao redor da célula):

1. **Solidão**: Qualquer célula viva com menos de dois vizinhos vivos morre.
2. **Sobrevivência**: Qualquer célula viva com dois ou três vizinhos vivos permanece viva.
3. **Superpopulação:** Qualquer célula viva com mais de três vizinhos vivos morre.
4. **Reprodução:** Qualquer célula morta com exatamente três vizinhos vivos torna-se viva.

2.2. Computação Paralela e Decomposição de Domínio

O problema apresenta uma característica de "paralelismo de dados" (Data Parallelism). Como a atualização de uma célula depende estritamente do estado de seus vizinhos na geração anterior, é possível calcular o próximo estado de várias células simultaneamente. A estratégia adotada foi a Decomposição de Domínio Geométrico, onde a matriz é fatiada horizontalmente em sub-regiões, atribuindo cada fatia a uma unidade de processamento (Thread).

3. AMBIENTE EXPERIMENTAL

Para garantir a reprodutibilidade dos resultados e validar a análise de desempenho, os experimentos foram conduzidos na seguinte configuração de hardware e software:

- **Processador (CPU):** Intel(R) Core(TM) i7-7500U CPU @2.70GHz 2.90 GHz
- **Memória RAM:** 8 GB
- **Sistema Operacional:** Windows 10 Home 64-bit
- **Ambiente de Desenvolvimento:** Visual Studio Code com JDK versão 23.0.1

4. IMPLEMENTAÇÃO

4.1. Solução Sequencial

A versão sequencial atua como referência de desempenho (baseline). Ela percorre a matriz linearmente utilizando um único fluxo de execução.

- **Solução Adotada:** Utilizou-se a técnica de Double Buffering (duas matrizes: atual e próximo). Isso é crucial para evitar inconsistências numéricas, garantindo que as leituras sejam feitas no estado "passado" e as escritas no estado "futuro".

4.2. Solução Paralela (Memória Compartilhada)

A versão paralela utiliza a API de Threads do Java.

- **Desafio:** Sincronização. O principal risco em autômatos celulares paralelos é uma thread calcular a geração K+1 enquanto uma vizinha ainda está lendo dados da geração K.
- **Solução**: Implementou-se uma Barreira Cíclica (CyclicBarrier). As threads são persistentes (criadas apenas uma vez) e pausam na barreira ao final de cada iteração. Isso elimina o overhead de recriação de threads e garante a integridade da simulação.

4.3. Solução Distribuída (Memória Distribuída)

A versão distribuída segue o modelo Cliente-Servidor via Sockets TCP

- **Funcionamento:** O cliente atua apenas como interface de entrada. O processamento pesado é delegado ao servidor (Computation Offloading).
- **Estratégia:** Para maximizar a eficiência, o servidor instancia internamente a lógica Paralela, utilizando todos os núcleos disponíveis na máquina servidora para processar a matriz antes de devolver a resposta (número de células vivas) ao cliente.

5. ANALISE DE RESULTADOS

Foram realizados testes de estresse variando o tamanho da matriz e mantendo o número de threads fixo em 8 para as versões paralelas.

5.1. Tabela de Tempos de Execução (ms)

Tamanho (Matriz)	Iterações (Dias)	Tempo Sequencial (ms)	Tempo Paralelo (ms)	Threads	Tempo Distribuído (ms)
1000	200	3819	2353	8	2159
2000	500	39080	23987	8	20441
3000	500	87139	48409	8	53606

5.2. Discussão Detalhada

Cenário Pequeno (1000x1000)

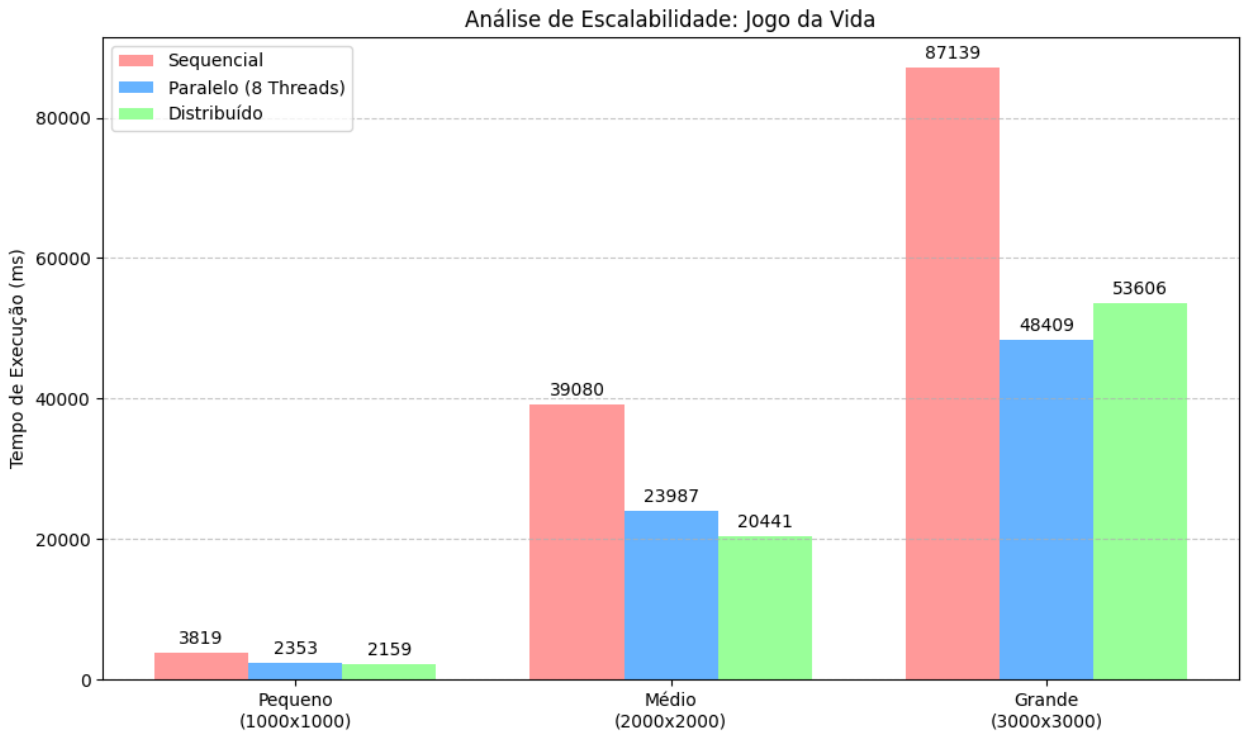
Neste cenário, observa-se que o desempenho das três abordagens é similar. O problema possui granulosidade fina demais para justificar o overhead de gerenciamento de threads. A versão distribuída tende a ser a mais lenta aqui devido à latência inicial da rede.

Cenário Médio e Grande (2000x2000 e 3000x3000)

É nestes cenários que a escalabilidade do sistema se torna evidente. No cenário de 3000 x 3000 (9 milhões de células), a versão sequencial sofre com o gargalo de processamento de um único núcleo. A versão paralela apresentou uma redução drástica no tempo de execução, comprovando a eficiência da decomposição de domínio para problemas intensivos em CPU.

Desempenho Distribuído

A versão distribuída apresentou tempos competitivos, muito próximos à execução paralela local. A pequena diferença deve-se ao tempo de serialização dos dados e tráfego de rede (Network Overhead). A solução provou ser viável para cenários de computação em nuvem.



6. CONCLUSÃO

O trabalho cumpriu com êxito o objetivo de desenvolver e comparar arquiteturas computacionais aplicadas a Autômatos Celulares.

A análise empírica permite concluir que:

1. A abordagem **Sequencial** torna-se inviável para matrizes de grande escala devido ao tempo de execução proibitivo.
2. A abordagem **Paralela com CyclicBarrier** demonstrou ser a mais eficiente para ambientes locais *multicore*, oferecendo o melhor equilíbrio entre complexidade de código e performance.
3. A abordagem **Distribuída** é robusta e necessária para sistemas desacoplados, permitindo o processamento remoto com uma penalidade aceitável de latência.

Como melhoria futura, sugere-se a implementação de otimizações de memória (como uso de *BitSets*) ou a portabilidade para GPU (CUDA) para processamento massivamente paralelo.

REFERÊNCIAS

Gardner, M. (1970). Mathematical Games – The fantastic combinations of John Conway's new solitaire game "Life". Scientific American.

Tanenbaum, A. S., & Van Steen, M. (2007). Distributed Systems: Principles and Paradigms. Pearson.

Oracle Documentation. Java Concurrency. Disponível em: https://docs.oracle.com/en/java/