Game of Life: Implementação com OpenCL

José Alan Teixeira[1], Nelson Cerqueira[2]

[1]Programa de Pós-graduação em Ciências da Computação

[2]Programa de Pós-graduação em Ciências da Computação

Universidade Estadual de Feira de Santana (UEFS)

Feira de Santana - BA - Brasil

{joseasteixeira7, nelsonccnetoba}@gmail.com

1. Introdução

O *Game of Life*, proposto por John Conway, é um autômato celular amplamente estudado em simulações de sistemas dinâmicos complexos. Sua simplicidade de regras e capacidade de gerar padrões complexos tornam-no ideal para explorar conceitos em computação paralela. Diante do crescimento da demanda por desempenho computacional, técnicas que exploram a capacidade massivamente paralela de GPUs vêm sendo cada vez mais aplicadas. Nesse contexto, o presente trabalho propõe uma implementação do *Game of Life* utilizando a plataforma OpenCL, que permite a execução paralela do código em dispositivos heterogêneos, como CPUs e GPUs.

A proposta deste artigo é descrever em detalhes a arquitetura do sistema implementado, abordando desde a preparação do ambiente OpenCL até a execução e controle dos *kernels* responsáveis pela evolução do autômato. Além disso, são discutidas as estratégias de paralelismo adotadas, as vantagens do uso da GPU no contexto específico do *Game of Life*, bem como as instruções para execução e os resultados obtidos. O objetivo é fornecer uma visão clara e prática da utilização de OpenCL para simular sistemas computacionais dinâmicos, servindo tanto como base para trabalhos futuros quanto como material didático para o ensino de computação paralela.

2. Descrição do código

2.1. Inicialização do OpenCL

O objetivo da função inicializar_opencl, é preparar todo o ambiente necessário para executar um kernel OpenCL, essa função recebe como argumento uma string com o caminho para o arquivo kernel_gol.cl contendo o código do kernel OpenCL, e retorna 1 (sucesso) ou 0 (erro).

A variável cl_int err é usada para verificar o sucesso de cada chamada OpenCL. Inicialmente é obtido o identificador da primeira plataforma OpenCL Intel, em seguida, é solicitado um dispositivo do tipo GPU da plataforma selecionada, o resultado é salvo na variável global device. É criado então um contexto OpenCL que agrupa recursos

compartilhados para um ou mais dispositivos, nesse caso o contexto é criado para apenas um dispositivo (device). A variável queue cria uma fila de comandos onde o host (CPU) envia instruções para a GPU.

Na sequência o arquivo kernel_gol.cl é aberto e o conteúdo é lido e copiado para um ponteiro char* source. Cria-se na variável program um objeto de programa OpenCL com base no código-fonte lido. Caso ocorra erro na compilação, essa linha recupera e imprime o log do compilador, útil para encontrar erros no kernel_gol.cl. Cria-se na variável kernel um objeto que é uma instância do código do arquivo kernel_gol.cl. Por último libera-se a memória usada para armazenar o código-fonte lido e retorna 1, indicando que tudo ocorreu bem.

2.2. Kernel

No arquivo kernel.cl está a o Kernel responsável por receber duas grades (grid_atual e grid_prox) e as dimensões da grade (width e height); cada work-item calcula o estado da célula em (x, y) na próxima geração; as regras do Jogo da Vida são aplicadas com bordas periódicas. O código está estruturado da seguinte forma:

- __kernel: indica que essa função será executada na GPU.
- __global: indica que os ponteiros acessam a memória global da GPU.
- grid_atual: a grade com o estado atual das células.
- grid_prox: a grade de saída (próxima geração).
- width, height: dimensões da grade.

Os loops aninhados contam o número de vizinhos vivos ao redor da célula, usando aritmética modular para bordas periódicas. As variáveis inteiras idx e estado são para calculara o índice linear da célula na grade em uma dimensão. E na sequência são aplicadas as regras do jogo da vida usando a condicional if, célula viva continua viva se tiver 2 ou 3 vizinhos, célula morta nasce se tiver exatamente 3 vizinhos.

2.3. Arquivo logic.c

No arquivo logic.c a função checkError verifica se uma chamada OpenCL teve sucesso, se ocorreu erro (err != CL_SUCCESS), imprime uma mensagem e termina o programa, é usada para facilitar o tratamento de erros ao longo do código.

A função gen_next_gpu é responsável por calcular a próxima geração do Jogo da Vida usando a GPU via OpenCL. Ela prepara os dados, envia-os para a GPU, executa o kernel e lê o resultado de volta. Inicialmente aloca dois vetores unidimensionais do tamanho da grade: um para a grade atual (grid_flat) e outro para a próxima geração (new_grid_flat), é feita a transformação da matriz grid[HEIGHT][WIDTH] (com structs Cell) em um vetor int* para facilitar o envio para a GPU, portanto a GPU utiliza apenas uma dimenssão para o processamento do código. Na sequência é feita a criação de buffers para GPU: d_grid é buffer de leitura com os dados da geração atual, d_new_grid o buffer de escrita para receber a próxima geração. São passados os argumentos esperados pelo kernel _gol: grid_atual recebe d_grid, grid_prox recebe d_new_grid, width e height. Ocorre então a configuração da execução paralela, definindo grade global de execução, um thread para cada célula (2D) e são lançadas HEIGHT × WIDTH instâncias do kernel. A execução do kernel ocorre ao chamar a função clEnqueueNDRangeKernel, a execução

do kernel é enfileirada, o parâmetro 2 indica que o kernel usa duas dimensões (X e Y) e todas as threads são executadas em paralelo pela GPU.

O término de todas as operações pendentes na fila, incluindo o kernel é aguardado, e os dados da nova geração da GPU são copiados de volta para new_grid_flat, o vetor new_grid_flat é convertido de volta para a estrutura 2D grid, atualizando o estado de cada célula, e por último ocorre a liberação da memória alocada na GPU (cl_mem) e na CPU (malloc).

3. Como executar

Para executar o código é preciso um compilador para a linguagem C e o drive do openCL compativel com a GPU utilizada. Para compilar o código utilizando os parametros do arquivo de texto deve-se executar no terminal o comando: [] e para executar usa-se o comando:

Caso queira compilar e executar o codigo por meio da Makefile basta utilizar o comando make run no terminal.

4. Resultados

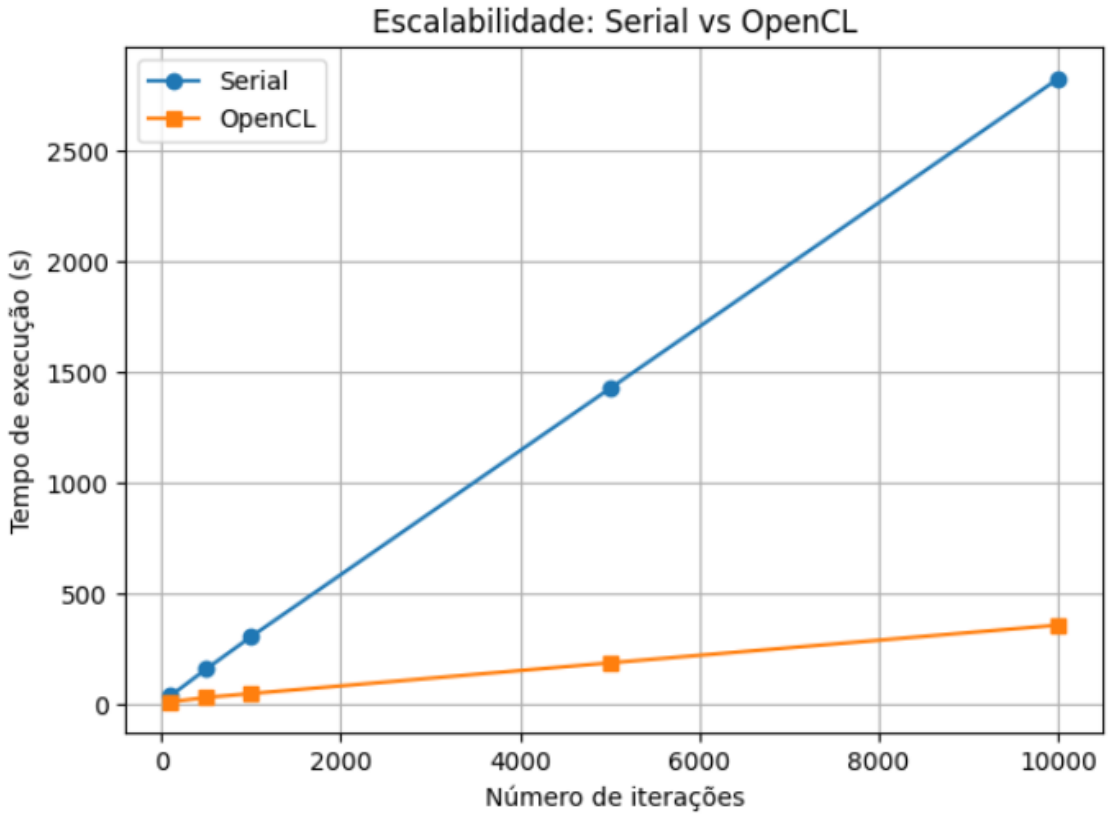
Para realizar os testes de execução a GPU utilizada possui as seguintes características:

- Name: Intel(R) UHD Graphics 620
- Version: OpenCL 3.0
- Max. Compute Units: 24
- Local Memory Size: 64 KB
- Global Memory Size: 6927 MB
- Max Alloc Size: 3463 MB
- Max Work-group Total Size: 256
- Max Work-group Dims: (256 256 256)

Ao avaliar o desempenho da implementação com OpenCL em comparação com a versão serial do algoritmo do *Game of Life*, foram realizados experimentos com diferentes cargas de entrada. A tabela a seguir apresenta os tempos de execução (em segundos) observados para cada abordagem:

Tabela 1: Tempos de execução serial e com OpenCL para cada carga.

Tempos(s)		
Carga	Serial	OpenCL
100	37,15	10.51
500	159,09	30.77
1000	305,13	48.16
5000	1.426,64	186.94
10000	2.827,55	357.69



Figira 1: Gráfico de comparação de desempenho.

Analisando a Figura 1 os resultados evidenciam ganhos significativos de desempenho com o uso do OpenCL, especialmente para cargas maiores. Na carga de 5000 interações, por exemplo, a execução paralela reduziu o tempo de simulação em mais de 87% em relação à versão serial. Mesmo em cargas menores, a aceleração com OpenCL já é perceptível, confirmando a eficiência da paralelização para esse tipo de problema.

Tabela 2: Speedup e Efeciência para cada carga.

Carga	Speedup	Eficiência
100	3,53	0,15
500	5,17	0,22
1000	6,34	0,26
5000	7,63	0,32
10000	7,90	0,33

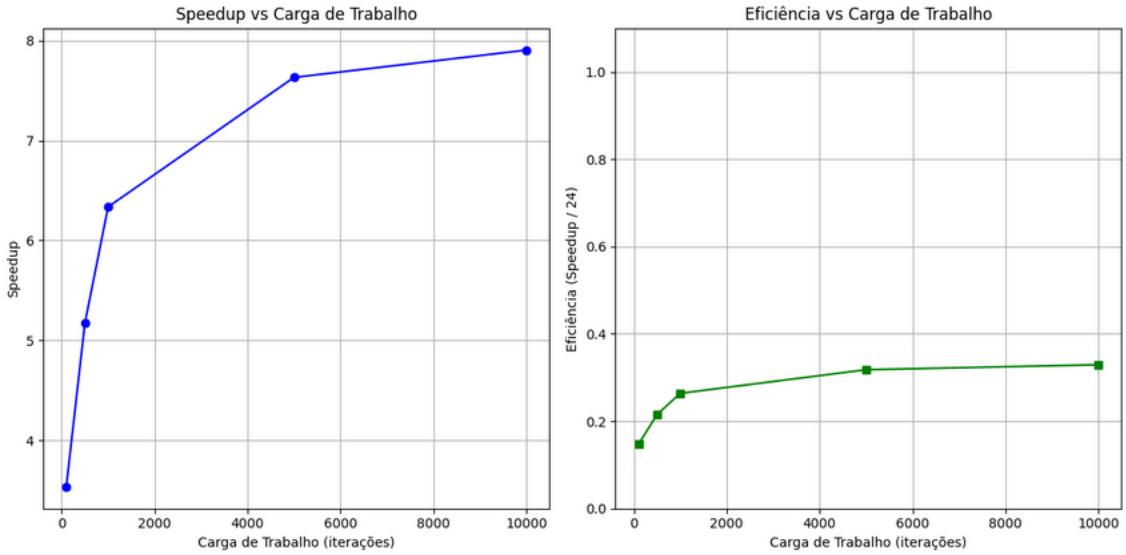


Figura 2: Gráficos do Speedup e da Eficiência

Observou-se que o speedup (razão entre o tempo serial e o tempo paralelo) aumentou conforme a carga de trabalho cresceu. Para uma carga de 100 iterações, o speedup foi de aproximadamente 3,53, enquanto para 10.000 iterações o valor alcançou 7,90. Esse comportamento acontece, pois, tarefas maiores tendem a aproveitar melhor os recursos de paralelismo, amortizando o overhead associado à inicialização e à comunicação entre as unidades de execução.

Além do speedup, também foi avaliada a eficiência paralela, definida como a razão entre o speedup obtido e o número total de unidades de execução disponíveis na GPU, que neste caso é de 24 Compute Units. A eficiência variou de 0,15 (15%) para a menor carga até cerca de 0,33 (33%) para a maior. Isso indica que, apesar do aumento do aproveitamento dos recursos com o crescimento da carga, ainda há uma subutilização considerável dos recursos paralelos, possivelmente devido a gargalos como limitação de largura de banda da memória global, sincronizações frequentes ou baixa granularidade da carga em execuções menores. Em resumo, o uso de OpenCL trouxe ganhos significativos de desempenho, especialmente para cargas maiores, mas a eficiência revela espaço para melhorias na distribuição de tarefas e no aproveitamento da arquitetura da GPU.

5. Conclusão

A implementação do *Game of Life* com uso da tecnologia OpenCL demonstrou-se eficiente na aplicação de conceitos de paralelismo em GPUs, aproveitando seu potencial de processamento massivo para simulações de autômatos celulares. O desenvolvimento da aplicação contemplou todas as etapas essenciais para a execução paralela do algoritmo, desde a configuração do ambiente OpenCL, passando pela construção e execução do *kernel*, até o retorno e atualização dos dados no host.

Além de oferecer ganhos de desempenho, essa abordagem permite uma compreensão prática dos desafios e vantagens da programação heterogênea, promovendo um aprendizado sólido sobre o uso de GPUs em aplicações computacionalmente intensivas. A estrutura modular do código facilita sua reutilização e expansão para projetos mais complexos, servindo como base para pesquisas futuras em simulações paralelas e computação de alto desempenho.

Portanto, este trabalho contribui não apenas como uma solução técnica funcional, mas também como um recurso didático relevante para estudantes e profissionais interessados em explorar os limites da computação paralela com OpenCL.

References

Uni.lu HPC School, 2021. Introduction to OpenCL Programming (C/C++). Disponivel em: https://ulhpc--tutorials-readthedocs-io.translate.goog/en/latest/gpu/opencl/?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=tc. Acesso em 09 de maio de 2025.

Menotti, Ricardo (2020), Programação Paralela: OpenCL (demo). Disponível em: https://youtu.be/f0QQdROR3Wo?si=3sB1g9og8L0_AKO5. Acesso em 09 de maio de 2025.