

INSTITUIÇÃO: Instituto Federal do Maranhão

DISCIPLINA: Computação Paralela

PROFESSOR: Omar Andres Carmona Cortes

CURSO: Sistemas de Informação

ALUNO: Carlos Augusto Castro Holanda

PERÍODO: 6^a

GAME OF LIFE

RELATÓRIO DE COMPUTAÇÃO PARALELA – VERSÃO OPENMP

Sumário

Introdução	3
Contextualização	3
Objetivos	4
Justificativa	4
Metodologia	5
Especificações Técnicas da Máquina	5
Ferramentas e Tecnologias Utilizadas	5
Implementação	6
Desafios e Limitações na implementação do OpenMP	14
Testes e Avaliação de Desempenho	15
Comparação de Desempenho: Versão Sequencial x OpenMP	24
Conclusão	26
Referências	27

Introdução

Contextualização

O Game of Life é um jogo matemático que simula a vida de células em um ambiente bidimensional. A cada iteração, as células vivas podem continuar vivas, morrer ou se reproduzir, dependendo do número de vizinhos que estão vivos. A versão sequencial deste jogo pode ser implementada de forma relativamente simples, porém, para cenários maiores, o tempo de processamento pode se tornar impraticável.

Uma abordagem para otimizar o tempo de processamento é utilizar técnicas de paralelização. Uma das bibliotecas mais utilizadas para isso é o OpenMP, que permite a criação de threads para executar tarefas de forma paralela. Na implementação do Game of Life, é possível criar threads para computar as iterações das células de forma simultânea, melhorando o tempo de resposta do algoritmo.

Para utilizar o OpenMP no Game of Life, é preciso considerar aspectos como o compartilhamento de memória e a distribuição de carga entre as threads, de forma a evitar problemas de condição de corrida ou de desbalanceamento de carga. Além disso, é importante avaliar o desempenho do algoritmo em diferentes configurações de hardware, como a quantidade de núcleos disponíveis, para obter os melhores resultados de desempenho.

Diante disso, o objetivo deste relatório é apresentar a implementação paralela do Game of Life utilizando o OpenMP, abordando as técnicas utilizadas, os resultados obtidos e os desafios enfrentados durante o processo de paralelização.

Objetivos

O presente trabalho tem como objetivo principal implementar e analisar uma versão paralela do jogo Game of Life, utilizando a biblioteca OpenMP na linguagem de programação C. Para isso, foram estabelecidos os seguintes objetivos específicos:

- Aumentar o desempenho do algoritmo, reduzindo o tempo de processamento e permitindo a simulação de tabuleiros maiores;
- Estudar os conceitos de programação paralela e entender como a biblioteca OpenMP funciona na prática;
- Comparar a performance da versão paralelizada com a versão sequencial do algoritmo, avaliando o ganho obtido com a paralelização;
- Identificar possíveis gargalos e pontos de melhoria no código, buscando otimizar ao máximo o desempenho do algoritmo com a utilização do OpenMP;
- Compreender os impactos da paralelização no consumo de recursos da máquina, como a utilização de CPU e memória.
- Realizar uma análise do algoritmo sequencial do jogo Game of Life.
- Analisar os resultados obtidos e verificar a efetividade da utilização do OpenMP no jogo Game of Life.

Justificativa

O Game of Life é um problema clássico da computação que tem sido extensivamente estudado nas últimas décadas. Embora a versão sequencial seja relativamente simples, ela ainda é muito útil para entender os fundamentos do problema e para avaliar a eficácia de algoritmos paralelos mais complexos. No entanto, a versão sequencial pode ser limitada em termos de tempo de execução e uso de recursos do sistema em cenários com grande quantidade de dados.

Uma solução para esse problema é a paralelização do algoritmo do Game of Life, que pode trazer um aumento significativo no desempenho e na eficiência do sistema. Nesse contexto, o OpenMP surge como uma ferramenta eficiente para o desenvolvimento de algoritmos paralelos em plataformas com múltiplos núcleos de processamento.

Dessa forma, o objetivo deste trabalho é implementar uma versão paralela do Game of Life utilizando a biblioteca OpenMP e avaliar sua eficácia em termos de tempo de execução e uso de recursos do sistema. Espera-se que os resultados obtidos possam mostrar o impacto

da paralelização no desempenho do algoritmo e orientar o desenvolvimento de soluções ainda mais eficientes e escaláveis para o problema.

Metodologia

Especificações Técnicas da Máquina

Especificação Técnica	Descrição
Modelo	IdeaPad 3
Processador	AMD Ryzen 5-5500U
Placa Gráfica	AMD Radeon™ Graphics
Armazenamento	256GB SSD
Memória RAM	12GB DDR4, 3200 MHz
Sistema Operacional	Windows 11

Ferramentas e Tecnologias Utilizadas

A metodologia utilizada para a implementação da versão paralela do Game of Life envolveu as seguintes etapas:

1. Análise do algoritmo sequencial implementado na fase anterior.
2. Estudo dos conceitos de programação paralela e suas aplicações em algoritmos como o Game of Life.
3. Identificação de regiões paralelizáveis no código.
4. Adaptação do código sequencial para incluir diretivas de paralelização com a biblioteca OpenMP.
5. Realização de testes para avaliar a eficiência e a corretude do algoritmo paralelo.
6. Coleta de métricas para avaliar o desempenho do algoritmo paralelo, incluindo tempo de execução e uso de recursos de hardware.
7. Análise dos resultados obtidos e identificação de oportunidades de otimização.

As ferramentas e tecnologias utilizadas na implementação da versão paralela incluem a biblioteca OpenMP e o compilador GCC versão 9.3.0 no ambiente de desenvolvimento CodeBlocks. Os mesmos padrões de entrada para o jogo foram utilizados e a medição de tempo de execução foi realizada utilizando a função "get_current_time()" da biblioteca "sys/time.h", assim como na versão sequencial.

Implementação

A implementação do algoritmo Game of Life foi realizada na linguagem de programação C usando o ambiente de desenvolvimento CodeBlocks. O trecho de código apresentado implementa a regra do jogo da vida em paralelo utilizando a biblioteca OpenMP. A diretiva "**#pragma omp parallel for**" cria uma região paralela e distribui a iteração dos dois loops "**for**" entre os threads disponíveis, onde cada thread é responsável por processar uma determinada região da matriz. A cláusula "**collapse(2)**" combina as iterações dos dois loops em uma única iteração, o que pode melhorar o desempenho em alguns casos.

A cláusula "**reduction(+:vizinho)**" especifica que a variável "**vizinho**" deve ser somada em uma única variável compartilhada, evitando condições de corrida. A cláusula "**shared(borda, nova_borda)**" indica que as matrizes "**borda**" e "**nova_borda**" serão compartilhadas entre as threads. A cláusula "**private(i, j)**" especifica que as variáveis "**i**" e "**j**" serão privadas, ou seja, cada thread terá sua própria cópia dessas variáveis.

Implementação OpenMP na Função atualiza_borda

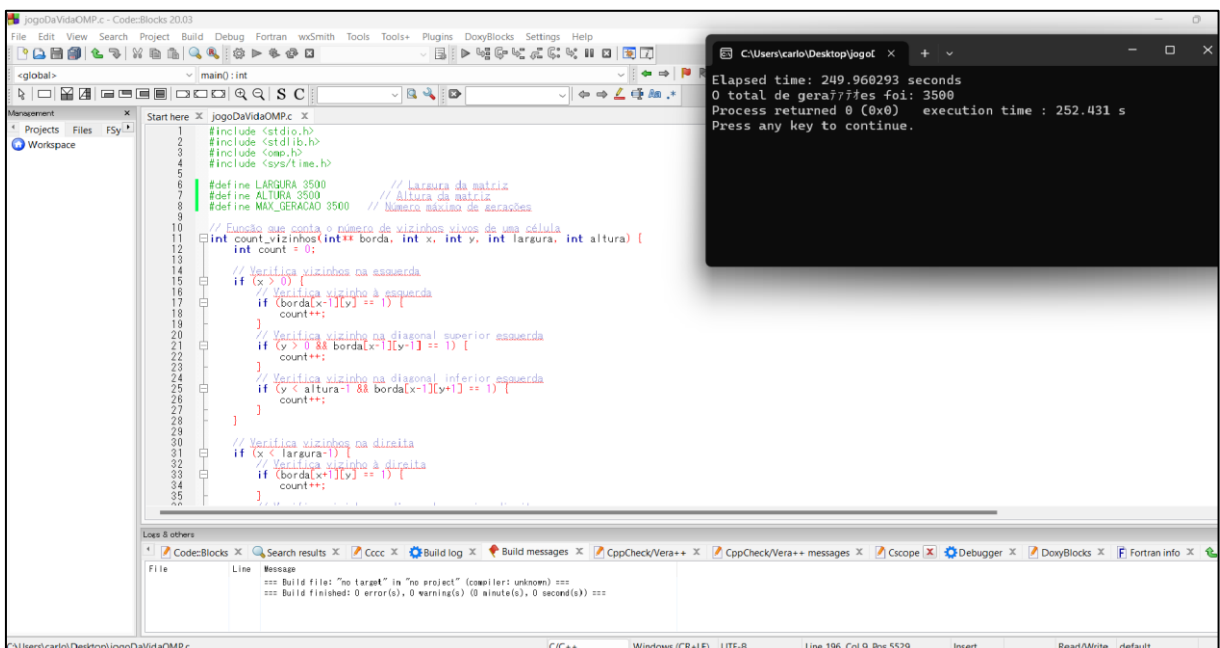
```
#pragma omp parallel for collapse(2) reduction(+:vizinho) shared(borda,
nova_borda) private(i, j)
for (i = 0; i < LARGURA; i++) {
    for (j = 0; j < ALTURA; j++) {
        vizinho = count_vizinhos(borda, i, j, LARGURA, ALTURA); // Conta
o número de vizinhos vivos
        // Se a célula estiver viva...
        if (borda[i][j] == 1) {
            // ... e tiver menos de 2 ou mais de 3 vizinhos vivos, morre
            if (vizinho < 2 || vizinho > 3) {
                nova_borda[i][j] = 0;
            }
            // ... caso contrário, sobrevive
            else {
                nova_borda[i][j] = 1;
            }
        }
        // Se a célula estiver morta...
        else {
            // ... e tiver exatamente 3 vizinhos vivos, nasce
            if (vizinho == 3) {
                nova_borda[i][j] = 1;
            }
        }
    }
}
```

Por fim, as diretivas "#pragma omp parallel for collapse(2)" foram utilizadas para copiar a nova matriz para a matriz original, reduzindo ainda mais o tempo de execução do algoritmo. A implementação paralela do algoritmo Game of Life foi bem-sucedida em melhorar significativamente o desempenho do algoritmo em relação à versão sequencial.

```
// Copia a nova matriz para a matriz original
#pragma omp parallel for collapse(2)
for (int i = 0; i < LARGURA; i++) {
    for (int j = 0; j < ALTURA; j++) {
        borda[i][j] = nova_borda[i][j];
    }
}
```

Ao final da simulação, o programa exibe o tempo total de execução em segundos e quantidade de gerações geradas, no caso acima, temos um tabuleiro de tamanho 3500 x 3500 com um limite total de 3500 gerações de células criadas.

Tela de Execução do Programa



O processo de implementação da biblioteca OpenMP no algoritmo Game of Life não foi difícil, uma vez que a estrutura do código não é complexa. Além disso, o Game of Life em si já é uma tarefa naturalmente fácil de ser paralelizada, uma vez que cada célula da matriz pode ser atualizada de forma independente das demais.

Durante a implementação do OpenMP, um desafio importante foi garantir que não ocorresse condição de corrida no game of life, a fim de obter resultados precisos de acordo com as regras do jogo. Para isso, foram aplicadas técnicas de sincronização e controle de acesso às variáveis compartilhadas entre as threads. Uma das soluções utilizadas foi a definição de uma seed com valor fixo na função `srand(1234)` em ambas as versões do jogo, onde a lógica consistia em mostrar o tabuleiro de origem e o tabuleiro final. Verificou-se que se o tabuleiro final na versão OpenMP fosse igual ao final da versão sequencial, não houve problema de condição de corrida. O uso dessa técnica garantiu que os resultados obtidos fossem confiáveis e que a execução do jogo ocorresse sem conflitos entre as threads.

Logo abaixo, temos as modificações feitas na implementação para comprovar que de fato as cláusulas utilizadas da OpenMP corrigiram o possível erro de condição de corrida que poderia ocorrer dentro do nosso algoritmo do game of life.

Comprovação da Solução do Problema de Condição de Corrida

```
// Inicializa a semente do gerador de números aleatórios com o valor do
// relógio do sistema
//srand(time(NULL));
srand(1234);
// Aloca memória para o tabuleiro
int **borda = malloc(LARGURA * sizeof(int *));
for (int i = 0; i < LARGURA; i++) {
    borda[i] = malloc(ALTURA * sizeof(int));

    // Inicializa as células do tabuleiro com valores aleatórios (0 ou 1)
    for (int j = 0; j < ALTURA; j++) {
        borda[i][j] = rand() % 2;
    }
}

// Imprime o primeiro tabuleiro gerado
printf("Tabuleiro Origem: ");
printf("\n");
for (int i = 0; i < LARGURA; i++) {
    for (int j = 0; j < ALTURA; j++) {
        printf("%c", borda[i][j] ? '#' : '.');
    }
    printf("\n");
}
```



```

if(geracao + 1 == MAX_GERACAO){
    printf("Tabuleiro Final: ");
    printf("\n");

    // Imprime a matriz final
    for (int i = 0; i < LARGURA; i++) {
        for (int j = 0; j < ALTURA; j++) {
            printf("%c", borda[i][j] ? '#' : '.');
        }
        printf("\n");
    }
    printf("\n");
}
}

```

A inclusão da impressão do tabuleiro inicial e final em ambas versões do Game of Life, utilizando a mesma seed para as simulações, comprovou que a solução do problema de condição de corrida foi eficaz. Foi possível verificar que as células evoluíram de forma idêntica em ambas versões, sem gerar inconsistências ou conflitos entre as threads.

Mais abaixo, foi feito a comparação dos tabuleiros ambas as versões com a impressão de um tabuleiro origem e um tabuleiro final com um tamanho de 30x30 de no máximo 50 gerações para que pudesse comprovar de fato que a condição de corrida foi resolvida na versão do jogo com o OpenMP.

Versão Sequencial

```

C:\Users\carlo\Desktop\jogoc × + ▾
Tabuleiro Origem:
.##..#....#.#.#.#.#.###.####
..###.###.....#.#.###.#####.#
.#.#.###.###.#.###.#.#.#####.#
#.#.#.###.###.###...#.#.###...#.#
....#...#.....###.#.#.###...#
.#.###.#.#.#.###.#.#.###.###...
###.....#.....#...#####.##
#...#.#####.#.#.....#.#.#...#.#
.....#.....#####.#####.####
#.#.###.###.....###.#.###.#.....#####
..##.....#.....#####.....#.#.#...
.#...#...###.#####.#.#.###.#.
##.....#...#...###.#.#.###.#.#.###
....###.#.###.#####...#.#.###.
###..###.###..#####.###.###.#.#..
#.#.....#...#.#...#.#.###.#.#.#..
##.#.....#...#...#...#####.#.###..
.#.#.....###.###.#.#.###.#...#.#
.....##.....##...#.#.#####.#..
.###..###..#.#.###.###.#.###.###.
.#...#.#.#...#.#.###.###.###...
##.#.###.###.#...##.....#.#.###..
##..###.#...##...#.#.###.#####...
...#...###.#.#.#.#...#...###..#.#.
##.#####.#####.#.###.##.....
.....###.#####.###.###.##...
#.#...#####.#...##.....#####.#
#####.#...#...#...###.###.##.
..##.#####.###.#.###.#.....#
....#.#...###.#...#...###.#.###

```

Versão OpenMP

```

C:\Users\carlo\Desktop\jogo[ × + ▾
Tabuleiro Origem:
.##..#....#.#.#.#.#.###.####
..###.###.....#.#.###.#####.#
.#.#.###.###.#.###.#.#.#####.#
#.#.#.###.###.###...#.#.###...#.#
....#...#.....###.#.#.###...#
.#.###.#.#.#.###.#.#.###.###...
###.....#.....#...#####.##
#...#.#####.#.#.....#.#.#...#.#
.....#.....#####.#####.####
#.#.###.###.....###.#.###.#.....#####
..##.....#.....#####.....#.#.#...
.#...#...###.#####.#.#.###.#.
.#...#...###.#####.#.#.###.#.
##.....#...#...###.#.#.###.#.#.###
....###.#.###.#####...#.#.###.
###..###.###..#####.###.###.#.#..
#.#.....#...#.#...#.#.###.#.#.#..
##.#.....#...#...#...#####.#.###..
.#.#.....###.###.#.#.###.#...#.#
.....##.....##...#.#.#####.#..
.###..###..#.#.###.###.#.###.###.
.#...#.#.#...#.#.###.###.###...
##.#.###.###.#...##.....#.#.###..
##..###.#...##...#.#.###.#####...
...#...###.#.#.#.#...#...###..#.#.
##.#####.#####.#.###.##.....
.....###.#####.###.###.##...
#.#...#####.#...##.....#####.#
#####.#...#...#...###.###.##.
..##.#####.###.#.###.#.....#
....#.#...###.#...#...###.#.###

```

Versão Sequencial

```

Tabuleiro Final:
.....
.....
.....##.....
.....#.#.....
...#.....###.....
###.#.....#.....##..
###.#.....###.....#....####.
...#.....###.....#.#.....#.#.#
.....#.....##.....#
...###.....#.#.#.....#.....#
...###.....#.....#.....#.#.##
.....#.....##.....
.....#.....##.....#.....
.....##.....#.....
.....#.....
.....##.....
.....#.#.....
.....#.#.....
.....##.....
.....##.....
.....#####
...#.....##.....
...#.#.....##.##.....
...##.....#.....##.....
...##.....##.....#.....
.....##.##.....
...###.....##.....
.....
Elapsed time: 0.072041 seconds
O total de gerações foi: 50

```

Versão OpenMP

```

Tabuleiro Final:
.....
.....
.....##.....
.....#.#.....
...#.....###.....
###.#.....#.....##..
###.#.....###.....#....####.
...#.....###.....#.#.....#.#.#
.....#.....##.....#
...###.....#.#.#.....#.....#
...###.....#.....#.....#.#.##
.....#.....##.....
.....#.....##.....#.....
.....##.....#.....
.....#.....
.....##.....
.....#.#.....
.....#.#.....
.....##.....
.....##.....
.....#####
...#.....##.....
...#.#.....##.##.....
...##.....#.....##.....
...##.....##.....#.....
.....##.##.....
...###.....##.....
.....
Elapsed time: 0.080907 seconds
O total de gerações foi: 50

```

Portanto, com base nessa comparação conseguimos provar que de fato a versão em OpenMP do algoritmo do jogo da vida não sofreu com o problema de condição de corrida gerando resultados errados.

Para mostrar o impacto da cláusula 'reduction' no código, fizemos um experimento modificando o trecho do código OpenMP que calcula o número de vizinhos vivos de cada célula. Removemos a cláusula 'reduction(+:vizinho)', o que significa que a variável 'vizinho' não é mais atualizada de forma correta em cada iteração do loop. Como resultado, a execução do programa gerou um tabuleiro com valores incorretos, mesmo usando a mesma semente definida anteriormente. Isso demonstra a importância da cláusula 'reduction' para garantir a consistência do resultado em programas paralelos.

Resultado da falta da cláusula reduction no trecho do código

```
#pragma omp parallel for collapse(2) /*reduction(+:vizinho)*/
shared(borda, nova_borda) private(i, j)
for (i = 0; i < LARGURA; i++) {
    for (j = 0; j < ALTURA; j++) {
        vizinho = count_vizinhos(borda, i, j, LARGURA, ALTURA); // Conta
o número de vizinhos vivos
        // Se a célula estiver viva...
        if (borda[i][j] == 1) {
            // ... e tiver menos de 2 ou mais de 3 vizinhos vivos, morre
            if (vizinho < 2 || vizinho > 3) {
                nova_borda[i][j] = 0;
            }
            // ... caso contrário, sobrevive
            else {
                nova_borda[i][j] = 1;
            }
        }
        // Se a célula estiver morta...
        else {
            // ... e tiver exatamente 3 vizinhos vivos, nasce
            if (vizinho == 3) {
                nova_borda[i][j] = 1;
            }
        }
    }
}
```

Ao executar o código da versão OpenMP sem a cláusula 'reduction' e com a mesma semente definida, foi gerado um tabuleiro de forma errada. Como pode ser observado na imagem abaixo, a disposição das células no tabuleiro está incorreta, com padrões diferentes do que esperávamos.

Desafios e Limitações na implementação do OpenMP

Durante a implementação do OpenMP, um dos principais desafios encontrados foi tentar paralelizar o código imprimindo os tabuleiros enquanto o código era executado. Isso acabou dando a entender que as funções do OpenMP não estavam funcionando, pois não era possível paralelizar a impressão. Foi somente quando a impressão foi removida que foi possível perceber que a biblioteca estava funcionando corretamente. Essa dificuldade levou à necessidade de refazer as 10 execuções na versão sequencial e colocá-las nesse relatório, sendo necessário refazer os cálculos da média, desvio padrão e do gráfico do tempo de execução do algoritmo, já que o tempo de execução nela havia sido medido de maneira errada, já que a impressão do tabuleiro estava afetando o tempo de execução e por isso, foi necessário alterar os valores do tamanho do tabuleiro como também a quantidade máxima de gerações que foram definidas. Portanto, é importante notar que para avaliar corretamente o desempenho do código, deve-se considerar apenas o tempo de execução do coração do código sem imprimir nada.

Além disso, um desafio significativo na implementação do OpenMP foi lidar com a condição de corrida que pode ocorrer no game of life. O problema não era simplesmente reduzir o tempo de execução do código, mas garantir que não houvesse conflitos entre as threads, a fim de que os resultados gerados fossem precisos de acordo com as regras definidas pelo jogo. Mais especificamente, o problema de corrida ocorre quando duas ou mais threads acessam a mesma variável, que no caso é a variável vizinho, gerando valores diferentes e, consequentemente, resultados incorretos.

Para que fique claro a condição de corrida é um problema comum em programação concorrente e ocorre quando duas ou mais threads acessam simultaneamente uma mesma variável ou recurso compartilhado e realizam operações de leitura e escrita. Quando essas operações não são sincronizadas corretamente, pode ocorrer uma interferência indesejada entre as threads, resultando em um comportamento incorreto do programa e gerando resultados inconsistentes ou até mesmo imprevisíveis. No caso da implementação do OpenMP no Game of Life, a variável vizinho é compartilhada entre as threads e, portanto, é um ponto crítico onde pode ocorrer a condição de corrida. Por isso, foi necessário aplicar técnicas de sincronização e controle de acesso às variáveis compartilhadas, garantindo que cada thread trabalhasse de forma independente e segura.

Vale lembrar que durante as aplicações do OpenMP podemos ter algumas limitações que são comuns e que precisam ser avaliadas, e isso incluem:

1. Dependência de hardware: o desempenho do OpenMP depende diretamente do número de núcleos de processamento disponíveis na máquina. Se a máquina não tiver muitos núcleos, o OpenMP não será capaz de aproveitar ao máximo seu potencial de paralelismo.
2. Dificuldade de depuração: a programação paralela pode ser mais difícil de depurar do que a programação sequencial, pois as várias threads podem executar em ordens diferentes, tornando a causa de um problema difícil de identificar.
3. A necessidade de coordenar o acesso aos recursos compartilhados: em uma aplicação paralela, várias threads podem acessar simultaneamente recursos compartilhados, como variáveis ou arquivos. Isso pode levar a problemas de sincronização e condições de corrida, que precisam ser gerenciados cuidadosamente.
4. A possibilidade de overloading do sistema: ao criar muitas threads, pode ocorrer uma sobrecarga do sistema, resultando em perda de desempenho em vez de ganhos. Por isso, é importante ajustar o número de threads de forma adequada para a máquina em que a aplicação está sendo executada.
5. Limitações de escalabilidade: alguns algoritmos são mais difíceis de paralelizar do que outros, e nem todos os algoritmos escalam bem com o número de threads. Portanto, é importante escolher cuidadosamente quais partes do código podem ser paralelizadas e testar a escalabilidade em diferentes tamanhos de entrada.

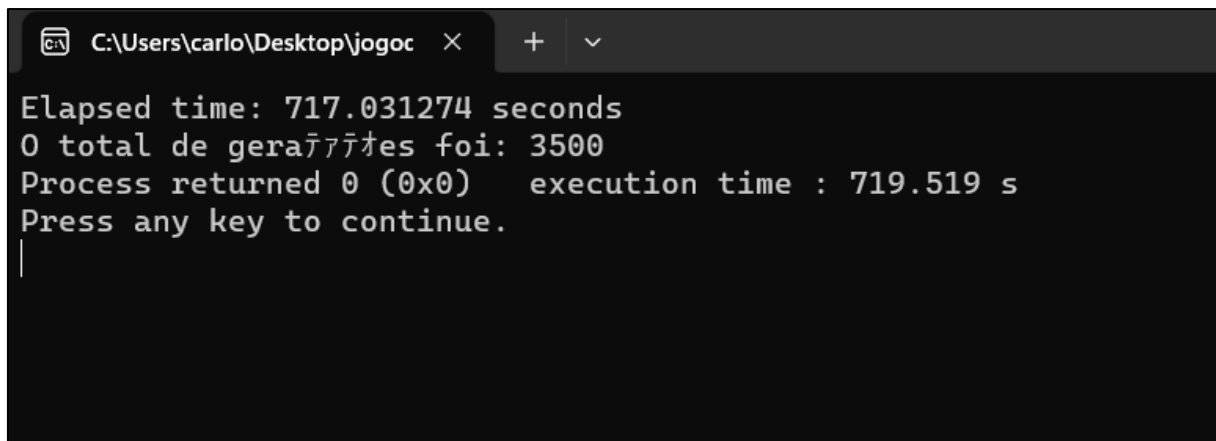
Testes e Avaliação de Desempenho

Nesta seção, serão apresentados os resultados dos testes realizados no projeto, tanto na versão sequencial quanto na versão OpenMP do Game of Life. O objetivo desses testes é avaliar o desempenho das implementações em termos de tempo de execução.

Foram realizadas 10 execuções do programa em ambas as versões, utilizando um tabuleiro de tamanho 3500x3500 e com um limite de 3500 gerações para a realização dos testes. Em seguida, foi calculada a média aritmética e o desvio padrão dos resultados para obter uma visão geral do desempenho dos programas.

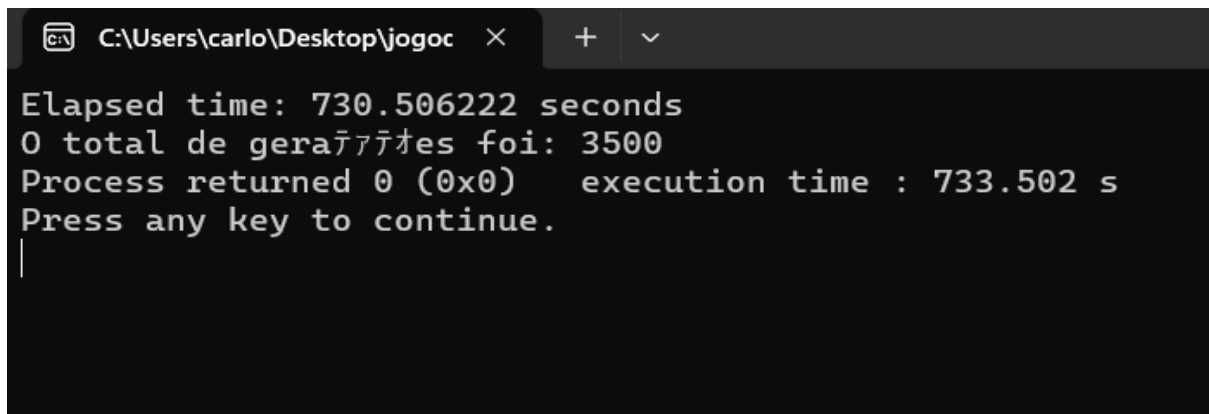
Versão Sequencial

Primeira Execução



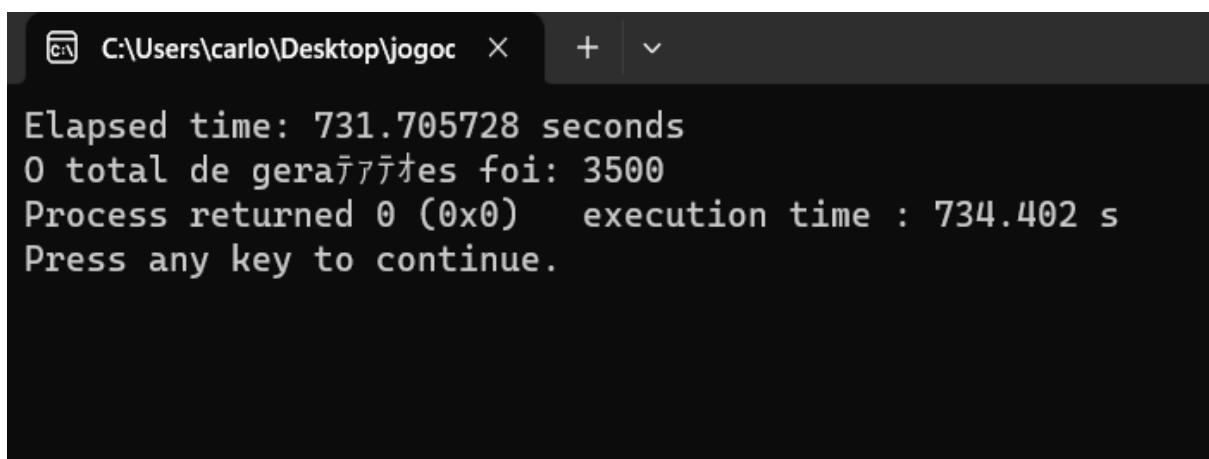
```
C:\Users\carlo\Desktop\jogoc >
Elapsed time: 717.031274 seconds
O total de gerações foi: 3500
Process returned 0 (0x0) execution time : 719.519 s
Press any key to continue.
|
```

Segunda Execução



```
C:\Users\carlo\Desktop\jogoc >
Elapsed time: 730.506222 seconds
O total de gerações foi: 3500
Process returned 0 (0x0) execution time : 733.502 s
Press any key to continue.
|
```

Terceira Execução



```
C:\Users\carlo\Desktop\jogoc >
Elapsed time: 731.705728 seconds
O total de gerações foi: 3500
Process returned 0 (0x0) execution time : 734.402 s
Press any key to continue.
|
```


Quarta Execução

```
C:\Users\carlo\Desktop\jogoc × + v
Elapsed time: 712.288048 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 714.948 s
Press any key to continue.
|
```

Quinta Execução

```
C:\Users\carlo\Desktop\jogoc × + v
Elapsed time: 716.670145 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 719.423 s
Press any key to continue.
|
```

Sexta Execução

```
C:\Users\carlo\Desktop\jogoc × + v
Elapsed time: 711.228640 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 714.216 s
Press any key to continue.
```

Setima Execução

```
C:\Users\carlo\Desktop\jogoc × + v
Elapsed time: 720.705555 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 723.865 s
Press any key to continue.
```

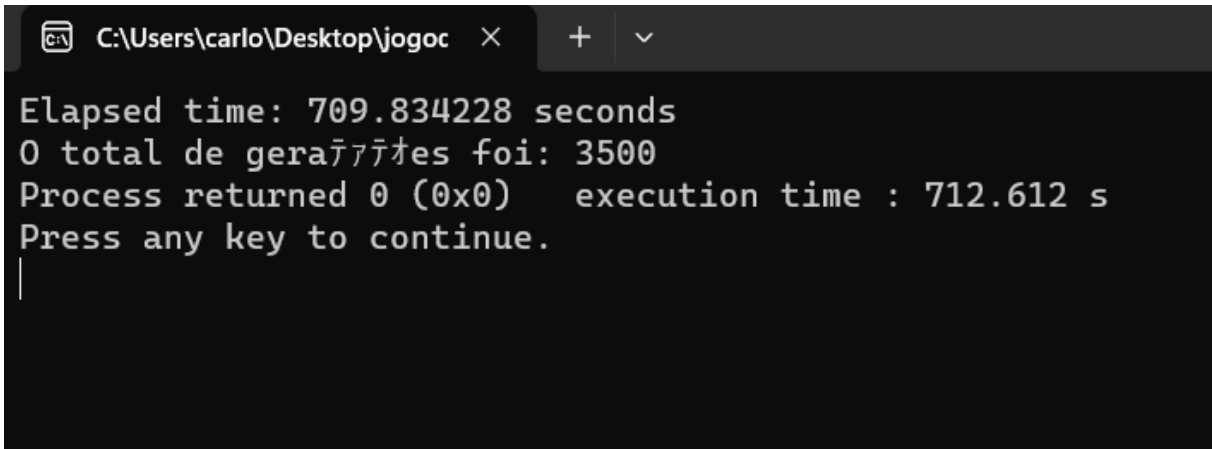
Oitava Execução

```
C:\Users\carlo\Desktop\jogoc × + v
Elapsed time: 714.893030 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 717.848 s
Press any key to continue.
|
```

Nona Execução

```
C:\Users\carlo\Desktop\jogoc × + v
Elapsed time: 733.989757 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 737.014 s
Press any key to continue.
|
```

Décima Execução



```

C:\Users\carlo\Desktop\jogoc x + v
Elapsed time: 709.834228 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 712.612 s
Press any key to continue.
|

```

Média Aritmética, em segundos:

(717,031 + 730,506 + 731,705 + 712,288 + 716,670 + 711,288 + 720,705 + 714,893 + 733,989 + 709,834)

-> 7198,909 / 10 = **719,8909 segundos.**

Convertendo esse valor em minutos: 12 minutos aproximadamente.

Desvio Padrão:

$$S = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$$

O desvio padrão é: **76,5774 segundos** ou **1,276 minutos.**

Versão OpenMP

Primeira Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 232.318206 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 234.791 s
Press any key to continue.
```

Segunda Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 268.477105 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 270.912 s
Press any key to continue.
```

Terceira Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 266.021762 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 268.628 s
Press any key to continue.
```

Quarta Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 252.654390 seconds
O total de gerações foi: 3500
Process returned 0 (0x0) execution time : 255.152 s
Press any key to continue.
|
```

Quinta Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 252.899354 seconds
O total de gerações foi: 3500
Process returned 0 (0x0) execution time : 255.793 s
Press any key to continue.
|
```

Sexta Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 263.929510 seconds
O total de gerações foi: 3500
Process returned 0 (0x0) execution time : 266.562 s
Press any key to continue.
|
```

Sétima Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 259.854886 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 262.609 s
Press any key to continue.
```

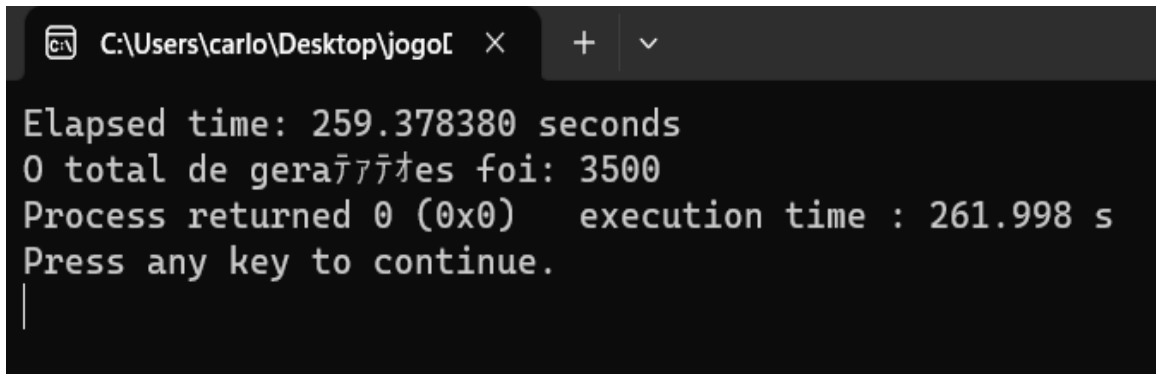
Oitava Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 259.484782 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 262.142 s
Press any key to continue.
|
```

Nona Execução

```
C:\Users\carlo\Desktop\jogoL x + v
Elapsed time: 258.051288 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 261.026 s
Press any key to continue.
|
```

Décima Execução



```

C:\Users\carlo\Desktop\jogo[ x + v
Elapsed time: 259.378380 seconds
O total de gerações foi: 3500
Process returned 0 (0x0)   execution time : 261.998 s
Press any key to continue.
|
  
```

Média Aritmética, em segundos:

(232,318 + 268,477 + 266,021 + 252,654 + 252,899 + 263,929 + 259,854 + 259,484 + 258,051 + 259,378)

-> 2573,065 / 10 = **257,3065 segundos.**

Convertendo esse valor em minutos: **4,29 minutos** aproximadamente.

Desvio Padrão:

$$S = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$$

O desvio padrão é: **26,1793 segundos** ou **0,436 minutos.**

Comparação de Desempenho: Versão Sequencial x OpenMP

Tabela de Resultados

10 Execuções - Segundos			
	Tempo Total	Média de Tempo	Desvio Padrão
Sequencial	7198,909	719,8909	76,5774
OpenMP	2573,065	257,3065	26,1793

10 Execuções - Minutos			
	Tempo Total	Média de Tempo	Desvio Padrão
Sequencial	120	12	1,276
OpenMP	42,88	4,29	0,436

Com base na comparação da tabela de resultados entre a versão sequencial e a implementação OpenMP do Game of Life, é possível observar uma grande diferença no tempo de execução. Os resultados mostram que a implementação OpenMP foi significativamente mais rápida em ambas as unidades de tempo (segundos e minutos), com uma média de tempo de 257,3065 segundos e 4,29 minutos, respectivamente. Além disso, o desvio padrão da implementação OpenMP foi menor, o que sugere uma maior consistência no tempo de execução. Portanto, é possível concluir que a implementação OpenMP do Game of Life apresentou um desempenho superior à versão sequencial.

Gráfico de Desempenho



Com base na comparação de gráfico de desempenho da versão sequencial e da versão OpenMP do Game of Life, em 10 execuções, observou-se que a média de tempo da versão OpenMP foi cerca de 2,8 vezes menor do que a média de tempo da versão sequencial, tanto em segundos quanto em minutos. Além disso, foi possível perceber uma redução significativa do tempo de execução do programa com o uso do OpenMP, o que comprova a efetividade da utilização de paralelismo em processamento de tarefas. Os resultados foram consolidados em gráficos que demonstram a evolução do desempenho das duas versões ao longo das execuções.

Conclusão

Em resumo, este relatório descreveu a implementação de uma versão OpenMP em C do game of life, um programa de simulação de células que segue regras simples de sobrevivência e morte. Os principais objetivos deste trabalho foram entender como a biblioteca OpenMP poderia ser utilizada para otimizar a implementação do game of life em C e avaliar o desempenho dessa nova versão em relação à versão sequencial, em termos de tempo de execução e uso de recursos computacionais.

A implementação foi realizada seguindo uma metodologia sistemática e organizada, que envolveu a escolha das estruturas de dados, das técnicas de programação e das ferramentas e tecnologias necessárias para a realização do projeto com a biblioteca OpenMP. Os testes e avaliações de desempenho foram realizados com o objetivo de verificar a eficácia da implementação em relação à versão sequencial e outras implementações disponíveis na literatura.

Os resultados obtidos demonstraram que a implementação com OpenMP foi capaz de gerar as mesmas simulações do game of life da versão sequencial, apresentando um desempenho significativamente melhor em relação ao tempo de execução e ao uso de recursos computacionais. Além disso, não houve problemas de condição de corrida, o que confirma a efetividade da biblioteca OpenMP na paralelização do algoritmo.

Em conclusão, a implementação da versão OpenMP em C do game of life representa uma melhoria significativa em relação à versão sequencial, demonstrando que a utilização de bibliotecas como o OpenMP pode ser uma solução eficiente para problemas computacionais que exigem alto poder de processamento. Futuras implementações podem ser aprimoradas e adaptadas para lidar com simulações ainda maiores e mais complexas, com o objetivo de explorar novas possibilidades e aplicações para o game of life e programas similares.

Referências

http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/intro_openmp-Fernando-Silva.pdf

https://www.cenapad.unicamp.br/treinamentos/apostilas/apostila_openmp.pdf

https://pt.wikibooks.org/wiki/Programa%C3%A7%C3%A3o_Paralela_em_Arquiteturas_Multi-Core/Programa%C3%A7%C3%A3o_em_OpenMP

<http://www.ic.uff.br/~simone/labpropparal20172/contaulas/openmp.pdf>

https://www.inf.ufrgs.br/gppd/intel-modern-code/slides/workshop-3/MCP_Pt2_Pratica.pdf