# Paralelização de computação e escrita em aplicações de simulação de propagação de ondas utilizando o padrão de comunicação MPI

**Rodrigo C. Machado, Arthur F. Lorenzon, Philippe O. A. Navaux**

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS) – Porto Alegre – RS – Brazil

{rcmachado,aflorenzon,navaux}@inf.ufrgs.br

***Resumo.*** *Este artigo tem como objetivo verificar os impactos da utilização das funções bloqueantes MPI_Send e MPI_Recv do padrão MPI de troca de mensagens para a paralelização de escrita e computação de uma aplicação rodando em um mesmo nó. A aplicação original simula a propagação de ondas na água, sendo programada de forma que a escrita e computação são executadas sequencialmente. Com a utilização da biblioteca Open MPI separamos a escrita em disco e a computação em GPU.*

## 1. Introdução

A exploração de óleo e gás é fundamental para o desenvolvimento humano, mas traz consigo desafios quanto ao meio ambiente, como a perfuração do solo de áreas de sensibilidade ambiental e o descarte de materiais residuais. Assim, com o objetivo de melhorar a sustentabilidade do processo de descoberta de recursos naturais, a indústria busca técnicas para reduzir o impacto ambiental utilizando *softwares* que possam fazer o mapeamento geológico de áreas subaquáticas. Este tipo de mapeamento requer o processamento de grandes volumes de dados, com isso é necessário utilizar computação de alto desempenho [Navaux et al. 2023].

O método *Fletcher* é uma aplicação que faz a simulação da propagação de ondas no tempo e gera imagens do fundo do oceano possibilitando encontrar locais onde pode haver petróleo e gás [Fletcher et al. 2009]. Durante esse processo é gerada uma grande quantidade de dados a serem armazenados em disco, aumentando consideravelmente o tempo total de execução. Neste cenário, a otimização simultânea da escrita e computação possibilita que o tempo total de alocação de recursos computacionais possa ser reduzida.

No entanto, nenhuma implementação do método *Fletcher* disponível atualmente realiza a sobreposição de computação com operações de E/S. Sendo assim, este trabalho propõe a utilização do padrão de troca de mensagens *MPI* [Message Passing Interface Forum 2015] através da biblioteca *Open MPI* para otimizar o desempenho da aplicação sobrepondo a escrita em disco com o fluxo de computação na GPU [Open MPI 2018] . Nós avaliamos a execução das versões original e *MPI* do método *Fletcher*, realizando testes utilizando parâmetros que geram saídas de 60 e 90 GB a serem escritos em disco. As duas versões foram testadas com a escrita sendo feita em HD e SSD. A versão *MPI* reduziu o tempo de execução em 11,6% em relação ao original nos testes feitos no SSD para os dois tamanhos de saída.

## 2. Método Fletcher

O método *Fletcher* simula a propagação de ondas no tempo. Seu pseudocódigo é descrito no Algorithm 1. A implementação desta aplicação recebe como parâmetro de en-

trada o tamanho da *grid* (*tamanhoGrid*) onde é feita a simulação, o passo de tempo entre propagações (*passo*) e o tempo total de simulação (*tempoTotal*). A aplicação começa inicializando a *grid* (*inicializaGrid*) e salvando o estado em disco (*gravaEmDisco*). Para cada iteração da aplicação é feita a simulação da propagação da onda naquele instante (*propagacaoOnda*). Sempre que a quantidade de tempo decorrido alcança ou passa um limite (*limiteParaEscrita*), inicialmente 0.01, o estado atual da *grid* é salvo em disco. O próximo limite de tempo para a próxima escrita é calculado multiplicando 0.01 e a quantidade de escritas em disco (*escritasEmDisco*).

**Algorithm 1** Pseudocódigo Fletcher Original

```
1:  function FLETCHER(tamanhoGrid, passo, tempoTotal)
2:      grid ← inicializaGrid(tamanhoGrid)
3:      limiteParaEscrita ← 0.01
4:      gravaEmDisco(grid)
5:      escritasEmDisco ← 1
6:      for it ← 1, it ≤ ceil(tempoTotal/passo), it++ do
7:          propagacaoOnda(grid)
8:          if it * passo ≥ limiteParaEscrita then
9:              gravaEmDisco(grid)
10:             escritasEmDisco++
11:             limiteParaEscrita ← 0.01 * escritasEmDisco
12:         end if
13:     end for
14: end function
```

## 3. Proposta

Para otimizar o tempo total de execução da aplicação nós modificamos a aplicação separando a escrita em disco e a computação utilizando as funções *MPI_Send* e *MPI_Recv* da biblioteca *Open MPI*. Na Figure 1 ilustramos na esquerda como é o fluxo da aplicação original, onde a computação e escrita são executados sequencialmente. No lado direito mostramos como é a computação utilizando MPI, aqui a escrita em disco é delegada a um processo diferente do que o que realiza a computação. Nessa versão, dois processos *MPI* (*RANK* 0 e *RANK* 1) são criados de maneira estática (isto é, ao iniciar a execução, através do comando *mpirun -n 2*). Estes processos utilizam a mesma estrutura da versão original, mas com comportamentos diferentes para cada *RANK*. O *RANK* 0, em vez de chamar a função *gravaEmDisco*, enviará a *grid* para o *RANK* 1 utilizando *MPI_Send*. Já o *RANK* 1 não realiza computação (*propagacaoOnda*), somente recebe os dados com *MPI_Recv* e os grava em disco.

## 4. Análise Experimental

Os testes foram feitos utilizando um tamanho fixo da *grid* com dimensão 504x504x504, passo de tempo como 0.001, variando o tempo total de simulação entre 1 e 1.5 (o que gera saídas de 60 e 90 GB respectivamente) e o tipo de disco entre SSD e HD. Foram coletados dados rodando 10 vezes cada uma das 4 configurações para as versões original e *MPI*. A máquina onde foram realizados os testes possui uma GPU P100 de 3584 CUDA cores, 256GB de RAM DDR4, um SSD Samsung 850 EVO de 1 TB com velocidade de escrita 520 MB/s e um HD Seagate 7E2000 de 2.5 TB com velocidade de escrita de 136 MB/s.

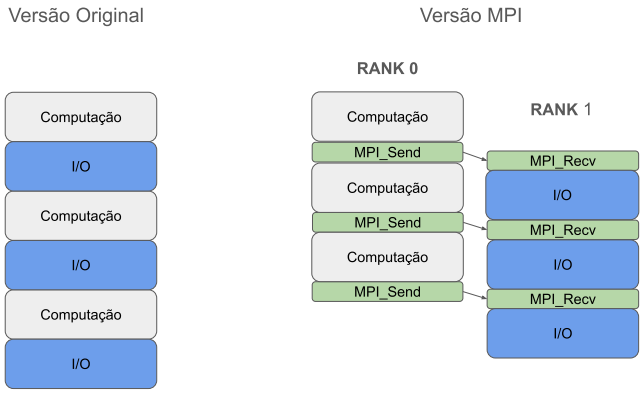


**Figura 1. Fluxo de execução Original x MPI**

Analisando primeiramente os resultados referentes às execuções com escrita em disco rígido (Figura 2), a versão *MPI* obteve um *speedup* para ambos os tamanhos de escrita. O *speedup* da versão *MPI* no teste que escreve 60 GB é de 1,091 em relação ao tempo da versão original, com o desvio padrão das versões sendo, respectivamente, 5,67 e 16,47 segundos. Já no teste com escrita de 90 GB o *speedup* da versão *MPI* diminui, sendo de 1,02 em relação a versão original, o desvio padrão destas versões é de, respectivamente, 40,45 e 37,59 segundos . A diminuição no *speedup* se deve à saturação dos *buffers* do disco rígido devido ao grande volume de dados escritos.

Nas execuções com escritas em SSD (Figura 3), a versão *MPI* obteve *speedups* parecidos para ambos os tamanhos de escrita. Para escrita de 60 GB o *speedup* foi de 1,131 e na de 90 GB de 1,132, com desvio padrão de 0,5 e 0,97 segundos, respectivamente, para a versão *MPI*, e de 1,54 segundos (60 GB) e 1,16 segundos (90 GB) para a versão Original.

A menor variabilidade dos resultados obtidos em execuções utilizando SSD se deve às características físicas de ambos dispositivos. No disco rígido, a cabeça de escrita e leitura deve se mover e esperar a rotação do disco chegar na parte onde serão feitas as escritas. Já o SSD possui uma memória *flash*, sem partes móveis. Os maiores tempos de escrita do HD se devem às menores taxas de escrita e à saturação do *buffer* de escrita. A saturação do *buffer* é uma consequência das menores taxas de escrita, já que muitos dados precisam ser escritos enquanto o disco está operando em sua capacidade máxima.

## 5. Conclusão

Com o aumento da demanda por soluções que utilizam computação de alto desempenho para o processamento e geração de grandes volumes de dados, se faz necessário otimizar a utilização de recursos computacionais. A paralelização dos fluxos de computação e escrita em disco é uma das formas que podem ser utilizadas para otimizar estes recursos. Com a biblioteca *Open MPI*, conseguimos obter bons resultados de *speedup* sem aumentar a complexidade da programação. A escolha do tipo de dispositivo de armazenamento tem grande influência no quanto podemos reduzir o tempo de execução das aplicações. Ficou muito claro que discos rígidos não são ideias para aplicações que exigem a escrita intensa de dados, pois suas taxas de escrita são pequenas e variam muito. Por outro lado, SSDs possuem uma maior estabilidade, além de maiores taxas de escrita, o que possibilitou ver

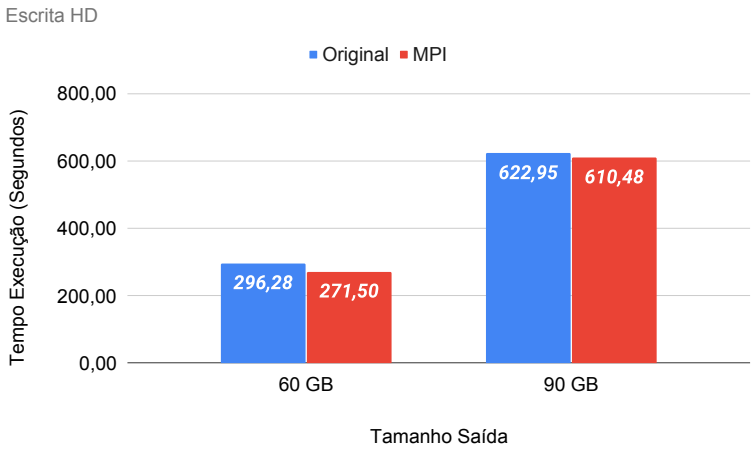


**Figura 2. Execução em HD: Original x MPI**

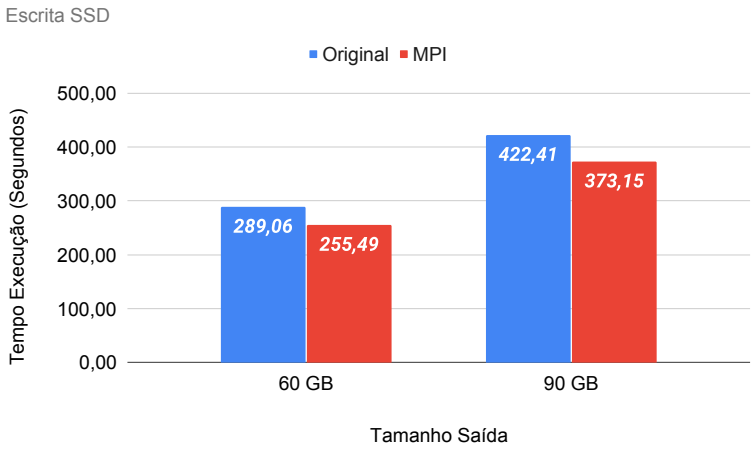


**Figura 3. Execução em SSD: Original x MPI**

*speedups* consistentes entre todas as execuções.

## Referências

Fletcher, R. P., Du, X., and Fowler, P. J. (2009). Reverse time migration in tilted transversely isotropic (TTI) media. *Geophysics*, 74(6):179–187.

Message Passing Interface Forum (2015). *MPI: A Message-Passing Interface Standard Version 3.1*.

Navaux, P. O. A., Lorenzon, A. F., and da Silva Serpa, M. (2023). Challenges in high-performance computing. *Journal of the Brazilian Computer Society*, 29(1):51–62.

Open MPI (2018). Open MPI: Open Source High Performance Computing. Acessado em 16/02/2024.