UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RODRIGO COSTA MACHADO

Sobreposição de computação e E/S do método Fletcher utilizando MPI

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Philipe O. A. Navaux
Co-orientador: Prof. Dr. Arthur Lorenzon

Porto Alegre
Julho 2024

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões
Vice-Reitora: Prof[a]. Patricia Pranke
Pró-Reitor de Graduação: Prof[a]. Cíntia Inês Boll
Diretora do Instituto de Informática: Prof[a]. Carla Maria Dal Sasso Freitas
Coordenador do Curso de Ciência de Computação: Prof. Marcelo Walter
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro

RESUMO

O método *Fletcher* é um algoritmo que simula a propagação de ondas sísmicas em função do tempo. Uma das implementações do método trabalha com computação em *GPU*, e realiza escritas periódicas em disco. O volume de dados a ser salvo é potencialmente grande, fazendo com que a E/S se torne um gargalo para esse tipo de aplicação. Neste trabalho, são propostas soluções para sobrepor a escrita e a computação do método *Fletcher* utilizando o padrão *MPI*. Com a utilização do *MPI*, foram desenvolvidas quatro soluções diferentes que exploram diversas abordagens e funcionalidades do *MPI*, tais como trocas de mensagens bloqueantes e não bloqueantes, criação estática e dinâmica de processos e a utilização da funcionalidade *MPI I/O*. A funcionalidade *MPI I/O* permite que diferentes processos tenham acesso a um mesmo arquivo, possibilitando a leitura e escrita paralela nesse arquivo. Isso pode reduzir o tempo de E/S, melhorando o desempenho da aplicação.

Palavras-chave: Paralelização. MPI. MPI I/O. Fletcher. Óleo e gás. E/S.

Parallelization of I/O and Computation in the Fletcher Method Using MPI

ABSTRACT

The Fletcher method is an algorithm that simulates the propagation of seismic waves over time.In one of its implementations, the method utilizes GPU computing to run the simulations. It generates data at each iteration, which needs to be stored periodically during the simulation. The volume of data to be saved is potentially large, making I/O a bottleneck for this type of application. In this work, four solutions are proposed to overlap the writing and computing of the Fletcher method using the MPI standard. Those solutions explore various approaches and functionalities of MPI, such as blocking and non-blocking message exchanges, static and dynamic process creation, and the use of MPI I/O functionality. The MPI I/O functionality allows different processes to access the same file, enabling parallel reading and writing to that file. This can reduce I/O time, improving application performance.

Keywords: ,parallelization, MPI, MPI I/O, Fletcher, oil and gas.

LISTA DE FIGURAS

Figura 1.1 Fluxo original x sobreposto 10

Figura 2.1 Formas de criação de processos.. 13
Figura 2.2 MPI-IO: Escrita paralela com *MPI_File_write_at* 15

Figura 3.1 Versões bloqueante e não-bloqueante.. 17
Figura 3.2 *all_at_once*: criação dinâmica de processos com escrita paralela 18
Figura 3.3 *one_at_time*: criação dinâmica de processos com múltiplos arquivos 19

Figura 4.1 Tempo médio de execução das versões *original*, *send_recv* e *isend_recv* no *HD* 23
Figura 4.2 Speedup das versões *send_recv* e *isend_recv* no *HD* 23
Figura 4.3 Desvio padrão das versões *original*, *send_recv* e *isend_recv* no *HD* 24
Figura 4.4 Tempo médio de execução das versões *original*, *send_recv* e *isend_recv* no *SSD* 25
Figura 4.5 Speedup das versões *send_recv* e *isend_recv* no *SSD* 26
Figura 4.6 Desvio padrão das versões *original*, *send_recv* e *isend_recv* no *SSD* 26
Figura 4.7 Tempo médio de execução das versões *original* e instâncias de *all_at_once* com 2, 4 e 8 processos escritores no *HD* 28
Figura 4.8 *Speedup* da *all_at_once* com 2, 4 e 8 processos escritores no *HD* 28
Figura 4.9 Desvio padrão dos tempos de execução das instâncias de *all_at_once* com 2, 4 e 8 processos escritores no *HD* 29
Figura 4.10 Tempo médio de execução das versões *original* e instâncias de *one_at_time* com 2, 4 e 8 processos escritores no *HD* 30
Figura 4.11 *Speedup* de *one_at_time* com 2, 4 e 8 processos escritores no *HD* 30
Figura 4.12 Desvio padrão dos tempos de execução das instâncias de *one_at_time* com 2, 4 e 8 processos escritores no *HD* 31
Figura 4.13 Tempo médio de execução das versões *original* e instâncias de *all_at_once* com 2, 4 e 8 processos escritores no *SSD* 32
Figura 4.14 *Speedup* de *all_at_once* com 2, 4 e 8 processos escritores no *SSD* 32
Figura 4.15 Desvio padrão dos tempos de execução das instâncias de *all_at_once* com 2, 4 e 8 processos escritores no *SSD* 33
Figura 4.16 Tempo médio de execução das versões *original* e instâncias de *one_at_time* com 2, 4 e 8 processos escritores no *SSD* 34
Figura 4.17 *Speedup* de *one_at_time* com 2, 4 e 8 processos escritores no *SSD* 34
Figura 4.18 Desvio padrão dos tempos de execução das instâncias de *one_at_time* com 2, 4 e 8 processos escritores no *SSD* 35
Figura 4.19 Tempo médio de execução da versão *isend_recv* utilizando *OpenMPI* e *MPICH* no *HD* 36
Figura 4.20 *Speedup* da versão *isend_recv* utilizando *OpenMPI* e *MPICH* no *HD* 36
Figura 4.21 Desvio padrão dos tempos de execução das versões original e *isend_recv* utilizando *OpenMPI* e *MPICH* no *HD* 37
Figura 4.22 Tempo médio de execução da versão *isend_recv* utilizando *OpenMPI* e *MPICH* no *SSD* 38
Figura 4.23 *Speedup* da versão *isend_recv* utilizando *OpenMPI* e *MPICH* no *SSD* 38
Figura 4.24 Desvio padrão dos tempos de execução das versões original e *isend_recv* utilizando *OpenMPI* e *MPICH* no *SSD* 39

LISTA DE ABREVIATURAS E SIGLAS

E/S	Entrada/Saída
GPU	Graphics Processing Unit
HD	Hard Disk
I/O	Input/Output
MPI	Message Passing Interface
PCAD	Parque Computacional de Alto Desempenho
SSD	Solid State Drive

SUMÁRIO

1 INTRODUÇÃO 9
1.1 Objetivo 10
1.2 Organização do Texto 11
2 FUNDAMENTAÇÃO TEÓRICA 13
2.1 MPI 13
2.2 Método Fletcher 15
3 SOLUÇÕES PROPOSTAS 17
3.1 Versões MPI 17
4 ANÁLISE EXPERIMENTAL 21
4.1 Ambiente de execução 21
4.2 Testes 21
4.3 Funções Bloqueantes x Não-bloqueantes 22
4.3.1 Desempenho no HD 22
4.3.2 Desempenho no SSD 25
4.4 Criação dinâmica de processos e MPI I/O 27
4.4.1 Desempenho no HD 27
4.4.2 Desempenho no SSD 31
4.5 Comparação MPICH e OpenMPI em operações não-bloqueantes 35
4.5.1 Desempenho no HD 35
4.5.2 Desempenho no SSD 37
5 CONCLUSÃO 41
REFERÊNCIAS 43

1 INTRODUÇÃO

Na economia global, petróleo e gás são fontes de energia de extrema importância, representando aproximadamente 3% do PIB global (FORUM, 2022). No Brasil em 2020, o setor de petróleo e gás correspondia a 13% do PIB nacional(AGÊNCIA NACIONAL DO PETRÓLEO, 2022). Encontrar novos reservatórios desses materiais pode envolver técnicas prejudiciais ao meio ambiente, como escavações de poços em locais ambientalmente sensíveis, como áreas de conservação ambiental e oceanos. Essas atividades podem causar destruição de habitats, poluição das águas e outros impactos ambientais severos.

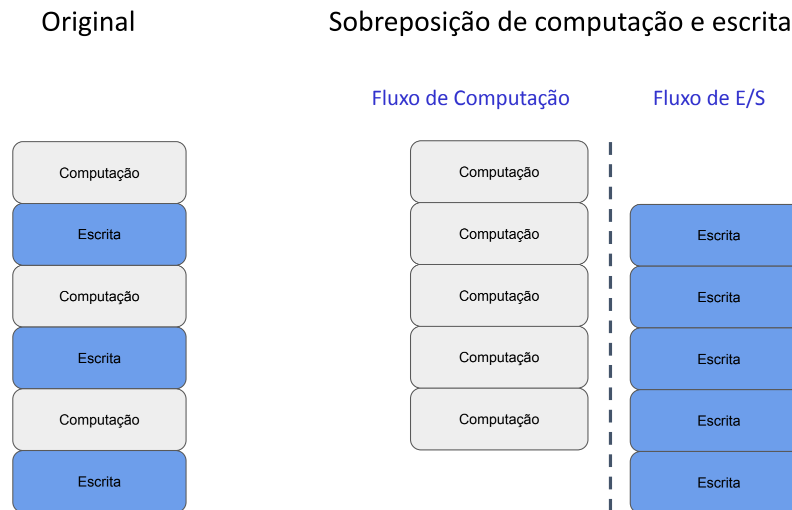
Nesse processo de busca, também são realizadas pesquisas sísmicas que buscam mapear possíveis reservatórios (AGÊNCIA NACIONAL DO PETRÓLEO, 2023). Para a realização dessas pesquisas, ondas sísmicas são lançadas a partir de equipamentos especiais na superfície de um corpo d'água. Essas ondas viajam em direção ao fundo, interagem com os materiais do solo sendo refletidas de volta à superfície, onde são coletadas. Com base nos dados coletados, são realizadas simulações que utilizam técnicas de modelagem matemática e física, considerando as propriedades geológicas e as características das ondas sísmicas para produzir imagens precisas do subsolo.

As pesquisas sísmicas podem mitigar os efeitos prejudiciais da exploração de novas fontes ao fornecer informações detalhadas sobre a localização e as características dos reservatórios. Com dados mais precisos, é possível minimizar a necessidade de perfurações exploratórias excessivas, reduzindo a destruição de habitats e a poluição ambiental. Isso resulta em uma abordagem mais sustentável e eficiente para a exploração de petróleo e gás.

A simulação de propagação de ondas sísmicas ocorre sobre grandes volumes de dados e sua computação é altamente paralelizável, podendo ser computada utilizando GPUs. O processo de simulação se dá em função do tempo, gerando um estado das ondas para cada instante de tempo computado. Alguns desses estados simulados devem ser salvos para análise posterior. Como resultado, obtemos volumes de dados que podem chegar à casa dos petabytes. No entanto, a necessidade de escrever os dados gerados em disco é um gargalo significativo para esse tipo de aplicação.

Portanto, a otimização da escrita dos dados gerados e disco é fundamental para melhorar o desempenho de aplicações de simulação sísmica. Nesse sentido, a sobreposição de computação e escrita em disco é um caminho que vem sendo utilizado para otimizar essas aplicações. Para esse fim, decidimos utilizar neste trabalho o MPI (*message-*

Figura 1.1: Fluxo original x sobreposto



Fonte: Autor

passing interface) para criar processos distintos para computação e I/O, para que estes rodem em paralelo, otimizando o tempo de computação necessário para realização das simulações.

1.1 Objetivo

Neste trabalho foi utilizado uma implementação do método *Fletcher* que permite realizar simulações sísmicas utilizando a computação em GPU. Atualmente as implementações do método *Fletcher* seguem um fluxo que alterna, sequencialmente, a computação e a escrita em disco. Dessa forma, a escrita em disco precisa ser concluída antes que as próximas rodadas de computação possam continuar. O objetivo deste trabalho é propor maneiras de sobrepor a computação e escrita dessa aplicação utilizando a norma *MPI*(FORUM, 2015), visando diminuir o tempo de execução da aplicação. A Figura 1.1 ilustra o fluxo sequencial de execução (atual) e a sobreposição dos fluxos de computação e escrita (proposta).

Para tal, foram implementadas várias versões do método *Fletcher* com *MPI* utilizando diferentes estratégias, como: comunicação entre processos, utilizando de trocas de mensagens síncronas e assíncronas. Criação estática e dinâmica de processos. Criação de um arquivo único e múltiplos arquivos para armazenar as ondas. Escrita paralela em um mesmo arquivo utilizando *MPI I/O*.

1.2 Organização do Texto

O texto deste trabalho está organizado da seguinte forma. No Capítulo 2 são abordados os conceitos importantes para o desenvolvimento do trabalho, incluindo a caracterização do método *Fletcher*, e a descrição do padrão *MPI*. No Capítulo 3 são descritas as soluções propostas para paralelizar a escrita e a computação do método *Fletcher* utilizando o padrão *MPI*. No Capítulo 4 são apresentadas a descrição do ambiente de testes, os parâmetros utilizados nos experimentos, e os dados coletados e a análise dos resultados obtidos. O Capítulo 6 apresenta a conclusão do trabalho, destacando as contribuições e os achados mais relevantes da pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

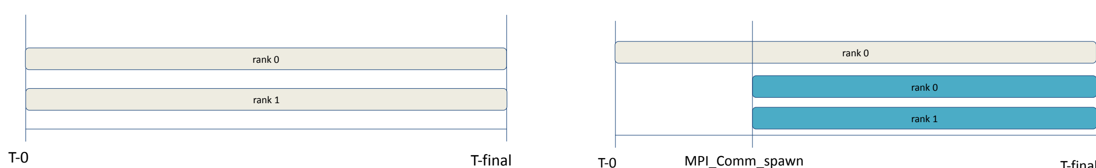
Neste capítulo, são abordados conceitos importantes para o desenvolvimento do trabalho, como: a explicação do funcionamento básico do padrão *MPI* e as suas funcionalidades utilizadas nas soluções apresentadas no Capítulo 3; uma introdução de alto nível ao método *Fletcher*.

2.1 MPI

MPI (*Message Passing Interface*) é um padrão para comunicação entre processos que possui diversas implementações, como *Open MPI* (OPENMPI, 2018) e *MPICH* (MPICH, 2019). O padrão *MPI* funciona de duas manerias básicas, uma delas é através da criação estática de processos onde a quantidade de processos é definida por um parâmetro passado ao *runtime* no momento de lançamento da aplicação (ex: *mpirun -np 2*). A Figura 2.1a ilustra a linha de tempo de execução de dois processos criados estaticamente. A outra forma é criando processos dinamicamente durante a execução da aplicação utilizando funções *MPI* como *MPI_Comm_spawn*. A Figura 2.1b ilustra a criação de dois novos processos (em azul) dinamicamente.

Processos criados no momento da inicialização executam utilizando o mesmo arquivo binário.Cada um dos processos recebe um identificador único chamado de *rank* e pertencem a um mesmo comunicador, neste caso, o intracomunicador global chamado de *MPI_COMM_WOLRD*. O controle de fluxo da execução de processos que compartilham um mesmo binário é feita através do número do *rank*. Assim, os processos podem executar diferentes partes do binário ou acessar diferentes porções de um dado.

Figura 2.1: Formas de criação de processos.



(a) Criação estática de processos (b) Criação dinâmica de processos

Fonte: Autor

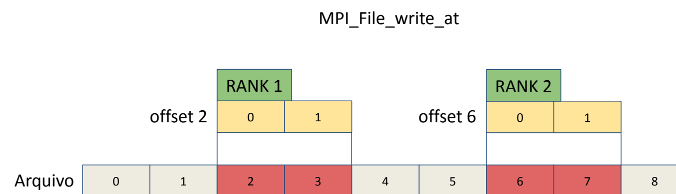
Comunicadores podem ser de dois tipos: intracomunicadores, onde acontecem as comunicações em um mesmo grupo de processos, e intercomunicadores, onde a comunicação entre dois grupos é realizada. A criação dinâmica de processos permite que um processo crie um ou mais processos novos especificando o arquivo binário que deve ser utilizado. Quando um ou mais processo são criados, eles passam a fazer parte de um novo intracomunicador. A *MPI_Comm_spawn* recebe como parâmetro um ponteiro para uma variável de comunicador. Essa variável comunicador passa a conter um intercomunicador que liga o intracomunicador ao qual o processo pai pertence e o novo intracomunicador dos processos criados.

Existem dois grupos de funções *MPI*, as funções bloqueantes e a não-bloqueantes. Funções bloqueantes executam até o seu fim antes de continuar a execução do programa. As funções não-bloqueantes liberam imediatamente a continuação do programa e a sua execução é gerenciada pelo runtime *MPI*. Na prática, diferentes implementações do *MPI* podem ter comportamentos diferentes, por exemplo, a implementação *OpenMPI* não tem um processo dedicado a dar continuidade as funções não-bloqueantes em *background*, já a implementação *MPICH* mantém um processo para isso (DENIS et al., 2022). O progresso das funções não-bloqueantes no *OpenMPI* acontecem nos momentos em que funções *MPI* são chamadas. A função *MPI_Wait* pode ser utilizada no caso de ser necessário garantir que uma função não-bloqueante seja totalmente executada antes de prosseguir com o programa. A troca de mensagens entre os diferentes processos pode ser feita com as funções bloqueantes de envio *MPI_Send* e de recebimento *MPI_Recv* e suas versões não-bloqueantes *MPI_Isend* e *MPI_Irecv*.

O *MPI* também define formas de acessar e lidar com arquivos em disco utilizando o que é chamado de *MPI I/O*. Utilizando *MPI I/O* é possível ler e escrever paralelamente em um mesmo arquivo. Isso permite que diversos processos pertencentes a um mesmo intracomunicador tenham acesso ao mesmo arquivo controladamente. Para isso a abertura do arquivo deve ser feita utilizando a função *MPI_File_open* que tem como um de seus argumentos o intracomunicador do grupo de processos que podem acessar o arquivo. Quando um processo for escrever em um arquivo ele pode utilizar a função *MPI_File_write_at* que recebe um *offset* que permite que cada processo escreva em um determinado local do arquivo. Da mesma forma, a leitura pode ser feita utilizando *MPI_File_read_at* passando o *offset*. A Figura 2.2 ilustra a escrita sendo realizada por dois processos, *RANK 1* e *RANK 2*, paralelamente, em um mesmo arquivo. O *MPI* traz otimizações para escrita para paralela, como: organizar diversos pedidos de escrita a fim

de agrupar os dados contíguos e ordenar as escritas pelo local onde estão salvas para melhorar o desempenho do sistema de arquivos, e eliminando requisições duplicadas por diferentes processos.

Figura 2.2: MPI-IO: Escrita paralela com *MPI_File_write_at*



Fonte: Autor

2.2 Método Fletcher

O método *Fletcher* é um algoritmo que simula a propagação de ondas sísmicas em um meio em função do tempo. A simulação é realizada sobre uma *grid* tridimensional, que representa um local do mundo real. Está grid é representada por uma estrutura de dados de 3 dimensões de pontos flutuantes. Para cada ponto desta *grid*, são calculadas equações diferenciais parciais relacionadas à propagação de ondas sísmicas (FLETCHER; DU; FOWLER, 2009).

O Algoritmo 1 representa o funcionamento do método *Fletcher*. Na inicialização, o algoritmo recebe como parâmetros as dimensões da *grid 3D* (*x, y* e *z*), o tempo que passa a cada computação (*passo*) e o intervalo de tempo onde a simulação é realizada (*tempoTotal*). A *grid* é inicializada (*inicializaGrid*) e o seu estado inicial é armazenada (*gravaEmDisco*). A grid é enviada para GPU usando *enviaParaGPU*. O algoritmo entra no *loop for*, onde para cada instante de tempo da simulação será calculada a propagação da onda (*propagacaoOnda*) através da resolução das EDPs na *GPU*. Quando o tempo de simulação transcorrido atinge um valor limite (*limiteParaEscrita*, a grid que está na GPU é copiada de volta para memória principal usando *copiaParaMemoriaPrincipal*. Então o estado atual da *grid* é salvo em disco e o limiteParaEscrita é atualizado multiplicando o valor de *limiteInicial* (0,01) pela quantidade de escritas realizadas.

Algoritmo 1 Pseudocódigo Fletcher da versão Original

```
1: function FLETCHER(x, y, z, passo, tempoTotal)
2:     grid ← inicializaGrid(x, y, z)
3:     gravaEmDisco(grid)
4:     escritasEmDisco ← 1
5:     enviaParaGPU(grid)                                  ▷ grid CPU → GPU
6:     for it ← 1, it ≤ ceil(tempoTotal/passo), it + + do
7:         propagacaoOndaGPU(grid)
8:         if it ∗ passo ≥ limiteParaEscrita then
9:             copiaParaMemoriaPrincipal(grid)             ▷ grid GPU → CPU
10:            gravaEmDisco(grid)
11:            escritasEmDisco + +
12:            limiteParaEscrita ← limiteInicial ∗ escritasEmDisco
13:        end if
14:    end for
15: end function
```

Com isso, no final da execução do algoritmo, temos vários momentos da simulação armazenados. O que permite que sejam realizadas análises posteriores.

3 SOLUÇÕES PROPOSTAS

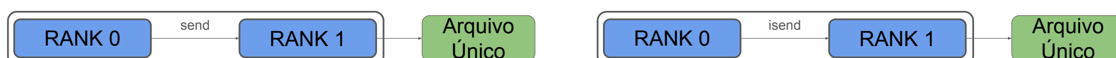
Neste capítulo são descritas as soluções propostas para a paralelização do método *Fletcher* com o uso de *MPI*. Para cada versão é descrita seu funcionamento, apresentando os tipos de funções utilizadas para a comunicação entre processos (bloqueante ou não-bloqueante), a maneira que os processos são criados (dinâmica ou estática), como os dados a serem escritos são divididos entre os processos escritores e como a escrita é realizada.

3.1 Versões MPI

Para este trabalho foram desenvolvidas quatro versões do método *Fletcher* utilizando a biblioteca *Open MPI* para sobrepor a computação da propagação de ondas e a escrita em disco. Em todas as versões o processo de *rank* 0 é responsável por realizar a computação e, sempre que a condição de escrita for atingida, esse processo enviará o estado atual da *grid* para um processo que deverá realizar a escrita em disco. Cada uma das versões tenta explorar uma estratégia diferente do processo de paralelização, variando a quantidade de processos criados, como esses processos são criados (estática ou dinâmica), a forma da comunicação entre processos (bloqueante ou não-bloqueante) e a maneira como a escrita é realizada em disco (único arquivo ou múltiplos arquivos).

- *send_recv*: nessa versão foi utilizada a criação estática de 2 processos *MPI*. A comunicação entre os dois processos é feita utilizando as funções bloqueantes *MPI_Send* e *MPI_Recv*. O processo de *rank* 1 executa em um laço, onde este chama a função *MPI_Recv* e aguarda o processo de *rank* 0 enviar a *grid*. Após receber a *grid* esse

Figura 3.1: Versões bloqueante e não-bloqueante.



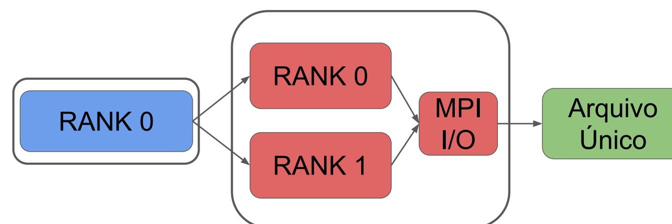
(a) *send_recv*: comunicação bloqueante (b) *isend_recv*: comunicação não-bloqueante

Fonte: Autor

processo realiza a sua escrita em disco. Este processo é repetido até que todos os estados sejam salvos em disco. A Figura 3.1a representa a implementação da versão descrita.

- *isend_recv*: de forma similar a versão *send_recv*, a criação de processos de forma é estática. Porém, aqui o processo de *rank* 0 utiliza a chamada não-bloqueante *MPI_Isend* para realizar o envio da *grid*. A ideia desta versão é que o envio da *grid* seja realizado em *background* enquanto a próxima rodada de computação está sendo realizada. A Figura 3.1b representa a implementação da versão descrita.

Figura 3.2: *all_at_once*: criação dinâmica de processos com escrita paralela



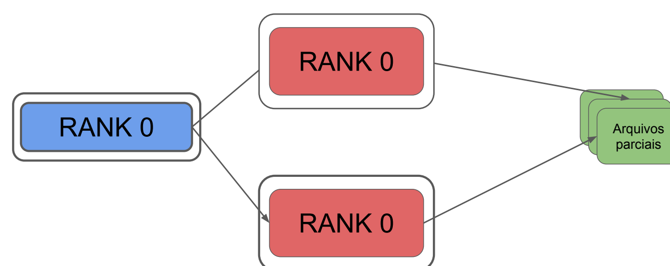
Fonte: Autor

- *all_at_once*: está versão inicializa somente um processo que recebe um argumento adicional que define o número de processos que deverão ser criados dinamicamente. No momento em que o critério de escrita é atingido, o processo de *rank* 0 chama a função *MPI_Comm_Spawn* passando por parâmetro o número de processos a serem criados e indicando um binário que deverá ser utilizado pelos processos. Os processos criados por essa chamada farão parte de um intracomunicador novo, no qual terão seus *ranks* iniciando em 0. O *rank 0* inicial passa a fazer o envio da *grid* para os processos filhos de forma circular partindo do processo de *rank* 0. Todos os processos criados chamam a função *MPI_File_open* indicando o nome do arquivo onde farão as escritas em disco. O arquivo passa a poder ser acessado paralelamente por todos esses processos. Cada processo recebe suas *grids* e escreve no arquivo utilizando a função *MPI_File_write_at* indicando o arquivo aberto anteriormente e o deslocamento a partir do início do arquivo até o local onde deverá ser feita a escrita. A Figura 3.2 representa a implementação da versão descrita, onde dois processos foram criados dinamicamente (em vermelho) para se encarregarem

de escrever em disco utilizando *MPI-IO*. Os novos processos são parte de um novo grupo de comunicação, por isso os *ranks* começam em 0.

- *one_at_time*: está versão também só inicializa um processo e recebe como argumento a quantidade de processos que devem ser criados. Cada vez que o processo de *rank* 0 tiver que enviar a *grid* ele irá criar um processo para realizar a escrita e em seguida enviará a *grid* para esse processo. Ao criar todos os processos definidos pelo argumento recebido na inicialização, o envio passará a ser feito de forma circular entre os processos. Como cada processo é criado separadamente, eles não fazem parte do mesmo grupo e, portanto, não podem utilizar *MPI I/O* para escrever paralelamente em um mesmo arquivo. Nessa versão cada processo cria um arquivo parcial contendo um único estado da *grid*. A Figura 3.3 representa a implementação da versão descrita, onde dois novos processos (em vermelho) realizam a escrita dos estados da *grid* em arquivos separados.

Figura 3.3: *one_at_time*: criação dinâmica de processos com múltiplos arquivos



Fonte: Autor

4 ANÁLISE EXPERIMENTAL

Neste capítulo é descrito o ambiente de onde foram realizados os testes, os parâmetros utilizados nos experimentos, e são apresentados e analisados os dados obtidos das execuções das soluções propostas e da versão original do método *Fletcher*. Cada seção de resultados é dividida entre testes realizados em *HD* e *SSD*. A Seção 4.3 contém as análises das versões *send_recv* e *isend_recv* a fim de comparar o desempenho das funções bloqueantes e não-bloqueantes. A Seção 4.4 contém as análises dos dados das versões *all_at_once* e *one_at_time* que utilizam a criação dinâmica de processos, sendo que a primeira utiliza a funcionalidade *MPI I/O* para escrita. Na Seção 4.5 é feita a comparação dos resultados de *isend_recv* executada com *OpenMPI* e com *MPICH*.

4.1 Ambiente de execução

Os testes foram realizados no Parque Computacional de Alto Desempenho (PCAD) utilizando o nó *Blaise*, que possui as seguintes configurações de hardware: dois processadores *Intel Xeon E5-2699 v4 Broadwell (Q1'16)*, 2,2 GHz, uma *GPU P100* de 3584 CUDA cores, 256 GB 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. O nó executa o Sistema Operacional *Debian* 10, o sistema de arquivos utilizado em ambas unidades é *ext4*. As versões do *MPI* utilizadas foram *OpenMPI v3.1* e *MPICH v3.3*.

4.2 Testes

Todas as versões implementadas e para versão original, descrita na Seção 2.2, foram executadas com os seguintes parâmetros:

- Tamanhos de *grid* partindo de 120 e incrementados em 32 até chegar em 504 (todas as dimensões têm o mesmo tamanho).
- Tempo total fixo em 1,5.
- Passo de tempo fixo em 0,001.
- Variação da escrita entre *SSD* e *HD*.

As versões *all_at_once* e *one_at_time* possuem um parâmetro adicional que define a quantidade de processos a serem criados para realizar a escrita. Os valores utilizados nos testes foram de 2, 4 e 8 processos.

Cada configuração foi executada 10 vezes utilizando a biblioteca *OpenMPI*. A versão *isend_recv* foi testada também com a biblioteca *MPICH* para comparar o impacto das diferentes implementações do *runtime* no que diz respeito ao comportamento das funções não-bloqueantes.

Para todos os testes foram coletados o tempo total de execução, foram calculados o tempo médio e desvio padrão para cada teste. Os valores obtidos foram utilizados para calcular o *speedup* das soluções propostas em relação a versão original (tempo médio da versão original dividido pelo tempo médio da versão *MPI*), para cada variação de parâmetros. O *speedup* é uma métrica utilizada para comparar o desempenho de uma tarefa executada com versões diferentes de uma aplicação ou recursos diferentes.

4.3 Funções Bloqueantes x Não-bloqueantes

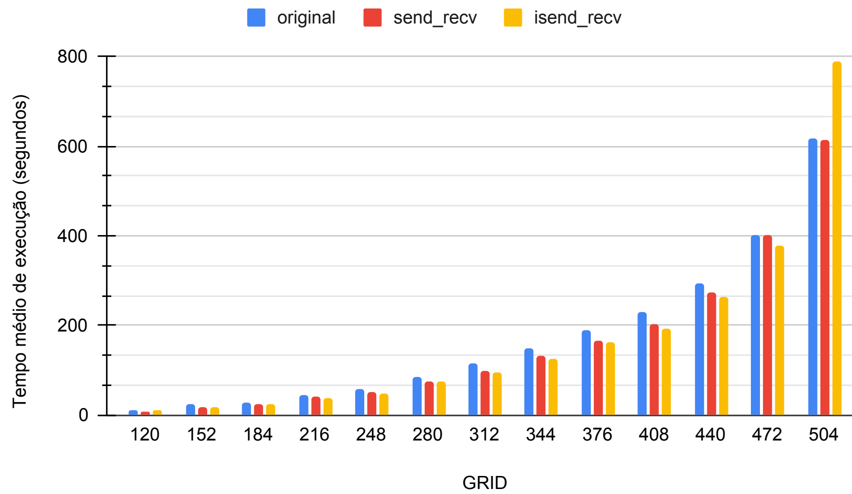
Nesta seção serão mostrados e analisados os dados referentes as execuções versões *send_recv* e *isend_recv* em comparação à versão original.

4.3.1 Desempenho no HD

O tempo das versões *MPI* é inferior ao original para todos os tamanhos, exceto a *grid* 504 para a versão *isend_recv*. A Figura 4.1 apresenta o gráfico do tempo médio de execução das funções *isend_recv*, *send_recv* e *original*, quanto menor a barra, melhor o desempenho. O maior *speedup* atingido pela versão *send_recv* foi de 1,30x para*grid* e tamanho 120 e a versão *isend_recv* teve seu maior *speedup* para*grid* de tamanho 408, com 1,19x. Para as *grid* maiores, a partir de 440, o desempenho de ambas as versões começam a diminuir, atingindo seu pior desempenho na maior *grid*. A versão *send_recv*, para *grid* 504, tem desempenho idêntico ao da versão original, enquanto a versão *isend_recv* tem um *speedup* de 0,78x. O *speedup* médio das versões *send_recv* e *isend_recv* foram de 1,12x. A Figura 4.2 mostra os *speedups* das versões *isend_recv* e *send_recv*, quanto maior a barra, melhor.

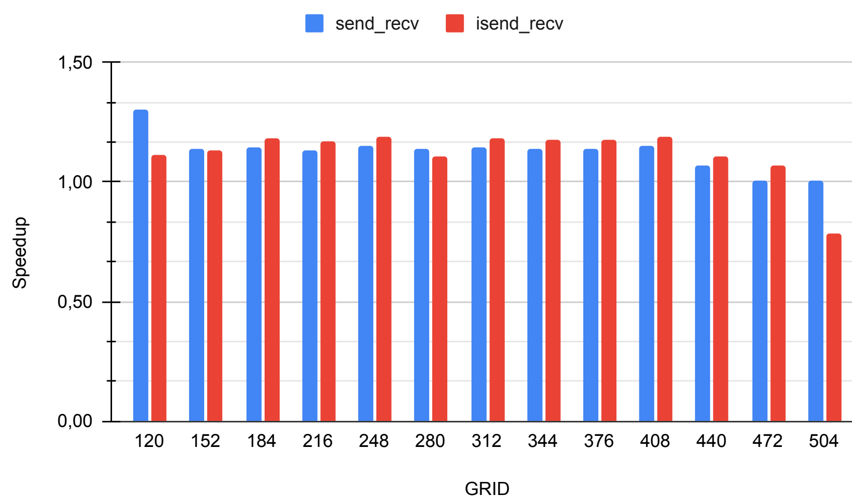
À medida que o tamanho das *grids* cresce, maior é a variabilidade dos tempos de

Figura 4.1: Tempo médio de execução das versões *original*, *send_recv* e *isend_recv* no *HD*



Fonte: Autor

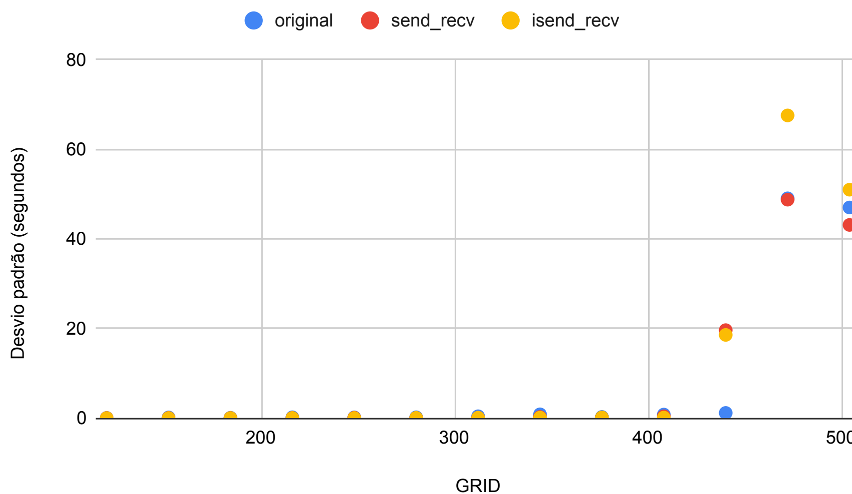
Figura 4.2: Speedup das versões *send_recv* e *isend_recv* no *HD*



Fonte: Autor

escrita, como pode ser visto na Figura 4.3, que mostra o desvio padrão dos tempos de escrita das versões avaliadas. O sistema operacional e disco rígido oferecem *buffers* que permitem esconder o tempo real de escrita. Ao que tudo indica, para os tamanhos de *grid* superiores a 440, a aplicação gera volumes de dados suficientes para saturar esses *buffers*. Esse esgotamento ocorre mais rapidamente nas versões *MPI*, pois elas diminuem o tempo entre cada escrita. Assim, o tempo de escrita passa a ser evidente desde mais cedo na execução dessas versões. Na versão *send_recv*, para que o envio da onda seja realizado é necessário que a onda anterior tenha sido salva, por ser somente nesse momento que o processo que realiza escrita volta a receber mensagens. Quando os *buffers* são esgotados, a escrita da onda anterior pode não ter sido concluída no momento em que o envio seria realizado. Isso somado ao overhead do runtime MPI pode ser a causa da piora do desempenho. Já na versão *isend_recv*, por mais que o envio seja não bloqueante, existe uma pré-condição no código que exige que o envio da onda anterior tenha sido completada antes de chamar *MPI_Isend*. Assim, o mesmo problema da versão bloqueante pode ocorrer aqui, mas desta vez o overhead do gerenciamento de uma operação não-bloqueante parece ser maior do que para operações bloqueantes.

Figura 4.3: Desvio padrão das versões *original*, *send_recv* e *isend_recv* no *HD*

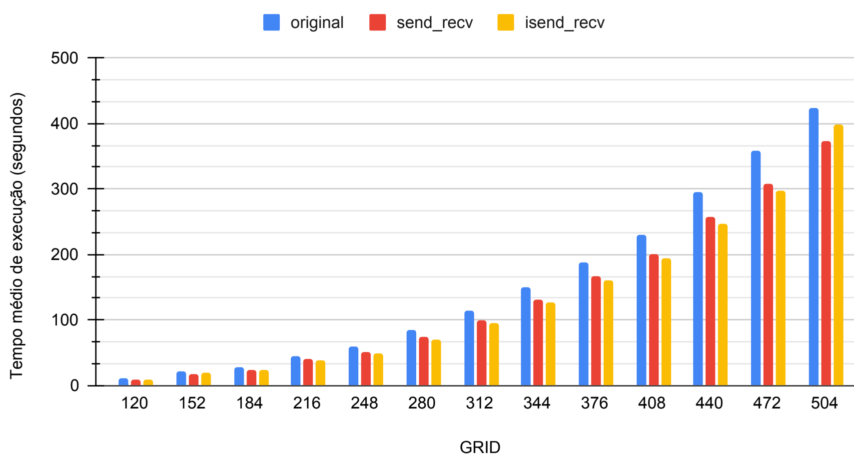


Fonte: Autor

4.3.2 Desempenho no SSD

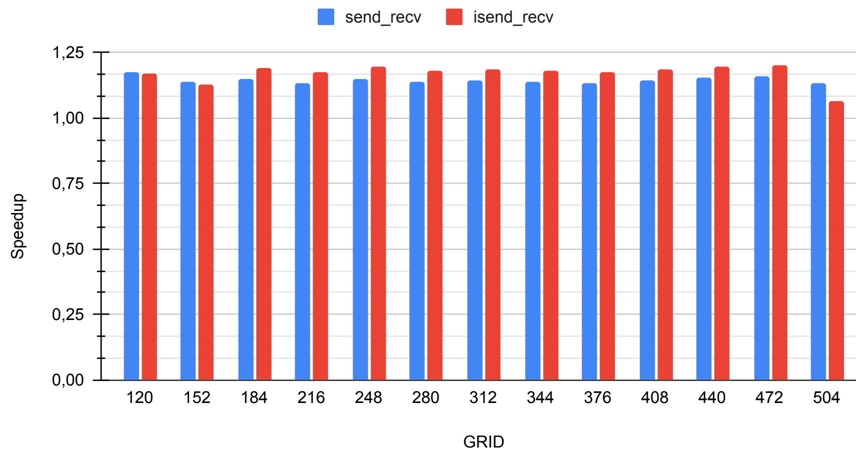
Aqui as versões *send_recv* e *isend_recv* tiveram tempos menores que a versão original para todos os tamanhos de *grid*. O tempo médio de execução das versões original, *send_recv* e *isend_recv* podem ser vistos na Figura 4.4. O melhor desempenho da versão *send_recv* foi para a *grid* de 120 obtendo um *speedup* de 1,17x, a *isend_recv* obteve seu melhor resultado para a *grid* 472 com 1,20x de *speedup*. Diferente das execuções em *HD*, aqui o tamanho da *grid* não é grande o suficiente para sobrecarregar o *SSD*. Na versão *send_recv*, todas as *grids* tiveram desvio padrão inferiora 0,3 segundos, com exceção do tamanho 504 que chegou a 5 segundos. O desvio padrão para todos os tamanhos em *isend_recv* foi inferior a 0,2 segundos. A Figura 4.5 mostra o desvio padrão para as três versões para cada tamanho de *grid*. O ganho de desempenho das versões *send_recv* e *isend_recv* são bastante próximos, tendo um *speedup* médio de 1,14x e 1,17x respectivamente.

Figura 4.4: Tempo médio de execução das versões *original*, *send_recv* e *isend_recv* no *SSD*



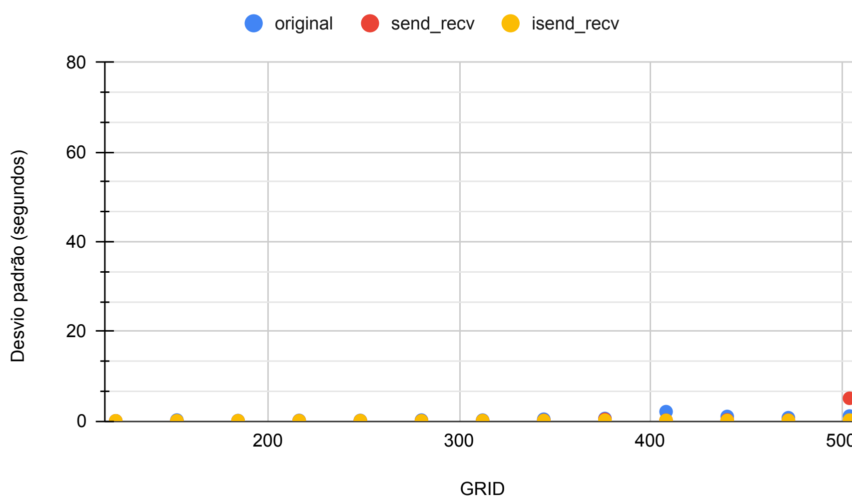
Fonte: Autor

Figura 4.5: Speedup das versões *send_recv* e *isend_recv* no *SSD*



Fonte: Autor

Figura 4.6: Desvio padrão das versões *original*, *send_recv* e *isend_recv* no *SSD*



Fonte: Autor

4.4 Criação dinâmica de processos e MPI I/O

Nesta seção serão mostrados os dados e análises referentes as execuções das versões que utilizam criação dinâmica de processos introduzida na versão 2 do MPI. Nas versões *all_at_once* e *one_at_time* os testes foram realizados utilizados os valores 2, 4 e 8 como quantidade de processos. A versão *all_at_once* utiliza a funcionalidade *MPI I/O* para escrever paralelamente em um único arquivo, enquanto *one_at_time* gera múltiplos arquivos.

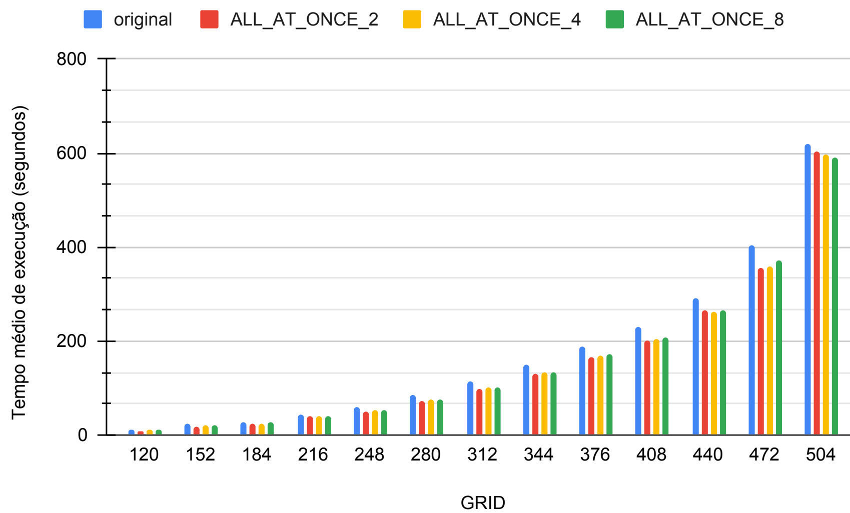
4.4.1 Desempenho no HD

Para as execuções com tamanho de *grid* de até 472, as instâncias com menos processos obtiveram melhores resultados. Na Figura 4.7 são mostrados os tempos médios de execução da versão original e das instâncias de *all_at_once*. Para *grid* de 504 essa questão se inverte, porém, todas as instâncias obtiveram *speedups* inferiores a 1,05x. O *speedup* médio da instância com 2 processos escritores foi de 1,11x, com 4 processos 1,09x e para 8 processos 1,08x. O maior *speedup* foi da instância com 2 processos para o tamanho 408, chegando a 1,14x. A Figura 4.8 mostra o *speedup* das instâncias de *all_at_once*.

As execuções para as *grids* de maior tamanho sofrem do mesmo problema relatado para execuções em *HD* relatadas na seção anterior, como pode ser visto na Figura 4.9 o desvio padrão dos tempos de execução aumenta consideravelmente para os tamanhos superiores a 440. Porém, aqui temos tempos médios menores para todas as instâncias da versão *all_at_once*. Duas coisas explicam isso, uma é que é possível ver a quantidade de processos escritores como *buffers*. Quanto maior o *buffer*, melhor a capacidade de esconder o tempo de escrita. A outra coisa é que essa versão utiliza *MPI I/O* com escritas paralelas em um mesmo arquivo. Essa funcionalidade do *MPI* é otimizada para escrita em paralelo, podendo ser responsável pela melhora do desempenho em relação às versões original e *send_recv* e *isend_recv* anteriormente apresentadas.

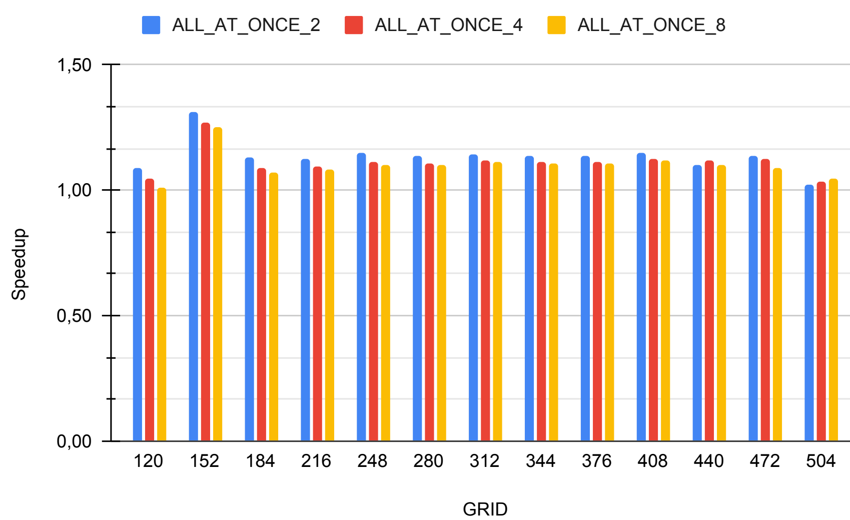
A Figura 4.10 mostra os tempos da versão *one_at_time* para as instâncias com 2, 4 e 8 processos escritores e a versão original para comparação. Na versão *one_at_time*, para o tamanho de 120, somente a instância com 2 processos obteve um melhor desempenho que a versão original. A instância com 2 processo teve um *speedup* de 1,04x, a de 4 processos 0,94x e a de 8 processos 0,79x. Como os processos são criados em momentos

Figura 4.7: Tempo médio de execução das versões *original* e instâncias de *all_at_once* com 2, 4 e 8 processos escritores no *HD*



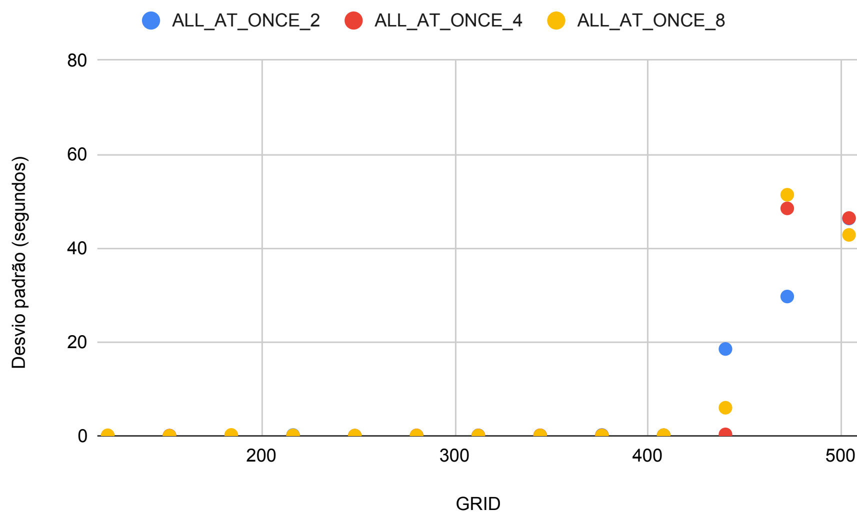
Fonte: Autor

Figura 4.8: *Speedup* da *all_at_once* com 2, 4 e 8 processos escritores no *HD*



Fonte: Autor

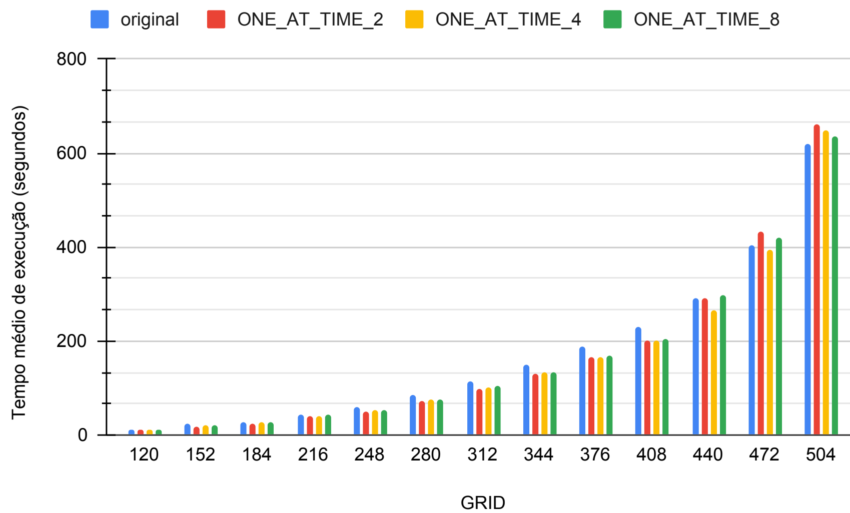
Figura 4.9: Desvio padrão dos tempos de execução das instâncias de *all_at_once* com 2, 4 e 8 processos escritores no *HD*



Fonte: Autor

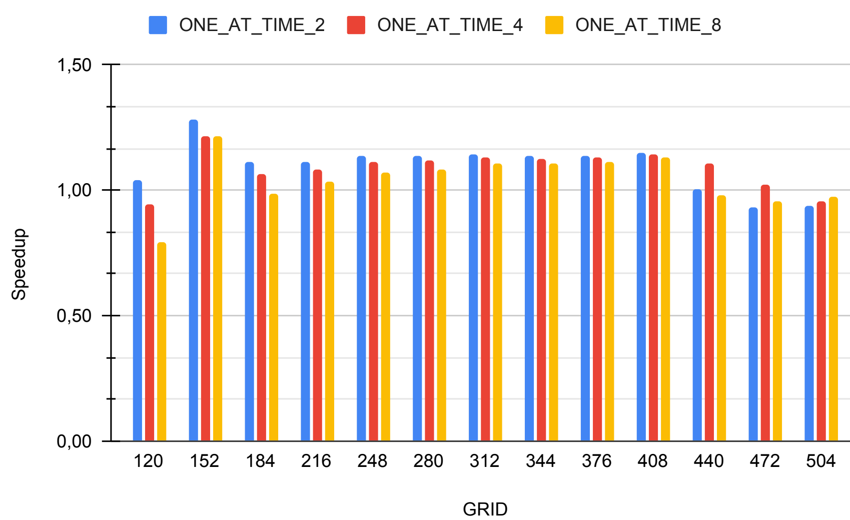
diferentes, cada um deles faz parte de um novo grupo. O *overhead* da criação de um processo e toda a estrutura que o *runtime* precisa criar para gerenciar diferentes comunicadores parece ter impacto no desempenho das instâncias com mais processos. Para os demais tamanhos de *grid* até 408 todas as instâncias possuem desempenho superior à versão original. O *speedup* médio das instâncias de 2, 4 e 8 processos foi de 1,08x, 1,07x e 1,02x, respectivamente. Da mesma forma que relatado para as outras versões executadas em *HD*, as instâncias passam a ter pior desempenho por volta do tamanho 440 em diante. Com o esgotamento dos *buffers* e o *HD* operando no limite, os tempos de execução de todas as versões variam muito, como pode ser visto na Figura 4.11 que mostra o desvio padrão das execuções.

Figura 4.10: Tempo médio de execução das versões *original* e instâncias de *one_at_time* com 2, 4 e 8 processos escritores no *HD*



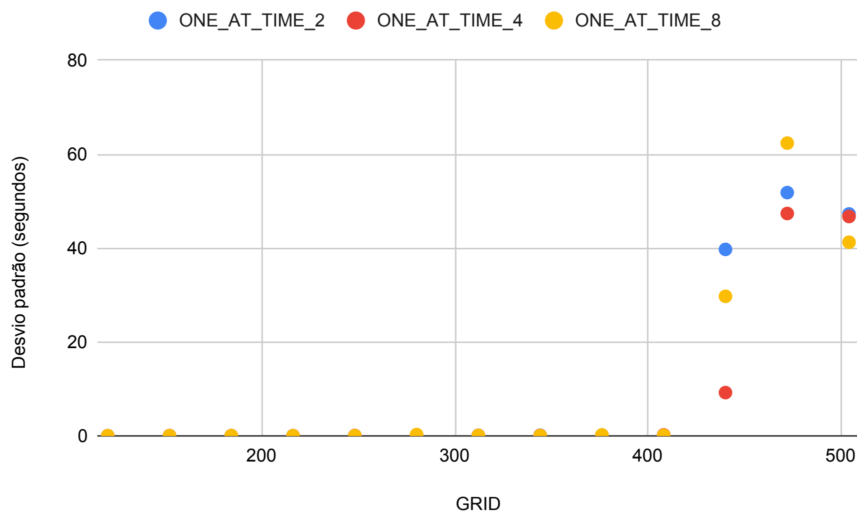
Fonte: Autor

Figura 4.11: *Speedup* de *one_at_time* com 2, 4 e 8 processos escritores no *HD*



Fonte: Autor

Figura 4.12: Desvio padrão dos tempos de execução das instâncias de *one_at_time* com 2, 4 e 8 processos escritores no *HD*



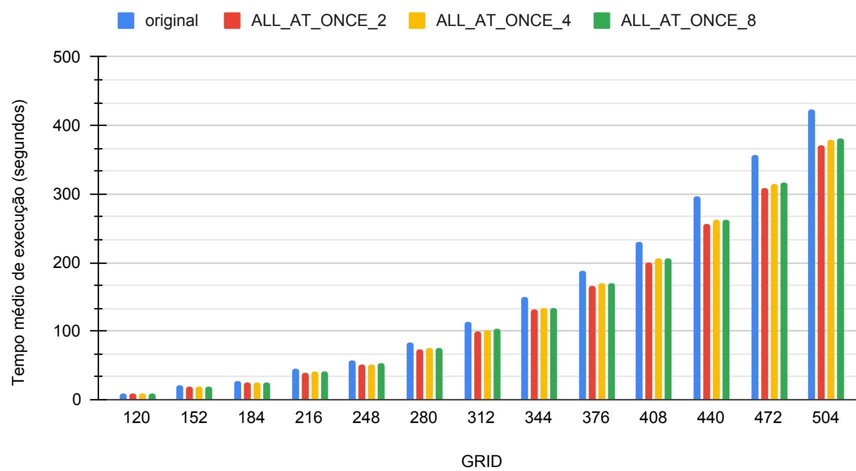
Fonte: Autor

4.4.2 Desempenho no SSD

Nas execuções em *SSD*, para todos os tamanhos de *grid*, todas as instâncias de *all_at_once* obtiveram tempos melhores que a versão original, como pode ser visto na Figura 4.13 que mostra o tempo médio das execuções. O *speedup* médio para a instância com 2 processos foi de 1,13x, para de 4 processos foi de 1,10x e para de 8 processos 1,09x. Para três instâncias o melhor desempenho ocorreu para a *grid* de tamanho 472, onde a instância de 2 processos obteve um *speedup* de 1,16x e as com 4 e 8 processos 1,13x. A 4.14 mostra os *speedups* das instâncias de *all_at_once* O ganho de desempenho diminui com o aumento da quantidade de processos. A razão para isso pode ser o *overhead* relacionado ao gerenciamento de processos pelo *runtime*. O desvio padrão para todas as instâncias e tamanho não passa de 0,4 segundos, como pode ser visto na Figura 4.15.

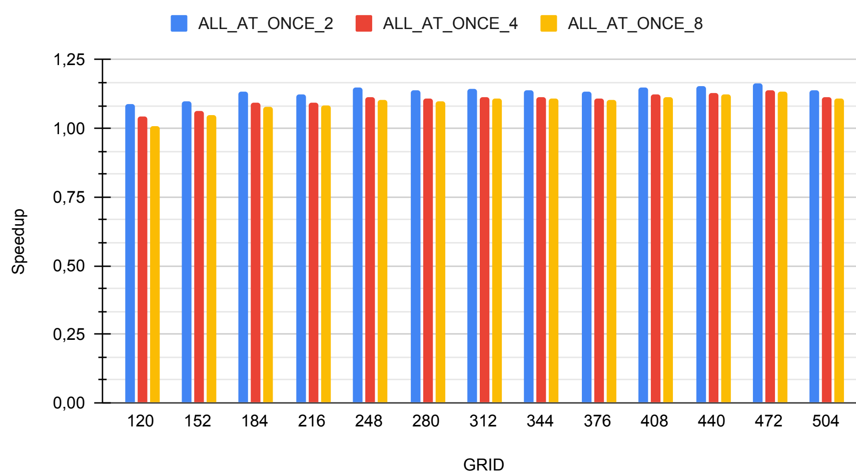
As instâncias da versão *one_at_time* tiveram tempos de execução menores que a versão original. A Figura 4.16 mostra o tempo médio da versão original e das três instancias da versão *one_at_time* com 2, 4 e 8 processos escritores. O *speedup* médio das instâncias foi de 1,12x para 2 processos, 1,10x para 4 processos e 1,06 para 8 processos. As três instancias tiveram seus melhores desempenhos para a *grid* de tamanho 472, onde a instância com 2 processos teve *speedup* de 1,16x e as com 4 e 8 processos tiveram 1,15x.

Figura 4.13: Tempo médio de execução das versões *original* e instâncias de *all_at_once* com 2, 4 e 8 processos escritores no *SSD*



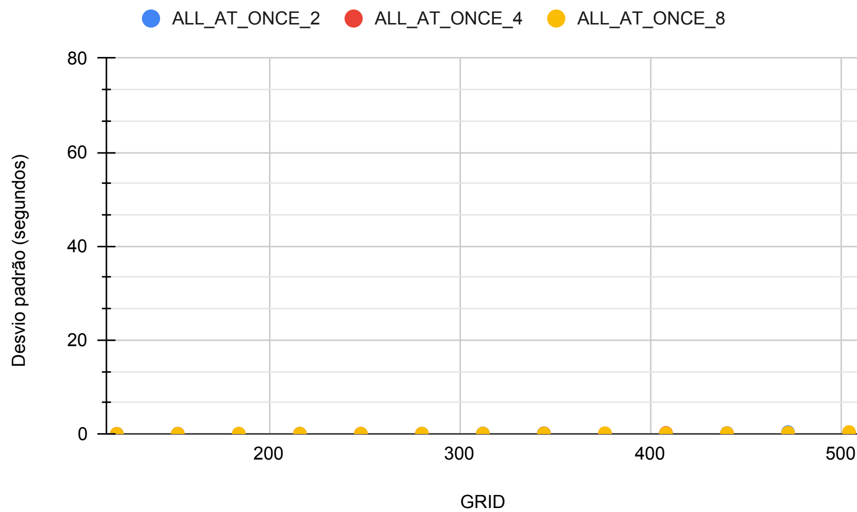
Fonte: Autor

Figura 4.14: *Speedup* de *all_at_once* com 2, 4 e 8 processos escritores no *SSD*



Fonte: Autor

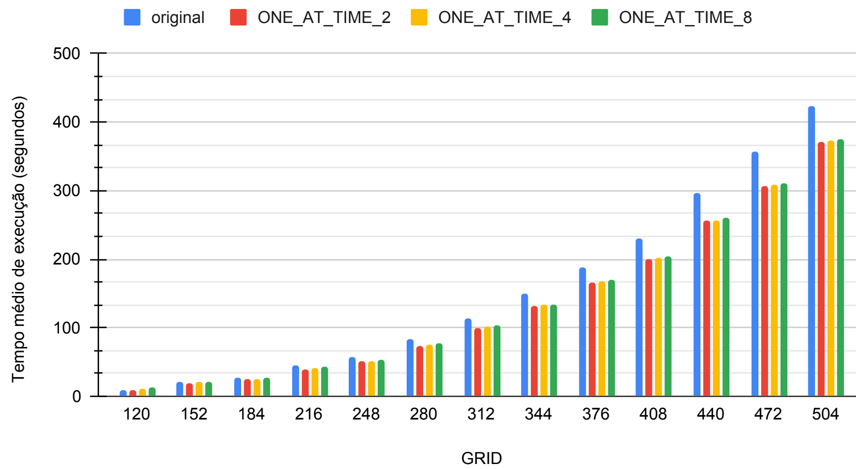
Figura 4.15: Desvio padrão dos tempos de execução das instâncias de *all_at_once* com 2, 4 e 8 processos escritores no *SSD*



Fonte: Autor

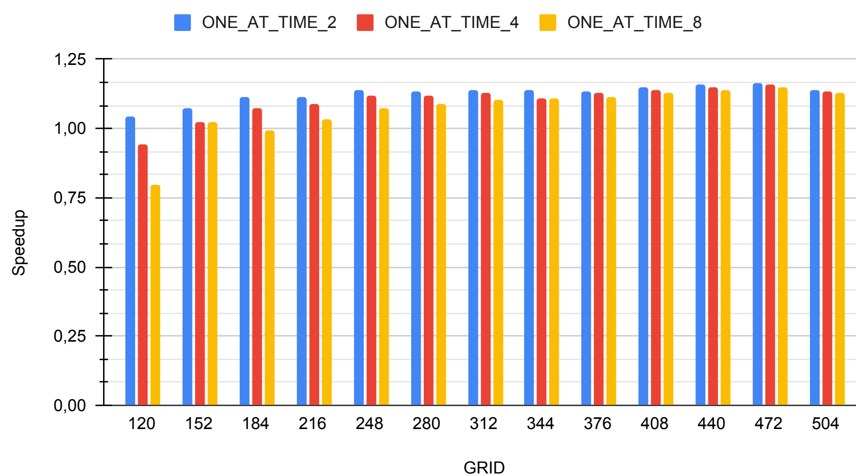
A Figura 4.17 mostra os *speedup* das instâncias para todos os tamanhos de *grid*. O ganho de desempenho se mantém estável a medida que os tamanhos de *grid* aumentam. Para as instâncias com mais processos o desempenho é inferior, devido ao *overhead* de gerenciamento dos processos. A Figura 4.18 mostra o desvio padrão para todas as instâncias e tamanhos de *grid*.

Figura 4.16: Tempo médio de execução das versões *original* e instâncias de *one_at_time* com 2, 4 e 8 processos escritores no *SSD*



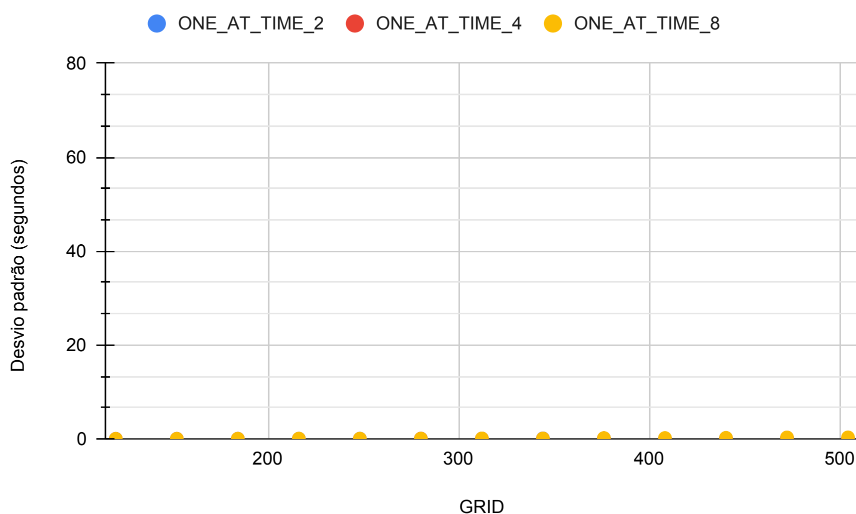
Fonte: Autor

Figura 4.17: *Speedup* de *one_at_time* com 2, 4 e 8 processos escritores no *SSD*



Fonte: Autor

Figura 4.18: Desvio padrão dos tempos de execução das instâncias de *one_at_time* com 2, 4 e 8 processos escritores no *SSD*



Fonte: Autor

4.5 Comparação MPICH e OpenMPI em operações não-bloqueantes

Nessa seção serão comparados os tempos da versão *isend_recv* rodando utilizando as implementações *MPICH* e *OpenMPI* do padrão *MPI*.

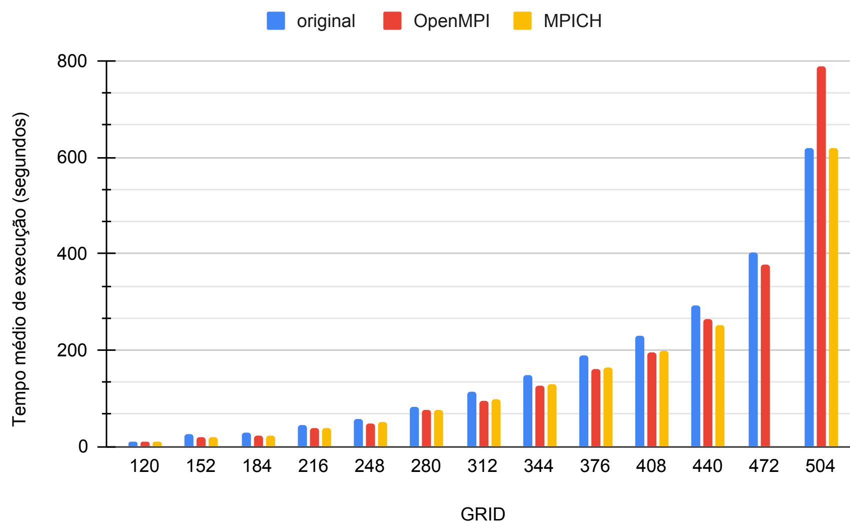
4.5.1 Desempenho no HD

As instâncias de *isend_recv* executadas com *MPICH* e *OpenMPI* tem melhor desempenho que a versão original para todos os tamanhos *grid*, com exceção do maior. A Figura 4.19 mostra o tempo médio de execução dessas instancias. Ambas as instâncias têm desempenhos similares, *speedup* médio delas é de 1,12x. O tamanho de *grid* onde as instâncias obtiveram o melhor *speedup* é a com a *grid* de tamanho 248, chegando a 1,19x. O *speedup* das instâncias é mostrado na Figura 4.20.

A *thread* de progresso da implementação *MPICH* não traz ganho de desempenho quando comparada com a implementação *OpenMPI*. O maior tamanho de *grid* mostra um melhor desempenho na instância que foi executada com *MPICH*. Essa diferença pode ser devida a algum fator interno do sistema, já que os testes foram realizados em momentos diferentes. Estados diferentes de distribuição dos dados no disco podem afetar o resultado quando levamos o disco ao limite, já que os mecanismos que tentariam disfarçar o tempo

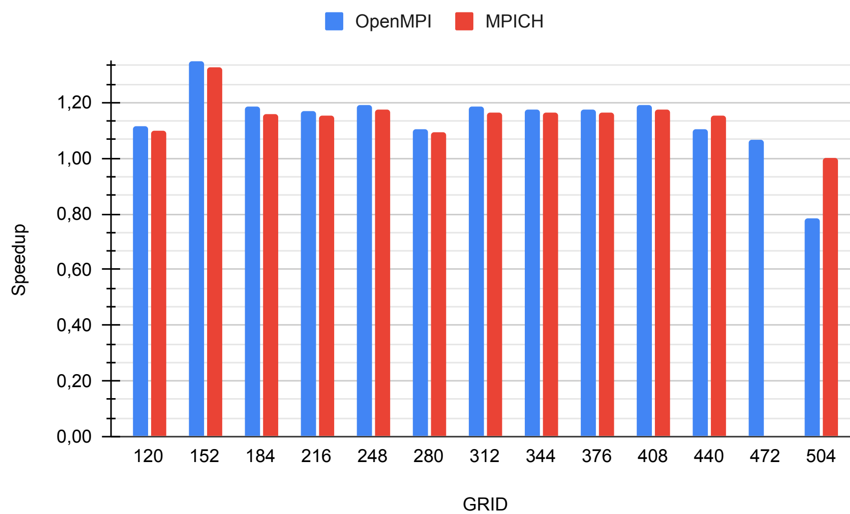
real de escrita foram saturados. O desvio padrão para as execuções pode é mostrado na Figura 4.21.

Figura 4.19: Tempo médio de execução da versão *isend_recv* utilizando *OpenMPI* e *MPICH* no *HD*



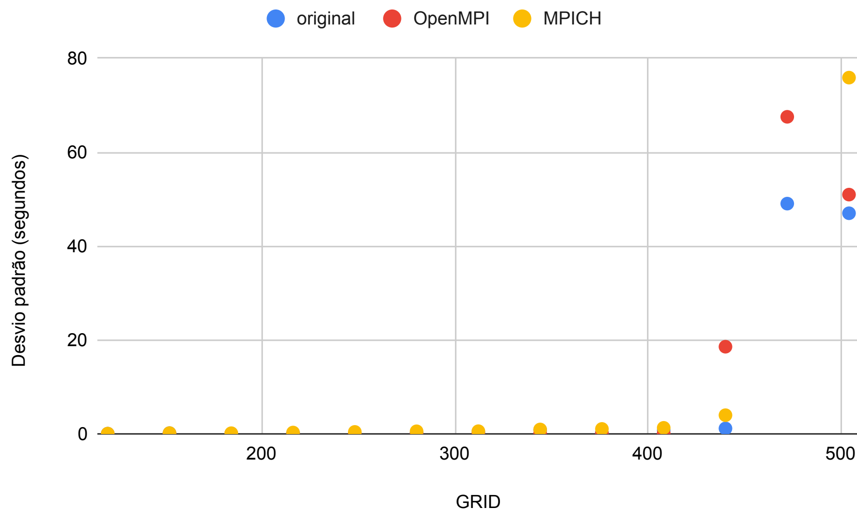
Fonte: Autor

Figura 4.20: *Speedup* da versão *isend_recv* utilizando *OpenMPI* e *MPICH* no *HD*



Fonte: Autor

Figura 4.21: Desvio padrão dos tempos de execução das versões original e *isend_recv* utilizando *OpenMPI* e *MPICH* no *HD*

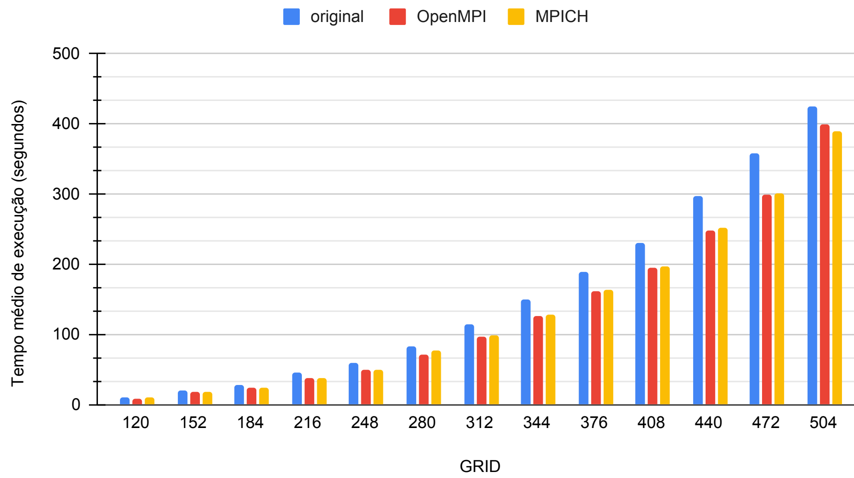


Fonte: Autor

4.5.2 Desempenho no SSD

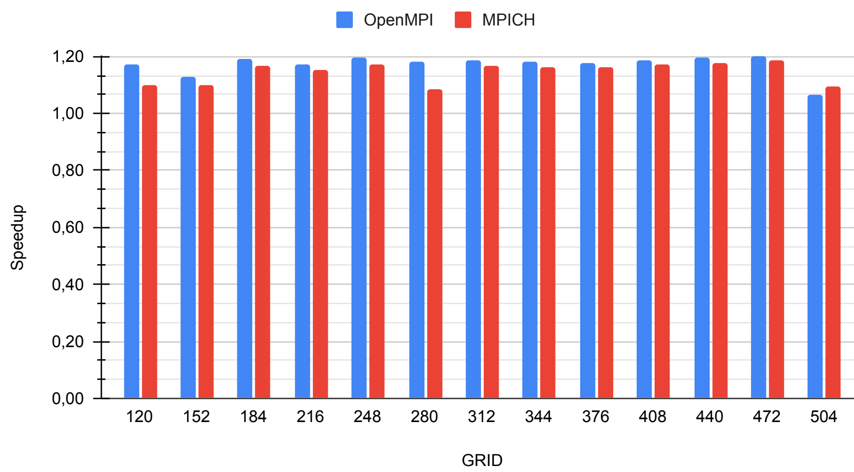
Ambas as instâncias de *isend_recv* utilizando *MPICH* e *OpenMPI* tiveram tempos melhores que a versão original. A Figura 4.22 mostra o tempo médio de execução das instâncias. O *speedup* médio da instância executadas com *OpenMPI* foi de 1,17x, enquanto o *speedup* para *MPICH* foi de 1,14x. O melhor desempenho das duas instancias foi para a *grid* de tamanho 472, onde a instância executada com *OpenMPI* teve *speedup* de 1,20 e a com *OpenMPI* teve 1,18x de *speedup*. A presença de uma *thread* de progresso não trouxe ganho de desempenho em relação à implementação sem *thread* de progresso para essa aplicação.

Figura 4.22: Tempo médio de execução da versão *isend_recv* utilizando *OpenMPI* e *MPICH* no *SSD*



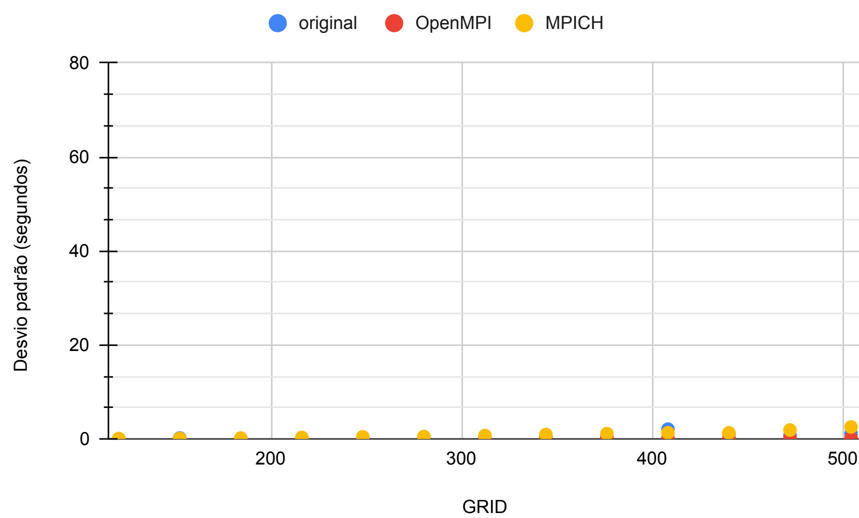
Fonte: Autor

Figura 4.23: *Speedup* da versão *isend_recv* utilizando *OpenMPI* e *MPICH* no *SSD*



Fonte: Autor

Figura 4.24: Desvio padrão dos tempos de execução das versões original e *isend_recv* utilizando *OpenMPI* e *MPICH* no *SSD*



Fonte: Autor

5 CONCLUSÃO

Analisando o desempenho das versões *MPI* com escrita em *SSD*, os melhores resultados foram obtidos pela versão *isend_recv* executada com *Open MPI*, que teve um *speedup* médio de 1,17x. Em seguida, as versões *send_recv* e *isend_recv* executadas com *MPICH* apresentaram um *speedup* de 1,14x. A utilização de mais processos para escrita em *SSD*, como nas versões *all_at_once* e *one_at_time*, não trouxeram ganhos quando comparadas às versões *send_recv* e *isend_recv*. O *overhead* do gerenciamento desses processos não se justifica, dado que o *SSD* consegue processar os dados gerados eficientemente. Com o aumento da quantidade de processos, o *overhead* cresce, impactando negativamente no desempenho.

Para os testes no *HD*, as versões *send_recv* e *isend_recv* (com ambas implementações de *MPI*) tiveram desempenho similar, ficando com 1,12x de *speedup* médio. Logo em seguida, a versão *all_at_once* com 2 processos escritores teve um *speedup* médio de 1,11x. O número de processos extra, e até mesmo a utilização de *MPI I/O*, não tiveram impacto significativo nos tempos totais. O melhor desempenho das versões *all_at_once* e *one_at_time* foi para as instâncias com 2 processos, com *speedups* de 1,11x e 1,08x, respectivamente. À medida que o tamanho das *grids* aumenta, o desempenho de todas as versões, incluindo a original, é prejudicado. Para os tamanhos de *grid* a partir de 440 os *buffers* do sistema e do *HD* são saturados e com isso o tempo de execução cresce. Outro fator que pode ser responsável pelo aumento da variação dos tempos de execução para *grids* maiores é a fragmentação do disco. Não existe garantia de que o dados serão armazenados de forma contígua, e nem mesmo que duas execuções em sequência ocupem o mesmo espaço em disco.

O ganho de desempenho obtido com as soluções propostas foi satisfatório enquanto a unidade de armazenamento não foi sobrecarregada. A utilização de *MPI I/O* pode ser melhor aproveitado quando existem escritas ocorrendo em paralelo, como foi percebido nas execuções em *HD* com tamanhos de *grid* que sobrecarregaram o *HD*. A utilização de *HD* atinge seu limite muito rápido, assim, não é uma tecnologia adequada para este tipo de aplicação. Já a utilização de *SSD* tem maior capacidade de dar vazão aos dados gerados. Pensando em simulações de maior porte, como as realizadas na indústria de óleo e gás, onde o tamanho de *grid* e volume de dados gerados é muito superior aos utilizados nos experimentos, é possível dizer que configurações com somente um *SSD* não são suficientes para dar vazão aos dados gerados sem serem sobrecarregados. Para con-

tornar essa questão, a utilização de *MPI I/O* poderia explorado com configurações com múltiplos SSDs e com a execução distribuída do método Fletcher utilizando sistema de arquivos distribuídos.

REFERÊNCIAS

AGÊNCIA NACIONAL DO PETRÓLEO, G. N. E. B. **Especial ANP 20 Anos**. 2022. Acesso em: 22 jul. 2024. Available from Internet: <https://www.gov.br/anp/pt-br/acesso-a-informacao/institucional/especial-anp-20-anos>.

AGÊNCIA NACIONAL DO PETRÓLEO, G. N. e. B. **Como funciona o processo de exploração e produção de petróleo e gás natural no Brasil**. 2023. Acesso em: 22 jul. 2024. Available from Internet: <https://www.gov.br/anp/pt-br/canais_atendimento/imprensa/kits-de-imprensa-1/como-funciona-o-processo-de-exploracao-e-producao-de-petroleo-e-gas-natural-no-brasil>.

DENIS, A. et al. **One Core Dedicated to MPI Nonblocking Communication Progression? A Model to Assess Whether It Is Worth It**. 2022. Acesso em: 22 jul. 2024. Available from Internet: <https://inria.hal.science/hal-03695835/document>.

FLETCHER, R. P.; DU, X.; FOWLER, P. J. Reverse time migration in tilted transversely isotropic (tti) media. **Geophysics**, Society of Exploration Geophysicists, v. 74, n. 6, p. 179–187, 2009.

FORUM, M. **MPI: A Message-Passing Interface Standard Version 3.1**. [S.l.], 2015. Available from Internet: <https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf>.

FORUM, W. E. **Why oil prices matter to the global economy: expert explains**. 2022. Acesso em: 22 jul. 2024. Available from Internet: <https://www.weforum.org/agenda/2022/02/why-oil-prices-matter-to-global-economy-expert-explains/>.

MPICH. **MPICH: A High-Performance Message Passing Library**. [S.l.], 2019. Acesso em: 22 jul. 2024. Available from Internet: <https://www.mpich.org/static/downloads/3.3/mpich-3.3-docs.pdf>.

OPENMPI. **Open MPI: Open Source High Performance Computing**. 2018. Acesso em: 16 fev. 2024. Available from Internet: <https://www.open-mpi.org/>.