DOI: 10.18605/2175-7275/cereus.v10n2p193-210

<<Recebido: 04/06/2018. Aceito: 13/07/2018>>

ARTIGO ORIGINAL

REVISTA CEREUS ISSN: 2175-7275

Análise do Desempenho de Comunicação Usando a Funcionalidade de Memória Compartilhada do MPI 3.0

Analysis of Communication Performance Using MPI 3.0 Shared Memory Functionality

Carlos Renato de Souza[1], Jairo Panetta[2], Stephan Stephany[3]

RESUMO

Na execução de um programa paralelizado com a biblioteca de comunicação por troca de mensagens MPI num nó computacional de memória compartilhada, a troca de mensagens entre processos pode ocasionar uma contenção pelo acesso à memória, prejudicando a escalabilidade do programa paralelo. A versão 3.0 do MPI implementou uma nova funcionalidade, a comunicação unilateral *Shared Memory* (SHM) que utiliza uma janela de memória comum aos processos executados no mesmo nó computacional na qual esses processos podem efetuar leituras e escritas diretamente, sem uso de funções MPI e sem armazenamento intermediário. Este trabalho avalia o desempenho computacional dessa nova funcionalidade do MPI na execução de um código de diferenças finitas em C e em Fortran 90 utilizando uma máquina paralela Cray. A comunicação unilateral SHM é comparada à comunicação bilateral convencional MPI.

Palavras-chave: MPI. Comunicação Unilateral. Memória Compartilhada. Diferenças Finitas.

ABSTRACT

In the execution of a program parallelized with the message passing communication library MPI in a shared memory computer node, message exchange between processes may cause contention in memory access, degrading parallel scalability. MPI 3.0 implemented a new functionality, the unilateral shared memory communication (SHM), which employs a shared memory window common to processes that are executed in the same computer node and allows these processes to perform loads and stores directly, i.e. without using MPI calls and intermediary buffers. This work evaluates the performance of this new MPI functionality in the execution of a finite differences code in C and Fortran 90 using a Cray parallel computer. SHM unilateral communication is compared to the standard bilateral MPI communication.

Keywords: MPI. One Sided Communication. Shared Memory. Finite Differences.

[1] Centro de Previsão de Tempo e Estudos Climáticos (CPTEC) / Instituto Nacional de Pesquisas Espaciais (INPE). Cachoeira Paulista, SP – Brasil.

E-mail: carlos.souza@inpe.br

[2] Divisão de Ciência da Computação (IEC) / Instituto Tecnológico de Aeronáutica (ITA). São José dos Campos, SP. Brasil.

E -mail: jairo.panetta@gmail.com

[3] Laboratório Associado de Computação e Matemática Aplicada (LAC) / Instituto Nacional de Pesquisas Espaciais (INPE). São José dos Campos, SP – Brasil.

E -mail: stephan.stephany@inpe.br

1. INTRODUÇÃO

A partir da década de 80, surgiram diversas bibliotecas proprietárias de troca de mensagens, dificultando a portabilidade de programas paralelos. Houve tentativas de padronização dessas bibliotecas, sendo a mais bem-sucedida, a biblioteca de comunicação por troca de mensagens *Message Passing Interface* (MPI), que resultou de um fórum que uniu fabricantes de computadores, desenvolvedores de software, a comunidade acadêmica e usuários. O MPI é composto pela sua API (*Application Programming Interface*) e pela sua biblioteca de funções, não sendo uma linguagem de programação. Criada em maio de 1994, sua versão 1.0 foi exclusivamente dedicada à comunicação entre processos relativos a programas escritos em linguagem Fortran 77 e C. Várias versões do MPI foram publicadas posteriormente, tais como a versão 2.0 de 1997/8 e a versão mais recente 3.1 (MPI, 2015) de 2015. Cada nova versão MPI incorpora as funcionalidades das versões anteriores, permitindo que programas escritos para essas versões sejam executados com as mais novas.

Desde sua primeira versão, o MPI suporta comunicações que envolvem apenas dois processos, denominadas ponto a ponto ou bilaterais, além de comunicações que envolvem grupos de processos, denominadas coletivas. A comunicação bilateral pode ser feita por funções diferentes, conforme as formas de envio e o recebimento das mensagens: síncrona ou assincronamente, com ou sem bloqueio, etc. Na comunicação bilateral, dois processos trocam mensagens, um enviando dados e o outro recebendo, havendo necessidade de sincronização entre eles. Visando minimizar o overhead de sincronização em certos casos, o MPI 2.0 apresentou uma nova funcionalidade em relação ao MPI 1.0: a comunicação unilateral (*One Sided Communication*), em que apenas um processo participa diretamente da comunicação ao acessar uma janela da área de memória do processo alvo para leitura ou escrita, utilizando RMA (*Remote Memory Access*). Esta funcionalidade permite o acesso remoto de um processo à memória do processo alvo executado num mesmo nó de memória compartilhada ou em nós distintos (DOBBELAERE; CHRISOCHOIDES, 2001) e (GROPP et al., 1999).

A versão MPI 3.0 apresenta novas funcionalidades de comunicação em relação ao MPI 2.0, tais como comunicações coletivas sem bloqueio e a comunicação *Shared Memory* (SHM), dentro do conceito da comunicação unilateral. A comunicação *Shared Memory* visa otimizar a comunicação unilateral de processos executados num mesmo nó de memória compartilhada, por meio da criação de uma janela visível a todos esses processos, e que

permite leituras e escritas diretas, isto é, sem uso de funções MPI. A execução concorrente de processos MPI num mesmo nó computacional gera uma contenção no acesso à memória compartilhada devido à troca de mensagens, prejudicando a escalabilidade do programa. Uma alternativa que já existia era a programação híbrida MPI + OpenMP, em que o domínio do problema seria dividido de tal forma que um ou mais processos MPI seriam executados em cada nó computacional, sendo por sua vez esses processos novamente paralelizados com threads por meio do OpenMP no nó computacional de memória compartilhada. Entretanto, programas MPI existentes, tais como modelos numéricos de previsão de tempo, não são *thread safe* e exigiriam uma extensa reescrita desses programas. A funcionalidade MPI SHM permitiria otimizar a comunicação intra-nó desses programas, deixando a comunicação inter-nó inalterada, com alterações significativamente menores nesses programas.

Este trabalho visa comparar o desempenho da comunicação MPI SHM com a comunicação bilateral convencional MPI para um estudo de caso particular, o cálculo de um estêncil 2D relativo à resolução de equações diferenciais parciais por diferenças finitas, executado em nós de memória compartilhada de uma máquina Cray, sendo exploradas as linguagens C e Fortran 90. As análises feitas em Fortran 90 são importantes para avaliar o desempenho do uso do MPI SHM em aplicações científicas implementadas em Fortran 90, como por exemplo o modelo numérico de previsão de tempo BRAMS (*Brazilian developments on the Regional Atmospheric Modeling System*) (FREITAS et al., 2016).

2. A COMUNICAÇÃO UNILATERAL EM MPI

O escopo deste trabalho é analisar a comunicação unilateral MPI *Shared Memory*, mas detalha-se a seguir a comunicação unilateral por *Remote Memory Access* (RMA), que a antecedeu permitindo a comunicação entre processos no mesmo ou entre nós diferentes.

2.1. Comunicação Unilateral por RMA

Na comunicação unilateral (*One Sided Communication*) por RMA, apenas um processo participa da comunicação ao acessar uma janela da área de memória do processo alvo utilizando RMA, mas ainda com necessidade de sincronização entre os processos (LOPES, 2010). Essa janela de memória é criada pelo processo alvo pela função MPI MPI_Win_create(), sendo acessível a todos os outros processos envolvidos no comunicador para escrita ou leitura utilizando respectivamente as funções MPI_Put() e

MPI_Get(). Essa janela é denominada janela local compartilhada RMA ou, simplesmente, janela RMA. Na comunicação unilateral por RMA a sincronização é provida por trechos de código denominados "épocas" (*epoch*). Uma forma de estabelecer uma época é delimitar o trecho desejado por invocações à função MPI_Win_fence() que são coletivas a todos processos que acessam a janela RMA. A invocação dessa função no início da época garante que os dados expostos na janela estão atualizados e elimina qualquer cópia local (cache e registradores) desses dados, os quais podem ser escritos ou lidos com funções MPI pelos demais processos envolvidos pela comunicação unilateral até a segunda chamada da função MPI_Win_fence(), que encerra a época.

A Figura 1 (esquerda) ilustra a comunicação unilateral ocorrendo dentro de uma época em uma janela RMA é disponibilizada pelo processo P1 e acessada para leitura e escrita pelos processos P0 e P2 utilizando também a função MPI_Accumulate() que permite efetuar reduções. A Figura 1 (direita) também ilustra a comunicação RMA para dois processos que disponibilizam janelas RMA aos demais processos (vermelhas) em suas áreas de memória local (cinzas). É exemplificada uma escrita (MPI_Put()) do processo 0 na janela RMA do processo 1 e uma leitura (MPI_Get()) do processo 1 na janela RMA do processo 0, referentes a variáveis diferentes. Note-se que tanto essa escrita como essa leitura ocorrem independentemente, com apenas a participação do processo que chamou a função.

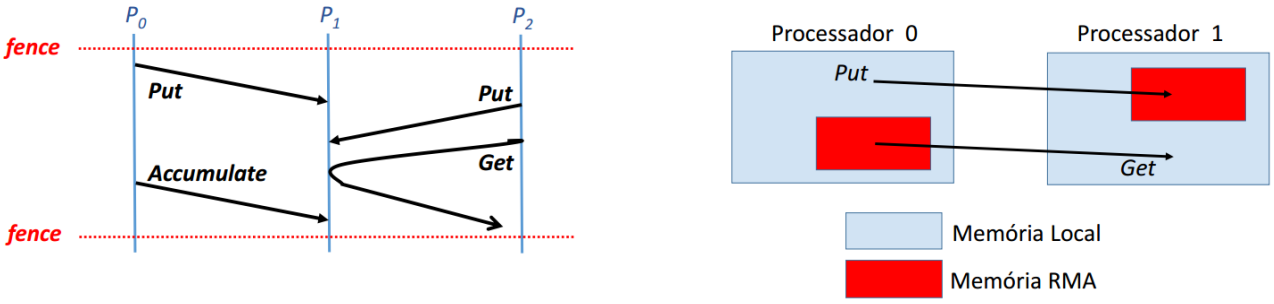

Figura 1. (À esquerda) Definição de uma época delimitada por duas chamadas da função MPI_Fence() para sincronização entre processos na comunicação unilateral RMA. (À direita) Esquema ilustrativo de uma leitura e uma escrita independentes na comunicação entre dois processos que disponibilizam janelas RMA utilizando as funções MPI_Put() e MPI_Get(). Fonte: Balaji et al. 2014.

2.2. A Comunicação Unilateral por Memória Compartilhada

A comunicação unilateral por memória compartilhada (MPI *Shared Memory*) foi introduzida na versão MPI 3.0, sendo aplicada a processos executados num mesmo nó de memória compartilhada, os quais podem fazer leituras e escritas diretas numa janela de

memória compartilhada (GROPP, 2016]. Como um programa MPI pode utilizar mais de um nó computacional, foi necessária a criação de funções específicas para identificar e mapear os processos que estão no mesmo nó. Uma dessas funções é a MPI_Comm_split_type(), que possibilita dividir o comunicador global em comunicadores disjuntos específicos de cada nó computacional de memória compartilhada (utilizando o argumento *split_type* do tipo *MPI_COMM_TYPE_SHARED*). Outra função importante é a MPI_Group_translate_ranks(), que mapeia os processos do grupo do comunicador global em processos do grupo do comunicador local a cada nó, que é do tipo *MPI_COMM_TYPE_SHARED*. Uma vez identificados quais processos são executados num mesmo nó, pode-se alocar uma área de memória e compartilhá-la entre eles com a função MPI_Win_allocate_shared() a qual cria e aloca uma área de memória chamada de janela de memória compartilhada, visível para todos os processos executados nesse nó e dividida entre eles. Uma outra função necessária na comunicação MPI *Shared Memory* é a MPI_Win_shared_query(), que fornece o endereço da parte da janela compartilhada de outro processo do sub-comunicador local para o processo que a executa efetuar leituras ou escritas diretas.

A Figura 2 (esquerda) exemplifica 12 processos executados em 3 nós computacionais de memória compartilhada, com a definição dos 3 sub-comunicadores locais a cada nó, cada um com 4 processos, e com as 3 correspondentes janelas de memória compartilhada acessíveis aos processos locais.

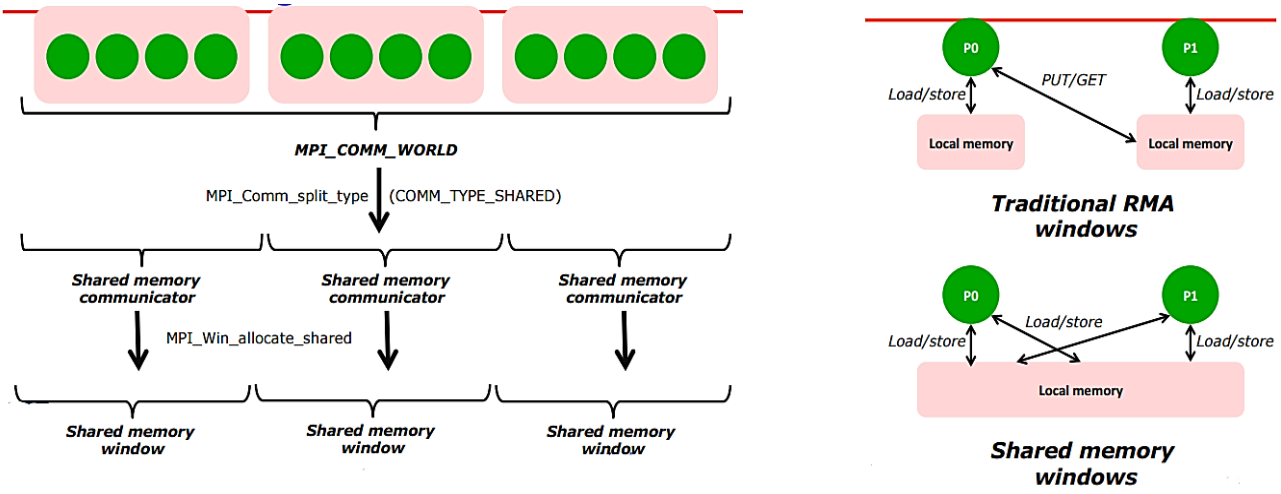

Figura 2. (À esquerda) Divisão do comunicador global em subgrupos de comunicadores específicos de cada nó de memória compartilhada. (À direita) Comparação da comunicação MPI unilateral com janela RMA e com janela SHM.

A Figura 2 (direita) ilustra a diferença entre a comunicação unilateral RMA e SHM. Na primeira, leituras e escritas são feitas pelas funções PUT/GET, e os processos envolvidos na comunicação não são necessariamente locais a um nó computacional, enquanto que na segunda, leituras e escritas são feitas diretamente, mas os processos têm que ser locais ao nó.

Na opção *default*, a janela de memória compartilhada é criada em endereços contíguos de memória pela função MPI_Win_allocate_shared(), se o terceiro argumento desta função for informado como *MPI_INFO_NULL*. Isso pode limitar o desempenho em algumas arquiteturas porque não permite que os dados sejam alocados nas regiões de memória mais próximas aos processadores, a fim de se reduzir latência de acesso. Definindo-se este terceiro argumento como *alloc_shared_noncontig* com valor *true*, a janela de memória compartilhada é criada para cada processo em um local da memória próximo desse processo, criando segmentos de memórias não contíguas, melhorando o desempenho do acesso à memória, dependendo de cada arquitetura [mpi 2015]. O acesso concorrente dos processos locais de um nó a uma janela de memória compartilhada exige também sincronização, pela definição de uma "época" de forma a organizar os acessos, à semelhança de como é feito na comunicação unilateral RMA.

Considerando-se a comunicação MPI entre processos executados em nós diferentes (inter-nó) e entre processos executados no mesmo nó (intra-nó), pode-se combinar a comunicação unilateral RMA no caso intra-nó e a SHM no caso inter-nó. Da mesma forma, pode-se utilizar a comunicação bilateral no caso inter-nó, em vez da comunicação RMA, combinada com a comunicação unilateral SHM no caso intra-nó, como neste trabalho. Isso deve-se à possibilidade de portar a comunicação bilateral de códigos legados existentes tais como modelos numéricos para a comunicação unilateral SHM no caso intra-nó (HOEFLER et al., 2013).

3. ESTUDO DE CASO DAS DIFERENÇAS FINITAS

Adotou-se como estudo de caso neste trabalho um programa implementado originalmente por Hoefler e Balaji (BALAJI et al., 2014], o qual resolve equações diferenciais parciais (EDP's) pelo método das diferenças finitas numa malha 2D e que requer o cálculo de um estêncil de 5 pontos. A resolução de EDP's por diferenças finitas é uma aproximação comum em modelos de previsão numérica de tempo, nas quais variáveis contínuas da atmosfera são discretizadas. Assim, o estudo de caso escolhido, embora em menor escala, tem relevância para a otimização de tais modelos.

Como o método dos elementos finitos resultantes da subdivisão do domínio do problema e concebido para problemas de valores de contorno. Há também o método dos volumes finitos, largamente utilizado na resolução de problemas envolvendo transferência de calor ou massa e em mecânica dos fluidos, o qual constitui uma evolução do método das diferenças finitas ao incorporar resolução de balanços de massa, energia e quantidade de movimento aos volumes finitos resultantes da subdivisão do meio contínuo do domínio do problema. A programação em MPI destes métodos é semelhante, envolvendo cálculos feitos a cada passo de tempo em partições do domínio, troca de dados relativos à fronteira entre subdomínios atribuídos a processos diferentes e também reduções envolvendo quantidades escalares e vetoriais desses subdomínios.

O estudo de caso escolhido envolve a resolução de uma EDP, a equação de Poisson em duas dimensões, muito utilizada em dinâmica dos fluidos. Trata-se da aplicação específica apresentada por (BALAJI et al., 2014) para modelar a distribuição de calor numa superfície:

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x,y) \tag{1}$$

Acima, “U” é definido numa malha discreta em duas dimensões (x,y) de resolução Δx = Δy = h. Aproximando-se as derivadas parciais de segunda ordem pelo método de diferenças finitas centradas, tem-se:

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U(x+h,y) - 2U(x,y) + U(x-h,y)}{h^2} \tag{2}$$

Simplificando-se a notação com $U(x,y) = U_{i,j}, U(x+h,y) = U_{i+1,j}$ vem:

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} \tag{3}$$

Finalmente, a aproximação por diferenças finitas aparece na equação abaixo:

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \approx \frac{U_{i+1,j} + U_{i,j+1} - 4U_{i,j} + U_{i-1,j} + U_{i,j-1}}{h^2} \tag{4}$$

A malha discretizada correspondente ao domínio do problema aparece na Figura 3 (esquerda), sendo particionada entre os processos pelas linhas verdes. A cada iteração do laço correspondente aos passos de tempo, cada ponto de grade da malha discretizada é atualizado por um estêncil de 5 pontos, ou seja, pela média de seu próprio valor e dos valores de 4 pontos vizinhos. Na mesma figura, para um subdomínio, há uma cruz vermelha que mostra os pontos envolvidos na atualização do ponto central da cruz. Na primeira iteração, cada ponto da malha é inicializado com valores de temperatura zero, sendo que a cada iteração introduz-se calor em determinados pontos da malha escolhidos aleatoriamente. Note-se que aparece na figura uma segunda cruz com um círculo em rosa acima, mostrando que num ponto de grade situado na borda, ou seja, na fronteira entre subdomínios, aparece um ponto de grade fora do subdomínio, que é atualizado por outro processo.

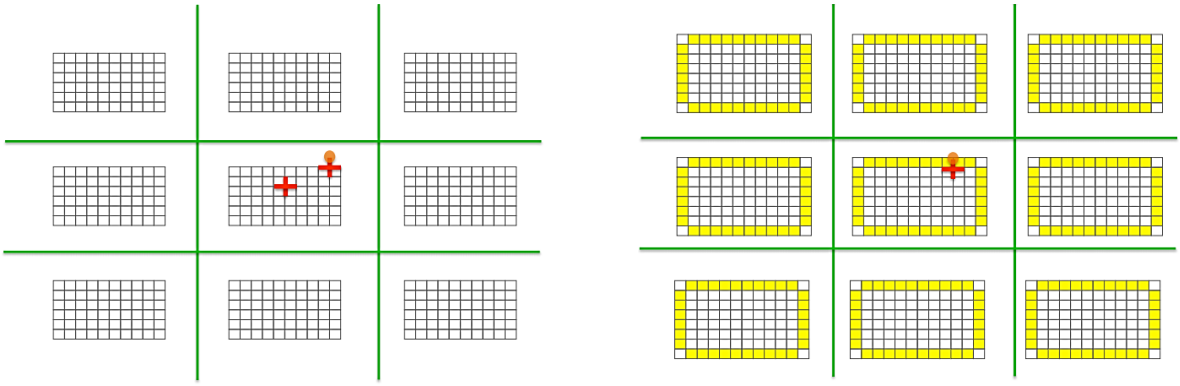

Figura 3. (À esquerda) Malha discretizada original do problema dividida em subdomínios com (bx) x (by) pontos, atribuídos aos processos. (À direita) Malha discretizada original do problema dividida em subdomínios acrescidos das correspondentes *ghost zones* (bx+2) x (by+2) pontos, atribuídos aos processos. Fonte: Balaji et al 2014.

O problema da atualização dos pontos na borda de cada subdomínio é resolvido pela adição de uma fileira extra de pontos de grade ao longo de cada borda, denominada *ghost zone* ou *halo zone*, conforme a Figura 3 (direita). Essa dependência de dados entre processos força a comunicação entre processos, na qual cada processo envia e recebe dos processos a cargo dos subdomínios vizinhos os valores correspondentes às bordas necessárias à atualização da grade. Se cada subdomínio tiver uma dimensão original (bx, by), passará a ter então uma dimensão (bx+2, by+2) com a adição da *ghost zone,* considerando-se o estêncil de 5 pontos.

4. TESTES DE DESEMPENHO COMPUTACIONAL

4.1. Ambiente Computacional

Os testes de desempenho foram executados em nós computacionais biprocessados de uma máquina Cray com processadores Intel Xeon *Broadwell* do tipo E5-2699.v4 de 2,2 GHz e 22 núcleos (44 por nó) com cache L1 de 64KB por núcleo, L2 de 256KB por núcleo e L3 compartilhado de 55MB. O ambiente de compilação escolhido foi o conjunto de compiladores PGI versão 16.10.0 e a biblioteca Cray-MPI.

4.2. Análise dos Resultados de Desempenho para Execução em Nó Único

Os códigos originais foram implementados e disponibilizados por Hoefler e Balaji [Balaji et al. 2014] originalmente em Linguagem C. Três versões foram implementadas e testadas neste trabalho: a versão sequencial, a versão paralela MPI com comunicação bilateral com as funções de envio e recebimento de mensagens com bloqueio MPI_Send() e MPI_Recv() (denotadas aqui por S/R) e a versão paralela MPI comunicação unilateral *Shared Memory* com alocação de memória contígua (SHM). Uma quarta versão, derivada desta última, foi criada com a janela de memória compartilhada não-contígua (nomeada por SHM-NC). Foi utilizada uma malha bidimensional discreta de 4.800 x 4.800 pontos, com 500 iterações no tempo e adicionando-se uma unidade de energia a cada passo de tempo em três posições da grade escolhidas aleatoriamente e mantidas fixas ao longo das iterações. As versões paralelas foram compiladas com o compilador PGI 17.10.0 com opções *default*, exceto pela opção *-O3*, sendo executadas com 1, 4, 9, 16, 25 e 36 processos num único nó computacional, pois as codificações admitem que cada eixo do domínio seja particionado pelo mesmo número de processos MPI.

A Tabela 1 mostra os tempos de execução dessas versões em linguagem C num único nó (média de 5 execuções para cada caso) em função do número de processos, notando-se que as versões C MPI SHM são mais rápidas para até 16 processos e que à medida que se aumenta o número de processos, a versão SHM-NC tende a ficar mais rápida que a SHM e a ter um tempo ligeiramente pior que a versão MPI S/R.

Tabela 1. Tempos de execução em segundos, *speedup's* e eficiências para as versões paralelas em linguagem C MPI S/R, MPI SHM e MPI SHM-NC compiladas com PGI e executadas com 1, 4, 9, 16, 25 e 36 processos num único nó computacional para o domínio de 4.800 x 4.800 pontos com 500 passos de tempo.

Seq. = 43,14s	**1P**	**4P**	**9P**	**16P**	**25P**	**36P**
MPI S/R	43,31s	10,88s	5,22s	4,51s	3,90s	2,77s
MPI SHM	38,08s	9,48s	5,02s	4,52s	4,59s	5,64s
MPI SHM-NC	37,95s	9,64s	4,98s	4,45s	4,23s	2,88s
			Speedup			
MPI S/R	0,99	3,97	8,26	9,56	11,06	15,58
MPI SHM	1,13	4,55	8,59	9,53	9,39	7,65
MPI SHM-NC	1,14	4,47	8,67	9,70	10,19	14,98
			Eficiência			
MPI S/R	0,99	0,99	0,92	0,60	0,44	0,43
MPI SHM	1,13	1,14	0,95	0,60	0,38	0,21
MPI SHM-NC	1,14	1,12	0,96	0,61	0,41	0,42

Um fato curioso é que as versões SHM executadas com 1 único processo foram mais rápidas que a própria versão sequencial. A mesma tabela e também a Figura 4 apresentam os correspondentes *speedup's* e eficiências, calculados em relação ao tempo da versão sequencial. Notam-se *speedup's* ligeiramente superlineares para as versões SHM e SHM-NC executadas com até 4 processos, mas o desempenho destas se degrada com o aumento do número de processos, com ênfase para a versão SHM, enquanto que a versão SHM-NC teve desempenho paralelo próximo da versão S/R. Nota-se uma degradação do desempenho de todas as versões paralelas C com o aumento do número de processos.

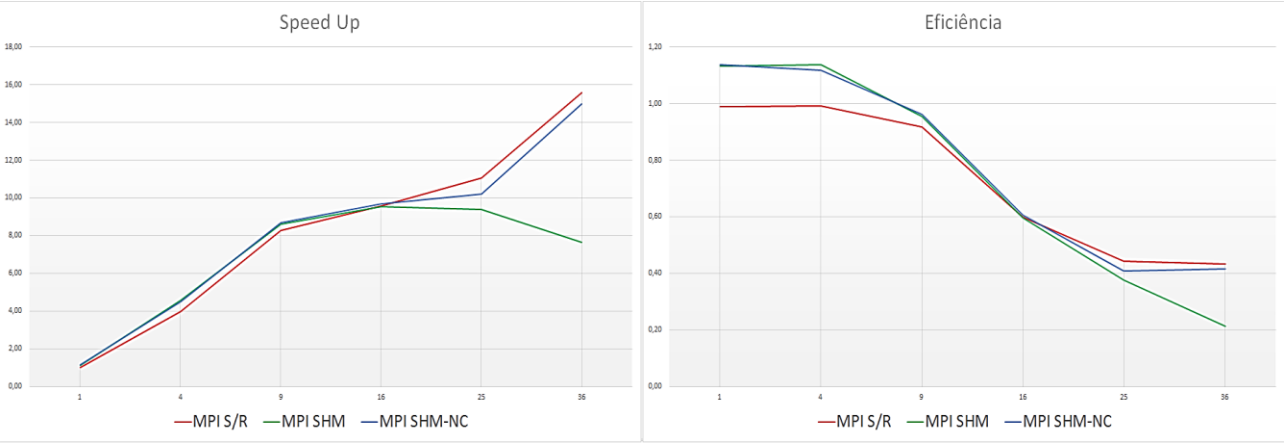

Figura 4. Desempenho paralelo das versões em linguagem C MPI S/R (em vermelho), MPI SHM (em verde) e MPI SHM-NC (em azul) compiladas com PGI e executadas com 1, 4, 9, 16, 25 e 36 processos num único nó computacional para o domínio de 4.800 x 4.800 pontos com 500 passos de tempo. (À esquerda) Gráficos de *speedup* e (à direita) de eficiência em função do número de processos.

Essas versões foram convertidas para Fortran 90 com o objetivo de comparar o desempenho do MPI SHM em ambas as linguagens e porque modelos numéricos de previsão de tempo e clima, como o modelo BRAMS por exemplo, são escritos em Fortran 90 [Freitas et al. 2016]. Assim, foram também geradas as quatro versões correspondentes àquelas em C (Fortran 90 sequencial, MPI S/R, MPI SHM e MPI SHM-NC) e uma quinta versão Fortran 90 denominada híbrida, que combina comunicação inter-nó S/R e comunicação intra-nó SHM-NC. Todas essas versões paralelas foram compiladas com o compilador PGI com opções *default*, exceto pela opção *-O3*, sendo executadas com 1, 4, 9, 16, 25 e 36 processos num único nó computacional (exceto pela versão híbrida) para o mesmo domínio de 4.800 x 4.800 pontos.

A Tabela 2 mostra os tempos de execução dessas versões Fortran 90 num único nó (média de 5 execuções para cada caso) em função do número de processos, notando-se que as versões Fortran 90 S/R são mais rápidas que as versões SHM e SHM-NC para qualquer número de processos, embora à medida que se aumenta o número de processos, a versão SHM-NC tende a melhorar seu desempenho em relação à versão S/R, a ficar mais rápida que a SHM e a ter um tempo ligeiramente pior que a versão MPI S/R.

Tabela 2. Tempos de execução em segundos, *speedup's* e eficiências para as versões em linguagem Fortran 90 MPI S/R, MPI SHM e MPI SHM-NC compiladas com PGI e executadas com 1, 4, 9, 16, 25 e 36 processos num único nó computacional para o domínio de 4.800 x 4800 pontos com 500 passos de tempo.

Seq. = 33,30s	**1P**	**4P**	**9P**	**16P**	**25P**	**36P**
MPI S/R	34,67s	7,93s	4,72s	4,40s	3,87s	2,67s
MPI SHM	46,18s	11,79s	5,56s	4,68s	5,26s	5,81s
MPI SHM-NC	46,12s	11,65s	5,54s	4,62s	4,90s	2,79s
			Speedup			
MPI S/R	0,96	4,20	7,06	7,56	8,61	12,46
MPI SHM	0,72	2,82	5,99	7,12	6,33	5,73
MPI SHM-NC	0,72	2,86	6,01	7,21	6,80	11,94
			Eficiência			
MPI S/R	0,96	1,05	0,78	0,47	0,34	0,35
MPI SHM	0,72	0,71	0,67	0,44	0,25	0,16
MPI SHM-NC	0,72	0,71	0,67	0,45	0,27	0,33

A mesma tabela e também a Figura 5 apresentam os correspondentes *speedup's* e eficiências, calculados em relação ao tempo da versão sequencial. Nota-se uma degradação do desempenho de todas as versões paralelas Fortran 90 com o aumento do número de processos.

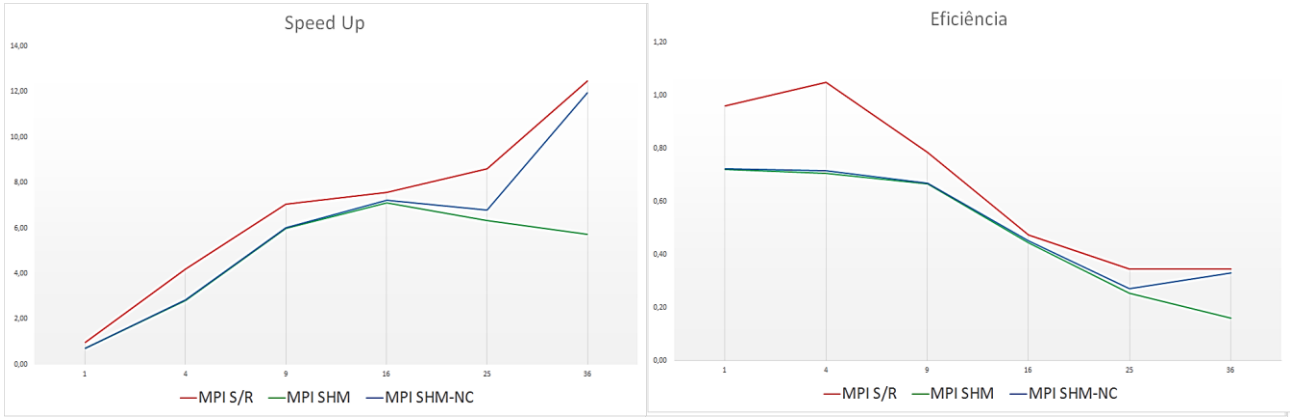

Figura 5. Desempenho paralelo das versões em linguagem Fortran 90 MPI S/R (em vermelho), MPI SHM (em verde) e MPI SHM-NC (em azul) compiladas com PGI e executadas com 1, 4, 9, 16, 25 e 36 processos num único nó computacional para o domínio de 4.800 x 4.800 pontos com 500 passos de tempo. (À esquerda) Gráficos de *speedup* e (à direita) da eficiência em função do número de processos.

Finalmente, a Figura 6 mostra a diferença percentual do tempo de execução das versões Fortran 90 em relação às correspondentes versões C em função do número de processos, expressa por $(T_f/T_c - 1)x100$. As versões Fortran 90 S/R mostram valores negativos dessa diferença porcentual por terem apresentado tempos de execução menores que as correspondentes versões C, embora a diferença fique menor à medida que se aumenta o número de processos. Por outro lado, as versões Fortran 90 MPI SHM e SHM-NC apresentam valores positivos dessa diferença porcentual pois têm tempos de execução menores que as correspondentes versões em C.

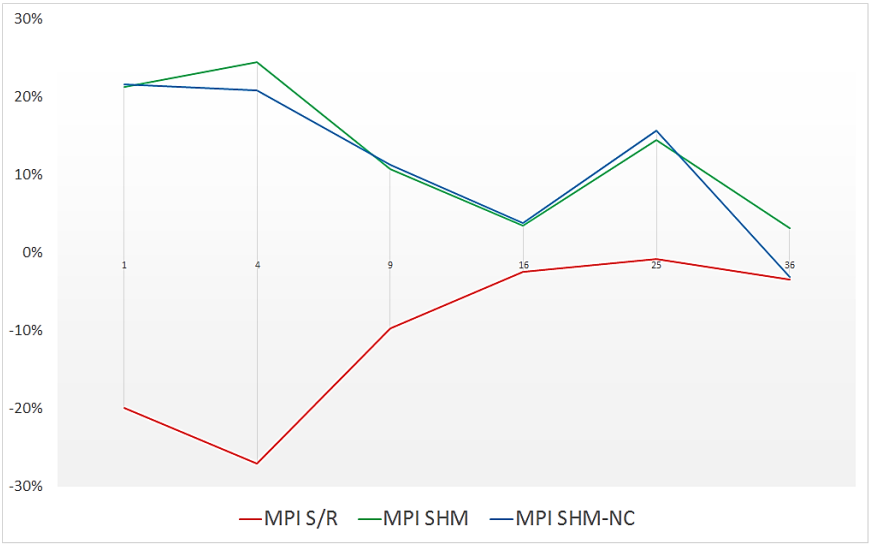

Figura 6. Diferença porcentual das versões Fortran 90 em relação às correspondentes versões em C função do número de processos (MPI S/R em vermelho, MPI SHM em verde e MPI SHM-NC em azul).

Em tese, as versões com a comunicação MPI SHM deveriam apresentar melhor desempenho tanto em C quanto em Fortran 90, uma vez que essa funcionalidade foi criada

para explorar a memória compartilhada na comunicação unilateral. Os testes indicaram que as versões C MPI SHM eram mais rápidas que as correspondentes versões S/R, para um número baixo de processos por nó, porém isso não aconteceu com as versões Fortran 90. Esse fato levou à investigação das diferenças de desempenho entre as versões das duas linguagens.

O tempo de execução da versão Fortran 90 S/R executada com um processo (1P) é muito próximo da versão Fortran 90 sequencial (4% maior), mas as versões Fortran 90 SHM 1P demandam quase 40% mais tempo. Uma simples comparação entre o código das versões MPI SHM C e Fortran 90 mostra que esse tempo adicional poderia ser causado pela conversão de ponteiros C para ponteiros Fortran 90, que obviamente não ocorre na versão C, conforme abordado na próxima subseção.

4.3. Comparação de Versões Sequenciais Fortran 90

As janelas de memórias compartilhada do MPI SHM são criadas pela função MPI_Win_allocate_shared() que retorna um endereço de memória por meio de um ponteiro, mas este ponteiro é do tipo C (TYPE(C_PTR)), o qual precisa ser convertido para um ponteiro Fortran 90 (*pointer*) utilizando-se a função c_f_pointer(), como ilustrado no trecho de programa a seguir para a matriz “avector”. Por outro lado, as versões Fortran 90 sequencial e MPI S/R não necessitam esse esquema de conversão de ponteiros.

Em consequência disso, foram feitos testes com diferentes versões sequenciais com o objetivo de identificar qual o tempo adicional demandado ao se utilizar ponteiros do tipo TYPE(C_PTR) em Fortran 90 em comparação com a utilização usual de alocação dinâmica da matriz, ou ainda de um ponteiro para matriz. Assim, foram desenvolvidas 3 versões, detalhadas a seguir, nas quais diferentes formas de declaração da matriz "avector" foram utilizadas. O trecho de código mostrado abaixo mostra a primeira versão sequencial do programa que emprega uma matriz alocada dinamicamente, denominada de versão "F-Alloc":

```
1 double precision, dimension(:, :), allocatable  :: avector
2 integer                                         :: n
3 ! Alocando a matriz:
4 allocate(avector(n, n))
5
6 avector(i, j)=0.0
```

Uma segunda versão sequencial do código foi criada com o uso de ponteiro Fortran 90, sendo denominada versão "F-Point":

```
double precision, dimension(:, :), pointer  :: avector
integer                                     :: n
! Alocando a matriz:
allocate(avector(n, n))

avector(i, j)=0.0
```

Finalmente, a terceira versão sequencial do código, que utiliza o conversor de ponteiros c_f_pointer() de C para Fortran 90, sendo denominada versão "F-Cptr":

```
USE, INTRINSIC                              :: ISO_C_BINDING
TYPE(C_PTR)                                 :: mem1
TYPE(MPI_Win)                               :: win1
integer, dimension(:), allocatable          :: memshape
double precision, dimension(:, :), pointer  :: avector
double precision                            :: heat=0.0
integer                                     :: size, n
! Atribuindo o shape dos ponteiros Fortran:
allocate(memshape(2))
! Atribuindo o tamanho de cada dimensao dos ponteiros Fortran:
memshape(1)=n
memshape(2)=n
size=n*n*c_sizeof(heat)
! Alocando a janela SHM :
call MPI_Win_allocate_shared(size, 1, MPI_INFO_NULL, shmcomm, mem1, win1, ierror)
! Convertendo o endere o SHM de ponteiro C para Fortran:
call c_f_pointer(mem1, avector, memshape)
avector(i, j)=0.0
```

Essas 3 versões sequenciais foram então executadas para 100 iterações simulando passos no tempo, sendo a matriz "avector" declarada com dimensão de 10.000 x 10.000 elementos. Cada versão foi temporizada de forma a se medir o tempo de alocação e inicialização dessa matriz, o tempo de acesso à matriz para leitura dos valores correntes e o tempo de acesso para escrita dos valores atualizados, a cada iteração correspondente a um passo de tempo. Esses tempos foram somados resultando nos tempos acumulados mostrados na Tabela 3, denominados respectivamente, de tempos de "inicialização", de "leitura de valor corrente" e de "escrita de valores atualizados" para as três versões F-Alloc, F-Point e F-Cptr.

A Tabela 3 mostra os tempos acumulados referentes à execução das 3 versões sequenciais (F-Alloc, F-Point e F-Cptr) compiladas com o compilador da PGI, opções *default*, exceto pela opção -O3. Nota-se que as versões que utilizam ponteiros demandam muito tempo na leitura da matriz, mas realizam sua escrita/atualização num tempo quase nulo, sendo que a versão F-Cptr gasta 35% a mais de tempo que a versão F-point, provavelmente devido ao uso do conversor c_f_pointer(). Entretanto, comparando-se as versões F-Alloc e F-point, nota-se que a primeira é mais rápida na leitura da matriz, porém mais lenta na escrita/atualização, sendo que o total de tempos acumulados de ambas é praticamente igual. Essa comparação de tempos acumulados dessas 3 versões sequenciais foi também efetuada com os compiladores Intel, GNU e CCE (suíte de compiladores da Cray), com resultados similares. Uma vez que a comunicação MPI unilateral SHM em Fortran 90 exige o uso desse conversor de ponteiros, isso pode explicar seu pior desempenho paralelo, mas não permite ainda excluir outros fatores inerentes a essa forma de comunicação.

Tabela 3. Tempos acumulados de execução para 100 iterações no tempo (em segundos) das versões sequencias implementadas em Fortran 90, com a matriz "avector" de dimensão 10.000 x 10.000 alocada dinamicamente (F-Alloc), alocada com ponteiro Fortran 90 (F-Point) e alocada com uso do conversor c_f_pointer() (F-Cptr).

Versões	**F-Alloc**	**F-Point**	**F-Cptr**
Inicialização	3	3	2
Leitura de valor corrente	95	162	218
Escrita de valores atualizados	70	~0	~0
Total	168	165	220

4.4. Análise dos Resultados de Desempenho para Execução com Vários Nós

Os códigos referentes ao mesmo problema de diferenças finitas foram executados utilizando vários nós computacionais da mesma máquina, descrita na Seção 4.1, sendo empregadas as versões MPI Fortran 90 S/R com comunicação convencional (S/R) e a versão denominada híbrida, que combina a comunicação inter-nó S/R com a comunicação intra-nó SHM-NC. O domínio do problema também foi de 4.800 x 4.800 pontos, mas agora com 5.000 iterações no tempo, sendo utilizados até 64 nós computacionais da máquina, sempre com o número máximo de núcleos por nó (44), num total de até 2.304 núcleos.

A Tabela 4 mostra os tempos de execução dessas duas versões em função do número total de processos (N1), do número de processos por nó (N2) e do número de nós computacionais (N3), sendo que N1 = N2 x N3. Esses tempos correspondem à média de 5

execuções para cada caso. Nessa mesma tabela, aparecem os correspondentes valores de *speedup's* e eficiências, sempre calculados em relação à versão sequencial Fortran 90. Analogamente à Seção 4.2, foram usadas as opções default do compilador PGI, exceto pela opção -O3.

Tabela 4. Tempos de execução (em segundos) e correspondentes *speedup's* e eficiências das versões Fortran 90 paralelas MPI S/R e MPI S/R + SHM-NC compiladas com PGI e executadas com N1 processos, alocados em N2 núcleos/nó em N3 nós computacionais para o domínio de 4.800 x 4.800 pontos com 5.000 passos de tempo.

[N1-N2-N3]	**MPI S/R**			**MPI S/R + SHM-NC**		
Seq. = 333,59s	Tempo (s)	*Speedup*	Eficiência	Tempo (s)	*Speedup*	Eficiência
1-1-1	343,84	0,97	0,97	450,87	0,74	0,74
4-4-1	78,36	4,26	1,06	113,66	2,93	0,73
9-9-1	48,07	6,94	0,77	54,05	6,17	0,69
16-16-1	44,36	7,52	0,47	44,85	7,44	0,46
25-25-1	38,56	8,65	0,35	40,41	8,26	0,33
36-36-1	26,82	12,44	0,35	28,47	11,72	0,33
64-32-2	15,26	20,52	0,32	16,77	19,89	0,31
100-25-4	8,17	40,83	0,41	10,06	33,16	0,33
144-36-4	2,61	127,81	0,89	6,24	53,46	0,37
225-25-9	1,53	218,03	0,97	4,60	72,52	0,32
256-32-8	1,36	245,29	0,96	4,33	77,04	0,30
400-40-10	0,94	354,88	0,89	3,14	106,24	0,27
576-36-16	0,70	476,56	0,83	2,28	146,31	0,25
625-25-25	0,63	529,51	0,85	1,88	177,44	0,28
900-36-25	0,48	694,98	0,77	1,61	207,20	0,23
1024-32-32	0,43	775,79	0,76	1,43	233,28	0,23
1600-40-40	0,38	877,87	0,55	1,29	258,60	0,16
2304-36-64	0,36	926,64	0,40	1,05	317,70	0,14

Nessa tabela, nota-se que, em todos os casos, a versão Fortran 90 MPI S/R foi mais rápida do que a Fortran 90 S/R + SHM-NC, demonstrando que esta última é penalizada seja pela conversão de ponteiros C para Fortran 90 seja pela saturação do uso da memória em cada nó. A eficiência da versão S/R é alta para até 4 processos, decai de 16 até 100 processos para valores abaixo de 50%, mas sobe significativamente a partir 144 processos, apresentando valores entre 70% e 80%, provavelmente devido ao tamanho do cache L3 para o problema considerado. A eficiência apenas decai para 1.600 e 2.304 processos, provavelmente devido à baixa granularidade decorrente para o problema considerado. Em comparação, a versão S/R + SHM-NC teve eficiências piores para qualquer número de

processos, especialmente a partir de 144 processos, casos em que seus valores foram inferiores à metade das eficiências da outra versão.

5. CONCLUSÕES

Neste trabalho objetivou-se avaliar se a comunicação unilateral *Shared Memory* (SHM) implementada no MPI 3.0 otimiza o desempenho de comunicação em nós de memória compartilhada, comparando-a com comunicação MPI bilateral convencional. Testes de desempenho paralelo foram realizados utilizando-se uma máquina Cray composta de vários nós computacionais na execução de códigos nas linguagens C e Fortran 90, correspondentes a um problema exemplo para resolução de equações diferenciais parciais pelo método das diferenças finitas. Os testes foram feitos para números variáveis de núcleos em um único nó computacional, sempre com a comunicação MPI SHM, ou então, em vários nós, com uma versão híbrida, em que se utilizou a comunicação MPI SHM na comunicação intra-nó e MPI bilateral convencional na comunicação inter-nó. A comunicação unilateral MPI SHM foi concebida para obter melhor desempenho na execução de vários processos num ambiente de memória compartilhada, mas os resultados de desempenho obtidos neste trabalho demonstram que a comunicação bilateral convencional ainda apresenta desempenho melhor, seja na execução utilizando-se um único nó computacional, seja em vários nós.

Aparentemente, tanto em C como em Fortran 90, a criação de uma janela de memória compartilhada comum aos processos executados num mesmo nó gera uma contenção no acesso à memória que penaliza o desempenho paralelo mais do que a troca de mensagens na comunicação MPI bilateral convencional S/R. Isso pode ser demonstrado pela degradação do desempenho relativo entre a comunicação bilateral e a unilateral SHM à medida que se saturam os núcleos disponíveis em cada nó computacional. Naturalmente, esses tempos referem-se a códigos compilados com PGI, executados com processadores *Broadwell* numa máquina paralela Cray específica, mas é provável que o desenvolvimento das máquinas recentes, sua arquitetura de processadores e de memória, além do próprio sistema operacional, favoreceram de alguma forma a comunicação MPI bilateral convencional. Assim, pode-se inferir que a comunicação unilateral SHM em MPI ainda não foi desenvolvida suficientemente. No caso da versão Fortran 90 do código com comunicação MPI unilateral SHM, a comparação de seu desempenho paralelos com a versão C permitiu mostrar que a versão Fortran 90 foi também penalizada pela necessidade

de utilizar de ponteiros C na alocação da janela SHM, a qual exige a conversão para ponteiros Fortran 90, custosa em termos de tempo.

O objetivo deste trabalho foi investigar o desempenho de da comunicação MPI unilateral SHM em Fortran 90 para explorar seu uso na otimização de modelos numéricos de previsão de tempo e clima, geralmente escritos em Fortran 90. Esta possibilidade constitui uma nova alternativa na execução intra-nó de processos em relação à programação mista MPI-OpenMP na qual o domínio do problema seria dividido entre nós entre processos MPI e, localmente a cada nó, dividido entre *threads* OpenMP. Entretanto, esta última abordagem torna-se inviável devido ao um grande esforço de reprogramação para tornar o modelo *thread-safe* como requerido na programação por *threads*. Uma continuação deste trabalho resultou na avaliação de desempenho paralelo da comunicação MPI unilateral SHM no modelo numérico BRAMS (SOUZA et al., 2018).

REFERÊNCIAS

BALAJI, P., GROPP, W., HOEFLER, T., and THAKUR, R. **Advanced MPI Programming Tutorial**. 2014.

DOBBELAERE, J. and CHRISOCHOIDES, N. **One-Sided Communication Over MPI-1**. 2001.

FREITAS, S. et al. **The Brazilian Developments on the Regional Atmospheric Modeling System (BRAMS 5.2): An Integrated Environment Model Tuned for Tropical Areas**. Geosci. Model Dev. Discuss., doi 10. 2016.

GROPP, W. MPI + MPI: **Using MPI-3 Shared Memory as a Multicore Programming System**. 2016.

GROPP, W. LUSK, E., and SKJELLUM, A. **Using MPI: Portable Parallel Programming with the Message-Passing Interface.** Volume 1, MIT press. 1999.

HOEFLER, T., DINAN, J., BUNTINAS, D., BALAJI, P., BARRET, B. BRIGHTWELL, R., GROPP, W., KALE, V., and THAKUR, R. **MPI + MPI: A New Hybrid Approach to Parallel Programming with MPI plus Shared Memory**. Computing 2013, 95(12):1121-1136.

LOPES, P., P. **Análise de Benefícios do Paralelismo por Comunicação Unilateral em Aplicações com Grades não Estruturadas**. Dissertação de Mestrado, Instituto de Matemática e Estatística, Universidade de São Paulo. 2010.

Message-Passing Interface. Version 3.1. Disponível em: <https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf>. Acesso em: 01 de junho de 2018.

SOUZA, C.R., PANETTA, J., STEPHANY, S. **Desempenho da Comunicação MPI Shared Memory no Modelo Meteorológico BRAMS**. Anais da Escola Regional de Alto Desempenho de São Paulo. São José dos Campos, SP. 2018.