XX ENMC
ENCONTRO NACIONAL DE MODELAGEM COMPUTACIONAL
VIII ECTM
ENCONTRO DE CIÊNCIA E TECNOLOGIA DE MATERIAIS
16 a 19 de Outubro de 2017
Instituto Politécnico - Universidade do Estado de Rio de Janeiro
Nova Friburgo - RJ

ANÁLISE DO DESEMPENHO DE COMUNICAÇÃO USANDO A FUNCIONALIDADE DE MEMÓRIA COMPARTILHADA DO MPI 3.0

Carlos Renato de Souza[1] - carlos.souza@inpe.br
Stephan Stephany[2] - stephan.stephany@inpe.br
Jairo Panetta[3] - jairo.panetta@gmail.com
[1]Centro de Previsão de Tempo e Estudo Climáticos (CPTEC)
Instituto Nacional de Pesquisas Espaciais (INPE), Cachoeira Paulista, SP, Brasil
[2]Laboratório Associado de Computação e Matemática Aplicada (LAC)
Instituto Nacional de Pesquisas Espaciais (INPE),São José dos Campos, SP, Brasil
[3]Divisão de Ciência da Computação (IEC)
Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, SP, Brasil

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 direto 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 utilizando uma máquina paralela Cray. A comunicação unilateral SHM é comparada à comunicação bilateral convencional MPI.*

Keywords: *Shared Memory, Comunicação MPI 3.0, Desempenho paralelo*

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 composta 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 (conhecida como

MPI-1) 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 (1) 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 (3) e (6).

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 estencil 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. As análises feitas em Fortran são importantes para avaliar o desempenho do uso do MPI SHM em aplicações científicas implementadas em Fortran, como por exemplo o modelo numérico de previsão de tempo BRAMS (*Brazilian developments on the Regional Atmospheric Modeling System*) (4).

2. A COMUNICAÇÃO UNILATERAL EM MPI

O escopo deste trabalho é analisar a comunicação unilateral MPI *Shared Memory*, detalhada a seguir.

2.1 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 (5). 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 1 (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.

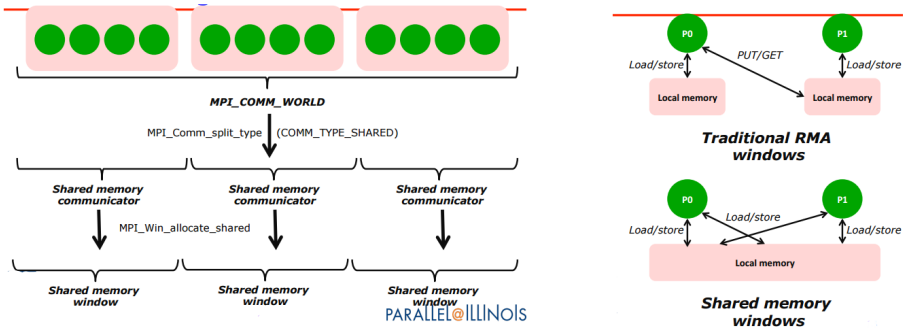

Figura 1- (À esquerda): Divisão do comunicador global em subgrupos de comunicadores específicos de cada nó de memória compartilhada. (À direita): Comparação da comunicação MPI unilateral com janela RMA e com janela SHM. Fonte: (5).

A Figura 1 (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 tem 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 reiões de memória mais próximas aos processadores, afim 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 (1). 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. (7)

3. ESTUDO DE CASO DAS DIFERENÇAS FINITAS

Adotou-se como estudo de caso neste trabalho um programa implementado originalmente por Hoefler e Balaji (2), o qual resolve equações diferenciais parciais (EDP's) pelo método das diferenças finitas numa malha 2D e que requer o cálculo de um estencil 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.

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 (2) 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 $\Delta x = \Delta y = h$. 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 $U(x, y) = U_{i,j}$, $U(x+h, y) = U_{i+1,j}$, finalmente, a aproximação por diferenças finitas aparece na Equação 4 abaixo:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx \frac{U_{i+1,j} + U_{i,j+1} - 4U_{i,j} + U_{i-1,j} + U_{i,j-1}}{h^2} \tag{3}$$

A malha discretizada correspondente ao domínio do problema aparece na Figura 2 (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 estencil 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. 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 2 (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 estencil de 5 pontos.

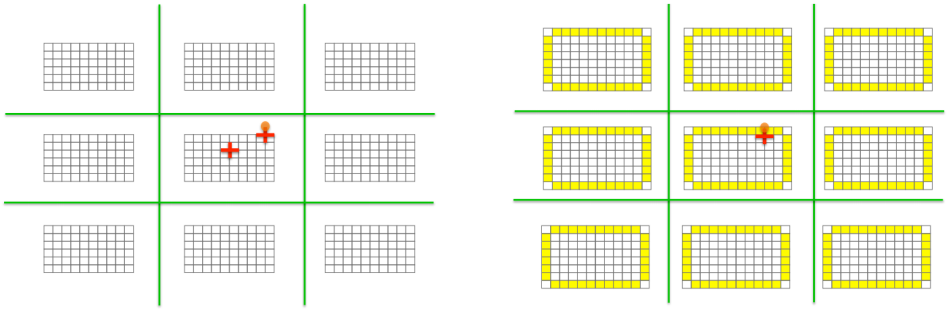

Figura 2- (À esquerda): Malha discretizada original do problema dividida em subdomínios com $b_x \times b_y$ pontos, atribuídos aos processos. (À direita): Malha discretizada original do problema dividida em subdomínios acrescidos das correspondentes *ghost zones* com $(b_x + 2) \times (b_y + 2)$ pontos, atribuídos aos processos. Fonte: (2)

4. TESTES DE DESEMPENHO 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 cores (44 por nó) com cache L1 de 64KB por core, L2 de 256KB por core e L3 compartilhado de 55MB. O ambiente de compilação escolhido foi o conjunto de compiladores PGI, com a biblioteca MPICH.

4.1 Análise dos resultados de desempenho para execução em nó único

Os códigos originais foram implementados e disponibilizados por Hoefler e Balaji (2) 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 4.800×4.800 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 16.10.0 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 3 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. Um fato curioso é que as versões SHM executadas com 1 único processo foram mais rápidas que a própria versão sequencial. A mesma tabela apresenta os correspondentes *Speed Ups* e Eficiências, calculados em relação ao tempo da versão sequencial. Notam-se *Speed Ups* 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.

Tabela 1- Tempos de execução em segundos, Speed Ups 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 (1P), 4 (4P), 9 (9P), 16 (16P), 25 (25P) e 36 processos (36P) num único nó computacional para o domínio de 4800 x 4800 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
Speed Up						
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

Essas versões foram convertidas para Fortran 95 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 (4). Assim, foram também geradas as quatro versões correspondentes àquelas em C (Fortran sequencial, MPI S/R, MPI SHM e MPI SHM-NC) e uma quinta versão Fortran 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 num único nó (média de 3 execuções para cada caso) em função do número de processos, notando-se que as versões Fortran 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. A mesma

tabela apresenta os correspondentes *Speed Ups* e Eficiências, calculados em relação ao tempo da versão sequencial. Nota-se uma degradação do desempenho de todas as versões paralelas Fortran com o aumento do número de processos.

Tabela 2- Tempos de execução em segundos, Speed Ups e Eficiências para as versões em linguagem Fortran MPI S/R, MPI SHM e MPI SHM-NC compiladas com PGI e executadas com 1 (1P), 4 (4P), 9 (9P), 16 (16P), 25 (25P) e 36 processos (36P) num único nó computacional para o domínio de 4800 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
			Speed Up			
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

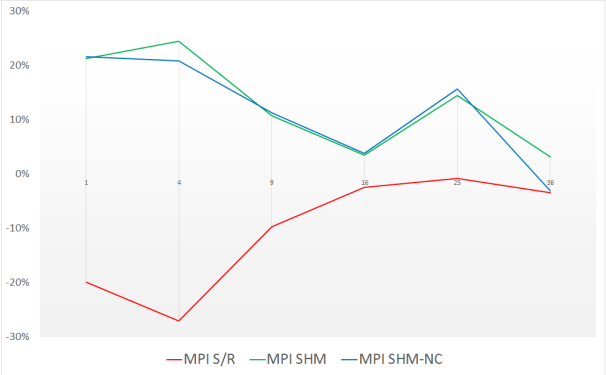

Figura 3- Diferença porcentual das versões Fortran em relação às correspondentes versões em C em função do número de processos (MPI S/R em vermelho, MPI SHM em verde e MPI SHM-NC em azul).

Finalmente, a Figura 3 mostra a diferença percentual do tempo de execução das versões Fortran em relação às correspondentes versões C em função do número de processos, expressa por $((T_f/T_c - 1) \times 100)$. As versões Fortran 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 MPI SHM e SHM-NC apresentam valores positivos dessa diferença porcentual pois têm tempos de execução menores que as correspondentes versões em C.

Em tese, as versões com a comunicação MPI SHM deveriam apresentar melhor desempenho tanto em C quanto em Fortran, 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. Esse fato levou à investigação das diferenças de desempenho entre as versões das duas linguagens. O tempo de execução da

versão Fortran S/R executada com um processo (1P) é muito próximo da versão Fortran sequencial (4% maior), mas as versões Fortran SHM 1P demandam quase 40% mais tempo. Uma simples comparação entre o código das versões MPI SHM C e Fortran mostra que esse tempo adicional deve-se à conversão de ponteiros C para ponteiros Fortran, que obviamente não ocorre na versão C, detalhada a seguir.

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 (`pointer`) utilizando-se a função `c_f_pointer()`, como ilustrado no trecho de programa a seguir para a matriz `avector`. A versão sequencial e MPI S/R em Fortran não usam esse esquema de conversão de ponteiros, apenas as versões SHM usa.

```
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
! 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
```

4.2 Análise dos resultados de desempenho para execução em vários nós

Os códigos referentes ao mesmo problema de diferenças finitas foram executados utilizando, para estes testes, vários nós computacionais da mesma máquina e do mesmo tipo que os empregados na Seção 4.1, com processadores Intel Xeon Broadwell de 2.2 GHz, sendo empregada a versão Fortran S/R com comunicação convencional S/R e a versão Fortran denominada híbrida, que combina as comunicações inter-nó S/R com comunicação intra-nó SHM-NC. O domínio do problema também foi de 4.800×4.800 pontos mas agora com 5.000 iterações no tempo, sendo utilizados até 64 nós computacionais com o número máximo de *cores* por nó (44), num total de 2.304 *cores*. A Tabela 3 mostra os tempos de execução dessas duas versões aumentando-se o número de processos (N1), com o consequente número de processos por nó (N2) e de nós computacionais (N3) sendo esses tempos correspondentes à média de 3 execuções para cada caso. Nessa mesma tabela, aparecem os correspondentes *Speed Ups* e Eficiências, calculados em relação à versão sequencial Fortran. Analogamente à Seção 4.1, foram usadas as opções *default* do compilador PGI, exceto pela opção *-O3*.

Nota-se que, em todos os casos, a versão Fortran MPI S/R foi mais rápida do que a Fortran S/R + SHM-NC, demonstrando que esta última é penalizada seja pela conversão de ponteiros C para Fortran seja pela saturação do uso da memória em cada nó. A eficiência da versão S/R é alta para 4 e 9 processos, decai entre 16 e 100 processos para valores abaixo de 50% e sobe significativamente ao se passar de 100 para 144 processos, provavelmente devido ao tamanho do cache L3 para o problema considerado. A eficiência permanece acima de 80% até 1024 processos, para decair para 1600 e 2304 processos, provavelmente devido à baixa granularidade decorrente para o problema considerado. Em comparação, a versão S/R + SHM-NC teve um

Tabela 3- Tempos de execução (em segundos) e correspondentes Speed Ups e Eficiências das versões Fortran paralelas MPI S/R e MPI S/R + SHM-NC compiladas com PGI e executadas com N1 processos, alocados em N2 cores de N3 nós computacionais para o domínio de 4800 x 4800 pontos com 5.000 passos de tempo.

[N1-N2-N3]		**MPI S/R**		**MPI S/R + SHM-NC**		
Seq.= 333,59s	Tempo (s)	*Speed Up*	Eficiência	Tempo (s)	*Speed Up*	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

desempenho pior para qualquer número de processos, especialmente a partir de 144 processos, em que suas efciências foram menos que a metade daquelas da outra versão.

5. CONCLUSÕES

Testes de desempenho paralelo foram realizados para um código exemplo de diferenças finitas paralelizado com a biblioteca de comunicação por troca de mensagens MPI de forma a 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 à versão com comunicação MPI bilateral convencional na execução utilizando-se um único nó e também na execução em vários nós computacionais de uma máquina Cray, em que a comunicação SHM é empregada intra-nó e a comunicação bilateral inter-nó. O objetivo da comunicação unilateral SHM introduzida no MPI 3.0 foi explorar o ambiente de memória compartilhada, que foi contestado pelos tempos de processamento obtidos pelos resultados mostrados.

Aparentemente, a criação de uma janela de memória compartilhada em Fortran 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 *send-receive*. Isso pode ser demonstrado pela degradação do desempenho relativo entre a comunicação bilateral e a unilateral SHM à medida que se saturam os *cores* disponíveis em cada nó computacional. Naturalmente, esses tempos referem-se aos 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. A comparação dos desempenhos das versões correspondentes C e Fortran, mostrou que a utilização de ponteiros

penaliza a otimização de desempenho pelo compilador, pois o uso de ponteiros C na alocação da janela SHM exige uma conversão para ponteiros Fortran que é custosa. Assim, pode-se inferir que a comunicação unilateral SHM em MPI ainda não foi desenvolvida suficientemente para se melhorar o desempenho paralelo. O interesse desse trabalho é investigar a otimização de modelos numéricos de precisão de tempo e clima, geralmente escritos em Fortran, na execução intra-nó pelo uso da comunicação SHM como alternativa a utilizar uma programação mista MPI-OpenMP explorando assim a paralelização intra-nó por *threads*, mas que requeriria um grande esforço de reprogramação para tornar os modelos existentes *thread-safe*.

Assim, pode-se concluir que o pior desempenho das versões MPI SHM em relação às correspondentes versões S/R deve-se à degradação do desempenho de acesso à memória compartilhada do nó computacional devido a criação da janela de memória compartilhada SHM. Pretende-se investigar isso na continuação, como trabalho futuro.

Referências

[1] MPI Forum. Message-Passing Interface Standard. Version 3.1. (2015), http://mpi-forum.org
[2] Balaji, Pavan and Gropp, William and Hoefler, Torsten and Thakur, Rajeev. (2014), Advanced MPI Programming Tutorial. http://www.mcs.anl.gov
[3] Jeffrey Dobbelaere and Nikos Chrisochoides. (2001), One-Sided Communication Over MPI-1.
[4] Freitas, SR and others. (2016), The Brazilian developments on the Regional Atmospheric Modeling System (BRAMS 5.2): An integrated environmental model tuned for tropical areas. Geosci. Model Dev. Discuss, vol 10.
[5] Gropp and William. (2016), "MPI + MPI: Using MPI-3 Shared Memory as a Multicore Programming System."http://www.caam.rice.edu/m̃k51/presentations/SIAMPP2016_4.pdf.
[6] Gropp, William and Lusk, Ewing and Skjellum, Anthony. (1999), Using MPI: portable parallel programming with the message-passing interface. Parallel Computing, MIT press, vol 1.
[7] Hoefler, Torsten. (2013), MPI + MPI: a new hybrid approach to parallel programming with MPI plus shared memory. Computing, vol 95, no 12, 1121-1136.
[8] Lopes, Pedro Pais. (2010), 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.

ANALYSIS OF COMMUNICATION PERFORMANCE USING MPI 3.0 SHARED MEMORY FEATURES

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 using a Cray parallel computer. SHM unilateral communication is compared to the standard bilateral MPI communication.*

Keywords: *Shared Memory, MPI 3.0 Communication, Parallel Performance*