Algoritmo de Árvore Binomial Balanceada para Broadcast com MPI

João Lucas Cordeiro[1], **Wagner M. Nunan Zola** [1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

joaolucascordeiro@gmail.com wagner@inf.ufpr.br

Resumo. *A operação broadcast do Open MPI é implementada com diferentes algoritmos. De acordo com os parâmetros de cada operação, a biblioteca escolhe automaticamente e executa uma dessas versões. Neste trabalho, apresentamos uma implementação do algoritmo de Árvore Binomial, assim como uma comparação experimental dessa implementação ao broadcast nativo do Open MPI em um cluster com nodos multicore em interconexão gigabit ethernet.*

1. Introdução e Fundamentos

Segundo um estudo [Laguna et al. 2019] feito sobre o uso do MPI em aplicações de alto desempenho, as operações coletivas representam 99% das funções utilizadas. O *Broadcast* é uma operação extremamente importante, sendo a terceira mais utilizada das funções coletivas. Logo, a sua implementação tem bastante impacto no desempenho de um grande número de aplicações. Em [Loch and Koslovski 2021], foram comparados diversos algoritmos coletivos utilizados na operação *allgather*, também implementada pelo MPI, mostrando o desempenho de vários algoritmos com diversos tamanhos de mensagens.

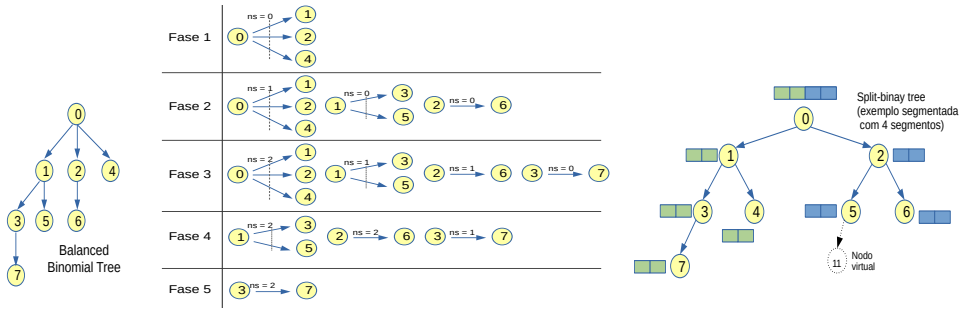
O Open MPI utiliza segmentação de mensagens [Nuriyev and Lastovetsky 2021] nos algoritmos de broadcast, com exceção do algoritmo *Linear Tree*. O trabalho também aborda a implementação do algoritmo *Binomial Tree* (BT) no MPI, que é feita usando uma combinação de *Linear Trees* com *sends* e *receives* não-bloqueantes. O algoritmo é eficiente para mensagens pequenas, mas não é a melhor escolha para tamanhos maiores [Thakur et al. 2005]. A distribuição de mensagens com o algoritmo binomial segmentado pode ser vista na Figura 1(b). Em algumas situações, o MPI opta pelo uso da *Split Binary Tree*. O algoritmo, apresentado na Figura 1(c), consiste em comunicar as mensagens respeitando a topologia de uma árvore binária, mas dividindo a mensagem que será comunicada no broadcast, enviando metade dos segmentos para cada sub-árvore, e fazendo a "troca de metades" entre nodos simétricos na topologia, em uma etapa final.

Neste trabalho, apresentamos os resultados dos experimentos feitos com uma implementação do algoritmo de *Árvore Binomial*, comparando-a ao broadcast nativo do MPI em um cluster com nodos multicore em interconexão gigabit ethernet.

2. Algoritmo de Árvore Binomial Balanceada para Broadcast com MPI

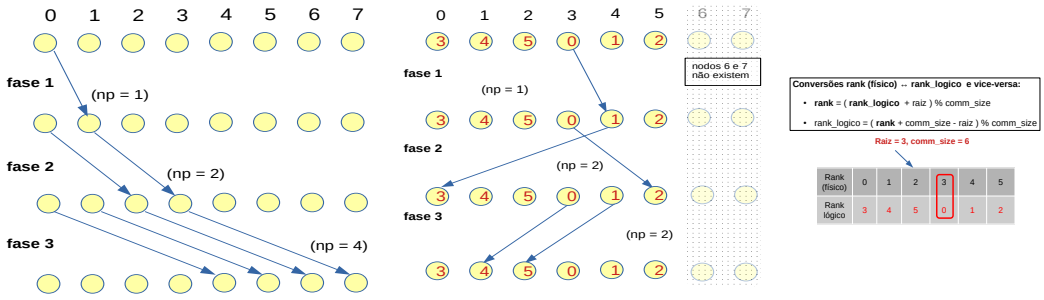
Nossa implementação do algoritmo de Árvore Binomial Balanceada para Broadcast é mostrada na Figura 2(a). A semelhança entre a nossa implementação, chamada de *Simple Balanced Binomial Tree (sBBT)*, e a do MPI está na topologia: a comunicação ocorre entre os mesmos processos. Porém, há diferenças na forma como a comunicação é feita:

a implementação da *Binomial Tree* no MPI utiliza mensagens não-bloqueantes e com segmentação, enquanto na **sBBT**, utilizamos o MPI_Send e o MPI_Recv nativos do MPI, que utilizam mensagens bloqueantes sem segmentação. Na **sBBT**, os nós não-raiz aguardam a chegada da mensagem que está sendo comunicada. Em seguida, cada processo envia as mensagens destinadas aos seus respectivos processos, seguindo a topologia da *Binomial Tree* conforme mostrado na Figura 2(a) para uma árvore completa. O algoritmo deve considerar também o caso de árvores não-completas como mostra o exemplo da Figura 2(b). Para funcionamento da distribuição de mensagens em que o nodo raiz da operação de broadcast for diferente de 0, utilizamos o conceito de *nodos lógicos* e *nodos físicos* com mapeamento mostrado na Figura 2(c).



(a) Topologia (b) Distribuição BT com segmentação (c) Split Binary Tree (balanceada)

Figura 1. Topologia(a) e distribuição(b) na balanced binomial tree. (c) Split-binary tree. O nodo 11 é virtual, seria responsável pela troca com o 7. Nesse caso, o nodo pai assume.



(a) Árvore binomial completa (b) Árvore binomial não-completa (c) mapeamentos

Figura 2. (a) Exemplo de comunicação entre processos numa *Simple Balanced Binomial Tree* completa. (b) Exemplo de comunicação entre processos em uma *Simple Balanced Binomial Tree* não-completa, com os ranks virtuais dos processos ilustrados em vermelho. Nesse exemplo a raiz da operação está no nodo físico 3. (c) Mapeamento dos ranks físicos dos processos pros seus ranks virtuais usados no algoritmo.

3. Metodologia

Os experimentos foram realizados em cluster de processadores Intel Core i5-7500, com 4 núcleos cada e 2 hyperthreads/núcleo, com 8 GiB de RAM e *clock* máximo de 3,8 GHz. A versão 4.1.0 do Open MPI foi utilizada nos experimentos. Foram usados 8 nodos computacionais que dividiam igualmente entre si o número de processos. O desempenho dos dois broadcasts: o nativo do MPI e a nossa implementação com *Binomial Tree* foi comparado. É importante ressaltar que a escolha de algoritmo realizada pelo MPI

depende da função nativa de seleção de algoritmos, que possui como parâmetros o tamanho das mensagens e o número de processos. Para os testes, usamos o MPI_Bcast ou o sBBT_Bcast com os mesmos argumentos, em execuções separadas. Em ambos os casos, após cada broadcast foi realizada uma simulação de trabalho, que consiste em chamar função *nanosleep* com tempo entre 200 μs a 10ms. Essa operação simula a situação normal de intercalações entre operações de comunicação e computação em aplicações. Os experimentos foram executados para mensagens de 8, 1K, 4K e 16KBytes e, para cada tamanho, o *broadcast* foi enviado para 8, 16, 32 e 64 processos (usando opção *oversubscribe* do MPI no último caso), tomando a média de envio de mais de 1250 operações de cada vez, com 10 repetições. Assim a média foi de no mínimo 12500 operações.

4. Resultados e discussão

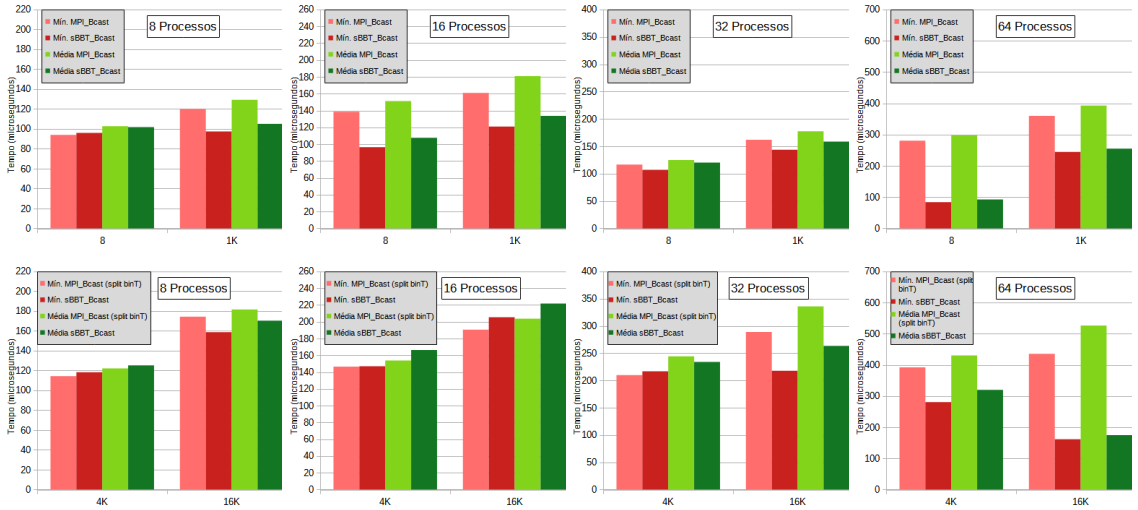


Figura 3. Gráficos ilustram tanto média (verdes) quanto os valores mínimos (vermelhos), do tempo médio de cada broadcast, tanto na **sBBT** (escuras), quanto no MPI (claras).

Conforme pode ser observado na Figura 3, a **sBBT** apresentou desempenho superior ao MPI em grande parte dos experimentos. O que pode justificar a variação no desempenho do MPI é a função de seleção de algoritmos para broadcast. A implementação atual dessa função no Open MPI escolhe seus algoritmos implementados nativamente para executar *broadcat* seguindo o seguinte critério: para as mensagens de tamanho 8Bytes e 1KB, o MPI escolhe a *Binomial Tree*, já para 4KB e 16KB, a *Split Binary Tree* é a opção utilizada. Para mensagens com os tamanhos acima, o número de processos não influencia

(a)

sBBT x MPI Binomial Tree								
Tamanho Mensagem	8 Bytes				1 KBytes			
Processos	8	16	32	64	8	16	32	64
Tempo MPI (μs)	102.68	151.27	124.99	297.73	129.20	180.80	177.33	392.82
Tempo sBBT (μs)	101.74	107.49	120.28	92.56	104.96	133.53	158.73	255.04
Speedup (%)	100.92	140.72	103.90	321.66	123.09	135.40	111.71	154.02

(b)

sBBT x MPI Split Binary Tree								
Tamanho Mensagem	4 KBytes				16 KBytes			
Processos	8	16	32	64	8	16	32	64
Tempo MPI (μs)	122.08	153.86	244.33	430.09	181.55	203.68	335.59	526.37
Tempo sBBT (μs)	125.15	166.31	233.97	319.51	170.20	221.72	263.42	174.54
Speedup (%)	97.54	92.51	104.42	134.61	106.66	91.86	127.39	301.58

Tabela 1. Desempenho da *Simple Balanced Binomial Tree* versus métodos nativos do MPI.

a escolha. Para os experimentos com mensagens menores, em média, a implementação **sBBT** é mais eficiente que a função escolhida pelo Open MPI. Ou seja, pelo menos no contexto do experimento, com mensagens de tamanho 8B e 1KB, em média, a *Binomial Tree* com segmentação e mensagens não-bloqueantes do MPI apresentou desempenho inferior à sem segmentação e com mensagens bloqueantes que implementamos.

Já para as mensagens de tamanho 4KB e 16KB, o MPI usa o algoritmo com *Split Binary Tree* e temos algumas mudanças nos resultados. Para 8 processos, o MPI foi mais eficiente para mensagens de 4KB, mas para mensagens de 16KB, a diferença foi pequena. Já para 16 processos, o MPI acabou se mostrando melhor em ambos os casos. Apesar desses resultados, a **sBBT** se apresentou mais rápida para 32 e, principalmente, para 64 processos. O que pode justificar esse alto desempenho das execuções com 64 processos, bem como o melhor desempenho em outras configurações, é o uso do *Eager Protocol* pelo MPI quando há mensagens menores que 64 KBytes. Para aumentar a eficiência desse protocolo, o MPI pode unir múltiplas mensagens para enviá-las em um único pacote de dados, evitando uma sobrecarga de comunicação, aumentando a eficiência.

5. Conclusões

Os resultados desses experimentos indicam que, para mensagens de tamanho pequeno (8Bytes e 1KBytes), pelo menos com máquinas com as mesmas propriedades das utilizadas nos experimentos, a implementação da *Binomial Tree* com mensagens segmentadas e não-bloqueantes não é uma boa opção se comparada à implementação da **sBBT** com mensagens completas e bloqueantes. Já para as mensagens maiores (4KBytes e 16KBytes), mudar o algoritmo para a *Split Binary Tree* não parece ser uma boa opção na maioria dos casos, principalmente com 64 processos, onde a diferença de tempo médio por mensagem é significativa.

Segundo [Nuriyev and Lastovetsky 2021], embora a função de seleção de algoritmos do MPI seja muito eficiente, ela não pode garantir a melhor opção para todas as situações. Uma maior análise de diversas situações diferentes seria interessante para que seus resultados fossem usados para criar uma nova função mais adequada para a escolha de algoritmos de comunicação, provavelmente com mais parâmetros para chegar em conclusões mais precisas, mas simples o suficiente para ainda ser eficiente.

Referências

Laguna, I., Marshall, R., Mohror, K., Ruefenacht, M., Skjellum, A., and Sultana, N. (2019). A large-scale study of MPI usage in open-source HPC applications. In *Proceedings of the Int. Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA. Association for Computing Machinery.

Loch, W. and Koslovski, G. (2021). Comparação experimental dos algoritmos coletivos para o MPI_Allgather no OpenMPI. In *Anais da XXI Escola Regional de Alto Desempenho da Região Sul*, Porto Alegre, RS, Brasil. SBC.

Nuriyev, E. and Lastovetsky, A. (2021). Efficient and accurate selection of optimal collective communication algorithms using analytical performance modeling. *IEEE Access*, 9:109355–109373.

Thakur, R., Rabenseifner, R., and Gropp, W. (2005). Optimization of collective communication operations in MPICH. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66.