CI1316 1º semestre 2023 Lucas Müller GRR20197160 Tiago Serique Valadares GRR20195138

Trabalho 2

Introdução

Neste trabalho, implementamos um algoritmo de broadcast em MPI e o comparamos com a versão utilizada no MPI nativo. O objetivo foi testar o desempenho do algoritmo em diferentes tamanhos de mensagens, sendo eles 4KB e 16KB. Optamos pela utilização da Split Binary Tree como topologia interna do MPI, devido à sua eficiência nesses casos.

Além de distribuir as mensagens de acordo com a nossa nova topologia de árvore, implementamos as etapas de "quebra de mensagens" em metades, seguindo o procedimento da Split Binary Tree, e a "troca de metades" na etapa final.

Em cada etapa do algoritmo, cada nó recebeu uma metade da mensagem, e na etapa final ocorreu a troca de metades entre os nós.

Implementação

A função `my_Bcast_rb()` implementa um algoritmo de broadcast em MPI (Message Passing Interface) usando uma topologia de árvore binária dividida. O objetivo é distribuir uma mensagem a partir de um nodo raiz para todos os outros nodos em um comunicador MPI.

A função recebe os seguintes parâmetros:

- `buffer`: Um ponteiro para o buffer contendo a mensagem a ser transmitida.
- `count`: O número de elementos no buffer.
- `datatype`: O tipo de dado dos elementos no buffer.
- `root`: O rank do nodo raiz que irá enviar a mensagem.
- `comm`: O comunicador MPI que define o grupo de nodos envolvidos na comunicação.

A implementação do algoritmo é baseada na ideia de que cada nodo recebe uma metade da mensagem em cada etapa e, na etapa final, ocorre a troca das metades entre os nodos.

A função começa obtendo o rank físico do nodo atual no comunicador MPI e o número total de nodos no comunicador. Em seguida, é calculado o rank lógico do nodo em relação ao nodo raiz.

Se o nodo não for o nodo raiz, ele espera para receber sua metade da mensagem. O nodo fonte de onde ele deve receber a mensagem é calculado com base no rank lógico. Dependendo do valor do rank lógico, o nodo recebe a metade esquerda ou direita do buffer.

Em seguida, ocorrem as etapas de envio das metades do buffer. O número de etapas é determinado pelo logaritmo base 2 do número total de nodos. O nodo verifica se ele está na posição correta para enviar sua metade do buffer na etapa atual. Se sim, ele calcula o destino lógico e físico do envio com base no rank lógico e no nodo raiz. O nodo utiliza a função `MPI_Send` para enviar sua metade do buffer para o destino correto. O nodo raiz e os nodos com ranks maiores do que o último nodo par não enviam suas metades.

Após as etapas de envio, ocorre a etapa final de completar a outra metade do buffer a partir do nodo vizinho que possui a outra metade. Se o nodo atual for par, ele envia sua metade para o nodo vizinho e recebe a outra metade desse mesmo nodo vizinho. Se o nodo atual for ímpar, ele recebe a metade do buffer do nodo vizinho e envia sua metade de volta. Essa etapa final também lida com o último nodo ímpar, que envia sua metade para o próximo nodo. O último nodo par não executa nenhuma ação nessa etapa.

No final da função, o broadcast é concluído e a mensagem é transmitida de forma eficiente para todos os nodos no comunicador MPI, seguindo a estratégia da árvore binária dividida.

Experimento

O experimento foi realizado em um cluster Xeon com as seguintes especificações:

- 18 nós Xeon;
- Cada nó possui 2 processadores;
- Cada processador possui 4 núcleos;
- O Xeon utilizado não possui suporte para hyperthreads;
- Portanto, cada nó pode executar até 8 processos MPI sem oversubscribe.

O objetivo do experimento foi avaliar o desempenho do algoritmo de broadcast implementado em comparação com a versão nativa do MPI, levando em consideração as restrições de hardware do cluster Xeon utilizado.

Tempo médio obtido

`my_Bcast_rb()`

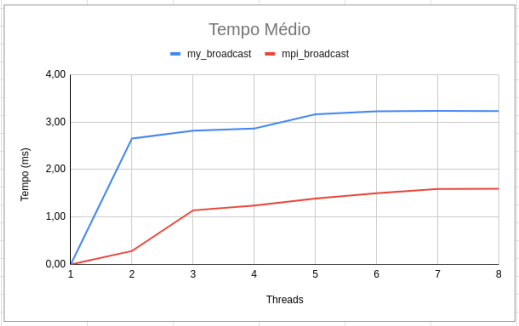
Threads	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Tempo médio (ms)	0.000171	2.588983	2.950064	3.041066	3.163413	3.249556	3.273740	3.213406
	0.000170	2.501962	2.628567	2.810101	3.152823	3.220351	3.243614	3.224082
	0.000156	2.855325	2.974200	2.913736	3.260141	3.207943	3.224020	3.217176
	0.000156	2.795320	2.673676	2.777451	3.157442	3.235442	3.244239	3.290740
	0.000154	2.605714	2.913419	3.011259	3.159491	3.197485	3.149234	3.221483
	0.000162	2.637307	2.717848	2.753401	3.069298	3.239960	3.256511	3.250986
	0.000165	2.770206	2.926418	2.986105	3.202553	3.228064	3.218138	3.297038
	0.000149	2.847580	2.710117	2.790916	3.218410	3.037839	3.223629	3.221423
	0.000155	2.486494	2.953607	3.012823	3.161155	3.231095	3.260629	3.254505
	0.000169	2.665114	2.709273	2.814479	3.161303	3.207926	3.219765	3.231530

`MPI_Bcast()`

Threads	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Tempo médio (ms)	0.000114	0.284021	1.189112	1.322329	1.409829	1.492847	1.611227	1.600242
	0.000117	0.284034	1.090372	1.200937	1.387592	1.501886	1.589668	1.585760
	0.000109	0.283995	1.179591	1.271387	1.391444	1.498855	1.662976	1.597653
	0.000118	0.284112	1.116883	1.226587	1.386153	1.555950	1.582545	1.581696
	0.000119	0.283987	1.176447	1.198152	1.409364	1.495353	1.590314	1.596153
	0.000136	0.283931	1.094215	1.292278	1.379637	1.501055	1.592416	1.593598
	0.000118	0.283989	1.178155	1.193171	1.387421	1.509790	1.618458	1.593960
	0.000114	0.321689	1.100281	1.256143	1.418303	1.485037	1.583206	1.601484
	0.000109	0.283993	1.162111	1.220941	1.392104	1.481118	1.607448	1.586992
	0.000125	0.284022	1.109053	1.370530	1.387890	1.512626	1.591672	1.605078

`my_Bcast_rb()` **x** `MPI_Bcast()`

	tempo medio							
threads	1	2	3	4	5	6	7	8
my_broadcast	0,000159	2,651211	2,815634	2,864108	3,161229	3,224208	3,233817	3,227806
mpi_broadcast	0,000118	0,284008	1,139497	1,241365	1,389667	1,499955	1,592044	1,595057



Vazão média obtida

`myBcast_rb()`

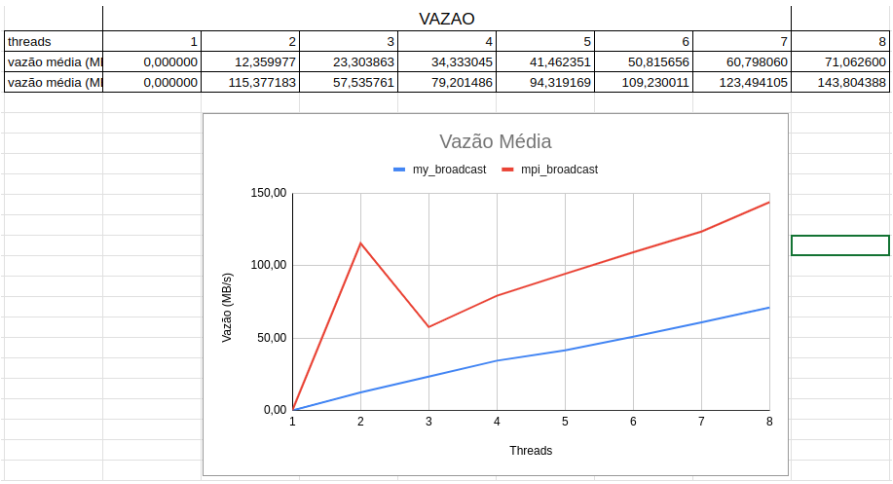
Threads	1	2	3	4	5	6	7	8
Vazão (MB/s)	0.000000	12.656705	22.215110	32.325509	41.433734	50.419201	60.056082	71.380965
	0.000000	13.096921	24.932215	34.982367	41.572902	50.876437	60.613865	71.144584
	0.000000	11.476103	22.034834	33.738132	40.204390	51.073230	60.982255	71.297315
	0.000000	11.722452	24.511574	35.393600	41.512091	50.639140	60.602197	69.703469
	0.000000	12.575439	22.494533	32.645485	41.485160	51.240279	62.430417	71.201990
	0.000000	12.424795	24.113193	35.702760	42.704221	50.568531	60.373815	70.555817
	0.000000	11.828724	22.394616	32.920476	40.927349	50.754875	61.093710	69.570322
	0.000000	11.507313	24.181979	35.222846	40.725701	53.933074	60.989641	71.203309
	0.000000	13.178394	22.188465	32.628534	41.463321	50.707266	60.297571	70.479546
	0.000000	12.295159	24.189514	34.927957	41.461380	51.073500	61.062842	70.980616

`MPI_Bcast()`

Threads	1	2	3	4	5	6	7	8
Vazão (MB/s)	0.000000	115.371857	55.113374	74.341561	92.970150	109.750026	122.023809	143.338298
	0.000000	115.366318	60.104284	81.856066	94.460070	109.089508	123.678660	144.647346
	0.000000	115.382508	55.558224	77.320258	94.198546	109.310123	118.226636	143.570591
	0.000000	115.334669	58.677576	80.144342	94.558103	105.299016	124.235360	145.019052
	0.000000	115.385714	55.706716	82.046376	93.000810	109.566071	123.628418	143.705518
	0.000000	115.408368	59.893160	76.070335	95.004727	109.149899	123.465243	143.935895
	0.000000	115.384817	55.625958	82.388881	94.471667	108.518391	121.478610	143.903257
	0.000000	101.862373	59.562990	78.258629	92.414641	110.327200	124.183422	143.227173
	0.000000	115.383169	56.393946	80.514930	94.153863	110.619154	122.310661	144.535042

Threads	1	2	3	4	5	6	7	8
	0.000000	115.371520	59.091833	71.727018	94.439792	108.314968	123.522967	142.906414

`my_Bcast_rb()` x `MPI_Bcast()`



Após a comparação dos gráficos da nossa implementação `my_Bcast_rb()` com a implementação original `MPI_Bcast()`, é possível perceber algumas diferenças.

Primeiramente, em relação ao tempo médio, observamos que a nossa implementação `my_Bcast_rb()` apresentou tempos mais altos em comparação com a implementação original `MPI_Bcast()`. Isso ocorre devido às etapas adicionais que foram inseridas no algoritmo para distribuir e trocar as metades da mensagem entre os nós. Essas etapas adicionam um custo extra de comunicação e processamento, o que resulta em um aumento no tempo necessário para completar o broadcast.

Em termos de vazão média, nossa implementação `my_Bcast_rb()` também apresentou valores mais baixos em comparação com a implementação original `MPI_Bcast()`. A vazão é influenciada pelo tempo de transmissão da mensagem e pelo tamanho da mensagem. Como nossa implementação adiciona etapas extras de comunicação e troca de metades, a vazão acaba sendo reduzida.

Essas diferenças podem ser explicadas pelo fato de que a nossa implementação prioriza a utilização da Split Binary Tree como topologia interna do MPI, o que implica em um maior número de etapas de comunicação e troca de metades. Embora essa topologia seja eficiente em casos específicos, como tamanhos de mensagem maiores, ela pode não ser a melhor escolha em termos de desempenho geral.

Portanto, podemos observar que nossa abordagem introduz um aumento no tempo médio e uma redução na vazão média do broadcast.