Computação de Alto Desempenho

2022/2023

Relatório 2º Projeto

Elaborado por:

Diogo Spencer 55481

Ruben Belo 55967

Docentes:

Vitor Duarte

Dezembro 20, 2022

Melhor solução

Ao longo deste projeto avaliamos 3 diferentes soluções distintas, sendo a primeira versão a paralelização da computação da difusão de calor usando a função MPI_Sendrecv para podermos enviar dados com tamanho elevado que é algo que não acontece se usarmos as funções MPI_Send e MPI_Recv, na segunda versão implementada fizemos uso das funções MPI_ISend, MPI_IRecv e MPI_Waitall para transferir linhas ou partes da matriz entre processos. Ambas estas soluções foram implementadas de maneira a que todos os cpu tivessem uma parte da matriz para computar e quando fosse necessário escrever a matriz num ficheiro todos os workers enviavam a sua parte da matriz para o worker nr° 0 e este sim procedia a escrita da matriz, a nossa terceira e última solução implementada foi uma variação da segunda versão em que um worker tem apenas o papel de escrever a matriz, funcionando como um gather para outros workers.

Para avaliar as diferentes soluções corremos os programas com diferentes configurações fazendo variar o tamanho da grid fazendo assim variar o número de elementos transferidos em cada step entre os diversos workers, variamos também o número de steps e os steps necessários para que o nosso programa produza um ficheiro.

Detalhes relevantes da implementação

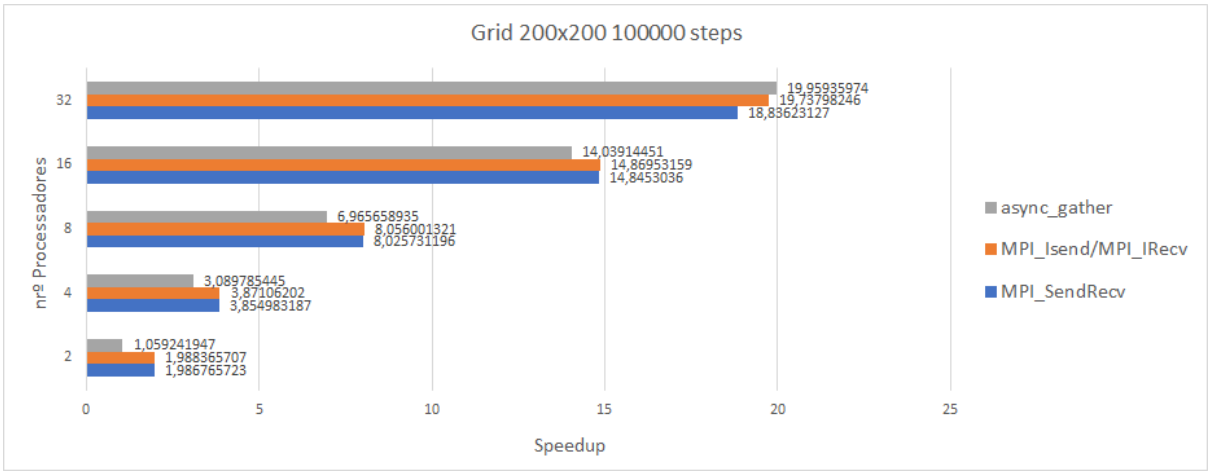
Inicialmente o programa era constituído por dois tipos de código, um dedicado ao master e outro dedicado aos workers, o master dividia o trabalho depois enquanto tratava da sua parte da matriz também recolhia os dados dos workers para depois escrever no ficheiro. Nesta versão inicial o programa usava as funções MPI_Send e MPI_Recv.

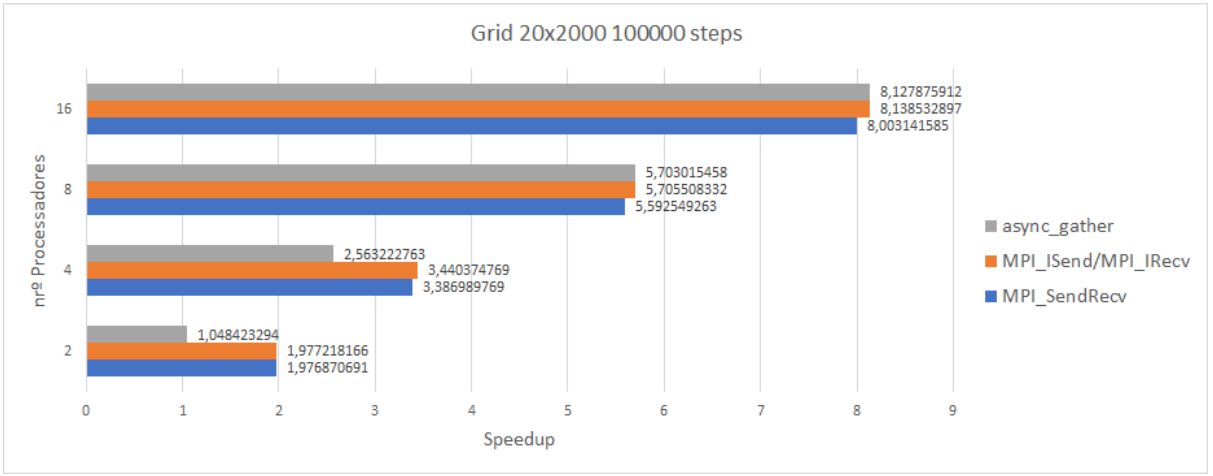
Posteriormente depois de conversar com o Professor abandonamos esta ideia e decidimos usar MPI_Sendrecv para substituir o uso das outras duas funções pois para um elevado número de dados poderia provocar um deadlock e usar um modelo em que cada worker calcula que parte da matriz ele vai computar e o worker nº0 é responsável por recolher todos os dados e escrever o ficheiro.

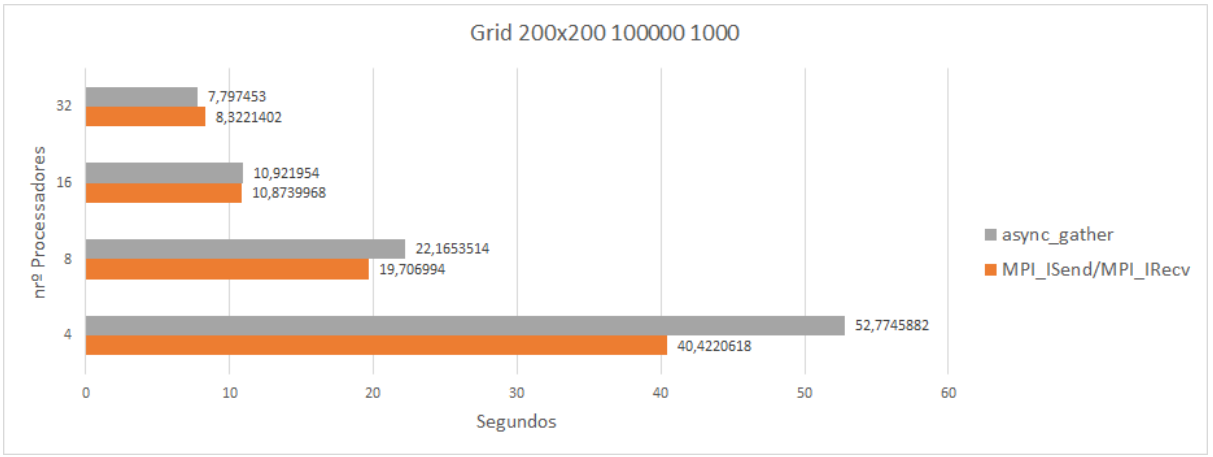
Na versão onde os dados são transferidos de modo assíncrono bastou trocar a função MPI_Sendrecv por MPI_ISend, MPI_IRecv e adicionar as funções necessárias para assegurar a correctness do programa como por exemplo MPI_Waitall.

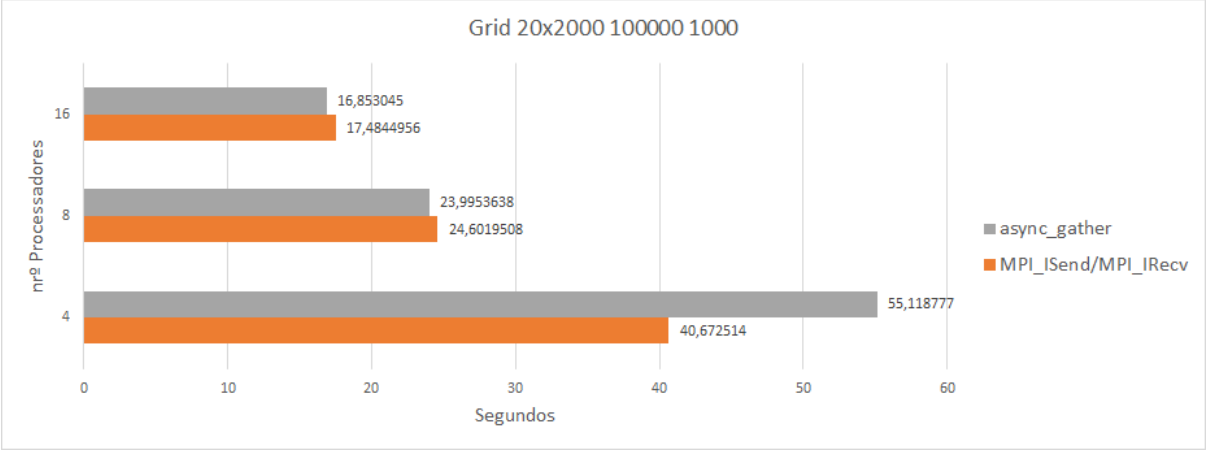
Para a última versão alteramos o código anterior para que um worker apenas fizesse a recolha dos dados e a sua escrita no ficheiro o worker escolhido foi o worker com o maior Id.

Resultados


Grid 200x200 100000 steps
nº Processadores
32
16
8
4
2
19,95935974
19,73798246
18,83623127
14,03914451
14,86953159
14,8453036
6,965658935
8,056001321
8,025731196
3,089785445
3,87106202
3,854983187
1,059241947
1,988365707
1,986765723
0
5
10
15
20
25
Speedup
async_gather
MPI_Isend/MPI_IRecv
MPI_SendRecv


Grid 20x2000 100000 steps
nº Processadores
16
8
4
2
8,127875912
8,138532897
8,003141585
5,703015458
5,705508332
5,592549263
2,563222763
3,440374769
3,386989769
1,048423294
1,977218166
1,976870691
0
1
2
3
4
5
6
7
8
9
Speedup
async_gather
MPI_ISend/MPI_IRecv
MPI_SendRecv


Grid 200x200 100000 1000
nº Processadores
32
16
8
4
7,797453
8,3221402
10,921954
10,8739968
22,1653514
19,706994
52,7745882
40,4220618
0
10
20
30
40
50
60
Segundos
async_gather
MPI_ISend/MPI_IRecv



configuração grid/steps	MPI_ISend/MPI_IRecv	Async gather
32 CPUs 200x200 1000000 1000	81,5659728 seg	75,8151044 seg
32 CPUs 300x300 100000 1000	15,4591676 seg	15,1401686 seg
32 CPUs 400x400 100000 1000	26,6797094 seg	25,5417474 seg
32 CPUs 500x500 100000 1000	42,12691 seg	39,0764256 seg
32 CPUs 750x750 10000 1000	28,4241538 seg	28,9137962 seg
32 CPUs 1000x1000 10000 1000	73,5071732 seg	74,8102598 seg

Análise dos resultados

Com os resultados apresentados acima concluímos que a melhor implementação difere da configuração da grid ,dos steps e do número de CPUs usados para correr o programa, sem saber qual seria a configuração usada para correr o programa recomendaria usar a implementação nº 2 pois foi a que obteve um resultado mais consistente de todas as implementações testadas.

Mas sabendo qual a configuração que vai correr no programa podemos obter ganhos na performance ao utilizar um worker apenas para recolher os dados e escrevê-los no ficheiro(implementação nº3), pois a partir dos resultados observados em cima esta solução funciona bem quando o trabalho por thread é menor. Isso é bastante visível na última tabela quando aumentamos muito o trabalho que cada thread tem que fazer a performance tende a diminuir.