Exercício 3: Paralelização do Jogo da Vida com MPI

Nome: Daniel Araujo Cavassani (COMP 25)

Data: 21/10/2024

Neste exercício, o objetivo foi paralelizar o Jogo da Vida utilizando MPI (Message Passing Interface) para execução no supercomputador Santos Dumont. Diferentemente do exercício anterior, que utilizava OpenMP, aqui o foco foi distribuir o trabalho entre múltiplos nós do cluster, cada um executando processos MPI que colaboram para calcular o avanço das gerações do tabuleiro.

Modificação no Código

O principal trabalho consistiu em implementar a troca de informações entre processos para atualizar as "ghost zones" após cada iteração, o que é essencial para garantir a continuidade do cálculo.

A função modificada foi UmaVida, onde a troca de linhas das bordas entre processos MPI vizinhos foi implementada utilizando as chamadas MPI_Send e MPI_Recv. Aqui está o trecho relevante do código adaptado:

```
void UmaVida(int* restrict tabulIn, int* restrict tabulOut, const int tam,
             const int tamLocal, const int procAntes, const int procDepois)
{

 MPI_Status status;

 // Enviar e receber as linhas para atualizar as ghost zones
 // Envia a primeira linha interna para o processo anterior e recebe a
última linha interna do processo anterior
 if (procAntes != MPI_PROC_NULL) {
   MPI_Sendrecv(&tabulIn[ind2d(1,0)], tam+2, MPI_INT, procAntes, 0,
                &tabulIn[ind2d(0,0)], tam+2, MPI_INT, procAntes, 0,
                MPI_COMM_WORLD, &status);
 }
```

```
 // Envia a última linha interna para o processo posterior e recebe a
primeira linha interna do processo posterior
 if (procDepois != MPI_PROC_NULL) {
   MPI_Sendrecv(&tabulIn[ind2d(tamLocal,0)], tam+2, MPI_INT, procDepois,
0,
                &tabulIn[ind2d(tamLocal+1,0)], tam+2, MPI_INT, procDepois,
0,
                MPI_COMM_WORLD, &status);
 }

 // Atualiza o tabuleiro local
 for (int i=1; i<=tamLocal; i++) {
   for (int j=1; j<=tam; j++) {
     const int vizviv =
       tabulIn[ind2d(i-1,j-1)] +
       tabulIn[ind2d(i-1,j  )] +
       tabulIn[ind2d(i-1,j+1)] +
       tabulIn[ind2d(i  ,j-1)] +
       tabulIn[ind2d(i  ,j+1)] +
       tabulIn[ind2d(i+1,j-1)] +
       tabulIn[ind2d(i+1,j  )] +
       tabulIn[ind2d(i+1,j+1)];
     if (tabulIn[ind2d(i,j)] && vizviv < 2)
       tabulOut[ind2d(i,j)] = 0;
     else if (tabulIn[ind2d(i,j)] && vizviv > 3)
       tabulOut[ind2d(i,j)] = 0;
     else if (!tabulIn[ind2d(i,j)] && vizviv == 3)
       tabulOut[ind2d(i,j)] = 1;
     else
       tabulOut[ind2d(i,j)] = tabulIn[ind2d(i,j)];
   }
 }
}
```

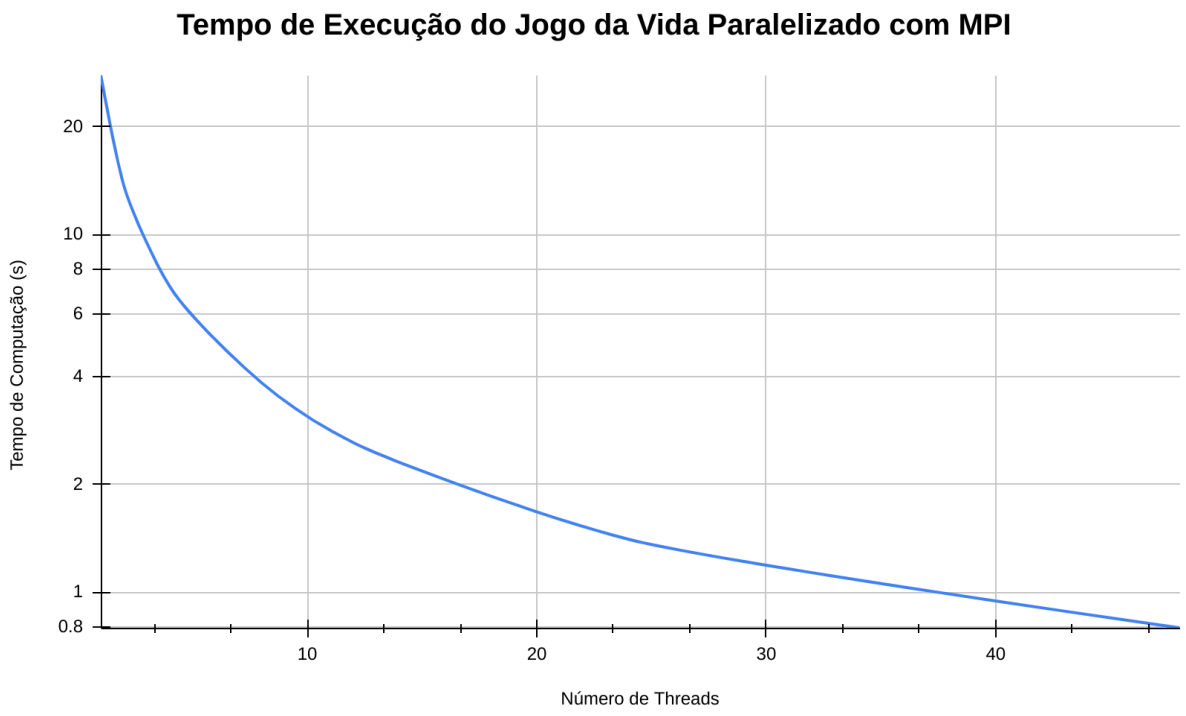
Com essa modificação, o programa agora é capaz de rodar corretamente para múltiplos processos MPI.

Resultados

Executei o programa para um tabuleiro fixo de tamanho 2048x2048 com diferentes números de processos MPI (1, 2, 4, 8, 12, 24 e 48). A tabela abaixo apresenta os tempos de execução para cada configuração:

Número de Threads	2048x2048
1	27.866859
2	13.746998
4	7.161992
8	3.853004
12	2.618369
24	1.404845
48	0.794787

Este gráfico mostra a redução no tempo de execução à medida que o número de processos aumenta:



Cálculo do Speed-up

O speed-up foi calculado dividindo o tempo de execução com 1 processo pelo tempo de execução com múltiplos processos. Os resultados foram:

Número de Threads	2048x2048
1	1.00
2	2.03
4	3.89
8	7.23
12	10.64
24	19.84
48	35.06

Análise

Os resultados mostram que a paralelização com MPI foi eficiente, com tempos de execução reduzidos proporcionalmente ao número de processos. O speed-up foi significativo, especialmente com até 24 processos, mas apresenta eficiência decrescente à medida que o número de processos aumenta.

Isso ocorre porque o overhead de comunicação e o custo de sincronização entre os processos começam a dominar, reduzindo o ganho do paralelismo. No caso de 48 processos, o speed-up foi 35.06, indicando uma boa eficiência geral, mas longe do ideal esperado (48.0). Essa diferença é explicada pelo aumento do custo de troca de mensagens e pela sobrecarga inerente do MPI em configurações de alta concorrência.

Conclusão

A paralelização com MPI mostrou-se eficiente para o Jogo da Vida em um tabuleiro de grande dimensão (2048x2048), especialmente com até 24 processos. O tempo de execução reduziu significativamente, e o speed-up alcançado foi expressivo, embora sublinear devido ao overhead de comunicação e sincronização entre processos.

Para configurações maiores, como 48 processos, o ganho adicional é menor, o que destaca a importância de balancear o número de processos com o tamanho do problema para otimizar a eficiência. Em relação ao OpenMP, o MPI é mais adequado para problemas distribuídos entre nós, enquanto o OpenMP é ideal para paralelismo em um único nó.