ENGENHARIA AEROESPACIAL E MECÂNICA
MECATRÔNICA
CE-265 PROCESSAMENTO PARALELO

Docente: Jairo Panetta
Discente: Lucas Kriesel Sperotto

Exercício 5 – RELATÓRIO

Paralelização por MPI do "Jogo da Vida" no Sistema "CROW"

Introdução:

Este relatório esta dividido em quatro partes distintas: procedimento, resultados "1D", resultados "2D" e anexos em arquivo anexado na submissão.

No tópico procedimento demostrarei o "passo a passo" para a realização do trabalho, bem como o raciocínio que me levou a deduzir a maneira com o qual paralelizei o código, por fim, serão apresentados os resultados de tempo e desempenho (speed-up e eficiência) para cada abordagem, decomposição unidimensional e bidimensional de domínios bidimensionais, com suas interpretações.

Os anexos se encontram em dois arquivos ".tgz" anexados na submissão do trabalho, o primeiro conterá arquivos referentes à decomposição unidimensional e o segundo à decomposição bidimensional. Nos arquivos anexados se encontrarão os códigos fonte paralelizados, o Makefile, os Script's de execução e os arquivos de retorno dos resultados.

Procedimento:

Como proposto no trabalho, os testes deveriam ser efetuados no sistema "Crow". Poderíamos também optar pela linguagem de programação para a implementação, junto com o aluno Marcio Moreira, convertemos o código Fortran para C, de forma a manter a mesma funcionalidade dos executáveis e das funções implementadas.

Após a conversão do código, foi testado e assegurado o funcionamento, o próximo passo foi adequar o Makefile e os Script's de execução. No Makefile foi especificado o compilador e os executáveis. Nos Script's foi adicionado o nome do arquivo executável e alterado parâmetros para MPI.

Esta paralelização foi muito mais trabalhosa que as anteriores, começamos pela decomposição unidimensional do domínio, para isso era necessário dividir as linhas da matriz entre os processos,

para isso foi criada a função “Divide()” junto ao arquivo “ModVida”, esta função é executada por todos os processos e retorna um vetor com os valores do tamanho de cada processo, isso ajuda pois os processos “sabem” o tamanho do domínio dos demais.

A próxima etapa consiste em alocar dinamicamente as matrizes com os tamanhos calculados mais a zona fantasma. A função “InitTabuleiro()” também é chamada por todos os processos, porém sua execução é de tal forma que somente o processo “0” inicia um veleiro, isso acarreta no fato de não podermos ter tabuleiros menores que “32” para 8 processos. Tentei implementar essa função de forma que ela mandasse uma mensagem para os demais processos com o número de linhas que cada um deveria inicias e qual posição do veleiro caberia a ele, porém não tive tempo de terminar. Se a função que eu pretendia implementar funcionasse seria possível ter tabuleiros de 6 linhas para até 6 processos.

Uma outra tentativa foi iniciar globalmente os tabuleiros e dividi-los entre os processos, mas não terminei por achar que não era o objetivo da tarefa. A função começou a ser implementada, porém não está completa. Por esse motivo, no arquivo se tem duas funções “InitTabuleiro()” uma local (que está sendo usada de fato) e outra global que não está em uso e nem funcional.

A função “UmaVida()” precisou receber o numero de linhas e coluna separadamente o restante se manteve inalterado.

O seguinte passo foi a troca de mensagens entre os processos para atualizar as zonas fantasmas, para isso bastava que todos os processos menos o ultimo mandasse sua ultima linha “útil” para o próximo e todos os processos menos o primeiro recebessem em sua zona fantasma “superior”, logo após todos os processos menos o primeiro mandam sua primeira linha “útil” e todos os processos menos o ultimo recebem em sua zona fantasma “inferior”.

Para a função “correto()”, ela verifica a correção para cada processo (todos com zero casas vivas) e para o ultimo verifica o número de casas vivas e a posição do veleiro. Para que o mestre componha a correção, todos os escravos mandam por mensagem ao mestre o resultado de suas correções, e o mestre imprime o tempo e o resultado.

Para o executável “funcionamento” tivemos que alterar a função “DumpTabuleiro()” de modo que todos os processos a chamem mas apenas o mestre imprima em tela o tabuleiro, para compor o tabuleiro os escravos enviam para o mestre seus tabuleiros posição por posição e o mestre imprime cada célula, sei que isto é inconvenientemente lento, porém preferi por não empacotar cada trecho e também por não compor um tabuleiro global por falta de tempo para a codificação.

Com tudo isso pronto partiu-se para a divisão bidimensional do domínio, mais uma vez tentando utilizar a função “MPI_Dims_create()”. Mas preferi utilizar a função implementada “Divide()” com alguns ajustes nos dados do vetor recebido.

Verificado que alguns casos de divisão seriam análogos a divisão unidimensional, e nos casos de 4, 6 e 8 processos o domínio poderia ser dividido de forma bidimensional. Foi duplicado o código com um “if” de controle para que os casos análogos executem de forma unidimensional e casos que podem ser divididos de forma bidimensional recebessem tratamento especial.

A divisão bidimensional deu mais trabalho, pois teríamos que enviar as colunas das matrizes, para as zonas fantasmas dos demais processos. Para esta divisão a função “InitTabuleiro()” permaneceu inalterada, bastando que o veleiro fosse iniciado no processo “0”. A função “ModVida()” também ficou inalterada pois ela já recebia o tamanho de cada parte para iterar uma vida.

A troca de mensagens foi mais trabalhosa, primeiro troca-se as colunas correspondentes, para depois trocar as linhas e elementos “diagonais” de forma análoga a divisão unidimensional.

A função “Correto()” não precisou de modificações, pois a verificação do tabuleiro foi análoga a divisão unidimensional. Um tratamento especial teve que ser dado a função “DumpTabuleiro()”, dessa forma temos duas funções no arquivo “ModVida”, uma para “1D” e outra para “2D”. A função “DumpTabuleiro()”, compõe em seu interior um tabuleiro completo e recebe de todos os processos as partes correspondentes e as coloca nas suas respectivas posições, para só então imprimir o tabuleiro.

Resultados Decomposição Unidimensional

Na tabela 1 estão mostrados os tempos de execução para decomposição unidimensional do domínio para diferentes números de processos e tamanhos de tabuleiros, os Speed-Up calculados se encontram na tabela 2 e as eficiências da paralelização em função do número de processos na tabela 3.

Tabela 1 – Resultados de Tempo – 1D

Tamanho Tabuleiro	Tempo 1 Processo	Tempo 2 Processos	Tempo 3 Processos	Tempo 4 Processos	Tempo 5 Processos	Tempo 6 Processos	Tempo 7 Processos	Tempo 8 Processos
32	**0,000961**	**0,000751**	**0,000688**	**0,000615**	**0,000629**	**0,000590**	**0,000598**	**0,000819**
64	**0,007934**	**0,004372**	**0,003174**	**0,002532**	**0,002200**	**0,001968**	**0,001736**	**0,001713**
128	**0,065094**	**0,033440**	**0,023154**	**0,017578**	**0,014572**	**0,012574**	**0,011030**	**0,009840**
256	**0,525322**	**0,265617**	**0,179618**	**0,135303**	**0,110150**	**0,092467**	**0,080182**	**0,070199**
512	**4,258558**	**2,139449**	**1,434733**	**1,068178**	**0,859554**	**0,720224**	**0,619532**	**0,541407**
1.024	**35,400059**	**17,774611**	**11,804255**	**8,652198**	**6,943850**	**5,736874**	**4,929206**	**4,301612**

Tabela 2 – Speed-up Calculados – 1D

Tamanho Tabuleiro	2 Processos	3 Processos	4 Processos	5 Processos	6 Processos	7 Processos	8 Processos
32	**1,280**	**1,397**	**1,563**	**1,528**	**1,629**	**1,607**	**1,173**
64	**1,815**	**2,500**	**3,133**	**3,606**	**4,032**	**4,570**	**4,632**
128	**1,947**	**2,811**	**3,703**	**4,467**	**5,177**	**5,902**	**6,615**
256	**1,978**	**2,925**	**3,883**	**4,769**	**5,681**	**6,552**	**7,483**
512	**1,990**	**2,968**	**3,987**	**4,954**	**5,913**	**6,874**	**7,866**
1024	**1,992**	**2,999**	**4,091**	**5,098**	**6,171**	**7,182**	**8,229**

Podemos perceber que o ganho de tempo não foi considerável para domínios pequenos, claro que para tabuleiros grandes os valores são mais significativos. Isso fica claro na tabela de eficiência. Onde temos um padrão decrescente de ganhos a medida que aumentamos o numero de processos, mas temos um padrão crescente a medida que aumentamos o tamanho do problema.

Tabela 3 – Eficiências Calculadas – 1D

Tamanho Tabuleiro	2 Processos	3 Processos	4 Processos	5 Processos	6 Processos	7 Processos	8 Processos
32	**63,47**	**46,56**	**14,98**	**31,15**	**11,45**	**23,31**	**8,14**
64	**90,65**	**83,14**	**28,52**	**71,34**	**23,62**	**64,57**	**21,21**
128	**97,28**	**93,54**	**46,22**	**89,00**	**42,42**	**83,40**	**39,79**
256	**99,41**	**97,94**	**63,97**	**95,92**	**61,20**	**93,99**	**59,20**
512	**99,55**	**98,96**	**78,08**	**99,05**	**76,49**	**98,27**	**75,42**
1024	**99,44**	**99,72**	**88,55**	**101,72**	**90,03**	**102,36**	**89,18**

Para melhor ilustrar os resultados, montei esses dois gráficos, que facilitam a observação dos resultados. Pode-se ver no gráfico um, os speed-up sendo que nestes temos um padrão de crescimento para todos os números de processos, porém temos valores extremamente baixos para tabuleiros pequenos. Na verdade se compararmos a paralelização OpenMP, posso dizer que obtive Speed-up praticamente iguais para tabuleiros de tamanho 512 e resultados melhores apenas para tabuleiros de tamanho 1024.

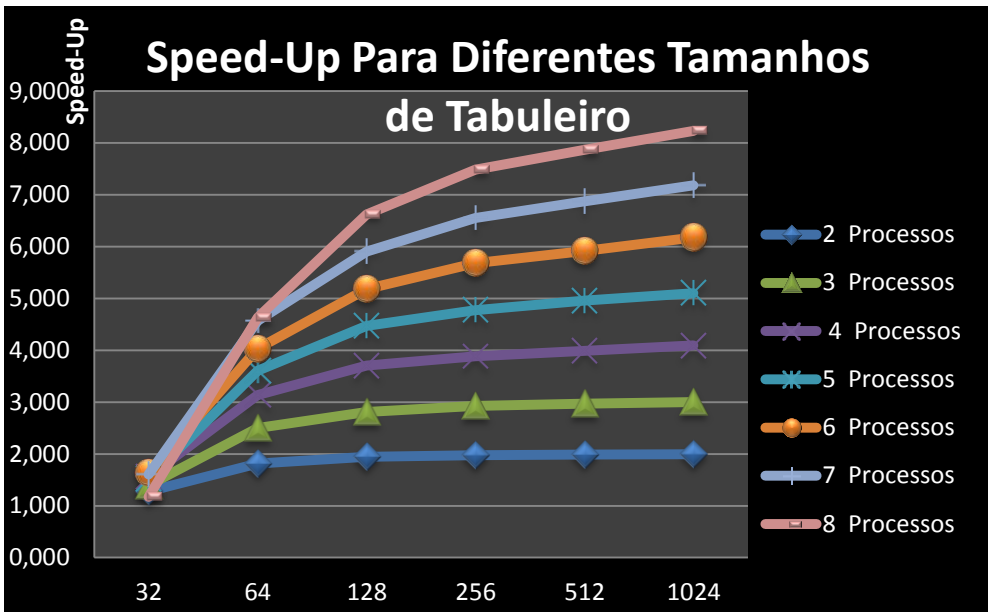

Gráfico 1 – Speed-Up 1D

O gráfico 2 nos mostra as eficiências, nele podemos ver claramente que, com o aumento do tabuleiro temos um aumento significativo da vantagem em se paralelizar o código. Já ele também reflete o efeito comentado anteriormente que para tabuleiros pequenos não se tem muito ganho em paralelizar o código. Temos para tamanhos grandes de tabuleiro eficiências maiores que 100%, isso se deve a termos Speed-up superiores ao numero de processos, isso garante uma ótima paralelização para o tamanho de tabuleiro, ou seja, que o tamanho do problema é perfeito para o número de processos utilizados.

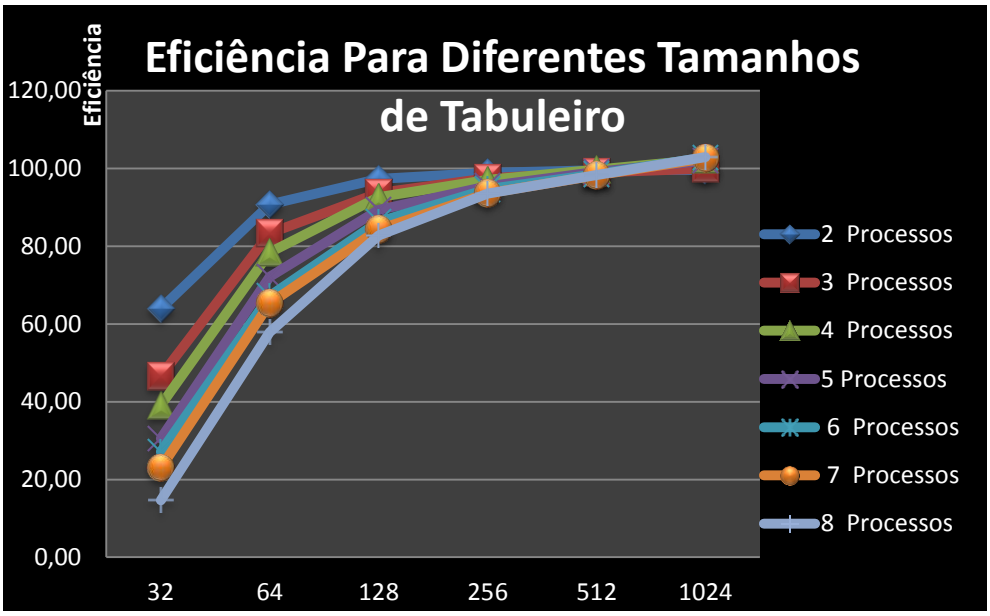

Gráfico 2 – Eficiência 1D

Resultados Decomposição Bidimensional

Na tabela 4 está descrito os resultados de tempo para a decomposição bidimensional do domínio, seguindo na tabela 5 e 6 temos respectivamente os speed-up e as eficiências calculadas.

Tabela 4 – Resultados de Tempo – 2D

Tamanho Tabuleiro	Tempo 1 Processo	Tempo 2 Processos	Tempo 3 Processos	Tempo 4 Processos	Tempo 5 Processos	Tempo 6 Processos	Tempo 7 Processos	Tempo 8 Processos
32	**0,000961**	**0,000757**	**0,000688**	**0,001604**	**0,000617**	**0,001399**	**0,000589**	**0,001475**
64	**0,007937**	**0,004378**	**0,003182**	**0,006958**	**0,002225**	**0,005601**	**0,001756**	**0,004678**
128	**0,065101**	**0,033460**	**0,023198**	**0,035216**	**0,014629**	**0,025575**	**0,011151**	**0,020454**
256	**0,528062**	**0,265587**	**0,179718**	**0,206355**	**0,110104**	**0,143799**	**0,080258**	**0,111502**
512	**4,259265**	**2,139261**	**1,434609**	**1,363759**	**0,860003**	**0,928048**	**0,619172**	**0,705928**
1.024	**35,321922**	**17,760307**	**11,806874**	**9,972869**	**6,945198**	**6,539065**	**4,929854**	**4,950853**

Tabela 5 – Speed-up Calculados – 2D

Tamanho Tabuleiro	2 Processos	3 Processos	4 Processos	5 Processos	6 Processos	7 Processos	8 Processos
32	**1,269**	**1,397**	**0,599**	**1,558**	**0,687**	**1,632**	**0,652**
64	**1,813**	**2,494**	**1,141**	**3,567**	**1,417**	**4,520**	**1,697**
128	**1,946**	**2,806**	**1,849**	**4,450**	**2,545**	**5,838**	**3,183**
256	**1,988**	**2,938**	**2,559**	**4,796**	**3,672**	**6,580**	**4,736**
512	**1,991**	**2,969**	**3,123**	**4,953**	**4,589**	**6,879**	**6,034**
1024	**1,989**	**2,992**	**3,542**	**5,086**	**5,402**	**7,165**	**7,135**

Tabela 6 – Eficiência Calculada – 2D

Tamanho Tabuleiro	2 Processos	3 Processos	4 Processos	5 Processos	6 Processos	7 Processos	8 Processos
32	**63,47**	**46,56**	**14,98**	**31,15**	**11,45**	**23,31**	**8,14**
64	**90,65**	**83,14**	**28,52**	**71,34**	**23,62**	**64,57**	**21,21**
128	**97,28**	**93,54**	**46,22**	**89,00**	**42,42**	**83,40**	**39,79**
256	**99,41**	**97,94**	**63,97**	**95,92**	**61,20**	**93,99**	**59,20**
512	**99,55**	**98,96**	**78,08**	**99,05**	**76,49**	**98,27**	**75,42**
1024	**99,44**	**99,72**	**88,55**	**101,72**	**90,03**	**102,36**	**89,18**

Podemos notar claramente as diferenças da abordagem anterior apenas com quatro, seis e oito processos, obviamente onde temos decomposição diferente da anterior, esta abordagem se deu pelo fato de não podermos dividir de forma uniforme domínios para determinados números de processos.

Temos uma perda significativa no speed-up para essa abordagem, creio que por causa das trocas de mensagens a mais, utilizadas para enviar as colunas e elementos "diagonais". Sei também que a forma para trocar as mensagens utilizadas não é a melhor e posso tentar empacotar os dados ou criar um tipo de dado MPI para enviar as colunas em uma única mensagem. Creio que assim ganharei em tempo, mas por enquanto os resultados são os apresentados.

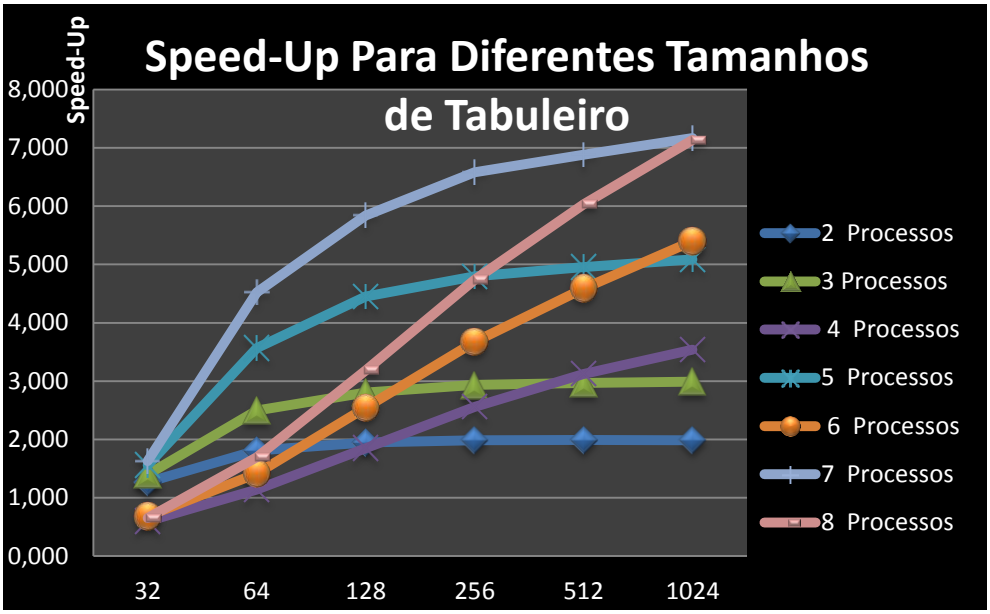

Gráfico 3 – Tempos de execução "CROW" X "CESUP"

Enfim pelos resultados de eficiência, não se obteve ganho para a abordagem bidimensional, neste caso a abordagem unidimensional é mais eficiente até mesmo que a paralelização por OpenMP (claro que para domínios maiores).

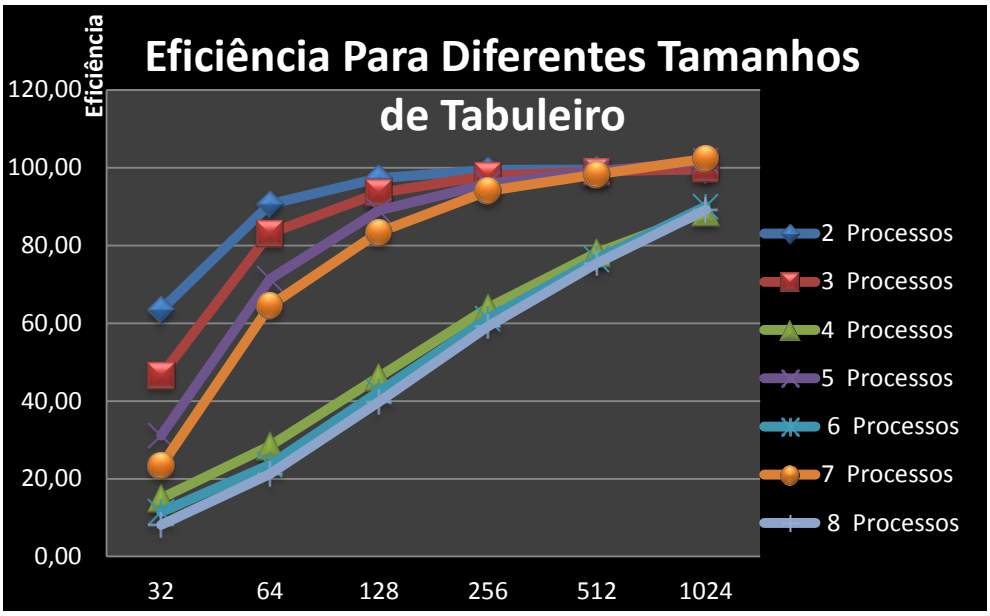

Gráfico 4 – Speed-Up "CROW" X "CESUP"

Para concluir quero dizer que esta atividade exigiu grande esforço para abstrair formas, que podem não ser as melhores, para que o domínio pudesse ser dividido, pode não parecer normal mas digo ser muito gratificante o fato de escrever um código, imaginar todo seu funcionamento e quando rodar ver que não funcionou apesar de parecer estar correto, e com mais um pouco de pensamento nota-se que na verdade faltou alguma coisa, um conceito sobre o real comportamento e com a mudança tudo funciona. Gostei muito da atividade, mesmo que não considere minha paralelização ideal.

Para torna-la ideal, a transferência das colunas deveria ser feitas de forma mais eficiente, como as empacotando e mandando em uma única mensagem, como também usar de outros métodos para que se evitassem muitas trocas de mensagens.