ISSN 1981-2841

COMPARAÇÃO ENTRE ABORDAGENS DE PARALELIZAÇÃO PARA O PROBLEMA DO JOGO DA VIDA[1]

COMPARISON BETWEEN BOARDINGS OF PARALEIZAÇÃO FOR THE PROBLEM OF THE GAME OF LIFE

Daniel Michelon de Carli[2], Eduardo Spolaor Mazzanti[2], Rodrigo Dewes[2], Ronaldo Canofre M. dos Santos[2], Valdir Stumm Júnior[2] e Andrea Schwertner Charão[3]

RESUMO

O "jogo da vida" é um exemplo clássico de autômato celular que simula a evolução de seres vivos, em um dado espaço, ao longo do tempo. Dependendo dos dados de entrada, o tempo de processamento de uma simulação pode ser elevado. Neste artigo, apresenta-se uma comparação entre duas abordagens de paralelização para o problema do jogo da vida, visando a redução do tempo de simulação, através do uso de múltiplos computadores em paralelo. Ambas abordagens utilizam o padrão MPI (Message Passing Interface) para implementação da troca de mensagens entre os computadores cooperantes, porém com diferentes estratégias de distribuição do trabalho e comunicação entre os processos. Os resultados obtidos evidenciam que a paralelização desse problema não é trivial, mas que, em certos casos, possibilita a obtenção de ganhos de desempenho com ambas abordagens. Além disso, mostra-se que a atuação de uma das implementações é ligeiramente superior à outra em todos os casos considerados.

Palavras-chave: autômato celular, seres vivos.

ABSTRACT

The Game of Life is a classical example of cellular automaton that plays the evolution of live beings in a given space, through time. Depending on the input data, processing time of a simulation can be high. This article presents a compari-

[1] Trabalho de iniciação científica - UFSM.
[2] Acadêmicos do Curso de Ciência da Computação - UFSM.
[3] Orientadora - UFSM.
{dcarli, mazzanti, dewes, canofre, junior, andrea}@inf.ufsm.br

son between two parallelization approaches for the Game of Life problem seeking to reduce simulation time using multiple parallel computers. Both approaches use the Message Passing Interface (MPI) to implement message exchange among cooperant computers,however, with different strategies for the distribution of work and communication among processes. The results obtained show that the parallelization of this problem is not trivial, but in certain cases it is possible to obtain a gain of performance with both approaches. Besides, it shows that the performance of one of the implementations is slightly superior to the other in all cases considered.

Keywords: *cell phone automaton, live beings.*

INTRODUÇÃO

O jogo da vida (*Game of Life*) é um autômato celular, desenvolvido pelo matemático Britânico John Horton Conway, em 1970. Esse autômato simula, através de um algoritmo simples, a evolução dinâmica de uma sociedade de organismos vivos, sendo aplicado em diversas áreas da ciência, como na biologia, na matemática, na economia, entre outras (BAK et al., 1989).

Esse autômato consiste em um jogo sem jogadores, pois recebe como dados de entrada uma população de seres dispostos sobre uma grade bidimensional (matriz) e a quantidade de tempo pelo qual essa população irá interagir. A cada iteração (passo de tempo), aplica-se um conjunto de regras simples à matriz, contendo os dados da "geração atual" e produzindo, assim, uma "nova geração".

A matriz utilizada para o jogo da vida é composta por células que podem possuir somente dois valores (0 ou 1). Caso a célula represente um ser vivo, ela será definida com o número 1, caso contrário, com 0. Uma célula que, inicialmente, contenha o valor 0 e possua três vizinhos contendo 1, assumirá o valor 1 (ou seja, irá nascer). Se a célula já possuir o valor 1 e tiver dois ou três vizinhos com o mesmo valor, ela se manterá viva. Em todos os outros casos, a célula passará a representar um ser morto (assumirá o valor 0).

O jogo da vida, bem como outros autômatos celulares, leva um longo tempo de processamento para dados de entrada que possuam um grande número de gerações e/ou uma matriz de considerável tamanho. A fim de reduzir esse tempo, pode-se utilizar o processamento paralelo.

A prática da programação paralela pode ser considerada como a divisão de um programa sequencial em partes menores que possam ser executadas, de forma simultânea, por processadores diferentes (WILKINSON; ALLEN,

2005). Com isso, torna-se possível reduzir o tempo total de processamento de um programa, permitir sua execução em um tempo aceitável, contar com a disponibilidade de mais recursos, como memória e capacidade de processamento e, também, diminuir custos com a utilização de um grande número de estações de trabalho, ao invés de um supercomputador.

A partir desse contexto, objetiva-se comparar duas implementações diferentes de paralelização de autômatos celulares, utilizando como exemplo o jogo da vida. Ambas implementações visam à redução do tempo de processamento para entradas com grades bidimensionais com uma ordem elevada ou um grande número de gerações. Para a realização de estudos comparativos, as duas abordagens utilizam estratégias diferenciadas para a realização da distribuição de carga entre os processos, tornando-se possível a visualização dos efeitos dessas escolhas nos resultados obtidos nas execuções.

Para descrever o trabalho realizado, o artigo encontra-se organizado da seguinte forma: nas seções um e dois, descreve-se uma metodologia clássica para paralelização e apresentam-se os algoritmos implementados; na seção três, expoe-se a metodologia utilizada para avaliação do desempenho das implementações; na seção quatro, analisa-se os resultados e, na seção cinco, por fim, conclui-se o trabalho desenvolvido.

METODOLOGIA DE PARALELIZAÇÃO

Se há um problema a ser paralelizado, existe mais de uma forma de realização da divisão do trabalho para que os processos possam realizar-se de modo paralelo. A fim de facilitar e padronizar o desenvolvimento de um algoritmo paralelo, segue-se uma abordagem metódica, a qual provê mecanismos para avaliação das possíveis alternativas de implementação. Pode-se seguir essa estratégia no desenvolvimento de programas paralelos por meio de quatro estágios: particionamento, comunicação, aglomeração e mapeamento (FOSTER, 1995).

Na primeira fase, identificam-se os pontos potenciais em que o paralelismo possa ser explorado. Essa fase chama-se particionamento e realiza-se a partir de duas estratégias básicas, de acordo com a solução para o problema: particionamento por decomposição de domínio ou por decomposição funcional.

No segundo estágio do método analisa-se a comunicação necessária a ser realizada entre as tarefas geradas pela fase de particionamento. Embora o projeto dos programas paralelos busque diminuir ao máximo a necessidade de comunica-

ção entre os processos, dificilmente se consegue a eliminação, por completo, dessa custosa atividade. As estratégias de comunicação utilizadas nesse trabalho são várias e se aplicam a diferentes pontos considerados na implementação da solução paralela para o problema proposto.

O próximo estágio do projeto do algoritmo paralelo chama-se aglomeração e visa, em geral, ao decréscimo do número de tarefas geradas pela fase de particionamento. Sendo assim, a fase avalia-se como de extrema importância para o desempenho do programa paralelo, pois, caso tome-se uma decisão equivocada na fase de aglomeração, deixando os processos com tempo de comunicação maiores que o tempo de processamento, o desempenho do programa paralelo poderá obter um baixo desempenho com o programa sequencial.

Na última fase, denominada mapeamento, define-se o computador em que cada tarefa executar-se-á. Neste trabalho, essa fase foi executada concomitantemente à fase de aglomeração.

ALGORITMOS IMPLEMENTADOS

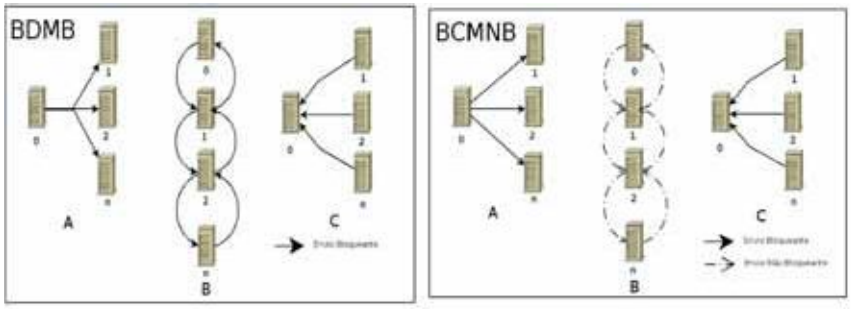
Para a implementação das duas abordagens comparadas neste artigo, utilizou-se a estratégia do particionamento por decomposição de domínio, cuja área de dados do problema representa-se pela grade bidimensional e se divide entre os processos, para que esses realizem o processamento local. A divisão, nas duas implementações, aconteceu por linhas, ou seja, cada processo envolvido na computação recebeu uma determinada quantidade de linhas e por elas se responsabilizou.

A escolha por essa estratégia justifica-se pelo fato dos dados do problema formarem-se por uma matriz que pode distribuir-se facilmente entre os processos. Essa divisão deve ser preferencialmente igualitária, para que todos os processadores sejam totalmente aproveitados. Caso haja processadores ociosos ou sobrecarregados, devido à má distribuição do trabalho, o desempenho do programa paralelo ficará aquém do esperado.

A outra alternativa de particionamento por decomposição funcional necessita de uma análise sobre a computação que se realizará. Porém, para que se utilize essa abordagem, é necessário que se consiga dividir o problema em tarefas independentes, a fim de realizar o particionamento dos dados que se destinarão a cada uma dessas. Portanto, de acordo com a descrição do problema proposto, essa alternativa de particionamento torna-se inviável, pois há bastante dependência entre as computações.

Na fase da comunicação, definem-se os pontos críticos em que é necessária a comunicação entre os processos. O primeiro é após a leitura dos dados vindos da entrada padrão, pois o processo que realiza essa leitura deverá compartilhar com os outros a matriz que representa o tabuleiro do jogo da vida. Ambas implementações utilizam o padrão MPI (*Message Passing Interface*) (GROPP et al., 1996) para comunicação entre os processos. No entanto, há, nesse ponto, uma das diferenças básicas entre as duas implementações:

- na primeira implementação, chamada de BDMB (Balanceamento distribuído com mensagem bloqueante), devido às suas características, um processo realiza a leitura da grade e envia, através de chamadas, à função de comunicação coletiva MPI_Bcast, a matriz inteira para todos os processos, como pode ser visto na figura 1(a), item A;
- na segunda implementação, nomeada de BCMNB (Balanceamento centralizado com mensagem não bloqueante) também pelas suas características de comunicação, a matriz divide-se no processo que realiza sua leitura e, através de chamadas, à função MPI_Send, envia-se às submatrizes aos processos, para que esses possam realizar seu processamento, como está ilustrado na figura 1(b), item A.



(a) (b)

Figura 1: Diagrama de implementação dos algoritimos.

Um ponto importante analisado durante a fase de comunicação consiste no compartilhamento de linhas entre os processos durante o cálculo das gerações. De acordo com a definição do problema do jogo da vida, para que se calcule um novo valor a um item da matriz, precisa-se que se consultem os valores dos ítens vizinhos a ele. Esse detalhe traz uma complicação adicional à paralelização do problema, pois, para que um processo calcule os valores dos ítens de sua primeira

linha, necessita-se que se acessem os valores atualizados da última linha do processo responsável pela submatriz imediatamente acima. Do mesmo modo, para se calcular a última linha, precisa-se que o processo disponha da primeira linha da submatriz que se localiza imediatamente abaixo.

Esse compartilhamento de dados exige que os processos troquem mensagens entre si a cada cálculo de geração. Desse modo, as duas abordagens seguem estratégias um pouco diferentes com relação à comunicação. Enquanto a implementação BDMB realiza a troca de linhas entre processos, por meio de rotinas de envio bloqueantes (MPI_Send), a implementação BCNMB utiliza como recurso rotinas de envio não bloquantes (MPI_Isend), para que os processos não fiquem bloqueados durante o envio de suas linhas. Na figura 1, item B, ilustram-se as diferenças entre as duas abordagens, quanto ao tipo de comunicação utilizada.

A última fase da execução realiza a comunicação entre os processos, quando eles terminam de executar seus cálculos. Nesse momento, devem devolver suas submatrizes para que o processo coordenador realize a montagem da matriz de saída do programa. Na figura 1, item C, ilustra-se essa fase para ambas as implementações.

METODOLOGIA DE AVALIAÇÃO DE DESEMPENHO

A fim de avaliar e comparar o desempenho das duas implementações descritas, realizou-se uma série de execuções paralelas variando-se os parâmetros de entrada e o número de computadores utilizados.

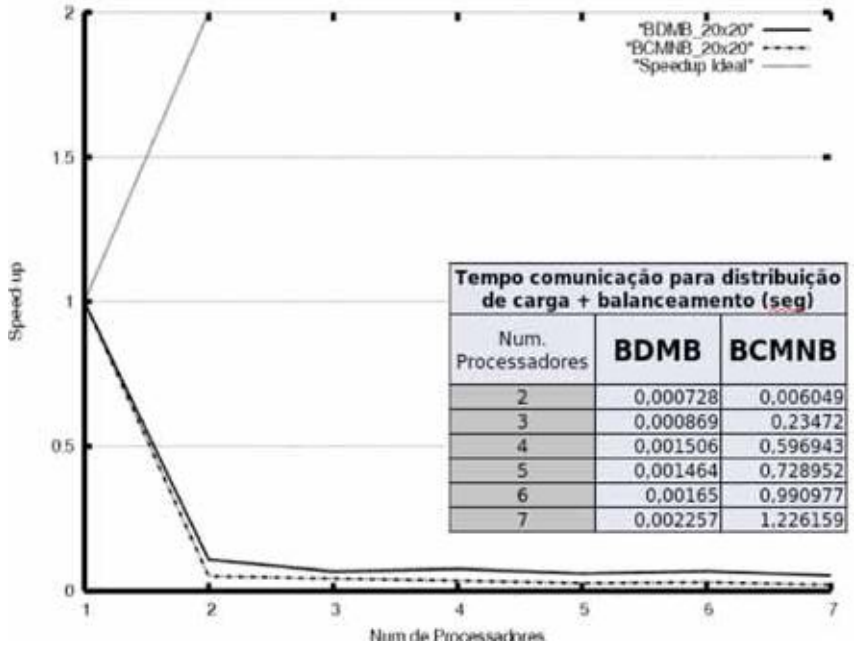
Os testes foram executados no núcleo da Ciência da Computação (NCC) da Universidade Federal de Santa Maria, em computadores *Intel*(R) *Pentium*(R) 4 de 2GHz, com 512Mb de memória RAM, rodando o sistema operacional *GNU/Linux*, versão 2.6.19, com distribuição Gentoo. Para a troca de mensagens entre as máquinas, acrescentou-se uma rede *fast ethernet* e para realização dos testes, criaram-se arquivos com grades de 20 linhas por 20 colunas (20x20), 500 linhas por 500 colunas (500x500) e 2500 linhas por 2500 colunas (2500x2500), todos com 5000 gerações (passos de tempo) - as diferentes implementações utilizaram os dados descritos.

A execução para a tomada dos tempos realizou-se com 1 processador (execução serial) e 2 a 7 processadores. Para ter uma melhor confiabilidade nos resultados, todos os testes foram executados 10 vezes e, então, calculou-se a média dos 5 melhores tempos, exceto quando os tempos medidos variavam 0,2 segundos da média. Nesse último caso, todos os tempos que estavam nessa margem de tolerância levaram-se em conta para o cálculo da média.

Essa metodologia foi adotada pela grande variação apresentada nos tempos para uma mesma matriz e um mesmo número de processadores. Esse problema deve-se ao fato das máquinas utilizadas não serem exclusivas para teste, podendo sofrer influência de outros usuários. Para complementar a análise dos resultados, calculou-se a aceleração (*speed-up*).

RESULTADOS

Para ambas soluções paralelas, o tempo de execução da matriz 20x20 constatou-se pior que o tempo do programa sequencial, fato que pode ser facilmente verificado pelo gráfico de *speed-up* mostrado na figura 2. Isso aconteceu porque os programas paralelos gastaram mais tempo com comunicação entre os processos do que propriamente acionando o algoritmo. Entretanto, o algoritmo BDMB obteve um desempenho ligeiramente superior pelo fato de ter gastado menos tempo na comunicação para distribuição das tarefas, conforme pode ser visto na tabela da figura 2.



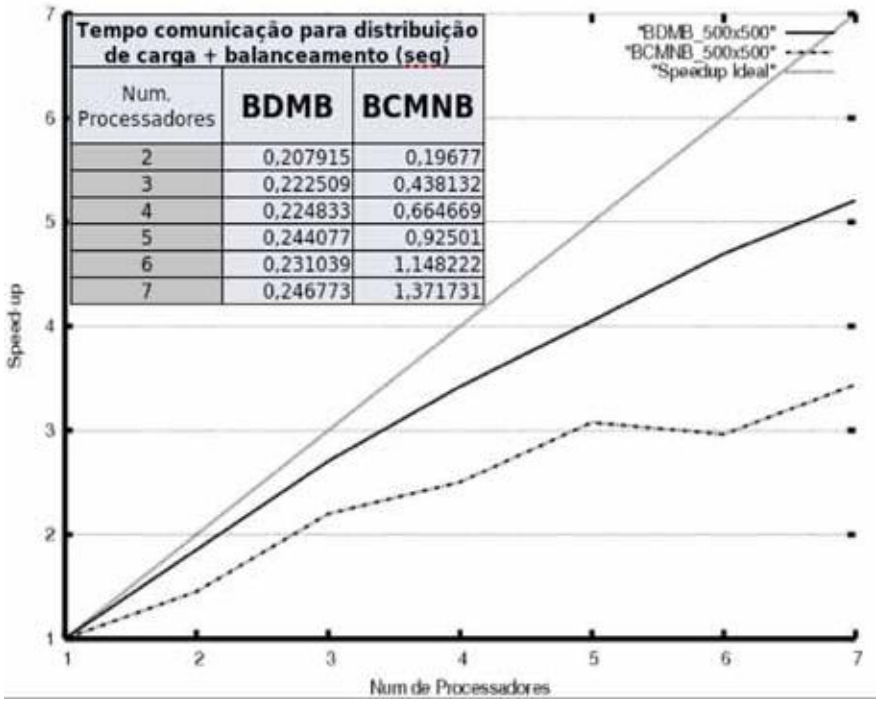
Tempo comunicação para distribuição de carga + balanceamento (seg)

Num. Processadores	BDMB	BCMNB
2	0,000728	0,006049
3	0,000869	0,23472
4	0,001506	0,596943
5	0,001464	0,728952
6	0,00165	0,990977
7	0,002257	1,226159

Figura 2: Gráfico do speed-up da matriz 20x20 e tabela comparativa de tempo gasto para distribuir as tarefas.

No caso da execução 500x500 e 2500x2500, pode-se avaliar, por meio dos gráficos da figuras 3 e 4, respectivamente, que a execução BDMB obteve um melhor desempenho.

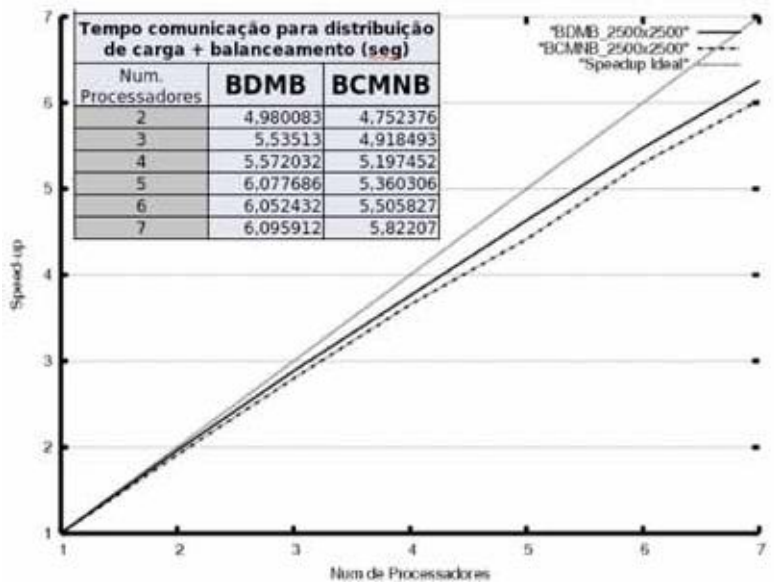
Para a matriz 500x500, isso se evidencia pela tabela da figura 3, a qual mostra que o tempo gasto pelo BDMB na comunicação e distribuição das tarefas é menor que a implementação que utiliza a estratégia BCMNB. Essa diferença de desempenho, em relação à implementação BCMNB, deve-se principalmente pelas características da rede *fast ethernet*. Essa rede, de forma geral, não apresenta um bom desempenho, principalmente em relação à sua latência. Nesse sentido, o BDMB facilita ao enviar somente uma única mensagem para todas as máquinas e cada máquina separa uma submatriz para efetuar seu processamento. Como o BCNMB divide a matriz em um todo centralizado, demora em redistribuir as tarefas aos demais processos. Desse modo, o BCMNB perde desempenho pelo fato da rede possuir uma latência relativamente alta.



Tempo comunicação para distribuição de carga + balanceamento (seg)

Num. Processadores	BDMB	BCMNB
2	0,207915	0,19677
3	0,222509	0,438132
4	0,224833	0,664669
5	0,244077	0,92501
6	0,231039	1,148222
7	0,246773	1,371731

Figura 3: Gráfico do *speed-up* da matriz 500x500 e tabela comparativa de tempo gasto para distribuir as tarefas.



Tempo comunicação para distribuição de carga + balanceamento (seg)		
Num. Processadores	BDMB	BCMNB
2	4,980083	4,752376
3	5,53513	4,918493
4	5,572032	5,197452
5	6,077686	5,360306
6	6,052432	5,505827
7	6,095912	5,82207

Figura 4: Gráfico do *speed-up* da matriz 2500x2500 e tabela comparativa de tempo gasto para distribuir as tarefas.

Contudo, ainda que o algoritmo BDMB apresente um melhor *speed-up* para a matriz 2500x2500, o desempenho do algoritmo BCMNB melhorou consideravelmente. Essa melhora deve-se ao fato da matriz 2500x2500 ocupar, aproximadamente, 12 Mb de memória RAM, sendo que, para o tráfego de uma matriz tão grande, utilizaram-se diversos datagramas. Nessa situação, essa abordagem mostrou-se eficiente, pois obteve desempenho mais próximo da implementação BDMB.

Deve-se ressaltar que, nesse último caso, a relação entre o tempo de computação e o tempo de comunicação é maior do que nos outros casos, uma vez que a matriz é de grande dimensão e cada processo calcula submatrizes de tamanho significativo. Esse fator beneficia ambas implementações, como se pode confirmar pelos resultados obtidos.

CONCLUSÃO

A apresentação da comparação entre a implementação serial e duas implementações paralelas de um problema do tipo autômato celular utilizou como exemplo o jogo da vida. Desse modo, demonstrou-se que a paralelização de autômatos auxulia na obtenção de resultados em casos de entradas muito grandes e com implementações diferenciadas.

A comparação dos algoritmos demonstra que, para matrizes de tamanho intermediário, existem algumas diferenças no tempo de execução, mas, à medida que aumenta a entrada, seus resultados aproximam-se. Sendo assim, ocorre a confirmação de que a paralelização tende a diminuir o tempo de processamento, por meio de respostas mais rápidas, independentemente de qual algoritmos utilizado.

No entanto, constata-se que a divisão ideal de cada matriz deve levar em consideração o número de processadores e a comunicação da rede, pois, do contrário, nem sempre o aumento do número de processadores melhorará o tempo de processamento.

REFERÊNCIAS

WILKINSON, B.; ALLEN, M. **Parallel Programming**. Pearson Prentice Hall, 2005.

BAK, P.; CHEN, K.; CREUTZ, M. **Self-organized criticality in the game of life**. **Nature**. n. 342, p.780–782, 1989.

FOSTER, I. **Designing and building parallel programs**. Addison-Wesley Inc, 1995.

GROPP, W. et al. **High-performance, portable implementation of the MPI message passing interface standard.** In: PARALLEL COMPUTING. v. 22, n. 6, p. 789–828, 1996.