Análise da Distribuição de Carga de Trabalho em MPI Utilizando o Jogo da Vida

Arthur Francisco Lorenzon[1], Márcia Cristina Cera[1], Fábio Diniz Rossi[2]

[1]Universidade Federal do Pampa - UNIPAMPA
CEP - 97546-550 - Alegrete - RS - Brasil

[2]Instituto Federal Farroupilha – Campus Alegrete
RS 377 - Km 27 - Caixa Postal 118 - Alegrete - RS - Brasil

af.lorenzon@gmail.com, marciacera@unipampa.edu.br,
fdrossi@al.iffarroupilha.edu.br

Resumo. *A eficiência de programas paralelos para memória distribuída está diretamente relacionada à distribuição da carga de trabalho entre os nós. Nesse sentido, apresentamos testes com a distribuição de carga com duas implementações paralelas da aplicação Jogo da Vida: uma que visa a distribuição da carga de forma igualitária entre os processos e outra que distribui chunks de dados sob demanda. As implementações baseiam-se no padrão MPI-1. Nossos testes mostraram que ambas obtiveram ganho de desempenho em relação a versão sequencial, porém a divisão igualitária, para este caso, foi a mais eficiente.*

1. Introdução

O desenvolvimento de programas MPI (*Message Passing Interface)* eficientes passa pela busca do método apropriado para dividir a carga de trabalho entre os processos MPI. Ademais, tal distribuição deve considerar fatores que permitam aumentar a escalabilidade da arquitetura alvo. Logo, a análise de diversos métodos de distribuição de dados se faz necessária, pois não existe uma solução ideal que contemple todos os problemas, havendo assim a necessidade de parametriza-lo buscando encontrar o método que melhor se adapta ao problema proposto.

Neste trabalho, o qual faz parte de um projeto maior, propomos a implementação do problema jogo da vida (*game of life)* em MPI-1, integrante da lista de problemas da suíte *Cowichan Problems*. Nosso objetivo é realizar a análise da divisão de trabalho de dois métodos utilizando troca de mensagens. A Seção 2 referencia alguns trabalhos desenvolvidos na área além de conter uma breve apresentação sobre o MPI e do nosso problema alvo. O desenvolvimento das duas versões implementadas pode ser encontrado na seção 3. Resultados experimentais são encontrados na seção 4. Na seção 5 é encontrada uma breve conclusão sobre os resultados obtidos além de conter os trabalhos futuros.

2. Referencial Teórico

Na literatura, [Hamd et al, 1995] realiza análise de métodos de distribuição de dados utilizando MPI. Uma análise sobre o jogo da vida utilizando MPI pode ser encontrada em [De Carli et al, 2007]. Nosso trabalho se torna complementar aos

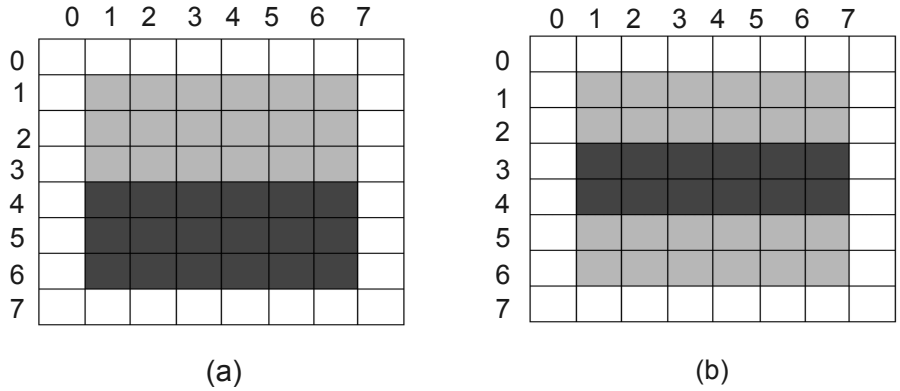

Figura 1. Distribuição de dados entre 2 processos trabalhadores (cinza claro e escuro). (a) Divisão dos dados de forma igualitária . (b) Representa a divisão dos dados em chunk de tamanho 2

trabalhos citados, pois realiza a análise de eficiência de duas diferentes formas de divisão de trabalho.

MPI é baseado no modelo de transmissão de mensagens e possui a grande vantagem de ser portável, tal padrão é um modelo no qual os nós de computação em um *cluster* não compartilham a memória [Gropp, 1999]. MPI provê comunicação ponto a ponto e coletiva, onde a primitiva `MPI_Send()` é responsável pelo envio dos dados ponto a ponto entre processos e o `MPI_Recv()` realiza o recebimento de dados.

Neste trabalho, utilizamos o modelo de programa Mestre/Trabalhadores, onde o mestre é responsável pela divisão e organização dos dados a serem processados pelos trabalhadores, e os trabalhadores são os processos que realizarão o trabalho a eles atribuídos. Com o propósito de analisar o impacto dos métodos de divisão de dados em aplicações MPI, optamos por realizar um estudo cuja aplicação alvo é o jogo da vida.

Proposto pelo matemático britânico John Horton Conway em 1970, o jogo da vida é um exemplo de autômato celular, isto é, um sistema de rede em que o estado de cada ponto da rede é determinado por regras locais [Gardner, 1970]. O jogo consiste em simular a evolução da vida em uma sociedade representada por uma estrutura bidimensional (matriz) de forma dinâmica seguindo algumas regras pré-definidas, as quais podem ser encontradas em [Gadner, 1970].

3. Implementação dos Algoritmos

Os algoritmos foram desenvolvidos utilizando a linguagem C. Implementou-se uma versão sequencial e duas versões paralelas utilizando MPI-1.

A Figura 1(a e b) mostra como foi realizada a divisão dos dados. Ambas as versões paralelas contemplam a mesma ideia, onde em um primeiro momento, o nó mestre designa e realiza o envio da carga de trabalho para os nós trabalhadores, além de realizar a gerência das bordas da matriz entre os trabalhadores. As bordas compreendem as células vizinhas entre as porções de dados, sendo a linha superior e inferior dos dados computáveis. Por exemplo, na figura 1(a) as bordas dos dados pertencentes ao processo um (em cinza claro) corresponde as linhas 0 e 5, enquanto as linhas 3 e 7 representam as bordas do processo dois (em cinza escuro).

Os trabalhadores computam a evolução das células e a cada iteração (evolução

da geração), realizam a troca de bordas com o mestre. Quando o número total de gerações for atingido, o mestre envia um sinal aos trabalhadores informando que a evolução da geração terminou, solicitando assim os dados locais para compor a matriz completa do jogo. Na implementação dos algoritmos, assumiu-se que para todas as execuções, um processo é representado pelo mestre e o restante pelos trabalhadores.

3.1. Algoritmo de Divisão Igualitária de Linhas

A implementação do Algoritmo de Divisão Igualitária de Linhas (ADIL), corresponde em realizar a divisão de trabalho de forma estática e igualitária da matriz pelo número de trabalhadores. A Figura 1(a), demonstra que cada nó trabalhador receberá de uma única vez os dados (cinza claro ou escuro) que deverá processar mais a borda.

3.2. Algoritmo de Divisão por *Chunk* de Linhas

A implementação do Algoritmo de Divisão por *Chunk* de Linhas (ADCL) realiza a divisão do trabalho baseada em um tamanho fixo de linhas definido no inicio da execução. O envio e recebimento dos dados ocorre de forma sincronizada. Desta forma, sempre que houver a demanda cada processo trabalhador receberá um *chunk* de linhas computáveis mais a borda enquanto houverem dados a serem computados. Na Figura 1(b) podemos visualizar o processamento dividido em *chunk* de 2.

4. Resultados Experimentais

As aplicações foram executadas em seis nós no *cluster* Labtec do GPPD/UFRGS[1]. Cada nó possui 2 processadores Pentium III 1.1 Ghz interligados por uma rede *fast ethernet.* O SO em uso foi o Debian *Lenny* e a distribuição MPI foi OpenMPI 1.4.4.

Foram realizadas 10 execuções para cada versão paralela e para cada quantidade de processos, onde o desvio padrão foi sempre inferior a 0,03. Para ambos os programas a entrada teve uma matriz de 4096x4096. Para a execução da versão ADCL, foi utilizado o *chunk* de 256, possuindo um total de 16 blocos. O *chunk* foi obtido através da análise de distribuição dos dados utilizando execuções prévias do algoritmo.

Analisando as Figura 2 e 3 afirmamos que em todas as execuções em paralelo a versão ADIL aproveitou melhor os recursos disponíveis devido a forma de distribuição, a qual é baseada no número de processos trabalhadores, o que faz com que nenhum dos trabalhadores fique ocioso. O mesmo ocorre para a versão ADCL quando executada com número de trabalhadores onde o número de blocos é divisível pelo número de trabalhadores.

Quando o número de blocos não é divisível pelo número de trabalhadores, há um

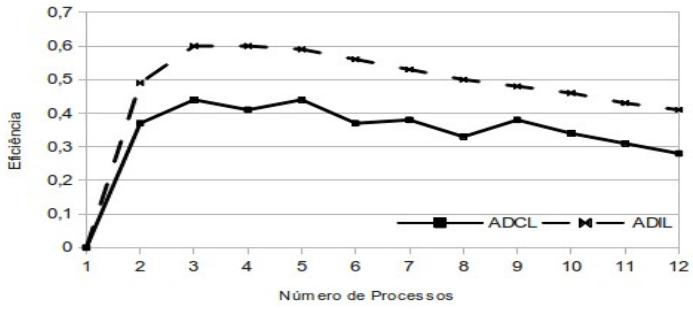

Figura 2. Grau de aproveitamento dos recursos computacionais

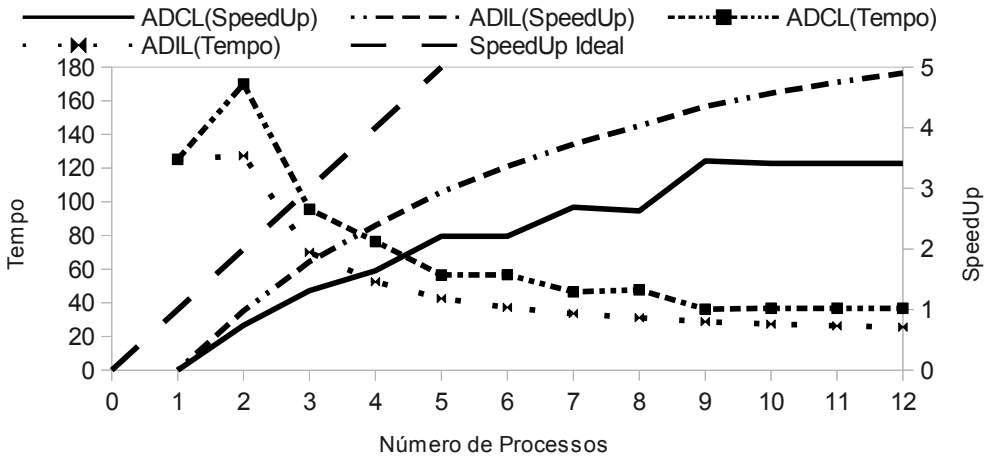

Figura 2. Tempo e SpeedUp obtidos em uma matriz de 4096, com a evolução da sociedade em 100 gerações

decréscimo na eficiência do ADCL, ou seja, para 4 processos (um mestre e três trabalhadores) enquanto os trabalhadores 2 e 3 realizarão o trabalho de 5 blocos cada, o trabalhador um realizará o mesmo trabalho em 6 blocos. Assim, enquanto o processo 1 executar sobre o bloco 6, os trabalhadores dois e três ficaram ociosos devido a transferência de dados sincronizados. A perda de desempenho da versão ADCL quando executada com 2 processos deve-se ao fato de que somente um trabalhador receberá 16 conjuntos de linhas separado, enquanto o trabalhador da versão ADIL receberá os dados de uma única vez, não tendo o ônus da rede como ocorre no ADCL.

5. Conclusão

Este trabalho buscou analisar a eficiência de dois métodos de distribuição de carga de trabalho. Mostramos que o desempenho de programas MPI é diretamente influenciado pela forma de divisão da carga de trabalho.

A fim de chegar próximo ao *speedup* ideal, pretendemos realizar melhorias evitando que trabalhadores fiquem ociosos devido às trocas de dados sincronizadas. Testaremos também o comportamento da distribuição de carga com métodos que dividem a matriz em blocos quadrados. Por fim, implementaremos utilizando MPI-2, a fim de avaliar a criação dinâmica de processos para o problema alvo.

6. Referências

De Carli, D. M., Mazzanti, E. S., Dewes, R., Dos Santos, R. C., Junior, V. S., Charão, A. S. (2007). “Comparação entre Abordagens de Paralelização para o Problema do Jogo da Vida.”, Anais do Simpósio de Informática da Região Centro do RS – 2007, Santa Maria.

Gardner, M. (1970-). “Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life", in Proceedings of the Scientific American 223, October – 1970. p. 120–123.

Gropp, W.; Lusk, E.; Skjellum, A. (1999) “Using MPI - Portable Parallel Programming with Message-Passing Interface”. 2.ed. The MIT Press, 1999. (Scientific and Engineering Computation Series).

Hamdi M. and Lee, C. K. (1995), “Dynamic load balancing of data parallel applications on a distributed network," in Proceedings of the 9th ACM Ining, p. 170-179