Análise de Desempenho do Algoritmo Paralelo de Monte Carlo Utilizando a Ferramenta OpenMP

Leonardo Lobo da Luz, Cristian Fernando F. Castañeda, Vitor Härter Guido, Gerson Geraldo H. Cavalheiro

Instituto de Informática
Universidade Federal de Pelotas
Caixa Postal 354 – 96.010-900 – Pelotas – RS – Brasil

{loboluz161,vitorharter}@gmail.com, {cristian.castaneda,gersonc}@anahy.org

Resumo. *Este artigo tem como objetivo analisar uma implementação paralela do algoritmo de Monte Carlo em termos de seu desempenho. Este método é muito utilizado em diversas áreas científicas, servindo como base para simular diversos fenômenos naturais e caracteriza-se pelo seu alto custo computacional. A implementação concorrente desse algoritmo se dá com o auxílio da ferramenta de programação paralela OpenMP e é apresentada uma avaliação de desempenho.*

1. Introdução

Monte Carlo é um algoritmo com um custo computacional razoavelmente elevado, mas, devido a sua natureza, é facilmente paralelizável. Arquiteturas e ferramentas de programação paralelas podem ser utilizadas para melhorar o desempenho de execução [Cercato 2005].

Este trabalho apresenta um estudo comparativo entre duas versões (seqüencial e paralela) do algoritmo de Monte Carlo. O objetivo é analisar o desempenho de ambas implementações em uma arquitetura *dual-core* (dois processadores).

Com relação a algoritmos para simulação de sistemas, o de Monte Carlo é um dos mais aceitos na comunidade acadêmica e científica, pois seus resultados são bastante próximos daqueles esperados, embora seja bastante ineficiente em termos computacionais. Isso deve-se ao fato dele realizar uma quantidade considerável de cálculos para guiar o curso da simulação a ser gerada. Conseqüentemente, o desempenho do sistema como um todo fica bastante prejudicado [Cercato 2005]. O interesse deste trabalho é avaliar o uso de OpenMP em arquiteturas *multi-core* como ferramental de suporte a implementação desta aplicação.

O restante deste artigo está organizado como segue. A Seção 2 apresenta o modelo de Potts Celular. A seção 3 apresenta o algoritmo de Monte Carlo. A Seção 4 discute a implementação paralela deste algoritmo e a Seção 5 apresenta resultados de desempenho do protótipo. Por fim, a Seção 6 conclui o trabalho.

2. O Modelo de Potts Celular

O modelo de Potts Celular é um modelo computacional desenvolvido por Glazier e Graner com a finalidade de simular a reorganização celular fruto da evolução das

interações entre células no ciclo evolutivo de um sistema [Graner 1992]. Este modelo é largamente utilizado em diversos tipos de simulações, dentre eles o crescimento de tumores, desenvolvimento dos membros de aves e a interação entre espumas de sabão [Cercato 2005].

Na sua forma convencional, o modelo de Potts Celular apresenta um grande custo computacional, devido ao elevado número de cálculos a ser realizados. Conforme o tipo de simulação a ser gerada e o número de células do sistema foco, a quantidade de processamento requerida em uma máquina mono-processada pode tornar inviável a simulação de determinados sistemas. Este modelo utiliza o método de Monte Carlo para evoluir a simulação do sistema específico [Cercato 2005].

O modelo de Potts Celular é baseado na Hipótese de Adesão Diferenciada para realizar o cálculo de adesão de energia entre as células. A hipótese de adesão é um modelo termodinâmico, segundo o qual a interação entre as células envolve uma energia de adesão superficial que depende das moléculas de adesão nas membranas celulares [Cercato 2005].

3. O Algoritmo de Monte Carlo

O algoritmo de Monte Carlo é um método numérico utilizado para encontrar soluções para problemas matemáticos, os quais podem ter muitas variáveis, e que não podem ser facilmente resolvidos através de outros métodos numéricos conhecidos. Esse método pode ser aplicado em áreas científicas diversas, como a Matemática, a Física e a Biologia. Devido ao seu grau probabilístico, serve de auxílio a diversas simulações de fenômenos naturais [Newman 1999].

O algoritmo de Monte Carlo, quando aplicado ao modelo de Potts Celular, permite selecionar células em um sistema para fazê-lo evoluir. Um passo de simulação através deste algoritmo corresponde ao sorteio de tantas células quantas houver no sistema, caracterizando, desta forma, seu custo computacional [Cercato 2005].

4. Implementação Multithread em OpenMP

OpenMP [Chandra et al 2001] é uma interface de programação proposta em 1997 que oferece diversos recursos para a criação, desenvolvimento e execução de programas *multithread*. O suporte a esta API encontra-se implementado em C/C++ e *Fortran* sobre diferentes sistemas computacionais, incluindo plataformas *Unix* e *Microsoft Windows*. Além de ser extremamente portável, OpenMP propicia aos programadores uma interface eficiente e flexível para desenvolver aplicações paralelas para plataformas diversas, desde *desktops* até supercomputadores. É composta por uma série de recursos, tais como: diretivas de compilação, chamadas a funções de biblioteca e variáveis de ambiente.

OpenMP pode ser caracterizada como uma interface de programação voltada para aplicações cuja natureza paralela é melhor expressa pela decomposição de dados. Isto significa que os dados são decompostos, resultando em várias partes que são divididas entre cada uma das *threads* envolvidas no processo de execução da aplicação em questão. Cada *thread* executa o mesmo código, mas sobre um conjunto diferente de dados.

5. Análise de Desempenho

O algoritmo analisado neste trabalho visa simular estruturas celulares a ser representadas em um espaço discreto. Neste caso, uma célula é constituída por um grupo de rótulos (posições de uma matriz) espacialmente conexos e de mesmo valor numérico. No protótipo implementado foi utilizada uma estrutura de dados bi-dimensional.

O funcionamento do algoritmo de Monte Carlo ocorre da seguinte forma: um rótulo qualquer é sorteado (rótulo preto central na Figura 1). Após, é sorteado um dentre os oito vizinhos daquele rótulo que fora escolhido anteriormente (rótulos cinza-escuro na Figura 1). É feita então uma verificação em torno desses oito vizinhos mais próximos e dos outros 12 vizinhos adjacentes (cinzas claro e escuro na Figura 1). Para cada vizinho diferente do rótulo escolhido, a energia inicial, que é o valor de energia atual do sistema, é incrementado de um (1). Ao final deste processo, o mesmo é repetido para o rótulo vizinho escolhido (qualquer um dos 8 vizinhos em cinza-escuro da Figura 1). Ou seja, é utilizado este rótulo vizinho, sorteado na figura para simular que foi feita a troca do rótulo sorteado com este rótulo vizinho. Para cada vizinho diferente, o valor da energia final, que é o valor que o sistema terá se ocorrer a troca do rótulo sorteado com um dos seus oito vizinhos, será incrementado de um (1). Após, a energia final do sistema será subtraída da energia inicial, constituindo a diferença de energia. Se a diferença de energia for igual a zero, ou seja, se a energia não foi alterada, deve-se considerar a probabilidade de troca definida para o sistema. Caso ocorra, a troca consiste em substituir o rótulo sorteado com o rótulo vizinho escolhido. Se a diferença de energia for menor do que zero, a troca é realizada incondicionalmente.

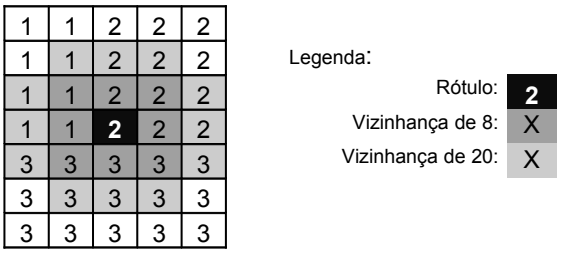

Figura 1 – Rótulo sorteado e vizinhanças de 8 e 20 rótulos

Neste trabalho, foram realizadas duas simulações, uma seqüencial e outra paralela do mesmo algoritmo. Ambas foram implementadas sobre o sistema operacional Mac OS X, utilizando a linguagem de programação C e o compilador icc (Intel). A arquitetura escolhida foi um Pentium IV *dual-core* com 1GB de memória RAM. Foram realizados dois casos de estudo, com matrizes de tamanhos 1000x1000 e 5000x5000. Desempenhos, tempos de execução (considerado em segundos), são apresentados nas Tabelas 1 e 2. A simulação foi evoluída por 1 milhão de sorteios aleatórios.

Como mostram os resultados, OpenMP apresenta-se como uma ferramenta eficiente na exploração do hardware disponível. A melhor execução, considerando ambos tamanhos de matrizes, foi obtida com duas *threads* de serviço. Um número maior de *threads* no suporte a execução implica em aumento do sobrecusto de execução sem ganhos de desempenho.

Tabela 1 – Desempenho de execução das versões em uma matriz 1000x1000

Versão	Nº Threads	Tamanho	Tempo Execução	Utilização CPU
Seqüencial	1	1000x1000	0.8753	99.8%
Paralelo	2	1000x1000	0.2061	181.7%
Paralelo	4	1000x1000	2.4317	180.4%
Paralelo	6	1000x1000	2.7421	177.8%

Também foi observado, no experimento, que a aplicação caracteriza-se por ocupar grande parte do tempo de processamento disponível na CPU. Nas versões seqüenciais, a taxa de utilização da CPU anotada foi muito próxima a 100%, enquanto que nas versões paralelas, a utilização esteve muito próxima a 180%, observando o fato da existência de dois processadores. A taxa de utilização apresentada refere-se a utilização máxima observada com ferramentas de monitaroção providas pelo sistema operacional.

Tabela 2 – Desempenho de execução do algoritmo em uma matriz 5000x5000

Versão	Nº Threads	Tamanho	Tempo Execução	Utilização CPU
Seqüencial	1	5000x5000	1.5215	99.7%
Paralelo	2	5000x5000	1.1522	181.1%
Paralelo	4	5000x5000	2.5954	180.7%
Paralelo	6	5000x5000	3.4982	179.5%

6. Conclusão

Este trabalho apresentou uma implementação paralela do modelo de Potts Celular evoluído através da estratégia de Monte Carlo. Foi possível observar os benefícios da utilização da ferramenta de programação OpenMP, por permitir, pelo uso de uma interface de programação simples, obter bons índices de desempenho em uma arquitetura *dual-core*. A continuação deste trabalho será a avaliação de outras estratégias para evolução de Potts Celular, como por exemplo, o algoritmo de *Random Walker*, que é utilizado como uma alternativa mais eficiente em termos de desempenho do que o algoritmo de Monte Carlo [Cercato 2005].

Referências

Cercato, F. P. (2005) “Um Algoritmo de Alto Desempenho para Evoluir o Modelo de Potts Celular”. Dissertação de Mestrado. São Leopoldo: UNISINOS.

Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., and Menon, R. (2001) “Parallel Programming in OpenMP”. Morgan Kaufmann, San Francisco.

Graner, F.; Glazier, J. A. (1992) “Simulation of Biological Cell Sorting Using a Twodimensional Extended Potts Model”. *Physical Review Letters*, v. 1, n. 69, p. 2013-2016.

Newman, M. E. J.; Barkema, G. T. (1999) “Monte Carlo Methods in Statistical Physics”. 1. ed. [S.l.]: Oxford University Press.