Modelagem concorrente para simulações de Monte Carlo baseadas no modelo de Ising

Adonize Bonetto, Fábio Mierlo

Unisinos - Universidade do Vale do Rio dos Sinos - Curso Informática - Software Básico
Av. Unisinos, 950 - CEP 93022-000 - Brasil
{adonize@sysmaker.inf.br, fmierlo@yahoo.com}

Introdução

A programação paralela tem sido utilizada em diversas áreas como um recurso para melhorar o desempenho na resolução de problemas com alto custo computacional, por exemplo, em simulações de fenômenos físicos. Em especial, simulações baseadas em técnicas de Monte Carlo permitem um considerável aumento de desempenho quando resolvidas em paralelo pois podem ser consideradas trivialmente paralelizáveis [LEM02].

Segundo [WIL99], a base do método de Monte Carlo é o uso de números aleatórios na solução de problemas físicos. Sendo cada cálculo independente dos outros, onde cada número gerado corresponde ao cálculo da evolução da simulação de um determinado ponto do sistema tratado, o problema pode ser facilmente expresso de forma concorrente. Em relação ao fator tempo, essa técnica é lenta, pois a qualidade do resultado depende da quantidade de números gerados, portanto depende do tamanho da simulação.

Neste trabalho são descritos um modelo e duas implementações concorrente para um algoritmo de simulação que utiliza a técnica de Monte Carlo, baseada no modelo de Ising, usando algoritmo de Metrópolis.

Abordagem Atual

A maioria dos sistemas físicos não são isolados [GOU96], eles trocam energia com seu ambiente. Sabe-se que estes ambientes agem como um reservatório (ex.: uma banheira) aquecido a uma temperatura fixa. Se um sistema de menor proporção é colocado em contato térmico com um reservatório aquecido, o sistema atinge o equilíbrio térmico trocando energia com este reservatório.

O modelo de Ising foi inicialmente proposto para descrever as propriedades de um ferromagneto [LEM02], ou seja de um ímã. A sua premissa fundamental é a de que as propriedades magnéticas de um ferromagneto decorrem das interações entre os momentos de dipolo magnético dos átomos que compõe o material. O modelo assume uma rede (que pode possuir a geometria e o número de dimensões que quisermos) com um momento de dipolo ou spin em cada sítio.

O algoritmo seqüencial proposto por [LEM98] é baseado no algoritmo de Metrópolis e utiliza o modelo de Ising de duas dimensões. Para facilitar a simulação todos os spins são armazenados em um vetor unidimensional e as matrizes up, down, left e right contém seus vizinhos. O número total de spins é $N = L^2$, onde L é o tamanho de um lado da rede. Esta rede é inter-conectada no formato de Torus, dessa forma os

spins da coluna esquerda interagem com os da coluna direita e os spins da linha superior com os da linha inferior.

Durante a sua execução várias cópias do sistema são evoluídas separadamente, inicialmente até atingirem o equilíbrio, depois por mais T passos. A cada passo a magnetização e a energia são medidos. Na última etapa os resultados provenientes de todos os processos são somados e o valor da magnetização e da energia são calculados. Por comodidade estas quantidades são medidas para vários valores da temperatura.

Nesta abordagem, o tamanho (largura e altura) do sistema é definido por uma constante chamada n, com n^2 spins.

Este algoritmo tem como características:

- não fazer I/O durante o processamento (somente nos dados de saída);
- a maior parte da inicialização dos dados são aleatórias, independentes e baseadas em n^2 (logo podem ser concorrentes);
- código principal (onde se passa 99% do processamento) também é baseado em n^2, é aleatório e como dado de saída, em alguns casos, ele apenas inverte o spin que está sendo verificado;
- custo de execução exponencial (e^n).

Modelo Concorrente

Dadas suas características (pouco sincronismo e comunicação) este algoritmo pode ser modelado para um processamento paralelo, explorando a independência das tarefas. Uma implementação concorrente para o problema pode considerar a divisão da matriz em regiões, onde estas são computadas separadamente. No entanto deve ser observado que próximo as fronteiras das regiões o cálculo de um spin vai necessitar de informações advindas dos spins presentes na região vizinha, gerando uma necessidade de sincronização no cálculo dos spins localizados na fronteira da região.

Para solucionar esta dependência de dados, pode-se utilizar uma seção crítica. Este mecanismo garante que apenas uma tarefa está acessando a um determinado dado.

Se um spin eleito pertencer a fronteira da matriz, a tarefa seguirá estes passos:

- faz uma requisição de acesso.
- aguarda a sua vez para entrar na seção crítica.
- entra na seção crítica
- faz o cálculo.
- sai da seção crítica.

Um instante da execução de uma simulação pode ser visto na Figura 1, onde 16 tarefas, uma para cada região, são executadas de forma concorrente. Uma destas tarefas encontra-se realizando cálculo na fronteira da região, três estão esperando para fazer o cálculo na fronteira da região, duas estão fazendo um requisição para o cálculo na fronteira da região e dez estão fazendo os cálculos sem dependência de outras tarefas.

Este modelo foi implementado sobre duas plataformas de hardware distintas: uma com memória compartilhada e outra com memória distribuída.

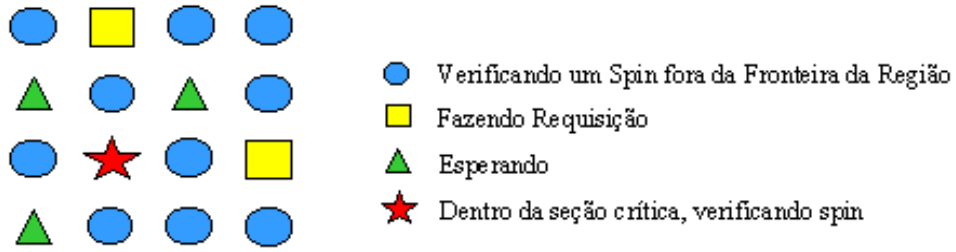

Figura 1: Estado das tarefas em um instante.

Implementação em Memória Compartilhada

A implementação do programa concorrente sobre uma arquitetura com memória compartilhada fez uso de threads disponibilizada pela biblioteca Pthreads.

Seguindo o modelo concorrente, realizou-se a repartição dos dados manipulados: a matriz foi dividida em linhas horizontais, sendo as fronteiras consideradas as posições Norte e Sul das linhas. Esta divisão foi escolhida para reduzir o número de fronteiras entre os nodos e consequentemente a quantidade de controle e interações entre eles.

O cálculo de cada região foi delegado à uma thread. O princípio de execução está na escolha aleatória de um spin e um cálculo baseado em seus vizinhos. Caso o spin escolhido pertença a uma fronteira, um procedimento especial deve ser realizado. Como o cálculo de um spin necessita dos valores dos vizinhos é necessário recuperar informações manipuladas por outras threads. Neste caso o algoritmo empregado é o seguinte:

- a thread inicia uma nova tarefa para cálculo do spin na fronteira;
- esta nova tarefa aguarda a liberação do mutex;
- uma vez em execução, a tarefa calcula o spin, se necessário altera seu valor;
- a tarefa termina e libera o mutex.

Nesta implementação a inicialização dos dados, as correlações de tempo e o cálculo para a saída de dados foram mantidas seqüenciais. A matriz não foi dividida fisicamente, mas sim através de cálculos. Por este motivo o mutex foi escolhido para garantir o acesso exclusivo nas fronteiras das regiões, sendo um mutex associado à cada fronteira.

Implementação em Memória Distribuída

Para a implementação do problema sobre uma arquitetura com memória distribuída, optou-se, a exemplo da implementação em Memória Compartilhada, de dividir a matriz em regiões horizontais. No entanto, as regiões encontram-se fisicamente distribuídas nos módulos de memória pertencentes aos nodos da arquitetura. Para a distribuição das regiões e para sincronização entre os nodos, foi utilizado MPI.

Devido a esta divisão geográfica, a inicialização dos dados, as correlações de tempo e o cálculo para a saída dos dados foram implementados de forma paralela. Tendo como característica adicional o uso de barreiras para a sincronização das tarefas e

um nó exclusivo para o armazenamento intermediários das correlações de tempo e dos cálculos para a saída de dados.

A complexidade do código e a correção do programa, estão intimamente ligados a sincronização entre as tarefas. As barreiras utilizadas permitem controlar as comunicações e a evolução do processo de simulação.

Conclusão

Através dos resultados obtidos foi possível verificar que as implementações concorrentes foram desenvolvidas conforme foi especificado no modelo. Notamos nos testes em máquinas com multiprocessador que a implementação usando Pthreads consegue obter um aumento de desempenho proporcional ao número de processadores em relação ao algoritmo seqüencial, resultado este esperado por se tratar de uma aplicação trivialmente paralelizavel.

Nos testes sobre uma arquitetura com memória distribuída, a implementação usando MPI se mostrou eficiente mesmo com o seu custo adicional de comunicação. No entanto, mecanismos de balanceamento de carga dinâmicos poderiam ser introduzidos para otimizar a exploração dos recursos computacionais disponíveis.

Neste sentido acreditamos que, este modelo concorrente pode evoluir, transformando-se em uma biblioteca para ser utilizada em problemas que envolvam matrizes, dependência de dados em tarefas vizinhas (desde que satisfaçam a condição de fronteira) e sincronismo entre estas tarefas.

Referências

[GOU96] GOULD, Harvey. An introduction to computer simulation methods: applications to physical systems. - 2. ed. - Reading: Addison-Wesley, 1996. xiv, 721 p.

[LEM02] LEMKE, Ney. Problemas Trivialmente Paralelizáveis. Anais: Segunda Escola Regional de Alto Desempenho -ERAD. São Leopoldo, 2002.

[LEM98] LEMKE, Ney. MODEL - 2D - Out of Equilibrium, Version 0.00 20.11.1998.

[WIL99] WILKINSON, Barry. Parallel programming : techniques and applications using networked workstations and parallel computers. - New Jersey : Prentice-Hall, 1999. xv, 431.