Implementação Paralela do Algoritmo Split utilizando a API OpenMP

Antonio Argeu Moreira de Lima[1], **César A. F. De Rose** [1]

[1]Instituto de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - Partenon - Porto Alegre/RS - CEP: 90619-900

antonio.lima@acad.pucrs.br,cesar.derose@pucrs.br

1. Introdução

A análise de desempenho de sistemas reais realizada por meio de formalismos para modelagem analítica tem grande aplicabilidade na previsão de desempenho de aplicações paralelas, bem como na avaliação de serviços presentes em sistemas distribuídos. Nesse contexto, o formalismo estruturado SAN (*Stochastic Automata Networks*) [Plateau 1984] oferece vantagens em relação ao formalismo de Cadeias de Markov [Stewart 1994], permitindo que a matriz de transição de estados seja armazenada de forma compacta, por meio de uma representação tensorial denominada Descritor Markoviano (DM).

Devido à estrutura do DM ser baseada em fundamentos da álgebra tensorial, o processo de solução numérica de SAN só é possível através de algoritmos especializados que realizam a chamada Multiplicação Vetor-Descritor (MVD). Algoritmos como o *Shuffle* [Fernandes et al. 1998] e o *Split* [Czekster et al. 2007] constituem-se exemplos de algoritmos que realizam a MVD. Contudo, mesmo com algoritmos sequenciais otimizados, pode-se levar dias para atingir os resultados desejados de modelagens com grande espaço de estados. Como a solução é constituída de um processo iterativo, a MVD é repetida diversas vezes, normalmente milhares de repetições até que a solução dita estacionária seja alcançada. Dessa forma, a paralelização da MVD representa uma alternativa para obter-se mais rapidamente os resultados de modelagens SAN.

2. Algoritmo Split

O processo de solução de modelagens SAN prevê a execução da MVD que representa a multiplicação do vetor de probabilidades π pelo Descritor Markoviano que, por sua vez, é formado por termos tensoriais. A solução que o algoritmo *Split* apresenta é a de quebrar um termo tensorial em duas partes. Para o conjunto de matrizes da primeira parte do termo tensorial, aplica-se a propriedade de *Decomposição Aditiva* que refere-se à decomposição de um termo tensorial na soma ordinária de matrizes, onde cada matriz possui um único elemento não-nulo [Czekster et al. 2007]. Cada matriz formada é armazenada por meio de uma estratégia esparsa, onde somente os elementos não-nulos são armazenados. A partir do conjunto de escalares gerados e seus índices (linha e coluna), aplica-se um algoritmo baseado no algoritmo *Sparse* [Czekster et al. 2007] e o resultado obtido é dado como entrada para o algoritmo *Shuffle* que, por sua vez, é aplicado para o segundo conjunto de matrizes do termo tensorial.

3. Implementação paralela utilizando a API OpenMP

A implementação paralela do algoritmo *Split* foi desenvolvida com base na biblioteca GOMP (*GNU OpenMP*) que fornece diretivas para programação paralela por memória

compartilhada do padrão OpenMP versão 2.5.

Inicialmente, existe a possibilidade de extrair paralelismo na MVD, multiplicando o vetor π por cada termo tensorial em paralelo. E, por outro lado, π é multiplicado por cada escalar e gerado por meio da aplicação da propriedade de decomposição aditiva. Essa operação também constitui-se em uma tarefa que pode ser executada concorrentemente, porém atua em uma granularidade mais fina do algoritmo. As duas abordagens de paralelização foram realizadas e devido ao algoritmo ser fortemente baseado em laços de repetição onde cada iteração independe de outras iterações, definiu-se a aplicação da diretiva *#pragma omp for* para paralelização dos laços. Em ambas as versões, o vetor π é tratado como um dado global, compartilhado entre as *threads*. Para possibilitar a atualização simultânea dos resultados de cada *thread* em π foi utilizada a diretiva *atomic* que apresenta-se como uma maneira eficiente de atualizar um dado compartilhado. Além disso, dentre as estratégias de escalonamento disponíveis para a diretiva *omp for*, optou-se pelo uso de um escalonamento dinâmico com granularidade de tarefa igual a 1 (cláusula *schedule(dynamic,1)*).

Através de testes realizados, pode-se verificar um bom ganho de desempenho. Entretanto, existem casos onde a paralelização por escalar pode não funcionar bem. A quantidade de escalares de um termo tensorial pode ser inferior ao número de *threads* e, assim, *threads* ficam sem trabalho. Da mesma maneira, o total de tarefas em uma estratégia de paralelização por termo tensorial pode ser menor que o número de *threads*. Logo, a quantidade de escalares por termo tensorial ou a quantidade de termos tensoriais pode variar em cada modelo, não sendo suficientes para realizar um balanceamento de carga ideal. Outro fator que pode prejudicar o balanceamento de carga, além da quantidade de tarefas, corresponde ao custo de cada tarefa que pode variar bastante.

Considerando esses dados, conclui-se que uma abordagem de paralelização sem o uso da diretiva *omp for* e com a aplicação de uma estratégia de particionamento de dados que considere tanto os escalares quanto os termos tensoriais como tarefas e, ainda, com um balanceamento de carga que leve em conta os custos teóricos de cada tarefa é uma estratégia que pode produzir uma distribuição de trabalho mais adequada e, assim, um melhor ganho de desempenho.

Referências

Czekster, R. M., Fernandes, P., Vincent, J. M., and Webber, T. (2007). Split: a flexible and efficient algorithm to vector-descriptor product. In *ValueTools '07: Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, 8p, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Nantes, France. ICST, Brussels, Belgium, Belgium.

Fernandes, P., Plateau, B., and Stewart, W. (1998). Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks. *JACM*, 45(3):381–414.

Plateau, B. (1984). De l'evaluation du parallelisme et de la synchronisation. Paris-Sud, Orsay.

Stewart, W. J. (1994). *Introduction to the Numerical Solution of Markov Chains*. Princenton University Press.