Proposta de uma Implementação Paralela de um Algoritmo para a Resolução de um Problema de Seqüenciamento de Padrões de Corte

Daniel Merli Lamosa[*, 1], Horacio Hideki Yanasse [**, 1], Airam Jônatas Preto [**, 2]
(1) Área de Pesquisa Operacional
Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais (INPE)
(2) Área de Computação Científica e Processamento de Alto Desempenho
Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais (INPE)
(*)Mestrado, Bolsa CAPES, e-mail: lamosa@lac.inpe.br; (**) Orientadores

Resumo

Neste trabalho propõe-se uma implementação paralela de um algoritmo *branch-and-bound*, sugerido anteriormente na literatura, para a resolução do problema de seqüenciamento de padrões de corte com o objetivo de minimizar o número máximo de pilhas abertas. Testes computacionais serão realizados para validar a política adotada de divisão do código visando um bom balanceamento de carga entre os processos. A motivação desse trabalho se dá pela escassez de artigos que tratam do problema e pela inexistência de publicações que implementem soluções paralelas para o caso específico do problema considerado.

Palavras chave: MOSP, branch-and-bound, Programação Paralela

Introdução

A ordem de processamento dos padrões de corte pode ser de grande importância em determinados processos produtivos. Diversos objetivos podem ser associados com o seqüenciamento de padrões de corte, por exemplo, a minimização do número máximo de pilhas abertas (**MOSP** acrônimo de *Minimization of Open Stack Problem*), a minimização do Espalhamento de Ordens (**MORP** acrônimo de *Minimization of Order Spread Problem*), a Minimização do Número de Descontinuidades (**MDP** acrônimo de *Minimization of Discontinuites Problem*), etc.

Nesse trabalho, focalizar-se-á o **MOSP** que consiste basicamente em encontrar uma seqüência de corte que abre, simultaneamente, o menor número de pilhas. Propõe-se o desenvolvimento de uma implementação paralela do algoritmo *branch-and-bound* (doravante denominado **b&b**) proposto por Yanasse [7] para a resolução do **MOSP**.

A motivação para desenvolver uma implementação paralela desse algoritmo se dá pela dificuldade em resolver este problema. De fato o **MOSP** é NP-Árduo, cf. [4], pois os algoritmos exatos encontrados na literatura (por exemplo, o trabalho de Limeira [3]), não conseguem ainda resolver um exemplo real, de tamanho moderado, de maneira exata em um tempo computacional satisfatório. Na próxima seção damos uma breve descrição do algoritmo **b&b** de Yanasse [7].

Lembramos que o **MOSP** tem uma relação intrínseca com um problema oriundo do desenho de circuitos **VLSI** conhecido como **GMLP**, acrônimo de *Gate Matrix Layout Problem* (vide [4]). O **GMLP** consiste de um conjunto de *gates* (fios verticais) com transistores (pontos) que são utilizados para conectar os *gates*. O objetivo desse problema consiste em diminuir a área do circuito que será impresso. Portanto, a resolução desse problema de maneira exata é bastante desejada.

Algoritmo branch-and-bound

No algoritmo exato de Yanasse [7] enumera-se a ordem em que cada tipo de item é completado. Na árvore de busca do método, cada nó ramificado representa um tipo de item que foi completado.

Dado um problema com *n* padrões e *m* itens diferentes podemos exemplificar o esquema de enumeração do algoritmo da seguinte forma:

Do nó inicial se ramificam os *m* possíveis tipos de itens completados por um determinado seqüenciamento de padrões. Ramifica-se, a partir do primeiro item completado, os demais itens que não foram completados ainda e assim sucessivamente até que todas as possibilidades sejam analisadas. No pior caso o algoritmo é exponencial, *cf*. [7]. Em cada nó ramificado *i* identifica-se um conjunto dos itens abertos S_o^i (número de pilhas abertas) e um conjunto dos itens ainda inacabados (não completados) S_u^i.

Para percorrer a árvore de busca é utilizado um critério guloso que escolhe o menor S_o^i para a próxima verificação no nível mais elevado corrente. Com isso obtêm-se, rapidamente, uma solução para o problema. Essa solução serve como uma solução inicial para a "poda" dos demais nós ainda não pesquisados. Um nó cujo S_o^i é maior ou igual que a solução obtida não precisa ser pesquisado, pois já se possui uma solução melhor.

Implementações Paralelas Propostas

Para iniciar a implementação paralela é necessário conhecer a estrutura computacional que será utilizada, pois a escolha da ferramenta adequada, a quantidade e de que maneira os processos serão executados concorrentemente dependem diretamente dessas informações. Dessa maneira alguns conceitos importantes serão apresentados.

A base do *hardware* que será utilizada consiste no *Computador de vonNeumann* que pode ser definido como uma máquina que contêm uma unidade central de processamento (CPU) conectada a uma unidade de armazenamento (memória).

Será utilizado um *multicomputador*, ou seja, um conjunto de nós (computadores de vonNeumann) interconectados em rede. Cada nó executa seu próprio programa e esse pode fazer acesso à memória local e enviar mensagens de leitura e escrita na rede. As mensagens são utilizadas para a comunicação entre os computadores para, por exemplo, requisitar dados da memória. No nosso caso específico, o multicomputador utilizado consiste de um *cluster* constituído de microcomputadores tipo PC com o sistema operacional Linux. A rede de interconecção consiste de uma rede *ethernet* de 100 *Mbits*.

As ferramentas (*softwares*) que serão utilizadas são: a biblioteca de comunicação de dados **MPI**, do acrônimo de *Message-Passing Interface*, e o suporte para multiprocessamento (*threads*) do sistema operacional Linux. Maiores detalhes podem ser encontrados em: [1], [2], [5] e [6].

Duas políticas básicas de divisão do algoritmo que se caracterizam pelas diferentes maneiras de gerenciamento das comunicações feitas entre os processos serão apresentadas e servirão como base para direcionar os estudos futuros de novas políticas.

A primeira e mais simples política adotada consiste em gerar o primeiro nível de um algoritmo **b&b** seqüencial e, a partir dele, aplicar, concorrentemente, um algoritmo **b&b** seqüencial para os níveis subseqüentes de cada nó criado. Esse procedimento pode ser resumido nos seguintes passos:

1. A partir do nível um aplicar, concorrentemente, um algoritmo **b&b** seqüencial para os *K* nós que abrem o menor número de pilhas, onde *K* = número de processos (programa em execução) que serão utilizados.
2. À medida que um processo é terminado, ou seja, um algoritmo **b&b** é finalizado, verificar se existe um nó, ainda não seqüenciado no nível um, que abra menos pilhas que a melhor solução encontrada até o momento. Caso encontre, seqüenciar o nó, independente se outros estejam sendo executados. Se não espere os outros processos terminarem e pare.

A principal característica desse procedimento foca-se no baixo nível de comunicação empregado entre os processos ativados no *cluster*, pois a comunicação restringe-se em verificar a solução corrente e atualizá-la se necessário. Esta política está em fase de implementação.

A segunda política que será implementada procura acelerar o processo de busca atualizando freqüentemente os limitantes para a solução ótima do problema. Pode-se resumi-la nos seguintes passos:

1. A partir do nível um, aplicar, concorrentemente, um algoritmo **b&b** seqüencial para os *K* nós que abrem o menor número de pilhas.
2. Quando alguma solução melhor for obtida (em qualquer processo) ela é atualizada e enviada para os demais processos através de um mensagem (*broadcast*).
3. Caso o número de pilhas abertas de cada ramo no primeiro nível seja maior ou igual a solução corrente, pode-o.
4. À medida que um processo é terminado, ou seja, um algoritmo **b&b** é finalizado, verificar se existe um nó, ainda não seqüenciado no nível um, que abra menos pilhas que a melhor solução encontrada até o momento. Caso encontre, seqüenciar o nó, independente se outros estejam sendo executados. Senão espere os outros processos terminarem e pare.

Com essa política espera-se um término de processamento mais rápido do que a primeira, pois provavelmente um número maior de nós serão podados. Embora esse procedimento pareça ser mais eficiente, o aumento de comunicação pode torná-lo bastante lento. Teoricamente, nos casos onde uma busca seqüencial gulosa encontra rapidamente a solução ótima, mas demora para comprová-la (percorrer os outros caminhos para verificar se não existe uma solução melhor), esse procedimento pode ser bastante eficiente, pois as podas podem ser realizadas concorrentemente.

Outras variações em torno dessas políticas básicas deverão também ser testadas.

Considerações Complementares

Nas implementações paralelas propostas far-se-á o gerenciamento de informações entre processos, ou seja, far-se-á o controle do algoritmo. A principal preocupação na paralelização é a manipulação do código seqüencial buscando uma boa relação entre balanceamento de carga entre os processos e minimização da comunicação.

Reconhecimento

Este trabalho é parcialmente financiado pela CAPES, FAPESP e CNPq.

Referências Bibliográficas

[1] Foster, I. ***Designing and Building Parallel Programs***. Addison-Wesley, 1995.

[2] Group, W.; Lusk E.; Thakur R. ***Using MPI-2: Advanced Features of the Message-Passing Interface***. The MIT Press, 1999.

[3] Limeira, M. S. ***Desenvolvimento de um Algoritmo Exato para a Solução de um Problema de Seqüenciamento de Padrões de Corte***. São José dos Campos. Dissertação (Mestrado em Computação Aplicada) – LAC, Instituto Nacional de Pesquisas Espaciais, 1998.

[4] Linhares, A.; Yanasse, H. H. ***Connections Between Cutting-Pattern Sequencing, VLSI design, and Flexible Machines***. A ser publicado na revista Computers & Operations Research.

[5] Pacheco, P. ***Parallel Programming with MPI***. Morgan Kaufmann Publishers, 1996.

[6] Robbins, K. A.; Robbins, S. ***Practical Unix Programming: A Guide to Concurrency, Communication and Multithreading***. Prentice-Hall, 1996.

[7] Yanasse, H. H. ***On a Pattern Sequencing Problem to Minimize the Maximum Number of Open Stack Problem***. European Journal of Operational Research 100, pg. 454-463, 1997.