Implementação Paralela de um Algoritmo para a Resolução de um Problema de Sequenciamento de Padrões de Corte: Resultados Parciais

Daniel Merli Lamosa
LAC/INPE
lamosa@lac.inpe.br

Horacio Hideki Yanasse
LAC/INPE
horacio@lac.inpe.br

Airam Jônatas Preto
LAC/INPE
airam@lac.inpe.br

Resumo

Neste trabalho foi feita uma implementação paralela de um algoritmo branch-and-bound, sugerido anteriormente na literatura, para a resolução do problema de sequenciamento de padrões de corte com o objetivo de minimizar o número máximo de pilhas abertas. Alguns testes computacionais foram realizados para se comparar às políticas de paralelismo adotadas visando um bom balanceamento de carga entre os processadores. 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.

1. Introdução

Problemas de corte em estoque apareceram em importantes situações práticas. Esses problemas consistem em cortar peças maiores em peças menores (itens) com dimensões e quantidades específicas. A solução desse problema gera os padrões de corte (como as peças maiores foram cortadas).

Associado ao problema de corte podemos ter um problema de sequenciamento de padrões visando alcançar um objetivo específico como, por exemplo, a minimização do número máximo de pilhas abertas (denotado por **MOSP,** acrônimo de *Minimization of Open Stack Problem*), a minimização do espalhamento de ordens, a minimização do número de descontinuidades, etc.

No caso do **MOSP** que é o problema focalizado nesse trabalho, deseja-se encontrar uma seqüência de corte que resulte num número mínimo de pilhas abertas simultâneas. Cada tamanho de item diferente a ser cortado, abre uma pilha nova que permanece aberta até que o último item daquele tamanho seja cortado.

Para tentar resolver o **MOSP** foi desenvolvida uma implementação paralela do algoritmo *branch-and-bound* (doravante denominado **b&b**) proposto por Yanasse [8].

A motivação para desenvolver essa implementação se dá pela dificuldade em se resolver exemplos reais este problema. De fato o **MOSP** é NP-Árduo, cf. [5]. Os algoritmos exatos encontrados na literatura (por exemplo, o trabalho de Limeira [3]), não conseguem ainda resolver exemplos de tamanho moderado, de maneira exata, em um tempo computacional satisfatório.

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 [5]). O **GMLP** consiste de um conjunto de *gates* (fios verticais) com transistores (pontos) que são utilizados para conectar os *gates*. O objetivo nesse problema consiste em diminuir a área do circuito que será impresso. Portanto, a resolução desse problema de maneira exata é bastante desejada.

Na próxima seção damos uma breve descrição do algoritmo **b&b** de Yanasse [8].

2. Algoritmo *branch-and-bound*

No algoritmo exato de Yanasse [8] 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 sequenciamento de padrões (primeiro nível da árvore); 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.* [8]. Em cada nó ramificado i identifica-se um conjunto dos itens abertos S_O^i 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 $| S_O^i |$ do nível mais elevado corrente para a próxima verificação. Com isso

obtêm-se, rapidamente, uma solução para o problema. Essa solução fornece um primeiro bom limitante para a "poda" de nós ainda não pesquisados. Um nó cujo $|S_O^i|$ é maior ou igual que o valor da melhor solução corrente não precisa ser pesquisado.

3. Implementação Paralela

Para iniciar a implementação paralela é necessário conhecer a estrutura computacional que será utilizada, pois 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 computadores de vonNeumann interconectados em rede formando uma máquina paralela com memória distribuída. Cada computador executa seu próprio programa e esse pode acessar a sua 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 de outro computador (conceito de memória distribuída). No nosso caso específico, o multicomputador utilizado consiste de uma arquitetura paralela baseada em processadores AMD compatíveis com a arquitetura IA-32 interconectados por uma rede padrão *FastEthernet* com o sistema operacional Linux.

A ferramenta (*software*) utilizada foi à biblioteca de comunicação de dados **MPI**, do acrônimo de *Message-Passing Interface*. Maiores detalhes sobre os conceitos de *hardware* e *software* podem ser encontrados em: [2], [3], [6] e [7].

Foi realizada uma implementação (versão 1) que consiste em dividir, a partir do primeiro nível da árvore, um conjunto de ramos para cada processador e cada um deles implementa um algoritmo **b&b** seqüencial [7] para encontrar a melhor solução dentre os ramos designados a ele. Após o término do processamento cada nó envia sua solução para um nó receptor que compara as soluções locais e seleciona a melhor delas (melhor solução global). Podemos resumir esta implementação como se segue:

1. Dividir, a partir do primeiro nível, os nós (itens completados) proporcionalmente entre os computadores da seguinte forma: até $\lceil m/K \rceil$ nós por computador, no qual m = número de itens e K = número de computadores. Caso $m < K$ utilizam-se m computadores.
2. Cada computador executa um algoritmo **b&b** seqüencial para seu conjunto de nós.
3. Elege-se um receptor, convencionou-se o primeiro, que consiste em um computador para receber as soluções dos demais nós.
4. Receptor seleciona a melhor solução global.

Também foi implementada outra versão (versão 2) com o objetivo de acelerar o processo de busca e conseqüentemente diminuir o tempo de execução da implementação paralela. Nessa outra versão encontra-se uma solução inicial que é "compartilhada" (na realidade cada computador calcula essa solução) entre os computadores para tentar aumentar as podas nos ramos da árvore de busca.

Foi escolhido o mesmo critério guloso do **b&b** seqüencial para determinar essa solução por ser rápido e obter boas soluções. A implementação com essa modificação pode ser resumida da seguinte forma:

1. Utilizar o critério guloso para encontrar uma solução inicial.
2. Dividir, a partir do primeiro nível, os nós (itens completados) proporcionalmente entre os computadores da seguinte forma: até $\lceil m/K \rceil$ nós por computador, no qual m = número de itens e K = número de computadores. Caso $m < K$ utilizam-se m computadores.
3. Cada computador executa um algoritmo **b&b** seqüencial para seu conjunto de nós.
4. Elege-se um receptor, convencionou-se o primeiro, que consiste em um computador para receber as soluções dos demais nós.
5. Receptor seleciona a melhor solução global.

4. Resultados Computacionais

Pela dificuldade de se obter dados reais para serem testados foi utilizado um gerador de casos sugerido em Becceneri [1].

Esse gerador consiste em criar um grafo completo e retirar arcos de forma aleatória. Pode-se utilizar esse tipo de gerador, pois o problema de sequenciamento de padrões de corte pode ser modelado como um problema em grafos. Ver, por exemplo, Becceneri [1] ou Limeira [4].

Foram geradas diversas classes de testes, porém apenas duas delas serão apresentadas por serem representativas dos demais.

A primeira classe de testes consiste em 10 exemplos de grafos g6070 que consiste de grafos com 60 nós (itens) e 70% de arcos (padrões) retirados e a segunda classe é constituída de 10 exemplos de grafos de g17025 que consistem de grafos de 170 nós e 25% dos arcos retirados.

Para cada classe de problemas foi executado um algoritmo **b&b** seqüencial para comparar os tempos obtidos com o das implementações paralelas. Cada implementação paralela foi executada com 8, 4 e 2 nós da máquina paralela.

Duas tabelas foram geradas para apresentar os resultados computacionais de cada classe. Cada tabela contém os dados da versão paralela empregada, **v1** = versão1 e **v2** = versão2. Os tempos são representados em segundos.

Tabela 1: Caso g6070 v1

Ex.	Seq.	8c	4c	2c	npa
01	***12.731***	12.766	14.352	24.381	38
02	***3.406***	21.965	13.925	6.065	39
03	15.767	***9.909***	17.489	31.028	39
04	***1.200***	2.833	1.566	2.763	38
05	44.870	***23.612***	46.407	80.724	40
06	2.600	***1.794***	2.824	4.709	39
07	6.982	***3.944***	*6.594*	12.803	39
08	6.338	***5.567***	7.762	11.006	40
09	***4.351***	6.079	8.572	11.435	39
10	***2.965***	13.925	3.985	6.476	38

Tabela 2: Caso g6070 v2

Ex.	Seq.	8c	4c	2c	npa
01	12.731	***12.675***	14.236	24.239	38
02	3.406	***1.920***	*3.401*	6.030	39
03	15.767	***9.842***	17.406	30.815	39
04	***1.200***	2.822	1.557	2.747	38
05	44.870	***23.456***	46.050	79.675	40
06	2.600	***1.772***	2.821	4.667	39
07	6.982	***3.932***	*6.566*	12.680	39
08	6.338	***5.531***	7.728	10.918	40
09	***4.351***	6.068	8.507	11.332	39
10	***2.965***	13.840	3.956	6.417	38

Tabela 3: Caso g17025 v1

Ex.	Seq.	8c	4c	2c	npa
01	10.978	***5.899***	*8.381*	11.814	160
02	10.238	***5.220***	*7.722*	11.320	160
03	10.223	***4.968***	*7.344*	10.665	160
04	11.664	***6.629***	*9.435*	12.709	160
05	7.069	***3.794***	*5.612*	8.019	159
06	9.536	***4.417***	*6.607*	10.008	160
07	10.824	***6.316***	*8.862*	12.562	160
08	8.462	***3.963***	*5.736*	9.041	160
09	9.837	***4.529***	*6.969*	9.876	160
10	12.395	***6.965***	*10.085*	14.315	160

Tabela 4: Caso g17025 v2

Ex.	Seq.	8c	4c	2c	npa
01	10.978	***5.182***	*7.412*	*10.529*	160
02	10.238	***4.571***	*7.383*	*10.045*	160
03	10.223	***4.370***	*6.527*	*9.420*	160
04	11.664	***5.823***	*8.323*	*11.241*	160
05	7.069	***3.328***	*4.959*	7.105	159
06	9.536	***3.872***	*5.842*	*8.824*	160
07	10.824	***5.549***	*7.875*	*10.045*	160
08	8.462	***3.483***	*5.042*	*7.952*	160
09	9.837	***3.956***	*6.115*	*8.737*	160
10	12.395	***6.149***	*8.975*	12.560	160

Nas Tabelas 1 e 2 temos os resultados do caso g6070. Podemos notar na tabela 1 que não houve um ganho significativo (maioria das vezes o tempo foi até maior) com a simples divisão dos ramos entre os computadores. Nessa situação ocorre que dentre os ramos alocados para um certo computador não se encontra uma boa solução e, conseqüentemente, a poda é realizada com menos freqüência. Assim, um número maior de caminhos é percorrido requerendo um tempo maior para se efetuar a enumeração. A partir das tabelas 1 e 2 pode ser observado que a versão 2 foi sempre mais eficiente que a versão 1. Comparando-se os tempos da versão paralela com o da versão seqüencial na tabela 2 percebe-se que o ganho de tempo da implementação paralela se dá apenas a partir do uso de 8 computadores, mesmo assim nem todos os casos foram resolvidos num tempo menor.

Nas Tabelas 3 e 4 temos os resultados do caso g17025. Essa classe de exemplos é de fácil resolução pelo grafo gerado ser mais denso.

Os tempos de processamento das implementações paralelas são, na maioria das vezes, menores o que pode caracterizar um bom balanceamento de carga entre os computadores.

5. Conclusão

Novas versões do código serão implementadas, onde será trocada uma quantidade maior de informações entre processadores. Espera-se que essas novas versões produzam um resultado melhor devido a uma maior troca de soluções parciais obtidas nos ramos processados em paralelo.

Assim ainda não foi possível estabelecer uma política de balanceamento de carga satisfatória de modo que a paralelização desse método se torne mais eficiente.

Reconhecimento: Este trabalho foi parcialmente financiado pela FAPESP, CNPq e CAPES.

6. Referências

[1] Becceneri, J. C. "O Problema de Sequenciamento de Padrões para a Minimização do Número Máximo de Pilhas Abertas em Ambientes de Cortes Industriais". São José dos Campos. Tese (Doutorado em Ciências no Curso de Engenharia Eletrônica e Computação na Área de Informática) – ITA, *Centro Tecnológico Aeroespacial*, 1999.

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

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

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

[5] Linhares, A.; Yanasse, H. H. "Connections Between Cutting-Pattern Sequencing, VLSI design, and Flexible Machines". *Computers & Operations Research* **29**(12), pg. 1759-1777, October 2002.

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

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

[8] 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.