Algoritmo *Branch-and-Bound* Distribuído e Tolerante a Falhas para Grades Computacionais

Alexandre D. Gonçalves[1], Lúcia M. A. Drummond[1], Eduardo Uchoa[2] e

M. Clicia S. de Castro[3]

[1]Departamento de Ciência da Computação – Universidade Federal Fluminense (UFF)
[2]Departamento de Engenharia de Produção – Universidade Federal Fluminense (UFF)
Passo da Pátria 156 – 24.210-240 – Niterói – RJ – Brasil

[3]Instituto de Matemática e Estatística – Universidade do Estado do Rio de Janeiro
São Francisco Xavier 524 – 20.550-013 – Rio de Janeiro – RJ – Brasil

{agoncalves,lucia}@ic.uff.br,uchoa@producao.uff.br,clicia@ime.uerj.br

Abstract. *This work introduces a new fault tolerant and distributed branch-and-bound algorithm applied to the Steiner Problem in Graphs (SPG), to be run on computational Grids. Many Grids are composed of clusters of processors connected via high-speed links and the clusters, geographically distant, are connected through low-speed links, in a hierarchical fashion. The algorithm proposed has the following features: i) it does not employ the usual master-worker paradigm; ii) it considers the hierarchical structure of such Grids in its procedures; and iii) it contains load balance and fault tolerance mechanisms. Good speepuds were obtained, allowing the resolution of hard instances in very reasonable times.*

Resumo. *Este trabalho apresenta um novo algoritmo branch-and-bound distribuído e tolerante a falhas, aplicado ao Problema de Steiner em Grafos, para grades computacionais. Muitas grades são compostas por clusters de processadores conectados por canais de baixa velocidade e os clusters, geograficamente distantes, são conectados através de canais de alta velocidade, de modo hierárquico. O algoritmo proposto tem as seguintes características: i) não emprega o paradigma usual mestre-escravo; ii) considera a estrutura hierárquica das grades nos seus procedimentos; e iii) contém mecanismos de balanceamento de carga e tolerância a falhas. Bons speedups foram obtidos, permitindo a solução de instâncias difíceis em tempos muito satisfatórios.*

1. Introdução

A técnica *branch-and-bound* tem sido uma das mais utilizadas para encontrar a solução ótima de problemas de otimização NP-difíceis. Os algoritmos *branch-and-bound* percorrem o espaço de soluções através de árvores de enumeração. Eles possuem um grande potencial de paralelização porque as computações das sub-árvores podem ser realizadas de modo quase independente.

Este trabalho propõe um novo algoritmo *branch-and-bound* distribuído e tolerante a falhas, aplicado ao Problema de Steiner em Grafos (SPG), para grades

computacionais. Em geral, as grades computacionais são compostas por diversos *clusters* de processadores, geograficamente distantes, conectadas por canais de baixa velocidade. Entretanto, existe uma estrutura hierárquica nas grades, considerando que a comunicação entre processadores num mesmo *cluster* é muito mais rápida do que entre processadores de diferentes *clusters*. Dessa forma, propomos um algoritmo *branch-and-bound* distribuído cujos procedimentos exploram esta estrutura hierárquica das grades.

Vários trabalhos recentes abordam algoritmos *branch-and-bound* paralelos para ambientes em grade [Roucairol *et al.* 2000]. Muitos deles adotam uma abordagem centralizada, na qual um único processador mantém a árvore e envia tarefas para os demais processos. Segundo Aida *et al* (2003) esta estratégia não é escalável e nem conveniente para ambientes em grade. Pode ocorrer uma degradação significativa no desempenho por causa do alto *overhead* de comunicação entre os processos. Eles propõem um algoritmo *branch-and-bound* distribuído, baseado em um paradigma mestre-escravo hierárquico. Um processo supervisor controla a ativação de múltiplos processos, cada um dos quais é composto de um mestre e vários escravos. Essa estratégia não é completamente distribuída e não aborda tolerância a falhas.

O controle central pode ser também um obstáculo à tolerância a falhas. Nesse contexto, Iamnitch e Foster (2000) propõem um algoritmo *branch-and-bound* completamente distribuído, e com mecanismos de tolerância a falhas. Esse algoritmo é baseado em tabelas mantidas em todos os processadores, contendo informações de sub-árvores resolvidas, e em mensagens usadas para propagar as tabelas. A recuperação de falhas é atingida quando um processo sem trabalho verifica sua tabela local e escolhe uma sub-árvore não resolvida para solucioná-la. Essa solução pode levar a muito trabalho redundante, ainda que não ocorram falhas.

Resumindo, nosso algoritmo difere dos anteriores nos seguintes pontos: i) não utiliza o paradigma usual mestre-escravo; ii) considera a estrutura hierárquica das grades nos seus procedimentos principais; e iii) inclui mecanismos de balanceamento de carga e tolerância a falhas. Desenvolvemos um algoritmo distribuído baseado em um algoritmo *branch-and-bound* seqüencial, já existente para SPG, que é um problema NP-difícil clássico. De um conjunto de 400 instâncias SPG difíceis, propostas por Duin (1993) como um *benchmark* e um desafio para algoritmos futuros, o algoritmo seqüencial foi capaz de solucionar quase todas as instâncias com tempos razoavelmente pequenos. As instâncias estão disponíveis no repositório SteinLib [Koch *et al* 2004], e somente 20 instâncias incidentes estão ainda em aberto. Um dos objetivos do algoritmo *branch-and-bound* distribuído é solucionar estas instâncias restantes. Este artigo está organizado da seguinte forma. As Seções 2 e 3 descrevem os algoritmos *branch-and-bound* seqüencial e distribuído, respectivamente. O ambiente experimental, os resultados preliminares e trabalhos futuros são apresentados na Seção 4.

2. Branch-and-Bound Seqüencial para SPG

Um algoritmo *branch-and-bound* soluciona um problema de otimização realizando operações de ramificação que dividem um problema em dois subproblemas similares. Cada subproblema (nó) trabalha com uma diferente parte do espaço de solução. A solução dos subproblemas, também, inclui operações de ramificação. Assim, o algoritmo induz a uma árvore de enumeração. O cálculo dos limites para o valor da

solução ótima é uma parte fundamental de um algoritmo *branch-and-bound*. Os limites são usados para limitar o crescimento da árvore.

O SPG é um dos problemas NP-difíceis mais estudados e pode ser computacionalmente muito custoso. Wong (1984) propôs um algoritmo denominado *dual ascent*, que rapidamente encontra uma solução aproximada. Soluções aproximativas, também, fornecem limites inferiores válidos. Poggi de Aragão *et al* (2001) propuseram três heurísticas adicionais para melhorar os limites do *dual ascent*, denominadas: *dual scaling*, *dual adjustment* e *active fixing by reduced costs*. O algoritmo *branch-and-bound* que utilizamos é baseado nestas heurísticas, e está completamente descrito em Uchoa (2001). Naquela época, ele pôde solucionar cerca de 60 instâncias incidentes pela primeira vez, a maior com 640 vértices e 200.000 arestas. Somente 20 instâncias incidentes da SteinLib não puderam ser solucionadas em tempo satisfatório (menos de um dia em um PC de 350 MHz). Com as máquinas atuais mais rápidas, o mesmo algoritmo poderia solucionar 5 dessas instâncias incidentes, em aberto, em no máximo uma semana de tempo de CPU.

3. Branch-and-Bound Distribuído para SPG

O algoritmo *branch-and-bound* possui um grande potencial de paralelização porque as computações das sub-árvores podem ser realizadas de modo quase independente. O algoritmo proposto é composto pelos seguintes procedimentos: distribuição inicial, balanceamento de carga, difusão do primal, detecção de terminação e tolerância a falhas, que estão descritos a seguir. Certos procedimentos necessitam de um líder em cada *cluster*. O processo líder é aquele que possui o menor número de identificação.

Note que assumimos que apenas um processo é atribuído a cada processador, então, usamos indistintamente essas duas palavras no restante do texto.

Distribuição inicial. A fase de distribuição inicial corresponde à distribuição de sub-árvores entre os clusters. Em cada operação de ramificação, um líder de cluster x envia uma mensagem com um novo nó para outro cluster. Esse cluster é definido como nextcluster = x+ 2^{level}. Onde level representa a profundidade do líder x na árvore. Quando nextcluster atinge um valor maior que o número de clusters disponíveis na grade, os processos iniciam o compartilhamento de suas cargas locais com outros processos dentro do mesmo cluster. Mais especificamente, level é reiniciado com zero, e cada processo i envia um novo nó para o processo nextproc = i + 2^{level}. Por exemplo, considere um cluster com 8 processadores. Identificamos cada processo como cl_0,...,cl_7. Inicialmente, cl_0 envia metade de sua carga local para cl_1, e continua processando a outra metade de sua árvore. No segundo nível da árvore, cl_0 compartilha sua carga agora com cl_2, enquanto cl_1 compartilha sua carga com cl_3, e assim por diante. A Figura 1 mostra a árvore resultante desta fase.

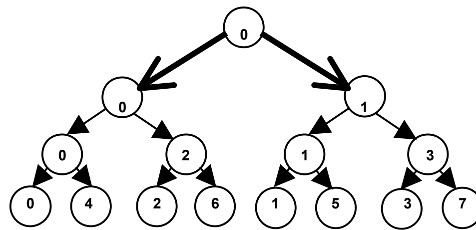

Figura 1. Exemplo de distribuição inicial de carga entre *clusters*.

Balanceamento de carga. O tempo para processar uma sub-árvore não é conhecido *a priori* e pode variar muito, o que torna a utilização de um algoritmo de balanceamento de carga fundamental para este problema. Neste trabalho implementamos três diferentes algoritmos para o balanceamento de carga, denominados: GlobalLB; LocalLB e HybriLB. No algoritmo GlobalLB, sempre que um processo se torna ocioso, este pede carga (sub-árvore) para qualquer processo, não havendo distinção entre processos de diferentes *clusters*. No algoritmo LocalLB, é considerado somente o envio de carga entre processos pertencentes ao mesmo *cluster*. HybridLB é um algoritmo híbrido que prioriza o balanceamento de carga entre processos de um mesmo *cluster*, mas permite também que haja transferência de carga entre *clusters* diferentes, quando todos os processos dentro de um *cluster* se tornam ociosos.

Difusão do primal e Detecção de terminação. Em ambos os procedimentos propomos um algoritmo hierárquico. O procedimento de difusão do primal é invocado quando um processo encontra um limite melhor do que o limite atual conhecido. Ele deve ser disseminado entre os processadores através da grade, permitindo a poda da árvore de enumeração. Um processo envia o limite do primal para os processos do *cluster* a que pertence, e o líder do *cluster* é responsável por enviar o limite do primal para os outros líderes dos *clusters* da grade, que o difundem entre seus processadores.

Quando um processo atinge sua condição de terminação local, isto é, não é capaz de obter sub-árvores dos seus vizinhos, ele informa o fato ao seu líder. Quando todos os processos daquele *cluster* atingirem a mesma condição, e o líder não for capaz de obter sub-árvores de outros líderes, considera-se que o *cluster* atingiu sua condição de terminação. Assim, o líder do *cluster* envia uma mensagem aos outros líderes indicando a condição de terminação e pode terminar (assim como os processos do seu *cluster*) após o receber a mesma mensagem de todos os outros líderes.

Tolerância a falhas. Os procedimentos de tolerância a falhas devem permitir que uma aplicação continue a sua execução mesmo na presença de falhas. Usualmente, essa meta é atingida pela utilização de técnicas de *checkpoint* e *rollback recovery* [Elnozahy *et al.*1996]. Em nosso algoritmo cada processo é executado independentemente. Por isso, optamos por *checkpoints* não coordenados, onde cada processo decide independentemente quando gravar o seu *checkpoint*. Nossa estratégia é capaz de tratar uma única falha permanente por *cluster*. Neste procedimento, um processo periodicamente envia uma mensagem contendo seu *checkpoint* para outro processo vizinho no mesmo *cluster*. Após receber uma mensagem de *checkpoint,* o processo vizinho armazena o último *checkpoint* recebido. O *checkpoint* é composto de informação referente ao último nó executado e um vetor cuja dimensão é igual ao nível

deste nó na árvore. Cada posição i do vetor indica se uma mensagem de ramificação, referente à sub-árvore de nível i, foi enviada para ser resolvida em outro processo (se não foi enviada, precisará ser resolvida futuramente). Durante a recuperação, o processo com o *checkpoint* do processo que falhou assume os serviços que lhe foram atribuídos.

A detecção de falhas é baseada no mecanismo *heartbeat*, onde processos trocam mensagens em intervalos de tempo regulares indicando que eles estão ativos [Robertson 2000].

Esta estratégia é estendida de modo a detectar falha de *cluster,* substituindo processadores por *clusters* (representados pelo seu líder), tendo *checkpoints* representando estados de *clusters* (ao invés de processos) e empregando o mecanismo de *heartbeat* entre líderes de *clusters*.

4. Resultados Preliminares e Trabalhos Futuros

Nosso algoritmo está desenvolvido em C++ e usa MPICH-G2, uma versão da biblioteca MPI para o Globus [Foster e Kessekman 1998, Snir *et a.* 1996].

Os resultados vem sendo obtidos no ambiente GridRio, que é uma iniciativa para criar uma grade computacional através de cinco universidades do Estado do Rio de Janeiro. Como a GridRio ainda não está totalmente operacional, nossos experimentos preliminares foram executados em uma pequena grade composta de *clusters* de somente duas universidades: PUC-Rio (com 16 processadores Pentium de 1.7GHz) e UFF (com 10 processadores Athlon de 1.3GHz). O algoritmo distribuído foi analisado para 5 instâncias incidentes, particularmente difíceis, onde o algoritmo seqüencial gastaria cerca de uma semana de tempo de CPU.

Para analisar o balanceamento de carga proposto, executamos nosso algoritmo, inicialmente, no *cluster* da PUC com 15 processadores, para 5 instâncias. Obtivemos os seguintes resultados (instância/valor ótimo): i640-211/11984; i640-212/11795; i640-213/11879; i6402-14/11898; e i640-215/12081. Os *speedups* médios do algoritmo foram 13.5 e 3.3, quando comparado com a versão seqüencial e a versão distribuída sem balanceamento de carga, respectivamente.

Outros testes foram realizados usando os *clusters* da PUC e da UFF. Executamos o algoritmo distribuído com as três versões de balanceamento de carga: GlobalLB, LocalLB e HybridLB. Observamos que o algoritmo LocalLB apresentou tempos de execução médios 41% piores do que GlobalLB, e o algoritmo HybridLB é ligeiramente melhor (9%) do que GlobalLB.

Os *speedups* obtidos foram bons, porém apresentaram irregularidade. Por exemplo, para a instância i640-214 o *speedup* foi quase duas vezes o número de processos. Essa anomalia nos algoritmos *branch-and-bound* paralelos são usuais [Bruin 1995, Lai e Sahni 1984]. Se uma solução é, por sorte, descoberta cedo, as sub-árvores podem ser melhor podadas e os tempos de execução são muito reduzidos. O oposto, também, é verdade.

O procedimento de tolerância a falhas tem apresentado funcionamento correto nas simulações de falhas. O custo adicional no tempo de execução da aplicação com a inclusão deste procedimento, quando não ocorrem falhas, não ultrapassou 8,3% do tempo de execução da aplicação sem a inclusão do mesmo.

Como trabalhos futuros pretendemos realizar testes mais detalhados para melhor analisar os procedimentos de balanceamento de carga e o *overhead* introduzido pelo procedimento de tolerância a falhas na ocorrência de falhas. Além disso, pretendemos estender o procedimento de tolerância a falhas para admitir duas ou mais falhas por *cluster*.

Referências

Aida, K., Natsume, W. e Futakata, Y. (2003) "Ditributed Computing with Hierarchical Master-Worker Paradigm for Parallel Branh-and-Bound Algorithms", In: Proceedings of the Third International Symposium on Cluster Computing and Grid, p. 156-162.

Bruin, A., Kindervater, G.A.P., Trienekens, H.W.J.M. (1995) "Asynchronous Parallel Branh-and-Bound and Anomalies", Erasmus University, Department of Computer Science, EUR-CS-95-05, Roterdam, Holand.

Duin, C. (1993) "Steiner´s Problems in Graphs", Ph.D. thesis, University of Amsterdam, Holand.

Iamnitch, A., Foster, I. (2000) "A Problem Specific Fault-Tolerance Mechanism for Asynchronous, Ditributed System", In: Proceedings of the International Conference on Parallel Processing, p. 4-14.

Foster, I., Kessekman, C. (1998) "The Globus Project: a Status Report", In: Proceedings of the Seventh Heterogeneous Computing Workshop (HCW´98), p. 4-18.

Koch, T., Martin, A., Voss, S. (2004) "SteinLib: An Update Library on Steiner Problems in Graphs", Konrad-Zuse-Zentrum für Informationstechnik Berlin, ZIB-Report 00-37, http://elib.zib.de/steinlib, last visited Oct 30th, 2004.

Lai, T.-H., Sahni, S. (1984) "Anomalies in Parallel Branch-and-Bound Algorithms", Communication of the ACM, 27, p. 594-602.

Poggi de Aragão, M, Uchoa, E., Wernwck, R.S. (2001) "Dual Heuristics on the Exact Solution of Large Steiner Problems", Eletronic Notes in Discrete Mathematics, 7.

Robertson, A. (2000) "Linux-HA: Heartbeat System Design", In: Proceedings of the Fourth Annual Linux Showcase and Conference (ASL).

Roucairol, C., Cung, V., Yafoufi, N. (2000) "Design, Implementation and Robustness in Parallel Branch-and-Bound", Techinical Rrepórt PRISM 2000/19, l´Université de Versailles Saint-Quentin.

Snir, M, Otto, S.M., Huss-Lederman, S., Dongarra, J., MPI: The Complete Reference, The MIT Press, 1996.

Uchoa, E. (2001) "Algoritmos para Problemas de Steiner com Aplicações em Projeto de Circuitos VLSI", Tese de Doutorado, PUC- Rio de Janeiro, Brasil.

Wong, R. (1984) "Dual Ascent Approach for Steiner Tree Problems on a Discrete Graph", Mathematical Programming, 28, p. 271-287.