Balanceamento de Carga em um algoritmo *Branch-and-Bound* para execução em *Grades* computacionais

Juliana M. Nascente[1], Lúcia M. A. Drummond[1], Eduardo Uchoa[2]
[1]*Departamento de Ciência da Computação – UFF*
[2]*Departamento de Engenharia de Produção – UFF*
{jsilva,lucia}@ic.uff.br,uchoa@produção.uff.br

Abstract

This work introduces new techniques of load balance for a distributed branch-and-bound algorithm, applied to the Steiner Problem in Graphs (SPG), to be executed on computational Grids. Many Grids are composed of cluster of processors connected via high-speed links and the clusters, geographically distant, are connected through low-speed links, in a hierarchical fashion. In Grids, the processor performance may vary a lot during a certain period of time due to the fact that they are usually shared with several other applications. The proposed load balance algorithms have the following features: i) they do not employ the usual master-worker paradigm; ii) they consider the hierarchical structure of such Grids and the processors performance iii) they estimate the future load of processes. Several experiments were executed showing the efficiency of the proposed algorithms.

1. Introdução

Branch-and-bound (B&B) é uma das técnicas 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 e possuem um grande potencial de paralelização porque as computações das subárvores podem ser realizadas de modo quase independente.

Drummond *et al.* em [14] propuseram um algoritmo *branch-and-bound* distribuído, aplicado ao Problema de *Steiner* em Grafos (SPG) para Grades computacionais. A versão seqüencial deste algoritmo, de um conjunto de 400 instâncias do SPG difíceis, propostas por Duin [3] como um *benchmark* e um desafio para algoritmos futuros, foi capaz de solucionar quase todas as instâncias com tempos razoavelmente pequenos. As instâncias estão disponíveis no repositório SteinLib [7]. Na versão distribuída deste trabalho [14], para conseguir um bom aproveitamento da capacidade de processamento da Grade, foram desenvolvidas duas estratégias de balanceamento de carga denominadas *Global* e *Híbrida*. Apesar dos bons resultados apresentados pelas duas estratégias, verificamos que a quantidade de subárvores transferidas de um processo sobrecarregado para outro subcarregado é pequena e fixa, independente das condições dos processadores, possibilitando, freqüentemente, que os processos se tornem ociosos rapidamente e diminuindo a eficiência paralela do algoritmo.

Neste trabalho, desenvolvemos duas novas estratégias baseadas nos algoritmos originalmente propostos em [14], que utilizam informações do próprio problema e dos recursos do ambiente para alcançar um melhor balanceamento de carga. As novas estratégias foram denominadas *GlobalAdapt* e *HibridaAdapt*.

Vários trabalhos recentes abordam algoritmos B&B paralelos para Grades [10]. Muitos deles adotam uma abordagem centralizada, na qual um único processo, chamado mestre, mantém a árvore de tarefas a serem resolvidas e as envia para os demais processos quando estes se tornam subcarregados [15] [16].

Roucairol *et al* em [16] propuseram três estratégias de balanceamento. Em tais abordagens, cada processo mantém uma fila de subproblemas classificados de acordo com a sua potencialidade, definida como a estimativa da capacidade de um subproblema gerar trabalho, e a distância entre o subproblema e a solução ótima. Esta fila é resolvida utilizando a técnica "*melhor primeiro*". O balanceamento de carga é centralizado, onde existe um processo mestre responsável por determinar a quantidade de subproblemas que serão transferidos de um processo sobrecarregado para outro subcarregado. A diferença entre as três estratégias consiste na variação da quantidade de subproblemas que serão enviados e na

posição em que são retirados da fila, ou seja, início ou fim da mesma. Nenhuma característica do ambiente utilizado é explorada.

Segundo Aida *et al* [1] a abordagem centralizada não é escalável e conveniente para ambientes de Grades, podendo ocorrer uma degradação significativa no desempenho devido ao alto *overhead* de comunicação entre os processos escravos e o mestre. Assim, propuseram um algoritmo B&B distribuído, baseado em um paradigma mestre-escravo hierárquico, onde um processo supervisor controla a ativação de um conjunto de processos formado por um processo mestre e vários escravos. Entretanto, esta estratégia não é completamente distribuída ficando a cargo do processo mestre toda a distribuição e redistribuição de carga.

As novas estratégias de balanceamento de carga que propomos diferem das anteriores nos seguintes pontos: i) não utilizam o paradigma usual mestre-escravo; ii) para fins de redução do custo de comunicação entre os processos, consideram que grades computacionais geralmente são compostas por diversos *clusters* de processadores, geograficamente distantes, conectados por canais de baixa velocidade, enquanto a comunicação entre processadores pertencentes a um mesmo *cluster* é muito mais rápida; e iii) utilizam informação sobre o desempenho dos processadores e sobre a capacidade de ramificação dos nós das subárvores para definir a quantidade de carga a ser transferida.

Comparamos as estratégias desenvolvidas, *GlobalAdapt* e *HibridaAdapt*, com as propostas em [14] e analisamos os resultados avaliando: i) a ociosidade dos processos durante a aplicação, ou seja, verificamos se a nova quantidade de subárvores enviadas durante uma transferência foi suficiente para evitar que os processos se tornassem ociosos rapidamente; ii) o número de mensagens trocadas entre os processos; iii) o tempo de execução da aplicação. Nossas estratégias apresentaram os melhores resultados, uma vez que reduziram o tempo de ociosidade dos processos e o número de mensagens enviadas durante a aplicação. Diminuíram, também, o tempo de relógio de parede gasto pela aplicação para todas as instâncias.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 descreve o algoritmo B&B seqüencial. Na Seção 3 estão descritos os procedimentos utilizados no desenvolvimento do algoritmo distribuído, bem como seu funcionamento. Na Seção 4 apresentamos os procedimentos de balanceamento de carga propostos, enquanto os resultados preliminares e trabalhos futuros são apresentados nas Seções 5 e 6, respectivamente.

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 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, chamados primal e dual, para o valor da solução ótima é um aspecto fundamental da estratégia *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 em [13] propôs um algoritmo denominado *dual ascent* que rapidamente encontra uma solução aproximada. Soluções aproximativas, também, fornecem limites inferiores válidos. Poggi de Aragão *et al* [9] 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 [12], que solucionou cerca de 60 instâncias de incidência pela primeira vez, a maior com 640 vértices e 200.000 arestas.

3. *Branch-and-Bound* Distribuído para SPG

O algoritmo *branch-and-bound* possui um grande potencial de paralelização porque as computações das subárvores podem ser realizadas de modo quase independente. A única informação necessária trocada entre os processos é o valor do primal.

O algoritmo distribuído proposto em [14] é composto pelos seguintes procedimentos: distribuição inicial, balanceamento de carga, difusão do primal e detecção de terminação.

Antes de descrever os procedimentos, para melhor entendimento, são necessárias algumas considerações:

- ✓ Cada processo está alocado a um processador diferente e recebe uma identificação *i* pertencente ao intervalo de [0 ; *m-1*], onde *m* é a quantidade de processos que participam da aplicação.
- ✓ Processos que pertencem a um mesmo *cluster* possuem identificações consecutivas.

- ✓ Para cada *cluster*, existe um processo líder (processo de menor identificação) denominado $l(C_j)$, onde j é a identificação do *cluster*. A atuação deste líder não deve ser confundida com a de um mestre no paradigma mestre-escravo. O líder possui as mesmas funções que qualquer outro processo da aplicação e não interfere na decisão da quantidade de carga a ser enviada aos processos, que por sua vez, não o identificam como um coordenador do balanceamento de carga.

3.1. Distribuição inicial

A fase de distribuição inicial corresponde à distribuição de subárvores entre os *clusters*. O processo de identificação igual 0 (líder da aplicação) inicia a execução da primeira subárvore. A cada operação de ramificação, envia uma mensagem que contém uma nova subárvore para o *cluster* *nextcluster* = $i + 2^{level}$, onde *level* representa a profundidade do líder i na árvore. Este mesmo procedimento é realizado por todos os outros líderes de *clusters* após terem recebido a primeira ramificação. Quando *nextcluster* atinge um valor maior que o número de *clusters* disponíveis na Grade, os processos iniciam o compartilhamento local de carga, ou seja, compartilham sua carga com os processos pertencentes a seu próprio *cluster*. Isto implica que *level* será reiniciado com zero, e cada processo i enviará uma nova subárvore ao processo *nextproc* = $i + 2^{level}$. Por exemplo, considere uma Grade composta por 8 processos divididos em 3 *clusters*, $C_0\{0, 1, 2 \text{ e } 3\}$, $C_1\{4, 5\}$ e $C_2\{7, 8\}$. Inicialmente, o processo líder do *cluster* 0, $l(C_0)$, envia uma ramificação (mensagem contendo uma subárvore) para o processo $l(C_1)$ e este envia a primeira ramificação para $l(C_2)$. Como não existem mais *clusters* a serem iniciados, $l(C_0)$ iniciará a distribuição local de carga, onde o processo 0 enviará carga para os processos 1 e 2, o processo 1 para o 3, e assim por diante. A Figura 1 ilustra a distribuição entre *clusters*.

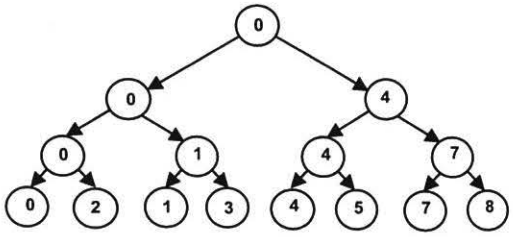

Figura 1. Exemplo de distribuição de carga entre *clusters*

3.2. Difusão do primal

A solução de uma subárvore é o valor encontrado para a função objetivo do SPG quando o limite inferior, dual, ultrapassa o valor do limite superior, primal, no espaço de busca da solução. Durante a execução da aplicação o procedimento de difusão do primal é invocado quando um processo encontra um limite melhor do que o limite atual conhecido, devendo este novo limite ser disseminado entre os processadores através da Grade, para que seja delimitado um novo espaço de busca da solução. Este procedimento é feito de forma hierárquica. Um processo envia o novo primal para os processos do *cluster* a que pertence, e o líder do *cluster* é responsável por enviá-lo aos outros líderes dos *clusters* da Grade, que o difundem entre seus processadores. A Figura 2 ilustra melhor este procedimento.

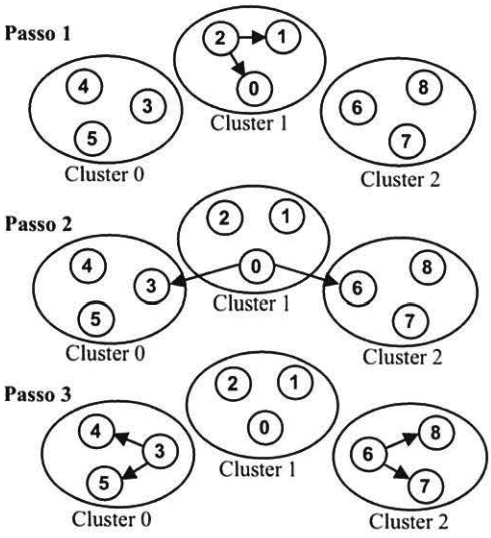

Figura 2. Difusão do primal

No Passo 1 o processo 2, pertencente ao *cluster* 0, inicia a difusão do primal enviando-o aos processos de seu próprio *cluster*. Ao receber o novo primal, o processo líder informa aos líderes dos *clusters* 1 e 2, mostrado no Passo 2. Finalmente, no Passo 3, os líderes 3 e 6 enviam o limite aos processos pertencentes a seus *clusters* respectivamente.

3.3. Balanceamento de carga

Utilizar estratégia de balanceamento de carga é fundamental para melhorar a eficiência paralela da aplicação. São muitos os fatores que provocam o desbalanceamento de carga. Grades computacionais possuem como principal característica a heterogeneidade dos processadores que as compõem. Além disto, cada processador está sujeito a uma quantidade de carga externa desconhecida *a priori*, assim, o tempo para processar uma subárvore varia muito em função do desempenho do processador no qual o processo está alocado. A profundidade alcançada por uma subárvore é imprevisível, visto que as podas nas mesmas acontecem quando um novo primal é encontrado. Portanto, não é difícil que existam ao mesmo tempo, durante a execução da aplicação, processos ociosos e sobrecarregados, fazendo com que o processamento oferecido pelo ambiente não seja aproveitado em sua totalidade. Podemos comprovar que a utilização de uma estratégia de balanceamento de carga é fundamental para aumentar a eficiência paralela do algoritmo através da Tabela 1, que mostra os tempos de execução (tempo de relógio de parede) do algoritmo seqüencial e de duas versões paralelas. Na primeira coluna (Instâncias) são apresentadas as instâncias utilizadas. Na segunda coluna (SEQ) os tempos (em horas) do algoritmo seqüencial proposto por [12], na terceira (PAR1) e quarta coluna (PAR2) os tempos de execução das versões paralelas sem a utilização de qualquer estratégia de balanceamento de carga e utilizando a estratégia de balanceamento *Global*, respectivamente, conforme apresentado em [17]. As duas últimas colunas (SP1 e SP2), apresentam os *speedups* das versões paralelas. Estes testes foram executados em um *cluster* com 16 processadores Pentium 2.4 GHz exclusivamente dedicados para a aplicação. O algoritmo seqüencial, por tomar decisões aleatórias, foi executado cinco vezes com diferentes sementes e o resultado apresentado é a média dos tempos de relógio apresentados por ele.

Instância	*SEQ*	*PAR1*	*PAR2*	*SP1*	*SP2*
i640-211	72,14	14,08	3,68	5,12	19,58
i640-212	5,75	0,73	0,15	7,92	39,36
i640-213	5,16	0,83	0,34	6,23	15,07
i640-214	52,02	7,90	1,51	6,58	34,47
i640-215	21,98	4,73	1,18	4,65	18,69

Tabela 1. Tempo de execução

Drummond *et al.* em [14] desenvolveram dois procedimentos para balanceamento de carga. O *Global*, que não faz distinção entre transferências de carga entre processos de um mesmo *cluster* ou de *clusters* diferentes, e o *Híbrido*, que prioriza o balanceamento de carga entre processos de um mesmo *cluster*, mas permite também que ocorra transferência de carga entre *clusters* diferentes, quando todos os processos dentro do *cluster* tornam-se ociosos. Entretanto, apesar destas estratégias terem obtido bons resultados em termos de tempo de relógio de parede, algumas deficiências foram observadas. Em ambas as estratégias, a carga transferida entre processos consiste sempre de uma única subárvore, o que pode se mostrar inadequado considerando-se as possíveis diferenças nos desempenhos dos processadores. Em especial, na versão *Híbrida*, quando a transferência é realizada entre *clusters*, o líder irá executar a subárvore recebida e a partir das suas ramificações os outros processos do *cluster* receberão carga, como acontece no Procedimento de Distribuição Inicial. Isto freqüentemente faz com que estes processos se tornem ociosos muito mais rapidamente.

Propomos, então, adaptações nestas estratégias com objetivo de melhorar a eficiência paralela do algoritmo que estão detalhadamente descritas na Seção 4.

3.4. Detecção de terminação

Este procedimento, como o de difusão do primal, também acontece de forma hierárquica. Na estratégia *Híbrida*, quando um processo atinge sua condição de terminação local, isto é, não é capaz de obter subárvores com os processos pertencentes a seu *cluster*, informa este fato ao seu líder. Quando todos os processos deste *cluster* atingirem a mesma condição e o líder não for capaz de obter subárvores 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 que alcançou a sua condição de terminação e pode terminar (assim como os processos do seu *cluster*) após o receber a mesma mensagem de todos os outros líderes. Na estratégia *Global*, um processo quando não consegue carga após um número p de pedidos, informa aos demais que atingiu sua condição de terminação e termina de fato, quando todos os demais processos atingem a mesma condição.

4. Balanceamento de Carga Adaptativo

Considerando os resultados obtidos pelos procedimentos de balanceamento de carga descritos

em [14], desenvolvemos adaptações para estas técnicas. As novas estratégias *GlobalAdapt e HibridaAdapt* são baseadas nas técnicas *Global* e *Hibrida*, respectivamente. Na estratégia Híbrida, um processo quando se torna subcarregado envia pedido de carga a um processo pertencente a seu próprio *cluster*. Se este não possuir carga, envia novamente o pedido, mas a outro vizinho, e assim sucessivamente na tentativa de obter carga. Caso todos os processos de seu *cluster* estiverem ociosos envia ao líder uma mensagem indicando o ocorrido. O líder tendo conhecimento que não mais existe carga a ser resolvida em seu *cluster*, envia pedido de carga ao *cluster* vizinho. Se receber carga como resposta ao pedido, o líder, redistribuirá a carga entre os processos de seu *cluster*, caso contrário, inicia a terminação do algoritmo. Na estratégia *Global*, a transferência de carga ocorre independente da distância geográfica dos processos envolvidos no balanceamento. Um processo quando se torna subcarregado envia pedido de carga a qualquer processo participante da aplicação.

Nas estratégias adaptativas aqui propostas quando um processo se torna ocioso envia, junto com a mensagem de requisição de carga, a informação relativa ao desempenho do processador no qual está alocado. O processo que receber esta requisição irá dividir proporcionalmente as subárvores ainda não resolvidas com o processo que enviou o pedido. As subárvores enviadas são as que estimativamente mais se ramificarão, evitando que o processo se torne ocioso rapidamente.

Note que na distribuição inicial, apenas uma única subárvore é enviada e durante o balanceamento de carga, a quantidade de subárvores transferida varia de acordo com o desempenho dos processos envolvidos.

Abaixo estão detalhadamente descritos os procedimentos utilizados por um processo durante o balanceamento para analisar o desempenho dos processadores e estimar o tamanho das subárvores.

4.1. Avaliação do desempenho dos processadores

O ambiente de Grades é heterogêneo, logo, não é correto assumir que todos os processadores possuem o mesmo poder computacional. Além disso, já que o ambiente, normalmente, não é dedicado, a aplicação disputa o processamento com processos de outros usuários. Portanto, é necessário, em tempo de execução, adaptar a demanda da aplicação (carga pendente) aos recursos disponíveis no ambiente.

A eficiência de um balanceamento de carga é inversamente proporcional ao tempo total em que os processos ficam ociosos durante a aplicação. Se durante a transferência não é enviada uma quantidade ideal de carga, ou seja, uma quantidade de carga compatível com o poder computacional do processador onde o processo está alocado, existirão processos sobrecarregados e subcarregados ao mesmo tempo durante a execução, o que resultará em um baixo desempenho da aplicação.

Para avaliar o desempenho dos processadores a seguinte estratégia foi adotada. Quando um processo i se torna ocioso, envia uma mensagem LOADREQUEST para o processo j, sendo este $j = i - 1$, com um valor correspondente à razão entre o número de nós resolvidos no período de tempo t, e o próprio tempo t, chamado M_i. Vale observar que t é o intervalo de tempo de relógio de parede iniciado a partir do recebimento de uma carga até o final da execução da mesma.

O processo j ao receber a mensagem, calcula M_j, de forma análoga ao processo i. A análise destas informações, M_i e M_j, permitirá ao processo j apontar qual dos processadores tem apresentado melhor desempenho e dividir sua carga proporcionalmente aos desempenhos apresentados pelos processadores em questão. Assim, considerando que a quantidade de subárvores ainda não resolvidas por j é QS_j, chamada de carga acumulada, j responderá à mensagem enviada por i com uma mensagem BRANCH contendo NN subárvores, onde $NN = \lfloor Mi * (QSj/Mj + Mi) \rfloor$. Desse modo a carga pendente estará sendo adaptada automaticamente aos recursos disponíveis do ambiente durante a execução.

As NN subárvores enviadas são escolhidas a partir de uma fila de subárvores (carga acumulada) mantida pelo processo, conforme descrito na próxima subseção.

4.2. Estimativa do tamanho das subárvores

No algoritmo B&B, o tamanho da subárvore gerada por um nó não é conhecido *a priori*. Por este motivo, em uma transferência de carga entre processos, não basta identificar o número de nós que deverão ser transferidos. É necessária, também, uma maneira de estimar o tamanho das subárvores enviadas. Sem este cuidado, um processo que recebeu, por exemplo, uma única subárvore que gera uma grande quantidade de ramificações poderá ficar sobrecarregado enquanto outros processos que receberam uma maior quantidade de subárvores, que produzem um menor número de ramificações, poderão ficar ociosos muito mais rapidamente.

Para resolver este problema, estimamos o tamanho da subárvore gerada por um nó n. O cálculo desta estimativa é baseado nas soluções encontradas pelos nós ancestrais de n. A solução de um nó é o valor encontrado para a função objetivo do SPG quando o limite inferior, chamado dual, ultrapassa o valor do limite superior (primal) no espaço de busca da solução. Sabemos que quanto mais estreito o espaço de busca, ou seja, quanto mais próximos estiverem os valores dos limites, dual e primal, mais próximos estaremos de uma solução.

Para estimarmos o tamanho das subárvores, para todo nó n gerado durante a execução do B&B foi associado um valor $E = 2^{primal/ganho}$ equivalente à estimativa do tamanho da subárvore a ser gerada por ele, onde, primal é a solução mínima para o problema encontrada até o momento e o $ganho = dual_p - dual_n / prof$, considerando que $dual_p$ é o valor do dual encontrado pelo primeiro nó ancestral de n, ou seja, o nó pai de n, $dual_n$ o dual gerado por n e $prof$ a profundidade da subárvore em que n é um nó folha. Para exemplificar, veja a Figura 2. Os nós hachurados são os nós ainda não resolvidos. Estes serão armazenados em uma fila T com as suas respectivas estimativas.

As subárvores enviadas em resposta a uma requisição de carga são aquelas cujas estimativas indicam que terão um maior número de nós. Considere que o nó j recebeu de i uma mensagem LOADREQUEST e calculou a quantidade NN de subárvores que deverá ser enviada a i. O próximo passo de j é selecionar em T_j as NN *subárvores* que possuem menor valor de E e, conseqüentemente, enviar uma mensagem BRANCH contendo as NN subárvores de sua fila T_j que gerarão a maior quantidade de nós. O processo i, ao receber a mensagem, armazenará as subárvores recebidas em T_i. Se i for líder de um *cluster*, redistribuirá a carga recebida com os processos de seu *cluster*. Assim, a quantidade de carga transferida entre *clusters* é proporcional ao desempenho do mesmo, fazendo com que este *cluster* fique devidamente carregado até que se torne ocioso novamente.

5. Resultados Preliminares

Os algoritmos aqui propostos foram desenvolvidos em C++, usando MPICH-G2, uma versão da biblioteca MPI para o Globus [6] [11]. Nossos experimentos preliminares foram executados em uma Grade simulada em um *cluster* da Universidade Federal Fluminense com 16 processadores Athlon de 1.3GHz. Dividimos logicamente o *cluster* em dois outros de 8 processadores cada um e simulamos a troca de

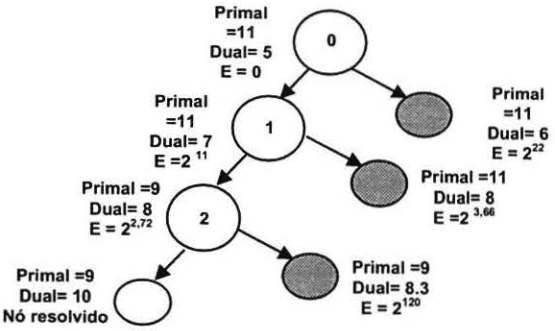

Figura 3. Estimativa do tamanho das subárvores

mensagens entre os mesmos atrasando o envio em ts milesegundos, onde ts é uma média de tempo gasto para enviar uma mensagem de um processo do *cluster* da UFF para a um processo do *cluster* de uma outra universidade do Estado do Rio de Janeiro, PUC-Rio. O algoritmo distribuído foi analisado para quatro instâncias de incidência, obtidas em [7].

Para avaliar o balanceamento de carga das estratégias propostas, executamos, simultaneamente, cada uma das duas versões com a sua correspondente apresentada em [14], com o intuito de obter o mesmo ambiente para os dois algoritmos, de modo que nenhuma abordagem ficasse prejudicada quanto a possíveis diferenças de quantidades de carga externa. Note que, como os algoritmos foram executados concorrentemente e uma versão apresentou um tempo de execução melhor do que o outro, o período final da execução da versão mais lenta poderia ser favorecido, pelo fato de existir um processo a menos disputando os processadores.

Apesar das estratégias terem sido executadas sobre as mesmas condições, existe um fator de aleatoriedade no B&B, que pode fazer com que duas de suas execuções possam apresentar tempos diversos. Para contornar este problema, cada instância foi executada recebendo como parâmetro para o primal o valor da solução ótima, obtido de execução prévia, fazendo com que em execuções sucessivas, aproximadamente, as mesmas quantidades de subárvores fossem resolvidas.

Para avaliar a ociosidade dos processos durante a aplicação, calculamos para cada processo i, a quantidade de tempo to_i em que i ficou ocioso durante a aplicação [15]. Considerando que W_i é o tempo de CPU gasto na aplicação pelo processo i, obtemos a

seguinte medida aproximada η de ociosidade dos processos: $(\sum_{i=1}^{m} to_i)/(\sum_{i=1}^{m} W_i)$.

Para obter os resultados apresentados a seguir, executamos o algoritmo cinco vezes para cada instância e calculamos a média dos tempos de execução, número de mensagens e a taxa de ociosidade. Para todas a execuções, obtivemos os seguintes resultados (instância/valor ótimo): i320-343/16281; i320-345/12689; i640-212/11795; i640-213/11879.

As estratégias propostas neste trabalho diminuíram a ociosidade dos processos, pois evitaram a transferência de subárvore pequenas. Reduziram, também, o número de mensagens trocadas entre os processos para balanceamento de carga. O Gráfico 1 mostra a medida η de processos ociosos quando comparamos as estratégias *HibridaAdapt* e *Hibrida* e o Gráfico 2, quando comparamos a *GlobalAdapt* com a *Global*. Pelos resultados obtidos podemos perceber que a quantidade de subárvores enviadas a um processo ocioso foi adequada, pois, diminuiu, consideravelmente, independente das características das instâncias, como o número de arestas e o número de vértices e o número de vértices terminais do grafo de entrada, a taxa de ociosidade dos processos durante a aplicação.

A Tabela 2 apresenta os valores obtidos quanto à quantidade de mensagens trocada entre os processos. Podemos perceber, pelas colunas onde apresentamos a porcentagem de melhoria em relação a quantidade de mensagens enviadas durante a aplicação, que o *overhead* de comunicação foi efetivamente reduzido, alcançando uma melhoria de quase 30% para todas as instâncias. Entretanto, apesar da diminuição do número, o tamanho das mensagens aumentou, uma vez que a quantidade de subárvores enviadas por transferência também cresceu. Mas, observamos que o acréscimo no tamanho não causou nenhum impacto significativo no desempenho da aplicação.

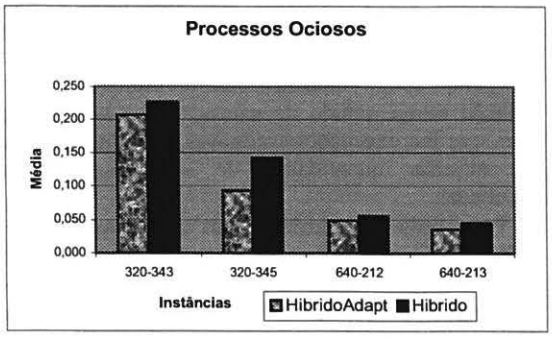

Gráfico 1. Comparação da média de processos ociosos entre as estratégias *HibridaAdapt* e *Hibrida*

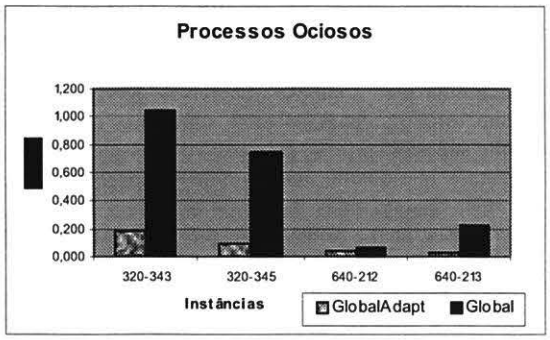

Gráfico 2. Comparação da média de processos ociosos entre as estratégias *GlobalAdapt* e *Global*

Inst	Global Adapt	Global	%	Híbrido Adapt	Hibrido	%
I320-343	**7116**	7759,2	**8,28**	**1975**	2717	**27,30**
I320-345	**8330,6**	13171,6	**36,71**	**3086**	5318,8	**41,97**
I640-212	**7811,8**	11943,6	**34,5**	**2666,8**	3547	**33,00**
I640-213	**8298,6**	11980,4	**30,73**	**3912**	5035,2	**22,30**

Tabela 2. Número de mensagens trocadas entre processos

No que se refere ao tempo de execução da aplicação, a estratégia *HibridaAdapt e GlobalAdapt* obtiveram melhores resultados que a *Híbrida* e a *Global,* respectivamente, como pode ser visto na Tabela 3. Apresentamos a porcentagem de melhoria no tempo de execução na coluna em negrito.

Avaliando os dados descritos acima, podemos comprovar, que a diminuição do *overhead* de comunicação e da taxa de ociosidade dos processos implica em redução do tempo de execução do algoritmo, mostrando que, estratégias de balanceamento de carga bem elaboradas aumentam a eficiência paralela de um algoritmo.

Inst	Glodal Adapt	Global	%	Híbrido Adapt	Hibrido	%
i320-343	512,53	515,38	**0,55**	529,89	536,7	**1,26**
i320-345	1153,23	1283,25	**10,13**	1131,87	1246,74	**9,21**
i640-212	1218,83	1284,56	**5,11**	1252,55	1260,65	**0,64**
i640-213	1615,04	1631,43	**1,00**	1697,59	1712,96	**0,89**

Tabela 3. Tempo médio de execução em segundos

6. Considerações Finais e Trabalhos Futuros

Neste trabalho, propusemos técnicas de balanceamento de carga para o B&B que diferem dos trabalhos encontrados na literatura pelas seguintes razões: não utilizam o paradigma usual mestre-escravo, consideram a estrutura hierárquica das Grades na comunicação entres os processos e utilizam informações referentes às estimativas dos tamanhos das subárvores e ao desempenho dos processadores para definição do tamanho da carga a ser transferida. Estas técnicas de balanceamento de carga foram aplicadas a um algoritmo distribuído previamente desenvolvido para o SPG [14].

Os resultados preliminares obtidos comprovam que a exploração dos dados do problema e características do ambiente nos procedimentos de balanceamento de carga melhoram a eficiência paralela do algoritmo B&B distribuído.

Testes futuros incluirão: análise da escalabilidade, considerando um número maior de *clusters*, e execução de instâncias que demandem maiores tempos computacionais.

7. Referências

[1] K. Aida, W. Natsume, Y. Futakata, "Ditributed Computing with Hierarchical Master-Worker Paradigm for Parallel Branch-and-Bound Algorithms", *In the Proceedings of the Third International Symposium on Cluster Computing and Grid*, 2003, p. 156-162.

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

[3] C. Duin, C. "Steiner's Problems in Graphs", Ph.D. thesis, University of Amsterdam, Holand, 1993.

[4] E. Elnozazhy, D. Johnson, Y. e Wang, "A Survey of Rollback-Recovey Protocols in Message Passing Systems", Technical Report, CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, 1996.

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

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

[7] T. Koch, A. Martin, S. Voss, "SteinLib: An Update Library on Steiner Problems in Graphs", Konrad-Zuse-Zentrum für Informationstechnik Berlin, ZIB-Report 00-37, 2003, http://elib.zib.de/steinlib, last visited Oct 30th, 2003.

[8] T.-H. Lai, S. Sahni, "Anomalies in Parallel Branch-and-Bound Algorithms", *Communications of the ACM*, 27, 1984, p. 594-602.

[9] M. Poggi de Aragão, E. Uchoa, R.S. Wernewck, "Dual Heuristics on the Exact Solution of Large Steiner Problems", *Eletronic Notes in Discrete Mathematics*, 7, 2001.

[10] C. Roucairol, V. Cung, N. Yafoufi, "Design, Implementation and Robustness in Parallel Branch-and-Bound", Techinical Report PRISM 2000/19, Université de Versailles Saint-Quentin, 2000.

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

[12] E. Uchoa, "Algoritmos para Problemas de Steiner com Aplicações em Projeto de Circuitos VLSI", Ph.D. thesis, Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil, 2001.

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

[14] L. Drummond, E. Uchoa, A. Gonçalves, J. M.N. Silva, M. Santos and M. de Castro, "A Grid-Enabled Distributed Branch-and-Bound Algorithm with Application on the Steiner Problem in Graphs", *Relatório técnico, Universidade Federal Fluminense, Niterói. Aceito para publicação na Parallel Computing* em 2005.

[15] k. Anstreicher, N. Brixius, J. Goux, J. Linderoth, "Solving Large Quadratic Assignment Problems on Computational Grids", *Mathematical Programming*, 2002, p 563-588.

[16] C. Roucairol, S. Dowaji, "Load Balancing Strategy and Priority of Tasks in Distributed Environments", *Computers and Communications*, 1995.

[17] L. Drummond, E. Uchoa, M. C. S. Castro, J. Viterbo, A. D. Gonçalves, "A Distributed Branch-and-Bound Algorithm for Wide-Area Environments", In Proceedings of the 6th International Meeting of High Performance Computing for Computational Science,2004, p 927-932.