Implementações de Algoritmos Paralelos FPT para o Problema da k-Cobertura por Vértices utilizando *Clusters* e Grades Computacionais*

Henrique Mongelli e Rodrigo Cesar Sakamoto
Universidade Federal de Mato Grosso do Sul
Departamento de Computação e Estatística
Caixa Postal, 549
Campo Grande, MS, 79.070-900 Brazil
{mongelli, rodrigo_sakamoto}@dct.ufms.br

Resumo

Em muitas aplicações problemas NP-completos precisam ser solucionados de forma exata. Um método promissor para tratar com alguns problemas intratáveis é através da Complexidade Parametrizada que divide a entrada do problema em uma parte principal e um parâmetro. A parte principal contribui polinomialmente com a complexidade total do problema, enquanto que o parâmetro é responsável pela explosão combinatorial. Consideramos o algoritmo paralelo FPT de Cheetham para solucionar o problema da k-Cobertura por Vértices e a implementação refinada e melhorada de Hanashiro. Como este é um problema em que grande parte do tempo de execução é feita de forma independente, sem a necessidade de comunicação entre os processadores, a utilização de grades computacionais torna-se bastante aplicável, com a possibilidade do emprego de um número grande de processadores. Este trabalho envolve a implementação no Integrade de algoritmos FPT paralelos para o problema da k-Cobertura por vértices. A grade computacional dos testes utiliza o middleware desenvolvido no Projeto Integrade. Estes algoritmos foram implementados usando a biblioteca BSPLib do Integrade e mostraram um desempenho muito bom e que pode ser melhorado com a adição de novos processadores. Em nossos experimentos no Integrade, em comparação a implementação em cluster, obtivemos tempos paralelos melhores do que os relatados por Hanashiro.

1. Introdução

A abordagem da intratabilidade dos problemas é um grande desafio teórico e prático da Ciência da Computação. Como muitos problemas NP-completos têm grande aplicação prática, é necessário buscar formas de contornar a falta de algoritmos eficientes que apresentem soluções exatas para todas as instâncias do problema.

A complexidade parametrizada tenta lidar com a intratabilidade e vem obtendo sucesso na solução de instâncias de problemas que antes eram consideradas muito grandes para serem resolvidas com os métodos já existentes. A idéia é dividir a entrada em uma parte principal n e um parâmetro fixo k [5]. A parte principal contribui polinomialmente no total da complexidade do problema, enquanto a inevitável explosão combinatorial fica limitada pelo parâmetro. Diz-se que um problema parametrizável é tratável por parâmetro fixo ou FPT (*Fixed-Parameter Tractable*) quando existe um algoritmo que o resolve em tempo $O(f(k)n^{\alpha})$, onde α é uma constante e f é uma função arbitrária. Existem duas técnicas comumente aplicadas no desenvolvimento de algoritmos FPT: a redução ao núcleo do problema e a árvore limitada de busca, as quais podem ser combinadas para resolver problemas tratáveis por parâmetro fixo.

A complexidade exponencial do parâmetro ainda pode resultar em custos proibitivos. A utilização do paralelismo, através da combinação dos modelos BSP (*Bulk Synchronous Parallel*) e CGM (*Coarsed Grained Multicomputer*), favorece a utilização de instâncias ainda maiores na solução de problemas FPT.

O modelo CGM, proposto por Dehne *et al.* [17], consiste em um conjunto de p processadores, cada um com memória local de tamanho $O(n/p)$ e conectados por uma rede de interconexão, onde n é o tamanho do problema e $n/p \geq p$. Este modelo é uma simplificação do modelo BSP proposto por Valiant [18], que também é um modelo dito realístico, pois define parâmetros para mapear as principais características de máquinas paralelas reais, ou seja, levando em consideração, dentre outras coisas, o tempo de comunicação entre os processadores.

*Este trabalho foi financiado por CAPES, CNPq e FUNDECT.

Um algoritmo BSP/CGM possui rodadas de computação local alternadas com rodadas de comunicação entre os processadores, onde cada processador envia e recebe, em cada rodada, $O(n/p)$ dados no máximo. As rodadas de computação local e de comunicação entre os processadores são separadas por barreiras de sincronização.

O tempo de execução de um algoritmo BSP/CGM é a soma dos tempos gastos tanto com computação local quanto com comunicações entre os processadores. Nas rodadas de computação local, geralmente utiliza-se o melhor algoritmo seqüencial para o processamento, além de estar-se interessado em minimizar o número de rodadas de computação.

As implementações em máquinas paralelas reais dos algoritmos projetados no modelo BSP/CGM têm obtido tempos bastante próximos aos previstos no modelo. A combinação do paralelismo e de algoritmos FPT tem se mostrado profícua na obtenção de soluções para problemas práticos, em especial para o algoritmo FPT que soluciona o problema da k-Cobertura por Vértices gastando pouca rodada de comunicação observada na implementação desenvolvida por Hanashiro [1].

O problema da k-Cobertura por Vértices consiste em, dados o grafo $G = (V, E)$ e um número inteiro k, determinar se existe um subconjunto de vértices V' de G de tamanho máximo k, tal que toda aresta tem pelo menos um vértice em V'. Além disso, como este é um problema em que grande parte do tempo de execução é feita de forma independente, sem a necessidade de comunicação entre os processadores, a utilização de grades computacionais torna-se bastante aplicável, com a possibilidade de envolvimento de um número grande de processadores.

O InteGrade [16] é um middleware de grade com o objetivo de integrar recursos, conectados ponto-a-ponto ou organizados de uma forma hierárquica, utilizando os recursos computacionais ociosos de máquinas disponíveis em diversas instituições para executar aplicações paralelas ou seqüenciais em grade; permitindo o acesso remoto a hardware e software de outras máquinas e o compartilhamento de recursos computacionais; facilitando o escalonamento e o gerenciamento através da estrutura hierárquica.

O InteGrade possui uma arquitetura orientada a objetos, onde cada módulo do sistema comunica-se com os demais a partir de chamadas remotas de métodos. A comunicação entre os nós da aplicação é realizado pelo Corba. É utilizada a biblioteca BSPLib da Universidade de Oxford, incluindo um cabeçalho diferente, recompilando a aplicação e religando-o à biblioteca BSP do InteGrade. O InteGrade é um projeto desenvolvido em conjunto por cinco instituições: Departamento de Ciência da Computação (IME-USP), Departamento de Informática (PUC-RIO), Departamento de Informática (UFMA), Instituto de Informática (UFG) e Departamento de Computação e Estatística (UFMS).

Neste trabalho discutiremos os resultados das implementações do algoritmo FPT paralelo para a k-Cobertura por Vértices de Hanashiro et al. [2] e de Cheetham [3], e compararemos a implementação em *clusters* desenvolvida por Hanashiro [1] e a nossa implementação no Integrade.

Foram utilizados três grafos de conflitos referentes a aminoácidos, os mesmos usados por Cheetham *et al.* [3] e Hanashiro [1] em seus experimentos. Os resultados da implementação no InteGrade obtiveram melhores tempos paralelos, em relação à implementação em *cluster* no ambiente descrito por Hanashiro [1] e no BIOPAD [1].

Este trabalho está organizado como se segue. Na Seção 2 são apresentados alguns conceitos necessários à apresentação deste trabalho. Na Seção 3 são descritos os três algoritmos FPT seqüenciais que resolvem o problema da k-Cobertura por Vértices e que foram utilizados nas implementações paralelas. Na Seção 4 são discutidos os resultados de nossos experimentos. Finalmente, na Seção 5 são mostrados as conclusões.

2. Intratabilidade e Algoritmos FPT

Nesta seção são abordados os conceitos necessários para o desenvolvimento dos algoritmos FPT para o problema da k-Cobertura por Vértices e sua definição.

2.1. O Problema da Cobertura por Vértices[4]

Uma cobertura por vérices para um grafo não dirigido $G = (V, E)$ é um conjunto de vértices V' de tamanho no máximo k com $k \leq V$ tal que, qualquer aresta em G é incidente em pelo menos um dos vértices de V', como ilustrado na Figura 1. O problema da Cobertura por Vértice consiste em calcular uma cobertura de vértices com o número mínimo de vértices.

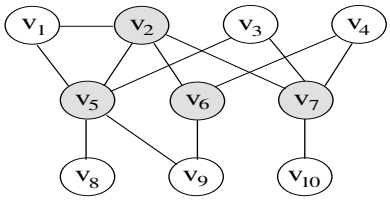

Figura 1. A cobertura $V' = \{v_2, v_5, v_6, v_7\}$ é uma das possíveis cobertura por vértices válida.

[1] Laboratório de Computação de Alto Desempenho e Bioinformática do Departamento de Computação e Estatística da Universidade Federal de Mato Grosso do Sul

2.2. Complexidade Parametrizada

A teoria da complexidade parametrizada [5-12] surgiu como uma tentativa promissora para lidar com a intratabilidade de alguns problemas. Para isso, deve-se isolar alguns aspectos da entrada, tornando-as parâmetros fixos [5], e limitar a aparentemente inevitável explosão combinatoral, confinando-a ao parâmetro. A parte principal da entrada contribui de forma polinomial para a complexidade total do problema. O objetivo principal é combinar ambas as partes, tanto a que contribui de forma polinomial como a que contribui de forma exponencial para a complexidade total do problema, possibilitando a resolução do problema, diferentemente da complexidade clássica onde a explosão combinatorial corresponde a toda a entrada de um problema intratável.

No entanto, precisamos limitar o parâmetro em um "pequeno intervalo" devido à complexidade computacional envolvida. Em muitas aplicações, o parâmetro pode ser considerado "bem pequeno" quando comparado ao tamanho da parte principal da entrada [1].

Esses problemas tratáveis por parâmetro fixo formam a classe de problemas denominada FPT (*Fixed-Parameter Tractability*). Os algoritmos que resolvem problemas dessa classe são chamados de algoritmos FPT, pois os problemas podem ser resolvidos eficientemente para valores pequenos do parâmetro fixo.

Dois métodos elementares são usados na construção de algoritmos para problemas tratáveis por parâmetro fixo: redução ao núcleo do problema e árvore limitada de busca. A aplicação destes métodos combinados, nessa ordem, como um algoritmo de duas fases, é a base de vários algoritmos FPT. Apesar de serem estratégias algorítmicas simples, elas não surgem naturalmente, pois envolvem custos exponenciais relativos ao parâmetro [10].

- Redução ao núcleo do problema: Este método reduz um problema de instância I com parâmetro k, transformando I, em tempo polinomial, em uma nova instância I' com parâmetro k' tal que o tamanho de I' é limitado por uma função que depende somente de k', $k' < k$, e (I, k) tem uma solução se, e somente se, (I', k') tem uma solução.

- Árvore limitada de busca: O objetivo deste método é computar uma árvore de tamanho exponencial, limitado em uma função do parâmetro k. A construção da árvore limitada de busca utiliza um algoritmo que ramifica o nó da árvore em vários ramos de forma eficiente, armazenando uma instância do problema. Geralmente o nó da raiz armazena a instância resultante da fase de redução ao núcleo do problema. A busca em profundidade na árvore limitada é preferido devido ao espaço de busca que pode ser proporcional à altura da árvore.

Além disso, um problema é tratável por parâmetro fixo se, e somente se, é redutível ao núcleo do problema [8].

2.3. A Versão Parametrizada do Problema da Cobertura por Vértices [7]

O problema da k-Cobertura por Vértices consiste de uma entrada para o problema representado por um grafo não orientado $G = (V, E)$ (a instância) e um número inteiro k (o parâmetro). O problema resume-se em descobrir se existe um subconjunto de vértices V' de G de tamanho máximo k tal que toda aresta tem pelo menos um vértice em V'. Podemos transformá-lo em um problema de otimização ao calcular uma cobertura de vértices com o número mínimo de vértices de tamanho no máximo k.

3. Algoritmos FPT para a k-Cobertura por Vértices

Nesta seção serão apresentados algoritmos FPT que resolvem o problema da k-Cobertura por Vértices.

3.1. Algoritmo de Buss

O algoritmo de Buss [13] recebe um grafo $G = (V, E)$ e um inteiro k. Este algoritmo utiliza o método de redução ao núcleo do problema e reduz em tempo polinomial o grafo G em G', sendo que o tamanho de G' é limitado por uma função do parâmetro k. O algoritmo remove todos os vértices do grafo G de grau maior que k e os adiciona ao conjunto H, que pertence a qualquer cobertura por vértices para o grafo G de tamanho menor ou igual a k. Também são removidos as arestas incidentes nos vértices de H e quaisquer vértices isolados, e $k' = k - |H|$. Caso haja mais que k vértices de grau maior ou igual a k, não haverá uma cobertura por vértices para G de tamanho no máximo k.

A partir deste momento, se o número de arestas for menor que $k.k'$, aplica-se um algoritmo de força bruta para determinar se existe ou não uma cobertura por vértices para G' de tamanho menor ou igual a k', para a instância $\langle G', k' \rangle$ e cobertura parcial por vértices H. O tempo total do algoritmo é $O(kn + (2k^2)^k k^2)$, considerando o tempo gasto no método de redução ao núcleo do problema mais o tempo gasto no algoritmo de força bruta.

3.2. Algoritmos de Balasubramanian *et al.*

Os algoritmos de Balasubramanian *et al.* [14] recebem um grafo $G = (V, E)$ e um inteiro k. Inicialmente, aplica-se o método de redução ao núcleo do problema, gerando a

instância $\langle G', k' \rangle$ e uma cobertura parcial H. Na segunda fase, uma árvore limitada de busca é gerada. As duas opções para a geração dessas árvores são mostradas em Balasubramanian *et al.* [14] e estão descritas a seguir e denominados Algoritmo B1 e Algoritmo B2. Em ambos os casos, fazemos uma busca exaustiva nos nós da árvore por uma solução para o problema da cobertura por vértices, através de uma busca em profundidade. A diferença entre estes dois algoritmos é a forma como os vértices são adicionados à cobertura parcial e, conseqüentemente, o formato dessa árvore.

Cada nó da árvore de busca armazena uma cobertura parcial e uma instância reduzida do grafo. Esta cobertura parcial é composta de vértices que pertencem à cobertura. A instância reduzida é formada pelo grafo resultante da remoção dos vértices de G que estão na cobertura parcial, bem como todas as arestas incidentes a eles e quaisquer vértices isolados. Este grafo G'' e um inteiro k'' é o tamanho máximo desejado da cobertura de G''. A raiz da árvore de busca, por exemplo, representa a situação após o método da redução o núcleo do problema. Em outras palavras, na cobertura parcial temos os vértices de grau menor ou igual a k e a instância $\langle G', k' \rangle$.

As arestas da árvore de busca representam as várias possibilidades de adição de vértices à cobertura parcial existente. Deve-se notar que o filho de um nó da árvore tem mais elementos na cobertura parcial e um grafo com menos nós e arestas que seu pai, visto que cada vez que um vértice é adicionado à cobertura parcial, ele é removido do grafo, juntamente com as arestas nele incidente e quaisquer vértices incidentes. Os nós são gerados a partir de uma busca em profundidade e não é necessário gerar todos os nós da árvore (aplica-se o método de *backtracking*). Cada nó da árvore armazena uma cobertura por vértices parcial H e uma instância reduzida do problema $\langle G'' = (V'', E''), k'' \rangle$. O grafo G'' é resultante da remoção das arestas incidentes em H e quaisquer vértices isolados, e o número k'' é o tamanho máximo da cobertura por vértices de G''.

A árvore de busca tem a seguinte propriedade: para cada cobertura existente para o grafo G de tamanho menor ou igual a k, existe um nó da árvore com um grafo vazio e uma cobertura por vértices (não necessariamente a mesma) de tamanho menor ou igual a k. Entretanto, se não existe cobertura por vértices de tamanho menor ou igual a k para o grafo G, então nenhum nó da árvore tem um grafo resultante vazio. Na realidade, o crescimento da árvore de busca é interrompido quando o nó tem uma cobertura parcial de tamanho menor ou igual a k ou um grafo resultante vazio (caso em que não encontramos uma cobertura por vértices válida para o grafo G). Deve-se notar que isto limita o tamanho da árvore em termos do parâmetro k. No pior caso, toda a árvore de busca deve ser atravessada para determinar se não existe uma cobertura por vértices de tamanho menor ou igual a k para o grafo G.

3.2.1 Algoritmo B1

Neste algoritmo, a escolha dos vértices de G'' a serem adicionados à cobertura parcial em qualquer nó da árvore é feito de acordo com o caminho gerado a partir de qualquer vértice v de G'' que passa por um caminho de no mínimo três arestas.

Se este caminho tem comprimento um ou dois, então o vizinho ou o nó de grau um é adicionado à cobertura parcial, remove-se suas arestas incidentes e quaisquer vértices isolados. Esta nova instância com a nova cobertura parcial é mantida no mesmo nó da árvore limitada de busca e o Algoritmo B1 é aplicado novamente neste nó.

Se o caminho é um caminho simples de comprimento três, passando pelos vértices v, v_1, v_2 e v_3, qualquer cobertura deve conter $\{v, v_2\}$ ou $\{v_1, v_2\}$ ou $\{v_1, v_3\}$. Se o cami-nho é um circuito simples de comprimento três, passando pelos vértices v, v_1, v_2 e v, qualquer cobertura deve conter $\{v, v_1\}$ ou $\{v_1, v_2\}$ ou $\{v, v_2\}$. Em ambos os casos, o nó da árvore é ramificado em três filhos para adicionar cada um dos três pares de vértices. O próximo nó da árvore é processado, chamando novamente a busca em profundidade.

Este algoritmo gera uma árvore terciária e em cada nível da árvore a cobertura parcial cresce de no máximo dois vértices. O Algoritmo B1 gasta tempo $O(kn + (\sqrt{3})^k k^2)$ para resolver o problema da k-Cobertura por Vértices.

3.2.2 Algoritmo B2

Neste algoritmo, a escolha dos vértices de G'' a serem adicionados à cobertura parcial em qualquer nó da árvore é feita de acordo com cinco casos, considerando o grau dos vértices do grafo resultante. Primeiramente são escolhidos os vértices de grau um (Caso 1), depois os de grau dois (Caso 2), depois dos de grau cinco ou mais (Caso 3), depois os de grau três (Caso 4) e, finalmente, os vértices de grau quatro (Caso 5).

Utiliza-se a seguinte notação: $N(v)$ representa o conjunto de vértices que são vizinhos do vértice v e $N(S)$ representa o conjunto $\bigcup_{v \in S} N(v)$.

No Caso 1, se existe um vértice v de grau um no grafo, então um novo filho é criado adicionando $N(v)$ à cobertura parcial.

No Caso 2, se existe um vértice v de grau dois no grafo, então tem-se três subcasos a serem testados na seguinte ordem. Sejam x e y os vizinhos de v. No Subcaso 1, se existe uma aresta entre x e y, então um novo filho é criado para adicionar $N(v)$ à cobertura parcial. No Subcaso 2, se x e y tem no mínimo dois vizinhos diferentes de v, então os nós da árvore são ramificados em dois filhos para adicionar $N(v)$ e $N(\{x, y\})$, respectivamente, às coberturas parciais. No Subcaso 3, se x e y compartilham um único vizinho a

diferente de v, então um novo filho é criado para adicionar $\{v, a\}$ à cobertura parcial.

No Caso 3, se existe um vértice de grau cinco ou maior no grafo, então o nó da árvore é ramificado em dois filhos para adicionar v e $N(v)$, respectivamente, às coberturas parciais.

Se nenhum dos três casos anteriores ocorrer, então temos um grafo 3 ou 4-regular. No caso 4, existe um vértice v de grau 3, então pode-se ter quatro subcasos a serem tratados na seguinte ordem. Sejam x, y e z os vizinhos de v. No Subcaso 1, se existe uma aresta entre dois vizinhos de v, digamos x e y, então os nós da árvore são ramificados em dois filhos para adicionar $N(v)$ e $N(z)$, respectivamente, à cobertura parcial. No Subcaso 2, se um par de vizinhos de v, digamos x e y, compartilham um outro vizinho em comum a (mas diferente de v), então o nó da árvore é ramificado em dois filhos para adicionar $N(v)$ e $\{v, a\}$, respectivamente, à cobertura parcial. No Subcaso 3, se o vizinho de v, digamos x, tem ao mínimo três vizinhos diferentes de v, então a árvore é ramificada em três filhos para adicionar $N(v)$, $N(x)$ e $x \bigcup N(\{y, z\})$ à cobertura parcial. No Subcaso 4, os vizinhos de v têm exatamente dois vizinhos privados, sem considerarmos o vértice v. Seja x um vizinho de v e a e b os vizinhos de x, então o nó da árvore é ramificado em três filhos para adicionar $N(v)$, $\{v, a, b\}$ e $N(\{y, z, a, b\})$ à cobertura parcial.

No Caso 5, o grafo é 4-regular e existem três subcasos para serem testados na seguinte ordem. Seja v um vértice do grafo e x, y, z e w seus vértices vizinhos. No Subcaso 1, se existe uma aresta entre dois vizinhos de v, digamos x e y, então o nó da árvore é ramificado em três filhos para adicionar $N(v)$, $N(z)$ e $z \bigcup N(w)$ à cobertura parcial. No Subcaso 2, se três vizinhos de v, digamos x, y e z compartilham um vizinho comum a, então o nó da árvore é ramificado em dois vizinhos para adicionar $N(v)$ e (v, a) à cobertura parcial. No Subcaso 3, se cada um dos vizinhos de v tem três vizinhos diferentes de v, então os nós da árvore são ramificados em quatro filhos para adicionar $N(v)$, $N(y)$, $y \bigcup N(w)$ e $\{y, w\} \bigcup N(\{x, z\})$ à cobertura parcial.

Ao contrário do Algoritmo B1, um nó na árvore de busca pode ser ramificado em dois, três, ou quatro filhos, e a cobertura parcial pode crescer em até oito vértices, dependendo do caso selecionado. O Algoritmo B2 gasta tempo $O(kn + 1.324718^k k^2)$ para solucionar o problema da k-Cobertura por Vértices.

3.3. Algoritmo de Cheetham *et al.*

O algoritmo de Cheetham *et al.* [3] paraleliza a fase de redução ao núcleo do problema e árvore limitada de busca.

Na fase de redução ao núcleo do problema utiliza-se o método seqüencial de redução ao núcleo do problema proposto pelo algoritmo de Buss [13]. Na versão paralela, cada processador $P_i, 0 \leq i \leq p-1$, é responsável por computar o grau dos vértices rotulados de $i*(n/p)$ a $((i+1)*(n/p))-1$, e recebe m/p arestas da lista de arestas do grafo G e as ordena pelo primeiro vértice no qual incide e informa aos demais processadores quais vértices locais têm grau maior que k, e, assim, todos os processadores serão capazes de remover as arestas que incidem em tais vértices e os respectivos vértices. Em seguida, cada processador, trocará mensagens enviando as arestas resultantes, de forma que cada processador tenha uma cópia da instância final da fase de redução ao núcleo.

Na fase da árvore limitada de busca é gerada a árvore ternária completa T com altura $h = log_3 p$ e com p folhas (y_0 a y_{p-1}), pelo algoritmo B1 de forma determinística, com o nó raiz armazenando a cobertura por vértices parcial e a instância $\langle G', k' \rangle$.

A árvore T não é explicitamente criada pelos processadores (o método de *backtracking* é aplicado). Cada processador $P_i, 0 \leq i \leq p-1$ percorre o único caminho entre a raiz e a folha y_i calculado pela fórmula $i/(p/3^{h+1})$. O Algoritmo B1 é interrompido quando atinge um nó de nível $log_3 p$ (folha y_i). O vértice inicial da busca em profundidade deve ser o mesmo para todos os processadores.

Em seguida, cada processador P_i executa localmente a instância $\langle G''_i, k''_i \rangle$ referente à folha y_i da árvore T usando o algoritmo B2, como apresentado na Figura 2. Esta fase será encerrada ou quando encontrarmos uma cobertura por vértices de tamanho menor ou igual a k para G, ou se todos os processadores percorreram toda a árvore, significando que não foi possível encontrar uma cobertura válida.

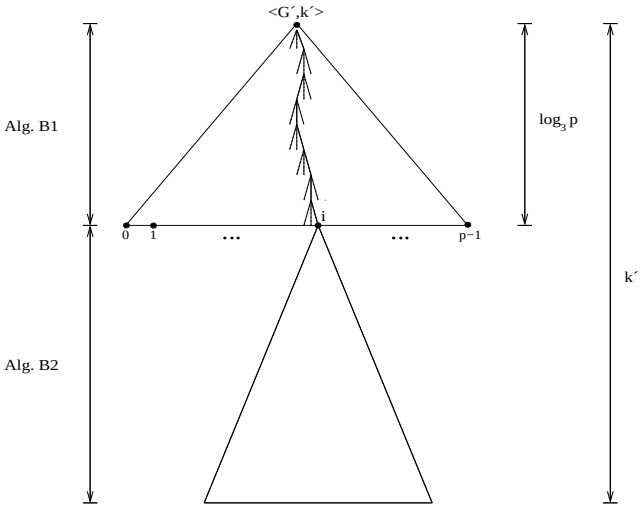

Figura 2. O processador P_i processa o caminho único até a folha y_i usando o Algoritmo B1. Depois, P_i processa toda a subárvore de raiz y_i usando o Algoritmo B2.

O algoritmo FPT de Cheetham *et al.* [3] foi implementado por Hanashiro [1]. Os resultados obtidos foram bastante superiores utilizando um *cluster* inferior. Os méritos da implementação de Hanashiro foram a utilização de estruturas de dados eficientes e do método de *backtracking* na geração dos nós da árvore limitada de busca.

4. Resultados das implementações em *Cluster* e Grade Computacional

Nesta seção, são apresentados os resultados experimentais obtidos na implementação BSP/CGM do algoritmo FPT de Cheetham *et al.*, baseados nas implementações de Hanashiro [1].

4.1. Ambiente Computacional e Metodologia

Os algoritmos BSP/CGM foram executados em um *cluster* composto por 12 nós: uma máquina AMD Athlon(tm) 1800+ 1GB de RAM; uma máquina Intel(R) Pentium(R) 4 CPU 1.70GHz 1GB de RAM; três máquinas Pentium IV 2.66GHz 512MB; uma máquina Pentium IV 2.8GHz 512MB; uma máquina Pentium IV 1.8GHz 480MB; quatro máquinas AMD Athlon(tm) 1.66GHz 480MB; uma máquina AMD Sempron(tm) 2600+ 480 MB. Os nós estão conectados por um *fast-Ethernet switch* de 1Gb. Cada nó executava o sistema operacional Linux Fedora 6 com g++ 4.0 e MPI/LAM 7.1.2. Na Grade Computacional foi utilizado o mesmo ambiente descrito, com o middleware Grade InteGrade versão 0.2 RC4.

Os dados de entrada usados são grafos de conflitos referentes a aminoácidos, os mesmos usados por Cheetham *et al.* [3] e Hanashiro [1] em seus experimentos. Cada valor dos gráficos nas subseções a seguir corresponde ao tempo médio de 30 experimentos. Os tempos obtidos são apresentados em segundos, incluindo o tempo para leitura dos dados de entrada, desalocação das estruturas utilizadas e impressão da saída, e o tempo entre o início do primeiro processo até o final do último processo. O tempo de execução seqüencial corresponde ao tempo de execução da implementação paralela em uma máquina *cluster* com um único processador.

4.2. Implementação em *Cluster* e Grade Computacional

A implementação paralela foi executado em 1, 3 e 9 máquinas reais para os grafos PHD, Somatostatin e WW.

Na Figura 3, pode-se observar que o número de processadores influencia a quantidade de pontos iniciais de busca, conseqüentemente, o tempo de execução. Porém, deve-se ressaltar que a quantidade de processadores não garante a solução em um tempo menor. Pode-se presumir que a quantidade de nós que contém soluções na árvore de busca também influencia no tempo de execução, conforme relato de Hanashiro [1].

A Figura 4 ilustra o ganho obtido com a utilização de um número maior de processadores. Foram obtidos desempenhos considerados satisfatórios para os grafos analisados, em especial para o grafo WW. O *speedup* superlinear para o grafo WW com 9 processadores pode ser devido à velocidade maior com que uma solução é encontrada, pela quantidade de pontos iniciais de busca.

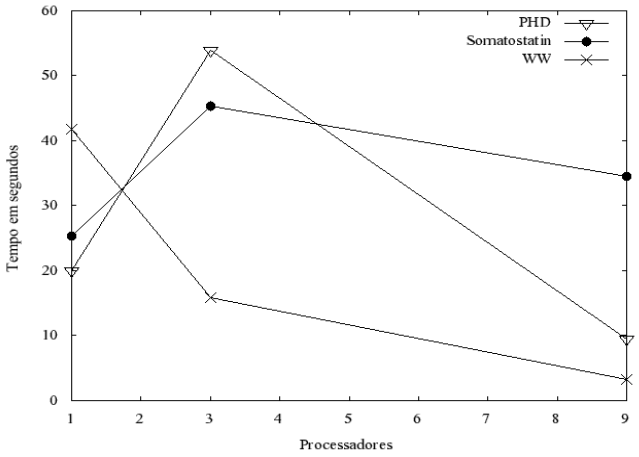

Figura 3. Gráfico Processadores x Tempo

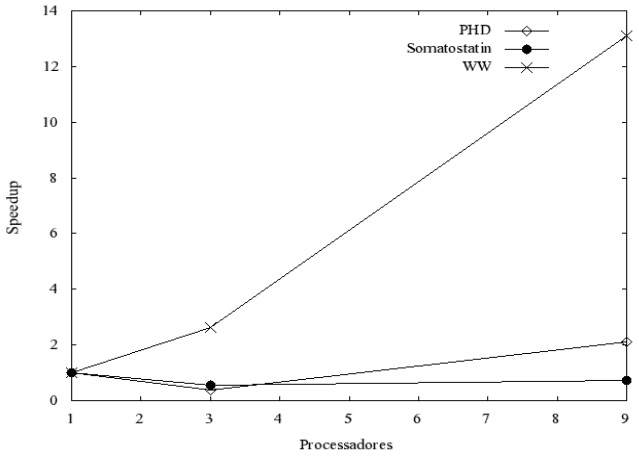

Figura 4. Gráfico Processadores x Speedup

Nas Figuras 5 e 6 pode-se observar que os tempos paralelos obtidos na implementação no InteGrade são melhores que os tempos paralelos obtidos na implementação em *cluster* no ambiente descrito por Hanashiro [1] e no ambiente computacional utilizado em nossos experimentos. Devemos destacar que o ambiente computacional utilizado em

nossos experimentos é superior ao utilizado por Hanashiro [1] e, conseqüentemente, foram obtidos melhores tempos paralelos em comparação a Cheetham *et al.* [3].

Como pode ser observado na Figura 5, para os grafos PHD, Somatostatin e WW, os resultados obtidos foram muito melhores, com tempos paralelos aproximadamente 25 vezes melhores para o grafo PHD, 4 vezes melhores para o grafo Somatostatin e 2 vezes melhores para o grafo Somatostatin em relação aos relatados por Hanashiro [1] em comparação à implementação no InteGrade. Além disso, em comparação à implementação em *cluster* no mesmo ambiente computacional, obteve-se resultados melhores, para o grafo PHD, com tempos paralelos aproximadamente 6 vezes melhores e 1.2 vezes melhores para o grafo Somatostatin.

Na Figura 6, para a implementação no InteGrade, o grafo PHD obteve melhor tempo paralelo tanto em comparação ao resultado da implementação em *cluster* como ao de Hanashiro [1], ambos aproxidamente 5 vezes melhores. Para o grafo Somatostin foram obtidos resultados melhores, mas para o grafo WW não foram obtidos bons resultados.

Outro ponto a destacar é que pode-se obter tempos paralelos melhores no InteGrade ao adicionar mais nós ao ambiente computacional utilizado nos experimentos conforme as Figuras 5 e 6.

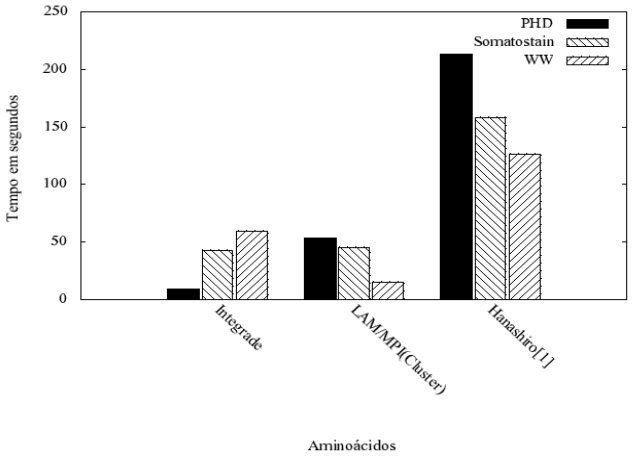

Figura 5. Gráfico InteGrade x LAM/MPI x Hanashiro[1]. Execução em 3 máquinas reais.

5. Conclusão

O problema da k-Cobertura por Vértices é um típico problema tratável por parâmetro fixo. A paralelização de algoritmos FPT para a k-Cobertura por Vértices vem demostrando bons resultados, devido, principalmente, ao fato do

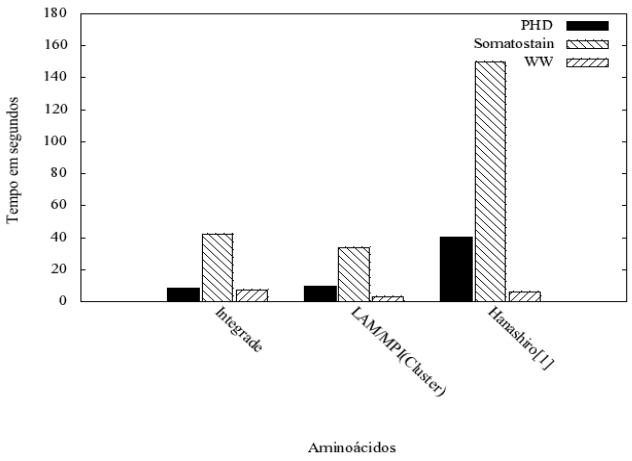

Figura 6. Gráfico InteGrade x LAM/MPI x Hanashiro[1]. Execução em 9 máquinas reais.

problema necessitar de pouca comunicação entre os processadores.

O algoritmo FPT de Cheetham *et al.* [3] para o problema da k-Cobertura por Vértices, pela primeira vez paralelizou as duas fases de um algoritmo FPT. Hanashiro [1] implementou o algoritmo FPT de Cheetham *et al.* [3] de maneira cuidadosa observando detalhes como a utilização de estrutura de dados eficientes e do método de *backtracking* na geração dos nós da árvore limitada de busca. Os tempos obtidos foram consideravelmente melhores, usando uma plataforma inferior.

Nossa implementação foi feita utilizando a biblioteca BSPLib. Essa implementação foi executada no InteGrade local, instalado na rede do BIOPAD. Ao compararmos nossa implementação com a implementação de Hanashiro, no mesmo conjunto de máquinas e utilizando o mesmo conjunto de dados, pôde-se observar um bom ganho. Para o grafo PHD, foram obtidos tempos paralelos aproximadamente 25 vezes melhores do que o relatado por Hanashiro [1] e aproximadamente 6 vezes melhores, em comparação a implementação em *cluster* utilizando o mesmo ambiente computacional.

A versão paralela do algoritmo FPT de Cheetham para o problema da k-Cobertura por Vértices, utilizando o InteGrade, teve ganhos significativos. A adição de mais nós na grade computacional utilizada em nossos experimentos, poderá fornecer tempos melhores. A utilização combinada do método FPT, do paralelismo e de grades computacionais apresenta ganhos expressivos na solução de problemas FPT.

Entre os trabalhos em desenvolvimento estão a implementação de algoritmos FPT paralelos para o problema da k-Cobertura por Vértices que utilizam outros algoritmos seqüenciais, mais rápidos, nas duas fases do al-

goritmo de Cheetham *et al.* [3]. Além disso, estão sendo estudados algoritmos FPT paralelos para outros problemas NP-Completos.

Referências

[1] E. J. Hanashiro. O problema da k-cobertura por Vértices: uma implementação FPT no modelo CGM. Dissertação de Mestrado, Universidade Federal de Mato Grosso do Sul, Março, 2004.

[2] E. J. Hanashiro, H. Mongelli, e S. W. Song. Efficient implementation of the BSP/CGM parallel vertex covert FPT algorithm. *In: Third International Workshop on Experimental and Efficient Algorithms - WEA 2004*, Angra dos Reis. Lecture Notes in Computer Science. Springer-Verlag. 3059: 253-268, 2004.

[3] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, e P. J. Taillon. Solving large FPT problems on coarse grained parallel machines. *Journal of Computer and System Sciences*, 67(4):691706, 2003.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, e C. Stein. Algoritmos: Teoria e Prática. Campus, 2002.

[5] R. G. Downey e M. R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM Journal on Computing*, 24:873921, 1995.

[6] R. G. Downey e M. R. Fellows. Fixed-parameter tractability and completeness II: completeness for W[1]. *Theoretical Computer Science*, 141:109131, 1995.

[7] R. G. Downey e M. R. Fellows. Parameterized computational feasibility. *In Feasible Mathematics II*, páginas 219244. Birkhauser Boston, 1995.

[8] R. G. Downey e M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.

[9] R. G. Downey e M. R. Fellows. Parameterized complexity after (almost) 10 years: *review and open questions. In Combinatorics, Computation & Logic, DMTCS'99 and CATS'99*, volume 21, número 3, páginas 133. Australian Computer Science Communications, Springer-Verlag, 1999.

[10] R. G. Downey, M. R. Fellows, e U. Stege. Computational tractability: *the view from Mars. Bulletin of the European Association for Theoretical Computer Science*, 69:7397, 1999.

[11] R. G. Downey, M. R. Fellows, e U. Stege. Parameterized complexity: a framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 de AMS-DIMACS Proceedings Series, páginas 4999. 1999.

[12] R. Niedermeier. Some prospects for efficient fixed parameter algorithms. In *Conference on Current Trends in Theory and Practice of Informatics*, páginas 168185. 1998.

[13] J. F. Buss e J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560572, 1993.

[14] R. Balasubramanian, M. R. Fellows, e V. Raman. An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters*, 65:163168, 1998.

[15] M. Pitanga. Construindo Supercomputadores com Linux. Brasport. 2004.

[16] A. Goldchleger, F. Kon, A. Goldman, M. Finger, e G. C. Bezerra. *InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines*. 2000. Disponível em $< \ http \ : //gsd.ime.usp.br/publications/cpe03_integrade.pdf >$. Acesso em 16 de Maio de 2007.

[17] L. G. Valiant. A bridging model for computation. *Communications of the ACM*, 33:103-111, 1990.

[18] F. Dehne, A. Fabri, e A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. *In Proceedings of the ACM 9th Annual Computational Geometry*, 298-307, 1993.