Boletim Técnico da Escola Politécnica da USP
Departamento de Engenharia de Computação e
Sistemas Digitais

ISSN 1413-215X
BT/PCS/9903

Implementação Paralela Distribuída da Dissecação Cartesiana Aninhada

Hilton Garcia Fernandes
Liria Matsumoto Sato

São Paulo - 1999

O presente trabalho é parte da dissertação de mestrado "Estratégias para implementação Paralela-Distribuída da Dissecação Aninhada Cartesiana", apresentada em 13/08/98, por Hilton Garcia Fernandes, sob orientação do Profa. Dra. Liria Matsumoto Sato.

A íntegra da dissertação encontra-se à disposição com o autor e na Biblioteca de Engenharia Elétrica da Escola Politécnica da USP.

FICHA CATALOGRÁFICA

Fernandes, Hilton Garcia
Implementação paralela distribuída da dissecação cartesiana aninhada / H.G. Fernandes, L.M. Sato. -- São Paulo : EPUSP, 1999.
19 p. -- (Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia de Computação e Sistemas Digitais, BT/PCS/9903)

1. Programação paralela (Computação) 2. Equações lineares 3. Matrizes esparsas I. Sato, Liria Matsumoto II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais III. Título IV. Série

ISSN 1413-215X

CDD 005.2
515.252
512.9434

Implementação Paralela Distribuída da Dissecação Cartesiana Aninhada

Hilton Garcia Fernandes
Liria Matsumoto Sato

Resumo

A solução de sistemas de equações lineares é um problema que surge em vários algoritmos numéricos; neles a esparsidade das matrizes de coeficientes do sistema permite que se tratem sistemas de ordem muito elevada. Os métodos iterativos em geral são preferidos devido ao fato de que os métodos diretos, em sua versão mais simples, tendem a introduzir um número inaceitável de elementos não nulos na matriz do sistema, o que é chamado preenchimento, ou *fill-in*. No entanto, através de várias propriedades do grafo associado à matriz de coeficientes do sistema linear, é possível se reduzir drasticamente o preenchimento. O método de Cholesky, para a solução de sistemas lineares cuja matriz é simétrica e definida positiva, é sofisticado com técnicas da teoria dos grafos, em um algoritmo projetado especialmente para sistemas paralelos distribuídos, a dissecação cartesiana aninhada. São apresentadas estratégias para a implementação deste algoritmo.

Abstract

*T*he solution of systems of linear equations is a problem that occurs within several numerical algorithms. The sparsity of the systems coefficient matrix allows very high system orders. Usually iterative numerical methods are chosen for the systems's solution because simple direct methods tend to introduce an unacceptable number of non-zero elements in the system matrix (fill-in). However, using several properties of the graph associated to the linear system matrix, it is possible to drastically reduce the fill-in. The Cholesky method for the solution of linear systems whose matrix is symmetric positive definite, is enhanced with graph techniques, in an algorithm specifically designed for parallel distributed computers, the cartesian nested dissection. Some strategies for the implementation of this algorithm.

1 Estratégias de implementação

Este artigo discute a estratégia de implementação paralela distribuída do algoritmo da *dissecação aninhada cartesiana*, do ponto de vista das escolhas iniciais desta dissertação, que moldaram sua face antes mesmo que ela fosse iniciada. Comenta-se também a filosofia de projeto do *software*, tanto visando sua construção, quanto do ponto de vista da depuração e detecção de erros.

Considera-se que tanto o projeto formal do *software*, quanto o projeto antecipado de suas formas de depuração e detecção de erros pontos importantes da dissertação, pois o algoritmo da *dissecação aninhada cartesiana* a ser implementado por ela é relativamente complexo.

Pontos importantes para as bases do projeto são o *hardware* disponível e também o tipo de paralelismo implicado no algoritmo da *dissecação aninhada cartesiana* [4], escolhido para esta dissertação.

Ainda, neste artigo se oferece uma visão panorâmica da implementação, quanto à implementação escolhida para o algoritmo, tanto do ponto de vista da interpretação de sua especificação, como do ponto de vista das poucas alterações sugeridas para ele nesta dissertação, tanto do ponto de vista do *hardware* escolhido, quanto de muito poucas propostas de alteração.

Este artigo corresponde a uma apresentação resumida de pontos da dissertação de mestrado com mesmo nome [2].

1.1 Bases do projeto

Deseja-se com "bases do projeto" referir às bases sobre as quais todo o projeto se assentou. Isto significa tanto os "dados", as premissas iniciais dadas pela conjuntura acadêmica, como as escolhas feitas quanto à forma de se conduzir o projeto, desde a filosofia de projetar o *software*, até a forma de garantir sua execução correta, e de encontrar eventuais erros.

1.1.1 Escolhas prévias do projeto

Por escolhas prévias do projeto entendem-se as escolhas já feitas, ou "dadas", pela estrutura organizacional da academia: desde a tradição de uso de equipamentos, linguagens e sistemas operacionais, até os ambientes de apoio disponíveis. Fala-se, naturalmente, de redes de computadores, executando uma variante do *Unix*, como o *Linux*; do uso da linguagem *C* [7], e dos ambientes de apoio ao paralelismo por passagem de mensagem, como o *PVM* [1], e o *MPI* [9].

Equipamentos e sistemas operacionais Não é exagero dizer que os mais poderosos supercomputadores atuais usam arquitetura de memória distribuída. Essa forma de se organizar computadores permite que mais facilmente se agreguem processadores

e memória a um computador. Contudo, seus custos sobem proibitivamente à faixa de milhões de dólares, o que torna sua compra um privilégio de apenas umas poucas entidades de pesquisa.

Deste modo, tem sido a escolha de muitas entidades de pesquisa, nacionais ou não, construir *metacomputadores* que são, em última análise, agregados de estações de trabalho dedicadas, colocadas em rede local. Devido ao notável aumento de poder computacional e à acentuada diminuição de custos das máquinas com arquitetura *Intel*, baseadas em processadores *Pentium*, tem sido uma tendência construir metacomputadores com base em equipamentos desse tipo.

Um outro fator se soma a essa tendência: a criação e o florescimento do sistema operacional *Linux*, compatível com várias versões do Unix e também com o padrão *Posix* de sistemas abertos. Desenvolvido por um grupo de programadores na *Internet*, hoje o *Linux* é quase que um sinônimo de sistema operacional usado na academia, por ser capaz de ser executado em estações de trabalho com arquitetura *Intel*. Outros fatores para o sucesso do *Linux* na academia por sua abertura, já que todo seu código-fonte é mantido em domínio público, por sua confiabilidade e robustez, e pela facilidade com que incorpora inovações e ao mesmo tempo se mantém compatível com as tradições do sistema operacional *Unix* usado na academia.

A linguagem de programação *C* A linguagem de programção *C* [7] foi originalmente criada para o desenvolvimento de sistemas operacionais, como o *Unix*. O *C* tem boas características de portabilidade, geração de código de máquina eficiente, recursos para programação estruturada e modular, e também acesso a características do processador antes disponível apenas em linguagem *assembly*.

Por esta razão, o *C* também teve um uso intenso no desenvolvimento de aplicativos. Neste trabalho, sugere-se a linguagem de programação *C* por sua facilidade de criação de estruturas de dados, úteis para a expressão de um algoritmo relativamente complexo como o da *dissecação aninhada cartesiana* [4].

Ambientes de passagem de mensagens Sendo o *hardware* um conjunto de estações de trabalho conectadas em rede local, e sistema operacional *Linux*, quase que simultaneamente esta escolha de estratégias foi levada a um ambiente de passagem de mensagens, onde diversas estações de trabalho executam programas independentemente e trocam informações de tempo em tempo. Assim, a passagem de mensagens é tanto uma forma de compartilhar informações quanto um modo de sincronizar processos distintos em computadores distintos. Sobre a mesma arquitetura de redes de computadores usadas como um único computador, um metacomputador, tem sido construídos ambientes de *softwares* de simulação de memória compartilhada, chamados *distributed shared memory*, ou *DSM*. Apesar destas ambientes serem muito pesquisados, o algoritmo da *dissecação aninhada cartesiana* foi projetado para ambientes de memória distribuída.

Existem vários ambientes de passagem de mensagem disponíveis. Entre eles os mais famosos e utilizados recentemente têm sido o PVM [1] e o MPI [9]. Neste trabalho sugere-se o MPI devido ao fato desse ambiente ter sido criado depois da experiência do uso e implementação de muitos outros ambientes do mesmo tipo, entre eles o próprio PVM: na equipe dos criadores do MPI encontram-se também os criadores do PVM.

1.1.2 Filosofia de projeto do *software*

Nestas estratégias para a implementação paralela distribuída da *dissecação aninhada cartesiana*, faz-se o projeto do *software* segundo a orientação ao objeto [10]. Em muito poucas palavras, a visão clássica da programação orientada ao objeto, ou *object-oriented programming* (OOP), é baseada nos conceitos de

objeto – um agregado de dados isolados, ou *encapsulados*, que podem ser acessados apenas através de funções apropriadas, que podem ser usadas para criar o objeto, consultar seu estado ou alterá-lo;

atributos — são os dados de um objeto, e contém seu *estado*;

métodos — são as funções que permitem acessar a um objeto. Funções que o criam são chamadas *construtores*, ou *constructors*; funções que o destróem são chamadas *destruidores*, ou *destructors*; funções para consulta de seu estado são chamadas *funções acessadoras*, ou *accessors*;

classes — são os padrões segundo os quais os *objetos* são criados. Meyer [10] se refere às classes como fábricas de objetos. Uma classe pode reaproveitar os recursos definidos em outra; neste caso tem-se o que se chama *herança*, ou *inheritance*, que é a maior originalidade da programação orientada ao objeto, já que a maioria de seus conceitos já estavam — isoladamente — disponíveis sob outros nomes em outras técnicas; entre elas, notadamente a programação modular.

A linguagem C não tem recursos especiais para a programação orientada ao objeto; em outras palavras, não é uma *object-oriented programming language*, como a linguagem $Eiffel$ [10], ou o $Java$ [3]; ela também não é uma linguagem multi-paradigmática como o $C++$ [11]. Apesar de que com sua reconhecida flexibilidade, e seu acesso a recursos de baixo nível do processador, não é difícil implementar a própria herança em C. Isso é demontrado pelo fato de que os primeiros compiladores $C++$ e $Eiffel$ geravam programas C como resultado de compilação, em vez do código-objeto habitual. Além do mais, isso é exemplificado em detalhe em livros como [6], que propõe a programação orientada ao objeto *stricto sensu* em C.

Contudo, não sendo C uma linguagem originalmente orientada ao objeto, essas soluções em geral têm pouco apelo estético, pouca simplicidade e, em última instância, dependem mais da disciplina do programador em se comprometer com seu uso.

Devido às dificuldades implícitas na programação orientada ao objeto na linguagem C, devido ao extremo interesse de se manter a máxima simplicidade, recomenda-se uma forma muito moderada e simplificada de programação ao objeto: em primeiro lugar, não se sugere o uso de herança ou de programação genérica. Mesmo havendo muitas ocasiões em que o uso desses recursos de programação permitem um grande reuso de *software*, como nas diversas listas usadas, tais como:

- $list(V, C)$, ou lista de vértices, onde C é ou a coordenada X, ou a coordenada Y, deve ser implementada através de classe `VERTEX_LIST`, com a indicação da coordenada através de *atributo*;
- $list(E, C)$, ou lista de arestas, a ser implementada através de classe `EDGE_LIST`, e
- $count(E, C)$ e $count(V, C)$, implementadas conjuntamente através de classe `COUNT_LIST`.

Nesta dissertação sugere-se apenas o recurso de estrutura, ou `struct` da linguagem C e de funções com nomes iniciados pelo nome da classe. Por exemplo, na classe `VERTEX_TAB`, uma tabela destinada ao armazenamento de vértices, sugere-se definir uma `struct` com esse nome e definir funções para as várias necessidades da classe. Por exemplo, a construção de objetos desta classe deve ser feita pela função

```
RET_VAL EDGE_TABcreat (VERTEX_TAB **this, FILE *inp_f);
```

A impressão deve ser feita pela função

```
RET_VAL EDGE_TABprint (VERTEX_TAB *this, FILE *out_f);
```

Através da convenção que relembra a linguagem $C++$, usa-se o nome `this` para indicar o objeto sendo manipulado. Ele é passado como

1. um apontador nos casos em que ele é consultado ou seus atributos são alterados, como em `VERTEX_TABprint()` acima, ou
2. um endereço de apontador nos casos em que é criado, como em `RET_VAL VERTEX_TABcreat (VERTEX_TAB **this, FILE *inp_f)`, ou então destruído, como o é em `RET_VAL VERTEX_TABdestruct (VERTEX_TAB **this)`.

O tipo `RET_VAL`, usado como valor de retorno da maioria das funções, também é uma classe, a ser usada fortemente nas estratégias de detecção de erros, que são apresentadas no item 1.1.3.

1.1.3 Técnicas de depuração e detecção de erros

Abordam-se aqui as estratégias de depuração e de detecção de erros. Um projeto de *software* de algum porte, como o da *dissecação aninhada cartesiana* deve prever essas estratégias desde sua concepção.

Sobre a detecção de erros Sendo relativamente complexo, o algoritmo da *dissecação cartesiana aninhada* enseja desde logo o desenvolvimento de técnicas para a detecção de erros. Sugere-se a noção de *pré-condições* e *pós-condições* de Meyer [10].

Nessa técnica, garante-se que quando na entrada de uma função, todos seus parâmetros atendam a condições de consistência; caso contrário, a função não é executada e o programa é interrompido; isto é chamado de *pré-condição*. Do mesmo modo, uma função deve garantir que à sua saída, os dados por ela gerados ou alterados atendam a certas condições de consistência; isto é chamado de *pós-condição*. Em geral, fazem-se execuções com de teste com todas verificações habilitadas. Naturalmente, a detecção de erros toma tempo de processamento e, por isso, tendo-se detectado os erros mais importantes, e o programa se tornado mais estável, o mecanismo de detecção de erros pode ser desligado mediante uma opção de compilação, sem nenhuma alteração no texto-fonte do programa.

O não atendimento a pré ou pós-condição é um erro do programa, que gera uma *exceção*, que pode ser tratada por mecanismos específicos de linguagens como o $Eiffel$, de Meyer [10], ou o $C++$, de Stroustrup [11].

Nesta propõe-se o uso da linguagem de programação C (ver item 1.1.1, na página 2), que não tem explicitamente um mecanismo para lidar com exceções, apesar de possuir funções e recursos, como `setjmp()` e `longjmp()` [7], que podem ser usados para implementá-lo.

Preferindo-se sempre que possível a simplicidade, sugere-se implementar a detecção de erros através do uso do pré-processador embutido na linguagem C. O recurso de detecção de erros permite apenas que a pilha de chamadas de funções executadas até o momento seja desfeita — o que é chamado de *stack unfolding* — e que sejam impressas mensagens descrevendo o erro e a linha do programa onde ele ocorreu. Este processo chega até a função principal e o programa é terminado.

Sugere-se implementar esse recurso através de macros de pré-processador, chamadas de *retorno condicional*: quando ocorrer uma condição de erro, a função deve retornar para a outra que a chamou. Nesta função há outra macro de retorno condicional, que verifica que a função chamada não retornou com o valor correto. Isto faz com que a função chamadora também retorne. E o processo de retorno condicional continua até que todo o programa seja terminado. Sugere-se que nas macros de retorno condicional sejam implementados recursos que permitam que cada retorno condicional gere mensagem informando qual o tipo de erro e a linha de programa onde ele ocorreu.

O uso de macros criadas pelo pré-processador embutido na linguagem *C* permite que as definições das macros possam ser habilitadas ou desabilitadas por opções de compilação: assim, do mesmo modo que em *Eiffel* [10], a detecção de erros pode ser integralmente desabilitada — sem nenhuma alteração no texto do programa-fonte —, gerando-se uma versão de execução mais rápida, a ser usada para medidas de desempenho.

Sugere-se também um estágio intermediário de detecção de erros: também por opção de compilação, pode-se permitir que apenas um ou mais tipos de erros sejam detectados, sendo os outros ignorados.

Os tipos de erros devem ser padronizados em uma enumeração, a ser chamada `RET_VAL`, de *RETurn VALue*. Na classe de funções `RET_VAL`, devem existir funções que permitam decodificar um valor de retorno. Assim, em vez do valor numérico do código de erro imprime-se uma mensagem descrevendo em termos gerais o erro detectado.

A título de exemplo, eis uma seqüência possível de detecção de erros, aplicado à criação de uma tabela de vértices, a ser implementada pela classe `VERTEX_TAB`, disparada a partir da impressão de um grafo, implementado pela classe `GRAPH`. Eis como deve ser a verificação de erros através da checagem de parâmetros na função-membro `VERTEX_TABprint`:

```
RET_VAL VERTEX_TABprint ( const VERTEX_TAB *this
                        ,   FILE           *out_f
                        )
{
    DECLARE_RET_VAL

    cret_INVPARM (this == NUL);
    cret_INVPARM (out_f == NULL);
    cret_INCONS (feof (out_f));
    cret_INCONS (ferror (out_f));
...
    return (RV_OK);
}
```

Neste trecho de programa faz-se a declaração de variáveis necessárias ao pacote de detecção de erros, através da macro `DECLARE_RET_VAL`. Verifica-se se não são nulos os apontadores para o objeto da classe `VERTEX_TAB`, de nome `this`, e para o arquivo de impressão, de nome `out_f`. Se algum desses apontadores for nulo, a função que chama esta recebe como retorno o valor `RV_INVPARM`, que significa que a função chamada recebeu um parâmetro inválido. Além disso, verifica-se se o arquivo de saída não está exaurido ou sob alguma condição de erro.

Caso todos os testes sejam corretos, a função retorna chega até sua última linha, quando retorna o valor `RV_OK`, que significa que todo o processamento foi realizado

corretamente.

Por sua vez, na função `GRAPHprint()`, onde são impressos grafos, deve haver uma linha da forma:

```
confe_OK (VERTEX_TABprint (this->vt, out_f));
```

Neste caso, busca-se a confirmação de que a função chamada terminou corretamente, retornando `RV_OK`. A letra `e`, no prefixo `confe_`, não é casual: ela significa que o parâmetro da macro deve ser *expandido* mesmo se a detecção de erros não estiver habilitada. Neste caso, a linha acima se transforma simplesmente em:

```
VERTEX_TABprint (this->vt, out_f);
```

Isto é, em uma impressão normal do objeto `VERTEX_TAB`.

Caso haja um erro, a função chamadora `GRAPHprint()` retorna a quem a chamou, que pode ser um objeto da classe `GRAPH_TREE`, que implementa o conceito de árvore de grafos. E neste caso, também a função chamadora, agora da classe `GRAPH_TREE` retorna a quem a tenha chamado. Neste caso o programa principal, chamado `main()` em *C*. Neste ponto, o programa termina. Não sem que tenham sido impressas mensagens informando as linhas onde os *retornos condicionais* ocorreram. E a condição que levou a esse retorno, que no exemplo dado pode ser `RV_INVPARM` — para parâmetros inválidos — ou `RV_INCONS`, caso um dos parâmetros seja inconsistente, apesar de aparentemente válido.

Sobre a depuração Na criação dos recursos de depuração sugere-se o uso de uma técnica similar à da detecção de erros: sugere-se a criação de uma biblioteca de depuração, chamada, por exemplo, `dbglib`, que conte com funções de impressão de valores e mensagens, e com um nível de pré-processamento; as macros devem possuir a mesma funcionalidade, o mesmo nome e argumentos das funções, mas em maiúsculas. Por exemplo, deve existir uma função chamada `dl_print_int()`, para a impressão de expressões de tipo `int` e de uma mensagem definindo-a, e macros como `DL_PRINT_INT()`.

As funções não devem ser chamadas diretamente, mas através das macros. Por sua vez, quando a opção de depuração está habilitada, a macro se expande para a função de depuração de nome equivalente. Quando não, a macro gera o texto nulo. Ou em outras palavras, não gera nenhuma instrução.

Assim, uma linha de programa da forma

```
DL_PRINT_INT ("ind_col", ind_col));
```

depois do preprocessamento se expande como

```
dl_print_int ("ind_col", ind_col));
```

se a depuração estiver habilitada; e na linha nula, se a depuração não estiver habilitada.

1.2 O algoritmo paralelo

Neste ponto faz-se uma apresentação detalhada do algoritmo da *dissecação aninhada cartesiana* [4], adaptado às condições pré-definidas para a realização destas estratégias.

Nesta parte do texto o objetivo não é a apresentação dos conceitos envolvidos na *dissecação aninhada cartesiana*, mas a apresentação do algoritmo paralelo, especialmente adaptado ao metacomputador que será usado. A ênfase é no paralelismo do processo. Partes do algoritmo que são seqüenciais não recebem a mesma atenção de partes paralelas.

1.2.1 Iniciação

Na *Iniciação*, como sugere o nome, são efetuados os passos que permitem que o processamento seja iniciado. Neste caso, isto significa ler os valores dos dados do problema, distribuí-los aos processadores e criar estruturas de dados que permitam o trabalho coordenado dos processadores para atingir a meta da reordenação da matriz de coeficientes do sistema segundo o algoritmo da dissecação aninhada cartesiana.

1.2.2 Leitura

Na fase de leitura, um processador centraliza o processo de leitura do arquivo de dados de aresta e vértice de grafo a ser separado. A seguir, esse processador — chamado de *mestre*, ou *master* — envia os dados para os outros processadores, que são usualmente chamados *escravos*, ou *slaves*.

Eis os passos de leitura da fase de inicialização do algoritmo. Para uma compreensão da leitura ocorrendo em paralelo no processador mestre e nos escravos, seus passos são mostrados em itens distintos.

1.2.3 Ordenação local

Fase inteiramente local, como seu nome sugere ... Nesta fase são criadas duas listas com valores de coordenadas em X e Y de cada um dos vértices residentes em cada processador. A seguir essas listas são ordenadas independentemente. As regiões de memória das duas listas são contíguas, para que os dados locais de um processador possa ser enviado em um único pacote de dados, através de uma única mensagem.

1.2.4 Inteirização e Atribuição das faixas de valores

Processamento hierárquico, dividido em passos, onde a cada passo, as informações são concentradas. Ao final desta fase, os vértices têm suas coordenadas "inteirizadas", ou *integerized*.

Em vez de se usar uma linguagem semelhante às linguagens de programação para descrever o algoritmo, faz-se antes a apresentação do algoritmo de modo informal, para os vários passos.

A título de exemplo, usam-se 8 processadores, mas qualquer outro número poderia ser usado. No entanto, a exposição do algoritmo é facilitada com quantidades de processadores que sejam potências inteiras de 2.

Envio para os representantes: O envio para os representantes pode ser entendido como a criação de um conjunto de vértices através da união de seus subconjuntos de 2 em 2. Num primeiro passo, são unidos dois dos subconjuntos destinados a cada processador.

Por exemplo, o envio de π_1 para π_2 equivale a se fazer uma união $V(\pi_1) \cup V(\pi_2)$.

No Passo 2 a seguir, o envio de π_2 para π_4 equivale a se fazer a união $(V(\pi_1) \cup V(\pi_2)) \cup (V(\pi_3) \cup V(\pi_4))$. De maneira geral, no Passo i são unidos 2^i subconjuntos.

Inteirização *stricto sensu* pelo representante geral: Nesta fase, estando de posse de todos os valores de coordenadas, o representante geral promove o que de fato é a inteirização, ou a transformação posicional de coordenadas em valores fracionários, ou de "ponto-flutuante", para coordenadas inteiras.

Esta fase é realizada unicamente pelo representante geral, que a realiza de modo inteiramente seqüencial.

Envio para os representados: Nesta fase, é efetuado o envio das listas de coordenadas inteirizadas para os processadores. Este também é um processo hierárquico, onde no Passo 3 o representante geral passa metade das informações para seu representado direto e retém para si a outra metade. A seguir as informações são de novo repartidas em dois, até que todos processadores recebam suas informações.

1.2.5 Identificação da coordenada separadora

Nesta grande seção do algoritmo paralelo, é identificada a coordenada separadora, que permitirá, na fase seguinte a criação do separador.

Listas de contagem locais Nesta etapa, cada nó calcula, localmente, para cada coordenada do problema, sua lista de contagem de vértices, e de arestas, esta com os itens $(i, \beta_i, \epsilon_i, \sigma_i)$, onde β_i é o número de arestas iniciadas na coordenada de valor i; ϵ_i é o número de arestas terminadas no dado valor i; e σ_i é o número de arestas "atravessadas", ou *straddled*, no dado valor i.

As listas de contagem de arestas e de vértices são enviadas conjuntamente. Isto é: para cada coordenada envia-se a quíntupla ordenada de valores $(i, c_i, \beta_i, \epsilon_i, \sigma_i)$,

onde β_i, ϵ_i e σ_i são definidos como antes e c_i é a contagem de vértices na coordenada i.

Isto permite minimizar o número de mensagens enviadas e maximizar a eficiência do envio, minimizando o tempo de *overhead* no estabelecimento de uma mensagem.

Comunicação de listas de contagem locais Neste passo é feita a comunicação das listas de contagem, nos moldes das comunicações hierárquicas anteriores. Depois deste passo do algoritmo cada processador tem os itens da lista de contagem das faixas de valores das coordenadas que lhe correspondem.

Transmissão de representados para representantes: Neste ponto a informação é acumulada nos representantes, até o ponto máximo onde apenas um representante geral concentra toda informação de listas de contagens de vértices e arestas ao longo dos diferentes valores de cada uma das coordenadas.

Transmissão de representantes para representados: Neste ponto do algoritmo, ocorre apenas envio e recepção de informações. Não é feito nenhum cálculo ou comparação.

O objetivo é fazer chegar a cada processador as informações referentes a contagens de vértices e arestas de seus dois blocos de coordenadas, tanto referentes a X quanto a Y.

Cálculo local da melhor coordenada separadora Com os dados das listas de contagem, cada processador tem condições de avaliar quais valores de cada coordenada satisfazem a equação de balanço e de, além disso, definir quais desses valores têm o menor valor possível da função η.

Comunicação de coordenadas separadoras Tendo cada processador estimado seu candidato a separador, esses valores são enviados aos outros, de novo num esquema hierárquico de comunicação. As informações têm o formato $(coord, i, \eta(i))$, onde *coord* é o eixo de coordenadas escolhido, ou x, ou y; i é o valor nessa coordenada que satisfaz a equação de balanço e que gera o menor valor da função η nos dois intervalos de coordenadas do processador, seja na coordenada x, seja na y. Para que possa ser feita a comparação com os valores de outros processadores, também é enviado $\eta(i)$, o valor da função η calculado para i.

Depois dessas informações terem chegado ao representante geral, este tem condições de decidir por uma dada coordenada, e por um dado valor dela. Isto é então comunicado a todos outros processadores.

Cálculo do conjunto corretor Neste caso, a introdução do paralelismo oferece uma dificuldade: normalmente seria necessário que os processadores comunicassem entre si os conjuntos corretores, para minimizar seu tamanho. Isto é, como observam Heath; Raghavan [4], se duas arestas (u, v) e (u, w) são atravessadas pela coordenada s escolhida, poder-se-ia incluir apenas o vértice u no conjunto corretor, em vez de dois vértices. Isto, naturalmente, só seria possível se cada processador π_i comunicasse com os outros seu conjunto $E_s(\pi_i)$, de arestas atravessadas por s.

Neste trabalho, inicialmente se preferirá a alternativa de Heath; Raghavan, pois mesmo na implementação seqüencial prefere-se um balanço entre os dois subgrafos à direita e à esquerda de onde são retirados os vértices para o conjunto corretor. Quando da seleção entre vértices das arestas atravessadas, sorteia-se aleatoriamente um deles.

Apesar dessa estratégia não dar origem aos menores separadores possíveis, ela dá origem a subgrafos mais balanceados em termos de tamanho, o que é desejável em termos de processamento paralelo.

Uma alternativa a ser considerada para implementação seria os custos de um trecho de algoritmo que visasse a criação de um conjunto separador. Apesar de haver um custo de comunicação inegável, talvez seja interessante avaliar qual a qualidade dos separadores obtidos desse modo. Não é impossível que uma melhoria na qualidade dos separador obtidos compense os custos de obtê-la.

Mesmo no caso em que o conjunto corretor é calculado localmente, é necessária ainda uma fase de comunicação. Depois de calculados os conjuntos corretores, cada processador π_i deve comunicar a seus vizinhos qual o tamanho de seu conjunto separador. Isso é necessário para que a fase de renumeração dos vértices possa ser feita.

Aqui é usada outra decisão de Heath; Raghavan [4]: como a ordem de renumeração de vértices é arbitrária, renumeram-se os vértices seguindo a ordem dos processadores. Isto é, os vértices atribuídos ao processador π_i precedem os vértices do processador π_{i+1}, que precedem os vértices do processador π_{i+2} etc.

Uma vez mais essa decisão é tomada para minimizar a comunicação necessária.

Cálculo de conjunto corretor O cálculo do conjunto corretor C_s é relativamente simples: identificam-se arestas que são atravessadas pela coordenada separadora escolhida s. Isto é, arestas que se iniciam num vértices à direita da coordenada separadora e terminam à esquerda dela. Supondo que tenha sido escolhido um valor s da coordenada x, a idéia é localizar arestas (u, v) nas quais seja verdade que $(u.x < s)$ e $(v.x > s)$.

Nessas arestas, escolhe-se arbitrariamente um dos vértices e ele passa a fazer parte de C_s.

Cálculo dos conjuntos de vértices corrigidos Neste ponto, busca-se calcular localmente para cada processador π_i seus conjuntos $V_1(\pi_i)$, $V_s(\pi_i)$ e $V_2(\pi_i)$, que serão, respectivamente, o conjunto de vértices do subgrafo à direita G_1, o conjunto separador e o conjunto de vértices do subgrafo à direita G_2.

Dados os conjuntos U_1, U_s e U_2, respectivamente os conjuntos de vértices com coordenadas menores do que s, iguais a s e maiores do que s, a correcção significa retirar de U_1 e U_2 alguns dos vértices das arestas atravessadas, contidos no conjunto corretor C_s, acrescentando-os a U_s. Formalmente,

$$V_1 = U_1 - C_s$$
$$V_s = U_s \cup C_s$$
$$V_2 = U_2 - C_s$$

Como isso é feito localmente, na verdade em cada processador π_i tem-se os conjuntos $V_1(\pi_i)$, $V_s(\pi_i)$ e $V_2(\pi_i)$, construídos a partir do corretor local $C_s(\pi_i)$.

Comunicação do tamanho dos conjuntos separadores locais: Neste item cada processador π_i comunica aos outros o tamanho da parte do conjunto separador que lhe cabe, ou seja $|V_s(\pi_i)|$. Conforme comentado, isto é necessário para que cada processador saiba que número usar para renumerar os vértices de seu subconjunto.

A renumeração dos vértices do conjunto separador é arbitrária: a heurística da dissecação aninhada não especifica uma forma de se renumerá-los. Contudo, neste texto segue-se a mesma estratégia recomendada por [4]: os vértices do subconjunto separador em π_1, chamados $V_s(\pi_1)$, são renumerados depois dos vértices em $V_s(\pi_2)$, que são renumerados depois dos vértices de $V_s(\pi_3)$, e assim por diante. Isto significa que os vértices em $V_s(\pi_1)$ são renumerados de

$$|V_s| - |V_s(\pi_1)| + 1 \quad a \quad |V_s| \ .$$

Por sua vez, os vértices em $V_s(\pi_2)$ serão renumerados

$$|V_s| - |V_s(\pi_1)| - |V_s(\pi_2)| + 1 \quad a \quad |V_s| - |V_s(\pi_1)| \ .$$

De maneira geral, os vértices do subconjunto separador contidos no processador π_i, ou $V_s(\pi_i)$, são renumerados de

$$|V_s| - |V_s(\pi_1)| - |V_s(\pi_2)| - \ldots - |V_s(\pi_{i-2})| + 1$$

a

$$|V_s| - |V_s(\pi_1)| - |V_s(\pi_2)| - \ldots - |V_s(\pi_{i-2})| - |V_s(\pi_{i-1})| \ .$$

Naturalmente, i deve estar entre 1 e P, o número de processadores disponíveis para o problema.

Para que a renumeração dos vértices em $V_s(\pi_1)$ seja possível, basta que π_1 receba $|V_s|$, o tamanho total do conjunto separador. Contudo, além de $|V_s|$, π_2 deve receber também $|V_s(\pi_1)|$, π_3 deve receber $|V_s(\pi_1)| + |V_s(\pi_2)|$. E, de modo geral, π_i deve receber $|V_s|$ e $|V_s(\pi_1)| + |V_s(\pi_2)| + \ldots + |V_s(\pi_{i-1})|$.

A seguir, no detalhamento do algoritmo, vê-se como a técnica da comunicação hierárquica pode ser usada para fazer chegar essas informações aos processadores.

1. Comunicação de representantes a representados

   Neste ponto, como de hábito, os processadores representantes acumulam informação, selecionam o que deve ser enviado a seus representantes no passo seguinte e o fazem.

   Esses procedimentos são repetidos até que um único processador, o representante geral acumule toda informação. A partir daí acontece o processo inverso e os representantes passam a fornecer informações selecionadas a seus representados.

2. Comunicação de representantes a representados

   Neste ponto, os processadores representantes enviam a seus representados o valor geral $|V_s|$ e os valores das somas parciais dos subconjuntos anteriores em cada um dos 3 passos da Comunicação de representantes a representados

   Antes de se fazer o detalhamento do algoritmo desta etapa, é interessante tentar definir o que é uma soma parcial dos subconjuntos anteriores de um dado passo.

   Na etapa de concentração de informações o fluxo de informações sempre é feito de processadores de índice menor, os representados, para aqueles de índice maior, os representantes. Na fase de dispersão, o movimento é no sentido inverso, dos representantes, com índice maior para os representados, com menor índice.

   Assim, num dado passo, cada representante tem as informções dos representantes de menor índice até o seu próprio. No passo de número 1, o processador representante tem informações sobre o processador imediatamente menor; no Passo 2 o representante tem informações sobre 3 processadores menores. De modo geral, no passo i cada representante tem informações sobre $2^i - 1$ processadores de índices menores do que os dele.

   Deste modo, em um dado passo i, subsoma do passo anterior é o número de $2^{i-1} - 1$ tamanhos de conjunto separador somados ao tamanho de seu próprio separador; em outras palavras, o tamamho de 2^{i-1} separadores. Simbolicamente, uma subsoma de tamanhos de separadores de um passo n de um processador π_j é:

$$|V_s(\pi_{j-2^n+1})| + |V_s(\pi_{j-2^n+2})| + \ldots + |VS\pi_j|$$

1.2.6 Construção paralela de um separador

Já que todos os conjuntos envolvidos na separação foram convenientemente tornados locais, e divididos entre cada processador participante na separação, o processo de construção de um separador se torna inteiramente paralelo, sem nenhuma necessidade de comunicação. Na terminologia de alguns autores, a construção paralela de um separador seria um processo *embarrassingly parallel*, ou "embaraçosamente paralelo".

1.2.7 Renumeração dos vértices de um separador

No caso da primeira separação de vértices, na qual o grafo inicial G_0 dá origem a dois subgrafos G_1 e G_2, o conjunto separador V_s é renumerado como segue. Em primeiro lugar,

$$V_s = V_s(\pi_1) \cup V_s(\pi_2) \cup \ldots \cup V_s(\pi_P)$$

Deste modo, $|V_s|$ — o número de elementos de V_s — é a soma do número de elementos dos subconjuntos locais em cada processador.

Como, de acordo com a técnica da dissecação aninhada cartesiana, os vértices do separador são renumerados com os maiores valores de índice disponíveis, tem-se para o primeiro separador que o maior valor disponível é mesmo o número de elementos do conjunto de vértices V, já que $G_0 = G = (V, E)$. Deste modo, os vértices do separador são renumerados de $|V| - |V_s| + 1$ a $|V|$. Deste modo, $V_s(\pi_1)$, ou os vértices do separador contido no processador π_1 são renumerados de $|V| - |V_s| + 1$ a $|V| - |V_s| + |V_s(\pi_1)|$.

Por sua vez, os vértices do separador contidos no processador π_i são renumerados de

$$|V| - |V_s| + |V_s(\pi_1)| + |V_s(\pi_2)| + |V_s(\pi_3)| + \ldots + |V_s(\pi_{i-1})| + 1$$

a

$$|V| - |V_s| + |V_s(\pi_1)| + |V_s(\pi_2)| + \ldots + |V_s(\pi_{i-1})| + |V_s(\pi_i)|.$$

1.2.8 Generalização do algoritmo para vários grafos

Para não carregar a notação, toda a apresentação do algoritmo foi feita em termos de apenas um grafo, o grafo inicial, $G = G_0 = (V, E) = (V_0, E_0)$. Na verdade, o interessante do algoritmo é a completa dissecação do grafo, o que só é conseguido após vários passos de separação.

Assim, no passo seguinte de separação haverá dois grafos que se deverá considerar: G_1 e G_2, resultados da separação de G_0. Como G_1 usa os vértices à direita da

coordenada separadora s escolhida no primeiro passo de separação, $G_1 = (V_1, E_1)$. Do mesmo modo, $G_2 = (V_2, E_2)$.

A separação de G_1, através de uma coordenada separadora e seu valor t, dará origem a um conjunto separador V_{1t}. A coordenada separadora de G_2 em geral será outra, de valor r. Assim, a separação de G_2 dará origem a um outro separador V_{2r}.

Como as duas separações estão sendo feitas simultaneamente, neste caso é necessário é informar nas trocas de mensagem a que grafo se referem as contagens e os valores de separação que estão sendo enviados.

A seguir é feita uma reapresentação do algoritmo para generalizá-lo para vários subgrafos sendo simultaneamente separados. Como os passos do algoritmo já foram apresentados em suficiente detalhe para a separação de apenas um grafo, agora apenas informaremos as alterações necessárias para tratar com múltiplos grafos simultaneamente nas estruturas de dados enviados e nos passos do algoritmo.

1.3 Enumeração das modificações sugeridas

Nesta parte do texto são apresentadas modificações efetuadas nesta dissertação em relação à especificação do algoritmo da *dissecação aninhada cartesiana* [4]. Elas se dividem em duas partes: modificações devidas ao *hardware* diferente utilizado e modificações experimentais, onde pontos do algoritmo original da dissecação aninhada cartesiana são alterados e os resultados obtidos são comparados com o original.

1.3.1 Modificações devido ao *hardware* distinto

Aqui se comentam as alterações efetuadas no algoritmo devidas ao uso de equipamento distinto. Por exemplo, em [4] os autores propõem um algoritmo para a transmissão de informações que usa a riqueza de caminhos alternativas de comunicaçõa disponível na arquitetura hipercubo. Essa mesma riqueza não estando disponível nas maquinas usadas, é preferível desenvolver algoritmos que levem em conta a limitação das comunicações em uma rede *Ethernet*.

Em termos muito simplificados, uma rede *Ethernet* pode ser descrita como sendo um canal seqüencial de informações, um "fio" onde apenas uma informação pode trafegar a cada vez [12]. Recursos como *switches* e *hubs* sofisticados podem mudar esse panorama, mas não estão ainda amplamente disponíveis. Ademais, seu resultado prático certamente não leva a riquezas de recursos de comunicação comparáveis àquelas de arquiteturas como a malha, ou *mesh*, e o hipercubo, uma vez que essas arquiteturas são altamente otimizadas e dimensionadas para alto desempenho, sem limitações de custo. Ao passo que acessórios para redes locais têm pelo menos a limitação de custo.

Assim, buscou-se algoritmos que minimizassem o número de mensagens que tem de ser enviadas a cada passo. Mesmo que a um custo maior do tamanho dessas mensagens. Contudo, mesmo assim foi possível obter eficiência maior do que aquela

do envio linear puro e simples: todos processadores enviam informações para todos os outros. Esse tipo de comunicação entre todos os processadores de um conjunto é chamado *all-to-all broadcast* na literatura de processamento paralelo [8], mas de *gossiping* na literatura de redes [5].

A título de comparação, para a troca de informações entre P processadores, onde $P = 2^k$ é uma potência inteira de 2, pela técnica de envio linear, cada processador envia aos $P - 1$ outros uma mensagem de, digamos, comprimento n bytes. Deste modo, são enviadas $P \times (P - 1)$ mensagens, num total de $P \times (P - 1) \times n$ bytes comunicados. A operação pela qual um processador envia informações para um grupo de outros é chamada *broadcast*.

Teoricamente seria possível otimizar o desempenho para redes *Ethernet*, nas quais existe uma primitiva de comunicação que permite o envio de apenas uma mensagem, na qual um *header* especial informa quais processadores devem recebê-la. Portanto, uma única mensagem pode ser recebida pelos $P-1$ processadores. Assim, seria possível fazer o *all-to-all broadcast* com apenas $P-1$ mensagens, transmitindo-se um total de $n \times (P - 1)$ bytes.

Contudo, os ambientes de passagem de mensagens, como o *MPI* [9] e o *PVM* [1], não costumam ser otimizados para o uso de redes locais. Assim, seu *broadcast* acaba correspondendo ao envio de $n - 1$ mensagens.

Projetou-se um algoritmo, chamado *concentração - dispersão*, que foi exposto em vários itens da seção 1.2. Basicamente ele corresponde a se agrupar informações em grupos de 2 processadores, depois em grupos de 4, depois de 8 etc. Supondo que P, o número de processadores seja uma potência inteira de dois, na etapa de *concentração* são necessários $log_2 P = k$ passos para que um único processador, chamado *representante geral*, concatene informações de todos outros processadores.

A partir daí, começa a etapa de *dispersão*: os representatnte geral envia metade das informações com outro processador; no passo seguinte esses dois processadores dividem metade das informações com mais dois outros processadores. E assim, depois de k passos, toda informação está distribuída pelos P processadores.

Como as duas partes do algoritmo são simétricas, pode-se fazer análise apenas da fase de concentração. No primeiro passo da concentração, são enviadas $\frac{P}{2}$ mensagens com comprimento n cada uma. No segundo passo, são enviadas $\frac{P}{4}$ mensagens, com comprimento $2n$. E assim por diante, até o passo k, quando é enviada apenas uma mensagem, com comprimento $\frac{P}{2} \times n$. Isto significa um total de $P - 1$ mensagens, com um volume transferido de $\frac{k \times P \times n}{2}$ bytes.

Deste modo, para o algoritmo como um todo, são $2(P - 1)$ mensagens, com um volume de $k \times P \times n$ bytes.

Para fixar idéias, supondo $P = 2^3 = 8$, o volume no caso dos *broadcasts* está em $P \times (P - 1) = 8 \times 7 = 56$ mensagens, com $56n$ bytes transmitidos. No caso da concentração - dispersão, esse número é de $2(P - 1) = 2 \times 7 = 14$ mensagens, com um total de $3 \times 8 \times n = 24n$ bytes transferidos.

1.3.2 Modificações experimentais

Em alguns trechos do algoritmo, parece ser interessante alterar as abordagens seguidas. Por exemplo, Heath; Raghavan [4] sugerem uma forma muito peculiar de função η para estimar o tamanho do conjunto corretor. Essa forma de função depende da comunicação de dois valores de contagem para cada valor s de uma dada coordenada, chamados β_s e ϵ_s.

A única vantagem dessa forma da função parece ser minorar levemente a quantidade de informação transmitida entre processadores. Ora, uma análise simples mostra que a economia é muito pequena. E que além disso, a cada passo de comunicação é necessário transmitir e receber dois valores, β_s e ϵ_s.

Assim, parece ser interessante avaliar o uso de um único parâmetro, que foi chamado σ_s, correspondente ao tamanho do conjunto E_s de arestas atravessadas pela coordenada s. Qual o custo em termos do algoritmo dessa alternativa? Qual o impacto na qualidade dos separadores?

Referências

[1] Anonymous: *An Introduction to PVM Programming.* `http://www.epm.ornl.gov/pvm/intro.html`. 1996;

[2] Estratégias para implementação paralela-distribuída da dissecação aninhada cartesiana. São Paulo, Depto. Engenharia da Computação e Sistemas Digitais, Escola Politécnica da Universidade de São Paulo, 184 + 11 pp. 1998;

[3] J. Gosling; B. Joy; G. L. Steele: *The Java language specification*; `ftp://ftp.javasoft.com/docs/specs/langspec-1.0.pdf`; *xxv* + 825; 1996;

[4] M. T. Heath; P. Raghavan: *A cartesian parallel nested dissection algorithm.* University of Illinois at Urbana-Champaign. Technical Report UIUDCS-92-1772, **18** pp. 1992;

[5] S. M.Hedetniemi; S. T. Hedetniemi: *A survey of Gossiping and Broadcasting in Communication Networks.* Networks, 18(4), pp. 319 – 349; 1988;

[6] A. Holub: *C + C + +: Programming with objects in C and C + +.* New York, McGraw-Hill; *xiv* + 427 pp.; 1992;

[7] B. W. Kernighan; D. M. Ritchie: *The C programming language, 2nd ed.* Murray Hill, Prentice-Hall; *xii* + 272 pp.; 1988;

[8] V. Kumar; A. Grama; A. Gupta; Ge. Karypis: *Introduction to parallel computing: design and analysis of algorithms.* Redwood City, Benjamin-Cummings, *xv* + 599; 1996;

[9] Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard.* `ftp://netlib.org/tennessee/ut-cs-94-230.ps`; 1994;

[10] B. Meyer: *Object-oriented software construction*; Hemel-Hempstead; Prentice-Hall; $xviii + 534$ pp.; 1988;

[11] B. Stroustrup: *The C++ programming language, 2nd ed.* Reading, Addison-Wesley, $xi + 699$. 1991;

[12] A. S. Tanenbaum: *Computer Networks*, 2nd. ed. New Jersey, Prentice-Hall, $xv + 658$ pp., 199;

Índice

BOLETINS TÉCNICOS - TEXTOS PUBLICADOS

BT/PCS/9301 - Interligação de Processadores através de Chaves Ômicron - GERALDO LINO DE CAMPOS, DEMI GETSCHKO

BT/PCS/9302 - Implementação de Transparência em Sistema Distribuído - LUÍSA YUMIKO AKAO, JOÃO JOSÉ NETO

BT/PCS/9303 - Desenvolvimento de Sistemas Especificados em SDL - SIDNEI H. TANO, SELMA S. S. MELNIKOFF

BT/PCS/9304 - Um Modelo Formal para Sistemas Digitais à Nível de Transferência de Registradores - JOSÉ EDUARDO MOREIRA, WILSON VICENTE RUGGIERO

BT/PCS/9305 - Uma Ferramenta para o Desenvolvimento de Protótipos de Programas Concorrentes - JORGE KINOSHITA, JOÃO JOSÉ NETO

BT/PCS/9306 - Uma Ferramenta de Monitoração para um Núcleo de Resolução Distribuída de Problemas Orientado a Objetos - JAIME SIMÃO SICHMAN, ELERI CARDOSO

BT/PCS/9307 - Uma Análise das Técnicas Reversíveis de Compressão de Dados - MÁRIO CESAR GOMES SEGURA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9308 - Proposta de Rede Digital de Sistemas Integrados para Navio - CESAR DE ALVARENGA JACOBY, MOACYR MARTUCCI JR.

BT/PCS/9309 - Sistemas UNIX para Tempo Real - PAULO CESAR CORIGLIANO, JOÃO JOSÉ NETO

BT/PCS/9310 - Projeto de uma Unidade de Matching Store baseada em Memória Paginada para uma Máquina Fluxo de Dados Distribuído - EDUARDO MARQUES, CLAUDIO KIRNER

BT/PCS/9401 - Implementação de Arquiteturas Abertas: Uma Aplicação na Automação da Manufatura - JORGE LUIS RISCO BECERRA, MOACYR MARTUCCI JR.

BT/PCS/9402 - Modelamento Geométrico usando do Operadores Topológicos de Euler - GERALDO MACIEL DA FONSECA, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9403 - Segmentação de Imagens aplicada a Reconhecimento Automático de Alvos - LEONCIO CLARO DE BARROS NETO, ANTONIO MARCOS DE AGUIRRA MASSOLA

BT/PCS/9404 - Metodologia e Ambiente para Reutilização de Software Baseado em Composição - LEONARDO PUJATTI, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9405 - Desenvolvimento de uma Solução para a Supervisão e Integração de Células de Manufatura Discreta - JOSÉ BENEDITO DE ALMEIDA, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9406 - Método de Teste de Sincronização para Programas em ADA - EDUARDO T. MATSUDA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/9407 - Um Compilador Paralelizante com Detecção de Paralelismo na Linguagem Intermediária - HSUEH TSUNG HSIANG, LÍRIA MATSUMOTO SAITO

BT/PCS/9408 - Modelamento de Sistemas com Redes de Petri Interpretadas - CARLOS ALBERTO SANGIORGIO, WILSON V. RUGGIERO

BT/PCS/9501 - Sintese de Voz com Qualidade - EVANDRO BACCI GOUVÊA, GERALDO LINO DE CAMPOS

BT/PCS/9502 - Um Simulador de Arquiteturas de Computadores "A Computer Architecture Simulator" - CLAUDIO A. PRADO, WILSON V. RUGGIERO

BT/PCS/9503 - Simulador para Avaliação da Confiabilidade de Sistemas Redundantes com Reparo - ANDRÉA LUCIA BRAGA, FRANCISCO JOSÉ DE OLIVEIRA DIAS

BT/PCS/9504 - Projeto Conceitual e Projeto Básico do Nível de Coordenação de um Sistema Aberto de Automação, Utilizando Conceitos de Orientação a Objetos - NELSON TANOMARU, MOACYR MARTUCCI JUNIOR

BT/PCS/9505 - Uma Experiência no Gerenciamento da Produção de Software - RICARDO LUIS DE AZEVEDO DA ROCHA, JOÃO JOSÉ NETO

BT/PCS/9506 - MétodOO - Método de Desenvolvimento de Sistemas Orientado a Objetos: Uma Abordagem Integrada à Análise Estruturada e Redes de Petri - KECHI HIRAMA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/9601 - MOOPP: Uma Metodologia Orientada a Objetos para Desenvolvimento de Software para Processamento Paralelo - ELISA HATSUE MORIYA HUZITA, LÍRIA MATSUMOTO SATO

BT/PCS/9602 - Estudo do Espalhamento Brillouin Estimulado em Fibras Ópticas Monomodo - LUIS MEREGE SANCHES, CHARLES ARTUR SANTOS DE OLIVEIRA

BT/PCS/9603 - Programação Paralela com Variáveis Compartilhadas para Sistemas Distribuídos - LUCIANA BEZERRA ARANTES, LIRIA MATSUMOTO SATO

BT/PCS/9604 - Uma Metodologia de Projeto de Redes Locais - TEREZA CRISTINA MELO DE BRITO CARVALHO, WILSON VICENTE RUGGIERO

BT/PCS/9605 - Desenvolvimento de Sistema para Conversão de Textos em Fonemas no Idioma Português - DIMAS TREVIZAN CHBANE, GERALDO LINO DE CAMPOS

BT/PCS/9606 - Sincronização de Fluxos Multimídia em um Sistema de Videoconferência - EDUARDO S. C. TAKAHASHI, STEFANIA STIUBIENER

BT/PCS/9607 - A importância da Completeza na Especificação de Sistemas de Segurança - JOÃO BATISTA CAMARGO JÚNIOR, BENÍCIO JOSÉ DE SOUZA

BT/PCS/9608 - Uma Abordagem Paraconsistente Baseada em Lógica Evidencial para Tratar Exceções em Sistemas de Frames com Múltipla Herança - BRÁULIO COELHO ÁVILA, MÁRCIO RILLO

BT/PCS/9609 - Implementação de Engenharia Simultânea - MARCIO MOREIRA DA SILVA, MOACYR MARTUCCI JÚNIOR

BT/PCS/9610 - Statecharts Adaptativos - Um Exemplo de Aplicação do STAD - JORGE RADY DE ALMEIDA JUNIOR, JOÃO JOSÉ NETO

BT/PCS/9611 - Um Meta-Editor Dirigido por Sintaxe - MARGARETE KEIKO IWAI, JOÃO JOSE NETO

BT/PCS/9612 - Reutilização em Software Orientado a Objetos: Um Estudo Empirico para Analisar a Dificuldade de Localização e Entendimento de Classes - SELMA SHIN SHIMIZU MELNIKOFF, PEDRO ALEXANDRE DE OLIVEIRA GIOVANI

BT/PCS/9613 - Representação de Estruturas de Conhecimento em Sistemas de Banco de Dados - JUDITH PAVÓN MENDONZA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9701 - Uma Experiência na Construção de um Tradutor Inglês - Português - JORGE KINOSHITA, JOÃO JOSÉ NETO

BT/PCS/9702 - Combinando Análise de "Wavelet" e Análise Entrópica para Avaliar os Fenômenos de Difusão e Correlação - RUI CHUO HUEI CHIOU, MARIA ALICE G. V. FERREIRA

BT/PCS/9703 - Um Método para Desenvolvimento de Sistemas de Computacionais de Apoio a Projetos de Engenharia - JOSÉ EDUARDO ZINDEL DEBONI, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9704 - O Sistema de Posicionamento Global (GPS) e suas Aplicações - SÉRGIO MIRANDA PAZ, CARLOS EDUARDO CUGNASCA

BT/PCS/9705 - METAMBI-OO - Um Ambiente de Apoio ao Aprendizado da Técnica Orientada a Objetos - JOÃO UMBERTO FURQUIM DE SOUZA, SELMA S. S. MELNIKOFF

BT/PCS/9706 - Um Ambiente Interativo para Visualização do Comportamento Dinâmico de Algoritmos - IZAURA CRISTINA ARAÚJO, JOÃO JOSÉ NETO

BT/PCS/9707 - Metodologia Orientada a Objetos e sua Aplicação em Sistemas de CAD Baseado em "Features" - CARLOS CÉSAR TANAKA, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9708 - Um Tutor Inteligente para Análise Orientada a Objetos - MARIA EMÍLIA GOMES SOBRAL, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9709 - Metodologia para Seleção de Solução de Sistema de Aquisição de Dados para Aplicações de Pequeno Porte - MARCELO FINGUERMAN, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9801 - Conexões Virtuais em Redes ATM e Escalabilidade de Sistemas de Transmissão de Dados sem Conexão - WAGNER LUIZ ZUCCHI, WILSON VICENTE RUGGIERO

BT/PCS/9802 - Estudo Comparativo dos Sistemas da Qualidade - EDISON SPINA, MOACYR MARTUCCI JR.

BT/PCS/9803 – The VIBRA Multi-Agent Architecture: Integrating Purposive Vision With Deliberative and Reactive Planning – ANNA H. REALI C. RILLO, REINALDO A. C. BIANCHI, LELIANE N. BARROS

BT/PCS/9901 – Metodologia ODP para o Desenvolvimento de Sistemas Abertos de Automação – JORGE LUIS RISCO BECCERRA, MOACYR MARTUCCI JUNIOR

BT/PCS/9902 – Especificação de Um Modelo de Dados Bitemporal Orientado a Objetos – SOLANGE NICE ALVES DE SOUZA, EDIT GRASSIANI LINO DE CAMPOS