Uma Avaliação Experimental de Algoritmos Distribuídos para Determinação do Fluxo Máximo em Redes

Cláudia Ferreira Portella

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Valmir C. Barbosa, Ph. D.
(presidente)

Prof. Claudio Luís Amorim, Ph. D.

Prof. Claudio Thomas Bornstein, Ph.D.

Prof. Nair Maria Maia de Abreu, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 1992

PORTELLA, CLÁUDIA FERREIRA

Uma Avaliação Experimental de Algoritmos Distribuídos para Determinação do Fluxo Máximo em Redes [Rio de Janeiro] 1992

V, 83 p., 29.7 cm, (COPPE/UFRJ, M. Sc. ENGENHARIA DE SISTEMAS E COMPUTAÇÃO, 1990)

TESE - Universidade Federal do Rio de Janeiro, COPPE

1 - Algoritmos Distribuídos 2 - Fluxo Máximo

I. COPPE/UFRJ II. Título(Série).

A meus pais

Resumo da Tese apresentada à COPPE como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

Uma Avaliação Experimental de Algoritmos Distribuídos para Determinação do Fluxo Máximo em Redes

Cláudia Ferreira Portella

Maio de 1992

Orientador: Valmir Carneiro Barbosa

Programa: Engenharia de Sistemas e Computação

A determinação do fluxo máximo em redes é fundamental para a solução de problemas em diversas áreas, especialmente em otimização combinatória. A crescente demanda por capacidade computacional e velocidade de processamento tem levado a grandes progressos na área de computação paralela e distribuída, o que motivou o estudo de algoritmos de determinação do fluxo máximo visando este tipo de processamento.

Este trabalho trata da implementação e avaliação de três algoritmos distribuídos para o cálculo do fluxo máximo em um computador paralelo constituído de oito processadores do tipo Transputer, interligados através de uma rede de comunicação assíncrona, em uma topologia hipercúbica. Os processadores não compartilham memória, comunicando-se através da troca de mensagens. Foram realizadas experiências com várias classes de redes, cada uma carcterizada por um parâmetro específico, cuja variação pode influenciar significativamente o desempenho dos algoritmos.

Abstract of Thesis presented to COPPE as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.)

An Experimental Evaluation of Distributed Parallel Max-Flow Algorithms

Cláudia Ferreira Portella

May 1992

Thesis Supervisor: Valmir C. Barbosa

Department: Programa de Engenharia de Sistemas e Computação

The determination of the maximum flow in a flow network is fundamental for the solution of problems in many fields, especially in combinatorial optimization. The increasing demand for computing power has spurred the development of parallel architedtures, what motivated the study of parallel maximum-flow algorithms.

This work presents the implementation and experimental evaluation of three distributed parallel maximum-flow algorithms in a parallel computer composed by an eight-node Transputer hypercube, conected by an asynchronous network, where the processors communicate with each other by exchanging messages. Experiments were performed on various classes of flow networks, each emphasizing differents aspects of relevance.

Índice

Capítulo I

Introdução

A determinação do fluxo máximo em redes é fundamental para a solução de problemas em diversas áreas, especialmente em otimização combinatória [22] [26], que trata de casos em que se deseja encontrar uma associação de valores discretos às variáveis de um problema, de forma a otimizar uma certa função objetivo.

O cálculo do fluxo máximo consiste em estabelecer como transportar fluxo entre dois pontos distintos de uma rede genérica, formada por arcos com capacidades limitadas, de modo que a quantidade de fluxo transferida seja máxima. Um grande número de problemas pode ser resolvido através do seu mapeamento em um caso de fluxo máximo em redes. O problema de cálculo do fluxo máximo foi formulado na década de 50 por FULKERSON & DANTZIG [16] [8] e desde então, algoritmos vêm sendo desenvolvidos para resolvê-lo.

A crescente demanda por capacidade computacional e velocidade de processamento tem levado a grandes progressos na área de computação paralela e distribuída, o que motivou o estudo de algoritmos de determinação do fluxo máximo visando este tipo de processamento.

Este trabalho trata da implementação e avaliação de algoritmos distribuídos para o cálculo do fluxo máximo em um computador paralelo constituído de oito processadores do tipo Transputer [20], interligados através de uma rede de comunicação assíncrona, em uma topologia hipercúbica. Os processadores não compartilham memória, comunicando-se através da troca de mensagens.

Foram implementados os algoritmos de GOLDBERG & TARJAN [17], versões síncrona e assíncrona, e de AWERBUCH [4], versão síncrona. Estes algoritmos se baseiam em duas técnicas de cálculo do fluxo máximo fundamentalmente distintas. O algoritmo de AWERBUCH aplica o conceito de caminhos aumentantes, introduzido por FORD & FULKERSON [15], e o princípio de construção de redes em camadas, proposto por DINIC [12]. Já o algoritmo de GOLDBERG & TARJAN emprega o conceito de pré-fluxo definido por KARZANOV [21].

Outro aspecto que distingue os algoritmos implementados é modelo de comunicação para os qual estes foram projetados. Enquanto duas das versões desenvolvidas visam uma rede de comunicação síncrona, a terceira foi projetada para operar em uma rede assíncrona, realizando a sua sincronização localmente, através de mensagens de aceitação e rejeição de fluxo. Como os algoritmos foram implementados em um ambiente assíncrono, foi necessária a introdução de um sincronizador nas versões originariamente síncronas. Um sincronizador e' um protocolo distribuído que permite executar algoritmos síncronos em redes assíncronas.

O sincronizador adotado neste trabalho é uma versão simplificada do sincronizador α proposto por AWERBUCH [5]. A utilização deste sincronizador alterou as complexidades computacionais das versões síncronas implementadas, que passaram a possuir a mesma complexidade de tempo, $O(n^2)$, e de mensagens, $O(n^2m)$, que a versão assíncrona, o que torna ainda mais interessante a avaliação dos algoritmos implementados.

Esta avaliação compreende a análise do comportamento dos algoritmos quando submetidos a diferentes tipos de grafos de entrada, a comparação de seus desempenhos, o *speedup* obtido para configurações com 2, 4 e 8 processadores e a análise da utilização da memória por cada algoritmo.

O capítulo II apresenta a definição do problema de cálculo do fluxo máximo em redes e os principais algoritmos, seqüenciais e paralelos, para resolvê-lo. O Capítulo III contém a descrição detalhada dos algoritmos de GOLDBERG & TARJAN e de AWERBUCH, enquanto o Capítulo IV aborda todos os procedimentos e algoritmos que dão suporte à implementação distribuída destes algoritmos

de cálculo do fluxo máximo, o que abrange o balanceamento de carga, a difusão de dados e parâmetros de controle, a comunicação entre processadores e o modelo do sincronizador utilizado.

O Capítulo V apresenta os algoritmos implementados após sua adaptaçao para um ambiente distribuído assíncrono. O Capítulo VI trata da metodologia de teste adotada e da análise dos resultados obtidos. Finalmente, o Capítulo VII apresenta as conclusões e as considerações finais.

Capítulo II

Fluxo Máximo em Redes

Neste capítulo é definido o problema de determinação do fluxo máximo em redes. Inicialmente são apresentados alguns conceitos básicos para a compreensão do problema, em seguida, é feita a definição formal de fluxo máximo e, por fim, são relacionados os principais algoritmos, seqüenciais e paralelos, para resolvê-lo.

II.1 Conceitos Básicos

Antes de definir o problema de cálculo do fluxo máximo em redes, é conveniente apresentar alguns conceitos básicos, são eles:

grafo - um grafo $G = (V, E)$ é um conjunto finito não vazio V e um conjunto E de pares não ordenados de elementos distintos de V. Os elementos de V são os vértices do grafo e os elementos de E as arestas. Cada aresta $e \in E$ é denotada pelo par de vértices que a forma, $e = (v, w)$. Neste caso, os vértices v e w são os extremos da aresta e, sendo denominados adjacentes.

grafo orientado - é um grafo em que o conjunto de arestas E é constituído de pares ordenados de vértices. Deste modo, cada aresta (v, w) possui uma única direção, de v para w. Os grafos orintados também são chamados de digrafos.

grau de entrada de v - é o número de aresta convergentes ao vértice v, em um grafo orientado.

grau de saída de v - é o número de aresta divergentes do vértice v, em um grafo orientado.

fonte - é um vértice com grau de entrada nulo.

dreno - é um vértice com grau de saída nulo.

II.2 Definição do Problema

Uma rede é um grafo orientado $G = (V, E)$ em que a cada aresta $e \in E$ está associado um número real positivo $c(e)$, denominado capacidade da aresta. Suponha que G possua uma fonte s, que alcança todos os vértices e um dreno t, que é alcançado por todos os vértices. Um fluxo f de s a t em G é uma função que associa a cada aresta $e \in E$ um número real não negativo $f(e)$, satisfazendo as seguintes condições:

1. $0 \leq f(e) \leq c(e)$, para toda aresta $e \in E$
2. Sejam $\alpha(v)$ e $\beta(v)$ os conjuntos de arestas convergentes e divergentes de v, respectivamente. Para todo vértice $v \in V - \{s, t\}$:

$$\sum_{e \in \alpha(v)} f(e) - \sum_{e \in \beta(v)} f(e) = 0$$

A primeira condição mostra que o fluxo através de uma aresta não pode ser maior que a sua capacidade, enquanto a segunda condição determina que a quantidade total de fluxo que chega a um vértice $v \in V - \{s, t\}$ é igual ao total de fluxo que sai de v, o que indica a conservação de fluxo nos vértices.

O fluxo total na rede F é definido como o fluxo que chega ao dreno:

$$F = \sum_{e \in \alpha(t)} f(e)$$

O fluxo total em uma rede pode variar de zero a um valor máximo, que representa a quantidade máxima de fluxo que pode ser transmitida da fonte para o dreno através da rede. O problema de cálculo do fluxo máximo consiste na

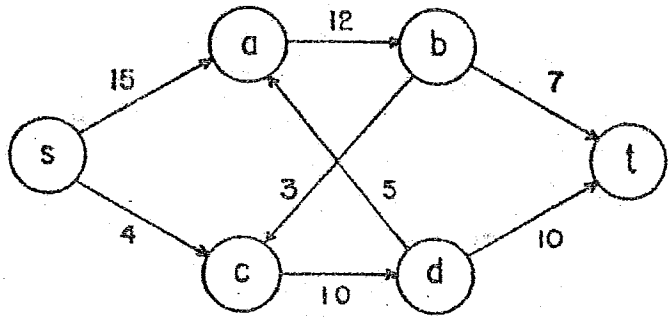

Figura II.1: Exemplo de rede com as respectivas capacidades de aresta

determinação desta quantidade. A figura II.1 apresenta um exemplo de rede com as respectivas capacidades das arestas, enquanto a figura II.2 mostra a distribuição de fluxo na rede, de modo que o fluxo total seja máximo. Maiores detalhes sobre a definição do problema de cálculo do fluxo máximo e suas aplicações podem ser encontrados em [14] [30].

II.3 Algoritmos de Fluxo Máximo

O problema de determinação do fluxo máximo em redes foi formulado na década de 50 por FULKERSON & DANTZIG [16] [8]. Desde então, algoritmos vêm sendo desenvolvidos para resolvê-lo. Nesta seção são apresentados, sucintamente, os principais algoritmos seqüenciais e paralelos para o cálculo do fluxo máximo. Antes, porém, são definidos mais alguns conceitos básicos e notações utilizadas ao longo do trabalho.

II.3.1 Definições

Seja $G = (V, E)$ um grafo orientado, com dois vértices especiais e distintos s e t, onde s é um a fonte e t um dreno. O número de vértices n corresponde ao número total de elementos do conjunto de vértices V. O número de arestas m corresponde ao número total de elementos do conjunto de arestas E.

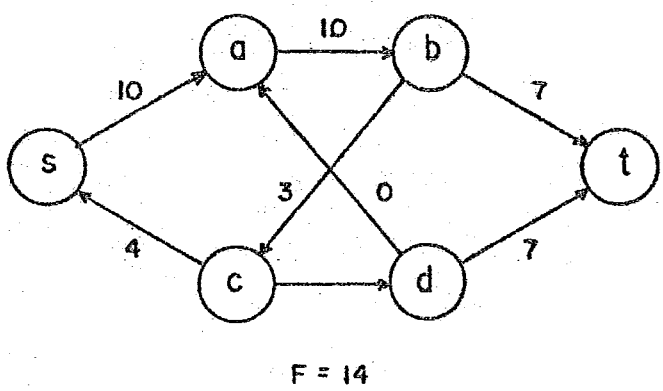

Figura II.2: Distribuição de Fluxo.

Os **vizinhos** de um vértice v são todos os vértices que estão ligados a v através de uma aresta, isto é, os vértices adjacentes a v.

Uma seqüencia de vértices $v_1, ..., v_k$, tal que $(v_j, v_{j+1}) \in E$, $1 \leq j < |k-1|$, é denominado **caminho** de v_1 a v_k.

A **distância** $d_G(v, w)$ entre os vértices v e w, corresponde ao menor número de arestas no caminho de v para w em G. Se este caminho não existe, então a distância $d_G(v, w)$ é infinita.

Uma aresta $(v, w) \in E$ é dita **saturada** quando o fluxo através dela é igual à sua capacidade, isto é, $f(v, w) = c(v, w)$. Um vértice $v \in V$ é dito saturado quando todas as arestas convergentes a v, ou todas as arestas divergentes de v, estão saturadas.

Um fluxo f é **maximal** se todo caminho de s a t em G contém alguma aresta saturada, ou seja, o valor do fluxo maximal não pode ser aumentado simplesmente por acréscimos de fluxos em algumas arestas. Todo fluxo máximo é maximal, porém nem todo fluxo maximal é máximo.

A **capacidade residual** de uma aresta representa a quantidade de fluxo que pode ser adicionada ao fluxo atual da aresta, sendo definida pela expressão:

$$cr(v, w) = c(v, w) - f(v, w)$$

Uma aresta e é chamada de **aresta residual** quando $cr(e) > 0$. Uma **rede residual** é uma rede formada pelos conjunto de vértices V da rede original e pelo conjunto de arestas residuais correspondentes a um dado fluxo.

Um caminho de s a t na rede residual é dito **caminho aumentante** para f. Se $s = v_1$ e $t = v_k$, no caminho aumentante $v_1...v_k$, então o fluxo na rede G, pode ser aumentado de um valor F', tal que:

$$F' = min\{cr(v_j, v_{j+1}) | 1 \leq j < k\}$$

Um **pré-fluxo** [21] é uma função sobre as arestas da rede, similar ao fluxo, exceto que a quantidade total de fluxo que chega a um vértice $v \in V - \{s, t\}$ pode ser maior ou igual ao fluxo total que sai de v. Para toda aresta $e \in E$:

$$\sum_{e \in \alpha(v)} f(e) - \sum_{e \in \beta(v)} f(e) \geq 0$$

O **excesso de fluxo** em um vértice v é justamente a diferença entre o fluxo que entra em v e o fluxo que sai de v.

$$E(v) = \sum_{e \in \alpha(v)} f(e) - \sum_{e \in \beta(v)} f(e)$$

II.3.2 Algoritmos Seqüenciais

O primeiro algoritmo de fluxo máximo conhecido foi formulado por FORD & FULKERSON [15]. Este algoritmo se baseia no estabelecimento de caminhos aumentantes, entre a fonte e o dreno, na rede residual. Quando não houver mais caminhos que possam ser utilizados para incrementar o fluxo total na rede, significa que o fluxo atual é máximo e o algoritmo termina. A complexidade deste algoritmo é $O(mF)$, onde F é o valor do fluxo máximo.

EDMONDS & KARP [13] alteraram o algoritmo de FORD & FULKENSON, introduzindo um critério para a seleção dos caminhos aumentantes, segundo o qual escolhe-se primeiro o caminho aumentante de menor comprimento, isto

é, com o menor número de arestas. Com esta modificação, o algoritmo passa a ter complexidade igual a $O(nm^2)$, que é garantidamente polinomial, o que não ocorre com o algoritmo original.

DINIC [12] propôs que todos os caminhos aumentantes de menor comprimento fossem determinados em uma única fase, através da construção de uma rede auxiliar, denominada rede em camadas. Deste modo, o problema de determinação do fluxo máximo em uma rede arbitrária G se transforma em $O(n)$ problemas de determinação de um fluxo maximal em uma rede em camadas G^m. O algoritmo de DINIC possui complexidade igual a $O(n^2m)$ e sua filosofia foi utilizada em grande parte dos algoritmos posteriores.

O algoritmo de MALHOTRA, KUMAR & MAHESHWARI [24] se baseia no algoritmo de DINIC, e associa um fluxo potencial a cada vértice da rede em camadas, que representa o fluxo adicional máximo que pode ser enviado através do vértice. No algoritmo de DINIC, obtinha-se o fluxo maximal eliminando-se uma aresta da rede a cada iteração, neste algoritmo elimina-se um vértice a cada iteração. Este algoritmo possui complexidade igual a $O(n^3)$.

II.3.3 Algoritmos Paralelos

Os avanços alcançados na área de processamento paralelo motivaram o desenvolvimento de algoritmos de fluxo máximo visando este tipo de computação.

SHILOACH & VISHKIN [29] propuseram um algoritmo paralelo síncrono baseado no conceito de pré-fluxo de KARZANOV [21] e na estratégia de DINIC. Este algoritmo foi projetado visando um modelo de memória compartilhada e utiliza uma estrutura de dados denominada árvore de somas parciais *(PS-trees)*. A complexidade deste algoritmo é $O(n^2 log n)$. Além do algoritmo paralelo, também foi proposta uma versão seqüencial.

SEGALL [28] apresentou versões descentralizadas dos algoritmos de FORD & FULKERSON, EDMONDS & KARP e DINIC para um modelo de memória distribuída onde a comunicação entre processadores é feita através de troca de men-

sagens. A versão referente ao algoritmo de DINIC, que obteve os melhores resultados em termos de complexidade, possui complexidade de tempo igual a $O(n^2m)$ e complexidade de mensagens $O(nm^2)$.

AWERBUCH [4] propôs uma versão distribuída do algoritmo de SHILOACH & VISHKIN, adaptada para operar em uma rede assíncrona através da utilização de protocolos de sincronização [5] [2]. Este algoritmo possui complexidade de tempo síncrono igual a $O(n^2)$ e de mensagens igual a $O(n^3)$.

GOLDBERG & TARJAN [17] introduziram um método de cálculo do fluxo máximo onde um pré-fluxo é mantido na rede original e os excessos de fluxo locais são enviados em direção ao dreno através dos menores caminhos estimados. GOLDBERG & TARJAN propuseram duas versões de seu algoritmo: uma visando um modelo síncrono de computação e outra projetada para um modelo assíncrono. A versão síncrona possui complexidade de tempo igual a $O(n^2)$ e complexidade de mensagens igual a $O(n^3)$, enquanto a versão assíncrona possui complexidade de tempo igual a $O(n^2)$ e complexidade de mensagens igual a $O(n^2m)$. Existe ainda uma versão seqüencial deste algoritmo que utiliza uma estrutura de dados denominada *Dynamic Tree*.

AHUJA & ORLIN [1] alteraram o algoritmo de GOLDBERG & TARJAN, utilizando os conceitos de escalonamento introduzidos por EDMONDS & KARP e posteriormente desenvolvidos por GABOW [18]. Este algoritmo possui complexidade de tempo igual a $O(n^2 log U log P)$, onde U representa a capacidade máxima das arestas e P o número de processadores da rede.

MARBERG & GAFNI [25] formularam um algoritmo que utiliza o conceito de pré-fluxo e a construção de redes em camadas. Este algoritmo foi projetado para um ambiente assíncrono, onde a comunicação é feita através da troca de mensagens, e se baseia na definição de super-camadas e camadas especiais. As suas complexidades de tempo e de mensagens são iguais a $O(n^3m^{1/2})$.

Capítulo III

Descrição dos Algoritmos

Este trabalho trata da implentação de três sistemas de cálculo do fluxo máximo. O primeiro baseado na versão síncrona do algoritmo de GOLDBERG & TARJAN, o segundo na versão assíncrona do mesmo algoritmo, e o último no algoritmo de AWERBUCH. A motivação para implementação destes algoritmos reside no fato deles adotarem duas técnicas essencialmente distintas para calcular o fluxo máximo. Enquanto o algoritmo de AWERBUCH emprega os conceitos de caminhos aumentantes e redes em camadas, o algoritmo GOLDBERG & TARJAN se baseia no conceito de pré-fluxo.

Outro aspecto que torna interessante o estudo do comportamento destes algoritmos é a diferença entre os modelos de comunicação que estes visam. Enquanto dois algoritmos foram projetados para operar em uma rede de comunicação síncrona o terceiro visa uma rede assíncrona, realizando a sua sincronização através de mensagens de aceitação e rejeição de fluxo. Este capítulo descreve a filosofia dos algoritmos implementados.

III.1 Transferência de Fluxo

Antes de iniciar a descrição dos algoritmos é conviniente definir como se processa a transferência de fluxo entre dois vértices adjacentes. Este procedimento é comum aos algoritmos de GOLDBERG & TARJAN e de AWERBUCH.

A quantidade de fluxo a ser transferida de um vértice v para um

vértice w é definida pelo valor do excesso de fluxo contido em v e a capacidade residual da aresta (v,w). Se o excesso em v e capacidade residual de (v,w) forem positivos, a quantidade de fluxo a ser transferida é igual ao menor dentre estes dois valores.

$$qf = min\{E(v), cr(v,w)\}$$

Uma aresta (v,w) pode ter capacidade residual positiva de duas maneiras: se (v,w) é uma aresta com fluxo menor que a sua capacidade, $f(v,w) < c(v,w)$, ou se (w,v) é uma aresta com fluxo positivo. No primeiro caso, ao transferir excesso de v para w está-se aumentando o fluxo de v para w, enquanto no segundo caso, está-se diminuindo o fluxo de w para v.

A transferência de fluxo de v para w, reduz o excesso em v, aumenta o fluxo de v para w e, conseqüentemente, reduz a capacidade residual da aresta (v,w).

$$E(v) \leftarrow E(v) - qf$$

$$f(v,w) \leftarrow f(v,w) + qf$$

$$cr(v,w) \leftarrow cr(v,w) - qf$$

Por outro lado a transferência de fluxo de v para w, aumenta o excesso em w, reduz o fluxo de w para v e aumenta a capacidade residual da aresta (w,v).

$$E(w) \leftarrow E(w) + qf$$

$$f(w,v) \leftarrow f(w,v) - qf$$

$$cr(w,v) \leftarrow cr(w,v) + qf$$

III.2 Algoritmo de GOLDBERG & TARJAN

Este algoritmo inicia com um pré-fluxo em todas as arestas divergentes da fonte igual às respectivas capacidades e zero nas demais arestas. Desta maneira, as arestas

divergentes da fonte se tornam saturadas e todos os seus vizinhos passam a conter um excesso de fluxo. Os vértices da rede, com exceção da fonte e do dreno, são examinados de modo que o excesso de fluxo que eles contêm seja enviado em direção ao dreno, através dos menores caminhos estimados. Esta estimativa e' feita através de um procedimento de rotulação dos vértices. As transferências de fluxo alteram a rede residual e, conseqüentemente, os caminhos que levam ao dreno podem se tornar saturados. O excesso de fluxo que não puder ser movidos em direção ao dreno, e' devolvido para a fonte, também através dos menores caminhos estimados. O algoritmo termina quando não houver mais excesso nos vértices, com exceção do dreno e da fonte. O pré-fluxo terá se tornado então um fluxo, que é o fluxo máximo.

III.2.1 Procedimento de Rotulação

Para estimar a distância de um vértice qualquer até a fonte ou ao dreno, foi definido um procedimento de rotulação. A rótulo d é uma função dos vértices, que pode assumir valores inteiros não negativos ou infinito, sendo que: $d(s) = n$, $d(t) = 0$ e $d(v) \leq d(w) + 1$, para toda aresta residual (v, w). O propósito é que, se $d(v) < n$, então $d(v)$ é um limite inferior para a distância entre v e t na rede residual. Se $d(v) \geq n$, então $d(v) - n$ é um limite inferior para a distância de v à s na rede residual.

Segundo o algoritmo, um vértice v só poderá enviar fluxo para um vértice w se $d(v) = d(w) + 1$ e $cr(v, w) > 0$. Caso v não possua nenhum vizinho que satisfaça estas duas condições, calcula-se um novo rótulo para v. Este novo rótulo é igual ao menor dentre os rótulos dos vizinhos de v, tais que $cr(v, w) > 0$, acrescido de uma unidade.

Inicialmente, o rótulo de todos os vértices, com exeção da fonte, e' igual a zero. O rótulo da fonte (igual a n) e o rótulo do dreno (igual a zero) permanecem constantes ao longo de todo algoritmo.

III.2.2 Implementação Distribuída

O modelo computacional descrito por GOLDBERG & TARJAN, para implementação paralela, é uma maquina PRAM *(parallel random-access machine)*, que permita leitura exclusiva e escrita exclusiva ou uma máquina DRAM *(distributed random-access machine)*. A cada vértice do grafo de entrada corresponde um processador e uma quantidade de memória proporcional ao número de vizinhos do vértice. Este processador pode comunicar-se diretamente com os processadores relativos a todos os seus vizinhos.

A versão síncrona do algoritmo é executada em pulsos, que são contituídos de operações aplicadas em paralelo aos vértices ativos da rede. Um vértice v é considerado ativo se $v \in V - \{s,t\}$, $d(v) < \infty$ e $E(v) > 0$. Cada pulso pode ser dividido em quatro estágios. Durante o primeiro estágio é realizado o envio de fluxo, no segundo é feita a rotulação dos vértices, a difusão dos novos rótulo ocorre no terceiro estágio, e o fluxo recebido por um vértice no primeiro estágio é somado ao seu excesso no quarto estágio. O algoritmo paralelo consiste em repetir o procedimento pulso, descrito na figura III.1, até que não haja nenhum vértice ativo na rede. A complexidade de tempo síncrono deste algoritmo é igual a $O(n^2)$ e complexidade de mensagens igual a $O(n^3)$.

III.2.3 Versão Assíncrona

O algoritmo de GOLDBERG & TARJAN pode ser modificado para funcionar em um ambiente assíncrono, dispensando o uso de um protocolo de sincronização. Nesta versão, o sincronismo é feito localmente, através de mensagens de aceitação e rejeição de fluxo.

Quando um vértice v decide enviar uma quantidade de fluxo qf para um vértice w, a informação local em v é que $d(v) = d(w) + 1$. O vertice v transmite, então, uma mensagem para w contendo a sua identidade, a quantidade de fluxo que está sendo enviada e o rótulo de v: $(v, qf, d(v))$ e atualiza o valor do excesso em v e da capacidade residual da aresta (v, w). O vértice v não enviará fluxo para w

---

Faça em paralelo para todos os vértice ativos

< Estágio 1 >

Envie fluxo de v até que E(v) = 0 ou
∀ w tal que d(v) = d(w)+1, cr(v,w) = 0

< Estágio 2 >

IF $E(v) > 0$ THEN
$d'(v) \leftarrow min\ \{d(w)+1 \mid cr(v,w) > 0\}$

< Estágio 3 >

IF $d(v) \neq d'(v)$ THEN
$d(v) \leftarrow d'(v)$
Envie o valor de d(v) para todos os vizinhos de v

< Estágio 4 >

Adicione o fluxo recebido por v durante o Estágio 1 a E(v)

Figura III.1: Procedimento Pulso

novamente até que receba de w uma mensagem de aceitação ou rejeição.

Quando o vértice w recebe a mensagem de v, verifica se realmente $d(v) = d(w) + 1$, pois o valor de $d(w)$ local em v pode estar incorreto. Se $d(v) = d(w)+1$, então w uma envia mensagem de aceitação para v: $(ACEITA, w, qf, d(w))$ e atualiza o excesso de w e a capacidade residual da aresta (w, v). Senão w envia uma mensagem de rejeição: $(REJEITA, w, qf, d(w))$, onde $d(w)$ é o valor corrigido do rótulo de w.

Ao receber uma mensagem de rejeição o vértice v, além de atualizar o valor de $E(v)$ e $cr(v, w)$, corrige o valor de $d(w)$ e se necessário o de $d(v)$ também. Quando um vértice muda de rótulo, este informa o novo valor a todos os seus vizinhos.

A complexidade de tempo assíncrono desta versão é igual a $O(n^3)$ e complexidade de mensagens igual a $O(n^2m)$.

III.3 Algoritmo de AWERBUCH

Este algoritmo é uma versão do algoritmo proposto por SHILOACH & VISHIKIN [29], que foi inicialmente projetado para um modelo computacional paralelo síncrono e modificado por AWERBUCH [4] para operar em uma rede distribuída assíncrona a partir da uso de protocolos de sincronização [5]. O algoritmo pode ser dividido em duas fases. Na primeira fase uma rede auxilar, denominada rede em camadas, é construída com base na rede residual original; na segunda fase, determina-se um fluxo maximal na rede em camadas obtida.

A rede em camadas é formada pelos menores caminhos aumentantes entre a fonte e o dreno, e pode ser construída realizando uma busca em largura ligeiramente modificada, a partir do vértice fonte. Se o dreno não estiver contido na rede em camadas obtida, significa que não há mais nenhum caminho através do qual seja possível enviar fluxo para o dreno. Neste ponto, o fluxo corrente é máximo e o algoritmo termina. Caso contrário, a segunda fase do algoritmo é executada e é calculado um fluxo maximal na rede em camadas obtida. O cálculo do fluxo maximal consiste em enviar a quantidade máxima de fluxo possível para o dreno através da rede em camadas. Quando não existirem mais caminhos através dos quais se possa enviar fluxo para o dreno, significa que o fluxo corrente é maximal e uma nova rede em camadas é construída. A figura III.2 apresenta a estrutura do algoritmo de AWERBUCH. A complexidade de tempo síncrono deste algoritmo é igual a $O(n^2)$ e complexidade de mensagens igual a $O(n^3)$.

III.3.1 Determinação do Fluxo Maximal

Inicialmente a fonte possui um excesso de fluxo igual ao somatório das capacidades residuais das suas arestas divergentes. No primeiro passo do algoritmo de determinação do fluxo maximal, a fonte envia fluxo para todos os seus vizinhos, saturando as suas arestas divergentes. Deste modo alguns vértices se tornam desbalanceados, isto é, o fluxo que chega a estes vétices é maior do que o fluxo que sai. Os vértices desbalanceados tentam enviar o máximo de fluxo possível em direção ao dreno.

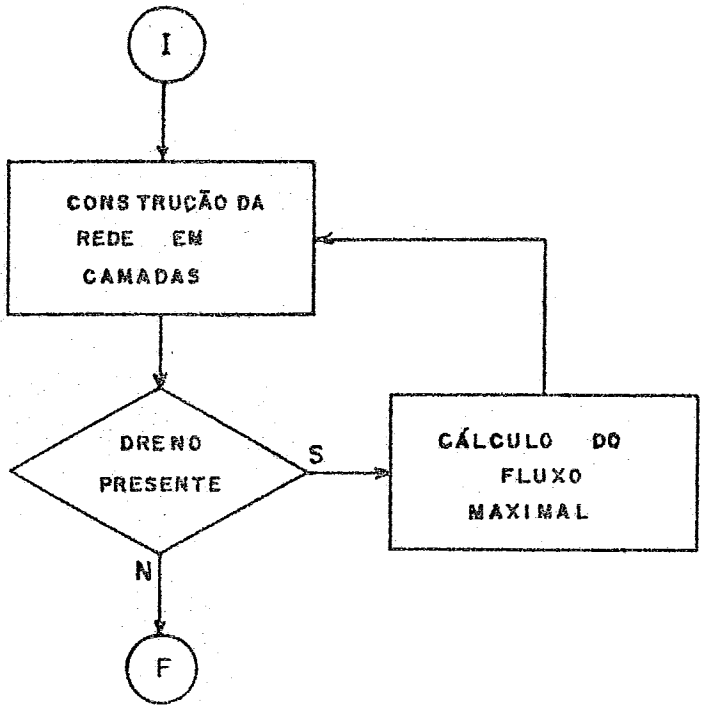

Figura III.2: Estrutura do Algoritmo de AWERBUCH.

Cada vértice possui uma lista de arestas divergentes, armazenadas em uma ordem arbitrária. Esta lista é formada durante a construção da rede em camadas. Quando um vértice tenta enviar o seu excesso em direção ao dreno, ele examina a primeira aresta da lista e envia através dela a maior quantidade de fluxo possível. Caso esta aresta se torne saturada sem que todo o excesso do vértice seja eliminado, a segunda aresta da lista passa a ser examinada e assim sucessivamente, até que todo o excesso de fluxo contido no vértice tenha sido seja eliminado ou todas as suas arestas divergentes tenham se tornado saturadas. No segundo caso, o excesso restante é devolvido, através das arestas convergentes, aos vértices de origem na ordem LIFO (*last-in-first-out*), isto é, o último vizinho do qual o vértice recebeu fluxo será o primeiro para o qual ele tentará devolver fluxo e assim sucessivamente.

Quando um vértice recebe uma quantidade de fluxo, ele armazena em uma pilha local a quantidade de fluxo recebida e a identidade do vértice origem. No caso de recepção de fluxo devolvido este procedimento não e' realizado. Na devolução de fluxo, a última informação a ser colocada na pilha é retirada. Caso a quantidade de fluxo a ser devolvida seja menor que a quantidade armazenada

pilha, devolve-se para o vértice origem a diferença entre estas quantidades e coloca-se novamente na pilha o vértice origem e a quantidade de fluxo atualizada, isto é, descontando-se o fluxo devolvido. Caso contrário, devolve-se para o vértice origem todo o fluxo recebido dele e continua-se retirando informações da pilha até que todo o excesso remanescente seja devolvido.

Um vértice que tenha devolvido fluxo é chamado de vértice bloqueado, pois este já saturou todas as suas arestas divergentes e desta forma é inútil enviar fluxo para ele, pois todo fluxo enviado será devolvido. Para evitar esforço em vão, quando um vértice se torna bloqueado, ele envia para todos os seus vizinhos convergentes uma mensagem informando que está bloqueado. Quando um vértice recebe este tipo de mensagem, retira o vértice que a enviou da sua lista de vizinhos divergentes e assim não tenta mais enviar fluxo para este vértice. O algoritmo de cálculo do fluxo maximal termina quando não houver mais excesso de fluxo em nenhum vértice com exceção do dreno e da fonte. A figura III.3 apresenta o algoritmo de determinação do fluxo maximal.

Sempre que um novo pulso começar, faça para cada vértice v

< Envio de Fluxo em Direção ao Dreno >

ENQUANTO $Excesso(v) > 0$ AND $Disponiveis(v) \neq \{\}$

ESCOLHA $w \in Disponiveis(v)$
$qf \leftarrow min\{cr(v,w), Excesso(v)\}$
$Excesso(v) \leftarrow Excesso(v) - qf$
$cr(v,w) \leftarrow cr(v,w) - qf$
$fluxo(v,w) \leftarrow fluxo(v,w) + qf$
ENVIE MENSAGEM $Env.Fluxo\{v,w,qf\}$ PARA w
SE $fluxo(v,w) = capacidade(v,w)$ ENTÃO
RETIRE w DE $Disponiveis(v)$
SE $Dispnoveis(v) = \{\}$ ENTÃO
ENVIE MENSAGEM $Bloqueado\{v\}$ PARA OS VIZINHOS CONVERGENTES A v

< Devolução de Fluxo em Direção a Fonte >

ENQUANTO $Excesso(v) > 0$

RETIRE DA PILHA: (k, qf)
$qf^* \leftarrow min\{qf, Excesso(v)\}$
$Excesso(v) \leftarrow Excesso(v) - qf^*$
$cr(v,k) \leftarrow cr(v,k) - qf^*$
$fluxo(v,k) \leftarrow fluxo(v,k) + qf^*$
ENVIE MENSAGEM $Dev.Fluxo\{v,k,qf\}$ PARA k
SE $qf - qf^* > 0$ ENTÃO
COLOQUE $(k, qf - qf^*)$ NA PILHA

< Tratamento de Mensagens >

SEMPRE QUE RECEBER $Bloqueado\{j\}$ DE j, FAÇA

SE $j \in Disponoveis(v)$ ENTÃO
RETIRE j DE $Disponiveis(v)$

SEMPRE QUE RECEBER $Env.Fluxo\{i,v,qf\}$ DE i, FAÇA

$Excesso(v) \leftarrow Excesso(v) + qf$
$cr(v,i) \leftarrow cr(v,i) + qf$
$fluxo(v,i) \leftarrow fluxo(v,i) - qf$
COLOQUE (i, qf) NA PILHA

SEMPRE QUE RECEBER $Dev.Fluxo\{l,v,qf\}$ DE l, FAÇA

$Excesso(v) \leftarrow Excesso(v) + qf$
$cr(v,l) \leftarrow cr(v,l) + qf$
$fluxo(v,l) \leftarrow fluxo(v,l) - qf$

Figura III.3: Algoritmo de Determinação do Fluxo Maximal.

Capítulo IV

Implementação Distribuída

Um sistema distribuído é composto por um conjunto de processadores independentes, conectados através de uma rede de comunicação, e que não compartilham memória. Um programa seqüencial especifica uma lista de instruções a serem executadas seqüencialmente, a execução desta lista é chamada de processo. Uma computação distribuída é um conjunto de processos alocados em dois ou mais processadores, que precisam se comunicar para atingir um objetivo comum. Esta comunicação permite que a execução de um processo interfira na execução de outro. A comunicação entre processos em um sistema distribuído é feita através da troca de mensagens.

Neste capitulo são apresentados os algoritmos e procedimentos que dão suporte à implementação distribuída dos sistemas de determinação do fluxo máximo. Primeiramente é apresentado o ambiente no qual o trabalho foi desenvolvida, em seguida são abordadas as tarefas executadas as preparação dos sistemas! o que abrange a entrada de dados, o balanceamento de carga, a difusão dos parâmetros de controle, etc. É definido também o conceito de processador vizinho. Por último, são descritos a, forma de comunicação entre processadores e o modelo de sincronização utilizado. Maiores detalhes sobre sistemas distribuídos podem ser encontrados em [23] [2].

IV.1 Ambiente de Desenvolvimento

Os algoritmos de cálculo do fluxo máximo foram implementados no computador paralelo NCP-1 [3], desenvolvido pela COPPE/UFRJ. Este computador é um sistema distribuído composto por 8 processadores do tipo Transputer T800 [20], conectados através de uma rede de comunicação assíncrona.

O contato com a rede de processadores é feito através de um hospedeiro, que oferece acesso a disco e a terminal de vídeo, funcionando como dispositivo de entrada e saída da rede. No momento, o hospedeiro é um microcomputador compatível com o IBM-PC. Somente um dos processadores da rede está ligado ao hospedeiro, sendo denominado de processador raiz.

Os processadores da rede, também chamados de nós, estão ligados através de canais bidirecionais de comunicação, segundo uma topologia hipercúbica. Cada nó possui 2 Mbytes de memória RAM disponível. A figura IV.1 mostra as configurações com 2, 4 e 8 processadores.

Embora os sistemas de determinação do fluxo máximo tenham sido desenvolvidos para operar em um hipercubo, eles podem ser adaptados para atuar em qualquer outra configuração, bastando para isto mudar a interface de comunicação que realiza o roteamento de mensagens.

O desenvolvimento de *software* foi realizado no TDS *(Transputer Development System)* [19] e a linguagem de programação utilizada foi o OCCAM2 [7].

IV.2 Iniciação do Sistema

A iniciação do sistema consiste no conjunto de tarefas realizadas em cada nó da rede, antes da execução do algoritmo de determinação do fluxo máximo propriamente dito. Esta iniciação visa fornecer a todos os nós da rede as informações necessárias para o começo do processamento. Como o nó raiz é o único que mantém contato com o hospedeiro, a entrada de dados é realizada através dele e, conseqüentemente, a iniciação deste nó é diferente da dos demais.

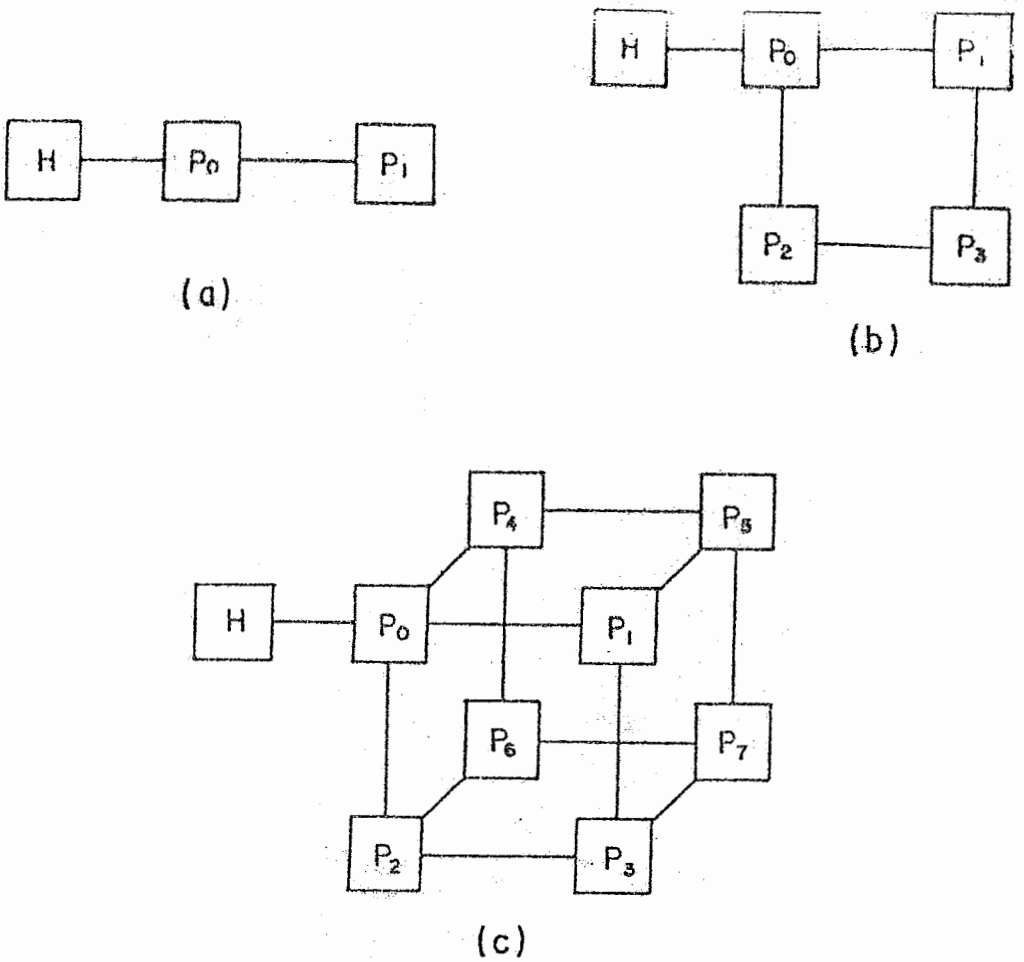

Figura IV.1: Configurações com 2, 4 e 8 Processadores.

Faça Seqüencialmente:

Leitura do número de vértices do grafo de entrada

Balanceamento de carga

Difusão dos parâmetros de controle

Leitura das aresta e difusão das listas de vizinhos e capacidades

Determinação dos processadores vizinhos

Iniciação das variáveis locais

Figura IV.2: Procedimento de Iniciação do Nó Raiz

Dado o nome do arquivo que descreve o grafo de entrada, o nó raiz lê o número de vértices que este contém. Neste ponto, é realizado um procedimento de balanceamento de carga, que é responsável pela alocação dos vértices nos processadores. Feita esta alocação, o processador raiz envia uma mensagem para todos os outros processadores informando qual a distribuição dos vértices pelos nós da rede. A partir desta informação os processadores têm condições de definir os paramêtros que irão orientar o processamento local.

O nó raiz começa, então, a ler a descrição do grafo, enviando as devidas informações para os outros nós. A figura IV.2 apresenta o procedimento de iniciação do nó raiz. Nas seções seguintes são vistas em detalhe todas as tarefas contidas na iniciação do sistema.

IV.2.1 Balanceamento de Carga

Com o objetivo de equilibrar o volume de processamento efetuado por cada processador, foi adotado um procedimento de balanceamento de carga. Como os algoritmos implementados se baseiam no processamento dos vértices do grafo de entrada, o balanceamento de carga se dá através da distribuição destes vértices pela rede de processadores.

O vértice fonte e o vértice dreno são sempre alocados no processador

Processador 0 - { 0, 1, 2, 12 }
Processador 1 - { 3, 4, 5 }
Processador 2 - { 6, 7, 8 }
Processador 3 - { 9, 10, 11 }

[0, 2, 3, 5, 6, 8, 9, 11]

Figura IV.3: Distribuição de 13 vértices em 4 processadores e o respectivo Vetor de Limites.

raiz, pois o tempo gasto com o processamento destes vértices é muito pequeno, uma vez que a fonte só envia fluxo no início da execução dos algoritmos e, durante a determinação do fluxo máximo propriamente dita, a única atividade destes vértices é receber fluxo.

O restante dos vértices é dividido pelo número total de processadores da rede. Caso a divisão não seja exata, o resto, que é no máximo igual ao número de processadores menos um, é distribuído pelos processadores da rede com exceção do raiz. Desta forma a diferença do número de vértices de um processador qualquer para outro é, no pior caso, igual a um. A diferença do processador raiz para outro processador qualquer é no máximo igual a dois, sendo que estes dois vértices correspondem à fonte e ao dreno.

Os vértices contidos em cada processador possuem identidades consecutivas. A identidade da fonte é sempre zero e a identidade do dreno é igual ao número total de vértices do grafo menos um. Os valores da menor e da maior identidade dos vértices de cada processador são armazenados em vetor denominado vetor de limites dos processadores *(Lim.Proc)*. As duas primeiras posições deste vetor correspondem, respectivamente, aos limites inferior e superior do processador 0, as duas posições seguintes correspondem aos limites do processador 1 e assim sucessivamente. Na construção deste vetor, o dreno não é consideradado. Caso o número de vértices do grafo de entrada seja menor que o número de processadores da rede, as posições referentes aos processadores vazios são preenchidas com zero.

IV.2.2 Difusão dos Parâmetros de Controle

O processamento em cada nó é regido por parâmetors que são definidos no início da execução dos sistemas de cálculo do fluxo máximo. O processador raiz, após realizar o procedimento de balanceamento de carga, difunde o vetor de limites obtido para todos os processadores da rede, inclusive para aqueles que porventura forem ficar vazios. Através do vetor de limites, cada processador têm condições de determinar os parâmetros que orientarão o processamento local, além de conhecer o conteúdo de todos os processadores da rede. Os parâmetros de controle são:

N.Proc - número de processadores ativos da rede

NV.Total - número total de vértices do grafo de entrada

NV.Local - número de vértices alocados no processador

Lim.Inf - identidade do menor vértice alocado no processador

Lim.Sup - identidade do maior vértice alocado no processador

Dreno - identidade do vértice dreno

OFS - *offset* de memória

O processador que possuir NV.Local igual a zero, não enviará nem receberá mais nenhuma mensagem, permanecendo inativo durante o resto da execução do sistema. Quando isto acontece o número de processadores ativos da rede é decrementado. A figura IV.4 apresenta o procedimento de determinação dos parâmetros de controle a partir do vetor de limites, onde *idp* é a identidade do processador.

IV.2.3 Formato do Arquivo de Entrada

Antes de abordar a difusão de vizinhos e capacidades é conviniente apresentar o formato do arquivo de entrada. Este arquivo contém a descrição do grafo no qual se deseja determinar o fluxo máximo, sendo composto pelo número total de vértices do grafo e pela descrição das arestas que o formam.

```
Lim.Inf = Lim.Proc[2*idp]
Lim.Sup = Lim.Proc[(2*idp)+1]
IF Lim.Inf = 0 AND Lim.Sup = 0
   THEN
     NV.Local = 0
   ELSE
     NV.Local = Lim.Sup - Lim.Inf + 1
     N.Proc = número total de processadores da rede
     Nao.Achou = TRUE
     WHILE Nao.Achou
       IF Lim.Proc[(2*N.Proc)-1] = 0
         THEN
           N.Proc = N.Proc - 1
         ELSE
           NV.Total = Lim.Proc[(2*N.Proc)-1] + 2
           Dreno = NV.Total - 1
           OFS = Lim.Inf
           Nao.Achou = FALSE
```

Figura IV.4: Procedimento de Determinação dos Parâmetros Controle.

A descrição de uma aresta é feita através da identidade do vértice origem, identidade do vértice destino e a sua capacidade. Os vértices são numerados a partir de zero, identidade da fonte, até o número total de vértices menos um, identidade do dreno.

IV.2.4 Difusão das Listas de Vizinhos e Capacidades

Cada vértice possui duas estrutura de dados básicas, que são: a lista de vizinhos e o vetor de capacidades. Um vértice é vizinho de outro quando existe uma aresta que os ligue. A lista de vizinhos de um vértice v é a relação de todos os vértices que estão ligados a v por uma aresta, seja ela convergente ou divergente. O vetor de capacidades de um vértice v contém as capacidades residuais das arestas que ligam v a seus vizinhos. Inicialmente as capacidades residuais são iguais as capacidades

originais do grafo de entrada.

Em cada processador existem duas matrizes encarregadas de armazenar as listas de vizinhos e os vetores de capacidades de seus vértices. Estas matrizes são indexadas pela identidade local do vértice e a posição da lista de vizinhos ou vetor de capacidades que se deseja acessar. A identidade local de um vértice é definida pela sua identidade subtraída do *offset* de memória do processador no qual está alocado. Este *offset* de memória corresponde ao valor da menor identidade dos vértices contidos no processador. Deste modo, um processador que contenha os vértices 15, 16 e 17, possui *offset* de memória igual a 15 e atribui identidades locais 0, 1 e 2 para estes vértices respectivamente. A primeira linha da matriz de vizinhos corresponde à lista de vizinhos do vértice que possui identidade local 0, a segunda linha corresponde ao vértice com identidade local 1 e assim sucessivamente. A indexação da matriz de capacidades é análoga. Este procedimento permite uma melhor utilização da memória disponível.

O número máximo de elementos da lista de vizinhos e do vetor de capacidades é igual ao número total de vértices do grafo de entrada. Como a alocação de memória é estática, o dimensionamento das estruturas de dados tem que ser feito considerando-se o pior caso. A lista de vizinhos de um vértice v é constituída de um contador, que indica o número de vértice que v possui, seguido da relação de vizinhos de v em uma ordem arbitrária. O vetor de capacidades de v contém as capacidades residuais das arestas que ligam v aos demais vértices do grafo de entrada. As posições referentes a arestas que não existem são preenchidas com zero.

A medida que o processadador raiz lê as arestas do grafo de entrada, ele envia mensagens para os devidos processadores para que estes preencham as listas de vizinhos e vetores de capacidades de seus vértices. Ao ler a seguinte descrição de aresta: v w $c(v,w)$, o processador raiz envia uma mensagem para o processador responsável por v contendo: $(v, w, c(v, w))$. Quando esta mensagem é recebida, w é colocado na lista de vizinhos de v, e a capacidade c(v,w) é colocada no vetor de capacidades de v. Em seguida o processador raiz envia para o processador responsável por w a mensagem: (v, w) e v é colocado na lista de vizinhos de w.

IV.2.5 Conceito de Processador Vizinho

Um processador A é considerado vizinho de um processador B, quando A contém pelo menos um vértice que seja vizinho de algum vértice contidos em B. Desta maneira um processador pode ser vizinho de outro mesmo que eles não sejam adjacentes na rede de processadores, isto é, mesmo que eles não estejam ligados por um canal físico de comunicação. Por outro lado, dois processadores podem ser adjacentes e não serem vizinhos. As relações de vizinhança são determinadas pelos vértices que cada processador contém, e não pela topologia da rede de processadores.

Antes de iniciar o cálculo do fluxo máximo cada processador determina quais são os seus processadores vizinhos. Esta determinação é feita com base nas listas de vizinhos dos vértices contidos no processador. A identidade dos processadores vizinhos é armazenada em uma lista *(Lis.Viz)*, que é utilizada na sincronização dos processadores.

IV.3 Comunicação entre Processadores

Em cada nó da rede, existe um processo de comunicação que é executado em paralelo com o processo de determinação do fluxo máximo. Os processos de comunicação constituem o Processador Virtual de Comunicação (CVP) [6], que é encarregado de realizar o roteamento, a transmissão e a recepção de mensagens entre os processadores.

Quando um processo de determinação de fluxo máximo deseja enviar uma mensagem para um outro, ele a envia para o seu respectivo processo de comunicação. Este, então, se encarrega de transmiti-la para o processo de comunicação alocado no processador destino, que finalmente envia a mensagem para o processo de determinação do fluxo máximo desejado. Como nem sempre a mensagem é destinada a um processador adjacente, torna-se necessário que esta passe por processadores intermediários até chegar ao processador destino. Os processos de comunicação são encarregados de determinar a rota que as mensagens devem seguir de modo a assegurar que não ocorra *deadlock* de comunicação.

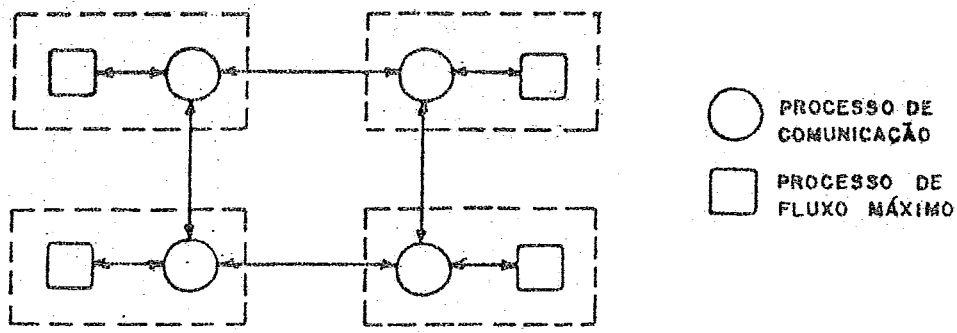

Figura IV.5: Estrutura de Comunicação entre Processadores.

A troca de mensagens entre os processos de comunicação e os respectivos processos de determinação do fluxo máximo é realizada através de canais virtuais internos a cada processador, enquanto a troca de mensagens entre os processos de comunicação é feita através dos canais físicos da rede de comunicação. A figura IV.5 apresenta a estrutura de comunicação entre processadores.

IV.4 Sincronização

Existem dois modelos básicos de comunicação entre processadores:

- rede de comunicação assíncrona
- rede de comunicação síncrona

Uma rede assíncrona é uma rede de comunicação ponto-a-ponto, descrita por um grafo $G = (V, E)$, onde o conjunto de vértices V representa os processadores da rede e o conjunto de arestas E corresponde aos canais bidirecionais que interligam os processadores. Não há memória compartilhada entre os nós da rede. Cada processador recebe mensagens de seus processadores adjacentes, realiza um processamento local e envia mensagens para os processadores adjacentes. Todas as mensagens chegam aos seus destinos em um tempo finito e indeterminado.

Uma rede síncrona é uma versão sincronizada do modelo anterior. Neste modelo, cada processador mantém um relógio local que gera pulsos satisfazendo a seguinte propriedade:

> *Uma mensagem enviada pelo processador* **i** *para o processador* **j** *enquanto* **i** *está no pulso* **p**, *chega a* **j** *antes que o pulso* **p+1** *seja gerado em* **j**.

Isto significa que quando um certo pulso é gerado em um processador, todas as mensagens enviadas para este processador em pulsos anteriores já foram recebidas e processadas.

Um algoritmo assíncrono é um algoritmo originalmente projetado para ser executado em uma rede assíncrona; do mesmo modo, um algoritmo síncrono é um algoritmo projetado para atuar em uma rede síncrona. Os algoritmos assíncronos geralmente são mais eficientes que os síncronos, entretanto são consideravelmente mais complexos em termos de projeto e análise.

Um sincronizador de alogoritmos é um protocolo distribuído que permite executar algoritmos síncronos em ambientes assíncronos. O sincronizador funciona gerando pulsos de relógio em cada nó da rede. Um pulso novo é gerado em um nó somente depois que este tenha recebido todas as mensagens referentes ao pulso anterior do algoritmo síncrono, enviadas por seus vizinhos. Um nó é dito seguro com respeito a um dado pulso do relógio síncrono quando todas as mensagens enviadas por ele naquele pulso tiverem chegado a seus destinos.

O sincronizador utilizado neste trabalho é uma versão modificada do sincronizador α proposto por AWERBUCH [5]. No sincronizador α, cada nó espera receber as confirmações de todas as mensagens que ele enviou em um determinado pulso. Quando isto ocorre, significa que o nó está seguro em relação a este pulso, e informa a todos os seus vizinhos que está seguro. Um nó só pode iniciar um novo pulso quando souber que todos os seus vizinhos estão seguros com relação ao pulso anterior.

A versão do sincronizador α proposta neste trabalho dispensa a confirmação das mensagens recebidas, pois cada nó envia e recebe mensagens de todos os seus vizinhos a cada pulso do algoritmo. Deste modo, todos os nós conhecem o número de mensagens que devem enviar e receber a cada pulso. Quando um nó tiver enviado e recebido todas as mensagens referentes a um determinado pulso, significa que este nó está apto a iniciar um novo pulso.

Cada mensagem de sincronismo contém as informações que um nó precisa transmitir para seu vizinho para que este continue seu processamento corretamente. Estas informações podem ser a respeito de transferências de fluxo, mudanças de rótulos, vértices bloqueados, etc.

É possível que em um certo pulso, um nó não tenha nenhuma informação para transmitir à um determinado vizinho. Neste caso é enviada uma mensagem vazia (DUMMY), isto é, uma mensagem sem valor para o processamento do nó destino, porém necessária para efeito de sincronização. Como cada nó é responsável por vários vértices do grafo de entrada, dificilmente uma mensagem vazia é enviada.

IV.4.1 Modelo do Sincronizador Utilizado

O sincronizador utilizado é composto de dois processos executados em paralelo, um é responsável pelo envio das mensagens de sincronismo, enquanto o outro é encarregado da recepção. A execução destes processos é feita em paralelo para evitar a formação de ciclos na comunicação, o que implicaria em *deadlock* [27].

Se houvesse um único processo encarregado de enviar e receber mensagens, poderia acontecer de um nó A tentar enviar uma mensagem para um nó B ao mesmo tempo que B tentasse enviar uma mensagem para A. Nesta situação, A ficaria indefinidamente tentando enviar a mensagem para B, um vez que B não poderia receber a mensagem de A porque estaria ocupado tentando enviar uma mensagem para A e vice versa. A figura IV.6 (a) ilustra a formação de ciclo na comunicação com a utilização de um único processo para receber e enviar mensagens, enquanto a figura IV.6 (b) mostra que com um processo para enviar mensagens e outro para

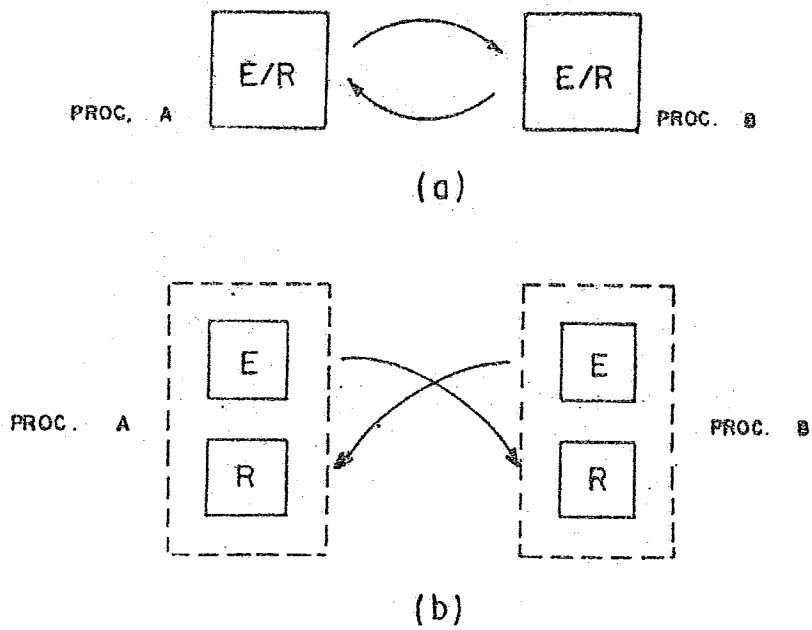

Figura IV.6: Sincronizador com 1 e 2 processos.

receber não ocorre formação de ciclos.

No processo de envio de mensagens de sincronismo, o processador transmite uma mesagem para cada elemento da sua lista de processadores vizinhos. Nesta mensagem estão contidas todas as informações que o processador vizinho necessita para continuar o seu processamento corretamente. O processo de envio de mensagens de sincronismo termina quando tiver enviado mensagens para todos os processadores vizinhos.

O processo de recepção de mensagens de sincronismo espera receber uma mensagem de cada processador vizinho, quando isto acontece este processo termina. Pode acontecer que um processador A, após enviar a mensagem de sincronismo referente ao pulso p para o processador B, consiga passar para o pulso $p+1$ e enviar para B a mensagem referente a este pulso antes que B tenha recebido recebidos todas as mensagens referentes ao pulso p. Para que mensagens referentes a pulsos distintos não sejam misturadas e o número de mensagens recebidas a cada pulso possa ser contado corretamente, foi adotado o seguinte procedimento: quando se recebe pela primeira vez uma mensagem de um determinado processador, marca-se a sua identidade. Se durante algum pulso chegarem duas mensagens do

mesmo processador, a segunda mensagem é armazenada em um buffer de mensagens adiantas e somente será tratada quando o próximo pulso for atingido.

Quando uma mensagem adiantada é recebida o contador de mensagens, que determina o fim do processo de recepção, não é incrementado, isto só acontecerá no pulso seguinte quando o buffer de mensagens adiantas for lido. Como o processador só pode receber uma mensagem adiantada por processador vizinho a cada passo, o número de buffers de mensagens adiantadas é igual a quantidade de processadores vizinhos. Ao fim do processo de recepção, as identidades de todos os processadores são desmarcadas e contandor de mensagens é zerado.

Cada mensagem de sincronização deve possuir a identidade do processador que a enviou para que possa ser realizado o procedimento de marcação da identidade dos processadores. Como pode haver mais de um tipo de mensagens de sincronização, toda mensagem deve possuir um identificador definindo qual é o seu tipo. O primeiro elemento de todas as mensagens representa o seu tipo, o segundo elemento representa a identidade do processador origem.

Para detectar a terminação dos algoritmos de cálculo do fluxo máximo, é necessário que cada processador se comunique com o processador raiz durante a etapa de sincronização. Ao final de cada iteração dos algoritmos, os processadores verificam se, segundo a informação local, o fluxo pode ser máximo. Neste caso, na etapa de sincronização da iteração seguinte, o processador que tiver detectado a possibilidade de fim de processamento envia uma mensagem de fim local para o processador raiz. Se o processador raiz receber, na mesma etapa de sincronização, mensagens de fim local de todos os processadores da rede e tiver detectado a possibilidade de fim de processamento nele próprio, significa que o fluxo máximo foi atingido. O processador raiz envia, então, uma mensagem de fim de processamento para todos os nós da rede.

```
< Inicialização das variáveis locais >
  Cont.Mens = 0
  FOR id = 0 TO N.Proc DO
    Recebida[id] = FALSE
< Leitura do buffer de mensagens adiantadas >
  FOR  ptr = 0 TO cont.buffer DO
    CASE
      Buffer.Adiantadas[ptr][0] = Tipo.1
        Trata mensagem do Tipo 1
      Buffer.Adiantadas[ptr][0] = Tipo.2
        Trata mensagem do Tipo 2
      Buffer.Adiantadas[ptr][0] = Tipo.N
        Trata mensagem do Tipo N
    Cont.Mens = Cont.Mens + 1
    cont.buffer = 0
< Leitura das mensagens vindas de outros processadores >
  WHILE Cont.Mens < número.de.processadores.vizinhos
    Canal.de.Entrada ? mensagem[ ]
      IF Recebidas[mensagem[1]] THEN
        Guarda mensagem no buffer de adiantadas
        cont.buffer = cont.buffer + 1
      ELSE
        CASE
          mensagem[0] = Tipo.1
            Trata mensagem do Tipo 1
          mensagem[0] = Tipo.2
            Trata mensagem do Tipo 2
          mensagem[0] = Tipo.N
            Trata mensagem do Tipo N
        Cont.Mens = Cont.Mens + 1
        Recebida[mensagem[1]] = TRUE
```

Figura IV.7: Algoritmo de Recepção de Mensagens de Sincronismo

Capítulo V

Sistemas Desenvolvidos

Este capítulo apresenta os sistemas de cálculo do fluxo máximo desenvolvidos. Estes sistemas são:

GTS - baseado na versão síncrona do algoritmo de GOLDBERG & TARJAN

GTA - baseado na versão assíncrona do algoritmo de GOLDBERG & TARJAN

AWS - baseado no algoritmo de AWERBUCH

Nos sistemas síncronos, cada nó contém um processo dedicado ao cálculo do fluxo máximo, que é responsável pelo processamento de todos os vértices nele alocados. A cada pulso do algoritmo síncrono estes processos executam três tarefas básicas:

- cálculos
- transmissão e recepção de informações
- tratamento das informações recebidas

Os procedimentos realizados durante a etapa de cálculos, normalmente, geram informações que devem ser transmitidas a outros processos de cálculo do fluxo máximo. Esta troca de informações entre processos serve de base para a sincronização do sistema.

Como foi visto no capítulo anterior, para efetuar a sincronização, os nós enviam mensagens para todos os seus vizinhos e recebem mensagens de todos eles a cada pulso do algoritmo síncrono. Nestas mensagens de sincronismo, estão contidas todas as informações que os processos de cálculo do fluxo máximo devem transmitir para os outros processos. Assim sendo, estas mensagens possuem dois propósitos: realizar a sincronização entre processos e promover o intercambio de informações.

Quando algum nó não tem nenhuma informação para transmitir a um determinado vizinho, é enviada uma mensagem vazia, isto é, sem valor para o processamento do vizinho, útil apenas para fins de sincronização. Como em cada nó estão alocados vários vértices, dificilmente uma mensagem vazia será enviada.

A terminação dos algoritmos síncronos é detectada através do nó raíz. Durante a sincronização, todos os nós, além de se comunicarem com os seus respectivos vizinhos, se comunicam também com o nó raiz, o que permite que este verifique se o fluxo corrente é máximo. Neste caso, o nó raiz envia uma mensagem de terminação para todos os nós da rede e o algoritmo acaba.

No sistema assíncrono, em cada nó da rede de processadores, existem dois processos dedicados ao cálculo do fluxo máximo, que são executados em paralelo. O primeiro é responsável pelos cálculos e pela recepção e tratamento de mensagens, enquanto o segundo é encarregado da transmissão de mensagens. Esta divisão em dois processos é necessária para evitar *deadlocks*. O procedimento de detecção do fim do algoritmo se baseia no algoritmo de terminação distribuída proposto por DIJKSTRA [9].

V.1 Complexidades de Tempo e Comunicação

A complexidade é um parâmetro que permite avaliar, analicamente, a eficiência de algoritmos. A complexidade de tempo de um algoritmo síncrono corresponde ao número de pulsos efetuados pelo o algoritmo desde seu início até seu término, enquanto a complexidade de tempo de um algoritmo assíncrono representa o tempo

dispendido com a sua execução, considerando que o atraso referente a um canal é proporcional ao número de mensagens do algoritmo que são transmitidas através do canal. A complexidade de comunicação, também chamada de complexidade de mensagens, expressa a quantidade de mensagens enviadas durante a execução do algoritmo.

A introdução do sincronizador alterou as complexidades dos algoritmos originalmente síncronos. As complexidades dos algoritmos assíncronos resultantes são dadas pelas seguintes expressões:

$$T_A = T_S * T_{pulso}$$

$$C_A = C_S + T_S * C_{pulso}$$

onde,

T_A é a complexidade de tempo assíncrono

T_S é a complexidade de tempo síncrono

T_{pulso} é a complexidade de tempo do sincronizador referente a cada pulso do algoritmo síncrono

C_A é a complexidade de comunicação assíncrona

C_S é a complexidade de comunicação síncrona

C_{pulso} é a complexidade de comunicação do sincronizador referente a cada pulso do algoritmo síncrono

As complexidades de tempo e comunicação do sincronizador utilizado são, respectivamente:

$$T_{pulso} = O(1) \text{ e } C_{pulso} = O(m)$$

As complexidades dos algoritmos síncronos de GOLDBERG & TARJAN e AWERBUCH são:

$$T_S = O(n^2) \text{ e } C_S = O(n^3)$$

Aplicando-se estes valores as expressões (1) e (2), as seguintes complexidades de tempo e comunicação assíncronas:

$$T_A = O(n^2) * O(1) = O(n^2)$$

$$C_A = O(n^2) + O(n^2) * O(m) = O(n^2 m)$$

Estas complexidades são iguais às da versão assíncrona proposta por GOLDBERG & TARJAN, o que significa que, analiticamente, os sistemas implementados possuem a mesma eficiência.

V.2 Sistema GTS

No sistema de cálculo do fluxo máximo baseado na versão síncrona do algoritmo de GOLDBERG & TARJAN (GTS), o processamento dos vértices pode ser dividido em cinco etapas, que são repetidas até que o fluxo máximo seja encontrado. Estas etapas são:

1. Cálculo do fluxo a ser enviado
2. Cálculo do novo rótulo
3. Sincronização
4. Tratamento do fluxo recebido
5. Detecção de fim local

Inicialmente, cada vizinho da fonte contém um excesso de fluxo igual à capacidade da aresta que o liga à fonte. Todos os vértices com exceção da fonte e do dreno, são examinados de modo a enviar o máximo de fluxo possível em direção ao dreno. O caminho que o fluxo deve seguir é definido pela rotulação dos vértices.

O cálculo do fluxo a ser enviado por um vértice v é feito da seguinte forma: enquanto houver excesso em v, este examina sua lista de vizinhos à procura de um vértice w que satisfaça as seguintes condições: $rotulo(v) = rotulo(w) + 1$ e $cr(v, w) > 0$. Neste caso v está apto a enviar fluxo para w. A quantidade de fluxo a ser enviada corresponde ao valor do excesso de v ou da capacidade residual da aresta (v, w), o que for menor. Determinada esta quantidade, o excesso de v e a capacidade residual da aresta (v, w) são decrementados deste valor. As identidades dos vértices v e w juntamente com a quantidade de fluxo a ser enviada, são armazenadas no Buffer de Transferência de Fluxo. Este procedimento é repetido até que todo o excesso de v seja eliminado ou que todos os seus vizinhos já tenham sido examinados.

Se após tentar enviar fluxo para todos os vizinhos ainda restar excesso em v, inicia-se a etapa de cálculo do novo rótulo. Para este fim, determina-se dentre os vizinhos de v com capacidade residual positiva aquele que possui o menor rótulo. O novo rótulo de v será igual ao rótulo deste vizinho acrescido de uma unidade.

Na etapa de sincronização cada nó envia uma mensagem para cada um de seus vizinhos. As mensagens de sincronismo podem ser de quatro tipos:

DADOS - contém as identidades dos vértices locais que estão enviando fluxo para vértices contidos no nó destino, a identidade destes e as respectivas quantidades de fluxo. Também estão contidos nesta mensagem os valores atualizados dos rótulos dos vértices locais.

SINC - mensagem vazia, útil apenas para fins de sincronização.

FIM.LOCAL - indica que, segundo a informação local, o fluxo corrente pode ser máximo.

MAX - indica o final de processamento. este tipo de mensagem só é enviado pelo nó raiz.

A mensagem do tipo DADOS é preenchida a partir do Buffer de Transferência de Fluxo. Cada mensagem tem pelo menos dois elementos: seu tipo e a identidade do processador destino.

Quando uma mensagem do tipo DADOS é recebida, os valores do excesso do vértice destino assim como da capaciade residual da aresta que une o vértice destino ao vértice origem são acrescidos da quantidade de fluxo recebida. Os valores dos rótulos dos vértices do processador que enviou a mensagem também são corrigidos.

A recepção do fluxo enviado por vértices contidos no mesmo processador é feita através do exame direto do Buffer de Transferência de Fluxo. Os valores do excesso do vértice destino assim como da capacidade residual entre o vértice destino e o vértice origem são atualizados da mesma forma que na recepção de mensagens de outro processador.

Após o tratamento de todas as informações recebidas, é determinado se, de acordo com o a situação dos vértices locais, o fluxo pode ser máximo. Isto é feito verificando-se se o excesso de todos os vértces locais é igual a zero, neste caso ativa-se uma indicação de fim local. Na etapa de sincronização, cada nó testa a indicação de fim local. Caso esta esteja ativa, o nó envia uma mensagem de FIM.LOCAL para o nó raíz. Se, em um dado pulso, o nó raíz receber mensagens de FIM.LOCAL de todos os nós da rede e tiver sua própria indicação de fim local ativa, significa que o fluxo corrente é máximo. No pulso seguinte, o nó raíz envia então uma mensagem do tipo MAX para todos os nós da rede e o algoritmo termina.

V.3 Sistema GTA

Na implementação deste sistema, assim como na dos outros, cada processador é responsável pelo processamento de um subconjunto dos vértices do grafo de entrada. Para realizar este processamento, existem dois processo básicos, que são executados em paralelo: o processo principal e o processo buffer. O primeiro é responsável pelo processamento dos vértices propriamente dito e engloba o cálculo do fluxo a ser enviado, a rotulação dos vértices e a recepção e tratamento das mensagens. O segundo funciona como um buffer que recebe as mensagens que o processo principal deseja enviar para os outros processadores e as transmite efetivamente para os seus destinos.

A utilização deste processo buffer é necessária para evitar a ocorrência de *deadlocks* de comunicação. Se o processo principal além de receber mensagens também as enviasse, quando dois processadores tentassem, simultaneamente, enviar mensagens um para o outro, o processamento nestes processadores ficaria travado, pois cada um ficaria esperando que o outro recebesse a sua mensagem para poder continuar o processamento. Com a execução em paralelo dos processos de envio e recepção de mensagens, pode-se garantir que esta situação não ocorre, uma vez que sempre haverá um processo pronto para receber mensagens.

O único problema desta solução ocorre quando o processo que recebe e trata mensagens envia para o processo buffer as mensagens que devem ser encaminhadas para outros processadores. No momento em que isto ocorre, o processo de recepção deixa de estar pronto para receber e novamente pode ser formado um ciclo na comunicação, caracterizando *deadlock*. Para evitar que isto aconteça basta impedir que o processo principal transmita mensagens para o processo buffer enquanto este estiver transmitindo mensagens para outros processadores. Isto é feito através de uma mensagem de liberação. Quando o processo principal tarnsmite uma mensagem para o processo buffer, o processo principal fica impedido de enviar uma nova mensagem para o processo buffer até que este lhe envie uma mensagem de liberação. Esta mensagem só é enviada quando o processo buffer acaba de transmitir as mensagens recebidas anteriomente para os devidos destinos. A figura V.1 apresenta a estrutura de comunicação em um nó.

V.3.1 Processo Principal

O estrutura do processo principal se baseia no conceito de comandos guardados de DIJKSTRA [10]. Um comando guardado é composto por um guarda e uma lista de comandos. Um guarda pode ser um canal pelo qual se espera receber uma mensagem, uma expressão booleana, ou uma composição dos dois. Quando um guarda fica habilitado, a lista de comandos correspondente é excutada. O processo principal é constituído de uma seqüência de comandos guardados cujos guardas são testados continuamente, até que o fluxo máximo seja encontrado. O estrutura do processo principal é apresentada na figura V.2.

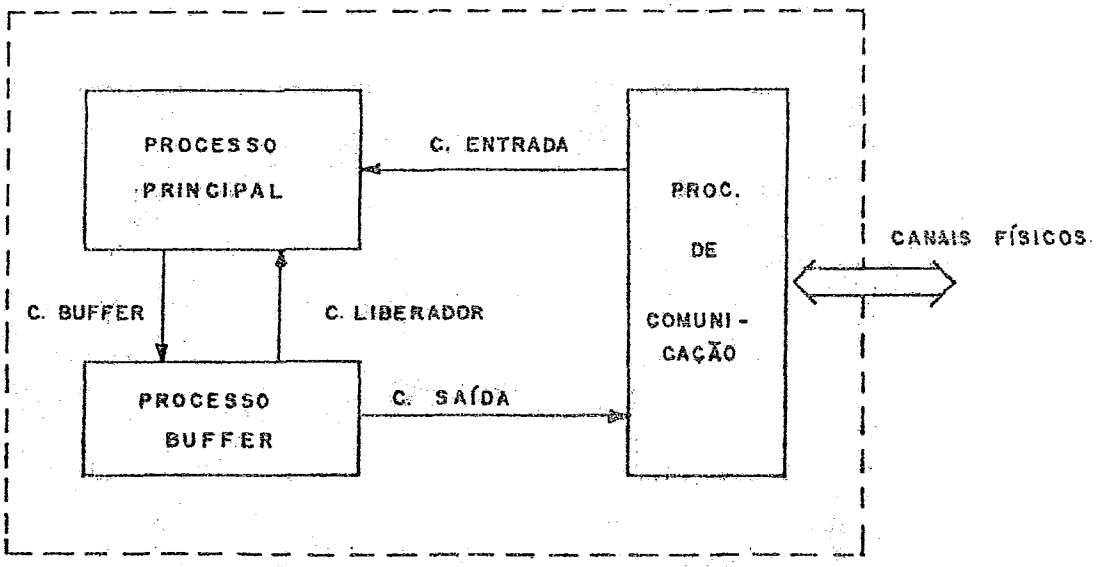

Figura V.1: Estrutura de Comunicação em um Nó

WHILE *Não.Terminou*

Tem.Excesso →
Cálculo do Fluxo a ser Enviado e Rotulação

Canal.de.Entrada ? mensagem[ ] →
Tratamento das mensagens recebidas de outros processadores

Canal.de.Liberação ? liberação →
Liberação do envio de mensagens para o processo buffer

Tem.Mensagem AND *Buffer.Liberado* →
Envio de mensagens para o processo buffer

Figura V.2: Estrutura do Processo Principal

O cálculo do fluxo a ser enviado e a rotulação são executados quando existe excesso de fluxo em algum dos vértices contidos no processador. Isto é indicado através de uma *flag* que funciona como guarda deste procedimento. O cálculo do fluxo a ser enviado é implementado de modo similar ao da versão síncrona, havendo apenas duas diferenças. A primeira é quanto ao armazenamento da quantidade de fluxo a ser enviada. Se o vértice para o qual o fluxo será enviado, vértice destino, estiver contido no mesmo processador que o vértice origem, então a atualização dos valores do excesso do vértice destino e da capacidade residual entre o vértice destino e o vértice origem é feita imediatamente, senão as identidades dos referidos vértices e a quantidade de fluxo a ser enviada são armazenadas em um buffer interno ao processo, que posteriormente será enviado para o processo buffer que transmitirá a mensagem para o processador destino. A segunda diferença determina que quando um vértice envia fluxo para um vértice contido em outro processador, ele só poderá enviar fluxo novamente para este vértice depois que receber dele uma resposta de aceitação ou rejeição em relação ao fluxo enviado. O procedimento de rotulação dos vértices é implementado do mesmo modo que na versão síncrona, sendo que neste caso coloca-se no buffer interno a indentidade do vértice que está sendo examinado e o seu novo rótulo.

A recepção de mensagens de outros processadores é feita através do canal que liga o processo principal ao processo que realiza a interface de comunicação. Este canal é o guarda do procedimento de recepção e tratamento de mensagens. Os tipos de mensagens exitentes são:

- FLUXO
- RESPOSTA
- RÓTULO
- CONFIRMAÇÃO
- TERMINAÇÃO

O primeiro elemento destas mensagens representa o tipo da mesma. A mensagem do tipo FLUXO é composta da seguinte maneira: tipo, vértice origem,

vértice destino e quantidade de fluxo, onde, vértice origem é a identidade do vértice que está enviando o fluxo e vértice destino é a identidade do vértice para o qual o fluxo está sendo enviado. A mensagem RESPOSTA tem o seguinte formato: tipo, aceitação/rejeição, vértice origem, vértice destino, quantidade de fluxo e rótulo do vértice origem. Nete caso, o vértice origem é aquele que está enviando a resposta e vértice destino é o que enviou o fluxo que esta sendo aceito ou rejeitado. A mesagem de RÓTULO é composta da identidade do vértice que mudou de rótulo e do valor do rótulo novo: tipo, vértice, rótulo do vértice. A mensagem de CONFIRMAÇÃO contém apenas o número de confirmações que estão sendo enviadas para um determinado processador: tipo, número de confirmações. A mesagem de TERMINAÇÃO não contém nenhum elemento além do tipo.

Quando uma mensagem tipo FLUXO é recebida, verifica-se se de acordo com informação local realmente o rótulo do vértice origem é igual ao rótulo do vértice destino mais um. Neste caso, o fluxo é aceito e os valores do excesso do vértice destino e da capacidade residual entre o vértice destino e o vértice origem são atualizados. Caso contrário o fluxo é rejeitado. Em ambas as situações é composta uma mensagem de RESPOSTA para o processador que contém o vértice origem. Esta mensagem é armazenada no buffer interno.

Ao se receber uma mensagem do tipo RESPOSTA, é verificado se o fluxo enviado foi aceito ou rejeitado. Caso o fluxo tenha sido rejeitado acrescenta-se ao excesso do vértce destino e a capacidade residual entre o vértice destino e o vértice origem a quantiadade de fluxo devolvida, e atualiza-se o valor do rótulo do vertice origem, caso contrário não há nada a fazer. Em seguida, o vértice destino é liberado para enviar fluxo para o vértice origem novamente.

Quando uma mensagem do tipo RÓTULO é recebida, atualiza-se o valor do rótulo do vértice recebido. Cada vez que chega uma mensagem do tipo FLUXO, RESPOSTA ou RÓTULO, é testado se esta é a primeira mensagem recebida pelo processador, neste caso a identidade do processador origem é guardada como "pai" do processador em questão, caso contrário incrementa-se o número de confirmações que serão mandadas para o processador origem.

Quando uma mensagem do tipo CONFIRMAÇÃO é recebida, subtrai-se do contador de mensagens, o número de confirmações contidos na mensagem. Ao receber a mensagem de TERMINAÇÃO, o processo principal envia um sinal de fim de processamrnto para o processo buffer e desabilita a *flag* que o mantém ativo.

A recepção da mensagem que libera o envio de mensagens para o processo buffer é feita através de um canal que liga o processo buffer ao processo principal. Quando uma mensagem de liberação é recebida habilita-se uma *flag* indicando que o processo buffer está livre.

O envio de mensagens para o processo buffer é feito sob duas condições: o processo buffer estar livre e haver mensagens no buffer interno para serem enviadas. O buffer interno é composto de quatro matrizes de buffers, uma para cada tipo de mensagem (FLUXO, RESPOSTA, RÓTULO e CONFIRMAÇÃO). O processador raiz além de enviar estes tipos de mensagens ainda envia a mensagem de TERMINAÇÃO.

V.3.2 Processo Buffer

O processo buffer pode ser dividido em duas etapas, na primeira é feita a recepção de todas mensagens contidas no buffer interno do processo de recepção e tratamento de mensagens. À medida que estas mensagens vão chegando, elas vão sendo armazendas em um buffer intermediário. Após a recepção de todas as mensagens é que começa a segunda etapa, onde as mensagens contidas neste buffer intermediário são lidas e encaminhadas para os respectivos destinos. A determinação do processador destino de uma mensagem é feita utilizando-se uma função que recebe como parâmetro o vértice destino da mensagem e a partir do vetor de limites, descobre-se em que processador este vértice está alocado.

Após o envio da última mensagem é verificado se foi recebido um sinal de terminação, neste caso a *flag* que mantém o processo buffer ativo é desabilitada, caso contrário envia-se a mensagem de liberação para o processo de recepção e tratamento de mensagens.

O processo buffer do processador raiz é ligeiramente diferente dos processos buffer contidos nos demais processadores, pois além de enviar as mensagens de FLUXO, RESPOSTA, RÓTULO E CONFIRMAÇÃO, ele também envia a mensagem de TERMINAÇÃO.

V.3.3 Procedimento de Terminação

Para detectar o fim de processamento, foi adotado o algoritmo de terminação distribuída de DIJKSTRA [9]. Segundo este algoritmo cada mensagem enviada por um processador deve ter o seu recebimento confirmado pelo processador destino. Ao receber a primeira mensagem, cada processador, com exceção do raiz, armazema a identidade do nó que a enviou como sendo seu *pai*, sem confirma-la. Desta maneira estabelece-se uma árvore de espalhamento [30] na rede de processadores, onde a origem é o nó raiz, que é o primeiro processador a enviar uma mensagem.

Cada nó possui um contador de mensagens, que é incrementado sempre que o nó envia uma mensagem e decrementado quando uma confirmação é recebida. Quando o contador de mensagens de um nó chega a zero, este, então, confirma a mensagem enviada pelo seu *pai*. Um nó pode trocar de *pai* durante o processmento. O fim de processamento é detectado quando o contador de mensagens do nó raiz chega a zero. Este, então, envia uma mensagem de terminação para todos os nós da rede.

V.4 Sistema AWS

O algoritmo de AWERBUCH é dividido em duas etapas, que são executadas até que o fluxo máximo seja encontrado. Na primeira etapa é construída uma rede em camadas e na segunda é calculado o fluxo maximal na rede em camadas obtida. O algoritmo prosegue até que se construa uma rede em camadas da qual o vértice dreno não faça parte, o que significa que não há mais nenhum caminho aumentante da fonte para o dreno, neste ponto fluxo atual é máximo. A implementação deste algoritmo possui a seguinte estrutura:

```
WHILE Não.Terminou
    Construção da Rede em Camadas
    IF ( Dreno.Presente )
        THEN Cálculo do Fluxo Maximal
        ELSE Nao.Terminou = FALSE
```

V.4.1 Construção da Rede em Camadas

A construção da rede em camadas começa com a colocação da fonte, s, no nível 0 da rede, isto é, na primeira camada. Em seguida todos os vizinhos da fonte são examinados. Um vizinho v da fonte é colocado no nível 1 da rede se $cr(s, v) > 0$. No passo seguinte, são examinados todos os vizinhos dos vértices do nível 1 da rede. Se v é um vértice do nível 1 e w é um vizinho de v, coloca-se w no nível 2 se $cr(v, w) > 0$ e w não pertencer a nenhum outro nível da rede em camadas. Este procedimento continua até que os vértices contidos na última camada preenchida não possuam nenhum vizinho que satisfaça as condições para colocação na rede em camadas. A Figura V.3 apresenta a estrutura da rede em camadas.

Durante a construção da rede em camadas são preenchidas as listas de vizinhos divergentes de cada vértice na rede. Estas listas servem de base para o cálculo do fluxo maximal na segunda etapa do algoritmo. A construção da rede em camadas é feita de forma distribuída, sendo que cada processador é responsável pelo exame dos vértices nele contidos. Cada nó possui uma lista que contém os próximos vértices que serão examinados naquele nó e um buffer onde são armazenados todos os vértices que o nó colocou na rede em camadas no passo corrente do algoritmo.

O algoritmo examina todos os vizinhos dos vértices contidos na lista de próximos, colocando na lista de vizinhos divergentes de um dado vértice o vizinhos que satisfizer as condições para colocação na rede em camadas. Caso este vizinho ainda não esteja na rede este é marcado com nível igual ao passo do algoritmo e é colocado no buffer. A cada passo do algoritmo de construção da rede em camadas um nível da rede é preenchido, deste modo o valor do passo corresponde ao nível da rede que está sendo construído.

Após todos os vizinhos da lista de próximos terem sido examinados esta é esvaziada. Verifica-se então quais os vértice contidos no buffer que estão alocados no nó em questão e estes são colocados na lista de próximos. Neste ponto é realizada a sincronização entre processadores, quando são trocadas informações sobre os vértices que cada processador colocou na rede no passo atual. Esta sincronização pode ser feita em uma fase ou em duas.

Na sincronização em uma fase, cada processador envia uma mensagem igual para todos os outros processadores da rede a cada etapa de sincronização. Um processador, ao receber uma mensagem destas, utiliza apenas parte do seu conteúdo. Na sincronização em duas fases, cada processador somente se comunica com seus vizinhos, enviando para eles apenas as informações necessárias. Esta comunicação, entretanto, ocorre duas vezes a cada etapa de sincronização.

Na sincronização em uma fase, cada processador envia o conteúdo do seu buffer para todos os outros processadores e recebe o conteúdo do buffer de todos eles. Quando um buffer é recebido, atribui-se a cada vértice nele contido, nível igual ao passo e coloca-se os vértices pertencentes ao nó na lista de próximos. O algoritmo termina quando os buffers enviados por todos os nós estiverem vazios.

Na sincronização em duas fases, na etapa inicial, cada processador envia para seus nós vizinhos um subconjunto do buffer, que contém apenas os vértices pertencentes a cada processador vizinho. Quando um processador recebe esta mensagem coloca seu conteúdo na lista de próximos. Na segunda etapa cada nó envia para seus vizinhos a sua lista de próximos completa e recebe a mesma informação referente a seus vizinhos. Quando uma mensagem deste tipo é recebida, atribui-se a todos os vértices nela contidos nível igual ao passo atual.

Na sincronização em duas fases a terminação do algoritmo de construção da rede em camadas é feita da seguinte maneira: na segunda etapa os nós além de se comunicarem com os vizinhos, se comunicam também com o nó raiz. Quando as suas lista de próximos estão vazias estes enviam uma mensagem de FIM.LOCAL para o processador raiz, caso contrário, se o nó raiz não for vizinho, enviam uma mensagem vazia.

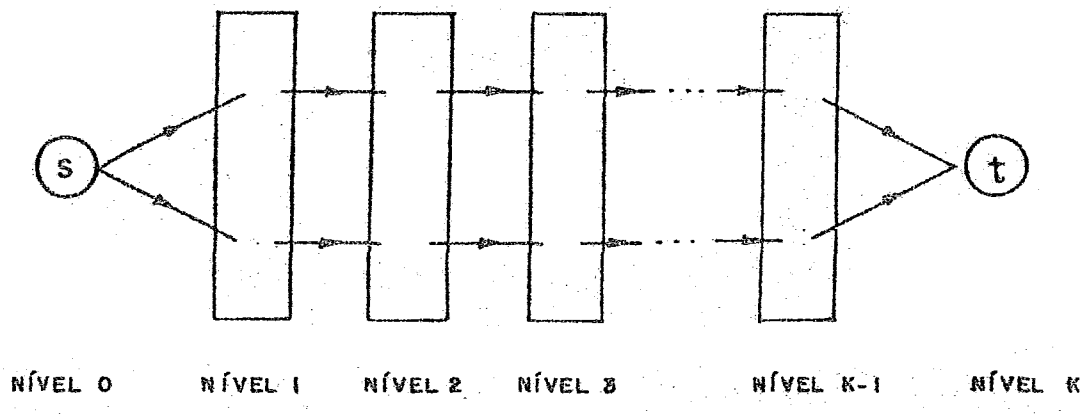

Figura V.3: Estrutura da Rede em Camadas

O nó raiz conta o número de mensagens de FIM.LOCAL recebidas, e se todos os nós inclusive o raiz tiverem terminado, no passo seguinte do algoritmo, na primeira etapa de sincronização, o nó raiz envia uma mensagem de FIM.RC para todos os outros nós. Quando isto ocorre a segunda etapa de sincronização não é executada e o algoritmo de construção da rede em camadas termina.

A vantagem da sincronização em uma fase é que na grande maioria dos casos o número de mensagens trocadas é menor que na sincronização em duas etapas. Por outro lado na sincronização em duas etapas só são enviadas as informações necessárias para cada processador.

V.4.2 Determinação do Fluxo Maximal

O algoritmo de determinação do fluxo maximal pode ser dividido em cinco etapas que são repetidas até que o fluxo maximal seja encontrado. Estas etapas são:

1. Cálculo do fluxo a ser enviado
2. Cálculo do fluxo a ser devolvido
3. Sincronização
4. Tratamento o fluxo recebido

5. Tratamento dos vértices bloqueados

6. Verificação de fim local

Na etapa de cálculo do fluxo a ser enviado, todos os vértices são examinados de modo a enviar a maior quantidade possível de fluxo para seus vizinhos divergentes. O cálculo do fluxo a ser enviado por um vértice v é feito da seguinte maneira: enquanto houver excesso em v, este envia para cada elemento da sua lista de vizinhos divergentes uma quantidade de fluxo igual ao seu excesso ou à capacidade residual da aresta que une v ao vizinho, o que for menor. Quando v envia fluxo para um vértice w, a quantidade de fluxo enviada é subtraída do excesso de v e da capacidade residual da aresra (v, w). As identidades de v e w assim como a quantidade de fluxo enviada de v para w são armazenadas no Buffer de Envio de Fluxo. Se a aresta (v, w) se tornar saturada, w é retirado da lista de vizinhos divergentes de v. A etapa de cálculo do fluxo a ser enviado termina quando o excesso de$v$ se torna igual a zero. No segundo caso, v é marcado como bloqueado.

Se após a etapa de cálculo do fluxo a ser enviado, ainda restar excesso em v, este começa a devolver o fluxo recebido em pulsos anteriores de seus vizinhos convergentes. Esta devolução de fluxo é feita a partir da pilha que contém a identidade dos vizinhos que enviaram fluxo para v e as quantidades de fluxo correspondentes. O devolução de fluxo começa retirando-se a última informação colocada na pilha. Caso o excesso em v seja menor que a quantidade armazenada na pilha, devolve-se para o vértice origem a diferença entre estas quantidades e coloca-se novamente na pilha o vértice origem e a quantidade de fluxo atualizada, isto é, descontando-se o fluxo devolvido. Caso contrário, devolve-se para o vértice origem todo o fluxo recebido dele e continua-se retirando informações da pilha até que todo o excesso remanescente seja devolvido. Quando v devolve fluxo para um vértice w, a quantidade de fluxo devolvida é subtraída do excesso de v e da capacidade residual da aresta (v, w). As identidades de v e w assim como a quantidade de fluxo enviada de v para w são armazenadas no Buffer de Devolução de Fluxo. Ao final da etapa de cálculo do fluxo a ser devolvido, o excesso nos vértices é igual a zero.

A etapa de sincronização deste algoritmo e o procedimento de detecção de fim de processamento são análogos aos da vers so síncrona do algoritmo de GOLDBERG & TARJAN, sendo que a mensagem do tipo DADOS ao invés de conter a lista dos novos rótulos, contêm a relação dos vértices que se tornaram bloqueados na iteração anterior do processador que enviou a mensagem.

Quando uma mensagem do tipo DADOS é recebida, os valores do excesso do vértice destino assim como da capaciade residual entre o vértice destino e vértice origem são atualizados imediatamente, o que evita que seja feita uma copia dos dados recebidos. No caso de fluxo devolvido, esta atualização consiste no acréscimo da quantidade de fluxo recebida ao excesso do vértice destino e a capacidade residual entre o vértice destino e o vértice origem. Já no caso de fluxo enviado além destas atualizações, a identidade do vértice origem e a correspondente quantidade de fluxo são armazenados na pilha referente ao vértice destino.

A recepção do fluxo enviado ou devolvido por vértices contidos no mesmo nó é feita através do exame dos Buffers de Envio e Devolução de Fluxo e o tratamento destes se realiza da mesma forma que o tratamento do fluxo vindo de outros nós.

Capítulo VI

Avaliação

Este capítulo trata da avaliação dos algoritmos implementados, o que compreende a análise do comportamento destes algoritmos quando submetidos a diferentes tipos de grafos de entrada, a comparação de seus desempenhos, o *speedup* obtido para configurações com 2, 4 e 8 processadores e a análise da utilização da memória por cada algoritmo.

Com o objetivo de realizar uma ampla bateria de testes, foram utilizadas três famílias de grafos de entrada. Cada uma destas famílias caracteriza-se por um parâmetro específico, cuja variação pode influenciar significativamente o desempenho dos algoritmos. Estes parâmetros são :

- densidade de arestas
- relação profundidade/largura
- capacidade máxima

Os valores assumidos por estes parâmetros definem as classes de entrada para suas respectivas famílias. O tamanho de um grafo é especificado pela quantidade de vértices que ele possui. Para obter o tempo médio de processamento para cada situação testada, foram geradas 8 amostras de grafos por classe de entrada para cada tamanho de grafo. Cada família possui seu próprio gerador de grafos.

A comparação do desempenho dos algoritmos implementados foi feita com base no tempo gasto exclusivamente com a determinação do fluxo máximo,

sem considerar o tempo dispendido com a iniciação do sistema, isto é, no tempo de processamento propriamente dito. Por limitações de memória foram testados grafos com até 2^9 vértices. Maiores detalhes sobre esta metodologia de testes e os geradores de grafos podem ser encontradas em [11].

VI.1 Utlilização da Memória

Esta seção aborda a utilização da memória por cada algoritmo implementado. Como a alocação de memória é feita estaticamente, torna-se necessária a definição de constantes para estabelecer as dimensões das principais estruturas de dados usadas pelos algoritmos. Estas constantes são:

$NVTM$ - representa o número total máximo de vértices do grafo de entrada previsto pelo algoritmo

$NVLM$ - representa o número máximo de vértices que pode ser alocado em um processador

O valor de $NVLM$ corresponde, aproximadamente ao valor de $NVTM$ dividido pelo número de processadores da rede, $NPROC$. Na versão síncrona do algoritmo de GOLDBERG & TARJAN, cada vértice possui as seguintes estruturas de dados:

Estrutura de Dados	Dimensão	Num. Bytes
Vizinhos	$NVTM$	$2 * NVTM$
Capacidade	$NVTM$	$4 * NVTM$
Cap. Residual	$NVTM$	$4 * NVTM$
Excesso	1	4
Rotulo	1	2

Assim sendo, cada vértice ocupa $(10 * NVTM + 6)$ bytes. Na versão assíncrona, os vértices possuem as mesmas estruturas de dados que na versão síncrona

e ainda um vetor de booleanos, que indica se o vértice pode ou não enviar fluxo para um determinado vizinho. Neste caso, cada vértice ocupa $(11 * NVTM + 6)$ bytes. No algoritmo de AWERBUCH, cada vértice ocupa $(18 * NVTM + 4)$ bytes e possui as seguintes estruturas de dados:

Estrutura de Dados	Dimensão	Num. Bytes
Vizinhos	$NVTM$	$2 * NVTM$
Capacidade	$NVTM$	$4 * NVTM$
Cap. Residual	$NVTM$	$4 * NVTM$
Viz. Divergentes	$NVTM$	$2 * NVTM$
Pilha Viz.	$NVTM$	$2 * NVTM$
Pilha Quant.	$NVTM$	$4 * NVTM$

Como pode ser observado, o espaço que cada vértice ocupa é proporcinal a $NVTM$ nas três versões implementadas. Cada processador comporta $NVLM$ vértices, ou seja, $NVTM/NPROC$. Deste modo, a memória ocupada pelos vértices em cada processador é da ordem de $NVTM^2/NPROC$. A memória disponível, 2 Mbytes, permitiu que fossem feitos testes com grafos com no máximo 2^9 vértices, utilizando 8 processadores. Já em configurações com 1, 2 e 4 processadores, foram feitos testes com nó máximo 2^8 vértices.

Além das estruturas de dados correspondentes aos vértices, cada algoritmo utiliza um série de buffers que também concorrem para a ocupação da memória.

VI.2 Influência da Densidade de Arestas

A densidade de um grafo corresponde ao número de arestas que ele contém e pode ser expressa através do grau de seus vértices. As classes de entrada para esta família vão desde de grafos esparsos, com poucas arestas, até grafos densos, com muitas arestas.

VI.2.1 Gerador de Grafos

O gerador desta família dispõe os vértices do grafo, com exceção da fonte e do dreno, em uma matriz quadrada, chamada de matriz de geração. Cada vértice possui um número específico de arestas divergentes que o ligam a vértices aleatoriamente escolhidos na coluna seguinte à que ele ocupa na matriz de geração. São passados como parâmetros para este gerador: a ordem da matriz quadrada a ser construída, o grau de saída dos vértices (o que caracteriza a densidade do grafo) e a capacidade máxima das arestas.

A fonte está ligada a todos os vértices da primeira coluna da matriz de geração, assim como o dreno está ligado a todos os vértices da última coluna. O valor da capacidade das arestas que ligam a fonte e o dreno aos seus respectivos vizinhos é igual ao triplo do valor especificado para a capacidade máxima. O valor da capacidade de cada aresta é um valor aleatoriamente escolhido entre zero e o valor da capacidade máxima. Nos testes realizados a capacidade máxima foi definida como 10^4.

VI.2.2 Comportamento Obtido

O gráfico da figura VI.1 apresenta o comportamento dos três algoritmos implementados com relação a variações na densidade de arestas, para cinco tamanhos de grafos distintos. O eixo vertical corresponde ao tempo de processamento em segundos e o eixo horizontal corresponde aos graus de saída dos vértices do grafo para todos os tamanhos de grafos testados.

Observando-se este gráfico, nota-se que à medida que a densidade aumenta, existe inicialmente uma elevação do tempo de processamento, porém, logo em seguida, se dá uma reversão desta tendência e o tempo de processamento começa a cair.

Este comportamento ocorre porque quando a densidade é muito alta, o fluxo inicial enviado pela fonte é quase todo transmitido para o dreno, neste caso o tempo de processamento é curto. À medida que a densidade diminui, o

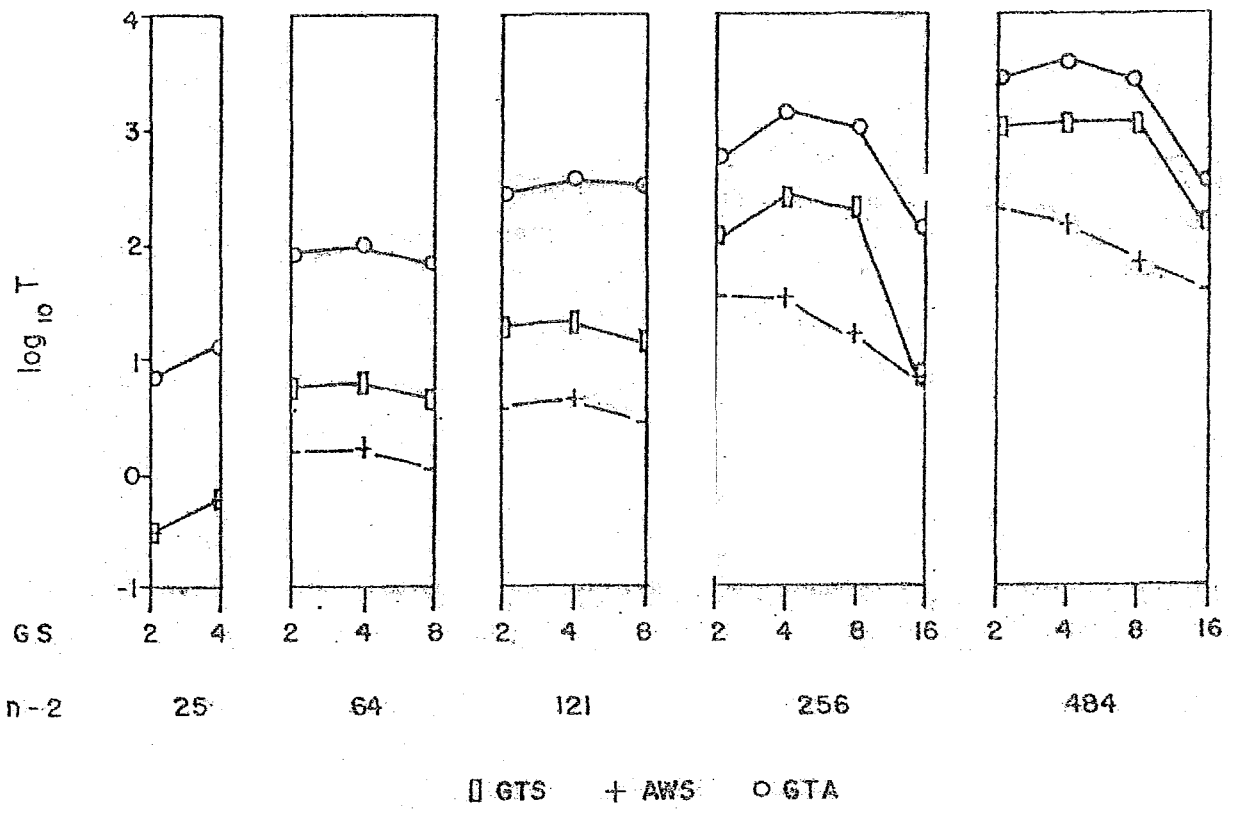

Figura VI.1: Influência da densidade de arestas em 5 tamanhos de grafos.

escoamento de fluxo em direção ao dreno se torna mais difícil, e a quantidade de fluxo devolvida para a fonte tende a aumentar, o que acarreta uma elevação do tempo de processamento.

Quando o grafo se torna muito esparso o fluxo a ser devolvido para fonte é logo identificado, transitando pouco pelo grafo, e o tempo de processamento volta a diminuir. Para grafos do mesmo tamanho, o fluxo inicial enviado pela fonte é constante. Dependendo do fluxo inicial o ponto de alteração do comportamento pode variar.

VI.2.3 Comparação entre Algoritmos

Para grafos pequenos, isto é, com poucos vértices, os sistemas baseados no algoritmo de AWERBUCH (AWS) e a versão síncrona de GOLDBERG & TARJAN (GTS) apresentaram praticamente o mesmo tempo de processamento. A medida que o tamanho do grafo aumenta, estes algoritmos permanecem equivalentes para densidades elevadas, quando a maior parte do fluxo enviado pela fonte chega até o dreno.

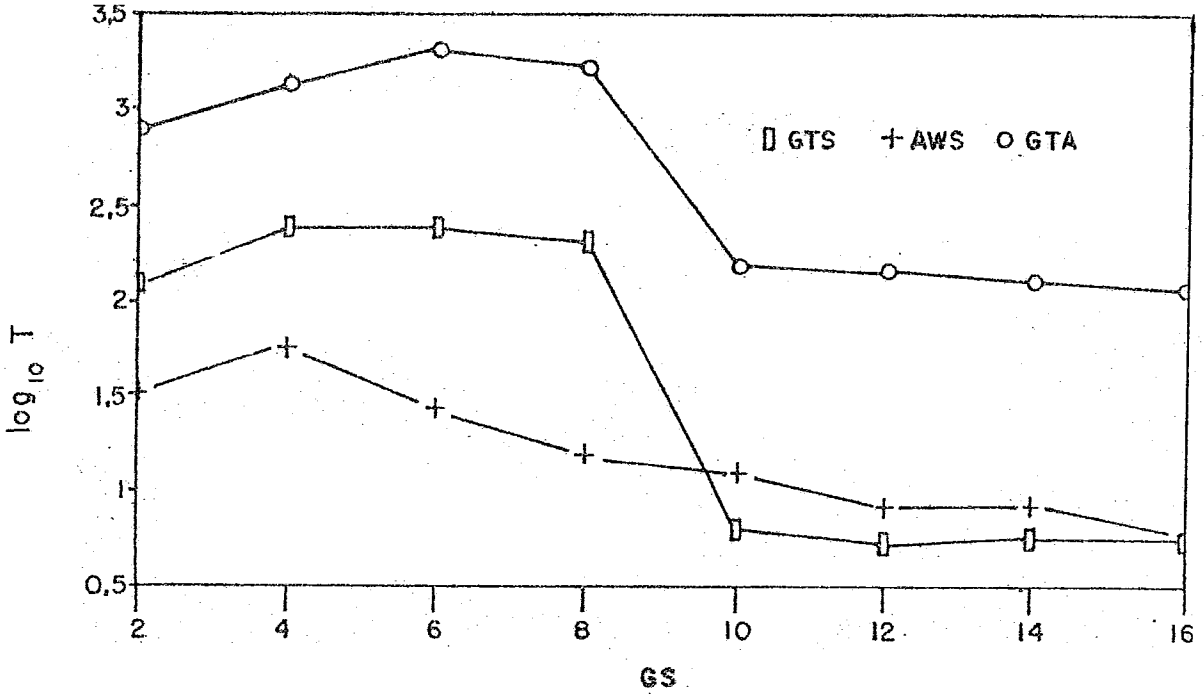

Figura VI.2: Influência da densidade de arestas em um grafo com 256 vértices.

Para densidades mais baixas, entretanto, o algoritmo de AWERBUCH se mostra mais eficiente. O gráfico da figura VI.2 apresenta o comportamento dos algoritmos para um grafo de 256 vértices, com vários valores de densidade.

O sistema baseado na versão assíncrona de GOLDBERG & TARJAN (GTA) apresentou os maiores tempos de processamento, porém, à medida que o tamanho dos grafos aumenta, a diferença dos tempos alcançados por este algoritmo para os demais diminui.

A figura VI.3 apresenta uma tabela contendo o tempo de processamento (em segundos) obtido por cada algoritmo para diferentes tamanhos de grafos e suas respecticas variações na densidades.

VI.2.4 Grafos Muito Densos

A densidade máxima proporcionada pelo gerador da família de densidade é bastante limitada, pois o grau de saída dos vértices pode ser no máximo igual à ordem da

Número de Vértices	Densidade (Grau de Saída)	Tempo de Processamento(s)		
		AWS	GTS	GTA
32	2	0.32	0.34	6.31
	4	0.63	0.58	14.12
64	8	1.65	5.40	76.48
	4	1.57	5.96	99.35
	2	1.12	4.45	68.91
128	8	3.68	18.07	258.73
	4	4.15	20.60	359.39
	2	3.19	13.61	328.07
256	16	33.56	121.47	497.67
	8	31.73	238.96	1461.72
	4	15.66	202.14	958.57
	2	6.14	5.96	111.74
512	16	204.20	948.19	2768.33
	8	136.25	1063.89	3826.04
	4	63.63	1077.27	2392.02
	2	39.63	126.51	298.25

Figura VI.3: Tabela do tempo de processamento relativo a variações na densidade de arestas.

matriz de geração. Para realizar uma avaliação mais profunda no que se refere ao comportamento obtido para grafos densos, foram feitos testes utilizando um gerador apropriado para este tipo de grafo. Segundo este gerador, cada vértice i possui arestas que o ligam a todos os vértice numerados de $i+1$ até $n-1$, onde $n-1$ é a identidade do dreno e corresponde a maior identidade dentre os vértices do grafo. Os parâmetros de entrada para este gerador são: o número de vértices do grafo e a capacidade máxima das arestas. Neste gerador, a fonte, que possui identidade igual a zero, está ligada a todos os vértices do grafo.

O comportamento obtido para este tipo de grafo é apresentado no gráfico da figura VI.4. Observando-se este gráfico, verifica-se que o algoritmo de AWERBUCH apresenta os melhores tempos de processamento. Para grafos pequenos a versão síncrona de GOLDBERG & TARJAN alcançou tempos muito próximos aos obtidos pelo algoritmo de AWERBUCH, porém, para grafos grandes apresentou tempos sensivelmente maiores.

A versão assíncrona de GOLDBERG & TARJAN apresentou os piores tempos para a faixa de tamanhos de grafos testada, entretanto observa-se que a medida que o tamanho do grafo aumenta, os tempos alcançados por esta versão tendem a se aproximar dos tempos atingidos pela versão síncrona.

VI.3 Influência da Relação Profundidade/Largura

Os grafos desta família são caracterizados pelo parâmetro profundidade, que corresponde ao comprimento dos caminhos entre a fonte e o dreno, e pelo parâmetro largura, que corresponde ao número caminhos através de um corte separando a fonte do dreno. As classes de entrada para esta família representam uma faixa de relações profundidade/largura. Outros fatores como estrutura do grafo, densidade de arestas e capacidade máxima são mantidos constantes.

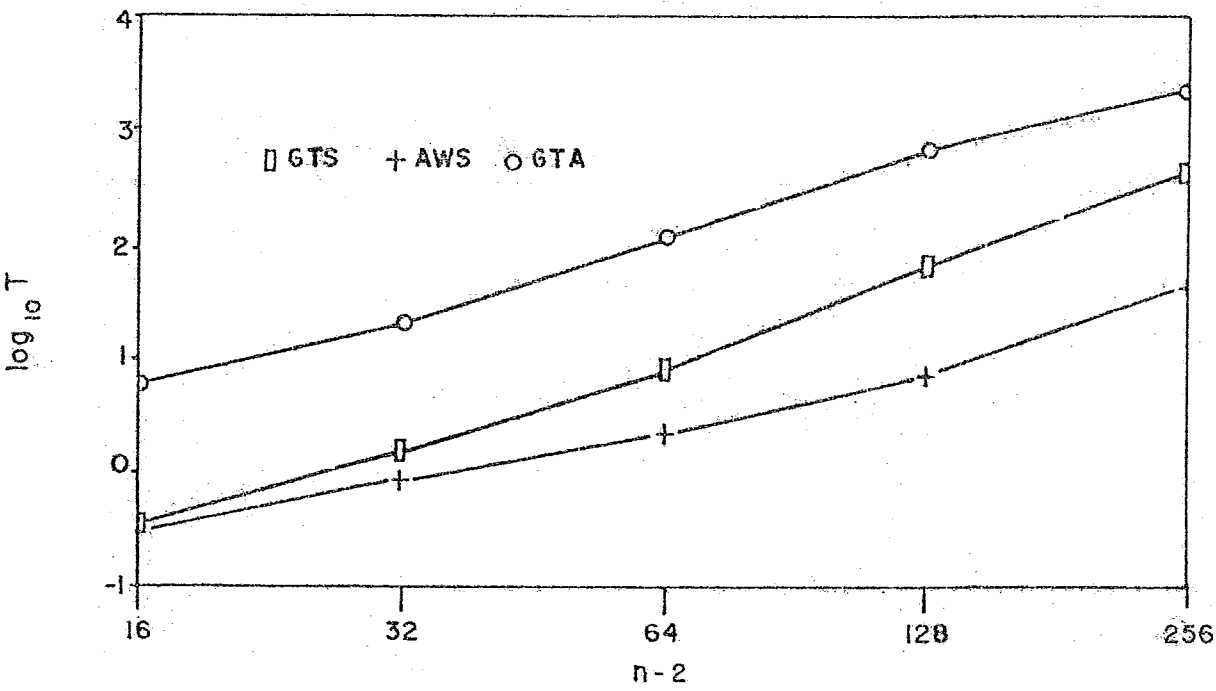

Figura VI.4: Influência da densidade de arestas em grafos muito densos.

VI.3.1 Gerador de Grafos

O gerador desta família também dispõe os vértices do grafo em uma matriz de geração, onde o número de linhas e colunas, assim como o valor da capacidade máxima, são argumentos de entrada para o gerador. O número de linhas está associado à largura do grafo e o número de colunas à profundidade. O produto entre o número de linhas e de colunas é igual à quantidade de vértices do grafo com exceção da fonte e do dreno.

Cada vértice possui três arestas divergentes que o ligam a vértices escolhidos aleatoriamente na coluna seguinte à que ele ocupa na matriz de geração. Como no gerador da família de densidade, a fonte está ligada a todos os vértices da primeira coluna da matriz de geração, assim como o dreno está ligado a todos os vértices da última coluna. O valor da capacidade das arestas que ligam a fonte e o dreno aos seus respectivos vizinhos é igual ao triplo do valor especificado para a capacidade máxima e o valor da capacidade de cada aresta é um valor aleatoriamente escolhido entre zero e o valor da capacidade máxima. A capacidade máxima das

arestas foi definida como 10^4. Para cada tamanho de grafo foram gerados exemplos com relações profundidade/largura distintas.

VI.3.2 Comportamento Obtido

Considerando um determinado tamanho de grafo, à medida que o número de linhas da matriz de geração (largura do grafo) cresce, o número de colunas (profundidade do grafo) diminui, uma vez que a quantidade de vértices é constante. O aumento da largura do grafo implica no aumento do número de caminhos entre a fonte e o dreno, assim como o diminição da profundidade significa a diminuição do comprimento destes caminhos e vice-versa.

O tempo de processamento necessário para que o fluxo máximo seja estabelecido aumenta à medida que a profundidade cresce, pois o número de vértices pelos quais o fluxo deve passar até atingir o dreno se torna cada vez maior.

A redução da largura provoca a diminuição do fluxo inicial enviado pela fonte, uma vez que ela está ligada a todos os vértices da primeira coluna da matriz de geração. Isto, porém, não leva a um tempo de processamento menor, pois o número de caminhos entre a fonte e o dreno também diminui.

O gráfico da figura VI.5 mostra o comportamento dos três algoritmos implemtados, no que se refere a tempo de processamento, para seis tamanhos de grafos distintos, considerando-se várias relações profundidade/largura para cada tamanho de grafo testado.

Este gráfico confirma o comportamento esperado, pois quando a profundidade aumenta e a largura diminui, o tempo de processamento aumenta. Nota-se, entretanto, que quando a profundidade fica muito grande e a largura muito pequena, o algoritmo de AWERBUCH apresenta uma queda no seu tempo de processamento. Isto se justifica através da tendência do número de redes em camadas se reduzir a medida que o grafo de entrada se torna muito estreito, como pode ser constatado no gráfico da figura VI.6 que apresenta o número médio de redes em camadas construídas para várias relações profundidade/largura, para cada tamanho

de grafos testado. Obeservando-se este gráfico percebe-se que o número máximo de redes em camadas ocorre quando a matriz de geração tende a se tornar quandrada.

VI.3.3 Comparação entre Algoritmos

Observando-se o gráfico da figura VI.5, nota-se que para grafos pequenos, isto é com menos de 40 vértices, os algoritmos síncronos de GOLDBERG & TARJAN e AWERBUCH são praticamente equivalentes no que se refere a tempo de processamento. À medida que o número de vértices aumenta, o algoritmo síncrono de GOLDBERG & TARJAN se torna mais lento que o de AWERBUCH para a maioria dos casos, só se equiparando a este para redes muito largas e pouco profundas.

A versão assíncrona de GOLDBERG & TARJAN se mostrou a mais lenta de todas, porém, à medida que o tamanho do grafo aumenta, seu tempo de processamento tende a se aproximar dos tempos obtidos pelos algoritmos síncronos. A tabela da figura VI.7 apresenta o tempo médio de processamento em segundos dos três algoritmos implementados para os tamanhos de grafos testados e suas respectivas relações profundidade/largura.

VI.4 Influência da Capacidade Máxima

Os grafos desta família representam variações no valor máximo da capacidade das arestas. O gerador para esta família é similar ao da família da relação profundidade/largura. Foram feitos testes com duas classes de entrada para esta família: capacidade máxima baixa, igual a 10^2, e capacidade máxima alta, 10^8. Os resultados obtidos mostraram que a variação do valor da capacidade máxima não exerceu influência no tempo de processamento dos algoritmos, apesar de proporcionar uma maior varidade de capacidades de arestas. O gráfico da figura VI.8 apresenta os tempos de processamento alcançados pelos três algoritmos implementados, para vários tamanhos de grafo, considerando as duas classes de entrada testadas.

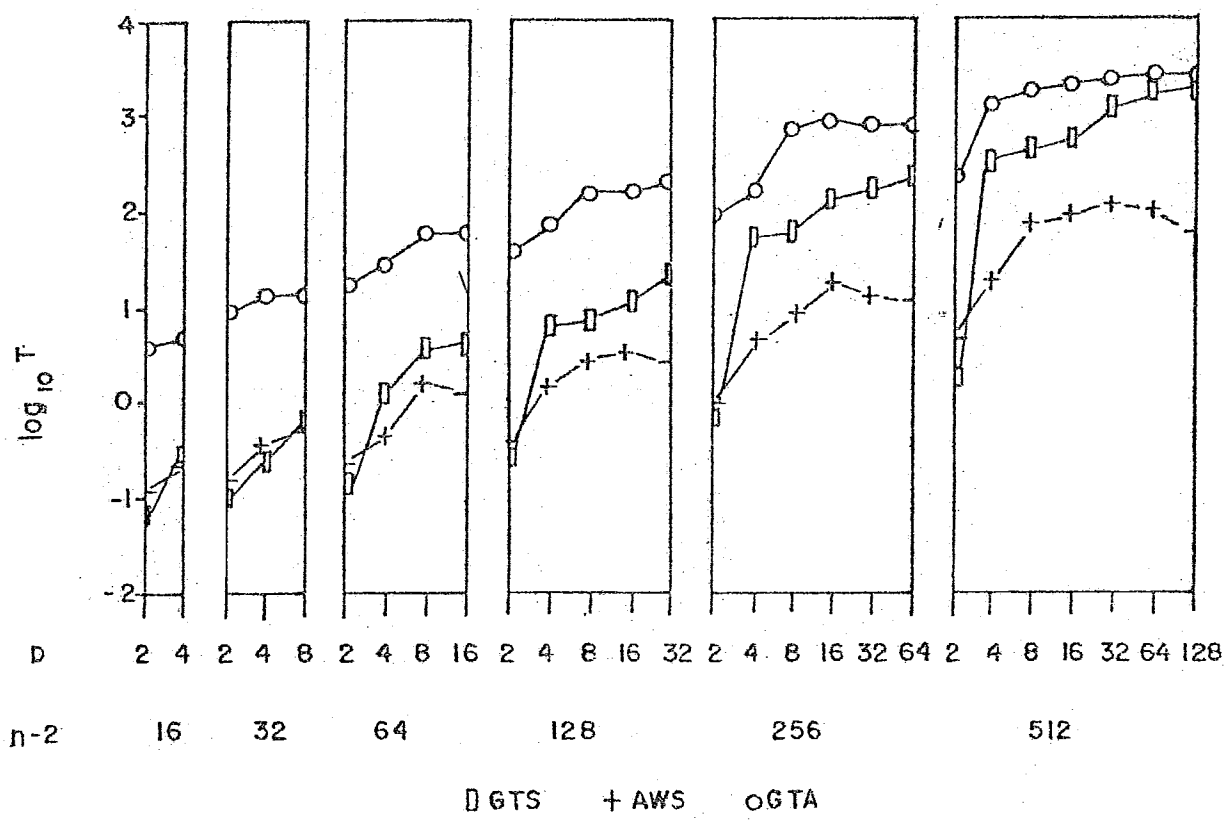

Figura VI.5: Influência da relação Profundidade/Largura em 6 tamanhos de grafos.

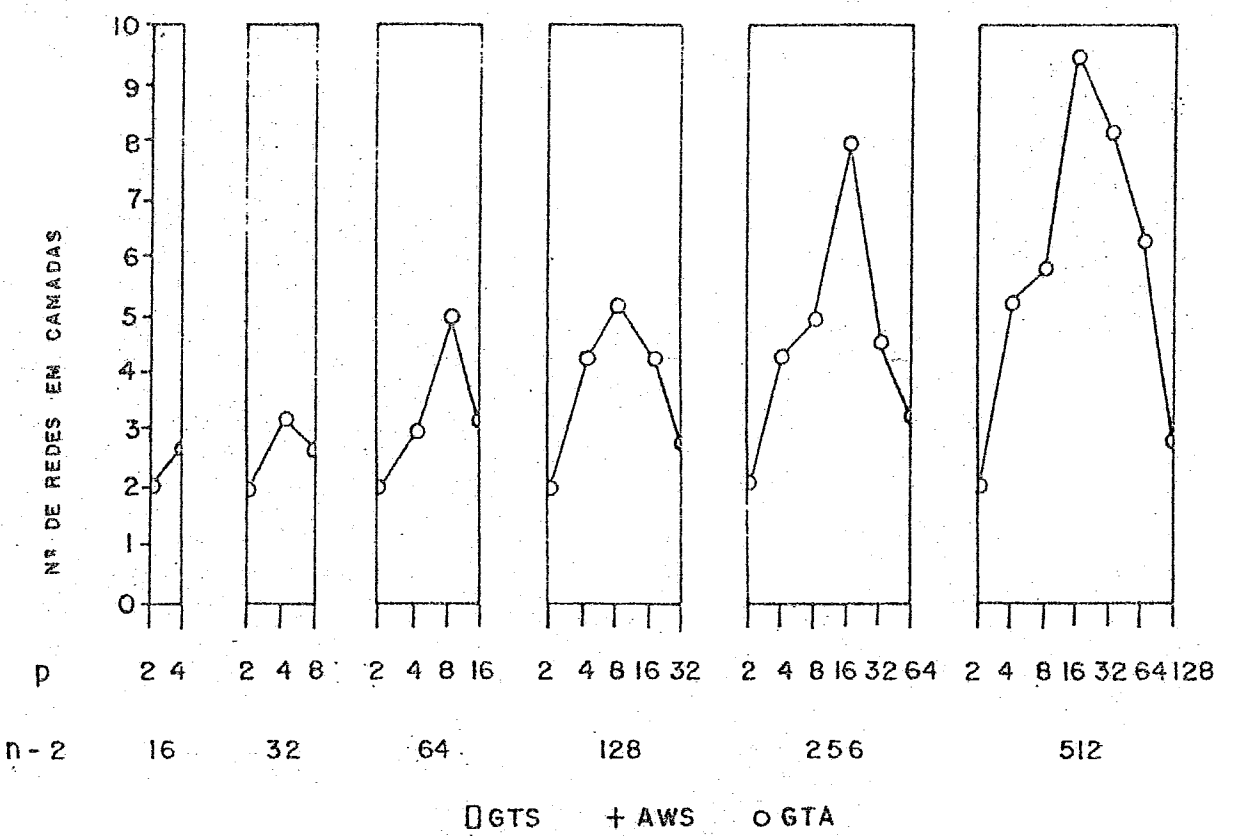

Figura VI.6: Número médio de Redes em Camadas.

Número de Vértices	Relação Profundidade/Largura	Tempo de Processamento(s)		
		ANS	GTS	GTA
16	2 x 8	0.11	0.07	4.24
	4 x 4	0.26	0.28	5.05
32	2 x 16	0.15	0.10	8.79
	4 x 8	0.37	0.25	14.32
	8 x 4	0.47	0.61	14.72
64	2 x 32	0.22	0.15	17.95
	4 x 16	0.45	1.23	30.13
	8 x 8	1.59	3.57	57.92
	16 x 4	1.15	4.13	58.93
128	2 x 64	0.39	0.29	36.49
	4 x 32	1.55	6.33	78.19
	8 x 16	2.64	6.81	159.06
	16 x 8	3.14	11.27	162.96
	32 x 4	2.26	21.82	192.72
256	2 x 128	0.86	0.66	79.60
	4 x 64	4.17	53.85	153.30
	8 x 32	7.07	60.10	787.84
	16 x 16	17.92	133.79	879.52
	32 x 8	12.87	157.10	807.27
	64 x 4	10.91	226.06	795.16
512	2 x 256	4.48	1.63	167.35
	4 x 128	18.37	324.95	1300.69
	8 x 64	69.28	444.41	1814.56
	16 x 32	87.36	540.97	2196.09
	32 x 16	109.91	1148.24	2446.18
	64 x 8	95.94	1952.90	2595.65
	128 x 4	50.61	2082.98	2531.66

Figura VI.7: Tabela do tempo de processamento relativo a variações da relação Profundidade/Largura.

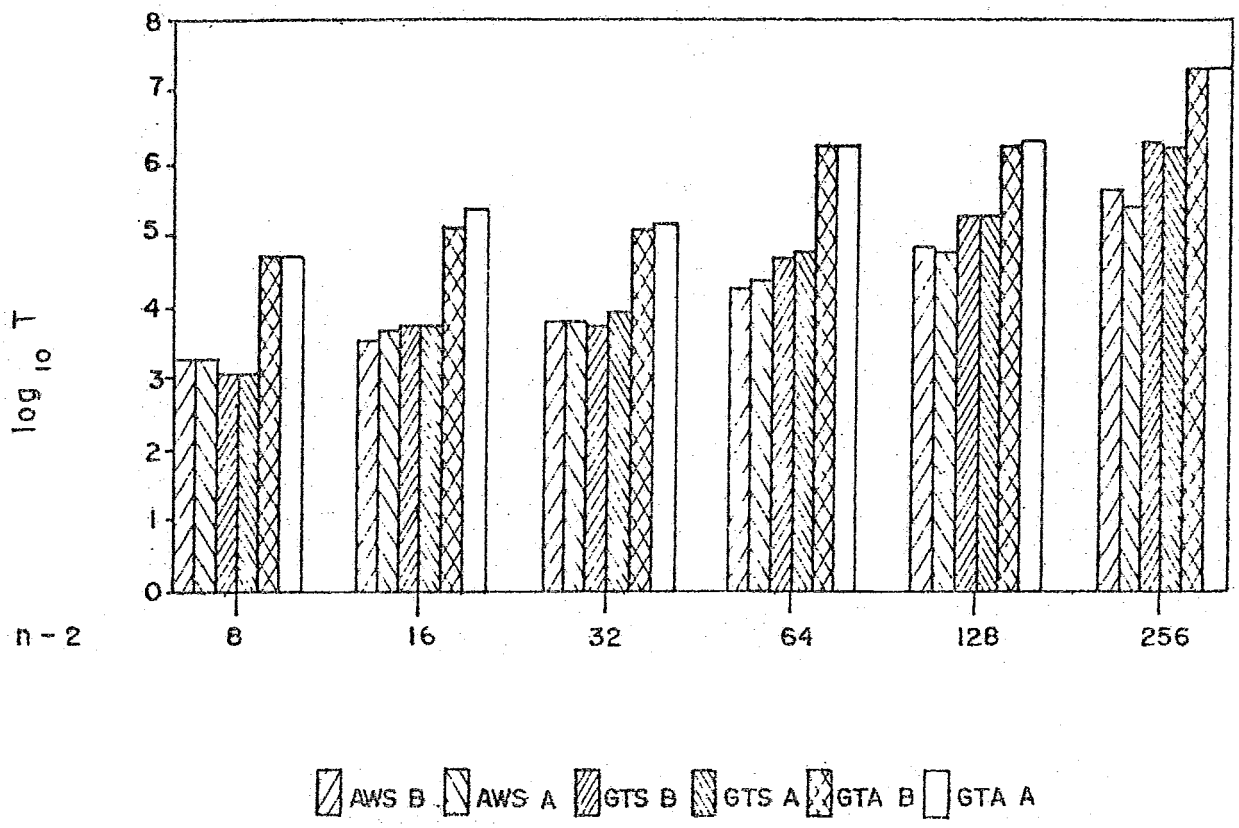

Figura VI.8: Influência da Capacidade Máxima.

VI.5 Comparação entre os Sincronizadores em 1 e 2 Estágios

Como foi visto no capítulo 5, a sincronização do procedimento de construção da rede em camadas do algoritmo de AWERBUCH, pode ser feita em um ou dois estágios. Na sincronização em um estágio, todos os processadores enviam uma mensagem para os demais a cada iteração do algoritmo de construção da rede em camadas, enquanto na sincronização em dois estágios os processadores só enviam mensagens para os seus vizinhos, o que ocorre duas vezes por iteração do algoritmo.

O gráfico da figura VI.9 apresenta o comportamento do algoritmo de AWERBUCH com a sincronização em um estágio e em dois estágios, em relação a grafos da família de densidade. Observando-se este gráfico, verifica-se que o desempenho obtido com estes dois tipos de sincronização é equivalente no que se refere a tempo de processamento.

Como o número de processadores utilizados é pequeno (no máximo

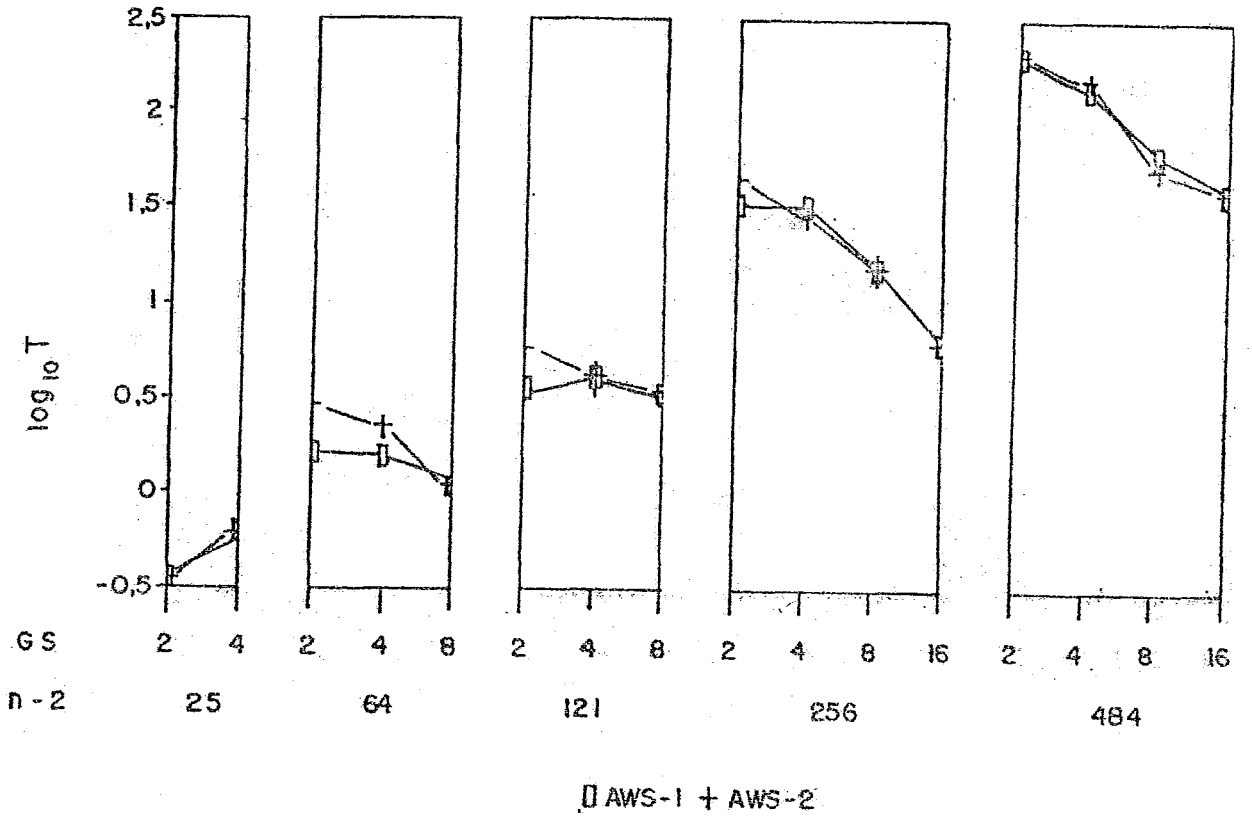

Figura VI.9: Comparação entre sincronização em 1 ou 2 estágios.

8) e o número de vértices dos grafos testados normalmente é grande em relação a o número de processadores, é natural que cada processador possua muitos vizinhos. Deste modo, o fato de, em toda etapa de sincronização, cada processador se comunicar com todos os outros ou apenas com seus vizinhos, não é relevante. Outro aspecto que deve ser considerado é que o tempo gasto com a construção das redes em camadas não é grande se comparado com o tempo total de processamento.

VI.6 Speedup

O *speedup* é um parâmetro fundamental para a avaliação do desempenho de sistemas paralelos, pois representa o ganho em velocidade de processamento obtido através da execução de uma aplicação em um ambiente com vários processadores. O *speedup* é definido como a relação entre o tempo de execução de uma aplicação em um processador e o tempo de execução da mesma aplicação quando operando uma rede de processadores.

O *speedup* ideal é igual ao número de processadores da rede, porém devido ao tempo gasto com o controle e a comunicação entre processos, o *speedup* ideal dificilmente é atingido. A *eficiência* de um sistema paralelo é o parâmetro que expressa o quanto um sistema está perto de atingir o *speedup* ideal, sendo definida como a relação entre o *speedup* e o número de processadores da rede.

Os gráficos das figuras VI.10, VI.11 e VI.12 apresentam o *speedup* obtido pelos sistemas AWS, GTS e GTA, respectivamente, em configurações com 2, 4 e 8 processadores, considerando grafos com densidade moderadamente baixa e com até 256 vértices.

O tempo de processamento é composto pelo tempo gasto com cálculos e o tempo gasto com a comunicação entre processadores. Nos algoritmos síncronos implementados, o número de mensagens por iteração é constante, entretanto, quanto maior o número de vértices do grafo de entrada, maior o tamanho das mensagens. Na versão assíncrona, quanto maior o número de vértices, maior o número de mensagens, que possuem sempre o mesmo tamanho.

Como pode ser observado nos gráficos, para grafos pequenos, isto é, com menos de 40 vértices, não compensa realizar o processamento em paralelo, pois o *speedup* é inferior a 1. Isto acontece porque o tempo gasto com a comunicação se torna muito grande em relação ao tempo gasto com os cálculos. À medida que o número de vértices aumenta torna-se cada vez mais vantajoso realizar o processamento em paralelo e o *speedup* se aproxima do ideal. A partir de um determinado tamanho de grafo o *speedup* tende a se estabilizar. Quanto menor o número de processadores, menor o tamanho de grafo para o qual o *speedup* se estabiliza.

O algoritmo de AWERBUCH apresentou bons resutados em termos de *speedup*, chegando próximo ao *speedup* ideal para configurações com 2 e 4 processadores, atingindo uma eficiência superior a 90%. Para 8 processadores a eficiência obtida foi de 87.5%, para grafos maiores, possivelmente uma eficiência maior seria atingida.

A versão síncrona de GOLDBERG & TARJAN também apresentou bons resultados. Na configuração com 2 processadores, o *speedup* se estabilizou

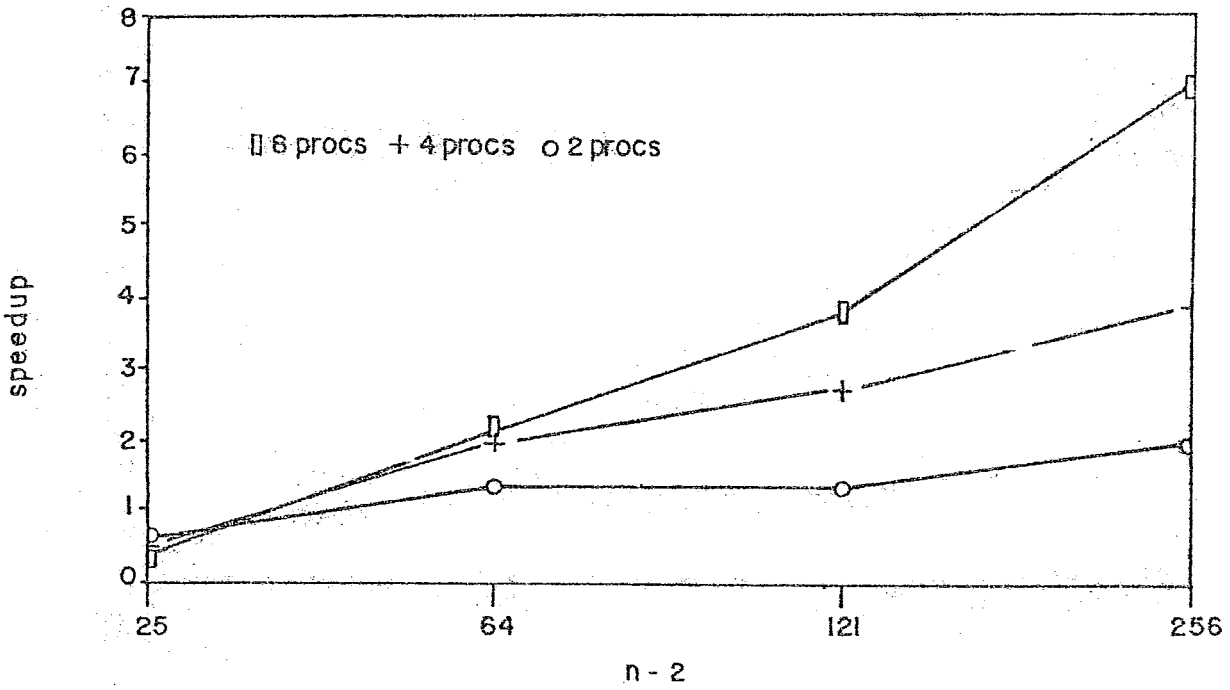

Figura VI.10: Speedup do sistema AWS

a partir de grafos com aproximadamente 60 vértices em 1.5, o que significa uma eficiência de 75%. Para 4 processadores a eficiência obtida foi de 80%, enquanto que para 8 processadores atingiu 90%.

A versão assíncrona de GOLDBERG & TARJAN apresentou os piores resultados. Para 2 processadores obteve uma eficiência próxima de 50%, para 4 processadores atingiu 42% e para 8 processadores chegou a 31%. Para analisar o desempenho obtido por esta versão, deve-se levar em conta, que os tamanhos de grafos testados foram pequenos por limitações de memória e que a versão assíncrona de GOLDBERG & TARJAN obteve seus melhores resultados para grafos grandes, apresentando a tendência de que quanto maior o número de vértices melhor o desempenho.

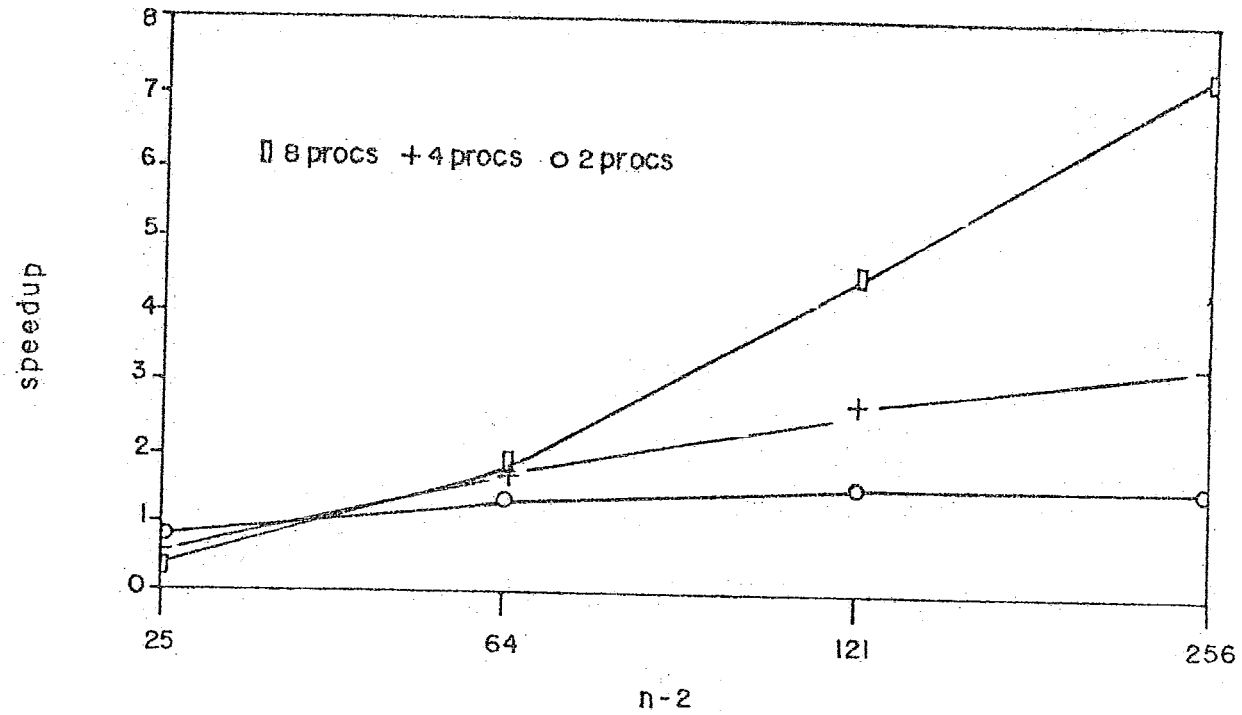

Figura VI.11: Speedup do sistema GTS

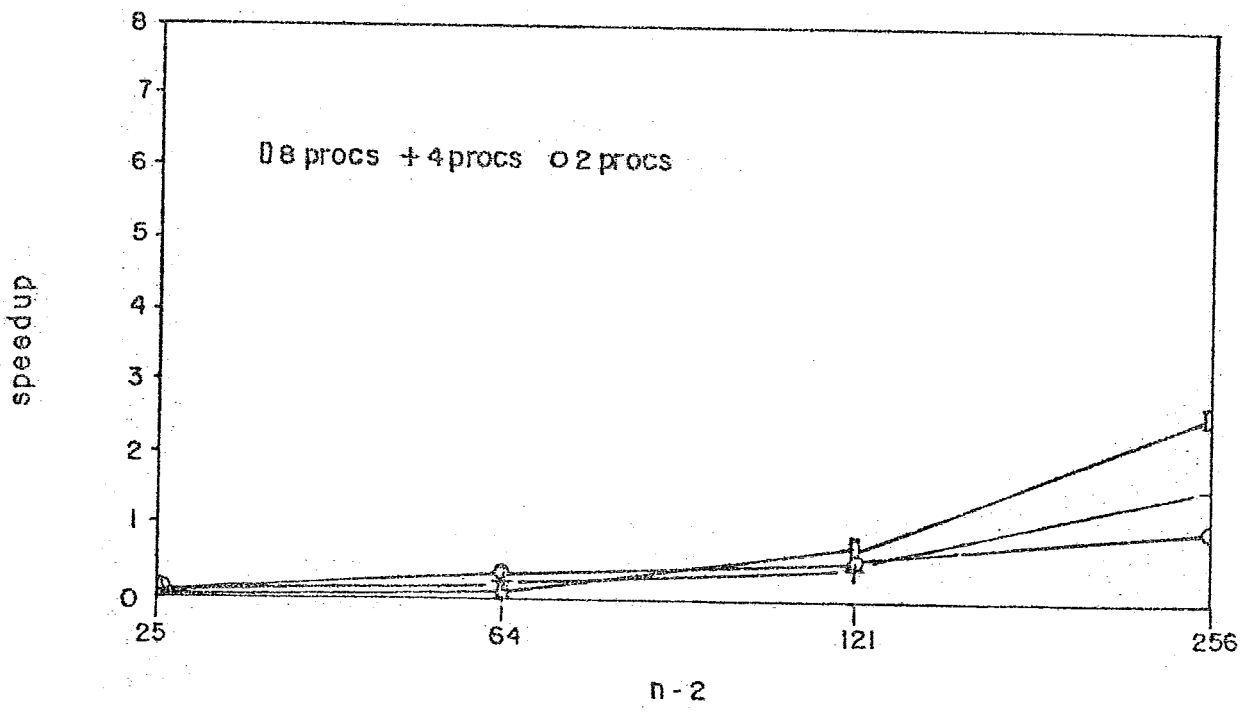

Figura VI.12: Speedup do sistema GTA

Capítulo VII

Conclusões

O sistema de cálculo do fluxo máximo baseado no algoritmo de AWERBUCH apresentou o melhor desempenho durante os testes realizados. O sistema baseado na versão síncrona do algoritmo de GOLDBERG & TARJAN também apresentou bons resultados, equiparando-se em alguns casos ao sistema derivado do algoritmo de AWERBUCH. Já o sistema correspondente à versão assíncrona de GOLDBERG & TARJAN mostrou-se o mais lento dos sistemas implementados, atingindo os maiores tempos de processamento, entretanto, à medida que o número de vértices do grafo de entrada aumenta, o desempenho deste sistema tende a se aproximar do obtido pelos demais.

A análise do comportamento dos sistemas implementados indica que o método de cálculo do caminho que o fluxo deve seguir, adotado por cada algoritmo, representa um aspecto fundamental no desempenho dos respectivos sistemas. No algoritmo de AWERBUCH, os caminhos que podem ser utilizados para enviar fluxo para o dreno (caminhos aumentantes) são determinados em uma etapa inicial, através da construção de uma rede auxiliar (rede em camadas). Neste algoritmo, o fluxo que não conseguir chegar ao dreno volta para a fonte através do mesmo caminho pelo qual veio, pois esta informação fica armazenada. No algoritmo de GOLDBERG & TARJAN, o caminho que o fluxo deve seguir, tanto para chegar ao dreno quanto para voltar para a fonte, é determinado através de cálculos baseados nos rótulos dos vértices.

A quantidade de fluxo que não consegue chegar ao dreno, e conse-

quentemente volta para fonte, é um fator de degradação do tempo de processamento. Quanto mais próximo ao dreno ocorre a retenção de fluxo, maior o custo em termos de tempo de processamento. O algoritmo de GOLDBERG & TARJAN é mais sensível a este fator que o de AWERBUCH, pois no algoritmo de GOLDBERG & TARJAN é necessário calcular o caminho de volta do fluxo. Na versão assíncrona do algoritmo de GOLDBERG & TARJAN ainda existe a possibilidade do fluxo enviado por um vértice ser rejeitado pelo vértice destino, devido a inconsistência entre os valores de seus rótulos. Neste caso, torna-se necessário corrigir o valor do rótulo desatualizado e tentar enviar o fluxo novamente, o que concorre para a degradação do tempo de processamento.

Ao analisar a influência da densidade de arestas no desempenho dos sistemas implementados observou-se que quando o grafo é muito denso o fluxo inicial enviado pela fonte tende a ser facilmente escoado através do grafo em direção ao dreno. À medida que a densidade diminui, a quantidade de fluxo que não consegue atravessar o grafo, e retorna para fonte, aumenta, assim como o tempo de processamento. Quanto mais cedo se identifica a quantidade de fluxo que ficará retida no grafo, menor será custo em relação ao tempo de processamento.

Ao considerar a relação profundidade/largura notou-se que ao passo que a profundidade aumenta e a largura diminui, o tempo de processamento cresce, pois o número de vértices pelos quais o fluxo passa até chegar ao dreno também cresce. A variação do valor da capacidade máxima não influênciou o desempenho dos algoritmos nas experiências realizadas.

A implementação dos algoritmos de cálculo do fluxo máximo em um computador paralelo proporcionou um ganho significativo em termos de velocidade de processamento. O *speedup* obtido para os sistemas baseados no algoritmo de AWERBUCH e na verão síncrona de GOLDBERG & TARJAN foi bastante satisfatório, alcançando uma eficiência na faixa de 90% para a configuração com 8 processadores. Os testes realizados para avaliar o *speedup* mostraram que para grafos pequenos o custo de executar a computação em paralelo é muito alto, uma vez que o tempo dispendido com o controle do processamento se torna grande em relação ao tempo gasto com o cálculo do fluxo máximo propriamente dito. À medida que

o tamanho do grafo aumenta, o tempo gasto com cálculos também aumenta e se torna vantajoso realizar o processamento em paralelo.

A pouca quantidade de memória disponível (2 Mbytes por processador) limitou consideravelmente o tamanho dos grafos testados, pois o processamento de cada vértice consome uma quantidade de memória proporcional ao quadrado do número de vértices total máximo previsto pelos sistemas. Desta maneira, um pequeno aumento no número de vértices de um grafo representa um significativo aumento na utilização da memória.

Tendo em vista os resultados obtidos pode-se concluir que o objetivo do trabalho foi plenamente atingido, pois consegui-se realizar um levantamento do comportamento dos algoritmos de cálculo do fluxo máximo para uma grande variedade de tipos de grafos, e avaliar as vantagens e desvantagens de realizar o processamento em paralelo. A implementação dos sistemas de cálculo do fluxo máximo em um sistema distribuído permitiu o estudo de problemas típicos, como por exemplo a prevenção de *deadlocks*, e motivou o desenvolvimento de um modelo simplificado de sincronizador. O processamento paralelo mostrou-se um meio eficaz de se obter capacidade computacional e velocidade de processamento, apesar das dificuldades de programação e da falta de ferramentas de depuração adequadas.

Referências Bibliográficas

[1] AHUJA, R.K. e ORLIN, J.B. *A fast and simple algorithm for the maximum flow problem.*, Tech. Rep. 1905-87, Sloan School of Manegement, Massachusetts Institute of Technology, Cambridge, Mass., 1987.

[2] AMORIM, C.L., BARBOSA, V.C., FERNANDES, E.S.T., *Uma Introdução à Computação Paralela e Distribuída*, VI Escola de Computação, Campinas (1988).

[3] AMORIM, C.L., CITRO, R., SOUZA, A.F. e CHAVES FILHO, E. M., *The NCP-1 parallel computer*, Thechnical Report ES-241/91, COPPE/UFRJ (1991).

[4] AWERBUCH, B. *Reducing complexities of the distributed max-flow and breadth-first-search algorithms by means of network sinchronization*, Networks 15 (1985), 425-437.

[5] AWERBUCH, B. *Complexity of network sinchronization*, J. ACM 32, (Oct. 1985), 804-823.

[6] BARBOSA, V.C., DRUMMOND, L.M. e HELLMUTH, A.L.H. *From distributed algorithms to Occam programs by successive refinements*, Thechinal Report ES-237/91, COPPE/UFRJ (1991).

[7] BURNS, A., 'Programming in Occam 2', Addison-Wesley Plubishing Company (1988).

[8] DANTZIG, G.B. e FULKERSON D.R., *On the Max-Flow Min-Cut Theorem of Networks In Linear Inequalities and Related Systems*, Annals of Mathematics Study 38, Princeton University Press, (1956) 215-221.

[9] DIJKSTRA, E. W., FEIJEN, W.H.J. e GASTEREN, A.J.M. *Derivation of a Termination Detection algorithm for Distributed Computations.* Information Processing Letters 16 (1983), 217-219.

[10] DIJKSTRA, E. W., *Guarded Commands, Non-Determinacy and Formal Derivation of Programs*, Communications of ACM, Vol. 18, Number 8 (1975), 453-457.

[11] DIMACS (Center for Discrete Mathematics and Theorical Computer Science), *The first DIMACS international algorithm implementation challenge: general information; problem definitions and specifications; the core experiments*, DIMACS, Rutgers University, Piscataway, NJ (1990).

[12] DINIC, E.A. *Algorithm for solution of a problem of maximum flow in networks with power estimation*, Sov. Math. Dokl. 11(1970), 1277-1280.

[13] EDMONDS, J. e KARP, R.M. *Theorical improvements in algorithmic efficiency for network flow problems* J. ACM (1972), 248-264.

[14] EVEN, S. *Graph Algorithms*, Computer Science Press., Potomac, Md., 1979.

[15] FORD, L.R.Jr., e FULKERSON, D.R. *Flows in Networks*, Princeton University Press, Princeton, N.J., 1962.

[16] FULKERSON D.R. e DANTZIG G.B., *Computations of Maximum Flow in Networks*, Naval Res. Log. Quart. 2 (1955), 277-283.

[17] GOLDBERG, V.A. e TARJAN, R.E., *A new approach to the maximum-flow problem*, J. ACM 35 (1988), 921-940.

[18] GABOW, H.N.,*Scaling algorithms for network problems*, J. Comput. Syst. Sci. 31 (1985), 148-168.

[19] INMOS Limited, 'TDS - Transputer Development System', Prentice Hall (1988).

[20] INMOS Limited, 'Transputer Reference Manual', Prentice Hall (1985).

[21] KARZANOV, A.V. *Determinig the maximal flow in a network by method of preflows*, Sov. Math. Dokl 15, (1974), 434-437.

[22] LAWLER, E.L. *Combinational Optmization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[23] MAEKAWA, J.C., OLDEHOEFT, A.E., OLDEHOEFT, R.R., 'Operating Systems - Advanced Concepts', The Benjamin/Cummings Publishing Company (1987).

[24] MALHOTRA, V.M., PRAMODH KUMAR, M. e MAHESHWARI, S.N., *An $O(|V|^3)$ Algorithm for Finding Maximum Flows in Networks*, Inform. Process. Lett. 7 (1978), 277-278.

[25] MARBERG, J.M., GAFNI, E., *An $O(n^2m^{1/2})$ Distributed Max-Flow Algorithm*, Internacional Conference on Parallel Processing (1987).

[26] PAPADIMITRIOU, C.H., e STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

[27] PETERSON, J.L., SILBERSCHATZ, A., 'Operating Systems Concepts', Addison-Wesley Publishing Company, 1988.

[28] SEGALL A., *Distributed Network Protocols*, IEEE Trans. on Info. Theory IT-29, (1983).

[29] SHILOACH Y. e VISHKIN U. *An $O(n_2 logn)$ Parallel Max-Flow Algorithm*, J. Algorithms 3, (1982), 128-146.

[30] SZWARCFITER J., 'Grafos e Algoritmos Computacionais', Editora Campus, Rio de Janeiro, R.J., 1984.