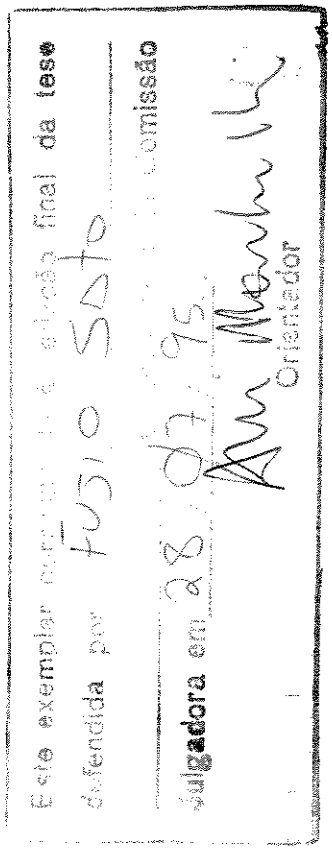
UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica
Departamento de Sistemas de Energia Elétrica

Um Estudo Comparativo da Análise de Curto-circuito Probabilístico em Ambientes Paralelo e Distribuído



Fujio Sato

Prof. Dr. Alcir José Monticelli

Orientador

Prof. Dr. Ariovaldo Verândio Garcia

Co-orientador

Tese apresentada à Faculdade de Engenharia Elétrica da UNICAMP, como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Elétrica.

Campinas, julho de 1995

Para

Kayoko,
Eduardo,
Fábio e
Gustavo.

AGRADECIMENTOS

A todos que contribuiram para a realização deste trabalho e em especial

ao Professor Alcir José Monticelli pelo incentivo, paciência, amizade e inestimável orientação,

ao Professor Ariovaldo Verândio Garcia pela amizade, preciosa co-orientação e todo o apoio na utilização dos recursos computacionais,

ao Departamento de Sistemas de Energia Elétrica - DSEE, por oferecer toda a infraestrutura do Laboratório de Sistemas de Energia Elétrica - LSEE,

ao Professor Antonio Cesar Baleeiro Alves pela inestimável colaboração na implementação das rotinas paralelas,

aos Professores André Morelato França, Carlos Alberto F. Murari e Carlos Alberto Castro Jr., pela colaboração,

à Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP, pelo suporte,

à Companhia Paulista de Força e Luz - CPFL e aos Gerentes dos órgãos do Departamento de Operação do Sistema - OS, nas pessoas dos Engos. Miguel Nucci, Rolando Antonio Thimmig, José Antonio Rennó Bittencourt e Celso Telles Penna Bastos, pelo apoio,

aos amigos do LSEE, pelo apoio e amizade, principalmente ao Marcelo Stehling de Castro e ao Eduardo Nicola F. Zagari, pela colaboração,

aos amigos do Setor de Estudos e Análise da Proteção - OSPP da CPFL, pelo apoio,

à Ana A.G. LLagostera pela orientação na organização das referências bibliográficas,

desejo expressar os meus sinceros agradecimentos.

Resumo

Este trabalho apresenta a paralelização de um programa de análise de curto-circuito probabilístico utilizando o método de *Monte Carlo* para sistemas de potência. O programa, originariamente desenvolvido e implementado em computadores seqüenciais, foi codificado para dois ambientes distintos de alto desempenho (paralelo e distribuído), tendo como um dos objetivo a verificação de alguns itens importantes concernentes ao processamento paralelo, tais como: portabilidade, desempenho, escalabilidade e comunicação.

As implementações paralela e distribuída desta aplicação foram feitas com dois modelos de programação concorrente: o *SPMD* (*Single Process Multiple Data*) e o Mestre/Escravo.

Os resultados foram obtidos através de testes em quatro sistemas elétricos da região Sul-Sudeste do Sistema Interligado brasileiro.

Abstract

This work presents the parallelization of a power system probabilistic short-circuit analysis program using *Monte Carlo* method. A sequential version of the code, originally developed for one-processor machine, was extended to two different high performance computer system architectures (parallel and distributed). The main objective of the research was to study issues such as portability, performance, scalability and communication.

Two programming models have been implemented on both architectures: *SPMD*, (*Single Process Multiple Data*) and *Master/Slave.*

The architectures and the models have been evalueted by simulation on four real-life networks of the brazilian South-Southeast interconnected system.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A necessidade de solucionar problemas com algoritmos que exigem computação intensiva tem levado ao desenvolvimento de sistemas computacionais de alto desempenho, genericamente conhecidos como sistemas paralelo e distribuído.

O termo **sistema distribuído** tem sido usado para definir uma larga faixa de sistemas computacionais, deste sistemas ligados em WAN[1] até aqueles ligados em LAN[2], e também sistemas paralelos com memória distribuída. O que eles têm em comum são as múltiplas unidades de processamento, cada um desempenhando a sua própria tarefa e ao mesmo tempo cooperando com outras unidades através de um sistema de comunicação.

Os computadores ligados em redes são logicamente integrados por um sistema operacional distribuído. Eles podem ser ligados de várias maneiras: desde redes geograficamente dispersas até estruturas de interligações com arquiteturas específicas. A Figura 1.1 mostra a estrutura básica de um sistema distribuído.

Um **sistema paralelo** é um tipo de sistema distribuído no qual as unidades de processamento estão fisicamente juntas e freqüentemente são integradas por um único sistema operacional.

A topologia de um sistema distribuído pode se alterar enquanto ele está em operação, devido a falhas ou reparos de *links* de comunicação, ou ainda quando se adicionam ou removem unidades de processamento. Isto reduz a confiabilidade dos *links* de comunicação e faz com que o *overhead* de comunicação seja imprevisível. Em comparação, a comu-

[1] *Wide Area Network*
[2] *Local Area Network*

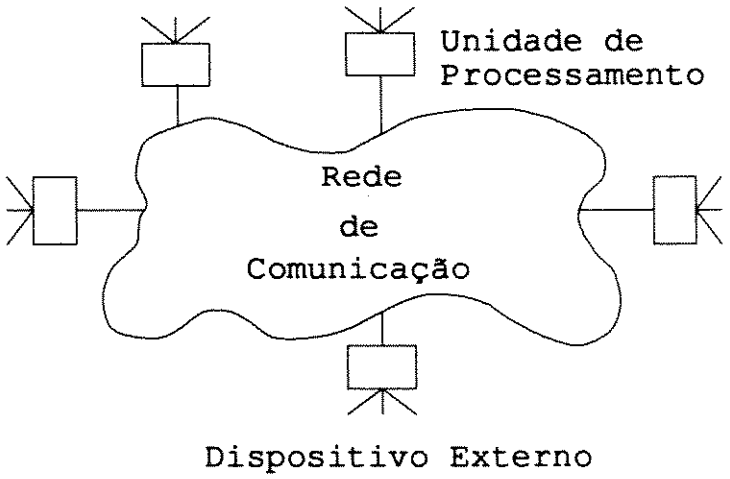


Figura 1.1: Sistema distribuído

nicação entre processadores em sistemas paralelos de memória distribuída é confiável e previsível.

Sistemas paralelos tanto podem ser de memória compartilhada como distribuída. Uma máquina com memória compartilhada tipicamente coordena a comunicação entre processadores através de uma memória global que todos os processadores compartilham. Uma máquina de memória distribuída combina uma memória local e um processador em cada nó da rede; os processadores se comunicam por troca de mensagens [48]. As Figuras 1.2 e 1.3 mostram respectivamente modelos genéricos para arquiteturas paralelas de memória compartilhada e de memória distribuída.

Neste trabalho o processamento paralelo foi utilizado para cálculos de curto-circuito probabilístico, talvez um dos problemas mais paralelizáveis no âmbito das análises do sistema de potência. O programa de curto-circuito probabilístico, originariamente desenvolvido e implementado em computadores seqüenciais (*IBM* e estação *Sun*), foi transportado para dois ambientes distintos (paralelo e distribuído), tendo como objetivos principais a verificação de alguns itens importantes concernentes ao processamento concorrente, tais como: portabilidade, desempenho, escalabilidade e comunicação.

Neste trabalho foram utilizados os seguintes ambientes computacionais:

1. Computador paralelo *nCUBE2* com 64 nós, conectado à uma rede local através de uma estação *Sun Sparc*1

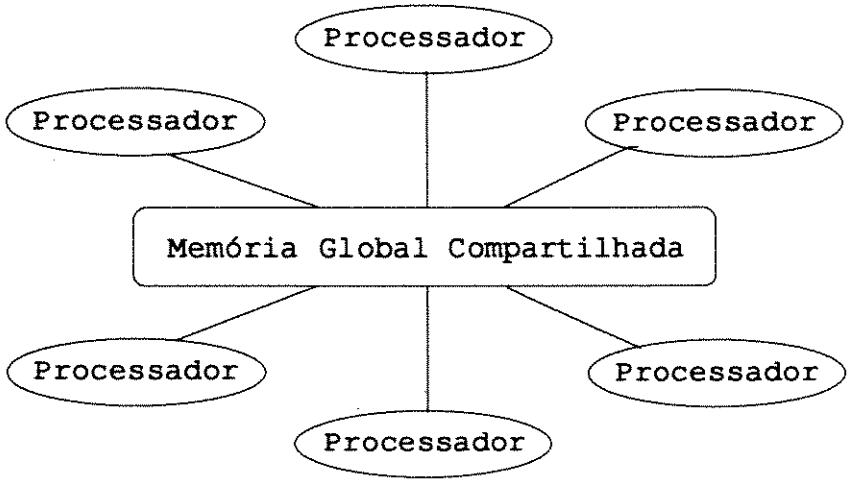


Figura 1.2: Memória compartilhada

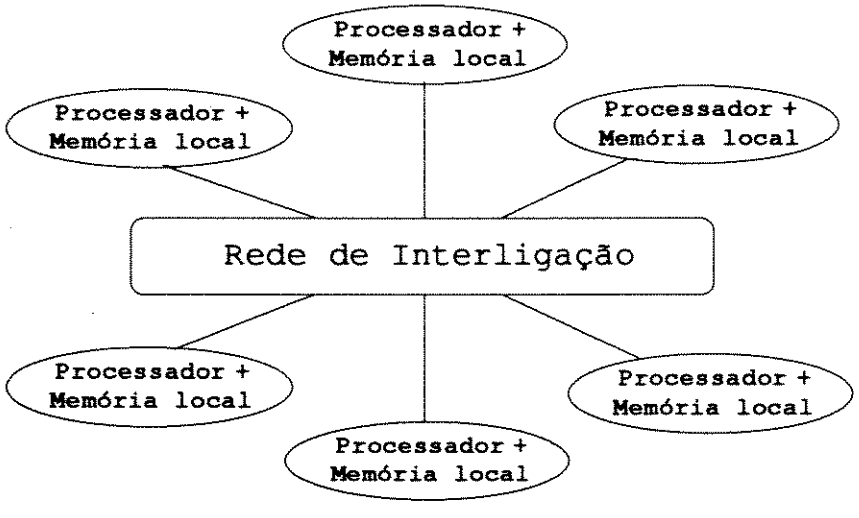


Figura 1.3: Memória distribuída

2. Rede de 11 estações *Sun* heterogênea (*Sparc*1 e *Sparc*2) conectadas via *Ethernet*
3. Rede de 09 estações *Sun* heterogênea (*Sparc*10 e *Classic*) conectadas via $FDDI$[3]
4. Computador paralelo $SP1$ com 08 nós conectados via $FDDI$
5. Rede de 08 estações $IBM\ RISC6000$ conectadas via *Ethernet*

Excluindo-se o computador paralelo $nCUBE$, os demais ambientes foram paralelizados com PVM[4] (versão 3.3), um software que configura as estações como uma máquina paralela virtual.

Em todos os ambientes foram implementados dois modelos de programação (ou paradigmas): um modelo centralizado, também chamado de Mestre/Escravo, e outro modelo descentralizado, baseado na programação $SPMD$ (*Single Program, Multiple Data*). Para cada um dos paradigmas o mesmo programa foi mantido em ambas as arquiteturas (computador paralelo e máquina paralela virtual), sendo alteradas somente as partes relativas às chamadas das primitivas.

Cada programa gera estimativas precisas das curvas de distribuição de probabilidade (função densidade) as quais podem ser usadas em uma variedade de estudos de projeto e planejamento, além de ter elevado potencial para utilização nos futuros Centros de Controle.

A obtenção de curvas de distribuição de probabilidade das correntes de curtos-circuitos, com precisão adequada, requer milhares de simulações. Mesmo quando se dispõem das facilidades oferecidas pelos sistemas computacionais atuais é imprescindível o uso apropriado de modelos e técnicas eficientes para se obter resultados desejados de forma otimizada. Com este fim, na programação do cálculo de curto-circuito probabilístico pelo método de *Monte Carlo* foram incorporadas três técnicas: (1) matrizes e vetores esparsos; (2) teorema da compensação para cálculos de curtos-circuitos ao longo da linha de transmissão e (3) equivalentes reduzidos de redes.

A Figura 1.4 mostra um fluxograma básico do programa de curto-circuito probabilístico.

O programa consiste basicamente de quatro etapas:

Etapa 1: Acessa o disco para ler os dados, portanto, depende do tamanho do sistema elétrico.

[3] ***Fiber Distributed Data Interface***
[4] ***Parallel Virtual Machine***

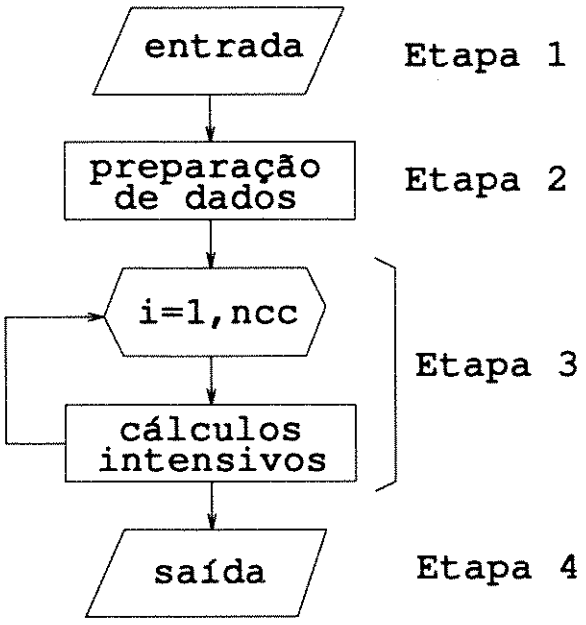


Figura 1.4: Fluxograma básico

Etapa 2: Prepara os dados (fatoração das matrizes das redes elétricas, inclusão dos acoplamentos mútuos, determinação das impedâncias de curtos-circuitos através da técnica de vetores esparsos). A etapa 2 é a parte seqüencial (PS).

Etapa 3: Executa a parte da computação intensiva, onde são simulados milhares de curtos-circuitos. A etapa 3 é a parte paralelizável (PP). *ncc* é o número de simulações de curtos-circuitos.

Etapa 4: Prepara os arquivos de saída.

Na Etapa 3 (parte paralelizável) são simulados milhares de curtos-circuitos (neste trabalho os resultados são apresentados para 100 000 simulações de curtos-circuitos), obtidos através do método de *Monte Carlo*. Este método baseia-se num processo de geração de números pseudo-aleatórios a partir de uma distribuição uniforme, que é utilizada para obter a função densidade de probabilidade das correntes de curtos-circuitos de uma determinada área de interesse do sistema elétrico de potência.

Para verificar o desempenho dos paradigmas de programação nas duas arquiteturas, as implementações foram testadas em várias redes elétricas que são partes do Sistema Interligado da região Sul-Sudeste do Brasil, com as seguintes dimensões: sistema completo,

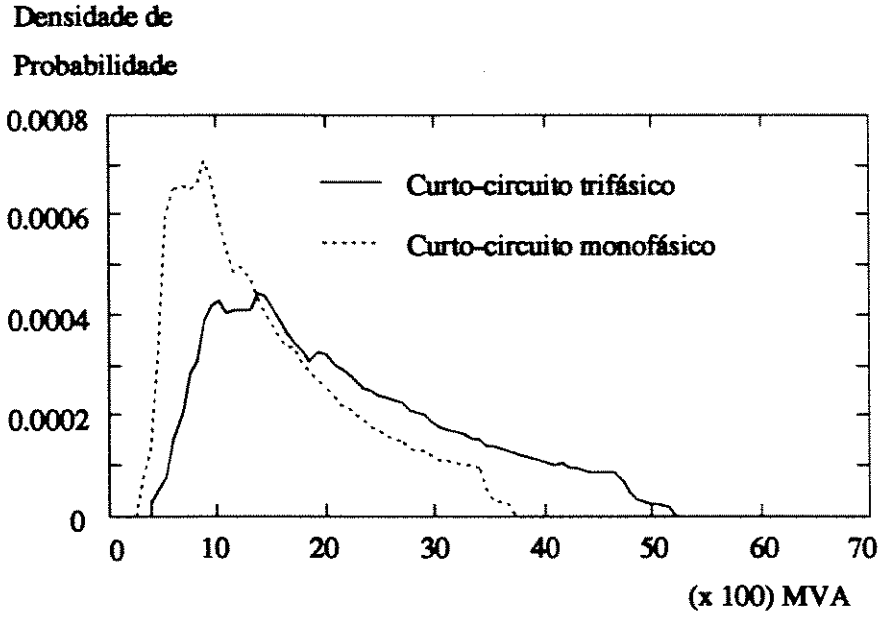


Figura 1.5: Distribuição de probabilidade típica das potências de curtos-circuitos

de 1176 barras, 2250 ramos e 87 mútuas (Sistema A) e sistema reduzido de 147 barras, 331 ramos e 37 mútuas (Sistema B - correspondendo ao Sistema CPFL[5], com a maior parte representada por equivalentes externos), sistema reduzido de 39 barras, 86 ramos e 37 mútuas (Sistema C - correspondendo à região de Campinas do Sistema CPFL , obtido por equivalentes externos a partir do Sistema A), sistema reduzido de 22 barras, 59 ramos e 130 mútuas (Sistema D - correspondendo ao Sistema *Light*[6]).

A Figura 1.5 mostra os histogramas de curtos-circuitos típicos (função densidade de probabilidade das potências de curtos-circuitos) preparados a partir das saídas do programa. Vale lembrar que o mesmo tipo de histograma pode ser obtido por qualquer um dos Sistemas A, B ou C.

A Figura 1.6 mostra a influência dos acoplamentos mútuos de seqüência zero na potência de curto-circuito monofásico. Os histogramas representam a função densidade de probabilidade de curto-circuito monofásico em torno da barra C, ilustrado na Figura 1.7. Para verificar esta influência foram simuladas duas situações: a primeira, considerando-se todos os acoplamentos mútuos (Sistema D) e a segunda, desprezando-se todos os acopla-

[5]Companhia Paulista de Força e Luz
[6]*Light* - Serviços de Eletricidade

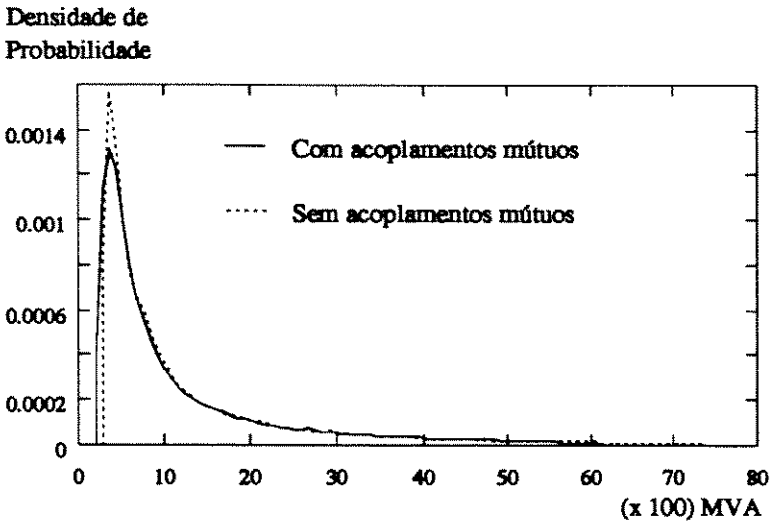


Figura 1.6: Efeito dos acoplamentos mútuos na intensidade do curto-circuito

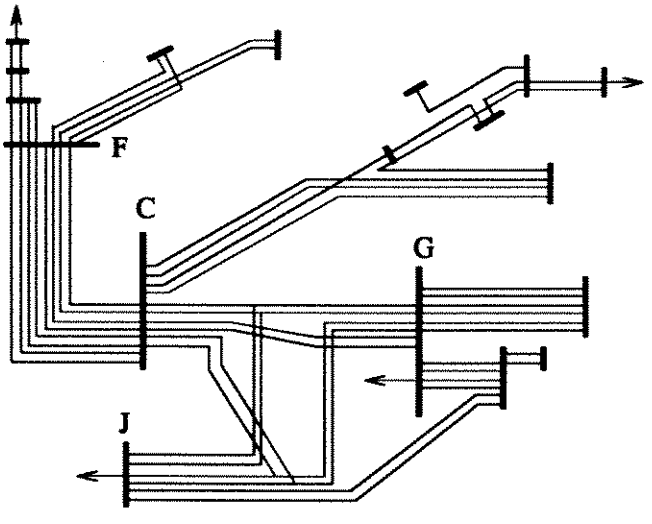


Figura 1.7: Diagrama Unifilar do Sistema D (*Light*)

mentos mútuos. Ressalte-se que o Sistema D é o que possui a maior densidade de linhas mutuamente acopladas no Sistema Interligado brasileiro.

O trabalho está organizado na seguinte forma: no Capítulo 2 são apresentados alguns conceitos de processamento paralelo; no Capítulo 3 discute-se a modelagem do sistema de potência, apresentando métodos importantes para análise de curtos-circuitos, tais como, matrizes/vetores esparsos, inclusão de acoplamentos mútuos, compensação (curtos-circuitos em linhas de transmissão) e redução da rede por equivalentes externos; no Capítulo 4 são introduzidos os conceitos principais de curto-circuito probabilístico, onde são discutidas a simulação de *Monte Carlo* e algumas análises de casos; no Capítulo 5 são apresentados as arquiteturas paralelas e os modelos de programação; no Capítulo 6 são mostrados os resultados dos testes em vários ambientes paralelos, utilizando dados dos Sistema Interligado da região Sul-Sudeste do Brasil; no Capítulo 7 estão as considerações finais e sugestões para o prosseguimento do trabalho.

Capítulo 2

Processamento Paralelo

2.1 Introdução

O *Tempo_de_Solucão* de um problema num computador pode ser avaliado pela seguinte fórmula:

$$Tempo\_de\_Solucão = \frac{Complexidade}{Taxa\_de\_Execucão} \tag{2.1}$$

A *Complexidade* é medida pelo número de operações necessários para resolver o problema. A *Taxa_de_Execucão* representa o número de operações que o computador realiza por segundo. Para ilustrar, considere-se a necessidade de processar 39 bilhões de bytes para solucionar um determinado problema. A Tabela 2.1 mostra o *Tempo_de_Solucão* utilizando três computadores com capacidades de processamento diferentes.

Mflops	Tempo de Solução
0,48	22,60 h
7,40	1,46 h
275,00	2,36 min

Tabela 2.1: Mflops x Tempo de Solução

Os computadores que apresentam o *Tempo_de_Solucão* extremamente rápido são conhecidos como aqueles que possuem a arquitetura de **alto desempenho**. Basicamente o

desempenho de um computador relaciona-se diretamente com a sua $Taxa\_de\_Execucão$ de operações, isto é, quantas operações de ponto flutuante ele executa por segundo ($Mflops$). A expressão que se segue indica os fatores que aumentam o processamento de um computador:

$$Taxa\_de\_Execucão = \frac{Multiplicidade}{Ciclo}.Concorrencia \tag{2.2}$$

O *Ciclo* de um computador representa o tempo necessário para executar a menor fase do processamento de uma instrução. No caso de computadores de alto desempenho, o *Ciclo* é o tempo mínimo requerido para produzir um ou mais resultados de uma operação aritmética. A *Multiplicidade* refere-se ao número de resultados produzidos num *Ciclo* por um processador. O nível de *Concorrência* indica o número de processadores que compõe o sistema. Analisando a Equação (2.2), conclui-se que para aumentar o desempenho duas alternativas são possíveis:

a. reduzir o *Ciclo* da máquina.

b. aumentar o nível de *Concorrência.*

A primeira alternativa é extremamente onerosa, pois implica em grandes investimentos no desenvolvimento tecnológico de componentes eletrônicos. Grandes fabricantes mundiais como *IBM*, *Cray*, *Fujitsu*, *Hitachi* e *NEC*,têm optado em seguir este caminho.

A velocidade de operação dos componentes eletrônicos, que é determinada pela tecnologia utilizada, influencia decisivamente no desempenho do sistema computacional. A Tabela 2.2 mostra as características principais de modernos supercomputadores:

Computador	Mflops	Ciclo	Tecnologia
NEC SX-2	1300	6,0	ECL
Cray-2	1600	4,0	ECL
Cray-3	20000	2,0	GaAs
ETA GF10	10000	7,0	CMOS
IBM 3090-600VF	696	17,2	ECL-Bip

Tabela 2.2: Tecnologia vs. desempenho para Supercomputadores

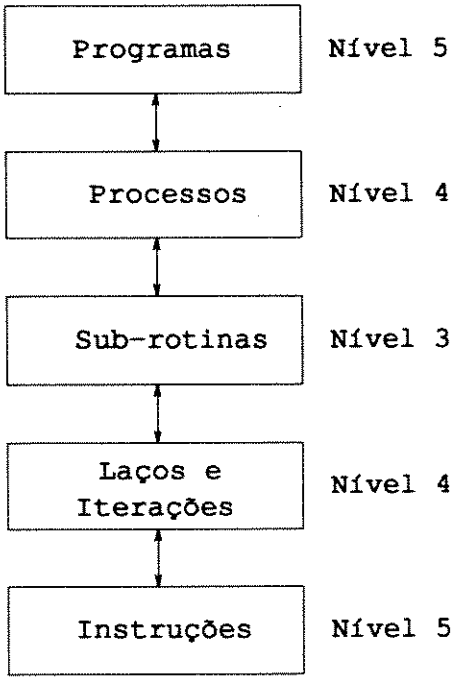


Figura 2.1: Níveis de paralelismo na execução do programa

A outra alternativa utiliza a tecnologia já desenvolvida. A melhoria do desempenho é conseguida combinando-se vários processadores existentes no mercado. É a alternativa escolhida pelos novos fabricantes de computadores de alto desempenho, tais como *nCUBE*, *Intel*, *Alliant*, *Floating Point Systems* e *Encore*.

A alta capacidade de processamento não depende apenas da velocidade dos componentes; o ganho extra fica a cargo da arquitetura do sistema e, em especial, do nível de concorrência ou paralelismo que a arquitetura oferece [45]. Paralelismo se refere ao processamento simultâneo de programas, processos, sub-rotinas, laços e iterações ou instruções, conforme ilustra a Figura 2.1. Dependendo em que nível o paralelismo é implementado tem-se os diferentes tipos de processamento.

É usual classificar os sistemas computacionais em termos do paralelismo utilizando a taxonomia de *Flynn*, que se baseia no fluxo de instruções e de dados simultâneos visto pelo processador durante a execução de um programa. *Flynn* propõe que as arquiteturas de computadores podem ser agrupadas em quatro categorias, entretanto, no contexto dos supercomputadores duas delas apresentam maior interesse:

- arquiteturas *SIMD* (*Single Instruction, Multiple Date streams*)
- arquiteturas *MIMD* (*Multiple Instruction, Multiple Date streams*)

Em sistemas *MIMD* vários processadores podem executar simultâneamente diferentes instruções com dados diferentes. Máquinas *MIMD* são computadores assíncronos que sincronizam os processos acessando dados na memória global compartilhada (ou transferindo mensagens em máquinas de memória distribuída). Sistemas *MIMD* se adaptam melhor para paralelismo de grão grosso (níveis de sub-rotinas e processos). Máquinas *SIMD* são computadores paralelos síncronos nos quais todos os processadores executam a mesma instrução ao mesmo tempo, ou então permanecem ociosos. Sistemas *SIMD* são tipicamente controlados por uma unidade de controle central que transmite uma única instrução para todos os processadores, os quais executam a instrução em fila, com seus dados locais. Paralelismo de grão fino (nível de instrução) são mais apropriados para tais sistemas.

2.2 Arquiteturas *SIMD*

Uma das arquiteturas típicas *SIMD* é mostrada na Figura 2.2. Emprega uma **Unidade de Controle (UC)**, **Unidades de Processamento (UPs)**, e uma **Rede de Interligação (RI)** tanto para comunicações processador-processador como para processador-memória. A **UC** envia uma única instrução para todos os **UPs**. A **RI** permite que os resultados dos cálculos da instrução sejam enviados a um outro processador para usá-los como operandos numa instrução subseqüente [42]. Apesar de os processamentos serem executados em paralelo, processadores individuais podem ser programados para ignorar determinadas instruções. Esta característica permite que a sincronização seja mantida através de vários controles, tais como cláusulas de uma declaração *if...then...else* [35]. Desde que essas máquinas são freqüentemente utilizadas para processamento de matrizes de grandes dimensões, elas são conhecidas como computadores vetoriais. Muitos computadores desta categoria são máquinas construídas para finalidades específicas, aplicadas principalmente para processamento de sinais e imagens [36].

2.3 Arquiteturas *MIMD*

As máquinas com arquitetura *MIMD* empregam múltiplos processadores que podem executar fluxos de instruções de forma independente com os dados armazenados local-

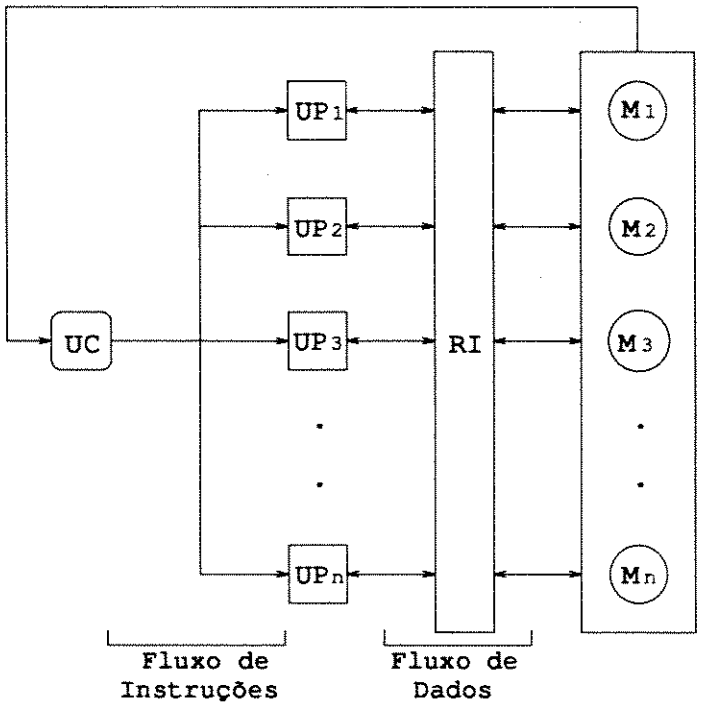


Figura 2.2: Arquitetura *SIMD*

mente favorecendo, assim, as soluções paralelas que requerem processadores que operem de uma maneira autônoma. As arquiteturas $MIMD$ são computadores assíncronos, caracterizadas pelo controle descentralizado [42]. Quando dois processadores trabalham em diferentes partes de um mesmo problema cada um deve receber os resultados. Assim, a comunicação é a parte crucial neste tipo de arquitetura [47].

Baseada na forma como os processadores se comunicam, os computadores $MIMD$ podem ser classificados em:

- **multiprocessadores** com memória compartilhada
- **multicomputadores** com memória distribuída

Nota: Conforme *Hwang* [36] o termo **multiprocessadores** é reservado para sistemas de múltiplos processadores com memória compartilhada e o termo **multicomputadores** para sistemas de múltiplos processadores com memórias distribuídas localmente.

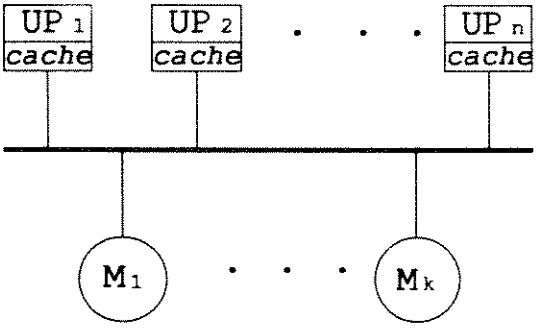


Figura 2.3: Arquitetura com interconexão por barramento

2.3.1 Multiprocessadores

Os computadores dotados de **multiprocessadores** com memória compartilhada são considerados fortemente acoplados e dependendo do tipo de interconexão dos processadores podem ser classificados em interconexão por barramentos e interconexão direta.

Na arquitetura com interconexão por barramentos, mostrada na Figura 2.3, todos os processadores têm acesso a uma memória comum através de um barramento de alta velocidade compartilhado. Se se considerar que cada processador deixa os resultados na memória e informa o seu endereço a outros processadores, a princípio a comunicação entre eles parece ser fácil, mas, na prática isto não é tão simples. Quando muitos processadores acessam a memória ao mesmo tempo, a disputa de barramento pode fazer com que todo o sistema se torne lento, com muitos processadores esperando por uma vaga no barramento deixada por um processador. Este problema é solucionado adicionando-se um *cache*, utilizado com memória local, mas, quando cada processador dispõe de seu próprio *cache*, seus resultados podem tornar inconsistentes. Os problemas de disputa do barramento e de inconsistência do *cache* limitam o número de processadores [47], acomodando cerca de 4 a 20 processadores [42].

Nas arquiteturas com conexão direta destaca-se aquela com interconexão por barramentos cruzados, que usa um *crossbusswitch* de n^2 pontos para ligar n processadores a n memórias, conforme mostra a Figura 2.4. Nesta configuração a disputa pelos *links* de comunicação é evitada em virtude da existência de um canal dedicado entre cada pares possíveis de processador/memória. Algumas características fazem com que o número de processadores seja limitado em 4 a 16 [42].

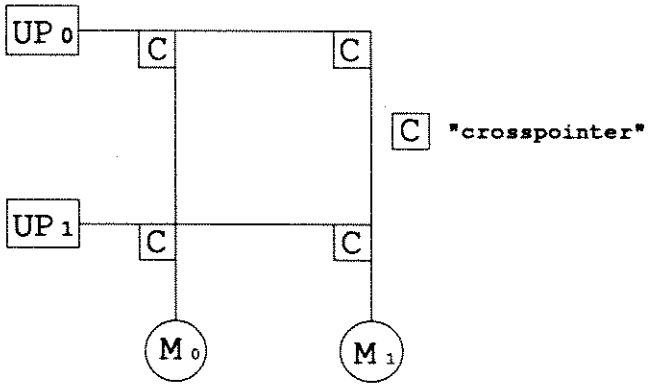


Figura 2.4: Arquitetura com interconexão por barramentos cruzados

2.3.2 Multicomputadores

Os computadores com arquitetura de memória distribuída são fracamente acoplados. Cada conjunto processador/memória, denominado de **nó**, pode ser considerado um computador completo, daí o termo **multicomputador** para definir este tipo de arquitetura [47]. Um produto das pesquisas dos anos '80, tem a característica de acomodar significativas adições de processadores [42]. Em virtude de outros nós não terem acesso à memória privada de um nó, os resultados devem ser trocados entre os nós através de uma rede de comunicação. Esta arquitetura é também conhecida como *message passing*. A configuração básica da arquitetura de memória distribuída é mostrada na Figura 2.5.

No **multicomputador** o par processador/memória é referido como **nó** e os elos de comunicação que ligam os **nós** como *links*. Os processadores se comunicam transferindo mensagens através desses *links*. Uma mensagem enviada de um **nó** para outro pode passar por **nós** intermediários. A lista ordenada de **nós**, por onde passam as mensagens, é um **caminho**. O **comprimento** de um **caminho** é o número de *links*. O maior **comprimento** entre dois **nós** é definido como **diâmetro** [38].

O desempenho de um processador é afetado tanto pela velocidade de comunicação como pela velocidade dos processadores. Talvez fosse melhor dizer que o balanço entre o tempo de comunicação e tempo de computação em um **multicomputador** varia de problema para problema, de acordo com o algoritmo utilizado e a topologia e desempenho da rede de comunicação. A implementação de mais nós nos computadores com estas arquiteturas não garante a melhoria no desempenho, pois, a máquina pode gastar quase todo o tempo na comunicação, enquanto muitos processadores permanecem ociosos [47].

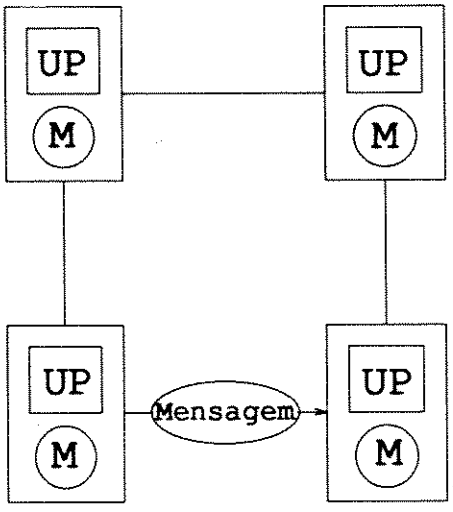


Figura 2.5: Estrutura básica de uma arquitetura de memória distribuída

Existem várias topologias de interconexão da rede que permitem a escalonabilidade da arquitetura e proporciona desempenho eficiente para programas paralelos com diferentes padrões de comunicação entre os nós. A Figura 2.6 representam as topologias dos **multicomputadores**.

- Topologia tipo anel (*ring*):

  Esta topologia é apropriada para poucos **nós**, executando algoritmos com baixa taxa de comunicação. Isto se explica pelo fato de num anel de n **nós** o **diâmetro** da comunicação ser $n/2$.

- Topologia tipo malha (*mesh*):

  Considerando-se n **nós** formando uma malha bi-dimensional, isto é, $n = p^2$, o *diâmetro* da comunicação será *2(p - 1)*. A pesquisa deste tipo de topologia tem sido estimulada pela existência de correspondência entre malhas e algoritmos orientados para matrizes.

- Topologia tipo árvore (*tree*):

  Na topologia tipo árvore o *diâmetro* da comunicação é *2(l -1)*, onde l é o nível de profundidade da árvore. Têm sido desenvolvidas para suportar algoritmos de busca e ordenação, processamento de imagens e outros [42]

- Topologia tipo hipercubo (*hypercube*):

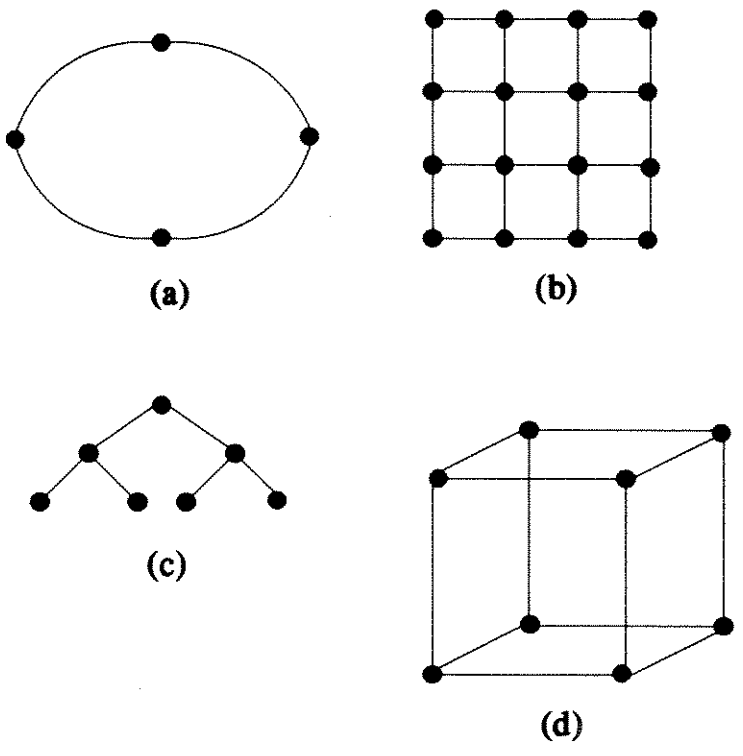


Figura 2.6: Topologias *MIMD*: (a) anel; (b) malha; (c) árvore; (d) hipercubo

Esta topologia utiliza $n = 2^d$ nós organizados em um cubo de d dimensões, onde cada nó tem $d = log_2 n$ ligações bidirecionais para nós adjacentes por um canal de comunicação ponto a ponto. Os nós de um hipercubo são identificados de 0 a $2^d - 1$ com valores numéricos de d bits. As conexões existentes são definidas pelo conjunto de arcos que podem ser traçados entre quaisquer dois nós cuja numeração difira em um bit. O **diâmetro de comunicação** do hipercubo é d.

2.4 Aspectos de Comunicação em Sistemas Paralelo e Distribuído

Em muitos algoritmos ou sistemas paralelo e distribuído o tempo gasto para comunicação entre procesadores é uma fração considerável do tempo total necessário para resolver um determinado problema. Neste caso diz-se que o algoritmo sofre uma substancial **Penalidade de Comunicação (PC)** ou atraso de comunicação. Então, pode-se definir:

$$PC = \frac{T_{total}}{T_{comp.}} \tag{2.3}$$

onde T_{total} é o tempo requerido pelo algoritmo para resolver um determinado problema e $T_{comp.}$ é o tempo que pode ser atribuído somente à computação, isto é, o tempo que deveria ser requerido se toda a comunicação fosse instantânea. Para analisar a questão da comunicação é útil visualizar o sistema de computação distribuído como uma rede de processadores ligados por *links* de comunicação. Cada processador usa sua própria memória local para armazenar alguns dados do problema e resultados intermediários, e troca de informações com outros processadores em grupo de *bits* chamados pacotes (*packets*) usando os *links* de comunicação da rede. Os comprimentos dos pacotes podem variar extensamente, desde poucas dezenas de *bits* até várias centenas de *bits*. Admite-se que quando um pacote percorre um *link* de comunicação, os *bits* do pacote são consecutivamente transmitidos sem interrupção. Uma memória compartilhada pode também ser vista como uma rede de comunicação, desde que cada processador possa enviar as informações para todos os outros processadores armazenando-as na memória compartilhada.

Os atrasos de comunicação podem ser divididos em quatro partes:

(a) **Tempo de processamento da comunicação**. É o tempo necessário para preparar a informação para a transmissão; por exemplo, montagem da informação em pacotes, endereçamento e controle de informação aos pacotes, seleção de um *link* para transmissão de cada pacote, movimentação dos pacotes para *buffers* apropriados, etc.

(b) **Tempo de enfileiramento**. Uma vez que a informação é montada em pacotes para a transmissão em algum *link* de comunicação, ela deve esperar em fila antes de ser disparada para transmissão, por algumas razões. Por exemplo, o *link* pode estar temporariamente indisponível, devido a sua utilização ou estar sendo decidida a sua alocação, em virtude dele estar sendo disputado por vários pacotes. Outra razão pode ser ocasionado pelo postergamento da transmissão de um pacote, ocasionado pela indisponibilidade momentânea de recursos de *hardware* (falta de espaço no *buffer*, por exemplo). No caso em que a possibilidade de erros na transmissão não é desprezível, o **tempo de enfileiramento** inclui o tempo para retransmissão do pacote, necessário para corrigir os erros. O **tempo de enfileiramento** geralmente é difícil se quantificado com precisão, mas modelos simplificados são freqüentemente adequados para obter conclusões quantitativa e qualitativa.

(c) **Tempo de transmissão**. É o tempo necessário para transmitir todos os *bits* do pacote.

(d) Tempo de propagação. É o tempo entre o fim da transmissão do último *bit* do pacote no processador transmissor e a recepção do último *bit* do pacote no processador receptor.

Dependendo do sistema e algoritmo, um ou mais tempos descritos anteriormente podem ser desprezíveis. Por exemplo, em alguns casos a informação é gerada com suficiente regularidade e os recursos de transmissão são suficientemente abundantes que nunca há a necessidade de **enfileiramento de pacotes**, enquanto que em outros casos a distância física entre o transmissor e o receptor é tão pequena que o **tempo de propagação** é desprezível.

Para muitos sistemas, pode-se assumir que o **tempo de processamento** e de **propagação** num dado *link* é constante para todos os pacotes, e o **tempo de transmissão** é proporcional ao número de *bits* (ou comprimento) do pacote. Assim, o atraso (A) que um pacote, ao ser transmitido num *link*, pode sofrer é dado pela Equação (2.4):

$$A = P + TL + F \tag{2.4}$$

onde P é o **tempo de processamento** e de **propagação**, T é o tempo requerido para a transmissão de um *bit*, L é o comprimento do pacote em *bits* e F é o **tempo de enfileiramento**.

Em alguns sistemas, o **tempo de transmissão** TL é muito maior do que o **tempo de processamento** e de **propagação** P, principalmente quando L inclui um número substancial de *bits* com *overhead*. Em outros casos pode ocorrer o inverso. Na grande maioria dos sistemas, mesmo quando o pacote não contém *overhead*, a soma $P + TL$ é muito maior do que tempo para executar uma operação elementar, tal como uma multiplicação em ponto flutuante. Isto significa que se o algoritmo paralelo requer transmissão de um pacote para todas as poucas operações que ele executa, o **tempo de comunicação** predomina sobre o **tempo de execução** [41].

Até aqui o pacote foi enfocado como uma unidade de comunicação, entretanto, algumas vezes ele pode ser parte de alguma mensagem que tem sentido quando recebido como um todo. Uma mensagem pode ser segmentada em vários pacotes para a transmissão por inúmeras razões, o que torna apropriado enfocar o atraso da mensagem total (do início da transmissão do primeiro pacote até o término da recepção do último pacote). Isto complica a situação porque o atraso da mensagem depende de como ela é segmentada em pacotes, e se a transmissão dos pacotes pode ser paralelizada. Por exemplo, se a mensagem é dividida em s pacotes de comprimentos iguais e transmitidos com atrasos

iguais, independente das rotas paralelas de comunicação, o atraso da mensagem será menor por um fator s comparando-se ao caso onde a mensagem inteira é transmitida por uma das rotas. (Esta contabilidade assume que o *overhead* extra de comunicação é desprezível quando a mensagem é segmentada em vários pacotes e que os atrasos de **processamento da comunicação**, **enfileiramento** e **propagação** também são desprezíveis.)

Outra possibilidade para a paralelização da comunicação surge quando uma mensagem é transmitida num caminho de $k > 1$ *links*. Se uma mensagem segmentada em s pacotes é transmitida seqüencialmente por um caminho de k *links* com um tempo de transmissão em cada *link* igual a T por pacote, o atraso da mensagem será de $(s + k - 1)T$, se comparado com o atraso de ksT caso a mensagem inteira for transmitida num único pacote. (Esta contabilidade desconsidera o efeito do *overhead*, atrasos de **processamento**, **propagação** e **enfileiramento**, e assume que um nó deve receber integralmente um pacote antes de transferir para qualquer outro nó). A redução do atraso é realizado " encanando " a transmissão dos pacotes nos k *links*, como mostra a Figura 2.7. Pode-se ver que segmentando a mensagem em pacotes muito pequenos, o retardo pode ser reduzido por um fator aproximadamente igual ao número de *links* do caminho, para um tempo quase próximo ao requerido para transmitir a mensagem por um único *link*.

Alguns fatores mais importantes que influenciam no retardo da comunicação são:

a. o algoritmo usado para controlar a comunicação, principalmente o erro de controle, roteamento e controle de fluxo;

b. a topologia da rede de comunicação, isto é, o número e a localização dos *links* de comunicação;

c. a estrutura do problema a ser resolvido e a concepção do algoritmo para "casar" esta estrutura, incluindo o grau de sincronização requerido pelo algoritmo.

2.4.1 *Links* de Comunicação

Independente do método pelo qual a mensagem é transmitida fisicamente num canal de comunicação o *link* de comunicação é visto como um **conduto virtual de** *bits*, pelo qual os *bit* fluem, partindo de uma origem (ou transmissor) e chegando a um destino (ou receptor). Existem vários tipos de **condutos**, tais como:

1. Conduto de *bit* síncrono: o transmissor envia continuamente *bits* a uma taxa fixa e mesmo quando não houver pacotes para envio são transmitidos *bits* artificiais.

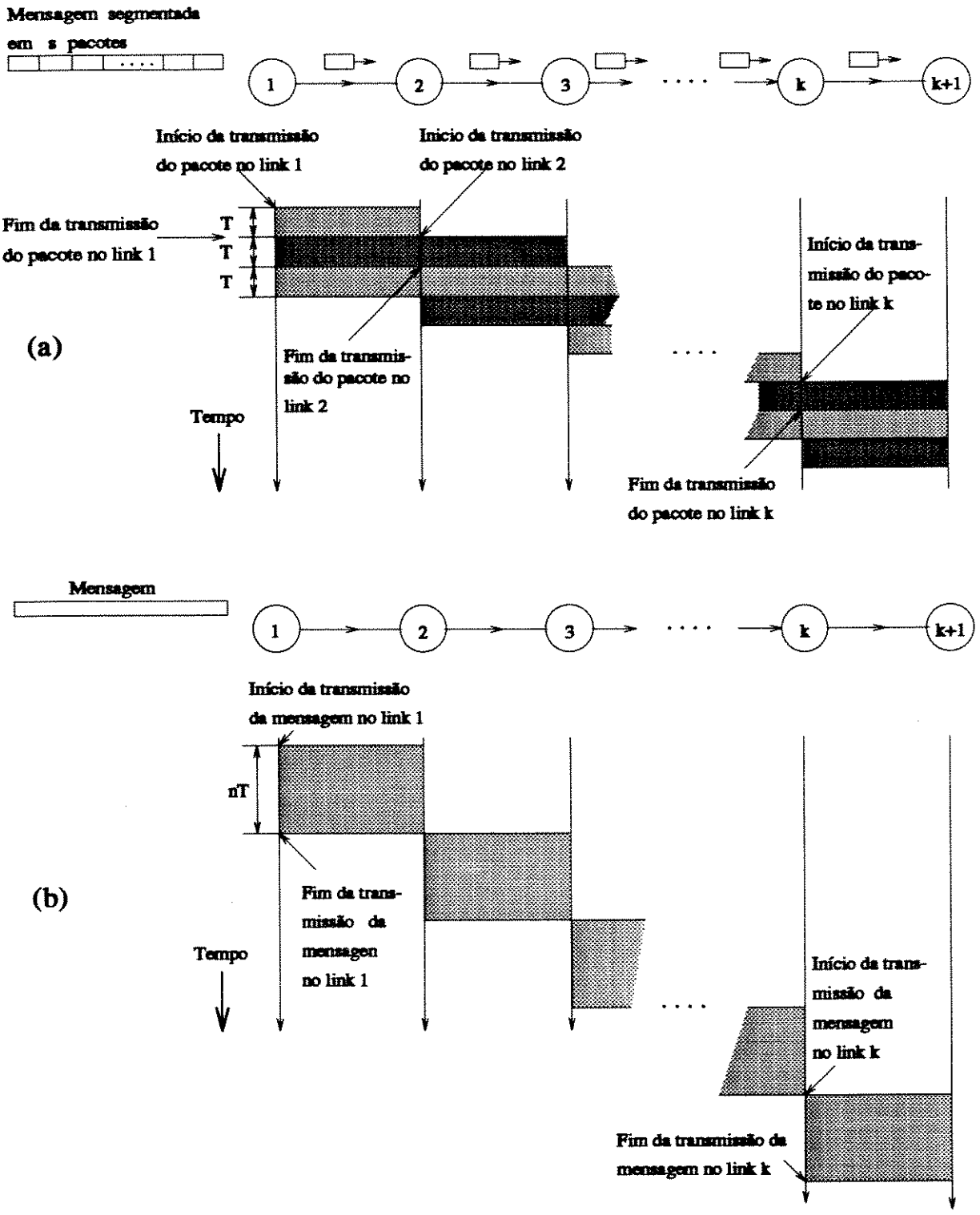


Figura 2.7: Transmissão de mensagem

Um exemplo é a conexão ponto à ponto de alta velocidade, que é muito comum em ambientes computacionais paralelo e distribuído.

2. Conduto de *bit* síncrono intermitente: o transmissor envia *bits* a uma taxa fixa quando existe pacotes para enviar, caso contrário nada envia. Um exemplo é o *Ethernet* que é uma *LAN*.

3. Conduto de caráter assíncrono: os pacotes são transmitidos em grupos de *bits* chamados caráteres (geralmente oito ou nove *bits*, alguns dos quais são usados para sincronização); os *bits* dentro de cada caráter são transmitidos a uma taxa fixa, mas caráteres sucessivos podem estar separados por retardos variáveis, sujeitos a um dado valor mínimo. Exemplos típicos envolve comunicação de baixa velocidade usando computadores pessoais, e/ou linhas discadas.

Em todos estes **condutos**, a representação física de *bits* pode ser tomados de diferentes formas, usando uma variedade de técnicas de modulação e codificação.

A capacidade de um *link* de comunicação é a máxima taxa no qual os *bits* podem ser transmitidos num **conduto** de *bit* correspondente, e é igual a $\frac{1}{T}$, onde T é o tempo de transmissão do *bit* usado na Equação (2.5)

No contexto de um dado algoritmo paralelo ou distribuído, a taxa de transmissão de *bits* é na realidade menor devido a alguns fatores:

a. cada pacote pode carregar *overhead* de *bits* usados para detectar o início e o fim do pacote e detectar erros de na transmissão; o efeito do *overhead* de *bits* pode ser geralmente computado em termos de P e L na Equação (2.5);

b. o *link* pode ser usado em parte para transmitir pacotes não relacionados no algoritmo distribuído. Por exemplo, alguns pacotes de controle para sustentar as estruturas da organização do sistema computacional necessitam ser transmitidos periodicamente;

c. em alguns *links*, *hardware* de comunicação pode ser compartilhado entre vários **condutos virtuais de** *bits* assim cada um desses **condutos** pode ser usado somente parte do tempo. Um exemplo típico é quando um processador pode, a qualquer momento, enviar pacotes pelo menos por um dos vários canais de comunicação incidentes. Outro exemplo ocorre nos casos de *links* de comunicação **multiacesso**, tais como *Ethernet* e outras redes locais, onde um canal físico é compartilhado entre vários **condutos virtuais de** *bits* na base da disputa. Quando o *hardware* de comunicação é compartilhado entre vários *links* de comunicação, o tempo de enfileiramento Q na Equação (2.5) não é desprezível, o que complica seriamente a análise da penalidade da comunicação;

d. algum recurso pode estar indisponível para o envio dos pacotes (por exemplo, *buffer* sem espaço). Nesta situação pacotes são forçados a esperar pela disponibilidade do recurso, mesmo que o *link* de comunicação esteja livre. Assim, a taxa de transmissão dos pacotes é reduzida para a taxa na qual o recurso se torna disponível. Algoritmos que efetuam este tipo de redução de taxa são chamados algoritmos de **controle de fluxo**;

e. em alguns *links* de comunicação, existe uma possibilidade, não desprezível, de que alguns *bits* sejam recebidos diferentemente de como eles foram transmitidos, ou perdidos totalmente. Em tais casos é necessário usar um esquema que detecte estas anomalias e retransmita os pacotes envolvidos tantas vezes quanto forem necessários para concluir uma recepção correta. Isso diminui a taxa de transmissão, e simultaneamente afeta o tempo de enfileiramento Q, desde que um pacote que é retransmitida pode ser conceitualmente considerado como um pacote na fila de espera para iniciar a sua primeira transmissão correta.

2.5 Medidas de Desempenho

Para a avaliação do processamento paralelo as seguintes medidas de desempenho são particularmente importantes: o *speedup*, a eficiência [41] e a eficácia [49].

Considerando n o número de processadores (ou de *nós*) e p o tamanho do problema, o *speedup* $S_n(p)$ descreve o ganho (em tempo) do programa paralelo executado com n processadores em relação ao melhor programa seqüencial, ambos executando o mesmo processo. Assim, a fórmula do *speedup* é dada pela Equação (2.5).

$$S_n(p) = \frac{T^*(p)}{T_n(p)} \tag{2.5}$$

O tempo global de execução é composto de três componentes, conforme mostra a Equação (2.6).

$$T_n(p) = T_{i/o} + T_{proc}(p) + T_{com}(p) \tag{2.6}$$

Usualmente, $T_{proc}(p)$ é uma função que decresce com o aumento de n, enquanto que $T_{com}(p)$ aumenta e $T_{i/o}$ depende do p (tamanho do problema).

A eficiência $E(p)$ é uma medida da fração do tempo que os processadores estão ocupados. Formalmente, $E(p)$ é a relação entre o *speedup* e o número de processadores.

$$E_n(p) = \frac{S_n(p)}{n} = \frac{T^*(p)}{nT_n(p)} \tag{2.7}$$

A eficácia $\eta_n(p)$ descreve o grau de utilização dos processadores e é definida pela Equação (2.8).

$$\eta_n(p) = \frac{S_n^2(p)}{n} \tag{2.8}$$

A eficácia aumenta até um valor máximo e depois decresce. O número de processadores que corresponde ao $\eta_n(p)$ máximo é definido como *pws* (*processor working set*), isto é, o número de processadores para o qual a relação entre o benefício (aumento no *speedup*) e o custo (decréscimo da eficiência) é máxima [49].

A eficiência seria igual a 1 se o *overhead* de explorar o paralelismo (leitura de dados, comunicação e chamadas adicionais de rotinas paralelas) fosse negligenciado e o trabalho fosse distribuído igualmente entre as UCPs, implicando conseqüentemente que nenhum processador ficaria ocioso durante o processamento. Esta situação evidentemente é ideal e, portanto, inatingível. Um objetivo mais realista deve almejar uma eficiência, quanto possível, longe de zero, conforme o crescimento de p e n. Assim, pode-se concluir que a eficiência $E_n(p)$ dá uma medida do grau de aproximação do ganho real obtido em relação ao *speedup* ideal.

Existe uma dificuldade com relação às definições acima em virtude de o tempo do programa seqüencial ótimo $T^*(p)$, em muitos casos, não ser conhecido. Por este motivo, $T^*(p)$ pode ser definido de uma forma diferente. Isto é, toma-se o $T_1(p)$ como o tempo dispendido por um único processador para executar um determinado algoritmo paralelo sob análise. Com esta nova definição de $T_1(p)$, a eficiência mostra o grau de paralelização alcançado num determinado algoritmo, mas, não fornece nenhuma informação sobre o mérito absoluto do algoritmo.

A questão fundamental é se o máximo *speedup* atingível $T_1(p)/T_n(p)$ pode ser feito arbitrariamente grande conforme aumenta o p e n tendendo para ∞. Existe uma considerável discussão sobre o valor do *speedup* que se consegue alcançar em situações reais. A principal dificuldade é que os programas possuem algumas partes que são facilmente paralelizáveis, mas também possuem outras que são inerentemente seqüenciais. Quando

se dispõe de um grande número de processadores, as partes paralelizáveis são rapidamente executadas, porém as partes seqüênciais são os gargalos do problema. Esta observação é conhecida como a lei de *Amdahl* e pode ser quantificada da seguinte forma: se o programa possui duas partes, uma que é inerentemente seqüencial e outra que é totalmente paralelizável, e se a parte seqüencial consome a fração f da computação total, então o *speedup* é limitado pela Equação (2.9)

$$S_n(p) \leq \frac{1}{f + \frac{(1-f)}{n}} \leq \frac{1}{f} \tag{2.9}$$

Capítulo 3

Modelagem do SEP para Análise de Curtos-circuitos

3.1 Introdução

Após a modelagem conveniente da rede de um sistema elétrico de potência (SEP), a simulação de curtos-circuitos consiste em resolver o sistema de equações algébricas lineares:

$$Y_{BARRA}\underline{E} = \underline{I} \tag{3.1}$$

sendo Y_{BARRA} - matriz de admitância nodal

$\underline{E}$ - vetor tensão nodal

$\underline{I}$ - vetor corrente nodal

A solução do sistema de Equações (3.1) é dada por:

$$Z_{BARRA}\underline{I} = \underline{E} \tag{3.2}$$

sendo Z_{BARRA} a matriz de impedância nodal.

Em termos computacionais a parte mais laboriosa está na obtenção da matriz de

pedância nodal (também conhecida como matriz de curto-circuito).

3.2 Métodos Matriciais

3.2.1 Método da Matriz Z_{BARRA}

O primeiro método eficiente surgiu no inicio da década de 60 [2] que consiste em formar a matriz passo a passo, simulando-se a própria construção da rede, a partir do nó de referência, acrescentando-se uma ligação por vez. Este método exige uma ordenação prévia das ligações da rede, para garantir a conexidade do sistema em qualquer estágio da formação da matriz e evitar a singularidade das matrizes parciais que aparecem durante o processo de formação [16].

A matriz Z_{BARRA} é cheia e esta característica traz limitação no armazenamento para grandes sistemas [5].

3.2.2 Métodos das Matrizes Esparsas

Em sistemas de potência certas matrizes dos coeficientes são esparsas. Entre elas esta matriz de admitância nodal Y_{BARRA}.

Os métodos das matrizes esparsas tiram vantagens da **esparsidade** da matriz de admitância nodal Y_{BARRA}, nos quais se utiliza a técnica de fatoração triangular de uma matriz por eliminação de Gauss [4].

A matriz Y_{BARRA} pode ser colocada na forma de produto de matrizes esparsas

$$Y_{BARRA} = LDU \tag{3.3}$$

em que os elementos das matrizes L, D e U são obtidos durante o processo de eliminação de Gauss [25]. Uma das maneiras de se realizar a fatoração da matriz Y_{BARRA} é o método de **bifatoração**, no qual se faz a eliminação de Gauss simultaneamente nos triângulos inferior e superior da matriz. No caso das matrizes simétricas fatoração e **bifatoração** constituem praticamente o mesmo método. Nas matrizes da forma fatorada aparecem elementos cujas posições na matriz Y_{BARRA} eram elementos nulos, causando com isso a redução da **esparsidade** dessas matrizes. Como a eficiência computacional depende

da **esparsidade** da matrizes dos fatores é importante minimizar o aparecimento desses elementos, que é conseguida através de um processo de renumeração dos nós da rede, que corresponde a uma reordenação do **pivoteamento**. Existem algumas maneiras de se efetuar a ordenação, como: ordenação tipo banda, ordenação segundo o grau mínimo e a ordenação segundo o grau mínimo dinâmico.

. Método da Bifatoração [7]

Este método consiste em se efetuar a eliminação de Gauss eliminando-se simultaneamente uma linha e uma coluna da matriz dos coeficientes, operando com matrizes dos tipos dado a seguir:

Seja Y_{BARRA} a matriz de admitância nodal, as operações com as matrizes L_j e U_j que a torna numa matriz identidade são:

$$\begin{aligned} Y_1 &= L_1 Y U_1 \\ Y_2 &= L_2 Y_1 U_2 \\ Y_3 &= L_3 Y_2 U_3 \\ . &= ..... \\ I &= L_n Y_{n-1} U_n \end{aligned} \tag{3.4}$$

que formalmente podem ser descrita pela expressão:

$$L_n L_{(n-1)} \cdots L_2 L_1 Y U_1 U_2 \cdots U_{(n-1)} = I \tag{3.5}$$

A Equação (3.5) pode ser transformada em:

$$Y^{-1} = U_1 U_2 \cdots U_{(n-1)} L_n L_{(n-1)} \cdots L_2 L_1 \tag{3.6}$$

. Método da Matriz Y_{BARRA} - fatorada

Neste método, obtém-se todos os elementos de uma determinada coluna k da matriz Z_{BARRA} pós-multiplicando-se os fatores esparsos da Equação (3.6) por um vetor coluna em que todos os elementos são nulos, exceto *k-ésimo* elemento que é 1,0. As operações indicadas na Equação (3.6) são executadas uma por vez sobre o vetor independente e correspondem as operações contidas nas matrizes elementares L_j e U_j.

. Método da Matriz Z_{BARRA} - esparsa

De acordo com a Equação (3.6), a matriz Z_{BARRA} pode ser obtida por (2n - 1) multiplicações sucessivas, em que são calculadas elementos cujas posições correspondentes na matriz Y_{BARRA} são elementos não nulos, além de alguns outros elementos necessários nas fases intermediárias do cálculo ($fill-in$). A obtenção da matriz Z_{BARRA} esparsa pode ser formalizada como se segue [12], [13] e [17].

Tendo-se:

$$Y_{BARRA}Z_{BARRA} = I \tag{3.7}$$

substituindo-se Y_{BARRA} pela sua forma fatorada tem-se:

$$LDUZ_{BARRA} = I \tag{3.8}$$

como Y_{BARRA} é uma matriz simétrica

$$U = L^t \tag{3.9}$$

portanto,

$$LDL^tZ_{BARRA} = I \tag{3.10}$$

Mediante operações algébricas na Equação (3.10) segue-se:

$$\begin{aligned} DL^tZ_{BARRA} &= L^{-1} \\ L^tZ_{BARRA} &= D^{-1}L^{-1} \\ 0 &= D^{-1}L^{-1} - L^tZ_{BARRA} \\ Z_{BARRA} &= D^{-1}L^{-1} + Z_{BARRA} - L^tZ_{BARRA} \\ Z_{BARRA} &= D^{-1}L^{-1} + (I - L^t)Z_{BARRA} \end{aligned} \tag{3.11}$$

e definindo,

$$(I - L^t) = T \tag{3.12}$$

finalmente,

$$Z_{BARRA} = D^{-1}L^{-1} + TZ_{BARRA} \tag{3.13}$$

As seguintes observações devem ser feitas:

1. L é uma matriz triangular inferior com diagonal unitária. A sua inversa L^{-1} mantém a mesma estrutura. Portanto, a diagonal da matriz D^{-1} L^{-1} é igual à da matriz D^{-1}.

2. Sendo Z_{BARRA} uma matriz simétrica é suficiente calcular só os elementos do triângulo superior da matriz Z_{BARRA} - esparsa.

   Baseando-se nestas observações conclui-se que a matriz L^{-1} não é necessária nos cálculos da matriz Z_{BARRA} - esparsa.

Portanto,

$$Z_{esp} = D^{-1} + TZ_{BARRA} \tag{3.14}$$

Os elementos da matriz podem ser calculados sucessivamente a partir da última linha, obtendo- se somente os elementos cujas posições na matriz L^{-1} são ocupadas por elementos não nulos. O algoritmo que calcula esses elementos para formar Z_{esp} e dado por:

$$Z_{ij} = \sum_k T_{ik}Z_{kj} \tag{3.15}$$

para $j > i, k < i$

$$Z_{ii} = D_{ii} + \sum_k T_{ik}Z_{ki} \tag{3.16}$$

para $k > i$

3.3 Método dos Vetores Esparsos

Na Seção 3.2 foram citados métodos que exploram a **esparsidade** da matriz. Esses métodos, há décadas, são utilizados com sucesso na resolução de problemas de sistemas de potência de grandes dimensões. Nesta Seção é abordada o método de vetores esparsos [28].

O método de vetores esparsos explora a **esparsidade** do vetor independente e/ou quando se necessita conhecer somente um subconjunto do vetor desconhecido. Tal fato aumenta a eficiência dos algoritmos para resolução dos problemas da rede elétrica, pois, elimina-se todas as operações matriciais desnecessárias.

Conforme visto na Seção 3.2, a matriz Y_{BARRA} da Equação (3.1) pode ser fatorada na forma mostrada na Equação (3.3). Assim, a solução da Equação (3.1) pode ser obtida em duas etapas, operando-se com as matrizes L, D e U, conforme se segue:

$$U\underline{E} = D^{-1}L^{-1}\underline{I} = \underline{Z} \tag{3.17}$$

$$\underline{E} = U^{-1}\underline{Z} \tag{3.18}$$

A Equação (3.17) define a seqüência de operações de substituição *forward* sobre LD, enquanto que a Equação (3.18) define a seqüência de operações de substituição *backward* sobre U.

Para o método de vetores esparsos é fundamental que a etapa *forward* seja executada por colunas e a etapa *backward* por linhas.

O vetor independente $\underline{I}$ é esparso em muitas aplicações (na análise de curto-circuito o caso é extremo, pois, somente um elemento do vetor é diferente de zero). Entretanto, o vetor solução $\underline{E}$, em geral, não é esparso.

Se o vetor $\underline{I}$ é esparso, somente um subconjunto das colunas de L é necessário para a solução *forward*. Esta metodologia de solução é chamada de *fast forward* (*FF*). Na situação em que apenas alguns elementos do vetor $\underline{E}$ são requeridos, somente um subconjunto das linhas de U são necessários para a solução *backward*, a qual é denominada de *fast backward* (*FB*).

O subconjunto de colunas para *FF* é uma função das estruturas esparsas de L e $\underline{I}$,

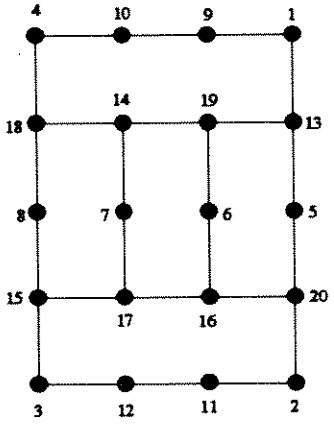


Figura 3.1: Sistema-exemplo de 20 barras

enquanto que o subconjunto de linhas para FB é uma função das estruturas esparsas de U e $\underline{E}$. As estruturas esparsas de L e U, por sua vez , dependem da **esparsidade** da matriz Y_{BARRA} e do algoritmo de ordenação adotado.

A tarefa primordial para a implementação do método de vetores esparsos é a identificação eficiente dos subconjuntos de L e U para FF e FB.

3.3.1 Caminho de Fatoração

Considere-se a rede representada na Figura 3.1, cuja representação formal é dada pela Equação (3.1).

A Figura 3.2 mostra a estrutura da matriz Y_{BARRA}

A solução da Equação (3.1) é dada pela Equação (3.2), onde se verifica a necessidade da inversão da matriz Y_{BARRA}.

A matriz Y_{BARRA} pode ser fatorada conforme mostrada na Equação (3.3).

Assim a Equação (3.2) pode ser resolvida em duas etapas, conforme mostram as Equações (3.17) e (3.18).

Uma maneira de inverter a matriz Y_{BARRA} consiste em se montar a seguinte estrutura:

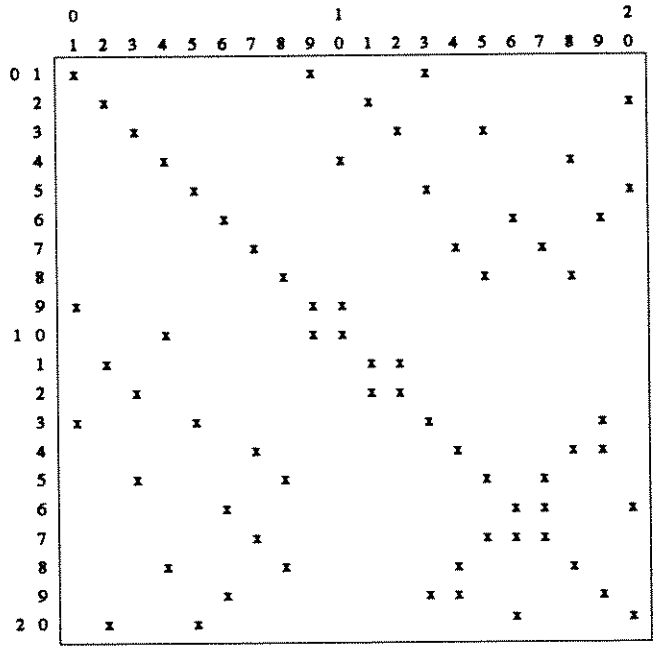


Figura 3.2: Estrutura da matriz Y_{BARRA}

$$Y_{BARRA}.I_n \tag{3.19}$$

onde I_n é uma matriz identidade da mesma ordem de Y_{BARRA}

Realizam-se, então, operações de forma a tornar o campo ocupado pela matriz Y_{BARRA} em uma matriz identidade. Quando isto ocorre, o campo ocupado originalmente pela matriz I_n passa a conter a matriz inversa de Y_{BARRA} e a situação final será:

$$I_n.Z_{BARRA} \tag{3.20}$$

O processo de cálculo da inversa da matriz Y_{BARRA} passa por três etapas:

1. zerar todos os elementos do triângulo inferior da matriz Y_{BARRA} (operação relacionada com a obtenção da matriz L);
2. tornar os elementos da diagonal iguais a 1 (operação relacionada com a obtenção da matriz D);

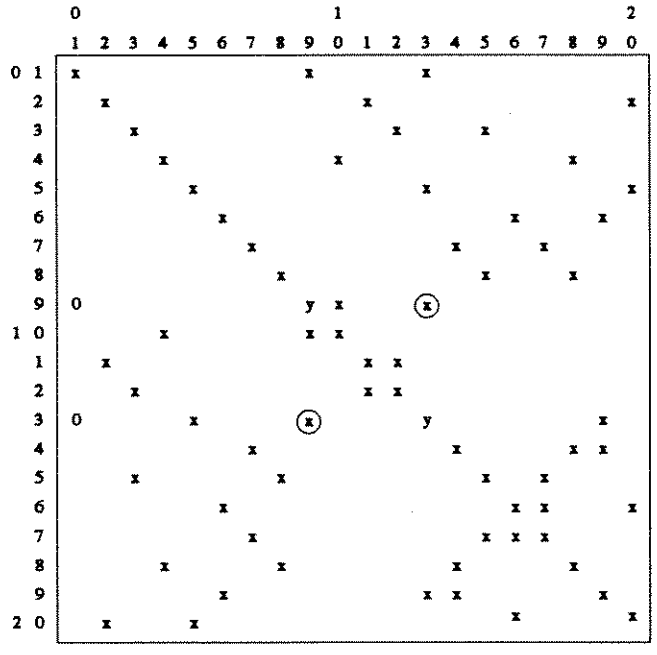


Figura 3.3: Estrutura da matriz Y_{BARRA} com a eliminação da barra 1

3. zerar todos os elementos do triângulo superior da matriz Y_{BARRA} (operação relacionada com a obtenção da matriz U).

Aplicando-se estas etapas à matriz Y_{BARRA} da Figura 3.1 obtém-se os fatores triangulares [39].

A Figura 3.3 mostra a estrutura da matriz Y_{BARRA} quando se aplica a etapa 1 nos elementos da coluna 1, o que corresponde a eliminação da barra 1 do sistema (mostrada na Figura 3.4).

Esta operação introduz alterações nas linhas/colunas 9 e 13 da matriz, conforme pode-se notar na Figura 3.3.

A continuação das operações da etapa 1, com a eliminação sucessiva todas as demais barras, possibilita formar a Tabela 3.1:

A estrutura da matriz resultante, após completar a etapa 1, é mostrado na Figura 3.5.

Dos resultados obtidos nota-se que para eliminar a barra 1 altera-se as linha/coluna

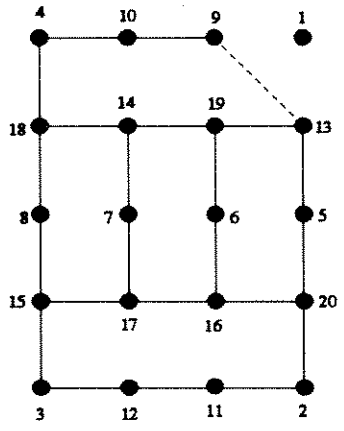


Figura 3.4: Sistema com a eliminação da barra 1

Barra Eliminada	Alterações
1	09-13
2	11-20
3	12-15
4	10-18
5	13-20
6	16-19
7	14-17
8	15-18
9	10-13
10	13-18
11	12-18
12	15-20
13	18-19-20
14	17-18-19
15	17-18-20
16	17-19-20
17	18-19-20
18	19-20
19	20

Tabela 3.1: Alterações nas linhas/colunas da matriz

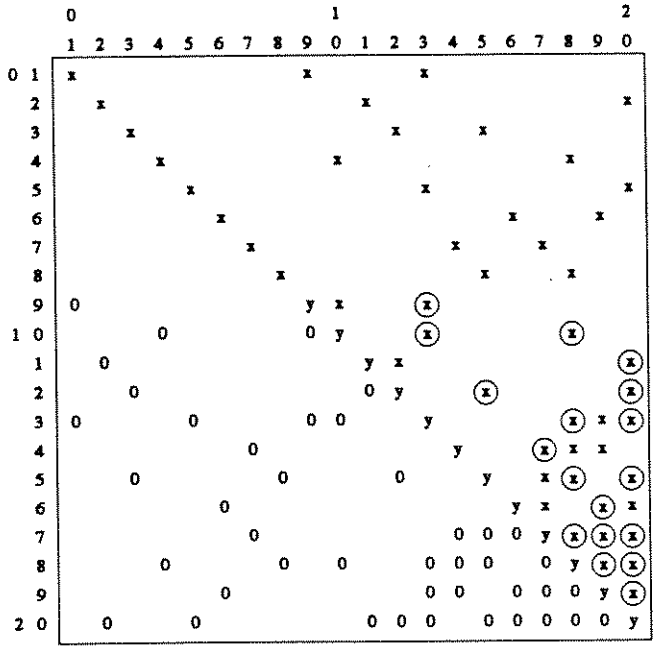


Figura 3.5: Estrutura da matriz Y_{BARRA} após etapa 1

9 e 13 e a eliminação da barra 9 implica em alterar as linha/coluna 10 e 13, e assim por diante. A partir desta análise surge o conceito de caminho de fatoração para um vetor esparso, que é definido como uma lista ordenada de colunas de L para FF, ou de linhas de U para FB.

Definindo-se um *singleton* como um vetor no qual somente um elemento é diferente de zero e considerando este elemento na posição k, o seguinte algoritmo determina o caminho de fatoração para este *singleton*:

1. Comece a lista com a coluna k de L;
2. Pegue a primeira linha que possua elemento diferente de zero na coluna k de L, faça k igual ao número da linha e inclua na lista;
3. Se k é a última coluna de L, pare. Caso contrário volte ao passo 2.

O caminho de fatoração do sistema de 20 barras é mostrado na Tabela 3.2, que pode ser representado por um grafo chamado de grafo do caminho de fatoração, mostrado na Figura 3.6.

Barra	Next	Barra	Next
1	09	11	12
2	11	12	15
3	12	13	18
4	10	14	17
5	13	15	17
6	16	16	17
7	14	17	18
8	15	18	19
9	10	19	20
10	13	20	0

Tabela 3.2: Tabela do caminho de fatoração

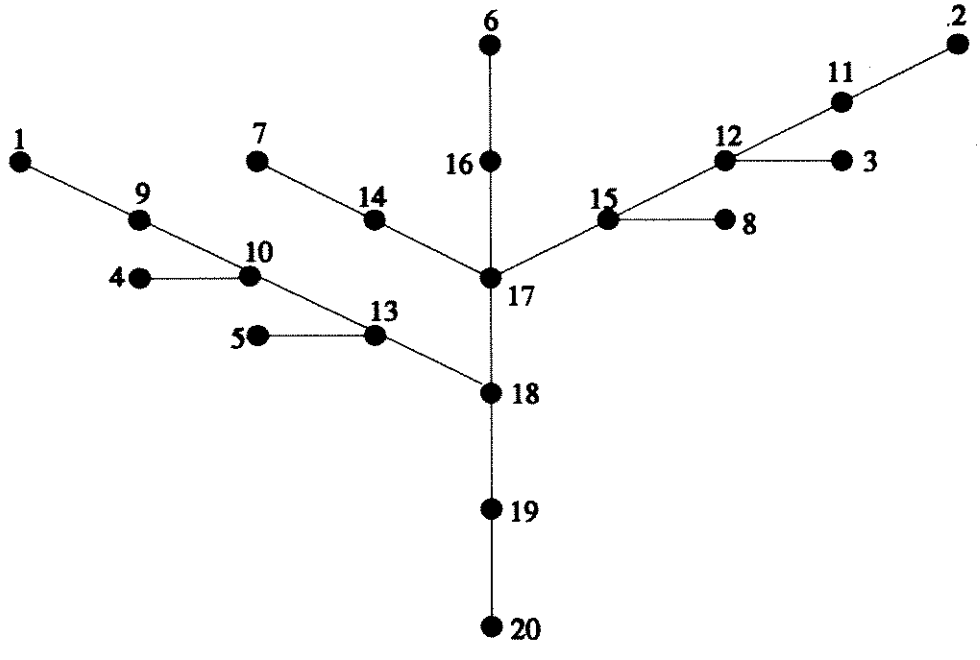


Figura 3.6: Grafo do caminho de fatoração

3.4 Acoplamentos Mútuos em Linhas de Transmissão

Num sistema de potência são comuns casos em que a energia elétrica é transmitida por linhas de transmissão em circuitos duplos, ou ainda, por vários circuitos duplos. Há casos ainda de circuitos de tensões diferentes caminharem paralelamente ao longo de seus trajetos, aproveitando a mesma faixa de servidão. Nestas situações a presença de fluxos de correntes desequilibradas, que decorrem de faltas à terra em uma linha, causam induções mútuas de correntes de seqüência zero em outras linhas paralelas. O valor da impedância mútua é cerca de 50 a 70% da impedância própria e a sua não inclusão na formação da matriz Z_{BARRA} de seqüência zero acarreta erros consideráveis nas magnitudes das correntes de curtos-circuitos à terra, fluxos de correntes e tensões de seqüências positiva e zero.

A inclusão das impedâncias mútuas na matriz Y_{BARRA} pode ser feita ou durante [11] ou após [13] a sua formação.

Sendo a impedância mútua um parâmetro entre ramos não há um modo de incluí-la diretamente na matriz Y_{BARRA}. Assim sendo, transforma-se o circuito contendo acoplamentos mútuos em um circuito equivalente no qual se introduz o efeito dessas indutâncias. No circuito equivalente aparecerão ligações artificiais, que serão tratadas como ramos comuns. As ligações artificiais do circuito equivalente são obtidas por meio de transformação de variáveis de ramos (variáveis da rede primitiva) em variáveis nodais.

Linhas paralelas com acoplamento mútuo são representadas pela Figura 3.7.

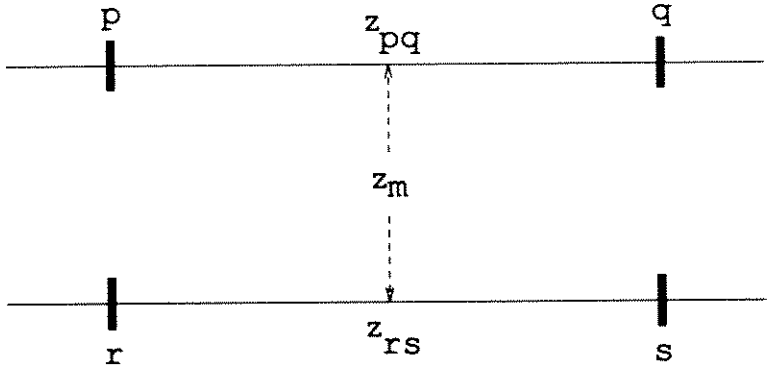


Figura 3.7: Linhas com acoplamentos mútuos

A equação que descreve este circuito é dada por:

$$\underline{V} = Z_{pri}\underline{I_r} \tag{3.21}$$

ou

$$\underline{I_r} = Y_{pri}\underline{V} \tag{3.22}$$

sendo:

$\underline{V}$ - vetor das quedas de tensão nos ramos;

Z_{pri} - matriz de impedância dos elementos da rede;

$\underline{I_r}$ - vetor das correntes nos ramos;

Y_{pri} - inversa da matriz Z_{pri}

O desenvolvimento que se segue mostra que partindo-se da Equação (3.22), obtém-se a Equação do tipo (3.1), utilizando a transformação singular.

As relações existentes entre as variáveis de ramos e as variáveis nodais são:

$$\underline{V} = A\underline{E} \tag{3.23}$$

e

$$\underline{I} = A^t\underline{I_r} \tag{3.24}$$

sendo:

A - a matriz de incidência.

Pelas Equações (3.22) e (3.23) tem-se:

$$\underline{I}_r = Y_{pri} A \underline{E} \tag{3.25}$$

Multiplicando-se ambos os membros da Equação (3.25) por A^t vem:

$$A^t \underline{I}_r = A^t Y_{pri} A \underline{E} \tag{3.26}$$

ou

$$\underline{I} = (A^t Y_{pri} A) \underline{E} \tag{3.27}$$

Portanto,

$$(A^t Y_{pri} A) = Y \tag{3.28}$$

A formulação é geral, isto é, pode ser estendida para várias linhas de transmissão mutuamente acopladas, entretanto, para ilustrar seja o caso mais simples como o mostrado na Figura 3.7.

As variáveis da rede primitiva estão relacionadas pela matriz Z_{pri}, conforme a Equação (3.21), que na forma matricial é dada por:

$$\begin{bmatrix} V_{pq} \\ V_{rs} \end{bmatrix} = \begin{bmatrix} z_{pq} & z_m \\ z_m & z_{rs} \end{bmatrix} \cdot \begin{bmatrix} I_{pq} \\ I_{rs} \end{bmatrix} \tag{3.29}$$

As quedas de tensão nos ramos em função das tensões nodais são:

$$V_{pq} = E_p - E_q \tag{3.30}$$

$$V_{rs} = E_r - E_s \tag{3.31}$$

ou

$$\begin{bmatrix} V_{pq} \\ V_{rs} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} E_p \\ E_q \\ E_r \\ E_s \end{bmatrix} \tag{3.32}$$

As correntes nodais em função das correntes nos ramos já na forma matricial é dada por:

$$\begin{bmatrix} I_p \\ I_q \\ I_r \\ I_s \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} I_{pq} \\ I_{rs} \end{bmatrix} \tag{3.33}$$

Portanto, para este caso a matriz de incidência A é:

$$A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.34}$$

Pela Equação (3.27) obtém-se:

$$Y = \begin{bmatrix} y_{pq} & -y_{pq} & y_{pq,rs} & -y_{pq,rs} \\ -y_{pq} & y_{pq} & -y_{pq,rs} & y_{pq,rs} \\ y_{pq,rs} & -y_{pq,rs} & y_{rs} & -y_{rs} \\ -y_{pq,rs} & y_{pq,rs} & -y_{rs} & y_{rs} \end{bmatrix} \tag{3.35}$$

A matriz de admitância nodal da Equação (3.35) é representada pelo circuito equivalente da Figura 3.8

3.5 Curtos-circuitos em Linhas de Transmissão

Esta Seção apresenta um método que possibilita cálculos de curtos-circuitos em linhas de transmissão sem a necessidade de criação de barras intermediárias. O ponto de simulação dos curtos- circuitos é variado automaticamente ao longo da linha subdividindo-a em tantos trechos quanto se queira. Estes cálculos podem ser efetuados mesmo em linhas que

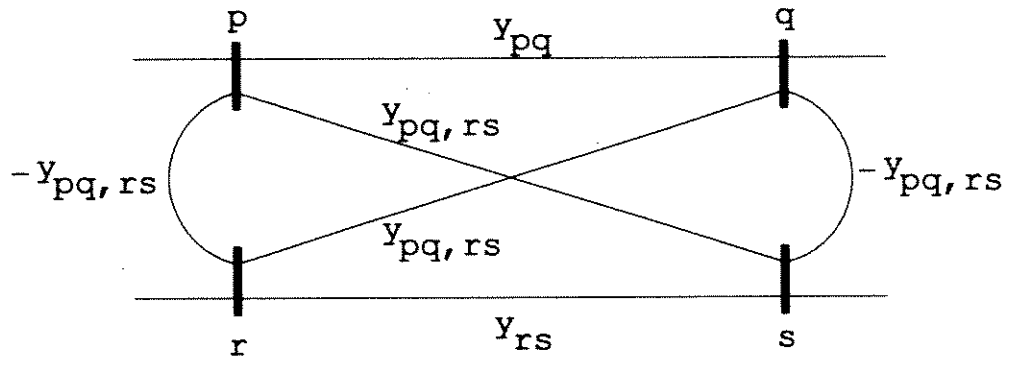


Figura 3.8: Circuito equivalente para inclusão dos acoplamentos mútuos

apresentem acoplamentos mútuos elevados [20]. Pode-se também considerar facilmente a resistência de falta no curto-circuito monofásico. São calculados curtos-circuitos trifásico e monofásico. Além do valor total no ponto de falta são obtidas também as contribuições e tensões até segunda vizinhança.

A Equação (3.22) pode ser escrita sob a forma:

$$I_{pq} = y_{pq}V_{pq} + y_{pq,rs}V_{rs} \tag{3.36}$$

$$I_{rs} = y_{rs,pq}V_{pq} + y_{rs}V_{rs} \tag{3.37}$$

Supondo que a corrente I_f é injetada num ponto arbitrário da linha $r-s$ com acoplamento mútuo a uma distância a de r conforme mostrado na Figura 3.9, a relação entre as correntes e as tensões nos ramos é dada pelas Equações:

$$V_{pq} = z_{pq}I_{pq} + az_mI_{rf} - (1-a)z_mI_{sf} \tag{3.38}$$

$$V_{rs} = z_mI_{pq} + az_mI_{rf} - (1-a)z_{rs}I_{sf} \tag{3.39}$$

$$I_f = -(I_{rf} + I_{sf}) \tag{3.40}$$

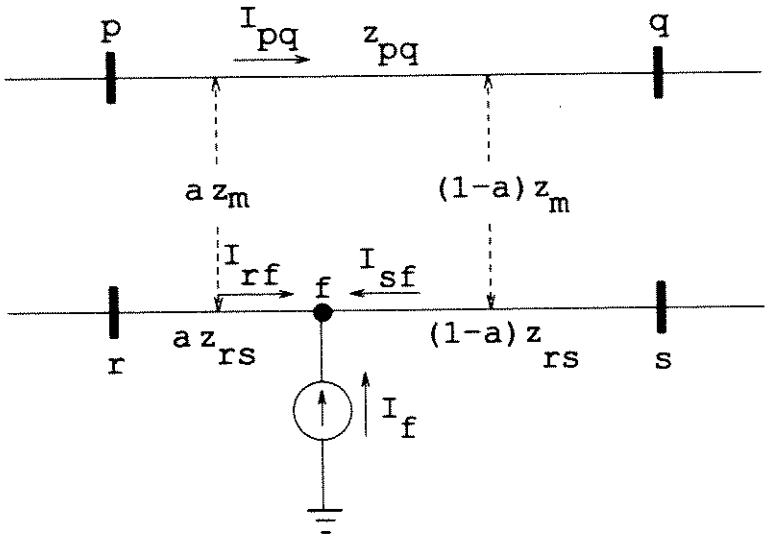


Figura 3.9: Ponto de curto-circuito na linha de transmissão

Escrevendo as Equações (3.38) e (3.39) na forma matricial tem-se:

$$\begin{bmatrix} V_{pq} \\ V_{rs} \end{bmatrix} = \begin{bmatrix} z_{pq} & z_m \\ z_m & z_{rs} \end{bmatrix} \cdot \begin{bmatrix} I_{pq} \\ aI_{rf} - (1-a)I_{sf} \end{bmatrix} \tag{3.41}$$

ou

$$\begin{bmatrix} I_{pq} \\ aI_{rf} - (1-a)I_{sf} \end{bmatrix} = \begin{bmatrix} y_{pq} & y_{pq,rs} \\ y_{rs,pq} & y_{rs} \end{bmatrix} \cdot \begin{bmatrix} V_{pq} \\ V_{rs} \end{bmatrix} \tag{3.42}$$

Das Equações (C.13) e (3.40) vem:

$$I_{pq} = y_{pq}V_{pq} + y_{pq,rs}V_{rs} \tag{3.43}$$

$$I_{rf} = y_{rs,pq}V_{pq} + y_{rs}V_{rs} - (1-a)I_f \tag{3.44}$$

$$I_{sf} = -y_{rs,pq}V_{pq} - y_{rs}V_{rs} - aI_f \tag{3.45}$$

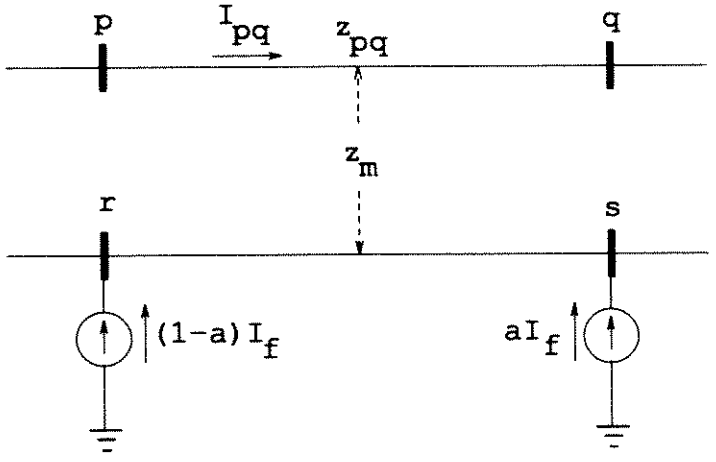


Figura 3.10: Representação da corrente de curto-circuito pela injeção das correntes compensadas nos terminais da linha de transmissão com curto-circuito

Comparando as Equações (3.43), (3.44) e (3.45) com as Equações (3.36) e (3.37), tem-se:

$$I_{rf} = I_{rs} - (1-a)I_f \tag{3.46}$$

$$I_{sf} = -I_{rs} - aI_f \tag{3.47}$$

ou

$$(1-a)I_f = I_{rs} - I_{rf} \tag{3.48}$$

$$aI_f = -I_{rs} - I_{sf} \tag{3.49}$$

Obtém-se assim o circuito equivalente mostrado na Figura 3.10

A tensão no ponto *f*, baseando-se no circuito da Figura 3.9 é dada por:

$$E_f = E_s + (1-a)z_m I_{pq} + (1-a)z_{rs}(-I_{sf}) \tag{3.50}$$

A Equação (3.39) pode ser expressa em termos das tensões nodais ou seja:

$$E_r - E_s = z_m I_{pq} + a z_{rs} I_{rf} - (1-a) z_{rs} I_{sf} \tag{3.51}$$

ou

$$z_m I_{pq} = E_r - E_s - a z_{rs} I_{rf} - (1-a) z_{rs} I_{sf} \tag{3.52}$$

Substituindo a Equação (3.52) na Equação (3.50) vem:

$$E_f = E_s - (1-a)[(E_r - E_s) - a z_{rs} I_{rf} - (1-a) z_{rs} I_{sf}] + (1-a) z_{rs}(-I_{sf}) \tag{3.53}$$

que simplificando resulta:

$$E_f = (1-a)E_r + aE_s + (1-a) z_{rs} I_f \tag{3.54}$$

As tensões nodais E_r e E_s são obtidas a partir do Sistema de Equação (3.2) e o vetor corrente nodal baseado no modelo apresentado na Figura 3.10, isto é:

$$I = \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ (1-a)I_f \\ \vdots \\ 0 \\ \vdots \\ aI_f \\ \vdots \\ \vdots \\ 0 \end{bmatrix} \tag{3.55}$$

portanto,

$$\begin{bmatrix} E_1 \\ \vdots \\ \vdots \\ E_p \\ \vdots \\ E_q \\ \vdots \\ E_r \\ \vdots \\ \vdots \\ E_s \\ \vdots \\ \vdots \\ E_n \end{bmatrix} = \begin{bmatrix} Z_{1r}(1-a)I_f + Z_{1s}aI_f \\ \vdots \\ \vdots \\ Z_{pr}(1-a)I_f + Z_{ps}aI_f \\ \vdots \\ Z_{qr}(1-a)I_f + Z_{qs}aI_f \\ \vdots \\ Z_{rr}(1-a)I_f + Z_{rs}aI_f \\ \vdots \\ \vdots \\ Z_{sr}(1-a)I_f + Z_{ss}aI_f \\ \vdots \\ \vdots \\ Z_{nr}(1-a)I_f + Z_{ns}aI_f \end{bmatrix} \tag{3.56}$$

Substituindo na Equação (3.54) os elementos E_r e E_s da Equação (3.56) tem-se:

$$E_f = [(1-a)^2 Z_{rr} + 2a(1-a)Z_{rs} + a^2 Z_{ss} + (1-a)aZ_{rs}]I_f \tag{3.57}$$

ou

$$Z_f = (1-a)^2 Z_{rr} + 2a(1-a)Z_{rs}) + a^2 Z_{ss} + (1-a)aZ_{rs} \tag{3.58}$$

onde Z_f é a impedância equivalente vista do ponto *f*.

As formulações para cálculos de curtos-circuitos em linhas de transmissão são desenvolvidas no Apêndice A.

3.6 Equivalentes Externos

O primeiro trabalho analítico sobre a obtenção de equivalentes externos em sistemas de potência foi publicado por Ward [1] em 1949, baseado na experiência adquirida com a utilização de analisadores de rede. O advento de computadores digitais diminuiu, por algum tempo, a necessidade de equivalentes externos. Entretanto, o crescimento das dimensões da rede bem como a necessidade de se analisarem casos múltiplos, tornaram novamente desejável, ou até mesmo indispensável em algumas aplicações, a utilização de equivalentes externos. Uma revisão do estado-da-arte pode ser encontrada nas Refêrencias [21] e [22].

Para obtenção do equivalente externo o sistema elétrico deve ser dividido em três subsistemas: interno (I), fronteira (F) e externo (E), conforme mostrado na Figura 3.11.

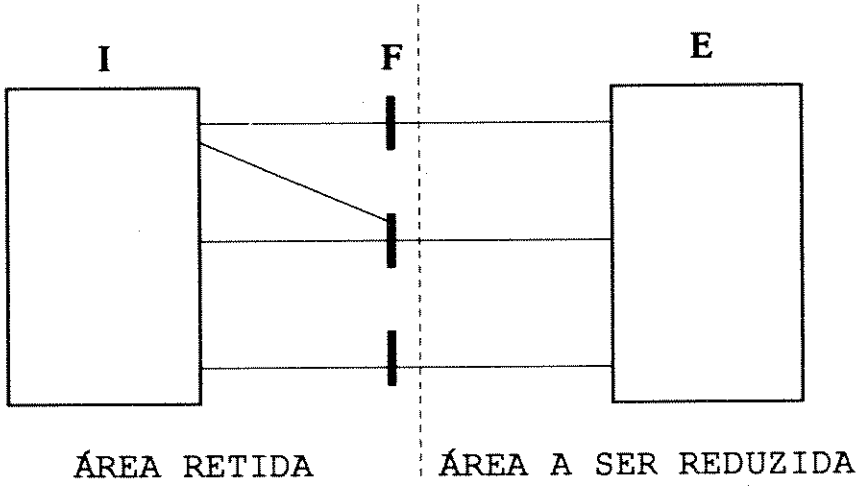


Figura 3.11: Sistema total

Considerando-se o desenvolvimento mostrado no Apêndice B chega-se ao sistema reduzido mostrado na Figura 3.12.

A utilização de equivalentes externos deve ser analisada com cuidado quando o subsistema fronteira apresentar um número elevado de barras. Neste caso, os circuitos equivalentes produzidos pela redução da rede elétrica serão mais densos do que a rede original, isto é, perde-se a característica esparsa da matriz dos coeficientes. Para ilustrar isso tomemos um sistema com 1000 barras e 1500 ramos. Supondo-se que o sistema original foi reduzido retendo-se 200 barras e que destas, 100 são barras de fronteira. Assim, serão produzidas $(100 * 99)/2 = 4950$ ramos equivalentes, que se adicionarão aos ramos do sis-

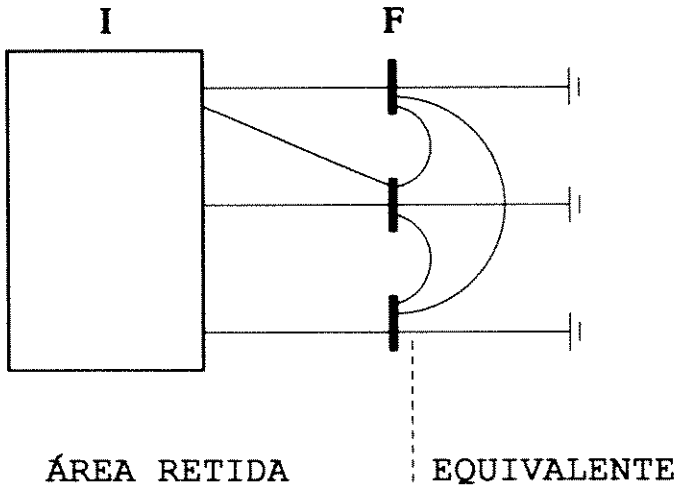


Figura 3.12: Sistema reduzido

tema retido. Evidentemente muitos desses ramos irão apresentar impedâncias elevadas que poderão ser desconsideradas. Mesmo assim, a redução pode produzir um sistema com aproximadamente três vezes mais ramos do que o sistema original [44].

Capítulo 4

Curto-circuito Probabilístico

4.1 Introdução

Cálculos de curtos-circuitos são efetuados rotineiramente em vários estudos de sistemas de potência para determinar correntes e tensões em diferentes partes da rede elétrica [27].

O crescimento da demanda de energia elétrica implica num sistema elétrico de potência cada vez mais interligado, onde grandes blocos de energia são gerados e transmitidos numa área extremamente extensa. Este fato faz com que o sistema de transmissão seja reforçado de dois modos: horizontalmente, pela adição de linhas de transmissão e verticalmente, pela instalação transformadores de força [10], tendo como conseqüência um crescimento, às vezes dramático, das potências de curtos-circuitos, fazendo com que a capacidade dos equipamentos seja aparentemente superada.

Considerando-se que as análises são baseadas em cálculos de curtos-circuitos determinísticos, os quais impõem a utilização de fatores de segurança conservativos devido ao reduzido conhecimento das aleatoriedades das variáveis envolvidas, alguns questionamentos podem surgir, tais como:

- qual a probabilidade de ocorrência da pior condição?
- os riscos de falha são aceitáveis?

Responder estas questões implica em ter o domínio sobre os riscos, que como conseqüência pode trazer economias consideráveis, pela adoção de fatores de segurança rea-

listas em novas instalações ou adiamento de substituição de equipamentos em instalações existentes. Para que isso seja possível exige-se uma análise sob a ótica probabilística.

Os primeiros trabalhos que consideravam a variabilidade estatística dos parâmetros, em estudos específicos, podem ser vistos nas Referências [3] e [9]. Na Referência [3] é mostrado o estudo de desempenho da linha de transmissão sob descargas atmosféricas pelo método de *Monte Carlo* e a Referência [9] faz um estudo probabilístico da coordenação de isolamento em linhas de transmissão de alta tensão. Em termos da aplicação de técnicas probabilísticas nos estudos de curtos-circuitos em sistema de potência foi abordado pela primeira vez no trabalho da Referência [18] e mais tarde no trabalho similar na Referência [23], utilizando o método de *Monte Carlo*. No início da década de 80 a análise de curto-circuito probabilístico pelo método de *Monte Carlo* era dispedioso em termos computacionais. Assim surgiu o método analítico, descrito na Referência [24] .

Assim, dois métodos são utilizados para estudo de curto-circuito probabilístico: método analítico e método de *Monte Carlo*. O método analítico baseia-se nas expressões derivadas da teoria da probabilidade. É aplicável somente para curto-circuito trifásico, mas, fornece resultados com boa precisão com custo computacional reduzido [32]. Era um método alternativo, em virtude de o método de *Monte Carlo* exigir recursos computacionais maiores do que existentes naquela época. O método de *Monte Carlo* baseia-se num processo de geração de números aleatórios a partir de uma distribuição uniforme. Esta simulação é utilizada para obter a função densidade de probabilidade das correntes de curtos-circuitos de uma determinada área de interesse do sistema elétrico. Para que esta função retrate a situação mais real possível é necessária uma quantidade extremamente grande de simulações.

As correntes de curtos-circuitos são os dados mais importantes que influenciam nos projetos de subestações. Na maioria dos países do mundo os projetos são baseados em valores obtidos pelo método determinístico considerando-se o pior caso, o que leva a admitir fatores de segurança conservadores [31]. O tratamento probabilístico das correntes de curtos-circuitos foi iniciado praticamente a partir do início da década de '80 pelos pesquisadores ligados às entidades canadenses, como *G. L. Ford, M. A. El-Kady* [26], [27] e outros. O aumento dos níveis de curtos-circuitos, que resulta em dificuldades técnicas e restrições operativas incentivou a Ontario Hydro a desenvolver técnicas que refinassem os cálculos de projetos de barramentos feitos da forma determinística e utilizassem métodos probabilísticos [30], [31] [33] e [34]. A distribuição de probabilidade das correntes de curtos-circuitos associada com uma certa região de interesse de um sistema elétrico de potência é uma informação básica necessária em vários estudos (de planejamento, de confiabilidade, de risco, etc).

A análise de curto-circuito probabilístico em um sistema de potência tem como ob-

jetivo, em princípio, fornecer as distribuições de probabilidade das correntes de curtos-circuitos numa determinada barra ou numa região de interesse particular. Alguns estudos podem, também, necessitar da distribuição de probabilidade dos fluxos de corrente em determinadas linhas de transmissão. Estas distribuições de probabilidade, apresentadas em forma de histogramas, são formadas acumulando-se as freqüências dos valores de curtos-circuitos. Para se ter uma estimativa precisa dos histogramas são necessários milhares de simulações de curtos-circuitos. Os histogramas podem ser transformados subseqüentemente em funções densidade de probabilidade ou distribuições de probabilidade acumulada de interesse para um determinado estudo.

As correntes de curtos-circuitos são influenciadas por alguns fatores; uns invariantes e outros que variam aleatoriamente. Assim, as correntes de curtos-circuitos dependem fundamentalmente [43]:

1. Das características estruturais do sistema, tais como as impedâncias dos diversos componentes e os comprimentos das linhas de transmissão. Estas características geralmente são bem conhecidas e invariantes.
2. Das características operacionais do sistema, definidas em função do nível de solicitação e da topologia da rede no instante da ocorrência do curto-circuito. Assim, pode-se dizer que o número de geradores, transformadores e linhas de transmissão que estão em operação num determinado instante varia aleatoriamente.
3. Do tipo, do período e do local do curto-circuito, que variam de modo aleatório.

Nas características operacionais estão implícitas as condições de carregamento do sistema, que varia conforme o horário do dia e a época do ano. Assim, pode-se afirmar que a topologia da rede elétrica depende da demanda.

O tipo de curto-circuito depende de alguns fatores, tais como a região e a época do ano. Por exemplo, as linhas de transmissão que passam por uma região de plantação de cana de açúcar são afetadas por queimadas, as quais provocam, na quase totalidade, curto-circuito bifásico. Já, as linhas de transmissão que atravessam regiões com níveis elevados de descargas atmosféricas, fatalmente estão sujeitas ao curto-circuito monofásico. Estes exemplos mostram também que a sazonalidade e o local têm influências no número de ocorrências de curtos-circuitos. Para considerar os efeitos de todas estas aleatoriedades seriam necessários os registros históricos do sistema elétrico.

O programa de curto-circuito probabilístico foi desenvolvido a partir de um programa determinístico, já existente, com a introdução de algumas sub-rotinas específicas. Definida a região de interesse, escolhendo-se previamente subestação em estudo, escolhe-se

aleatoriamente a linha de transmissão e em seguida, também aleatoriamente, o ponto de curto- circuito na linha. O método que possibilita os cálculos de curtos- circuitos nos pontos da linha de transmissão já foi abordado na Seção 3.5 do Capítulo 3.

O programa fornece as funções densidade de probabilidade das correntes:

- numa determinada barra de interesse;
- nos pontos atribuídos aleatoriamente numa determinada área de interesse;
- em determinadas linhas de transmissão.

Várias simulações foram feitas para avaliar o desempenho do sistema computacional, a flexibilidade do ambiente de programação paralela, os algoritmos de curto-circuito probabilístico e alguns itens relativos ao sistema elétrico. Para isso foram utilizados alguns sistemas reais das empresas do sistema elétrico brasileiro. No item relativo ao sistema elétrico foram analisadas as influências nos histogramas devidos aos:

- acoplamentos mútuos;
- polarização dos locais dos curtos-circuitos nas linhas de transmissão;
- representação por equivalentes externos.

Foi também verificada a influência das quantidades de simulações nas formas dos histogramas. Para isso as quantidades de simulações foram variadas de 5 000 a 300 000.

4.2 Simulação de *Monte Carlo*

Simulação é uma das ferramentas mais poderosas disponível para analisar projetos e operação de processos complexos ou sistemas [14]. Simular sobre modelos que apresentem o maior número possível de características de sistemas reais permite chegar a resultados precisos num tempo extremamente inferior ao da ocorrência do processo real. Estudar um sistema real por meio de uma representação artificial do processo requer a construção do modelo de simulação, que através de ajustes adequados chega-se a um resultado esperado [15].

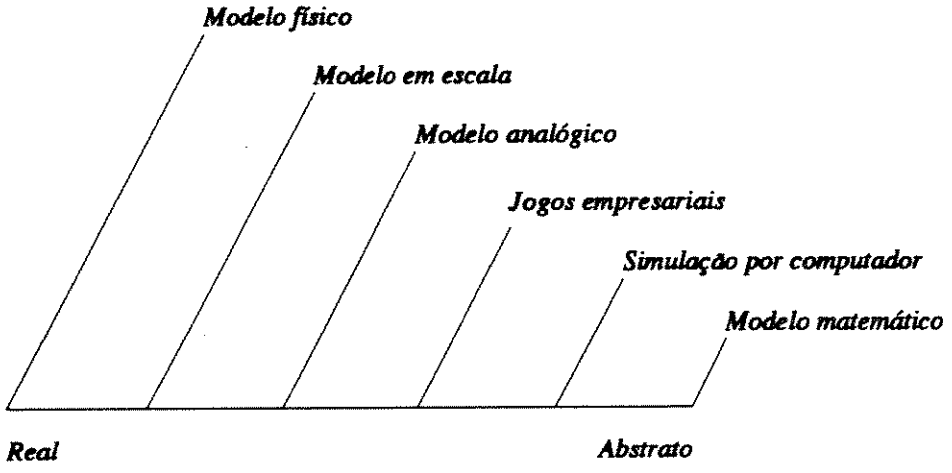


Figura 4.1: Espectro do modelo de simulação

Segundo *A. J. Rowe*, citado por *Shannon* na Referência [14], pode-se pensar no modelo de simulação como um espectro contínuo, iniciando com o modelo exato e terminando com o modelo matemático completamente abstrato, conforme ilustrado na Figura 4.1. Os modelos que iniciam este espectro são freqüentemente chamados de físico, porque se assemelham ao sistema que está sendo estudado. Os modelos reais podem ter as suas dimensões aumentadas ou diminuídas (modelos em escala). Os modelos analógicos são aqueles que as propriedades do sistema real são substituídas por propriedades que se comportam de maneiras similares. Um computador analógico no qual uma grandeza elétrica pode representar uma grandeza mecânica é um excelente exemplo de simulação analógica. Conforme se avança no espectro, chega-se naqueles modelos em que o comportamento humano e o computador se interagem. Tais simulações são freqüentemente chamados de jogos empresariais (gerenciamento, planejamento, guerra etc.) A extensão deste modelo permite-se chegar às simulações completamente computadorizadas (simulação por computador). No final do espectro tem-se os modelos matemáticos, onde se exige uma boa dose de abstração, tais modelos são largamente usados em estudos de sistema [14].

A simulação é, portanto, um processo de imitar uma realidade com modelos, que podem conservar ou não as características físicas e lógicas do sistema real. No espectro do modelo de simulação os três primeiros são denominados de **simulação física** e os três restantes de **simulação simbólica**. Na simulação simbólica os modelos não conservam as características físicas do sistema real. Neste caso, a parte lógica do sistema real, que é conservada, é traduzida através de várias equações matemáticas, onde as variáveis representam as componentes do sistema. Sem dúvida, este ramo da simulação teve um grande avanço após o aperfeiçoamento do computador digital. É interessante notar que

as análises de sistemas elétricos de potência evoluíram seguindo este espectro. No caso específico da análise de curtos-circuitos, que muitas vezes são requeridas numerosas soluções extremamente laboriosas, recorreu-se a métodos experimentais. As primeiras simulações utilizavam-se de três métodos: primeiro, usando-se o sistema real; segundo, usando um sistema experimental; e terceiro, um sistema artificial.

O primeiro método, conforme o nome sugere, utilizava-se o próprio sistema para simular: isto é, os curtos-circuitos eram provocados em locais estratégicos do sistema, os quais eram registrados para posteriores análises. Por retratar fielmente os fenômenos a serem analisados apresentava resultados reais, entretanto, por motivos de dificuldades operacionais o seu uso foi sendo restringido. No sistema experimental a rede elétrica era representada montando-se um sistema em escala reduzida, tentando reproduzir fielmente o sistema real, entretanto, alguns fenômenos inerentes às linhas longas e sobretensões eram impossíveis de serem simulados. No sistema artificial os parâmetros do sistema eram representados por resistores, reatores e capacitores (parâmetros concentrados). Este método, também conhecido como modelo miniatura, por apresentar uma série de vantagens foi muito difundido e evoluiu para possibilitar outros tipos de análises em sistemas de potência (fluxo de carga e estabilidade). Da evolução deste método surgiu o analisador de rede (*a.c. network analyser*), que possibilitava efetuar quase todas as análises em sistemas de potência. Este equipamento podia ser utilizado também para estudos de sistemas não-elétricos, só que nestes casos as suas variáveis precisavam ser convertidas em grandezas elétricas. Os analisadores de rede foram utilizados de forma intensiva até o advento dos computadores digitais.

A evolução da ciência de computação e o aparecimento de computadores digitais de alta velocidade, tornou bastante atrativo o desenvolvimento de métodos computacionais aplicados às análises de sistemas de potência. O primeiro passo para a análise de um sistema de potência é o desenvolvimento de modelos matemáticos que devem descrever as características individuais dos componentes do sistema elétrico, assim como as relações decorrentes das suas interligações. A aplicação da notação matricial para representar as equações algébricas da rede permitiu a sistematização de problemas complexos tornando possível a utilização intensa do computador digital.

A simulação simbólica, isto é, a simulação que usa o computador digital, muitas vezes é chamada de método de *Monte Carlo* e pode ser classificada em dois tipos:

a. simulação de problemas determinísticos;

b. simulação de problemas estocásticos e probabilísticos.

A simulação de problemas determinísticos se refere à resolução desses tipos de proble-

mas, tais como, integrais, derivadas, equações diferenciais, matrizes, etc., através de modelagem conveniente, em computadores digitais. A simulação de problemas estocásticos e probabilísticos abrange os casos em que por sua natureza estocástica ou probabilística não podem ser resolvidos através de métodos matemáticos usuais e a simulação é o melhor ou muitas vezes o único método de resolução [15]. Este processo de simulação é feito através de números denominados **aleatórios ou randômicos**, os quais podem apresentar as ocorrências aleatórias que caracterizam os problemas a serem resolvidos pelo método de *Monte Carlo*.

O idéia do método de *Monte Carlo* surgiu durante uma conferência no Laboratório Nacional de *Los Alamos* após a 2a. guerra mundial. Nesta conferência, quando foram apresentadas as experiências adquiridas com *ENIAC*, *Dr. Stanislaw Ulam* logo pressentiu que a utilização da potencialidade dessa nova máquina era apropriado para técnicas de amostragem estatística. *Dr. John von Neumann*, consultor do Laboratório, notou que a observação de *Ulam* tinha aplicação, entre outras coisas, no estudo da difusão do neutron através dos materiais. Além disso, ele se interessou na geração de números uniformemente distribuídos, a base para obter algoritmos apropriados, e várias outras distribuições desejadas [29].

Antes do advento do computador digital, números aleatórios eram gerados por algum tipo de dispositivo, tais como roletas, dados especiais, cartões numerados retirados manualmente, etc. Tais métodos eram adequados somente para casos em que se necessitavam de poucos números. Com a necessidade de se gerar números aleatórios em grande quantidade, e obviamente com grande rapidez, foram construídos dispositivos eletrônicos para tal finalidade. Sem dúvida, o mais conhecido foi o desenvolvido pela *RAND Corporation*, no qual usava-se um gerador eletrônico de pulso comandado por uma fonte de ruído. Um milhão de números aleatórios gerados por este dispositivo foi publicado num livro em 1955 (*"A million Random Digits with 100,000 Normal Deviates"*) que era, também disponível em fita magnética. Entretanto, a utilização dos resultados do tal dispositivo apresentava dificuldades operacionais [14].

Com o advento do computador digital desenvolveu-se métodos matemáticos para gerá-los **dentro do próprio computador**. Isto é, introduz-se no programa principal uma subrotina geradora de números aleatórios que é chamada cada vez que que for necessária durante o processo de simulação. Partindo da premissa de que os números aleatórios devem se comportar segundo a mesma distribuição de probabilidade do fenômeno físico dos eventos aleatórios simulados, a questão é: estes números são realmente aleatórios? A resposta a esta questão é que testes estatísticos são feitos com os números aleatórios gerados de modo a garantir o seu caráter aleatório. Entretanto, seriam necessários gerar infinitos números aleatórios por um mesmo processo e submetê-los a um número infinito de testes estatísticos. Nesse sentido, jamais poderíamos ter números aleatórios genuínos,

mas, **números pseudo-aleatórios** [15]. Os números pseudo-aleatórios gerados por um computador digital constituem uma seqüência de números calculados matematicamente por uma regra pré-fixada e que são aprovados em testes estatísticos de aleatoriedade porque são gerados em quantidades extremamente grandes, antes de ser iniciada a geração da mesma seqüência.

4.2.1 Geração de Números Pseudo-aleatórios

No modelo de simulação computadorizado, devemos considerar: 1. a obtenção de números aleatórios uniformemente distribuídos e 2. utilização desses números aleatórios para gerar variáveis aleatórias com características desejadas. Geralmente os computadores já possuem sub-rotinas que fazem parte da sua biblioteca de programas.

Método do quadrado médio

Proposto por *Metropolis e von Neumann* em 1946 e cuja idéia básica era obter um número formado por m dígitos médios de um número obtido através do quadrado de um número de m dígitos. Este método, infelizmente, era difícil de analisar, relativamente lento e estatisticamente insatisfatório, sendo aqui citado mais pela importância histórica por ser um dos primeiros métodos.

Colocando em passos tem-se:

1. escolha um número de quatro dígitos;

2. eleve este número ao quadrado e coloque zeros à esquerda se necessário, formando oito dígitos;

3. selecione os quatro dígitos do meio para usar como número aleatório a ser usado;

4. com o número obtido no passo 3 repita o passo 2;

5. repita o passo 3 e 4 até que se obtenha a quantidade de números aleatórios desejados.

Método congruencial linear

O método mais popular em uso atualmente para geração de números pseudo-aleatórios é o método congruencial. Ele é baseado no conceito matemático da congruência ou resíduos da teoria dos números e foi originalmente apresentado por *D. H. Lehmer* em 1951. Várias relações recursivas podem ser usadas e muitas variações deste método foram implementadas.

A relação fundamental deste método e:

$$IX_{i+1} = (aIX_i + c) \pmod{m}, i \geq 0 \tag{4.1}$$

Onde IX_0 é a semente, a o multiplicador, c o incremento e o m o módulo. A expressão (4.1) significa que IX_{i+1} é o resto da divisão de $(aIX_i + c)$ por m.

Esta relação recursiva é iniciada ajustando-se $IX_i = IX_0$.

Os números pseudo-aleatórios rn_1, rn_2, $\cdots$ gerados por (4.1) satisfazem as relações:

$$0 \leq IX_{i+1} \leq (m-1) \tag{4.2}$$

e

$$0 \leq rn_{i+1} = \frac{IX_{i+1}}{m} \leq 1 \tag{4.3}$$

A expressão (4.1), quando $c > 0$ é conhecido como método congruencial misto e quando $c = 0$ é chamado método congruencial multiplicativo. Este último método tem a seguinte relação básica:

$$IX_{i+1} = aIX_i \pmod{m} \tag{4.4}$$

Resumindo, esta relação recursiva obedece os seguintes passos:

. pegue o último número aleatório IX_i;

. multiplique-o pela constante a;

. divida aIX_i por m;

. tome o resto como IX_{i+1}.

Qualquer que seja o gerador de números pseudo-aleatórios, somente um número finito de inteiros distintos pode ser gerado, após o qual a seqüência se repete. O comprimento desse ciclo é geralmente conhecido como período P. O período máximo e um grau mínimo de correlação entre os números gerados são influenciados pela escolha de a, IX_0 e m. A escolha de m depende do sistema numérico do computador (normalmente escolhe-se $m = 2^{b-1} + 1$, sendo b o número de bits do computador).

O algoritmo para obter os números pseudo-aleatórios, utilizando o método congruencial multiplicativo é o seguinte:

1. escolha a semente (IX_0) com menos de nove dígitos.
2. multiplique-o por a, de pelo menos quatro dígitos, e armazene-o como iy.
3. multiplique o resultado do passo 2 pelo número decimal m^{-1}.
4. escolha a parte decimal do passo 3 como um número aleatório rn.
5. faça $IX = iy$
6. repita os passos 2 a 5.

A codificação deste algoritmo em *FORTRAN* é mostrada na Figura 4.2. O valor da semente é lido como IX no programa principal. O produto $a*IX$ consiste de $2b$ bits (sendo b o número de bits do computador) dos quais os bits de ordem superior são descartados e os bits de ordem inferior permanecerão como IX_{i+1} (a instrução de multiplicação de inteiros no *FORTRAN* automaticamente descarta os bits de ordem superior). A rotina prevê uma proteção para o caso de iy assumir um valor negativo, resultado do descarte de bits de ordem superior do produto $a * IX$. Esta proteção consiste em adicionar m $(2^{b-1} + 1$, ou seja, o valor máximo inteiro suportado pelo computador). rn é a saída em ponto flutuante (número pseudo-aleatório entre 0 e 1).

```
      subroutine randu(IX, iy, rn)
      iy = a*IX
      if(iy)1, 2, 2
1     iy = iy + m
2     rn = iy
      rn = rn*(1/m)
      IX = iy
      return
      end
```

Figura 4.2: Codificação *FORTRAN* da sub-rotina para gerar números aleatórios.

4.2.2 Exemplos Ilustrativos

Geração de função de distribuição uniforme

Foram executados testes com o método congruencial multiplicativo para obtenção de função de distribuição uniforme, variando-se os tamanhos da amostra e as Figuras 4.3, 4.4 e 4.5 mostram respectivamente as distribuições obtidas com amostras de 10 000, 100 000 e 1 000 000.

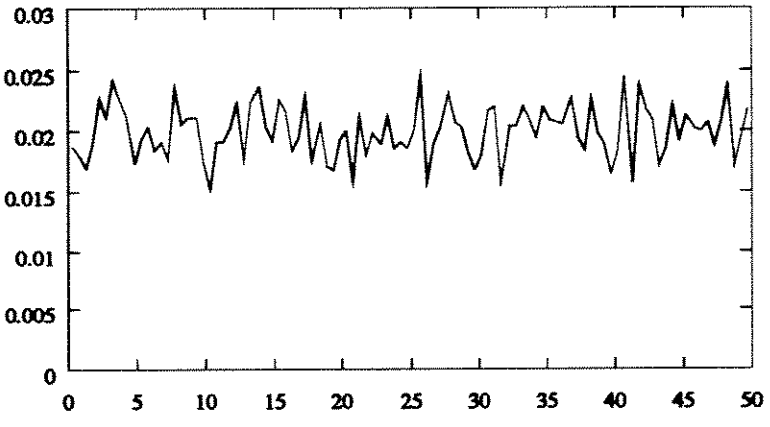


Figura 4.3: Distribuição obtida com 10 000 amostras

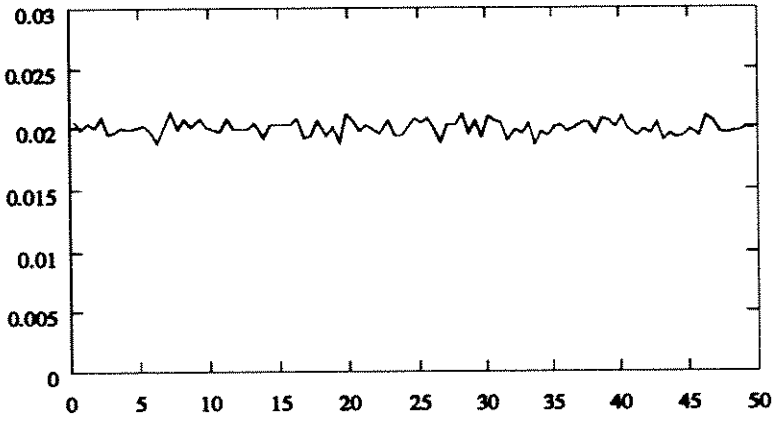


Figura 4.4: Distribuição obtida com 100 000 amostras

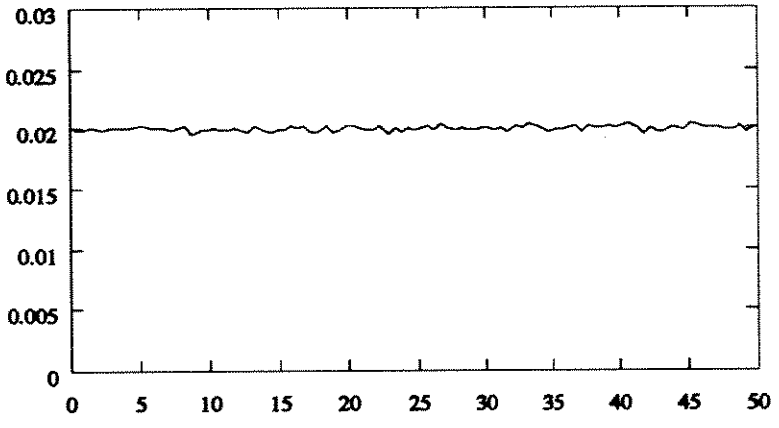


Figura 4.5: Distribuição obtida com 1 000 000 amostras

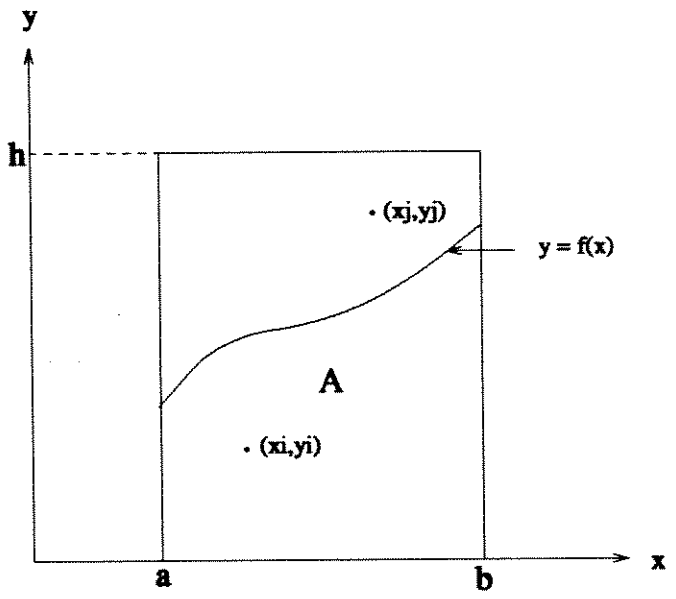


Figura 4.6: Área sob a curva $y = f(x)$

Integral de uma função

A simulação de *Monte Carlo* pode ser explicada por um exemplo simples. Na Figura 4.6 a área A limitada por uma função $y = f(x)$ no intervalo $[a, b]$ é dada pela integral definida

$$A = \int_a^b f(x)dx \tag{4.5}$$

onde $0 \leq f(x) \leq h$ e $a \leq x \leq b$.

A área A é uma parte da área total dentro do retângulo com base $(b-a)$ e altura h. Se sortearmos N pontos (x, y) ao acaso, tais que $a \leq x \leq b$ e $0 \leq y \leq h$ e se N_A desses pontos cairem dentro da área A então a freqüência relativa $\frac{N_A}{N}$ tende para $\frac{A}{h(b-a)}$ quando N tende para o infinito.

Assim sendo,

$$A = \frac{N_A}{N} h(b-a) \tag{4.6}$$

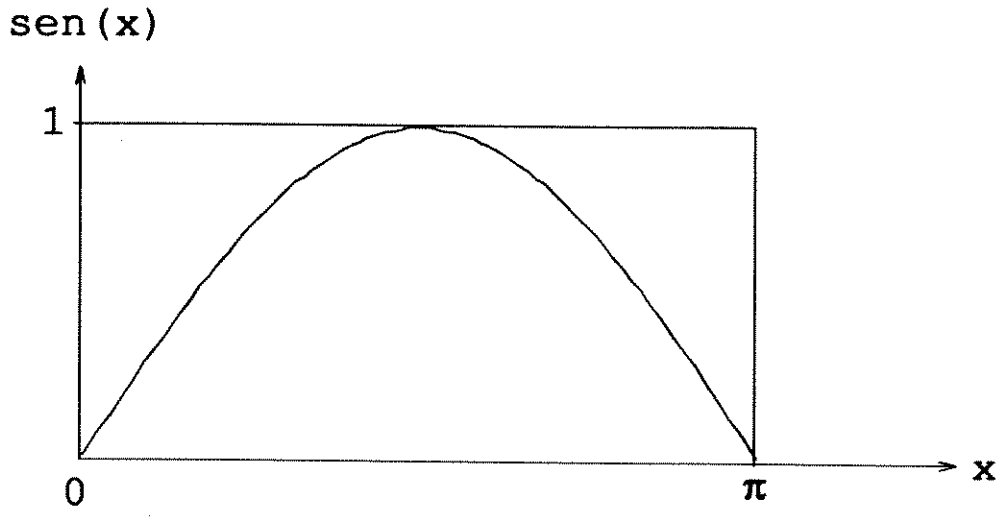


Figura 4.7: Integral da função sen(x) no intervalo $[0, \pi]$

```
No. de Pontos N  =   50000     Area  =  2,006033
No. de Pontos N  =  150000     Area  =  1,999247
No. de Pontos N  =  250000     Area  =  1,998970
No. de Pontos N  =  350000     Area  =  1,999588
No. de Pontos N  =  450000     Area  =  1,999826
```

Figura 4.8: Resultados da simulação de *Monte Carlo*.

A área A é computada usando-se um gerador de números aleatórios que sorteia as coordenadas x e y de cada ponto. Isto é feito pelas mudanças de variáveis. Se x_i e y_i são dois números aleatórios reais sorteados, calculamos $x = x_i(b-a)+a$ e $y = hy_i$ e verificamos se $y \leq f(x)$; se a desigualdade for verdadeira significa que o ponto caiu dentro da área A [37].

A integral da função mostrada na Figura 4.7 é 2,0 e o resultado da simulação de *Monte Carlo*, variando N, é mostrado na Figura 4.8.

4.3 Curto-circuito Probabilístico: Análises de Casos

4.3.1 Método Analítico

Sistema com uma fonte e uma linha radial

Considere-se o sistema da Figura 4.9 no qual deseja-se analisar a ocorrência de um curto-circuito trifásico.

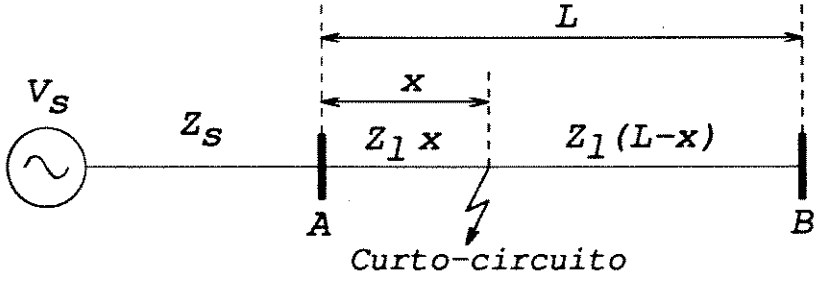


Figura 4.9: Diagrama unifilar de um sistema radial

Sendo:

Vs = tensão fase-neutro da fonte;

Zs = impedância de seqüência positiva da fonte;

Zl = impedância de seqüência positiva da linha, por unidade de comprimento;

L = comprimento total da linha.

A ocorrência de um curto-circuito trifásico a distância x da barra A é calculada deterministicamente pela Equação (4.7):

$$I_{cc} = \frac{V_s}{Z_s + Z_l x} \tag{4.7}$$

A Equação (4.7) permite determinar a variação da corrente de curto-circuito trifásico em função da distância, conforme mostrado na Figura 4.10.

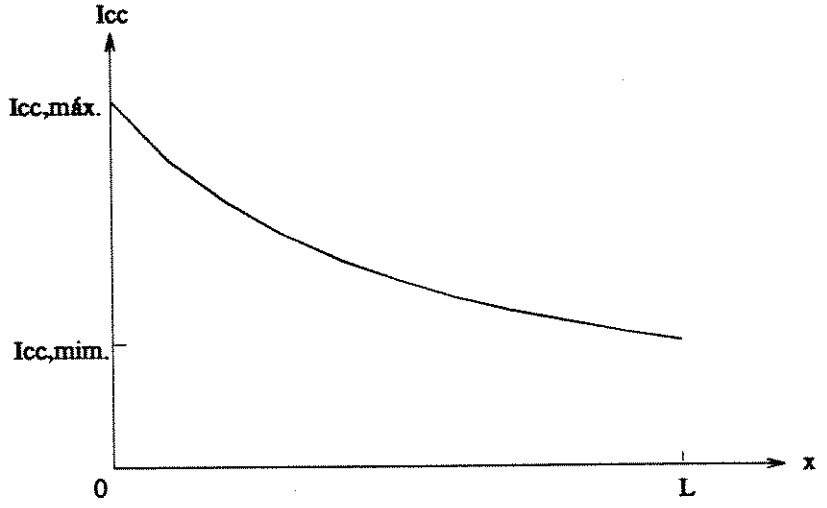


Figura 4.10: Curto-circuito em função da distância

Os valores máximo e mínimo são determinados por:

$$I_{cc,max} = \frac{V_s}{Z_s} \tag{4.8}$$

$$I_{cc,min} = \frac{V_s}{Z_s + Z_l L} \tag{4.9}$$

Estas informações, entretanto, não respondem a uma das perguntas importantes que influencia no processo de decisão, qual seja: quais as probabilidades de ocorrências das correntes máxima e mínima de curto-circuito?

Pela análise da Equação (4.7) verifica-se que a corrente de curto-circuito depende de duas características do sistema (Z_s e Z_l), de uma característica operacional (V_s, normalmente variando numa faixa restrita, o que permite considerá-lo constante) e uma característica de natureza aleatória (local do curto-circuito, ou seja, distância entre a barra em análise e o ponto de curto-circuito).

Assim sendo, a corrente de curto-circuito também passa a ser uma variável aleatória de característica a determinar.

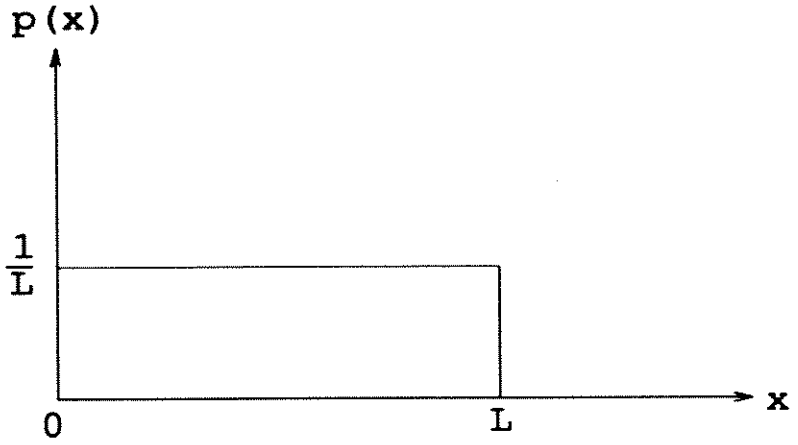


Figura 4.11: Densidade de probabilidade da variável aleatória x

Conhecendo-se a densidade de probabilidade do local do curto-circuito na linha ($p(x)$), a densidade de probabilidade da magnitude da corrente de curto-circuito ($p(I_{cc})$) pode ser obtida usando uma relação conhecida da teoria da probabilidade [8]:

$$p(I_{cc}) = p(x)\left|\frac{dx}{dI_{cc}}\right| \tag{4.10}$$

Considerando-se uma distribuição uniforme do curto-circuito na linha de transmissão a densidade de probabilidade da variável aleatória x é ilustrada na Figura 4.11 e a sua correspondente função representada pela Equação (4.11).

$$p(x) = \frac{1}{L} \tag{4.11}$$

A seguir é desenvolvida a função densidade de probabilidade para a variável aleatória I_{cc}.

Pela Equação (4.7) tem-se:

$$x = \frac{V_s}{Z_l I_{cc}} - \frac{Z_s}{Z_l} \tag{4.12}$$

Derivando x em função de I_{cc}:

$$\frac{dx}{dI_{cc}} = -\frac{V_s}{Z_l I_{cc}^2} \tag{4.13}$$

Então,

$$\left|\frac{dx}{dI_{cc}}\right| = \frac{V_s}{Z_l I_{cc}^2} \tag{4.14}$$

Substituindo as Equações (4.11) e (4.14) na Equação (4.10) resulta:

$$p(I_{cc}) = \frac{V_s}{Z_l L I_{cc}^2} \tag{4.15}$$

Esta função é ilustrada na Figura 4.12 e satisfaz as seguintes condições:

$$\frac{V_s}{Z_s + Z_l L} \leq I_{cc} \leq \frac{V_s}{Z_s} \tag{4.16}$$

e

$$p(I_{cc}) = 0 \tag{4.17}$$

para todo I_{cc} fora do intervalo.

Sistema com uma fonte e duas linhas radiais

Um sistema com uma fonte e duas linhas radiais de comprimentos diferentes é mostrado na Figura 4.13.

Neste caso, a relação básica desenvolvida previamente para uma linha de transmissão radial aplica-se para cada linha.

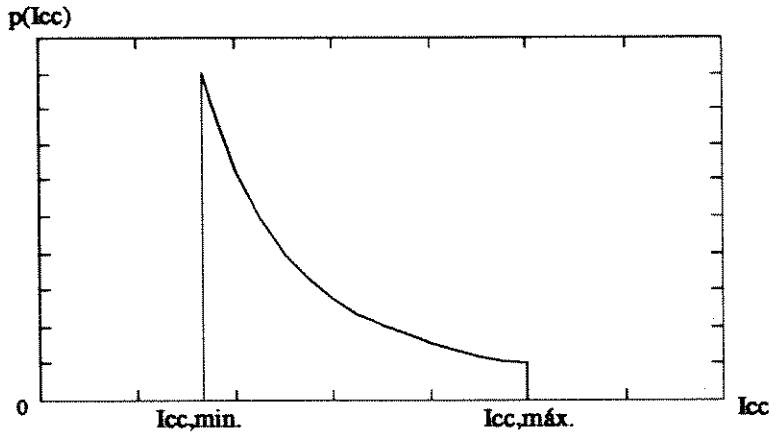


Figura 4.12: Densidade de probabilidade da variável aleatória I_{cc}

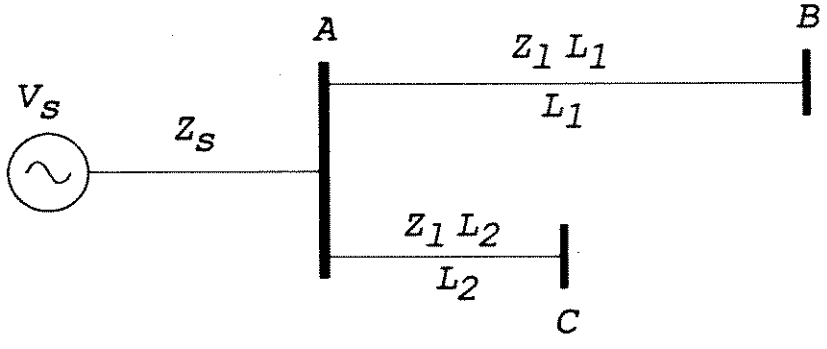


Figura 4.13: Diagrama unifilar de um sistema com duas linhas radiais

$$p_1(I_{cc}) = \frac{V_s}{Z_l L_1 I_{cc}^2} \tag{4.18}$$

no intervalo

$$\frac{V_s}{Z_s + Z_l L_1} \leq I_{cc} \leq \frac{V_s}{Z_s} \tag{4.19}$$

$$p_2(I_{cc}) = \frac{V_s}{Z_l L_2 I_{cc}^2} \tag{4.20}$$

no intervalo

$$\frac{V_s}{Z_s + Z_l L_2} \leq I_{cc} \leq \frac{V_s}{Z_s} \tag{4.21}$$

Estas duas relações definem a densidade de probabilidade da magnitude da corrente dado que o curto-circuito pode ocorrer nas linhas L_1 e L_2, respectivamente.

A densidade de probabilidade da magnitude da corrente total é dada por:

$$p_t(I_{cc}) = p(I_{cc}|cc\ na\ L_1)P(cc\ na\ L_1) + p(I_{cc}|cc\ na\ L_2)P(cc\ na\ L_2) \tag{4.22}$$

onde $p(I_{cc}|cc\ na\ L_1)$ e $p(I_{cc}|cc\ na\ L_2)$ são respectivamente as densidades de probabilidade condicional das linhas L_1 e L_2, enquanto $P(cc\ na\ L_1)$ e $P(cc\ na\ L_2)$ são as probabilidades de ocorrência de curto-circuito nas linhas L_1 e L_2.

Sendo:

$$p(I_{cc}|cc\ na\ L_1) = p_1(I_{cc}) \tag{4.23}$$

$$p(I_{cc}|cc\ na\ L_2) = p_2(I_{cc}) \tag{4.24}$$

$$P(cc\ na\ L_1)\ +\ P(cc\ na\ L_2)\ =\ 1,0 \tag{4.25}$$

tem-se:

$$p_t(I_{cc})\ =\ p_1(I_{cc})P(cc\ na\ L_1)\ +\ p_2(I_{cc})P(cc\ na\ L_2) \tag{4.26}$$

Na falta de uma análise estatística do histórico consistente do sistema que permita estimar os valores das probabilidades de ocorrências de curto-circuito nas linhas de transmissão, é razoável pressupor que elas sejam proporcionais aos seus comprimentos.

Assim,

$$P(cc\ na\ L_1)\ =\ \frac{L_1}{L_1+L_2} \tag{4.27}$$

$$P(cc\ na\ L_2)\ =\ \frac{L_2}{L_1+L_2} \tag{4.28}$$

Portanto:

$$p_t(I_{cc}) = p_1(I_{cc})\frac{L_1}{L_1+L_2} + p_2(I_{cc})\frac{L_2}{L_1+L_2} \tag{4.29}$$

As funções $p_1(I_{cc})$, $p_2(I_{cc})$ e $p_t(I_{cc})$ são mostradas na Figura 4.14.

Generalizando para n linhas de transmissão radiais a densidade de probabilidade total das magnitudes das correntes de curto-circuito é dada por:

$$p(I_{cc}) = \sum_{i=1}^{n} P(cc\ na\ L_i)p_i(I_{cc}) \tag{4.30}$$

onde:

$$\sum_{i=1}^{n} P(cc\ na\ L_i) = 1,0 \tag{4.31}$$

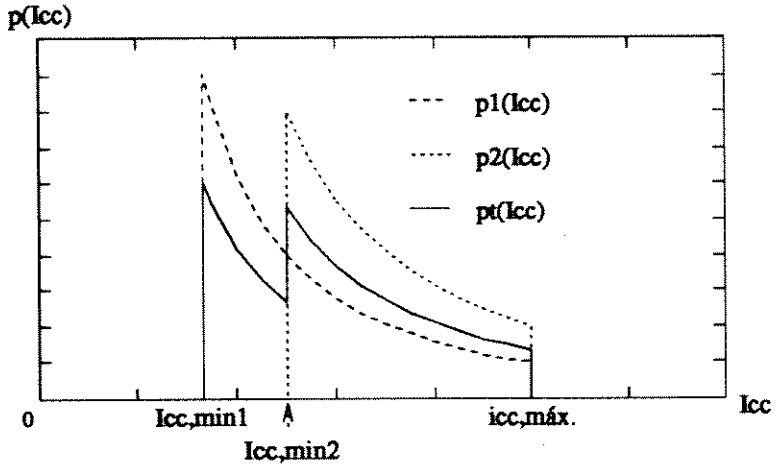


Figura 4.14: Densidade de probabilidade da variável aleatória I_{cc}

4.3.2 Método de *Monte Carlo*

Nesta Subseção são mostrados os resultados das simulações de curtos-circuitos pelo método de *Monte Carlo* para os mesmos sistemas analisados na Subseção anterior. Nestas simulações são também obtidas as densidades de probabilidade (histogramas) considerando-se uma distribuição de probabilidade não uniforme dos locais dos curtos-circuitos (neste caso, diz-se que os locais dos curtos-circuitos nas linhas foram polarizados).

Em seguida, um sistema com 3 fontes, 3 linhas, 3 barras e 2 acoplamentos mútuos é analisado, sem e com polarização.

Resultados das simulações com dois sistemas reais (Sistema B e Sistema D) são apresentados, mostrando-se os efeitos da polarização e dos acoplamentos mútuos nas funções densidades de probabilidade.

Os histogramas dos sistemas pequenos e dos sistemas reais foram obtidos respectivamente com 500 000 e 100 000 simulações de curtos-circuitos.

Em todos os casos analisados são apresentados os histogramas de curtos-circuitos trifásico e monofásico.

Para a polarização dos locais de curtos-circuitos nas linhas de transmissão, em todas as simulações, foi utilizada a distribuição de probabilidade mostrada na Figura 4.15.

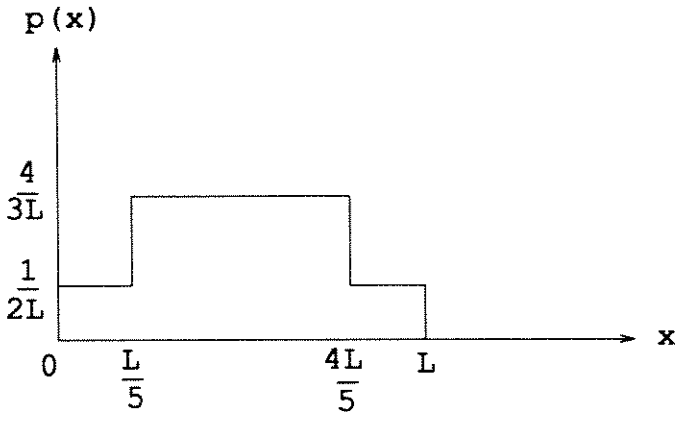


Figura 4.15: Distribuição de probabilidade dos locais dos curtos-circuitos

Sistema com uma fonte e uma linha radial

As Figuras 4.16 e 4.17 mostram respectivamente os histogramas dos curtos-circuitos não polarizados e polarizados.

Sistema com uma fonte e duas linhas radiais

A Figura 4.18 mostra o histograma dos curtos-circuitos sem polarização, enquanto que a Figura 4.19 refere-se ao histograma polarizado.

Sistema com três fontes e três linhas

Neste sistema também foi analisada a influência da polarização, conforme mostradas nas Figuras 4.20 e 4.21

Influência da polarização nos histogramas

Para analisar o efeito da polarização foi utilizado o Sistema B e os resultados são apresentados nas Figuras 4.22 e 4.23. A Figura 4.24 mostra o detalhe.

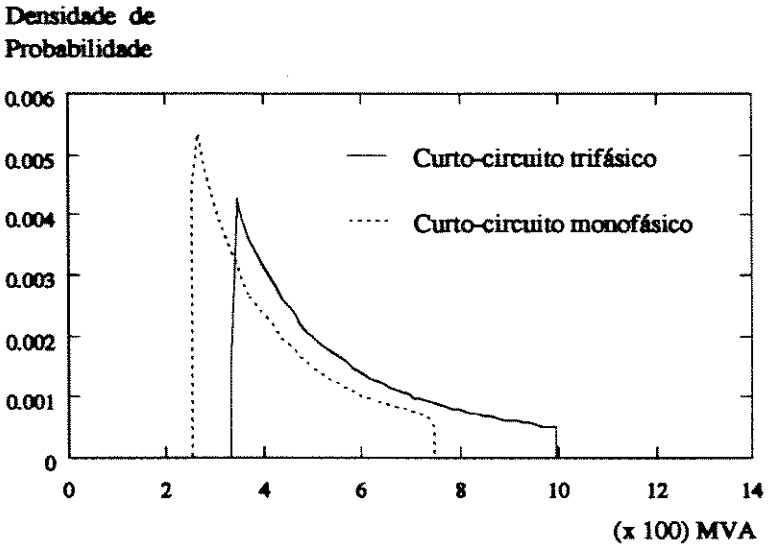


Figura 4.16: Densidade de probabilidade sem polarização

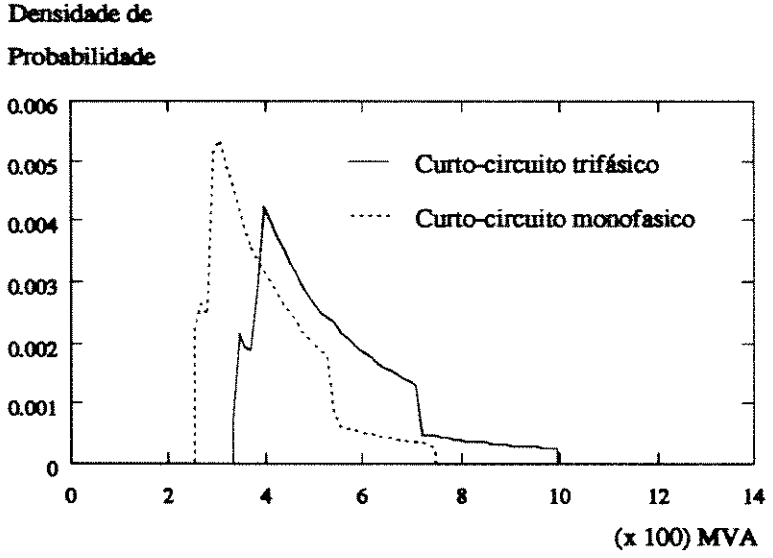


Figura 4.17: Densidade de probabilidade com polarização

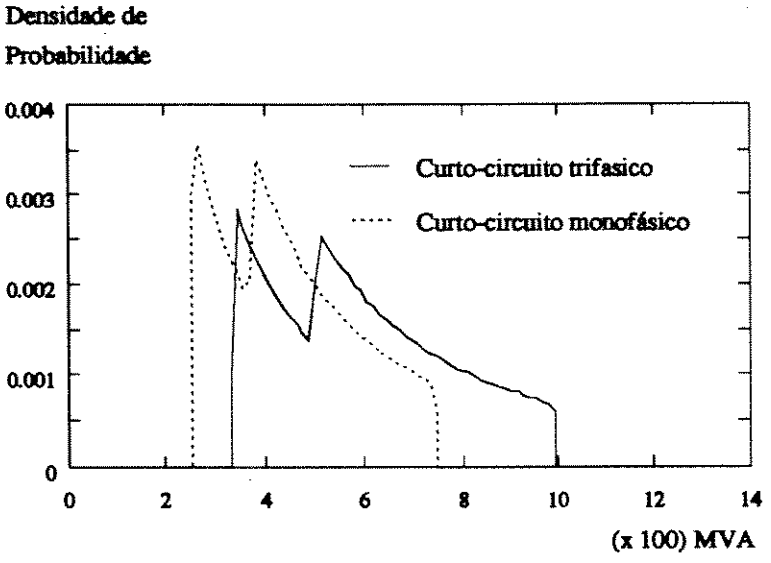


Figura 4.18: Densidade de probabilidade sem polarização

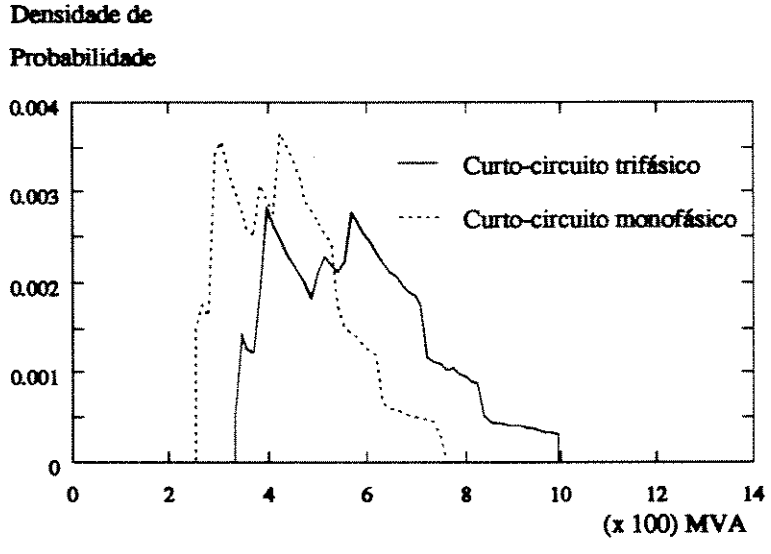


Figura 4.19: Densidade de probabilidade com polarização

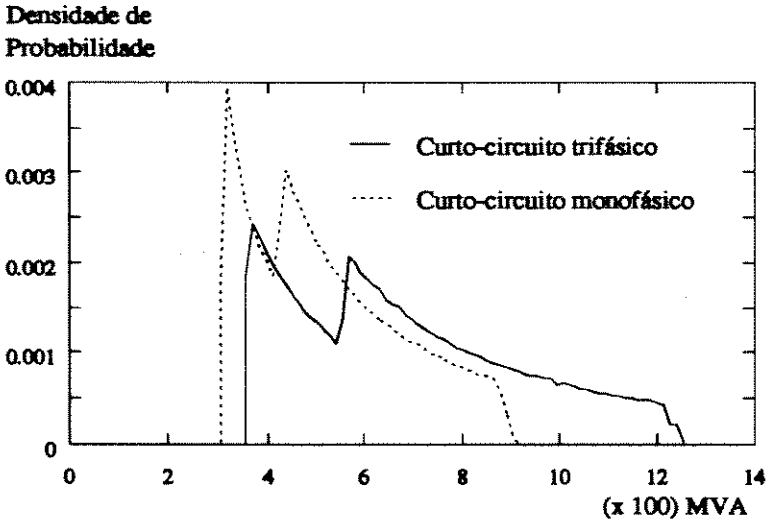


Figura 4.20: Densidade de probabilidade sem polarização

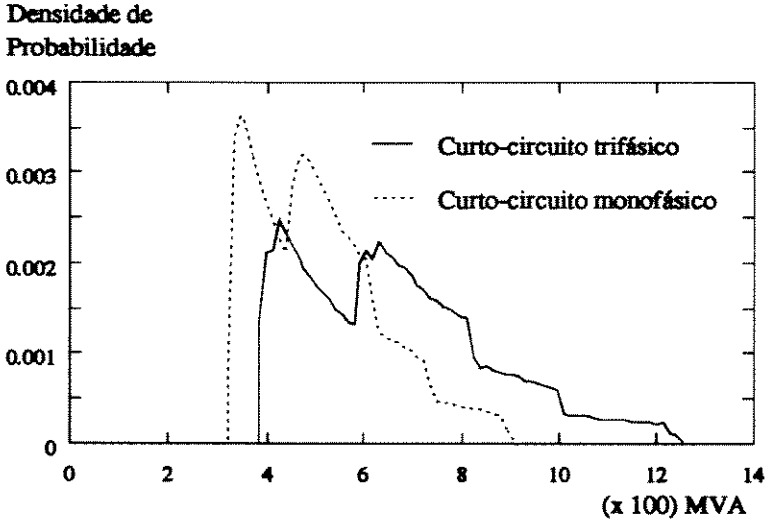


Figura 4.21: Densidade de probabilidade com polarização

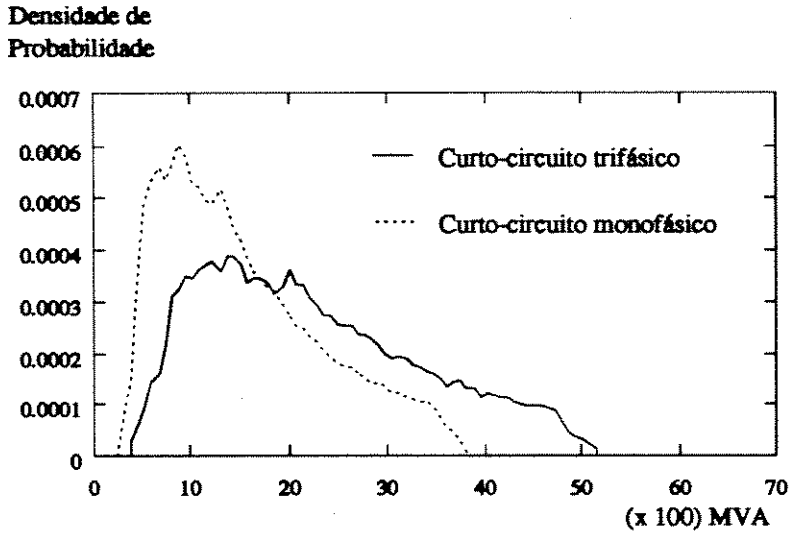


Figura 4.22: Histogramas sem polarização - Sistema B (CPFL)

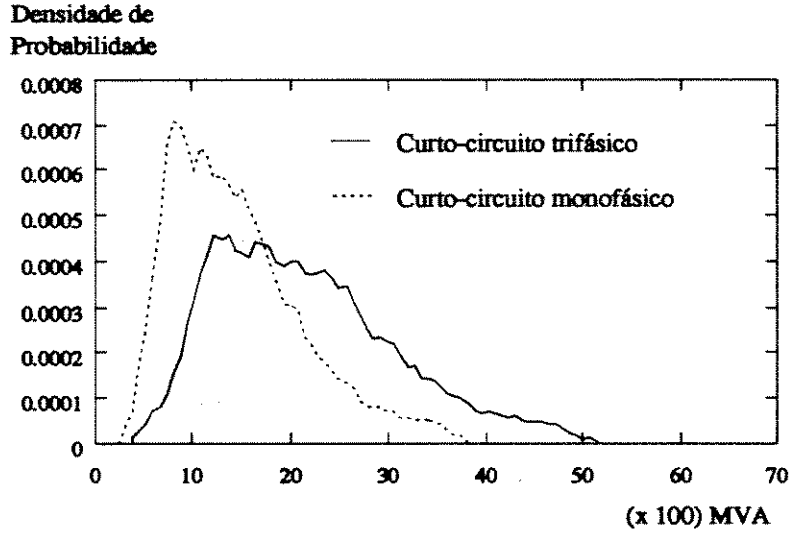


Figura 4.23: Histogramas com polarização - Sistema B (CPFL)

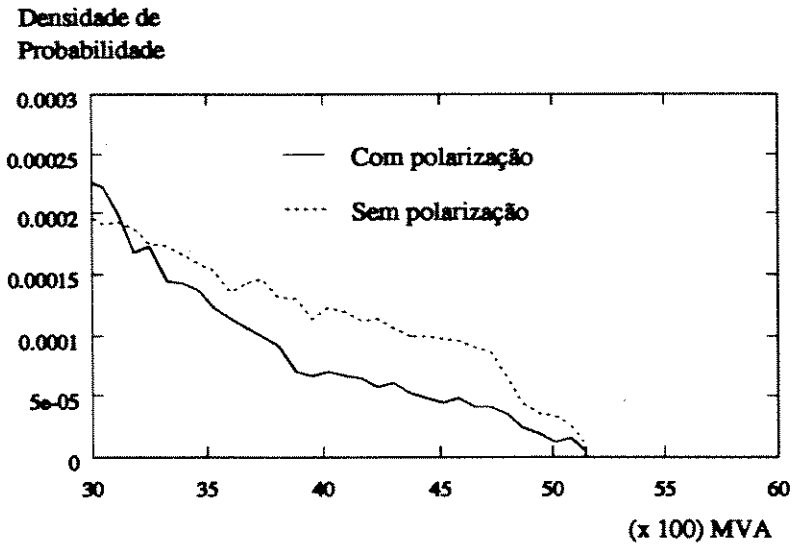


Figura 4.24: Detalhe dos histogramas do curto-circuito trifásico

Influência dos acoplamentos mútuos

A Figura 4.25 mostra a influência dos acoplamentos mútuos na rede de seqüência zero no histograma de curto-circuito monofásico. Para esta simulação foi considerado o Sistema D, que apresenta uma elevada densidade de acoplamentos mútuos.

Influência do tamanho da amostra

A Figura 4.26 mostra as formas dos histogramas variando-se os tamanhos da amostra.

Curtos-circuitos numa determinada área de interesse

A Figura 4.27 mostra as densidades de probabilidade de curtos-circuitos numa determinada área de interesse.

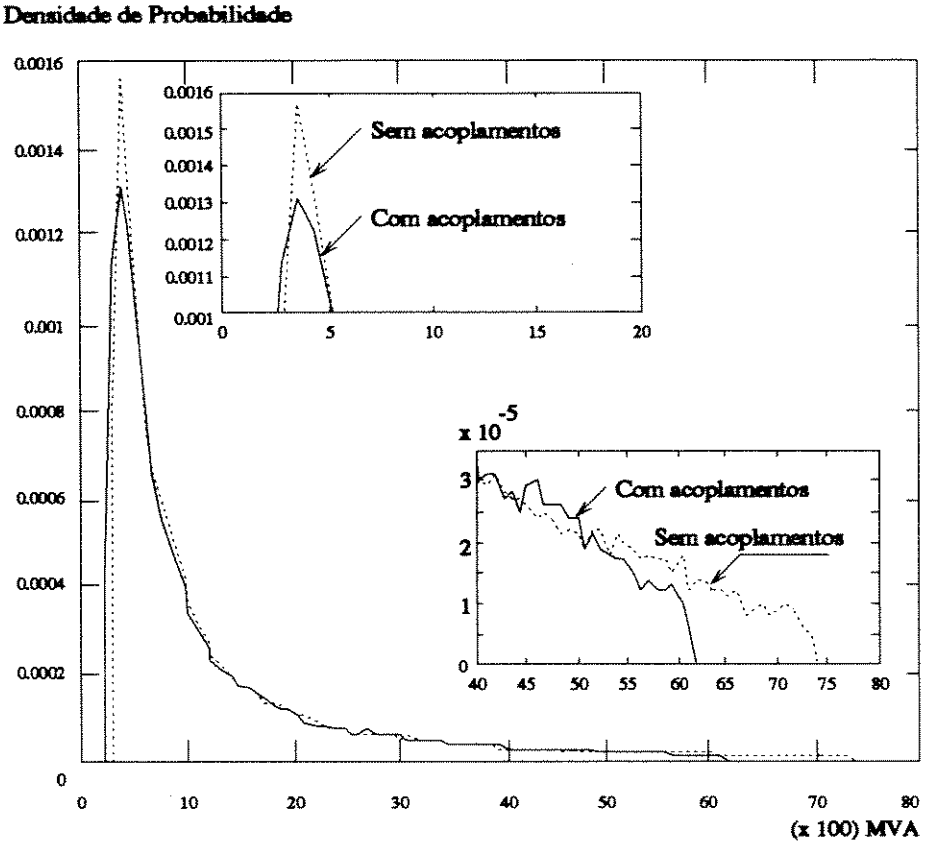


Figura 4.25: Influência dos acoplamentos mútuos - Sistema D (*LIGHT*)

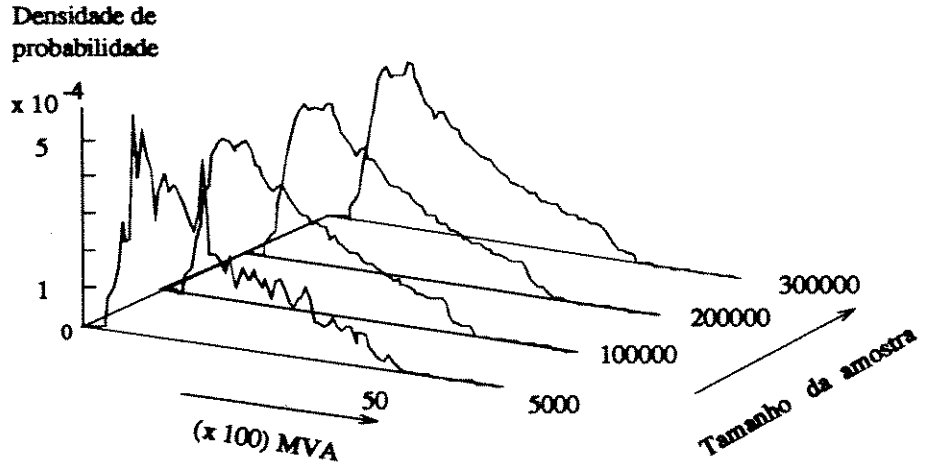


Figura 4.26: Influência dos tamanhos da amostra no histograma

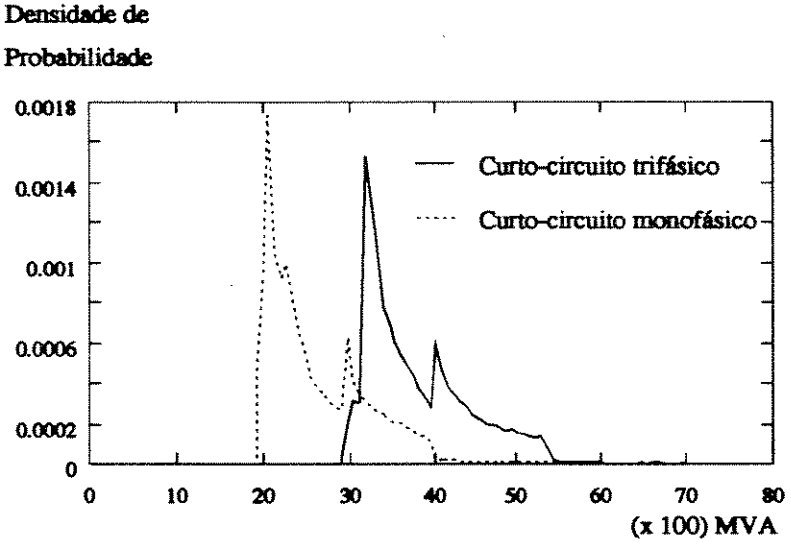


Figura 4.27: Histogramas dos curtos-circuitos numa região de interesse

Capítulo 5

Programação Paralela

5.1 Introdução

Estruturar um programa paralelo é similar a dividir um programa seqüencial em sub-rotinas. Dividir um programa em programas menores chama-se **decomposição**. Uma boa decomposição reduz um grande problema em pequenos problemas, definindo-se claramente a função para cada um deles. Numa programação convencional, a execução do programa pára enquanto a *UCP* executa uma sub-rotina, enquanto que na programação paralela as *UCPs* trabalham paralelamente.

A chave para entender os paradigmas de programação paralelo é o conceito de um **programa paralelo**. Um programa paralelo é um conjunto de instruções executado num grupo de processadores para realizar uma tarefa claramente definida. Um programa paralelo é decomponível quando os processadores podem ser divididos em sub-grupos, cada qual executando uma sub-tarefa claramente definida a partir da tarefa original. Cada sub-tarefa, portanto, também satisfaz a definição de programa paralelo.

Como exemplo, consideremos a decomposição de um programa que executa a manipulação de dados. Tal programa pode ter alguns processadores dedicados para ler a entrada de dados do disco, outros para realizar a manipulação de dados, e outros para exibir a saída gráfica num vídeo. A manipulação de dados pode, portanto, ser dividida em três programas paralelos:

- Entrada - leitura de dados do disco

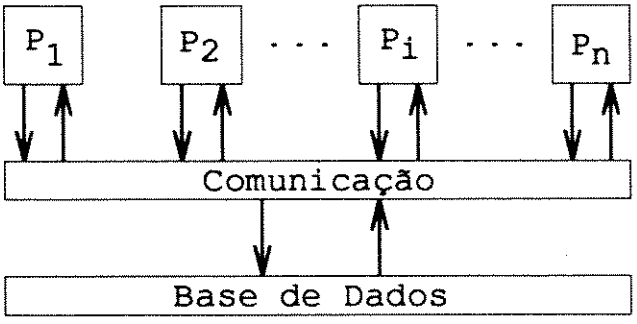


Figura 5.1: Modelo de programação abstrato

- Manipulação de dados
- Saída - exibição de um gráfico no vídeo

O programa global consiste da composição destas três partes.

Ambientes de simulação paralela são os mais variados possíveis. Sejam quais forem os ambientes eles podem ser vistos como modelo de programação paralela representado na Figura 5.1.

O modelo de programação (também chamado de paradigma de programação) é uma visão abstrata de como os computadores paralelos são programados. Este modelo de programação abstrato oculta os detalhes do *hardware* dos ambientes paralelos; escondendo-se os detalhes, a programação paralela se torna mais fácil e trazendo como conseqüência a maior portatibilidade dos programas. De acordo com este modelo, o computador paralelo é visto simplesmente como um conjunto de processadores ligados através de uma rede de comunicação.

Ambientes computacionais paralelos estão se tornando populares e recursos computacionais formados por estações de trabalho, interligadas por uma rede local, têm sua capacidade aumentada com a utilização de *softwares* que permitem o processamento distribuído, como o *PVM*. Este tipo de ambiente, quando conectado a um computador paralelo escalonável, pode vir a ter aceitação cada vez maior por parte da indústria de energia elétrica, seja para finalidades de planejamento como para configuração de futuros **Centros de Controle**. Na verdade, este trabalho é parte de um projeto mais abrangente, que pretende pesquisar a viabilidade do uso deste tipo de ambiente numa larga faixa de estudos de planejamento e operação de sistemas elétricos de potência [51].

5.2 Ambientes Utilizados

Neste trabalho foram utilizados os seguintes ambientes computacionais:

1. Computador paralelo $nCUBE2$ conectado à uma rede local através de uma estação *Sun Sparc*1
2. Rede de estações *Sun Sparc*2 com PVM
3. Rede de estações *Sun Classic* com PVM
4. Computador paralelo $SP1$ com PVM
5. Rede de estações $IBM\ RISC6000$ com PVM

Os três primeiros ambientes fazem parte de recursos disponíveis no **Departamento de Sistemas de Energia Elétrica da Faculdade de Engenharia Elétrica - DSEE/FEE**, enquanto que as duas últimas são os recursos do **Centro Nacional de Processamento de Alto Desempenho em São Paulo - CENAPAD-SP**.

O primeiro ambiente é um computador paralelo com topologia tipo hipercubo ($nCUBE2$) com 64 nós, conectado à rede local (*Local Area Network - LAN*) através de uma estação *Sun Sparc1* . Desta forma, o ambiente de programação é uma rede convencional de estações, exceto pelo fato de que a partir de qualquer estação pode-se executar aplicações no computador paralelo, que pode ser visto como um **servidor** de *number crunching* para as estações na rede. De qualquer estação, o hipercubo pode ser visto como um co-processador matemático a ela ligado, apesar dele estar fisicamente ligado a apenas uma das estações. A Figura 5.2 ilustra a configuração básica de um dos ambientes computacionais utilizados.

O segundo é uma rede heterogênea (*Ethernet*) de estações *Sun Sparc1 e Sparc2* composta de onze máquinas (rede de estações onde está conectado o $nCUBE2$) envolvida com PVM versão 3, que configuram as estações como um ambiente computacional paralelo, isto é, uma máquina paralela virtual.

Apesar do fato de ambos os ambientes poderem ser representados por um mesmo modelo abstrato, há uma diferença básica entre os dois sistemas: enquanto o sistema com a rede *Ethernet* é baseado num canal de comunicação simples, o hipercubo oferece rotas alternativas para comunicação entre os nós. A principal conseqüência prática é que enquanto o hipercubo pode ser configurado para um número grande de nós, o uso efetivo

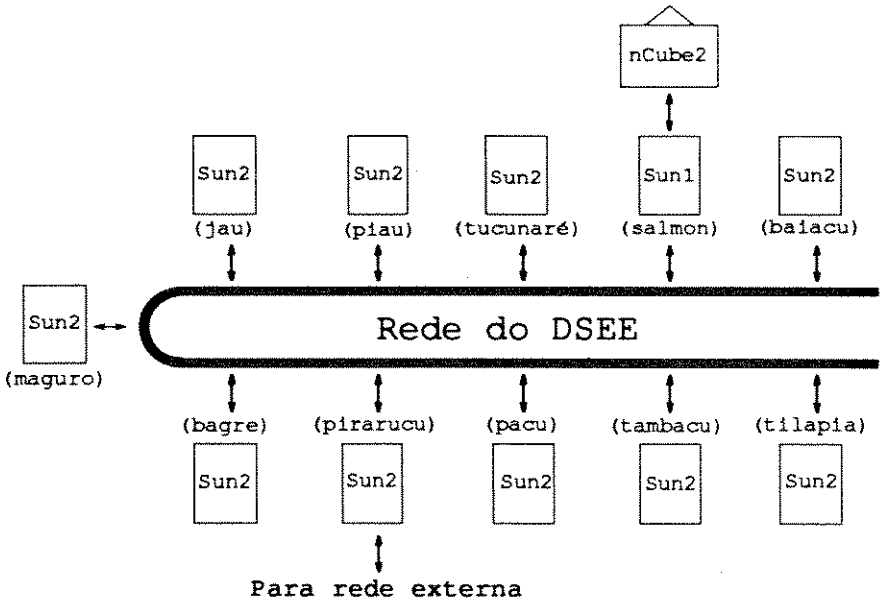


Figura 5.2: Ambiente de simulação

de uma rede de estações se limita a um número modesto de computadores, pelo menos para o tipo de aplicação desenvolvido neste trabalho.

O terceiro é uma rede heterogênea (*FDDI*) de estações *Sun Sparc10 e Classic* composta de nove máquinas envolvidas com *PVM* versão 3.

O quarto é um computador paralelo *SP*1 composto de oito nós, interligados através de *FDDI* e envolvidas com *PVM* versão 3.

O quinto é uma rede homogênea (*Ethernet*) de estações *IBM RISC*600 composta de oito máquinas envolvidas com *PVM* versão 3.

Todos os ambientes de simulação aqui utilizados são sistemas baseados no modelo *message passing.*

5.2.1 Arquitetura Hipercúbica

Hipercubos são máquinas fracamente acopladas configuradas numa rede binária de n-cubo e conhecidos pelos diferentes nomes (*cosmic cube, n-cube, binary n-cube, Boolean n-cube,*

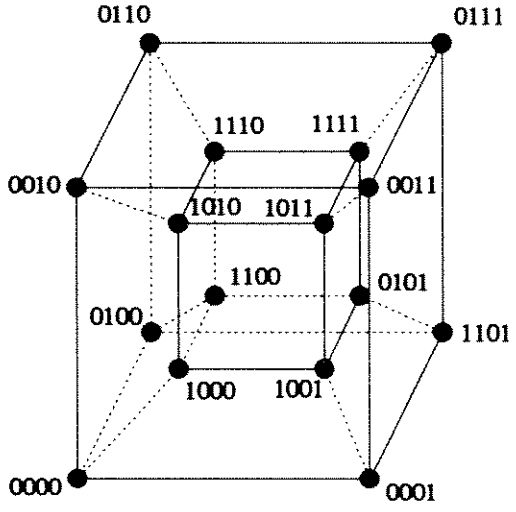


Figura 5.3: Hipercubo de 4 dimensões

etc.). Consistem de 2^n processadores idênticos, cada um com a sua própria memória e interligados com n nós vizinhos. Um cubo de terceira dimensão é um cubo ordinário de com 8 (2^3) vértices (nós). Genericamente, um *n-cube* pode ser construído como se segue: primeiro, 2^n nós são rotulados por 2^n números binários de 0 até $2^n - 1$, em seguida um *link* entre dois nós é puxado se e somente se os números binários diferirem de um bit. A primeira propriedade importante do *n-cube* é que ele pode ser construído recursivamente a partir de cubos de dimensões menores. Consideremos dois (n - 1)-cubos cujos vértices são enumerados de 0 até $2^n - 1$. Ligando-se todos os vértices do primeiro (n - 1)-cubo com os vértices do segundo, de mesmo número, obtém-se um n-cubo. Em seguida, basta renumerar os nós do primeiro (n - 1)-cubo por $0 \wedge a_i$ e os do segundo por $1 \wedge a_i$, onde a_i é um número binário representando dois nós similares dos (n - 1)-cubos e $\wedge$ denota a concatenação dos números binários [40]. A Figura 5.3 mostra um cubo de quarta dimensão.

A topologia tipo hipercubo apresenta uma grande vantagem na **escalonabilidade**. Oferece um melhor compromisso entre o caminho mais longo entre os nós e o número de ligações físicas requeridas para cada nó. Por exemplo, 16 processadores interligados formando um hipercubo de 4 dimensões cada nó terá 4 ligações. Assim, o caminho entre dois nós mais **distante** passará por 4 ligações. Para 256 processadores e uma topologia hipercúbica de 8 dimensões as ligações por nó serão 8, e o caminho mais longo terá 8 ligações. Comparando-se as duas configurações citadas, enquanto o número de processadores aumentou 16 vezes, tanto o número de ligações por nó como as ligações para o caminho mais longo aumentaram 2 vezes.

Considerando-se que a latência (atraso introduzido pela comunicação durante as trocas de informações entre os nós) é aproximadamente proporcional ao caminho mais longo, esta topologia apresenta um ótimo desempenho com relação a esta característica [47].

A arquitetura hipercúbica apresenta uma propriedade importante, que minimiza o *ovehead* de comunicação. Isto se explica pelo fato de que enquanto o número de *nós* aumenta de 2^n a distância máxima entre quaisquer dois *nós* do sistema cresce somente com n.

O $nCUBE2$ é um computador paralelo de memória distribuída (uma arquitetura $MIMD$, pela classificação de *Flynn*), em que cada nó é um computador completo, compreendendo as unidades de processamento e memória; este tipo de máquina é também conhecido como **multicomputador**, para diferenciá-lo de **multiprocessador**, que é normalmente de memória compartilhada. No $nCUBE2$ cada nó executa suas próprias instruções e processa dados armazenados na sua memória local e se comunica com os demais nós através da *message passing*. A máquina usada no presente trabalho permite a alocação de 1 a 64 nós (cubos de 1 a 6 dimensões). A Figura 5.4 mostra um hipercubo de 4 dimensões conectado a um computador *host* que faz parte de uma LAN.

O $nCUBE2$ consiste basicamente das seguintes partes:

- **Nós** - A base de um computador $nCUBE2$ é uma rede de UCPs independentes, cada um com a sua própria memória e *hardware* de comunicação. Cada nó possui uma cópia completa do sistema operacional nCX. O $nCUBE2$ pode rodar vários programas paralelos simultâneamente, alocando-se sub-cubos diferentes para cada programa. Cada nó é um computador seqüencial operando com 2,5 $Mflops$. O número de nós é escalonável, para possibilitar a resolução de problemas maiores ou reduzir o tempo de processamento de um determinado problema.

- **Canais de comunicação** - Cada nó é conectado aos nós vizinhos por canais de comunicação. Estes nós suportam a transferência de dados e controle de mensagens entre dois nós quaisquer a uma taxa de 2,2 $Mbytes$/segundo.

- **Processadores** I/O - Além dos canais de comunicação entre os nós, cada nó pode também ter um canal dedicado separado para enviar e receber mensagens através dos processadores I/O. Estes canais I/O operam na mesma taxa dos canais de comunicação entre os nós. Os processadores I/O possuem a mesma UCP e configuração de memória dos nós, porém com *hardware* e *software* adicionais de interface aos discos, monitores e outros periféricos. Da mesma forma que os nós o número de processadores I/O é **escalonável**, possibilitando que as aplicações tirem vantagens da capacidade de I/O paralelo no $nCUBE2$.

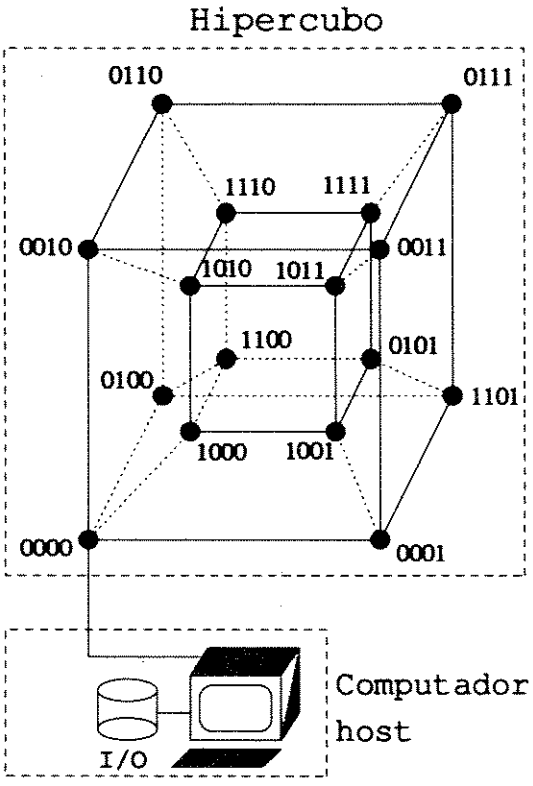


Figura 5.4: Hipercubo de 4 dimensões conectado a um *host*

No momento estão disponíveis no mercado hipercubos de até 8192 (2^{13}) nós. Cada nó consiste de um processador e uma memória com capacidade de endereçar de 1 a 64 Mbytes.

Cada processador consiste de:

- **Uma UCP de propósito-geral de 64-bit** incluindo;
  1. uma ID/ICU (Decodificador de Instrução e Controladora de Interupção)
  2. uma EU (Unidade de Execução)
  3. um operando FIFO/cache
- **Uma MMU (unidade gerenciadora de memória)**
- **Uma NCU (unidade de rede comunicação)**

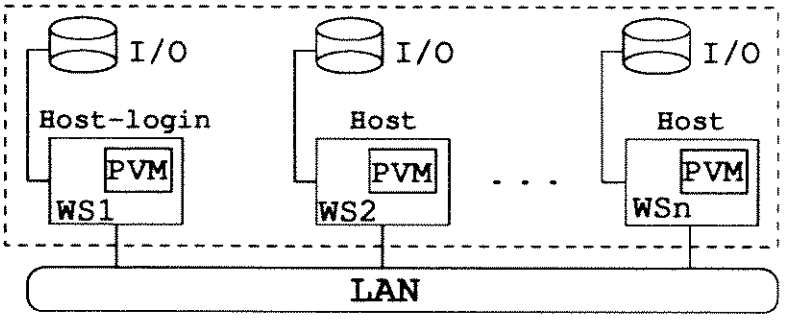


Figura 5.5: Rede de estações como uma máquina paralela virtual sob *PVM* 3.

A comunicação entre processadores é efetuada via mensagens. A transmissão de mensagens ocorre obedecendo a três estágios: **Criação do Caminho**, **Transmissão de Dados** e **Remoção do Caminho**. Cada estágio é executado por vez por cada processador no caminho da mensagem.

A arquitetura da *NCU* tem três camadas: a **Camada de Interligação**, a **Camada da Rota** e a **Camada da Mensagem** . A **Camada de Interligação** estabelece, ao *hardware* requerido, *links* físicos de comunicação. A **Camada da Rota** estabelece um arbitramento e lógica de chaveamento para criação, manutenção e remoção de caminhos de comunicação entre processadores na rede. A **Camada da Mensagem** estabelece uma transferencia, confiável e eficiente de dados ponto a ponto entre processadores.

5.2.2 *PVM* - Máquina Paralela Virtual

A Figura 5.5 mostra uma rede local de estações de alto desempenho que podem ser configurada para ser usada como computador paralelo *virtual*, quando provida com um sistema de *software* apropriado como o *PVM* (*Parallel Virtual Machine*). O sistema *PVM* foi desenvolvido no *Oak Ridge National Laboratory*, a partir de 1989, para explorar mais efetivamente os recursos das redes de computadores. Apesar de o sistema *PVM* suportar uma rede heterogênea, composta por uma variedade muito grande de computadores (conectados por *WAN e LAN*), neste trabalho foram utilizadas redes locais de estações interligadas via *Ethernet* ou *FDDI*. Sob o sistema *PVM*, um conjunto de computadores (seqüenciais, paralelos e vetoriais) definidos pelo usuário se comporta como um grande computador de memória distribuída, definido como **máquina virtual**. Cada um dos computadores que compõe este conjunto é denominado *host*. Uma **tarefa** (*task*) é definida como uma unidade de computação no *PVM*, análogo a um processo *UNIX*.

O *PVM* fornece as funções que automaticamente colocam em funcionamento as tarefas na *máquina virtual* e permitem que as tarefas se comuniquem e se sincronizem reciprocamente. As aplicações, codificadas em *Fortran*77 ou *C*, podem ser paralelizadas usando o modelo *message passing* [50].

O sistema *PVM* é composto de duas partes. A primeira parte é um *daemon*, chamado *pvmd3* (abreviado por *pvmd*), residente em todos os *hosts* (um exemplo de um programa *daemon* é o *sendmail* que gerencia o correio eletrônico no sistema *UNIX*). Quando da necessidade do sistema *PVM* o usuário, inicialmente, cria uma **máquina virtual** configurando o *PVM*. A aplicação pode, então, ser iniciada no *prompt* do *UNIX* de qualquer um dos *hosts* Uma característica importante do sistema *PVM* é que vários usuários podem configurar *máquinas virtuais* simultaneamente, cada um executando sua aplicação.

A segunda parte do sistema é biblioteca de rotinas de interface do *PVM* (*libpvm3.a*). Estas rotinas podem ser chamadas pelo usuário para *message passing*, desova de processos, coordenação de tarefas e modificações da *máquina virtual*. Programas aplicativos devem ser *linkados* com esta biblioteca para usar o sistema *PVM*.

Características do *PVM*

- **Identificador de Tarefas**

  Todos os processos do *PVM* 3 são identificados por um número inteiro denominado *tid*. O *tid* é fornecido pelo *pvmd* local e é único para cada processo.

- **Interface com Usuário**

  Na versão *PVM* 3 o nome das rotinas começam com *pvm_* para *C* e *pvmf* para *Fortran*.

- **Controle de Processos**

  O sistema *PVM* dispõe de rotinas que habilitam um processo do usuário a tornar uma tarefa do *PVM* e voltar novamente a ser um processo normal. Existem rotinas que adicionam e retiram *hosts* na **máquina virtual**, rotinas que iniciam e encerram tarefas do *PVM*, rotinas que enviam sinais para outras tarefas do *PVM* e rotinas que descobrem informações a respeito da configuração da **máquina virtual** e tarefas ativas no *PVM*.

- **Tolerância à Falha**

  Na eventualidade da falha de um *host*, o *PVM* detecta automaticamente esta condição, excluindo-o da *máquina virtual*. O estado dos *hosts* pode ser requisitado pela aplicação e caso seja solicitado um *host* pode ser adicionado pela aplicação.

É ainda da responsabilidade de quem desenvolve a aplicação fazê-lo tolerante à falha do *host*. O *PVM* não recupera automaticamente as tarefas que são canceladas decorrente da falha de um *host*. Esta característica permite também que *hosts* sejam adicionados conforme as suas disponibilidades (por exemplo, no fim de semana, ou quando a aplicação determina dinamicamente a necessidade de maiores recursos computacionais).

- **Grupos Dinâmicos de Processos**

  Nesta implementação, um processo pode pertencer a vários grupos e pode mudar dinamicamente durante o processamento. Funções que tratam com grupos de tarefas, tais como transmissão (*broadcast*) e barreira (*barrier*) utilizam como argumentos os nomes de grupos explicitamente definidos pelo usuário.

- **Sinalização**

  O *PVM* fornece duas maneiras de sinalizar outras tarefas do *PVM*. Um dos métodos envia um sinal *UNIX* para outra tarefa. O outro notifica uma tarefa com respeito a um evento enviando uma mensagem, com uma etiqueta definido pelo usuário, que a aplicação pode verificar. Várias notificações de eventos são disponíveis no *PVM* 3, incluindo o abandono de uma tarefa, a exclusão (ou falha) de um *host* e a adição de um *host*.

- **Comunicação**

  PVM dispõe de rotinas para empacotamento e envio de mensagens entre tarefas. O modelo permite que qualquer tarefa envie uma mensagem para qualquer outra tarefa *PVM*, não havendo limite de tamanho ou número de tais mensagens. Enquanto todos os *hosts* têm limitações físicas de memória (espaço no *buffer*), o modelo de comunicação não se restringe às limitaçoes de uma máquina particular e assume que dispõe de memória suficiente. O modelo de comunicaçao *PVM* proporciona bloqueio do envio assíncrono, bloqueio da recepção assíncrona e funções de recepção não-bloqueios.

- **Integração de *hosts***

  O sistema *PVM* permite a integração de sistemas computacionais heterogêneas interligadas por redes.

Interface com usuário

O sistema *PVM* é dotado de várias rotinas que são chamadas pelo usuário. Estas rotinas servem como interfaces padrões que suportam modelos de programação concorrente

comum na forma de primitivas bem definidas que são embutidas nas linguagens procedurais (*C ou Fortran*), utilizadas nas aplicações. Estas primitivas podem ser classificadas nos seguintes grupos:

- Controle de processo
- Informação
- Configuração dinâmica
- Sinalização
- Ajustes e escolha de opções
- *Message passing*

Estas rotinas permitem a programação da **máquina virtual** basicamente da mesma forma que no computador paralelo, entretando, com uma diferença: as primitivas da **máquina virtual** são de nível mais baixo se comparadas com as primitivas do computador paralelo.

5.2.3 Arquitetura Heterogênea

O *PVM* pode ser implementado em *hardwares* com várias arquiteturas diferentes, desde computadores monoprocessadores, computadores vetoriais e **multicomputadores**. Estas máquinas interligadas por uma ou mais redes, podendo ser diferentes entre sí (*Ethernet*, *Internet* e *FDDI*), são acessadas pelas aplicações por meio de uma interface padrão que suporta paradigmas comuns de processamento paralelo na forma de primitivas bem definidas que são embutidas nas linguagens procedurais (*C* ou *Fortran*). Assim, uma arquitetura heterogênea funciona como um único recurso computacional concorrente, conforme ilustrado na Figura 5.6.

No processamento distribuído utilizando o *PVM* os programas aplicativos devem ser decompostos em sub-tarefas com nível de granularidade moderadamente grande, ou seja, as sub-tarefas devem requerer um tempo de computação da parte paralela relativamente elevado em comparação com o tempo gasto na comunicação entre as sub-tarefas.

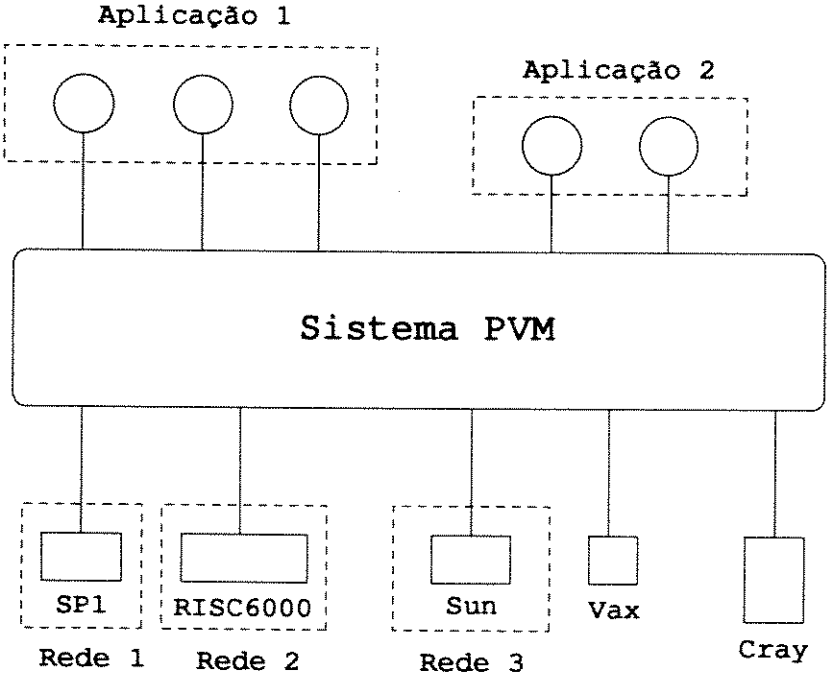


Figura 5.6: Arquiteturas diversas no ambiente *PVM*

O *nCUBE2* é uma das poucas exceções no qual não permite configurar o *PVM*. Nesta Subseção é discutida uma forma de incluir o *nCUBE2* como parte da máquina virtual paralela, utilizando-se de recursos do *Unix*. Como resultado tem-se um sistema no qual

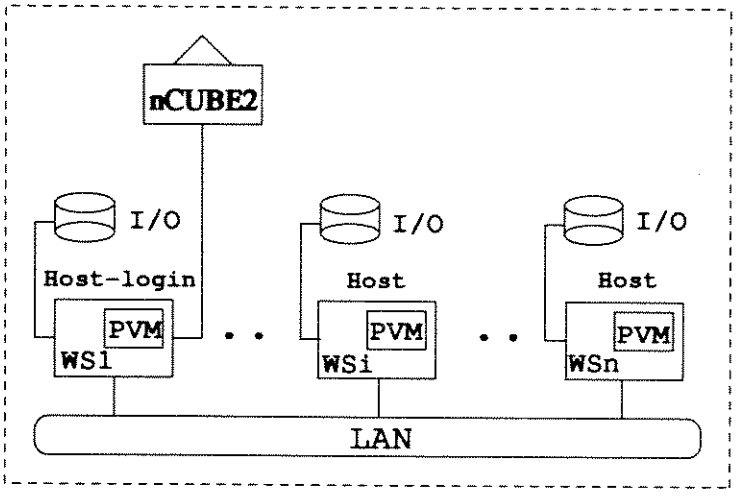


Figura 5.7: Máquina paralela virtual

é permitida a distribuição de tarefas com diferentes combinações de nós do $nCUBE2$ e computadores da rede de estações. A Figura 5.7 mostra a arquitetura desta máquina paralela virtual.

Nesta arquitetura o subcubo configurado é dedicado para a simulação, enquanto que as máquinas da estação podem compartilhar os recursos com outras aplicações. Desta forma, a parte que cabe à estação pode ser afetada pelas condições de carregamentos das $UCPs$ e da rede.

Balanceamento das cargas

Em um ambiente com recursos compartilhados é muito importante o balanceamento das cargas. Os esquemas mais comuns de balanceamento de cargas são:

estático: Neste esquema as tarefas são divididas à priori ($off-line$) antes de a aplicação ser iniciada, tendo como critério a divisão por potência computacional das máquinas. Este esquema tem bom desempenho em redes vazias e moderadamente ocupadas.

***pool* de tarefas:** Este esquema é aplicado quando as cargas são variáveis.

A Figura 5.8 ilustra o esquema de programação para o esquema *pool* de tarefas.

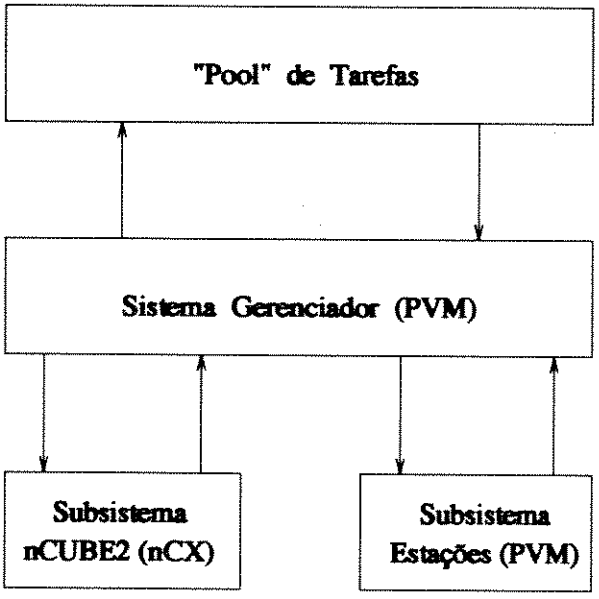


Figura 5.8: Esquema *pool* de tarefas

5.3 Modelos de Programação

5.3.1 Partes do Programa Seqüencial

O programa de análise probabilística de curto-circuito é subdividido em três partes principais:

1. A parte **essencialmente seqüencial**, onde são lidos os dados (acesso a disco) e são chamadas todas as rotinas de preparação de dados (fatoração de matrizes, inclusão dos acoplamentos mútuos, técnicas de vetores esparsos, cálculos de impedâncias de curtos-circuitos) para cálculos de curtos-circuitos.
2. A parte **paralelizável**, onde são executados os **cálculos intensivos**. Nesta parte devem ser destacadas três sub-tarefas importantes:
   a. a chamada da rotina para geração de números aleatórios (**inicializada** com uma semente adequada).
   b. cálculos das correntes de curtos-circuitos (monofásico e trifásico).
   c. obtenção das freqüências dos valores de curtos-circuitos.

3. Geração de histogramas das funções densidade de probabilidade dos valores de curtos-circuitos e saída de resultados em arquivos. Os histogramas são gerados em escala e de forma normalizada.

Este código seqüencial foi paralelizado para os ambientes citados na Seção 5.2, utilizando dois modelos de programação: *(1) SPMD* e *(2) Mestre/Escravo.*

5.3.2 Modelo de Programação *SPMD*

Neste modelo, também denominado de descentralizado, o computador *host* (ou *host-login* no caso do *PVM*) lê os dados de entrada no disco e os repassa para todos os nós (ou *hosts* no caso do *PVM*). Os nós (ou *hosts*) executam simultaneamente a parte essencialmente seqüencial do programa, que consiste na preparação dos dados. Terminada esta etapa inicia-se a de computação explicitamente paralela, onde cada nó (ou *host*) recebe as tarefas para a execução de cálculos intensivos. Os resultados de cada nó (ou *host*) são totalizados no nó 0 (ou *host-login*), onde é também preparada a saída de resultados. Uma característica deste modelo de implementação paralela é que um único programa é executado igualmente em todos os processadores e as operações de *I/O* são realizadas apenas pelo computados *host* no caso do *nCUBE* e pelo *host-login* no cado do *PVM*.

Modelo *SPMD* para o *nCUBE2* com *nglobal/nlocal*

O *nglobal* faz com que todos os nós sejam habilitados para a leitura de dados nos seus respectivos discos. Na configuração atual os nós do *nCUBE2* não possuem discos, assim sendo, somente o nó 0 lê os dados através do computador *host*. Todos os nós do sub-cubo definido *(0, 1, 2, . . , n-1)* recebem os dados lidos pelo *host* e a cópia do programa. A preparação dos dados é feita concomitantemente por todos os nós, inclusive pelo nó 0. De posse dos dados preparados, cada nó executa a parte dos cálculos de curtos-circuitos que lhe cabe. Terminados os cálculos cada nó dispõe das freqüências dos valores das potências de curtos-circuitos que são enviadas ao nó 0, onde são totalizados. Os arquivos de saída no disco são preparados no computador *host*.

A Figura 5.9 mostra a estrutura básica do modelo de programação *SPMD* no *nCUBE* com *nglobal/nlocal*.

A Figura 5.10 ilustra o fluxograma do modelo *SPMD* no *nCUBE* com *nglobal/nlocal*. Observe-se que a leitura de dados do sistema elétrico só é executada no nó 0, através do

```
whoami()
     ! habilita todos os nos a lerem os dados nos discos
     nglobal
       le dados do sistema eletrico no disco do host
       os dados sao preparados concomitantemente em todos os nos
     nlocal
       obtem-se a semente baseado no mynode
       calcula os curtos-circuitos
     ! envia os resultados para o no 0
     nisumn()
if( mynode .eq. 0) THEN
     totaliza os resultados de todos os nos
     prepara os arquivos de saida
ENDIF
end
```

Figura 5.9: Modelo *SPMD* no *nCUBE2* com *nglobal/nlocal*.

computador *host*.

Sub-rotinas do *nCUBE2* utilizadas no modelo *SPMD* com *nglobal/nlocal*

- *whoami (mynode, mypid, myhost, ndim)*

mynode: número relativo do nó corrente; este número sempre fica entre zero e a quantidade de nós do sub-cubo menos um.

mypid: os 16 bits inferiores contém o número relativo do nó corrente; os 16 bits superiores contém o ID do processo corrente.

myhost: o ID do *host*; o *host* é usado como destino e fonte das mensagens trocadas com o programa *host*.

ndim: a dimensão do sub-cubo corrente; o número de nós de um sub-cubo é 2^{ndim}.

whoami retorna informações a um nó processador de um sub-cubo, incluindo o número relativo do nó (ID lógico), número relativo do nó processador do processo chamado, o ID do *host* e a dimensão do sub-cubo.

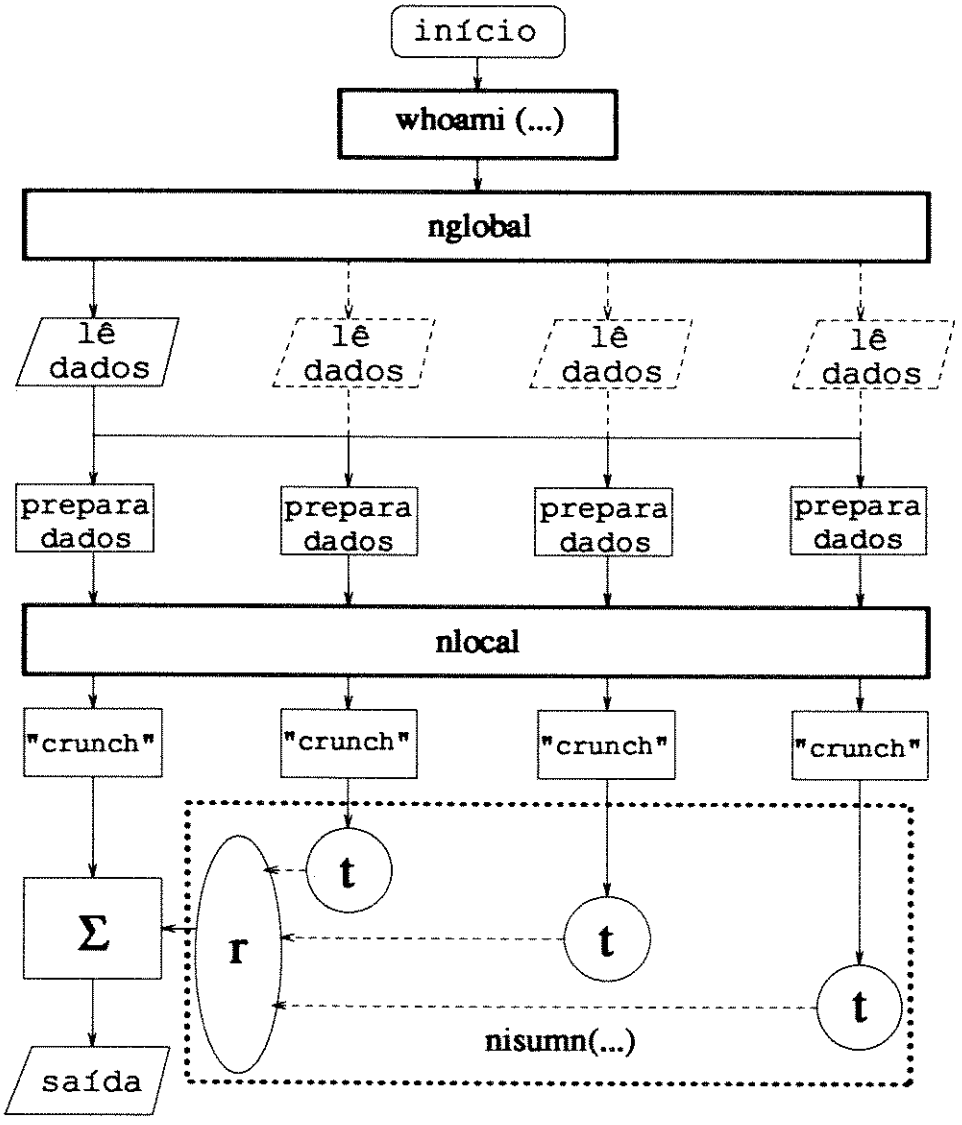


Figura 5.10: Fluxograma do modelo *SPMD* no *nCUBE* com *nglobal/nlocal*

- *nglobal*

nglobal ajusta o I/O para o modo global. No modo global, todos os nós devem fazer a chamada do mesmo I/O. Todos os processadores obtém os mesmos dados de entrada; a saída é realizada somente pelo nó 0. O modo global é necessário quando da leitura dos parâmetros de entrada.

- *nlocal*

nlocal ajusta o I/O para o modo local. No modo local, todos os nós podem fazer a chamada do I/O independentemente. Isto é necessário, por exemplo, se um determinado nó encontra um erro e precisa imprimir uma mensagem. Entretanto, o modo local geralmente não é requerido quando se lê dados ou parâmetros no modo global.

nglobal e *nlocal* só podem ser chamados em programas carregados de forma homogênea (o mesmo programa carregado em todos os nós do sub-cubo configurado). As chamadas de *nglobal* e *nlocal* devem ser sincronizadas; isto é, *nglobal* e *nlocal* devem ser chamadas por todos os nós do sub-cubo configurado e não simplesmente por um único nó ou alguns dos nós do sub-cubo.

- *nisumn (isend, nveclen, target, msgtyp, mask)*

isend: contém o vetor da contribuição do nó corrente para o resultado final.

nveclen: contém o número de elementos do vetor.

target: determina o nó que recebe os resultados.

msgtyp: toda a comunicação no *nisumn* é feita usando tipo de mensagem *msgtyp*.

mask: o valor do *mask* determina quais nós do sub-cubo corrente participam da soma (se *mask* = *-1*, todos os nós participam).

nisumn efetua a comunicação dos resultados de uma somatória global, ou conjunto de vetores inteiros de quatro *bytes*, cujas contribuições vêm de todos os nós do sub-cubo corrente.

Modelo *SPMD* para o *nCUBE2* com *nbroadcast*

O computador *host* se encarrega da leitura dos dados no disco, de um arquivo do sistema elétrico. Todos os nós do sub-cubo definido *(0, 1, 2, . . , n-1)* recebem os dados lidos pelo *host* e a cópia do programa. Neste modelo, o processamento da parte seqüencial do programa, que é composta por chamadas de sub-rotinas de preparação de dados, é feito concomitantemente por todos os nós, inclusive pelo nó 0.

De posse de dados preparados necessários para os cálculos de curtos-circuitos (que se constituem na repetição da avaliação das intensidades das potências de curtos-circuitos milhares de vezes), cada nó executa a parte que lhe cabe no processo global dos cálculos intensivos (cada nó processa uma fração do número total de amostras). Ao terminar a sua tarefa, cada nó dispõe das freqüências dos valores das potências de curtos-circuitos. Estes resultados parciais são enviados e totalizados no nó 0. Finalmente, este nó repassa o resultado totalizado ao computador *host* que prepara os arquivos de saída no disco.

A Figura 5.11 mostra a estrutura básica do modelo de programação *SPMD* no *nCUBE*.

A Figura 5.12 ilustra o fluxograma do modelo *SPMD* no *nCUBE* com *nbroadcast*.

Sub-rotinas do *nCUBE2* utilizadas no modelo *SPMD* com *nbroadcast*

- *whoami (mynode, mypid, myhost, ndim)*

mynode: número relativo do nó corrente; este número sempre fica entre zero e a quantidade de nós do sub-cubo menos um.

mypid: os 16 bits inferiores contém o número relativo do nó corrente; os 16 bits superiores contém o *ID* do processo corrente.

myhost: o *ID* do *host*; o *host* é usado como destino e fonte das mensagens trocadas com o programa *host*.

ndim: a dimensão do sub-cubo corrente; o número de nós de um sub-cubo é 2^{ndim}.

whoami retorna informações a um nó processador de um sub-cubo, incluindo o número relativo do nó (*ID* lógico), número relativo do nó processador do processo chamado, o *ID* do *host* e a dimensão do sub-cubo.

```
whoami()
if( mynode .eq. 0) THEN
     le dados do sistema eletrico no disco
     ! transmite dados para outros nos do subcubo
     nbroadcast()
     ...
  ELSE
     ! recebe dados do no 0
     nread() ! le dados no buffer
     ...
ENDIF
     prepara dados
     obtem-se a semente baseado no mynode
     calcula curtos-circuitos
if( mynode. ne. 0) THEN
     ! envia resultados para no' 0
     nisumn()
  ELSE
     recebe resultados de outros nos do subcubo e totaliza
     prepara os arquivos de saida
ENDIF
end
```

Figura 5.11: Modelo *SPMD* no nCUBE2 com *nbroadcast*.

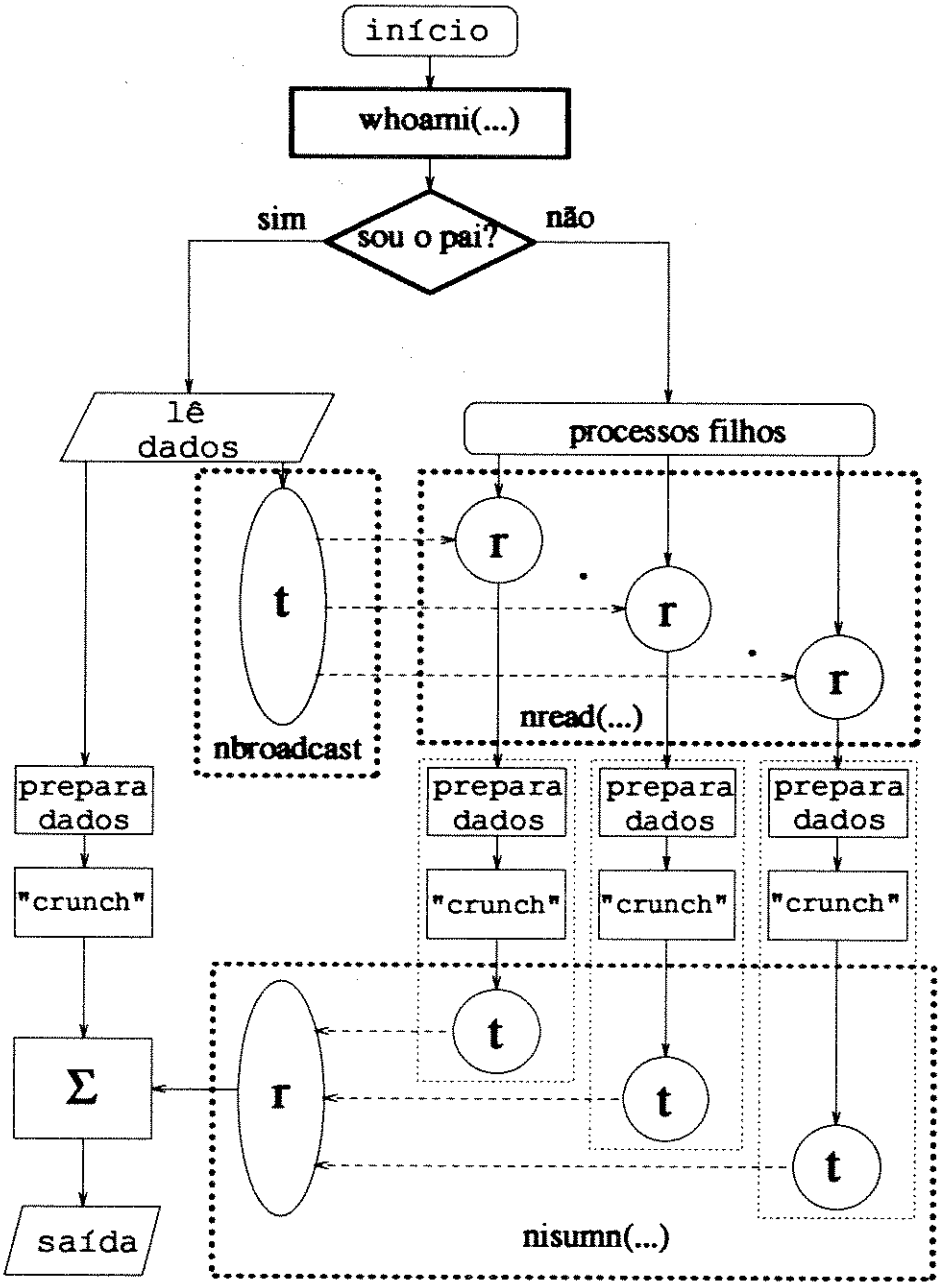


Figura 5.12: Fluxograma do modelo *SPMD* no *nCUBE* com *nbroadcast*

- *nbroadcast (buf, nbytes, nofonte, msgtyp, mask)*

buf: endereço no nó de origem da mensagem.

nbytes: comprimento em *bytes* da mensagem a ser transmitida.

nofonte: nó de origem da mensagem.

msgtyp: argumento que seleciona o tipo de mensagem a ser usado em todos os *nbroadcast*.

mask: determina qual porção do subcubo corrente irão receber a mensagem (se *mask* = *-1*, todos os nós irão receber)

nbroadcast transmite mensagens de um nó para todos os nós do subcubo configurado ou para um subcubo completo dentro do subcubo configurado.

Para que o *nbroadcast* tenha sucesso as seguintes condições devem ser satisfeitas:

1. Cada vizinhança dos nós do hipercubo que tem os canais selecionados pelo argumento *mask* deve chamar o *nbroadcast*.
2. Os argumentos *nofonte*, *msgtyp* e *mask* devem ser os mesmos em todas as chamadas do *nbroadcast*.

- *read (buf, nbytes, nofonte, msgtyp, mask)*

buf: local onde os dados da mensagem está armazenado.

nbytes: comprimento em *bytes* da mensagem a ser lida.

nofonte: nó de origem da mensagem.

msgtyp: argumento que seleciona o tipo de mensagem a ser lido.

mask: argumento não utilizado

nread copia uma mensagem do *buffer* para um endereço especificado.

Modelo *SPMD* para o *PVM*

O computador *host-login* (que também faz parte da máquina virtual) lê o arquivo de dados do sistema elétrico. Estes dados são enviados para todos os computadores da máquina virtual que possuírem processos desovados a partir do processo chamado *pai* (isto é, aquele que é executado no *host-login*). O processo **pai** executa tarefas idênticas àquelas executadas pelos processos **filhos**, além de ler os dados de entrada e preparar a saída de resultados. Neste modelo, todos os processos (**pai** e **filhos**) alocados nos *hosts* executam concomitantemente a parte essencialmente seqüencial do programa, que consiste fundamentalmente na preparação dos dados.

De posse dos dados preparados, necessários para os cálculos de curtos-circuitos (que se constituem na repetição da avaliação das intensidades das potências de curtos-circuitos milhares de vezes), cada *host* executa a parte que lhe cabe no processo global dos cálculos intensivos (cada *host* processa uma fração do número total de amostras). Ao terminar a sua tarefa, cada *host* dispõe das freqüências dos valores das potências de curtos-circuitos. Estes resultados parciais são enviados ao processo **pai** que os totaliza e prepara os arquivos de saída, conseqüentemente acessível no disco do *host-login*.

A Figura 5.13 mostra a estrutura básica do modelo de programação *SPMD* no *PVM*.

A Figura 5.14 ilustra o fluxograma do modelo *SPMD* no *PVM*.

Sub-rotinas do *PVM* utilizadas no modelo *SPMD*

1. Sub-rotinas para controle do processo

   - *pvmfmytid(mytid)*

   mytid: inteiro que identifica a tarefa do processo *PVM*.

   pvmfmytid registra o processo corrente no *PVM* na sua primeira chamada e gera um único *mytid* se este processo não foi criado pelo *pvmfspawn*. *pvmfmytid* retorna o *mytid* do processo chamado e pode ser chamado várias vezes na aplicação. *mytid* é um inteiro positivo de 32 bit criado pelo *daemon* (*pvmd*).

   - *pvmfspawn (task, flag, where, ntask, info)*

   task: nome do programa executável, identificado por um *string* de caráteres.

   flag: inteiro que especifica as opções da desova.

```
pvmfmytid()
if( mynode .eq. 0) THEN
     le dados do sistema eletrico no disco
     ! transmite dados para hosts do PVM
     pvmfspawn()
     pvmfinitsend()
     pvmfpack()
     ...
     pvmfmcast()
  ELSE
     ! recebe dados do host-login do PVM
     pvmfrecv()
     pvmfunpack()
     ...
ENDIF
     prepara dados
     obtem-se a semente baseado no mynode
     calcula curtos-circuitos
if( mynode .ne. 0) THEN
     envia resultados para no 0
  ELSE
     recebe resultados dos hosts do PVM e totaliza
     prepara os arquivos de saida
ENDIF
end
```

Figura 5.13: Modelo *SPMD* no *PVM*

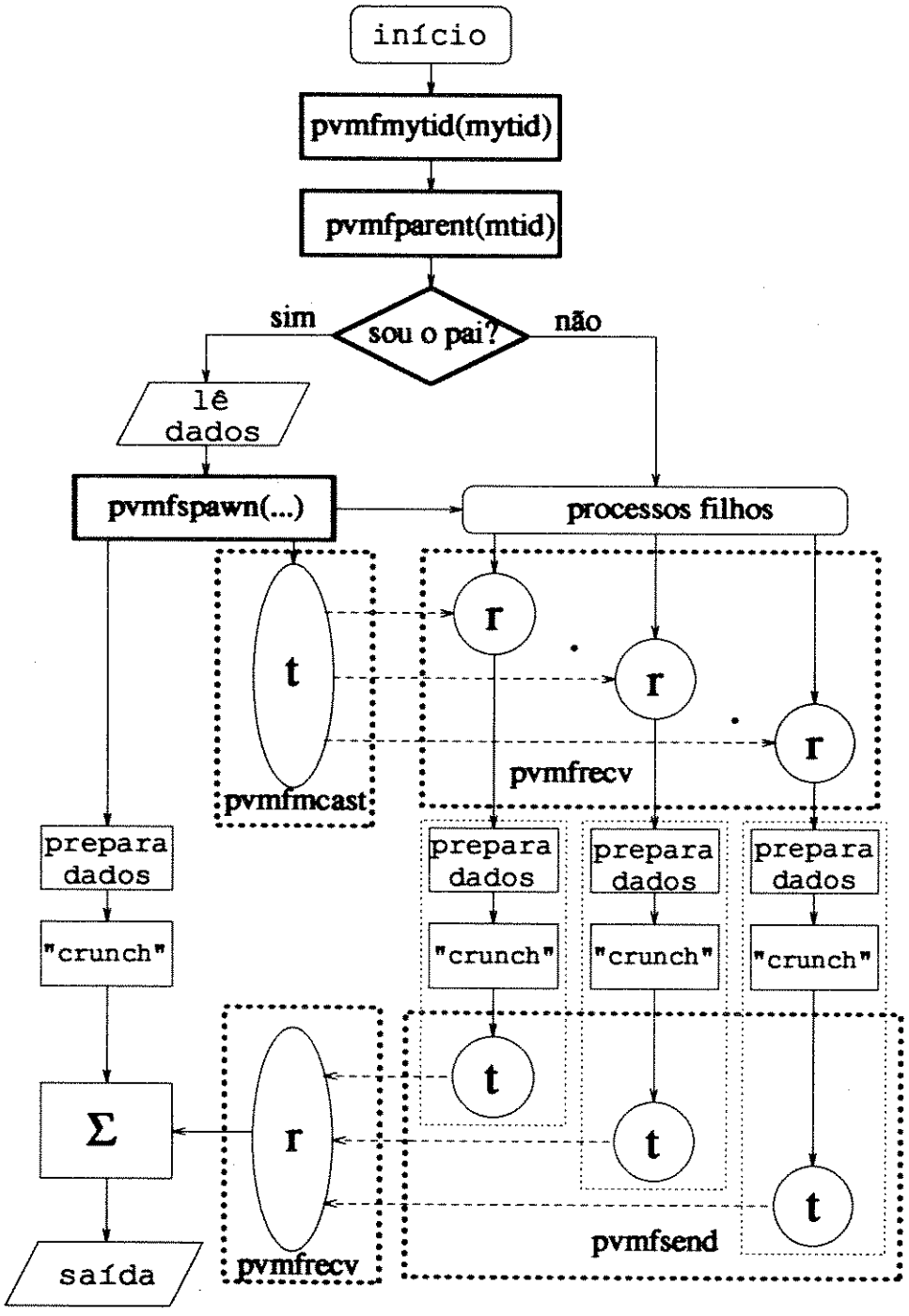


Figura 5.14: Fluxograma do Modelo *SPMD* no *PVM*

where: *string* de caráteres que especifica onde **inicializar** processo PVM. Dependendo do valor do *flag* , *where* pode ser o nome de um *host* ou uma classe da arquitetura PVM. Se *flag* for 0, então *where* é ignorado e o PVM irá selecionar o *host* mais apropriado.

ntask: inteiro que especifica o número de cópias do programa executável.

info: inteiro que retorna o número de tarefas **inicializadas**.

pvmfspawn **inicializa** *ntask* cópias do programa executável identificado como *task*.

- *pvmfexit (info)*

info: inteiro que indica um erro se o seu valor for menor que zero.

pvmfexit informa o *daemon* que este processo está deixando o PVM

2. Sub-rotinas para Informação

   - *pvmfparent(mtid)*

mtid: inteiro que identifica a tarefa do processo *pai*.

pvmfparent retorna o *mtid* do processo que desovou o processo chamado.

- *pvmfconfig(nhost, narch, dtid, name, arch, speed, info)*

nhost: inteiro que retorna o número de *hosts* (pvmds) na máquina virtual

narch: inteiro que retorna o número de diferentes formatos de dados usados

dtid: inteiro que retorna o ID da tarefa do *host*

name: *string* de caráteres que retorna o nome do *host*

arch: *string* de caráteres que retorna o nome da arquitetura do host

speed: inteiro que retorna a velocidade relativa do host

info: inteiro que retorna um número código, que informa se *pvmfconf* foi bem sucedido ou não

pvmfconf retorna as informações a respeito da configuração da máquina virtual.

- *pvmftidtohost(tid, dtid)*

tid: inteiro que identifica a tarefa especificada

dtid: inteiro que identifica o *host*.

pvmftidtohost retorna o ID do *host* no qual a tarefa especificada está sendo processado.

3. Sub-rotinas para *Message Passing*

No PVM o envio de uma mensagem é composto por três etapas. Primeiramente, um *send buffer* deve ser **inicializado** chamando a sub-rotina *pvmfinitsend()*. Em seguida, a mensagem deve ser **empacotada** neste $buffer$ pela sub-rotina *pvmfpack()*. Finalmente, a mensagem é enviada para outro processo pela sub-rotina *pvmfsend()* ou para vários processos pela sub-rotina *pvmfmcast()*. A mensagem é recebida pela chamada da sub-rotina de recepção *pvmfrecv()* (*blocking receive*) e em seguida **desempacotada** pela sub-rotina *pvmfunpack()*.

- *pvmfinitsend(encoding, bufid)*

encoding: inteiro que especifica o próximo esquema de codificação da mensagem.

$bufid$: inteiro que identifica o $buffer$.

pvmfinitsend limpa o *send buffer* e prepara para "empacotar" uma nova mensagem.

- *pvmfpack(what, xp, nitem, stride, info)*

what: inteiro que especifica o tipo de dado a ser **empacotado**.

xp: apontador do começo de um bloco de *bytes*.

nitem: numero total de itens a ser **empacotado** (não é o número de *bytes*).

stride: o passo a ser usado quando **empacota** os itens.

$info$: valor menor que 0 indica um erro.

pvmfpack **empacota** os dados a serem enviados.

- *pvmfmcast(ntask, tids, msgtyp, info)*

ntask: inteiro que especifica o número de tarefas a serem enviadas.

tids: vetor de inteiros com comprimento de pelo menos *ntask*, contendo IDs das tarefas a serem enviadas.

msgtyp: inteiro fornecido pelo usuário, devendo ser maior ou igual a 0.

$info$: valor menor que 0 indica um erro.

pvmfmcast envia uma mensagem armazenada num *send buffer* ativo para *ntask* tarefas especificados no vetor *tids*.

- *pvmfrecv(tid, msgtyp, bufid)*

tid: inteiro que identifica a tarefa do processo enviado, fornecido pelo usuário.
msgtyp: inteiro fornecido pelo usuário, devendo ser maior ou igual a 0.
bifid: inteiro que que retorna o valor do identificador do novo *receive buffer* ativo.

pvmfrecv recebe uma mensagem.

- *pvmfunpack(what, xp, nitem, stride, info)*

what: inteiro que especifica o tipo de dado a ser **desempacotado**.
xp: apontador do começo de um bloco de *bytes*.
nitem: numero total de itens a ser **desempacotado** (não é o número de *bytes*).
stride: o passo a ser usado quando **empacota** os itens.
info: valor menor que 0 indica um erro.

pvmfpack **desempacota** o *message buffer* ativo.

5.3.3 Modelo de Programação Mestre/Escravo

Este modelo, também conhecido como centralizado, se compõe de duas partes:

O programa principal, denominado de mestre, encarrega-se da preparação dos dados para os cálculos intensivos a partir dos dados de entrada lidos no disco do computador *host* (ou *host-login*) e é preparado para comunicar dados. O programa mestre é executado no nó 0 (ou *host-login*).

A rotina de computação intensiva, que pode ser um programa ou sub-rotina, executada nos demais nós (ou *hosts*) e é habilitada para receber dados preparados e enviados pelo mestre, a partir dos quais são executados os cálculos intensivos. Esta rotina é denominada escravo e é reproduzida tantas vezes quanto o processo mestre determinar.

Modelo Mestre/Escravo para o *nCUBE2*

O nó 0 (com o seu processo mestre) por intermédio do computador *host* encarrega-se da leitura do arquivo dos dados do sistema elétrico, no disco, e a partir deles prepara os dados relevantes para os cálculos intensivos. A etapa de preparação dos dados corresponde ao processamento da parte essencialemente seqüencial, que é executada apenas no nó 0. Os dados preparados são enviados para todos os outros nós que executam a parte intensiva dos cálculos. Cada nó (com seu processo escravo), após receber estes dados, inicia a execução dos cálculos que lhe cabe. Ao término desta tarefa cada nó dispõe das freqüências dos valores das potências de curtos-circuitos. Estes resultados parciais são enviados para o nó 0, onde são totalizados e preparadas as saídas em arquivo, para o obtenção do histogramas completos. Deve ser ressaltado que, neste modelo, o nó 0 fica ocioso, no intervalo de tempo compreendido entre o envio de dados preparados e a recepção dos resultados parciais.

A Figura 5.15 mostra a estrutura básica do modelo de programação Mestre/Escravo no *nCUBE*.

A Figura 5.16 ilustra o fluxograma do modelo Mestre/Escravo no *nCUBE*.

Sub-rotinas do *nCUBE2* utilizadas no modelo Mestre/Escravo

- ***whoami (mynode, mypid, myhost, ndim)***

mynode: número relativo do nó corrente; este número sempre fica entre zero e a quantidade de nós do sub-cubo menos um.

mypid: os 16 bits inferiores contém o número relativo do nó corrente; os 16 bits superiores contém o *ID* do processo corrente.

myhost: o *ID* do *host*; o *host* é usado como destino e fonte das mensagens trocadas com o programa *host*.

ndim: a dimensão do sub-cubo corrente; o número de nós de um sub-cubo é 2^{ndim}.

whoami retorna informações um nó processador de um sub-cubo, incluindo o número relativo do nó (*ID* lógico), número relativo do nó processador do processo chamado, o *ID* do *host* e a dimensão do sub-cubo.

- ***nbroadcast (buf, length, bnode, type, mask)***

```
whoami()
if( mynode .eq. 0) THEN
     le dados do sistema eletrico no disco
     prepara dados
     ! transmite dados preparados para outros nos
     nbroadcast()
     ...
     ! aguarda resultados de outros nos do subcubo
     ! recebe resultados de outros nos do subcubo
     nread()
     ...
     prepara os arquivos de saida
   ELSE
     ! recebe dados preparados do no 0
     nread() ! le dados no buffer
     ...
     obtem-se a semente baseado no mynode
     calcula curtos-circuitos
     !envia resultados para no 0
     nwrite()
     ...
ENDIF
end
```

Figura 5.15: Modelo Mestre/Escravo no *nCUBE2*

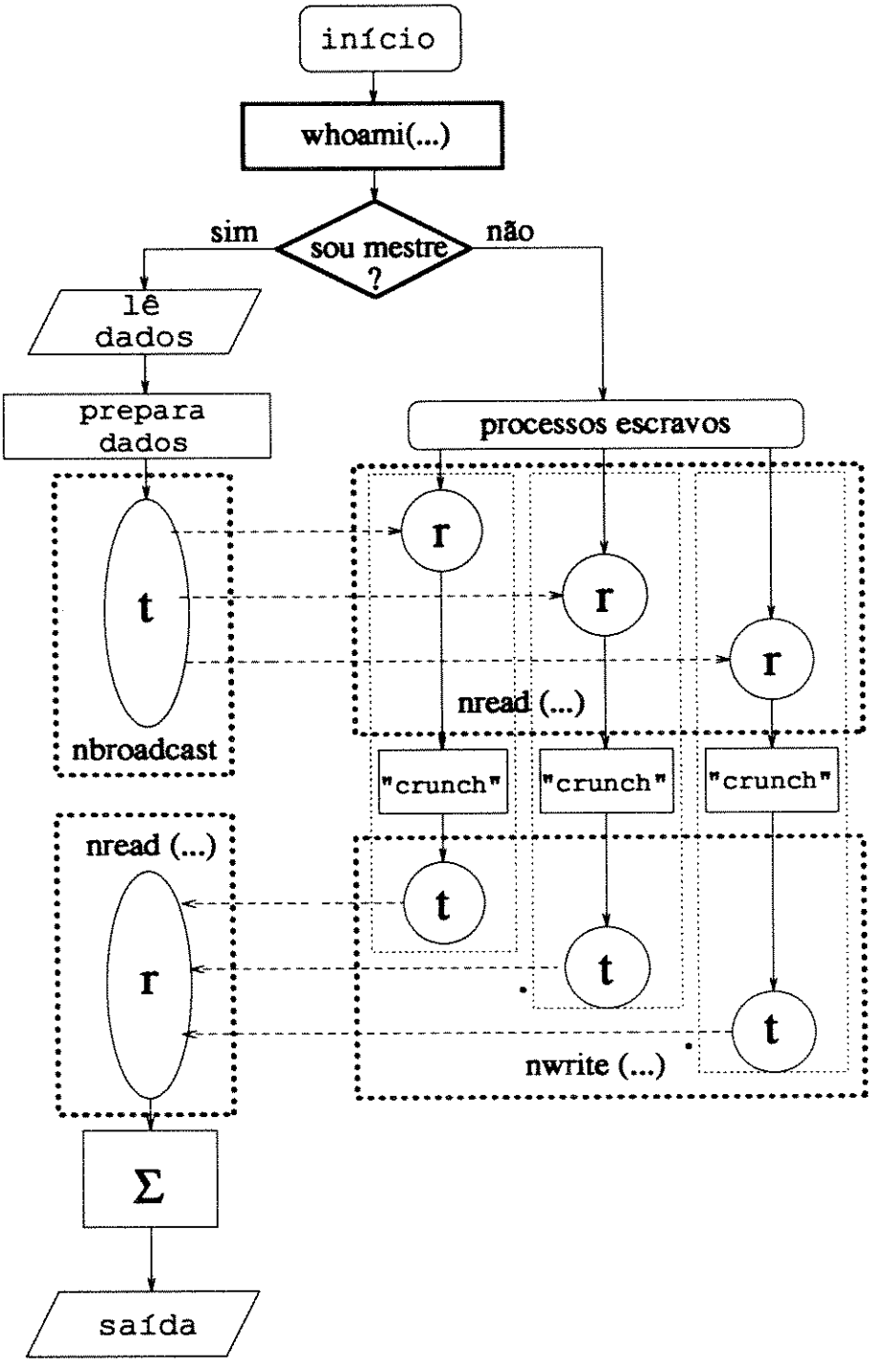


Figura 5.16: Fluxograma do modelo Mestre/Escravo no *nCUBE*

buf: endereço no nó que originou a mensagem.

length: comprimento da mensagem em bytes.

bnode: número relativo do nó origem da mensagem.

type: tipo de mensagem a ser usado em toda a comunicação do *nbroadcast*.

mask: porção do sub-cubo corrente que recebe a mensagem.

nbroadcast transmite mensagens de um nó para todos os nós do subcubo configurado ou para um subcubo completo dentro do subcubo configurado.

- *nread (buffer, nbytes, source, type, flag)*

buffer: local no qual mensagem é armazenada.

nbytes: comprimento da mensagem a ser lida, em bytes.

source: nó fonte da mensagem.

type: tipo de mensagem usado para especificar a espécie da mensagem a ser lida.

flag: argumento não usado.

nread copia uma mensagem do *buffer* para um endereço especificado.

- *nwrite (buffer, nbytes, dest, type, flag)*

buffer: endereço do primeiro *byte* a ser enviado.

nbytes: número de bytes a ser enviado.

dest: nó destino da mensagem.

type: tipo de mensagem a ser enviada.

flag: argumento não usado.

nwrite escreve uma mensagem de um nó do *nCUBE2*

Modelo Mestre/Escravo para o *PVM*

O *host-login* (com o seu processo mestre) é encarregado da leitura dos dados da rede elétrica, no disco. Estes dados são preparados e enviados juntamente com os processos escravos para os demais *hosts* (que participam da máquina virtual). Cada *host*, após receber os dados preparados, inicia a execução dos cálculos intensivos que lhe cabe. Ao término desta tarefa cada *host* dispõe das freqüências dos valores das potências de curtos-circuitos. Estes resultados parciais são enviados ao *host-login*, onde são totalizados e preparadas as saídas em arquivo, para o obtenção do histogramas completos. Deve ser ressaltado que, neste modelo, como ocorre no *nCUBE2*, o *host-login* fica ocioso, no intervalo de tempo compreendido entre o envio de dados preparados e a recepção dos resultados parciais.

A Figura 5.17 mostra a estrutura básica do modelo de programação Mestre/Escravo no *PVM*.

A Figura 5.18 ilustra o fluxograma do modelo Mestre/Escravo no *PVM*.

Sub-rotinas do *PVM* utilizadas no modelo Mestre/Escravo

As sub-rotinas utilizadas neste modelo são as mesmas utilizadas no modelo *SPMD*.

```
            <<programa 'master'>>
pvmfmytid()
pvmfspawn() ! dissemina o programa 'slave'
     le dados do sistema eletrico no disco
     prepara dados
     ! transmite dados preparados para o 'slave'
     pvmfinitsend()
     pvmfpack()
     ...
     pvmfmcast()
     ! aguarda resultados do programa 'slave'
     ! recebe resultados do programa 'slave'
     pvmfrecv()
     pvmfunpack()
     ...
     prepara os arquivos de saida
end
            <<programa 'slave'>>
pvmfmytid()
pvmfparent()
     ! recebe dados preparados do 'slave'
     pvmfrecv()
     pvmfunpack()
     ...
     obtem-se a semente baseado no mynode
     calcula curtos-circuitos
     ! envia resultados para o programa 'master'
     pvmfinitsend()
     pvmfpack()
     ...
     pvmfsend()
end
```

Figura 5.17: Modelo Mestre/Escravo no *PVM*

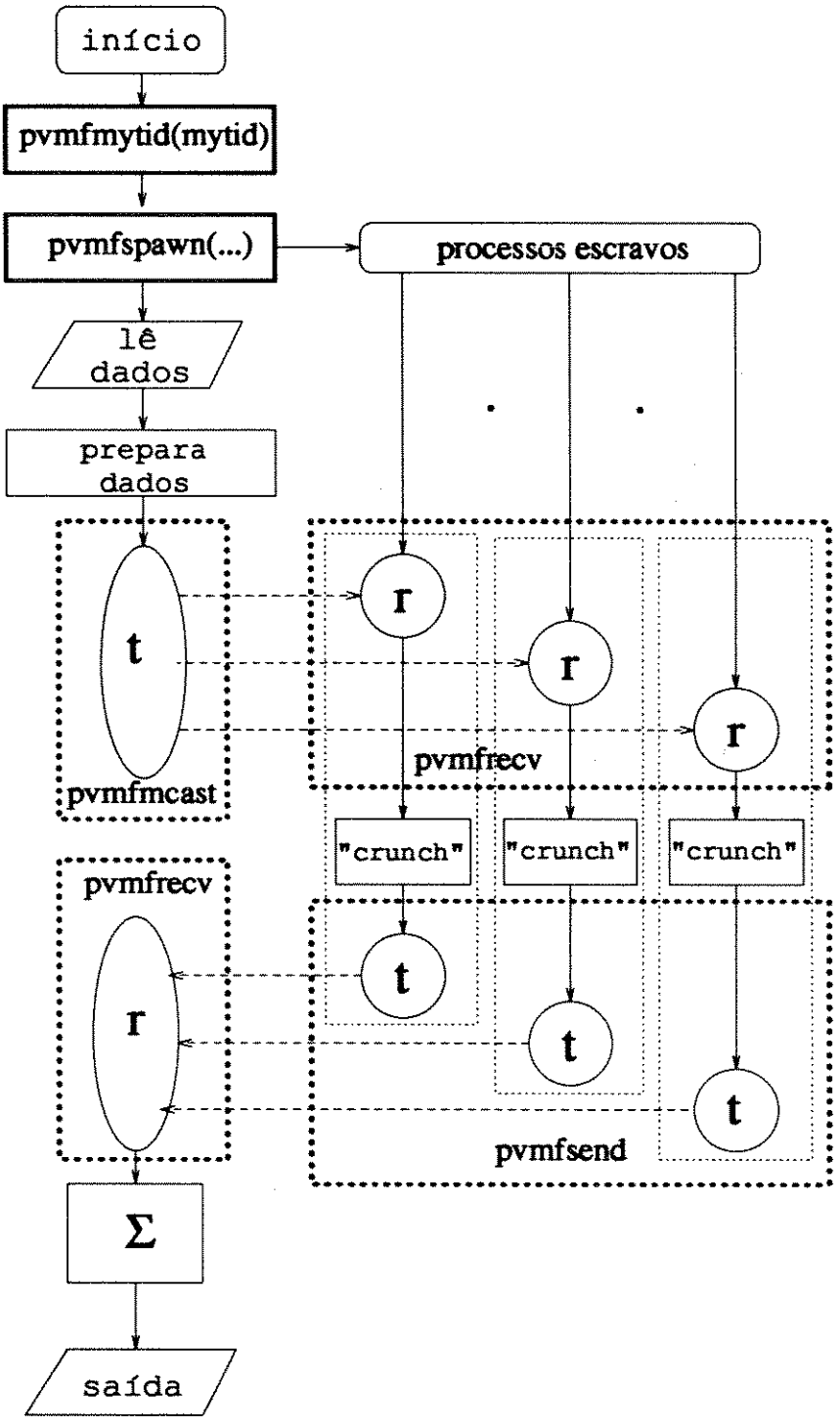


Figura 5.18: Fluxograma do modelo Mestre/Escravo no *PVM*

Capítulo 6

Testes e Resultados

6.1 Sistemas Testados

Este Capítulo descreve os testes e os resultados[1] obtidos através de quatro redes reais do Sistema Interligado da região Sul-Sudeste do Brasil. O Sistema A, formado por 1176 barras, 2250 ramos e 87 linhas com acoplamentos mútuos. Os Sistemas B e C foram obtidos pela redução apropriada do Sistema A, pelo método de equivalentes externos. O Sistema B (reduzido) é formado por 147 barras, 331 ramos e 37 linhas com acoplamentos mútuos, correspondendo ao Sistema CPFL, enquanto o Sistema C (reduzido) abrange a região de Campinas, constando de 39 barras, 86 ramos e 37 linhas com acoplamentos mútuos. Devido ao fato de que o modelo da rede é linear todas as quantidades relevantes nos sistemas reduzidos são exatos, comparando-se com os resultados do sistema completo. Foi também utilizado o Sistema D, correspondendo à rede da *Light*, de 26 barras, 84 ramos e 57 linhas com acoplamentos mútuos. Este sistema serviu para verificar o efeito dos acoplamentos mútuos na função densidade de probabilidade do curto-circuito monofásico.

Os Sistemas A, B e C foram utilizados para testes nos ambientes $nCUBE2$ e na rede de estações *Sun Sparc2* interligados via *Ethernet* e envolvidas com PVM.

Nos testes nos demais ambientes (rede de estações *Sun Classic* interligados via $FDDI$ e ambientes do **CENAPAD-SP**) foi utilizado apenas o Sistema B.

[1]Em todas as Tabelas deste Capítulo os tempos são dados em segundos.

6.2 Implementação Seqüencial

Os resultados mostrados na Tabela 6.1 ilustram as velocidades relativas das máquinas usadas nos testes, quando operadas como computadores monoprocessadores. O tempo de computação é subdividido em duas partes: a parte serial, que corresponde à preparação de dados, incluindo formação e fatoração das matrizes esparsas (matrizes das seqüências positiva e zero), e a parte paralela que corresponde basicamente aos cálculos de curtos-circuitos probabilísticos. O tempo de acesso ao disco não foi considerado nesta Tabela.

Máquina	Sistema	Tempos		
		PS	PP	Total
nCUBE2	A	5,50	78,58	84,08
(nó 0)	B	0,59	78,57	79,16
SPARC1	A	2,41	76,15	78,56
	B	0,32	77,40	77,72
SPARC2	A	1,47	40,33	41,80
	B	0,20	40,93	41,13

Tabela 6.1: Tempos de execução para versão seqüencial em diferentes computadores monoprocessadores

A Tabela 6.2 ilustra a influência do tamanho do sistema elétrico, comparando os tempos de execução para sistemas B e C, reduzidos por equivalentes externos, com os tempos do sistema original A.

Sistema	Tempos			
	Leitura	PS	PP	Saida
A	2,16	1,47	40,33	0,24
B	0,37	0,20	40,93	0,23
C	0,12	0,12	41,64	0,24

Tabela 6.2: Tempo de execução para versão seqüencial na estação *Sparc 2*

Os resultados da Tabela 6.2 mostram a redução drástica dos tempos de leitura e de cálculos da parte seqüencial, na medida em que se reduz o tamanho do sistema. Esta Tabela foi obtida na estação *Sparc 2* e os tempos de processamento foram sub-divididos em quatro componentes: leitura de dados no disco, a parte seqüencial, a parte paralela e a saída.

6.3 Implementação no Computador *nCUBE2*

Modelo *SPMD* com *nglobal/nlocal*

As Tabelas 6.3, 6.4 e 6.5 mostram os tempos obtidos no modelo de programação *SPMD* utilizando as sub-rotinas ***nglobal*** e ***nlocal*** e as Figuras 6.1, 6.2 e 6.3 ilustram respectivamente as curvas do ***speedup***, da eficiência e do tempo de comunicação.

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	1,30	6,17	0,00	5,78	79,20	92,45	1,00
2	1,30	6,17	0,28	5,78	39,60	53,13	1,74
4	1,30	6,17	0,73	5,78	19,81	33,79	2,74
8	1,30	6,17	1,56	5,78	9,90	24,71	3,74
16	1,30	6,17	3,09	5,78	5,00	21,34	4,33
32	1,30	6,17	6,39	5,78	2,48	22,12	4,18
64	1,30	6,17	18,30	5,78	1,24	32,79	2,82

Tabela 6.3: Tempos obtidos para o sistema A no *SPMD*

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	1,30	2,22	0,00	0,54	78,77	82,83	1,00
2	1,30	2,22	0,17	0,54	39,38	43,61	1,90
4	1,30	2,22	0,59	0,54	19,70	24,35	3,40
8	1,30	2,22	1,29	0,54	9,85	15,20	5,45
16	1,30	2,22	2,77	0,54	4,93	11,76	7,04
32	1,30	2,22	6,05	0,54	2,46	12,57	6,59
64	1,30	2,22	11,53	0,54	1,23	16,82	4,92

Tabela 6.4: Tempos obtidos para o sistema B no *SPMD*

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	1,30	1,83	0,00	0,33	78,20	81,66	1,00
2	1,30	1,83	0,09	0,33	39,60	43,15	1,89
4	1,30	1,83	0,18	0,33	19,81	23,45	3,48
8	1,30	1,83	0,98	0,33	9,90	14,35	5,69
16	1,30	1,83	2,57	0,33	4,97	11,00	7,42
32	1,30	1,83	5,44	0,33	2,49	11,39	7,17
64	1,30	1,83	11,08	0,33	1,25	15,79	5,17

Tabela 6.5: Tempos obtidos para o sistema C no *SPMD*

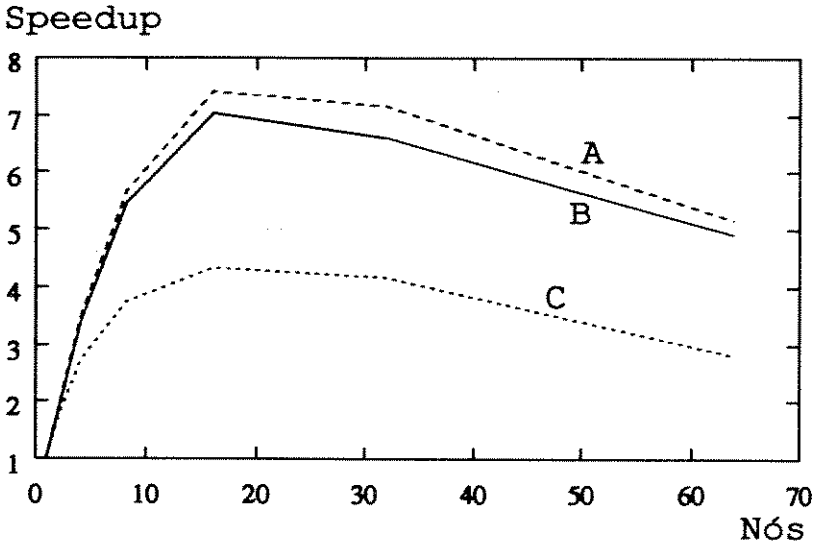


Figura 6.1: Curvas do *speedup* do modelo *SPMD* no *nCUBE2* com *nglobal/nlocal*

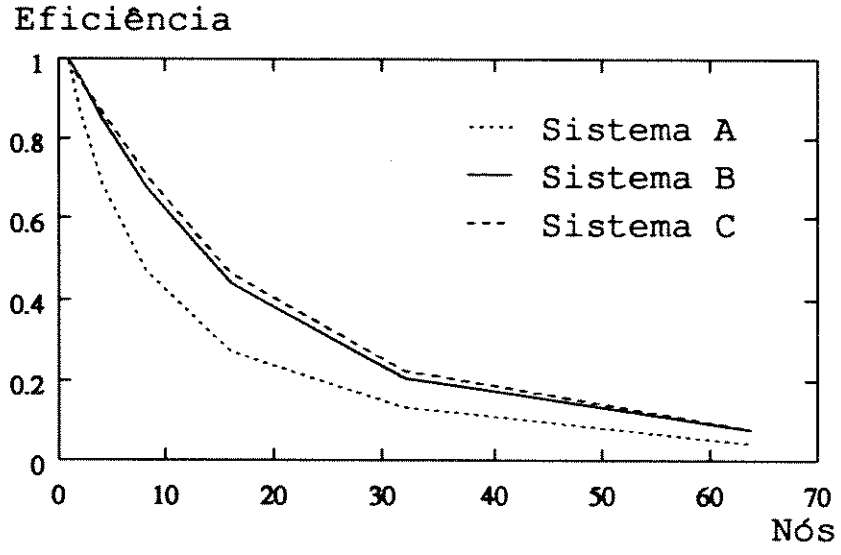


Figura 6.2: Curvas da eficiência do modelo *SPMD* no *nCUBE2* com *nglobal/nlocal*

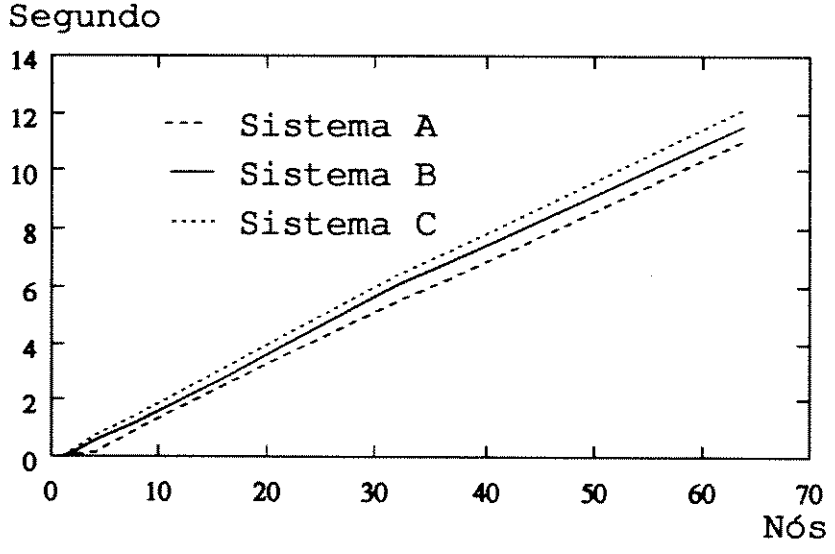


Figura 6.3: Tempo de comunicação no modelo *SPMD* no *nCUBE2* com *nglobal/nlocal*

Modelo *SPMD* com *nbroadcast/nread*

As Tabelas 6.6, 6.7 e 6.8 mostram os tempos obtidos no modelo de programação *SPMD* utilizando as sub-rotinas *nbroadcast* e *nread*.

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	1,30	6,00	0,01	5,54	78,41	91,26	1,00
2	1,30	6,00	0,06	5,54	39,21	52,11	1,75
4	1,30	6,00	0,11	5,54	19,61	32,56	2,80
8	1,30	6,00	0,16	5,54	9,80	22,80	4,00
16	1,30	6,00	0,21	5,54	4,91	17,96	5,08
32	1,30	6,00	0,26	5,54	2,45	15,55	5,87
64	1,30	6,00	0,31	5,54	1,23	14,38	6,35

Tabela 6.6: Tempos obtidos para o sistema A no *SPMD*

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	1,30	1,94	0,01	0,54	77,98	81,77	1,00
2	1,30	1,94	0,04	0,54	38,99	42,81	1,91
4	1,30	1,94	0,06	0,54	19,50	23,34	3,50
8	1,30	1,94	0,09	0,54	9,75	13,62	6,00
16	1,30	1,94	0,11	0,54	4,88	8,77	9,32
32	1,30	1,94	0,14	0,54	2,44	6,36	12,86
64	1,30	1,94	0,16	0,54	1,22	5,16	15,85

Tabela 6.7: Tempos obtidos para o sistema B no *SPMD*

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	1,30	1,50	0,01	0,33	78,40	81,54	1,00
2	1,30	1,50	0,03	0,33	39,20	42,36	1,92
4	1,30	1,50	0,05	0,33	19,61	22,79	3,58
8	1,30	1,50	0,08	0,33	9,80	13,01	6,27
16	1,30	1,50	0,10	0,33	4,91	8,14	10,02
32	1,30	1,50	0,12	0,33	2,45	5,70	14,31
64	1,30	1,50	0,14	0,33	1,23	4,50	18,12

Tabela 6.8: Tempos obtidos para o sistema C no *SPMD*

As Figuras 6.4, 6.5 e 6.6 ilustram respectivamente as curvas do *speedup*, da eficiência e do tempo de comunicação.

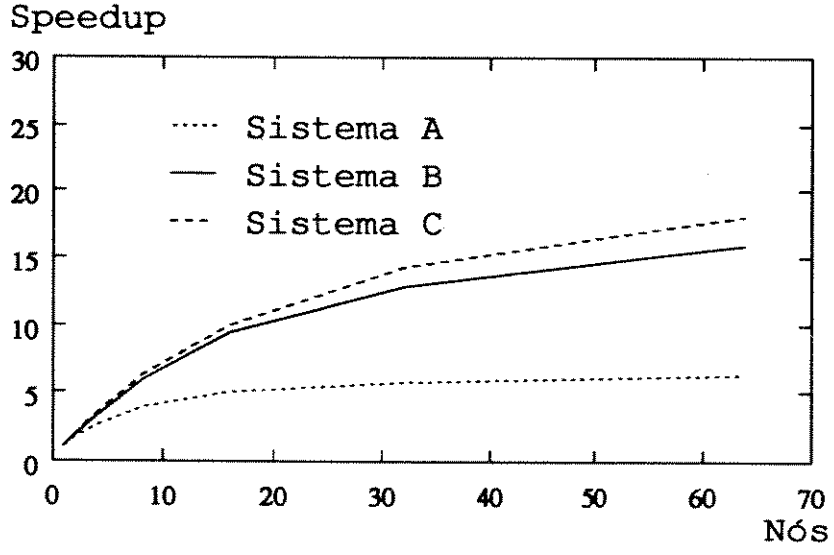


Figura 6.4: Curvas do *speedup* do modelo *SPMD* no *nCUBE2* com *nbroadcast/nread*

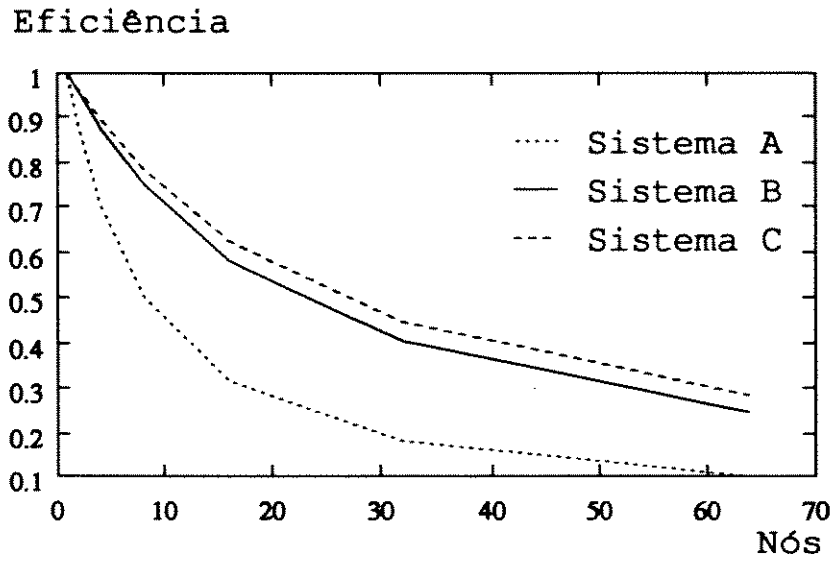


Figura 6.5: Curvas da eficiência do modelo *SPMD* no *nCUBE2* com *nbroadcast/nread*

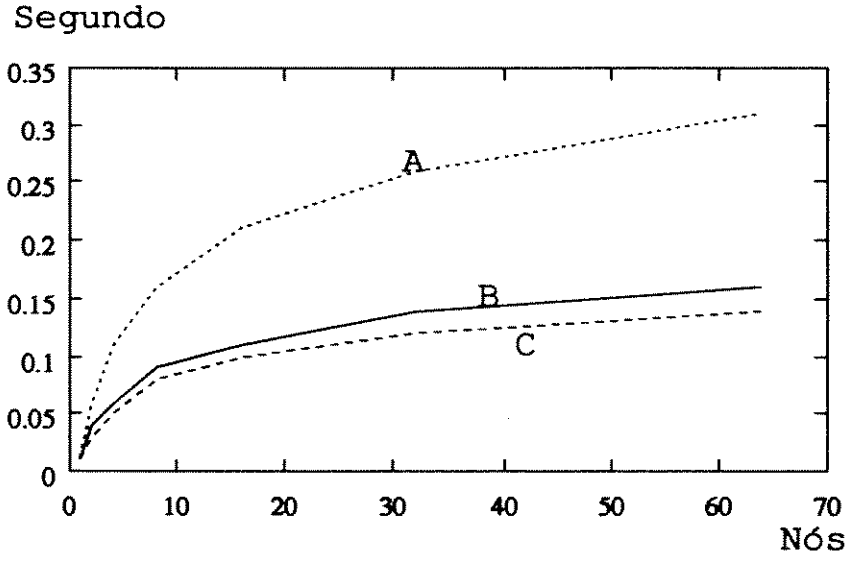


Figura 6.6: Tempo de comunicação no modelo *SPMD* no *nCUBE2* com *nbroadcast/nread*

Modelo Mestre/Escravo com *nbroadcast/nread*

As Tabelas 6.9, 6.10 e 6.11 mostram os tempos obtidos no modelo de programação Mestre/Escravo utilizando as sub-rotinas *nbroadcast* e *nread*.

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
2	1,30	6,25	0,33	6,12	80,20	94,20	1,00
4	1,30	6,25	0,61	6,12	26,74	41,02	2,30
8	1,30	6,25	0,88	6,12	11,47	26,02	3,62
16	1,30	6,25	1,17	6,12	5,36	20,20	4,66
32	1,30	6,25	1,49	6,12	2,58	17,74	5,31
64	1,30	6,25	1,87	6,12	1,27	16,81	5,60

Tabela 6.9: Tempos obtidos para o sistema A no Mestre/Escravo

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
2	1,30	2,18	0,33	0,83	79,79	84,43	1,00
4	1,30	2,18	0,61	0,83	26,60	31,52	2,68
8	1,30	2,18	0,88	0,83	11,41	16,60	5,09
16	1,30	2,18	1,17	0,83	5,33	10,81	7,81
32	1,30	2,18	1,48	0,83	2,58	8,37	10,09
64	1,30	2,18	1,87	0,83	1,27	7,45	11,33

Tabela 6.10: Tempos obtidos para o sistema B no Mestre/Escravo

Nó	Inic.	I/O	Com.	PS	PP	Total	Speedup
2	1,30	1,64	0,33	0,65	80,21	84,13	1,00
4	1,30	1,64	0,61	0,65	26,74	30,94	2,72
8	1,30	1,64	0,88	0,65	11,47	15,94	5,28
16	1,30	1,64	1,17	0,65	5,36	10,12	8,31
32	1,30	1,64	1,48	0,65	2,59	7,66	10,98
64	1,30	1,64	1,87	0,65	1,27	6,73	12,50

Tabela 6.11: Tempos obtidos para o sistema C no Mestre/Escravo

As Figuras 6.7, 6.8 e 6.9 ilustram respectivamente as curvas do *speedup*, da eficiência e do tempo de comunicação.

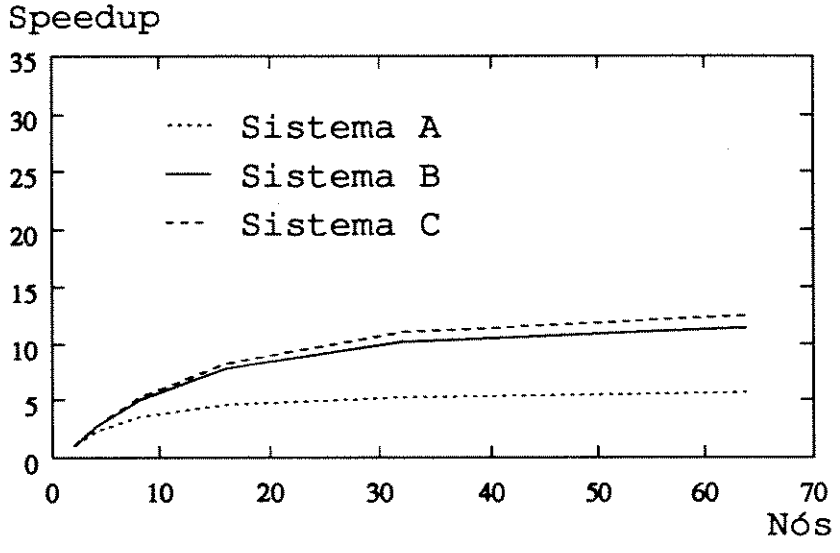


Figura 6.7: Curvas do *speedup* do modelo Mestre/Escravo no *nCUBE2* com *nbroadcast/ nread*

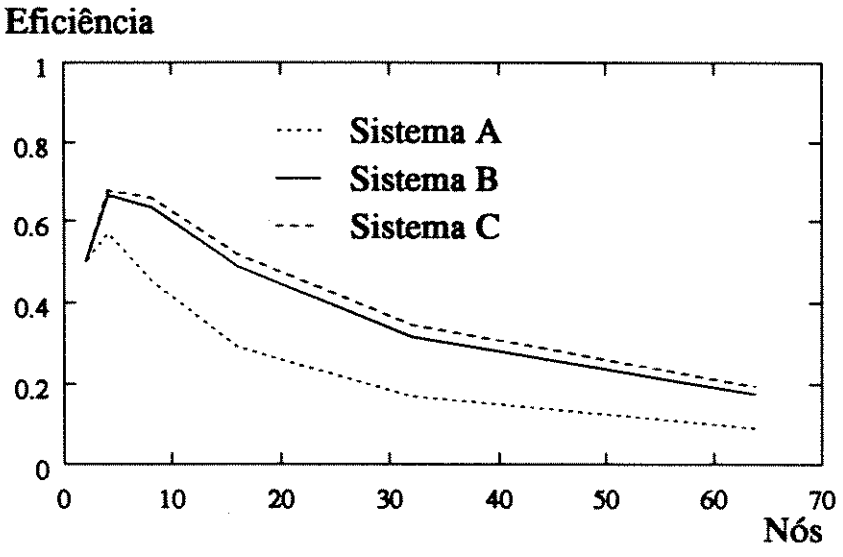


Figura 6.8: Curvas da eficiência do modelo Mestre/Escravo com *nbroadcast/nread*

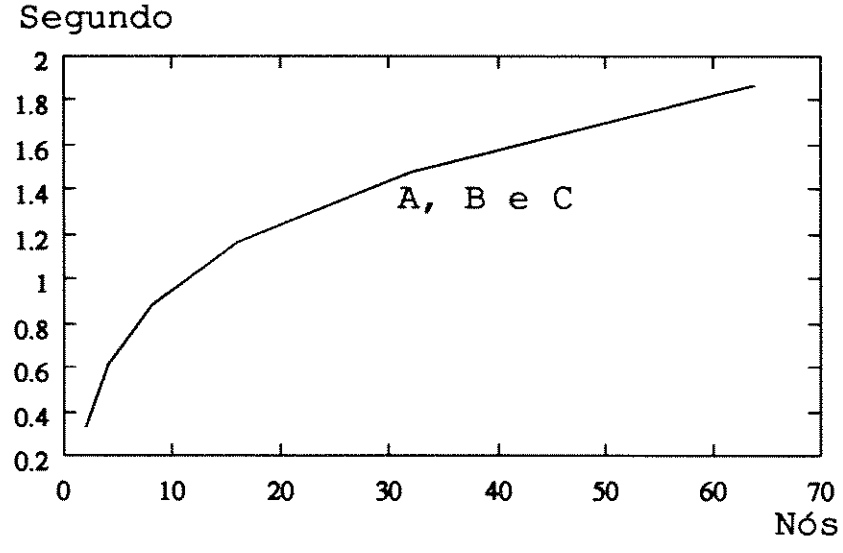


Figura 6.9: Tempo de comunicação no modelo Mestre/Escravo com *nbroadcast/nread*

6.4 Implementação na Rede de Estações *Sun SPARC2* com *PVM*

Modelo *SPMD* com *multicast*

As Tabelas 6.12, 6.13 e 6.14 mostram os tempos obtidos no modelo de programação *SPMD* utilizando a sub-rotina *pvmfmcast*.

N	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	0,35	2,94	0,03	2,09	40,76	46,17	1,00
2	0,39	2,87	0,09	1,93	20,34	25,62	1,80
3	0,39	2,87	0,10	2,01	13,53	18,90	2,44
4	0,47	2,84	0,11	1,97	10,27	15,66	2,95
5	0,51	2,87	0,13	1,96	8,17	13,64	3,38
6	0,59	2,91	0,15	1,93	6,79	12,37	3,73
7	0,59	2,86	0,14	1,99	5,53	11,11	4,16
8	0,67	2,91	0,15	2,06	5,10	10,89	4,24
9	0,71	2,88	0,16	1,98	4,55	10,28	4,49
10	0,74	2,99	0,34	1,95	4,11	10,13	4,56

Tabela 6.12: Tempos obtidos para o sistema A no *SPMD*

N	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	0,34	1,01	0,02	0,28	41,43	43,08	1,00
2	0,39	1,03	0,05	0,29	21,28	23,04	1,87
3	0,43	1,00	0,06	0,27	14,42	16,18	2,66
4	0,48	0,98	0,07	0,33	10,80	12,63	3,41
5	0,52	0,91	0,07	0,29	8,91	10,70	4,03
6	0,58	1,01	0,08	0,33	7,74	9,74	4,42
7	0,58	1,01	0,09	0,32	6,45	8,45	5,10
8	0,65	1,00	0,11	0,39	5,81	7,96	5,41
9	0,69	1,06	0,14	0,38	5,40	7,67	5,62
10	0,90	0,90	0,19	0,28	4,77	7,04	6,12

Tabela 6.13: Tempos obtidos para o sistema B no *SPMD*

N	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	0,33	0,68	0,02	0,19	42,69	43,91	1,00
2	0,40	0,65	0,07	0,17	22,01	23,30	1,88
3	0,47	0,66	0,05	0,15	14,78	16,11	2,73
4	0,49	0,62	0,06	0,16	11,57	12,90	3,40
5	0,54	0,62	0,06	0,15	9,54	10,95	4,01
6	0,58	0,64	0,08	0,18	8,08	9,56	4,59
7	0,57	0,62	0,08	0,19	7,08	8,54	5,14
8	0,62	0,67	0,08	0,23	6,68	8,28	5,30
9	0,68	0,66	0,12	0,25	6,07	7,78	5,64
10	0,93	0,64	0,22	0,25	5,28	7,32	6,00

Tabela 6.14: Tempos obtidos para o sistema C no *SPMD*

As Figuras 6.10, 6.11 e 6.12 ilustram respectivamente as curvas do *speedups*, da eficiência e do tempo de comunicação.

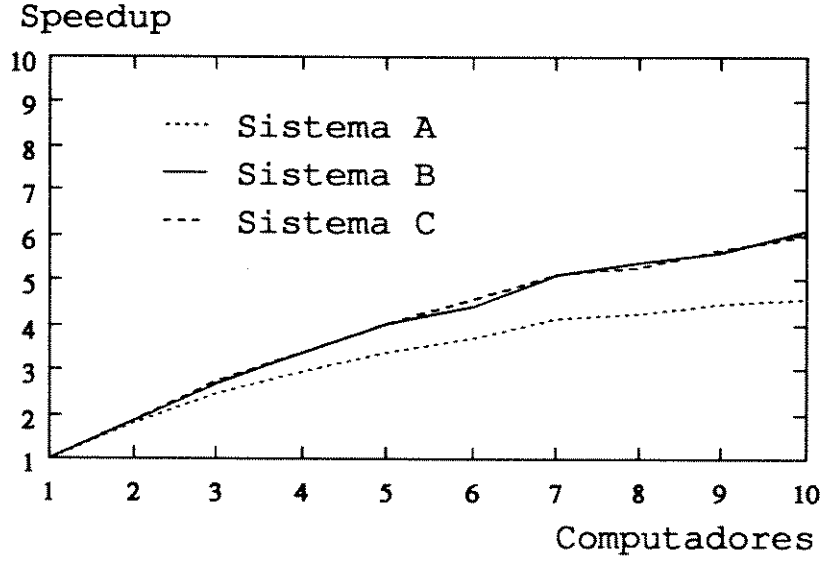


Figura 6.10: Curvas do *speedup* do modelo *SPMD* no *PVM* com *multicast*

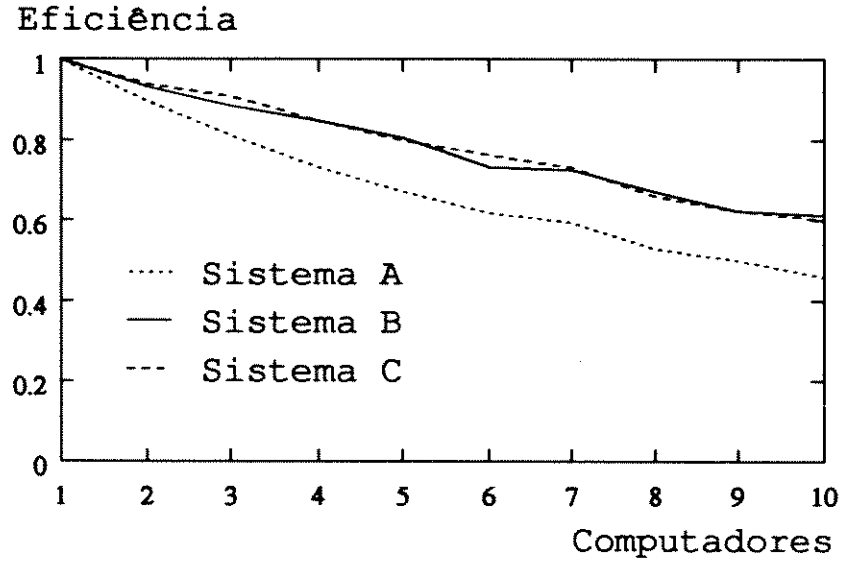


Figura 6.11: Curvas da eficiência do modelo *SPMD* no *PVM* com *multicast*

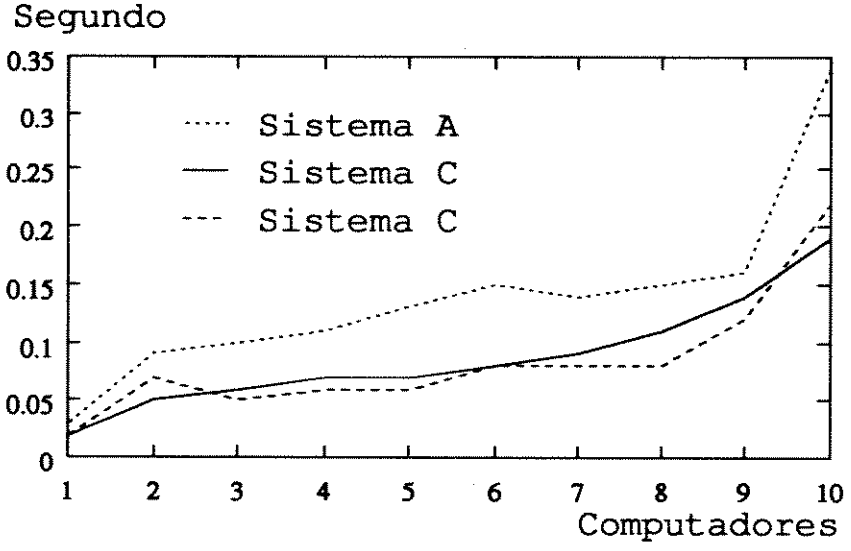


Figura 6.12: Tempo de comunicação no modelo *SPMD* no *PVM* com *multicast*

Modelo Mestre/Escravo com *multicast*

As Tabelas 6.15, 6.16 e 6.17 mostram os tempos obtidos no modelo de programação Mestre/Escravo utilizando a sub-rotina *pvmfmcast*.

N	Inic.	I/O	Com.	PS	PP	Total	Speedup
2	0,66	3,71	1,01	1,55	41,74	48,40	1,00
3	0,70	3,24	1,03	1,65	21,54	28,16	1,72
4	0,92	3,39	1,61	1,91	15,05	22,88	2,12
5	0,62	3,25	1,74	1,50	11,30	18,41	2,63
6	0,72	3,46	1,96	1,51	9,16	16,81	2,88
7	0,58	4,17	2,34	1,60	7,86	16,55	2,92
8	0,69	3,80	2,60	1,49	6,86	15,44	3,13
9	0,70	3,47	3,53	1,53	6,05	15,28	3,17
10	0,69	3,89	3,43	1,53	5,77	15,31	3,16

Tabela 6.15: Tempos obtidos para o sistema A no Mestre/Escravo

N	Inic.	I/O	Com.	PS	PP	Total	Speedup
2	0,48	0,97	0,19	0,25	41,33	43,22	1,00
3	0,42	0,94	0,20	0,24	21,32	23,12	1,87
4	0,52	0,96	0,27	0,25	14,58	16,58	2,61
5	0,43	0,99	0,30	0,26	11,38	13,36	3,24
6	0,45	0,98	0,40	0,29	9,24	11,36	3,80
7	0,52	1,02	0,54	0,24	7,82	10,14	4,26
8	0,55	1,09	0,56	0,25	6,84	9,29	4,65
9	0,63	1,05	0,58	0,26	6,14	8,66	4,99
10	0,56	1,04	0,65	0,26	5,57	8,08	5,35

Tabela 6.16: Tempos obtidos para o sistema B no Mestre/Escravo

N	Inic.	I/O	Com.	PS	PP	Total	Speedup
2	0,39	0,71	0,14	0,17	41,55	42,96	1,00
3	0,46	0,69	0,12	0,19	21,52	22,98	1,87
4	0,53	0,65	0,48	0,17	14,94	16,77	2,56
5	0,68	0,68	0,26	0,16	11,30	13,08	3,28
6	0,51	0,66	0,41	0,20	9,16	10,94	3,93
7	0,57	0,68	0,30	0,21	7,93	9,69	4,43
8	0,55	0,67	0,27	0,17	6,99	8,65	4,97
9	0,61	0,67	0,48	0,17	6,25	8,18	5,25
10	0,56	0,71	0,48	0,21	5,65	7,61	5,65

Tabela 6.17: Tempos obtidos para o sistema C no Mestre/Escravo

As Figuras 6.13, 6.14 e 6.15 ilustram respectivamente as curvas do *speedup*, da eficiência e do tempo de comunicação.

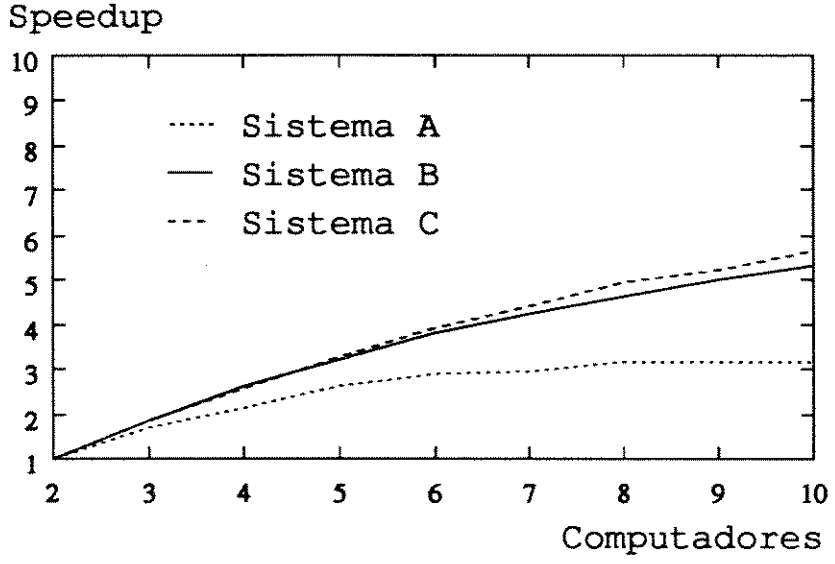


Figura 6.13: Curvas do *speedup* do modelo Mestre/Escravo no *PVM* com *multicast*

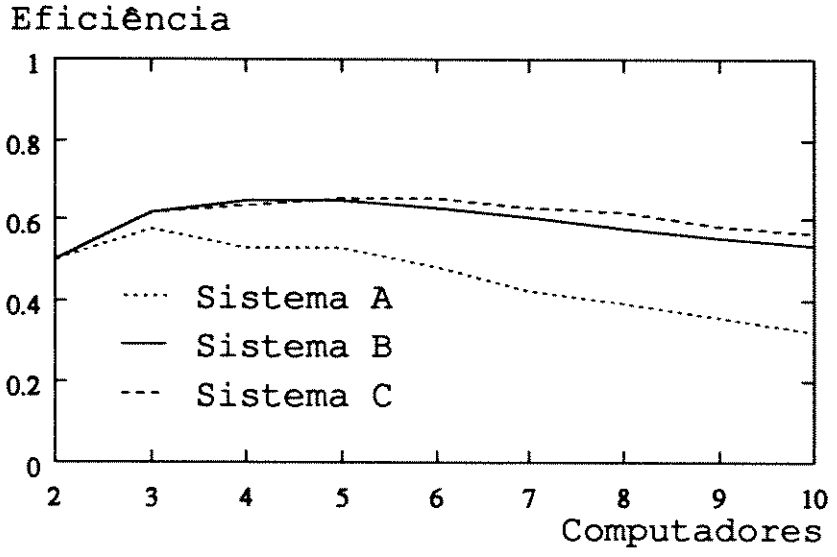


Figura 6.14: Curvas da eficiência do modelo Mestre/Escravo no *PVM* com *multicast*

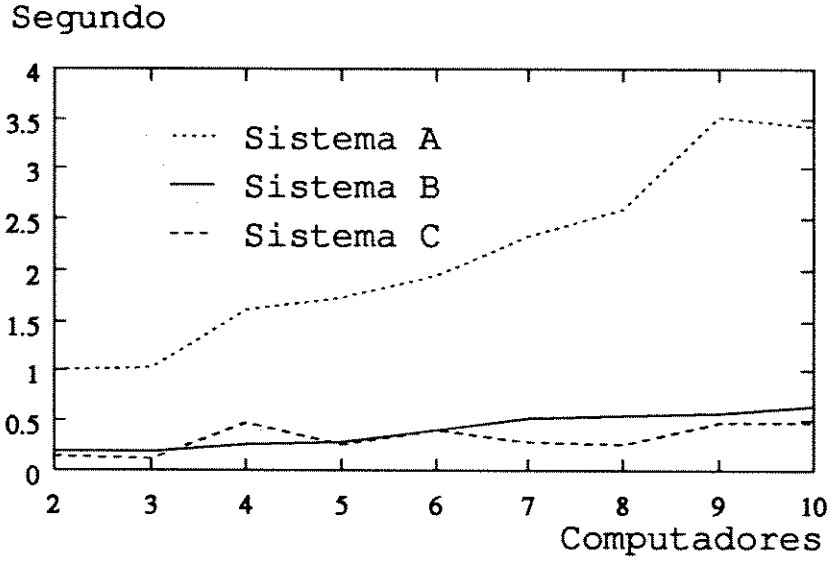


Figura 6.15: Tempo de comunicação no modelo Mestre/Escravo no *PVM* com *multicast*

Rede *Ethernet* moderadamente ocupada

A Tabela 6.18 mostra os tempos obtidos para o sistemas B no PVM, para o modelo $SPMD$ com rede moderadamente ocupada.

N	Inic.	I/O	Com.	PS	PP	Total	Speedup
1	0,35	1,14	0,03	0,30	45,20	47,02	0,91
2	0,41	1,05	17,87	0,34	23,06	42,39	1,00
3	0,57	1,03	11,78	0,29	14,80	28,48	1,50
4	0,72	1,07	7,87	0,31	11,66	21,63	1,96
5	0,72	1,00	6,35	0,28	8,81	17,16	2,49
6	0,83	1,07	6,55	0,39	7,31	16,15	2,65
7	1,09	1,07	2,99	0,34	7,59	13,08	3,54
8	0,91	1,18	3,24	0,40	5,90	11,63	3,67
9	1,13	1,01	3,29	0,40	4,80	10,63	4,02
10	1,22	1,06	2,43	0,48	4,28	9,47	4,51

Tabela 6.18: Tempos obtidos para o sistema B no $SPMD$

As Figuras 6.16 e 6.17 ilustram as curvas do *speedup* e da eficiência, nas quais são comparadas dois níveis de utilização: rede livre e rede moderadamente ocupada.

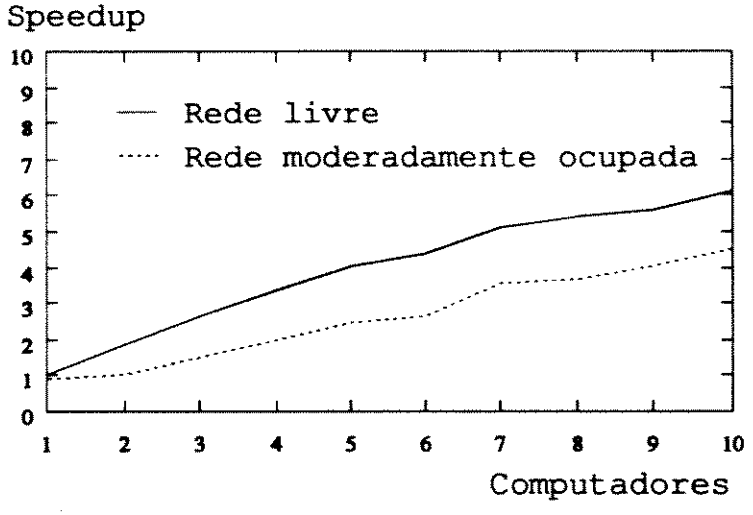


Figura 6.16: Curvas do *speedup* para dois níveis de utilização no modelo $SPMD$

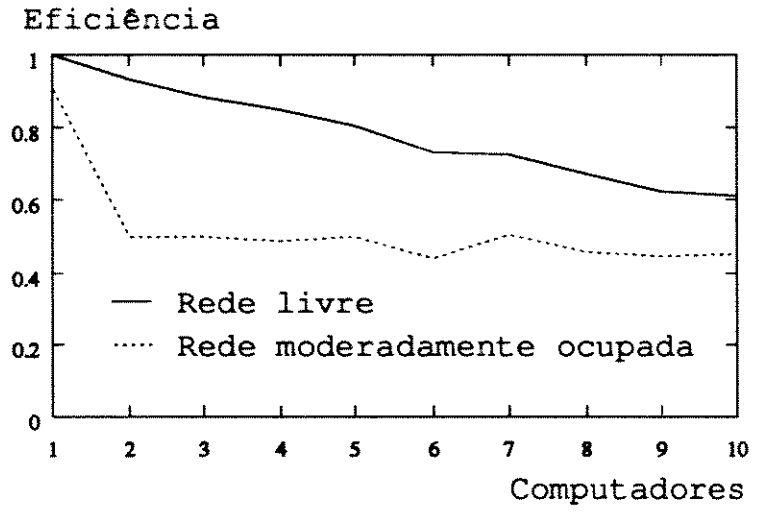


Figura 6.17: Curvas da eficiência para dois níveis de utilização no modelo *SPMD*

A Tabela 6.19 mostra os tempos obtidos para o sistema B no *PVM*, para o modelos Mestre/Escravo com rede moderadamente ocupada.

N	Inic.	I/O	Com.	PS	PP	Tempos	Speedup
2	0,28	1,30	0,26	0,26	57,67	59,77	0,72
3	0,40	1,15	0,32	0,28	24,20	26,35	1,62
4	0,30	1,15	0,26	0,30	17,23	19,24	2,22
5	0,36	1,10	0,41	0,29	14,64	16,80	2,54
6	0,33	1,12	0,47	0,24	12,03	14,19	3,01
7	0,40	1,18	0,63	0,25	12,01	14,47	2,95
8	0,39	1,14	0,57	0,27	10,50	11,73	3,32
9	0,35	1,25	0,75	0,30	8,47	11,12	3,84
10	0,40	1,10	1,36	0,28	8,07	11,21	3,81

Tabela 6.19: Tempos obtidos para o sistema B no Mestre/Escravo

As Figuras 6.18, 6.19 e 6.20 ilustram respectivamente as curvas do *speedup*, da eficiência e do tempo de comunicação, nas quais são comparadas dois níveis de utilização: rede livre e rede moderadamente ocupada.

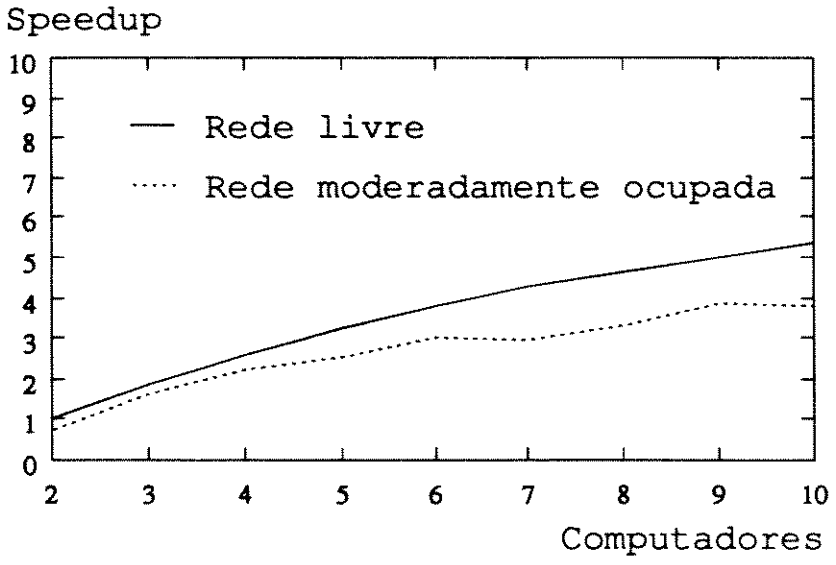


Figura 6.18: Curvas do *speedup* para dois níveis de utilização no modelo Mestre/Escravo

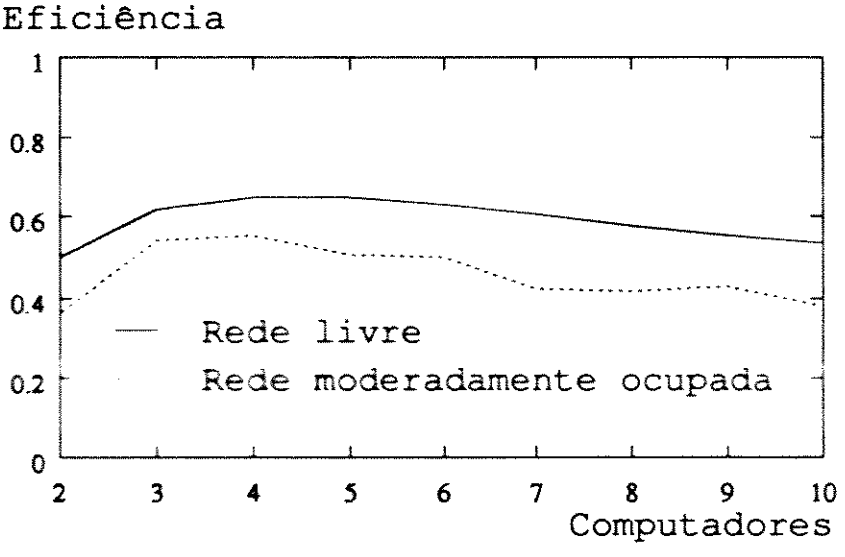


Figura 6.19: Curvas da eficiência para dois níveis de utilização no modelo Mestre/Escravo

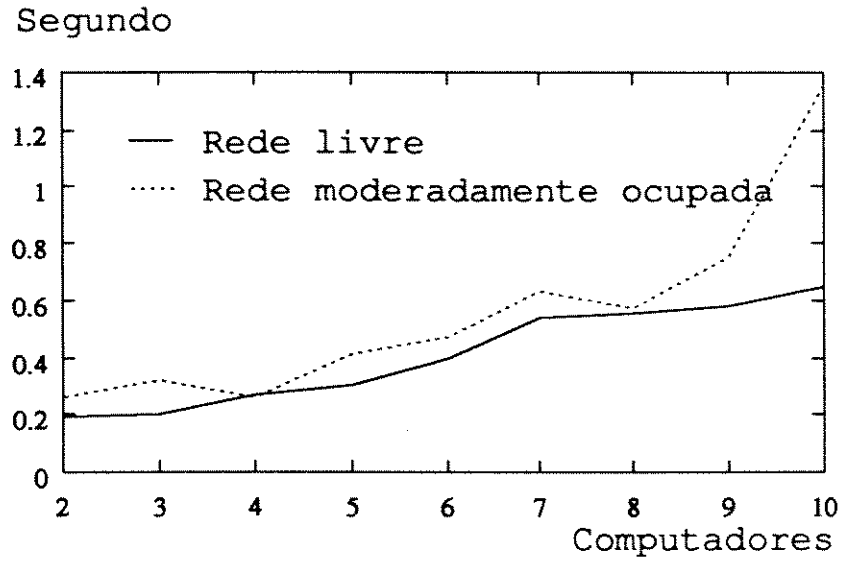


Figura 6.20: Tempo de comunicação para dois níveis de utilização no modelo *Mestre/ Escravo*

Modelo *SPMD* com *broadcast* - Grupos Dinâmicos de Processos

A Tabela 6.20 mostra os tempos obtidos para o sistema B no *PVM*, para o modelo *SPMD* - grupos dinâmicos com máquinas vazias.

Nó	Inic.	I/O	Comun.	PS	PP	Total	Speedup
1	0,35	0,94	0,04	0,21	41,12	42,66	1,00
2	0,41	0,85	0,08	0,21	20,92	22,47	1,90
3	0,57	0,83	0,10	0,20	13,74	15,43	2,76
4	0,72	0,82	0,11	0,24	10,31	12,20	3,50
5	0,72	0,86	0,12	0,24	8,23	10,17	4,20
6	0,83	0,83	0,14	0,37	6,86	9,03	4,73
7	1,09	0,83	0,13	0,32	5.89	8,26	5,16
8	0,91	0,84	0,18	0,41	5,18	7,52	5,67
9	1,13	0,83	1,07	0,35	4,61	8,07	5,35
10	1,22	0,82	2,59	0,39	4,17	9,20	4,63

Tabela 6.20: Tempos obtidos para o sistema B no *SPMD* - grupos dinâmicos

As Figuras 6.21 e 6.22 ilustram respectivamente as curvas do *speedup* e da eficiência.

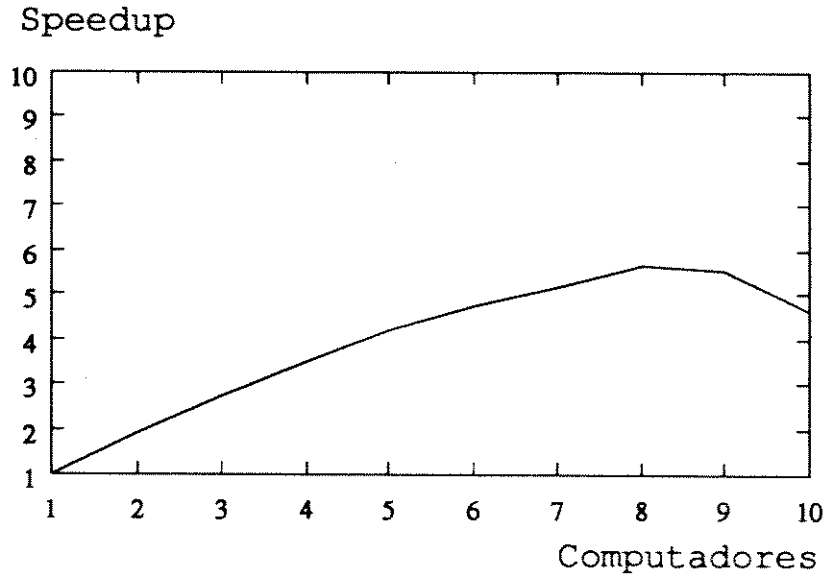


Figura 6.21: Curva do *speedup* do modelo *SPMD* no *PVM* com *multicast*

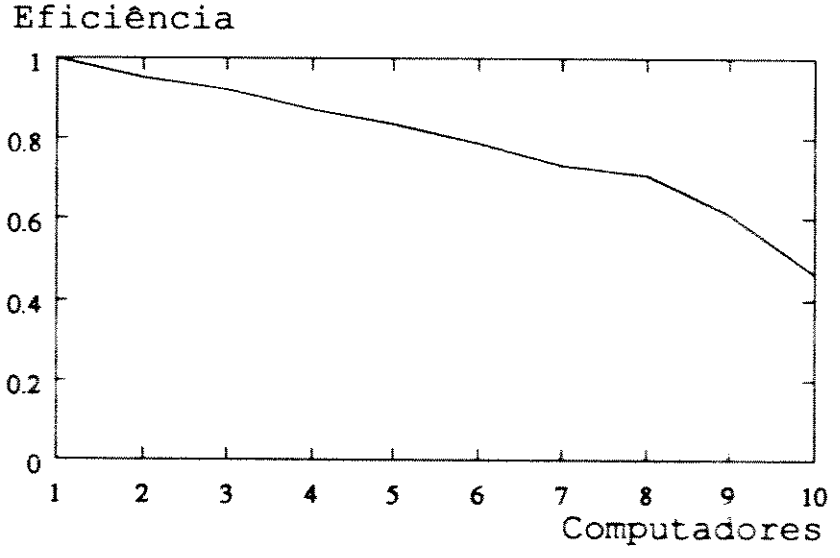


Figura 6.22: Curva da eficiência do modelo *SPMD* no *PVM* com *multicast*

6.5 Implementação na Rede de Estações *Sun Classic* com *PVM*

Modelo *SPMD* com *multicast*

A Tabela 6.21 mostra os tempos obtidos para o sistema B para o modelo *SPMD* com rede vazia e as Figuras 6.23 e 6.24 ilustram respectivamente as curvas ddo *speedup* e da eficiência.

Nó	Inic.	I/O	Comun.	PS	PP	Total	Speedup
1	0,33	0,90	0,02	0,49	65,09	66,84	1,00
2	0,30	0,87	0,78	0,44	32,71	34,97	1,91
3	0,50	0,85	0,70	0,43	21,71	23,95	2,79
4	0,52	0,85	0,61	0,49	16,25	18,40	3.63
5	0,57	0,86	0,48	0,52	13,15	15,15	4,41
6	0,71	0,89	0,55	0,55	10,88	13,03	5.13
7	0,82	0,88	0,54	0,65	9,30	11,52	5.80
8	0,78	0,85	0,54	0,68	8,18	10,36	6,45

Tabela 6.21: Tempos obtidos para o sistema B no *SPMD*

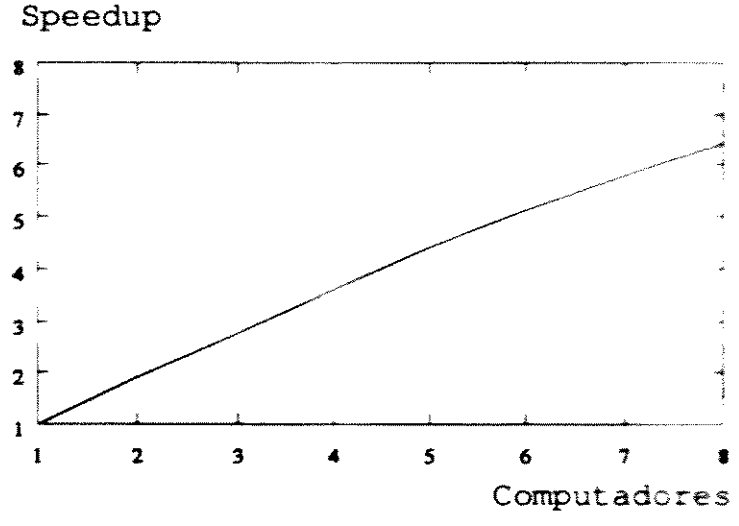


Figura 6.23: Curva do *speedup* do modelo *SPMD* no *PVM* com *multicast*

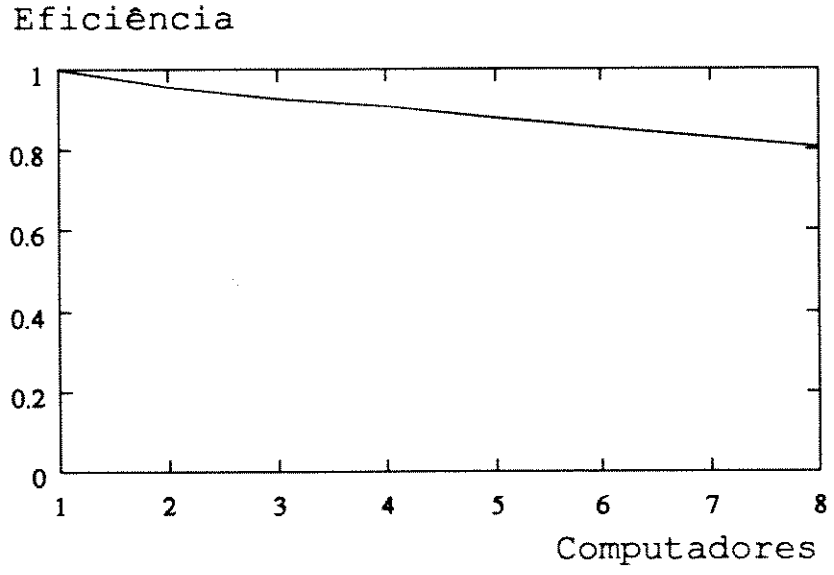


Figura 6.24: Curva da eficiência do modelo *SPMD* no *PVM* com *multicast*

Modelo Mestre/Escravo com *multicast*

A Tabela 6.22 mostra os tempos obtidos para o sistema B para o modelos Mestre/Escravo com rede vazia e as Figuras 6.25 e 6.26 ilustram respectivamente as curvas do *speedup* e da eficiência.

No	Inic.	I/O	Comun	PS	PP	Total	Speedup
2	0,39	0,86	0.29	0.44	63,24	65,21	1,00
3	0,39	1,03	0.23	0.42	32,26	34,33	1,90
4	0,60	0,87	0.2[illegible]	0.40	21.08	23,22	2,81
5	0,58	0,96	0.3[illegible]	0.41	16,00	18,32	3,56
6	0,77	0,88	0.4[illegible]	0.41	12,70	15,20	4,29
7	0,51	1,01	0.43	0.41	10.57	12.93	5.04
8	0,47	0,96	0.53	0.42	9,16	11,54	5,65

Tabela 6.22: Tempos obtidos para o sistema B no Mestre/Escravo

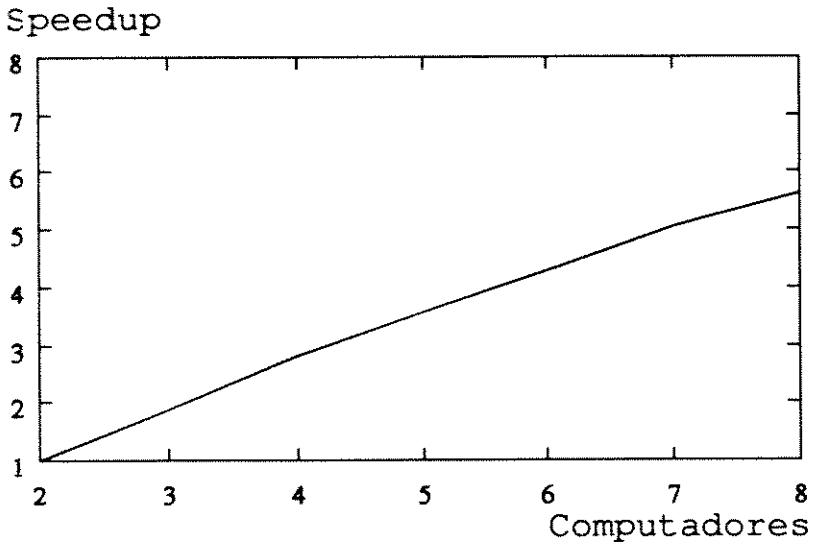


Figura 6.25: Curva do *speedup* do modelo Mestre/Escravo no *PVM* com *multicast*

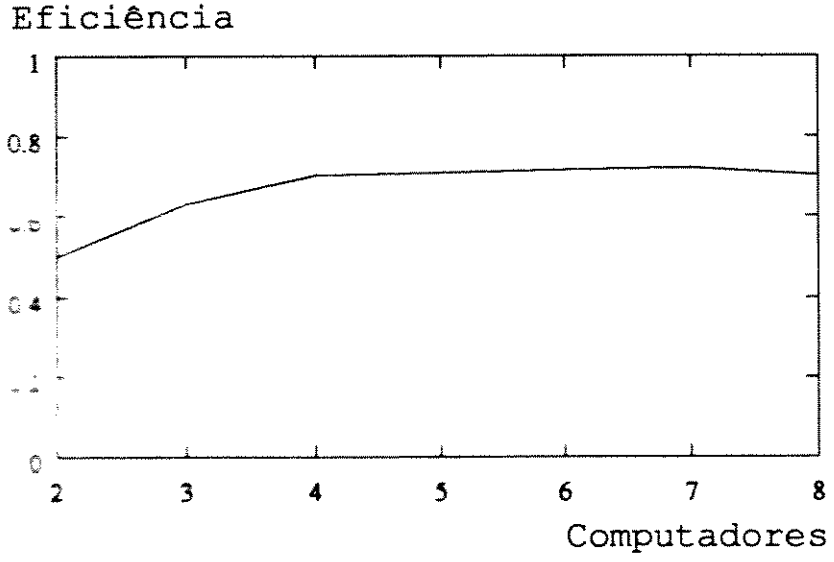


Figura 6.26: Curva da eficiência do modelo Mestre/Escravo no *PVM* com *multicast*

Modelo *SPMD* com *broadcast* - Grupos Dinâmicos de Processos

A Tabela 6.23 mostra os tempos obtidos para o sistema B para o modelo *SPMD* com máquinas vazias e as Figuras 6.27 e 6.28 ilustram respectivamente as curvas do *speedup* e da eficiência.

Nó	Inic.	I/O	Comun.	PS	PP	Total	Speedup
1	0,32	0,83	0,11	0,46	64,20	65,92	1,00
2	0,30	0,82	0,32	0,42	31,84	33,70	1,96
3	0,50	0,81	0,33	0,46	21,20	23,28	2,83
4	0,51	0,80	0,47	0,47	15,94	18,03	3,66
5	0,55	0,77	0,31	0,45	12,78	14,85	4,44
6	0,68	0,80	0,37	0,49	10,62	12,97	5,08
7	0,75	0,82	0,41	0,49	9,16	11,63	5,67
8	0,78	0,83	0,51	0,51	8,03	10,66	6,18

Tabela 6.23: Tempos obtidos para o sistema B no *SPMD* - grupos dinâmicos

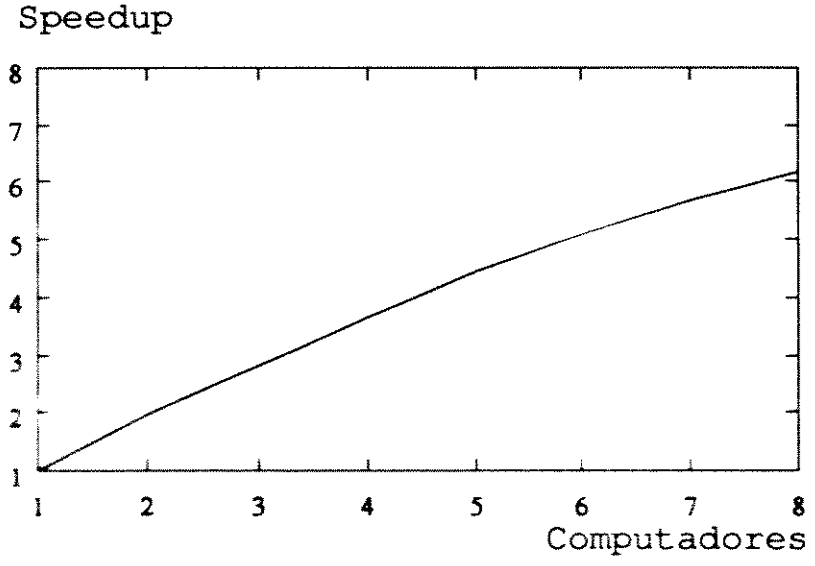


Figura 6.27: Curva do *speedup* do modelo *SPMD* no *PVM* com *broadcast*

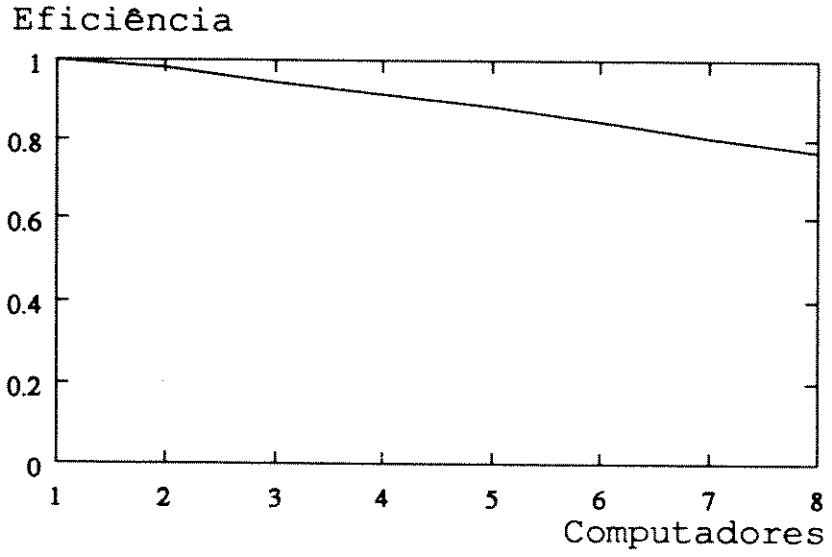


Figura 6.28: Curva da eficiência do modelo $SPMD$ no PVM com *broadcast*

6.6 Implementação no Computador $SP1$ com PVM

Modelo $SPMD$ com *multicast*

A Tabela 6.24 mostra os tempos obtidos para o sistema B para o modelo $SPMD$ com máquinas ocupadas e as Figuras 6.29 e 6.30 ilustram respectivamente as curvas do *speedup* e da eficiência.

Nó	Inic.	I/O	Comun.	PS	PP	Total	Speedup
1	0,13	0,35	0,01	0,09	12,19	12,77	1,00
2	0,09	0,32	0,02	0,07	6,50	7,00	1,83
3	0,11	0,31	3,55	0,07	3,98	8,02	1,60
4	0,16	0,32	3,24	0,07	2,93	6,72	1,93
5	0,15	0,31	2,59	0,07	2,33	5,45	2,38
6	0,19	0,34	2,22	0,07	1,78	4,60	2,85
7	0,20	0,31	1,79	0,07	1,50	3,87	3,40
8	0,22	0,34	3,74	0,07	0,87	5,24	2,51

Tabela 6.24: Tempos obtidos para o sistema B no $SPMD$

Máquina	Carga
n01.cna.unic	0.99
n04.cna.unic	1.07
n06.cna.unic	1.07
n08.cna.unic	2.02
n05.cna.unic	1.16
n07.cna.unic	2.04
n03.cna.unic	2.97
n02.cna.unic	3.02

Tabela 6.25: Ocupação média das máquinas do $SP1$

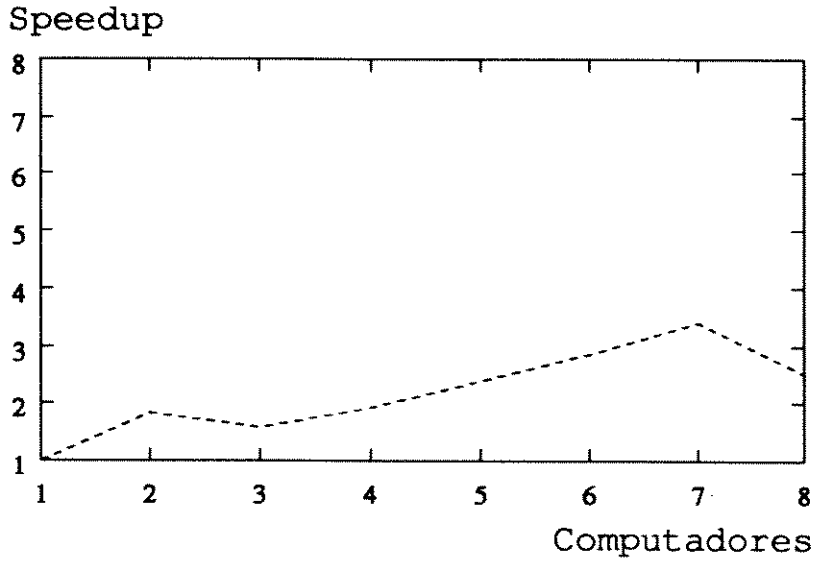


Figura 6.29: Curva do *speedup* do modelo *SPMD* no *PVM* com *multicast*

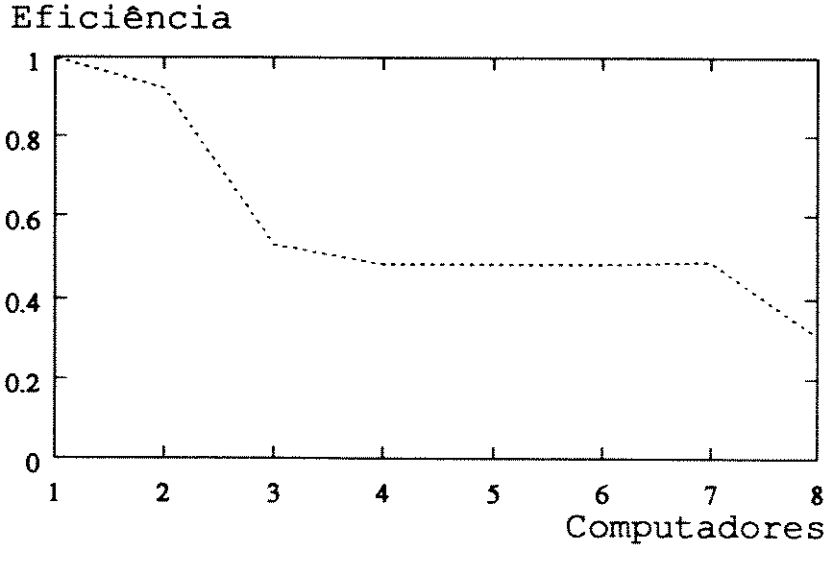


Figura 6.30: Curva da eficiência do modelo *SPMD* no *PVM* com *multicast*

Modelo Mestre/Escravo com *multicast*

A Tabela 6.26 mostra os tempos obtidos para o sistema B no PVM, para o modelo $SPMD$ com máquinas ocupadas e as Figuras 6.31 e 6.32 ilustram respectivamente as curvas do *speedup* e da eficiência.

Nó	Inic.	I/O	Comun.	PS	PP	Tempos	Speedup
2	0,10	0,38	0,22	0,07	29,21	29,98	1,00
3	0,13	0,32	0,21	0,07	15,34	16,07	1,87
4	0,10	0,34	0,38	0,07	10,34	11,23	2,67
5	0,17	0,35	0,79	0,08	7,32	8,71	3,44
6	0,11	0,32	2,41	0,07	5,70	8,61	3,48
7	0,11	0,33	1,31	0,07	4,69	6,51	4,60
8	0,11	0,35	1,90	0,07	3,62	6,05	4,96

Tabela 6.26: Tempos obtidos para o sistema B no Mestre/Escravo

Máquina	Carga
n08.cna.unic	3.09
n05.cna.unic	3.17
n01.cna.unic	3.39
n06.cna.unic	4.02
n07.cna.unic	4.02
n04.cna.unic	4.04
n02.cna.unic	4.09
n03.cna.unic	5.02

Tabela 6.27: Ocupação média das máquinas do $SP1$

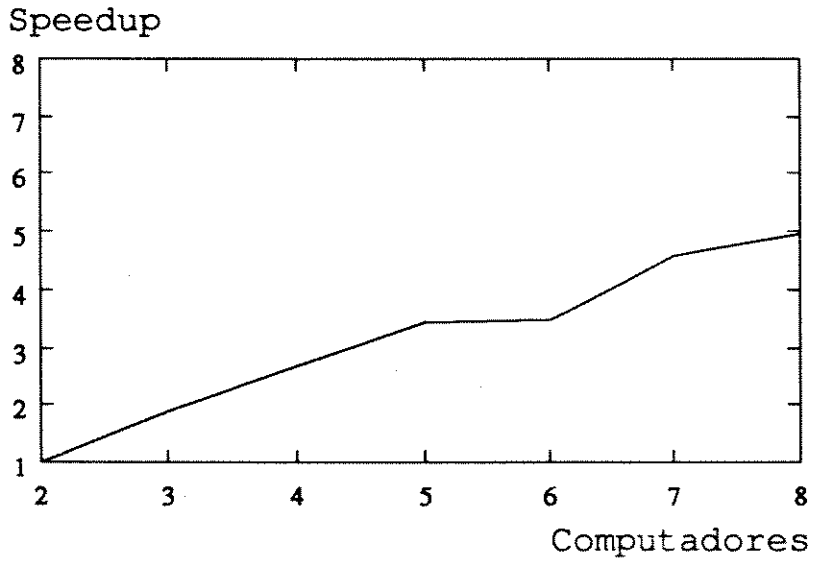


Figura 6.31: Curva do *speedup* do modelo Mestre/Escravo no *PVM* com *multicast*

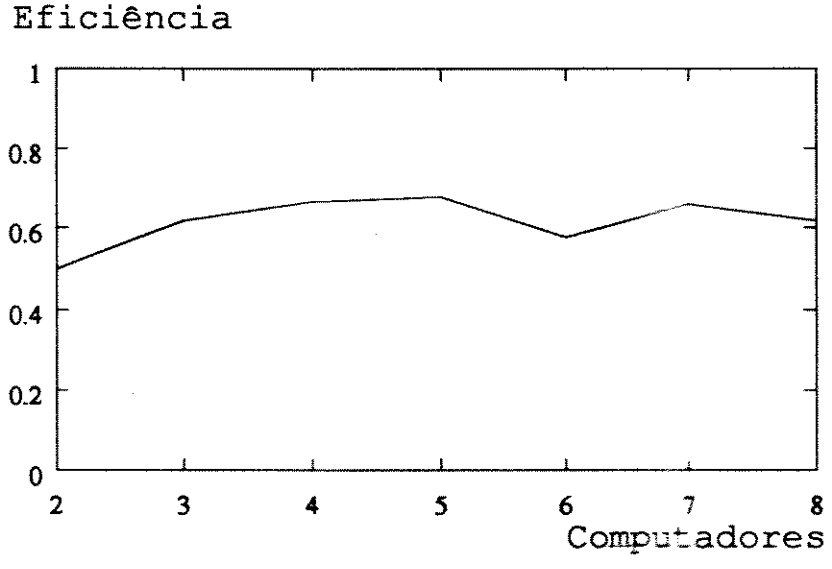


Figura 6.32: Curva da eficiência do modelo Mestre/Escravo no *PVM* com *multicast*

6.7 Implementação na Rede de Estações *IBM RISC*6000 com *PVM*

Modelo *SPMD* com *multicast*

A Tabela 6.28 mostra os tempos obtidos para o sistema B no *PVM*, para o modelo *SPMD* com máquinas ocupadas e as Figuras 6.33 e 6.34 ilustram respectivamente as curvas do *speedup* e da eficiência.

Máq.	Inic.	I/O	Comun.	PS	PP	Tempo Total	*Speedup*
1	0,15	0,58	0,01	0,11	20,10	20,95	1,00
2	0,11	0,55	0,14	0,10	9,69	10,54	1,98
3	0,16	1,22	1,56	0,12	6,06	9,03	2,32
4	0,30	0,82	4,81	0,15	4,33	10,29	2,04
5	0,25	0,75	3,14	0,16	3,91	8,07	2,59
6	0,33	0,54	3,13	0,17	2,64	6,56	3,19

Tabela 6.28: Tempos obtidos para o sistema B no *SPMD*

Máquina	Carga
imbuia.cna.u	2.10
ipe.cna.unic	2.12
alamo.cna.un	3.03
frejo.cna.un	3.04
pinho.cna.un	3.04
mogno.cna.un	3.05
peroba.cna.u	3.06
cedro.cna.un	3.09

Tabela 6.29: Ocupação média das máquinas *IBM RISC*6000

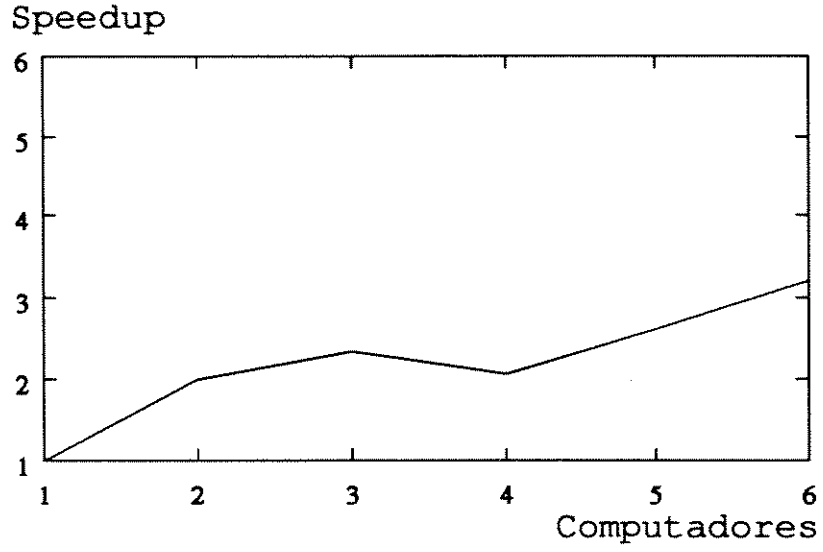


Figura 6.33: Curva do *speedup* do modelo *SPMD* no *PVM* com *multicast*

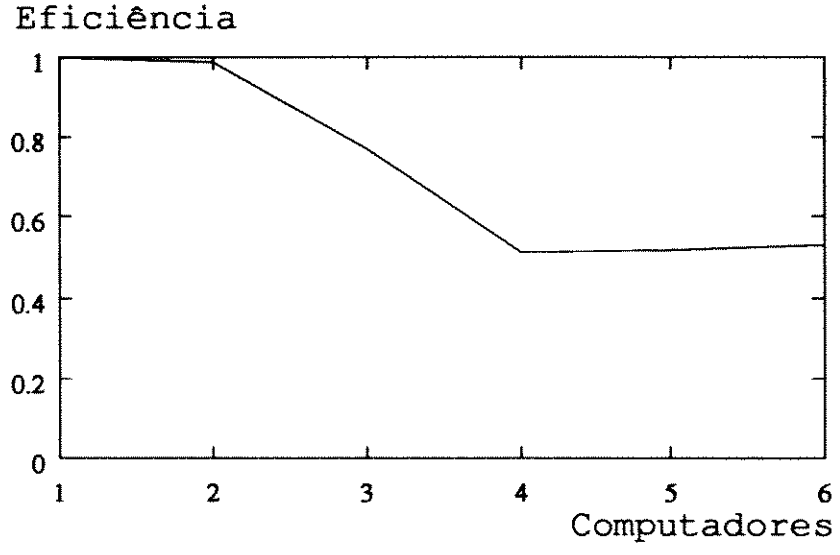


Figura 6.34: Curva da eficiência do modelo *SPMD* no *PVM* com *multicast*

Modelo *SPMD* com *broadcast* - Grupos Dinâmicos de Processos

A Tabela 6.30 mostra os tempos obtidos para o sistema B no *PVM*, para o modelo *SPMD* com máquinas ocupadas e as Figuras 6.35 e 6.36 ilustram respectivamente as curvas do *speedup* e da eficiência.

Máq.	Inic.	I/O	Comun.	PS	PP	Tempo Total	*Speedup*
1	0,10	0,58	0,03	0,09	18,56	19,35	1,00
2	0,15	0,56	1,97	0,10	9,16	11,95	1,62
3	0,20	0,57	0,17	0,11	6,05	7,09	2,73
4	0,25	0,55	0,13	0,15	4,40	5,48	3,53
5	0,22	0,59	0,10	0,22	3,48	4,60	4,20
6	0,30	0,56	0,24	0,25	2,94	4,30	4,50

Tabela 6.30: Tempos obtidos para o sistema B no *SPMD* - grupos dinâmicos

Máquina	Carga
ipe.cna.unic	2.01
pinho.cna.un	2.25
cedro.cna.un	3.00
mogno.cna.un	3.03
peroba.cna.u	3.12
alamo.cna.un	4.01
imbuia.cna.u	4.01
frejo.cna.un	4.05

Tabela 6.31: Ocupação média das máquinas *IBM RISC*6000

As Figuras 6.35 e 6.36 ilustram respectivamente as curvas de *speedups* e eficiências com máquinas ocupadas.

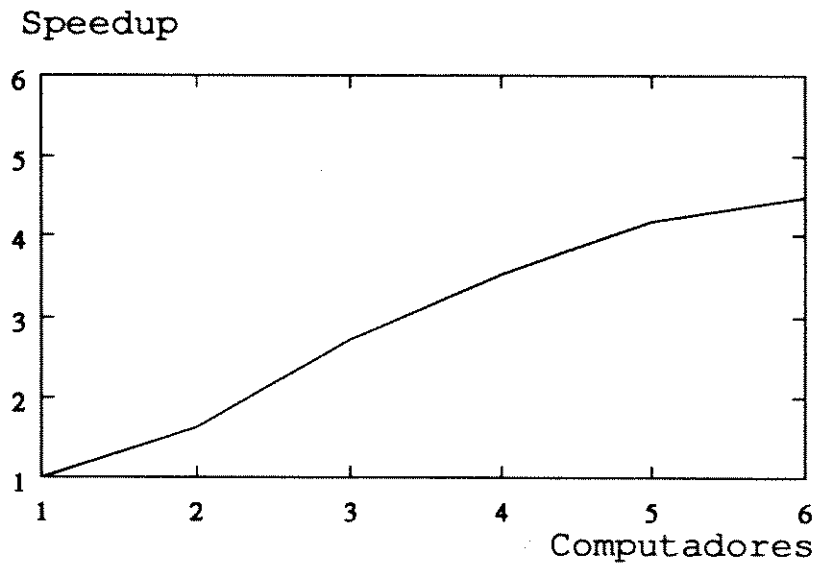


Figura 6.35: Curva do *speedup* do modelo *SPMD* no *PVM* com *broadcast*

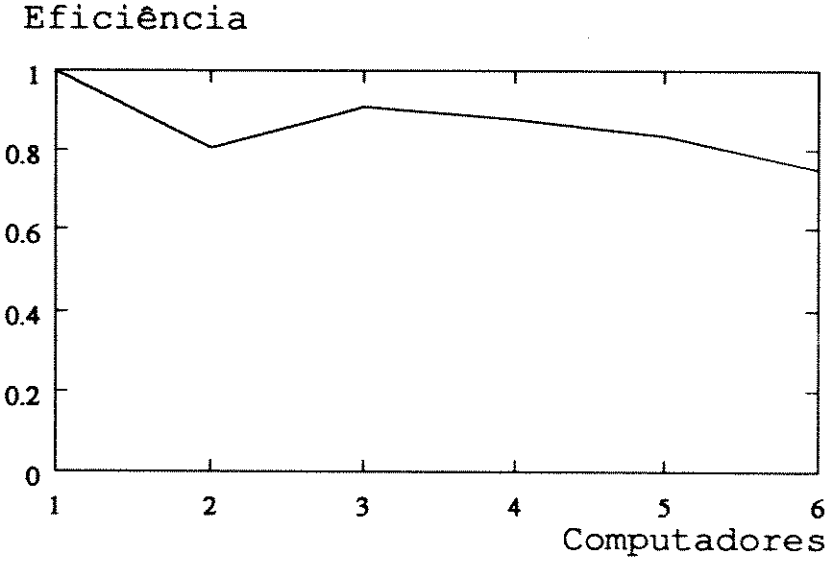


Figura 6.36: Curva da eficiência do modelo *SPMD* no *PVM* com *broadcast*

6.8 Otimização do Código Objeto

A Tabela 6.32 apresenta os tempos de processamento com a otimização do código objeto do Modelo $SPMD$ na rede de estações $IBM\ RISC6000$ com PVM,utilizada a opção $-O2$. A comparação com a Tabela 6.28 permite concluir que este procedimento possibilita um considerável ganho de tempo na computação da parte paralela.

Máq.	Inic.	I/O	Comun.	PS	PP	Tempo Total	*Speedup*
1	0,11	0,77	0,01	0,03	10,02	10,94	1,00
2	0,17	0,56	0,37	0,04	4,67	5,77	1,89
3	0,17	0,59	0,88	0,04	2,92	4,53	2,42
4	0,28	0,57	0,57	0,07	2,08	3,46	3,16
5	0,21	0,56	0,47	0,05	1,63	2,78	3,94
6	0,24	0,57	0,37	0,05	1,25	2,30	4,76

Tabela 6.32: Tempos obtidos para o sistema B no $SPMD$ com a otimização do código objeto

Capítulo 7

Comentários Finais

Neste trabalho abordou-se a paralelização de um programa de curto-circuito probabilístico, utilizando o método de *Monte Carlo*, em ambientes paralelo e distribuído. Foram implementadas técnicas avançadas para análises de sistema de potência para melhorar o desempenho do programa. Em ambos os ambientes foram implementados dois paradigmas de programação: o *SPMD* e o Mestre/Escravo. Os testes foram executados em quatro redes reais do Sistema Interligado da região Sul-Sudeste do Brasil e todos os resultados foram tomados para 100 000 simulações e cada resultado foi obtido da média de dez medidas.

Comentários:

- O uso de modelos apropriados e técnicas eficientes é necessário, mesmo quando se dispõe de recursos computacionais de alto desempenho, pois para estimativas precisas das curvas de densidade de probabilidade de curtos-circuitos (histogramas) são requeridas milhares de simulações. Neste trabalho foram utilizadas três técnicas, que são cruciais para melhorar o desempenho do programa computacional:

  1. Equivalente externo.
  2. Análise de curto-circuito em linhas de transmissão usando a formulação matricial do caso básico.
  3. Técnicas de matriz/vetor esparso para determinar as partes relevantes da matriz de impedância nodal envolvidas nos cálculos de curtos-circuitos.

- Devido ao fato de o modelo da rede ser linear, os resultados de simulações obtidos dos Sistemas A, B e C são idênticos, considerando-se a mesma região. Vale lembrar

que os Sistemas B e C foram obtidos pela redução apropriada do Sistema A, pelo método de equivalentes externos.

- Para análise de curto-circuito monofásico a consideração detalhada dos acoplamentos mútuos entre linhas de transmissão na rede de seqüencia zero é fundamental, para garantir histogramas com boa precisão.

- A utilização do método de vetores esparsos permite um ganho de tempo da ordem de cinco vezes no Sistema A (tempos obtidos no programa seqüencial), em relação ao método de matrizes esparsas.

- A polarização tem influência significativa na forma do histograma. Neste trabalho foram considerados dois tipos de distribuição de probabilidade dos locais dos curtos-circuitos.

- O número de simulações é fundamental na forma do histograma. 100 000 simulações apresentam boa precisão e foi adotado para a obtenção de todos os resultados.

- Uma das características importantes na programação paralela (e distribuída) é a portatibilidade, que é mais facilitada se o código original (programa seqüencial) estiver com bom grau de modularidade.

- As implementações das rotinas paralelas no $nCUBE2$ e no PVM, tanto no modelo de programação $SPMD$ como no Mestre/Escravo, aumentaram o código em cerca de 15% e 20% respectivamente.

- O tempo de comunicação no $nCUBE2$, no modelo $SPMD$ com *nglobal/nlocal*, cresce linearmente com o número de processadores (Figura 6.3), fazendo com que o *speedup* (Figuras 6.[illegible]) sofra uma uma queda significativa a partir de 16 nós. Isto se deve ao fato de esta configuração em particular não possuir os processadores I/O e conseqüentemente impossibilitando a paralelização da comunicação.

- O tempo de comunicação no $nCUBE2$, com *nbroadcast/nread* aumenta logaritmicamente com o número de nós (típico para este tipo de arquitetura). Por exemplo, para o Sistema A o tempo de comunicação é dado aproximadamente por $t_{com} = 0,515 \log_2(n)$ (Figura 6.6), onde n é o número de nós; isto significa que o tempo de comunicação para difundir as mensagens apresenta um aumento linear com a dimensão do subcubo. Isto se deve ao fato de o hipercubo permitir o paralelismo não só no processamento como também na comunicação, quando utiliza a primitiva *nbroadcast* para comunicação.

- A eficácia é uma das medidas de desempenho que descreve o grau de utilização dos processadores. As Figuras 7.1, 7.2 e 7.3 mostram respectivamente as curvas da eficácia para os modelos $SPMD$ (com *nglobal/nlocal* e *nbroadcast/nread*) e

Mestre/Escravo (com *nbroadcast/nread*). A análise destas curvas mostra que no modelo *SPMD* com *nglobal/nlocal* a máxima eficácia é atingida com 8 nós e nos modelos *SPMD* e Mestre/Escravo com *nbroadcast/nread* a máxima eficácia ocorre com 16 nós.

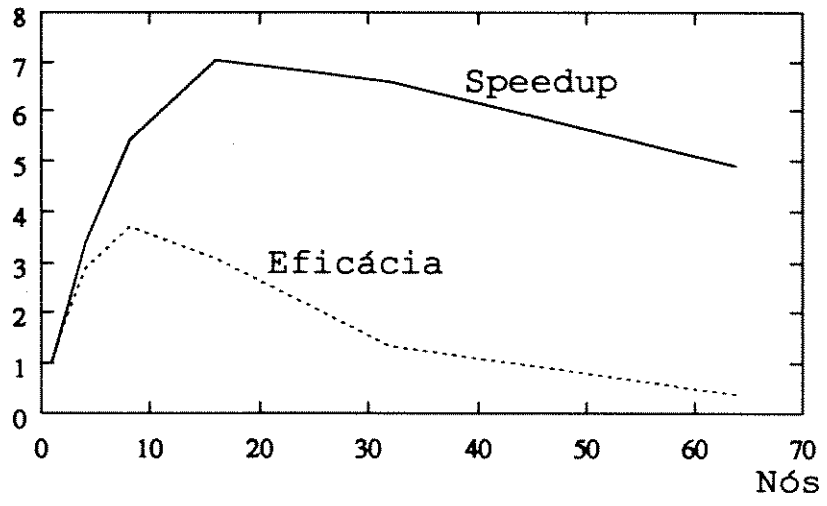


Figura 7.1: Curva da eficácia do modelo *SPMD* no *nCUBE2* com *nglobal/nlocal*

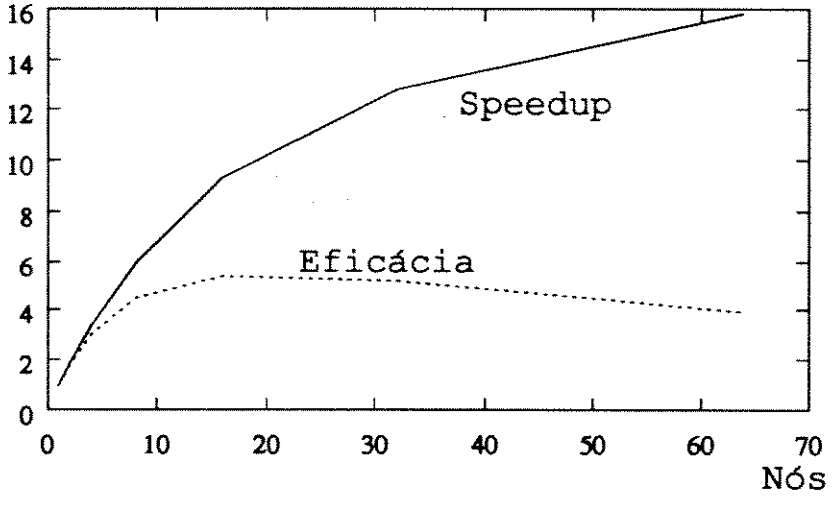


Figura 7.2: Curva da eficácia do modelo *SPMD* no *nCUBE2* com *nbroadcast/nread*

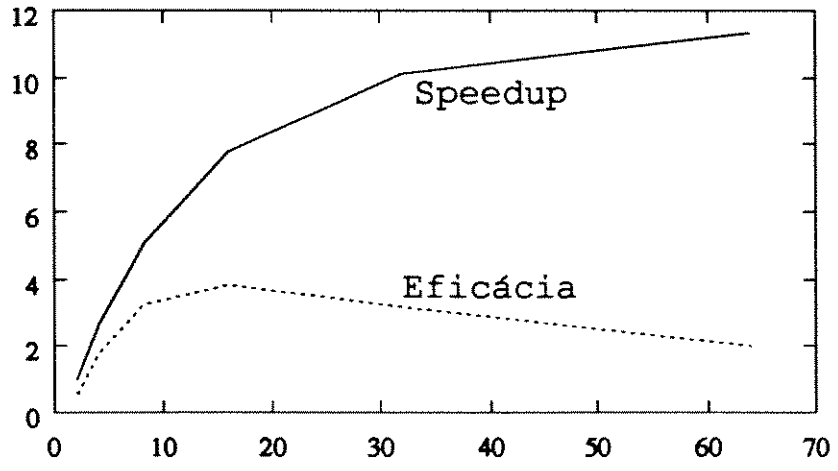


Figura 7.3: Curva da eficácia do modelo Mestre/Escravo no *nCUBE2* com *nbroadcast/ nread*

- No modelo *SPMD* o tempo de comunicação depende do tamanho do sistema, isto porque as mensagens difundidas são dados crús. Ao contrário, no modelo Mestre/Escravo o tempo de comunicação independe do tamanho do sistema, pois as mensagens difundidas são dados já preparados.

- O *PVM* pode rodar em ambientes heterogêneos, podendo envolver vários tipos de computadores. Uma característica importante do *PVM* é que ele não requer estações dedicadas para o processamento de uma aplicação. Assim sendo, várias aplicações podem rodar de forma concorrente na máquina paralela virtual.

- O desempenho da arquitetura distribuída (rede de estações provida com *PVM*) depende das cargas das estações, conforme pode-se observar nas Figuras 6.16 à 6.19, obtidas em processamentos com dois níveis diferentes de utilização da rede (livre e moderadamente ocupada).

- O tempo de comunicação na arquitetura distribuída (rede de estações provida com *PVM*) cresce de forma acentuada conforme se aumenta o número de estações (Figura 6.12), conseqüencia de um único canal de comunicação.

- Os programas com modelo *SPMD* utilizando ***broadcast*** (grupos dinâmicos de processos) e ***multicast*** apresentaram desempenhos similares.

- Tanto o computador paralelo *SP1* como a rede de estações *IBM RISC6000* tiveram o seu desempenho afetado pelo tempo de comunicação elevado, justificado pela solicitação intensa destes ambientes por parte de outras aplicações.

- A otimização do código objeto possibilita um considerável ganho no tempo de processamento.
- O modelo de programação $SPMD$ apresentou vantagens em relação ao modelo Mestre/Escravo em termos de portabilidade e de desempenho, em ambos os ambientes (paralelo e distribuído).

Como possíveis prosseguimentos deste trabalho, sugerimos os seguintes itens:

- Estudar e pesquisar a utilização do curto-circuito probabilístico em funcões de tempo-real em sistema de energia elétrica. Para isso é fundamental que sejam implementadas análises de contingências para detecção de condições críticas do sistema. Conforme abordado no Apêndice C as contingências simples, como a retirada de linhas, já são utilizadas em curto-circuito determinístico para determinação dos ajustes das proteções e dimensionamento de equipamentos.
- Explorar a arquitetura heterogênea ($nCUBE2$ e estações com PVM) implementando balanceamento automático de cargas (*pool* de tarefas).
- Efetuar a otimização do código objeto em todos os ambientes e avaliar o seu desempenho.

Bibliografia

[1] WARD, J. B. Equivalent circuit for power flow studies. *Transactions of the AIEE*, v. 68, p. 373-382, 1949.

[2] BROWN, H. E. et al. Digital calculation of three-phase short-circuit by matrix methods. *AIEE Power Apparatus Systems*, n. 52, p. 1277-1282, Feb. 1961.

[3] ANDERSON, J. G. Monte Carlo computer calculation of transmission-line lightning performance. *IEEE Power Apparatus and Systems*, n. 55, p. 414-420, Aug. 1961.

[4] TINNEY, W. F. & WALKER, J. W. Direct solution for sparse network equations by optimally ordered triangular factorization. *Proceeding of the IEEE*, v. 55 n.11, p. 1801-1809, Nov. 1967.

[5] BROWN, H. E. & PERSON, C. E. Short-circuit studies of large systems by the impedance matrix method. In: *IEEE POWER INDUSTRY COMPUTER APPLICATION CONFERENCE.* Pittisburgh, 1967. Proceeding...New York: IEEE, 1967. p. 335-342.

[6] DY LIACCO, T. E. & RAMARAO, K. A. Short-circuit calculation for multiline switching and end fault. *IEEE Transactions on Power Apparatus Systems*, v. 89, n. 6, p. 1226-1237, July/Aug. 1970.

[7] ZOLLENKOPF, K. Bi-fatorization basic computation algorithm and programming techniques. In: *REID, J. K. ed. Large sparse sets of linear equations.* New York: Academic Press, 1971, p. 75-97.

[8] FREUND, J. E. *Mathematical statistics.* Englewood Cliffs, NJ. Prentice-Hall, Inc., 1971.

[9] BARTHOLD, L. O. & PARIS, L. The probabilistc approach to insulation coordination. *Electra*, n.13, p. 41-58, 1971.

[10] FALCONE, C. A. et al. Current limiting device - a utility's need. *IEEE Transactions on Power Apparatus Systems*, v. 93, n. 6, p.1768-1775, Nov./Dec. 1974.

[11] TOSH, J. B. & RINDT, L. J. An elimination method for calculation short-circuit equivalent. In: *IEEE POWER INDUSTRY COMPUTER APPLICATION CONFERENCE, Minneapolis*, 1973. Proceeding...New York: IEEE, 1973. p. 383-385.

[12] TAKAHASHI, K. et al. Formation of a sparse bus impedance matrix and its application to short-circuit study. In: *IEEE POWER INDUSTRY COMPUTER APPLICATION CONFERENCE, Minneapolis*, 1973. Proceeding...New York: IEEE, 1973. p. 63-69.

[13] ZOLLENKOPF, K. Sparse nodal impedance matrix generated by the bi- fatorization method and applied to short-circuit studies. In: *POWER SYSTEM COMPUTER CONFERENCE, London*, 1975. Proceeding... London: 1975. p. 1-13.

[14] SHANNON, R. E. *Systems Simulation: the art and science.* Prentice-Hall, Inc., 1975.

[15] SHIMIZU, T. *Simulação em computador digital.* Edgard Blücher, 1975.

[16] BROWN, H. E. *Solution of large network by matrix methods.* New York, John Wiley and Sons, 1975.

[17] BROUSSOLE, F. State estimation on power systems: detecting bad data through the sparse inverse matrix methods. *IEEE Transactions on Power Apparatus Systems*, v. 97, n. 3, p. 678-682, May/June 1978.

[18] FORD, G. L. & SRIVASTAVA, K. D. Probabilistic short-circuit design of substation bus systems. In: *IEEE POWER ENGINEERING WINTER MEETING*, 1978.

[19] SATO, F. *Análise de curto-circuito em sistemas elétricos de potência.* Campinas: UNICAMP, mar. 1979. Tese de mestrado Faculdade de Engenharia de Campinas).

[20] SUZUKI, K. et al. Interative computation system on distance relay setting for large scale EHV power system. *IEEE Transactions on Power Apparatus and Systems*, v. 99, n. 1, p. 165-173, Jan./Feb. 1980.

[21] DECKMANN, S., PIZZOLANTE, A., MONTICELLI, A. et al. Numerical testing of power system load flow equivalent. *IEEE Transactions on Power Apparatus and Systems*, v. 99, n. 6, p. 2292-2300, Nov./Dec. 1980.

[22] DECKMANN, S., PIZZOLANTE, A., MONTICELLI, A. et al. Studies on power system load flow equivalencing. *IEEE Transactions on Power Apparatus and Systems*, New York, v. 99 n.6, p. 2301-2310, Nov./Dec. 1980.

[23] FORD, G. L. & SRIVASTAVA, K. D. The probabilistics approach to substation bus short-circuit design. *Electric Power System Research*, v. 4, n. 3, p. 191-200, 1981.

[24] FORD, G. L. & SENGUPTA, S. S. Analytical methods for probabilistic short-circuit studies. *Electric Power System Research*, v. 5, p. 13-20 ,1982.

[25] MONTICELLI, A. *Fluxo de carga em redes de energia elétrica.* São Paulo: Edgard Blücher, 1983.

[26] EL-KADY, M. A. & FORD, G. L. An advanced probabilistic short-circuit program. *IEEE Transactions on Power Apparatus and Systems*, v. 102, p. 1240-1247, 1983.

[27] EL-KADY, M. A. Probabilistic short-circuit analysis by Monte Carlo simulations. *IEEE Trans. Power Apparatus and Systems*, v. 102, p. 1308-1315, 1983.

[28] TINNEY, W. F., BRANDWAJN, V. & CHAN, S. M. K., Sparse vector method. *IEEE Transactions on Power System*, v. 104, No. 2, p. 295-301, Feb. 1985.

[29] METROPOLIS, N. Monte Carlo: In the beginning and some great expectations. *Lecture Notes in Physics, Monte Carlo Methods and Applications in Neutronics, Photonics and Statistical Physics*, v. 240, p. 62-70, Apr. 22-26, 1986.

[30] HAVARD, D. G. et al. Probabilistic short-circuit uprating of station strain bus system: mechanical aspects. *IEEE Transactions on Power Delivery*, v. 1, n. 3, p. 104-110, July 1986.

[31] GERMANI, M. D. et al. Probabilistic short-circuit uprating of station strain bus system: overview and application. *IEEE Transactions on Power Delivery*, v. 1, n. 3, p. 111-117, July 1986.

[32] BALOUKTSIS, A. et al. Probabilistic short-circuit analisys by Monte Carlo simulations and analytical methods. *IEEE Transactions on Power Systems*, v. 1, n. 3, p. 135- 141. Aug. 1986.

[33] VAINBERG, M. et al. Probabilistic short-circuit uprating of station strain bus system: probabilistic formulation. *IEEE Transactions on Power Delivery*, v. 1, n. 4, p. 129-136. Oct. 1986.

[34] RAMANI, N. et al. Probabilistic short-circuit uprating of station strain bus system: reliability and operating considerations. *IEEE Transactions on Power Delivery*, v. 1, n. 4, p. 137-141, Oct. 1986.

[35] QUINN, M. *Designing efficient algorithms for parallel computer.* Ed. McGraw-Hill,1987.

[36] HWANG, K. Advanced parallel processing with supercomputer architectures. *Proceedings of the IEEE*, v. 75, n. 4, p. 1348-1379, Oct. 1987.

[37] NEELAMKAVIL, F. *Computer simulation and modelling.* .New York:John Wiley, 1988.

[38] KATSEFF, H. P. Incomplete Hypercube. *IEEE Transactions on Computer*, v. 37, n. 5, p. 604-608, May 1988.

[39] CASTRO Jr, C. Métodos de solução local para análise de contingências. *UNICAMP/FEE/DSEE*, jun. 1988. (Relatório interno).

[40] SAAD Y. et al. Topological properties of hypercubes. *IEEE Transactions on Computer*, v. 37, n. 7, p. 867-872, july 1988.

[41] BERTSEKAS, D. P. & TSITSIKLIS, J. N. *Parallel and distributed computation:numerical methods.* New York: Prentice-Hall,1989.

[42] DUNCAN, R. A survey of parallel computer architectures. *Computer*, v. 23 , n. 2, p. 5-16, Feb. 1990.

[43] NUNES, R. A. F. & COUTINHO, I.P. Curto-circuito probabilístico: técnicas e aplicações. In: *III SEMINÁRIO TÉCNICO DE PROTEÇÃO E CONTROLE*, 1991, Rio de Janeiro. Anais... Rio de Janeiro : FURNAS, 1991.

[44] ENNS, M. K. & QUADA, J. J. Sparsity-enhanced network reduction for fault studies. *IEEE Transactions on Power System*, v. 6, n. 2, p. 613-621, May 1991.

[45] ALMEIDA, V. A. F. & ÁRABE, J. N. C. *Introdução à supercomputação.* Rio de Janeiro: Livros Técnicos e Científicos, 1991.

[46] nCUBE *nCUBE 2 Programmer's guide - release 3.0 version.* California, 1992.

[47] POUNTAIN, D. & BRYAN, J. All systems go. *Byte*, v. 17, n. 8, p. 112-136, Aug. 1992.

[48] EL-REWINI, H. et al. From theory to practice. *IEEE Parallel and Distributed Technology, System and Applications.* v. 1, n. 3, p. 7-11, Aug. 1993.

[49] CALZAROSSA, M. & SERRAZI, G. Workload characterization: A survey. *Proceedings of the IEEE.* v. 81, n. 8, p. 1136-1150, Aug. 1993.

[50] GEIST, A. et al. *PVM 3 User's guide and reference manual.* Oak Ridge: Oak Ridge National Laboratory, may 1994. (ORNL/TM-12187)

[51] SATO, F., GARCIA. A. V. & MONTICELLI, A. Parallel implementation of probabilistic short-circuit analysis by Monte Carlo approach. *IEEE Transaction on Power System.* v. 9, n. 2, p. 826-832, May 1994.

Apêndice A

Curtos-circuitos em Linhas de Transmissão

A.1 Curto-circuito Trifásico

A.1.1 Corrente de Curto-circuito Trifásico no Ponto f

$$I_{3f} = \frac{-1,0}{Z_f^+} \tag{A.1}$$

A.1.2 Tensões nas Barras

O perfil real das tensões nas barras é calculado por:

$$E_i^R = 1,0 - E_i \; i = 1,2,.....,n \tag{A.2}$$

A.2 Curto-circuito Monofásico

A.2.1 Corrente de Curto-circuito Monofásico no Ponto *f*

$$I_{1f} = \frac{-3,0}{2Z_f^+ + Z_f^0} \tag{A.3}$$

As correntes de seqüências no ponto f são:

$$I_f^+ = I_f^- = I_f^0 = \frac{-1,0}{2Z_f^+ + Z_f^0} \tag{A.4}$$

A.2.2 Tensões de Sequências Positiva e Zero no Ponto *f*

$$E_f^{R+} = \frac{Z_f^+ + Z_f^0}{2Z_f^+ + Z_f^0} \tag{A.5}$$

$$E_f^{R0} = \frac{-Z_f^0}{2Z_f^+ + Z_f^0} \tag{A.6}$$

A.2.3 Tensões de Sequências Positiva e Zero na Barra

O perfil real das tensões de seqüência positiva é calculado por:

$$E_i^{R+} = 1,0 - E_i^+ \qquad i = 1, 2, ..., n \tag{A.7}$$

e de seqüência zero por:

$$E_i^{0+} = E_i^0 \qquad i = 1, 2, ..., n \tag{A.8}$$

A.3 Fluxos de Corrente nas Linhas

As Equações (3.43), (3.44) e (3.45) são utilizadas para os cálculos dos fluxos das correntes nas linhas, porém em termos computacionais é vantajoso apresentá-las com outra formulação, como desenvolvido a seguir.

As tensões de malha podem ser expressas em função das tensões nodais, isto é:

$$V_{pq} = E_p - E_q \tag{A.9}$$

$$V_{rs} = E_r - E_s \tag{A.10}$$

Substituindo as tensões nodais nas Equações (A.9) e (A.10) pelos elementos correspondentes da Equação (3.56) tem-se:

$$V_{pq} = Z_{pq}(1-a)I_f + Z_{ps}aI_f - Z_{qr}(1-a)I_f - Z_{qs}aI_f \tag{A.11}$$

$$V_{rs} = Z_{rr}(1-a)I_f + Z_{rs}aI_f - Z_{sr}(1-a)I_f - Z_{ss}aI_f \tag{A.12}$$

Simplificando:

$$V_{pq} = [(Z_{pq} - Zqr)(1-a) + (Z_{ps} - Z_{qr})a]I_f \tag{A.13}$$

$$V_{pq} = [(Z_{rr} - Zsr)(1-a) + (Z_{rs} - Z_{ss})a]I_f \tag{A.14}$$

ou

$$V_{pq} = Z^*_{pq}I_f \tag{A.15}$$

$$V_{rs} = Z^*_{rs}I_f \tag{A.16}$$

Substituindo as Equações (A.15) e (A.16) nas Equações (3.43), (3.44) e (3.45) tem-se:

$$I_{pq} = y_{pq} Z^*_{pq} I_f + y_{pq,rs} Z^*_{rs} I_f \tag{A.17}$$

$$I_{rf} = y_{rs,pq} Z^*_{pq} I_f + y_{rs} Z^*_{rs} I_f - (1-a) I_f \tag{A.18}$$

$$I_{sf} = -y_{rs,pq} Z^*_{pq} I_f + y_{rs} Z^*_{rs} I_f - a I_f \tag{A.19}$$

As Equações (A.17), (A.18) e (A.19) possibilitam calcular as contribuições tanto para curto-circuito monofásico como para curto-circuito trifásico, sendo que para o segundo caso não existirão os elementos $y_{pq,rs}$ e $y_{rs,pq}$.

Na Equação (A.17) I_{pq} pode ser fluxo de corrente em uma linha ou em um grupo de linhas mutuamente acopladas com a linha rs.

Apêndice B

Equivalentes Externos

B.1 Obtenção da Rede Equivalente

O sistema interligado a ser analisado é dividio em três sub-sistemas: interno (I), fronteira (F) e externo (E). Considerando-se os três sub-sistemas tem-se a Equação (B.1), em que a matriz de admitância nodal é formada por matrizes particionadas.

$$\begin{bmatrix} Y_{EE} & Y_{EF} & \\ Y_{FE} & Y_{FF} & Y_{FI} \\ & Y_{IF} & Y_{II} \end{bmatrix} \cdot \begin{bmatrix} \underline{V}_E \\ \underline{V}_F \\ \underline{V}_I \end{bmatrix} = \begin{bmatrix} \underline{I}_E \\ \underline{I}_F \\ \underline{I}_I \end{bmatrix} \tag{B.1}$$

Nesta Equação serão eliminadas as variáveis do vetor $\underline{V}_E$, o que equivale a eliminar todos os nós do sub-sistema externo (E). Isolando $\underline{V}_E$ e substituindo nas demais Equações tem-se

$$(Y_{FF} - Y_{FE}^{-1} Y_{EE} Y_{EF})\underline{V}_E + Y_{FI}\underline{V}_I = \underline{I}_F - \underline{I}_E Y_{FE}^{-1} Y_{EE} \tag{B.2}$$

e

$$(Y_{IF}\underline{V}_F + Y_{II}\underline{V}_I) = \underline{I}_I \tag{B.3}$$

ou

$$\begin{bmatrix} Y_{FF} - Y_{FE}^{-1}Y_{EE}Y_{EF} & Y_{FI} \\ Y_{IF} & Y_{II} \end{bmatrix} \cdot \begin{bmatrix} \underline{V}_F \\ \underline{V}_I \end{bmatrix} = \begin{bmatrix} \underline{I}_F - \underline{I}_E Y_{FE}^{-1}Y_{EE} \\ \underline{I}_I \end{bmatrix} \tag{B.4}$$

sendo

$$(Y_{FF} - Y_{FE}Y_{EE}^{-1}Y_{EF}) = Y_{FF}^{eq} \tag{B.5}$$

A Equação (B.5) é conhecida como redução de *Kron* e nela estão refletidas todas as mudanças na rede elétrica provenientes da eliminação dos barras externas. Normalmente ela é pouco esparsa, significando que existirão ligações equivalentes entre quase todas as barras de fronteira, entretanto, as que apresentarem baixa admitância poderão ser ignoradas sem que isso influencie nos valores dos cálculos de curto-circuito. Note-se após o processo de substituição as partições Y_{FI}, Y_{IF} e Y_{II} permaneceram inalteradas.

Uma maneira sistemática de se obter Y_{FF}^{eq} é a eliminação de *Gauss* na partição da matriz referentes às barras externas Y_{EE} e como as sub-matrizes envolvidas Y_{EE}, Y_{EF}, Y_{FE} e Y_{FF} são esparsas, a eliminação pode ser efetuada usando-se tanto as técnicas de fatoração triangular com a bi-fatoração.

Após a eliminação tem-se:

$$\begin{bmatrix} Y_{EE}^{eq} & Y_{EF}^{eq} & \\ & Y_{FF}^{eq} & Y_{FF} \\ & Y_{IF} & Y_{II} \end{bmatrix} \tag{B.6}$$

As sub-matrizes Y_{EE}^{eq} e Y_{EF}^{eq} contém os fatores triangulares resultantes da eliminação das barras externas, enquanto que as demais sub-matrizes representam a rede reduzida.

As ligações equivalentes são geradas operando-se com os elementos das sub-matrizes Y_{EE}^{eq} e Y_{FI}.

Lembrando-se da formação da matriz Y_{BARRA}, em que o elemento ii (diagonal) é a somatória de todas as admitâncias ligadas à barra i e que o elemento ij (fora da diagonal) é o valor negativo da admitância ij, as admitâncias equivalentes *shunt* e série são obtidas respectivamente pelas equações:

$$y_{0i}^{eq} = Y_{ii} + \sum_{j=1}^{k} Y_{ij} \qquad k = NE + 1 \quad e \quad j \neq i \tag{B.7}$$

e

$$y_{ij}^{eq} = -Y_{ij} - y_{ij} \tag{B.8}$$

sendo y_{ij} a admitância da linha ij.

Apêndice C

Análise de Contingências

No estudo de curto-circuito, além do caso básico, é necessário simular a retirada ou adição de linhas de transmissão. Estas contingências, por alterarem a configuração da rede, afetam todos os elementos da matriz Z_{BARRA}. Entretanto, pode-se recalcular apenas os elementos relevantes da matriz, isto é, somente a impedância no ponto e as impedâncias de transferência que são utilizados na análise, evitando assim recalcular elementos desnecessários.

C.1 Retirada de uma linha

Seja:

$$\underline{I} = Y_{BARRA}\underline{E} \tag{C.1}$$

Qualquer modificação na impedância de um ramo irá provocar uma variação na matriz de admitância Y_{BARRA}, passando a ser:

$$Y'_{BARRA} = Y_{BARRA} + \Delta Y' \tag{C.2}$$

Seja a retirada de uma linha com admitância y_{rs}, ligada entre as barras r e s.

A variação $\Delta Y'$ pode ser escrita sob a forma:

$$\Delta Y' = -y_{rs}\underline{v}\underline{v}^t \tag{C.3}$$

onde

$$\underline{v} = \begin{bmatrix} 0 \\ \vdots \\ +1 \\ \vdots \\ -1 \\ \vdots \\ 0 \end{bmatrix} \tag{C.4}$$

e

$$\underline{v}^t = [0 \cdots +1 \cdots -1 \cdots 0] \tag{C.5}$$

Portanto,

$$Y'_{BARRA} = Y_{BARRA} + (-y_{rs}\underline{v}\underline{v}^t) \tag{C.6}$$

Aplicando a fórmula de *Woodbury* na Equação (C.6) vem:

$$(Y'_{BARRA})^{-1} = Y^{-1}_{BARRA} - \frac{(Y^{-1}_{BARRA}\underline{v})(\underline{v}^t Y^{-1}_{BARRA})}{-\frac{1}{y_{rs}} + \underline{v}^t Y^{-1}_{BARRA}\underline{v}} \tag{C.7}$$

ou

$$(Z'_{BARRA}) = Z_{BARRA} - \frac{(Z_{BARRA}\underline{v})(\underline{v}^t Z_{BARRA})}{-z_{rs} + \underline{v}^t Z_{BARRA}\underline{v}} \tag{C.8}$$

onde

$$Z_{BARRA}\underline{v} = \underline{c} \tag{C.9}$$

ou

$$\underline{c} = \begin{bmatrix} Z_{1r} - Z_{1s} \\ \vdots \\ Z_{rr} - Z_{rs} \\ \vdots \\ Z_{sr} - Z_{ss} \\ \vdots \end{bmatrix} \tag{C.10}$$

e

$$\underline{v}^t Z_{BARRA} = \underline{l} \tag{C.11}$$

ou

$$\underline{l} = [(Z_{rl} - Z_{sl}) \cdots (Z_{rr} - Z_{rs}) \cdots (Z_{rs} - Z_{ss}) \cdot] \tag{C.12}$$

e

$$\underline{v}^t Z_{BARRA}\underline{v} = Z^{eq} \tag{C.13}$$

ou

$$Z^{eq} = (Z_{rr} - Z_{rs}) - (Z_{rs} - Z_{ss}) \tag{C.14}$$

Portanto,

$$(Z'_{BARRA}) = Z_{BARRA} - \frac{\underline{c}\underline{l}}{Z^{eq} - z_{rs}} \tag{C.15}$$

Para recalcular o elemento ij da nova matriz o algoritmo será:

$$Z'_{ij} = Z_{ij} - \frac{(Z_{ir} - Z_{is})(Z_{rj} - Z_{sj})}{Z_{rr} + Z_{ss} - 2Z_{rs} - z_{rs}} \tag{C.16}$$

No caso da linha retirada estar mutuamente acoplada com outras linhas o algoritmo para recalcular o elemento ij da nova matriz é dado por:

$$Z'_{ij} = Z_{ij} - \frac{C_i C_j}{P} \tag{C.17}$$

O desenvolvimento deste algoritmo pode ser encontrado com detalhes nas Refêrencias [6] e [19].