UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE MÁQUINAS, COMPONENTES E SISTEMAS INTELIGENTES

PROCESSAMENTO PARALELO DE ALGORITMOS DE CONTROLE HIERÁRQUICO

AUTOR : JOSÉ TARCISIO COSTA FILHO
ORIENTADOR : PROF. DR. CELSO PASCOLI BOTTURA

Tese apresentada a Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas- UNICAMP - como parte dos requisitos exigidos para obtenção do título de Mestre em Ciências.

Este exemplar corresponde à redação final da tese defendida por José Tarcisio Costa Filho e aprovada pela comissão julgadora em 29/7/1988.

Celso Pascoli Bottura

- julho de 1988 -

Aos

Meus Pais, Tarcisio e Maria do Carmo

Meus Irmãos, Gercina, Helena, Carlos

e Tadeu

AGRADECIMENTOS

Agradeço a todas as pessoas que direta ou indiretamente contribuiram para a realização deste trabalho.

Em especial, quero destacar meu orientador e amigo Prof. Dr. Celso Pascoli Bottura, pelo incentivo e orientação que transcendem este trabalho.

A Fundação Núcleo de Tecnologia Industrial, em especial, aos engenheiros Decio Pinheiro e Pedro Urbano.

Aos engenheiros do Centro Tecnológico para Informática - CTI, em especial, Paulo Valente, Ailton Santa Barbara, Adilson Lopes e Marcius F. H. de Carvalho.

Aos engenheiros do Centro de Pesquisa e Desenvolvimento da Telebrás - CPqD, em especial, José Henrique Zilberberg e Artur Pestana.

Aos engenheiros do Centro de Pesquisa de Energia Elétrica - CEPEL, em especial, Mario J. Teixeira.

Ao Laboratório Nacional de Computação Científica-LNCC na pessoa do professor e amigo Augusto Cesar Gadelha pelo apoio e incentivo.

Aos amigos Noritsuna Furuya, Gilmar Barreto, Marconi K. Madrid pelo apoio.

PROCESSAMENTO PARALELO DE ALGORTIMOS

DE CONTROLE HIERARQUICO

R E S U M O

Neste trabalho estudamos e alteramos a estrutura de cálculo de algoritmos de controle hierárquico com a finalidade de obter procedimentos de paralelização que permitam implementação eficiente em arquiteturas de múltiplos processadores, bem como realizamos experimentos em processamento paralelo destes algoritmos.

A B S T R A C T

In this work we have studied and changed the calculation structure of hierarchical control algorithms with the objective of developing parallelization procedures allowing the efficient implementation in multiprocessors architecture, as well as we made experiments the parallel processing of these algorithms.

INTRODUÇÃO

A computação e a implementação de algoritmos de controle ótimo hierárquico em sistemas de múltiplos processadores têm recebido pouca atenção, apesar ser grande o número de estudos teóricos sobre estes algoritmos. No entanto, o progresso na tecnologia de microcomputadores e o enorme aumento de suas aplicações em controle têm provocado o desenvolvimento da metodologia de controle hierárquico na prática de automação.

As metodologias de controle hierárquico são bastante atraentes para o processamento paralelo, uma vez que o paralelismo natural dos algoritmos multiníveis possibilitam suas execuções numa arquitetura de múltiplos microprocessadores.

Neste trabalho, analisamos e implementamos algoritmos multiníveis sobre sistemas de múltiplos processadores especificados, explorando numa primeira etapa o paralelismo natural na solução de problemas de controle ótimo. Esta implementação tem apresentado um bom desempenho computacional com relação ao tempo de processamento global quando comparada com a implementação sequencial dos algoritmos. No entanto um maior grau de paralelismo pode ser obtido, mantendo uma dependência com a arquitetura utilizada e por conseguinte melhorando o desempenho computacional dos algoritmos com relação as implementaçõs previamente mencionadas. Isto nos leva a propor uma segunda etapa que implica na modificação da estrutura de cálculo dos algoritmos. As modificações introduzidas e as análises das implementações sobre as arquíteturas de múltiplos microprocessadores utilizadas bem como os resultados obtidos da aplicação das metodologias de otimização em problemas de controle são apresentadas. No capítulo 1, descrevemos os

sistemas de múltiplos processadores utilizados. No capítulo 2, apresentamos os ambientes de programação paralela que permitem utilizar a multiprogramação no estudo de algoritmos paralelos. No capítulo 3, apresentamos as metodologias de otimização e definimos procedimentos de paralelização. No capítulo 4, fazemos análises de desempenho computacional das implementações dos algoritmos, descrevemos as formas de programação e implementações dos algoritmos e apresentamos resultados numéricos de aplicações. No capítulo 5, realizamos a conclusão do trabalho.

CAPITULO 1 - SISTEMAS DE MÚLTIPLOS PROCESSADORES

1.1 - Introdução

A maioria dos usuários de computador acredita que o programador não necessita ter conhecimentos do hardware que está utilizando. No entanto, quando o desempenho computacional torna-se crítico, os programadores tem usado seus conhecimentos de áreas físicas de memória, pilha e assim por diante para otimizar a velocidade de execução de seus programas. A situação é mais significativa para sistemas de processamento paralelo do que para sistemas de computação convencional (processamento sequencial em uma única unidade de processamento), devido a grande variedade de arquiteturas existentes. Há muitas pesquisas em andamento relativas a como organizar sistemas de processamento paralelo, bem como a utilizar eficientemente as unidades de processamento na solução de uma aplicação.

Uma alternativa para sistemas de processamento paralelo é o sistema de múltiplos processadores. A possibilidade de utilização de arquiteturas de múltiplos processadores como base para um sistema de computação eficaz e poderoso é bastante atraente por várias razões.

a) um sistema de múltiplos processadores pode obter um aumento significativo na velocidade de processamento por permitir a paralelização na estrutura dos algoritmos ;

b) uma arquitetura de múltiplos processadores oferece a potencialidade de se obter um sistema confiável através da redundância de suas unidades de

processamento ;

c) o custo de processadores tem sido reduzido, conseqüentemente, o custo de se adicionar processadores a um sistema de múltiplos processadores torna-se menos significativo em relação ao custo de sistemas tradicionais de maior porte.

Um sistema de múltiplos processadores, basicamente, consiste de P processadores mais interconexões para a troca de dados (comunicação) e para o controle de informações (sincronização) entre os processadores com o objetivo dos P processadores executarem diferentes partes da computação, tal que o processamento total seja realizado em alta velocidade pelo uso eficiente dos P processadores. Fica claro, portanto, que o emprego de sistemas de múltiplos processadores tem um efeito significativo sobre a metodologia do software que deve ser considerada na determinação de um ambiente de programação, como também, na estrutura de uma aplicação. Quando comparamos as experiências que realizamos, utilizando sistemas de monoprocessadores e algoritmos sequenciais, verificamos que muito pouco é sabido sobre os problemas relacionados com o emprego de múltiplos processadores e programação paralela.

1.2 - Classificação de sistemas de computação

Sistemas de múltiplos processadores fazem parte da classe de sistemas de processamento paralelo. Muitas taxonomias têm sido publicadas sobre esse assunto. Entre estas, a mais representativa na literatura foi a proposta por Flynn [20] que introduziu, baseado no fluxo de controle

e no fluxo de dados, as três classes seguintes de arquiteturas de computação :

1. SISD "Single-Instruction Stream, Single-Data Stream".
   Computador serial com único fluxo de instruções (seqüencia de operações que é executada num processador) e único fluxo de dados. Esta arquitetura compreende os computadores convencionais monoprocessadores. O termo serial é empregado para designar esta classe de computadores.

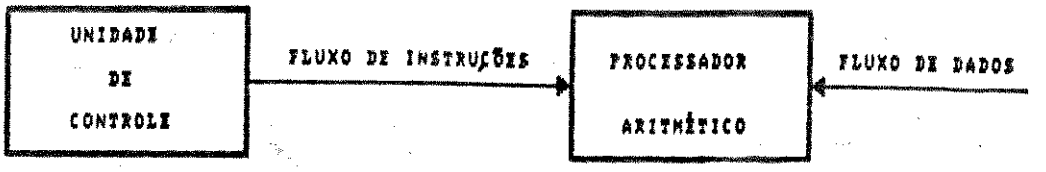


Figura 1.1 Arquitetura SISD

2. SIMD "Single-Instruction, Multiple-Data Stream" (único fluxo de instruções-múltiplos fluxos de dados).
   Esta classe é representada pelos processadores vetoriais. Todos os processadores executam a mesma instrução, fornecida pela unidade de controle, sobre vetores de dados, geralmente diferentes, como unidade de informação. A execução das instruções é síncrona no sentido de que cada processador executando uma instrução em paralelo deve terminá-la antes que a próxima instrução seja fornecida para execução.

3. MIMD "Multiple-Instruction Stream,Multiple-data Stream" (múltiplos fluxos de instruções, múltiplos fluxos de dados).

   Esta arquitetura combina paralelismo nos fluxos de dados e de instruções;é composta de unidades de processamento, cada qual capaz de realizar operações lógicas e aritméticas padrões. As unidades operam assincronamente sob o controle de fluxo de instruções individuais. Durante a computação as unidades comunicam resultados a outras unidades. Vários modelos de comunicação entre os processadores para computadores MIMD têm sido propostos na literatura. Esta classe de sistema de computação compreende os sistemas de múltiplos processadores.

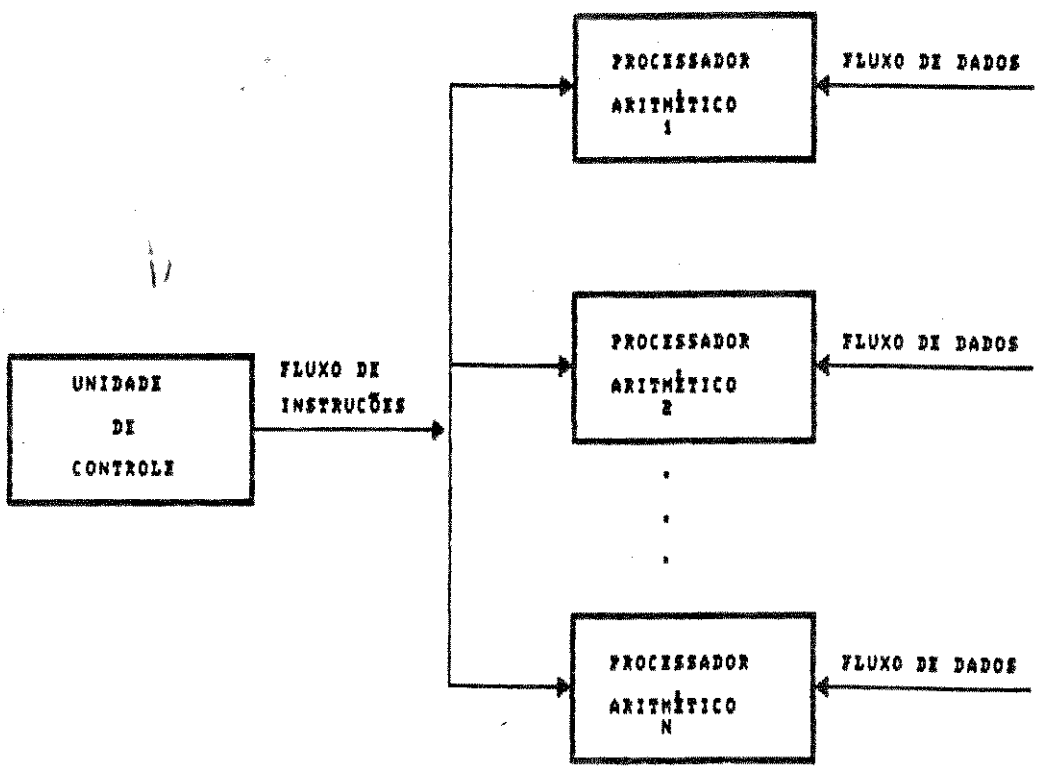


Figura 1.2 Arquitetura SIMD

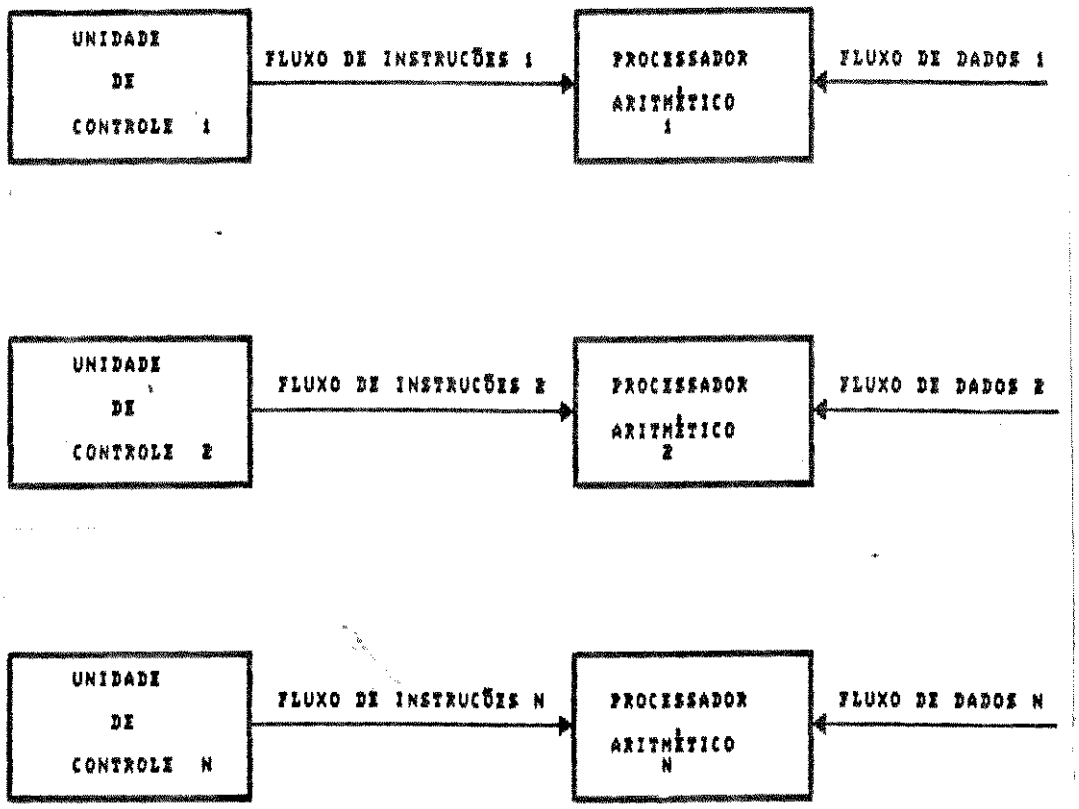


Figura 1.3 Arquitetura MIMD

Uma outra forma de classificação [21] de tipos de sistemas de processamento paralelo considera a frequência com que as unidades de processamento se comunicam e a quantidade de informações que são trocadas entre as atividades que são executadas em paralelo pelo sistema. Neste contexto, a classificação especifica :

1) redes distribuídas de computadores (cada unidade de processamento detem uma forte autonomia e dedica somente uma parte limitada da potência de processamento para as atividades comuns);

2) sistemas de múltiplos processadores (sistemas fracamente acoplados - processadores interligados por linhas seriais ou paralelas e sistemas fortemente acoplados - processadores podem acessar uma mesma área de dados);

3) computadores de proposta especial (sistemas de processamento projetados para resolverem problemas específicos - "systolic array", processadores vetoriais e outras estruturas).

Estas e outras classificações não oferecem esquemas precisos sobre os quais seja possível inserir ineqüivocamente uma arquitetura conhecida. Por exemplo, na primeira classificação, todos os sistemas de múltiplos processadores pertencem a classe MIMD independente das características dos processadores, no entanto, o computador Cray X-MP é composto de duas unidades de processamento fortemente acopladas onde cada unidade é um processador "pipeline", e Flynn [20] considera tal processador como pertencente a classe SIMD. No caso de computadores tradicionais que atualmente apresentam canais DMA ("Direct Memory Acess"), tais canais são processadores independentes bastante simples que operam sobre fluxos de entrada e de saída de dados com um reduzido repertório de instruções. Então, um sistema de computação com um processador e um canal DMA ajusta-se ao modelo de um sistema de múltiplos processadores. Com relação a segunda classificação, por exemplo, num sistema de múltiplos processadores, o nível de cooperação entre as atividades não é somente definido pela arquitetura do sistema, mas também pelo sistema operacional, possivelmente distribuído, ou pela própria aplicação.

Melhor do que tentar uma classificação completa para inserir as arquiteturas que utilizamos em esquemas pre-

cisos, nos restringimos a classificação de sistemas de processamento paralelo em relação aos seguintes aspectos :

1) Unidades de processamento de proposta geral - unidades capazes de executar uma grande variedade de aplicações com razoável eficiência. Computadores de proposta especial são projetados para resolver um tipo de aplicação e freqüentemente necessitam de operações específicas ;

2) Arquitetura de computação pertencente a classe MIMD - nesta organização a topologia de interconexão e a estratégia de comunicação entre as unidades de processamento é um ponto crítico do sistema de processamento, e uma classificação mais detalhada deve ser baseada na estrutura da rede de interconexão. Vários tipos de redes de interconexão para a arquitetura de computação MIMD têm sido propostos na literatura ;

3) Sistema de computação moderadamente paralelo - sistemas com no máximo dez unidades de processamento ;

4) Sistema operacional monoprocessador (UNIX, MS-DOS, VAX/VMS e outros) - na maioria dos casos a utilização de sistemas operacionais monoprocessadores tem gerado desempenho aceitável quando executado sobre um número reduzido de processadores. Para um usuário executar um conjunto de programas sobre diferentes unidades de processamento, é importante que o sistema operacional organize e gere mecanismos para permitir a execução eficiente dos programas concorrentes. A maioria dos sistemas operacionais comerciais não possue ainda tais facilidades ;

5) Linguagem de programação Pascal - linguagem bem adaptável para programação científica. Devido as extensões presentes em algumas versões da linguagem, o paralelismo de um algoritmo pode ser razoavelmente expressado. Outras linguagens, tais como Fortran, C podem ser utilizadas.

6) Paralelismo explicitamente declarado - o paralelismo é controlado pelo programador que se encarrega de distribuir os programas pelas unidades de processamento.

Sob tais aspectos,classificamos sistemas de processamento paralelo em duas classes :

a) Sistemas de múltiplos processadores com memória compartilhada.

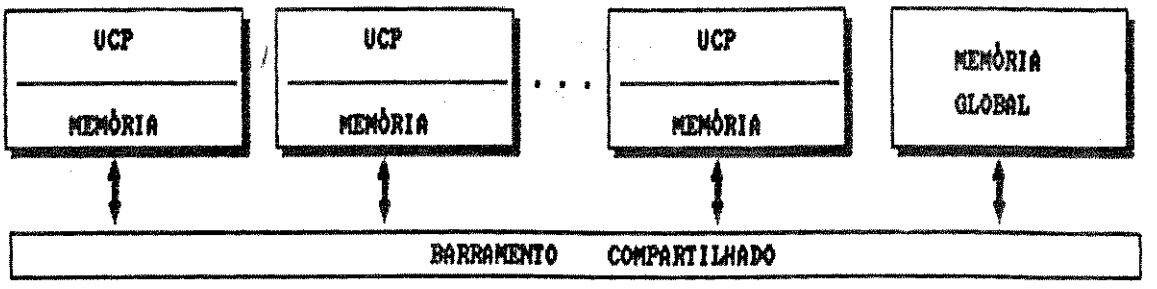


Figura 1.4 - Arquitetura do sistema com memória compartilhada

b) Sistemas de múltiplos processadores com troca de mensagens.

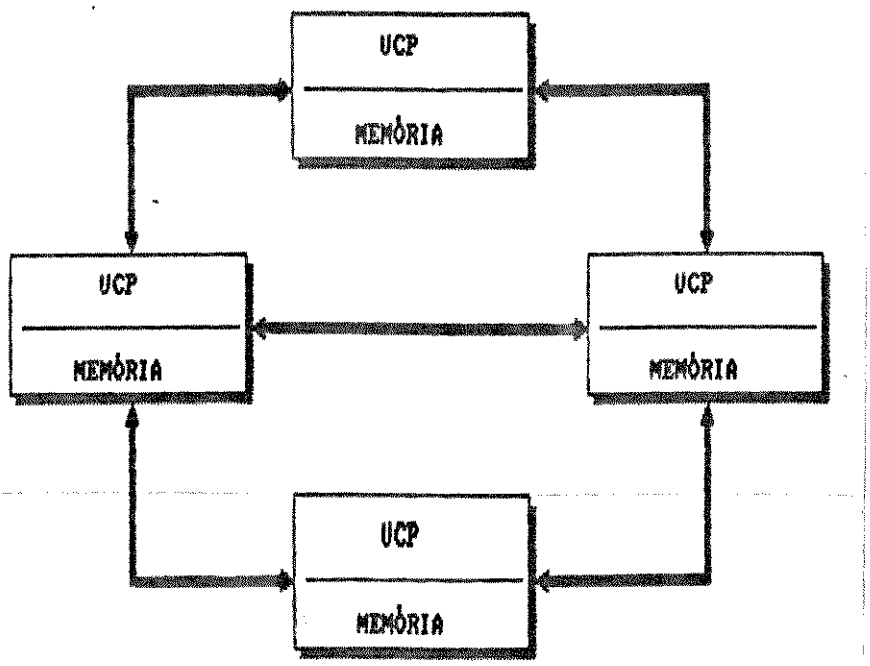


Figura 1.5 - Arquitetura do sistema com troca de mensagens fortemente interconectado

Não esperamos que esta simples classificação descreva a variedade de arquiteturas de sistemas de múltiplos processadores, mas ela é suficiente para possibilitar diferentes soluções em processamento paralelo, utilizando organização e estrutura de computadores de custo reduzido e mais facilmente disponíveis.

1.3. - Sistemas de Múltiplos Processadores com Memória Compartilhada

Um sistema com memória compartilhada tem uma memória global acessível a todas as unidades de processamento (UCP).

A cada UCP é associada uma memória local somente acessível pela UCP. Esta memória retem, por exemplo, os programas que cada UCP executará e o conjunto de dados não compartilhados. Um barramento compartilhado é a única via de comunicação na qual as unidades funcionais (memória global, UCP e etc.) estão conectadas, conforme figura 1.4. Do ponto de vista lógico é a estrutura de interconexão mais simples entre as UCPs e os módulos de memória. Quando duas ou mais UCPs estão conectadas no barramento, alguma estratégia de decisão deve ser usada para estabelecer o acesso ao barramento : um intervalo de tempo fixado é designado a cada UCP ou a requisição de acesso pelas UCPs deve ser tratada por um mecanismo arbitrador de barramento. É claro também, que o barramento não permite a transferência simultânea entre diferentes pares processador/memória e portanto o sistema deve ter um número limitado de processadores afim de evitar o "gargalo" (redução significativa da frequência de comunicação) do sistema completo.

Para o sistema de múltiplos processadores com memória compartilhada, utilizamos o Processador Preferencial (PP), hardware desenvolvido pelo Centro de Pesquisa e Desenvolvimento da TELEBRAS - CPqD.

1.3.1 - Estrutura do Processador Preferencial

A configuração do hardware [35] está baseada numa arquitetura modular constituída de várias placas processadoras de 16 bits, placas de memória e um conjunto de placas para controle de dispositivos periféricos, organizadas em torno de um barramento assíncrono dedicado de alta velocidade (10 Mbytes/s). A possibilidade de operação em multiprocessamento deve-se a presença de um arbitrador de barramento em sua unidade central de processamento.

A estrutura do sistema para processamento paralelo com médio acoplamento entre as unidades processadoras consiste de :

a) Barramento Global (PP-BAR) : utilizado para transferência de informações entre as várias unidades. É assíncrono de 16 bits, de alta velocidade (10 Mbytes/s), permitindo endereçamento de até 16 Mbytes de memória. Contem sinais para : indicação de interrupções, iniciação de todas as unidades que estão ligadas ao barramento, multiprocessamento e DMA, comunicação de falhas e árbitro "daisy-chain" ou paralelo. A especificação do PP-BAR é aberta, possibilitando a implementação de placas específicas, suportando até 16 placas ;

b) Unidade Central de Processamento (PP-UPN) - microcomputador baseado no microprocessador iAPX286 (6, 8, 10 MHz). A unidade contem : co-processador numérico 80287, barramento interno para a comunicação dos seguintes blocos :

- memória dinâmica interna (RAM) de 512 Kbytes ,
- memória permanente de 64 bytes (EPROM) ;

dois relógios programáveis, arbitrador de barramento, linhas para interrupções externas e interfaces de E/S ;

c) Unidade de Controle de Discos Rígidos e Flexíveis (PP-DIS) : unidade de Entrada/Saida inteligente para gerenciar os seguintes periféricos : 4 unidades de disco rígido de 5 1/4", 4 unidades de disco flexível de 5 1/4", e/ou 8", unidades de fitas e 1 canal RS232C.

d) Unidade Controladora de Interfaces Compatíveis (PP-CON) : esta unidade permite a ligação de placas periféricas comerciais (IBM-PC) ao barramento PP-BAR. Contem 1 interface para ligação de teclados, 128 Kbytes de RAM

local possibilitando expansão de memória da UCP de 512 Kbytes para 640 Kbytes.

e) Unidade de Rastreamento (PP-RAS) : esta unidade permite monitorar as ocorrências no barramento, suportando inclusive o multiprocessamento. Ela possibilita : rastreamento de todas as linhas do barramento, medição de tempo decorrido entre 2 condições de gatilho de até 6,6 segundos, em intervalos de 200ns, formatação e visualização dos dados rastreados.

Para a arquitetura apresentada :

1) Uma das unidades de processamento age como mestre e as outras unidades como escravos ;

2) Os servicos de alocação e inicialização de programas em cada UPN são realizados pela unidade mestre ;

3) Todas as UPNs têm acesso direto com a mesma prioridade à memória RAM (128 Kbytes) existente na placa PP-CON, utilizada como área comum de dados (memória global). Esta memória é utilizada pelos programas concorrentes apenas para comunicação de dados e para sincronização dos mesmos ;

4) Uma UPN permanece possuidora do barramento até que outra(s) unidade(s) faça(m) um pedido de acesso externo ;

5) Um sistema operacional monoprocessador compatível com o sistema MS-DOS é carregado em cada UPN, possibilitando a execução de programas compilados na linguagem Pascal.

6) A sincronização e a comunicação entre os programas independe da localização dos mesmos e é realizada através de variáveis compartilhadas que residem em endereços específicos na memória global. Com a utilização de comandos de atribuição presentes na linguagem Pascal, é possivel fazer uma cópia das variáveis compartilhadas da memória global para a memória interna (local) das UPNs e vice-versa.

1.4 - Sistemas de Múltiplos Processadores com Troca de Mensagens

Neste sistema, as UCPs não compartilham uma memória global e estão interconectadas através de linhas de entrada e de saída de dados. Em geral, este sistema é definido como sistema de múltiplos computadores. A única forma para os dados serem compartilhados entre as UCPs é pelo programador definir explicitamente comandos que transfiram dados de um computador para outro. A estrutura de interconexão pode ser constituída de linha serial ou paralela. A taxa de transmissão da dados está compreendida entre alguns Kbytes por segundo a 10 Mbytes por segundo. Na figura 1.5 mostramos um sistema inteiramente interconectado, com cada UCP tendo uma conexão direta com cada uma das outras UCPs. Tal esquema é impraticável quando existe um grande número de UCPs.

1.4.1 - Estrutura de Múltiplos Microcomputadores

O sistema de múltiplos microcomputadores (μCs) utilizado consiste de três microcomputadores Digirede 8000 interconectados através de cabos seriais. Cada

microcomputador possui :

(a) Microprocessador Motorola 68010 de 10 MHz ;

(b) Memória RAM de 2 Mbytes e memória EPROM de 16 Kbytes ;

(c) DMA ("Direct Memory Access") : responsável pela transferência direta entre a memória e os periféricos (unidade de disco rígido, unidade de discos flexiveis, etc) sem intervenção da unidade central de processamento (UCP) ;

(d) Processador Inteligente de Comunicação (PIC) : microprocessador Z-80, operando na frequência 4 MHz e dedicado ao controle de comunicação com terminais, linhas externas, micro, etc ;

(e) canais seriais (/DEV/ tty A), onde A é o número da porta de comunicação à qual se liga o cabo serial. A velocidade de comunicação permitida é de até 9600 bps.

Cada micro está conectado a um outro através de uma ligação dupla, formando uma rede que possibilita a transferência de arquivos e de dados entre os μCs.

A principal vantagem dessa estrutura é a autonomia dos microcomputadores.

Para esta arquitetura :

a) cada microcomputador é autônomo, no sentido de que não há necessidade de compartilhar recursos comuns (memória, barramento) com outros micros ;

b) cada micro contem seu próprio sistema operacional DIGIX (UNIX versão III). Esta versão do UNIX não apresenta alguns comandos de comunicação presentes em versões mais recentes, tais como o UNIX versão V, mas estamos desenvolvendo mecanismos de troca de mensagens e de sincronização entre os μCs, empregando recursos de software já existentes no UNIX ;

c) a combinação de comandos do UNIX (cat, mknod, cp, cu), de operações de redirecionamento (<, >, >>), de canalização (|) e de recursos de programação concorrente (&, &&, ;) e a utilização de arquivos especiais (/DEV/ tty A) tem nos permitido, com certa sobrecarga computacional resultante da comunicação e da sincronização, o processamento paralelo dos programas ;

d) o processamento paralelo com relação ao hardware apresenta dois pontos críticos : os processadores de comunicação e os canais de transmissão. O processador inteligente de comunicação atualmente utilizado no computador não permite uma alta velocidade de comunicação entre os μCs, dificultando uma interação mais intensa entre os μCs. Os canais de transmissão têm velocidade máxima de transmissão de 9600 bps com um caractere por vez sendo transmitido. Estes pontos críticos resultam do fato de que os computadores DIGIREDE 8000 não foram projetados para as solicitações de processamento paralelo. Estamos estudando soluções alternativas para remover tais pontos, e elas compreendem a substituição do processador de comunicação baseado no microprocessador Z-80, por "transputer"; e a utilização de canais paralelos para comunicação

micro a micro.

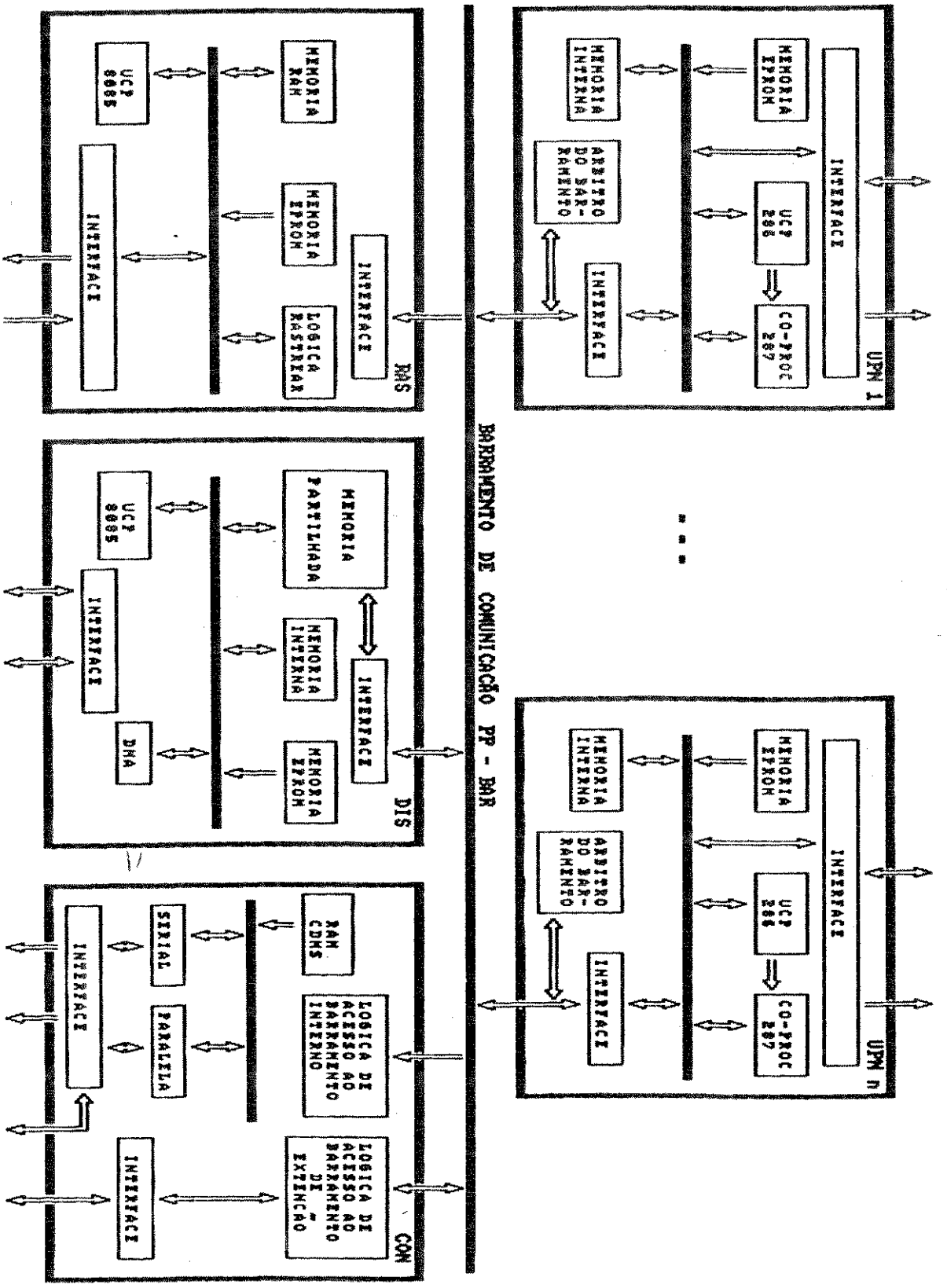


Figura 1.6 - Estrutura do Processador Preferencial para processamento paralelo

Figura 1.7 - Arquitetura de multiplos microcomputadores (DIONEDE 0000)

1.5 - COMENTÁRIO

Com a necessidade continua de sistemas de processamento paralelo de grande velocidade de computação e devido as limitações tecnológicas, os usuários ávidos de grande desempemho computacional tem buscado atualmente por inovações nas arquiteturas de computadores que não são adequadamente descritas pela classificação de Flynn, embora esta seja ainda bastante empregada. A noção de taxonomia requer que todos os sistemas sejam inequivocamente identificados. A classificação restrita a sistemas de múltiplos processadores que apresentamos representa uma tentativa de descrever as características das arquiteturas com relação ao grau de concorrência, a forma de implementação do algoritmo, características de processamento e a estrutura de controle (sincronização).

Com a classificação que adotamos, foi definida duas classes de arquiteturas de múltiplos processadores : sistema com memória compartilhada e sistema com troca de mensagens. Esta classificação permite uma variedade de formas de processamento paralelo sobre os sistemas especificados, dependendo das aplicações do usuário.

Com relação as arquiteturas apresentadas, o Processador Preferencial permite um processamento paralelo mais efetivo e com bom desempenho computacional, enquanto o sistema de múltiplos microcomputadores apresenta restrições referentes aos mecanismos de comunicação o que não permite um maior grau de interação entre os microcomputadores. O sistema de múltiplos microcomputadores foi utilizado mais para verificarmos as características de implementação dos algoritmos, do que para uma análise de desempenho computacional dos algoritmos paralelos.

CAPÍTULO 2 - AMBIENTES DE PROGRAMAÇÃO PARALELA

2.1 - Introdução

As implementações de programas em diferentes arquiteturas de múltiplos processadores requerem uma consideração cuidadosa do algoritmo e do ambiente de programação para se atingir um desempenho computacional ótimo sobre uma arquitetura específica. Em adição, a maioria dos programadores e dos usuários de computadores é relativamente experiente com algoritmos sequenciais, mas têm pouca ou nenhuma experiência com a computação paralela. Tradicionalmente, os esforços têm sido concentrados em encontrar algoritmos sequenciais computacionalmente mais eficientes. A implicação fundamental desta atitude associada a dificuldades de se utilizar hardware e software para processamento paralelo é bastante simples : sistemas para processamento paralelo, notadamente sistemas de múltiplos processadores, raramente são utilizados para aplicações práticas, a menos que o uso de um sistema para processamento paralelo seja requerido pelas restrições das aplicações. Se nós concordamos com isto,é inquestionavel que o ponto crítico para o qual se dirige este capítulo, é que a maioria dos usuários não sabe como programar e/ou utilizar as novas arquiteturas para processamento paralelo. Para resolver este problema, acreditamos que o desenvolvimento de ambientes de software é necessário e essencial. Em adição a esta estratégia, consideramos que um esforço para desenvolver novos algoritmos ou novas formas de computação dos algoritmos existentes é também necessário, iniciando com os princípios básicos do que constitui um algoritmo, e de como implementá-lo numa arquitetura específica para processamento paralelo. Um algoritmo pode ter um número de diferentes soluções em

processamento paralelo, onde cada uma pode ou não ser bem adaptável a implementação sobre uma arquitetura particular. As soluções alternativas para o algoritmo são frequentemente limitadas mais pela criatividade de quem resolve o algoritmo, do que pelas características do algoritmo.

Neste contexto, exploramos através de formas diferentes de multiprogramação, o paralelismo existente em algoritmos sequenciais, com o objetivo principal de podermos entender melhor a execução de algoritmos de de otimização multinível com relação aos seguintes fatores :

(a) Ordem e estrutura do problema de otimização ;
(b) Natureza do algoritmo de otimização : tipo, método de decomposição e coordenação ;
(c) Transferência de dados (interação) entre os subsistemas ;
(d) Distribuição da memória total de cálculo ;
(e) Tempo de processamento ;
(f) Sobrecarga computacional devido a sicronização dos programas e a comunicação de dados ;
(g) Arquiteturas de sistemas de múltiplos processadores.

Neste sentido, a multiprogramação é utilizada como uma ferramenta de simulação para o estudo rápido e eficiente do paralelismo existente nos algoritmos.

2.2 - Sistemas de Multiprogramação

Um programa sequencial determina uma execução sequencial de um conjunto de comandos. Um programa concorrente determina que dois ou mais programas sequenciais podem ser executados concorrentemente. Um programa concorrente pode ser executado, permitindo que programas compartilhem um mesmo processador. Quando programas compartilham um processador, é como se cada programa fosse executado sobre um processador com velocidade variável. Esta velocidade é determinada pelas interrupções, pela politica de gerenciamento do processador e pelo uso do tempo do processador por outros programas. A característica que todos os sistemas de múltiplos processadores compartilham que é a redundância de seus elementos de processamento, é precisamente a característica simulada pela multiprogramação que é gerada por um sistema operacional. Esta forma de multiplexação do processador permite que cada programa seja executado sobre um processador virtual que é um processador abstrato que implementa uma unidade de paralelismo.

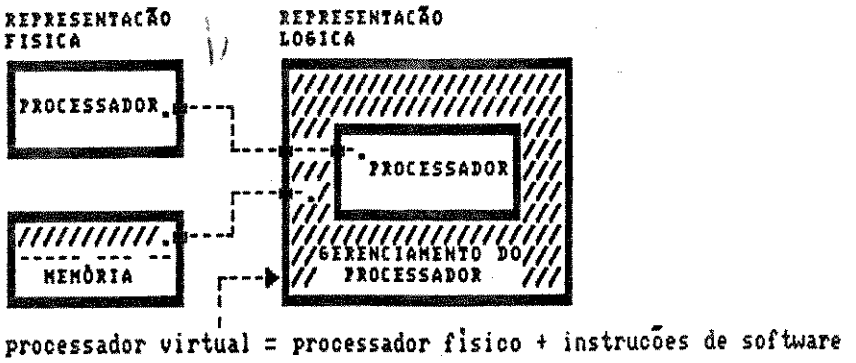


Figura 2.1 - Processador físico e processador virtual

A multiprogramação,portanto,é importante porque fornece um ambiente abstrato para o estudo do paralelismo sem

complicacões desnecessárias com detalhes de implementação que apesar de significativos em outras ocasiões, podem ser ignorados numa primeira fase.

Nós tratamos com duas formas de multiprogramação: uma do tipo tempo compartilhado e a outra do tipo gerenciamento de interrupções.

2.2.1 - Multiprogramação com tempo compartilhado

Em um sistema de computador de tempo compartilhado, os processos (programas em execução) competem, para execução, do partilhamento do tempo. O escalonamento é a série de ações que decide qual o processo a ser executado. Obviamente, o escalonamento é um dos elementos chaves de um sistema de tempo compartilhado. O ato fundamental de compartilhamento de tempo é a suspensão de um processo e o seu reinício mais adiante. Na maioria dos sistemas de tempo compartilhado (ex. : o sistema UNIX) a atividade de suspensão/reinício ocorre muitas vezes por segundo para que pareça externamente que o computador esta executando vários programas simultaneamente. A explicação muito simples de um sistema de compartilhamento de tempo é que cada unidade de tempo é dividida em um número de partes. Cada programa sendo executado recebe uma parte do tempo. Quando mais programas estão sendo executados , cada programa recebe uma parte menor do que quando somente uns poucos programas estão sendo executados. Em alguns sistemas (por exemplo, o sistema UNIX) também levam em conta o fato de que alguns processos na espera de uma operação de Entrada/Saída podem não estar aptos a se aproveitarem de sua fatia de tempo. Para evitar a descriminação contra os processos que executam E/S, o sistema operacional calcula dinamicamente as prioridades dos processos para determinar qual processo inativo, mas

pronto para execução será executado quando o processo atual ativo por suspenso.

Para esta forma de multiprogramação utilizamos o sistema operacional UNIX e a linguagem de programação Pascal.

2.2.2 - Multiprogramação com Gerenciamento de Interrupções

É uma técnica, onde o gerenciador ("scheduler") de um sistema operacional escolhe o próximo programa a se executado ou quando um programa termina sua execução ou quando um programa solicita uma operação de Entrada/Saída ou finalmente quando uma operação de Entrada/Saída é concluída. O gerenciador, então, procura o programa de maior prioridade que não está esperando o final de uma operação de E/S para inicializar sua execução.

Para este tipo de ambiente de multiprogramação, utilizamos um sistema operacional com suporte para um ambiente de programação concorrente [32] com as seguintes funções básicas :

a) Gerenciamento do uso do processador de maneira a permitir o seu compartilhamento pelos programas ;

b) Permitir que programas concorrentes executem sob um esquema de prioridades ;

c) Suporte para comunicação e sincronização entre programas de modo a permitir que estes cooperem entre si ;

O ambiente de multiprogramação utilizado é semelhante ao sistema CONIC [29], ele compreende uma metodologia

de projeto e duas linguagens : a Linguagem de Programação de Módulos - LPM, usada para programar módulos (programas) e a Linguagem de Configuração de Módulos - LCM, usada para descrever um sistema de instâncias interconectados de tipos de módulos. Estas linguagens permitem expressar o paralelismo e desenvolver programas concorrentes de grande confiabilidade e desempenho.

2.3 - Ambiente UNIX para programação concorrente

Quando vários programas são executados concorrentemente as comunicações entre eles estabelecem uma rede de interdependências que levam a um problema ainda mais sutil - a sincronização. O sistema UNIX fornece formas padrões e flexíveis de interconexão de entrada e de saída de programas em execução e permite a coordenação de vários programas para realizar uma determinada tarefa. Sob estes aspectos analizamos algumas características do sistema UNIX que facilitam a execução concorrente de programas paralelos. A descrição completa das características do sistema UNIX podem ser encontrados na literatura [25,26].

2.3.1 - Sistemas de Arquivo

Como qualquer sistema operacional, o UNIX gera facilidades de execução de programas e um sistema de arquivos para armazenamento de informações. Um aspecto fundamental do sistema de arquivos é o compartilhamento de arquivos que podem ser utilizados como uma área comum de dados por vários programas. Os arquivos fornecem meios de se estabelecer uma comunicação entre os programas concorrentes. Arquivos estão organizados em diretórios numa estrutura hierárquica. Os arquivos presentes em um diretório

podem ser arquivos ordinários,arquivos diretórios e arquivos especias. Para a programação concorrente numa unidade de processamento utilizamos os arquivos ordinários como áreas comuns de dados, enquanto para a computação distribuida, empregamos os arquivos especiais como canais de comunicação.

2.3.1a - Arquivos Ordinários

Um arquivo ordinário no UNIX é uma sequência de bytes. No UNIX, os arquivos não têm uma estrutura interna. Os bytes presentes num arquivo são apenas aqueles postos pelo usuário ou programa. Não há prealocação de espaço para os arquivos; um arquivo é tão grande quanto ele seja necessário; não há distinção entre acesso sequencial ou randômico; os bytes de um arquivo são acessíveis em qualquer ordem.

Embora os arquivos não tenham uma estrutura, o(s) programa(s) que interage(m) com outro(s) programa(s) podem impor uma estrutura para tratar um fluxo de dados.

2.3.1b - Arquivos Especiais

Uma das características do sistema UNIX é a associação de todo hardware de E/S com arquivos especiais. O acesso ao próprio hardware de E/S imita o acesso aos arquivos ordinários. Arquivos especiais são utilizados como um canal conveniente para o acesso aos mecanismos de E/S. Para cada mecanismo de E/S (leitora de cartões, terminais, discos, etc.) que é conectado ao computador há pelo menos um arquivo especial. Utilizamos os arquivos especiais para terminais (tty0,tty1,...,tty9) como canais de comunicação com outros computadores. Um arquivo especial para cada terminal manipula um caractere por vez, não possuindo um comprimento porque não é uma região de armazenamento, mas uma ligação com um dispositivo de E/S. Esta característica importante difere um arquivo especial de um arquivo

ordinário, ou seja, quando um programa tenta ler ou escrever num arquivo ordinário (arquivo em disco), ele certamente completará a operação dentro de algumas centenas de milisegundos na maioria das vezes, enquanto que um programa para ler de um arquivo especial terá de esperar até que algum dado seja posto no arquivo.

Um programa não tem conhecimento sobre os detalhes do mecanismo de E/S para ler ou escrever no arquivo especial.

2.3.2 - SHELL

Uma das características mais interessante do sistema UNIX é o shell, que é um poderoso interpretador de comandos interativos entrelaçados dentro de uma linguagem de programação de alto nível. Da linguagem de programação shell, resaltamos os aspectos que são mais necessários para realizarmos a multiprogramação e os mecanismos de software para a computação distribuida. Uma descrição mais detalhada do shell pode ser encontrada nas referências [27,28].

2.3.2a - Processos de Retaguarda e Vanguarda.

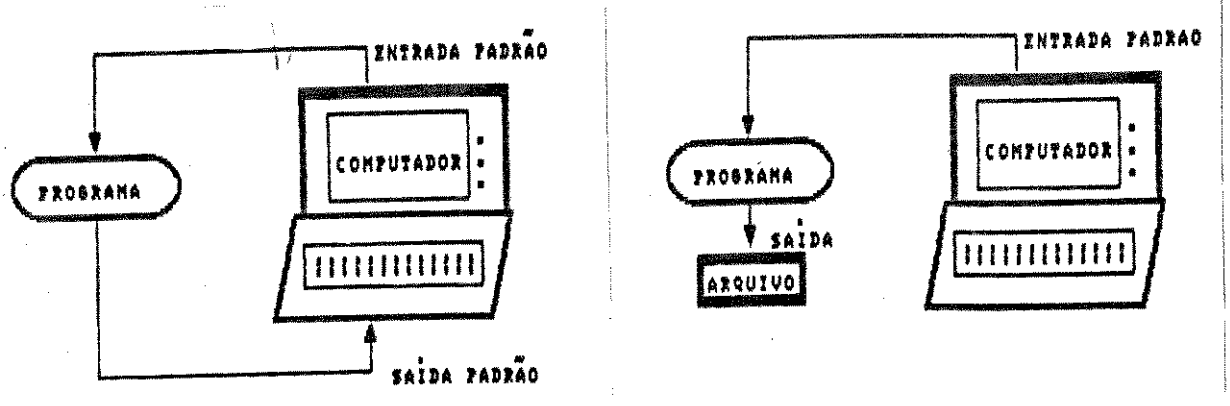
O shell tem uma característica especial que o habilita a iniciar um programa e deixá-lo executando, enquanto outros programas podem iniciar suas execuções. Quando um programa está sendo executado sem o operador, dizemos que ele está sendo executado em retaguarda, enquanto os programas que são inseridos estão rodando em vanguarda. Os programas de vanguarda e de retarguarda são executados simultaneamente. Os caracteres, a seguir, podem ser utilizados para especificar uma ordem de execução :

a) ";" indica uma execução sequencial. Por exemplo : PROGRAMA 1 ; PROGRAMA 2. O PROGRAMA 2 será executado depois do PROGRAMA 1 ser executado.

b) "&" indica a execução de retaguarda (assíncrona). Por exemplo : PROGRAMA 1 & PROGRAMA 2. O PROGRAMA 1 será executado em retaguarda e o PROGRAMA 2 em vanguarda.

2.3.2b - Redirecionamento de Entrada e de Saída

A entrada e a saída padrões são normalmente conectadas ao terminal do computador. Entretanto, já que elas são estabelecidas pelo Shell, é possível que sejam redesignadas pelo shell. O ">" é um caractere especial do shell que específica que a saída padrão do programa deve ser redirecionada ao arquivo (ordinário ou especial) indicado.



(a) (b)

Figura 2.2 - Redirecionamento de saída

A saída do PROGRAMA 1 (figura 2.2b) é um arquivo e não uma saída padrão. O ">>" é um caractere que permite redirecionar a saída que será acrescentada ao final de um arquivo especificado.

A entrada padrão também pode ser redirecionada pelo shell. Neste caso, é possível fazer com que um programa adquira dados de arquivos.

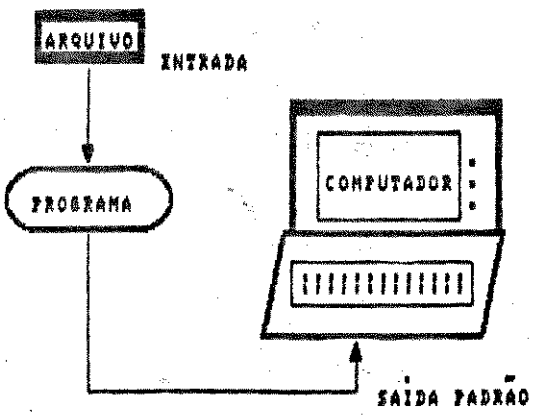


Figura 2.3 - Redirecionamento de entrada

PROGRAMA 1 < arquivo B , o caractere "<" faz com que o PROGRAMA 1 adquira dados entrada do arquivo B.

2.3.2c - Tubos ("pipes")

Um tubo conecta a saída padrão de um programa com a entrada padrão de um outro programa. Como resultado, os programas não necessitam saber quando eles estão escrevendo ou lendo de outro programa, permitindo, portanto, que uma cadeia de programas seja executada.

Por exemplo.: PROGRAMA 1 | PROGRAMA 2 > arquivo A. Se

o tubo está vazio, ou seja, se PROGRAMA 1 ainda não escreveu nenhum dado na sua saída padrão, o PROGRAMA 2 terá de esperar que o PRAGRAMA 1 escreva algum dado para realizar uma leitura. A saída do PROGRAMA 2 é redirecionada para o arquivo A.

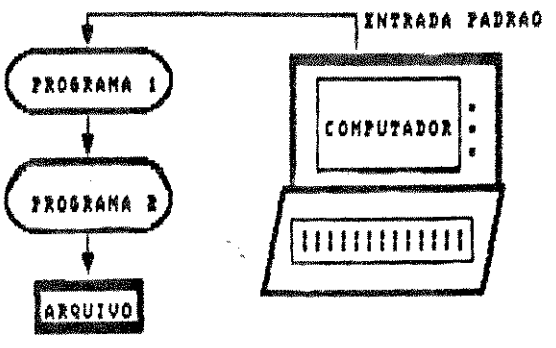


Figura 2.4 - Canalização

2.3.2.d - Programas utilitários

Uma das grandes vantagens do sistema UNIX é o seu conjunto de programas utilitários. O sistema UNIX possui vários programas para manipulação de arquivos. Alguns desses programas podem ser utilizados para desenvolver estratégias de comunicação. Por exemplo :

```
cat /dev/tty9 | PROGRAMA 1
```

O programa cat (utilitário que gera arquivos no terminal e que combina vários arquivos em um, usando redirecionamento de saída) ler dados do arquivo especial

/DEV/tty9, que pode ser um canal de comunicação com outro computador. Sua saída é canalizada como entrada de dados para o PROGRAMA 1. Os utilitários mais úteis para gerar facilidades de comunicação que estamos atualmente empregando são : cat, cp, cu, mail, mknod, write, kermit. A versão III do UNIX não tem alguns comandos de comunicação (mailx, rmail, xmodem, rn, shar, uucp e etc) que podem tornar o nível de cooperação entre os programas mais confiável e substancial e que já estão disponíveis nas versões mais recentes.

2.4 - Características Gerais das Linguagens de Programação e de Configuração de Módulos.

2.4.1 - A Linguagem de Programação de Módulos

LPM é baseado na linguagem Pascal com extensões para a modularidade e a troca de mensagens [31]. Na linguagem, o conceito de programa em Pascal é substituido pelo conceito de módulos. Os módulos são constituidos por tarefas (processos no sentido computacional) sequenciais tendo interface especificada para troca de mensagens através de portas de entrada (entryports) e de saída (exitports). As interconexões e troca de informações entre os módulos são especificados em termos de portas. Uma porta de saída designa a interface na qual transações de mensagens podem ser inicializadas e especifica um nome local e o tipo de mensagem em lugar do nome do destino da mensagem.

Na fig.2.5, mensagem ps é enviada a porta de saída ps. Na configuração ou execução a porta de saída pode ser interligada a uma porta de entrada compatível de qualquer módulo que deseja receber a mensagem ps. As portas de entra-

da pe1 e pe2 na fig.2.5, designam a interface na qual as transações de mensagens podem ser recebidas. Na configuração ou na execução, qualquer módulo com uma porta de saída compatível pode ser interligada a estas portas de entrada. A linguagem de programação utiliza nomes locais dentro do módulo, ao invés de nomear diretamente a fonte e o destino das mensagens. Isto assegura um alto nível de independência em relação a programação e permite reutilizar os módulos em várias situações.

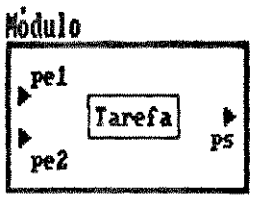


Figura 2.5 - Estrutura de um módulo

2.4.2 - Modelo de comunicação

Uma programação distribuida é um conjunto de módulos executando, onde cada módulo é executado assincronamente com outros módulos. A computação dos módulos ocorre em ocorre em diferentes áreas de endereçamento de memória e eles interagem uns com os outros somente através da comunicação de mensagens.

As operações "send" e "receive" sobre as portas são as primitivas de comunicação criadas para enviar uma mensagem a uma porta de saída ou receber uma mensagem de uma porta de entrada. Os tipos de mensagem devem corresponder aos tipos de portas. Diferenças no desempenho

entre uma comunicação local (dentro de uma estação) e uma remota (entre estações) são inevitáveis, uma vez que nas conexões locais a troca de mensagem entre duas portas consiste basicamente, na cópia da mensagem da área onde foi gerada para o "buffer" da porta de entrada destino, enquanto nas conexões remotas a troca de mensagens pressupõe uma rede de comunicação sujeita a atrasos e erros. Esta transparência da comunicação permite que módulos sejam alocados numa mesma estação ou em estações diferentes. As primitivas definem dois tipos de transações de mensagens : "request-reply" e "notify". A transação "request-reply" gera troca de mensagens síncronas bidirecional, enquanto a "notify" é unidirecional e assíncrona.

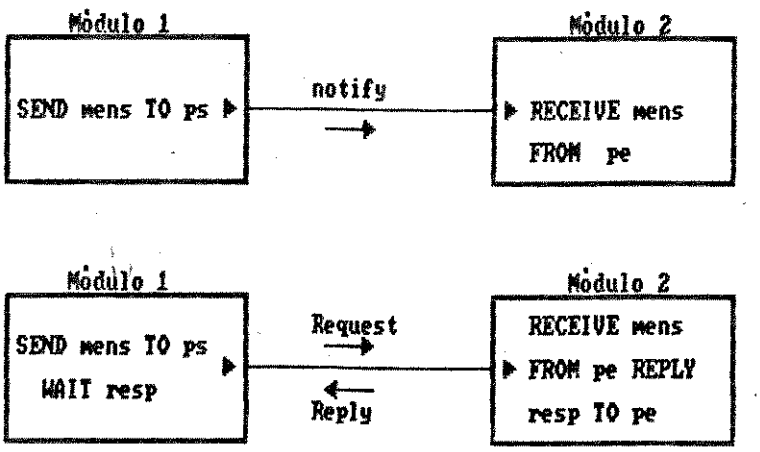


Figura 2.6 - Tipos de transação.

2.4.3 - Linguagem de Configuração de Módulos

LCM é uma linguagem extremamente simples que

permite um sistema ser descrito como um conjunto de instâncias interconectadas de tipos de módulos. Instâncias de módulos são conectadas pela ligação das portas de saída com as portas de entrada. Estas conexões são validadas pela garantia de que as portas de entrada e de saída ligadas são do mesmo tipo. A LCM fornece construções que permitem a declaração e a remoção de instâncias, conexão e desconexão de portas e ativação das instâncias dos módulos. A linguagem LCM possibilita os seguintes tipos de conexão de portas :

(i) uma porta de saída a uma porta de entrada ;

Figura 2.7

(ii) uma porta de saída a várias portas de entrada ;

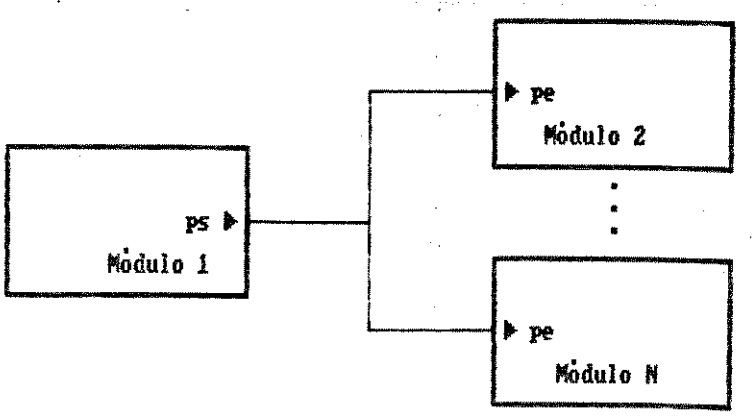


Figura 2.8

(iii) várias portas de saída a uma porta de entrada ;

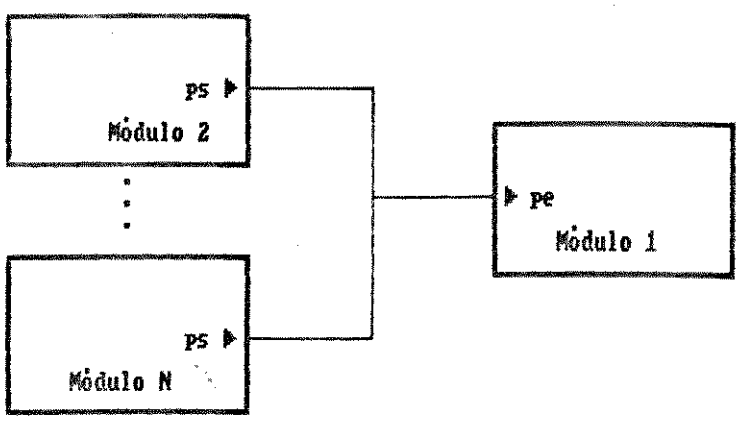


Figura 2.9

O processo da configuração, é atualmente, estático o que aumenta a consistência do sistema. Sobre um suporte de tempo real para programação concorrente, são executados os módulos escritos em LPM e fisicamente realizadas as configuções descritas no LCM.

Para os usuários não familiares com o CONIC, LCM, e LPM sugerimos as referências [29,30,31].

2.5 - Exemplo Ilustrativo

Para mostrar os aspectos da programação paralela

de algoritmos nos ambientes de programação descritos,abordamos um problema clássico de controle para gerenciamento de sistemas de potência.

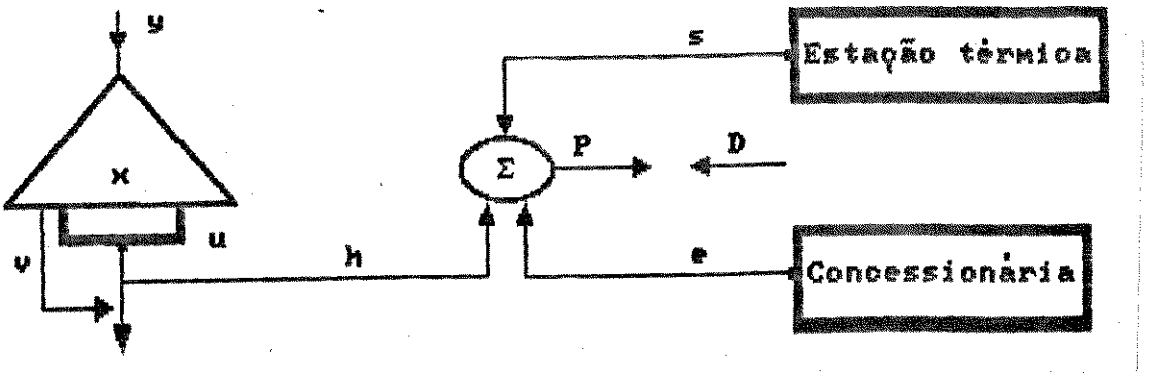


Figura 2.10 - Representação esquemática de um parque gerador

Os símbolos usados no modelo do sistema são :

s - geração térmica ;
e - intercâmbio recebido ;
h - geração hidráulica ;
u - turbinagem ;
x - estoque do reservatório ;
y - entrada de água independente ;
v - vertimento ;
P - oferta ;
D - demanda ;

Uma política de operação de custo mínimo,atendendo a demanda e outras restrições operacionais do sistema, pode ser obtida pela resolução do seguinte problema (P) :

$$\min J = \sum_{t=1}^{T} \left(G(s[t]) + p(t) \right) - w\, v(x[T+1]) \; ;$$

s.a : 1) $0 \leq s[t] \leq \bar{s}$;

2) $0 \leq e[t] \leq \bar{e}$;

3) $h[t] = f(x[t], u[t])$;

4) $h[t] + s[t] + e[t] = D[t]$;

5) $\underline{u} \leq u[t] \leq \bar{u}$;

6) $x[t+1] = x[t] + y[t] - v[t]$;

7) $p[t] = m + n\, e[t]$;

8) $x[1]$ é dado .

$t=1,\ldots,T$

A solução do problema (P) pode ser obtida por técnica de decomposição por dualidade [2]. Avaliando a função dual $H(\lambda[t])$:

$$H(\lambda[t]) = \min L(s[t], e[t], x[t], u[t], \lambda[t])$$

s.a: (1),(2),(4),(8)

onde :

$$L(s[t], e[t], x[t], u[t], \lambda[t]) = \sum_{t=1}^{T} \Big(G(s[t]) - p[t]e[t] + \lambda[t](D[t] - h[t] - s[t] - e[t]) \Big)$$

Como a função Lagrangeana é separável em relação as variáveis $(s[t], e[t], x[t], u[t])$ para um dado multiplicador $\lambda[t]$, a minimização pode ser subdividida em três subproblemas :

1 - Subproblema TÉRMICO : $\min \sum_{t=1}^{T} \left(G(s[t]) - \lambda[t]s[t] \right)$

s.a : $0 \leq s[t] \leq \bar{s}$

2 - Subproblema INTERCAMBIO : $\min \sum_{t=1}^{T} (p[t]e[t] - \lambda[t]e[t])$

$$s.a : 0 \leq e[t] \leq \bar{e}$$

3 - Subproblema HIDROELÉTRICO : $\max \sum_{t=1}^{T} \lambda[t] f(x[t],u[t]) + w\, v(x[T+1])$

$$s.a : h[t] = f(x[t],u[t])$$

$$\underline{u} \leq u[t] \leq \bar{u}$$

Na fig.2.11 apresentamos o diagrama de blocos para o procedimento computacional.

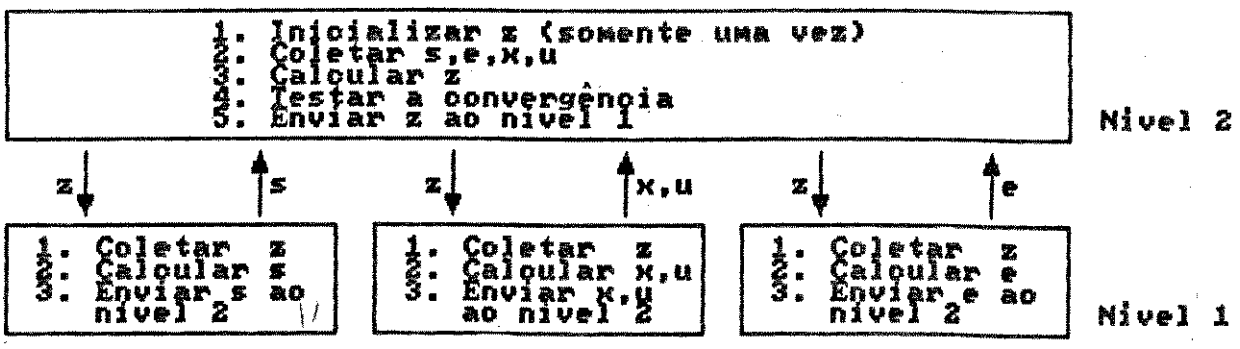


Figura 2.11

Os dados do problema são :

$G(s) = 0{,}02s^2$; $\bar{s} = 1000$ MW ;

$h = f(x,u) = (0{,}85 + 0{,}25.10^{-4}x)u - 0{,}66.10^{-4}v^2$ MW ;

$v(x[T+1]) = 20x[T+1] - 0{,}66.10^{-9}(x[T+1])^2$; $w = 1$;

$\underline{u} = 200$ m^3/s ; $\bar{u} = 1500$ m^3/s ; $x_1 = 6000$ $(10^6 m^3)$;

$p = 10 + 0{,}04e$; $\bar{e} = 500$ MW.

t	1	2	3	4	5	6	7	8	9	10	11	12
y (m^3/s)	786	882	708	774	949	945	819	1050	1066	853	926	926

D 1850 1880 2010 1980 1920 1860 1790 1750 1780 1820 1980 1990
(MW)

2.5.1 - Implementação em Multiprogramação

Os subproblemas do nível inferior e o coordenador foram resolvidos sequencialmente numa primeira fase e concorrentemente numa segunda fase, utilizando as duas formas de multiprogramação apresentadas. Na segunda fase, associamos a cada programa relativo a cada subproblema um processador virtual conforme a fig.2.12. Para a solução através da multiprogramação com tempo compartilhado, utilizamos o computador CADMUS 9200, baseado na UCP 68010, sobre o sistema operacional CUNIX versao V) e a linguagem de programação Pascal. A arquitetura de computação simulada é a de sistemas de múltiplos processadores com memória compartilhada, onde os arquivos ordinários são utilizados como áreas comuns de dados, conforme a fig.2.12.

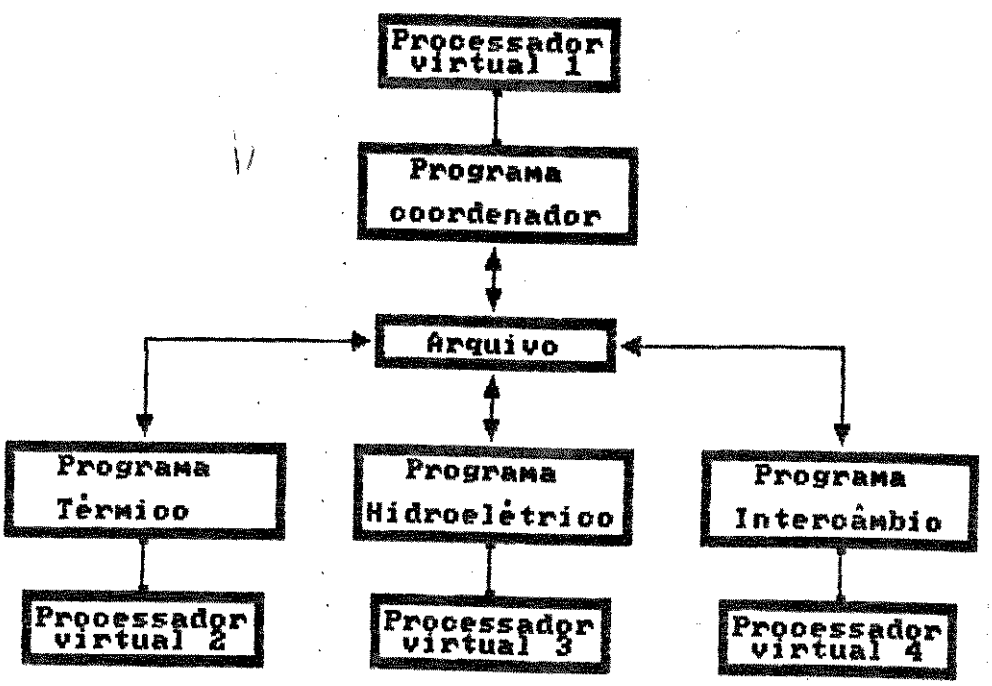


Figura 2.12 - Esquema de simulação

Para a segunda forma de multiprogramação, utilizamos um computador do tipo IBM-PC. A arquitetura de computação simulada é a de sistemas de múltiplos computadores interconectados , conforme a fig.2.13. Os módulos referentes aos subproblemas do nível inferior têm a mesma prioridade e o módulo referente ao coordenador tem prioridadade menor.

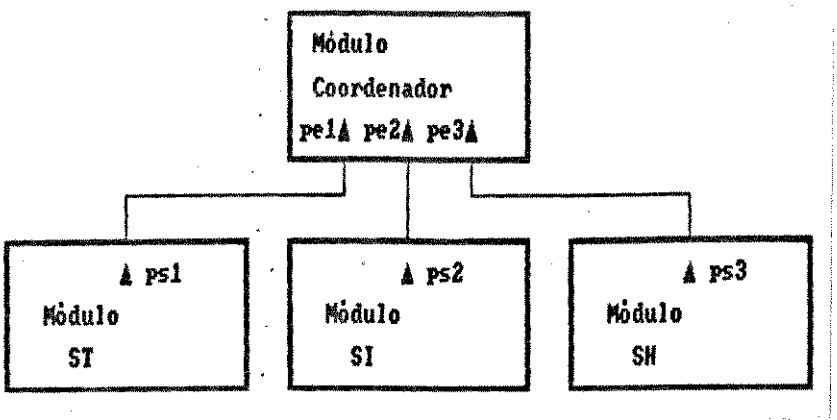


Figura 2.13 - Configuração dos módulos

As fases de implementação são as seguintes :

1ª Fase :
Implementação sequencial
(linguagem Pascal)

```
PROGRAM controle;...

PROCDURE ST;...
BEGIN ...
 calcular s;...
END;

PROCDDURE SI;...
BEGIN ...
 calcular s;...
END;
```

2ª Fase :
Implementação via multiprogramação (tempo compartilhado)

```
PROGRAMA ST;...
  VAR ...
     arqt,arqc: FILE OF REAL;
  BEGIN ...
    ler arqc;...
    calcular S;...
   escrever em arqt;...
   END.

PROGRAM SI;...
VAR ...
       arqi,arqc:FILE OF REAL;
```

```
PROCEDURE SH;...
BEGIN ...
 calcular x,u;...
END;

PROCEDURE COORDENADOR;...
BEGIN ...
 calcular λ;...
 testar se convergiu;...
END;

(* PROGRAMA PRINCIPAL *)
BEGIN ...
 inicializar λ;...
 WHILE (...) DO
 BEGIN
   ST;
   SI;
   SH;
   COORDENADOR;
 END;
END.
```

```
BEGIN ...
 ler arqc;...
calcular e;...
 escrever em arqi;...
END.

PROGRAM SH;...
 VAR ...
 arqxu,arqc:FILE OF REAL;
BEGIN ...
 ler arqxu, arqc;...
calcular X,U;...
escrever em arqxu;...
 END.

PROGRAM coordenador;...
 VAR ...
   arqt,arqi,arqxu,arqc:FILE
    OF REAL;
 BEGIN ...
  ler arqt,arqi,arqxu,arqc;...
 testar se convergiu;
  caluclar λ;...
  escrever em arqc;...
 END.
```

Programa para execução na linguagem de programação shell dos programas da 2ª fase :

```
CONTROLE
.
.
gerar arqxu, arqc (inicialização)
WHILE (...)
DO
   (ST & SI & SH) ; COORDENADOR
DONE
.
.
```

2ª Implementação em Multiprogramação com Suporte para Tempo Real

```
MODULE ST;
USE  bloco.inc
EXITPORT
 portex1:vetor REPLY vetor;
```

```
MODULE SI;
USE  bloco.inc
EXITPORT
 portex1:vetor REPLY vetor;
```

```
MESSAGE
  λ,S:vetor;
BEGIN_MODULE ...
LOOP
calcular S;
 SEND S TO portex1 WAIT λ;
 END_LOOP;...
END_MODULE.
```

```
MESSAGE
  λ,S:vetor;
 BEGIN_MODULE ...
LOOP
calcular θ;
SEND θ TO portex1 WAIT λ;
END_LOOP;...
  END_MODULE.
```

```
MODULE SH;
USE   bloco.inc;
EXITPORT
 portex1:vetor REPLY vetor;
MESSAGE
  λ,S:vetor;
BEGIN_MODULE ...
 LOOP
 calcular X,U;
 SEND X,U TO portex1 WAIT λ;
 END_LOOP;...
END_MODULE.
```

```
MODULE COORDENADOR;
USE   bloco.inc;
ENTRYPORT
portεn1:vetor REPLY vetor
portεn2:vetor REPLY vetor
portεn3:vetor REPLY vetor
MESSAGE
  X,U,θ,S,λ:vetor;
BEGIN_MODULE ...
 LOOP
 RECEIVE S FROM portne1;
 RECEIVE θ FROM portεn2;
  RECEIVE X,U FROM portεn3;
  END_LOOP;...
 END_MODULE.
```

Programa para configuração de módulos :

```
CONFIGURATION CONTROLE;
     INSTANCE
          S1:ST;
          S2:SI;
          S3:SH;
          C1:COORDENADOR;
CREATE
    S1,S2,S3,C1/P=LOWER;
LINK
    S1.portex1 TO C1.portεn1;
    S2.portex2 TO C1.portεn2;
    S3.portex3 TO C1.portεn3;
END_CONFI.
```

2.5.1 - Resultados

Os implementações do algoritmo de otimização em dois

níveis que fizemos para o sistema de potência convergiu em 7 iterações e alguns resultados são apresentados na Fig. 1 e nas tabelas seguintes.

A fim de comparar o tempo de computação e a requisição de memória, as tabelas 1,2,3 são apresentadas. Os resultados mostados nas tabelas foram obtidos na simulação sobre o sistema operacional UNIX no computador CADMUS 9200.

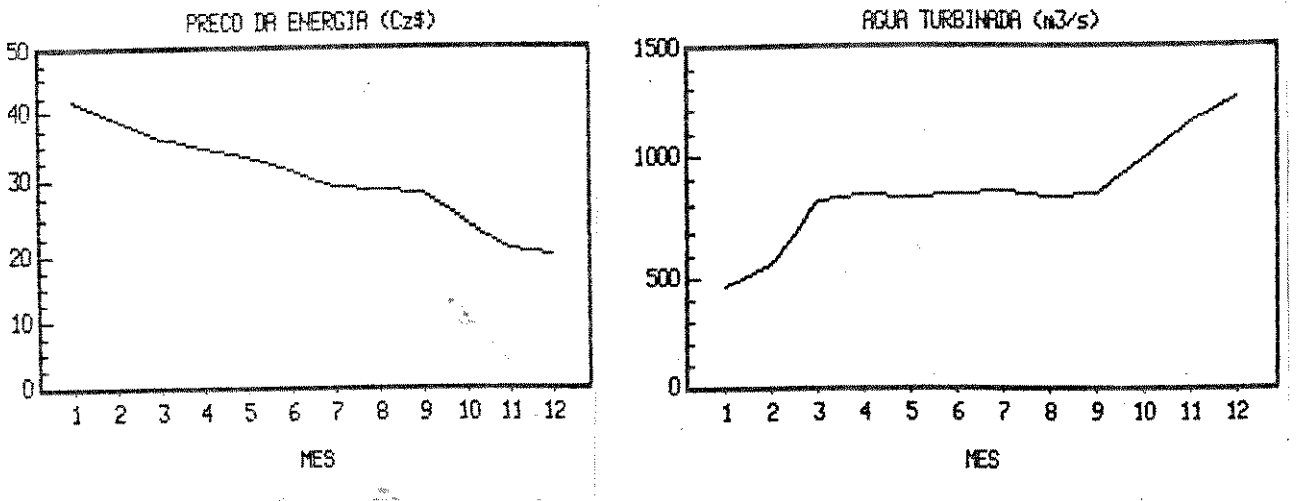


Figura 2.14

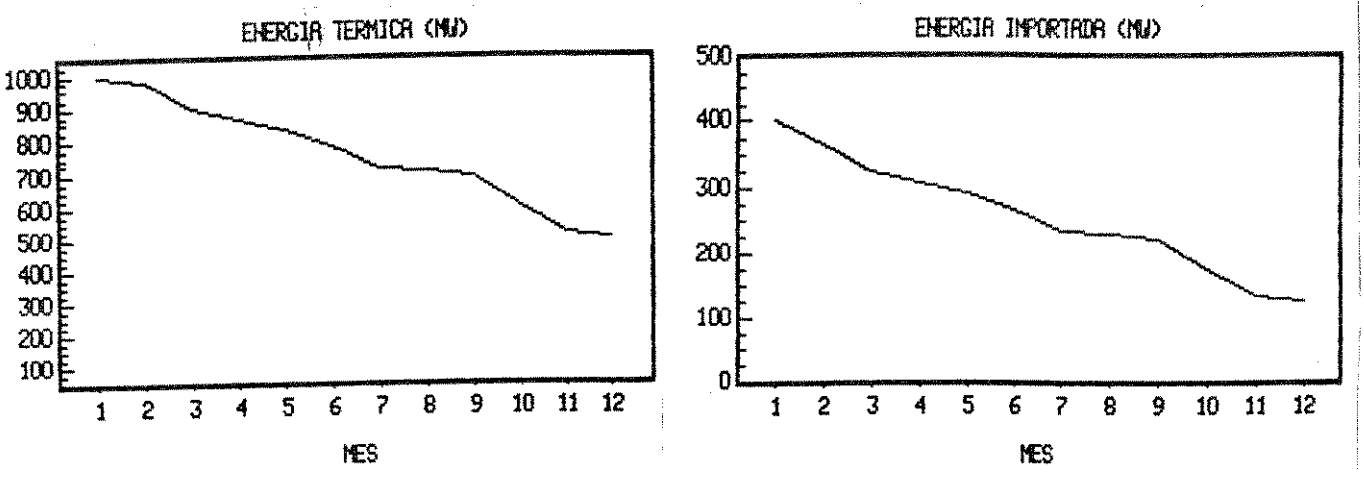


Figura 2.15

Tabela 1

Programação paralela	No. de linhas	programa compilado
Programa Térmico (PT)	22	13562 bytes
Programa Interc. (PI)	19	14264 bytes
Programa Hidroe. (PH)	85	20076 bytes
Pragrama Coorde. (PC)	76	30142 bytes

Tabela 2

tempo médio por iteração			
	T1	T2	T3
PT	0,36	0,05	0,10
PI	0,33	0,01	0,10
PH	0,60	0,13	0,16
PC	1,98	0,36	0,30

Tabela 3

Programa Sequencial		
No. de linhas 147		
Programa compilado 35870 bytes		
Tempo de solução		
T1	T2	T3
12,91	3,13	1,51

T1 - tempo gasto desde o início da execução do programa

T2 - tempo gasto executando o programa

T3 - quantidade de tempo gasto com a execução de chamadas ao sistema.

Nota : os tempos (T1,T2,T3) são medidos em segundos.

2.6 - Comentários

Alguns aspectos importantes da implementação via multiprogramação :

(a) Se a implementação sequencial exige mais memória os cálculos dos níveis em um único computador, o armazenamentodos programas compilados em ambiente de multiprogramação exige maior quantidade de memória ;

(b) O tempo gasto na resolução sequencial dos subproblemas no nível inferior foi maior do que o tempo gasto no subproblema paralelo mais lento em cada iteração ;

(c) A partir da multiprogramação, podemos verificar que a relação tempo de processamento - requisição de memória não são proibitivos quando utilizamos sistemas de múltiplos processadores para processar algoritmos de otimização multinível ;

(d) Depois dos experimentos que realizamos no nível da multiprogramação, ganhamos intuição e experiência para implementar as soluções de multiprocessamento bem como elementos para melhor entendimento e condições para a realização dos sistemas de processamento paralelo efetivo que são agora

desenvolvidos para aplicações em sistemas de grande porte .

CAPITULO 3 - PROCESSAMENTO PARALELO DE ALGORITMOS DE CONTROLE HIERÁRQUICO

3.1 - Estratégia de Programação Paralela

A elaboração de algoritmos paralelos torna-se tão importante quanto o desenvolvimento de arquiteturas para processamento paralelo. A (re)introdução de paralelismo no projeto de sistema de computação requer o desenvolvimento de algoritmos que devem ser processados eficientemente sobre tais arquiteturas. Não há no entanto nenhum método universal para o desenvolvimento de um algoritmo paralelo (conjunto de tarefas que podem ser executadas em paralelo e que se comunicam entre si durante a execução do algoritmo). Uma técnica para se obter algoritmos paralelos explora o grau de paralelismo existente em algoritmos sequenciais. Na maioria dos algoritmos sequenciais há algum grau de paralelismo, e cabe ao analista explorá-lo e avaliar o grau de concorrência resultante. Quando utilizamos sistemas de múltiplos processadores, tais como os descritos anteriormente, há um limite para o grau de paralelismo que pode ser obtido com um mínimo de sobrecarga com a comunicação e a sincronização. Particularmente, o sistema de múltiplos processadores com memória compartilhada utilizando o Processador Preferencial apresenta um médio acoplamento entre as unidades de processamento, o sistema operacional é monoprocessador e a linguagem empregada é inerentemente sequencial. Tais fatôres não permitem implementar com eficiência o paralelismo de pequenas operações de rápida execução devido a existência de muitos pontos de sincronização e uma grande sobrecarga computacional daí resultante. Uma vez que a implementação e o desempenho computacional de algoritmos paralelos dependem

significativamente do sistema de computação, há portanto, necessidade de especificá-lo quando consideramos o grau de paralelismo que pode ser obtido e eficientemente implementado.

Para a estratégia de paralelização de algoritmos sequenciais é conveniente considerarmos os seguintes problemas :

a) problema de partição;
b) pròblema de distribuição;
c) problema de comunicação e de sincronização.

3.1.1 - Problema de Partição

Dividir um algoritmo em tarefas que podem ser executadas por unidades de processamento independentes.

O critério para a escolha da partição depende :

a) da dimensão do problema (aplicação) : particionar um problema de grande dimensão em subproblemas de dimensões menores que podem ser resolvidos em paralelo, levando em consideração a dependência de dados entre os subproblemas, não é facil. As tarefas relativas aos subproblemas refletem quando ou não o algoritmo paralelizado apresenta uma comunicação mais intensa. Neste sentido, quanto menor a dimensão dos subproblemas maior será a quantidade de interação entre as tarefas.

b) do número de unidades de processamento utilizadas para a execução de um algoritmo paralelo : para tentarmos particionar um algoritmo num certo número de tarefas que podem ser executadas em paralelo é importante que saibamos com antecedência o número de unidades de processamento disponíveis.

c) das relações de dependência entre as equações presentes no algoritmo : quando se realiza a paralelização de um algoritmo sequencial deve se ter o cuidado de garantir que o algoritmo paralelizado seja equivalente ao algoritmo sequencial no que se refere ao conjunto de equações e aos resultados da solução de uma aplicação. Neste sentido uma análise das relações de dependência nos oferece condições de especificarmos um conjunto de equações que podem ser executadas simultaneamente. O ganho resultante na precisão da análise das relações de dependência é importante para a qualidade da paralelização do algoritmo.

3.1.2 - Problema de Distribuição

Determinar a cada tarefa uma ou mais unidades de processamento e vice-versa para a execução do processamento global.

Consideramos apenas um modo de distribuição das tarefas : o modo estático. Neste modo de distribuição, cada unidade de processamento executará exatamente a mesma tarefa durante o processamento global. Devido a distribuição no modo estático, o algoritmo paralelo não convergirá a menos que todas as unidades de processamento estejam operando. Particularmente, nos sistemas de múltiplos processadores que especificamos, todas as unidades de processamento são idênticas. Neste sentido, é importante que a distribuição das tarefas determine um bom balanceamento de carga, isto é, que um dos processadores em paralelo não fique esperando muito tempo que outros processadores terminem suas atividades para iniciar uma nova execução.

3.1.3 - Problema de Comunicação e Sincronização

Assegurar a integridade de dados compartilhados pelas tarefas e especificar uma ordem de execução que garanta resultados corretos.

A frequência com que as tarefas se comunicam e a quantidade de informações que deve ser comunicada influenciam a velocidade com a qual o algoritmo paralelo pode ser executado. Particularmente, nas implementações que realizamos, dividimos cada tarefa em duas partes lógicas :

a) passo de cálculo : é a parte da tarefa que pode ser executada sem interação com outras tarefas;

b) ponto de comunicação : é a parte da tarefa que ocorre ao final de cada passo de cálculo, permitindo a troca de dados com outra(s) tarefa(s).

O ponto de comunicação compreende :

a) pedido e permissão para leitura ou escrita de dados na memória global (sistemas de múltiplos processadores com memória compartilhada). Empregamos variáveis compartilhadas como informações de controle para sincronizar o acesso aos dados compartilhados pelas tarefas;
b) leitura e escrita dos dados compartilhados na memória global (sistema com memória compartilhada) ou troca de mensagens em sistemas de múltiplos computadores.

Definimos dois tipos básicos de relação de precedência para especificar uma sequência correta de execução dos algoritmos paralelos que obtivemos :

1) relação "Fork" : uma relação F : TAREFA j → TAREFA i,(i = 1,...,N) determina que para qualquer pedido de execução destas tarefas, a TAREFA j deve ser completada antes de cada TAREFA i ter sua execução iniciada ou seja, terminando a execução da TAREFA j, cada TAREFA i será executada.

2) relação"Join" : uma relação J : TAREFA i, (i = 1,...,N) → TAREFA j determina que para qualquer pedido para execução destas tarefas, cada TAREFA i deve ser completada antes da TAREFA j ter sua execução iniciada, ou seja, terminando a execução de cada TAREFA i, a TAREFA j será executada.

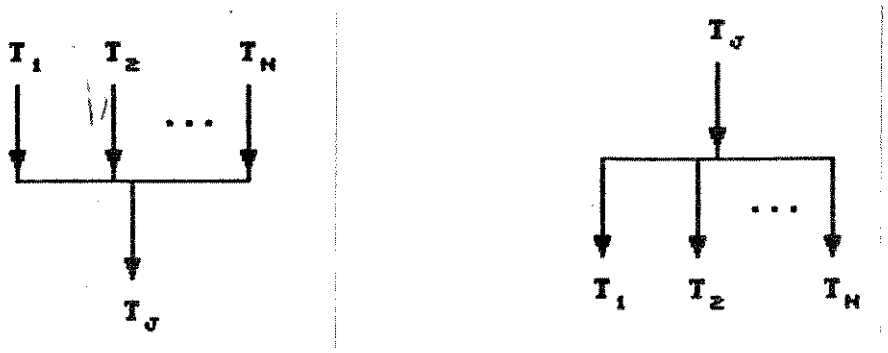


Figura 3.1 - a) relação Join b) relação Fork

Utilizamos as seguintes notações para representar a comunicação assíncrona e a comunicação com múltipla sincronização (várias tarefas são necessárias para que uma tarefa continue sua atividade) que podem ocorrer no corpo de

uma tarefa.

a) comunicação assíncrona b) sincronização múltipla

Figura 3.2 - Notações

O estudo cuidadoso desses três problemas nos permitirá obter e implementar o paralelismo que pode ser explorado num algoritmo sequencial.

3.2 - Metodologias de otimização e paralelização

A otimização de sistemas dinâmicos com vetores de estado de grandes dimensões é possível, em princípio, usando métodos bem estabelecidos tais como a Programação Dinâmica e o Princípio do Máximo de Pontryagin. A aplicação desses métodos na solução de um problema global de controle, envolvendo um grande número de variáveis, pode tornar a carga e o tempo de computação necessários para a otimização excessivos, a despeito da alta eficiência dos computadores atuais. A quantidade de computação para o controle ótimo de sistemas dinâmicos cresce muito rapidamente com a ordem do sistema dinâmico. Isto nos leva a considerar a decomposição do problema de otimização.

A maioria dos sistemas dinâmicos de grande porte consiste de subsistemas interconectados conforme a figura 3.3.

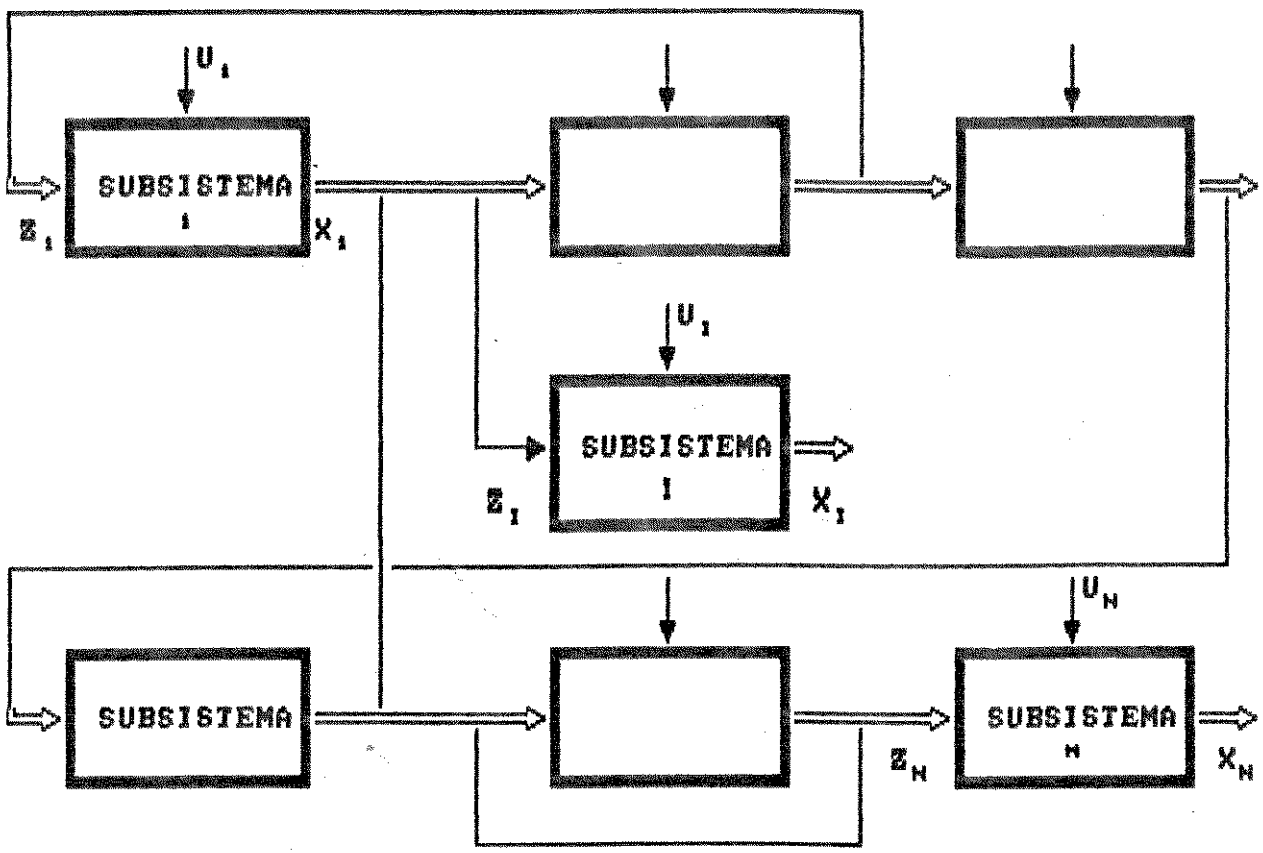


Figura 3.3 - Sistema dinâmico com subsistemas interconectados

Cada subsistema tem seus próprios vetores de estado, controle e saída. Os vetores de estado e controle do sistema dinâmico global são conjuntos definidos daqueles vetores. Neste caso, as soluções dos problemas de otimização dos subsistemas podem ser determinadas em paralelo e combinadas de alguma forma para a obtenção de uma solução ótima para o problema global.

Controle hierárquico multinível é uma técnica de decomposição e coordenação para se resolver problemas de

otimização dinâmica de sistemas de grande porte. O controle hierárquico implica na decomposição de um problema global P (otimizar um critério de desempenho associado com um sistema dinâmico) em subproblemas independentes P_i (i=1,...,N) que se mantém interligados via uma unidade de coordenação separada.

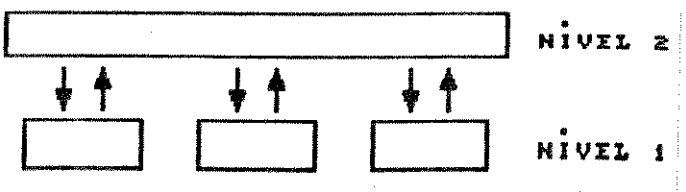


Figura 3.4 - Estrutura hierárquica em dois níveis

As principais vantagens do ponto de vista computacional das aplicações de algoritmos de controle ótimo hierárquico a sistemas de grande porte são :

(i) Uma redução significativa do tempo de computação pode ser obtida, especialmente se um sistema de múltiplos processadores for utilizado para explorar o paralelismo natural e intrínseco da estrutura multinível.Esta característca é extremamente importante para permitir uma investigação da aplicabilidade dos algoritmos de controle ótimo multinível na computação on-line de problemas de controle prático.

(ii) A decomposição do problema de controle em subproblemas independentes permite, apesar do aumento do número de variáveis envolvidas no procedimento de otimização, uma distribuição da

memória total de cálculo por várias unidades de processamento de um sistema de múltiplos processadores, uma vez que as unidades de processamento locais têm de resolver somente um problema de dimensão reduzida.

(iii) Decomposição do problema global consiste de vários módulos separados que podem ser indivíualmente implementados e testados de forma mais conveniente do que no caso do problema global.

Estas vantagens podem apresentar um maior ou menor grau de utilização eficiente de um sistema de múltiplos processadores quando um método de decomposição e coordenação é aplicado a um problema de controle ótimo particular. Os principais fatôres que afetam a eficiência da solução do problema de controle ótimo são :

a) a estrutura e a dimensão do problema ;

b) a simplicidade dos subproblemas, uma vez que os métodos iterativos envolvem a resolução dos subproblemas várias vezes ;

c) as características do sistema de múltiplos processadores.

Um desempenho computacional superior pode ser obtido através da investigação do paralelismo subjacente na estrutura de cálculo. Esta atitude implica na modificação do algoritmo para se atingir um maior grau de paralelismo, mantendo uma dependência com a arquitetura de computação utilizada. A partir deste contexto, adotamos o seguinte procedimento para combinar a escolha da metodologia de controle

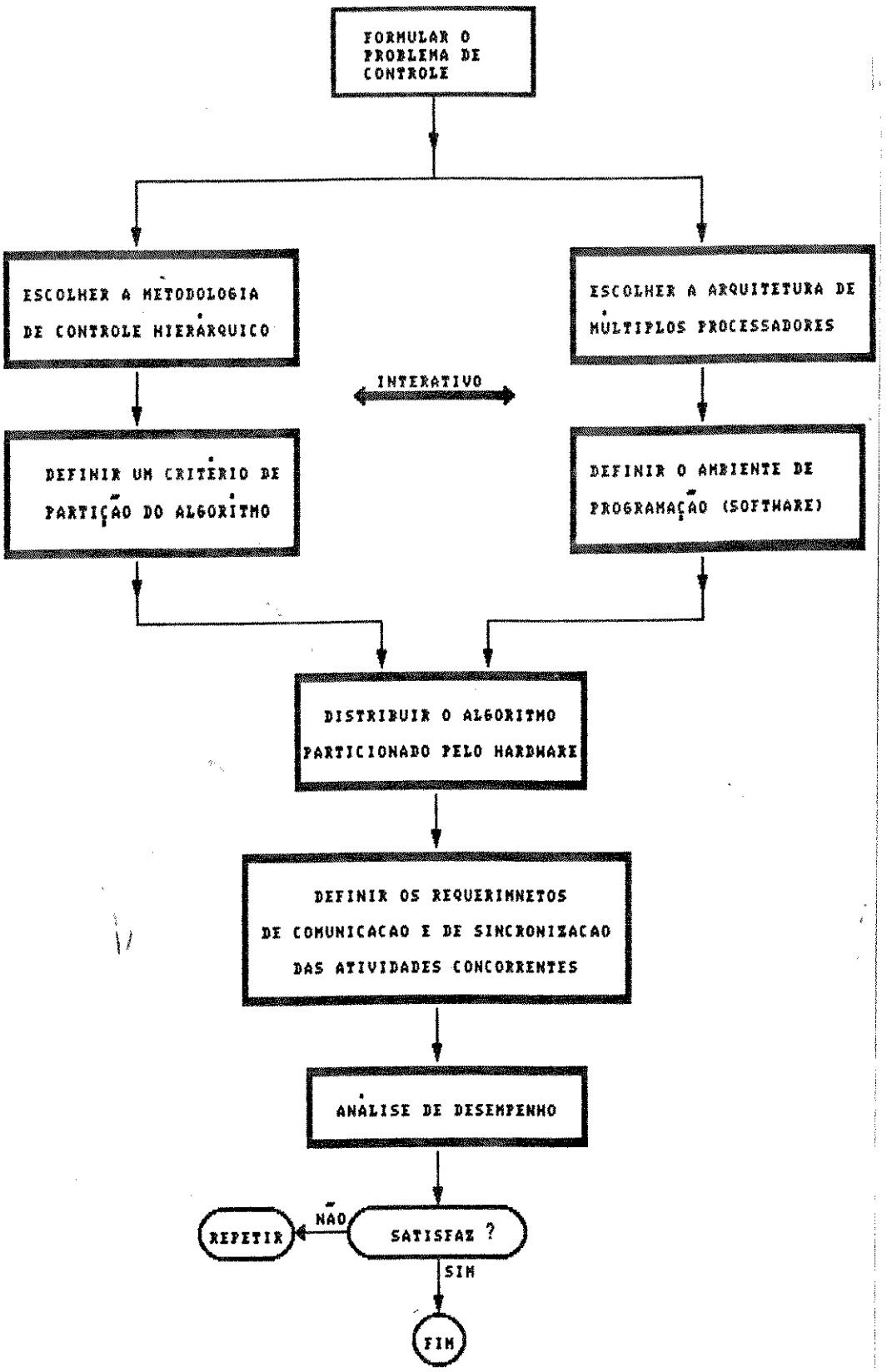


Figura 3.5 - Procedimento de paralelização e implementação

hierárquico e a estratégia de paralelização para o processamento paralelo sobre sistemas de múltiplos processadores.

3.3 - Formulação do problema de otimização dinâmica

Considere o sistema dinâmico S descrito pelas equações a diferenças :

$$x[k+1] = F(x[k], u[k]) \; ; \qquad (1.a)$$

$$x[0] = x_0 \; ; \qquad k = 0, \ldots, T-1 \qquad (1.b)$$

Este sistema, mostrado na fig. 3.6, pode ser visto como um conjunto de restrições de igualdade, onde :

$$x[k] = \begin{bmatrix} x_1[k] \\ \vdots \\ x_n[k] \end{bmatrix} \in \mathbb{R}^n \text{ é o vetor de estado}$$

$$u[k] = \begin{bmatrix} u_1[k] \\ \vdots \\ u_n[k] \end{bmatrix} \in \mathbb{R}^m \text{ é o vetor de controle}$$

$$F(\,,\,) = \begin{bmatrix} F_1(\,,\,) \\ \vdots \\ F_n(\,,\,) \end{bmatrix} \quad F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n \quad \text{é uma}$$

função de classe C^1 que fornece a estrutura do sistema S.

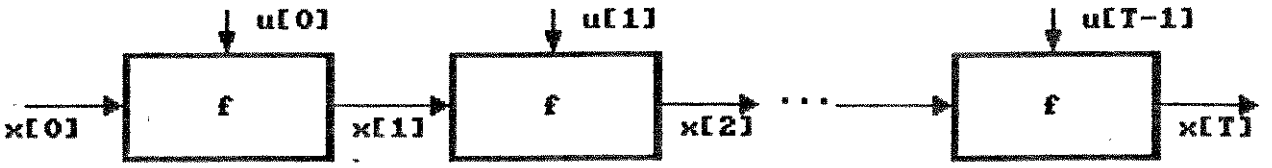


Figura 3.6 - Sistema dinâmico discreto

Essencialmente, este sistema pode ser visto como um conjunto sequencial de restrições de igualdade, onde a sequência x [k] é determinada pela sequência u [k].

Para o sistema (1), consideramos o funcional do custo da forma :

$$J = (1/2)\, x^t[k]\, W\, x\,[k] + \sum_{k=0}^{T-1} (1/2) \left\{ x^t[k]\, Q\, x\,[k] + u^t[k]\, R\, u\,[k] \right\} \quad (2)$$

onde : matrizes W e Q são diagonais e semi-definidas positivas, e a matriz R é diagonal e definida positiva. No segundo membro, o primeiro termo de (2), chamado de função terminal, mostra a dependência do funcional do custo com o estado final e a quantidade no somatório, chamada de função intermediária, mostra a dependência do funcional com as sequências de controle e de estado.

O problema de otimização dinâmica é encontrar a sequência u [k] que minimize J.

3.4 - Decomposição do problema

Vamos considerar que o sistema S pode ser decomposto em N subsistemas interconectados S_i pela partição do vetor de estado x e do vetor de controle u :

Para $i = 1,\ldots,N$,

$$S_i : \; x_i[k+1] = g_i(x_i[k],u_i[k]) + h_i(x[k]) \; , \qquad (3)$$

onde $x_i[k] \in \mathbb{R}^{n_i}$ é o vetor de estado do i-ésimo subsistema, tal que $\mathbb{R}^{n} = \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \ldots \times \mathbb{R}^{n_N}$; $u_i[k] \in \mathbb{R}^{m_i}$ é o vetor de controle local para controlar o i-ésimo subsistema tal que $\mathbb{R}^{m} = \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \times \ldots \times \mathbb{R}^{m_N}$; $g_i : \mathbb{R}^{n_i} \times \mathbb{R}^{m_i} \to \mathbb{R}^{n_i}$ representa a dinâmica dos subsistemas desacoplados; $h_i : \mathbb{R}^{n} \to \mathbb{R}^{n_i}$ é a função que representa a interconexão do i-ésimo subsistema dentro do sistema global S. A divisão do sistema S em N subsistemas pode ser realizada de forma puramente matemática.

A fim de garantir um desempenho satisfatório do sistema S a despeito da partição "on-off" dos subsistemas, vamos supor que o funcional do custo (1) seja aditivamente separável e representado por :

$$\min_{u} J = \sum_{i=1}^{N} \min_{u_i} J_i \qquad (4)$$

onde :

$$J_i = (1/2)\, x_i^t[k]\, W_i\, x_i[k]$$

$$+ \sum_{k=0}^{T-1} (1/2) \left\{ x_i^t[k]\, Q_i\, x_i^t[k] + u_i^t[k]\, R_i\, u_i[k] \right\} \quad (5)$$

Agora, o problema é obter as sequências de estado e de controle que minimize (5) sujeito a (3).

3.5 - Método Predição de Interação

Para a aplicação deste método, consideramos o sistema dinâmico S linear descrito da seguinte forma :

$$x[k+1] = A\, x[k] + B\, u[k] \quad (6)$$

Uma comparacao entre (1) e (6) revela as seguintes associações :

$$g_i(x_i[k], u_i[k]) \longrightarrow A_i x_i[k] + B_i u_i[k]$$

$$h_i(x[k]) \longrightarrow \sum_{\substack{j=1 \\ j \neq i}}^{N} A_{ij}\, x_j[k]$$

e a dinâmica do i-ésimo subsistema toma a seguinte forma :

$$x_i[k+1] = A_i x_i[k] + B_i u_i[k] + z_i[k] \quad ; \quad x_i[0] = x_{i0} \quad (7)$$

$$z_i[k] = \sum_{j \neq i}^{N} A_{ij}\, x_j[k] \quad (8)$$

com

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & \ddots & & \vdots \\ \vdots & & A_{ii} & \vdots \\ A_{N1} & \cdots & & A_{NN} \end{bmatrix}$$

B = matriz diagonal $[B_1 \ \ldots \ B_i \ \ldots \ B_N]$

Vamos, primeiramente, introduzir um conjunto de multiplicadores de Lagrange $\beta_i[k]$ de dimensão n_i e o vetor adjunto $\lambda_i[k+1]$ de dimensão n_i para acrescentar a restrição de igualdade (8) e a restrição dinâmica (7) ao funcional do custo (5). O Hamiltoniano para o i-ésimo subsistema é definido por :

$$H_i(\ .\ ) = (1/2)\Big(x_i^t[k]\ Q_i\ x_i[k] + u_i^t[k]\ R_i u_i[k] + \beta_i^t[k]\ z_i[k] - \sum_{j \neq i}^{N} \beta_j^t\ A_{ji}\ x_i[k] + \lambda_i^t[k+1]\ \Big(A_i\ x_i[k] + B_i\ u_i[k\ ] + z_i[k]\Big) \quad (9)$$

O procedimento de solução adotado é resolver as condições necessárias de otimalidade iterativamente numa estrutura em em dois níveis. As condições necessárias para otimalidade são :

$$\frac{\partial H_i}{\partial x_i[k]} = \lambda_i[k] \quad \therefore \quad \lambda_i[k] = Q_i x_i[k] + A_i^t\ \lambda_i[k+1]$$

$$- \sum_{j \neq i}^{N} A_{ji}^{t} \beta_j[k] \quad ; \quad \lambda_i[T] = W_i x_i[T] \qquad (10)$$

$$\frac{\partial H_i}{\partial u_i[k]} = 0 \quad \therefore \quad u_i[k] = - R_i^{-1} B_i^{t} \lambda_i[k+1] \qquad (11)$$

$$\frac{\partial H_i}{\partial \lambda_i[k+1]} = 0 \quad \therefore x_i[k+1] = A_i x_i[k] + B_i u_i[k] + z_i[k];$$

$$x[o] = x_{io} \qquad (12)$$

Observe que as equações (10) e (12) representam um problema de dupla condição de contorno. Estas equações podem ser desacopladas pela definição do vetor adjunto $\lambda_i[k]$ como segue :

$$\lambda_i[k] = P_i[k] \, x_i[k] + s_i[k] \qquad (13)$$

Substituindo (13) em (10) e (12) temos :

$$x_i[k+1] = \psi_i[k+1] \left(A_i x_i[k] + z_i[k] - B \, R_i^{-1} B_i^{t} s_i[k+1] \right) \qquad (14)$$

e $P_i[k] \, x_i[k] + s_i[k] = Q_i x_i[k] + A_i^{t} P_i[k+1] \, x_i[k+1]$

$$+ A_i^{t} s_i[k+1] - \sum_{j \neq i}^{N} A_{ji}^{t} \beta_j[k] \qquad (15)$$

onde :

$$\psi_i[k+1] = \left(I_i - B_i \left[R_i + B_i^{t} P_i[k+1] \, B_i \right]^{-1} B_i^{t} P_i[k+1] \right) \qquad (16)$$

e I_i é a matriz identidade ($m_i \times m_i$).

Resolvendo para $x_i[k+1]$ e eliminando-o, obtemos :

$$P_i[k] = Q_i + A_i^t P_i[k+1]\ \psi_i[k+1]\ A_i \qquad (17)$$

$$s_i[k] = -A_i^t P_i[k+1]\ \psi_i[k+1] A_i B_i R_i\ B_i^{-1} s_i^t[k+1] + A_i^t P_i[k+1]\ \psi_i[k+1] A_i z_i[k] + A_i^t s_i[k+1] - \sum_{j \neq i}^{N} A_{ji}^t\ \beta_j[k] \qquad (18)$$

onde $P_i[k]$ é a matriz $(n_i \times n_i)$ simétrica solução da equação discreta Riccati. As condições finais para as equações (17) e (18) podem ser determinada comparando a equação (13) para $k = T$ com a equação (15) :

$$P_i[T] = W_i \ ; \qquad (19)$$

$$s_i[T] = 0 \qquad (20)$$

Seguindo esta formulação, o controle ótimo para o problema de otimização pode ser escrito como :

$$u_i[k] = Y_i[k+1]\ A_i x_i[k+1] - \left[\ Y_i[k+1] - R_i^{-1} B_i^t\ \right] s_i[k+1] - Y_i[k+1]\ z_i[k] \qquad (21)$$

$$Y_i[k+1] = \left(\ R_i + B_i^t P_i[k+1]\ B_i\ \right)^{-1} B_i^t P_i[k+1] \qquad (22)$$

A escolha da estratégia de otimização, no entanto, não pode atingir um desempenho ótimo

$$J^* = \sum_{i=1}^{N} J_i^* \qquad (23)$$

a menos que todos os acoplamentos estejam ausentes ($z_i[k] = 0$, $i = 1,\ldots,N$). A solução deste problema está na formação de um problema de segundo nível que essencialmente prediz novos vetores de coordenação z [k] e β [k]. Para esta pro-

posta a função Lagrangeana L (x[k],u[k],z[k],β[k],λ[k+1]) torna-se aditivamente separável :

$$L = \sum_{i=1}^{N} L_i \qquad (24)$$

onde :

$$L_i = \Big\{ (1/2)\, x_i^t[T]\, W_i\, x_i^t[T]$$

$$\sum_{k=0}^{T-1} \Big[(1/2)\Big(x_i^t[k]\, Q_i\, x_i[k] + u_i^t[k]\, R_i u_i[k] \Big)$$

$$+ \beta_i^t[k]\, z_i[k] - \sum_{j \neq i}^{N} \beta_j^t\, A_{ji}\, x_i[k]$$

$$+ \lambda_i^t[k+1]\big(- x_i[k+1] + A_i x_i[k] + B_i\, u_i[k] + z_i[k]\big)\Big]\Big\} \qquad (25)$$

Os valores de $z_i[k]$ e $\beta_i[k]$ podem ser obtidos por :

$$\frac{\partial L_i}{\partial\, \beta_i[k]} = 0 \quad \therefore \quad z_i[k] - \sum_{j \neq i}^{N} A_{ij} x_j[k] = 0 \qquad (26)$$

$$\frac{\partial\, L_i}{\partial\, z_i[k]} = 0 \quad \therefore \quad \beta_i[k] + \lambda_i[k+1] = 0 \qquad (27)$$

$i = 1,\ldots,N$

Este procedimento de otimização resulta na seguinte estrutura em dois níveis :

Nivel 1. Resolver para $i = 1,\ldots,N$, $k = 0,\ldots,T-1$

$$P_i[k] = Q_i + A_i^t P_i[k+1]\ \psi_i[k+1]\ A_i$$

$$s_i[k] = -A_i P_i[k+1]\ \psi_i[k+1]\ B_i R_i\ B_i^{-1} s_i^t[k+1]$$

$$+ A_i^t P_i[k+1]\ \psi_i[k+1]\ z_i[k]$$

$$+ A_i^t s_i[k+1] - \sum_{j \neq i}^{N} A_{ji}^t\ \beta_j[k] \quad ; \quad s_i[o] = 0$$

$$x_i[k+1] = \psi_i[k+1]\ A_i x_i[k]$$

$$- \psi_i[k+1]\ B_i R_i^{-1} B_i^t s_i[k+1]$$

$$+ \psi_i[k+1]\ z_i[k] \quad ; \quad x_i[o] = x_{io}$$

$$u_i[k] = Y_i[k+1]\ A_i x_i[k+1]$$

$$- \left(Y_i[k+1] - R_i^{-1} B^t{}_i \right) s_i[k+1]$$

$$- Y_i[k+1]\ z_i[k]$$

$$\lambda_i[k] = A_i^t \lambda_i[k+1] + Q_i x_i[k] - \sum_{j \neq i}^{N} A_{ji}^t\ \beta_j[k]$$

Nível 2. Da iteração de l a $(l+1)$

$$\begin{bmatrix} z[k] \\ \beta[k] \end{bmatrix}^{l+1} = \begin{bmatrix} \text{col}\ \left(\sum A_{ij}\ x_j[k] \right) \\ -\ \text{col}\ \left(\lambda_i[k+1] \right) \end{bmatrix}^{l}$$

O algoritmo para a estrutura em dois níveis é :

Passo 1 : inicializar $z^0[k]$ e $\beta^0[k]$ $(k=0,\ldots,T-1)$ e enviar ao nível 1. Fazer $l=1$.

Passo 2 : no nível 1, solucionar os N subproblemas de otimização independentes :

(a) resolver (17) com a condição final (19) para obter $P_i[k]$ e armazenar (somente uma vez), $i = 1,\ldots,N$;

(b) utilizar a condição final (20) em (18) para obter $s_i^l[k]$, $i = 1,\ldots,N$;

(c) da condição inicial $x_i[0] = x_{i0}$ obter $x_i^l[k]$ de (14) e subsequentemente $u_i^l[k]$. Também $\lambda_i^l[k+1]$ é obtido de (10). Enviar $x_i^l[k]$ e $\lambda_i^l[k+1]$ ao nível 2. $i = 1,\ldots,N$.

Passo 3 : No nível 2, $x^l[k]$ e $\lambda^l[k+1]$ são subsitituídos em (26) e (27) respectivamente e obter $z^{l+1}[k]$ e $\beta^{l+1}[k]$. Se

$$\begin{bmatrix} z[k] \\ \beta[k] \end{bmatrix}^{l+1} = \begin{bmatrix} z[k] \\ \beta[k] \end{bmatrix}^{l}$$

O valor de u [k] armazenado é a trajetória ótima de controle. Caso contrário fazer $(l=l+1)$ e voltar ao Passo 2.

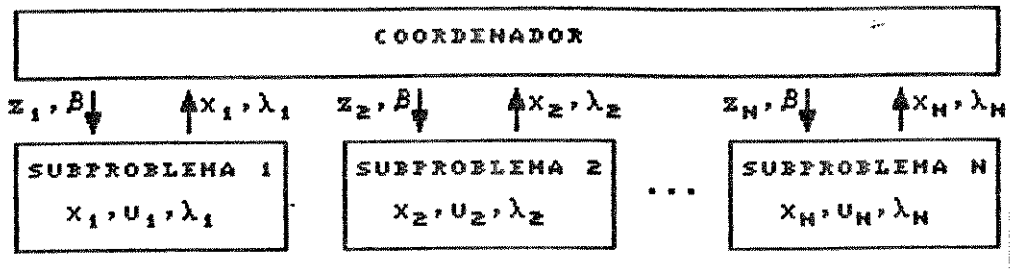


Figura 3.7 - Estrutura em dois níveis

3.5.1 - Paralelização do algoritmo

A paralelização do algoritmo para processamento paralelo numa arquitetura de múltiplos processadores é determinada pelo critério adotado para a escolha da partição do algoritmo. Para o estudo da partição do algoritmo, definimos dois critérios : o primeiro critério explora apenas o paralelismo natural resultante da estrutura hierárquica em dois níveis; o segundo critério implica na modificação da estrutura de cálculo do algoritmo para a obtenção de um maior grau de paralelismo. Sob este aspecto definimos duas etapas de implementação.

3.5.1.1 - Primeira Etapa

i) - Partição do Algoritmo

Da decomposição do problema de otimização, podemos observar três características importantes :

(1) Os subproblemas apresentam o mesmo conjunto de equações, ou seja, os subproblemas são semelhantes;

(2) há a possiblidade de se ter um número arbitrário de subproblemas de dimensões variadas;

(3) os subproblemas são assíncronos, isto é, a execução

do cálculo de otimização não envolve interação com outros subproblemas.

Sob o enfoque das características apresentadas, podemos especificar para os N subproblemas de otimização do nível inferior da estrutura em dois níveis N tarefas tal que os subproblemas tenham a mesma dimensão. Considerando a disponibilidade de (N+1) unidades de processamento idênticas será mais eficiente para o processamento paralelo que haja N subsistemas de mesma dimensão. Se isto não ocorrer, o sistema de múltiplos processadores pode ser visto como um conjunto de processadores de velocidades de processamento diferentes e por conseguinte uma consideração especial deve ser dada a alocação adicional de partes do algoritmo para os processadores mais rápidos, o que complica consideravelmente implementação do algoritmo. Sob este aspecto, especificamos as seguintes tarefas :

TAREFA i, ($i=1,\ldots N$)
- coletar $z_i[k]$ e $\beta[k]$;
- resolver $P_i[k]$ (somente uma vez) e $s_i[k]$;
- calcular $x_i[k]$, $u_i[k]$ e $\beta_i[k]$;
- enviar para a TAREFA (N+1)

$$\begin{bmatrix} x_i[k] \\ \beta_i[k] \end{bmatrix}$$

TAREFA j, ($j=N+1$)
- predizer $z[k]$ e $\beta[k]$ iniciais (so uma vez);
- enviar $z_i[k]$ e $\beta_i[k]$ a cada TAREFA i ;
- coletar $x[k]$ e $\beta[k]$;
- calcular $z[k]$;
- testar $\begin{bmatrix} z^{l+1}[k] - z^{l}[k] \\ \beta^{l+1}[k] - \beta^{l}[k] \end{bmatrix} < \varepsilon$ (número muito pequeno)

Da partição do algoritmo, o cálculo de :

$$\beta_i[k] \text{ e } \lambda_i[k+1]$$

é realizado por cada TAREFA i (i=1,...,N), reduzindo o cálculo da tarefa de coordenação.

ii) - Distribuição das Tarefas

Nas implementações que realizamos, a disponibilidade de unidades de processamento corresponderam ao número de de subproblemas, de forma que para cada tarefa associamos uma unidade de processamento.

Um ponto a ser ressaltado é o fato de que durante a execução da TAREFA (N+1) na unidade de processamento (N+1) as outras unidades ficarão ociosas e vice-versa.

iii) - Comunicação e Sincronização

A estrutura de paralelização obtida, resulta numa comunicação síncrona entre as TAREFA i, (i=1,...,N) e a TAREFA j, (j=1,...,N). No sistema de múltiplos processadores com memória compartilhada, a comunicação entre as tarefas se realiza através de variáveis compartilhadas na memória global. A sincronização das tarefas é muito simples, uma vez que a execução da tarefa de coordenação e das tarefas relativas aos subproblemas não ocorrem simultaneamente. Neste caso, cada TAREFA i (i=1,..,N) tem dados (variáveis) informativos para indicar que :

1) TAREFA (N+1) poderá ler a solução do seu subproblema;
2) TAREFA (N+1) fez uma nova especificação;
3) TAREFA (N+1) determinou a solução global.

Assim, nas relações Fork e Join, associamos N dados, caracterizando dois pontos de sincronização. Uma consideração especial foi dada ao problema de conflito quando do acesso a memória global pelas tarefas conforme pode ser depreendido nas relações Fork e Join da fig.3.8.Para reduzir o conflito utilizamos a seguinte estratégia :

1) As variáveis compartilhadas na memória global referentes a cada tarefa são copiadas na memória local da respectiva unidade de processamento e vice-versa durante a escrita e a leitura respectivamente;
2) Introduzimos um retardo entre dois pedidos consecutivos de acesso a memória global por uma tarefa nas relações Fork e Join.

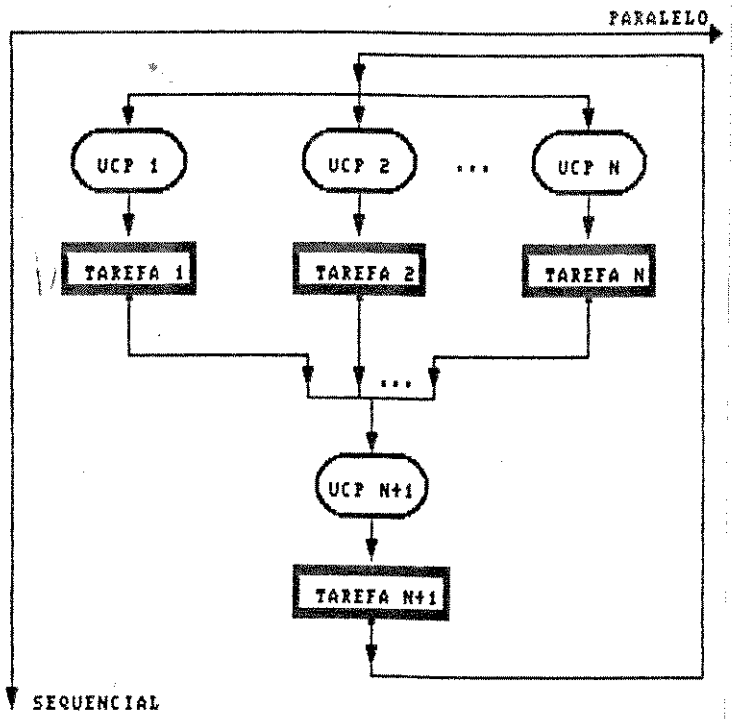


Figura 3.8 - Estrutura de paralelização

3.5.1.2 - Segunda Etapa

A paralelização do algoritmo obtida a partir do paralelismo natural apresenta um bom desempenho computacional quando comparada com a execução sequencial do algoritmo. No entanto, podemos verificar o seguinte aspecto : a tarefa de coordenação tende a limitar a potência de processamento global do algoritmo. A TAREFA (N+1) manipula variáveis globais, ou seja, $z[k] \in \mathbb{R}^n$ e $\beta[k] \in \mathbb{R}^n$ independente da ordem dos subproblemas. Embora o tempo de execução de cada TAREFA i cresça rapidamente com a ordem do subsistema (uma análise mais detalhada é realizada no capítulo 4), o tempo de execução da TAREFA (N+1) praticamente permanece o mesmo. Vamos supor então, que possamos variar o número de subsistemas de mesma ordem tal que o tempo de execução seja o mesmo para cada TAREFA i e desprezar os tempos gastos com a comunicação e a sincronização. Considere que o número de P unidades de processamento idênticas seja igual a (N+1). Na fig. 3.9, mostramos uma aproximação de como o tempo total de execução tende para o tempo de execução da TAREFA (N+1) mais o tempo de execução das tarefas relativas aos subproblemas de menor dimensão.

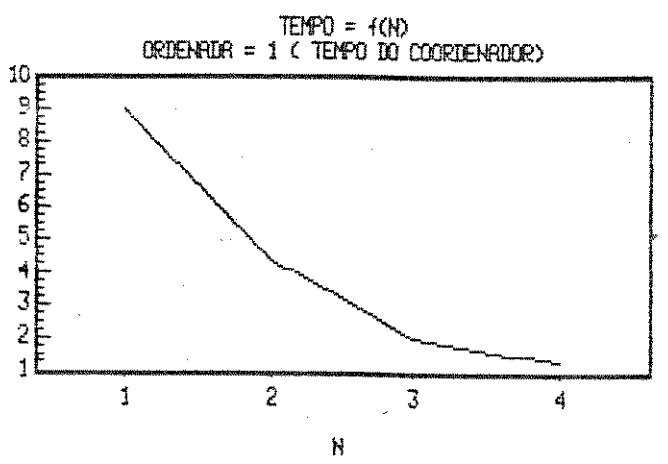


Figura 3.9

Em adição, é possível adicionarmos mais unidades de processamento para cada TAREFA i,de tal forma que o tempo total de execução tendesse para o tempo de execução da TAREFA (N+1).

Isto nos levou a propor um segundo critério que explora o paralelismo na tarefa de coordenação.

i) - Partição do Algoritmo

Esta implementação se caracteriza pela subdivisão da tarefa de coordenação em N subtarefas. O vetor de coordenação é formado pelo vetor de interconexão e pelos multiplicadores de Lagrange. Como não há nenhuma relação explicita entre o cáluclo de z [k] e β [k] é possível particionar o vetor de coordenação em N subvetores ou seja :

$$\text{subvetor i-ésimo de coordenacão} = \begin{bmatrix} z_i[k] \\ \beta_i[k] \end{bmatrix} \text{ tal que : } \sum_{i=1}^{N} n_i = n$$

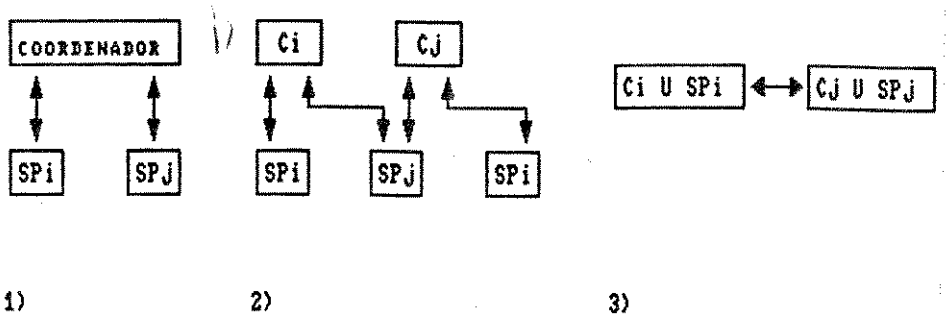
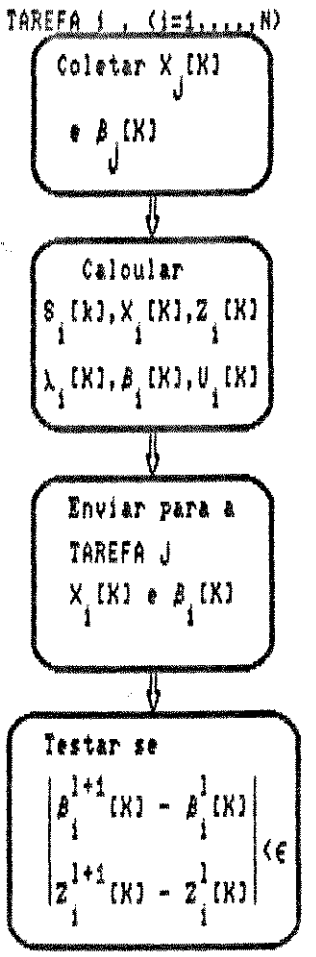


Figura 3.10 - Etapas de partição

O cálculo de $z_i[k]$ $(i=1,\ldots,N)$ depende da variável de estado $x_j[k]$ $(j=1,\ldots,N \; ; \; j\neq i)$ e o cálculo de $s_i[k]$ e de

$\lambda_i[k+1]$ dependem de $\beta_j[k]$. Quando associamos os N subvetores de coordenação aos N subproblemas de otimização, as relações de dependência mencionadas determinam o grau de concorrência resultante deste novo critério de partição da estrutura de cálculo. Com este procedimento, especificamos as seguintes tarefas :



Da partição adotada podemos verificar que:

(a) O número de tarefas é igual ao número de subsistemas;

(b) Todas as tarefas apresentam o mesmo conjunto de equações, ou seja, as tarefas são semelhantes;

(c) Todas as tarefas estão restritas a um só nível computacional.

ii) - Distribuição das Tarefas

Seguindo o mesmo procedimento da primeira implementação, associamos às N tarefas P processadores, tal que P = N.

iii) - Comunicação e Sincronização

Nesta implementação dividimos cada tarefa em quatros pontos de comunicação, dois para escrita e dois para a leitura. Nos pontos de comunicação para escrita não há sincronização entre as tarefas. Os pontos de comunicação para leitura correspondem ao ponto de sincronização global. A execução das tarefas é sincronizada em cada iteração, ou seja, a execução de uma tarefa não pode iniciar uma nova iteração enquanto as outras tarefas não tenham terminado sua fase de leitura das variáveis compartilhadas. É possível tentarmos remover a sincronização considerando os seguintes pontos :

a) Se o tempo de execução dos passos de cálculo entre os pontos de leitura e escrita ser muito significativos em relação ao tempo de comunicação ;

b) Se a velocidade relativa dos programas seja mínima, isto é, os programas estejam bem balanceados.

Sob este aspecto, a execução do algoritmo paralelo será totalmente assíncrona. Apesar de que em

algumas aplicações que fizemos foi possível tal relaxamento na sincronização, preferimos manter o ponto de sincronização global para assegurar uma forma correta de execução, uma vez que o tempo extra gasto com a sincronização por iteração foi pouco significativo e que a convergência do algoritmo na execução assíncrona não foi ainda garantida.

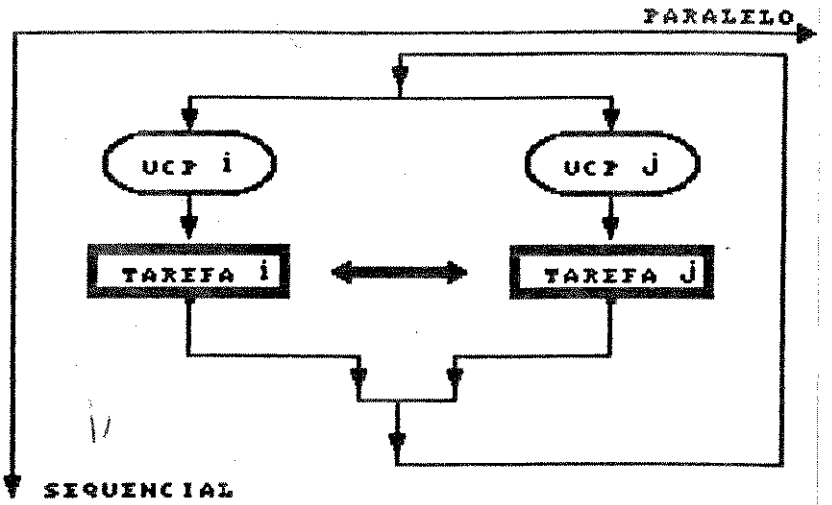


Figura 3.11 - Estrutura de Paralelização

3.6 - Metodologia de Coordenação pelo Coestado

Para a aplicação deste método, consideramos o sistema dinâmico S descrito da seguinte forma :

$$x[k+1] = f(x[k]) + B\,u[k] \quad , \quad x[o] = x_o \qquad (28)$$

onde f(x[k]) é convexa. Usando o esquema de decomposição dado em [5] a equação (28) pode ser escrita como :

$$x_i[k+1] = f_i(x_i[k]) + B_i u_i[k] + z_i[k] \quad ; \quad x_i[o] = x_{io} \qquad (29)$$

$$z_i[k] = \phi_i(x_j[k]) \qquad j=1,\ldots,N \quad e \quad j\neq i \qquad (30)$$

Uma comparação entre (1) e (6) revela as seguintes associações :

$$g_i(x_i[k], u_i[k]) \longrightarrow f_i(x_i[k]) + B_i u_i[k]$$

$$h_i(x[k]) \longrightarrow \sum_{j\neq i}^{N} \phi_{ij}(x_j[k])$$

com

$$B = \text{matriz diagonal } [B_1 \ \ldots \ B_i \ \ldots \ B_N]$$

e onde: $z_i[k] \in \mathbb{R}^{n_i}$ e $\phi_i(x_j[k])$ representa o vetor de inteconexão de cada subsistema.

Enquanto as equações (28) e (2) especificam as equações do problema global, as equações (5), (29) e (30) caracterizam o i-ésimo subproblema.

O procedimento de solução pelo Princípio do Máximo compreende : a definição do Hamiltoniano H como :

$$H = \sum_{i=1}^{N} H_i(x_i[k], u_i[k], z_i[k], \beta[k], \lambda_i[k+1]) \qquad (31)$$

onde o sub-Hamiltoniano é :

$$H_i(\,.\,) = (1/2)\left(x_i^t[k]\, Q_i\, x_i[k] + u_i^t[k]\, R_i\, u_i[k] \right.$$

$$+ \beta_i^t[k]\, z_i[k] - \sum_{j \neq i}^{N} \beta_j^t[k] \phi_{ji}(x_i[k])$$

$$\left. + \lambda_i^t[k+1] \left(f_i(x_i[k]) + B_i\, u_i[k] + z_i[k] \right) \right) \qquad (32)$$

bem como a satisfação das condições necessária de otimalidade em termos dos gradientes generalizados g_p, $(p=1,2,3,4,5)$:

$$g_1(x_i[T]) = W_i x_i[T] - \lambda_i[T] = 0 \; ; \qquad (33)$$

$$g_2(x_i[k]) = \frac{\partial H_i}{\partial x_i[k]} - \lambda_i[k] = 0 \; ; \qquad (34)$$

$$g_3(u_i[k]) = \frac{\partial H_i}{\partial u_i[k]} = 0 \; ; \qquad (35)$$

$$g_4(z_i[k]) = \frac{\partial H_i}{\partial z_i[k]} = 0 \; ; \qquad (36)$$

$$g_5(\beta_i[k]) = \frac{\partial H}{\partial \beta_i[k]} = 0 \qquad (37)$$

A estrutura em dois níveis é obtida explorando a idéia básica dos gradientes generalizados na divisão das variáveis do sistema global em dois grupos distintos : variáveis independentes e dependentes. Na estrutura em dois níveis, as variáveis dependentes são aquelas que afetam a otimalidade dos subproblemas locais, enquanto as variáveis independentes são as quantidades que devem ser modificadas para coordenar as soluções locais até que a solução global seja atingida. A estrutura em dois níveis baseada na coordenação pelo coestado considera as variáveis da seguinte forma :

Variáveis dependentes : $x_i[k]$, $u_i[k]$, $\beta_i[k]$, $z_i[k]$

Variável independente : $\lambda_i[k]$

Neste // caso, as equações (35), (36), (37) juntamente com (29) são escolhidas para representar os subproblemas do primeiro nível enquanto as equações (33) e (34) são manipuladas no segundo nível. A estrutura em dois níveis toma a seguinte forma :

Nível 1 (subproblemas)

Resolver para $i=1,\ldots,N$; $k=0,\ldots,T-1$

$$x_i[k+1] = f_i(x_i[k]) + B_i u_i[k] + z_i[k]; \quad x[0] = x_{i0}$$

$$u_i[k] = - R_i^{-1} B_i^t \lambda_i[k+1]$$

$$z_i[k] = \sum_{j \neq i}^{N} \phi_{ij}(x_j[k])$$

$$\beta_i[k] - \lambda_i[k+1] = 0$$

Nível 2 (Coordenador)

$$\lambda_i[k] = Q_i x_i[k] + \left[\frac{\partial f_i(x_i[k])}{\partial x_i[k]} \right]^t \lambda_i[k+1]$$

$$- \left[\sum_{j \neq i}^{N} \frac{\partial \phi_{ji}(x_i[k])}{\partial x_i[k]} \right]^t \beta_i[k] \quad ; \quad \lambda_i[T] = W_i x_i[T]$$

O algoritmo toma a seguinte forma :

Passo 1 : Dado o valor inicial das variáveis de coestado $\lambda_i^0[k]$, $i=1,\ldots,N$ e $k=0,\ldots,T-1$ envia-los aos subproblemas de otimização. Fazer a iteração $l=1$

Passo 2 : Resolver os subproblemas para λ_i^l e obter para $i=1,\ldots,N$:

(i) $u_i[k]$, $k=0,\ldots,T-1$, através da equação (35)

(ii) $x_i^l[k]$, $z_i[k]$, $\beta_i[k]$ $k=0,\ldots,T$, utilizando as equações (36)-(38).

e enviar ao primeiro nível.

Passo 3 : No nível superior calcular :

$$\varphi(\lambda_i^l[k]) = \left\{ Q_i x_i[k] + \left(\frac{\partial f_i(x_i[k])}{\partial x_i[k]} \right)^t \lambda_i[k+1] \right.$$

$$\left. - \sum_{j \neq i}^{N} \frac{\partial \phi_{ji}^t(x_i[k])}{\partial x_i[k]} \beta_j[k] \right\}^l ;$$

$$G(\lambda_i^l[k]) \equiv \sum_{k=0}^{T-1} (H_i^l - \lambda_i^l[k+1] \; x_i^l[k+1]) + x_i^t[T] \; W_i x_i[T]$$

Passo 4 : Comparar se

$$\| \lambda_i^l[k] - \varphi_i^l(\lambda_i[k]) \| \leq \varepsilon \quad (i=1,\ldots,N)$$

ε é um número positivo muito pequeno. Se verdadeiro a solução global foi atingida. Caso contrário ir para o passo 5

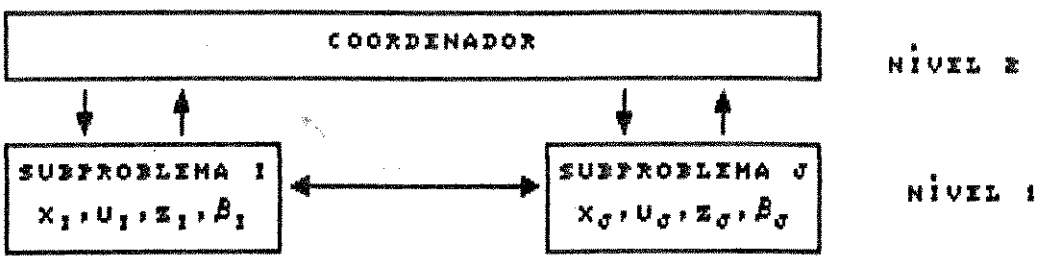
Passo 5 : Uma variação relativa de $G(\lambda_i^l[k])$ com relação a uma pequena variação de $\lambda_i^l[k]$ é calculada e disto determina-se o tamanho do passo ρ correspondente a um certo aumento percentual em $G_i(\lambda_i^l[k])$.

Passo 6 : Atualizar o coestado conforme :

$$\lambda_i^{l+1}[k] = \lambda_i^l[k] + \rho_i^l \mid \lambda_i^l[k] - \varphi_i^l(\lambda_i^l[k]) \mid$$

$i=1,\ldots,N$ e $k=0,\ldots,T$

Passo 7 : Fazer $l=l+1$ e voltar ao Passo 2.



3.12 - Estrutura em dois níveis

São características básicas dessa estrutura em dois níveis :

1) As restrições de interconexão são sempre satisfeitas durante o procedimento iterativo, significando que as soluções são factíveis.

2) Os subproblemas de otimização não necessitam resolver o problema de dupla condição de contorno.

3) Um procedimento interativo entre os subsistemas resultante da factibilidade das soluções.

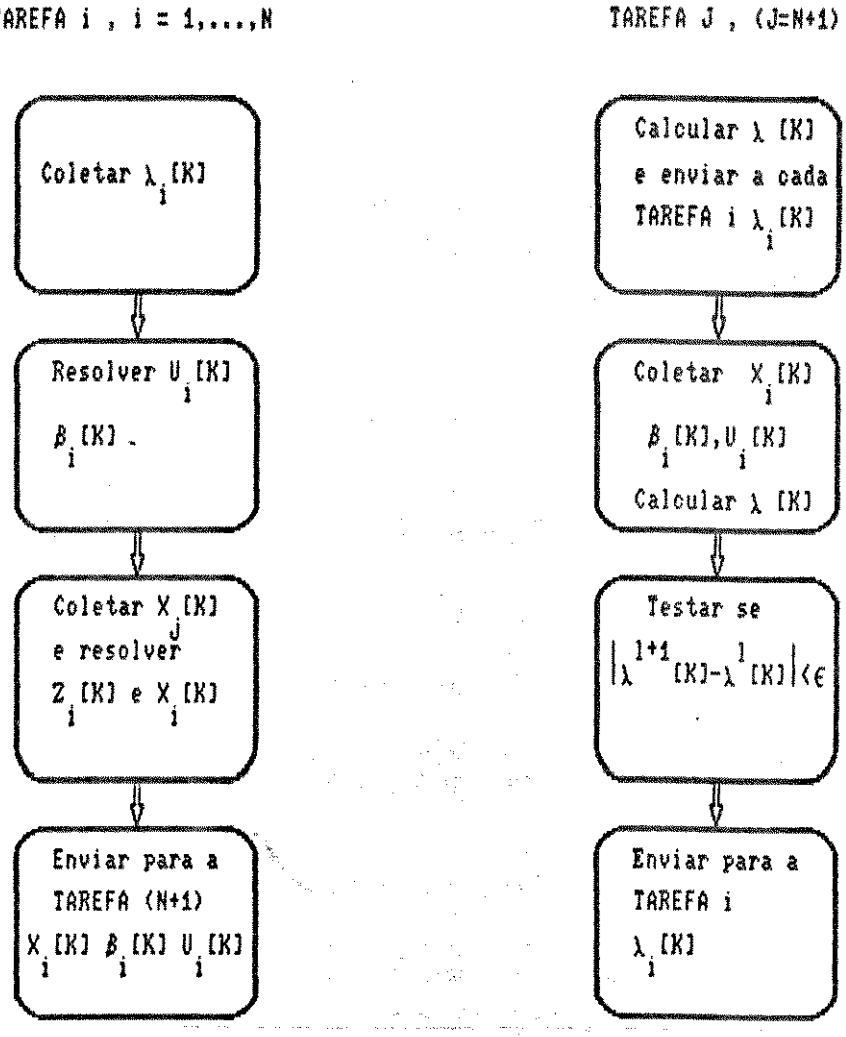
3.6.1 - Paralelização do algoritmo

Para o estudo da paralelização do algoritmo, definimos duas fases : a primeira fase compreende a detecção do paralelismo natural resultante na estrutura hierárquica em dois níveis de maneira que o algoritmo possa ser particionado em (N+1) tarefas onde cada TAREFA i (i=1,...,N) representa um subproblema de otimização do primeiro nível e a TAREFA j,(j=N+1) representa o coordenador; a segunda fase é dirigida para a análise das relações de dependência entre cada TAREFA i,(i=1,...,N), devido as restrições de interconexão (30) que são sempre satisfeitas durante o procedimento iterativo, e entre cada TAREFA i e a TAREFA j, uma vez que os multiplicadores de Lagrange, as variávelis de interconxão, de controle e de estado dependem das variáveis de coestado. Objetivamos com esta segunda fase aumentar o grau de paralelismo introduzindo uma ligeira modicação na metodologia apresentada com base no método predição de coestado [15].

3.6.1.1 - Primeira Fase

1.1 - Partição do Algoritmo

Considerando a disponiblidade de (N+1) unidades de processamento idênticas,especificamos as seguintes tarefas :



Do critério de partição adotado, podemos verificar:-

1) As tarefas relativas aos subproblemas apresentam o mesmo número de equações ;

2) É possível ter-se um número arbitrário de subproblemas de dimensões variadas ;

3) As tarefas relativas aos subproblemas de otimização são síncronas, ou seja, a execução dessas tarefas envolve interação em cada iteração.

4) A presença de termos não lineares em :

$$x_i[k+1] = f_i(x_i[k]) + B_i u_i[k] + z_i[k]$$

pode trazer desbalanceamento no processamento paralelo das tarefas.

Considerando que todos os subsistemas são fortemente interconectados :

$$z_i[k] = \phi_i(x_j[k]) \qquad (i=1,\ldots,N)$$

$$z_i[k] \neq 0$$

podemos representar a distribuição das tarefas e a comunicação e sincronização das tarefas segundo o critério de partição mencionado na fig. 3.14.

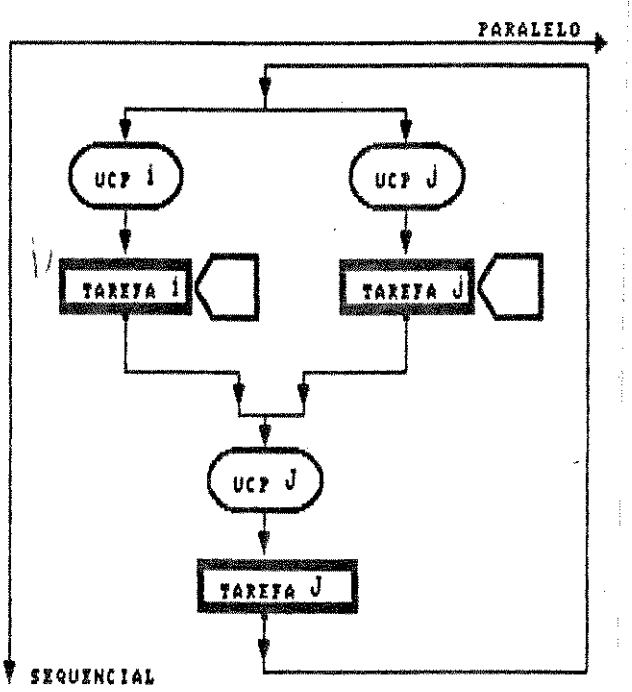


Figura 3.14 - Estrutura de paralelização

A frequência da comunicação (número de vezes que uma tarefa troca informações com outra(s) tarefa(s)) em cada iteração é dada pela seguinte representação matricial :

TAREFA	1	2	...	N	N+1
1	0	T		T	1
2	T	0		T	1
.			.		
.			.		
.			.		
N	T	T		0	1
N+1	1	1		1	0

Como podemos observar na fig.3.14 e na representação matricial, um ponto crítico a considerar na paralelização natural do algoritmo é a ocorrência na estrutura de decomposição do problema de otimização de um forte procedimento interativo em cada iteração, o que determina muitos pontos de sincronização e de comunicação e um baixo grau de paralelismo. Isto implica num ganho pouco significativo na velocidade de processamento global. Em adição, a execução sequencial deste algoritmo não tem apresentado um bom desempenho computacional com relação ao tempo de processamento quando comparado com a execução sequencial de outros algoritmos [10,14].

Neste contexto, decidimos por um novo critério de partição do algoritmo a partir da análise das relações de dependência e da consequente introdução de modificações na metodologia original.

3.6.1.2 - Segunda Fase

A análise das relações de dependência nos permitirá estabelecer as condições necessárias para a aplicação de um critério de partição que reduza os requerimentos de comunicação pelas tarefas, limitando ao mínimo o procedimento interativo entre elas. As informações obtidas

da análise das relações serão decisivas para a qualidade do processamento paralelo das tarefas nas arquiteturas de computação apresentadas.

Nesta seção, examinamos duas relações de dependência críticas para o processamento paralelo :

1) A primeira relação é referente ao processamento interativo entre os subproblemas de otimização em cada iteração e é dada pelas seguintes equações :

para $i=1,\ldots,N$ e $k=0,\ldots,T-1$

$$z_i[k] = \phi_i(x_j[k]) \qquad (j=1,\ldots,N \text{ e } j\neq i)$$

$$x_i[k+1] = f_i(x_i[k]) + B_i u_i + z_i$$

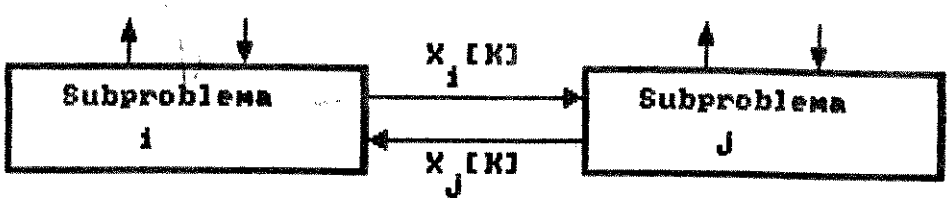


Figura 3.15 - Procedimento interativo

Em cada intervalo de k a (k+1), o vetor $z_i[k]$ e calculado antes do vetor $x_i[k+1]$. Como $z_i[k]$ depende da variável $x_j[k]$, o cálculo de $x_j[k]$ deve preceder o cálculo de $x_i[k+1]$. Neste caso, uma troca de informações entre o

subproblema i e cada subproblema j é necessária, originando uma dependência envolvendo o cálculo dessas variáveis. Esta interação entre os subproblemas torna praticamente sequencial o cálculo de $x_i[k]$ e portanto pouco significativo o cálculo com o processamento paralelo.

2) A segunda relação de dependência é oriunda da transferência de informações que ocorre em cada iteração entre os subproblemas e o coordenador. Da estrutura de otimização hierárquica obtida, o cálculo da trajetória de coestado $\lambda_i[k]$ no nível superior requer não somente a variável de estado $x_i[k]$ que afeta os dois primeiros termos da equação do nível 2, como também das variáveis de estado $x_j[k]$. Por outro lado, a mesma equação tem a função de prover a otimalidade das trajetórias de estado e de controle. Esta relação, como no caso da metodologia anterior, reduz a atividade de processamento das unidades de processamento quando a TAREFA j esta sendo executada. O tempo de espera para a execução de cada TAREFA i pode tornar-se ainda mais significativo quando da determinação do passo ρ.

Antes de definirmos um novo critério de partição, vamos introduzir uma modificação na metodologia da coordenação de coestado que é bastante atraente para o processamento paralelo e que permite validar o novo critério. Esta modificação é desenvolvida no método predição de coestado [10].

1.3 - Modificação no Método de Coordenação pelo Coestado

Consideremos o subsistema dinâmico descrito em (29) que pode ser reescrito da seguinte forma :

$$x_i[k+1] = g_i(x_i[k])x_i[k] + B_i u_i[k] + z_i[k] \; ; \; x_i[0] = x_{i0} \qquad (39)$$

$$z_i[k] = \phi_i(x_j[k]) \qquad j=1,\ldots,N \; e \; j \neq i \qquad (40)$$

A idéia principal do método predição de coestado para resolver o problema é gerar valores preditos de $\pi[k]$ e usá-los para $x[k]$ nas equações (39) e (40) para fixar $g_i(\,.\,)$ e $\phi_i(x_j[k])$. Então, podemos reescrever o i-ésimo subsistema como :

$$x_i[k+1] = g_i(\pi_i[k])x_i[k] + B_i u_i[k] + z_i[k] \; ; \; x_i[0] = x_{i0} \qquad (41)$$

$$z_i[k] = \sum_{j \neq i}^{N} \phi_{ij}(\pi_j[k]) \qquad (42)$$

$$\pi[k] = x[k] \qquad (43)$$

Para resolver o novo problema, o Hamiltoniano H definido como :

$$H = \sum_{i=1}^{N} H_i \qquad (44)$$

é aditivamente separável para $\pi[k]$ dado, com :

$$H_i(\,.\,) = (1/2)\left(x_i^t[k]\, Q_i\, x_i[k] + u_i^t[k]\, R_i u_i[k] \right)$$
$$+ \beta_i^t[k]\left(x_i[k] - \pi_i[k] \right)$$
$$+ \lambda_i^t[k+1]\left(g_i(\pi_i[k])x_i[k] + B_i\, u_i[k] + z_i[k] \right) \qquad (45)$$

onde $\beta_i[k] \in \mathbb{R}^{n_i}$ é o multiplicador de Lagrange associado

com as restrição extra expressa em (43).

O procedimento de solução compreende as condições necessárias de otimalidade :

$$W_i x_i[T] - \lambda_i[T] = 0 \; ; \tag{46}$$

$$\frac{\partial H_i}{\partial x_i[k]} - \lambda_i[k] = 0 \quad \therefore \lambda_i[k] = Q_i x_i[k] + g_i^t(\pi_i[k])\lambda_i[k+1]$$

$$+ \beta_i[k] \tag{47}$$

$$\frac{\partial H_i}{\partial u_i[k]} = 0 \quad \therefore \quad u_i[k] = - R_i^{-1} B_i^t \lambda_i[k+1] \tag{48}$$

$$\frac{\partial H}{\partial \pi_i[k]} = 0 \quad \therefore \quad \beta_i[k] = \left\{ \frac{\partial g_i(\pi_i[k])}{\partial \pi_i[k]} x_i[k] \right\}^t \lambda_i[k+1]$$

$$+ \sum_{j \neq i}^{N} \left\{ \frac{\partial \phi_{ji}(\pi_i[k])}{\partial \pi_i[k]} \right\}^t \lambda_j[k+1]$$

(49)

$$\frac{\partial H_i}{\partial \beta_i[k]} = 0 \quad \therefore \quad \pi_i[k] = x_i[k] \tag{50}$$

$$\frac{\partial H_i}{\partial \lambda_i[k+1]} = 0 \quad \therefore x_i[k+1] = g_i(\pi_i[k])x_i[k] + B_i u_i[k] + z_i[k]; \tag{51a}$$

$$x_i[o] = x_{io} \tag{51b}$$

Substituindo a equação (50) em (51) e as equações (49-50) em (47) obtem-se :

$$x_i[k+1] = g_i(x_i[k])x_i[k] + B_i u_i[k] + z_i[k] \qquad (52)$$

$$\lambda_i[k] = Q_i x_i[k] + g_i^t(x_i[k])\lambda_i[k+1]$$

$$+ \left\{ \frac{\partial\, g_i(x_i[k])}{\partial\, x_i[k]}\, x_i[k] \right\}^t \lambda_i[k+1]$$

$$+ \sum_{j \neq i}^{N} \left\{ \frac{\partial\, \phi_{ji}^t(x_i[k])}{\partial\, x_i[k]} \right\} \lambda_j[k+1] \qquad (53)$$

Consequentemente, as condições necessárias de otimalidade se reduzem as equações (48), (50), (52) e (53).

Vamos considerar as seguintes observações que identificam a filosofia principal para a abordagem da estrutura em dois níveis :

1) Uma vez que os parâmetros de interação $\pi[k]$ e vetores de coestado $\lambda_i[k]$ correspondentes são utilizados pelos vetores de estado locais (equação (52)), o cálculo desses vetores é realizado independentemente ;

2) Para calcular $\lambda_i[k]$, não somente a variável de estado $x_i[k]$ é requerida como também as outras variáveis de estado são necessárias ;

3) Para a determinação dos parâmetros de interação $\pi_i[k](i=1,\ldots,N)$, os valores das diferentes variáveis de estado devem ser conhecidos. Não há uma relação explicita

dos parametros de interação com os vetores de coestado.

4) Uma vez que, das equações relativas as condições necessárias de otimalidade, apenas o termo $z_i[k]$ na equação (51) é, explicitamente, função de $\pi_j[k]$, a predição dos parâmetros de interação implica na predição dos vetores de interação.

Com base nestas observações obtivemos uma estrutura de otimização descentralizada.

A metodologia predição de coestado e a estrutura hierárquica em dois níveis é descrita [10].

1.4 - Descentralização e distribuição

A partir da análise das relações de dependência, concluimos pela necessidade de um novo critério de partição que será baseado na descentralização da estrutura de otimização e na distribuição da coordenação, objetivando aumentar o grau de paralelismo.

O critério adotado compreende :

(i) descentralização : consiste da decomposição do sistema dinâmico :

$$S : x[k+1] = f(x[k]) + B\,u[k]$$

em N subsistemas

$$S_i : x_i[k+1] = f_i(x_i[k]) + B_i u[k] + z_i[k]$$

$(i=1,\ldots,N)$

mutuamente disjuntos, tal que :

a) $S = \bigcup_{i=1}^{N} S_i$.

b) Para qualquer i,j com i,j = 1,...,N e $j \neq i$, S_i e S_j não tenham elementos comuns.

c) S_i é associado com critério de desempenho (5) definindo o i-ésimo subproblema.

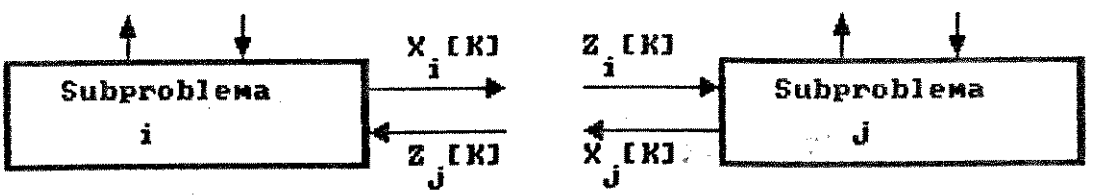


Figura 3.16 - Estrutura descentralizada

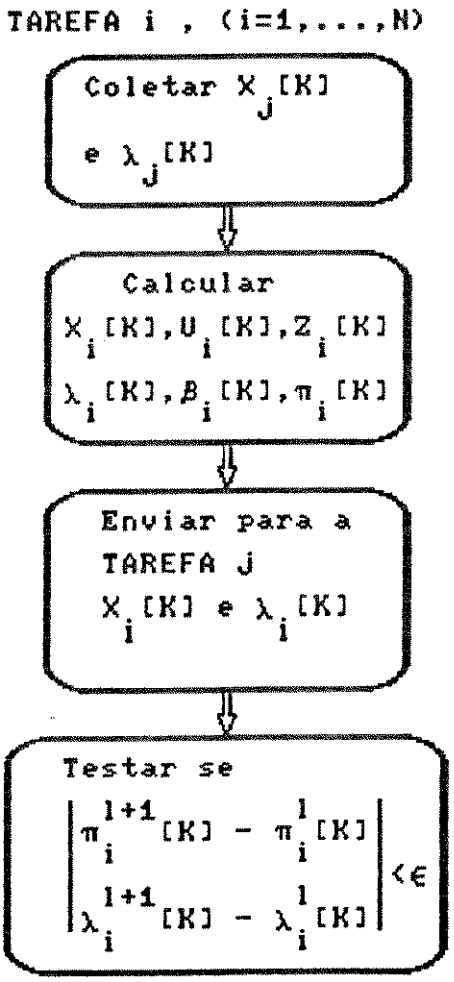
(ii) especificação de subtarefas de coordenação : Consiste na distribuição da tarefa de coordenação C em N subtarefas C_i (i=1,...,N) tal que :

a) $C = \bigcup_{i=1}^{N} C_i$.

b) Para qualquer i,j com $i,j = 1,\ldots,N$ e $j \neq i$ C_i e C_j tenham o vetor $x[k]$ comum.

Para a aplicação desse critério de partição, é necessário que no cálculo do i-ésimo subproblema em cada iteração, os vetores de coestado $\lambda_i[k]$ e de interação $z_i[k]$ sejam conhecidos antecipadamente. Fazendo a predição de $z_i[k]$, a solução do i-ésimo subproblema em cada iteração não necessita explicitamente dos valores de $x_j[k]$ para iniciar o cálculo $x_i[k+1]$. Portanto, a computação dos vetores de estado $x_i[k+1]$ é realizada independentemente (descentralizada) das outras variáveis de estado. Os valores preditos de $z_i[k]$ podem ser estimados com base nos valores de $x_j[k]$ calculados na iteração anterior através da equação (29). Então, o vetor de coordenação é formado pelo parâmetro de interação e pelo vetor de coestado obtido na equação (52) a partir das variáveis de estado dos subproblemas. Como não há uma relação explicita entre o cálculo numérico dos vetores de coestado e dos parâmetros de interação, o vetor de coordenação pode ser particionado em N subvetores. O i-ésimo subvetor de coordenação será composto por $\lambda_i[k]$ e $\pi_i[k]$ de tal forma que possamos distribuir os N subvetores de coordenação pelos N subproblemas de otimização.

A partir do critério de partição adotado, definimos, para processamento paralelo, as tarefas :



Em síntese,podemos representar esquematicamente as etapas de partição do algoritmo conforme a fig.3.17.

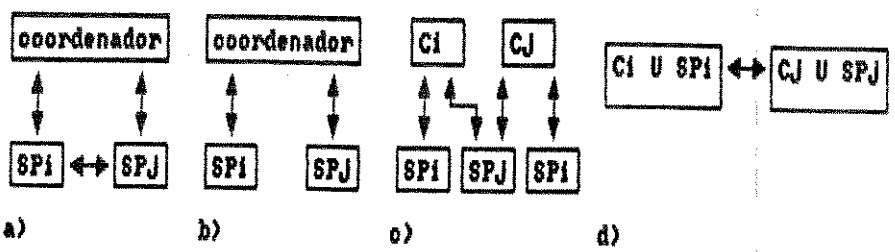


Figura 3.17 - Procedimentode paralelização

Na fig (3.17) representamos a estrutura de paralelização

do algoritmo com :

(i) comunicação síncrona entre as TAREFA i e a TAREFA j, para i = 1,...,N e j = N + 1;

(ii) comunicação síncrona entre as TAREFA i.

A estrutura na fig.3.17b, resultante do critério de partição (1.4i) adotado exibe efetivamente a execução assíncrona das TAREFA i (i = 1,...,N). Esta estrutura tem apresentado um desempenho computacional superior ao da estrutura representada na fig.3.17a. Nas fig.3.17c e 3.17d, etapas de partição da tarefa de coordenação e paralelização final respectivamente, levamos em consideração a distribuição da tarefa de coordenação e a consequente eliminação da comunicação síncrona entre as TAREFA i e a SUBTAREFA i , obtendo a estrutura definitiva da fig.3.17d . Com esta última estrutura, conseguimos reduzir os requisitos de comunicação entre as tarefas paralelas,limitando ao mínimo o procedimento interativo entre elas.

1.5 - Distribuição das Tarefas

Seguindo o mesmo procedimento das implementações anteriores, para cada tarefa foi associada uma unidade de processamento.

1.6 - Comunicação e sincronização

A representação matricial da frequência de comunicação para esta nova paralelização é dada por :

TAREFA	1	2 ...	N
1	0	1	1
2	1	0	1
⋮		⋱	
N	1	1	0

A estrutura de implementação final pode então ser representada como na fig.3.18.

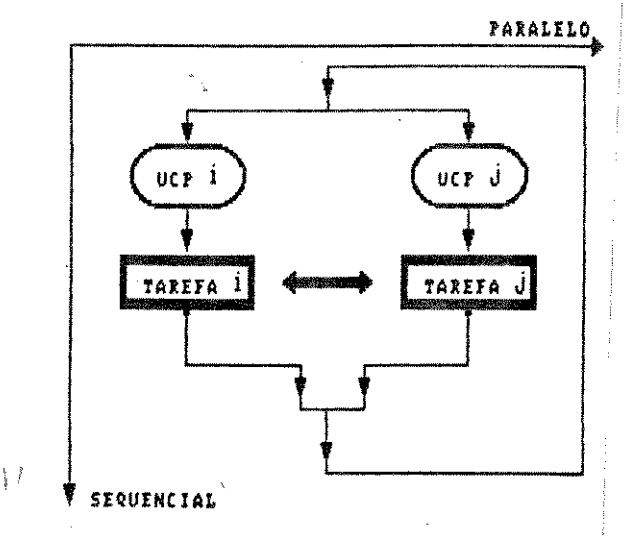


Figura 3.18 - Estrutura de paralelização

Da fig.3.18, pode-se verificar que há apenas um ponto de sincronização global.

Como podemos observar a sobrecarga resultante da comunicação e da sincronização entre as tarefas foi significativamente reduzida.

CAPÍTULO 4 - ANÁLISE DE DESEMPENHO COMPUTACIONAL

4.1 - Introdução

Neste capítulo, apresentamos os requerimentos computacionais que nos permitem avaliar o desempenho computacional dos algoritmos, considerando tanto as implementações paralelas propostas como as sequenciais.

Pela análise dos requerimentos de tempo de computação, de memória utilizada e de dados de comunicação, obtemos expressões para avaliação do desempenho computacional dos algoritmos sobre os sistemas de múltiplos processadores apresentados com relação aos problemas de partição, de distribuição e de comunicação tratados no capítulo 3.

4.2 - Requerimentos de Tempo de Computação

O tempo de computação gasto por cada tarefa durante o processamento paralelo compreende as seguintes quantidades :

1) Tempo de comunicação : resultante da leitura e escrita de dados na memória global ;

2) Tempo de espera : relativo a sincronização com outra(s) tarefa(s) no acesso aos dados compartilhados ;

3) Tempo de execução : referente ao tempo gasto na execução dos passos de cálculo da tarefa.

Nas aplicações que temos realizado sobre o Processador Preferencial, os dois primeiros tempos são poucos significativos pelas seguintes razões :

1) O barramento (PP-BAR) é de alta velocidade (10 Mbytes/segundo);

2) Reduzida taxa de comunicação;

3) Diminuição de conflito no acesso aos dados compartilhados resultante do emprego das estratégias já mencionadas.

Supondo, para o problema de otimização proposto e para as condições de inicialização dadas, que o tempo de execução do cálculo de um subproblema é uma função não linear da dimensão do respectivo subsistema :

$$t_i = \xi(n_i) \qquad i=1,\ldots,N \ , \qquad (1)$$

podemos, então, analizar a implementação dos algoritmos na forma seqüencial e paralela. Esta análise, se extende para os algoritmos paralelizados nas duas etapas, ou seja, considerando a coordenação como uma tarefa a parte na primeira etapa e a distribuição da coordenação numa segunda etapa.

a) Implementação sequencial : consiste na implementação de um algoritmo sobre uma das unidades de processamento do sistema de processamemto paralelo utilizado. O tempo t_1 gasto no cálculo de todos os N subproblemas na primeira iteração é :

$$t_1 = \sum_{i=1}^{N} \xi(n_i) \qquad (2)$$

b) Implementação paralela (primeira etapa) : o tempo gasto para resolver os N subproblemas na primeira iteração, utilizando N unidades de processamento é :

$$t_1(N) = \max \langle \xi(n_1), \xi(n_2), \ldots, \xi(n_N) \rangle \qquad (3)$$

c) Implementação paralela (segunda etapa) : o tempo gasto para calcular os N subproblemas na primeira iteração, utilizando N unidades de processamento é :

$$t_2(N) = \max \langle \xi(n_1), \xi(n_2), \ldots, \xi(n_N) \rangle \qquad (4)$$

O tempo total de execução do algoritmo , respectivamente, para cada implementação mencionada é :

$$t_1 = \sum_{l=0}^{L-1} \left\{ \sum_{i=1}^{N} \xi(n_i) + t_c \right\} \gamma_l \qquad (5)$$

$$t_1(N+1) = \sum_{l=0}^{L-1} \left\{ \max \langle \xi(n_1), \ldots, \xi(n_N) \rangle + t_c \right\} \gamma_l \qquad (6)$$

$$t_2(N) = \sum_{l=0}^{L-1} \left\{ \max \langle \xi(n_1), \ldots, \xi(n_N) \rangle + \frac{t_c}{N} \right\} \gamma_l \qquad (7)$$

onde : t_c é o tempo necessário para a execução da tarefa de coordenação; (t_c / N) é o tempo necessário para a execução de cada subtarefa de coordenação ; γ_l é uma variável que

depende do tempo de solução das trajetórias em cada iteração em relação a primeira iteração.

Para uma medida aproximada do desempenho que pode ser conseguido com a execução paralela dos algoritmos analizados definimos o ganho de velocidade de processamento do algoritmo como :

$$g(\eta) = \frac{\text{tempo do algoritmo paralelo executado em } P = 1}{\text{tempo do algoritmo paralelo executado em } P = \eta} \leq \eta \qquad (8)$$

onde P é o número de processadores. Idealmente, desejariamos que $g(\eta)$ fosse igual a η, mas na realidade há sempre algum tempo gasto com a comunicação e a sincronização.

A razão dada por :

$$E(\eta) = \frac{g(\eta)}{\eta} \qquad (9)$$

pode ser utilizada como a medida de eficiência de um algoritmo paralelo sendo executado sobre $P = \eta$ unidades de processamento simultâneamente.

Supondo que todos os susbsistemas têm a mesma dimensão ($n_1 = n_2 = \ldots n_N = \mu$) podemos considerar :

$$t_1 = \sum_{l=0}^{L-1} \left\{ N\, \xi(\mu) + t_c \right\} \gamma_l \qquad (10)$$

$$t_1(N+1) = \sum_{l=0}^{L-1} \left\{ \xi(\mu) + t_c \right\} \gamma_l \qquad (11)$$

$$t_2(N) = \sum_{l=0}^{L-1} \left\{ \xi(\mu) + \frac{t_c}{N} \right\} \gamma_l \qquad (12)$$

Definindo $t_p = N\,\xi(\mu)$ obtemos os seguintes ganhos :

$$g(N+1) = \frac{t_1}{t_1(N+1)} \quad \therefore \quad g(N+1) \cong \frac{t_p + t_c}{\frac{t_p}{N} + t_c}$$

$$g(N+1) \cong \frac{N}{1 + (N-1)\nu} \quad \text{onde} \quad \nu = \frac{t_c}{t_p + t_c} \qquad (13)$$

$$g(N) = \frac{t_1}{t_2(N)} \quad \therefore \quad g(N) \cong \frac{t_p \quad t_c}{\frac{t_p}{N} + \frac{t_c}{N}} \qquad (14)$$

$$g(N) \cong N \qquad (15)$$

Desta análise , podemos observar que :

a) O tempo total de execução depende tanto do sistema de computação utilizado como do grau de parlelismo obtido ;

b) O desempenho da execução paralela é degradada se o tempo de execução da tarefa de coordenação é muito significativo ;

c) A eficiência $E(\eta)$ para uma execução particular do algoritmo decesce com o aumento do número η de

unidades de processamento, uma vez que o tempo gasto com comunicação e sincronização aumenta com o número η de unidades de processamento ;

d) As relações :

$$\frac{t_1}{t_1(N)} \quad \text{e} \quad \frac{t_1}{t_2(N)}$$

nos permitem determinar a qualidade do balanceamento que pode ser atingido pelas tarefas paralelas, servindo como orientação para a estratégia de realocação do processamento das η unidades quando de uma implementação particular de um algoritmo.

4.3 - Requerimento de Memória

O emprego de memória pelos algoritmos nas diferentes implementações realizadas pode ser comparadas através das trajetórias armazenadas.

4.3.1 - Algoritmo predição de interação

Para este algoritmo as trajetórias armazenadas são :

$\sum_{i=1}^{N} n_i$ trajetórias de x [k]

$\sum_{i=1}^{N} n_i$ trajetórias de z [k]

$\sum_{i=1}^{N} n_i$ trajetórias de s [k]

$\sum_{i=1}^{N} n_i$ trajetórias de λ [k]

$\sum_{i=1}^{N} m_i$ trajetórias de u [k]

$\sum_{i=1}^{N} n_i (n_i + 1)/2$ trajetórias de P [k]

Para a implementação sequencial a quantidade de memória local necessária é dada por :

$$M_s = \Gamma_s + \sum_{i=1}^{N} (4n_i + m_i + (n_i + 1)n_i /2) \; T \; B \qquad (16)$$

onde Γ_s é a quantidade de memória local para armazenar o programa relativo ao algoritmo; B é o número de bytes necessário para armazenar uma variável.

Para a primeira implementação em paralelo, a quantidade necessária de memória local para armazenar cada TAREFA i é dada por :

$$M_i = \Gamma_i + (4n_i + m_i + (n_i + 1)n_i /2) \; T \; B \qquad (17)$$

onde Γ é a quantidade de memória local necessária para armazenar o programa relativo a cada TAREFA i. Para a TAREFA j

$$M_j = \Gamma_j + \sum_{i=1}^{N} (2n_i) \; T \; B \qquad (18)$$

A quantidade total de memória local utilizada pela primeira implementação em paralelo é :

$$M_d = M_j + \sum_{i=1}^{N} \Gamma_i + \sum_{i=1}^{N} (4n_i + m_i + (n_i + 1)n_i/2)\ T\ B \qquad (19)$$

$$M_d = M_s - \Gamma_s + \Gamma_j + \sum_{i=1}^{N} \Gamma_i + \sum_{i=1}^{N} (2n_i) \qquad (20)$$

Devido a sobrecarga adicional oriunda da partição do algoritmo temos :

$$\Gamma_j + \sum_{i=1}^{N} \Gamma_i > \Gamma_s \qquad (21)$$

Consequentemente,

$$M_d > M_s + \sum_{i=1}^{N} (2n_i)\ T\ B \qquad (22)$$

Para a segunda implementação em paralelo, a memória local necessária para cada TAREFA i é dada por :

$$M_i = \Gamma_i + (4n_i + m_i + n_i(n_i + 1)/2)\ T\ B \qquad (23)$$

A quantidade total de memória local utilizada para armazenar as trajetórias é :

$$M_P = \sum_{i=1}^{N} \Gamma_i + \sum_{i=1}^{N} (4n_i + m_i + n_i(n_i + 1)/2)\ T\ B \qquad (24)$$

$$M_P = M_S - \Gamma_S + \sum_{i=1}^{N} \Gamma_i \qquad (25)$$

Pelas mesmas razões citadas anteriormente temos :

$$\sum_{i=1}^{N} \Gamma_i > \Gamma_S \qquad (26)$$

Portanto, podemos concluir :

$$M_P > M_S \qquad (27)$$

4.3.2 - Algoritmo de coordenação de coestado

As trajetórias armazenadas são :

$\sum_{i=1}^{N} n_i$ trajetórias de x [k]

$\sum_{i=1}^{N} m_i$ trajetórias de u [k]

$\sum_{i=1}^{N} n_i$ trajetórias de λ [k]

A quantidade total de memória utilizada para armazenar a implementação sequencial é dada por :

$$M_S = \Gamma_S + \sum_{i=1}^{N} (2n_i + m_i)\ T\ B \qquad (28)$$

Com a modificação introduzida, as trajetórias armazenadas

são :

$\sum_{i=1}^{N} n_i$ trajetórias de x [k]

$\sum_{i=1}^{N} m_i$ trajetórias de u [k]

$\sum_{i=1}^{N} n_i$ trajetórias de λ [k]

$\sum_{i=1}^{N} n_i$ trajetórias de π [k]

Com esta modificação a quantidade de memória local necessária para armazenar a TAREFA i é dada por :

$$M_i = \Gamma_i + (3n_i + m_i)\ T\ B \qquad (29)$$

A quantidade total de memória armazenada é :

$$M_P = \sum_{i=1}^{N} \Gamma_i + \sum_{i=1}^{N} (3n_i + m_i)\ T\ B \qquad (30)$$

Por analogia com o algoritmo anterior podemos concluir :

$$M_P > M_S \qquad (31)$$

onde M_S é a memória total para armazenar o algoritmo paralelo para execução sequencial.

Da análise do requerimento de memória pelos algoritmos citados, podemos verificar os seguintes pontos :

a) Há uma sobrecarga adicional de memória associada com a distribuição de tarefas pelas unidades de processamento ;

b) A quantidade de memória local utilizada em cada iteração permanece a mesma ;

c) A quantidade de memória utilizada por cada unidade de processamento na solução de uma tarefa é bem menor do que a utilizada na solução do algoritmo numa única unidade, apesar de que a quantidade total de memória utilizada ser maior no sistema de múltiplas unidades de processamento.

4.4 - Dados de Comunicação

No sistema de múltiplos processadores com memória compartilhada, a comunicação entre as tarefas se realiza através da troca de dados na memória global. Nesta forma de comunicação, há necessidade de se distinguir quais dados sao privativos de cada unidade de processamento e quais são conhecidos de todas as unidades de processamento. Sob esse enfoque, determinamos a quantidade total de dados necessários para a comunicação entre as unidades de processamento. Os dados utlizados para sincronizar o acesso as variáveis compartilhadas consistem de alguns bytes e portanto não serão considerados.

4.4.1 Algoritmo predição de interação

Para a primeira implemantação, as trajetórias armazenadas na memória global são :

$\sum_{i=1}^{N} n_i$ trajetórias de x [k]

$\sum_{i=1}^{N} n_i$ trajetórias de λ [k]

$\sum_{i=1}^{N} n_i$ trajetórias de z [k]

A quantidade total de dados compartilhados é :

$$D = L\ T\ B \sum_{i=1}^{N} (3n_i) \qquad (32)$$

Para a segunda implementação, as trajetórias armazenadas são :

$\sum_{i=1}^{N} n_i$ trajetórias de x [k]

$\sum_{i=1}^{N} n_i$ trajetórias de λ [k]

A quantidade D total de dados armazenados é :

$$D = L\ T\ B \sum_{i=1}^{N} (2n_i) \qquad (33)$$

4.4.2 - Algoritmo de coordenação de coestado

Com a modificação introduzida, as trajetórias armazenadas na memória global são :

$$\sum_{i=1}^{N} n_i \text{ trajetórias de } x\,[k]$$

$$\sum_{i=1}^{N} n_i \text{ trajetórias de } \lambda\,[k]$$

A quantidade total de dados compartilhados é :

$$D = L\ T\ B \sum_{i=1}^{N} (2n_i) \qquad (34)$$

Os critérios de partição que levaram em consideração a distribuição da tarefa de coordenação pelas mesmas unidades de processamento das tarefas relativas aos subproblemas diminuiram a quantidade de dados de comunicação.

4.5 - Aspectos computacionais

Nesta seção, apresentamos três exemplos numéricos que nos permitiram fazer algumas comparações e avaliar o grau de desempenho computacional das paralelizações obtidas e implementadas. As análises de desempenho medidas foram realizadas sobre o Processador Preferencial, cuja configuração utilizada dispunha de somente quatro unidades de processamento. Realizamos uma implementação sobre a arquitetura de troca de mensagens, utilizando dois computadores Digirede - 8000. Uma vez que esta arquitetura apresenta ainda deficiência com relação aos mecanismos de comunicação e de sincronização tanto a nível de hardware como de software, apresentamos simplesmente a descrição dos programas paralelos e a forma de comunicação e sincronização empregados.

4.5.1 - Primeiro Problema

Considere o seguinte sistema dinâmico :

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \\ x_5[k+1] \\ x_6[k+1] \end{bmatrix} = \begin{bmatrix} 0,5 & 0,18 & 0,22 & 0,3 & 0,018 & 0,03 \\ 0,2 & 0,51 & 0,10 & 0,06 & 0,3 & 0,2 \\ 0,22 & 0,09 & -0,25 & 0,25 & 0,012 & 0,3 \\ 0,3 & 0,05 & 0,25 & -0,25 & -0,13 & 0,06 \\ 0,09 & 0,33 & 0,012 & 0,015 & -0,6 & -0,35 \\ 0,039 & 0,31 & 0,30 & 0,06 & 0,35 & -0,35 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \end{bmatrix}$$

$$\begin{bmatrix} 1,0 & 0,0 & 0,0 \\ 1,0 & 0,0 & 0,0 \\ 0,0 & 1,0 & 0,0 \\ 0,0 & 1,0 & 0,0 \\ 0,0 & 0,0 & -1,0 \\ 0,0 & 0,0 & 1,0 \end{bmatrix} \begin{bmatrix} u_1[k] \\ u_2[k] \\ u_3[k] \end{bmatrix}$$

$x[0] = [1, -1, 1, -1, 1, -1]$

Particionamos o sistema em três subsistemas :

subsistemas	variáveis	
	estado	controle
subsistema 1	$x_1[k], x_2[k]$	$u_1[k]$
subsistema 2	$x_3[k], x_4[k]$	$u_2[k]$
subsistema 3	$x_5[k], x_6[k]$	$u_3[k]$

As matrizes de ponderação são :

$$Q_i = \begin{bmatrix} 0,1 & 0,0 \\ 0,1 & 0,1 \end{bmatrix} \quad ; \quad R_i = 100$$

4.5.2 - Segundo problema

Sistema dinâmico :

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \end{bmatrix} = \begin{bmatrix} 0,1 & 0,1 \\ 0,2 & (0,1-0,1\ x_2[k]) \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \end{bmatrix}$$

$$+ \begin{bmatrix} 0,1 & 0,0 \\ 0,0 & 0,1 \end{bmatrix} \begin{bmatrix} u_1[k] \\ u_2[k] \end{bmatrix}$$

$x_1[0] = 10$

$x_2[0] = 4,5$

Critério de desempenho :

$$J = \sum_{k=0}^{50} \frac{0,1}{2} \left(x_1^2[k] + x_2^2[k] + 2\, u_1^2[k] + 0,1\, u_2^2[k] \right)$$

Partição do sistema :

Subsistema 1 : $x_1[k]$ e $u_1[k]$

Subsistema 2 : $x_2[k]$ e $u_2[k]$

4.5.3 - Terceiro problema

A fig.4.2 mostra o sistema de malha aberta que consiste de uma máquina síncrona conectada a um barramento infinito através de um transformador e de uma linha de transmissão. A voltagem de excitação V_{ex} é utilizada como uma variável de controle. A referência de velocidade do mecanismo regulador de velocidade é também uma segunda variável de controle.

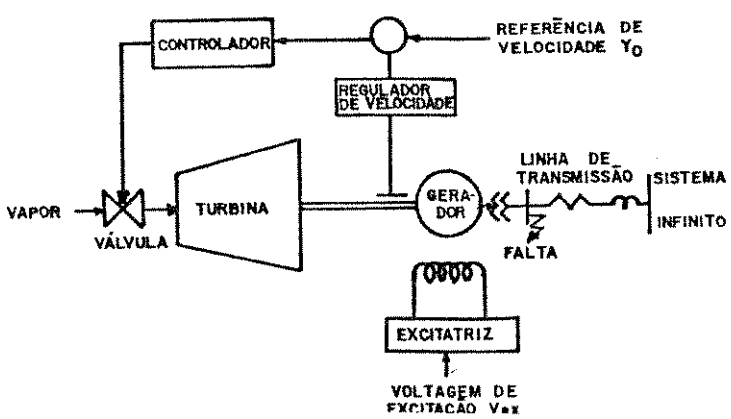


Figura 4.2 - Sistema turbo gerador

As equações dinâmicas para este modelo são estudadas e analizadas em [18,19]. As equações do sistema discretizado com intervalo de discretização de 0,05 segundos podem ser escritas como :

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \\ x_5[k+1] \\ x_6[k+1] \end{bmatrix} = \begin{bmatrix} 1,0 & 0,0 & 0,0 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,891 & 0,0 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,949 & 0,0 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,0 & 0,75 & 0,0 & 0,0 \\ 0,0 & 0,0 & 0,0 & 0,0 & -0,6 & 0,0 \\ 0,0 & 0,0 & 0,0 & 0,0 & 0,0 & 0,975 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \end{bmatrix}$$

$$+ \begin{bmatrix} 0,0 & 0,0 \\ 0,0 & 0,0 \\ 0,0 & 1,0 \\ 0,472 & 1,0 \\ 0,0 & 0,0 \\ 0,0 & 0,025 \end{bmatrix} \begin{bmatrix} u_1[k] \\ u_2[k] \end{bmatrix}$$

$$+ \begin{bmatrix} 0,5\ x_2[k] \\ 1,47\ x_5[k] - 0,699\ x_3[k]\ \text{sen}\ x_1[k] + 1,389\ \text{sen}\ 2x_1[k] \\ 0,05\ x_6[k] + 0,2024\ \cos x_1[k] \\ 0,0509\ x_2[k] \\ 0,102\ x_4[k] \\ 0,0 \end{bmatrix}$$

$x_1[0] = 0,7105$; $x_2[0] = 0,0$; $x_3[0] = 5,604$;

$x_4[0] = 0,8$; $x_5[0] = 0,8$; $x_6[0] = 2,645$;

O problema de controle ótimo é determinar os valores de u (V_{ex}) e de u (Y_o) para obter a variação desejada no ângulo do rotor x (δ) segundo o critério de desempenho a ser minimizado :

$$J = \frac{(0,05)}{2} \sum_{k=0}^{40} \left(2\,(x_1[k] - x_{d1})^2 + 2\,(x_2[k] - x_{d2})^2 + 0,5\,(u_1[k] - u_{d1})^2 + 0,5\,(u_2[k] - u_{d2})^2 \right)$$

O sistema foi particionado em dois subsistemas :

Subsistema 1 : $x_1[k]$, $x_3[k]$, $x_6[k]$

$u_2[k]$

Subsistema 2 : $x_2[k]$, $x_4[k]$, $x_5[k]$

$u_1[k]$

4.6 - Descrição dos programas implementados

Os algoritmos paralelizados e implementados sobre o Processador Preferencial foram escritos na linguagem de programação Pascal e executados sobre o sistema operacional MS-DOS. Para o sistema de múltiplos microcomputadores, utilizando o micro DIGIREDE 8000 os programas foram escritos nas linguagens Pascal e C, e executados sobre o sistema operacional UNIX. Nesta seção, ressaltamos os aspectos dos mecanismos de comunicação e de sincronização. O procedimento computacional da parte de cálculo dos algoritmos não são descritos.

4.6.1 - Algoritmo predição de interação

1 - Primeira implementação

Nesta implementação, descreveremos apenas o programa referente a TAREFA j,(j=N+1) e o programa referente uma das TAREFA i,(i=1,...N) para o primeiro problema.

```
program COORDENADOR;  (* programa relativo a TAREFA j *)
type
 VETOR = array [1..Ni] of real;(* Ni é a dimensão dos
                                   dos subsistemas *)
var
:
:
 Z : array [1..N,0..T] of VETOR; (* Vetor de interação *)
 BETA : array [1..N,0..T] of VETOR;  (*Multiplicador de La-
                                       grange *)
 X : array [1..N,0..T] of VETOR; (* Vetor de estado *)
 (* Variáveis de sincronização *)
 FLAG1 [origin 16#9000:16#0000]:integer;
 FLAG2 [origin 16#9010:16#0000]:integer;
 FLAG3 [origin 16#9020:16#0000]:integer;
 (* Variáveis compartilhadas *)
 X1 [origin 16#9040:16#0000]:array [0..9] of VETOR;
 X2 [origin 16#9100:16#0000]:array [0..9] of VETOR;
 X3 [origin 16#9200:16#0000]:array [0..9] of VETOR;
 BETA1 [origin 16#9300:16#0000]:array [0..9] of VETOR;
 BETA2 [origin 16#9400:16#0000]:array [0..9] of VETOR;
 BETA3 [origin 16#9500:16#0000]:array [0..9] of VETOR;
 Z1 [origin 16#9600:16#0000]:array [0..9] of VETOR;
 Z2 [origin 16#9700:16#0000]:array [0..9] of VETOR;
 Z3 [origin 16#9800:16#0000]:array [0..9] of VETOR;

procedure INTERCON;
begin
  calcular o vetor de interação;
  testar a convergência;
end;

(*        PROGRAMA PRINCIPAL          *)
begin
  inicialização;
  FLAG1:=1;
```

```
    FLAG2:=1;
    FLAG3:=1;
    repeat
      if ((FLAG1=0) and (FLAG2=0) and (FLAG3=0))
      then
      begin
        for K:=0 to T
        do
        begin
         BETA[1,K]:=BETA1[1,K];
         X[1,K]:=X1[1,K];
         BETA[2,K]:=BETA2[1,K];
         X[2,K]:=X2[1,K];
         BETA[3,K]:=BETA3[1,K];
         X[3,K]:=X3[1,K];
        end;
        INTERCON;
        for K:=0 to T
        do
        begin
          Z1[1,K]:=Z[1,K];
          Z2[1,K]:=Z[2,K];
          Z3[1,K]:=Z[3,K];
        end;
        FLAG1 :=1;
        FLAG2 :=1;
        FLAG3 :=1;
    until (...)
end.

program SUBSISTEMA1; (* Programa referente TAREFA i,(i=1) *)
type
 VETOR = array [1..Ni] of real;
var
```

```
 Z : array [1..N,0..T] of VETOR; (* Vetor de interconexão *)
 BETA : array [1..N,0..T] of VETOR;
 X : array [1..N,0..T] of VETOR; (* Vetor de estado *)
 (* variável de sincronização *)
 FLAG1 [origin 16#9000:16#0000]:integer;
 (* Variáveis compartilhadas *)
 BETA1 [origin 16#9300:16#0000]:array [0..T] of VETOR;
 BETA2 [origin 16#9400:16#0000]:array [0..T] of VETOR;
 BETA3 [origin 16#9500:16#0000]:array [0..T] of VETOR;
 X1 [origin 16#9040:16#0000]:array [0..T] of VETOR;
 Z1 [origin 16#9600:16#0000]:array [0..T] of VETOR;

procedure RICCATI;
begin
 resolver a equação matricial de Riccati;
end;

procedure COMPENSADOR;
begin
 resolver a equação do compensador;
end;

procedure ESTADO;
begin
 calcular o vetor de estado e de controle;
 for K:=0 to T
   do
    X1[1,K]:=X[1,K];
end;

procedure COESTADO;
begin
 calcular o vetor de coestado e o multiplicador de Lagran-
 ge;
```

```
    for K:=0 to T
    do
     BETA1[1,K]:=BETA[1,K];
end;

(* PROGRAMA  PRINCIPAL *)

begin
 inicialização;
 RICCATI;
 repeat
   if FLAG1=1
   then
   begin
     for K:=0 to T
     do
     begin
        BETA[2,K]:=BETA2[1,K];
        BETA[3,K]:=BETA3[1,K];
        Z[I,K]:=Z1[1,K];
     end;
   COMPENSADOR;
   ESTADO;
   COESTADO;
   FLAG1:=0
  until (...)
end.
```

2 - Segunda implementação

Nesta implementação, apresentamos a descrição de dois programas referentes as tarefas obtidas na segunda etapa de paralelização para o primeiro problema.

```
(* Programa referente a TAREFA i,(i=1) *)
program SUBSISTEMA1;
type
 VETOR = array [1..Ni] of real;
var
:
 BETA : array [1..N,0..T] of VETOR;
 X : array [1..N,0..T] of VETOR; (* Vetor de estado *)
 (* variável de sincronização *)
 FLAG [origin 16#9000:16#0000]:array [1..N] of integer;
 SEMAF [origin 16#A000:16#0000]:array [1..N] of integer;
 (* Variáveis compartilhadas *)
 BETA1 [origin 16#9300:16#0000]:array [0..T] of VETOR;
 BETA2 [origin 16#9400:16#0000]:array [0..T] of VETOR;
 BETA3 [origin 16#9500:16#0000]:array [0..T] of VETOR;
 X1 [origin 16#9040:16#0000]:array [0..T] of VETOR;
 X2 [origin 16#9100:16#0000]:array [0..T] of VETOR;
 X3 [origin 16#9200:16#0000]:array [0..T] of VETOR;

procedure RICCATI;
begin
 resolver a equação matricial de Riccati;
end;

procedure COMPENSADOR;
begin
 resolver a equação do compensador;
end;

procedure ESTADO;
begin
 calcular o vetor de estado e controle;
 for K:=0 to T
   do
    X1[1,K]:=X[1,K];
```

```
end;

procedure COESTADO;
begin
 calcular o vetor de coestado e o multiplicador de Lagran-
 ge;
for K:=0 to T
do
  BETA1[1,K]:=BETA[1,K];
end;

(* PROGRAMA  PRINCIPAL *)

begin
 inicialização;
 RICCATI;
 repeat
   COMPENSADOR;
   ESTADO;
   COESTADO;
   FLAG[1]:=0;
   while (FLAG[3]=0 or FLAG[2]=0)
   do
   begin end;
   for K:=0 to T
   do
   begin
    X[2,K]:=X2[1,K];
    X[3,K]:=X3[1,K];
   end;
   INTERCON;
   for K:=0 to T
   do
   begin
    BETA[2,K]:=BETA2[1,K];
```

```
     BETA[3,K]:=BETA3[1,K];
    end;
    SEMAF[1]:=1;
    while (SEMAF[1]+SEMAF[2]+SEMAF[3]< 3)
    do
    begin end;
    FLAG[1]:=0;
    while (FLAG[3]=1 or FLAG[2]=1)
    do
    begin end;
    SEMAF[1]:=0;
 until (...)
end.

(* Programa referente a TAREFA i,(i=2) *)
program SUBSISTEMA2;
type
 VETOR = array [1..Ni] of real;
var
:
 BETA : array [1..N,0..T] of VETOR;
 X : array [1..N,0..T] of VETOR;
 (* variável de sincronização *)
 FLAG [origin 16#9000:16#0000]:array [1..N] of integer;
 SEMAF [origin 16#A000:16#0000]:array [1..N] of integer;
 (* Variáveis compartilhadas *)
 BETA1 [origin 16#9300:16#0000]:array [0..T] of VETOR;
 BETA2 [origin 16#9400:16#0000]:array [0..T] of VETOR;
 BETA3 [origin 16#9500:16#0000]:array [0..T] of VETOR;
 X1 [origin 16#9040:16#0000]:array [0..T] of VETOR;
 X2 [origin 16#9100:16#0000]:array [0..T] of VETOR;
 X3 [origin 16#9200:16#0000]:array [0..T] of VETOR;

procedure RICCATI;
begin
```

```
 resolver a equação matricial de Riccati;
end;

procedure COMPENSADOR;
begin
 resolver a equação do compensador;
end;

procedure ESTADO;
begin
 calcular o vetor de estado e controle;
 for K:=0 to T
   do
    X2[1,K]:=X[2,K];
end;

procedure COESTADO;
begin
 calcular o vetor de coestado e o multiplicador de Lagran-
 ge;
for K:=0 to T
do
  BETA2[1,K]:=BETA[2,K];
end;
(* PROGRAMA  PRINCIPAL *)

begin
 inicialização;
 RICCATI;
 repeat
   COMPENSADOR;
   ESTADO;
   COESTADO;
   FLAG[2]:=0;
   while (FLAG[1]=0 or FLAG[3]=0)
```

```
    do
    begin end;
    for K:=0 to T
    do
    begin
     X[1,K]:=X1[1;K];
     X[3,K]:=X3[1,K];
    end;
    INTERCON;
    for K:=0 to T
    do
    begin
     BETA[1,K]:=BETA1[1,K];
     BETA[3,K]:=BETA3[1,K];
    end;
    SEMAF[2]:=1;
    while (SEMAF[1]+SEMAF[2]+SEMAF[3]< 3)
    do
    begin end;
    FLAG[2]:=0;
    while (FLAG[3]=1 or FLAG[1]=1)
    do          //
    begin end;
    SEMAF[2]:=0;
 until (...)
end.
```

4.6.2 - Algoritmo de coordenação de coestado modificado

Nesta seção, descrevemos a implementação dos programas relativos as tarefas definidas na segunda etapa de paralelização para o segundo problema.

```
(* Programa referente a TAREFA i,(i=1) *)
program SISTEMA1;
type
    VETOR = array [0..T] of real;
var
   X1 : VETOR;     (* variável de estado *)
   Y1,Y2 : VETOR; (* variáveis de coestado *)
   Z2 : VETOR;     (* variável de interação *)
   (* variáveis compartilhadas na memória global *)
   Y1T [origin 16#9000:16#0000]:VETOR;
   Z1T [origin 16#9300:16#0000]:VETOR;
   Y2T [origin 16#9600:16#0000]:VETOR;
   Z2T [origin 16#9800:16#0000]:VETOR;
   (* variáveis de sincronização *)
   R [origin 16#9E00:16#0000]:integer;
   S [origin  16#9E10:16#0000]:integer;
   SEMAF1 [origin  16#9E20:16#0000]:integer;
   SEMAF2 [origin  16#9E22:16#0000]:integer;

procedure SUBSISTEMA1;
begin
 Calular a variável X1;
 Z1T:=X1;
 Calcular a variável Y1;
 Y1T:=Y1;
end;

(* PROGRAMA PRINCIPAL *)
begin
 inicialização;
 repeat
  SUBSISTEMA;
  R:=1;
  while S=0 do begin end;
  Y2:=Y2T;
```

```
  Z2:=Z2T;
  SEMAF1:=1;
  while ((SEMAF1+SEMAF2)<2) do begin  end;
  R:=0;
  while (S=1) do begin end;
  SEMAF1:=0;
 until (...);
end.

program SISTEMA2;
type
    VETOR = array [0..T] of real;
var
   X2 : VETOR;     (* variavel de estado *)
   Y1,Y2 : VETOR;     (* variaveis de coestado *)
   Z1 : VETOR;     (* variavel de interação *)
   (* variáveis compartilhadas na memória global *)
   Y1T [origin 16#9000:16#0000]:VETOR;
   Z1T [origin 16#9300:16#0000]:VETOR;
   Y2T [origin 16#9600:16#0000]:VETOR;
   Z2T [origin 16#9800:16#0000]:VETOR;
   (* variáveis de sincronização *)
   R [origin 16#9E00:16#0000]:integer;
   S [origin 16#9E10:16#0000]:integer;
   SEMAF1 [origin  16#9E20:16#0000]:integer;
   SEMAF2 [origin  16#9E22:16#0000]:integer;

procedure SUBSISTEMA2;
begin
 Calcular a variável X2;
 Z2T:=X2;
 Calcular a variável Y2;
 Y2T:=Y2;
end;
```

```
(* PROGRAMA PRINCIPAL *)
begin
 S:=0;
 R:=0;
 SEMAF1:=0;
 SEMAF2:=0;
 inicialização;
 repeat
  SUBSISTEMA2;
  S:=1;
  while (R=0) do begin end;
  Y1:=Y1T;
  Z1:=Z1T;
  SEMAF2:=1;
  while ((SEMAF2+SEMAF1)<2) do begin end;
  S:=0;
  while (R=1) do begin end;
  SEMAF2:=0;
 until (...);
end.
```

4.6.2 - Implementação sobre o sistema DIGIREDE 8000

Ojetivamos com esta seção, ressaltar os aspectos do estilo da programação e os mecanismos de comunicação, descrevendo uma implementação sobre dois computadores para o processamento paralelo do algoritmo anterior para o segundo problema. Embora seja ainda deficiente os mecanismos de comunicação empregados, uma vez que a versão III do UNIX não apresenta ferramentas de software mais poderosas presentes nas versões mais novas, e aliado ao fato de que os microcomputadores, no momento, não oferecem recursos de hardware mais propícios para a computação paralela foi possível realizarmos o "processamento paralelo".

Como na seção anterior, implementamos dois programas. Nesta seção, descreveremos apenas o programa

relativo a TAREFA i,(i=2), uma vez que como na seção anterior os programas têm a mesma estrutura.

```
 program SISTEMA2;
type
    VETOR = array [0..T] of real;
var
   X2 : VETOR;     (* variável de estado *)
   Y1,Y2 : VETOR;     (* variáveis de coestado *)
   Z1 : VETOR;     (* variável de interação *)
procedure SUBSISTEMA2;
begin
 Calcular a variável X2;
 for k:=0 to T
 do
  writeln(X2[k]);
 Calcular a variável Y2;
 for k:=0 to T
 do
  writeln(Y2[k]);
end;

(* PROGRAMA PRINCIPAL *)
begin
 inicialização;
 repeat
  SUBSISTEMA2;
  for k:=0 to T
  do
   readln(Y1[k]);
  for k:=0 to T
  do
   readln(Z1[k]);
 until (...);
end.
```

Para a comunicação com o outro programa, apresentamos um dos mecanismos de software utilizados :

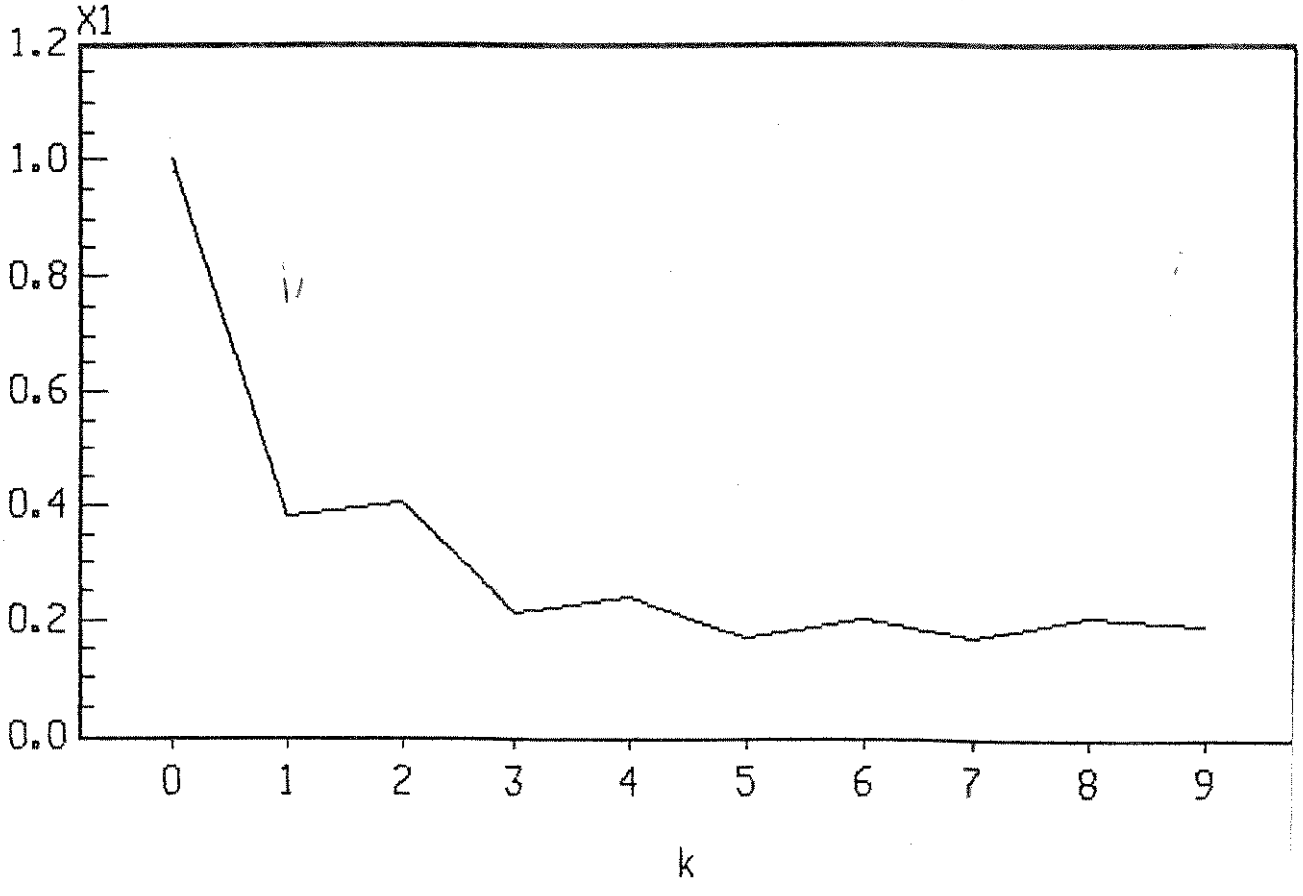
```
prog1 < /dev/tty8 | SISTEMA2 > /dev/tty9
```

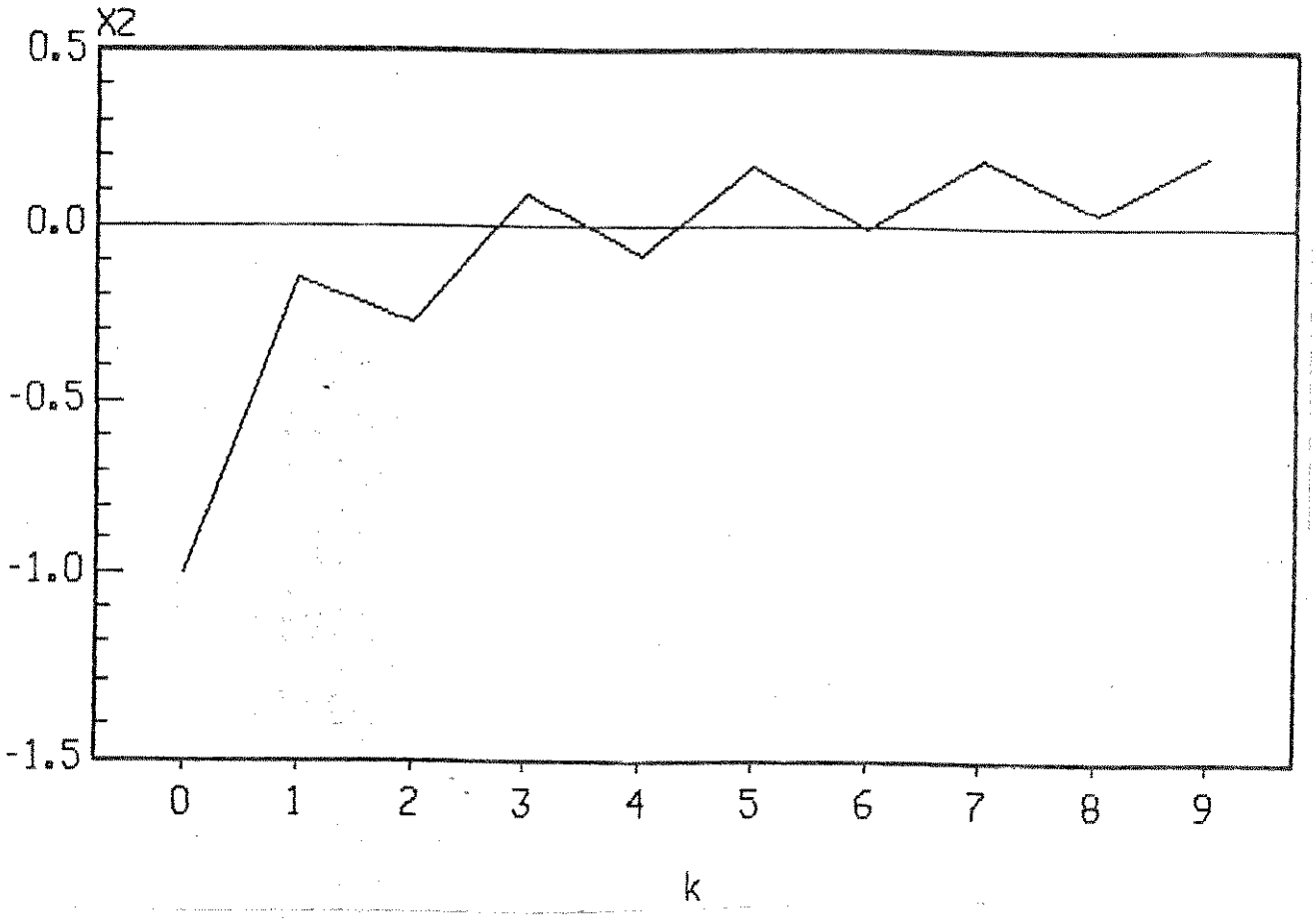
Nesta forma de comunicação, o programa prog1 se encarrega de ler os dados do arquivo /dev/tty8 e os canaliza para entrada do programa SISTEMA2 que após ser executado envia dados para o arquivo especial /dev/tty9.

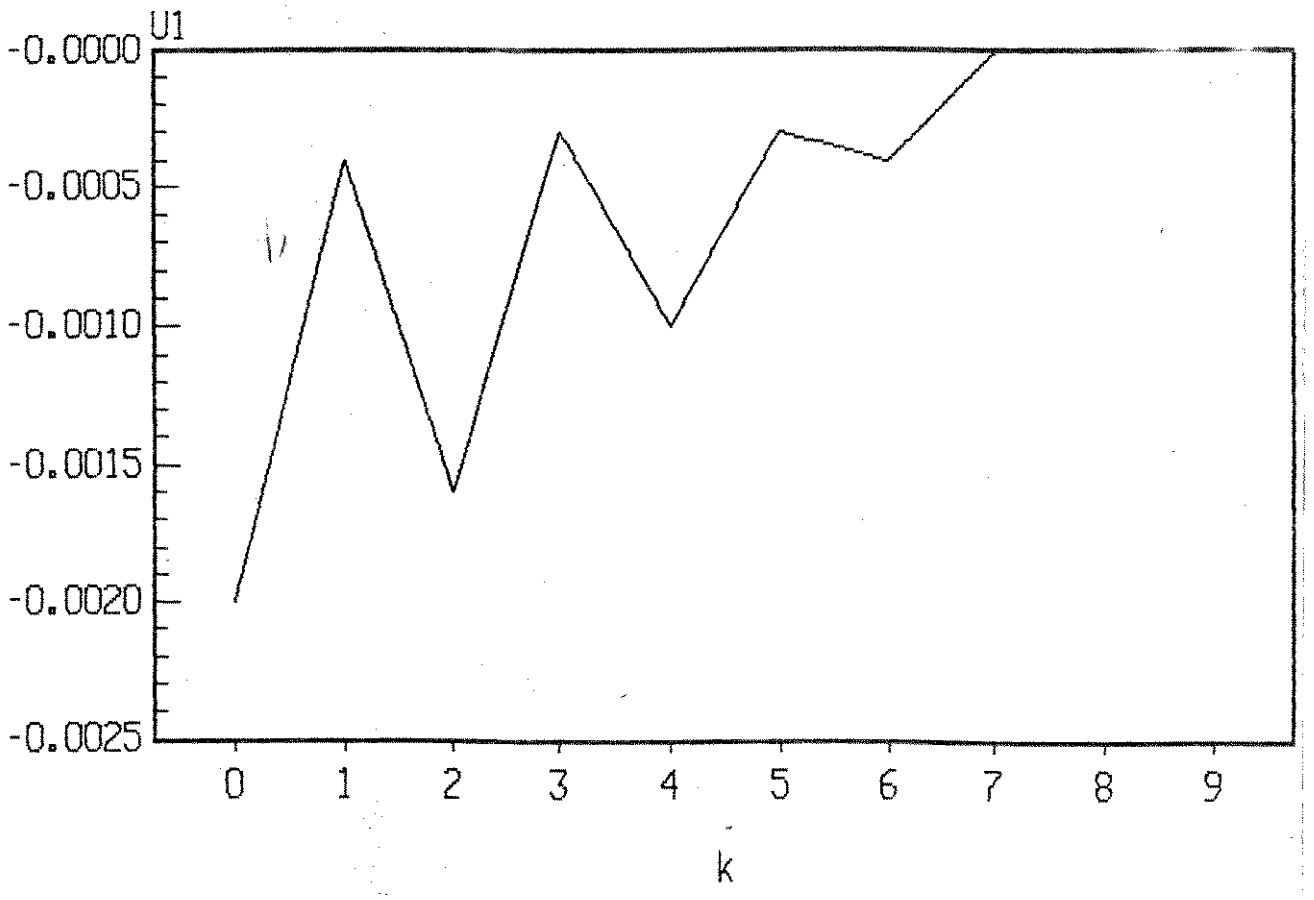
Esta forma de comunicação apresenta um ponto crítico com relação a sincronização, uma vez que os canais de comunicação não armazenam os dados de comunicação.

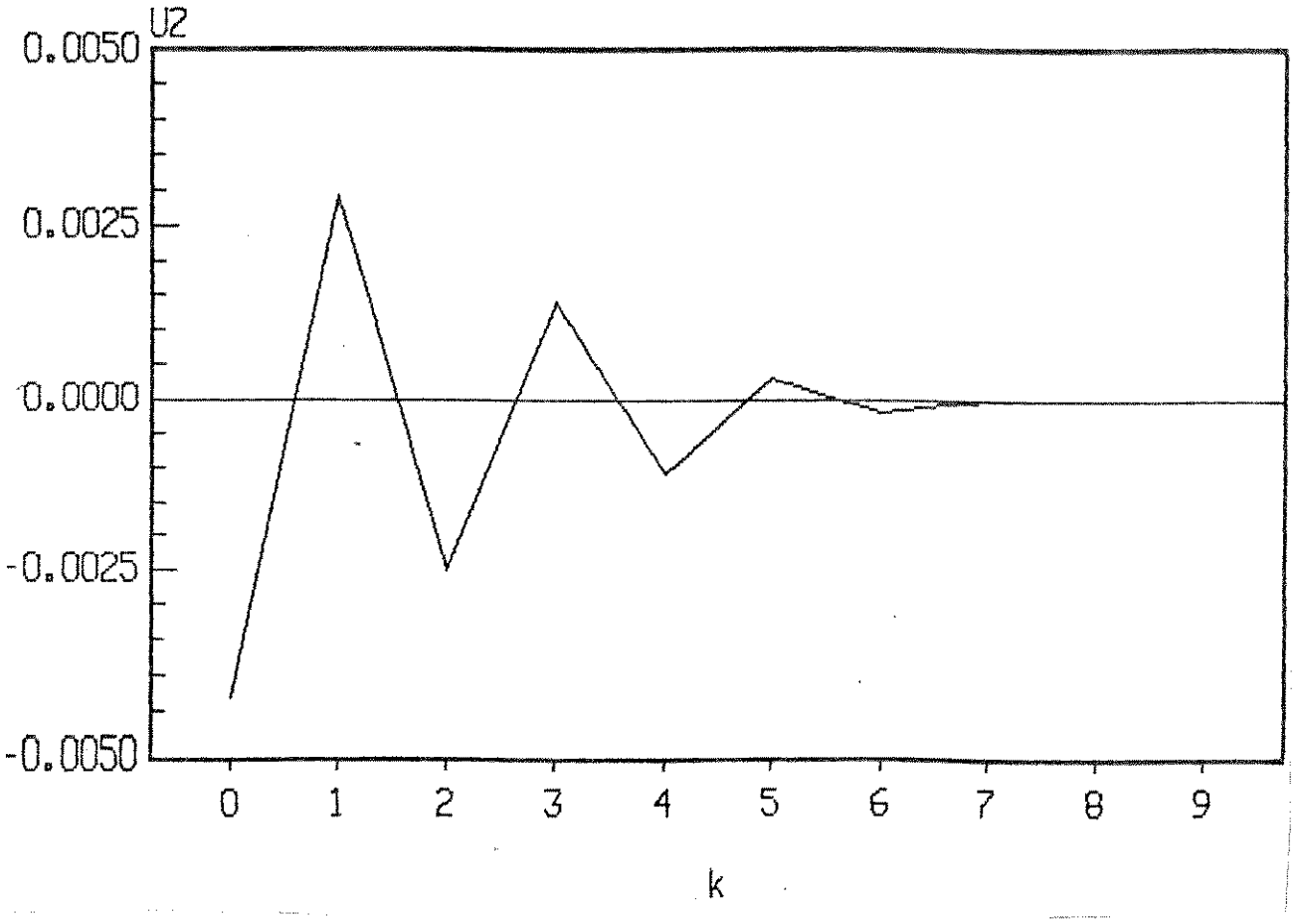
4.7 - Resultados numéricos

4.7.1 - Primeiro problema

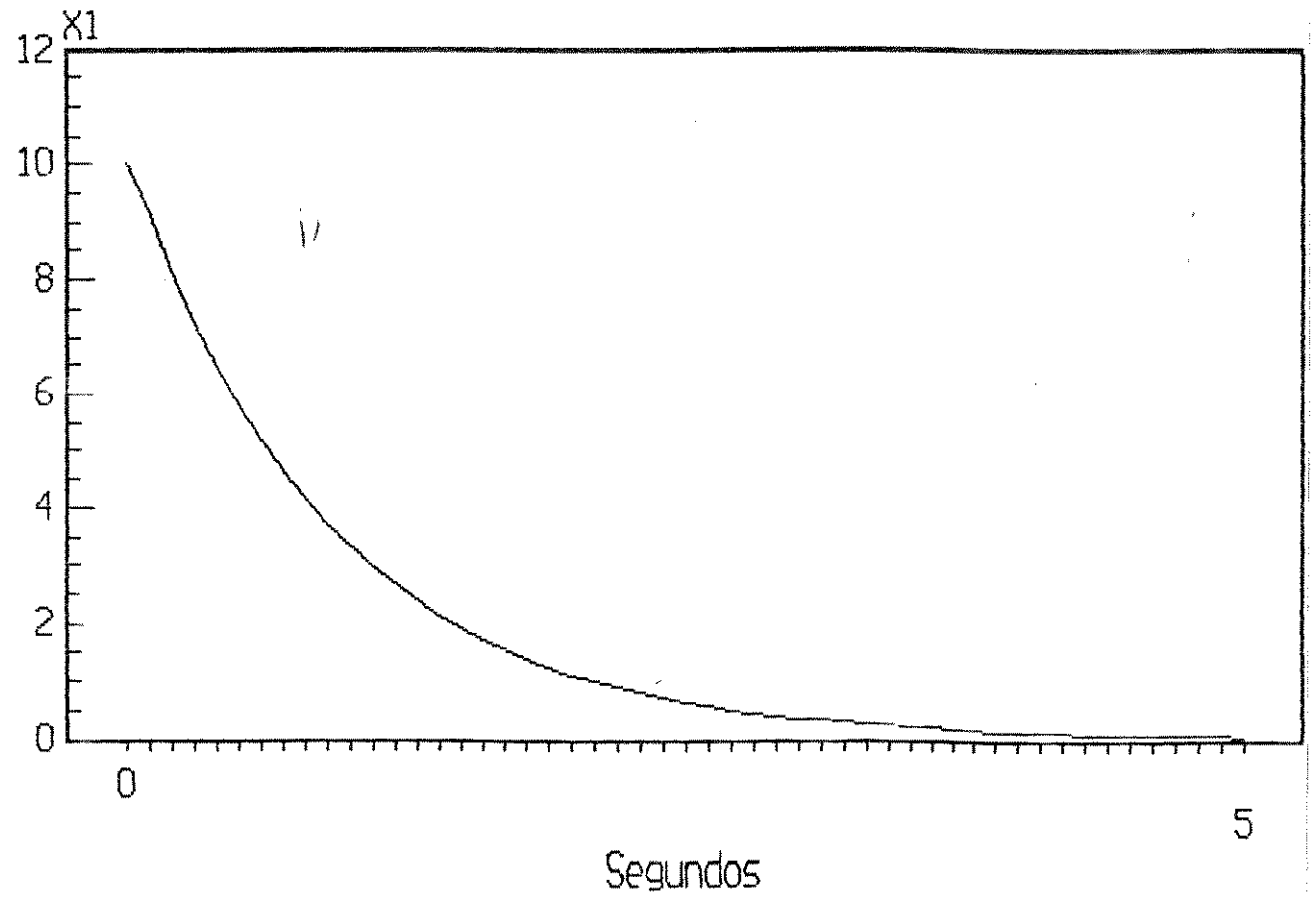


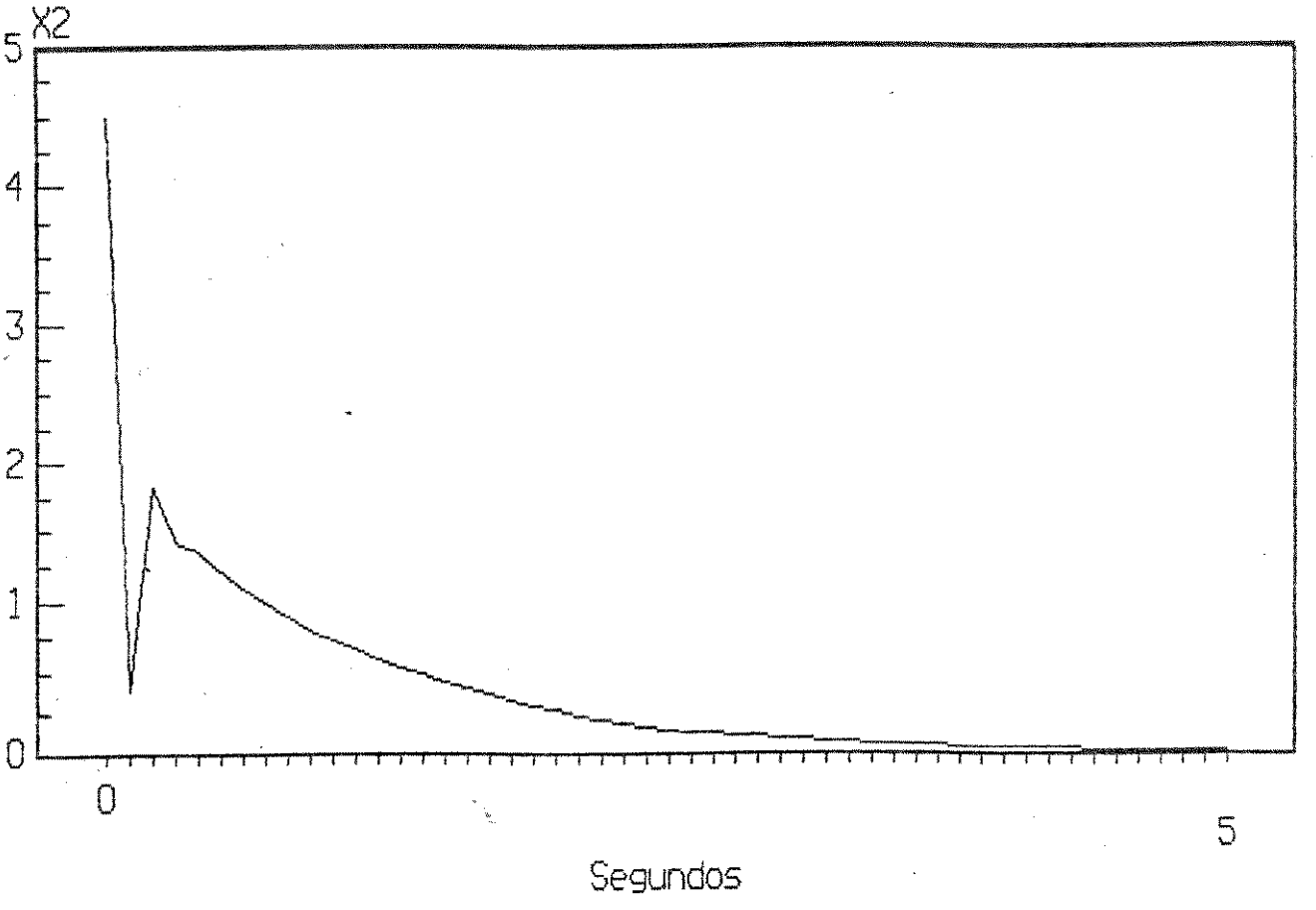

X2
0.5
0.0
-0.5
-1.0
-1.5
0
1
2
3
4
5
6
7
8
9
k

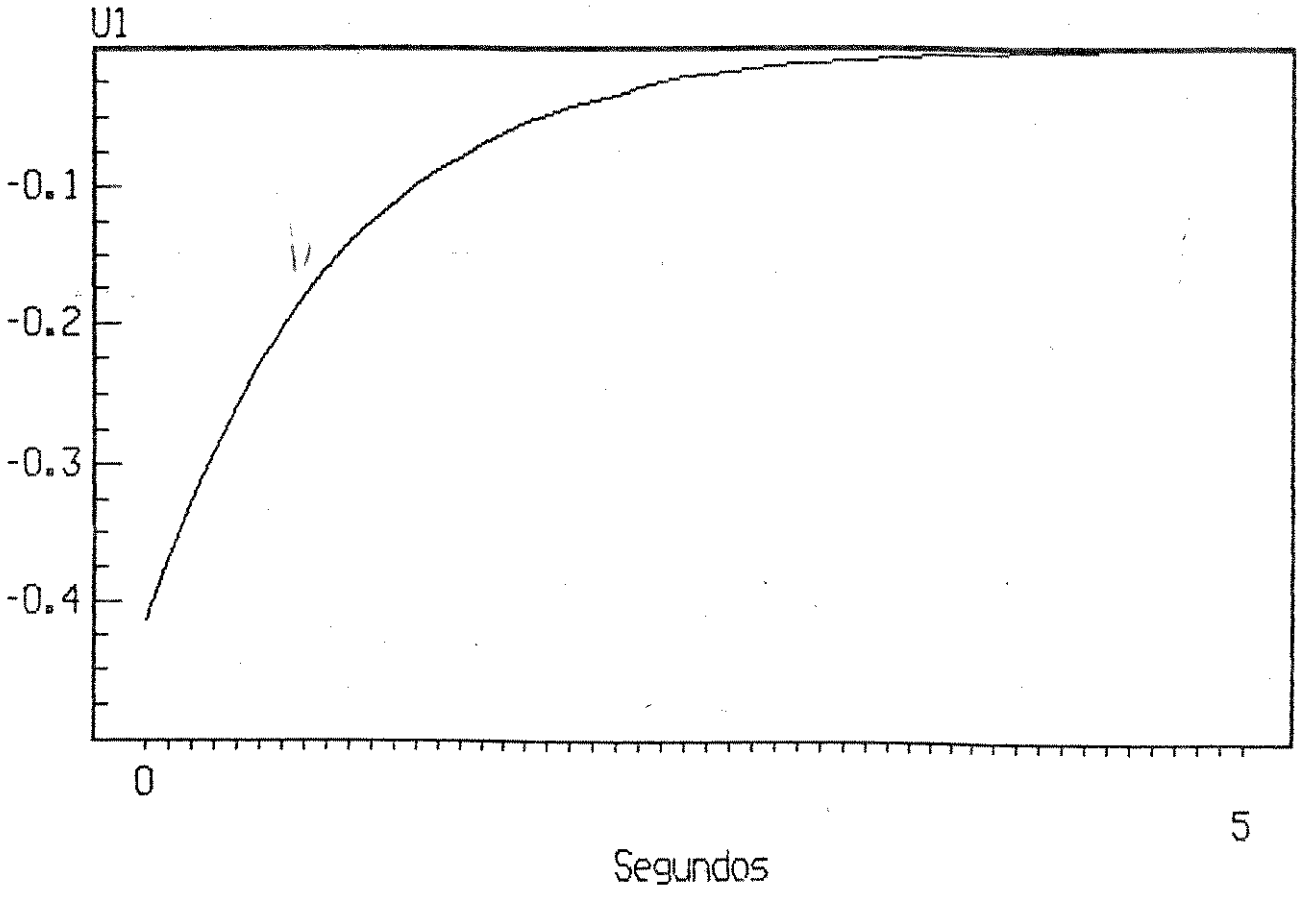

U1
-0.0000
-0.0005
-0.0010
-0.0015
-0.0020
-0.0025
0
1
2
3
4
5
6
7
8
9
k



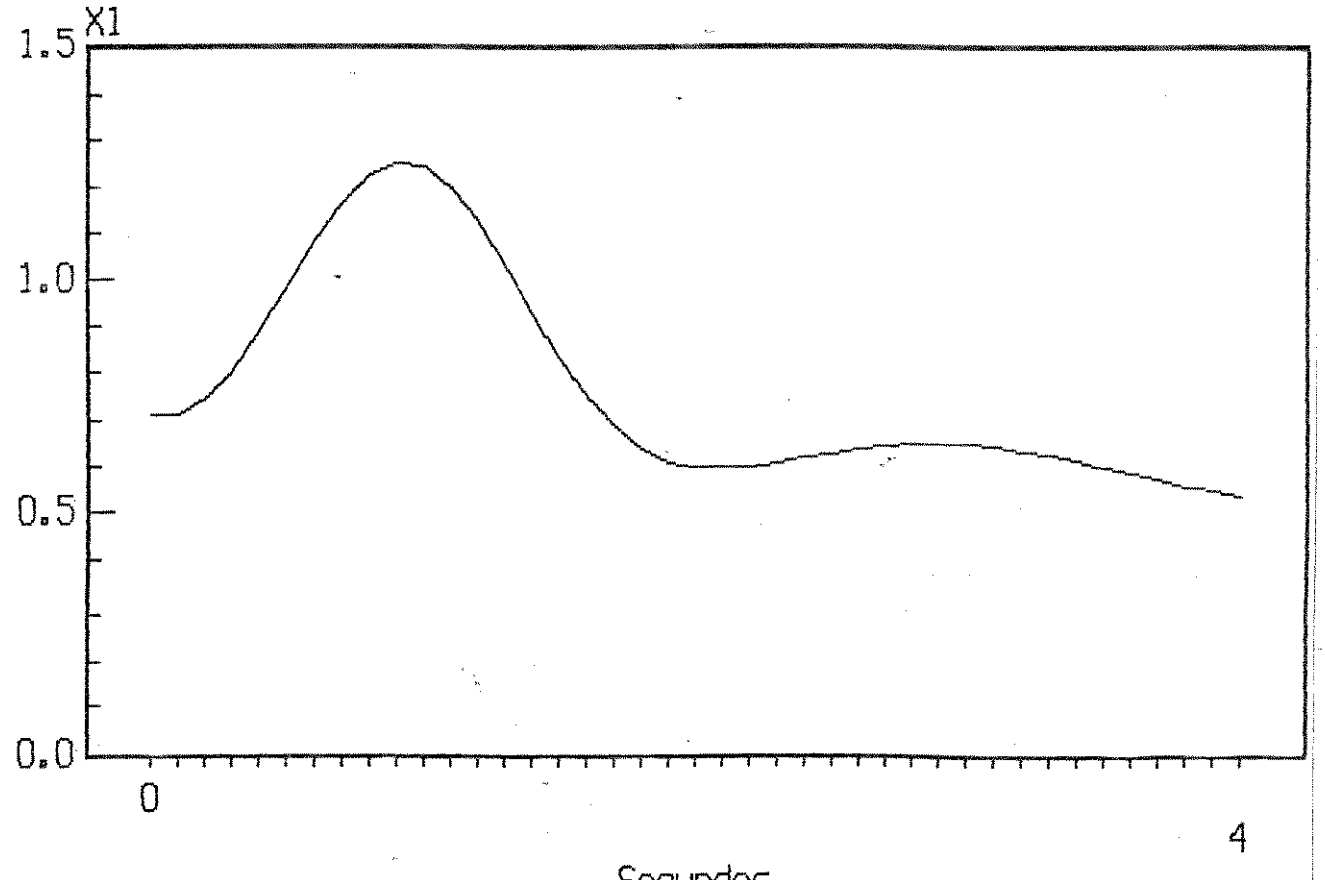
4.7.2 - Segundo Problema

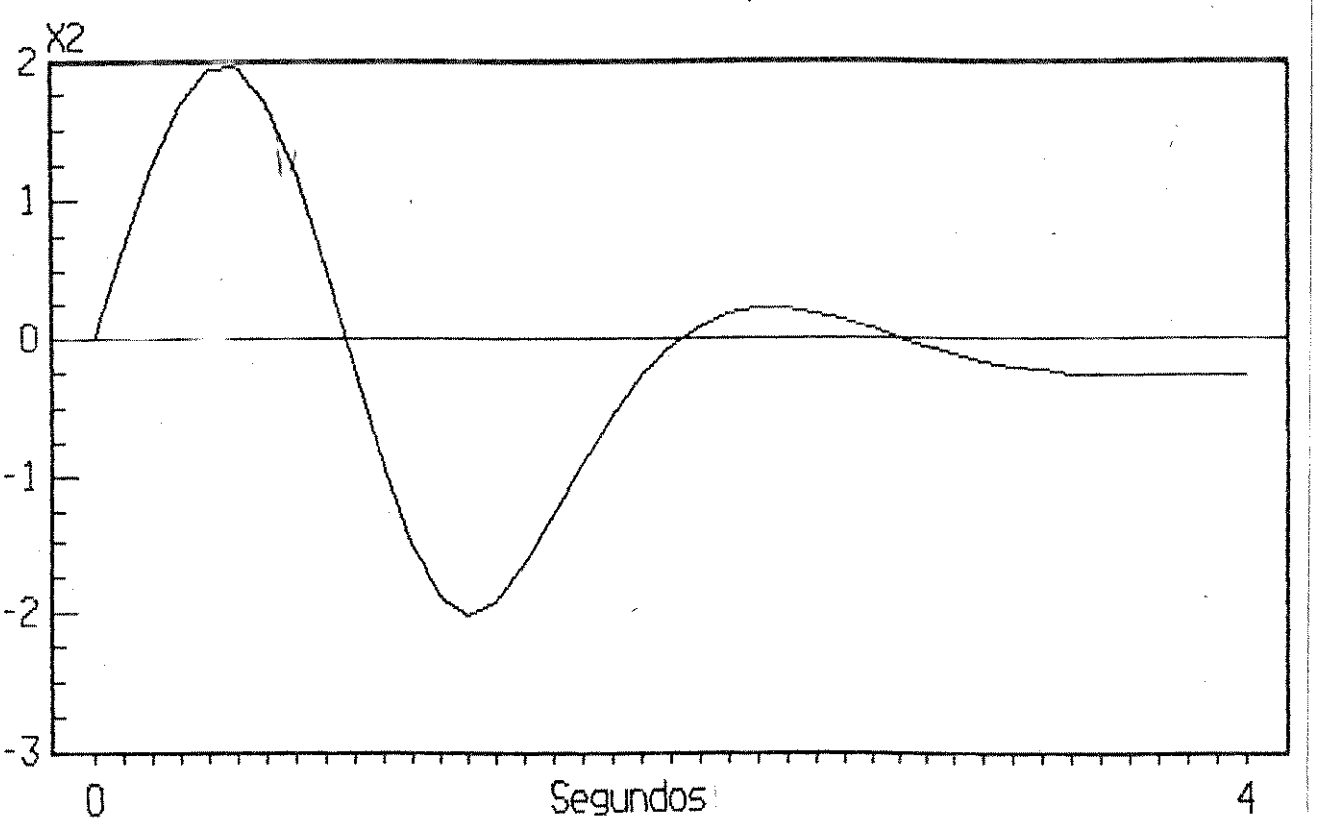


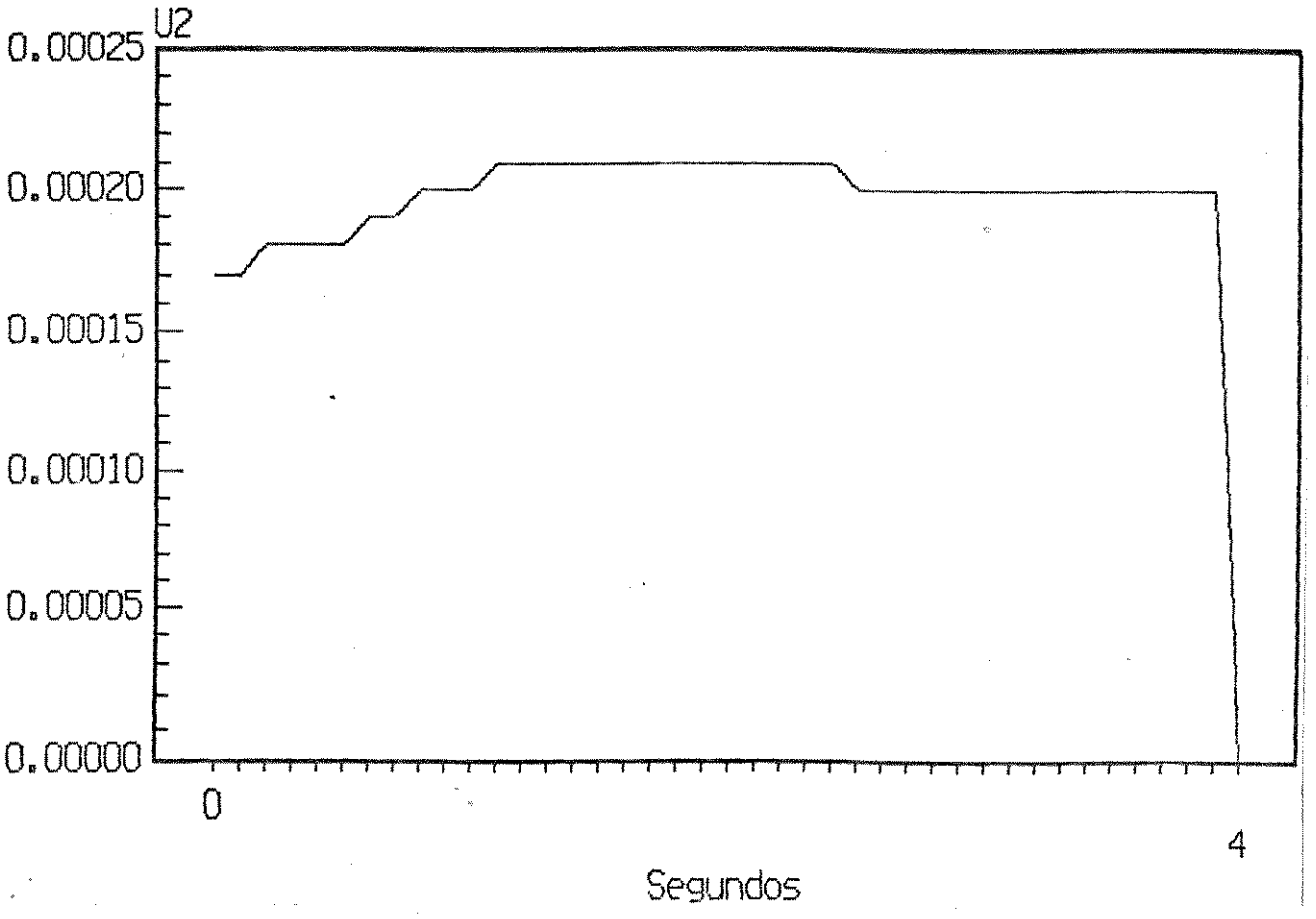

X2
5
4
3
2
1
0
0
5
Segundos


U1
-0.1
-0.2
-0.3
-0.4
0
5
Segundos

4.7.3 - Terceiro problema







4.8 - Medidas de desempenho

4.8.1 - Algoritmo predição de interação

As medidas de desempenho foram realizadas sobre o Processador Preferencial para o primeiro problema.

	IMPLEMENTAÇÃO		
	SEQUENCIAL	PARALELA 1	PARALELA 2
MT	42,7	96,3	82,8
MG		6,9	4,6
TP	4,8	2,5	1,56
TE		94,6	107,5
TC		41,5	
TS		96,0	36,0
	1 UCP	4 UCPs	3 UCPs

4.8.2 - Algoritmo de coordenação de coestado

Este algoritmo (A1) foi implementado para execução sequencial para os dois primeiros problemas. Com a modificação introduzida (predição de coestado) o algoritmo (A2) foi implementado para execução sequencial e paralela para todos os problemas.

4.8.2.1 - Primeiro problema

	IMPLEMENTAÇÃO		
	A1 (SEQUENCIAL)	A2 (SEQUENCIAL)	A2 (PARALELO)
MT	11,2	10,3	18,8
MG			3,58
TP	1,16	1,08	0,365
TE			21,02
TC			
TS			29,42
	1 UCP	1 UCP	3 UCPs

4.8.2.2 - Segundo problema

	IMPLEMENTAÇÃO		
	A1 (SEQUENCIAL)	A2 (SEQUENCIAL)	A2 (PARALELO)
MT	4,7	3,0	5,4
MG			13,80
TP	2,4	0,97	0,507
TE			29,17
TC			
TS			36,71
	1 UCP	1 UCPs	2 UCPs

4.8.2.3 - Terceiro problema

	IMPLEMENTAÇÃO		
	A2 (SEQUENCIAL)	A2 (PARALELO 1)	
MT	6,2	10,3	
MG		1,3	
TP	23,4	12,73	
TE		450,0	
TC			
TS		15,20	
	1 UCP	2 UCPs	

MT : memória total para armazenar os programas em Kbytes
MG : quantidade total de dados compartilhados em Kbytes
TP : tempo total de computação (segundos);
TE : tempo de execução da TAREFA i por iteração(miliseg.)
TC : tempo de execução da TAREFA j por iteração(miliseg.)
TS : tempo total gasto com sincronização e comunicação (milisegundos)

CAPÍTULO 5 - CONCLUSÃO

A paralelização de algoritmos de controle hierárquico para execução em sistemas de múltiplos processadores foi motivada por fatores tais como o tempo de processamento e a necessidade de se resolver problemas de grande porte.

Procuramos mostrar :

- Que a multiprogramação é uma ferramenta importante para o desenvolvimento e a simulação de algoritmos paralelos numa fase preliminar ;

- Um procedimento de paralelização de algoritmos de controle ótimo hierárquico para implementação em sistemas de múltiplos processadores ;

- Requerimentos para análise de desempenho que permitem verificar o desempenho computacional dos algoritmos paralelizados ;

- Que o desempenho computacional depende tanto da arquitetura de computação utilizada como do grau de paralelismo obtido e eficientemente implementado ;

- Que a paralelização dos algoritmos nas implementações apresentadas resultou da modificação da estrutura de cálculo.

A paralelização que realizamos dos algoritmos de controle hierárquico permitiu analizar e interpretar a estrutura de cálculo multinível, bem com propor a

modificação de esquemas de decomposição e descentralização, sobretudo no segundo algoritmo, bem como propor a distribuição do cálculo da coordenação pelo sistema de múltiplos processadores que acreditamos contribuirão para viabilizar a aplicação dos métodos apresentados na computação on-line da estratégia de controle ótimo de sistemas de grande porte.

REFERÊNCIAS BIBLIOGRÁFICAS:

[1] Xinogalas T.C., Dasigi S., Singh M.G.,"Coordination in Hierarchical Algorithms", IEEE Transactions on Systems,Man,and Cybernetic, vol.SMC-13, No.3,397-405, May/June 1983.

[2] Lasdon, L.S., Optimization Theory for Large Scale Systems, MacMillan, 1970.

[3] Mahamoud, M.S. ." Multilevel Systems Control and Applications: a survey".IEEE Trans. Syst.,Man Cybernet. SMC-7, 125-143,1977.

[4] Mahamuod, M.S. Assiri, J.A. "Performance Analysis of Two-level Structure on Finite-precision Machines". Automatica,Vol. 22, No. 3, 371-375,1986.

[5] Titli, A. Singh M.G. Mohamed, H.F., "Hierarchical Optimization of Dynamical Systems Using Multiprocessors". Comput. Elect. Engen. Vol. 5, 3-14, 1978.

[6] Papageorgiou, M. Schimidt, G., "Implementation of a Hierarchical Optimization Algorithm on a Multimicrocomputer System". IEEE Transactions on Systems, Man, and Cybernetic. Vol. SMC-13 No.1, 11-18,1983.

[7] Mahamoud M.S., "Dynamic Multilevel Optimization for a Class on Nonlinear Systems". International Journal of Control, vol. 30, No. 6, 927-948, December 1979.

[8] Mahmoud, M.S.,Sing M.G. Discret Systems-Analysis, Control and Optimization, SPRINGER-VERLAG 1984.

[9] Bottura, C.P., Tavares, H.M.F., Costa, E.A., Hierarchical Control of a Production-Transportation Network with Buffer Storages, Proceedings of the IEEE-1979 Conference on Cybernetics and Society, Denver Colorado, USR, October 8-10.

[10] Singh, M.G., Titili, A., Systems: Decomposition, Optimization and Control, Pergamon Press, 1978.

[11 Bottura, C.P., Costa Filho J.T. "On Parallel Computing for a Multilevel Optimization Algorithm", IFAC Workshop: Control Application of Nonlinear Programming, June 21-27/1988, Tbilisi, Russia.

[12] Parkinson D. , "Organizational Aspects of Using Parallel Computers", Parallel Computing, No.5, 75-83, 1987.

[13] Anita K.J., Peter S., "Experience Using Multiprocessor Systems - A Status Report." Computing Surveys, vol. 12, No. 2, 121-165, June, 1980.

[14] Gomide, F.A.C. "Análise e Implementação de Algoritmos de Controle Hierárquico de Sistemas Dinâmicos ", tese de mestrado, Campinas, 1979.

[15] Singh, M.G., Hassan, M., "A Two Level Prediction Algorithm for Non-Linear Systems", Automatica, January 1977.

[16] Bottura, C.P., Costa Filho,J.T. "Processamento Paralelo de Algoritmo de Controle Hierárquico", 7° Congresso Brasileiro de Automática, agosto/1988

[17] Schendell U., Introdution to Numerical Methods for Parallel Computer, Ellis Harwood Ltd., 1984.

[18] Iyer, S.N., Cory, B.J. "Optimisation of Turbo-Generator Transient Performace by Differential Dynamic Programming", IEEE Trans. Power Appar. Syst., 90, 2149-2157, 1971.

[19] Mukhopadhyay, B.K., Malik, O.P. "Optimal Control of Non-linear Power Systems by an Imbedding Method", International Journol of Control,17, 1041-1058,1973.

[20] Flynn M.J.,"Some Computer Organization and their Effectiveness ", IEEE Transaction on Computers, September, 1972.

[21] Conte, G., Corso D.D."Multi-Microprocessor System for Real-Time Applications", D. Reidel Publishing Company, 1985.

[22] Kanakia, H.R., Tobagi F.A., ."On distributed Computation with Limited Resources", IEEE Transactions on Computers vol. C-36, No. 5, 517-528, 1987.

[23] Bottura, C.P., Costa Filho, J.T., " Programação Paralela de Algoritmos de Otimização Multinível" , 1°Simpósio Brasileiro de Arquitetura de Computadores Processamento Paralelo, Gramado, 13 a 15 de maio de 1987. Comunicação.

[24] Karp, A.H. , Programming for Parallelism". Computer, May, 43-44,1987.

[25] Thomas, R. Yates, J.A User Guide to UNIX systems, Osborne/Magrow Hill, 1982.

[26] Kernighan, B.W., Moshey, J.R., The UNIX Programming Enviroment, Computer, April, 1981, pp 12-22.

[27] Bourne, J.R., The UNIX System, Addison Wesley, 1983

[28] Documentation MUNIX V.2/03 - Cadmus-PCS Gmbih, Munchen, 1984.

[29] Kramer,J. , Magee j., Sloman, M., "The CONIC Toolkit for Building Distributed Systems",IEEE Proceedings, vol.134, No.2, March, 1987.

[30] Lopes,Adilson Barbosa, LPM e LCM: Linguagens para Programação e Configuração de Aplicações de Tempo-Real, Tese de Mestrado em Engenharia Elétrica, Campinas, UNICAMP, 1986.

[31] Kramer,J. et. alli. "CONIC: An Integrated Approach to Distributed Computer Control Systems", IEE Proceedings, 130(1): 1-10, January, 1983.

[32] Adan Coelho, J.M.," Suporte de Tempo-Real para um Ambiente de Programacao Concorrente", Dissertação de Mestrado em Engenharia Elétrica, Campinas, UNICAMP, 1986.

[33] Magee J.N. -"Provision of Flexbility in Distributed Systems",Doctor's Thesis, University of London, April 1984.

[34] Bottura, C.P., Costa Filho, J.T., " Computação de Algoritmo de Otimização Hierárquica Via Multiprogramação". X Congresso Nacional de Matemática Aplicada e Computacional. Vol. 1, 82-88, 1987.

[35] CPqd, Especificação e características gerais do Processador Preferencial, doc. PP.EEA.001/CA-01-AB,1986.

[36] CPqd, Sistema Operacional PP-SO/P, ref. PPSO1.doc,1986.

[37] Zima H.P., Bast H.J., Geradt M.,"SUPERB: a Tool for Semi-Automatic MIMD/SIMD Parallelization", Parallel Computing, No. 6, 1-8, 1988.

[38] Zenios, S.A., Mucvey, J.M " A Distributed Algorithm for Convex Network Optimization Problems", Parallel Computing, Vol. 6, No. 1, 45-56, 1988.

[39] Bottura, C.P., Costa Filho,J.T., "Controle Hierárquico via Predição de Co-estado Utilizando um Sistemas de Múltiplos Microcomputadoes". Anais do 3o. Congresso Latino-Americano de Automática, 7-11 de novembro de 1988, Vina del Mar, Chile.

[40] Bottura, C. P., Costa Filho, J. T., "Processamento Paralelo assíncrono de um Algoritmo Multínivel". II-SBAC-PP-SBC, 26-28 setembro de 1988. Águas de Lindoia, SP.