UM MODELO ESTOCÁSTICO PARA AVALIAÇÃO DE DESEMPENHO EM PROCESSAMENTO DISTRIBUÍDO

por

José Everardo Bessa Maia

Este exemplar corresponde à redação final da tese defendida por Jose Everardo Bessa Maia e aprovada pela Comissão Julgadora em 25 de Julho de 1989

Orientador: Prof. Dr. João Bosco Ribeiro do Val

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

- MAIO DE 1989 -

À

Rafael, Joana e Eloisa

UM MODELO ESTOCÁSTICO PARA AVALIAçÃO DE DESEMPENHO EM PROCESSAMENTO DISTRIBUIDO

RESUMO

É apresentado um modelo de Cadeia de Markov a Tempo Contínuo (CMTC) para avaliação de desempenho de sistemas multiprocessadores na execução de sistemas de tarefas com estrutura hierárquica. O trabalho desenvolve a conceituação necessária à apresentação do problema e apresenta solução computacional para o modelo. São estudadas duas arquiteturas: arquitetura radial e arquitetura por barramento compartilhado. Um índice de desempenho é estabelecido que permite comparar as arquiteturas. Os resultados são aplicados na execução de algoritmos de otimização hierárquicos. Os casos não-Markovianos são tratados através de técnicas de simulação de eventos discretos.

ÍNDICE

INTRODUÇÃO

1. INTRODUÇÃO

A complexidade e o volume de cálculos demandados em aplicações de engenharia e ciência crescem de forma vertiginosa impondo solicitações extremas aos sistemas de computação. Dois contextos se destacam por requisitarem capacidade de processamento de ordem elevada: processamento científico e aplicações de tempo real.

No processamento científico, algumas áreas exigem o processamento de massas gigantescas de dados e colocam problemas da ordem de milhões de operações. Exemplos são simulação nuclear (3D)[34], devido a complexidade computacional, e processamento de imagens de satélite (4Kx4K)[35], pelo grande volume de dados. Os setores de metereologia e defesa (estratégia) são exemplos de aplicações onde também os requisitos de tempo impõem limites rígidos nos critérios de desempenho. Claramente, o processamento paralelo se delineia como a alternativa futura, mas já em andamento, para os problemas de processamento científico.

Nas aplicações de tempo real, onde o sistema de computadores deve fornecer respostas em tempo hábil sob pena de operação deficiente ou desastrosa, uma capacidade de processamento cada vez maior vem sendo exigida pelos sistemas de controle e automação. Com isto, os limites técnicos, de confiabilidade e de custo das soluções baseadas em unidade processadora única de alta capacidade foram sendo atingidos. E cada vez mais, a decisão entre usar um processador de maior capacidade ou um sistema multiprocessadores, tende para a segunda.

Alem das razões mencionadas, o processamento paralelo atende a natureza distribuida de certas aplicações e permite explorar o

paralelismo intrínseco de certos problemas ou algoritmos. Esta última motivação surge em certos problemas de tempo crítico, onde o paralelismo é procurado, muitas vezes mesmo, para viabilizar a aplicação. Muito esforço de pesquisa tem sido feito, no sentido de obter algoritmos paralelos para os mais variados problemas, como demonstra o número de artigos e periódicos especializados existentes na literatura.

Por outro lado, também cresce o número de sistemas complexos que são concebidos sobre arquiteturas distribuidas em áreas de aplicação não numéricas. Exemplos são as centrais de comutação telefônicas CPA e estações de trabalho que compartilham recursos, onde se colocam os mesmos problemas de análise de desempenho tratados aqui.

A integração VLSI tem possibilitado o surgimento de processadores relativamente potentes, a custo cada dia mais acessível. Com isto, sistemas multiprocessadores tem sido implementados e experimentados com mais frequência nas universidades e centros de pesquisa.

Com os sistemas multiprocessadores, novos conceitos são introduzidos na área de confiabilidade já que estes sistemas não são do tipo falha/funciona. Isto é, com falha em uma ou algumas das unidades, os sistemas continuam funcionando embora com desempenho degradado. Assim, conceitos de desempenho e confiabilidade ficam estreitamente relacionados.

De uma forma geral, dentre as principais motivações que estimularam o aparecimento da computação paralela e distribuida, podemos assinalar[33]:

.desempenho- como diversas unidades estão operando em conjunto na execução do programa do usuário, podemos esperar que o tempo de processamento seja inversamente proporcional ao número de unidades.

.modularidade- tanto para o fabricante quanto para o usuário, é importante que a arquitetura possa ser expandida incrementalmente através da adição de módulos: aplicações pertencentes a diversas faixas de demanda computacional poderiam ser atendidas a partir de um mesmo produto. Muitas máquinas paralelas comercialmente disponíveis apresentam essa característica de modularidade.

.tolerância a falhas- muitas aplicações requerem que o sistema de computação assegure operação contínua na presença de falhas de hardware. Dessa forma, sistemas multiprocessadores podem verificar se falhas ocorrem, diagnosticando-as. O componente hardware defeituoso pode então ser retirado pelo sistema, que continua em operação com uma nova distribuição de tarefas essenciais, ainda que com sua configuração degradada.

Quando utilizamos processamento pararelo, do ponto de vista de ganho de velocidade, o ideal é que o desempenho da máquina aumente linearmente com o número de processadores utilizados. Isto é, dada uma máquina com p processadores, queremos resolver nosso problema num tempo tão próximo quanto possível da razão 1/p do tempo necessário para resolver o mesmo problema numa versão uniprocessador da mesma máquina, empregando o melhor algoritmo serial possível. Para isso, procura-se explorar a execução de eventos concorrentes[13]. Concorrência implica paralelismo, simultaneidade ou execução por estágios(pipeline), ou uma combinação destes. Paralelismo é a ocorrência de eventos em

múltiplos recursos em um mesmo intervalo de tempo. Simultaneidade é a ocorrência de múltiplos eventos em um mesmo instante de tempo. Execução por estágio é a execução de eventos cascata em pares de intervalo de tempo sobrepostos. Estes mecanismos de execução concorrente podem ser implementados em vários níveis de processamento. Por vários fatores, veremos que uma linearidade no ganho de velocidade não acontece.

Uma tarefa é um conjunto de operações relacionadas que podem ser executadas sobre dados de entrada. É um segmento de programa que pode ser executado em bloco sem requerer dados de outro segmento durante sua execução. A execução de uma tarefa pode ou não gerar dados de entrada para outras tarefas. Exemplos de tarefas são funções, subrotinas ou processos. Um algoritmo é uma coleção de tarefas com uma estrutura determinada. O principal aspecto desta estrutura é a relação de precedência. Esta define o sequenciamento e a comunicação entre as tarefas. Um programa é um exemplo de algoritmo.

Os algoritmos de execução pararela podem pertencer a duas categorias diferentes: síncronos ou assíncronos[13]. Nos algoritmos sincronizados, algumas tarefas não podem ser ativadas até que outras tarefas se tenham completado. Na execução destes algoritmos, em geral os processadores tem períodos de bloqueio, onde ficam esperando mensagens de outras tarefas. Nos algoritmos assíncronos, as tarefas não precisam esperar pelo término de outras executadas em outros processadores para continuarem sua execução. Seja porque não há acoplamento entre as tarefas, seja porque algum mecanismo de atualização antecipada das variáveis é

implementado, como por exemplo, predição linear. Esta solução minimiza o fenômeno do bloqueio.

Um **recurso** é qualquer 'facilidade' que possa ser utilizada por um processador. Barramento, memória, canais de entrada ou saída, ou mesmo outro processador são exemplos de recursos. Quando um recurso pode ser utilizado por mais de um processador dizemos que está sendo compartilhado.

Num sistema de processamento paralelo, os processadores devem trabalhar de forma cooperante trocando mensagens e dados entre si. Uma parte integrante de qualquer destes sistemas é sua rede de interconexões, usada para interligar processadores e, possivelmente, módulos de memória. Módulos de memória e rede de interconexões geralmente são os recursos compartilhados. Assim, surgem fenômenos de **comunicação**, de **sincronização** e de **bloqueio** entre os processadores. Os atrasos devidos a estes fenômenos são responsáveis por aumento no tempo de execuçao dos algoritmos. Os atrasos de comunicação referem-se a troca de mensagens, uma vez que não é instantânea. A sincronização gera tempos em que os processadores não realizam trabalho útil esperando por outros processadores. O bloqueio refere-se à espera de processadores por um recurso compartilhado que esteja sendo utilizado por outro processador. A análise de desempenho é fundamental na avaliação dos efeitos destes fenômenos com o objetivo de prever comportamentos e comparar soluções. Ela é necessária na análise e síntese de algoritmos e de arquiteturas, e no estudo do comportamento global do conjunto. Deve fornecer subsídios sobre os tipos de computadores paralelos que serão mais eficientes em aplicações específicas ou mesmo em uso geral de processamento.

A análise de desempenho de um sistema de computadores requer a definição clara de dois parâmetros[06]: a carga de trabalho(workload) e o índice de desempenho. A carga de trabalho de um sistema de computação é a coleção de tarefas de processamento de dados submetidas ao sistema durante um período de tempo especificado. Assim, algorítmos diferentes dão origem a cargas de trabalho diferentes para um mesmo sistema de processamento. Índices de desempenho são medidas numéricas da eficiência com que uma arquitetura executa uma determinada estrutura de tarefas. Os detalhes do algoritmo não são importantes para propósitos da análise de desempenho, exceto os passos que envolvem um bloco de computação e comunicação entre processadores.

O desempenho do sistema computacional é grandemente influenciado pelo grau de ajuste entre a arquitetura do sistema de computadores(física) e a estrutura do sistema de tarefas(lógica). Um maior ou menor acoplamento entre as partes de um algoritmo que são executadas em paralelo define o volume de dados e mensagens trocadas entre os processadores. As facilidades oferecidas pela arquitetura para esta troca de mensagens dá a magnitude dos atrasos. Neste sentido, arquiteturas que facilitem a comunicação entre processadores deverão ter melhores índices de desempenho para uma mesma carga de trabalho.

Esta dependência do desempenho com a carga de trabalho é um conceito importante quando se trata de análise de desempenho.

Diferenciaremos duas classes de aplicações dos sistemas de computadores. Numa primeira categoria, que chamaremos de

'aplicações gerais de processamento de dados', o sistema atende uma variedade de usuários com requisitos de serviço completamente diversos gerando cargas de trabalho totalmente heterogêneas. Noutra categoria estão as aplicações dedicadas, muitas vezes operando em tempo real, caracterizando-se por uma estrutura fixa de carga de trabalho. O caso extremo mais representativo é a execução de um algoritmo específico.

Entre estes dois extremos existem aplicações com maior ou menor regularidade na estrutura da carga de trabalho. Na análise de desempenho, diferentes abordagens são adotadas, dependendo do contexto e dos objetivos do estudo. A literatura apresenta desde estudos determinísticos muito específicos até modelos probabilísticos bastante gerais. Um dos principais fatores de complexidade destes modelos são as exigências de sincronização impostas pela estrutura da carga de trabalho.

Com a crescente utilização de sistemas multiprocessadores, os estudos em modelagem e simulação para avaliar o desempenho destes sistemas tem-se tornado mais frequentes.

Modelos estocásticos tem sido desenvolvidos na literatura para analisar o desempenho de sistemas multiprocessadores na execução de um conjunto de tarefas. Basicamente, os modelos encontrados baseiam-se em Redes de filas, Cadeias de Markov e Redes de Petri Estocástica. Utilizam-se resultados analíticos da teoria estocástica sempre que possível ou técnicas estatísticas em simulação, de outro modo.

Em [04], modelos bastante gerais são desenvolvidos para sistemas com um só barramento e em [03] são apresentados modelos para múltiplos barramentos interligando os processadores. Em

[06], uma carga de trabalho típica de sistemas de controle para operação em tempo real é assumida e um modelo de redes de filas é apresentado. Nos casos citados, a carga de trabalho é bastante geral, sendo que básicamente a única hipótese assumida é de referência uniforme entre os processadores. Por referência uniforme queremos dizer que cada processador gera acessos a qualquer outro processador com mesma probabilidade.

Modelos de referência não uniformes são empregados em [19] e [05]. Em [19] é tratado o caso em que há uma probabilidade localizada de acesso a um dos processadores, e em [05] o problema é generalizado para distribuição de probabilidade quaisquer entre os processadores.

Tendo em vista principalmente o problema de compilação de algoritmos em circuitos integrados, onde existem limitações de volume (dissipação), [12] faz uma análise comparativa entre processamento distribuido e processamento centralizado para certas estruturas de tarefas acíclicas. Os tempos de comunicação são desprezados. O trabalho usa modelos de filas e impõe a restrição de que a capacidade de processamento da unidade centralizada é a soma das capacidades na arquitetura distribuida.

Todos os modelos anteriores assumem uma estrutura de tarefas assíncrona e acíclica.

Em [07], [08], [24] e [25], é tratado o problema do processamento paralelo de uma classe de estruturas de tarefas acíclicas com sincronismo. O modelo assume que grupos de tarefas que admitem processamento paralelo chegam segundo um processo de Poisson e que tem um ponto de sincronismo após o processamento. O

processo admite um modelo de rede de filas, no entanto, devido à complexidade do problema, solução fechada só é conseguida para 2 processadores em [24] e [25], e soluções aproximadas são dadas para um número maior de processadores em [07] e [08].

Por outro lado, no estudo de algoritmos paralelos os trabalhos se concentram principalmente sobre algoritmos específicos sendo executados em arquiteturas específicas. Para exemplos, veja [23] e [27]. Ao contrário dos modelos estocásticos, esta abordagem permite uma análise detalhada em termos de número de operações e transferência de dados que otimiza a execução para cada caso.

Uma abordagem menos específica para previsão de desempenho aparece em [10] e [11]. Os algoritmos são agrupados segundo critérios de decomposição na obtenção do paralelismo. Classes de decomposição são caracterizadas, permitindo prever o desempenho sem se deter em detalhes de algoritmos particulares. Nestes estudos, o principal objetivo é avaliar os efeitos da comunicação sobre o ganho de velocidade.

Este trabalho situa-se num plano intermediário. Nele se isola uma classe de algoritmos paralelos e obtem-se um modelo estocástico de desempenho para a classe focalizada. Especificamente, analiza-se a carga de trabalho imposta por algoritmos hierárquicos sincronizados de dois níveis. A estrutura de tarefas é cíclica e possui um ponto de forte sincronização. Descrevemos o aspecto estocástico dos tempos de execução e comparamos o índice de desempenho para duas arquiteturas amplamente utilizadas, admitindo diferentes distribuições de probabilidade para os tempos de processamento e comunicação.

Em [01] e [02], este problema é apresentado e resolvido de forma simplificada, sem considerar no modelo os efeitos de comunicação e bloqueio. Com esta simplificação, o caso de tempos de processamento exponencialmente distribuidos é de análise imediata. Para distribuição geral, o problema é abordado pelo método dos estágios. A solução é dada para 2 processadores.

A nossa análise coloca outra vez o problema apresentado em [01] e [02] e faz uso dos critérios de agrupamento baseados em estratégias de decomposição expostos em [10] e [11]. O problema que analisamos não admite modelo de rede de filas. Sob hipóteses adequadas, o modelo é uma Cadeia de Markov a Tempo Contínuo (CMTC) e incorpora sincronização, comunicação e bloqueio na execução do algoritmo. Esta categoria de modelos não representa qualquer algoritmo específico na referida classe, mas mantem todas as restrições estruturais comuns à classe estudada. A principal contribuição do estudo é que ele permite ganhar compreensão das questões envolvidas na execução paralela de algoritmos e fornece parâmetros para comparação de arquiteturas.

No capítulo 2 introduzimos vários conceitos importantes para a compreensão do restante do trabalho. Na seção 2.1 são descritas as duas arquiteturas estudadas. Na seção 2.2 definimos a simbologia e as estruturas básicas para a representação da carga de trabalho. Na seção 2.3 descrevemos o modo de operação da classe de algoritmos e exemplificamos algumas aplicações. Na seção 2.4 o índice de desempenho é definido.

No capítulo 3 são desenvolvidos os modelos de análise. Na seção 3.1 é analisado o caso onde a comunicação é negligenciada.

Na seção 3.2 desenvolvemos um modelo de operação com comunicação para a arquitetura radial. Na seção 3.3 a operação da arquitetura por barramento é analisada por simulação. Na seção 3.4 é feita uma análise comparativa das arquiteturas.

No capítulo 4 os resultados são aplicados a problemas mais complexos de decomposição de algoritmos multinível relativos a partições distintas.

No capítulo 5 são apresentadas algumas conclusões e apontadas algumas linhas de trabalhos futuros.

O resultado mais importante do trabalho, representado pelo Lema 2 do capítulo 3 é provado formalmente no apêndice 1. O apêndice 2 é uma revisão dos principais conceitos utilizados no processo de simulação.

CAPÍTULO 2: CONCEITUAÇÃO GERAL

2.1. ARQUITETURAS MULTIPROCESSADORES

As características de paralelismo da estrutura de processamento referidas anteriormente são exploradas pelas arquiteturas em diversos níveis de processamento. Emprega-se desde o paralelismo a nível de instrução, como nas máquinas vetoriais, até os sistemas multiprocessadores fracamente acoplados. Neste trabalho abordaremos sistemas onde processadores autônomos podem se comunicar através de uma rede de interconexão que permita um médio acoplamento.

Procuraremos aqui não entrar em classificações, mas descrever diretamente as arquiteturas que analizaremos. Consideramos duas arquiteturas neste trabalho, que chamaremos arquitetura radial e arquitetura por barramento.

Na arquitetura radial temos um processador B no nível superior e vários processadores Ai, i=1,2,..,n , no nível inferior(fig.1b). Existe um canal de comunicação direto e privado entre cada processador Ai e o processador B, e não há comunicação de quaisquer dois processadores Ai entre si. Consideramos que o processador B não aceita comunicação simultânea com mais de um processador Ai e que portanto haverá bloqueio se mais de um processador Ai tentar comunicação simultâneamente. Consideramos tambem que o processador B tem prioridade sobre qualquer dos processadores Ai para realizar uma comunicação e que os processadores Ai são atendidos na disciplina FCFS do instante em que solicitam comunicação. Referências a sistemas multiprocessadores com esta arquitetura podem ser

encontradas na literatura[01][02], no entanto, ela surge mais frequentemente por imposição de aplicações com subsistemas descentralizados.

Na arquitetura por barramento, temos um barramento compartilhado onde vários processadores Pi, i=1,2,..,n+1 , estão ligados, e ao qual tambem está ligada uma memória comum(MC)(fig.1a). Cada processador tem sua memória própria onde acessa dados e programas, enquanto a memória comum será usada para troca de mensagens entre processadores. Muitas implementações são feitas nesta arquitetura pelo seu alto grau de padronização e possibilidades de expansão modular. A troca de mensagem acontece em duas fases: inicialmente o processador origem transfere a mensagem de sua memória local para a memória comum, e numa segunda fase, o processador destino lê dados da memória comum transferindo para sua memória local. Observe que um processador que precise de dados atualizados não tem como saber a priori se estes dados estão disponíveis ou não, a não ser através de um acesso a memória comum. Isto significa que muitos acessos poderão ser feitos até que os dados desejados sejam obtidos, o que contribuirá para aumentar o tempo total de execução. O problema de bloqueio surge sempre que mais de um processador tenta acessar a memória comum simultâneamente, o que causará aumento no tempo de comunicação por atraso na obtenção do recurso pelos processadores.

Assim, inerente a toda arquitetura organizada por barramento está um mecanismo que disciplina o uso do barramento. A decisão pode ser centralizada em um controlador de barramento ou

distribuida entre as unidades de processamento. Os três principais mecanismos de decisão centralizada estudados na literatura são: daisy chain, polling e independent request, descritos abaixo.

.Daisy chain(Fig.2a)- Existem duas linhas comuns a todos os processadores. Uma, chamada "request", através da qual é feita a solicitação para uso do barramento. Outra, chamada "Grant", interligando cada processador sequencialmente. Sempre que há uma solicitação para uso do barramento, um sinal que libera o acesso é propagado a partir do controlador através da linha Grant. A primeira unidade que solicitou o barramento e receber o sinal bloqueia a propagação e toma para si o barramento. Ao liberar, um novo sinal é emitido pelo controlador. Neste caso, há um esquema de prioridade inerente, onde uma unidade fisicamente mais próxima do controlador terá prioridade sobre uma unidade mais distante.

Uma vantagem óbvia de tal esquema é a simplicidade. Poucas linhas de controle são necessárias e o número delas é independente do número de dispositivos; portanto, dispositivos adicionais podem ser introduzidos simplesmente conectando-os ao barramento.

Uma desvantagem do esquema daisy-chain é a sua susceptibilidade a falhas. Se uma falha ocorre no circuito de utilização do barramento de um dispositivo, ele pode privar os dispositivos subsequentes de terem acesso ao barramento ou pode ainda permitir que mais de um dispositivo tenha acesso simultâneo ao barramento. Uma falha de alimentação de um dispositivo ou a necessidade de desativá-lo pode tambem levar a problemas com o método daisy-chain. Entretanto, a lógica envolvida é muito

simples.

Outra desvantagem é a estrutura de prioridade fixa. Os dispositivos mais próximos do controlador sempre recebem primeiro o acesso ao barramento. Se estes dispositivos tem alta demanda pelo barramento, os dispositivos mais distantes ficam sem chances. Uma vez que o sinal de utilização do barramento deve se propagar sequencialmente através dos dispositivos, este mecanismo pode tambem ser muito lento.

.Polling(Fig.2b)- Há um sequenciamento das oportunidades de uso do barramento igualmente por todos os processadores. É um sistema cíclico, onde o controlador oferece a cada processador uma oportunidade, um após o outro.

O sequenciamento pode tambem recomeçar do zero após atender um dispositivo. Neste caso, estabelece o mesmo tipo de prioridade que no daisy-chain, mas as prioridades não precisam serem fixas porque a sequência pode ser mudada facilmente. Este esquema não sofre dos problemas de confiabilidade e localização do daisy-chain, mas o número de dispositivos é limitado ao número de linhas polling.

.Independent request(Fig.2c)- Cada processador tem sua linha de solicitação independente e qualquer estrutura(inclusive adaptativa) de decisão pode ser implementada. É tambem possível desabilitar um dispositivo que se conheça ou se suspeite ter falhas. É neste mecanismo de decisão que pode ser implementada a disciplina de fila FCFS que analizaremos ou uma disciplina de prioridade qualquer.

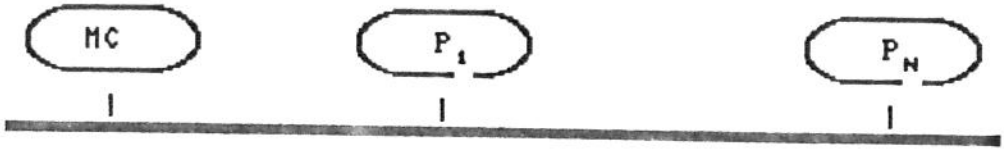
A maior desvantagem do independent request é o número de

linhas e conectores requeridos para o controle. É claro, a complexidade dos algoritmos de alocação permitidos por este esquema se reflete na quantidade de hardware do controlador de barramento.

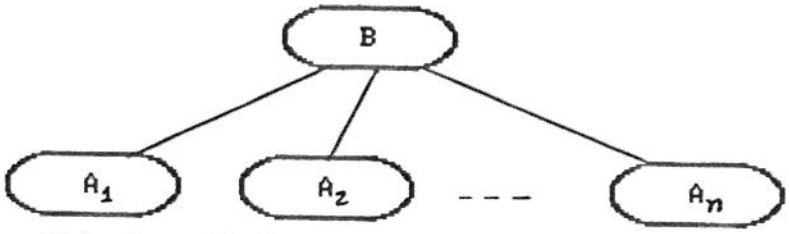
Evidentemente cada uma destas formas de decidir o conflito influencia de maneira diferente a comunicação entre os processadores. A fig.15, seção 3.3, mostra claramente este fato.

Nos sistemas multiprocessadores que compartilham recursos para o acesso a variáveis globais, podemos ter dois modos de operação. Na operação assíncrona, um processador, quando de posse do recurso, o retém pelo tempo necessário para transmitir toda sua mensagem. Na operação síncrona, quando de posse do recurso, um processador só transmite um tamanho fixo de mensagem. Assim, para transmitir uma mensagem maior deve fazer vários acessos. Todos os modelos aqui expostos serão de operação assíncrona. Vale alertar que é importante não confundir operação síncrona do sistema de processadores com exigências de sincronismos do sistema de tarefas.

No funcionamento de qualquer computador, tempos são consumidos pelo sistema operacional em inicialização e gerenciamento. Nos multiprocessadores devemos citar ainda os tempos de disciplinamento(scheduling). Como veremos nossos modelos não incluem qualquer destes fatores.

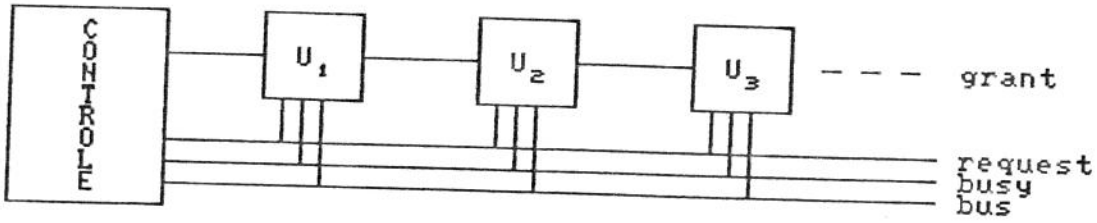


(a) Arquitetura por barramento

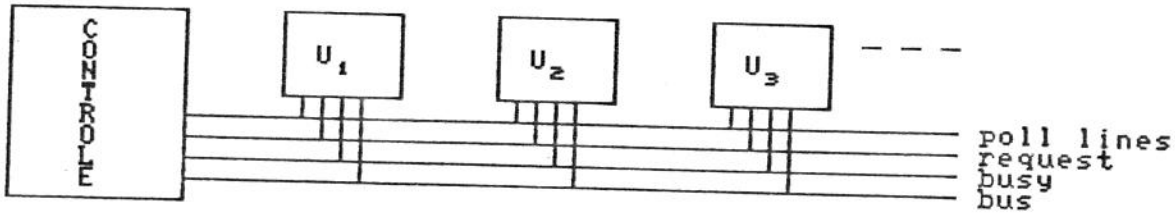


(b) Arquitetura radial

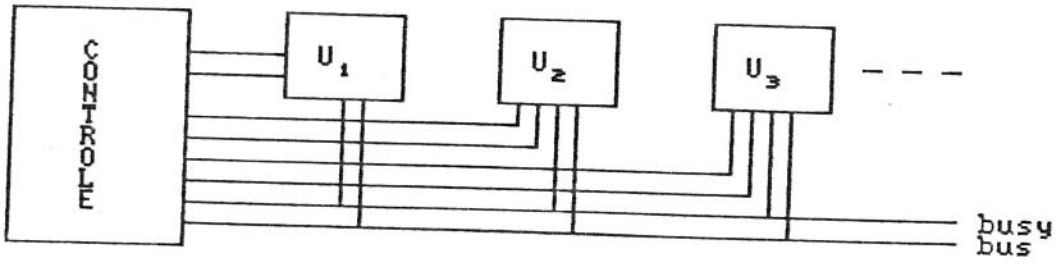
Fig. 1



(a) Daisy-Chain



(b) Polling



(c) Independet request

Fig. 2

2.2. REPRESENTAÇÃO DA CARGA DE TRABALHO

A representação da carga de trabalho é um fator importante no estudo do desempenho de sistemas de computadores, pois o desempenho do sistema está diretamente relacionado ao tipo de carga de trabalho 'manuseada'.

Nas aplicações gerais de processamento de dados, a representação da carga de trabalho é um problema complexo, pois elaborar um modelo que leve em consideração interdependência entre tarefas, não estacionariedade e contenção por recursos na execução de tarefas concorrentes é extremamente difícil. Adotam-se então hipóteses de geração de acessos segundo processos de Poisson de mesma taxa, independentes e uniformemente distribuidos entre os processadores. Estes modelos são conhecidos na literatura como modelos de referência uniforme[03].

Nas aplicações dedicadas, a carga de trabalho usualmente é um conjunto fixo de tarefas que são executadas em uma ordem pré-estabelecida e a intervalos regulares. Deste modo, as interdependências física e lógica são mais tratáveis. Evidentemente, o estudo de desempenho nas aplicações dedicadas leva a uma certa compreensão do problema de desempenho em aplicações gerais através do estudo de casos ou interdependências entre tarefas típicas.

A carga de trabalho nas aplicações dedicadas será representada por um grafo composto de nós e arcos orientados. Nesta notação, os nós representam as tarefas consumindo e produzindo dados e arcos orientados a relação de precedência e o fluxo de dados entre as tarefas. Os arcos são dirigidos da fonte para o sumidouro. Portanto, um grafo de computação representa as

necessidades de sincronização e comunicação entre as tarefas.

A seguir definimos cinco tipos básicos de relação de precedência com suas notações gráfica e simbólica. Os símbolos Fa e Fb são usados para indicar duas tarefas distintas e (Fi) e (Fj) são usados para indicar dois conjuntos distintos de tarefas que podem ou não ter algum elemento comum[12].

1) Sequencial(Fig.3a): Uma relação sequencial S:Fa-->Fb especifica que em qualquer execução destas duas tarefas, a tarefa Fa deve ser completada antes que a tarefa Fb possa ter início. Após completar Fa, Fb é executada.

2) If-Then-Else(Fig.3b): Uma relação if-then-else IF: Fa-->(Fi) especifica que em qualquer execução destas tarefas, a tarefa Fa deve ser completada antes que alguma das tarefas em (Fi) possa ter início. Após o término da tarefa Fa, uma e somente uma das tarefas em (Fi) é selecionada de acordo com algum critério de seleção.

3) Merge(Fig.3c): Uma relação merge M: (Fi)-->Fb especifica que em qualquer execução destas tarefas, alguma das tarefas em (Fi) deve ser completada antes que Fb passa ter início. Supõe-se aqui que numa execução, somente uma das tarefas em (Fi) estará realmente sendo executada. Após completar a execução desta tarefa, Fb é executada.

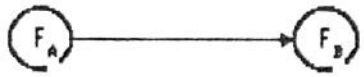
4) Fork(Fig.3d): Uma relação fork F: Fa-->(Fi) especifica que em qualquer execução destas tarefas, a tarefa Fa deve ser completada antes que as tarefas em (Fi) possam ter início. Após completar Fa, todas as tarefas em (Fi) são necessariamente executadas.

5) Join(Fig.3e): Uma relação join J: (Fi)-->Fb especifica que em

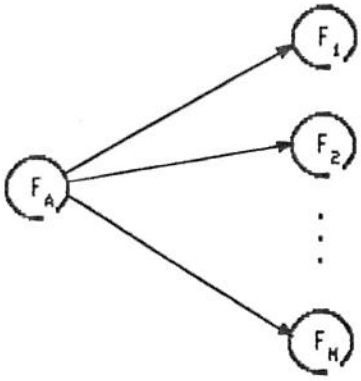
qualquer execução destas tarefas, todas as tarefas em {Fi} devem ser completadas antes que Fb possa ter início. Supõe-se aqui que numa execução todas as tarefas em {Fi} estarão sendo executadas. A ordem na qual as tarefas de {Fi} se completam não é relevante. Somente após o término das tarefas em {Fi}, a tarefa Fb é executada.

Combinando estas cinco estruturas básicas, podemos construir o grafo de computação estrutural de um algoritmo. É o diagrama de fluxo de computação e mostra as dependências entre variáveis e as operações sobre variáveis requeridas na execução do algoritmo. Os grafos (algoritmos) podem ser cíclicos ou acíclicos. Um grafo é acíclico se toda tarefa que o compõem só é executada uma única vez. Caso contrário, o grafo é cíclico.

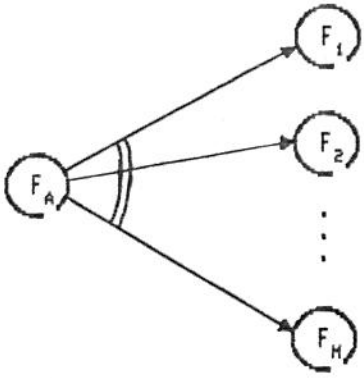
Um estudo de desempenho analisa a eficiência do mapeamento de diagramas de fluxo de computação sobre uma rede de elementos de processamento. Se a arquitetura de processadores for representada, de forma similar, por um grafo com cada nó representando um processador com sua memória local e os arcos representando as interconexões, então o problema de mapeamento torna-se um problema de imergir um grafo no outro. O efeito da alocação processador-tarefa sobre o desempenho pode ser estudado escolhendo diferentes mapeamentos. Para interconexões diferentes pode ser desejável escolher mapeamentos diferentes. A principal dificuldade em aplicações gerais é que este é um problema dinâmico devido ao carater dinâmico do grafo de fluxo de computação.



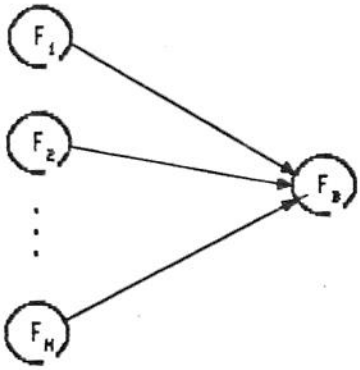
(a) Sequencial



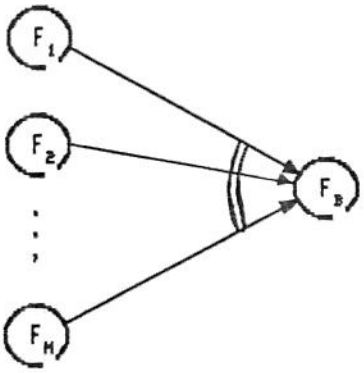
(b) If-then-else



(c) Fork



(d) Merge



(e) Join

Fig. 3

2.3. MODO DE OPERAÇÃO

Vamos considerar neste estudo a carga de trabalho oferecida por estruturas de computação com ciclos de execução hierárquicos sincronizados. Especificamente, o grafo estrutural de um algoritmo hierárquico sincronizado em dois níveis está mostrada na fig.5c. Ele se caracteriza por uma tarefa B de coordenação no nível superior e um grupo de tarefas Ai, i=1...n, no nível inferior. Tarefas do grupo A trocam dados e mensagens com a tarefa B, mas não se comunicam entre si. Um ciclo se inicia com o processamento da tarefa B. Após concluida, a tarefa B passa dados e dá início a todas as tarefas Ai. Cada tarefa Ai, após concluida, passa dados à tarefa B e um novo ciclo se inicia. A tarefa B só pode ser iniciada após receber dados de todas as tarefas Ai do ciclo anterior. Observa-se assim que são algoritmos sincronizados, sendo a tarefa B um ponto de forte sincronização.

Para fixar idéias, vamos descrever duas aplicações que se caracterizam por ciclos de execução hierárquicos sincronizados. Esta estrutura surge frequentemente em muitas áreas de aplicação e será tomada como carga de trabalho nas duas arquiteturas descritas.

Em computação científica, uma classe de algoritmos paralelos sincronizados que surge cobrindo uma ampla gama de aplicações é a classe dos algoritmos hierárquicos sincronizados em dois níveis[02]. Esta classe de algoritmos cobre aplicações tais como controle de plantas industriais e robôs, sistemas de equações algébricas lineares, sistemas de equações diferenciais parciais, algoritmos de otimização, simulação, processamento de voz e imagem, etc.[1] Estes algoritmos se originam usualmente da

aplicação de uma técnica de decomposição/coordenação na solução de problemas relacionados com grandes sistemas. A técnica consiste em dividir o domínio da solução em subdomínios. A busca das soluções parciais nos subdomínios é feita por tarefas independentes Ai e o requisito de que as soluções parciais componham uma solução global é controlado por uma tarefa de coordenação B. A necessidade de coordenação surge para considerar o acoplamento entre as tarefas Ai.

Considere agora uma aplicação no controle de processos industriais(fig.4). Num primeiro estágio, processadores independentes fazem aquisição e processamento de dados de sensores. Os dados de saida destas tarefas independentes são transmitidos ao processador do segundo estágio. Este processador analisa as informações recebidas de todos os processadores do primeiro estágio para decidir as ações de controle. Então, informa de volta, para atuação pelos processadores do primeiro estágio. Observe que só de posse dos dados de todos os processadores do primeiro estágio é que o processador do segundo estágio pode iniciar seu processamento. Cada aplicação exige do sistema de computação um tempo máximo(tempo crítico) no qual um ciclo destes deve se realizar para operação adequada do processo. Evidentemente, dependendo das exigências da aplicação, a tarefa B deste exemplo pode ser um algoritmo hierárquico sincronizado de controle ou otimização onde se esteja usando um sistema multiprocessador na obtenção da trajetória ótima.

Nesta classe de aplicações, de uma forma geral, um processo tem um ponto de operação que pode ser definido por um ponto no

espaço de estados do sistema. Sob a ação de um distúrbio externo o estado do processo é deslocado para além de uma vizinhança do ponto original. O objetivo do sistema de controle é calcular uma trajetória ótima através da qual possa trazer o processo de volta ao ponto de operação. O estado para o qual o processo se desloca sob a ação do distúrbio é a condição inicial para o cálculo da trajetória. Para atender exigências de tempo crítico, o processamento paralelo dos algoritmos tem sido proposto. Vários exemplo são encontrados no estudo de sistemas de grande porte[32].

Aos processadores que executam as tarefas Ai chamaremos de processadores A e ao que executa a tarefa B de processador B.

O modelo que utilizaremos neste trabalho considera tempos de execução de tarefas como variáveis aleatórias. Para ilustrar o caráter estocástico do tempo de execução, considere o algoritmo de controle hierárquico. Em geral o número de iterações necessárias para convergência de um algoritmo destes é dependente da condição inicial. Com referência ao controle industrial discutido antes, a condição inicial é composta pelos vetores de dados passados do primeiro para o segundo estágio por todos os processadores. E esta condição é função de distúrbios aleatórios incidentes sobre o sistema. Outros fatores que tornam os tempos de execução aleatórios são recursividade e testes de convergência. Assim, de uma forma geral, tanto o tempo de execução de B(fig.5c) como os tempos de execução dos Ais serão aleatórios. Nestes casos, dizemos que as tarefas tem tempo de execução dependentes dos dados. No exemplo, os dados são a condição inicial. Deste ponto de vista é portanto necessária uma

análise probabilística para obtenção do índice de desempenho. Assim, qualquer análise comparativa como a que propomos, só fará sentido em termos estatísticos, ou seja, valores médios e medidas de desvio ou intervalos de confiança.

Para uma representação desta carga de trabalho como um grafo de fluxo de computação considere a fig.5. A fig.5a mostra as estruturas básicas fork e join. Suponha que o conjunto de tarefas (Fi) do grafo fork seja o mesmo conjunto (Fj) do grafo join. Então temos a fig.5b. Se também a tarefa Fa é a mesma tarefa Fb, então temos o grafo cíclico(fig.5c). Cada ciclo representa uma iteração da estrutura hierárquica.

O modelo de operação empregado é assíncrono e estocástico. O modelo usa esta representação para a carga de trabalho e assume que os tempos de processamento de cada uma das tarefas, os tempos de comunicação e o número de iterações para convergência são variáveis aleatórias de distribuição conhecidas. O modelo não considera inicialização, gerenciamento pelo sistema operacional, disciplinamento (scheduling) e dependência entre as tarefas.

Um problema que surge neste contexto é, uma vez definida a aplicação e o algoritmo particular a ser usado, projetar um experimento para estimação das distribuições de probabilidade do tempo de execução de B, dos Ais e do número de iterações para convergência. Neste trabalho, já supomos disponíveis as distribuições de probabilidade dos tempos de execução e comunicação.

Neste ponto, para destacar a afirmação anterior de que o desempenho é função tanto da arquitetura do sistema(física)

quanto da estrutura das tarefas(lógica), considere o seguinte: a estrutura lógica deste sistema de tarefas impõe que um dos processadores (o que executa a tarefa B) assuma um papel de coordenador, independentemente de existir ou não na arquitetura uma hierarquia inerente entre os processadores. Por exemplo, na arquitetura por barramento não há, a princípio, qualquer diferença de status entre os processadores. No entanto, ao alocar a tarefa B a um deles, todos os outros passam a trabalhar na sua dependência.

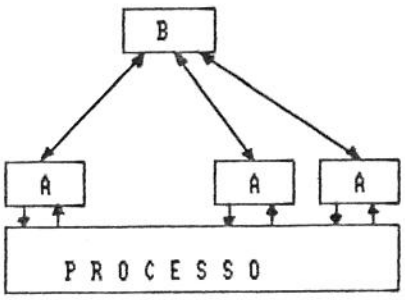


Fig. 4 - Controle de processos industriais

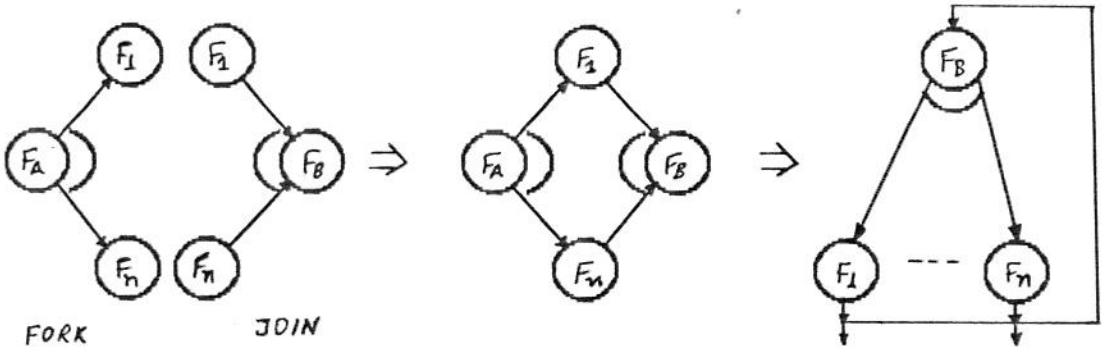


Fig. 5 - Representacao da carga de trabalho gerada por um algoritmo hierarquico com dois niveis

2.4. ÍNDICE DE DESEMPENHO

A definição do índice de desempenho não é uma questão trivial. Em se tratando da execução paralela de um algoritmo, gostaríamos de ter um índice que relacionasse o tempo para o melhor algoritmo rodando num único processador com o tempo para o algoritmo paralelo. No entanto, toda a literatura sobre o assunto foge desta relação exatamente por não existir consenso sobre o melhor algoritmo rodando num processador. Nesta questão, entra inclusive a eficiência do código empregado. É mais fácil então considerar a relação do tempo de uma implementação do algoritmo paralelo em um processador com o tempo do mesmo algoritmo rodando na máquina paralela. Com isto mede-se os efeitos opostos da relação decomposição versus comunicação, já que, como tendência geral, uma maior decomposição leva a maiores requisitos de comunicação.

No caso de aplicações gerais em processamento de dados, os índices de desempenho mais adequados e comumente utilizados são os fatores de utilização(número médio de processadores em serviço, tempo médio em serviço de cada processador, etc.). Eles medem a ocupação média dos recursos e portanto a eficiência com que estão sendo utilizados. É basicamente uma medida de eficiência interna.

Nas aplicações de tempo real, por serem de tempo crítico, os parâmetros que influenciam diretamente o tempo de resposta do sistema são os mais relevantes para o estudo de desempenho. Na execução paralela de um sistema de tarefas, aos tempos de processamento somam-se os tempos de comunicação. Uma alta utilização do processador poderá não se traduzir em baixo tempo

de resposta se uma parte significativa do trabalho for dedicado à comunicação. Neste caso, os fatores de utilização não são os índices mais adequados. Com efeito, alta utilização dos processadores é uma condição necessária para diminuir o tempo de resposta, mas não suficiente: os processadores precisam realizar trabalho útil.

Os dois índices geralmente utilizados e que adotaremos são:

ganho de velocidade de execução = $$S(n) = \frac{E[T(1)]}{E[T(n)]} \qquad (1a)$$

eficiência do paralelismo = $$e(n) = \frac{S(n)}{n} \qquad (2a)$$

onde:

n = número de processadores

E[T(1)]= valor esperado do tempo de execução em um processador

E[T(n)]= valor esperado do tempo de execução em n processadores

Para a carga de trabalho específica descrita nas seções 2.2 e 2.3, se o número de ciclos na execução total não se altera com o número de processadores n, as relações (1a) e (2a) se mantém para cada ciclo. Assim podemos definir:

ganho de velocidade no ciclo $$S_c(n) = \frac{E[T_c(1)]}{E[T_c(n)]} \qquad (1b)$$

eficiência no ciclo $$e_c(n) = \frac{S_c(n)}{n} \qquad (2b)$$

O problema de variação do número de ciclos com n é um tema mais específico e será abordado no capítulo 4. Até então

trabalharemos com figuras de desempenho para o ciclo.

O valor E[Tc(n)] pode ser visto se constituir de duas parcelas: uma devida a tempos de processamento e possíveis efeitos de sincronização, E[Tp(n)], e outra devida à comunicação, E[Tcom(n)]. Ou seja,

$$E[T_c(n)] = E[T_p(n)] + E[T_{com}(n)]$$

Expressando E[Tcom(n)] como um aumento relativo em E[Tp(n)], através de E[ΔTp(n)], onde E[Tp(n)] = E[Tcom(n)]/E[Tp(n)], temos,

$$E[T_c(n)] = E[T_p(n)].(1 + E[\Delta T_p(n)])$$

Definimos a parcela E[ΔTp(n)] como o "adicional de comunicação"(overhead). O valor de E[Tp(n)] é obtido analíticamente na seção 3.1. Na seção 3.2 desenvolvemos um modelo para determinar E[ΔTp(n)] na arquitetura radial. Na seção 3.3 E[ΔTp(n)] na arquitetura por barramento é obtido por simulação. O efeito de E[ΔTp(n)] é reduzir tanto s(n) quanto e(n). Veremos que a eficiência decresce rapidamente se o número de processadores aumenta alem de um certo valor, que depende do tamanho do problema e do tempo relativo de comunicação.

Reescrevedo as relações (1b) e (2b) em função de E[Tp(n)], temos:

$$s(n) = S_c(n) = \frac{E[T_c(1)]}{E[T_p(n)][1 + E[\Delta T_p(n)]]} \qquad (1)$$

$$e(n) = e_c(n) = \frac{S_c(n)}{n} \qquad (2)$$

CAPÍTULO 3: MODELOS DE ANÁLISE

3.1. MODELO SIMPLIFICADO

O caso onde não se considera o tempo de comunicação será analisado inicialmente para estabelecer o valor de E[Tp(n)] em (4) e também para ganhar melhor compreensão sobre o problema. Neste caso E[ΔTp(n)] = 0. Isto corresponde a situação em que os processadores teriam todos os registros de dados disponíveis para acesso imediato na execução do algoritmo. Ou seja, a transferência de B para os Ais é instantânea e simultânea e as transferências dos Ais para B são tambem instantâneas. Nesta situação o procedimento da análise que se segue é válido para ambas arquiteturas e quaisquer distribuições de probabilidade para tempos de processamento e comunicação.

A fig.6 mostra o modo de execução desta classe de algoritmos e os tempos envolvidos, para o caso de tempo de comunicação zero: após executar a tarefa B, o processador transfere dados e dá partida na execução das tarefas (A_i), i=1..n, em paralelo. Cada processador Ai ao concluir a sua tarefa, transfere dados ao processador B. Este processador só pode iniciar sua tarefa após receber dados de todos os Ais, quando então um novo ciclo se inicia. Um ciclo é composto de dois períodos: um período A, de duração Ta, e um período B de duração Tb. Tc é a duração do ciclo. O período A é determinado pela tarefa de maior duração neste período. Todas as tarefas tem tempo de execução aleatórios com distribuição conhecidas. Observe que neste diagrama não foram incluídos os tempos de comunicação. Sejam Fa(t) e Fb(t) as distribuições de probabilidade de Ta e Tb, respectivamente.

Para esta situação, o efeito da comunicação não se manifesta, e o tempo de execução é determinado por:

- o tempo de ciclo Tc é a soma de Ta e Tb , cuja distribuição é obtida pela convolução das duas distribuições independentes,

$$Fc(t) = Fa(t) * Fb(t) \qquad (4)$$

Sendo conhecida a distribuição de Tb, Fb(t), o problema que se coloca é o de obter a distribuição de Ta, Fa(t), a partir das distribuições dos (Tai), Fai(t).

- como $T_A = \max (Ta_1, Ta_2, \ldots, Ta_n)$ e (Ta_i) é uma coleção de variáveis aleatórias independentes com distribuição $(Fa_i(t))$, segue-se que

$$\begin{aligned} Fa(t) = P(T_A \leqslant t) &= \max \{ Fa_1(t), Fa_2(t), \ldots, Fa_n(t) \} \\ &= P(Ta_1 \leqslant t).P(Ta_2 \leqslant t)\ldots P(Ta_n \leqslant t). \\ &= \prod_{i=1}^{n} Fai(t) \end{aligned} \qquad (5)$$

- a distribuição do tempo total T de uma corrida do algoritmo com um número N de ciclos será então,

$$F_T(t) = \underbrace{Fc(t)*Fc(t)*\ \ldots\ *Fc(t)}_{N \text{ vezes}} \qquad (6)$$

e sua média e variância podem ser obtidas através de:

$$E[T] = N.E[Tc] = N.E[Ta] + N.E[Tb] \qquad (7)$$

$$Var[T] = N^2.Var[Tc] = N^2.Var[Ta] + N^2.Var[Tb] \qquad (8)$$

A distribuição do tempo de execução do algoritmo é imediata tendo-se a distribuição do número de ciclos. Sendo o número de ciclos N uma v.a., o tempo médio de execução do algoritmo, considerando independência entre os ciclos será,

$$E[T] = E[N].E[Tc] \qquad (9)$$

e a variância,

$$Var[NTc] = Var[N]Var[Tc]+E[N]^2Var[Tc] +E[Tc]^2Var[N] \qquad (10)$$

No que se segue nos preocupamos especialmente com Fa(t), E[Ta], e Var[Ta], distribuição, média e variância dos períodos A, respectivamente, já que os argumentos acima associados a Tc, podem ser obtidos através da convolução (4).

Exemplo: Distribuição exponencial.

Suponha que todos os tempos de processamento {Tai} são exponencialmente distribuidos com taxa uniforme α. Então temos de (5):

$$Fa(t) = \prod_{i=1}^{n} Fai(t) = (1 - e^{-\alpha t})^n$$

$$= \sum_{K=0}^{n} \binom{n}{K} (-1)^k \, e^{-\alpha k t}$$

A média e variância ficam

$$E[Ta] = 1/\alpha \sum_{K=1}^{n} 1/k \qquad (11)$$

$$Var[Ta] = 1/\alpha^2 \sum_{K=1}^{n} 1/k^2 \qquad (12)$$

A expressão (11) do valor médio será utilizada posteriormente para o caso de tempos de processamento e comunicação exponenciais para o cálculo de E[Tp(n)]. Ela estabelece uma referência para avaliarmos o efeito da comunicação no tempo de ciclo.

Índice de Desempenho 1: Distribuição exponencial

Vamos considerar os índices de desempenho s(n) e e(n) definidos em (1) e (2) quando $\{Ta_i\}$ são v.a. iid. exponencialmente distribuidas, assim como Tb tambem exponencial. Temos:

$$s_C(n) = \frac{E[T_C(1)]}{E[T_C(n)]} = \frac{E[T_B] + n.\,E[T_{a_i}]}{E[T_B] + E[T_{a_i}] \sum_{K=1}^{n} 1/K}$$

Normalizando esta equação para E[Tai] = 1, fica

$$s_C(n) = \frac{E[T_B] + n}{E[T_B] + \sum_{K=1}^{n} 1/K} \qquad (13)$$

Da fig.6 e pela estrutura que o algoritmo apresenta, a tarefa B é sempre executada em série independente de quantos processadores se adote no nível A. De forma que é assim importante observar a influência de E[Tb] sobre a eficiência que se pode obter.

Para estudar a influência da duração da tarefa B sobre os índices, os cálculos foram feitos para vários valores de E[tb] relativos a E[tai]. Os resultados estão mostrados na fig. 7.

Da fig.7(a) podemos ver a relação não linear de s(n) com n, o que se traduz numa diminuição da eficiência na utilização dos processadores. Tambem podemos observar que o efeito de aumentar E[tb] é uma redução em s(n) e e(n). Este efeito é maior para valores maiores de n (distância entre as curvas s(n)).

Índice de Desempenho 2: Distribuição constante

As relações (13) para o caso em que os períodos de duração são constantes ficam,

$$s_C(n) = \frac{E[T_B] + n}{E[T_B] + 1} \qquad (14)$$

fig.8 mostra curvas de s(n) para as distribuições constante e exponencial num mesmo gráfico. O caso de distribuição constante

tem índices superiores e taxas de crescimento maiores com n. Isto é um efeito direto do coeficiente de variação (Cv= $\sigma_x / E[x]$) da distribuição.

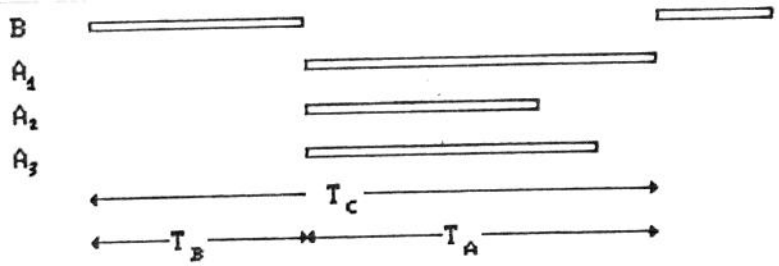


Fig.6 - Um ciclo de execução idealizado

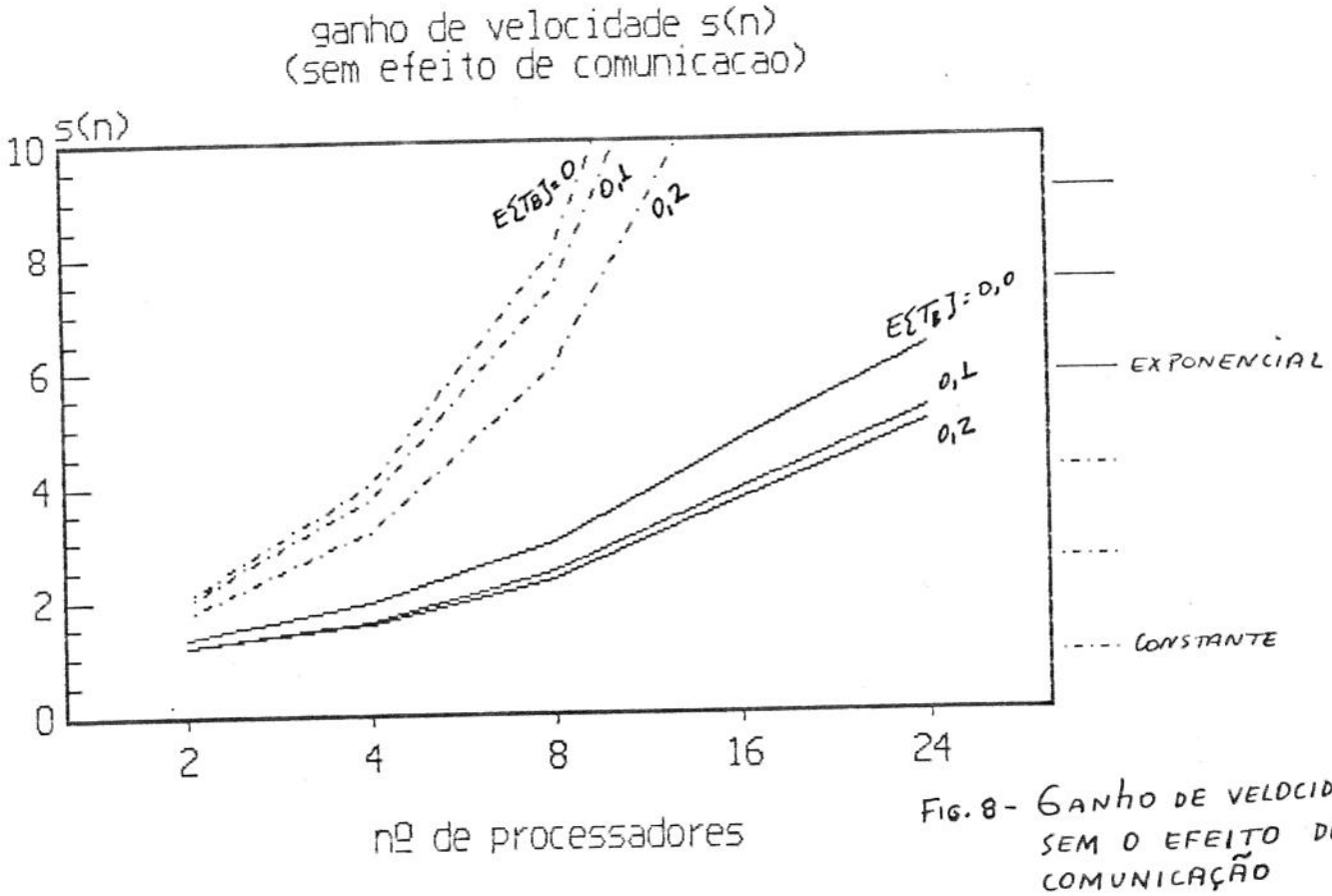


Fig. 8 - Ganho de velocidade sem o efeito de comunicação

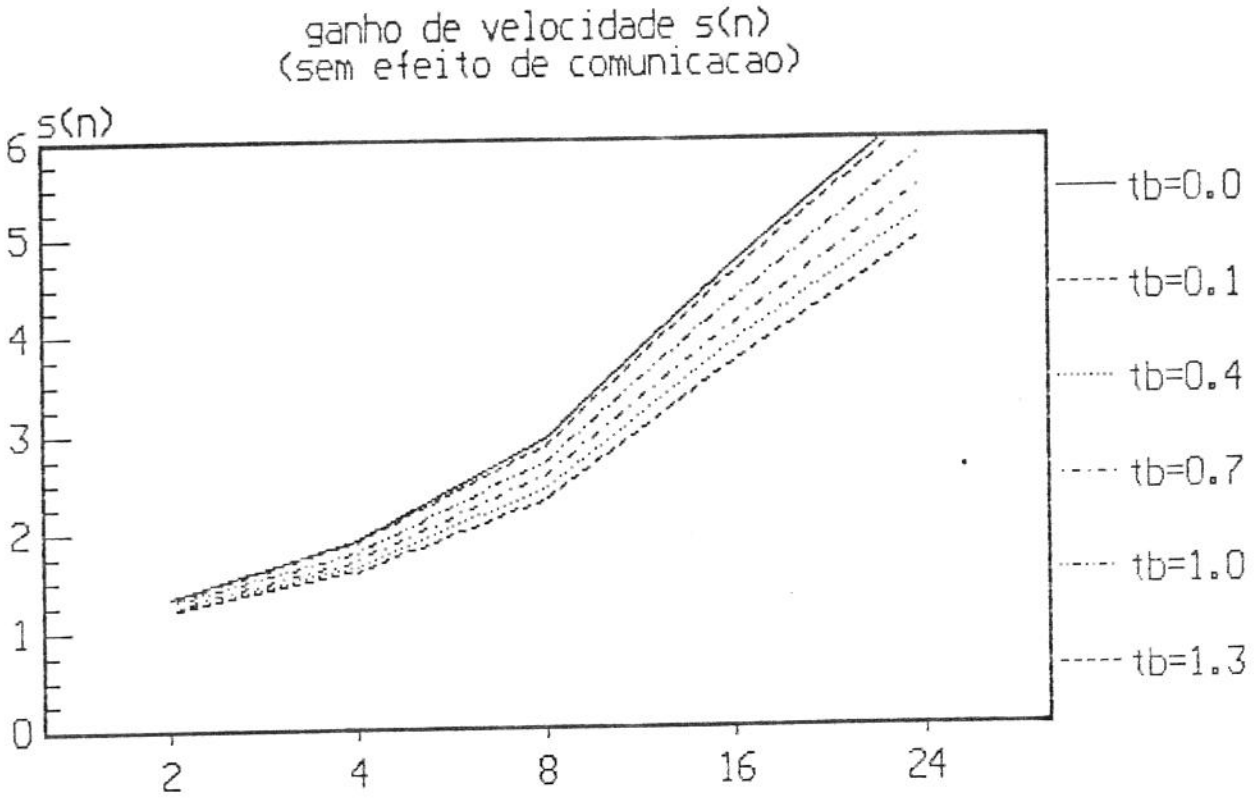
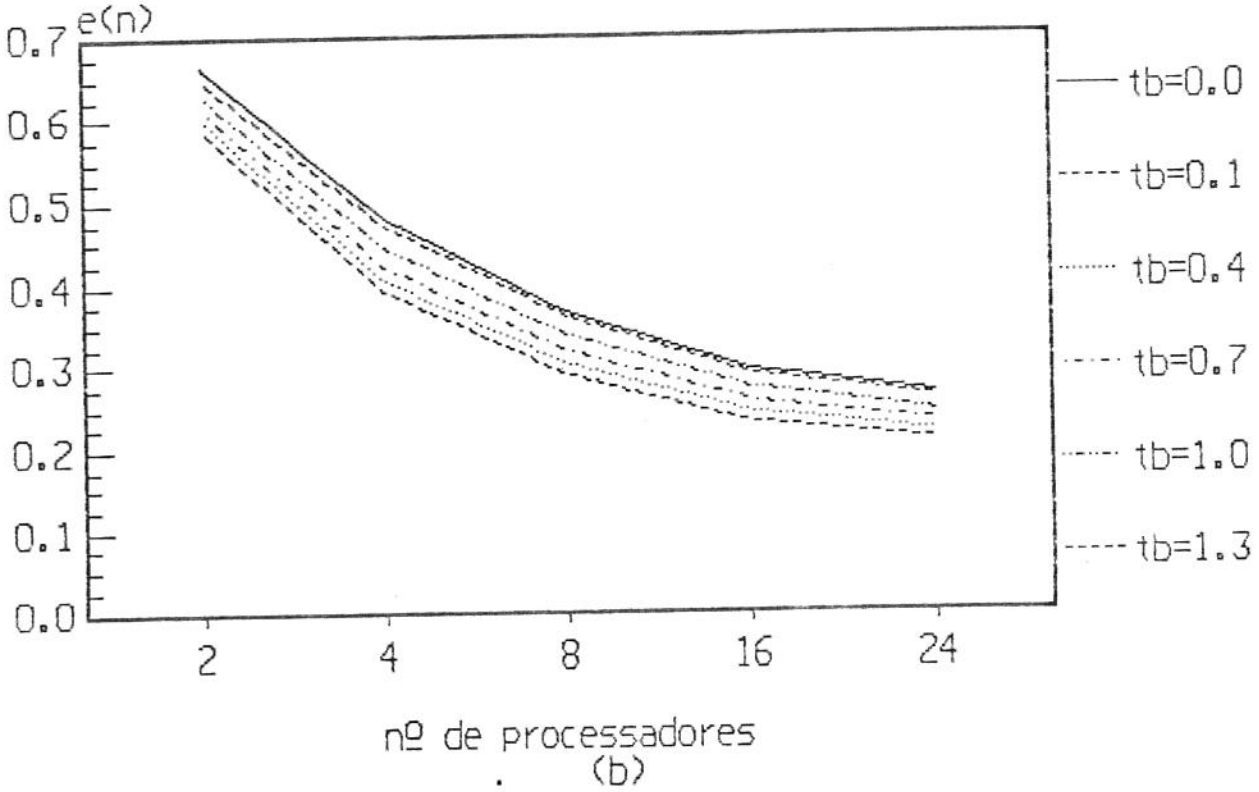


Fig.7 - GANHO DE VELOCIDADE E EFICIENCIA PARA DISTRIBUIÇÃO EXPONENCIAL DOS TEMPOS DE PROCESSAMENTO E TEMPOS DE COMUNICAÇÃO NULOS.



3.2. MODELO COM COMUNICACAO: ARQUITETURA RADIAL

Nesta seção desenvolvemos um modelo para avaliar o efeito da comunicação na arquitetura radial.

3.2.1. O MODELO

No item anterior, como a comunicação entre processadores era instantânea, a subtarefa de maior tempo de processamento determinava a duração do ciclo. A fig.9 mostra que considerando a comunicação isto não é verdade. A figura é para uma arquitetura radial, onde há comunicação direta entre os processadores do nível A e o processador B, como descrito na seção 2.1. Ela não cobre o caso de um barramento com comunicação via memória comum uma vez que neste caso cada comunicação envolve dois acessos(um para escrever o dado na MC e outro para ler da MC), o que não está representado na figura. Para o caso de barramento compartilhado, veja fig.15, seção 3.3.3 para as diversas disciplinas de decisão.

Podemos observar na figura 9 o efeito da comunicação e do bloqueio sobre a duração do ciclo.

Neste ponto fazemos as seguintes hipóteses:

1) A duração dos tempos de processamento e períodos de comunicação são variáveis aleatórias exponencialmente distribuidas e independentes;

2) Quando um processador deseja comunicar e há recurso disponível, a comunicação inicia-se imediatamente;

3) Após a comunicação ser concluída, o recurso utilizado é liberado imediatamente;

4) Se um processador deseja comunicar e não há recurso disponível, então ele espera ocioso até que o recurso esteja

livre.

As hipóteses 2 e 3 dizem que não há atraso na tomada ou liberação do recurso quando da comunicação. Isto equivale a não considerar os tempos de gerenciamento do sistema operacional(inicialização de periféricos, chamada de rotinas, arbitragem,etc).

Por uma descrição de estado adequadamente escolhida, a duração do ciclo pode ser modelado como uma Cadeia de Markov a Tempo Continuo.

Com as hipóteses assumidas reconhecemos cinco estados distintos para um processador Ai:

1. bloqueado para leitura
2. lendo
3. processando
4. bloqueado para escrita
5. escrevendo

e, verifica-se facilmente a validade do lema seguinte:

Lema 1. Seja E definido por

$$E = \{ p = (p_1,p_2,\ldots,p_n) : p_i \in [1,2,3,4,5],\ i = 1,2,\ldots,n \}$$

Se p_i indica o estado do processador i, então o processo $t \longrightarrow p(t) = (p_1(t),p_2(t), \ldots ,p_n(t))$ é uma Cadeia de Markov a Tempo Contínuo com espaço de estado E.

Este lema caracteriza o processo estocástico que descreve o modo de operação do sistema. A conveniência desta caracterização é que a distribuição do ciclo completo é obtida identificando um estado unicamente caracterizado que sinaliza o final do ciclo. Ou seja, (1,1,...,1).

Para obter a duração do período A a partir da CMTC do Lema 1 considere o seguinte. Vamos supor sem perda de generalidade que o processador 1 é o primeiro a receber dados através de comunicação no início do ciclo. Então o estado que caracteriza início de ciclo será Eo=(2,1,1,...,1). O estado final seria caracterizado por Ef=(1,1,1,...,1), indicando claramente o fim do ciclo dos processadores do nível A, já que no estado Ef todos os processadores Ai estão bloqueados para leitura, esperando que um novo ciclo se inicie. A distribuição da duração do período A pode ser obtida calculando a distribuição do tempo de passagem pelo estado Ef a partir do estado inicial Eo.

Infelizmente, com o estado descrito desta forma, a dimensão da CMTC explode($O(n^5)$), e torna qualquer análise impraticável mesmo para um pequeno número de processadores. No entanto, esta descrição é desnecessariamente detalhada, se as taxas de processamento e comunicação forem iguais para cada processador, uma vez que ela nos fornece o estado de cada processador separadamente, enquanto que para nossos propósitos basta saber quando o último processador terminou sua comunicação no ciclo. Ora, para isto basta reconhecer o número de processadores em cada estado e não quais os processadores se encontram em um estado. Isto nos leva a uma redefinição do vetor de estado que na verdade é uma agregação(lump) de estados em relação ao anterior.

Defina agora os seguintes elementos de um novo vetor de estado q:

q1= número de processadores A em comunicação ou bloqueio na direção de B para A (isto é, recebendo ou esperando receber dados de B);

q_2= número de processadores A processando

q_3= número de processadores A em comunicação ou bloqueio na direção de A para B (isto é, enviando ou esperando enviar dados à B).

Considere também a seguinte versão da hipótese 1:

1". A duração dos períodos de processamento e comunicação são v.a. exponencialmente distribuidas e independentes, com mesmas taxas.

Lema 2. Seja D definido por

$D = \{ q = (q_1,q_2,q_3) : q_i \in [0,1,2,\ldots,n], \ i = 1,2,3\}$.

Então o processo $t \longrightarrow q(t) = (q_1(t),q_2(t),q_3(t))$ é uma Cadeia de Markov a Tempo Contínuo com espaço de estados D.

Prova: Veja APENDICE 1.

Tomemos os diagramas de transição de estados para 2 e 4 processadores no nível A na fig.11. Nesta figura temos:

λ^{-1} = tempo médio de processamento das tarefas Ai

μ^{-1} = tempo médio de comunicação

λ^{-1} = tempo médio de duração da tarefa B

O diagrama representa a operação do algoritmo/arquitetura e por construção é uma CMTC. Utilizaremos esta cadeia para obter a duração do período A do ciclo. O ponto de entrada da tarefa B do segundo nível, ou seja, o ponto de sincronismo é representado pelo estado (0,0,0). O tempo de passagem pelo estado (0,0,0) a partir do estado inicial (n,0,0) é a duração do periodo A, T_A. Isto porque, em cada ciclo, quando a tarefa B é concluída, um período A se inicia no estado (n,0,0), ou seja, inicia-se a primeira comunicação de B para A. Quando o estado (0,0,0) é

atingido, significa que para aquele período A o último processador concluiu seu processamento e comunicação.

Lema 3. O número de estados da CMTC do Lema 2 é $(n+1)^2$.

Prova. Por inspeção, o diagrama de estados da CMTC pode ser desenhado com cada estado ocupando uma posição numa matriz (n+1)x(n+1) como nos dois exemplos (Fig.11). Daí o número de estados é $(n+1)^2$.

Lema 4. (i) O número de elementos da matriz geradora é $(n+1)^4$.

(ii) O número de elementos não nulos na matriz geradora da CMTC é $2n^2 + n + 1$.

Prova. A ordem da matriz geradora é $(n+1)^2 x (n+1)^2$, já que o número de estados é $(n+1)^2$. Cada transição possível no diagrama de estados é uma posição não nula na matriz. Também, todos os estados no interior do diagrama(Fig.11) tem 2 transições possíveis e todos os estados na fronteira do diagrama tem uma transição possível, com exceção dos da primeira coluna, onde n-1 estados tem 2 transições. Portanto,

$$2(n+1)^2 - 2(n+1) - (n-1) = 2n^2 + n + 1.$$

A fig.12 mostra a matriz-A geradora para 2 e 3 processadores no nível A.

3.2.2. SOLUCÃO COMPUTACIONAL

Como resultado do Lema 3, a dimensão da matriz geradora da CMTC que modela a evolução do sistema é demasiadamente grande apesar da redução considerável do número de estados obtida com a agregação proposta. Por exemplo, para 2 processadores A teremos uma matriz-A de dimensão 4x4 e para 4 processadores A de

dimensão 16x16.

Para o processo caracterizado pelo Lema 2:

$$P\,(\,q_{t+s} = q_j \;/\; q_t = q_i\,) = P_s(q_i\,,q_j\,),$$

é valido pois o processo é estacionário, onde q_i , q_j são dois estados quaisquer de D. A função $t \longrightarrow P_t\,(q_i\,,q_j\,)$ é uma função de transição; a matriz P_t , $t \geq 0$, de funções de transição $P_t(q_i\,,q_j\,)$, é chamada Função de Transição da CMTC [15].

Seja a matriz $A = (A(q_i\,,q_j\,))$ com

$$A(q_i\,,q_j\,) = \begin{cases} -\lambda(q_i\,) & , \text{ se } i=j \\ \lambda(q_i).Q(q_i\,,q_j\,) & , \text{ se } i\neq j \end{cases}$$

A a matriz geradora do processo.

Então P_t pode ser obtido de A pelas equações de Kolmogorov:

$$\dot{P}_t = A.P_t \quad , \quad Po = I \qquad (15)$$

ou

$$\dot{P}_t = P_t.A \quad , \quad Po = I \quad \text{(I=matriz identidade)}$$

cuja solução é

$$P_t = \exp[At] = \sum_{n=0}^{\infty} A^n t^n/n! \quad . \qquad (16)$$

Um vetor de probabilidades estacionárias é dado pelo seguinte limite, que sempre existe se a CMTC é homogênea e irredutível, e é independente do estado inicial:

$$p = \lim_{t\to\infty} P_t$$

Esta distribuição estacionária é dada pela solução da equação

$$pQ = 0 \quad , \quad P_i = 1 \qquad (17)$$

Com o modelo descrito, podemos obter a duração do período A de duas formas: a partir das probabilidades estacionárias ou integrando a equação de Kolmogorov.

1.Solução a partir das probabilidades estacionárias

Para ver como a duração do período A pode ser obtida a partir das probabilidades estacionárias, considere o diagrama de estados da fig.11 para dois processadores A. Seja: p o vetor de probabilidades estacionárias; p_i, um elemento de p referente ao estado i; e p_f o elemento de p referente ao estado (0,0,..,0). O tempo do período A é obtido a partir do seguinte argumento: 1. interpretamos pi como valores relativos de tempo médio que a CMTC permanece no estado i; isto é possível pois a cadeia é tambem ergódica; 2. se identificarmos um estado qualquer pelo qual a CMTC necessariamente passe em um ciclo, e tivermos o tempo médio de permanência naquele estado, então podemos obter o tempo médio em qualquer outro estado da CMTC ou grupo de estados, bastando tomar esse tempo conhecido como referência.

No caso do sistema de tarefas que estamos estudando, o o estado associado a tarefa B, na nossa notação p_f, serve perfeitamente a este propósito. Reconhecendo que, se o processo não está no período B, ele necessariamente estará no período A, as equações ficam:

$$E[T_A]/E[T_B] = (1 - p_f)/p_f$$

Como estamos interessados em tempos médios de A com relação a B, adotamos $E[Tb] = 1$ de forma que

$$E[T_A] = (1 - p_f)/p_f$$

Assim, o problema se resume em resolver (17) para a nossa CMTC, e dela extrair P_f.

O problema desta solução é a ordem da matriz A. Para 24 processadores no nível A, por exemplo, a ordem de A é 625x625. Os requisitos de memória e tempo de computação tornam-se elevados. Além disso, esta solução não faz uso da esparcidade da matriz A, como mostrado pelo Lema 4.

Para este trabalho, escolhemos obter o valor de $E[T_A]$ integrando a equação de Kolmogorov. Como veremos, a implementação computacional é mais simples.

II.Solução a partir da equação de Kolmogorov

Este método de solução se baseia no seguinte argumento. Seja Ta o instante de entrada do processo num estado qa. Se qa é um estado absorvente que marca o final do ciclo de operação dos processadores A, então

$$P_t(q_o,q_a) = P(Ta < t) \qquad \text{onde} \quad q_o = (n,0,0)$$

Como o estado inicial está fixado, a equação de kolmogorov nos dá

$$\dot{P}_t = P_t \cdot A\,, \quad P_o = q_o \tag{19}$$

A escolha apropriada para o problema é tornar o estado (0,0,0) absorvente, isto é q_a (0,0,0), e para tanto basta fazer $\lambda_B = 0$ (retirar o arco com λ_B) nos diagramas da fig.11. Isto equivale a tornar toda nula a linha correspondente ao estado (0,0,0) na matriz A. Assim, se integramos a equação (19), a coordenada de P_t correspondente ao estado (0,0,0) nos dá a distribuição acumulada da duração do ciclo, P(Ta < t). Desta

forma devemos obter o comportamento transitório da CMTC.

A fig.12 mostra a matriz-A geradora para 2 e 3 processadores A com tempo médio de processamento λ^{-1} e tempo médio de comunicação μ^{-1}, iguais para todos. Pelo Lema 4, o número de elementos diferentes de zero é $2n^2 + n + 1$, o que é muito menor que $(n+1)^4$. Tendo em vista requisitos de memória e também para evitar perda de tempo computacional com multiplicações por zero, desejamos um método de solução que preserve a esparcidade do problema. O cálculo de P_t através de métodos que utilizam séries de potência não tem esta característica uma vez que nas potências successivas da matriz-A, Eq.16, valores diferentes de zero tendem a migrar em direção a posições que anteriormente eram zeros. Optamos então pela solução numérica da equação de Kolmogorov[15], usando uma rotina Runge-Kutta de quarta ordem para obter $P(Ta \leqslant t)$.

Queremos obter E[Ta]. Ocorre que, se uma v.a. X tomar somente valores não negativos (caso de período de tempo), então [15,pg.110]:

$$E[X] = \int_0^\infty [1 - F_X(x)]dx = \int_0^\infty P(X > x)dx \qquad (20)$$

Este resultado é usado para calcular E[Ta] a partir da relação (19), permitindo que o programa acumule o valor da média a medida que faz a integração da equação diferencial. Neste procedimento numérico usamos a regra do trapézio.

Um algoritmo foi programado para que a matriz fosse montada pelo computador. Assim, ao programa se fornece apenas quatro

dados:

n = número de processadores A

h = passo de integração

λ = taxa relativa a distribuição do tempo de processamento

μ = taxa relativa a distribuição do tempo de comunicação

A partir destes dados, o programa monta a matriz geradora da CMTC (na estrutura de dados projetada), integra a equação de transição e calcula E[Ta]. O programa usa apenas $3(n+1)^2$ entradas reais para armazenar a matrix-A.

O aumento relativo no tempo do período A devido à comunicação E[Δ Tp(n)], como definido na seção 2.4, é dado por

$$E[\Delta T_p(n)] = E[Ta] / \sum_{K=1}^{n} 1/k \quad - 1$$

onde $\sum_{K=1}^{n} 1/K$ dado pela Eq.11, com α =1.0, é a duração do ciclo sem comunicação, tomado como referência.

O programa foi executado num computador PC-AT-Compatível, usando co-processador aritmético e relógio de 10MHz e linguagem PASCAL. Nestas condições, cada curva da fig.13 foi construida em aproximadamente 1 hora.

3.2.3. Tempo de comunicação constante

A operação com tempos de processamento exponencial e tempos de comunicação constante é de muito interesse, como veremos no cap.4 Evidentemente que nesta situação as hipóteses markovianas que permitiram o modelo anterior ficam totalmente destruidas. Como veremos na seção 3.3., esta situação será analisada por técnicas de simulação. No entanto, os resultados são apresentados

aqui para efeito de comparação.

3.2.4. RESULTADOS

Os resultados estão mostrados nas tabelas I e II e fig.13 Estas figuras mostram também os resultados para tempo de comunicação constante, com valor idêntico a média exponencial, obtido por simulação como explicado na seção 3.3. As curvas para os dois casos foram desenhadas sobre um mesmo sistema de eixos para comparação.

Os resultados estão parametrizados por pelo fator ρ definido como a relação entre o tempo médio de comunicação e o tempo médio de processamento para um processador A, ou seja,

$$\rho = E[ta(n)]/E[tp(n)].$$

Observamos que de uma forma geral $E[\Delta Tp(n)]$ é maior para comunicação com duração exponencial que para duração constante com mesma média.

Na fig.14 temos as curvas para s(n)[Eq.01] e e(n)[Eq.02] considerando-se os efeitos da comunicação para o caso totalmente exponencial. Os valores de $E[\Delta Tp(n)]$ são obtidos da fig.13 e usados na equação 13. No mesmo gráfico estão as curvas para o caso sem comunicação para comparação.

Como estamos analisando aqui somente o acréscimo no período A do ciclo, tomamos $E[T_B] = 0$.

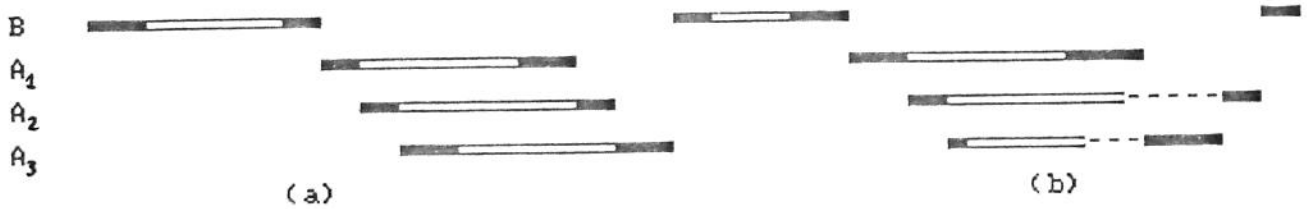


Fig.9 - Ciclos de execucao com comunicacao
(a) Um ciclo sem bloqueio
(b) Um ciclo com bloqueio

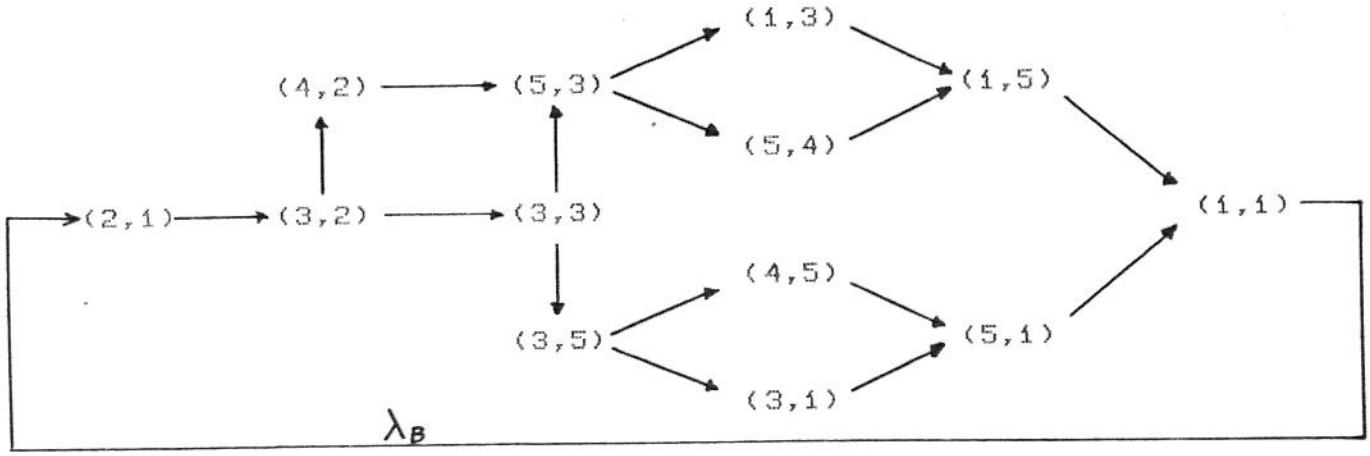


Fig. 10 - Diagrama de estados para a CMTC do Lema 1.

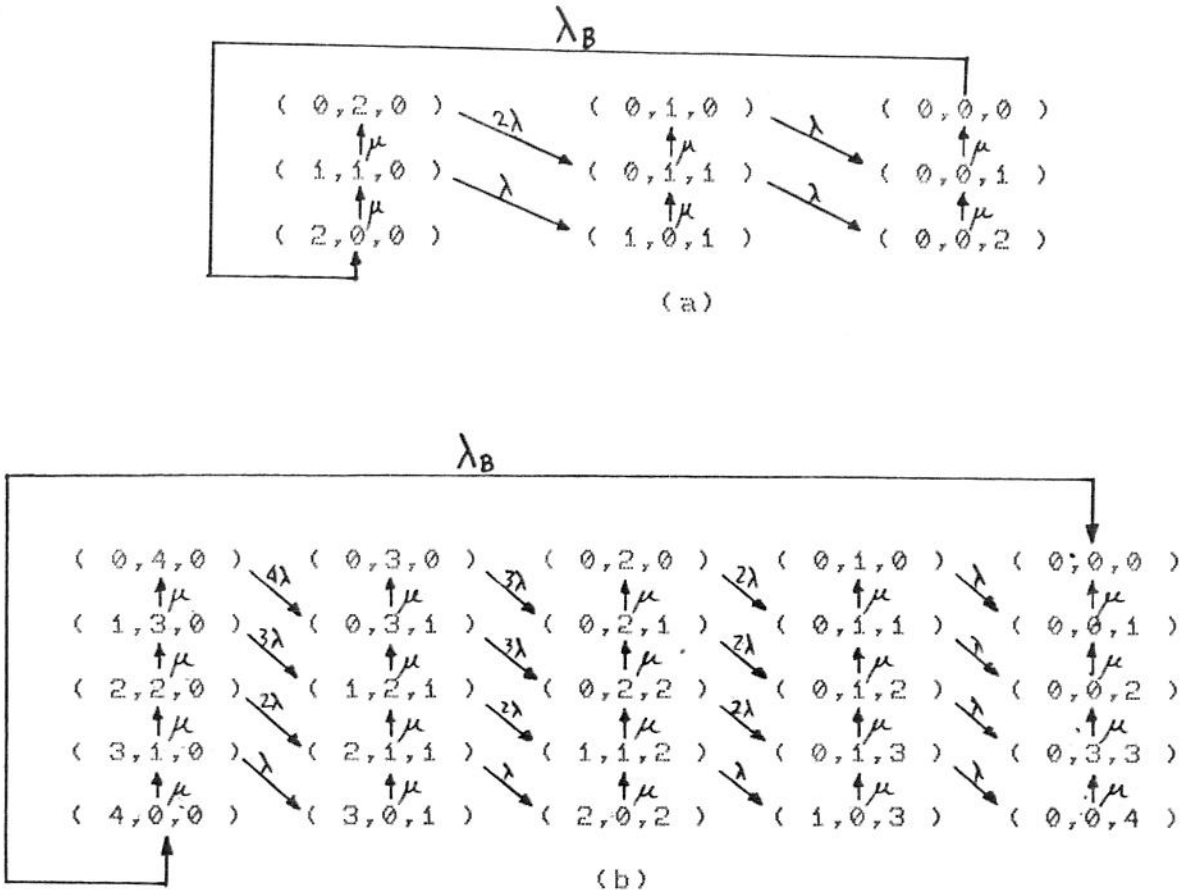
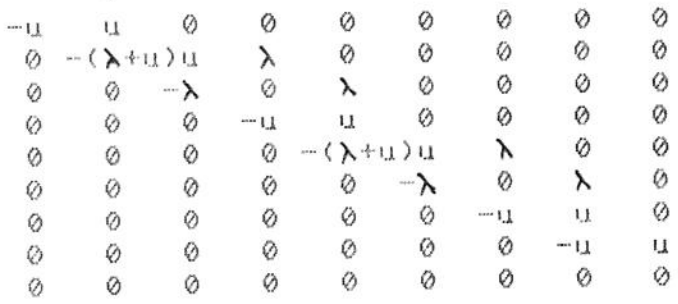
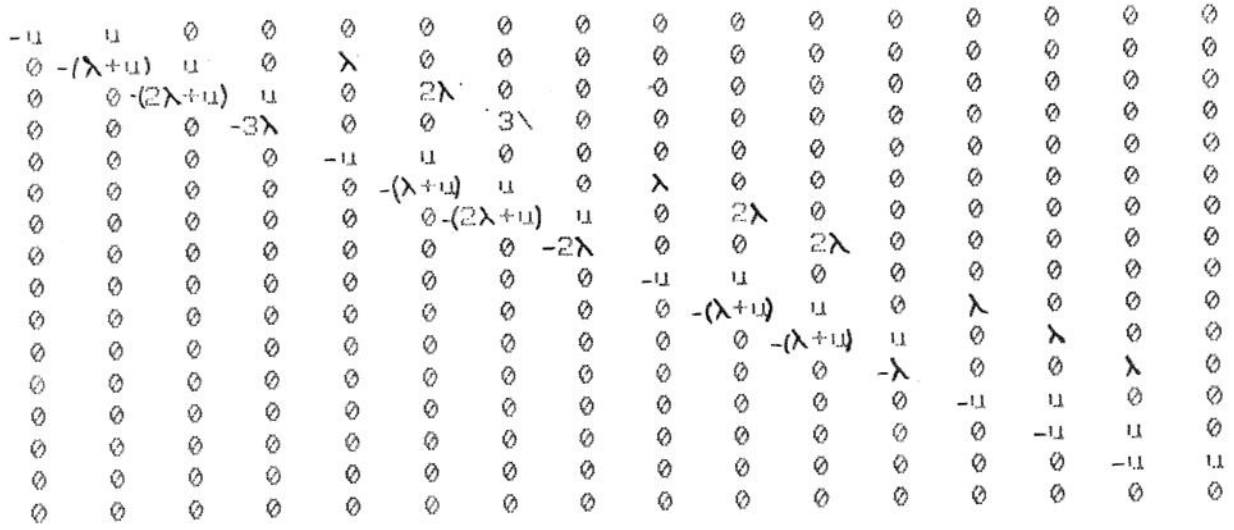


Fig.11 - (a) Diagrama de transição p/ 2 processadores
(b) Diagrama de transição p/ 4 processadores



(a) Sistema com 2 processadores



(b) Sistema com 3 processadores

Fig.12 - Matriz-A de taxas de transição

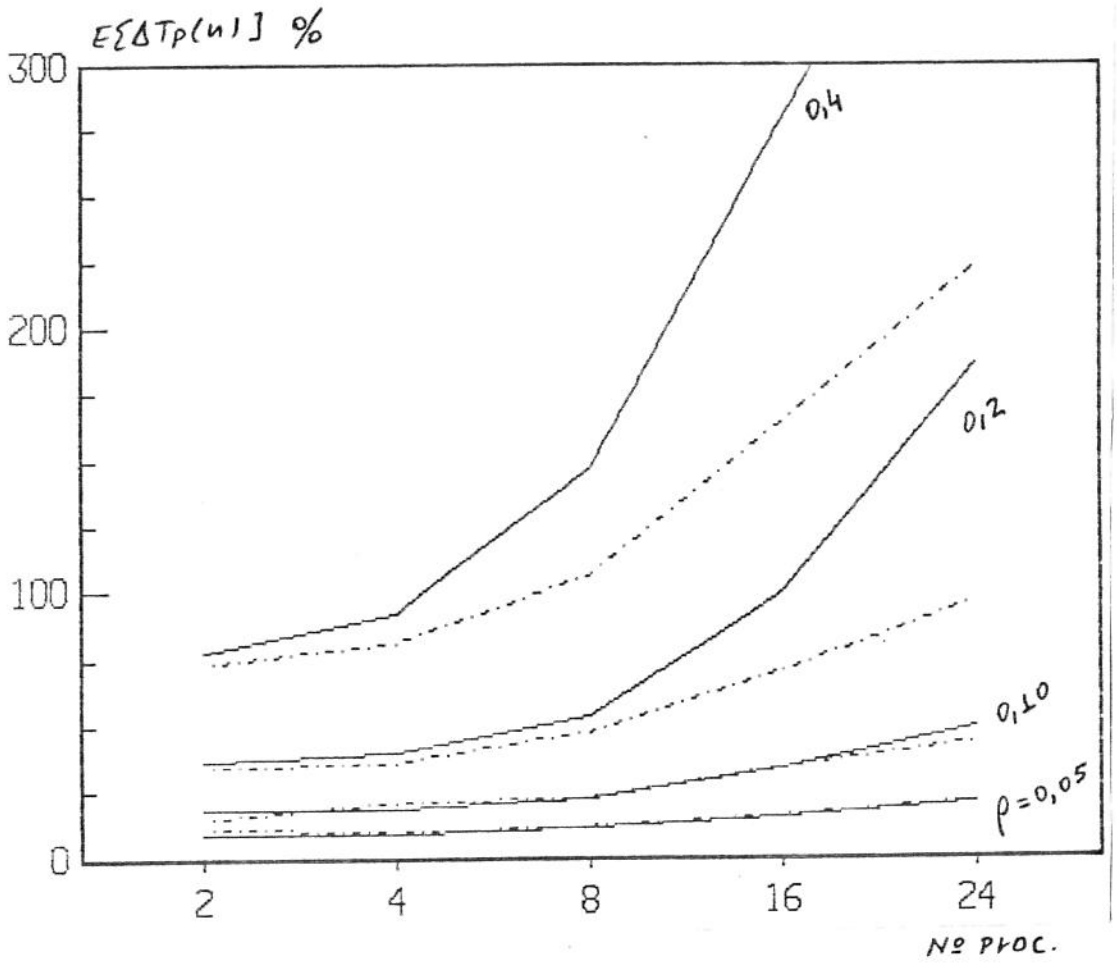
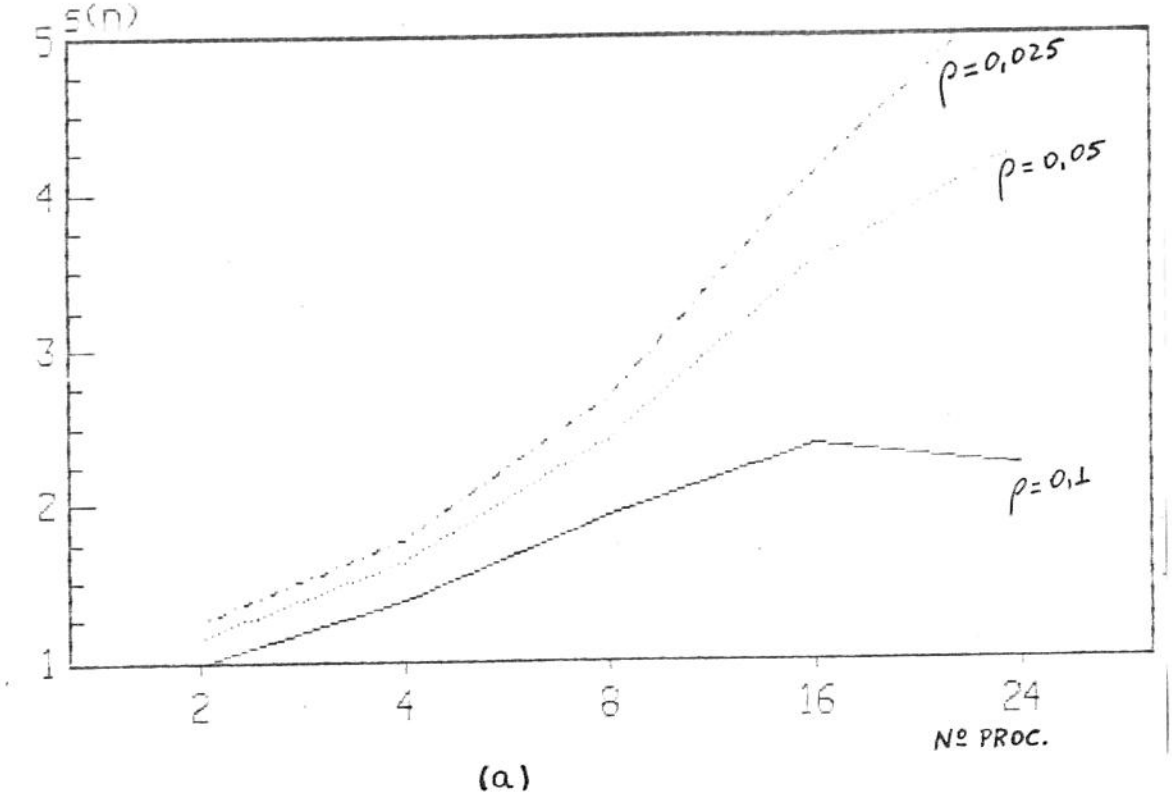


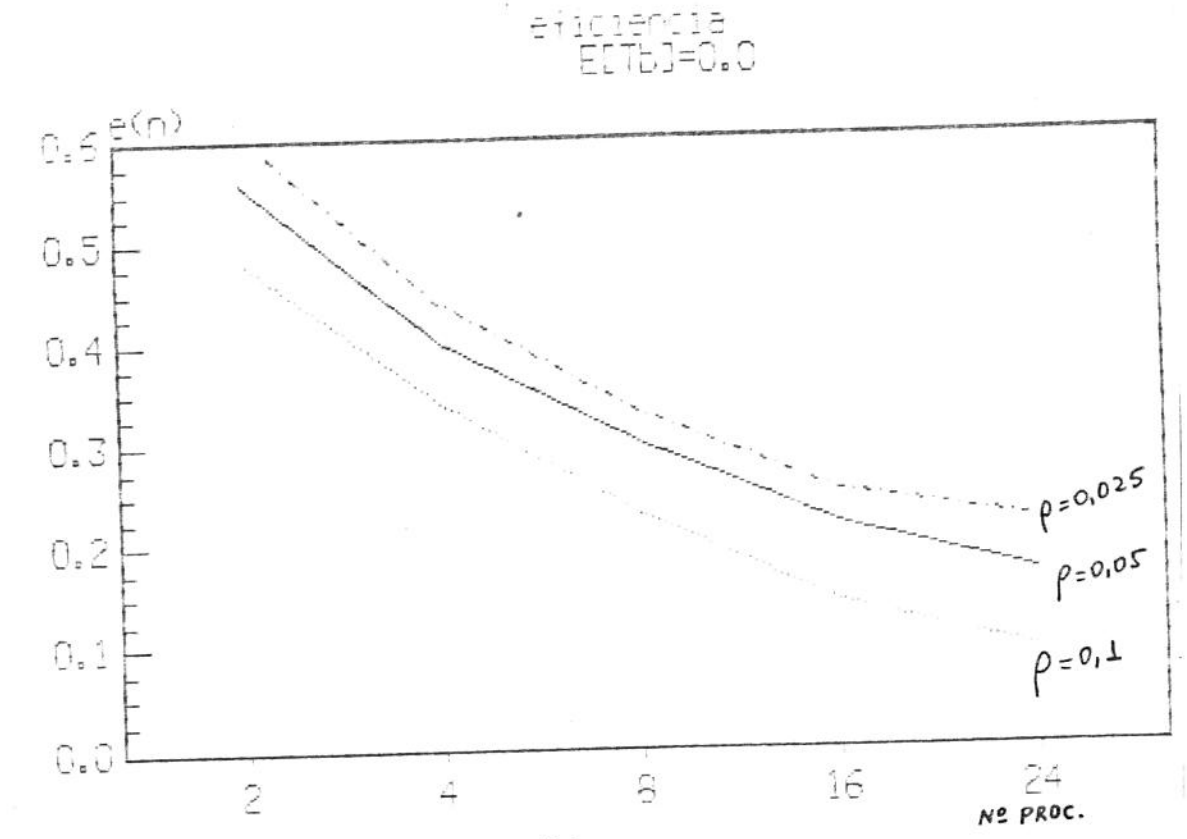
FIG.13 - AUMENTO NO CICLO PARA A ARQUITETURA RADIAL

—— EXPONENCIAL

---- CONSTANTE



(a)



(b)

FIG. 14 - GANHO DE VELOCIDADE (a) E EFICIENCIA (b) NA ARQUITETURA RADIAL.

Tabela I - Aumento relativo no período A
p/ tempos de comunicação constante(%)

ρ	2	4	8	16	24
0.005	0.1800	00.720	0.2000	0.9000	1.0200
0.010	2.1400	2.5800	1.1400	1.7400	4.3600
0.025	0.9900	2.2300	0.5200	0.4800	1.3600
0.050	10.300	8.3200	11.050	15.350	19.820
0.100	13.570	19.960	22.090	32.930	42.490
0.200	33.900	35.910	46.350	69.420	96.370
0.400	72.300	80.430	104.95	164.23	222.91
0.800	85.660	172.92	236.96	368.07	493.34

Tabela II - Aumento relativo no período A
p/ tempos de comunicação com
distribuição exponencial(%)

ρ	2	4	8	16	24
0.005	1.0000	0.8100	0.9700	1.3780	1.7800
0.010	1.6600	1.7300	2.0200	2.8400	3.6500
0.025	4.2200	4.2800	5.1600	5.7700	7.4400
0.050	8.5300	8.6900	10.560	15.080	19.780
0.100	17.500	18.040	22.340	33.880	49.290
0.200	36.600	39.400	53.340	98.970	186.75
0.400	78.020	92.410	145.93	279.21	508.48
0.800	168.82	223.96	372.08	657.13	916.89

3.3. MODELO COM COMUNICAÇÃO: ARQUITETURA POR BARRAMENTO

O processo de comunicação na arquitetura por barramento é mais complexo que na arquitetura radial. Considerando os tempos de gerenciamento e controle, as hipóteses markovianas ficam completamente destruidas. Se não se considera estes períodos, um modelo markoviano é possível em alguns casos. No entanto, o número de estados é excessivamente grande para um pequeno número de processadores, pois há necessidade de se expandir a dimensão do vetor de estado. Assim, esta arquitetura será estudada por simulação.

Um estudo de simulação envolve uma série de etapas e procedimentos para elaboração e validação de modelos e resultados nos quais não vamos nos deter demoradamente pois o nosso sistema é simples e bem compreendido. Maiores elaborações se justificam normalmente por dois motivos: primeiro, na simulação de grandes sistemas onde os custos envolvidos na simulação são altos, e segundo, quando não se tem uma compreensão muito clara do sistema que se vai simular, o que exige um controle fino sobre efeitos de variáveis desconhecidas. Os principais conceitos e resultados utilizados são revisados no apêndice 2.

O nosso problema de aplicação é um problema de simulação terminante(veja apêndice 2), pois o evento "último processador a comunicar" determina o final do ciclo.

3.3.1. MECANISMOS DE DECISÃO

Primeiramente devemos ressaltar que na arquitetura por barramento, a comunicação envolve dois acessos (um para escrever, outro para ler) à memória comun.

Nesta arquitetura devemos especificar a disciplina de decisão na utilização do barramento. As implementações adotam com frequência um dos mecanismos descritos na seção 2.1 deste trabalho: Daisy-Chain, polling e independent request.

Não vamos considerar aqui o efeito do tempo de decisão e propagação de sinais de controle. No entanto devemos alertar que esses períodos podem ser significativos em algumas implementações[06], podendo mesmo gerar degradação substancial na eficiência. A não consideração destes efeitos resulta nas seguintes simplificações:

.Daisy-Chain- Consideramos dois casos:

1. O processador coordenador B transfere todos os dados à memória comum em um único acesso no inicio do ciclo (Daisy-chain 1). Neste caso, se tentarmos manter a definição de estado do Lema 2, as hipóteses markovianas ficam destruidas. De fato, a duração do primeiro acesso não é exponencial.

2. O processador B só transfere os dados relativos a um processador em cada acesso (Daisy-chain 2). Desde que as distribuições são exponenciais, o processo continua sendo markoviano. No entanto, o modelo que adotamos já não descreve o processo.

Em ambos os casos, uma descrição de estado mais complexa poderia modelar este caso, mas o número de estados certamente tornaria o tratamento proibitivo.

.Independent Request- Nesta forma, qualquer que seja a prioridade fixa que adotemos, será equivalente ao Daisy-Chain 2 já que não poderemos diferençar uma prioridade fixa de outra

qualquer.

.Polling- Como os tempos de decisão são assumidos serem nulos, tambem este caso admitiria um modelo markoviano. Portanto, valem as mesmas observações feitas para Daisy-chain 2.

Assim, restam três mecanismos de decisão a serem simulados, quando as distribuições são exponenciais. Para distribuição constante, todos devem ser simulados.

3.3.2. OS MODELOS DE SIMULAÇÃO

Os modelos de simulação que adotamos é tal que desprezam os tempos de propagação dos sinais de controle, os tempos de arbitragem, os efeitos do gerenciamento do sistema operacional e outros, mas mantém a sequência lógica que estes fatores determinariam. Isto significa que consideramos exclusivamente as tarefas de processamento e comunicação como consumindo tempo dos processadores, mas que os outros fatores, embora não consumindo tempo, são considerados nos seus efeitos sobre a sequência de operações gerada. De outro modo, alguns impasses poderiam surgir: por exemplo, no caso Daisy-Chain, onde o processador B é o processador mais próximo do controlador de barramento, nunca seria dada a vez a outro processador já que o processador B sempre terá uma verificação a ser feita na memória compartilhada (MC).

A fig.15 mostra o comportamento de um ciclo típico para os três mecanismos a serem simulados. O exemplo utiliza mesmos tempos de processamento e comunicação para cada processador. Observamos que algumas disciplinas geram bloqueio e outras não, e

que a duração do período-A é diferente para cada caso. O efeito do mecanismo da disciplina utilizada aparece claramente nesta figura.

3.3.3. RESULTADOS

O procedimento descrito na seção 3.3.1 para simulação terminante foi implementado para os três mecanismos de decisão. O algoritmo básico é o mesmo para os três casos. O que muda de um para o outro é um procedimento interno que gera as amostras Y_i.

Os resusltados para um intervalo de confiança de 95% e precisão relativa de 0.075 (valores recomendados na literatura) estão mostrados nas figuras 16 e 17 e Tabelas III, IV e V.

A figura 17 apresenta os resultados das fig. 16a, 16b e 16c sobrepostos no mesmo gráfico para alguns valores de p. Podemos observar que sob as hipóteses estudadas, e considerando-se como critério o adicional de comunicação, a sequência preferível é Daisy-Chain 2, Daisy-Chain 1 e Polling, nesta ordem. O que é surpreendente: com maior simplicidade de implementação, obtem-se o melhor resultado, para mesmo p é claro.

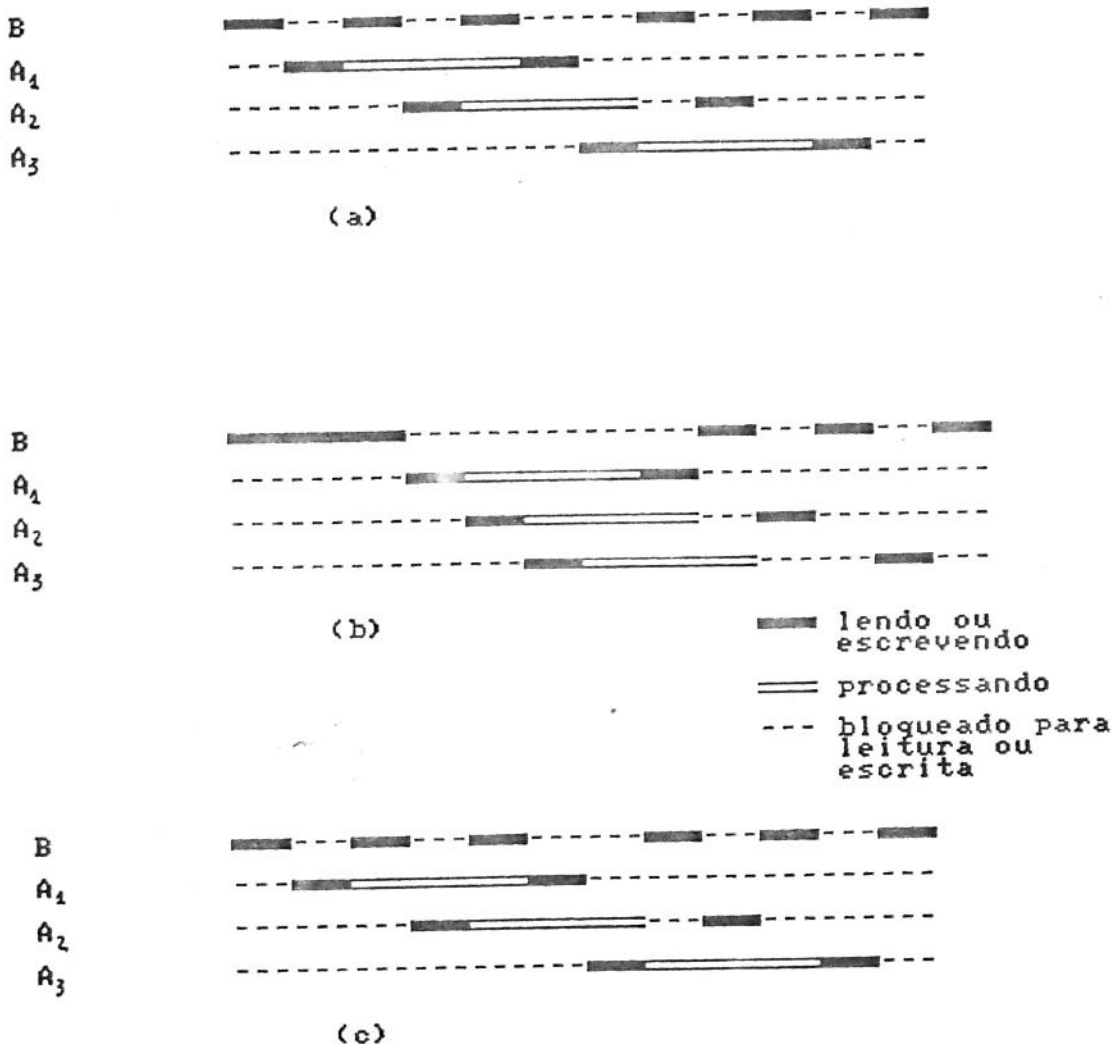
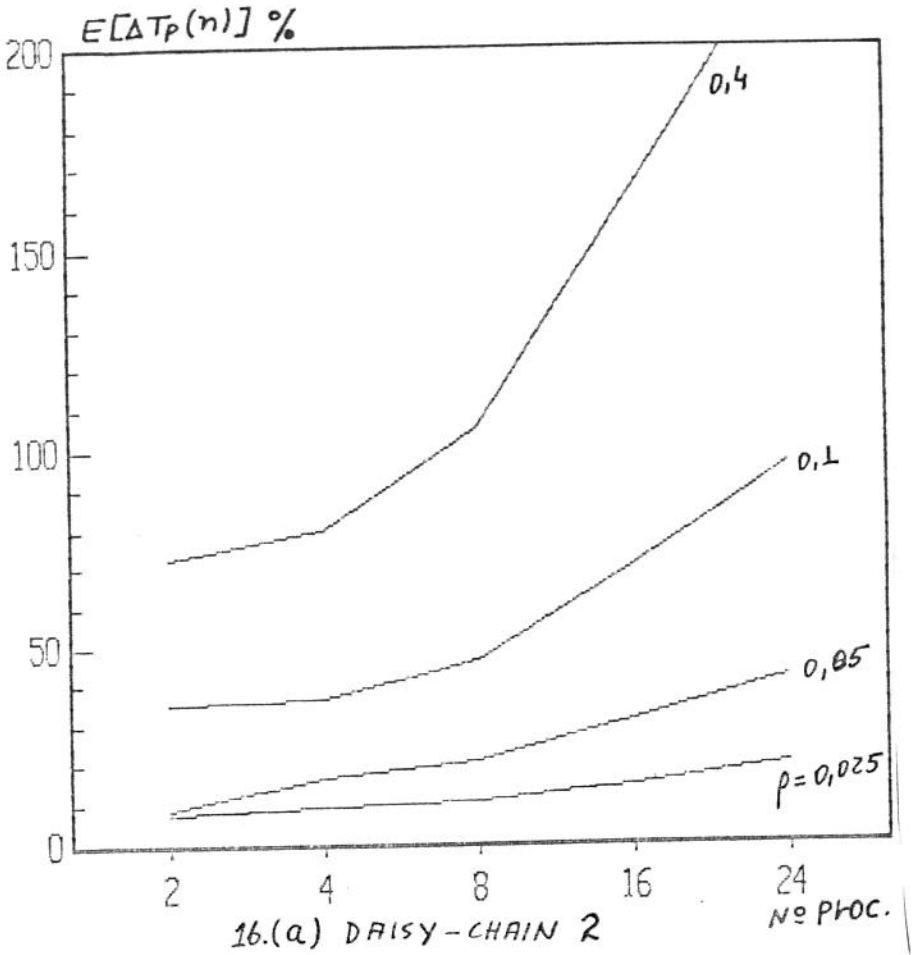
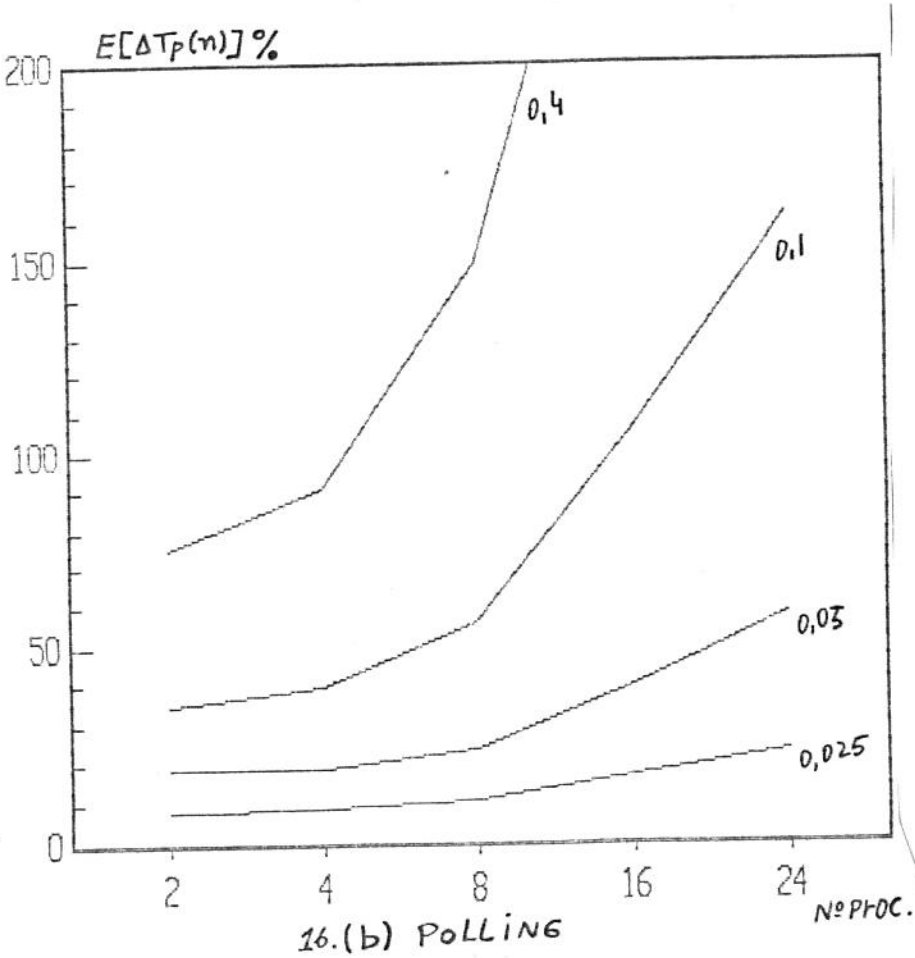


Fig.15 - Um ciclo processamento-comunicação para as três disciplinas do barramento
(a) Daisy-chain 1: O proc. B só transfere uma mensagem em cada acesso
(b) Daisy-chain 2: O proc. B transfere todas as mensagens em um acesso
(c) Polling



16.(a) DAISY-CHAIN 2



16.(b) POLLING

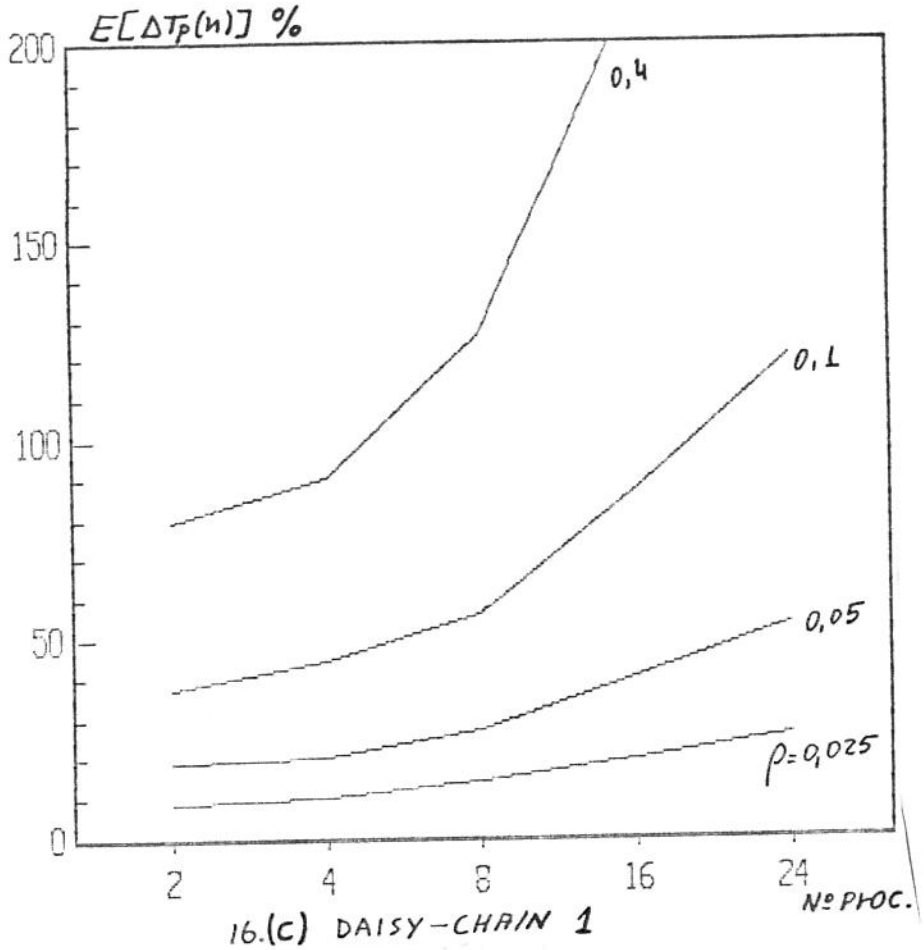
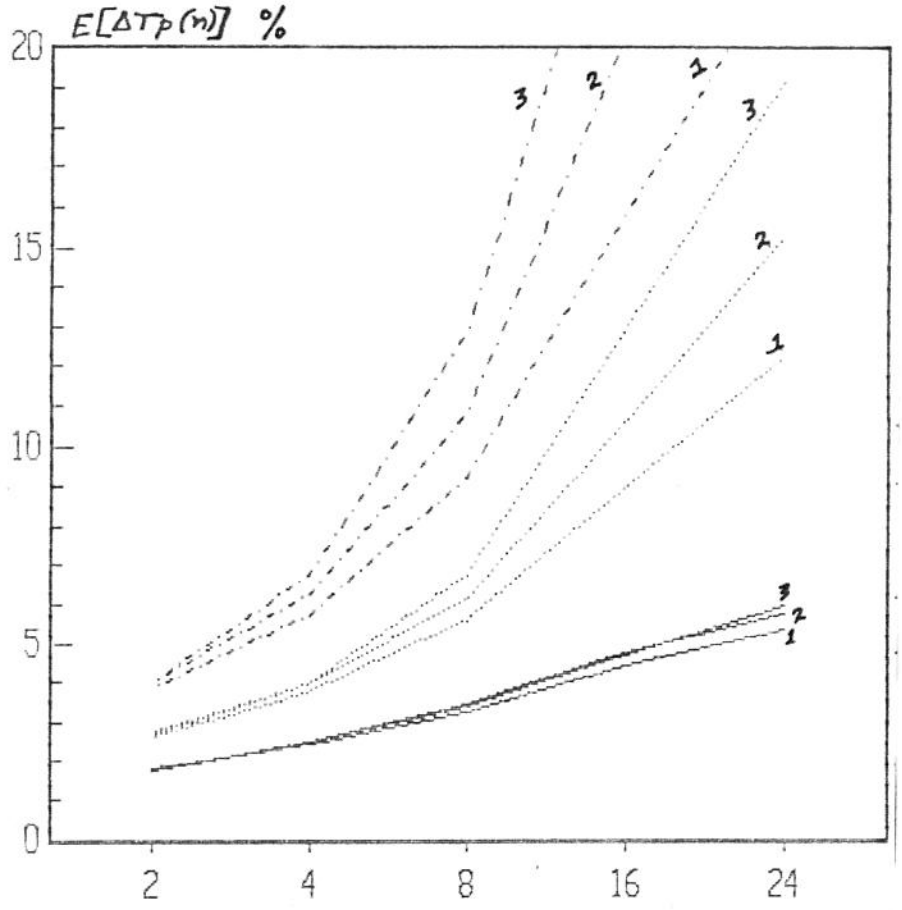


Fig. 16 - AUMENTO RELATIVO NO CICLO PARA A ARQUITETURA POR BARRAMENTO



1 - DAISY-CHAIN 2

2 - DAISY-CHAIN 1

3 - POLLING

$\rho_2 = 0,2$

$\rho_1 = 0,1$

$\rho_0 = 0,01$

FIG. 17 - COMPARAÇÃO ENTRE MECANISMOS DE DECISÃO

Tabela III- Daisy-chain 2

1.00000	1.10000	1.72000	3.05000	3.87000
3.16000	4.38000	4.22000	6.34000	7.37000
7.34000	9.11000	10.2900	14.6000	19.9300
8.78000	16.7100	20.9400	31.0700	42.0900
35.5400	36.3800	46.0400	70.1400	96.3100
71.9800	79.8500	104.810	164.010	223.330
152.800	174.320	237.740	368.000	491.720
314.220	472.680	512.030	780.560	1132.86

Tabela IV- Polling

1.58000	2.21000	2.30000	2.65000	3.86000
2.76000	3.22000	3.65000	6.27000	7.51000
8.12000	8.41000	10.6200	16.9800	23.2600
18.5400	19.0900	23.7700	39.5600	58.0600
34.6600	39.6700	55.8300	105.550	161.370
75.2200	90.7100	148.980	280.900	408.890
161.310	222.510	373.250	657.250	916.950
350.300	518.180	841.950	1514.40	1933.89

Tabela V- Daisy-chain 1

0.76000	2.02000	2.99000	3.82000	5.02000
3.12000	4.37000	5.80000	8.05000	9.82000
8.43000	9.95000	14.0800	19.6700	25.4600
18.5900	20.2500	26.9600	40.2500	53.2000
37.4400	44.0300	56.1100	86.3400	120.420
78.8100	89.9100	125.960	212.020	303.530
164.080	199.970	297.690	501.180	692.320
343.400	442.600	666.780	1083.69	1468.89

3.4. ANÁLISE COMPARATIVA

Diversas estratégias foram usadas para teste e validação dos mecanismos de simulação e do processo de integração da equação de Kolmogorov. A solução por probabilidades estacionárias foi implementada para pequenos números de processadores e os resultados foram comparados com a integração. Também foi simulado o caso exponencial para a arquitetura radial, para cruzamento dos resultados. Nos modelos para a arquitetura por barramento diversos ciclos de execução foram acompanhados passo a passo e longas corridas do processo foram testadas. Particularmente, todos os casos foram também validados fazendo os tempos de comunicação nulos.

Uma síntese dos resultados obtidos nas seções anteriores aparece na fig.18a, para efeito de comparação. Tomamos o valor r=0.1. As principais observações que se tira desta figura são:

1. Na arquitetura por barramento, a sequência das disciplinas de decisão, em ordem crescente de aumento na duração do ciclo, é Daisy-chain 2, Daisy-chain 1, e Polling.

É importante lembrar, que no Daisy-chain, o sinal que dá acesso ao barramento deve se propagar por várias unidades até chegar à unidade que vai ser servida. Já no polling, existe uma linha de acesso para cada unidade. Isto significa que o efeito dos mecanismos de decisão é mais forte no Daisy-chain que no polling. É de se esperar que este fenômeno poderá diminuir a diferença entre as duas formas de decisão.

2. Numa comparação entre as arquiteturas, o adicional por comunicação é substancialmente maior na arquitetura por

barramento.

Aqui é relevante o fato de que temos por hipótese que os tempos de comunicação e processamento são os mesmos para os dois casos. No entanto, por limitações de tecnologia e custos, é mais frequente que ligações diretas entre processadores (radial) ocorram na forma de interface serial e nas arquiteturas por barramento são invariavelmente na forma paralela. Evidentemente, que se tratando da mesma mensagem (mesmo problema a ser resolvido) o tempo de comunicação será sensivelmente reduzido se a comunicação é feita via barramento paralelo. Neste caso teríamos de comparar as duas arquiteturas para diferentes valores de ρ, dependendo dos sistemas sendo comparados.

3. Na arquitetura radial, a fig.18a sugere que o modelo para tempos com distribuição exponencial (CMTC) pode ser perfeitamente usado para o caso de comunicação com distribuição constante para uma ampla faixa do parâmentro ρ e de número de processadores. As tabelas I e II mostram que o erro que se comete ao fazer esta aproximação, aumenta com o valor de ρ. O erro máximo na faixa até 24 processadores é de 4 %.

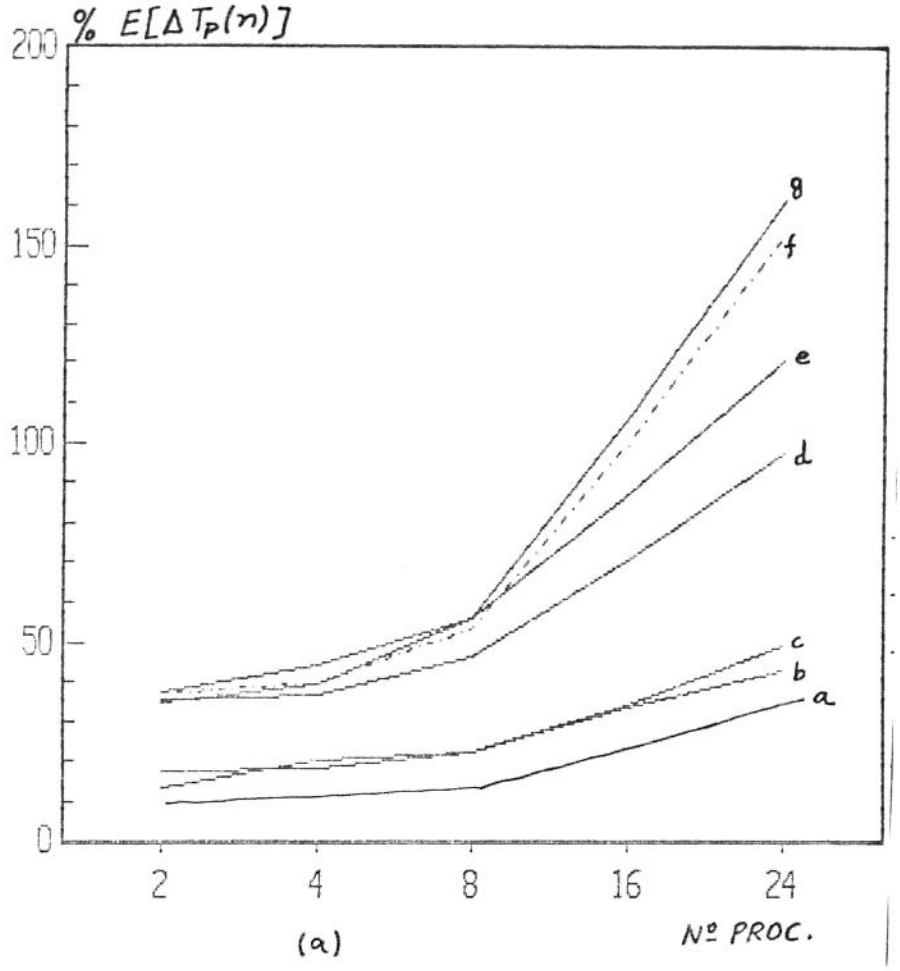
4. Na curva f da fig.18a, foi plotada o caso ρ =0.2 para a arquitetura radial, ou seja, com o dobro do tempo de comunicação. (A fig.18a é para ρ = 0.1). Isto foi sugerido pelo fato de que na arquitetura por barramento, existem dois acessos para cada comunicação. O objetivo é verificar a possibilidade de se usar o modelo markoviano como uma aproximação para o caso do barramento. A figura mostra que o modelo se presta para esta

aproximação de forma adequada para o caso do barramento com polling. Nos outros dois casos no entanto, para um número de processadores maior que 16 o erro torna-se significativo. Os erros máximos desta aproximação até 24 processadores são: 5 % para Polling; 20 % para Daisy-Chain 1; e 45 % para Daisy-Chain 2.

A fig.18b testa a sensibilidade desta aproximação em relação aos valores de ρ. Verificamos que a aproximação se mantem para a ampla faixa de valores analisada.

Estas aproximações são importantes porque passamos a dispor de um modelo unificado para o problema. Apesar do modelo não ter uma solução analítica, tanto as entradas para o programa de computador (n,λ,μ) como sua execução são extremamente simples e evita a simulação que é certamente mais dispendiosa.

Devemos ainda lembrar que no atual estágio de desenvolvimento ainda existem muitos aperfeiçoamentos a serem feitos na tecnologia dos sistemas operacionais para máquinas paralelas, de formas que, a tendência será de que a hipótese de tempos de gerenciamento desprezíveis tende a ser cada vez mais válida no futuro.



(a) SEM COMUNICAÇÃO

(b) RADIAL / COMUNICACÃO CONSTANTE ($\rho = 0.1$)

(c) RADIAL / COMUNICACÃO EXPONENCIAL ($\rho = 0.1$)

(d) BARRAMENTO / DAISY-CHAIN 2 / COMUNICACÃO EXPONENCIAL ($\rho = 0.1$)

(e) BARRAMENTO / DAISY-CHAIN 1 / COMUNICACÃO EXPONENCIAL ($\rho = 0.1$)

(f) RADIAL / COMUNICAÇÃO EXPONENCIAL ($\rho = 0.2$)

(g) BARRAMENTO / POLLING / COMUNICACÃO EXPONENCIAL ($\rho = 0.1$)

FIG. 18a- COMPARAÇÃO DOS RESULTADOS PARA AS DUAS ARQUITETURAS E DIVERSAS DECISÕES

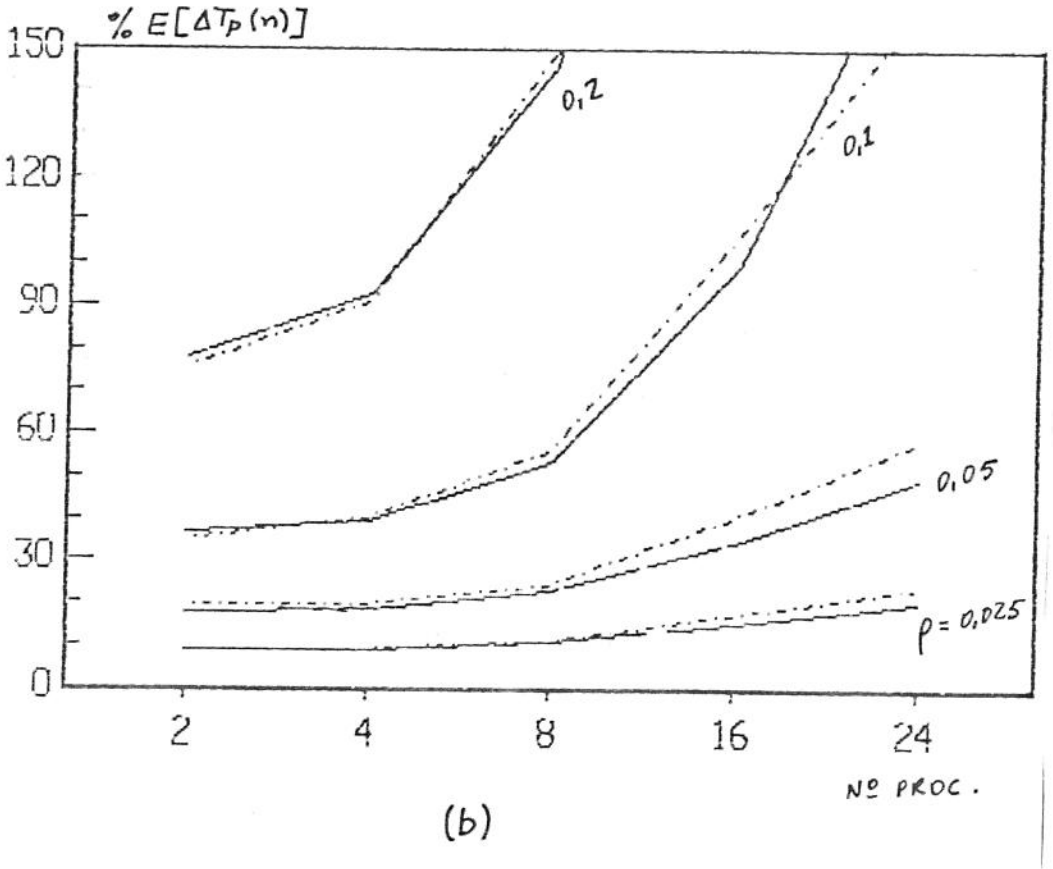


Fig. 18b.- Verificação da aproximação do modelo de CMTC do Lema 2 para o caso do barramento.

CAPÍTULO 4: APLICAÇÃO AO PROBLEMA DE DECOMPOSIÇÃO DE ALGORITMOS

4. APLICAÇÃO: DECOMPOSIÇÃO DE ALGORÍTMOS MULTINÍVEL

Os capítulos anteriores analisaram os efeitos de comunicação, sincronização e bloqueio sobre a duração média de um ciclo de execução hierárquico. O aumento no ciclo esteve representado pelo fator $E[\triangle Tp(n)]$ e os gráficos obtidos estavam parametrizados pelo parâmentro $p = E[Ta(n)]/E[Tp(n)]$. Isto nos possibilita comparar arquiteturas e fornece subsídios para se optar entre elas.

No contexto de algoritmos multinível, a execução total do algoritmo envolve um número variado de ciclos. Este capítulo procura extender os resultados obtidos para um ciclo e aplicá-los na estimativa do tempo total de execução de algoritmos multinível. Este problema é relevante quando queremos avaliar se um determinado par algoritmo-arquitetura atende os requisitos de tempo de uma aplicação. Neste caso, alguns fatores novos são introduzidos no problema. Verificaremos o compromisso entre a relação processamento-comunicação e os índices de ganho de velocidade e eficiência da decomposição.

Considere que temos um algoritmo sequencial iterativo para a solução de um determinado problema de grande porte, que chamaremos de solução global. Suponha que pelas características próprias do problema, é possível implementar uma solução hierárquica, com o objetivo de diminuir o tempo de execução. Em geral, por um processo de decomposição, o problema global é dividido em n subproblemas dirigidos por um coordenador. O número ótimo de subsistemas é dependente de ambos, a arquitetura e o problema, uma vez que o custo de comunicação por iteração e aumento da relação processamento-comunicação, maior solicitação

sobre a rede de interconeção, rede de interconecção mais complexa,...) e o número total de iterações tende a aumentar com o aumento do número de subsistemas.

Se as tarefas não forem homogêneas, ou se o número de processadores for diferente do número de subsistemas, surge o problema de alocação processador-tarefa no sentido de otimizar a relação balanceamento da carga x eficiência da comunicação. No entanto, aqui vamos supor cargas balanceadas, um processador para cada subsistema e mesmas distribuições dos tempos de comunicação e de processamento para todos os processadores. A referência [10] descreve vários algoritmos que usam esta técnica em otimização de sistemas de grande porte ou descentralizados, baseados em decomposição lagrangeana.

Para avaliarmos a eficiência do processo de decomposição, considerando-se vários ciclos, devemos verificar o que acontece com cada um dos cinco seguintes fatores quando se aumenta o número de sub-sistemas: o esforço computacional dos subsistemas e do coordenador na decomposição, a variação destas tarefas em cada ciclo, o efeito da comunicação e o número de ciclos para convergência.

Vamos inicialmente dar algumas definições[10]. Seja Tp o tempo de processamento e Ta o tempo de acesso a dados na solução global. Defina uma relação comunicação-processamento inicial como $\rho o = Ta/Tp$. Seja agora tp(n) e ta(n) os tempos de processamento e comunicação para os subsistemas resultantes da decomposição em n subproblemas. Usualmente $tp(n) < Tp$ e $ta(n) < Ta$, mas frequentemente não é verdade que $n.tp(n) = Tp$ ou $n.ta(n) = Ta$.

As relações entre Tp, Ta, tp(n) e ta(n) são dependentes da estratégia de decomposição que o algoritmo admite. Assumindo que tp(n) e ta(n) não mudam de um ciclo para o outro, vamos então definir funções de decomposição fp(n) e fa(n) como fp(n) = Tp/tp(n) e fa(n) = Ta/ta(n). Defina uma "decomposição" como o par ordenado (fp : fa). Funções de decomposição típicas são dadas em [10] e [11]. É fácil ver que o fator ρ como uma função de n é dado por

$$\rho(n) = \rho o . fp(n)/fa(n) \qquad (21)$$

Assim, a determinação da função $\rho(n)$ para um algoritmo, através da especificação da decomposição (fp:fa) admitida e do valor de no, considera o efeito dos três primeiros fatores citados: comunicação, e esforço computacional de subsistemas e coordenador.

Dependendo das funções fp(n) e fa(n), um comportamento típico destes métodos de decomposição é que, por um lado a decomposição em um número cada vez maior de subsistemas tende a diminuir substancialmente o tempo de processamento; por outro lado, o valor de $\rho(n)$ tende a crescer, o que pelos resultados obtidos anteriormente, poderá comprometer a eficiência do processo de decomposição. Isto significa que explorar o paralelismo alem de um certo nível, pode ser contra producente devido ao adicional de comunicação inerente ao processo de decomposição.

Para os exemplos apresentados em [10] e [11], onde a sequência de operações é bem determinada, as funções fp(n) e fa(n) podem ser calculadas exatamente, já que os cálculos envolvidos são bem definidos em termos de número de operações.

Considere agora os seguintes exemplos:

Exemplo 1[30][31]: Problema de Alocação Generalizado

O problema de alocação generalizado consiste na alocação de n tarefas a m agentes. Definimos:

C_{ij} = custo incorrido se a tarefa j é alocada ao agente i

b_i = disponibilidade de recursos do agente i

a_{ij} = recurso necessário para o agente i executar a tarefa j

$$x_{ij} = \begin{cases} 1, \text{ se o agente i realiza a tarefa j} \\ 0, \text{ caso contrário} \end{cases}$$

Declaração do problema:

$$Z = \min \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} \cdot x_{ij} \qquad (22)$$

$$\text{s.a.} \quad \sum_{i=1}^{m} x_{ij} = 1, \quad j=1,..,n \qquad (23)$$

$$\sum_{j=1}^{n} a_{ij} \cdot x_{ij} \leq b_i, \quad i=1,..,m \qquad (24)$$

$$x_{ij} \in \{ 0,1 \}, \quad \forall ij \qquad (25)$$

Uma solução usando Relaxação Lagrangeana consiste em dualizar as restrições (23). Tomando u_j, $j=1,..,n$ como multiplicadores de Lagrange, temos:

$$Z_D(u) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} \cdot x_{ij} + \sum_{j=1}^{n} u_j \left(\sum_{i=1}^{m} x_{ij} - 1 \right)$$

$$\text{s.a.} \quad (24) \text{ e } (25)$$

ou

$$Z_D(u) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} (C_{ij} + u_j)x_{ij} - \sum_{j=1}^{n} u_j \qquad (26)$$

s.a. (24) e (25)

A solução $Z_D(u)$, para u fixado, é obtido através da minimização de m subproblemas. Para cada i=1,..,m, resolva:

$$\min \sum_{j=1}^{n} (C_{ij} + u_j)x_{ij} + \sum_{j=1}^{n} u_j \qquad (27)$$

$$\text{s.a.} \quad \sum_{j=1}^{n} a_{ij} \cdot x_{ij} < b_i \ , \qquad (28)$$

$$x_{ij} \in \{ 0,1 \} \qquad (29)$$

Reconhecemos cada um dos subproblemas como um "problema da mochila"(Knapsack).

A tarefa do coordenador é determinar uma sequência de valores para u, $\{ u_k \}$, tal que a solução dos subproblemas leve à solução ótima.

Vários métodos de atualização de u são sugeridos na literatura[30]. De uma forma geral são dados pela equação:

$$u^{k+1} = u^k + t_k d_k \qquad (30)$$

onde $t_k > 0$ = tamanho do passo

d_k = uma direção na qual $Z_D(u)$ cresce.

A determinação de t_k e d_k , dependendo do método usado, pode envolver busca unidimensional, determinação de subgradientes, etc.

Vamos caracterizar os requisitos de comunicação e processamento para este exemplo:

Comunicação: B --> A : um vetor de n valores reais

A --> B : um vetor de m valores reais

Processamento:

-Tarefa do coordenador: testar a otimalidade das soluções em cada iteração e atualizar os multiplicadores u_j, $j=1,..,n$

-Tarefa de cada subproblema: Resolver um "problema da mochila"

Se o número de processadores disponíveis para os subproblemas é diferente de m, é possível, por exemplo, alocar dois subproblemas a cada processador. Pode-se ver que para este exemplo, mantendo-se a distribuição de tarefas balanceada, e sendo n o número de processadores utilizados,temos:

- o tamanho das mensagens de comunicação diminui com 1/n;
- o esforço computacional de cada processador do nível A diminue com 1/n;
- o esforço computacional do coordenador não se altera com n

Também, pode-se concluir pela natureza aleatória dos tempos de processamento e do número de ciclos para convergência, tanto do processador B quanto dos processadores Ai.

Exemplo 2[32][26]: Controle Multinível

Considere agora a seguite formulação do problema discreto de controle multinível para sistemas interconectados:

Dinâmica do sistema: Para $i=1,..,N$,

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

$$x_i(0) = x_{io}, \quad k=0,...,p-1 \qquad (31)$$

Interconecção:

$$\sum_{i=1}^{N} h_i[x_i(k),u_i(k),k] \leq 0, \quad k=0,..,p-1 \qquad (32)$$

Restrições: Para $i=1,...,N$,

$$R_i[x_i(k),u_i(k),k] \leq 0, \quad k=0,..,p-1 \qquad (33)$$

$$\theta_i[x_i(p)] \leq 0 \qquad (34)$$

Índice de desempenho: Minimizar J onde

$$J = \sum_{i=1}^{N} \left(S_i[x_i(p)] + \sum_{K=0}^{P-1} F_i[x_i(k),u_i(k),k] \right) \qquad (35)$$

As equações (31)-(34) representam N subsistemas acoplados, onde x_i e u_i são vetores de estado e controle respectivamente. A introdução do problema dual resulta na formulação de dois níveis com o nível A resolvendo o problema primal e o nível B resolvendo o problema dual. O dual é:

$$\max_{\lambda} \phi(\lambda) \quad , \quad \lambda \geq 0 \qquad (36)$$

com

$$\phi(\lambda) = \min_{x,u} L(x,u,\lambda) \qquad \text{s.a.} \quad (31),(33),(34) \qquad (37)$$

onde o funcional Lagrangeano é

$$L(x,u,\lambda) = \sum_{i=1}^{N} \left(S_i[x_i(p)] + \sum_{K=0}^{P-1} F_i[x_i(k),u_i(k),k] + \lambda_i . h_i[x_i(k),u_i(k),k] \right) \qquad (38)$$

O funcional lagrangeano (38) é aditivamente separável já que as restrições de interconeção (32) representam acoplamento somente entre subsistemas e não entre estados ou controles. Consequentemente ficam definidos N problemas de minimização independentes. Assim temos:

Tarefa do nível A:

$$\min L_i(x,u,\lambda) =$$

$$= S_i[x_i(p)] + \sum_{K=0}^{P-1} F_i[x_i(k),u_i(k),k] + \lambda_i h_i[x_i(k),u_i(k),k]$$

$$\text{s.a.} \quad x_i(k+1) = f_i[x_i(k),u_i(k),k]$$

$$x_i(0) = x_{i0}, \quad k=0,..,p-1$$

$$R_i[x_i(k),u_i(k),k] \leq 0, \quad k=0,..,p-1$$

$$Q_i[x_i(k)] \leq 0$$

Tarefa do nível B:

Resolver (36)-(38)

Geralmente é impraticável para o nível B minimizar o Lagrangeano como uma função explícita dos multiplicadores, e portanto a solução é trocada por uma busca sequencial para o ótimo[32].

Tambem neste caso, os métodos de atualização dos multiplicadores podem ser escritos de uma forma geral como:

$$\lambda^{k+1} = \lambda^{k} + a_k . d_k \qquad (39)$$

onde: $a_k \rangle 0$, tamanho do passo e

d_k , uma direção.

Com a solução sequencial, as tarefas do coordenador e subsistemas ficam:

Coordenador: verificar a otimalidade das soluções em cada iteração, e atualizar os multiplicadores λ .

Subproblemas: resolver um problema de otimização definido pelas equações (36)-(38).

Os requisitos de comunicação são:

B --> A: vetor de multiplicadores λ_i

A --> B: vetores x_i e u_i, de estado e controle respectivamente

A forma de acoplamento deste exemplo é específica. No entanto, um tratamento generalizado pode ser dado ao problema de controle e otimização de sistemas dinâmicos interconectados, o qual leva a várias técnicas de decomposição e coordenação[32].

Para os exemplos 1 e 2, os tempos de processamento são v.a. Situações onde tambem os tempos de comunicação sejam melhor representados por v.a. poderiam ser descritas. Assim, para generalizar as definições dadas em [10] e [11], vamos definir

fp(n) e fa(n) como a relação entre valores médios para os tempos de processamento e comunicação:

$$fp(n) = E[T_p(1)]/E[T_p(n)], \quad fa(n) = E[T_A(1)]/E[T_A(n)]. \quad (40)$$

As situações onde Tp(n) e Ta(n) são valores determinísticos, é claro, estão incluidas nesta definição.

Para exemplificar como avaliar os efeitos da função r(n) que um determinado algoritmo admite, sobre o ganho de velocidade na execução paralela, considere uma situação onde a decomposição seja dada por ($N : \sqrt{N}$), e $p_o = 0.1$. Pela relação (31) temos $p(n) = p_o.\sqrt{N}$. Então temos:

$p(2) = 0.14$; $p(4) = 0.20$; $p(8) = 0.28$

$p(16) = 0.40$; $p(24) = 0.49$

A figura 19 mostra como evolui o adicional de comunicação por ciclo a medida que se decompõe o algoritmo. O efeito é que, para cada n, a relação comunicação x processamento do algoritmo está sobre uma curva diferente na família de curvas parametrizada por r. Os gráficos são para a arquitetura radial.

Não faz parte dos propósitos deste trabalho fazer um estudo exaustivo para determinar o valor de ro e a decomposição (fa ; fp) para algoritmos reais. Portanto, vamos adotar como paradígma os três grupos de decomposição estudados em [10],[11] e [33], no sentido de estabelecer faixas para os efeitos da decomposição sobre o ganho de velocidade. Específicamente, são as decomposições (N:N), ($N : \sqrt{N}$) e (N : 1).

Aplicando estes dados à relação (1) obtemos as curvas de

ganho de velocidade para as três decomposições, mostradas na fig.20. A figura não inclue o efeito da tarefa B (a qual é sempre executada em série), já que foram traçadas para E[Tb]=0. A extensão para incluir E[Tb] e para os casos onde E[Tb] varia com n é imediata. O efeito de E[Tb] será uma redução no ganho de velocidade, como pode ser deduzido da equação (13).

O índice s(n) fornece o ganho incremental obtido pela introdução de um processador adicional na estrutura. A eficiência e(n) é uma medida da proporção do período em que cada processador permanece ocioso. Da fig.20 vemos que existe um número a partir do qual torna-se contra producente a introdução de novos processadores em termos de ganho de velocidade.

Dos cinco fatores citados, resta-nos considerar no modelo o número de ciclos para convergência e a variação do esforço computacional em cada ciclo, sobre o ganho de velocidade. Neste caso vamos usar as relações (1a)-(1b) da seção 2.4.

Introduziremos um modelo preliminar para considerar estes fatores.

O primeiro deles, a variação do esforço computacional em cada ciclo é exemplificado pelo fato de que em certos problemas de otimização os tempos de processamento por ciclo tendem a diminuir cada vez que a iteração se inicia mais próximo do ponto de solução. De fato, algoritmos com convergência do tipo geométrica são frequentemente encontrados na literatura. Adotamos um modelo de convergência geométrica e assumiremos que a duração média do processamento no ciclo(iteração) é proporcional a uma distância para a solução convenientemente definida. Isto equivale a assumir que os períodos de processamento no ciclo também

diminuem geometricamente. Adotando o índice i para a i-ésima iteração(ciclo) do algoritmo e sendo α o fator de decréscimo, podemos escrever:

$$f_p^i(n) = \frac{f_p(n)}{\alpha^{i-1}} \qquad e \qquad \rho^i(n) = \alpha^{i-1}\rho(n)$$

onde $0 < \alpha \leq 1$ e $\rho(n)$ é dado pela equação (21).

Com esta definição o tempo de ciclo fica

$$T_c^i(n) = \left[T_B + T_p^i(n)(1 + \Delta T_p(n, \rho^i(n)))\right]$$

$$= \left[T_B + \frac{T_p(1)}{f_p^i(n)}(1 + \Delta T_p(n, \rho^i(n)))\right]$$

$$= \left[T_B + \alpha^{i-1}\frac{T_p(1)}{f_p(n)}(1 + \Delta T_p(n, \alpha^{i-1}\rho(n)))\right] \quad (41)$$

onde E[ΔTp(n,r(n))] é obtido a partir das curvas encontradas nas seções anteriores, sendo dados a arquitetura utilizada e o grupo de decomposição. A equação (41) mostra que o tempo de execução do algoritmo decomposto é função tanto da razão entre as funções de decomposição, através do adicional de comunicação E[ΔTp(n)], como dos seus valores, através de fp(n) (vide def. de $\rho(n)$, eq. (21)).

O caso $\alpha = 1$ modela situações onde os tempos de processamento não se alteram de um ciclo para o outro. Este caso não é de todo incomun. Por exemplo, um passo de gradiente em um algoritmo de otimização (estático) poderá não significar menos cálculos pelo fato de partir de um ponto próximo da solução.

Se o número de ciclos não se altera com n, então as relações (99) se mantém. Esta situação não é de todo desprezível. Seja o

caso dos exemplos 1 e 2. Se o número de subproblemas na decomposição fica definido e a variação no número de processadores acontece pelo fato de se alocar, por exemplo, dois ou mais subproblemas a cada processador Ai, então o número de ciclos para convergência se mantem para os dois exemplos.

No entanto, ainda no caso do exemplo 2, se os subsistemas são alterados na decomposição, incluindo, retirando ou agrupando de forma diferente as equações dinâmicas, então não podemos afirmar que o número de ciclos para convergência de manterá.

Seja N(n,no) uma v.a. com distribuição f_N (n,no) uma função que dá o número de ciclos para convergência, onde n é o número de subsistemas e no é o número de ciclos para convergência na solução global. Então a distribuição do tempo médio total de execução do algoritmo seria:

$$T(n) = \sum_{j=1}^{N(n,no)} E\{T_c^j(n)\} \qquad (42)$$

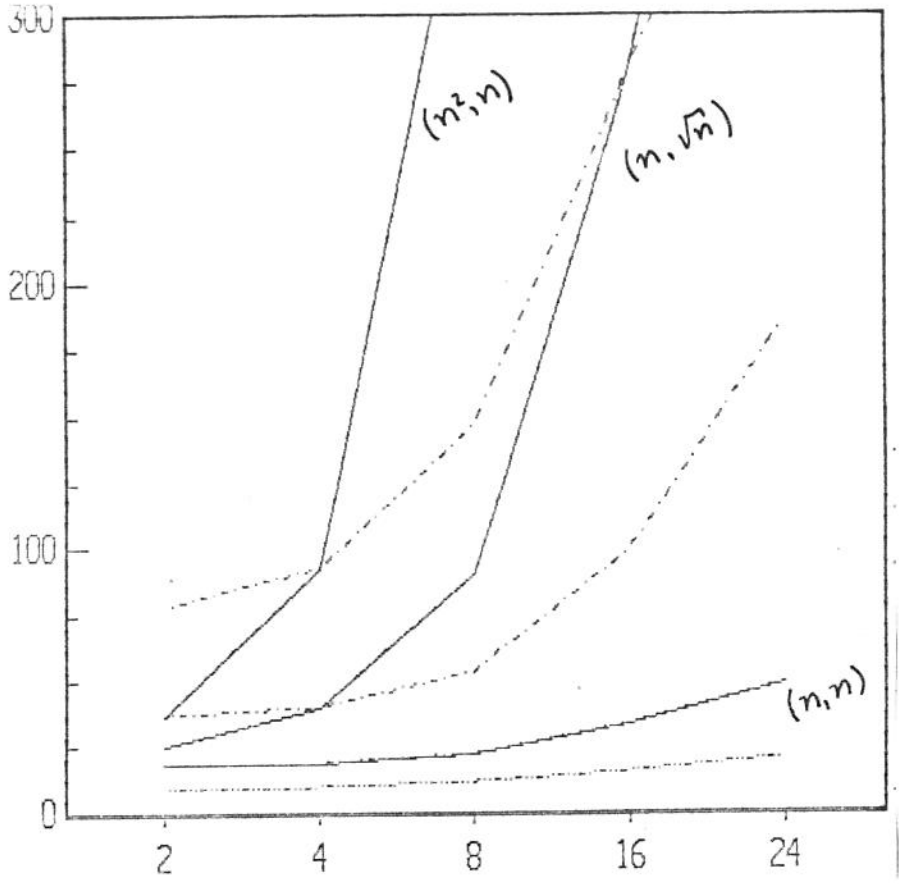


FIG. 19 - Adicional de comunicação na decomposição de um algoritmo ($\rho_0 = 0,1$)

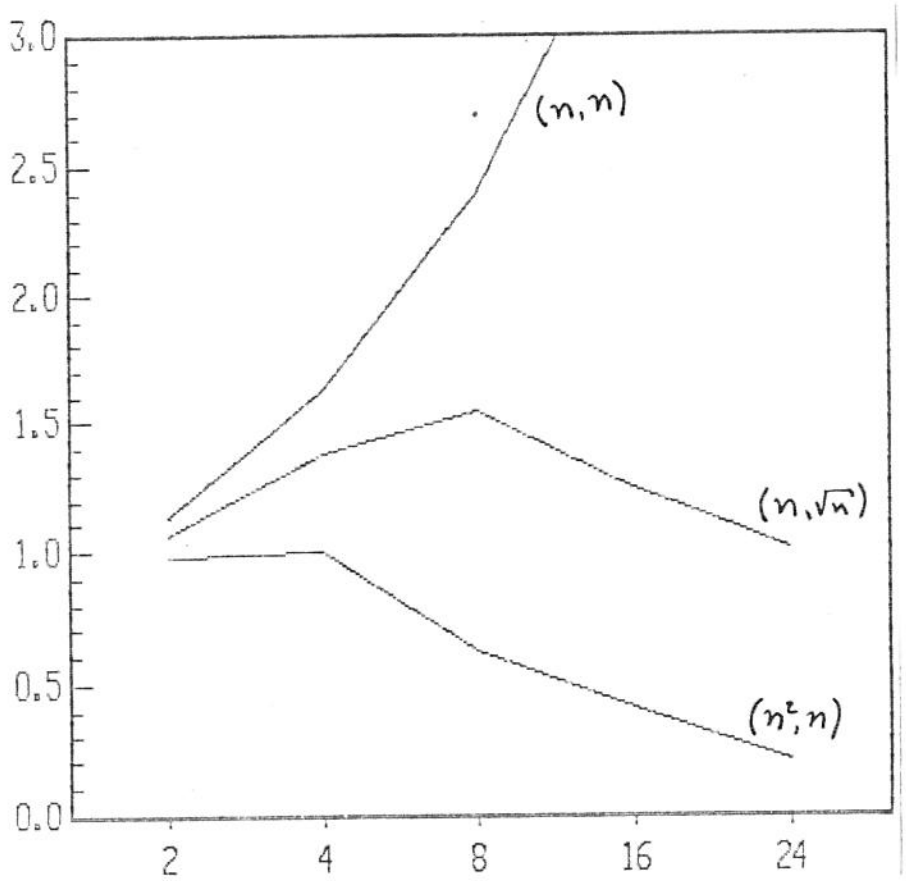


FIG. 20 - Ganho de velocidade na decomposição de algoritmo ($\rho_0 = 0,1$)

CAPÍTULO 5: CONCLUSÕES / TRABALHOS FUTUROS

5.CONCLUSOES / TRABALHOS FUTUROS

Sob algumas hipóteses simplificadoras, o modelo de CMTC proposto neste estudo e o uso de técnicas de simulação permitiu estabelecer um índice de desempenho para comparação entre as arquiteturas radial e por barramento, na execução de estruturas de tarefas hierárquicas sincronizadas. Sob essas mesmas hipóteses, o modelo de CMTC se mostrou apropriado para os dois casos, com erros de aproximação aceitáveis dentro da faixa de números de processadores estudada, escolhido p de forma adequada.

No contexto de processamento paralelo de estruturas hierárquicas sincronizadas, os resultados obtidos poderão ser valiosos para estimar se o tempo de ciclo de um par arquitetura-sistema de tarefas atende as exigências de uma determinada aplicação.

Por outro lado, a aplicação dos resultados ao problema da decomposição de algoritmos que admitem estruturas multinível foi apenas iniciada. Mesmo assim, sob hipóteses de período de processamento com distribuição exponencial foi possível mostrar que a partir de certo número de processadores, o ganho incremental na velocidade de execução torna-se desprezível ou mesmo menor que a unidade, vide Fig.19.

Particularmente, o modelo representado pelas equações (41) e (42) é apenas uma proposta teórica inicial com o objetivo de não passar à margem do problema, mas que é também um ponto de partida para algumas possibilidades de pesquisa.

A partir das hipóteses assumidas durante o estudo e do

contexto de aplicação dos resultados, podemos apontar as seguintes linhas de continuidade deste trabalho:

- estender o tratamento analítico do problema para casos não markovianos. Isto incluiria casos mais realistas como distribuições discretas. Possibilidades são modelos Semi-Markovianos, emprego de Variáveis Suplementares e de Processos Determinísticos por Partes.

-realização de experimentos computacionais sobre um sistema real para avaliar os efeitos dos tempos de arbitramento, propagação e decisão. Este abordagem poderia ser feita usando o conceito de "Carga de Trabalho Sintética"[06].

-realização de experimentos computacionais para estimação das distribuições envolvidas no modelo, para algoritmos que admitem decomposição hierárquica em classes de aplicações particulares.

-estender a análise visando propor arquiteturas específicas para implementação em circuitos integrados dedicados, para a estrutura de tarefas focalizada.

6. BIBLIOGRAFIA

[01] Herzog,U.:"Performance Characteristics for Hierarchically Organized Multiprocessor Computer Systems With Generally Distributed Processing Times", Proc. ITC-9, Torremolinos, Spain, Oct 1979.

[02] Herzog,U.,Hoffmann,W.,Kleinoder,W.:"Performance Modeling and Evalution for Hierarchically Organized Multiprocessor Computer Systems", Int. Conf. Parallel Processing, Bellaire, Michigan, Aug 1979.

[03]Marsan,A.M.,Gerla,M.:"Markov Models for Multiple Bus Multiprocessor Systems", IEEE Trans. Computer, Vol. C-31, n 3, Mar 1982.

[04] Marsan,M.A.,Balbo,G.,Conte,G.:"Comparative Performance Analysis of Single Bus Multiprocessor Architectures", IEEE Trans. Computer, Vol.C-31, n 12, Dec 1982.

[05] Chiola,G.,Marsan,M.A.,Balbo,G.:"Product-Form Solution Techniques for the Performance Analysis of Multiple-Bus Multiprocessor Systems With Nonuniform Memory References", IEEE Trans. Computer, Vol. C-37, n 5, May 1988.

[06] Woodbury,M.H., Shin,K.G.:"Performance Modeling and Measurement of Real-Time Multiprocessor With Time-Shared Buses", IEEE Trans. Computer, Vol.C-37, n 2, Fev 1988.

[07] Nelson,R.,Tantawi,A.N.:" Approximate Analysis of Fork/Join Synchonization in Parallel Queues", IEEE Trans. Computers, Vol C-37, n 6, June 1988.

[08] Nelson,R.,Towsley,D.,Tantawi,A.N.:"Performance Analysis of Parallel Processing Systems", IEEE Trans. Soft. Eng., Vol SE-14, n 4, April 1988.

[09] Saltz,J.H.,Naik,V.K.,Nicol,D.M.:"Reduction of the Effects of the Communication Delay in Scientific Algorithms on Message Passing MIMD Architectures", SIAM J. Sci. and Stat. Computing, Vol.8, n 1, 1987.

[10] Vrsalovic,D.,Gehringer,E.F.,Segall,Z.Z.,Siewiorek,D.P.:"The influence of Parallel Docomposition Strategies on the Performance of Multiprocessor Systems", Proc. 12th Ann. Int. Symp. Comput. Architect., Boston, MA.

[11] Cvetanovic,Z.:"The Effects of Problem Partitioning, Allocation, and Granulatity on the Performance of Multiple-Processor Systems", IEEE Trans. Computer, Vol. C-36, n 4, April 1987.

[12] Kanakia,H.R., Tobagi,F.A.:"On Distributed Computations with Limited Resources", IEEE Trans. Computer, Vol. C-36, n 5, May 1987.

[13] Hwang,K.,Fayé,A.B.:"Computer Architecture and Paralell Processing", McGraw-Hill, 1984.

[14] Moler,C.,Loan,C.V.:"Nineteen Dubios Ways to Compute the Exponential of a Matrix", SIAM REVIEW, Vol 20, n 4, Oct 1978.

[15] Çinlar,E.:"Introduction to Stochastic Processes", Prentice-Hall, Englewood Cliffs, N.J., 1975.

[16] Calzarossa,M.,Serazzi,G.:"A Characterization of the Variation in Time of Workload Arrival Patterns", IEEE Trans. Computer, Vol. C-34, n 2, 1985.

[17] Siegel,L.J.,Siegel,H.J.,Swain,P.H.:"Performance Measures for Evaluating Algorithms for SIMD Machines", IEEE Trans. Software Engineering, Vol. SE-8, n 4, 1983.

[18] Chu,W.W.,Lan,M.T.,Hellerstein,J.:"Estimation of Intermodule Communication(IMC) and Its Applications in Distributed Processing Systems", IEEE Trans. Computer, Vol. C-33, n 8, 1984.

[19] Sethi,A.S.,Deo,N.:"Interferece in Multiprocessors Systems with Localized Memory Acess Probabilities", IEEE Trans. Computer, Vol. C-26, Oct 1977.

[20] James,B.R.:"Probabilidade: Um curso de Nivel Intermediàrio", INPA-Instituto de Mat. Pura e Aplicada, Brasília, 1981.

[21] Kleinrock,L.:"Queueing Systems: Theory, Vol. I", Jonh Wiley & Sons, New York, 1975.

[22] Bowen,B.A.,Buhr,R.J.A.:"The Logical Design of Multiple-Microprocessor Systems", Prentice-Hall, Englemood Cliffs, N.J., 1980.

[23] Shaffer,P.L.:"Parallel Implementation Real-Time Control Programs", Proc. 27th Conf. on Decision and Control, Austin, Texas, Dec 1988.

[24] Flatto,L.,Hahn,S.:"Two Parallel Queues Created by Arrivals with Two Demands I", SIAM J. Appl. Math., Vol.44, Oct 1984.

[25] Flatto,L.:"Two Parallel Queues Created by Arrivals with Two Demands II", SIAM J. Appl. Math., Vol.45, Oct 1985.

[26] Dirickx,Y.M.I.,Jennergren,L.P.:"Systems Analysis by Multilevel Methods with Applications to Economics and Management", John Wiley & Sons, Chichester, 1979.

[27] Polychronopoulos,C.D.,Kuck,D.J.,Padua,D.A.:"Execution of Parallel Loops on Parallel Processor Systems", Proc. Int. Conf. Parallel Processing, 519-527, 1986.

[28] Law,A.M.,Kleton,N.D.:"Simulation Modeling and Analysis", McGraw-Hill, New York, 1982.

[29] Bratley,P.,Bennett,L.F.,Linus,E.S.:"A Guide to Simulation", Springer-Verlang, New York, 1983.

[30] Fisher,M.L.:"The Lagrangian Relaxation Method for Solving Integer Programming Problems", Management Science, Vol.27, Jan 1981.

[31] Armentano,V.A.: Notas de Aula do Curso Pragramação Inteira, Segundo Semestre 1988, UNICAMP-FEE.

[32] Mahmoud,M.S.:"Multilevel Systems Control and Applications: A Survey", IEEE Trans. SMC, Vol. SMC-7, Jan 1977.

[33]

[34] Boyer,L.L.:"Molecular Dynamics of Clusters of Particles Interacting with Pairwise Forces Using a Massively Parallel Computer", J. of Computational Physics, Vol.78, n 2, 1988.

[35] Kadyroba,G.KH.,Sadyrov,S.S.:"Development and Use of Image Processing Networks: A Review", Automatic Control and Computer Sciences", Vol.22, n 2, 1988.

[36] Bessa Maia,J.E.,Ribeiro do Val,J.B.:"Modelo Estocastico para Avaliacao de Desempenho de uma Classe de Algoritmos Paralelos", A ser apresentado no Seminario Franco-Brasileiro em Sistemas Distribuidos", set 1989.

[37] Ribeiro do Val,J.B.,Bessa Maia,J.E.:"Models for Performance Evaluation of Multilevel Algoritms in Parallel Processing", A ser apresentado no IFAC/IFORS/IMACS Symposium on Large Scale Systems 89, Berlin, Ago. 1989.

[38] Ribeiro do Val,J.B.,Bessa Maia,J.E.:"A Stochastic Model for Performance Evaluation of Decentralized Control Parallel Algoritms", submetido ao IFAC Symposium on Power Systems and Power Plant Control, Seoul, Ago. 1989.

APÊNDICE 1:

Prova do Lema 2.

Seja (Ti) a sequencia ordenada dos instantes de transição do processo; qi o estado do processo no instante Ti; I_n o conjunto que indica quais os processadores que estão processando no estado qi; Xm o intervalo desde Ti até o m-ésimo processador concluir seu processamento (se existir), $m \in I_n$; Xc o intervalo desde Ti até o processador que está se comunicando (se existir) concluir sua comunicação.

Vamos calcular:

$$P\{q_{n+1} = q_j, T_{n+1} - T_n > t / q_0, \ldots, q_i; T_0, \ldots, T_n\} =$$

$$P\{q_{n+1} = q_j / q_0, \ldots, q_i; T_0, \ldots, T_n; T_{n+1} - T_n > t\}.$$

$$P\{T_{n+1} - T_n > t / q_0, \ldots, q_i; T_0, \ldots, T_n\} \qquad (A1.1)$$

O segundo termo de (A1.1) pode ser calculado como se segue.

Temos:

$$T_{n+1} - T_n = \min\left[\{X_1, X_2, \ldots, X_m\}, X_c\right] \qquad (A1.2)$$

Pela propriedade "sem memória" da distribuição exponnecial, $\{X_i\}$, $i \in I_n$ é uma coleção de v.a. exponencial i.i.d., uma vez que o tempo já gasto em processamento é irrelevante.

Pela mesma razão a probabilidade do segundo termo de (A1.2) só depende de qn. Então:

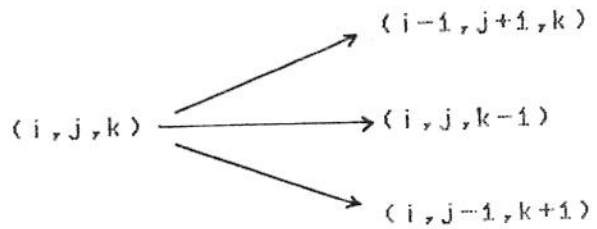
$$P\{T_{n+1} - T_n > t / q_i\} = P\{\min[\{X_1, X_2, \ldots, X_m\}, X_c] > t / q_i\}$$

$$\overset{ind.}{=} \left[P(X_i > t)\right]^m \cdot \left[P(X_c > t)\right]$$

$$= \left[e^{-\lambda t}\right]^m \left[e^{-\lambda_c t}\right] = EXP\left[m\lambda + \lambda_c\right] \quad (A1.3)$$

onde λ e λ_c são as taxas correspondentes a processamento e comunicação.

A probabilidade do primeiro termo de (A1.1) tambem só depende de q_n, pois o dado $T_{n+1} - T_n > t$ é tambem irrelevante pelas mesmas propriedades da distribuição exponencial. Neste ponto observamos que de um dado estado $q_i=(i,j,k)$, o processo admite no máximo duas das seguintes três transições:



Para as transições dos tipos 1 e 2 temos:

$$P\{q_{n+1} = q_j / q_i\} = P\{X_c > \min(X_1, X_2, \ldots, X_m)\}$$

$$= \int_0^\infty P\{X_c > t / \min(X_1, \ldots, X_m) = t\} \cdot dP\{\min(X_1, \ldots, X_m)\}$$

(t / $\min$: ind.)

$$= \int_0^\infty P\{X_c > t\}\, dP\{\min(X_1, X_2, \ldots, X_m)\}$$

$$= \int_0^\infty e^{-\lambda_c t}\, d\left(e^{-m\lambda t}\right) = \frac{m\lambda}{\lambda_c + m\lambda} \triangleq Q(q_i, q_j) \quad (A1.4)$$

Semelhantemente, temos para o terceiro tipo de transição que

$$P\{q_{n+1} = q_j / q_i\} = P\{\{\min(X_1, X_2, \ldots, X_m)\} > X_c\}$$

$$= \frac{\lambda_c}{\lambda_c + m\lambda} = Q(q_i, q_j) \qquad \text{(A1.5)}$$

Concluimos das equações (A1.3), (A1.4) e (A1.5) que para o processo em questão todas as probabilidades de transição são do tipo:

$$P\{q_{n+1} = q_j, T_{n+1} - T_n > t / q_0, \ldots, q_i; T_0, \ldots, T_n\} =$$

$$= Q(q_i, q_j) \exp[m\lambda + \lambda_c] \qquad \text{(A1.6)}$$

Pelo teorema [xx], ref.[15], o processo é uma Cadeia de Markov a Tempo Contínuo.

APENDICE 2: SIMULAÇÃO DE EVENTOS DISCRETOS

Neste apêndice vamos recaptular os principais conceitos e resultados que utilizamos no processo de simulação.

.Números aleatórios: A base matemática para a simulação discreta é a geração de números aleatórios. A precisão dos resultados obtidos tem relação direta com a qualidade do gerador de números aleatórios usado.

Um bom gerador de números aleatórios deve possuir várias propriedades. Os números produzidos devem ser uniformemente distribuidos em [0,1] e não deverão ser correlacionados entre si. Do ponto de vista prático, devemos acrescentar:

-deve ser rápido

-deve usar pouca memória

-deve permitir reproduzir uma sequência de números gerada: primeiro, para facilitar a correção e verificação de programas; segundo, porque podemos querer re-usar os mesmos números aleatórios para aumentar a precisão da simulação[28].

A geração de tais números em computador não é realmente aleatória, uma vez que possui uma lei de formação. Por isto são chamados pseudoaleatórios.

A literatura descreve várias técnicas para geração de números pseudo-aleatórios e testes para os geradores. Usamos o método do gerador multiplicativo[28]. A sequência de números é dada por

$$Z_i = (a.Z_{i-1}).(\mathrm{mod}\ m) \qquad (A2.1)$$

onde m(módulo) e a(multiplicador) são inteiros. Os valores sugeridos em [29] são:

$a = 5^5 = 3125$; $m = 2^b - q$, $b = 35$, $q = 31$

Por outro lado, as linguagens de alto nível incorporam um procedimento para geração de números pseudo-aleatórios U[0,1]. Trabalhamos com PASCAL e fizemos várias simulações usando tanto o gerador da linguagem quanto o gerador multiplicativo e não encontramos diferenças substanciais.

.Variáveis aleatórias: Qualquer simulação envolvendo aspectos aleatórios requer a geração de uma ou mais variáveis aleatórias. Os nossos experimentos envolve apenas a geração de v.a. exponenciais e usamos o método da transformação inversa. O algorítmo é o seguinte:

1. Gerar $u \sim U[0,1]$
2. obter $X = F^{-1}(u)$, onde $F^{-1}(u) = -\beta \, Ln(u)$ se quisermos uma v.a. exponencial com média β^{-1}.

.Análise dos dados de saida: O procedimento comumente usado em simulação é fazer uma corrida de simulação por um tempo suficientemente longo e tratar os resultados da simulação como medidas verdadeiras para o modelo. Uma vez que estas medidas são variáveis aleatórias que podem ter grandes variâncias, em uma simulação particular elas podem diferir significativamente do valor verdadeiro. No entanto, técnicas estatísticas podem ser aplicadas a resutados de simulação para ajudar a ter uma idéia mais precisa das medidas obtidas. O objetivo é construir um intervalo de confiança para o estimador pontual obtido da simulação.

A literatura classifica dois tipos de simulação com respeito à análise dos dados de saida:

.Simulação terminante: existe um evento E que deve ocorrer durante a simulação que determina o tempo T relativo ao qual se define as medidas de desempenho [0,T]. Observe que T é uma variavel aleatória. Neste tipo de simulação a condição inicial do sistema é relevante e deve ser considerada.

.Simulação de estado estacionário: é aquela para a qual as medidas de desempenho são definidas como um limite quando o tempo da simulação vai a infinito.

Para um mesmo sistema, qualquer dos tipos de simulação pode ser apropriado, dependendo do que se deseja saber sobre o sistema.

O nosso problema de aplicação é um problema de simulação terminante(veja apêndice 2), pois o evento "último processador a comunicar" determina o final do ciclo.

3.3.1. SIMULAÇÃO TERMINANTE

Para se obter um procedimento para construção de um intervalo de confiança para a média em simulação terminante, o primeiro passo aqui é usar um resultado da estatística clássica que diz o seguinte: Seja Xi um estimador para a média u de X obtido em uma corrida de simulação. Suponha que realizamos um experimento n vezes e tenhamos n estimadores para a média μ de X. Se os n estimadores $X_1,\ldots,X_n$ são v.a. IID, então o intervalo de confiança de 100(1-α)% para μ é dado por

$$\bar{X}(n) \pm t_{n-1,1-\alpha/2}\sqrt{\frac{s^2(n)}{n}} \qquad (A2.2)$$

onde

$$\bar{X}(n) = \frac{\sum_{i=1}^{n} X_i}{n} \quad \text{e} \quad S^2(n) = \frac{\sum_{i=1}^{n} [X_i - \bar{X}(n)]^2}{n-1}$$

(A2.3)

são estimadores não polarizados da média e variância de X, e $t_{n-1,1-\alpha/2}$ é o ponto crítico superior da distribuição t com n-1 graus de liberdade. Isto será chamado procedimento por Tamanho de Amostra Fixo (Fixed-Sample-Size procedure).

O uso direto deste resultado tem a desvantagem de que ela não estabelece antecipadamente uma relação entre o intervalo de confiança e o número de corridas necessárias, e assim o simulador não tem controle sobre o intervalo de confiança: se um determinado n não atende, não se tem qualquer indicação de qual novo n se deverá testar.

A ref.[28] dá um procedimento sequencial para evitar que realizemos simulações alem do necessário: Defina a precisão relativa para o intervalo de confiança desejado, denotado por γ, como a razão entre a metade do intervalo de confiança (A2.2) e o valor do estimador pontual $\bar{x}(n)$. Então uma expressão aproximada para o número total de repetições $n^*(\gamma)$, necessário para reduzir a precisão relativa do intervalo de confiança a um valor desejado γ, ($0 < \gamma < 1$) é dado por:

$$n^*(\gamma) = \min \left\{ i \geq n : \frac{t_{n-1,1-\alpha/2} \sqrt{\frac{S^2(n)}{n}}}{|\bar{X}(n)|} \leq \gamma \right\} \qquad \text{(A2.4)}$$

(a precisão relativa pode ser pensada como a proporção de μ pela qual $\bar{x}(n)$ difere de μ).

O procedimento é dado a seguir:

Escolha um número de repetições $n_0 \geq 2$, e seja

$$\delta(n,\alpha) = t_{n-1,1-\alpha/2}\sqrt{\frac{s^2(n)}{n}} \qquad (A2.5)$$

0. Faça n_0 repetições da simulação e faça $n = n_0$.

1. Calcule $\bar{x}(n)$ e $\delta(n,a)$ de $X_1, X_2, X_3, \ldots, X_n$.

2. Se $\delta(n,a)/|x(n)| < y$, use

$$I(a,y) = [\,\bar{x}(n) - \delta(n,a),\ \bar{x}(n) + \delta(n,a)\,] \qquad (A2.6)$$

como um intervalo de confiança aproximado de $100(1-\alpha)\%$ para μ , adote $\bar{x}(n)$ como valor estimado, e pare. Caso contrário, faça $n = n + 1$, faça uma simulação adicional e vá ao passo 1.

Note que o número de repetições é uma v.a., uma vez que as estimativas de $\mu(X)$ e $\sigma^2(X)$ são calculadas a cada repetição.

3.3.2. SIMULAÇÃO DE ESTADO ESTACIONÁRIO

Para simulação de estado estacionário, são sugeridos vários procedimentos por Tamanho de Amostra Fixo para construção do intervalo de confiança[28]. Vamos descrever o procedimento mais geral e que tem maior aplicabilidade a problemas reais. É o procedimento por 'médias em lotes'(Batch mean).

Suponha que os dados de saida da simulação $Y_1, Y_2, \ldots$ são observações de um processo estacionário $\{Y_i , i \geq 1\}$ com $E[Y] = \nu$, $\forall$ i. Suponha que fazemos uma simulação de tamanho m e dividimos em n lotes de tamanho l (m=nl). Seja $\bar{Y}_j(l)(j=1,\ldots,n)$ a média por lote e $\bar{\bar{Y}}(n,l) = \sum_{j=1}^{n} \frac{\bar{Y}_j(l)}{n} = \sum_{i=1}^{m} \frac{Y_i}{m}$ a média geral. Usaremos $\bar{\bar{Y}}(n,l)$ como nosso estimador pontual. Se l é suficientemente grande, os $\bar{Y}_j(l),(j=1,\ldots,n)$ serão não correlacionados e aproximadamente normais de mesma média e

variância. Tratamos então $\bar{Y}_j(1)$ como v.a. normais IID com média ν, e construimos um intervalo de confiança aproximado de 100(1-a) para ν usando as relações (A2.2) e (A2.3):

$$\bar{\bar{Y}}(n,\ell) \pm t_{n-1,1-\alpha/2} \sqrt{\frac{S^2_{\bar{Y}_j(\ell)}(n)}{n}} \qquad (A2.7)$$

onde

$$S^2_{\bar{Y}_j(\ell)}(n) = \frac{\sum_{j=1}^{n} \left[\bar{Y}_j(\ell) - \bar{\bar{Y}}(n,\ell)\right]^2}{n-1} \qquad (A2.8)$$

As principais fontes de erro para o intervalo estimado consiste em que raramente o processo $(Y_i, i \geq 1)$ atende as hipóteses de estacionariedade e não correlação, o que fará o estimador em (A2.8) ser polarizado.

Seja agora $\rho_i(1) = \text{cor}[\bar{y}_j(1), \bar{y}_{j+i}(1)]$ a correlação entre duas médias de lotes, cada uma baseada em 1 observações, separadas de i lotes, e seja b(n,1) tal que

$$E\left\{\hat{\sigma}^2\left[\bar{\bar{Y}}(n,\ell)\right]\right\} = b(n,\ell)\,\sigma^2\left[\bar{\bar{Y}}(n,\ell)\right] \qquad (A2.9)$$

onde $$\hat{\sigma}^2\left[\bar{\bar{Y}}(n,\ell)\right] = \frac{S^2_{\bar{Y}_\ell(\ell)}(n)}{n}$$

Pode ser mostrado[28] que $\rho_i(1) \dashrightarrow 0$ $(i=1,\ldots,n-1)$ quando o tamanho do lote $1 \dashrightarrow \infty$, e que isto implica que $b(n,1) \dashrightarrow 1$ quando $1 \dashrightarrow \infty$. Assim, se pudermos desenvolver um método para escolher o tamanho do lote tal que as médias dos lotes sejam essencialmente não correlacionadas, o estimador para $\sigma^2[\bar{\bar{y}}(n,1)]$ será não polarizado.

Uma abordagem sugerida é que fixemos o número n de lotes e então aumentemos o tamanho 1 dos lotes até que o valor estimado

de $\rho_1(1)$ seja suficientemente pequeno. A dificuldade desta abordagem é que os estimadores de correlação são geralmente polarizados e tem variância muito grande para n pequeno. Não raramente leva a requerer um número proibitivo de observações[ ].

A partir de considerações experimentais e de recomendações de outros pesquisadores, [28] sugere o seguinte procedimento sequencial baseado no estudo exaustivo do comportamento de $\rho_1(1)$ para um grande número de aplicações típicas. A idéia é usar fn lotes de tamanho 1 para inferir quando n lotes de tamanho f1 são aproximadamente não correlacionados. Neste procedimento o estimador para $\rho_1(n,1)$ sugerido é:

$$\tilde{\rho}_1(n,\ell) = 2\hat{\rho}_1(n,\ell) - \frac{\hat{\rho}_1^1\left(\frac{n}{2},\ell\right) + \hat{\rho}_1^2\left(\frac{n}{2},\ell\right)}{2} \qquad \text{(A2.10)}$$

onde

$$\hat{\rho}_1(n,\ell) = \frac{\sum_{j=1}^{N-1}\left[\bar{y}_j(\ell) - \bar{\bar{y}}(n,\ell)\right]\left[\bar{y}_{j+1}(\ell) - \bar{\bar{y}}(n,\ell)\right]}{\sum_{j=1}^{n}\left[\bar{y}_j(\ell) - \bar{\bar{y}}(n,\ell)\right]^2}$$

e $\hat{\rho}_1^1$ refere-se às primeiras n/2 amostras, e
$\hat{\rho}_1^2$ refere-se às últimas n/2 amostras.

O procedimento é o seguinte:

0. Fixe $n=40, f=10, m_0 = 600, m_1 = 800$, $u = 0.4$ e uma precisão relativa $y>0$; faça i=1; calcule m observações.

1.a. Divida as m observações em fn lotes de tamanho $l=m/fn$. Calcule $\tilde{\rho}_1(n,1)$ de $\bar{Y}_j(1), (j=1,\ldots,fn)$. Se $\tilde{\rho}_1(fn,1) > u$, vá ao passo 2. Se $\tilde{\rho}_1(fn,1) < 0$ vá ao passo 1.c. Caso contrario vá ao passo 1.b.

1.b. Divida m_i em $fn/2$ lotes de tamanho 21. Calcule $\tilde{\rho}_1(fn/2,21)$ de $\bar{Y}_j(A2.1),(j=1,\ldots,fn/2)$. Se $\tilde{\rho}_1(fn/2,21) < \tilde{\rho}_1(fn,1)$, vá ao passo 1.c. Caso contrário, vá ao passo 2.

1.c. Divida m_i em n lotes de tamanho f1. Calcule $\bar{\bar{Y}}(n,f1)$ e

$$\delta = t_{n-1,1-a/2}\sqrt{\sigma^2[\,y(n,f1)\,]} \quad \text{de } \bar{Y}_j(f1),(j=1,\ldots,n).$$

Se $\delta / |y(n,f1)| < y$, use $\bar{\bar{y}}(n,f1) + \delta$ como um intervalo de confiança aproximado $100(1-\alpha)$ para ν e pare. Caso contrário, vá ao passo 2.

2. Faça: $i = i+1$, $m_i = 2m_{i-1}$, colete $m_i - m_{i-1}$ observações e vá ao passo 1.a.

Para o processo que estamos analisando e o índice de desempenho que estamos interessados, a simulação do tipo terminante é o mais apropriado. Claramente, o evento E = (último processador A conclui sua comunicação na direção A-->B) e que determina o fim do ciclo, é tomado como definido na seção 3.3.1.