ALOCAÇÃO DE TAREFAS EM SISTEMAS DISTRIBUÍDOS

Lysia Maria Monteiro de Barros Canaley

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

Aprovada por:

Prof. Cláudio Luís de Amorim, Ph.D.
(Presidente)

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof. Edil Severiano Tavares Fernandes, Ph.D.

Prof. Adriano Joaquim de Oliveira Cruz, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 1993

CANALEY, LYSIA MARIA MONTEIRO DE BARROS

Alocação de Tarefas em Sistemas Distribuídos
(Rio de Janeiro) - 1993.
VI, 82 p. 29,7 cm (COPPE/UFRJ, M.Sc.,
Engenharia de Sistemas e da Computação, 1993)
Tese - Universidade Federal do Rio de Janeiro,
COPPE.
I. COPPE/ UFRJ II. Título (série).

Agradeço ao meu orientador e à minha família pelo apoio dado para a realização desta tese.

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

ALOCAÇÃO DE TAREFAS EM SISTEMAS DISTRIBUÍDOS

Lysia Maria Monteiro de Barros Canaley

ABRIL, 1993

Orientador: Prof. Cláudio Luís de Amorim

Programa: Engenharia de Sistemas e da Computação

Esta tese trata do problema da alocação de tarefas em sistemas distribuídos, que, fundamentalmente, consiste em, dado um conjunto de tarefas que se comunicam entre si, a ser executado em um sistema de processadores sem memória compartilhada, determinar a qual processador cada tarefa deve ser atribuída.

Estudamos a questão através de duas abordagens bem distintas: o escalonamento por listas de prioridade, com o objetivo de minimizar o tempo total de execução; e a alocação por cortes sucessivos e "clustering", visando a minimização do somatório de todos os custos de execução, comunicação inter-processador e interferência incorridos.

Verificamos a relação entre os objetivos de desempenho considerados e as limitações associadas aos seus modelos. Avaliamos ambos os métodos com relação aos seus próprios objetivos, e também quanto a outras métricas, como volume de comunicação inter-processador e eficiência.

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

TASK ALLOCATION IN DISTRIBUTED SYSTEMS

Lysia Maria Monteiro de Barros Canaley

APRIL, 1993

Chairman: Cláudio Luís de Amorim

Department: Systems and Computing Engineering

This thesis addresses the problem of task allocation in distributed systems, which basically consists of, given a set of communicating tasks to be executed in a distributed configuration of heterogeneous processors, determining to which processor each task should be assigned.

We study the problem using two different approaches: priority list scheduling, where the performance goal is to minimize the completion time of all tasks; and task allocation by successive cuts and clustering, aiming at the minimization of the sum of all execution, inter-processor communication and interference costs incurred.

We investigate the relationship between the performance criteria considered, and the restrictions associated with their models. We also evaluate both methods with respect to their respective goals, and to other metrics, such as inter-processor communication load and efficiency.

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

Este trabalho trata do gerenciamento de tarefas paralelas, um dos mais importantes aspectos de ambientes distribuídos. A questão principal é como distribuir as tarefas entre os processadores de forma a explorar o paralelismo da aplicação e aproveitar os recursos do sistema.

A distribuição de tarefas é fundamental não apenas para a execução eficiente de aplicações em sistemas paralelos, como também para a etapa de projeto de sistemas na qual se investiga quais configurações (número de processadores, topologia da rede, largura de banda dos canais de comunicação, etc.) são capazes de oferecer o nível desejado de desempenho na execução da aplicação.

Estamos interessados em métodos gerais para a alocação estática de estruturas de computação paralela em sistemas multiprocessados distribuídos.

Inúmeros fatores interferem no nível de desempenho de uma alocação. Um dos principais fatores é, sem dúvida, a comunicação inter-processador (IPC), que diz respeito ao atraso pelo trânsito das mensagens nos canais de comunicação, e ocorre, evidentemente, quando tarefas comunicantes são alocadas a processadores diferentes. Quando, ao contrário, tarefas são alocadas a um mesmo processador, elas competem por seus recursos (CPU, memória, serviços de comunicação), incorrendo em

custos adicionais por trocas de contexto, gerenciamento de "buffers" compartilhados, etc.

A questão da determinação de alocações ótimas, em quase todas as suas formulações, pertence à classe de problemas NP-difícil. As exceções constituem casos extremamente restritos. Isto é um forte argumento de que obter soluções ótimas eficientemente é impossível. A estratégia, então, é concentrar-se em algoritmos de tempo polinomial que fornecem soluções subótimas.

Neste estudo, avaliamos e comparamos duas classes de algoritmos heurísticos de alocação com objetivos de desempenho diferentes. Os algoritmos são todos centralizados, com conhecimento global das características da força-tarefa e do sistema distribuído.

O escalonamento por listas de prioridade, com o objetivo de minimizar o tempo total de execução, requer as tarefas representadas através de um grafo de precedências. Estendendo os trabalhos de Shirazi e Wang [SHI90] e de Hwang, Anger, Chow e Lee [HWA90], apresentamos dois algoritmos de escalonamento capazes de tratar atrasos de comunicação relevantes, e processadores heterogêneos interconectados arbitrariamente através de canais com capacidades distintas. Consideramos também a combinação dos algoritmos a alguns critérios heurísticos de prioridade, com a intenção de incrementar a qualidade das soluções produzidas.

A alocação por cortes sucessivos e "clustering", visando a minimização do somatório dos custos de execução, comunicação inter-processador e interferência, utiliza modelos de fluxo de redes. Aqui os processadores são também heterogêneos, mas completamente interconectados através de canais de comunicação idênticos. Os

algoritmos por cortes e "clustering" foram propostos por Lo [LO83, LO88], e podem ser utilizados para ambos os modelos sem e com interferência. Para melhor tratar a interferência, no entanto, sugerimos algumas alterações simples.

A partir de dados gerados aleatoriamente, investigamos a relação entre o tempo total de execução e a soma dos custos de execução, comunicação inter-processador e interferência como critérios de desempenho, e analisamos as respostas de cada método com relação a essas medidas, bem como quanto ao volume de comunicação inter-processador e à eficiência.

De acordo com os nossos experimentos, ambos os métodos considerados se mostraram adequados para a obtenção de soluções satisfatórias quanto aos seus respectivos objetivos. Observou-se ainda o potencial da interferência na redução do tempo de execução, conseqüência indireta da maior distribuição das tarefas entre os processadores. Ainda assim, as soluções ótimas para o modelo com interferência não se mostraram suficientemente boas quanto ao tempo de execução. Este resultado foi ainda pior para as alocações subótimas obtidas na prática pelos algoritmos de cortes e "clustering".

Uma definição básica e um breve histórico do problema encontram-se no capítulo 2.

No capítulo 3, discutimos a alocação de tarefas com o fim de minimizar o tempo total de execução. Primeiramente, apresentamos os modelos utilizados para a representação da aplicação e do sistema distribuído, relacionando as restrições introduzidas pelos modelos e descrevendo superficialmente meios de contorná-las em alguns casos

especiais. Em seguida, descrevemos as heurísticas baseadas em listas de prioridade.

Iniciamos o capítulo 4 com a alocação de tarefas para minimizar o somatório dos custos de execução e comunicação inter-processador. Então, acrescentamos na função objetivo custos de interferência, incorridos quando duas tarefas são atribuídas ao mesmo processador. Os modelos de fluxo de redes e suas limitações são descritos, seguidos dos algoritmos de cortes sucessivos e "clustering".

O capítulo 5 contém uma exposição detalhada das experiências realizadas. Primeiramente, comentamos o processo de geração dos dados de entrada e as medições efetuadas para uso pelos experimentos. Então, apresentamos os procedimentos de testes adotados e analisamos detidamente seus resultados.

Finalmente, as conclusões depreendidas se encontram no capítulo 6.

CAPÍTULO 2

ALOCAÇÃO DE TAREFAS EM SISTEMAS DISTRIBUÍDOS

O problema da alocação de tarefas em sistemas distribuídos consiste, basicamente, em, dado um conjunto de k tarefas $T=\{t_1,...,t_k\}$ que se comunicam entre si, a ser executado em um sistema de n processadores $P=\{p_1,...,p_n\}$ sem memória compartilhada, escolher uma atribuição $\mathcal{A}:T\rightarrow P$ de tarefas a processadores, dentre as n^k possíveis.

Quando se considera ainda a precedência, o instante em que cada tarefa é iniciada, e, portanto, a ordem de execução das tarefas são importantes. Neste caso, o número de soluções possíveis é multiplicado por um termo fatorial. A precedência determina uma relação temporal entre as tarefas, segundo a qual algumas delas podem ser iniciadas apenas após o término de certas outras.

A preocupação deste estudo é com a atribuição estática de tarefas a processadores, seja ela definitiva ou não, isto é, seguida ou não de migração de tarefas em decorrência de mudanças nas cargas dos processadores e na rede de comunicação. Neste tipo de alocação, o comportamento da aplicação e a arquitetura devem ser antecipados em tempo de compilação. Justamente por este motivo, uma de suas principais dificuldades é o tratamento de condicionais. Por outro lado, a alocação dinâmica tem como desvantagem o sobrecusto pelo cômputo da alocação em tempo de execução, o que pode provocar degradação no desempenho.

Existem, na realidade, diversos elementos que afetam o

desempenho da aplicação no sistema distribuído. Vários desses elementos dizem respeito aos custos incorridos entre tarefas que se comunicam, pelo uso do mecanismo de comunicação de processos, contenção para acesso a "buffers", manipulação de filas, sincronização para envio de mensagens, e, quando as tarefas comunicantes estão alocadas a processadores diferentes, pela contenção para uso dos canais de comunicação, e, principalmente, pelo atraso decorrente da transmissão das mensagens trocadas entre as tarefas através dos canais de comunicação entre os processadores. Outros elementos são a quantidade de memória local disponível, a contenção para acesso à memória, e outras características específicas de cada processador.

Por tratabilidade, faz-se necessária uma série de hipóteses simplificadoras. No mais, trabalhamos com aproximações extraídas de uma análise da aplicação e da arquitetura. Tal análise é complicada, e os fatores considerados são difíceis de serem quantificados. Mas supomos que, de algum modo, essas quantias estão disponíveis, expressas em unidades de medida convenientes.

A qualidade de uma solução pode ser avaliada sob vários aspectos, através de métricas diversas, dentre as quais destacam-se o tempo total de execução, com o aproveitamento do paralelismo potencial da aplicação, o atraso por comunicação inter-processador (que possui impacto dramático no desempenho), o balanceamento da carga computacional e a utilização das CPU's e dos demais recursos do sistema.

De fato, a alocação de tarefas em sistemas distribuídos deveria, idealmente, cumprir requisitos opostos (embora relacionados): o balanceamento de carga e a minimização do volume de comunicação

inter-processador; a maximização do paralelismo e a preservação da ordem de precedência na execução das tarefas.

A política de alocação ótima deve ser um compromisso entre esses requisitos, os quais, obviamente, não podem ser satisfeitos ao mesmo tempo. Nesse sentido, deve-se definir objetivos para a avaliação das alocações e seus algoritmos.

Diferentes métodos podem estabelecer objetivos distintos, expressos por funções de decisão que reduzem o espaço de soluções durante a alocação. Conseqüentemente, as estratégias de alocação são extremamente influenciadas pelas funções objetivo. Dentre as muitas funções já adotadas, podemos citar aquelas baseadas em volume de comunicação inter-processador [GYL76, EFE82, LEE87], no número de tarefas comunicantes mapeadas em processadores adjacentes (denominado cardinalidade) [BOK81a], no custo de execução e comunicação inter-processador total [STO77, LO83, SIN84, EZZ86, SIN87, LO88] e máximo por processador [LO83, SHE85, CHU87, KHE88, KIM88], e no tempo total de execução, dado pelo instante em que termina a última tarefa da aplicação, assumindo tarefas independentes ou não, contando com atrasos de comunicação ou não [GRA69, KRU87, HWA89, LEU89, REW90, SHI90]; cabe observar que o custo de execução e comunicação máximo por processador também pode ser percebido como uma definição do tempo total de execução [LO83, LO88]. As funções objetivo podem estar sujeitas a restrições adicionais como quanto à utilização de memória, distribuição da carga de trabalho, tempo de resposta, entre outras.

Não obstante, há um consenso geral quanto ao fato de que a redução do volume de comunicação inter-processador, o balanceamento de

carga e o incremento de paralelismo são metas desejáveis na determinação de alocações.

Mesmo impondo diversas limitações na formulação do problema, encontrar uma solução ótima para n e k arbitrários é NP-completo. Problemas NP-completos são tais que é improvável que existam algoritmos computacionalmente eficientes (tempo polinomial) para resolvê-los.

Em particular, quando o objetivo da alocação é a otimização do tempo total de execução, o problema foi demonstrado ser NP-completo até para tarefas independentes, com tempos de execução arbitrários [KAR72]. Quando se considera precedência entre as tarefas, até mesmo o caso restrito do escalonamento de tarefas com tempos de execução unitários é NP-completo [ULL75], assim como o escalonamento de tarefas com tempos arbitrários de execução em sistemas de n=2 processadores [PRA87]. De fato, a única situação para a qual é viável a resolução ótima do problema com respeito ao tempo total de execução é o caso trivial de tarefas independentes com tempos de execução idênticos.

Também foi mostrado que, para a alocação de tarefas comunicantes, não relacionadas por precedência, em sistemas de n>2 processadores heterogêneos, quando os atrasos por transferência de mensagens entre os processadores são considerados relevantes, a determinação de uma solução com custo total mínimo de execução e comunicação inter-processador é NP-completo [BOK81b].

No restante deste capítulo, revemos em linhas gerais diversas de estratégias propostas na literatura para a resolução do problema.

2.1. Estratégias de Alocação Ótimas

Soluções ótimas são importantes para aplicações executadas repetidamente em um mesmo ambiente distribuído.

A enumeração exaustiva de todas as possíveis soluções obviamente encontra as alocações ótimas. No entanto, este procedimento requer, na melhor hipótese (quando não se considera precedência), tempo da ordem de $O(n^k)$, sendo proibitivo mesmo para n e k pequenos.

Conseqüentemente, foram estudados meios mais eficientes de se obter soluções ótimas. Basicamente, duas abordagens são utilizadas para a resolução ótima do problema: por programação inteira, e através de modelo e algoritmos de fluxo de redes.

Na abordagem por fluxo de redes de Stone [STO77], a alocação tem por objetivo minimizar o somatório dos custos de execução e comunicação inter-processador, uma medida da utilização geral dos recursos do sistema. A aplicação e o sistema são modelados por uma rede, onde cada corte corresponde a uma alocação distinta de tarefas a processadores, e a capacidade do corte equivale ao custo da alocação. Portanto, tem-se uma alocação ótima com a obtenção de um corte de capacidade mínima na rede, mas este procedimento é viável apenas para n=2. O problema da determinação de um corte n-way mínimo, n>2, é NP-completo [GUR81].

Mesmo para n=2, este método possui algumas limitações críticas. O balanceamento de carga não faz parte do seu objetivo. Os atrasos nos canais de comunicação que interconectam os processadores devem ser uniformes (isto implica em processadores conectados através de um barramento de dados ou em um conjunto de processadores completamente interconectado por canais idênticos), e relações de dependência ou restrições dos recursos não são consideradas.

Uma extensão do método para *n* arbitrário e atrasos de comunicação não uniformes foi proposta por Stone e Bokhari [STO78] para casos em que o padrão de referências entre módulos tem a estrutura de uma árvore.

Outra possibilidade é a busca de soluções através do método "Branch-and-Bound" [HOR78, CHU80], conhecida técnica de programação inteira 0-1 não linear que procura abreviar o espaço de soluções, o qual é representado por uma árvore. Funções subestimativas de custo servem para guiar e terminar a expansão da árvore de soluções. A função a ser otimizada pode ser, também, a soma dos totais de execução e comunicação inter-processador, possivelmente sujeita a restrições como na distribuição de carga, na quantidade de memória local dos processadores, ou no tempo de resposta.

Ma et al. [MA82] utilizaram um algoritmo "Branch-and-Bound" para obter alocações ótimas com a imposição de várias restrições inerentes às aplicações, e responsáveis pela contração do espaço de soluções viáveis. Sinclair [SIN84, SIN87] também estudou a aplicação do método ao problema da alocação, propondo e avaliando funções subestimativas locais e globais, e sugerindo algumas técnicas adicionais para incrementar a redução da árvore de soluções, como a combinação com o método de corte mínimo de Stone (no caso de atrasos de comunicação uniformes), e a classificação de tarefas, de modo a alocar as tarefas mutualmente independentes somente após as demais.

A utilização deste método é limitada pelo fato de que as quantidades de tempo e memória necessárias para a determinação de uma solução ótima são funções exponenciais da magnitude do problema. Além

disso, não há provisão para a especificação de relações de dependência.

2.1. Estratégias de Alocação Subótimas

A complexidade computacional de estratégias de alocação ótimas as tornam impróprias para aplicações onde o fator tempo é crítico. Há também casos em que uma solução ótima não é estritamente necessária.

Daí a importância de heurísticas para a obtenção de alocações subótimas, as quais devem ser simples, rápidas e, em virtude da natureza imprecisa da medição dos fatores envolvidos, pouco sensíveis a pequenas variações na entrada. Algoritmos heurísticos podem utilizar também modelos de fluxo de rede ou da programação matemática, com a vantagem de, em geral, permitir a incorporação de um número maior de elementos pertinentes ao problema (já que são menos sensíveis às suas dimensões).

A partir do trabalho de Shen e Tsai [SHE85], Khei [KHE88] apresentou um método subótimo de busca de estados A*, com a limitação da lista de nós a expandir (abertos). O algoritmo começa com uma ordenação das tarefas a fim de privilegiar na alocação as tarefas com maior demanda de comunicação. Khei também sugeriu alterações na função de custo, baseada no total de execução e comunicação máximo por processador, cuja motivação foi favorecer o balanceamento de carga. Propôs, ainda, uma técnica de redes neuronais para a otimização da função.

Stone [STO77] generalizou seu modelo de rede para $n>2$ processadores e criou um algoritmo subótimo para sistemas com $n=3$. Wu e Liu [WU80] apresentaram, para o mesmo critério de minimização dos custos totais de execução e comunicação, algoritmos subótimos para n

arbitrário. Em ambos os casos, a idéia foi invocar repetidamente o algoritmo de Fluxo Máximo/ Corte Mínimo em porções da rede.

Lo [LO83, LO88] propôs outro algoritmo para n arbitrário, que, em síntese, consiste em reduzir sucessivamente a rede para n=2 e obter o corte mínimo da rede reduzida. Como esse algoritmo produz, no caso geral, uma atribuição apenas parcial, Lo sugeriu sua combinação com um algoritmo do tipo guloso. Além disso, com o propósito de estimular diretamente o balanceamento de carga nas alocações, Lo introduziu no modelo um novo fator, ao qual deu o nome de interferência, que reflete os custos relacionados com a alocação de tarefas a um mesmo processador.

Para resolver o problema quando o objetivo é minimizar o tempo que o conjunto de tarefas, relacionadas por precedência, leva para terminar, os métodos mais adequados pertencem à conhecida classe de algoritmos de escalonamento por listas de prioridade. A idéia básica é impor uma ordem à alocação, privilegiando neste processo determinadas tarefas, de acordo com algum critério heurístico. Os algoritmos de listas de prioridade podem ser preemptivos ou não, e permitir ou não replicação de tarefas, sendo que a maioria assume atrasos de comunicação irrelevantes [GRA69, SHI90]. Entre os que tratam da comunicação inter-processador, destacamos os propostos por Kruatrachue [KRU87], Hwang, Chow, Anger e Lee [HWA89], e El-Rewini e Lewis [REW90].

Diversas outras heurísticas para problemas de alocação em sistemas distribuídos constam na literatura, as quais podem envolver teoria de filas [EZZ86], sistemas probabilísticos [CHO82], técnicas de perturbação [BOK81a, LEE87, KIM88, ERC90], "clustering" [GYL76, CHU80, EFE82, LO83, LO88], etc.

CAPÍTULO 3

ALOCAÇÃO PARA MINIMIZAR O TEMPO TOTAL DE EXECUÇÃO

Neste capítulo, investigamos a atribuição de tarefas comunicantes a um sistema de processadores com o objetivo de executar as tarefas em tempo mínimo.

Para a minimização do tempo total de execução, é importante não só determinar a qual processador cada tarefa será alocada, mas também o instante em que cada tarefa deverá iniciar, o que requer as tarefas representadas por grafos de precedências.

Os métodos mais adequados para resolver o problema assim formulado pertencem à conhecida classe de algoritmos de escalonamento por listas de prioridade. A idéia básica é impor uma ordem nas alocações, privilegiando neste processo determinadas tarefas, segundo algum critério heurístico.

Em particular, utilizamos algoritmos de listas de prioridade não preemptivos, baseados em propostas de Shirazi e Wang [SHI90], e de Hwang, Chow, Anger e Lee [HWA89], adaptadas conforme necessário para o nosso modelo de sistema distribuído, que assume processadores heterogêneos, interconectados arbitrariamente, por canais de comunicação com capacidades diferentes, e considera relevantes os atrasos de comunicação.

3.1. Conceitos Básicos

Um *grafo de precedências* $G_T(N_T,E_T)$, é um grafo direcionado cujos nós consistem em um conjunto de tarefas ordenado parcialmente [KAR72]. Denotamos $t_i < t_j$ se $e_{ij} \in E_T$, e chamamos t_i e t_j de *origem* e *destino*, respectivamente. Em G_T, *raiz* é todo nó que não possui predecessores e *folha* é todo nó que não precede nó algum.

Seja $G_P(N_P,E_P)$ um sistema de processadores, sobre o qual G_T deve ser mapeado. O *tempo acumulado* ou *carga* de um processador $p_q \in N_P$, ACM_q é o tempo total necessário para p_q terminar todas as tarefas alocadas a p_q.

Uma tarefa $t_i \in N_T$ é dita *madura* em um dado instante se está pronta para ser alocada a um processador. Isto é verdade se a tarefa t_i não tem pais no grafo G_T, ou se cada um de seus pais já foi alocado a algum processador $p_q \in N_P$, e ACM_q é menor ou igual que o instante considerado. Assim, asseguramos que uma tarefa não será alocada antes que todos os seus predecessores tenham terminado. Aqui não se contam eventuais atrasos de comunicacão entre t_i e seus pais, o que não é definido até que se especifique algum processador $p_q \in N_P$ para a alocação de t_i.

O instante em que t_i pode efetivamente começar a executar em p_q, que chamaremos de INI_{iq}, é o instante em que t_i recebe a última das mensagens enviadas imediatamente após o término de seus pais. Isto depende de quando os pais de t_i terminaram e aonde estão alocados, e quanto tempo tomam os atrasos de comunicação. Observe que INI_{iq} nunca é anterior ao instante em que t_i se torna madura. O instante mais cedo em que t_i é capaz de terminar em p_q, FIM_{iq}, é dado por $INI_{iq} + x_{iq}$.

3.2. Formulação do Problema

Sabemos que o objetivo aqui é obter uma solução tal que o tempo total de executar a aplicação no sistema distribuído seja ótimo (ou subótimo). O tempo de execução correspondente a uma solução $\mathcal{S}$ é dado por:

$$\mathcal{C}_1(\mathcal{S}) = \max_{p_q \in N_P} \{ACM_q\}$$

Começamos esta seção com a definição do contexto assumido para o problema, relacionando nossas suposições a respeito da aplicação, do sistema distribuído e do esquema de alocação. Em seguida, é apresentada a descrição formal dos modelos da aplicação e do sistema distribuído, juntamente com a notação e a terminologia adotadas. As limitações dos modelos são, então, levantadas e discutidas.

3.2.1. Hipóteses

A computação distribuída é especificada em termos de tarefas e de dependências entre essas tarefas, sendo, por este motivo, também referenciada neste trabalho pela denominação de força-tarefa.

Tarefas são seqüências arbitrárias de operações em dados, a diferentes níveis de abstração, podendo ser constituídas, por exemplo, por coleções quaisquer de instruções ou subrotinas.

As tarefas são relacionadas por precedência, de modo que uma tarefa não pode iniciar até que tenha recebido todas as mensagens de cada tarefa precedente. Por este motivo, o seqüenciamento entre as tarefas não pode formar ciclos. Assumimos, ainda, que todas as mensagens são enviadas logo após o término da tarefa de origem.

Supomos que a computação já se encontra adequadamente particionada em tarefas coesas. Ou seja, a quantidade de inter-referências deve ser reduzida. Tarefas intensamente dependentes entre si devem ser aglutinadas, transformadas em uma única tarefa.

O particionamento da computação em tarefas, de fundamental importância no processamento paralelo e distribuído, constitui problema complexo por si só, merecendo, portanto, tratamento independente da alocação.

Os recursos do sistema são descentralizados. Não existe memória global à qual todos os processadores têm acesso. Cada processador tem memória suficiente para comportar o armazenamento e a execução de todos os módulos da computação.

O atraso por transmissão de mensagens entre processadores é relevante em comparação com os tempos de execução das tarefas. Os canais de comunicação entre os processadores são bidirecionais e simétricos, assim sendo, o tempo para transmitir uma mensagem de um processador para o outro é idêntico ao tempo de transmitir a mesma mensagem na direção reversa. São ainda independentes, podendo a comunicação entre um par de tarefas sobrepor-se à comunicação e ao processamento de outras tarefas. Além disso, os canais têm capacidade suficiente para servir toda a comunicação sem atraso significativo por contenção, e, assim como os processadores, não são sujeitos a falhas; como conseqüência, a transmissão das mensagens é feita sempre através das rotas de menor distância entre cada par de processadores (rotas alternativas de maior comprimento não são consideradas).

Uma tarefa pode ser alocada a qualquer processador, mas, devido

a possíveis diferenças nas velocidades e em outros atributos dos processadores, o tempo de execução de uma tarefa pode variar conforme o processador.

Os protocolos de comunicação são livres de colisão, de modo que não há perda de mensagens e todas as mensagens enviadas alcançam o destino em um intervalo de tempo finito. Qualquer custo adicional de protocolo de canal é considerado na estimativa de sua largura de banda.

Não há replicação de tarefas, sendo cada tarefa alocada a um único processador.

3.2.2. Modelagem da Força-Tarefa e do Sistema Distribuído

Os modelos que utilizamos incorporam todas as informações necessárias à definição da seqüência de eventos de uma computação e à sua alocação no sistema distribuído. Através deles, várias classes de estruturas de computação paralela e diferentes arquiteturas podem ser representadas.

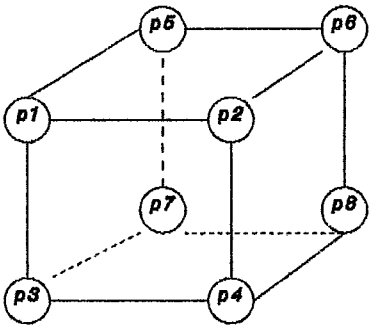
Um sistema distribuído de $n>2$ processadores heterogêneos $P=\{p_1,...,p_n\}$ interconectados arbitrariamente pode ser modelado por uma dupla (G_P,L).

$G_P(N_P,E_P)$ é um grafo não direcionado, cujos nós $N_P=P$ são os processadores do sistema e as arestas E_P representam a topologia da arquitetura, logo, cada aresta $e_{qr} \in E_P$ indica que existe um canal de comunicação físico entre p_q e p_r.

A função L associa a cada par (p_q,p_r), p_q, $p_r \in N_P$, o peso $l_{qr}>0$, $l_{qr} \in \mathbb{R}$, quantidade máxima de comunicação que pode ser transmitida de p_q para p_r dentro de uma unidade de tempo. O valor l_{qr} equivale à largura

de banda de um suposto canal de comunicação entre os processadores p_q e p_r.

Tomemos por exemplo uma arquitetura hipercúbica, que não provê conexões diretas entre todos os pares de processadores. Seja 1 a largura de banda dos canais de comunicação físicos. Aos pares de processadores que não são diretamente interligados, que em uma arquitetura hipercúbica distam de 2 ou 3 unidades, a função L atribuirá o valor $1/2$ ou $1/3$, conforme o caso.



Largura de Banda								
L	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
p_1	-	1	1	1/2	1	1/2	1/2	1/3
p_2	1	-	1/2	1	1/2	1	1/3	1/2
p_3	1	1/2	-	1	1/2	1/3	1	1/2
p_4	1/2	1	1	-	1/3	1/2	1/2	1
p_5	1	1/2	1/2	1/3	-	1	1	1/2
p_6	1/2	1	1/3	1/2	1	-	1/2	1
p_7	1/2	1/3	1	1/2	1	1/2	-	1
p_8	1/3	1/2	1/2	1	1/2	1	1	-

Figura 3.1: Modelagem (G_p,L) de um hipercubo de dimensão 3

Desta forma, a distância entre os processadores e a velocidade dos canais são englobadas pela função L. Vale lembrar que, pelas nossas hipóteses, a transmissão das mensagens é feita sempre através das rotas de menor distância entre cada par de processadores.

O modelo de uma computação paralela, composta de um conjunto de tarefas $T=\{t_1,...,t_k\}$ que se comunicam entre si, consiste em uma tripla (G_T,X,C).

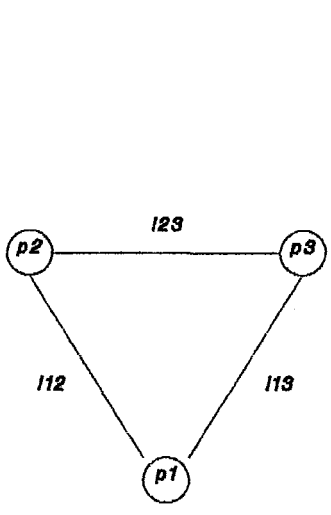
$G_T(N_T,E_T)$ é um dígrafo acíclico, cujos nós $N_T=T$ são as tarefas e cujas arestas E_T representam transmissão de dados e/ ou informação de sincronização. Denotamos $t_i < t_j$ se $e_{ij} \in E_T$, e chamamos t_i e t_j de origem e destino, respectivamente.

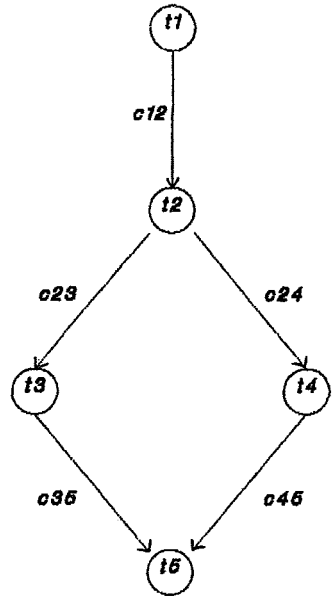
A cada par de nós (t_i,t_j), $e_{ij} \in E_T$, a função C atribui um valor $c_{ij} > 0$, $c_{ij} \in \mathbb{N}$, correspondente à quantidade de comunicação entre t_i e t_j.

Assim, a existência de uma aresta direcionada e_{ij} contém, em conjunto com a função C, toda a informação de seqüenciamento e de dependência de dados entre t_i e t_j.

As funções C e L devem ser expressas em unidades de medida compatíveis. Por exemplo, se C é expresso em número de bytes, L deve ser expresso em número de bytes por unidade de tempo.

A cada par (t_i,p_q), $t_i \in N_T$ e $p_q \in N_P$, é associado, pela função X, um peso $x_{iq}>0$, $x_{iq} \in \mathbb{N}$, que equivale ao tempo de execução de t_i em p_q. De fato, sendo o sistema composto de processadores distintos, o tempo de processamento de uma tarefa varia conforme os atributos específicos do processador ao qual está alocada, tais como velocidade, conjunto de instruções, presença de memória cache, hardware de ponto flutuante, etc.





Largura de Banda			
L	p_1	p_2	p_3
p_1	-	1	1
p_2	1	-	1
p_3	1	1	-

Execução					
X	t_1	t_2	t_3	t_4	t_5
$\overline{p}$	2	4	3	4	4
p_1	2	4	3	2	6
p_2	2	4	2	4	5
p_3	2	4	4	6	1

Comunicação					
C	t_1	t_2	t_3	t_4	t_5
t_1	-	4	0	0	0
t_2	-	-	2	1	0
t_3	-	-	-	0	1
t_4	-	-	-	-	1
t_5	-	-	-	-	-

Figura 3.2: Exemplo de (G_P, L) e (G_T, X, C)

Representamos por $\bar{x}_i$ o tempo de computação médio de t_i nos processadores do sistema, isto é, $\bar{x}_i = \frac{1}{n} \sum_{p_q \in N_P} x_{iq}$.

A figura 3.2 contém um exemplo de força-tarefa e sistema distribuído modelados da forma descrita acima, baseado no qual a figura 3.3 mostra um escalonamento ótimo, representado através de um diagrama de Gantt.

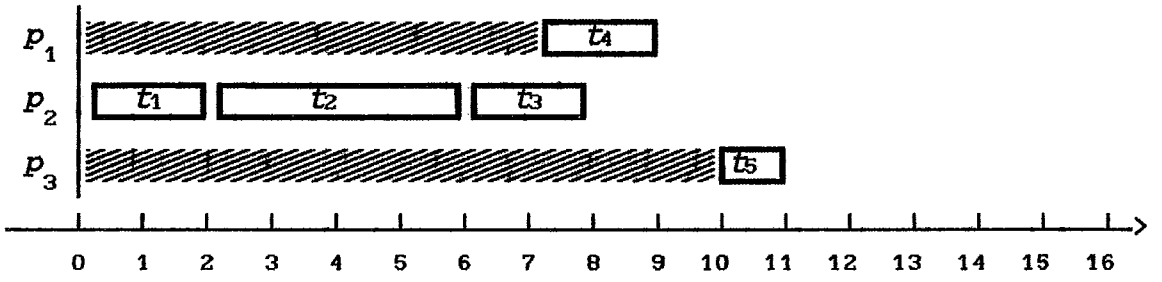


Figura 3.3: Escalonamento ótimo de (G_T, X, C, I) em (G_P, L)

Os processadores são representados no eixo vertical e o tempo é representado no eixo horizontal. Para cada processador, o intervalo de tempo correspondente à computação de cada tarefa que lhe é atribuída é rotulado com a identificação da tarefa, e o tempo restante aparece hachurado.

Segundo este escalonamento, t_1 está madura no instante inicial 0, t_2 se torna madura em 2, t_3 e t_4 em 6, e t_5 em 9. Ao fim do escalonamento, os tempos acumulados de p_1, p_2 e p_3 são, respectivamente, 9, 8, e 11. O escalonamento ótimo tem tempo total de execução igual a 11 unidades de tempo.

3.2.3. Restrições

Uma limitação do modelo da computação distribuída é que o grafo G_T não pode conter ciclos. Ciclos seqüenciais devem ser embutidos dentro de tarefas. Os ciclos cujos passos são executáveis em paralelo devem ser expandidos, e as tarefas resultantes devem ter seus tempos de computação e comunicação ajustados. Há ainda casos especiais em que um ciclo pode ser desfeito através de esquemas mais convenientes, os quais buscam remover a aresta que fecha o ciclo [MAR67, KIM88], evitando, assim, sua expansão completa.

Além disso, uma aresta $e_{ij} \in E_T$ constitui uma via de comunicação unidirecional de t_i para t_j ($t_i < t_j$). Isto implica na não representação da ocorrência de troca de mensagens a nível de tarefas.

As restrições acima estão ligadas ao uso de um grafo de precedências para a representação das tarefas. A necessidade de eliminar qualquer incerteza na determinação dos requerimentos para execução da aplicação no sistema distribuído impõe algumas restrições adicionais.

A configuração do sistema distribuído é estática, tal como as características da força-tarefa, ou seja, os grafos G_P e G_T e as funções L, e X, C e I não variam em tempo de execução. "Amarrações" dinâmicas entre operações e dados podem ocorrer apenas dentro das tarefas, de forma que não há alterações estruturais nos grafos (nem nós nem arestas novas são criados em tempo de execução).

Outra conseqüência da proibição de ambigüidade na definição da computação diz respeito à dificuldade na representação de condicionais. Em alguns casos, os desvios de maior probabilidade podem ser assumidos na representação da computação; alternativamente, os desvios condicionais são encerrados dentro das tarefas.

3.2.4. Outras Considerações

Supomos que os dados necessários para a construção dos grafos G_P e G_T e para a definição das funções L, C e X estão disponíveis, e encontram-se expressos em unidades de medida convenientes.

Esses dados devem ser derivados de uma análise das características da força-tarefa e do sistema distribuído.

Valores iniciais podem, por exemplo, ser diretamente especificados pelo programador ou deduzidos automaticamente pelo compilador.

Em particular, é uma aproximação razoável assumir que o tempo de execução de uma tarefa em um processador é o produto de uma constante derivada do número e tipo de instruções contidas na tarefa por uma medida da potência relativa do processador, sem levar em consideração, a princípio, a natureza precisa da tarefa e os atributos específicos do processador. Da mesma forma, os custos de comunicação podem ser aproximados a partir de informações tais como o número e o tipo de mensagens trocadas entre as tarefas.

Através da monitoração de execuções da força-tarefa, informações adicionais podem ser deduzidas para o refinamento das estimativas iniciais.

3.3. Algoritmos de Listas de Prioridade

Para a minimização do tempo total de execução, utilizamos heurísticas de listas de prioridade, baseadas nos trabalhos de Hwang, Chow, Anger e Lee [HWA89], Shirazi e Wang [SHI90]. Trata-se de dois algoritmos de escalonamento e quatro critérios auxiliares de atribuição

de prioridade. No capítulo 6, avaliaremos o desempenho de cada combinação resultante.

3.3.1. Escalonamento Simples

As tarefas são alocadas à medida que vão amadurecendo. Conflitos surgem sempre que existir mais de uma tarefa madura em um dado instante (isto é, se houver algum paralelismo na aplicação), e se resolvem arbitrariamente ou através de outras heurísticas.

Originalmente, cada tarefa era alocada ao processador com menor tempo acumulado no momento da decisão [SHI90]. Na nossa adaptação para configurações arbitrárias de processadores heterogêneos e custos de comunicação inter-processador relevantes, a opção é pelo processador no qual a tarefa terminará mais cedo (levando-se em consideração os atrasos de comunicação).

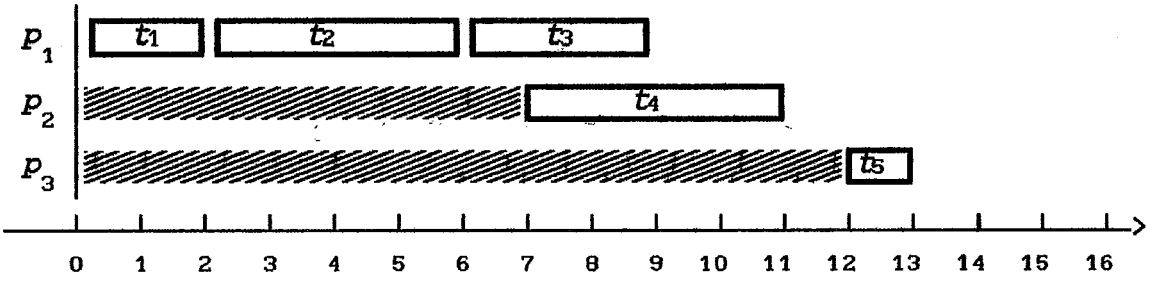


Figura 3.4: Solução Completa de *ESCALONAMENTO-SIMPLES*

Exemplificamos o funcionamento de *ESCALONAMENTO-SIMPLES* com a força-tarefa e o sistema distribuído da figura 3.2. As tarefas t_1 e t_2 são alocadas a um mesmo processador, digamos p_1. Com o fim de t_2, no instante 6, t_3 e t_4 se tornam maduras. Supondo que o algoritmo opte por

alocar t_3 primeiramente (decisão arbitrária ou baseada em algum critério heurístico), t_3 é alocada a p_1, pois este é o processador capaz de terminá-la mais cedo (no instante 9). Então, t_4 seria alocada a p_2, terminando no instante 11, e, por fim, t_5 seria alocada a p_3, terminando no instante 13. O escalonamento completo é mostrado na figura 3.4.

```
procedimento ESCALONAMENTO-SIMPLES
    INSTANTE ⇐ 0;
    repita
        seja T* ={t_i tq. t_i ∈ N_T madura em INSTANTE};
        enqto T* ≠ ∅ faça
            escolha t_i ∈ T* arbitrariamente ou por critério auxiliar;
            escolha p_q ∈ N_P tq. FIM_iq ≤ FIM_ir, ∀ p_r ∈ N_P;
            escalone t_i a p_q no instante INI_ir;
        INSTANTE ⇐ instante em que a próxima tarefa termina;
    ate que INSTANTE = ∞;
fim
```

Cada uma das k tarefas é examinada somente uma vez; para cada tarefa, são considerados no máximo n processadores. Logo, o algoritmo toma tempo da ordem $O(nk)$. Se há emprego de algum critério de adicional de diferenciação de tarefas para a dissolução de conflitos, a complexidade resultante é dada pela maior entre $O(nk)$ e a complexidade correspondente ao critério.

3.3.2. Escalonamento por Término

A versão original (chamada "Earliest Ready Task"), que supõe processadores idênticos, privilegia as tarefas capazes de resolver suas dependências e iniciar mais cedo em algum processador do sistema, contando com os atrasos de comunicação [HWA89].

Pela nossa adaptação, passam a ter prioridade as tarefas capazes de terminar mais cedo. Isto é, a tarefa madura $t_i \in N_T$ capaz de terminar mais cedo em algum processador p_q do sistema é considerada para alocação imediata (em p_q).

Como se supõem processadores com velocidades diferentes e atrasos de comunicação arbitrários, uma tarefa que ainda não está madura pode ser capaz de terminar, em algum processador do sistema, antes do que a tarefa selecionada. Assim, sempre que é selecionada para a alocação uma tarefa t_i, cujo término se daria em um momento posterior ao término de alguma outra tarefa t_j, adiamos a decisão até o término de t_j para examinar as tarefas (se alguma) que se tornaram maduras nesse momento.

Quando há tarefas capazes de terminar mais cedo em um mesmo instante, a decisão é pela tarefa que, para tanto, deve iniciar antes (maior tempo computacional). Se ocorre nova coincidência, uma heurística auxiliar pode ser empregada.

Utilizamos novamente a figura 3.2. *ESCALONAMENTO-POR-TERMINO* pode alocar ambas as tarefas t_1 e t_2 ao processador p_1. Admitindo essa hipótese, o algoritmo prossegue com a decisão de alocar t_4 antes de t_3, pois t_4, alocada a p_1, termina no instante 8, mais cedo do que t_3 poderia terminar em qualquer processador. Agora é considerada para alocação imediata a tarefa t_3, capaz de terminar em p_2 no instante 10. O

algoritmo espera até o término de t_4, no instante 8 < 10, mas acaba concretizando a alocação (com o fim de t_4, nenhuma tarefa se torna madura). Finalmente, t_5 é alocada a p_3, e o tempo de execução resultante é de 12 unidades. Vemos esse escalonamento na figura 3.5.

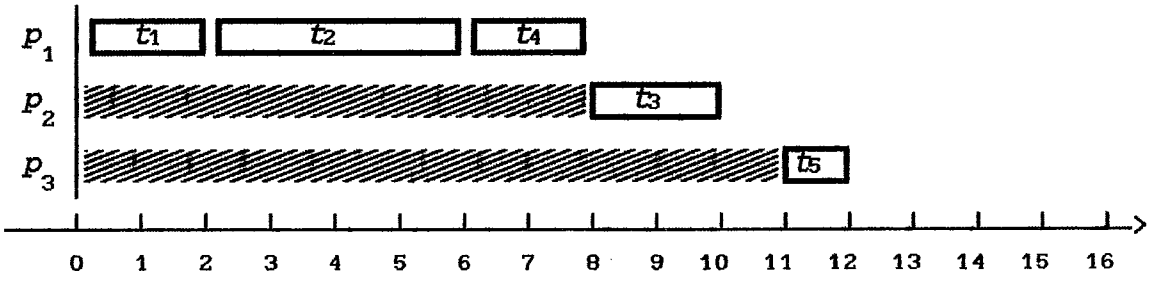


Figura 3.5: Solução Completa de *ESCALONAMENTO-POR-TERMINO*

procedimento *ESCALONAMENTO-POR-TERMINO*;

INSTANTE ⇐ 0;

repita

seja T^* ={(t_i,p_q) tq. $t_i \in N_T$ madura em *INSTANTE*, $p_q \in N_P$, e
[∀ $t_j \in N_T$ madura em *INSTANTE* e ∀ $p_r \in N_P$,
[($FIM_{iq} \leq FIM_{ir}$) ∨ (($FIM_{iq} = FIM_{ir}$) ∧ ($INI_{iq} \leq INI_{ir}$))};

enqto $T^* \neq \emptyset$ faça

escolhe (t_i,p_q) arbitrariamente ou por critério auxiliar;

se $FIM_{iq} \leq$ momento em que a próxima tarefa termina então

escalone t_i a p_q no instante INI_{iq};

INSTANTE ⇐ instante em que a próxima tarefa termina;

ate que *INSTANTE* = ∞;

fim

O cálculo para a escolha do melhor par de tarefa e processador

em cada passo custa $O(nk)$, e é efetuado no máximo $2k$ vezes, já que, em cada passo, o algoritmo ao menos escalona uma tarefa ou termina uma tarefa (adiando a decisão do escalonamento para o próximo passo).

Portanto, a complexidade de *ESCALONAMENTO-POR-TERMINO* é $O(nk^2)$, possivelmente multiplicado pela complexidade de um critério adicional de diferenciação das tarefas.

3.3.3. Critérios de Prioridade Auxiliares

As heurísticas de prioridade descritas a seguir servem como um método suplementar de diferenciação das tarefas.

Todas as heurísticas são baseadas nos tempos de execução das tarefas. Mas, como assumimos processadores heterogêneos, os valores exatos não podem ser determinados antes da alocação; assim, tomamos os tempos médios de execução das tarefas $\bar{X}$.

Note que o emprego de critérios auxiliares deverá ser mais importante para *ESCALONAMENTO-SIMPLES*, que a cada alocação considera todas as tarefas maduras, e menos importante para o *ESCALONAMENTO-POR-TERMINO*, já que, para a ocorrência de conflitos, as tarefas maduras devem coincidir nos instantes mínimos em que são capazes de iniciar e terminar.

4.3.3.1. Maior Tempo de Execução

Trata-se de simples análise local dos nós, com a noção de que o privilégio das tarefas mais pesadas computacionalmente na ordem das alocações contribui para o balanceamento de carga.

Considerando a força-tarefa da figura 3.2, t_2, t_4 e t_5 são, de acordo com esta heurística, as tarefas mais prioritárias, seguidas de t_3

e, então, de t_1.

3.3.3.2. Maior Caminho de Saída

O *caminho de saída* de uma tarefa $t_i \in N_T$ é definido como o caminho de maior comprimento dentre os caminhos em G_T que partem de t_i e alcançam um nó folha em G_T. O *caminho crítico* de G_T é o maior caminho de saída de suas raízes. Em modelos de processadores idênticos, que não consideram atrasos de comunicação, o comprimento de um caminho em G_T é dado pela soma dos tempos de execução das tarefas pertencentes ao caminho. Ao introduzir processadores heterogêneos e atrasos de comunicação no modelo, o comprimento de um caminho deixa de ser estático e pode mudar de acordo com a alocação. Neste trabalho, ignoramos os atrasos de comunicação ao calcular o comprimento de um caminho, definindo-o como a soma dos tempos médios $\bar{X}$ de execução das tarefas que compõe o caminho.

O caminho crítico é de grande interesse, pois contém a porção do programa que representa a maior oportunidade para melhora, já que as tarefas no caminho crítico determinam o menor tempo necessário para a execução da força-tarefa. Além disso, as tarefas no caminho crítico devem ser executadas em seqüência. De fato, muitos dos esquemas clássicos de alocação são baseados em heurísticas de caminho crítico.

Em síntese, a idéia é que as tarefas que têm grandes porções da computação dependentes de si devem ter prioridade na escolha dos processadores e também na ordem de execução. Para tanto, podemos identificar o comprimento de caminho de saída de cada tarefa, e usá-lo como sua prioridade.

O procedimento *PRIORIDADE-CSA* mostra o cômputo dos caminhos de saída das tarefas. Para cada tarefa t_i, cujo caminho de saída ainda não é conhecido, mas os caminhos de saída de seus filhos já foram calculados, tome o maior caminho de saída dos filhos e some ao peso computacional médio $\bar{x}_i$ de t_i.

procedimento *PRIORIDADE-CSA*

para $\forall\ t_i \in G_T$, t_i é folha em G_T faça

$CSA_i \Leftarrow \bar{x}_i$;

para $\forall\ t_i \in G_T$, $CSA_i = ?$ tq. $\forall\ t_j$, $e_{ij} \in E_T$ e $CSA_j \neq ?$ faça

$$CSA_i \Leftarrow \bar{x}_i + \max_{e_{ij} \in E_T} CSA_j;$$

fim

A complexidade deste algoritmo é $O(k^2)$, pois o maior caminho de saída dos filhos de t_i é obtido em até k-1 iterações. Isto é efetuado para cada tarefa t_i que não é folha em G_T, portanto, da ordem de k vezes.

Na figura 3.2, o caminho de saída de t_5 é 4, de t_4 é 4 + 4 = 8, de t_3 é 3 + 4 = 7, de t_2 é 4 + 8 = 12 e de t_1 é 2 + 12 = 14. Sendo t_1 a única raiz, o caminho crítico de G_T é 14. Portanto, de acordo com este critério, as tarefas são inseridas em uma lista decrescente de prioridades na ordem t_1, t_2, t_4, t_3 e t_5.

3.3.3.3. Maior Caminho de Saída Ponderado

O problema principal com *PRIORIDADE-CSA* é que apenas o maior subgrafo dependente de cada tarefa é considerado.

Então, Shirazi e Wang sugeriram uma heurística segundo a qual a prioridade de uma tarefa é função não apenas do comprimento de seu caminho de saída, mas também do número e peso de seus filhos e descendentes [SHI90].

O procedimento *PRIORIDADE-CSP* calcula o caminho de saída ponderado de cada tarefa. Para cada tarefa t_i, cujo caminho de saída ponderado ainda não é conhecido, mas os caminhos de saída de seus filhos já foram calculados, tome a soma dos caminhos de saída ponderados de todos os filhos de t_i, CSV_i, e o maior caminho ponderado de seus filhos, CSU_i. O caminho ponderado de t_i é dado por:

$$\bar{x}_i + CSU_i + CSV_i / CSU_i$$

procedimento *PRIORIDADE-CSP*;

para $\forall\ t_i \in G_T$, t_i é folha em G_T faça

$CSP_i \Leftarrow \bar{x}_i$;

para $\forall\ t_i \in G_T$, $CSP_i = ?$ tq. $\forall\ t_j$, $e_{ij} \in E_T$ e $CSP_j \neq ?$ faça

$$CSU_i \Leftarrow \max_{e_{ij} \in E_T} CSP_j;$$

$$CSV_i \Leftarrow \sum_{e_{ij} \in E_T} CSP_j;$$

$CSP_i \Leftarrow \bar{x}_i + CSU_i + CSV_i / CSU_i$;

fim

A complexidade do algoritmo é $O(k^2)$, pois os cálculos de CSU_i e de CSV_i podem exigir, cada um, até k-1 iterações. Esses cálculos são efetuados para cada tarefa t_i que não é folha em G_T, portanto da ordem de k vezes.

Por exemplo, na figura 3.2, o caminho de saída ponderado de t_5 é 4, de t_4 é 4 + 4 + 4/4 = 9, de t_3 é 3 + 4 + 4/4 = 8, de t_2 é 4 + 9 + (9 + 8)/9 = 21.5 ≅ 21 e de t_1 é 2 + 21 + 21/21 = 24. A lista decrescente de prioridades é, como no critério anterior, t_1, t_2, t_4, t_3 e t_5.

CAPÍTULO 4

ALOCAÇÃO PARA MINIMIZAR O SOMATÓRIO DE EXECUÇÃO, COMUNICAÇÃO INTER-PROCESSADOR E INTERFERÊNCIA

A alocação de tarefas de modo a minimizar os custos totais de execução e comunicação tem sido freqüentemente associado a modelos e algoritmos de fluxo de rede [STO77, STO78, WU80, LO83, LO88].

Lo [LO83, LO88] desenvolveu uma heurística que combina invocação sucessiva de algoritmos de Fluxo Máximo/ Corte Mínimo com um algoritmo do tipo guloso para obter mapeamentos subótimos de tarefas a processadores.

Além disso, estendeu o modelo e os algoritmos para incluir na função objetivo interferência entre tarefas co-residentes, com o propósito de estimular o balanceamento de carga.

4.1. Conceitos Básicos

Apresentamos alguns conceitos [SZW86] úteis para a compreensão do modelo e algoritmos descritos adiante.

Uma *rede* é um dígrafo $R\ (N_R, E_R)$, tal que, a cada aresta $e \in E_R$ está associado um número inteiro não negativo $c(e)$, denominado capacidade da aresta e. Adicionalmente, R possui dois nós s e t, $s, t \in N_R$, com as seguintes propriedades: s alcança todos os nós $u \in N_R$, $u \neq s, t$, e t é alcançado por todos os nós $u \in N_R$, $u \neq s, t$. Diz-se que s e t são nós diferenciados, e os demais nós

$u \neq s, t, u \in N_R$, são chamados nós ordinários. Os nós s e t também são conhecidos por, respectivamente, *fonte* e *sumidouro*.

Um *corte* em R, denotado por $(S,\bar{S})$, é uma coleção de arestas que induz uma partição de R em dois subconjuntos disjuntos de nós, S e $\bar{S}$, tal que $s \in S$ e $t \in \bar{S}$. Portanto, $(S,\bar{S})$ consiste de todas as arestas que ou vão de um nó em S para um nó em $\bar{S}$ ou de um nó em $\bar{S}$ para um nó em S. A *capacidade do corte* $(S,\bar{S})$, $c(S,\bar{S})$, é definida como a soma das capacidades das arestas em $(S,\bar{S})$ que vão de nós em S para nós em $\bar{S}$.

Uma *função de fluxo* f em R é a atribuição de um inteiro não negativo a cada aresta da rede, de modo que o valor atribuído a qualquer aresta não exceda sua capacidade, isto é, $f(e) \leq c(e), \forall\ e \in E_R$, e tal que, para qualquer nó ordinário $u \neq s, t, u \in N_R$, a soma dos valores de fluxo das arestas entrando em u seja igual à soma dos valores de fluxo das arestas partindo de u. O *fluxo total* $\mathcal{F}$ associado com a função de fluxo f é o total dos valores de fluxo das arestas que saem da *fonte* menos a soma dos valores de fluxo das arestas que entram na *fonte*.

Pelo conhecido teorema do Fluxo Máximo/ Corte Mínimo [FOR62], o valor máximo do fluxo total $\mathcal{F}$ sobre todas as funções de fluxo f em uma rede R é igual à capacidade do corte mínimo de R. Ou seja:

$$\max_{\forall\ f \text{ em } R} \{\mathcal{F}(f)\} = \min_{\forall\ (S,\bar{S}) \text{ em } R} \{c(S,\bar{S})\}$$

As definições e os resultados acima podem ser estendidos para redes não direcionadas e para $n>2$ nós diferenciados.

Em redes não direcionadas, cada aresta não direcionada pode ser vista como duas arestas direcionadas com sentidos opostos, cada qual com capacidade idêntica à da aresta original (não direcionada). A capacidade

de um corte passa a ser simplesmente a soma das capacidades de todas as arestas (não direcionadas) pertencentes ao corte.

Para n>2 nós diferenciados, procedemos da seguinte forma: Um *corte n-way* é definido como um conjunto de arestas que induz uma partição de R em n subconjuntos distintos de nós, e tal que nenhum subconjunto próprio do corte n-way é também um corte n-way. Assim, o corte n-way é composto por todas as arestas cujas extremidades não pertencem ao mesmo subconjunto S_i, $i = 1,...,n$. A *capacidade* ou *custo de um corte n-way* é a soma das capacidades de suas arestas.

4.1.1. Aplicação

Stone utiliza um modelo de fluxo de rede e um algoritmo de Fluxo Máximo/ Corte Mínimo para obter uma atribuição ótima em tempo polinomial em sistemas de n=2 processadores [STO77].

Constrói-se o modelo de rede não direcionada descrito na seção anterior. Os n=2 processadores $\{p_1,p_2\}$ são os nós diferenciados (fonte e sumidouro), e as $k \geq 1$ tarefas $\{t_1,...,t_k\}$ são nós ordinários. Há uma aresta e_{ij} entre cada par de tarefas t_i e t_j que se comunicam, de capacidade igual ao custo de comunicação c_{ij}. Duas arestas, e'_{i1} e e'_{i2}, partem de cada tarefa t_i para os processadores p_1 e p_2, respectivamente. A aresta e'_{i1}, da tarefa t_i para o processador p_1, tem capacidade igual a x_{i2}, custo de execução de t_i em p_2. Inversamente, a capacidade da aresta e'_{i2}, de t_i para p_2, é o custo de executar t_i no processador p_1, x_{i1}.

Com a remoção das arestas de um corte, cada tarefa da rede fica conectada a um e apenas um processador; e, portanto, um corte

corresponde a uma atribuição de tarefas a processadores. A capacidade do corte, dada pela soma das capacidades de suas arestas, equivale ao somatório dos custos de execução e comunicação incorridas pela atribuição correspondente.

Na figura (4.1) vemos um tal corte, de acordo com o qual as tarefas t_1, t_2 e t_3 são alocadas a p_1, e t_4 e t_5, a p_2.

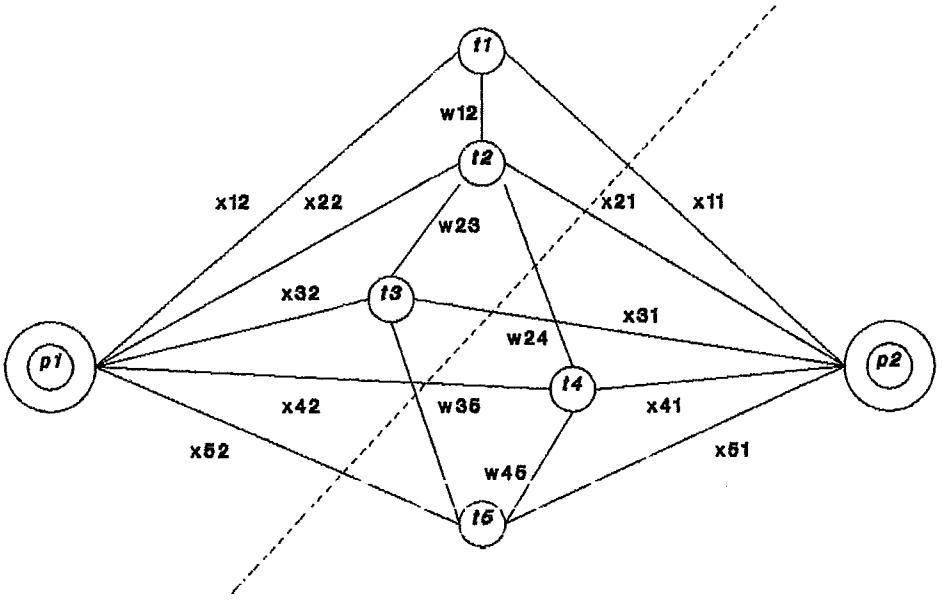


Figura 4.1: Corte 2-way

Um corte 2-way de peso mínimo de uma rede R, cujas arestas têm capacidades não negativas, pode ser obtido em tempo polinomial utilizando algoritmos de Fluxo Máximo/ Corte Mínimo. Vários algoritmos com esta finalidade foram propostos [FOR62, DIN70, EDM72, KAR74, MAL78, SLE81,GOL88], de complexidade tão baixa quanto $O(|N_R|^3)$ [KAR74, MAL78], ou $O(|N_R||E_R|\log(|N_R|^2/|E_R|))$ [GOL88].

Generalizando a idéia para $n \geq 2$ processadores, temos uma rede na qual os n processadores $\{p_1,...,p_n\}$ são nós diferenciados e as k

tarefas $\{t_1,...,t_k\}$ são nós ordinários. Há uma aresta e_{ij} entre cada par de nós t_i e t_j, de capacidade w_{ij}. De cada tarefa t_i, partem, também, n arestas e'_{iq}, para cada um dos n processadores p_q, de capacidade w'_{iq}.

Um corte n-way em tal rede particiona seus nós em n subconjuntos distintos com exatamente um nó processador em cada conjunto e, define uma atribuição de tarefas a processadores. É conveniente escolher W e W' tais que a capacidade do corte seja igual ao custo da alocação correspondente.

Com essa finalidade, faz-se:

$$w_{ij} = c_{ij}$$

$$w'_{iq} = \frac{1}{n-1} \sum_{r \neq q} x_{ir} - \frac{n-2}{n-1} x_{iq}$$

Stone mostra que, com essa escolha de W e W', o custo do corte é igual ao custo da alocação correspondente.

Assim, o problema de se encontrar uma alocação ótima se torna equivalente ao de se obter um corte n-way de peso mínimo. De fato, note que, se a tarefa t_i é atribuída ao processador p_q, então t_i é cortada de todos os processadores exceto p_q, incorrendo em um custo de:

$$\sum_{r \neq q} w'_{ir} = \sum_{r \neq q} \left(\frac{1}{n-1} \sum_{s \neq q} x_{is} - \frac{n-2}{n-1} x_{ir} \right) = x_{iq}$$

As arestas entre pares de tarefas t_i e t_j alocadas a processadores diferentes, de peso $w_{ij}=c_{ij}$, também serão incluídas no corte n-way.

Uma dificuldade deste esquema é que, pela escolha de W', podem ocorrer arestas de capacidade negativa na rede R. Isto é facilmente contornado através do seguinte procedimento. Para cada $t_i \in N_T$, considere as n arestas e'_{ir}. Seja e'_{iq} a aresta de menor capacidade w'_{iq}. Se $w'_{iq} < 0$,

então acresça a capacidade w'_{ir} de cada uma das n arestas e'_{ir} da quantidade $w'_{iq} + 1$, e é evidente que t_i não mais terá arestas de capacidade negativa. Além disso, observe que, como qualquer corte n-way em R isola t_i de n-1 processadores, exatamente n-1 arestas e'_{ir}, de capacidade $w'_{ir} + w'_{iq} + 1$, pertencerão ao corte, o qual terá, portanto, um peso adicional de $(n-1)(w'_{iq} + 1)$. Sendo este termo constante e independente das alocações, segue-se que um corte mínimo na rede modificada da forma descrita acima equivale a um corte mínimo na rede original.

Ao introduzir interferência no modelo de Stone, o que é proposto por Lo [LO83, LO88], o modelo de rede sofre alterações quanto aos cálculos de W e W', a fim de manter a correspondência entre a capacidade de um corte e o custo da alocação correspondente. Devemos então fazer:

$$w_{ij} = c_{ij} - i_{ij}$$

$$w'_{iq} = \frac{1}{n-1} \sum_{r \neq q} x_{ir} - \frac{n-2}{n-1} x_{iq} + \frac{1}{2(n-1)} \sum_{1 \leq j \leq k} i_{ij}$$

Lo mostra que, com essa escolha de W e W', a capacidade de cada corte é igual ao custo total de execução, comunicação inter-processador e interferência da alocação correspondente.

Neste caso, pela definição de W', é possível que a rede R possua arestas entre tarefas e_{ij} com capacidade $w_{ij} = c_{ij} - i_{ij} < 0$. Aqui, porém, o artifício de somar uma constante ao peso de cada aresta e_{ij} não funciona, pois o número de arestas alteradas (arestas e_{ij} entre tarefas) que pertencem a um corte n-way é variável; conseqüentemente, o peso de um corte na rede modificada não estaria simplesmente acrescido de um valor constante, e um corte mínimo nessa rede não corresponderia a um

corte mínimo na rede original.

4.2. Formulação do Problema

No modelo original de Stone [STO77], o custo associado a uma alocação $\mathcal{S}$ é dado por:

$$\mathcal{C}_2(\mathcal{S}) = \sum_{\mathcal{A}(t_i)=p_q} x_{iq} + \sum_{\substack{\mathcal{A}(t_i)=p_q \\ \mathcal{A}(t_j)\neq p_q}} c_{ij}$$

No modelo proposto por Lo [LO83, LO88], que inclui interferência, o custo associado à alocação passa a ser:

$$\mathcal{C}_3 = \sum_{\mathcal{A}(t_i)=p_q} x_{iq} + \sum_{\substack{\mathcal{A}(t_i)=p_q \\ \mathcal{A}(t_j)\neq p_q}} c_{ij} + \sum_{\substack{\mathcal{A}(t_i)=p_q \\ \mathcal{A}(t_j)=p_q}} i_{ij}$$

A seguir, relacionamos as diferenças quanto as hipóteses formuladas no capítulo anterior. Então, descrevemos formalmente a modelagem da aplicação e do sistema distribuído, e discutimos suas limitações.

4.2.1. Hipóteses

Ao contrário das tarefas relacionadas por precedência do capítulo anterior, pode ocorrer troca de mensagens entre tarefas concorrentes.

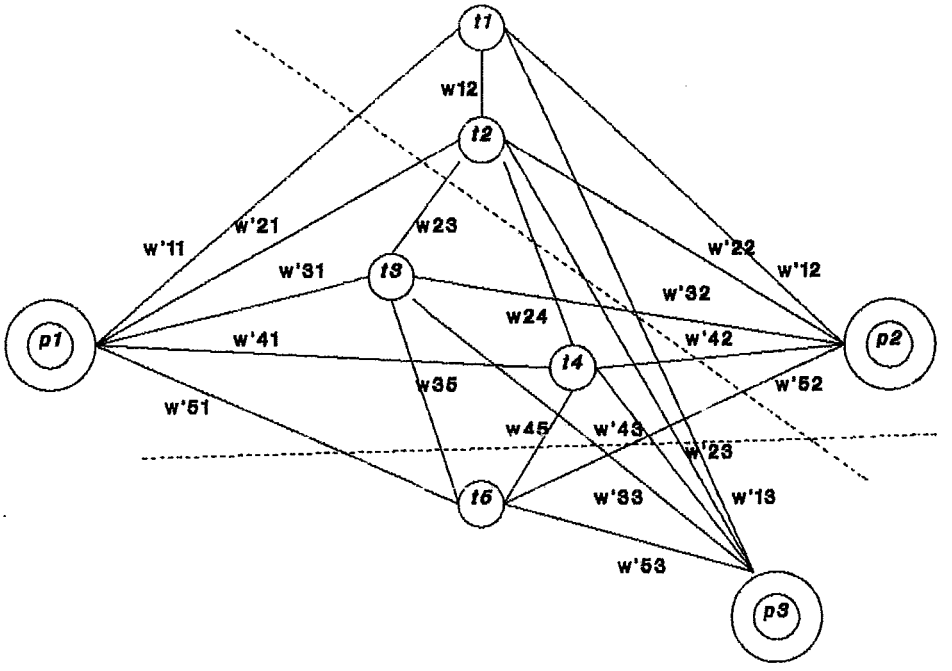
Os processadores são completamente interconectados através de canais de comunicação idênticos, de modo que o tempo de transmissão de mensagens entre um par tarefas independe dos processadores aos quais as tarefas estão alocadas.

4.2.2. Modelagem da Força-Tarefa e do Sistema Distribuído

A computação e a arquitetura são representadas por uma rede *n*-way não direcionada $R\ (N_R, E_R)$, pelo processo descrito em 4.1.1. Temos $N_R = P + T$, onde P são os nós diferenciados e T são os nós ordinários. E_R é composto de arestas e_{ij} de capacidade w_{ij} entre cada par de tarefas t_i e $t_j \in N_T$, e arestas e'_{iq} de capacidade w'_{iq} entre cada processador $p_q \in N_P$ e cada tarefa $t_i \in N_T$. W e W' são calculados com base em X, C e I.

A função C atribui a cada par de nós (t_i, t_j), $e_{ij} \in E_T$, um valor $c_{ij} \geq 0$, $c_{ij} \in \mathbb{N}$, que equivale ao tempo para transmitir toda a comunicação entre t_i e t_j quando t_i e t_j estão alocados a processadores diferentes.

A função I, interferência, atribui a cada par de tarefas (t_i, t_j), t_i, $t_j \in N_T$, um valor $i_{ij} \geq 0$, $i_{ij} \in \mathbb{N}$. A interferência expressa a "incompatibilidade" entre tarefas quando alocadas a um mesmo processador. Conceitualmente, a interferência se refere à contenção de tarefas co-residentes pelos recursos do processador ao qual ambas estão alocadas (trocas de contexto e sincronização para acesso a CPU, memória, dispositivos de E/S) e de tarefas co-residentes comunicantes pela utilização dos mecanismos de comunicação entre processos (sincronização para o uso de buffers e para transmissões de mensagens, etc.). Apesar de ser claro que a interferência está ligada diretamente à quantidade de recursos disponível em cada processador, assumimos, por tratabilidade, que a interferência é independente do processador ao qual as tarefas são alocadas. Isto ocorre, por exemplo, para processadores idênticos ou com muitos recursos disponíveis.



Execução					
X	t_1	t_2	t_3	t_4	t_5
$\overline{p}$	2	4	3	4	4
p_1	2	4	3	2	6
p_2	2	4	2	4	5
p_3	2	4	4	6	1

Comunicação					
C	t_1	t_2	t_3	t_4	t_5
t_1	–	4	0	0	0
t_2	–	–	2	1	0
t_3	–	–	–	0	1
t_4	–	–	–	–	1
t_5	–	–	–	–	–

Interferência					
I	t_1	t_2	t_3	t_4	t_5
t_1	–	2	0	0	0
t_2	–	–	1	0	0
t_3	–	–	–	0	0
t_4	–	–	–	–	0
t_5	–	–	–	–	–

Figura 4.2: Exemplo de $R(N_R, E_R)$

Tal como no capítulo anterior, o tempo de processamento das tarefas nos processadores é dado pela função X, que associa a cada par (t_i,p_q), $t_i \in N_T$ e $p_q \in N_P$, um peso $x_{iq}>0$, $x_{iq} \in \mathbb{N}$.

A figura 4.2 exemplifica a modelagem descrita acima. Temos uma rede 3-way, correspondente à alocação de uma força-tarefa semelhante à da figura 3.2, em um sistema de 3 processadores. Mostramos também um corte na 3-way nessa rede, segundo o qual temos t_3 e t_4 alocadas a p_1, t_1 e t_2 a p_2, e t_5 a p_3.

4.2.3. Restrições

Além das restrições decorrentes do fato da alocação ser feita estaticamente, já vistas no capítulo 3, é importante notar que os algoritmos de cortes e "clustering" requerem que o custo de unir/ separar tarefas seja independente do processador ao qual as tarefas são alocadas.

Logo, os atrasos de comunicação por transmissão das mensagens pelos canais entre os processadores devem ser uniformes. Conseqüentemente, os processadores, devem ser completamente interconectados através de canais idênticos. Analogamente, a interferência entre as tarefas deve ser independente da alocação.

4.2.4. Outras Considerações

Supomos que os dados necessários para a construção da rede $R(N_R,E_R)$ estão disponíveis, expressos em unidades de medida convenientes.

Novamente, esses dados devem ser derivados de uma análise das características da força-tarefa e do sistema distribuído.

Ao contrário do caso de X e C, contudo, não há um procedimento definido para identificar e estimar adequadamente a interferência entre tarefas, a qual envolve tantos elementos distintos, como contenção por CPU, memória, buffers, dispositivos de E/S, etc., e as conseqüentes trocas de contexto. Assim, o uso de interferência implica em grande inconveniência, seja pela maior dificuldade em estimar valores razoáveis, ou pela maior imprecisão introduzida na representação da computação.

4.3. Algoritmos de Cortes Sucessivos e "Clustering"

Inicialmente são alocadas as tarefas que têm forte preferência por processadores específicos, através da invocação repetida do algoritmo de Fluxo Máximo/ Corte Mínimo em R. A alocação resultante é ótima, mas é, no caso geral, apenas parcial.

Completa-se a alocação anterior com apenas um processador caso sua otimalidade seja mantida, o que é verificado através da comparação com um limite inferior do custo de separar as tarefas.

Caso contrário, segue um algoritmo guloso que identifica e agrupa pares de tarefas que se "atraem" mais do que a média; após definidos os grupos, cada um deles é alocado ao seu melhor processador. A alocação resultante pode ser sub-ótima.

No restante deste capítulo, descrevemos detalhadamente o algoritmo para o modelo original, sem interferência, e, então discutimos a consideração da interferência nas alocações, e também a resolução do problema para o modelo que inclui este novo fator.

4.3.1. Modelo sem Interferência

A idéia é reduzir a rede sucessivamente para n=2 nós diferenciados, $p_q \in N_p$, e $\bar{p}_q$, que representa os outros n-1 processadores, e obter o corte mínimo da rede reduzida. As tarefas ainda alcançadas por p_q após o corte são alocadas a ele. Essas alocações fazem parte de qualquer solução ótima. A repetição desse procedimento para cada $p_q \in N_p$ constitui um passo de *GRAB*.

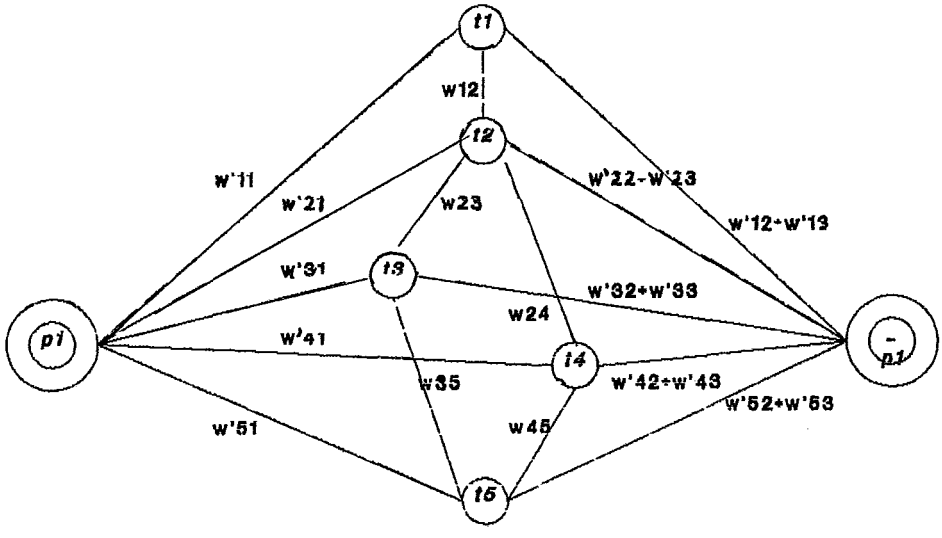


Figura 4.3: Rede de 3 Processadores Reduzida

Ao fim de cada passo, a rede R é reconfigurada, eliminando-se as tarefas alocadas no passo anterior, e recalculando-se as capacidades w' das arestas para refletir a alocação parcial. *GRAB* termina quando em algum passo não ocorrem novas alocações de tarefas a processadores.

Lo mostra que as alocações feitas por *GRAB* fazem parte de qualquer solução ótima. Em particular, se *GRAB* produz uma alocação completa, essa alocação é ótima.

Sejam:

T^m= conjunto de tarefas não alocadas até m^o passo;

S_q^m= conjunto de tarefas alocadas a p_q pelo m^o passo;

X^m= pesos de execução recalculados para refletir alocações até m^o passo:

$$x_{iq}^{m} \Leftarrow x_{iq}^{m-1} + \sum_{r \neq q} \sum_{t_j \in S_r^m} c_{ij}, \forall (t_i, p_q) \in E_R;$$

R^m= rede R reconfigurada a cada passo m, como a seguir:

$$N_R \Leftarrow \{p_q \text{ tq. } \forall\ p_q \in N_P\} \cup \{t_i \text{ tq. } \forall\ t_i \in T^m\};$$

$$E_R \Leftarrow \{e'_{iq} \text{ tq. } \forall\ t_i, p_q \in N_R\} \cup \{e_{ij} \text{ tq. } \forall\ t_i, t_j \in N_R\};$$

$$w_{ij}^{m} \Leftarrow c_{ij}, \forall\ e_{ij} \in E_T;$$

$$w'^{m}_{iq} \Leftarrow \frac{1}{n-1} \sum_{r \neq q} x_{ir}^{m} - \frac{n-2}{n-1} x_{iq}^{m}, \forall\ e'_{iq} \in E_R;$$

procedimento *GRAB*

$m \Leftarrow 0$; $T^0 \Leftarrow T$; $S^o \Leftarrow \emptyset$; $X^0 \Leftarrow X$;

repita

construa R^m a partir de T^m, S^m e X^m;

para $\forall\ p_q,\ p_q \in N_P$ faça

reduza R^m para n=2, com p_q e $\bar{p}_q = N_P - \{p_q\}$;

ache o corte mínimo de R^m, obtendo T^{m+1} e S_q^{m+1};

$m \Leftarrow m + 1$;

até que T^m=0 ou $T^m = T^{m-1}$;

fim

Em *GRAB* há no máximo k iterações, com n cortes cada. Como existem algoritmos de Fluxo Máximo/ Corte Mínimo de complexidade

$O(|N_R||E_R|\log(|N_R|^2/|E_R|))$, e sabemos que $|N_R| = n+k$ e que, em geral, $k > n$, podemos dizer que a complexidade resultante *GRAB* é proporcional a $k\,|E_R|\,\log\,(k^2/|E_R|)$.

Assinalamos que o algoritmo de Fluxo Máximo/ Corte Mínimo utilizado em nossa implementação, no entanto, foi aquele proposto por Edmonds e Karp [EDM72], de complexidade $O(k^2|E_R|)$.

GRAB produz uma atribuição parcial ótima, possivelmente completa, de tarefas a processadores. Se a alocação não estiver completa, a próxima etapa, *LUMP*, compara um limite inferior do custo de distribuir os módulos restantes entre os processadores com o menor custo de atribuí-los todos a um único processador.

O limite inferior da solução ótima pode ser definido como:

$$\mathcal{L} = \min_{\substack{t_i \neq t_{arb} \\ t_i \in T^m}} \{\min_{R^{*i}} (S,\bar{S})\} + \sum_{T_i \in T^m} \min_{p_q \in N_P} \{x_{iq}\}$$

O primeiro termo consiste em um limite inferior dos custos de comunicação, quando mais de um processador é utilizado na alocação. Para calculá-lo, fixamos uma tarefa $t_{arb} \in T^m$ arbitrária. Então, para cada $t_i \neq t_{arb}$, $t_i \in T^m$, construímos uma rede R^{*i}, de fonte t_{arb} e sumidouro t_i. O menor corte mínimo, dentre todas as redes R^{*i}, é um limite inferior dos custos de comunicação inter-processador, pois em uma alocação que emprega mais de um processador, a tarefa t_{arb} terá que ser separada de ao menos uma outra tarefa $t_i \in T^m$.

O segundo termo representa um limite inferior dos custos de execução, dado pela soma dos tempos de execução incorridos se cada tarefa for atribuída ao seu melhor processador.

procedimento *LUMP*

se existir $p_q \in N_P$ tq. $\sum_{t_i \in T^m} x_{iq} \leq \tilde{\mathcal{L}}$ então

atribua todas as tarefas de T^m a p_q

fim

O cálculo do limite inferior do custo de um corte n-way quando tarefas são atribuídas a mais de um processador implica em obter k-1 cortes mínimos em uma rede composta unicamente por tarefas (no máximo k). Novamente, a complexidade é proporcional a $k\,|E_R|\,\log\,(k^2/|E_R|)$. .

A etapa final é um algoritmo do tipo guloso, que localiza tarefas entre as quais os custos de comunicação são acima da média $\bar{c}$, e as agrupa em "clusters".

Tarefas em um mesmo "cluster" são alocadas conjuntamente ao processador que minimiza a soma dos tempos de execução de todas as tarefas que contidas no "cluster".

procedimento *GREEDY*

marque todas as arestas e_{ij} tq. $c_{ij} \leq \bar{c}$;

enqto houver uma aresta e_{ij} não marcada faça

marque e_{ij} e todas as arestas entre tarefas de G_i e G_j

$G_i \Leftarrow G_i \cup G_j$; $G_j \Leftarrow \emptyset$;

aloque cada grupo $G_i \neq \emptyset$ ao processador p_q que minimize $\sum_{t_i \in G} x_{iq}$

fim

A média dos custos de comunicação $\bar{c}$ é dada pelo somatório dos pesos de comunicação entre cada par de tarefas de N_T dividido pelo total

de arestas, ou seja:

$$\bar{c} = \sum_{1 \le i \le j \le k} \frac{c_{ij}}{\binom{k}{2}}$$

O procedimento *GREEDY* simplesmente examina cada aresta de comunicação. Logo, sua complexidade é de $O(|E_R|)$.

O algoritmo resultante da combinação descrita de *GRAB*, *LUMP* e *GREEDY* é chamado *A*.

4.3.2. Modelo com Interferência

Uma deficiência do modelo de Stone é que não há esforço direto no sentido de obter concorrência, muito embora algum paralelismo seja introduzido como consequência de se evitar custos totais altos. O resultado são alocações pouco balanceadas.

Isto motivou a extensão do modelo proposta por Lo [LO88] para incluir um fator adicional, a interferência, idealizada para atuar diretamente no incremento do grau de concorrência das alocações.

Vimos que os custos de interferência são incorridos por tarefas co-residentes, contabilizando custos devido a trocas de contexto e sincronizações para acesso a recursos compartilhados (memória, CPU, dispositivos de E/S), além de custos relacionados com a comunicação entre tarefas, devido à contenção para "buffers" e sincronização para transferência de mensagens.

Como no caso da comunicação, a interferência entre cada par de tarefas deve ser independente dos processadores aos quais as tarefas são alocadas. Na massa de entrada simulada para os experimentos, fizemos ainda $i_{ij} < c_{ij}$. Isto ocorre, por exemplo, quando todos os processadores

têm capacidade suficiente para acomodar muitas tarefas, de modo que elas não competem por recursos limitados, e o custo devido a trocas de contexto é irrelevante. A interferência se torna, então, função exclusiva da comunicação, ocorrendo, por conseguinte, apenas entre pares de tarefas comunicantes alocadas a um mesmo processador.

Apenas o modelo de rede é adadptado para tratar interferência, pelas alterações no cálculo de W e W'.

Quando houver em R arestas e_{ij} com capacidade $w_{ij} = c_{ij} - i_{ij} < 0$, a determinação da alocação deve ser efetuada isoladamente pelo procedimento *GREEDY*. Porém, isto não é necessário para a massa de entrada que geramos, já que fizemos $i_{ij} < c_{ij}$.

Lo não alterou *GREEDY* para tratar interferência, e comenta a qualidade insatisfatória das alocações obtidas por *A*. O fato de *GREEDY* ignorar a interferência é particularmente ruim devido ao fato observado por Lo de que seu desempenho isolado é comparável ao do algoritmo completo *A*, o que sugere que *GREEDY* efetua a maior parte das alocações.

Em uma tentativa de melhorar o desempenho de *A*, realizamos algumas modificações simples em *GREEDY*, válidas com $i_{ij} < c_{ij}$, $\forall\ t_i, t_j \in N_T$.

Ao invés de agrupar tarefas entre as quais os custos de comunicação são maiores do que a média de comunicação, agrupa-se tarefas entre as quais a diferença entre os custos de comunicação e interferência é maior do que a média das diferenças sobre todas as arestas, a qual é dada por:

$$\bar{\Delta} = \sum_{1 \le i \le j \le k} \frac{c_{ij} - i_{ij}}{\binom{k}{2}}$$

Tarefas em um mesmo "cluster" são alocadas conjuntamente ao processador que minimiza a soma dos tempos de execução e de interferência.

Ao algoritmo completo (A) estendido demos o nome de A'.

CAPÍTULO 5

EXPERIMENTOS E RESULTADOS

Investigamos a relação entre as funções que utilizamos em nosso trabalho como critérios de desempenho, com a intenção de verificar se a introdução da interferência atua no sentido de estimular o balanceamento de carga e explorar o paralelismo das aplicações. Além disso, revisamos a qualidade dos métodos selecionados na otimização dos seus objetivos, além de avaliar suas respostas quanto ao volume de comunicação inter-processador e à eficiência.

A coleção de grafos e funções que representam as aplicações distribuídas é organizada em categorias com características diversas. Os resultados dos experimentos são analisados conjuntamente e também por categorias.

A seguir, detalhamos a composição da massa de entrada e introduzimos a notação e as medidas utilizadas. Então, discutimos os experimentos conduzidos, apresentando os resultados em forma gráfica, juntamente com suas interpretações.

5.1. Massa de Entrada

Um total de 600 forças-tarefa, compostas por $6 \leq k \leq 36$ tarefas, foram simuladas.

As forças-tarefa são organizadas em três categorias distintas, cada qual compondo 1/3 do total da entrada. As categorias se

caracterizam pela razão de arestas de comunicação não nulas, ou densidade. Temos 200 processos com comunicação não nula para 1/6, 1/3 e 1/2 dos $\binom{k}{2}$ pares possíveis de tarefas, que constituem, respectivamente, as assim designadas categorias I, II e III.

O padrão de inter-referências das tarefas é arbitrário, o que é conseguido da seguinte forma: a matriz triangular de comunicação entre tarefas é representada linearmente, de modo que cada par de tarefas corresponde a um índice. Os pares de tarefas que vão se comunicar são determinados gerando-se números (índices) aleatoriamente no intervalo $\left[1, \binom{k}{2}\right]$.

As forças-tarefa se diferenciam também pela razão entre os pesos de execução e comunicação. Há aplicações para os quais a função X é dominante sobre a função C, e vice-versa. Mais precisamente, geramos para X valores no intervalo [10,30], enquanto que C assume aleatoriamente valores nos intervalos [1,5], [5,15], e [15,30]. A imagem da função interferência I é a mesma de C em todos os casos, mas garantimos sempre que $i_{ij} < c_{ij}$.

A tabela 5.1 sumariza toda a informação sobre a massa de entrada com que trabalhamos. Os parâmetros para a sua geração foram escolhidos arbitrariamente, mas com a intenção de favorecer o paralelismo nas forças-tarefa. Por exemplo, para a Categoria III, cujas aplicações são compostas de tarefas com uma maior quantidade de interação (mais densas), mantivemos a razão entre os valores de X e C mais altos em média do que para as outras categorias. Também limitamos a proporção de arestas não nulas a um máximo de 1/2 do total possível, o que já é bem elevado (para uma força-tarefa com k=6 tarefas, podemos ter até 7

arestas de comunicação não nulas; para k=36, até 630).

Tabela 5.1: Forças-Tarefa Simuladas para os Experimentos			
Razão C não Nulo	Nº Simulações	Intervalo X	Intervalo $C \equiv I$
1/6	100	[10,30]	[5,15]
(Categoria I)	100	[10,30]	[15,30]
1/3	100	[10,30]	[5,15]
(Categoria II)	100	[10,30]	[15,30]
1/2	100	[10,30]	[1,5]
(Categoria III)	100	[10,30]	[5,15]

Mapeamos cada força-tarefa em um sistema distribuído aleatório de $3 \leq n \leq 6$ processadores heterogêneos. A menos de quando se indica o contrário, os processadores são completamente interconectados através de canais idênticos. Com isto, os atrasos de comunicação inter-processador independem dos processadores aos quais as tarefas estão alocadas. Atrasos de comunicação uniformes são importantes para o método de cortes e "clustering". Para os algoritmos de listas de prioridade, experimentamos ainda com a função largura de banda L variando aleatoriamente no intervalo [1/3,1] (na prática, isto significa que a comunicação entre um par de tarefas pode tomar até três vezes mais tempo, dependendo de onde as tarefas estão alocadas).

5.2. Notação e Métricas

Abaixo apresentamos um resumo da notação utilizada nesta seção e

no restante deste capítulo.

$\mathcal{C}_1$, $\mathcal{C}_2$ e $\mathcal{C}_3$ são as funções objetivo que consideramos, definidas nos capítulos 4 e 5.

$\mathcal{O}_1$, $\mathcal{O}_2$ e $\mathcal{O}_3$ são soluções ótimas em relação aos objetivos $\mathcal{C}_1$, $\mathcal{C}_2$ e $\mathcal{C}_3$, respectivamente.

As medidas utilizadas pelos experimentos são associadas com $\mathcal{O}_1$, $\mathcal{O}_2$ e $\mathcal{O}_3$.

Determinamos uma alocação $\mathcal{O}_2$, tal que o somatório de seus custos de execução e comunicação inter-processador, que denotamos por $\mathcal{C}_2(\mathcal{O}_2)$, é ótimo. Computamos um escalonamento consistente com a alocação $\mathcal{O}_2$, e medimos o tempo total de execução $\mathcal{C}_1(\mathcal{O}_2)$ desse escalonamento. Procedemos de maneira similar para a alocação $\mathcal{O}_3$, ótima quanto à soma dos custos de execução, comunicação inter-processador e interferência, obtendo $\mathcal{C}_3(\mathcal{O}_3)$ e $\mathcal{C}_1(\mathcal{O}_3)$. Além disso, determinamos uma alocação $\mathcal{O}_1$ com tempo total de execução ótimo, $\mathcal{C}_1(\mathcal{O}_1)$.

As alocações ótimas foram obtidas através de algoritmos "Branch-and-Bound" com heurísticas para guiar e abreviar a expansão do espaço de soluções. Porém, a obtenção de $\mathcal{O}_1$, $\mathcal{O}_2$ e $\mathcal{O}_3$ requer tempos computacionais longos mesmo para n e k pequenos. Então, determinamos soluções ótimas para $k \leq 15$. Para $k > 15$, introduzimos nos algoritmos "Branch-and-Bound" técnicas de poda (expandimos apenas aqueles caminhos para os quais a diferença entre a subestimativa e o valor ótimo encontrado até o momento corrente é superior a 10%) e limitamos a lista de abertos, de modo que passamos a obter soluções subótimas. De qualquer modo, tratamos $\mathcal{O}_1$, $\mathcal{O}_2$ e $\mathcal{O}_3$ como soluções ótimas em relação a seus respectivos critérios.

Para computar escalonamentos consistentes com $\mathcal{O}_2$ e $\mathcal{O}_3$, adotamos como modelo o grafo de precedências, e atribuímos uma ordem de prioridade às tarefas, de acordo com o clássico critério do maior caminho de saída, descrito no capítulo 4.

Utilizamos a força-tarefa e o sistema distribuído da figura 5.1 para exemplificar este processo. Mostramos na figura 5.1 uma solução $\mathcal{O}_3$, ótima com relação a $\mathcal{C}_3$, e um escalonamento associado (que, neste caso, é o único consistente com $\mathcal{O}_3$, e, coincidentemente, é ótimo também quanto a $\mathcal{C}_1$).

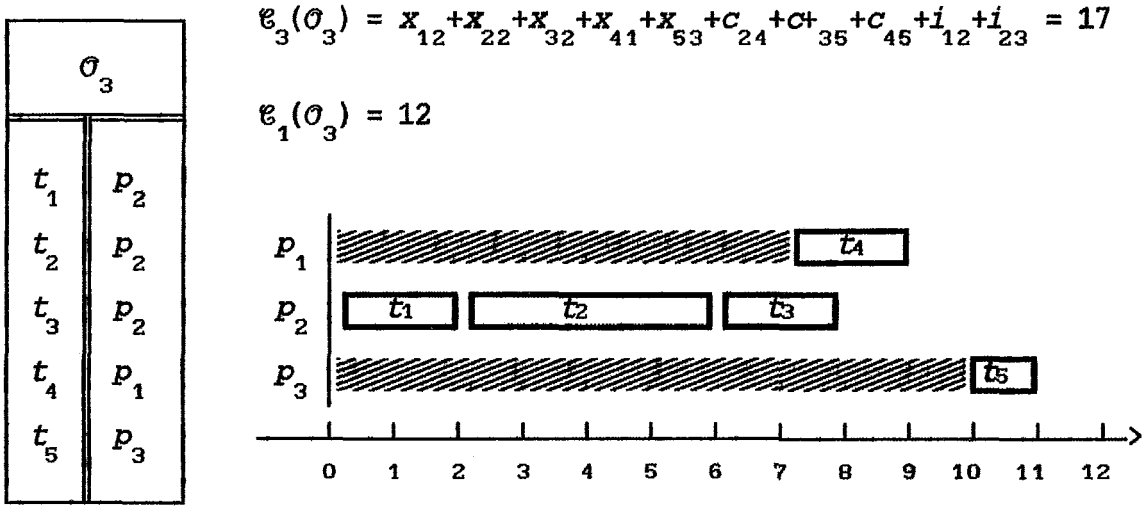


Figura 5.1: Alocação $\mathcal{O}_3$ e um Escalonamento Associado

5.3. Avaliação dos Objetivos de Desempenho

Pretendemos verificar se a introdução da interferência atende ao propósito de estimular, em relação ao modelo sem interferência, o paralelismo e o balanceamento da carga computacional entre os

processadores.

Para isso, medimos os valores do tempo de execução das soluções ótimas segundo os critérios com e sem interferência, $\mathcal{C}_1(\sigma_3)$ e $\mathcal{C}_1(\sigma_2)$, e comparamos com o tempo ótimo, $\mathcal{C}_1(\sigma_1)$.

Lo realizou um experimento com o mesmo intuito [L088] quando da apresentação da interferência [L083, L088]. Entretanto, em seu trabalho, o tempo total de execução era definido, para o modelo sem interferência, como o valor máximo, por processador, do somatório dos custos de execução e comunicação inter-processador e, para o modelo com interferência, como o valor máximo, por processador, do somatório dos custos de execução e comunicação inter-processador e interferência.

Comparando as figuras 5.1 e 5.2, vemos que a adoção do modelo com interferência leva a uma melhora expressiva nos tempos de execução $\mathcal{C}_1$.

Por exemplo, para 53.8% das soluções ótimas segundo o modelo com interferência (σ_3), e apenas 10% das soluções ótimas no modelo sem interferência (σ_2), computamos escalonamentos, fazendo uso do critério de prioridade baseado nos caminhos de saída, com tempos de execução que não superam o valor ótimo por mais 30%. A tabela 5.2 sintetiza este tipo de informação.

Apesar de se poder perceber que, potencialmente, a introdução da interferência tem como conseqüência uma redução do tempo total de execução, as alocações ótimas para o modelo com interferência não são suficientemente boas quanto ao tempo de execução $\mathcal{C}_1$, pois, para aproximadamente metade das soluções σ_3 obtidas, tem-se $\mathcal{C}_1(\sigma_3) > 1.3\ \mathcal{C}_1(\sigma_1)$.

Tabela 5.2: Distribuições $\mathcal{C}_1(\sigma_2)/\mathcal{C}_1(\sigma_1)$ e $\mathcal{C}_1(\sigma_3)/\mathcal{C}_1(\sigma_1)$						
Modelo	Ótimo	≤ 1.1	≤ 1.2	≤ 1.3	≤ 1.5	≤ 2.0
sem *I*	3.5%	5.7%	7.5%	10.0%	15.5%	51.1%
com *I*	9.8%	28.0%	42.8%	53.8%	67.8%	92.1%

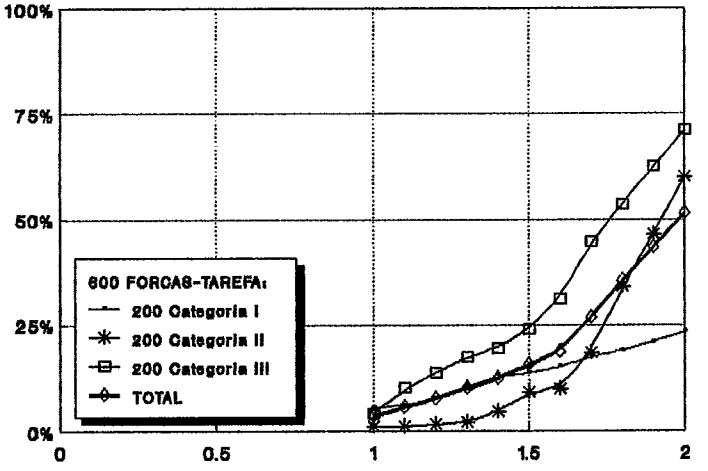


Gráfico 5.1: Distribuição da Razão $\mathcal{C}_1(\sigma_2)$ / $\mathcal{C}_1(\sigma_1)$

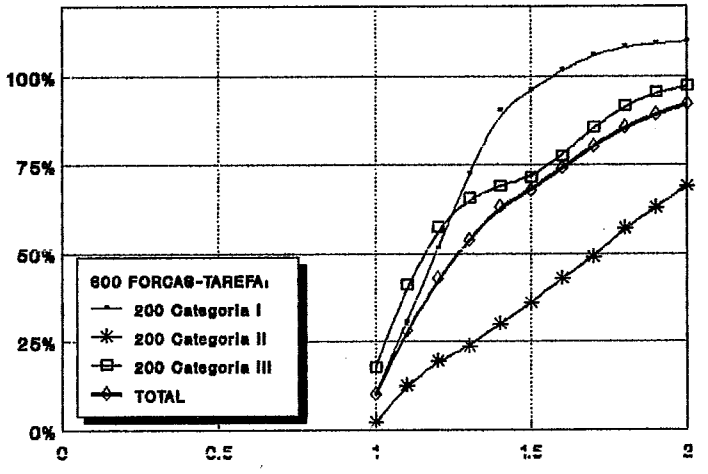


Gráfico 5.2: Distribuição da Razão $\mathcal{C}_1(\sigma_3)$ / $\mathcal{C}_1(\sigma_1)$

5.4. Avaliação dos Algoritmos

Ao contrário do procedimento para investigar os critérios de desempenho, que utiliza exclusivamente soluções ótimas, os experimentos a seguir invocam as heurísticas descritas nos capítulos 4 e 5 para efetuar alocações, as quais são avaliadas sob diferentes aspectos.

5.4.1. Os Algoritmos Quanto aos seus Objetivos

Nesta seção verificamos a qualidade dos algoritmos em gerar soluções subótimas com relação aos seus próprios objetivos.

5.4.1.1. Listas de Prioridade e o Tempo Total de Execução

Na literatura há uma série de trabalhos contendo revisões e comparações de algoritmos pertencentes a classe de algoritmos de escalonamento por listas de prioridade, onde, em geral, se assume atrasos de comunicação irrelevantes e processadores idênticos [ADA74, SHI90].

Aqui avaliamos diferentes combinações de heurísticas baseadas em listas de prioridade, incluindo extensões de alguns algoritmos já propostos [HWA89, SHI90], as quais tratam tarefas relacionadas por precedência, atrasos de comunicação relevantes, e processadores heterogêneos, interconectados arbitrariamente por canais de capacidades quaisquer.

Primeiramente, averiguamos a relevância da utilização, pelos algoritmos *ESCALONAMENTO-SIMPLES* e *ESCALONAMENTO-POR-TERMINO*, de critérios de prioridades auxiliares.

Confirmamos, para a massa de entrada simulada, o fato antecipado

Tabela 5.3: Distribuição $\mathcal{C}_1(\mathcal{S}_1)/\mathcal{C}_1(\mathcal{O}_1)$						
Algoritmo	Ótimo	≤ 1.1	≤ 1.2	≤ 1.3	≤ 1.5	≤ 2.0
SIMPLES	13.0%	50.5%	75.5%	90.0%	97.5%	100.0%
TERMINO	10.0%	50.5%	76.7%	91.5%	98.4%	100.0%

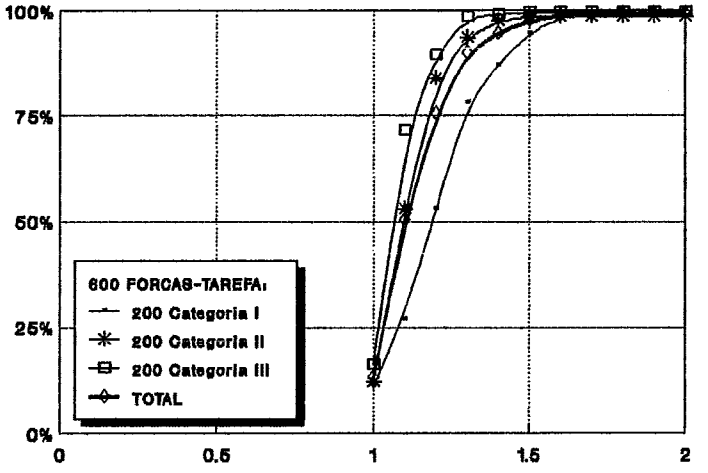


Gráfico 5.3: O Escalonamento Simples e a função $\mathcal{C}_1$

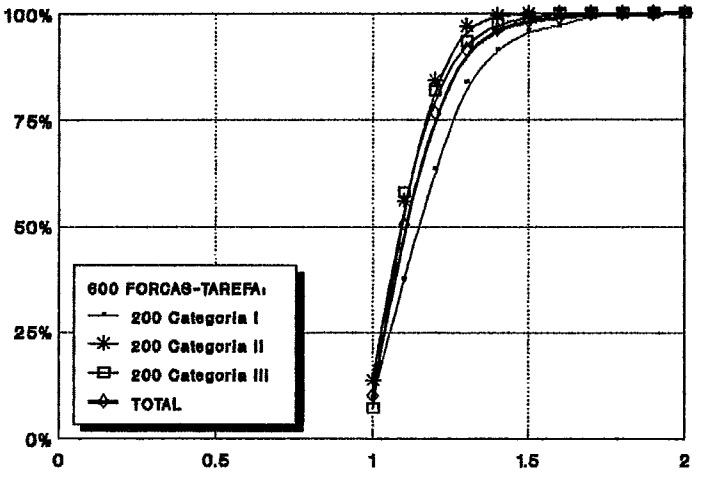


Gráfico 5.4: O Escalonamento por Término e a função $\mathcal{C}_1$

no capítulo 4, de que os critérios auxiliares de prioridade não exercem grande influência sobre as alocações efetuadas por *ESCALONAMENTO-POR-TERMINO* . De fato, em grande parte dos casos não observamos qualquer variação do tempo de execução em função da resolução arbitrária de conflitos ou do uso de alguma das heurísticas auxiliares. A complexidade de *ESCALONAMENTO-POR-TERMINO* é $O(nk^2)$.

Para o *ESCALONAMENTO-SIMPLES*, observamos superioridade acentuada do critério segundo o qual as tarefas que requerem tempos de execução mais longos possuem maior prioridade. Com este critério, a complexidade de *ESCALONAMENTO-SIMPLES*, de $O(nk)$, não se altera.

O que se segue é um estudo do comportamento de *ESCALONAMENTO-SIMPLES* associado ao critério que privilegia tarefas com pesos de execução maiores, e de *ESCALONAMENTO-POR-TERMINO*, com resolução arbitrária de conflitos.

Examinando os gráficos (5.3) e (5.4), e a tabela (5.3), nota-se que ambas as heurísticas alcançaram desempenho semelhante, e muito bom. Em particular, 90% das alocações obtidas por *ESCALONAMENTO-SIMPLES*, e 91.5% das alocações feitas por *ESCALONAMENTO-POR-TERMINO* fornecem tempos de execução que superam os valores ótimos correspondentes em não mais que 30%, e, para quase 100% das alocações de ambos, essa diferença não excede 50%.

Notamos também em (5.3) e (5.4) que para a categoria I, em cujas forças-tarefa há menos arestas de comunicação, os resultados são ligeiramente inferiores. Isto ocorre em decorrência do espaço de soluções ser maior (mais ordenações de tarefas são possíveis). *ESCALONAMENTO-POR-TERMINO* teve melhor desempenho neste caso.

Tabela 5.4: Distribuição $\mathcal{C}_1(\mathcal{S}_1)/\mathcal{C}_1(\mathcal{O}_1)$, função L variável						
Algoritmo	Ótimo	≤ 1.1	≤ 1.2	≤ 1.3	≤ 1.5	≤ 2.0
SIMPLES	12.0%	42.0%	62.0%	78.3%	95.3%	100.0%
TERMINO	15.0%	48.0%	74.0%	90.0%	98.0%	100.0%

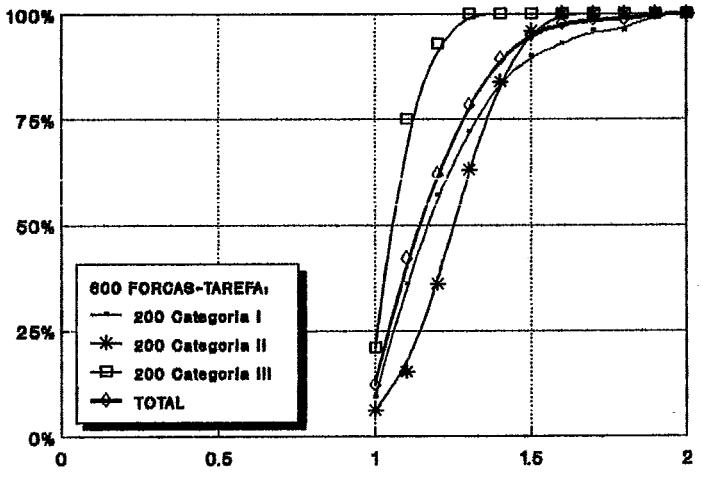


Gráfico 5.5: O Escalonamento Simples e a função $\mathcal{C}_1$, L variável

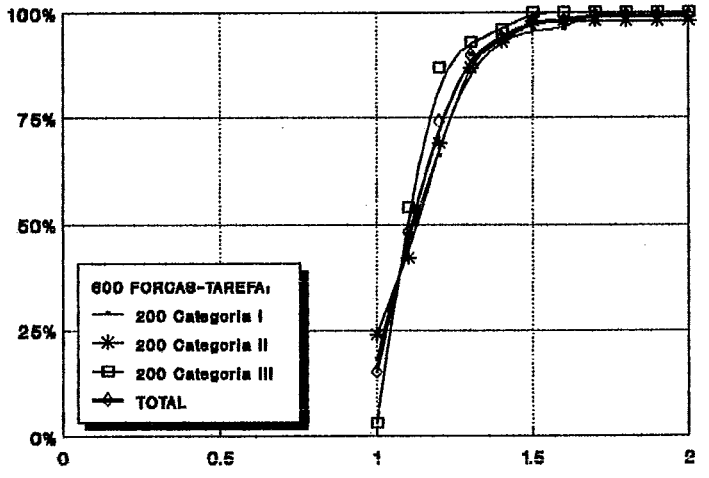


Gráfico 5.6: O Escalonamento por Término e a função $\mathcal{C}_1$, L variável

Acima, mapeamos as forças-tarefa em sistemas onde os custos de comunicação são uniformes. Agora repetimos o experimento para canais com larguras de banda variáveis (no intervalo [1/3,1]).

Nos gráficos 5.5 e 5.6 e na tabela 5.4, podemos observar que *ESCALONAMENTO-POR-TERMINO* tem novamente desempenho muito bom, e, desta vez, consideravelmente melhor do que *ESCALONAMENTO-SIMPLES*. Por exemplo, enquanto 90% das soluções produzidas por *ESCALONAMENTO-POR-TERMINO* possuem tempos de execução que excedem o ótimo em não mais que 30%, para *ESCALONAMENTO-SIMPLES* a porcentagem correspondente se reduz a 78.3%. Contudo, a diferença entre o tempo de execução de quase 100% das alocações produzidas por ambos os algoritmos e o tempo de execução ótimo permanece menor ou igual a 50%.

5.4.1.2. Cortes e "Clustering" e o Somatório de Execução, Comunicação [Inter-Processador e Interferência

Avaliamos o desempenho dos algoritmos de Cortes e "Clustering" de Lo [LO83, LO88], referenciados coletivamente por *A*, para os modelos com e sem interferência. Lo realizou análise similar com uma massa de dados diversa.

Adicionalmente, avaliamos a nossa extensão *A'* do algoritmo para o modelo com interferência.

Os gráficos 5.7 e 5.8 e a tabela 5.5 e mostram que os algoritmos de cortes e "clustering" *A* foram bem sucedidos na otimização de ambos os critérios com e sem interferência. Para ambos os critérios, *A* obteve alocações com valores de $\mathcal{C}_2$ ou $\mathcal{C}_3$ bastante próximos dos respectivos ótimos (diferença de 10% ou menos) em mais de 80% dos casos. No entanto, destacamos que *A* alcançou maior sucesso para o modelo sem

Tabela 5.5: Distribuições $\mathcal{C}_2(\mathcal{S}_2)/\mathcal{C}_2(\mathcal{O}_2)$ e $\mathcal{C}_3(\mathcal{S}_3)/\mathcal{C}_3(\mathcal{O}_3)$						
Algoritmo	Ótimo	≤ 1.1	≤ 1.2	≤ 1.3	≤ 1.5	≤ 2.0
A (sem I)	68.2%	83.9%	91.2%	94.2%	97.4%	100.0%
A (com I)	35.5%	81.5%	96.7%	98.8%	100.0%	100.0%

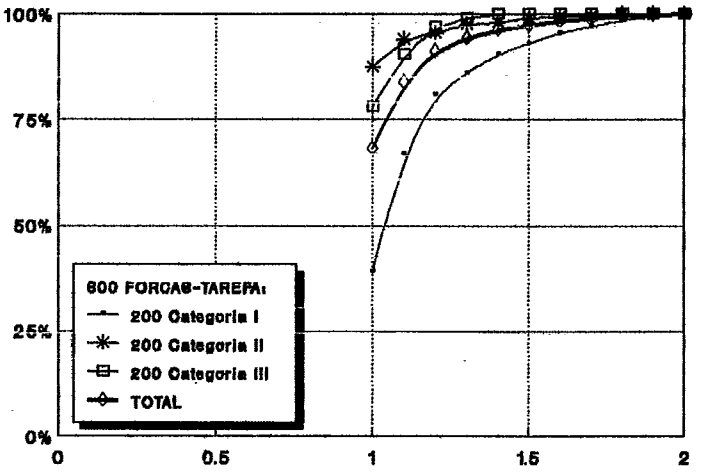


Gráfico 5.7: Cortes e "Clustering" (A) e a função $\mathcal{C}_2$

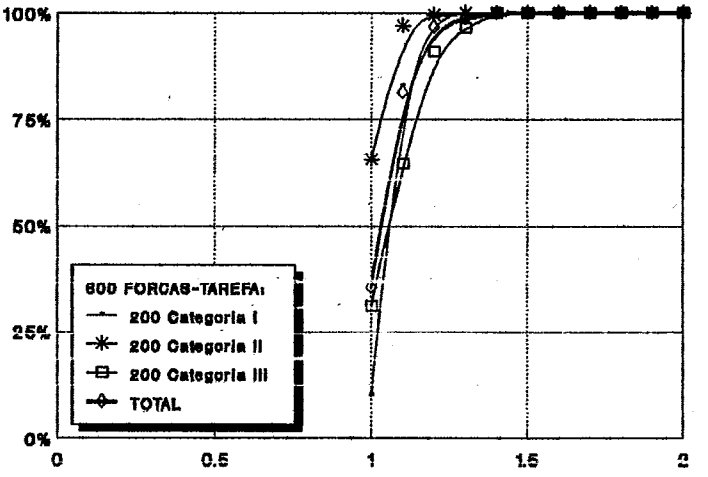


Gráfico 5.8: Cortes e "Clustering" (A) e a função $\mathcal{C}_3$

Tabela 5.6: Distribuição $\mathcal{C}_3(\mathcal{S}_3)/\mathcal{C}_3(\mathcal{O}_3)$						
Algoritmo	Ótimo	≤ 1.1	≤ 1.2	≤ 1.3	≤ 1.5	≤ 2.0
A'(com *I*)	32.7%	86.0%	99.0%	99.8%	100.0%	100.0%

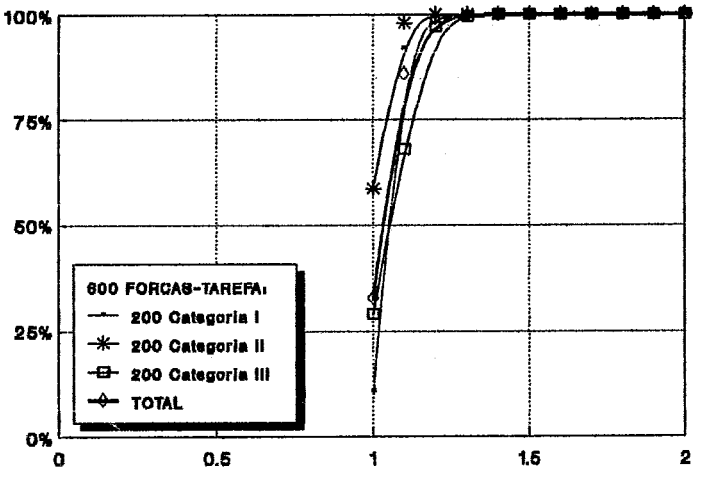


Gráfico 5.9: Cortes e "Clustering" (A') e a função $\mathcal{C}_3$

interferência (68.2% de ótimos quanto a $\mathcal{C}_2$ e 35.5% quanto a $\mathcal{C}_3$).

O gráfico 5.9 e na tabela 5.6 se referem ao algoritmo A'. A' não apresentou resultados significativamente melhores quanto à otimização de $\mathcal{C}_3$, mas sim no balanceamento da carga computacional. Enquanto a maior parte das alocações produzidas por A utilizam apenas um processador, as efetuadas por A' utilizam em média mais de três processadores. Portanto, em comparação com A, A' tende a produzir alocações com a carga computacional mais distribuída.

Outra observação interessante é que *GRAB e LUMP*, os quais são responsáveis pela maior parte do tempo computacional tomado pelos

algoritmos *A* e *A'*, realizaram alocações para aproximadamente 30% dos casos simulados apenas, e, mesmo assim, de somente uma ou duas tarefas, em geral.

5.4.2. Os Algoritmos quanto a Outros Objetivos

As experiências desta seção tratam de avaliar os algoritmos de cortes sucessivos e "clustering" quanto ao tempo total de execução, e de verificar o nível de comunicação inter-processador e de eficiência das alocações produzidas através dos métodos de listas de prioridades e de cortes e "clustering".

5.4.2.1. Cortes e "Clustering" e o Tempo Total de Execução

Comparando valores ótimos dos tempos de execução com os tempos de execução de escalonamentos associados às alocações produzidas pelos algoritmos de cortes e "clustering", examinamos o grau de paralelismo dessas alocações.

Computamos escalonamentos consistentes com as alocações de cortes e "clustering" através do mesmo procedimento descrito na seção 5.2, adotando como modelo o grafo de precedência, e atribuindo uma prioridade a cada tarefa, de acordo com o seu caminho de saída.

Como indicam os gráficos 5.10, 5.11 e 5.12 e as tabelas 5.7 e 5.8, para as alocações produzidas pelo algoritmo *A* no modelo sem interferência, e pelos algoritmos *A* e *A'* no modelo com interferência, obtivemos escalonamentos com tempos de execução extremamente insatisfatórios. A introdução da interferência não resultou em incremento significativo do paralelismo nas alocações produzidas na prática.

Tabela 5.7: Distribuições $\mathcal{C}_1(\mathcal{S}_2)/\mathcal{C}_1(\mathcal{O}_1)$ e $\mathcal{C}_1(\mathcal{S}_3)/\mathcal{C}_1(\mathcal{O}_1)$						
Algoritmo	Ótimo	≤ 1.1	≤ 1.2	≤ 1.3	≤ 1.5	≤ 2.0
A (s/*I*)	1.0%	1.3%	2.3%	4.3%	11.8%	41.1%
A (c/*I*)	1.0%	1.8%	2.6%	4.9%	12.2%	55.9%

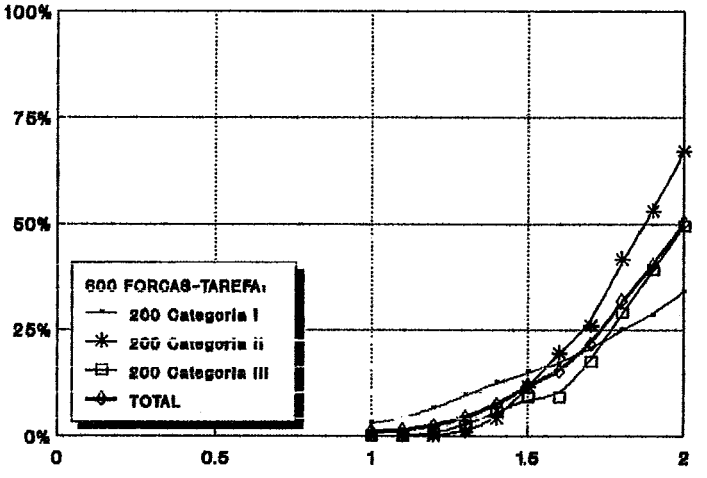


Gráfico 5.10: Cortes e "Clustering"(*A*) e a função $\mathcal{C}_1$, s/Interferência

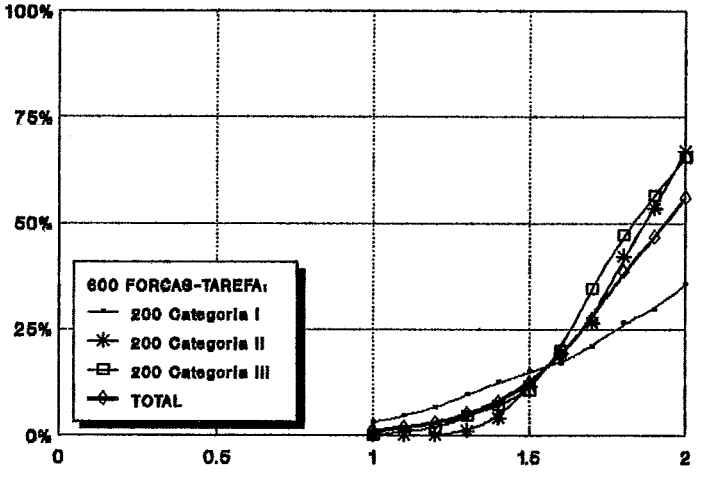


Gráfico 5.11: Cortes e "Clustering"(*A*) e a função $\mathcal{C}_1$, c/Interferência

Tabela 5.8: Distribuição $\mathcal{C}_1(\mathcal{S}_3)/\mathcal{C}_1(\mathcal{O}_1)$						
Algoritmo	Ótimo	≤ 1.1	≤ 1.2	≤ 1.3	≤ 1.5	≤ 2.0
$A'(c/I)$	2.8%	4.1%	6.6%	10.8%	24.0%	67.9%

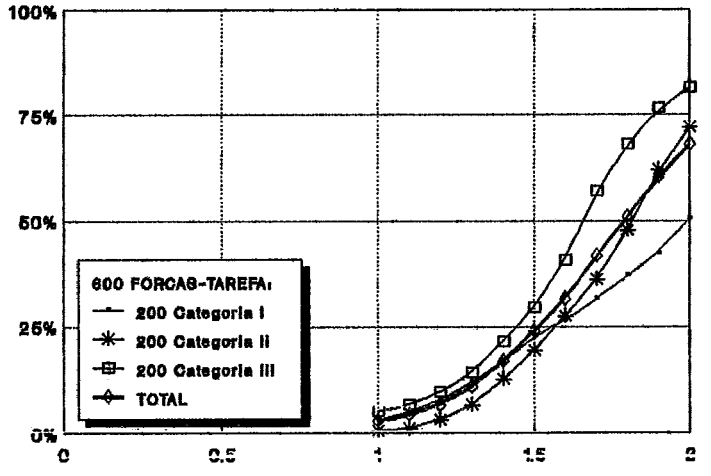


Gráfico 5.12: Cortes e "Clustering"(A') e a função $\mathcal{C}_1$, c/Interferência

5.4.2.2. Volume de Comunicação Inter-Processador

Comparamos a quantidade de comunicação existente entre os módulos componentes da força-tarefa com a quantidade de comunicação inter-processador incorrida pela alocação e com a porção desse valor que causa aumento do tempo total de execução da solução (não sobreposta com o processamento de outras tarefas).

Os algoritmos de listas de prioridade, com o objetivo de minimizar o tempo total de execução $\mathcal{C}_1$, buscam reduzir os atrasos de comunicação, e não a comunicação inter-processador em si. Nas alocações

realizadas para a nossa massa de entrada, constatamos volume de comunicação inter-processador considerável, de 55% a 72% do total, mas a quantidade que causa aumento efetivo do tempo de execução é bem inferior, cerca de 13% em média. Estas proporções não variaram muito em função dos algoritmos ou das heurísticas de prioridade auxiliares, mas sim em função da comunicação existente nas próprias aplicações.

Em contraste, a redução do volume de comunicação inter-processador constitui objetivo direto dos algoritmos de cortes e "clustering", fazendo parte das funções de custo $\mathcal{C}_2$ e $\mathcal{C}_3$. De fato, em todas as alocações realizadas pelo algoritmo *A*, inclusive para o modelo com interferência, observamos níveis bem reduzidos de comunicação inter-processador, não excedendo 10%. Para as alocações resultantes da utilização de *A'*, esses níveis são um pouco mais elevados, em torno de 15%.

5.4.2.3. Eficiência

Utilizamos como medida de eficiência a razão do tempo em que os processadores estão ocupados (processando alguma tarefa) sobre o tempo total de cada processador tomado pela solução.

A eficiência, assim definida, está relacionada com o balanceamento de carga (cargas computacionais desbalanceadas implicam em alguns processadores ociosos durante longos períodos) e com a espera de tarefas para a resolução de dependências (término das predecessoras e recebimento de suas mensagens).

Este experimento pode ser útil também para mostrar a variação do "speedup" em função da topologia do sistema e do número de

processadores.

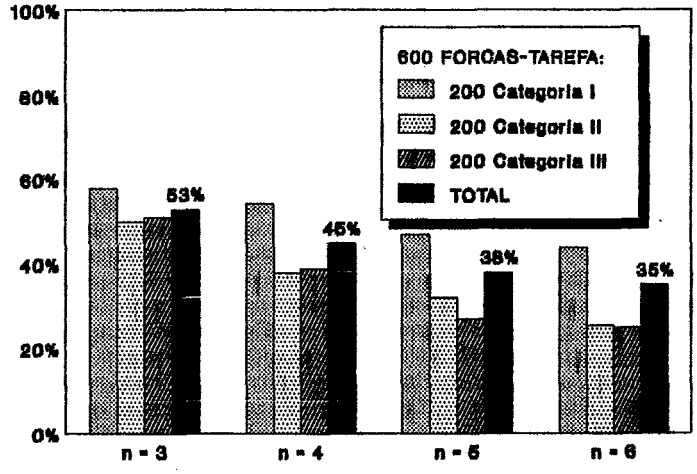


Gráfico 5.13: Escalonamento Simples e a Eficiência

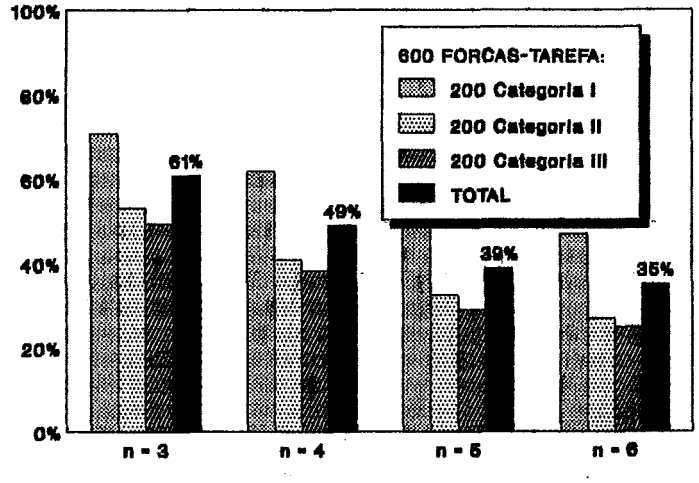


Gráfico 5.14: Escalonamento por Término e a Eficiência

Como podemos conferir nas figuras 5.14 e 5.15, ambos os algoritmos de escalonamento obtém alocações razoáveis quanto à eficiência. Na verdade, os resultados estão muito mais relacionados com o número de processadores e com o potencial de paralelismo de cada

aplicação, portanto, é lógico o fato de termos encontrado os melhores resultados na Categoria I.

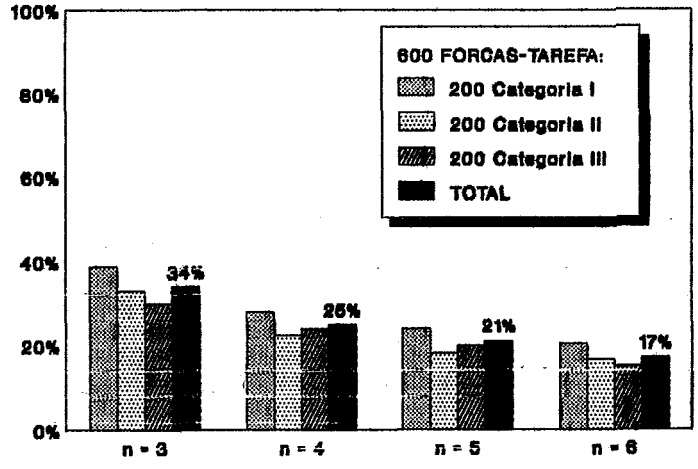


Gráfico 5.15: Cortes e "Clustering"(*A*) e a Eficiência, s/Interferência

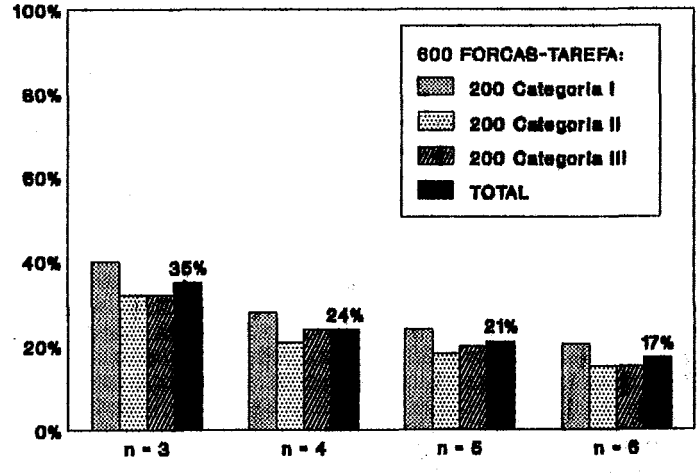


Gráfico 5.16: Cortes e "Clustering"(*A*) e a Eficiência, c/Interferência

Verificamos, pelos gráficos 5.15, 5.16 e 5.17, que a eficiência dos métodos de cortes e "clustering", com e sem interferência, é mais

baixa. Também não se nota uma relação acentuada com o paralelismo das aplicações.

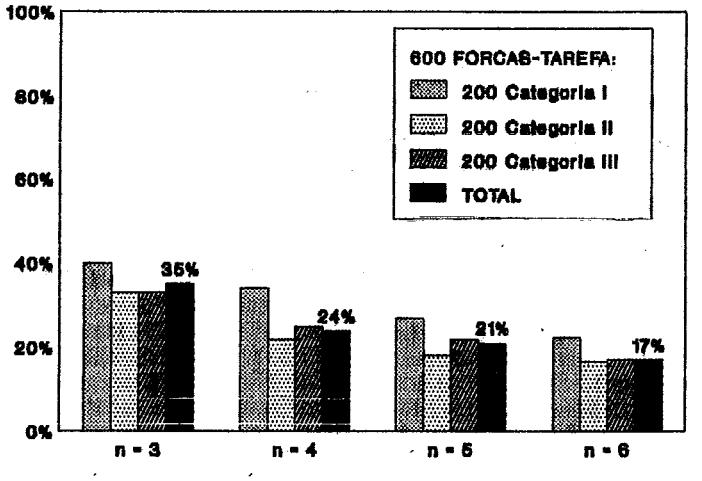


Gráfico 5.17: Cortes e "Clustering"(A') e a Eficiência, c/Interferência

CAPÍTULO 6

CONCLUSÃO

Neste trabalho consideramos a questão da alocação de tarefas em sistemas distribuídos.

Estudamos a resolução subótima do problema através de duas abordagens bem distintas: o escalonamento por listas de prioridade, a fim de minimizar o tempo total de execução; e a alocação por cortes sucessivos e "clustering", visando a minimização do somatório dos custos de execução, comunicação inter-processador e interferência incorridos.

A alocação por cortes e "clustering" utiliza o modelo de fluxo de redes de Stone [STO77], cujo objetivo de desempenho é, originalmente, otimizar o somatório dos custos de execução e comunicação inter-processador. Este critério, uma medida da utilização global dos recursos do sistema, é deficiente, no sentido de que não consegue reconhecer o grau de concorrência das soluções. Conseqüentemente, suas soluções ótimas são, com freqüência, pouco balanceadas.

Este fato motivou a introdução do fator interferência por Lo [LO83, LO88] no modelo de fluxo de redes e na sua função objetivo. Conceitualmente, a interferência envolve todos os custos relacionados com a alocação de tarefas a um mesmo processador. Isto inclui contenção para acesso aos recursos do processador (CPU, memória, dispositivos de E/S, etc.) ao qual as tarefas estão alocadas, e, no caso de tarefas

comunicantes, o custo decorrente dessa comunicação (intra-processador).

Entretanto, assume-se que a interferência, assim como a comunicação, independe dos processadores, o que ocorre no caso de processadores com recursos abundantes e completamente interconectados através de canais de comunicação idênticos ou por um barramento de dados. Esta restrição está associada à modelagem por fluxo de redes, que requer que o custo de separar/ unir cada par de tarefas seja independente dos processadores aos quais as tarefas estão alocadas.

Os algoritmos por cortes e "clustering" são propostas de Lo [LO83, LO88], aplicáveis a ambos os modelos com e sem interferência, porém mais apropriados para este último. Sugerimos algumas alterações simples para tratar melhor o modelo com interferência.

A alocação por listas de prioridade se baseia na teoria clássica de escalonamento determinístico. O modelo é um grafo de precedências, que determina uma relação temporal entre as tarefas, segundo a qual algumas tarefas podem executar apenas após o término de certas outras (suas predecessoras). O objetivo é minimizar o tempo total de execução, e, para isso, é importante determinar não apenas a qual processador cada tarefa será atribuída, mas também o instante em que deverá iniciar.

Estendendo os trabalhos de Shirazi e Wang [SHI90] e de Hwang, Anger, Chow e Lee [HWA89], apresentamos dois algoritmos de escalonamento capazes de tratar atrasos de comunicação relevantes, e processadores heterogêneos interconectados arbitrariamente através de canais com capacidades distintas. Consideramos também a combinação dos algoritmos a alguns critérios heurísticos de prioridade, com a intenção de

incrementar a qualidade das soluções produzidas.

Avaliamos a qualidade dos métodos de listas de prioridade e cortes e "clustering" na otimização de seus respectivos objetivos, e em relação ao volume de comunicação e à eficiência (definida como a porcentagem total do tempo útil dos processadores durante a execução do conjunto de tarefas).

Através de um dos algoritmos de listas de prioridade, obtivemos, em todos os casos simulados, escalonamentos com resultados bastante satisfatórios em termos do tempo de execução. Ambos os algoritmos mostraram bom desempenho quando se tem atrasos de comunicação independentes dos processadores. Nas soluções produzidas, constatamos considerável quantidade de comunicação inter-processador, mas a porção que causa aumento no tempo de execução (não sobreposta com a execução de outras tarefas) é reduzido. A eficiência se mostrou muito relacionada com o potencial de paralelismo das aplicações, sendo relativamente alta para aplicações com grande potencial de paralelismo.

Os algoritmos de cortes e "clustering" foram bem sucedidos na otimização de ambos os critérios com e sem interferência. As modificações que sugerimos para o modelo com interferência não resultaram significativamente melhores neste sentido, mas sim no balanceamento da carga computacional. Outra observação interessante é que os algoritmos de cortes realizaram alocações em poucos casos, e, mesmo assim, de apenas 1 ou 2 tarefas, em geral. O volume de comunicação inter-processador observado foi bem reduzido em todas as alocações realizadas (um pouco superior para o modelo com interferência, quando o

algoritmo estendido é utilizado). A eficiência, calculada sobre escalonamentos consistentes com as atribuições obtidas, é baixa em todos os casos, e não se nota uma relação acentuada com o paralelismo das aplicações. O maior balanceamento da carga pelo modelo com interferência não promove uma melhora da eficiência das alocações. Isto ocorre provavelmente porque o modelo de fluxo de redes não possui informação sobre o sequenciamento entre as tarefas, sendo a carga computacional distribuída pelos processadores sem levar em conta a sincronização entre as tarefas componentes. O resultado é uma tendência a haver muita espera nos processadores.

Investigamos se a introdução da interferência atua no sentido de explorar melhor o paralelismo das alocações. Com esta finalidade, computamos escalonamentos consistentes com soluções ótimas para os modelos com e sem interferência. Então, comparamos os tempos de execução desses escalonamentos com os tempos de execução ótimos. Também examinamos os tempos de execução associados às soluções produzidas pelos algoritmos de cortes e "clustering".

Os resultados desses experimentos indicam que a introdução da interferência tem efeito positivo sobre o tempo de execução. Isto é uma conseqüência indireta da maior distribuição das tarefas entre os processadores. Ainda assim, as soluções ótimas para o modelo com interferência são insatisfatórias com relação a esse critério. Este resultado é ainda pior para as alocações subótimas obtidas na prática.

Há vantagens e desvantagens associados à adoção de cada objetivo de desempenho e seu modelo associado.

O somatório da execução e comunicação como função objetivo é deficiente em termos do balanceamento de carga das alocações produzidas. A introdução da interferência contorna este problema, mas se há intensa sincronização ou precedência entre as tarefas da aplicação, a ociosidade dos processadores em decorrência destes fatores prejudicará substancialmente o desempenho.

No modelo de fluxo de redes, a comunicação e a interferência entre tarefas são independentes dos processadores aos quais as tarefas são alocadas. Assim, a interferência perde parte de seu significado (ligado aos recursos disponíveis em cada processador) e os métodos de fluxo de rede se aplicam apenas a um conjunto restrito de arquiteturas.

Além disso, apesar de que a interferência é um conceito interessante, é difícil inferir aproximações adequadas, e o seu uso acaba por aumentar o grau de imprecisão na representação do problema.

A minimização do tempo de execução requer as tarefas representadas através de grafos de precedências. Este modelo impõe algumas limitações críticas, como a não existência de ciclos e, sendo a comunicação representada através de arcos direcionados, a não ocorrência de troca de mensagens entre as tarefas.

Outra dificuldade é que, neste caso, é importante definir não só a atribuição de tarefas a processadores, mas também o instante em que as tarefas deverão iniciar. Isto acrescenta um grau de complexidade ao problema. Para tratá-lo, convém assumir que não há preempção, isto é, que as tarefas não são interrompidas uma vez que começam a executar. Além disso, na realidade, as tarefas serão iniciadas assim que

efetivamente habilitadas. Quando o início real de uma tarefa é posterior ao instante determinado no escalonamento (devido a contenção por recursos compartilhados, trocas de contexto ou mesmo pela não exatidão dos pesos de execução e comunicação estimados), haverá um atraso que poderá se propagar até o caminho crítico. Conseqüentemente, os escalonamentos poderão ter níveis de desempenhos bastante aquém do esperado.

REFERÊNCIAS BIBLIOGRÁFICAS

[ADA74] Adam,T.L.,Chandy K.M., and Dickson J.R.,"A Comparision of List Schedules for Parallel Processing Systems,"*CACM*,No.12,Vol.17, December 1974,pp.685-690.

[ANG90] Anger F.D.,Hwang J.-J.,Chow Y.-C.,"Scheduling with Sufficient Loosely Coupled Processors,"*Journal of Parallel and Distributed Computing*,No.9,1990,pp.87-92.

[BOK81a] Bokhari,S.H.,"On the Mapping Problem,"*IEEE-TC*,Vol.C-30,No.3, March 1981,pp.207-214.

[BOK81b] Bokhari,S.H.,"A shortest tree algorithm for optimal assignments across space and time in a distributed processor system,"*IEEE--TSE*,Vol.7,No.6,November 1981,pp. 583-589.

[BRU74] Bruno,J.,Coffman,E.G.,Jr., and Sethi,R."Scheduling Independent Tasks to Reduce Mean Finishing Time,"*CACM*,Vol.17,No.7,July 1974,pp. 382-387.

[CHO82] Chou,T.C.K.,and Abraham,J.,"Load Balancing in Distributed Systems,"*IEEE-SE*,Vol.SE-8,No.4,July 1982,pp.401-412.

[CHU80] Chu,W.W.,Holloway,L.J.,Lan,M.-T., and Efe,K., "Task Allocation in Distributed Data Processing,"*IEEE-TC*,Vol.13,No.11,Nov.1980, pp.57-69.

[CHU87] Chu,W.W., and Lan,L.M.-T.,"Task Allocation and Precedence Relations for Distributed Real-Time Systems,"*IEEE-TC*, Vol.C-36, No.6, June 1987,pp.667-679.

[COR79] Cornett,D.H. and Franklin,M.A.,"Scheduling Independent Tasks with Communications,"Washington University, Dept. of Electrical Engineering, Technical Report, 1979

[DIN70] Dinic,E.A.,"Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation,"*Soviet Math. Dokl.*, No.11, 1970, pp.1277-1280.

[ERC90] Ercal,F.,Ramanujam,J., e Sadayappan,P.,"Task Allocation onto a Hypercube by Recursive MinCut Bipartitioning,"*Journal of Parallel and Distributed Computing* 10,1990,pp.35-44.

[EDM72] Edmonds,J.,and Karp,R.M.,"Theoretical improvements in algorithmic efficiency for network flow problems,"*J.of the ACM*, No.19, 1972, pp.248-264.

[EFE82] Efe,K.,"Heuristic Models of Task Assignment Scheduling in Distributed Systems,"*IEEE Computer*,Vol.15,No.6,June 1982, pp.50-56.

[EZZ86] Ezzat,A.K., Bergeron,R.D, and Pokoski,J.L., "Task Allocation Heuristics for Distributed Computing Systems," *IEEE*, 1986, pp.337-346.

[FER73] Fernández,E.B., and Bussel,B.,"Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules,"*IEEE-TC*,Vol.C-22,No.8,August 1973,pp.745-751.

[FOR62] Ford,L.R.,and Fulkerson,D.R., *Flows in Networks*, Priceton University Press, Princeton, N.J., 1962

[GOL88] Goldberg,A.V. and Tarjan,R.E.,"A New Approach to the Maximum-Flow Problem,"*J.of the Association for Computing Machinery*, Vol.35,No.4,October 1988,pp. 921-940.

[GRA69] Graham,R.L.,"Bounds on Multiprocessing Timing Anomalities,"*SIAM J.Applied Math*,Vol.17,No.2,March69,pp. 416-429.

[GUR81] Gursky,M.,"Some Complexity Results for a Multi-processor Scheduling Anomalities,"private communication from H.S.Stone,

[GYL76] Gylys,V.B.,and Edwards,J.A.,"Optimal Partitioning of Workload for Distributed Systems,"*COMPCON*,Fall 1976.

[HOR78] Horowitz,E. e Sahni,S.,*Fundamentals of Computer Algorithms*, Computer Science Press, Rockville,1978.

[HWA89] Hwang,J.-J.,Angers F.D., and Lee,C.-Y.,"Scheduling Precedence Graphs in Systems with Interprocessor Communication Times,"*SIAM Journal Comput.*,Vol.18,No.2,April 1989,pp.244-257.

[KAR72] Karp,R.M.,"Reducibility Among Combinatorial Problems,"Dept. of Computer Science,U. of Calif. at Berkeley,TR-3,April 1972.

[KAR74] Karzanov,A.V,"Determining the Maximal Flow in a Network by the Method of Preflows,"*Soviet Math. Dokl.*,Vol.15, No.2, 1974, pp.434-437.

[KHE88] Khei,H.H.,"Técnicas para a Alocação Estática de Tarefas em Sistemas Distribuídos",Universidade Federal do Rio de Janeiro, Tese de Mestrado,Junho 1988.

[KIM88] Kim,S.J.,"A General Approach to Multiprocessor Scheduling",Ph.D. Thesis,The Univ.of Texas at Austin,December 1988.

[KRU87] Kruatrachue,B.,"Static Task Scheduling and Grain Packing in Parallel Processing Systems,",Ph.D. Thesis, Department of Computer Science, Oregon State Univ., 1987.

[LEE87] Lee,S.-Y., and Aggarwal,J.K.,"A Mapping Strategy for Parallel Processing",*IEEE-TC*,Vol.C-36,No.4,April 1987,pp.433-441.

[LEU89] Leung,J.Y.-T., and Young,G.H.,"Minimizing Schedule Length Subject to Minimum Flow Time,"*SIAM Journal Comput.*,Vol.18,No.2, April 1989,pp.314-326.

[LO83] Lo,V.M.,"Task Assignment in Distributed Systems,", Dept.of Computer Science, Univ.of Illinois, Ph.D. Thesis, October 1983.

[LO88] Lo,V.M.,"Heuristic Algorithms for Task Assignment in Distributed Systems,"*IEEE-TC*,Vol.37,No.11,November 1988,pp.1384-1397.

[MA82] Ma,P.-Y.R.,Lee,E.Y.S., and Tsuchiya,M.,"A Task Allocation Model for Distributed Computing Systems,"*IEEE-TC*,Vol.C-31,No.1, January 1982,pp.41-47.

[MAR67] Martin,D.E.,and Estrin,G.,"Models of Computational Systems - Cyclic to Acyclic Graph Transformations,"*IEEE-TC*,Vol.C-16,No.1, February 1967,pp.70-79.

[MAL78] Malhotra,V.M.,Pramodh Kumar,M.,and Maheshwari,S.N.,"An $O(|V^3|)$ Algorithm for Finding Maximum Flows in Networks,"Indian Institute of Technology, Computer Science Program, Kanpur 208016, India, 1978.

[PAP87] Papadimitriou,C.H., and Ullman J.D.,"A Communication-Time Trade Off",*SIAM Journal Comput.*,Vol.16,No.4,August 1987,pp.639-646.

[PRA87] Prastein,M.,"Precedence-Constrained Scheduling with Minimum Time and Communication,",MS.Thesis, Univ.of Illinois, 1987.

[REW90] El-Rewini,H. and Lewis,T.G.,"Scheduling Parallel Program Tasks onto Arbitrary Target Machines,"*Journal of Parallel and Distributed Computing*,No.9,1990,pp.138-153.

[SHE85] Shen,C.-C., and Tsai,W.-H."A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion,"*IEEE-TC*,Vol.C-34,No.3,March 1985,pp.197-203.

[SHI90] Shirazi,B.,Wang M., and Pathak G.,"Analysis and Evaluation of Heuristic Methods for Static Task Scheduling," *Journal of Parallel and Distributed Computing*,No.10,1990,pp.222-232.

[SIN84] Sinclair,J.B. and Lu,M.,"Module Assignment in Distributed Systems,"Proc.1984 Computer Networking Symposium, Gaithersburg, MD, December 1984,pp.105-111.

[SIN87] Sinclair,J.B.,"Efficient Computation of Optimal Assignments for Distributed Tasks,"*Journal of Parallel and Distributed Computing*,No.4,1987,pp.342-362.

[SLE81] Sleator,D.,"An order O(*nm*) Algorithm for Maximum Network Flow," Dept. of Computer Science, Stanford University, TM-STAN-CS-80-831, 1981.

[STO77] Stone,H.S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms,"*IEEE-SE*,Vol.SE-3,No.1,Jan.1977,pp.85-93.

[STO78] Stone,H.S., and Bokhari,S.H.,"Control of Distributed Processes," *IEEE Computer*,Vol.11,No.7,July 1978,pp.97-106.

[SZW86] Szwarcfiter,J.L.,*Grafos e Algoritmos Computacionais*,2.Ed.,Ed. Campos,Rio de Janeiro,1986.

[ULL75] Ullman,J.D.,"NP-complete Scheduling Problem,"*Journal of Computer System Science*,Vol.10,1975,pp.384-393.

[WU80] Wu,C.S.,and Liu,M.T.,"Assignment of Tasks and Resources for Distributed Processing,"*COMPCON*,Fall 1980,pp.665-662.