Algoritmos de Escalonamento em Multiprocessadores com Arquitetura Heterogênea

Stella C. da S. Porto *

Daniel A. Menascé **

RESUMO

Arquiteturas Paralelas Heterogêneas têm-se mostrado uma área promissora de pesquisa no desenvolvimento de computadores de alto desempenho. O escalonamento de tarefas neste ambiente computacional implica em diversos problemas, já que decisões sobre **quais** processadores serão alocados (e não apenas **quantos** processadores devem ser alocados) tem que ser tomadas. Este artigo apresenta vários algoritmos de escalonamento estático para multiprocessadores heterogêneos.

ABSTRACT

Heterogeneous Parallel Architectures have been shown to be a promising area of research in the development of high performance computers. Scheduling of tasks in such a framework poses difficult problems since decisions as to **which** processors to allocate (and not only **how many** processors should be allocated) have to be taken. This paper presents several static scheduling algorithms for heterogeneous multiprocessors.

---

*

Estagiária do Centro Científico Rio - IBM Brasil;
Mestranda em Informática pela - PUC/RJ;
Áreas de Interesse: Algoritmos de Escalonamento em Arquiteturas Paralelas, Multiprocessadores Homogêneos e Heterogêneos.
Endereço: Caixa Postal 4624, telefone: 271-2802.

**

Prof. Associado do Departamento de Informática, PUC/RJ;
PhD. em Ciência da Computação pela UCLA;
Áreas de Interesse: Modelagem Analítica de Desempenho, Processamento Paralelo e Arquiteturas de Alto Desempenho.

Endereços: 22453 Rio de Janeiro - RJ/Brasil, telefone: 529-9526 ou menascé@brlncc.bitnet.

1. INTRODUÇÃO

Os multiprocessadores formam uma categoria bastante vasta e diversificada de máquinas com estrutura paralela e podem ser definidos como sendo sistemas de computação formados por um conjunto de elementos processadores independentes que trabalham de maneira coordenada em prol de um objetivo comum. Como exemplos de máquinas multiprocessadoras temos o IBM 3081 [1], NYU UltraComputer [2], o CRAY-XMP [3] e a Connection Machine [4] (as três primeiras com memória compartilhada e o último com uma organização distribuída).

O enfoque do multiprocessamento introduz três requisitos não encontrados em outros tipos de arquiteturas paralelas: cada problema (aplicação) deve ser particionado em subproblemas (tarefas); cada tarefa deve ser designada para execução num dado processador do sistema e o o fluxo de dados deve ser sincronizado ao longo de todo o período de execução.

O desempenho de multiprocessadores está intimamente relacionado tanto à organização do hardware, quanto à organização da aplicação que será executada. Podemos listar parâmetros que possuem influência significativa no desempenho de multiprocessadores: 1) O paralelismo existente na aplicação, 2) O método de decomposição de um problema em subproblemas menores, 3) O método utilizado para a alocação destes subproblemas a processadores, 4) A granularidade do subproblema executado em cada processador, 5) A possibilidade de simultaneidade entre processamento e comunicação entre processadores, 6) O modo de acesso aos dados , onde os dados podem ser acessados, tanto diretamente de uma memória global, ou copiados para uma memória local e então acessados, 7) A estrutura de interconexão dos processadores, 8) A velocidade dos processadores, memórias e a rede de interconexão.

As tarefas de uma aplicação paralela devem ser executadas de acordo com uma ordenação, que reflete o fluxo de dados da aplicação. Esta ordenação pode ser representada por relações de precedência entre as tarefas e traduz a fração sequencial de execução existente em qualquer aplicação paralela. Isto descreve a primeira instância do problema de escalonamento: dada uma aplicação composta por um conjunto de tarefas que obedecem a certas relações de precedência, como designar tarefas a processadores com o intuito de otimizar o desempenho.

A segunda instância do problema decorre da utilização de estruturas mutliprocessadoras em ambientes multiprogramados. De acordo com Sevcik [5], multiprocessadores com centenas ou mesmo milhares de processadores independentes têm sido dedicados a aplicações individuais, ou compartilhados entre diversas aplicações seriais. Tal

compartilhamento serial não leva a uma utilização eficiente do sistema como um todo por dois motivos. Primeiro, o tempo para terminar uma aplicação e começar outra é substancial. Segundo, diversas aplicações paralelas não são capazes de manter uma alta proporção dos processadores ocupados continuamente. Quanto maior o número de processadores um sistema possui, menor o número de aplicações que podem fazer uso de todos os processadores simultaneamente.

Existem muitos estudos quanto a possíveis estratégias para a determinação do **número** de processadores que devem ser designados a uma aplicação. Sevcik [5] realiza um estudo de algoritmos de escalonamento estático que iniciam uma aplicação, alocando um certo número de processadores a ela até o seu término. Em [6], apresenta-se uma investigação preliminar das características de desempenho de algumas disciplinas de escalonamento para sistemas paralelos multiprogramados. Em [9], são consideradas estratégias alternativas para escalonamento, que determinam um certo número de processadores para uma dada aplicação paralela.

Até o momento, toda a pesquisa realizada a respeito de multiprocessadores tem se concentrado sobre as estruturas homogêneas, onde os elementos processadores são idênticos, tanto funcionalmente quanto em relação à capacidade de processamento. Nestes casos, o problema de escalonamento, além da questão brevemente desenvolvida acima, sobre **quantos** processadores devem ser designados a cada aplicação (caso de multiprogramação), aborda ainda a definição de quais tarefas possuem prioridade de alocação.

De maneira geral, o escalonamento pode ser estático ou dinâmico. No caso estático, as tarefas são alocadas aos processadores antes da execução do programa paralelo. Os custos do escalonamento são pagos apenas uma vez mesmo que o programa seja rodado diversas vezes; além disso, não existe qualquer "overhead" durante a execução. A principal desvantagem é que as decisões de alocação se baseiam em valores estimados para o tempo de execução de cada uma das tarefas. Ao contrário, o escalonamento dinâmico, realizado em tempo de execução, apresenta um grau menor de incerteza quanto às informações utilizadas para a alocação.

Além deste tipo de classificação (estática e dinâmica) para os algoritmos de escalonamento, existe uma gama muito grande de critérios de avaliação e estudo destes algoritmos. Em [7], é apresentada uma taxonomia bastante pormenorizada dos enfoques dados ao problema de gerenciamento de recursos na tentativa de prover uma terminologia comum e um mecanismo de classificação necessários à discussão do problema. Muitos estudos tem sido feitos sobre algoritmos de escalonamento de tarefas em estruturas homogêneas [8, 10, 11, 12, 13 e 14].

O estudo desenvolvido em [16] serve como pano de fundo e principal motivação para a realização deste relatório. O artigo faz uma análise do paradigma do processamento paralelo heterogêneo, focalizando o compromisso entre custo e desempenho. Já foi demonstrado em [18] que a fração serial de qualquer processamento paralelo domina o tempo de execução em qualquer arquitetura paralela, o que limita as vantagens oferecidas por este tipo de arquitetura. A introdução da heterogeneidade nestas estruturas surge da conclusão de que processadores de maior capacidade são capazes de reduzir significativamente o tempo de processamento destas frações seriais e melhorar consequentemente o desempenho da aplicação, em comparação com arquiteturas que utilizam um número maior de processadores paralelos de menor capacidade num custo total equivalente. A arquitetura paralela heterogênea, enfim, faz uso do melhor de dois mundos: combina a velocidade de processamento, que é característica de um único processador de grande capacidade presente numa arquitetura centralizada, com o crescimento ilimitado de um conjunto de processadores homogêneos, mais baratos e de menor capacidade, característico de uma estrutura paralela. O artigo [16] mostra que a arquitetura heterogênea exibe um melhor desempenho (medido através da diminuição do tempo de execução de aplicações paralelas) quando comparadas com arquiteturas paralelas homogêneas ou centralizadas de custo equivalente.

A questão do escalonamento de tarefas, agora numa estrutura paralela e heterogênea, apresenta um aspecto adicional bastante interessante, não existente no caso das estruturas homogêneas: além de determinar qual tarefa deverá ser executada, é necessário determinar **onde** esta tarefa será executada. Isto porque não é mais irrelevante **qual o processador** a ser escolhido para a execução de uma dada tarefa, pois os processadores do sistema heterogêneo não a executam de maneira idêntica.

Os estudos de algoritmos de escalonamento já realizados considerando uma estrutura multiprocessadora heterogênea, como o artigo [15], consideram o escalonamento num ambiente onde o conjunto de tarefas a serem escalonadas é composto por tarefas independentes, que não possuem qualquer relação de precedência entre si.

O presente trabalho apresenta algoritmos para o escalonamento de tarefas de aplicações paralelas com regras de precedência fixas e pré-determinadas em um sistema multiprocessador, formado por elementos processadores heterogêneos, caracterizados pela velocidade de execução das tarefas.

A próxima seção descreve de forma mais detalhada e formal o ambiente da arquitetura heterogênea, o ambiente das aplicações paralelas e a questão do escalonamento de

tarefas. A seção 3 apresenta os algoritmos de escalonamento propriamente ditos e a seção 4 tece algumas considerações finais.

2. DESCRIÇÃO DE CONCEITOS BÁSICOS

2.1. Descrição do Ambiente de Arquiteturas Heterogêneas

O sistema considerado para a adaptação e criação dos diversos algoritmos de escalonamento de tarefas é composto por múltiplos processadores, que formam grupamentos (classes) distintos. Cada grupamento é composto por processadores idênticos, tanto funcionalmente, quanto em relação a sua capacidade de processamento. As diversas classes se distinguem apenas com relação à capacidade de processamento dos processadores. Todas as classes são funcionalmente idênticas e possuem o mesmo conjunto de instruções que pode ser dividido em I partições. Cada partição do conjunto de instruções é caracterizada por possuir um tempo específico de execução em cada uma das classes de processadores. A cada classe de processadores está associado um vetor $\mathbf{TI}_p$ que a caracteriza. Este vetor $\mathbf{TI}_p = (t_{1p}, t_{2p}, \ldots, t_{Ip})$ tem como componentes o valor t_{ip}, que é o tempo de execução de uma instrução do tipo i, onde $i = 1, \ldots, I$, nos processadores da classe p.

Uma característica importante do ambiente de execução aqui considerado, é que ele é monoprogramado. Isto significa que apenas uma aplicação é executada no sistema a cada vez. No item 2.2, a aplicação paralela aqui considerada será então caracterizada. Mas, é importante lembrar que não existe qualquer preocupação quanto ao escalonamento de processadores para diferentes aplicações; as aplicações a serem executadas formam uma fila, e serão executadas nesta ordem, quando o sistema for liberado.

2.2. Descrição das Aplicações Paralelas

O tipo de aplicação considerada no decorrer do estudo a ser feito sobre arquiteturas heterogêneas pode ser qualificada como uma aplicação paralela composta por um número inteiro de tarefas ou módulos. Estas tarefas possuem uma relação de dependência entre si, que pode ser traduzida ou representada por grafo de tarefas ou grafo de precedência. Neste grafo, os nós representam tarefas, que podem ser operações independentes ou partes de um programa único e relacionadas entre si no tempo; os arcos direcionados representam as relações de precedência. Uma tarefa T_j só se torna executável quando todas as tarefas que a precedem já foram executadas. Note-se que os grafos são acíclicos; não existe qualquer tipo de ciclo ou "loop" interno ao grafo. A existência de um ciclo num grafo impede o escalonamento estático de um grafo, pois a condição que controla o número de interações não pode ser resolvida até o momento de execução. Além disso, o

grafo não possui nós condicionais ou nós de decisão, (um nó de decisão é aquele, cujo resultado após sua execução pode afetar o fluxo de controle de um programa).

As tarefas são unidades indivisíveis de processamento lógico, e não podem ter sua execução interrompida por qualquer outra tarefa.

Cada tarefa T_t possui um vetor $\mathbf{I}_t$ a ela associado. Este vetor $\mathbf{I}_t = (n_{1t}, n_{2t}, \ldots, n_{It})$ tem como componentes valores do tipo n_{it} = número de instruções do tipo i da tarefa t (com i = 1,...,I), sendo que I é o número de tipos de instruções com um tempo de execução idêntico em cada uma das classes de processadores.

Como observação final, vale ressaltar que as aplicações paralelas, aqui consideradas, são do tipo *CPU bound*. São tipicamente aplicações científicas que fazem uso quase que essencialmente dos recursos de processamento de CPU de um sistema. Sendo assim, todos os recursos de E/S não são aqui considerados para efeito de escalonamento: as tarefas não liberam um processador para executar qualquer instrução de E/S.

2.3. Escalonamento

Um algoritmo de escalonamento é um conjunto de regras e critérios que devem ser seguidos para a associação das diversas tarefas aos processadores com objetivos diversos, como por exemplo: minimização do tempo de execução, balanceamento da carga entre os processadores e maximização da utilização dos recursos.

Consideramos neste trabalho apenas os algoritmos estáticos não-preemptivos, apesar de que as disciplinas preemptivas geram eventualmente escalonamentos melhores do que os gerados por disciplinas não-preemptivas.

Algumas medidas de desempenho utilizadas com frequência para avaliar a eficiência dos algoritmos de escalonamento são: tempo de término da aplicação (tempo de execução); número de processadores necessários ao sistema e utilização dos processadores. O objetivo dos algoritmos aqui apresentados é o de minimizar o tempo de execução da aplicação.

A questão do escalonamento pode ser tratada através de informações locais ou informações globais. No primeiro caso, considera-se que no momento de decisão da alocação de um grupo de tarefas, apenas os dados sobre o tempo de execução destas tarefas nos processadores do sistema são relevantes e analisados. No segundo, dados sobre a topologia e o subgrafo associado a cada tarefa (consideramos como subgrafo de

uma tarefa i, a sequência de tarefas sucessoras diretas e indiretas desta tarefa i até a tarefa final do grafo) são analisados na tentativa de buscar uma alocação ótima.

3. ALGORITMOS DE ESCALONAMENTO

3.1. Definições

Considera-se um conjunto de tarefas parcialmente ordenadas $T=\{T_1,T_2,...,T_n\}$ e G seu grafo de precedência. No caso de estruturas heterogêneas, os tempos estimados são informações um pouco mais complexas, já que dependem do processador onde cada tarefa será executada. Como mencionamos durante a introdução, faz-se uso do vetor **I** e de um vetor **TI** para a caracterização de uma tarefa e de um processador respectivamente. Repetimos aqui a definição dos vetores **I** e **TI**:

- $\mathbf{I_t} = (n_{1t},...,n_{lt})$; onde n_{it} determina o número de instruções do tipo i que compõem a tarefa t.
- $\mathbf{TI_p} = (t_{1p},...,t_{lp})$; onde t_{ip} determina o tempo de execução de uma instrução do tipo i no processador p.

A partir destes dois vetores, pode-se construir um vetor $\mathbf{TE_t}=(t_1',...,t_p')$ onde t_i' é o tempo médio estimado para a realização da tarefa t no processador i. Por definição pode-se estabelecer que:

$$\mathbf{TE_t} = (\mathbf{I_t} \,.\, \mathbf{TI_1}, \ldots, \mathbf{I_t} \,.\, \mathbf{TI_p})$$

Onde cada termo do vetor $\mathbf{TE_i}$ representa o tempo estimado para a execução da tarefa $\mathbf{T_i}$ no processador $\mathbf{P_j}$.

O grafo de precedência G é formado por arcos orientados que ligam vértices. Cada vértice i de G representa uma tarefa T_i e cada arco orientado entre um vértice j (onde se origina o arco) e um vértice k (onde o arco termina) representa a relação de precedência entre as tarefas T_j e T_k, determinando que T_j é uma tarefa predecessora de T_k ou T_k é uma tarefa sucessora de T_j. Além disto, uma tarefa T_i só se torna executável quando todas as suas predecessoras foram executadas. Chamamos de caminho orientado entre dois vértices j e k ($j<k$) a uma sequência de arcos e vértices consecutivos que devem ser percorridos para se ir do vértice j ao vértice k, incluindo os próprios (j e k). Um vértice de saída de um grafo G é aquele que não possui nenhum sucessor e um vértice de entrada de um grafo G é aquele que não possui nenhum predecessor. Supomos que todos os grafos G aqui considerados possuem apenas um vértice de entrada e apenas um vértice de saída.

Um sistema de tarefas com **n** tarefas e **m** processadores pode ser representado por uma matriz μ do tipo nXm com entradas em R^+ ($\infty \in R$) para $n,m \geq 1$ onde para toda a

tarefa t existe um processador p ($1 \leq t \leq n$, $1 \leq p \leq m$) tal que $\mu(t,p)$ seja o tempo estimado de execução da tarefa t no processador p. Este valor, denominado anteriormente de t_{tp}, é obtido a partir dos vetores $\mathbf{TE}_t$ associados a cada tarefa t. O caso em que $\mu(t,p) = \infty$ significa que um processador p é incapaz de executar uma tarefa t. Assim, é possível representar situações nas quais determinados processadores têm funções específicas e não realizam algumas instruções dentre o conjunto completo de instruções considerado.

A execução de tarefas é modelada pela noção de função de alocação para um sistema de tarefas. Uma função de alocação **A** é um mapa $\mathbf{A} : \mathbf{T} \rightarrow \mathbf{P}$ onde $T=\{1,...,n\}$ e $P=\{1,...,m\}$, tal que $\mu(\mathbf{t,A(t)}) \in R^+$ para $t \in T$. Se $A(t)=p$, se diz frequentemente que a tarefa t está alocada ao processador p.

O tempo de término de uma aplicação, dada uma alocação de tarefas A, denominado de f(A), representa o tempo necessário para que os m processadores terminem de executar as n tarefas da aplicação. Uma alocação ótima é a alocação que determina o menor tempo de término em comparação com qualquer outra possível alocação para uma dada aplicação.

É útil associar uma função de tempo de início com a função de alocação **A**. Intuitivamente, para $t \in T$ o valor **s(t)** representa o tempo no qual o processador **A(t)** começa a executar a tarefa t. Formalmente, a função de tempo de início **s** é um mapa **s**: $\mathbf{T} \rightarrow \mathbf{R}$ cujo valor é chamado de tempo de início para a tarefa t. A tarefa t está sendo executada no processador p no instante x se $p=A(t)$ e $s(t) \leq x \leq s(t) + \mu(t,p)$. A função de tempo de término de uma tarefa t é $f(t) = s(t) + \mu(t,A(t))$. De maneira similar o tempo de término do conjunto de tarefas que pertencem a T é $F(T) = \max_T f(t)$.

A função A(t) deve satisfazer a:

1) para $p=1;...,m$ no máximo uma tarefa está sendo executada em qualquer instante num processador p.
2) para $p=1,...,m$ se $s(t) \leq x \leq f(t)$ e $A(t)=p$, então existe uma tarefa sendo executada no instante x no processador p.
3) para $p=1,...,m$ apenas tarefas executáveis podem ser iniciadas no processador p.

Intuitivamente, a primeira condição força todas as tarefas alocadas a um processador a serem executadas sequencialmente; a segunda condição determina que a tarefa t é executada no processador p, ao qual foi previamente alocada, sem qualquer interrupção, e a terceira condição não permite que tarefas, que não sejam executáveis (permissão através do grafo de precedência), sejam iniciadas em qualquer processador.

3.2. Estrutura Geral para o Escalonador Básico

Os algoritmos apresentados são estáticos, ou seja todo o escalonamento das tarefas de uma dada aplicação é realizado antes do período de execução. Quando a aplicação começa a ser executada, a alocação já está definida; cada processador possui uma fila de tarefas a ele associadas, e as executa em sequência, assim que se tornem executáveis.

Existem diversas heurísticas para se realizar a escolha do processador que deverá ser alocado a uma determinada tarefa. Inicialmente apresentamos uma estrutura geral para o escalonador, denominado **Escalonador Básico**, que possa ser utilizada por qualquer tipo de algoritmo estático. Esta estrutura geral deixa em aberto o passo em que a heurística é utilizada na escolha do processador p, tal que $A(t)=p$. Assim, fica simples descrever posteriormente apenas estes critérios de escolha, sem que seja necessário descrever o algoritmo por inteiro.

O funcionamento do **Escalonador Básico** consiste na realização de uma **simulação** do processo real de execução. Nesta **simulação**, a decisão a respeito de qual processador alocar a cada tarefa, será tomada utilizando-se heurísticas de escalonamento especificadas na seção 3.3.

Duas outras variáveis devem ser definidas para construção da estrutura geral do **Escalonador Básico**: **temp** e $\mathbf{livre}_p$. A primeira mede o tempo de simulação e funciona como um relógio utilizado para marcar todos os eventos da simulação. Já a variável $livre_p$ determina o último momento em que o processador p se tornou livre (desocupado).

As tarefas podem ser classificadas de acordo com o estado em que se encontram no decorrer da simulação no **Escalonador Básico**:

- **não-executável** → a tarefa ainda não está num estado de poder ser alocada a um processador.
- **executável** → a tarefa está em condições de ser alocada a um processador.
- **em execução** → a tarefa já foi alocada a um processador e está em execução neste processador escolhido.
- **executada** → a tarefa terminou sua execução no processador ao qual foi alocada.

Os processadores podem ser classificados como **ocupados** ou **livres** (desocupados).

Na figura 1 apresentamos a Estrutura do **Escalonador Básico**, utilizada por todas as heurísticas apresentadas na próxima seção.

Dados de Entrada:

- m ← número de processadores do sistema;
- n ← número de tarefas do grafo da aplição;
- informações sobre as relações de precedência entre as tarefas, expressas através do grafo de tarefas.
- matriz $\mu(T_i,P_j)$, que determina os tempos de execução de todas as tarefas T_i em qualquer processador P_j.

Passo 1: Inicialização

- Todos os processadores são classificados como **livres** e todas as tarefas como **não-executáveis;**
- **temp** $\rightarrow 0$,
- Para todos os processadores P_j faça **livre**$_{P_j} \rightarrow 0$;
- Cálculos Iniciais: referem-se ao cálculo de possíveis parâmetros necessários à utilização dos critérios de escolha do processador a ser alocado às tarefas executáveis.

Início da Iteração 1 - Enquanto existirem tarefas **não-executáveis** ou **executáveis** faça:

Passo 2

Dentre as tarefas **não-executáveis** determine aquelas que podem se tornar **executáveis.**

Passo 3

Início da Iteração 2 - Enquanto existirem tarefas **executáveis** e processadores **livres** faça:

Passo 3.1: Algoritmo de Escolha - este algoritmo determina o par (T_{atual},P_{atual}) que indica o processador P_{atual} que é alocado à tarefa T_{atual}. Dependendo deste algoritmo (alocador), este par pode ser nulo, caso em que nenhuma alocação foi feita.

Passo 3.2: Se o par (T_{atual},P_{atual}) não for nulo faça:

1) $A(T_{atual}) = P_{atual}$;
2) $s(T_{atual})$ = **temp**;
3) $f(T_{atual}) = s(T_{atual}) + \mu(T_{atual},P_{atual})$;
4) livre$_{P_{atual}} = f(T_{atual})$;
5) T_{atual} é classificada como **em execução** e P_{atual} como **ocupado.**

Fim da Iteração 2

Passo 4

- Atualize o relógio de simulação determinando dentre as tarefas **em execução** a(s) próxima(s) tarefa(s) a terminarem a execução através da equação : **temp** $= \min_k\{f(T_k) | T_k$ é uma tarefa **em execução**$\}$.
- Todas as tarefas T_i que terminam em **temp** ($f(T_i)$ = **temp**) devem ser classificadas como **executadas.**
- Todos os processadores P_j que ficam livres em **temp** (livre$_{P_j}$ = **temp**) devem ser classificados como **livres.**

Fim da Iteração 1

Passo 5

- Faça **temp** $= f(T_{final})$, onde T_{final} é a última tarefa do grafo G da aplicação. A variável **temp** mede agora o tempo de término estimado de execução da aplicação realizada sob uma alocação A.

FIGURA 1 - Estrutura do Escalonador Básico

3.3. Especificação de Critérios para Algoritmos Estáticos

Neste item são apresentadas algumas maneiras de se implementar o **Passo 3** e mais especificamente o **Passo 3.1**, descrito superficialmente na Estrutura Geral do Escalonador Estático (vide Figura 1).

Como já foi mencionado, o **Passo 3.1** é o responsável pela escolha de um processador a ser alocado a cada uma das tarefas então executáveis. Esta escolha baseia-se na utilização de heurísticas que buscam minimizar o tempo de execução da aplicação considerada e caracteriza o algoritmo.

3.3.1. Algoritmos de Eficiência

Os algoritmos aqui apresentados utilizam informações sobre a eficiência de cada processador com relação às tarefas, para então determinar sua alocação. Este tipo de algoritmo tem como idéia base o desenvolvimento feito para estruturas homogêneas em [15].

O melhor tempo para a t-ésima tarefa, denotado por **b(t)**, é o menor dos m valores existentes para cada uma das tarefas ($b(t) = \min_p\{\mu(t,p)\}$). A eficiência do p-ésimo processador para a t-ésima tarefa, denotada por ef(t,p) ($0 < ef(t,p) \leq 1$) é dada por $b(t)/\mu(t,p)$.

A medida de eficiência é obviamente uma medida estritamente local, no sentido de que não procura antecipar informações sobre as próximas alocações. Não são utilizadas informações globais ou de previsão que avaliam o impacto das possíveis escolhas de alocação. A única informação determinante é o tempo de execução estimado das tarefas nos processadores livres. Este critério baseia-se na simples idéia de que, minimizar o tempo de execução de cada tarefa, auxilia na minimização do tempo de execução de toda a aplicação. A topologia do grafo não exerce qualquer influência; neste sentido, o escalonamento obtido pode não determinar um resultado ótimo, em contraposição a sua extrema simplicidade de implementação e execução.

Estão descritos nas Figuras 2.1 e 2.2 o **Passo 3.1** de 2 tipos de Algoritmos de Eficiência.

Passo 3: Calcule $b(T_i)$ para as tarefas executáveis desta instância do algoritmo - $b(T_i)=\min_j \mu(T_i,P_j)$. Calcule a eficiência de cada uma das tarefas executáveis T_i nos processadores P_{livres} - $ef(T_i,P_j)=b(T_i)/\mu(T_i,P_j)$. Construir para cada um dos processadores livres uma lista de tarefas em ordem decrescente de suas eficiências.

Passo 3.1: Enquanto existirem processadores livres e tarefas executáveis (ativas e não-alocadas) faça:

1) Atualize a lista dos processadores livres, retirando possíveis tarefas já alocadas.
2) Encontre dentre as listas dos processadores livres P_j, a tarefa executável T_i que determine o maior valor para $ef(T_i,P_j)$. (Isto é o mesmo que comparar as eiciências das tarefas que iniciam cada uma das listas dos processadores livres).
3) Faça $A(T_i)=P_j$ para o par encontrado.
4) Siga para o Passo 3.2

Pode ser notado que este algoritmo pode ser aplicado mesmo que os tempos necessários estimados não sejam conhecidos anteriormente, desde que as eficiências sejam conhecidas.

FIGURA 2.1 - Algoritmo 1 de Eficiência

Passo 3: A construção das listas é idêntica a do **Passo 3** do Algoritmo 1, exceto que se $ef(T_i,P_j)=ef(T_k,P_j)$, então T_i será ordenada antes de T_k se $\mu(T_i,P_j) \geq \mu(T_k,P_j)$.
O **Passo 3.1.** permanece igual do Algoritmo 1 de Eficiência.

FIGURA 2.2 - Algoritmo 2 de Eficiência

3.3.2. Algoritmo de Tempo Mínimo de Término

Este algoritmo procura escolher dentre o conjunto completo de processadores aquele que proporcionará à cada tarefa o menor tempo de término. No entanto, a alocação só é determinada quando os processadores estão livres. Depois que o par T_i e P_j com o menor tempo de término é determinado, verificamos se P_j está realmente disponível (livre) neste instante. Se P_j neste momento estiver em estado ocupado, T_i é colocada num estado de "espera". Assim ao final de cada iteração, quando se procura alocar todas as tarefas executáveis a todos os processadores livres, pode-se encontrar uma situação onde algumas tarefas executáveis não foram alocadas mesmo com a existência de processadores livres.

Esta heurística se baseia na premissa de que, ao minimizar o tempo de término de cada uma das tarefas pertencentes ao grafo de tarefas G, consegue-se minimizar o tempo de término de toda a aplicação. Baseia-se em algoritmos apresentados em [17], onde considera-se uma arquitetura paralela heterogênea e um conjunto de tarefas independentes.

Este algoritmo ainda pode ser considerado como aquele que faz uso apenas de informações meramente locais. No entanto, ao contrário dos algoritmos de eficiência, neste caso todos os processadores (mesmo ainda ocupados) são analisados para a determinação daquele que terminará a tarefa o mais cedo possível. Isto significa muitas

vezes optar por esperar, que um outro processador se torne livre, para realizar a alocação de uma determinada tarefa. No entanto, mesmo neste caso, as características topológicas do grafo ainda não são consideradas importantes no momento do escalonamento.

A implementação do **Passo 3.1** para este tipo de algoritmo está descrito na Figura 3.

Passo 3.1

1) Determine o conjunto de processadores livres (desocupados). Isto significa delimitar o conjunto de processadores dentre os m do sistema, que serão considerados na busca pelo par T_i e P_j com menor tempo de término.
2) Encontre o par $T_{corrente}$ e $P_{corrente}$, dentre as tarefas T_i executáveis e os P_j alocáveis (conjunto determinado acima), de tal que determina: $min_{ij}\{max_j\{temp, livre_j\} + \mu T_i, P_j)\}$.
3) Se $livrep_{corrente} > temp$ então classifique $T_{corrente}$ como não-executável, senão faça $P_{atual} = P_{corrente}$ e $T_{atual} = T_{corrente}$ (o que permite a execução do Passo 3.2).

FIGURA 3 - Algoritmo de Tempo Mínimo de Término

3.3.3. Algoritmo de Nível

O algoritmo apresentado aqui procura de maneira similar medir o peso de cada tarefa na execução do grafo de precedências. Através de um critério heurístico, procura-se "medir" as características topológicas do grafo em questão. Isto significa que uma tarefa tem prioridade de alocação em relação a outra, não apenas através de critérios meramente locais, mas em face de seu posicionamento no grafo.

O nível é uma medida inicialmente descrita na referência [14] e em outras referências, como uma maneira de medir o "tamanho" de caminhos críticos de cada tarefa de um grafo. Definimos a seguir como esta medida foi até o momento considerada:

O nível l_j de um vértice T_j de saída é igual a t_j (tempo estimado de execução da tarefa T_j). Já o nível do vértice T_i, que não é de saída, é dado pelo tamanho do maior caminho do vértice T_i ao vértice T_j de saída.
Formalmente podemos descrever como a seguir:

$$l_i = max_k \sum_j t_j \text{ ; para todos os j que pertencem a } \Pi_k,$$

onde Π_k é o k-ésimo caminho entre o nó de saída e o nó da tarefa T_i.

Como se pode observar, esta medida leva em consideração que cada nó possui apenas um valor (peso) a ele associado, isto porque a estrutura considerada é homogênea. No caso heterogêneo, cada tarefa possui um conjunto de valores de tempos estimados de execução, um para cada processador do sistema. Assim é necessário uma adaptação, para que se possa calcular o nível de cada tarefa no novo ambiente de processamento. O enfoque aqui utilizado é o de obter um valor ponderado a partir de uma análise probabilística.

Dada uma tarefa, existe uma probabilidade de que ela seja alocada a um dado processador do sistema.

Seja $Pr_{t,p}$ a probabilidade de que a tarefa t seja alocada ao processador p. Então $\Sigma_p Pr_{t,p} = 1$, ou seja a tarefa t necessariamente será alocada a um dos processadores existentes no sistema.

Dado que os valores de $Pr_{t,p}$ existem para todo t e p do sistema e aplicação, podemos então determinar um valor médio ponderado do tempo de execução estimado para cada tarefa do grafo, como: $T_t = \Sigma_p Pr_{t,p}\mu(t,p)$, onde $\mu(t,p)$ é o tempo de execução estimado da tarefa t no processador p.

É necessário determinar os valores destas probabilidades $Pr_{t,p}$. Podemos inicialmente estabelecer algumas características do tipo de distribuição que deve reger estas probabilidades de escolha de processadores para a alocação de um processador a uma dada tarefa. A distribuição é discreta e decrescente com relação ao tempo estimado de execução de qualquer tarefa. Ou seja, quanto maior o tempo estimado de execução da tarefa num dado processador, menor é a probabilidade de que este seja o processador escolhido.

Considera-se então uma distribuição geométrica do tipo: $Pr_{t,p}(n = \mu(t,p)) = CK(1-K)^n$; onde K deve também refletir esta última característica. (1-K) deve ser menor que 1 e quanto menor o seu valor, mais brusca é a queda da curva de probabilidades. Deseja-se que a queda seja mais brusca, quanto maior for a diferença entre os tempos estimados de execução de cada um dos processadores para uma dada tarefa. Podemos então estabelecer de forma coerente com esta idéia que:

$$1\text{-}K = \min_p\{\mu(t,p)\}/\max_p\{\mu(t,p)\}$$

Consequentemente, temos:

$$K = 1 - \min_p\{\mu(t,p)\}/\max_p\{\mu(t,p)\}$$

É necessário então acrescentar a constante de normalização C para que a soma das probabilidades para cada uma das tarefas seja 1.

$$\Sigma_p Pr_{t,p} = 1 \rightarrow \Sigma_p CK(1-K)^n = 1 \rightarrow C\Sigma_p K(1-K)^n = 1 \rightarrow$$
$$C = [\Sigma_p K(1-K)^n]^{-1}, \text{ onde } n = \mu(t,p)$$

Finalmente podemos escrever:

$$Pr_{t,p} = K(1-K)^n/\Sigma_p K(1-K)^n, \text{ com } n = \mu(t,p)$$
$$T_t = \Sigma_p Pr_{t,p}\mu(t,p)$$

Este último é o valor atribuído a cada nó do grafo para cálculo do nível.

Detalhadas as definições necessárias à obtenção do nível é possível descrever o Passo 3.1 do algoritmo de escolha, apresentado na Figura 4.

Passo 3.1

- Primeiramente, teríamos cálculos iniciais que poderiam ser feitos durante o primeiro passo do bloco geral descrito no item 3.2 desta seção. Estes cálculos consistem da determinação do nível das tarefas do grafo.
- Com estes valores calculados inicalmente construa uma lista de prioridades para as tarefas do grafo, na ordem decrescente do valor do nível associado à tarefa.

1) Determine os processadores livres e as tarefas executáveis
2) Dentre as tarefas executáveis escolha aquela com maior prioridade de acordo com a lista construída. Esta é a tarefa T_{atual}
3) Busque dentre os processadores livres aquele que executa a tarefa T_{atual} em menor tempo - P_{atual} é aquele que possui $\min_p\{\mu(T_{atual},p)$. tal que p esteja livre.$\}$.

FIGURA 4 - Algoritmo de Nível

4. CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma série de algoritmos de escalonamento estáticos para multiprocessadores com arquiteturas heterogêneas. Com o objetivo de fazer uma análise comparativa do desempenho destes algoritmos, estão sendo conduzidos pelos autores estudos de simulação cujos resultados preliminares mostram que o algoritmo baseado na heurística de Tempo Mínimo de Término apresenta vantagens em relação aos demais. Considerações adicionais a respeito da comparação entre as diversas heurísticas serão apresentadas em um trabalho futuro.

Referências

1) M.S. Pittler, D.M. Powers, D.L. Schanabel, "System development and technology aspects of the IBM 3081 processor complex", IBM Journal of Research and Development, Vol.26, No.1, January 1982 New York, IBM.
2) A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAullffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer - Designing a MIMD, Shared Memory Parallel Machine", IEEE Transactions on Computers, February 1983, pp.175-189.
3) Cray Research Inc., "Cray X-MP Series Mainframe Reference Manual", HR-0032, 1982.
4) W.D. Daniel Hillis, "The Connection Machine", MIT Press, 1985.
5) Kenneth C. Sevcik, "Characterization of Parallelism in Adaptation and Their Use in Scheduling", Performance Evaluation Review, Vol.17, No.1, May 1989.
6) Shikharesh Majumbar, Derek L. Eager and Richard B. Bunt, "Scheduling in Multiprogrammed Parallel Systems", ACM -1988
7) Thomas L. Casavant and Jon G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems", IEEE Transactions on Software Engineering, Vol.14, No.2, Feb. 1988
8) Mario Gonzalez Jr., "Deterministic Processor Scheduling", Computing Surveys, Vol.9, No.3, Sept. 1977.
9) John Zahorjan and Cathy McCann, "Scheduling Processors in Shared Memory Multiprocessors", Technical Report.

10) Clyde P. Kruskal and Alan Weiss, "Allocating Subtasks on Parallel Processors", IEEE Trans. on Software Engineering, Vol.SE-11, No.10, Oct. 1985.
11) Thomas L. Casavant and Jon G. Kuhl, "Effects of Response and Stability on Scheduling in Distributed Computing Systems", IEEE Trans. on Software Engineering, Vol.14, No.11, Nov. 1988.
12) Zarka Cvetanovic, "The Effects of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Systems", IEEE Transactions on Computers, Vol.C-36, No.4, April 1987.
13) Jacek Blazewicz, Mieczyslaw Drabowski and Jan Weglarz, "Scheduling Multiprocessors Tasks to Minimize Schedule Length", IEEE Transactions on Computers, Vol.C-35, No.5, May 1986.
14) Thomas L. Adam, K.M. Chandy & J.R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems", Communication of the ACM - December 1974, Vol.17, No.12.
15) Ernest Davis and Jeffrey M. Jaffe, "Algorithms for Scheduling Tasks on Unrelated Processors", Journal of the ACM - October 1981, Vol.28, No.4, pp.721-736.
16) Daniel Menascé and Virgílio Almeida, "Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures, Proceedings of the Supercomputing'90 Conference, New York, EUA, Nov. 1990.
17) Ellis Horowitz and Sartaj Sahni, "Exact and Approximate Algorithms for Scheduling Nonidentical Processors", Journal of the ACM - April 1976, Vol.23, No.2, pp.317-327.
18) G. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capability", Proc. AFIPS Spring Joint Comp. Conf. 30, 1967.
19) Hinori Kasahara and Seinosuke Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing", IEEE Transactions on Computers - November 1984, Vol.C-33, No.11.