UMA COMPARAÇÃO DA EFICIÊNCIA DE DUAS HEURÍSTICAS PARA ALOCAÇÃO ESTÁTICA DE PROCESSOS

Cristiana Bentes Seidel[1] Lysia Maria M.B. Canaley[2] Claudio Luis Amorim[3]

RESUMO

Neste trabalho apresentamos os resultados da implementação de dois algoritmos heurísticos para alocação estática de processos em arquiteturas paralelas. Comparamos a eficiência das alocações geradas pelos algoritmo segundo o tempo gasto com execução por cada processador e o volume de comunicação realizada entre diferentes processadores. Analisamos, também, o efeito do balanceamento de carga nas referidas alocações. Nos experimentos realizados apontamos falhas nas duas heurísticas e sugerimos possíveis melhoras.

ABSTRACT

In this work we describe the results obtained from the implementation of two heuristic algorithms for static task allocation in parallel architectures. We present a performance comparison of the allocations produced by the two heuristics, based on processor execution time and communicating time betwen processors. Also, we analyse the effects of the load balancing on each allocation. The results indicated faults in both heuristcs and we suggest some improvements.

[1]MSc (COPPE-1991); Aluna de Doutorado COPPE-Sistemas (UFRJ); Professora Assistente da UERJ; áreas de interesse: escalonamento, arquiteturas paralelas e sistemas operacionais.

[2]MSc (COPPE-1993); áreas de interesse: escalonamento e arquiteturas paralelas.

[3]MSc (COPPE-1979), PhD (Imperial College - 1984); Professor Adjunto da COPPE-Sistemas (UFRJ); áreas de interesse: supercomputação e processamento paralelo. email: amorim@rio.cos.ufrj.br.

1. INTRODUÇÃO

Em resposta à crescente necessidade de maior desempenho computacional exigido pelas aplicações científicas, uma variedade de arquiteturas paralelas têm sido desenvolvidas. Em tais arquiteturas, diferentes partes da aplicação (chamadas processos) podem ser atribuídas a elementos de processamento para execução paralela, reduzindo substancialmente o seu tempo de execução. Desse modo, esperamos aumentar ainda mais o desempenho da aplicação à medida que aumentamos o número de elementos de processamento (processadores) do sistema. Não podemos, contudo, desconsiderar o *overhead* causado pela comunicação entre os processos paralelos (imposto por protocolos de comunicação, tempo de espera em filas, congestionamento no acesso a memória, etc...). Na verdade, a performance de um sistema paralelo está relacionada a dois objetivos conflitantes: maximizar o paralelismo (aumentar o número de processos que executam em paralelo) e minimizar a comunicação (diminuir a quantidade de mensagens trocadas entre os processos paralelos).

A forma como os processos são alocados aos diversos processadores do sistema tem, assim, peso fundamental na execução de um aplicação, uma vez que ela definirá tanto o grau de paralelismo como a quantidade de comunicação realizada interprocessador.

A alocação de processos aos processadores do sistema pode ser realizada dinamicamente durante a execução, ou estaticamente em tempo de carregamento nos processadores. A alocação dinâmica é especialmente útil quando não se pode determinar o comportamento dos processos antes de sua execução. Entretanto, a redistribuição dos processos pode se tornar impraticável se considerarmos uma arquitetura heterogênea fracamente acoplada devido ao alto overhead gerado durante a execução e a dificuldade de se transportar processos entre processadores heterogêneos com linguagem de máquina diferentes.

Abordaremos neste trabalho o problema da alocação estática dos processos de uma aplicação, onde a partir de informações passadas pelo usuário ou pelo compilador é possível atribuir processos a processadores antes do período de execução da aplicação.
Nosso modelo foi desenvolvido para arquiteturas fortemente ou fracamente acopladas, sendo que é mais facilmente adaptado para sistemas distribuídos.

Chegar a uma alocação ótima (que maximiza o desempenho) é, conhecidamente, um problema intratável polinomialmente — NP-completo [BOK81]. O que encontramos na literatura são algoritmos heurísticos que resolvem o problema de forma subótima com resultados bastante satisfatórios de acordo com certos critérios de desempenho previamente estabelecidos. Os critérios utilizados por cada algoritmo são os mais diversos, i.e., os algoritmos diferem principalmente em relação ao objetivo a ser alcançado (minimizar a comunicação interprocessador, manter um bom balanceamento de carga, minimizar o tempo de execução da aplicação, etc...). Alguns desses objetivos são conflitantes, impossibilitando a comparação de um algoritmo com outro. Por isso não se chegou, até hoje, a um consenso de qual a melhor heurítica para solucionar o problema da alocação estática.

Este trabalho descreve os resultados da implementação de dois algoritmos heurísticos que chamaremos de Algoritmo de Cortes Sucessivos e Algoritmo de Agrupamento Hierárquico descritos respectivamente em [LO88] e [BOW92]. Nosso objetivo foi comparar o desempenho das alocações geradas pelos algoritmos segundo o tempo gasto com execução em cada processador e o custo da comunicação interprocessador.

O algoritmo de Agrupamento Hierárquico foi desenvolvido para minimizar o custo de comunicação, mantendo a carga de trabalho bem balanceada no sistema. Já o Algoritmo de Cortes

Sucessivos tenta minimizar o *turnaround* de uma aplicação. Este é apresentado em duas versões: **sem interferência** e **com interferência**. Estas versões diferem no que diz respeito ao balanceamento da carga, somente a segunda se preocupa em distribuir os processos.

Nossos testes foram realizados de forma a verificar como o balanceamento da carga influencia o desempenho das aplicações que serão executadas na arquitetura paralela.

Na próxima Seção faremos um breve resumo da literatura existente. Na Seção 3 apresentamos uma descrição formal do problema. As Seções 4 e 5 explicam, brevemente, o funcionamento dos algoritmos que foram implementados. Na Seção 6 descrevemos os experimentos realizados e analisamos seus resultados. Finalmente, na Seção 7 concluímos o trabalho, apontando possíveis evoluções futuras.

2. TRABALHOS RELACIONADOS

Na tentativa de obter uma solução ótima para o problema de alocação estática de processos, muitos autores restringiram o modelo da aplicação e da máquina paralela. Stone [STO77], por exemplo, chegou a solução ótima para máquinas com somente dois processadores. Já Bokhari em [BOK81] apresenta um algoritmo que mapeia aplicações em uma matriz NxN de processadores conectados sob a forma de vizinhos próximos e em [BOK88] ele examina somente aplicações em forma de cadeia.

Soluções restritas são importantes quando se quer atacar uma determinada classe de problemas em arquiteturas paralelas dedicadas. Arquiteturas de propósito geral, por outro lado, devem ser capazes de resolver, eficientemente, uma gama muito maior de aplicações.

A alocação dinâmica foi estudada principalmente para arquiteturas que possuem memória compartilhada onde a migração de processos entre processadores é menos custosa. Squillance e Nelson [SQU91] examinaram o efeito da migração de processos no balanceamento de carga de sistemas fortemente acoplados. Na mesma classe de máquinas, Gupta *et al* [GUP91] analisaram diversas políticas de escalonamento e Markatos e LeBlanc [MAR91] compararam os efeitos do balanceamento dinâmico de carga (distribuição de processos) contra o gerenciamento da localidade (manter os processos perto de seus dados), eles mostraram que sendo as duas atividades conflitantes, em geral, é preferível manter a localidade.

A alocação estática é tratada em outros diversos trabalhos. O trabalho de Lee e Aggarwal [LEE87] propõe uma maneira mais cuidadosa de quantificar o overhead de comunicação entre processos paralelos. Chu *et al* [CHU88] mostram como estimar o custo de comunicação entre processos, seja a comunicação realizada por um *link* entre processadores ou por uma memória compartilhada.

Sobre os algoritmos heurísticos para alocação estática temos basicamente duas classes, aqueles que consideram uma ordem de precedência entre os processos e os que não a consideram. Dentre os trabalhos que levam em consideração a precedência, podemos citar o de Shirazi e Wang [SHI90] baseado em métodos de lista de prioridade, onde num procedimento iterativo, cada processo é atribuído a um processador. São utilizadas diferentes heurísticas para determinar a prioriade de atribuição e é verificado que as heurísticas mais simples apresentam um melhor resultado global. O trabalho de Kim [KIM88] também considera precedência entre processos sendo que o algoritmo é mais complexo, ele realiza diversos níveis de agrupamentos lineares de caminhos críticos de processos, os grupos formados são atribuídos aos processadores.

Quanto aos algoritmos que não usam a precedência entre as tarefas temos os trabalhos de Lo [LO88] e Bowen *et al* [BOW92]. Estes são os algoritmos utilizados em nossa implementação e suas descrições detalhadas encontram-se nas Seções 4 e 5 respectivamente.

3. O MODELO

Estamos considerando um modelo único de aplicação e de arquitetura a ser utilizado em ambos os algoritmos.

Uma aplicação é especificada por um conjunto de processos. Supomos que a aplicação já se encontra adequadamente particionada em processos (o particionamento da aplicação constitui um problema complexo por si só, merecendo tratamento independente da alocação). A representação da aplicação é feita por um grafo não direcionado $G_a = (N_a, E_a)$, onde $N_a = \{p_1, p_2, ..., p_n\}$ é o conjunto de processos que a compõem e E_a é o conjunto de arestas de G_a. A cada aresta e_k está associado um valor c_{ij} que representa o custo de comunicação entre os processos p_i e p_j. Todos os custos c_{ij} estão representados em uma matriz C (nxn) em que $c_{ij} = 0$ se p_i não se comunica com p_j.

A arquitetura paralela utilizada em nossos testes é composta de um conjunto de processadores (o número de processadores pode ser 2, 4, 8 ou 16) totalmente conectados. Essa arquitetura é modelada por um grafo não direcionado $G_p=(N_p,E_p)$, onde $N_p = \{P_1, P_2, ..., P_m\}$ é o conjunto de processadores do sistema e E_p o conjunto de arestas de G_p, para cada par (P_i,P_j) existe uma aresta ligando P_i a P_j. Uma matriz X (nxm) armazena os valores x_{ij} que representam o tempo de execução de p_i em P_j.

O problema da alocação estática consiste em mapear os nós do grafo G_a nos nós do grafo G_p de modo a minimizar tanto a comunicação interprocessador como o tempo total de execução da aplicação.

O modelo citado acima se adapta melhor a sistemas fracamente acoplados. Para suportar também arquiteturas com memória compartilhada, os custos da matriz C devem ter uma parcela dinâmica representando os problemas de contenção de memória e dos mecanismos de sincronização.

4. O ALGORITMO DE CORTES SUCESSIVOS

Este algoritmo constitui-se de três passos básicos chamados de GRAB, LUMP e GREEDY e executados nesta ordem. Cada passo tenta fazer uma alocação total (i.e. alocar todos os processos de G_a), caso não consiga o passo seguinte é ativado.

A heurística utilizada pela autora baseia-se no modelo descrito por Stone [STO77] onde não há esforço direto no sentido de obter paralelismo. O resultado são alocações pouco balanceadas. O modelo de Stone foi, então, estendido para incluir um fator adicional, chamado interferência. O custo de interferência reflete o grau de incompatibilidade entre processos quando alocados a um mesmo processador. Conceitualmente, a interferência se refere à competição de processos pelos recursos do processador ao qual estão alocados.

Implementamos as duas versões do algoritmo, sem interferência e com interferência. Para ambas as versões os passos do algoritmo têm funcionamento idêntico, sendo que para o caso de se considerar a interferência, o peso de uma aresta é dado pelo custo de comunicação menos o custo de interferência.

GRAB

Esse passo pretende alocar processos que têm forte preferência por processadores específicos (por exemplo, um processo tem preferência por um processador que o execute muito mais rápido do que os outros). Para tal é utilizado um algoritmo de Fluxo Máximo/Corte Mínimo que realiza alocações ótimas para dois processadores. Considera-se o grafo de processadores como tendo somente dois nós P_i e Q_i de modo que Q_i seja um supernó que representa todos os processadores diferentes de P_i. Para este grafo é aplicado o algoritmo de Fluxo Máximo/Corte Mínimo e dessa forma são feitas alocações a P_i.

Esse processo é realizado para todos os processadores do sistema, se a atribuição realizada for completa, ela é ótima.

LUMP

Esse passo tenta realizar uma alocação bastante simples: atribuir todos os processos que não foram alocados pelo GRAB a um único processador. Antes de realizar esta alocação o LUMP compara seu custo com um limite inferior do custo de distribuir os módulos restantes entre os processadores. Se o custo da alocação for menor que esse limite inferior, então a alocação é realizada e uma atribuição completa é obtida. Caso contrário, o LUMP não faz nenhuma alocação.

GREEDY

O passo final é um algoritmo do tipo guloso que localiza processos entre os quais os custos de comunicação são acima da média e os coloca em grupos. Processos em um mesmo grupo são alocados conjuntamente ao processador que minimiza a soma dos tempos de execução de todos os processos contidos no grupo.

5. O ALGORITMO DE AGRUPAMENTO HIERÁRQUICO

Este algoritmo é composto de duas fases distintas: a Fase de Agrupamento e a Fase de Alocação. A primeira Fase realiza agrupamentos tanto no grafo de processos como no grafo de processadores. No grafo de processos, são reunidos em grupos processos que mais se comunicam. Os grupos de processadores são formados por processadores que tenham determinada "afinidade", por exemplo, estão perto topologicamente, são idênticos, etc... Ao final dessa Fase são geradas duas árvores uma de processos outra de processadores.

A segunda Fase é reponsável por atribuir os nós da árvore de grupos de processos aos nós da árvore de grupos de processadores. Essa alocação é feita visando minimizar o custo de comunicação e mantendo determinadas cargas de trabalho em cada processador.

FASE DE AGRUPAMENTO

Nessa Fase são realizados vários níveis de agrupamentos. Para cada nível, escolhe-se um nó do grafo como pivô, ele será o nó central de um grupo. O pivô é o nó adjacente à aresta de maior peso e como dois nós têm essa característica, é selecionado o nó que possui o maior número de arestas. Juntamente com o pivô são agrupados seus vizinhos (nós diretamente ligados ao pivô) cujos pesos das arestas são aproximadamente iguais (o que é considerado aproximadamente igual depende de um valor limiar passado como parâmetro). A inclusão de vizinhos dos vizinhos do pivô, ou de vizinhos de vizinhos dos vizinhos do pivô e assim por diante, no mesmo grupo dependerá de um parâmetro k pre-determinado (em nossos testes consideramos k=2, o que significa que foram incluídos no máximo os vizinhos dos vizinhos do pivô no mesmo grupo). Sempre que restarem nós no grafo que não foram agrupados, escolhe-se um outro pivô e o mesmo processo é repetido, até que todos os nós do grafo façam parte de um algum grupo.

No próximo nível, o grafo considerado contém apenas os nós pivôs selecionados no nível anterior e é realizado o mesmo processo de agrupamento do nível anterior. São efetuados, então, outros diversos níveis de agrupamento, até que no último nível é selecionado um único pivô em todo o grafo.

A árvore de grupos gerada é tal que o nó raiz corresponde ao grupo formado no último nível de agrupamento, seus filhos os grupos formados no penúltimo nível e assim por diante, até que nas folhas da árvore encontram-se individualmente cada nó do grafo.

Na Figura 1 apresentamos um exemplo do funcionamento da Fase de Agrupamento e a respectiva árvore de grupos gerada.

FASE DE ALOCAÇÃO

A segunda Fase do Algoritmo de Agrupamento Hierárquico constitui-se do mapeamento da árvore de processos na árvore de processadores (geradas na Fase anterior). Para cada processador do sistema são atribuídos dois parâmetros que representam a carga mínima e
máxima (em termos de número de processos) que deve ser alocada a ele, esses parâmetros são chamados respectivamente de **L** e **U.** A alocação realizada deve atingir a carga mínima de cada processador para garantir o balanceamento dos processos do sistema e não se pode exceder a carga máxima para evitar problemas de contenção ou de um processador sobrecarregado.

Antes de iniciar essa fase é testado se o número de processos total está entre os limites mínimo e máximo total da carga de todos os processadores, caso contrário, a alocação não poderá ser realizada.

O mapeamento da árvore de processos na árvore de processadores é então realizado da seguinte forma: sejam r e R respectivamente os nós raízes da árvore de processos e da árvore de processadores, a árvore de processos é alterada de tal forma que r tenha o mesmo número de filhos que R. Se R é uma folha, e portanto representa um único processador, a alocação é realizada de modo que todos os filhos de r sejam alocados a esse processador. Caso contrário o algoritmo é chamado recursivamente para cada subárvore formada a partir dos filhos de r e R.

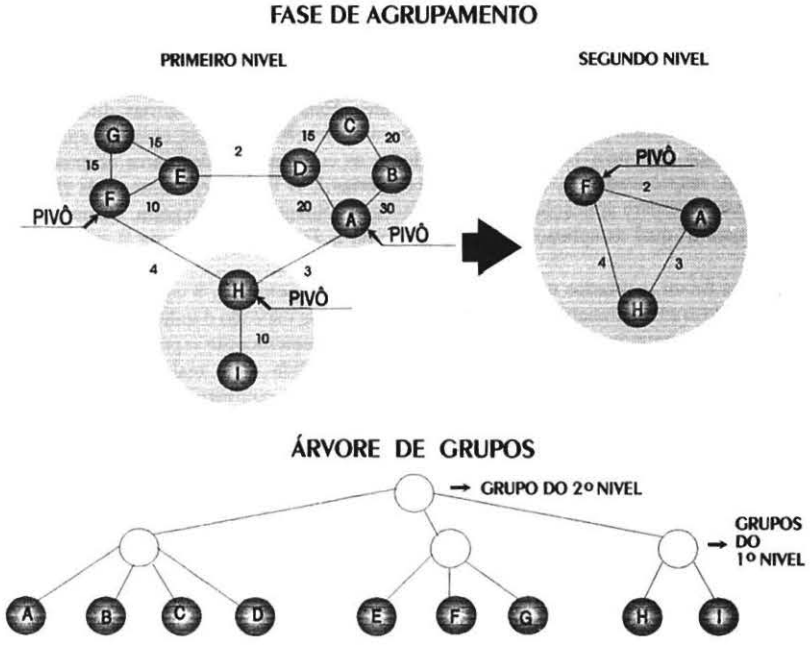

Figura 1: Primeira Fase do Algoritmo de Agrupamento Hierárquico

6 - EXPERIMENTOS E RESULTADOS

Nas experiências realizadas com o Algoritmo de Cortes Sucessivos (A.C.S.) e com o Algoritmo de Agrupamento Hierárquico (A.A.H.) procuramos utilizar os mesmos grafos e parâmetros de entrada de modo que fosse possível a comparação entre eles. Consideramos que um conjunto de quatro aplicações independentes executam simultaneamente em uma arquitetura paralela. Investigamos as alocações realizadas pelos algoritmos em relação ao tempo de execução das aplicações e o custo de comunicação interprocessador. Verificamos, também, a distribuição da carga realizada por cada algoritmo e como esta distribuição afeta o desempenho da aplicação

A seguir, detalhamos a composição dos dados de entrada. Discutimos, então, os experimentos conduzidos, apresentando os resultados em forma gráfica, juntamente com suas interpretações.

6.1 - DADOS DE ENTRADA

O grafo G_a utilizado em nossos testes é composto de quatro subgrafos desconexos S_1, S_2, S_3 e S_4 (cada subgrafo representa uma aplicação), i.e., $c_{ij} = 0$ para i e j pertencentes a subgrafos diferentes. Cada aplicação é formada por 10 processos concorrentes. Existem três categorias diferentes de aplicações que se caracterizam pela razão de arestas de comunicação não nulas. Criamos aplicações com comunicação não nula para 1/2, 1/3 e 1/6 de todas as combinações de pares de processos, constituindo assim, respectivamente, as chamadas categorias I, II e III. Realizamos três tipos de testes, em cada teste as quatro aplicações de G_a pertencem a somente uma categoria. Chamamos de *Teste 1* o grafo para o qual as aplicações são da categoria I, *Teste 2* da categoria II e *Teste 3* da categoria III.

Os mapeamentos foram realizados em uma arquitetura paralela com 2, 4, 8 ou 16 processadores. Os processadores são completamente interconectados e a forma de conexão é indiferente (por canais de comunicação ou por memória compartilhada), os custos de comunicação representam o volume de comunicação realizada entre dois processadores (uma medida mais real deveria levar em consideração a bandapassante dos canais ou a contenção na memória compartilhada).

Consideramos que todos os processos apresentam o mesmo tempo de execução nos diferentes processadores e normalizamos este tempo de execução para 1, i.e., $x_{ij} = 1$ para todo i e j. Esta restrição é necessária para o correto funcionamento do Algoritmo de Agrupamento Hierárquico.

Com relação ao custo de comunicação, c_{ij}, entre processos, criamos três classes diferentes de matrizes. Em todas as três classes geramos os valores não nulos de c_{ij} aleatoriamente dentro de um determinado intervalo (a quantidade de valores não nulos respeita uma determinada categoria). Na primeira classe, os elementos não nulos estão no intervalo [1,1], ou seja, quando há comunicação entre dois processos, ela é comparável ao tempo de execução do mesmo, dizemos assim que a razão **r** entre o custo de execução e o custo de comunicação é 1/1. Na segunda classe de matriz c_{ij} pertence a [1,3], o custo médio de comunicação é 2 e **r**=1/2. Para a terceira classe, estamos supondo que a comunicação tem peso exageradamente maior do que a execução, c_{ij} pertence a [1,20], o custo médio de comunicação é 10 e **r** = 1/10.

Para a versão com interferência do Algoritmo de Cortes Sucessivos geramos uma matriz I (nxn) onde i_{ij} representa o custo de interferência entre os processos i e j. Com o objetivo de tentar distribuir os processos e as aplicações geramos custos de interferência no mesmo intervalo que os custos de comunicação. Consideramos que a maior interferência está entre processos de aplicações diferentes.

O Algoritmo de Agrupamento Hierárquico requer como parâmetros o número mínimo, **L**, e máximo, **U**, de processos que cada processador suporta, sendo que ele espera que **L** 1. Numa primeira série de experimentos aplicamos os valores **L**=1 e **U**=40 para testar as alocações sem restrições de carga de trabalho. Numa segunda série, diminuímos o valor de **U** para 6.

6.2 - AVALIAÇÃO DOS RESULTADOS

A versão sem interferência do A.C.S. realizou, para todos os testes, a mesma alocação: todas as quatro aplicações foram atribuídas a um único processador do sistema. Obviamente, o custo de comunicação interprocessador encontrado foi mínimo e igual a zero, mas para isso estamos pagando o preço de perder todo o paralelismo da arquitetura. Essa alocação é resultado da heurística

utilizada, que só tenta fazer uma distribuição de processos se um processador executar um determinado processo mais rapidamente. Consideramos este resultado totalmente insatisfatório pela ausência de paralelismo. Nossas comparações serão feitas, portanto, entre o A.A.H e a versão com interferência do A.C.S.

Quando a arquitetura apresenta 2 ou 4 processadores, ambos os algoritmos realizaram a mesma alocação, que por sinal é a ótima. No caso de 2 processadores, duas aplicações foram alocadas em um processador e as outras duas no segundo processador. Para 4 processadores foi atribuída uma aplicação por processador. Esses dois casos não são considerados de maior interesse pois devido ao reduzido número de processadores não seria viável distribuir os processos de uma aplicação.

Estudaremos as alocações realizadas em arquiteturas com 8 e 16 processadores. Nessas arquiteturas podemos analisar o efeito de paralelizar a aplicação no desempenho do sistema.

Um sistema que comporta quatro aplicações em execução simultânea não deve se preocupar somente com o desempenho de cada aplicação isolada, mas com o *throughput* do sistema. O A.A.H espera obter bons resultados para o *throughput* com o balanceamento da carga entre os processadores. O A.C.S. minimiza o tempo de execução de cada tarefa, o que pode também levar a um bom *throughput.*

Em todos os testes realizados, O A.C.S. aloca uma aplicação por processador, deixando os processadores restantes ociosos. O A.A.H. distribui os processos das aplicações pelos processadores do sistema.

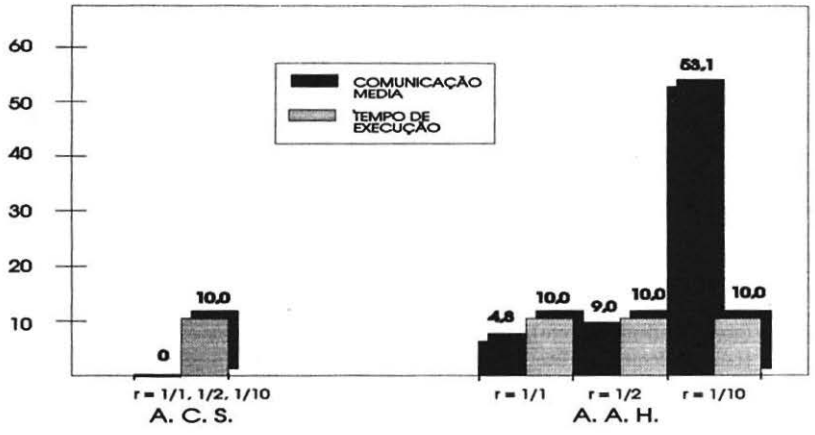

Figura 2: *Teste1* para 8 processadores

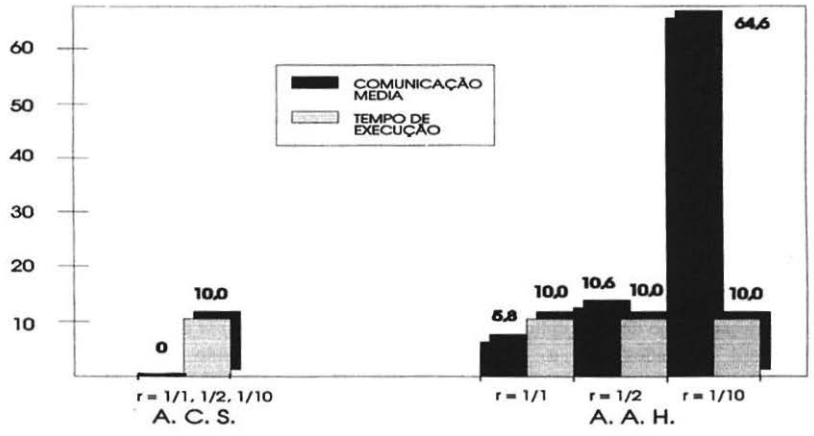

Figura 3: *Teste 1* para 8 processadores

Nas Figuras 2 e 3 apresentamos os resultados para o *Teste 1* com 8 e 16 processadores, nas Figuras 4 e 5, para o *Teste 2* com 8 e 16 processadores e nas Figuras 6 e 7, para o *Teste 3* com 8 e 16 processadores. Estamos comparando as alocações com relação ao tempo de execução e a comunicação média interprocessador. O tempo de execução exposto representa o maior tempo de execução de todos os processadores. A comunicação interprocessador é o volume médio de comunicação realizado por um processador do sistema.

Pelos gráficos apresentados podemos notar que para o *Teste 1*, o A.A.H obteve resultados bastante inferiores ao A.C.S. (obtém o mesmo tempo de execução com um custo de comunicação elevado). Tanto para 8 como para 16 processadores, a utilização do A.A.H. só é viável se o custo de comunicação entre processos for muito menor do que o custo da execução dos mesmos nos processadores.

No *Teste 2* o A.C.S. continua tendo melhores resultados que o A.A.H., sendo que o custo de comunicação apresentou uma ligeira redução.

O *Teste 3* mostra que o A.A.H. consegue reduzir o tempo de execução com um custo baixíssimo de comunicação (aproximademente zero), suas alocações são melhores do que as do A.C.S.

Por esses três testes podemos verificar que o A.C.S., no passo Greedy, encontra quatro grupos distintos de processos e realiza a distribuição desses grupos, mas por hipótese alguma ele distribui processos dentro de um grupo. Embora tenha apresentado melhores resultados nos *Testes 1* e *2*, sua heurística explora somente o paralelismo entre as aplicações, tornando-as puramente sequenciais.

Já o A.A.H. não obtém bons resultados tentando distribuir processos dos *Testes 1* e *2*. Nesses testes, processos de uma aplicação apresentam elevado grau de comunicação, o que leva a heurística a falhar. No *Teste 3* as aplicações são menos "densas" em termos de comunicação e, portanto, o balanceamento da carga é bastante recomendável.

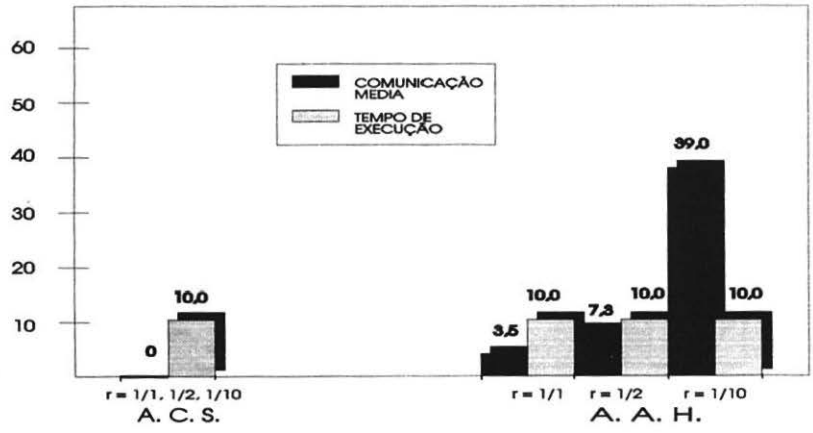

Figura 4: *Teste 2* para 8 processadores

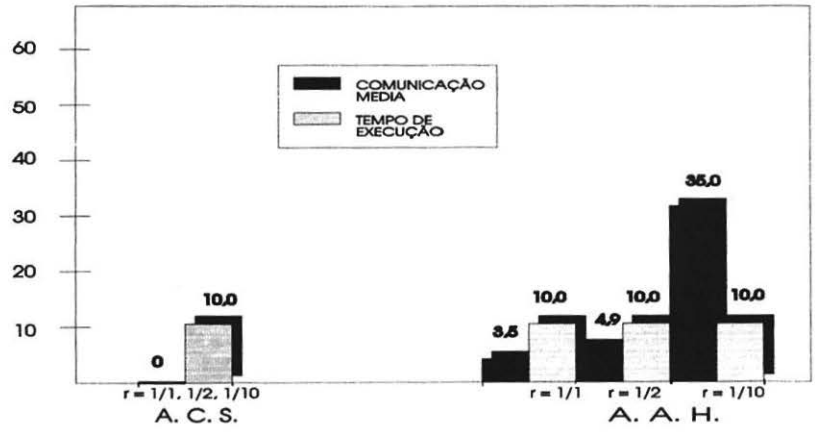

Figura 5: *Teste 2* para 16 processadores

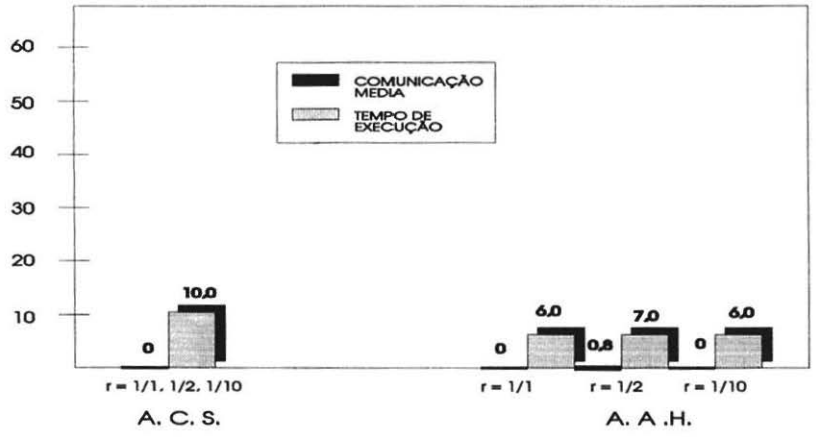

Figura 6: *Teste 3* para 8 processadores

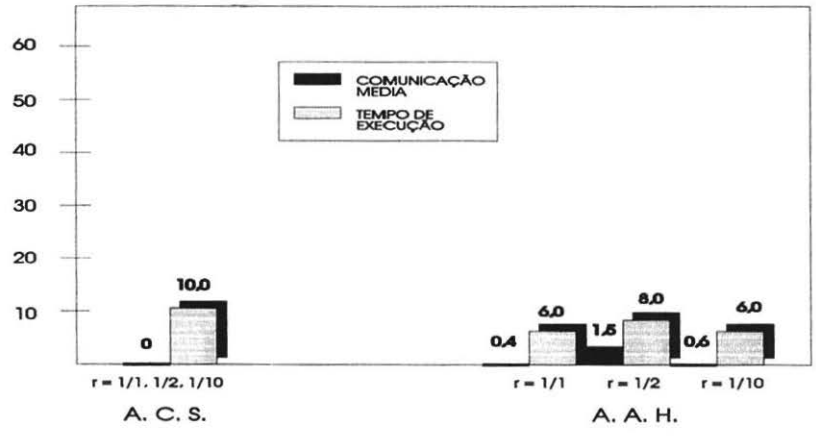

Figura 7: *Teste 3* para 16 processadores

Nesses primeiros experimentos, ressaltou-nos o fato de que nos *Testes 1* e *2* o A.A.H. alocava aos primeiros processadores aplicações inteiras e depois distribuía somente uma ou duas aplicações pelos processadores restantes, levando à distribuição de processos com alta carga de comunicação à processadores diferentes. Realizamos, então, um outro experimento em que acertamos o valor de **U** com 6. Dessa forma, o algoritmo alocaria no máximo 6 processos por processador. Esse experimento visa testar se distribuindo processos de todas as aplicações o algoritmo apresenta melhores resultados.

Nas Figuras 8 e 9 apresentamos os resultados dos *Testes 1*, *2* e *3* no A.A.H. com **U**=6. Os resultados para r=1/10 não foram descritos devido ao custo de comunicação gerado ser muito elevado (da ordem de 100 a 200), não cabendo na figura. De qualquer forma, os Testes1 e 2 levaram a alocações ainda piores que o experimento passado.

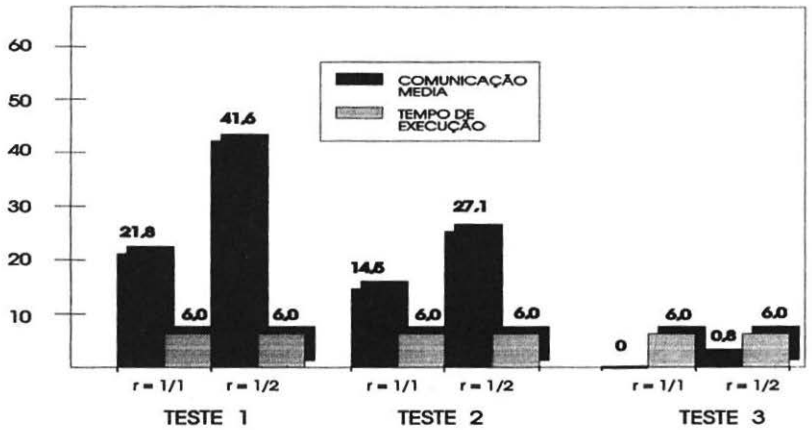

Figura 8: A.A.H para 8 processadores e U = 6

A explicação para esse fato está na heurística do A.A.H. O mapeamento da árvore de processos na árvore de processadores é feito de forma a atribuir nós de uma árvore em nós da outra. Quando o nó atribuído da árvore de processos representa um grupo, há uma boa alocação (a alocação de um grupo de processos em um processador mantém processos que mais se comunicam juntos). Contudo, para atingir os limites **L** e **U** dos processadores, o algoritmo pode precisar realizar atribuições com as folhas da árvore de processos, nesse caso estaremos alocando processos de um mesmo grupo a processadores diferentes, o que o leva a péssimos resultados.

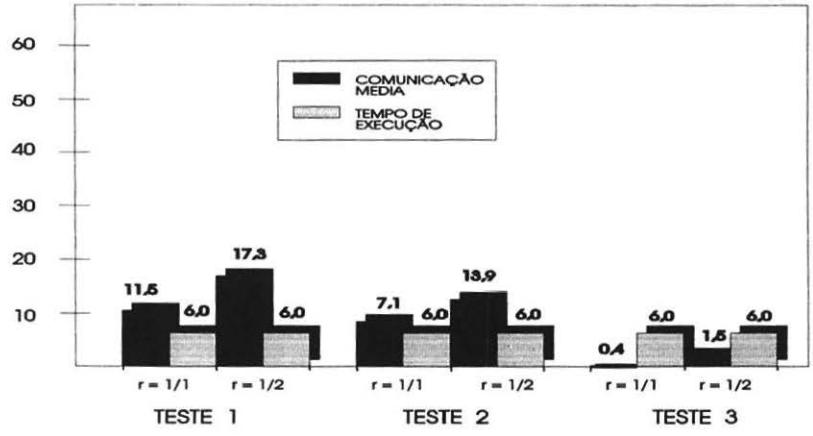

Figura 9: A.A.H. para 16 processadores e U = 6

7. CONCLUSÕES

Neste trabalho implementamos dois algoritmos heurísticos para alocação estática (o Algoritmo de Cortes Sucessivos descrito em [LO88] e o Algoritmo de Agrupamento Hierárquico descrito em [BOW92]) e testamos seus resultados comparativamente. Nosso objetivo foi estudar o desempenho das alocações realizadas por cada algoritmo, em termos de tempo de execução e custo de comunicação, com relação ao balanceamento da carga do sistema.

Diante dos experimentos realizados pudemos notar que a heurística do Algoritmo de Cortes Sucessivos, mesmo considerando a interferência entre processos, não explora o paralelismo da arquitetura. Já o Algoritmo de Agrupamento Hierárquico tenta distribuir os processos da aplicação a todo custo. Quando a aplicação apresenta processos com pequeno grau de comunicação essa distribuição mostrou-se bastante vantajosa. Nos outros casos ela foi muito ruim.

Analisando mais profundamente a heurística do A.A.H. verificamos que uma mudança na Fase de Alocação pode levar a atribuições mais eficientes. Essa mudança deveria levar em consideração que os nós na folha da árvore só podem ser atribuídos a processadores diferentes caso não tenham o mesmo pai (i.e., não sejam do mesmo grupo).

Pelos resultados obtidos, a classe de heurísticas representada pelo Algoritmo de Agrupamento Hierárquico é a que oferece melhores estratégias para o problema da alocação estática de processos. Pretendemos continuar a realizar testes com outros tipos de grafos e outras heurísticas.

REFERÊNCIAS

[BOK81] Bokhari, S.H., "On the Mapping Problem", *IEEE Transactions On Computers*, Vol C-30, March 1981, pp 207-214.

[BOK88] Bokhari, S.H., "Partioning Problems in Parallel, Pipelined and Distributed Computing", *IEEE Transactions on Computers*, Vol 37, January 1988, pp 48-57.

[BOW92] Bowen, N.S., Nikolau, C.N., Ghafoor, A., "On The Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems", *IEEE Transactions on Computers*, Vol 41, March 1992, pp 257-273.

[CAN93] Canaley, L.M.M.B, *Alocação de Tarefas em Sistemas Distribuídos*, Tese de Mestrado COPPE-UFRJ, Abril 1993.

[CHU88] Chu, W.W., Holloway, L.J., Lan, M.T., Efe, K., "Task Allocation in Distributed Data Processing", *IEEE Transactions on Computers*, Vol C-30, November 1988, pp57-70.

[GUP91] Gupta, A., Tucker, A. and Urushibara, S., "The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications", *Proceedings of the 1991 ACM SIGMETRICS Conference On Measurements and Modeling of Computer Systems*, May 1991, San Diego, CA, pp143-155.

[KIM88] Kim S.J., *A General Approach to Multiprocessor Scheduling*, PhD. Thesis, The University of Texas at Austin, December 1988.

[LEE87] Lee,S-Y and Aggarwal, J.K., "A Mapping Strategy for Parallel Processing", *IEEE Transactions on Computers*, Vol C-36, April 1987, pp 433-442.

[LO88] Lo, V.M., "Heuristic Algorithms for Task Assignment in Distributed Systems", *IEEE Transactions on Computers*, Vol 37, November 1988, pp 1384-1397.

[MAR91] Markatos, E.P. and LeBlanc, T.J., "Load balancing vs. Locality Management in Shared Memory Multiprocessors", *Technical Report 399*, University of Rochester, October 1991.

[SHI90] Shirazi, B. and Wang,M.,"Analysis and Evaluation of Heuristic Methods for Static Scheduling", *Journal of Parallel and Distributed Computing*, Vol 10, 1990, pp 222-232.

[SQU91] Squillance,M.S. and Nelson, R.D., "Analysis of Task Migration in Shared Memory Multiprocessor Scheduling", *Proceedings of the 1991 ACM SIGMETRICS Conference On Measurements and Modeling of Computer Systems*, May 1991, San Diego, CA, pp 143-155.

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