Algoritmos Paralelos de Gerência e Alocação de Processadores em Máquinas Multiprocessadoras Hipercúbicas

César A. F. De Rose[1]
Philippe O. A. Navaux[2]

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Pós-Graduação em Ciência da Computação
Caixa Postal 15064
CEP 91501-970 – Porto Alegre, RS, Brasil
Tel.: (051)336-8399 Fax: (051)336-5576

Resumo

O desempenho obtido por uma máquina maciçamente paralela depende em grande parte da eficácia e eficiência do algoritmo de gerência e alocação de processadores utilizado. Este trabalho analisa a possibilidade de se melhorar o desempenho dos algoritmos de gerência e alocação de processadores em máquinas multiprocessadoras hipercúbicas através de sua paralelização. São propostas versões paralelas dos principais algoritmos encontrados na literatura e apresentados os resultados obtidos.

Palavras-chave: processamento paralelo, alocação de processadores, máquinas multiprocessadoras hipercúbicas

Abstract

The performance achieved by a massive parallel machine depends greatly of the efficacy and efficiency of the processor allocation algorithm in use. This work analyze the possibility of achieving a better performance of the processor allocation algorithm in hypercubic multiprocessor machines with paralelization. Parallel versions of the most important algorithms found in the literature are proposed and the obtained results are presented.

Keywords: parallel processing, processor alocation, hipercube multiprocessor

[1]Bacharel em Informática (PUC/RS,1990); Mestrando do CPGCC/UFRGS; Arquitetura de Computadores, Processamento Paralelo e Distribuído; E-mail: derose@inf.ufrgs.br

[2]Professor UFRGS/CPGCC; Dr. Eng. em Informática (Instituto Nacional Politécnico de Grenoble, França,1979); Arquitetura de Computadores, Processamento Paralelo, Avaliação de Desempenho; E-mail: navaux@inf.ufrgs.br

1 Introdução

Com a diminuição no custo dos microprocessadores e com o desenvolvimento de tecnologias para a sua interconexão em grande escala, máquinas maciçamente paralelas compostas de centenas de elementos processadores se tornaram um alternativa para a construção de supercomputadores.

Neste novo conceito de processamento de dados, grandes velocidades são alcançadas através da cooperação entre os diversos elementos processadores na resolução de um problema. A forma como estes elementos processadores estão interligados afeta diretamente o desempenho do sistema como um todo, já que muitas vezes a comunicação entre eles se tornará necessária, tanto para a troca de valores intermediários como para efeito de sincronização [HWA 85].

A interconexão hipercúbica é considerada hoje, uma das formas mais eficientes de interligação neste contexto, devido a sua ótima relação do número de elementos processadores com a distância máxima entre eles, e a sua grande flexibilidade [FEN 81].

Os algoritmos de gerência e alocação de processadores são responsáveis pela obtenção e controle de um ou mais processadores da máquina multiprocessadora para a execução de tarefas de um determinado usuário. Estes algoritmos são a parte do sistema operacional da máquina que se responsabiliza pelo compartilhamento dos processadores entre diversos usuários. A precisão e o tempo de resposta destes algoritmos afetam diretamente o desempenho obtido pela máquina paralela.

Neste trabalho são propostas versões paralelas dos principais algoritmos de gerência e alocação dinâmica de processadores em máquinas multiprocessadoras hipercúbicas encontrados na literatura. Os algoritmos para os quais já foi proposta alguma versão paralela tem o modelo desta analisado e o desempenho medido. É feita uma avaliação das possibilidades de obtenção de um melhor tempo de resposta através da aplicação de modelos paralelos e os resultados obtidos com a implementação destes modelos são comparados com o desempenho dos respectivos algoritmos seqüênciais.

Na seção 2 é apresentado um resumo dos principais algoritmos seqüenciais encontrados na literatura. Nas seções 3 e 4 são propostas versões paralelas destes algoritmos e analisado o seu desempenho respectivamente. As conclusões do trabalho podem ser encontradas na seção 5 seguidas da bibliografia.

2 Estratégias de Gerência e Alocação

O problema da gerência de alocação de subcubos (subconjunto de processadores do hipercubo compartilhado que continuam respeitando a topologia hipercúbica) por parte do sistema operacional pode ser melhor compreendido se for feita uma analogia ao problema de alocação de memória. Inicialmente, com a memória totalmente livre, a alocação de segmentos de tamanhos diferentes é simples, sendo feita de forma seqüencial. Porém, ao longo do compartilhamento, segmentos vão sendo liberados após o uso e novos segmentos são requisitados.

O problema se complica devido a fragmentação da memória, ou seja, o surgimento de espaços livres entre áreas alocadas. O aproveitamento destes espaços não contíguos para atender as novas requisições se torna necessário aumentando a complexidade do procedimento de gerência. Sem este aproveitamento a área compartilhada será utilizada de forma

ineficiente, e requisições serão rejeitadas apesar de se encontrarem espaços disponíveis. No caso do compartilhamento dos nodos de um hipercubo, os nodos livres que não são reconhecidos pelo sistema operacional como subcubos são como as áreas de memória livre entre espaços alocados (figura 1(c)). O compartilhamento eficiente do hipercubo só ocorrerá se estes nodos puderem ser utilizados em novas requisições e se a fragmentação do hipercubo for mantida baixa (figura 1(b)).

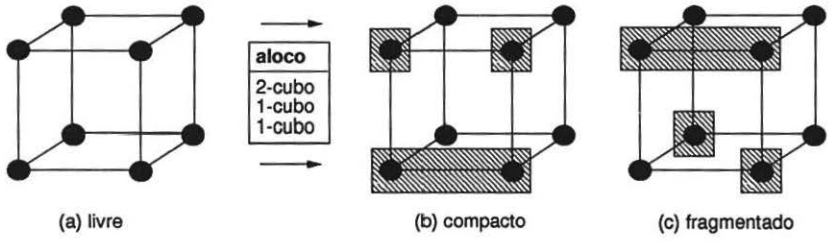

Figura 1: Disposição de processadores alocados no Hipercubo

Existem duas modalidades de alocação de subcubos: *on-line* ou dinâmica e *off-line* ou estática. Na alocação *off-line* ou estática o sistema operacional coleta um número suficiente de pedidos antes de iniciar a alocação propriamente dita. Desta forma é possível uma melhor compactação dos nodos alocados e conseqüentemente uma menor fragmentação do hipercubo. Porém, existe uma demora maior na resposta dos pedidos de alocação que muitas vezes não pode ser suportada em sistemas de tempo real. Na alocação *on-line* ou dinâmica os pedidos são aceitos ou negados imediatamente após a sua chegada, sem se levar em consideração os próximos pedidos. O tempo de resposta é melhor mas o controle da fragmentação do cubo é mais complexo [DUT 91].

Nos ítens abaixo serão descritos os algoritmos mais citados na literatura para a gerência *on-line* ou dinâmica de subcubos. São analisadas suas características de desempenho, complexidade, reconhecimento de subcubos disponíveis, tolerância a falhas e contenção da fragmentação.

2.1 Estratégia Buddy

A estratégia de gerência e alocação BUDDY [CHE 90] é a mais simples entre as estratégias que serão vistas nesta seção, reconhece apenas uma fração dos subcubos disponíveis e não se preocupa em evitar uma rápida fragmentação do hipercubo. O mecanismo de alocação utilizado atua sobre um vetor de bits numerado de 0 até $2^k - 1$ sendo k o grau do hipercubo. Cada posição deste vetor indica se o nodo em questão já está (1) ou não (0) alocado. Quando um j-cubo é requisitado ocorre uma procura no vetor pelo menor inteiro m, de tal forma que todos os nodos entre m x 2^j até $((m+1)$ x $2^j) - 1$ estejam livres. Se estes nodos são encontrados as respectivas posições no vetor são colocadas em 1. Quando este subcubo for desalocado as respectivas posições no vetor de bits são novamente colocadas em 0.

A simplicidade do mecanismo de alocação de nodos resulta em um ótimo tempo de resposta, porém, com uma baixa capacidade de reconhecimento de subcubos disponíveis e rápida tendência a fragmentação.

2.2 Estratégia GC

A estratégia de alocação GC [CHE 90] pode ser considerada uma evolução da técnica BUDDY. A técnica GC também se utiliza de uma lista de alocação para efetuar o controle dos nodos disponíveis, porém o mapeamento desta lista com os nodos do hipercubo é feito segundo os códigos de gray. Normalmente o código de gray utilizado é o BRGC (Binary Reflected Gray Code) [CHE 87].

A alocação de subcubos na lista de alocação é feita através da procura de 2^j nodos consecutivos na forma p até $p + 2^j - 1$, sendo k o grau do subcubo desejado e p o menor inteiro múltiplo de $2^j - 1$.

Foi provado em [CHE 87] que a estratégia GC pode reconhecer 2^{n-k+1} subcubos, sendo k o grau do subcubo, n o grau do hipercubo a ser compartilhado e $1 \leq k \leq n-1$, o que é o dobro de subcubos reconhecidos pela estratégia BUDDY. Nesta técnica também não são tomados cuidados para reduzir a fragmentação do hipercubo compartilhado.

A estratégia GC pode reconhecer todos os subcubos disponíveis se forem utilizados múltiplos códigos de gray (MGC) [CHE 87]. O procedimento de procura se tornaria mais complexo e o tempo de resposta maior pois diversas tabelas teriam que ser consultadas. Além disto a geração de múltiplos códigos de gray é demorada e teria que ser feita previamente e armazenada em tabelas de consulta. Desta forma uma alteração na numeração dos nodos, causada por exemplo por uma reconfiguração após alguma falha, resultaria em mau funcionamento do procedimento de gerência.

A tabela 2.2 indica o número de códigos de gray necessários para uma taxa de reconhecimento de 100% em relação a dimensão do hipercubo a ser compartilhado.

Dimensão	4	5	6	7	8	9	10	11	12	13	14
Processadores	16	32	64	128	256	512	1024	2048	4096	8192	16384
Códigos	6	10	20	35	70	126	252	462	924	1716	3432

Tabela 1: Número de códigos de gray necessários para reconhecimento total de subcubos na técnica MGC

A utilização desta estratégia não é viável em hipercubos de maior grau devido ao rápido crescimento do número de códigos necessários para que seja mantida uma alta taxa de reconhecimento.

2.3 Lista de Cubos Livres

A estratégia de alocação denominada lista de cubos livres (*Free List*) [KIM 91] efetua o controle dos nodos alocados através de listas de subcubos disponíveis no hipercubo compartilhado, utilizando uma lista para cada dimensão. Esta técnica é capaz de reconhecer todos os subcubos disponíveis porém sua complexidade é extremamente alta, superior a $O(n^3)$.

A figura 2 apresenta um exemplo da estrutura de dados utilizada pela técnica de lista de cubos livres para um hipercubo de grau 4 demonstrando as alterações sofridas após operações de alocação e desalocação.

Uma requisição pra um subcubo de dimensão k é atendida através da alocação do primeiro subcubo encontrado na lista de cubos livres de dimensão k. Caso esta lista esteja vazia, é

procurado em listas de dimensão maior o primeiro subcubo disponível que tenha um grau maior do que k. Se não existir nenhum, a requisição é rejeitada pois não existe subcubo desta dimensão disponível no momento. Caso seja encontrado um subcubo de grau maior do que k este subcubo é decomposto em dois subcubos de dimensão menor, retirado de sua lista e os novos cubos são adicionados na lista de dimensão inferior (figura 2 (a)(c)). Este procedimento é repetido até que um subcubo de dimensão k seja criado. Este subcubo então é alocado e retirado da lista de cubos livres.

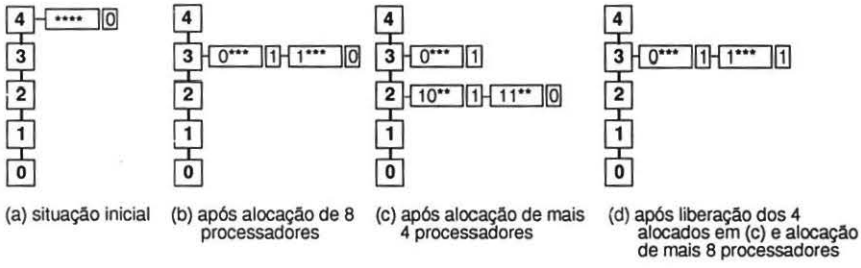

Figura 2: Atualização da estrutura utilizada pelo algoritmo FL

Apesar do procedimento de alocação ser relativamente simples, o processo de liberação de subcubos é muito complexo pois quando um subcubo é liberado é necessário verificar se não é possível re-compor algum subcubo de grau superior (figura 2 (d)). Este tipo de operação pode envolver todas as listas de subcubos livres e sua complexidade é proporcional ao tamanho do hipercubo a ser compartilhado.

Mesmo reconhecendo todos os subcubos disponíveis, a complexidade de $O(n^3)$ torna este método inviável para hipercubos de maior grau em um sistema de tempo real.

2.4 Estratégia TC

A estratégia de alocação denominada TC (*Tree Collapsing*) [CHU 90] é uma extensão da estratégia BUDDY capaz de detectar a totalidade dos subcubos disponíveis em um hipercubo compartilhado sendo menos complexa que a estratégia de múltiplos códigos gray (MGC).

A idéia básica desta estratégia é de altearar a ordem dos nodos na lista de alocação da estratégia BUDDY, gerando novas seqüências de códigos dinâmicamente de acordo com a necessidade. Os nodos que estavam distantes na tabela de alocação, e consequentemente não eram reconhecidos como subcubos, são aproximados através da consecutiva alteração nos ramos de uma árvore binária (*tree collapsing*) que representa o hipercubo compartilhado, permitindo assim seu reconhecimento (figura 3 (a)).

Esta estrátégia não difere muito da aplicação dos diferentes códigos de gray para o reconhecimento completo dos subcubos possíveis de alocação. A vantagem em relação ao método GC é que a geração destas seqüências, através da operação de *collapse* em árvores binárias, é menos complexa que a geração de múltiplos códigos de gray e pode ser feita em tempo real quando necessário.

A operação de *collapse* consiste no entrelaçamento das subárvores de um determinado

nível, gerando uma nova seqüência nos nodos associados as suas folhas. A figura 3 (b) demonstra o processo de entrelaçamento da operação de *collapse* no nível 1 da árvore primária.

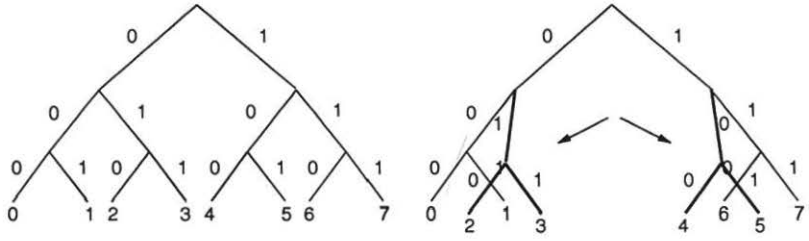

Figura 3: Estrutura utilizada pelo algoritmo TC

No atendimento de um pedido de alocação, as folhas de uma árvore binária inicial são consultadas através de um algoritmo semelhante ao algoritmo utilizado na estratégia BUDDY. Caso a resposta da procura seja negativa, são efetuadas sucessivas operações de *collapse* seguidas de uma nova consulta até que seja encontrado um subcubo que atenda o pedido de alocação ou que se esgote o número de tentativas do algoritmo. Como o algoritmo TC tem uma taxa de reconhecimento de 100%, o fato de se esgotar o número de tentativas do algoritmo indica que efetivamente não existe subcubo disponível que atenda o pedido de alocação.

O processo de desalocação de um subcubo se resume à alteração dos bits de alocação de seus nodos de 1 para 0, através de uma varredura seqüencial nas folhas da árvore.

2.5 Avaliação dos Algoritmos

A tabela 2 apresenta um resumo das características dos algoritmos para alocação e gerência de processadores em máquinas hipercúbicas vistos até agora.

	BUDDY	GC	MGC	TC	FL
Complexidade	$O(n)$	$O(n)$	$O(n^2)$	$O(n^3)$	$O(n^3)$
Uso de Memória	baixo	baixo	muito alto	médio	alto
Taxa de Reconhecimento	60%	70%	100%	100%	100%
Compactação	nenhuma	explícita	explícita	implícita	implícita
Fragmentação	rápida	rápida	rápida	média	lenta
Sistema Tempo-Real	apto	apto	não-apto	não-apto	não-apto

Tabela 2: Resumo das características encontradas nos algoritmos

Uma descrição detalhada de alguns dos ítens analisados é fornecida para uma melhor compreensão da tabela.

Taxa de Reconhecimento refere-se à capacidade do algoritmo em reconhecer subcubos disponíveis dentre os nodos não alocados. Como todo o processo de alocação é baseado no fornecimento de subcubos, uma baixa taxa de reconhecimento aumenta o número de requisições negadas e resulta em uma pior taxa de utilização do hipercubo compartilhado.

Compactação refere-se à capacidade do algoritmo em manter o hipercubo compartilhado o mais compacto possível. Quanto mais compacto o hipercubo estiver menos fragmentação existirá e mais subcubos livres serão reconhecidos. Alguns algoritmos possuem mecanismos de compactação implícitos no processo de alocação. Outros fornecem ferramentas de compactação que exigem uma parada no servidor para que possam ser executadas (explícitas).

Fragmentação refere-se à velocidade com que o hipercubo compartilhado se fragmenta ao longo do tempo de execução do algortimo. Esta velocidade está diretamente ligada com os mecanismos de compactação do algoritmo utilizado.

Sistema Tempo-Real refere-se à possibilidade da utilização do algoritmo em um sistema tempo-real. Neste tipo de sistema o tempo de resposta tem que respeitar certos limites. Os algoritmos mais complexos não satisfazem estas condições.

3 Paralelizando os Algoritmos

Como vimos anteriormente, a operação de alocação e gerência de processadores tem uma influência muito grande no desempenho final de uma máquina multiprocessadora. Uma melhora no tempo de resposta destes algoritmos acarretaria um melhor desempenho global destas máquinas.

A paralelização dos algoritmos de alocação e gerência para máquinas hipercúbicas busca não apenas esta melhora no desempenho (eficiência) mas principalmente possibilitar a utilização em um sistema tempo-real, de algoritmos que são considerados muito complexos para este tipo de ambiente. Com a utilização de algoritmos mais avançados é obtida um melhor resultado na operação de alocação (eficácia) e a máquina hipercúbica pode ser melhor compartilhada.

A maioria das máquinas hipercúbicas encontradas no mercado é ligada como máquina agregada em uma estação hospedeira. Esta estação hospedeira por sua vez está normalmente ligada a uma rede de estações permitindo que a máquina multiprocessadora seja compartilhada por diversos usuários, evitando assim que seja subutilizada. Como os algoritmos de alocação e gerência rodam na máquina hospedeira, a paralelização pode ser feita tanto com processadores das estações vizinhas como com nodos do hipercubo.

Nas subseções abaixo são apresentadas versões paralelas dos algoritmos de alocação e gerência de processadores em máquinas hipercúbicas descritos na seção 2. Um estudo mais completo sobre os algoritmos aqui propostos pode ser encontrada em [DER 93]. A análise detalhada dos resultados obtidos pelas versões paralelas é apresentada na seção 4.

3.1 Algoritmo Buddy/GC Paralelo

Na versão paralela do algoritmo BUDDY o processador da máquina hospedeira é visto como mestre e os outros como escravos, como pode ser visto na figura 4 (a). O mestre é responsável por receber os pedidos de alocação ou desalocação, distribuir o trabalho a ser feito entre os escravos, sincronizar as operações quando necessário, e responder aos pedidos.

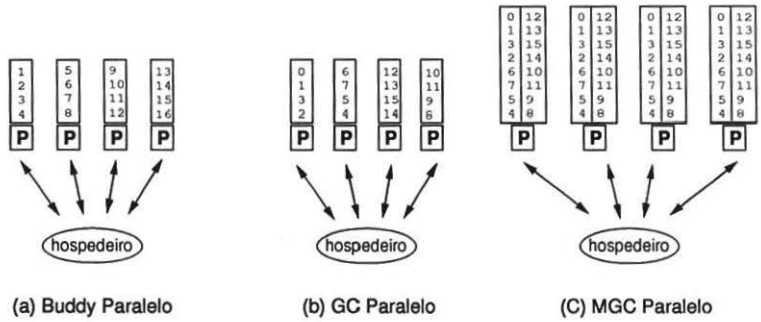

Figura 4: Modelo mestre-escravo das versões paralelas baseadas em listas

- **Inicialização** A lista de alocação, estrutura utilizada para marcar os processadores alocados, é fragmentada e distribuída entre os processadores envolvidos.
- **Alocação Paralela**
  1. o processador mestre recebe o pedido de alocação e envia para os processadores escravos o número de processadores desejados e o pid (process ID - identificação do processo requisitante) associado ao pedido;
  2. mestre e escravos executam a operação de alocação em suas listas locais;
  3. o primeiro que encontrar um resultado manda o sinal de EOF para os outros processadores e se não receber EOF marca suas listas locais com o pid como alocadas e envia o número dos processadores alocados para o processador mestre (se não o for);
  4. caso os escravos não encontrem livre o número de processadores desejados em suas listas, enviam um sinal NOT para o mestre;
  5. ao receber um EOF o processador aborta a operação de procura;
  6. para que a operação seja concluída pelo mestre este (i) espera todos os escravos responderem com NOT e nega o pedido de alocação, (ii) espera a lista de nodos alocados de um de seus escravos ou (iii) não espera por nada por ter ele mesmo encontrado os processadores desejados em sua lista local.
- **Desalocação Paralela**

1. o mestre recebe pedido de desalocação e envia o pid a ser desalocado para escravos;
2. mestre e escravos liberam os nodos do pid recebido em suas listas locais;
3. não existe a necessidade de sincronização no final da operação, estando o mestre liberado para novos pedidos logo após o final da consulta a suas listas locais.

Como o processador mestre tem mais atribuições que os escravos e conseqüentemente mais carga de trabalho, pode ser feito um balanceamento de carga estático antes do início das operações atribuindo-se fragmentos maiores da lista de alocação aos escravos do que ao mestre.

Mecanismos simples que neguem ou aceitem pedidos de forma imediata pelo mestre, sem que ocorra consulta aos escravos, aumentam consideravelmente o desempenho da versão paralela do algoritmo BUDDY. Um exemplo deste tipo de mecanismo seria a negação imediata de um pedido de alocação de um número de processadores maior do que os que estão disponíveis no momento (controle local e remoto), ou a aceitação imediata de um pedido por parte do mestre quando o número de processadores alocados no hipercubo for muito pequeno e estiver disponível localmente.

O controle da alocação de todo o hipercubo compartilhado é feito no processador da máquina hospedeira (mestre) com apenas um bit. Se este mecanismo não fosse implementado não seria possível a alocação de todo o hipercubo compartilhado pois sua lista de alocação está partida em pedaços e os nodos não seriam reconhecidos como vizinhos. Caso este algoritmo paralelo seja implementado em mais de 2 processadores, mecanismos semelhantes para a alocação de dimensões menores tem que ser implementados. Se, por exemplo, a lista de alocação de um hipercubo de grau k for dividida entre 4 processadores de forma igual, nenhum deles será capaz de alocar um subcubo com 2^{k-1} processadores.

O mesmo algoritmo paralelo descrito acima pode ser utilizado para a estratégia GC alterando-se as listas de alocação em cada processador e a forma de procura dos nodos livres durante a fase de alocação de processadores, como indicado na subseção 2.2.

3.2 Múltiplos Códigos de Gray Paralelo

Na versão paralela do algoritmo MGC é utilizado o mesmo modelo mestre-escravo descrito no algoritmo BUDDY paralelo (figura 4 (c)).

O cálculo e a distribuição dos códigos de gray entre os processadores é feito *off-line* e não pode ser alterada durante a operação (estático).

- **Inicialização** O número de códigos de gray necessários para atingir a taxa de reconhecimento desejada são distribuídos igualmente entre os processadores envolvidos (mestre e escravos).
- **Alocação Paralela**
  1. o mestre recebe o pedido de alocação e envia para os escravos o número de processadores desejados e o pid associado ao pedido;
  2. mestre e escravos iniciam a procura por processadores livres em suas listas locais;

3. o primeiro que encontrar um resultado manda o sinal de EOF para os outros processadores e se não receber EOF marca suas listas locais com o pid como alocadas e envia o número dos processadores alocados para o mestre (se não o for);
4. caso os escravos não encontrem o número de processadores livres desejados em suas listas, enviam um sinal NOT para o mestre;
5. ao receber um EOF o processador aborta a operação de procura;
6. para que a operação seja concluída pelo mestre este (i) espera todos os escravos responderem com NOT e nega o pedido de alocação, (ii) espera a lista de nodos alocados de um de seus escravos e atualiza as listas dos restantes, ou (iii) não espera por nada por ter ele mesmo encontrado o número de processadores desejados em suas listas locais.

- **Desalocação Paralela**

1. mestre recebe um pedido de desalocação e envia o pid a ser desalocado para escravos;
2. mestre e escravos liberam os nodos do pid recebido em suas listas locais;
3. não existe a necessidade de sincronização no final da operação, estando o mestre liberado para novos pedidos logo após o final da consulta a suas listas locais.

O mesmo balanceamento de carga estático proposto no algoritmo BUDDY paralelo pode ser aplicado aqui se antes do início das operações for atribuído mais códigos de gray aos escravos do que ao mestre.

Os mesmos mecanismos de aceitação e negação imediata de pedidos propostos no algoritmo BUDDY paralelo também aumentam consideravelmente o desempenho do algoritmo MGC paralelo.

3.3 Tree Collapse Paralelo

Uma versão paralela do algoritmo de *Tree Collapse* é proposta por Chuang e Tzeng em [CHU 90]. Como a operação de *collapse* é bastante complexa e durante a fase de alocação é executada normalmente por diversas vezes, a versão paralela se baseia na execução desta operação por vários processadores em paralelo. A operação de desalocação é executada, devido a sua simplicidade, apenas no processador da máquina hospedeira.

Requisições de subcubos são gerenciadas pelo processador da máquina hospedeira que mantém o vetor completo de bits de alocação. No recebimento de um pedido de alocação, o processador hospedeiro fornece uma cópia do vetor de bits de alocação para cada processador envolvido, juntamente com a seqüência de operações que deve ser efetuada sobre ele. Os processadores envolvidos efetuam suas seqüências de operações e a procura pelo subcubo desejado em paralelo. O primeiro processador que encontrar, avisa o processador hospedeiro que aborta a operação incompleta dos processadores restantes. O vetor de bits de alocação é então atualizado no processador hospedeiro.

O número de processadores envolvidos no processo de alocação pode ser pré-determinado ou escolhido em tempo de execução. A segunda forma resulta em melhores resultados pois

evita que vários processdores sejam envolvidos em um procedimento de alocação simples que poderia ser facilmente resolvido até mesmo pelo processador hospedeiro. O processamento necessário para uma alocação pode ser estimado pelo processador hospedeiro com base na taxa de ocupação e na taxa de fragmentação do hipercubo compartilhado.

3.4 Lista de Cubos Livres Paralelo

Uma versão paralela do algoritmo de lista de cubos livres é proposta por Kim e Das em [KIM 91].

- **Inicialização** A lista de cubos livres para cada dimensão do hipercubo a ser compartilhado é mantida em um processador diferente. Para um hipercubo compartilhado de grau n são necessários $(n+1)$ processadores, incluindo o processador da máquina hospedeira.

- **Alocação Paralela**

  1. o processador da máquina hospedeira envia o grau do subcubo desejado (k) para os processadores que mantém as listas de dimensão $\geq k$;
  2. todos os processadores, dentre os acima, que possuírem um subcubo livre de dimensão m para um $m \geq k$ respondem à requisição enviando o subcubo disponível;
  3. o processador da máquina hospedeira escolhe o subcubo de dimensão mais próxima a k, decompondo-o se necessário ($> k$) e enviando os subcubos decompostos para os respectivos processadores. O subcubo escolhido já é marcado como alocado;
  4. o processador hospedeiro envia uma mensagen para que o processador da dimensão do subcubo que foi decomposto, se existir, o marque como alocado.

- **Desalocação Paralela**

  1. o subcubo k desalocado é enviado e adicionado na lista do processador responsável pelos subcubos de dimenão k;
  2. o processador da máquina hospedeira envia o subcubo k liberado para todos os processadores que mantém dimensões $> k$. Todos estes processadores geram possíveis subcubos comparando suas listas com o subcubo recém liberado. Os subcubos i gerados são enviados para o processador responsável pela dimensão i;
  3. cada nodo envia uma cópia da sua lista para o processador da dimensão superior. Se o processador que recebeu a mensagem estiver com sua lista vazia a mensagem é redirecionada para o processador de dimensão superior. Ao mesmo tempo, cada processador gera subcubos sobrepostos utilizando sua lista e os os subcubos recebidos pelo vizinho de dimensão inferior. Se um dos subcubos recebidos não puder ser combinado, o processador o envia para o processador responsável pela lista de dimensão diretamente superior;
  4. do processador que controla a lista da maior dimensão, é enviado um subcubo por vez para todos os outros processadores. Este passo é repetido sucessivamente pelos processadores responsáveis pelas dimensões menores, um a um;

5. cada processador decompoe seus subcubos que possuem nodos comuns com os subcubos recebidos. Os subcubos comuns são apagados e os subcubos restantes de menor dimensão são enviados para os processadores correspondentes para atualização. Os passos 4 e 5 podem ser feitos em paralelo.

Como podemos ver, o algoritmo de listas de cubos livres, como foi originalmente proposto por [KIM 91], tem uma alta complexidade e gera um grande número de mensagens entre os processadores, principalmente na fase de desalocação.

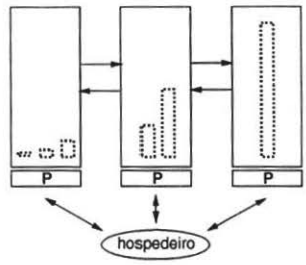

Figura 5: Modelo da versão paralela otimizada do algoritmo TC

Uma forma de diminuir este pico de tráfego entre processadores durante a fase de desalocação consiste no agrupamento de várias listas em um mesmo processador (figura 5). Desta forma, muitas das mensagens que seriam trocadas entre listas para sua atualização são transformadas em processamento local.

Um balanceamento de carga estático pode ser feito com o agrupamento das listas de forma desigual entre os processadores. Como o tamanho máximo das listas varia de 1 até 2^k, sendo k a dimensão do hipercubo compartilhado, listas de menor tamanho (maior dimensão) podem ser agrupadas em maior número enquanto as listas de menor dimensão (maior tamanho) ficam sob responsabilidade de um só processador.

4 Os Resultados Obtidos

O modelo de simulação utilizado para comparação dos algoritmos de alocação encontrados na literatura e suas versões paralelas baseia-se em um gerador de requisições e um servidor que executa os algoritmos e atende as requisições. O gerador de requisições aceita parâmetros como tempo de simulação, número de clientes, tempo médio de alocação e número médio de processadores a serem requisitados. O servidor atende os pedidos de acordo com a capacidade do algoritmo em teste enfileirando as requisições que não puderem ser atendidas. A política de escalonamento da fila de espera é FCFS. Durante a simulação o gerador de requisições monitora o funcionamento do algoritmo em termos do número de pedidos negados, tempo médio de espera na fila, tempo total da simulação e taxa de utilização do hipercubo compartilhado.

Para fins de comparação foram implementadas versões seqüenciais dos algoritmos descritos na seção 2 e as versões paralelas propostas na seção 4. A linguagem utilizada foi "C" e as unidades de tempo medidas correspondem a uma estação SUN Sparc Station IPC. A

paralelização dos algoritmos foi feita utilizando recursos da rede de estações e a comunicação implementada sobre a diretiva *socket*.

Dentre as dificuldades encontradas durante o processo de paralelização dos algoritmos de alocação, o alto custo da comunicação entre os processos paralelos é sem dúvida a mais significativa. Apesar de, na maioria dos casos, não existir grande dependência entre as operações que foram paralelizadas, a comunicação entre os processos é utilizada para distribuir as tarefas entre os processadores e para a troca de sinais de sincronização.

Como o número de mensagens de distribuição de tarefas e de sincronização entre processos cresce à medida que aumentamos o número de processadores envolvidos na operação de paralelização, a maioria das versões paralelas propostas neste trabalho só é viável quando executada por um pequeno número de processadores.

Os melhores resultados foram obtidos quando os algoritmos mais complexos são paralelizados em um pequeno número de processadores. Desta forma existe trabalho suficiente para ser feito por vários processadores, valendo a pena distribuí-lo mesmo com os altos custos de comunicação envolvidos.

Outro fator importante é que quanto mais alto o grau do hipercubo a ser compartilhado, mais custoso será o processo de alocação de processadores, aumentando assim a carga de trabalho a ser distribuído e melhorando os resultados das versões paralelas.

		256	512	1024	2048	4096	8192	16384
BUDDY/GC	W_a	0.46	0.97	2.15	4.5	11.8	19.5	42.07
	W_r	0.05	0.09	0.16	0.32	0.63	1.26	2.69
	Δtr	0.06	0.08	0.22	0.55	1.17	2.16	3.84
BUDDY/GC ||	W_a	0.30	0.43	0.68	1.10	2.43	4.35	6.55
	W_r	0.12	0.17	0.27	0.41	0.72	1.36	2.64
	Δtr	0.41	0.42	0.48	0.57	0.71	0.93	1.21
MGC	W_a	15.15	54.42	195.91	705.98	2539	9140.47	32905.69
	W_r	3.5	11.34	40.32	147.84	582.12	2162.16	9232.08
	Δtr	0.84	2.01	11.08	50.82	216.21	741.31	2635.77
MGC ||	W_a	9.46	23.45	60.46	171.6	516.05	1820.71	5062
	W_r	2.15	5.45	12.59	39.90	117.14	413.63	1150
	Δtr	1.86	2.19	9.83	35.88	121.46	319.53	784.45
TC	W_a	62.35	70.55	86.06	112.98	140	189	221.13
	W_r	0.05	0.09	0.16	0.32	0.63	1.26	2.63
	Δtr	9.25	13.15	19.86	26.59	32.88	47.68	54.83
TC ||	W_a	50.68	58.28	69.93	73.43	91.79	123.92	154.9
	W_r	0.12	0.17	0.27	0.41	0.72	1.36	2.64
	Δtr	10.02	11.23	13.08	15.25	17.86	25.02	30.8
FL	W_a	3.54	7.28	14.83	29.35	58.6	115.23	245.66
	W_r	11.5	25.65	75.95	195.20	501.68	1289.32	3314.57
	Δtr	9.75	24.87	58.46	149.09	365.27	858.39	2017.23

Tabela 3: Resultados obtidos pelos algoritmos implementados em centésimo de segundo

A tabela 3 apresenta uma comparação dos resultados obtidos pelas versões paralelas e as versões seqüenciais dos algoritmos de gerência e alocacão para hipercubos compartilhados de grau 8, 9, 10, 11, 12, 13 e 14 utilizando 2 processadores nas versões paralelas. Todos os resultados são fornecidos em centésimos de segundo. Os valores relativos a W_a e W_r

referem-se ao tempo gasto pela pior alocação (*Worst Allocation*) e pior desalocação (*Worst Release*) respectivamente. Os valores relativos a Δtr referem-se ao tempo médio gasto com o atendimento de uma requisição.

4.1 Buddy/GC Paralelo

A fraca dependência entre as operações executadas nas listas de alocação distribuídas entre os processadores envolvidos mantém o fluxo de mensagem entre os processadores baixo e permite uma distribuição de trabalho simples e balanceada.

A operação de alocação tem um desempenho bem superior a desalocação devido a sua maior carga de trabalho. Na desalocação o trabalho a ser feito é muito pequeno para justificar os custos de comunicação quando do envolvimento de outro processador na operação.

O fluxo de mensagens durante a execução do algoritmo é muito baixo devido ao fato de só serem usadas mensagens para a distribuição do trabalho a ser feito e devolução dos resultados. As mensagens são de tamanho pequeno, variando de um a três bytes.

O desempenho do algoritmo depende muito da capacidade do processador mestre em detectar a necessidade ou não do envolvimento de mais processadores na operação requisitada. Com as diretivas propostas na subseção 3.1 o algoritmo atingiu um *Speed-Up* de 1.64 com 2 estações de trabalho gerênciando um hipercubo de grau 12.

4.2 MGC Paralelo

A principal diferença no funcionamento da versão paralela do algoritmo BUDDY/GC e a versão MGC é que no segundo caso a lista de alocação não é partilhada entre os processadores envolvidos. Cada processador recebe uma cópia desta lista que após cada operação tem que ser atualizada. Esta atualização tem um custo significativo pois aumenta o tempo de comunicação, principalmente quando são alocados subcubos de grande dimensão.

Por outro lado, cada nodo possui uma ou mais listas de códigos de gray para processar, o que aumenta consideravelmente a carga computacional de cada nodo em hipercubos compartilhados de grande dimensão. Com este aumento de carga de trabalho a paralelização deste procedimento obtém melhores resultados do que a versão paralela do algoritmo BUDDY/GC. Foi obtido um *Speed-Up* de 1.78 na gerência de um subcubo de grau 12.

4.3 Tree Collapse Paralelo

A operação de *Collapse*, sobre a qual se baseia o algortimo TC, tem uma alta complexidade e é chamada várias vezes durante uma operação de alocação o que resulta em bons resultados quando este procedimento é paralelizado. A estrutura de dados que controla as alocações, porém, é copiada para cada processador e necessita de atualização como descrito em 4.2. O principal problema no entanto é a grande variação da carga de trabalho de uma requisição de alocação. A capacidade do algoritmo paralelo de detectar quando a ativação de um nodo remoto é justificada pela carga de trabalho a ser feita é fundamental para o bom desempenho da versão paralela. Na simulação executada foi obtido um *Speed-Up* de 1.84 na gerência de um subcubo de grau 12.

4.4 Free List Paralelo

Na tabela 3 não foram colocados os resultados obtidos pela versão paralela do algoritmo FL. Isto decorre do fato desta versão, proposta por Kim e Das em [KIM 91] ser muito complexa e gerar um tráfego de mensagens muito alto que não é suportado de forma alguma pelo ambiente de compartilhamento proposto. Nem mesmo as otimizações propostas na subseção 3.4 tornaram esta versão paralela viável neste tipo de ambiente.

Este péssimo desempenho da versão paralela resulta da dificuldade de se paralelizar o algoritmo FL de forma eficiente. O algoritmo opera sobre uma estrutura de múltiplas listas que é tratada de forma seqüencial existindo um forte dependêcia entre as operações.

5 Conclusões

Neste trabalho é analisada a utilização de técnicas de paralelização com o objetivo de otimizar o desempenho dos algoritmos de alocação e gerência de processadores para máquinas hipercúbicas encontrados na literatura.

Foram propostas versões paralelas do algoritmo BUDDY, GC e MGC e sugeridas melhorias nas versões paralelas dos algoritmos *Tree Collapsing* [CHU 90] e Lista de Cubos Livres [KIM 91] propostas pelos respectivos autores.

Tanto os algoritmos seqüenciais como as versões paralelas foram implementadas e seu desempenho foi avaliado através de simulação das condições do ambiente alvo desejado.

Os resultados demonstram que é possível reduzir em média pela metade o tempo de atendimento a uma requisição para máquinas compartilhadas de dimensão elevada (grau acima de 10) e utilizando-se de recursos da rede de estações na paralelização dos algoritmos. É importante resaltar que com a utilização de processadores da máquina paralela no processo de paralelização seriam seguramente obtidos resultados melhores, devido ao menor tempo de comunicação entre o hospedeiro e estes processadores.

A necessidade de sincronização e o alto tráfego de mensagens gerado nas versões paralelas restringem o número de processadores que podem ser usados na paralelização. Os resultados mostraram que o melhor desempenho foi obtido com um pequeno número de processadores (2 a 4), dependendo do algoritmo.

Com as versões paralelas dos algoritmos mais complexos como MGC, TC foi possível melhorar o seu tempo de resposta, viabilizando estes algoritmos, sob certas condições, para sistemas compartilhados em tempo real.

A possibilidade da utilização destes algoritmos neste tipo de sistema melhora a taxa de utilização da máquina multiprocessadora hipercúbica permitindo um uso mais racional do recurso compartilhado.

A utilização de algoritmos avançados de alocação e gerência de processadores em máquinas multiprocessadoras de grau elevado em sistemas de tempo real será apenas possível com a aplicação eficiente de técnicas de paralelização que reduzam consideravelmente a complexidade temporal destes algoritmos.

Referências

[ALD 89] AL-DHELAAN, A. & BOSE, B. A New Strategy for Processor Allocation in an

N-cube Multiprocessor. Phoenix Conference Comp. and Comm. **Proceedings.** March 1989.

[CHE 87] CHEN, M. & SHIN, K. Processor Allocation in an n-cube multiprocessor using Gray codes. **IEEE Transactions on Computers vol 36(12).** December 1987.

[CHE 90] CHEN, M. & SHIN, K. Subcube allocation and task migration in hypercube multiprocessors. **IEEE Transactions on Computers vol 39(9).** September 1990. pp. 1146-1155.

[CHU 90] CHUANG, P. & TZENG, N. Dynamic Processor Allocation in Hypercube Computers. The 17th Annual Symposium on Computer Architecture, Seattle. **Proceedings.** May 1990. pp. 40-49.

[CHU 92] CHUANG, P. & TZENG, N. A Fast Recognition–Complete Processor Allocation Strategy for Hypercube Computers. **IEEE Transactions on Computers, vol 41(4).** April 1992.

[DER 93] DE ROSE, Cesar **Alocação de processadores em máquinas hipercúbicas.** Dissertação de Mestrado. UFRGS - CPGCC. 1993. (em fase de conclusão)

[DUT 91] DUTT, S. & HAYES, P. Subcube allocation in Hypercube Computers. **IEEE Transaction on Computers vol 40(3).** March 1991.

[FEN 81] FENG, T. A Survey of Interconection Networks. IEEE Computer. Dec 1981.

[HUA 90] HUANG, C., JUANG, J. A Partial Compaction Scheme for Processor Allocation in Hypercube Multiprocessors. International Conference on Parallel Processing. **Proceedings.** 1990.

[HWA 85] HWANG, K., BRIGGS, A. Computer Architecture and Parallel Processing. McGraw-Hill International Editions. 1985.

[KIM 89] KIM, J., DAS, R. & LIN W. A processor Allocation Scheme for Hypercube Computers. International Conference on Parallel Processing. **Proceedings.** August 1989.

[KIM 91] KIM, J., DAS, R. & LIN W. A Top-Down Processor Allocation Scheme for Hypercube Computers. **IEEE Transactions on Parallel and Distributed Systems, vol 2(1).** January 1991.

[TRI 90] TRINDADE Jr., O. & SANTANA, M. j. Um Servidor de Processamento Paralelo Baseado em Transputers – Requisitos e Definição. III Simpósio Brasileiro de Arquitetura de Computadores - Processamento Paralelo III SBAC — PAD. **Proceedings.** Rio de Janeiro, 7 a 9 de novembro de 1990. pp. 225-237.