SIMULAÇÃO DE MODELOS PARALELOS DE PROGRAMAÇÃO EM LÓGICA

Ruy Marinho da Costa[1]
Cláudio Luis de Amorim[2]

Resumo

Apresentamos neste trabalho o projeto e a implementação de um simulador que tem por finalidade avaliar a eficiência de modelos de execução paralela de programas lógicos. O simulador permite otimizar modelos e obter medidas de desempenho tais como número de unificações, comunicações e processos criados. A arquitetura do simulador é descrita em termos de seus componentes: processadores virtuais, processos, comunicações e sincronização. Os resultados obtidos na implementação dos modelos *Backup* e *Kabu-Wake* são discutidos e confirmam o simulador como uma ferramenta simples, versátil e útil para a pesquisa em modelos paralelos de programação em lógica.

Abstract

In this work we present the design and implementation of a parallel logic programming models simulator. The simulator allows the efficiency of parallel execution models of logic programming to be evaluated and optimized by producing performance measures such as the number of unifications, communications and new processes which are created by each model on executing typical programs. The simulator architecture is described in terms of their main components namely virtual processors, processes, communication and synchronization activities. The results produced by running the *Backup* and *Kabu-Wake* models are discussed and they confirm the simulator is a simple and versatile tool for carryng out research in parallel logic programming models.

[1]MsC (COPPE Sistemas / UFRJ - 1991); 2º Ten (CQC-CA); BASE NAVAL DO RIO DE JANEIRO, CONTRATORPEDEIRO MARCÍLIO DIAS; Tel.: (021) 716-1495

[2]PhD (Imperial College - 1984); COPPE Sistemas / UFRJ Cx.P. 68511 - CEP 21.945-970, Rio de Janeiro - RJ. Tel.: (021) 590- 2552 Fax: (021) 290-6626; E-Mail: amorim@rio.cos.ufrj.br cos99203@ufrj.bitnet

1 - Introdução

Uma das dificuldades à realização de estudos comparativos entre os modelos de execução paralela propostos para Programação em Lógica, basicamente PROLOG, é a falta de uniformidade entre as implementações realizadas e, freqüentemente, a própria falta de uma implementação. Essas, quando realizadas, sempre o foram em condições específicas, em máquinas diferentes (muitas das quais experimentais) e utilizando-se de técnicas também diversas, tornando sem sentido realizar alguma comparação efetiva entre os resultados obtidos por essas implementações. Por outro lado, criar um ambiente comum, dotado de diversas facilidades, onde apenas os modelos, e não as implementações, pudessem ser avaliados, sob as mesmas condições, foi a principal motivação para o desenvolvimento do simulador apresentado neste trabalho.

A simulação possibilita medir as operações fundamentais (quantidade de unificações, comunicações, etc...) realizadas pelos modelos e aquelas que possam ser consideradas importantes para o seu desempenho, tornando-se, desta forma, uma ferramenta realmente capaz, não só de proporcionar condições de avaliação adeqüada do desempenho de um modelo, mas também de se comparar diferentes modelos.

Outra vantagem muito significativa da simulação é o de poder facilmente ajustar ou alterar um modelo e avaliar o efeito resultante, confrontando os resultados produzidos pelas diferentes versões. Vale notar que, na maioria dos casos, os resultados publicados na literatura são oriundos de execuções de programas diferentes, dificultando ainda mais um estudo comparativo dos modelos. Para ser obtida uniformidade nas medições, o mesmo conjunto de programas deve ser empregado. Existem ainda as vantagens naturais de um simulador: à medida que não se fazem necessários nem a implementação real de um modelo, nem a construção do *hardware* específico do modelo (quando for exigido por este). Podemos implementar diversos modelos (ou variações de um mesmo modelo) com pequeno dispêndio de tempo e de recursos. Por fim, lembramos que condições tais como a variação do número de processadores, tornam-se viáveis se realizadas através da simulação.

A próxima seção descreve o simulador e cada um dos seus componentes, fornecendo uma visão geral do funcionamento do mesmo. A seguir são apresentados o ambiente de implementação, o ambiente simulado e os modelos simulados. A quarta seção descreve os procedimentos necessários para a simulação de um modelo. Na quinta seção são apresentados e comentados os testes dos modelos e realizada uma comparação entre os resultados dos modelos. Finalmente, na última seção, são apresentadas as conclusões sobre o trabalho realizado.

2 - Arquitetura do Simulador

O simulador é composto basicamente pelos seguintes componentes:

O **Compilador**: que atua previamente ao simulador propriamente dito, realizando as análises léxica e sintática do programa fonte PROLOG a ser simulado. É responsável por gerar o código a ser utilizado durante o processo de simulação, que é armazenado em um **Banco de Cláusulas**, ao qual todos os processos têm acesso, emulando o comportamento tanto de uma memória global como o de uma memória distribuída. As cláusulas são postas no banco preservando a ordem

de declaração das mesmas no fonte. Este procedimento é fundamental para que determinados algoritmos PROLOG venham a funcionar corretamente, em uma estratégia de controle nos moldes da utilizada nas implementações PROLOG;

O **Sincronizador:** que funciona como um relógio global, determinando o tempo para operações dos **Processadores Virtuais**, onde a cada unidade de tempo os processadores ativos realizam uma ou parte de uma tarefa e os processadores inativos registram este tempo como ocioso. Como cada tarefa tem complexidade diferente, também o tempo para realização das mesmas difere. São consideradas as seguintes tarefas:

- *Pesquisa no banco de cláusulas;*
- *Unificação de termos simples;*
- *Criação de processos locais;*
- *Criação de processos remotos;*
- *Término de processo;*
- *Envio de mensagem local;*
- *Envio de mensagem remota;*
- *Recepção de mensagens;* e
- *Particularidades do modelo.*

Os tempos para cada tarefa devem ser obtidos por medições na máquina alvo e a seguir passados para o simulador, adaptados a uma escala onde o menor valor é 1. As particularidades devem ser definidas pelo usuário e têm como objetivo principal contabilizar os pontos de *overhead* que um modelo possua;

Os **Processadores Virtuais:** são em número determinado pelo usuário, obedecendo à topologia de interligação determinada pelo mesmo, são formados por estruturas de dados que permitem a coordenação dos **Processos** dos modelos simulados, por parte do **Sincronizador**, através da realização de bloqueios em determinadas operações fundamentais;

Os **Processos:** são formados pela implementação dos algoritmos dos modelos e criados como processos reais da máquina hospedeira. Para permitir ao modelo o processamento de um programa PROLOG, o simulador oferece uma biblioteca com funções que permitem:

- *Obter consultas;*
- *Obter cláusulas;*
- *Obter o número de literais em uma cláusula;*
- *Obter o i-ésimo literal de uma cláusula;*
- *Aplicar uma lista a um literal;*
- *Aplicar uma lista a uma cláusula;*
- *Obter uma lista com as variáveis de um literal;*
- *Unir duas listas;*
- *Combinar duas listas;* e
- *Obter uma lista de instanciações de variáveis.*

A biblioteca de funções permite a interpretação do programa armazenado no **Banco de Cláusulas** através do método de cópia de valores e da manutenção de listas de instanciações pelo usuário;

Controlador de Comunicações: que manipula as mensagens internas e externas dos **Processos** de modo a garantir que tanto o roteamento quanto o tempo de transmissão sejam corretamente emulados. Normalmente, a comunicação entre processos *PROLOG* é realizada entre pai e filho, ou entre processos previamente definidos e distribuídos segundo uma regra bem definida. Já o recebimento de mensagens normalmente exige que estas sejam recebidas de uma fonte específica, embora em algumas ocasiões um processo possa esperar por uma mensagem qualquer. Agrega-se o fato de que um processo pode querer ou não esperar pela chegada de uma mensagem, ou esperar ou não pela recepção de uma transmissão sua.

O sistema de comunicações deve simular diversos canais de capacidade a ser definida pelo usuário (com *default* 0). Estes canais seriam criados e eliminados automaticamente, de acordo com as necessidades dos processos, sem a participação explícita do usuário. Sendo assim, os próprios canais ficam responsáveis pelo armazenamento das mensagens;
O esquema de ligação desses componentes em um processador real pode ser visto na figura 1. Cabe aqui observar que a arquitetura do simulador não influencia a dos modelos, sendo possível que qualquer topologia e esquema de memória (distribuída ou compartilhada) sejam aplicadas a um modelo.

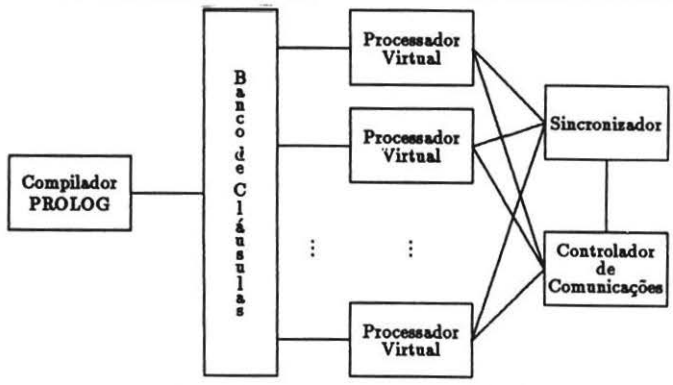


Figura 1: Arquitetura do Simulador

3 - Implementação do Simulador

O simulador foi implementado no NCP-I da COPPE/UFRJ, uma máquina paralela de arquitetura híbrida (distribuída + compartilhada), composta por uma unidade para a realização de E/S (SCSI, 600 Mbytes), um micro-computador hospedeiro (IBM-XT/AT compatível) e outra para o processamento. A unidade que realiza o processamento é composta por um hipercubo de TRANSPUTERS, sendo que a cada unidade TRANSPUTER está conectada uma unidade i860, sendo essas últimas conectadas a um barramento VME. O simulador foi implementado apenas no hipercubo, tendo sido utilizados o sistema operacional, tipo UNIX distribuído (HELIOS), e a linguagem C existente no referido ambiente.

Para a realização das simulações foram escolhidos os modelos *Backup*, **[Furukawa 82]**, e *Kabu-Wake*, **[Sohma 85]**, por suas características de simplicidade de implementação e alteração, como também pelas estratégias completamente opostas que optaram para realizar a exploração de paralelismo. Também um modelo seqüencial PROLOG foi implementado e simulado para servir como referência.

[Furukawa 82] apresenta uma técnica que pode ser considerada como uma variação do paralelismo *AND*. Durante a resolução do corpo de uma cláusula com *n* objetivos, após a resolução de um dos objetivos e enquanto se tenta resolver o objetivo corrente, uma (e somente uma) nova solução para o anterior é procurada simultaneamente. A idéia é acelerar um possível retrocesso, no caso de falha do objetivo corrente. Portanto estando-se no i-ésimo

objetivo da cláusula, os i - 1 anteriores estarão com uma solução alternativa ou em busca da mesma. É importante notar que o processo se repete em cada cláusula sendo executada.

[Sohma 85] procura evitar os problemas gerados pela exploração de paralelismo *OR* criando uma arquitetura específica onde cada processador trabalha seqüencialmente. Deste modo o número de tarefas realizadas em paralelo será no máximo igual ao número de processadores no sistema. Dispõe de duas redes de comunicação: uma para troca de contextos e outra para que os processadores ociosos possam realizar pedidos de trabalho. Um dos aspectos mais interessantes deste modelo, denominado *Kabu-Wake*, está no seu modo de determinar os pontos de quebra para processamento paralelo. Dentre os vários que são determinados ao longo do processamento, o ponto escolhido é aquele mais próximo à raiz.

Cada modelo foi implementado na arquitetura que lhe era mais favorável. Entretanto, para servir de base na determinação dos tempos das operações básicas para o simulador, foi utilizado o TRANSPUTER como processador base, devido às suas características na gerência de processos e comunicações locais e remotas. Os tempos das operações básicas foram calculados com base nos valores obtidos em medições destas operações na máquina base. O menor valor obtido por estas medições é usado para normalizar os demais valores obtidos. Os valores resultantes são então arredondados. Assim, o menor valor será sempre a unidade e todos os valores serão inteiros. A relação dos tempos calculados, utilizada nas simulações dos modelos implementados, pode ser encontrada na tabela 1.

Pesquisa no banco de cláusulas	1
Unificação de termos simples	10
Criação de processos locais	6
Criação de processos remotos	7
Término de processo	1
Envio de mensagem local	1
Envio de mensagem remota	2
Recepção de mensagens	2
Particularidades do modelo	1

Tabela 1: Tempo das Operações Básicas

É importante notar que nenhum dos modelos simulados poderia ser implementado em TRANSPUTERS sem modificações significativas das suas características, devido principalmente à problemas de topologia. Ressalta-se que qualquer outro ambiente poderia ser escolhido como base, e que o ambiente simulado é totalmente independente do ambiente no qual o simulador foi implementado.

4 - Modelagem

Para se avaliar um modelo usando o simulador, alguns procedimentos básicos devem ser adotados. Primeiro, deve ser realizada uma análise do modelo buscando identificar os seguintes aspectos principais:

Topologia: deve ser identificada e analisada as implicações

desta nas comunicações e nos processos;

Comunicações: o tamanho e os tipos de mensagens, bem como quais os tipos de processos e o fluxo que estas seguem (se sempre de pai para filho, por exemplo);

Controle: como o modelo decide a distribuição das tarefas entre processos e de processos entre processadores; e

Processos: os diferentes tipos encontrados e seus respectivos algoritmos.

Outros aspectos relevantes, particulares de um modelo, também devem ser analisados e suas implicações levadas em consideracão na simulação.

Finalmente o modelo deve ser programado utilizando-se das funções disponíveis na biblioteca do simulador (como as citadas na segunda seção, no tópico sobre **Processos**) e ligado às mesmas. A topologia é definida quando se programa a distribuição, a iniciação e a comunicação dos processos.

A tabela 2 expõe as principais características dos modelos *Backup* e *Kabu-Wake*, sob o ponto de vista dos procedimentos citados anteriormente.

Ítem	Backup	Kabu-Wake
Topologia	Totalmente Conectado	2 Redes : Anel e Similar a Estrela
Comunicações	Capacidade Zero, do Filho para o Pai	Orientada, no Anel
Controle	Busca em Profundidade à Esquerda	Busca em Largura no nó, cont. em Prof. à Esq.
Processos	AND e OR, *n* por processador, sob demanda	1 Tipo, estático 1 por processador
Paralelismo	AND, para retrocessos	OR, sob demanda

Tabela 2: Comparação dos Modelos *Backup* e *Kabu-Wake*

5 - Testes e Resultados

Foram realizadas simulações de cada modelo com 2, 4, 8 e 16 processadores e o de referência seqüencial **PROLOG** com 1 processador, para cada um dos 4 programas **PROLOG** selecionados e descritos a seguir. Observe que os resultados obtidos no processo de simulação estão ligados aos valores atribuídos aos tempos das operações básicas na máquina alvo e, portanto, as análises realizadas são restritas à mesma.

Programa Interseção

Este programa relaciona os elementos comuns a duas listas. As listas utilizadas no evento de simulação resultaram em um conjunto com 3 elementos, redundando no mesmo número de soluções.

O modelo *Backup*, por explorar a modalidade *AND* de paralelismo, bastante restrita neste programa, obteve um fraco desempenho registrando em todos os eventos (soluções ou fim) um tempo sempre superior em pelo menos 40% ao do seqüencial. Um ponto decisivo certamente advém do fato de este modelo seguir à risca a árvore do seqüencial e possuir *overhead* com a manutenção de processos e

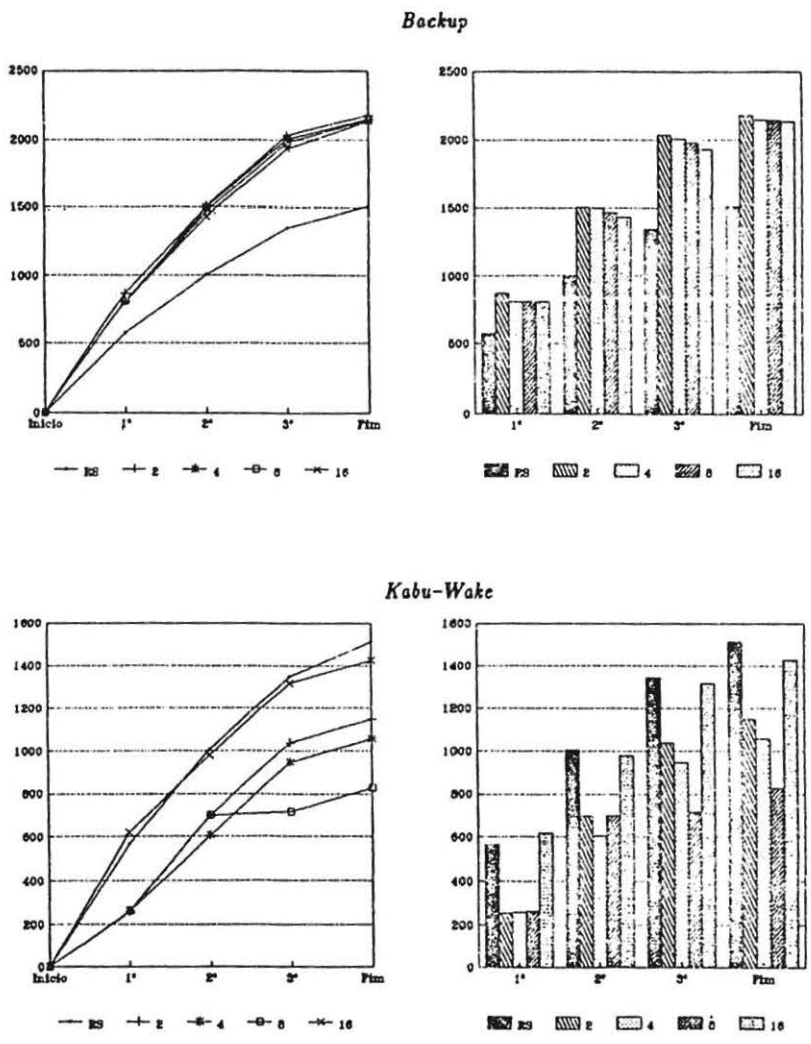


Gráfico 1: Tempo para obtenção de soluções e término do processamento
- Interseção -

comunicações. Como não pode explorar o paralelismo, os tempos de *overhead* simplesmente somaram-se ao do processamento normal.

Por sua vez, seria esperado que o modelo *Kabu-Wake* tivesse pouco espaço para expandir-se devido às cláusulas possuirem poucas alternativas e de estas serem predominantemente recursões ou fatos. Entretanto, superando as espectativas, o modelo obteve um bom desempenho neste programa sendo de 23 a 54% mais rápido que o seqüencial. A conclusão obtida é que o modelo ao dividir a árvore permitiu que os ramos que levam às soluções fossem explorados simultaneamente, mostrando ser, neste caso específico, bastante hábil na exploração de paralelismo, mesmo em condições mínimas.

Programa Paper

Este programa foi extraído de **[Conery 87]** e implementa um Banco de Dados Relacional, onde estão armazenadas informações sobre artigos diversos. São obtidas por este programa um total de 6 soluções.

O modelo *Backup*, apesar ter pouco espaço, beneficia-se da pouca profundidade da árvore de busca associada, uma vez que as metas a serem processadas são todas cláusulas sem corpo (fatos). Apesar de não apresentar bom desempenho com 2 processadores (entre 10 e 164% pior), com 4 ou mais processadores o seu desempenho melhorou muito, sempre à partir da segunda solução. Nestas condições, o modelo foi aproximadamente 11% mais rápido que o seqüencial. Apenas a primeira solução foi obtida em tempo 49% superior à do seqüencial.

O modelo *Kabu-Wake*, por sua vez, ao explorar o paralelismo *OR*, consegue tirar bastante proveito das características da árvore de busca associada a este programa. Sua vantagem se torna mais evidente na obtenção da primeira solução, que é a última dos demais modelos apresentados, obtida rapidamente (devido à busca em largura promovida pelo paralelismo *OR* em uma cláusula) enquanto que simultaneamente os demais processadores progridem em busca de novas soluções. O modelo foi entre 40 e 82% mais veloz que o seqüencial a cada evento.

Programa Densidade

O programa densidade fornece quais os pares de países, dentre os de sua lista, em que a população do primeiro país seja superior ao dobro da população do segundo país e que a área do primeiro país seja inferior à metade da área do segundo país. É gerado um sucesso para cada par de países encontrado e uma falha ao término da busca. Este programa possui complexidade da ordem do número de países ao quadrado (n^2), sendo portanto polinomial e, certamente, grande consumidor de recursos computacionais. Este fato atrai a atenção para estudos de otimização do tempo de processamento. Para a lista de países utilizada no programa são obtidas 3 soluções.

Tudo indica que o modelo *Backup* encontrou dificuldades em explorar paralelismo nesse programa devido a problemas de inicialização da sua árvore de processos, pois se sua primeira solução foi de 42 a 80% pior que o seqüencial, no término esta marca caiu para apenas de 12 a 33% pior, mostrando boa recuperação. Provavelmente, se o programa contivesse mais dados, o modelo superaria o seqüencial e certamente, devido à sensibilidade de PROLOG à ordem de declaração das cláusulas, qualquer mudança desta mudaria o desempenho do modelo, sendo possível obter marcas melhores. Ficou claro, contudo, que o fator determinante do mau

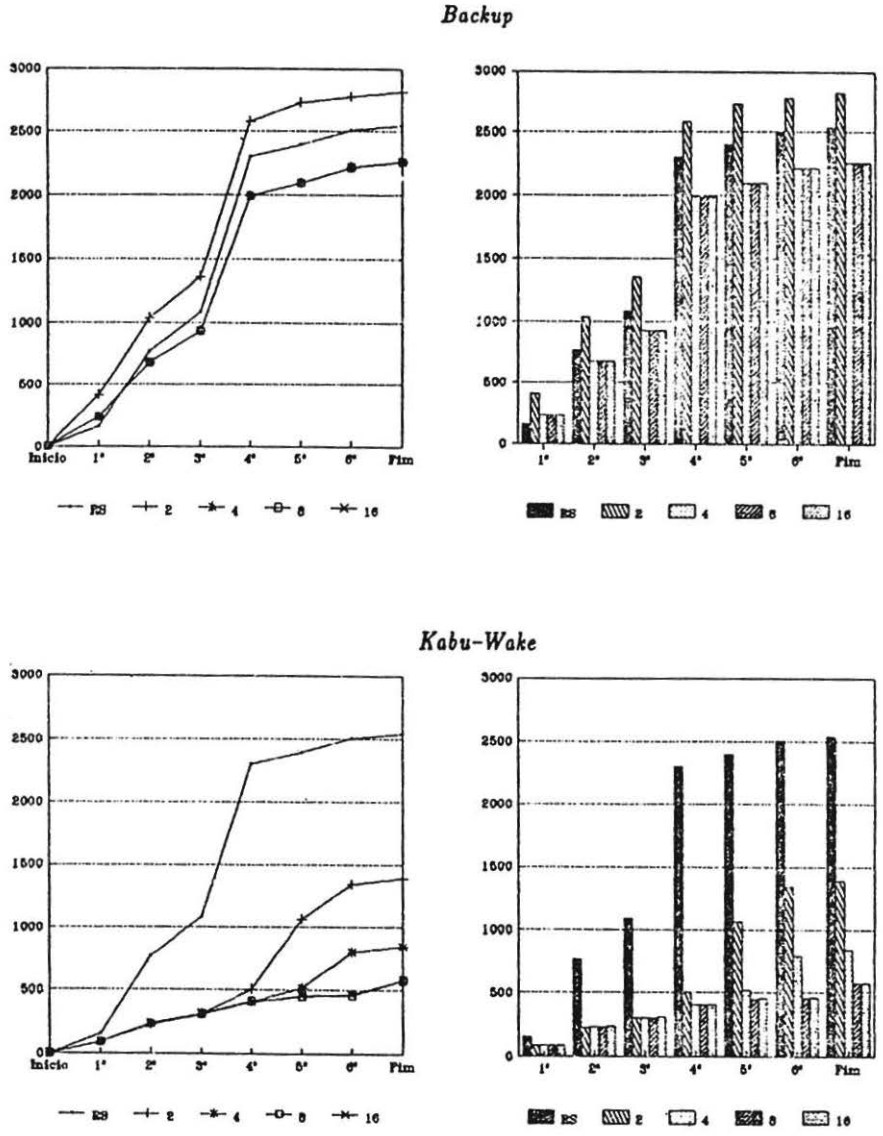
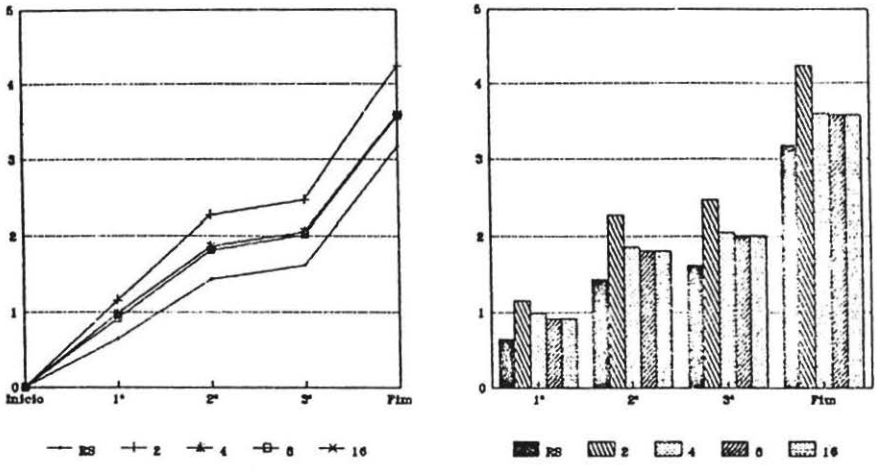


Gráfico 2: Tempo para obtenção de soluções e término do processamento
- Paper -

Backup



Kabu-Wake

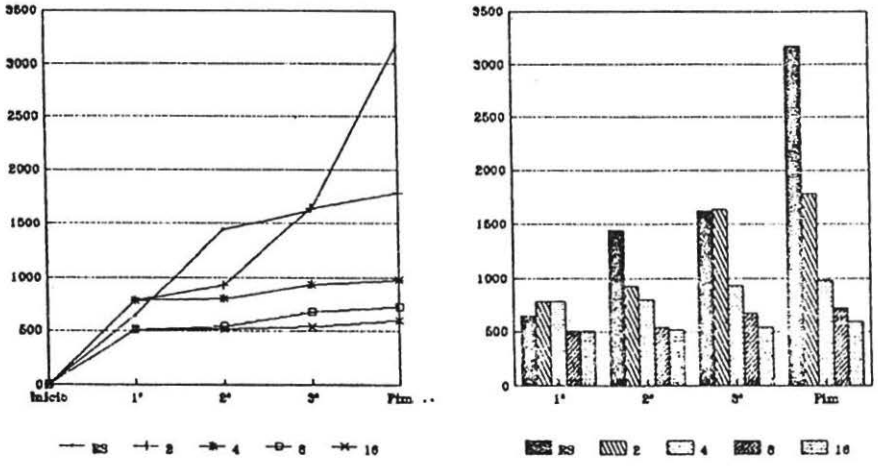


Gráfico 3: Tempo para obtenção de soluções e término do processamento
- Densidade -

desempenho do modelo foi o alto *overhead* registrado (44%), que supera a diferença entre *Backup* e Seqüencial.

Assim como o modelo *Backup*, o *Kabu-Wake* teve problemas com a inicialização do programa, mas apenas com 2 e 4 processadores. Os melhores resultados, obtidos com 8 e 16 processadores, situaram-se entre 20% melhor no início e 81% melhor ao final. O pior foi obtido pela simulação com 2 processadores que ficou entre 20% pior no início e 43% melhor que o seqüencial ao final.

Programa Coloração

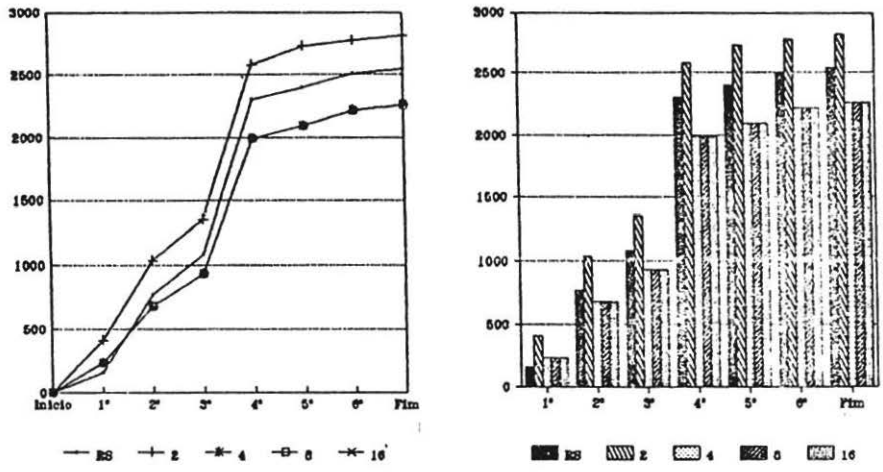
Este programa foi extraído de [**Casanova 87**] e determina quais as possíveis combinações de determinadas cores que poderiam colorir uma área plana dividida em regiões sem que duas dessas regiões vizinhas tenham a mesma cor. É gerado um sucesso para cada combinação encontrada e uma falha ao término das combinações possíveis. Igualmente ao programa de densidade, possui complexidade polinomial. No problema proposto a área é dividida em 4 regiões e dispõe-se de 3 cores para preenchê-las. Existem 6 combinações de cores que satisfazem às condições do problema.

O paralelismo *AND* presente no programa está todo contido na cláusula que define o mapa. Existem dependências entre variáveis, entretanto são possíveis diversas combinações de metas que minimizam o problema. O paralelismo *OR* também está presente nas declarações das combinações de cores. A simplicidade deste programa ajuda a combinar as diferentes modalidades de paralelismo de modo a se obter um bom desempenho.

Para o modelo *Backup* o único ponto desfavorável neste programa advém do fato de o modelo criar processos *OR* no mesmo processador do processo *AND* que é o pai deles. É criada desta forma uma séria limitação na exploração de paralelismo pelo modelo neste programa, que só não é pior porque estes criam processos *AND* em outro processador para resolver os fatos. Com 2 processadores o modelo foi pior que o seqüencial, mas com maior número de processadores foi pior apenas na primeira solução (5%), e oscilou entre 1 e 7% melhor da segunda em diante. Vale notar que devido às características da árvore gerada e à política de utilização de processadores pelo modelo, o mapeamento da árvore nescessita de 3 a 7 processadores. Qualquer número de processadores superior à este será desnecessário e inútil, obrigando que as simulações com 8 e 16 processadores obtenham sempre o mesmo tempo, o que ocorreu inclusive com 4 processadores, indicando que 4 tenha sido o número máximo de processadores usado pelo modelo.

O modelo *Kabu-Wake* é muito beneficiado pela estrutura do programa, pois seu trabalho consistirá em duplicar o corpo da declaração do mapa através do preenchimento das variáveis das primeiras metas, e depois em verificar a existência de fatos equivalentes às últimas. Com 2 e 4 processadores o modelo encontrou alguma dificuldade na obtenção da primeira solução, provavelmente devido ao grande número de combinações possível, com tempos 28 e 18% piores, respectivamente. Os demais tempos situaram-se entre 30 e 86% melhores. Com 8 e 16 processadores, o comportamento previsto acima torna-se evidente, com algum volume de processamento até a primeira solução e a seguir as demais sendo obtidas em intervalos curtíssimos.

Backup



Kabu-Wake

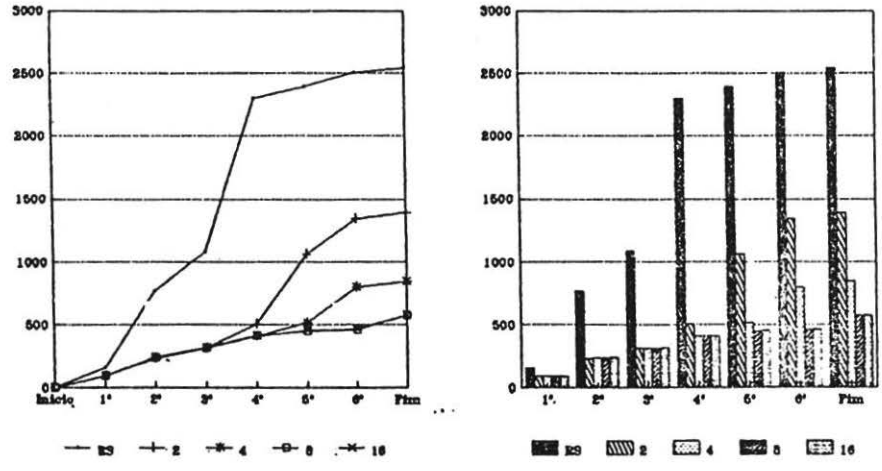


Gráfico 4: Tempo para obtenção de soluções e término do processamento - Coloração

5.1 - Avaliação do Modelo *Backup*

Em dois programas (interseção e densidade) o desempenho foi inferior ao do seqüencial e nos programas em que conseguiu ser mais veloz que o seqüencial, só o foi à partir da segunda solução e por um fator máximo de 13%. Os desempenhos nos programas seguiram a seguinte ordem, do melhor para o pior:

1° Paper
2° Coloração
3° Densidade
4° Interseção

No que se refere ao uso de processadores, o modelo mostrou-se bastante limitado, devido ao tempo de processamento normalmente tender a uma constante com o aumento do número de processadores. Isto indica que o modelo tem distribuições de sub-árvores ótimas (para si) e que atingindo estas distribuições não as modifica. O modelo se adapta ao número de processadores apenas quando este é menor que o ideal para si, deixando ociosos os processadores que excederem a esse número, levando o modelo a ter sempre o mesmo desempenho para qualquer número de processadores superior ao seu ótimo. Como conseqüência da constância do tempo de processamento, a ociosidade tem crescimento proporcional ao número de processadores. Os programas paper e coloração (a partir de 4 processadores) e densidade (a partir de 8) ilustram bem esse comportamento.

Como pontos deficientes apresentados pelo modelo enfatizamos a limitação no uso de processadores e adicionamos:

- a manutenção de processos, que é normalmente muito onerosa;
- grande *overhead*, que oscilou entre 20 e 45% do processamento útil;
- a criação de processos filhos no mesmo processador do pai, que atrapalha a exploração do paralelismo e obriga o uso de escalonadores nos processadores, incrementando o *overhead*;
- a criação de processos *and* para fatos, absolutamente desnecessária;
- a limitação do paralelismo imposta pelos processos *or*, que criam um novo filho apenas quando o atual termina; e
- a comunicação bloqueante, fator de restrição da exploração do paralelismo.

Embora os dois últimos pontos acima sejam usados pelos autores para controlar o paralelismo e evitar uma possível explosão no número de processos, os resultados mostram que houve excesso de precaução, pelo menos no que se refere aos programas testados. Para o modelo tornar-se competitivo é necessário que algumas dessas restrições sejam relaxadas.

Um fator que pode ser considerado favorável ao modelo é este manter a estratégia de busca PROLOG. Embora possa ser também considerada como um inibidor de paralelismo, esta estratégia é a mais conhecida pelos programadores, que a usam em diversos programas, produzindo respostas erradas quando a mesma não é seguida. É discutível se este fator realmente constitui vantagem.

Realmente, o modelo *Backup*, um dos primeiros a serem propostos, revelou-se não ser um modelo competitivo. É provável que apresente em alguns programas, mais particulares e maiores, melhor desempenho sem entretanto exibir valores expressivos de desempenho. Com algumas alterações para incrementar a exploração de paralelismo é possível que para uma classe maior de programas o modelo possa obter melhor desempenho e justificar a sua utilização.

5.2 - Avaliação do Modelo *Kabu-Wake*

O modelo *Kabu-Wake* obteve, em todos os programas simulados, desempenho superior tanto ao do modelo *Backup* quanto ao do seqüencial. Em poucos eventos (temporários) este modelo foi superado e em 3 dos 4 programas chegou a ser 80% (ou 5 vezes) mais rápido que o seqüencial e no quarto foi 50% (ou 2 vezes) melhor. Desta forma pode-se considerar como satisfatório o desempenho do modelo. A classificação dos programas, segundo o melhor desempenho do modelo, foi a seguinte:

1° Coloração
2° Densidade
3° Paper
4° Interseção

O uso dos processadores por parte do modelo foi sempre razoavelmente eficiente, sempre proporcional à quantidade de trabalho disponível. A favor deste argumento está o fato de a ordem de desempenho acima ser idêntica à de maior percentual de processamento útil (ou menor tempo ocioso). Em muitas das simulações o processador que inicia o processamento trabalha perto de 100% do tempo e a utilização dos demais varia, às vezes em função da posição na rede em anel, às vezes aleatoriamente, tudo depende da árvore gerada. O único ponto invariável é que sempre existe pelo menos um processador servindo de produtor para os demais. Registrou-se apenas um caso, em interseção, onde o aumento do número de processadores decididamente atrapalhou o desempenho do modelo. Tal fato deveu-se, provavelmente, a uma pulverização dos objetivos, sendo estes excessivamente distribuídos. Desta forma o modelo passou a maior parte do tempo distribuindo objetivos, e deixando os processadores ociosos. Em algus outros casos, este problema foi localizado em um ou outro evento, normalmente com diferenças de tempo pouco significativas.

Um ponto que merece destaque como possível problema do modelo, sem que isso signifique que seja uma deficiência do mesmo, é o de ocorrerem adversidades com programas que foram escritos para rodarem particularmente em PROLOG. Como o modelo não obedece à ordem de avaliação desta linguagem, determinados programas fornecem soluções que não satisfazem ao enunciado do programa, mas no entanto estas falsas soluções seriam obtidas se pedidas novas soluções em PROLOG. O problema está em se esperar que seja obtida apenas a primeira solução, alcançada por busca em profundidade à esquerda, e que depois pare o processamento, o que para o modelo *Kabu-Wake* é impossível. Surge então um problema de compatibilidade com alguns programas existentes.

Outro ponto significativo, que pode ser considerado uma dificldade à utilização do modelo *Kabu-Wake*, é a sua arquitetura peculiar, que embora praticamente elimine a existência de *overhead* (pois a absoluta maioria da comunicação é realizada pelo co-processador e não existem processos), exige processadores e redes de comunicação de dados específicos e altamente complexos. Certamente seria possível adaptar este sistema em arquiteturas hipercúbicas de memória privativa (ex. rede de TRANSPUTERS), entretanto forçosamente seria introduzido algum *overhead*.

Como ponto forte do modelo (além de desempenho, baixo *overhead* e boa utilização de recursos), destaca-se a possibilidade de serem executados simultaneamente diversos programas diferentes, ou mesmo diversas consultas para um mesmo programa e aumentar o *throughput* do sistema. Devido às características do modelo, este tipo de utilização, na maioria dos casos, não deve provocar grande

desaceleração nos programas e certamente reduziria a ociosidade. É evidente que seria aconselhável a junção de programas apenas dispondo-se de muitos processadores e para programas que trabalhem com grande ociosidade. Esta possibilidade de unir programas muito provavelmente resultaria em menor tempo para obter-se todos os resultados que a soma dos tempos individuais dos programas, sendo portanto um modo de otimizar o tempo e o uso dos processadores.

O modelo *Kabu-Wake* demonstrou possuir diversas qualidades que podem ser muito bem aproveitadas na exploração de paralelismo em Programação Lógica. Apesar de necessitar de um *hardware* complexo e possivelmente inacessível para a maioria dos usuários, há à possibilidade deste modelo ser implementado em outras máquinas. No mínimo, pode-se usá-lo como base na confecção de outros modelos mais próximos do estágio atual de desenvolvimento do *hardware*. Certamente este modelo representa uma importante contribuição à execução paralela de Programas Lógicos.

5.3 - *Backup* x *Kabu-Wake*

No ítem desempenho o modelo *Kabu-Wake* foi amplamente superior ao *Backup*. Registrou melhores tempos, usou melhor os recursos disponíveis e teve menor *overhead*. Em termos de facilidade de implementação ambos são razoavelmente simples, demandando pouco trabalho para realizar a implementação de suas principais rotinas (aquelas que caracterizam o modelo). Neste ponto vale ressaltar que o modelo *Backup* é mais fácil de ser modificado, aceitando mais alterações que o *Kabu-Wake*, que é mais fechado. Uma outra vantagem do modelo *Backup* é a maior facilidade para adaptá-lo a um *hardware* qualquer, provavelmente provocando poucas mudanças em seu desempenho. A mesma tarefa se realizada para o modelo *Kabu-Wake*, certamente seria mais complexa e provocaria danos muito maiores ao seu desempenho. Mesmo assim, podemos concluir que o modelo *Kabu-Wake*, de um modo geral, é superior ao *Backup*, sem que isto signifique que seja uma melhor referência para estudos ou uma melhor base para implementações em *hardware*.

6 - Conclusão

O Simulador de Modelos Paralelos de Programação em Lógica revelou-se uma ferramenta bastante adeqüada e útil para o estudo e a análise de modelos. Mesmo com poucos recursos (não é obrigatório o uso de multi-processadores), é possível realizar a simulação de programas de razoável complexidade, que poderá crescer em função dos recursos disponíveis (principalmente memória).

Os resultados apresentados mostraram-se coerentes e, na sua maioria, muito próximos do esperado. Mesmo onde os resultados representavam surpresa, sempre foi possível verificar a veracidade dos mesmos, lembrando-nos que nossas estimativas, por um mero detalhe, podem estar erradas. A complexidade dos modelos combinada à existente nos programas está, muitas vezes, além da nossa capacidade de realizar análises, tornando obrigatório o uso de ferramentas adeqüadas. A realidade apontada pelo simulador torna-se mais amena, e está sempre dentro da nossa capacidade de compreensão. Os dados obtidos pelo simulador são tratáveis de diversas maneiras, sendo possível realizar as mais complexas e detalhadas análises.

Referências Bibliográficas

[Bianchini 89] Ricardo G. Bianchini "EXECUÇÃO PARALELA DE PROGRAMAS LÓGICOS". *Projeto Final de Graduação*, UFRJ, 1989.
[Bianchini 90] Ricardo G. Bianchini "UM NOVO MODELO DE EXECUÇÃO PARALELA DE PROGRAMAS LÓGICOS". *Tese de Mestrado*, COPPE/UFRJ, 1990.
[Casanova 87] A. M. Casanova, F. A. C. Giorno e A. L. Furtado "PROGRAMAÇÃO EM LÓGICA E LINGUAGEM PROLOG". Edgard Blucher, 1987.
[Cohen 85] J. Cohen "DESCRIBING PROLOG BY ITS INTERPRETATION AND COMPILATION". CACM - v. 28, nº 12, Dec 1985.
[Conery 87] John S. Conery "PARALLEL EXECUTION OF LOGIC PROGRAMS". Kluwer Academic Publishers, 1987.
[Costa 91] Ruy Marinho da Costa "SIMULAÇÃO DE MODELOS PARALELOS DE PROGRAMAÇÃO EM LÓGICA". *Tese de Mestrado*, COPPE/UFRJ, 1991.
[DeGroot 84] D. DeGroot "RESTRICTED AND PARALLELISM. *Proceedings of the International Conference on Fifth Generation Computer Systems*, (Tokio, Japão), 1984, pp. 471-478.
[Dutra 88a] Inês de Castro Dutra "IMPLEMENTAÇÃO DE UMA MÁQUINA VIRTUAL PROLOG - TRADUÇÃO E EXECUÇÃO DE PROGRAMAS". *Tese de Mestrado*, COPPE/UFRJ, 1988.
[Dutra 88b] Inês de C. Dutra, Ricardo G. Bianchini, Leila M. Eizirik e Cláudio L. de Amorim "EM DIREÇÃO A UMA ESTAÇÃO PROLOG DE ALTO DESEMPENHO - TRADUÇÃO DE PROGRAMAS". *V Simpósio Brasileiro de Inteligência Artificial*. Natal, RN, Outubro de 1988.
[Fox 88] Geoffrey C. Fox (et ali) "SOLVING PROBLEMS ON CONCURRENT PROCESSORS" VOL I. Prentice Hall, 1988.
[Furukawa 82] K. Furukawa, K. Nitta e Y. Matsumoto "PROLOG INTERPRETER BASED ON CONCURRENT PROGRAMMING". *Proceedings of the First International Logic Conference*, (Faculté des Sciences de Luminy, Marseille, France, Sept.), 1982, pp. 38-44.
[Gregory 87] Steve Gregory "PARALLEL LOGIC PROGRAMMING IN PARLOG". Addison-Wesley, 1987.
[Hermenegildo 86] Manuel V. Hermenegildo "AN ABSTRACT MACHINE FOR RESTRICTED AND-PARALLEL EXECUTION OF LOGIC PROGRAMS". *Proceedings of the 3rd Int'l. Conference on Logic Programming*. Springer-Verlag, 1986, pp. 25-39.
[Hogger 84] C. J. Hogger "INTRODUCTION TO LOGIC PROGRAMMING". Academic Press, 1984.
[Jacquet 87] J. M. Jacquet "A GUIDED TOUR THROUGH PARALLELISM IN LOGIC PROGRAMMING", National Fund for Scientific Reserch, Institut d'Informatique - F.N.D.P., 1987.
[Lima 86] Priscila M. V. Lima "CONSIDERAÇÕES SOBRE UM SISTEMA PROLOG PARA AMBIENTE DE MULTIPROCESSAMENTO". *VI Congresso da Sociedade Brasileira de Computação*, Anais (Volume I), Recife - Olinda, Jul 1986.
[Lima 87] Priscila M. V. Lima "IMPLEMENTAÇÃO DE COMPILADORES PROLOG". *Tese de Mestrado*, COPPE/UFRJ, 1987.
[Lipovski 85] G. J. Lipovski e Manuel V. Hermenegildo "B-LOG: A BRANCH AND BOUND METHODOLOGY FOR THE PARALLEL EXECUTION OF LOGIC PROGRAMS". *Proceedings of the 1985 International Conference on Parallel Processing*, Agosto de 1985.
[Peterson 85] James L. Peterson, Abraham Silberschatz "OPERATING SYSTEM CONCETPTS". Addison Wesley, 1985.
[Stone 87] Harold S. Stone "HIGH-PERFORMANCE COMPUTER ARCHITECTURES". Addison Wesley, 1987.
[Sohma 85] Yukio Sohma, K. Saroh, K. Kumon, H. Masuzawa e A. Itashiki "A NEW PARALLEL INFERENCE BASED ON SEQUENTIAL PROCESSING". *Proceedings of the IFIP TC 10 Working Conference on Fifth Generation Computer Architetures 85*, (July 15-18), 1985, pp. 3-14.
[Warren 77] David h. D. Warren "IMPLEMENTING PROLOG - COMPILING LOGIC PROGRAMS". Vols 1,2, DAI Research Reports nº 39, 40, Department of Artificial Intelligence, University of Edinburgh, May 1977.
[Warren 80] David H. D. Warren "LOGIC PROGRAMMING AND COMPILER WRITING". Software - Pratice and Experience, v 10, pp. 97 - 125, 1980.
[Warren 83] David H. D. Warren "AN ABSTRACT PROLOG INSTRUCTION SET". Technical Note 309, SRI International, AI Center, Computer Science and Technology Division, 1983.
[Wise 87] Michael J. Wise "PROLOG MULTIPROCESSORS". Prentice Hall, 1987.