MPA - MÁQUINA PROLOG ASSOCIATIVA

Malena O. Hor-Meyll [1]
Raul Q. Feitosa [2]
Cláudio L. de Amorim [3]

Resumo

As arquiteturas convencionais de computadores, baseadas no modelo de von Neumann, não se mostram adequadas para o desenvolvimento de um ambiente eficiente para a linguagem Prolog, fundamentada em mecanismos de unificação e retrocesso automático. Este trabalho propõe a arquitetura de um acelerador, baseado em memória associativa, para ser interligado a uma máquina virtual Prolog desenvolvida na COPPE/UFRJ. A arquitetura proposta aumenta a eficiência da máquina virtual explorando o paralelismo da memória associativa na realização do mecanismo de retrocessso e desrefereciamento. O impacto no desempenho da máquina virtual decorrente da nova arquitetura foi simulado e avaliado utilizando programas de teste simples encontrados na literatura.

Abstract

Conventional computer architectures based on Von Neumann's model do not efficiently support the Prolog language, based on unification and automatic backtracking mechanisms. This work presents the architecture of an accelerator based on associative memory to be connected to a virtual Prolog machine which has been developed at COPPE/UFRJ. The accelerator architecture increases the virtual machine's efficiency by exploring the content addressable memory's parallelism to implement the backtracking and dereferencing mechanism. The impact on the virtual machine's performance was evaluated by simulation, using simple test programs found in the literature.

[1]MSc (PUC/RJ - 1992); áreas de interesse: Arquiteturas não Convencionais, Inteligência Artificial, Processamento Paralelo; Pesquisadora da COPPE/Sistemas - UFRJ (RHAE /CNPQ); e-mail: malena@rio.cos.ufrj.br.
[2]M.E. (ITA - 1983), Dr. Ing. (Universidade Erlanger - Nürnberg - 1988); áreas de interesse: Arquiteturas não Convencionais, Sistemas Paralelos, Modelagem; Professor Assistente da PUC-RJ; e-mail: raul@gsc.ele.puc-rio.br.
[3]MSc (COPPE - 1979), PhD (Imperial College - 1984); áreas de interesse: Supercomputação e Processamento Paralelo; Professor Adjunto da COPPE/Sistemas - UFRJ; e-mail: amorim@rio.cos.ufrj.br.

COPPE/Sistemas - Universidade Federal do Rio de Janeiro
Caixa Postal 68511 - CEP 21945-970 - Rio de Janeiro - RJ

Pontifícia Universidade Católica do Rio de Janeiro
Caixa Postal 38063 - CEP 22453-900 - Rio de Janeiro - RJ

1 - Introdução

Originalmente os computadores foram projetados para o processamento numérico. Entretanto, à medida que seu uso foi se diversificando, novas filosofias no desenvolvimento de sistemas foram emergindo, como forma de atender às particularidades de cada classe de aplicação. Uma dessas filosofias introduziu as linguagens *declarativas*, destacando-se dois ambientes de programação hoje bastante difundidos - a programação funcional e programação em lógica. Ambos estão baseados no processamento simbólico, exigindo novos modelos de execução, bem diferentes daqueles concebbidos para o cálculo numérico

Nos ambientes de programação em lógica, a linguagem Prolog [Cloc84] tem sido reconhecida como uma poderosa ferramenta em aplicações de inteligência artificial, bancos de dados, linguagem natural, robótica e sistemas especialistas.

A arquitetura clássica de von Neumann, orientada para o processamento de linguagens imperativas, revela-se ineficiente para a implementação da linguagem Prolog, baseada em mecanismos de unificação e retrocesso automático. Este fato motivou várias frentes de pesquisa em torno de arquiteturas não convencionais que dessem suporte a nível de "hardware" às primitivas da linguagem Prolog, permitindo o desenvolvimento de um ambiente eficiente de programação.

Este trabalho está inserido nesta linha de pesquisa ao propor o projeto de uma arquitetura baseada em memória associativa para a execução eficiente da linguagem Prolog. Esta proposta constitui uma extensão da máquina virtual Prolog descrita pela primeira vez por Warren [Warr77], que foi implementada pelos grupos de Inteligência Artificial e Processamento Paralelo da COPPE/UFRJ, como parte do projeto da "Estação Prolog Seqüencial" [Bian88]. A arquitetura resultante da integração da memória associativa à máquina de Warren foi denominada **Máquina Prolog Associativa - MPA**.

O processamento simbólico caracteriza-se fundamentalmente por operações de busca, comparação e ordenação. Neste contexto o tempo de acesso à memoria é mais determinante do desempenho do que nas aplicações de processamento numérico. Visando reduzir o "gargalo" existente entre processador e a memória, na execução de programas Prolog, arquiteturas baseadas em memória associativa tem sido alvo de pesquisas mais recentes [Naga88] [Robi85] [Robi86]. As memórias associativas se caracterizam por permitirem o acesso aos dados pelo conteúdo e por realizarem operações em paralelo sobre um conjunto de células de memória [Chis89].

A arquitetura proposta baseia-se no mecanismo de retrocesso paralelo projetado para a ASCA [Naga88], uma arquitetura seqüencial especializada e dedicada à execução Prolog utilizando memória associativa. A ASCA propõe um modelo de controle baseado em uma informação semântica - a profundidade de inferência - que permite fazer o acesso às informações de dados e controle armazenadas na memória associativa, prescindindo completamente de manipulação de endereços. O controle por profundidade de inferência foi incorporado à máquina virtual Prolog desenvolvida na COPPE/UFRJ para operar com a nova arquitetura. Embora as informações de controle permaneçam armazenadas em uma memória convencional, todas as informações relativas aos dados do programa passaram a ser armazenadas na memória associativa.

Embora a motivação inicial para a utilização de memória associativa tenha sido o aumento de eficiência decorrente do mecanismo de retrocesso paralelo, o acesso aos dados pelo conteúdo, aliado ao controle por profundidade de inferência, trouxe inúmeras simplificações à máquina original - eliminação de pilhas e registradores - que também contribuíram para o aumento de desempenho.

2 - A Máquina Virtual Prolog - PLM

Muitas pesquisas foram feitas no sentido de se conseguir um ambiente eficiente para a execução de programas em Prolog [Dobr85] [Uchi84] [Yoko84] [Taki84] [Mura84] [Naka85]. [Cive89] [Camp84]. A PLM ("Prolog Language Machine") - máquina virtual Prolog proposta por Warren em 1977 se destacou por apresentar pela primeira vez um sistema com compilação em separado. Até então todos os ambientes desenvolvidos eram interpretados. A Figura 1 ilustra a arquitetura da máquina virtual Prolog implementada na COPPE/UFRJ [Lima87] [Dutr88], baseada na proposta de Warren:

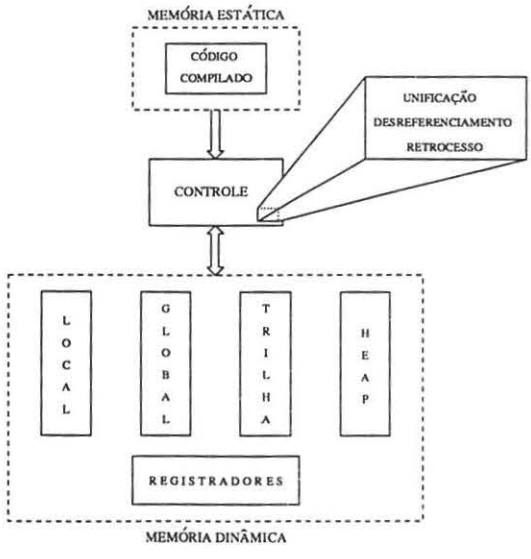

Figura 1 - A Máquina Virtual Prolog

A memória da máquina virtual está organizada em duas áreas principais: estática e dinâmica. A memória estática (leitura) armazena o código compilado, onde o controle busca as instruções a serem executadas. A memória dinâmica (leitura e escrita) é constituída pelas pilhas local, global, trilha e heap e pelos registradores. A execução das instruções do programa compilado se baseia na chamada dos mecanismos de unificação, desreferenciamento e retrocesso. Estes mecanismos manipulam as pilhas e os

registradores, que por sua vez suportam toda parte de armazenamento dinâmico de dados e informações de controle.

Para cada objetivo a ser resolvido durante a execução de um programa Prolog é alocada dinamicamente na pilha local uma área de memória, denominada moldura local, que armazena informações de controle (ambiente), ponteiros para as pilhas global e trilha e valores associados a variáveis locais resultantes do processo de unificação (variáveis locais são aquelas que só aparecem fora de termos compostos). A pilha global armazena os valores associados às variáveis que aparecem como subtermo de um termo composto (por exemplo, uma lista). O mecanismo de desreferenciamento atua tanto na pilha local quanto global na busca do valor associado a uma determinada variável. A trilha armazena os endereços das variáveis que devem voltar a ser indefinidas por ocasião de um retrocesso. A heap é utilizada para armazenar pares de ponteiros associados à representação dos termos compostos. A memória dinâmica compreende ainda um conjunto de registradores que contêm ponteiros de topo das quatro pilhas além de informações de controle.

3 - A Máquina Prolog Associativa - MPA

A Figura 2 apresenta o diagrama em blocos da MPA. O acelerador da MPA, composto pela memória associativa e seu controle (vide bloco tracejado na Figura 2) aumenta a eficiência dos mecanismos de retrocesso e desreferenciamento.

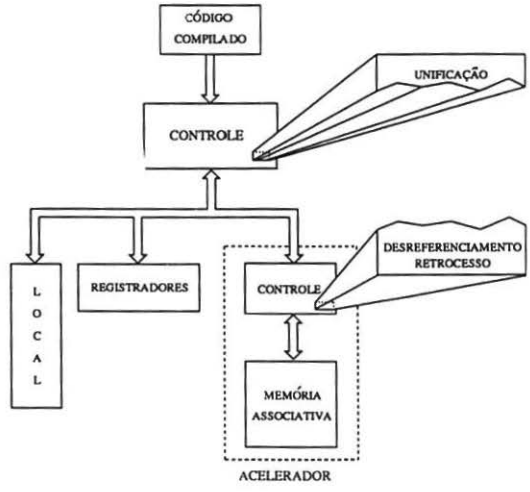

Figura 2 - A arquitetura da MPA

Comparando-se as Figuras 1 e 2, visualizam-se as simplificações efetuadas na máquina virtual em decorrência da incorporação da memória associativa. Três pilhas de execução foram eliminadas: trilha, global e heap. Todas as associações referentes às variáveis locais e globais passaram a ser armazenadas na memória associativa, o que justifica a eliminação da pilha global. O controle por profundidade de inferência, aliado a característica da memória associativa de manipular os dados pelo conteúdo, permitiu suprimir a trilha e também contribuiu para eliminação da heap. Finalmente, todos os registradores de gerenciamento das pilhas global, trilha e heap foram eliminados. Apenas dois novos registradores foram criados para dar suporte ao mecanismo de controle por profundidade de inferência. Com isso o número total de registradores na implementação com a memória associativa é menor. Algumas instruções do código compilado foram suprimidas, tendo sido igualmente eliminados parâmetros de algumas instruções.

A memória associativa introduz uma nova concepção no modo de armazenar os dados. Não há uma célula específica para armazenar o valor associado a cada variável. Quando uma associação é feita, armazena-se o par unificado na memória associativa (em qualquer posição, uma vez que a recuperação dos dados é feita pelo conteúdo). Junto com o par unificado acrescenta-se uma informação que identifica o objetivo no qual ocorreu a determinada associação. A memória associativa é capaz de realizar operações em paralelo em várias células de memória. Sendo assim, quando ocorre um retrocesso, todas as associações relativas ao objetivo para o qual se tentar nova alternativa podem ser desfeitas simultaneamente.

Nas seções 3.1 e 3.2 são discutidos sucintamente os mecanismos de retrocesso e desreferenciamento na arquitetura MPA. Uma descrição mais detalhada pode ser encontrada em [Hor92].

3.1 - Retrocesso na Arquitetura MPA

O mecanismo de retrocesso, através do controle por profundidade de inferência, pode ser realizado com muito mais eficiência, explorando-se as facilidades de busca e escrita em paralelo que a memória associativa provê. O controle por profundidade de inferência permite a eliminação da trilha e de várias operações durante a execução. O tempo consumido pelo retrocesso deixa de crescer com o número de associações a serem desfeitas.

A execução de um programa Prolog pode ser vista como uma busca em profundidade ao longo de uma *árvore de inferência*. A árvore de inferência de um programa adaptado de [Naga88] é apresentada na Figura 3.

Programa "conterrâneo":

1 - nascimento (joão,josé,eua).

2 - nascimento (pedro,léo,rj).

3 - país (eua).

4 - cidade (rj).

5 - conterrâneo (U,V) :- nascimento (U,V,W), cidade (W).

6 - ?conterrâneo (X,Y).

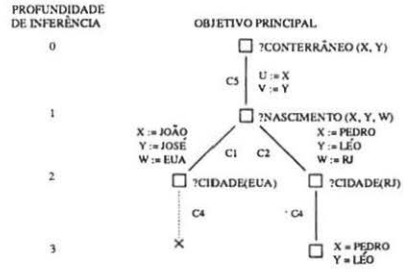

Figura 3 - Árvore de Inferência do Programa "conterrâneo"

Os níveis da árvore de inferência são numerados seqüencialmente a partir da raiz que recebe o número zero. O número associado a cada nível da árvore é denominado *profundidade de inferência*. Cada nó da árvore está associado a um objetivo. O nó da raiz corresponde ao objetivo da cláusula consulta (objetivo principal) e os nós restantes a objetivos intermediários. Cada objetivo de uma determinada profundidade de inferência invoca um conjunto de cláusulas. As cláusulas invocadas em um determinado nível tem o mesmo predicado do objetivo que as invocou. A árvore contém um nó para cada cláusula invocada.

Quando uma variável é associada a um valor, a *ligação* (par unificado variável/valor) é armazenada na memória associativa junto com a profundidade de inferência corrente. Quando ocorre um retrocesso em uma dada profundidade de inferência, efetua-se uma busca na memória associativa por todas as ligações associadas àquela profundidade. Todas as células que responderem à busca são desinstanciadas simultaneamente, através do *indicador livre/ocupada*, presente em todas as células. A desinstanciação paralela de um conjunto de células consiste em escrever-se "1" (livre) no indicador das respectivas células.

Na profundidade de inferência "1" os argumentos X e Y do objetivo "?conterrâneo (X,Y)", que estão associados ao nível de inferência "0", são ligados aos argumentos correspondentes, U e V, da cláusula 5, que estão associados à profundidade de inferência "1". As ligações:

$$(U, 1) \leftrightarrow (X, 0)$$

$$(V, 1) \leftrightarrow (Y, 0)$$

são armazenadas na memória associativa junto com a profundidade de inferência corrente "1". A Figura 4 representa o estado da memória associativa após a primeira unificação de variáveis.

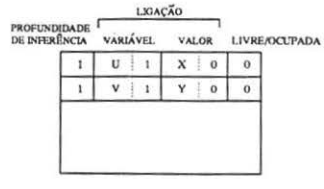

Figura 4 - Estado da Memória Associativa Após a Primeira Unificação

Na profundidade de inferência "2", a invocação da cláusula 1 para atender ao objetivo "?nascimento(X,Y,W)" resulta nas ligações:

$$(X, 0) \leftrightarrow \text{joão}$$
$$(Y, 0) \leftrightarrow \text{josé}$$
$$(W, 1) \leftrightarrow \text{eua}$$

que são armazenadas na memória associativa junto com a profundidade de inferência "2", de acordo com a Figura 5.

PROFUNDIDADE DE INFERÊNCIA	LIGAÇÃO				LIVRE/OCUPADA
	VARIÁVEL		VALOR		
1	U	1	X	0	0
1	V	1	Y	0	0
2	X	0	JOÃO		0
2	Y	0	JOSÉ		0
2	W	1	EUA		0

Figura 5 - Estado da Memória Associativa após a Segunda Unificação

No estágio seguinte, correspondente à profundidade de inferência "3", a invocação da cláusula 4 para atender ao objetivo "?cidade(eua)" resulta em falha ao se tentar unificar:

$$\text{eua} \leftrightarrow \text{rj}$$

Como não há mais cláusulas com predicado "cidade" a serem tentadas deve-se retornar ao objetivo anterior mais recente para tentar nova alternativa (retrocesso profundo). Neste caso retorna-se ao objetivo "?nascimento (X,Y,W)" e tenta-se a segunda cláusula com o mesmo predicado, a cláusula 2. Antes de invocá-la, no entanto, deve-se desfazer todas as ligações criadas pela invocação da cláusula 1 (operação de desinstanciação). Para isto basta transformar em células livres todas as células cujo campo "profundidade de inferência" tenha valor "2", pois nesta profundidade é que se realizaram as unificações relativas à cláusula 1. Em apenas dois ciclos (busca e escrita), a memória associativa toma o aspecto da Figura 6 (equivalente ao da Figura 4):

PROFUNDIDADE DE INFERÊNCIA	LIGAÇÃO				LIVRE/OCUPADA	
	VARIÁVEL		VALOR			
1	U	1	X	0	0	
1	V	1	Y	0	0	
2	X	0	JOÃO		1	CÉLULAS LIVRES
2	Y	0	JOSÉ		1	
2	W	1	EUA		1	

Figura 6 - Estado da Memória Associativa após o Retrocesso

O tempo consumido pela desinstanciação não cresce com o número de ligações a serem desfeitas, como ocorre na arquitetura original. O tempo consumido pela desinstanciação depende apenas da diferença entre a profundidade de inferência corrente e a profundidade de inferência do ponto de retorno. Para exemplificar, suponha-se que uma unificação falha na profundidade de inferência corrente *m*. Se o

retrocesso consistir em tentar uma nova cláusula na profundidade de inferência *n*, todas as células cujo campo profundidade de inferência seja maior ou igual a *n* devem se tornar livres. Neste caso, o tempo consumido na desinstanciação é de (*m*-*n*+1) ciclos, portanto independente do número de variáveis desinstanciadas em cada nível de profundidade.

3.2 - Desreferenciamento na Arquitetura MPA

Na máquina virtual Prolog original, cada variável está associada a uma determinada posição (endereço) na pilha correspondente. Quando duas variáveis livres são unificadas, uma delas recebe um ponteiro para a outra. A medida que o programa vai sendo executado, criam-se cadeias de variáveis ligadas. A operação de desreferenciamento de variáveis, que consiste em percorrer uma cadeia de associações (ligações), nas pilhas local e/ou global, envolve uma série de manipulações de endereços e testes para verificar se o ponteiro é para a pilha local ou global, o que compromete a eficiência de sua execução. Como a memória associativa busca os dados pelo conteúdo, o desreferenciamento pode ser realizado sem manipulação de endereços e sem necessidade de testes, uma vez que tanto variáveis locais quanto globais são armazenadas na memória associativa.

Para desreferenciar uma variável é fornecido à memória associativa o tipo (global ou local), a profundidade de inferência e um índice que identifica a variável na cláusula. Se o resultado da busca for um valor constante, a operação está encerrada e o valor encontrado é devolvido como resultado do desreferenciamento. Se no entanto, o resultado da busca for uma variável, o procedimento se repete, mas agora com novos valores de tipo, profundidade de inferência e índice relativo a variável retornada. É importante notar que se o tipo for *variável*, o desreferenciamento prossegue sem haver necessidade de testes para verificar se a variável é global ou local, como ocorria na realização do desreferenciamento pela arquitetura original. O próprio valor retornado é utilizado diretamente na próxima busca. Na arquitetura PMA não há necessidade da variável estar associada ao valor *indefinido* para ser considerada livre, como ocorria na arquitetura original. A simples ausência de uma associação envolvendo a variável na memória associativa é suficiente para considerá-la livre.

4 - Alterações na Máquina Virtual Original

Parte do controle da máquina virtual Prolog original foi transferido para o acelerador baseado em memória associativa, que passou a realizar integralmente as operações de desinstanciação e desreferenciamento de variáveis. A eliminação da pilha global, da trilha e da heap implicou em alterações no controle da máquina virtual, pois todas as operações envolvidas no gerenciamento destas pilhas foram eliminadas. Por outro lado, novas operações foram acrescentadas para se fazer o controle por profundidade de inferência. Resume-se a seguir as principais alterações efetuadas na máquina original. A descrição completa destas alterações é apresentada em [Hor92].

- Redução do número de informações armazenadas na moldura local: os ponteiros para a trilha e para a pilha global são eliminados. Apenas uma nova informação de controle é acrescentada na moldura local: a profundidade de inferência.

-Redução do número de registradores: eliminação de cinco registradores [4] associados às pilhas global, trilha e heap. Dois novos registradores [5] foram acrescentados para se fazer o controle por profundidade de inferência .

-Redução do número de instruções e do número de parâmetros de algumas instruções.

5 - Avaliação de Desempenho

Nesta seção são avaliados os méritos da arquitetura proposta comparando seu desempenho com o da arquitetura original. Com este objetivo definiu-se um conjunto de operações básicas a nível de "hardware", de tal forma que cada instrução do código compilado pudesse ser expressa por uma seqüência destas operações. Para avaliar o tempo de execução de um programa, foi introduzido em ambas as máquinas virtuais um monitor que totaliza o número de vezes que cada operação básica é executada.

Com base no estágio atual da tecnologia foi atribuído um tempo de execução para cada operação básica. Os tempos de execução foram normalizados de modo que a operação básica mais simples (acesso a registrador) tivesse tempo de execução igual a uma unidade [Moto90] [Inte89] [Mudg91]. O tempo de execução de um programa em cada arquitetura foi estimado como sendo a soma dos produtos entre o número de vezes que cada operação básica foi executada e o seu tempo de execução. O conjunto de operações básicas e seus correspondentes tempos de execução estão na Tabela 1.

Operação	Tempo de Execução
acesso a registrador	1
acesso à memória	3
operação lógica	1
operação aritmética	1
acesso à memória associativa	1

Tabela 1 - Tempos de Execução Normalizados

As operações lógicas, de adição e subtração, apresentam pequenas diferencas em relação ao tempo de acesso a registrador [Moto90] [Inte89]. Assim, atribuiu-se também a estas operações tempo de execução unitário.

Admite-se o uso de memória "cache" com tempo de acesso igual a duas vezes o tempo de acesso a registrador [Henn90]. O tempo em um "miss" é de 4 a 20 vezes o tempo de acesso à "cache" em um "hit" [Ston87]. Para efeito desta análise admite-se que o tempo de acesso à memória, quando ocorre um "miss", é

[4] V1, X1, VV1, TR e heap

[5] IDV: profundidade de inferência corrente, IDX: profundidade de inferência do objetivo pai

a média destes valores, ou seja, 12 vezes o tempo de acesso à "cache". Considera-se para o "hit ratio" um valor médio igual a 0.95. Em função do tempo de acesso à "cache" (thit), à memória principal (tmiss) e do "hit ratio" (h) pode-se calcular o tempo médio de acesso à memória (tm):

$$tm = h \times thit + (1 - h) \times tmiss$$

- h = 0.95
- thit = 2
- tmiss = 12 × thit = 24

$$tm \approx 3$$

A memória associativa proposta para a arquitetura MPA é do tipo totalmente paralelo. Admitindo-se que a mesma tecnologia é empregada na fabricação da "cache" e da memória associativa, supõe-se que o tempo de acesso, é igual para ambas as arquiteturas. Todas as operações de leitura e escrita na memória associativa são precedidas por uma operação de busca ("match"). A memória "cache" também realiza uma operação de busca, antes de fazer um acesso propriamente dito, para verificar se o dado procurado está armazenado na mesma. No entanto, o tempo consumido pela operação de busca na "cache" já está embutido no tempo de acesso. Como o tempo de acesso à memória "cache" é duas vezes o tempo de acesso a registrador, atribui-se às operações de acesso à memória associativa tempo igual a 1.

5.1 - Testes Realizados

Como os benefícios da memória associativa decorrem principalmente do paralelismo nas operações de retrocesso, mediu-se sua eficiência na execução de um programa não determinístico proposto em [Naga88]. Este programa permite alterar o número de retrocessos realizados durante a execução e o número de associações a serem desfeitas em cada retrocesso através da variação de certos parâmetros. Além disso, foram utilizados programas clássicos para avaliação de desempenho encontrados na literatura: ordenação, composição de listas e testes numéricos.

a) Programa não Determinístico

O programa utilizado para avaliar a eficiência do mecanismo de retrocesso, extraído de [Naga88], é apresentado a seguir:

Programa "p1":

```
p1( f( X1 ), f( X2 ), ..., f( Xm ), a ) :-
p2( X1, X2, ..., Xm, a ). p1( g( X1 ), g( X2 ), ..., g( Xm ), b ).
p2( f( X1 ), f( X2 ), ..., f( Xm ), a ) :- p3( X1, X2, ..., Xm, a ).
p2( g( X1 ), g( X2 ), ..., g( Xm ), b ). pn-1( f( X1 ), f( X2 ), ..., f( Xm ), a ) :-
pn( X1, X2, ..., Xm, a ). pn-1( g( X1 ), g( X2 ), ..., g( Xm ), b ).
pn ( g( X1 ), g( X2 ), ..., g( Xm ), b ).
?p1( X1, X2, ..., Xm, a ).
```

O programa "p1" possibilita avaliar o desempenho da MPA em relação à máquina original em função do número *m* de associações a serem desfeitas em cada operação de retrocesso e em função do número *n* de retrocessos. A eficiência aumenta a medida que estes parâmetros crescem pois a máquina original desfaz as associações relativas a cada retrocesso seqüencialmente, enquanto a MPA o faz em paralelo. O gráfico da Figura 7 apresenta a variação do *Speedup* em função de *m* e *n*.

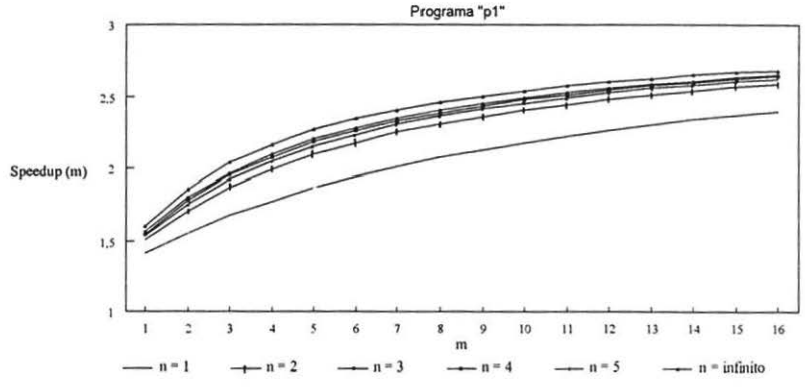

Figura 7 - *Speedup* em Função do Número *m* de Associações Desfeitas em cada Retrocesso e do Número *n* de Retrocessos para o Programa "p1".

A medida que o número de associações a serem desfeitas em cada retrocesso cresce, mais rápida se torna a MPA em relação à PLM. Isto se dá porque na MPA todas as associações são desfeitas em paralelo e em consequência, o tempo consumido pela operação de desinstanciação não cresce com *m*. De acordo com o Figura 7, para qualquer valor de *n*, a razão entre os tempos relativos totais de execução da máquina original e da máquina virtual baseada em memória associativa cresce com o aumento de *m*. Observa-se, contudo, que quanto maior o valor do parâmetro *n*, tanto maior é a variação do *Speedup* em função do parâmetro *m*. Conclui-se que, a medida que o número *n* de retrocessos cresce, maior é a sensibilidade do tempo de execução total para variações do número *m* de associações desfeitas em cada retrocesso. Traçando-se uma linha vertical no gráfico da Figura 7 pode-se avaliar o comportamento do *Speedup* a medida que *n* varia, para um valor constante de *m*. A medida que *n* cresce, a razão entre os tempos de execução aumenta. Assim como o gráfico em função apenas de *m*, o valor do *Speedup* em função de *n* também tende para um valor constante.

b) Programas Genéricos

Com o objetivo de avaliar os benefícios da memória associativa em programas reais, mediu-se o *Speedup* para um conjunto de programas genéricos [Corb89] [Lima87] [Cloc84] [Dutr88] [Naga88] freqüentemente utilizados na literatura para avaliação de desempenho. Os resultados são apresentados na Tabela 2.

Programa	*Speedup*
append	1.65
avô	1.48
fatorial	1.66
hanoi	1.62
membro(1)	1.36
membro(20)	1.69
parser	1.67
query	1.48
quicksort	1.91
rainhas	1.84
reverse	1.79

Tabela 2 - *Speedup* para um Conjunto de Programas Genéricos

Nota-se pela Tabela 2 que em todos os casos obtém-se *Speedup* superior a 1. Na maior parte dos casos, a flutuação nos valores de *Speedup* é função de uma maior ou menor utilização da trilha ou da heap pela arquitetura original. O programa "avô", por exemplo, não possui termos compostos (portanto não utiliza a heap) e realiza um número reduzido de retrocessos. Este comportamento se reflete no *Speedup* e pode ser avaliado através da comparação, por exemplo, com o *Speedup* encontrado no programa "rainhas", cuja execução se baseia em um algoritmo de tentativa e erro, e que realiza por isso grande número de retrocessos. Durante a fase de testes observou-se, para um mesmo programa, uma tendência do *Speedup* crescer a medida que o volume de processamento era maior. Isto está ilustrado nos resultados do programa "membro". O teste com uma lista de apenas 1 elemento apresentou *Speedup* de 1.36, ao passo que este valor aumentou para 1.69, quando o teste foi feito com uma lista de 20 elementos.

6 - Conclusões

Neste trabalho, foi proposto um acelerador baseado em memória associativa para ser interligado a uma máquina virtual Prolog. Obteve-se um aumento significativo de desempenho pela exploração do paralelismo da memória associativa na realização do mecanismo de retrocesso.

A introdução da memória associativa possibilitou uma série de simplificações na máquina virtual, que também contribuiu para o aumento de eficiência. Entre estas, destaca-se a redução do número de pilhas e registradores. Face a estas simplificações, propôs-se um conjunto mais simples de instruções .

Para avaliar o impacto no desempenho decorrente da introdução da memória associativa, um conjunto de programas clássicos para análise de desempenho ("benchmarks" padrões) foram executados na máquina original Prolog e na máquina virtual baseada em memória associativa. Foram elaboradas versões

especiais de ambas as máquinas para análise de desempenho, que incorporam no "software" primitivas para computar os tempos relativos de execução.

Os resultados dos testes de desempenho se mostraram amplamente favoráveis à utilização da memória associativa. Os melhores resultados foram obtidos a partir de programas não determinísticos, devido sobretudo ao mecanismo de retrocesso paralelo. Entretanto, o aumento de eficiência em programas genéricos foi também bastante expressivo. Com base nos resultados da avaliação de desempenho, conclui-se que a memória associativa apresenta grande potencial para acelerar a execução de programas Prolog.

Algumas questões relacionadas a arquitetura proposta merecem ainda ser estudadas: determinação do número e largura das palavras da memória associativa de modo que a máquina possa executar eficientemente programas reais; determinação de mecanismos eficientes utilizando memória convencional que garantam a operação da máquina nas situações em que a carga exceda a capacidade da memória associativa.

7 - Referências Bibliográficas

[Bian88] BIANCHINI, R., I. C. DUTRA, P. M. V. LIMA, L. M. R. Eizirik e C. L. AMORIM
Em direção a uma Estação Prolog: Implementação e Avaliação de Desempenho da Máquina Virtual
V Simpósio Brasileiro de Inteligência Artificial, Natal - RN, Novembro 1988.

[Camp84] CAMPBELL, J. A.
Implementations of Prolog
Ellis Horwood Limited, John Willey & Sons, 1984.

[Chis89] CHISVIN, L., R. J. DUCKWORTH
Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM
Computer, July 1989, pp.51-64.

[Cive89] CIVERA, Pierluigi, Gianluca PICCININI, Maurizio ZAMBONI
Implementations Studies for a VLSI Prolog Processor
IEEE MICRO, Fevereiro 1989, pp. 10-23.

[Cloc84] CLOCKSIN, William F., Christopher S. MELLISH
Programming in Prolog (Second Editiont)
Springer-Verlag, 1984.

[Corb89] CORBUCCI, Dante
LispLog: Uma Linguagem para a Programação Funcional e para a Programação Lógica
Tese de Mestrado, PUC/RJ, Setembro, 1989.

[Dobr85] DOBRY, T. P., A. M. DESPAIN e Y. N. PATT
Performance Studies of a Prolog Machine Architecture
Proceedings of 12th International Symposium on Computer Archictecture, Boston, MA, June 1985, pp.180-190.

[Dutr88] DUTRA, Inês de C.
Implementação de uma Máquina Virtual Prolog - Tradução e Execução de Programas
Tese de Mestrado, COPPE/UFRJ, Novembro 1988.

[Henn90] HENNESSY, J., D. PATTERSON
Computer Architecture: A Quantitative Approach
Morgan Kauffman Publishers Inc, Palo Alto, 1990.

[Hor92] HOR-MEYLL, Malena Osorio
Uma Arquitetura baseada em Memória Associativa para Suporte a uma Máquina Virtual Prolog
Tese de Mestrado, PUC/RJ, Setembro, 1992.

[Inte89] Intel Corporation
Microprocessors and Peripheral Handbook, vol. 1, 1989.

[Lima87] LIMA, Priscila M. V.
Implementação de Compiladores Prolog
Tese de Mestrado, COPPE/UFRJ, Marco 1987.

[Moto90] Motorola Semiconductors
Enhanced 32-bit Microprocessor User's Manual
Prentice Hall, 1990.

[Mura84] MURAKAMI, K., T. KAKUTA, R. ONAI
Architectures and Hardware Systems: Parallel Inference Machine and Knowledge Base Machine
Proceedings of the International Conference on Fifth Generation Computer Systems, ICOT, November 1984, pp.118-36.

[Mudg91] MUDGE, Trevor N., R. B. BROWN, W. P.Birmingham, J. A. DYKSTRA, A. I. KAYSSI, R. J. LOMAX, O. A. OLUKOTUN and K. A. SAKALLAH
The Design of a Microsupercomputer
Computer, vol 24, no. 1, January 1991, pp. 57-64.

[Naga88] NAGANUMA, J., T. OGURA, S. YAMADA, T. KIMURA
High-Speed CAM-Based Architecture for a Prolog Machine (ASCA)

IEEE Transactions on Computers, vol 37, no.t11, November 1988, pp. 1375-1383.

[Naka85] NAKAZAKI, R., A. KONAGAYA, S. HABATA, H. SHIMAZU, M. UMEMURA, M. YAMAMOTO, M. YOKOTA, T. CHIKAYAMA
Design of a High-speed Prolog Machine (HPM)
Proceedings of 12th Symposium on Computer Architecture, Boston, MA, Junho 1985, pp. 191-197.

[Robi85] ROBINSON, Phillip
The SUM: An AI Coprocessor
BYTE, vol. 10, June 1985, pp.t169-180.

[Robi86] ROBINSON, I.
A Prolog Processor Based on a Pattern Matching Memory Device
Proceedings of 3rd International Conference on Logic Programming, July 1986, pp. 172-179.

[Ston87] STONE, H.
High Performance Computer Architectures
Addison-Wesley, 1987.

[Taki84] TAKI, K., M. YOKOTA, A. YAMAMOTO, H. NISHIKAWA, S. UCHIDA, H. NAKASHIMA e A. MITSUISHI
Hardware Design and Implementation of a Personal Sequential Inference Machine (PSI)
Proceedings of the International Conference on Fifth Generation Computer Systems, ICOT, November 1984, pp.t398-409.

[Uchi84] UCHIDA Shunichi, T. YOKOI
Sequential Inference Machine: SIM - Progress Report
Proceedings of the International Conference on Fifth Generation Computer Systems, ICOT, November 1984, pp.t58-69.

[Warr77] WARREN, David H. D.
Implementing Prolog - Compiling Predicate Logic Programs
Research Reports no 39, no 40, Dept of Artificial Inteligence, University of Edinburgh, 1977.

[Yoko84] YOKOTA, M., A. YAMAMOTO, K. TAKI, H. NISHIKAWA, S. UCHIDA, K. NAKAJIMA e M. MITUSI
A Microprogrammed Interpreter for the Personal Sequential Inference Machine
Proceedings of the International Conference on Fifth Generation Computer Systems, ICOT, November 1984, pp.t410-418.