# UMA ANÁLISE DOS MODELOS QUE INTEGRAM O PARALELISMO E/OU NA PROGRAMAÇÃO EM LÓGICA

**Cristiano André da Costa**[1]

Universidade Católica de Pelotas
Escola de Informática
Rua Félix da Cunha, 412 - CEP 96010-000
Pelotas, RS, Brasil
cac@atlas.ucpel.tche.br

**Cláudio Fernando Resin Geyer**[2]

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Caixa Postal 15064 - CEP 91591-970
Porto Alegre, RS, Brasil
geyer@inf.ufrgs.br

RESUMO

Este trabalho estuda as várias formas de integração do paralelismo E e do paralelismo OU na Programação em Lógica. O trabalho engloba os principais sistemas, a saber: ANDORRA-I, ACE, ROPM e IDIOM. Ao final, é feito um estudo comparativo das propostas, a título de contribuição para as pesquisas na área de Processamento Paralelo e Programação em Lógica. São destacadas características como linguagem e arquitetura suportadas, análise em tempo de compilação e compatibiliade com semântica de Prolog. A partir do estudo comparativo, é possível determinar as características mais importantes de ambientes para exploração do paralelismo E e OU na Programação em Lógica.

Palavras-chaves: Paralelismo E, Paralelismo OU, Programação em Lógica, Processamento Paralelo.

ABSTRACT

This work shows the various forms of integrating Or-parallelism and And-parallelism in Logic Programming. This work cover the most popular systems: ANDORRA-I, ACE, ROPM and IDIOM. At the end, a comparison among the proposals is presented, contributing to Parallel Processing and Logic Programming areas. The most important features are supported language and machine, compile-time analysis and semantics compatibility with Prolog. The comparative study shows the best characteristics for AND/OR parallel exploitation of Logic Programming.

Keywords: AND-Parallelism, OR-Parallelism, Logic Programming, Parallel Processing.

---

[1]Professor na Universidade Católica de Pelotas (UCPel/RS), Técnico em Eletrônica (ETFPel/RS, 1990), Bacharel em Ciência da Computação (UCPel/RS, 1994), Mestre em Ciência da Computação (UFRGS/RS,1998). E-mail: cac@atlas.ucpel.tche.br

[2]Professor na Universidade Federal do Rio Grande do Sul (UFRGS/RS), Engenheiro Mecânico (UFRGS/RS, 1978), Mestre em Ciência da Computação (UFRGS/RS, 1986), Doutor em Informática (Université Joseph Fourier, Grenoble, França, 1991). E-mail: geyer@inf.ufrgs.br

# UMA ANÁLISE DOS MODELOS QUE INTEGRAM O PARALELISMO E/OU NA PROGRAMAÇÃO EM LÓGICA


RESUMO

Este trabalho estuda as várias formas de integração do paralelismo E e do paralelismo OU na Programação em Lógica. O trabalho engloba os principais sistemas, a saber: ANDORRA-I, ACE, ROPM e IDIOM. Ao final, é feito um estudo comparativo das propostas, a título de contribuição para as pesquisas na área de Processamento Paralelo e Programação em Lógica. São destacadas características como linguagem e arquitetura suportadas, análise em tempo de compilação e compatibiliade com semântica de Prolog. A partir do estudo comparativo, é possível determinar as características mais importantes de ambientes para exploração do paralelismo E e OU na Programação em Lógica.



Palavras-chaves: Paralelismo E, Paralelismo OU, Programação em Lógica, Processamento Paralelo.



ABSTRACT

This work shows the various forms of integrating Or-parallelism and And-parallelism in Logic Programming. This work cover the most popular systems: ANDORRA-I, ACE, ROPM and IDIOM. At the end, a comparison among the proposals is presented, contributing to Parallel Processing and Logic Programming areas. The most important features are supported language and machine, compile-time analysis and semantics compatibility with Prolog. The comparative study shows the best characteristics for AND/OR parallel exploitation of Logic Programming.



Keywords: And Parallelism, Or Parallelism, Logic Programming, Parallel Processing.

## 1 INTRODUÇÃO

O processamento paralelo tem sido utilizado como uma alternativa para o aumento de desempenho nos sistemas de computação uma vez que os processadores estão atingindo seus limites físicos, impostos pela tecnologia dos componentes eletrônicos utilizados em sua implementação. Outro fator que colabora para a exploração do paralelismo é o contínuo crescimento na complexidade das aplicações.

Para dar suporte ao paralelismo, existem dois grandes grupos de linguagens, a saber: as que utilizam o processamento paralelo de forma explícita, através de construções específicas para tal fim, e as que o fazem de maneira implícita, mantendo sua sintaxe original. O objetivo das linguagens que exploram o processamento paralelo implícito é a não re-implementação de todo o *software* utilizado quando da substituição de máquinas seqüenciais por paralelas.

Por outro lado, Programas em Lógica podem ser computadados seqüencialmente ou em paralelo sem alteração na sua semântica declarativa. A paralelização implícita é conseqüência de uma característica deste tipo de programação que pode ser expressa por: "programas = lógica + controle".

Um Programa em Lógica pode ser decomposto em uma parte lógica que especifica o significado declarativo do programa e uma parte de controle que provê um meio de executar esta especificação ([HOG 84], [STE 86]). Dentre as linguagens que implementam a programação em lógica destaca-se Prolog.

Prolog é uma linguagem de programação baseada na programação em lógica com cláusulas de Horn, proposta por Kowalski nos anos 70, e implementada pela primeira vez por Alain Colmerauer na Universidade de Marselha. Atualmente, existem várias implementações de Prolog para multiprocessadores[1], ver [SWE 95], [KER 94], [YAM 92] e [COS 96].

Existem basicamente duas técnicas para a exploração do paralelismo implícito na programação em lógica, a saber: o paralelismo E que avalia concorrentemente metas de uma determinada cláusula, e o paralelismo OU que consiste em processar paralelamente cláusulas de um procedimento em determinada etapa da resolução de um programa.

Uma vez que as metas são executadas independentemente no Paralelismo E, é necessária a compatibilidade das ligações feita por cada ramo. Isto leva a dois modelos de execução diferentes. No Paralelismo E *Independente* somente metas que não compartilham variáveis são executadas em paralelo, enquanto que as outras metas são executadas seqüencialmente. No Paralelismo E *Dependente* a execução concorrente de metas que compartilham variáveis é possibilitada através de uma relação produtor / consumidor, isto é, o consumidor aguarda o produtor ter uma ou mais soluções disponíveis.

Importantes vantagens podem ser obtidas com a combinação do paralelismo E e OU na Programação em Lógica, conforme pode ser observado em [COS 96] e [KER 94]. Cresce o número de programas que podem ser beneficiados pelo paralelismo e diminui o espaço de busca de alguns programas, apresentando assim maior desempenho quando executados paralelamente.

Este artigo está organizado da seguinte forma: a seção 2 descreve sucintamente os principais modelos que integram o paralelismo E/OU na Programação em Lógica. Na seção 3 são feitas comparações entre as diversas propostas estudadas. Por fim, a seção 4, apresenta as conclusões do trabalho.

## 2 PROPOSTAS QUE INTEGRAM O PARALELISMO E/OU

Os sistemas que integram o paralelismo E/OU na Programação em Lógica podem ser divididos em três grupos principais:

---

[1]A paralelização da semântica de Prolog completo automática não é possível. Por exemplo, *pruning operators* e *side-effects* são seqüenciais ou não totalmente paralelizáveis;

- Sistemas que exploram o paralelismo E Dependente / OU; deste grupo foi estudado o Andorra-I ([COS 91a] e [COS 91b]);
- Sistemas que exploram o paralelismo E Independente / OU; representados no trabalho pelos sistemas ACE ([GUP 93] e [GUP 94]) e ROPM ([RAM 89] e [RAM 92]);
- Sistemas que exploram o paralelismo E Dependente, E Independente / OU; dentre os quais o sistema estudado é o IDIOM ([GUP 91]).

### *2.1 Andorra-I*

O Andorra-I é um sistema que explora o paralelismo E Dependente /OU e estabelece a execução inicialmente de metas determinísticas (metas que unificam com no máximo uma cláusula). O Andorra-I combina duas formas de paralelismo implícito e provê a capacidade das linguagens *flat commited-choice* [SHA 89].

Uma *flat language* é caracterizada por utilizar um conjunto fixo de predicados primitivos (principalmente testes de unificação e aritmética). Nestas linguagens uma cláusula pode realizar somente uma computação simples, especificada por uma conjunção de átomos com predicados primitivos, antes de fazer a escolha não-determinística (*commited-choice*).

Os programas no Andorra-I são executados por times de trabalhadores (grupos de processadores). Cada um destes times pode estar executando em uma fase determinística ou não-determinística, a saber:

- **Determinística**: Todas as metas determinísticas são candidatas para avaliação imediata. Esta fase acaba quando não existem mais metas determinísticas disponíveis ou quando uma meta falha. No primeiro caso, o time passa para a fase não-determinística. No segundo ocorrerá o *backtracking* para que um novo nodo OU seja explorado.
- **Não-determinísitica:** Se nenhuma meta determinística existe, a meta mais a esquerda (ou uma especificada pelo usuário) é avaliada. Um *choice-point* é criado, gerando vários nodos OU. O time atual explora um destes nodos. Caso existam mais times disponíveis, estes times podem explorar os outros nodos gerados.

Em um time de trabalhadores, um processador é o mestre e os outros são escravos. Sendo que:

a) cada time trabalha em um nodo OU diferente (na fase não-determinística e no backtracking só o mestre trabalha);

b) cada trabalhador no mesmo time trabalha no mesmo nodo OU, explorando o paralelismo E Dependente.

A quantidade de times determina o paralelismo OU máximo, enquanto que a quantidade de trabalhadores por time indica o paralelismo E máximo em um time.

O Andorra-I possui dois componentes principais: uma máquina (*engine*) e um pré-processador. A máquina do Andorra utiliza programas descritos em dois níveis: o código principal representando as definições de cláusulas e o código de determinação, que representa gráficos de decisão para cada procedimento (gerado pelo pré-processador).

A versão seqüencial do Andorra-I é 2.5 vezes mais lenta que o SICStus Prolog[2] [SWE 95] e a sobrecarga para o suporte de implementações paralelas é na média de 40%. Para programas que contém paralelismo OU e E, na versão paralela do Andorra-I podem ser obtidos desempenhos melhores do que se um único tipo de paralelismo for explorado.

[2]Ambiente de programação Prolog disponível comercialmente e amplamente difundido na comunidade acadêmica;

### *2.2 ACE*

O *And/Or-parallel Copying-based Execution of Logic Programs* (ACE) é um sistema Prolog que explora o paralelismo OU e E Independente. O sistema suporta a linguagem Prolog de uma maneira elegante pois recomputa metas quando explora o paralelismo E Independente, ao contrário dos outros modelos estudados que utilizam o compartilhamento das soluções obtidas paralelamente.

Quando utiliza-se o compartilhamento, é necessário que o conjunto de soluções das metas envolvidas sejam estáticas (quando executadas várias vezes as metas devem gerar as mesmas respostas). Porém isto só é verdadeiro quando as cláusulas utilizam lógica pura. Entretanto se for utilizada a recomputação o paralelismo E pode ser utilizado mesmo com *cuts* e *side-effects*. Além disso, é importante ressaltar que o uso de recomputação não causa diminuição no desempenho do sistema, pois só é utilizada pelo ACE nas mesmas situações empregadas por Prolog.

Para que a implementação da linguagem fosse possível sem alteração da semântica seqüencial de Prolog utilizou-se um modelo abstrato chamado Árvore de Composição (*Composition Tree*), *ou seja*, uma árvore contendo nodos OU e E. Os nodos OU representam as múltiplas cláusulas que unificam com uma meta, enquanto que os nodos E representam as várias submetas no corpo da cláusula sendo executada em paralelismo E Independente.

No ACE os processadores são organizados em times de forma análoga ao Andorra-I. O paralelismo E Independente é explorado entre processadores de um mesmo time, enquanto que o processamento OU é realizado entre times. Desta forma, um processador pertencente a um time comporta-se como um processador em sistemas puramente E, ao passo que, times de processadores comportam-se como processadores nos sistemas puramente OU.

Até o presente momento só está implementado o componente de paralelismo E Independente do ACE, chamado de &ACE. O &ACE apresenta um *overhead* de aproximadamente 10% sobre o SICStus Prolog. Os tempos de execução obtidos no &ACE são em geral melhores do que os obtidos pelo &- Prolog[3] [BUE 93], apesar do fato de que o &ACE contém uma implementação mais completa e permite *backtracking* total entre metas paralelas. Além disso, o sistema possui a vantagem de ter sido construído para trabalhar também com o paralelismo OU.

### *2.3 ROPM*

O ROPM (REDUCE-OR Process Model), muitas vezes chamado apenas como REDUCE-OR, é um modelo que combina o paralelismo E Independente e OU para a implementação de Prolog padrão. O sistema é capaz de gerar o paralelismo máximo disponível em programas Prolog. Além disso, todas as soluções de determinado problema são encontradas. O REDUCE-OR utiliza o compartilhamento das soluções obtidas pelo paralelismo E, ao invés da recomputação.

No REDUCE-OR cada regra é representada por um gráfico chamado *Data Join Graph* (DJG). Neste gráfico cada arco (*arc-relation*) denota um literal no corpo de uma cláusula, enquanto que os nodos (*node-relation*) armazenam as soluções parciais até aquele ponto. O objetivo do gráfico é mostrar a dependência dos dados entre literais no corpo de uma cláusula.

O ROPM começa executando a meta de mais alto nível. Após acontecer a unificação, as ligações de variáveis são inseridas no primeiro nodo do DJG e todos os arcos que originam-se neste nodo são disparados.

Em problemas com muitas soluções diferentes o REDUCE-OR cria um processo OU encarregado de encontrar todas as cláusulas que unificam com determinado nome de predicado. Podem ser encontrados dois tipos de cláusulas, a saber: *fatos*, o REDUCE-OR faz a unificação

[3]Sistema que explora o paralelismo E Independente.

do fato com o predicado e retorna as ligações como resposta; *regras,* o REDUCE-OR tenta inicialmente unificar a meta e a cabeça da cláusula. Se conseguir, um processo é disparado para executar o corpo da cláusula, retornando as ligações ao final.

Nodos em que entram dois ou mais arcos unem as soluções dos arcos para produzir uma solução que é inserida no nodo da junção. A junção de duas soluções obtidas pelo paralelismo E envolve essencialmente a união das duas soluções (compartilhamento), uma vez que os dois nodos E são independentes. Porém para que isto ocorra de forma eficiente alguns algoritmos foram propostos, dentre os quais destacam-se os algoritmos propostos em [RAM 92].

Quando uma solução está disponível para o último nodo ela é unificada (*back-unified)* com as variáveis da consulta que antecede o nodo, fornecendo assim as ligações para estas variáveis.

O desempenho obtido pelo ROPM quando comparado com o SBProlog ou o Quintus Prolog é considerado satisfatório [RAM 89], porém nos testes não foi utilizado o *cut,* uma vez que o REDUCE-OR não o suporta (o *cut* representa em média uma melhora de desempenho de 25%).

### *2.4 IDIOM*

O Integrated Dependent- Independent- and Or-parallel Model, ou simplesmente IDIOM, é a primeira estratégia de implementação que explora as três formas de paralelismo. Além de suportar a linguagem Prolog tradicional, o IDIOM suporta as linguagens *flat commited choice*, ao estilo do Andorra-I.

Assim como no sistema Andorra foi utilizado o princípio de execução imediata de metas determinísticas e a execução de metas não determinísticas por paralelismo OU. Além disso, foram empregadas as técnicas de execução paralela de metas independentes e o compartilhamento de soluções.

A combinação de soluções no IDIOM é feita através de um produto cartesiano (do inglês, *cross-product*). Neste esquema, supondo-se duas metas a serem executadas, $p$ e $q$, não é necessário que para cada nova solução de $p$, $q$ seja recomputado. O IDIOM explora o paralelismo OU quando uma meta unifica com mais de uma cláusula, e as resolvantes resultantes são executadas em paralelo.

O sistema IDIOM utiliza CGEs para expressar o paralelismo E Independente. Estas CGEs podem ser obtidas em tempo de compilação através de um pré-processador. A execução do IDIOM é constituída de três fases principais:

- fase de Paralelismo E Dependente ou *Dependent And-Parallel phase (*DAP);
- fase de Paralelismo E Independente ou *Independent And-Parallel phase* (IAP);
- fase de Paralelismo OU ou *Or-Parallel phase* (ORP).

Inicialmente o IDIOM executa na fase DAP todas as metas determinísticas em paralelo (incluindo aquelas dentro das CGEs). Quando não existirem mais metas determinísticas a serem executadas, a meta mais a esquerda é examinada para determinar se é uma meta com CGE. Em caso afirmativo o IDIOM entra na fase IAP, caso contrário entra na fase ORP.

Na fase ORP, as várias alternativas de uma cláusula são executadas em paralelismo OU. Após ocorrer a unificação da cabeça da cláusula em uma destas alternativas, o IDIOM retorna para a fase DAP.

Na fase IAP, por sua vez, ocorre a avaliação da CGE. Se verdadeira, as metas são executadas em paralelismo E Independente e será iniciada então a fase ORP, processando a meta mais a esquerda em paralelismo OU. Se a avaliação da CGE resultar falsa, a fase ORP é

iniciada imediatamente para que a meta mais a esquerda seja executada em paralelismo OU.

Quando a execução de uma meta nas fases ORP ou DAP tiver sucesso e se a meta for componente de uma CGE, é feito o cruzamento das soluções incrementalmente. A continuação da execução retorna para a fase DAP.

A conhecimento dos autores não está disponível uma implementação do IDIOM até o presente momento. Sendo assim, o estudo foi feito com base na proposta teórica.

## 3 COMPARAÇÃO ENTRE OS MODELOS ESTUDADOS

Nesta seção são apresentadas as comparações entre os modelos estudados. As comparações são na sua maioria qualitativas e estão dividas de acordo com características[4] consideradas importantes para um modelo de exploração de paralelismo E/OU na Programação em Lógica.

### *3.1 Semântica de Prolog seqüencial*

Uma característica que pode ser avaliada quanto aos sistemas de Prolog paralelos é o fato de seguir a semântica de Prolog. Muitos sistemas estudados, mesmo suportando a linguagem Prolog padrão, não executam programas da mesma forma. Dentre os sistemas estudados apenas o ACE mantém a semântica de Prolog. O Andorra-I e o IDIOM mudam esta semântica pois executam metas determinísticas primeiro.

Outra questão importante relativa a semântica de Prolog ocorre em sistemas que exploram o paralelismo E Independente. Sistemas como o ACE recomputam as metas à direita, de forma análoga a linguagem Prolog padrão. O REDUCE-OR e o IDIOM, por sua vez, utilizam uma semântica diferente de Prolog pois ao invés de recomputar as metas de determinada cláusula, eles compartilham as soluções.

Em geral para duas metas independentes $G_1$ e $G_2$ com $m$ e $n$ soluções respectivamente, o custo de computação pode diminuir de $m * n$ para $m+n$ quando compartilham-se as soluções através do produto cartesiano. Entretanto, para metas com pequenas granulosidades, o ganho do compartilhamento não compensa o custo do produto cartesiano. Além disso, a recomputação é melhor para metas independentes que contém *side-effects* e predicados extra-lógicos. O uso de compartilhamento ou recomputação não alteram os resultados do programa nem a semântica a nível de usuário.

### *3.2 Linguagem Suportada*

Outro fator que deve ser observado é o fato de suportar as características extra-lógicas (*pruning operators*, *side-effects*, etc.). Dentre os sistemas estudados o REDUCE-OR é o único a não suportar estas características.

Além disso, o Andorra-I e o IDIOM têm a opção de suportar também as características das *flat commited choice languages*.

### *3.3 Arquiteturas Suportadas*

Realizada uma análise de todos os sistemas estudados, é constatado que todos eles foram projetados para multiprocessadores[5]. O Andorra-I e o ACE suportam arquiteturas multiprocessadoras com memória compartilhada, como o *Sequent Symmetry*. Já o REDUCE-OR além de suportar estas arquiteturas, suporta também multiprocessadores com memória distribuída, como o Intel iPCS/2.

Alguns dos sistemas estudados podem ser executados em estações de trabalho, porém seqüencialmente.

---

[4] A ordem das características não é relevante;
[5] Excluindo o IDIOM que não foi implementado;

### *3.4 Baseados em processos versus baseados em processos leves*

Existem basicamente duas maneiras para a implementação de sistemas para a Programação em Lógica paralela, a saber: baseados em processos e baseados em processos leves (*threads*).

Em sistemas baseados em processos, como o REDUCE-OR, um processo é criado para cada meta encontrada durante a execução. Estes processos comunicam-se para a troca de ligações e informações de controle. Este tipo de abordagem é mais adequado para implementações em multiprocessadores com memória distribuída do que com memória compartilhada, uma vez que diferentes processos podem ser mapeados em diferentes processadores em tempo de execução.

Por outro lado, em sistemas baseados em processos leves múltiplas *threads* são criadas e executadas em paralelo, sendo que cada uma age de forma semelhante a uma WAM[6]. O ACE, o ANDORRA-I e o IDIOM são exemplos de sistemas desta categoria. Os sistemas baseados em processos leves são mais adequados para multiprocessadores com memória compartilhada. Além disso sistemas deste tipo mostraram ser eficientes e podem herdar todas as melhorias da compilação seqüencial.

### *3.5 Escalonamento*

O Escalonador é um dos componentes cruciais para um bom desempenho de sistemas Prolog paralelos. Por exemplo, em sistemas que suportam *cut*, o escalonador deve optar por realizar primeiro trabalhos que não estão no escopo de nenhum *cut*, uma vez que este *pruning operator* pode fazer com que o trabalho realizado possa se tornar inútil.

O Andorra-I e o ACE realizam escalonamento em duas fases, aproveitando os escalonadores já existentes para sistemas que exploram um tipo de paralelismo. Estes sistemas dividem os processadores em times e devem ser capazes de decidir quando processos devem migrar de um time para outro ou quando deve criar novos times. Uma vantagem importante do Andorra-I é que ele pode usar um dentre os vários escalonadores disponíveis para o Aurora [YAM 92].

### *3.6 Suporte à análise em tempo de compilação*

A análise em tempo de compilação é outra característica importante para a eficiência de um sistema paralelo para a Programação em Lógica. Ferramentas para este tipo de análise, baseadas em Interpretação Abstrata foram utilizadas em todos os sistemas que foram implementados.

O ACE faz análise de *sharing* e *freeness* para geração automática de CGEs em tempo de compilação. Estas características foram herdadas do sistema &- Prolog. O Andorra-I utiliza análise em tempo de compilação para detecção de determinância de metas. O ROPM, por sua vez, faz análise de dependência durante a compilação gerando os *Data Join Graphs*.

### *3.7 Overhead do Paralelismo*

Dentre os sistemas aqui apresentados, o &ACE e o REDUCE-OR apresentam *overheads* pequenos se comparados a ambientes Prolog seqüencias amplamente difundidos, como o SICStus Prolog e o Quintus Prolog.

O Andorra-I apresenta uma sobrecarga de 40% sobre o SICStus, enquanto que no &-ACE a sobrecarga é de apenas 10%. O REDUCE-OR quando comparado ao Quintus Prolog apresenta uma sobrecarga considerado satisfatória pelos seus criadores.

---

[6] Warren Abstract Machine (Máquina Abstrata proposta por Warren para execução de programas Prolog);

### *3.8 Desempenho Geral dos Sistemas*

O desempenho geral dos sistemas estudados foi considerado satisfatório. Muitas vezes foram obtidos *speedups* próximos ao limite teórico máximo, conforme pode ser observado em [YAN 93], [PON 94] e [RAM89].

Neste trabalho é muito difícil fazer uma comparação de desempenho entre os sistemas, uma vez que os testes foram obtidos de trabalhos publicados e não apresentavam os algoritmos. Configurações de arquiteturas e número de processadores utilizados também foram diferentes. Além disso, não foi possível realizar os testes no Instituto de Informática da UFRGS, uma vez que até o término do trabalho nenhuma arquitetura multiprocessadora suportada pelos sistemas estava disponível.

### *3.9 Quadro Comparativo*

Neste item é apresentado um quadro comparativo resumindo as principais características dos sistemas estudados para a exploração do paralelismo E/OU na Programação em Lógica. A **¡Error!Argumento de modificador desconocido.** apresenta o quadro comparativo.

Tabela **¡Error!Argumento de modificador desconocido.** Quadro comparativo dos sistemas que integram o paralelismo E/OU

| Características: | Andorra-I | ACE | REDUCE-OR | IDIOM |
|---|---|---|---|---|
| Paralelismo OU | X | X | X | X |
| Paralelismo E Dependente | X | | | X |
| Paralelismo E Independente | | X | X | X |
| Mantém semântica de Prolog seqüencial | | X | | |
| Recomputação de metas[7] | N/A | X | | |
| Compartilhamento de soluções[¡Error!Argumento de modificador desconocido.] | N/A | | X | X |
| Suporte a predicados extra-lógicos[8] | X | X | | N/A |
| Suporte à linguagens *flat commited choice* | X | | | X |
| Suporte à arquitetura com memória compartilhada[¡Error!Argumento de modificador desconocido.] | X | X | X | N/A |
| Suporte à arquitetura com memória distribuída[¡Error!Argumento de modificador desconocido.] | | | X | N/A |
| Sistemas baseados em processos | | | X | |
| Sistemas baseados em processos leves | X | X | | X |
| Escalonamento em duas fases | X | X | | |
| Suporte à análise em tempo de compilação | X | X | X | N/A |

N/A → Não aplicável

## 4 CONCLUSÕES

A Programação em Lógica é uma linguagem que pode ser paralelizada sem comprometer sua semântica declarativa. Para sua paralelização existem basicamente três modos: paralelismo OU, E Dependente e E Independente.

A combinação destas formas de paralelismo propõe três classes de sistemas, a saber:

[7]Somente para sistemas que exploram o paralelismo E Independente;
[8]Excluindo o IDIOM que não foi implementado;

modelos que integram o paralelismo E Dependente/OU; modelos que integram o paralelismo E Independente/OU; e modelos que integram os três tipos de paralelismo. Dentre os modelos do primeiro grupo, foi estudado o Andorra-I. Como representantes do segundo grupo, foram apresentados os sistemas ACE e REDUCE-OR. O IDIOM foi apresentado como modelo que integra os três tipos de paralelismo.

A partir das comparações, foi possível identificar características comuns aos sistemas paralelos de Programação em Lógica, bem como evidenciar aquelas consideradas importantes. Deste modo, como contribuição do trabalho temos a constatação das principais características para um sistema que integre o paralelismo E/OU na Programação em Lógica.

## REFERÊNCIAS BIBLIOGRÁFICAS


[BUE 93] BUENO, F. et al. **The And-Prolog Compiler System - Automatic Parallelization Tools for LP**. Madrid: Universidad Politécnica de Madrid, Jun. 1993. 40p. (Technical Report DIA/CLIP5/93.0).

[COS 91a] COSTA, V. S.; WARREN D. H. D.; YANG, R. The Andorra-I Engine: A Parallel Implementation of the Basic Andorra Model. In: INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING, 8, Aug. 1991. **Proceedings...** Cambridge: MIT Press, 1991. p.825-839.

[COS 91b] COSTA, V. S.; WARREN D. H. D.; YANG, R. The Andorra-I Preprocessor: Supporting Full Prolog on the Basic Andorra Model. In: INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING, 8, Aug. 1991. **Proceedings...** Cambridge: MIT Press, 1991. p.443-456.

[COS 96] COSTA, Cristiano A. **Um Estudo das Propostas que Integram o Paralelismo E/OU na Programação em Lógica**. Porto Alegre: CPGCC-UFRGS, 1996. 64p. (Trabalho Individual, 493).

[GUP 91] GUPTA, G.; COSTA, V. S.; YANG, R. IDIOM. Integrating Dependent and-, Independent and-, and Or-parallelism. In: INTERNATIONAL SYMPOSIUM ON LOGIC PROGRAMMING, 4, Oct. 1991. **Proceedings...** Cambridge: MIT Press, 1991. p.152-166.

[GUP 93] GUPTA, C.; HERMENEGILDO, M. PONTELLI, E.; COSTA, V. S. **ACE: And/Or-parallel Copying-based Execution of Logic Programs**. Las Cruces: New Mexico State University, 1993. 16p. (Technical Report).

[GUP 94] GUPTA, G.; HERMENEGILDO, V.; PONTELLI, E.; COSTA, V. S. ACE: and/or-parallel copying-based execution of logic programming. In: INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING, 1994. **Proceedings...** Italy: MIT Press, 1994. p.93-109.

[HOG 84] HOGGER, C. J. **Introduction to Logic Programming**. London: Academic Press, 1984.

[KER 94] KERGOMMEAUX, J.C.; CODOGNET, Philippe. **Parallel Logic Programming Systems.** Grenoble: Universite Joseph Fourier-Grenoble I, 1994. 52p. (Technical Report).

[PON 94] PONTELLI, E.; GUPTA, C.; HERMENEGILDO, M. &ACE: a high-performance parallel prolog system. In: INTERNATONAL SYMPOSIUM ON PARALLEL SYMBOLIC COMPUTATION, 1, 1994. **Proceedings...** 1994.

[RAM 89] RAMKUMAR, B. & KALÉ, L. V. **Compiled Execution of the REDUCE-OR Process Model on Multiprocessor.** Urbana-Champaing: University of Illinois, May. 1989. 22p. (Technical Report UIUCDCS-R-89-1513).

[RAM 92] RAMKUMAR, B. & KALÉ, L. V. A Join Algorithm for Combining AND Parallel Solutions in AND/OR Parallel Systems. **International Journal of Parallel Programming**, v.21, n.1, 1992.

[SHA 89] SHAPIRO, E. The Family of Concurrent Logic Programming Languages. **ACM Computing Surveys**, New York, v.21, n.3, p.413-510, Sep. 1989.

[STE 86] STERLING, L; SHAPIRO, E. **The Art of Prolog.** Cambridge: MIT Press, 1986

[SWE 95] SWEDISH INSTITUTE OF COMPUTER SCIENCE. **SICStus Prolog User's Manual**. June 1995. 373 p.

[YAM 92] YAMIN, Adenauer C. **Modelos de Implementação do Paralelismo OU na Programação em Lógica**. Porto Alegre: CPGCC-UFRGS, 1992. 114p. (Trabalho Individual, 280).

[YAN 93] YANG, R; BEAUMONT, T; DUTRA, I; COSTA, V. S.; WARREN, D. H. D. Performance of the Compiler-based Andorra-I System. In: INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING, 10, Jun. 1993. **Proceedings...** Cambridge: MIT Press, 1993. p.150-166.