Parallel Execution of Logic Programs: Back to the Future (??)

Enrico Pontelli
Dept. Computer Science
New Mexico State University
Overview

1. Some motivations
   *[Logic Programming and Parallelism]*

2. The Past
   *[Types of Parallelism, Basic Schemes]*

3. The Present
   *[Recent Schemes]*

4. (Back to) The Future
   *[New Directions]*
MOTIVATIONS AND BASIC DEFINITIONS
Logic Programming

• **Definite programs**
  – collection of first-order Horn clauses
    
    \[
    \text{reachable}(X) :- \text{edge}(Y,X), \text{reachable}(Y).
    \]
  – semantics based on least Herbrand model

• **Normal programs**
  – enter negation as failure
    
    \[
    \text{color}(X, \text{red}) :- \text{node}(X), \text{not color}(X, \text{blue}).
    \]
  – several alternative semantics
    • well-founded semantics [XSB, tabling]
    • answer set semantics [Answer Set Programming]
Logic Programming: Systems

- **Definite Programs (Prolog, CLP)**
  - SLD-resolution
  - WAM-based

- **Answer Set Programs**
  - bottom-up execution models (answer = set)
    - variations of DPLL
    - mapping to SAT
Logic Programming: Smodels

Rules

Head
PosBody
NegBody
Index: j

Atoms

Rule j

Atom i

Head Rules
Pos Rules
Neg Rules
Truth Value
Index: i

Partial Answer Set

Atom e
Atom d
Atom c
Atom b
Atom a
LP and Parallelism

• LP considered suited for parallel execution since its inception
  – Kowalski “Logic for Problem Solving” (1979)
• Interest spawned by
  – LP ⇒ Declarative Language ⇒
    Limited or No Control ⇒ Limited Dependences ⇒ Easy Parallelism
  – Everlasting myth of “LP = slow execution”
LP and Parallelism

• Several good surveys


Overview

1. Some motivations
   [Logic Programming and Parallelism]

2. The Past
   [Types of Parallelism, Basic Schemes]

3. The Present
   [Recent Solutions]

4. (Back to) The Future
   [New Directions]
The Past

• Approaches
  – Explicit Schemes
    • message passing primitives (e.g., Delta-Prolog)
    • blackboard primitives (e.g., Jinni, CIAO Prolog)
    • dataflow/guarded languages (e.g., KLIC)
  – Implicit Schemes
  – Hybrid Schemes
while (Query not empty) do
  select literal B from Query
repeat
  select clause (H :- Body) from Program
until (unify(H,B) or no clauses left)
if (no clauses left) then FAIL
else
  \( \sigma = \text{MostGeneralUnifier}(H,B) \)
  Query = ((Query \ {B}) \cup Body)\( \sigma \)
endif
endwhile
Unification Parallelism

• Parallelize term-reduction stage of unification

\[ f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n) \mapsto \begin{cases} 
  t_1 = s_1 \\
  \vdots \\
  t_n = s_n 
\end{cases} \]

• Not a major focus
  – fine grained
  – dependences – common variables
  – SIMD algorithms (e.g., Barklund & Millroth)
Or-Parallelism

- Parallelize “don’t known” non-determinism in selecting matching clauses
  - processes exploring distinct solutions to the goal
  - computations are independent

- Environment representation problem
  - conditional variables
  - at minimum: each thread keeps copies of unbound ancestor conditional variables

New Mexico State University
Or-Parallelism: Classification Schemes

- Formalized in terms of three basic operations
  - binding management scheme
  - task switching scheme
  - task creation scheme
- Binding Scheme
  - Shared-tree methods
    - Binding arrays
    - Version vectors
  - Non shared-tree methods
    - Stack copying
- Task Switching Scheme
  - copying schemes
  - recomputation schemes
Or-Parallelism: two popular schemes

- Binding Arrays
Or-Parallelism: two popular schemes

• Stack Copying

- need for binding installation
- incrementality
And-Parallelism

- Concurrent execution of different literals in a resolvent

```
integ(X + Y, Z) :- integ(X,A), integ(Y,B), Z = A + B.
```

Parallel literals continuation

```
?- integ(X+Y,Z)
?- integ(X,A), integ(Y,B)
integ(X,A)
integ(Y,B)
Z = A + B
```
And-Parallelism

- Two traditional forms
  - Independent and-parallelism
  - runtime access to independent sets of variables

```prolog
quick(In, Out) :- partition(In, First, Low, High),
  (indep(Low, High) =\rightarrow quick(Low, SLow) \& quick(High, SHigh)
  test ; quick(Low, SLow), quick(High, SHigh)
  ),
append(SLow, [First | SHigh], Out).
```

*parallel case*

*sequential case*
And-Parallelism

\[ p, (p_1 \& p_2 \& p_3), p_4 \ldots \]

Processor 1

\[ \text{Chpt Stack} \]

\[ p_1 \]

\[ \text{parcall frame} \]

Processor 2

\[ \text{Chpt Stack} \]

\[ p_2 \]

Processor 3

\[ \text{Chpt Stack} \]

\[ p_3 \]

Goal Stack

\[ p_2 \]

\[ p_3 \]
And-Parallelism

\[ p, (p_1 \& p_2 \& p_3), p_4, \ldots \]

Processor 1

Processor 2

Processor 3

Chpt Stack

Goal Stack

Output frame

Parcall frame
And-Parallelism

- Backtracking

$p1(\cdot), (\text{<cond>} \Rightarrow p2(\cdot) \& p3(\cdot) \& p4(\cdot)), p5(\cdot)$

`p(X) & q(Y)`

**Processor 1**
- Chpt Stack
- Input marker
- End marker

**Processor 2**
- Chpt Stack
- Input marker
- End marker
And-Parallelism

- Outside backtracking
  \[ p_1(\_), (\langle \text{cond} \rangle \implies p_2(\_) \& p_3(\_) \& p_4(\_)), p_5(\_) \]

- Standard right-to-left
  - across processors
  - skip deterministic goals
  - restart in parallel
And-Parallelism

• Inside backtracking

\[ p_1(\cdot), \text{(<cond>} \Rightarrow p_2(\cdot) \land p_3(\cdot) \land p_4(\cdot)), p_5(\cdot) \]
And-Parallelism

- Dependent and-parallelism
  \[ p(X) \& q(X) \]
  - Goals
    - consistent bindings
    - reproduce Prolog observable behavior
  - Common approach
    - dynamic classification of subgoals as producers/consumers
    - several complex schemes (e.g., filtered binding model, DDAS)
  - Complex backtracking
Overview

1. Some motivations
   [Logic Programming and Parallelism]

2. The Past
   [Types of Parallelism, Basic Schemes]

3. The Present
   [Recent Solutions]

4. (Back to) The Future
   [New Directions]
The Present (or recent past...)

- Parallelism: Stack Splitting
  - or parallelism on clusters
  - stack copying is promising, but complexity of sharing choice points
    - communication during backtracking, or
    - revert to topmost scheduling
  - subdivide alternatives and/or choice points between processors
  - horizontal splitting
  - vertical splitting
    - partition: CP* $\rightarrow$ CP* $\times$ CP* e.g.,
      - alternate($a_1b_1a_2b_2a_3b_3...$) = ($a_1a_2a_3...$),($b_1b_2b_3...$)
      - block($a_1a_2a_3...a_n$) = ($a_1...a_{n/2-1}$),($a_{n/2}...a_n$)

---

Knowledge representation, Logic, and Advanced Programming Laboratory

New Mexico State University
Understanding the Problems...

• Formalizations
  – modeling key aspects of parallel LP as problems on dynamic trees
  – investigation of computational properties

• Some interesting results
  – environment representation problem
    • operations: create_tree, expand, remove, assign, dereference
    • $\Omega(\lg n)$
Further Issues

- Prolog as a “real” programming language
  - Side-effects, order-sensitive predicates
  - Goal: recreate the same observable behavior of sequential Prolog
  - Sequentialize order-sensitive predicates
  - Sequential is opposite of Parallel...
  - Dynamic vs. Static management of order-sensitive predicates
Order-sensitive Executions

- Idea: a side-effect should be delayed until all “preceeding” side-effects have been completed
- determining the exact time of execution: undecidable
- safe approximation: delay until all “impure” branches on the left of the side-effects have been completed
Order-sensitive predicates

- Standard Technique: maintain subroot nodes for each node
- Subroot(X) = root of largest subtree containing X in which X is leftmost

- Aurora, Muse: O(n) algorithms for maintaining subroot nodes
- possible to perform O(1) on shared memory
- approximated on distributed memory: block splitting, give away top part of branch
- Preemptive scheduling
Overview

1. Some motivations
   [Logic Programming and Parallelism]

2. The Past
   [Types of Parallelism, Basic Schemes]

3. The Present
   [Recent Solutions]

4. The Future
   [New Directions]
(Back to) The Future

This page describes KLIC, which is a concurrent logic programming language.

KLIC was developed in 1993 and distributed as an IPS, and is now distributed by KLIC Association.

Please read the README file for details:
- Current version
- Available platforms
- Parallel implementations
- Changes
- Documents
- Mailing lists

Current version

Version 3.0.03 is the latest and is currently distributed from the following sites:

- http://www.klic.org/files/v3.0/klic-3.0.03.tar.gz (primary site)
- ftp://pwd.chroot.org/pub/klic/v3.0/klic-3.0.03.tar.gz (secondary site)

This distribution contains the following three implementations:
- Sequential (pseudo parallel) implementation
- Distributed memory parallel implementation
- Shared memory parallel implementation

Unfortunately, the current shared memory parallel implementation still has fatal bugs. If you are using the old shared memory parallel implementation (previous version is 2.0.02), please do not replace it with the current version.
(Back to) The Future… NOT!

• **Multi-core**
  – back to shared memory platforms
  – small/medium/large scale
  – the future is not the same as the past…
    • GPUs: large number of simple threads, limited interactions, complex memory model
    • CPUs: tricky cache behavior
    • Other upcoming platforms (e.g., cell processors)

• **Requirements**
  – better investigation of locality
  – hybrid architectures ✈ hybrid models
    • ACE, Andorra-I, FIRE, AOW
Perspectives

• Mapping forms of parallelism to hw levels
  – Originally designed in ACE
    • Teams of processors
    • Each team as an or-parallel agent (stack copying)
    • Each member of a team as an and-parallel agent (&ACE)
  – Experimented with in Jsmodels
    1. Propagation Parallelism – threads within a multicore node
    2. Or-Parallelism – processes allocated to distinct nodes of a cluster
    • Pathways 10
  – Other possibilities
    • GPUs for unification parallelism
(Back to) The Future… NOT!

- Making parallel LP boring
  - High level parallel primitives
  - Implement forms of parallelism in Prolog
  - Original idea
    - Codish & Shapiro (1987)
    - &-Prolog CGE (1988)
    - &-ACE DAP (1995)

- An Example: primitives for and-parallelism
  - G&>Handler (post goal in a goal queue)
  - Handler &< (wait for goal associated to Handler)
    - A&B :- A&>Handler, call(B), Handler &<.
  - Operations for accessing goal list
  - Mutex on variables
(Back to) The Future... NOT!

• Parallelism in
  – Parallel Answer Sets
    • Jsmodels
    • Platypus
  – Jsmodels
    • sequential m
      semantics are

function jsmodels(P)
  \[ S = (\emptyset, \emptyset) \]
  loop
    \[ S = \text{wfm}(P, S) \]
    if \( S^+ \cap S^- \neq \emptyset \) then fail
    if (S is complete) then return S
    pick \( f \) and choose
    \[ S^+ = S^+ \cup \{f\} \]
    \[ S^- = S^- \cup \{f\} \]
  endloop
(Back to) The Future… NOT!

- **Parallel jsmodels**
  - search parallelism
    - parallel
    - environment
    - 3 sharing schemes
      - copying
      - recomputation
      - recomputation with backtracking
    - both sender initiated and receiver initiated scheduling
  - lookahead parallelism
    - find unknown \( \mathcal{F} \) s.t. either \( \text{wfm} (S \cup \{ \mathcal{F} \}) \) or \( \text{wfm} (S \cup \{ \neg \mathcal{F} \}) \) is inconsistent
    - lookahead steps are deterministic and can be performed in parallel

---

New Mexico State University

Knowledge representation, Logic, and Advanced Programming Laboratory

[Graph showing speedup factors for different benchmarks]
Conclusion

- Parallel LP is coming back
- Novel perspectives
  - architectural impact
  - portability & maintainability at the price of greater overheads
  - language specific schemes
- Bright future ahead!
Acknowledgments

• KLAP = Knowledge representation, Logic, and Advanced Programming
ICLP 2008

24th International Conference on Logic Programming

Udine (Italy)
December 9-13, 2008

http://icl08.dimi.uniud.it
Thank You

Questions?