Towards a High-Level Implementation of Flexible Parallelism Primitives for Symbolic Languages

Amadeo Casas¹  Manuel Carro²  Manuel Hermenegildo¹,²

¹University of New Mexico (USA)
²Technical University of Madrid (Spain)

PASCO’07 - July 28th, 2007
Introduction (I) - Motivation

- Parallelism (finally!) becoming mainstream thanks to multicore architectures – even on laptops!

- Declarative languages interesting for parallelization:
  - Notion of control provides more flexibility.
  - Amenability to semantics-preserving automatic parallelization.

- And also well-suited to write symbolic computation algorithms:
  - Program close to problem description.

- Much previous work:
  - Logic programming (LP) languages.
  - Functional languages: Erlang, Sisal, etc.

- Two objectives in this work:
  - New, efficient, more flexible approach for exploiting parallelism in LP.
  - Automatic parallelization of logic programs.
Introduction (II) - Types of Parallelism in LP

- Two main types:
  - Or-parallelism: explores in parallel alternative computation branches.
  - And-parallelism: executes procedure calls in parallel.
    - Traditional parallelism: parbegin-parend, loop parallelization, divide-and-conquer, etc.
    - Often marked with & operator: fork-join nested parallelism.
Introduction (II) - Types of Parallelism in LP

- Two main types:
  - Or-parallelism: explores in parallel alternative computation branches.
  - And-parallelism: executes procedure calls in parallel.
    - Traditional parallelism: parbegin-parend, loop parallelization, divide-and-conquer, etc.
    - Often marked with & operator: fork-join nested parallelism.

Example (QuickSort: sequential and parallel versions)

```
qsort([], []).  qsort([], []).  
qsort([X|L], R) :-  qsort([X|L], R) :-
  partition(L, X, SM, GT),  partition(L, X, SM, GT),
  qsort(GT, SrtGT),  qsort(GT, SrtGT) &
  qsort(SM, SrtSM),  qsort(SM, SrtSM),
  append(SrtSM, [X|SrtGT], R).  append(SrtSM, [X|SrtGT], R).
```

- We will focus on and-parallelism.
  - Need to detect independent tasks.
Introduction (III) - Notion of Independence

- **Correctness**: same results as sequential execution.
- **Efficiency**: execution time \( \leq \) than seq. program (no slowdown), assuming parallel execution has no overhead.

\[
\begin{array}{c|c|c|c}
  & s_1 & s_2 & Y := W+2, \\
  & & & X := Y+Z, \\
  & (imperative) & (functional) & Y = W+2, X = Y+Z, \\
\end{array}
\]

\[
\begin{array}{c}
\text{main :-} \\
p(X) :- X = [1,2,3]. \\
s_1 \quad p(X), \\
s_2 \quad q(X), \\
\text{write}(X). \\
\end{array}
\]

- **Fundamental issue**: \( p \) affects \( q \) (prunes its choices).
  - \( q \) ahead of \( p \) is speculative.
- **Independence**: correctness + efficiency.
Introduction (IV) - Ciao

- **Ciao**, new generation multi-paradigm language.
  - Supports ISO-Prolog fully (as a library).

- Predicates, functions (including laziness), constraints, higher-order, objects, etc.

- Global analyzer which **infers** many properties such as:
  - Types, pointer aliasing, non-failure, determinacy, termination, data sizes, cost, etc.

- Automatic verification of program assertions (and bug detection if assertions are proved false).

- Parallel, concurrent and distributed execution primitives + automatic parallelization and automatic granularity control.
Automatic Parallelization (I) - CDGs

- **Conditional dependency graph:**
  - Vertices: possible tasks (statements, calls, etc.).
  - Edges: conditions needed for independence: variable sharing.

- Local or global analysis to remove checks in the edges.
- Annotation process converts graph back to parallel expressions in source.

```prolog
foo(...) :-
g1(...),
g2(...),
g3(...).
```

Alternative:

```prolog
"Annotation"
Local/Global analysis and simplification

( test(1-3) -> ( g1, g2 ) & g3 ; g1, ( g2 & g3 ) ).
```

Alternative:

```prolog
\ g1, ( g2 & g3 )
```

Casas, Carro, Hermenegildo (UNM, UPM)
Towards a High-Level Implementation.

PASCO'07 - July 28th, 2007
More flexible constructions to represent parallelism:

- \( G \ &> \ H \) — schedules goal \( G \) for parallel execution and continues executing the code after \( G \ &> \ H \).
  - \( H \) is a handler which contains the state of goal \( G \).
- \( H \ &< \) — waits for the goal associated with \( H \) to finish.
  - Bindings made for the output variables of the parallel goal associated to \( H \) are available (i.e., goal has produced a complete solution).

Operator \& written as:

\[
A \ & B :- A \ &> H, \ call(B), H \ &<.\\
\]

Optimized deterministic versions: \&!/2, \&!/1.
Automatic Parallelization (III) - Flexible Parallelism Primitives (II)

- More parallelism can be exploited with these primitives:

\[ a(X,Z) \rightarrow b(X) \]
\[ c(Y) \rightarrow d(Y,Z) \]

- (sequential)
- (restricted IAP)
- (unrestricted IAP)
Shared-Memory Implementation

- Versions of and-parallelism previously implementated: &-Prolog, &-ACE, AKL, Andorra-I.
- They rely on complex low-level machinery:
  - Each agent: goal stack, parcall frames, markers, etc.
- Current implementation for shared-memory multiprocessors:
  - Each agent: sequential Prolog machine + goal list + Prolog code.
- Approach: rise components to the source language level:
  - **Prolog-level**: goal publishing, goal searching and goal scheduling.
  - **C-level**: low-level threading, locking, stack management, sharing of memory and untrailing.
  - Simpler machinery and more flexibility.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Number of processors</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seq.</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>AIAKL</td>
<td>1.00</td>
<td>0.94</td>
<td>1.76</td>
<td>1.80</td>
<td>1.80</td>
<td>1.79</td>
<td>1.52</td>
<td>1.77</td>
<td>1.76</td>
</tr>
<tr>
<td>Ann</td>
<td>1.00</td>
<td>0.97</td>
<td>1.77</td>
<td>2.61</td>
<td>3.22</td>
<td>3.98</td>
<td>4.52</td>
<td>5.14</td>
<td>5.61</td>
</tr>
<tr>
<td>Boyer</td>
<td>1.00</td>
<td>0.17</td>
<td>0.33</td>
<td>0.49</td>
<td>0.60</td>
<td>0.70</td>
<td>0.81</td>
<td>0.89</td>
<td>0.94</td>
</tr>
<tr>
<td>BoyerGC</td>
<td>1.00</td>
<td>0.86</td>
<td>1.66</td>
<td>2.45</td>
<td>3.13</td>
<td>3.66</td>
<td>4.17</td>
<td>4.63</td>
<td>5.10</td>
</tr>
<tr>
<td>Deriv</td>
<td>1.00</td>
<td>0.16</td>
<td>0.33</td>
<td>0.45</td>
<td>0.57</td>
<td>0.68</td>
<td>0.80</td>
<td>0.90</td>
<td>0.99</td>
</tr>
<tr>
<td>DerivGC</td>
<td>1.00</td>
<td>0.77</td>
<td>1.40</td>
<td>2.05</td>
<td>2.66</td>
<td>3.24</td>
<td>3.66</td>
<td>4.13</td>
<td>4.57</td>
</tr>
<tr>
<td>FFT</td>
<td>1.00</td>
<td>0.30</td>
<td>0.48</td>
<td>0.59</td>
<td>0.67</td>
<td>0.75</td>
<td>0.77</td>
<td>0.80</td>
<td>0.82</td>
</tr>
<tr>
<td>FFTGC</td>
<td>1.00</td>
<td>0.97</td>
<td>1.72</td>
<td>2.16</td>
<td>2.65</td>
<td>2.77</td>
<td>2.94</td>
<td>3.06</td>
<td>3.19</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>1.00</td>
<td>0.15</td>
<td>0.29</td>
<td>0.42</td>
<td>0.55</td>
<td>0.67</td>
<td>0.81</td>
<td>0.95</td>
<td>1.09</td>
</tr>
<tr>
<td>FibonacciGC</td>
<td>1.00</td>
<td>0.99</td>
<td>1.94</td>
<td>2.88</td>
<td>3.81</td>
<td>4.75</td>
<td>5.69</td>
<td>6.63</td>
<td>7.52</td>
</tr>
<tr>
<td>Hamming</td>
<td>1.00</td>
<td>0.89</td>
<td>1.19</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
</tr>
<tr>
<td>Hanoi</td>
<td>1.00</td>
<td>0.46</td>
<td>0.83</td>
<td>1.19</td>
<td>1.50</td>
<td>1.75</td>
<td>1.86</td>
<td>2.21</td>
<td>2.44</td>
</tr>
<tr>
<td>HanoiDL</td>
<td>1.00</td>
<td>0.24</td>
<td>0.45</td>
<td>0.68</td>
<td>0.85</td>
<td>1.07</td>
<td>1.28</td>
<td>1.47</td>
<td>1.67</td>
</tr>
<tr>
<td>HanoiGC</td>
<td>1.00</td>
<td>0.98</td>
<td>1.80</td>
<td>2.33</td>
<td>2.89</td>
<td>3.32</td>
<td>3.70</td>
<td>3.80</td>
<td>4.07</td>
</tr>
<tr>
<td>MMatrix</td>
<td>1.00</td>
<td>0.76</td>
<td>1.46</td>
<td>2.11</td>
<td>2.82</td>
<td>3.46</td>
<td>4.02</td>
<td>4.59</td>
<td>5.18</td>
</tr>
<tr>
<td>QuickSort</td>
<td>1.00</td>
<td>0.57</td>
<td>1.08</td>
<td>1.52</td>
<td>1.90</td>
<td>2.25</td>
<td>2.56</td>
<td>2.81</td>
<td>2.98</td>
</tr>
<tr>
<td>QuickSortDL</td>
<td>1.00</td>
<td>0.52</td>
<td>0.97</td>
<td>1.32</td>
<td>1.69</td>
<td>2.11</td>
<td>2.35</td>
<td>2.63</td>
<td>2.86</td>
</tr>
<tr>
<td>QuickSortGC</td>
<td>1.00</td>
<td>0.98</td>
<td>1.78</td>
<td>2.30</td>
<td>2.85</td>
<td>3.18</td>
<td>3.44</td>
<td>3.62</td>
<td>3.68</td>
</tr>
<tr>
<td>Takeuchi</td>
<td>1.00</td>
<td>0.11</td>
<td>0.21</td>
<td>0.31</td>
<td>0.40</td>
<td>0.47</td>
<td>0.56</td>
<td>0.61</td>
<td>0.69</td>
</tr>
<tr>
<td>TakeuchiGC</td>
<td>1.00</td>
<td>0.87</td>
<td>1.53</td>
<td>2.16</td>
<td>2.59</td>
<td>2.60</td>
<td>2.60</td>
<td>2.60</td>
<td>2.60</td>
</tr>
</tbody>
</table>
Performance Results (II) - Granularity Control

- Boyer-Moore
- Boyer-Moore with granularity control
- Derivation
- Derivation with granularity control
- Fibonacci
- Fibonacci with granularity control
- QuickSort
- QuickSort with granularity control
Performance Results (III) - Unrestricted And-Parallelism

<table>
<thead>
<tr>
<th>Benchm.</th>
<th>And-P</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>FibFun</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Restr.</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>Unrestr.</td>
<td>0.99</td>
<td>1.94</td>
<td>2.88</td>
<td>3.81</td>
<td>4.75</td>
<td>5.69</td>
<td>6.63</td>
</tr>
<tr>
<td>Takeuchi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Restr.</td>
<td>0.87</td>
<td>1.53</td>
<td>2.16</td>
<td>2.59</td>
<td>2.60</td>
<td>2.60</td>
<td>2.60</td>
</tr>
<tr>
<td></td>
<td>Unrestr.</td>
<td>0.87</td>
<td>1.53</td>
<td>2.25</td>
<td>3.21</td>
<td>3.88</td>
<td>4.27</td>
<td>4.97</td>
</tr>
<tr>
<td>FFT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Restr.</td>
<td>0.97</td>
<td>1.72</td>
<td>2.16</td>
<td>2.65</td>
<td>2.77</td>
<td>2.94</td>
<td>3.06</td>
</tr>
<tr>
<td></td>
<td>Unrestr.</td>
<td>0.97</td>
<td>1.73</td>
<td>2.19</td>
<td>2.69</td>
<td>2.83</td>
<td>3.01</td>
<td>3.18</td>
</tr>
<tr>
<td>Hamming</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Restr.</td>
<td>0.89</td>
<td>1.19</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
<td>1.43</td>
</tr>
<tr>
<td></td>
<td>Unrestr.</td>
<td>0.89</td>
<td>1.21</td>
<td>1.51</td>
<td>1.51</td>
<td>1.51</td>
<td>1.51</td>
<td>1.51</td>
</tr>
<tr>
<td>WMS2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Restr.</td>
<td>0.99</td>
<td>1.01</td>
<td>1.01</td>
<td>1.01</td>
<td>1.01</td>
<td>1.01</td>
<td>1.01</td>
</tr>
<tr>
<td></td>
<td>Unrestr.</td>
<td>0.99</td>
<td>1.10</td>
<td>1.10</td>
<td>1.10</td>
<td>1.10</td>
<td>1.10</td>
<td>1.10</td>
</tr>
</tbody>
</table>
Performance Results (IV) - Comparison Restr./Unrestr. Takeuchi

Casas, Carro, Hermenegildo (UNM, UPM) Towards a High-Level Implementation... PASCO’07 - July 28th, 2007 13 / 1
Conclusions and Future Work

- New implementation approach for exploiting and-parallelism:
  - Simpler machinery.
  - More flexibility.

- Preliminary results:
  - Reasonable speedups are achievable.
  - Additional overhead → granularity control.
  - Also, advances in compilation and improved implementation should (partly) recover efficiency lost due to overhead.
  - Unrestricted and-parallelism provides better speedups.

- Currently working on improving implementation and developing compile-time (automatic) parallelizers for this approach.
# Example (Unrestricted Annotation)

<table>
<thead>
<tr>
<th>a(X,Z)</th>
<th>b(X)</th>
<th>c(Y)</th>
<th>d(Y,Z)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Indep</th>
<th>Dep</th>
<th>Joinable</th>
<th>Fork</th>
<th>And</th>
<th>Join</th>
<th>Pub</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

---

Casas, Carro, Hermenegildo (UNM, UPM) Towards a High-Level Implementation... PASCO'07 - July 28th, 2007 15 / 1
Example (Unrestricted Annotation)

<table>
<thead>
<tr>
<th>Indep</th>
<th>Dep</th>
<th>Joinable</th>
<th>Fork</th>
<th>And</th>
<th>Join</th>
<th>Pub</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>∅</td>
</tr>
<tr>
<td>{a, c}</td>
<td>{b, d}</td>
<td>{a}</td>
<td>{c}</td>
<td>{a}</td>
<td>∅</td>
<td>{a, c}</td>
</tr>
</tbody>
</table>

\[ p(X,Y,Z) :- \\
\quad c(Y) \&> Hc, \\
\quad a(X,Z), \]
Example (Unrestricted Annotation)

<table>
<thead>
<tr>
<th>Indep</th>
<th>Dep</th>
<th>Joinable</th>
<th>Fork</th>
<th>And</th>
<th>Join</th>
<th>Pub</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>∅</td>
</tr>
<tr>
<td>{a, c}</td>
<td>{b, d}</td>
<td>{a}</td>
<td>{c}</td>
<td>{a}</td>
<td>∅</td>
<td>{a, c}</td>
</tr>
<tr>
<td>{b, c}</td>
<td>{d}</td>
<td>{c}</td>
<td>{b}</td>
<td>∅</td>
<td>{c}</td>
<td>{a, c}</td>
</tr>
</tbody>
</table>

\[ p(X,Y,Z) :\]
\[ c(Y) \&> Hc, \]
\[ a(X,Z), \]
\[ b(X) \&> Hb, \]
\[ Hc \&<. \]
**Example (Unrestricted Annotation)**

<table>
<thead>
<tr>
<th>Indep</th>
<th>Dep</th>
<th>Joinable</th>
<th>Fork</th>
<th>And</th>
<th>Join</th>
<th>Pub</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, c}</td>
<td>{b, d}</td>
<td>{a}</td>
<td>{c}</td>
<td>{a}</td>
<td>∅</td>
<td>{a, c}</td>
</tr>
<tr>
<td>{b, c}</td>
<td>{d}</td>
<td>{c}</td>
<td>{b}</td>
<td>∅</td>
<td>{c}</td>
<td>{a, c}</td>
</tr>
<tr>
<td>{b, d}</td>
<td>∅</td>
<td>{b, d}</td>
<td>∅</td>
<td>{d}</td>
<td>{b}</td>
<td>{a, b, c}</td>
</tr>
</tbody>
</table>

\[p(X,Y,Z) :-
   c(Y) &> Hc,
   a(X,Z),
   b(X) &> Hb,
   Hc <&,
   d(Y,Z),
   Hb <&.\]
Minimum Time to Execute a Parallel Expression (I)

\[ p(X,Y,Z) :- \\
\quad a(X,Z) \& c(Y), \\
\quad b(X) \& d(Y,Z) . \]

\[ T_{fj1} = \max(T_a, T_c) + \max(T_b, T_d) \]

\[ p(X,Y,Z) :- \\
\quad (a(X,Z), b(X)) \& c(Y), \\
\quad d(Y,Z). \]

\[ T_{fj2} = \max(T_a + T_b, T_c) + T_d \]
Minimum Time to Execute a Parallel Expression (II)

\[\text{dep}\]

\[\begin{align*}
p(X,Y,Z) & :- \\
c(Y) & \& Hc \\
a(X,Z) & \\
b(X) & \& Hb \\
Hc & <& \\
d(Y,Z) & \\
Hb & <& \\
T_1 & = 0 \\
T_2 & = T_1 \\
T_3 & = T_2 + T_a \\
T_4 & = T_3 \\
T_5 & = \max(T_3, T_1 + T_c) \\
T_6 & = T_5 + T_d \\
T_7 & = \max(T_6, T_3 + T_b) = T_{dep}
\end{align*}\]
Minimum Time to Execute a Parallel Expression (III)

\[ Tfj_1 = \max(a, c) + \max(b, d) \]

\[ tfj1(A,B,C,D,T) :- \]
\[ \text{positive([A,B,C,D,T])}, \]
\[ \max(A,C,MAC), \]
\[ \max(B,D,MBD), \]
\[ T = MAC + MBD. \]

\[ Tfj_2 = \max(a+b, c) + d \]

\[ tfj2(A,B,C,D,T) :- \]
\[ \text{positive([A,B,C,D,T])}, \]
\[ AB = A + B, \]
\[ \max(AB,C,MaxABC), \]
\[ T = D + MaxABC. \]

\[ T_{dep} = \max(a+b, d + \max(a,c)) \]

\[ tdep(A,B,C,D,T):- \]
\[ \text{positive([A,B,C,D,T])}, \]
\[ AB = A + B, \]
\[ \max(A, C, MaxAC), \]
\[ DAC = D + MaxAC, \]
\[ \max(AB, DAC, T). \]
### Minimum Time to Execute a Parallel Expression (IV)

**In any fork-join parallelization always better than the other one?**

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>?- tfj1(A,B,C,D,T1), tfj2(A,B,C,D,T2), T1 .&lt;. T2.</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Can fork-join parallelization be better than unrestricted parallelization?**

<table>
<thead>
<tr>
<th>Case</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>?- tfj1(A,B,C,D,T1), tdep(A,B,C,D,T2), T1 .&lt;. T2.</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

- No combination of execution times can make the unrestricted parallelization be worse than the restricted parallelization!