Parallel Backtracking with Answer Memoing for Independent And-Parallelism

Pablo Chico de Guzman[1] Amadeo Casas[2] Manuel Carro[1,3]
Manuel V. Hermenegildo[1,3]

[1]School of Computer Science, Technical University of Madrid, Spain
[2]Samsung Research, USA
[3]IMDEA Software Institute, Spain

ICLP 2011
Lexington, KY – July 7, 2011

Introduction

- Two main sources of parallelism in LP:
  - OR-Parallelism: Aurora, MUSE, etc.
  - AND-Parallelism: &-Prolog, DDAS, etc.
  - Both: (&)ACE, AKL, Andorra-I, EAM, etc.
- Classical approach in Independent AND-Parallelism (IAP):
  - Conery approach:
    - Copying goals overhead.
    - Nonterminating programs.
  - Recomputation + sequential backtracking.
    Saves memory and preserves sequential semantics, but...
    - Recomputation can be inefficient.
    - Sequential backtracking limits parallelism.

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2)}^{g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5)}^{g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2 + 5 + 5)}^{g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2)}^{g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2+5)}^{g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2+5+5)}^{g2} = 36 \text{ secs.}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2+5+5)}^{g2} = 36 \text{ secs.}$
- $\text{Time}_{main2} = \overbrace{max(2,2)}^{g1\&g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2+5+5)}^{g2} = 36$ secs.
- $\text{Time}_{main2} = \overbrace{max(2,2)}^{g1\&g2} + \overbrace{(5)}^{g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2+5+5)}^{g2} = 36 \text{ secs.}$
- $\text{Time}_{main2} = \overbrace{max(2,2)}^{g1\&g2} + \overbrace{(5+5)}^{g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2+5+5)}^{g2} = 36 \text{ secs.}$
- $\text{Time}_{main2} = \overbrace{max(2,2)}^{g1\&g2} + \overbrace{(5+5)}^{g2} + \overbrace{max(10,2)}^{g1\&g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2+5+5)}^{g2} = 36 \text{ secs.}$
- $\text{Time}_{main2} = \overbrace{max(2,2)}^{g1\&g2} + \overbrace{(5+5)}^{g2} + \overbrace{max(10,2)}^{g1\&g2} + \overbrace{(5)}^{g2}$

Classical approach in IAP

Example (Traditional IAP execution - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $Time_{main1} = \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} + \overbrace{(2+5+5)}^{g2} = 36$ secs.
- $Time_{main2} = \overbrace{max(2,2)}^{g1\&g2} + \overbrace{(5+5)}^{g2} + \overbrace{max(10,2)}^{g1\&g2} + \overbrace{(5+5)}^{g2} = 32$ secs.
- Speedup $= 1.12$. Can we do better?

Our approach for IAP

- We propose an improved solution to backtracking in IAP, which combines:
  - Parallel backtracking.
  - Answer memoing.
  - Incremental computation of answers.
- We maintain high-level approach [ICLP'08].
  - Easier to maintain and extend.

Our approach for IAP

Example (IAP with parallel backtracking - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = 2 + (2 + 5 + 5) \; + \; 10 + (2 + 5 + 5) = 36$ secs.

Our approach for IAP

Example (IAP with parallel backtracking - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = 2 + (2 + 5 + 5)\ +\ 10 + (2 + 5 + 5) = 36$ secs.
- Time_{main1} with memoization $= \overbrace{2}^{g1}$

Our approach for IAP

Example (IAP with parallel backtracking - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = 2 + (2+5+5) \; + \; 10 + (2+5+5) = 36$ secs.
- Time_{main1} with memoization $= \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2}$

Our approach for IAP

Example (IAP with parallel backtracking - main1 vs. main2)

```
main1(X, Y) :-
  g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
  g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = 2 + (2 + 5 + 5) \; + \; 10 + (2 + 5 + 5) = 36$ secs.
- Time_{main1} with memoization $= \overbrace{2}^{g1} + \overbrace{(2 + 5 + 5)}^{g2} + \overbrace{10}^{g1} = 24$ secs.

Our approach for IAP

Example (IAP with parallel backtracking - main1 vs. main2)

```
main1(X, Y) :-
    g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
    g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $\text{Time}_{main1} = 2 + (2+5+5) \; + \; 10 + (2+5+5) = 36$ secs.
- Time_{main1} with memoization $= \overbrace{2}^{g1} + \overbrace{(2+5+5)}^{g2} + \overbrace{10}^{g1} = 24$ secs.
- Time_{main2} with memoization $= \overbrace{max(2,2)}^{g1 \& g2}$

Our approach for IAP

Example (IAP with parallel backtracking - main1 vs. main2)

```
main1(X, Y) :-
   g1(X), g2(Y).

g1(X) :- X=1, work(2).
g1(X) :- X=2, work(10).
```

```
main2(X, Y) :-
   g1(X) & g2(Y).

g2(Y) :- Y=1, work(2).
g2(Y) :- Y=2, work(5).
g2(Y) :- Y=3, work(5).
```

- $Time_{main1} = 2 + (2 + 5 + 5) \ + \ 10 + (2 + 5 + 5) = 36$ secs.
- $Time_{main1}$ with memoization $= \overbrace{2}^{g1} + \overbrace{(2 + 5 + 5)}^{g2} + \overbrace{10}^{g1} = 24$ secs.
- $Time_{main2}$ with memoization $= \overbrace{max(2, 2)}^{g1\&g2} + \overbrace{max(10, 5 + 5)}^{g1\&g2} = 12$ secs.
- Speedup w.r.t. sequential with memoization = 2.00.
- Speedup w.r.t. sequential = 3.00 (superlinear).

Our approach for IAP - Forward execution

- *Forward execution*: similar to classical IAP.
  - If a goal fails for first solution, conjunction fails (no slow-down!).
  - If/when all goals find a solution, execution proceeds.
- *Speculation*: if conjunction not finished yet, additional solutions are sought for goals that already computed an answer (if free agents).
  - Need to stash away generated solutions to continue searching for more answers (which are also saved).
- *Suspension*: when all goals find a solution, any speculative goals are suspended, their state saved, and their first answer reinstalled.
  - New **answers saved in memoing area** to avoid future recomputation.
  - Every new solution is **combined with** previously available solutions.

Our approach for IAP - Backward execution

- *Parallel backtracking*: if more solutions needed, backward execution performed in parallel. Suspended goals resume.
- *Backward* execution: performed over goals on top of the stack.
  - More efficient implementation, i.e., *trapped goals very infrequent*.
  - Solution order differs from sequential execution.

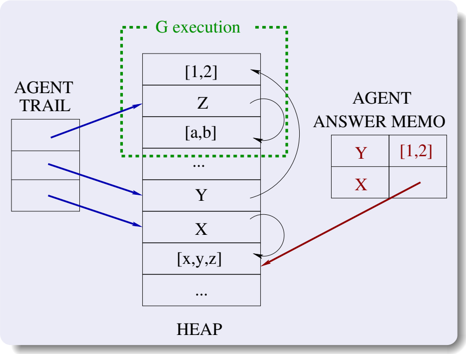
Memoization of Answers

- Goals are independent:
  - Makes sense to store answers to avoid recomputation (we know better how to do this now).
- Answer memoing different/simpler from tabling:
  - Termination not an issue. No need for detecting repeated calls, suspending/resuming consumers, etc.
  - All stored answers are discarded as soon as the conjunction continues after its last answer, or after the conjunction fails.
  - Visibility of stored answers restricted to only the parallel conjunction.

$\longrightarrow$ Easier case to implement/maintain than tabling.

Memoization of Answers - Implementation

- Answer memoization = saving a combination of heap and trail terms.
  - Trail segment corresponding to execution of parallel goal, and terms created during the goal execution pointed to by these variables.



- Reinstalling an answer consists of copying saved terms back to heap, and updating the trail.
- Memoization has a cost, but it can provide by itself substantial speedups since it avoids recomputation.

Answer Combination

- When last goal pending to generate an answer in a conjunction produces a solution, it needs to be combined (from right to left) with the rest of the answers to continue with forward execution.

Example (Combining answers in a parallel conjunction)

```
main(X, Y, Z) :-
  g1(X) & g2(Y) & g3(Z).
```

```
g1(a).  g1(b).  g1(c).
g2(x).  g2(y).  g2(z).
g3(1).  g3(2).  g3(3).
```

- Assume $g1$ and $g2$ have computed two answers, and $g3$ only one.
- Then, $g3$ finds a new answer $\{Z = 2\}$.
  - First combination of answers will be $(a, x, 2)$.
  - Next combination will be $(a, y, 2)$.
  - Final combinations will be $(b, x, 2)$ and $(b, y, 2)$.

Answer Combination - Implementation

- A *ghost* choice point is created with an alternative that retrieves the saved answers and installs the bindings.
  - Heap top pointer of ghost choice point points to the current heap top after copying terms, to protect those terms from backtracking for future answer combinations.
  - All these copied terms are released when the ghost choice point is eliminated.

Scheduler for the Parallel Backtracking IAP Engine

```
agent :- work, agent.

work :-
    find_parallel_goal(Handler) ->
    (
       goal_not_executed(Handler) ->
       (
          save_init_execution(Handler),
          call_parallel_goal(Handler)
       ;
          move_execution_top(Handler),
          fail
       )
    ;
       suspend,
       work
    ).
```

```
parcall_back(LGoals, NGoals) :-
    fork(PF,NGoals,LGoals,[Handler|LH]),
    (
       goal_not_executed(Handler) ->
       call_local_goal(Handler,Goal)
    ;
       true
    ),
    look_for_available_goal(LH),
    join(PF).

look_for_available_goal([]) :- !, true.
look_for_available_goal([Handler|LH]) :-
    (
       goal_available(Handler) ->
       call_local_goal(Handler,Goal)
    ;
       true
    ),
look_for_available_goal(LH).
```

Deterministic Parallel Goals

- Machinery proposed can be greatly simplified if goals deterministic.
  - Answer memoization and answer combination are not needed.
  - High-level scheduler can be greatly simplified.
- Some optimizations can be performed dynamically.
- Performance improvements of up to a factor of two in the execution of deterministic benchmarks.

Deterministic Benchmarks - Sun Fire T2000

- 8 cores x 4 threads, 8 Gb of memory, average of 10 runs.
- Speedups w.r.t. sequential (unparallelized) execution.

Benchmark	Approach	Number of processors				
		1	2	4	6	8
Fibo	*&-Prolog*	**0.98**	1.93	3.70	5.65	**7.34**
	seqback	0.95	1.89	3.70	5.36	6.96
	parback	0.95	1.88	3.69	5.33	6.94
	$parback_{det}$	**0.96**	1.91	3.74	5.41	**7.04**
MMatrix	*&-Prolog*	**1.00**	1.99	3.98	5.96	**7.93**
	seqback	0.78	1.55	2.99	4.29	5.55
	parback	0.76	1.52	2.95	4.22	5.45
	$parback_{det}$	**0.80**	1.60	3.01	4.55	**5.87**
QSort	*&-Prolog*	**1.00**	1.92	3.03	3.89	**4.65**
	seqback	0.50	0.98	1.74	2.27	2.67
	parback	0.49	0.97	1.74	2.27	2.69
	$parback_{det}$	0.56	1.10	1.96	2.57	3.02
	seqbackGC	0.97	1.77	3.02	3.77	4.15
	parbackGC	0.97	1.76	3.00	3.74	4.12
	$parbackGC_{det}$	**0.97**	1.78	3.04	3.79	**4.21**

Nondeterministic Benchmarks - Sun Fire T2000

Benchmark	Approach	Number of processors				
		1	2	4	6	8
CheckFiles	$seqback_{first}$	0.99	1.09	1.12	1.12	1.13
	$seqback_{all}$	0.99	1.05	1.07	1.08	1.08
	$parback_{first}$	3917	8612	17111	17116	44222
	pb_rel_{first}	1.00	2.20	4.37	4.37	11.29
	$parback_{all}$	**12915**	23409	45818	46955	**89571**
	pb_rel_{all}	**1.00**	1.81	3.55	3.64	**6.94**
Illumination	$seqback_{first}$	1.00	1.37	1.56	1.61	1.67
	$seqback_{all}$	1.00	1.16	1.24	1.25	1.27
	$parback_{first}$	1120	1725	3380	4028	6910
	pb_rel_{first}	1.00	1.54	3.02	3.60	6.17
	$parback_{all}$	**8760**	16420	31818	31888	**65314**
	pb_rel_{all}	**1.00**	1.87	3.63	3.64	**7.46**
QSortND	$seqback_{first}$	0.94	1.72	2.92	3.59	3.92
	$seqback_{all}$	0.91	0.96	0.99	1.00	1.00
	$parback_{first}$	0.94	1.72	2.91	3.57	3.91
	$parback_{all}$	**4.29**	6.27	9.90	10.90	**11.30**
	pb_rel_{all}	**1.00**	1.46	2.31	2.54	**2.64**

Conclusions

- Developed a parallel (out-of-order) backtracking approach for IAP, which relies on answer memoization to reuse and combine answers.
- Our approach provides significant performance improvements in the execution of nondeterministic parallel calls, due to the answer memoization mechanism and parallel backtracking.
- Optimizations allow avoiding the overhead for deterministic goals.