On a Tabling Engine that Can Exploit Or-Parallelism

Ricardo Rocha Fernando Silva Vítor Santos Costa

DCC-FC & LIACC, University of Porto, Portugal
{*ricroc,fds*}*@ncc.up.pt*

COPPE Systems and Engineering, University of Rio de Janeiro, Brazil
vitor@cos.ufrj.br

Overview

Tabling and Parallelism
Motivation and development guidelines.

Tabling Concepts
Execution model, tabled nodes, completion and leader nodes.

The Or-Parallel Tabling Engine
The OPT model and the public completion problem.

Performance Evaluation
Running times on a set of non tabled and tabled benchmarks.

Conclusions

Tabling and Parallelism

Tabling consists of storing intermediate answers for subgoals so that they can be reused when a repeated subgoal appears.

Tabling based models are able to:

- Avoid redundant subcomputations.
- Deal with infinite loops.

Tabling applications often perform search:

- We need to exploit alternatives for solving goals.
- Parallelise the search like in or-parallel Prolog.

Develop an efficient or-parallel tabling system

Tabling and Parallelism

Development Guidelines

- Exploit maximum parallelism.
- Extract parallelism from both tabled and non-tabled subgoals.
- Separate tabling and parallelism as much as possible.
- Take maximum advantage of current technology for parallel and tabling systems.
- Base performance comparable with current *state of the art* systems.

OPTYap = YapOr + YapTab + Tabling/Parallelism Integration

Tabling Concepts

Basic Execution Model

- Whenever a tabled subgoal is called for the first time, a new entry is allocated in the *table space*. This entry will collect all the answers generated for the subgoal.
- Variant calls to tabled subgoals are resolved by consuming the answers already stored in the table.
- As new answers are found, they are inserted into the table and returned to all variant subgoals.

Node (subgoal) Classification

- **Generator**: nodes that first call a tabled subgoal.
- **Consumer**: nodes that consume answers from the table space.
- **Interior**: nodes that are evaluated by the standard resolution.

Tabling Concepts

Completion

- A tabled subgoal is said to be *completely evaluated* when:
  - no more answers can be generated;
  - the variant subgoals have consumed all the available answers.

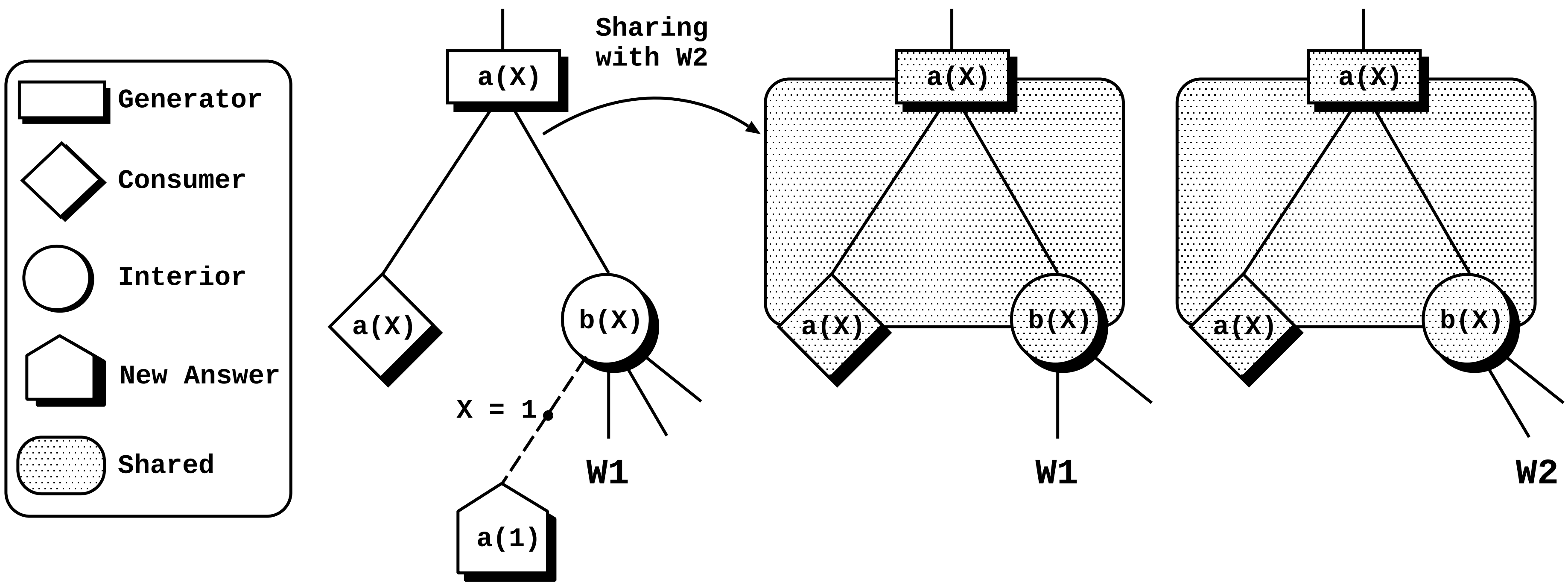
SCCs and Leader Nodes

- A number of subgoals may be mutually dependent, forming a *SCC*.
- A SCC is said to be completely evaluated when each subgoal belonging to the SCC is completely evaluated.
- Completion is performed at the *leader node*, i.e., the oldest subgoal in a SCC.

Overview of OPTYAP

Each worker physically owns a copy of the environment and shares a large area related to tabling and scheduling. Work is shared through *environment copying*.

Most of the time workers execute as if they were sequential tabling engines. The or-parallel component is triggered to schedule work and to access the shared region.



Contributions in OPTYap

Parallel Data Structures for Tabling

- Concurrent table accesses
- Dependency frames

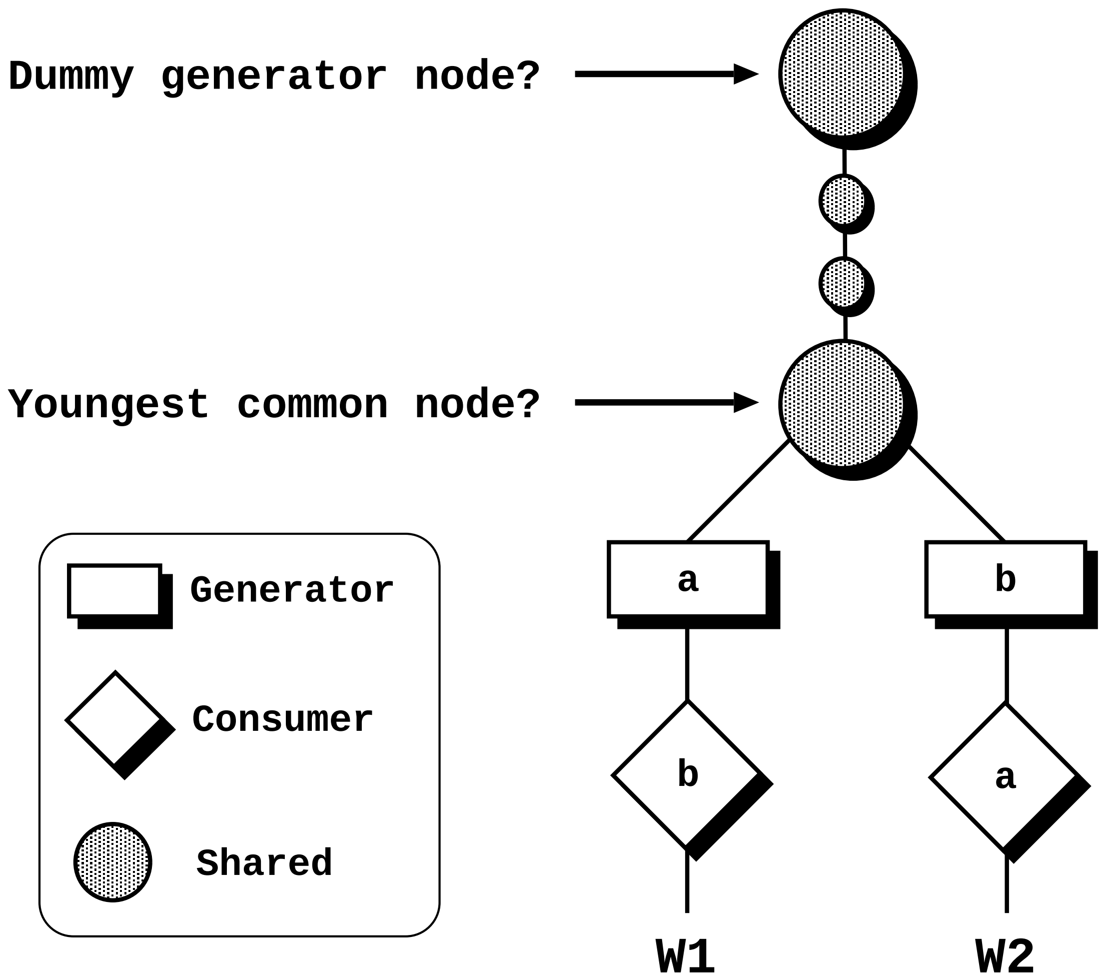
Work Sharing

- Scheduler support for tabled work

Completion of Parallel Work

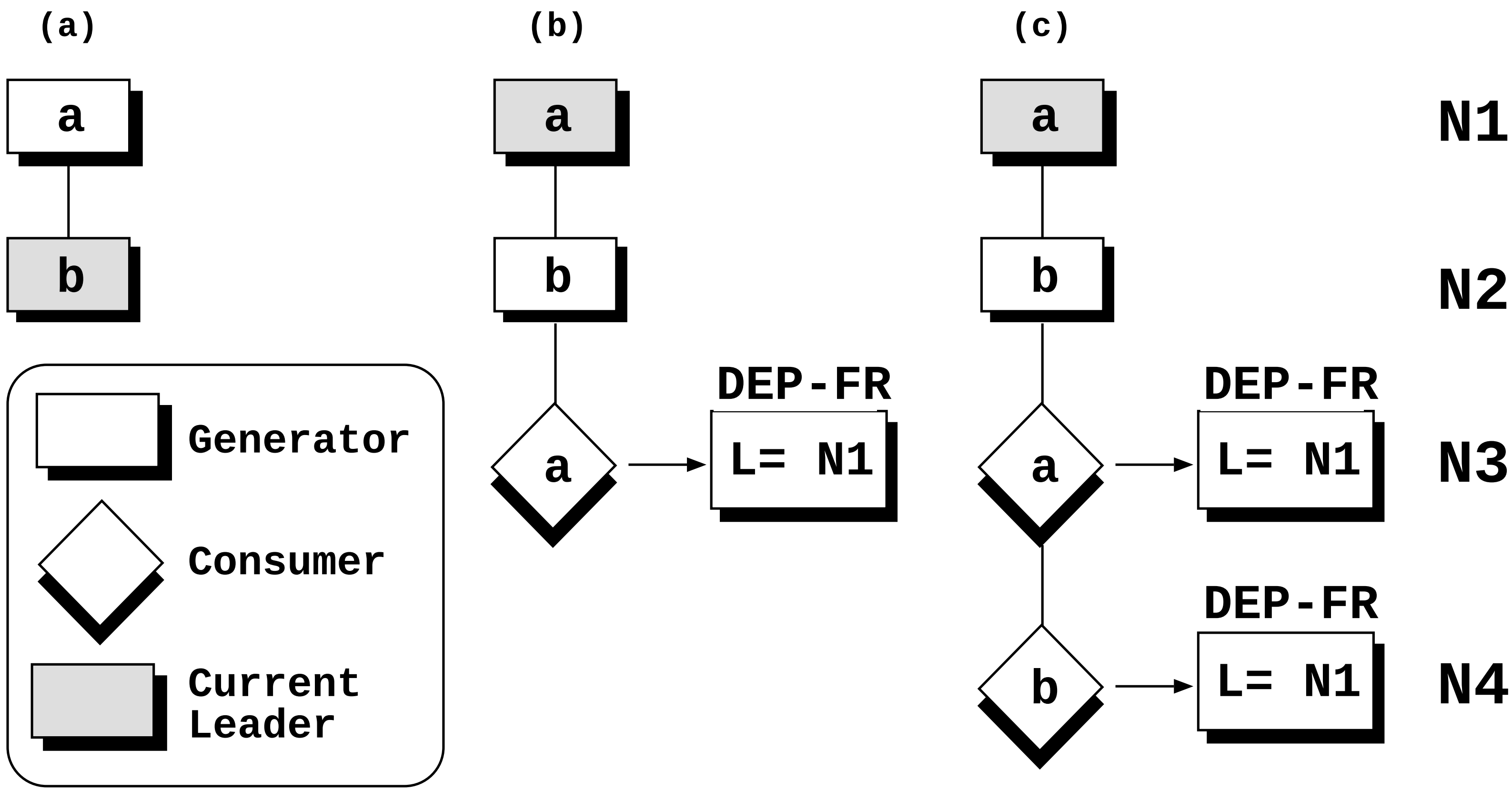
- Public leader node
- SCC suspension

Which is the Leader Node?



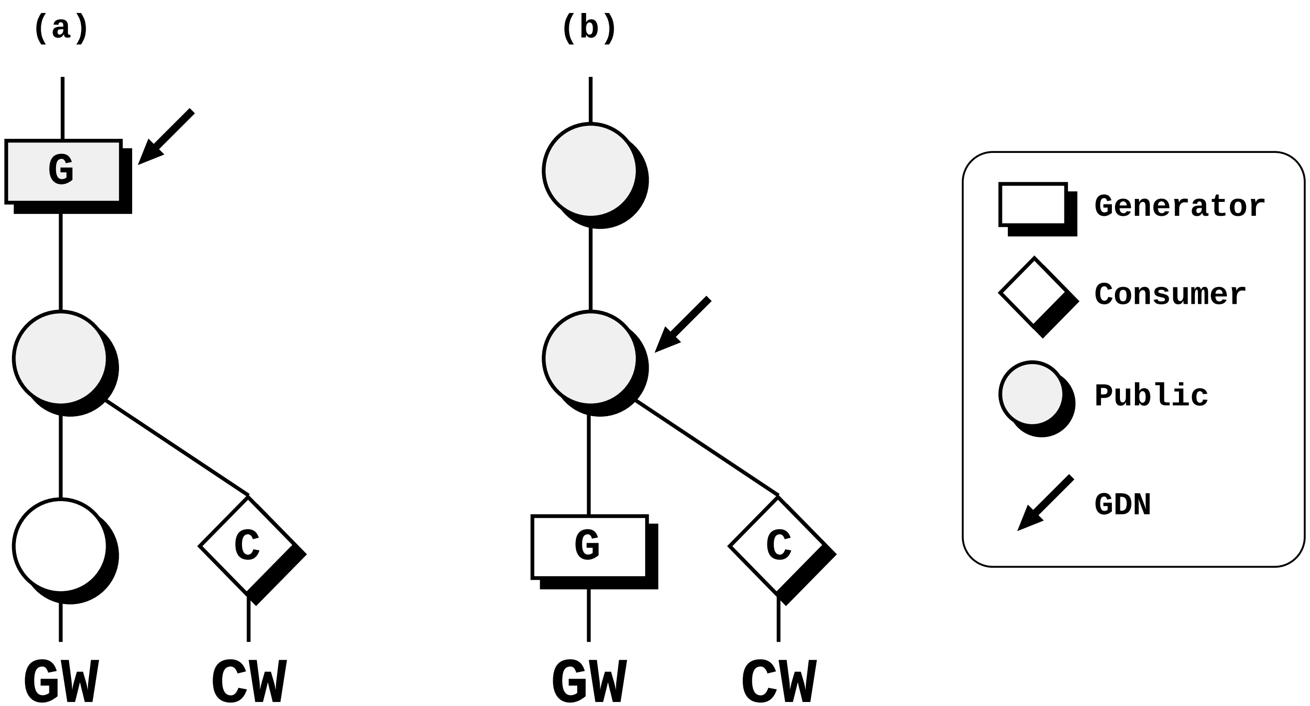
Computing the Leader Node Information

Key Idea: the youngest dependency frame always holds the current leader node.



The Generator Dependency Node (GDN) Concept

A GDN is defined as the youngest node $\mathcal{D}$ on the current branch of a consumer node $\mathcal{C}$, that is an ancestor of the generator node $\mathcal{G}$ for $\mathcal{C}$.



Its purpose is to signal the nodes that are candidates to be leader nodes.

Completion in Public Leader Nodes

Completion Conditions

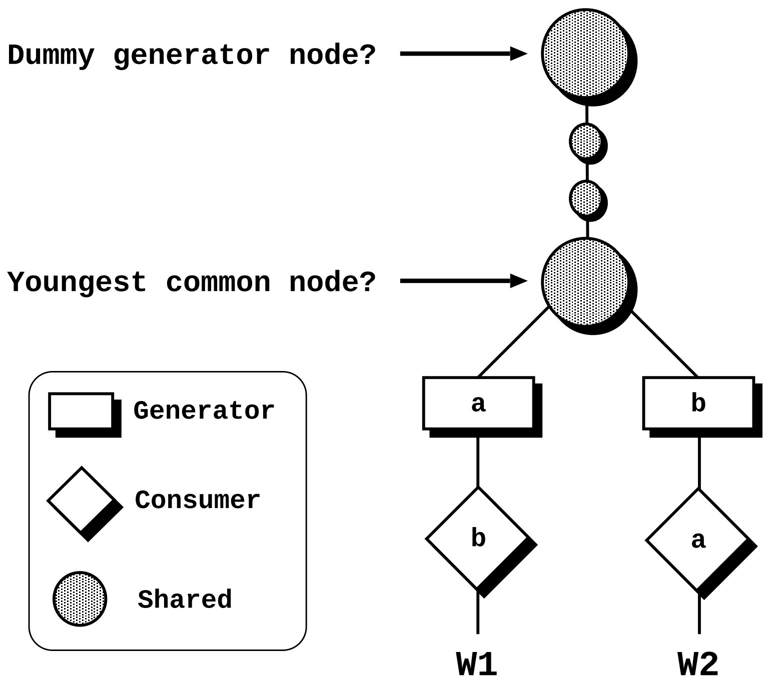
- No unconsumed answers in the leader's SCC.
- Be the single worker owning the leader node.

Problem

Other workers can still influence the leader's SCC.

Solution

Suspend the SCC.



Performance Evaluation

Program	Yap	YapOr (1)	YapTab	OPTYap (1)	XSB
cubes	1.97	2.06(1.05)	2.05(1.04)	2.16(1.10)	4.81(2.44)
ham	4.04	4.61(1.14)	4.28(1.06)	4.95(1.23)	10.36(2.56)
map	9.01	10.25(1.14)	9.19(1.02)	11.08(1.23)	24.11(2.68)
nsort	33.05	37.52(1.14)	35.85(1.08)	39.95(1.21)	83.72(2.53)
puzzle	2.04	2.22(1.09)	2.19(1.07)	2.36(1.16)	4.97(2.44)
queens	16.77	17.68(1.05)	17.58(1.05)	18.57(1.11)	36.40(2.17)
Average		(1.10)	(1.05)	(1.17)	(2.47)

Execution time (in seconds) on non-tabled programs.

- Same compilation flags for Yap, YapOr, YapTab and OPTYap.
- XSB with the default confi guration and execution parameters.
- Results obtained on a Silicon Graphics Cray Origin2000 parallel computer, 96 MIPS 195 MHz R10000 processors, 256 MBytes each (24 GBytes total), IRIX 6.5.12 kernel.

Performance Evaluation

Program	**YapTab**	**OPTYap (1)**	**XSB**
sieve	235.31	268.13(1.14)	433.53(1.84)
leader	76.60	85.56(1.12)	158.23(2.07)
iproto	20.73	23.68(1.14)	53.04(2.56)
samegen	23.36	26.00(1.11)	37.91(1.62)
lgrid	3.55	4.28(1.21)	7.41(2.09)
lgrid/2	59.53	69.02(1.16)	98.22(1.65)
rgrid/2	6.24	7.51(1.20)	15.40(2.47)
Average		(1.15)	(2.04)

Execution time (in seconds) on tabled programs.

Performance Evaluation

Program	**Number of Workers**				
	4	**8**	**16**	**24**	**32**
sieve	3.99	7.97	15.87	23.78	31.50
leader	3.98	7.92	15.78	23.57	31.18
lgrid/2	3.63	7.19	13.53	19.93	24.35
samegen	3.72	7.27	13.91	19.77	24.17
iproto	3.05	5.08	9.01	8.81	7.21
Average	3.67	7.09	13.62	19.17	23.68
rgrid/2	0.94	1.15	0.72	0.77	0.65
lgrid	0.65	0.68	0.55	0.46	0.39
Average	0.80	0.92	0.64	0.62	0.52

Speedups for OPTYap on tabled programs.

Conclusions

- We presented the design and implementation of OPTYap, a first parallel tabling engine for logic programming systems.
- First results show that OPTYap introduces low overheads for sequential execution.
- Parallel execution of tabled programs showed superb speedups for a well known application, and quite good results globally.
- Further work:
  - Improve scheduling;
  - Improve concurrency in table access;
  - Experiment with more applications.