YapOr: an Or-Parallel Prolog System based on Environment Copying

Ricarado Rocha Fernando Silva Vítor Santos Costa
{*ricroc,fds,vsc*}*@ncc.up.pt*

DCC-FC & LIACC
University of Porto
Portugal

Summary

Introduction
Logic Programming and Or-Parallelism

The Environment Copying Model
Basic execution model and the Incremental Copying technique

Extending Yap Prolog to support YapOr
Memory organization, choice points and or-frames

Performance Evaluation
Execution times, speedups and overheads

Conclusions

Logic Programming and Parallelism

Why parallel implementations

- Declarativeness of the language
- Execution model allows parallelism to be exploited implicitly
- Efficiency of sequential implementations

Main forms of implicit parallelism present in logic programs

- Or-Parallelism
- And-Parallelism
  - Independent
  - Dependent

```
a(X,Y) :- b(X), c(Y).
a(X,Y) :- d(X,Y), e(Y).
a(X,Y) :- f(X,Z), g(Z,Y).
```

Or-Parallelism

Main Problems

- Variable binding representation
- Scheduling

Successful Execution Models and Systems

- Binding Arrays / Aurora System
- Environment Copying / Muse System

Question?
The good results previously obtained with Aurora and Muse are repeatable with other Prolog systems in modern parallel machines?

The Environment Copying Model

Basic Execution Model

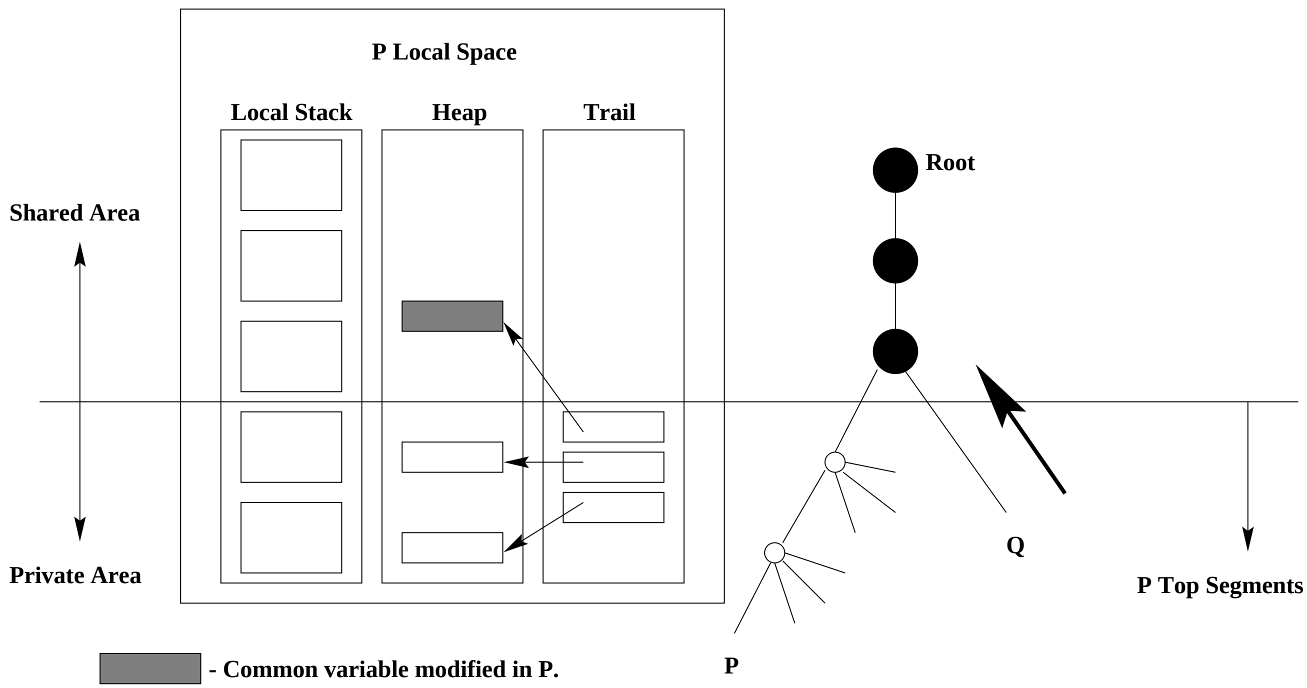
- A parallel execution is performed by a set of workers, initially all but one are idle;
- Whenever a worker executes a predicate with several execution alternatives it creates a choice point;
- As soon the idle workers finds that there is available work in the system, they will request for that work from the busy workers;
- The busy worker synchronizes its computation state with the idle one through the sharing work operation;
- At some point, a worker will fully explore its branch and become idle again;
- Eventually the execution tree will be fully explored and all workers became idle.

Incremental Copying

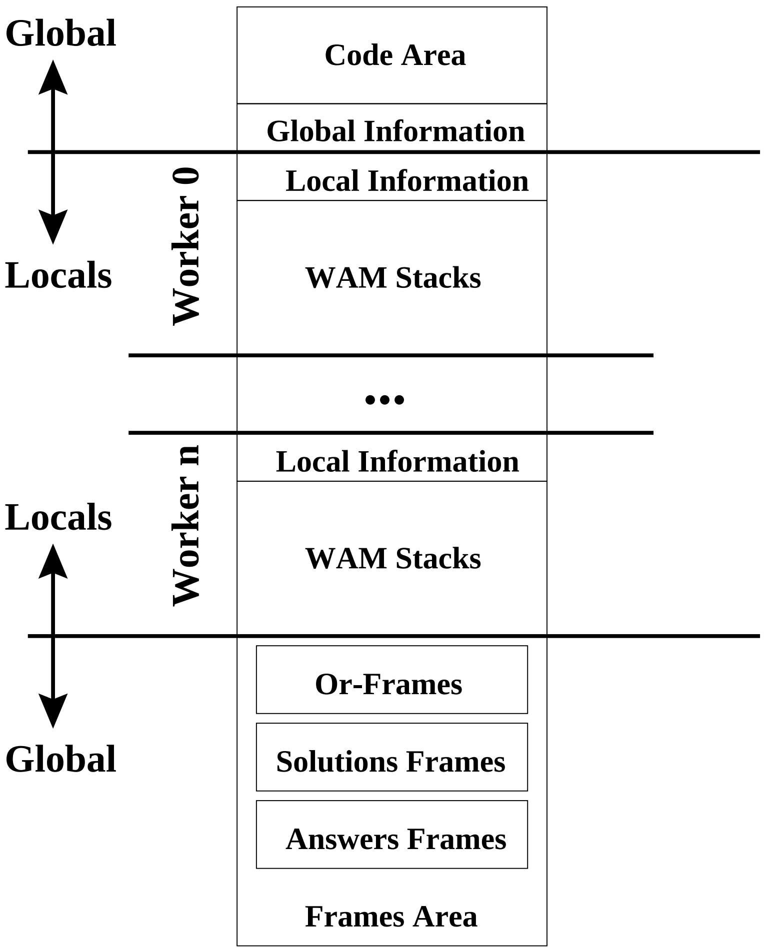
Goal: Position the workers involved in the operation in the same computational state.

Problem: Copying stacks between workers poses a major overhead to the system.

Solution: Keep the parts that are consistent and only copy the differences.



Memory Organization

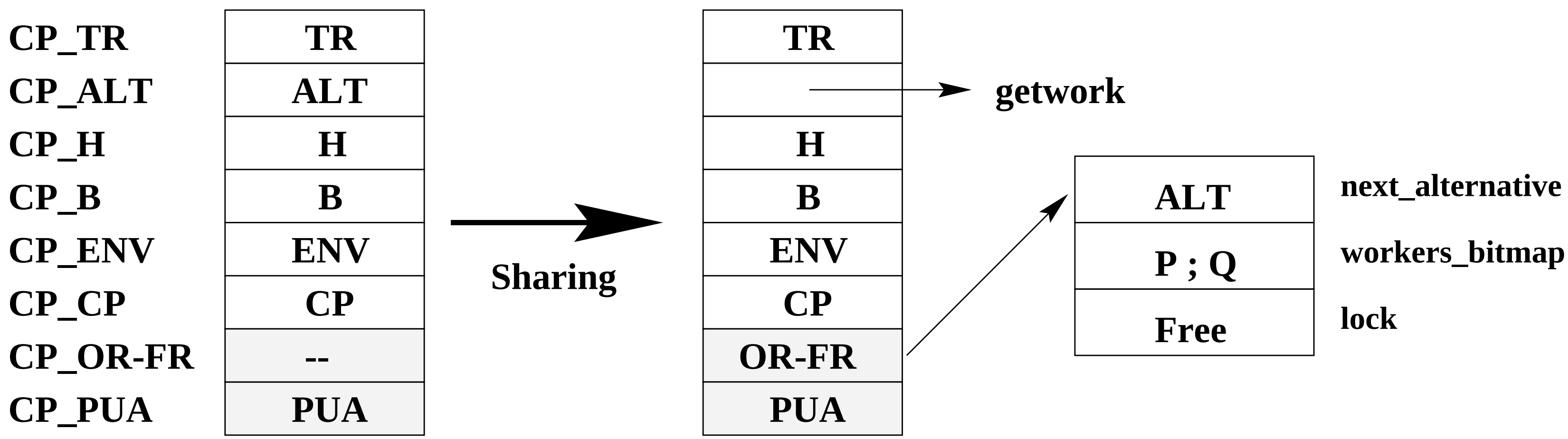


Question?
How to map the local memory in order to meet the requirements of Incremental Copying?

- The starting worker asks for shared memory in the system's initialization phase.
- The remaining workers are created and inherit the addressing space previously mapped.
- Each new worker rotates the local spaces in such a way that all workers will see their own spaces at the same address.

Choice Points and Or-Frames

Problem: Synchronize access to shared choice points.



Solutions

- Store the alternative pointer in a shared structure.
- Use a pseudo-instruction to synchronize access to the untried alternatives.

YapOr Performance Evaluation

Programs	Number of Workers					
	1	2	4	6	7	8
puzzle	10.042	4.835(2.08)	2.316(4.34)	1.550(6.48)	1.339(7.50)	1.172(8.57)
9-queens	4.085	2.047(2.00)	1.026(3.98)	0.690(5.92)	0.596(6.85)	0.519(7.87)
ham	1.802	0.908(1.98)	0.474(3.80)	0.324(5.56)	0.281(6.41)	0.245(7.36)
5cubes	1.029	0.516(1.99)	0.260(3.96)	0.181(5.69)	0.170(6.05)	0.145(7.10)
8-queens2	1.063	0.606(1.75)	0.288(3.69)	0.202(5.26)	0.159(6.69)	0.149(7.13)
8-queens1	0.450	0.225(2.00)	0.118(3.81)	0.080(5.63)	0.072(6.25)	0.067(6.72)
nsort	2.089	1.191(1.75)	0.609(3.43)	0.411(5.08)	0.354(5.90)	0.315(6.63)
sm*10	0.527	0.274(1.92)	0.158(3.34)	0.128(4.12)	0.118(4.47)	0.115(4.58)
db5*10	0.167	0.099(1.69)	0.065(2.57)	0.068(2.46)	0.060(2.78)	0.061(2.74)
db4*10	0.133	0.079(1.68)	0.056(2.38)	0.055(2.42)	0.052(2.56)	0.060(2.22)
YapOr Σ	21.387	10.780(1.98)	5.370(3.98)	3.689(5.80)	3.201(6.68)	2.848(7.51)
YapOr Average		(1.88)	(3.53)	(4.86)	(5.55)	(6.09)

All evaluations performed on a Sun SparcCenter 2000 with 8 processors,
256 MBytes of main memory, two level cache and running SunOS 5.6.

Muse Performance Evaluation

Programs	Number of Workers					
	1	**2**	**4**	**6**	**7**	**8**
puzzle	12.120	6.660(1.82)	3.720(3.26)	2.670(4.54)	2.230(5.43)	2.140(5.66)
9-queens	3.890	2.030(1.92)	1.110(3.54)	0.690(5.64)	0.630(6.17)	0.560(6.95)
ham	2.550	1.480(1.72)	0.820(3.11)	0.520(4.90)	0.520(4.90)	0.460(5.54)
5cubes	1.130	0.560(2.02)	0.280(4.04)	0.180(6.28)	0.160(7.06)	0.150(7.53)
8-queens2	1.350	0.690(1.96)	0.390(3.46)	0.270(5.00)	0.240(5.63)	0.220(6.14)
8-queens1	0.550	0.290(1.90)	0.160(3.44)	0.120(4.58)	0.110(5.00)	0.100(5.50)
nsort	2.650	1.450(1.83)	0.810(3.27)	0.550(4.82)	0.510(5.20)	0.450(5.89)
sm*10	0.670	0.360(1.86)	0.220(3.05)	0.170(3.94)	0.160(4.19)	0.150(4.47)
db5*10	0.190	0.110(1.73)	0.080(2.38)	0.070(2.72)	0.070(2.72)	0.070(2.72)
db4*10	0.160	0.090(1.78)	0.060(2.67)	0.070(2.29)	0.060(2.67)	0.070(2.29)
Muse Σ	25.260	13.720(1.84)	7.650(3.30)	5.310(4.76)	4.690(5.39)	4.370(5.78)
Muse Average		(1.85)	(3.22)	(4.47)	(4.90)	(5.27)

Parallel Execution Overheads

	Number of Workers					
Activity	**1**	**2**	**4**	**6**	**7**	**8**
puzzle						
Prolog	100.00	99.95	99.56	99.20	99.02	98.68
Search	0.00	0.02	0.16	0.32	0.41	0.60
Sharing	0.00	0.02	0.17	0.32	0.38	0.50
Get-Work	0.00	0.01	0.10	0.17	0.19	0.23
Cut	0.00	0.00	0.00	0.00	0.00	0.00
sm						
Prolog	100.00	97.68	86.71	74.56	69.08	63.29
Search	0.00	0.81	5.02	11.50	13.85	16.87
Sharing	0.00	0.86	5.64	10.17	13.14	15.76
Get-Work	0.00	0.61	2.51	3.52	3.61	3.88
Cut	0.00	0.04	0.13	0.25	0.32	0.20

Conclusions

Presentation

- We presented YapOr, an or-parallel Prolog system based on environment copying.
- YapOr has good sequential and parallel performance on a large set of benchmarks.
- The good performance is explained by the fact that for most benchmarks YapOr spends its time mainly executing reductions and not managing parallelism.

Current and Further Work

- We are now working on adjusting YapOr to support parallel tabling execution.
- So far, we have extended Yap to execute sequential tabling and to support another two or-parallel models: SBA and αCOW.

Download Yap Prolog: `www.ncc.up.pt/~vsc/Yap`