Thor: Threads and Or-Parallelism Unified

Vítor Santos Costa
Inês Dutra
Ricardo Rocha

DCC/FCUP and
CRACS/INESC Porto LA

Overview

- Motivation
- Background
- Thor
- Evaluation
- Conclusions and Future Work

Computing is Changing

- Parallelism is Important
- Multi-cores:
  - ⋆ 2-cores is now standard
  - ⋆ 4-cores common
  - ⋆ 6-cores entered desktop
  - ⋆ servers with 24, 32, . . .
- Distributed Computing
- GPUs

Motivation

- Parallelism is easy in LP
- Implicit Parallelism:
  - ⋆ Or-Parallelism
  - ⋆ And-Parallelism
  - ⋆ Combined
- Explicit Parallelism
  - ⋆ MPI Interfaces
  - ⋆ Threads

Issues with Parallelism

1. Parallel Machines were not widely available
2. Significant changes to the underlying Prolog engine:
   - hard to maintain
3. Large Prolog Programs are hard to Parallelize
   - Need to support actual applications

Thor

- Designed for multi-cores
- Relies on a combination of:
  - ⋆ Recent Work on Thread Libraries (SWI, ciao, XSB, YAP)
  - ⋆ Old work on or-parallelism (MUSE, YapOr)
- Is this approach feasible?

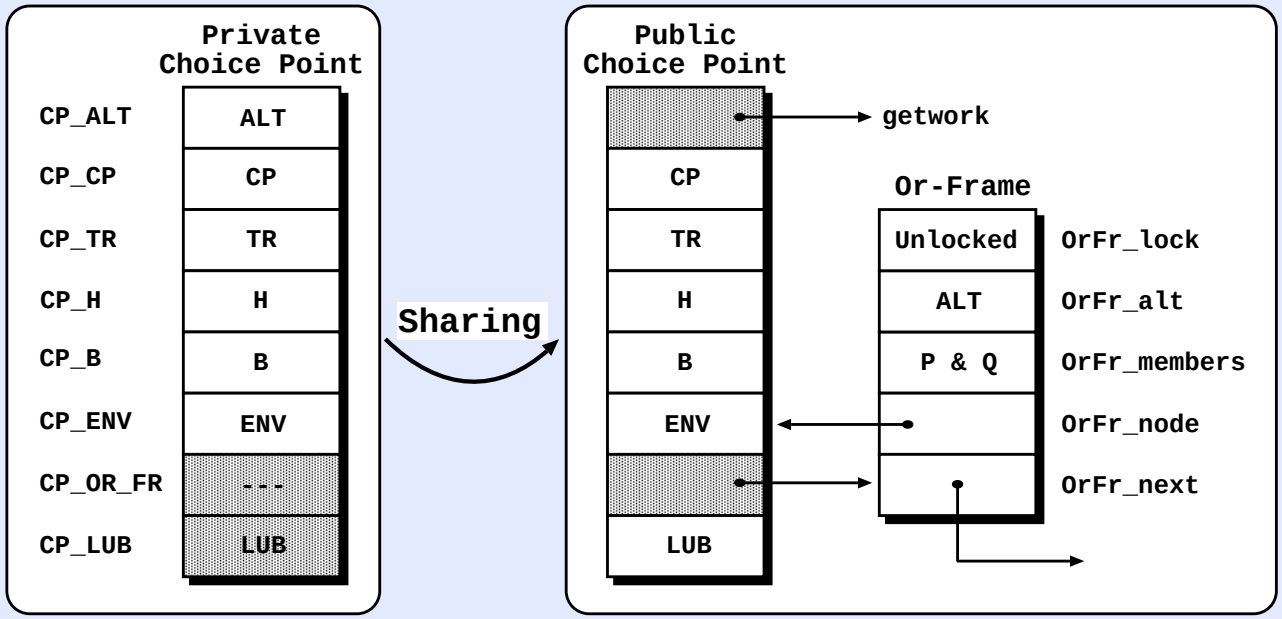
Review: Thread Libraries

- YAP Threads follows SWI approach
- Separate Engines:
  - ⋆ No Term sharing
  - ⋆ No global variable sharing
- DataBase shared by default
  - ⋆ Private predicates are allowed
- Locking and Message-Passing Primitives:
  - ⋆ can be used to construct parallel engines

Review: YapOr

- Follows Muse
- Multi Sequential Model
- Workers keep copies of the search tree
- They synchronise by *sharing*
  1. Choice-Points are made public;
  2. Execution stacks are copied
  3. The sharer continues forward execution;
  4. The requester backtracks.

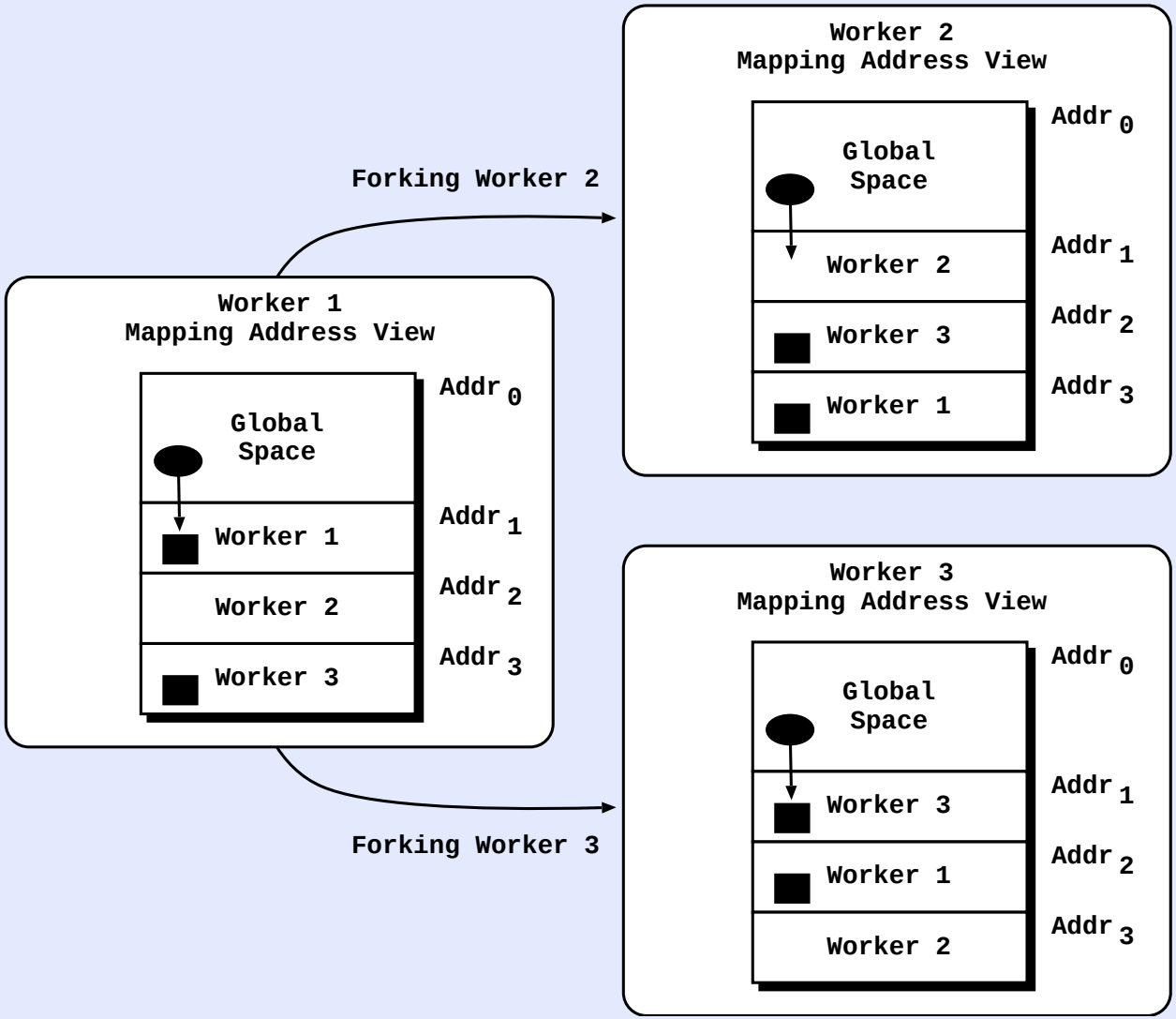
Review: Sharing



Review: YapOr

- *Incremental Copying*: just copy increments
  - ⋆ requires searching the cells
- Bottom-most scheduling
  - ⋆ Stack-splitting was also supported.
- Double Map of Memory:
  - ⋆ All stacks are mapped at *two addresses*
  - ⋆ One for work
  - ⋆ The other for sharing
  - ⋆ Requires a Process Model

Review: YapOr



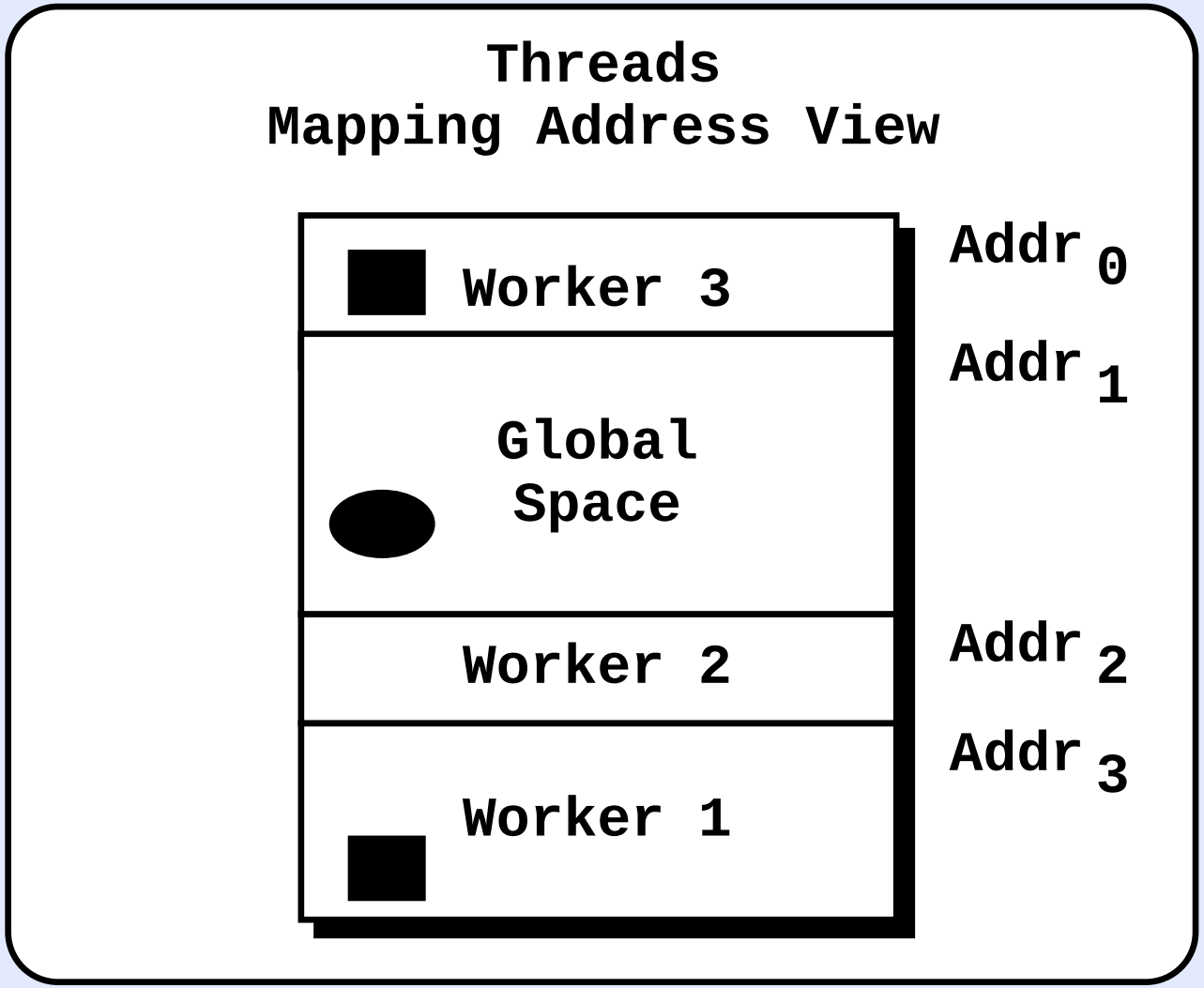
Thor

- Parallel workers are threads
  - ⋆ but threads may not be workers!
- We shall use copying:
  - ⋆ Independent stacks
  - ⋆ Copy contiguous chunks
  - ⋆ Less engine intrusive
  - ⋆ Seems to fit well with thread design.

Thor: Shifted Copying

- Use YAP's stack shifter
- To adjust addresses *after copying*
- Allows:
  - ⋆ Using current thread engine
  - ⋆ Different stack sizes
- Allows incremental copying

Thor: Shifted Copying



Thor: Work Sharing Protocol

P_SHARE (p, q) {
  $share\_private\_nodes(q)$
  $signal[q] =$ `nodes_shared`
  wait $(signal[q] ==$ `ready`$)$
}

Q_SHARE (p, q) {
  wait $(signal[q] \neq$ `ready`$)$
  $copy\_registers(p, q)$
  if $(INCREMENTL\_COPY)$
    $copy\_stacks(p, q, deltas)$
    $copy\_trailed\_entries(q, p)$
    $signal[q] =$ `ready`
    $adjust\_stacks(deltas)$
  else
    $copy\_stacks(p, q, full\_stacks)$
    $signal[q] =$ `ready`
    $adjust\_stacks(full\_stacks)$
  endif
}

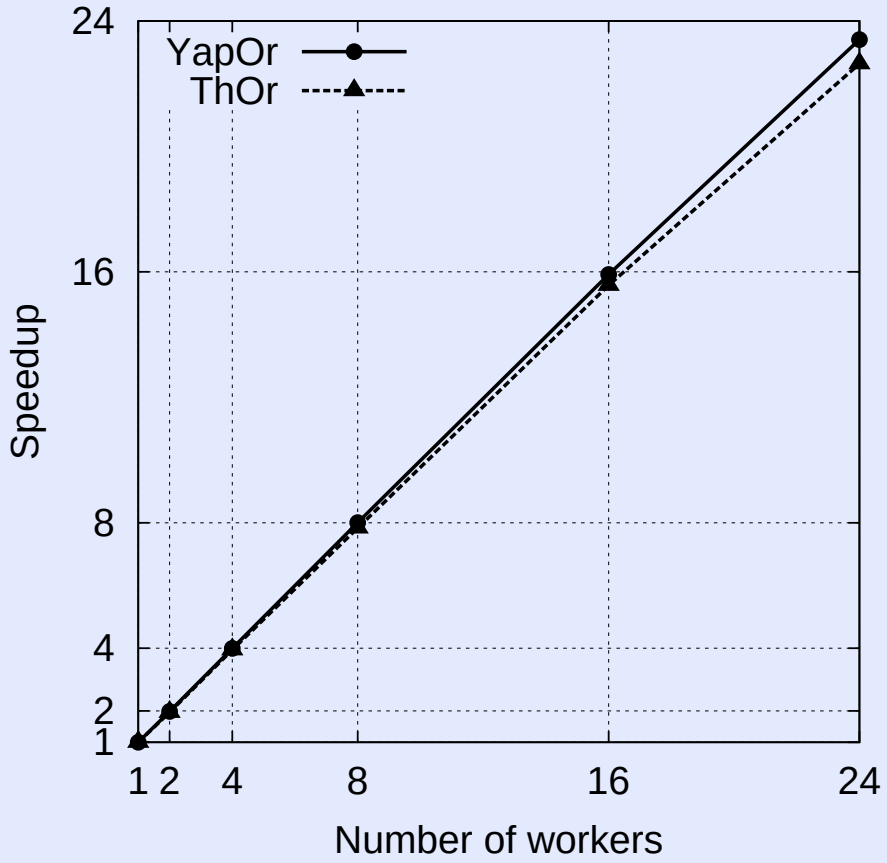
Thor: Scheduling

- Thor server thread creates initial threads:
  - ⋆ Right now thread 0
- Each Thor client thread starts at an `getwork` engine
- Waits until work becomes available
- We reuse YapOr scheduler, as is
- Cut:
  - ⋆ We reuse YapOr

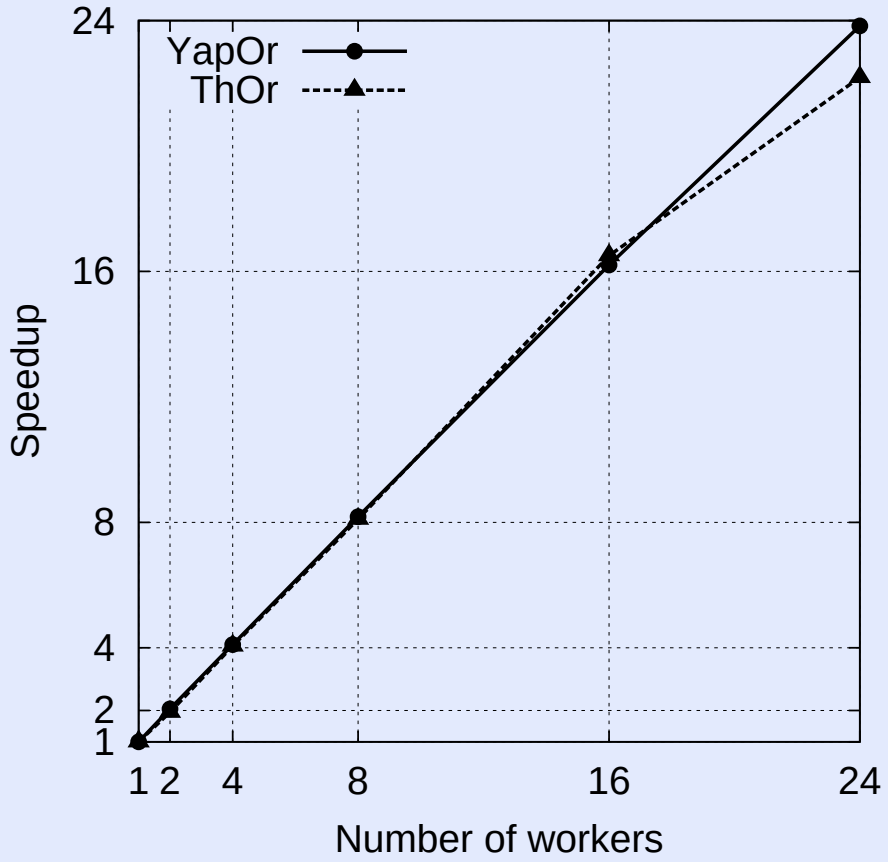
Thor: Overheads

Benchmark	Quad-Core			4 Six-Core		
	YAP	ThOr	YapOr	YAP	ThOr	YapOr
cubes	0.11	0.11 (1.00)	0.11 (1.00)	0.20	0.23 (1.15)	0.20 (1.00)
fp	1.47	2.36 (1.61)	1.71 (1.16)	2.51	3.29 (1.31)	2.67 (1.06)
ham	0.15	0.29 (1.93)	0.19 (1.27)	0.33	0.46 (1.36)	0.34 (1.03)
magic	25.07	27.55 (1.10)	27.80 (1.11)	40.29	48.88 (1.21)	41.16 (1.02)
map	12.20	20.25 (1.66)	14.60 (1.20)	24.06	30.45 (1.26)	23.94 (0.99)
mapbigger	33.01	55.26 (1.67)	39.63 (1.20)	64.46	81.09 (1.25)	65.90 (1.02)
puzzle	0.08	0.13 (1.63)	0.08 (1.00)	0.15	0.20 (1.34)	0.17 (1.13)
puzzle4x4	6.02	7.18 (1.19)	6.47 (1.07)	9.17	10.34 (1.12)	9.38 (1.02)
queens	21.79	24.54 (1.13)	24.12 (1.11)	48.10	51.63 (1.07)	48.73 (1.01)
Average		(1.44)	(1.12)		(1.23)	(1.03)

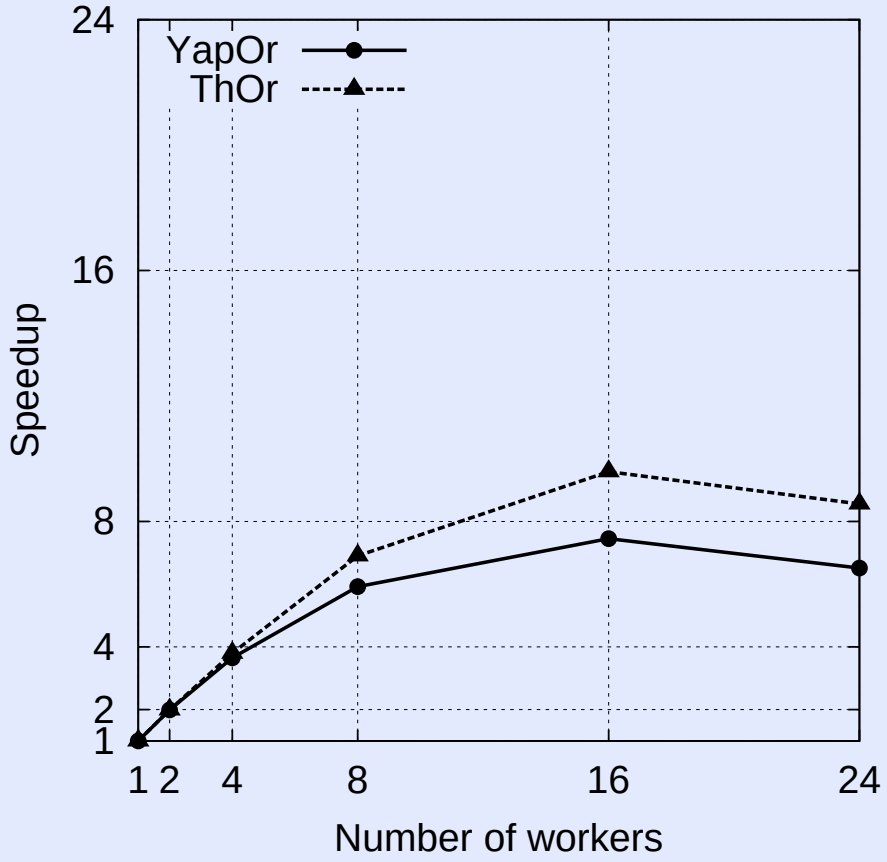
Thor: Queens



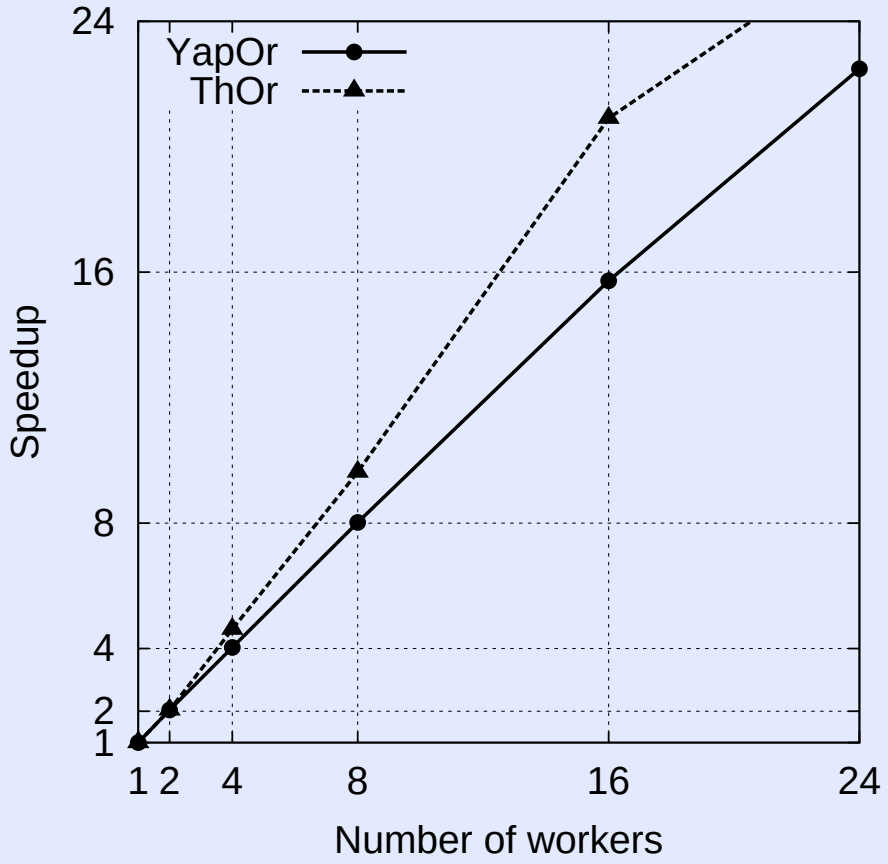
Thor: Map Coloring



Thor: Cubes



Thor: Map



Thor: “Real Machines”

Benchmark	OS X		Windows Vista	
	1	2	1	2
`cubes`	0.103	0.056 (1.84)	0.156	0.078 (2.00)
`fp`	1.930	1.040 (1.86)	3.500	1.790 (1.96)
`ham`	0.220	0.110 (2.00)	0.530	0.280 (1.89)
`magic`	27.500	15.000 (1.83)	30.700	15.600 (1.97)
`map`	15.800	9.000 (1.76)	33.500	17.700 (1.89)
`mapbigger`	43.500	23.500 (1.85)	92.900	49.000 (1.90)
`puzzle`	0.100	0.055 (1.82)	0.220	0.110 (2.00)
`puzzle4x4`	6.900	3.700 (1.86)	8.980	4.600 (1.95)
`queens`	24.100	13.200 (1.83)	29.700	15.000 (1.98)
Average		(1.85)		(1.94)

Thor: Real World

- Sequential and Parallel Predicates
- Locking
- Global Variables:
  - ⋆ Important for constraint systems
  - ⋆ `b_`: naturally belong to a stack
  - ⋆ `nb_`: shared between stacks?

Conclusions

- Thor Works!
- Great Speedups:
  - ⋆ Or-Parallelism works well with current multi-cores
- Shifted Copying does not impose much overheads:
  - ⋆ Thanks to Incremental Copying
  - ⋆ and Coarse-Grained Work

Future work

- Just the beginning!
- Global variables and constraint programs
- Better integration with thread packages:
  - ⋆ or-parallelism should be seen as an engine
- Avoid Implicit vs explicit:
  - ⋆ Both have a role!