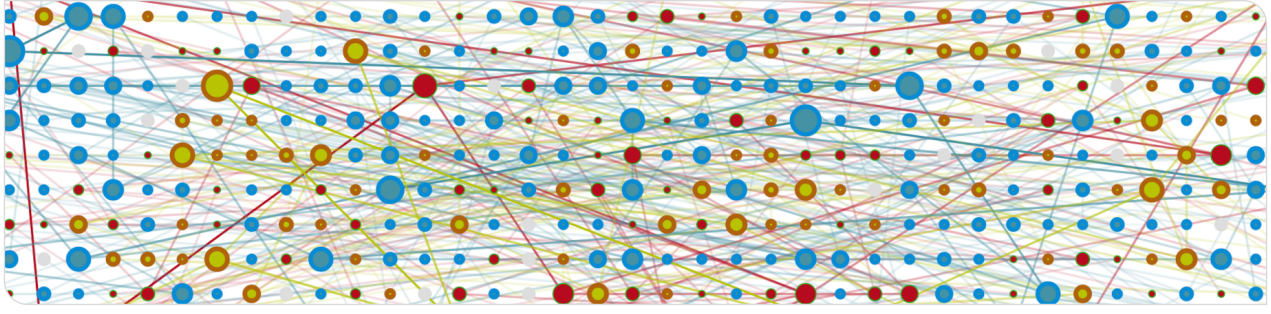
Karlsruhe Institute of Technology

Scalable SAT Solving in the Cloud

24th International Conference on Theory & Practice of Satisfiability Testing

Dominik Schreiber, Peter Sanders | July 9, 2021

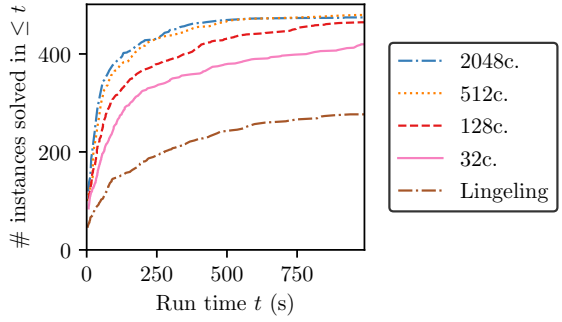


Motivation

Massively parallel SAT solving . . .

- Decent speedups for many industrial instances
- More cores → less resource-efficient

  ⇒ Failure to scale beyond ∼ 500 cores



Extracted from Balyo et al. (2015)

Motivation

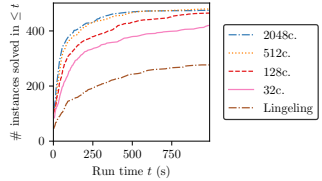
Massively parallel SAT solving . . .

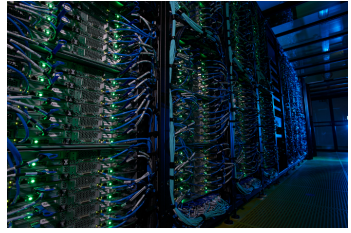
- Decent speedups for many industrial instances
- More cores → less resource-efficient

  ⇒ Failure to scale beyond ∼ 500 cores

. . . As A Service?

- High Performance Computing environment ($>$ 1000 cores)
- Many users at once: Job processing on demand
- Need for low latencies, quick response times





https://wiki.scc.kit.edu/hpc/index.php?title=Category:ForHLR

Our Contributions

1. Framework **Mallob**

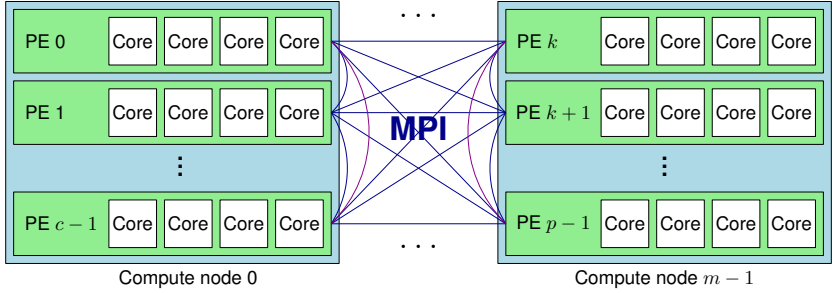
   $\Rightarrow$ Job scheduling & load balancing platform

2. **Mallob SAT engine** (a.k.a. *Mallob-mono*)

   $\Rightarrow$ Scalable distributed SAT solver based on HordeSat

3. Combination: Scalable resolution of SAT jobs on demand

Mallob: System Architecture



Communication between PEs (*Processing Elements*) via Message Passing Interface (MPI)

Mallob: Job Scheduling

Karlsruhe Institute of Technology

Animated Illustration @ `https://dominikschreiber.de/animallob`

Mallob SAT Engine

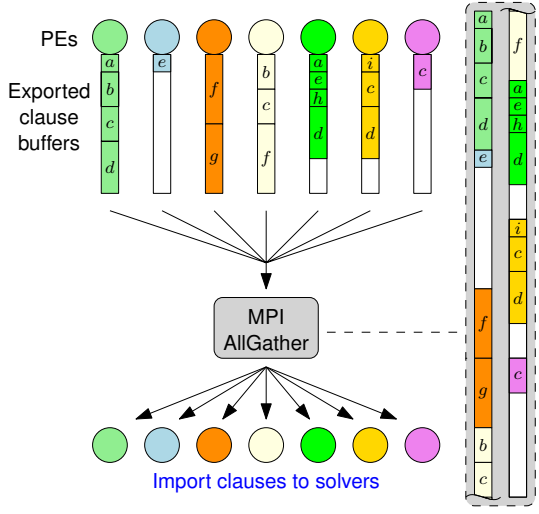
	HordeSat (Balyo et al. 2015)	**Mallob SAT Engine**
Environment	Set of PEs fixed at program start	Set of PEs can grow/shrink (Malleability)
Communication	Synchr. collective operations of MPI	Asynchronous routing through job tree
Core solvers	(P)Lingeling (Biere 2014)	(P)Lingeling + YalSAT (Biere 2018)
Mode of execution	Solver threads in main (MPI) process	Solver threads in separate child process

+ numerous performance improvements (lock-free clause import, memory awareness, . . .)

Clause Exchange in HordeSat

Periodic collective operation **AllGather**

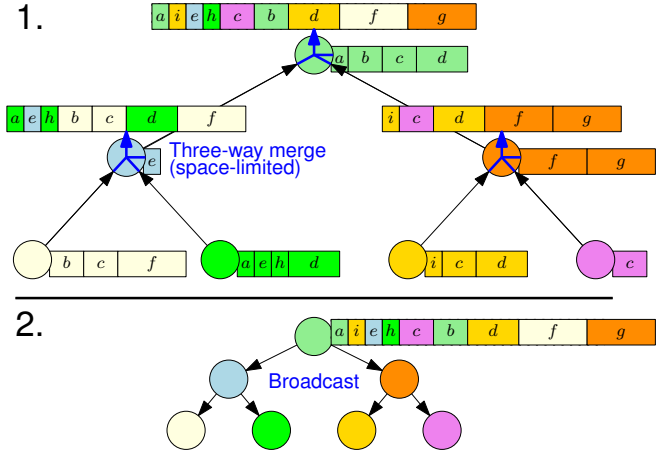
- Duplicate clauses
- “Holes” carrying no information
- Buffer grows proportionally with num. PEs
  $\Rightarrow$ Bottleneck w.r.t communication *and* local work



Clause Exchange in Mallob

Custom collective operation

- Malleable: Realized through job tree
- Detect duplicates during merge
- Result is of compact shape
- Sublinear buffer size growth:
  Discard longest clauses as necessary



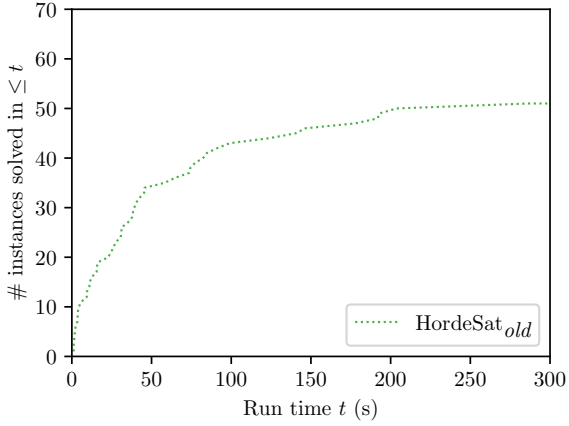
SAT Solving on 2560 Cores (128 Machines)

Setup

- 80 selected instances from ISC 2020
- 300 s per instance
- 5 PEs à 4 threads per machine

Configurations

- HordeSat: *old*, *new* portfolio



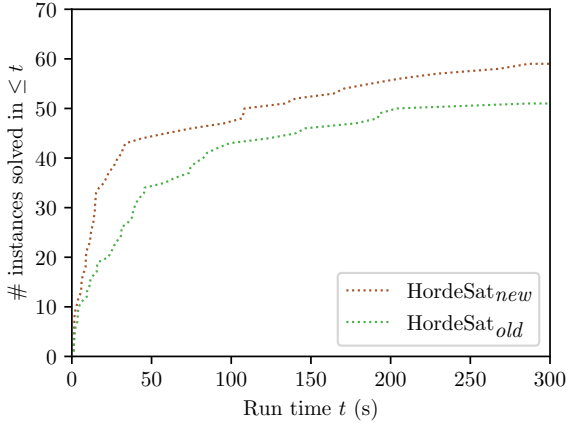
SAT Solving on 2560 Cores (128 Machines)

Setup

- 80 selected instances from ISC 2020
- 300 s per instance
- 5 PEs à 4 threads per machine

Configurations

- HordeSat: *old*, *new* portfolio



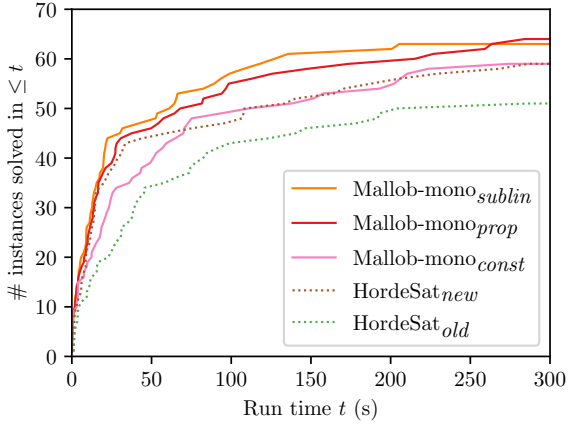
SAT Solving on 2560 Cores (128 Machines)

Setup

- 80 selected instances from ISC 2020
- 300 s per instance
- 5 PEs à 4 threads per machine

Configurations

- HordeSat: *old*, *new* portfolio
- *const*, *sublin*, *prop* : constant / sublinear / proportional clause buffer size in # PEs



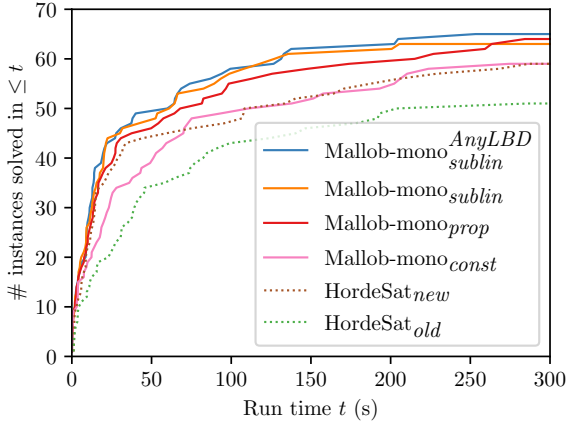
SAT Solving on 2560 Cores (128 Machines)

Setup

- 80 selected instances from ISC 2020
- 300 s per instance
- 5 PEs à 4 threads per machine

Configurations

- HordeSat: *old*, *new* portfolio
- *const*, *sublin*, *prop* : constant / sublinear / proportional clause buffer size in # PEs
- *AnyLBD* : Drop HordeSat's successively increasing LBD limit on shared clauses



Scaling Experiments

Mallob-mono$^{AnyLBD}_{sublin}$ vs. HordeSat$_{new}$

Speedups

Instance F solved by parallel approach
$\Rightarrow$ Par. run time $T_{par}(F) \leq 300\,s$
$\Rightarrow$ Seq. run time $T_{seq}(F) \leq 50\,000\,s$
($T_{seq}(F) := 50\,000\,s$ if unsolved)

Total speedup S_{tot}:
$\sum_F T_{seq}(F) \;/\; \sum_F T_{par}(F)$

Median speedup S_{med}:
$median_F\{T_{seq}(F)/T_{par}(F)\}$

Scaling Experiments

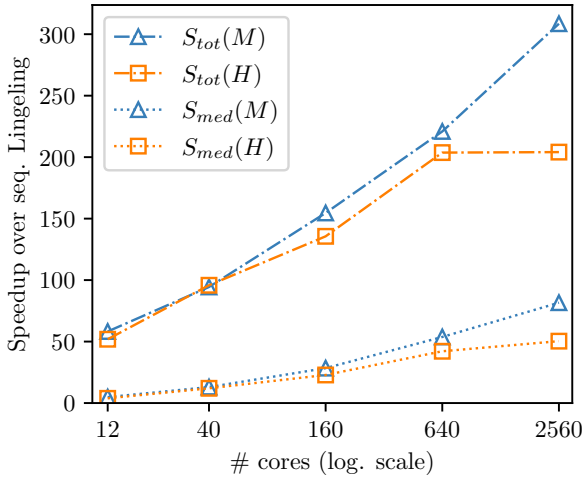
Mallob-mono$^{AnyLBD}_{sublin}$ vs. HordeSat$_{new}$

Speedups

Instance F solved by parallel approach
$\Rightarrow$ Par. run time $T_{par}(F) \leq 300\,s$
$\Rightarrow$ Seq. run time $T_{seq}(F) \leq 50\,000\,s$
($T_{seq}(F) := 50\,000\,s$ if unsolved)

Total speedup S_{tot}:
$\sum_F T_{seq}(F) \;/\; \sum_F T_{par}(F)$

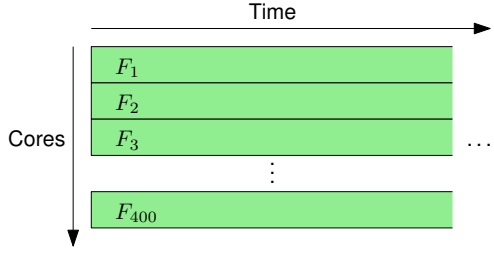
Median speedup S_{med}:
$median_F\{T_{seq}(F)/T_{par}(F)\}$



Solving 400 Formulae on 2560 Cores

400×{Lingeling, Kissat}

- Run 400 sequential SAT solvers



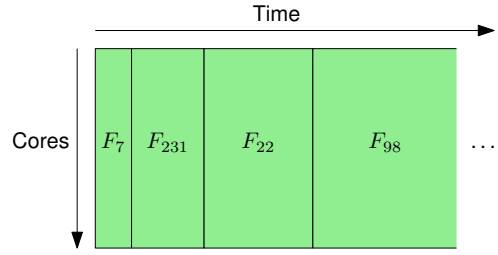
Solving 400 Formulae on 2560 Cores

400×{Lingeling, Kissat}

- Run 400 sequential SAT solvers

Optimal Scheduling of Mallob-mono

- Run Mallob-mono on 2560 cores for each job
- Sort jobs by run time in ascending order



Solving 400 Formulae on 2560 Cores

400×{Lingeling, Kissat}

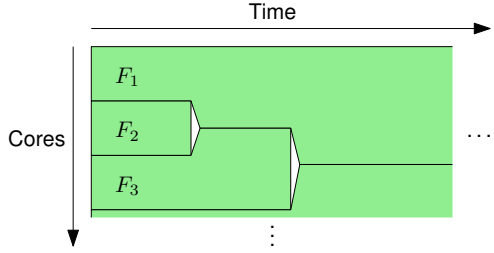
- Run 400 sequential SAT solvers

Optimal Scheduling of Mallob-mono

- Run Mallob-mono on 2560 cores for each job
- Sort jobs by run time in ascending order

Mallob

- Introduce all 400 jobs at system start
- Automatic scheduling & load balancing



Solving 400 Formulae on 2560 Cores

400×{Lingeling, Kissat}

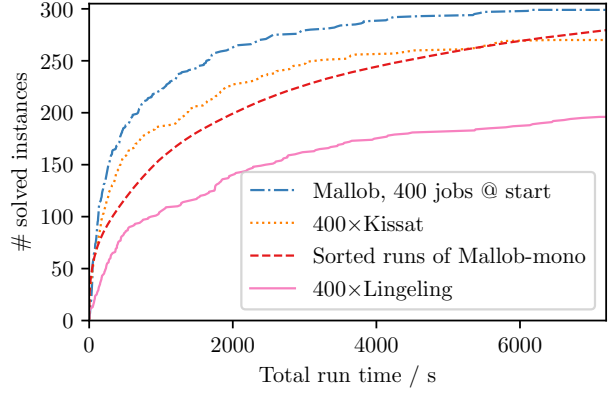
- Run 400 sequential SAT solvers

Optimal Scheduling of Mallob-mono

- Run Mallob-mono on 2560 cores for each job
- Sort jobs by run time in ascending order

Mallob

- Introduce all 400 jobs at system start
- Automatic scheduling & load balancing



Conclusion

- Distributed SAT solving system scaling up to 2.5k cores
- In HPC environments, combine resource-efficiency of parallel job processing with speedups of flexible parallel SAT solving
- Exploit malleability for low scheduling latencies, quick response times

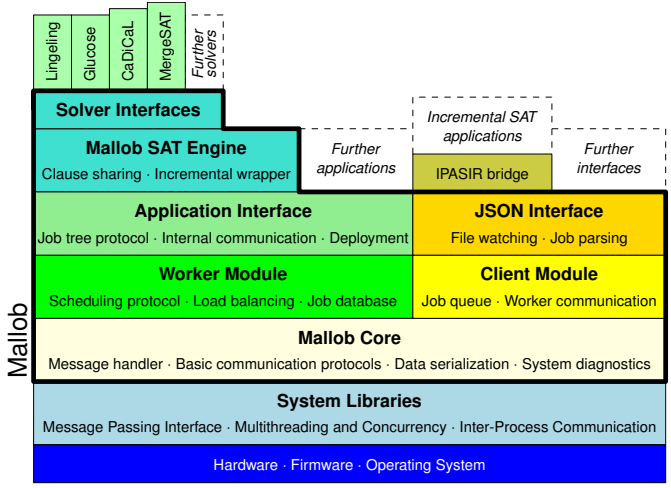
Conclusion

- Distributed SAT solving system scaling up to 2.5k cores
- In HPC environments, combine resource-efficiency of parallel job processing with speedups of flexible parallel SAT solving
- Exploit malleability for low scheduling latencies, quick response times

Work in Progress & Outlook

- Clause re-sharing strategies for malleable SAT solving
- Integration of further SAT solver backends (Glucose, CaDiCaL, MergeSAT, . . .)
- Enable incremental SAT solving for applications like planning, verification

Mallob: Technology Stack



Mallob SAT Engine

Karlsruhe Institute of Technology

Reworked Communication

- Supports malleability: Fluctuating resources during computation
- Succinct, communication-efficient clause exchange

Mallob SAT Engine

Karlsruhe Institute of Technology

Reworked Communication

- Supports malleability: Fluctuating resources during computation
- Succinct, communication-efficient clause exchange

Improved core solvers

- New portfolio, also including stochastic local search
- Lock-free clause import via ring buffers

Mallob SAT Engine

Karlsruhe Institute of Technology

Reworked Communication

- Supports malleability: Fluctuating resources during computation
- Succinct, communication-efficient clause exchange

Improved core solvers

- New portfolio, also including stochastic local search
- Lock-free clause import via ring buffers

Technical features

- JSON API to introduce jobs, receive results
- SAT solver threads in a separate child process
  $\Rightarrow$ Seamless preemption, termination of solvers via OS signals

Scaling Experiments

Total Speedups

$\mathcal{I}_x :=$ Instances solved by parallel approach with x cores

Instance $I \in \mathcal{I}_x$: Parallel run time $T_{\text{par}}(I) \leq 300\,s$, sequential (Lingeling) run time $T_{\text{seq}}(I) \leq 50\,000\,s$

Total speedup: $S_x := \frac{\sum_{i \in \mathcal{I}_x} T_{\text{seq}}(I)}{\sum_{i \in \mathcal{I}_x} T_{\text{par}}(I)}$

Config.	1×3×4	2×5×4	8×5×4	32×5×4		128×5×4
# Cores	12	40	160	640		2560
HordeSat (new)	51.9	**95.8**	135.6	203.7	+0.2% →	204.1
Mallob-mono (best)	**58.2**	94.4	**154.6**	**220.9**	+39.7% →	**308.5**

Full Scaling Results

	All instances					Hard instances					
		Lingeling		Kissat			Lingeling		Kissat		
Config.	#	S_{med}	S_{tot}	S_{med}	S_{tot}	#	S_{med}	S_{tot}	#	S_{med}	S_{tot}
H1×3×4	36	3.84	51.90	2.22	29.55	32	4.39	52.01	31	4.03	32.49
H2×5×4	40	12.00	95.80	5.06	64.44	35	12.27	96.83	33	9.11	69.63
H8×5×4	49	22.83	135.55	9.76	90.08	38	32.00	142.76	32	24.88	105.94
H32×5×4	56	42.12	203.66	15.25	112.14	34	97.61	231.77	19	114.86	208.68
H128×5×4	59	50.35	204.10	17.38	111.46	21	356.33	444.12	10	243.42	375.04
M1×3×4	35	4.83	58.15	3.62	64.66	31	5.37	58.24	30	5.29	66.08
M2×5×4	44	12.98	94.44	10.52	67.71	39	14.37	95.28	37	11.54	69.25
M8×5×4	52	28.38	154.62	12.06	89.61	41	34.29	162.23	34	23.43	106.85
M32×5×4	60	53.75	220.92	23.41	148.57	37	152.19	245.54	23	134.07	262.04
M128×5×4	65	81.60	308.48	25.97	175.58	25	363.32	447.97	12	363.32	483.11

Solving Several Formulae At Once

- Compare Mallob-mono on 32 (8, 2) machines with Mallob with 4 (16, 64) jobs on 128 machines
- Mallob: Keeps 5% of PEs idle for scheduling, employs one "client" PE for introducing jobs
- Same priority, time limit (300 s) for each instance

Approach	Solved	(SAT,	UNSAT)	PAR-2
Mallob $J = 4$	58	26	32	192.7
Mb-mono $m = 32$	**60**	**28**	32	**181.4**
Mallob $J = 16$	**54**	**24**	**30**	**232.7**
Mb-mono $m = 8$	52	23	29	240.1
Mallob $J = 64$	**49**	**21**	**28**	**279.0**
Mb-mono $m = 2$	44	19	25	299.8

Solving Several Formulae At Once

- Compare Mallob-mono on 32 (8, 2) machines with Mallob with 4 (16, 64) jobs on 128 machines
- Mallob: Keeps 5% of PEs idle for scheduling, employs one "client" PE for introducing jobs
- Same priority, time limit (300 s) for each instance

Approach	Solved	(SAT,	UNSAT)	PAR-2
Mallob $J = 4$	58	26	32	192.7
Mb-mono $m = 32$	**60**	**28**	32	**181.4**
Mallob $J = 16$	**54**	**24**	**30**	**232.7**
Mb-mono $m = 8$	52	23	29	240.1
Mallob $J = 64$	**49**	**21**	**28**	**279.0**
Mb-mono $m = 2$	44	19	25	299.8

Results:

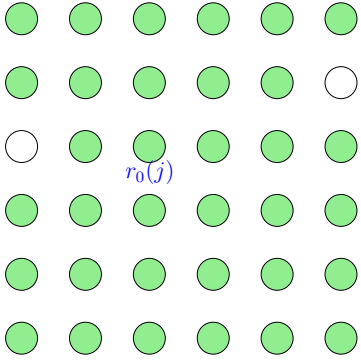
- $J = 4$: Worse performance than Mallob-mono (fewer available PEs)
- $J = 16, 64$: Noticeable improvements! Jobs toward the end receive additional PEs from finished jobs
- Scheduling times: min 0.003 s, average 0.061 s, median 0.006 s, max 0.781 s

Resource Efficiency

System	# solved	core hours	(ch. for solved,	unsolved)
Mallob	299	4378		
Sorted runs of Mallob-mono	270	4378		
	299	7358		
Mallob-mono (ISC'20)[1]	299	29449	7005	22444
P-MCOMSPS-STR-32 (ISC'20)[1]	284	6548	1392	5156

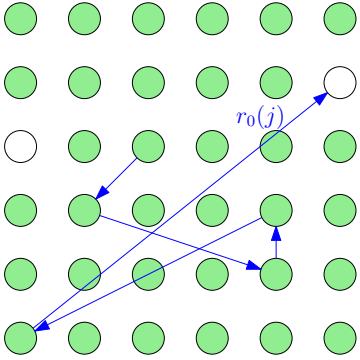
[1] Hardware comparable in per-core performance to "ours"

Mallob Framework: Randomized Scheduling



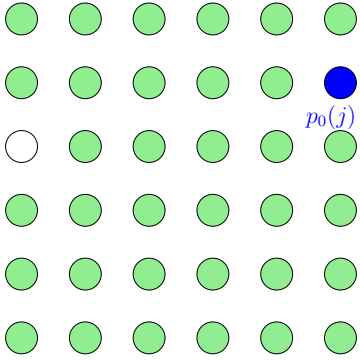
6×6 grid of PEs

Mallob Framework: Randomized Scheduling



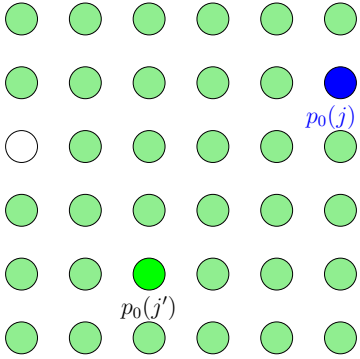
6×6 grid of PEs

Mallob Framework: Randomized Scheduling



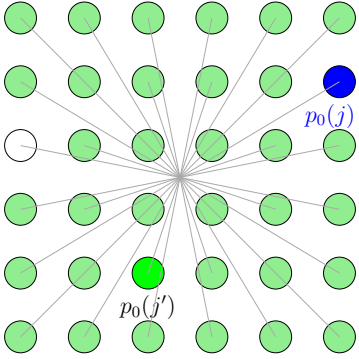
6×6 grid of PEs

Mallob Framework: Randomized Scheduling



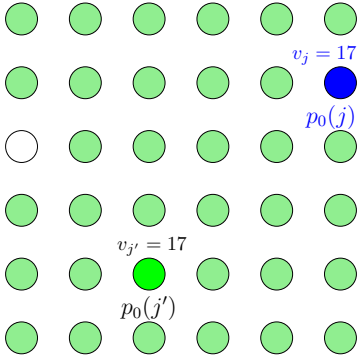
6 $\times$ 6 grid of PEs

Mallob Framework: Randomized Scheduling



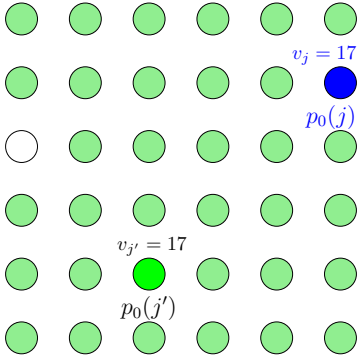
6×6 grid of PEs

Mallob Framework: Randomized Scheduling

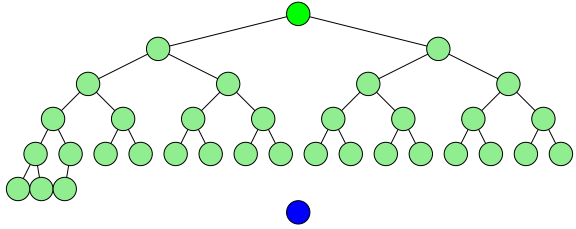


6×6 grid of PEs

Mallob Framework: Randomized Scheduling

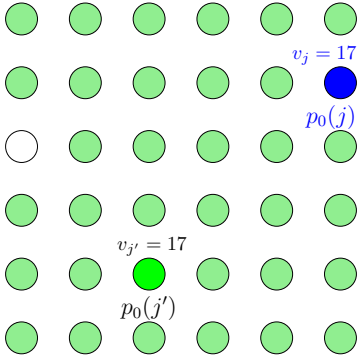


6 × 6 grid of PEs

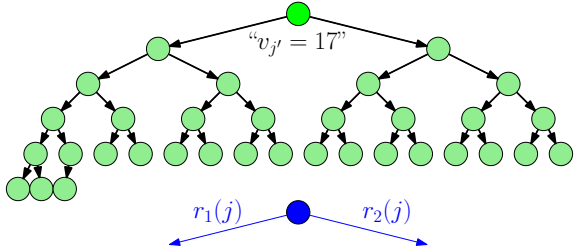


Job tree view

Mallob Framework: Randomized Scheduling

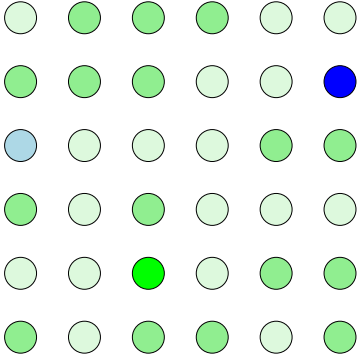


6 × 6 grid of PEs

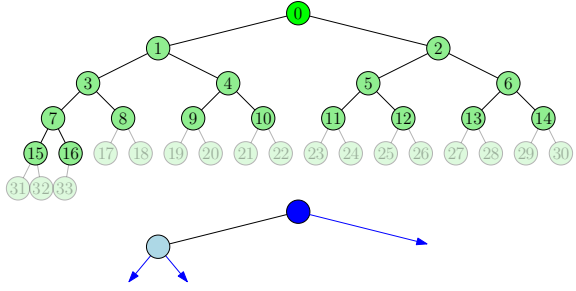


Job tree view

Mallob Framework: Randomized Scheduling

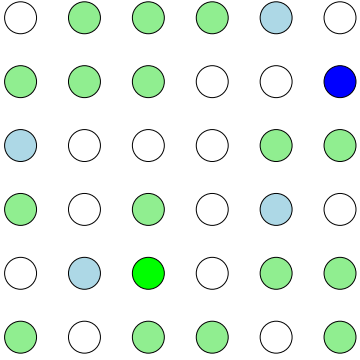


6 × 6 grid of PEs

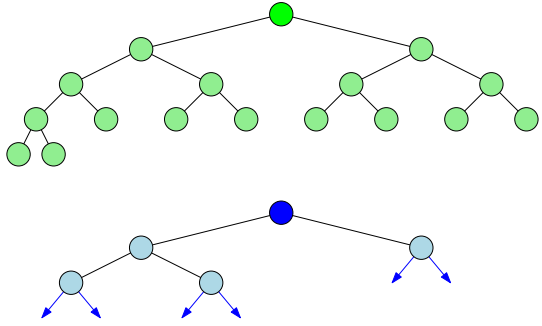


Job tree view

Mallob Framework: Randomized Scheduling

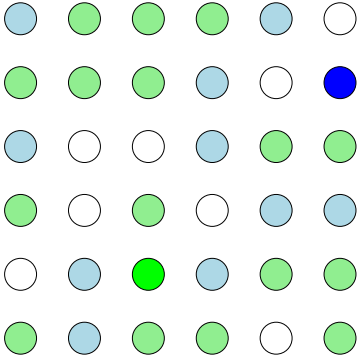


6 × 6 grid of PEs

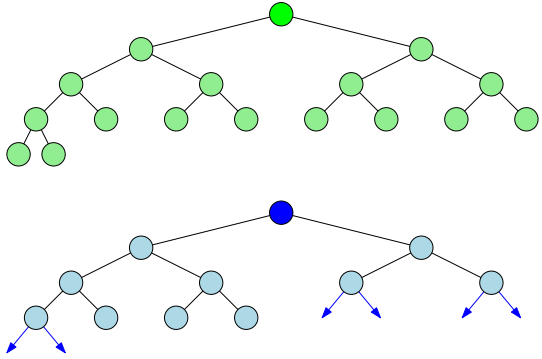


Job tree view

Mallob Framework: Randomized Scheduling

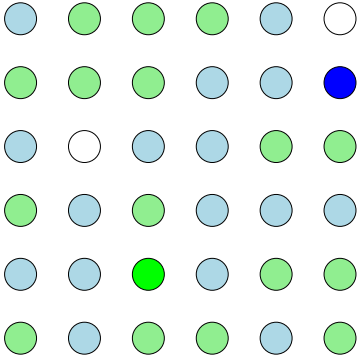


6 × 6 grid of PEs

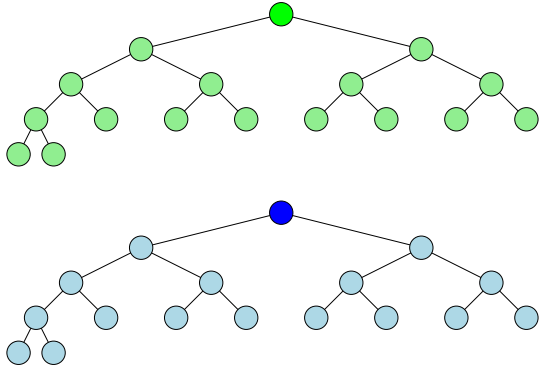


Job tree view

Mallob Framework: Randomized Scheduling



6×6 grid of PEs



Job tree view