ParaSCIP and FiberSCIP libraries to parallelize a customized SCIP solver

Yuji Shinano

Zuse Institute Berlin

Outline

1. Background and goal
2. Runtime behavior of parallel branch-and-bound
3. Ubiquity Generator Framework, ParaSCIP and FiberSCIP
4. How to parallelize a customized SCIP solver
5. How large scale can you expect
6. Concluding remarks

Background and Goal

- SCIP (Solving Constraint Integer Programs)
  - a framework for Constraint Integer Programming
    - can handle large classes of optimization problems (MIP, MINLP)
    - can extent (Scheduling, etc. in example directory)
    - can control solving algorithms using many parameters
    - …
- ParaSCIP and FiberSCIP
  - parallel extensions of SCIP
  - can run on a variety of computing environments
    - PC with multi-cores, PC cluster, supercomputers

We want to parallelize customized SCIP solvers, too!

Background and Goal

- SCIP (Solving Constraint Integer Programs)
  - a framework for Constraint Integer Programming
    - can handle large classes of optimization problems (MIP, MINLP)
    - can extent (Scheduling, etc. in example directory)
    - can control solving algorithms using many parameters
    - …
- ParaSCIP and FiberSCIP
  - parallel extensions of SCIP
  - can run on a variety of computing environments
    - PC with multi-cores, PC cluster, supercomputers

Goal

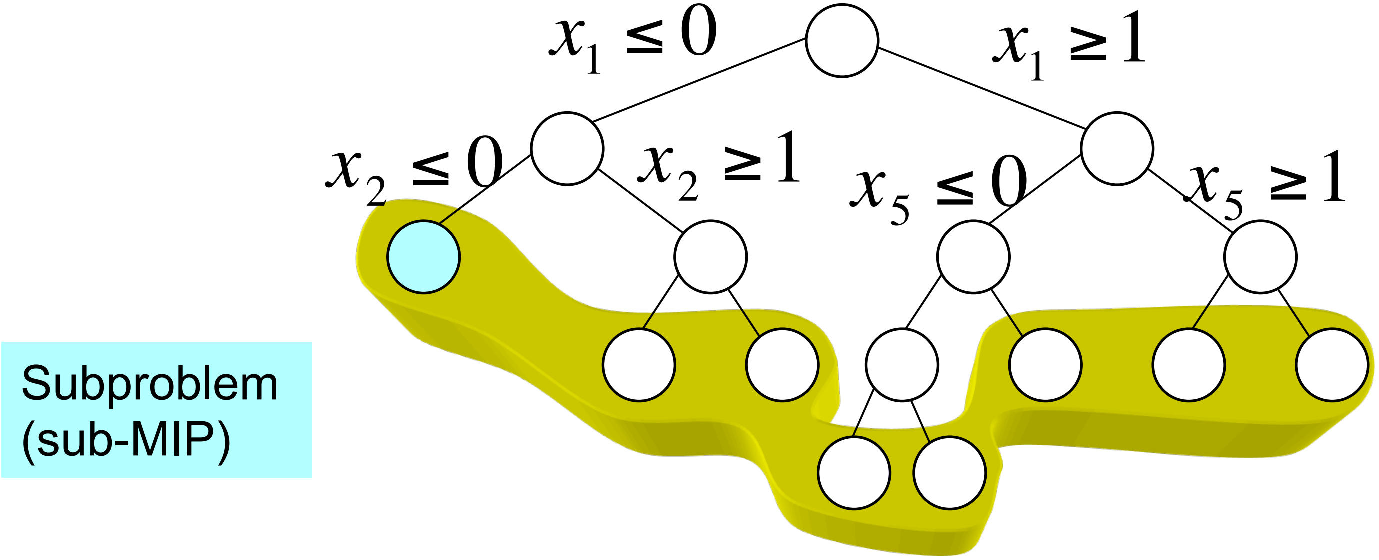
Development of
ParaSCIP and FiberSCIP libraries to parallelize a customized SCIP solver
with the least effort

Runtime behavior of parallel branch-and-bound

Parallelization of MIP solvers

It looks suitable for parallelization

- MIP solvers: LP based Branch-and-cut algorithm



Subproblems (sub-MIPs) can be processed independently

→ Utilize the large number of processors for solving hard problem instances

Anomalies in Parallel Branch-and-bound

- T.-H. Lai, S. Sahni (1984). Anomalies in parallel branch-and-bound algorithms. Comm. ACM 27, 594–602.

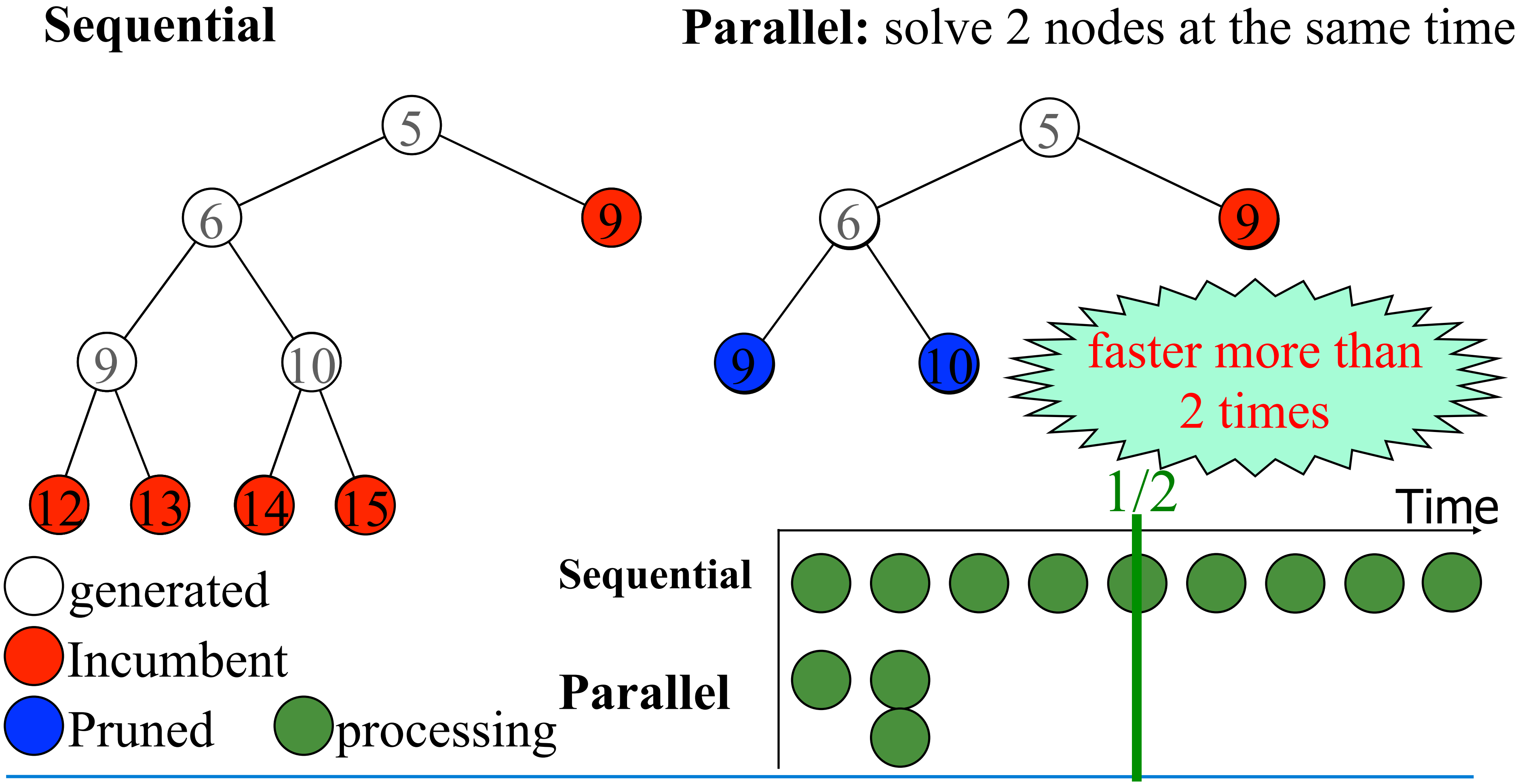
- T.-H. Lai, A. Sprague (1985). Performance of parallel branch-and-bound algorithms. IEEE Trans. Comput. C-34, 962–964.

- G.-J. Li, B.W. Wah (1986). Coping with anomalies in parallel branch-and-bound algorithms. IEEE Trans. Comput. C-35, 568–573.

Anomalies in Parallel Branch-and-bound

Assume that **all nodes are solved by a unit of time** to simplify

Speedup Anomaly



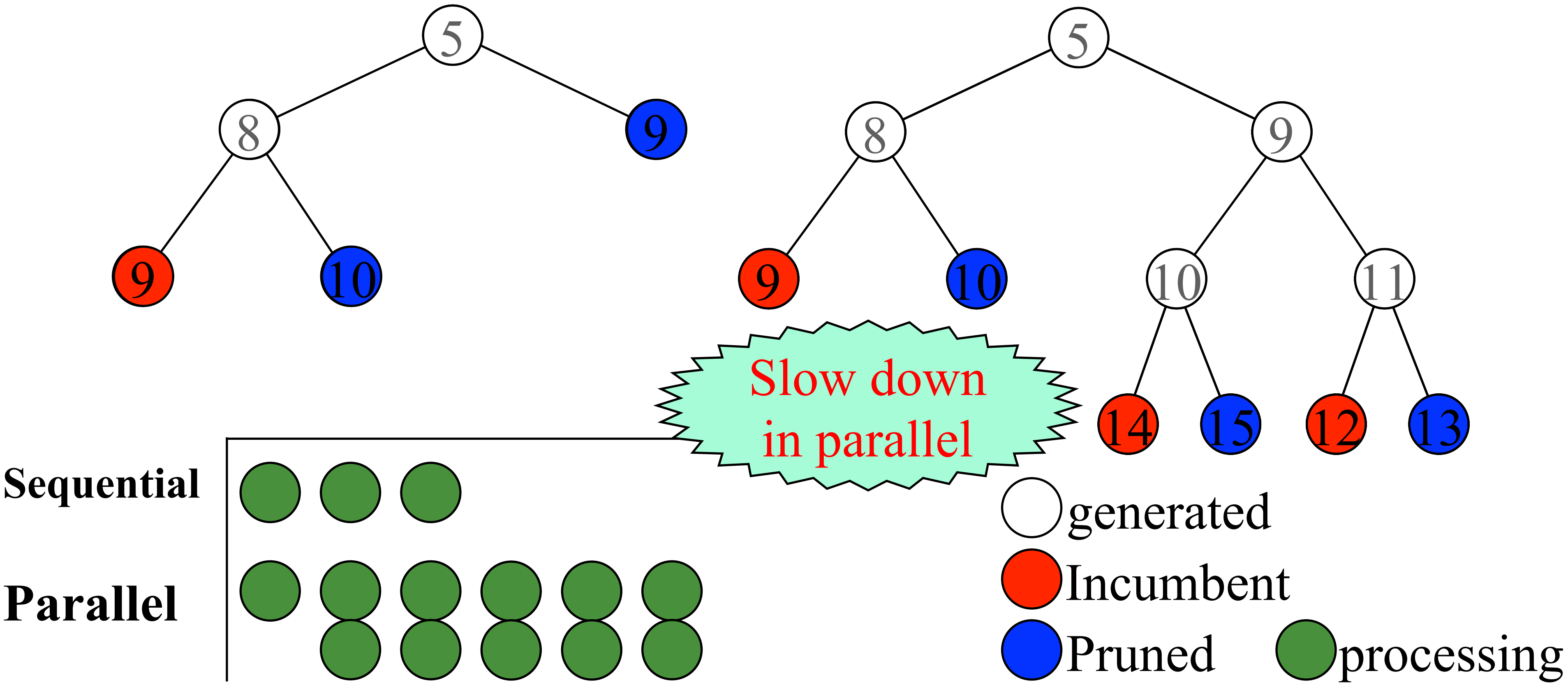
Anomalies in Parallel Branch-and-bound

Assume that **all nodes are solved by a unit of time** to simplify

Detrimental anomaly

Sequential

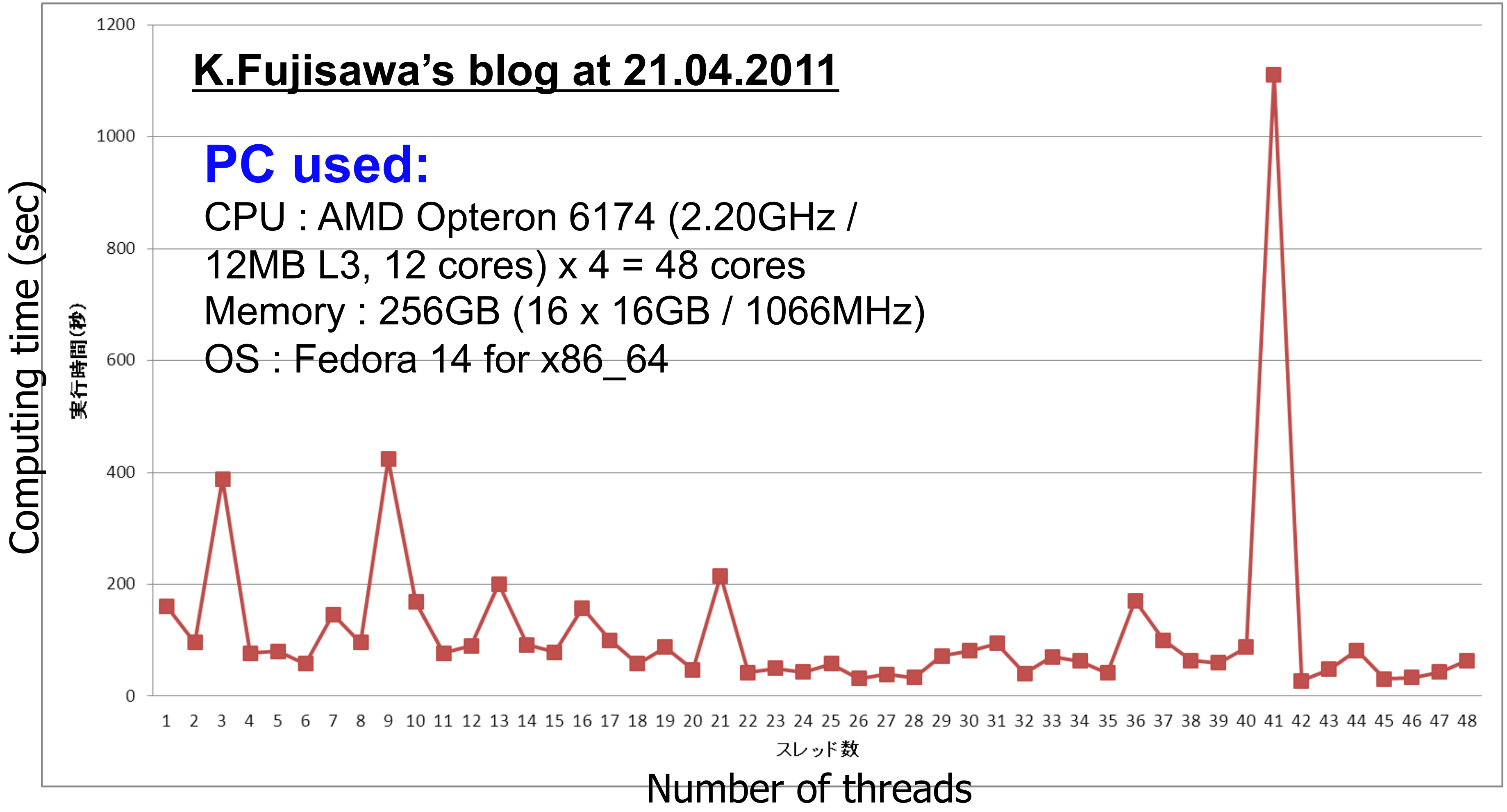
Parallel: solve 2 nodes at the same time



Slow down in parallel

Anomalies in MIP solver

Real observation for solving roll3000 in parallel with Grurobi 4.0.1



K.Fujisawa's blog at 21.04.2011

PC used:

CPU : AMD Opteron 6174 (2.20GHz / 12MB L3, 12 cores) x 4 = 48 cores

Memory : 256GB (16 x 16GB / 1066MHz)

OS : Fedora 14 for x86_64

Performance variability depending on # of threads

Real observation for solving roll3000 (MIPLIB2010 paper)

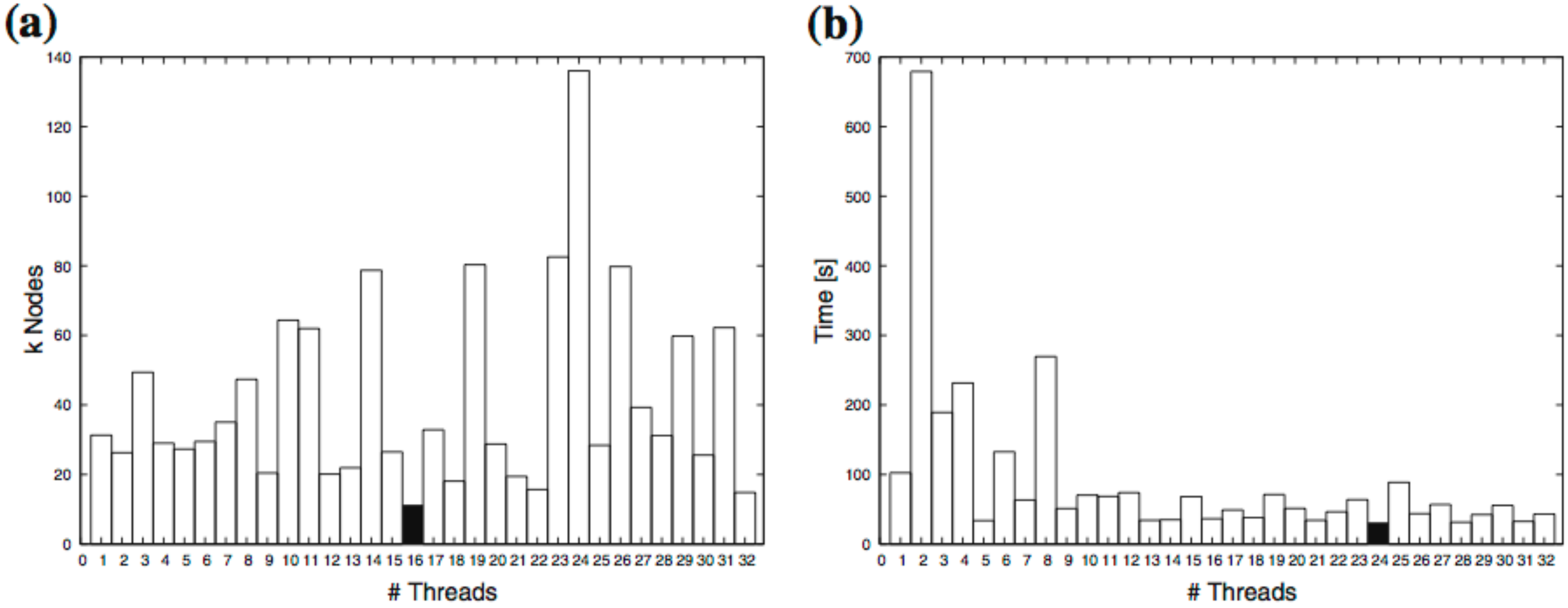
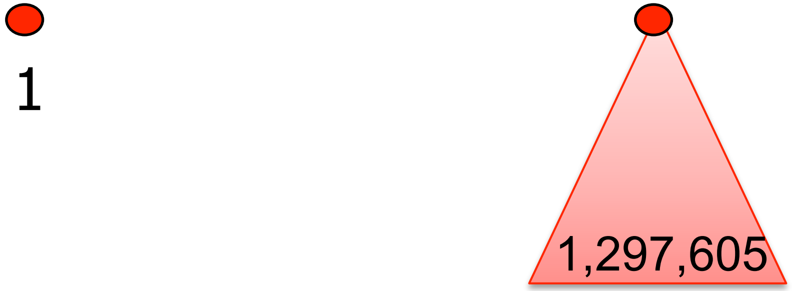


Fig. 4 Example of performance variability depending on the number of threads. Instance roll3000 on a 32 core computer. Filled bar indicates minimum. **a** Total number of nodes explored (CPLEX). **b** Wall clock solution time (GUROBI)

What makes the parallelization difficult?

- How to combine the mathematically sophisticated algorithm implementations in parallelizations?
    - Outside parallelization, that is,
      a state-of-the-art MIP solver is used as a black box solver
    - How to handle performance variability of the state-of-the-art solvers?
- Dynamic load balancing is needed
    - Two types of irregularity can be handled well
        - Irregular # of nodes are generated by a sub-MIP



        - Irregular computing time for a node solving

0.001sec
1.5h

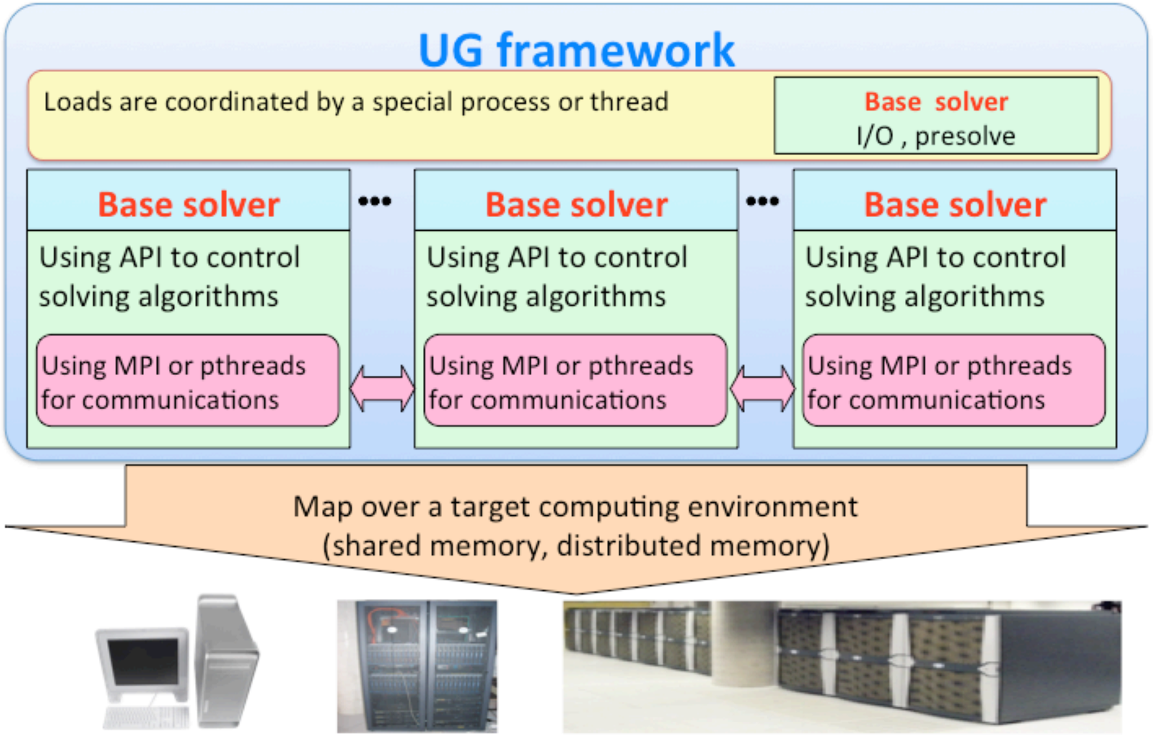
Real observation
for solving ds in parallel
with 4095 solvers

Ubiquity Generator Framework, ParaSCIP and FiberSCIP

UG Ubiquity Generator Framework

UG is a generic framework to parallelize branch-and-bound based solvers (e.g., MIP, MINLP, ExactIP) in a distributed or shared memory computing environment.

- Exploits powerful performance of state-of-the-art "base solvers", such as SCIP, CPLEX, etc.
- Without the need for base solver parallelization


UG framework
Loads are coordinated by a special process or thread
Base solver
I/O , presolve
Base solver
Using API to control solving algorithms
Using MPI or pthreads for communications
•••
Base solver
Using API to control solving algorithms
Using MPI or pthreads for communications
•••
Base solver
Using API to control solving algorithms
Using MPI or pthreads for communications
Map over a target computing environment
(shared memory, distributed memory)

Current Project of UG

Notation of the UG framework

ug[Base solver name, communication library used]

Currently the following parallel solvers have been developed

- ug[SCIP, MPI] : ParaSCIP
  - To investigate a large scale parallelization with SCIP
- ug[SCIP, Pthreads]: FiberSCIP
  - Enables parallelization on single desktop computers
  - To investigate SCIP solver oriented parallelization
  - NOTE: Extension to general CIP is straightforward

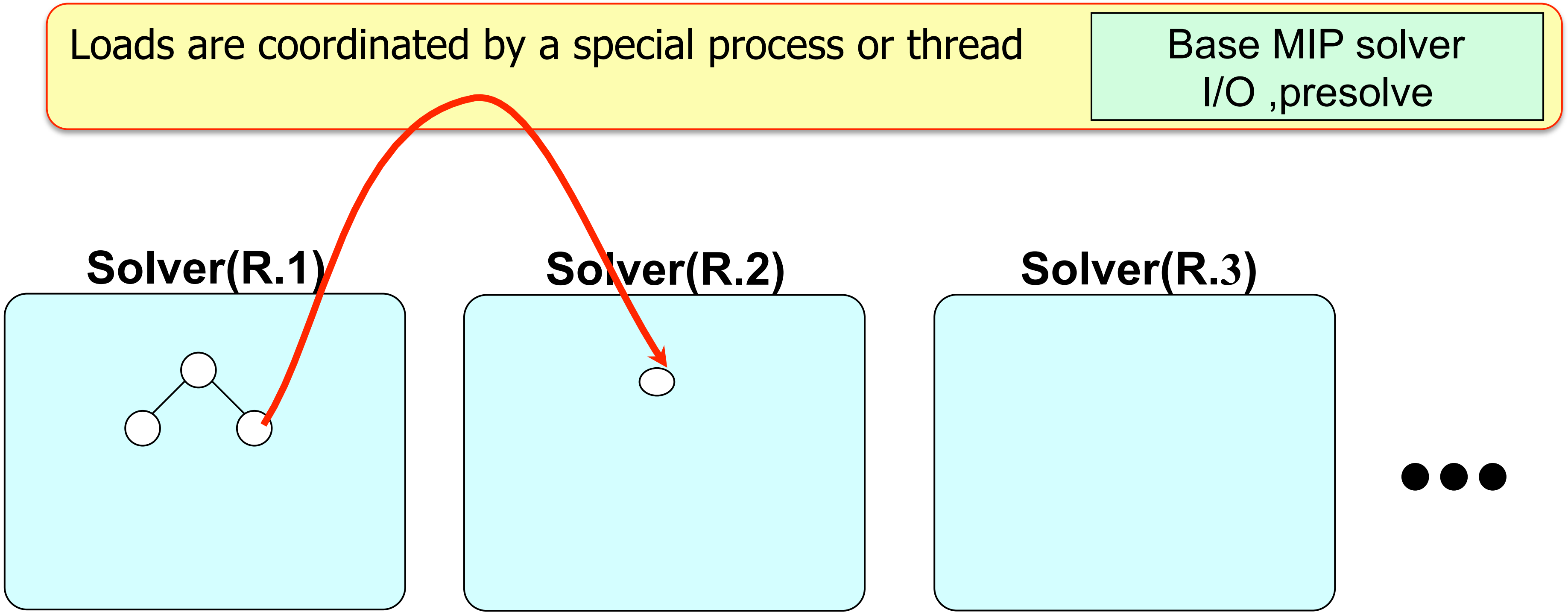
Libraries to parallelize a customized SCIP solver

- ug[CPLEX, MPI]: ParaCPLEX
  - To investigate a large scale parallelization with CPLEX
- ug[CPLEX, Pthreads]: FiberCPLEX
  - Development environment of ParaCPLEX

Main Features of UG

- Ramp-up (process until all solvers become busy) mechanisms
  - Normal ramp-up
  - Racing ramp-up
- Dynamic load balancing mechanism
- Check pointing and restarting mechanism

Normal Ramp-up



Loads are coordinated by a special process or thread

Base MIP solver
I/O ,presolve

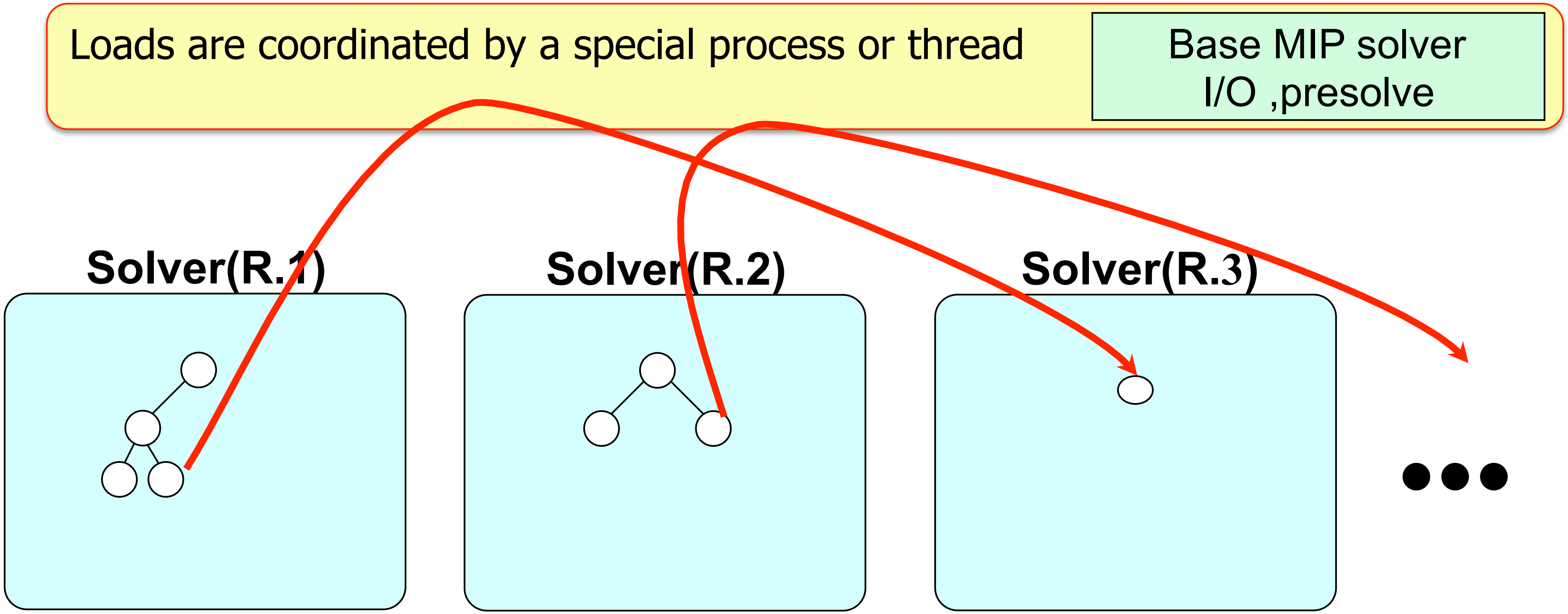
Solver(R.1)

Solver(R.2)

Solver(R.3)

For each branch, one of the branched nodes is transferred until all solvers become busy

Normal Ramp-up



Loads are coordinated by a special process or thread

Base MIP solver
I/O ,presolve

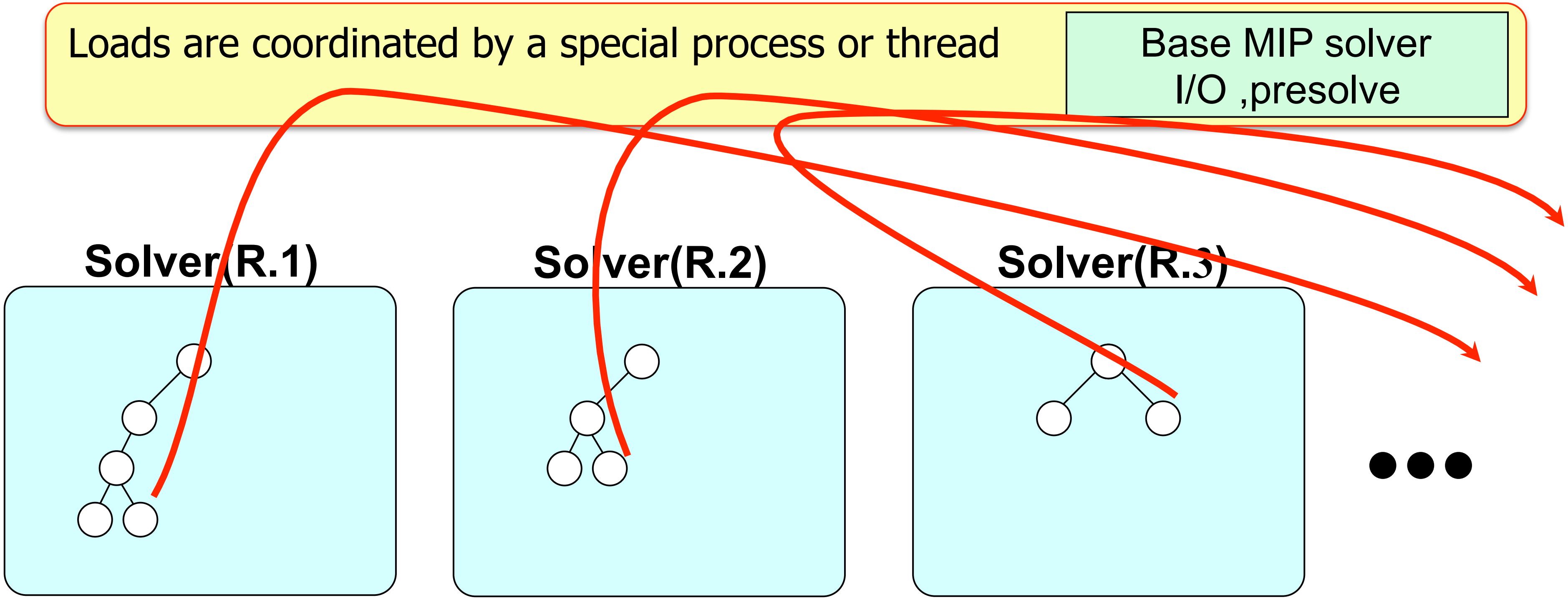
Solver(R.1)

Solver(R.2)

Solver(R.3)

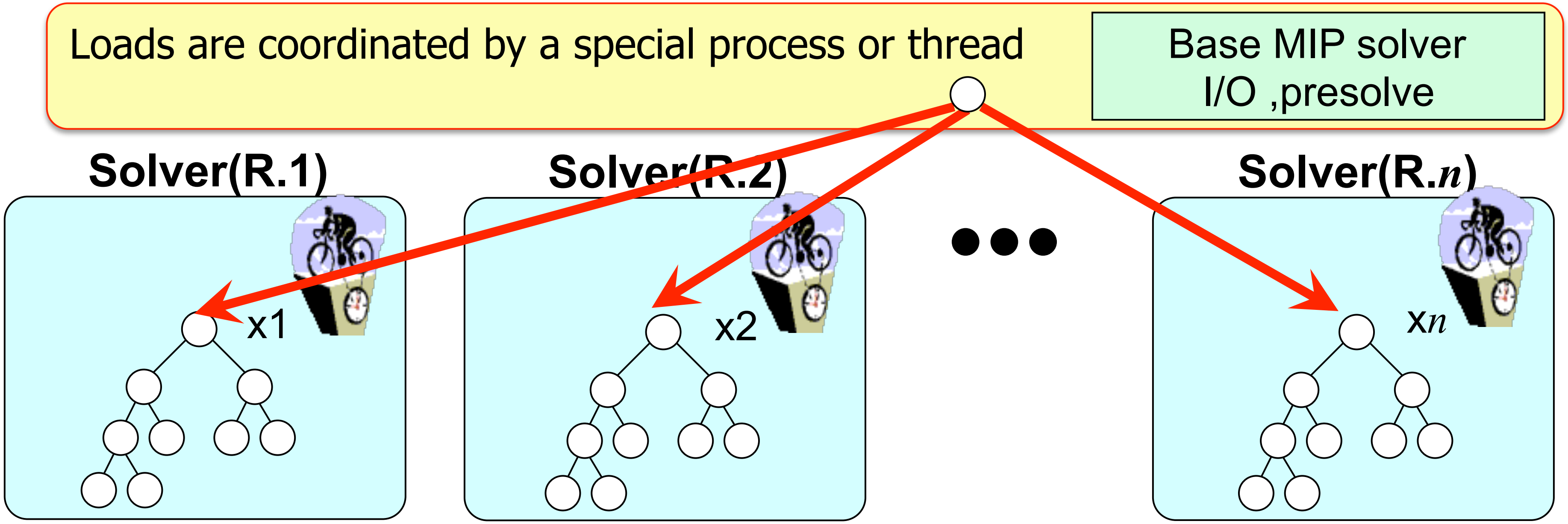
For each branch, one of the branched nodes is transferred until all solvers become busy

Normal Ramp-up



If root node processing takes a long time, many solvers remain idle

Racing Ramp-up

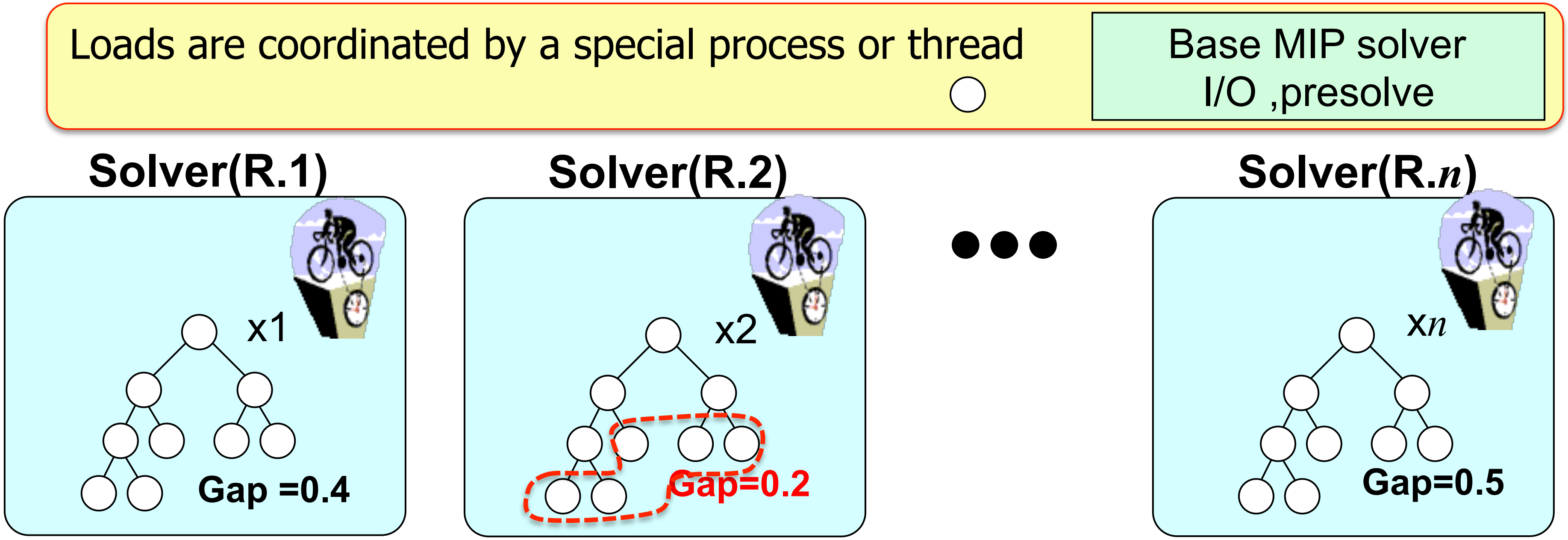


All solvers start immediately and generate different search trees independently based on:

- different parameter settings
- different branching variable selection
- different permutations of columns
- etc.

Upper bound is communicated to all solvers as usual

Racing Ramp-up



Loads are coordinated by a special process or thread

Base MIP solver
I/O ,presolve

Solver(R.1)

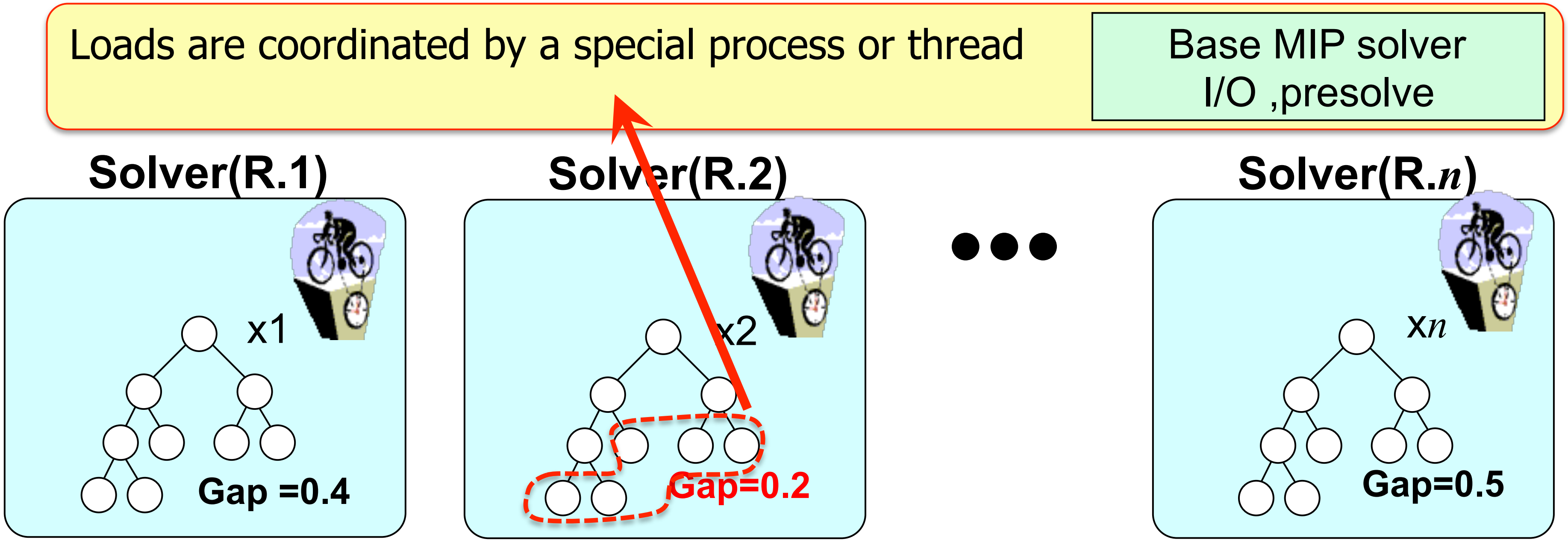
Solver(R.2)

Solver(R.n)

The winner is selected under some criteria considering:

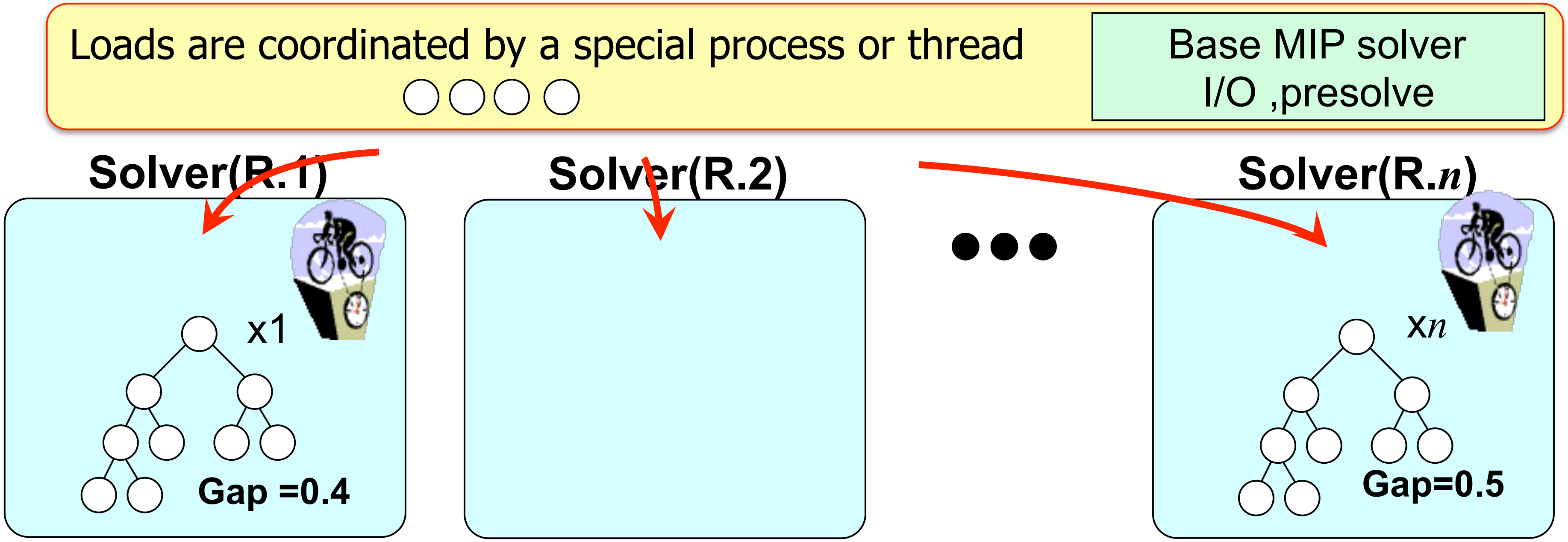
- lower bound quality
- number of nodes remaining
- time limit

Racing Ramp-up



All open nodes of the winner are collected to LoadCoordinator

Racing Ramp-up



Loads are coordinated by a special process or thread

Base MIP solver
I/O ,presolve

Solver(R.1)

Solver(R.2)

Solver(R.n)

x1

Gap =0.4

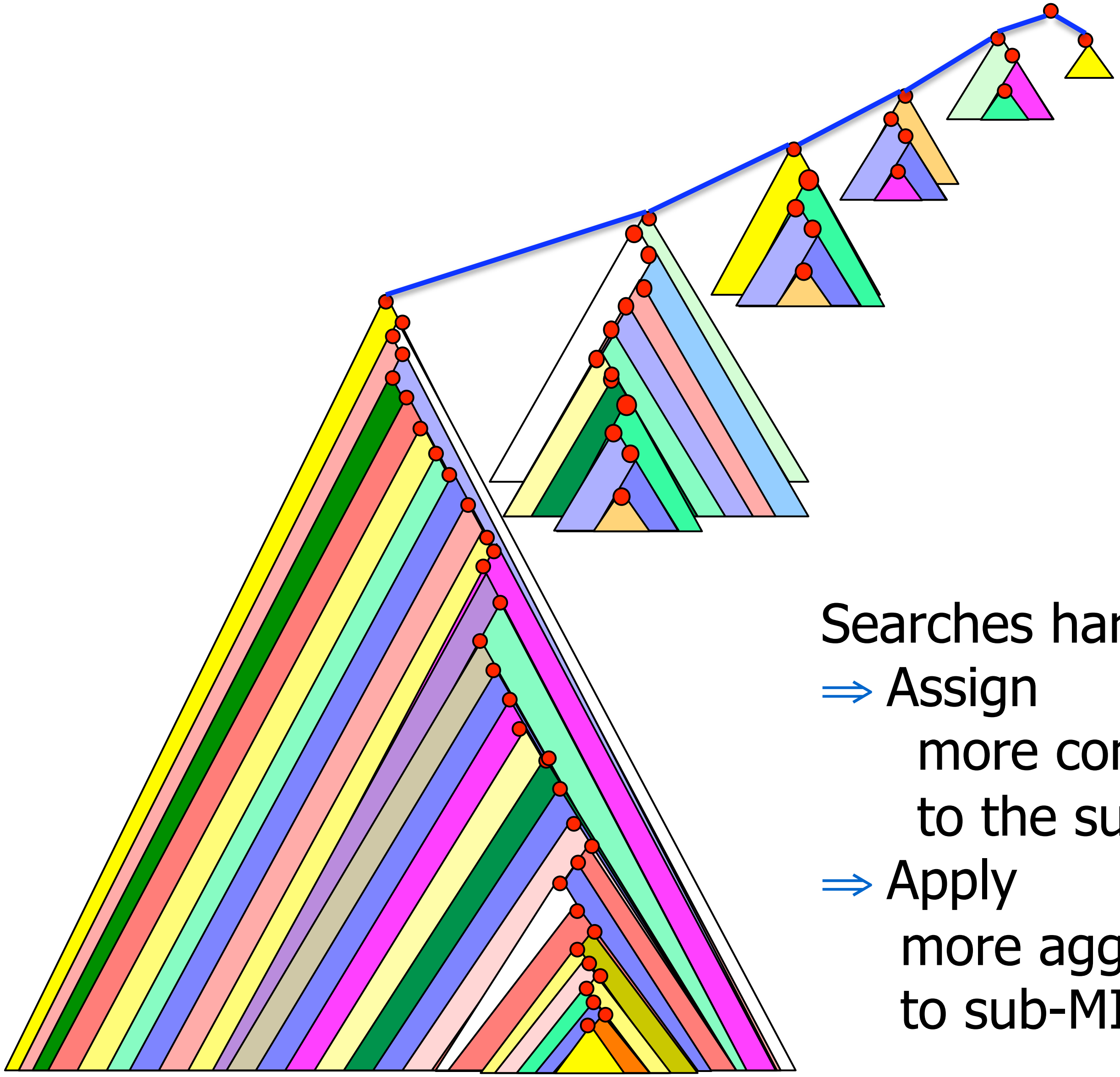
xn

Gap=0.5

Distributes nodes to the other solvers in order of dual bound
Switch to normal ramp-up, if it is necessary

Racing stage can be considered
a learning or tuning process

What does the dynamic load balancing do?



Searches hardest part of sub-trees

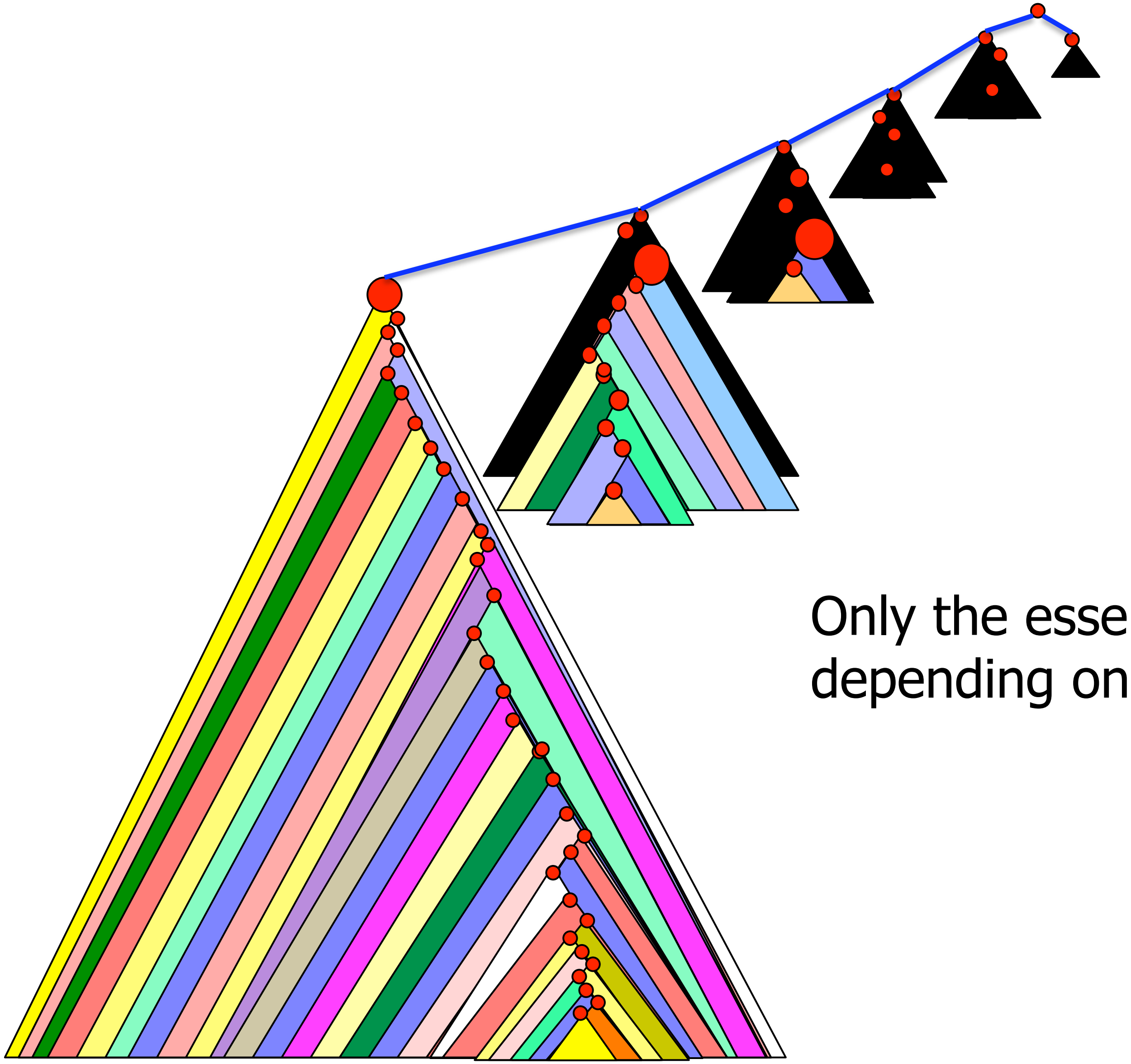
⇒ Assign

more computing resources to the sub-trees

⇒ Apply

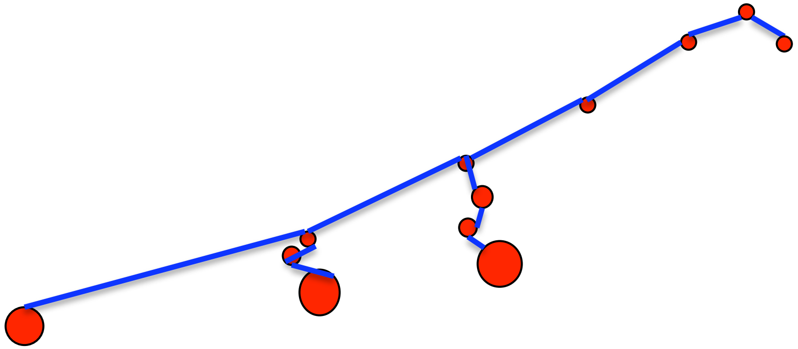
more aggressive presolving to sub-MIPs in the sub-tree.

Check pointing



Only the essential nodes are saved depending on run-time situation

Restarting



Only the essential nodes are saved depending on run-time situation

Huge trees might throw away,
but the saved nodes' dual bound values are calculated more precisely.

Some facts on ParaSCIP

- Computer used so far:
  - Alibabab cluster at ZIB (PC cluster)
    - maximum cores cores for a job: 320
  - HLRN II supercomputer (SGI Altix ICE 8200EX)
    - maximum cores for a job: 4096, tried to run up to 8,000 cores
  - HLRN III supercomputer (Cray XC30 system)
    - maximum cores for a job: 17,088
  - Titan at Oak Ridge National Laboratory (Cray XK7 system)
    - maximum cores for a (normal) job: 179,984, tried up to **35,200** cores
  - ISM supercomputer (Fujitsu PRIMERGY RX200S)
    - maximum cores for a job: 512

Some facts on ParaSCIP

- MIPLIB2003
  - Solved first time: ds, stp3d
    - Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, and T. Koch. ParaSCIP - a parallel extension of SCIP. In C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum, editors, Competence in High Performance Computing 2010, pages 135-148. Springer, February 2012.
  - updated primal bound for current four open instances
    - momentum3: 235248.256846256 (Prev. 236426.335)
    - dano3mip: 676.481481481482(Prev. 687.733333)
    - liu: 1095.9999999989 (Prev. 1102)
    - t1717: 161740 (Prev. 170195)
- MIPLIB2010
  - Computed optimal solution for:
    50-10v, probportfolio, reblock354, rmatr200-p20,
    dg012142, dc1c, germany50-DBM, dolom1

    (see, MIPLIB2010 page)

How to parallelize a customized SCIP solver

Just make a customized SCIP solver

- Make user plugins in SCIP way
  - The following plugins and functions are needed
    - You must have Reader plugin for the customized SCIP solver
    - You must have copy functions for
      - Probdata
      - ConstraintHandler

- But, you do NOT need to take into account parallelization
- RESTRICTION:
  - branch on variables

- Ensure the customized SCIP solver stable (with single thread)

- Tune the solver specific parameters

Add glue code

LoadCoordinator

Loads are coordinated by a special process or thread

Base MIP solver

I/O , **presolve**

Solver(R.1)

Base
MIP solver
Solve
sub-MIP

●●●

FiberSCIP and ParaSCIP as libraries

- Use “ScipUserPlugin class”

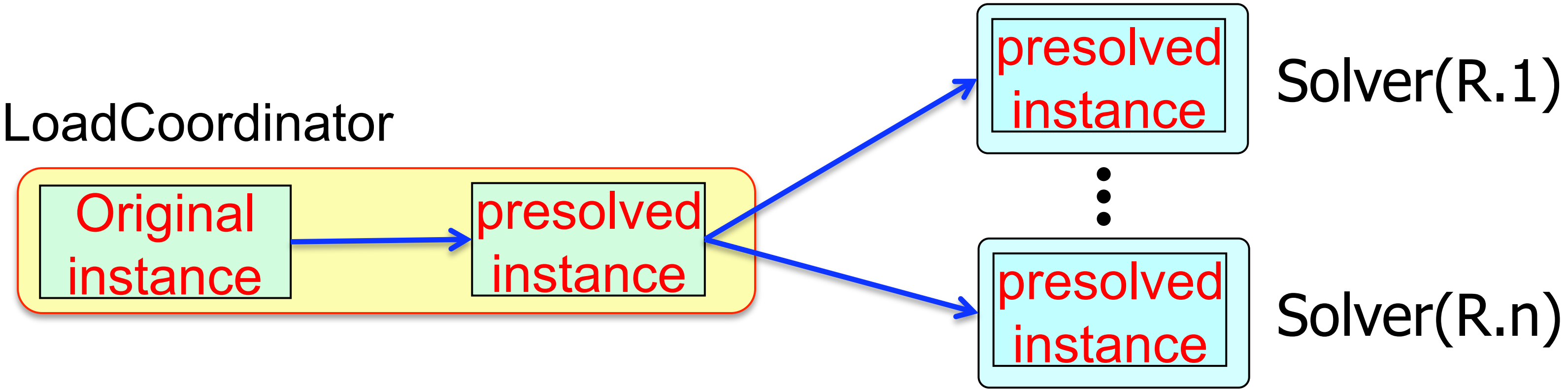
Copy from cmain.c

```
class StpUserPlugins: public ScipUserPlugins {
   void operator()(SCIP *scip)
   {
      /* include stp pricer */
      SCIP_CALL_ABORT( SCIPincludePricerStp(scip) );
      /* include steiner tree reader */
      SCIP_CALL_ABORT( SCIPincludeReaderStp(scip) );
      /* include steiner tree constraint handler */
      SCIP_CALL_ABORT( SCIPincludeConshdlrStp(scip) );
      /* include Takahashi Matsuyama heuristic */
      SCIP_CALL_ABORT( SCIPincludeHeurTM(scip) );
      /* include local heuristics */
      SCIP_CALL_ABORT( SCIPincludeHeurLocal(scip) );   }
};
```

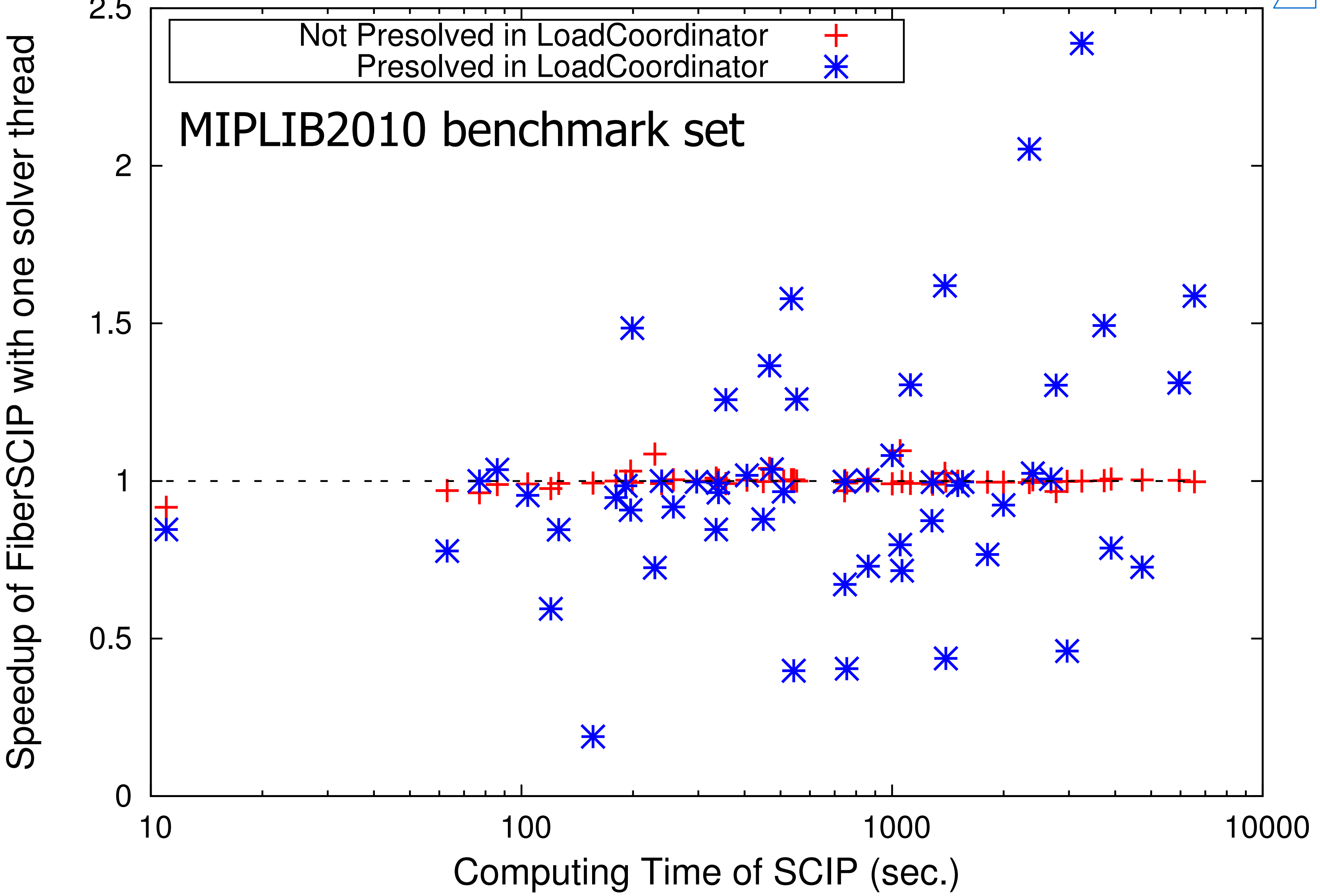
Build shared memory version

All codes including the extended ScipUserPlugin class linked to the FiberSCIP library (libugscip-0.7.3.linux.x86_64.gnu.dbg.pth.a).

- Test a parallel version of the customized SCIP (shared memory)
  - Can use GDB for debugging
  - Can use deterministic mode
    - Repeated runs with the same parameter settings and with the same number of solvers are expected to generate the same search tree
  - NOTE:
    - SCIP (and SoPlex) library compiled with "PARASCIP=true" option.
    - The parallel version solver solve the different model in General



SCIP and FiberSCIP (single solver) comparison



Can control parallel tree search well

LoadCoordinaor(R.0)

Loads are coordinated by a special process or thread

● ● ● ●

Base MIP solver

I/O , **presolve**

Solver(R.1)

MIP solver

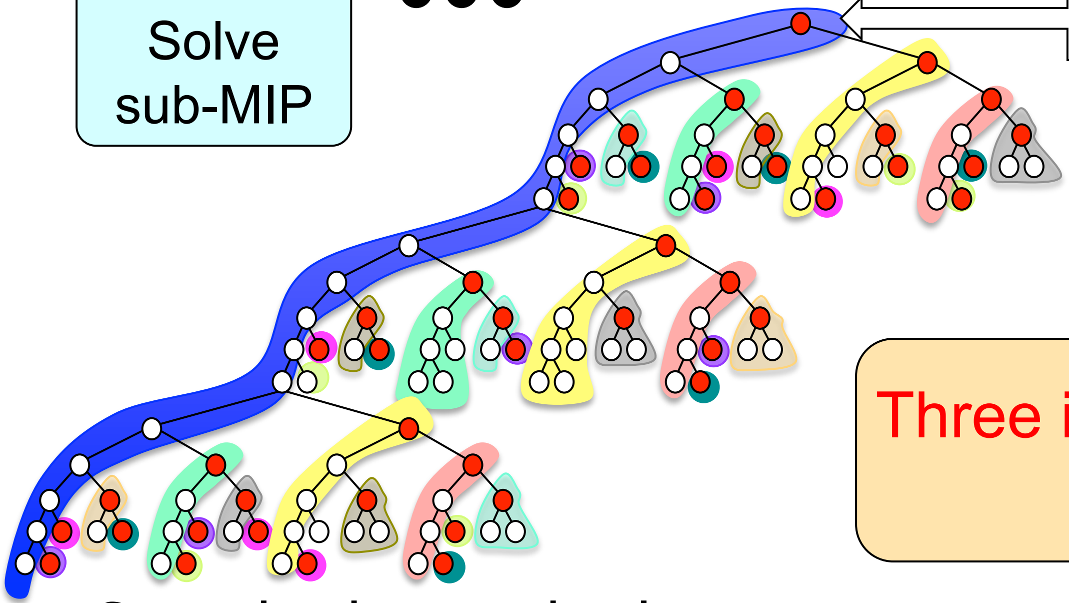
Solve sub-MIP

Solver

presolving

● ● ●

root subtree solving



Three independent parameter sets can be specified

Same background color

⇒ Solved by the same MIP solver

● : transferred node

Build distributed memory version

All codes including the extended ScipUserPlugin class linked to the ParaSCIP library (libugscip-0.7.3.linux.x86_64.gnu.dbg.mpi.a).

- Should work immediately
  - You can use deterministic mode, if you need to debug
- Unfortunately, in current released version,
  - Sheared memory version probably work well (libugscip-0.7.3.linux.x86_64.gnu.dbg.pth.a)
  - Distributed memory version does not work well (libugscip-0.7.3.linux.x86_64.gnu.dbg.mpi.a)
- You can expect stable version
  with an example for Steiner Tree Problem
  in the next release

An example: Steiner Tree Problems

DIMACS Implementation Challenge

"The DIMACS Implementation Challenges address questions of determining realistic algorithm performance where worst case analysis is overly pessimistic and probabilistic models are too unrealistic: experimentation can provide guides to realistic algorithm performance where analysis fails.

Experimentation also brings algorithmic questions closer to the original problems that motivated theoretical work. It also tests many assumptions about implementation methods and data structures. It provides an opportunity to develop and test problem instances, instance generators, and other methods of testing and comparing performance of algorithms.

And it is a step in technology transfer by providing leading edge implementations of algorithms for others to adapt."

Previous Challenges, 11th challenge

1. Network Flows and Matching (1990)
2. Maximum Clique, Graph Coloring, and Satisfiability (1992)
3. Effective Parallel Algorithms for Combinatorial Problems (1993)
4. Fragment Assembly and Genome Rearrangements (1994)
5. Priority Queues, Dictionaries, and Multi-Dimensional Point Sets (1995)
6. Near Neighbor Searches (1998)
7. Semidefinite and Related Optimization Problems (2000)
8. Traveling Salesman Problem (2001)
9. The Shortest Path Problem (2006)
10. Graph Partitioning and Clustering (2012)
11. Steiner Tree Problems (2013 - 2014)

    http://dimacs11.cs.princeton.edu/

SCIP-Jack – A massively parallel STP solver

- Sequential and parallel solvers have been developed concurrently
- Special settings for SCIP-Jack in parallelization
  - Local cuts also transferred between solvers
    - cf. only bound changes in MIP case as default
  - Normal ramp-up
- PUC test set (as of 11th September):
  - solved to optimality (3/32 unsolved instances)
  - improved primal bound (13/32 unsolved instances)

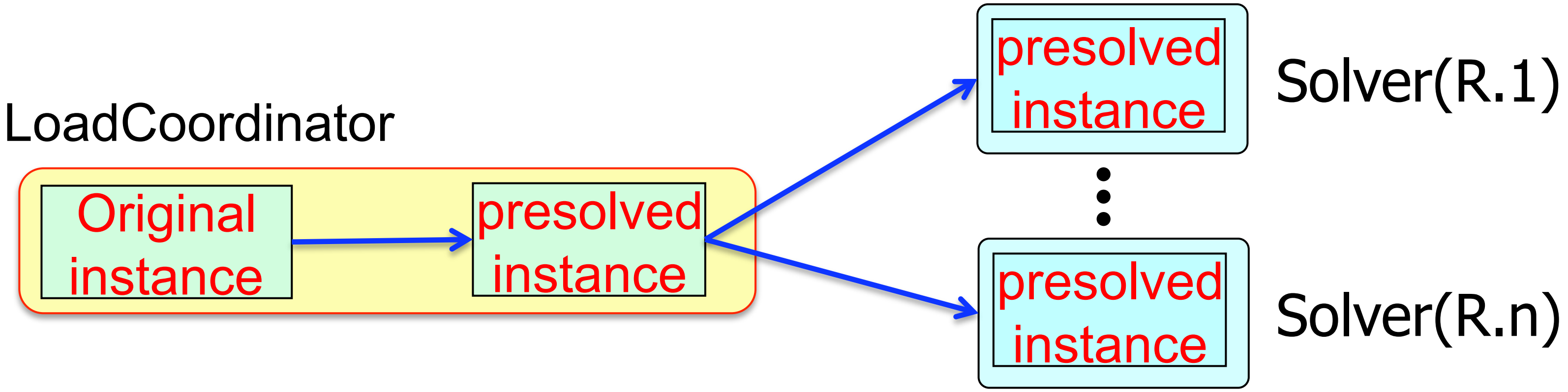
Reference:
Gerald Gamrath, Thorsten Koch, Daniel Rehfeldt and Yuji Shinano,
SCIP-Jack – A massively parallel STP solver, ZIB Report-14-35.

This article was submitted to the 11th DIMACS Implementation Challenge on Steiner Tree Problems

How large scale can you expect

Note on initialization phase

All solvers read problem instance data at initialization



LoadCoordinator

Original instance

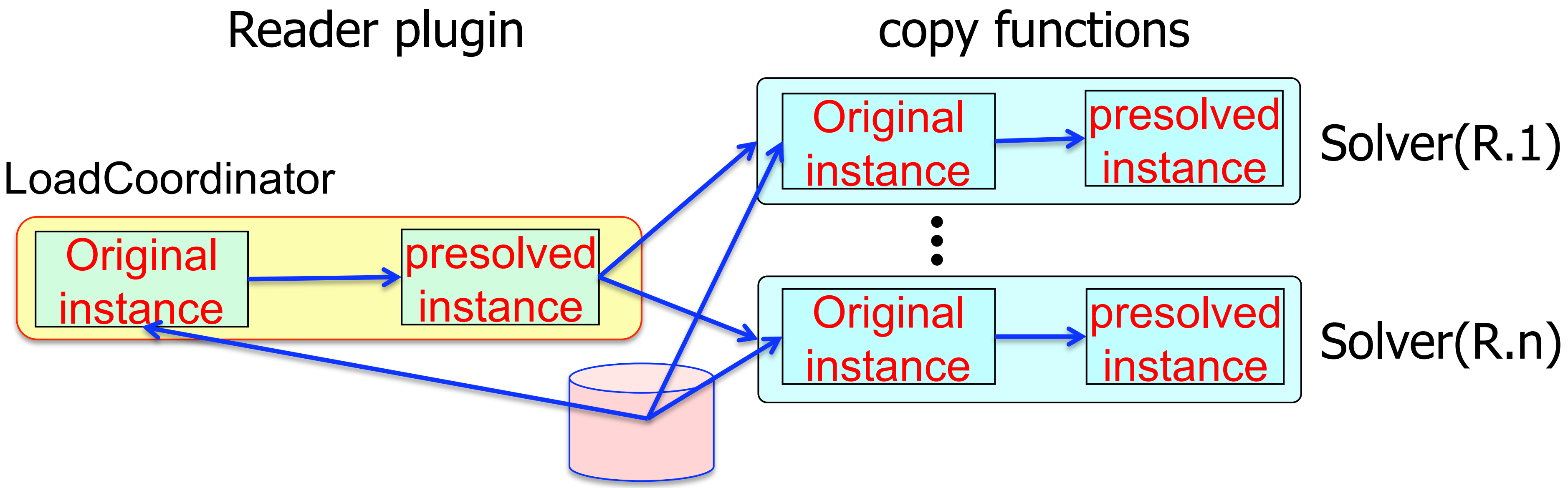
presolved instance

presolved instance

Solver(R.1)

presolved instance

Solver(R.n)



Reader plugin

copy functions

LoadCoordinator

Original instance

presolved instance

Original instance

presolved instance

Solver(R.1)

Original instance

presolved instance

Solver(R.n)

Is this initialization a big issue?

Initialization overhead (solving t1717)

- SGI ICE X (connection: InfiniBand 4x FDR)
  - 36 nodes, 864 processes: 75.4975(sec.) to send the first node
- Titan (connection: Cray’s high-performance Gemini network)
  - 1,000 nodes, 16,000 processes: 190.995(sec) to send the first node
  - 2,000 nodes, 32,000 processes: waiting in the queue

The initialization looks like not too big an issue.
Instance data read occurs only once!

Scalability test : a1c1s1

Normalized factor: alibaba 0.85, HLRN II 1.0, Titan 0.66

computer	#Solver	norm.	mean	1	2	3	4	5
alibaba	239	47174.3	55499.2	60026.3	52654.5	54929.6	53512.2	56373.4
HLRN II	239	46468.9	46468.9	-	-	-	-	-
HLRN II	4095	3519.6	3519.6	-	-	-	-	-

(sec.)

17 times scale up, 13 times speedups

computer	#Solver	# nodes	1	2	3	4	5
alibaba	239	241504596	268003383	232803876	236883629	231497054	238335038
HLRN II	239	282919399	-	-	-	-	-
HLRN II	4095	241534277	-	-	-	-	-

Scalability test : timtab2

Normalized factor: alibaba 0.85, HLRN II 1.0, Titan 0.66

computer	#Solver	norm.	mean	1	2	3	4	5
alibaba	239	62603.5	73651.1	72203.4	75113.7	71721.6	72754.8	76462.2
HLRN II	239	53797.5	53797.5	-	-	-	-	-
HLRN II	4095	8187.4	8187.4	-	-	-	-	-
Titan	9999	1916.3	3903.5	-	-	-	-	-

17 times scale up, 6 times speedups

(sec.)

2 times scale up, 4 times speedups

computer	#Solver	# nodes	1	2	3	4	5
alibaba	239	3823061040	3631940876	3891858496	3743162996	3834979497	4013363333
HLRN II	239	3881705732	-	-	-	-	-
HLRN II	4095	4863279000	-	-	-	-	-
Titan	9999	4122861597	-	-	-	-	-

Concluding remarks

- Its time to use FiberSCIP library
  - Next release should include a stable version
  - You can expect over 16,000 solvers parallelization
    - in case that only bound changes are transfer
    - in case that local cuts transfer results will be known soon
  - Internally, we have developed parallelized customized solvers:
    - Scheduler
    - Steiner tree problem
    - Minimum coloring problem
- For the development,
  you do not have to care about parallelization at all
  - we are pleased to help you to make the parallel version!

Reference

You can download the following paper from UG web page

- Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, Thorsten Koch, ParaSCIP -- a parallel extension of SCIP.
- Yuji Shinano, Stefan Heinz, Stefan Vigerske, Michael Winkler, FiberSCIP -- A shared memory parallelization of SCIP.

Thank you very much for your attention!