Ubiquity Generator Framework:
Current Status via a SCIP Application Example

Yuji Shinano
Zuse Institute Berlin

Outline

- Notes about the UG design and its dynamic load balancing
- Main computational results of ParaSCIP and ParaXpress
  - Solving previously unsolvable instances of MIP
- Some other projects with ParaSCIP
  - A new feature of SCIP is going to be parallelized
  - An application of ug[SCIP,*] libraries
- How UG applications were developed
- Future plan
- Concluding remarks

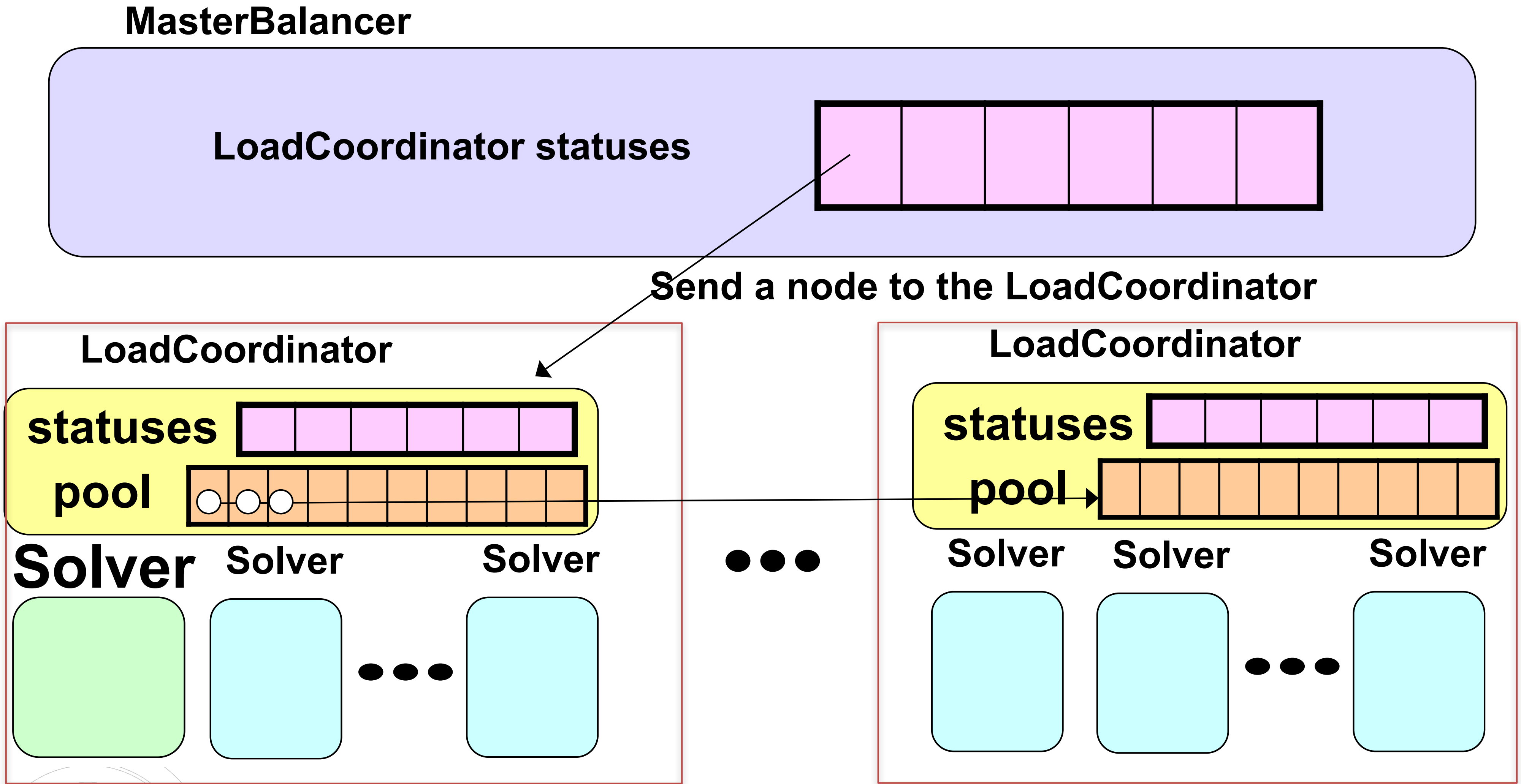
Outline

- Notes about the UG design and its dynamic load balancing
- Main computational results of ParaSCIP and ParaXpress
  - Solving previously unsolvable instances of MIP
- Some other projects with ParaSCIP
  - A new feature of SCIP is going to be parallelized
  - An application of ug[SCIP,*] libraries
- How UG applications were developed
- Future plan
- Concluding remarks

Initial design of ParaSCIP

Two Layered Load Balancing

Goal: run with 10,000 cores



Solving hc9p

Table 1: Statistics for solving hc9p on supercomputers

Run	Computer	Cores	Time (sec.)	Idle (%)	Trans.	Primal bound (Upper bound)	Dual bound (Lower bound)	Gap (%)	Nodes	Open nodes
1	ISM	72	604,796 (317)	< 0.3	738	30,242.0000	29,879.3721	1.21	0	0
						30,242.0000	30,058.9366	0.61	110,012,624	1,257,112
2	ISM	2,304	604,794	< 1.5	979,695	30,242.0000	30,058.7930	0.61	0	15
						30,242.0000	30,102.7556	0.46	3,758,532,600	723,167
3	HLRN III	24,576	86,336	< 1.7	8,811,512	30,242.0000	30,102.6645	0.46	0	35
						30,242.0000	30,116.3592	0.42	2,402,406,311	575,678
4	HLRN III	12,288	43,199	< 1.5	1,709,027	30,242.0000	30,115.3331	0.42	0	3,709
						30,242.0000	30,120.4801	0.40	664,909,985	602,323
5	HLRN III	12,288	118,259	1.5	9,158,920	30,242.0000	30,120.4801	0.40	0	285
						30,242.0000	30,242.0000	0.00	1,677,724,126	0

Supercomputers used:

- ISM: HPE SGI 8600 with 384 compute nodes, each node has two Intel Xeon Gold 6154 3.0GHz CPUs(18 cores$\times$2) sharing 384GB of memory, , and an Infiniband (Enhanced Hypercube) interconnect
- HLRN III: Cray XC40 with 1872 compute nodes, each node with two 12-core Intel Xeon Ivy- Bridge/Haswell CPUs sharing 64 GiB of RAM, and with an Aries interconnect

How open nodes and active solvers evolved (hc9p)

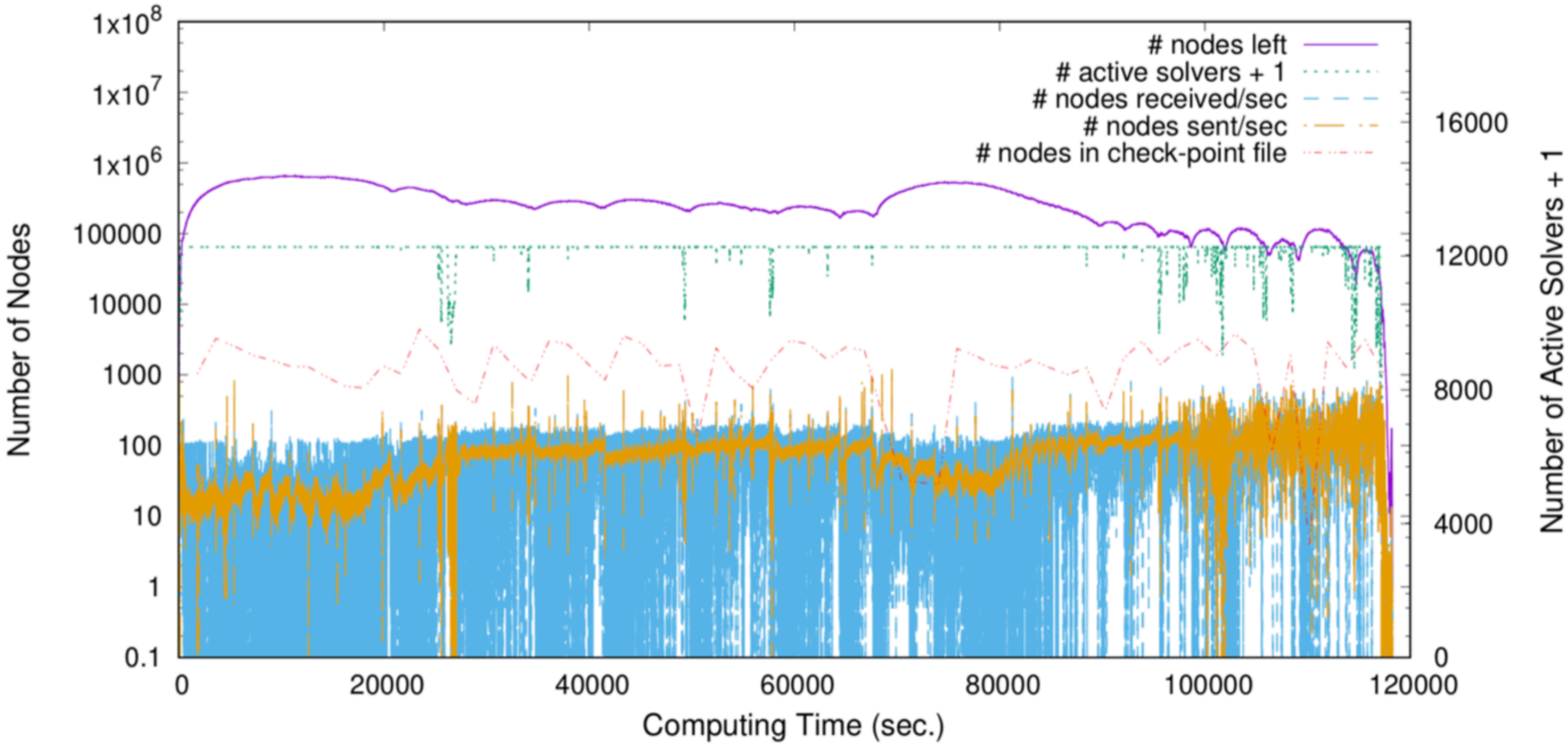


Figure 1: Evolution of computation for solving `hc9p` by using 12,288 cores (Run 5)

Solving hc11p

Table 2: Statistics for solving hc11p on supercomputers

Run	Computer	Cores	Time (sec.)	Idle (%)	Trans.	Primal bound (Upper bound)	Dual bound (Lower bound)	Gap (%)	Nodes	Open nodes
1.1	ISM	72	604,799 (2,558)	< 0.3	71	119,492.0000	117,388.8528	1.79	0	0
						119,297.0000	117,496.5470	1.53	4,314,198	1,109,629
1.2	HLRN III	12,288	43,149 (7,164)	< 0.5	31,304	119,297.0000	117,388.7971	1.63	0	0
						119,297.0000	117,426.2226	1.59	28,491,470	5,433,482
2	HLRN III	43,000	86,354	< 4.9	86,152	119,297.0000	117,426.2226	1.59	0	103
						119,297.0000	117,468.8459	1.56	267,513,609	40,499,188

Supercomputers used:

- ISM: HPE SGI 8600 with 384 compute nodes, each node has two Intel Xeon Gold 6154 3.0GHz CPUs(18 cores$\times$2) sharing 384GB of memory, , and an Infiniband (Enhanced Hypercube) interconnect
- HLRN III: Cray XC40 with 1872 compute nodes, each node with two 12-core Intel Xeon Ivy- Bridge/Haswell CPUs sharing 64 GiB of RAM, and with an Aries interconnect

How open nodes and active solvers evolved (hc11p)

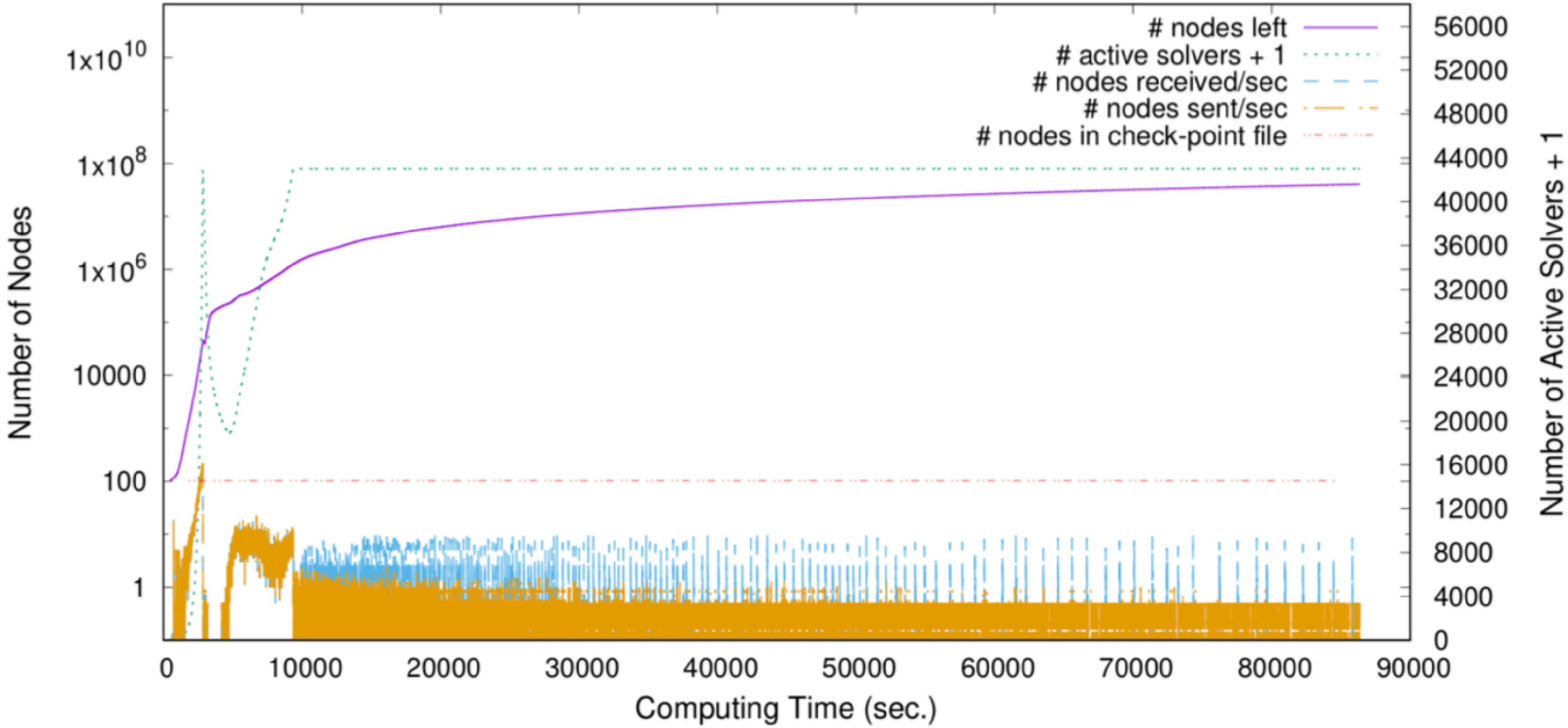
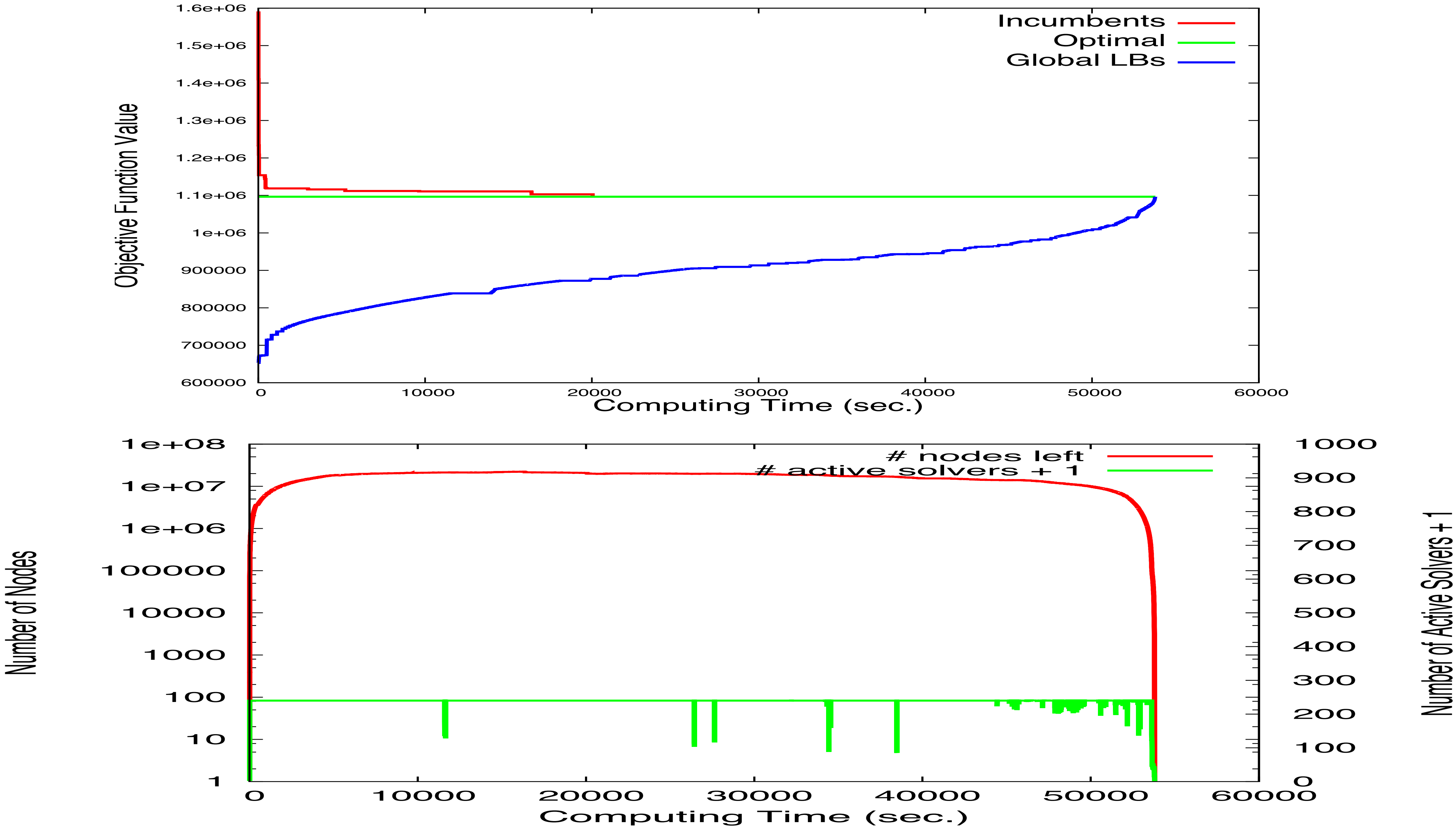


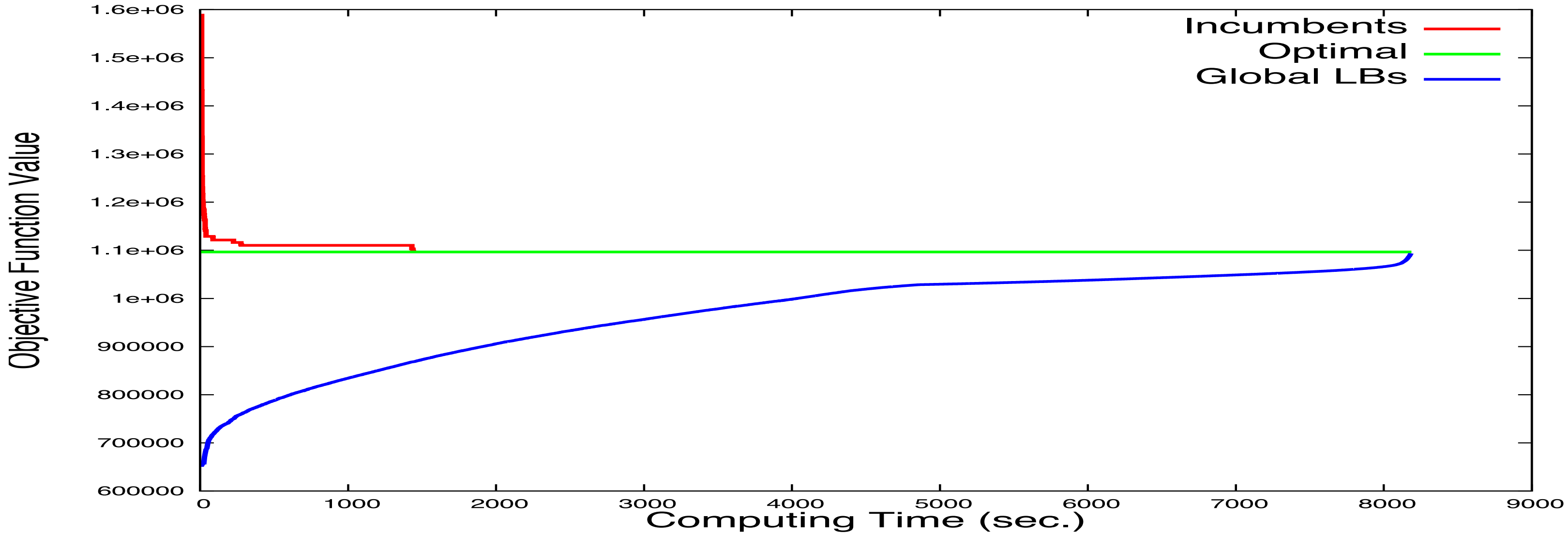
Figure 2: Evolution of computation for solving `hc11p` by using 43,000 cores (Run 2)

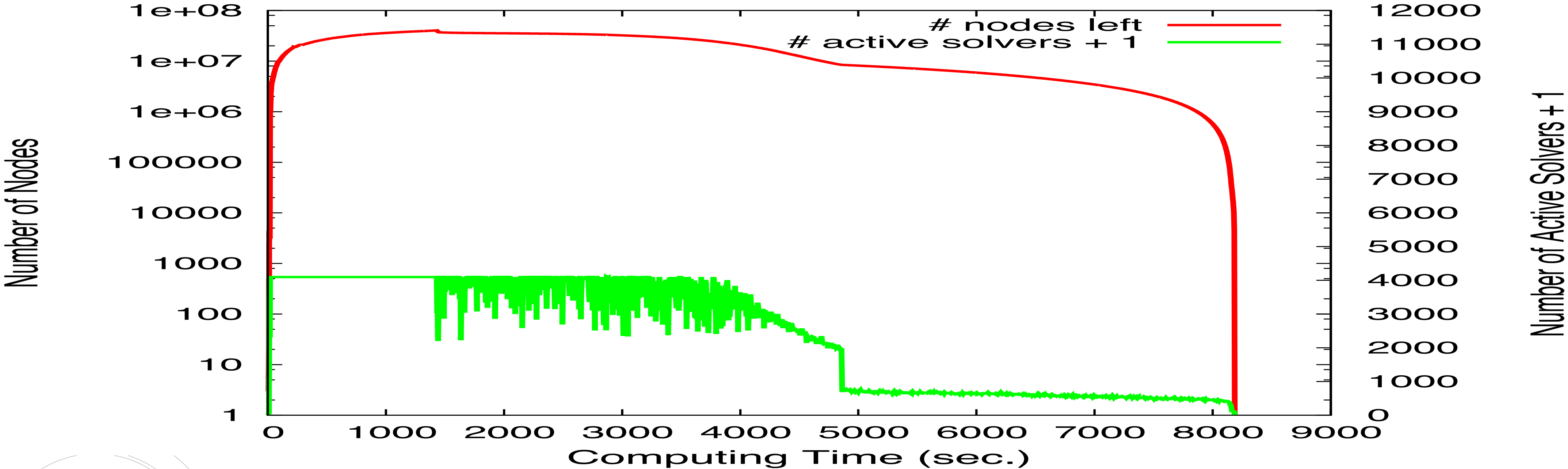
Runtime behavior: timtab2 – 240 cores



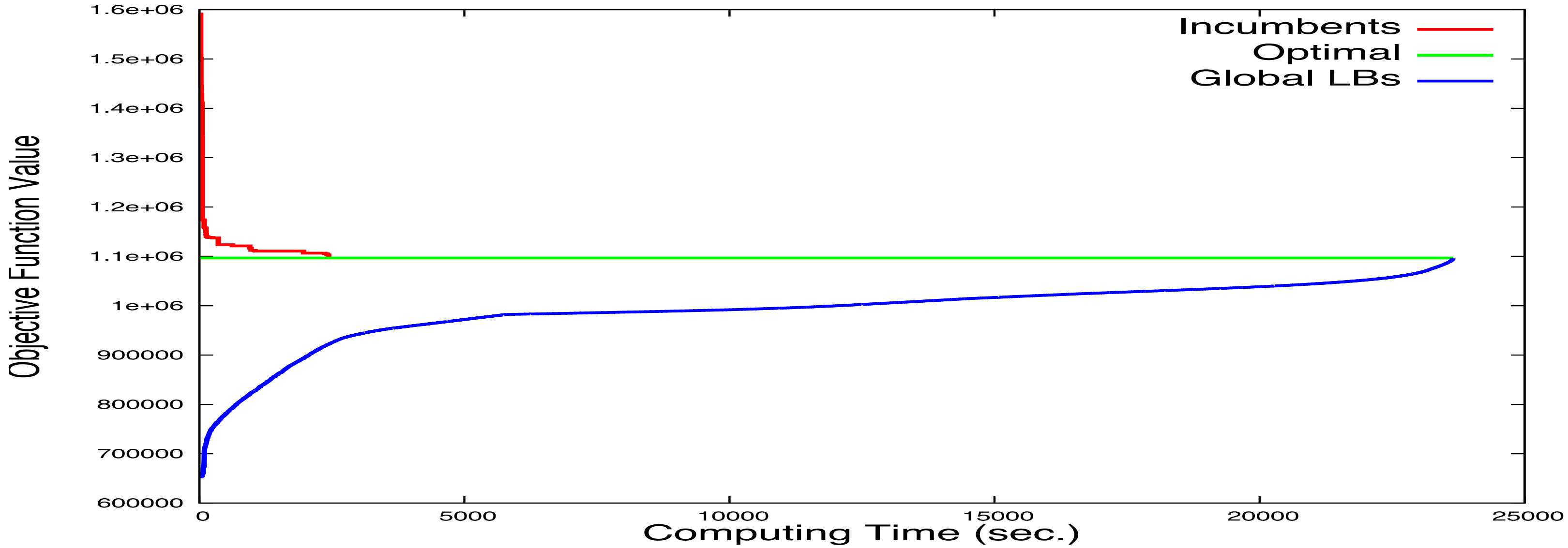
MODAL Mathematical Optimization and Data Analysis Laboratories

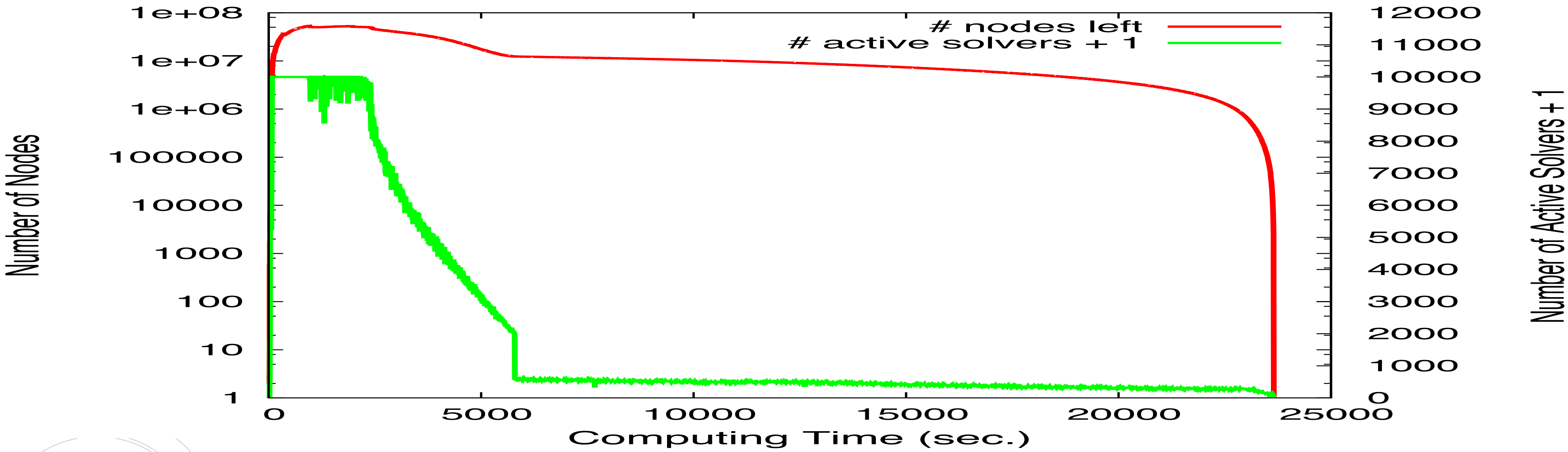
Runtime behavior: timtab2 – 4096 cores





Runtime behavior: timtab2 – 10000 cores

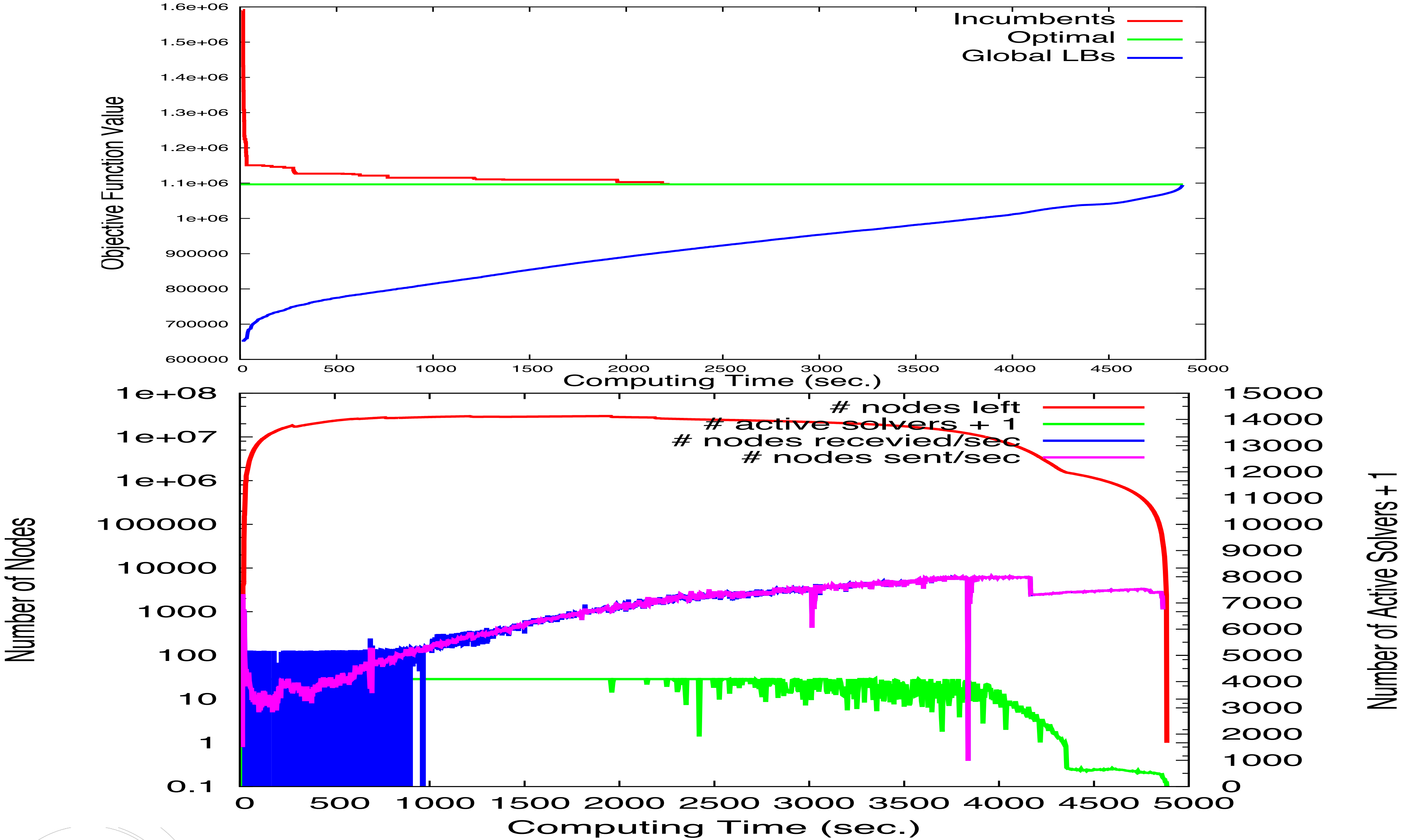




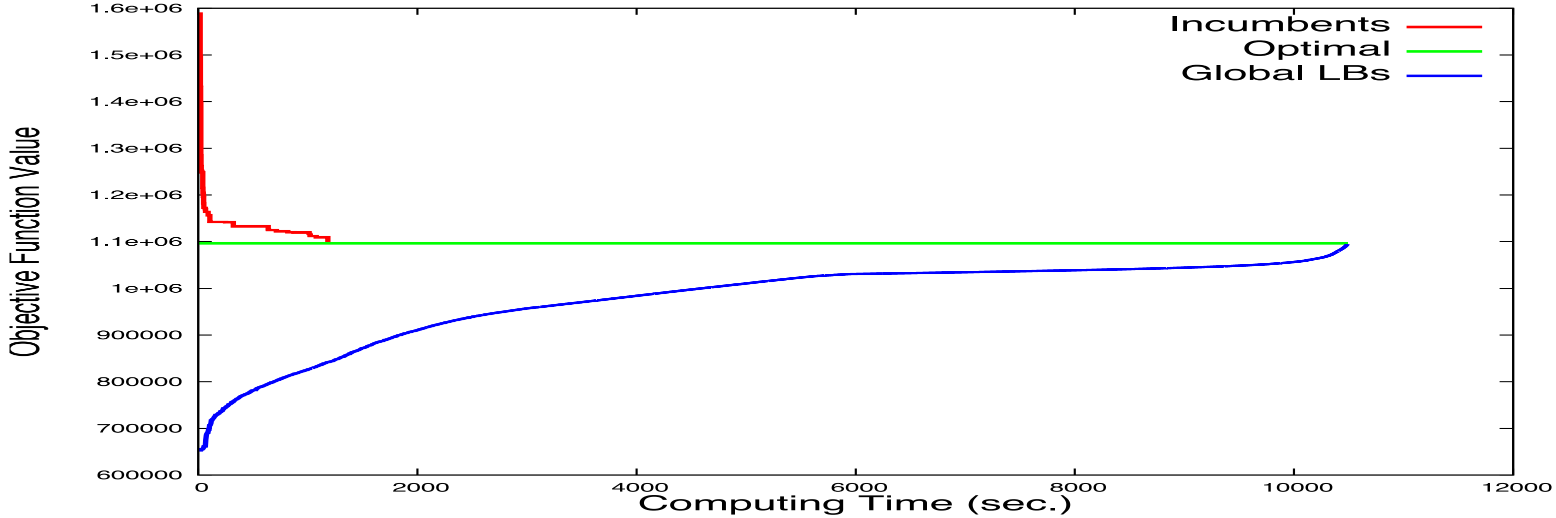
MODAL
Mathematical Optimization and Data Analysis Laboratories

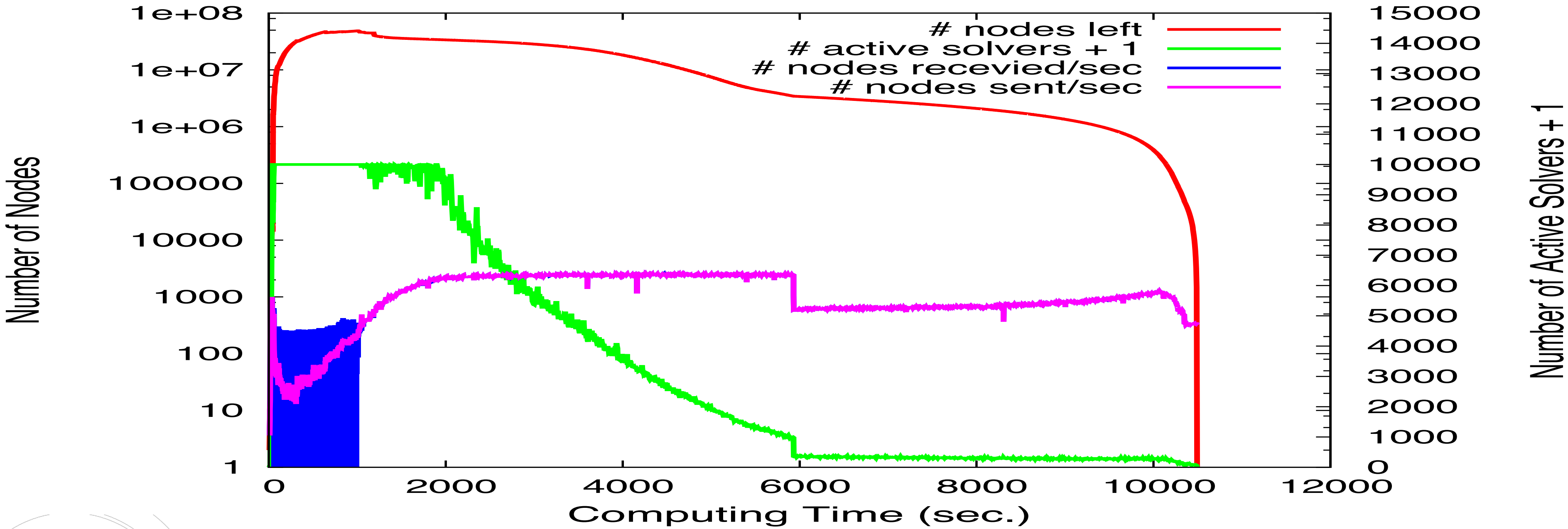
- Algorithmic improvements

Runtime behavior: timtab2 (improved) – 4096 cores

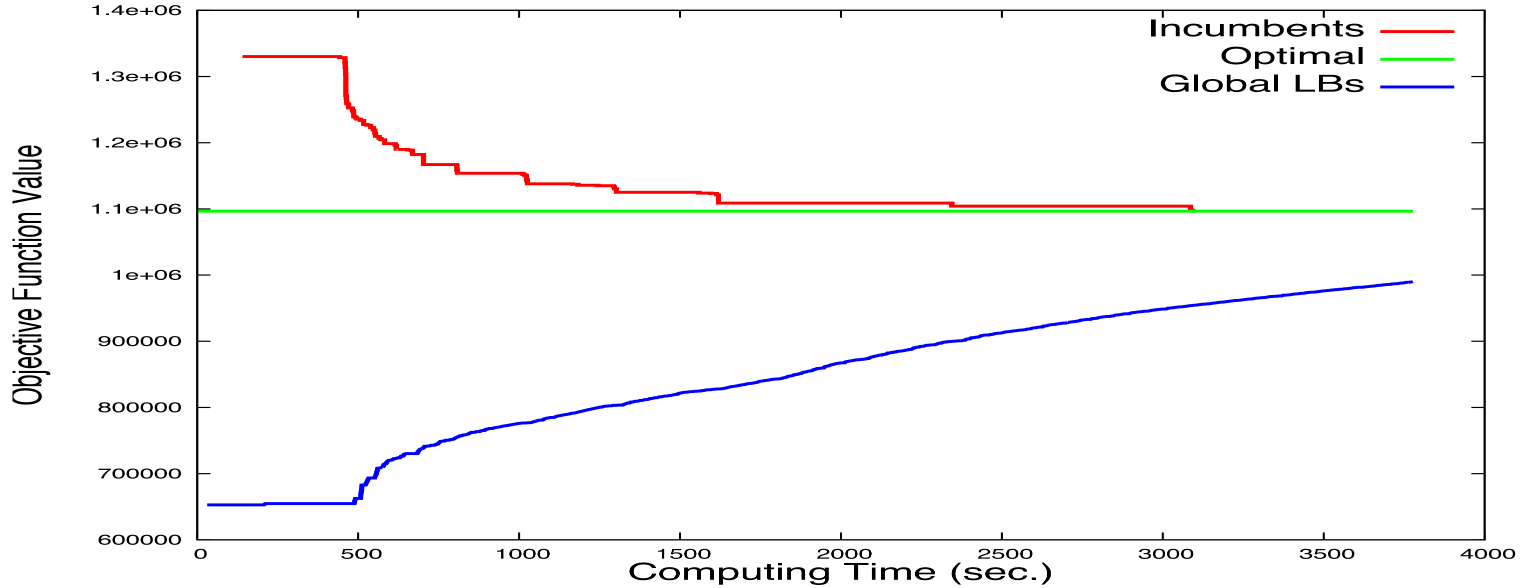


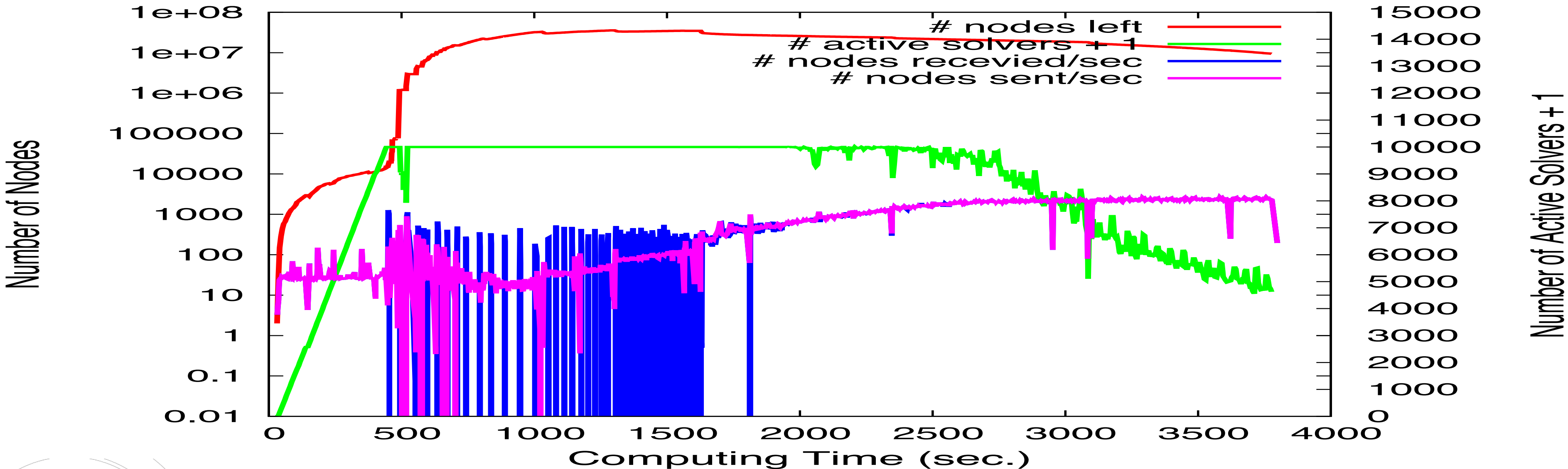
Runtime behavior: timtab2 (improved) – 10000 cores





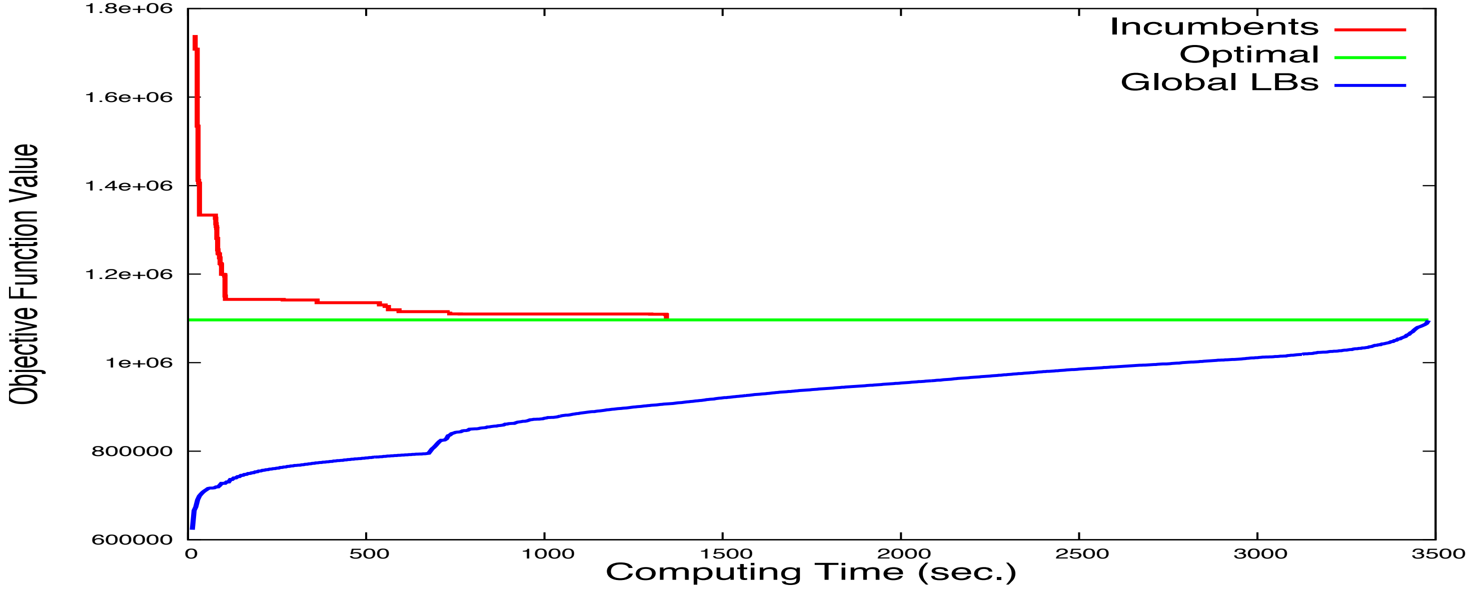
Runtime behavior: timtab2 (improved2) – 10000 cores

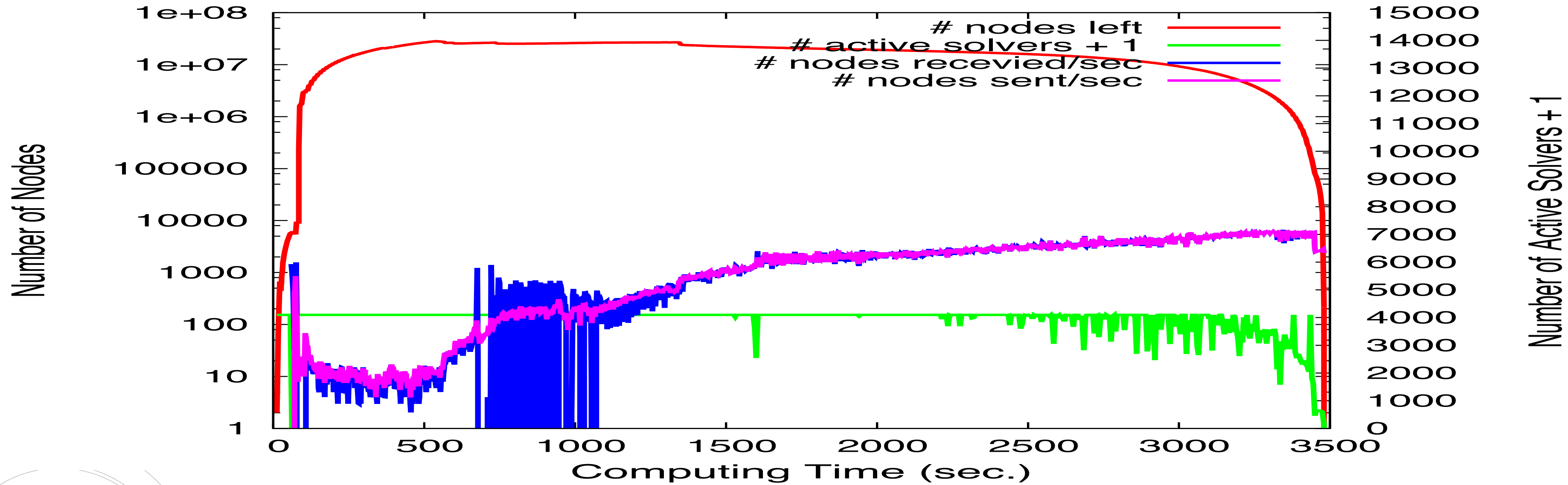




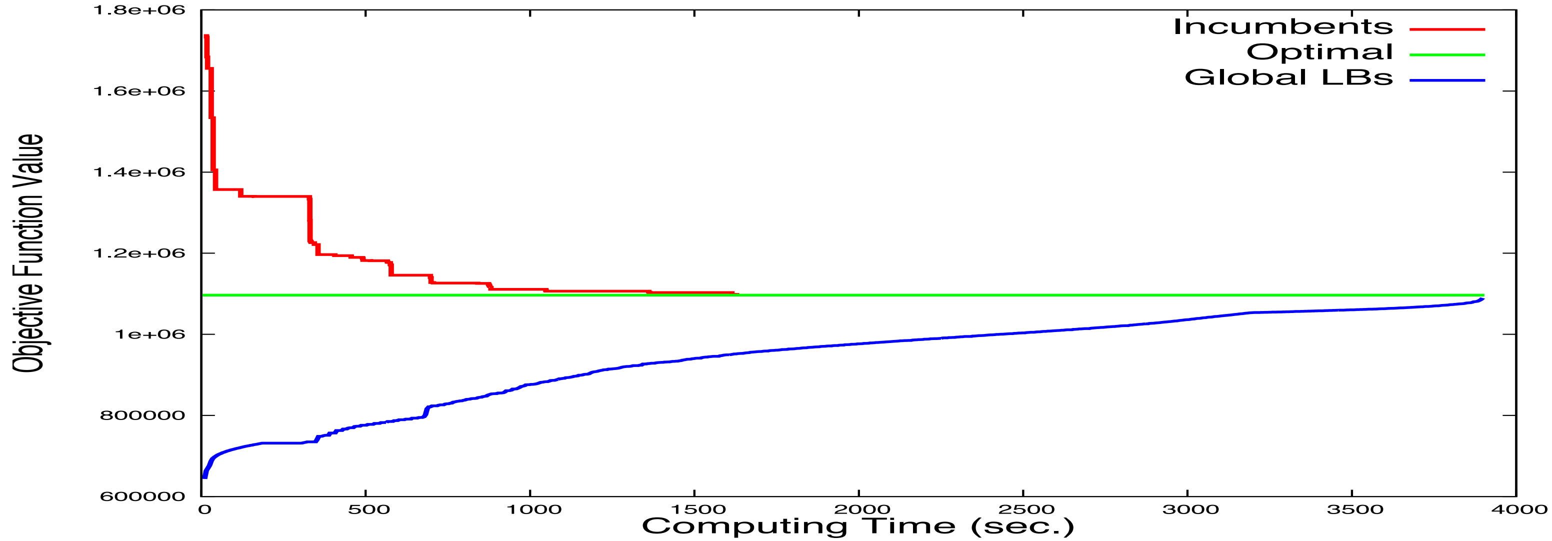
- Improvements of transfer methods
  (with racing ramp-ups)

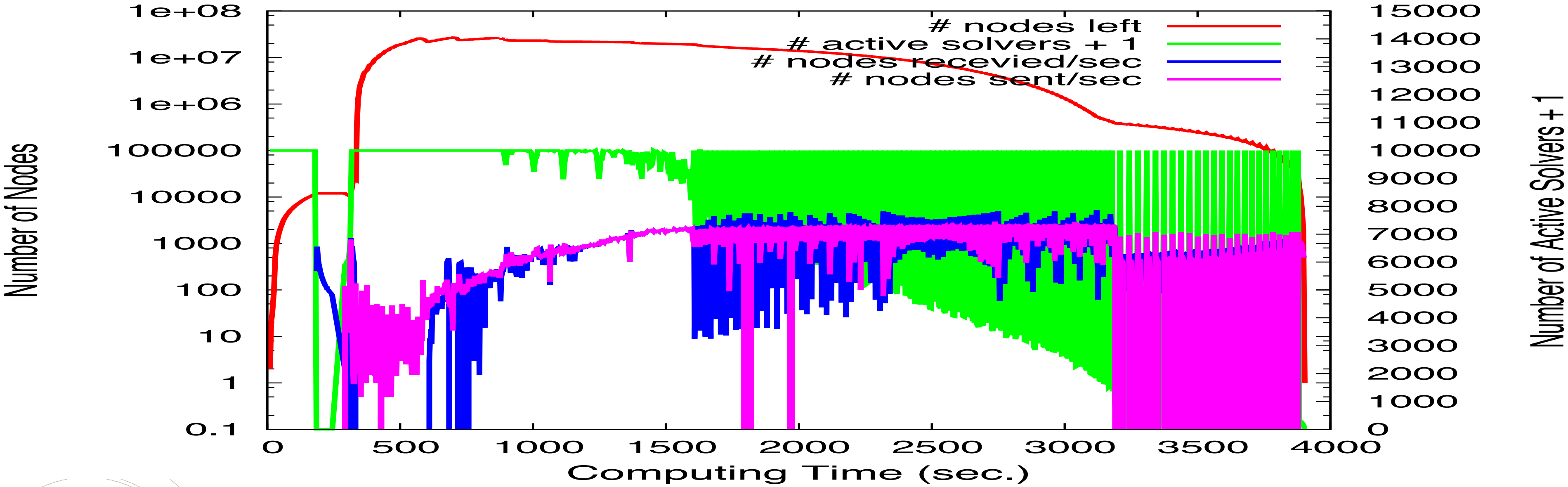
Runtime behavior: timtab2 (racing) – 4096 cores





Runtime behavior: timtab2 (racing) – 10000 cores

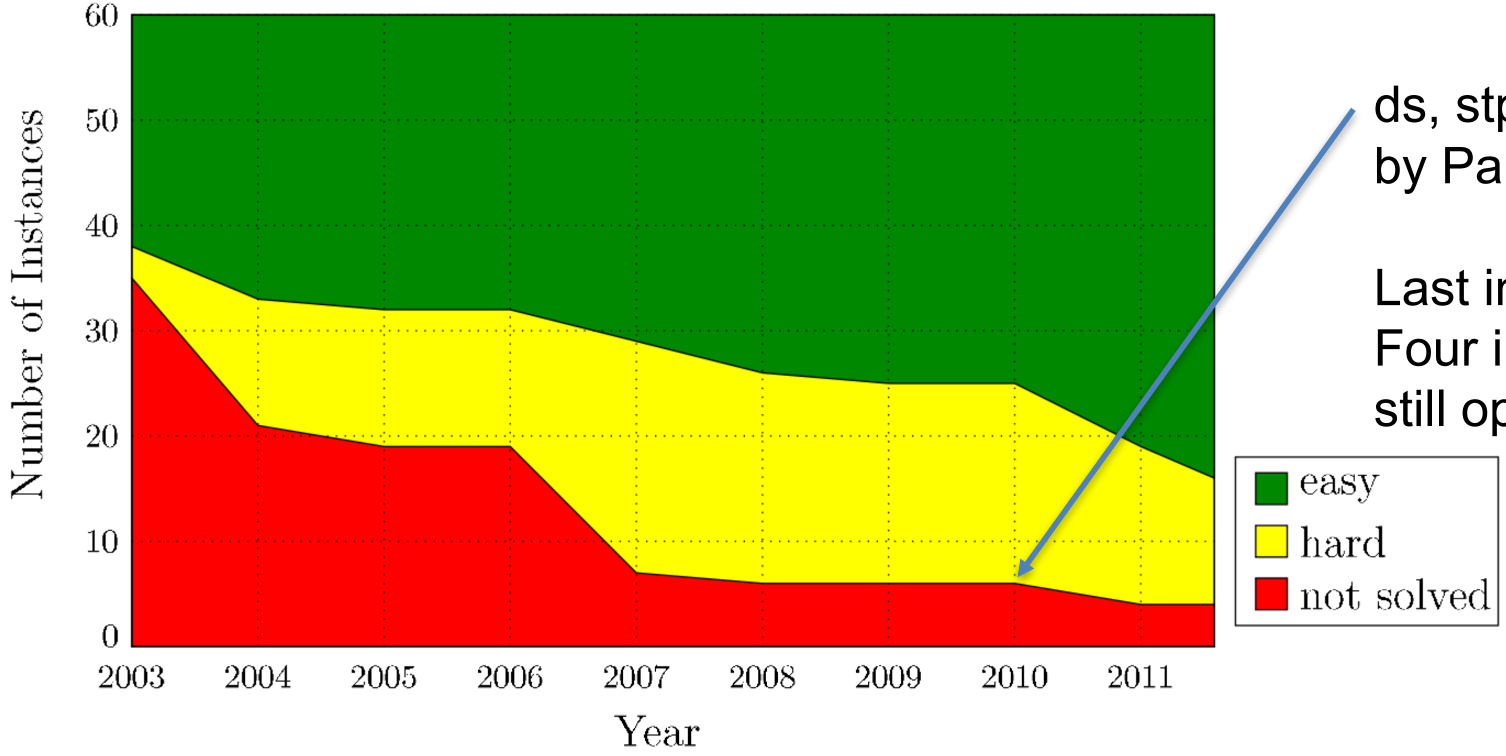




Outline

- Notes about the UG design and its dynamic load balancing
- Main computational results of ParaSCIP and ParaXpress
  - Solving previously unsolvable instances of MIP
- Some other projects with ParaSCIP
  - A new feature of SCIP is going to be parallelized
  - An application of ug[SCIP,*] libraries
- How UG applications were developed
- Future plan
- Concluding remarks

Solving open instances from MIPLIB2003



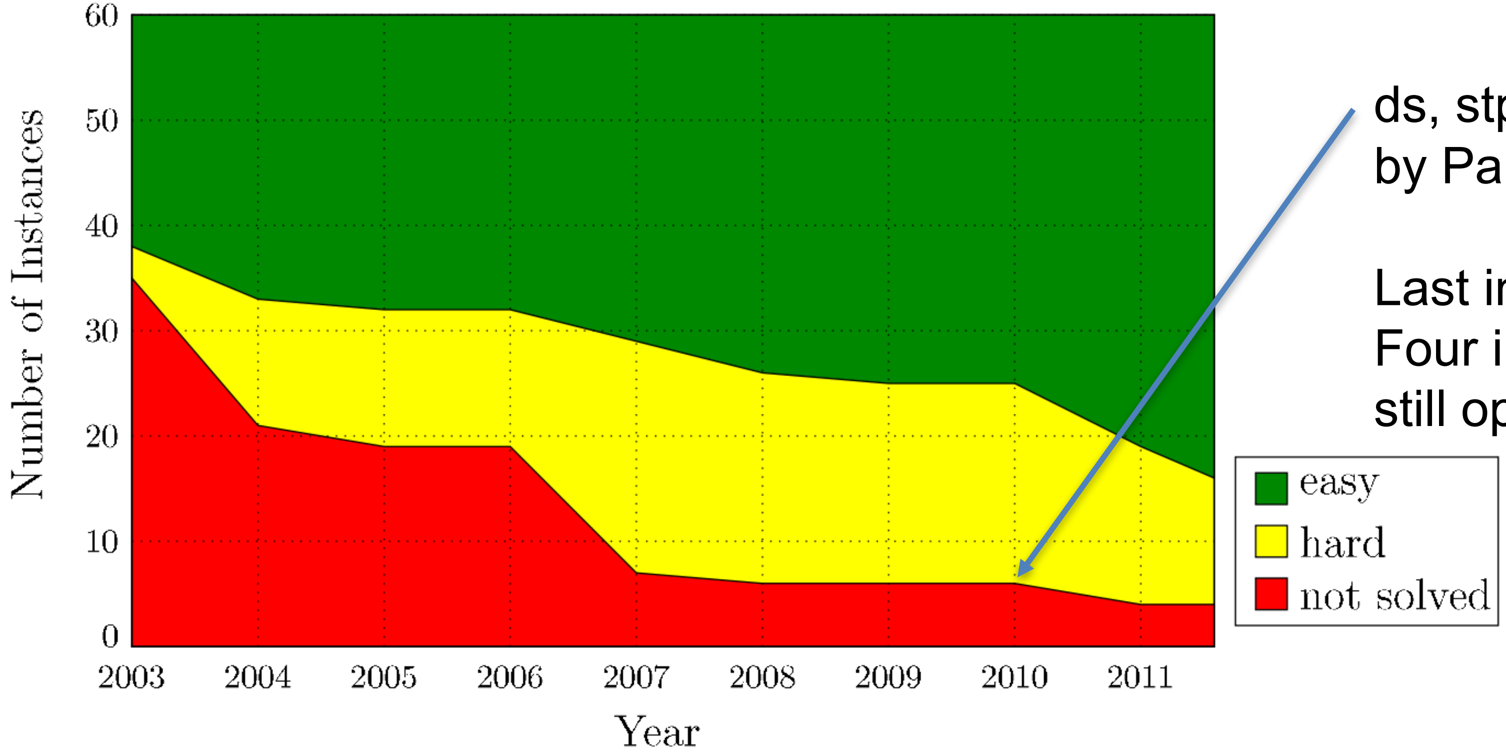
ds, stp3d were solved by ParaSCIP

Last instances solved. Four instances are still open.

6 instances have not been solved 7 Years

- ds – 656 constraints, 67,732 binary variables
- stp3d – 159,488 constraints, 204,880 binary variables
  After applying SCIP presolving 9 times:
  88,388 constraints, 123,637 binary variables

Solving open instances from MIPLIB2003



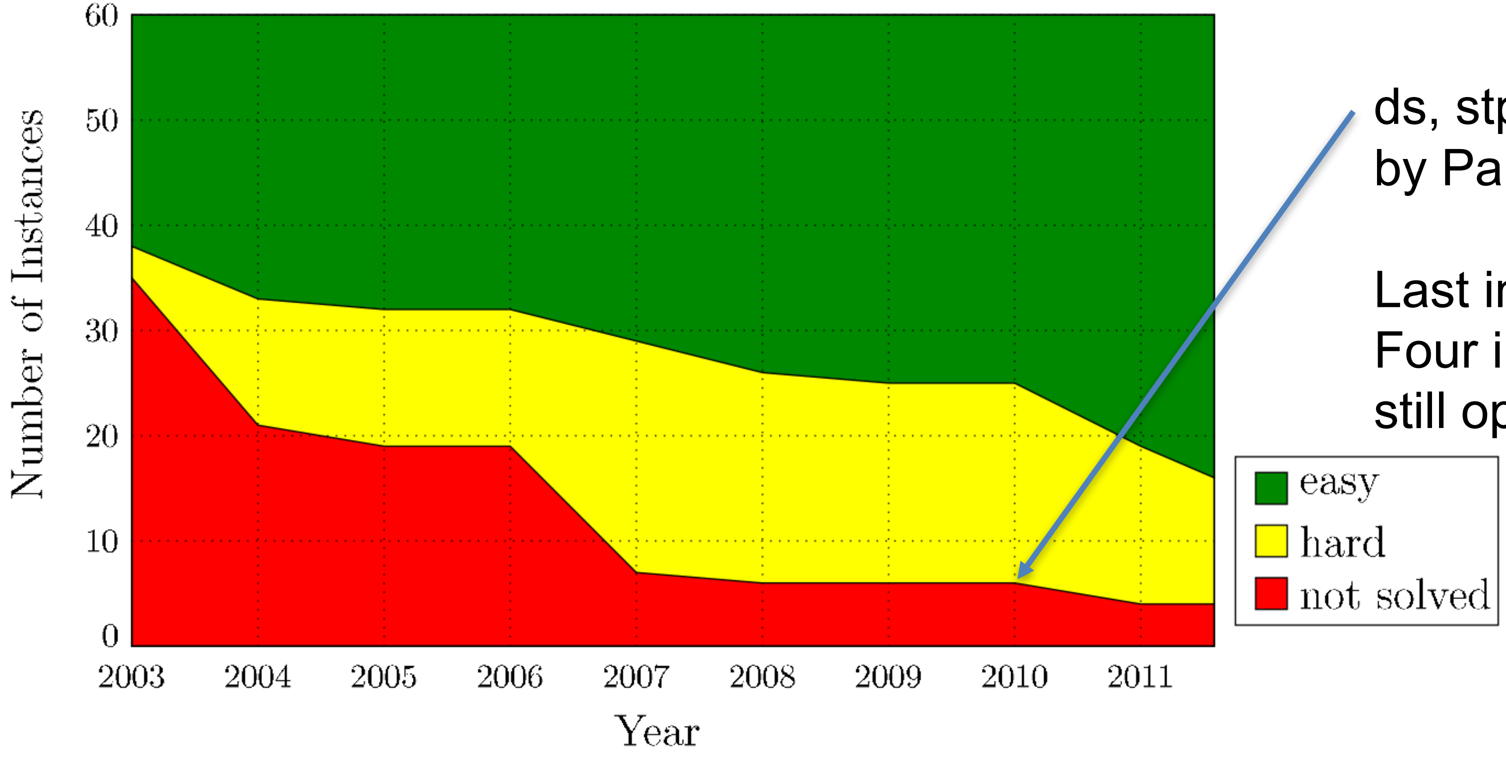
ds, stp3d were solved by ParaSCIP

Last instances solved. Four instances are still open.

ds: Set partitioning problem originating from public transport service planning.

This instance was solved by a first implementation of ParaSCIP using up to 2048 cores of HLRN-II(http://www.hlrn.de). ParaSCIP, mainly developed by Yuji Shinano, is an extension of SCIP and realizes a parallelization on a distributed memory computing environment. For being able to interrupt and warmstart the computations, ParaSCIP has a checkpoint mechanism. Therefore, selected subproblems are stored as warm start information, which allows to virtually run ParaSCIP, although the HLRN-II environment imposes a time limit of 48 hours per run. It took approximately 86 hours to solve this instance.

Solving open instances from MIPLIB2003



ds, stp3d were solved by ParaSCIP

Last instances solved. Four instances are still open.

stp3d: 3D Steiner Tree packing problem (VLSI routing problem in a multi-layer grid graph)

This instance was solved by a first implementation of ParaSCIP using up to 2048 cores of HLRN-II(http://www.hlrn.de). ParaSCIP, mainly developed by Yuji Shinano, is an extension of SCIP and realizes a parallelization on a distributed memory computing environment. For being able to interrupt and warmstart the computations, ParaSCIP has a checkpoint mechanism. Therefore, selected subproblems are stored as warm start information, which allows to virtually run ParaSCIP, although the HLRN-II environment imposes a time limit of 48 hours per run. The problem was presolved several times with SCIP presolving techniques. After that, it took approximately 114 hours to solve this instance.

Restarted runs vs. single run

- Which run was efficient from a view point of solving ds?
  - restarted runs: [STAGE 1(old)]
  - approximated comp. time: 86 [h]
  - real resource requested time: 96 [h]
  - cumulative time: 181,248 [h]
- single run: [STAGE 2(new)]
  - comp. time: 76[h]
  - cumulative time: 4096 x 76 = 311,296 [h]

HLRN II supercomputer (SGI Altix ICE 8200EX)
maximum cores for a job: 4096

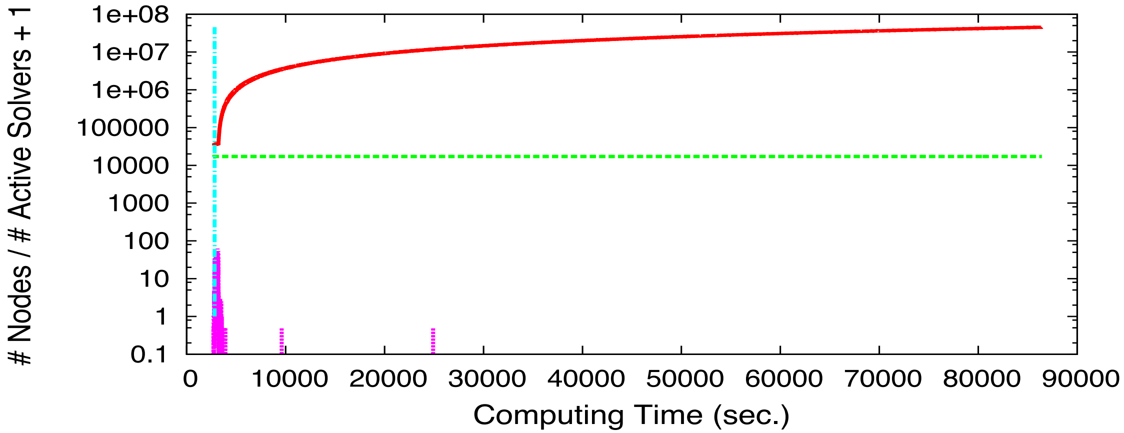
Answer is "restarted runs"

run	cores	time[h]
1	512	4
2	1024	5
3	2048	10
4	2048	7
5	2048	4
6	2048	5
7	2048	4
8	2048	5
9	2048	5
10	2048	4
11	2048	4
12	2048	5
13	2048	10
14	2048	4
15	2048	4
16	2048	12
17	2048	4
	Summary	96
	Accumulated time	181248

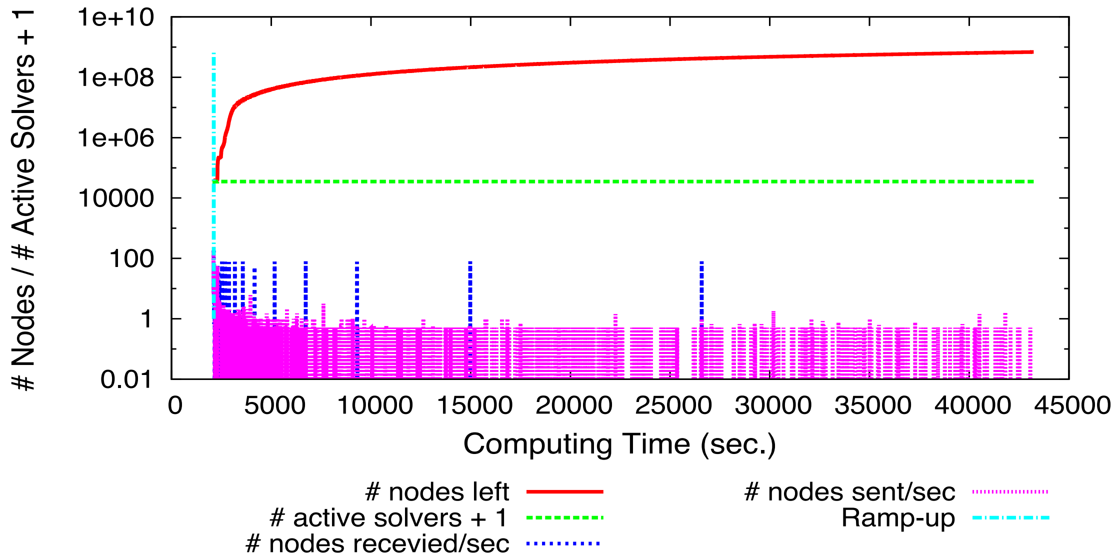
FiberSCIP and its paper

- Preliminary results were presented in MIPLIB2010 paper (Started working in 2010)
- Computational results had been checked two years
  - 87 instances of many different setttings with five repeated runs
  - All numbers in the supplement were checked carefully
- Submitted the paper in 2013
- Publish the paper in 2018

What does scale up do for a hard instance? (dano3mip)



on HLRN III
by using 17,088 cores
starting from 33,332
branch-and-bound nodes

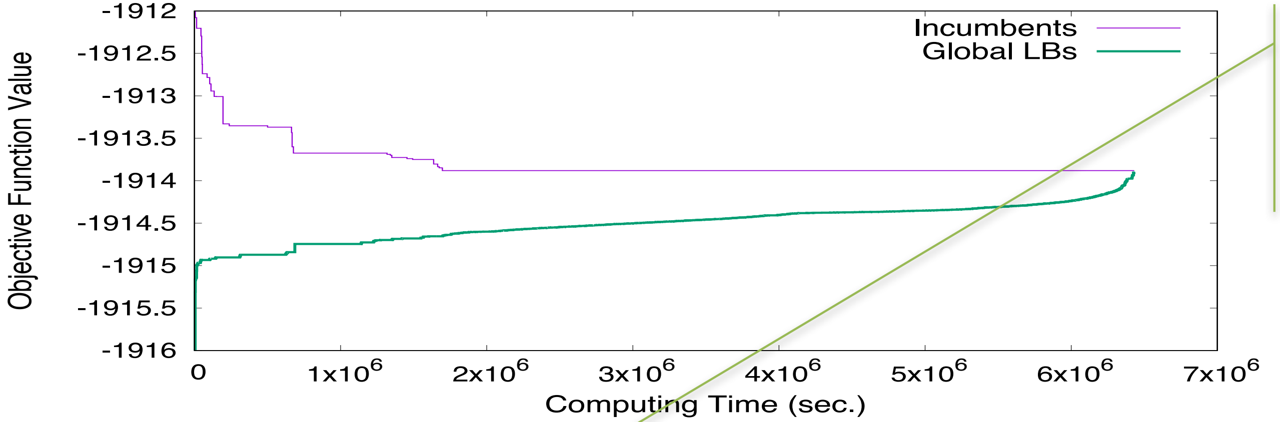


on Titan
by using 35,200 cores
starting from 33,481
branch-and-bound nodes

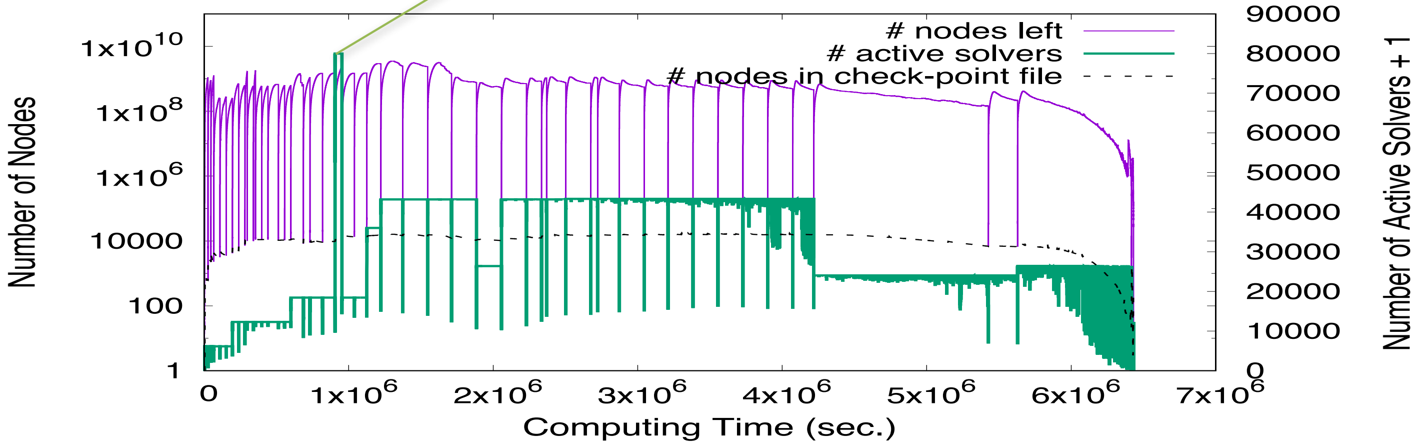
Large scale can break down problem into solvable sub-MIPs

The biggest and the longest computation

Solving rmine10: 48 restarted runs with 6,144 to 80,000 cores



How upper and lower bounds evolved



How open nodes and active solvers evolved

Titan with 80,000 cores
The others: HLRN III

It took about 75 days and 5,660 years of CPU core hours!

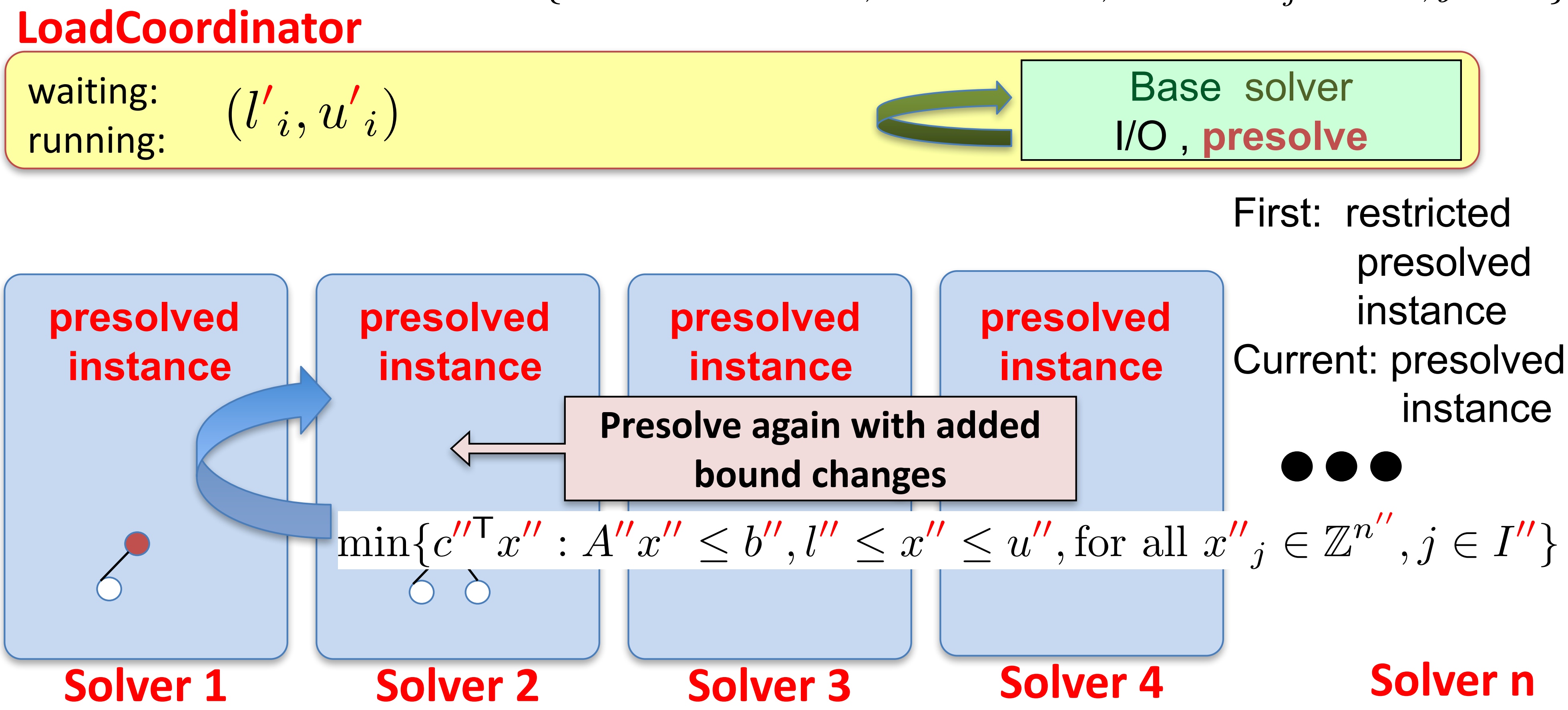
UG can handle up to 80,000 MPI process

MODAL
Mathematical Optimization and Data Analysis Laboratories

ParaXpress: Need to corporate with Xpress developers

Original $\min\{c^{\mathsf{T}}x : Ax \leq b, l \leq x \leq u, \text{for all } x_j \in \mathbb{Z}^n, j \in I\}$

$\min\{c'^{\mathsf{T}}x' : A'x' \leq b', l' \leq x' \leq u', \text{for all } x'_j \in \mathbb{Z}^{n'}, j \in I'\}$

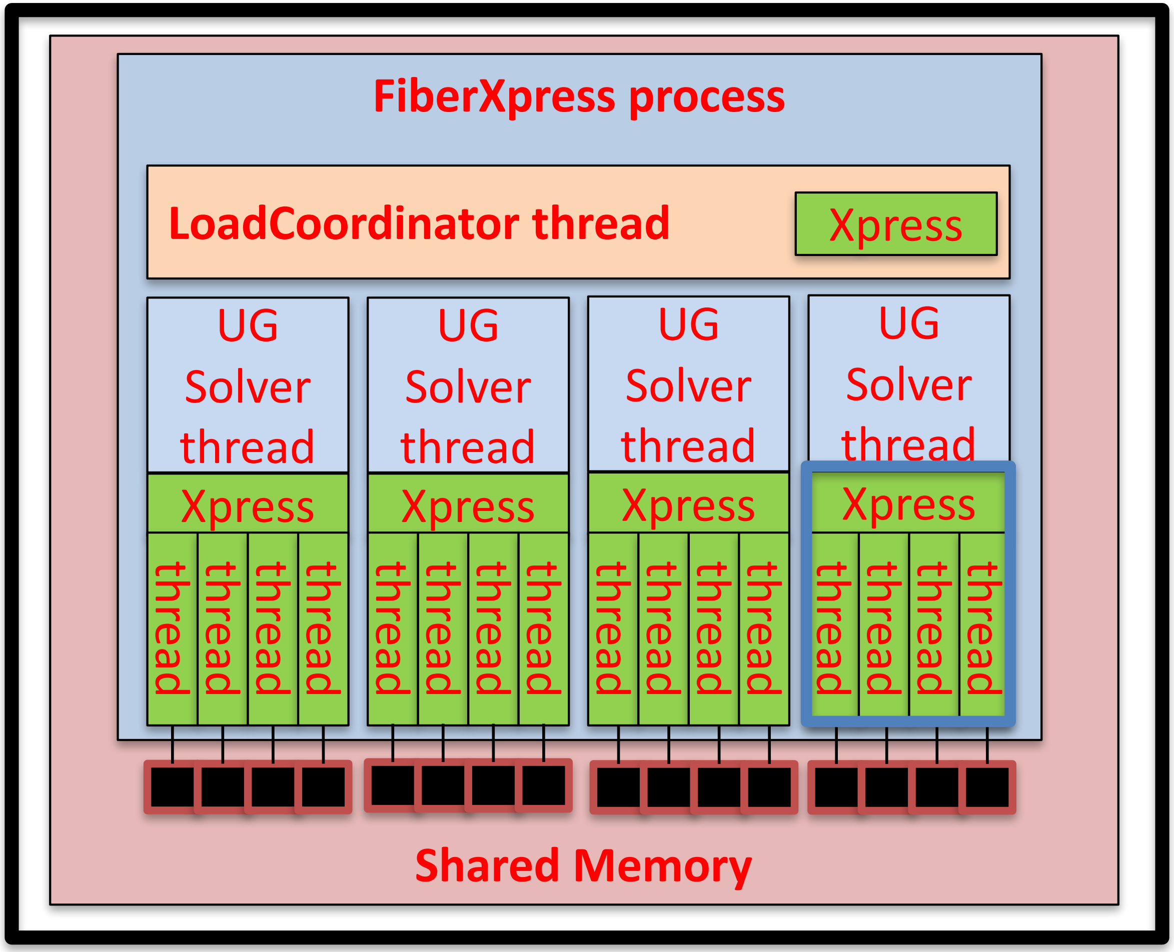


All transfer data need to be converted back for the presolved instance

All feasible solutions need to be converted back for the original instance

FiberXpress: UG Solver threads with Xpress threads

UG 4 Xpress 4



Shared Memory Machine (PC)

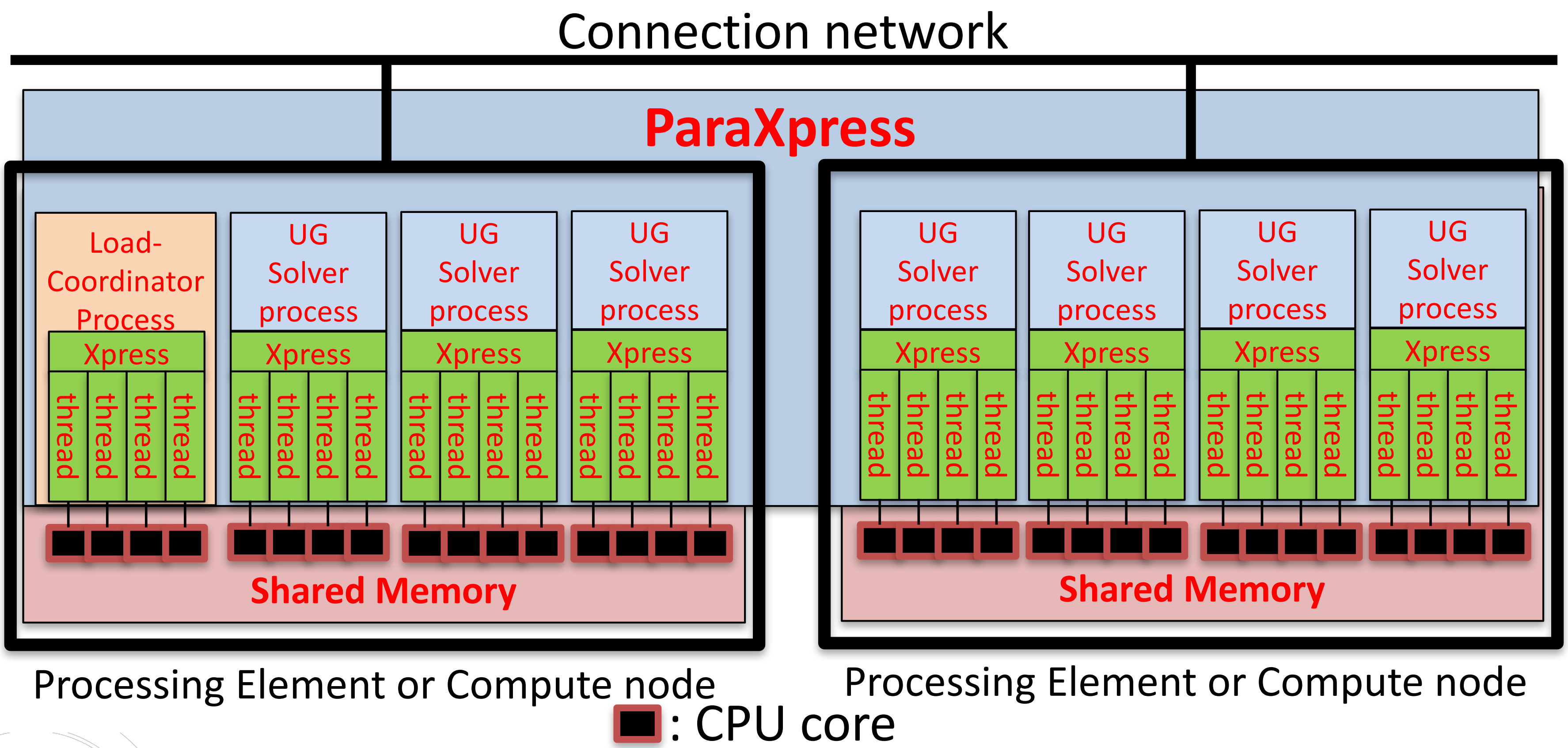
Berthold T., Farmer J., Heinz S., Perregaard M. (2016) Parallelization of the FICO Xpress-Optimizer. In: Greuel GM., Koch T., Paule P., Sommese A. (eds) Mathematical Software – ICMS 2016. ICMS 2016. Lecture Notes in Computer Science, vol 9725. Springer, Cham

■ : CPU core

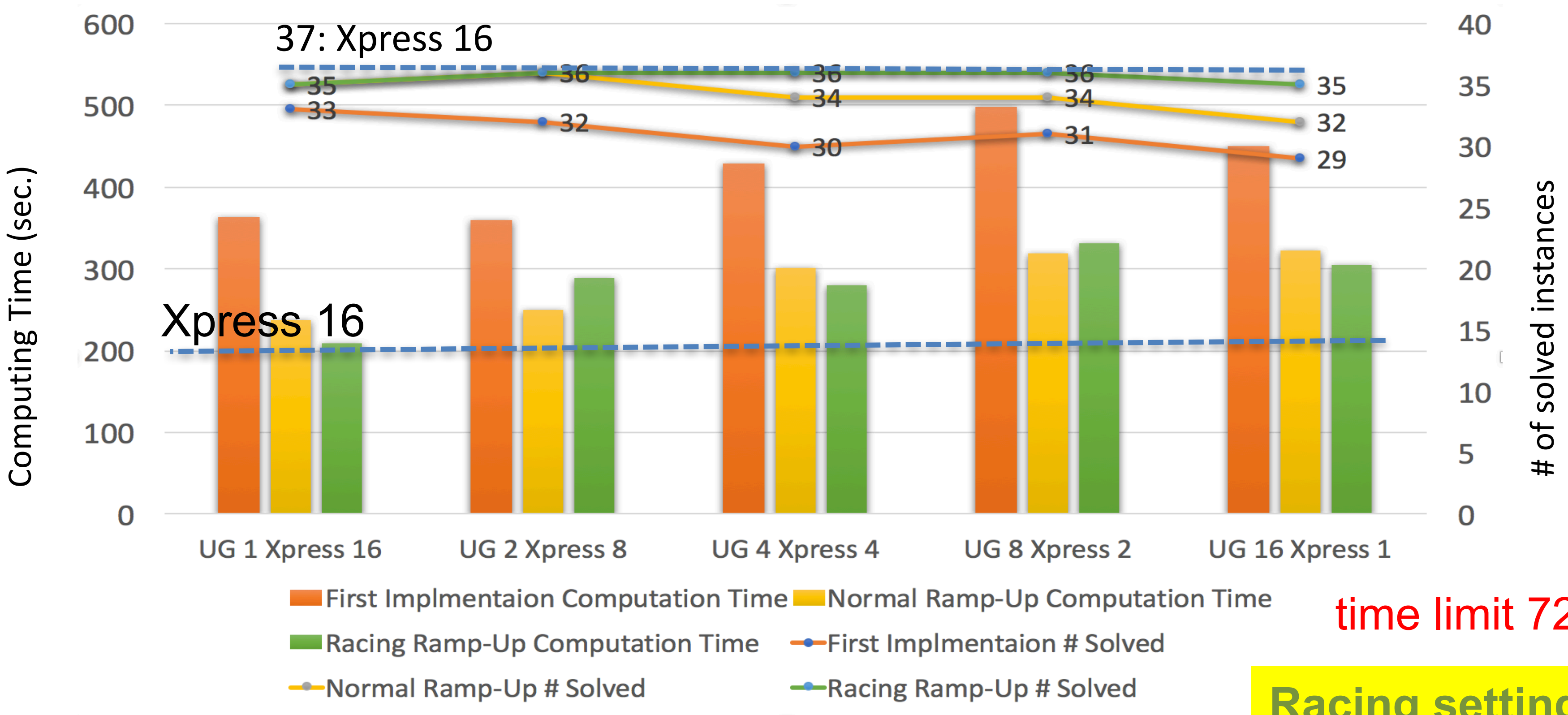
ParaXpress: UG Solver processes with Xpress threads

ug[Xpress,MPI] : ParaXpress

- A powerful massively parallel MIP solver
  - Can handle, hopefully efficiently, up to 80,000 (MPI processes) x 24 (threads) = 1,920,000 (cores)



Computational results of FiberXpress



time limit 7200 sec

Racing settings:
25 parameter settings
+ random seed

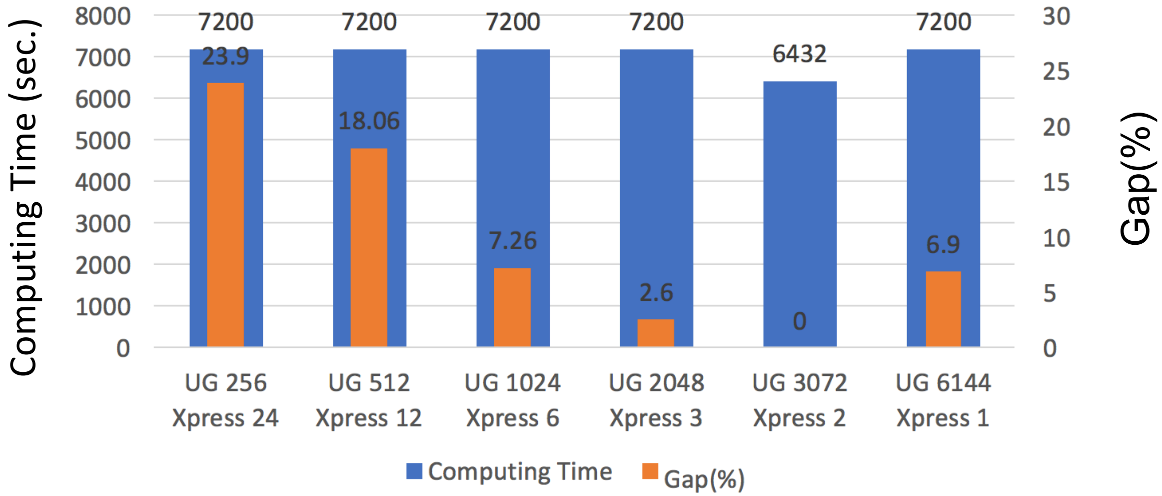
- **Xpress 8.3** and UG-0.8.4 – dev.
- 28 core Intel Xeon CPU E5-2690 v4 CPUs at 2.6 GHz with 35MB cache and 128GB memory
- 52 instances from Tree test set from MIPLIB2010 (6 instances were solved at root by Xpress 16)
- Geometric mean (Time out is counted as 7200 sec.)

Price of externalization is dramatically decreased

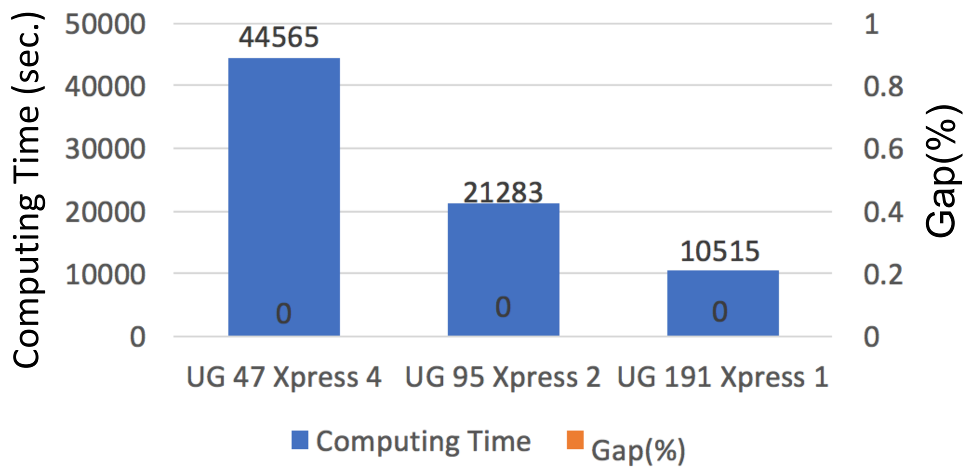
MODAL
Mathematical Optimization and Data Analysis Laboratories

Computational results of ParaXpress

- Solving timtab2 from MIPLIB2003
  - The experimental implementation
    - **Xpress 7.9** and UG-0.8.2
    - HLRN III: Cray XC30 with 24 core Intel E5-2695 v2 CPUs at 2.4GHz, 64GB Memory per computing node **6144 cores are used**

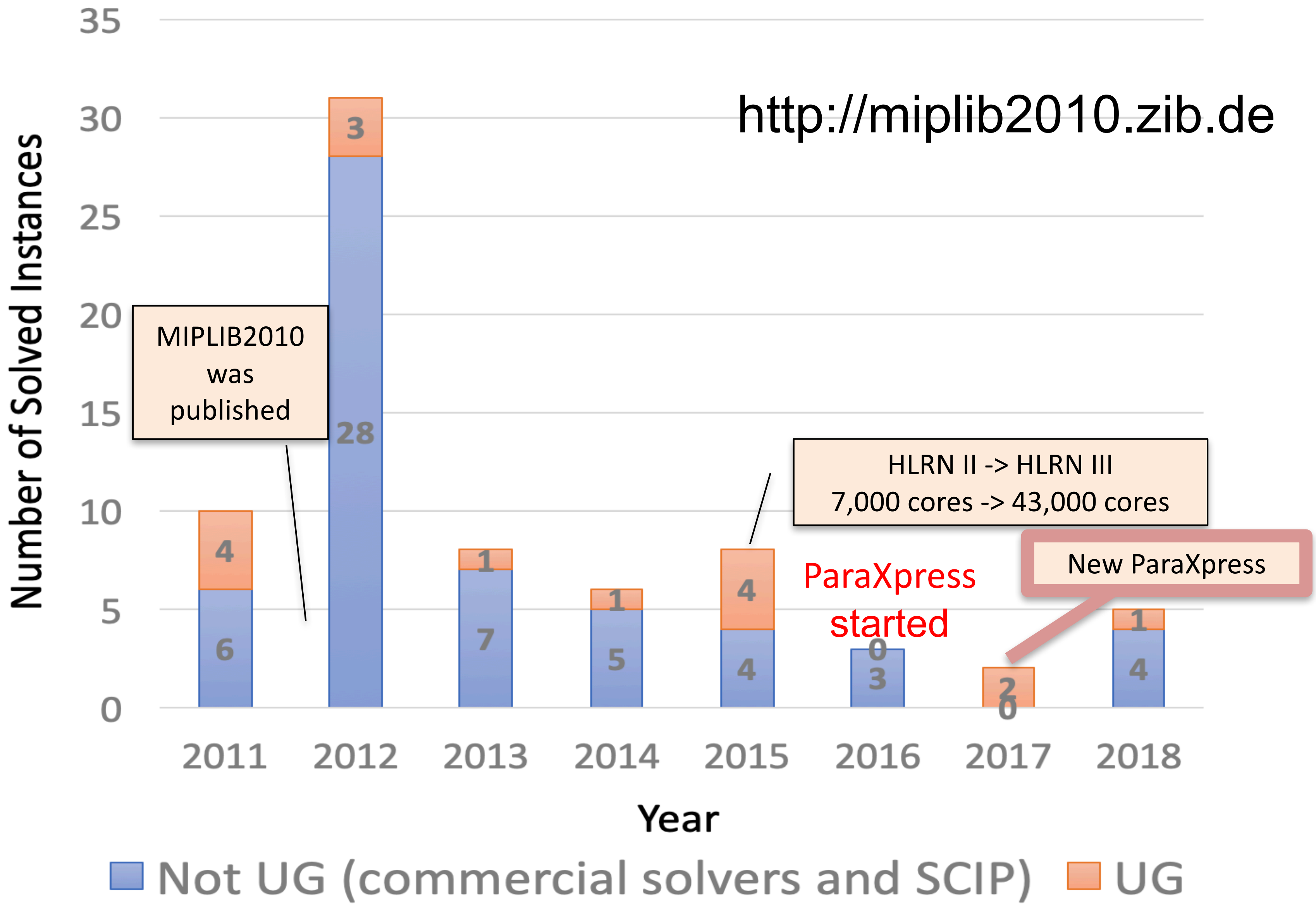


  - New implementation
    - **Xpress 8.3** and UG-0.8.4 – dev.
    - ISM Supercomputer System "I": SGI ICE X with 12 cores Intel Xeon E5-2697v2 at 2.7GHz (2CPUs), 128GB memory per computing node
    - **192 cores are used**



MODAL Mathematical Optimization and Data Analysis Laboratories

Solving open instances from MIPLIB2010



MODAL
Mathematical Optimization and Data Analysis Laboratories

Solving open instances from MIPLIB2017

- Open instances which could not be generated any feasible solution for all solvers

Instance	Solver	#Cores	Time to first sol. (sec.)	Value of first sol.	Time to solve (sec.)	Optimal value
fhnw-sq2	ParaSCIP	72	46,863	0	46,883	0
neos-4409277-trave	ParaSCIP	72	181,150	8	441,575	3
supportcase3	ParaSCIP	72	1,539	0	1,551	0
woodlands09	ParaSCIP	72	218,528	8	245,205	0
neos-3214367-sovi	ParaXpress	72	40,188	180,025	604,810(cont.)	178,326.9644(LB)
	ParaXpress	72	-	179,985(UB)	604,793(cont.)	178,236.7278(LB)
	ParaXpress	2,304	-	179,985(UB)	604,788(cont.)	179,186.6611(LB)
	ParaXpress	2,304	-	179,965(UB)	11,016	179,965
neos-3211096-shag	ParaSCIP	72	-	-	42,484	Infeasible
	ParaXpress	72	-	-	370,983	Infeasible
neos-3631363-vilnia	ParaSCIP	72	-	-	4,632	Infeasible
	ParaXpress	72	-	-	16,335	Infeasible

ISM (Institute of Statistical Mathematics) supercomputer, which is a HPE SGI 8600 with 384 compute nodes, each node has two Intel Xeon Gold 6154 3.0GHz CPUs (18 cores × 2) sharing 384GB of memory, by using ParaSCIP and ParaXpress

ParaXpress (= ug[Xpress, MPI]) status

- ParaXpress can be upgraded new version of Xpress without UG code change
- ParaXpress **is ready to run on over a million CPU cores!**
  - We need an application in which the optimal solution is very important in the research field

Outline

- Notes about the UG design and its dynamic load balancing
- Main computational results of ParaSCIP and ParaXpress
  - Solving previously unsolvable instances of MIP
- Some other projects with ParaSCIP
  - A new feature of SCIP is going to be parallelized
  - An application of ug[SCIP,*] libraries
- How UG applications were developed
- Future plan
- Concluding remarks

A new feature of SCIP (Added by Stephen J. Maher)

Features

- very fast standalone solver for linear programming (LP), mixed integer programming (MIP), and mixed integer nonlinear programming (MINLP)
- framework for branching, cutting plane separation, propagation, pricing, and Benders' decomposition,
- large C-API, C++ wrapper classes for user plugins

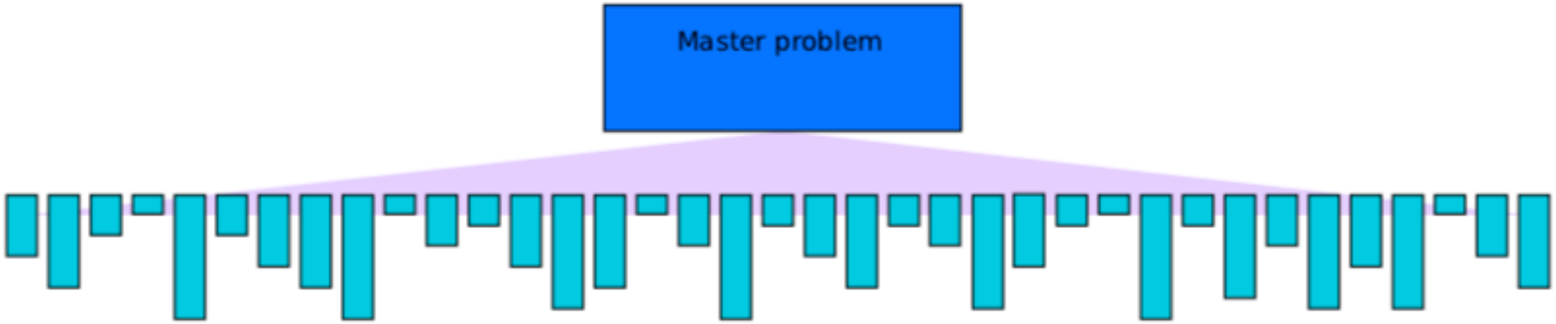
Parallelization of Benders' decomposition

- ▷ Benders' decomposition is easily parallelisable – however, not embarrassingly parallel
- ▷ There are lots of bottlenecks – long running master or subproblems
- ▷ Parallelization of Benders' decomposition can better employ available computational resources
- ▷ For stochastic programs, will help increase the scale of scenarios that can be handled and lead to improved solution quality

A new feature of SCIP (Added by Stephen J. Maher)

Three methods of parallelization

1. Tree search parallelization
   - Using the UG Framework – extended to transfer Benders' cuts between solvers
2. Subproblem parallelization
   - OpenMP used for shared memory parallelization of subproblem solving
3. Hybrid tree search and subproblem parallelization
   - Distributed memory tree search parallelization with shared memory subproblem parallelization



Outline

- Notes about the UG design and its dynamic load balancing
- Main computational results of ParaSCIP and ParaXpress
  - Solving previously unsolvable instances of MIP
- Some other projects with ParaSCIP
  - A new feature of SCIP is going to be parallelized
  - An application of ug[SCIP,*] libraries
- How UG applications were developed
- Future plan
- Concluding remarks

Overview

Developed by Keiji Kimura

Problems

- AIC-based variable selection
  in linear regression and logistic regression
  $\Rightarrow$ mixed integer nonlinear programming problems (**MINLP**)

Algorithm proposed in [1, 2]

- Branch-and-bound (B&B) algorithm using SCIP and UG
- Developed additional plugins:
  - Relaxation handler (to compute lower bounds efficiently)
  - Primal heuristics (to find good solutions early)
  - Branching rules (to reduce branch-and-bound nodes)

---

[1] K. Kimura and H. Waki: Minimization of Akaike's information criterion in linear regression analysis via mixed integer nonlinear program. OMS, 33(3), 633–649, 2018.
[2] K. Kimura: Application of a mixed integer nonlinear programming approach to variable selection in logistic regression. JORSJ, 62(1), to appear.

Problems

Developed by Keiji Kimura

Variable Selection

- provides a simple statistical model
- finds a subset of relevant variables
- improves prediction performance

Direct Objective Optimization in Variable Selection

- f : the goodness-of-fit (how well a model fits a given dataset)
- $\lambda \sum z_j (\lambda > 0)$: a penalty for the number of variables

- The problems in [1, 2] are of the following form

$$\begin{aligned} \min_{\beta, z} \quad & f(\beta) + \lambda \sum_{j \in J} z_j \\ \text{s.t.} \quad & z_j = 0 \Rightarrow \beta_j = 0 \quad \forall j \in J, \\ & \beta_j \in \mathbb{R}, z_j \in \{0, 1\} \quad \forall j \in J. \end{aligned}$$

[1] K. Kimura and H. Waki: Minimization of Akaike's information criterion in linear regression analysis via mixed integer nonlinear program. OMS, 33(3), 633–649, 2018.
[2] K. Kimura: Application of a mixed integer nonlinear programming approach to variable selection in logistic regression. JORSJ, 62(1), to appear.

Numerical Experiments in [2]

Problem: AIC-based variable selection in logistic regression

Comparison of Approaches:

- Customized B&B algorithm [2]
  - using SCIP and UG
- Piecewise linear approximation approach [Sato, et al., 2016]
  - using CPLEX to solve MILP problems
  - employing existing heuristics for initial solutions

Benchmark datasets: UCI machine learning repository

(https://archive.ics.uci.edu/ml/)

[2] K. Kimura: Application of a mixed integer nonlinear programming approach to variable selection in logistic regression. JORSJ, 62(1), to appear.

Numerical Experiments in [2]

(16 threads)			Approach 1		Approach 2	
Name	n	p	AIC	Time(s)	AIC	Time(s)
`bumps`	2584	22	**1097.1**	**20.0**	1098.1	41.5
`breast-P`	194	34	147.0	**25.8**	147.0	112.4
`biodeg`	1055	42	653.3	**221.5**	653.3	>5000
`stat-G`	1000	62	958.2	>5000	958.2	>5000
`madelon`	2000	500	**2502.1**	>5000	2504.0	>5000

- Approach 1: Customized B&B algorithm [2] (SCIP and UG)
- Approach 2: Piecewise linear approximation approach (CPLEX)
- n: the number of observations
- p: the number of candidates for variables

[2] K. Kimura: Application of a mixed integer nonlinear programming approach to variable selection in logistic regression. JORSJ, 62(1), to appear.

Outline

- Notes about the UG design and its dynamic load balancing
- Main computational results of ParaSCIP and ParaXpress
  - Solving previously unsolvable instances of MIP
- Some other projects with ParaSCIP
  - A new feature of SCIP is going to be parallelized
  - An application of ug[SCIP,*] libraries
- How UG applications were developed
- Future plan
- Concluding remarks

Instantiated parallel solvers by UG

The following solvers have developed by Yuji Shinano cooperated with each solver developers

- Single thread base solver
  - ParaSCIP: ug[SCIP, MPI], FiberSCIP: ug[SCIP, Pthreads/C++11]
- Multi-threaded base solver
  - ParaXpress: ug[Xpress, MPI], FiberXpress: ug[Xpress, Pthreads/C++11]
- Concorde: TSP solver
  - ug[Concorde, MPI], ug[Concorde, Pthreds/C++11]
  - ug[Concorde/MPI, MPI]
    - **Concorde/MPI** has developed by Utz-Uwe Haus

Instantiated parallel solvers by UG

The following solvers have developed
by each base solver developers.

Skelton code was distributed.
Design and coding together with Yuji Shinano
about one week in the beginning

- Multi-threaded base solver
  - ParaNUOPT: ug[NUOPT, MPI], FiberNUOPT: ug[NUOPT, Pthreads/C++11]
    - Only tested with single thread NUOPT
- Distributed base solver
  - ug[PIPS-SBB, MPI]
    - PIPS-SBB: a distributed memory solver for Stochastic MIP

Skelton code

- baseSolverParaComm.h
- baseSolverParaCommMpi.cpp
- baseSolverParaCommMpi.h
- baseSolverParaCommPointHdlr.cpp
- baseSolverParaCommPointHdlr.h
- baseSolverParaCommTh.cpp
- baseSolverParaCommTh.h
- baseSolverParaDef.h
- baseSolverParaDeterministicTimer.h
- baseSolverParaDiffSubproblem.cpp
- baseSolverParaDiffSubproblem.h
- baseSolverParaDiffSubproblemMpi.cpp
- baseSolverParaDiffSubproblemMpi.h
- baseSolverParaDiffSubproblemTh.h
- baseSolverParaInitiator.cpp
- baseSolverParaInitiator.h

- baseSolverParaInstance.h
- baseSolverParaInstanceMpi.cpp
- baseSolverParaInstanceMpi.h
- baseSolverParaInstanceTh.cpp
- baseSolverParaInstanceTh.h
- baseSolverParaSolution.cpp
- baseSolverParaSolution.h
- baseSolverParaSolutionMpi.cpp
- baseSolverParaSolutionMpi.h
- baseSolverParaSolutionTh.cpp
- baseSolverParaSolutionTh.h
- baseSolverParaSolver.cpp
- baseSolverParaSolver.h
- fbaseSolver.cpp
- parabaseSolver.cpp

Rename: baseSolver->your_solver_name, Fill codes for all virtual functions

baseSolverPara**CommPoint**Hdlr.cpp: **CommPoint** is a callback of solver to communicate with LoadCorrdinator

Built parallel solvers with ug[SCIP,*] libraries

The following solvers have developed
by each SCIP plugins developers.

Debugged with Yuji Shinano

- ug[SCIP-Jack,*]
  - Parallel solvers for solving Steiner Tree Problems
- ug[SCIP-SDP,*]
  - Parallel solvers for solving Mixed Integer Semidefinite Programs

Outline

- Notes about the UG design and its dynamic load balancing
- Main computational results of ParaSCIP and ParaXpress
  - Solving previously unsolvable instances of MIP
- Some other projects with ParaSCIP
  - A new feature of SCIP is going to be parallelized
  - An application of ug[SCIP,*] libraries
- How UG applications were developed
- Future plan
- Concluding remarks

Plan for UG 1.0

- If you have your own state-of-the-art solver,
  please try to parallelize it with UG
- If you have SCIP application,
  please try to parallelize it with ug[SCIP,*] libraries

We hope to have feedbacks from user side

- Second UG workshop will be held in two years,
  since "Sustainable Infrastructures for Archiving and Publishing High-Performance Optimization Software" is three years project
  - Right after the workshop, UG 1.0 would be released

Outline

- Notes about the UG design and its dynamic load balancing
- Main computational results of ParaSCIP and ParaXpress
  - Solving previously unsolvable instances of MIP
- Some other projects with ParaSCIP
  - A new feature of SCIP is going to be parallelized
  - An application of ug[SCIP,*] libraries
- How UG applications were developed
- Future plan
- Concluding remarks

Concluding remarks

- A research platform for parallel branch-and-bound
  - Users can run all parallel solvers by themselves for fair direct comparison
    - source codes are publically available
    - test data are publically available