Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

YUJI SHINANO

# The Ubiquity Generator Framework: 7 Years of Progress in Parallelizing Branch-and-Bound

This work has also been supported by the Research Campus Modal *Mathematical Optimization and Data Analysis Laboratories* funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM), and partially supported by the BMWi project Realisierung von Beschleunigungsstrategien der anwendungsorientierten Mathematik und Informatik für optimierende Energiesystemmodelle - BEAM-ME (fund number 03ET4023DE). All responsibility for the content of this publication is assumed by the authors.

ZIB Report 17-60 (November 2017)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

# The Ubiquity Generator Framework: 7 Years of Progress in Parallelizing Branch-and-Bound*

Yuji Shinano†

November 7, 2017

**Abstract**

Mixed integer linear programming (MILP) is a general form to model combinatorial optimization problems and has many industrial applications. The performance of MILP solvers has improved tremendously in the last two decades and these solvers have been used to solve many real-word problems. However, against the backdrop of modern computer technology, parallelization is of pivotal importance. In this way, ParaSCIP is the most successful parallel MILP solver in terms of solving previously unsolvable instances from the well-known benchmark instance set MIPLIB by using supercomputers. It solved two instances from MIPLIB2003 and 12 from MIPLIB2010 for the first time to optimality by using up to 80,000 cores on supercomputers. ParaSCIP has been developed by using the Ubiquity Generator (UG) framework, which is a general software package to parallelize any state-of-the-art branch-and-bound based solver. This paper discusses 7 years of progress in parallelizing branch-and-bound solvers with UG.

## 1 Introduction

This paper deals with the *Ubiquity Generator (UG) framework* [20, 24], a software package that allows to parallelize branch-and-bound (B&B) solvers—in particular solvers for mixed integer linear programming (MILP) problems. The standard algorithm used to solve MILP is an LP-based branch-and-bound with many advanced procedures such as primal heuristics, preprocessing and conflict analysis, which implicitly enumerates the whole solution space to find an optimal solution. The reader is referred to [10] for details about these procedures and the latest survey of parallel MILP solvers. This paper presents the ground design and general features of UG, current development based on it, and discusses 7 years of progress in parallelizing branch-and-bound solvers with UG.

*This work has been supported by the Research Campus MODAL *Mathematical Optimization and Data Analysis Laboratories* funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM). All responsibility for the content of this publication is assumed by the authors.

†Zuse Institute Berlin, Takustraße 7, 14195 Berlin, Germany, shinano@zib.de

# 2 Towards a General Branch-and-Bound Parallelization

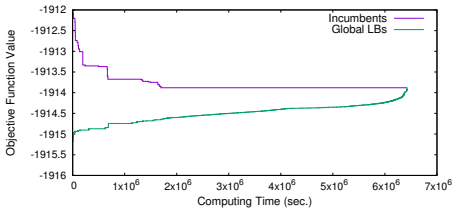
Standardization of the message passing interface started in the mid-90s. In the same period of time, general parallel branch-and-bound software framework/library development started [21, 2, 23]. Comparing between a sequential sophisticated B&B implementation and a naive parallel B&B one for solving an optimization problem, the former is overwhelmingly high-performance in terms of solvability. In order to investigate effectiveness of parallelization for a sophisticated B&B implementation, the CPLEX solver was parallelized by using PUBB2 [19]. However, it soon turned out that parallelizing a black-box solver with a general parallel B&B framework does not easily lead to a significantly enhanced performance. Therefore, the development of ParaLEX [18] was started, which was specialized for the CPLEX solver and could run on distributed memory environments. Yet, when ParaLEX was redesigned in 2008 [16] by the author of this article, the idea of developing a general software framework to exploit state-of-the-art MILP solvers re-emerged and subsequently gave rise to the UG framework described in the following.

## 2.1 The Ubiquity Generator (UG) Framework

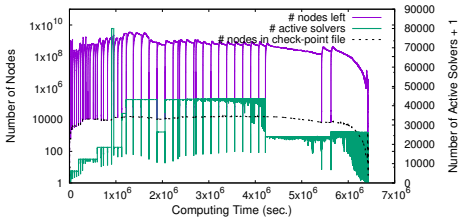
UG is a generic framework to parallelize any existing state-of-the-art B&B based solver, subsequently referred to as *base solver*. UG is composed of a collection of base C++ classes, which define interfaces that can be customized for any base solver and allow descriptions of subproblems and solutions to be translated into a solver independent form. Additionally, there are base classes that define interfaces for different message-passing protocols. Implementations of ramp-up, dynamic load balancing, and check-pointing and restarting mechanisms are available as a generic functionality (see details in [20]). The B&B tree is maintained as a collection of subtrees by the base solvers, while UG only extracts and manages a small number of subproblems from the base solvers for load balancing.

The concept of UG is thus to abstract from a base solver and parallelization library and to provide a framework that can be used, in principle, to parallelize any powerful state-of-the-art base solver on any computational environment. For a particular base solver, only the interface to UG in form of specializations of base classes needs to be implemented. Similarly, for a particular parallelization library, a specialization of an abstract UG class is necessary.

A particular instantiated parallel solver is referred to as ug [a specific solver name, a specific parallelization library name]. Here, the specific parallelization library is used to realize the message-passing based communications. In [10], recent parallel MILP solvers are summarized in terms of aspects such as load coordination mechanisms, or granularity of working unit. According to the term defined in [10], UG employs a Supervisor-Worker coordination mechanism with subtree-level parallelism (the unit of work is a subtree). One of the most important characteristics of UG is that it makes algorithmic changes to that of the base solver, such as multiple presolving, and performs very adaptive algorithms, such as racing ramp-up [20] and distributed domain propagation [5].




Upper and lower bounds




Nodes and active solvers

Figure 1: Evolution of computation for solving **rmine10** by using up to 80,000 cores

## 2.2 Instantiated parallel solvers by UG

The following parallel solvers are instantiated by UG. The current distribution of UG has the capability to use the parallelization libraries MPI (Message Passing Interface) and pthreads (POSIX Threads).

**Academic solver SCIP as the base solver** Two solvers have been developed for the academic SCIP solver [11], ParaSCIP (= ug [SCIP, MPI]) [13] and FiberSCIP (= ug [SCIP, Pthreads]) [20]. Algorithmically, both solvers are identical, since they are parallelized by the same software framework UG. The run-time behavior has been investigated in detail for the MIPLIB2010 benchmark instances by using FiberSCIP. ParaSCIP successfully solved 14 previously unsolved instances from MIPLIB2003 and MIPLIB2010 as of writing this document [14, 15]. The longest and the biggest scale computation conducted to solve an open instance by ParaSCIP is presented in Fig.1. The `rmine10` instance was solved for the first time with 48 restarted runs from checkpoint files that were generated by previous runs using between 6144 and 80,000 cores. In total, it took about 75 days and 6,405 years of CPU core hours.

**Commercial solver FICO Xpress as the base solver** Two solvers have been developed for the commercial Xpress the solvers, ParaXpress (= ug [Xpress, MPI]) and FiberXpress (= ug [Xpress, Pthreads]). Xpress itself is a shared memory parallel MILP solver. Therefore, FiberXpress can be viewed as a multi-level threaded parallel shared memory MILP solver. When there is more than one core, it is necessary to decide how cores are assigned to UG threads and how many to the Xpress threads. The assignment also changes the solving behavior of the algorithm. ParaXpress has the same assignment issue in between UG processes and FICO Xpress internal threads. The difference in assignments was investigated in [17].

**Distributed memory parallel solver PIPS-SBB as the base solver** UG has also been used to parallelize the PIPS-SBB [8] solver for two-stage stochastic programming problems (ug [PIPS-SBB,MPI]) [9]. PIPS-SBB can solve large-scale LPs on distributed memory computing environments. Therefore, this parallel solver instantiation shows that UG is capable of parallelizing distributed memory parallel base solvers.

### 2.3 Instantiated parallel solvers by ug [SCIP,*] libraries

UG has been developed mainly in concert with SCIP. Therefore, ug [SCIP,*] is the most mature and also has user-customizable libraries. By using these libraries with the plug-in architecture of SCIP, a customized parallel solver can be developed with minimal effort.

**ug [SCIP-Jack, *** ] One of the most successful results of using this development mechanism is the SCIP-Jack solver for Steiner tree problems and its variants: ug [SCIP-Jack, MPI] solved three open instances from the SteinLib[22] benchmark set [4]. The SCIP Optimization Suite contains all source codes of this parallel solver. Only one file with 116 lines of code (without comments) in the source code of ug is required for the parallelization of SCIP-Jack.

**ug [SCIP-Scheduler, *** ] The SCIP applications moreover contain a Scheduler, which is a solver for resource-constrained project scheduling problems [1]. Also this solver can be parallelized by ug [SCIP,*] libraries.

**ug [SCIP-SDP, *** ] The Mixed Integer Semidefinite Programming (MISDP) solver has been developed in a project of SCIP-SDP [12] at TU Darmstadt. The solver is realized as plugin for SCIP. Therefore, this solver can be parallelized by ug [SCIP,*]; libraries and the code will be published in the near future.

## 3 UG synthesizer (UGS)

The strategy of composing multiple heuristic algorithms within a single solver that chooses the best suited one for each input is called *algorithm portfolio*. In order to exploit performance variability [3, 6] for MILP solving, a solver may solve an instance in parallel with several different configurations of parameters (including parameter for permutation of columns and rows of input data). This procedure is called *racing* [20]. It is a natural idea to run several (parallel) heuristic solvers together with several B&B based solvers in parallel to share a good primal solution to cut-off search trees in the B&B based solvers. *UG synthesizer (UGS)* is a software framework to realize this strategy on a distributed memory computing environment; it is a general framework to realize any combinations of algorithm portfolio and racing.

Although a parallel solver instantiated by UG has a single executable file and runs as a SPMD (Single Program, Multiple Data) model MPI program, a parallel solver configured by UGS has several executable files and it runs as MPMD (Multiple Program, Multiple Data) model MPI program. In UGS solvers, a heuristic solver or a B&B solver has a separate executable file and is referred to as a UGS solver that can be a distributed memory parallel solver. Currently, shared memory parallel B&B UGS solvers ugs_Xpress, ugs_CPLEX, and ugs_Gurobi have been developed. As for these solvers, the corresponding commercial solvers are extended to run as UGS solvers. And also, distributed memory heuristic UGS solvers ugs_PAC_CPLEX and ugs_PAC_Xpress have been developed. These are implementations of alternative criteria search [7] using different MILP solvers. Any ug [*,*] solver can run as a UGS solver.UGS provides one special executable file ugs, which mediates incumbent solutions among the UGS solvers.

A parallel UGS solver can be configured at run-time flexibly. For example, it runs with ugs, ugs_Xpress1, ugs_Xpress2, ugs_CPLEX1, ugs_PAC_CPLEX1 and ug [Xpress,MPI]1. The different numbers at the end of the same solver name denote the multiple solvers for one UGS solver can run in parallel with different parameter settings. The solver configuration is specified by a special file and is passed to each solver at run-time. Therefore, the configuration can be decided flexibly depending on the computing environment used to solve an instance. On top of that, whenever a new promising algorithm implementation has appeared, a new UGS solver can be added without any modification of the other existing solvers, since the executable file is separate.

# 4 Concluding remarks

Some of the instances solved by ParaSCIP for the first time are currently solvable by commercial solvers on a common desktop machine in a reasonable amount of time. This can be taken as an indication that algorithmic improvements are more crucial than parallelizations. Nevertheless, by providing a way to apply large-scale parallelization to the latest algorithm implementations, UG has in many cases succeeded to "look ahead in time" and in some cases helped to guide sequential solver development.

Besides this fact, a solver instantiated by UG causes algorithmic changes to that of the base solver by adding more cores, though the program code is the same. An open question is whether this kind of algorithmic change can fundamentally help to increase the solvability of problems or not.

# References


[1] Berthold, T., Heinz, S., Lübbecke, M.E., Möhring, R.H., Schulz, J.: A constraint integer programming approach for resource-constrained project scheduling. In: A. Lodi, M. Milano, P. Toth (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, *Lecture Notes in Computer Science*, vol. 6140, pp. 313–317. Springer (2010)

[2] Cun, B.L., Roucairol, C., The PNN Team: BOB: a Unified Platform for Implementing Branch-and-Bound like Algorithms. Rapports de Recherche 95/16, PRiSM (1995)

[3] Danna, E.: Performance variability in mixed integer programming (2008). Presentation, Workshop on Mixed Integer Programming (MIP 2008), Columbia University, New York. http://coral.ie.lehigh.edu/~jeff/mip-2008/talks/danna.pdf

[4] Gamrath, G., Koch, T., Maher, S.J., Rehfeldt, D., Shinano, Y.: SCIP-Jack—a solver for stp and variants with parallelization extensions. Mathematical Programming Computation **9**(2), 231–296 (2017)

[5] Gottwald, R.L., Maher, S.J., Shinano, Y.: Distributed domain propagation. ZIB-Report 16-71, Zuse Institute Berlin, Takustr. 7, 14195 Berlin (2016). Leibniz International Proceedings in Informatics SEA 2017 (to appear)

[6] Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Math. Prog. Comp. **3**, 103–163 (2011)

[7] Munguía, L.M., Ahmed, S., Bader, D.A., Nemhauser, G.L., Shao, Y.: Alternating criteria search: A parallel large neighborhood search algorithm for mixed integer programs. Submitted for publication (2016)

[8] Munguía, L.M., Oxberry, G., Rajan, D.: PIPS-SBB: A parallel distributed-memory branch-and-bound algorithm for stochastic mixed-integer programs. In: 2016 IEEE IPDPSW, pp. 730–739 (2016)

[9] Munguía, L.M., Oxberry, G., Rajan, D., Shinano, Y.: Parallel pips-sbb: Multi-level parallelism for stochastic mixed-integer programs. ZIB-Report 17-58, Zuse Institute Berlin (2017)

[10] Ralphs, T., Shinano, Y., Berthold, T., Koch, T.: Parallel solvers for mixed integer linear programming. Tech. Rep. 16-74, ZIB, Takustr.7, 14195 Berlin (2016)

[11] SCIP: Solving Constraint Integer Programs. http://scip.zib.de/

[12] SCIP-SDP: a mixed integer semidefinite programming plugin for SCIP. http://www.opt.tu-darmstadt.de/scipsdp/

[13] Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T.: ParaSCIP – a parallel extension of SCIP. In: C. Bischof, H.G. Hegering, W.E. Nagel, G. Wittum (eds.) Competence in High Performance Computing 2010, pp. 135–148. Springer (2012)

[14] Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T., Winkler, M.: Solving hard MIPLIB2003 problems with ParaSCIP on supercomputers: An update. In: Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, pp. 1552–1561 (2014)

[15] Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T., Winkler, M.: Solving open MIP instances with ParaSCIP on supercomputers using up to 80,000 cores. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 770–779. IEEE Computer Society, Los Alamitos, CA, USA (2016)

[16] Shinano, Y., Achterberg, T., Fujie, T.: A dynamic load balancing mechanism for new ParaLEX. In: Proceedings of ICPADS 2008, pp. 455–462 (2008)

[17] Shinano, Y., Berthold, T., Heinz, S.: A First Implementation of ParaXpress: Combining Internal and External Parallelization to Solve MIPs on Supercomputers, pp. 308–316. Springer International Publishing, Cham (2016). URL http://dx.doi.org/10.1007/978-3-319-42432-3_38

[18] Shinano, Y., Fujie, T.: ParaLEX: A parallel extension for the CPLEX mixed integer optimizer. In: F. Cappello, T. Herault, J. Dongarra (eds.)

Recent Advances in Parallel Virtual Machine and Message Passing Interface. Proceedings, pp. 97–106. Springer Berlin Heidelberg (2007). DOI 10.1007/978-3-540-75416-9_19

[19] Shinano, Y., Fujie, T., Kounoike, Y.: Effectiveness of parallelizing the ILOG-CPLEX mixed integer optimizer in the PUBB2 framework. In: H. Kosch, L. Böszörményi, H. Hellwagner (eds.) Euro-Par 2003 Parallel Processing: Proceedings, pp. 451–460. Springer Berlin Heidelberg (2003)

[20] Shinano, Y., Heinz, S., Vigerske, S., Winkler, M.: FiberSCIP – a shared memory parallelization of SCIP. INFORMS Journal on Computing **30**(1), 11–30 (2018). DOI 10.1287/ijoc.2017.0762. URL https://doi.org/10.1287/ijoc.2017.0762

[21] Shinano, Y., Higaki, M., Hirabayashi, R.: A generalized utility for parallel branch and bound algorithms. In: Proceedings.Seventh IEEE Symposium on Parallel and Distributed Processing, pp. 392–401 (1995). DOI 10.1109/SPDP.1995.530710

[22] SteinLib Testdata Library. http://steinlib.zib.de/steinlib.php

[23] Tschöke, S., Polzer, T.: Prortabl Parallel Branch-and-Bound Library PPBB-Lib. User manual version 2.0, University of Paderborn (1996)

[24] UG: Ubiquity Generator framework. http://ug.zib.de/