Generalized UG

Generalization of Ubiquity Generator Framework to handle non-branch-and-bound based solvers and could run on more than a million CPU cores

***Yuji Shinano, Thorsten Koch**, Zuse Institute Berlin*

In Short

- Develop the generalized UG, a single unified framework to parallelize both branch-and-bound and non-branch-and-bound based solvers.
- The generalized UG can customize its dynamic load balancing algorithm with respect to the solver used.
- The ability of run parallel solvers instantiated by the generalized UG framework on more than a million CPU cores.

Optimization solvers have been studied and developed in the mathematical optimization research community and parallel computing one. For NP-hard problem solvers, the solvers developed by mathematical optimization community are superior in solving hard instances to those developed by parallel computing community. Solvers developed by the parallel computing community can only expect at most linear speed-up from parallelization, which is not enough to solve challenging NP-hard problems. Mathematics is instrumental in advancing our ability to solve this class of problems. A fundamental question is if the combination of sophisticated mathematical techniques and a huge amount of cores, like over a million CPU cores, could help to solve hard optimisation problems. In order to answer this question we must develop scalable parallel solvers with mathematically supercharged algorithms.

There are a few successful results of previously unsolved instances being solved by using large scale computing environments. ParaSCIP has solved numerous open instances of mixed integer programming problems (MIP)[1] and UG [SCIP-Jack, MPI] has solved many open instances of Steiner tree problems (STP)[3]. These successful results have been made possible by the parallelization software framework Ubiquity Generator (UG)[5], which parallelizes existing state-of-the-art branch-and-bound based solvers. Basic concept of the UG framework is to exploit powerful performance of state-of-the-art "base solvers", such as SCIP, Xpress, etc., without the need for base solver parallelization. The base solvers and communication libraries are then abstracted within UG.

Recently, an experimental implementation for finding shortest lattice vectors based on UG has been developed. Lattice-based cryptography has received attention as a next-generation encryption technique, because it is believed to be secure against attacks by classical and quantum computers. Its essential security depends on the hardness of solving shortest vector problems (SVP). The parallelized algorithms for solving SVP based on UG is *not* branch-and-bound based. In spite of this fact, the experimental implementation successfully parallelized the algorithm and improved record for several dimensions of the problems in SVP challenges by using over 100,000 cores [4].

This project's ultimate goal is to study efficient ways to use over a million CPU cores to solve hard optimization problems. In this project, we will deal with a wide range of NP-hard problems, which are solved using branch-and-bound, including mixed integer linear and non-linear programming problems, and classical combinatorial optimization problems, such as the traveling salesman problem and the quadratic assignment problem. Further, a major innovation of this proposal is the development of a software framework to parallelize non-branch-and-bound based solvers for problems such as the SVP.

The software framework UG is known as one of the most successful software tools in order to parallelize solvers that are mathematically supercharged branch-and-bound based algorithms. Its parallelization paradigm is *supervisor-worker*, in which a single supervisor controller performs load balancing by sending and receiving small numbers of messages on demand to workers. In UG, the worker is an existing solver and well defined message passing protocols for branch-and-bound algorithms between the solvers are defined. UG was first applied to parallelize the MIP solver SCIP. More recently, the framework has been extended to parallelize the commercial MIP solver Xpress[2], which is a multi-threaded solver.

From a software engineering point of view, UG's approach has the inherent benefit of high functional cohesion. Additionally, from a parallelization point of view it has inherent benefits about locality, since it uses an existing solver. For mathematicians, this approach also has benefits, since algorithm designers need not be concerned about parallelization and can focus on more mathematical algorithmic improvements. A key feature of UG is a well defined message passing protocol of load balancing among

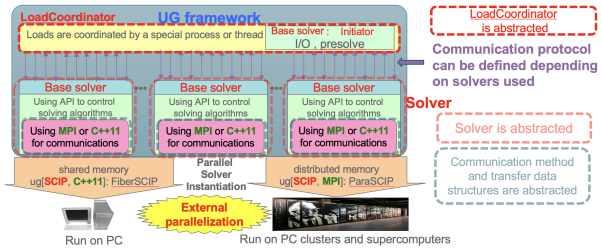

Figure 1: *Processes structure and abstractions of code*

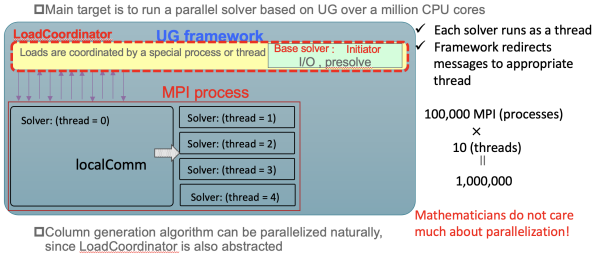

Figure 2: *Redirect messages by local communicator for threaded solver version of UG*

branch-and-bound based solvers. In order to become more efficient, UG needs to have a feature so that users can define the message passing protocol depending on the underlying solver. Addressing this requirement is a major task of this proposed project. When used for non branch-and-bound based solvers, the message passing protocol needs to be developed from scratch.

In the first phase of this project we will re-design UG so it enables users to design message passing protocols depending on the used solver. This flexibility will allow users to design parallel solvers sharing user defined information by message passing that is solver specific. We call the re-designed UG *generalized UG*. The second phase of this project is dedicated to re-implementing solvers currently parallelized using UG to use the generalized UG framework. In the third phase, we will improve the performance on each solver by adding new message passing protocols depending on which solver has been used and what kind of information should be shared.

Two types of processes exist when running the parallelized solver by UG on distributed memory environment. First, there is a single LOADCOORDINATOR, which makes all decisions concerning the dynamic load balancing and distributes subproblems (or some task) of a target problem to be solved. Second, all other processes are SOLVERs that solve the distributed subproblem (or do some task) by regarding it as root node in case of branch-and-bound. Figure 1 shows the processes structures. As we aforementioned, base solvers and communication libraries abstracted in UG. In the generalized UG, the LOADCOORDINATOR is also abstracted so that dynamic load balancing mechanism can be customized depending on base solver used.

Even if the base solver used is a single threaded solver, the generalized UG can use it as a thread within an MPI process. In this case, each solver runs as a thread within an MPI process, and thread id zero solver receives messages from LOADCOORDINATOR and it redirects the messages to the target threads as shown in Figure 2. Current UG can handle up to 100,000 MPI processes [4]. Therefore, the threaded solver version of massively parallel solvers can be potentially handle over a million CPU cores.

WWW

`https://ug.zib.de/`

More Information

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

[2] Y. Shinano, T. Berthold, and S. Heinz, *ParaXpress: an experimental extension of the FICO Xpress-Optimizer to solve hard MIPs on supercomputers*, Optimization Methods and Software, 33 (2018), pp. 530–539.

[3] Y. Shinano, D. Rehfeldt, and T. Koch, *Building optimal steiner trees on supercomputers by using up to 43,000 cores*, in Integration of Constraint Programming, Artificial Intelligence, and Operations Research, L.-M. Rousseau and K. Stergiou, eds., Cham, 2019, Springer International Publishing, pp. 529–539.

[4] N. Tateiwa, Y. Shinano, S. Nakamura, A. Yoshida, M. Yasuda, S. Kaji, and K. Fujisawa, *Massive parallelization for finding shortest lattice vectors based on ubiquity generator framework*, in 2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Los Alamitos, CA, USA, nov 2020, IEEE Computer Society, pp. 834–848.

[5] *UG: Ubiquity Generator framework*. `http://ug.zib.de/`.

Funding

BMBF Research Campus MODAL