ParaConcorde: A Massively Parallel Solver for TSPs

Massively Parallel Solution of Challenging Large Scale Traveling Salesman Problems

***Y.Shinano**, Zuse Institute Berlin*

In Short

- Development of a massively parallel traveling salseman problem(TSP) solver.
- Solve previously unsolved challenging TSP instances.

Given an undirected graph $G = (V, E)$ with non-negative integer edge cost $c(u, v)$ for each edge $u, v \in E$, the *Traveling Salesman Problem* (TSP) is to find a hamiltonian cycle of G with minimum cost. The TSP is a classic $\mathcal{NP}$-hard problem [5] and one of the most studied problems in combinatorial optimization. This project aims to develop a massively parallel solver for TSP based on the Ubiquity Generator (UG) framework [13], which makes Concorde running on supercomputers to solve extremely hard instances.

The well-known benchmark for the TSP is the TSPLIB [7,12], which was first published in 1991 and all of the instances were solved to optimaliy in 2007 due to enormous algorithmic progress. We have developed the Ubiquity Generator (UG) Framework[13], which is a software framework to parallelize branch-and-bound based solvers. There are massively parallel branch-and-bound system, such as PEBBL [4], ParSSSE [11] and ALPS [14]. However, only a handful of publications [2,8,10] address effective parallelization of state-of-the-art solvers capable of running on modern supercomputers. The essential design concept of UG is to parallelize the state-of-the-art solvers on supercomputers. ParaSCIP [9] is the role model for a massively parallelized Mixed Integer Programming (MIP) solver. It is well-known in the discrete optimization community that the most powerful TSP solver is Concorde [3], in which the most sophisticated Linear Programming (LP) based branch-and-cut algorithm is implemented specialized for TSP. Concorde includes cutting plane algorithms, branching rules, etc. specialized for TSP and also data structures are specialized for TSP. There is a book “The Traveling Salesman Problem: A Computational Study”(606 pages) [1] which describes them. The Concorde source code is publicly available. Its version date is Dec 19, 2003, but still it is the strongest TSP solver to obtain optimal solutions. William J. Cook has kept improving the code, and we can use the latest version of Concorde in this project. In this project, we clarify how much we could accelerate the solution process by using supercomputers and provide optimal solutions for challenging, previously unsolved TSP instances.

Although the algorithmic components are similar to a MIP solver, Concorde implementation is quite different from state-of-the-art MIP solvers. For example, the variables part of an optimal solution are generated on the fly and each branch-and-bound node is saved as a file that represents an LP-relaxation of the node, and each branch-and-bound node computation time is very long to obtain good lower bound and to select a good branching variable etc. and their execution can be parallelized. From a parallelization point of view, Concorde itself is a parallel solver. In order to run parallelized Concorde by UG on supercomputers, it should work with original parallelization code inside of Concorde.

We developed an experimental parallel TSP solver based on UG to check if it can be realized or not. Throughout this document, we call it ParaConcorde. Unfortunately, the UG framework itself had to be modified, but it was realized by adding several callback functions to Concorde. We conducted computational experiments to solve the Mona Lisa TSP challenge instance (100,000-city instance) [6] on the ISM (Institute of Statistical Mathematics) supercomputer HPE SGI 8600, which is a liquid cooled, tray-based, high-density clustered computer system. The ISM supercomputer has 384 computing nodes and each node has two Intel Xeon Gold 6154 3.0GHz CPUs (36 cores) and 384GB of memory. On the ISM supercomputer, we can submit a job with a time limit of one week. For racing, we used 1721 racing solvers, that is, 1721 random seeds are tested to find the winner solver and 5183 solvers used for the following jobs.

Figure 1 shows how the gap between upper bound and lower bound changed during computation. During racing, each color line shows each solver’s gap and the red bold line shows the winner’s gap. Different from parallel MIP solvers based on UG, ParaConcorde could continue racing ramp-up from the checkpoints, since Concorde itself has mechanism to check-pointing and restarting and ParaConcorde extended the feature. The racing termination criterion is that the winner has more than 20 open nodes. The racing works well to select the most promising search tree, but it takes quite long to terminate the racing even when the termination criteria was that only 20 open nodes were generated. However, once the winner was selected, the single tree search parallelization makes a big search tree relatively fast compared to the racing phase.

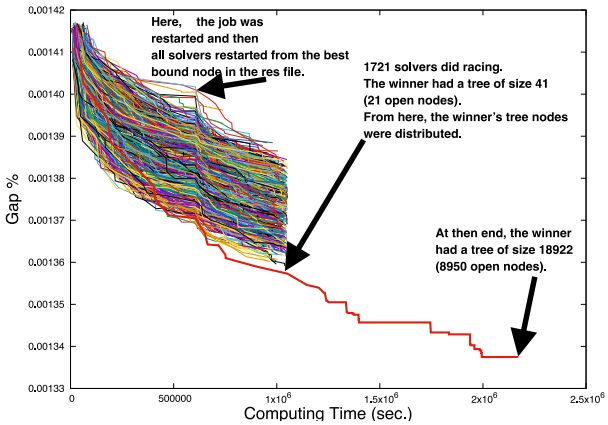

Figure 1: *Solving the Mona Lisa TSP (100,000-city instance) by using ParaConcorde with racing ramp-up*

Concorde can offer internal parallelization features, but these are based on UNIX socket based communication for coordination, and data exchange between workers via the file system. We have ported these to directly use MPI send/recv operations, to enable full use of the High-Speed Networks (HSN). This native MPI version of Concorde will allow us to integrate it with UG in an even better way: The most recent version of UG is able to coordinate multiple distributed solvers, each in its own MPI-GROUP. We plan to extend this capability to allow Concorde to make use of a dynamic number of additional ranks for internal parallelization, and UG can accommodate that via a dynamic MPI Group modification mechanism to re-balance the number of ranks per group of all the Concorde instance running at a given time. Since Concorde contains a number of tunable parameters, and also employs randomization for tie breaking, running multiple Concorde solvers with differing computational resources assigned to them appears very appealing. It can also be seen as building an infrastructure to simplify such dynamic MPI group rebalancing for other applications, a critically missing toolset to increase the efficiency of large-scale MPI applications where computational load shifts during the lifetime of a program.

We have had a series of HLRN projects related to the UG framework. The latest one is "Generalization of Ubiquity Generator Framework to handle non-branch-and-bound based solvers and could run on more than a million CPU cores'(bem00052)", which aims to build a unified and generalized software framework to develop a massively parallel solver that exploits state-of-the-art search-based algorithms to solve optimization problems. In the project bem00052, we redesign and refactor the UG Framework to enable easier use on a broader range of optimization applications on more cores. As a consequence, in this project, we can develop a much more customized ParaConcorde, which extends Concorde's check-pointing and restarting mechanism and combines external parallelization of UG and internal parallelization of Concorde.

WWW

https://www.zib.de/members/shinano,
https://www.math.uwaterloo.ca/~bico/,
https://https://www.math.uwaterloo.ca/tsp/concorde.html,
https://ug.zib.de/

More Information

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*, Princeton University Press, USA, 2007.

[2] M. R. Bussieck, M. C. Ferris, and A. Meeraus, *Grid-enabled optimization with GAMS*, IJoC, 21 (2009), pp. 349–362.

[3] *Concorde TSP Solver*. http://www.math.uwaterloo.ca/tsp/concorde/index.html.

[4] J. Eckstein, W. E. Hart, and C. A. Phillips, *Pebbl: an object-oriented framework for scalable parallel branch and bound*, Mathematical Programming Computation, 7 (2015), pp. 429–469.

[5] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. Miller and J. Thatcher, eds., Plenum Press, 1972, pp. 85–103.

[6] *Mona Lisa TSP Challenge*. http://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html.

[7] G. Reinelt, *Tsplib—a traveling salesman problem library*, ORSA Journal on Computing, 3 (1991), pp. 376–384.

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

[9] 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 Proc. of 30th IEEE International Parallel & Distributed Processing Symposium, 2016.

[10] Y. Shinano, T. Achterberg, and T. Fujie, *A dynamic load balancing mechanism for new ParaLEX*, in In: Proceedings of ICPADS 2008, 2008, pp. 455–462.

[11] Y. Sun, G. Zheng, P. Jetley, and L. V. Kalé, *ParSSSE: An adaptive parallel state space search engine*, Parallel Processing Letters, 21 (2011), pp. 319–338.

[12] *TSPLIB*. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

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

[14] Y. Xu, T. K. Ralphs, L. Ladányi, and M. Saltzman, *Alps version 1.5.2*, 2015.

Project Partners

Prof. W. J. Cook, University of Waterloo,
Dr. U.U. Haus, HPE HPC/AI EMEA Research Lab

Funding

BMBF Research Campus MODAL