An Extended GLB Library for Optimization Problems

Shota Izumi, Daisuke Ishii
University of Fukui
{ji170011,dsksh}@u-fukui.ac.jp

Kazuki Yoshizoe
AIP, RIKEN
kazuki.yoshizoe@riken.jp

ABSTRACT

We propose a library for parallelizing generic search-based optimization processes. We also propose a benchmark optimization problem for our library. Two experimental results are reported about the promising performance of the library.

ACM Reference Format:
Shota Izumi, Daisuke Ishii and Kazuki Yoshizoe. 2018. An Extended GLB Library for Optimization Problems. In *Proceedings of International Conference on High Performance Computing in Asia Pacific Region (HPC Asia).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Optimization problems consist of sets of variables and constraints and an objective function; we aim to obtain an assignment to the variables that minimizes the value of the objective function, at the same time satisfying the constraints. Various practical problems are translated to optimization problems and solved. To solve large and complex instances by a search technique, parallelizing optimizers to run efficiently on PC clusters is a promising approach [1, 3]. However, most existing methods more or less assume a centralized structure and their scalability on massive PC clusters is limited.

This research aims to provide a decentralized parallelization scheme for generic optimization problems that are solved with search. For this purpose, we propose a library for parallelizing various search-based optimization processes. Our library extends the X10 GLB library (Section 2) for parallel and distributed search computation (Section 3), which is processed by a group of homogeneous workers those not assuming a centralized topology. To analyze the characteristics of the library, we design an optimization benchmark (Section 4). We have experimented to solve several instances of the benchmark using the extended GLB library on up to 504 cores of a supercomputer; some promising results are reported in Section 5.

2 X10 GLB LIBRARY

X10 (http://x10-lang.org/) is a productive programming language for HPC. *GLB* (Global Load Balancing)[4] is a standard library of X10 that provides load balancing and termination mechanisms for cooperative parallel *workers* whose workloads are not predictable. Each worker typically runs on a CPU core and homogeneously processes a divided portion of the whole workload. Load balancing between workers is done in two steps: first, a starving worker sends a request to other workers either randomly chosen or connected via a network of workers (so-called lifeline); second, loaded workers will respond by sending a portion of its workload. Termination is also detected by a communication via the lifeline.

3 EXTENDED GLB FOR OPTIMIZATION PROBLEMS

GLB is appropriate to a search procedure for solving constraint satisfaction problems [2]. In an optimization problem, an objective function is given and evaluated while enumerating feasible solutions. In a parallel setting, it is essential to share a tentative optimal assignment among the parallel tasks; with this information, a task may abandon its workload when nonexistence of an optimal value is obvious.

In this work, we extend GLB to enable sharing a tentative information among workers. When a worker finds a local optimum, it notifies the value to other workers each time a certain amount of sequential computation is done. To settle a trade-off between the distribution speed and the communication cost, the notification is sent to n randomly selected workers.

4 OPTIMIZATION BENCHMARK

To evaluate the performance of the extended GLB library, we design a benchmark problem. The problem is intended to be easy to configure the size and the degree of parallelism of its search space; the possibility of search space pruning is also designed to be configurable.

The designed problem is based on a perfect b-ary tree whose nodes are weighted with integers; the objective is to find a path with a smallest sum weight. An instance of the problem is represented with three integers b, d, and h, which represent the degree of branching, the depth of the tree, and the seed for random weight generation, respectively. Tree nodes are randomly weighted with integers 0–255.

As a generalization of the problem, we can consider a problem to rank the top k feasible solutions with better weights. The higher the k, the more chances to have a tentative optimum, i.e., an updated ranking, and a parallel solving requires an efficient distribution process.

HPC Asia, 2018, Tokyo, Japan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

4.1 Implementation

We have implemented a benchmark solver with the extended X10 GLB that searches along with a given tree in a depth-first fashion while generating the weight each time a new node is traversed. Subtrees of the search tree are separated and distributed among GLB workers. When a worker reaches a leaf and the sum weight is optimum, this value is notified to other workers using our extension to GLB.

The `TaskQueue` interface is implemented as follows:

- The `process()` method of GLB's `TaskQueue` interface implements a unit of the solving process. `TaskQueue` maintains a stack of the tree nodes to be searched. First, `process()` pops a node, expands a child of the node, and computes its weight. Next, the current search may terminate as a result of comparison between the sum weight and the tentative optimum. Finally, `process()` updates the optimum if the node is a leaf, or otherwise pushes the node in the stack.
- The `split()` and `merge()` methods of the `TaskQueue` implement the process of load balancing. In response to a starving worker, a sender provides a half of the content of the stack of tree nodes, which is returned by `split()`; then, `merge()` of a receiver appends them to its stack.

5 EXPERIMENTAL RESULTS

We ran two experiments to solve the benchmark problem with the GLB-based solver on a supercomputer. We used up to 14 nodes where each node has two Xeon 2.1GHz processors (18×2 cores, max. 504 cores in total) and 128GB RAM. For X10 compilation, we used the C++ native compiler version 2.5.4 with MPI back-end.

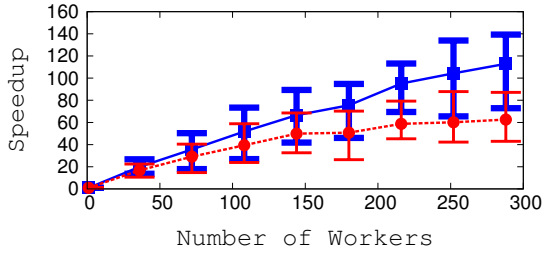
5.1 First Experiment

We evaluated the efficiency of the parallel solver using up to 288 cores when various instances were given. Instances for every combination of $(b, d) \in \{(6, 30), (8, 26)\}$ and $h \in \{0, 3, 15, 35\}$ were prepared and the topmost ($k = 1$) or top ten ($k = 10$) feasible solutions were computed. In this experiment, tentative optima were broadcasted.

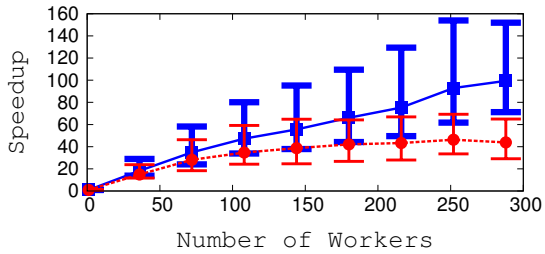
Measured speedups for the instances are illustrated in Figures 1(a) and 1(b). Each line represents the average for the instances with different hashes h. The averages and ranges of sequential execution timings are shown in Table 1. Despite broadcasting the optima, we had monotonic speedups for all the runs. We observed that the range of the speedups for each instance group was reasonably small as we had expected in the benchmark design. The benchmark will help further development of the GLB library e.g. by controlling the number of feasible solutions by changing the values of b and k, while restricting the effect of search space pruning.

5.2 Second Experiment

We checked that the distribution of tentative optima with random sending improves the performance when the communication cost is a bottleneck. We solved the instances in the first experiment using up to 504 cores. Two distribution methods, which send an optimum to either all other workers (i.e. broadcasting) or randomly selected 1–2 workers (i.e. random sending), were used. The speedups are shown in Figures 2(a) and 2(b). We confirmed that random sending improves the distribution efficiency for both instances.



(a) Speedups for the instances where $b = 6$, $d = 30$, and $k = 1$ (blue plain line) or $k = 10$ (red dotted line).



(b) Speedups for the instances where $b = 8$, $d = 26$, and $k = 1$ (blue plain line) or $k = 10$ (red dotted line).

Figure 1: First experimental results.

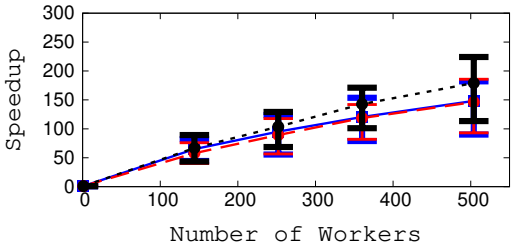
Table 1: Execution timings.

(b, d)	k	ave. time	time range
$(6, 30)$	1	840s	496–1460s
	10	1185s	702–1970s
$(8, 26)$	1	440s	353–511s
	10	585s	502–643s

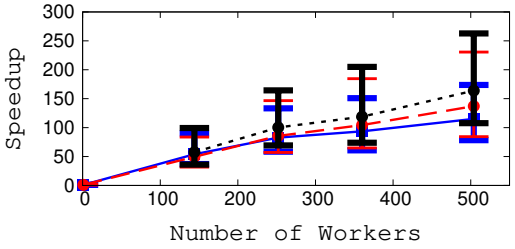
6 CONCLUSION

This paper has reported preliminary but promising results for parallelizing search-based optimization processes. Future research will explore improvements of the distribution process, e.g., based on a hierarchical network of workers.

Acknowledgments. This work was partially funded by JSPS (KAKENHI 15K15968). We used the supercomputer of AC-CMS, Kyoto University.



(a) Speedups for the instances where $b = 6, d = 30$, and $k = 10$.



(b) Speedups for the instances where $b = 8, d = 26$, and $k = 10$.

Figure 2: Second experimental results. Tentative optima were either broadcasted (blue solid lines), sent to a randomly selected worker (red dashed lines), or sent to two randomly selected workers (black dotted lines).

REFERENCES

[1] D. Bergman et al. 2014. Parallel Combinatorial Optimization with Decision Diagrams. In *CPAIOR (*LNCS*8451)*. 351–367.
[2] D. Ishii, K. Yoshizoe, and T. Suzumura. 2015. Scalable Parallel Numerical Constraint Solver Using Global Load Balancing. In *ACM SIGPLAN Workshop on X10*. 33–38.
[3] J. Jaffar et al. 2004. Scalable distributed depth-first search with greedy work stealing. In *ICTAI*. 98–103.
[4] W. Zhang et al. 2014. GLB : Lifeline-based Global Load Balancing Library in X10. In *PPAA*. 31–40.