Implementing the Tascell Task-Parallel Language Tascell Using Multithreaded MPI

Daiki Kojima
Graduate School of Informatics, Kyoto University
Kyoto, Japan
d-kojima@sys.i.kyoto-u.ac.jp

Tasuku Hraishi
Hiroshi Nakashima
Academic Center for Computing and Media Studies, Kyoto University
Kyoto, Japan

Masahiro Yasugi
Department of Artificial Intelligence, Kyushu Institute of Technology
Iizuka, Fukuoka, Japan

The Tascell language is a task-parallel language that achieves high performance and distributed memory environment support using the backtracking-based load balancing approach [1]. At first, inter-node communications in Tascell are implemented using TCP/IP. In this implementation, Tascell server processes are employed in addition to computing processes. Each computing process that contains one or more worker threads is TCP/IP connected to a Tascell server, which relays inter-node messages.

Muraoka et al. developed an MPI-based implementation of Tascell [2] to support environments where TCP/IP is not available for inter-node communications and solve bottlenecks at Tascell servers. In this implementation, Tascell servers are not employed and inter-node mesasges are transferred directly among computing processes. Each computing process employs a messaging thread and a send request queue. In order to send an inter-node message, a worker thread asks the messaging thread to send the message by adding it to the request queue. The messaging thread sends the queued message using `MPI_Isend`. This thread also handles messages from outside the node. It checks the existence of an incoming message using `MPI_Iprobe` and receives it using `MPI_recv`. This implementation has an advantage that it only requires the two-sided communication paradigm and the `MPI_THREAD_FUNNELED` support level for MPI implementations. However, there is the problem that the messaging thread uses busy-waiting for incoming or outgoing messages, which is unavoidable without requiring the `MPI_THREAD_MULTIPLE` support level, in which multiple threads can make MPI calls in parallel.

Recently, more supercomputers are providing MPI implementations with the `MPI_THREAD_MULTIPLE` support level, which motivated us to develop a busy-waiting free implementation of Tascell by requiring this support. In our implementation, a worker thread itself sends outgoing messages using `MPI_send`. For incoming messages, we employ a incoming message queue and two support threads, a receiving thread and a handling thread, in each process. The receiving threads waits for and receives incoming messages using `MPI_recv` and adds them to the message queue. The handling thread takes a message from the queue and handles it. Note that we cannot let the receiving thread also handle the received message because some messages requires sending other outgoing messages for handling it. When it occurs, the the receiving thread cannot receive any incoming message until the sending messages is completed, which causes a deadlock[1].

[1]The busy-waiting free implementation requiring the `MPI_THREAD_MULTPLE` support level employed as a competitor in the performance evaluation in [2], whose implementation details are not presented, has this problem.

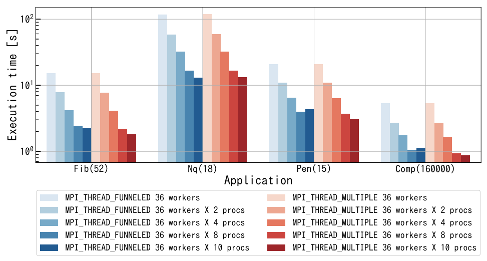

Figure 1: The result of the performance measurements of the MPI-based implementations of Tascell.

We evaluated our implementation using up to 10 nodes of the Laurel 2 supercomputer of ACCMS, Kyoto University. Each node has two 18-core Xeon Broadwell E5-2695 v4 processors. We used Intel C compiler 17.0.6.256 with the `-O3` option, and Intel MPI 2017.4. One MPI process having 36 worker threads was created on each node. The result of the performance measurements of the original (`MPI_THREAD_FUNNELED`) and our (`MPI_THREAD_MULTIPLE`) implementations are showed in Fig. 1. Our implementation achieved higher performance in almost all the benchmark programs. In particular, we achieved 40% speedup compared to the original implementation in the executions using 10 nodes for `Pen`.

Furthermore, we are developing another MPI-based implementation of Tascell, which uses the `MPI_THREAD_MULTPLE` support level and one-sided communications. In our current implementation, a worker packs an outgoing message into a buffer before sending it. Such redundant copying operations may be omitted using one-sided communications. Since a message often contains large data such as a vector as an input of a task, it is expected that we can improve the performance by this optimization.

ACKNOWLEDGMENTS

We thank Daisuke Muraoka, who developed the original MPI-based implementation of Tascell and gave us lots of technical knowledge about it. This work was supported in part by JSPS KAKENHI Grant Numbers JP17K00099 and JP19H04087.

REFERENCES

[1] Tasuku Hiraishi, Masahiro Yasugi, Seiji Umatani, and Taiichi Yuasa. 2009. Backtracking-based Load Balancing. In *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2009)*. 55–64.

[2] Daisuke Muraoka, Masahiro Yasugi, Tasuku Hiraishi, and Seiji Umatani. 2016. Evaluation of an MPI-Based Implementation of the Tascell Task-Parallel Language on Massively Parallel Systems. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*. 161–170.