Scalability of OpenMP

Project Manager
Dr. Tim Jammer

Principal Investigator
Prof. Dr. Christian Bischof

Project Term
2020 - 2021

Clusters
Lichtenberg Cluster Darmstadt

Institute
Institute of Scientific Computing

University
Technische Universität Darmstadt

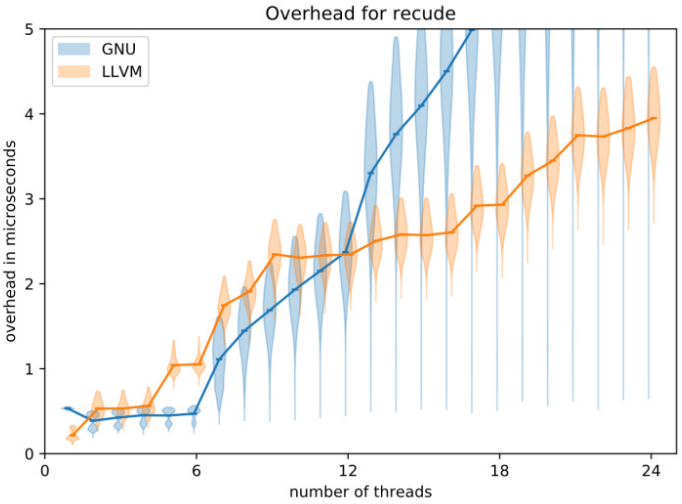

Figure 1: Comparison of the Overhead of an OpenMP reduction operation for the GNU and LLVM implementation. The Y axis denotes the Overhead in microseconds, while the X axis shows the number of Threads.

Introduction

This project is concerned with promoting a more efficient usage of the HPC Systems. The two major technologies used to write parallel scientific applications for HPC systems are MPI and OpenMP. In this project, we want to see if tools like the compiler can be used, in order to help the programmer of an application to write it as efficient as possible. We especially focus on MPI, as we see that new MPI features are not yet widely adopted, as Most Applications use only version 1 of the MPI standard, which has been around since 1995.

Methods

Four main studies are part of this project:

First, we analyzed the scalability of different OpenMP implementations, by measuring the time needed to complete OpenMP constructs. Our second study evaluated different tools, that were developed to aid developers write correct MPI programs, e.g., Intel Trace Analyzer and Collector (ITAC), MUST, Par coach and MPI-Checker. To enable a structured comparison and improve the cover- age and reliability of available MPI correctness tools, we propose MPI-CorrBench as a common test harness. MPI-CorrBench enables a structured comparison of the different tools available w.r.t. various types of errors. The MPI-CorrBench benchmark suite, as well as the visualization of the test results can be found on our github: https://github.com/tudasc/MPI-Corrbench. In order to help with the adoption of the new version 4.0 of the MPI Standard, we developed two compiler based tools. One feature, that was

added into the new standard, is the specification of application info assertions. This allows an application to signal MPI that it will adhere to certain assertions. This in turn will allow the MPI implementation to choose more efficient communication routines. For Example, if an application asserts that it does not need a particular ordering of messages, the MPI implementation can skip the part where it ensures a consistent message ordering. We propose to analyze a program at compile time by the compiler, in order to find out if some of these assertions hold for the application. An additional new MPI feature is the added partitioned communication, which is designed for a multi-threaded usage, e.g. together with OpenMP. Therefore, we developed a compiler based tool, that automatically transforms MPI Operations into a, equivalent partitioned operations if this is applicable in the given application. This way, the user does not need to learn the details about those new features while still being able to get a performance benefit from them.

Results

Our Analysis of different OpenMP implementations shows, that there is still more potential for efficiency improvement, which will become more and more important, when HPC Systems grow larger and larger. In Figure 1 one can see the comparison of the overhead incurred by LLVM and GNU's OpenMP implementation for a reduction operation. We see that with lower thread count, GNU's implementation performs better, while LLVM's implementation works best with higher thread count. The reason here is, that the over- head of LLVM's implementation grows logarithmically with the number of threads. But managing the tree alike data structures behind this is more complex, which leads to a higher overhead with lower thread counts. Our analysis of different MPI correctness tools shows that they are quite sophisticated for the commonly used MPI features. But there is room fore improvement, as the tools still lack support for more modern MPI features, such as one-sided communication. As one-sided communication is more complex to implement, the comparative lack of tool support for it, is another factor why these modern MPI features are not yet widely adopted. We address this gap with our compiler-based tool support. Our initial testing revealed that it is possible to automatically detect MPI assertions in many occasions, while the correctness of the program is always kept. This compiler based approach is also promising for the partitioned MPI operations, as we managed to automatically use the partitioned operations, with only minimal overhead, which will still allow for an exploitation of a performance benefit. Our tools can be found at https://github.com/tudasc.

Publications

Lehr, J.-P.; Jammer, T.; Bischof, C. (2021): MPI-CorrBench: Towards an MPI Correctness Benchmark Suite. In: HPDC '21: Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing, S. 69-80, ACM, 30th International Symposium on High-Performance Parallel and Distributed Computing, virtual Conference, 21.-25.06.2021, ISBN 978-1-4503-8217-5.
https://doi.org/10.1145/3431379.3460652

Jammer, T.; Iwainsky, C.; Bischof, C. (2020): Automatic Detection of MPI Assertions. In: High Performance Computing, S. 34-42, Springer, 35th Inter- national ISC High Performance 2020 Conference, Frankfurt, Germany, 21.- 25.06.2020, ISSN 0302-9743, ISBN 978-3-030-59850-1.
https://doi.org/10.1007/978-3-030-59851-8_3

Jammer, T.; Iwainsky, C.; Bischof, C. (2020): A Comparison of the Scalability of OpenMP Implementations. In: European Conference on Parallel Processing, S. 83-97, Springer, 26th International Conference on Parallel and Distributed Computing (Euro-Par 2020), Virtual Conference, 24.-28.08., ISBN 978-3-030-57674-5.
https://doi.org/10.1007/978-3-030-57675-2_6

Last Update: 2023-03-16 01:28