Binary Rewriting at Runtime for Efficient Dynamic Domain Map Implementations

Josef Weidendorfer, Jens Breitbart
Department of Informatics
Technical University of Munich
Munich, Germany
EMail: {Josef.Weidendorfer, j.breitbart}@tum.de

I. Overview

HPC programs are written with an obvious focus on performance. Among others, they try to rely on static work partitioning whenever possible, as this allows for better compiler optimization and has inherently less communication. However, with the need for more and more sophisticated energy saving strategies, we expect an upcoming growth in dynamics at hardware level, resulting in more dynamic changes to clock frequency and voltage. These changes will make dynamic (re)partitioning a requirement for all large scale simulations, even those which originally could rely on a static scheme. Dynamic workload partitioning (for data parallel workloads) is typically represented by abstracting the actual data distribution and have each node query the distribution to identify which part of the workload it has to compute. As repartitioning data itself can be costly, the exact distribution stays constant for some time. In this talk we will give a (very) brief introduction into DBrew[1] (Dynamic Binary REWriting) [1], a library that allows for binary rewriting of functions at runtime that we use to optimize code for given data distributions. We furthermore present preliminary results of applying it at small Chapel programs and hope to start a discussion on how such a feature could be integrated into the Chapel software stack.

II. DBREW

DBREW is still in an early stage of its development and the source is not yet available to the public, but a release as open source is planned before IPDPS. DBREWs target group is not application developers, but rather tuning experts that for example use it as part of a runtime system. The most important feature of DBREW is the specialization of functions for a given set of parameters. As an input, `dbrew_rewrite()` expects a pointer to the function that should be specialized as well as the parameters for which the function should be specialized. The result is a function pointer that can be called as drop-in replacement of the original function, using exactly the same signature. At runtime, DBREW parses the binary machine code[2] of that function and uses the information of the constant values to produce a specialized version. This version is typically faster than the original code, but the exact benefit varies between functions. DBREW also applies some simple optimizations like function inlining. However, we expect that the original code was already well optimized by a compiler.

III. Chapel integration

So far, we have applied DBREW to simple Chapel programs that mostly consist of arrays with Domain Maps and simple loops using the arrays. We added DBREW by manually changing the C code generated by the Chapel Compiler and used it to specialize the function `dsi_access2()` for a given data distribution. The generated code consist of 54% fewer instructions. The manual integration can obviously only serve as a small testbed and we would like to gather feedback on how to integrate DBREW into the Chapel compiler/runtime to use it for larger tests.

References

[1] J. Weidendorfer and J. Breitbart, "The case for binary rewriting at runtime for efficient implementation of high-level programming models in HPC," in *Proc. 2016 IEEE Int. Symp. Parallel and Distributed Processing Workshops and Phd Forum*, May 2016.

[1] https://github.com/lrr-tum/dbrewhttps://github.com/lrr-tum/dbrew

[2] Currently, x86-64 is supported.