Lightweight threading support in LLVM

Kavon Farvardin
University of Chicago

John Reppy
University of Chicago

A lightweight and efficient mechanism for user-space threads is an important building block for parallel and concurrent language implementations. It is particularly advantageous when the compiler has an internal representation for the threading primitives that can enable optimization a scheduling code. In our Parallel ML (PML) compiler (part of the Manticore project), we have taken an approach based on supporting heap-allocated first-class continuations in the intermediate language. This approach has allowed us to build efficient work-stealing, user-level threading, and other mechanisms for parallel and concurrent programming [4, 1, 7].

We used the MLRisc [6, 5] code generation framework as the backend of our PML compiler. MLRisc gave us the flexibility to customize our calling and runtime conventions to support efficient first-class continuations, but it is not being actively developed anymore. As an alternative, we have recently been exploring how to achieve the same lightweight support for threading using the LLVM code generation infrastructure.

While LLVM has been used to support a wide range of languages, it has very strong biases toward traditional C-like runtime conventions. To support first-class continuations in LLVM, we had to develop a new calling convention for LLVM as well as a mechanism for reifying implicit continuations inside LLVM functions [3, 2] Building on this work, we implemented four different runtime models for threading in our compiler on top of LLVM. These implementations provide an apples-to-apples comparison of the different approaches.

In the full paper, we will describe how LLVM can be made to support non-traditional runtime conventions that, in turn, can be used to implement lightweight efficient threading mechanisms.

References

[1] Lars Bergstrom, Matthew Fluet, Mike Rainey, John Reppy, and Adam Shaw. Lazy tree splitting. *JFP*, 22(4-5):382–438, September 2012.

[2] Kavon Farvardin. Weighing continuations for concurrency. Master's thesis, Department of Computer Science, University of Chicago, Chicago IL, 2017.

[3] Kavon Farvardin and John Reppy. Compiling with continuations and LLVM. In *ML '16*, September 2016.

[4] Matthew Fluet, Mike Rainey, John Reppy, and Adam Shaw. Implicitly-threaded parallelism in Manticore. *JFP*, 20(5–6):537–576, 2011.

[5] Lal George and Andrew Appel. Iterated register coalescing. *ACM TOPLAS*, 18(3):300–324, May 1996.

[6] Lal George, Florent Guillame, and John Reppy. A portable and optimizing back end for the SML/NJ compiler. In *CC '94*, number 786 in LNCS, pages 83–97, New York, NY, April 1994. Springer-Verlag.

[7] John Reppy, Claudio Russo, and Yingqi Xiao. Parallel Concurrent ML. In *ICFP '09*, pages 257–268, New York, NY, August–September 2009. ACM.