Experiments in Compiling R Code

Vincent Buffalo, University of California at Davis

Duncan Temple Lang, University of California at Davis

Table of Contents

We describe some experiments and an approach to compiling R code to native machine instructions. The primary goal is to make R code execute significantly faster, e.g. by a factor of 100 or 500 or more. An important part of our approach is exploiting information about the type of R objects. We rely on the caller providing this information at present.

Background

It is clear that using high-level programming languages such as R, MATLAB, Python and Perl lead to improved productivity and use of human time. However, these interpreted languages often lead to code that is considerably slower than compiled languages such as C++, C, FORTRAN, Java, etc. This trade-off between development and maintenance time and compute time is one that everyone would like to remove by making the high-level code run as fast as the low-level code. This is very challenging. However, we believe that there is a continuum between the use of high-level languages and low-level languages. Specifically, as we interactively develop code, we want the facilities of the high-level language - namely no need for type declarations, the disruptive compile-load-run cycle, etc. But after we have finalized our script or functions, we want the speed of low-level languages. We are prepared to endure compilation times if the code will run significantly faster. We further expect that many users of R would be willing to provide additional information about their "final" code to make it run ten or one hundred times faster. Motivated by this expectation, we have developed a prototype system for compiling R code to machine code that exploits the types of the different R objects to attain significant speed-up in the execution of the code.

This paper describes our initial experiments with compiling R code to machine instructions by leveraging a third party library - LLVM, the Low-Level Virtual Machine. LLVM is a C++ library that facilitates creating intermediate representations of code and then optimizing that code and generating machine-level instructions for many different target platforms. In this respect, it is a general compiler infrastructure that works on many platforms. It is being used in many different applications, including alternatives to the compilers gcc and g++. It is a robust, comprehensive compiler toolkit with a dynamic developer and user community. This is what makes it attractive as we can passively leverage the ongoing improvements to the code base without squandering resources within our own limited community. The optimization passes of the code this library presents, and continues to provide, makes this a promising direction to which we can connect compilation of R code.

A very important motivation to us in this approach is that we are building on work of a vibrant, reasonably large community, i.e. the LLVM developers and users, that will continue to maintain, adapt and improve

LLVM to future platforms and opportunities. We can develop a compiler ourselves within the R or statistics community, but we feel that there are too few of us to maintain and develop this successfully in the future. We applaud those who are doing this as it is a worthwhile endeavor.

We are making extensive use of information of the types of each variable. At present, the caller is required to specify this information for every variable. We can make this system a great deal smarter and infer the types of intermediate variables in many cases. We may even be able to infer the types of [DiamondRuby]

Future Directions

We plan to make our compiler handle more R idioms and language features.

We will provide interfaces to more of the LLVM C++ API from within R.

We plan to analyze R functions and scripts so that we can identify where copying R objects can be avoided, given that we are running in a special "optimized" mode. The CodeDepends package will help here in identifying where variables are no longer used, when they are redefined, etc.