Review, AC: Composable Asynchronous IO for Native Languages

Robert Hoff

October 20, 2011

1 Paper Summary

The paper describes a language variant, Asynchronous C (AC), that implements the C language with some new asynchronous constructs. The motivation for developing the language is a new type of multi-kernel operating system architecture based on message-passing. The new language constructs take advantage of message-passing by making asynchronous programming easier, because traditional asynchronous implementations use callback functions that lead to complex and unmaintainable code. The performance measurements show that asynchronous applications implemented with AC perform comparable to implementations using C with callbacks.

2 The Problem

Current trends in hardware design indicate growing number of cores, but scaling is inhibited by contemporary shared-memory OS implementations. Multi-kernel OS designs, instead, based on message-passing is suggested to be a more scalable model. The problem in this paper, are the implications this imposes on the language design.

3 The Solution

The language offers a standard C programming interface, in addition to providing asynchronous constructs. There are two slightly different implementation, either to rewrite the compiler, or use preprocessing macros based on existing functions. The compiler based approach is faster because of better stack management. Although AC is backward compatible with C, either method makes it necessary to rewrite applications to take advantage of message-passing performance potentials.

4 Evaluation

Performance is evaluated in two ways, against microbenchmarks (function calls) and on larger applications. In all tests it is shown that AC performs almost identically to C using asynchronous AI. The language supports a standard library interface, which provides compatibility with existing applications, but also becomes the responsibility of the implementation to take advantage of the new features.

5 My Opinion

It looks like the authors have achieved the new language implementations without affecting performance. So I would conclude the language has been enhanced, and that may be welcome. Although the paper is not conclusive about how useful these additions really are.

6 Possible Questions

1. Why are performances compared against C# and F#, are these comparisons relevant? It seems to me that no matter what the results are on these tests, it wouldn't say much about the success of adapting a different language.

2. I'm wondering if perhaps software engineering practises would be an alternative to introducing a new language. Are the new language constructs really useful, what are the actual difficulties of rewriting applications using callback functions?