AN ARCHITECTURE FOR COMBINATOR GRAPH REDUCTION

Philip J. Koopman Jr.

Functional programming offers a new way of writing programs and a new way of thinking about problem solving. Claimed advantages of functional programs are that they are amenable to automatic verification techniques, are easier to write, and are more reliable than other types of programs. Lazy functional languages provide powerful features such as implicit coroutining and support for infinite length data structures. Furthermore, it is possible that lazy functional languages will provide an easy path to exploit parallelism without programmer intervention.

A major problem with lazy functional languages is that they are notoriously slow, often as much as two orders of magnitude slower than “eager” functional languages and imperative languages. This speed problem has hindered exploration of the strengths and weaknesses of functional programming languages.

This thesis investigates an efficient evaluation technique for lazy functional programs that achieves a substantial speedup over previous combinator graph reduction and closure reduction techniques using comparable levels of hardware and compiler support. First, an abstract machine called the Threaded Interpretive Graph Reduction Engine (TIGRE) is designed and evaluated. Then, the architectural requirements of efficient support for TIGRE are explored.

The TIGRE approach to combinator graph reduction involves directly executing combinator graphs. Direct execution eliminates the cost of the tag checking case analysis associated with previous graph reduction techniques. The TIGRE abstract machine has been mapped efficiently onto a variety of conventional hardware platforms, and executes programs approximately two times faster than the quickest existing special purpose graph reduction hardware. Further speed improvements are demonstrated when using super-combinator compilation and strictness analysis techniques.

TIGRE programs behave in ways that are different than imperative programs, resulting in unconventional architectural support tradeoffs. The cache behavior of TIGRE displays very high temporal and spatial locality, but also a very high percentage of memory write operations. TIGRE can benefit from designs that provide cache with a large block size, minimize bus write traffic, support fast subroutine call instructions, support a top-of-stack buffer, and support double-word memory write operations.