Optimizing Compilation for Symbolic-Numeric Computation

Stephen Watt
Ontario Research Centre for Computer Algebra
University of Western Ontario

Dating back to the first programming languages, Fortran and Lisp, we have seen different forces direct the evolution of software for symbolic and numeric computation.

Numeric computation has been driven by the need for efficiency of floating point computation and by the need to specify the details of arithmetic operations. Symbolic computation has generally concerned itself with the creation and manipulation of complex dynamic data structures. Where numeric computation has focused on improved precision, symbolic computation has focused on lowering computational complexity.

To a great extent, we have evolved two scientific communities using different mathematical techniques and different software tools. This is an unsatisfying state of affairs, especially as these two communities have been starting to share similar concerns and methods. For example, numerical methods now use rather sophisticated data structures, e.g. to represent sparse matrices or multiple computational grids. Symbolic systems are now starting to treat approximate objects of various sorts and begin to incorporate analytic methods.

One of the primary concerns in the design of the Aldor programming language was that it be able to treat both symbolic and numeric quantities with the correct order of efficiency. That is, it was intended that its handling of numerics should be on the same order of speed as Fortran, and its handling of symbolics should be as rich and flexible as any other system. This was to be achieved through a language designed to admit optimizing compilation, yet with sufficiently rich semantics to be able to express subtle mathematical relationships.

This talk presents the aspects of the Aldor language design that provide a rich environment for efficient symbolic-numeric computation.