Activity Analysis for Operator Overloading Automatic Differentiation

Automatic Differentiation of Leadership-Class Applications

Drew Wicke
York College of Pennsylvania
441 Country Club Road, York, PA 17403

Sri Hari Krishna Narayanan, Paul D. Hovland
Argonne National Laboratory
9700 S Cass Ave, Argonne, IL 60439

Abstract

Automatic Differentiation (AD) automatically performs the differentiation of models expressed in a high level programming language. Several DOE-sponsored tools implement AD in the source-to-source transformation approach and the operator overloading approach. Operator overloading AD exploits the operator overloading features of the programming language that the model is expressed in. Runtime libraries implement methods that overload operators such as $+$, $-$, $*$, $/$ and mathematical functions such as *sin* and *sqrt* to compute derivative values along with elementary computation. Source-to-source transformation generates an output code that contains the original input code as well additional code to compute derivatives. It is typically faster than operator overloading AD as it can use compiler analysis such as activity analysis to optimize the generated code. Operator overloading AD is more robust as the tool does not have to handle esoteric syntactic and semantic programming language features itself.

Often, AD users are interested in computing the derivatives of a subset of the output variables in the model with respect to a subset of the input variables (known as dependent and independent variables respectively). Depending on the role a temporary variable plays in the model source code, it may need to be involved in the computation of derivatives as well. In operator overloading AD, such variables usually are identified manually using a compiler that detects type mismatch. The overheads associated with operator overloading are exacerbated by the overestimation of such variables.

Activity analysis is a two-step method to accurately identify those temporary variables that need to be involved in derivative computation. First, all variables whose value is affected by independent variables are detected. Second, all variables that affect the value of a dependent variable are detected. Variables that are detected in both steps are considered to be active.

We have developed a framework based on the activity analysis implementation within OpenAnalysis [1] to detect active variables within code that was used as input to Sacado - an operator overloading AD tool [3]. The types of variables that were determined to be active were changed to be of Sacado Active Type. The parsing of input code and type change was performed using the ROSE compiler framework and the useOA-ROSE interface [2, 4]. Preliminary experiments show that our tool is able to accurately distinguish those variables that must be active from those that should not.

References

[1] OpenAnalysis Web Page. `http://openanalysis.berlios.de/`.

[2] ROSE Web Page. `http://rosecompiler.org/`.

[3] Sacado Web Page. `http://trilinos.sandia.gov/packages/docs/dev/packages/sacado/doc/html/index.html`.

[4] UseOA-ROSE Web Page. `http://developer.berlios.de/projects/useoa-rose/`.