Practical Effects of Local Jacobian Preaccumulation[1]

I. Karlin[2] and J. Utke[3]

Computing derivatives of numerical models $f(x) \mapsto y : I\!R^n \mapsto R^m$ given as a computer program P is an important but also compute-intensive task. Automatic differentiation (AD)[4] provides the means to obtain such derivatives. AD can be implemented as a source transformation applied to P. Considering certain sets of semantically valid transformations we can impose efficiency criteria on the transformed program. The efficient preaccumulation of local Jacobians J_i is a crucial combinatorial problem within this context. Usually the objective is to minimize the number of fused multiply-add operations implied by a sequence of elimination steps that transforms a given computational graph into a bipartite graph representing J_i. The longstanding conjecture that this problem is NP-complete was proven recently.[5] Because of the complexity, practical solutions have centered on heuristics for the general case and constructing an optimal solution for the special case of single-expression-use graphs. The latter is of practical importance because it provides a means to optimally preaccumulate any right-hand-side expression in numerical programs.

However, the locally preaccumulated information (i.e., the J_i) only becomes useful in the context of the entire program P. To gauge the overall efficiency of a given approach we therefore need to take into account the effort needed to obtain the global from the local information. In theory, this could be modeled as a chained sparse matrix product of the preaccumulated local Jacobians along control flow path through P taken for a given argument x_0. This yields the global Jacobian $J|_{x=x_0}$ but is often prohibitively expensive and not actually required by the application context. Instead, the desired information is most often a set of Jacobian-vector products $J\dot{x}$ or $\bar{y}^T J$. Given an argument x_0, the sequence of k locally preaccumulated $J_i|_{x=x_0}$ at this argument the Jacobian vector products are the computed as $(J_k \cdot \ldots \cdot (J_1 \dot{x}) \ldots)$ (propagation in forward mode) or $(\ldots (\bar{y}^T J_k) \cdot \ldots \cdot J_1)$ (propagation reverse mode) respectively. However, the formulae do not readily expose the following two major concerns.

(I): Neither the exact scope of the program section represented by any of the individual J_i, nor their number is fixed. A natural choice for the scope of a single J_i is a consecutive sequence of l assignment statements, that is, typically the body of a basic block. Baring the presence of aliasing one can construct the computational graph[6] and approximate the optimal elimination order using various heuristics. A different but equally natural choice is the scope of a single assignment in which case the computational graph is a single expression use

[1]Abstract for Combinatorial Scientific Computing (CSC) 2007

[2]University of Colorado, ian.karlin@colorado.edu

[3]Argonne National Laboratory, utke@mcs.anl.gov

[4]see A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000.

[5]see U. Naumann, *Optimal Jacobian accumulation is NP-complete*. to appear in Math. Prog.

[6]see J. Utke, *Flattening Basic Blocks*. in Automatic Differentiation: Applications, Theory, and Implementations; M. Bücker et al Eds., LNCS, vol 50, pp 121–133, Springer, 2006.

graph for which one can construct an optimal elimination sequence. In other words, the J_i with basic block scope is replaced by a sequence of $(J_i^{(1)}, \ldots, J_i^{(l)})$. This implies a shift of computations from preaccumulation to propagation. At the same time we cannot make any general statement comparing the effort for $J_i\dot{v}$ or $\bar{v}^T J_i$ versus $(J_i^{(l)} \cdot \ldots \cdot (J_i^{(1)}\dot{v}) \ldots)$ or $(\ldots (\bar{v}^T J_i^{(l)}) \cdot \ldots \cdot J_i^{(1)})$ respectively.

(II): The reverse mode preaccumulates the J_i in index order but performs the vector-matrix propagation operations in reverse order. For large computations, the elements of $J_i, i = 1 \ldots k$ cannot all be stored in memory. Instead, a checkpointing and re-computation scheme is employed to compute p subsequences $(J_{s_t}), s = p \ldots 1, t = 1 \ldots t_s$ with $J_{p_{t_p}} = J_k$ and $J_{1_1} = J_1$. The principal goal is to compute as many J_i as fit into the available memory immediately followed by the reverse propagation $(\ldots (\bar{v}^T J_{s_{t_s}}) \cdot \ldots \cdot J_{s_1})$ for $s = p \ldots 1$. Not only are the storage and retrieval operations for the J_i elements a bottleneck themselves but reducing the storage requirements for preaccumulated information also implies fewer checkpoints and re-computations.

Because we are considering heuristics applied during the source transformation, the numbers of operations or data items mentioned above are static, without considering loop iterations and branching. Using the software package OpenAD, we examined some test examples taken from practically relevant programs. Concerning (I) we observe the quite plausible effect that switching from basic block level to statement level preaccumulation causes a decrease in preaccumulation operations which can for forward mode be accompanied by an increase accumulation operations. In some cases the total number of operations does not change significantly. In other test cases, however, the switch to statement level preaccumulation does also incur a decrease in propagation operations to a total reduction of as much as 90% which directly translates into sizeable runtime savings. Among the choices for graph scope, basic block and statements are only two of many options. On the other hand, considering the optimal preaccumulation problem nested in varying graph scopes appears somewhat artificial as the underlying dependency information does not change. There is evidence that a similar reduction in total operations should be obtainable by modified heuristics that determine the elimination order while considering scarcity [7] applied to the basic block level graphs.

Concerning (II) the main objective is the data reduction. A slightly different heuristic that also preserves scarcity should achieve this goal. There are fundamentally different approaches to reverse mode that do not include any preaccumulation. Consequently there is no simple heuristic to mix and match these approaches other than a runtime comparison.

The preaccumulation approach as described here restricts the possible transformations but allows applying a heuristic to a combinatorial problem in order to approximate an efficient solution within the considered transformation subset. We will discuss our approach in the context of our findings and present results.

[7] see A.Griewank, *A Mathematical View of Automatic Differentiation*, in Acta Numerica, vol 12, pp 321–398, Cambridge University Press, 2003.