Fast Statistical-based Interconnect Modeling Using Automatic Differentiation[1]

Lucas Roh[2]

Math and Computer Science Division, Argonne National Laboratory, Argonne, IL USA

Christian Bischof[3]

Institute for Scientific Computing, Technical University Aachen, Aachen, Germany

Norman Chang[4], **Ken Lee, Valery Kanevsky, O. Sam Nakagawa, and Soo-Young Oh**

Hewlett Packard Laboratory, 3500 Deer Creek Rd. Palo Alto, CA USA

Abstract

Automatic differentiation is a technique for computing derivatives accurately and efficiently with minimal human effort. We employed this technique to generate derivative information of FCAP2/3 programs that simulate the parasitic effects of interconnects. This derivative information is used in the statistical modeling of worst-case interconnect delays and on-chip crosstalks. The ADIC (Automatic Differentiation in C) tool generated new versions of FCAP2 and FCAP3 programs that compute both the original results and the derivative information. We report on the use of automatic differentiation and divided difference approaches for computing derivatives for FCAP3 programs. The results show that ADIC-generated code computes derivatives more accurately, more robustly, and faster than the divided difference approach.

1. Introduction

The parasitic effects of interconnects become very important as the geometry of VLSI/ULSI chips shrinks, and thus interconnect delay can easily be more than 70% of the total delay. Because of process variations, the critical path delay varies with the set of interconnects and devices. Thus, we need accurate models of the sensitivities of the delay with respect to these interconnects and devices in order to determine the worst-case behaviors. For this purpose, the modeling of statistically based worst-case (i.e., 3-sigma) delays is more desirable than determining the traditional skew-corner worst cases. Since a chip has millions of interconnects, a fast method is necessary to generate statistical-based worst case modeling of interconnects and devices.

We have developed a methodology [2] for obtaining 3-sigma R (resistance), C (capacitance), crosstalk, and delay given variations in interconnect-related process parameters. This methodology uses FCAP2 and FCAP3 [3] (Fast Capacitance Extraction 2-D and 3-D simulators, respectively) that have been developed at Hewlett Packard Laboratory to study parasitic electrical effects of interconnects and devices. Using this methodology for a long critical net analysis on a 0.35 um process, we realized a more than 70% improvement in 3-delay delay estimation compared with the traditional skew-corner worst case delay. This

[1] This work was supported by the Mathematical, Information, and Computational Sciences Division Subprogram of the Department of Energy, under contract W-31-109-Eng-38, and by Technical Service Agreement No. 85G50 with the Hewlett-Packard Corporation.

[2] roh@mcs.anl.gov

[3] bischof@sc.rwth-aachen.de

[4] nchang@hplabs.hp.com

methodology relies on accurate derivatives of the FCAP-generated results. A single run of on-chip statistical modeling takes on the order of a week on a fast workstation, and most of this time is spent in computing the derivatives.

Previously, our methodology obtained derivatives by estimating them by using divided difference schemes. The advantage of this traditional approach is that the function (in this case, the simulator) can be treated as a black box. The disadvantage is that the time required to compute derivatives grows linearly with the number of independent variables, and the accuracy of derivatives may be compromised severely as a result of truncation and cancellation errors.

Recently, the automatic differentiation technique has been gaining popularity due to its capability to produce accurate derivative codes in an automated fashion and for its generality. An automatic differentiation tool takes a code comprising a function, such as FCAP2/FCAP3, and generates a derivative code that evaluates the derivative of the function with respect to the specified independent variables. No limits are imposed on the length or the complexity of the program. Hence, general techniques that rely on the output of semiconductor computer simulation models, such as optimal design and sensitivity or reliability analysis can all benefit from using automatic differentiation.

2. Automatic Differentiation Using ADIC

In this section, we briefly review automatic differentiation techniques and describe our tool that implements them. Every function, no matter how complicated, is executed on a computer as a (potentially very long) sequence of elementary operations (addition, multiplication, etc.) and elementary functions (sine, cosine, etc.). By applying the chain rule of differential calculus over and over again to the composition of those elementary operations, one can compute the derivative information exactly (up to the machine precision) and in a completely mechanical fashion [4,7].

Derivative accuracy is a built-in feature of automatic differentiation. Improvements in the complexity are driven mainly be smarter ways of exploiting the associativity of the chain rule of differential calculus and by exploiting mathematical insight concerning the algorithms governing the underlying program.

Several tools have been developed to handle the automatic differentiation process. They include ADIFOR [5], ODYSSEE, and ADOL-F for Fortran programs and ADOL-C and ADIC for C programs. For an up-to-date account, readers are referred to the documentation available on the World Wide Web under URL http://www.mcs.anl.gov/autodiff/adtools/.

In our work, we employed the ADIC (Automatic Differentiation in C) [1] tool. Given an ANSI C routine or a collection of routines describing a function, ADIC uses the source-to-source program transformation technique to produce a new, portable C code that computes derivatives of the output variables with respect to any independent variables. Hence, general techniques that rely on the output of computer simulation models, such as optimal design and sensitivity or reliability analysis, can all benefit from using automatic differentiation. Currently, ADIC is the only automatic differentiation tool for ANSI C that employ source transformation. ADIC has already been successfully applied to a 3-D volume grid generator for CFD applications [6], a vehicle simulator, and a neural network specification.

3. Statistical Modeling and Differentiating of FCAP Codes

Our methodology addresses the problem of quantifying the impact of process-induced interconnect variations on resistance (R) and capacitance (C) and circuit performance. Our methodology for obtaining statistically based worst-case (i.e., 3-sigma) R (resistance), C (capacitance), crosstalk, and delay given variations in interconnect-related process parameters is divided into three phases.

In the first phase, the 3-sigma values of capacitance, resistance, and partial derivatives of capacitances with respect to selected interconnect process parameters are generated in batch-mode computation as part of an enhanced version of HIVE [2], which is a parameterized interconnect model generator and library for R and

C, and the Derivative HIVE Generator. Most of time is spent in derivative calculation in this phase. In the second phase, randomized but correlated R and C are generated via a Monte Carlo method in a distributed N-Pi network for a given net via the randomized RC generator. In the last phase, the randomized RC net and nominal/3-sigma device models can be combined to characterize delay or crosstalk variation based on device and interconnect variations.

Capacitance and resistance are computed using FCAP2/FCAP3. Originally, the derivatives were computed using divided differences method. Then by using ADIC, we generated the differentiated version of FCAP2/FCAP3 codes, respectively named FCAP2.AD/FCAP3.AD. FCAP2.AD/FCAP3.AD compute both the original function values plus their derivative values. We have also post-optimized FCAP3.AD program by hand that specially optimized the inner loop of the FCAP's Poisson solver. As ADIC tool is improved, the tool should be able to perform this automatically. A particular run of the statistical modeling methodology with FCAP2 takes about 5-10 days of CPU time on an HP9000/755 workstation. The computational complexity increases by an order of magnitude when FCAP3 is employed. Most of this time is spent in computing derivatives; therefore, any method that reduces the derivative computation cost is significant.

4. Measurements

For our experiments, we use two input models that compute (1) capacitances of two layers of 5-trace signals routed orthogonally between two ground layers, and (2) potentials of two vertically parallel signals routed between two ground layers. The experiments were performed on a Hewlett Packard 9000/780 workstation running HP-UX 10.20 and compiled using the Softbench C compiler with full optimizations.

Table 1 shows the runtime performance using divided-difference approximations versus ADIC "out of the box" and postoptimized derivative code for FCAP3. Central differences, which, unlike one-sided differences, usually deliver acceptable derivative approximations for FCAP2/3, would have required 2p+1 function evaluations to compute p derivatives plus the function values. Comparing the derivative values computed via central differences and automatic differentiation, we found that these values agreed to within one half of a percent.

The measurements are made for 2, 5, and 10 independent variables. The columns **AD/Func.** represent the runtime ratio of FCAP3.AD over FCAP3. The columns **DD/AD** represent the runtime ratio of using central divided difference approximations versus FCAP3.AD. We see that derivative code generated by ADIC out of the box (AD) is 1.2 to 2.0 times faster than the divided-difference method (DD); and as the number of independent variable increases, the speedup increases. This is due to the cost of non-floating-point computations as well as certain computations are amortized over larger number of derivative computations. In the case of postoptimized FCAP3.AD, the results are an additional factor of 1.7 to 2.4 times faster than ADIC out of the box. Hence, the postoptimization steps can significantly improve the runtime performance. In either case, since the statistical modeling of on-chip interconnect properties is dominated by the cost of computing derivatives, considerable improvements are realized in the overall modeling process.

# of Indep. Variables	**2**		**5**		**10**	
Model 1 **Function = 63.6 sec.**	**AD/Func.**	**DD/AD**	**AD/Func.**	**DD/AD**	**AD/Func.**	**DD/AD**
AD	**4.08**	**1.23**	**6.92**	**1.59**	**12.1**	**1.74**
AD Post-optimized	**2.10**	**2.38**	**3.52**	**3.13**	**4.93**	**4.26**
Model 2 **Function = 31.3 sec.**	**AD/Func.**	**DD/AD**	**AD/Func.**	**DD/AD**	**AD/Func.**	**DD/AD**

AD	**3.58**	**1.40**	**6.28**	**1.75**	**10.4**	**2.02**
AD Post-optimized	**2.08**	**2.40**	**3.71**	**2.96**	**4.99**	**4.21**

Table 1. Comparison of FCAP3.AD versus central divided differences for two different input models and three different sets of independent variables (3, 5, and 7). AD/Func. represents the runtime ratio of FCAP3.AD over function evaluation (FCAP3). DD/AD represents the runtime ratio of central divided differences over FCAP3.AD.

6 Conclusions

FCAP2 and FCAP3 are 2-D and 3-D, respectively, simulators to measure the parasitic electrical effects of interconnects and devices. In the statistical modeling of interconnects, the evaluation of gradients of FCAP-generated results are required. Conventional techniques cannot be relied upon to deliver fast and accurate derivatives. Divided differences may not be accurate and are obtained slowly, symbolic approaches do not appear to be feasible, and hand coding of derivatives is impractical. In contrast, automatic differentiation can be used to obtain fast and accurate derivatives for functions defined by large codes.

The experiments show that ADIC-generated derivatives reduce dependence on grid variations compared with the central divided difference method that had been employed before, while at the same time executing up to twice as fast. By postoptimizing ADIC-generated derivative codes, (namely, identifying inactive functions and employing the reverse mode of differentiation across a code block of the linear solver), performance was improved by roughly another factor of two. Since the cost of interconnect modeling is dominated by derivative computation, these derivative improvements result in considerable speedup overall.

Automatic differentiation is a field in its infancy. Improvements in the complexity of AD-generated derivative codes are driven by using smarter ways to exploit the associativity of the chain rule of differential calculus, by exploiting mathematical insight concerning the algorithms governing the underlying program, and by improving the program analysis capabilities of AD tools.

References

[1] C. Bischof , L. Roh, A. Mauer, "ADIC: An Extensible Automatic Differentiation Tool for ANSI-C," Software: Practice and Experience, Vol. 27(12), pp. 1427-1456, December 1997.

[2] N. Chang, V. Kanevsky, O. S. Nakagawa, K. Rahmat, and S. Y. Oh, "Fast Generation of Statistically Based Worst Case Modeling of On-Chip Interconnects," International Conference on Computer Design, Oct. 1997.

[3] K. Cham, S. Y. Oh, D. Chin, J. L. Moll, K. Lee, and P. V. Voorde, "Computer-Aided Design and VLSI Device Development," second edition, Kluwer, 1988.

[4] M. Berz, C. Bischof, G. Corliss, and A. Griewank, "Computational Differentiation: Techniques, Applications, and Tools," SIAM, Philadelphia, 1996.

[5] C. Bischof, A. Carle, P. Khademi, and A. Mauer, "ADIFOR 2.0: Automatic Differentiation of FORTRAN 77 Programs," IEEE Computational Science & Engineering, 3(3):18-32, 1996.

[6] C. Bischof , W. Jones, A. Mauer, and J. Samareh, "Experiences with the Application of the ADIC Automatic Differentiation Tool to the CSCMDO 3-D Volume Grid Generation Code," in Proceedings of the 34th AIAA Aerospace Sciences Meeting, AIAA Paper 96-0716, American Institute of Aeronautics and Astronomics, 1996.

[7] A Griewank and G. F. Corliss, "Automatic Differentiation of Algorithms: Theory, Implementation, and Application," SIAM, Philadelphia, 1991.