Tempo: Specializing Systems Applications and Beyond

C. Consel, L. Hornof, R. Marlet, G. Muller, S. Thibault, E.-N. Volanschi,
IRISA / INRIA - University of Rennes, France
and
J. Lawall
Oberlin College, Ohio
and
J. Noyé
École des Mines de Nantes, France

Tempo is a partial evaluator designed to exploit specialization opportunities in systems applications. Obtaining efficient systems has become increasingly important in the context of pervasive high-speed networks, interactive multimedia, and large distributed databases. Toward this end, Tempo was developed by first studying specific applications to be optimized, and then working backwards to create a tool that would perform the desired specialization. As Tempo consists of many subcomponents, many of which are research topics themselves, our strategy was to combine the simplest subcomponents that our studies showed were needed. Whenever possible, we simply reused existing technology. When limitations were encountered, we identified the fundamental problem and developed new techniques to overcome it. This paper summarizes the development process, assesses the resulting partial evaluator, and provides some perspectives for future work.

1. SPECIALIZING SYSTEMS PROGRAMS

Numerous specialization opportunities exist in operating systems programs since, for software engineering reasons, they must be general and modular [Consel et al. 1993; Hornof 1997; Pu et al. 1995; Volanschi et al. 1996]. Some of these opportunities are a result of the way components are combined. For example, calling contexts of generic library functions often include invariants that can be exploited. Other opportunities are created while the operating system is running. For example, values that become known when a file is opened remain invariant during

subsequent reads and writes to that file [Pu et al. 1995]. To take advantage of such opportunities, a partial evaluator must:

—*Treat the C programming language,* since operating systems are typically written in C.
—*Exploit specialization opportunities,* such as those due to generality and modularity.
—*Exploit run-time invariants,* like those found in file reads and writes.
—*Handle large applications,* because certain operating systems contain up to a million lines of code.
—*Ensure predictability,* by informing the user how the code will be transformed during specialization.

Tempo addresses each of these needs as follows:

Reusing Existing Analyses. In order to treat the complex constructs of the C programming language, Tempo includes a number of standard analyses [Consel et al. 1996]. Dealing with pointers and imperative features requires both an alias analysis and a side-effect analysis [Banning 1979; Emami et al. 1994]. For these analyses, a flow-sensitive, context-insensitive analysis based on existing research was sufficient for our purposes.

New Analysis Features. A number of improvements to the binding-time analysis were necessary to exploit the aforementioned specialization opportunities [Hornof 1997]. For example, return and use sensitivity were needed to effectively treat the complex way in which data is passed interprocedurally [Hornof and Noyé 1997; Hornof et al. 1997].

Template-Based Run-Time Specialization. The fundamental challenge in exploiting run-time invariants is to generate the best possible code in the least amount of time, since the cost of code generation is incurred at run-time. Tempo provides a template-based run-time specializer [Consel and Noël 1996; Noël et al. 1996]. Templates are automatically generated, compiled, and optimized at compile time. This approach minimizes run-time costs while producing high-quality code.

Module-Oriented Specialization. The bottleneck of large applications is often located in only small parts of the code. Tempo facilitates the optimization of these small parts with a feature known as *module-oriented* specialization [Consel et al. 1996]. The set of functions to be optimized is extracted from the original application and reinserted after specialization. An interface file describes the context from which the functions are extracted, and Tempo preserves the interface during specialization.

Visualization tools. To enable a systems programmer to understand and follow the specialization process, Tempo includes visualization tools that display the results of each analysis.

2. ASSESSMENT

By incorporating the features described above into Tempo, we were able to successfully specialize systems programs as well as programs from other domains.

Operating Systems

One major concern in distributed systems is obtaining a fast remote procedure call protocol (RPC), a protocol that makes a remote procedure look like a local one [Sun Microsystem 1989]. A considerable amount of work has been dedicated to optimizing RPC in different operating systems, either by building new implementations or by manually optimizing critical sections of existing code [Schroeder and Burrows 1990]. Once a given client/server interface is fixed, specialization opportunities exist in the data encoding/decoding functions. We used Tempo to automatically optimize the Sun RPC, a commercial RPC implementation that has become a *de facto* standard [Muller, Marlet, Volanschi, Consel, Pu, and Goel 1997; Muller, Volanschi, and Marlet 1997]. Optimized encoding/decoding functions ran up to 3.7 times faster, yielding an overall speedup of 1.5 for the round-trip call. Tempo has also been used to optimize other operating system applications, including signal delivery and memory allocation.

Other domains

Specializing systems programs required the development of a sophisticated partial evaluator. Tempo is also capable of successfully specializing programs in other domains, which are often simpler or less hand-optimized.

Scientific Algorithms. Significant speedups were obtained by specializing scientific algorithms, such as cubic spline interpolation, Romberg integration, and fast Fourier transformation (FFT) [Noël et al. 1996].

Graphics Programs. Specializing graphics programs, such as image filtering [Noël et al. 1996] and ray tracing, has also proven effective.

Software Engineering. Tempo has also been used in software architectures in order to tightly integrate separate components into an efficient implementation [Marlet et al. 1997]. Tempo plays a key role in a framework of application-generator design, exploiting the fact that partial evaluation is especially suited to specialize interpreters [Thibault and Consel 1997; Thibault et al. 1997].

3. FUTURE WORK

We are continuing to use Tempo to specialize more applications. Additionally, using Tempo as a core transformation engine, we are pursuing two directions to widen its applicability and improve its usability.

Back-End Specializer. In order to extend the applicability of Tempo, we are investigating ways of using it in a multi-language specialization platform. By translating languages such as Fortran, C++, and Java into C, we can use Tempo as a *back-end* specializer. For example, we have added a front-end containing the

`cfront` translator to specialize part of the Chorus IPC subsystem, written in C++. We have also developed Harissa, a translator from Java to C, to specialize Java programs [Muller, Moura, Bellard, and Consel 1997].

Specialization Toolkit. We are also working on complementing Tempo with other tools to make specialization easier for non-programming language experts. Some tools to assist module-oriented specialization have already been developed in collaboration with systems researchers at the Oregon Graduate Institute. TypeGuard and MemGuard analyze large programs and provide information concerning invariant usage; Replugger allows a generic version of a function to be replaced with a specialized version on the fly.[1] A compiler for *specialization classes*,[2] when interfaced with Tempo, allows the user to express what, when, and how to specialize a program using a high-level specification language [Volanschi et al. 1997]. Finally, we are developing a tool that uses profiling information to detect bottlenecks and specialization opportunities in programs.

Acknowledgements

This research was supported in part by SEPT/France Telecom grant CTI 951W009, ARPA grant N00014-94-1-0845, and NSF grant CCR-92243375.

REFERENCES

Banning, J. 1979. An efficient way to find the side effects of procedure calls and the aliases of variables. In *Conference Record of the* 6^{th} *annual ACM Symposium on Principles Of Programming Languages* (San Antonio, TX, USA, Jan. 1979), pp. 29–41. ACM Press.

Consel, C., Hornof, L., Noël, F., Noyé, J., and Volanschi, E. 1996. A uniform approach for compile-time and run-time specialization. In O. Danvy, R. Glück, and P. Thiemann Eds., *Partial Evaluation, International Seminar, Dagstuhl Castle*, Number 1110 in Lecture Notes in Computer Science (Feb. 1996), pp. 54–72.

Consel, C. and Noël, F. 1996. A general approach for run-time specialization and its application to C. In *Conference Record of the* 23^{rd} *Annual ACM SIGPLAN-SIGACT Symposium on Principles Of Programming Languages* (St. Petersburg Beach, FL, USA, Jan. 1996), pp. 145–156. ACM Press.

Consel, C., Pu, C., and Walpole, J. 1993. Incremental specialization: The key to high performance, modularity and portability in operating systems. In *Partial Evaluation and Semantics-Based Program Manipulation* (Copenhagen, Denmark, June 1993), pp. 44–46. ACM Press. Invited paper.

Emami, M., Ghiya, R., and Hendren, L. 1994. Context-sensitive interprocedural points-to analysis in the presence of function pointers. In *Proceedings of the ACM SIGPLAN '94 Conference on Programming Language Design and Implementation* (June 1994), pp. 242–256. ACM SIGPLAN Notices, 29(6). ACM Press.

Hornof, L. 1997. *Static Analyses for the Effective Specialization of Realistic Applications.* Ph. D. thesis, Université de Rennes I.

Hornof, L. and Noyé, J. 1997. Accurate binding-time analysis for imperative languages: Flow, context, and return sensitivity. In *ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation* (Amsterdam, The Netherlands, June 1997), pp. 63–73. ACM Press.

[1] TypeGuard, MemGuard, and Replugger are publically available at `http://www.cse.ogi.edu/DISC/projects/synthetix/toolkit/`.

[2] The Java Specialization Classes compiler is publically available at `http://www.irisa.fr/compose/sc/`.

Hornof, L., Noyé, J., and Consel, C. 1997. Effective specialization of realistic programs via use sensitivity. In P. Van Hentenryck Ed., *Proceedings of the Fourth International Symposium on Static Analysis, SAS'97*, Volume 1302 of *Lecture Notes in Computer Science* (Paris, France, Sept. 1997), pp. 293–314. Springer-Verlag.

Marlet, R., Thibault, S., and Consel, C. 1997. Mapping software architectures to efficient implementations via partial evaluation. In *Conference on Automated Software Engineering* (Lake Tahoe, Nevada, Nov. 1997), pp. 183–192. IEEE Computer Society.

Muller, G., Marlet, R., Volanschi, E., Consel, C., Pu, C., and Goel, A. 1997. Fast, optimized Sun RPC using automatic program specialization. Rapport de recherche RR-3220 (July), INRIA, Rennes, France.

Muller, G., Moura, B., Bellard, F., and Consel, C. 1997. Harissa: A flexible and efficient Java environment mixing bytecode and compiled code. In *Proceedings of the 3rd Conference on Object-Oriented Technologies and Systems* (Portland (Oregon), USA, June 1997), pp. 1–20. Usenix.

Muller, G., Volanschi, E., and Marlet, R. 1997. Scaling up partial evaluation for optimizing the Sun commercial RPC protocol. In *ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation* (Amsterdam, The Netherlands, June 1997), pp. 116–125. ACM Press.

Noël, F., Hornof, L., Consel, C., and Lawall, J. 1996. Automatic, template-based runtime specialization : Implementation and experimental study. Rapport de recherche 1065 (Nov.), IRISA, Rennes, France.

Pu, C., Autrey, T., Black, A., Consel, C., Cowan, C., Inouye, J., Kethana, L., Walpole, J., and Zhang, K. 1995. Optimistic incremental specialization: Streamlining a commercial operating system. In *Proceedings of the 1995 ACM Symposium on Operating Systems Principles* (Copper Mountain Resort, CO, USA, Dec. 1995), pp. 314–324. ACM Operating Systems Reviews, 29(5), ACM Press.

Schroeder, M. and Burrows, M. 1990. Performance of Firefly RPC. *ACM Transactions on Computer Systems 8*, 1 (February), 1–17. ACM Press.

Sun Microsystem 1989. NFS: Network file system protocol specification. RFC 1094 (March), Sun Microsystem. ftp://ds.internic.net/rfc/1094.txt.

Thibault, S. and Consel, C. 1997. A framework of application generator design. In *Proceedings of the Symposium on Software Reusability* (May 1997), pp. 131–135. ACM Press.

Thibault, S., Marlet, R., and Consel, C. 1997. A domain-specific language for video device drivers: from design to implementation. In *Conference on Domain Specific Languages* (Santa Barbara, CA, Oct. 1997), pp. 11–26. Usenix.

Volanschi, E., Consel, C., Muller, G., and Cowan, C. 1997. Declarative specialization of object-oriented programs. In *OOPSLA'97 Conference Proceedings* (Atlanta, USA, Oct. 1997), pp. 286–300. ACM Press.

Volanschi, E., Muller, G., and Consel, C. 1996. Safe operating system specialization: the RPC case study. In *Workshop Record of WCSSS'96 – The Inaugural Workshop on Compiler Support for Systems Software* (Tucson, AZ, USA, Feb. 1996), pp. 24–28.