A Performance Test Platform

R. Heinzl, G. Mach, P. Schwaha, and S. Selberherr

Institute for Microelectronics, TU Wien
Gußhausstraße 27-29/E360, A-1040 Vienna, Austria
E-mail: {heinzl|mach|schwaha|selberherr}@iue.tuwien.ac.at

KEYWORDS
Performance test, benchmark, graphical user interface, programming languages, web-application, Linux, posix, Bash

ABSTRACT

We implemented a fully automated benchmarking system, the performance test platform (PTP). It is developed with special emphasis on flexibility and therefore uses library centric application design combining several orthogonal modules. The modules consist of a base of Bash shell scripts and an orthogonal web front-end written in PHP. We thereby ensure that no restrictions are imposed on the benchmarked systems. Furthermore we show the platform's application and results obtained by benchmarking different algebraic equation specification approaches.

INTRODUCTION AND MOTIVATION

We present the performance test platform (PTP), a fully automated benchmark system to consequently analyze compile-time and run-time performance of different algorithms, programming paradigms, or even complete applications. The compiler attributes are fully parametrized, which allows the use of different compilers and options on multiple hosts with several operating systems. It is not only possible to pass various options to the compiler, but it is also possible to pass different options to the newly compiled executable. The rapid development of micro processors and the huge range of currently available processor types requires very complex tools to analyze applications in order to optimize their run-time performance. Additionally, compilers also undergo their own evolution and different compilers introduce a new set of optimization possibilities to control the new features of modern micro processors with each new generation. The possibilities of combining all of these switches and options is increasing even more rapidly and results in a staggering amount of possibilities to choose from. A manual selection of settings to obtain optimal performance is therefore not an efficient task as even different types of applications require a different set of options and switches.

A project was therefore initiated with the goal to determine which compiler-options result in optimal code on different architectures, operating systems, and from different programming paradigms. During the development of a batch-job-system using Bash shell scripts [12] the idea to extend this to a complete performance test platform was born. Once a system consisting of several shell scripts and a hierarchical directory structure was developed, usability had to be increased to make this system accessible on various platforms without manual interaction. A database system, a graphical user interface, as well as graphical statistic modules have therefore been developed.

THE DEVELOPED PLATFORM

The base of the developed system consists of a collection of Bash shell scripts and an orthogonally implemented graphical web front-end in the dynamic web programming language `PHP` [5], cf. Figure 1. Both are supplemented by a `MySQL`-database [13] and `GNU R` [7], a statistical and graphical suite used to process the results and create graphical output.

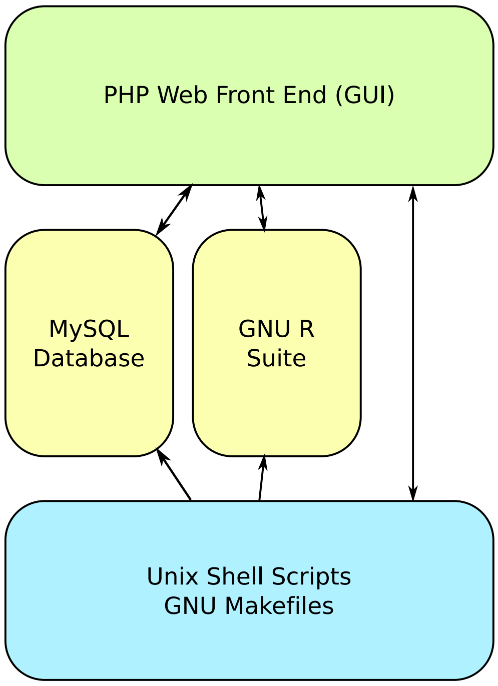


Figure 1: Overview of the performance test platform.

We chose a shell-script backend to ensure the possibility to easily change parameters, templates and scripts manually or by the graphical user interface. We thereby achieve a maximum of freedom and openness for a large variety of systems, languages, compilers, and algorithms to test. The database was implemented in `MySQL`, because it is widely available, easy to use, and very stable when using InnoDB-engines [15, 11] with foreign constraints. Using these features and a graphical front-end such as phpMyAdmin have the possibility to apply even complex changes (i.e. changing the constraints or dependencies of tables) in an efficient and safe way.

We also implemented a graphical user interface module for GNU R to finally obtain HTML pages. It provides the facilities to produce graphical output in the style of common spreadsheet-applications from performance test results. The templates used guarantee an easy way to modify the style of the resulting figures.

We chose a web front-end written in PHP to control the test platform for many reasons. First, we can use nearly every system providing a browser to control the platform, second we did not need to implement a separate GUI, and, finally, the PHP language provides facilities to execute programs on different systems. We can therefore use the web-front-end with other underlying systems as well.

The most important feature of the base system is its modularity and the resulting exchangeability of modules. The main modules can be used without the graphical front-end. This is ensured by using a sophisticated directory structure and small task specific shell scripts allowing modification or substitution of single tasks in the benchmark system or single configurations for one benchmark. The appearance and style of the final figures can thereby be changed without any modification of the core system itself.

The directory structure consists of a directory for each test. This root directory then contains directories for configuration files, source files, and templates, as well as directories for additional files, e.g. input files, concerning the simulation process and for each configuration, option, and host. Figure 2 shows the directory tree of the test-directory.

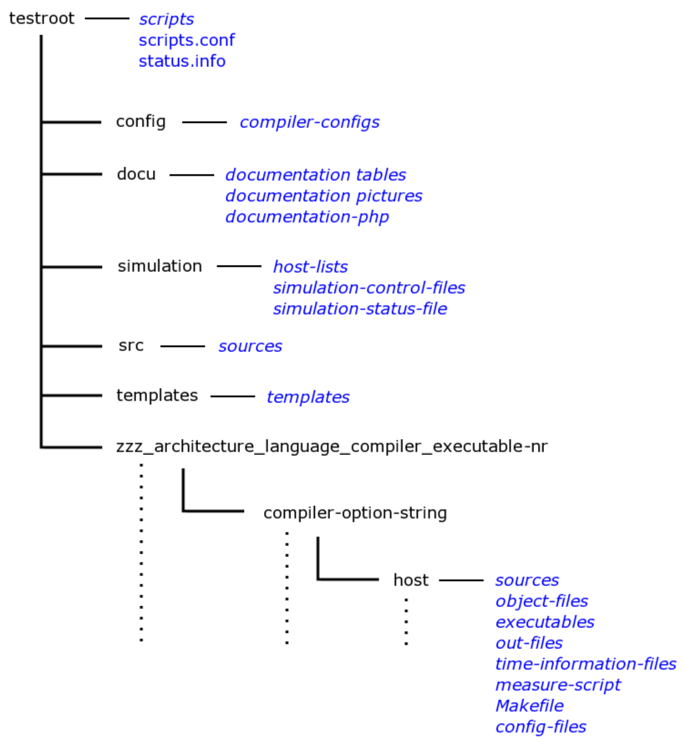


Figure 2: The directory tree of the PTP.

The config directory contains all configuration files for a job, where each test (different language, executable, or compiler) has its own configuration. The following table gives an overview of some of the possible tags.

Tag	Explanation
xxxARCHITECxxx	architecture of the system
xxxLANGUAGExxx	language of the tested program
xxxHOSTxxx	space-separated list of hosts to use for the tests

The template directory contains templates for Makefiles and time-measure scripts. The time-measure scripts are invoked by the runonhost.sh script and measure the compile time as well as the execution time. The src directory contains the sources for all tests of this job (different programs, compilers, and hosts). The zzz_* directories are named according to the used architecture, the language, the compiler, and a unique executable number. These directories contain one subdirectory per compiler-option-string. Each of these directories contains one directory per host. There are several files in these directories: makefile.conf and measure.conf are the configuration files for the Makefile and measure.sh script. Additionally, the source files along with the resulting object files and executables and, after the tests have been run, the files containing the test results are located here.

The simulation directory contains, beside a few control files such as simulation.started and simulation.ended, the status file simulation.status and for each host the job lists and output files. The simulation can be controlled by editing the host job lists or the two control files.

After running the tests from the web-front-end, the docu directory contains all tables, figures and a PHP-file, which is included in the front-end-documentation. In this way it is guaranteed, that the documentation is accessible from the front-end in the typical style, but can also be used without the front-end.

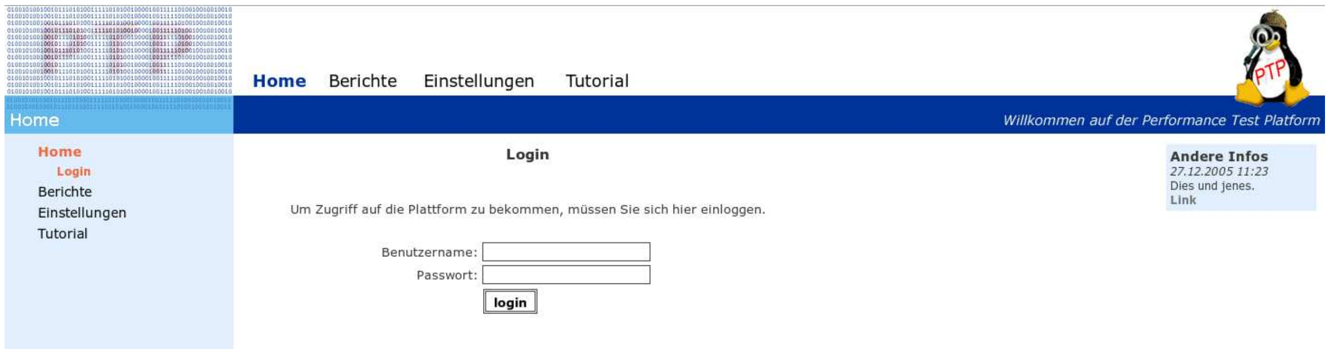
The shell-scripts mirror the procedure of the benchmark test:

- First the directory structure is created and set up.
- The files are then distributed and the actual tests are run.
- The final task is to document the results.

THE WEB-FRONT-END

PHP-Scripts control the basic layout, the login (see Figure 3), the connection to the database as well as the report section, the control section, and the tutorial section.

Authentication is performed against the login management of the server (i.e. local accounts using PAM [1] or network accounts using NIS [2]). Therefore no separate login-management nor accounts have to be pro-



Login

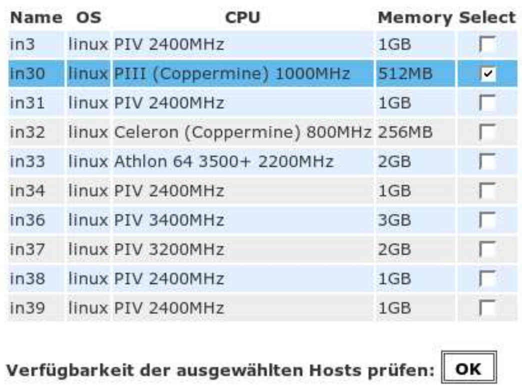
Um Zugriff auf die Plattform zu bekommen, müssen Sie sich hier einloggen.

Benutzername:
Passwort:
login

Andere Infos
27.12.2005 11:23
Dies und jenes.
Link

Figure 3: Login to the performance test platform.

vided, but all users having valid logins at the server can login to the platform out of the box. The biggest advantage of this login management is its modularity, so even advanced authentication systems such as `kerberos` [3] can easily be incorporated. Cookies are used to avoid outdated login sessions. The basic graphical layout of the PTP is different for normal users and for admins, respectively.
All reports concerning hosts, jobs and the platform can be found in the report-section ('Berichte'). Access to reports dealing with the platform is restricted to admins only, all other reports are viewable by every user of the platform.

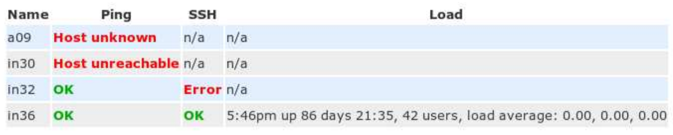


Name	OS	CPU	Memory	Select
in3	linux	PIV 2400MHz	1GB	☐
in30	linux	PIII (Coppermine) 1000MHz	512MB	☑
in31	linux	PIV 2400MHz	1GB	☐
in32	linux	Celeron (Coppermine) 800MHz	256MB	☐
in33	linux	Athlon 64 3500+ 2200MHz	2GB	☐
in34	linux	PIV 2400MHz	1GB	☐
in36	linux	PIV 3400MHz	3GB	☐
in37	linux	PIV 3200MHz	2GB	☐
in38	linux	PIV 2400MHz	1GB	☐
in39	linux	PIV 2400MHz	1GB	☐

Verfügbarkeit der ausgewählten Hosts prüfen: OK

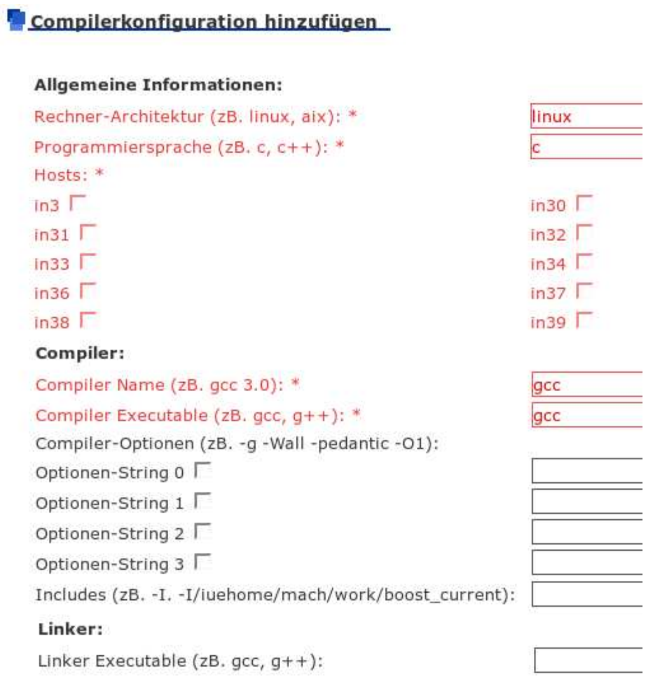
Figure 4: List of hosts in report section.

Reports about the hosts (Figure 4) contain the names of the hosts, their operating systems, the CPU type and speed, and the amount of main memory. Information on the availability of selected hosts which can be used by the system for benchmarks is provided on a separate page (see Figure 5).
Reports about the jobs contain the root-directory of the job and its current status. Information about logins is available in the platform report section.
All controls about hosts, jobs, and the platform can be found in the control section ('Einstellungen'). A user can change only the settings of his own jobs.



Name	Ping	SSH	Load
a09	Host unknown	n/a	n/a
in30	Host unreachable	n/a	n/a
in32	OK	Error	n/a
in36	OK	OK	5:46pm up 86 days 21:35, 42 users, load average: 0.00, 0.00, 0.00

Figure 5: Overview of the availability of selected hosts.



Compilerkonfiguration hinzufügen

Allgemeine Informationen:

Rechner-Architektur (zB. linux, aix): * linux
Programmiersprache (zB. c, c++): * c
Hosts: *
in3 ☐ in30 ☐
in31 ☐ in32 ☐
in33 ☐ in34 ☐
in36 ☐ in37 ☐
in38 ☐ in39 ☐

Compiler:

Compiler Name (zB. gcc 3.0): * gcc
Compiler Executable (zB. gcc, g++): * gcc
Compiler-Optionen (zB. -g -Wall -pedantic -O1):
Optionen-String 0 ☐
Optionen-String 1 ☐
Optionen-String 2 ☐
Optionen-String 3 ☐
Includes (zB. -I. -I/iuehome/mach/work/boost_current):

Linker:

Linker Executable (zB. gcc, g++):

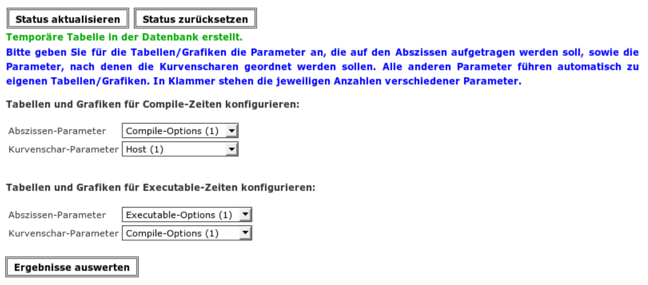
Figure 6: Mask to add a configuration: the red parts are mandatory.

The mask for adding configurations (Figure 6) refers to the configuration tags shown earlier in this paper. It is divided into several sections:

- The section 'Allgemeine Informationen' deals with the architecture and the language and the selection of hosts that should be used.

- The 'Compiler' section contains the name and executable of the compiler. Compiler option strings and include paths can be specified optionally.
- The next section is called 'Linker'. If this entry is empty, the compiler executable is assumed to be used for linking as well. There are optional fields for linker flags, library paths, and libraries to link against.
- In the section 'Source-Files' the names of the source files along with the header files needed to compile the program are specified.
- In the last section 'Make and Executable' it is possible to specify the name of the make command and the executable. Flags and switches for the make command or the executable option strings can also be entered here.

The important part of analyzing the results of the performance test runs is managed by a wrapper tool to `GNU R`. Different templates are used to interpret the corresponding output of the single tasks obtained by the shell scripts. Transformations of the output data, selections, and finally illustration templates have been developed to ease the complex process of intelligence collection. The 'Analyze Results' mask is shown in Figure 7.



Temporäre Tabelle in der Datenbank erstellt.

Bitte geben Sie für die Tabellen/Grafiken die Parameter an, die auf den Abszissen aufgetragen werden soll, sowie die Parameter, nach denen die Kurvenscharen geordnet werden sollen. Alle anderen Parameter führen automatisch zu eigenen Tabellen/Grafiken. In Klammer stehen die jeweiligen Anzahlen verschiedener Parameter.

Tabellen und Grafiken für Compile-Zeiten konfigurieren:

Abszissen-Parameter: Compile-Options (1)
Kurvenschar-Parameter: Host (1)

Tabellen und Grafiken für Executable-Zeiten konfigurieren:

Abszissen-Parameter: Executable-Options (1)
Kurvenschar-Parameter: Compile-Options (1)

Figure 7: The 'Analyze Results' mask.

The parameter for the abscissa and the set of curves can be chosen for compile times as well as execute times. All other parameters result in separate tables and figures based on given and adjustable templates.

RESULTS OF A BENCHMARK

Using the performance test platform we compared different equation specification approaches, based on a generic scientific simulation environment [9, 14, 10] and the corresponding performance analysis [8]. Therefore we analyzed several techniques which are available in C++.

CPU type	Clock speed	RAM	Compiler	MFLOPS
P4	2.8 GHz	2 GB	GCC 4.0.2	2310.9
Core Duo	2.4 GHz	2 GB	GCC 4.3	8804.0
Athlon 64	2.2 GHz	2 GB	GCC 4.1.2	10134.0

We compared the following equation specification methods:

- The expression template technique was used as described in [4].
- The GNU g++ implementation of the `valarray` data type was used as a test.
- Blitz++ [16]
- Naive C++ implementation
- The GSSE approach [9] based on topological traversal and an equation concept based mostly on the Boost Phoenix library [6].

The test is performed using a vector addition $A_f = A_b + A_c + A_d$, evaluated with different vector sizes. The definitions of the corresponding data types are given in the next code snippet:

```
typedef std::valarray<double>   Array_t;
typedef blitz::Array<double,1>  Array_t;
typedef std::vector<double>     Array_t;
```

The next figures presents and compare the results obtained by the different approaches. The y-axis is labeled with operations per second. The vector addition is built by three operations, two additions and one assignment. Figure 8 shows the high runtime performance of an Intel P4 microprocessor as long as the values are cached.

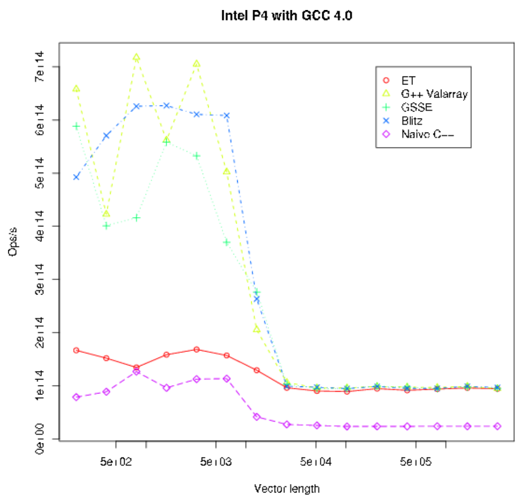


Figure 8: Comparison of functional specification on an Intel P4 microprocessor.

Figure 9 presents the results for an Intel Core Duo microprocessor. The sudden drop in runtime performance at certain vector lengths should be noted.

Figure 10 depicts the best benchmark results for an AMD Athlon 64 microprocessor with AMD rating 3500+.

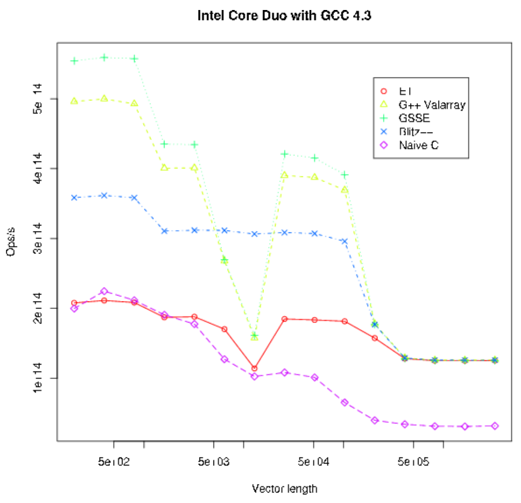


Figure 9: Comparison of functional specification on an Intel Core Duo microprocessor.

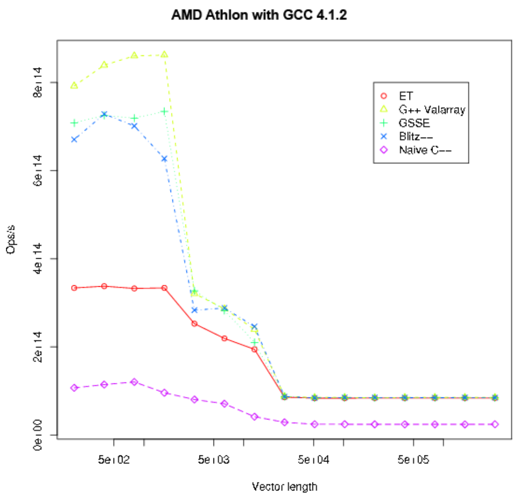


Figure 10: Comparison of functional specification on an AMD Athlon 64 microprocessor.

CONCLUSION

Based on a set of Bash shell scripts, a statistical analysis tool, and a comprehensive graphical user interface, a performance test platform was developed to ease the benchmarking processes. A completely automated process chain is thereby obtained to analyze the performance optimization possibilities of modern microprocessor architectures and new compiler versions.

REFERENCES

[1] *Pluggable Authentication Module (PAM)*, 1995. `http://www.kernel.org/pub/linux/libs/pam/`.

[2] *Network Information System (NIS)*, 2004. RFC 3898.

[3] *Kerberos: The Network Authentication Protocol*, 2007. `http://web.mit.edu/kerberos/www/`.

[4] D. Abrahams and A. Gurtovoy. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond (C++ in Depth Series).* Addison-Wesley Professional.

[5] Achour, Betz, Dovgal, Lopes, Olson, Richter, Seguy, Vrana, et al. PHP Manual. http://www.php.net/manual/en/, 2006.

[6] Boost Phoenix 2. *Boost Phoenix 2.* Boost, 2006. http://spirit.sourceforge.net/.

[7] R Foundation. The R Manuals. http://cran.r-project.org/manuals.html, 2007.

[8] R. Heinzl, P. Schwaha, M. Spevak, and T. Grasser. Performance Aspects of a DSEL for Scientific Computing with C++. In *Proc. of the POOSC Conf.*, pages 37–41, Nantes, France, July 2006.

[9] R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser. A High Performance Generic Scientific Simulation Environment. In *Proc. of the PARA Conf.*, page 61, Umea, Sweden, June 2006.

[10] R. Heinzl, M. Spevak, P. Schwaha, and S. Serlberherr. Performance Analysis for High-Precision Interconnect Simulation. In *Proc. of the ESMC. Conf.*, pages 113–116, Toulouse, France, 23-25. October 2006.

[11] A. Heuer and G. Saake. *Datenbanken: Konzepte und Sprachen.* mitp, 2nd edition, 2000.

[12] M. Cooper. Advaced Bash Scripting Guide. http://www.tldp.org/LDP/abs/html/, 2006.

[13] MySQL AB. MySQL 5.1 Reference Manual. http://dev.mysql.com/doc/refman/5.1/en/index.html, 2006.

[14] P. Schwaha, R. Heinzl, M. Spevak, and T. Grasser. Advanced Equation Processing for TCAD. In *Proc. of the PARA Conf.*, page 64, Umea, Sweden, June 2006.

[15] H. Tuuri et al. The InnoDB Documentation. http://www.innodb.com/support/documentation, 2007.

[16] T. L. Veldhuizen. *Active Libraries: Rethinking the Roles of Compilers and Libraries, in* Proc.ISCOPE'98, Lecture Notes in Computer Science. Springer-Verlag, 1998.

BIOGRAPHIES

RENÉ HEINZL studied electrical engineering at the Technische Universität Wien. He joined the Institute for Microelectronics in November 2003, where he finished his doctoral degree in September 2007. In April 2005 he achieved first place at the doctoral competition at the EEICT in Brno. His research interests include computational science, programming paradigms, high performance programming techniques, process simulation, solid modeling, scientific visualization, and algebraic topology for TCAD.
GEORG MACH was born in Vienna, Austria, in 1979. He studies electrical engineering at the Technische Universität Wien. He joined the Institut für Mikroelektronik in June 2003, where he is currently working on his diploma thesis.
PHILIPP SCHWAHA studied electrical engineering at the Technische Universität Wien. He joined the Institute for Microelectronics in June 2004, where he is currently working on his doctoral degree. His research activities include circuit and device simulation, device modeling, and software development.
SIEGFRIED SELBERHERR received the doctoral degree in technical sciences from the Technische Universität Wien in 1981. Since that time he has been with the Technische Universität Wien as professor. Dr. Selberherr has been holding the "venia docendi" on "Computer-Aided Design" since 1984. As of 1988 he has been chair professor of the Institut für Mikroelektronik. From 1998 to 2005 he served as Dean of the "Fakultät für Elektrotechnik und Informationstechnik" at the Technische Universität Wien. His current topics of interest are modeling and simulation of problems for microelectronics engineering.