Annual Report of the Chair of Computer Science 2 (Programming Systems)

Address: Martensstr. 3, 91058 Erlangen
Phone: +49-9131-85-27621
Fax: +49-9131-85-28809
E-Mail: info@i2.informatik.uni-erlangen.de

Ordinarius:
Prof. Dr. Michael Philippsen
Honorary Professor:
Hon.-Prof. Dr.-Ing. Bernd Hindel
Hon.-Prof. Dr.-Ing. Detlef Kips
Professor Emeritus:
Prof. em. Dr. Hans Jürgen Schneider
Secretary:
Margit Zenk
Scientific Staff:
Michael Baer, M. Sc. (since February 01, 2016)
Dipl.-Inf. Thorsten Blaß
Dipl.-Inf. Daniel Brinkers (until April 30, 2016)
Dipl.-Inf. Georg Dotzler (until January 31, 2016)
Marius Kamp, M. Sc.
Dipl.-Math. Jakob Krainz
Patrick Kreutzer, M. Sc.
Andreas Kumlehn, M. Sc. (until October 31, 2016)
Dr.-Ing. Christopher Mutschler
Dipl.-Inf. Daniela Novac
Dr.-Ing. Norbert Oster, Akad. ORat
Dipl.-Inf. Tobias Werth (until January 31, 2016)
IT-support:
Dipl.-Ing. (FH) Helmut Allendorf
Manfred Uebler
Guest:
Dr.-Ing. Josef Adersberger
Dr.-Ing. Martin Jung
Florian Lautenschlager, M. Sc.
Norbert Tausch, M. Eng.
External Teaching Staff:
Dr.-Ing. Klaudia Dussa-Zieger
Dr.-Ing. Martin Jung

Since 2002 Prof. Michael Philippsen is heading the Chair of Computer Science 2 which was founded in 1972. The group's main focus is on **Programming Systems for heterogeneous Multicores**.

Up to 2014/2015, we mainly took a system-level perspective and worked on the interface to both the parallel hardware and the OS, on runtime system issues, and on how to best express parallelism in program code. We strived to best harvest the potential of parallelism that is slumbering within applications and to find the most efficient ways to map it to the parallelism provided by the hardware. As such systems-oriented topics continue to be relevant, some of our projects continue to address them.

Nowadays the Programming Systems Group mainly finds answers for professional software engineers who develop industry-sized parallel programs for multicores, for distributed networks of multicores, for parallel cloud computing, and for networks of embedded systems.

1 Focus of research

A few corner stones stake out our field of **programming system research**: (a) We develop and evaluate uniform **programming models** for heterogeneous systems. To make the investment worthwhile, modules of parallel software need homogeneous interfaces to gain portability across changing and heterogeneous configurations of multicores, GPUs, accelerators, FPGAs etc. (b) There is growing commercial relevance in migrating legacy software to run on multicores. We investigate novel tools that work on **source code repositories**, analyze them, and help developers to **migrate, refactor and parallelize** software. In special cases, there are effective ways to even perform this task automatically. (c) Since parallelism and synchronization issues make parallel software more complex, software engineers need more support in their development and maintenance tasks. Only with such tool support they can avoid, detect and fix mistakes early on. And only with such tool support these tasks do no longer need the specialized expertise of low-level systems programmers. We thus develop **fast, interactive, and incremental code analyses** (that themselves may execute in parallel) that not only detect race conditions, conflicting accesses to resources, etc., but that also provide support and suggestions to developers while they are working in their codes with their IDEs. (d) During the life cycle of parallel software, the nondeterministic behavior of concurrency poses new challenges for **testing of parallel code**, for the assurance of quality and **code-authenticity**, as well as for operation and **diagnosis**. We explore, when parallel code is tested thoroughly enough, how to guard parallel code against attacks, how to generate sets of test data for parallel programs, how to systematically track down the reasons of erratic behavior of parallel codes, etc. Established techniques from sequential software engineering cannot yet deal with the parallelism in the code that adds a new dimension to these problems.

We follow a **program-code-centric approach**, we build operational **prototypes** for the tools that we conceive, and we carry forward our principle of a thorough **qualitative and quantitative evaluation** of our ideas.

2 Research projects

2.1 Analysis of Code Repositories

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Georg Dotzler
Marius Kamp, M. Sc.
Patrick Kreutzer, M. Sc.
Start: 1.1.2010

Software developers often modify their projects in a similar or repetitive way. The reasons for these changes include the adoption of a changed interface to a library, the correction of mistakes in functionally similar components, or the parallelization of sequential parts of a program. If developers have to perform the necessary changes on their own, the modifications can easily introduce errors, for example due to a missed change location. Therefore, an automatic technique is desireable that identifies similar changes and uses this knowledge to support developers with further modifications.

Extraction of Code-Changes

This led us to the development of a new migration and refactoring tool SIFE. The foundation of our tool lies in the comparison of two versions of the same program. It extracts the changes from a software repository that were made between two source code versions, derives generalized patterns of suboptimal and better code from these changes, and saves the patterns in a database. Our tool then uses these patterns to suggest similar changes for the source code of different programs automatically.

The extraction of code changes is based on trees. In 2016 we developed (and visibly published) a new tree-based algorithm (MTDIFF) that improves the accuracy of the change extraction.

Symbolic Execution of Code-Fragments

In 2014 we developed a new symbolic code execution engine called SYFEX to determine the behavioral similarity of two code fragments. In this way we aim to improve the quality of the recommendations. Depending on the number and the generality of the patterns in the database, it is possible that without the new engine SIFE generates some unfitting recommendations. To present only the fitting recommendations to the developers, we compare the summary of the semantics/behavior of the recommendation with summary of the semantics/behavior of the database pattern. If both differ too severely, our tool drops the recommendation from the results. The distinctive features of SYFEX are its applicability to isolated code fragments and its automatic configuration that does not require any human interaction.

In 2015 SYFEX was refined and applied to code fragments from the repositories of different software projects. In 2016 we investigated to which extend SYFEX can be used to gauge the semantic similarity of submissions for a programming contest.

Detection of Semantically Similar Code Fragments

SYFEX computes the semantic similarity of two code fragments. Therefore, it allows to identify pairs or groups of semantically similar code fragments (semantic clones). However, the high runtime of SYFEX (and similar tools) limit their applicability to larger software projects. In 2016, we started the development of a technique to accelerate the detection of semantically similar code fragments. The technique is based on so-called base comparators that compare two code fragments using a single criterion (e.g., the number of used control structures or the structure of the control flow graph) and that have a low runtime. These base comparators can be combined to form a hierarchy of comparators. To compute the semantic similarity of two code fragments as accurately as possible, we use genetic programming to search for hierarchies that approximate the similarity values as reported by SYFEX for a number of pairs of code fragments. A prototype implementation confirmed that the method is capable of detecting pairs of semantically similar code fragments.

Clustering of Similar Code-Changes

To create generalized change patterns, it is necessary that the set of extracted code changes is split into subsets of changes that are similar to each other. In 2015 this detection of similar code changes was improved and resulted in a new tool called C3. We developed and evaluated different metrics for a pairwise similarity comparison of the extracted code changes. Subsequently, we evaluated different clustering algorithms known from the literature and implemented new heuristics to automatically choose the respective parameters to replace the previous naive approach for the detection of similar code changes. This clearly improved the results compared to the previous approach, i.e., C3's new techniques detect more groups of similar changes that can be processed by SIFE to generate recommendations.

The aim of the second improvement is to automatically refine the resulting groups of similar code changes. For this purpose we evaluated several machine learning algorithms for outlier detection to remove those code changes that have been spuriously assigned to a group.

In 2016 we implemented a new similarity metric for the comparison of two code changes that essentially considers the textual difference between the changes (as generated, for example, by the Unix tool 'diff'). We published both a paper on C3 and the dataset (consisting of groups of similar changes) that we generated for the evaluation of our tool under an open-source license, see https://github.com/FAU-Inf2/cthree . This dataset can be used as a reference or as input data for future research. In addition, we prototypically extended C3 by techniques for an incremental similarity computation and clustering. This allows us to reuse results from previous runs and to only perform the absolutely necessary work whenever new code changes are added to a software archive.

2.2 Automatic Detection of Raceconditions

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Michael Baer, M. Sc.
Start: 1.1.2016
Contact:
Michael Baer, M. Sc.
Phone: +49-9131-85-27622
Fax: +49-9131-85-28809
E-Mail: michael.baer@fau.de

Large software projects built by hundreds of developers are difficult to test and contain many bugs. Automatic tests are a well established technique. They test the whole project (system test) or each module by itself (unit test) and can reliably detect and reproduce bugs at least for sequential and deterministic software. However, recent software contains more and more parallelism that introduces several new bug types, like deadlocks and concurrent memory accesses, that are harder or even impossible to detect by conventional test methods. Whether or not a faulty behavior can be noticed depends on the concrete scheduling of the threads which is indeterministic, varies between individual executions, and depends on the underlaying system. Due to this unpredictable behavior such bugs do not necessarily manifest in an arbitrary test run or may never arise in the testing environment at all. As a result, conventional tests are not well suited for modern, concurrent software.

The AuDeRace project develops methods to efficiently and reliably detect concurrent bugs while keeping the additional effort for developers low possible. In an initial approach we add a specification of a scheduling plan to a test case so that during test execution we regain determinism. Another major problem still remains open: The developer has to identify and implement well suited test cases that hold the potential fault and to execute them in a special deterministic way in order to trigger the failure. Especially in the context of concurrency, it is difficult to imagine the behavior of a program and to identify the problematic parts. To overcome this, the critical parts shall automatically be narrowed down before even writing dedicated test cases. Existing approaches and tools for this purpose generate too many false positives or the analysis is very time consuming, making their application to real world code prohibitive. The goal of this project is to generate fewer false positives and to increase the speed of the analysis by combining existing static and dynamic analysis. This allows for the efficient use not only in small example codes but also in large and complex software projects.

In 2016 existing approaches were studied regarding their usability as a starting basis for our project. The most promising method uses model checking and predefined assertions to construct thread schedules that trigger the faulty behavior. However, the approach is currently infeasible for larger projects because it can analyze only very small codes in reasonable time. Currently we focus on automatically detecting and removing statements that are unrelated to the concurrent, potentially faulty code parts in order to decrease the execution time of the preliminary static analysis.

2.3 Design for Diagnosability

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Andreas Kumlehn, M. Sc.
Dr.-Ing. Josef Adersberger
Florian Lautenschlager, M. Sc.
Duration: 15.5.2013–31.7.2016
Sponsored by:
IuK Bayern
Contact:
Prof. Dr. Michael Philippsen
Phone: +49-9131-85-27625
Fax: +49-9131-85-28809
E-Mail: michael.philippsen@fau.de

Many software systems behave obtrusively during the test phase or even in normal operation. The diagnosis and the therapy of such runtime anomalies is often time consuming and complex, up to being impossible. There are several possible consequences for using the software system: long response times, inexplicable behaviors, and crashes. The longer the consequences remain unresolved, the higher is the accumulated economic damage.

”Design for Diagnosability” is a tool chain targeted towards increasing the diagnosability of software systems. By using the tool chain that consists of modeling languages, components, and tools, runtime anomalies can easily be identified and solved, ideally already while developing the software system. Our cooperation partner QAware GmbH provides a tool called Software EKG that enables developers to explore runtime metrics of software systems by visualizing them as time series.

The research project Design for Diagnosability enhances the eco-system of the existing Software EKG. The Software-Blackbox measures technical and functional runtime values of a software system in a minimally intrusive way. We store the measured values as time series in a newly developed time series database, called Chronix. Chronix is an extremely efficient storage of time series that optimizes disk space as well as response times. Chronix is an open source project (www.chronix.io) and is free to use for everyone.

The newly developed Time-Series-API analyzes these values, e.g., by means of an outlier detection mechanism. The Time-Series-API provides multiple additional building blocks to implement further strategies for identifying runtime anomalies.

The mentioned tools in combination with the existing Software EKG will become the so-called Dynamic Analysis Workbench. This tool enables developers to diagnose, explain, and fix any occurring runtime anomalies both quickly and reliably. It will provide diagnosis plans to localize and identify the root causes of runtime anomalies.

The full tool chain aims at increasing the quality of software systems, particularly with respect to the metrics mean-time-to-repair and mean-time-between-defects.

Before we have successfully completed the project in July 2016, we have made the following contributions:

- We have linked Chronix and a framework for distributed data processing so that our anomaly analyses now scale to huge sets of time series data.
- We extended Chronix with additional components. Among them are, for example, a more efficient storage model, some adapters for more time series databases, additional server-side analysis functions, and some new time series types.
- We have published our benchmark for time series databases.

- We have investigated and implemented an approach to link application-level calls, e.g., a login of a user, down to the resulting calls on the OS level.

2.4 Adaptive Algorithms for RF-based Locating Systems

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Christopher Mutschler
Duration: 15.5.2016–14.5.2018
Sponsored by:
Fraunhofer Institut für Integrierte Schaltungen
Contact:
Prof. Dr. Michael Philippsen
Phone: +49-9131-85-27625
Fax: +49-9131-85-28809
E-Mail: michael.philippsen@fau.de

The goal of this project is the development of adaptive algorithms for radio-based realtime locating systems. In the scope of this project we cover three essential topics:

Automatic configuration of event detectors. In previous research projects we built the basics for the analysis of noisy sensor data streams. However, event detectors still need to be parameterized carefully to yield satisfying results. This work package explores the possibilities of an automatic configuration of the event detectors on existing sensor and event data streams. In 2016 we investigated concepts to extract optimal configurations from available sensor data streams. For soccer sport scientists manually annotated matches and scenes (e.g. player A kicks the ball with his/her left foot at time t). These manually annotated scenes may later by used to optimize the hierarchy of event detectors.

Evaluation of machine learning techniques for locating applications. In previous research projects we already developed machine learning algorithms for radio-based locating systems (e.g., evolutionary algorithms to estimate antenna positions and orientations). This work package investigates further approaches that use machine learning to enhance the performance of realtime locating systems. In 2016 we evaluated concepts to replace parts of the position estimation algorithms by machine learning algorithms. Up to now a signal processing chain (analog/digital conversion, time of arrival estimation, Kalman filtering, motion estimation) uses the raw sensor data to calculate a position. This often results in high installation and configuration costs for the setup of locating systems in the application environment.

Evaluation of vision-based techniques to support radio-based realtime locating. Radio-based locating systems have strengths if objects are occluded as microwaves may pass through the occluding objects. However, metallic surfaces in the environment pose challenges as they reflect RF-signals. Hence, the RF-signal that a transmitter emits arrives at the antennas over multiple paths. It is then often difficult to extract the directly received parts of the signal at the antenna and hence it is a challenge to properly estimate the distance between the antenna and the emitter. In this work package we investigate vision-based locating techniques that may help RF-based systems in calculating positions. In 2016 we developed two systems: CNNLok may be used by objects carrying a camera (self-localization), i.e., inside-out tracking, whereas InfraLok uses cameras installed in the environment to track objects with infrared light. CNNLok uses a convolutional neural network (CNN) that is trained on several camera images taken in the environment (at known places). At runtime the CNN receives a camera image and calculates the position of the camera. InfraLok detects infrared LEDs using a multi-camera system and calculated the position of objects in space.

2.5 Incremental Code Analysis

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Math. Jakob Krainz
PD Dr. Ronald Veldema
Duration: 1.4.2012–30.6.2017

To ensure that errors in a program design are caught early in the development process, it is useful to detect mistakes already during the editing of the code. For that the employed analysis has to be fast enough to enable interactive use. One method to achieve this is the use of incremental analysis, which combines analysis results of parts of the program to analyze the whole program. As an advantage, it is then possible to re-use large parts of the analysis results when a small change to the program occurs, namely for the unaffected parts of the program and for libraries. Thus the work required for analysis can be drastically reduced, which makes the analysis useful for interactive use.

Our analysis is based on determining, for (parts of) functions, which effects their execution can have on the state of a program at runtime. To achieve this, the runtime state of a program is modeled as a graph, which describes the variables and objects in the program's memory and their points-to relationship. The function is executed symbolically, to determine the changes made to the graph or, equivalently, to the runtime state described by it. To determine the effects of executing pieces of code in order, function calls, loops, etc., the change descriptions for smaller parts of the program can be combined in various ways, resulting in descriptions for the behavior of larger parts of the program. The analysis works in a bottom-up fashion, analyzing a called method before analyzing the callee (with recursion being analyzable as well).

In 2016 we focused on improving the algorithms and data structures used for the analysis. We improved both the scalability of the analysis towards large code bases with more than 1 mio. statements, and the incremental analysis, where we re-used the analysis results for unmodified program parts, drastically speeding up the analysis for typical software projects (i.e. with a large code base and small, incremental changes).

2.6 International Collegiate Programming Contest at the FAU

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Daniela Novac
Michael Baer, M. Sc.
Dipl.-Math. Jakob Krainz
Dipl.-Inf. Tobias Werth
Start: 1.11.2002
Contact:
Dipl.-Inf. Daniela Novac
Phone: +49-9131-85-27622
Fax: +49-9131-85-28809
E-Mail: daniela.novac@fau.de

The Association for Computing Machinery (ACM) has been hosting the International Collegiate Programming Contest (ICPC) for many years. Teams of three students try to solve nine to eleven programming problems within five hours. What makes this task even harder, is that there is only one computer available per team. The problems demand for solid knowledge of algorithms from all areas of computer science and mathematics, e.g. graphs, combinatorics, strings, algebra, and geometry.

The ICPC consists of three rounds. First, each participating university hosts a local contest to find the up to three teams that are afterwards competing in one of the various regional contests. Germany lies in the catchment area of the Northwestern European Regional Contest (NWERC) with competing teams from e.g. Great Britain, Benelux, and Scandinavia. The winners of all regionals in the world (and some second place holders) advance to the world finals in spring of the following year.

In 2016 two local contests took place in Erlangen. In the winter semester we conducted a team contest with teams consisting of at most three students. The main goal of this contest was to interest new students in the contests. We had 35 FAU teams plus 26 more teams from universities all over Europe. Before the second contest, as in the previous years, in the summer term the seminar ”Hello World - Programming for the Advanced” served to prepare students from different disciplines in algorithms and contest problems. In the German-wide contest of the summer term we selected the students that would represent the FAU at the NWERC 2016 in Bath (UK). 21 teams with students of computer science, computational engineering, mathematics as well as informations and communication technology took the challenge.

We formed the NWERC teams out of the best participants in the qualifications, given the age restrictions. Our teams reached places 20, 45 and 59 out of the 114 teams participating to NWERC in Bath, UK. These results are encouraging for the future, given the fact that the teams consisted of first time participants, with no previous competition experience.

2.7 Parallel code analysis on a GPU

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Thorsten Blaß
PD Dr. Ronald Veldema
Start: 1.7.2013

In compiler construction there are analyses that propagate information along the edges of a graph and modify it, until they reach a fix point.

In this project we build a framework to accelerate such analyses by exploiting the massive parallelism of graphic cards.

In 2016, our research focus was on **synchronization mechanisms for GPUs**. Known synchronization methods for CPUs (e. g. a spin lock) cannot be used without further adjustment on GPUs since their special architectural properties easily lead to dead- and livelocks. Synchronization is required (even for predominantly dataparallel graph implementations) if data dependences occur dynamically. We have therefore developed a novel synchronization mechanism which solves two non-trivial problems related to GPUs: First, we prevent dead- and livelocks. Second, we retain as much parallelism as possible by allowing dataparallel threads to work on disjoint areas of a data structure concurrently. For example, think of threads that modify disjoint locations of a graph without affecting its structural integrity. In our approach, a programmer can provide rules that describe the conditions under which a parallel access is allowed. At runtime, we check these rules and determine how many threads can run in parallel.

We currently extend the above synchronisation mechanism with a scheduler that re-distributes conflicting data access so that the SIMD execution on a GPU causes less serialization than without the re-ordering. Hence, the degree of parallelism grows. The underlying idea exploits that GPUs organize threads in hierarchical units. If the above synchronization mechanism detects a conflicting access in one of these units, it checks on the next smaller unit whether the conflict can also be found there. If this is not the case, the fewer threads in that unit can run in parallel. This is much better than serializing all threads in the enclosing unit. In this situation it is the scheduler's task to re-distribute the detected collisions across the units so that as many threads as possible can run in parallel. As the scheduling is performed at run-time it needs to be efficient, must itself run in parallel, and potentially make use of the dynamic thread creation capabilities of modern GPUs.

2.8 Software Watermarking

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Daniela Novac
Start: 1.1.2016

Software watermarking means hiding selected features in code, in order to identify it or prove its authenticity. This is useful for fighting software piracy, but also for checking the correct distribution of open-source software (like for instance projects under the GNU license). The previously proposed methods assume that the watermark can be introduced at the time of software development, and require the understanding and input of the author for the embedding process. The goal of our research is the development of a watermarking framework that automates this process by introducing the watermark while compiling the code and can also be used for existing code. As a first approach we study a method that is based on symbolic execution and function synthesis.

3 Publications 2016

– Dotzler, Georg ; Philippsen, Michael: Move-Optimized Source Code Tree Differencing . In: ACM (Ed.) : Proceedings of the 31st IEEE/ACM International Con-

ference on Automated Software Engineering (ASE 2016) (31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016) Singapore 03.-07.09.2016). New York, NY, USA : ACM, 2016, pp 660-671. - ISBN 978-1-4503-3845-5

- Edelhäußer, Thorsten ; Frühauf, Hans Holm ; Philippsen, Michael ; Kókai, Gabriella ; Nilson, Jörg: Concept for encoding data defining coded positions representing a trajectory of an object . Schutzrecht US 9.384.387 B2 examined granted patent (05.07.2016)

- Gradl, Stefan ; Eskofier, Björn ; Eskofier, Dominic ; Mutschler, Christopher ; Otto, Stephan: Virtual and augmented reality in sports: an overview and acceptance study . In: ACM (Ed.) : Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16): Adjunct (2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16) Heidelberg, Germany September 12 - 16, 2016). 2016, pp 885-888. - ISBN 978-1-4503-4462-3

- Kreutzer, Patrick ; Dotzler, Georg ; Ring, Matthias ; Eskofier, Björn ; Philippsen, Michael: Automatic clustering of code changes . In: ACM (Ed.) : Proceedings of the 13th International Workshop on Mining Software Repositories (MSR 2016) (13th International Workshop on Mining Software Repositories Austin, Texas 14.-15. Mai 2016). New York, USA : ACM, 2016, pp 61-72. - ISBN 978-1-4503-4186-8

- Lautenschlager, Florian: Chronix - A fast and efficient time series storage based on Apache Solr .Talk: Open Source Data Center Conference (OSDC 2016), Berlin, Germany, 2016

- Lautenschlager, Florian ; Kammerer, Moritz: Chronix as Long Term Storage for Prometheus .Talk: CloudNativeCon, Seattle, WA, 08.11.2016

- Lautenschlager, Florian: The new time series kid on the block .Talk: Apache Con: Big Data, Vancouver, Canada, 2016

- Mohammad Alawieh ; Niels Hadaschik ; Norbert Franke ; Mutschler, Christopher: Inter-Satellite Ranging in the Low Earth Orbit . In: IEEE (Ed.) : Proceedings of the 10th IEEE/IET International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP'16) (10th IEEE/IET International Symposium on Communication Systems, Networks & Digital Signal Processing Prague, Czech Republic 20.-22.07.2016). 2016, pp 1-6. - ISBN 978-1-5090-2526-8

- Mutschler, Christopher ; Feigl, Tobias ; Daxer, Christian ; Otto, Stephan ; Bercea, Cosmin-Ionut: Verfahren zum Einstellen einer Blickrichtung in einer Darstellung einer virtuellen Realität . Schutzrecht DE 10 2016 109 153.7 patent application (18.05.2016)
- Sackenreuter, Benjamin ; Hadaschik, Nils ; Fassbinder, Marc ; Mutschler, Christopher: Low-Complexity PDoA-based Localization . In: IEEE (Ed.) : Proceedings of the 7th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2016) (7th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2016) Madrid, Spain 4.10.-7.10.2016). 2016, pp 1-6. - ISBN 978-1-5090-2425-4

4 Exam theses (german only)

- Bachelor Thesis: Automatische Generierung von Joern-Abfragen. Bearbeiter: Simon Rainer (beendet am 01.03.2016); Betreuer: Dipl.-Inf. Georg Dotzler; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen
- Bachelor Thesis: Konzeption und Implementierung eines Benchmarks für Zeitreihendatenbanken auf Basis operationaler Laufzeitdaten. Bearbeiter: Max Wehner (beendet am 08.04.2016); Betreuer: Prof. Dr. Michael Philippsen; Florian Lautenschlager, M. Sc.
- Bachelor Thesis: Verbesserte Erkennung von Laufzeitanomalien zur automatisierten Ursacheneingrenzung. Bearbeiter: Michael Wiesenbauer (beendet am 08.04.2016); Betreuer: Andreas Kumlehn, M. Sc.; Prof. Dr. Michael Philippsen
- Studien-/Bachelor-/Diplom-/Masterarbeit: Optimierung eines Verfahrens zur symbolischen Ausführung von Code-Fragmenten. Bearbeiter: Lukas Spranger (beendet am 19.04.2016); Betreuer: Dipl.-Inf. Georg Dotzler; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen
- Studien-/Bachelor-/Diplom-/Masterarbeit: Beschleunigung von Propagationsalgorithmen mittels Pfadkompression auf der GPU. Bearbeiter: Andreas Lindsteding (beendet am 22.04.2016); Betreuer: Dipl.-Inf. Thorsten Blaß; PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen
- Master Thesis: Automatische Optimierung der Granularität von rekursiven task-parallelen Programmen auf Vielkernrechnern. Bearbeiter: Daniel Schmidt (beendet am 02.05.2016); Betreuer: Prof. Dr. Michael Philippsen; Dipl.-Inf. Tobias Werth

- Bachelor Thesis: Erweiterung einer computergestützten Diagnose von Laufzeitanomalien. Bearbeiter: Moritz Müller (beendet am 04.05.2016); Betreuer: Andreas Kumlehn, M. Sc.; Prof. Dr. Michael Philippsen

- Master Thesis: Automatische Parallelisierung von rekursiven Programmen auf Vielkernrechnern mittels spekulativer Ausführung. Bearbeiter: Frederik Simon (beendet am 01.06.2016); Betreuer: Prof. Dr. Michael Philippsen; Dipl.-Inf. Tobias Werth

- Bachelor Thesis: Inkrementelles Clustering ähnlicher Quelltext-Änderungen. Bearbeiter: Martin Endrizzi (beendet am 08.06.2016); Betreuer: Patrick Kreutzer, M. Sc.; Dipl.-Inf. Georg Dotzler; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Entwicklung eines Simulink Targets. Bearbeiter: Thorsten Schwachhofer (beendet am 04.10.2016); Betreuer: Dipl.-Inf. Thorsten Blaß; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Erweiterung (und Beschreibung) eines parametrisierbaren skriptfähigen Web Interfaces. Bearbeiter: Markus Straussberger (beendet am 04.10.2016); Betreuer: Dipl.-Inf. Thorsten Blaß; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Implementierung des Kerns eines Konverters für grafische Sprachen von Speicherprogrammierbaren Steuerungen. Bearbeiter: Michael Sammler (beendet am 04.10.2016); Betreuer: Dipl.-Inf. Thorsten Blaß; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Implementierung und Evaluierung einer Hierarchie von Verfahren zum semantischen Vergleich von Code-Fragmenten. Bearbeiter: Adrian Kretschmer (beendet am 03.11.2016); Betreuer: Patrick Kreutzer, M. Sc.; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Entwicklung eines Benchmarks zur Bewertung von Zeitreihendatenbanken. Bearbeiter: Sebastian Thürauf (beendet am 30.11.2016); Betreuer: Florian Lautenschlager, M. Sc.; Prof. Dr. Michael Philippsen

- Master-Projekt: Ein E-Übersetzer in Rust. Bearbeiter: Sebastian Hahn (beendet am 14.12.2016); Betreuer: Dipl.-Math. Jakob Krainz; Dipl.-Inf. Tobias Werth; Prof. Dr. Michael Philippsen