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:
Dipl.-Inf. Thorsten Blaß
Dipl.-Inf. Daniel Brinkers
Dipl.-Inf. Georg Dotzler
Marius Kamp, M. Sc.
Dipl.-Math. Jakob Krainz
Patrick Kreutzer, M. Sc. (from December 01, 2015)
Andreas Kumlehn, M. Sc.
Dr.-Ing. Christopher Mutschler
Dr.-Ing. Norbert Oster
Norbert Tausch, M. Eng. (until September 30, 2015)
PD Dr. Ronald Veldema (until September 15, 2015)
Dipl.-Inf. Tobias Werth
IT-support:
Dipl.-Ing. (FH) Helmut Allendorf
Manfred Uebler
Guest:
Dr.-Ing. Josef Adersberger
Dr.-Ing. Martin Jung
Florian Lautenschlager, M. Sc.
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.

Over the last two years, the Programming Systems Group has achieved a major **thematic re-alignment**. We now find 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.

In our thematic re-alignment from a more systems-oriented focus towards the interface between programming systems and software engineering, we retain well-established lab practices. We keep our **program-code-centric approach**, we continue to 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.
PD Dr. Ronald Veldema
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 2015 we developed five optimizations for such tree-based differencing algorithms in order to improve 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.

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 within the context of a new tool C3. To this end 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 way, we achieved a clear improvement of the results compared to the previous approach, i.e., the new techniques implemented in C3 lead to the detection of 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.

The latest results of our tool SIFE are found at:
https://www2.cs.fau.de/staff/dotzler/SIFE_results.tar.gz

2.2 Automatic Code Parallelization at Runtime

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
PD Dr. Ronald Veldema
Dipl.-Inf. Daniel Brinkers
Duration: 1.1.2011–30.4.2016

Our automatic parallelization efforts are currently focused on dynamic parallelization. While a program is running, it is analyzed to find loops where parallelization can help performance. Our current idea is to run long-running loops three times. The first two runs analyze the memory accesses of the loop and can both run in parallel. The first run stores in a shared data structure for every memory address in which loop iteration a write access happens. We do not need any synchronization for this data structure, only the guarantee that one value is written to memory, when two concurrent writes happen. In the second pass we check for every memory access, if it has a dependency to one of the stored write accesses. A write access is part of any data-dependency, so we can find all types of data dependencies. If we do not find any, the loop is actually run in parallel. If we find dependencies, the loop is executed sequentially. We can execute the analyses in parallel to a modified sequential execution of the first loop iterations.

In 2014 we enhanced the analysis so that a loop can start running while the remainder of the loop is analyzed to see if it can be run in parallel. To allow the sequential loop to execute while the tail of the loop is analyzed we needed to instrument the sequential loop slightly. The result is that the loop runs only slightly slower if the loop cannot be parallelized, but if the loop is found to be parallelizable, speedup is near to linear.

Finally, we also created a new language that uses the above library for run-time parallelization. Any loops that the programmer marked as candidates for run-time parallelization are analyzed for constructs that the library cannot yet handle. If the loop is clean, code is generated that uses the library's macros.

The ErLaDeF project is a contribution of the Chair of Computer Science 2 (Programming Systems) to the IZ ESI (Embedded Systems Initiative, http://www.esi-anwendungszentrum.de/)

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.
Start: 15.5.2013
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.

Our main contributions of the year 2015 are:

- We have optimized Chronix and released it in an open source project (www.chronix.io).

- We have designed and implemented a novel benchmark for time series databases.
- We have investigated and implemented a novel approach to reduce the overhead of the Software-Blackbox. We now achieve a fixed memory footprint.
- Furthermore, the Dynamic Analysis Workbench now offers a set of detectors for anomalous time series and a plug-in mechanism to extend it with additional detectors.

2.4 Efficient Software Architectures for Distributed Event Processing Systems

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Christopher Mutschler
Duration: 15.11.2010–31.12.2015
Sponsored by:
Fraunhofer Institut für Integrierte Schaltungen

Localization Systems (also known as Realtime Location Systems, or RTLS) become more and more popular in industry sectors such as logistics, automation, and many more. These systems provide valuable information about whereabouts of objects at runtime. Therefore, processes can be traced, analyzed, and optimized. Besides the research activities at the core of localization systems (like resilience and interference-free location technologies or methods for highly accurate positioning), algorithms and techniques emerge that identify meaningful information for further processing steps. Our research focuses on automatic configuration methods for RTLSs as well as on the generation of dynamic motion models and techniques for event processing on position streams at runtime.

In 2011, we investigated whether events can be predicted after analyzing and learning event streams from the localization system at runtime. As a result, we are able to deduce models that represent the information buried in the event stream to predict future events.

We developed several methods and techniques in 2012 that process and detect events with low latency. Events (composite, complex) can be detected by means of a hierarchical aggregation of sub-events that themselves are detected by (several) event detectors processing sub-information in the event stream. This greatly reduces the complexity of the detection components and renders them fully maintainable. They even can use parallel or distributed cluster architectures more efficiently so that important events can be detected within a few milliseconds.

In 2013 we further minimized detection latency in distributed event-based systems: first, a new migration technique modifies and optimizes the allocation of software components in a networked environment at runtime to minimize networking overhead and detection latencies. Second, a speculative event processing technique uses conservative buffering techniques to exploit available system resources. We also created and published a representative data set (consisting of realtime position data and event streams) and a corresponding task description.

In 2014 we investigated fundamental approaches zu handle uncertainties (both w.r.t. the definition of event detectors and to the events). We implemented a promising prototype of an event-based system that is no longer deterministic but instead evaluates several possible system states in parallel to achieve a detection with a much higher robustness and correctness. The domain expert can parameterize the event detectors by attaching probabilities or probability functions to the generated events.

In 2015 we improved, optimized and published our approach. Furthermore we started to investigate approaches to learn optimal parameter sets for the event detectors. Thus, a manual adjustment and tuning of parameters (like thresholds) becomes unnecessary.

The project is a contribution of the Programming Systems Group to the [IZ ESI]http://www.esi.uni-erlangen.de/

2.5 Incremental Code Analysis

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

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 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 2015 we focused on improving the algorithms and data structures used for the analysis. We were able to significantly improve the runtime and memory requirements for analyzing a given program. Additionally, the analyzed program may now contain more, and more expressive language features.

2.6 Interactive and parallel code analysis

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

Not only 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. To achieve this objective the following sub- questions have been solved:

Efficient Graph implementations on the GPU

Compilers use graph structures to represent the program to be analyzed in a suitable way (e.g. abstract syntax tree, control flow graph, data flow graph). Analyses (e.g. alias analysis) are using graphs to store their results, too. We therefore investigated what kind of graph implementations allow an efficient use of the GPU. In the alias analysis multiple parallel threads insert/modify information in the graph. This results in performance degradation. In 2015 we reduced this performance degradation as far as possible. We implemented twelve different graph data structures and analyzed for which type of applications they best fit (frequent read access, many edges/nodes are added, nodes are often removed, ...). The data structures differ in how they store the nodes and edges and how they implement locking. Based on our results we can pick a suitable graph structure for a given problem. A publication is on the way.

Synchronization mechanisms on the GPU

In 2015 we also explored efficient synchronization mechanisms for the GPU. Known synchronization methods for the CPU (e. g. a spin lock) cannot be used without further adjustment on the GPU, due to special properties of the GPU architecture. A naive use of CPU synchronization mechanisms on the GPU results in deadlocks. Synchronization is frequently required for mutual exclusion, for example, for graph implementations. We have therefore developed a novel synchronization mechanism that firstly prevents deadlocks and secondly retains as much parallelism as possible (threads working on non-colliding points of a data structure are not blocked). For example, threads that modify disjoint locations of a graph, do not affect the structural integrity of the graph and can thus run in parallel. A programmer can provide rules that describe the conditions that allow this. At runtime, we check these rules and determine how many threads can run in parallel. A publication of this novel synchronization mechanism is work in progress.

2.7 International Collegiate Programming Contest at the FAU

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Tobias Werth
Dipl.-Inf. Daniel Brinkers
Dipl.-Math. Jakob Krainz
Start: 1.11.2002
Contact:
Dipl.-Inf. Tobias Werth
E-Mail: tobias.werth@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 2015 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 2015 in Linköping. 21 teams with students of computer science, computational engineering, mathematics as well as informations and communication technology took the challenge. We selected nine students for the NWERC, forming three teams.

At the NWERC in Linköping, the best FAU team finished tenth with seven solved problems. The other two FAU teams ranked 15th and 25th (out of 99 teams) with six and seven solved problems

In May The FAU Team, which qualified last year for the worldfinals with reaching the second place at NWERC 2014, ranked 58th out of 128 teams from all over the world at the world finals in Marrakesch, Marokko.

2.8 OpenMP/Java

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
PD Dr. Ronald Veldema
Dipl.-Inf. Georg Dotzler
Duration: 1.10.2009–1.10.2015

JaMP is an implementation of the well-known OpenMP standard adapted for Java. JaMP allows one to program, for example, a parallel for loop or a barrier without resorting to low-level thread programming. For example:

```
class Test {
...void foo(){
......//#omp parallel for
......for (int i=0;i<N;i++) {
.........a[i]= b[i]+ c[i]
......}
...}
}
```

is valid JaMP code. JaMP currently supports all of OpenMP 2.0 with partial support for 3.0 features, e.g., the collapse clause. JaMP generates pure Java 1.5 code that runs on every JVM. It also translates parallel for loops to CUDA-enabled graphics cards for extra speed gains. If a particular loop is not CUDA-able, it is translated to a threaded version that uses the cores of a typical multi-core machine. JaMP also supports the use of multiple machines and compute accelerators to solve a single problem. This is achieved by means of two abstraction layers. The lower layer provides abstract compute devices that wrap around the actual CUDA GPUs, OpenCL GPUs, or multicore CPUs, wherever they might be in a cluster. The upper layer provides partitioned and replicated arrays. A partitioned array automatically partitions itself over the abstract compute devices and takes the individual accelerator speeds into account to achieve an equitable distribution. The JaMP compiler applies code-analysis to decide which type of abstract array to use for a specific Java array in the user's program.

In 2015 we added OpenMP tasks (OpenMP 3.0) to JaMP. This makes it possible to parallelize recursive algorithms with JaMP.

2.9 Software Project Control Center

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Josef Adersberger
Norbert Tausch, M. Eng.
Duration: 1.11.2009–31.12.2015
Sponsored by:
Bundesministerium für Wirtschaft und Technologie
Contact:
Prof. Dr. Michael Philippsen
Phone: +49-9131-85-27625
Fax: +49-9131-85-28809
E-Mail: michael.philippsen@fau.de

Prototypical implementation of a new tool for quality assurance during software development

Modern software systems are growing increasingly complex with respect to functional, technical and organizational aspects. Thus, both the number of requirements per system and the degree of their interconnectivity constantly increase. Furthermore the technical parameters, e.g., for distribution and reliability are getting more complex and software is developed by teams that are not only spread around the globe but also suffer from increasing time pressure. Due to this, the functional, technical, and organizational control of software development projects is getting more difficult.

The ”Software Project Control Center” is a tool that helps the project leader, the software architect, the requirements engineer, or the head of development. Its purpose is to make all aspects of the development process transparent and thus to allow for better project control. To achieve transparence, the tool distills and gathers properties from all artifacts and correlations between them. It presents/visualizes this information in a way suitable for the individual needs of the users.

The Software Project Control Center unifies the access to relations between artifacts (traceability) and to their properties (metrics) within software development projects. Thus, their efficiency can be significantly increased. The artifacts, their relations, and related metrics are gathered and integrated in a central data store. This data can be analyzed and visualized, metrics can be computed, and rules can be checked.

For the Software Project Control Center project we cooperate with the QAware GmbH, Munich. The AIF ZIM program of the German Federal Ministry of Economics and Technology funded the first 30 months of the project.

The Software Project Control Center is divided into two subsystems. The integration pipeline gathers traceability data and metrics from a variety of software engineering tools. The analysis core allows to analyze the integrated data in a holistic way. Each subsystem is developed in a separate subproject.

The project partner QAware GmbH implemented the integration pipeline. The first step was to define TraceML, a modeling language for traceability information in conjunction with metrics. The language contains a meta-model and a model library. TraceML allows to define customized traceability models in an efficient way. The integration pipeline is realized using TraceML as lingua franca in all processing steps: From the extraction of traceability information to its transformation and integrated representation. We used the Eclipse Modeling Framework to define the TraceML models on each meta-model level. Furthermore, we used the Modeling Workflow Engine for model transformations and Eclipse CDO as our model repository. A set of wide-spread tools for software engineering are connected to the integration pipeline including Subversion, Eclipse, Jira, Enterprise Architect and Maven.

The main contribution of our group to this project is the analysis core, i.e., the design and implementation of a domain-specific language for graph-based traceability analysis. Our Traceability Query Language (TracQL) significantly reduces the effort that is necessary to implement traceability analyses. This is crucial for both industry and the research community as lack of expressiveness and inefficient runtimes of other known approaches used to hinder the implementation of traceability analysis. TracQL eases not only the extraction, but also the analysis of traceability data using graph traversals that are denoted in a concise functional programming style. The language itself is built on top of Scala, a multi-paradigm programming language, and was successfully applied to several real-world industrial projects.

In 2015, we evaluated and documented our approach in order to emphasize its core attributes and to show its effectiveness. The three core attributes are:

- Representation independence: TracQL is adaptable to various data sources at which their data types are available statically typed.
- Modularity: the approach is both modifiable and extendable in terms of structure and operations.
- Applicability: the language has a better expressiveness and performance than other approaches.

2.10 Compiler-supported parallelization for multi-core architectures

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Tobias Werth
Start: 1.1.2011
Contact:
Dipl.-Inf. Tobias Werth
E-Mail: tobias.werth@fau.de

Several issues significantly retard the development of quicker and more efficient computer architectures. Traditional technologies can no longer contribute to offer more hardware speed. Basic problems are the divergent ratio of the latencies of memory access and CPU speeds as well as the heat and waste of energy caused by increasing clock rates. Homogeneous and heterogeneous multi-core and many-core architectures were presented as a possible answer and offer enormous performance to the programmer. The multi-level cache hierarchy and decreased clock rates help avoid most of the above problems. Potentially, performance can increase even further by specialization of some hardware components. Current target architectures are GPUs with hundreds of arithmetic units and the Intel XeonPhi processor that provides 60 and more cores including hyper threading on a single board.

While data parallel problems can be relatively simple accelerated by using the new hardware architectures, the implementation of task parallel problems is our main research focus. The difficulty is often the irregularity of the resulting task tree and thus the different task run times. From the point of view of a programming systems research group, there are - among others - the following open questions: Which core executes which work packet in which order? When do you donate a work packet from one compute node to another? Which data belongs to a work packet, are multiple cores/compute nodes allowed to access the data simultaneously? How do we have to merge data from multiple compute nodes? How can a compiler together with a runtime system create tasks and distribute work packets?

In 2011, we have implemented and extended the Cilk programming model for the heterogeneous CellBE architecture (one PowerPC core (PPU) with eight SPU "coprocessors"). The CellBE architecture offers an enormous computing potential on a single chip. To move a work packet in the heterogeneous architecture, we have extended the Cilk programming model by an extra keyword. A source to source transformation then creates code for both, the PPU and SPU cores. Furthermore, we have moved the data along with the work packets in the SPU local stores and used a garbage collection technique to free memory from remote SPUs later.

In 2012, we focused on graphic cards (GPUs) as a second target architecture. GPUs offer a lot more performance than ordinary CPUs, however achieving peak performance may be difficult. For data parallel problems, the performance can be achieved using Cuda (NVidia) or OpenCL (AMD) relatively easy. However, it is much more difficult to port task parallel problems with reasonable performance to the GPU, which is one of the goals on our roadmap. Thus, we design, implement and compare various load balancing algorithms. In 2012 we designed a first approach with hierarchical queues under the principle of work donation.

In 2013, while further developing the load balancing algorithms for the GPU, we also targeted our work towards the Intel XeonPhi processor. With its many-core architecture and large register sets (and thus the ability to issue vector instructions on multiple data), the XeonPhi processor is a new challenge for load balancing algorithms. In practice, we extended and adopted Cilk for the XeonPhi such that we can automatically merge functions during the source-to-source transformation. This increases the Intel compiler's chances to automatically parallelize. We implemented several analyses that not only increase the number of candidate functions for merging but also avoid (or at least handle) divergence in those merged functions.

In 2014, we have extended our existing implementation for XeonPhi processors in a way that we can distribute the work over multiple XeonPhi processors. In contrast to the technique of work stealing that is used to distribute work over the many cores of a single XeonPhi, we use work donation to distribute the work to other XeonPhi processors. With a new source code annotation it is possible to mark the necessary data ranges for a work packet. These data ranges are then distributed along with the work packet and merged at synchronization points, which was the main challenge of the implementation. Furthermore, we have started to extend the clang compiler of the LLVM framework with support for Cilk in order to automatically generate CUDA code for execution on GPUs. Along with the generated CUDA code, we have designed a lightweight but general runtime system that manages execution and execution order of the work packets. We plan to implement analyses to avoid execution divergence as much as possible.

In 2015, we evaluated and compared multiple load balancing algorithms to execute Cilk programs on the GPU. Therefore, we implemented queuing algorithms for parallel access and improved the automated generation of the necessary CUDA code. The correct placement of Cilk keywords for synchronization is still a challenge for the programmer. Thus, we generate from plain, recursive C code multiple, "plausible" code variants including synchronisation statements. These code variants will then be executed speculatively and the result from the fastest, correct variant will be used for further computations. Furthermore, the size of the base case of the recursion is crucial for an optimal performance improvement. Consequently, we started to optimize the size of the base case using analysis during compile and run time and will replace recursive calls with function inlining and vectorisation.

3 Publications 2015

– Brinkers, Daniel ; Philippsen, Michael ; Veldema, Ronald: Simultaneous inspection: Hiding the overhead of inspector-executor style dynamic parallelization . In: Springer (Ed.) : Proceedings of the International Workshop on Languages and Compilers for Parallel Computing (LCPC 2014) (International Workshop on Languages and Compilers for Parallel Computing (LCPC 2014) Hillsboro, OR, USA 15.-17.09.2014). Berlin : Springer, 2015, pp 101-115. (Lecture Notes in Computer Science Vol. 8967) - ISBN 978-3-319-17472-3

– Lautenschlager, Florian: Apache Solr as a compressed, scalable and high performance time series database .Talk: Free and Open Source Software Developers' European Meeting (FOSDEM 2015), Brussels, Belgium, 31.01.2015

– Lautenschlager, Florian ; Kumlehn, Andreas ; Adersberger, Josef ; Philippsen, Michael: Rahmenwerk zur Ausreißererkennung in Zeitreihen von Software-Laufzeitdaten . In: GI (Ed.) : Tagungsband Software Engineering & Management (SE 2015) (Fachtagung Software Engineering & Management (SE 2015) Dresden, Deutschland 17.03.-20.03.2015). 2015, pp 177-182. - ISBN 978-3-88579-633-6

- Löffler, Christoffer ; Mutschler, Christopher ; Philippsen, Michael: Approximative Event Processing on Sensor Data Streams (Best Poster and Demostration Award) . In: ACM (Ed.) : Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (9th ACM International Conference on Distributed Event-Based Systems Oslo, Norway 29.06. - 03.07.2015). 2015, pp 360-363. - ISBN 978-1-4503-3286-6

- Schönwetter, Dominik ; Veldema, Ronald ; Fey, Dietmar: FREACSIM - A Framework for Creating and Simulating Real-Time Capable Network on Chip Systems and Applications . In: Theodoropoulos, Georgios ; Tan Soon Huat, Gary ; Szabo, Claudia (Ed.) : Proceedings of the Eighth EAI International Conference on Simulation Tools and Techniques (Eighth EAI International Conference on Simulation Tools and Techniques Athens, Greece 24.08. - 26.08.2015). European Union Digital Library : ACM, 2015. - ISBN 978-1-63190-079-2

4 Exam theses (german only)

- Master Thesis: Automatisierte Ursacheneingrenzung von Laufzeitanomalien durch dynamische Instrumentierung im Java Bytecode. Bearbeiter: Sven Marschke (beendet am 31.3.2015); Betreuer: Andreas Kumlehn, M. Sc.; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Developing a QoS Component to guarantee Operator Reliability in Distributed Event-Based Systems. Bearbeiter: Cosmin Bercea (beendet am 31.03.2015); Betreuer: Dr.-Ing. Christopher Mutschler; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Konfliktlösungsstrategie für Entwicklungsdatenmodelle. Bearbeiter: Rebekka Rupp (beendet am 01.07.2015); Betreuer: Hon.-Prof. Dr.-Ing. Detlef Kips; Dr.-Ing. Martin Jung

- Master Thesis: Entwurf und Implementierung einer Lastverteilung für taskparallele Programme auf Grafikkarten. Bearbeiter: Georg Altmann (beendet am 17.08.2015); Betreuer: Dipl.-Inf. Tobias Werth; Prof. Dr. Michael Philippsen

- Diplom-/Masterarbeit: Extern steuerbare Physiksimulation von Anlagen und Geräten. Bearbeiter: Martin Sturm (beendet am 26.08.2015); Betreuer: PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen

- Bachelor Thesis: Integration of Task-Parallelism into the JaMP-Framework. Bearbeiter: Benjamin Röder (beendet am 08.09.2015); Betreuer: Dipl.-Inf. Georg Dotzler; Prof. Dr. Michael Philippsen

- Master Thesis: Implementierung eines statischen Ablaufplaners ("Schedulers") innerhalb eines Übersetzers einer Sprache mit blockierenden Threads. Bearbeiter: Felix Sauer (beendet am 01.10.2015); Betreuer: PD Dr. Ronald Veldema; Dipl.-Inf. Thorsten Blaß; Prof. Dr. Michael Philippsen
- Studien-/Bachelor-/Diplom-/Masterarbeit: Untersuchung von Fehlertoleranz durch Software. Bearbeiter: Jean-Christopher Jäger (beendet am 01.10.2015); Betreuer: PD Dr. Ronald Veldema; Dipl.-Inf. Thorsten Blaß; Prof. Dr. Michael Philippsen
- Studien-/Bachelor-/Diplom-/Masterarbeit: Evaluation of Machine Learning Algorithms for Outlier Detection in Clustered Code Fragments. Bearbeiter: James Wafula (beendet am 30.10.2015); Betreuer: Dipl.-Inf. Georg Dotzler; Matthias Ring, M. Sc.; Prof. Dr. Björn Eskofier; Prof. Dr. Michael Philippsen
- Studien-/Bachelor-/Diplom-/Masterarbeit: Clustering von ähnlichen Code-Fragmenten. Bearbeiter: Patrick Kreutzer (beendet am 11.11.2015); Betreuer: Dipl.-Inf. Georg Dotzler; Matthias Ring, M. Sc.; Prof. Dr. Björn Eskofier; Prof. Dr. Michael Philippsen
- Studien-/Bachelor-/Diplom-/Masterarbeit: Verbesserung von Mutationstests mit Hilfe der symbolischen Ausführung. Bearbeiter: Michael Baer (beendet am 16.12.2015); Betreuer: Dipl.-Inf. Georg Dotzler; Marius Kamp, M. Sc.; Dr.-Ing. Norbert Oster; Prof. Dr. Michael Philippsen
- Master Thesis: Werkzeug zur computergestützten Diagnose von Laufzeitanomalien in Zeitreihen operationaler Daten. Bearbeiter: Stefan Ettl (beendet am 22.12.2015); Betreuer: Prof. Dr. Michael Philippsen; Andreas Kumlehn, M. Sc.