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. (from January 01, 2015)
Demian Kellermann, M. Sc. (until April 30, 2014)
Dr.-Ing. Stefan Kempf (until September 30, 2014)
Dipl.-Math. Jakob Krainz
Andreas Kumlehn, M. Sc.
Dr.-Ing. Christopher Mutschler
Dr.-Ing. Norbert Oster
Norbert Tausch, M. Eng.
PD Dr. Ronald Veldema
Dipl.-Inf. Tobias Werth
IT-support:
Dipl.-Ing. (FH) Helmut Allendorf
Manfred Uebler
Guest:
Dr.-Ing. Josef Adersberger
Dipl.-Inf. Samir Al-Hilank
Dipl.-Inf. Ralf Ellner
Dr.-Ing. Martin Jung
Florian Lautenschlager, M. Sc.
Dr.-Ing. Stephan Otto
Dipl.-Inf. Mykola Protsenko

External Teaching Staff:
Dr.-Ing. Klaudia Dussa-Zieger
Dr.-Ing. Martin Jung
Dr.-Ing. Stephan Otto

The Chair of Computer Science 2 (programming systems) was founded in 1972 and is headed by Prof. Michael Philippsen (as the successor of Prof. H.-J. Schneider) since April 2002. Closely associated with the programming systems group is the professorship for Didactics of Computer Science.

1 Focus of research

The main research topics in the programming systems group are programming of parallel or distributed systems and programming of embedded or mobile systems. Software (and its development) for such systems should ideally be as complex, portable, maintainable and robust as existing software for single core systems and workstations. It is our long-term goal to allow applications to take full advantage of the available computing and network power. A particular focus lies on programming systems for multi-cores because more and more cheap multi-core high-performance parallel hardware (for example graphics cards) is available. This will have an unpredictable impact on the future of the software landscape. Research results of the group are always evaluated by means of prototypes and demonstrators.

Important Research Areas

- **Exploit the available parallelization potential.** In the future the clock rate of multi-core systems will grow only slowly whereas the number of cores will grow. This makes it necessary to exploit the parallelization potential of already older, existing software to allow it to benefit from the new hardware. As a consequence, in most application areas a change to parallel computing is unavoidable. Therefore, the programming systems group develops tools to support the programmer interactively in reengineering existing sequential applications. It also develops architectural patterns for new software projects that scale automatically to support a growing number of cores.

- **Achieve portability in high-performance applications.** Up to the present, application programmers achieve the best possible performance results only if they handle latency issues and communications between different components of the system manually, optimize their code with hardware specific ”tricks” and split

their application into multiple sections to outsource them to other hardware (for example graphics cards). To change this situation, the programming systems group researches the performance impact of higher programming abstraction layers that would improve programming productivity and software portability. The improvements are caused by generated code that allows the distribution of the program onto multiple heterogeneous system components to permit parallel execution. The higher abstraction layer makes the communication between the components transparent for the developer. To increase the efficiency of this approach it is necessary to give the programmer the possibility to express available domain knowledge in the programming language. For the higher abstraction layer, the details of the hardware architecture are hidden from the developer (for example by library functions or programming language extensions).

- **Adapt the degree of parallelism dynamically**. High-performance applications are often developed for a fixed number of cores. As requested cluster nodes of a batch system are statically assigned for a fixed time period, inefficient reservation gaps are unavoidable. Similar problems appear in multi-threaded applications on multi-core systems. The programming systems group works on the dynamic adaptation of the extent of parallelism by the means of code transformations (under consideration of the resulting data redistribution) and operating system interactions. As control flow based synchronization measures interfere with the necessary analyzes, the programming systems group researches new programming constructs that can replace the existing ones and allow to specify the synchronization in a data-centric way.

- **Develop Testing for Parallelism**. In software engineering, testing has always assumed an important role. Code coverage, test data generation, reliability assessment etc. are tools of the trade. Unfortunately, current research insufficiently covers the indeterminism caused by concurrency. To deal with that issue, the programming systems group develops tools that consider (based on the coverage criteria) interleavings of parallel threads in their test data generation. This topic also includes research on operating systems and schedulers. As concurrency considerably increases the search space of the test generation it is necessary to develop infrastructures that allow the test generation and execution on a cluster.

- **Improve of Software Development Processes**. The current development practice of complex, business or security critical software in global distributed teams (commonly found in the software industry) demands compliance with well-defined software development processes. To support the enforcement of this requirement, appropriate development tools are used. The Practical Software Engineering research group that is lead by the honorary professors Dr. Bernd Hindel

and Dr. Detlef Kips cover the corresponding research area. Both possess long term experience in industrial software projects as managers of medium sized software companies. The goal of the Practical Software Engineering group is the development of a machine executable notation for modeling of software development processes. For that purpose the research group examines the semi-automatic retrieval of traceability information from the artefacts of different tools and notations as well as the model based development, integration and configurations of software components, used in the design of automotive embedded systems.

2 Research projects

2.1 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

offers 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 ecosystem 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. This time series database is geared towards an extremely efficient storage of a multitude of time series that optimizes disk space as well as response times.

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 in regard to the metrics mean-time-to-repair and mean-time-between-defects.

2.2 Efficient Software Architectures for Distributed Event Processing Systems

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Christopher Mutschler
Duration: 15.11.2010–15.5.2016
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 project is a contribution of the Programming Systems Group to the IZ ESI[1]

2.3 Embedded Systems Institute

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Stefan Kempf
Dipl.-Inf. Georg Dotzler
Dipl.-Inf. Thorsten Blaß
Dipl.-Inf. Tobias Werth
Dr.-Ing. Christopher Mutschler
Andreas Kumlehn, M. Sc.
Dr.-Ing. Norbert Oster
Demian Kellermann, M. Sc.
Start: 1.9.2007

[1] URL: http://www.esi.uni-erlangen.de/

In September 2007 the ESI – Embedded System Institute – was founded as an interdisciplinary center at the Friedrich-Alexander-University (FAU) with the goal to coordinate and organize research, teaching, and further education in the field of embedded systems.

ESI brings together existing skills within the university and interests, activities, and goals of large and medium sized companies in the field of embedded systems.

Companies obtain access to latest research results and the opportunity to develop common projects, to establish ties, and to find co-operation partners. The ESI concentrates the skills of the chairs of computer sciences and makes them usable for co-operation projects. Hence, the latest research results can be transferred into products in a speedy way. Finally, the ESI may serve as a platform for recruiting excellent students and highly qualified young academics at an early stage.

The chair of computer science department 2 (Prof. Philippsen) is one of the active founders of the ESI and carries out research projects within the ESI.

More information can be found at http://www.esi.uni-erlangen.de and http://www.esi-anwendungszentrum.de

2.4 ErLaDeF - Embedded Realtime Language Development Framework

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
PD Dr. Ronald Veldema
Dipl.-Inf. Thorsten Blaß
Dipl.-Math. Jakob Krainz
Dipl.-Inf. Daniel Brinkers
Start: 1.1.2012
Contact:
PD Dr. Ronald Veldema
Phone: +49-9131-85-27622
Fax: +49-9131-85-28809
E-Mail: ronald.veldema@fau.de

ErLaDeF is our test-bed for new programming language and compiler techniques. Our main focus is on building infrastructure for easier (hard + soft) real-time embedded parallel systems programming.

We focus on hard real-time embedded systems as they are about to go massively parallel in the near future.

Real-time and embedded systems also have hard constraints on resource usage. For example, a task should complete in a fixed amount of time, have guaranteed upper-limits on the amount of memory used, etc.

We are developing different ways to manage this concurrency using a combination of strategies: simpler language features, automatic parallelization, libraries of parallel programming patterns, deep compiler analysis, model checking, and making compiler analysis fast enough for interactive use.

Runtime Parallelization of Programs

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.

Script based language for embedded systems (Pylon)

Pylon is a language that is close to scripting languages (but is statically typed). A large part of the complexity that a programmer would normally take care of when creating an application, is moved to the compiler (i.e., type inference). The programmer does not have to think about types at all. By analyzing the expressions in the program, types are inferred (duck typing). The language is also implicitly parallel, the programmer does not need to have expert knowledge to parallelize an application. The compiler automatically

decides what to run in parallel. Finally, the language is kept simple so that learning the language remains easy for novice programmers. For example, we kept the number of keywords small.

Any language constructs that make analyzing the program hard for a compiler has been omitted (pointer arithmetic, inheritance, etc.). Any removed features have been replaced by simpler variants that can be easily analyzed. The current focus of this project lies in supporting the programmer in designing the code. The previous programming language research results have been absorbed in the Pylon project. For example, the prior research results on alternatives to polymorphism and inheritance have been added to the Pylon project. This allows us to report errors at compile time where other languages can only find them at run-time.

Analysis

Object-oriented languages offer the possibility to dynamically allcoate objects. The memory required for this is allocated at run-tie. However, in contrast to desktop systems, embedded systems typically have very little memory. If the 'new' operator is used often in an embedded system (that are now starting to be programmed with higher-level languages such as Java and C++ that include 'new'), memory can be exhausted at run-time causing an embedded system to crash.

In 2014, we created an analysis to find this problem at compile-time and report this to the developer.

To detect memory exhaustion at compile-time, the analysis determines the live-time of references to objects. If there are no more references to an object, the object can be removed from memory. Normally, such reference counting schemes are performed at run-time, we, however, perform reference counting at compile-time in an interactive fashion. The result is that memory management errors can be found at compile-time instead of at run-time. Additionally, the static reference counting increases a program's performance as the reference counts do not have to be manipulated at run-time. If it is statically determined that an object can be removed, the developer needs to insert a 'delete' statement. With explicit memory management, we are now able to statically determine a program's worst-case memory requirements. The whole analysis outlined above is integrated into Pylon and a predicate propagation framework previously reported on. Note that the analysis is language independent, however, and can be applied to other languages as well (Java, C++, etc.). However, we can in that case no guarantee that reference counts are correctly computed as we require Pylon's analyzability for this.

In 2014, we also have continued to work on interactive program analysis. The framework we developed in 2013 to do program analysis in a lazy manner, suitable for interactive use, was expanded and modified in order to support big code bases, to analyze them and keep the analysis results for later use. This enables us to precisely analyze

programs that use libraries, by first analyzing the library, and then using the library's analysis result to get precise analysis results for our program.

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/)

2.5 Graphs and Graph Transformations

Project manager:
Prof. em. Dr. Hans Jürgen Schneider
Start: 1.10.2004
Contact:
Prof. em. Dr. Hans Jürgen Schneider
Phone: +49-9131-85-27620
Fax: +49-9131-85-28809
E-Mail: hans.juergen.schneider@fau.de

Graphs are often used as an intuitive aid for the clarification of complex matters. Examples of outside computer science include, e.g., chemistry where molecules are modeled in a graphical way. In computer science, data or control flow charts are often used as well as entity relationship charts or Petri-nets to visualize software or hardware architectures. Graph grammars and graph transformations combine ideas from the fields of graph theory, algebra, logic, and category theory, to formally describe changes in graphs.

Category theory is an attractive tool for the description of different structures in a uniform way, e.g., the different models for asynchronous processes: Petri-Nets are based on standard labeled graphs, state charts use hierarchical graphs, parallel logic programming can be interpreted in a graph-theoretical way using so-called jungles, and the actor systems can be visualized as graphs, whose labeling alphabet is a set of term graphs.

Lately, we have concentrated our attention on a theoretical aspect.

Our work on graph transformation is based on notions borrowed from category theory. The so-called double-pushout approach represents a production by two morphisms starting at a common interface graph. One pushout glues the left-hand side of the production into the context, the other does with the right-hand side. Effectively constructing a derivation step, however, requires finding a pushout complement on the left-hand side. Some people consider this disadvantageous. In 1984, Raoult has proposed to model graph rewriting by a single pushout; Loewe has extensively studied this approach, but the discussion was mainly restricted to injective morphisms. Under this assumption, the

approaches are equivalent. Some relevant applications such as term graph rewriting, however, lead to non-injective morphisms. We have examined these cases in detail, and we could show that the equivalence also holds for non-injective cases as long as the handle satisfies some reasonable conditions.

2.6 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
Phone: +49-9131-85-28865
Fax: +49-9131-85-28809
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 2014 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 24 FAU teams plus 35 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 2014 in Linköping. 19 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 second and like the winning team from Denmark also managed to solve nine problems. Thus, FAU is the only German university that qualified for the upcoming World Finals in Morocco 2015. The other two FAU teams also did a great job. They solved six problems and finished on rank 17 and 24 of an total of 96 teams.

2.7 OpenMP/Java

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
PD Dr. Ronald Veldema
Dipl.-Inf. Georg Dotzler
Dipl.-Inf. Thorsten Blaß
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 2014 we developed a JaMP implementation for Android 4.0. Currently this version only supports the SIMD construct of OpenMP.

2.8 PATESIA - Parallelization techniques for embedded systems in automation

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Stefan Kempf
Dipl.-Inf. Georg Dotzler
Dipl.-Inf. Thorsten Blaß
Marius Kamp, M. Sc.
Start: 1.6.2009
Sponsored by:
ESI-Anwendungszentrum
Contact:
Dipl.-Inf. Georg Dotzler
Phone: +49-9131-85-27624
Fax: +49-9131-85-28809
E-Mail: georg.dotzler@fau.de

This project was launched in 2009 to address the refactorization and parallelization of applications used in the field of industry automation. These programs are executed on specially designed embedded systems. This hardware forms an industry standard and is used worldwide. As multicore-architectures are increasingly used in embedded systems, existing sequential software must be parallelized for these new architectures in order to improve performance. As these programs are typically used in the industrial domain for the control of processes and factory automation they have a long life cycle. Because of this, the programs often are not being maintained by their original developers any more. Besides that, a lot of effort was spent to guarantee that the programs work reliably. For these reasons the software is only extended in a very reluctant way.

Therefore, a migration of these legacy applications to new hardware and a parallelization cannot be done manually, as it is too error prone. Thus, we need tools that perform these tasks automatically or aid the developer with the migration and parallelization.

Research on parallelization techniques

We developed a special compiler for the parallelization of existing automation programs. First, we examined automation applications with respect to automatic parallelizability. We found that it is hard to perform an efficient automatic parallelization with existing techniques. Therefore this part of the project focuses on two steps to handle this situation. As first step, we developed a data dependence analysis that identifies potential critical sections in a parallel program, presents them to the programmer and adds their protection to the code. We ware able to show that our approach to identify critical sections finds atomic blocks that closely match the atomic blocks that an expert would add to the code. Besides that, we showed in 2014 that the impacts on execution times are negligible if our technique finds atomic blocks that are larger than necessary or that are not necessary at all.

As second step we have refined and enhanced existing techniques (software transactional memory (STM) and lock inference) to implement atomic blocks. In our approach, an atomic block uses STM as long as lock inference would lead to coarse-grained synchronization. The atomic block switches from STM to lock inference as soon as a fine-grained synchronization is possible. With this technique, an atomic block always uses fine-grained synchronization while the runtime overhead of STM is minimized at the same time. We showed that (compared to a pure STM or lock inference implementation) our technique speeds up execution times by a factor between 1.1 and 6.3. Although fine-grained synchronization in general leads to better performance than a coarse-grained solution, there are cases where a coarse-grained implementation shows equal performance. We therefore presented a runtime mechanism for an STM that also works together with our combined technique. This runtime mechanism starts with a small number of locks, i.e., a coarse-grained locking, where accesses to different shared variables are protected by the same lock. If this coarse-grained locking leads to too many non-conflicting accesses waiting for the same lock, our approach gradually increases the number of locks. This makes the locking more fine-grained so that non-conflicting accesses can be executed concurrently. Our runtime mechanism that dynamically tunes the locking-granularity makes the programs run up to 3.0 times faster than a fixed coarsegrained synchronization.

We completed this project part in 2014.

Research on migration techniques

Our research on the migration of legacy applications originally consisted of having a tool that automatically replaces suboptimal code constructs with better code. The code

sequences that had to be replaced as well as the replacement codes were specified by developers by means of a newly developed pattern description language. However, we found this approach to be too difficult for novice developers.

This led us to the development of a new tool that automatically learns and generalizes patterns from source code archives, recognizes them in other projects, and presents recommendations to developers. The foundation of our tool lies in the comparison of two versions of the same program. It extracts the changes that were made between two source codes, 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.

In 2014 we developed a new symbolic code execution engine to minimize the number of wrong recommendations. Depending on the number and the generality of the patterns in the database, it is possible that without the new engine our tool generates some unfitting recommendations. To discard the unfitting ones, 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 our approach are its applicability to isolated code fragments and its automatic configuration that does not require any human interaction.

The latest results of our tool SIFE are found here[2] (last update: 2014-05-09).

Parts of the project are funded by the ”ESI-Anwendungszentrum”[3]

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

[2]URL: https://www2.cs.fau.de/staff/dotzler/SIFE_results.tar.gz
[3]URL: http://www.esi-anwendungszentrum.de/.

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

Modern software systems are getting 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 that gathers traceability data and metrics from a variety of software engineering tools, and the analysis core, that allows to analyze the integrated data in a holistic way. Each subsystem is developed in a separate subproject.

The project partner QAware GmbH implements 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 use 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 research 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 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 2014, we improved the modularity of the language to make it both more adaptable and extendable in terms of structure and operations. This not only increases its expressiveness but also improves the reusability of existing traceability analyses.

2.10 Compiler-supported parallelization for multi-core architectures

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Tobias Werth
Start: 1.3.2007
Contact:
Dipl.-Inf. Tobias Werth
Phone: +49-9131-85-28865
Fax: +49-9131-85-28809
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.

2.11 WEMUCS - Techniques and tools for iterative development and optimization of software for embedded multicore systems

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Demian Kellermann, M. Sc.
Dr.-Ing. Norbert Oster
PD Dr. Ronald Veldema
Duration: 15.10.2012–30.11.2014
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

Multicore processors are of rising importance in embedded systems as these processors offer high performance while maintaining low power consumption. Developing parallel software for these platforms poses new challenges for many industrial sectors because established tools and software libraries are not multicore enabled. The efficient development, optimization and testing of multicore-software is still open research, especially for reliable real-time embedded systems.

In the multi-partner project WEMUCS[4] new methods and tools for efficient iterative development, optimization, and testing of multicore software have been created over the past two years. Innovative tools and technologies for modeling, simulation, visualization, tracing, and testing have been developed and integrated into a single tool chain.

[4]URL: http://www.multicore-tools.de/

Using case studies from different industries (automotive, telecommunications, industry automation) these tools were evaluated and improved.

Although several well-known methods for test case generation and best practice coverage measures for classical single-core applications exist, no such methods for multi-core software have established themselves yet. Unfortunately, it is the interaction of concurrent threads that can cause faults that cannot be discovered by testing the individual threads in isolation. As part of the WEMUCS project (more precisely: work package AP3[5]) and based on an industrial size case study, we developed a generic technique (called a "testing pipeline" below) that systematically creates test cases to find and analyze the impact of concurrent side effects.

To evaluate the new testing pipeline (including the automated parallelization of sequential code) on real world examples, our project partner Siemens created a complete model of a large luggage conveyor belt, including the code to control the belt. Such a luggage conveyor can be found at every airport. The case study's model is used to automatically derive luggage conveyor belt systems of different sizes, i.e., built from an arbitrary number of feeder or outlet belts. The hardware of the conveyor belt is emulated on the SIMIT simulation tool from Siemens and the control software (written in the programming language AWL) runs on a software-based PLC.

The first step of the testing pipeline converts the AWL code into a more comprehensible and human-readable programming language called HLL. We have completed this converter during this reporting period. In step two, our tool then transforms previously sequential parts of the AWL code into HLL units that are executed in parallel. When applied to a luggage conveyor belt system built from eight feeders, eight outlets and an inteconnecting circular belt of straight and curved segments, our tool automatically transcoded 11.704 lines of sequential AWL code into 34 KB of parallel HLL code.

In step three another tool developed by the chair then analyzes the HLL code and automatically generates a testing model. This model represents the interprocedural control flow of the concurrent subroutines and also holds all the thread switches that might be relevant for testing. The testing model consists of a set of hierarchically organized UML activities (currently encoded as an XMI document that can be imported into Enterprise Architect by Sparx Systems). When applied to the case study outlined above, our tool automatically generates 103 UML activity diagrams (with 1,302 nodes and 2,581 edges).

Step four is optional. The tester can manually adapt the testing model as needed (e.g. by changing priorities or inserting additional verification points). Then the completed model can be loaded into the MBTsuite tool, a model-based testing tool developed by our project partner sepp.med GmbH. This tool is highly configurable to generate test

[5]URL: http://www.multicore-tools.de/de/test.html

cases that cover as many parts of the testing model as possible. We ran MBTsuite on a standard PC and applied it to the testing model of our case study; within six minutes MBTSuite generated a highly optimized test set consisting of only 10 test cases that nevertheless cover 99of the edges.

In cooperation with our project partner sepp.med GmbH we built two export modules for the above-mentioned MBTsuite. One module outputs the generated test cases as a human-readable spreadsheet, the other module outputs an executable test set in the Java language. The exported spreadsheet contains one individual sheet per test case, with one column per thread. The rows visualize the thread interleaving that gets tested. The Java file holds a Java class with a test method per test case. Every method holds a sequence of test steps that discretely control thread interleavings. This way, each test case execution leads to a unique and reproducible execution of the parallel System Under Test (SUT) written in HLL. Each run of a test instructs our HLL emulator to load and initialize the SUT and each subsequent test step instructs the HLL emulator to execute a certain set of instructions from a certain thread. During this fully controlled execution, each test case emits a detailed protocol of its execution for the final visualization step.

A plug-in from sepp.med that creates a visualization layer in Enterprise Architect's UML editor visualizes the resulting log file. Colored nodes and edges tell the user which control flow paths a test has covered. If a test case "fails" (if a race condition or a logic error is found in the tested program), its graphical trace ends at the failing statement. The tester can then follow the control flow back in time in order to understand the underlying reason for the failure.

We have implemented a prototype of the full testing pipeline and demonstrated its applicability to an industrial size case study. This tool is a major contribution to testing concurrent code for embedded systems. It is a contribution of the Programming Systems Group to the ESI initiative[6].

[6]URL: http://www.esi-anwendungszentrum.de

3 Publications 2014

– Al-Hilank, Samir ; Jung, Martin ; Kips, Detlef ; Husemann, Dirk ; Philippsen, Michael: Using Multi Level-Modeling Techniques for Managing Mapping Information . In: ACM/IEEE (Ed.) : Proceedings of the 1st Int. Workshop on Multi-Level Modelling, ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (1st Int. Workshop on Multi-Level Modelling, ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems Valencia, Spain Sept. 28 - Oct. 3). Aachen : CEUR-WS, 2014, pp 103-112. (CEUR Workshop Proceedings Vol. 1286)

– 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). 2014, pp -.

– Kempf, Stefan ; Veldema, Ronald ; Philippsen, Michael: Combining Lock Inference with Lock-Based Software Transactional Memory . In: Cascaval, Calin ; Montesinos, Pablo (Ed.) : Proceedings of the 26th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2013) (26th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2013) Santa Clara, California, USA 25.-27.09.2013). Berlin : Springer-Verlag Berlin Heidelberg, 2014, pp 325-341. (Lecture Notes in Computer Science (LNCS) Vol. 8664) - ISBN 978-3-319-09966-8

– Kempf, Stefan: Compiler and Runtime Techniques to Identify and Optimize Atomic Blocks in Parallel Programs . Göttingen : Cuvillier Verlag, 2014. Zugl.: Erlangen, Friedrich-Alexander-Universität Erlangen-Nürnberg, Ph.D. thesis, 2014. - 161 pages.

– Lautenschlager, Florian ; Adersberger, Josef ; Kumlehn, Andreas ; Philippsen, Michael: Design for Diagnosability . In: Java Magazin (Software-&-Support-Verlag, Frankfurt am Main) (2014), No. 5, pp 44-50, ISSN 1619-795X

– Mutschler, Christopher ; Philippsen, Michael: Adaptive Speculative Processing of Out-of-Order Event Streams . In: ACM Transactions on Internet Technology (TOIT) 14 (2014), No. 1, pp 4:1-4:24, ISSN 1557-6051

– Mutschler, Christopher: Latency Minimization of Order-Preserving Distributed Event-Based Systems . München : Verlag Dr. Hut, 2014. Zugl.: Erlangen, Friedrich-Alexander-Universität Erlangen-Nürnberg, Ph.D. thesis, 2014. - 229 pages. ISBN 978-3-8439-1472-7

– Mutschler, Christopher ; Loeffler, Christoffer ; Witt, Nicolas ; Edelhäußer, Thorsten ; Philippsen, Michael: Predictive Load Management in Smart Grid Environments (Best Paper Award) . In: ACM (Ed.) : Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (8th ACM International Conference on Distributed Event-Based Systems Mumbai, India 26.05. - 29.05.2013). 2014, pp 282-287. - ISBN 978-1-4503-2737-4

– Tausch, Norbert ; Philippsen, Michael: A Modular and Statically Typed Effectful Stack for Custom Graph Traversals . In: Tichy, Matthias ; Westfechtel, Bernhard (Ed.) : Proceedings of the 8th International Workshop on Graph-Based Tools (GraBaTs 2014) (8th International Workshop on Graph-Based Tools (GraBaTs 2014) York, UK 25.07.2014). 2014, pp -. (Electronic Communications of the EASST, No. 68)

4 Exam theses (german only)

– Bachelor Thesis: Vectorization of recursive parallel programs. Bearbeiter: Patrick Kreutzer (beendet am 31.01.2014); Betreuer: Dipl.-Inf. Tobias Werth; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Entwurf und Realisierung einer Programmierschnittstelle zur Software-Diagnose basierend auf aspekt-orientierten Paradigmen. Bearbeiter: Pascal Wagner (beendet am 04.04.2014); Betreuer: Andreas Kumlehn, M. Sc.; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Abbildung der LLVM-Zwischensprache (Bitcode) auf die Zwischensprache des LS2-Programmanalyse-Frameworks.. Bearbeiter: Andreas Grünwald (beendet am 08.09.2014); Betreuer: Dipl.-Inf. Thorsten Blaß; PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen

– Master Thesis: Entwicklung eines Werkzeugs zum Vergleich von Code-Fragmenten durch symbolische Ausführung. Bearbeiter: Marius Kamp (beendet am 01.10.2014); Betreuer: Dipl.-Inf. Georg Dotzler; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Integration of new SIMD-Features into the Android Dalvik VM. Bearbeiter: Christian Cardello (beendet am 02.10.2014); Betreuer: Dipl.-Inf. Georg Dotzler; PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Entwurf und Umsetzung einer Bibliothek zur automatisierten Analyse von Zeitreihen im Umfeld der Software-Diagnose. Bearbeiter: Marc Rosenbauer (beendet am 30.10.2014); Betreuer: Andreas Kumlehn, M. Sc.; Prof. Dr. Michael Philippsen

– Master Thesis: Optimierungen für ein CUDA-basiertes JavaScript-System. Bearbeiter: Eugen Meissner (beendet am 01.11.2014); Betreuer: PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Realisierung eines Serialisierungsmechanismus für tiefe Modelle. Bearbeiter: Florian Gerdes (beendet am 1.11.2014); Betreuer: Dipl.-Inf. Samir Al-Hilank; Dr.-Ing. Martin Jung; Hon.-Prof. Dr.-Ing. Detlef Kips

– Bachelor Thesis: Synchronisationsfreie Graphstruktur auf der GPU. Bearbeiter: Daniel Wust (beendet am 6.11.2014); Betreuer: Dipl.-Inf. Thorsten Blaß; PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen

– Bachelor Thesis: CudaMPI: Nachrichtenaustausch zwischen Rechnerknoten über Cuda und MPI. Bearbeiter: Matthias Huth (beendet am 10.11.2014); Betreuer: PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Berichtgenerierung für tiefe Modelle. Bearbeiter: Valerie Wiedemann (beendet am 17.11.2014); Betreuer: Dipl.-Inf. Samir Al-Hilank; Dr.-Ing. Martin Jung; Hon.-Prof. Dr.-Ing. Detlef Kips

– Bachelor Thesis: Entwurf und Implementierung einer Lastverteilung auf mehrere XeonPhi-Prozessoren im Cluster. Bearbeiter: Christian Eichler (beendet am 18.11.2014); Betreuer: Dipl.-Inf. Tobias Werth; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Measurement and Analysis of Runtime-Metrics in a Continuous Integration Environment. Bearbeiter: Victor Simon (beendet am 25.11.2014); Betreuer: Andreas Kumlehn, M. Sc.; Prof. Dr. Michael Philippsen

– Master Thesis: Entwurf und Evaluation eines Ansatzes zur approximativen Ereignisverarbeitung auf Sensordatenströmen. Bearbeiter: Christoffer Löffler (beendet am 1.12.2014); Betreuer: Dr.-Ing. Christopher Mutschler; Prof. Dr. Michael Philippsen