

# **Jahresbericht 2016 des Lehrstuhls für Informatik 2 (Programmiersysteme)**

**Anschrift:** Martensstr. 3, 91058 Erlangen

**Tel.:** +49-9131-85-27621

**Fax:** +49-9131-85-28809

**E-Mail:** info@i2.informatik.uni-erlangen.de

## **Ordinarius:**

Prof. Dr. Michael Philippsen

## **Honorarprofessor:**

Hon.-Prof. Dr.-Ing. Bernd Hindel

Hon.-Prof. Dr.-Ing. Detlef Kips

## **Emeritus:**

Prof. em. Dr. Hans Jürgen Schneider

## **Sekretariat:**

Margit Zenk

## **Wiss. Mitarbeiter:**

Michael Baer, M. Sc. (ab 01.02.2016)

Dipl.-Inf. Thorsten Blaß

Dipl.-Inf. Daniel Brinkers (bis 30.04.2016)

Dipl.-Inf. Georg Dotzler (bis 31.01.2016)

Marius Kamp, M. Sc.

Dipl.-Math. Jakob Krainz

Patrick Kreutzer, M. Sc.

Andreas Kumlehn, M. Sc. (bis 31.10.2016)

Dr.-Ing. Christopher Mutschler

Dipl.-Inf. Daniela Novac

Dr.-Ing. Norbert Oster, Akad. ORat

Dipl.-Inf. Tobias Werth (bis 31.01.2016)

## **IT-Betreuer:**

Dipl.-Ing. (FH) Helmut Allendorf

Manfred Uebler

## **Gast:**

Dr.-Ing. Josef Adersberger

Dr.-Ing. Martin Jung

Florian Lautenschlager, M. Sc.

Norbert Tausch, M. Eng.

## **Externes Lehrpersonal:**

Dr.-Ing. Klaudia Dussa-Zieger

Dr.-Ing. Martin Jung

Seit Prof. Dr. Michael Philippsen 2002 die Leitung des 1972 gegründeten Lehrstuhls übernommen hat, stehen **Programmiersysteme für heterogene Multicore-Rechner** im Zentrum der Forschungsarbeiten des Lehrstuhls.

Bis 2014/2015 standen dabei vor allem systemnahe Forschungsthemen im Vordergrund, die näher an der Hardware, den Laufzeitsystemen, den unterliegenden Betriebssystemen und der Frage der Programmierung dieser Rechensysteme ausgerichtet waren. Es ging dem Lehrstuhl bisher insbesondere darum, das Parallelisierungspotential einer Anwendung verlustarm an die in der Hardware vorhandene Parallelität anzupassen und die von der Hardware bereitgestellten Möglichkeiten effizient nutzbar zu machen. Weil diese systemnahen Fragen noch immer von aktueller Bedeutung sind, werden sie auch in Zukunft in einigen Projekten des Lehrstuhls bearbeitet.

Jetzt konzentriert sich der Lehrstuhl auf Forschungsthemen, die ingenieurwissenschaftliche Antworten für den Software-Ingenieur liefern, der im Rahmen industrieller Software-Entwicklung für Multicore-Rechner, für daraus bestehende verteilte Systeme, für paralleles Cloud-Computing sowie für vernetzte eingebettete Systeme parallele Software entwickelt.

## 1 Forschungsschwerpunkte

Die Eckpunkte der vom Lehrstuhl bearbeiteten **Programmiersystemforschung** lassen sich wie folgt abstecken: (a) Aufgrund des weiterhin dringenden Bedarfs entwickeln und evaluieren wir auch zukünftig **Programmiermodelle**, die heterogene parallele Komponenten mit einer einheitlichen Schnittstelle versehen und portablen Code ermöglichen, der auf unterschiedlichen Konfigurationen (aus Multicores, GPUS, Acceleratoren, FPGAs etc.) lauffähig ist. Nur so können sich Investitionen in parallele Software längerfristig rechnen. (b) Es wird weiter an wirtschaftlicher Bedeutung gewinnen, vorhandene Software für Multicore-Rechner zu parallelisieren. Wir erforschen neuartige Werkzeuge, die auf **Code-Repositories** arbeiten, diese analysieren und den Entwickler bei **Migration, Refaktorisierung und Parallelisierung** unterstützen. In speziellen Situationen und Kontexten ist eine automatische Parallelisierung möglich und effektiv. (c) Wegen der durch die Parallelität und den Synchronisierungsbedarf gestiegenen Komplexität muss der Entwickler stärker als bisher bei der Neu- und Weiterentwicklung von parallelem Code unterstützt werden, um Fehler im Vorfeld zu vermeiden und frühzeitig zu erkennen und zu beheben, auch weil diese Tätigkeiten nicht länger dem systemnah arbeitenden Spezialisten vorbehalten sind. Wir entwickeln dazu **schnellere, interaktive, inkrementelle und ggf. selbst parallel ausgeführte Code-Analysen**, durch die nicht nur Wettlaufsituationen, konkurrierende Ressourcenzugriffe etc. entdeckt werden, sondern die dem Programmierer auch punktgenau und interaktiv in der Entwicklungsumgebung Verbesserungsvorschläge für den gerade in der Entwicklung befindlichen

Code liefern. (d) Im Lebenszyklus von paralleler Software stellen sich durch den Indeterminismus der Nebenläufigkeit auch beim **Test paralleler Programme**, bei ihrer Qualitätssicherung, bei der Sicherstellung der **Code-Authentizität** sowie bei ihrem Betrieb und ihrer **Diagnose** neuartige und an Bedeutung gewinnende Fragen. Wir untersuchen, wann paralleler Code ausreichend gut getestet ist, wie man parallele Programme gegen Angriffe schützen kann, wie Testdaten für parallele Programme erzeugt werden können, wie Ursachen von erraticem Verhalten des parallelen Codes systematisch gefunden werden können etc. Diese Fragen erhalten durch das Vorhandensein der Parallelität im Code eine neue Dimension, die etablierte Techniken aus dem Software-Engineering nur unzureichend beantworten können.

Der Lehrstuhl arbeitet stets **Programm-Code-basiert** und erstellt funktionsfähige **Prototypen** der erforschten Werkzeuge. Uns ist dabei eine **qualitative und quantitative Evaluation** sehr wichtig.

## 2 Forschungsprojekte

### 2.1 Analyse von Code-Repositories

**Projektleitung:**

Prof. Dr. Michael Philippsen

**Beteiligte:**

Dipl.-Inf. Georg Dotzler

Marius Kamp, M. Sc.

Patrick Kreutzer, M. Sc.

**Beginn:** 1.1.2010

Bei der Weiterentwicklung von Software führen die Entwickler oftmals sich wiederholende, ähnliche Änderungen durch. Dazu gehört beispielsweise die Anpassung von Programmen an eine veränderte Bibliotheksschnittstelle, die Behebung von Fehlern in funktional ähnlichen Komponenten sowie die Parallelisierung von sequentiellen Programmteilen. Wenn jeder Entwickler die nötigen Änderungen selbst erarbeiten muss, führt dies leicht zu fehlerhaften Programmen, beispielsweise weil weitere zu ändernde Stellen übersehen werden. Wünschenswert wäre stattdessen ein automatisiertes Verfahren, das ähnliche Änderungen erkennt und mit dieser Wissensbasis Software-Entwickler bei weiteren Änderungen unterstützt.

#### **Änderungsextraktion**

Im Rahmen dieses Projekts entwickelten wir daher das Migrations- und Refaktorisierungswerkzeug SIFE. Der Ansatz basiert darauf, dass zwei Versionen eines Programms,

die in einem Versionsarchiv vorhanden sind, miteinander verglichen werden. Das Werkzeug extrahiert dabei automatisch, welche Änderungen sich zwischen den beiden Versionen ergeben haben, und leitet daraus generalisierte Muster aus zu ersetzenden Code-Sequenzen ab. Diese Muster werden in einer Datenbank gespeichert und können anschließend von unserem Werkzeug dazu verwendet werden, analoge Änderungen für den Quellcode anderer Programme automatisch vorzuschlagen.

Zur Extraktion der Änderungen verwenden wir ein baumbasiertes Verfahren. Im Jahr 2016 wurde ein neuer Algorithmus (MTDIFF) für solche baumbasierten Verfahren entwickelt und gut sichtbar publiziert, der die Genauigkeit der Änderungsbestimmung verbessert.

### **Symbolische Ausführung von Code-Fragmenten**

Im Jahr 2014 wurde ein neues Verfahren zur symbolischen Code-Ausführung namens SYFEX entwickelt, welches die Ähnlichkeit des Verhaltens zweier Code-Teilstücke bestimmt. Mit diesem Verfahren soll eine Steigerung der Qualität der Verbesserungsvorschläge erreicht werden. Abhängig von der Anzahl und Generalität der Muster in der Datenbank kann SIFE ohne das neue Verfahren unpassende Vorschläge liefern. Um dem Entwickler nur die passenden Vorschläge anzuzeigen, wird das semantische Verhalten des Vorschlags mit dem semantischen Verhalten des Musters aus der Datenbank verglichen. Weichen beide zu sehr voneinander ab, wird der Vorschlag aus der Ergebnismenge entfernt. Die Besonderheit von SYFEX besteht darin, dass es auf herausgelöste Code-Teilstücke anwendbar ist und keine menschliche Vorkonfiguration benötigt.

SYFEX wurde im Jahr 2015 verfeinert und auf Code-Teilstücke aus Archiven von verschiedenen Software-Projekten angewendet. Der Schwerpunkt im Jahr 2016 lag auf einer Untersuchung, inwieweit SYFEX zum semantischen Vergleich von Abgaben eines Programmierwettbewerbs geeignet ist.

### **Detektion von semantisch ähnlichen Code-Fragmenten**

SYFEX erlaubt es, die semantische Ähnlichkeit zweier Code-Fragmente zu bestimmen. So ist es damit prinzipiell möglich, Paare oder Gruppen von semantisch ähnlichen Code-Fragmenten (semantische Klone) zu identifizieren. Auf Grund des hohen Laufzeitaufwands verbietet sich der Einsatz von SYFEX – wie auch von anderen Werkzeugen dieser Art – allerdings, um in größeren Code-Projekten nach semantisch ähnlichen Code-Fragmenten zu suchen. Im Jahr 2016 wurde deshalb mit der Entwicklung eines Verfahrens begonnen, mit dessen Hilfe die Detektion semantisch ähnlicher Code-Fragmente beschleunigt werden kann. Grundlage dieses Verfahrens ist eine Reihe von sog. Basiskomparatoren, die zwei Code-Fragmente jeweils hinsichtlich eines Kriteriums (beispielsweise die Anzahl bestimmter Kontrollstrukturen oder die Beschaffenheit der Kontrollflussgraphen) miteinander vergleichen und dabei möglichst geringen Laufzeitaufwand haben. Diese Basiskomparatoren können anschließend zu einer Hierarchie von

Verfahren verknüpft werden. Um damit die semantische Ähnlichkeit zweier Fragmente möglichst genau bestimmen zu können, wird mit Hilfe der Genetischen Programmierung nach Hierarchien gesucht, die die von SYFEX für eine Reihe von Code-Paaren berechneten Ähnlichkeitswerte möglichst gut approximieren. Im Rahmen einer ersten Untersuchung hat sich gezeigt, dass sich das implementierte Verfahren tatsächlich für die Bestimmung von semantisch ähnlichen Code-Paaren eignet.

### **Cluster-Bildung von ähnlichen Code-Änderungen**

Voraussetzung für die Erzeugung generalisierter Änderungsmuster ist es, die Menge aller aus einem Quelltext-Archiv extrahierten Code-Änderungen in Teilmengen zueinander ähnlicher Änderungen aufzuteilen. Im Jahr 2015 wurde diese Erkennung ähnlicher Änderungen im Rahmen eines neuen Werkzeugs C3 verbessert. In einem ersten Schritt wurden verschiedene Metriken für den paarweisen Ähnlichkeitsvergleich der extrahierten Code-Änderungen implementiert und evaluiert. Darauf aufbauend wurden aus der Literatur bekannte Clustering-Algorithmen evaluiert und neue Heuristiken zur automatisierten Bestimmung der jeweiligen Parameter implementiert, um das bisherige naive Verfahren zur Identifizierung ähnlicher Änderungen zu ersetzen. Mit den im Rahmen von C3 implementierten Verfahren konnte im Vergleich zum bisherigen Ansatz eine deutliche Verbesserung erzielt werden. So können mit den neuen Verfahren mehr Gruppen ähnlicher Änderungen identifiziert werden, die sich für die Weiterverarbeitung im Rahmen von SIFE zur Generierung von Vorschlägen eignen.

Die zweite Verbesserung zielt darauf ab, die erhaltenen Gruppen ähnlicher Änderungen zusätzlich automatisiert zu verfeinern. Zu diesem Zweck wurden verschiedene Verfahren aus dem Umfeld des maschinellen Lernens zur Ausreißererkennung untersucht, um Änderungen, die fälschlicherweise einer Gruppe zugeordnet wurden, wieder zu entfernen.

Im Jahr 2016 wurde C3 um eine weitere Metrik zum Vergleich zweier Code-Änderungen erweitert, die im Wesentlichen den textuellen Unterschied zwischen den Änderungen (wie er beispielsweise von dem Unix-Werkzeug 'diff' erzeugt wird) bewertet. Des Weiteren wurde das in C3 implementierte Verfahren im Rahmen eines Konferenzbeitrags veröffentlicht. In diesem Zusammenhang wurde auch der zur Evaluation des Verfahrens erzeugte Datensatz von Gruppen ähnlicher Änderungen unter einer Open-Source-Lizenz veröffentlicht, siehe <https://github.com/FAU-Inf2/cthree>. Dieser kann zukünftigen Arbeiten als Referenz oder Eingabe dienen. Außerdem wurden prototypisch Verfahren implementiert, mit denen die Ähnlichkeitsberechnung und das Clustering in C3 inkrementell erfolgen können. Diese erlauben es, dass bei neuen Änderungen, die zu einem Software-Archiv hinzugefügt werden, die zuvor bereits berechneten Ergebnisse weiterverwendet werden können und nur ein Teil der Arbeit wiederholt werden muss.

## 2.2 Automatische Erkennung von Wettlaufsituationen

### **Projektleitung:**

Prof. Dr. Michael Philippsen

### **Beteiligte:**

Michael Baer, M. Sc.

**Beginn:** 1.1.2016

### **Kontakt:**

Michael Baer, M. Sc.

Tel.: +49-9131-85-27622

Fax: +49-9131-85-28809

E-Mail: michael.baer@fau.de

Große Softwareprojekte, an denen hunderte Entwickler arbeiten, sind schwer zu überblicken und enthalten viele Fehler. Zum Testen solcher Software haben sich automatisierte Tests bewährt, die Teilbereiche der Software (Unit-Tests) oder die komplette Software (Systemtest) testen und Fehler reproduzierbar erkennen können. Dieses Vorgehen funktioniert gut bei sequentiellen Programmen, die ein deterministisches Verhalten aufweisen. Moderne Software enthält jedoch vermehrt Parallelität. Durch diese Nebenläufigkeit treten etliche neue, teils schwer zu lokalisierende, Fehler auf, die nicht mehr durch herkömmliche Testverfahren sicher detektiert werden können. Dazu gehören vor allem Verklemmungen und gleichzeitige Zugriffe auf die selbe Speicherstelle. Ob ein solcher Fehler auftritt, hängt von dem konkreten Ausführungsplan der Aktivitätsfäden ab. Dieser unterscheidet sich jedoch bei jeder Ausführung und ist auch stark von dem darunterliegenden System abhängig. Somit treten entsprechende Fehler normalerweise nicht bei jedem Testlauf auf, je nach Testsystem sogar niemals. Aus diesem Grund sind die herkömmlichen Testverfahren nicht mehr ausreichend für moderne, nebenläufige Software.

In dem Projekt AuDeRace werden Verfahren entwickelt, die Nebenläufigkeitsfehler reproduzierbar, effizient und mit möglichst geringen Aufwand für Entwickler erkennen können. Unter anderem wird eine Technik entwickelt, die es ermöglicht, Tests um einen Ablaufplan zu erweitern, durch den die Ausführung weiterhin deterministisch bleibt. Ein weiteres generelles Problem an Tests bleibt dabei aber bestehen: Der Entwickler muss sich zunächst Testfälle überlegen und diese anschließend implementieren. Insbesondere im Kontext von Parallelität ist es aber extrem schwer, sich das Verhalten von Programmen vorzustellen und problematische Stellen zu identifizieren. Deshalb sollen kritische Stellen automatisiert erkannt werden, bevor überhaupt entsprechende Tests geschrieben wurde. Es existieren bereits Ansätze, potentiell kritische Stellen zu detektieren, allerdings werden dabei entweder sehr viele fälschliche Warnungen generiert oder die Analyse ist immens zeitintensiv. Ziel dieses Projektes ist es, durch eine geeignete

Kombination aus statischer und dynamischer Analyse, die Anzahl an falschen Erkennungen zu reduzieren, bzw. langsame Analysen zu beschleunigen, sodass die Verfahren nicht nur in kleinen Testbeispielen funktionieren, sondern auch komplexe Softwareprojekte effizient analysieren können.

Im Jahr 2016 wurden bestehende Ansätze und Programme auf ihre Tauglichkeit untersucht. Dabei wurde ein vielversprechendes Verfahren ausgemacht, das mithilfe von Model Checking und vorgegebenen Bedingungen Ablaufpläne konstruiert, die ungewolltes Verhalten erzeugen. Allerdings zeigen die Ergebnisse ein deutliches Problem hinsichtlich eines produktiven Einsatzes, da in sinnvoller Zeit nur sehr kleine Programme analysiert werden können. Aktuell beschäftigt sich das Projekt damit, die Programme um möglichst viele Anweisungen zu kürzen, die nichts mit den gesuchten Wettlaufsituationen zu tun haben. Dadurch sollen spätere Analysen beschleunigt werden.

## 2.3 Design for Diagnosability

### **Projektleitung:**

Prof. Dr. Michael Philippsen

### **Beteiligte:**

Andreas Kumlehn, M. Sc.

Dr.-Ing. Josef Adersberger

Florian Lautenschlager, M. Sc.

**Laufzeit:** 15.5.2013–31.7.2016

### **Förderer:**

IuK Bayern

### **Kontakt:**

Prof. Dr. Michael Philippsen

Tel.: +49-9131-85-27625

Fax: +49-9131-85-28809

E-Mail: michael.philippsen@fau.de

Viele Software-Systeme verhalten sich während der Testphase oder sogar im Regelbetrieb im negativen Sinne auffällig. Die Diagnose und die Therapie solcher Laufzeitanomalien ist oft langwierig und aufwändig bis hin zu unmöglich. Mögliche Folgen bei der Verwendung des Software-Systems sind lange Antwortzeiten, nicht erklärbares Verhalten oder auch Abstürze. Je länger die Folgen unbehandelt bleiben, desto höher ist der entstehende wirtschaftliche Schaden.

”Design for Diagnosability” beschreibt eine Werkzeugkette mit Modellierungssprachen, Bausteinen und Werkzeugen, mit denen die Diagnosefähigkeit von Software-Systemen gesteigert wird. Mit dieser Werkzeugkette werden Laufzeitanomalien schneller erkannt

und behoben – idealerweise noch während der Entwicklung des Software-Systems. Unser Kooperationspartner QAware GmbH bringt ein Software EKG ein, mit dem die Exploration von Laufzeit-Metriken aus Software-Systemen, visualisiert als Zeitreihen, möglich ist.

Das Forschungsprojekt Design for Diagnosability erweitert das Umfeld dieses bestehenden Software-EKG. Die Software-Blackbox misst minimal-invasiv technische und fachliche Laufzeitdaten des Systems. Die Speicherung der erfassten Daten erfolgt in Form von Zeitreihen in einer neu entwickelten Zeitreihendatenbank Chronix. Chronix ist darauf ausgelegt, eine Vielzahl an Zeitreihen äußerst effizient hinsichtlich Speicherplatzbedarf und Zugriffszeiten zu speichern. Chronix ist ein Open Source Projekt ([www.chronix.io](http://www.chronix.io)) und kann frei benutzt werden. Die Zeitreihen werden mit der Time-Series-API analysiert, z.B. mittels einer automatisierten Strategie zur Erkennung von Ausreißern. Die Time-Series-API bietet Grundbausteine, um weitere Strategien zur Identifikation von Laufzeitanomalien in Zeitreihen umzusetzen.

Die aufgeführten Werkzeuge werden in Kombination mit dem bestehenden Software-EKG zum Dynamic Analysis Workbench ausgebaut, um eine zeitnahe Diagnose und Behebung von Laufzeitanomalien zu ermöglichen. Hierzu sind Diagnosepläne vorgesehen, die einen Software-Entwickler unterstützen, eine Laufzeitanomalie schneller und zuverlässiger einzugrenzen und zu erkennen. Das Ziel der Werkzeugkette ist die Qualität von Software-Systemen zu erhöhen, insbesondere hinsichtlich der Kennzahlen Mean-Time-To-Repair sowie Mean-Time-Between-Defects.

Vor dem erfolgreichen Projektabschluss im Juli 2016 konnten noch eine Reihe wesentlicher Beiträge geleistet werden:

- Wir haben Chronix und ein Framework zur verteilten Berechnung gekoppelt. Dadurch skaliert die Anomalieanalyse jetzt auf riesige Mengen an Zeitreihendaten.
- Wir haben Chronix fortentwickelt und weitere Komponenten ergänzt, wie z. B. ein noch effizienteres Speichermodell, mehrere Adapter für diverse Zeitreihendatenbanken, weitere server-seitige Analysefunktionen und neue Zeitreihentypen.
- Wir haben unseren Benchmark für Zeitreihendatenbanken veröffentlicht.
- Wir haben zur Analyse von Anomalien einen Ansatz entwickelt, der Aufrufe ausgehend von der Anwendung bis hinein auf die Betriebssystemebene nachvollzieht.



## 2.4 Entwicklung adaptiver Algorithmen in funkbasierten Lokalisierungssystemen

### **Projektleitung:**

Prof. Dr. Michael Philippsen

### **Beteiligte:**

Dr.-Ing. Christopher Mutschler

**Laufzeit:** 15.5.2016–14.5.2018

### **Förderer:**

Fraunhofer Institut für Integrierte Schaltungen

### **Kontakt:**

Prof. Dr. Michael Philippsen

Tel.: +49-9131-85-27625

Fax: +49-9131-85-28809

E-Mail: michael.philippsen@fau.de

Ziel dieses Projektes ist die Entwicklung adaptiver Algorithmen für den Einsatz in Funklokalisierungssystemen. Im Rahmen dieses Projekts werden drei wesentliche Themen bearbeitet:

**Automatisierte Konfiguration der Ereignis-Detektoren.** In vorangegangenen Forschungsprojekten wurden die Grundlagen zur Analyse verrauschter Sensordatenströme gelegt. Allerdings bestand hierbei noch das Problem, dass Ereignis-Detektoren aufwändig und genau parametrisiert werden müssen, zufriedenstellende Ergebnisse zu produzieren. Dieses Arbeitspaket betrachtet Möglichkeiten einer automatisierten Konfiguration der Ereignis-Detektoren auf Basis vorhandener Ereignis- und Sensordatenströme.

In 2016 wurden erste Konzepte untersucht, um aus einer Vielzahl vorhandener Spieldaten die optimale Konfiguration der Ereignis-Detektoren zu bestimmen. Dabei wurden in einer Fußballanwendungen Spiele und Spielszenen durch Sportwissenschaftler manuell annotiert (z.B. Spieler A tritt Ball mit linkem Fuß zum Zeitpunkt  $t$ ). Diese manuell annotierten Spielszenen sollen später zur Optimierung der Parameter in der Hierarchie von Ereignisdetektoren herangezogen werden.

**Evaluierung von Methoden und Techniken des maschinellen Lernens für Anwendungen zur Lokalisierung.** In vorhergehenden Forschungsprojekten wurden bereits erste Algorithmen des maschinellen Lernens im Kontext funkbasierter Lokalisierungssysteme entwickelt (z.B. evolutionäre Algorithmen zur Bestimmung von Antennenpositionen und -ausrichtungen). Im Rahmen dieses Arbeitspakets werden weitere Ansätze untersucht, um Lokalisierungssysteme durch derartige Methoden zu unterstützen.

Im Jahr 2016 wurden erste Ansätze evaluiert, um Teile der Positionsrechnung laufzeit-basierter Funklokalisierungssysteme durch Methoden des maschinellen Lernens zu ersetzen. Bislang werden die Rohdaten solcher Systeme durch eine Signalverarbeitungskette (Analog-Digital-Wandlung, Ankunftszeitbestimmung, Kalman-Filterung, Bewegungsanalyse) zu einer Position verrechnet. Dies erfordert einen vergleichsweise hohen Installations- und Konfigurationsaufwand für die Inbetriebnahme eines Lokalisierungssystems in der Zielumgebung und für die Zielanwendung.

**Evaluierung bildgebender Verfahren zur Unterstützung funkbasierter Lokalisierungssysteme.** Funkbasierte Lokalisierungssysteme können ihre Stärken gegenüber Kamera-basierten Lokalisierungssystemen immer dann ausspielen, wenn es zu Verdeckungen von Objekten kommen kann. Im Gegenzug haben funkbasierte Systeme Probleme mit metallischen Aufbauten/Oberflächen, da die Funkwellen an metallischen Oberflächen reflektiert werden und damit über mehrere Pfade an den Empfangsantennen empfangen werden. In diesem Arbeitspaket sollen Algorithmen für eine Bild-basierte Ortungskomponente entwickelt werden, um Funk-Lokalisierungssysteme bei der Positionsrechnung zu unterstützen.

In 2016 wurde damit begonnen zwei unterschiedliche Systeme zu entwickeln: CNN-Lok, ein System zur kamerabasierten Eigenlokalisierung von Objekten (sog. inside-out tracking), sowie InfraLok ein System zur Lokalisierung mit Kameras in der Infrastruktur (sog. outside-in tracking) auf Infrarot-Basis. CNNLok nutzt ein Convolutional Neural Network, trainiert auf einer Vielzahl von in der Zielumgebung erstellter Kamerabilder. Das Netzwerk liefert anschließend zu einem aktuellen Kamerabild die Position der Kamera (bzw. des Objektes) im Raum. InfraLok detektiert Infrarot-LEDs über ein Multi-Kamerasystem und ermittelt deren Position im Raum.

## 2.5 Inkrementelle Code-Analyse

### **Projektleitung:**

Prof. Dr. Michael Philippsen

### **Beteiligte:**

Dipl.-Math. Jakob Krainz

PD Dr. Ronald Veldema

**Laufzeit:** 1.4.2012–30.6.2017

Um sicherzustellen, dass Fehler im Programmdesign schon früh im Entwicklungsprozess gefunden werden, ist es nützlich, Fehler möglichst schon während des Editierens des Programms zu finden. Dazu sollte die verwendete Analyse so schnell sein, dass ein interaktiver Einsatz möglich ist. Eine Möglichkeit, dies umzusetzen, ist der Einsatz von inkrementeller Analyse, bei der die Analyseergebnisse von Teilen

eines Programms zu Gesamtergebnissen kombiniert werden. Vorteil von inkrementeller Programmanalyse ist, dass bei kleineren Änderungen ein großer Teil der Analyseergebnisse wieder verwendet werden kann, wie auch z.B. Analyseergebnisse von u.U. verwendeten Programmbibliotheken. Hierdurch kann der Analyseaufwand drastisch reduziert werden, wodurch die Analyse interaktiv nutzbar wird.

Unsere Analyse basiert darauf, für (Teile von) einzelnen Funktionen zu bestimmen, welche Auswirkungen die Ausführung auf den Zustand des Programms zur Laufzeit haben kann. Hierzu wird der Laufzeitzustand eines Programms abstrakt durch einen Graphen dargestellt, der die im Speicher befindlichen Variablen und Objekte und ihre Verzeigerung beschreibt. Die Funktion wird symbolisch ausgeführt und dabei wird bestimmt, welche Änderungen an dem Laufzeitzustand bzw. an dem diesen darstellenden Graphen verursacht werden können. Um die Effekte von Hintereinanderausführung von Programmteilen, Funktionsaufrufen, Schleifen, etc. zu bestimmen, können die Änderungsbeschreibungen der Programmteile dann zu größeren Änderungsbeschreibungen kombiniert werden. Die Analyse geht dabei Bottom-Up vor, analysiert also eine aufgerufene Funktion vor der aufrufenden Funktion (wobei Rekursion analysierbar ist).

In 2016 lag der Schwerpunkt der Forschung darauf, die eingesetzten Algorithmen und Datenstrukturen weiter zu entwickeln. Im Fokus lagen hierbei zum einen die Skalierbarkeit der Analyse bis hin zu Programmen mit mehr als 1 Mio. Anweisungen und zum anderen die inkrementelle Analyse, bei der die Analyseergebnisse von unveränderten Programmteilen weiterverwendet werden, was die Analyse bei typischen Software-Entwicklungspraktiken (großes Code-Volumen, meist sehr kleine Änderungen) deutlich beschleunigt.

## **2.6 International Collegiate Programming Contest an der FAU**

### **Projektleitung:**

Prof. Dr. Michael Philippsen

### **Beteiligte:**

Dipl.-Inf. Daniela Novac

Michael Baer, M. Sc.

Dipl.-Math. Jakob Krainz

Dipl.-Inf. Tobias Werth

**Beginn:** 1.11.2002

### **Kontakt:**

Dipl.-Inf. Daniela Novac

Tel.: +49-9131-85-27622

Fax: +49-9131-85-28809

E-Mail: [daniela.novac@fau.de](mailto:daniela.novac@fau.de)

Die Association for Computing Machinery (ACM) richtet seit Jahrzehnten den International Collegiate Programming Contest (ICPC) aus. Dabei sollen Teams aus je drei Studenten in fünf Stunden neun bis elf Programmieraufgaben lösen. Als Erschwernis kommt hinzu, dass nur ein Computer pro Gruppe zur Verfügung steht. Die Aufgaben erfordern solide Kenntnisse von Algorithmen aus allen Gebieten der Informatik und Mathematik, wie z.B. Graphen, Kombinatorik, Zeichenketten, Algebra und Geometrie.

Der ICPC wird jedes Jahr in drei Stufen ausgetragen. Zuerst werden innerhalb der Universitäten in lokalen Ausscheidungen die maximal drei Teams bestimmt, die dann zu den regionalen Wettbewerben entsandt werden. Erlangen liegt seit dem Jahr 2009 im Einzugsbereich des Northwestern European Regional Contest (NWERC), an dem u.a. auch Teams aus Großbritannien, den Benelux-Staaten und Skandinavien teilnehmen. Die Sieger aller regionalen Wettbewerbe der Welt (und einige Zweitplatzierte) erreichen die World Finals, die im Frühjahr des jeweils darauffolgenden Jahres (2017 in Rapid City, South Dakota, USA) stattfinden.

Im Jahr 2016 fanden zwei lokale Wettbewerbe an der FAU statt. Im Wintersemester wurde ein Mannschaftswettbewerb ausgetragen mit dem Ziel, neue Studierende für die Wettbewerbe zu begeistern - es meldeten sich 35 Erlanger Teams an. Jedes Team bestand aus maximal drei Studenten. Außerdem nahmen noch 26 Teams von anderen europäischen Universitäten teil. Im Sommersemester fand vor dem Wettbewerb zum wiederholten Mal das Hauptseminar "Hallo Welt! - Programmieren für Fortgeschrittene" statt, um Studierende verschiedener Fachrichtungen in Algorithmen und Wettbewerbsaufgaben zu schulen. Der Wettbewerb im Sommersemester diente danach der Auswahl der studentischen Vertreter der FAU für den NWERC 2016 in Bath (GB). Insgesamt nahmen an dem deutschlandweit organisierten Ausscheidungskampf 21 Teams der FAU mit Studenten verschiedenster Fachrichtungen teil.

Aus den besten Teams und gegeben der Altersbegrenzung, wurden neun Studenten ausgewählt, die für den NWERC Dreierteams bildeten. Diese erreichten dann beim NWERC in Bath, UK die Plätze 20, 45 und 59 von insgesamt 114 teilnehmenden Teams und somit eine solide Platzierung. Insbesondere in Anbetracht der vielen Erstteilnehmer ohne Wettbewerberfahrung, sind die Ergebnisse ein guter Anfang für die nächsten Jahre.

## **2.7 Parallele Code-Analyse auf einer GPU**

### **Projektleitung:**

Prof. Dr. Michael Philippsen

### **Beteiligte:**

Dipl.-Inf. Thorsten Blaß

PD Dr. Ronald Veldema

**Beginn:** 1.7.2013

Im Übersetzerbau (und auch an anderen Stellen) gibt es Analyseverfahren, bei denen Informationen solange durch einen Graph propagiert und dabei verändert werden, bis sich das Analyseergebnis als Fixpunkt einstellt.

Ziel dieses Projektes ist ein Programmrahmen, in dem verschiedene derartige Verfahren parallel und dadurch schneller auf der Graphikkarte ablaufen können.

Der Forschungsschwerpunkt des Jahres 2016 lag auf der Weiterentwicklung eines **Synchronisationsmechanismus für GPUs**. Bekannte Synchronisationsverfahren für die CPU (z. B. Spin-Lock) können nicht ohne weitere Anpassung auf der GPU verwendet werden, weil sie aufgrund spezieller Eigenschaften der GPU-Architektur zu Dead- bzw. Livelocks führen. Synchronisation wird jedoch (auch bei vorwiegend datenparallelen Graphimplementierungen) benötigt, wenn sich Abhängigkeiten dynamisch ergeben. Der im Projekt entwickelte GPU-Synchronisationsmechanismus löst zwei wesentliche, auf GPUs nicht-triviale Probleme: Erstens verhindern wir Dead- bzw. Livelocks. Zweites erreichen wir einen maximalen Parallelitätsgrad, indem datenparallele Threads, die an nicht-kollidierenden Stellen der Datenstruktur arbeiten, nicht blockiert werden sondern die Datenstruktur zeitgleich verändern können. Beispiele sind Threads, die disjunkte Stellen eines Graphen modifizieren, ohne die strukturelle Integrität des Graphen zu beeinflussen. Bei unserem Ansatz hat der Programmierer die Möglichkeit, Regeln zu formulieren, unter welchen Umständen eine derartige parallele Ausführung eines kritischen Abschnitts erlaubt ist. Zur Laufzeit wird dann geprüft, welcher Grad an Parallelität ausgenutzt werden kann.

In den folgenden Arbeitsschritten wird der Synchronisationsmechanismus um einen Ablaufplaner erweitert, der kollidierende Zugriffe auf eine Datenstruktur so umsortiert, dass die SIMD-Ausführungsweise der GPU weniger Serialisierung bedingt, als ohne diese Umsortierung. Dadurch steigt der Parallelitätsgrad. Die zugrundeliegende Idee nutzt aus, dass GPUs Threads in einer Hierarchie von Organisationseinheiten verwalten. Stellt der oben beschriebene Synchronisationsmechanismus einen kollidierenden Zugriff auf einer Hierarchieebene fest, wird auf der nächstkleineren Organisationseinheit erneut auf das Vorhandensein des Konflikts geprüft. Falls dieser dort nicht besteht, dann ist eine parallele Ausführung von wenigen Threads möglich, was immer noch besser ist als die größere Hierarchieebene sequentiell auszuführen. Ziel des Ablaufplaners ist es dann, die entdeckten Kollisionen so über die Organisationseinheiten zu verteilen, dass möglichst viele Threads parallel ausgeführt werden können. Da diese Ablaufplanung zur Laufzeit ausgeführt wird, muss sie effizient und selbst parallel sein sowie evtl. ausnutzen, dass neuere GPUs zur Laufzeit weitere Threads dynamisch starten können.

## 2.8 Software Watermarking

### **Projektleitung:**

Prof. Dr. Michael Philippsen

### **Beteiligte:**

Dipl.-Inf. Daniela Novac

**Beginn:** 1.1.2016

Unter Software Wasserzeichen versteht man das Verstecken von ausgewählten Merkmalen in Programme, um sie entweder zu identifizieren oder zu authentifizieren. Das ist nützlich im Rahmen der Bekämpfung von Softwarepiraterie, aber auch um die richtige Nutzung von Open-Source Projekten (wie zum Beispiel unter der GNU Lizenz stehende Projekte) zu überprüfen. Die bisherigen Ansätze gehen davon aus, dass das Wasserzeichen bei der konzeptionellen Entwicklung des Codes hinzugefügt wird und benötigen somit das Verständnis und den Beitrag der Programmierer für den Einbettungsprozess. Ziel unseres Forschungsprojekts ist es ein Wasserzeichen-Framework zu entwickeln, dessen Verfahren automatisiert beim Übersetzen des Programms Wasserzeichen hinzufügen, auch für existierende Programme. Als ersten Ansatz untersuchen wir eine Wasserzeichenmethode, die auf einer symbolischen Ausführung und anschließender Funktionssynthese basiert.

## 3 Publikationen 2016

- Dotzler, Georg ; Philippsen, Michael: Move-Optimized Source Code Tree Differencing . In: ACM (Hrsg.) : Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016) (31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016) Singapore 03.-07.09.2016). New York, NY, USA : ACM, 2016, S. 660-671. - ISBN 978-1-4503-3845-5
- Edelhäuser, Thorsten ; Frühauf, Hans Holm ; Philippsen, Michael ; Kókai, Gabriella ; Nilson, Jörg: Concept for encoding data defining coded positions representing a trajectory of an object . Schutzrecht US 9.384.387 B2 Patentschrift (05.07.2016)
- Gradl, Stefan ; Eskofier, Björn ; Eskofier, Dominic ; Mutschler, Christopher ; Otto, Stephan: Virtual and augmented reality in sports: an overview and acceptance study . In: ACM (Hrsg.) : Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'16): Adjunct (2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing

(UbiComp'16) Heidelberg, Germany September 12 - 16, 2016). 2016, S. 885-888. - ISBN 978-1-4503-4462-3

- Kreutzer, Patrick ; Dotzler, Georg ; Ring, Matthias ; Eskofier, Björn ; Philippsen, Michael: Automatic clustering of code changes . In: ACM (Hrsg.) : Proceedings of the 13th International Workshop on Mining Software Repositories (MSR 2016) (13th International Workshop on Mining Software Repositories Austin, Texas 14.-15. Mai 2016). New York, USA : ACM, 2016, S. 61-72. - ISBN 978-1-4503-4186-8
- Lautenschlager, Florian: Chronix - A fast and efficient time series storage based on Apache Solr .Vortrag: Open Source Data Center Conference (OSDC 2016), Berlin, Germany, 2016
- Lautenschlager, Florian ; Kammerer, Moritz: Chronix as Long Term Storage for Prometheus .Vortrag: CloudNativeCon, Seattle, WA, 08.11.2016
- Lautenschlager, Florian: The new time series kid on the block .Vortrag: Apache Con: Big Data, Vancouver, Canada, 2016
- Mohammad Alawieh ; Niels Hadaschik ; Norbert Franke ; Mutschler, Christopher: Inter-Satellite Ranging in the Low Earth Orbit . In: IEEE (Hrsg.) : Proceedings of the 10th IEEE/IET International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP'16) (10th IEEE/IET International Symposium on Communication Systems, Networks & Digital Signal Processing Prague, Czech Republic 20.-22.07.2016). 2016, S. 1-6. - ISBN 978-1-5090-2526-8
- Mutschler, Christopher ; Feigl, Tobias ; Daxer, Christian ; Otto, Stephan ; Bercea, Cosmin-Ionut: Verfahren zum Einstellen einer Blickrichtung in einer Darstellung einer virtuellen Realität . Schutzrecht DE 10 2016 109 153.7 Patentanmeldung (18.05.2016)
- Sackenreuter, Benjamin ; Hadaschik, Nils ; Fassbinder, Marc ; Mutschler, Christopher: Low-Complexity PDoA-based Localization . In: IEEE (Hrsg.) : Proceedings of the 7th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2016) (7th International Conference on Indoor Positioning and Indoor Navigation (IPIN 2016) Madrid, Spain 4.10.-7.10.2016). 2016, S. 1-6. - ISBN 978-1-5090-2425-4

## 4 Studien- und Abschlussarbeiten

- Bachelor Thesis: Automatische Generierung von Joern-Abfragen. Bearbeiter: Simon Rainer (beendet am 01.03.2016); Betreuer: Dipl.-Inf. Georg Dotzler; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen
- Bachelor Thesis: Konzeption und Implementierung eines Benchmarks für Zeitreihendatenbanken auf Basis operationaler Laufzeitdaten. Bearbeiter: Max Wehner (beendet am 08.04.2016); Betreuer: Prof. Dr. Michael Philippsen; Florian Lautenschlager, M. Sc.
- Bachelor Thesis: Verbesserte Erkennung von Laufzeitanomalien zur automatisierten Ursacheneingrenzung. Bearbeiter: Michael Wiesenbauer (beendet am 08.04.2016); Betreuer: Andreas Kumlehn, M. Sc.; Prof. Dr. Michael Philippsen
- Studien-/Bachelor-/Diplom-/Masterarbeit: Optimierung eines Verfahrens zur symbolischen Ausführung von Code-Fragmenten. Bearbeiter: Lukas Spranger (beendet am 19.04.2016); Betreuer: Dipl.-Inf. Georg Dotzler; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen
- Studien-/Bachelor-/Diplom-/Masterarbeit: Beschleunigung von Propagationsalgorithmen mittels Pfadkompression auf der GPU. Bearbeiter: Andreas Lindstedting (beendet am 22.04.2016); Betreuer: Dipl.-Inf. Thorsten Blaß; PD Dr. Ronald Veldema; Prof. Dr. Michael Philippsen
- Master Thesis: Automatische Optimierung der Granularität von rekursiven task-parallelen Programmen auf Vielkernrechnern. Bearbeiter: Daniel Schmidt (beendet am 02.05.2016); Betreuer: Prof. Dr. Michael Philippsen; Dipl.-Inf. Tobias Werth
- Bachelor Thesis: Erweiterung einer computergestützten Diagnose von Laufzeitanomalien. Bearbeiter: Moritz Müller (beendet am 04.05.2016); Betreuer: Andreas Kumlehn, M. Sc.; Prof. Dr. Michael Philippsen
- Master Thesis: Automatische Parallelisierung von rekursiven Programmen auf Vielkernrechnern mittels spekulativer Ausführung. Bearbeiter: Frederik Simon (beendet am 01.06.2016); Betreuer: Prof. Dr. Michael Philippsen; Dipl.-Inf. Tobias Werth
- Bachelor Thesis: Inkrementelles Clustering ähnlicher Quelltext-Änderungen. Bearbeiter: Martin Endrizzi (beendet am 08.06.2016); Betreuer: Patrick Kreutzer, M. Sc.; Dipl.-Inf. Georg Dotzler; Prof. Dr. Michael Philippsen



- Bachelor Thesis: Entwicklung eines Simulink Targets. Bearbeiter: Thorsten Schwachhofer (beendet am 04.10.2016); Betreuer: Dipl.-Inf. Thorsten Blaß; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen
- Bachelor Thesis: Erweiterung (und Beschreibung) eines parametrisierbaren skriptfähigen Web Interfaces. Bearbeiter: Markus Straussberger (beendet am 04.10.2016); Betreuer: Dipl.-Inf. Thorsten Blaß; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen
- Bachelor Thesis: Implementierung des Kerns eines Konverters für grafische Sprachen von Speicherprogrammierbaren Steuerungen. Bearbeiter: Michael Sammler (beendet am 04.10.2016); Betreuer: Dipl.-Inf. Thorsten Blaß; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen
- Bachelor Thesis: Implementierung und Evaluierung einer Hierarchie von Verfahren zum semantischen Vergleich von Code-Fragmenten. Bearbeiter: Adrian Kretschmer (beendet am 03.11.2016); Betreuer: Patrick Kreutzer, M. Sc.; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen
- Bachelor Thesis: Entwicklung eines Benchmarks zur Bewertung von Zeitreihendatenbanken. Bearbeiter: Sebastian Thürauf (beendet am 30.11.2016); Betreuer: Florian Lautenschlager, M. Sc.; Prof. Dr. Michael Philippsen
- Master-Projekt: Ein E-Übersetzer in Rust. Bearbeiter: Sebastian Hahn (beendet am 14.12.2016); Betreuer: Dipl.-Math. Jakob Krainz; Dipl.-Inf. Tobias Werth; Prof. Dr. Michael Philippsen