Jahresbericht 2015 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:
Dipl.-Inf. Thorsten Blaß
Dipl.-Inf. Daniel Brinkers
Dipl.-Inf. Georg Dotzler
Marius Kamp, M. Sc.
Dipl.-Math. Jakob Krainz
Patrick Kreutzer, M. Sc. (ab 01.12.2015)
Andreas Kumlehn, M. Sc.
Dr.-Ing. Christopher Mutschler
Dr.-Ing. Norbert Oster
Norbert Tausch, M. Eng. (bis 30.09.2015)
PD Dr. Ronald Veldema (bis 15.09.2015)
Dipl.-Inf. Tobias Werth
IT-Betreuer:
Dipl.-Ing. (FH) Helmut Allendorf
Manfred Uebler
Gast:
Dr.-Ing. Josef Adersberger
Dr.-Ing. Martin Jung
Florian Lautenschlager, M. Sc.
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.

In den vergangenen zwei Jahren nahmen wir aber eine **thematische Neuausrichtung** auf Forschungsthemen vor, 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 nun 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 erratischem 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.

Bei der vorgenommenen thematischen Neuausrichtung von einem Blick, der vorwiegend auf die systemtechnischen Fragen gerichtet war, zu einem Blick in Richtung des Software-Engineerings bleibt viel Etabliertes an Bord. Der Lehrstuhl arbeitet auch weiterhin **Programm-Code-basiert** und erstellt funktionsfähige **Prototypen** der erforschten Werkzeuge. Wie bisher sind uns 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.
PD Dr. Ronald Veldema
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 2015 wurden fünf Optimierungen für solche baumbasierten Verfahren entwickelt, die die Genauigkeit der Änderungsbestimmung verbessern.

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.

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 Ähnlichkeits-Vergleich 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 mit dem 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.

Die aktuellen Ergebnisse von unserem Werkzeug SIFE finden Sie unter:
https://www2.cs.fau.de/staff/dotzler/SIFE_results.tar.gz

2.2 Automatische Code-Parallelisierung zur Laufzeit

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
PD Dr. Ronald Veldema
Dipl.-Inf. Daniel Brinkers
Laufzeit: 1.1.2011–30.4.2016

Aktuell konzentrieren wir uns bei der automatisierten Parallelisierung auf Laufzeit-Parallelisierung. Das zu parallelisierende Programm wird während seiner Ausführung auf Schleifen untersucht, die parallel ausführbar sind. Unser Ansatz besteht darin, laufzeitintensive Schleifen zunächst in zwei Durchläufen auf ihre Parallelisierbarkeit zu untersuchen. Die Analysedurchläufe werden parallel zueinander und parallel zu einer sequenziellen Ausführung der Schleife ab der ersten Iteration ausgeführt. Im ersten Analysedurchlauf werden die Adressen aller Schreibzugriffe auf den Hauptspeicher in einer gemeinsam genutzten Datenstruktur gespeichert. Zugriffe auf diese Datenstruktur müssen nicht synchronisiert werden, wenn sichergestellt wird, dass bei konkurrierenden Schreibzugriffen überhaupt ein Wert geschrieben wird. Im zweiten Durchlauf wird für jeden Speicherzugriff (auch für lesende!) überprüft, ob eine Datenabhängigkeit zu einem Schreibzugriff besteht. Damit die sequenzielle Ausführung parallel zur Analyse gestartet werden kann, muss diese die Speicherzugriffe, die sie durchführt, protokollieren und überprüfen. Falls keine Datenabhängigkeiten gefunden werden, wird die Schleife parallel ausgeführt. Anderenfalls wird die sequentielle Ausführung ohne weitere Instrumentisierung fortgeführt.

Im Jahr 2015 haben wir begonnen, das Verfahren in eine bestehende Javascript-Engine einzubauen. Die Wahl ist auf Javascriptcore aus Webkit gefallen, weil dort in der letzten Optimierungsstufe LLVM verwendet wird. Damit ist es einfacher möglich, das Verfahren auch für andere LLVM Sprachen zu verwenden, oder auf Analysen zurückzugreifen, die für LLVM entwickelt wurden. Javascriptcore verwendet ein mehrstufiges JIT Verfahren. In der letzten (am meisten optimierten) Stufe wird LLVM Zwischencode er-

zeugt, der dann von dem LLVM-Compiler mit den dort verfügbaren Optimierungen übersetzt wird. An diesen Punkt bauen wir unsere Parallelisierung ein. Wenn diese Optimierungsstufe erreicht ist, ist sichergestellt, dass der Code oft ausgeführt wird und sich unser aufwändiges Verfahren lohnt. Wir setzen unser Verfahren vor der Erzeugung des LLVM Code an, um Zugriff auf die JIT-Protokolldaten zu haben, die uns bei der Entscheidung unterstützen, welche Schleifen parallelisiert werden sollen.

Das Projekt stellt einen Beitrag des Lehrstuhls Informatik 2 zum IZ ESI dar, siehe auch http://www.esi.uni-erlangen.de .

2.3 Design for Diagnosability

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Andreas Kumlehn, M. Sc.
Dr.-Ing. Josef Adersberger
Florian Lautenschlager, M. Sc.
Beginn: 15.5.2013
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) 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.

Unsere wesentlichen Beiträge aus dem Jahr 2015 sind:

- Wir haben Chronix weiterentwickelt und es in einem Open Source Projekt (www.chronix.io) veröffentlicht.
- Wir einen neuartigen Benchmark für Zeitreihendatenbanken konzipiert und umgesetzt.
- Die Software-Blackbox haben wir weiter optimiert und den Overhead durch einen neuartigen Ansatz mit konstantem Speicherbedarf reduziert.
- Die Dynamic Analysis Workbench haben wir mit einem Plug-In-Konzept ausgestattet und Detektoren zur Erkennung von auffälligen Zeitreihen umgesetzt.

2.4 Effiziente Software-Architekturen für verteilte Ereignisverarbeitungssysteme

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Dr.-Ing. Christopher Mutschler
Laufzeit: 15.11.2010–31.12.2015
Förderer:
Fraunhofer Institut für Integrierte Schaltungen

Funkortungssysteme, auch bekannt als Real-Time Location Systems (RTLS), geraten immer mehr in den Fokus der Logistik, Produktion und vieler weiterer Prozesse. Diese Systeme liefern wertvolle Informationen über den Aufenthaltsort von beteiligten Objekten zur Laufzeit. Damit können Prozesse verfolgt, analysiert und optimiert werden. Neben den Forschungsbereichen an der Basis von Ortungssystemen, wie robuste und störsichere Ortungstechnologien oder Verfahren zur hochgenauen Positionsbestimmung, rücken mehr und mehr Methoden in den Vordergrund, die aus Positionsdatenströmen wertvolle Informationen für weitere Verarbeitungsstufen gewinnen. In diesem Kontext erforscht das Projekt Verfahren zur Ereignisdetektion in Positionsdatenströmen zur Laufzeit.

2011 wurde damit begonnen, auftretende Ereignisse in Lokalisierungssystemen zu erkennen und vorherzusagen. Hierfür werden Ereignisströme zur Laufzeit analysiert und ausgewertet. Somit konnten Modelle erlernt werden, um Ereignisse aus Ereignisströmen zu prädizieren.

2012 wurden für die Laufzeitanalyse von Positionsdatenströmen mehrerer Methoden entwickelt, um Ereignisse mit möglichst geringer Latenz detektieren zu können. Hierbei können einzelne Teilereignisse durch sog. Ereignisdetektoren dazu verwendet werden, höhere Zusammenhänge in den Daten hierarchisch zusammenzusetzen. Hierdurch wird die Komplexität der einzelnen Detektionskomponenten drastisch reduziert. Diese werden somit wartbarer und durch die Ausnutzung paralleler und verteilter Rechnerstrukturen wesentlich effizienter. Es ist nun möglich, Ereignisse in den Positionsdatenströmen innerhalb von nur einigen hundert Millisekunden zu erkennen.

2013 wurde die Verzögerung v.a. verteilter Ereignisverarbeitungssysteme weiter minimiert und ein spezielles Migrationsverfahren entwickelt, das die Verteilung von Softwarekomponenten im laufenden Betrieb so modifizt, dass möglichst wenig zeitliche Verluste durch Netzwerkkommunikation auftreten. Des Weiteren wurde ein spekulatives Verfahren puffernder Ereignisverarbeitungssysteme zur Optimierung erarbeitet. Überschüssige Systemressourcen werden effizient verwendet, um die Latenz in Ereignisverarbeitungssystemen auf ein Minimum zu reduzieren. Es wurde außerdem ein repräsentativer Datensatz (mit Sensor- und Positionsdatenströmen sowie manuell eingefügten Ereignissen) sowie eine praxisrelevante Aufgabenstellung veröffentlicht.

2014 wurden grundlegende Verfahren zur Bewältigung von Unsicherheiten (bezüglich der Definition von Ereignisdetektoren) und Unschärfe (im Sinne von Ungenauigkeiten innerhalb von Ereignissen an sich) untersucht. Im Rahmen des Forschungsprojekts wurde ein vielversprechender Ansatz verfolgt, der ohne den üblichen Determinismus ereignisverarbeitender Systeme auskommt und stattdessen parallele Berechnungspfade verfolgt und dabei durch die parallel Betrachtung mehrerer möglicher Zustände eine robustere und genauerere Ereignisverarbeitung erreicht. Dabei kann ein Anwendungs-

entwickler Wahrscheinlichkeiten oder Funktionen hinterlegen, welche die Ereignisdetektoren bzw. die generierten Ereignisse parametrisieren.

Dieses Verfahren wurde 2015 weiter verbessert, optimiert und veröffentlicht. Des Weiteren wurden erste Ansätze zum Erlernen optimaler Parameterkonfigurationen für Ereignisdetektoren untersucht. Damit wird eine manuelle Einstellung und Optimierung von Threshold-Parametern innerhalb der Detektoren unnötig.

Das Projekt stellt einen Beitrag des Lehrstuhls Informatik 2 zum IZ ESI (http://www.esi-anwendungszentrum.de) dar.

2.5 Inkrementelle Code-Analyse

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Dipl.-Math. Jakob Krainz
PD Dr. Ronald Veldema
Beginn: 1.4.2012

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 bestimmmen, 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 2015 lag der Schwerpunkt der Forschung darauf, die eingesetzten Algorithmen und Datenstrukturen weiter zu entwickeln. So konnte die für die Analyse eines gegebenen Programmes benötigte Laufzeit deutlich verringert werden. Zum anderen dürfen die zu analysierenden Programme mehr und mächtigere Features und Elemente der Programmiersprache benutzen.

2.6 Interaktive und parallele Code-Analyse

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 Programmrahmen, in dem verschiedene derartige Verfahren parallel und dadurch schneller auf der Graphikkarte ablaufen können. Zur Erreichung dieses Ziels sind folgende Teilfragen in Arbeit:

Effiziente Graphimplementierungen auf der GPU

Übersetzer nutzen Graphstrukturen, um das zu analysierende Programm intern geeignet darstellen zu können (z. B. AST, Kontrollflussgraphen, Datenflussgraphen). Auch Analysen verwenden Graphstrukturen (z. B. Alias- Analyse), um ihre Ergebnisse zu speichern. In diesem Teilprojekt wurde untersucht, welche Graphimplementierungen eine besonders effiziente Verwendung auf der GPU erlauben.

Das Problem besteht darin, dass bei der erwähnten Alias-Analyse mehrere Threads parallel Informationen in einen Graphen einfügen oder diesen ändern, was zu Leistungseinbußen führt. Um diese so weit wie möglich zu reduzieren, wurden im Bereichszeitraum 2015 zwölf unterschiedliche Implementierungen von Graphen erarbeitet und überprüft, für welche Anwendungsfälle sie sich jeweils eignen (häufige Lesezugriff, viele Kanten/Knoten werden hinzugefügt, Knoten werden häufig entfernt, ...). Die Implementierungen unterscheiden sich in den Datenstrukturen zum Speichern der Knoten bzw. Kanten und den verwendeten Lock-Mechanismen. Die Ergebnisse dieser Arbeit dienen dazu, geeignete Graphstrukturen für das jeweilige Problem herauszusuchen, um

größtmögliche Effizienz auf der GPU zu erreichen. Eine Veröffentlichung der Ergebnisse ist in Arbeit.

Synchronisationsmechanismen auf der GPU

Ein weiterer Forschungsschwerpunkt des Jahres 2015 war die Erforschung von Synchonisationsmechanismen auf der GPU. Bekannte Synchronisationsverfahren für die CPU (z. B. Spin-Lock) können nicht ohne weitere Anpassung auf der GPU verwendet werden. Die Ursache hierfür liegt in den speziellen Eigenschaften der GPU-Architektur. Eine naive Verwendung von CPU Synchronisationsmechanismen auf der GPU führt zu Deadlocks. Synchronisation wird jedoch häufig zum gegenseitigen Ausschluss benötigt, z.B. bei Graphimplementierungen. Am Lehrstuhl wurde ein neuer Synchronisationsmechanismus entwickelt, der erstens Deadlocks verhindert und zweitens einen möglichst hohen Grad an Parallelität erhält, indem Threads, die an nicht kollidierenden Stellen der Datenstruktur arbeiten, nicht blockiert werden. Beispiele sind Threads, die disjunkte Stellen eines Graphen modifizieren können, ohne die strukturelle Integrität des Graphen zu beeinflussen. Der Programmierer hat die Möglichkeit, Regeln zu formulieren, unter welchen Umständen eine parallele Ausführung eines kritischen Abschnitts möglich ist. Zur Laufzeit wird geprüft, welcher Grad an Parallelität ausgenutzt werden kann. Mit Hilfe der Laufzeittests kann entschieden werden, wie viele Threads parallel ausgeführt werden können. Eine Veröffentlichung dieses neuartigen Synchronisationsmechanismus ist in Arbeit.

2.7 International Collegiate Programming Contest an der FAU

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Dipl.-Inf. Tobias Werth
Dipl.-Inf. Daniel Brinkers
Dipl.-Math. Jakob Krainz
Beginn: 1.11.2002
Kontakt:
Dipl.-Inf. Tobias Werth
E-Mail: tobias.werth@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 (2015 in Marrakesh, Marokko) stattfinden.

Im Jahr 2015 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 2015. Insgesamt nahmen an dem deutschlandweit organisierten Ausscheidungskampf 21 Teams der FAU mit Studenten verschiedensten Fachrichtungen teil. Aus den besten Teams wurden neun Studenten ausgewählt, die für den NWERC Dreierteams bildeten.

Das beste Team der FAU löste beim nordwesteuropäischen Wettbewerb NWERC 2015 in Linköping 7 Aufgaben und erreichte damit den 10. Platz. Die weiteren Erlanger Teams erreichten mit sechs und sieben gelösten Aufgaben den 15. bzw. 25. Platz von 99 angetretenen Teams.

Im Mai trat ein Erlanger Team, das sich im Vorjahr mit dem zweiten Platz beim NWERC qualifiziert hat, bei den World Finals im Marrakesch, Marokko an. Es erreichte mit 6 gelösten Aufgaben den 58. Platz von 128 Teams aus aller Welt.

2.8 OpenMP/Java

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
PD Dr. Ronald Veldema
Dipl.-Inf. Georg Dotzler
Laufzeit: 1.10.2009–1.10.2015

JaMP ist eine Implementierung des bekannten OpenMP Standard für Java. JaMP erlaubt es (unter anderem) Schleifen zu parallelisieren, ohne sich mit der low-level Thread API von Java befassen zu müssen. Eine parallele Schleife hätte in JaMP folgende Form:

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

JaMP implementiert im Moment die Funktionalität von OpenMP 2.0 und Teile der Spezifikation 3.0 (z.B. die collapse clause). Die aktuelle JaMP Version erzeugt reinen Java Code und ist auf jeder JVM (ab Java 1.5) lauffähig. Die neueste Version kann sogar CUDA-fähige Hardware zur Ausführung von Schleifen verwenden, wenn der Schleifenrumpf eine Transformation nach CUDA möglich macht. Ist die Transformation nicht möglich, wird nebenläufiger Code für gängige Multicore-Prozessoren erzeugt. JaMP unterstütz auch die gleichzeitige Nutzung von mehreren Maschinenund Acceleratoren. Dieses wurde durch die Entwicklung von zwei Abstraktionsbibliotheken ermöglicht. Die untere Abstraktionsschicht bietet abstrakte Recheneinheiten, die von den eigentlichen Berechnungseinheiten wie CPUs und GPUs und ihrem Ort in einem Rechnerbündel abstrahieren. Eine weitere Abstraktionsschicht baut auf dieser Schicht auf und bietet Operationen zur Verwaltung partitionierter und replizierter Arrays. Ein partitioniertes Array wird dabei automatisch über die abstrakten Berechnungseinheiten verteilt, wobei die Geschwindigkeiten der einzelnen Berechnungseinheiten berücksichtigt werden. Welcher abstrakte Array-Typ für ein Array in einem Java-Programm konkret eingesetzt wird, wird vom JaMP-Übersetzer bestimmt, der erweitert wurde, um ein Programm entsprechend zu analysieren.

Im Jahr 2015 wurde das Task-Konzept (OpenMP 3.0) in JaMP integriert. Dadurch lassen sich rekursive Algorithmen mit JaMP parallelisieren.

2.9 Softwareleitstand

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Dr.-Ing. Josef Adersberger
Norbert Tausch, M. Eng.
Laufzeit: 1.11.2009–31.12.2015
Förderer:
Bundesministerium für Wirtschaft und Technologie
Kontakt:
Prof. Dr. Michael Philippsen
Tel.: +49-9131-85-27625
Fax: +49-9131-85-28809
E-Mail: michael.philippsen@fau.de

Prototypische Entwicklung eines neuartigen Werkzeugs zur Qualitätsabsicherung bei der Softwareentwicklung.

Moderne Softwaresysteme werden sowohl fachlich, technisch als auch organisatorisch zunehmend komplexer: So steigen die Anzahl und der Vernetzungsgrad der zu realisierenden Anforderungen pro System stetig, die technischen Vorgaben z.B. an den Verteilungsgrad und die Zuverlässigkeit der Systeme werden komplexer und die Softwareentwicklung selbst findet zunehmend in global verteilten Teams und mit wachsendem Zeitdruck statt. Aus diesen Gründen wird es auch zunehmend schwieriger, Softwareentwicklungsprojekte fachlich, technisch und organisatorisch zu steuern.

Als Softwareleitstand bezeichnen wir ein Werkzeug, das leitenden Projektrollen wie dem Projektleiter, dem Softwarearchitekten, dem Anforderungsarchitekten und dem Entwicklungsleiter eine hohe Transparenz und damit verbesserte Steuerbarkeit von Softwareentwicklungsprojekten ermöglicht.

Transparenz herrscht dann, wenn sowohl Zusammenhänge zwischen den vielfältigen Erzeugnissen eines Softwareentwicklungsprojekts als auchderen Eigenschaften schnell und gesamtheitlich zugänglich sind und entsprechend dem individuellen Informationsbedarf eines Projektbeteiligten aufbereitet sind.

Der Softwareleitstand ist ein Werkzeug, das den Zugang zu den Zusammenhängen (Traceability) und den Eigenschaften (Metriken) der Erzeugnisse von Softwareentwicklungsprojekten vereinheitlicht. Damit kann die Effizienz von Softwareentwicklungsprojekten maßgeblich gesteigert werden. Es sollen Erzeugnisse des Softwareentwicklungsprojekts (Artefakte) und ihre Zusammenhänge (Relationen), sowie zu den Artefakten zuordenbare Metriken zentral erfasst, integriert und analysiert werden können. Die ent-

sprechenden Analysen werden in Form von Visualisierungen des Artefaktgraphen mitsamt den zugeordneten Metriken und Regelprüfungen durchgeführt.

Das Projekt Softwareleitstand wird in Kooperation des Lehrstuhls mit der QAware GmbH München durchgeführt. Die ersten 30 Projektmonate wurden aus Mitteln des BMWi gefördert.

Die Umsetzung des Softwareleitstands erfolgte dabei in zwei Arbeitssträngen, die auch den beiden Subsystemen des Werkzeugs entsprechen: Der Integration Pipeline, die Traceability Informationen und Metriken aus verschiedensten Werkzeugen der Softwareentwicklung zusammen sammelt, sowie dem Analysis Core (Analysekern), der eine gesamtheitliche Auswertung der integrierten Daten ermöglicht.

Die Integration Pipeline wurde durch den Projektpartner QAware GmbH entwickelt. Dabei wurde zunächst eine Modellierungssprache für Traceability Informationen in Kombination mit Metriken (TraceML) definiert. Die Sprache besteht dabei aus einem Meta-Modell sowie einer Modellbibliothek zur einfachen Definition von angepassten Traceability Modellen. Aufbauend auf der TraceML wurde das Integration Pipeline Framework auf Basis des Eclipse Modeling Projekts entwickelt. Dabei wird sowohl das Eclipse Modeling Framework zur Abbildung der Modelle und Metamodelle, als auch die Modeling Workflow Engine zur Modelltransformation und Eclipse CDO als Modell-Repository verwendet. Auf Basis des Integration Pipeline Frameworks wurden dann eine Reihe von gängigen Werkzeugen der Softwareentwicklung wie z.B. Subversion, Eclipse, JIRA, Enterprise Architect und Maven angebunden.

Der Analysekern wurde durch den Lehrstuhl entwickelt. Zentrales Thema waren dabei die Konzeption und Realisierung einer domänenspezifischen Sprache für die graphbasierte Traceability-Analyse. Die Traceability Query Language (TracQL) reduziert den Aufwand zur Umsetzung von Traceability-Analysen zu reduzieren. TracQL erleichtert dabei sowohl die Extraktion als auch die Transformation der Traceability-Daten, so dass diese dann mittels kurzer funktional formulierter Graph-Traversierungen analysiert werden können. Die Sprache baut auf der multi-paradigmen Sprache Scala auf und wurde bereits mehrfach in realen Industrieprojekten zur Analyse erfolgreich eingesetzt.

Der Schwerpunkt des Jahres 2015 lag auf der Evaluation und Dokumentation des Ansatzes. Ziel war dabei, die zentralen Eigenschaften des Ansatzes hervorzuheben und deren Wirksamkeit nachzuweisen. Im Wesentlichen handelt es sich dabei um folgende drei Eigenschaften:

- Repräsentationsunabhängigkeit: TracQL ist an eine Vielzahl an Datenquellen anbindbar und macht deren Datentypen statisch typisiert verfügbar.
- Modularität: Der Ansatz ist sowohl strukturell als auch operational anpassbar und erweiterbar.

- Anwendbarkeit: Die Sprache besticht durch Ausdrucksstärke und Performanz im Vergleich zu anderen Ansätzen.

2.10 Übersetzerunterstützte Parallelisierung für Mehrkern-Architekturen

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Dipl.-Inf. Tobias Werth
Beginn: 1.1.2011
Kontakt:
Dipl.-Inf. Tobias Werth
E-Mail: tobias.werth@fau.de

Die Entwicklung von schnelleren und immer effizienteren Rechnerarchitekturen ist in den letzten Jahren an verschiedene Grenzen gestoßen. Althergebrachte Techniken trugen nicht mehr oder nur noch wenig zur Beschleunigung der Hardware bei. Grundprobleme sind dabei das auseinander driftende Verhältnis der Latenzen von Speicher und CPU und die Abwärme bei steigenden Taktfrequenzen. Als Lösung drängen sich homogene und heterogene Mehr- und Vielkern-Architekturen auf, die durch verringerte Taktfrequenzen und mehrschichtige Speicherhierarchien einen Großteil der genannten Problematik vermeiden. Unter Umständen wird mittels Spezialisierung einzelner Komponenten die Rechenleistung weiter erhöht. Aktuelle Zielarchitekturen sind dabei vor allem Grafikkarten mit Hunderten von Recheneinheiten und der Intel XeonPhi-Prozessor, der auf einem Board 60 Kerne inkl. Hyperthreading zur Verfügung stellt.

Während sich datenparallele Probleme mit Hilfe der neuen Architekturen meist relativ einfach beschleunigen lassen, ist die Umsetzung von auf Arbeitspaketen basierenden (sog. Task-parallelen) Probleme Gegenstand der aktuellen Forschung. Die Schwierigkeit ergibt sich dabei meist durch die Irregularität des entstehenden Task-Baumes und der damit verbundenen unterschiedlichen Ausführungsdauer. Dadurch ergeben sich die folgenden Fragestellungen: Welcher Kern führt welche Arbeitspakete in welcher Reihenfolge aus? Wann werden Arbeitspakete vom aktuellen Rechenknoten auf einen anderen Rechenknoten verschoben? Welche Daten gehören zu einem Arbeitspaket, können mehrere Kerne/Rechnerknoten parallel auf die Daten zugreifen? Wie müssen die Daten von verschiedenen Rechnerknoten wieder zusammengeführt werden? In welcher Form kann der Übersetzer in Zusammenarbeit mit dem Laufzeitsystem selbständig Arbeitspakete erzeugen und verteilen?

Im Jahr 2011 wurde in diesem Projekt das Programmiermodell Cilk für die heterogene CellBE-Architektur (ein PowerPC-Kern (PPU) mit acht SPU "Koprozessoren" zur Ausführung paralleler Berechnungen) implementiert und erweitert. Die CellBE-Architektur bietet sich aufgrund ihrer enormen Leistung auf einem einzelnen Chip als Zielarchitektur an. Um ein Arbeitspaket in der heterogenen Architektur verschieben zu können, haben wir das Cilk-Programmiermodell um ein weiteres Schlüsselwort ergänzt. Dazu entwickelten wir eine Quell-zu-Quelltext-Transformation, die aus einem Quelltext den entsprechenden Quelltext sowohl für die PPU als auch für die SPU erzeugen kann. Desweiteren wurden die entsprechenden Daten zu den Arbeitspaketen per DMA-Transfer passend in die lokalen Caches der Koprozessoren verschoben und nach Abarbeitung eines Teilbaumes auf den entfernten Koprozessoren der Speicher mit Hilfe eines automatischen Speicherbereinigers wieder freigegeben.

Im Jahr 2012 wurden Grafikkarten (GPUs) als weitere Zielarchitektur ins Auge gefasst, die heutzutage eine weitaus höhere Rechenleistung im Gegensatz zu normalen Prozessoren anbieten. Diese Rechenleistung kann durch die Programmierung mit Cuda (NVidia) oder OpenCL (AMD) aufgrund ihrer Struktur für datenparallele Probleme relativ einfach abgerufen werden. Weitaus schwieriger ist die Umsetzung Task-paralleler Anwendungen, die im weiteren Verlauf des Projekts untersucht werden sollen. Dazu werden verschiedene Lastausgleichsalgorithmen entworfen, prototypisch umgesetzt und miteinander verglichen werden. Im Jahr 2012 wurde ein erstes Verfahren mit mehrstufigen Warteschlangen und dem Prinzip der Arbeitsspende (work donation) entworfen.

Im Jahr 2013 wurden neben der Weiterentwicklung der Lastausgleichsalgorithmen für die Grafikkarte mit dem Intel XeonPhi-Prozessor die Gruppe der Zielarchitekturen weiter vergrößert. Der XeonPhi-Prozessor stellt mit seiner Vielzahl an Kernen und der großen Registerbreite und der damit verbundenen Vektorisierbarkeit neue Herausforderung an die Algorithmen zur Verteilung der Arbeitspakete. Konkret wurde das Cilk-Verfahren für den XeonPhi erweitert und angepasst, um automatisch während der Quellcode-Transformation Funktionen zu verschmelzen und die Möglichkeiten zur (automatischen) Parallelisierung durch den Intel-Compiler zu erhöhen. Dabei wurden mehrere Analysen implementiert, die nicht nur die Anzahl verschmelzbarer Funktionen erhöhen, sondern darüber hinaus auch das Auseinanderlaufen von Kontrollflusspfaden in den verschmolzenen Funktionen verhindern (bzw. zumindest abfangen).

Im Jahr 2014 wurde die vorhandene Implementierung der Lastausgleichsalgorithmen für den Intel XeonPhi-Prozessor so erweitert, dass die Arbeit auf mehrere XeonPhi-Prozessoren verteilt werden kann. Im Gegensatz zum Arbeitsdiebstahl, der weiterhin auf einer einzelnen Instanz des XeonPhi zur Lastverteilung zwischen den Kernen verwendet wird, wird die Arbeit aktiv an andere Prozessoren abgegeben. Mit einer neu entwickelten Annotation werden zur Abarbeitung des Arbeitspakets notwendige Daten gekennzeichnet und mit verschoben. Die Herausforderung dabei ist das Verschmelzen der Daten an Synchronisationspunkten. Desweiteren wurde begonnen, Cilk in den clang-

Übersetzer des LLVM-Frameworks einzubauen, um automatisiert den CUDA-Code zur Ausführung auf Grafikkarten zu erzeugen. Dieser Code soll dabei auf ein leichtgewichtiges Laufzeitsystem zur Ordnung und Ausführung der Arbeitspakete aufsetzen. Dazu werden Analysen implementiert, die Divergenz soweit möglich vermeiden sollen.

Im Jahr 2015 wurden verschiedene Lastverteilungsalgorithmen zur Auführung von Cilk-Programmen auf der Grafikkarte evaluiert und verglichen. Dazu wurden Warteschlangenalgorithmen mit parallelem Zugriff implementiert und die automatische Erzeugung des CUDA-Codes verfeinert. Da weiterhin das korrekte Setzen der Cilk-Schlüsselwörter zur Synchronisation eine Herausforderung für den Programmierer darstellt, werden zur Übersetzungszeit aus rekursivem Quelltext ohne Cilk-Annotationen ”plausible” Code-Varianten inkl. Synchronisationsanweisungen erzeugt. Diese Code-Varianten werden dann zur Laufzeit spekulativ ausgeführt und das Ergebnis der schnellsten, korrekten Variante für die weitere Berechnung verwendet. Desweiteren ist die Größe des Basisfalls der Rekursion entscheidend für den optimalen Geschwindigkeitsgewinn. Deswegen wurde begonnen, die Größe des Basisfalls durch Analysen zur Übersetzungs- und Laufzeit zu optimieren und rekursive Aufrufe durch Funktionseinbettung und Vektorisierung zu ersetzen.

3 Publikationen 2015

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

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

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

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

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

4 Studien- und Abschlussarbeiten

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

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

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

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

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

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

– Master Thesis: Implementierung eines statischen Ablaufplaners (”Schedulers”) innerhalb eines Übersetzers einer Sprache mit blockierenden Threads. Bearbeiter: Felix Sauer (beendet am 01.10.2015); Betreuer: PD Dr. Ronald Veldema; Dipl.-Inf. Thorsten Blaß; Prof. Dr. Michael Philippsen

– Studien-/Bachelor-/Diplom-/Masterarbeit: Untersuchung von Fehlertoleranz durch Software. Bearbeiter: Jean-Christopher Jäger (beendet am 01.10.2015); Betreuer: PD Dr. Ronald Veldema; Dipl.-Inf. Thorsten Blaß; Prof. Dr. Michael Philippsen

– Studien-/Bachelor-/Diplom-/Masterarbeit: Evaluation of Machine Learning Algorithms for Outlier Detection in Clustered Code Fragments. Bearbeiter: James Wafula (beendet am 30.10.2015); Betreuer: Dipl.-Inf. Georg Dotzler; Matthias Ring, M. Sc.; Prof. Dr. Björn Eskofier; Prof. Dr. Michael Philippsen

– Studien-/Bachelor-/Diplom-/Masterarbeit: Clustering von ähnlichen Code-Fragmenten. Bearbeiter: Patrick Kreutzer (beendet am 11.11.2015); Betreuer: Dipl.-Inf. Georg Dotzler; Matthias Ring, M. Sc.; Prof. Dr. Björn Eskofier; Prof. Dr. Michael Philippsen

– Studien-/Bachelor-/Diplom-/Masterarbeit: Verbesserung von Mutationstests mit Hilfe der symbolischen Ausführung. Bearbeiter: Michael Baer (beendet am 16.12.2015); Betreuer: Dipl.-Inf. Georg Dotzler; Marius Kamp, M. Sc.; Dr.-Ing. Norbert Oster; Prof. Dr. Michael Philippsen

– Master Thesis: Werkzeug zur computergestützten Diagnose von Laufzeitanomalien in Zeitreihen operationaler Daten. Bearbeiter: Stefan Ettl (beendet am 22.12.2015); Betreuer: Prof. Dr. Michael Philippsen; Andreas Kumlehn, M. Sc.