Jahresbericht 2024 des Lehrstuhls für Informatik 2 (Programmiersysteme)

1 Mitarbeiterinnen und Mitarbeiter

Julian Brandner, M. Sc., Tobias Heineken, M. Sc., Hon.-Prof. Dr.-Ing. Bernd Hindel, Hon.-Prof. Dr.-Ing. Detlef Kips, Florian Mayer, M. Sc. (bis 31.07.2024), Dr.-Ing. Norbert Oster, Akad. ORat, Prof. Dr. Michael Philippsen (Ordinarius), Prof. em. Dr. Hans Jürgen Schneider (Emeritus), David Schwarzbeck, M. Sc. (seit 01.10.2024), Alma Sinanović (IT-support, seit 01.06.2024), Margit Zenk (Sekretariat).

Gäste und externes Lehrpersonal am Lehrstuhl: Jonas Butz, M. Sc. (Lehrbeauftragter), Dr.-Ing. Klaudia Dussa-Zieger (Lehrbeauftragte), Dr.-Ing. Tobias Feigl (Lehrbeauftragter), Dr.-Ing. Martin Jung (Lehrbeauftragter), Patrick Kreutzer, M. Sc., Dipl.-Inf. Daniela Novac.

2 Überblick

Wir liefern ingenieurwissenschaftliche Antworten für Software-Ingenieure, die **parallele Software** im industriellen Rahmen für **Multicore-Rechner**, für daraus bestehende verteilte Systeme, sowie für vernetzte eingebettete Systeme entwickeln. Wir arbeiten **Programm-Code-basiert**, erstellen lauffähige **Prototypen** und **evaluieren** diese quantitativ und qualitativ. Eckpunkte unserer Forschungsthemen:
(a) Wir arbeiten an **Programmiermodellen** für **heterogene** Parallelität und erzeugen dafür portablen und effizienten Code für Multicores, GPUs, Acceleratoren, Mobile Geräte, FPGAs u.ä.
(b) Wir unterstützen die **Parallelisierung** von Software für Multicore-Rechner. Unsere Werkzeuge analysieren **Code-Repositories** und helfen dem Entwickler bei der **Migration** und **Refaktorisierung**.
(c) Wir analysieren Programme. Unsere **Code-Analysewerkzeuge** sind schnell, interaktiv, inkrementell und arbeiten teilweise selbst parallel. Sie finden Wettlaufsituationen, konkurrierende Ressourcenzugriffe etc. im Code und zeigen dem Entwickler Verbesserungsvorschläge punktgenau und in der Entwicklungsumgebung an.
(d) Wir **testen** parallelen Code und **diagnostizieren** Problemursachen. Unsere Werkzeuge erzeugen Testdaten, finden Ursachen von erratischem Laufzeitverhalten und schützen gegen **Authentizitätsangriffe**.

3 Forschung

3.1 AutoCompTest – *Automatisiertes Testen von Übersetzern*

Übersetzer für Programmiersprachen sind äußerst komplexe Anwendungen, an die hohe Korrektheitsanforderungen gestellt werden: Ist ein Übersetzer fehlerhaft (d.h. weicht sein Verhalten vom dem durch die Sprachspezifikation definierten Verhalten ab), so generiert dieser u.U. fehlerhaften Code oder stürzt bei der Übersetzung mit einer Fehlermeldung ab. Solche Fehler in Übersetzern sind oftmals schwer zu bemerken oder zu umgehen. Nutzer erwarten deshalb i.A. eine (möglichst) fehlerfreie Implementierung des verwendeten Übersetzers.

Leider lassen sowohl vergangene Forschungsarbeiten als auch Fehlerdatenbanken im Internet vermuten, dass kein real verwendeter Übersetzer fehlerfrei ist. Es wird deshalb an Ansätzen geforscht, mit deren Hilfe die Qualität von Übersetzern gesteigert werden kann. Da die formale Verifikation (also der Beweis

der Korrektheit) in der Praxis oftmals nicht möglich oder rentabel ist, zielen viele der Forschungsarbeiten darauf ab, Übersetzer möglichst umfangreich und automatisiert zu testen. In den meisten Fällen erhält der zu testende Übersetzer dabei ein Testprogramm als Eingabe. Anschließend wird das Verhalten des Übersetzers bzw. des von ihm generierten Programms überprüft: Weicht dieses vom erwarteten Verhalten ab (stürzt der Übersetzer also beispielsweise bei einem gültigen Eingabeprogramm mit einer Fehlermeldung ab), so wurde ein Fehler im Übersetzer gefunden. Soll dieser Testvorgang automatisiert stattfinden, ergeben sich drei wesentliche Herausforderungen:

- Woher kommen die Testprogramme, auf die der Übersetzer angewendet wird?
- Was ist das erwartete Verhalten des Übersetzers bzw. des von ihm erzeugten Codes? Wie kann bestimmt werden, ob das tatsächliche Verhalten des Übersetzers korrekt ist?
- Wie können Testprogramme, die einen Fehler im Übersetzer anzeigen, vorbereitet werden, um der Behebung des Fehlers im Übersetzer bestmöglich behilflich zu sein?

Während die wissenschaftliche Literatur diverse Lösungen für die zweite Herausforderung vorstellt, die auch in der Praxis bereits etabliert sind, stellt die automatisierte Generierung zufälliger Testprogramme noch immer eine große Hürde dar. Damit Testprogramme zur Detektion von Fehlern in allen Teilen des Übersetzers verwendet werden können, müssen diese allen Regeln der jeweiligen Programmiersprache genügen, d.h. die Programme müssen syntaktisch und semantisch korrekt (und damit übersetzbar) sein. Auf Grund der Vielzahl an Regeln „echter“ Programmiersprachen stellt die Generierung solcher übersetzbarer Programme eine schwierige Aufgabe dar. Dies wird zusätzlich dadurch erschwert, dass das Programmgenerierungsverfahren möglichst effizient arbeiten muss: Die wissenschaftliche Literatur zeigt, dass die Effizienz eines solchen Verfahrens maßgebliche Auswirkungen auf seine Effektivität hat – nur wenn in kurzer Zeit viele (und große) Programme generiert werden können, kann das Verfahren sinnvoll zur Detektion von Übersetzerfehlern eingesetzt werden.

In der Praxis scheitert das automatisierte Testen von Übersetzern deshalb oftmals daran, dass kein zugeschnittener Programmgenerator verfügbar ist und die Entwicklung eines solchen einen zu hohen Aufwand bedeutet. Ziel unseres Forschungsprojekts ist daher die Entwicklung von Verfahren, die den Aufwand für die Implementierung von effizienten Programmgeneratoren reduzieren.

Weil in der automatische Generierung zufälliger Testprogramme zumeist große Programme generiert werden müssen, um eine effiziente Fehlersuche zu ermöglichen, können diese Programme nur schwer zur Behebung des Fehlers verwendet werden. Üblicherweise ist nur ein sehr kleiner Teil des Programms ursächlich für den Fehler, und möglichst viele anderen Teile müssen automatisch entfernt werden, bevor die Behebung des Fehlers plausibel ist. Diese sogenannte Testfallreduktion nutzt auch die bereits angesprochenen Lösungen für die Erkennung des erwarteten Verhalten und ist für automatisch generierte Programme unerlässlich, sodass eine gemeinsame Betrachtung mit den anderen Komponenten sinnvoll ist. Üblicherweise ist die Testfallreduktion so gestaltet, dass fehlerauslösende Programm aus allen Quellen verarbeitet werden können.

Leider ist oftmals nicht klar, welches der verschiedenen in der wissenschaftlichen Literatur vorgestellten Verfahren sich am besten für welche Situation eignet. Außerdem dauert eine Testfallreduktion in einigen Fällen sehr lange. Ziel unseres Forschungsprojekts ist daher, eine aussagekräftige Testfallsammlung zu schaffen und an dieser bestehende Verfahren zu vergleichen und zu verbessern.

Im Jahr 2018 haben wir mit der Entwicklung eines entsprechenden Werkzeugs begonnen. Als Eingabe dient eine Spezifikation der syntaktischen und semantischen Regeln der jeweiligen Programmiersprache in Form einer abstrakten Attributgrammatik. Eine solche erlaubt eine knappe Notation der Regeln auf hohem Abstraktionsniveau. Ein von uns neu entwickelter Algorithmus erzeugt dann Testprogramme, die allen spezifizierten Regeln genügen. Der Algorithmus nutzt dabei diverse technische Ideen aus, um eine

angemessene Laufzeit zu erreichen. Dies ermöglicht die Generierung großer Testfallmengen in vertretbarer Zeit, auch auf üblichen Arbeitsplatzrechnern. Eine erste Evaluation hat nicht nur gezeigt, dass unser Verfahren sowohl effektiv als auch effizient ist, sondern auch dass es flexibel einsetzbar ist. So haben wir mit Hilfe unseres Verfahrens nicht nur Fehler in den C-Übersetzern gcc und clang entdeckt (unser Verfahren erreicht dabei eine ähnliche Fehleraufdeckungsgüte wie ein sprachspezifischer Programmgenerator aus der wissenschaftlichen Literatur), sondern auch diverse Bugs in mehreren SMT-Entscheidern. Einige der von uns entdeckten Fehler waren den jeweiligen Entwicklern zuvor noch unbekannt.

Im Jahr 2019 haben wir zusätzliche Features für das Schreiben von Sprachspezifikationen implementiert und die Effizienz des Programmgenerierungsverfahrens gesteigert. Durch die beiden Beiträge konnte der Durchsatz unseres Werkzeugs deutlich gesteigert werden. Des Weiteren haben wir mit Hilfe neuer Sprachspezifikationen Fehler in Übersetzern für die Programmiersprachen Lua und SQL aufgedeckt. Die Ergebnisse unserer Arbeit sind in eine Ende 2019 eingereichte (und inzwischen angenommene) wissenschaftliche Publikation eingeflossen. Neben der Arbeit an unserem Verfahren zur Programmgenerierung haben wir außerdem mit der Arbeit an einem Verfahren zur Testfallreduzierung begonnen. Das Verfahren reduziert die Größe eines zufällig generierten Testprogramms, das einen Fehler in einem Übersetzer auslöst, um die Suche nach der Ursache des Fehlers zu vereinfachen.

Im Jahr 2020 lag der Fokus des Forschungsprojekts auf sprachunabhängigen Verfahren zur automatischen Testfallreduzierung. Die wissenschaftliche Literatur schlägt unterschiedliche Reduzierungsverfahren vor. Da es bislang allerdings keinen aussagekräftigen Vergleich dieser Verfahren gibt, ist unklar, wie effizient und effektiv die vorgeschlagenen Reduzierungsverfahren tatsächlich sind. Wie wir festgestellt haben, gibt es dafür im Wesentlichen zwei Gründe, die darüber hinaus auch die Entwicklung und Evaluation neuer Verfahren erschweren. Zum einen verwenden die vorhandenen Implementierungen der vorgeschlagenen Reduzierungsverfahren unterschiedliche Implementierungssprachen, Programmrepräsentationen und Eingabegrammatiken. Mit diesen Implementierungen ist ein fairer Vergleich dieser Verfahren deshalb kaum möglich. Zum anderen gibt es keine umfangreiche Sammlung von (noch unreduzierten) Testprogrammen zur Evaluation von Reduzierungsverfahren. Dies hat zur Folge, dass die publizierten Reduzierungsverfahren jeweils nur mit sehr wenigen Testprogrammen evaluiert wurden, was die Aussagekraft der präsentierten Ergebnisse beeinträchtigt. Da in manchen Fällen darüber hinaus nur Testprogramme in einer einzigen Programmiersprache verwendet wurden, ist bislang unklar, wie gut diese Verfahren für Testprogramme in anderen Programmiersprachen funktionieren (wie sprachunabhängig diese Verfahren also tatsächlich sind). Um diese Lücken zu schließen, haben wir im Jahr 2020 mit der Implementierung eines Frameworks begonnen, das die wichtigsten Reduzierungsverfahren beinhaltet und so einen fairen Vergleich dieser Verfahren ermöglicht. Außerdem haben wir mit der Erstellung einer Testfallsammlung begonnen. Diese beinhaltet bereits etwa 300 Testprogramme in den Sprachen C und SMT-LIB 2, die etwa 100 unterschiedliche Fehler in realen Übersetzern auslösen. Diese Testfallsammlung erlaubt nicht nur aussagekräftigere Vergleiche von Reduzierungsverfahren, sondern verringert außerdem den Aufwand für die Evaluation zukünftiger Verfahren. Anhand erster Experimente konnten wir feststellen, dass es bislang kein Reduzierungsverfahren gibt, dass in allen Fällen am besten geeignet ist.

Außerdem haben wir uns im Jahr 2020 mit der Frage beschäftigt, wie das im Rahmen des Forschungsprojekts entstandene Framework zur Generierung zufälliger Testprogramme erweitert werden kann, um neben funktionalen Fehlern auch Performance-Probleme in Übersetzern finden zu können. Im Rahmen einer Abschlussarbeit ist dabei ein Verfahren entstanden, das eine Menge zufällig generierter Programme mit Hilfe von Optimierungstechniken schrittweise so verändert, dass die resultierenden Programme im getesteten Übersetzer deutlich höhere Laufzeiten als in einer Referenzimplementierung auslösen. Erste Experimente haben gezeigt, dass so tatsächlich Performance-Probleme in Übersetzern gefunden werden können.

Im Jahr 2021 haben wir die Implementierung der wichtigsten Reduzierungsverfahren aus der wissenschaftlichen Literatur sowie die Erstellung einer Testfallsammlung für deren Evaluation abgeschlossen.

Aufbauend darauf haben wir außerdem einen quantitativen Vergleich der Verfahren durchgeführt; soweit wir wissen, handelt es sich dabei um den mit Abstand umfangreichsten und aussagekräftigsten Vergleich bisheriger Reduzierungsverfahren. Unsere Ergebnisse zeigen, dass es bislang kein Verfahren gibt, das in allen Anwendungsfällen am besten geeignet wäre. Außerdem konnten wir feststellen, dass es für alle Verfahren zu deutlichen Ausreißern kommen kann, und zwar sowohl hinsichtlich Effizienz (also wie schnell ein Reduzierungsverfahren ein Eingabeprogramm reduzieren kann) als auch Effektivität (also wie klein das Ergebnis eines Reduzierungsverfahrens ist). Dies deutet darauf hin, dass es noch Potenzial für die Entwicklung weiterer Reduzierungsverfahren in der Zukunft gibt, und unsere Ergebnisse liefern einige Einsichten, was dabei zu beachten ist. So hat sich beispielsweise gezeigt, dass ein Hochziehen von Knoten im Syntaxbaum unabdingbar für die Generierung möglichst kleiner Ergebnisse (und damit eine hohe Effektivität) ist und dass eine effiziente Behandlung von Listenstrukturen im Syntaxbaum notwendig ist. Die Ergebnisse unserer Arbeit sind in eine im Jahr 2021 eingereichte und angenommene Publikation eingeflossen.

Außerdem haben wir im Jahr 2021 untersucht, ob bzw. wie sich die Effektivität unseres Programmgenerierungsverfahrens steigern lässt, wenn bei der Generierung die Überdeckung der zugrundeliegenden Grammatik berücksichtigt wird. Im Rahmen einer Abschlussarbeit wurden dazu unterschiedliche, aus der wissenschaftlichen Literatur stammende kontextfreie Überdeckungsmetriken für den Anwendungsfall adaptiert sowie implementiert und evaluiert. Dabei hat sich gezeigt, dass die Überdeckung hinsichtlich einer kontextfreien Metrik nur bedingt mit der Fehleraufdeckung korreliert. In zukünftigen Arbeiten sollte deshalb untersucht werden, ob Überdeckungsmetriken, die auch kontextsensitive, semantische Eigenschaften berücksichtigen, besser für diesen Anwendungsfall geeignet sind.

Im Jahr 2022 wurde im Rahmen einer Abschlussarbeit mit der Entwicklung eines Rahmenwerks für die Realisierung sprachadaptierter Reduktionsverfahren begonnen. Dieses Rahmenwerk stellt eine domänenspezifische Sprache (DSL) zur Verfügung, mit deren Hilfe sich Reduktionsverfahren auf einfache und knappe Art und Weise beschreiben lassen. Mit diesem Rahmenwerk und der entwickelten DSL soll es möglich sein, bestehende Reduktionsverfahren mit möglichst wenig Aufwand an die Besonderheiten einer bestimmten Programmiersprache anpassen zu können. Die Hoffnung dabei ist, dass solche sprachadaptierten Verfahren noch effizienter und effektiver arbeiten können als die bestehenden, sprachunabhängigen Reduktionsverfahren. Darüber hinaus soll das Rahmenwerk auch den Aufwand für die Entwicklung zukünftiger Reduktionsverfahren verringern und könnte so einen wertvollen Beitrag für die Forschung auf diesem Gebiet leisten.

In Jahr 2023 lag der Fokus des Forschungsprojekts auf den Listenstrukturen, die bereits in 2021 kurz angesprochen wurden: Nahezu alle seit 2021 untersuchten Verfahren gruppieren Knoten im Syntaxbaum in Listen, um aus diesen dann mittels eines Listenreduktionsverfahrens nur die notwendigen Knoten auszuwählen. Unsere Experimente haben gezeigt, dass teilweise 70% und mehr der Reduktionszeit in Listen mit mehr als 2 Elementen verbracht wird. Diese Listen sind deswegen relevant, weil es in der wissenschaftlichen Literatur verschiedene Listenreduktionsverfahren gibt, diese sich aber für Listen mit 2 oder weniger Elementen nicht unterscheiden. Da der zeitliche Anteil so hoch sein kann, haben wir diese verschiedenen Listenreduktionsverfahren in unsere in 2020/2021 entwickelten Implementierungen der wichtigen Reduktionsverfahren integriert. Dabei haben wir neben den Verfahren aus der Literatur auch solche aufgenommen, die nur auf einer Website beschrieben oder nur in einer frei zugänglichen Implementierung umgesetzt waren.

Es wurde auch untersucht, wie man eine Listenreduktionen an einer Stelle unterbrechen und später fortsetzen kann. Die Idee war, auf der Basis einer Priorisierung zwischenzeitlich eine andere Liste zu reduzieren, die eine größere Auswirkung auf die Verkleinerung hat. In einigen Fällen trat der erhoffte Geschwindigkeitsgewinn zwar ein, es bleiben aber Fragen offen, die weitere Experimente mit priorisierenden Reduzierern und Listenreduktionsverfahren erfordern.

Im Jahr 2024 konnten wir erfolgreich erste Ergebnisse aus der Untersuchung der Listenreduktionsverfahren publizieren: Durch das Austauschen der Listenverfahren können etablierte Reduzierungsverfahren um bis zu 74,7% beschleunigt werden. Erwartungsgemäß profitieren Verfahren, die lange Listen verursachen, am stärksten von der Änderung. Außerdem wurde untersucht, welchen Einfluss die Reihenfolge der Listenelemente hat. Hier können auch bis zu 44,1% der Laufzeit eingespart werden, aber zwei Aspekte verringern die Effektivität einer Umsortierung:

1. Die textuelle Reihenfolge, in der die Listenelemente üblicherweise aufgereiht werden, ist bereits eine vergleichsweise gute Reihenfolge.
2. Die gleichen Aspekte, die ein Listenverfahren schnell machen, sorgen für einen geringeren Einfluss der Reihenfolge.

Im Rahmen von zwei Abschlussarbeiten wurden zwei weitere Aspekte untersucht:

1. Das von 2018 - 2021 entwickelte Werkzeug zur Erzeugung von Testprogrammen nutzt den zu testenden Übersetzer nur als sogenannte „black box“, d.h. es erzeugt Programme, ohne auf Informationen aus dem Übersetzer zurückzugreifen. In einer Abschlussarbeit wurde versucht, mittels Überdeckungsinformationen aus dem getesteten Übersetzer die erzeugten Programme zu verbessern.
2. Bei den Reduzierungsverfahren ist ein Cache wichtig, um für sich wiederholende Reduktionskandidaten nicht immer den zu testenden Übersetzer auszuführen. Naive Implementierungen dieser Caches werden jedoch sehr groß. Im Jahr 2023 wurde bereits ein dediziertes Cache-Verfahren vorgestellt, das die Cache-Größe um ca. 90% verkleinern kann. Leider ist dieses Cache-Verfahren nicht für alle in unserem Framework enthaltenen Reduzierungsverfahren geeignet und wurde deswegen in einer Abschlussarbeit entsprechend erweitert.

3.2 ORKA-HPC – *OpenMP für rekonfigurierbare heterogene Architekturen*

High-Performance Computing (HPC) ist ein wichtiger Bestandteil für die europäische Innovationskapazität und wird auch als ein Baustein bei der Digitalisierung der europäischen Industrie gesehen. Rekonfigurierbare Technologien wie Field Programmable Gate Array (FPGA) Module gewinnen hier wegen ihrer Energieeffizienz, Performance und ihrer Flexibilität immer größere Bedeutung.
Es wird außerdem zunehmend auf HPC-Systeme mit heterogenen Architekturen gesetzt, auch auf solche mit FPGA-Beschleunigern. Die große Flexibilität dieser FPGAs ermöglicht es, dass eine große Klasse von HPC-Applikationen mit FPGAs realisiert werden kann. Allerdings ist deren Programmierung bisher vorwiegend Spezialisten vorbehalten und sehr zeitaufwendig, wodurch deren Verwendung in Bereichen des wissenschaftlichen Höchstleistungsrechnens derzeit noch selten ist.
Im HPC-Umfeld gibt es verschiedenste Programmiermodelle für heterogene Rechnersysteme mit einigen Typen von Beschleunigern. Gängige Programmiermodelle sind zum Beispiel OpenCL (opencl.org), OpenACC (openacc.org) und OpenMP (OpenMP.org). Eine produktive Verwendbarkeit dieser Standards für FPGAs ist heute jedoch noch nicht gegeben.

Ziele des ORKA Projektes sind:

1. Nutzung des OpenMP-4.0-Standards als Programmiermodell, um ohne Spezialkenntnisse heterogene Rechnerplattformen mit FPGAs als rekonfigurierbare Architekturen durch portable Implementierungen eine breitere Community im HPC-Umfeld zu erschließen.
2. Entwurf und Implementierung eines Source-to-Source-Frameworks, welches C/C++-Code mit OpenMP-4.0-Direktiven in ein ausführbares Programm transformiert, das die Host-CPUs und FPGAs nutzt.

3. Nutzung und Erweiterung existierender Lösungen von Teilproblemen für die optimale Abbildung von Algorithmen auf heterogene Systeme und FPGA-Hardware.
4. Erforschung neuer (ggf. heuristischer) Methoden zur Optimierung von Programmen für inhärent parallele Architekturen.

Im Jahr 2018 wurden folgende wesentlichen Beiträge geleistet:

- Entwicklung eines source-to-source Übersetzerprototypen für die Umschreibung von OpenMP-C-Quellcode (vgl. Ziel 2).
- Entwicklung eines HLS-Übersetzerprototypen, der in der Lage ist, C-Code in Hardware zu übersetzen. Dieser Prototyp bildet die Basis für die Ziele 3 und 4.
- Entwicklung mehrerer experimenteller FPGA-Infrastrukturen für die Ausführung von Beschleunigerkernen (nötig für die Ziele 1 und 2).

Im Jahr 2019 wurden folgende wesentlichen Beiträge geleistet:

- Veröffentlichung zweier Papiere: „OpenMP on FPGAs - A Survey“ und „OpenMP to FPGA Offloading Prototype using OpenCL SDK“.
- Erweiterung des source-to-source Übersetzerprototypen um OpenMP-Target-Outlining (incl. Smoke-Tests).
- Fertigstellung des technischen Durchstichs für den ORKA-HPC-Prototypen (OpenMP-zu-FPGA-Übersetzer).
- Benchmark-Suite für die quantitative Leistungsanalyse von ORKA-HPC.
- Erweiterung des source-to-source Übersetzerprototypen um das Genom für die genetische Optimierung der High-Level-Synthese durch Einstellen von HLS-Pragmas.
- Prototypische Erweiterung des TaPaSCo-Composers um ein (optionales) automatisches Einfügen von Hardware-Synchronisationsprimitiven in TaPaSCo-Systeme.

Im Jahr 2020 wurden folgende wesentlichen Beiträge geleistet:

- Weiterentwicklung der Genetischen Optimierung.
- Aufbau eines Docker-Containers für zuverlässige Reproduzierbarkeit der Ergebnisse.
- Integration der Softwarekomponenten der Projektpartner.
- Plugin-Architektur für Low-Level-Platformen.
- Implementation und Integration zweier LLP-Plugin-Komponenten.
- Erweiterung des akzeptierten OpenMP-Sprachstandards.
- Erweiterung der Test-Suite.

Im Jahr 2021 wurden folgende wesentlichen Beiträge geleistet:

- Erweiterung der Benchmark-Suite.
- Erweiterung der Test-Infrastruktur.
- Erfolgreicher Projektabschluss mit Live-Demo für den Projektträger.
- Evaluation des Projekts.
- Veröffentlichung der Publikation „ORKA-HPC - Practical OpenMP for FPGAs“.
- Veröffentlichung des Quell-Codes und der Disseminationsumgebung auf Github.
- Erweiterung des akzeptierten OpenMP-Sprachstandards um neue OpenMP-Klauseln für die Steuerung der FPGA-bezogenen Transformationen.

- Weiterentwicklung der Genetischen Optimierung.
- Untersuchung des Verhältnisses von HLS-Leistungsschätzwerten und tatsächlichen Leistungskennzahlen.
- Aufbau eines linearen Regressionsmodells für die Vorhersage der tatsächlichen Leistungskennzahlen auf Basis der HLS-Schätzwerte.
- Entwicklung von Infrastruktur für die Übersetzung von OpenMP-Reduktionsklauseln.
- Erweiterung um die Übersetzung vom OpenMP-Pragma „parallel for“ in ein paralleles FPGA-System.

Im Jahr 2022 wurden folgende wesentlichen Beiträge geleistet:

- Generierung und Veröffentlichung eines Datensatzes zur Untersuchung des Verhältnisses von HLS-Ressourcenschätzwerten und tatsächlichen Leistungskennzahlen.
- Erstellung und vergleichende Evaluierung verschiedener Regressionsmodelle zur Vorhersage der tatsächlichen Systemperformanz aus frühen Schätzwerten.
- Analyse und Bewertung der durch die HLS generierten Ressourcenabschätzungen.
- Veröffentlichung der Publikation „Reducing OpenMP to FPGA Round-trip Times with Predictive Modelling“.
- Entwicklung eines auf dem Polyeder-Modell beruhenden Verfahrens zur Detektion und Entfernung von redundanten Lese-Operationen in FPGA-Stencil-Codes.
- Implementierung dieses Verfahrens in ORKA-HPC.
- Quantitative Evaluation der Stärken dieses Verfahrens und Ermittlung der Voraussetzungen, unter denen das Verfahren anwendbar ist.
- Veröffentlichung der Publikation „Employing Polyhedral Methods to Reduce Data Movement in FPGA Stencil Codes“.

Im Jahr 2023 wurden folgende wesentlichen Beiträge geleistet:

- Entwicklung und Implementierung eines Optimierungsverfahrens von kanonischen Schleifenschachteln (z.B. aus OpenMP-Target-Regionen) für die FPGA-Hardware-Erzeugung mittels HLS. Der Kern des Verfahrens ist eine auf dem Polyeder-Modell basierende Schleifenrestrukturierung, welche mithilfe von Schleifen-Kachelungen, Fließbandverarbeitung, und Port-Verbreiterung unnötige Datentransfers vom/zum FPGA-Board-RAM vermeidet, die Anzahl der parallel aktiven Schaltkreise erhöht, den Datendurchsatz zum FPGA-Board-RAM maximiert und Schreib/Lese-Latenzen versteckt.
- Quantitative Evaluation der Stärken und Anwendungsfelder dieses Optimierungsverfahrens mithilfe von ORKA-HPC.
- Veröffentlichung des Verfahrens im Konferenz-Papier „Employing polyhedral methods to optimize stencils on FPGAs with stencil-specific caches, data reuse, and wide data bursts“.
- Veröffentlichung eines Reproduktionspakets für das Optimierungsverfahrens.
- Vorstellung des Verfahrens auf der Tagung „14th International Workshop on Polyhedral Compilation Techniques“ im Rahmen eines halbstündigen Vortrags.
- Entwicklung eines Verfahrens für die vollautomatische Integration von Multi-Purpose-Caches in aus OpenMP generierte FPGA-Lösungen.
- Evaluation von Multi-Purpose-Caches in Kombination mit HLS-generierten Hardwareblöcken.
- Veröffentlichung der Publikation „Multipurpose Cacheing to Accelerate OpenMP Target Regions on FPGAs“ (Best Paper Award).

Im Jahr 2024 wurden folgende wesentlichen Beiträge geleistet:

- Anpassung mehrerer bereits publizierter Cacheing-Ansätze auf ausgelagerte OpenMP-Codes und Einbau der Verfahren in ORKA-HPC.
- Entwicklung und Evaluation neuartiger mehrstufiger Caches für HLS-Kernel.
- Veröffentlichung der Ergebnisse in der Publikation „Multilayer Multipurpose Caches for OpenMP Target Regions on FPGAs“ und Vorstellung der Arbeit bei der IWOMP 2024 in Perth.

3.3 SoftWater – *Software-Wasserzeichen*

Unter Software-Wasserzeichnen 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 Software-Piraterie, 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 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 sowohl in neu entwickelte als auch in bestehende Programme hinzufügen. Als ersten Ansatz untersuchten wir eine Wasserzeichenmethode, die auf einer symbolischen Ausführung und anschließender Funktionsynthese basiert.
Im Jahr 2018 wurden im Rahmen von zwei Bachelorarbeiten Methoden zur symbolischen Ausführung und Funktionssynthese untersucht, um zu ermitteln, welche sich für unseren Ansatz am Besten eignet.
Im Jahr 2019 wurde ein Ansatz auf Basis der LLVM Compiler Infrastruktur untersucht, der mittels konkolischer Ausführung (concolic execution, eine Kombination aus symbolischer und konkreter Ausführung) ein Wasserzeichen in einem ungenutzten Hardwareregister versteckt. Hierzu wurde der LLVM-Registerallokator dahingehend verändert, dass er ein Register für das Wasserzeichen freihält.
Im Jahr 2020 wurde das inzwischen LLWM genannte Rahmenprogramm für das automatische Einfügen von Software-Wasserzeichen in Quellcode auf Basis der LLVM Compiler Infrastruktur um weitere dynamische Verfahren erweitert. Grundlage der hinzugefügten Verfahren sind, unter anderem, das Ersetzen/Verschleiern von Sprungadressen sowie Modifikationen des Aufrufgraphen.
Im Jahr 2021 wurde das Rahmenprogramm LLWM um weitere angepasste, bereits in der Literatur bekannte, dynamische Verfahren sowie um das eigene Verfahren erweitert, das wir nun IR-Mark nennen Die hinzugefügten Verfahren basieren unter anderem auf der Umwandlung von bedingten Konstrukten in semantisch äquivalenten Schleifen oder auf Integrieren von Hashfunktionen, die die Funktionalität des Programms unverändert lassen, die Widerstandsfähigkeit aber erhöhen. IR-Mark wählt nun nicht nur gezielt die wenigen Funktionen aus, in denen die Registerverwendung bei der Code. Erzeugung verändert wird, sondern umfasst nun auch dynamische Aspekte um in den freigehaltenen Registern sinnvoll erscheinende Tarnwerte zu berechnen. Ein Artikel über LLWM und IR-Mark konnte publiziert werden.
Im Jahr 2022 wurde das Rahmenprogramm LLWM um ein weiteres angepasstes Verfahren ergänzt. Die Methode nutzt Ausnahmebehandlungen, um das Wasserzeichen zu tarnen.
Im Jahr 2023 wurden mehr Methoden angepasst, um das LLWM-Framework zu erweitern. Hierzu zählen Techniken zum Einbetten, die auf Prinzipien der Zahlentheorie und des Aliasings beruhen.
Im Jahr 2024 wurden drei neue Wasserzeichen entwickelt: *Register Expansion*, *SemaCall* und *SideData*. Diese Techniken konstruieren streufunktionsartige Arithmetiken, um während der Laufzeit einen Schlüsselwert in die Wasserzeichennachricht umzurechnen.
Die ersten beiden Techniken wurden in dem Papier „Register Expansion and SemaCall: 2 Low-overhead Dynamic Watermarks Suitable for Automation in LLVM“ auf dem CheckMATE'24 Workshop in Salt Lake City publiziert. Eine erweiterte Fassung des Papiers enthält auch das SideData Wasserzeichen und befindet sich im Peer-Review-Prozess für das DTRAP-Journal.

4 Lehre

Der Lehrstuhl für Programmiersysteme bietet im Wintersemester die Pflichtmodule *Algorithmen und Datenstrukturen (AuD)* sowie *Parallele und Funktionale Programmierung (PFP)* an. Aufgrund einer Änderung der Prüfungsordnung fand die Vorlesung zu AuD letztmals im Wintersemester 2021/22 und der Übungsbetrieb letztmals im Wintersemester 2023/24 statt. Da beide Module fakultätsübergreifend für diverse Studiengänge (Informatik, Informations- und Kommunikationstechnik, Mathematik u.v.a.) angeboten werden, erreichten die Hörerzahlen mit 264 bzw. 259 (PFP im WS2023/24 bzw. WS2024/25) im Berichtszeitraum erneut sehr hohe Werte, die sich schließlich auch in der hohen Zahl an Prüfungsanmeldungen (49 in AuD sowie 548 in PFP - jeweils im WS2023/24, SS2024 und WS2024/25) niedergeschlagen haben. In der Vertiefungsrichtung Programmiersysteme bietet der Lehrstuhl verschiedene Module zu den Themen *Übersetzerbau* und *Testen von Softwaresystemen* an. Die Seminare *Hallo Welt! für Fortgeschrittene* und *Machine Learning* waren erneut innerhalb kürzester Zeit restlos ausgebucht.
Insgesamt betreute der Lehrstuhl für Programmiersysteme im Berichtsjahr acht Masterarbeiten und sechs Bachelorarbeiten.

ICPC – *International Collegiate Programming Contest an der FAU*: Seit 1977 wird der International Collegiate Programming Contest (ICPC) ausgetragen. Dabei sollen Teams aus je drei Studierenden ca. 13 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. Bei der Lösung kommt es darauf an, einen effizienten und richtigen Algorithmus zu finden und zu implementieren.
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 stattfinden.
Am 27. Januar 2024 fand erneut der Winter Contest statt. Daran beteiligten sich 108 Teams von 17 Hochschulen und Universitäten, darunter 15 Teams aus Erlangen. Unser bestes Team erreichte den 32. Platz. Am 22. Juni wurde der German Collegiate Programming Contest an verschiedenen deutschen Universitäten und Hochschulen ausgetragen, mit 15 Teams aus Erlangen. Das beste FAU-Team belegte den 22. Platz unter den 94 teilnehmenden Teams aus ganz Deutschland. Der NWERC fand am 24. November in Delft statt. Dort wurde die FAU durch ein Team vertreten, das Platz 19 von insgesamt 80 teilnehmenden Teams erreichte. Das Hauptseminar „Hallo Welt! - Programmieren für Fortgeschrittene“ fand auch im Jahr 2024 wieder statt.

5 Publikationen

[1] Julian Brandner, Florian Mayer, and Michael Philippsen. Multilayer Multipurpose Caches for OpenMP Target Regions on FPGAs. In A. Espinosa, M. Klemm, B.R. de Supinski, M. Cytowski, and J. Klinkenberg, editors, *OpenMP: Advancing OpenMP for Future Accelerators*, volume 15195 of *Springer's Lecture Notes in Computer Science (LNCS)*, page 79–93, Cham, 2024. Springer. doi:10.1007/978-3-031-72567-8_6.

[2] Julian Brandner, Florian Mayer, and Michael Philippsen. Multilayer Multipurpose Caches for OpenMP Target Regions on FPGAs [Data set], 2024. doi:10.5281/zenodo.12755510.

[3] Leon Brasseler, Maximilian Stahlke, Thomas Robert Altstidl, Tobias Feigl, and Christopher Mutschler. Non-Line-of-Sight Detection for Radio Localization using Deep State Space Models. In *The fourteenth International Conference on Indoor Positioning and Indoor Navigation 2024 (IPIN 2024)*, pages 1–6, 2024.

[4] David Franco Contreras, Inigo Cortes, Georgios Kontes, Tobias Feigl, Christopher Mutschler, and Alexander Ruegamer. Reinforcement Learning Framework for Robust Navigation in GNSS Receivers. In *Proceedings of the 37th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2024)*, pages 2392–2408, 2024. doi:10.33012/2024.19853.

[5] Tobias Heineken and Michael Philippsen. Replication Package for 'The Impact of List Reduction for Language Agnostic Test Case Reducers', 2024. doi:10.5281/zenodo.13835515.

[6] Lucas Heublein, Felix Ott, and Tobias Feigl. Research Avenues for GNSS Interference Classification Robustness: Domain Adaptation, Continual Learning & Federated Learning. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 1–4, 2024.

[7] Lucas Heublein, Nisha Lakshmana Raichur, Tobias Feigl, Tobias Brieger, Fin Heuer, Lennart Asbach, Jonathan Hansen, Alexander Ruegamer, and Felix Ott. Evaluation of (Un-)Supervised Machine Learning Methods for GNSS Interference Classification with Real-World Data Discrepancies. In *Proc. Intl. Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+) 2024*, volume 1, pages 1–18, 2024.

[8] Lucas Heublein, Nisha Lakshmana Raichur, Tobias Feigl, Tobias Brieger, Fin Heuer, Lennart Asbach, Alexander Ruegamer, and Felix Ott. GNSS Interference Monitoring: Resilience of Machine Learning Methods on Public Real-World Datasets. *Navigation, Journal of the Institute of Navigation*, 7:1–20, 2024.

[9] Marius Kamp. *Detecting Unrealizable Bit Vector Program Synthesis Problems*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2024. URL: https://open.fau.de/handle/openfau/33619, doi:10.25593/978-3-96147-794-4.

[10] Marius Kamp. Replication Package for "Detecting Unrealizable Bit Vector Program Synthesis Problems", Jan 2024. doi:10.5281/zenodo.10593916.

[11] Florian Mayer, Julian Brandner, and Michael Philippsen. Employing Polyhedral Methods to Optimize Stencils on FPGAs with Stencil-specific Caches, Data Reuse, and Wide Data Bursts. In *14th International Workshop on Polyhedral Compilation Techniques, (IMPACT 2024, in conjunction with HiPEAC 2024)*, page 12p, Jan 2024. URL: https://impact-workshop.org/impact2024/#mayer24-fpgas, doi:10.48550/arXiv.2401.13645.

[12] Florian Mayer, Julian Brandner, and Michael Philippsen. Employing polyhedral methods to optimize stencils on FPGAs with stencil-specific caches, data reuse, and wide data bursts [Reproduction Package], Jan 2024. doi:10.5281/zenodo.10396084.

[13] Felix Ott, Lucas Heublein, Nisha Lakshmana Raichur, Tobias Feigl, Jonathan Hansen, Alexander Ruegamer, and Christopher Mutschler. Few-Shot Learning with Uncertainty-based Quadruplet Selection for Interference Classification in GNSS Data. In *International Conference on Localization and GNSS*, page 7, 2024. doi:10.1109/ICL-GNSS60721.2024.10578525.

[14] Jonathan Ott, Jonas Pirkl, Maximilian Stahlke, Tobias Feigl, and Christopher Mutschler. Radio Foundation Models: Pre-training Transformers for 5G-based Indoor Localization. In *The fourteenth*

edition of the International Conference on Indoor Positioning and Indoor Navigation (IPIN), pages 1–6, 2024.

[15] David Schwarzbeck. Erweiterung eines Rahmenprogramms für das automatische Einfügen von Software- Wasserzeichen in Quellcode. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2024. URL: https://cs2-gitlab.cs.fau.de/softwater/ma-david/-/raw/master/thesis/thesis.pdf?ref_type=heads.

[16] David Schwarzbeck, Daniela Novac, and Michael Philippsen. Register Expansion and SemaCall: 2 Low-overhead Dynamic Watermarks Suitable for Automation in LLVM. In *CheckMATE '24: Proceedings of the 2024 Research on offensive and defensive techniques in the context of Man At The End (MATE) attacks*, pages 1–10, New York, 2024. ACM. URL: https://dl.acm.org/doi/10.1145/3689934.3690815#, doi:10.1145/3689934.3690815.

[17] David Schwarzbeck, Jan Schuh, Mercel Hammrich, Michael Philippsen, and Daniela Novac. Register Expansion and SemaCall: 2 low-overhead dynamic Watermarks suitable for Automation in LLVM [Source code and Raw Experiment data], 2024. doi:10.5281/zenodo.13337275.

[18] Maximilian Stahlke, Tobias Feigl, Sebastian Kram, Björn Eskofier, and Christopher Mutschler. Uncertainty-Based Fingerprinting Model Monitoring for Radio Localization. *IEEE Journal of Indoor and Seamless Positioning and Navigation*, 2024.

[19] Johannes Rossouw Van Der Merwe, David Franco Contreras, Tobias Feigl, and Alexander Ruegamer. Optimal machine learning and signal processing synergies for low-resource GNSS interference detection and classification. *IEEE Transactions on Aerospace and Electronic Systems*, pages 3–17, Jan 2024. doi:10.1109/TAES.2023.3349360.