The goal of my research is to develop techniques and tools for automated reasoning about numerical programs and thus help scientists and engineers write correct, accurate and efficient software. The first computers were designed to perform numerical calculations and until today, numerical programs remain widely used in mathematics, science and engineering. Despite their success, programming language support for writing computations with *known accuracy* remains scarce. One aspect that makes verification of numerical computations difficult is the inherent gap between the continuous nature of much of mathematics and physical processes and the discrete implementation on today's digital computers. However, reliably controlling accuracy is not enough; performance and energy efficiency have become key design constraints for most applications. These objectives are often conflicting so we need to find an acceptable trade-off.
Science and engineering is approximate. While the physical world and mathematical models are continuous in nature, measurement equipment has finite accuracy and computers have only finite resources. We thus inevitably have to use finite-precision arithmetic, stop algorithms after a finite number of iterations or steps and replace complex computations by simpler ones. All such approximations are a trade-off between accuracy and efficiency: more accurate results come at a higher resource cost in terms of memory and time. This trade-off however, is also an opportunity. Indeed, many applications are robust with respect to errors, so we can employ less accurate algorithms and still obtain acceptable results, but at a lower cost. As energy is increasingly becoming a rare resource, these considerations will become more and more critical. In order to exploit approximate computing fully, we need to know which approximations are adequate for a particular application. Current automated tool support is scarce, both in *finding* and in *evaluating* approximation candidates.

Dissertation Research

The research of my dissertation focused on methods and tools for automated, sound and accurate reasoning about the accumulation and propagation of numerical errors, including roundoff errors from floating-point or fixed-point arithmetic. We developed a comprehensive set of techniques for bounding worst-case errors in programs with nonlinear computations, loops as well as discontinuities, for writing and compiling numerical programs more naturally in a real-valued programming language and for synthesizing expressions with improved numerical accuracy.

A New Programming Model Today, when writing numerical code, the programmer has to choose a data type up front and then remember to verify the accuracy of the results, leaving accuracy as an afterthought. We developed a framework with a real-valued specification language [4] which explicitly includes numerical errors. Our verifying compiler [4, 2], Rosa, then selects a suitable data type which fulfills the error specification. This approach separates the real-valued mathematical specification of a problem from its low-level implementation details, letting the programmer write programs naturally in reals and potentially opening up new (real-valued) verification avenues. Additionally, with a real-valued specification language, the compiler may perform compiler optimizations which are valid in a real-valued semantics and can, for instance, take advantage of associativity which would not be semantically sound in finite precision.

Algorithms for Accurate Error Estimation A crucial step in making a verifying compiler viable is to have a way to accurately bound numerical errors, and in particular roundoff errors. A naive approach using for example standard interval arithmetic produces sound, but unusably large error bounds. In addition, discontinuities from conditional statements may cause the concrete execution to take a different path than a real-valued one would take. Loops, in general, increase errors with every iteration without a constant upper bound, making sound, accurate and automated error estimation extremely difficult. The error estimation that we developed crucially relies on a combination of different techniques. We used techniques such as interval and affine arithmetic for computing ranges of variables, and derivatives for characterizing the sensitivity of arithmetic expressions to initial errors. We applied them in novel ways to track roundoff errors and other uncertainties, and combined them with a nonlinear Satisfiability Modulo Theories (SMT) solver to significantly increase both the accuracy of the results and the automation of the analysis. Our verifying compiler called Rosa performs a static analysis of the numerical errors using these techniques and based on this information selects a suitable floating-point or fixed-point data type. We demonstrated the effectiveness of the analysis on a number of real-world examples from embedded systems, physics and biology.

Combining Numerical Analysis with Automated Reasoning Many iterative algorithms are self-stabilizing in that individual iterations are independent of each other and errors from one iteration get corrected in subsequent ones. It is thus not helpful to track round-off errors throughout an entire computation as described in the previous paragraph. However, the exact result can usually only be obtained after an infinite number of iterations. Stopping the iteration early introduces another type of error, commonly called truncation error, which essentially captures the magnitude of the truncated sequence.
We used numerical analysis theorems [7] and integrated them together with our sound reasoning techniques into a programming language. The result is a runtime library for certifying solutions of systems of equations up to a given tolerance [3]. We further use programming language techniques such as macros to obtain a sound yet efficient verification method. I believe that many more applications can benefit from a (more profound) combination of traditional mathematics and computer science.

Using Genetic Programming to Improve Accuracy Sometimes accuracy can be gained by simply re-ordering the non-associative finite-precision computation. We used genetic programming together with our numerical error estimation technique to search for a re-write of a fixed-point arithmetic expression that minimizes roundoff errors [5]. Our experiments with embedded controllers show that the accuracy gains can be substantial. What is more, these improvements come essentially for free, since our static approach does not require larger data types or a runtime overhead.

Future Directions

My dissertation work is a first step towards my goal of helping scientists write accurate and efficient numerical programs. I want to build upon this work and develop techniques and tools for *automated, rigorous* and *resource-aware* verification and synthesis approaches in the area of *approximate computing*. I plan to go beyond finite-precision roundoff and truncation errors and consider numerical approximations in general, and more specifically the trade-off between accuracy and efficiency. To this end, I want to continue with the common thread of using rigorous mathematical techniques and theorems, and combine them with automated reasoning and machine learning tools. I plan to apply such methods in different domains of numerical computations such as embedded systems, scientific computing and machine learning.
Approximate computing studies opportunities in software and hardware to perform numerical computations with lower accuracy in order to save energy, memory and time. This approach is particularly interesting for applications which are inherently error tolerant or noisy so that effects of reduced precision are either not noticeable or acceptable. Today, most approximations are written by hand, which is very difficult, error prone and comes with little guarantees about the final accuracy of the results. Sample current work considered automatic optimizations on a low level, but without worst-case guarantees [8] or required the user to provide candidate approximations as well as the corresponding accuracy specifications [1, 9]. I want to improve on this state of affairs by providing the tools for both the principled synthesis of approximation candidates and their evaluation in terms of accuracy and efficiency.

Improve Resource Efficiency by Approximation We can improve resource efficiency by reducing the accuracy of numerical computations, both in software and hardware. Focusing mainly on the software side, examples of possible approximations are replacing trigonometric functions by polynomials or by linearizing nonlinear arithmetic. I plan to address several challenges which emerge when we try to approximate complex computations by simpler ones in a principled and practical way.
Firstly, in a complete system, only some parts of the computation are amenable to approximation. I will explore methods that identify the most promising parts of a program both in terms of accuracy and resources. Secondly, the space of possible approximations is huge. I want to explore ways to combine classical approximations in mathematics with automated reasoning and machine learning techniques to synthesize suitable candidates. Thirdly, an automated accuracy verification of the generated approximations goes beyond current capabilities. I want to develop techniques to automatically check that an approximation is close enough to its corresponding ideal computation for all possible inputs, including all possible uncertainties and roundoff errors. I expect such a 'correctness up-to tolerance' concept to be useful in its own right as noise and uncertainty appear in many different domains, including probabilistic programming. The ideal goal is to have an end-to-end static technique that is efficient and accurate. However, this may be too much to hope for, in which case I want to investigate different combinations of static and dynamic techniques, an approach I have successfully taken in the past. A further possible avenue is to relax the soundness requirement in favor of a probabilistic analysis.

Finally, I would like to collaborate with researchers from the hardware community to explore hardware-software co-design.

Handling and Leveraging Non-Associativity It is well known that re-ordering non-associative finite-precision arithmetic produces different roundoff errors. On one hand, this fact can be used to improve accuracy, but current approaches for automated rewriting are rare and limited [6, 5]. On the other hand, there are applications, in particular parallel ones, where a re-ordering of numerical computations is inherent and often an important concern.
I plan to build on my current work [5] of leveraging associativity to improve accuracy and extend it in terms of scalability as well as applicability to be able to handle real-world programs beyond embedded controllers. This requires in particular the development of new error estimation methods capable of handling larger program fragments and additional mathematical functions sufficiently accurately, but which need to be light-weight at the same time to be effective on a very large search space. Furthermore, as different rewrites tend to be appropriate for different inputs, I will develop sound specification generation techniques with respect to numerical accuracy. These will serve as preconditions for individual rewrites and I believe that such automatically generated specifications will be beneficial in other areas as well.
I believe that the development of such techniques will aid in understanding the overall effect and potential of computation re-ordering and can be used to provide accuracy guarantees even for parallel programs.

Usability and Applicability Automated techniques can be a double-edged sword, especially when blindly applied in potentially safety-critical applications. While a black-box nature is an excellent feature for productivity, the user needs to understand where a synthesized program fragment comes from and what its effect is on the program execution. Only with this information is the user able to judge whether a technique and its result is appropriate for a particular program. I plan to explore techniques to provide the user not only with a final result, but also with a traceable explanation or certificate.
Another important aspect in making verification and synthesis techniques accepted and used by the community is the integration within a programming language and with the work flow. While my main focus is on the underlying methods, I want to continue to implement all techniques in practical and open source tools and dedicate effort to the integration as well.
Finally, it is my goal to build up collaborations with other scientific communities that use numerical programs such as scientific computing, embedded systems and machine learning. Ultimately, I would like to apply my methods and tools to their real-world problems and thus help scientists and engineers write accurate and efficient software.

References

[1] Woongki Baek and Trishul M. Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, 2010.

[2] Eva Darulova. Rosa, the real verifier. `https://github.com/malyzajko/rosa`.

[3] Eva Darulova and Viktor Kuncak. Certifying Solutions for Numerical Constraints. In *RV*, 2012.

[4] Eva Darulova and Viktor Kuncak. Sound Compilation of Reals. In *POPL*, 2014.

[5] Eva Darulova, Viktor Kuncak, Rupak Majumdar, and Indranil Saha. Synthesis of Fixed-point Programs. In *EMSOFT*, 2013.

[6] Hassan Eldib and Chao Wang. An smt based method for optimizing arithmetic computations in embedded software code. In *FMCAD*, 2013.

[7] Siegfried M. Rump. Verification methods: rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.

[8] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic Optimization of Floating-point Programs with Tunable Precision. In *PLDI*, 2014.

[9] Zeyuan Allen Zhu, Sasa Misailovic, Jonathan A. Kelner, and Martin Rinard. Randomized accuracy-aware program transformations for efficient approximate computations. In *POPL*, 2012.