With the end of classical processors' clock-rate scaling, future gains in performance must be derived from parallelism; however, parallel computer architectures are more complex and harder to program. The difficulty of programming parallel systems is further exacerbated by the difficulty of software tuning. Autotuning is an emerging approach that assists the programmer in managing this complexity, but autotuners are not easy to integrate within existing compilation infrastructures. Furthermore, programs should be optimized not only for performance, but also for energy consumption and approximation accuracy. This scenario calls for new solutions in terms of better program analyses, novel high-level programming models and enhanced software optimization based on advanced autotuning solutions.

Better analyses are proposed for two general-purpose programming languages: static optimization in PHP and autovectorization in C, which is followed by an overview of SIMD programming models for C++. Successively, the thesis introduces three high-level interfaces to program parallel architectures: libWater, which relies on a task-DAG to target heterogeneous clusters; CELERITY, a high-level SYCL/C++ extension based on a combination of compiler analysis and runtime system; and OpenABL, a domain-specific language for agent-based simulations. The next part focuses on tuning techniques and presents different autotuning methods based on machine learning: classification for heterogeneous task partitioning; regression for vectorization cost modeling; ordinal regression for stencil computations autotuning. Finally, two examples of multi-objective tuning scenarios are presented in contexts where performance optimization is in conflict with approximation quality or energy consumption.

Mit dem Ende der klassischen Möglichkeit die Taktrate von Prozessoren zu erhöhen, können zukünftige Leistungssteigerungen nur durch Parallelisierung erreicht werden. Parallele Computerarchitekturen sind jedoch komplexer und schwieriger zu programmieren. Erschwerend hinzu kommen hohe Aufwände bei der Justierung dieser Programme. Autotuning ist ein neuerer Ansatz, der die Entwickler bei der Lösung dieser komplexen Aufgabe unterstützt. Allerdings sind Autotuner nicht einfach in eine bestehende Compiler-Infrastruktur zu integrieren. Darüber hinaus sollten Programme nicht nur hinsichtlich der Leistung, sondern auch des Energieverbrauchs und der Näherungsgenauigkeit optimiert werden. Dieses Szenario erfordert neue Lösungen für bessere Programmanalyse, neue high-level Programmiermodelle und verbesserte Programmoptimierungen basierend auf Autotuning.

Für zwei Allzweck-Programmiersprachen werden bessere Analysen vorgeschlagen: statische Optimierung in PHP und Autovektorisierung in C. In der Arbeit werden nacheinander drei Schnittstellen auf hoher Ebene vorgestellt, um parallele Architekturen zu programmieren: libWater, eines task-DAG für heterogene Cluster; CELERITY, eine high-level SYCL/C++ Erweiterung; OpenABL, eine domänenspezifische Sprache für Agenten-basierte Simulationen. Der nächste Teil konzentriert sich auf Tuning-Techniken und stellt verschiedene Autotuning-Methoden vor, die auf maschinellem Lernen basieren: Klassifizierung für heterogene Aufgabenpartitionierung; Regression zur Kostenmodellierung; ordinale Regression für das Autotuning von Stencil-Berechnungen. Zum Schluss werden zwei Beispiele für ein Mehrzieloptimierungsszenario vorgestellt, bei denen die Leistungsoptimierung im Widerspruch zur Qualität der Approximierung oder dem Energieverbrauch stehen.