Professur Praktische Informatik

OpenTuner: An Extensible Framework for Program Autotuning

Matthias Tietz
Betreuer: Dr. Michael Hofmann

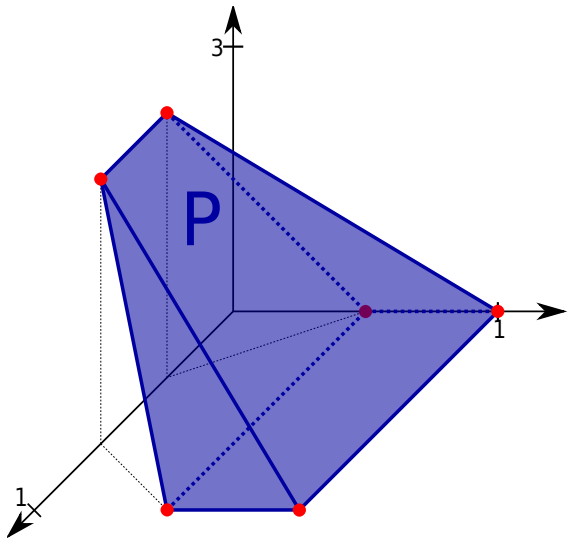
18. November 2016

Gliederung

1. Einleitung

2. Das OpenTuner Framework

- Suchraum: Menge von Parametern die durchsucht werden soll
- geeignete Suchverfahren abhängig von der Beschaffenheit dieser Menge
- komplexe Struktur und Größe des Suchraums macht Handoptimierung oder vollständige Suche unmöglich (bzw. extrem ineffizient)
  $\rightarrow$ Nadel im Heuhaufen



- Ziele:
  - automatisierter und einfacher Optimierungsprozess
  - bessere und portierbare Performance von domänenspezifischen Programmen

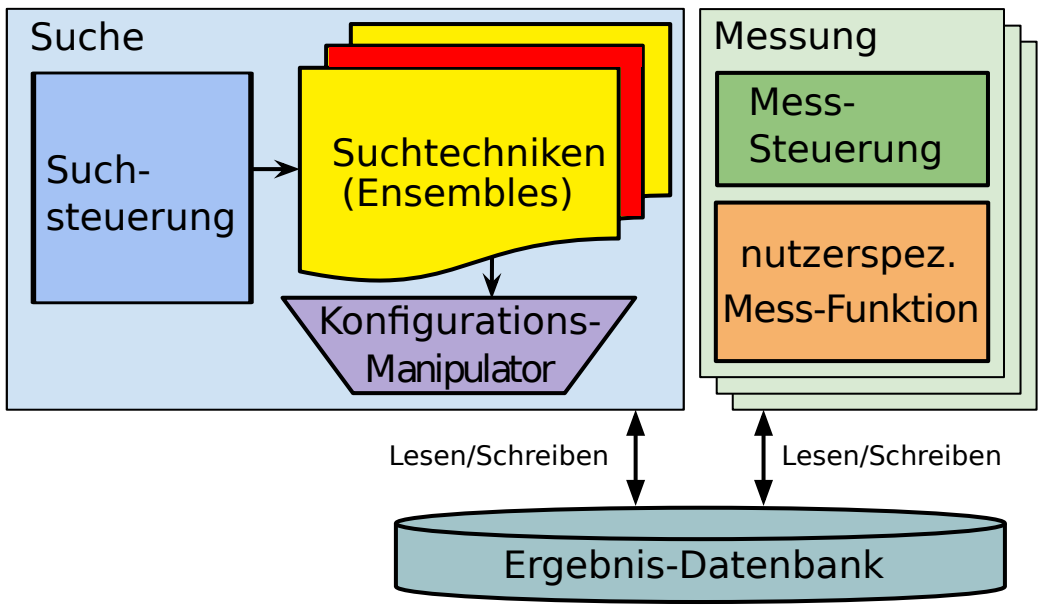
Die 3 wesentlichen Anforderungen an ein Autotuning-Framework:

- 1. Eine passende Konfigurations-Repräsentation
  - Darstellung der domänenspezif. Datenstrukturen und Bedingungen
  - Qualität dieser Repräsentation entscheidend für Effizienz des Autotuners
- 2. Größe des validen Konfigurations-Raumes
  - durch Kürzen des Konfigurations-Raumes geht für viele Probleme gute Lösungen verloren (bei bisherigen Autotunern ist dies gängige Praxis, da vollständige Suche)
  - rießige Konfigurationsräume möglich $\rightarrow$ intelligente Suchtechniken notwendig
- 3. Landschaft des Konfigurations-Raumes
  - Suchräume in der Praxis meist sehr komplex
  - domänenspezif. Suchtechniken notwendig um optimale Lösung effizient zu ermitteln

Deshalb OpenTuner:

- Erstellen domänenspezifischer und multi-objective Programm-Autotuner
- vollständig anpassbare Konfigurations-Repräsentation
- erweiterbare Repräsentation für Suchtechniken und Datentypen
- Kombination mehrerer Suchtechniken (*Ensembles*), dynamische Zuweisung der Testanteile für die jeweiligen Suchtechniken
- einfache Schnittstelle zur Kommunikation mit dem zu optimierenden Programm

- Autotuning-Problem $\rightarrow$ Suchproblem
- Suchraum: Menge der Konfigurationen (Belegung von Parametern)
- Messung: 1 konkrete Konfig. wird gemessen: Ausführung $\rightarrow$ Ergebnis
- Möglichkeit mehrere Messungen parallel auszuführen



Verwendung

- 1. Suchraum definieren (Konfig.-Manipulator)
- 2. `run()`-Methode definieren: Auswerten der Konfig. im Suchraum $\rightarrow$ Ergebnis
- 3. Festlegen des Optimierungsziels (Zeit, Energie, Genauigkeit, Kombination...)
- Umsetzung mittels kleinem Python-Programm (OpenTuner API)

Suchtechniken

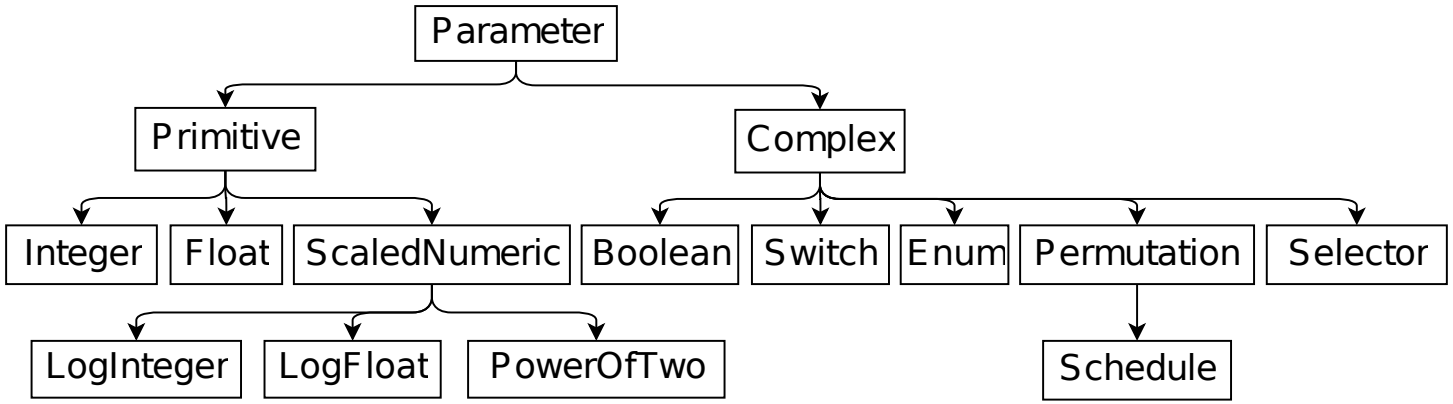
- OpenTuner stellt Suchtechniken für viele Suchraum-Typen bereit
- Ausführen mehrerer Suchtechniken gleichzeitig (Ensembles)
- dynamische Testzuweisung anhand Erfolges dieser Techniken
- erweiterbar: benutzerdefinierte Suchtechniken

Konfigurations-Manipulator

- Abstraktionsschicht zwischen Suchtechnik und roher Konfigurations-Struktur
- Liste der Parameter/Datenstruktur ist dynamisch erweiterbar
- Konfiguration wird als Dictionary verwaltet

Parameter-Typen

- jeder Parametertyp ist verantwortlich für Schnittstelle zwischen roher Parameterrepräsentation und stand. Ansicht dieses Parameters für die Suchtechnik
- Parameterrepräsentation und Abstraktion erweiterbar/konfigurierbar



Primitive Parameter

- numerische Werte mit Unter-/Obergrenze
- `Float` und `LogFloat (-Int)` gleiche Repräsentation in der Konfiguration, aber untersch. Ansicht des zugrundeliegenden Wertes für die Suchtechnik (skaliert)
- Grund: ohne Logskal. würde Effekt der Wertänderung mit steigender Parametergröße sinken
- ähnlich bei `PowerOfTwo` $\rightarrow$ Quadrat nur zulässiger Wert des Parameters

Komplexe Parameter

- haben ein variables Set an Manipulatoren, welche stochastische Änderungen an den Parametern vornehmen
- einfach domänenspezif. Strukturen zum Suchraum hinzuzufügen
- `Boolean, Switch` und `Enum` bewusst als komplex. Parameter, da Suchtechniken bei primitiven Parametern nach Gradients (Steigungen) suchen. Diese Parameter sind aber ungeordnete Sammlung $\rightarrow$ es existiert dafür kein Gradient.
- `Permutation` : Liste von Werte inkl. Manipulatoren zur randomisierten Änderung der Reihenfolge
- `Schedule` ist Sonderfall von `Permutation` : topolog. Sortierung nach jeder Änderung
- `Selector` : Mapping von Integer-Input auf Enum-Type (Darstellung als Baum)

Parameter-Interaktion

- Zusätzlich existieren erweiterbare Methoden für die Suchtechniken um zwischen mehreren Parametern zu interagieren. (z.B. Differenz-Funktion)

Optimierungsziele

- OpenTuner unterstützt mehrere Ziele, standardmäßig wird nach der `Zeit` optimiert
- `Genauigkeit, Energie, Größe` oder ein nutzerdef. Ziel
- Es können auch mehrere Ziele zugleich verfolgt werden, bspw. `Genauigkeit einhalten, gleichzeitig Zeit minimieren`

Suchen und Messen

- kommunizieren ausschließlich über die Ergebnis-Datenbank
- Motivation für die Unterteilung zwischen diesen beiden Modulen:
  - Ermöglichen der Parallelität zwischen mehreren Prozessen (Suchen und Messen)
  - Autotuning während Ausführung der Anwendung oder in Wartezeit (Online/Sideline Learning)
  - Mess-Modul einfach ersetzbar ohne das gesamte Framework zu modifizieren (Domäne: Embedded/Mobil $\rightarrow$ leichtgewichtiges Mess-Modul)

Ergebnis-Datenbank

- vollfunktionale SQL-Datenbank
- alle grundlegenden DB-Typen unterstützt, default: SQLite
- Abfragen und Eintragen der Ergebnisse in einer Vielzahl von Möglichkeiten
- nützlich für die Performance-Beobachtung der Suchtechniken