



OpenTuner: An Extensible Framework for Program Autotuning

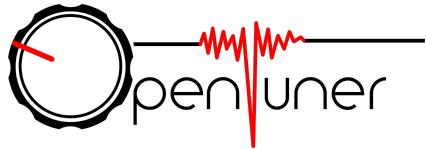
Matthias Tietz

Hauptseminar
Bachelor Informatik

10.01.2017



1. Einleitung
2. OpenTuner Framework
3. Anwendungsbeispiele
4. Aktueller Stand
5. Zusammenfassung



1. Einleitung

2. OpenTuner Framework

3. Anwendungsbeispiele

4. Aktueller Stand

5. Zusammenfassung

Paper

- ▶ OpenTuner: An Extensible Framework for Program Autotuning
- ▶ Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, Saman Amarasinghe (MIT 2014)

Motivation

- ▶ Tuning-Prozess: automatisierbar, effizient?
- ▶ bestehende Autotuning-Frameworks domänenspezifisch (ATLAS, FFTW)

OpenTuner:

- ▶ komfortables Optimierungsverfahren
- ▶ mehrere Optimierungsziele kombinierbar, hohe Erweiterbarkeit
- ▶ Performance-Portabilität

Wichtige Begriffe:

- ▶ **Parameter:** Variable eines bestimmten Typs
- ▶ **Konfiguration:** konkrete Wert-Zuweisung (Parameter)
- ▶ **Konfigurations-Manipulator:** Bearbeiten der Parameter

- ▶ **Suchraum:** zu durchsuchende Menge von Konfigurationen
- ▶ **Suchtechnik:** Suchraum erforschen, Anfragen für Messung
- ▶ **Messresultat:** Ergebnis der Messung, abhängig von Optimierungsziel(en)
- ▶ **Ergebnisdatenbank:** Festhalten aller Ergebnisse des Tuning-Vorgangs

Herausforderungen bei der Entwicklung eines Autotuning-Frameworks:

- ▶ 1. Eine passende Konfigurations-Repräsentation
 - ▶ Darstellung der Datenstrukturen und Bedingungen
 - ▶ entscheidend für Effizienz des Autotuners

- ▶ 2. Größe des gültigen Suchraumes
 - ▶ riesige Konfigurationsräume möglich → intelligente Suchtechniken
 - ▶ Kürzen des Suchraumes → Verlieren guter Lösungen

- ▶ 3. Beschaffenheit des Suchraumes
 - ▶ Suchräume in der Praxis meist sehr komplex
 - ▶ domänenspezifische Suchtechniken notwendig

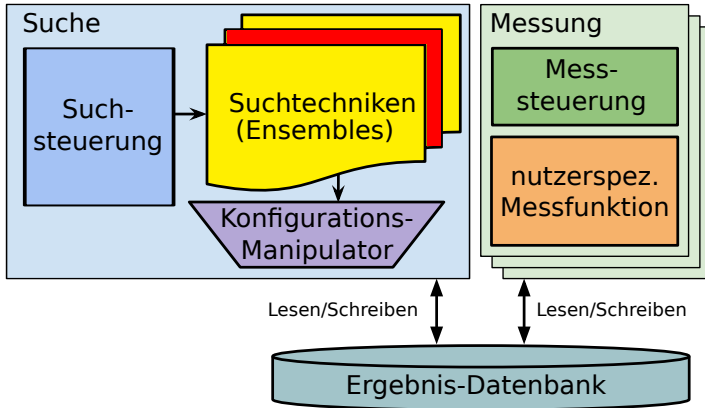
1. Einleitung

2. OpenTuner Framework

3. Anwendungsbeispiele

4. Aktueller Stand

5. Zusammenfassung



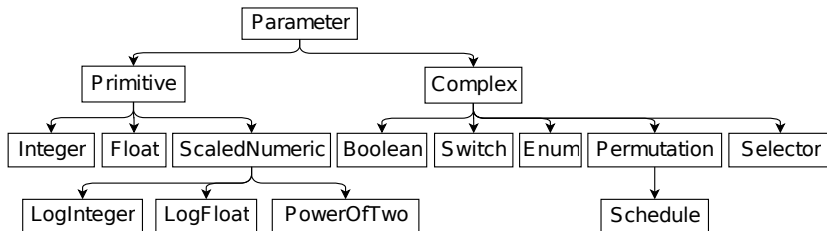
Verwendung

- ▶ 1. Suchraum definieren
- ▶ 2. `run()`-Methode definieren: Auswerten der Konfiguration → Ergebnis
- ▶ 3. Festlegen des Optimierungsziels
- ▶ Umsetzung: Python-Programm (OpenTuner API)

Suchtechniken

- ▶ OpenTuner stellt grundlegende Suchtechniken bereit
- ▶ Ausführen mehrerer Suchtechniken (Ensembles)
- ▶ Austausch von Ergebnissen über Datenbank
- ▶ dynamische Testzuweisung anhand Erfolg der Suchtechniken

Parameter-Arten



Primitive Parameter

- ▶ numerische Werte mit Unter-/Obergrenze
- ▶ Float und LogFloat(-Int) gleiche Repräsentation, Werte skaliert

Komplexe Parameter

- ▶ `Permutation`: Liste von Werten inkl. Reihenfolge-Manipulator
- ▶ `Schedule`, `Selector`

Optimierungsziele

- ▶ OpenTuner unterstützt mehrere Ziele, default: `time`
- ▶ `accuracy`, `energy`, `size`, `confidence` oder ein nutzerdef. Ziel
- ▶ Ziel-Kombination: `time-accuracy`

1. Einleitung

2. OpenTuner Framework

3. Anwendungsbeispiele

4. Aktueller Stand

5. Zusammenfassung

GCC/G++ Flags

- ▶ klassischer Parameter-Autotuner
- ▶ unterstützte Flags: `g++ --help=optimizers`
- ▶ Parameter inkl. zulässiger Wertebereiche: `params.def` (gcc source code)
- ▶ Zielprogramme zur Laufzeit-Optimierung:
 - ▶ `fft.c` - Schnelle Fourier-Transformation
 - ▶ `matrixmultiply.cpp` - Matrix-Multiplikation

```
import optuner
from optuner import ConfigurationManipulator
from optuner import EnumParameter
from optuner import IntegerParameter
from optuner import MeasurementInterface
from optuner import Result

GCC_FLAGS = [
    'align-functions', 'align-jumps', 'align-labels',
    'branch-count-reg', 'branch-probabilities',
    # ... (176 total)
]

# (name, min, max)
GCC_PARAMS = [
    ('early-inlining-insns', 0, 1000),
    ('gcse-cost-distance-ratio', 0, 100),
    # ... (145 total)
]
```

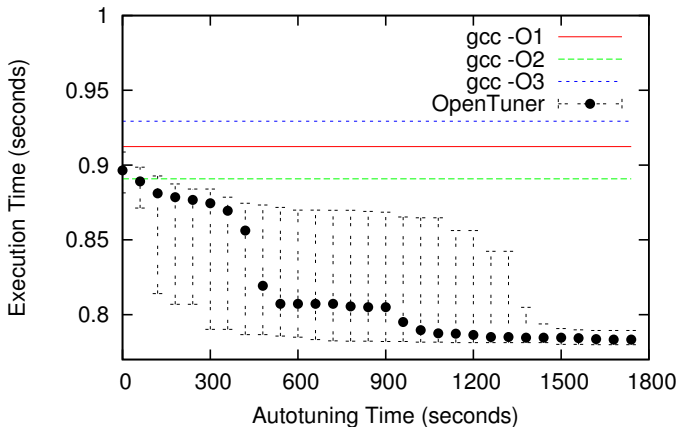
```

class GccFlagsTuner(MeasurementInterface):

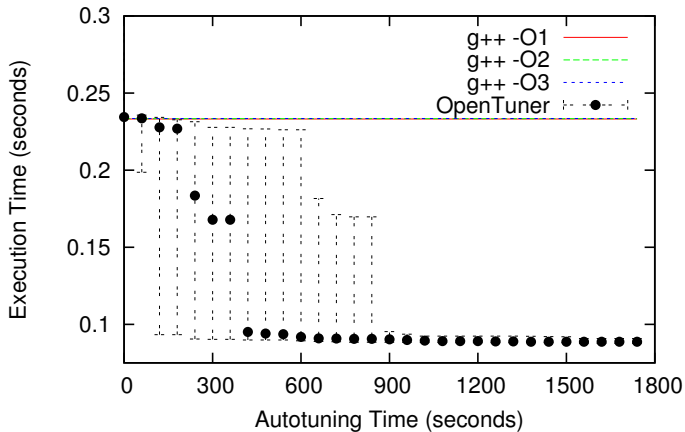
    def manipulator(self):
        """
        Define the search space by creating a
        ConfigurationManipulator
        """
        manipulator = ConfigurationManipulator()
        manipulator.add_parameter(
            IntegerParameter('opt_level', 0, 3))
        for flag in GCC_FLAGS:
            manipulator.add_parameter(
                EnumParameter(flag,
                              ['on', 'off', 'default']))
        for param, min, max in GCC_PARAMS:
            manipulator.add_parameter(
                IntegerParameter(param, min, max))
        return manipulator
  
```

```
def run(self, desired_result, input, limit):
    """
    Compile and run a given configuration then
    return performance
    """
    cfg = desired_result.configuration.data
    gcc_cmd = 'g++ raytracer.cpp -o ./tmp.bin'
    gcc_cmd += ' -O{0}'.format(cfg['opt_level'])
    for flag in GCC_FLAGS:
        if cfg[flag] == 'on':
            gcc_cmd += ' -f{0}'.format(flag)
        elif cfg[flag] == 'off':
            gcc_cmd += ' -fno-{0}'.format(flag)
    for param, min, max in GCC_PARAMS:
        gcc_cmd += ' --param {0}={1}'.format(
            param, cfg[param])

    compile_result = self.call_program(gcc_cmd)
    assert compile_result['returncode'] == 0
    run_result = self.call_program('./tmp.bin')
    assert run_result['returncode'] == 0
    return Result(time=run_result['time'])
```

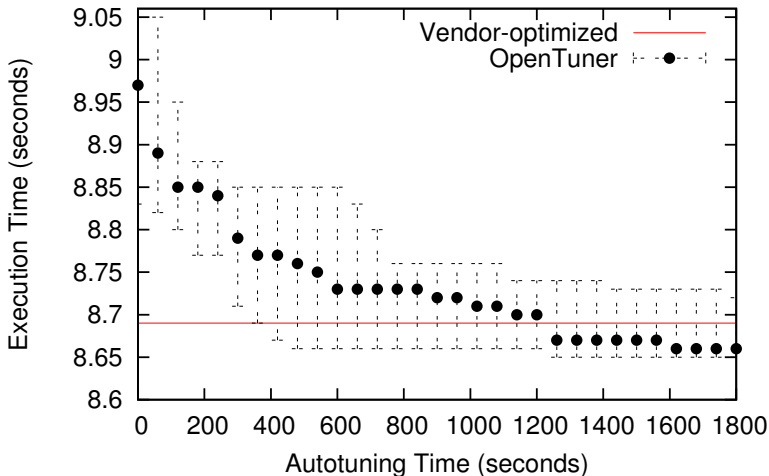
(a) `fft.c`



(b) matrixmultiply.cpp

High Performance Linpack Benchmark (HPL)

- ▶ Bestimmen der floating point performance von Computern
- ▶ Kriterium für Wahl der Top 500 Supercomputer
- ▶ HPL misst Geschwindigkeit für Lösen großer LGS
- ▶ ca. 15 verschiedene Parameter
- ▶ größter Performance-Einfluss: Matrix-Blockgröße
- ▶ HPL besitzt eingebauten Autotuner (vollst. Suche)
- ▶ OpenTuner im Vergleich zu HPL-Implementierung von Intel



Ähnliche Anwendungen des Frameworks

Halide

- ▶ Sprache und Compiler für Bildverarbeitung
- ▶ Ausführungsreihenfolge in Grafik-Pipeline
- ▶ Pipeline-Schedule bestimmt die Performance

PetaBricks

- ▶ Kombination versch. Algorithmen (`algorithmic selectors`)
- ▶ `block sizes, thread counts, iteration counts...`

Super Mario

- ▶ Absolvieren des ersten Levels: Sequenz von Tasten-Eingaben (Konfig.)
- ▶ Tasten-Eingaben → Speichern in Datei → Abspielen in NES-Emulator
- ▶ Wiedergabe in Emulator schneller als in Echtzeit
- ▶ Auswerten mehrerer Instanzen des Emulators parallel

- ▶ Ziel: Maximieren der #Pixel die Mario nach rechts läuft
- ▶ Suchraum: horizontale Bewegung nach rechts
- ▶ Eingaben-Modellierung:
 - ▶ welche Taste
 - ▶ zu welchem Zeitpunkt (Frame)
 - ▶ wie lange gedrückt wird

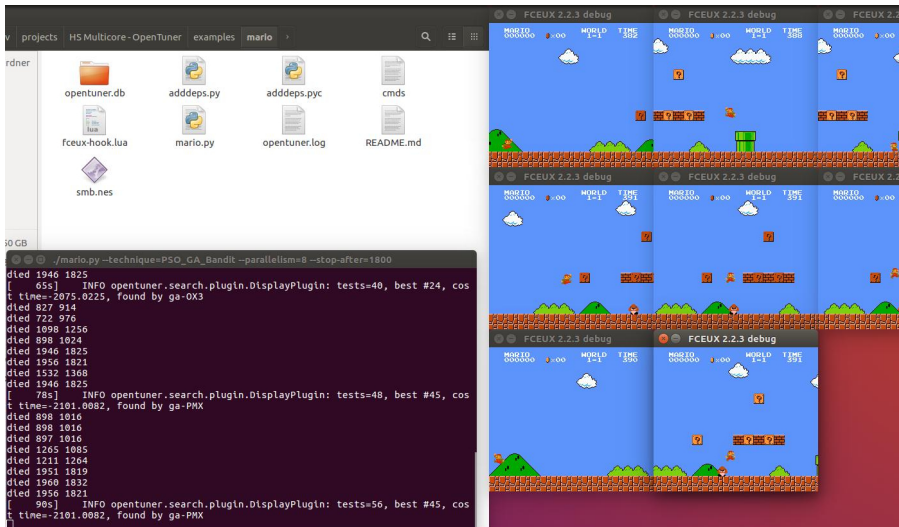


Abbildung: Tuning-Prozess

```
./mario.py --technique=PSO_GA_Bandit --parallelism=8 --stop-after=1800
died 2763 2360
died 2588 2149
died 2763 2360
died 2763 2360
[ 168s] INFO opentuner.search.plugin.DisplayPlugin: tests=96, best #75, cost time=-3234.8174, found by ga-0X1
died 898 1016
died 1215 1262
died 1558 1538
died 2009 1931
died 2009 1931
died 2763 2360
died 1764 1766
won 3161 2356
[ 183s] INFO opentuner.search.plugin.DisplayPlugin: tests=104, best #101, cost time=-4241.0531, found by ga-0X3
died 296 536
died 704 757
died 826 912
died 898 1016
died 1522 1379
died 1492 1612
died 1765 1765
won 3161 2453
```

Abbildung: Ziel erreicht

Demo: Erstes Level in Super Mario

1. Einleitung
2. OpenTuner Framework
3. Anwendungsbeispiele
- 4. Aktueller Stand**
5. Zusammenfassung

- ▶ www.github.com/jansel/opentuner
- ▶ MIT-Lizenz
- ▶ Entwicklung 2014-2015
- ▶ Erweiterung durch Nutzer
- ▶ gute Dokumentation
- ▶ Ziel: Einsatz in bestehenden und neuen Domänen

1. Einleitung

2. OpenTuner Framework

3. Anwendungsbeispiele

4. Aktueller Stand

5. Zusammenfassung



PRO

- ▶ Flexibilität und Erweiterbarkeit
- ▶ einfache und klare Handhabung
- ▶ viele Beispielanwendungen
- ▶ mehrere Optimierungsziele (Kombination)
- ▶ bessere Ergebnisse als bisherige Autotuner

CONTRA

- ▶ Optimierungsgrad variiert stark
- ▶ Trade-off: Zeitaufwand \leftrightarrow Nutzen

Quellen

-  Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, Saman Amarasinghe. *OpenTuner: An Extensible Framework for Program Autotuning*. 2014.
-  Shoaib Kamil. *Hands on session*. Programming Language Design and Implementation 2015
-  www.github.com/jansel/opentuner (08.01.2017)
-  www.opentuner.org (05.12.2016)