CFFI

Python im Hochleistungsrechnen

Leon Fritz

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

17.06.2019

Gliederung

Einleitung

- Wofür steht CFFI (C Foreign Function Interface)
- Verwendungszweck von CFFI

Warum ein CFFI nutzen?

- Neue funktionalität in Python einbinden
  - USB-Zugriff etc.
  - proprietäre Bibliotheken verwenden (bsp. Steam API)
  - Sicherheitskritische Bibliotheken (Kryptographie)
  - C,C++ Standardbibliotheken und Platformabhängige Bibliotheken verfügbar
- Geschwindigkeitsvorteile aus anderen Sprachen nutzen
  - aufwendige Berechnungen (z.B. Matrizenmultiplikation)
  - Unterschiede in Daten feststellen, große Daten lesen und/oder verarbeiten
  - vorteile bei Datenstrukturen (z.B. traversieren von großen Bäumen)

Vergleich mit Alternativen

- ctypes (x) [Ctya]
  - Teil der Python standard bibliothek
  - Langsamer als CFFI [Ctyb]
  - C-Deklarationen müssen ctypes beschrieben werden in Python-Syntax
- Cython etc (Python zu C kompilieren) [Cyt]
  - Der gesamte Python code muss extra kompiliert werden von einem C-Kompiler
  - Abhängigkeit von der C-Umgebung
  - Platformabhängigkeit
  - Für mehr Geschwindigkeit muss der Code angepasst werden
- Selber C++ Extensions schreiben [Pyt]
  - Erfordert wissen über C++
  - Sehr aufwendig
  - Funktioniert nur auf einer implementation von Python (CPython)
  - Platformabhängigkeit

Überblick

- Man muss nicht extra einen neue Sprache lernen
- Kompatibel mit Python2, Python3, CPython und PyPy
- Jeder Codeteil der mit Python zu tun hat ist auch in Python
- Funktioniert auf dem ABI und API level

Application Binary Interface [CFF]

- Schnittstelle auf Maschinenebene
  - Registerebene
  - Architekturabhängig
- Code welche zusammengelinkt werden für eine kompilierte ausführbare Datei muss auf der gleichen ABI laufen
- Kommunikation zwischen unterschiedlichen ABI nur mit RPC oder ähnlichen möglich
- CPythons ABI ist unterschiedlich von PyPys ABI -> C Extensions für den einen laufen nicht auf den anderen. [Cpy]

Application Programming Interface [CFF]

- Schnittstelle auf Anwendungsebene
- Die API von CPython und PyPy sind bsp. kompatibel -> Kann Python programme auf PyPy ausführen.
- Python code hat auf jeder Platform die selbe API

Installation von CFFI [CFF]

- Auf Windows - prerequisite Windows Build Tools
- Linux - prequisite gcc oder ähnliches build too
- python -m pip install cffi
- fertig!

Meistgenutze CFFI Funktionen [CFF]

- ffi = cffi.FFI() – Eine Instanz von CFFI initialisieren
- ffi.cast("datentyp", daten) - Casten wie in C
- ffi.new("datentyp", optionalezuweisung) - Speicher allozieren wie in C
- ffi.cdef(code) – C header Dateien deklarieren
- ffi.dlopen(bibliothek) – Eine Bibliothek laden z.B. eine dll mit Crypto-Funktionen die wir gebrauchen können.
- ffi.set_source(modulname, source, schlüsselwörter)– Erklärt wo er die definitionen findet wie in C
- ffi.compile(verbos) – Erlaubt uns ausführbare Dateien zu kompilieren

Was brauchen wir für ein Hello World?

- CFFI importieren
- CFFI initialisieren
- Funktionsdeklarationen (bsp. printf)
- Funktionsdefinition
- Eine Zeichenkette

Hello World mit CFFI

```
1 from cffi import FFI
2 ffi = FFI()
3 ffi.cdef("int printf(const char *format, ...);")
4 C = ffi.dlopen("msvcrt.dll") #in Python2 arg=Null
5 zeichenkette = ffi.new("char[]", b"world")
6 C.printf(b"hi there, %s.\n", zeichenkette)
```

printf

Live-Demo vom Hello World mit CFFI

Typumwandlung mit CFFI

- C/C++ Datentypen werden von FFI anders behandelt als Pythons
- In C/C++ kann man Datentypen umwandeln
- Um in Python Umwandlungen möglich zu machen gibt es ffi.cast

Typumwandlung mit CFFI

```
1 ffi = FFI()
2 arg = ffi.new("char[]", b"D")
3 print(arg[0])
4 ffi.cast("int8_t", arg)
5 ffi.cast("intptr_t", arg)
6 hex(int(ffi.cast("intptr_t", arg)))
7 ffi.addressof(arg, 0)
```

Umwandlungsbeispiele [CFF]

Modul in C++ für CFFI

- CFFI bedient sich der exporte der Bibliotheken
- Exporte werden von den Entwicklern selber in den Modulen angegeben
- Keyword: export "C"benötigt
- Module welche so erstellt werden können auch von anderen Sprachen verwendet werden
- Können wir auch eine Bibliothek welche mit CFFI funktioniert erstellen?

Modul in C++ für CFFI

```
1 from cffi import FFI
2 ffi = FFI()
3 ffi.cdef("""
4     int MessageBoxes(const char *test);
5 """)
6 C = ffi.dlopen("box.dll")
7 arg = ffi.new("char[]", b"world")
8 C.MessageBoxes(arg);
```

Eine normale C/C++-Biblitothek importieren und nutzen [CFF]

Live-Demo von unseren Modul mit CFFI

Ein Modul kompilieren mit CFFI

- Mit CFFI können Python-Extensions erstellt werden
- Verwendet dafür auf dem OS-installierte Build-Tools (z.B. gcc, msvc)
- Die inline-API mode ermöglicht direktes nutzen von C/C++ Code in Python (ffi.verify)

Ein Modul kompilieren mit CFFI

```
1  from cffi import FFI
2  ffi = FFI()
3  ffi.cdef(
4      """
5      int printf(const char *format, ...);
6      """
7  )
8  C = ffi.set_source("nicemod",
9      """
10     #include <stdio.h>
11     """
12 )
13 ffi.compile()
14 C.printf(b"Hi there!\n")
```

Modulkompilieren [CFF]

Ein Modul kompilieren mit CFFI

```
1 from nicemod import ffi, lib
2 from pprint import pprint
3 lib.printf(b"wow")
```

Das Modul ausführen [CFF]

Live-Demo von Kompilierungsbeispiel

Videospiel-Cheat mithilfe von CFFI

- Spiel hat Speicher, wir können darauf zugreifen
- Speichermodifikationen erlauben änderungen des Prozessverhaltens
- Nahezu alle modernen Spiele und Programme haben Debug-Optionen etc., bsp. CSGO
- C/C++ Modul bietet Python Funktionen um den Prozessspeicher zu lesen und zu verändern

Videospiel-Cheat mithilfe von CFFI

```
1  #Lets patch 1 byte to enable any debug cheat convars,
       ↪ attempt nr.2
2  from cffi import FFI
3  ffi = FFI()
4  ffi.cdef("""
5      int32_t RPM(intptr_t address);
6      bool WPM(int address, const char* writethis,
           ↪ int32_t writelen);
7      intptr_t GetModuleAddress(const char *module);
8      intptr_t LeyFindPatternIDAStyle(void* addr,
           ↪ size_t len, const char* mask);
9  """)
10
11 #load our module
12 C = ffi.dlopen("manipmem_game.dll")
13 #prepare the strings for the module
14 targetmodule = ffi.new("char[]", b"engine.dll")
15 lookforsig = ffi.new("char[]", b"75 3A 38 05 ? ? ? ?
```

Live-Demo vom Videospiel-Cheat

Zusammenfassung/Fazit

- CFFI erlaubt es auf einfache weise C-Funktionen aufzurufen
- API ist einfach zu lernen da es keine Wrapper-Sprache etc. gibt
- Erlaubt API Zugriff was wiederum sicher ist als Zugriff auf ABI Zugriff
- Hat eine hohe Kompabilität (Python2, Python3, PyPy, etc.)
- Ist grundsätzlich die goto Variante :)

Literatur

[CFF] https://cffi.readthedocs.io/en/latest/. Zugriff am: 09.06.2019.

[Cpy] https://morepypy.blogspot.com/2018/09/inside-cpyext-why-emulating-cpython-c.html. Zugriff am: 09.06.2019.

[Ctya] https://docs.python.org/3/library/ctypes.html. Zugriff am: 09.06.2019.

[Ctyb] https://github.com/traildb/traildb-python/issues/4. Zugriff am: 17.06.2019.

[Cyt] http://docs.cython.org/en/latest/. Zugriff am: 09.06.2019.

[Pyt] https: