Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

Python on high performance computing (HPC)

vorgelegt von

Pierre Ruge

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

Studiengang:	Informatik
Matrikelnummer:	6676727
Erstgutachter:	Hermann Lenhart
Zweitgutachter:	Nabeeh Juma
Betreuer:	Hermann Lenhart, Nabeeh Juma

Hamburg, 2018-08-31

Abstract

Die Ausarbeitung zum Thema Python im Hochleistungsrechnen befasst sich anfangs sehr intensiv mit der Programmiersprache Python. Dabei werden die Stärken und Schwächen der Sprache intensiv beleuchtet. Anschließend wird Python mit der Programmiersprache C, welche einer der beiden vorherrschenden Sprachen im Hochleistungsrechnen ist, verglichen. Dabei werden die erläuterten Vor- und Nachteile mit denen von der Sprache C in Relation gesetzt. Abschließend wird ein Fazit gezogen, welches den Nutzen von Python in der Informatik und speziell dem Hochleistungsrechnen einschätzt.

Inhaltsverzeichnis

1 Einleitung

In dieser Ausarbeitung wird das Hochleistungsrechnen von verschiedenen Seiten beleuchtet. Dabei wird insbesondere die Programmiersprache Python genauer betrachtet. Nachdem das Augenmerk anfangs auf die speziellen Eigenschaften und Besonderheiten der Programmiersprache Python gerichtet wurde, wird noch einmal auf die wichtigen Merkmale des Hochleistungsrechnen eingegangen. Dabei geht es auch um die im Hochleistungsrechnen, neben Fortran, vorherrschenden Programmiersprachen C. Insbesondere sollen die Vor- und Nachteile, Gemeinsamkeiten und Unterschiede zwischen der Programmiersprache Python und, der in dieser Ausarbeitung näher betrachteten Sprache, C behandelt werden. Dabei basiert diese Ausarbeitung basiert in erster Linie auf dem wissenschaftlichen Artikel Python for Scientific Computing von Travis Oliphant [Oli07], in welchem die Programmiersprache Python im Vordergrund steht. Der Autor stellt gleich zu Anfang seines Artikels die These "Python is an interpreted language with expressive syntax that some have compared to executable pseudocode."[Oli07] auf. Diese Ausarbeitung versucht in jeder Hinsicht diese Aussage zu überprüfen, zu untermauern und ihre Aussagekraft anschließend auf den Nutzen dieser Programmiersprache für das Hochleistungsrechnen zu ermitteln. Um dies gleich als ersten Einstieg in Python zu nutzen, halten wir uns den standardisierten Ablauf eines jeden Programmes, anhand eines einfachen Hello World Programmes, vor Augen. Ein solches Programm startet, gibt über einen Print-Befehl den String Hello World auf der Konsole aus und beendet sich wieder. In der Sprache C wird hierfür eine Main Funktion benötigt, die mit oder ohne Parameter die Umgebung des Programmes bildet und es ausführt. Innerhalb dieser Main Funktion wird dann mit dem Print-Befehl die Ausgabe geschrieben und das Programm mit einem Rückgabewert beendet. In der anderen im Hochleistungsrechnen vorherrschenden Programmiersprache Fortran würde erst ein Programm mit seinem Namen gestartet, dann der Print-Befehl ausgeführt und abschließend das Programm mit seinem Namen beendet. In Python wird keine Main Funktion oder ein Programmaufruf benötigt. Der einfache Aufruf des Print-Befehls ist vollkommen ausreichend, um das Hello World Programm fertig zu stellen. Schon dieses einfache Beispiel zeigt, wie sehr die Programmiersprache Python dem eigentlichen Programmablauf eines Pseudocodes ähnelt und bestärkt die Aussage von Oliphant. Ob die Aussage auch auf andere Aspekte von Python zutrifft und inwiefern sich diese Eigenschaft für das Hochleistungsrechnen zu Nutze gemacht werden kann, ist die grundlegende Frage dieser Ausarbeitung.

2 Python

Die Programmiersprache Python kommt mit einigen sehr nützlichen Eigenschaften. Der wohl wichtigste Aspekt ist die Open Source Lizenz, die es nicht nur ermöglicht den Code der Sprache eigenständig für seine Zwecke zu modifizieren und erweitern, sondern auch die kommerzielle Nutzung mit Python geschriebener Programme zu legitimieren. Dabei ist es möglich, den mit Python geschrieben Code plattformübergreifend zu verwenden. Im Gegensatz zu viele Programmiersprachen, die einen Programmierer dazu zwingen sich für eine Art der Programmierung zu entscheiden, ist es in Python Möglich sowohl prozedural, als auch objektorientiert und sogar funktional zu Programmieren. Daher ist es jedem Programmierer möglich ein Programm, in dieser Sprache, in seinem bevorzugten Stil zu schreiben, aber auch auf andere zurück zugreifen beziehungsweise verschiedene Stile miteinander zu kombinieren. Der integrierte mächtige Interpreter ermöglicht außerdem schnelle Echtzeitprogrammierung. Die mögliche Interaktion von Python mit anderer Software ermöglicht es neue in Python geschriebene Programme in bereits bestehende zu integrieren. Außerdem existieren für Python sehr viele verschiedene Module, welche beispielsweise die Datensammlung und -speicherung oder die E-Mail Verwaltung ermöglichen. All diese Funktionen können mit Python nicht nur Code technisch umgesetzt werden, sondern auch über eine erstellbare grafische Benutzeroberfläche (GUI) für verschiedene Nutzer einfach dargestellt werden. [Oli07]

2.1 Python 2 und 3

Zum jetzigen Zeitpunkt ist es sehr empfehlenswert für neue Programme die neuste Python Version 3.7 zu verwenden. Dennoch sollte man sich bewusst sein, dass viele Python Programme in Python 2 geschrieben sind und noch immer geschrieben werden. Der wichtigste Punkt hierbei ist, dass die beiden Versionen 2 und 3 nicht kompatibel sind. Python 2 gibt es seit dem 16.10.2000 und beinhaltete gegenüber seinem Vorgänger eine Garbage Collection und unterstützte nun Unicode. Seit der Version 2.6 ist eine Inkompatibilitätserkennung zu Python 3 integriert. Der Grund für die Fortführung von Python 2 neben Python 3 liegt an der Inkompatibilität. Da die Python Entwickler ihre Nutzer nicht sofort dazu zwingen wollten ihre alten Programme anzupassen. Außerdem, waren zur Veröffentlichung von Python 3, am 03.12.2008, nur die wenigsten der viele erweiterten Module an den neuen Standard angepasst und selbst ein umschreiben sämtlicher Python 2 Programme währe problematisch gewesen, weil auf die Nutzung dieser Module hätte verzichtet werden müssen. Die Inkompatibilität der Versionen entstand durch die Entfernung redundanter Befehlssätze und veralteter Konstruktionen. Inzwischen sind 10 Jahre vergangen und die meisten Module sind Python 3 kompatibel. Seit dem 27.06.2018

ist die neuste Version Python 3.7 verfügbar und neue Programme werden bevorzugt in Python 3 geschrieben. Des Weiteren wurde die Einstellung der Python 2 Unterstützung seitens der Entwickler für das Jahr 2020 angekündigt. [Wik18]

Im Folgenden werden zwei der prägnantesten Änderungen vorgestellt. Die erste begegnet einem bereits bei seinem Einstieg in die Programmiersprache mit dem ersten Programm, dem Hello World. Wie in der Einleitung bereits erwähnt ist in Python hierfür nur die Ausführung des Print-Befehls erforderlich. Dieser hat sich allerdings von Python 2 nach 3 verändert. So wurde in Python 2 noch eine Print-Anweisung verwendet mit dem Schlüsselwort print. In Python 3 hingegen ist der Print-Befehl nun eine Funktion die Aufgerufen werden muss und einen String als Parameter übergeben bekommt. So wurde aus:
Python 2: print "Hello World!"
Python 3: print("Hello World!")
Sämtliche Systemausgaben sind somit hinfällig und nicht mehr Versionsübergreifend. Beim zweiten Beispiel handelt sich um die in Programmen oft verwendete Integer-Division. Hat man in Python 2 drei durch zwei geteilt, so erhielt man eine als Integer gerundete 1, da die zwei einmal in die drei passt. In Python 3 hingegen erhält man den Float 1.5, da dies das eigentliche Ergebnis darstellt:
Python 2: $3 / 2 = 1$
Python 3: $3 / 2 = 1.5$
An allen Stellen von Python 2 Programmen, an denen sich Programmierer auf die Tatsache verlassen haben, dass die Division zweier Integer einen Integer ergeben, würde bei einer Benutzung des alten Codes in Python 3 ein Problem auftauchen. [Ras14]

2.2 Pythons Besonderheiten

Im Folgenden werden die speziellen Besonderheiten Pythons, welche sich so nur in wenigen anderen Programmiersprachen finden erläutert. Einem Programmierer, der die Sprache Python neu lernt wird als erstes die im Vergleich zu anderen Sprachen deutlich geringere Dichte an Klammern auffallen. Den anders als die meisten Programmiersprachen nutzt Python zur Strukturierung seiner Abläufe keine Blöcke, die durch Mengenklammern umrahmt sind, sondern regelt alles durch die Einrückung. Dabei sind die Anzahl der Einrückung und seine Art, ob Leertasten oder Tabulatoren, vollkommen irrelevant. Dies ist anders als in Makefiles, wo die Einrückung nur durch Tabulator realisiert wird und ansonsten zu Fehlern führt. In Python ist es einzig und allein wichtig die Einrückung kontinuierlich durchzuführen. Zur besseren Lesbarkeit empfiehlt sich trotzdem eine Einrückung von mindestens vier Leerzeichen pro Einrückungstiefe. Ist dies verstanden, werden als nächstes die ersten Variablen folgen, die definiert werden. Hierbei ist es nicht nötig einen Typ für die Variable mit anzugeben, da Python mit einer dynamischen Typisierung arbeitet und sich der Typ einer Variable zur Laufzeit durchaus von einem Integer zu einem Float oder gar einem String ändern kann. Benötigt man einen bestimmten Typ, so lässt sich der aktuelle problemlos abfragen und im Zweifelsfall ein bestimmter Typ erzwingen. Auch ist es in Python möglich, wenn es benötigt wird, eigene

Typen zu definieren. Wie in den meisten Programmiersprachen sind auch in Python einige Grundstrukturen enthalten. Dazu gehören Listen und Wörterbücher. Die Listen in Python sind sehr besonders, da sie Elemente verschiedenen Typs enthalten können. So ist es möglich die Liste [”one”, 2, 3.0] zu erzeugen, welche einen String, einen Integer und ein Element des Typs Float enthält. Da die Liste ebenfalls ein Typ ist, kann sie natürlich auch enthalten sein. Dabei lässt sich in der Liste l = [”one”, 2, 3.0, [”Liste”, ”in”, ”List”]] ganz einfach mit l[3][1] auf das Element ”in” zugreifen. Die Wörterbücher lassen sich genauso einfach implementieren. So schreibt der Aufruf print(f”{a[”one”]}”) eine 1 auf die Konsole, für a = {2: ”two”, ”one”: 1}. Dabei weist das f, vor dem String in der Print Funktion, daraufhin, dass es sich um einen formatierten String handelt. In einem formatierten String lassen sich Variablen oder Ausdrücke in Mengenklammern übergeben werden so in den String eingefügt. Ruft man a nun mit seinem anderen Schlüsselwort 2 auf, würde der String ”two” auf die Konsole geschrieben werden. Des Weiteren ist eine Dateiverwaltung in Python integriert, die es mit den Funktionen open(”filename”, ”mode”), mit mode = r, w, a oder r+, write(”message”), read() und close() ermöglicht Dateien zu öffnen, zu bearbeiten, zu lesen und zu schließen. Sollten weitere Mechanismen von Nöten sein, die nicht direkt in Python integriert sind, lassen sie sich meist in einem der vielen, bereits erwähnten, Module für Python finden. Dabei gibt es in Python die Möglichkeit ganze Module, in ihnen enthaltenen Klassen oder einzelne Funktionen zu importieren. So importiert der Befehl from numpy.linalg import inv eine Funktion, aus der Klasse Lineare Algebra des Moduls numpy, welche das Inverse einer Matrix berechnet. [Oli07]

Diese Besonderheiten vereinfachen in vielerlei Hinsicht die Lesbarkeit des Python Codes. Die entfallenden Klammern und die dynamische Typisierung entlasten das Auge beim Lesen und die einfache Verwendung von Funktionen und integrierter Grundstrukturen bilden fast schon ausgeschrieben Sätze. Um dies in einem Beispiel zu betrachten, werfen wir einen Blick auf die nachfolgende Funktion, welche den Sinus Cardinalis ($\frac{\sin(\pi x)}{\pi x}$) berechnet:

```
from math import sin, pi
def sinc(x):
    try:
        val = (x*pi)
        return sin(val) / val
    except ZeroDivisionError:
        return 1.0
output = [sinc(x) for x in input] [Oli07]
```

Nach dem Import der benötigten Funktion sin und der Variablen pi, definieren wir die Funktion sinc, welche einen Wert x übergeben bekommt. Zuerst berechnen wir den in Zähler und Nenner benötigten Wert val = x * pi. Anschließend wird über den return Befehl das Ergebnis, der Division des sin(val) durch val zurückgegeben. Dies alles geschieht in einem try Block, da wir die Möglichkeit der null als Eingabewert nicht ausgeschlossen haben und nun für den ZeroDivisionError Fall eine Alternative bieten. Soweit zu der einfach zu lesenden Funktion. Nun richten wir unseren Blick auf die letzte Zeile, welche in eine Variable output eine Liste von Werten, der sinc Funktion, speichert

für alle Elemente x in der Liste input. Allein diese Zeile klingt bis auf ein, zwei Füllwörter schon nach einem vollständigen englischen Satz. Bedenkt man nun die Besonderheiten Pythons und die einfach zu lesende Funktion sinc als Ganzes lässt sich die Aussage von Oliphant, Python mit ausführbarem Pseudocode zu vergleichen nicht mehr bestreiten. Allerdings muss erwähnt werden, dass die Ausführung von Python Code im Vergleich zu anderen höheren Programmiersprachen sehr langsam ist.

2.3 Objektorientierte Programmierung in Python

In der objektorientierten Programmierung werden verschiedene Klassen definiert, welche Attribute und Methoden von Objekten definieren. Die Sichtbarkeit dieser Attribute und Methoden sorgt automatisch für eine gewisse Zugriffskontrolle innerhalb eines Programmes und fördert zusätzlich die bessere Lesbarkeit und Wiederverwendung von geschriebenem Code. Diese spezielle Art der Programmierung lässt sich auch in Python umsetzen. Im Folgenden ist die Klasse robot definiert:

```
class Robot:
    def __init__(self, name, constructionyear):
        self.__name = name
        self.__constructionyear = constructionyear
        print(f"{self.__name} was created!")
    def __del__(self):
        print("Robot was destroyed")
    def SayHello(self):
        print("Hello, my name is " + self.__name)
```

Die Klasse robot enthält einen Konstruktor, welcher mit einem Namen und einem Konstruktionsjahr ein Objekt der Klasse robot erzeugt, dies wird durch x = Robot("Marvin", 1979) aufgerufen. Anschließend erscheint auf der Konsole die Meldung: Marvin was created!. Ruft man nun an dem Objekt mittels x.SayHello() eine Methode auf, erscheint auf der Konsole: Hello, my name is Marvin. Abschließend wird mit dem Destruktor der Speicherplatz des Objektes wieder freigegeben: del x, dadurch erscheint die abschließende Meldung: Robot was destroyed.

Dieses kleine Beispiel zeigt nur sehr abstrahiert den Umfang und ei Möglichkeiten der objektorientierten Programmierung. Da sich viele moderne Programme auf diese Programmierart stützen, ist es umso wichtiger, dass die aufsteigende Programmiersprache Python die Strukturen für dieses Konzept bereitstellt und effektiv unterstützt.

3 HPC

Ein moderner Zweig der Informatik ist das sogenannte Hochleistungsrechnen. Dieses Teilgebiet beschäftigt sich mit der Verarbeitung und Berechnung großer Datenmengen, wie sie zum Beispiel in der Klimaforschung anfallen. Der große Unterschied zur klassischen Programmierung liegt darin, dass ein Programm nicht mehr sequenziell alle Berechnungen hintereinander ausführt, sondern das der Hauptthread des Programmes das gesamte Problem in einzelne, kleinere Teilprobleme aufteilt und diese von verschiedenen Nebenthreads gleichzeitig berechnen lässt, bevor die berechneten Ergebnisse anschließend vom Hauptthread wieder zusammengeführt werden. Die Ausführung solcher parallelen Programme wird in der Regel auf großen Computerclustern ausgeführt, ist allerdings so effektiv, dass auch schon moderne PC Prozessoren mit mehreren Kernen ausgestattet sind und diverse allgemein bekannte Software parallele Programmierung unterstützen und nutzen, sofern es auf dem Endgerät möglich ist. Allerdings arbeiten Computercluster nicht nur mit einzelnen Prozessoren, die mehrere Kerne haben, die sich ein und denselben Speicher teilen, sondern es ist eine Vielzahl von Prozessoren involviert, die alle ihren eigenen Speicher haben. Daher kommt es häufig vor, dass ein Nebenthread Informationen benötigt, auf die er keinen Zugriff hat. Die einzelnen Threads müssen daher miteinander kommunizieren, was in der Regel durch das Senden und Empfangen der notwendigen Daten umgesetzt wird.

Die parallele Programmierung für das Hochleistungsrechnen wird normalerweise in den Vorherrschenden Programmiersprachen Fortran und C geschrieben. Allerdings lässt sie sich auch in Python realisieren. Um dies zu zeigen betrachten wir als erstes ein kleines Programm, welches den Rang eines Threads auf die Konsole schreibt, wobei der nullte Rang immer dem Hauptthread entspricht:

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print(f"I am rank {rank}")
```

Als erstes importieren wir aus dem Modul mpi4py die Klasse MPI. Anschließend erzeugen wir mit MPI.COMM_WORLD eine eigene Welt und speichern diese in der Variable comm. Jeder in dieser Welt enthaltene Thread speichert nun seinen Rang mittels comm.Get_rank() in der Variable rank. Schließlich schreibt jeder Thread seinen Rang mit der print Funktion auf die Konsole. Es entsteht also die Zeile I am Thread X für jeden Thread einmal auf der Konsole.

Nach diesem kurzen Programm zum Einstieg in die parallele Programmierung wollen wir uns nun anschauen wie die Kommunikation zwischen den verschiedenen Threads realisiert ist:

```
from mpi4py import MPI
```

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
    data = {"a": 7, "b": 3.14}
    comm.send(data, dest=1, tag=11)
elif rank == 1:
    data = comm.recv(source=0, tag=11)
```

Die ersten drei Zeilen entsprechen wieder unserem ersten Programm. Statt nun den Rang auf die Konsole zu schreiben nehmen wir eine Fallunterscheidung vor. Wenn der ausführende Thread der nullte Thread ist, speichern wir eine Wörterbuch in die Variable data und senden dieses anschließend mit der comm.send Funktion and den ersten thread, der als dest and die Funktion übergeben wird. Außerdem übermitteln wir noch einen tag, welcher die Nachricht eindeutig markiert. Ist der ausführende Thread aber der erste Thread, so erstellen wir kein Wörterbuch, sondern empfangen, mit der an der comm aufgerufenen Funktion recv, vom Thread null eine Nachricht mit der Markierung 11 und speichern die übermittelten Daten in der Variable data. [Enk]

Wir sind also in der Lage mit Python parallele Programmierung, wie sie für das Hochleistungsrechnen zwingend erforderlich ist umsetzen. Dabei lässt sich der Code genauso leicht lesen wie der Python Code den wir bisher gesehen haben. Allerdings ist auch die Umsetzung der parallelen Programmierung in Python sehr langsam in seiner Ausführungszeit.

4 Numpy

Bisher haben wir gesehen, dass die Programmiersprache Python sehr umfangreich in seinen Anwendungsmöglichkeiten ist und der Quellcode sehr leicht verständlich ist. Auch wurde erwähnt, dass dieser Quellcode sehr langsam ausgeführt wird. Jedoch gibt es für Python das Numpy Modul. Dieses Modul ist explizit dafür entwickelt worden wissenschaftliche Berechnungen in Python zu optimieren. Dies geschieht indem n-dimensionale Arrays effektiv miteinander verrechnet werden. Dazu gibt es universelle Funktionen, welche die Dimensionen ihrer Eingabearrays so interpretieren, wie sie am besten miteinander verrechnet werden ohne dabei einzelne Teile zu kopieren und extra Zwischenspeicher zu allokieren und wieder freizugeben, also Inplace zu arbeiten. Numpy beinhaltet dabei unterschiedlichste Klassen zur Arrayerzeugung und -manipulation. Diese ermöglichen schnelle Erzeugung aus Eingabewerten und im Vergleich zu den in Python integrierten Listen verbesserte Verarbeitung dieser erzeugten Arrays. Insbesondere gibt es auch Klassen zu den im Hochleistungsrechnen vielfach vertretenen Fourier Transformationen verschiedener Dimensionalität. Diese schnelle Berechnung von Numpy-Arrays lässt sich außerdem in die im Hochleistungsrechnen vorherrschenden Programmiersprachen Fortran und C integrieren. Achtet man nun ein bisschen auf die Art seiner Implementation und nutzt die effektiven universellen Funktionen sinnvoll indem man beispielsweise die Berechnung der Matrix a durch die b, c und d:
a = (b + 4) * c * d
ein wenig umschreibt, wird aus der unnötigen dreifachen Speicherallokation für die Matrix b + 4, die intern gespeicherte Matrix b + 4 multipliziert mit der Matrix c und diese nun erneut intern gespeicherte Matrix multipliziert mit der Matrix d eine Inplace Berechnung. Einzige Voraussetzung ist, dass die Matrizen Numpy-Arrays sind und der Code nicht mehr ganz so intuitiv aussieht:
a = b + 4
np.multiply(a, c, a)
np.multiply(a, d, a)
Was hier geschieht ist nun die nötige Allokation für die neue Variable a, die gleich mit dem Ergebnis der Berechnung Matrix b + 4 erfolgt. Anschließend findet statt einer erneuten Allokation die Berechnung und Inplace-Speicherung der Multiplikation a * c und anschließend a * d über die universelle Funktion multiply aus dem Modul Numpy statt. [Oli07]

Nach einer kleinen Umgewöhnung an die Aufrufe merkt man schnell, dass die Verwendung des Numpy Moduls sehr viel schneller ist als Python selbst. So lässt sich sie Multiplikation von zwei Matrizen der Dimension 200 mit Numpy in 0.01 sec berechnen, wohingegen eine reine Python Implementation 5.30 sec benötigt. Numpy schlägt mit dieser Zeit sogar eine einfache Implementierung in C, mit 0.09 sec aufwand. [Enk]

5 C und Python

Die bisherigen Kapitel haben sich mit den Stärken und Schwächen der Programmiersprache Python beschäftigt. Um ihren Nutzen für das Hochleistungsrechnen nun einordnen zu können, vergleichen wir sie mit der Programmiersprache C, ohne jedoch so detailliert auf C einzugehen. Die Sprache C als hohe Programmiersprache, mit den dazugehörigen Konstrukten, ist durch seine hardwarenahe Programmierung und schnelle Ausführungszeit bei geringem Ressourcenbedarf wohl eine der mächtigsten Programmiersprachen, die es gibt. Allerdings kann C Code sehr schnell unleserlich werden und der direkte Hardwarezugriff kann zu schwerwiegenden Problemen führen. Alles in allem ist die uneingeschränkte Freiheit eines C Programmierers solange nützlich, bis sie zu Problemen führt. [Wol09]

Im Vergleich zu Python, bietet C also sehr effektiven Code, der allerdings schnell unübersichtlich wird und somit das genaue Gegenteil darstellt. Doch wie wir wissen ist Python sehr flexibel und kann mit anderen Programmiersprachen interagieren. Es ist möglich Python Code in ein C Programm zu integrieren. So können wir den Großteil eines Programmes in sehr lesbarem und übersichtlichen Python Code schreiben und die rechenintensiven Teile mit dem schnellen Numpy Modul für wissenschaftliche Berechnungen in ein C Programm auslagern. Das folgende Beispiel zeigt den benötigten C Code, welcher sowohl aus einer Funktion my_C_func, die für die eigentliche Arbeit zuständig ist, als auch aus den Aufrufen, zur Initialisierung der Python Funktion, besteht:

```
#include <Python.h>
#define NO_IMPORT_ARRAY
#include <numpy/arrayobject.h>
PyObject* my_C_func(PyObject *self, PyObject *args) {
PyArrayObject* a;
if (!PyArg_ParseTuple(args, "O", &a))
return NULL;
int size = PyArray_SIZE(a);
double *data = (double *) a > data;
for (int i = 0; i < size; i++) {
/* Process data */ }
Py_RETURN_NONE;
}

static PyMethodDef functions[] =
{
{"myfunc", my_C_func, METH_VARARGS, 0},
{0, 0, 0, 0}
};
```

```
PyMODINIT_FUNC initmyext(void)
{
(void) Py_InitModule("myext", functions);
}
```

Da die Sprache C im Gegensatz zu Python keine interpretierte Sprache ist, muss der C Code zuerst mit dem folgenden Befehl kompiliert werden:

```
gcc -shared -o myext.so -I/usr/include/python2.6 -fPIC myext.c
```

Die mit der -shared Option erzeugte myext.c Datei kann nun in ein Python Programm importiert und dort ihre enthaltenden Funktionen benutzt werden:

```
import myext
a = np.array(...)
myext.myfunc(a) [Enk]
```

Wir sehen auch hier wieder, wie kurz und übersichtlich der Python Code zum Aufruf der Methoden im C Code ist. Wohingegen der C Code selbst unter der Berücksichtigung, dass initmyext und functions nur zur Initialisierung dienen, wirkt die eigentliche C Funktion my_C_func sehr unübersichtlich. In Anbetracht der Tatsache, dass sie eigentlich nur grundlegende Parameter bereitstellt und keinen eigentliche Aufgabe implementiert ist.

6 Fazit

Nach der ausführlichen Behandlung der Programmiersprache Python, welche sich insbesondere durch den Verzicht auf Klammern zur Eingrenzung von Blöcken und die dynamische Typisierung von anderen Programmiersprachen unterscheidet und sich durch weitere Besonderheiten mehr wie ausführbarer Pseudocode liest, als eine klassische Programmiersprache konnten wir die Aussage von Oliphant bekräftigen. Die sehr simple, leicht zu verstehende und lesende Programmiersprache ist nicht nur für Neulinge ein leichter Einstieg in die Informatik, sondern zeigt auch erfahrenen Programmierern wie einfach ihr Handwerk mit den richtigen Bausteinen sein kann. Dennoch hat Python bezüglich der Ausführungsgeschwindigkeit als interpretierte Programmiersprache gegenüber kompilierten durchaus seine Nachteile. Für den allgemeinen Anwender lässt sich dieser Mangel insbesondere bei anspruchsvollen Berechnungen durch die Nutzung des wissenschaftlichen Modules Numpy beheben. Da sich dies nur auf das Numpy Modul beschränkt, lässt die allgemein und in Bezug auf die parallele Programmierung langsame Ausführung es nicht zu Python als Alternative im Hochleistungsrechnen gegenüber Fortran oder C zu betrachten. Dennoch lässt sich der größte Vorteil Pythons, der definitiv in seiner Einfachheit liegt und die Möglichkeit Python mit Fortran oder C zu kombinieren im Hochleistungsrechnen nutzen. Somit können viele Programme durch die Kombination der einfachen Sprache Python mit der Mächtigkeit von C zu effektiveren Programmen im Bereich des Hochleistungsrechnen führen. Doch nicht nur für dieses Teilgebiet der Informatik ist Python eine nützliche Ergänzung, denn auch über die Grenzen des Teilgebietes hinausschauend ist Python beispielsweise im Machine Learning eine wichtige Komponente.

Die Kernfrage dieser Ausarbeitung, ob die Aussage Oliphants, dass Python Code sich als ausführerbarer Pseudocode beschreiben lässt, zutrifft und inwiefern sie die Programmiersprache Python für Programmierer, insbesondere im Bereich des Hochleistungsrechnen, attraktiv macht, kann nach der näheren Betrachtung durchaus als Kernaussage der Ausarbeitung angesehen werden.

Literaturverzeichnis

[Enk] Jussi Enkovaara. Python in high performance computing.

[Oli07] Travis Oliphant. Python for scientific computing. 9:10–20, 06 2007.

[Ras14] Sebastian Raschka. The key differences between python 2.7.x and python 3.x with examples, June 2014.

[Wik18] Wikipedia. Python (programmiersprache), May 2018.

[Wol09] Jürgen Wolf. C von a bis z, 2009.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum	Unterschrift

Veröffentlichung

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek des Fachbereichs Informatik eingestellt wird.

Ort, Datum	Unterschrift