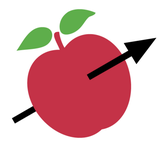
Supersonic Algorithms

Anton Rodenwald (18), Schillerschule Hannover



Ideenfindung und Forschungsfrage

- Sortieralgorithmen für Zahlen im Informatikunterricht kennengelernt
- Ich fragte mich, wie man Zahlen am schnellsten sortieren kann
- Sortieralgorithmen sind bereits bekannt und erforscht, deswegen Fokus auf Implementation bekannter Algorithmen
- Forschungsfrage: Wie lassen sich Programme (z. B. Sortieralgorithmen) durch geschickte Implementation in ihrer Ausführung beschleunigen?
- Kein Vergleich von Sortieralgorithmen, fast immer Quicksort verwendet
- Fokus auf die Programmiersprachen Python, C++ und C
- In Internetquellen wurde C++ um ein vielfaches schneller als Python beschrieben was auch meiner Vermutung entsprach

Schwierigkeiten

- Meine größte Schwierigkeit war, dass ich noch nicht viel Erfahrung mit der Optimierung und dem wissenschaftlichen Arbeiten hatte
- Erstellung der Dokumentation mit LaTeX teils frustrierend
- Mit einigen Bibliotheken und Sprachen Schwierigkeiten
- Mir war das testen mit 10 Millionen Zahlen in Go nicht möglich, da dort Rekursion nicht so gut funktioniert

Beantwortung der Forschungsfrage und Ausblick

- Die besten Möglichkeiten zur Optimierung in Python sind NumPy und Numba; Cython schlechter, CTypes komplex
- In C++ bietet AVX2 interessante Möglichkeiten der Optimierung
- Python kann ähnlich schnell wie C++ sein kann
- Insgesamt profitiert Python vorallem von schnellen C Bibliotheken
- Python ist nur dann schnell, wenn kein Python Code, sondern C Funktionen ausgeführt werden
- Weiter erforschen könnte man den Aufbau von Interpretern und Compilern (z. B. am Beispiel von Python, C++)
- Auch denkbar ist der vertiefte Vergleich von Quicksort und Radixsort zur Sortierung verschiedener Daten

Material, Vorgehen und Methode

- Ich nutze meinen Linux PC und Linux Laptop zur Implementation
- Ich verwende die IDE Visual Studio Code und die Compiler und Interpreter für meine Programmiersprachen
- Ich implementierte anschließend die Quicksort auf verschiedene Arten und erhielt so viele Variationen zum testen
- Die Tests führte ich auf meinem Desktop PC durch (CPU ist der Ryzen 7 2700, 16 GB Arbeitsspeicher), der immer gleich ausgelastet war, um Vergleichbarkeit zu gewährleisten
- Die Zeit stoppte ich mit den Zeitfunktionen der Sprachen

Ergebnissdiskussion

- Ein Ansatz zur Erklärung der Ergebnisse ist die Unterscheidung von interpretierten und kompilierten Programmiersprachen
- Bei interpretierten Sprachen führt ein spezielles Programm, der Interpreter, ein in Textform vorliegendes Programm aus
- Bei kompilierten Sprachen muss vor dem Ausführen der Quellcode von einem Compiler umgewandelt werden in ein ausführbares Programm
- Dieser Prozess kostet Zeit, aber erlaubt viele verschiedene Optimierungen
- Beim Interpreter ist dies bei Beginn der Ausführung alles noch nicht geschehen
- Python ist deswegen langsamer, weil während der Ausführung der Programmcode der Quelldatei noch zeitintensiv analysiert werden muss, bevor dieser ausgeführt werden kann
- Bei kompilierten Sprachen wie C++ ist dies bereits geschehen und der Programmcode wurde vom Compiler optimiert, weswegen diese Programme dann schneller sind
- Python ist nur dann schnell, wenn der Interpreter vorher kompilierte Programmteile nutzen kann, also wenn kaum Python Code interpretiert werden muss
- Solche vorher kompilierten Programmteile stellt z. B. NumPy bereit
- Die Bibliothek Numba macht Python deshalb schneller, weil sie die Kompilierung von Python ermöglicht
- Zu Programmbeginn kompiliert Numba einige Teile des Python Programms, was etwas Zeit kostet
- Werden diese Teile besonders häufig benutzt, kann sich das aber wieder rechnen und das Programm schneller machen
- Diesen Vorgang nennt man JIT-Kompilierung, also “Just-in-Time” Kompilierung, weil er eben beim Ausführen geschieht