# Aufgabenblatt 8
# Abhängigkeit der Performanz und Komplexität von tieferen Schichten

Cao, Thi Huyen(Lilli)-Kochinky,Dennis

November 16, 2016

# Contents

# 1 Hardware-gestützte Beschleunigungstechnicken

Hardware gestützte Beschleunigungstechniken:

CUDA ist eine ist eine Programmier-Technik von Nvidia mit der Programmteile durch den Grafikprozessor (GPU) bearbeitet werden können. Dies lohnt sich allerdings nur bei sehr hohem Rechenaufwand, da durch die Anbindung über PCIe der Datendurchsatz geringer als bei einer CPU ist. Ein weiterer Nachteil ist, dass GPUs sehr spezifische Aufgaben haben und deshalb mit selten genutzten Datentypen arbeiten und Standard Datentypen aufwändig emuliert werden müssen. Einen ähnlichen Ansatz hat OpenCL ist aber im Vergleich zu CUDA nicht auf Nvidia Hardware spezialisiert.

SIMD: Single Instruction Multiple Data ist eine Rechnerarchitektur bei der gleichartige Rechenoperationen auf mehrere Eingangsdatenströme angewendet werden können. Dies wird vor allem bei Bild-, Video- und Tondaten eingesetzt, weil die zu verarbeitenden Daten hoch parallelisierbar sind z.B. bei der Bildbearbeitung ist sind die Operationen für die einzelnen Bildpunkt gleich. In modernen Prozessorarchitekturen wie dem x86 gibt es dafür zusätzliche Register/Befehlssätze.

OpenMP ist eine Programmierschnittstelle zur Shared-Memory-Programmierung auf Multiprozessor Computern. Dabei werden Programme auf der Ebene von Schleifen parallelisiert, welche auf verschiedene Threads verteilt werden.

Quellen:
http://www.itwissen.info/definition/lexikon/single-instruction-multiple-data-SIMD-SIMD-Rechnerarchitektur.html
https://de.wikipedia.org/wiki/CUDA
https://de.wikipedia.org/wiki/OpenCL
https://de.wikipedia.org/wiki/X86-Prozessor
https://de.wikipedia.org/wiki/OpenMP

# 2 Komplexitätuntersuchen unter verschiedenen Implementierungen und Optimierungen

## 2.1 Java Implementierung

(1) Laufzeit von Bubblesort und Insertionsort( in ms) in Java-Implementierung (Aufgabenblatt 4)
Problemgrösse N= 10000

| | Min | Max | Durchschnittsdauer |
|---|---|---|---|
| Bubblesort | 1101ms | 1822ms | 1228,18ms |
| Insertionsort | 61ms | 300ms | 96,79ms |

## 2.2 C Implementierung

(2). Laufzeit von Bubblesort in C-Implementierung

```
gcc -Wall -o bubblesort_c bubblesort_c.c
```

Laufzeit: 470ms, 480ms, 480ms,480ms, 470ms...
(3) Laufzeit von Bubblesort in C-Implementierung mit verschiedenen Optimierungen
(a)

```
gcc -Wall -O3 -o bubblesort_c bubblesort_c.c
```

Laufzeit: 470ms, 470ms, 470ms, 470ms, 490ms...
(b) Bubblesort in C-Implementierung ist viel schneller als in Java-Implementierung
(4)

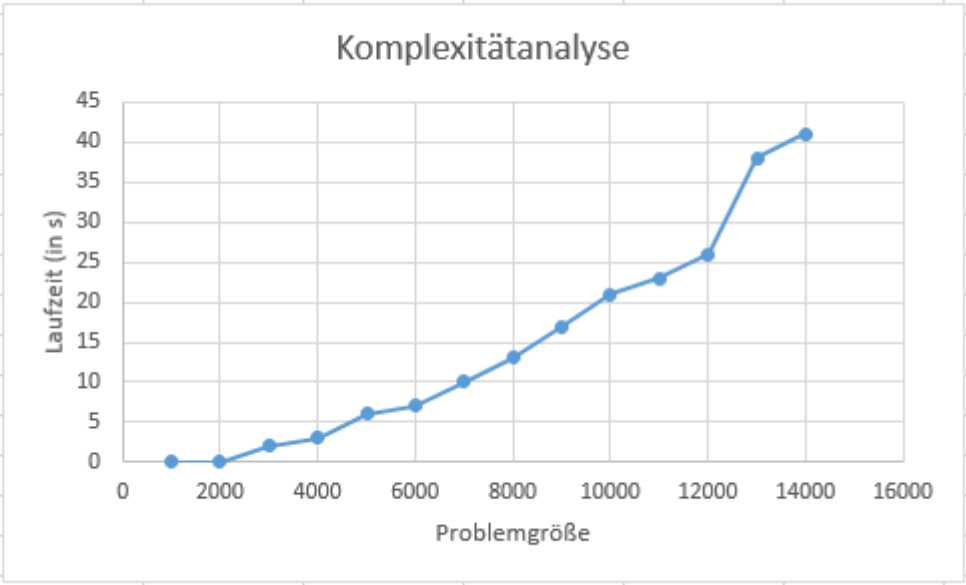
```
#pragma omp parallel for

gcc -Wall -O3 -fopenmp -o bubblesort_c bubblesort_c.c
```

Laufzeit: 190ms, 230ms, 230ms, 230ms, 220ms...
(5)Pragma Direktive direkt vor For-Schleife in dem Methode bubblesort
Laufzeit:1600ms, 1590ms, 1580ms ...

```
C:\Users\cao\Desktop>gcc -Wall -o3 -fopenmp -o prog  bubblesort_c.c

C:\Users\cao\Desktop>prog
Ergebnis: 1, 2, 3, ...9998, 9998, 9998, 9998, 9998, 9998
Verstrichene Zeit in C:  158 s (1.580000 s je Lauf)

C:\Users\cao\Desktop>prog
Ergebnis: 1, 2, 3, ...9998, 9998, 9998, 9998, 9999, 10000
Verstrichene Zeit in C:  160 s (1.600000 s je Lauf)

C:\Users\cao\Desktop>prog
Ergebnis: 1, 2, 3, ...9997, 9997, 9997, 9998, 10000, 10000
Verstrichene Zeit in C:  159 s (1.590000 s je Lauf)
```

Es ist leicht zu sehen dass manche Elementen mehr Mals aus der Konsole ausgegeben wurden. OpenMP(Open Multi-Processing) ist ein API, die multiplatform shared memory multiprocessing Programmieren in C/C++ usw. unterstützt.Das Pragma omp parallel wird verwendet, um mehrere Threads zu erzeugen, die die Arbeit in dem geschlossenen Block erledigen.
Flag -fopenmp wird verwendet,um die compiler (GCC) zu sagen, dass es um ein multithread Programm geht. Da mehrere Threads gleichzeitig auf gleichem Ressource eingreifen, könnten zu Problemen(Race Condition, Synchronisieren...) führen
(6) Komplexitätsklasse der schnellsten Implementierung

| N | T(N) |
|---|---|
| 1000 | 0 |
| 2000 | 0 |
| 3000 | 2 |
| 4000 | 3 |
| 5000 | 6 |
| 6000 | 7 |
| 7000 | 10 |
| 8000 | 13 |
| 9000 | 17 |
| 10000 | 21 |
| 11000 | 23 |
| 12000 | 26 |
| 13000 | 38 |
| 14000 | 41 |




Die Komplexitätsklasse ist nicht mehr $N^2$, sondern eher N