

Untersuchung der Skalierbarkeit von parallelem Sortieren auf einem Multicore-Prozessor

Bachelorarbeit

Studiengang: Informatik

Bearbeiter: Leon Zoerner

Inhaltsverzeichnis

1 Einleitung	3
1.1 Motivation	3
1.2 Zielsetzung und Forschungsfrage	3
1.3 Threading anhand des einfachen Beispiels Inkrement-Array erklärt	3
2 Theoretische Grundlagen	5
2.1 Sortieralgorithmen: Quicksort und Mergesort	5
2.2 Grundlagen der Parallelisierung	5
2.3 Thread-Modelle, Overheads und Skalierungsgrenzen	5
3 Methodik und Versuchsaufbau	6
3.1 Messumgebung und Hardware	6
3.2 Implementierungsvarianten	6
3.3 Messmethodik	6
4 Ergebnisse und Analyse	7
4.1 Grundlegende Laufzeiten abhängig von der Arraygröße	7
4.1.1 Messziel	7
4.1.2 Erwartung	7
4.1.3 Diagramm	7
4.1.4 Analyse und Interpretation	7
4.2 Einfluss des Listentyps	8
4.2.1 Messziel	8
4.2.2 Erwartung	8
4.2.3 Diagramm	8
4.2.4 Analyse und Interpretation	8
4.3 Einfluss der Arraygröße im Detail	9
4.3.1 Messziel	9
4.3.2 Erwartung	9
4.3.3 Diagramm	9
4.3.4 Analyse und Interpretation	9
4.4 Tiefenbasierte Thread-Erzeugung	10
4.4.1 Messziel	10
4.4.2 Erwartung	10
4.4.3 Diagramm	10
4.4.4 Analyse und Interpretation	10
4.5 Workerthreads	11
4.5.1 Messziel	11
4.5.2 Erwartung	11
4.5.3 Diagramm	11
4.5.4 Analyse und Interpretation	11
4.6 Vergleich der Threading-Methoden	12
4.6.1 Messziel	12
4.6.2 Erwartung	12
4.6.3 Diagramm	12
4.6.4 Analyse und Interpretation	12
4.7 Einfluss des Datentyps der Liste	13

4.7.1	Messziel	13
4.7.2	Erwartung	13
4.7.3	Diagramm	13
4.7.4	Analyse und Interpretation	13
5	Diskussion und Fazit	14
5.1	Interpretation aller Ergebnisse	14
5.2	Beantwortung der Forschungsfrage	14
5.3	Zusammenfassung	14
6	Anhang	15
6.1	Code	15
6.2	Hardware-Spezifikationen	15

1 Einleitung

1.1 Motivation

Ziel dieser Arbeit ist es, die Grenzen von Threads und Parallelisierung aufzuzeigen. Dabei soll insbesondere untersucht werden, wie groß der Overhead durch Threads ist und welchen Performanceunterschied es macht, bereits initialisierte Workerthreads zu verwenden, im Vergleich zur Erstellung neuer Threads. Da sich für diese Untersuchungen ein geeigneter, leicht verständlicher und programmierbarer Anwendungsfall anbietet, habe ich mich für Sortieralgorithmen entschieden, die sich zudem sehr gut parallelisieren lassen.

1.2 Zielsetzung und Forschungsfrage

Ziel dieser Bachelorarbeit ist die systematische Analyse der Laufzeitentwicklung paralleler Sorterverfahren. Dabei soll untersucht werden, wie sich parallele Implementierungen von Quicksort und Mergesort im Vergleich zu ihren sequentiellen Varianten verhalten. Im Fokus stehen insbesondere folgende Punkte:

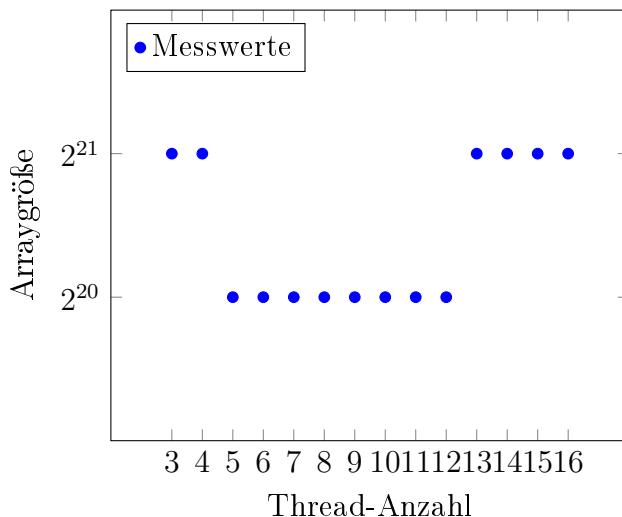
- der Einfluss verschiedener Threadingstrategien auf die Laufzeit,
- die Frage, ab welcher Eingangsgröße und bei welcher Anzahl von Threads ein messbarer Geschwindigkeitsvorteil entsteht,
- sowie die Identifikation von Thread-Management-Techniken, die für Sortieralgorithmen die besten Laufzeiten erzielen.

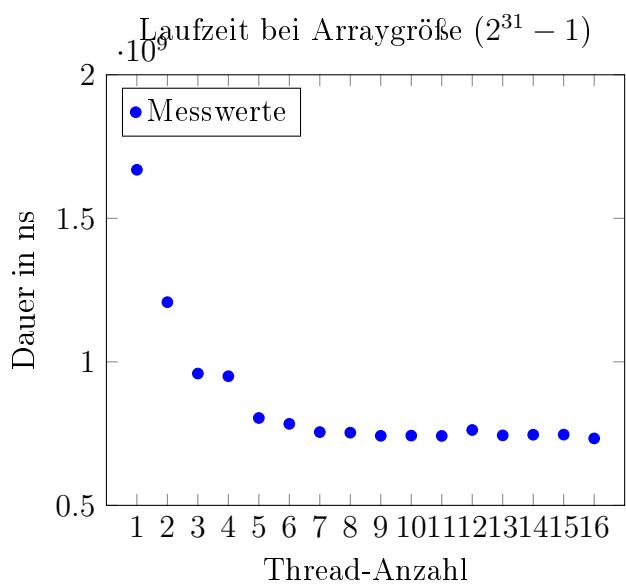
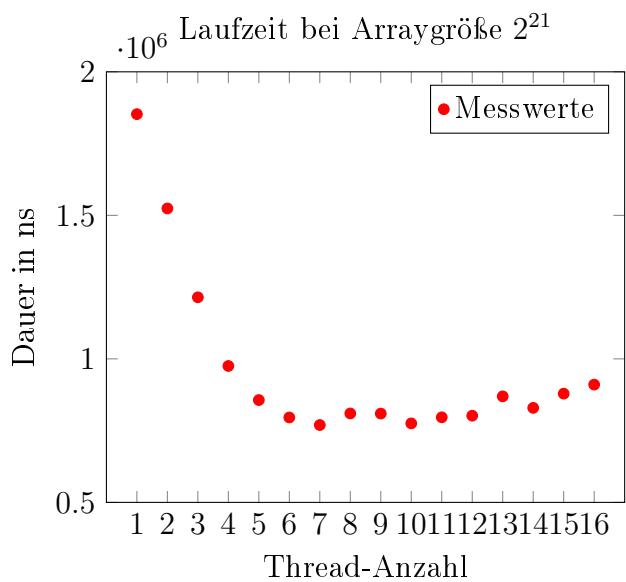
Aus diesen Aspekten ergibt sich die zentrale Forschungsfrage dieser Arbeit:

Unter welchen Bedingungen liefern parallele Sortieralgorithmen anhand von Quicksort und Mergesort einen signifikanten Laufzeitvorteil gegenüber der sequentiellen Ausführung, und welche Threadingstrategien führen dabei zur besten Laufzeit?

1.3 Threading anhand des einfachen Beispiele Inkrement-Array erklärt

Zeitpunkt der 50% Geschwindigkeitssteigerung (incAray)





2 Theoretische Grundlagen

- 2.1 Sortieralgorithmen: Quicksort und Mergesort**
- 2.2 Grundlagen der Parallelisierung**
- 2.3 Thread-Modelle, Overheads und Skalierungsgrenzen**

3 Methodik und Versuchsaufbau

3.1 Messumgebung und Hardware

3.2 Implementierungsvarianten

3.3 Messmethodik

4 Ergebnisse und Analyse

4.1 Grundlegende Laufzeiten abhängig von der Arraygröße

4.1.1 Messziel

4.1.2 Erwartung

4.1.3 Diagramm

4.1.4 Analyse und Interpretation

4.2 Einfluss des Listentyps

- 4.2.1 Messziel**
- 4.2.2 Erwartung**
- 4.2.3 Diagramm**
- 4.2.4 Analyse und Interpretation**

4.3 Einfluss der Arraygröße im Detail

4.3.1 Messziel

4.3.2 Erwartung

4.3.3 Diagramm

4.3.4 Analyse und Interpretation

4.4 Tiefenbasierte Thread-Erzeugung

- 4.4.1 Messziel**
- 4.4.2 Erwartung**
- 4.4.3 Diagramm**
- 4.4.4 Analyse und Interpretation**

4.5 Workerthreads

4.5.1 Messziel

4.5.2 Erwartung

4.5.3 Diagramm

4.5.4 Analyse und Interpretation

4.6 Vergleich der Threading-Methoden

4.6.1 Messziel

4.6.2 Erwartung

4.6.3 Diagramm

4.6.4 Analyse und Interpretation

4.7 Einfluss des Datentyps der Liste

4.7.1 Messziel

4.7.2 Erwartung

4.7.3 Diagramm

4.7.4 Analyse und Interpretation

5 Diskussion und Fazit

- 5.1 Interpretation aller Ergebnisse**
- 5.2 Beantwortung der Forschungsfrage**
- 5.3 Zusammenfassung**

6 Anhang

6.1 Code

Der gesamte Quellcode dieser Arbeit ist öffentlich unter folgendem Link verfügbar:
<https://github.com/Leon333M/Sortierverfahren>

6.2 Hardware-Spezifikationen

Zur besseren Einordnung der Leistungsfähigkeit der verwendeten Hardware befindet sich unter folgendem Link ein Benchmark:

<https://www.userbenchmark.com/UserRun/70984567>

Ich liste aber jetzt hier auch nochmal die relevanten Hardwarekomponenten auf.

CPU: AMD Ryzen 7 5800X, 8C/16T, 3.80-4.70GHz

CPU Kühler: be quiet! Dark Rock Pro 4)

RAM: G.Skill Aegis UDIMM 16GB Kit, DDR4-3200, CL16-18-18-38 (2 Kits, insgesamt 4x8 GB = 32 GB)

Betriebssystem: Windows 10 Version 22H2