Paralle Sortieralgorithmen

Christoph Niederseer, Michaela Mayr, Alexander Aichinger, Fabian Küppers

Inhalt

1. Was ist paralleles Programmieren
2. Bitoner Sortieralgorithmus
3. Quicksort
   a) sequenzielles Quicksort
   b) paralleles Quicksort
4. Radix Sort

Einführung– Übersicht

- Definitionen und Abgrenzung
- Komplexitätsklasse NC
- Amdahlsches Gesetz
- Gustafsons Gesetz

Einführung

- Sequentielle Programmierung
- Parallele Programmierung:
  - Threads werden nebenläufig ausgeführt
  - Nebenläufige Threads synchronisieren
  - Möglich, wenn gleiche Resultate durch Parallelisierung erzielt wird, wie sequentielle

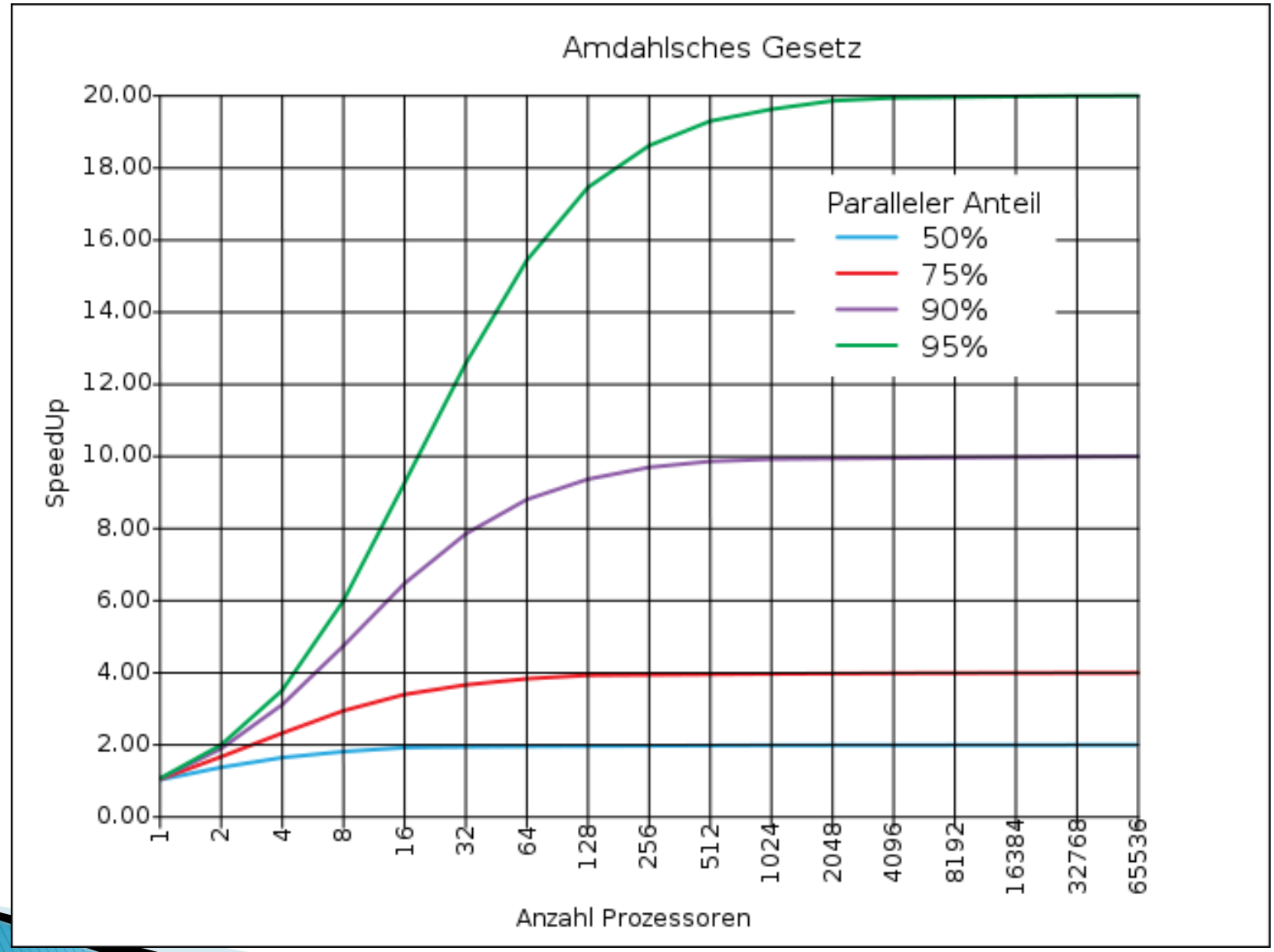
- Komplexität (Ermittlung von Zeit- und Speicherplatzaufwand) kann bei parallelen Algorithmen schwieriger nachvollziehbar sein.
- Eigene Komlexitätsklasse NC (Nick´s Class nach Nick Pippenger)
  - polylogarithmische Zeit ($O(\log^c n)$, c konstant)
  - polynomieller Aufwand (also $O(n^k)$, k konstant)

- Grenzen von parallel lösbaren Problemen
  - Gesetz von Gene Amdahl 1967
  - Gesetz von John Gustafson 1988

Amdahlsche Gesetz

- Geschwindigkeitzuwachs wird vor allem durch sequentiellen Anteil des Problems beschränkt. (z.B.: Prozess-Initialisierung, Speicherplatz-Allokation)
- Kritik:
  - Günstiger Fall: gesamte Problemgröße im Cache statt im langsameren Hauptspeicher
  - Je größer Problemmenge, desto weniger fällt Zeit für Initialisierung ins Gewicht

Amdahlsche Gesetz



Quelle: http://de.wikipedia.org/w/index.php?title=Datei:AmdahlsLaw_german.svg&filetimestamp=20090311122810

Gustafsons Gesetz

- Eine genügend große Problemgröße kann effizient parallelisiert werden.
- Unterschied:
  - paralleler Anteil wächst mit der Anzahl der Prozessoren -> sequentieller Anteil vernachlässigbar.
  - Geht Anzahl der Prozessoren gegen unendlich, so wächst SpeedUp linear mit Anzahl der Prozessoren.
- Kritik
  - Probleme mit bestimmter Größe
  - Echtzeit

Bitoner Sortieralgorithmus

- Einleitung
- Definition: Bitonische Reihe
- Batcher Algorithmus
  - Ablauf
  - Umsetzung als paralleler Algorithmus
  - Merge
  - Darstellung für Netzwerk Ordnung 8
  - Darstellung für vereinfachtes Netzwerk Ordnung 8
  - Konkretes Beispiel Ordnung 8

Einleitung

- Probleme große Datenmengen zu sortieren
- Grenzen bei Computern mit einem Prozessor
- Folge: mehrere Rechner verwenden
- Basis für parallele Sortieralgorithmen ist bitonisches Sortieren

Definition: bitonische Folge

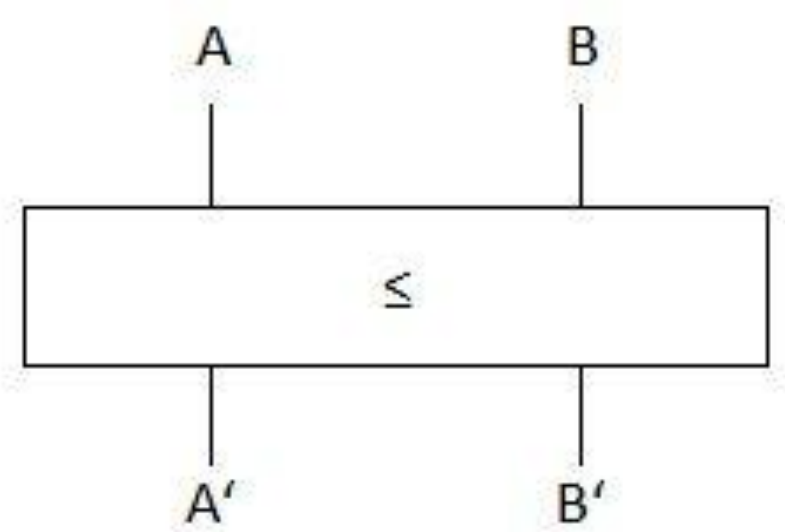
Eine Folge $a_0,\ldots,a_n$ von ganzen Zahlen heißt bitonisch, wenn der erste Teil der Folge aufsteigend und der zweite Teil absteigend sortiert ist, oder wenn man die Folgenglieder so durchschieben kann, dass die Bedingung erfüllt ist.

Ablauf des bitonischen Sortierens

1. Input: unsortierte Folge
2. In 2 Hälften teilen
3. a. Hälfte aufsteigend
   b. Hälfte absteigend sortieren
4. Verbinden durch merge
5. Sortieren mit merge

Animation

Umsetzung als paralleler Algorithmus



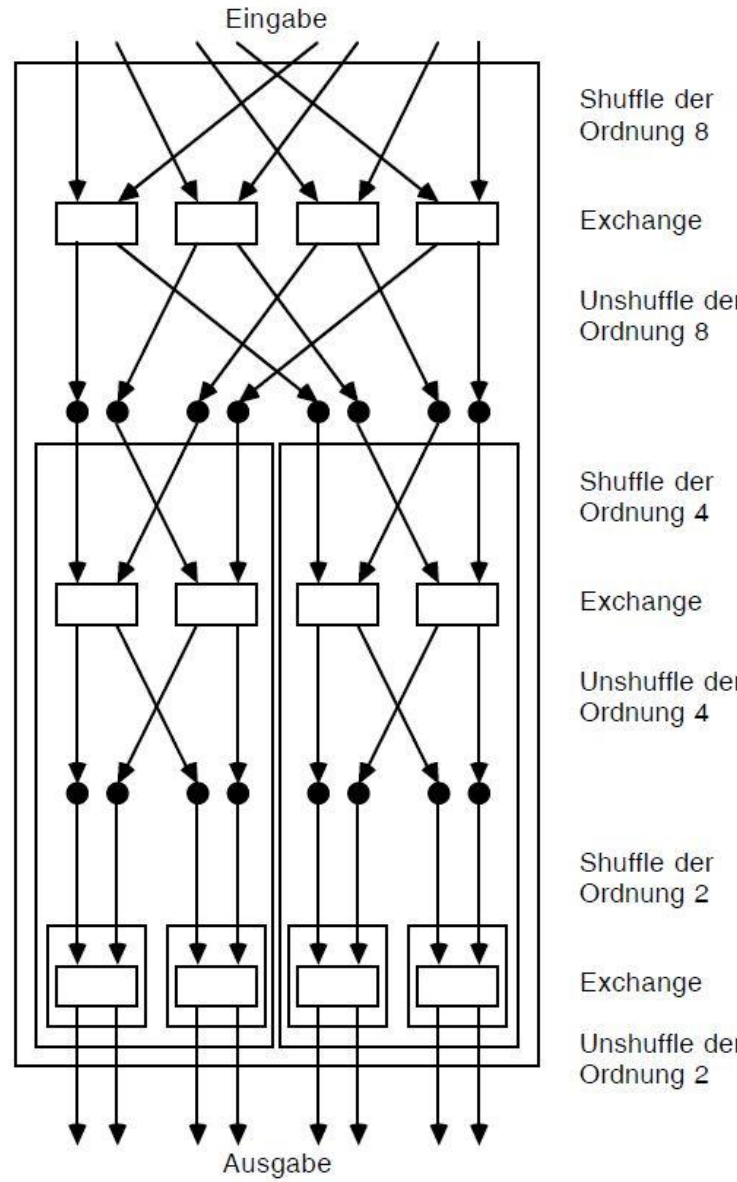
merge

- Shuffle
- Exchange
- Unshuffle

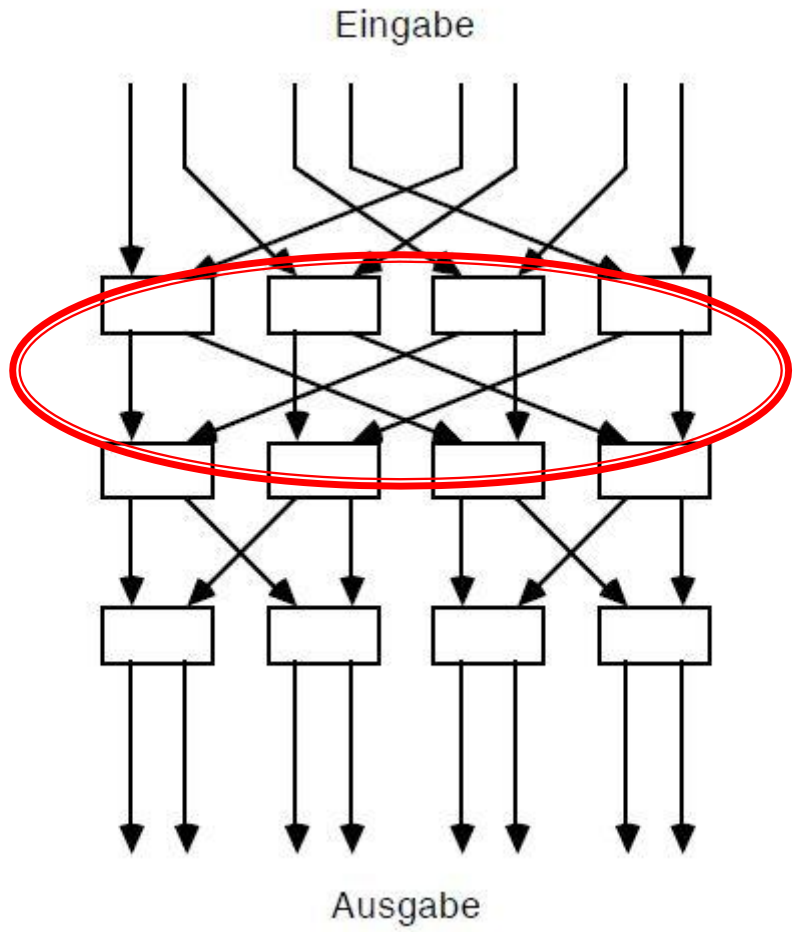
Oder:

- $a_1,\ldots,a_i$ (aufsteigend) und $a_{i+1},\ldots,a_n$ (absteigend) mit $i=n/2$
  merge:
  $\min(a_1,a_{i+1}),\ldots,\min(a_i,a_n)$,
  $\max(a_1,a_{i+1}),\ldots,\max(a_i,a_n)$.

Netzwerk der Ordnung 8



Vereinfachtes Netzwerk Ord8

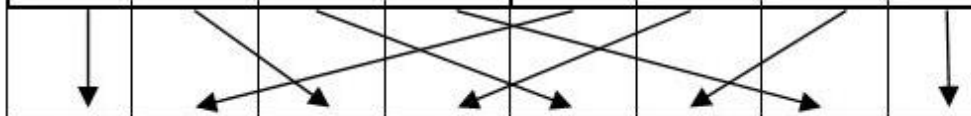
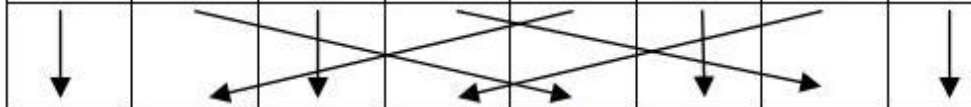
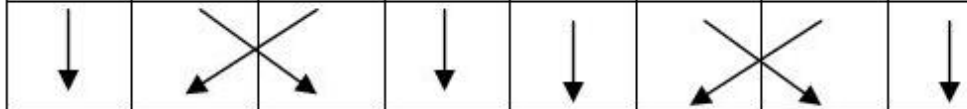


Ein konkretes Beispiel

- 1.Phase (sortieren auf bitonische Folge)

B	X	D	Y	M	E	F	A	Input: unsortierte Folge
<		>		<		>		„Erzeugen“ von biton. Folgen der Ord. 2
B	X	Y	D	E	M	F	A	
↓	✕	↓		↓	✕	↓		Schuffle Ordnung 4
B	Y	X	D	E	F	M	A	
<		<		>		>		exchange
B	Y	D	X	F	E	M	A	
↓	✕	↓		↓	✕	↓		Unshuffle Ordnung 4
B	D	Y	X	F	M	E	A	
<		<		>		>		exchange
B	D	X	Y	M	F	E	A	Bitonische Folge erzeugt

- 2.Phase (sortieren der bitonischen Folge)

B	D	X	Y	M	F	E	A	Bitonische Folge
								shuffle Ordnung 8
B	M	D	F	X	E	Y	A	
<		<		<		<		exchange
B	M	D	F	E	X	A	Y	
								unshuffle Ordnung 8 (für verkürzte Berechnung)
B	E	D	A	M	X	F	Y	
<		<		<		<		exchange
B	E	A	D	M	X	F	Y	
								Shuffle Ordnung 4
B	A	E	D	M	F	X	Y	
<		<		<		<		exchange
A	B	D	E	F	M	X	Y	Output: sortierte Folge

Komplexität

- Rekursionsebenen + Vergleichsoperationen
- $\Theta(\log^2 n)$
- Schnellster sequentieller Algorithmus: $\Theta(n \log n)$

Sequenzieller Quicksort

- Rückblick:
  - 1962, England, Tony Hoare
  - „Quick“ – Schnell, „sort“ – sortieren
  - Rekursiver, nicht stabiler Algorithmus
  - „Divide and Conquer“ – Teile und Herrsche

Theoretisches:

- Wahl eines Pivotelements
- Alle Elemente < Pivot: linke Teilliste
- Alle Elemente > Pivot: rechte Teilliste

- Selbstaufrufe: Rekursion
- Änderung der Position der zueinander gleichen Elemente: nicht stabiler Algorithmus

Paralleler Quicksort

- 1. Phase: Die Partitionierung
  - Alle Elemente in Blöcke
  - Größe der Blöcke: pro Kern 2 Blöcke gleichzeitig im First Level Cache der CPU
  - PID = Process Identifier
  - Aufgabe: Blöcke neutralisieren
- 2. Phase: Sequentielle Partitionierung
  - Blöcke nochmal richtig positionieren
  - Array in 2 Teilarrays teilen

Paralleler Quicksort

- 3. Phase: Prozess Partitionierung
  - Prozesse werden in 2 Gruppen geteilt
  - Pro Gruppe 1 Teilarray
  - Neubeginn mit der 1. Phase
- 4. Phase: Sequenzielles Sortieren mit Hilfe
  - Aufteilen der Teilarrays
  - Sortierung durch Quicksort
  - Sobald 1 Prozess fertig: Unterstützung für andere

Radix Sort

Zwei Arten:

LSD: least significant digit first

MSD: most significant digit first

- Sequentieller Radix Sort
- Parallelisierung

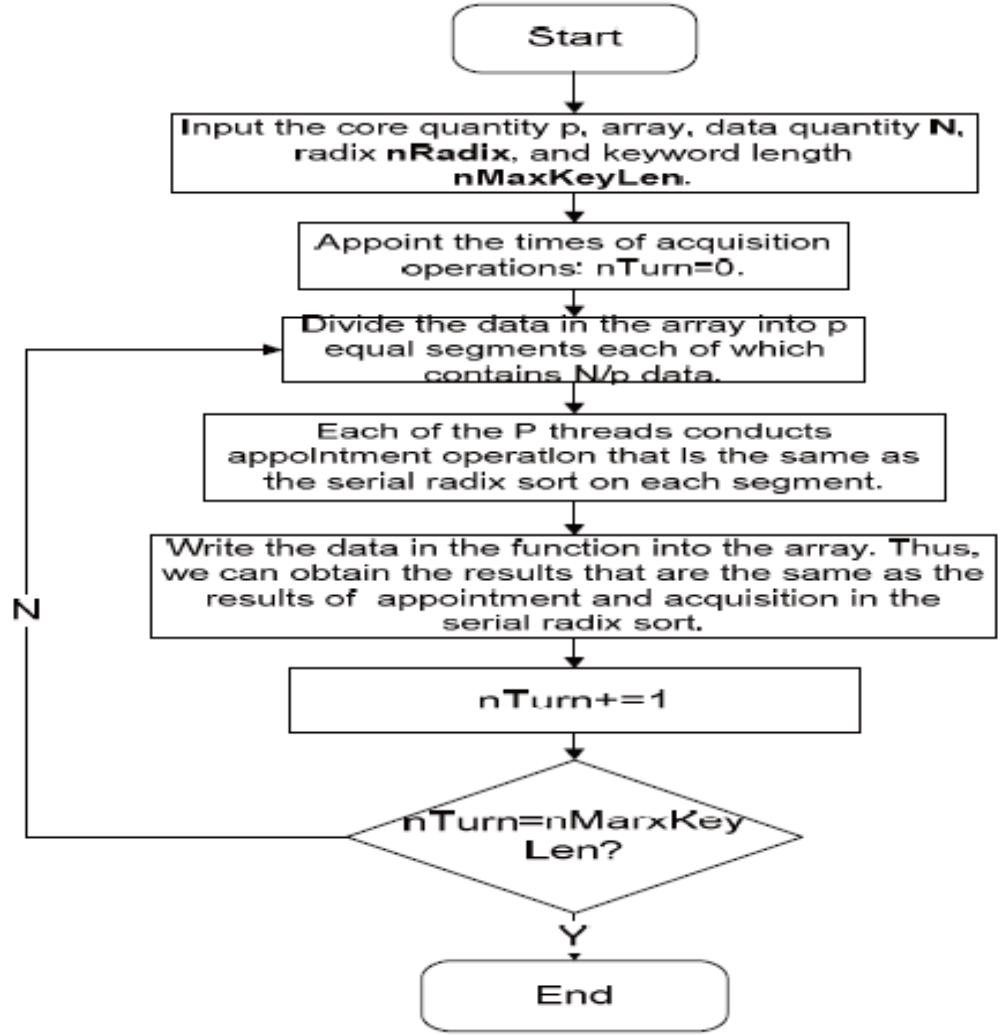
Sequentieler Radix Sort

Grundüberlegung:

2 Phasen:

- Partitionierung:
  - Daten werden in entsprechende Fächer aufgeteilt
- Sammeln
  - Daten werden aus den Fächern gesammelt

Parallelisierung



Vielen Dank für Ihre Aufmerksamkeit!