Diplomarbeit

Paralleles Quicksort für große Datenmengen auf Gittern

Thomas Hansch

Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler
Universität Karlsruhe
Prof. Dr.-Ing. R. Vollmar

Betreuer: Dipl.-Inform. Peter Sanders

Juni 1996

Hiermit versichere ich, die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe angefertigt zu haben. Die verwendeten Literaturquellen sind im Literaturverzeichnis aufgeführt.

Karlsruhe, den 17.Juni 1996

Kurzfassung

Sortieralgorithmen für sequentielle Rechner sind bereits seit Jahren ausgiebig untersucht worden. Dabei hat sich Quicksort als eines der besten universellen Sortierverfahren für Ein-Prozessor-Systeme erwiesen.

In dieser Arbeit wird gezeigt, daß Quicksort auch auf Parallelrechnern mit Prozessoren in Gitteranordnung ein guter Algorithmus ist. Die Leistung des parallelen Quicksorts hängt aber stark von der Implementierung ab. Deswegen werden drei verschiedene Varianten untersucht. In der erstens Variante wird die lokale Sortierung der Daten auf jedem Prozessor am Ende durchgeführt. Diese Variante erweist sich als erheblich schlechter, als wenn man die Daten am Anfang auf jedem Prozessor sortiert. Die dritte Variante arbeitet mit mehreren Pivotelementen und erweist sich damit als beste Implementierung. Drei Pivotelemente stellen sich dabei als optimale Anzahl für die Pivotelemente heraus.

Auf alle drei Varianten lassen sich verschiedene Optimierungstechniken wie z.B. “reduzierte Kommunikation”, “exakte Teilung” oder “exakte Partnerprozessoren” einsetzen. Diese Techniken werden in dieser Arbeit erläutert und an den verschiedenen Varianten getestet. Die Numerierung der Prozessoren hat ebenfalls einen Einfluß auf die Sortiergeschwindigkeit. Deshalb werden mehrere Prozessornumerierungen eingeführt und untersucht.

Die sich aus all diesen Möglichkeiten ergebende beste Implementierung des parallelen Quicksorts wird anschließend mit anderen parallelen Sortieralgorithmen auf dem selben Parallelrechner verglichen. Dabei stellt sich Quicksort als bester Algorithmus heraus. Durch den Einsatz mehrerer Pivotelemente und der verschiedenen Optimierungstechniken ist Quicksort auch auf Parallelrechnern ein guter Sortieralgorithmus.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Sortieralgorithmen für sequentielle Rechner sind bereits seit Jahren ausgiebig untersucht worden. Dabei hat sich Quicksort als eines der besten universellen Sortierverfahren erwiesen.

Das Gebiet der parallelen Sortieralgorithmen ist komplexer als das der sequentiellen Algorithmen, da andere Aspekte im Vordergrund stehen. Beim sequentiellen Sortieren auf einem Prozessor ist es das Ziel, die Anzahl der Speicherzugriffe und der Vergleiche zu minimieren. Beim parallelen Sortieren hingegen ist es wichtiger, die Arbeit möglichst gleichmäßig auf alle Prozessoren zu verteilen. Dabei entsteht ein Kommunikationsaufwand, der maßgeblich den Gesamtaufwand bestimmt. Ein schneller paralleler Sortieralgorithmus muß also die richtige Kombination aus gleichmäßiger Verteilung der Arbeit und geringem Kommunikationsaufwand aufweisen. Hierbei spielt die Hardwarearchitektur des Parallelrechners eine entscheidende Rolle.

In der Praxis sind heute Parallelrechner verfügbar, deren Prozessorarchitektur ein zweidimensionales $n \times n$-Gitter ist. Für diese Klasse von Parallelrechnern soll das parallele Quicksort eingesetzt werden.

In dieser Arbeit soll gezeigt werden, daß durch den Einsatz verschiedener Optimierungstechniken paralleles Quicksort auf Parallelrechnern mit Gitterstruktur sehr gut für die Sortierung von Datenmengen im Bereich von 128 bis 32768 Elementen pro Prozessor geeignet ist.

1.2 Bisherige Ergebnisse

Der Parallelrechner GCel der Firma Parsytec, der an der Universität Paderborn zur Verfügung steht, ist ein Parallelrechner, bei dem die 1024 Prozessoren in einem 32×32-Gitter verschaltet sind. Für diesen Parallelrechner wurden die Sortierverfahren Bitonicsort, Gridsort, Radixsort, Samplesort und Shearsort implementiert. Die Experimente wurden mit 32-Bit-Ganzzahlen durchgeführt, die gemäß dem NAS Parallel Benchmark[1] erzeugt wurden. Dabei erwies sich Bitonicsort bei kleinen Datenmengen (bis 2048 Elementen pro Prozessor) als bestes Sortierverfahren, während Samplesort bei größeren Datenmengen alle anderen Verfahren schlägt ([DGLM94]).

1.3 Zielsetzung

Der Zeitaufwand beim parallelen Quicksort läßt sich grob in zwei Gruppen unterteilen: Erstens in die Zeit, in der jeder Prozessor lokal arbeitet, und zweitens in die Zeit, in der alle Prozessoren miteinander Daten austauschen und kommunizieren müssen. Die lokale Arbeit besteht hauptsächlich aus der lokalen Sortierung der Elemente. Hier wird das sequentielle Quicksort aus der C-Standard-Bibliothek eingesetzt. Auf Optimierungen wie z.B. Codetuning usw. wird verzichtet, da diese Techniken aus dem Bereich der Programmiertechnik stammen und prinzipiell nichts mit parallelen Algorithmen zu tun haben. Statt dessen soll hier der parallele Algorithmus so verbessert werden, daß der Zeitaufwand für den Datenaustausch und die Kommunikation so weit wie möglich reduziert wird. Dabei stellen sich folgende Fragen:

Welche Prozessoren tauschen miteinander Daten aus? Wie ist die Entfernung der miteinander kommunizierenden Prozessoren? Wie kann man den Datenaustausch minimieren? Welche Prozessornumerierung muß gewählt werden, damit die kommunizierenden Prozessoren möglichst nahe beieinander liegen?

Die Aufgabe dieser Diplomarbeit besteht aus folgenden Teilen:

- Implementierung von Quicksort für Parallelrechner mit Prozessoren in Gitteranordnung
- Entwicklung von Optimierungstechniken, die zu einer Verbesserung der Sortierzeiten führen
- Implementierung dieser Optimierungstechniken
- Messungen und Analysen, um die optimale Implementierung zu ermitteln

[1] siehe [BBBB94]

Unabhängig von den Optimierungstechniken werden drei Varianten des parallelen Quicksorts implementiert:

- Quicksort mit lokaler Sortierung am Ende
- Quicksort mit lokaler Sortierung am Anfang
- Quicksort mit mehreren Pivotelementen

Ob bei der letzten Variante die lokale Sortierung am Anfang oder am Ende durchgeführt wird, hängt von den zuvor ermittelten Ergebnissen ab. Auf alle drei Varianten lassen sich die verschiedenen Optimierungstechniken einzeln oder in Kombination anwenden.

Ziel dieser Arbeit ist es, das parallele Quicksort in den verschiedenen Varianten mit den verschiedenen Optimierungstechniken zu implementieren und die beste Kombination zu ermitteln. Dabei wird erwartet, daß das parallele Quicksort durchaus in der Lage ist, die besten parallelen Sortierverfahren, die heute eingesetzt werden, zu schlagen. Um eine bessere Vergleichbarkeit der Ergebnisse zu haben, wurde das parallele Quicksort ebenfalls auf dem GCel der Universität Paderborn ausgeführt.

1.4 Gliederung

Im Kapitel 2 werden zuerst die Voraussetzungen und Rahmenbedingungen für das parallele Quicksort beschrieben. Das Kapitel 3 stellt die kollektiven Operationen vor, die man für die Kommunikation unter den Prozessoren benötigt. Als erste Variante wird in Kapitel 4 das parallele Quicksort mit lokaler Sortierung am Ende eingeführt. Anhand dieser ersten Variante werden dann in Kapitel 5 die verschiedenen Optimierungstechniken beschrieben und getestet. Als zweite Variante wird in Kapitel 6 das parallele Quicksort mit lokaler Sortierung am Anfang eingeführt, wobei die verschiedenen Optimierungstechniken gleich mit angewendet werden. In Kapitel 7 folgt die dritte Variante, die mit mehreren Pivotelementen arbeitet. Auch hier werden die verschiedenen Optimierungstechniken gleich mit verwendet. Zum Schluß erfolgt in Kapitel 8 eine Zusammenfassung der Ergebnisse. Dabei werden Vergleiche mit den Sortierverfahren angestellt, die an der Universität Paderborn ermittelt worden sind. Im Anhang A sind Datenblätter enthalten, die die Meßergebnisse jeder einzelnen Messung in Form von Grafiken und Tabellen wiedergeben.

Kapitel 2

Voraussetzungen

2.1 Hardwarearchitektur

Als Hardwarearchitektur dient ein nachrichtengekoppelter MIMD-Rechner, bei dem die Prozessoren in einem Gitter angeordnet sind. Konkret wurden zwei Transputersysteme eingesetzt:

- Ein Parsytec SuperCluster mit 64 T805 Transputern, der am Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler der Universität Karlsruhe zur Verfügung steht. Dieser Rechner diente hauptsächlich zu Entwicklungs- und Testzwecken.
- Ein Parsytec GCel mit 1024 T805 Prozessoren, der an der Universität Paderborn zur Verfügung steht. Beim GCel und SuperCluster ist die Kommunikation etwa gleich schnell, die Prozessorgeschwindigkeit liegt beim GCel aber höher. Für Messungen wurde ausschließlich der GCel verwendet.

Ein T805-Prozessor verfügt über 4 MB Hauptspeicher. Die Prozessoren sind in einem Gitter angeordnet, wobei die Numerierung zeilenweise erfolgt. Generell gilt, daß die Kommunikation zwischen zwei benachbarten Prozessoren schneller abläuft als zwischen zwei weiter entfernten Prozessoren. Außerdem treten bei kürzeren Kommunikationswegen weniger Kollisionen mit anderen Nachrichten auf.

2.2 Betriebssystem

Als Betriebssystem wurde COSY[1] eingesetzt. COSY wurde an der Universität Karlsruhe als hardwareunabhängiges Betriebssystem für Parallelrechner entwickelt.

[1] siehe [BuGi94]

Das Versenden von Nachrichten von einem Prozessor an einen anderen entfernten Prozessor erfolgt Schritt für Schritt, indem die Nachricht immer an einen Nachbarprozessor weitergegeben wird. Für die Weiterleitung von Nachrichten sind sogenannte Router zuständig, die auch den Weg bestimmen, auf dem eine Nachricht verschickt wird. Für die Kommunikation unter den Prozessoren stehen in Cosy zwei verschiedene Router zur Verfügung, wobei aber nur der Gridrouter verwendet wird, weil dieser prinzipiell die besseren Ergebnisse liefert. Die Funktionsweise des Gridrouters ist so, daß eine Nachricht zuerst immer so weit wie möglich in die eine Richtung geschickt wird. Anschließend erfolgt das Verschicken in die dazu orthogonale Richtung. Hat man z.B. Kommunikationswege, die nur entlang einer Achse verlaufen, so würde der Gridrouter die Daten wirklich nur auf dieser Achse versenden.

Die maximale Länge einer Nachricht beträgt 4096 Bytes bzw. 1024 32-Bit-Ganzzahlen.

2.3 Softwarearchitektur

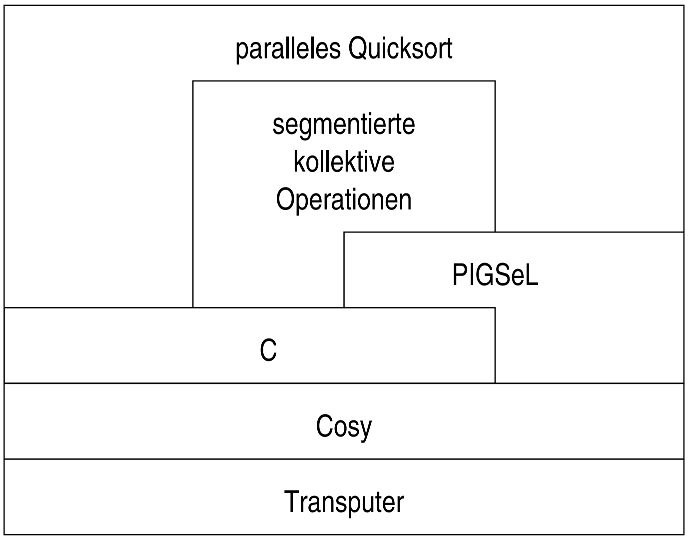


Abbildung 2.1: Softwarearchitektur

Die Implementierung des parallelen Quicksorts erfolgt unter Verwendung der PIGSeL-Bibliothek[2], der segmentierten kollektiven Operationen[3] und der Programmiersprache C. In Abbildung 2.1 ist die Softwarearchitektur dargestellt.

[2]siehe [Sand96]
[3]siehe [Hans95]

2.3.1 Programmiermodell

Die parallele Ablaufstruktur des Algorithmus wird als SPMD (Single Program Multiple Data) bezeichnet. Das bedeutet in diesem Fall, daß das gleiche Programm auf allen Prozessoren gleichzeitig abläuft. Zur Kommunikation unter den Prozessoren werden ausschließlich Sende- und Empfangsoperationen (send/receive) verwendet.

2.3.2 Bibliothek PIGSeL

Die Bibliothek PIGSeL[4] (Parallel Implicit Graph Search Library), die am Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler der Universität Karlsruhe entwickelt wird, dient dazu, parallele Algorithmen hardwareunabhängig zu implementieren. Diese Bibliothek stellt Sende- und Empfangsoperationen (send/receive) zur Verfügung und sorgt für die korrekte Umsetzung unter verschiedenen parallelen Betriebssystemen wie z.B. COSY, Parix oder PVM.

2.3.3 Segmentierte kollektive Operationen

Häufig werden neben den Sende- und Empfangsoperationen (send/receive) aber auch komplexere Operationen benötigt, die über eine größere Zahl von Prozessoren agieren, wie z.B.

- *broadcast*-Operation: Rundruf an alle Prozessoren,
- *reduce*-Operation: Reduktion der Daten aller Prozessoren,
- *scan*-Operation: Postfix-Berechnung über Daten aller Prozessoren,
- *barrier*-Operation: Synchronisation aller Prozessoren.

Für die *reduce*- und *scan*-Operationen können verschiedene kommutative und assoziative Verknüpfungsoperationen wie z.B. Addition, Multiplikation, Maximum, Minimum, UND-Verknüpfung und ODER-Verknüpfung eingesetzt werden.

Die kollektiven Operationen agieren in der Regel über alle Prozessoren. Es gibt aber auch Fälle, in denen man diese kollektiven Operationen nur auf einem Teil der Prozessoren ausführen möchte. Deshalb gibt es segmentierte kollektive Operationen, die auf einem Segment von fortlaufend numerierten Prozessoren ausgeführt werden können. Die kollektiven Operationen über alle Prozessoren sind somit nur ein Spezialfall der segmentierten kollektiven Operationen.

Die segmentierten kollektiven Operationen[5] werden in Kapitel 3 detaillierter erläutert.

[4]siehe [Sand96]
[5]siehe [Hans95]

2.4 Variable Prozessornumerierung

Um eine flexible Anpassung des Quicksort-Algorithmus an verschiedene Prozessorarchitekturen zu ermöglichen, wird intern mit einer logischen Prozessornumerierung gearbeitet. Durch die Anwendung einer Projektionsfunktion wird die logische Prozessornummer auf die physische Prozessornummer abgebildet. So besteht die Möglichkeit, durch Änderung der Projektionsfunktion verschiedene Prozessornumerierungen (z.B. Zeilennumerierung oder Hilbertnumerierung) zu verwenden, ohne am Algorithmus etwas ändern zu müssen.

Zur Vereinfachung wird bei der Prozessornummer nicht mehr zwischen logischer und physischer unterschieden, sondern es ist darunter immer die logische Prozessornummer zu verstehen.

2.5 Vereinbarungen

Folgende Bezeichner haben hier eine feste Bedeutung:

- P: Gesamtzahl der Prozessoren im Gitter
- N: Anzahl der Elemente pro Prozessor
- p, q: logische Prozessornummern
- x, y: Elementnummern
- $D_{p,x}$: Datenelement x von Prozessor p

Eine Reihe von Begriffen haben eine besondere Bedeutung und werden deshalb an dieser Stelle erläutert:

- Segment: Eine Teilmenge von Prozessoren, die eine fortlaufende Numerierung besitzen.
- Wurzelprozessor: Der Prozessor mit der kleinsten Prozessornummer in einem Segment.
- Sprung: Ein Sprung zwischen zwei Prozessoren bedeutet, daß die Prozessoren zwar benachbarte Prozessornummern haben, physikalisch aber nicht direkt miteinander verbunden sind.
- Variante: Als (Quicksort-)Varianten werden die drei verschiedenen Versionen des Quicksort-Algorithmus bezeichnet, die hier untersucht werden. Es handelt sich um die Varianten, die schon in der Zielsetzung (Kapitel 1.3) erwähnt wurden.

- Optimierungstechnik: Verschiedene Verbesserungen, die an einer Quicksort-Variante vorgenommen werden können, werden als Optimierungstechnik bezeichnet.

- Konfiguration: Als Konfiguration wird die Kombination von verschiedenen Optimierungstechniken bezeichnet.

2.6 Datengenerierung

Sortiert werden 32-Bit-Ganzzahlen. Die Generierung erfolgt gemäß dem NAS Parallel Benchmark[6].

Dazu wird der Zufallszahlengenerator eingesetzt, der ebenfalls in [BBBB94] beschrieben ist. Die Datenelemente müssen im Bereich $[0, B_{max})$ liegen. Der Zufallszahlengenerator liefert eine Zahlenfolge z_j aus Zahlen im Bereich $[0, 1]$. K_i stellt das i-te Datenelement dar, das sich wie folgt berechnet:

$$K_i = \lfloor B_{max}(z_{4i+0} + z_{4i+1} + z_{4i+2} + z_{4i+3})/4 \rfloor \qquad \text{für } i = 0, 1, ..., P * N - 1$$

Alle Berechnungen werden mit 64-Bit-Fließkommazahlen durchgeführt und erst am Ende wird das Ergebnis auf eine 32-Bit-Ganzzahl konvertiert. Die Startwerte für den Zufallszahlengenerator und für B_{max} werden ebenfalls in [BBBB94] vorgeschrieben.

Durch die obige Berechnungsweise haben die Elemente eine Gauß-Verteilung.

Für die verteilte Berechnung der Daten auf den einzelnen Prozessoren ist zu beachten, daß jeder Prozessor genau die Daten berechnet, die seiner Lage in der Reihenfolge der Prozessoren entspricht. Das heißt, alle Prozessoren berechnen die gleichen Zufallszahlen, aber Prozessor 0 behält nur die Daten K_0 bis K_{N-1}, Prozessor 1 behält nur die Daten K_N bis K_{2N-1} usw.

Auf jedem Prozessor befinden sich am Anfang gleich viele Daten.

Bei einem praktischen Einsatz der in dieser Arbeit beschriebenen Algorithmen kann man natürlich keine gleichmäßig verteilten Zufallszahlen erwarten. Allerdings ist es möglich, dem Sortiervorgang eine Datenaustauschphase vorschalten, in der alle Daten zufällig und gleichmäßig über die Prozessoren verteilt werden. Damit werden dann die hier angenommenen Voraussetzungen erfüllt. Untersuchungen über das Sortieren nicht zufälliger Daten sind nicht Gegenstand dieser Arbeit, aber im Ausblick (Kapitel 8.6) wird noch einmal darauf eingegangen.

---

[6] siehe [BBBB94]

2.7 Parallele Sortierung

Eine Folge von Daten, die sich über mehrere Prozessoren erstreckt, ist sortiert, wenn gilt:

1. lokale Sortierung: $\forall p \forall x \forall y : x \leq y \Rightarrow D_{p,x} \leq D_{p,y}$
2. globale Sortierung: $\forall p \forall q \forall x \forall y : p < q \Rightarrow D_{p,x} \leq D_{q,y}$

Dabei stellen p und q die logischen Prozessornummern und x und y die Nummern des jeweiligen lokalen Datenelementes dar.

Die erste Bedingung sichert zu, daß die Daten lokal sortiert sind. Die zweite Bedingung sorgt dafür, daß die lokal sortierten Daten in sortierter Reihenfolge auf den Prozessoren liegen.

Die Daten sind vor und nach dem Sortiervorgang über alle Prozessoren verteilt.

Kapitel 3

Segmentierte kollektive Operationen

3.1 Funktionalität der kollektiven Operationen

Bei den segmentierten kollektiven Operationen hat immer ein Prozessor eine besondere Bedeutung, da er Daten an alle Prozessoren versendet oder Daten von allen anderen Prozessoren empfängt. Bei den hier verwendeten segmentierten kollektiven Operationen ist das immer der Prozessor mit der kleinsten Prozessornummer. Zur Vereinfachung wird dieser Prozessor als Wurzelprozessor bezeichnet.

Die segmentierten kollektiven Operationen haben folgende Funktionen:

- *broadcast*-Operation: Der Wurzelprozessor verschickt eine Nachricht an alle anderen Prozessoren des Segments.

- *reduce*-Operation: Der Wurzelprozessor empfängt Daten von allen Prozessoren des Segments, die mit einer assoziativen und kommutativen Operation $\otimes$ verknüpft werden. Besteht ein Segment aus den Prozessoren p bis q, so ist das Ergebnis beim Wurzelprozessor $K_p = \bigotimes_{j=p}^{q} k_j$ mit z.B. $\otimes \in \{\text{Add, Mult, Max, Min, And, Or}\}$. Dabei ist k_i der Wert, der vom i-ten Prozessor stammt. Die Ergebnisse bei den anderen Prozessoren des Segments K_i mit $p < i \leq q$ sind undefiniert.

  Beispiel: Mit der Verknüpfungsoperation Add kann man eine Summe über alle Prozessoren eines Segments bilden. Es lassen sich aber auch komplexere Operationen für $\otimes$ einsetzen, wie z.B. eine Vektoraddition, die mit einer einzigen *reduce*-Operation die Summe über alle Komponenten des Vektors bildet.

- *scan*-Operation: Für den Wurzelprozessor unterscheidet sich die *scan*-Operation nicht von der *reduce*-Operation. Im Gegensatz zur *reduce*-Operation erhalten aber die anderen Prozessoren des Segments auch ein Ergebnis. Dieses Ergebnis ergibt sich aus den Daten der Prozessoren, deren Prozessornummern größer oder gleich der eigenen Nummer sind. Es handelt sich also um eine Postfix-Berechnung. Formal berechnet sich das Ergebnis für den i-ten Prozessor ($p \leq i \leq q$) zu $K_i = \bigotimes_{j=i}^{q} k_j$ mit z.B. $\otimes \in \{$Add, Mult, Max, Min, And, Or$\}$.

- *barrier*-Operation: Diese Operation dient zur Synchronisation von Prozessoren in einem Segment. Die *barrier*-Operation ist dabei eine Kombination aus einer *reduce*-Operation, die leere Nachrichten einsammelt, und einer *broadcast*-Operation, die nach der *reduce*-Operation den Prozessoren mitteilt, daß alle Prozessoren die *barrier*-Operation aufgerufen haben und somit diese Operation beendet ist.

Für die *reduce*- und *scan*-Operationen können verschiedene kommutative und assoziative Verknüpfungsoperationen wie z.B. Addition, Multiplikation, Maximum, Minimum, UND-Verknüpfung und ODER-Verknüpfung eingesetzt werden. Weitere Verknüpfungsoperationen sind Concat und Merge. Die Concat-Operation hängt zwei Folgen hintereinander, während die Merge-Operation zwei sortierte Folgen zu einer neuen sortierten Folge zusammensetzt.

Als Verknüpfungsoperationen kommen beim Quicksort nur die Addition bzw. die Vektoraddition und die Merge-Operation zum Einsatz. Alle Operationen beziehen sich dabei auf den Datentyp Ganzzahl.

Weitere Informationen über die segmentierten kollektiven Operationen können in [Hans95] nachgelesen werden.

3.2 Optimierung der *scan*-Operation

Für die Verwendung der kollektiven Operationen beim Quicksort wurde die *scan*-Operation geändert.

Bei der ursprünglichen Implementierung handelte es sich um eine asynchrone Version, die sofort nach dem Eintreffen von Daten diese an die betreffenden Söhne weitergeschickt hat. Somit wurden mehrfach verschiedene Daten an den gleichen Sohn geschickt.

In der neuen Implementierung werden die Daten für einen Sohn zuerst beim Vater gesammelt. Nachdem alle Daten eingetroffen sind, werden diese mit der entsprechenden Verknüpfungsoperation verknüpft und dann als eine einzige Nachricht an den Sohn geschickt. Als Folge dieser Vorgehensweise ergibt sich, daß die Daten alle zuerst in Richtung der Wurzel fließen. Erst wenn dort alle Informationen angekommen sind, werden Daten an die Söhne zurückgegeben. Diese Vorgehensweise entspricht einer Synchronisation mit einer *barrier*-Operation. Somit ist die *scan*-Operation jetzt eine synchrone Operation und bewirkt als Seiteneffekt eine Synchronisation.

Kapitel 4

Paralleles Quicksort mit lokaler Sortierung am Ende

Als abkürzende Schreibweise für die lange Bezeichnung dieser ersten Quicksort-Variante wird der Begriff paralleles Quicksort verwendet, wenn die Art der lokalen Sortierung aus dem Kontext heraus klar wird.

4.1 Das Vorbild: sequentielles Quicksort

Das sequentielle Quicksort ist ein Teile- und Herrsche-Algorithmus (Divide- and Conquer-Algorithmus), der sich auf einem sequentiellen Rechner sehr einfach rekursiv implementieren läßt. Der Algorithmus soll an dieser Stelle (zitiert aus [OtWi90]) noch einmal kurz in Erinnerung gerufen werden:

Quicksort(F:Folge)

1. Falls F die leere Folge ist oder nur aus einem Element besteht, bleibt F unverändert.

2. Divide: Wähle ein Pivotelement v von F und teile F ohne v in zwei Teilfolgen bezüglich v, so daß gilt:

   - F_1 enthält nur Elemente $\leq v$
   - F_2 enthält nur Elemente $\geq v$

3. Conquer: Führe nacheinander Quicksort(F_1) und Quicksort(F_2) aus.

4. Merge: Bilde die Ergebnisfolge F durch Hintereinanderhängen von F_1, v, F_2 in dieser Reihenfolge.

4.2 Der parallele Quicksort-Algorithmus

In Analogie zum obigen sequentiellen Quicksort-Algorithmus sieht der parallele Quicksort-Algorithmus wie folgt aus:

Quicksort(F:Folge, S:Segment)

1. Lokale Sortierung: Falls das Segment S nur aus einem Prozessor besteht, sortiere F lokal auf S. Ende.

2. Pivotwahl: Wähle ein Pivotelement v von F.

3. Datenaustausch: Teile F in zwei Teilfolgen bezüglich v und teile S in zwei Teilsegmente, so daß gilt:

   - F_1 im Segment S_1 enthält nur Elemente $\leq v$
   - F_2 im Segment S_2 enthält nur Elemente $> v$

   und führe parallel Quicksort(F_1, S_1) und Quicksort(F_2, S_2) aus.

Die Implementierung erfolgt allerdings nicht in rekursiver, sondern in iterativer Form. Dennoch wird deutlich, daß der Algorithmus im wesentlichen aus drei Komponenten (lokaler Sortierung, Pivotwahl, Datenaustausch) besteht, die für die Gesamtlaufzeit des Algorithmus verantwortlich sind.

4.2.1 Lokale Sortierung

Falls ein Segment nur noch aus einem Prozessor besteht, wird die lokale Sortierung durchgeführt. Als Sortierverfahren kommt das sequentielle Quicksort zum Einsatz, da es sich hierbei um eines der besten Sortierverfahren handelt. Das sequentielle Quicksort stammt aus der C-Standard-Bibliothek. Auf Optimierungen, z.B. durch eine eigene Implementierung mit Codetuning und Anpassung auf 32-Bit-Ganzzahlen, wird an dieser Stelle verzichtet, da diese Techniken aus dem Bereich der Programmiertechnik stammen und prinzipiell nichts mit parallelen Algorithmen zu tun haben.

Der maximale Aufwand für die lokale Sortierung hängt in der Regel von dem Prozessor ab, der die meisten Daten zu sortieren hat. Um also eine gute Auslastung der Prozessoren zu erzielen, muß man versuchen, daß die Daten gleichmäßig über die Prozessoren verteilt sind. Entscheidenden Einfluß auf die Verteilung der Daten über die Prozessoren haben dabei die Pivotwahl und der Datenaustausch.

4.2.2 Pivotwahl

Die zu sortierenden Elemente werden gemäß dem Pivotelement aufgeteilt. Ein Teil der Prozessoren bekommt die Elemente kleiner oder gleich dem Pivotelement, der andere Teil der Prozessoren die Elemente größer dem Pivotelement zugeteilt. Da eine gleichmäßige Verteilung der Daten wünschenswert ist, damit für die lokale Sortierung möglichst überall gleich viele Elemente vorliegen, wäre das ideale Pivotelement der Median der zu sortierenden Folge. Mit diesem idealen Pivotelement würden die Daten genau in der Mitte aufgespalten. Der Zeitaufwand für die Ermittlung des Medians über alle Elemente ist aber so hoch, daß man sich darauf beschränkt, als Pivotelement Elemente zu wählen, die dem Median ziemlich nahe kommen.

Im sequentiellen Fall wird als Pivotelement in der Regel das erste, mittlere, letzte oder ein zufälliges Element gewählt. Weiterhin gibt es Strategien, die sich bemühen, ein möglichst gutes Pivotelement zu ermitteln. Zum Beispiel bietet der 3-Median, also der Median aus dem ersten, mittleren und letzten Element, eine gute Möglichkeit, schnell ein Pivotelement zu erhalten, das recht gut geeignet ist.

Für die Ermittlung eines Pivotelements im parallelen Fall bieten sich mehrere Möglichkeiten an. Auch wenn die zu sortierenden Daten eine bestimmte Verteilung haben, wie z.B. hier die Gauß-Verteilung, so sind diese Daten dennoch zufällig über die Prozessoren verteilt. Die Daten auf jedem Prozessor haben somit die gleiche Verteilung wie die Gesamtdatenmenge. Deshalb kann das Pivotelement auf einem beliebigen Prozessor ermittelt werden. Der Einfachheit halber bietet sich der Wurzelprozessor für die Wahl des Pivotelements an.

Entscheidend für die Güte des Pivotelements ist die Frage, wie man das Pivotelement lokal wählt. Es bietet sich an, den Median aus verschiedenen großen Teilmengen zu ermitteln.

- 3-Median: Der Wurzelprozessor ermittelt den Median aus dem ersten, mittleren und letzten Element der Folge.
- $\log(N)$-Median: Der Wurzelprozessor nimmt die ersten $\log(N)$ Elemente der Folge und bestimmt daraus den Median.
- $\sqrt{N}$-Median: Der Wurzelprozessor nimmt die ersten $\sqrt{N}$ Elemente der Folge und bestimmt daraus den Median.

Ein cN-Median mit $c \ll 1$ ist nicht sinnvoll, weil der Aufwand zur Ermittlung des Pivotelements dann $O(cN \log(cN))$ ist, und somit in der gleichen Größenordnung liegt wie die lokale Sortierung ($O(N \log(N))$) selbst. Durch $c \ll 1$ hat die Ermittlung des Pivotelements zwar eine viel kleinere Konstante, aber da die Wahl des Pivotelements $\log(P)$ mal durchgeführt werden muß, beträgt der Gesamtaufwand für die Pivotwahl dann $O(\log(P)cN \log(cN))$. Damit der Aufwand für die Pivotwahl nicht in der gleichen Größenordnung liegt wie die lokale Sortierung, wird maximal der $\sqrt{N}$-Median verwendet.

4.2.3 Prozessornumerierung

Beim Laufzeitverhalten von Quicksort spielt die Art der Prozessornumerierung in Bezug auf den Datenaustausch eine große Rolle. Der Grund dafür ist, daß durch die iterative Aufteilung der Prozessoren in Teilsegmente die Kommunikation und der Datenaustausch in immer kleineren Segmenten stattfindet. Damit in diesen kleiner werdenden Segmenten die Kommunikation möglichst schnell erfolgt, sollten die Prozessoren eines Segments so nahe wie möglich beieinander liegen. Bei der Verwendung einer Zeilennumerierung (Abbildung 4.1) ist das meistens nicht der Fall, weil Segmente Sprünge vom Ende der einen Zeile zum Anfang der nächsten Zeile enthalten können. Bei der Schlangennumerierung (Abbildung 4.2) tritt dieses Problem nicht auf, aber dafür haben beide Numerierungen gemeinsam den Nachteil, daß kleine Segmente letztendlich nur noch aus einer 1-dimensionalen Reihe von Prozessoren bestehen. Innerhalb dieser Reihe sind die Kommunikationswege dann wieder recht lang, so daß der Aufwand für die Kommunikation steigt.

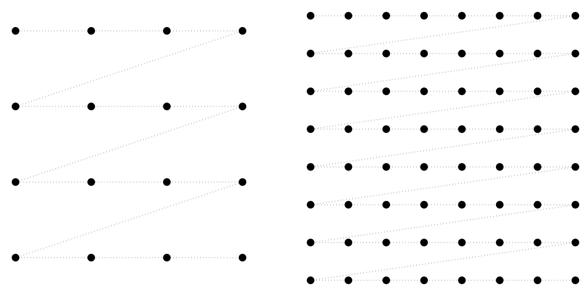


Abbildung 4.1: Zeilennumerierung für 16 und 64 Knoten

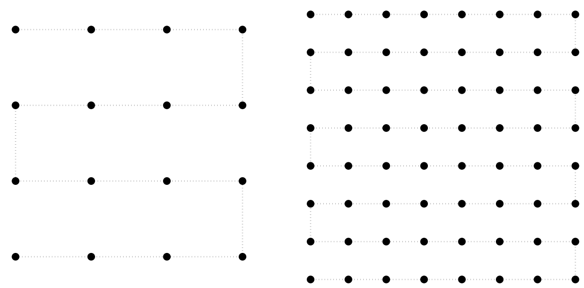


Abbildung 4.2: Schlangennumerierung für 16 und 64 Knoten

Bei der Hilbertnumerierung (Abbildung 4.3) liegen die Prozessoren eines Segments statt dessen immer in einem "Klumpen" beieinander.

In [Hans95] ist die Hilbertnumerierung gegenüber der Zeilennumerierung am Beispiel Quicksort getestet worden. Dabei spart man durch die Hilbertnumerierung bis zu 35% der Laufzeit ein.

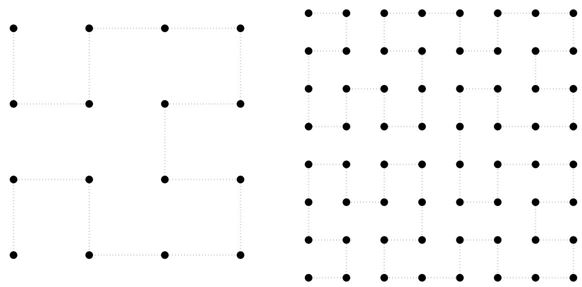


Abbildung 4.3: Hilbertnumerierung für 16 und 64 Knoten

4.2.4 Datenaustausch

Nach der Pivotwahl ist das Pivotelement allen Prozessoren bekannt. Um den Datenaustausch vorzubereiten, zählt jeder Prozessor lokal, wie viele seiner Elemente kleiner oder gleich bzw. größer dem Pivotelement sind. Anschließend wird die Folge auf jedem Prozessor so umsortiert, daß alle Elemente kleiner oder gleich dem Pivotelement links vom Pivotelement liegen, alle anderen rechts davon. In der Abbildung 4.4 repräsentieren die Kästchen die Prozessoren. Die Prozessornummern befinden sich oberhalb der Kästchen. Die Zahlen in den Kästchen geben an, wie viele Elemente auf jedem Prozessor vorhanden sind, wobei die Schreibweise mit | gleichzeitig angibt, wie viele Elemente kleiner oder gleich bzw. größer dem Pivotelement sind.

Nachdem die Zahl der Elemente für die beiden Teilfolgen ermittelt worden ist, findet eine *scan*-Operation mit Vektoraddition über die Größe der beiden Teilfolgen statt. Das Ergebnis, das jeder Prozessor erhält, ist ebenfalls in der Abbildung 4.4 dargestellt. Die entsprechende Zeile ist mit S gekennzeichnet. Das Ergebnis dieser Postfix-Operation stellt eine Numerierung der Elemente über alle Prozessoren dar. Anhand der Numerierung können später die Prozessoren ermittelt werden, zu denen diese Elemente geschickt werden müssen.

Anhand des Beispiels in Abbildung 4.4 soll der Datenaustausch einmal exemplarisch dargestellt werden. Jeder Prozessor hat für sich ermittelt, wie viele seiner Elemente kleiner oder gleich bzw. größer dem Pivotelement sind. Die Schreibweise 6 | 4 bedeutet, daß sechs Elemente kleiner oder gleich und vier Elemente größer dem Pivotelement sind.

Durch eine *scan*-Operation mit einer Vektoraddition werden die Elemente durchnumeriert. Die vier Elemente bei Prozessor 15, die kleiner oder gleich dem Pivotelement sind, erhalten die Positionen 1 bis 4, die fünf Elemente bei Prozessor 14 die Positionen 5 bis 9, die sechs Elemente bei Prozessor 13 die Positionen 10 bis 15 usw. Die drei Elemente bei Prozessor 0 erhalten die Positionen 84 bis 86. Das gleiche passiert mit den Elementen größer dem Pivotelement, nur mit dem Unterschied, daß sich die Positionsangaben auf das andere Segment beziehen. Die sechs Elemente von Prozessor 15 erhalten die Positionen 1 bis 6, die fünf Elemente von Prozessor 14 die Positionen 7 bis 11 usw.

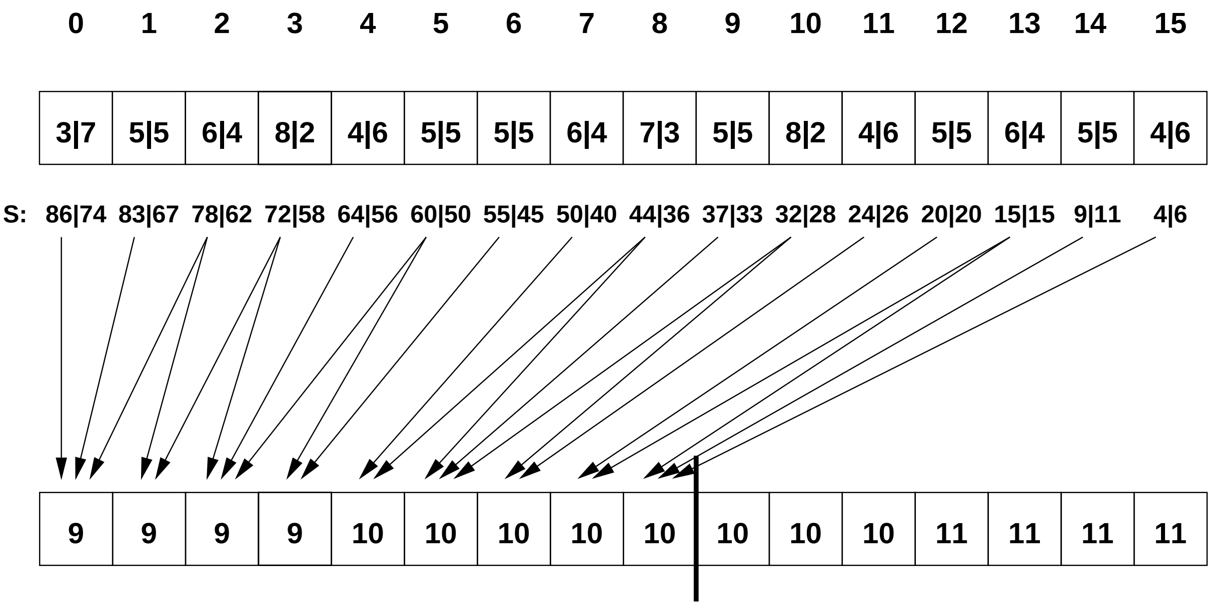


Abbildung 4.4: Datenaustausch (Kommunikationswege nur für Elemente kleiner oder gleich dem Pivotelement)

Das Ergebnis der *scan*-Operation bei Prozessor 0 gibt ebenfalls an, wie viele Elemente insgesamt kleiner oder gleich bzw. größer dem Pivotelement sind. Diese Ergebnisse sind für die anderen Prozessoren aber auch von Bedeutung und werden deshalb mit einer *broadcast*-Operation verschickt. Mit diesen Gesamtsummen kann jetzt jeder Prozessor selbst berechnen, wie viele der Prozessoren sich um die eine und um die andere Teilfolge kümmern müssen. Aus den Gesamtsummen berechnet sich das Verhältnis $86 : 74$, in dem auch die Prozessoren aufgeteilt werden müßten, nämlich $8{,}6 : 7{,}4$. Durch Rundung bedeutet das in diesem Beispiel, daß neun Prozessoren die Elemente kleiner oder gleich dem Pivotelement und sieben Prozessoren die Elemente größer dem Pivotelement erhalten werden. Außerdem weiß jeder Prozessor auch, wie viele Elemente jeder andere Prozessor erhalten wird, denn diese Anzahl berechnet sich aus der Gesamtsumme für ein Segment dividiert durch die Anzahl der Prozessoren. Der Rest dieser Division gibt an, wie viele Prozessoren ein Element mehr erhalten werden. Wegen $86/9 = 9$ *Rest* 5 erhalten in Abbildung 4.4 die Prozessoren 0 bis 3 jeweils neun Elemente, die restlichen fünf Prozessoren 4 bis 8 jeweils zehn Elemente. Natürlich kann jeder Prozessor auch für sich entscheiden, zu welchem Segment von Prozessoren er gehört und wie viele Elemente der Teilfolge er erhalten wird.

Nun kann der Datenaustausch beginnen. In Abbildung 4.4 sowie in allen folgenden Beispielen werden aus Gründen der Übersichtlichkeit nur die Pfeile eingezeichnet, die das Verschicken der Elemente kleiner oder gleich dem Pivotelement darstellen.

Die 1-dimensionale Anordnung der Prozessoren dient aber nur der Anschaulichkeit. In

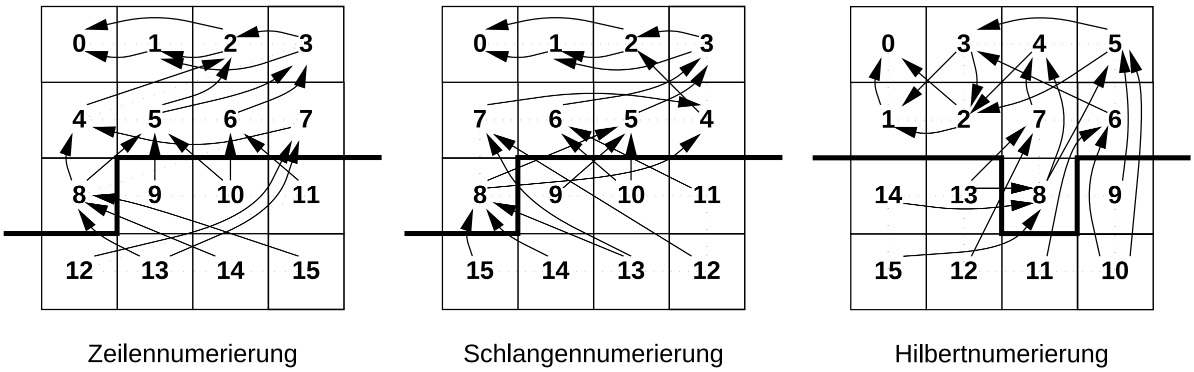


Abbildung 4.5: Datenaustausch im Gitter mit verschiedenen Numerierungen

Abbildung 4.5 sind die 16 Prozessoren im Gitter angeordnet. Hier wird dann der Einfluß der Prozessornumerierung deutlich. Es sind ebenfalls nur die Kommunikationswege dargestellt, die sich auf die Elemente kleiner oder gleich dem Pivotelement beziehen. Diese Kommunikationswege stellen nicht die realen Pfade dar, über die tatsächlich die Informationen transportiert werden, denn diese hängen vom verwendeten Router ab[1]. Aber wenn sich zwei Kommunikationswege in der Grafik überschneiden, kann man in der Regel davon ausgehen, daß sich auch die realen Kommunikationspfade überschneiden. Außerdem kann man annehmen, daß die Länge der eingetragenen Kommunikationswege mit der Länge der realen Kommunikationspfade unabhängig vom verwendeten Router korreliert.

Bei der 2-dimensionalen Anordnung der Prozessoren zeigt sich unabhängig von der verwendeten Numerierung, daß die Kommunikationswege kreuz und quer durch das Gitter verlaufen.

Nach dem Datenaustausch ist eine Synchronisation mit einer *barrier*-Operation nötig. Anschließend arbeiten die Prozessoren der beiden Segmente autonom weiter. Die neue Iteration beginnt dann wieder mit der Wahl eines Pivotelements.

An dieser Stelle wird der Algorithmus einmal exemplarisch als Pseudocode wiedergegeben, um den prinzipiellen Ablauf zu demonstrieren:

[1] siehe Kapitel 2.2

```
Solange das Segment eine Mindestgröße nicht unterschreitet
{
    Wahl eines Pivotelements
    Vorbereitung des Datenaustausches
    Versenden der Daten
    Empfangen der Daten
    Umschalten auf neues Segment
    Synchronisation mit barrier-Operation
}
Lokale Sortierung
```

4.3 Meßverfahren

Alle verwendeten Meßergebnisse wurden auf dem GCel der Universität Paderborn ermittelt. Wegen der schlechten Verfügbarkeit des kompletten Rechners erfolgten die Messungen immer nur mit 256 Prozessoren. Lediglich beim Vergleich der Ergebnisse mit anderen parallelen Sortierverfahren wurden Messungen mit 1024 Prozessoren durchgeführt.

Alle Messungen sind im Anhang in Form von Datenblättern dargestellt. Als Muster ist auf Seite 28 das Datenblatt 1 zu finden.

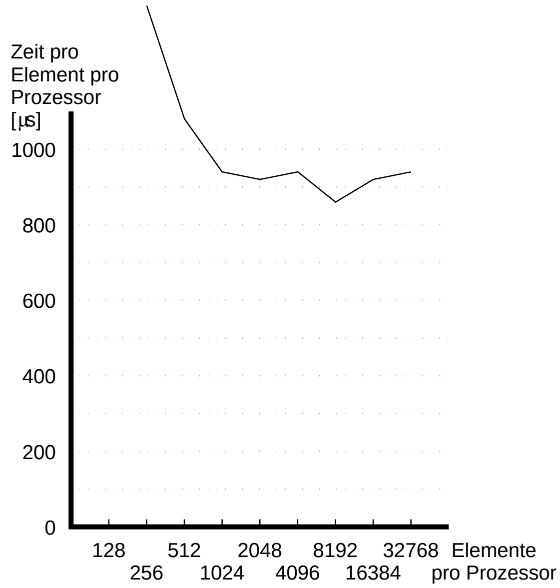
Die kleinste zu sortierende Datenmenge beträgt 128 Elemente pro Prozessor. Jeweils durch Verdopplung der Datenmenge wurden Messungen bis zu 32768 Elementen pro Prozessor durchgeführt. Für jede Datenmenge wurde die Messung dabei zehnmal wiederholt. Die daraus gemittelten Werte werden als Meßergebnisse verwendet.

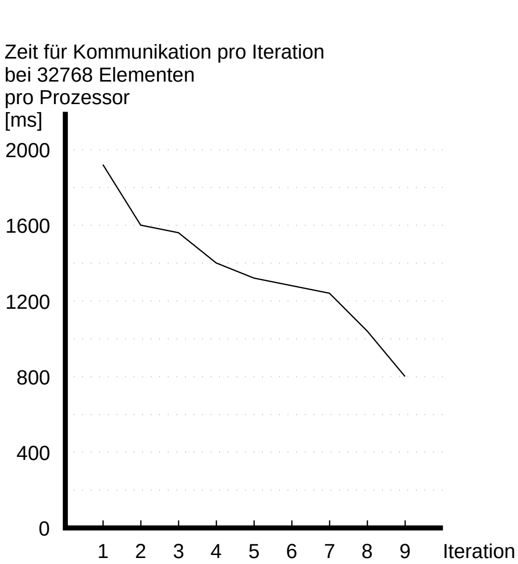
In der ersten Tabelle eines Datenblattes werden die verwendete Variante und alle verwendeten Optimierungstechniken beschrieben. Eine Erläuterung dieser Optimierungstechniken erfolgt in Kapitel 5.

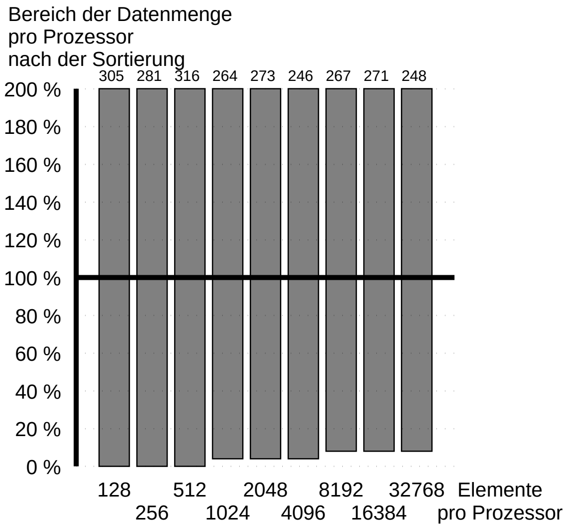
Die Grafik links oben auf dem Datenblatt gibt die Zeit pro Element pro Prozessor an. Die Einheit dabei ist Mikrosekunden (μs). Als weitere Bezeichnung für die Zeit pro Element pro Prozessor wird hier der Begriff Sortierzeit verwendet. Durch die Angabe der Sortierzeit in Abhängigkeit der Datenmenge kann man sehr leicht die Güte eines Verfahrens erkennen. Das Minimum gibt dann die Datenmenge an, bei der das Verfahren optimal ist. Die genauen Meßwerte sind in der unteren Tabelle auf dem Datenblatt nachzulesen.

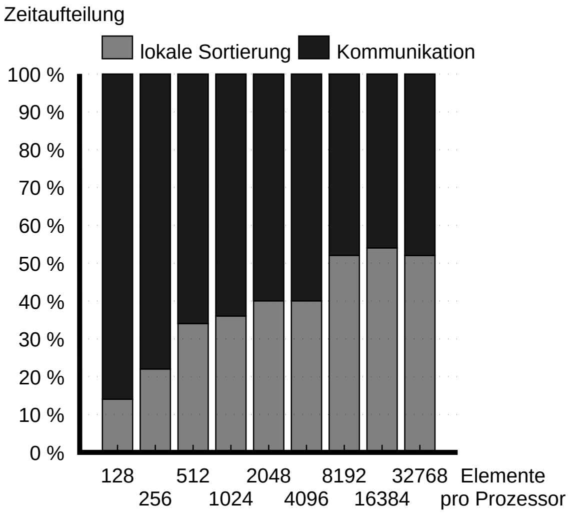
Die Grafik oben rechts auf dem Datenblatt bezieht sich nur auf die Messung mit 32768 Elementen pro Prozessor. Dabei wird die Zeit in Millisekunden (ms) dargestellt, die für die Kommunikation in jeder einzelnen Iteration benötigt wurde. Durch die Aufteilung der Segmente in jeder Iteration werden die Entfernungen der kommunizierenden Prozessoren immer kürzer, so daß man meistens einen fallenden Kurvenverlauf erhält. Die Meßwerte für jede einzelne Iteration werden als Maximum über alle Prozessoren ermittelt. Diese

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	3-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein









Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1976	1378	1081	944	924	942	859	917	931
Gesamtzeit [s]	0.25	0.35	0.55	0.97	1.89	3.86	7.05	15.04	30.54
Zeit für Kommunikation [ms]	232	294	396	671	1344	2506	3856	7553	16362
Zeit für lokale Sortierung [ms]	39	83	212	382	880	1689	4049	8813	17231
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	2	3	29	49	136	516	806	2682
Maximale Anzahl Elemente	391	720	1620	2704	5588	10090	21900	44475	81214
Anzahl Datenpakete	6849	7167	6756	6827	9049	13920	23457	41581	84270
Gesamtweglänge aller Datenpaket	34608	36224	34628	35120	46604	67850	114075	196278	409104
Durchschn. Iterationstiefe	9.62	10.10	9.48	9.51	9.82	9.74	9.48	9.07	9.60

maximalen Werte werden wiederum über die zehnfach durchgeführte Messung gemittelt.

Die Grafik links unten auf dem Datenblatt gibt die Verteilung der Daten nach der Sortierung wieder. Während die Daten am Anfang gleichmäßig über alle Prozessoren verteilt waren, ist das nach der Sortierung nicht mehr der Fall. Die Menge der Daten, die sich am Ende auf einem Prozessor befinden kann, ist durch einen Balken dargestellt. Die Angaben erfolgen in Prozent, um eine Vergleichbarkeit mit den verschiedenen Datenmengen zu haben. Befindet sich nach der Sortierung mehr als die doppelte Menge an Daten auf einem Prozessor ($> 200\%$), dann ist über dem Balken die maximale Prozentzahl angegeben. Beispiel: Bei 32768 Elementen pro Prozessor liegt die Datenmenge nach der Sortierung zwischen 8% und 248%. Die genauen Werte berechnen sich aus den Meßwerten der unteren Tabelle auf dem Datenblatt.

Die Grafik rechts unten auf dem Datenblatt stellt das Verhältnis der Zeit für die lokale Sortierung der Zeit für die Kommunikation gegenüber. Die Zeit für die lokale Sortierung ist der Zeitanteil, in der alle Prozessoren voll ausgelastet sind und nicht miteinander kommunizieren müssen. Der Zeitanteil für die Kommunikation ist der zusätzliche Aufwand, der bei parallelen Algorithmen wegen der erforderlichen Kommunikation entsteht. Anhand dieser Grafik zeigt sich das Verhältnis dieser beiden Zeitanteile in Abhängigkeit der zu sortierenden Datenmenge. Die genauen Werte berechnen sich aus den Meßwerten der unteren Tabelle auf dem Datenblatt.

Die untere Tabelle auf dem Datenblatt enthält die wichtigsten Meßwerte für die verschiedenen Datenmengen:

- Gesamtzeit: Die Gesamtzeit ist die gemessene Zeit vom Anfang der Sortierung bis zum Zeitpunkt, an dem der letzte Prozessor seine Arbeit beendet hat.
- Zeit pro Element pro Prozessor: Als alternative und kürzere Bezeichnung wird hier auch der Begriff Sortierzeit verwendet. Die Sortierzeit ist die Gesamtzeit dividiert durch die Datenmenge pro Prozessor. Dieser Wert wird für die Grafik oben links auf dem Datenblatt verwendet.
- Zeit für Kommunikation: Während dieser Zeit werden Daten versandt und empfangen. Der Wert stammt von dem Prozessor, der die meiste Zeit für die Kommunikation aufgewendet hat.
- Zeit für lokale Sortierung: Während dieser Zeit werden die Daten auf dem Prozessor lokal sortiert. Der Wert stammt von dem Prozessor, der die meiste Zeit für die Sortierung benötigt hat.
- Zeit für Odd-Even-Sort: Wird Odd-Even-Sort eingesetzt, so werden die Meßwerte wie bei der Zeit für die lokale Sortierung bestimmt. Die Summe über die Zeit für Kommunikation, lokale Sortierung und Odd-Even-Sort übersteigt in der Regel die Gesamtzeit, weil die jeweiligen Maximalwerte von verschiedenen Prozessoren stammen können.

- Minimale Anzahl Elemente: Diese Zahl gibt an, wie viele Daten sich nach der Sortierung auf jedem Prozessor mindestens befinden. Die Grafik unten links auf dem Datenblatt basiert u.a. auf diesen Meßwerten.

- Maximale Anzahl Elemente: Diese Zahl gibt an, wie viele Daten sich nach der Sortierung auf jedem Prozessor maximal befinden. Die Grafik unten links auf dem Datenblatt basiert u.a. auf diesen Meßwerten.

- Anzahl Datenpakete: Diese Zahl gibt an, wie viele Datenpakete bei einem Sortiervorgang durchschnittlich verschickt werden.

- Gesamtweglänge: Die Summe der Entfernungen aller Datenpakete wird als Gesamtweglänge bezeichnet. Diese Gesamtweglänge in Verbindung mit der Anzahl der Datenpakete ist ein guter Indikator für die Belastung des Netzwerkes.

- Durchschnittliche Iterationstiefe: Durch die nicht immer gleichmäßige Aufteilung der Prozessoren braucht jeder Prozessor unterschiedlich viele Iterationen bis die Sortierung beendet ist. Nach einem Sortiervorgang wird die durchschnittliche Iterationstiefe aller Prozessoren ermittelt. Die durchschnittliche Iterationstiefe ist ein Indikator dafür, wie gleichmäßig die Prozessoren aufgeteilt werden.

Die Messungen wurden für jede Datenmenge zehnmal wiederholt. Alle angegebenen Meßwerte sind über diese zehn Messungen gemittelt worden.

4.4 Meßergebnisse

Die Ergebnisse jeder getesteten Konfiguration sind in Form von Datenblättern im Anhang zu finden.

Bei einem Vergleich der verschiedenen Numerierungen (Abbildung 4.6 und 4.7) zeigt sich, daß die Hilbertnumerierung immer besser abschneidet als die Zeilen- oder Schlangennumerierung. Die Kurvenverläufe bei dem 3-Median des Wurzelprozessors (Abbildung 4.6) sehen noch ziemlich zackig aus. Anhand der Tabelle 4.1 erkennt man deutlich, daß die Gesamtweglänge aller Datenpakete stark von der Numerierung abhängt.

Während die Anzahl der versandten Datenpakete bei allen Numerierungen mehr oder weniger gleich ist, ist die Gesamtweglänge bei der Hilbertnumerierung weitaus am kleinsten. Diese Reduzierung der Gesamtweglänge durch die Wahl einer besseren Numerierung läßt sich für alle Datenmengen beobachten.

Verbessert man die Pivotwahl, indem man den 3-Median durch den $\log(N)$-Median oder den $\sqrt{N}$-Median des Wurzelprozessors ersetzt (Abbildung 4.6 und 4.7), so zeigt sich deutlich, daß sich die Sortierzeiten weiter reduzieren lassen. Die angegebene Kommunikationszeit entspricht dabei der Sortierzeit ohne Berücksichtigung der lokalen Sortierung. Die

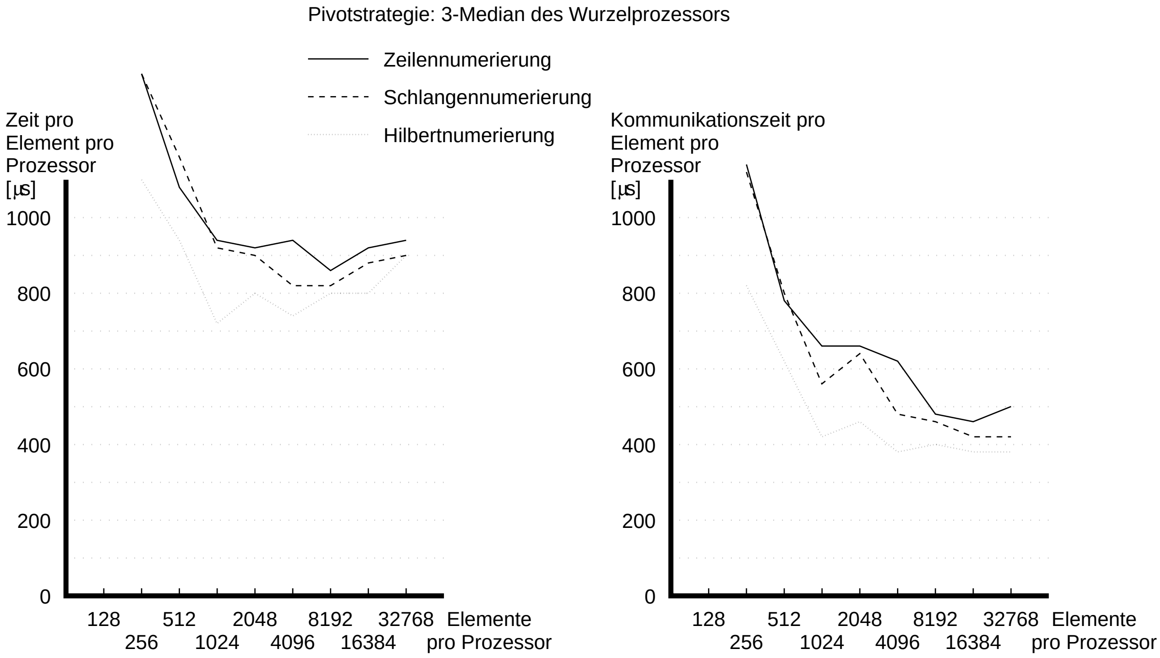


Abbildung 4.6: Sortier- und Kommunikationszeiten für verschiedene Numerierungen (Datenblätter 1 bis 3)

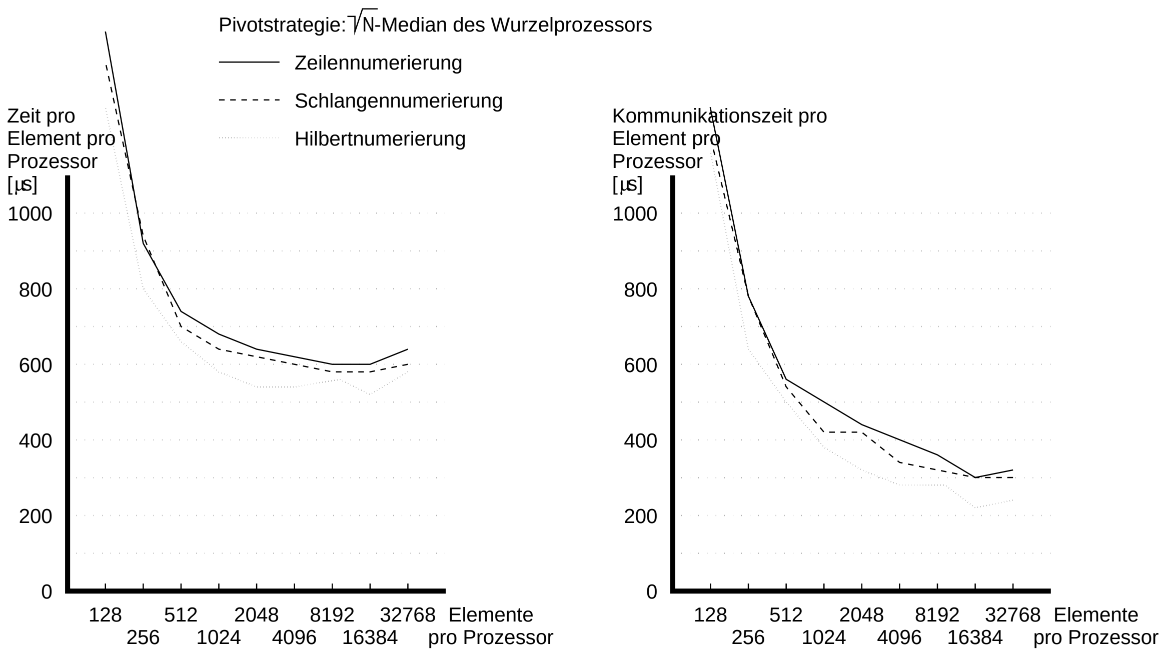


Abbildung 4.7: Sortier- und Kommunikationszeiten für verschiedene Numerierungen (Datenblätter 7 bis 9)

Datenmenge/Proz.	128	256	512	1024	2048	4096	8192	16384	32768
Zeilennumerierung									
Anzahl Datenpakete	6849	7167	6756	6827	9049	13920	23457	41581	84270
Gesamtweglänge	34608	36224	34628	35120	46604	67850	114075	196278	409104
Schlangennumerierung									
Anzahl Datenpakete	6804	7286	7042	6836	8896	13302	23046	40781	80270
Gesamtweglänge	30809	32701	32638	31290	41059	57847	98459	171073	337330
Hilbertnumerierung									
Anzahl Datenpakete	6891	7122	6722	6960	8844	13339	23220	42702	82651
Gesamtweglänge	27368	28037	27440	27985	34109	49197	84703	154494	296643

Tabelle 4.1: Datenpakete und Gesamtweglänge (Datenblätter 7 bis 9)

Wirkung der verbesserten Pivotstrategien lassen sich gut an der durchschnittlichen Iterationstiefe ablesen. Im Idealfall, d.h. wenn ein Segment immer genau halbiert wird, kommt man bei 256 Prozessoren mit exakt 8,00 Iterationen aus. In Tabelle 4.2 zeigt sich deutlich, daß der $\sqrt{N}$-Median bei großen Datenmengen schon ziemlich gut an die optimale Iterationstiefe herankommt.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
3-Median des Wurzelproz.	9.68	10.03	9.43	9.68	9.64	9.42	9.41	9.34	9.41
$\log(N)$-Median des Wurzelproz.	8.72	8.76	8.47	8.69	8.44	8.38	8.38	8.42	8.40
$\sqrt{N}$-Median des Wurzelproz.	8.75	8.40	8.40	8.28	8.22	8.13	8.13	8.07	8.07

Tabelle 4.2: Durchschnittliche Iterationstiefen bei der Hilbertnumerierung (Datenblätter 3, 6 und 9)

Der Verlauf der in Abbildung 4.7 dargestellten Kurvenverläufe läßt sich wie folgt erklären: Die Gesamtzeit der Sortierung setzt sich aus der Zeit für die lokale Sortierung und aus der Zeit für die Kommunikation zusammen. Bei einer Verdopplung der Datenmenge werden bei der Kommunikation zwar doppelt soviele Daten versandt, dennoch verdoppelt sich der Zeitaufwand nicht ganz. Die Verdopplung der Datenmenge führt hingegen bei der lokalen Sortierung zu mehr als zu einer Verdopplung, weil der Aufwand der Sortierung $O(N \log(N))$ ist. Bis zu einer gewissen Datenmenge kompensiert also der sublineare Zeitanstieg der Kommunikation den überlinearen Zeitanstieg der lokalen Sortierung.

In jeder Iteration werden die Segmente, in denen die Daten ausgetauscht werden, kleiner. Dementsprechend sollte auch der Zeitbedarf für die Kommunikation in jeder Iteration kleiner werden. Daß dem nicht immer so ist, läßt sich ebenfalls an den Meßergebnissen im Anhang (Datenblätter 1 bis 9) erkennen. In manchen Fällen bleiben die Zeiten für die Kommunikation konstant bzw. steigen sogar. Die Ursache dafür ist die Form des Segments. Liegen die Prozessoren ungünstig zueinander, oder haben sie einen Sprung vom Ende der einen Zeile zum Anfang der nächsten[2], so kann der Aufwand für die Kommunikation auch wieder ansteigen. Durch die Wahl einer geeigneten Numerierung für die Prozessoren kann man somit Zeit gewinnen.

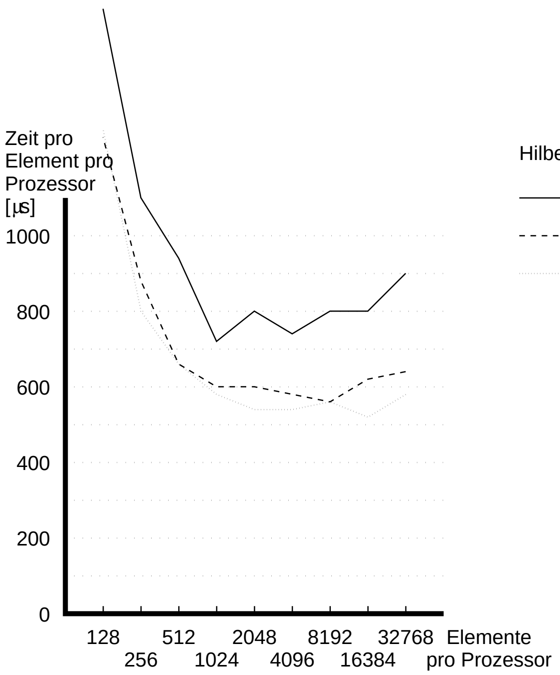


Abbildung 4.8: Sortierzeiten für verschiedene Pivotstrategien (Datenblätter 3, 6 und 9)

Bei der Pivotstrategie erweist sich der $\sqrt{N}$-Median des Wurzelprozessors erwartungsgemäß als die bislang beste Pivotstrategie (Abbildung 4.8). Der Aufwand, der in die Wahl eines guten Pivotelements gesteckt wird, macht sich folglich bezahlt.

Allerdings ist die Verteilung der Daten nach der Sortierung ziemlich unbefriedigend. So kann es bei der schlechtesten Pivotstrategie vorkommen, daß ein Prozessor am Ende die dreifache Menge an Daten besitzt, während andere Prozessoren fast keine Daten mehr besitzen. Mit dem $\log(N)$-Median oder dem $\sqrt{N}$-Median erreicht man zumindest bei großen Datenmengen eine bessere Verteilung der Daten über die Prozessoren. Dennoch kann die Datenmenge nach der Sortierung immer noch doppelt so hoch sein wie zu Beginn. Da die lokale Sortierung am Ende durchgeführt wird, wenn die Daten nicht mehr gleichmäßig verteilt sind, führt die doppelt so große Datenmenge zu mehr als zu einer Verdopplung der lokalen Sortierzeit.

[2]nur bei Zeilennumerierung möglich

Kapitel 5

Optimierungen für das parallele Quicksort

An den einzelnen Komponenten des parallelen Quicksorts können verschiedene Verbesserungen vorgenommen werden.

5.1 Pivotwahl

Während die Pivotwahl bisher lokal auf dem Wurzelprozessor vorgenommen wurde, bietet es sich für den praktischen Einsatz an, bei der Pivotwahl alle Prozessoren gleichberechtigt zu behandeln. Eine Möglichkeit besteht darin, daß jeder Prozessor lokal ein Pivotelement ermittelt, die als Kandidaten über die kollektive *reduce*-Operation mit der Verknüpfungsoperation Merge eingesammelt werden. Beim Wurzelprozessor kommt somit eine sortierte Folge der Pivotkandidaten an, aus der der Wurzelprozessor sehr einfach den Median bestimmen kann. Der Mehraufwand bei der Pivotwahl beträgt eine kollektive Operation.

- Median der 3-Mediane: Der Wurzelprozessor ermittelt den Median aus den 3-Medianen aller Prozessoren.
- Median der $\log(N)$-Mediane: Der Wurzelprozessor ermittelt den Median aus den $\log(N)$-Medianen aller Prozessoren.
- Median der $\sqrt{N}$-Mediane: Der Wurzelprozessor ermittelt den Median aus den $\sqrt{N}$-Medianen aller Prozessoren.

Hierbei ist zu beachten, daß die Segmentgröße bei den globalen Pivotstrategien einen Einfluß auf die Pivotwahl hat. Welche Pivotstrategie die besten Ergebnisse liefert, hängt davon ab, wie groß die Datenmenge ist. Entscheidend für die Gesamtlaufzeit ist hierbei nicht die Zeit für die Wahl des Pivotelements, sondern dessen Güte, die sich in einer gleichmäßigen Verteilung der Daten bemerkbar macht.

Nach der Bestimmung des Pivotelements wird dieses vom Wurzelprozessor durch eine *broadcast*-Operation an alle Prozessoren im Segment mitgeteilt.

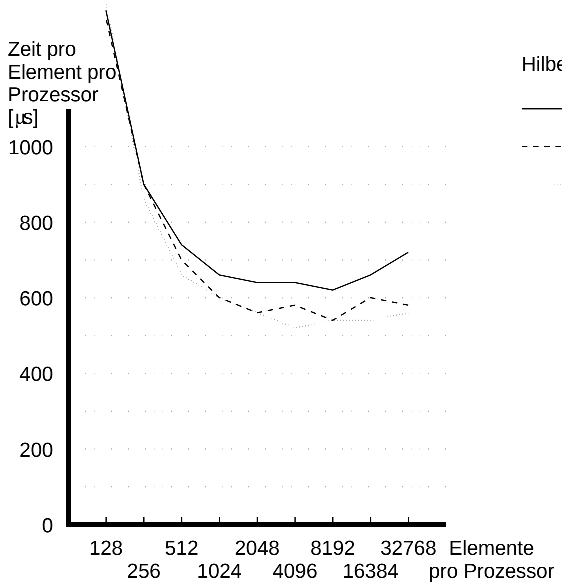


Abbildung 5.1: Meßwerte für verschiedene Pivotstrategien (Datenblätter 10, 11 und 14)

Die Meßergebnisse machen deutlich, daß bei der bislang besten Numerierung (Hilbertnumerierung), der Median der $\sqrt{N}$-Mediane die beste Pivotstrategie darstellt (Abbildung 5.1). Im Vergleich zum $\sqrt{N}$-Median des Wurzelprozessors ergibt sich aber kein nennenswerter Gewinn bei den Sortierzeiten. Dennoch läßt sich feststellen, daß sich der Mehraufwand für die Ermittlung eines Pivotelements über alle Prozessoren hinweg zumindest nicht negativ auswirkt. Aber wenn sich die globale Pivotstrategie bei den Sortierzeiten schon nicht positiv bemerkbar macht, so ergeben sich dennoch zwei Vorteile: Erstens ist es für den praktischen Einsatz wünschenswert, daß eine faire Pivotstrategie eingesetzt wird, die alle Prozessoren gleichberechtigt betrachtet. Zweitens verbessert sich die durchschnittliche Iterationstiefe erheblich, wie an der Tabelle 5.1 zu sehen ist.

Der ideale Wert für die durchschnittlichen Iterationstiefe bei 256 Prozessoren, der maximal erreicht werden könnte, ist 8,00. Mit 8,02 bei großen Datenmengen kommt der Median der $\sqrt{N}$-Mediane diesem Ziel schon sehr nahe. Da sich diese Pivotstrategie eindeutig als die beste erwiesen hat, wird zukünftig nur noch diese Strategie angewendet.

Betrachtet man bei dieser Pivotstrategie wiederum verschiedene Numerierungen (Abbildung 5.2), so ergibt sich die bekannte Reihenfolge: Die Schlangennumerierung ist besser als die Zeilennumerierung, aber beide werden von der Hilbertnumerierung übertroffen. Anhand der Tabelle 5.2 wird deutlich, woher dieser Gewinn bei den Sortierzeiten kommt.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
$\sqrt{N}$-Median des Wurzelprozessors	8.75	8.40	8.40	8.28	8.22	8.13	8.13	8.07	8.07
Median der $\sqrt{N}$-Mediane	8.19	8.10	8.10	8.06	8.07	8.03	8.03	8.02	8.02

Tabelle 5.1: Durchschnittliche Iterationstiefen bei der Hilbertnumerierung (Datenblätter 9 und 14)

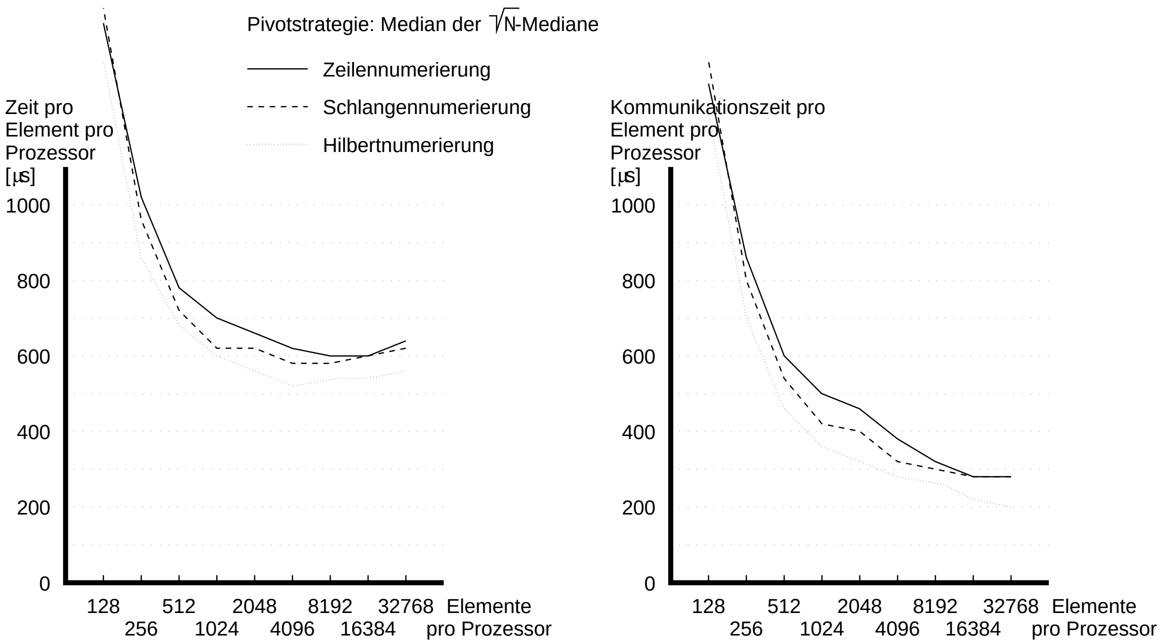


Abbildung 5.2: Sortier- und Kommunikationszeiten für verschiedene Numerierungen (Datenblätter 12 bis 14)

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Zeilennumerierung mit Median der $\sqrt{N}$-Mediane									
Anzahl Datenpakete	5764	5692	5706	5670	7278	11655	20062	36818	70216
Gesamtweglänge	29686	30942	30967	30143	38225	57784	96999	164197	330664
Schlangennumerierung mit Median der $\sqrt{N}$-Mediane									
Anzahl Datenpakete	5769	5696	5706	5674	7246	11692	20037	36813	70185
Gesamtweglänge	27277	26706	26779	26880	32974	54059	91350	168108	317981
Hilbertnumerierung mit Median der $\sqrt{N}$-Mediane									
Anzahl Datenpakete	5757	5698	5699	5673	7214	11727	20075	36834	70184
Gesamtweglänge	23968	24315	24404	24358	29534	48470	82400	151431	286745

Tabelle 5.2: Datenpakete und Gesamtweglänge für verschiedene Numerierungen (Datenblätter 12 bis 14)

Die Anzahl der versandten Datenpakete ist bei allen Numerierungen nur ungefähr gleich, weil unterschiedliche Nachrichtenlaufzeiten zu einer nicht deterministischen Wahl der Pivotelemente führen. Die Unterschiede können jedoch als statistische Schwankungen betrachtet werden. Bei den Gesamtweglängen erweist sich die Hilbertnumerierung als überlegen.

5.2 Lokale Sortierung

Der Zeitaufwand für die lokale Sortierung wird einerseits durch das eingesetzte Sortierverfahren bestimmt, andererseits durch die Anzahl der zu sortierenden Elemente. Da als lokales Sortierverfahren schon sequentielles Quicksort eingesetzt wird, ist hier kaum eine Verbesserung mehr möglich. Von größerer Bedeutung ist aber die Anzahl der zu sortierenden Elemente pro Prozessor. Durch die iterative Aufteilung der Elemente in zwei Teilsegmente ergibt sich eine ungleichmäßige Verteilung der Daten.

Beispiel: Besteht ein Segment nur noch aus drei Prozessoren und teilt ein gutes Pivotelement die Datenelemente fast gleichmäßig auf, so müßte jedes Segment ca. 1,5 Prozessoren erhalten. Da dies nicht möglich ist, bekommt ein Segment zwei Prozessoren, das andere einen Prozessor. Wenn die Hälfte der Daten nur einem der drei Prozessoren zugeordnet wird, so hat dieser Prozessor doppelt soviele Elemente zu sortieren wie die beiden anderen Prozessoren. Somit steigt die lokale Sortierzeit des einzelnen Prozessors stark an und bestimmt die Gesamtlaufzeit des Algorithmus. Diese ungünstige Aufteilung der Prozessoren ergibt sich hauptsächlich bei kleinen Segmenten. Sobald die Segmentgröße unter eine bestimmte Mindestgröße fällt, bietet es sich deshalb an, mit Quicksort aufzuhören und zu einem anderen Algorithmus überzugehen, der bei kleinen Segmenten bessere Sortierzeiten liefert.

Eines dieser Sortierverfahren ist Odd-Even-Sort[1]. Bei Odd-Even-Sort werden die Daten zuerst lokal sortiert. Anschließend erfolgt alternierend ein Datenaustausch mit dem rechten bzw. linken Partner, wobei der Prozessor mit der kleineren (bzw. größeren) Prozessornummer die Hälfte der Daten mit den kleineren (bzw. größeren) Elementen behält.

Ab welcher Segmentgrenze das Odd-Even-Sort einen Vorteil bringt, kann durch Tests ermittelt werden. Als Konfiguration wird die beste Medianstrategie (Median der $\sqrt{N}$-Mediane) und die bislang beste Numerierung (Hilbertnumerierung) gewählt.

Datenmenge/Proz.	128	256	512	1024	2048	4096	8192	16384	32768
ohne Odd-Even-Sort									
Sortierzeiten [μs]	1390	870	671	594	561	529	534	544	567
Gesamtweglänge	23968	24315	24404	24358	29534	48470	82400	151431	286745
Odd-Even-Sort für 2 Prozessoren									
Sortierzeiten [μs]	1280	847	644	553	545	536	537	546	557
Gesamtweglänge	25299	25741	25855	26657	33469	56437	97230	182348	346793
Odd-Even-Sort für 3 Prozessoren									
Sortierzeiten [μs]	1320	823	630	539	521	500	494	490	501
Gesamtweglänge	26655	26854	26897	28134	35910	58494	101810	185917	355964
Odd-Even-Sort für 4 Prozessoren									
Sortierzeiten [μs]	1312	855	632	547	527	505	497	490	510
Gesamtweglänge	27711	28354	28372	31361	41090	71718	126290	241762	463534
Odd-Even-Sort für 5 Prozessoren									
Sortierzeiten [μs]	1328	835	632	555	535	509	494	489	499
Gesamtweglänge	28859	29526	29626	32754	43916	74510	131764	246694	474808

Tabelle 5.3: Sortierzeiten und Gesamtweglängen für verschiedene Odd-Even-Sort-Strategien (Datenblätter 14 und 17 bis 20)

Die Tabelle 5.3 zeigt die Sortierzeiten ohne Odd-Even-Sort, sowie mit Odd-Even-Sort für zwei, drei, vier und fünf Prozessoren. In dieser Konfiguration läßt sich nicht abschätzen, für welche Segmentgröße Odd-Even-Sort letztendlich die besten Ergebnisse liefert. Die Sortierzeiten für drei bis fünf Prozessoren sehen ziemlich ähnlich aus. Im Idealfall, wenn alle Segmente exakt halbiert werden, treten nur Zweierpotenzen als Segmentgrößen auf. Obwohl dieser Idealfall hier nicht erreicht wird, liegen nur wenige Segmentgrößen dicht neben den Zweierpotenzen. Die Segmentgröße 6 tritt kaum auf, so daß die Meßwerte denen von Odd-Even-Sort mit fünf Prozessoren entsprechen. Eine signifikante Veränderung der Meßwerte stellt man erst wieder fest, wenn man die Segmentgröße 8 wählt. Die sich durch die nicht exakte Halbierung ergebende Streuung läßt sich voraussichtlich mit der Segmentgröße 10 auffangen. Die nächste nennenswerte Veränderung der Meßwerte ist danach erst wieder bei einer Segmentgröße von 16 zu erwarten.

Bei den kleinen Segmentgrößen wird schon deutlich, daß die Gesamtweglänge aller versandten Datenpakete durch den Austausch der Daten bei jeder Odd-Even-Sort-Iteration

[1]Der genaue Algorithmus ist in [KGGK94] und [UmVo94] beschrieben.

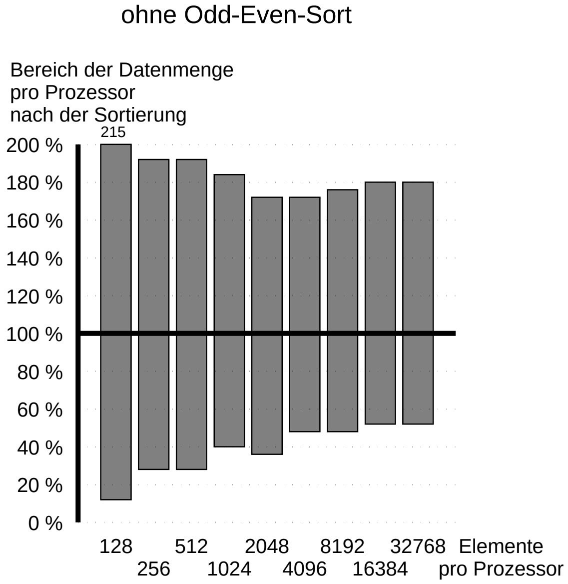


Abbildung 5.3: Verteilung der Daten nach der Sortierung ohne Odd-Even-Sort (Datenblatt 14)

stark ansteigt. Dieser Nachteil wird dadurch kompensiert, daß die Verteilung der Daten nach der Sortierung erheblich besser ist, wie man an den Abbildungen 5.3 und 5.4 deutlich sehen kann. Durch die gleichmäßigere Verteilung der Daten sinkt dann die Zeit für die lokale Sortierung. Bei größeren Segmenten ergibt sich keine signifikante Verbesserung der Verteilung der Daten nach der Sortierung, so daß die Sortierzeiten dann insgesamt steigen.

Odd-Even-Sort bringt unabhängig von der Numerierung einen Zeitgewinn und eine bessere Verteilung der Daten nach der Sortierung. Allerdings kann man nicht sagen, ab welcher Segmentgröße sich der Einsatz von Odd-Even-Sort lohnt, denn der Erfolg von Odd-Even-Sort ist abhängig von der jeweiligen Konfiguration. Deshalb wird zunächst einmal auf den weiteren Einsatz von Odd-Even-Sort verzichtet, um weitere Techniken zur Optimierung der Sortierzeiten unabhängig von Odd-Even-Sort zu erproben. Ist die beste Konfiguration dann ermittelt worden, kann diese mit Odd-Even-Sort noch weiter verbessert werden.

5.3 Datenaustausch

Die Organisation des Datenaustausches bietet die meisten Möglichkeiten zur Optimierung. An einem Beispiel mit 16 Prozessoren sollen die verschiedenen Optimierungstechniken demonstriert werden.

Wie der Datenaustausch normalerweise vorgenommen wird, wurde schon in Kapitel 4.2.4

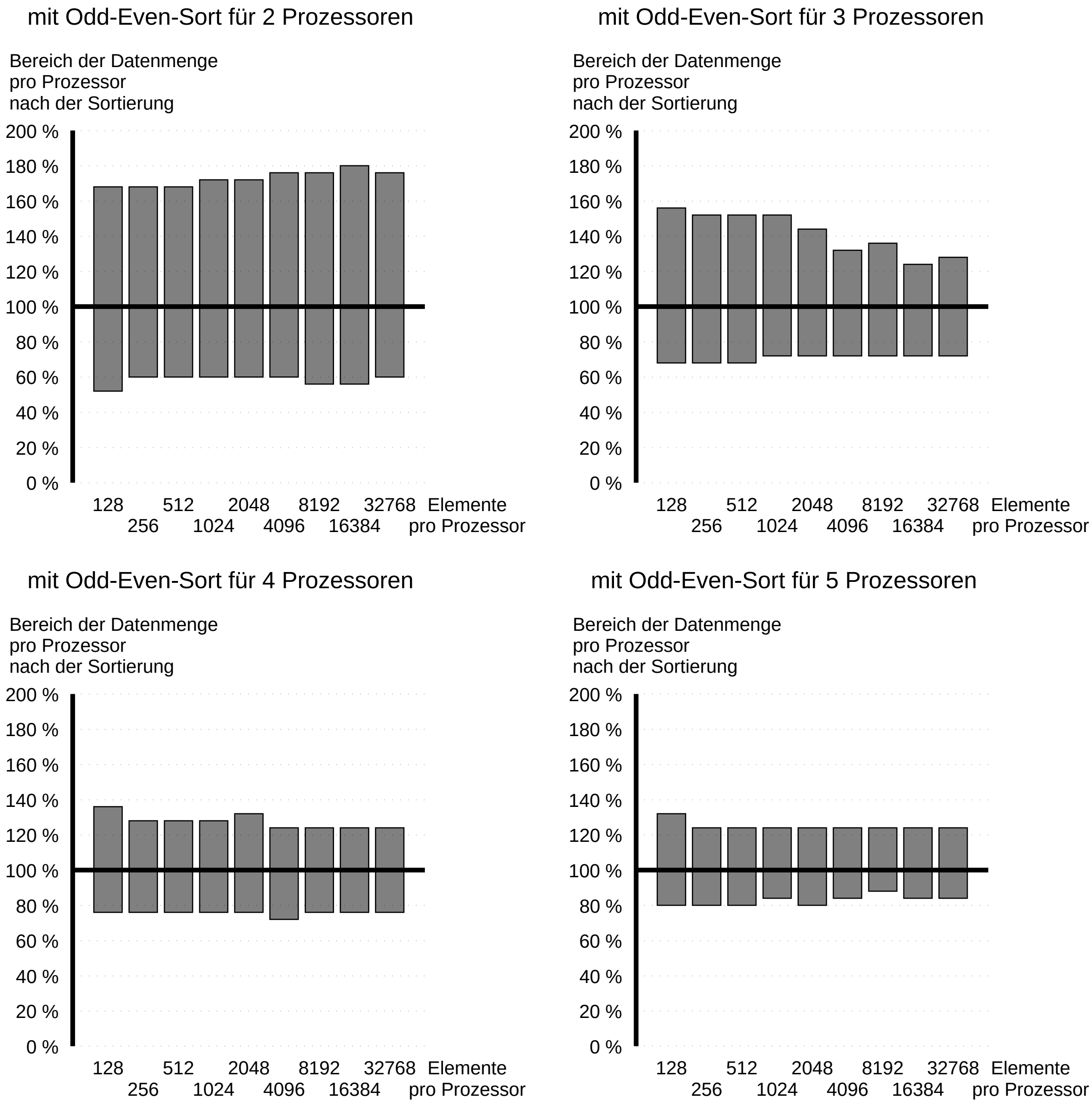


Abbildung 5.4: Verteilung der Daten nach der Sortierung mit Odd-Even-Sort (Datenblätter 17 bis 20)

erläutert. Zur anschaulichen Darstellung der Optimierungstechniken werden die Prozessoren zuerst immer als 1-dimensionale Reihe angeordnet. Anschließend werden die Kommunikationswege in das 2-dimensionale Gitter mit 16 Prozessoren eingetragen. Hier kommen dann die verschiedenen Prozessornumerierungen (Zeilen-, Schlangen- und Hilbertnumerierung) zum Tragen. In Kapitel 5.4 werden dann weitere Numerierungen eingeführt.

Die Datenmengen, die über jeden einzelnen Kommunikationsweg transportiert werden, sind aus den Abbildungen allerdings nicht zu erkennen.

Optimierungen beim Datenaustausch zielen darauf ab, die folgenden vier Ziele zu erreichen:

1. Die zu transportierende Datenmenge sollte so klein wie möglich sein.
2. Die Gesamtzahl der Nachrichten sollte möglichst gering sein.
3. Die Prozessoren sollten so angeordnet sein, daß die Wege zwischen den kommunizierenden Prozessoren möglichst kurz sind.
4. Die Wege der zu versendenden Datenpakete sollten sich so wenig wie möglich überschneiden.

Bei der Betrachtung der Kommunikationswege in Abbildung 4.4 fällt auf, daß es zwei verschiedene Arten von Kommunikationswegen gibt: Erstens Kommunikationswege von einem Segment in das andere, zweitens Kommunikationswege innerhalb des Segments. Bei den Kommunikationswegen innerhalb eines Segments ist es aber so, daß Daten, die auf Prozessoren im richtigen Segment liegen, an andere Prozessoren im gleichen Segment verschickt werden. Diese Kommunikation ist vermeidbar und liefert somit den ersten Ansatz für eine Optimierung.

5.3.1 Reduzierung des Datenaustausches

Jeder Prozessor bekommt beim Datenaustausch Elemente zugeschickt, die entweder kleiner oder gleich bzw. größer dem Pivotelement sind. Die eigenen Daten sind im Durchschnitt zur Hälfte ebenfalls kleiner oder gleich bzw. größer dem Pivotelement. Deshalb bietet es sich an, daß jeder Prozessor den Teil seiner Daten behält, der schon im richtigen Segment liegt. Da es sich hierbei im Durchschnitt um die Hälfte der Daten handelt, kann die Datenmenge, die insgesamt über das Gitter verschickt werden muß, halbiert werden. Damit erreicht man zwar eine merkliche Reduzierung der Kommunikationszeit, aber keine Halbierung, weil die wegfallenden Kommunikationswege in der Regel die kürzeren sind.

Wenn jetzt ein Teil der Elemente nicht am Datenaustausch teilnimmt, muß die Datenaustauschphase neu gestaltet werden. Es ist mehr Organisationsaufwand nötig, was zu einer zusätzlichen kollektiven Operation führt. Die Abbildung 5.5 zeigt das Beispiel mit reduzierter Kommunikation.

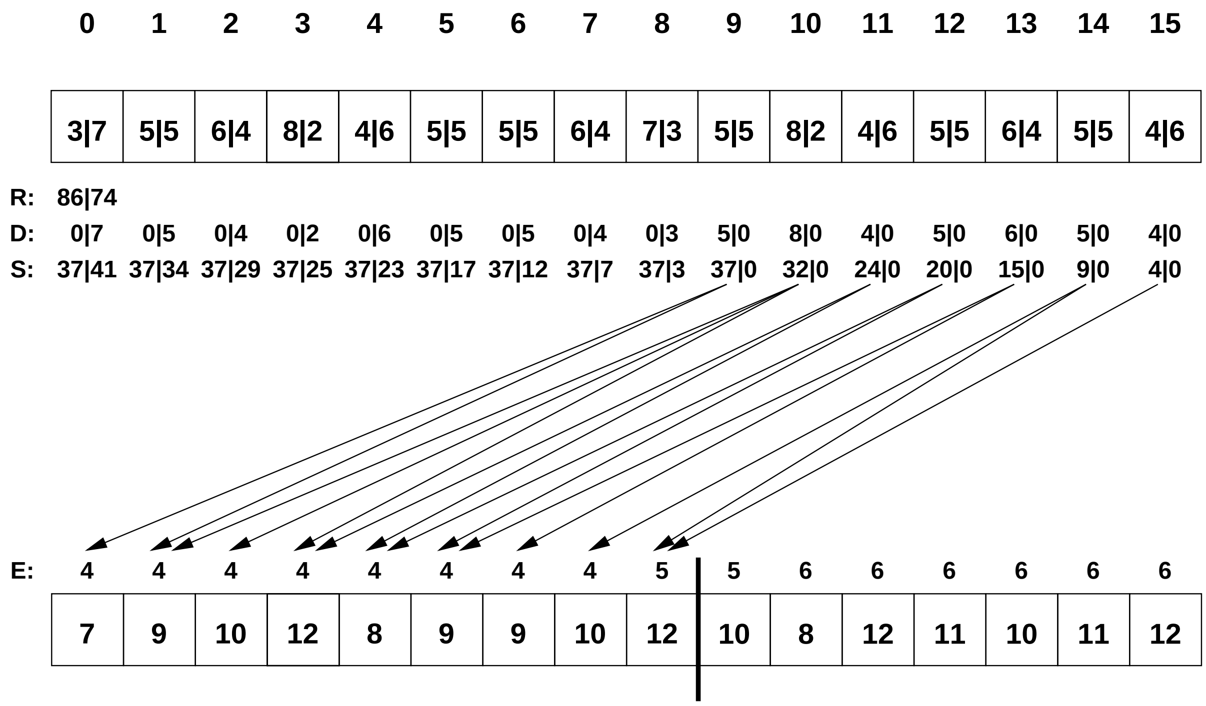


Abbildung 5.5: Datenaustausch bei reduzierter Kommunikation

Der Ablauf des Datenaustausches sieht wie folgt aus:

- Jeder Prozessor ermittelt, wie viele seiner Elemente kleiner oder gleich bzw. größer dem Pivotelement sind. Das Ergebnis ist für jeden Prozessor innerhalb des Kästchens ersichtlich.

- Es wird eine *reduce*-Operation mit Vektoraddition über die Anzahl der Elemente kleiner oder gleich und größer dem Pivotelement durchgeführt. Das Ergebnis ist in der Grafik in der Zeile R abzulesen.

- Das Ergebnis dieser *reduce*-Operation wird durch eine *broadcast*-Operation allen anderen Prozessoren im Segment mitgeteilt.

- Jeder Prozessor kann jetzt selbst berechnen, zu welchem Segment er gehören wird und welche seiner Daten er behalten kann. Für die Daten, die er behalten darf, setzt er seine Anzahl auf 0. In der Grafik sind diese Werte für jeden Prozessor in der Zeile D dargestellt.

- Auf diesen Daten der einzelnen Prozessoren wird jetzt die bekannte *scan*-Operation mit Vektoraddition ausgeführt. Das Ergebnis ist in der Zeile S dargestellt.

- Nun kann der eigentliche Datenaustausch beginnen. Anhand seiner vorliegenden Werte kann jeder Prozessor selbst berechnen, wie viele Daten er zu erwarten hat. Diese Anzahl ist in der Zeile E dargestellt. Addiert man zu den empfangenen Daten die behaltenen Daten hinzu, so erhält man die Anzahl der Daten pro Prozessor, die in den Kästchen angegeben ist.

Gegenüber der vollständigen Kommunikation, bei der 86 + 74 Elemente verschickt wurden, sind es bei reduzierter Kommunikation nur 37 + 41 Elemente.

Nachteilig an dieser Vorgehensweise ist, daß man erstens eine zusätzliche kollektive Operation benötigt und zweitens, daß die Daten nach dem Datenaustausch nicht mehr so gleichmäßig im Segment verteilt sind. Allerdings kann man bzgl. der Verteilung folgende Zusicherung machen:

- Die Summe der Elemente pro Segment ist bei reduzierter Kommunikation im Vergleich zur vollständigen Kommunikation nach jeder Iteration identisch.

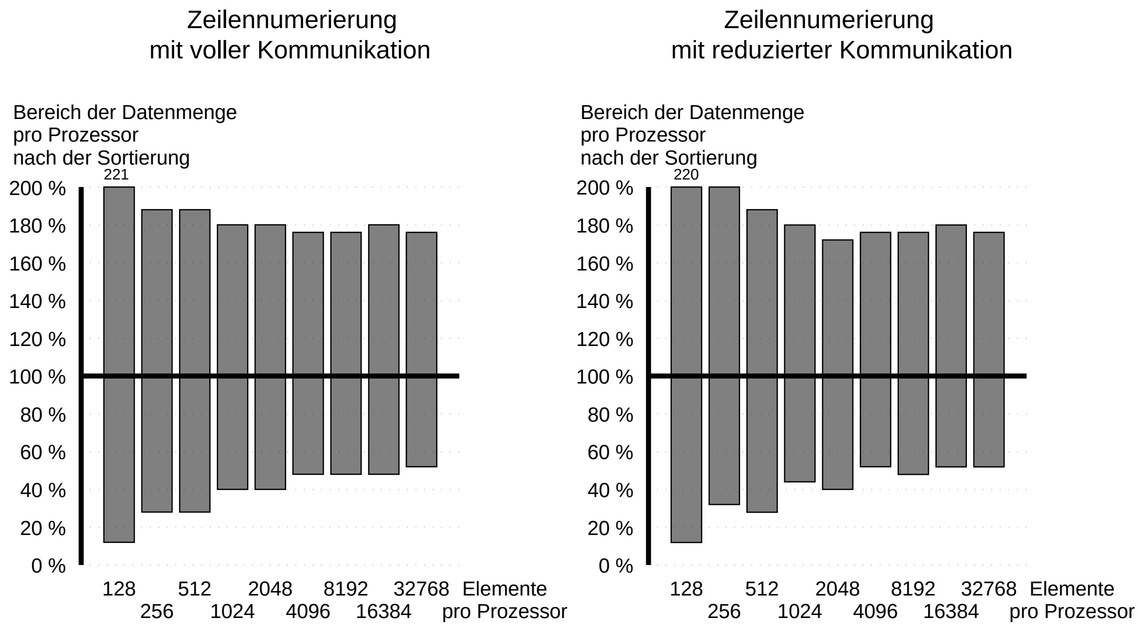


Abbildung 5.6: Verteilung der Daten bei voller und reduzierter Kommunikation mit Zeilennumerierung (Datenblätter 12 und 21)

Geht man davon aus, daß die Pivotwahl nicht durch die Verteilung der Daten beeinflußt wird, so ergibt sich letztendlich in den Segmenten mit nur einem Prozessor die gleiche Verteilung als wenn bei der Kommunikation alle Daten verschickt worden wären. Bei einem Vergleich der Verteilung der Daten bei voller und reduzierter Kommunikation (Abbildung 5.6) zeigt sich dementsprechend kaum ein Unterschied.

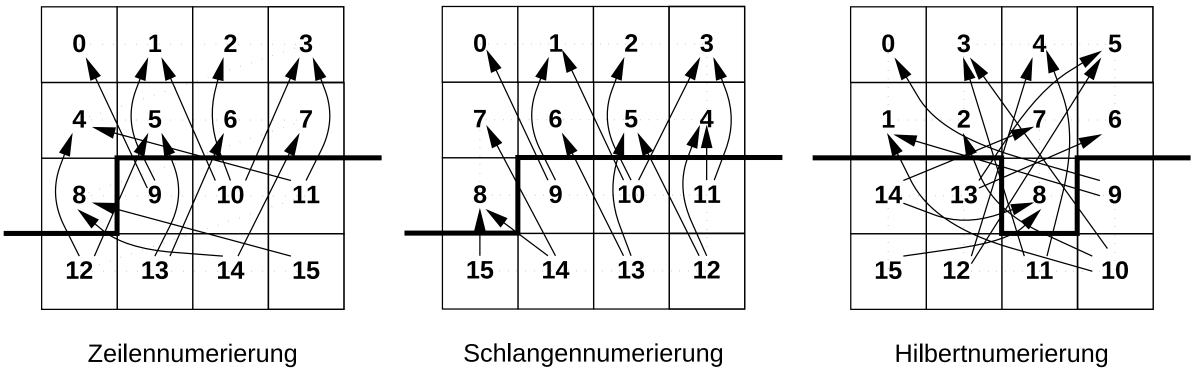


Abbildung 5.7: Datenaustausch bei reduzierter Kommunikation

Bei der 2-dimensionalen Darstellung in Abbildung 5.7 zeigt sich bei der Zeilennumerierung und besonders bei der Schlangennumerierung, daß sich nicht mehr soviele Kommunikationswege überschneiden. Dabei fällt auch auf, daß viele Kommunikationswege schon recht kurz sind und nur um zwei Prozessoren nach oben führen. Hingegen scheinen sich bei der Hilbertnumerierung alle Kommunikationswege in der Mitte des Gitters zu überschneiden. Beide Beobachtungen stellen keinen Zufall dar und werden deshalb später noch näher betrachtet.

An den Meßergebnissen zeigt sich deutlich, daß die Kommunikationszeiten bei großen Datenmengen durch die Reduzierung der Kommunikation fast halbiert werden. Bei kleineren Datenmengen hingegen macht sich der Mehraufwand der einen kollektiven Operation pro Iteration geringfügig bemerkbar, so daß dort die Zeiten leicht ansteigen. Abhängig von der Numerierung kann man im besten Fall zwischen 35% und 45% der Kommunikationszeit einsparen. Auf die Gesamtsortierzeit macht sich das durch einen Zeitgewinn von fast 20% bemerkbar.

Interessant ist der Verlauf der Kurven für die Zeit beim Datenaustausch bei 32768 Elementen pro Prozessor und Zeilennumerierung (Abbildung 5.8). Bei voller Kommunikation scheint der Kurvenverlauf nach ca. vier Iterationen auf einem Niveau zu verweilen, um dann weiter zu fallen. Bei reduzierter Kommunikation liegt an dieser Stelle ein lokales Minimum vor. Die Begründung für beide Kurvenverläufe ist aber identisch: Im Idealfall besteht bei 256 Prozessoren ein Segment in der vierten Iteration aus zwei Zeilen mit je 16 Prozessoren. Die Daten werden zwischen der einen und der anderen Zeile ausgetauscht. Die kommunizierenden Prozessoren liegen dann bei der Zeilennumerierung ziemlich nahe beieinander. In der fünften Iteration ist im Idealfall nur noch eine Zeile mit 16 Prozessoren vorhanden. Die Kommunikation erfolgt dann innerhalb dieser Prozessorzeile und führt aufgrund der größeren Entfernungen und kollidierenden Nachrichten zu einem erheblichen Zeitanstieg, obwohl sich die Segmentgröße halbiert hat.

Bei der Schlangennumerierung (Abbildung 5.9) zeigen sich ähnliche Kurvenverläufe. Al-

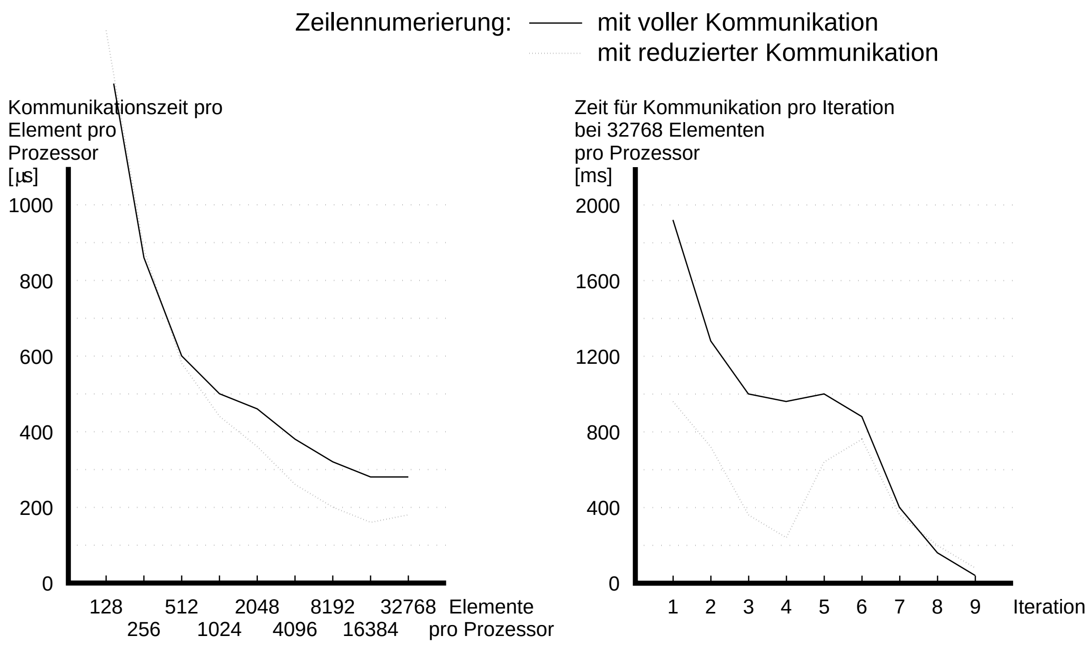


Abbildung 5.8: Kommunikationszeit und Zeit pro Iteration mit Zeilennumerierung (Datenblätter 12 und 21)

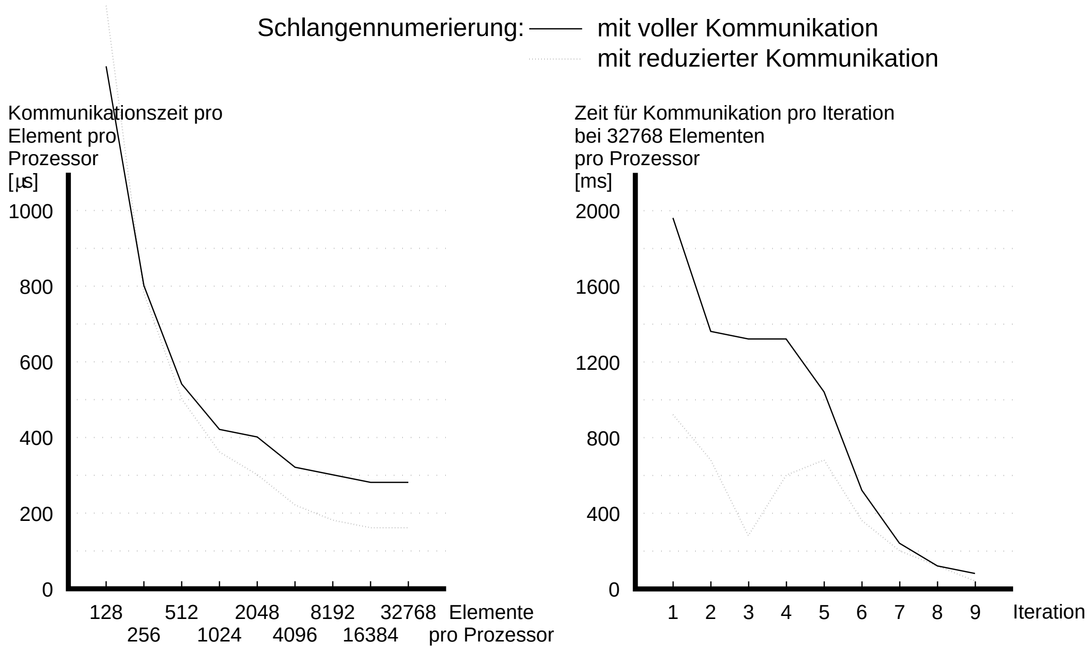


Abbildung 5.9: Kommunikationszeit und Zeit pro Iteration mit Schlangennumerierung (Datenblätter 13 und 22)

lerdings liegt das konstante Niveau bzw. das lokale Minimum hier nicht bei der vierten Iteration, sondern schon bei der dritten. Die Begründung ist für beide Kurvenverläufe identisch: Wenn ein Segment im Idealfall in der vierten Iteration nur noch aus zwei Prozessorzeilen besteht, so sind diese bei der Zeilennumerierung in die gleiche Richtung durchnumeriert, und die kommunizierenden Prozessoren liegen meistens übereinander. Bei der Schlangennumerierung hingegen ist die Numerierung der Prozessoren in der zweiten Zeile gegenläufig zu der in der ersten. Somit liegen kommunizierende Prozessoren nicht mehr übereinander, sondern die Kommunikationswege führen von der rechten Hälfte einer Zeile in die linke Hälfte der anderen Zeile und umgekehrt. Im ungünstigsten Fall findet eine Kommunikation vom Ende einer Zeile zum Anfang der nächsten statt. Aus diesem Grund steigt bei der Schlangennumerierung die Kommunikationszeit in der vierten Iteration schon wieder an, obwohl die Segmentgröße sich halbiert hat.

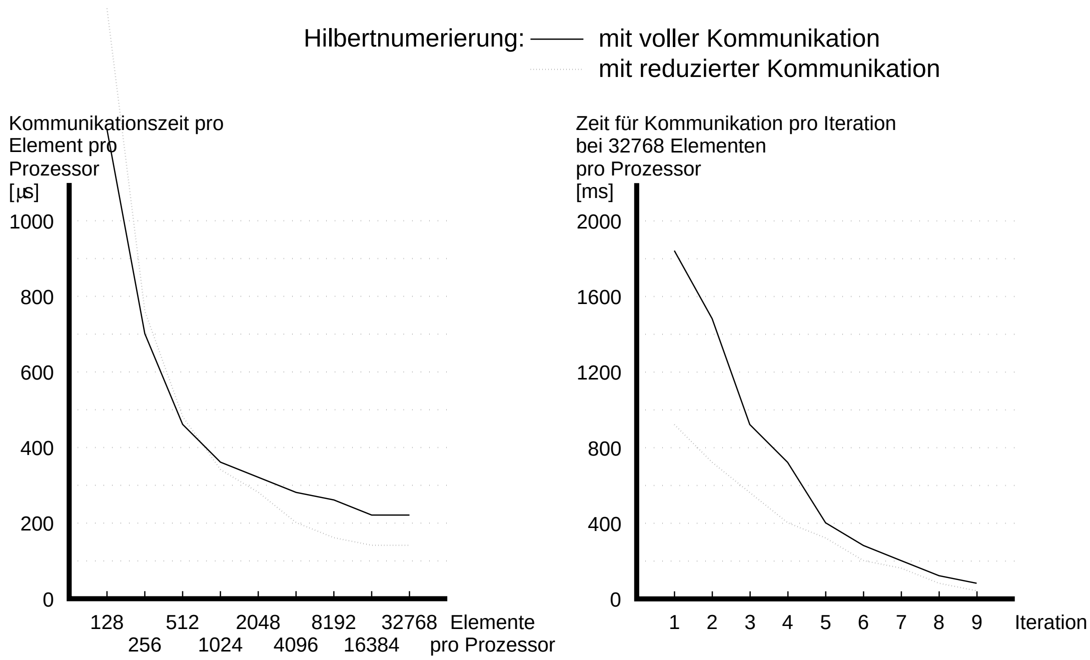


Abbildung 5.10: Kommunikationszeit und Zeit pro Iteration mit Hilbertnumerierung (Datenblätter 14 und 23)

Für die Hilbertnumerierung (Abbildung 5.10) ergeben sich für die Zeiten pro Iteration gleichmäßige Kurvenverläufe, weil im Idealfall ein Segment nach zwei Iterationen immer wieder aus einem Quadrat besteht. Die Entfernungen für die Kommunikationswege innerhalb dieser Segmente werden somit kontinuierlich kleiner. Gerade dieses Verhalten ist ein charakteristischer Vorteil der Hilbertnumerierung.

Die Verteilung der Daten nach der Sortierung hat sich für alle Numerierungen nur unwesentlich geändert, wie man an den Datenblättern 12 bis 14 und 21 bis 23 im Anhang erkennen kann. Interessant ist an dieser Stelle noch ein Vergleich der versandten Daten-

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Zeilennumerierung mit voller Kommunikation									
Anzahl Datenpakete	5764	5692	5706	5670	7278	11655	20062	36818	70216
Gesamtweglänge	29686	30942	30967	30143	38225	57784	96999	164197	330664
Zeilennumerierung mit reduzierter Kommunikation									
Anzahl Datenpakete	3663	3620	3631	3606	3919	6175	10323	18824	35498
Gesamtweglänge	26629	22662	24765	23142	24967	34219	55362	89271	183908

Tabelle 5.4: Datenpakete und Gesamtweglänge bei der Zeilennumerierung (Datenblätter 12 und 21)

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Schlangennumerierung mit voller Kommunikation									
Anzahl Datenpakete	5769	5696	5706	5674	7246	11692	20037	36813	70185
Gesamtweglänge	27277	26706	26779	26880	32974	54059	91350	168108	317981
Schlangennumerierung mit reduzierter Kommunikation									
Anzahl Datenpakete	3667	3623	3631	3617	3925	6139	10337	18699	35520
Gesamtweglänge	23179	20150	21369	20573	21964	31703	52735	93023	178591

Tabelle 5.5: Datenpakete und Gesamtweglänge bei der Schlangennumerierung (Datenblätter 13 und 22)

pakete und der Gesamtweglängen (Tabellen 5.4 bis 5.6): Während sich die Anzahl der Datenpakete, die versandt werden, fast halbiert hat, reduziert sich die Gesamtweglänge aller Datenpakete nicht auf die Hälfte. Diese Feststellung bestätigt die Erwartung, daß nur die kurzen Kommunikationswege innerhalb eines Segments durch die Reduzierung der Kommunikation wegfallen.

5.3.2 Exakte Halbierung

Im vorhergehenden Kapitel wurde häufig von einem Idealfall gesprochen, in dem die Segmente immer exakt halbiert werden. Dieser Idealfall wird aber nicht immer erreicht, wie man an der durchschnittlichen Iterationstiefe erkennen kann. Auch wenn das Pivotelement immer recht gut ist, kommt es doch hin und wieder vor, daß es nicht zu einer exakten Halbierung kommt, und somit in einem Segment mehr als die nötige Anzahl von Iterationen ausgeführt werden müssen. Deshalb wäre es wünschenswert, wenn man die exakte Halbierung erzwingen könnte.

Die Optimierungstechnik der exakten Halbierung ist zwar unabhängig von der Optimierungstechnik der reduzierten Kommunikation, dennoch sollen gleich beide miteinander

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Hilbertnumerierung mit voller Kommunikation									
Anzahl Datenpakete	5757	5698	5699	5673	7214	11727	20075	36834	70184
Gesamtweglänge	23968	24315	24404	24358	29534	48470	82400	151431	286745
Hilbertnumerierung mit reduzierter Kommunikation									
Anzahl Datenpakete	3666	3624	3625	3616	3943	6127	10344	18805	35533
Gesamtweglänge	24048	24097	24154	24219	25107	38256	64560	116644	219934

Tabelle 5.6: Datenpakete und Gesamtweglänge bei der Hilbertnumerierung (Datenblätter 14 und 23)

kombiniert werden.

Der Ablauf des Datenaustausches (Abbildung 5.11) ist fast identisch mit dem Datenaustausch bei der reduzierten Kommunikation. Der einzige Unterschied ist, daß jeder Prozessor weiß, daß es eine exakte Halbierung gibt:

- Jeder Prozessor ermittelt, wie viele seiner Elemente kleiner oder gleich bzw. größer dem Pivotelement sind. Das Ergebnis ist für jeden Prozessor innerhalb des Kästchens angegeben.
- Jeder Prozessor kann jetzt selbst berechnen, zu welchem Segment er gehören wird und welche seiner Daten er behalten kann. Für die Daten, die er behalten möchte, setzt er seine Anzahl auf 0. In der Grafik sind diese Werte für jeden einzelnen Prozessor in der Zeile D dargestellt.
- Über diese Daten der einzelnen Prozessoren wird jetzt die bekannte *scan*-Operation mit Vektoraddition durchgeführt. Das Ergebnis ist in der Zeile S dargestellt.
- Das Ergebnis der *scan*-Operation beim Wurzelprozessor wird durch eine *broadcast*-Operation allen anderen Prozessoren im Segment mitgeteilt.
- Jetzt kann der eigentliche Datenaustausch beginnen. Anhand der bei jedem Prozessor vorliegenden Werte kann jeder Prozessor berechnen, wie viele Daten er zu erwarten hat. Diese Anzahl ist in der Zeile E dargestellt. Fügt man zu den empfangenen Daten die behaltenen Daten hinzu, so erhält man die Anzahl der Daten pro Prozessor. Diese ist in den Kästchen angegeben.

Der Nachteil an der erzwungenen Halbierung ist, daß ein Segment durchschnittlich mehr Elemente bekommen kann, als wenn man die Prozessoren fair aufteilt. Somit läßt die erzwungene Halbierung eine schlechtere Verteilung der Daten erwarten, was wiederum zu schlechteren Zeiten bei der lokalen Sortierung führt. Falls das Pivotelement aber gut

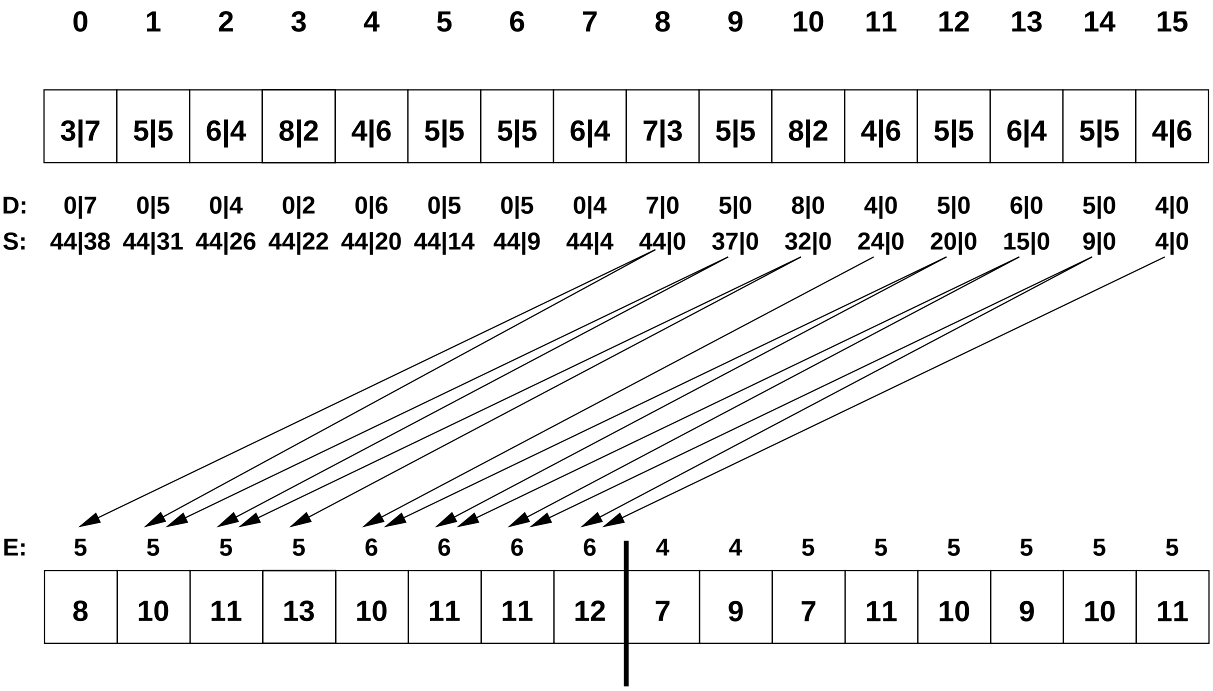


Abbildung 5.11: Datenaustausch bei exakter Halbierung und reduzierter Kommunikation

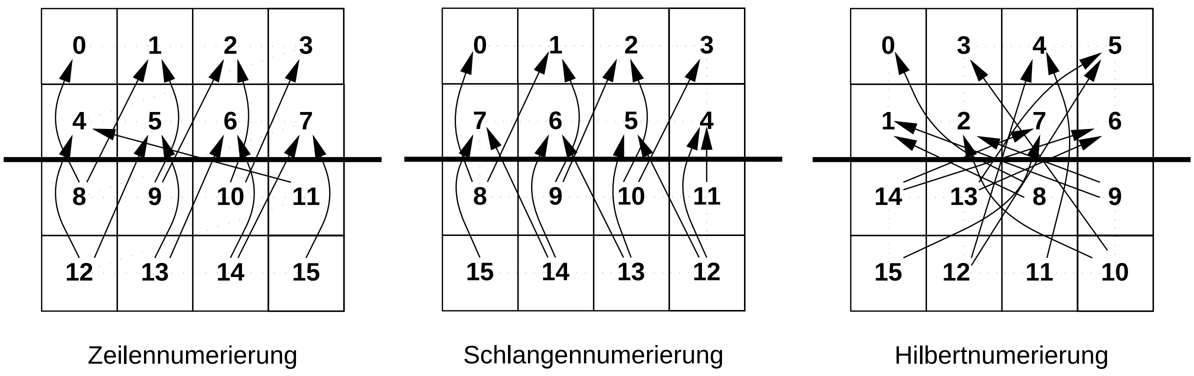


Abbildung 5.12: Datenaustausch bei exakter Halbierung und reduzierter Kommunikation

ist, wird das Segment sowieso immer halbiert. Somit wäre die exakte Halbierung dann wirkungs- und bedeutungslos.

Bei der Betrachtung der Kommunikationswege im 2-dimensionalen Gitter bei der Zeilen- und Schlangennumerierung (Abbildung 5.12) zeigt sich eine nahezu regelmäßige Struktur. Es gibt wenige Überschneidungen und die Kommunikationswege sind sehr kurz. Nur der Kommunikationsweg von Prozessor 11 nach Prozessor 4 bei der Zeilennumerierung fällt ein wenig aus dem Rahmen.

Bei der Hilbertnumerierung scheinen die Kommunikationswege besonders lang zu sein. Sie führen fast ausschließlich über die vertikale Symmetrieachse des Gitters. Die Ursache für dieses ungünstige Verhalten liegt an der Berechnung der Empfänger für den Datenaustausch. Die Prozessoren am Anfang eines Segments schicken ihre Daten an den Anfang des neuen Segments. Bei exakter Halbierung erfolgt dann der Datenaustausch z.B. von Prozessor 8 nach Prozessor 0, von Prozessor 9 nach Prozessor 1, von Prozessor 10 nach Prozessor 2 usw. Dies ist gerade für die Zeilennumerierung vorteilhaft, weil diese Prozessoren auf der gleichen vertikalen Achse liegen. Im Gegensatz dazu liegen die Prozessoren bei der Hilbertnumerierung besonders weit auseinander.

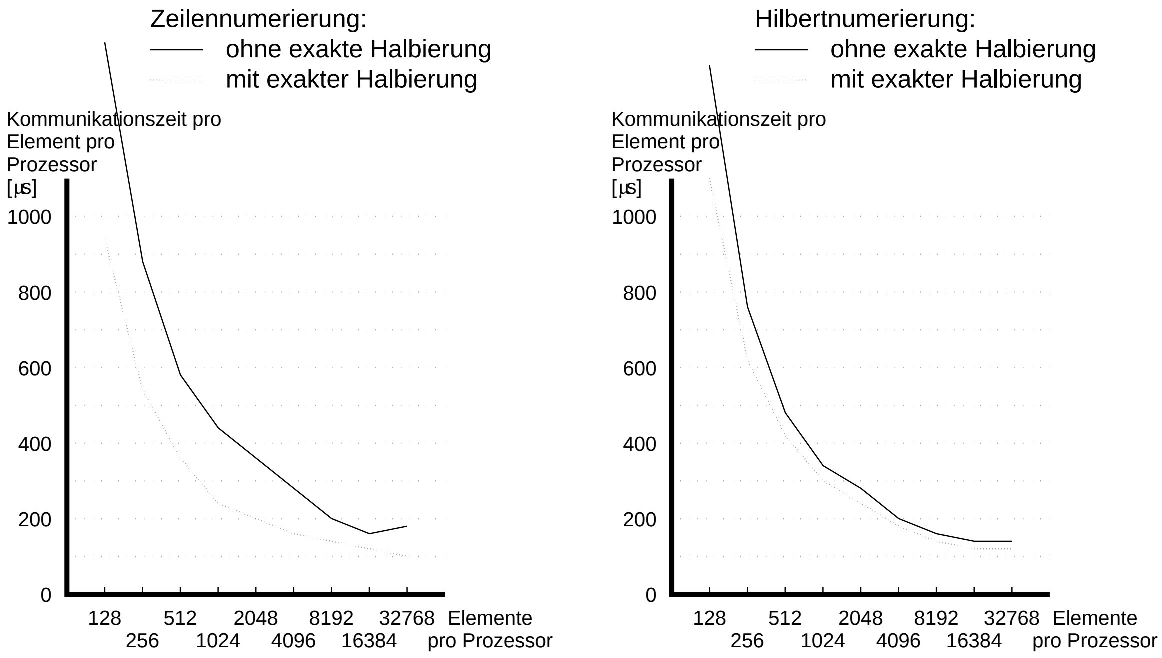


Abbildung 5.13: Vergleich der Kommunikationszeiten pro Element pro Prozessor mit und ohne exakter Halbierung (links: Datenblätter 12 und 24; rechts: Datenblätter 14 und 26)

Die Meßergebnisse für die exakte Halbierung zeigen deutlich, daß der Aufwand für die Kommunikation bei allen drei Numerierungen zurückgeht. Für die Zeilen- und Hilbertnumerierung sind die Kommunikationszeiten in Abbildung 5.13 dargestellt. Auch hier wird deutlich, daß die exakte Halbierung bei der Zeilennumerierung erheblich mehr bringt als bei der Hilbertnumerierung. Anhand der Darstellung der Kommunikationswege im 2-dimensionalen Gitter wurde aber schon deutlich, daß z.B. die Hilbertnumerierung schlechter als die Zeilennumerierung abschneidet. Und dennoch ist bei der Hilbertnumerierung ein Gewinn durch die exakte Halbierung zu beobachten.

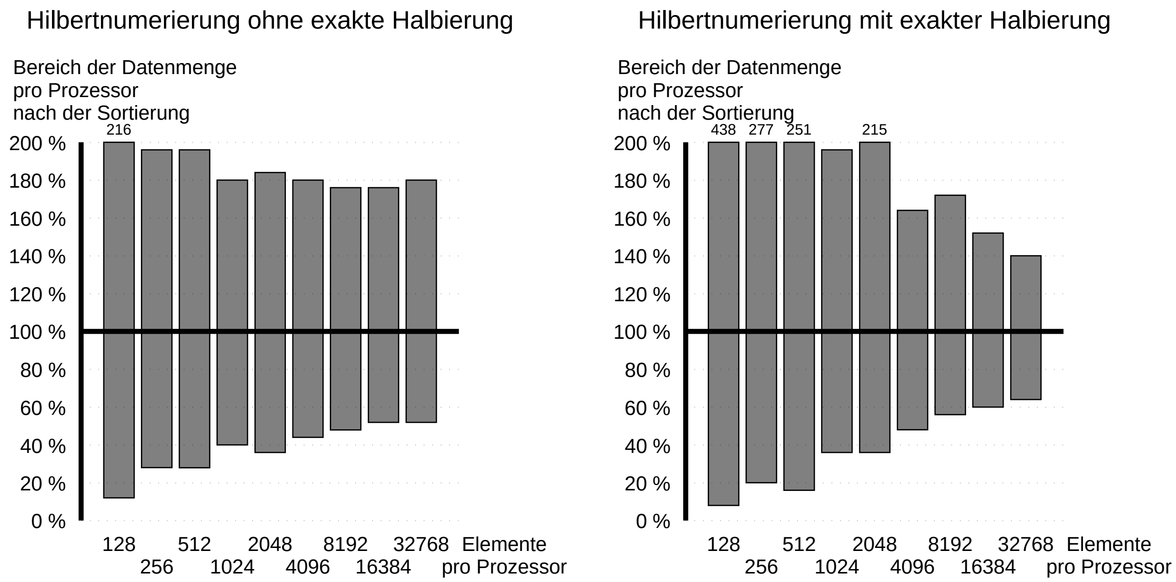


Abbildung 5.14: Verteilung der Daten nach der Sortierung bei Hilbertnumerierung mit und ohne exakter Halbierung (Datenblätter 14 und 26)

Der erwartete Nachteil, daß durch eine schlechtere Verteilung der Daten die Zeit für die lokale Sortierung ansteigt, tritt erstaunlicherweise nur für die kleineren Datenmengen ein. Wie die Abbildung 5.14 zeigt, führt ab 4096 Elementen pro Prozessor die exakte Halbierung sogar zu einer besseren Verteilung der Daten, wodurch die Zeiten für die lokale Sortierung geringer werden. Somit ergibt sich insgesamt eine Reduzierung der Sortierzeiten um bis zu 20%.

Bei der Darstellung der Kommunikationswege im 2-dimensionalen Gitter mit der Hilbertnumerierung hat sich gezeigt, daß diese Kommunikationswege weder besonders kurz noch überschneidungsfrei sind. Die Ursache dafür liegt in der Organisation des Datenaustausches. Dies liefert den Ansatz für eine weitere Verbesserung.

5.3.3 Datentauschinvertierung

Beim Datenaustausch senden die Prozessoren am Ende eines Segments ihre Daten an die Prozessoren am Ende des neuen Segments. Die sich so ergebenden Kommunikationswege hängen von der verwendeten Numerierung ab. In dem Beispiel des letzten Kapitels

(Abbildung 5.12) wurden die Kommunikationswege für verschiedene Numerierungen bei 16 Prozessoren dargestellt. Dabei zeigten sich kurze und viele überschneidungsfreie Kommunikationswege bei der Zeilen- und Schlangennumerierung, während die Kommunikationswege bei der Hilbertnumerierung alle länger waren und sich häufig überschnitten haben.

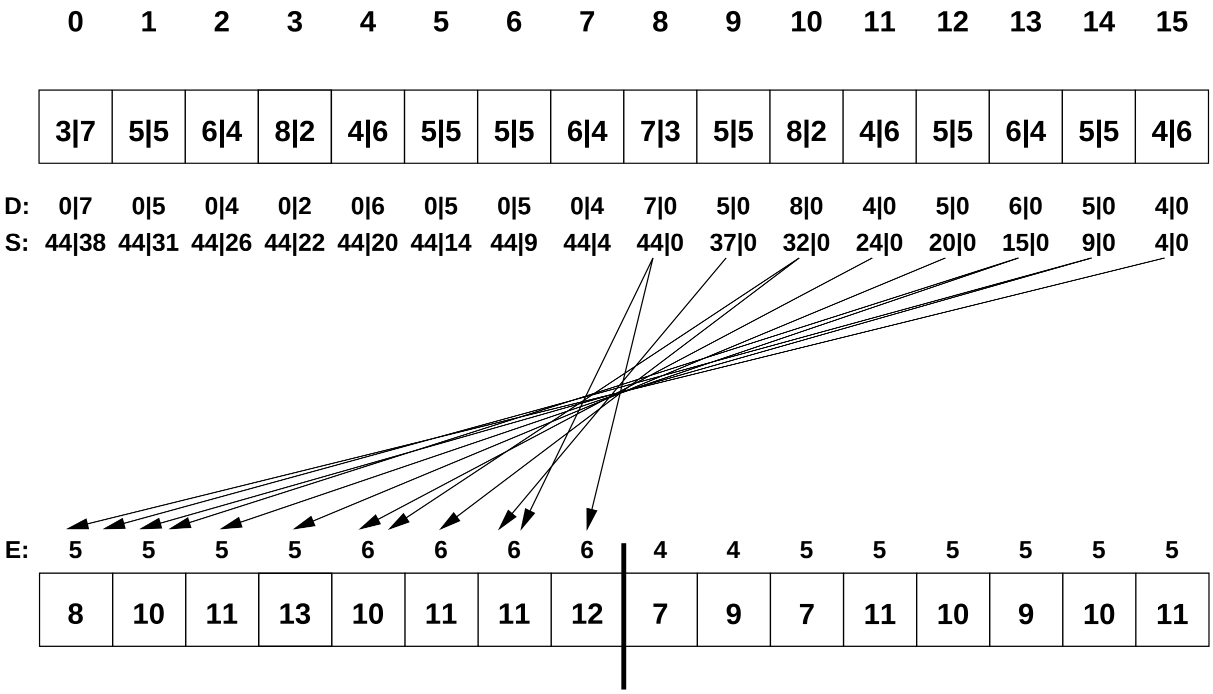


Abbildung 5.15: Datenaustausch bei Datentauschinvertierung, reduzierter Kommunikation und exakter Halbierung

Deshalb könnte es für die Hilbertnumerierung von Vorteil sein, wenn man den Datenaustausch so organisiert, daß die Prozessoren am Segmentanfang ihre Daten an die Prozessoren schicken, die im neuen Segment am Ende liegen. Diese Optimierungstechnik wird hier als Datentauschinvertierung bezeichnet. Das Beispiel in Abbildung 5.15 und 5.16 zeigt Datentauschinvertierung in Verbindung mit reduzierter Kommunikation und exakter Halbierung.

Wie zu erwarten war, zeigt die Datentauschinvertierung bei der Hilbertnumerierung fast nur vertikale Kommunikationswege. Für die Schlangennumerierung ergibt sich ein ähnliches Bild, während sich die Kommunikationswege bei der Zeilennumerierung häufig überschneiden.

Von diesen Darstellungen darf man sich jedoch nicht täuschen lassen, denn es werden nur die Kommunikationswege in der ersten Iteration dargestellt. In jeder weiteren Iteration wiederholt sich aber die Kommunikation, wobei das Segment dann in der Regel nicht mehr aus einem Quadrat besteht. Lediglich bei der Hilbertnumerierung ergibt sich in

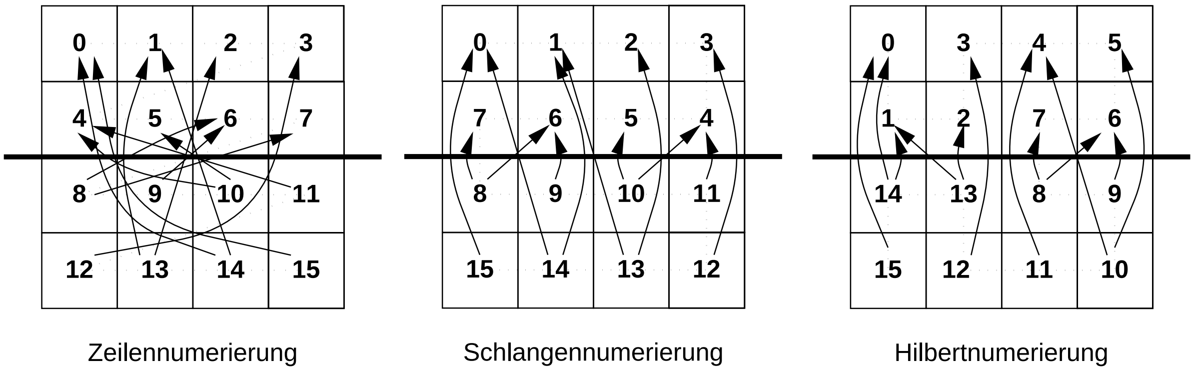
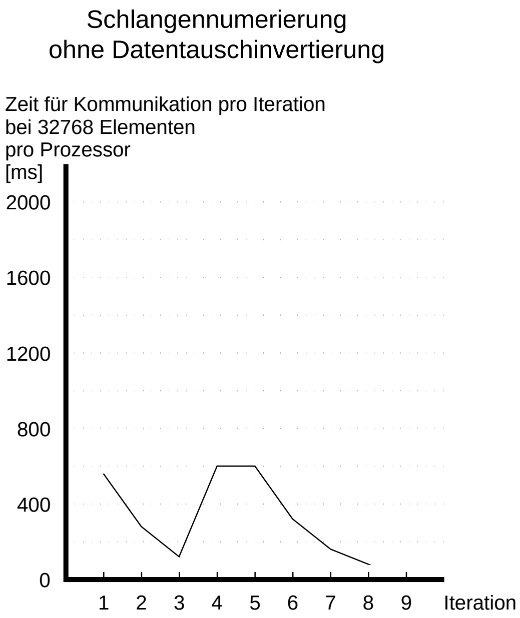


Abbildung 5.16: Datenaustausch bei Datentauschinvertierung, reduzierter Kommunikation und exakter Halbierung

jeder zweiten Iteration wieder die Quadratform.

Betrachtet man die Meßergebnisse bei Verwendung der Datentauschinvertierung, so zeigt sich das erwartete Bild: Für die Zeilennumerierung werden die Sortierzeiten schlechter, während sie sich für die Schlangen- und Hilbertnumerierung verbessern. Die Wirkung der Datentauschinvertierung zeigt sich deutlich, wenn man die Zeit für den Datenaustausch für jede Iteration betrachtet. Bei der Schlangennumerierung (Abbildung 5.17) sind die Zeiten in jeder Iteration fast identisch. Nur in der vierten Iteration wird die Zeit durch die Datentauschinvertierung von 600 μs auf 80 μs gedrückt. Die Ursache liegt darin, daß ein Segment nur noch aus zwei Zeilen mit je 16 Prozessoren besteht. Ohne Datentauschinvertierung bedeutet dies, daß der letzte Prozessor einer Zeile seine Daten an den Prozessor am Anfang der anderen Zeile senden muß. Dieser Weg ist recht lang. Bei Verwendung der Datentauschinvertierung schickt hingegen der letzte Prozessor der einen Zeile seine Daten an den letzten Prozessor der anderen Zeile. Somit sind diese Prozessoren direkt miteinander verbunden und die Kommunikationswege äußerst kurz.

Bei der Hilbertnumerierung (Abbildung 5.18) zeigt sich ein anderes Phänomen. Die Zeiten für den Datenaustausch pro Iteration bilden eine Art Treppenfunktion, sieht man von den ersten beiden Iterationen ab. Das Segment besteht bei der Hilbertnumerierung am Anfang aus einem Quadrat. In der folgenden Iteration handelt es sich um ein Rechteck und danach wieder um ein Quadrat. Die Kommunikationswege im Quadrat und anschließend im Rechteck sind gleich lang, so daß sich die gleichen Kommunikationszeiten ergeben. In der nächsten Iteration handelt es sich bei dem Segment wieder um ein Quadrat, in dem die Entfernungen nur halb so groß sind. Deshalb sinken auch die Kommunikationszeiten.



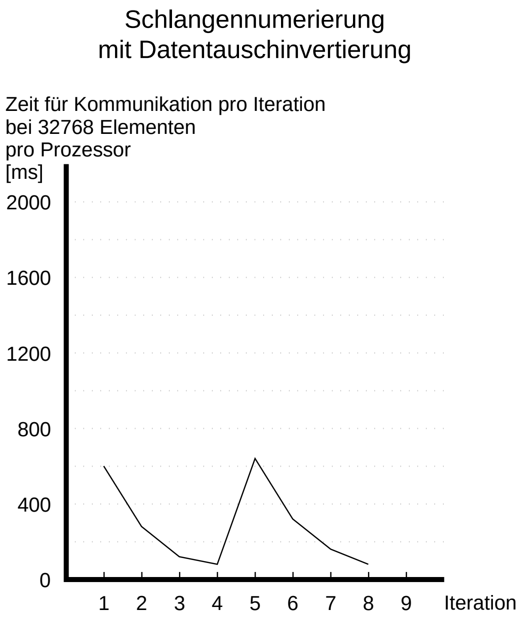


Abbildung 5.17: Zeit für Datenaustausch pro Iteration bei Schlangennumerierung mit und ohne Datentauschinvertierung bei 256 Prozessoren (Datenblätter 25 und 28)

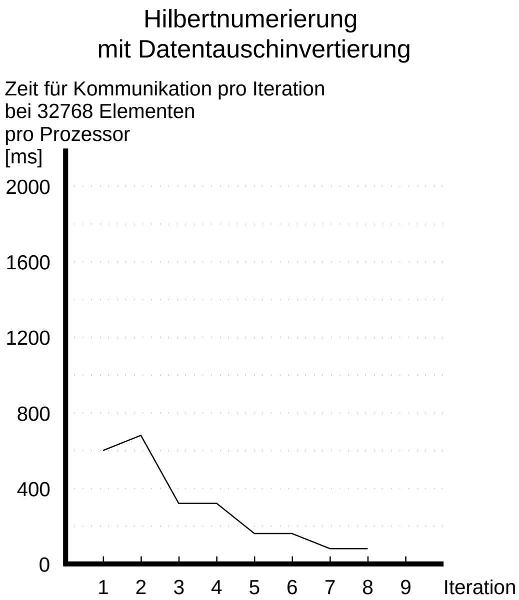


Abbildung 5.18: Zeit für Datenaustausch pro Iteration bei Hilbertnumerierung mit Datentauschinvertierung (Datenblatt 29)

5.3.4 Exakte Partnerprozessoren

Bislang wurde die exakte Halbierung recht erfolgreich eingesetzt. Dennoch hat sich, wie man an Abbildung 5.11 sieht, gezeigt, daß es Prozessoren gibt, die ihre Daten an zwei verschiedene Prozessoren schicken müssen. Im Idealfall sollte es aber so sein, daß ein Prozessor nur an exakt einen Prozessor Daten verschickt. Diese exakten Partnerprozessoren lassen sich auf Kosten einer etwas schlechteren Verteilung der Daten über die Prozessoren erzwingen.

Da die exakte Halbierung der Prozessoren eine Vorbedingung der exakten Partnerprozessoren ist, ist vorab sichergestellt, daß beide Segmente gleich groß sind. Jedem Prozessor muß jetzt nur noch ein eindeutiger Partner zugeordnet werden, was natürlich über die Prozessornummer erfolgt. Dadurch lassen sich auch die kollektiven *scan*- und *broadcast*-Operationen einsparen, denn es muß nun nicht mehr ermittelt werden, zu welchem Prozessor die Daten gesandt werden müssen. Dies ist durch den exakten Partnerprozessor bereits klar. Die Anzahl der Elemente, die jeder Prozessor erwartet, kann jetzt jedoch nicht mehr selbst berechnet werden, sondern wird mit der ersten Nachricht vom versendenden Prozessor mitgeteilt. Die Abbildung 5.19 zeigt den Datenaustausch unter Verwendung von exakten Partnerprozessoren.

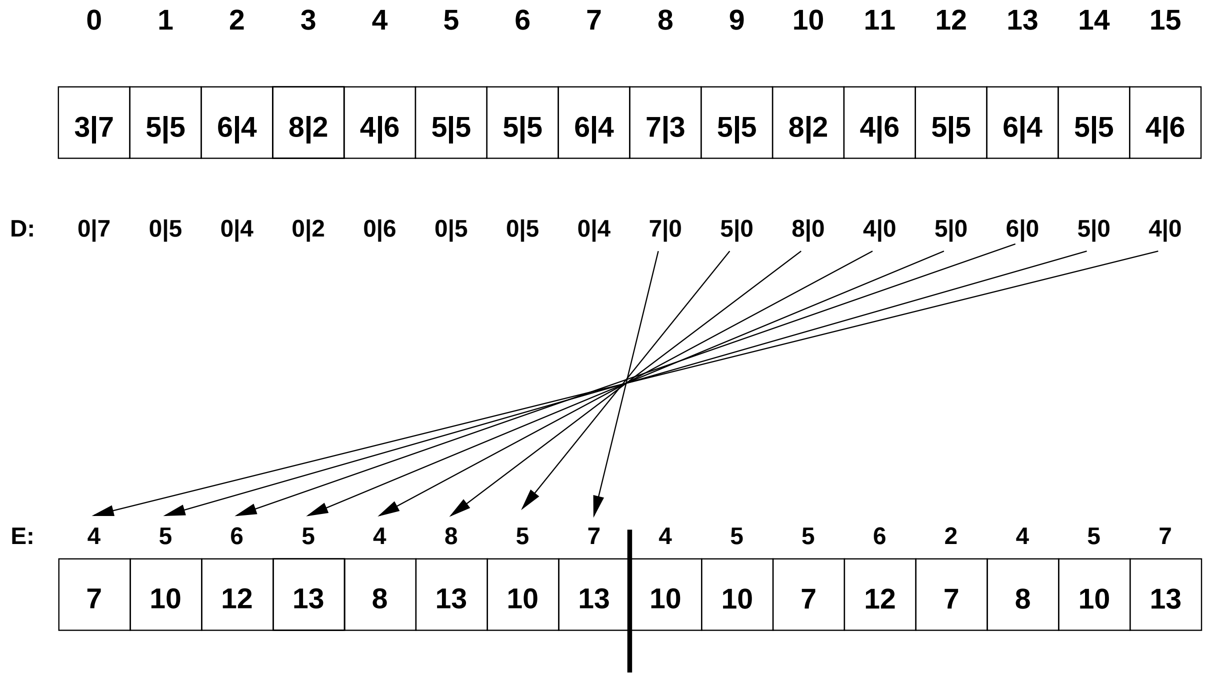


Abbildung 5.19: Datenaustausch bei Datentauschinvertierung, reduzierter Kommunikation, exakter Halbierung und exakten Partnerprozessoren

Wie sich an der Verteilung der Daten nach der Datenaustauschphase zeigt, ist diese in beiden Segmenten schlechter geworden. Betrachtet man die Kommunikationswege im Gitter, so ergibt sich, wie in Abbildung 5.20 zu sehen, das gewünschte Bild.

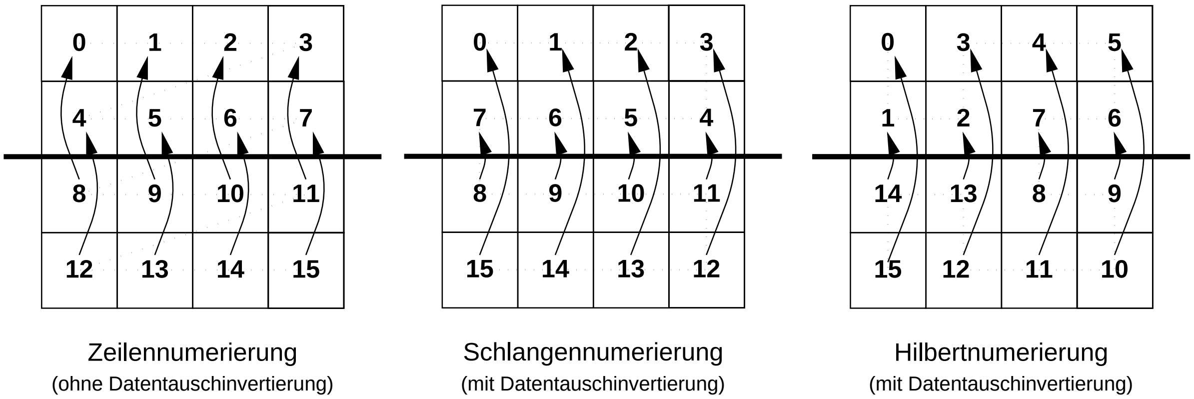


Abbildung 5.20: Datenaustausch bei reduzierter Kommunikation, exakter Halbierung und exakten Partnerprozessoren

Allerdings muß erwähnt werden, daß es sich bei der Betrachtung der Kommunikationswege um die statische Betrachtung der ersten Iteration handelt. Die Veränderung der Segmentgröße und Segmentform hat aber einen erheblichen Einfluß auf die Kommunikationswege in den folgenden Iterationen. Die Segmentform ist ausschließlich abhängig von der Prozessornumerierung, worauf in dem Kapitel 5.4 näher eingegangen wird.

Bei kleinen Datenmengen ergibt sich durch die exakten Partnerprozessoren der Vorteil, daß die Daten mit nur einer Nachricht verschickt werden können. In der Regel wären sonst wegen der unterschiedlichen Empfänger mehrere kleinere Nachrichten erforderlich.

An den Tabellen 5.7 bis 5.9 erkennt man, wie die Anzahl der Nachrichten und die Gesamtweglänge bei den einzelnen Numerierungen zurückgegangen ist.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Zeilennumerierung ohne exakte Partnerprozessoren (ohne Datentauschinvertierung)									
Anzahl Datenpakete	3537	3551	3558	3571	4357	6576	10760	19123	35785
Gesamtweglänge	15540	15568	15586	15684	18469	26565	42540	73573	135956
Zeilennumerierung mit exakten Partnerprozessoren (ohne Datentauschinvertierung)									
Anzahl Datenpakete	2048	2048	2048	2049	3217	5375	9572	17929	34582
Gesamtweglänge	7680	7680	7680	7681	12018	20303	36134	67504	129597

Tabelle 5.7: Datenpakete und Gesamtweglänge bei der Zeilennumerierung mit und ohne exakten Partnerprozessoren (Datenblätter 24 und 30)

Bei der Zeilennumerierung wird ab sofort auf die Datentauschinvertierung verzichtet, da diese hier nur Nachteile bringt. Durch die exakten Partnerprozessoren liegen die kommunizierenden Prozessoren genau auf einer Achse und alle Kommunikationswege sind gleich lang. Das führt insgesamt zu einer starken Reduzierung der Gesamtweglänge.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Schlangennumerierung ohne exakte Partnerprozessoren (mit Datentauschinvertierung)									
Anzahl Datenpakete	3530	3547	3551	3577	4380	6520	10776	19150	35834
Gesamtweglänge	15013	15070	15047	15187	18029	25894	41908	73226	135705
Schlangennumerierung mit exakten Partnerprozessoren (mit Datentauschinvertierung)									
Anzahl Datenpakete	2048	2048	2048	2048	3255	5453	9535	17942	34566
Gesamtweglänge	7680	7680	7680	7680	12141	20583	35933	67606	129575

Tabelle 5.8: Datenpakete und Gesamtweglänge bei der Schlangennumerierung mit und ohne exakten Partnerprozessoren (Datenblätter 28 und 31)

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Hilbertnumerierung ohne exakte Partnerprozessoren (mit Datentauschinvertierung)									
Anzahl Datenpakete	3525	3547	3553	3573	4387	6499	10776	19141	35822
Gesamtweglänge	17110	17231	17267	17376	20097	29210	48124	83724	155318
Hilbertnumerierung mit exakten Partnerprozessoren (mit Datentauschinvertierung)									
Anzahl Datenpakete	2048	2048	2048	2048	3254	5451	9537	17939	34568
Gesamtweglänge	8800	8800	8800	8800	14346	23937	41525	77527	148246

Tabelle 5.9: Datenpakete und Gesamtweglänge bei der Hilbertnumerierung mit und ohne exakten Partnerprozessoren (Datenblätter 29 und 32)

Bei der Schlangen- und Hilbertnumerierung liegen bei Verwendung der Datentauschinvertierung alle kommunizierenden Prozessoren ebenfalls auf der gleichen Achse. Im Unterschied zur Zeilennumerierung haben die Kommunikationswege bei der Schlangen- und Hilbertnumerierung aber innerhalb einer Iteration unterschiedliche Längen.

Insgesamt zeigt sich, daß die Gesamtweglänge durch die exakten Partnerprozessoren weiter reduziert werden konnte. Die Gesamtweglänge bei der Hilbertnumerierung ist aber deutlich höher als bei der Schlangen- oder Zeilennumerierung. Die Ursache hierfür liegt darin, daß durch die exakte Halbierung und die exakten Partnerprozessoren der Ablauf von Quicksort regelmäßig und strukturiert erfolgt. Deshalb bieten die dazu passenden regelmäßigen Numerierungen gegenüber der Hilbertnumerierung einen Vorteil. Die Hilbertnumerierung hat hingegen dort ihren Vorteil, wo man keine Aussagen und Annahmen über die Segmentaufteilung machen kann.

5.4 Prozessornumerierung

Bislang wurden mit der Zeilen-, Schlangen- und Hilbertnumerierung bereits drei verschiedene Numerierungen eingesetzt. Die Meßergebnisse haben dabei gezeigt, daß die Hilbertnumerierung die besten Sortierzeiten liefert. Benutzt man allerdings die Optimierungstechnik der exakten Partnerprozessoren, so läuft der Quicksort-Algorithmus vollkommen regelmäßig ab. Es ist von Anfang an bekannt, wie die Prozessoren aufgeteilt werden und welche Prozessoren miteinander kommunizieren werden. Deshalb können spezielle Numerierungen, die diese Vorkenntnisse ausnutzen, besser als die Hilbertnumerierung sein. Die Hilbertnumerierung hingegen ist die beste Numerierung, wenn man keine weiteren Vorkenntnisse über den Ablauf des Algorithmus hat.

Dennoch werden noch einmal alle Numerierungen betrachtet, allerdings unter dem Aspekt, daß die Optimierungstechnik der exakten Partnerprozessoren eingesetzt wird.

Bei den bisherigen Beispielen wurde immer nur der Datenaustausch einer Iteration bei 16 Prozessoren betrachtet. Bei einem kompletten Sortiervorgang kommen aber mehrere Iterationen vor, in denen sich die Segmentgröße jeweils halbiert. Aus diesem Grund muß der Datenaustausch nicht ausschließlich für die erste Iteration betrachtet werden, sondern auch für alle weiteren. Für die lokale Sortierung mit Odd-Even-Sort, bei der in einem kleinen Segment Daten unter benachbarten Prozessoren ausgetauscht werden, ist die Art der Numerierung von großer Bedeutung.

Wünschenswert ist eine Numerierung, bei der die Gesamtweglänge insgesamt minimal ist. Sind die Datenmengen so klein, daß man beim Datenaustausch mit einer Nachricht auskommt, so läßt sich die minimale Gesamtweglänge vorhersagen. Besteht das Gitter aus $2^n \times 2^n = P$ Prozessoren, so ist die Gesamtweglänge $2P(\sqrt{P} - 1)$. Bei 256 Prozessoren beträgt die minimale Gesamtweglänge dann 7680.

Weiterhin wäre es wünschenswert, wenn die Weglängen der einzelnen Nachrichten innerhalb einer Iteration gleich wären.

5.4.1 Zeilen- und Schlangennumerierung

Die Zeilennumerierung ist die Standardnumerierung für die Prozessoren im Gitter. Benachbarte Prozessornummern gehören fast immer zu benachbarten Prozessoren. Die Ausnahme bilden die Prozessoren am Anfang und Ende einer Reihe, bei denen benachbarte Prozessornummern nicht zu benachbarten Prozessoren führen. Dieser Nachteil läßt sich durch die Schlangennumerierung beheben (Abbildung 5.21).

Dennoch bieten diese beiden Numerierungen den Nachteil, daß nach mehreren Iterationen ein Segment nur noch aus einer 1-dimensionalen Reihe besteht. Ein Vorteil hingegen ist, daß die kommunizierenden Prozessoren immer auf einer Achse liegen und alle Kommunikationswege gleich lang sind.

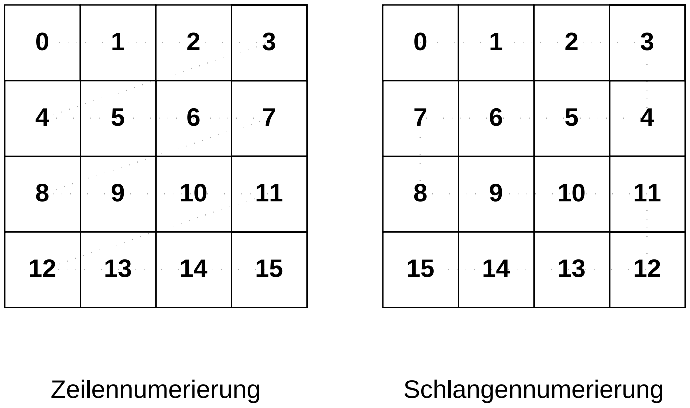


Abbildung 5.21: Zeilen- und Schlangennumerierung

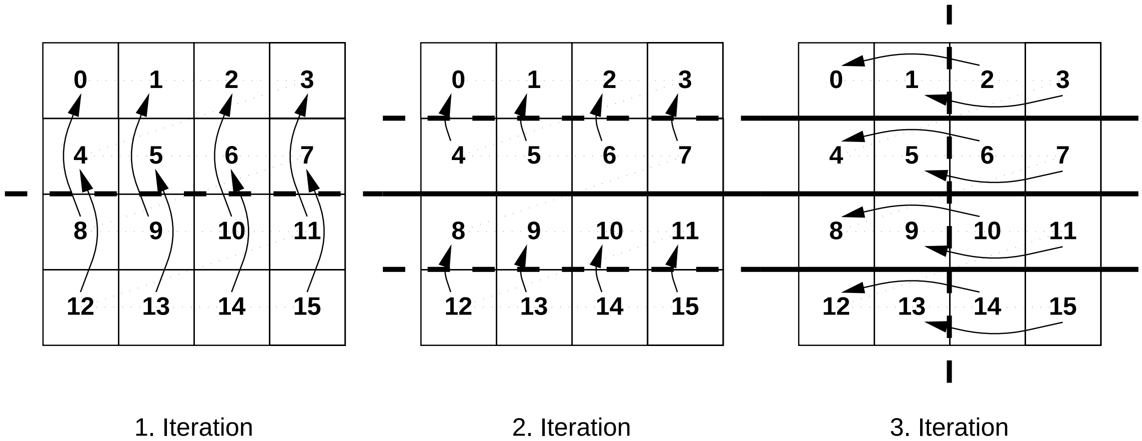


Abbildung 5.22: Drei Iterationen mit Zeilennumerierung bei Datenaustausch mit reduzierter Kommunikation, exakter Halbierung und exakten Partnerprozessoren

Am Beispiel in Abbildung 5.22 wird der Nachteil der Zeilennumerierung deutlich, denn in der dritten Iteration besteht ein Segment nur noch aus einer einzigen Zeile. Deshalb sind die Kommunikationswege in der dritten Iteration länger als in der zweiten.

Bei 256 Prozessoren führt dies zu der bereits bekannten Verlauf der Zeit für die Kommunikation pro Iteration (Abbildung 5.23). Ab der fünften Iteration steigt die Zeit wieder auf das Niveau der ersten Iteration an, weil die Entfernungen in beiden Fällen gleich lang sind. Die Gesamtweglänge aller zu versendenden Datenpakete ist bei kleinen Datenmengen mit 7680 minimal.

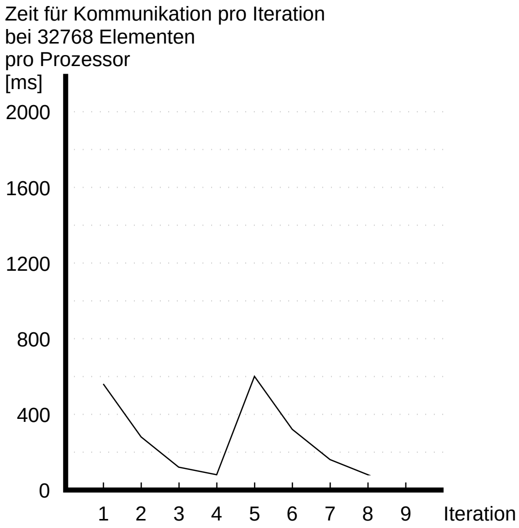


Abbildung 5.23: Typisches Verhalten der Zeit für Kommunikation pro Iteration bei der Zeilen- bzw. Schlangennumerierung (Datenblatt 24 bzw. 28)

In den Kapiteln 5.4.2 bis 5.4.5 werden Numerierungen gesucht, bei denen Segmente eine hohe Lokalität haben, die Kommunikationswege alle entlang einer Achse verlaufen und alle Kommunikationswege gleich lang sind.

5.4.2 Hilbertnumerierung

Bei der Hilbertnumerierung werden die Prozessoren des Gitters gemäß der Hilbert-Kurve numeriert. Diese Numerierung weist keine Sprünge auf und benachbarte Prozessoren sind immer direkt miteinander verbunden (Abbildung 5.24).

Die Hilbertnumerierung und alle folgenden Numerierungen berechnen sich rekursiv, wobei sich die Anzahl der Prozessoren jedesmal vervierfacht. Daraus folgt, daß nach zwei Halbierungen, also in jedem Viertel der Prozessoren, wieder eine exakte Numerierung mit den gleichen Eigenschaften vorliegt.

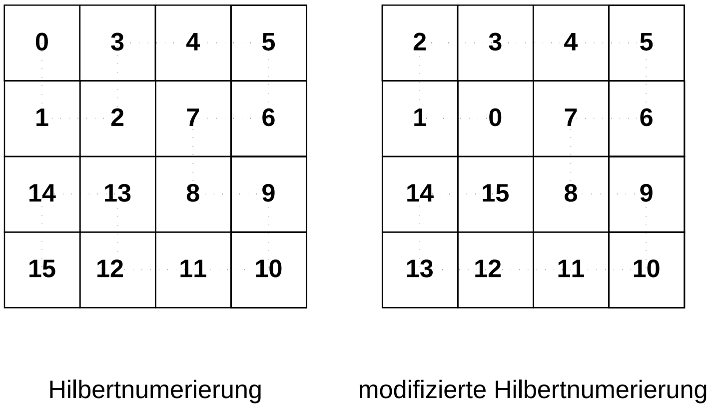


Abbildung 5.24: Hilbert- und modifizierte Hilbertnumerierung

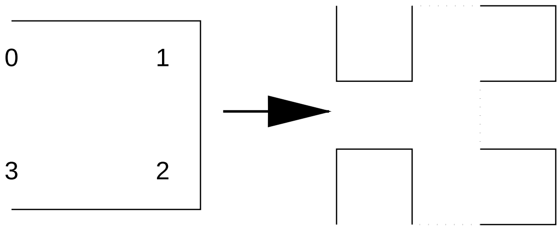


Abbildung 5.25: Regel zur Erzeugung der Hilbertnumerierung

Eine detaillierte Beschreibung zur Berechnung der Hilbertnumerierung ist in [Hans95] gegeben. Die generelle Vorgehensweise soll aber noch einmal kurz erläutert werden, weil sie sich bei der Berechnung der folgenden Numerierungen jedesmal wiederholt.

Bei 4 Prozessoren werden diese entlang eines Hufeisens numeriert. In jeder Iteration der Berechnung wird jeder Prozessor durch vier Prozessoren ersetzt, die wiederum in der Reihenfolge des Hufeisens (Abbildung 5.25) numeriert werden. Die gepunkteten Linien verdeutlichen die entstehenden Verbindungen. Speziell bei der Hilbertnumerierung ist zu beachten, daß die Numerierung im Hufeisen manchmal rechts- und manchmal linksherum erfolgen muß.

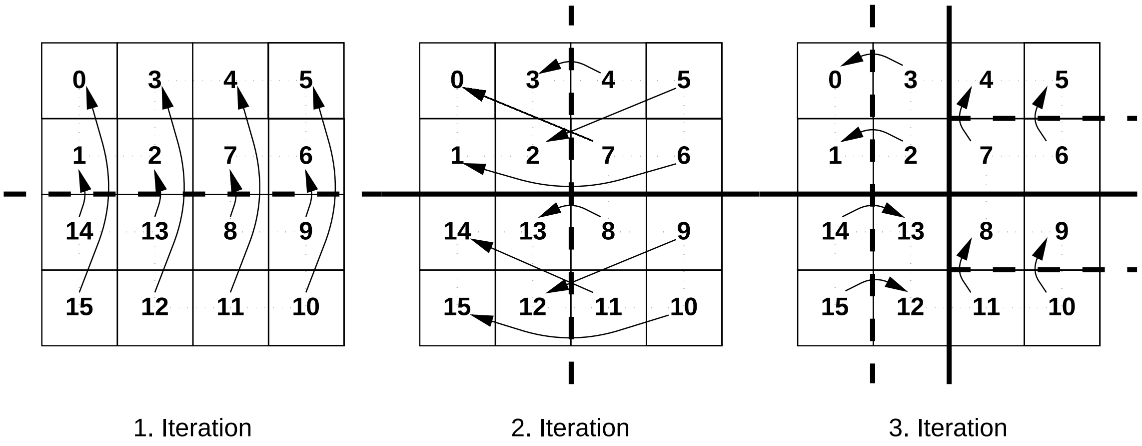


Abbildung 5.26: Drei Iterationen mit Hilbertnumerierung bei Datenaustausch mit reduzierter Kommunikation, exakter Halbierung und exakten Partnerprozessoren

Die Hilbertnumerierung ist bezüglich der ersten Halbierung symmetrisch, bezüglich der zweiten aber nicht mehr. Diese Eigenschaft zeigt sich sehr anschaulich in der Abbildung 5.26, bei der die Kommunikationswege in der ersten und dritten Iteration regelmäßig verlaufen, während es in der zweiten Iteration etwas durcheinander geht. Die Gesamtweglänge bei kleinen Datenmengen und 256 Prozessoren ist mit 8800 gegenüber 7680 bei der Schlangennumerierung bei weitem nicht optimal. Somit erfüllt die Hilbertnumerierung nicht die gewünschten Anforderungen.

Wünschenswert wäre eine Numerierung, bei der in jeder Iteration die Kommunikationswege nur entlang einer Achse verlaufen. Ein erster Ansatz dazu wäre eine modifizierte Hilbertnumerierung, die auch bei der zweiten Iteration bezüglich der Segmentgrenze symmetrisch ist. Dies erreicht man, indem die ersten 16 Prozessoren so numeriert werden, wie es in Abbildung 5.24 zu sehen ist. Die Numerierung wird mit der Bildungsregel aus Abbildung 5.25 fortgesetzt. Bei der modifizierten Hilbertnumerierung liegen die Anfangs- und Endpunkte benachbart in der Mitte, so daß sich ein Hamiltonscher Kreis ergibt.

Bei den Kommunikationswegen in Abbildung 5.27 zeigt sich das gewünschte Bild mit

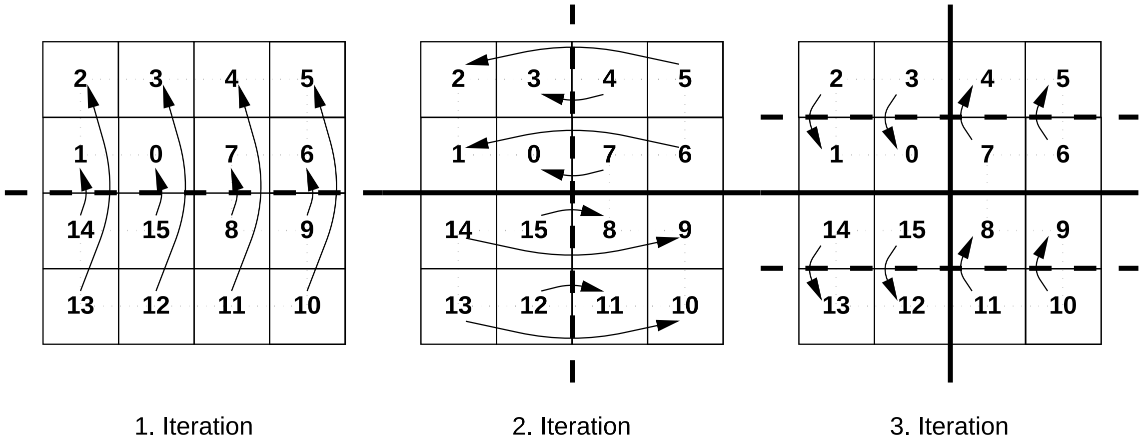


Abbildung 5.27: Drei Iterationen mit modifizierter Hilbertnumerierung bei Datenaustausch mit reduzierter Kommunikation, exakter Halbierung und exakten Partnerprozessoren

überschneidungsfreien Kommunikationswegen. Ab der vierten Iteration verhält sich die modifizierte Hilbertnumerierung allerdings genauso wie die Hilbertnumerierung. Bei kleinen Datenmengen und bei 256 Prozessoren sinkt die Gesamtweglänge gegenüber der normalen Hilbertnumerierung zwar von 8800 auf 8128, ist aber immer noch weit von dem optimalen Wert der Schlangennumerierung mit 7680 entfernt.

Deshalb erfüllt auch die modifizierte Hilbertnumerierung nicht die Anforderungen der kurzen Kommunikationswege. Wie die Meßergebnisse im Anhang (Datenblatt 33) belegen, bringt die modifizierte Hilbertnumerierung kaum einen Gewinn.

5.4.3 H-Numerierung

Wenn man eine Numerierung erhalten möchte, die bei exakten Partnerprozessoren immer Kommunikation entlang einer Achse betreibt und bei der die Kommunikationswege innerhalb einer Iteration gleich lang sind, muß man darauf verzichten, daß benachbarte Prozessornummern immer zu benachbarten Prozessoren führen. Die Stellen, an denen fortlaufend numerierte Prozessoren nicht zu direkt miteinander verbundenen Prozessoren führen, werden als Sprünge bezeichnet. Für die Durchführung der lokalen Sortierung sind solche Sprünge hinderlich, so daß die Sprünge nur an den Stellen auftreten sollen, an denen später Segmentgrenzen liegen werden. Eine mögliche Numerierung zeigt die Abbildung 5.28, die von nun an wegen den querliegenden H‘s als H-Numerierung bezeichnet wird. Die gestrichelten Linien deuten Sprünge in der Numerierung an. Bei der H-Numerierung sind Einheiten von 16 Prozessoren miteinander verbunden, ohne daß ein Sprung darin vorkommt. Somit kann man bei der lokalen Sortierung ab einer Segmentgröße von 16 Prozessoren Odd-Even-Sort einsetzen, falls dies zu einer Verbesserung der Sortierzeiten führt. Die Regel zur Erzeugung dieser H-Numerierung zeigt die Abbildung 5.29.

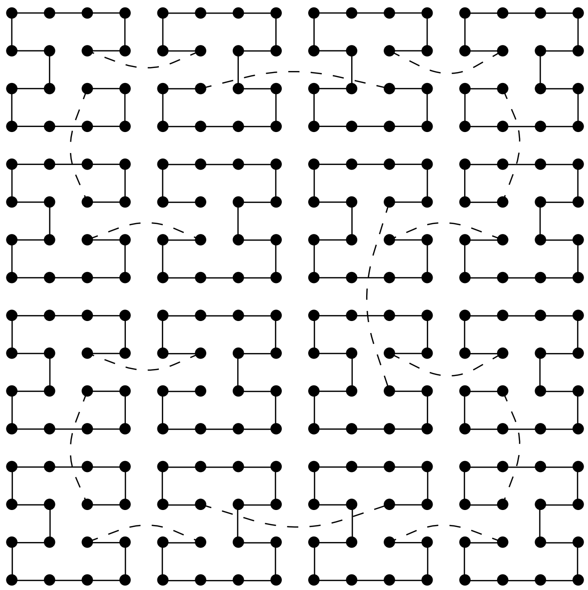


Abbildung 5.28: H-Numerierung für 256 Prozessoren

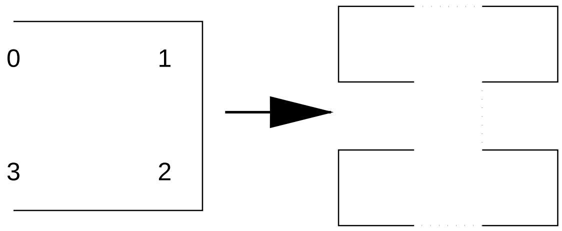


Abbildung 5.29: Regel zur Erzeugung der H-Numerierung

Betrachtet man die ersten beiden Iterationen des Datenaustausches mit der H-Numerierung, so erfolgt die Kommunikation wie gewünscht immer entlang einer Achse. Die Gesamtweglänge ist wie bei der Schlangennumerierung minimal. Allerdings sieht man schon in Abbildung 5.30, daß die Kommunikationswege innerhalb einer Iteration unterschiedlich lang sind.

Bei 16 Prozessoren sieht die H-Numerierung wie die modifizierte Hilbertnumerierung aus. Unterschiede ergeben sich erst bei höheren Prozessorzahlen.

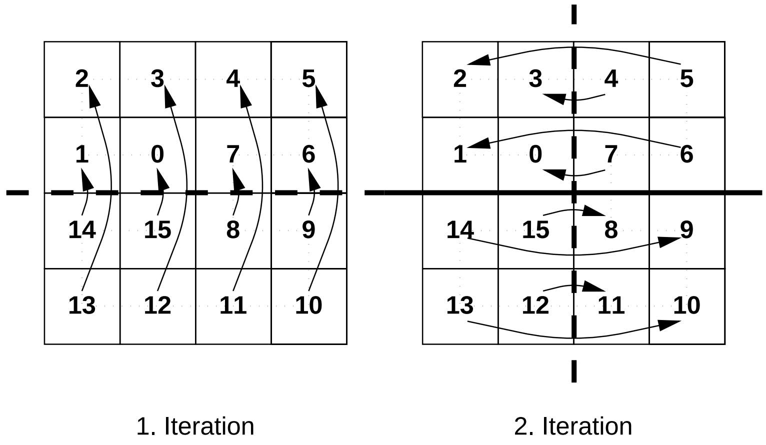


Abbildung 5.30: Die ersten beiden Iterationen beim Datenaustausch mit der H-Numerierung

5.4.4 Z-Numerierung

Bei der oben vorgestellten H-Numerierung sind die Kommunikationswege nicht alle gleich lang. Dieser Nachteil läßt sich durch die folgende Numerierung beheben, die wegen ihrer Bildungsregel von nun an Z-Numerierung genannt wird (Abbildung 5.31).

Die Sprünge sind wiederum durch gestrichelte Linien dargestellt. Durch exakte Halbierung liegen die Segmentgrenzen genau an den Stellen, an denen die Sprünge vorhanden sind.

Die Regel zur Bildung dieser Z-Numerierung ist in Abbildung 5.32 dargestellt. In der letzten Iteration geht man dazu über, die verbleibenden vier Prozessoren entlang des Hufeisens zu numerieren. Bei der Z-Numerierung sind somit Segmente von vier Prozessoren miteinander verbunden, ohne daß ein Sprung darin vorkommt.

Betrachtet man in Abbildung 5.33 die ersten beiden Iterationen des Datenaustausches mit der Z-Numerierung, so zeigt sich das gewünschte Ergebnis: Alle Kommunikationswege sind gleich lang. Dieses Verhalten gilt bis zur vorletzten Iteration der Berechnung, d.h. bis ein Segment nur noch aus vier Prozessoren besteht. Deshalb ist die Gesamtweglänge bei kleinen Datenmengen und bei 256 Prozessoren mit 7936 nicht mehr optimal. Dieser Nachteil tritt jedoch nicht auf, wenn man als lokale Sortierung Odd-Even-Sort ab 4 Prozessoren einsetzt.

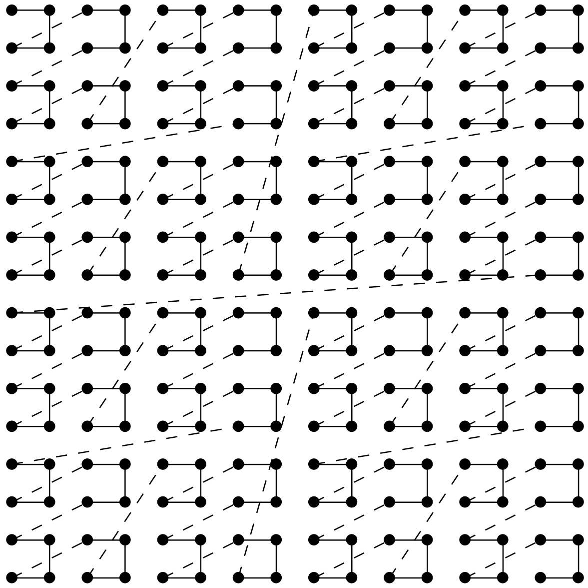


Abbildung 5.31: Z-Numerierung für 256 Prozessoren

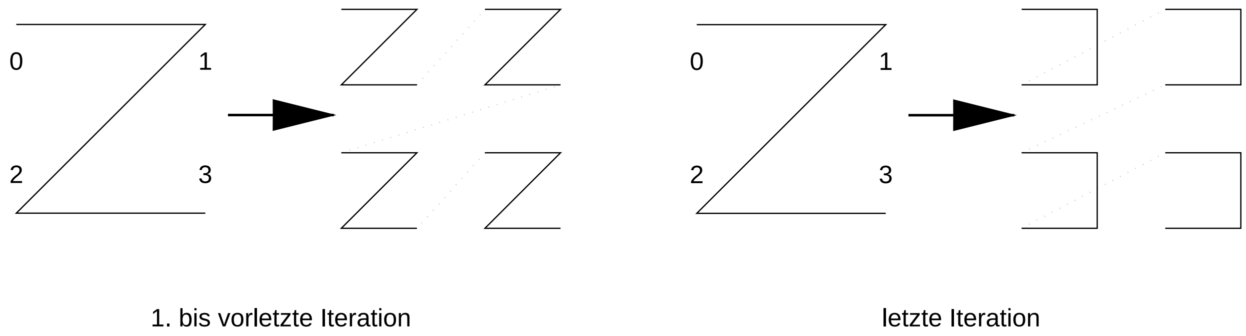


Abbildung 5.32: Regel zur Erzeugung der Z-Numerierung

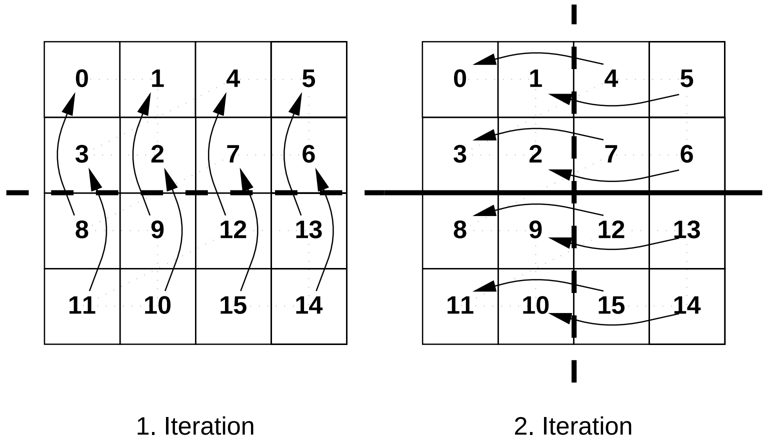


Abbildung 5.33: Die ersten beiden Iterationen beim Datenaustausch mit der Z-Numerierung

5.4.5 Shuffle-Numerierung

Die zuletzt beschriebene Z-Numerierung unterscheidet sich kaum von der bekannten Shuffle-Numerierung[2] (Abbildung 5.34). Die Regel zur Bildung dieser Shuffle-Numerierung ist identisch mit der Regel der Z-Numerierung (Abbildung 5.35). Während bei der Z-Numerierung in der letzten Iteration eine Sonderbehandlung für vier Prozessoren vorgenommen wurde, wird nun bei der Shuffle-Numerierung die gleiche Bildungsregel bis zum Ende verwendet.

In der Abbildung 5.34 sind die Sprünge wiederum durch gestrichelte Linien dargestellt. Durch die exakte Halbierung liegen die Segmentgrenzen genau an den Stellen, an denen sich auch die Sprünge befinden.

Betrachtet man in Abbildung 5.36 die ersten beiden Iterationen des Datenaustausches mit der Shuffle-Numerierung, so zeigt sich das gleiche Bild wie bei der Z-Numerierung. Der entscheidende Unterschied zwischen der Z-Numerierung und der Shuffle-Numerierung liegt, wie Abbildung 5.37 zeigt, lediglich in einem anderen Verhalten beim Datenaustausch bei Segmenten von nur noch vier Prozessoren. Wird bei vier Prozessoren nochmals eine Quicksort-Iteration ausgeführt, so sind die kommunizierenden Prozessoren bei der Shuffle-Numerierung direkt miteinander verbunden, während bei der Z-Numerierung die Kommunikation überkreuz erfolgt. Verwendet man bei vier Prozessoren aber Odd-Even-Sort, so sind bei der Z-Numerierung die kommunizierenden Prozessoren miteinander verbunden. Insgesamt ist die Gesamtweglänge bei der Shuffle-Numerierung minimal. Da aber jeweils nur zwei Prozessoren ohne Sprung verbunden sind, kann Odd-Even-Sort nur für zwei Prozessoren sinnvoll eingesetzt werden.

[2]siehe [KGGK94]

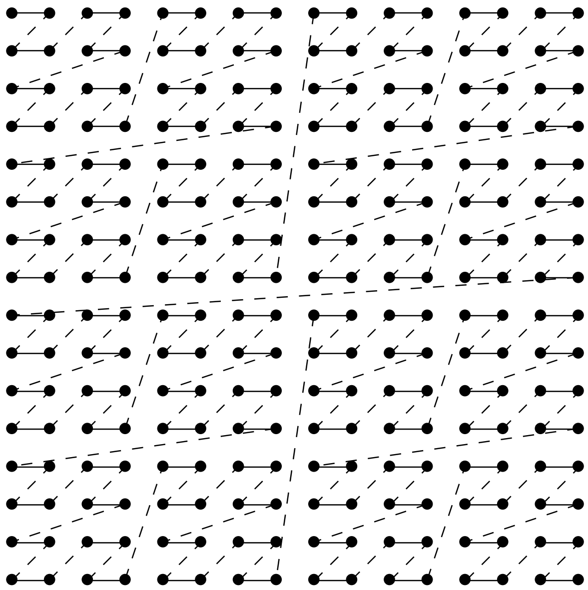


Abbildung 5.34: Shuffle-Numerierung für 256 Prozessoren

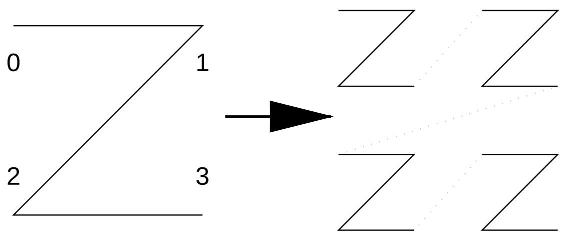


Abbildung 5.35: Regel zur Erzeugung der Shuffle-Numerierung

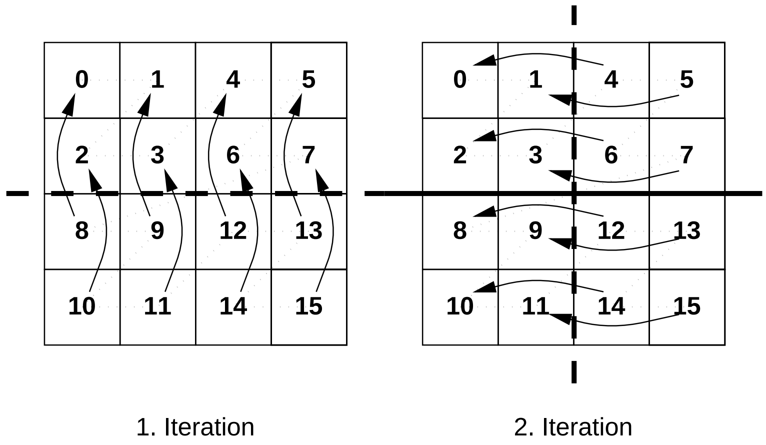


Abbildung 5.36: Die ersten beiden Iterationen beim Datenaustausch mit der Shuffle-Numerierung

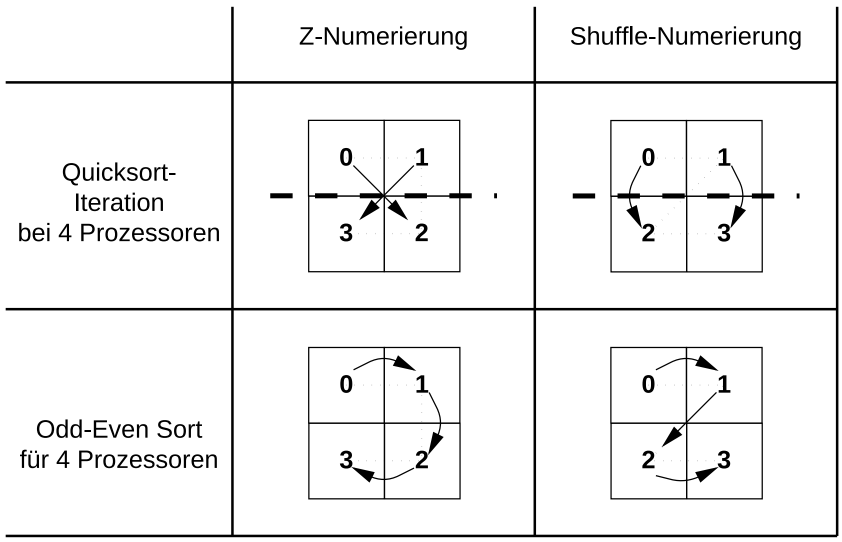


Abbildung 5.37: Shuffle-Numerierung und Z-Numerierung im Vergleich bei einem Segment mit vier Prozessoren

5.4.6 Numerierung und lokale Sortierung mit Odd-Even-Sort

Bei den Messungen (siehe Datenblätter 37 bis 40 im Anhang) hat sich gezeigt, daß sich bei exakten Partnerprozessoren Odd-Even-Sort nur für zwei Prozessoren sinnvoll einsetzen läßt. Die besten Gesamtergebnisse in Abhämgigkeit der Prozessornumerierung liefern somit die Schlangen- und die Shuffle-Numerierung.

5.4.7 Überblick Numerierungen

In diesem Kapitel 5.4 wurden viele Numerierungen vorgestellt, die sich jeweils in kleinen Details unterschieden haben. Anhand der Tabelle 5.10 sollen diese Unterschiede noch einmal verdeutlicht werden.

Numerierung	Zeilennum.	Schlangen-num.	Hilbert-num.	modifizierte Hilbert-num.	H-Num.	Z-Num.	Shuffle-Num.
Sprünge	Ja	Nein	Nein	Nein	Ja	Ja	Ja
monoton fallende Kommunikationszeiten	Nein	Nein	Ja	Ja	Ja	Ja	Ja
Kommunikation entlang einer Achse	Ja	Ja	Nein	Nein	Ja	Ja, bis vier Prozessoren	Ja
Kommunikationswege gleich lang	Ja	Ja	Nein	Nein	Nein	Ja	Ja
größte Segment ohne Sprung	$\sqrt{P}$	P	P	P	16	4	2

Tabelle 5.10: Vergleich der Numerierungen bei exakten Partnerprozessoren

- Sprünge: Prozessoren, die benachbarte logische Prozessornummern haben, sind nicht direkt miteinander verbunden.
- Monoton fallende Kommunikationszeiten: Nach jeder Halbierung der Prozessoren sollten die Kommunikationszeiten fallen oder zumindest gleich bleiben. Bei der Zeilen- und Schlangennumerierung ist das aber nicht der Fall (siehe Kapitel 5.3.1).
- Kommunikation entlang einer Achse: Alle miteinander kommunizierenden Prozessoren liegen auf einer Achse, so daß sich die Kommunikationswege erstens nicht überschneiden können und die Gesamtweglänge zweitens minimal ist.

- Kommunikationswege gleich lang: Die Entfernung zwischen zwei kommunizierenden Prozessoren ist in jeder Iteration konstant.
- Größte Segment ohne Sprung: Diese Zahl gibt an, wie groß ein Segment, das keine Sprünge enthält, maximal sein kann. Erst ab dieser Größe kann es sinnvoll sein, Odd-Even-Sort einzusetzen.

Aus der Tabelle 5.10 und den bisherigen Meßergebnissen lassen sich folgende Empfehlungen festhalten:

Die Hilbertnumerierung ist die beste Numerierung, wenn die Optimierungstechnik der exakten Halbierung und der exakten Partnerprozessoren nicht verwendet wird.

Wenn Odd-Even-Sort und die exakten Partnerprozessoren eingesetzt werden sollen, so erweisen sich mehrere Numerierungen als sinnvoll:

- Wenn bei der lokalen Sortierung kein Odd-Even-Sort oder Odd-Even-Sort für zwei Prozessoren eingesetzt werden soll, so ist die Shuffle-Numerierung zu verwenden.
- Soll für die lokale Sortierung Odd-Even-Sort für vier Prozessoren angewendet werden, so ist die Z-Numerierung zu verwenden.
- Soll für die lokale Sortierung Odd-Even-Sort für acht oder 16 Prozessoren angewendet werden, so ist die H-Numerierung zu verwenden.
- Als Alternative zur Shuffle-, Z-, und H-Numerierung kann aber auch die Schlangennumerierung eingesetzt werden.

5.5 Zusammenfassung

Alle vorgestellten Optimierungstechniken lassen sich fast in beliebiger Art und Weise miteinander kombinieren.

- Pivotwahl: Als beste Pivotstrategie hat sich der Median der $\sqrt{N}$-Mediane erwiesen.
- Odd-Even-Sort: Die lokale Sortierung wird schon ausgeführt, wenn ein Segment aus mehr als einem Prozessor besteht. Die Verteilung der Daten ist zum Zeitpunkt der lokalen Sortierung gleichmäßiger, so daß die Zeiten dafür sinken. Anschließend wird Odd-Even-Sort eingesetzt, wodurch sich die Verteilung der Daten nochmals verbessert. Ab welcher Anzahl von Prozessoren sich Odd-Even-Sort sinnvoll einsetzen läßt, hängt von den anderen gewählten Optimierungstechniken ab.
- Reduzierung der Kommunikation: Durch diese Optimierungstechnik wird der Kommunikationsaufwand verringert. Die Verteilung der Daten nach der Sortierung verschlechtert sich kaum.

- Exakte Halbierung: Durch die exakte Halbierung werden die Segmente regelmäßig aufgeteilt. Der Vorteil dieser Optimierungstechnik ist größer als ihr Nachteil der etwas schlechteren Verteilung der Daten.

- Datentauschinvertierung: Bei dieser Optimierungstechnik wird die Reihenfolge der miteinander kommunizierenden Prozessoren vertauscht. Diese Optimierungstechnik ist nur bei der Schlangennumerierung, Hilbertnumerierung, modifizierten Hilbertnumerierung und bei der H-Numerierung sinnvoll.

- Exakte Partnerprozessoren: Durch die exakten Partnerprozessoren wird der Ablauf des Quicksort-Algorithmus noch weiter strukturiert, so daß man Aussagen darüber machen kann, welche Prozessoren miteinander kommunizieren werden. Die Optimierungstechnik der exakten Partnerprozessoren bringt Vorteile, die jeweils von der gewählten Numerierung abhängen. Als Nachteil ergibt sich eine Verschlechterung der Verteilung der Daten.

- Prozessornumerierung: Durch verschiedene Prozessornumerierungen kann die Kommunikation über das Netzwerk minimiert und optimiert werden.

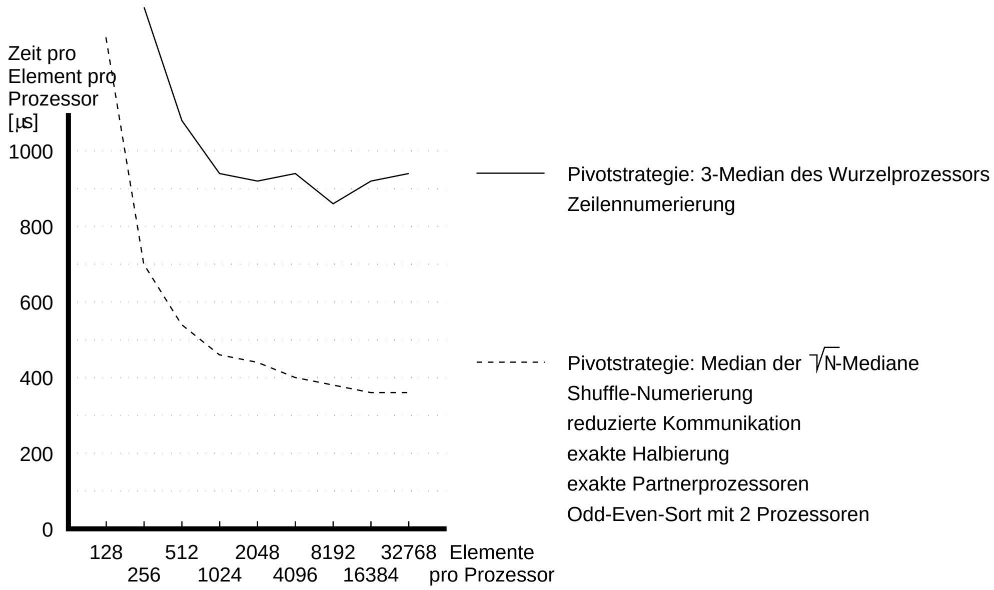


Abbildung 5.38: Vergleich der Sortierzeiten der ersten und der besten Konfiguration (Datenblätter 1 und 37)

Durch die richtige Kombination der oben beschriebenen Optimierungstechniken lassen sich die Sortierzeiten mehr als halbieren. Die Abbildung 5.38 zeigt den Vergleich der Sortierzeiten zwischen der ersten und der besten Konfiguration.

Als beste Konfiguration für die hier zu sortierenden Daten hat sich die Shuffle-Numerierung mit der Pivotstrategie Median der $\sqrt{N}$-Mediane mit reduzierter Kommunikation, exakter Halbierung und exakten Partnerprozessoren erwiesen.

Bei der Optimierungstechnik der exakten Partnerprozessoren hat sich gezeigt, daß die Kommunikation über das Netzwerk minimiert werden konnte. Die resultierende Sortierzeit ist also nur noch abhängig von der Verteilung der Daten über die Prozessoren. Durch den Einsatz von Odd-Even-Sort läßt sich die ungleiche Verteilung der Daten zwar noch verringern, aber dennoch bietet sich hier ein Ansatz, um die Sortierzeiten weiter zu verbessern.

Kapitel 6

Paralleles Quicksort mit lokaler Sortierung am Anfang

In den beiden vorangegangenen Kapiteln 4 und 5 wurde ausschließlich nur die Quicksort-Variante betrachtet, bei der die lokale Sortierung am Ende stattfand. Durch verschiedene Optimierungstechniken konnte der Aufwand, der für die Kommunikation notwendig ist, reduziert werden. Der Aufwand, der von der lokalen Sortierung herrührt, wird maßgeblich durch den Prozessor beeinflußt, der die meisten Daten zu sortieren hat. Die ungleiche Verteilung der Daten konnte zwar durch den Einsatz von Odd-Even-Sort gemindert werden, dennoch bietet sich hier ein Ansatzpunkt für eine weitere Variante des Algorithmus.

Am Anfang liegen die zu sortierenden Daten gleichmäßig über alle Prozessoren verteilt. Deshalb bietet es sich an, die lokale Sortierung zu diesem Zeitpunkt durchzuführen, da dann alle Prozessoren gleichmäßig ausgelastet sind. Anschließend wird der eigentliche Quicksort-Algorithmus in der bekannten Weise ausgeführt. Der einzige, aber gravierende Unterschied besteht darin, daß jetzt sortierte Daten über das Netzwerk verschickt werden müssen. Beim empfangenden Prozessor können diese Daten dann nicht mehr einfach aneinander gehängt werden, sondern müssen zu einer neuen sortierten Folge zusammengesetzt werden. Hier entsteht somit ein neuer zusätzlicher Aufwand.

Der Algorithmus bleibt ansonsten unverändert. Deshalb wird weiterhin die Bezeichnung paralleles Quicksort verwendet, wobei ab jetzt immer Quicksort mit lokaler Sortierung am Anfang gemeint ist, falls nichts anderes angegeben wird.

6.1 Lokale Sortierung

Die lokale Sortierung wird jetzt am Anfang durchgeführt. Als sequentieller Sortieralgorithmus wird nach wie vor Quicksort aus der C-Standard-Bibliothek verwendet.

Als die lokale Sortierung noch am Ende erfolgte, konnte man Odd-Even-Sort einsetzen, um die ungleiche Verteilung der Daten zu verringern. Dabei hat man die Sortierung auf jedem Prozessor schon durchgeführt als das Segment noch aus mehr als einem Prozessor bestand und wendete anschließend Odd-Even-Sort auf die sortierten Daten an. Bei der Quicksort-Variante mit lokaler Sortierung am Anfang wird die Sortierung der Daten auf den Anfang vorgezogen. Danach beginnt der eigentliche parallele Quicksort-Algorithmus. Da das parallele Quicksort jetzt mit sortierten Datenmengen arbeitet, besteht für den Einsatz von Odd-Even-Sort kein Bedarf mehr.

Der entscheidende Vorteil für die Sortierung am Anfang ist, daß jeder Prozessor gleich viele Daten zu sortieren hat. Somit ist die Zeit für die lokale Sortierung auf allen Prozessoren annähernd gleich.

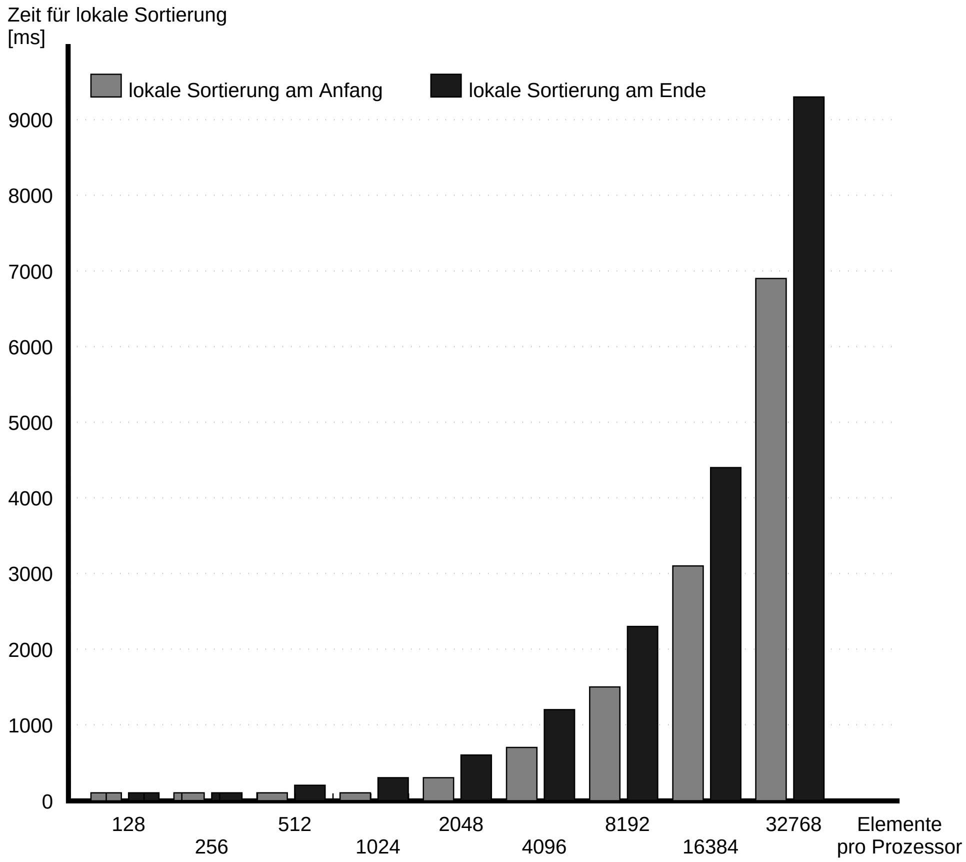


Abbildung 6.1: Zeiten für die lokale Sortierung am Anfang und am Ende (Datenblätter 41 und 36)

Die Zeit für die lokale Sortierung am Anfang ist unabhängig von allen anderen Optimierungstechniken. In der Abbildung 6.1 wird deutlich, daß sie einen erheblichen Gewinn gegenüber der besten Konfiguration mit Sortierung am Ende bietet.

6.2 Prozessornumerierung

Die Prozessornumerierung ist vollkommen unabhängig davon, ob die Daten am Anfang oder am Ende lokal sortiert werden. Es können somit alle Numerierungen eingesetzt werden.

6.3 Pivotwahl

Die Pivotwahl gestaltet sich jetzt etwas einfacher, denn aus einer sortierten Folge läßt sich lokal ein echter Median mit konstantem Aufwand ermitteln. Durch eine kollektive *reduce*-Operation, die die lokalen Mediane mit der Verknüpfungsoperation Merge einsammelt, kann der Wurzelprozessor den Median der lokalen Mediane ermitteln. Im Gegensatz zu den bislang verwendeten Pivotstrategien, bei denen man über die Güte des gewählten Pivotelements keine Aussage machen konnte, ist hier eine klare Aussage möglich: Bei der Pivotstrategie Median der Mediane werden die Daten im schlechtesten Fall im Verhältnis $\frac{1}{4}$ zu $\frac{3}{4}$ aufgeteilt. Somit kann die Datenmenge pro Prozessor am Ende auf keinem Prozessor auf Null fallen.

Die Pivotwahl ist also etwas schneller und auf jeden Fall besser geworden.

6.4 Datenaustausch

Die Umstellung auf die lokale Sortierung am Anfang hat den größten Einfluß auf den Datenaustausch. Denn neben dem oben erwähnten Vorteil bei der lokalen Sortierung handelt man sich einen anderen Nachteil ein: Die Daten, die zwischen den Prozessoren ausgetauscht werden, sind jetzt keine unsortierten Folgen mehr, die zu einer beliebigen Folge zusammengesetzt werden können, sondern es handelt sich jetzt um sortierte Teilfolgen, die entsprechend behandelt werden müssen. Ein empfangender Prozessor weiß nicht, von welchen Prozessoren er Daten empfängt. Aus diesem Grund müssen alle Datenpakete mit dem Absender gekennzeichnet werden. Weiterhin kann ein Prozessor von einem anderen Prozessor mehr als ein Datenpaket erhalten. Damit die Datenpakete wieder in der richtigen Reihenfolge zusammengesetzt werden können, erfolgt eine fortlaufende Numerierung der Datenpakete. Diese Maßnahmen stellen bislang keinen nennenswerten Mehraufwand dar. Zeitkritisch hingegen ist, daß die von mehreren Prozessoren eingetroffenen sortierten Teilfolgen zu einer neuen sortierten Teilfolge zusammengemischt werden müssen. Sei n die Anzahl der insgesamt empfangenen Elemente, dann muß beim Zusammensetzen von zwei Teilfolgen jedes Element einmal bewegt werden. Der Aufwand ist n. Liegen vier Teilfolgen vor, dann können jeweils zwei Teilfolgen zu einer Teilfolge und diese dann zur endgültigen Teilfolge zusammengesetzt werden. Jedes Element muß zweimal bewegt werden, der Aufwand ist folglich $2n$. Insgesamt hängt der Aufwand vom Logarithmus der eingetroffenen sortierten Teilfolgen ab ($O(n \log(n))$). In der Regel treffen aber nur zwei bis drei Teilfolgen bei einem Prozessor ein. Dieser zusätzliche Aufwand ist dann zwar gering, war aber beim ursprünglichen Quicksort mit lokaler Sortierung am Ende nicht vorhanden.

Andererseits wird dieser Mehraufwand teilweise kompensiert, denn vor dem Datenaustausch muß jeder Prozessor lokal ermitteln, wie viele Elemente kleiner oder gleich bzw. größer dem Pivotelement sind. Sei n die Anzahl der lokal vorhandenen Elemente, dann ist der Aufwand bei einer unsortierten Teilfolge $O(n)$, bei einer sortierten Teilfolge aber nur $O(\log(n))$.

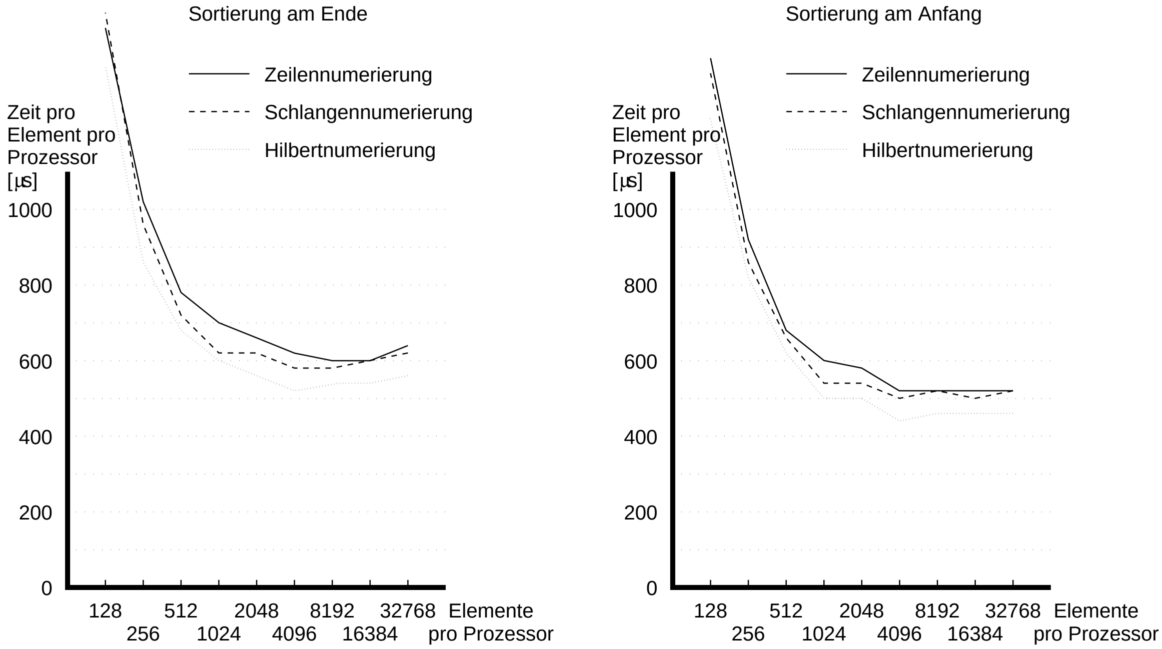


Abbildung 6.2: Vergleich der Sortierzeiten bei lokaler Sortierung am Ende und lokaler Sortierung am Anfang (Datenblätter 12 bis 14 und 41 bis 43)

Vergleicht man die Sortierzeiten bezüglich lokaler Sortierung am Anfang und lokaler Sortierung am Ende (Abbildung 6.2), so ergibt sich ein deutlicher Gewinn für die lokale Sortierung am Anfang. Dieser Gewinn ist unabhängig von der verwendeten Numerierung.

Betrachtet man beispielsweise bei der Hilbertnumerierung die Zeiten für die Kommunikation und die lokale Sortierung (Tabelle 6.1), so zeigen sich hier die Unterschiede zwischen diesen beiden Varianten. Die Zeit für die lokale Sortierung ist aufgrund der gleichmäßigen Verteilung der Daten stark gesunken. Die Zeit für die Kommunikation, die auch den Aufwand für das Zusammensetzen der eintreffenden, sortierten Daten beinhaltet, ist hingegen angestiegen. Dennoch verbleibt ein erheblicher Gewinn bei der lokalen Sortierung am Anfang.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
lokale Sortierung am Ende									
Zeit für Kommunikation [ms]	158	180	240	363	656	1120	2068	3553	6900
Zeit für lok. Sortierung [ms]	30	55	120	261	535	1136	2512	5605	12090
lokale Sortierung am Anfang									
Zeit für Kommunikation [ms]	160	190	274	408	752	1253	2430	4684	9421
Zeit für lok. Sortierung [ms]	19	30	69	141	319	670	1523	3180	6904

Tabelle 6.1: Zeit für Kommunikation und lokale Sortierung bei Hilbertnumerierung und bei lokaler Sortierung am Ende und lokaler Sortierung am Anfang (Datenblätter 14 und 43)

6.4.1 Reduzierung des Datenaustausches

Die Reduzierung des Datenaustausches ist vollkommen unabhängig davon, ob die lokale Sortierung am Anfang oder am Ende erfolgt. Sie kann deshalb, wie in Kapitel 5.3.1 beschrieben, eingesetzt werden und liefert auch die erwarteten Ergebnisse. Für die Schlangen- und Hilbertnumerierung wurde gleich die Datentauschinvertierung eingesetzt, da diese Optimierungstechnik in Verbindung mit der Reduzierung des Datenaustausches bei diesen Numerierungen Vorteile bietet.

6.4.2 Exakte Halbierung

Die exakte Halbierung kann auch hier, wie in Kapitel 5.3.2 beschrieben, eingesetzt werden. Die Sortierzeiten reduzieren sich ebenfalls erwartungsgemäß. Allerdings läßt sich durch das Erzwingen der exakten Halbierung eine schlechtere Verteilung der Daten nach der Sortierung erwarten. Dies ist allerdings nicht so problematisch, da es keinen negativen Einfluß mehr auf die lokale Sortierung hat, die am Anfang durchgeführt wird.

Erstaunlicherweise findet aber statt der Verschlechterung der Verteilung der Daten eine erhebliche Verbesserung statt (Abbildung 6.3). Dieses Phänomen wurde schon bei Quicksort mit lokaler Sortierung am Ende beobachtet. Die Ursache dafür liegt darin begründet, daß bei exakter Halbierung nur Zweierpotenzen als Segmentgrößen auftreten. Diese gerade Anzahl von Prozessoren läßt sich durch ein gutes Pivotelement leicht halbieren. Erzwingt man keine exakte Halbierung, so kann ein Segment mit P Prozessoren (P ist eine Zweierpotenz) trotz gutem Pivotelement auch einmal in zwei Segmente mit $\frac{P}{2}+1$ bzw. $\frac{P}{2}-1$ Prozessoren unterteilt werden. Ein Segment mit einer ungeraden Anzahl von Prozessoren kann nicht halbiert werden, so daß einem Segment immer überproportional viele Daten zugeteilt werden. Außerdem besteht nach der Aufteilung einer ungeraden Anzahl von Prozessoren wieder ein Segment mit einer ungeraden Anzahl von Prozessoren. Führt man diese Aufteilung bis zum Ende durch, so gibt es einen Prozessor, der jedesmal überproportional viele Daten erhalten hat. Durch die exakte Halbierung ist sichergestellt, daß sich die Segmente immer gut halbieren lassen und daß das obige Problem nicht mehr auftritt.

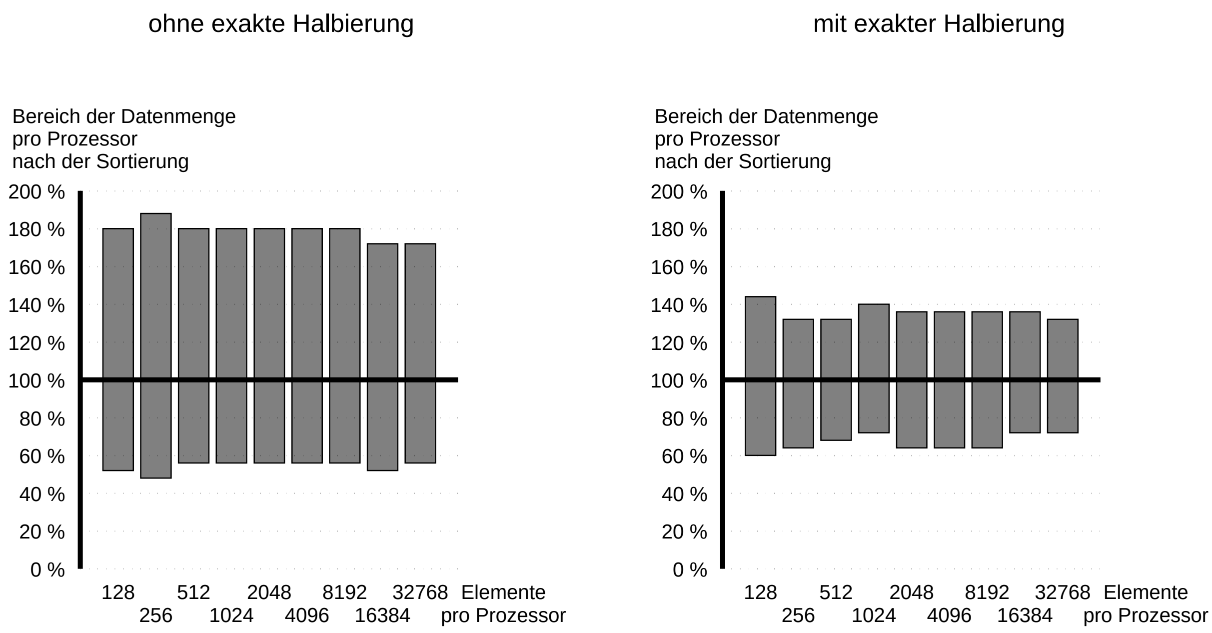


Abbildung 6.3: Vergleich der Verteilung der Daten ohne und mit exakter Halbierung (Datenblätter 46 und 49)

6.4.3 Exakte Partnerprozessoren

Die exakten Partnerprozessoren können, wie in Kapitel 5.3.4 beschrieben, eingesetzt werden. Durch den Einsatz dieser Optimierungstechnik ergibt sich noch ein besonderer Vorteil, denn jeder Prozessor hat genau zwei Teilfolgen zusammenzusetzen, nämlich die seines Partnerprozessors und seine eigene.

Anhand der Meßergebnisse (Abbildung 6.4) zeigt sich, daß durch die exakten Partnerprozessoren nochmals ein Gewinn zu verzeichnen ist. Durch das Wegfallen der *scan*-Operation, die in der Regel dafür sorgte, daß die Daten gleichmäßig auf alle Prozessoren in einem Segment verteilt werden, würde man jetzt eine schlechtere Verteilung der Daten erwarten. Bei der Quicksort-Variante mit lokaler Sortierung am Ende hat sich aber keine großartige Veränderung der Verteilung ergeben. Bei lokaler Sortierung am Anfang kommt es erstaunlicherweise sogar zu einer erheblichen Verbesserung der Verteilung der Daten. Dies wird aus Abbildung 6.5 deutlich.

Die Grund für diese Verbesserung der Verteilung kann nur darin liegen, daß bei dieser Quicksort-Variante mit lokaler Sortierung am Anfang erstens die Pivotstrategie bessere Pivotelemente liefert und zweitens nur sortierte Teilfolgen verschickt werden. Wendet man die Optimierungstechnik der exakten Partnerprozessoren nicht an, so sorgt die *scan*-Operation dafür, daß alle zu versendenden Daten gleichmäßig über die Prozessoren in einem Segment verteilt werden. Dementsprechend kann es sein, daß ein Prozessor die

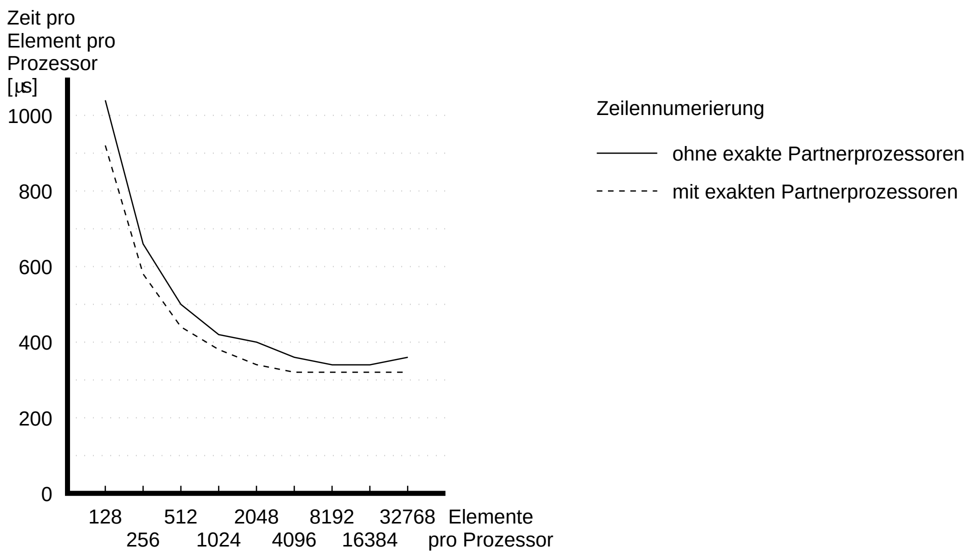


Abbildung 6.4: Sortierzeiten ohne und mit exakten Partnerprozessoren (Datenblätter 47 und 50)

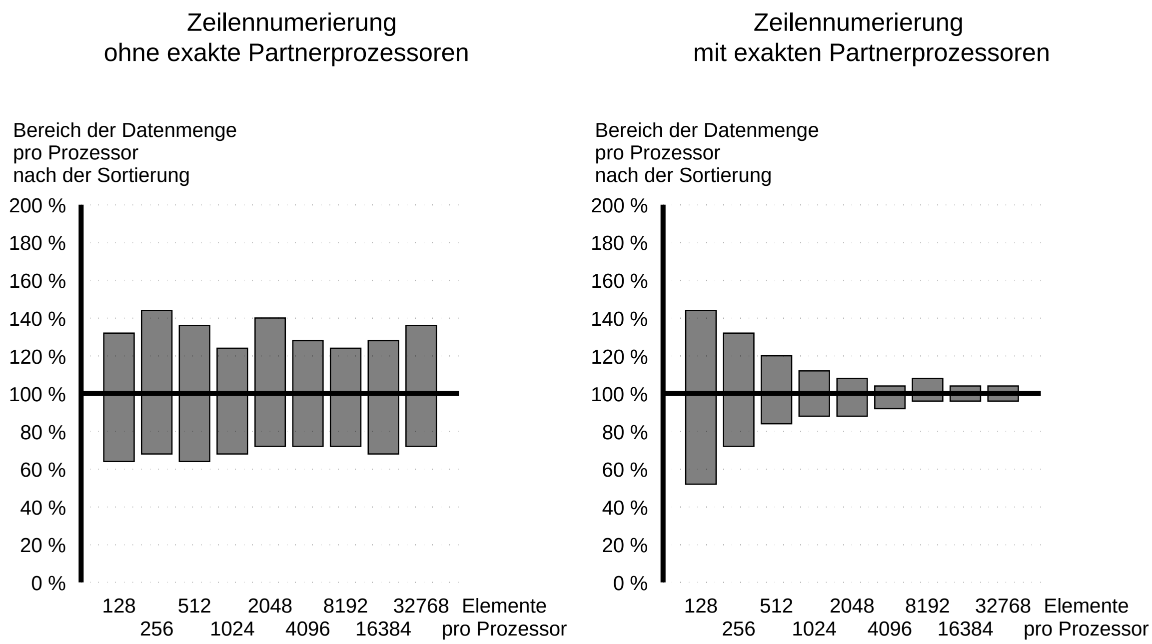


Abbildung 6.5: Verteilung der Daten ohne und mit exakten Partnerprozessoren (Datenblätter 47 und 50)

Daten, die er nicht behalten kann, an zwei verschiedene Prozessoren schicken muß. Da die Daten sortiert vorliegen, bekommt der eine Prozessor den einen Teil mit den höheren Daten, der andere Prozessor den mit den niedrigeren. Die Folge davon ist, daß nach einer Iteration die Datenmenge zwar gleichmäßig über alle Prozessoren verteilt wurde, die Daten auf einem Prozessor aber nicht mehr einer Zufallsverteilung entsprechen. Verwendet man hingegen die Optimierungstechnik der exakten Partnerprozessoren, so verschickt jeder Prozessor seine Daten, die trotz ihrer Sortierung eine Zufallsverteilung besitzen, an genau einen Prozessor. Dieser setzt die sortierte Folge mit seinen behaltenen Daten zusammen, so daß wieder eine sortierte Folge entsteht, deren Daten aber immer noch einer Zufallsverteilung entsprechen. Man kann also sagen, daß nach einer Iteration in einem Segment die Daten nicht gleichmäßig, aber zufällig über die Prozessoren verteilt sind. Verwendet man die Optimierungstechnik der exakten Partnerprozessoren nicht, so sind die Daten zwar gleichmäßig aber nicht mehr zufällig über alle Prozessoren verteilt. Deshalb führt der Einsatz der exakten Partnerprozessoren zu einer besseren Verteilung der Daten nach der Sortierung.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Zeit für Kommunikation [ms] bei ...									
... Zeilennumerierung	114	131	180	269	448	699	1299	2327	4591
... Schlangennumerierung	130	160	210	309	509	762	1360	2392	4702
... H-Numerierung	120	140	199	290	497	750	1345	2374	4689
... Shuffle-Numerierung	110	130	180	260	446	694	1294	2322	4604

Tabelle 6.2: Zeit für die Kommunikation bei exakten Partnerprozessoren (Datenblätter 50, 51, 53 und 55)

Betrachtet man auch die Schlangen-, Hilbert, H-, Z- und Shuffle-Numerierung, so ergeben sich nur geringe Unterschiede. Die Gesamtweglänge ist aber lediglich bei der Zeilen-, Schlangen-, H- und Shuffle-Numerierung minimal (siehe Datenblätter 50 bis 55). Obwohl fast alle gemessenen Daten gleich sind, ergeben sich bei den Kommunikationszeiten leichte Unterschiede (Tabelle 6.2).

Demzufolge liefern die Zeilennumerierung und die Shuffle-Numerierung die besten Sortierzeiten. Trotz nahezu gleicher Meßwerte gibt es aber einen gravierenden Unterschied bei der Aufteilung der Prozessoren (Tabelle 6.3).

Abhängig von der Segmentform ergeben sich zwar ganz unterschiedliche Zeiten für die Kommunikation in jeder Iteration (Abbildung 6.6), die Summe über alle Iterationen ist aber identisch.

Iteration	Segmentform bei Zeilennumerierung	Segmentform bei Shuffle-Numerierung
1	16×16	16×16
2	16×8	16×8
3	16×4	8×8
4	16×2	8×4
5	16×1	4×4
6	8×1	4×2
7	4×1	2×2
8	2×1	2×1

Tabelle 6.3: Segmentform in jeder Iteration bei Zeilen- und Shuffle-Numerierung (256 Prozessoren)

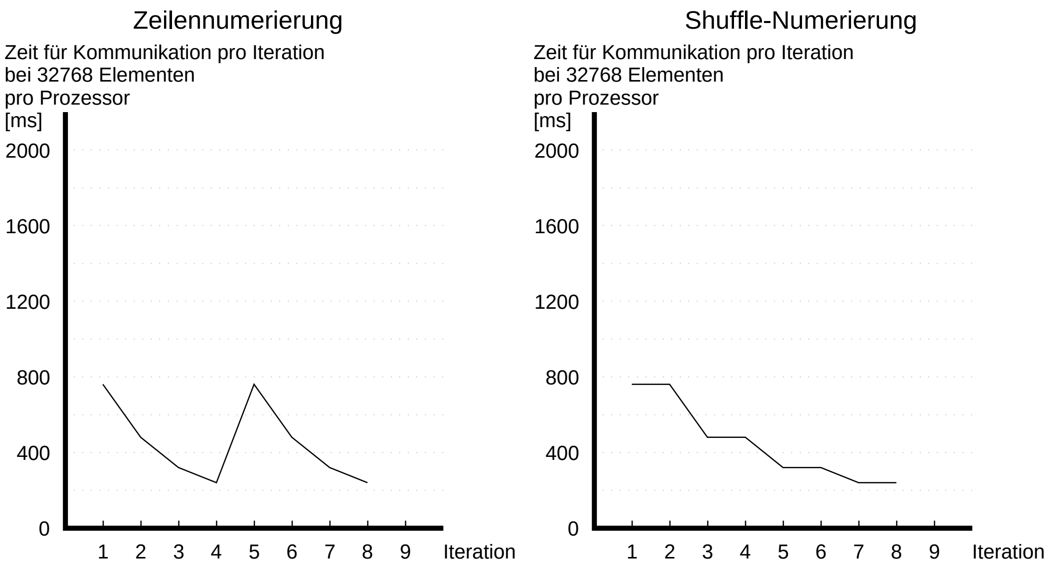


Abbildung 6.6: Zeit für Kommunikation pro Iteration bei Zeilen- und Shuffle-Numerierung (Datenblätter 50 und 55)

6.5 Zusammenfassung

Durch die lokale Sortierung am Anfang erzielt man wegen der gleichmäßigen Verteilung der Daten einen Zeitgewinn, der bei allen sinnvollen Kombinationen der verschiedenen Optimierungstechniken deutlich wird.

- Pivotwahl: Da nur noch mit sortierten Teilfolgen gearbeitet wird, wird als Pivotstrategie der Median der Mediane der einzelnen Prozessoren gewählt. Diese Strategie ist besser als alle bisher verwendeten Pivotstrategien.

- Reduzierung der Kommunikation: Diese Optimierungstechnik verringert den Kommunikationsaufwand deutlich. Es ergibt sich zwar nach der Sortierung eine etwas schlechtere Verteilung der Daten, was aber nicht mehr von Bedeutung ist, da die lokale Sortierung am Anfang durchgeführt wurde.

- Exakte Halbierung: Die regelmäßige Aufteilung der Prozessoren durch diese Optimierungstechnik führt zu einer gleichmäßigen Auslastung der Prozessoren und zu einer identischen Iterationstiefe auf allen Prozessoren. Durch den regelmäßigen Ablauf des Algorithmus kommt es sogar zu einer etwas besseren Verteilung der Daten nach der Sortierung.

- Exakte Partnerprozessoren: Bei dieser Optimierungstechnik hat jeder Prozessor nur genau einen Partnerprozessor, mit dem Daten ausgetauscht werden. Somit müssen nur genau zwei sortierte Teilfolgen zu einer neuen sortierten Teilfolge zusammengesetzt werden. Wendet man diese Optimierungstechnik nicht an, so treffen bei einem Prozessor in der Regel mehr als zwei sortierte Teilfolgen ein, die mit entsprechendem Aufwand zu einer neuen sortierten Teilfolge zusammengesetzt werden müssen. Erstaunlicherweise führt die Optimierungstechnik der exakten Partnerprozessoren zu einer erheblichen Verbesserung der Verteilung der Daten am Ende.

- Prozessornumerierung: Für verschiedene Optimierungstechniken empfehlen sich verschiedene Prozessornumerierungen gegebenenfalls mit oder ohne Datentauschinvertierung. Die Hilbertnumerierung erweist sich als die beste Numerierung, wenn der Algorithmus nicht regelmäßig abläuft, d.h. solange man keine exakte Halbierung oder exakte Partnerprozessoren einsetzt. Bei der Verwendung der exakten Partnerprozessoren führen mehrere Numerierungen zu einem minimalen Kommunikationsaufkommen. Dennoch liefern die Zeilen- und Shuffle-Numerierung die besten Sortierzeiten.

Durch die richtige Kombination der oben beschriebenen Optimierungstechniken lassen sich die Sortierzeiten gegenüber der ersten Iteration auf fast ein Drittel reduzieren. Die Abbildung 6.7 zeigt einen Vergleich der Sortierzeiten zwischen der ersten Konfiguration

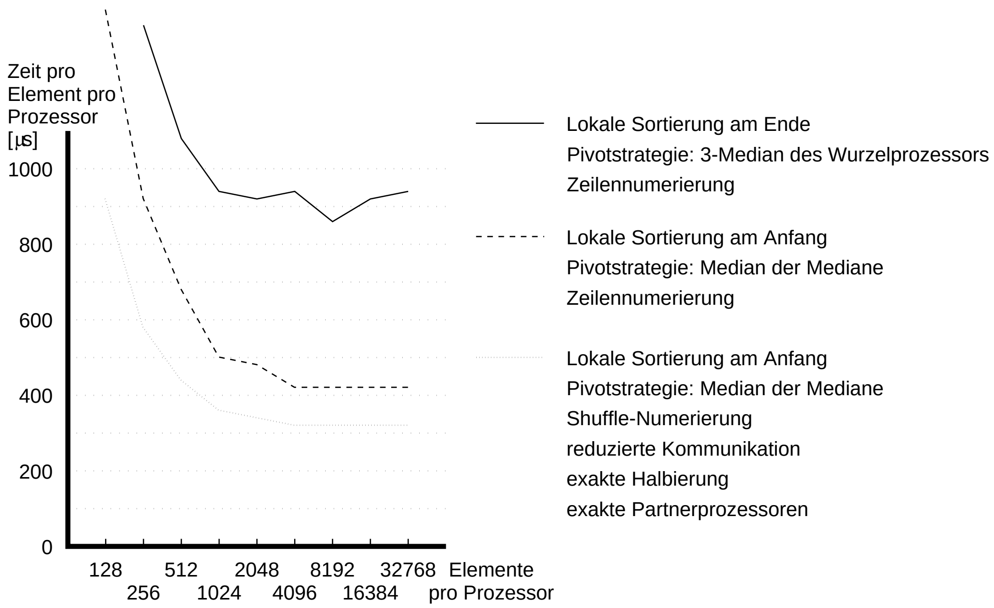


Abbildung 6.7: Sortierzeiten für die erste Konfiguration mit lokaler Sortierung am Ende, die erste Konfiguration und die beste Konfiguration mit lokaler Sortierung am Anfang (Datenblätter 1, 41 und 55)

und der besten Konfiguration bei lokaler Sortierung am Ende und der besten Konfiguration bei lokaler Sortierung am Anfang. Allein durch die Verlagerung der lokalen Sortierung vom Ende auf den Anfang verbesserten sich die Sortierzeiten um bis zu 20% .

Kapitel 7

Paralleles Quicksort mit mehreren Pivotelementen

Der Aufwand des parallelen Quicksorts setzt sich bekanntlich aus dem Aufwand für die lokale Sortierung und dem Aufwand für den Datenaustausch zusammen. Der Aufwand für den Datenaustausch wurde bereits bei der ersten Quicksort-Variante durch die Optimierungstechnik der exakten Partnerprozessoren optimiert. Sowohl die transportierte Datenmenge als auch die insgesamt zurückgelegte Weglänge ist dabei minimal. Bei der zweiten Quicksort-Variante wurde der Aufwand für die lokale Sortierung minimiert, indem die Sortierung auf den Anfang verlegt wurde. Der Aufwand für die lokale Sortierung ist damit minimal und kann nicht weiter verbessert werden. Deshalb ist es nur sinnvoll, für weitere Optimierungen beim Datenaustausch anzusetzen.

Die grundlegende Überlegung hierbei ist es, ein Datenelement mit weniger Sendeoperationen schneller zu dem Prozessor zu schicken, zu dem es letztendlich hingehört. Deshalb werden die Prozessoren nicht mehr in genau zwei Segmente unterteilt, sondern in eine beliebige Anzahl, die vorgegeben werden kann. Durch die Vorgabe von drei Pivotelementen erreicht man z.B. eine Vierteilung der Prozessoren in jeder Iteration, wofür man vorher zwei Iterationen benötigt hat. Die erhoffte Zeitersparnis liegt darin, daß die gesamten Daten bei einer Vierteilung genau einmal versandt werden, während bei der ursprünglichen Unterteilung in jeweils zwei Segmente die Daten insgesamt zweimal versandt werden mußten. Für die Gesamtweglänge sind ebenfalls kleine Verbesserungen zu erwarten. Andererseits handelt man sich mit der Unterteilung in mehrere Segmente einen erheblichen zusätzlichen Aufwand ein, denn erstens müssen mehrere Pivotelemente bestimmt werden und zweitens muß eine *scan*-Operation über einen Vektor durchgeführt werden, wobei die Vektorgröße von der Anzahl der Segmente anhängt.

Wählt man die Anzahl der Pivotelemente so, daß genau soviele Segmente entstehen wie es Prozessoren P gibt, so kommt man zu einer Version des Samplesort-Algorithmus. Das parallele Quicksort mit mehreren Pivotelementen ist also zwischen dem bislang betrachteten parallelen Quicksort mit nur einem Pivotelement und dem parallelen Sortierverfahren

Samplesort skalierbar. Ein gut implementiertes Samplesort wird dabei immer effizienter sein als paralleles Quicksort mit $P-1$ Pivotelementen, aber die entscheidende Frage ist, ob es zwischen einem und $P-1$ Pivotelementen irgendwo ein Optimum gibt. Zuvor werden aber alle bisher behandelten Optimierungstechniken in Verbindung mit mehreren Pivotelementen getestet, wobei die Anzahl der Pivotelemente erst einmal auf 3 festgelegt wird. Dadurch erreicht man eine Vierteilung der Prozessoren. Im Anschluß daran wird untersucht, ob eine andere Anzahl von Pivotelementen gegebenenfalls zu besseren Ergebnissen führt.

7.1 Lokale Sortierung

Bei den beiden vorangegangenen Varianten von Quicksort hat sich gezeigt, daß die lokale Sortierung am Anfang erheblich besser ist als die lokale Sortierung am Ende. Deshalb wird bei dem parallelen Quicksort mit mehreren Pivotelementen ausschließlich die lokale Sortierung am Anfang verwendet.

7.2 Prozessornumerierung

Es können alle Numerierungen gegebenenfalls mit oder ohne Datentauschinvertierung eingesetzt werden, die sich bislang als sinnvoll erwiesen haben.

7.3 Pivotwahl

Der Pivotwahl kommt eine ganz besondere Bedeutung zu. Bei den beiden vorangegangenen Varianten von Quicksort wurde jeweils immer nur ein Pivotelement benötigt, das die Daten in möglichst zwei gleich große Teilmengen teilen sollte. Für eine Unterteilung in mehrere Teilmengen werden mehrere Pivotelemente benötigt, um die Daten gleichmäßig auf die Segmente zu verteilen. Die Wahl von möglichen Pivotkandidaten auf einem Prozessor stellt kein Problem dar, denn die Daten liegen in sortierter Form vor. Durch einen Zugriff mit äquidistanten Abständen auf die lokalen Daten kann man einen Vektor von Pivotkandidaten zusammenstellen. Um eine faire Pivotwahl über alle Prozessoren hinweg zu gewährleisten, werden die lokal ermittelten Pivotkandidaten mit einer kollektiven Operation eingesammelt, um auf dem Wurzelprozessor die endgültigen Pivotelemente zu wählen. Diese Pivotstrategie wird als Multi-Median der Pivotkandidaten bezeichnet. Die Anzahl der erforderlichen kollektiven Operationen entspricht der Anzahl der benötigten Pivotelemente. Hier entsteht durch die Verwendung mehrerer Pivotelemente ein zusätzlicher Aufwand.

Nach der Wahl der Pivotelemente werden diese an alle Prozessoren im Segment mit Hilfe einer *broadcast*-Operation verschickt. Dies ist mit einer einzigen Operation möglich.

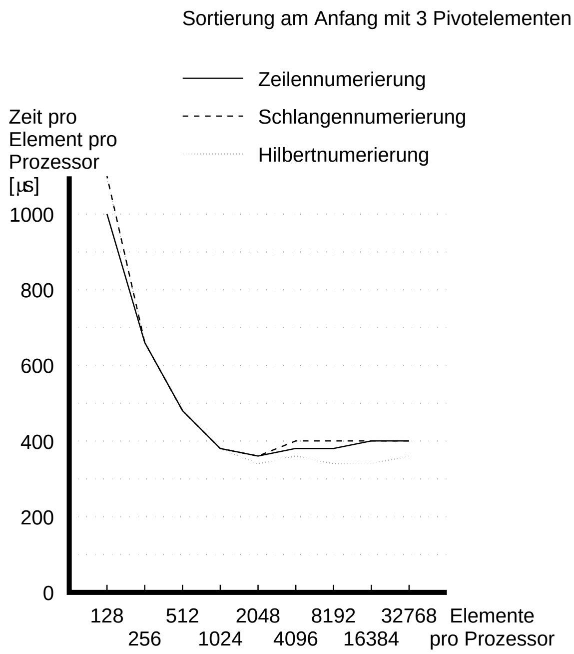


Abbildung 7.1: Sortierzeiten für verschiedene Numerierungen und drei Pivotelementen (Datenblätter 56 bis 58)

Bei den Numerierungen ergeben sich bei den größeren Datenmengen leichte Vorteile für die Hilbertnumerierung (Abbildung 7.1).

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
lokale Sortierung am Ende									
Anzahl Datenpakete	5757	5698	5699	5673	7214	11727	20075	36834	70184
Gesamtweglänge	23968	24315	24404	24358	29534	48470	82400	151431	286745
lokale Sortierung am Anfang									
Anzahl Datenpakete	5617	5639	5633	5636	7341	11494	19781	36514	69954
Gesamtweglänge	24423	24723	24428	24668	31324	48124	81763	151296	286555
lokale Sortierung am Anfang mit 3 Pivotelementen									
Anzahl Datenpakete	4763	4775	4772	4778	4779	7209	11591	20042	36735
Gesamtweglänge	21809	21863	21493	21501	21468	31948	50800	87116	158737

Tabelle 7.1: Datenpakete und Gesamtweglänge bei der Hilbertnumerierung für alle drei Quicksort-Varianten (Datenblätter 14, 43 und 58)

Ein Vergleich der Sortierzeiten mit den anderen Quicksort-Varianten (Abbildung 7.2) macht deutlich, daß die Verwendung mehrerer Pivotelemente nochmals einen erheblichen

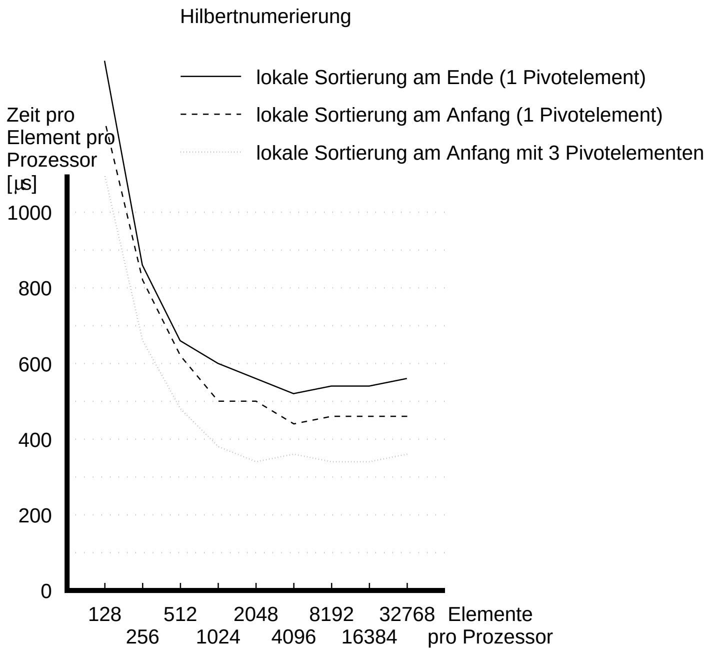


Abbildung 7.2: Sortierzeiten für alle drei Quicksort-Varianten bei Hilbertnumerierung (Datenblätter 14, 43 und 58)

Gewinn gegenüber den Quicksort-Varianten mit nur einem Pivotelement bringt. Die Anzahl der Datenpakete und die Gesamtweglänge sind bei Verwendung mehrerer Pivotelemente, wie erwartet, stark zurückgegangen (Tabelle 7.1).

7.4 Datenaustausch

Um den Datenaustausch zu optimieren, können alle schon bekannten Optimierungstechniken eingesetzt werden. Die Datentauschinvertierung wird ab sofort nur noch bei der Hilbertnumerierung eingesetzt. Bei der Schlangennumerierung wirkt sie sich hier nachteilig aus, da bei mehreren Pivotelementen die miteinander kommunizierenden Prozessoren durch die Datentauschinvertierung anscheinend nicht näher zusammenrücken.

7.4.1 Reduzierung des Datenaustausches

Bei der Reduzierung des Datenaustausches behält der Prozessor die Daten, die sonst innerhalb seines neuen Segments verschickt worden wären. Mit der Anzahl der Pivotelemente fällt die Menge der Daten, die nicht verschickt werden muß. Somit beträgt die Reduzierung des Datenaustausches bei drei Pivotelementen nur ein Viertel. Da es sich bei der ersparten Kommunikation jeweils um die kürzesten Kommunikationswege handelt, ist von dieser Optimierungstechnik bei vielen Pivotelementen keine nennenswerte Ersparnis zu erwarten. Davon abgesehen wird eine zusätzliche kollektive Operation benötigt.

Wie sich aus den Datenblättern 59 bis 61 im Anhang zeigt, ist der tatsächliche Gewinn für diese Optimierungstechnik nicht allzu groß. Bei den kleinen Datenmengen ergibt sich aufgrund der zusätzlichen kollektiven Operation sogar eine Verschlechterung. Ab 4096 Elementen pro Prozessor sind Verbesserungen von maximal 5% bei der Hilbertnumerierung zu beobachten.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
lokale Sortierung am Anfang mit 3 Pivotelementen									
Durchschn. Iterationstiefe	4.03	4.02	4.01	4.00	4.00	4.01	4.00	4.00	4.00

Tabelle 7.2: Durchschnittliche Iterationstiefe bei drei Pivotelementen und Hilbertnumerierung (Datenblatt 61)

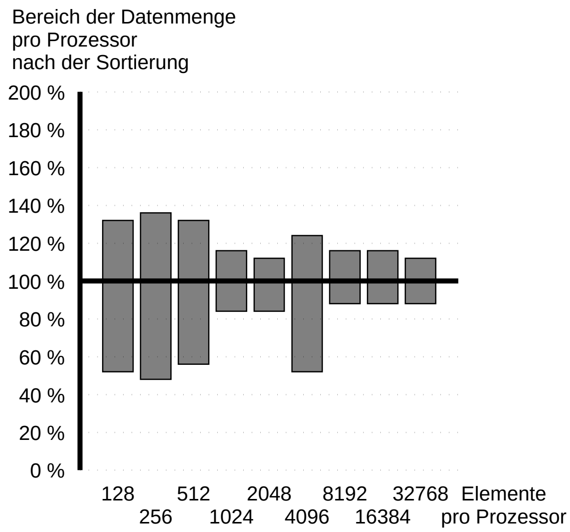


Abbildung 7.3: Verteilung der Daten nach der Sortierung bei drei Pivotelementen und Hilbertnumerierung (Datenblatt 61)

Interessanter ist schon die Beobachtung der durchschnittlichen Iterationstiefe. In den meisten Fällen wird die minimale Iterationstiefe von 4,00 erreicht (Tabelle 7.2). Kommt es einmal nicht zu einer optimalen Aufteilung der Prozessoren, so verschlechtert sich die Verteilung der Daten nach der Sortierung (Abbildung 7.3).

Führt man Messungen durch, um zu ermitteln, für welche Anzahl von Pivotelementen die Sortierzeiten am besten sind, so ergibt sich das in Tabelle 7.3 dargestellte Bild.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
1 Pivotelement									
Zeit pro Element pro Prozessor [μs]	1507	929	624	507	465	405	394	360	379
2 Pivotelemente									
Zeit pro Element pro Prozessor [μs]	1249	757	552	459	416	380	365	369	407
3 Pivotelemente									
Zeit pro Element pro Prozessor [μs]	1241	734	505	379	340	342	337	336	344
4 Pivotelemente									
Zeit pro Element pro Prozessor [μs]	1186	710	507	409	353	332	349	357	362
5 Pivotelemente									
Zeit pro Element pro Prozessor [μs]	1343	816	552	427	366	337	364	349	376

Tabelle 7.3: Sortierzeiten für verschiedene Anzahlen von Pivotelementen bei Hilbertnumerierung (Datenblätter 45, 62, 61, 63 und 64)

In der Tabelle 7.3 und der Abbildung 7.3 zeigt sich, daß drei Pivotelemente die besten Sortierzeiten liefern, wenn die optimale durchschnittliche Iterationstiefe von 4,00 erreicht wird. Liegt die durchschnittliche Iterationstiefe etwas höher, so empfehlen sich vier Pivotelemente, so daß man dann auch mit 4,00 Iterationen auskommen.

Eine gute Aufteilung der Prozessoren spielt also eine entscheidende Rolle für den Gesamtaufwand des Algorithmus.

7.4.2 Exakte Teilung

Bei den bisherigen Quicksort-Varianten wurde die Optimierungstechnik der exakten Halbierung verwendet, um eine gleichmäßige Aufteilung der Prozessoren zu erzwingen. Hat man aber mehrere Pivotelemente, so kann man nicht mehr von Halbierung sprechen, sondern nur noch von Teilung. Eine exakte Teilung ist nur dann möglich, wenn die Gesamtzahl der Prozessoren mehrfach durch die Anzahl der Pivotelemente +1 ohne Rest teilbar ist. Lediglich in der letzten Iteration dürfen gegebenenfalls weniger Prozessoren zur Verfügung stehen als die Anzahl der Pivotelemente +1. Da hier für die Anzahl der Prozessoren nur Zweierpotenzen zum Einsatz kommen, ist eine exakte Teilung lediglich mit $2^n - 1$ Pivotelementen möglich.

Die Wirkung der exakten Teilung tritt nur dann in Kraft, wenn die Pivotelemente nicht von alleine für die exakte Teilung sorgen. In diesen Einzelfällen wird die durchschnittliche Iterationstiefe auf das Optimum gedrückt.

7.4.3 Exakte Partnerprozessoren

Die Optimierungstechnik der exakten Partnerprozessoren kann nur verwendet werden, wenn eine exakte Teilung der Prozessoren vorliegt. Nur so hat jeder Prozessor in jedem Segment exakt einen Partner, mit dem er Daten austauschen kann. Die in Frage kommende Anzahl von Pivotelementen ist somit stark eingeschränkt. Damit die kommunizierenden Prozessoren möglichst nahe auf horizontalen und vertikalen Achsen liegen, kommt der Prozessornumerierung eine entscheidende Bedeutung zu.

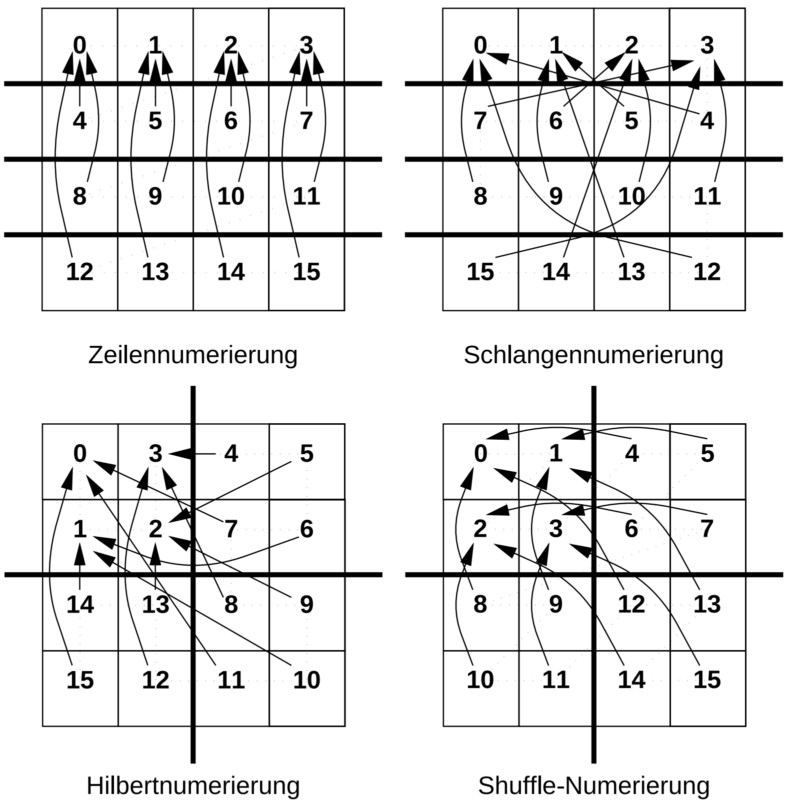


Abbildung 7.4: Kommunikationswege bei drei Pivotelementen und bei verschiedenen Numerierungen

In der Abbildung 7.4 sind die Kommunikationswege für die Daten dargestellt, die in das erste Segment geschickt werden. Bei der Zeilennumerierung ergeben sich dabei Kommunikationswege, die alle auf einer Achse liegen, aber unterschiedliche Längen haben. Es findet entweder ausschließlich Kommunikation in vertikaler oder horizontaler Richtung statt. Somit werden immer nur zwei der vier Ein-/Ausgänge des Transputers genutzt. Die Schlangennumerierung besitzt sehr viele sich überschneidende Kommunikationswege, die zudem sehr lang sind. Die Kommunikationswege bei der Hilbertnumerierung sind unregelmäßig und überschneiden sich häufig. Bei der Shuffle-Numerierung ergeben sich zwar auch Überschneidungen, aber die meisten Kommunikationswege laufen entlang einer

Achse, sind gleich lang und ergeben ein regelmäßiges Kommunikationsmuster. Alle Ein-/Ausgänge des Transputers werden dabei gleichmäßig genutzt. Die Shuffle-Numerierung hat den weiteren Vorteil, daß nach einer Teilung wieder Quadrate mit den gleichen Eigenschaften zur Verfügung stehen, was bei der Zeilen- und Schlangennumerierung nicht der Fall ist.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
Zeilennumerierung									
Sortierzeit [μs]	781	507	386	312	292	314	313	310	320
Gesamtweglänge	12800	12800	12800	12800	12800	22121	35928	62385	114410
Schlangennumerierung									
Sortierzeit [μs]	921	585	429	341	317	339	332	327	336
Gesamtweglänge	16896	16896	16896	16896	16896	29050	47350	82292	151079
Hilbertnumerierung									
Sortierzeit [μs]	937	569	419	332	299	313	312	311	325
Gesamtweglänge	17376	17376	17376	17376	17376	30469	49274	85123	155717
Shuffle-Numerierung									
Sortierzeit [μs]	781	503	372	302	283	303	307	308	321
Gesamtweglänge	15360	15360	15360	15360	15360	26790	43506	75266	137601

Tabelle 7.4: Sortierzeiten und Gesamtweglänge für verschiedene Numerierungen bei drei Pivotelementen und exakten Partnerprozessoren (Datenblätter 66 bis 69)

Betrachtet man die Gesamtweglänge in Tabelle 7.4, so stellt sich die Zeilennumerierung als die beste heraus, gefolgt von der Shuffle-Numerierung. Die anderen Numerierungen spielen keine Rolle. Bei den Sortierzeiten hat die Shuffle-Numerierung wegen ihres rekursiven Aufbaus aber die besten Ergebnisse.

Auch bei dem Einsatz der Optimierungstechnik der exakten Partnerprozessoren stellt sich die Frage, für wie viele Pivotelemente die besten Sortierzeiten erzielt werden können. Es kommt allerdings nicht jede Anzahl in Betracht, da die exakte Teilung gewährleistet werden muß. Aus diesem Grund können nur drei, sieben oder 15 Pivotelemente gewählt werden. Die Sortierzeiten, die Anzahl der Datenpakete und die Gesamtweglänge sind in der Tabelle 7.5 wiedergegeben. Dabei stellt sich kein eindeutiges Ergebnis heraus. Allerdings läßt sich feststellen, daß die Konfiguration mit den wenigsten Datenpaketen die besten Sortierzeiten liefert. Die Zeitunterschiede bei den großen Datenmengen sind bei weitem nicht so groß wie bei den kleinen Datenmengen, so daß man für den universellen Einsatz drei Pivotelemente empfehlen kann.

7.5 Zusammenfassung

Durch die Verwendung mehrerer Pivotelemente erzielt man einen Zeitgewinn, der bei allen sinnvollen Kombinationen der verschiedenen Optimierungstechniken deutlich wird.

Datenmenge/Prozessor	128	256	512	1024	2048	4096	8192	16384	32768
3 Pivotelemente									
Sortierzeit [μs]	781	503	372	302	283	303	307	308	321
Anzahl Datenpakete	4096	4096	4096	4096	4096	7019	11421	19919	36600
Gesamtweglänge	15360	15360	15360	15360	15360	26790	43506	75266	137601
7 Pivotelemente									
Sortierzeit [μs]	1022	620	431	332	300	284	307	303	315
Anzahl Datenpakete	5120	5120	5120	5120	5120	5816	9875	16445	29114
Gesamtweglänge	26624	26624	26624	26624	26624	27328	47330	77489	134747
15 Pivotelemente									
Sortierzeit [μs]	1569	933	583	400	329	293	292	306	313
Anzahl Datenpakete	8192	8192	8192	8192	8192	8192	8192	14034	22888
Gesamtweglänge	51200	51200	51200	51200	51200	51200	51200	89284	144688

Tabelle 7.5: Sortierzeiten, Anzahl der Datenpakete und Gesamtweglänge für drei, sieben und 15 Pivotelementen (Datenblätter 69 bis 71)

- Pivotwahl: Für die Ermittlung mehrerer Pivotelemente steht nur eine Pivotstrategie (Multi-Median der Pivotkandidaten) zur Verfügung. Wegen der Durchführung der lokalen Sortierung am Anfang sind die so ermittelten Pivotelemente recht gut und sorgen für eine ziemlich gleichmäßige Aufteilung der Daten.

- Reduzierung des Datenaustausches: Durch diese Optimierungstechnik wird der Kommunikationsaufwand verringert. Die Einsparung sinkt aber mit der Anzahl der Pivotelemente, so daß diese Optimierungstechnik hier keine große Verbesserung bringt.

- Exakte Teilung: Durch diese Optimierungstechnik werden die Segmente gleichmäßig unterteilt. Allerdings ist diese Optimierungstechnik nur einsetzbar, wenn die Gesamtzahl der Prozessoren durch die Anzahl der Pivotelemente +1 mehrfach teilbar ist. Durch den regelmäßigen Ablauf des Algorithmus kommt es zu einer gleichmäßigen Auslastung der Prozessoren und zu einer etwas besseren Verteilung der Daten nach der Sortierung.

- Exakte Partnerprozessoren: Bei dieser Optimierungstechnik hat jeder Prozessor in jedem Segment genau einen Partnerprozessor, mit dem Daten ausgetauscht werden. Das Kommunikationsaufkommen wird dadurch reduziert. Diese Optimierungstechnik der exakten Partnerprozessoren führt zu einer Reduzierung der Sortierzeiten und zu einer erheblichen Verbesserung der Verteilung der Daten.

- Prozessornumerierung: Für verschiedene Optimierungstechniken empfehlen sich verschiedene Prozessornumerierungen gegebenenfalls mit oder ohne Datentauschinvertierung. Die Hilbertnumerierung erweist sich als die beste Numerierung, wenn der Algorithmus nicht regelmäßig abläuft, d.h. solange man keine exakte Teilung oder exakte Partnerprozessoren einsetzt. Bei der Verwendung der exakten Partnerprozessoren führen mehrere Numerierungen zu einem minimalen Kommunikationsaufkommen. Dennoch liefern die Zeilen- und Shuffle-Numerierung die besten Sortierzeiten.

- Anzahl der Pivotelemente: Verzichtet man auf die exakte Teilung, so kann man jede beliebige Anzahl von Pivotelementen wählen. Die besten Sortierzeiten haben sich aber bei ungefähr drei Pivotelementen eingestellt. Dies entspricht jeweils einer Vierteilung der Prozessoren.

  Setzt man die exakte Teilung und die exakten Partnerprozessoren ein, so kommen nur $2^n - 1$ Pivotelemente in Frage. Auch hier erweisen sich drei Pivotelemente besser als ein Pivotelement bzw. sieben oder 15 Pivotelemente. Bei einer rekursiv aufgebauten Numerierung wie der Shuffle-Numerierung führt dies auch physikalisch zu einer exakten Viertelung des Prozessorgitters, so daß ein Segment immer aus einem Quadrat von Prozessoren besteht. Die Kommunikation erfolgt gleichmäßig in horizontale und vertikale Richtung, so daß die Auslastung für die Ein-/Ausgänge der Transputer auch gleichmäßig ist.

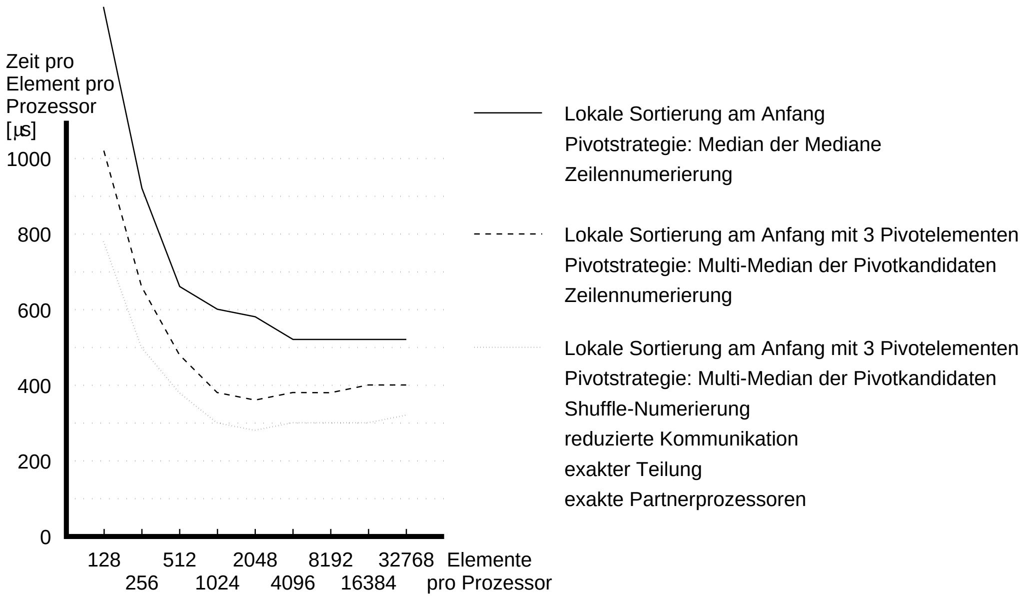


Abbildung 7.5: Sortierzeiten für die erste Konfiguration mit lokaler Sortierung am Anfang sowie für die erste und die beste Konfiguration mit lokaler Sortierung am Anfang mit drei Pivotelementen (Datenblätter 41, 56 und 69)

Wie Abbildung 7.5 zeigt, ist selbst die erste Konfiguration mit drei Pivotelementen schon erheblich besser als die erste Konfiguration mit lokaler Sortierung am Anfang. Durch den Einsatz reduzierter Kommunikation, exakter Teilung, exakten Partnerprozessoren und der Shuffle-Numerierung lassen sich die Sortierzeiten nochmals um ca. 20% reduzieren.

Vergleicht man die jeweils besten Konfigurationen der drei verschiedenen Quicksort-Varianten, so ergibt sich ein recht großer Gewinn durch die Verlagerung der lokalen Sortierung auf den Anfang (Abbildung 7.6). Der Gewinn durch die Verwendung mehrerer Pivotelemente fällt dagegen etwas geringer aus.

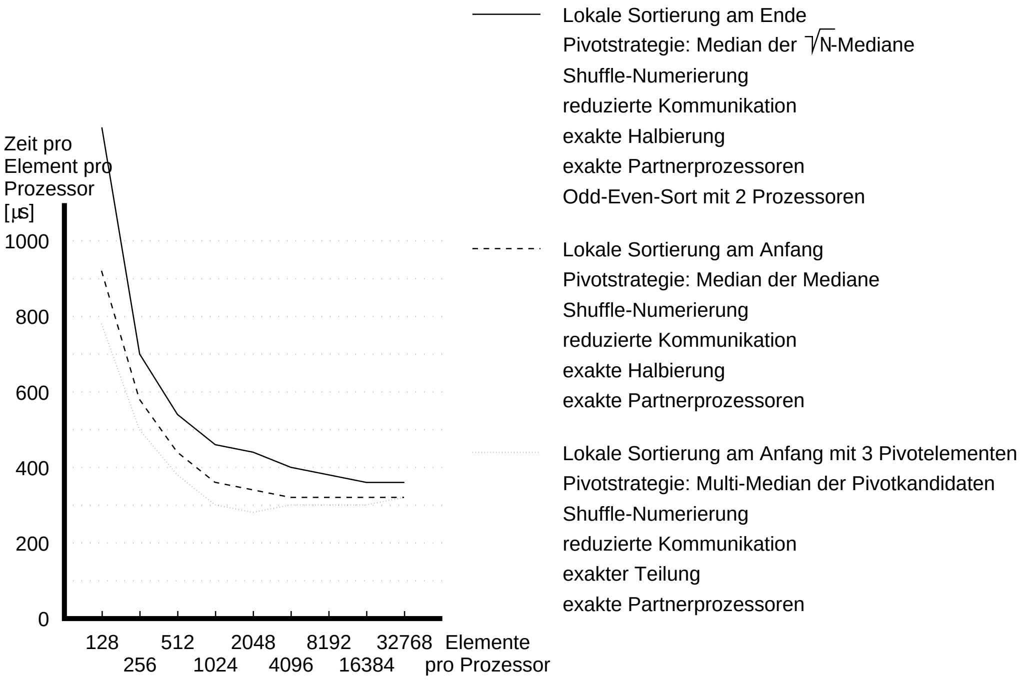


Abbildung 7.6: Sortierzeiten für die drei jeweils besten Quicksort-Varianten (Datenblätter 37, 55 und 69)

Als Fazit läßt sich festhalten, daß die Verwendung mehrerer Pivotelemente bei allen Datenmengen uneingeschränkt empfehlenswert ist. Dabei haben sich drei Pivotelemente als sinnvoll erwiesen.

Kapitel 8

Zusammenfassung und Ausblick

Paralleles Quicksort hat sich bei diesen Untersuchungen als ein sehr gutes Sortierverfahren erwiesen. Dennoch hat sich gezeigt, daß die Leistungsfähigkeit des Algorithmus stark von der Wahl der Quicksort-Variante und der Optimierungstechniken abhängt.

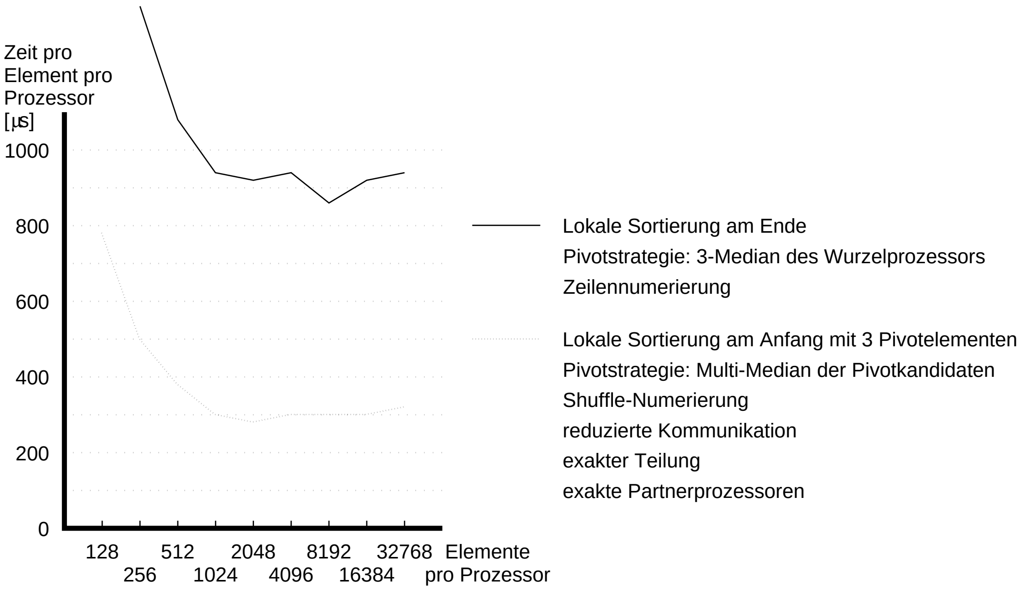


Abbildung 8.1: Sortierzeiten für die erste und die beste Konfiguration (Datenblätter 1 und 69)

Die Abbildung 8.1 zeigt noch einmal den Unterschied zwischen den Sortierzeiten bei der ersten Konfiguration und der letztendlich besten Konfiguration. Die Sortierzeiten konnten in der Regel auf unter $\frac{1}{3}$ gesenkt werden.

8.1 Die Wahl der Quicksort-Variante

Unabhängig von den Optimierungstechniken wurden drei Varianten des parallelen Quicksorts implementiert und untersucht:

- Quicksort mit lokaler Sortierung am Ende
- Quicksort mit lokaler Sortierung am Anfang
- Quicksort mit mehreren Pivotelementen

Der Unterschied zwischen der ersten und der zweiten Variante liegt darin, daß die lokale Sortierung auf den Anfang vorgezogen wurde. Aus den Abbildungen 8.2 und 8.3 wird deutlich, daß die lokale Sortierung am Anfang gegenüber der lokalen Sortierung am Ende immer Vorteile hat. Dieses Ergebnis ist unabhängig davon, ob irgendwelche Optimierungstechniken eingesetzt werden oder nicht. Für die dritte Variante wurde deshalb auch die lokale Sortierung am Anfang gewählt.

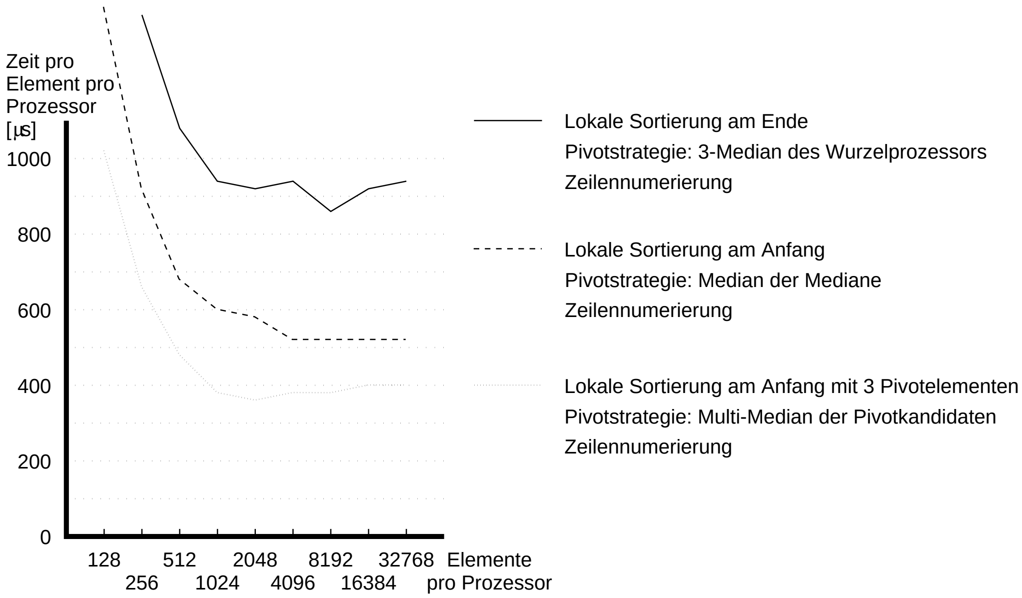


Abbildung 8.2: Sortierzeiten für die drei Quicksort-Varianten ohne Verwendung weiterer Optimierungstechniken (Datenblätter 1, 41 und 56)

Bei der dritten Variante wurden statt einem Pivotelement mehrere Pivotelemente eingesetzt. Dadurch ist es möglich, die Prozessoren in mehr als zwei Segmente aufzuteilen und die Daten mit weniger Sendeoperationen zu dem Prozessor zu schicken, zu dem sie letztendlich hingehören. Insgesamt ergibt sich durch die Verwendung mehrerer Pivotelemente ein geringeres Kommunikationsaufkommen. Aus den Abbildungen 8.2 und 8.3 wird

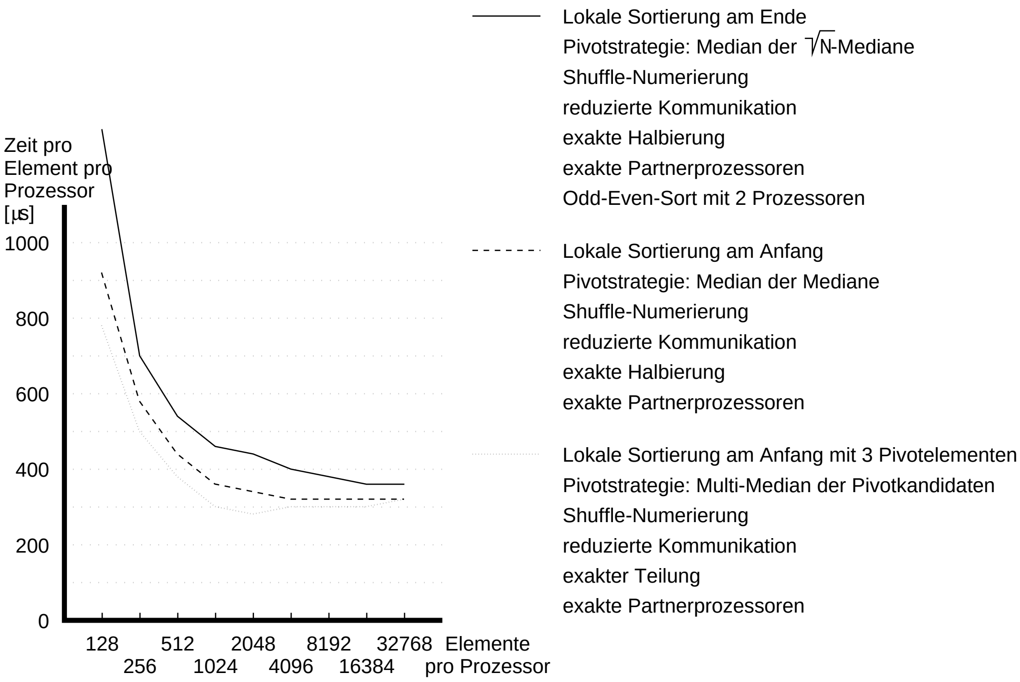


Abbildung 8.3: Sortierzeiten für die drei jeweils besten Quicksort-Varianten (Datenblätter 37, 55 und 69)

deutlich, daß bei Verwendung mehrerer Pivotelemente die Sortierzeiten gegenüber den Sortierzeiten der anderen beiden Varianten immer besser sind, egal ob bzw. welche Optimierungstechniken eingesetzt werden.

Insgesamt erweist sich die Quicksort-Variante mit mehreren Pivotelementen als der beste Ansatz, die Sortierzeiten zu minimieren. Bei den Untersuchungen über die richtige Anzahl der Pivotelemente haben sich drei Pivotelemente als optimal herausgestellt.

8.2 Die Wahl der Optimierungstechniken

Die Wirkungen der einzelnen Optimierungstechniken sind bereits in den Kapiteln 5 bis 7 für jede Quicksort-Variante ausführlich beschrieben worden. Leider gibt es nicht eine einzige Optimierungstechnik, die zu einer drastischen Verbesserung der Sortierzeiten führt, sondern jede einzelne Optimierungstechnik liefert einen kleinen Beitrag, so daß letztendlich ein großer Zeitgewinn entsteht. In der Abbildung 8.4 sind für sechs Konfigurationen die Sortierzeiten dargestellt. Zur Veranschaulichung wurde die Quicksort-Variante mit lokaler Sortierung am Ende gewählt, weil die Wirkung der einzelnen Optimierungstechniken dort besonders deutlich wird. Der Zeitgewinn, der durch jede einzelne Optimierungstechnik erzielt wird, stellt keine Aussage über die Güte einer Optimierungstechnik dar.

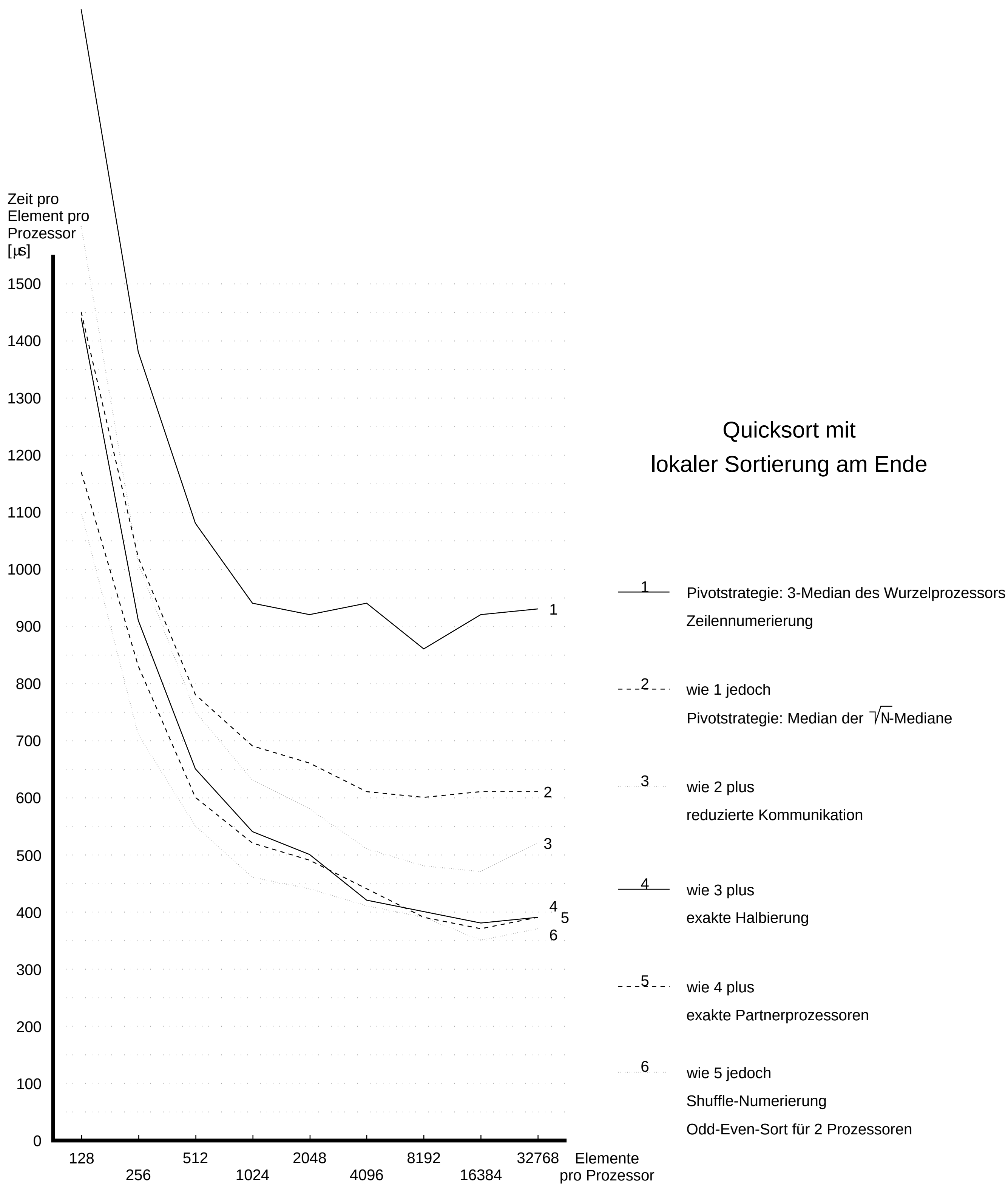


Abbildung 8.4: Sortierzeiten für die erste Quicksort-Varianten unter Verwendung verschiedener Optimierungstechniken (Datenblätter 1, 12, 21, 24, 30 und 36)

8.2.1 Pivotwahl

Die Pivotwahl hat eine recht große Bedeutung bei der Verbesserung der Sortierzeiten. Das wurde besonders im Kapitel 4 bei der Variante mit lokaler Sortierung am Ende deutlich. Dort wurden mehrere Pivotstrategien untersucht. Dabei hat sich als Ergebnis herausgestellt, daß der zusätzliche Aufwand für die Bestimmung eines guten Pivotelements sich durch die bessere Datenaufteilung mehr als bezahlt macht. Bei der zweiten und dritten Quicksort-Variante, bei denen die lokale Sortierung am Anfang erfolgt, ist die Bestimmung eines guten Pivotelements viel einfacher, da die Daten vor der Pivotwahl auf jedem Prozessor sortiert zur Verfügung stehen. Daher rührt auch die starke Verbesserung der Sortierzeiten beim Übergang von der ersten zur zweiten Quicksort-Variante. Bei der zweiten und dritten Quicksort-Variante stand jeweils nur eine Pivotstrategie zur Verfügung, so daß hier keine Wahlmöglichkeit besteht.

8.2.2 Reduzierte Kommunikation

Die Optimierungstechnik der reduzierten Kommunikation ist vollkommen unabhängig von der jeweiligen Quicksort-Variante. Es ergibt sich jeweils eine Einsparung an Kommunikationsaufwand, weil jeder Prozessor die Daten, die eigentlich schon im richtigen Segment liegen, für sich behält, und nicht an einen anderen Prozessor in gleichen Segment verschickt. Bei den ersten beiden Quicksort-Varianten kann man somit ungefähr die Hälfte der versandten Daten einsparen, allerdings handelt es sich dabei um die kürzeren Kommunikationswege. Bei der Quicksort-Variante mit mehreren Pivotelementen sind die Segmente und somit auch die Datenmengen kleiner, die dadurch nicht verschickt werden müssen. Nachteilige Auswirkungen hat diese Optimierungstechnik nicht, so daß sie uneingeschränkt eingesetzt werden sollte.

8.2.3 Exakte Halbierung/Teilung

Verfügt man über eine gute Pivotwahl, so sorgt das Pivotelement dafür, daß die Daten und die Prozessoren in der Regel halbiert werden. Kommt es einmal nicht zu dieser genauen Halbierung, so muß mindestens ein Prozessor eine Iteration mehr ausführen, was entscheidend für die Gesamtlaufzeit sein kann. Die Optimierungstechnik der exakten Halbierung erzwingt in solch einem Fall die exakte Halbierung, so daß alle Prozessoren exakt die gleiche Anzahl von Iterationen ausführen müssen.

Bei der dritten Quicksort-Variante mit mehreren Pivotelementen heißt diese Optimierungstechnik exakte Teilung, weil mehr als zwei Segmente entstehen.

Diese Optimierungstechnik greift also dann ein, wenn die Pivotelemente nicht von selbst für die exakte Halbierung/Teilung sorgen. Somit kann auch diese Optimierungstechnik uneingeschränkt eingesetzt werden.

8.2.4 Exakte Partnerprozessoren

Die Optimierungstechnik der exakten Partnerprozessoren führt dazu, daß jeder Prozessor mit genau einem Prozessor in jedem Segment Daten austauscht. Voraussetzung für den Einsatz der exakten Partnerprozessoren ist die exakte Halbierung/Teilung der Prozessoren. Der entscheidende Vorteil dieser Optimierungstechnik ist, daß man ein regelmäßiges Kommunikationsmuster erhält. Durch die Wahl der richtigen Prozessornumerierung ergeben sich gleichmäßige und gleich lange Kommunikationswege, was sich in der Reduzierung des Kommunikationsaufwands bemerkbar macht. Diese Optimierungstechnik sollte in jedem Fall mit der exakten Halbierung/Teilung eingesetzt werden.

8.2.5 Prozessornumerierung

Für die Art der Prozessornumerierung läßt sich leider keine Numerierung als universell beste Numerierung angeben. Bei verschiedene Konfigurationen haben sich verschiedenen Numerierungen als die besten erwiesen. Solange der Quicksort-Algorithmus nicht durch die Optimierungstechnik der exakten Halbierung/Teilung zu einem regelmäßigen Ablauf gezwungen wird, stellt sich die Hilbert-Numerierung als Favorit dar. Bei der Verwendung der exakten Partnerprozessoren führen mehrere Numerierungen zu einem minimalen Kommunikationsaufkommen. Dennoch liefern die Zeilen- und Shuffle-Numerierung die besten Sortierzeiten. Bei der Verwendung mehrerer Pivotelemente stellt sich die Shuffle-Numerierung als beste Numerierung dar.

8.2.6 Datentauschinvertierung

Die Optimierungstechnik der Datentauschinvertierung ist ganz stark abhängig von der verwendeten Prozessornumerierung, der Quicksort-Variante und von den Optimierungstechniken. Lediglich bei der Hilbertnumerierung scheint es immer sinnvoll zu sein, diese Optimierungstechnik einzusetzen. Für alle anderen Numerierungen muß man es gegebenenfalls ausprobieren.

8.3 Die beste Konfiguration für paralleles Quicksort

Die beste Konfiguration für paralleles Quicksort sieht wie folgt aus:

- lokale Sortierung am Anfang
- drei Pivotelemente
- Pivotstrategie: Multi-Median der Pivotkandidaten

- Shuffle-Numerierung
- reduzierte Kommunikation
- exakte Teilung
- exakte Partnerprozessoren

Unter dem Begriff paralleles Quicksort wird ab sofort die oben beschriebene Konfiguration verstanden.

8.4 Vergleich zwischen parallelem und sequentiellem Quicksort

Für die Einschätzung der Effizienz eines parallelen Algorithmus ist der Speedup ein gutes Kriterium. Hierbei wird bei gleicher Problemgröße die Ausführungszeit auf einem Prozessor mit der Ausführungszeit auf allen Prozessoren verglichen. Als sequentielles Quicksort auf einem Prozessor wird die qsort-Funktion aus der C-Standard-Bibliothek des Transputers verwendet. Da ein Prozessor des Transputers nur über 4 MB Speicher verfügt, ist es lediglich möglich, Messungen bis zu 524288 Ganzzahlen mit 32 Bit durchzuführen. Oberhalb dieser Grenze müssen die Werte durch Extrapolation bestimmt werden. Der Aufwand für sequentielles Quicksort ist im Durchschnitt $O(n \log(n))$. Als Ansatz wird deshalb die Funktion

$$\tilde{t}(n) = a_1 n + a_2 n \mathrm{ld}(n)$$

gewählt. Durch Äquivalenzumformung und durch die Substitution $m = \mathrm{ld}(n)$ ergibt sich die Gleichung

$$\frac{\tilde{t}(2^m)}{2^m} = a_1 + a_2 m$$

Mit der Methode der kleinsten Quadrate ergeben sich für a_1 und a_2 die folgenden Werte:

$$a_1 = -0,0145209$$
$$a_2 = 0,0138777$$

Die Ergebnisse der Extrapolation sind in der Tabelle 8.1 dargestellt. Der maximale Fehler der extrapolierten Funktion $\tilde{t}$ liegt bei +0,32% gegenüber den gemessenen Werten $t(n)$. Somit ergibt sich im Meßbereich eine sehr gute Approximation, so daß man die extrapolierten Werte als gut annehmen kann.

n	$t(n)$ [ms]	$m = \mathrm{ld}(n)$	$\frac{t(2^m)}{2^m}$	$\frac{\tilde{t}(2^m)}{2^m}$	$\tilde{t}(n)$ [ms]	Fehler [%]
2048	282	11	0,137695	0,138134	283	+0,32
4096	623	12	0,152100	0,152012	623	-0,06
8192	1360	13	0,166016	0,165889	1359	-0,08
16384	2949	14	0,179993	0,179767	2045	-0,13
32768	6350	15	0,193787	0,193645	6345	-0,07
65536	13623	16	0,207870	0,207522	13600	-0,16
131072	29024	17	0,221436	0,221400	29019	-0,02
262144	61493	18	0,234577	0,235278	61677	+0,29
524288	130717	19	0,249323	0,249155	130629	-0,07
1048576		20		0,263033	275810	
2097152		21		0,276911	580724	
4194304		22		0,290789	1219655	
8388608		23		0,304666	2555725	
16777216		24		0,318544	5344280	
33554432		25		0,332422	11154218	

Tabelle 8.1: Extrapolation für die Sortierzeiten auf einem Prozessor

In der Tabelle 8.2 sind die extrapolierten Werte und der sich daraus errechnende Speedup *kursiv* dargestellt. Für verschiedene Prozessorzahlen sind die Sortierzeiten angegeben, die der gleichen Datenmenge entsprechen. Die Spalte Speedup gibt dabei an, um welchen Faktor die parallele Ausführung schneller ist.

Betrachtet man die Abbildung 8.5, so zeigt sich, daß man bei 16 und 64 Prozessoren einen Speedup erzielt, der teilweise sogar superlinear ist. Dies spricht einerseits für eine optimale und gleichmäßige Ausnutzung aller Prozessoren, andererseits muß der superlineare Speedup noch eine weitere Ursache habe. Eine mögliche Erklärung wäre z.B., daß die Pivotwahl beim sequentiellen Quicksort schlechter ist als beim parallelen Quicksort mit lokaler Sortierung am Anfang. Bei 256 Prozessoren liegt der Speedup bei den größeren Datenmengen weit über 200, so daß auch hier von einer guten Auslastung der Prozessoren gesprochen werden kann. Die in der Abbildung gepunktet dargestellten Speedups basieren auf einem Vergleich mit den extrapolierten Sortierzeiten auf einem Prozessor. Aus diesem Grund kann man für 1024 Prozessoren keine gesicherten Aussagen machen. Ein geschätzter Speedup von über 800 bei großen Datenmengen stellt aber auch ein gutes Ergebnis dar. Insgesamt ist der Speedup aber um so besser, je größer die Datenmenge ist.

Würde man versuchen, das sequentielle Quicksort zu optimieren, so hätte dies nur teilweise eine Verschlechterung des Speedups zur Folge. Denn das sequentielle Quicksort wird auch beim parallelen Quicksort für die lokale Sortierung eingesetzt und macht bei großen Datenmengen teilweise über die Hälfte der Gesamtzeit aus.

Alle vier Speedup-Kurven haben eine kleine Delle bei 4096 Elementen pro Prozessor. Die Ursache hierfür ist wiederum die maximale Nachrichtenlänge. Bis 2048 Elementen

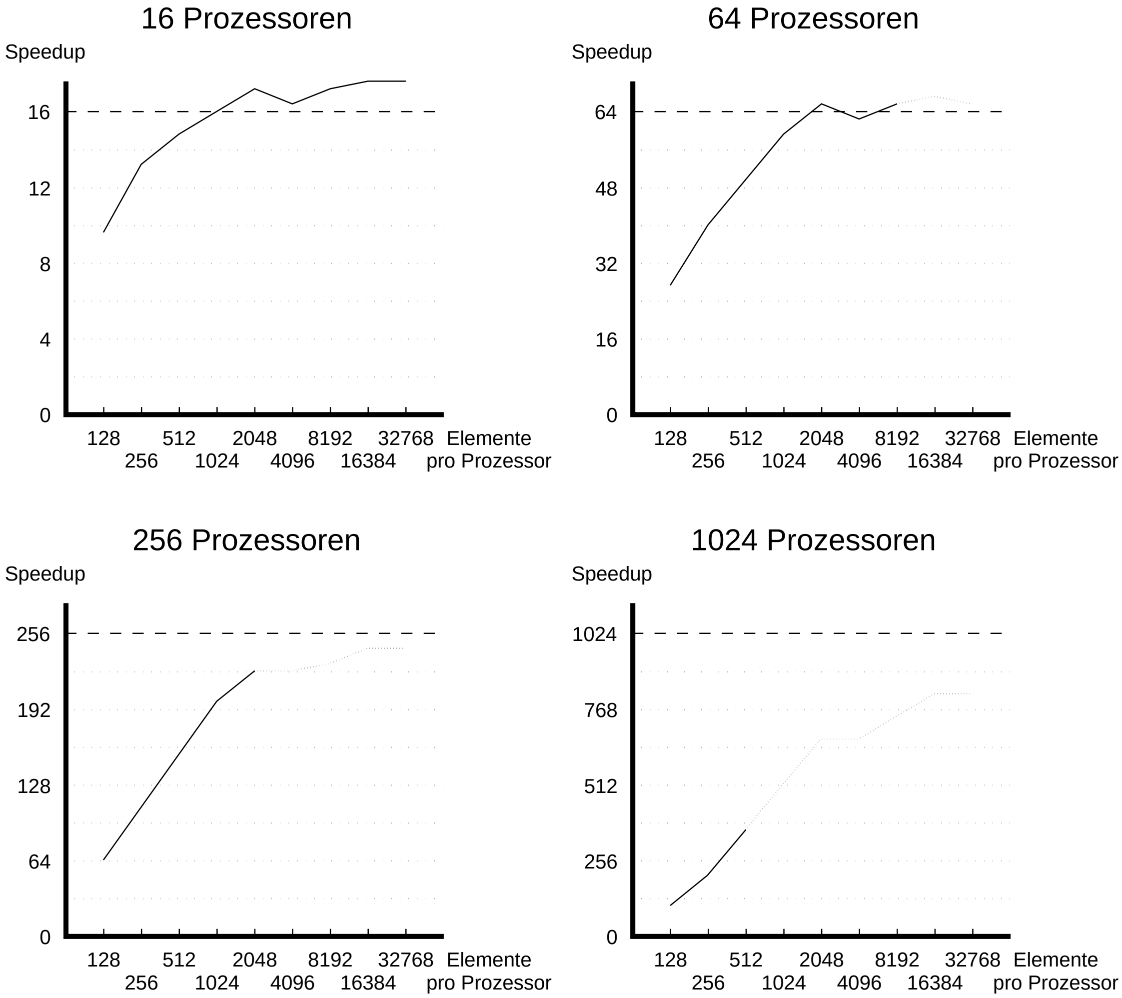


Abbildung 8.5: Speedup beim parallelen Quicksort für verschiedene Prozessorzahlen (gepunktete Linien basieren auf einem Vergleich mit extrapolierten Werten)

Datenmenge	1 Prozessor Zeit [ms]	16 Prozessoren Zeit [ms]	16 Prozessoren Speedup	64 Prozessoren Zeit [ms]	64 Prozessoren Speedup	256 Prozessoren Zeit [ms]	256 Prozessoren Speedup	1024 Prozessoren Zeit [ms]	1024 Prozessoren Speedup
2048	282	29	9,7						
4096	623	47	13,3						
8192	1360	91	14,9	50	27,2				
16384	2949	186	15,9	74	39,9				
32768	6350	370	17,1	127	50,0	99	64,1		
65536	13623	831	16,4	229	59,5	128	106,4		
131072	29024	1698	17,1	442	65,7	190	152,8	267	108,7
262144	61493	3548	17,3	981	62,7	310	198,4	309	199,0
524288	130717	7565	17,3	1958	66,8	579	225,8	388	366,9
1048576	*275810*			4096	*67,3*	1243	*221,9*	539	*511,7*
2097152	*580724*			8789	*66,1*	2527	*229,8*	875	*663,7*
4194304	*1219655*					5057	*241,2*	1826	*667,9*
8388608	*2555725*					10537	*242,5*	3454	*739,9*
16777216	*5344280*							6598	*810,0*
33554432	*11154218*							13534	*824,2*

Tabelle 8.2: Speedup gegenüber der Sortierung auf einem Prozessor. Extrapolierte Werte und sich daraus berechnende Ergebnisse sind *kursiv* dargestellt.

pro Prozessor passen die zu versendenden Daten immer noch in eine einzige Nachricht, während ab 4096 Elementen pro Prozessor häufig zwei Nachrichten an einen Prozessor geschickt werden müssen.

8.5 Vergleich zwischen parallelem Quicksort und anderen parallelen Sortieralgorithmen

In [DGLM94] wurden die fünf verschiedenen parallelen Sortieralgorithmen Gridsort, Shearsort, Bitonicsort, Radixsort und Samplesort untersucht. Dabei hat sich gezeigt, daß Bitonicsort bis zu 2048 Elementen pro Prozessor das beste Verfahren ist, während sich für größere Datenmengen Samplesort als unschlagbar erweist.

Um die Leistungsfähigkeit des parallelen Quicksorts im Vergleich zu den oben erwähnten Sortierverfahren zu demonstrieren, wurden annähernd gleiche Testbedingungen geschaffen:

- Es wurde exakt der selbe Rechner verwendet (GCel mit 1024 Prozessoren an der Universität Paderborn)
- Die Daten wurden gemäß dem NAS Parallel Benchmark erzeugt

Der einzige Unterschied bei den Testbedingungen besteht darin, daß bei den Messungen in [DGLM94] als Betriebssystem Parix verwendet wurde, während Quicksort unter dem Betriebssystem COSY zum Einsatz kam.

Die Meßergebnisse für paralleles Quicksort zeigen aber, daß es in weiten Bereichen erheblich besser ist als alle anderen parallelen Sortierverfahren. In der Abbildung 8.6 sind die Sortierzeiten aus [DGLM94] wiedergegeben. Die Sortierzeiten für paralleles Quicksort sind gestrichelt eingetragen. Die Skala auf der linken Seite ist logarithmisch dargestellt, um prozentuale Unterschiede leichter ablesen zu können.

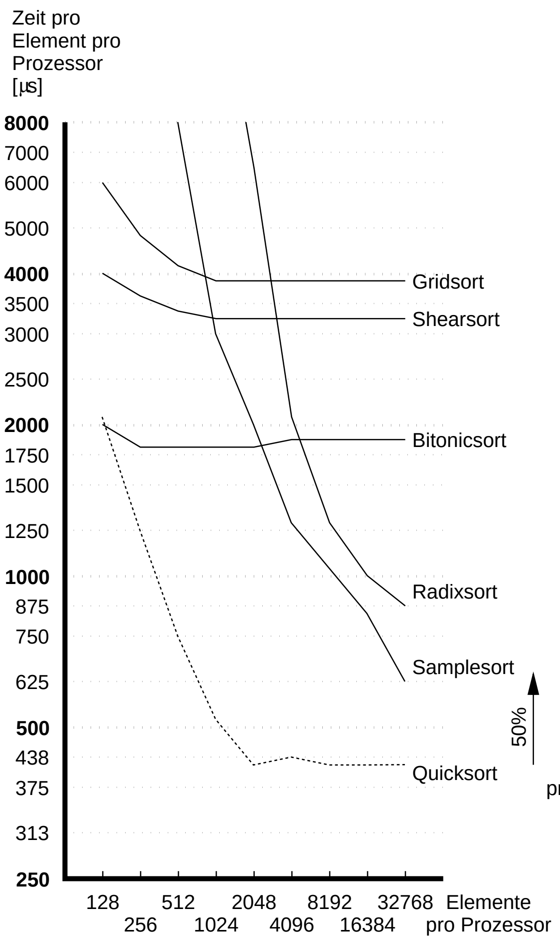


Abbildung 8.6: Sortierzeiten von parallelen Sortieralgorithmen

Bereits ab 256 Elementen pro Prozessor ist paralleles Quicksort besser als Bitonicsort und wird selbst bis 32768 Elementen pro Prozessor nicht von Samplesort eingeholt. Samplesort spielt seine Vorteile erst bei sehr großen Datenmengen aus, da der gesamte Datenbestand insgesamt nur einmal unter den Prozessoren ausgetauscht werden muß. Wie man aus der Abbildung 8.6 erkennen kann, ist das parallele Quicksort aber gerade im mittleren Datenbereich erheblich besser als Bitonic- und Samplesort. Beispielsweise bei 2048 Elementen pro Prozessor liegt die Sortierzeit bei Bitonicsort mehr als viermal so hoch wie beim parallelen Quicksort.

Selbst wenn die Kommunikation unter COSY etwas schneller ist als unter Parix, so bleibt Quicksort trotzdem ungeschlagen. Belastet man Quicksort mit einem Handicap, bei dem alle Daten doppelt übertragen werden, so verdoppelt sich der Kommunikationsaufwand zwar annähernd, aber die Sortierzeiten erhöhen sich um maximal 30%. Quicksort bleibt dabei in den meisten Datenbereichen immer noch erheblich besser als Bitonic- oder Samplesort.

8.6 Ausblick

Paralleles Quicksort eignet sich für den universellen Einsatz bei mittleren und großen Datenmengen. Für sehr große Datenmengen wird Samplesort aber immer besser sein, da alle Daten nur einmal ausgetauscht werden, was beim Quicksort in jeder Iteration der Fall ist. Dennoch bietet sich eine Integration von Quicksort in Samplesort an. Beim Samplesort wird zuerst ein sogenanntes Sample (in [DGLM94] 1% bis 2% der Gesamtdatenmenge) sortiert. Gemäß dem Ergebnis dieser Sortierung wird jedem Prozessor ein Zahlenbereich zugeordnet. Dann beginnt die eigentliche Datenaustauschphase. Jeder Prozessor verschickt seine Daten an die betreffenden Prozessoren. Mit einer lokalen Sortierung der Daten auf jedem Prozessor ist die Sortierung abgeschlossen. Die Sortierung des Samples ist ebenfalls eine parallele Sortierung, nur über einen Bruchteil der Gesamtdatenmenge. Da gerade Quicksort für mittlere Datenmengen gut geeignet ist, kann man Quicksort zur Sortierung des Samples bei Samplesort einsetzen.

Bei den Untersuchungen in dieser Arbeit wurden Zufallszahlen gemäß dem NAS Parallel Benchmark sortiert, die lokal auf jedem Prozessor erzeugt wurden. In der Praxis werden diese Randbedingungen nur in seltenen Fällen anzutreffen sein. Untersuchungen für reale Daten waren nicht Gegenstand dieser Arbeit. Dennoch lassen sich dazu folgende Bemerkungen machen: Für den Fall, daß die Daten zentral anfallen und auf die Prozessoren erst verteilt werden müssen, kann man durch eine zufällige Verteilung der Daten die optimalen Startbedingungen für das parallele Quicksort erzeugen. Möchte man hingegen Daten sortieren, die zwar schon über die Prozessoren verteilt sind, aber keiner zufälligen Verteilung entsprechen, ist es erforderlich, die Daten erst einigermaßen zufällig unter den Prozessoren auszutauschen. Der zusätzliche Aufwand dafür entspricht dem einer Datenaustauschphase beim parallelen Quicksort. Die maximale Zeit für eine Datenaustauschphase, die beobachtet wurde, liegt bei ca. 20% der Gesamtzeit. Durch das Vorschalten dieser Datenaustauschphase zur zufälligen Verteilung der Daten verschlechtert sich zwar die Gesamtlaufzeit des Algorithmus etwas, dafür entspricht aber der Aufwand im worst-case dem Aufwand im durchschnittlichen Fall. Für detailliertere Ergebnisse wären weitere und umfassende Untersuchungen erforderlich.

Literaturverzeichnis

[BBBB94] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, S. Weeratunga. *The NAS Parallel Benchmarks.* RNR Technical Report RNR-94-007, 1994

[BBDS94] D. Bailey, E. Barszcz, L. Dagum, H. Simon. *NAS Parallel Benchmark Results 10-94.* NAS Technical Report NAS-94-001, 1994

[BMNR93] A. Bachem, T. Meis, K. Nagel, M. Riemeyer, M. Wottawa. *Programming, Porting and Performance Tests on a 1024-processor Transputercluster.* Proceedings, Transputer Applications and Systems '93, Vol. 2, pp. 1068-1075, 1993

[BuGi94] R. Butenuth, S. Gilles. *Cosy - ein Betriebssystem für hochparallele Computer.* Tagungsband TAT'94, Aachen, 1994

[DGLM94] R. Diekmann, J. Gehring, R. Lüling, B. Monien, M. Nübel, R. Wanka. *Sortieren großer Datenmengen auf einem massiv parallelen System.* Universität-GH Paderborn, Fachbereich Mathematik/Informatik, 1994

[Hans95] T. Hansch. *Segmentierte kollektive Operationen auf Gittern mit Hilbertnumerierung.* Universität Karlsruhe, Lehrstuhl für Ingenieure und Naturwissenschaftler, 1995

[Hilb90] D. Hilbert. *Ueber die stetige Abbildung einer Linie auf ein Flächenstück.* Gesellschaft deutscher Naturforscher und Aerzte, Bremen, 1890

[JáJá92] J. JáJá. *An Introduction to parallel algorithms.* Addison-Wesley Publishing Company, 1992

[KGGK94] V. Kumar, A. Grama, A. Gupta, G. Karypis. *Introduction to parallel computing.* The Benjamin/Cummings Publishing Company, Inc., 1994

[Leig92] F. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes.* Morgan Kaufmann Publishers, 1992

[LiSe94] H. Li, K. Sevcik. *Parallel Sorting by Overpartitioning.* SPAA 94, Cape May, New Jersey, 1994

[OtWi90] T. Ottmann, P. Widmayer. *Algorithmen und Datenstrukturen.* BI-Wissenschaftsverlag, 1990

[PeJS92] Peitgen, Jürgens, Saupe. *Fractals for the Classroom.* Volume I und II, Spinger Verlag, 1992

[ReSo84] F. Reinhardt, H. Soeder. *dtv-Atlas zur Mathematik, Tafeln und Texte, Band 2.* 5. Auflage, Deutscher Taschenbuch Verlag GmbH & Co. KG, München, 1984

[SaBa95] S. Sainim, D. Bailey. *NAS Parallel Benchmark Results 3-95.* Report NAS-95-011, 1995

[Sand96] P. Sanders. *A Scalable Parallel Tree Search Library.* 2nd Workshop on Solving Irregular Problems on Distributed Memory Maschines, Honolulu, Hawaii, 1996

[UmVo94] T. Umland, R. Vollmar. *Transputerpraktikum.* Teubner, 1992

[Unge94] T. Ungerer. *Parallelrechner und Parallelprogrammierung (Skriptum zur gleichnamigen Vorlesung WS 1994/95).* 2.Auflage, Universität Karlsruhe, Fakultät für Informatik, 1994

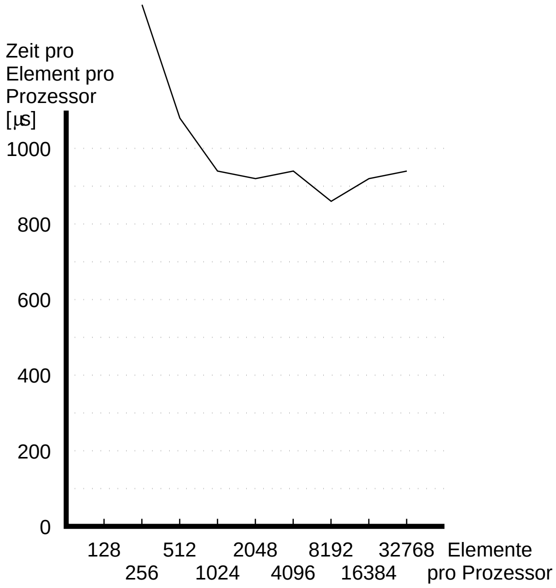
Anhang A

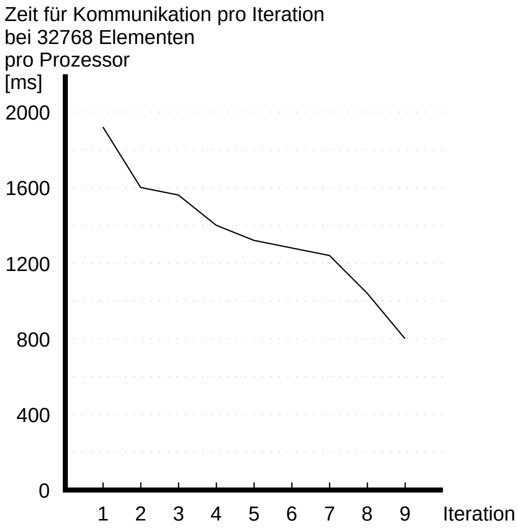
Datenblätter

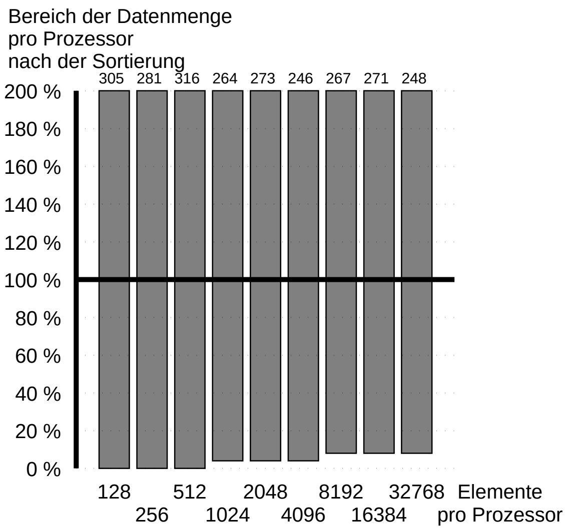
Zu den meisten Konfigurationen des parallelen Quicksorts sind hier Datenblätter zu finden, in denen die wichtigsten Meßergebnisse tabellarisch und graphisch dargestellt sind. Ein Erläuterung der Meßbedingungen und der dargestellten Meßwerte ist in Kapitel 4.3 zu finden.

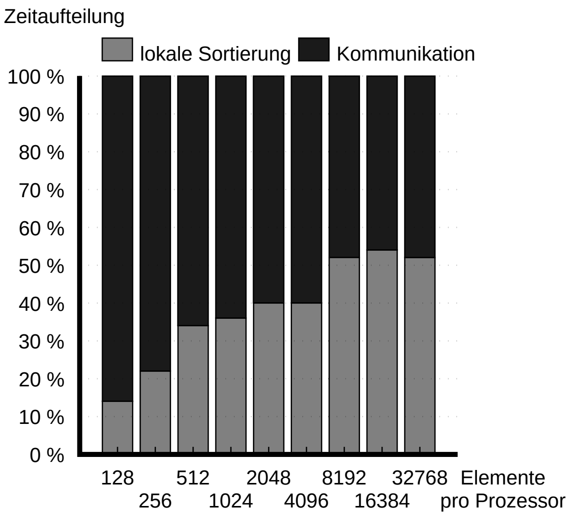
Datenblatt 1

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	3-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





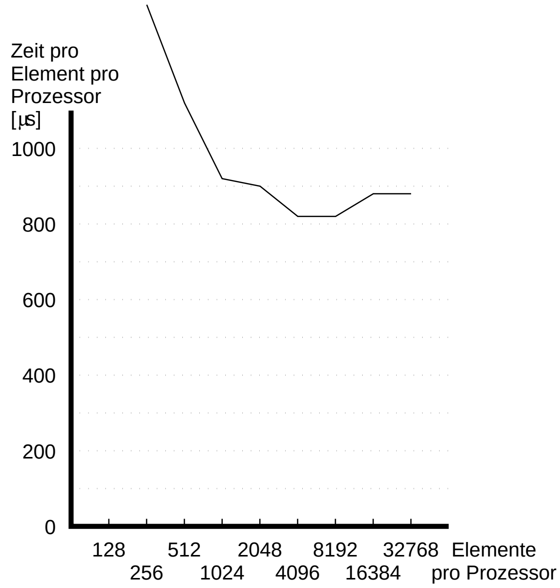


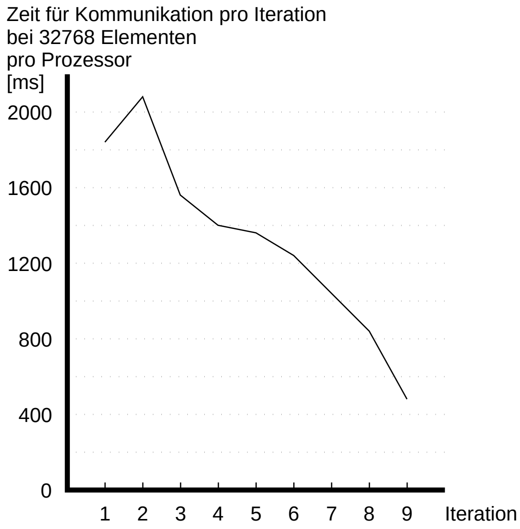


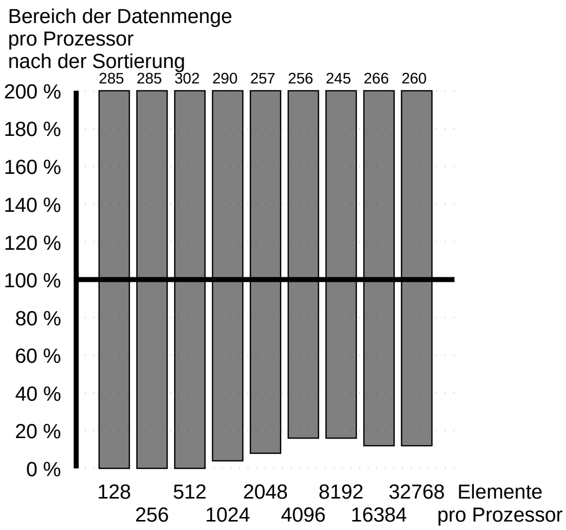
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1976	1378	1081	944	924	942	859	917	931
Gesamtzeit [s]	0.25	0.35	0.55	0.97	1.89	3.86	7.05	15.04	30.54
Zeit für Kommunikation [ms]	232	294	396	671	1344	2506	3856	7553	16362
Zeit für lokale Sortierung [ms]	39	83	212	382	880	1689	4049	8813	17231
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	2	3	29	49	136	516	806	2682
Maximale Anzahl Elemente	391	720	1620	2704	5588	10090	21900	44475	81214
Anzahl Datenpakete	6849	7167	6756	6827	9049	13920	23457	41581	84270
Gesamtweglänge aller Datenpaket	34608	36224	34628	35120	46604	67850	114075	196278	409104
Durchschn. Iterationstiefe	9.62	10.10	9.48	9.51	9.82	9.74	9.48	9.07	9.60

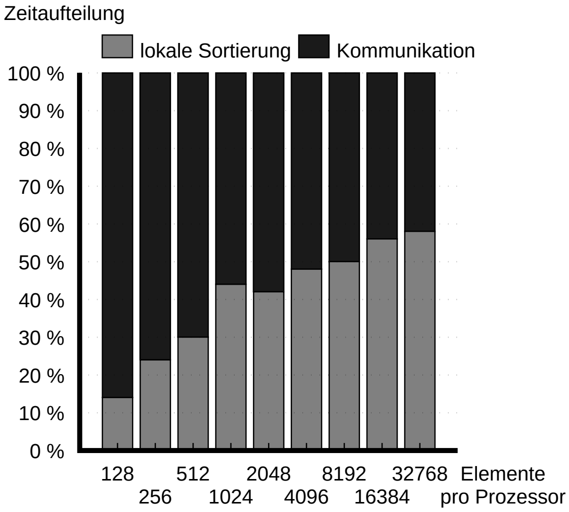
Datenblatt 2

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	3-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





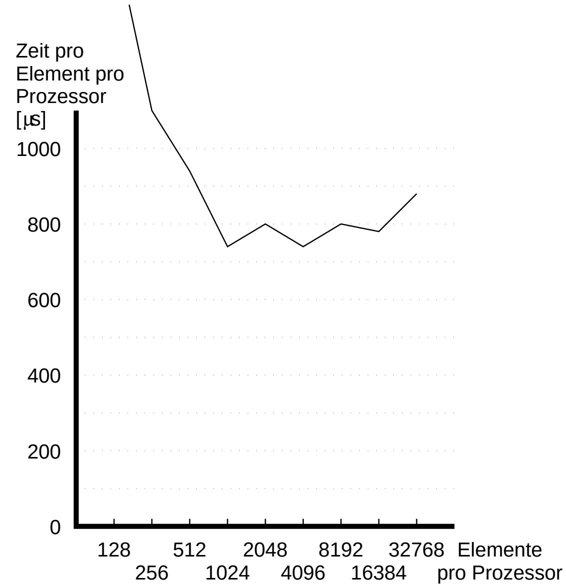


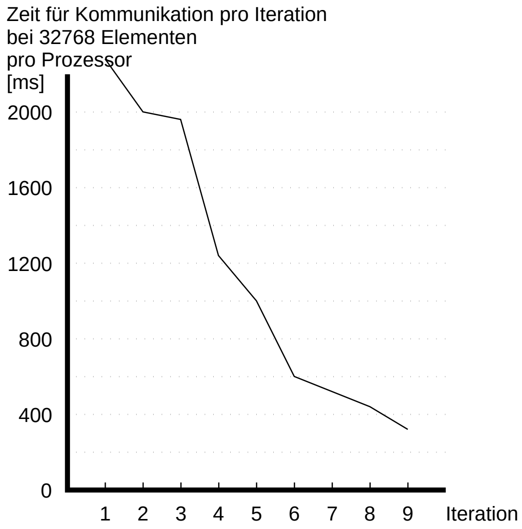


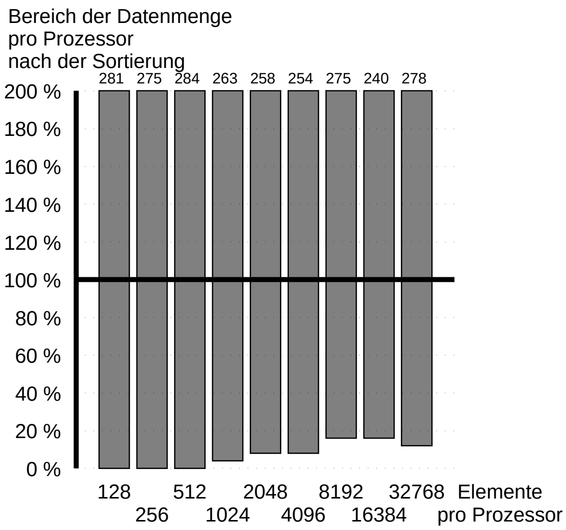
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	2077	1374	1159	913	903	815	830	878	887
Gesamtzeit [s]	0.27	0.35	0.59	0.94	1.85	3.34	6.80	14.39	29.10
Zeit für Kommunikation [ms]	240	283	459	575	1313	1979	3821	6976	13645
Zeit für lokale Sortierung [ms]	38	83	200	426	825	1780	3691	8563	18149
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	1	0	4	21	74	342	648	853	2049
Maximale Anzahl Elemente	366	731	1545	2970	5280	10510	20139	43670	85506
Anzahl Datenpakete	6804	7286	7042	6836	8896	13302	23046	40781	80270
Gesamtweglänge aller Datenpaket	30809	32701	32638	31290	41059	57847	98459	171073	337330
Durchschn. Iterationstiefe	9.59	10.25	9.85	9.51	9.70	9.40	9.36	8.92	9.17

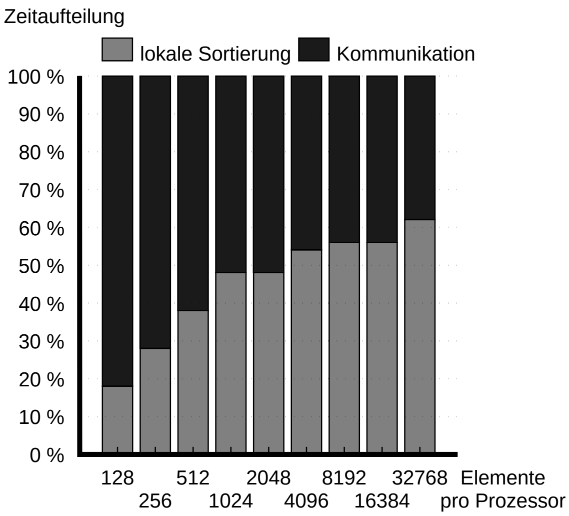
Datenblatt 3

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	3-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





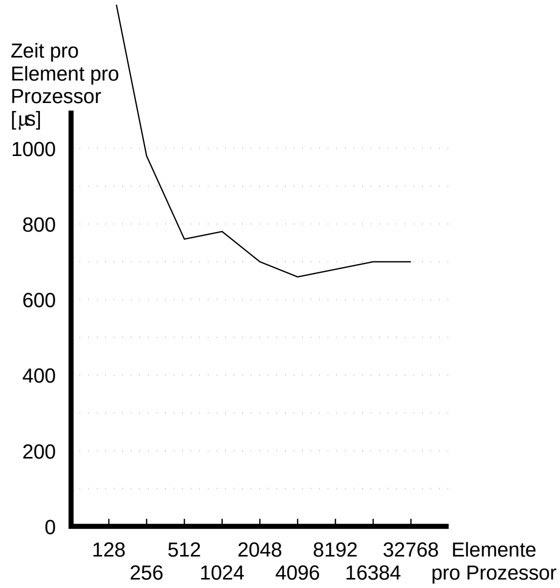


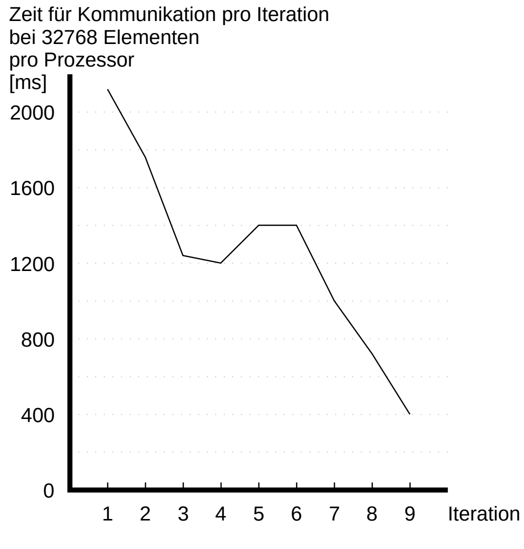


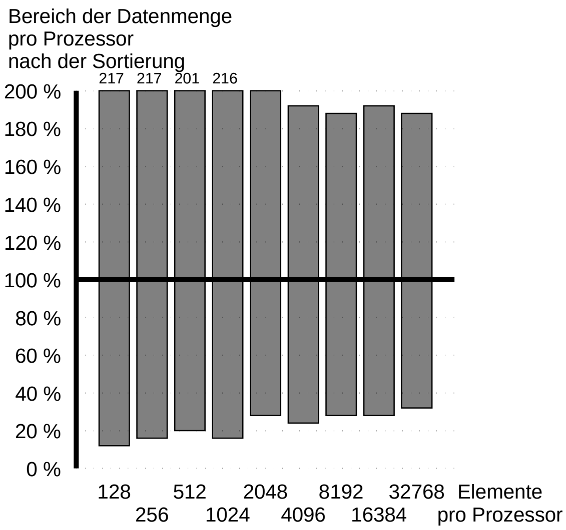
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1577	1096	935	729	791	741	804	790	888
Gesamtzeit [s]	0.20	0.28	0.48	0.75	1.62	3.04	6.59	12.96	29.13
Zeit für Kommunikation [ms]	176	211	320	436	922	1549	3248	6186	12417
Zeit für lokale Sortierung [ms]	38	82	192	383	830	1749	4191	7681	19263
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	1	6	20	76	147	617	1238	1932
Maximale Anzahl Elemente	360	705	1456	2698	5277	10405	22497	39281	90938
Anzahl Datenpakete	6891	7122	6722	6960	8844	13339	23220	42702	82651
Gesamtweglänge aller Datenpaket	27368	28037	27440	27985	34109	49197	84703	154494	296643
Durchschn. Iterationstiefe	9.68	10.03	9.43	9.68	9.64	9.42	9.41	9.34	9.41

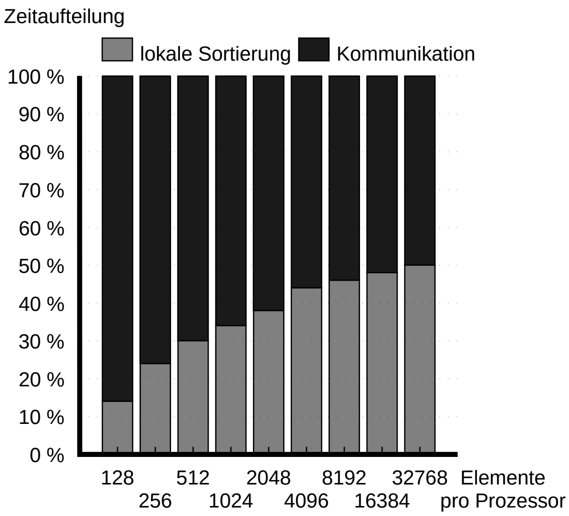
Datenblatt 4

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	log(N)-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





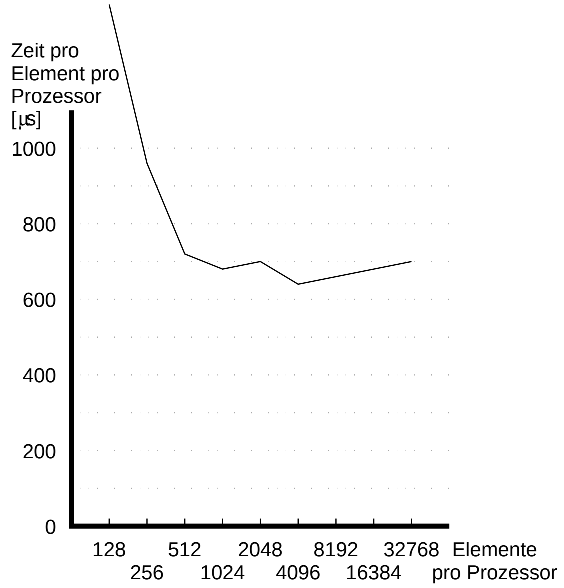


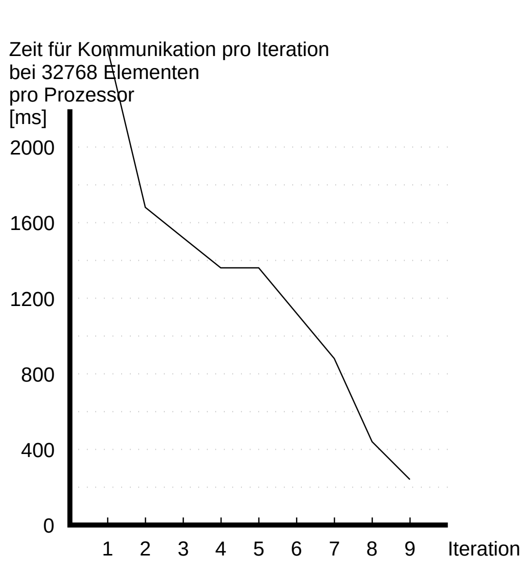


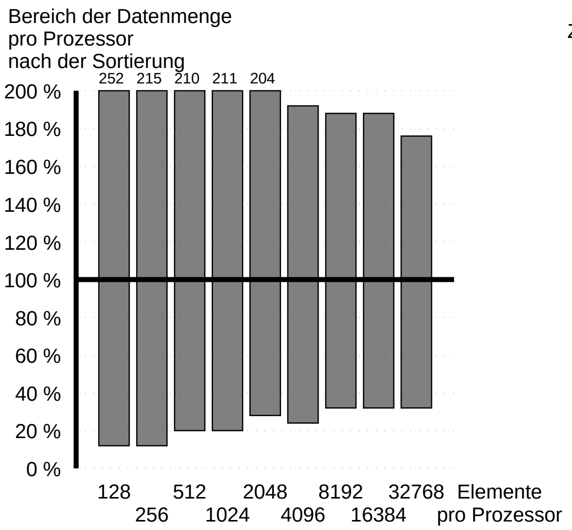
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1484	983	757	778	694	657	675	700	703
Gesamtzeit [s]	0.19	0.51	0.39	0.80	1.42	2.69	5.54	11.49	23.07
Zeit für Kommunikation [ms]	168	208	295	576	970	1676	3304	6503	12443
Zeit für lokale Sortierung [ms]	28	62	128	306	611	1307	2739	5998	12522
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	16	41	107	181	612	1082	2326	4764	11044
Maximale Anzahl Elemente	278	558	1030	2217	4065	7923	15382	31478	61567
Anzahl Datenpakete	6153	6224	5949	6109	7464	11815	21070	38690	73278
Gesamtweglänge aller Datenpaket	32113	31529	31367	31987	37352	60089	104205	192348	359369
Durchschn. Iterationstiefe	8.71	8.79	8.42	8.62	8.37	8.38	8.64	8.53	8.44

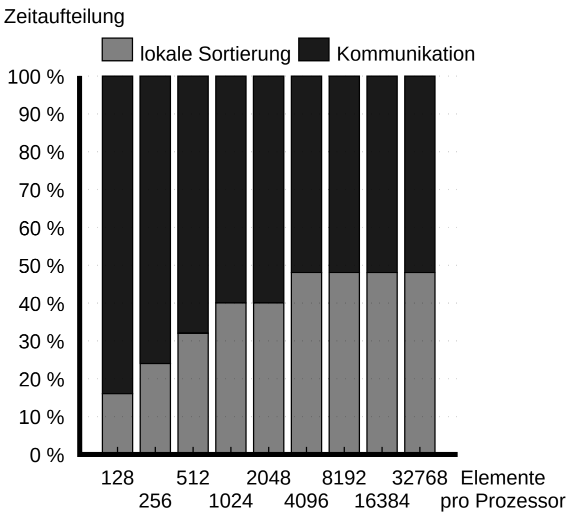
Datenblatt 5

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	log(N)-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





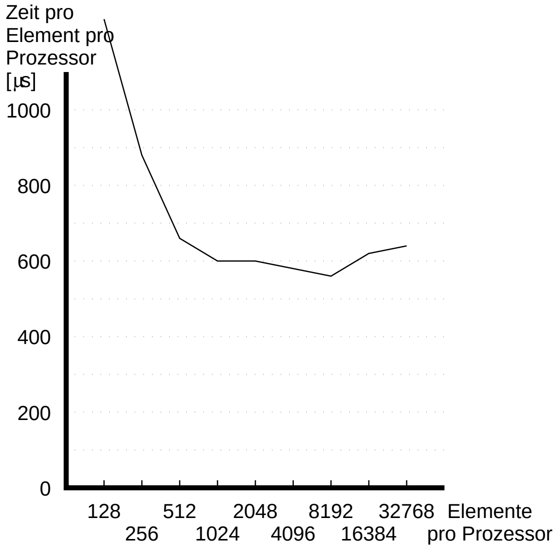


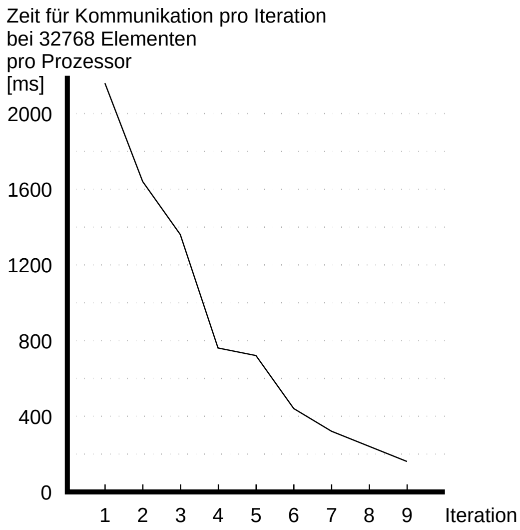


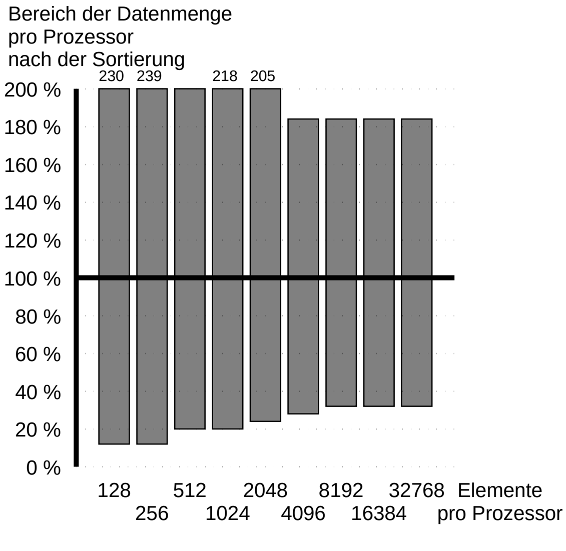
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1406	956	727	674	709	649	665	685	694
Gesamtzeit [s]	0.18	0.25	0.37	0.69	1.45	2.66	5.45	11.23	22.79
Zeit für Kommunikation [ms]	154	203	285	454	929	1603	3102	6233	12388
Zeit für lokale Sortierung [ms]	33	61	133	290	633	1293	2728	5866	11717
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	14	30	94	206	574	1053	2478	4811	10092
Maximale Anzahl Elemente	323	552	1074	2158	4177	7922	15469	30830	57797
Anzahl Datenpakete	6074	6173	6004	6059	7486	11792	20701	37930	72472
Gesamtweglänge aller Datenpaket	28490	28269	28145	28067	34264	52630	92083	168224	322600
Durchschn. Iterationstiefe	8.61	8.71	8.50	8.56	8.38	8.36	8.44	8.38	8.35

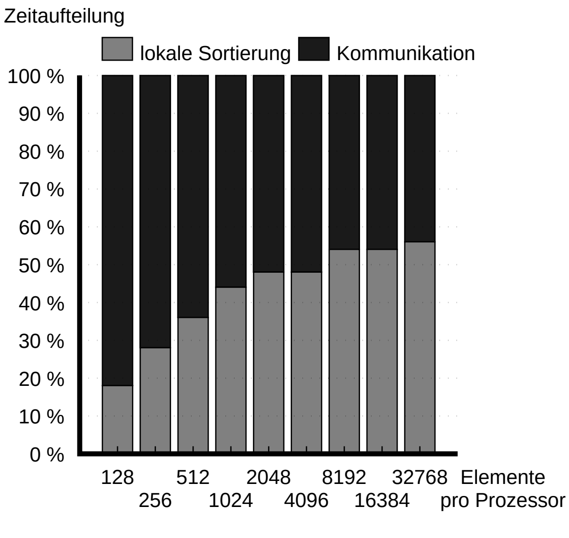
Datenblatt 6

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	log(N)-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





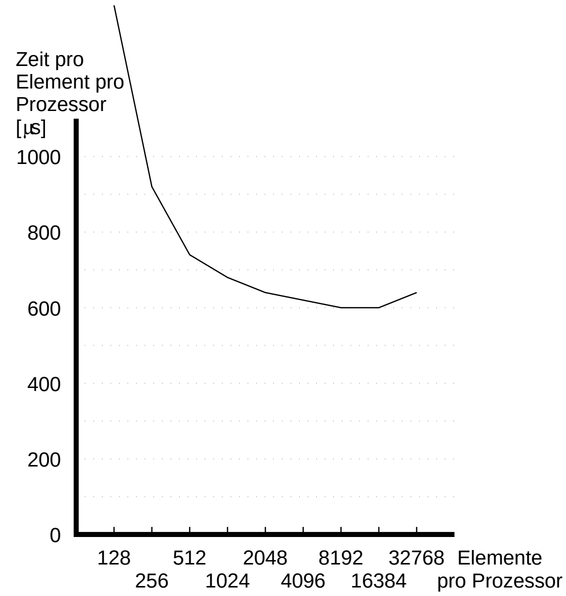


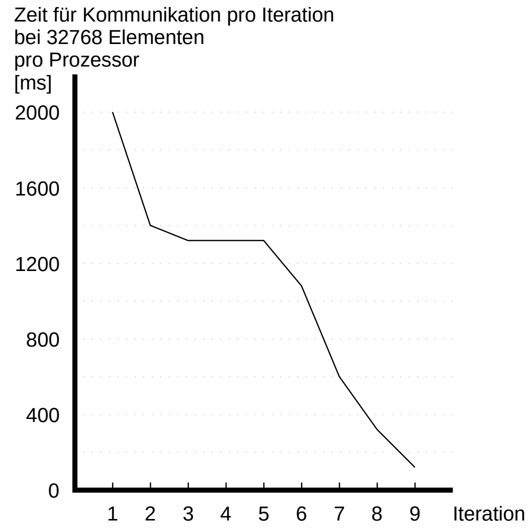


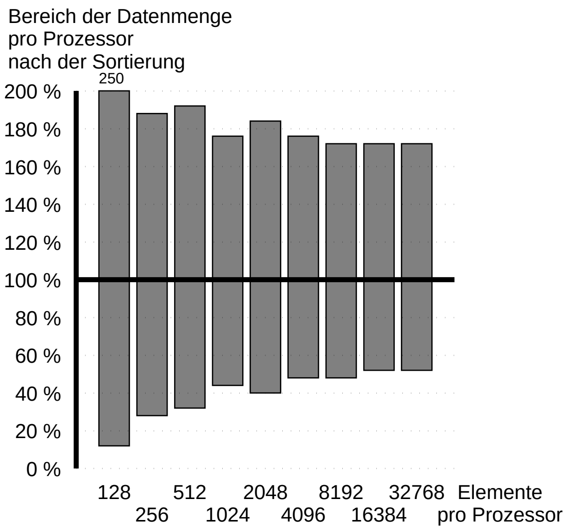
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1249	882	655	597	600	583	565	616	635
Gesamtzeit [s]	0.16	0.23	0.36	0.61	1.23	2.39	4.63	10.10	20.84
Zeit für Kommunikation [ms]	141	185	239	370	706	1330	2269	5019	9191
Zeit für lokale Sortierung [ms]	31	69	126	311	628	1226	2632	5749	12538
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	18	37	106	187	527	1128	2515	5564	10080
Maximale Anzahl Elemente	295	613	1026	2229	4190	7541	15047	30291	60987
Anzahl Datenpakete	6159	6209	5982	6163	7473	11791	20490	38096	72909
Gesamtweglänge aller Datenpaket	25275	25208	25186	25197	29643	46810	78296	147809	277799
Durchschn. Iterationstiefe	8.72	8.76	8.47	8.69	8.44	8.38	8.38	8.42	8.40

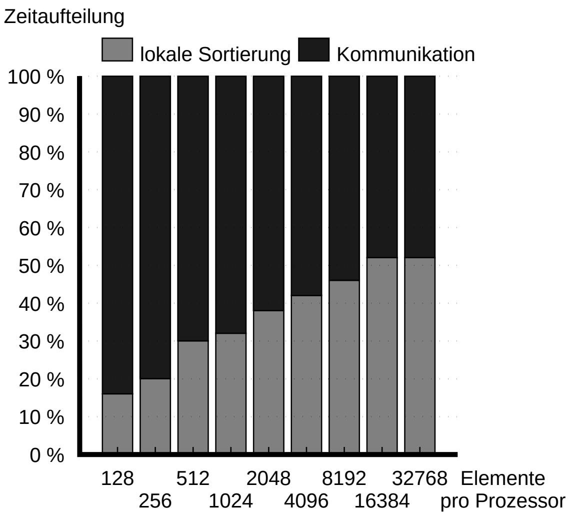
Datenblatt 7

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	$\sqrt{N}$-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





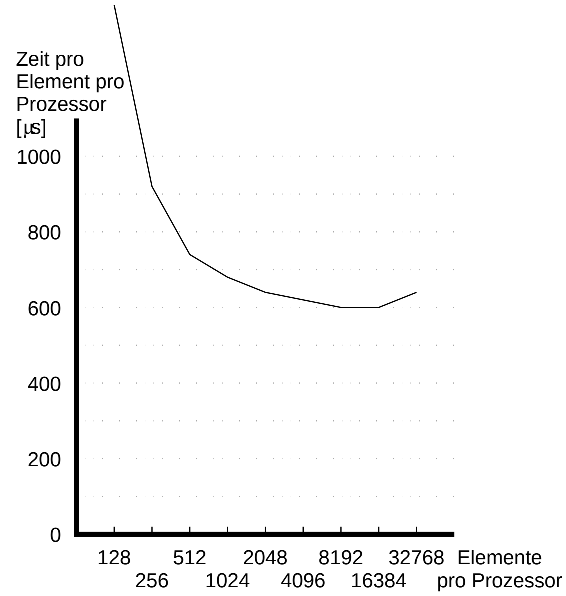


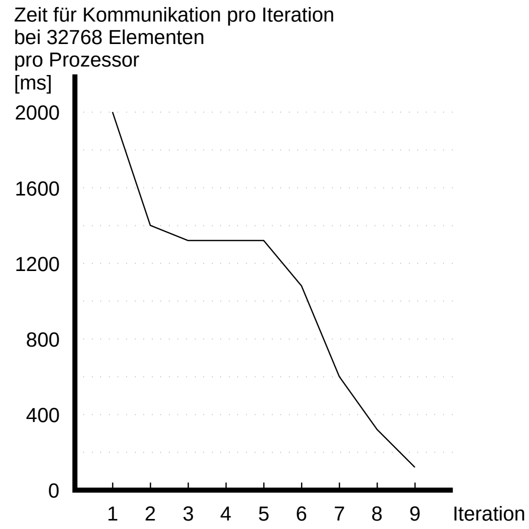


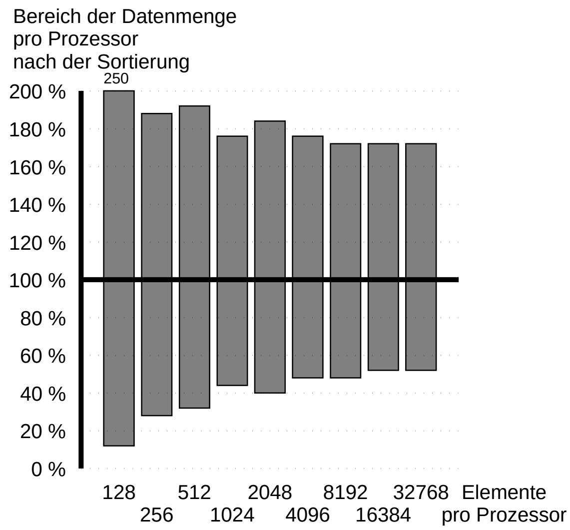
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1484	917	731	685	644	619	610	592	642
Gesamtzeit [s]	0.19	0.23	0.37	0.70	1.32	2.54	5.00	9.71	21.05
Zeit für Kommunikation [ms]	163	199	288	513	887	1598	2908	4904	10524
Zeit für lokale Sortierung [ms]	32	53	120	245	551	1173	2472	5321	11462
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	17	76	154	463	860	1998	4069	8732	16885
Maximale Anzahl Elemente	320	486	981	1821	3747	7184	14190	28262	56076
Anzahl Datenpakete	6198	5852	5840	5797	7270	11438	19813	36510	70074
Gesamtweglänge aller Datenpaket	31947	30858	30691	30128	37588	57776	102811	185348	363402
Durchschn. Iterationstiefe	8.77	8.30	8.29	8.22	8.28	8.14	8.12	8.07	8.08

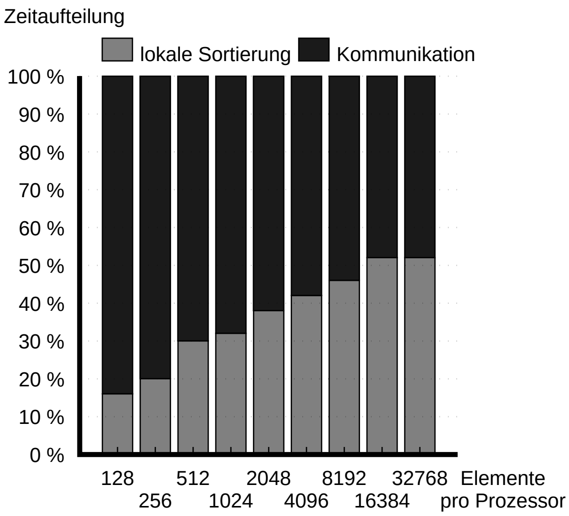
Datenblatt 8

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	$\sqrt{N}$-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





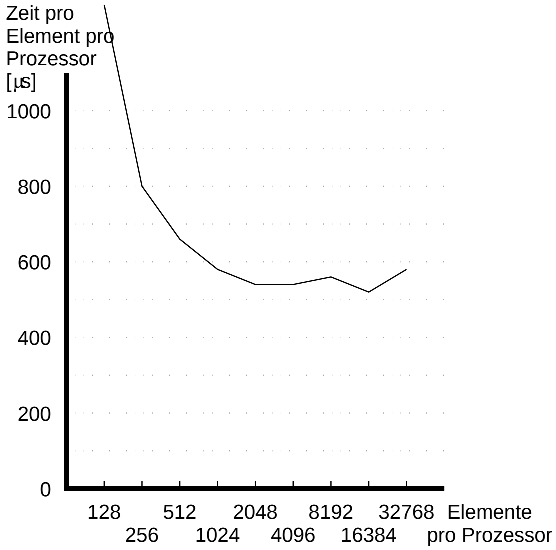


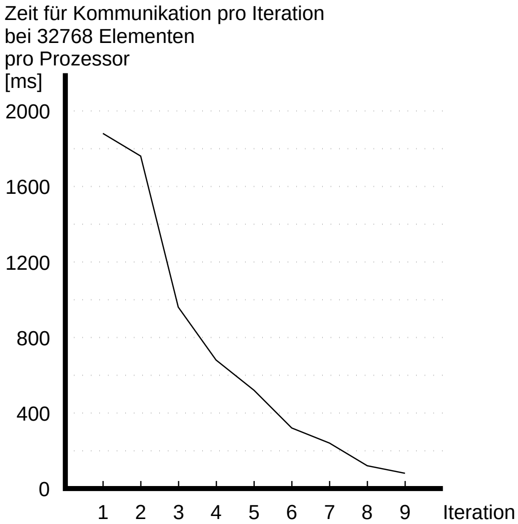


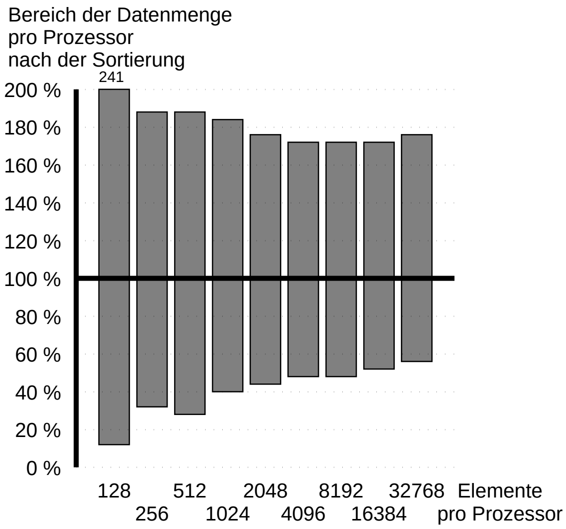
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1409	944	702	638	624	596	585	590	605
Gesamtzeit [s]	0.18	0.24	0.36	0.65	1.28	2.44	4.80	9.69	19.85
Zeit für Kommunikation [ms]	153	202	277	438	853	1418	2646	4958	9577
Zeit für lokale Sortierung [ms]	32	51	112	254	527	1174	2530	5385	11290
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	14	69	170	424	930	2081	4048	8390	17568
Maximale Anzahl Elemente	319	477	940	1853	3533	7146	14223	28478	55696
Anzahl Datenpakete	6194	5956	5943	5810	7274	11330	19860	36550	70034
Gesamtweglänge aller Datenpaket	28180	27784	27930	27482	33556	51712	89644	163064	312645
Durchschn. Iterationstiefe	8.76	8.44	8.41	8.24	8.23	8.11	8.13	8.06	8.07

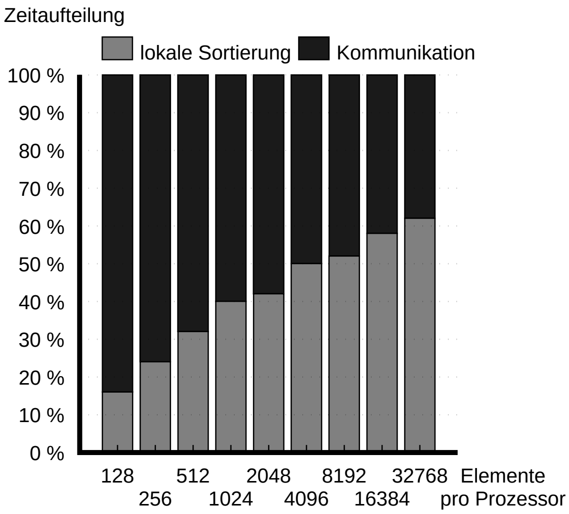
Datenblatt 9

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	$\sqrt{N}$-Median des Wurzelprozessors		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





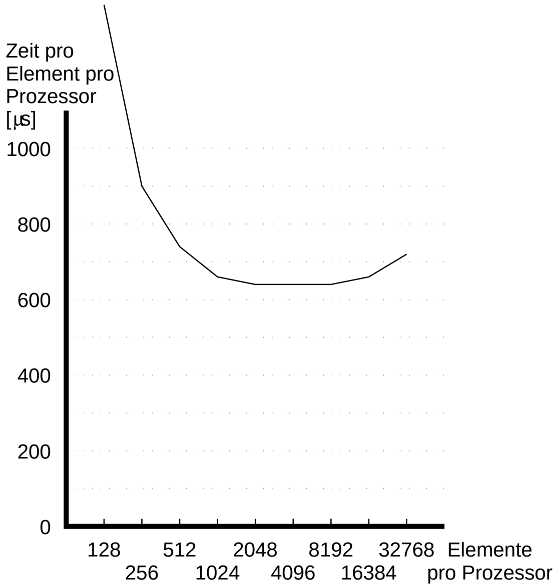


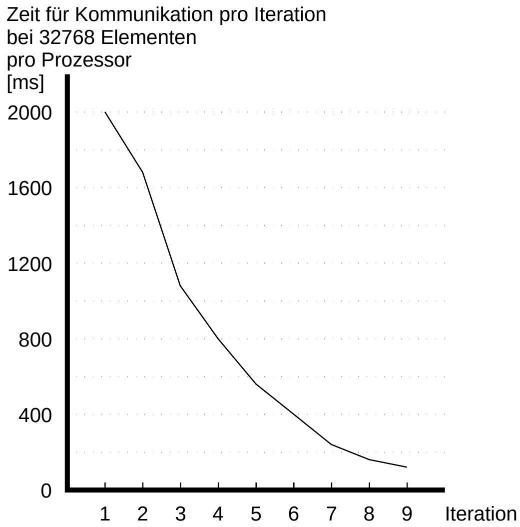


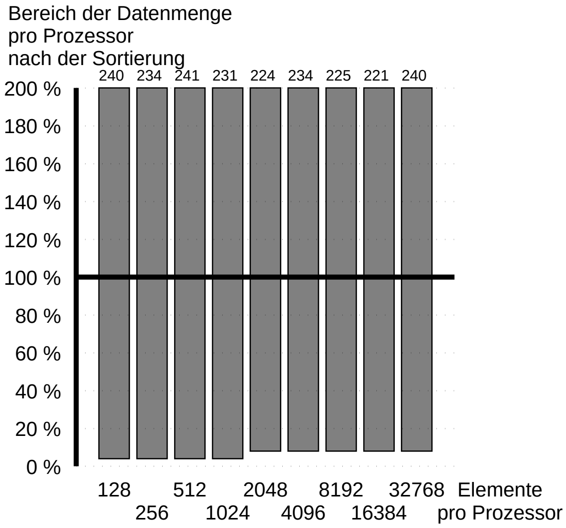
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1273	800	669	579	545	544	555	527	571
Gesamtzeit [s]	0.16	0.20	0.34	0.60	1.12	2.23	4.55	8.65	18.75
Zeit für Kommunikation [ms]	149	166	251	381	647	1154	2238	3754	7548
Zeit für lokale Sortierung [ms]	30	54	117	258	528	1137	2459	5289	11880
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	19	80	142	410	877	2028	4046	8707	17788
Maximale Anzahl Elemente	308	482	955	1895	3600	6975	14082	28177	58829
Anzahl Datenpakete	6183	5926	5937	5797	7284	11262	19866	36510	70104
Gesamtweglänge aller Datenpaket	25382	25133	25043	24682	29441	46005	78937	148757	283101
Durchschn. Iterationstiefe	8.75	8.40	8.40	8.28	8.22	8.13	8.13	8.07	8.07

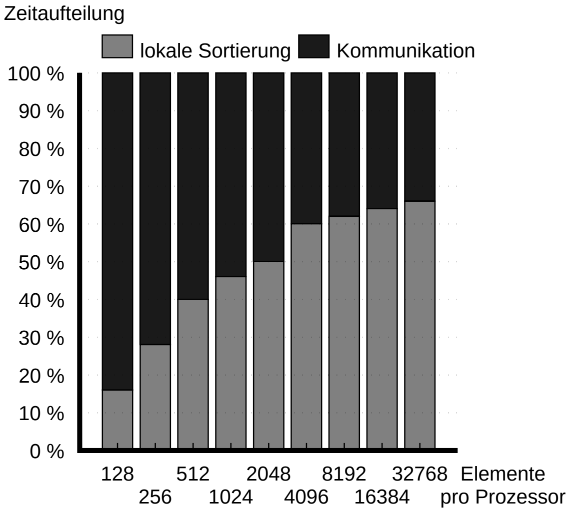
Datenblatt 10

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der 3-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





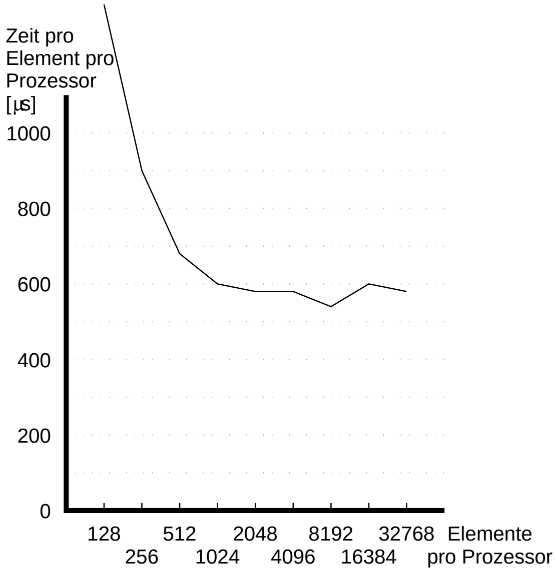


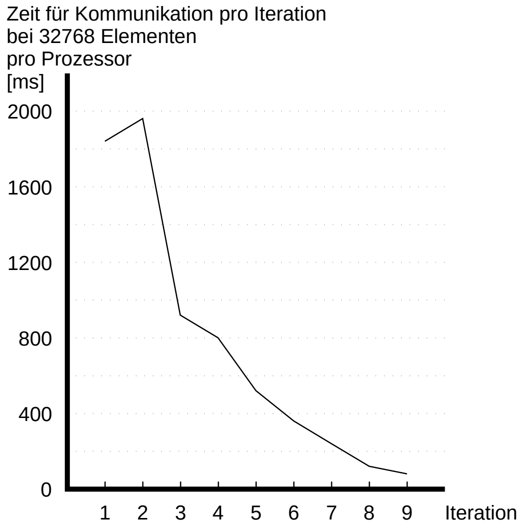


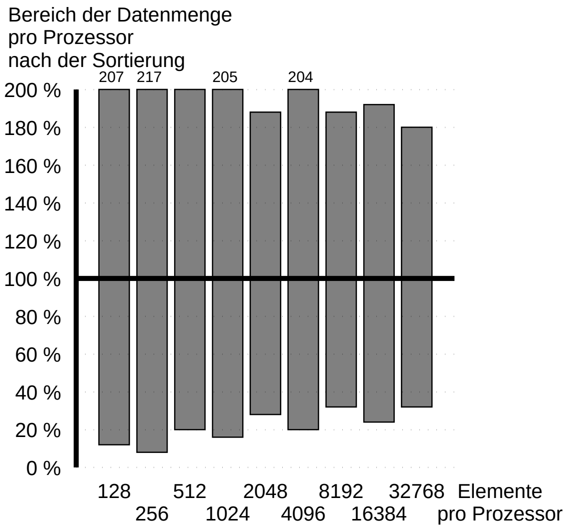
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1359	905	739	653	644	633	629	661	719
Gesamtzeit [s]	0.17	0.23	0.38	0.67	1.32	2.60	5.16	10.84	23.59
Zeit für Kommunikation [ms]	151	179	241	378	679	1098	2078	4148	8281
Zeit für lokale Sortierung [ms]	30	67	155	332	702	1641	3341	7124	16436
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	6	6	25	53	159	230	548	1294	2071
Maximale Anzahl Elemente	307	598	1233	2367	4584	9595	18466	36287	78649
Anzahl Datenpakete	5759	5768	5768	5792	7407	11575	20020	37037	71320
Gesamtweglänge aller Datenpaket	24390	24362	24466	24469	30786	46741	80102	147258	283578
Durchschn. Iterationstiefe	8.20	8.19	8.18	8.20	8.20	8.20	8.20	8.20	8.20

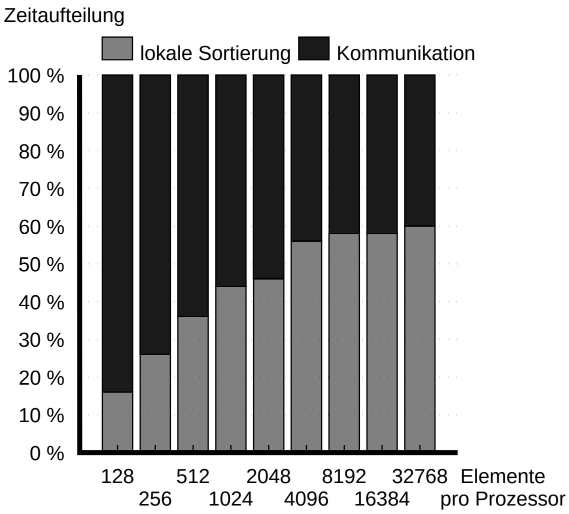
Datenblatt 11

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der log(N)-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





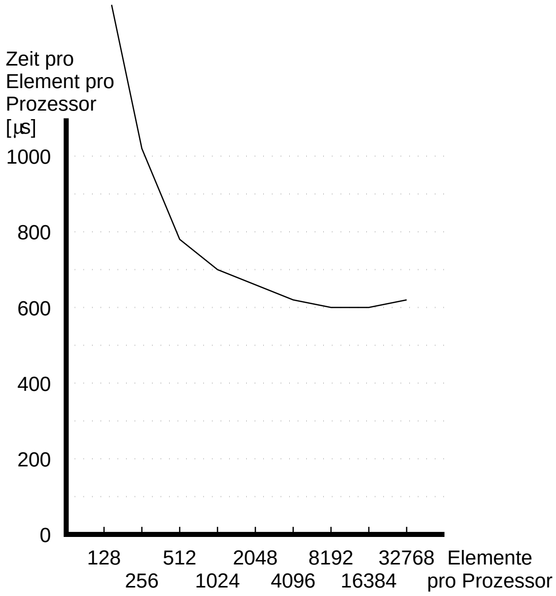


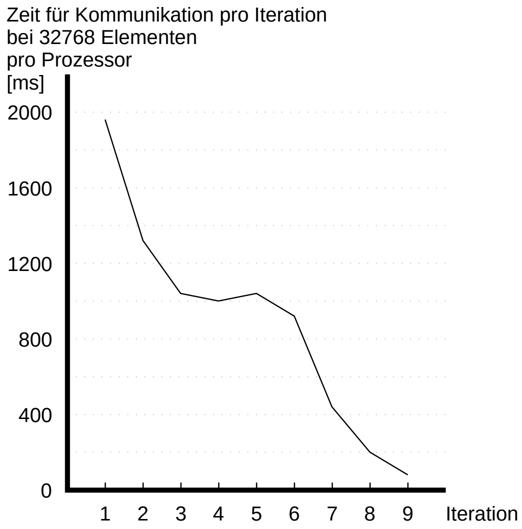


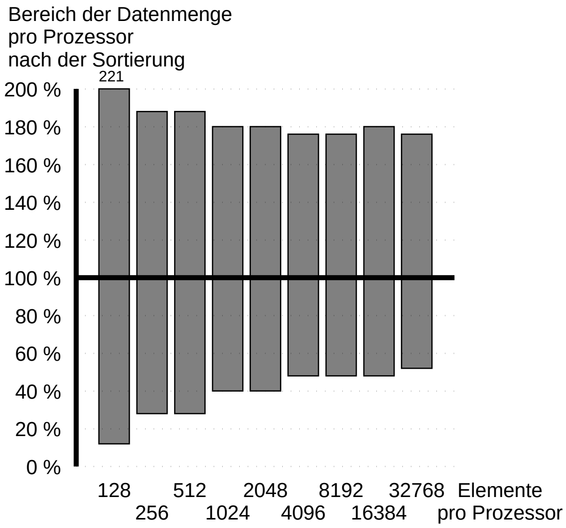
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1343	905	685	602	572	585	549	600	580
Gesamtzeit [s]	0.17	0.23	0.35	6.62	1.17	2.40	4.51	9.85	19.03
Zeit für Kommunikation [ms]	151	182	230	355	632	1104	1976	4244	8045
Zeit für lokale Sortierung [ms]	28	59	126	286	570	1387	2714	6023	11825
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	22	33	96	180	554	823	2558	3885	10515
Maximale Anzahl Elemente	265	545	1026	2095	3866	8377	15520	31581	58370
Anzahl Datenpakete	5698	5774	5703	5753	7359	11377	19939	36587	70153
Gesamtweglänge aller Datenpaket	24396	24092	24441	24216	30787	45293	82017	147053	284238
Durchschn. Iterationstiefe	8.12	8.20	8.11	8.16	8.10	8.14	8.08	8.12	8.08

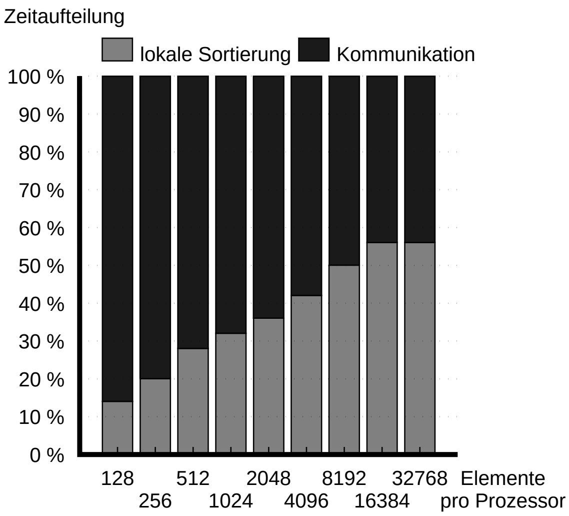
Datenblatt 12

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





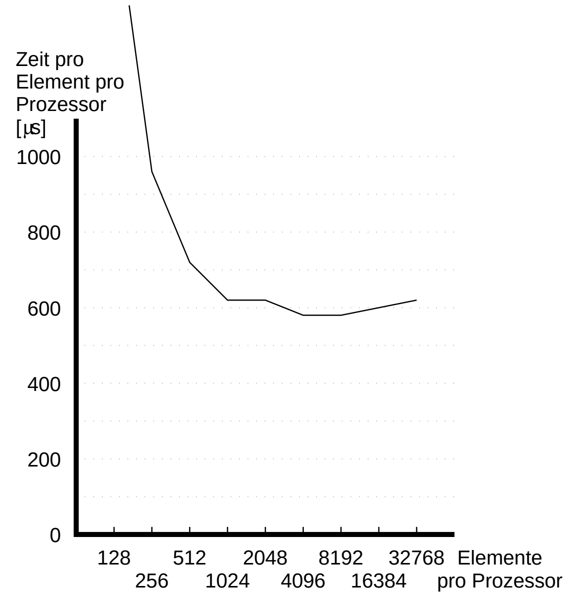


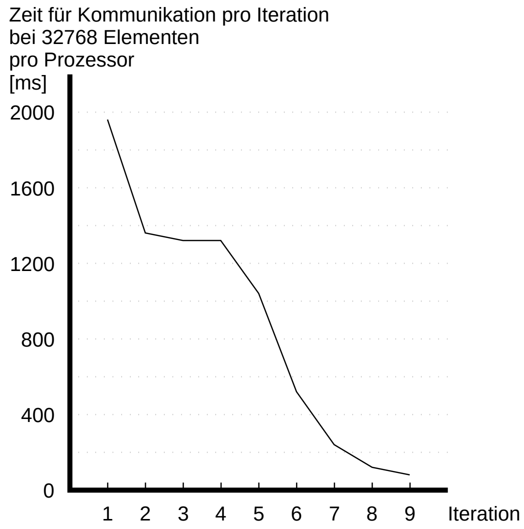


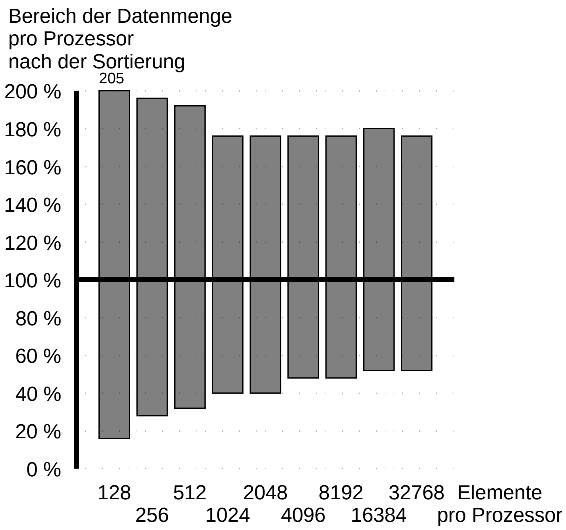
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1452	1022	784	695	663	614	597	610	611
Gesamtzeit [s]	0.19	0.26	0.40	0.71	1.36	2.52	4.90	10.04	20.05
Zeit für Kommunikation [ms]	168	218	305	517	945	1571	2581	4661	9237
Zeit für lokale Sortierung [ms]	29	52	117	248	547	1172	2552	5705	11727
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	17	74	151	426	840	1919	3813	8157	16812
Maximale Anzahl Elemente	284	483	959	1855	3682	7267	14469	29788	57356
Anzahl Datenpakete	5764	5692	5706	5670	7278	11655	20062	36818	70216
Gesamtweglänge aller Datenpaket	29686	30942	30967	30143	38225	57784	96999	164197	330664
Durchschn. Iterationstiefe	8.20	8.10	8.11	8.06	8.06	8.03	8.03	8.02	8.01

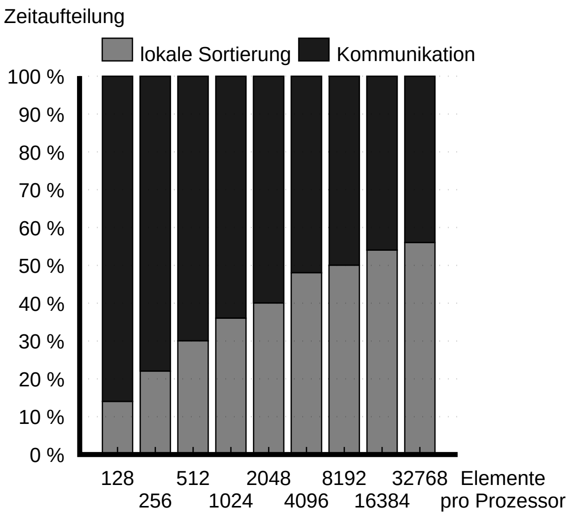
Datenblatt 13

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





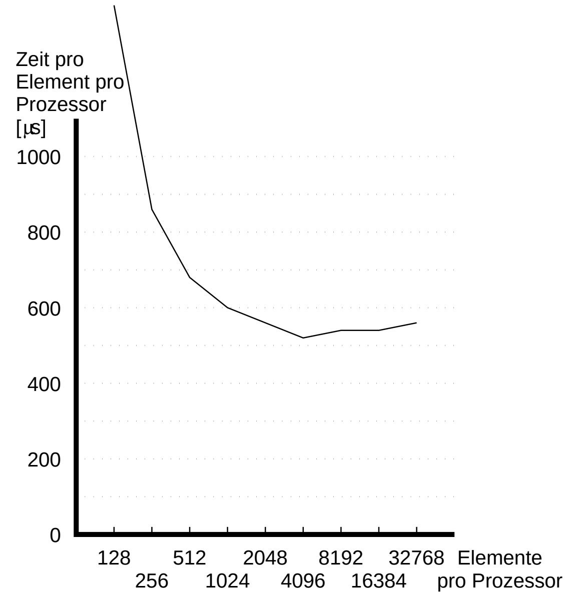


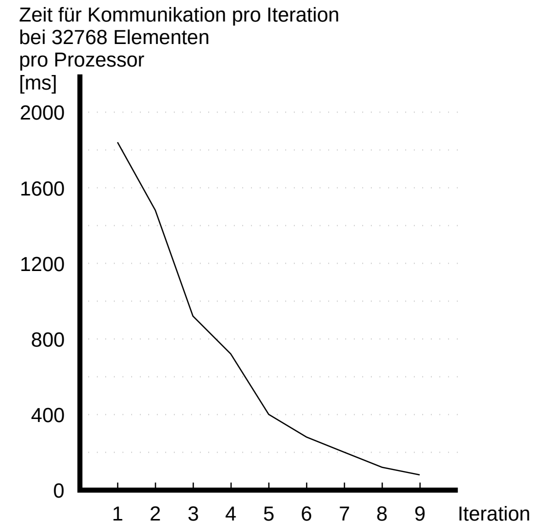


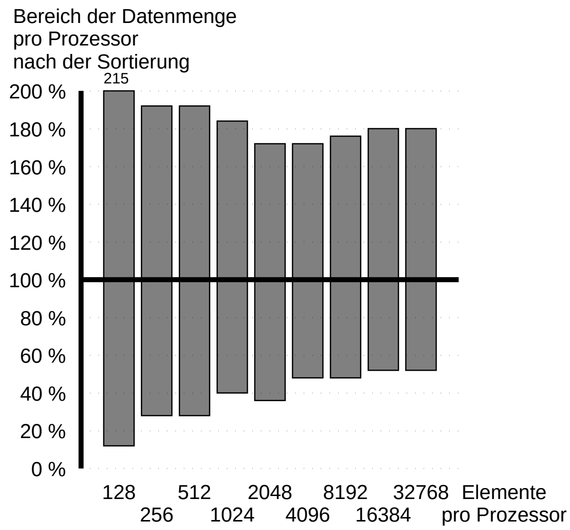
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1515	952	722	619	622	585	580	605	615
Gesamtzeit [s]	0.19	0.24	0.37	0.63	1.28	2.40	4.76	9.92	20.17
Zeit für Kommunikation [ms]	176	203	274	420	812	1312	2451	4655	9081
Zeit für lokale Sortierung [ms]	28	55	114	246	540	1184	2508	5604	11755
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	23	75	167	414	861	1996	3933	8266	17336
Maximale Anzahl Elemente	263	505	958	1810	3605	7266	14273	29606	57932
Anzahl Datenpakete	5769	5696	5706	5674	7246	11692	20037	36813	70185
Gesamtweglänge aller Datenpaket	27277	26706	26779	26880	32974	54059	91350	168108	317981
Durchschn. Iterationstiefe	8.21	8.11	8.11	8.07	8.06	8.03	8.03	8.02	8.01

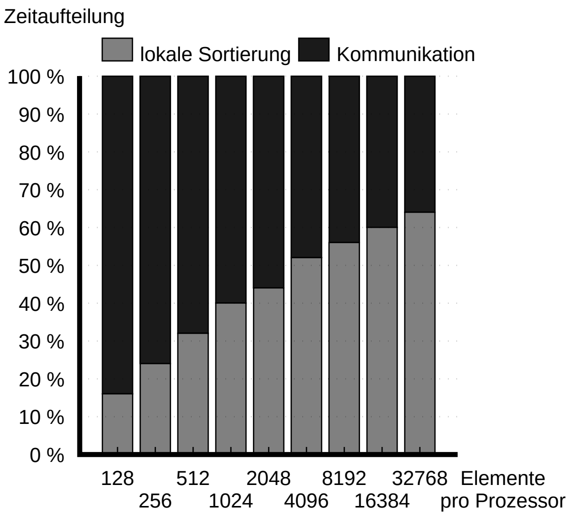
Datenblatt 14

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





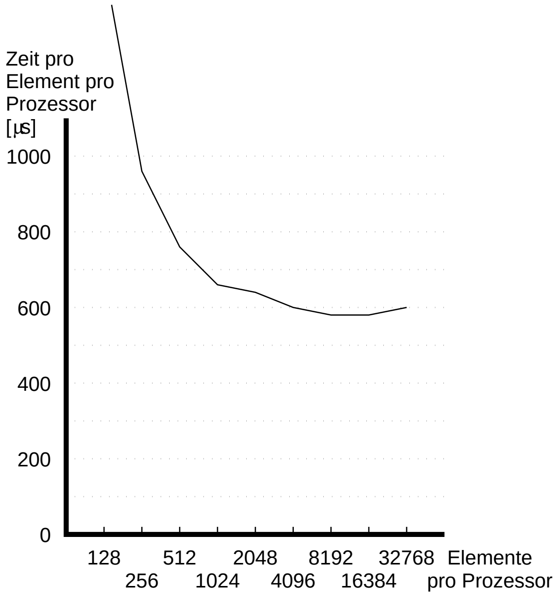


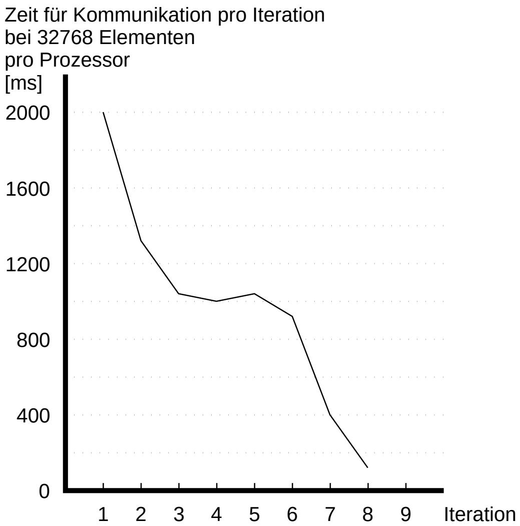


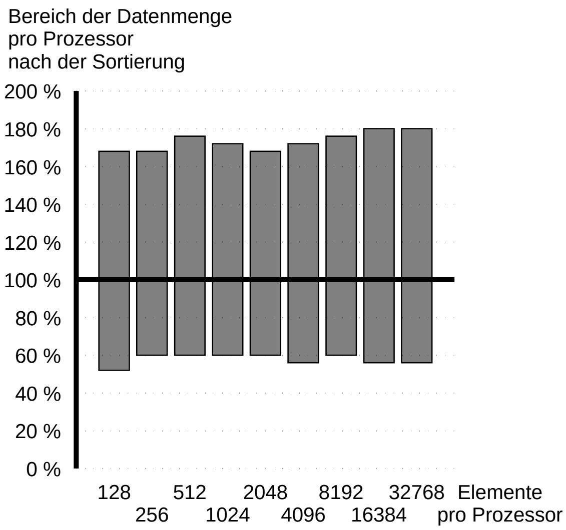
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1390	870	671	594	561	529	534	544	567
Gesamtzeit [s]	0.18	0.22	0.34	0.61	1.15	2.17	4.38	8.93	18.61
Zeit für Kommunikation [ms]	158	180	240	363	656	1120	2068	3553	6900
Zeit für lokale Sortierung [ms]	30	55	120	261	535	1136	2512	5605	12090
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	16	75	150	415	768	2024	3858	8271	17014
Maximale Anzahl Elemente	275	490	991	1903	3563	7105	14279	29407	59538
Anzahl Datenpakete	5757	5698	5699	5673	7214	11727	20075	36834	70184
Gesamtweglänge aller Datenpaket	23968	24315	24404	24358	29534	48470	82400	151431	286745
Durchschn. Iterationstiefe	8.19	8.10	8.10	8.06	8.07	8.03	8.03	8.02	8.02

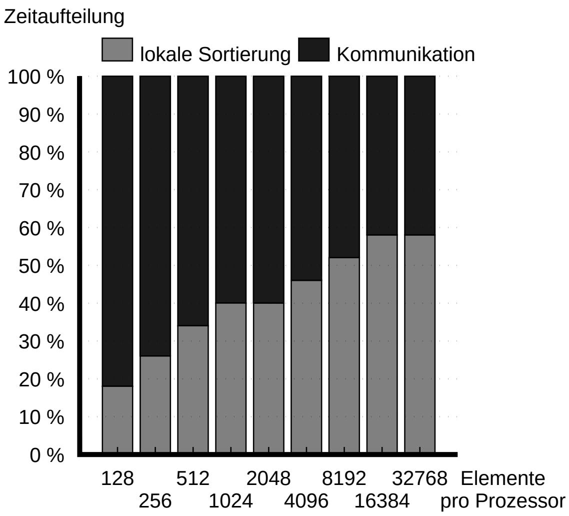
Datenblatt 15

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 2 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





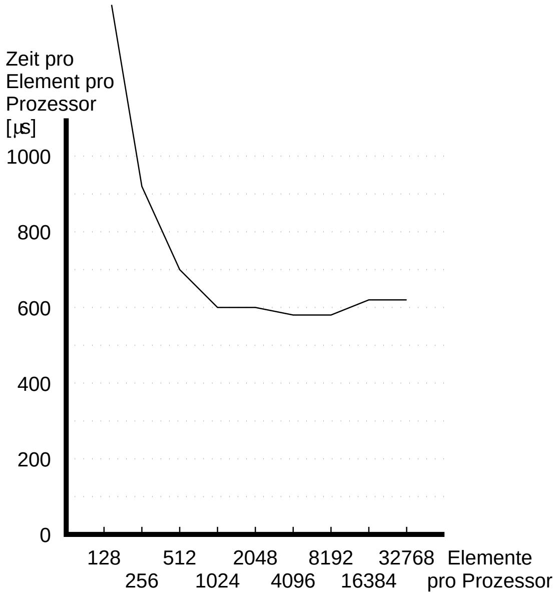


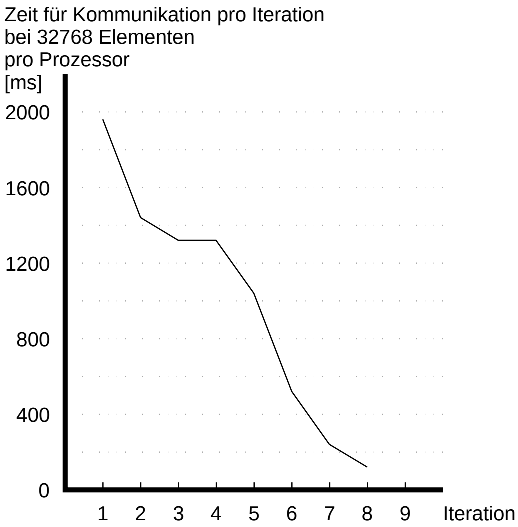


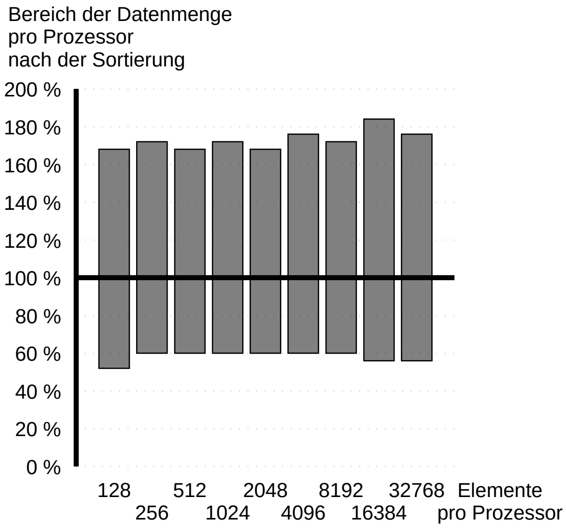
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1460	960	757	669	649	605	584	588	610
Gesamtzeit [s]	0.19	0.25	0.39	0.69	1.33	2.48	4.79	9.66	20.01
Zeit für Kommunikation [ms]	167	203	0.30	482	910	1472	2489	4486	8942
Zeit für lokale Sortierung [ms]	20	47	110	243	513	1157	2534	5563	11836
Zeit für Odd-Even-Sort [ms]	15	24	39	69	93	144	234	486	870
Minimale Anzahl Elemente	68	154	315	611	1192	2365	4762	9344	18988
Maximale Anzahl Elemente	213	432	896	1775	3470	7149	14423	29331	58502
Anzahl Datenpakete	5572	5496	5498	5550	7178	11517	19949	36759	70056
Gesamtweglänge aller Datenpaket	29837	30935	30934	30484	38660	58960	98820	167187	335409
Durchschn. Iterationstiefe	7.44	7.30	7.30	7.19	7.19	7.09	7.07	7.05	7.04

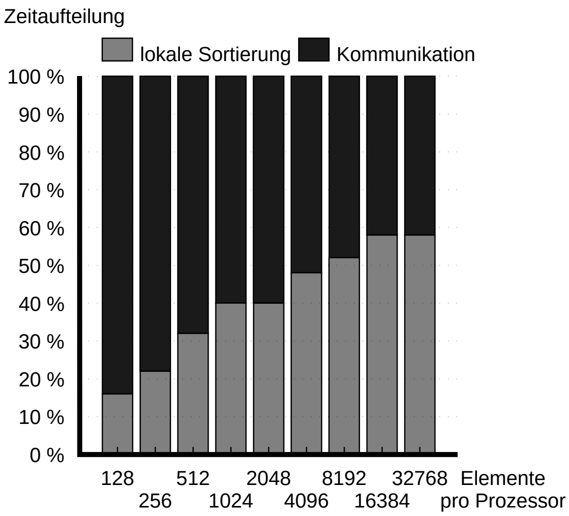
Datenblatt 16

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 2 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





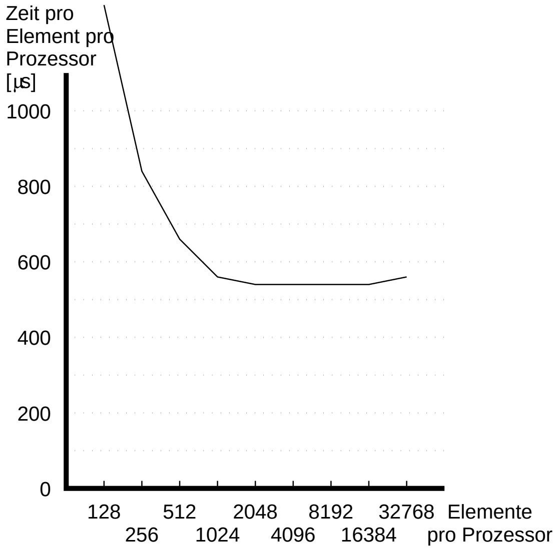


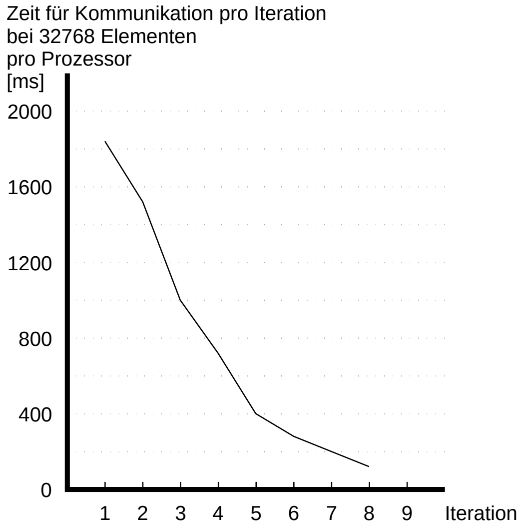


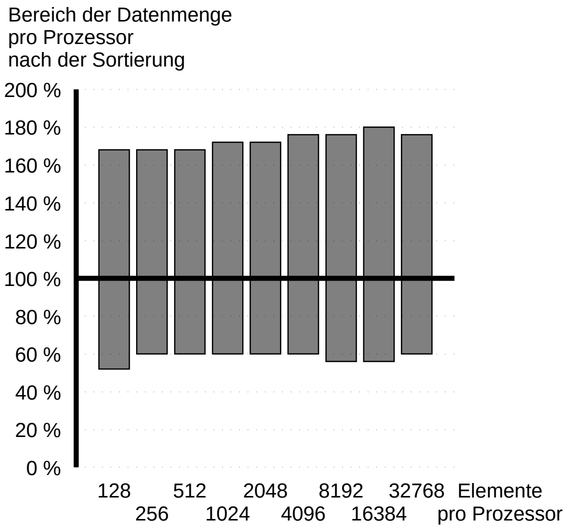
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1460	921	698	602	601	576	583	614	616
Gesamtzeit [s]	0.19	0.24	0.36	0.62	1.23	2.36	4.78	10.07	20.21
Zeit für Kommunikation [ms]	169	195	266	409	790	1320	2414	4567	8913
Zeit für lokale Sortierung [ms]	20	48	102	237	515	1154	2531	5729	11805
Zeit für Odd-Even-Sort [ms]	10	10	20	30	52	97	186	350	709
Minimale Anzahl Elemente	70	154	312	603	1235	2376	4780	9378	18905
Maximale Anzahl Elemente	212	436	863	1760	3448	7188	14244	29981	57631
Anzahl Datenpakete	5559	5479	5502	5549	7165	11583	19905	36645	70074
Gesamtweglänge aller Datenpaket	27944	27429	27496	28151	35211	58343	99494	184515	350933
Durchschn. Iterationstiefe	7.42	7.28	7.30	7.18	7.19	7.09	7.09	7.05	7.03

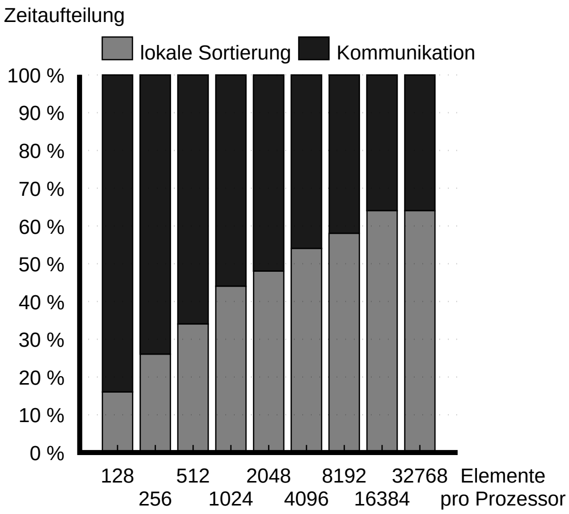
Datenblatt 17

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 2 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





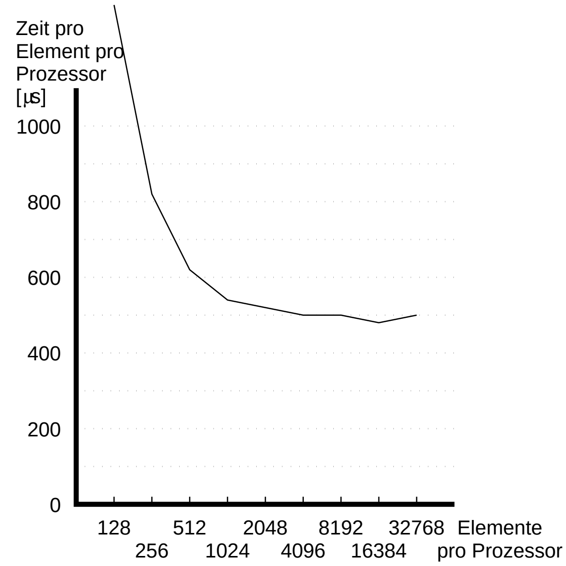


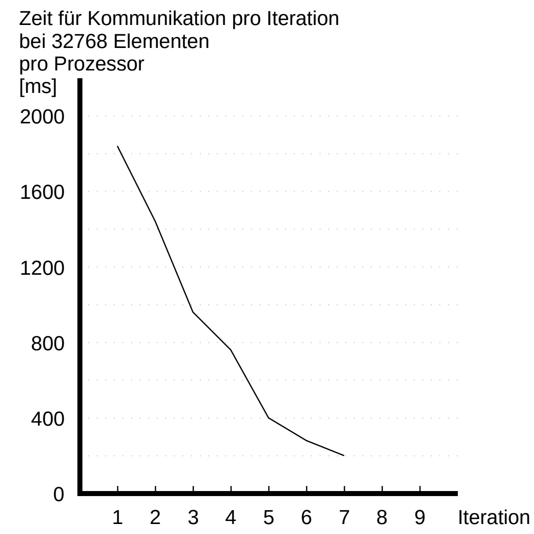


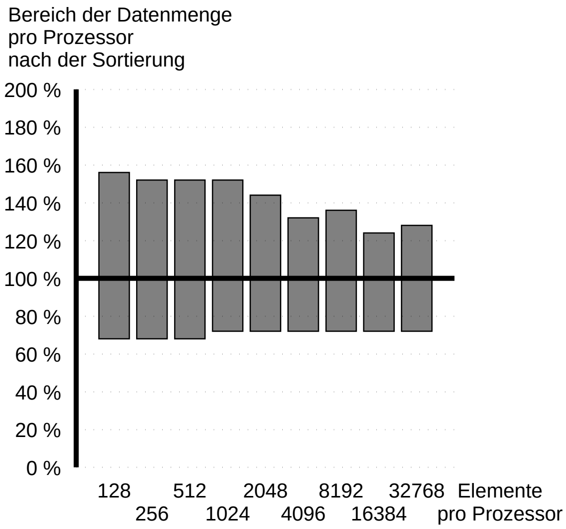
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1280	847	644	553	545	536	537	546	557
Gesamtzeit [s]	0.16	0.22	0.33	0.57	1.11	2.20	4.41	8.96	18.27
Zeit für Kommunikation [ms]	151	173	232	353	641	1091	2004	3452	6789
Zeit für lokale Sortierung [ms]	20	48	102	240	525	1176	2596	5614	11677
Zeit für Odd-Even-Sort [ms]	10	10	20	30	51	96	186	359	708
Minimale Anzahl Elemente	71	156	310	611	1195	2407	4693	9652	18846
Maximale Anzahl Elemente	216	434	867	1760	3511	7189	14621	29541	58123
Anzahl Datenpakete	5577	5498	5483	5552	7107	11623	19940	36775	70049
Gesamtweglänge aller Datenpaket	25299	25741	25855	26657	33469	56437	97230	182348	346793
Durchschn. Iterationstiefe	7.46	7.30	7.26	7.20	7.19	7.09	7.09	7.03	7.04

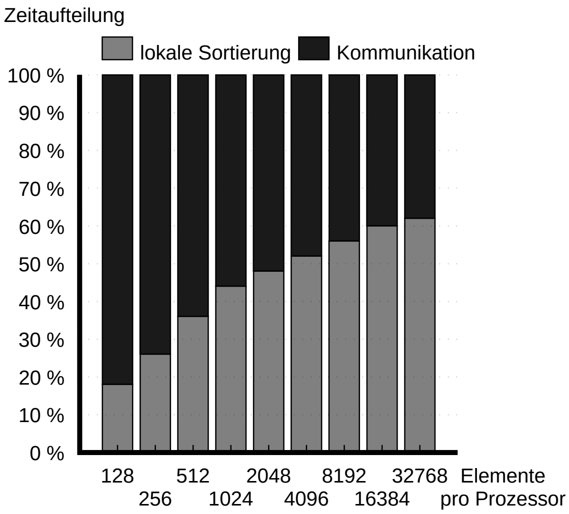
Datenblatt 18

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 3 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





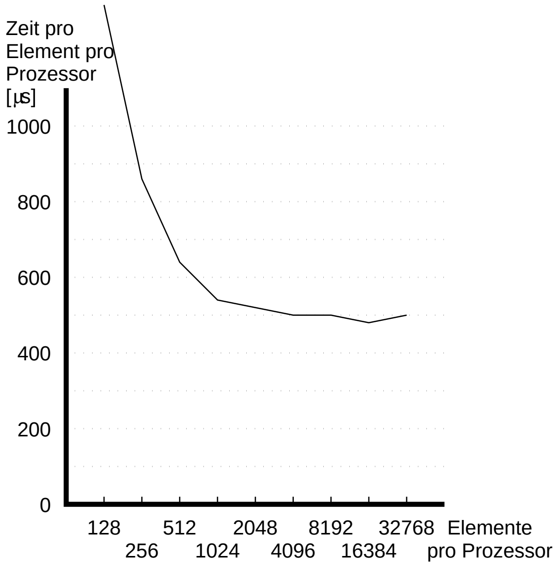


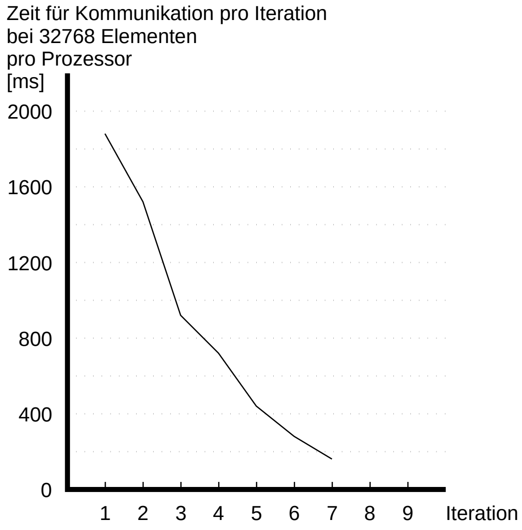


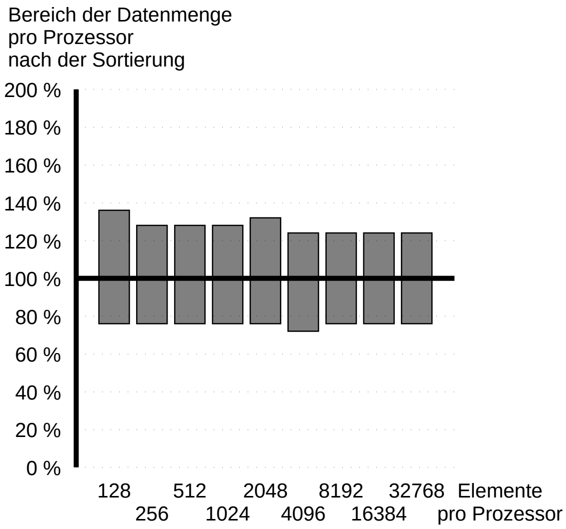
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1320	823	630	539	521	500	494	490	501
Gesamtzeit [s]	0.17	0.21	0.32	0.55	1.07	2.05	4.05	8.04	16.44
Zeit für Kommunikation [ms]	149	170	226	344	623	1073	1950	3397	6615
Zeit für lokale Sortierung [ms]	20	41	90	207	428	862	1933	3890	8419
Zeit für Odd-Even-Sort [ms]	13	20	37	69	128	261	517	1032	2076
Minimale Anzahl Elemente	85	179	352	735	1445	3020	5925	12082	23485
Maximale Anzahl Elemente	198	387	784	1563	2934	5406	11158	20541	41331
Anzahl Datenpakete	5416	5367	5368	5552	7166	11699	20099	36873	70453
Gesamtweglänge aller Datenpaket	26655	26854	26897	28134	35910	58494	101810	185917	355964
Durchschn. Iterationstiefe	6.85	6.80	6.80	6.84	6.83	6.93	6.90	6.96	6.94

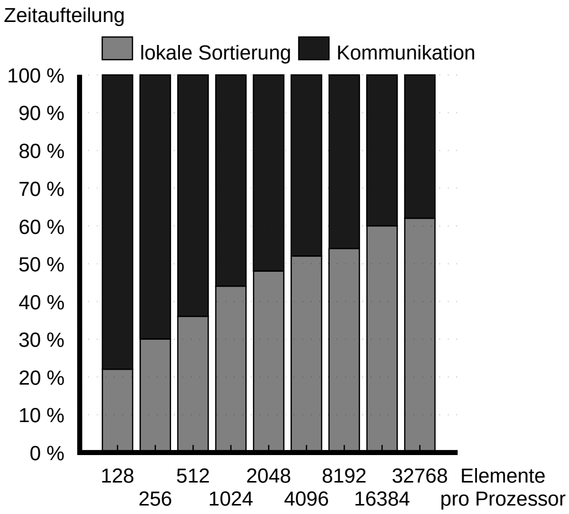
Datenblatt 19

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 4 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





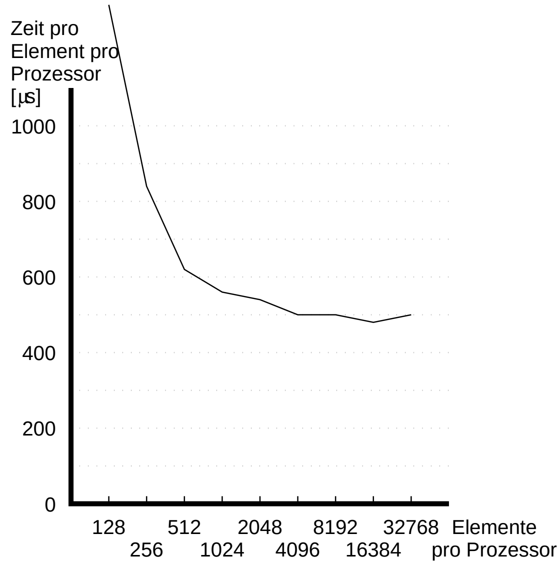


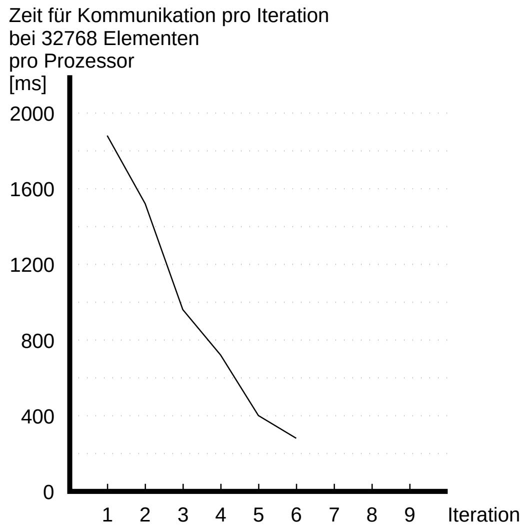


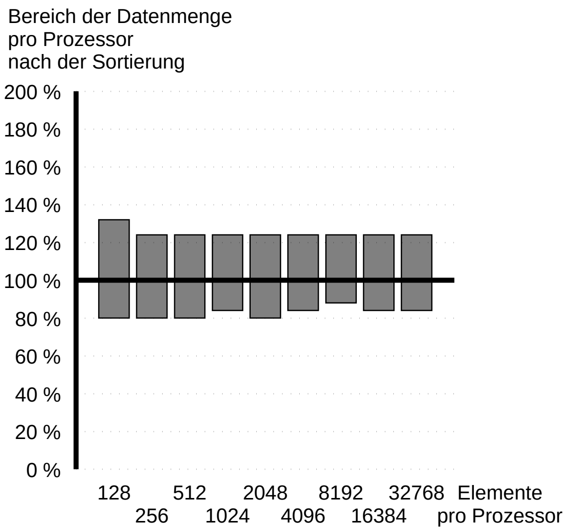
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1312	855	632	547	527	505	497	490	510
Gesamtzeit [s]	0.17	0.22	0.32	0.56	1.08	2.07	4.08	8.04	16.75
Zeit für Kommunikation [ms]	150	170	225	340	616	1046	1948	3357	6614
Zeit für lokale Sortierung [ms]	20	40	83	177	396	815	1761	3821	8442
Zeit für Odd-Even-Sort [ms]	20	30	46	84	157	303	604	1178	2363
Minimale Anzahl Elemente	96	195	393	778	1553	3025	6105	12305	24311
Maximale Anzahl Elemente	175	330	664	1297	2667	5071	10122	20238	40840
Anzahl Datenpakete	5356	5303	5303	5634	7393	12425	21568	40541	77585
Gesamtweglänge aller Datenpaket	27711	28354	28372	31361	41090	71718	126290	241762	463534
Durchschn. Iterationstiefe	6.50	6.37	6.37	6.24	6.29	6.14	6.15	6.07	6.08

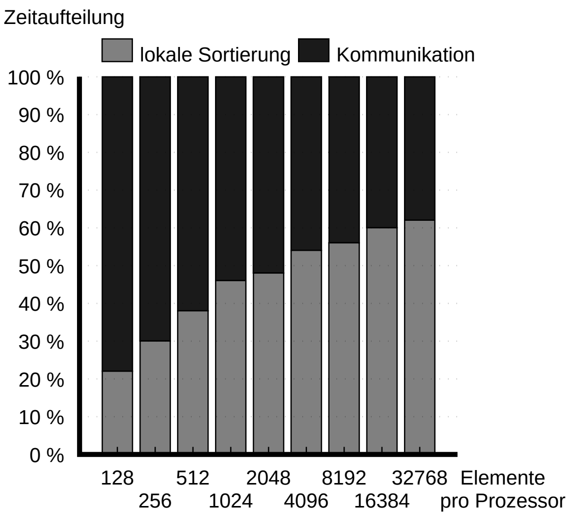
Datenblatt 20

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 5 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





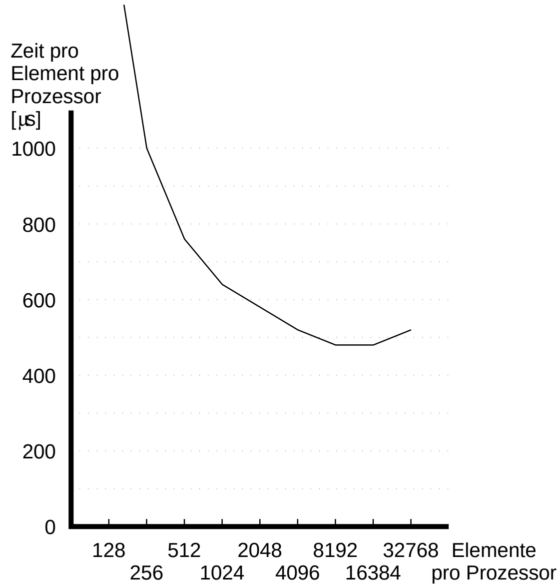


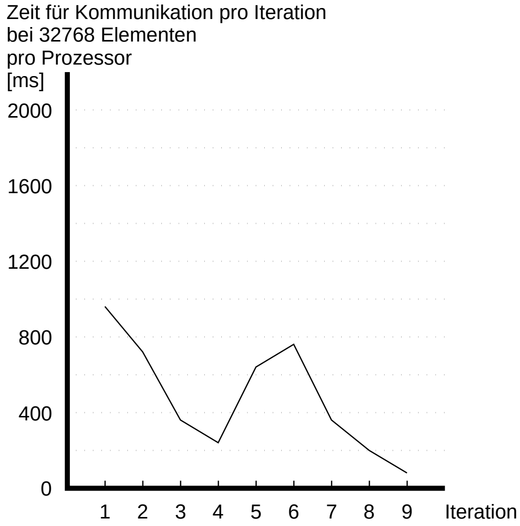


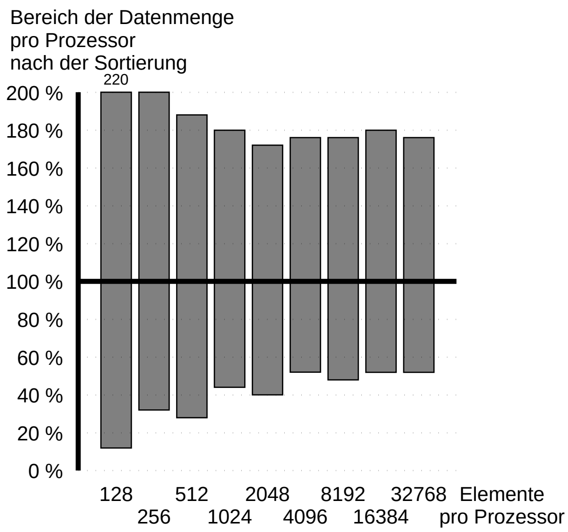
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1328	835	632	555	535	509	494	489	499
Gesamtzeit [s]	0.17	0.21	0.32	0.57	1.10	2.09	4.05	8.04	16.40
Zeit für Kommunikation [ms]	141	169	218	334	608	1023	1880	3257	6360
Zeit für lokale Sortierung [ms]	20	39	80	176	379	828	1777	3818	8087
Zeit für Odd-Even-Sort [ms]	20	31	58	99	195	342	688	1267	2545
Minimale Anzahl Elemente	101	206	416	852	1679	3481	7052	13984	28094
Maximale Anzahl Elemente	167	321	642	1267	2540	5060	10105	20325	40130
Anzahl Datenpakete	5332	5287	5291	5675	7580	12604	21963	40891	78483
Gesamtweglänge aller Datenpaket	28859	29526	29626	32754	43916	74510	131764	246694	474808
Durchschn. Iterationstiefe	6.19	6.06	6.05	6.05	6.03	6.00	6.00	6.00	6.00

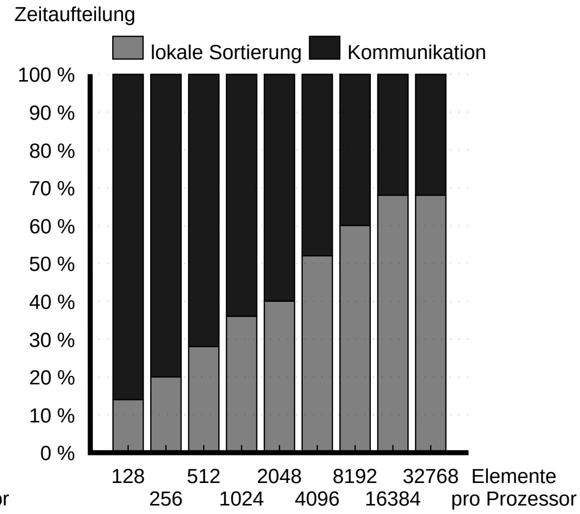
Datenblatt 21

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





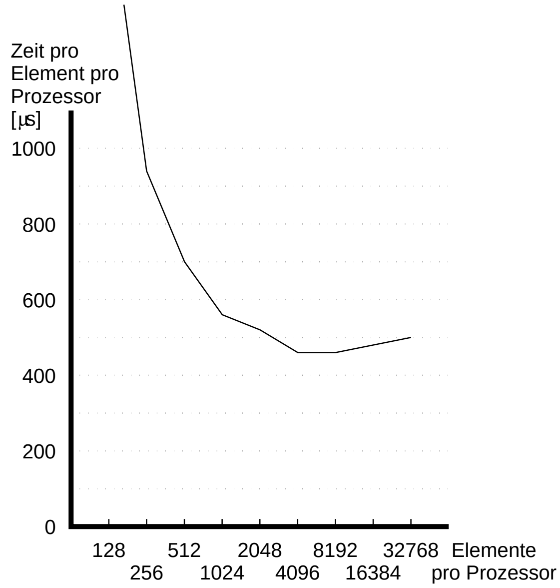


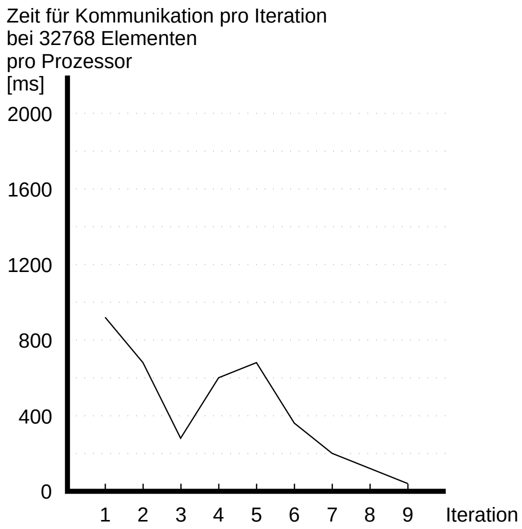


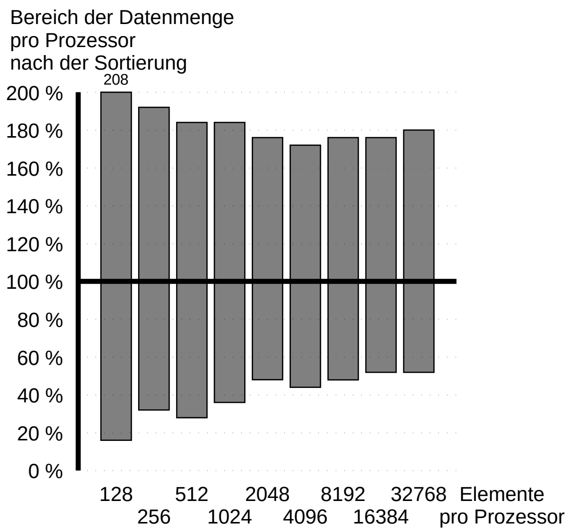
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1601	1011	753	633	582	514	485	471	521
Gesamtzeit [s]	0.20	0.26	0.39	0.65	1.19	2.11	3.98	7.73	17.10
Zeit für Kommunikation [ms]	188	223	302	456	749	1110	1691	2726	5777
Zeit für lokale Sortierung [ms]	29	58	116	252	534	1177	3553	5535	11935
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	17	79	141	448	819	2072	3853	8207	17181
Maximale Anzahl Elemente	284	515	944	1842	3560	7261	14410	29339	58252
Anzahl Datenpakete	3663	3620	3631	3606	3919	6175	10323	18824	35498
Gesamtweglänge aller Datenpaket	26629	22662	24765	23142	24967	34219	55362	89271	183908
Durchschn. Iterationstiefe	8.20	8.10	8.11	8.06	8.06	8.03	8.03	8.02	8.02

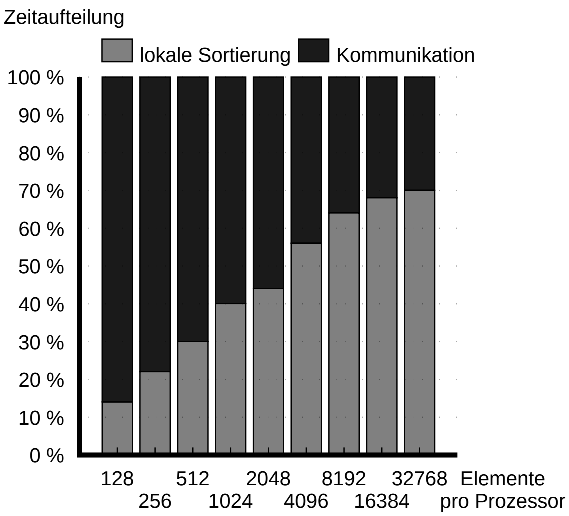
Datenblatt 22

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





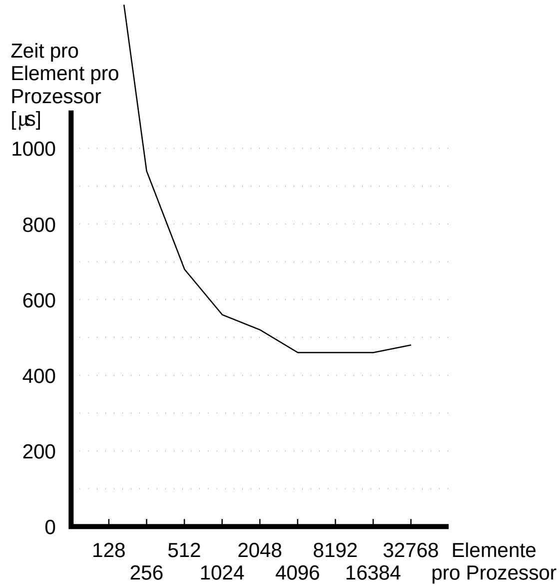


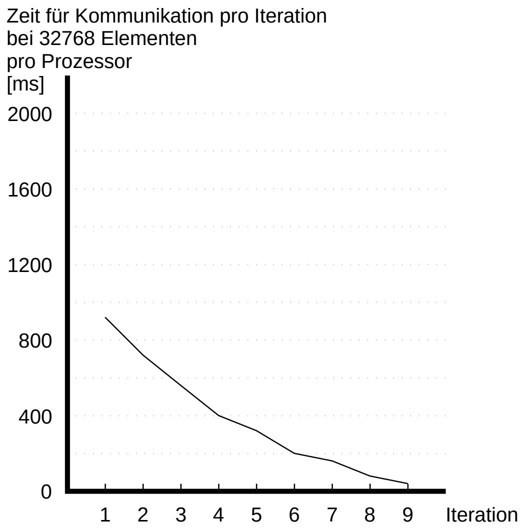


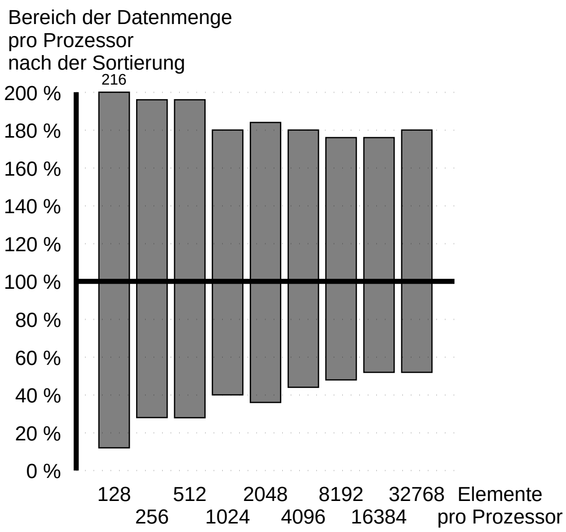
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1679	948	691	560	515	467	463	471	501
Gesamtzeit [s]	0.21	0.24	0.35	0.57	1.06	1.92	3.80	7.73	16.43
Zeit für Kommunikation [ms]	198	200	258	369	596	884	1443	2583	5024
Zeit für lokale Sortierung [ms]	30	56	115	251	542	1129	2540	5551	12036
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	18	80	145	374	951	1868	4027	8330	16937
Maximale Anzahl Elemente	267	489	940	1884	3622	7014	14288	28977	58834
Anzahl Datenpakete	3667	3623	3631	3617	3925	6139	10337	18699	35520
Gesamtweglänge aller Datenpaket	23179	20150	21369	20573	21964	31703	52735	93023	178591
Durchschn. Iterationstiefe	8.21	8.10	8.10	8.07	8.06	8.02	8.03	8.02	8.02

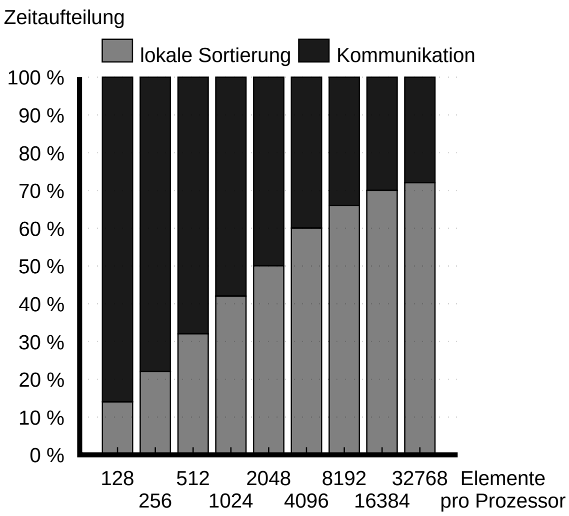
Datenblatt 23

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





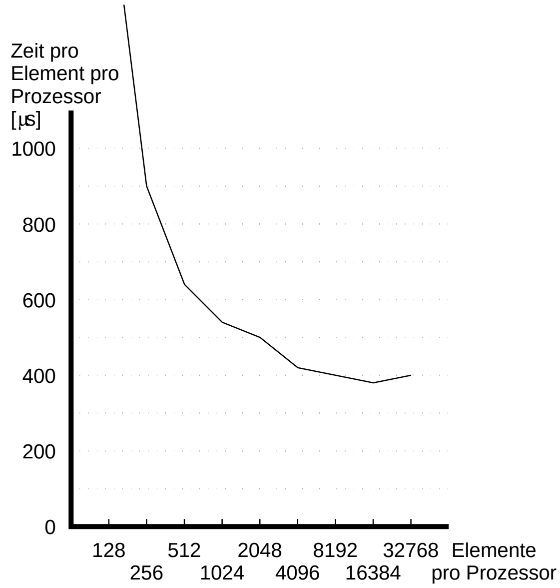


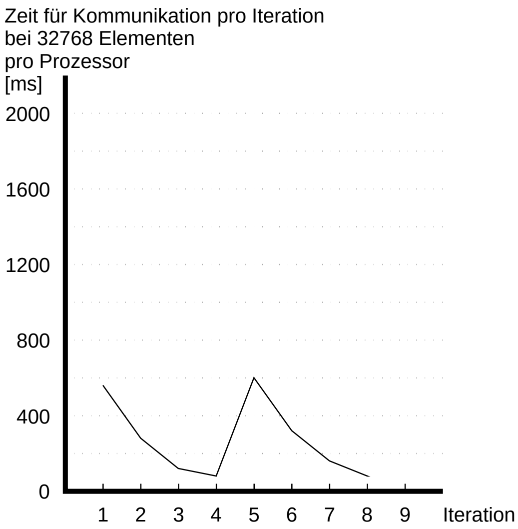


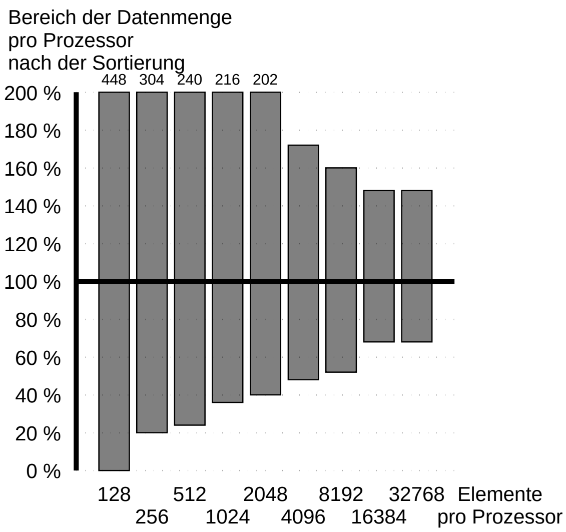
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1554	944	689	555	518	468	456	466	489
Gesamtzeit [s]	0.20	0.24	0.35	0.57	1.06	1.92	3.74	7.65	16.03
Zeit für Kommunikation [ms]	179	197	244	350	564	815	1337	2371	4505
Zeit für lokale Sortierung [ms]	29	55	120	246	552	1176	2522	5498	12039
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	17	74	153	430	771	1869	3856	8393	16738
Maximale Anzahl Elemente	279	506	1008	1826	3732	7357	14443	28849	58837
Anzahl Datenpakete	3666	3624	3625	3616	3943	6127	10344	18805	35533
Gesamtweglänge aller Datenpaket	24048	24097	24154	24219	25107	38256	64560	116644	219934
Durchschn. Iterationstiefe	8.21	8.10	8.10	8.07	8.07	8.03	8.03	8.02	8.02

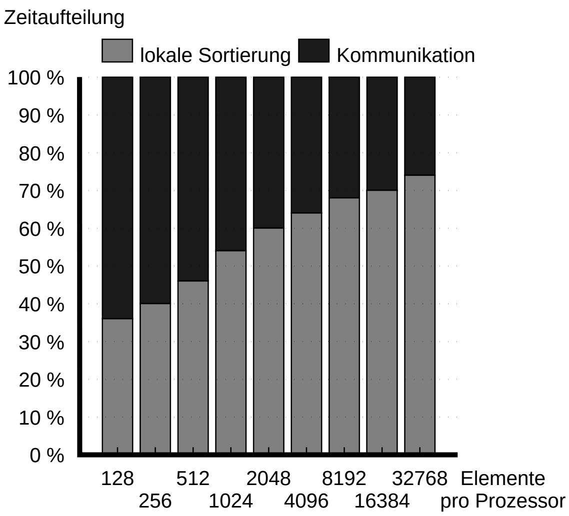
Datenblatt 24

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





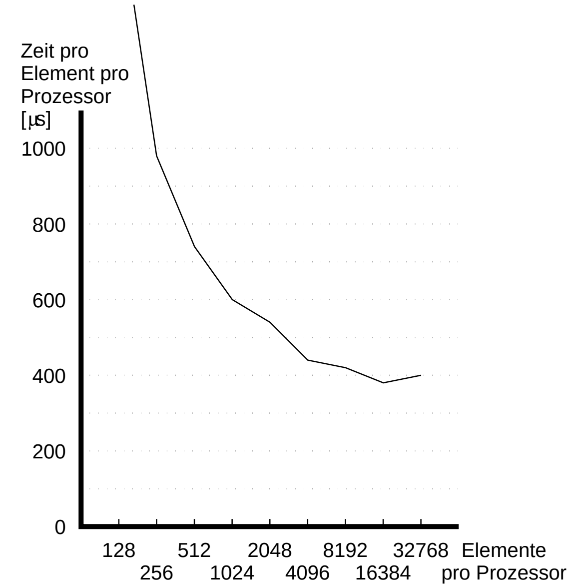


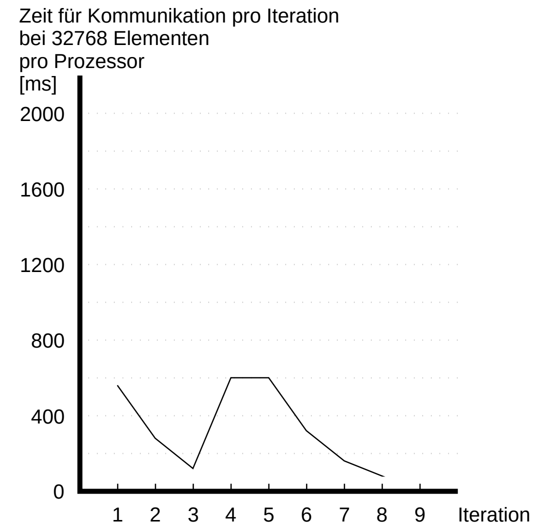


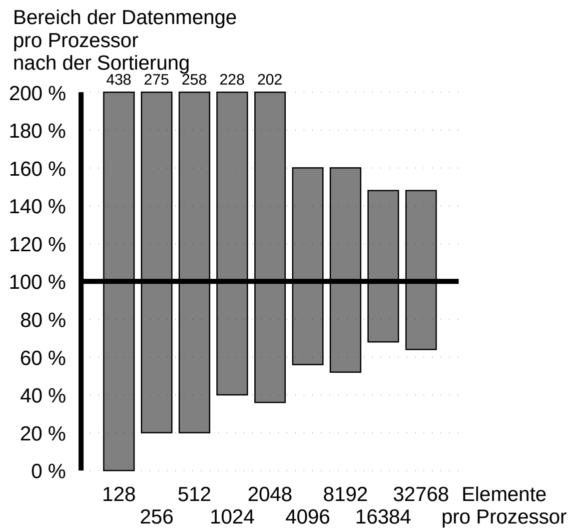
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1445	909	647	540	496	423	400	381	394
Gesamtzeit [s]	0.18	0.23	0.33	0.55	1.02	1.74	3.28	6.26	12.94
Zeit für Kommunikation [ms]	120	140	181	251	406	618	1057	1851	3399
Zeit für lokale Sortierung [ms]	65	94	159	305	615	1130	2273	4452	9637
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	1	52	116	368	789	1907	4246	10954	22338
Maximale Anzahl Elemente	573	781	1239	2206	4134	6965	13025	24030	48776
Anzahl Datenpakete	3537	3551	3558	3571	4357	6576	10760	19123	35785
Gesamtweglänge aller Datenpaket	15540	15568	15586	15684	18469	26565	42540	73573	135956
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

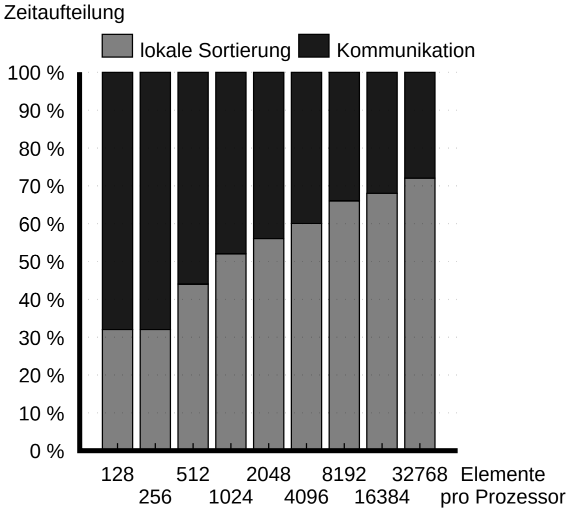
Datenblatt 25

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





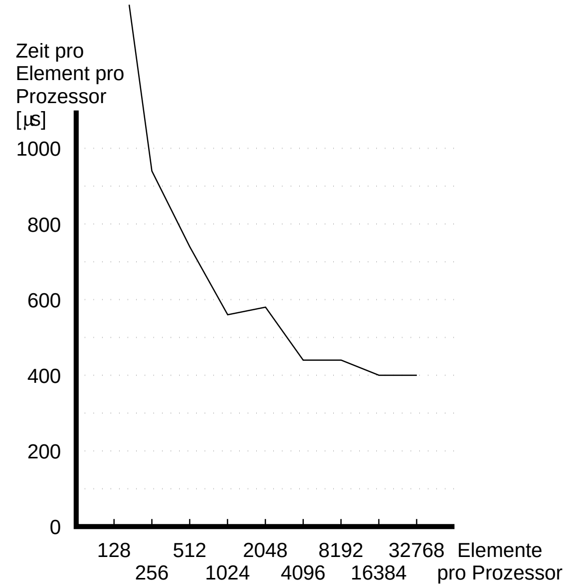


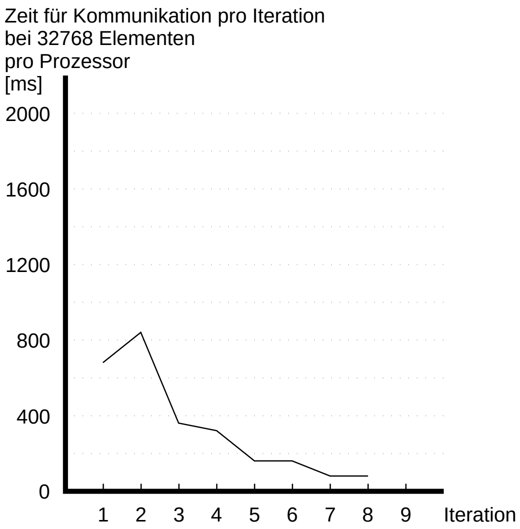


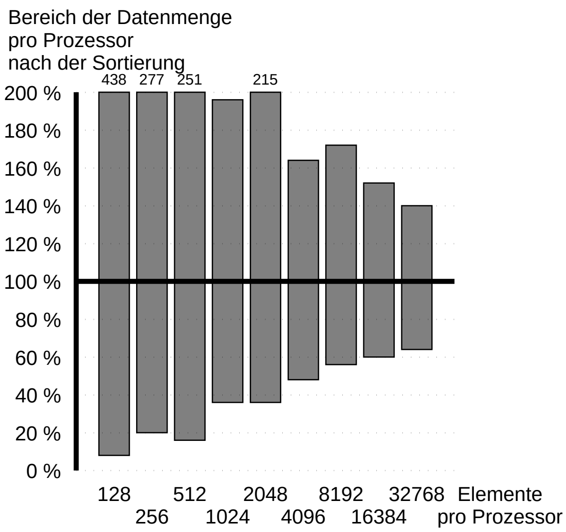
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1593	972	732	605	535	437	422	389	406
Gesamtzeit [s]	0.20	0.25	0.37	0.62	1.10	1.79	3.46	6.40	13.34
Zeit für Kommunikation [ms]	140	167	216	300	483	718	1174	2093	3911
Zeit für lokale Sortierung [ms]	63	82	169	322	617	1.08	2292	4338	9450
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	1	47	101	397	766	2218	4135	11052	20640
Maximale Anzahl Elemente	560	704	1323	2334	4139	6632	13106	23401	47224
Anzahl Datenpakete	3535	3542	3559	3574	4372	6571	10777	19121	35793
Gesamtweglänge aller Datenpaket	18539	18588	18628	18779	22414	32932	52930	92193	171421
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

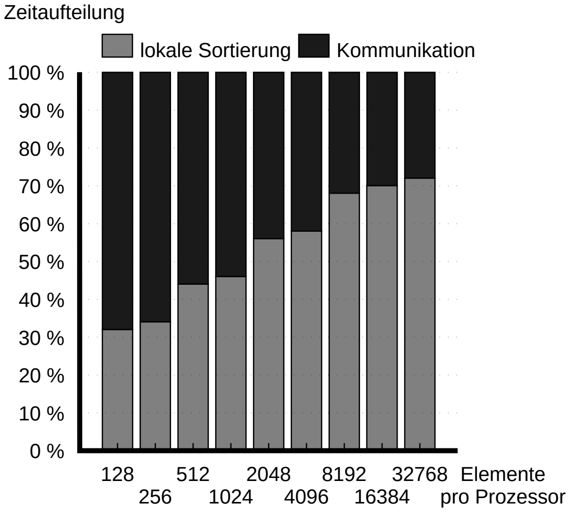
Datenblatt 26

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





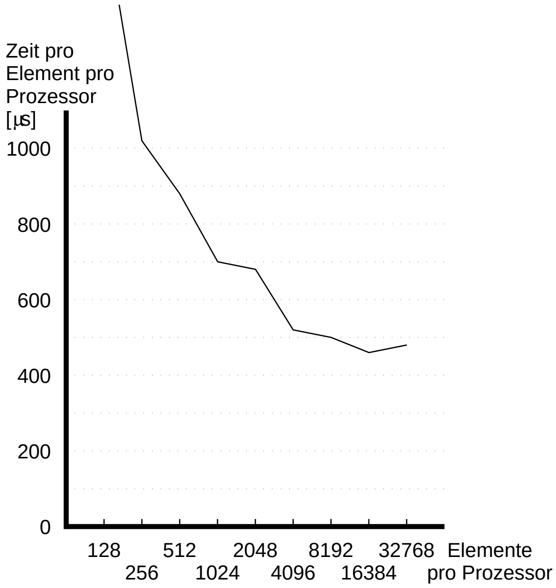


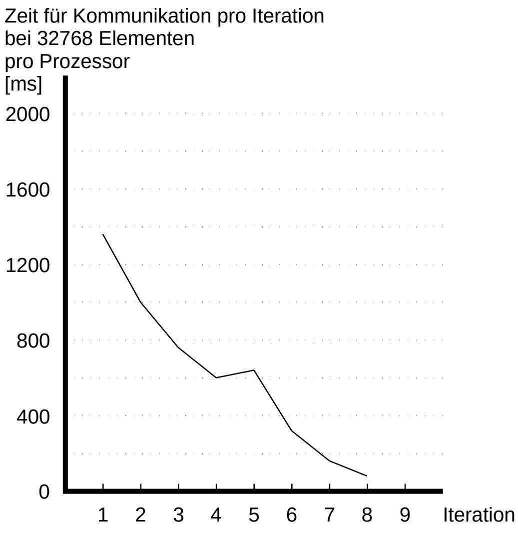


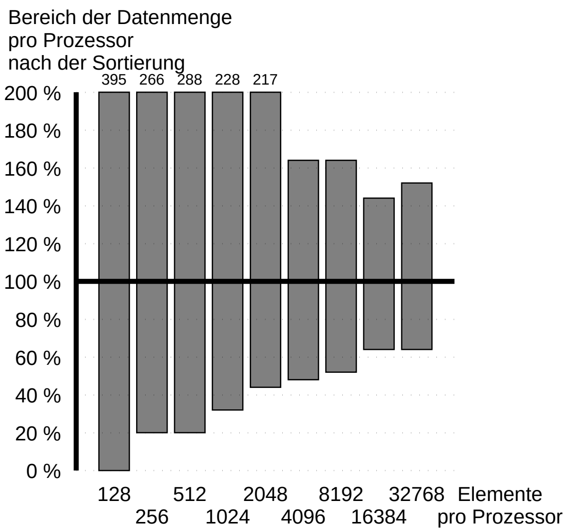
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1538	948	734	563	577	439	445	414	395
Gesamtzeit [s]	0.20	0.24	0.38	0.58	1.18	1.80	3.65	6.80	12.96
Zeit für Kommunikation [ms]	140	161	217	311	513	753	1161	2037	3773
Zeit für lokale Sortierung [ms]	63	81	165	274	670	1056	2502	4779	9307
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	8	54	90	373	756	1969	4610	10038	20923
Maximale Anzahl Elemente	560	711	1288	2018	4397	6657	14205	25086	46402
Anzahl Datenpakete	3536	3555	3559	3573	4386	6557	10788	19131	35772
Gesamtweglänge aller Datenpaket	23919	23913	23966	24109	28185	41429	68071	118941	221541
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

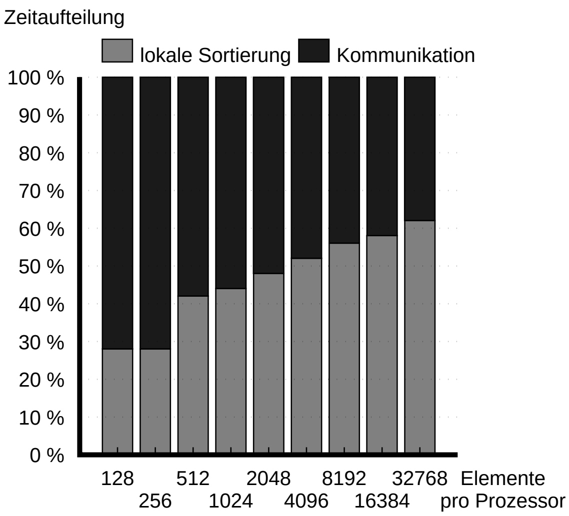
Datenblatt 27

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





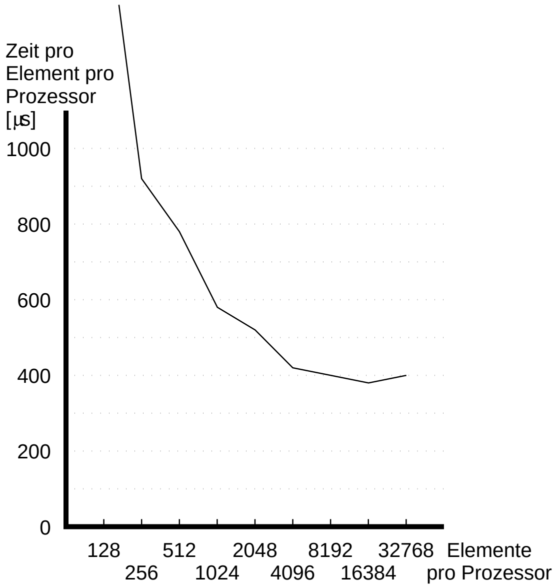


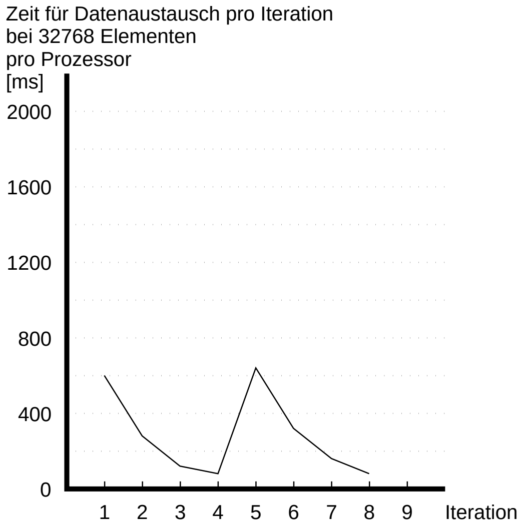


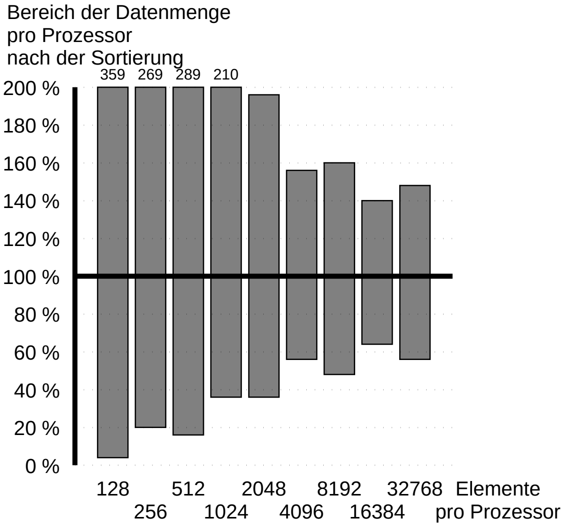
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1499	1015	888	708	680	526	497	463	482
Gesamtzeit [s]	0.19	0.26	0.45	0.73	1.39	2.18	4.08	7.60	15.83
Zeit für Kommunikation [ms]	142	181	260	418	736	1086	1790	3177	6054
Zeit für lokale Sortierung [ms]	55	76	194	321	685	1086	2312	4451	9891
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	1	47	110	339	870	2014	4259	10258	20750
Maximale Anzahl Elemente	505	681	1478	2337	4446	6677	13493	23589	49536
Anzahl Datenpakete	3531	3556	3555	3571	4412	6536	10775	19121	35814
Gesamtweglänge aller Datenpaket	29201	29433	29322	29507	35432	51741	84664	149301	278246
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

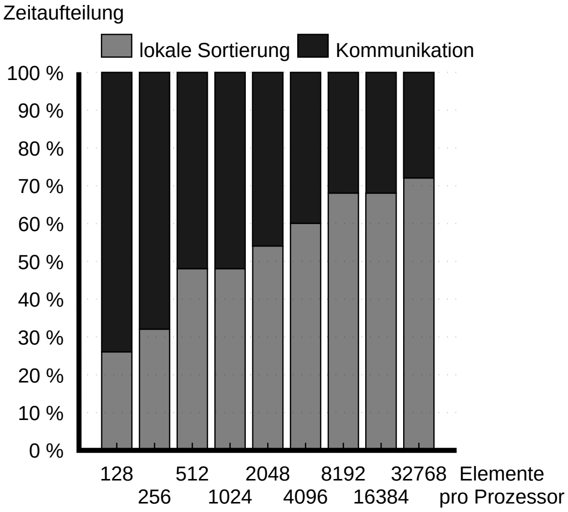
Datenblatt 28

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





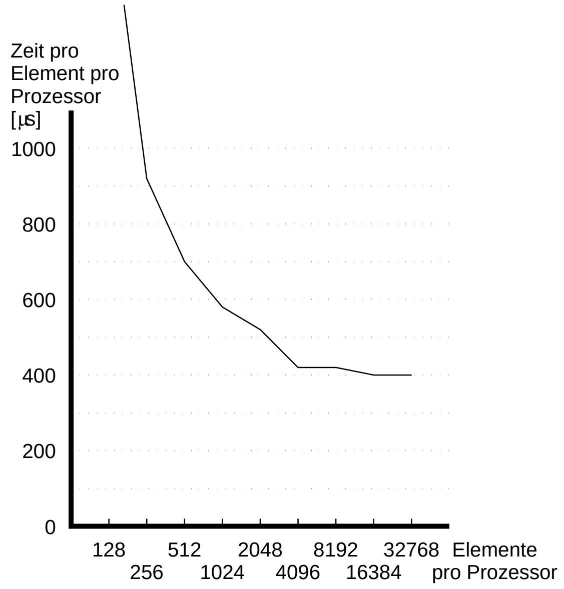


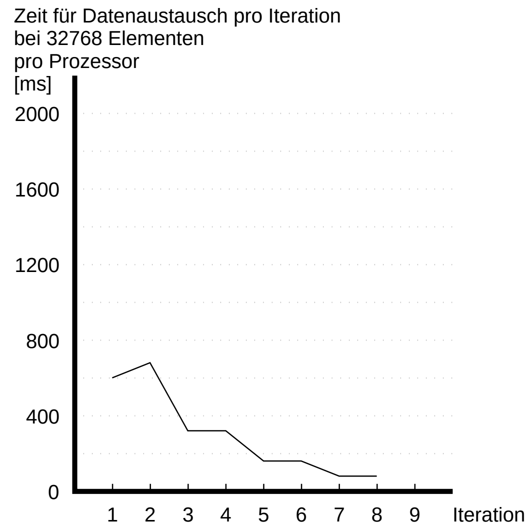


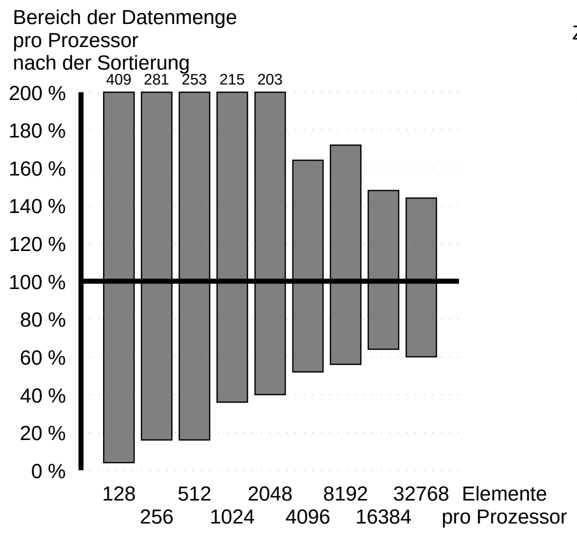
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1468	917	790	582	514	412	410	378	402
Gesamtzeit [s]	0.19	0.23	0.40	0.60	1.05	1.69	3.36	6.22	13.19
Zeit für Kommunikation [ms]	140	160	213	310	467	670	1076	1867	3498
Zeit für lokale Sortierung [ms]	49	76	193	298	591	1029	2299	4348	9735
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	3	52	85	374	747	2267	3910	10464	17727
Maximale Anzahl Elemente	459	689	1480	2158	4005	6435	13096	23233	49005
Anzahl Datenpakete	3530	3547	3551	3577	4380	6520	10776	19150	35834
Gesamtweglänge aller Datenpaket	15013	15070	15047	15187	18029	25894	41908	73226	135705
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

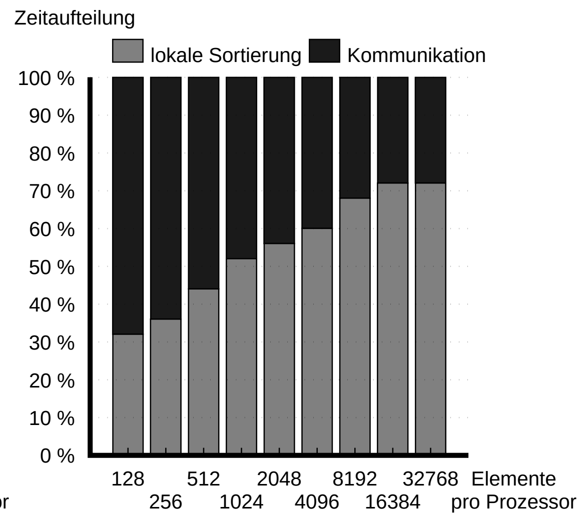
Datenblatt 29

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





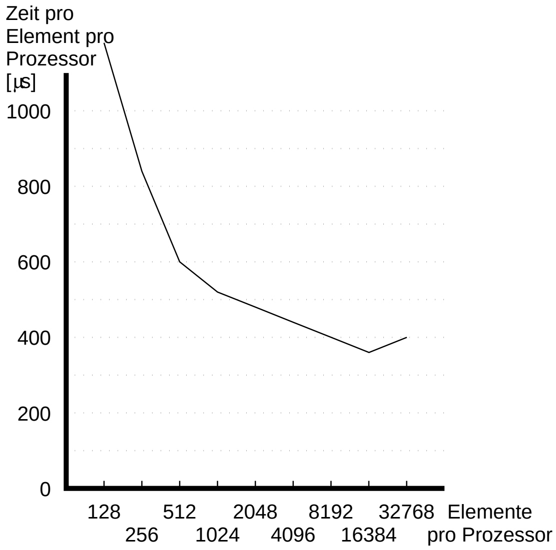


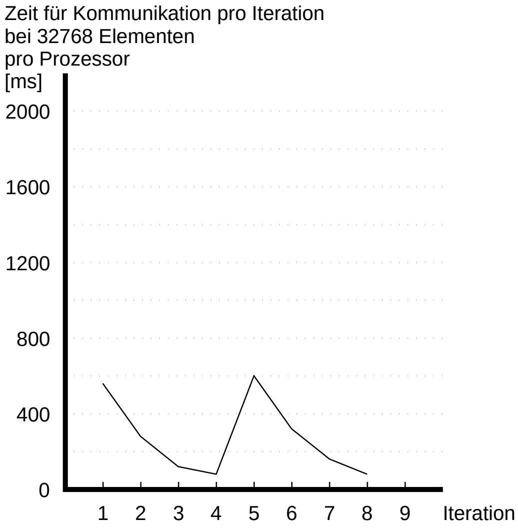


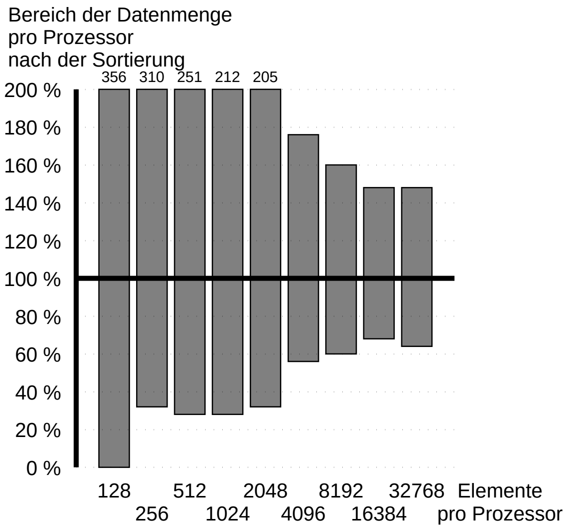
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1491	925	694	579	523	427	427	392	395
Gesamtzeit [s]	0.19	0.24	0.36	0.59	1.07	1.75	3.51	6.44	12.99
Zeit für Kommunikation [ms]	130	157	199	290	465	680	1093	1913	3512
Zeit für lokale Sortierung [ms]	57	83	167	303	607	1080	2431	4565	9518
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	3	46	81	336	823	2150	4733	10711	19446
Maximale Anzahl Elemente	524	721	1296	2205	4155	6694	14028	24176	47667
Anzahl Datenpakete	3525	3547	3553	3573	4387	6499	10776	19141	35822
Gesamtweglänge aller Datenpaket	17110	17231	17267	17376	20097	29210	48124	83724	155318
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

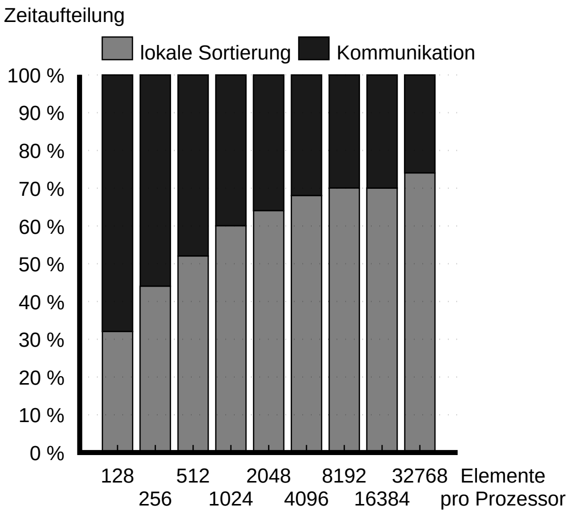
Datenblatt 30

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





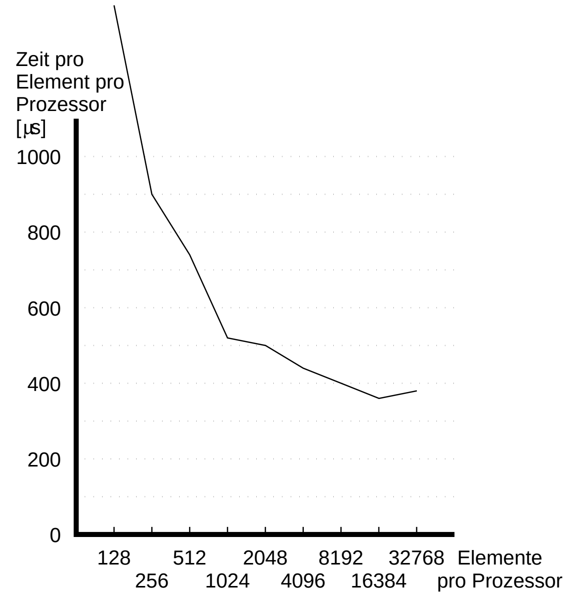


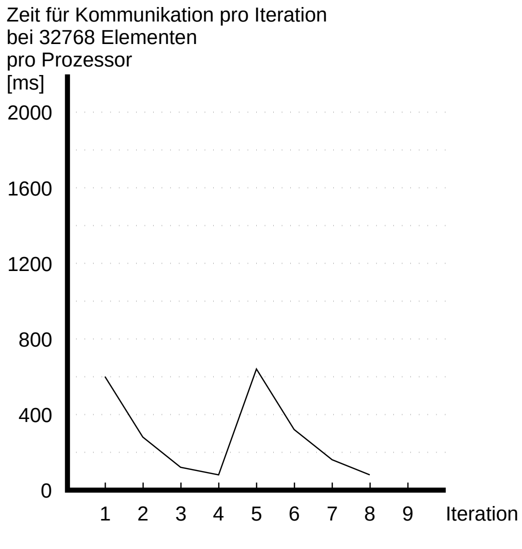


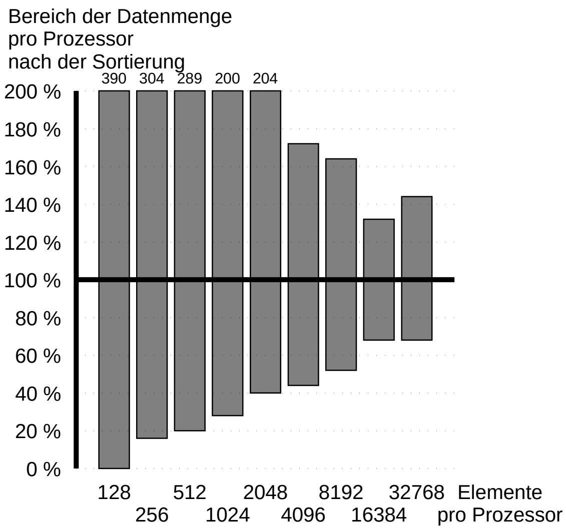
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1171	831	597	525	488	437	395	368	391
Gesamtzeit [s]	0.15	0.21	0.31	0.54	1.00	1.79	3.24	6.05	12.83
Zeit für Kommunikation [ms]	109	120	150	219	376	589	980	1814	3369
Zeit für lokale Sortierung [ms]	49	93	156	318	630	1200	2292	4335	9506
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	80	144	300	656	2298	4772	10983	21139
Maximale Anzahl Elemente	456	794	1286	2180	4215	6953	13183	23528	47285
Anzahl Datenpakete	2048	2048	2048	2049	3217	5375	9572	17929	34582
Gesamtweglänge aller Datenpaket	7680	7680	7680	7681	12018	20303	36134	67504	129597
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

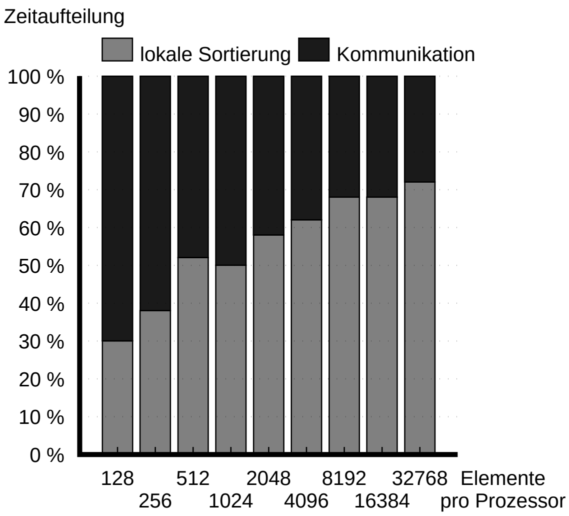
Datenblatt 31

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





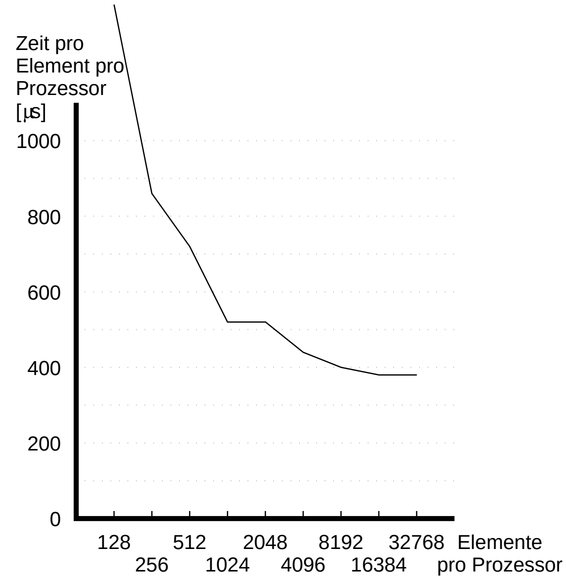


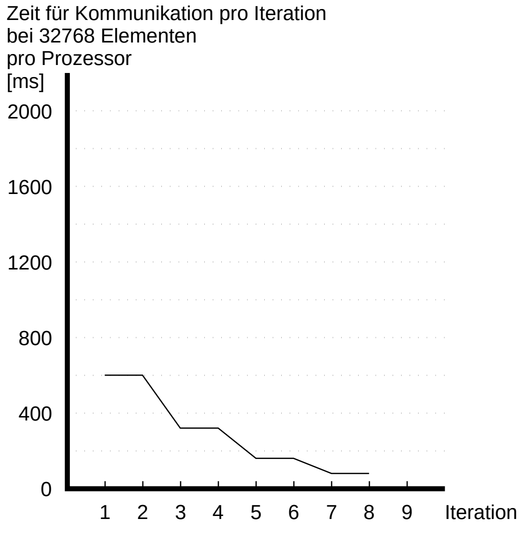


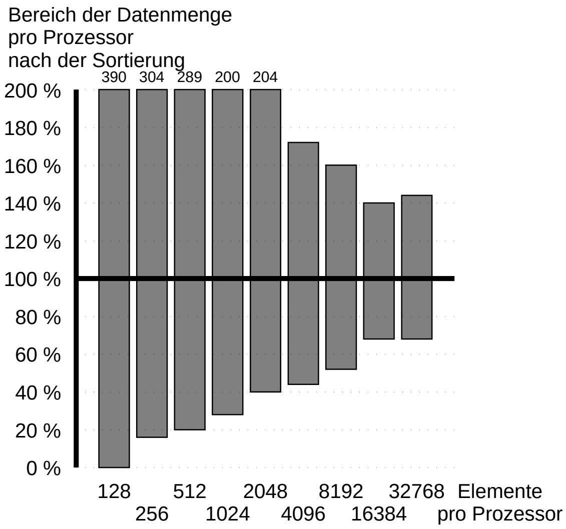
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1406	779	740	518	508	433	401	356	386
Gesamtzeit [s]	0.18	0.40	0.38	0.53	1.04	1.78	3.30	5.85	12.66
Zeit für Kommunikation [ms]	125	140	180	263	433	650	1069	1886	3453
Zeit für lokale Sortierung [ms]	54	89	196	269	609	1120	2282	4009	9278
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	40	99	297	800	1763	4412	10879	21962
Maximale Anzahl Elemente	500	779	1482	2037	4180	7000	13357	21845	46942
Anzahl Datenpakete	2048	2048	2048	2048	3255	5453	9535	17942	34566
Gesamtweglänge aller Datenpaket	7680	7680	7680	7680	12141	20583	35933	67606	129575
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

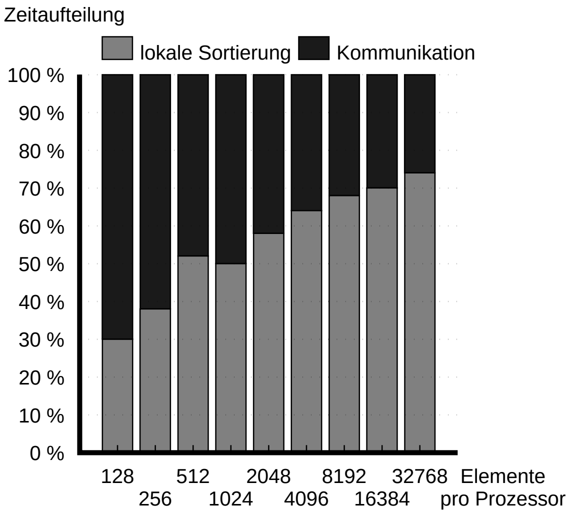
Datenblatt 32

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





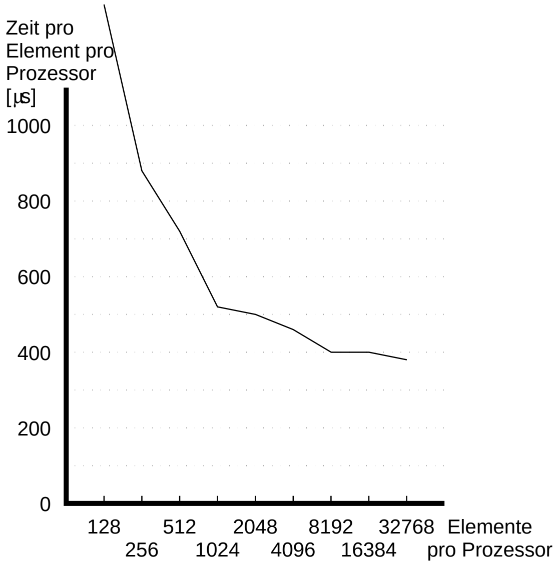


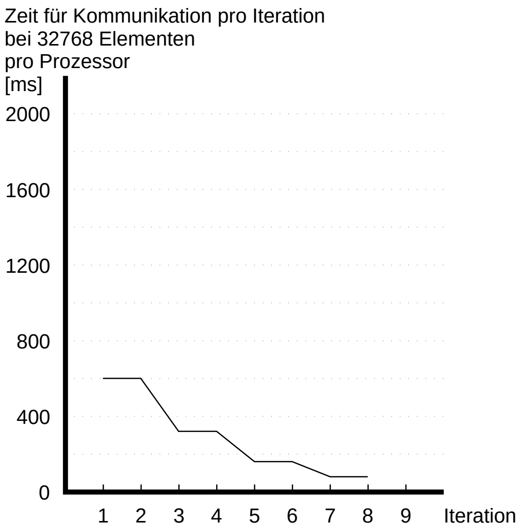


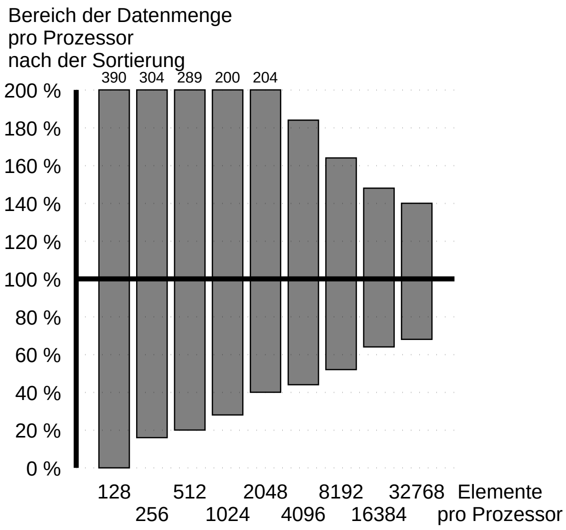
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1351	870	726	511	511	433	396	377	388
Gesamtzeit [s]	0.17	0.22	0.37	0.52	1.05	1.78	3.25	6.19	12.74
Zeit für Kommunikation [ms]	120	140	180	259	439	650	1063	1850	3413
Zeit für lokale Sortierung [ms]	53	84	198	269	608	1126	2196	4343	9386
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	40	99	297	800	1763	4413	10879	21962
Maximale Anzahl Elemente	500	779	1482	2037	4180	6999	13142	23088	46942
Anzahl Datenpakete	2048	2048	2048	2048	3254	5451	9537	17939	34568
Gesamtweglänge aller Datenpaket	8800	8800	8800	8800	14346	23937	41525	77527	148246
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

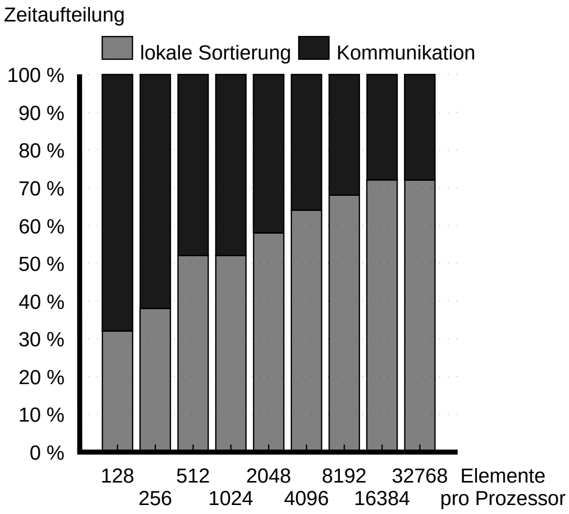
Datenblatt 33

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	modifizierte Hilbertnumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





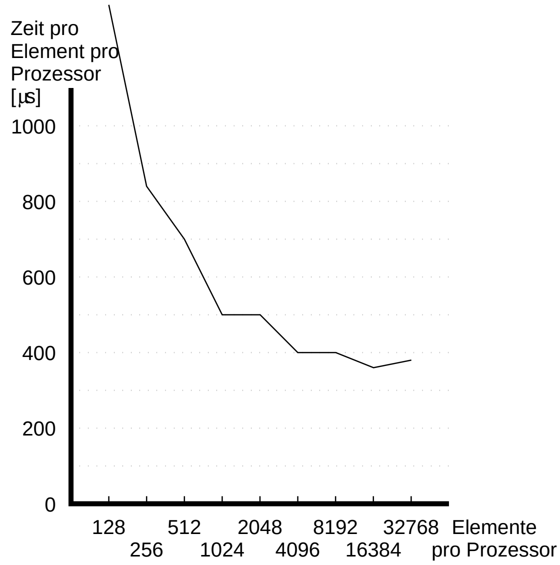


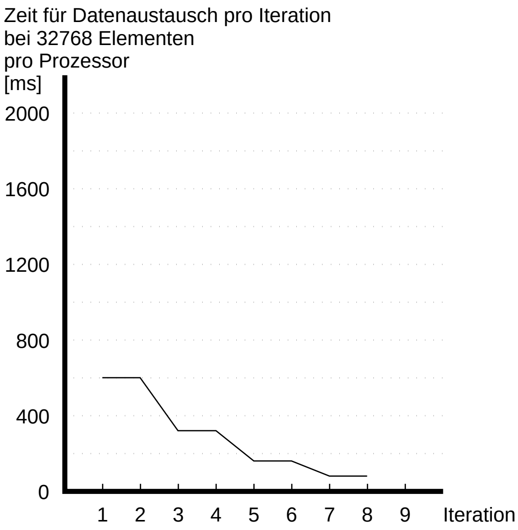


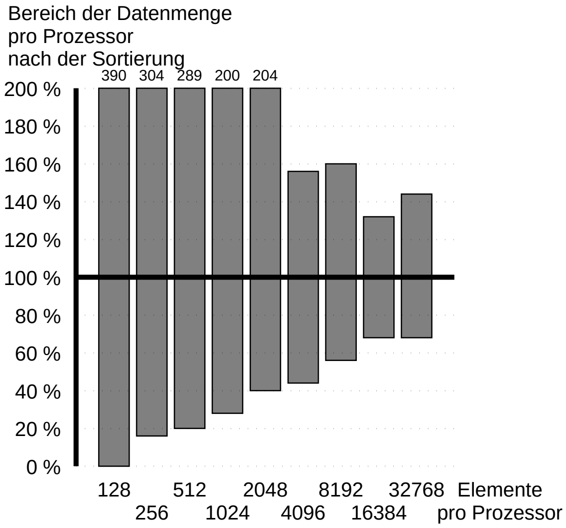
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1328	874	722	514	507	462	403	390	378
Gesamtzeit [s]	0.17	0.22	0.37	0.53	1.04	1.90	3.30	6.40	12.41
Zeit für Kommunikation [ms]	120	140	178	258	439	645	1050	1863	3404
Zeit für lokale Sortierung [ms]	57	86	199	269	606	1244	2287	4537	9019
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	40	99	297	800	1763	4285	10546	21962
Maximale Anzahl Elemente	500	779	1482	2037	4180	7559	13357	24179	45651
Anzahl Datenpakete	2048	2048	2048	2048	3255	5450	9535	17930	34562
Gesamtweglänge aller Datenpaket	8128	8128	8128	8128	13211	22074	38355	71574	137052
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

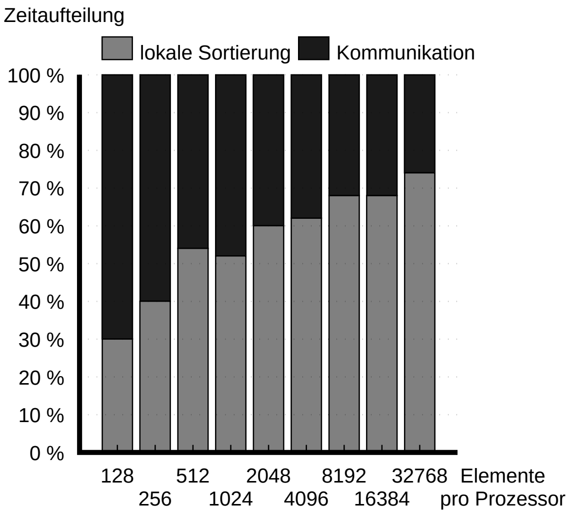
Datenblatt 34

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	H-Numerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





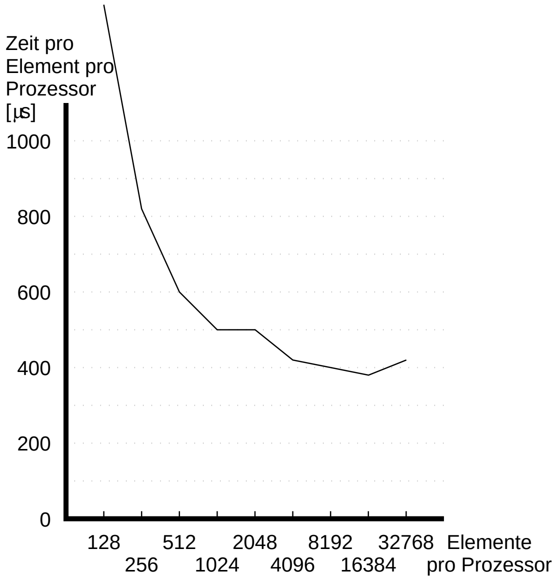


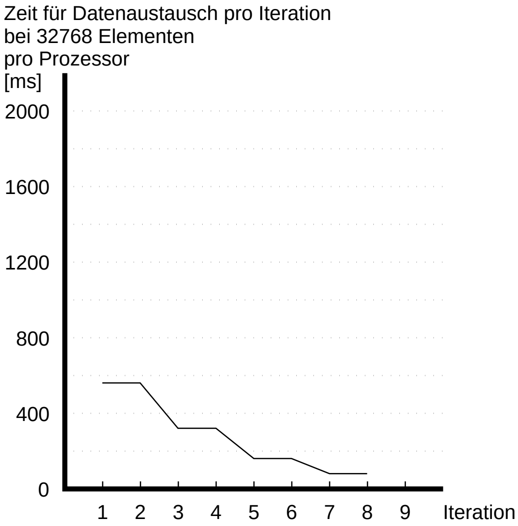


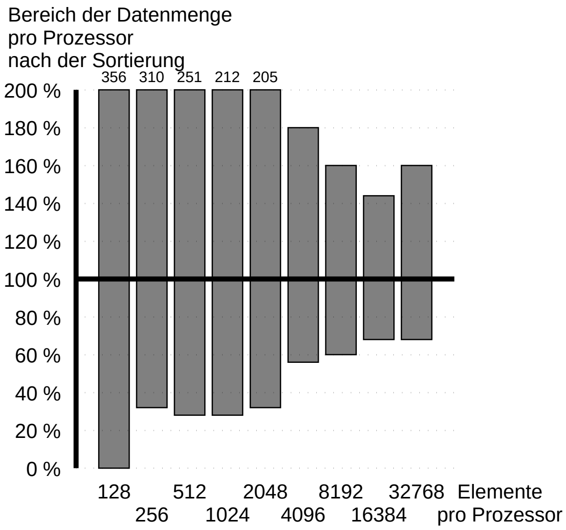
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1320	831	703	500	502	400	393	354	390
Gesamtzeit [s]	0.17	0.21	0.36	0.51	1.03	1.64	3.23	5.82	12.81
Zeit für Kommunikation [ms]	110	130	170	248	421	629	1037	1844	3396
Zeit für lokale Sortierung [ms]	49	89	197	268	608	1.02	2196	4006	9459
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	40	99	297	800	1763	4458	10879	21962
Maximale Anzahl Elemente	500	779	1482	2037	4180	6337	13142	21845	46942
Anzahl Datenpakete	2048	2048	2048	2048	3254	5455	9531	17947	34563
Gesamtweglänge aller Datenpaket	7680	7680	7680	7680	12466	20839	36279	67742	129469
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

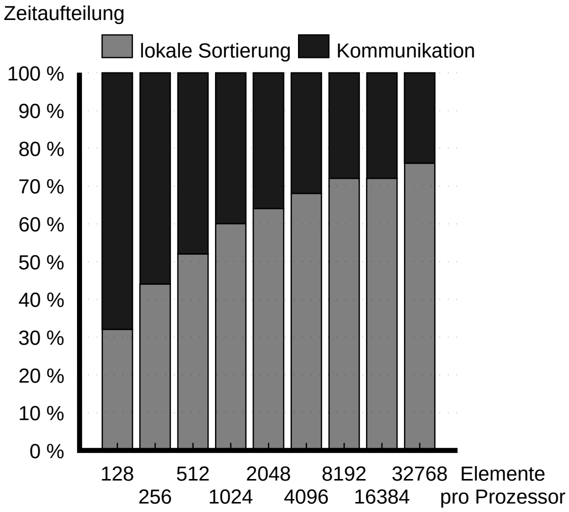
Datenblatt 35

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Z-Numerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





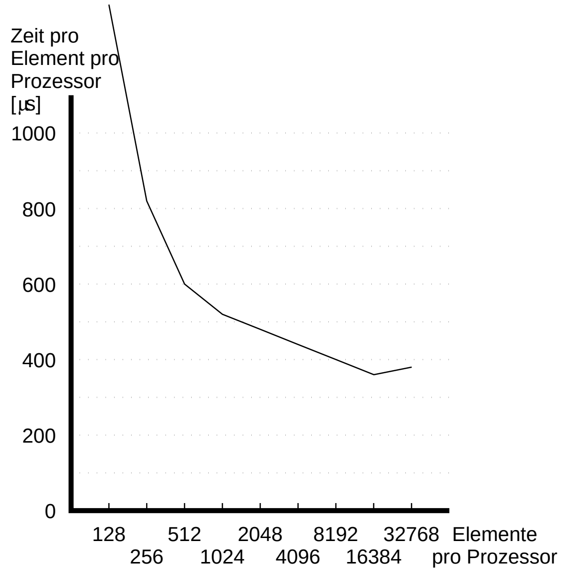


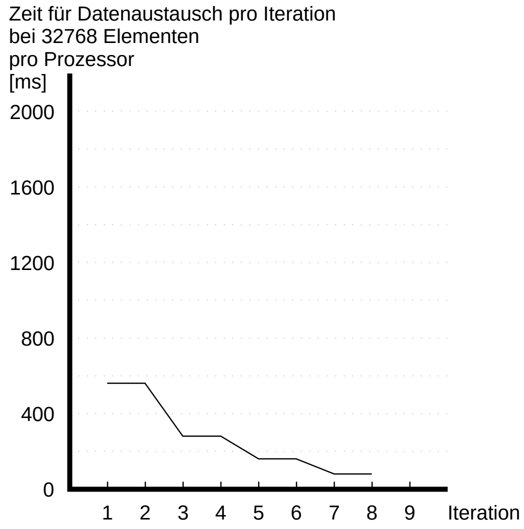


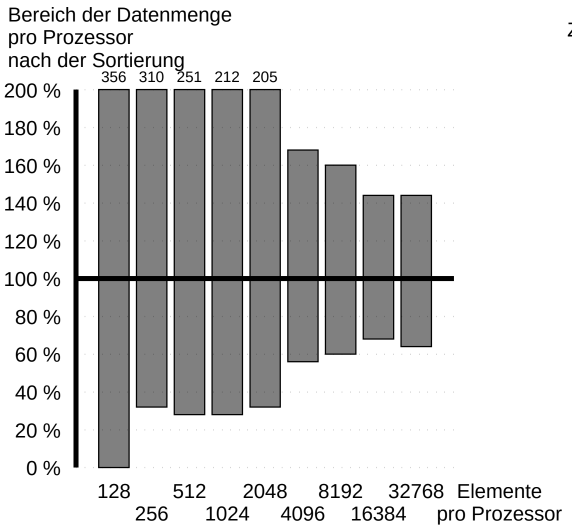
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1171	820	595	496	491	428	400	379	424
Gesamtzeit [s]	0.15	0.21	0.30	0.51	1.01	1.76	3.28	6.23	13.91
Zeit für Kommunikation [ms]	102	120	150	210	379	589	976	1789	3343
Zeit für lokale Sortierung [ms]	49	89	154	294	629	1160	2323	4476	10562
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	80	144	300	656	2341	4772	10945	22040
Maximale Anzahl Elemente	456	794	1286	2179	4215	7434	13183	23771	52519
Anzahl Datenpakete	2048	2048	2048	2049	3219	5372	9576	17936	34562
Gesamtweglänge aller Datenpaket	7936	7936	7936	7937	12676	21240	37522	69965	133795
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

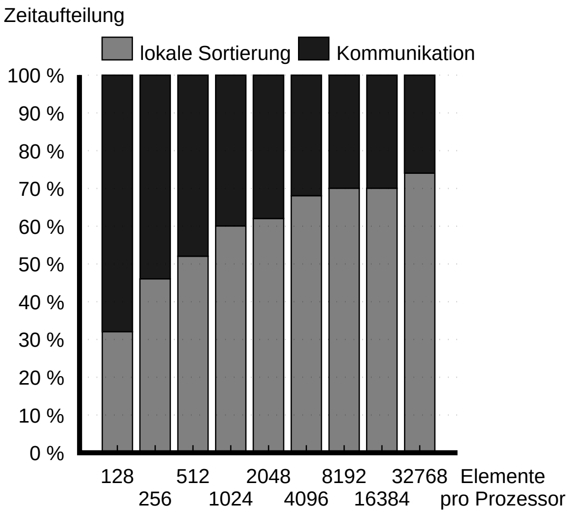
Datenblatt 36

Quicksort	mit lokaler Sortierung am Ende		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Shuffle-Numerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





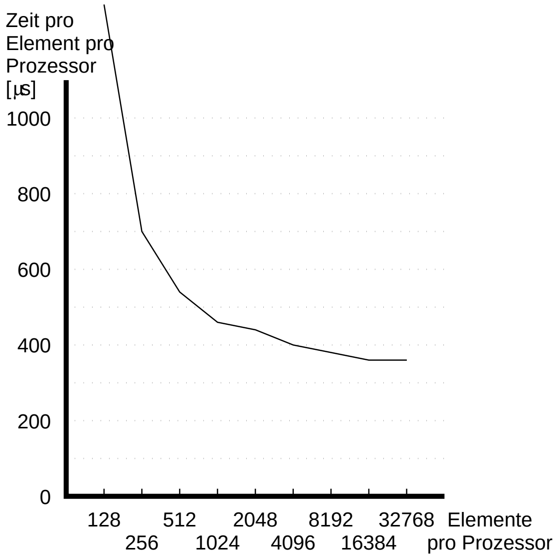


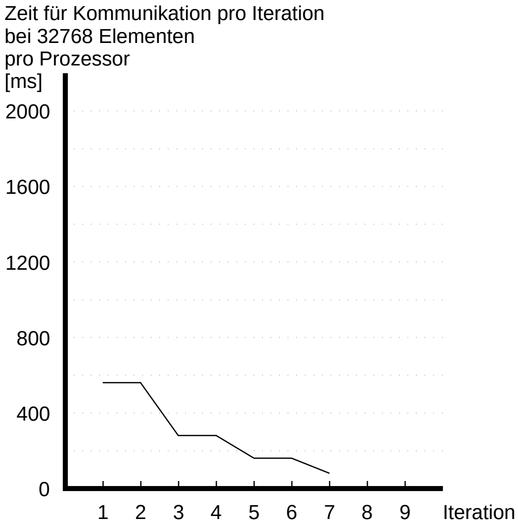


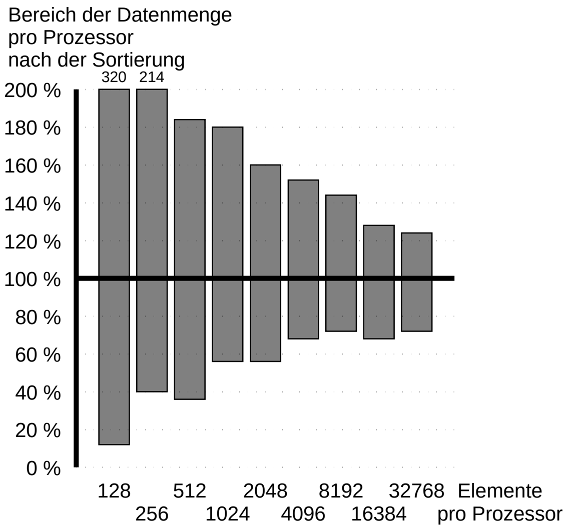
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1139	820	603	517	488	437	395	370	386
Gesamtzeit [s]	0.15	0.21	0.31	0.53	1.00	1.79	3.25	6.08	12.68
Zeit für Kommunikation [ms]	100	117	148	208	377	580	973	1784	3330
Zeit für lokale Sortierung [ms]	49	96	157	319	633	1201	2292	4354	9377
Zeit für Odd-Even-Sort [ms]	0	0	0	0	0	0	0	0	0
Minimale Anzahl Elemente	0	80	144	300	656	2341	4772	10984	21105
Maximale Anzahl Elemente	456	794	1286	2180	4215	6953	13195	23628	47139
Anzahl Datenpakete	2048	2048	2048	2049	3217	5372	9571	17930	34578
Gesamtweglänge aller Datenpaket	7680	7680	7680	7681	12269	20599	36332	67729	129507
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

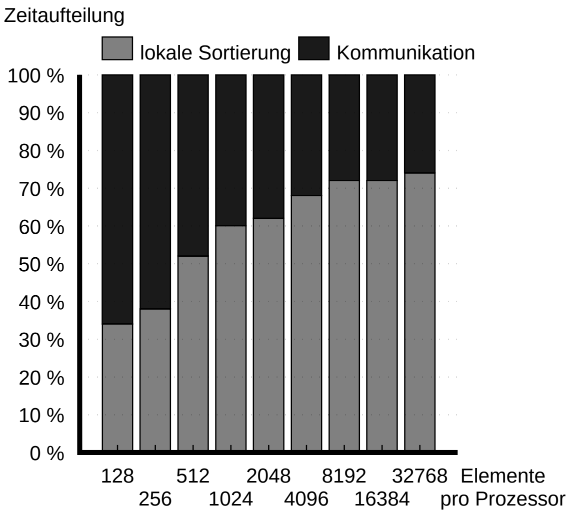
Datenblatt 37

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 2 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Shuffle-Numerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





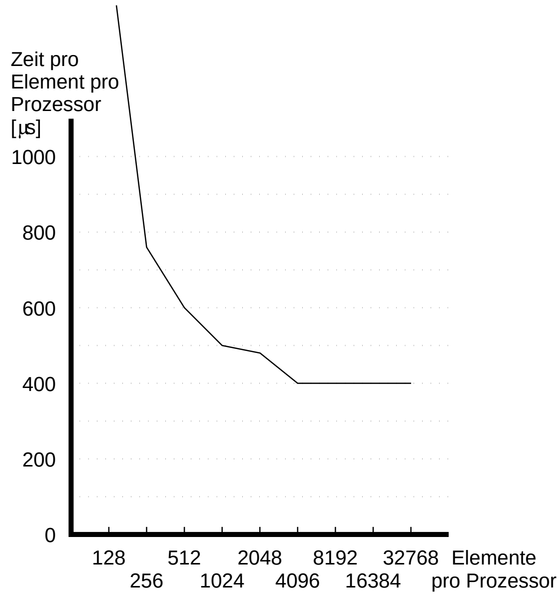


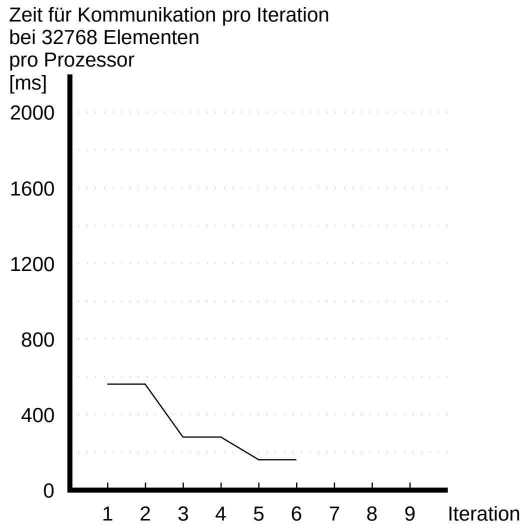


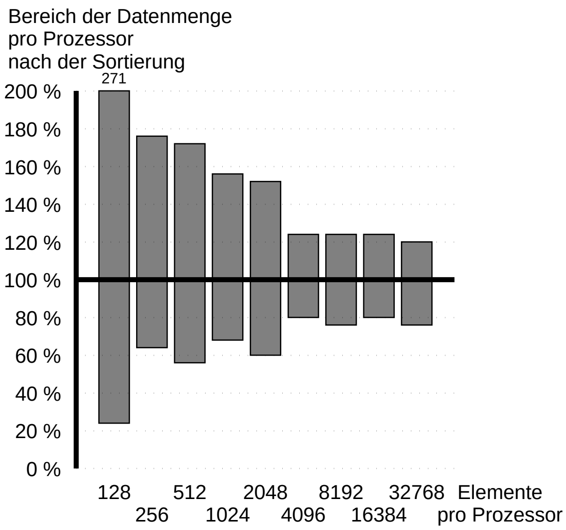
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1108	706	548	460	439	406	390	351	370
Gesamtzeit [s]	0.14	0.18	0.28	0.47	0.90	1.67	3.20	5.76	12.14
Zeit für Kommunikation [ms]	100	110	140	194	350	548	900	1654	3092
Zeit für lokale Sortierung [ms]	41	60	129	250	489	1019	2094	3756	8363
Zeit für Odd-Even-Sort [ms]	10	10	20	32	60	111	208	363	722
Minimale Anzahl Elemente	17	102	180	570	1133	2780	5875	11357	23794
Maximale Anzahl Elemente	410	550	952	1829	3246	6220	11919	20796	41102
Anzahl Datenpakete	2048	2048	2048	2186	3475	5890	10616	20031	38765
Gesamtweglänge aller Datenpaket	7680	7680	7680	7818	12527	21117	37376	69830	133696
Durchschn. Iterationstiefe	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00

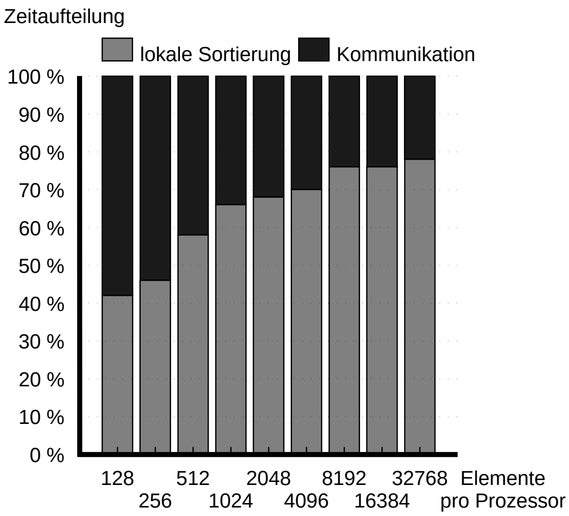
Datenblatt 38

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 4 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Z-Numerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





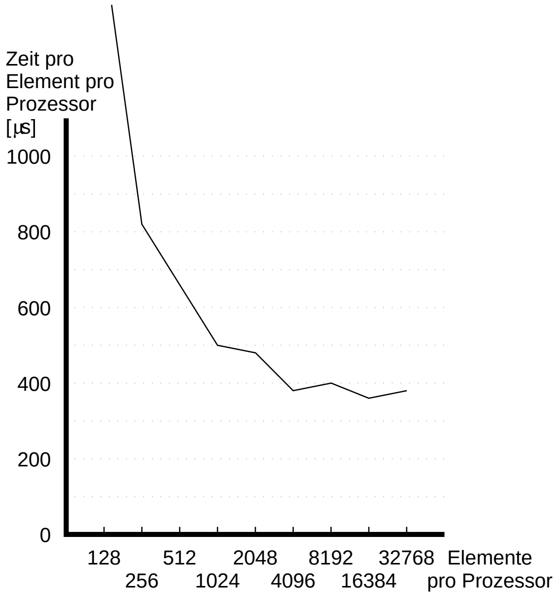


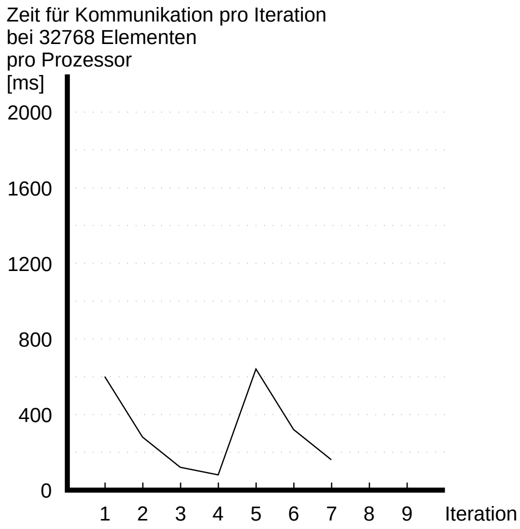


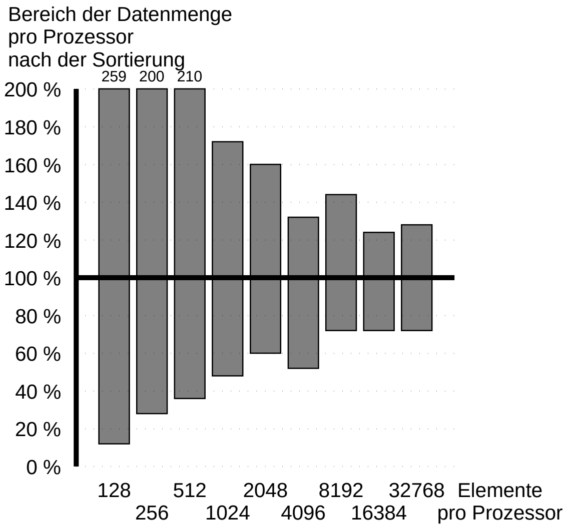
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1241	761	606	507	484	402	410	402	406
Gesamtzeit [s]	0.16	0.19	0.31	0.52	0.99	1.65	3.36	6.61	13.34
Zeit für Kommunikation [ms]	90	108	130	180	330	510	835	1530	2865
Zeit für lokale Sortierung [ms]	39	59	120	229	479	829	1860	3782	7932
Zeit für Odd-Even-Sort [ms]	28	34	63	109	204	338	669	1303	2580
Minimale Anzahl Elemente	33	168	280	686	1244	3270	6321	13465	25471
Maximale Anzahl Elemente	347	455	876	1610	3076	5130	10261	20042	39625
Anzahl Datenpakete	2304	2304	2304	2696	4400	7598	13931	26415	51460
Gesamtweglänge aller Datenpaket	7936	7936	7936	8328	13472	22825	40701	76217	146367
Durchschn. Iterationstiefe	6.00	6.00	6.00	6.00	6.00	6.00	6.00	6.00	6.00

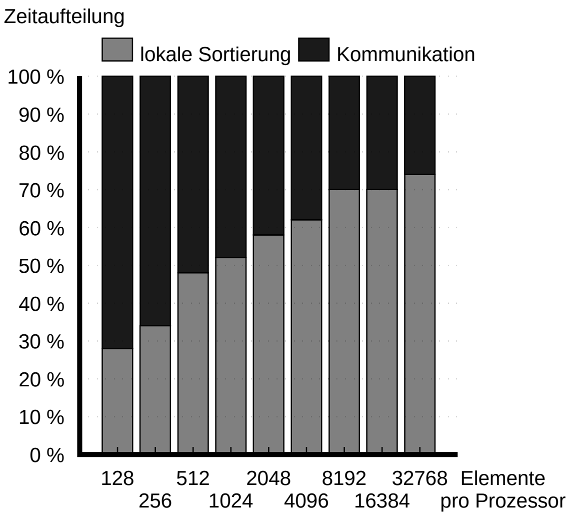
Datenblatt 39

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 2 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





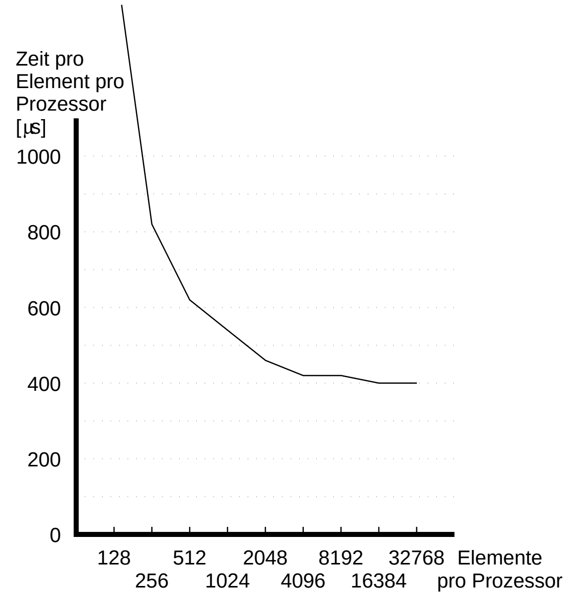


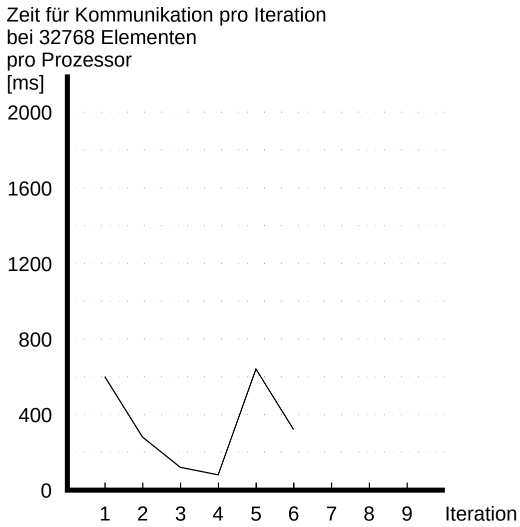


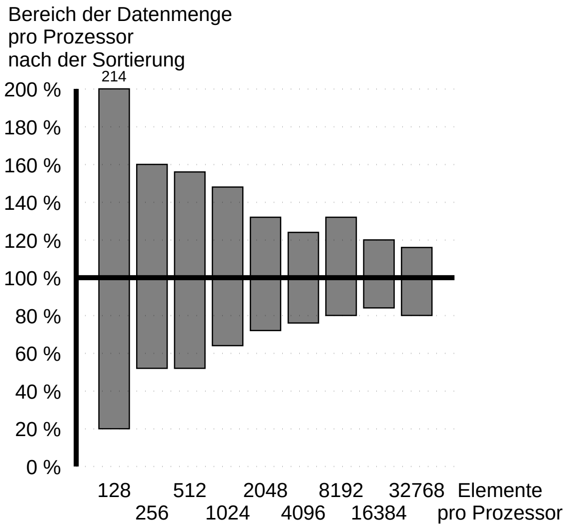
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1249	820	660	507	471	390	404	352	390
Gesamtzeit [s]	0.16	0.21	0.34	0.52	0.97	1.60	3.31	5.78	12.78
Zeit für Kommunikation [ms]	120	140	177	250	410	610	996	1761	3220
Zeit für lokale Sortierung [ms]	36	61	144	240	500	899	2118	3733	8856
Zeit für Odd-Even-Sort [ms]	10	10	21	35	60	99	207	352	737
Minimale Anzahl Elemente	17	73	181	479	1200	2161	5988	11849	23489
Maximale Anzahl Elemente	332	514	1079	1751	3312	5475	11875	20000	42159
Anzahl Datenpakete	2048	2048	2049	2175	3526	5981	10587	20040	38760
Gesamtweglänge aller Datenpaket	7680	7680	7681	7807	12412	21111	36983	69704	133768
Durchschn. Iterationstiefe	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00

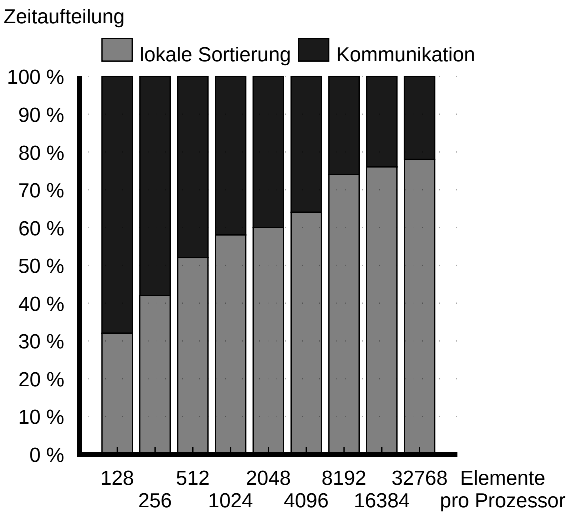
Datenblatt 40

Quicksort	mit lokaler Sortierung am Ende und Odd-Even-Sort für 4 Proz.		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der $\sqrt{N}$-Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





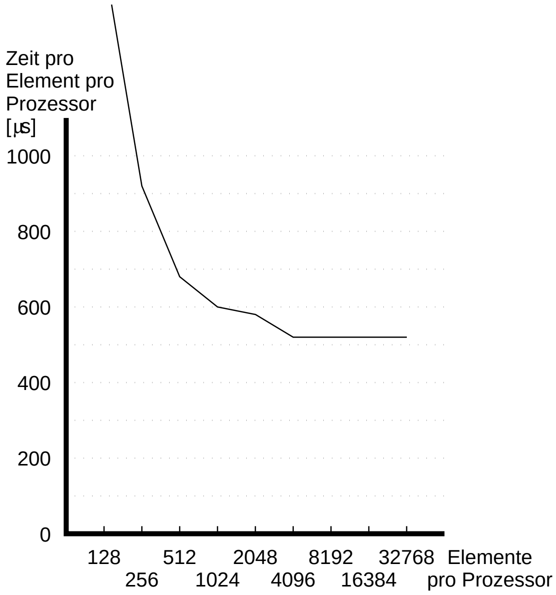


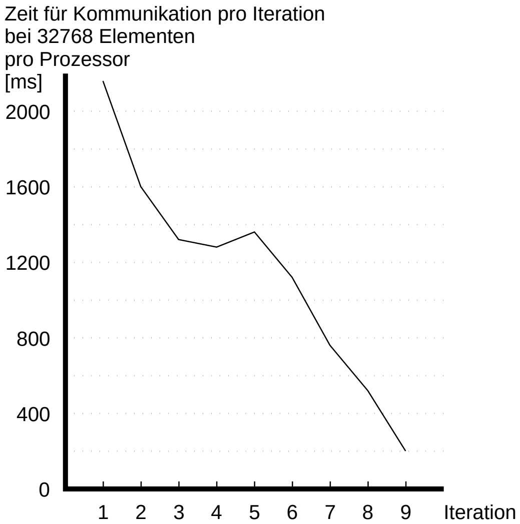


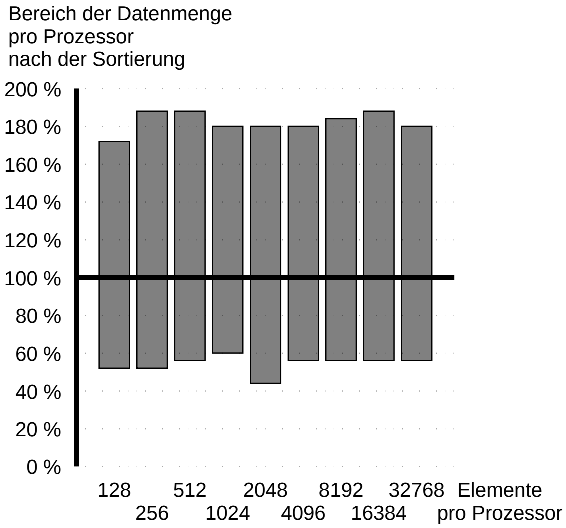
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1249	820	628	531	460	419	430	396	402
Gesamtzeit [s]	0.16	0.21	0.32	0.54	0.94	1.72	3.53	6.51	13.21
Zeit für Kommunikation [ms]	111	130	160	230	381	560	900	1610	2919
Zeit für lokale Sortierung [ms]	30	58	107	210	400	839	1931	3647	7901
Zeit für Odd-Even-Sort [ms]	24	33	60	99	177	326	700	1263	2493
Minimale Anzahl Elemente	26	129	257	675	1458	3038	6570	13710	26183
Maximale Anzahl Elemente	274	409	792	1527	2683	5031	10857	19421	38340
Anzahl Datenpakete	2304	2304	2304	2703	4453	7687	13856	26432	51475
Gesamtweglänge aller Datenpaket	7680	7680	7680	8079	12963	22155	39072	73872	142159
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

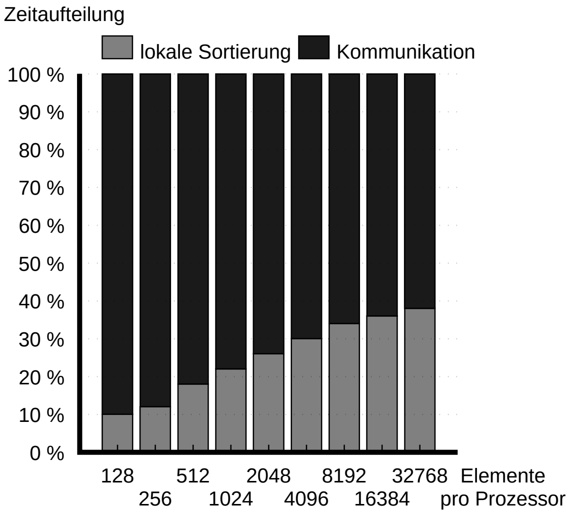
Datenblatt 41

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





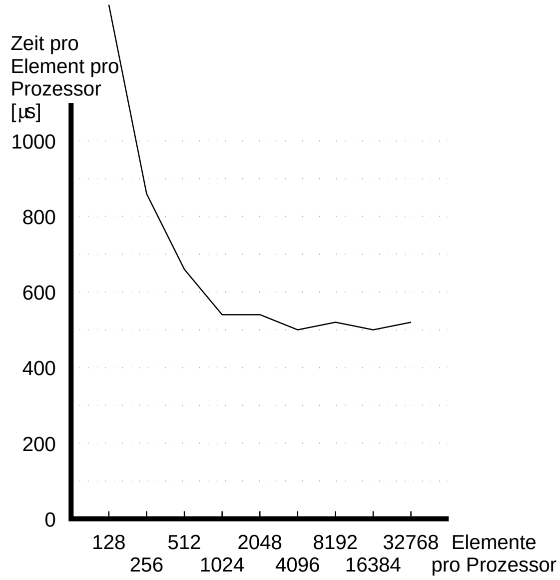


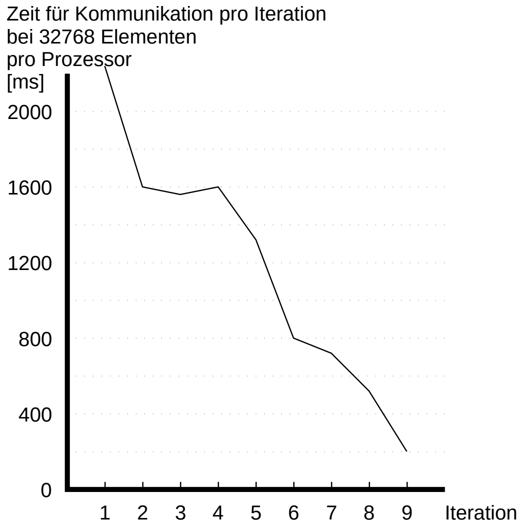


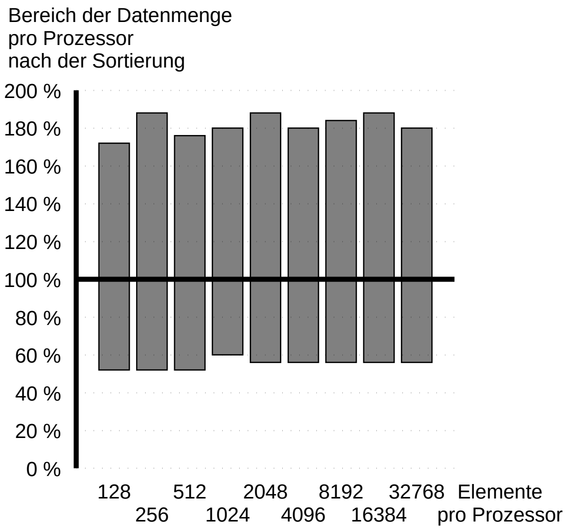
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1406	913	671	608	585	515	522	515	525
Gesamtzeit [s]	0.18	0.23	0.34	0.62	1.20	2.11	4.28	8.45	17.22
Zeit für Kommunikation [ms]	171	220	300	506	940	1525	3005	5669	11237
Zeit für lokale Sortierung [ms]	19	30	69	140	319	670	1524	3182	6905
Minimale Anzahl Elemente	66	135	284	607	938	2319	4612	9216	18345
Maximale Anzahl Elemente	218	485	963	1845	3696	7433	14987	31081	58893
Anzahl Datenpakete	5617	5639	5637	5642	7340	11493	19777	36510	69962
Gesamtweglänge aller Datenpaket	26799	26039	24244	25567	31968	47915	83747	149185	284578
Durchschn. Iterationstiefe	8.01	8.03	8.02	8.02	8.02	8.01	8.02	8.02	8.03

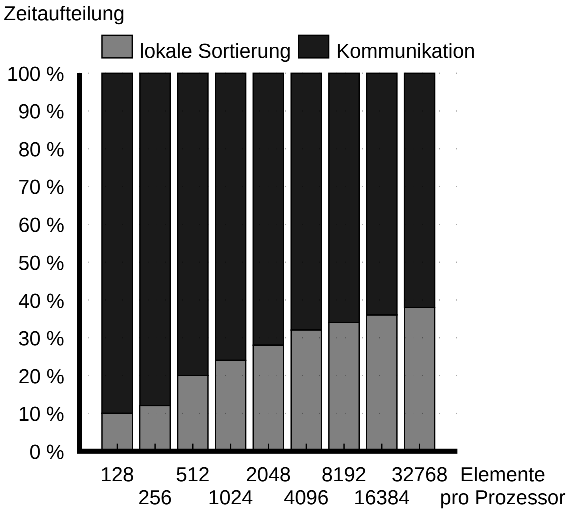
Datenblatt 42

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





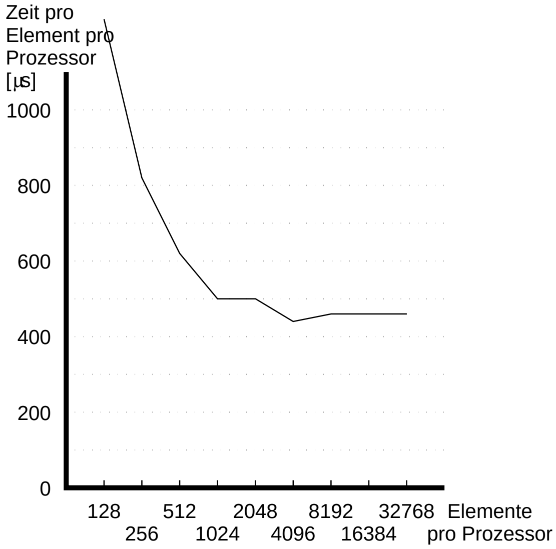


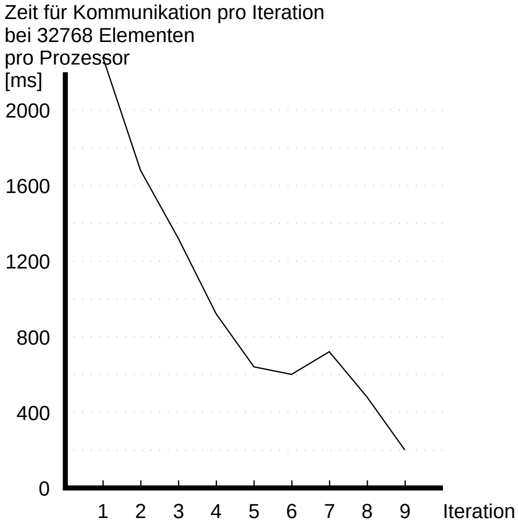


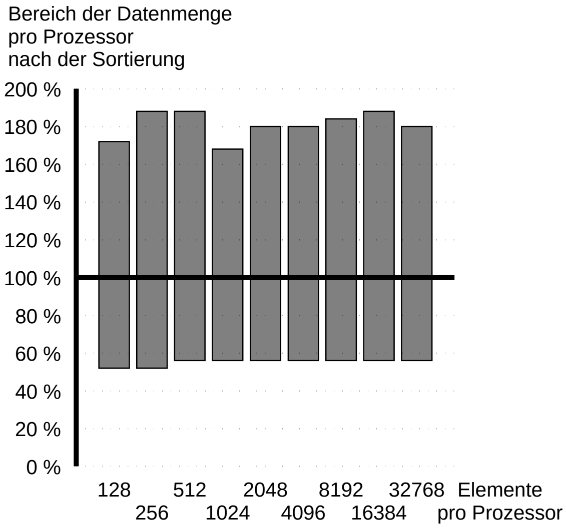
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1351	862	664	537	541	500	524	512	522
Gesamtzeit [s]	0.17	2.20	3.40	0.55	1.11	2.05	4.30	8.40	17.15
Zeit für Kommunikation [ms]	170	209	290	439	848	1465	3023	5614	11209
Zeit für lokale Sortierung [ms]	19	30	69	142	319	670	1526	3181	6905
Minimale Anzahl Elemente	66	135	262	609	1153	2319	4612	9225	18353
Maximale Anzahl Elemente	218	485	910	1845	3849	7433	14988	31093	58893
Anzahl Datenpakete	5617	5639	5628	5640	7342	11494	19779	36514	69964
Gesamtweglänge aller Datenpaket	26977	27227	26957	27213	34255	53212	90824	166741	318452
Durchschn. Iterationstiefe	8.01	8.03	8.01	8.02	8.03	8.01	8.02	8.02	8.03

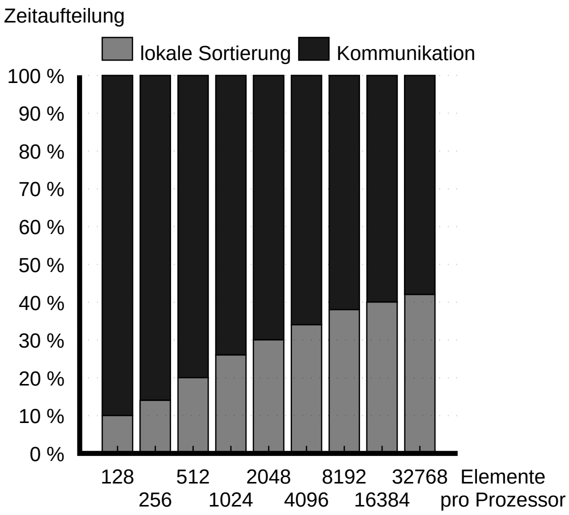
Datenblatt 43

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Nein	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





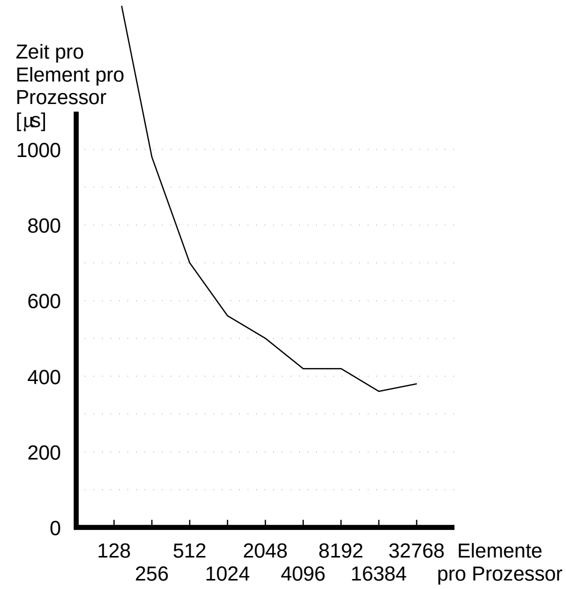


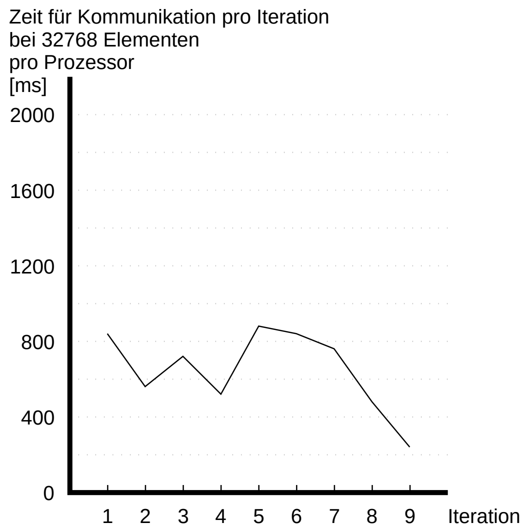


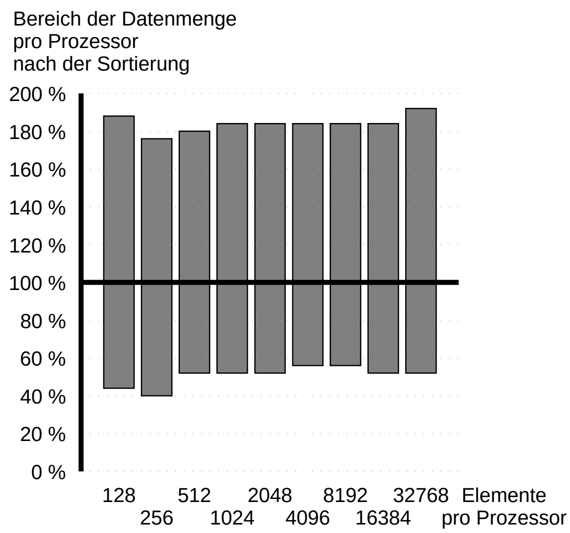
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1249	820	624	505	494	449	451	455	468
Gesamtzeit [s]	0.16	0.21	0.32	0.52	1.01	1.84	3.70	7.46	15.36
Zeit für Kommunikation [ms]	160	190	274	408	752	1253	2430	4684	9421
Zeit für lokale Sortierung [ms]	19	30	69	141	319	670	1523	3180	6904
Minimale Anzahl Elemente	66	135	284	600	1166	2319	4612	9208	18363
Maximale Anzahl Elemente	218	485	963	1724	3722	7433	14914	31104	58893
Anzahl Datenpakete	5617	5639	5633	5636	7341	11494	19781	36514	69954
Gesamtweglänge aller Datenpaket	24423	24723	24428	24668	31324	48124	81763	151296	286555
Durchschn. Iterationstiefe	8.01	8.03	8.01	8.01	8.02	8.01	8.02	8.02	8.02

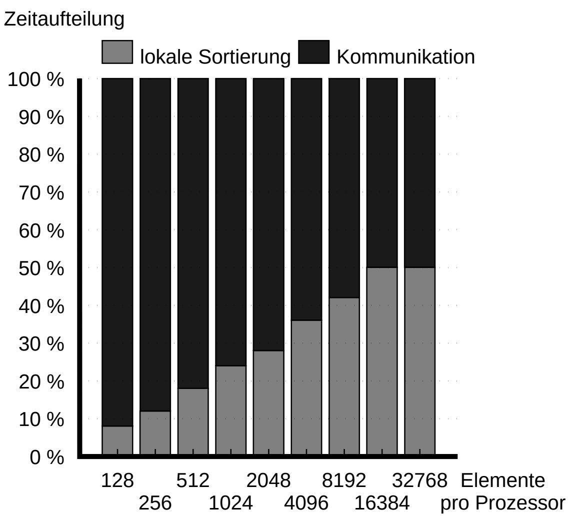
Datenblatt 44

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





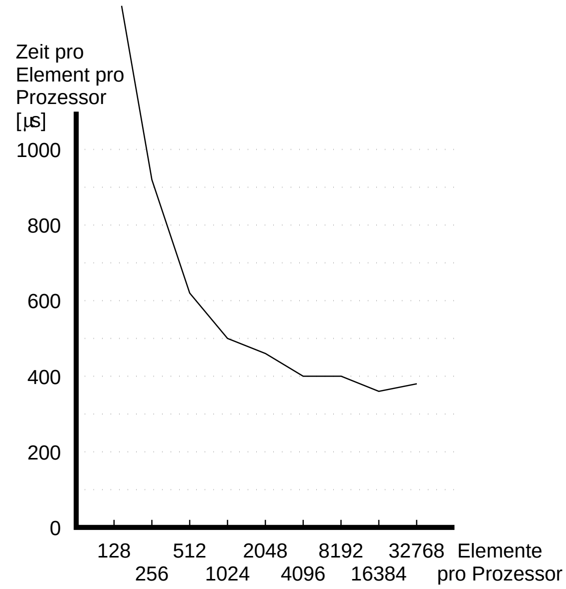


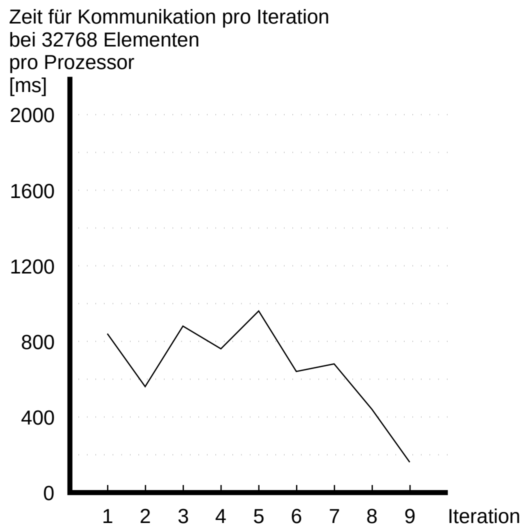


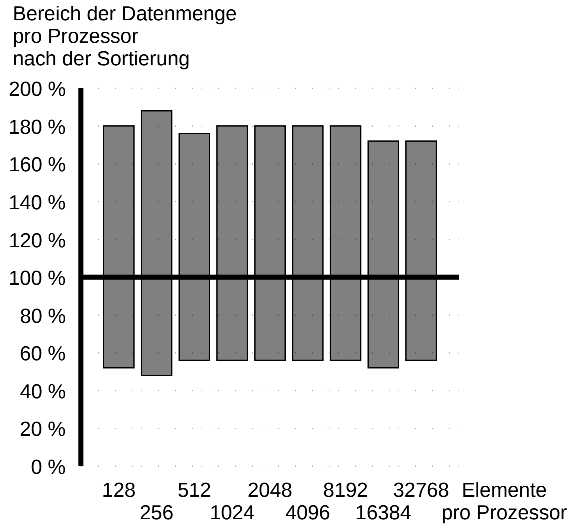
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1585	979	706	555	506	429	412	362	390
Gesamtzeit [s]	0.20	0.25	0.36	0.57	1.04	1.76	3.38	5.94	12.80
Zeit für Kommunikation [ms]	200	230	310	450	780	1175	2105	3162	6819
Zeit für lokale Sortierung [ms]	19	30	69	141	316	670	1523	3183	6904
Minimale Anzahl Elemente	56	101	275	546	1061	2262	4466	8792	17445
Maximale Anzahl Elemente	241	446	917	1881	3747	7546	15203	30064	62857
Anzahl Datenpakete	3569	3582	3584	3589	4040	6150	10310	18346	35002
Gesamtweglänge aller Datenpaket	19855	17382	17960	18115	19941	28397	48484	74728	159889
Durchschn. Iterationstiefe	8.03	8.02	8.02	8.03	8.03	8.02	8.03	8.02	8.02

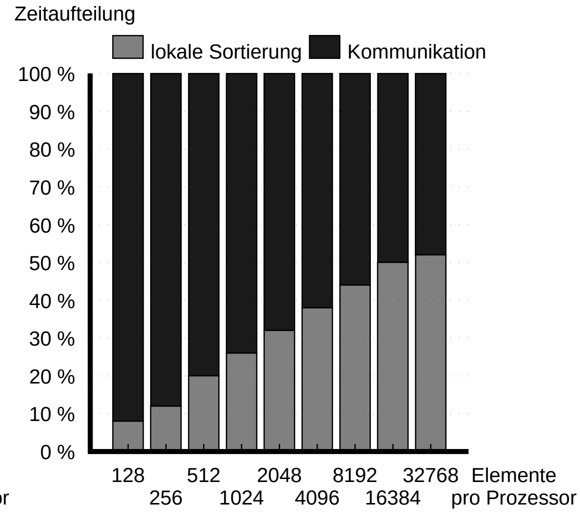
Datenblatt 45

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





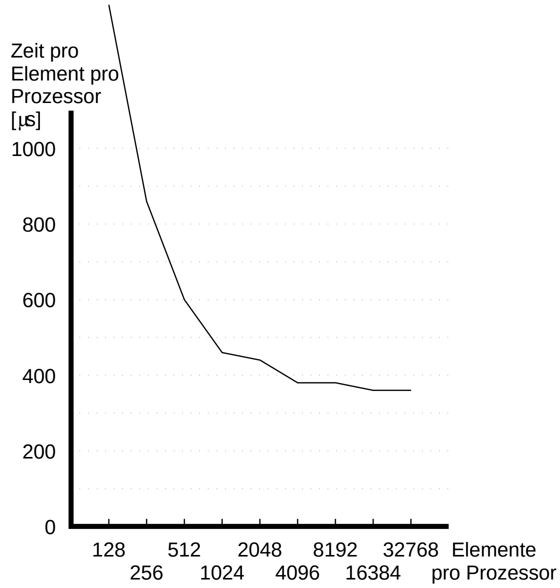


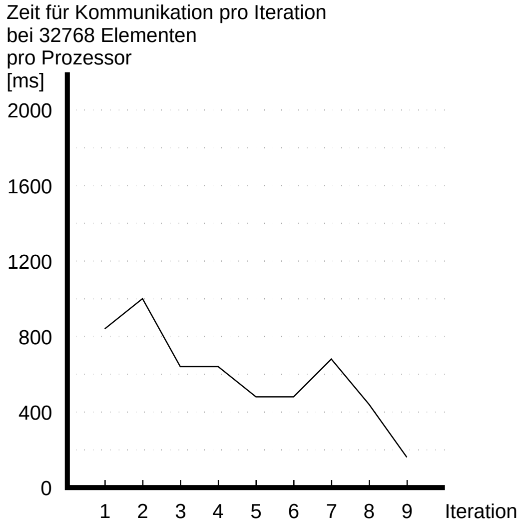


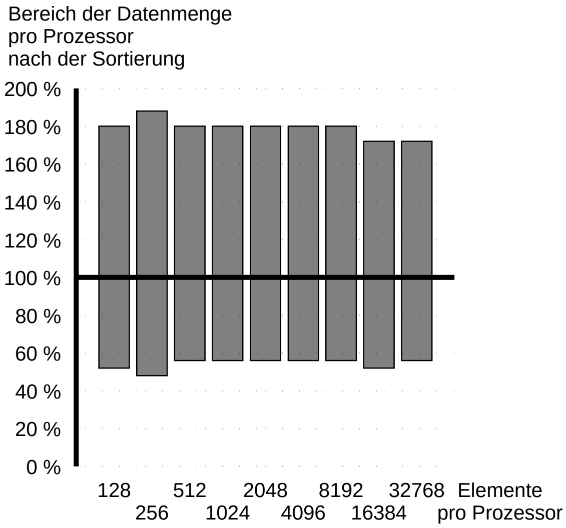
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1507	929	624	507	465	405	394	360	379
Gesamtzeit [s]	0.19	0.24	0.32	0.52	0.95	1.66	3.23	5.90	12.43
Zeit für Kommunikation [ms]	190	219	270	403	692	1073	1959	3122	6489
Zeit für lokale Sortierung [ms]	18	30	69	140	319	670	1523	3181	6903
Minimale Anzahl Elemente	67	122	289	579	1133	2297	4586	8752	17844
Maximale Anzahl Elemente	231	481	893	1856	3690	7432	14614	28355	55954
Anzahl Datenpakete	3580	3589	3576	3588	3998	6101	10217	18359	34990
Gesamtweglänge aller Datenpaket	18791	17577	15638	16467	17716	25882	42998	72260	142993
Durchschn. Iterationstiefe	8.03	8.03	8.01	8.02	8.02	8.02	8.02	8.01	8.02

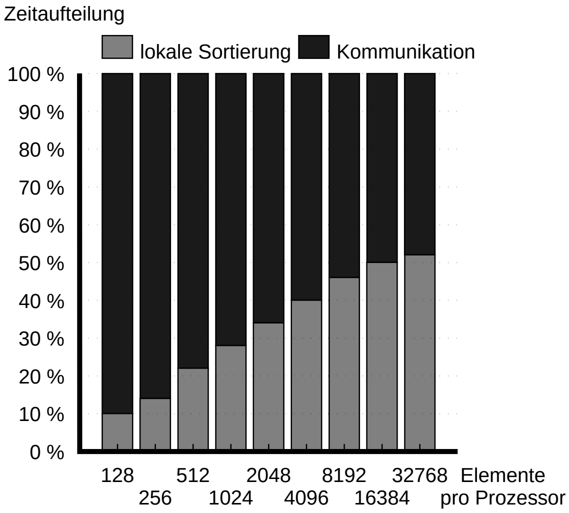
Datenblatt 46

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





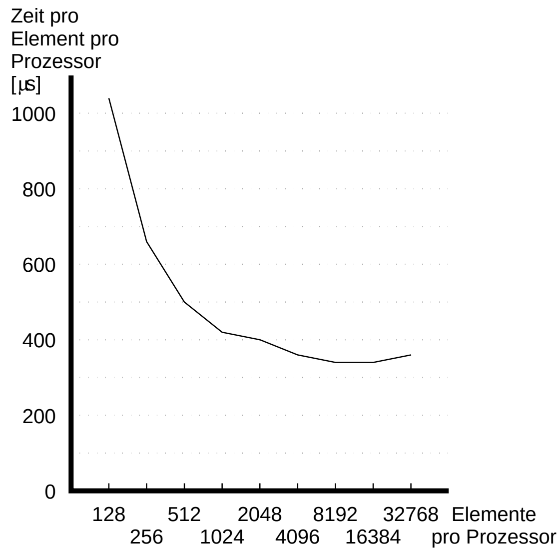


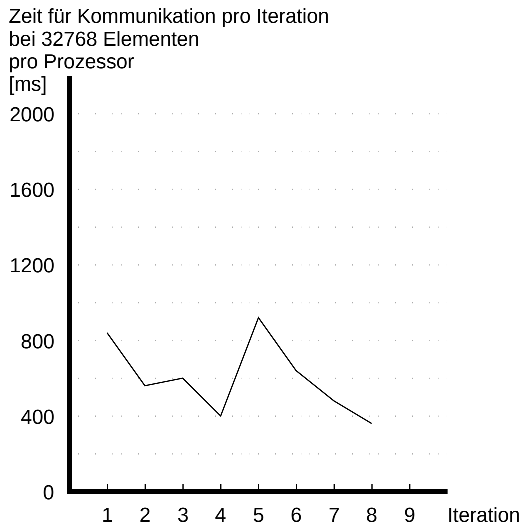


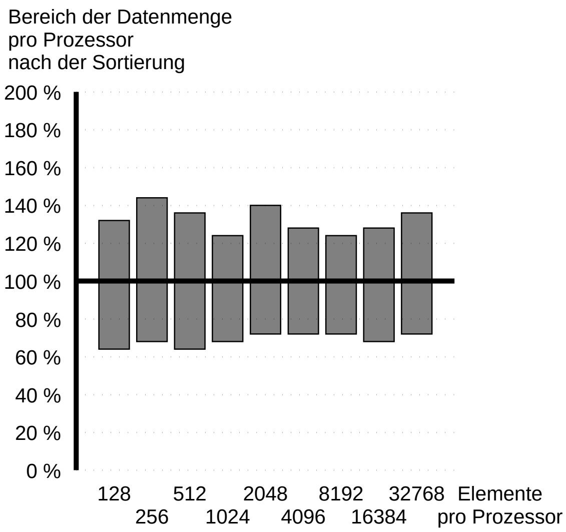
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1390	859	603	468	433	381	372	357	367
Gesamtzeit [s]	0.18	0.22	0.31	0.48	0.89	1.57	3.05	5.86	12.07
Zeit für Kommunikation [ms]	178	200	260	369	629	979	1779	3076	6129
Zeit für lokale Sortierung [ms]	19	30	70	143	319	670	1524	3181	6909
Minimale Anzahl Elemente	67	122	279	563	1171	2297	4587	8752	17844
Maximale Anzahl Elemente	231	481	922	1840	3649	7425	14614	28355	55954
Anzahl Datenpakete	3580	3589	3581	3591	4018	6100	10218	18356	34991
Gesamtweglänge aller Datenpaket	18717	18267	17772	17950	19211	28556	47291	81575	160750
Durchschn. Iterationstiefe	8.03	8.03	8.02	8.03	8.02	8.02	8.02	8.01	8.02

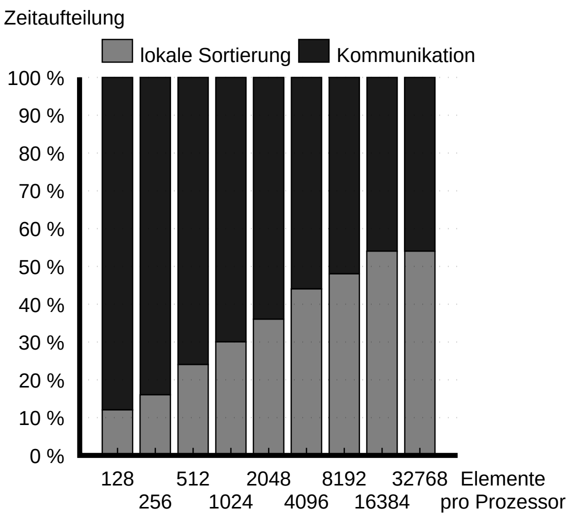
Datenblatt 47

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





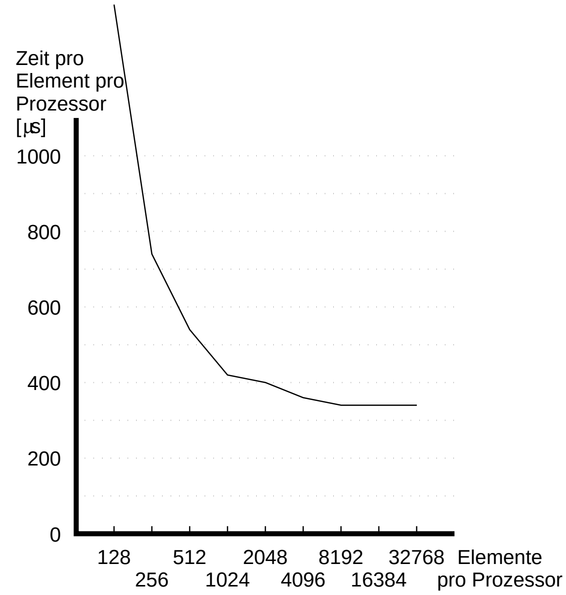


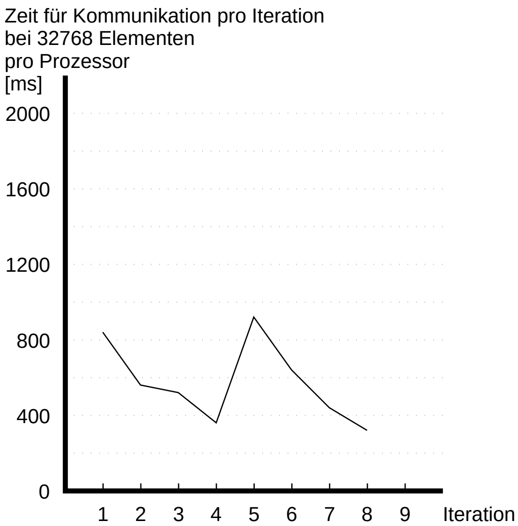


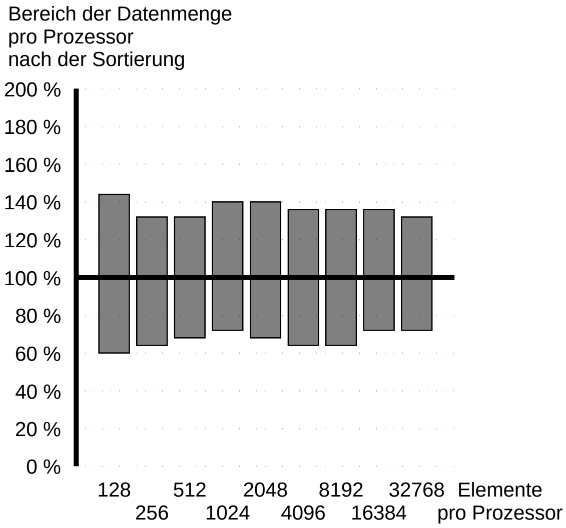
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1038	664	507	426	395	356	349	341	355
Gesamtzeit [s]	0.13	0.17	0.26	0.44	0.81	1.46	2.86	5.59	11.66
Zeit für Kommunikation [ms]	130	160	219	320	554	879	1589	2810	5678
Zeit für lokale Sortierung [ms]	18	30	69	143	319	670	1523	3183	6907
Minimale Anzahl Elemente	80	169	321	703	1462	2886	6011	11303	23933
Maximale Anzahl Elemente	169	366	706	1290	2881	5246	10225	20950	44019
Anzahl Datenpakete	3539	3562	3560	3580	4114	6194	10373	18412	35225
Gesamtweglänge aller Datenpaket	16060	15901	15758	16403	17862	26203	42352	73119	141266
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

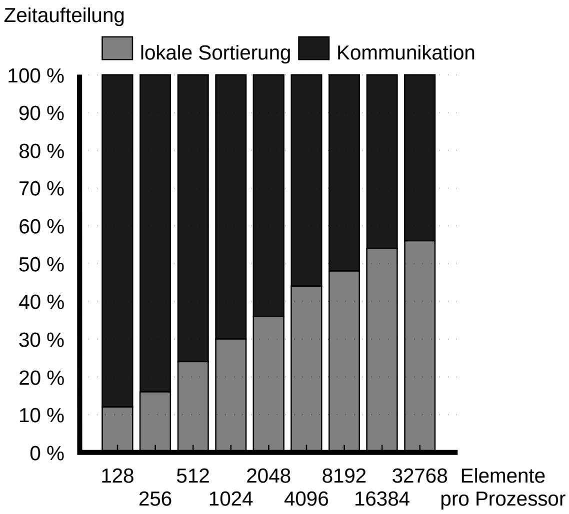
Datenblatt 48

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





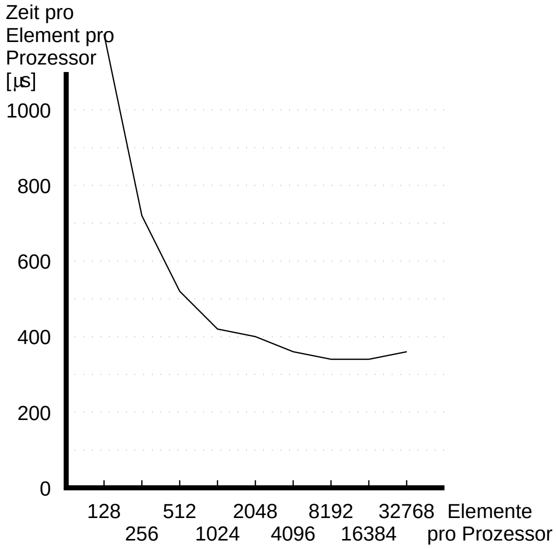


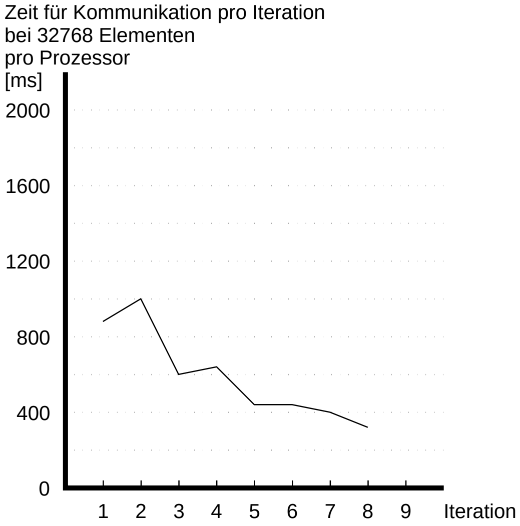


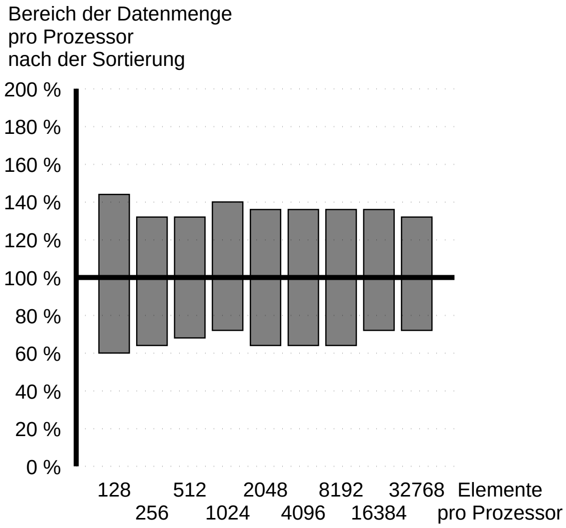
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1202	742	540	439	401	356	345	338	349
Gesamtzeit [s]	0.15	0.19	0.28	0.45	0.82	1.46	2.83	5.56	11.46
Zeit für Kommunikation [ms]	150	170	229	336	560	879	1558	2778	5526
Zeit für lokale Sortierung [ms]	18	30	69	142	319	670	1526	3183	6901
Minimale Anzahl Elemente	75	168	355	732	1351	2548	5217	11619	23319
Maximale Anzahl Elemente	184	336	669	1449	2836	5587	11160	22554	42954
Anzahl Datenpakete	3559	3559	3570	3575	4082	6117	10193	18338	34880
Gesamtweglänge aller Datenpaket	15699	15580	15386	15580	17135	24960	40445	71766	136219
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

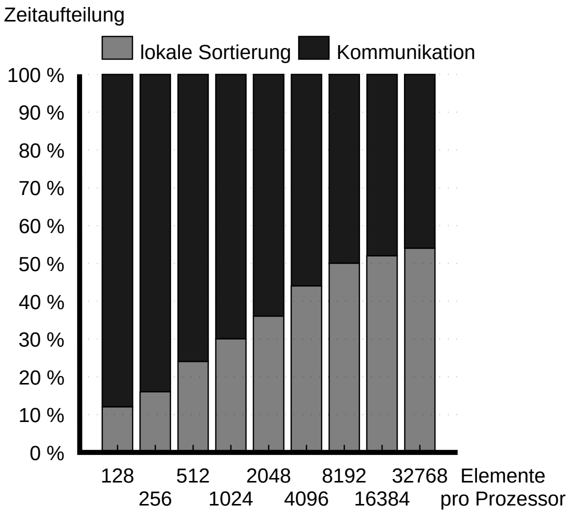
Datenblatt 49

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





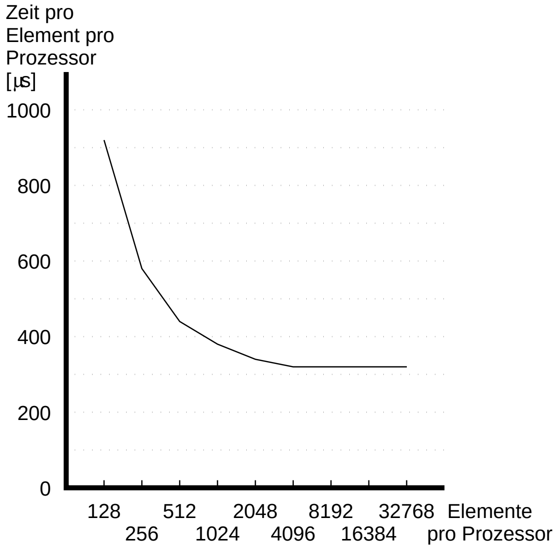


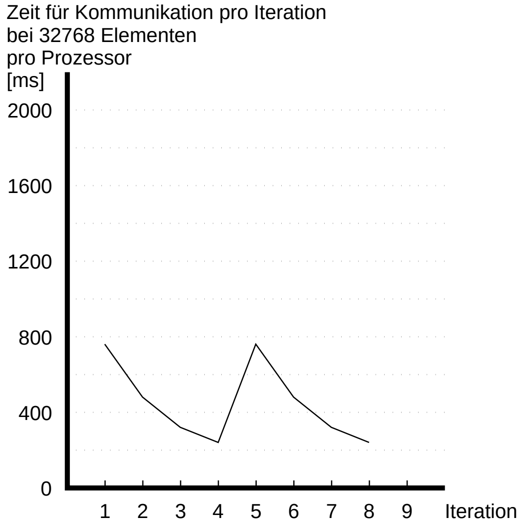


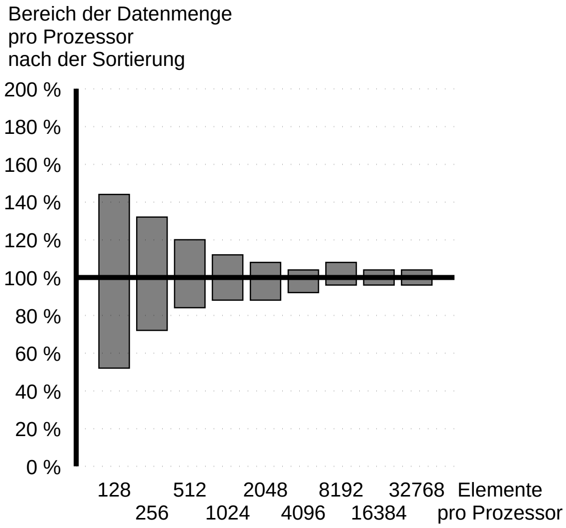
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1093	726	525	429	396	357	346	342	354
Gesamtzeit [s]	0.14	0.19	0.27	0.44	0.82	1.46	2.84	5.62	11.62
Zeit für Kommunikation [ms]	140	164	220	330	558	879	1569	2830	5685
Zeit für lokale Sortierung [ms]	19	30	69	141	317	670	1525	3182	6904
Minimale Anzahl Elemente	75	168	351	757	1331	2548	5217	11619	23319
Maximale Anzahl Elemente	184	336	675	1449	2826	5587	11160	22554	42954
Anzahl Datenpakete	3559	3559	3575	3575	4088	6117	10192	18337	34877
Gesamtweglänge aller Datenpaket	17291	17376	17300	17406	18736	27730	45821	81227	154633
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

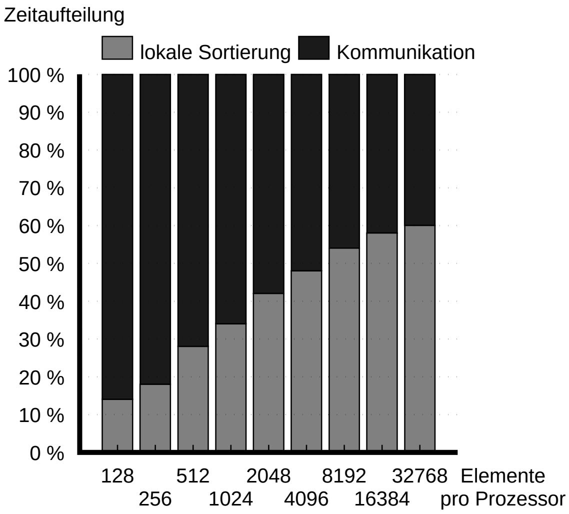
Datenblatt 50

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





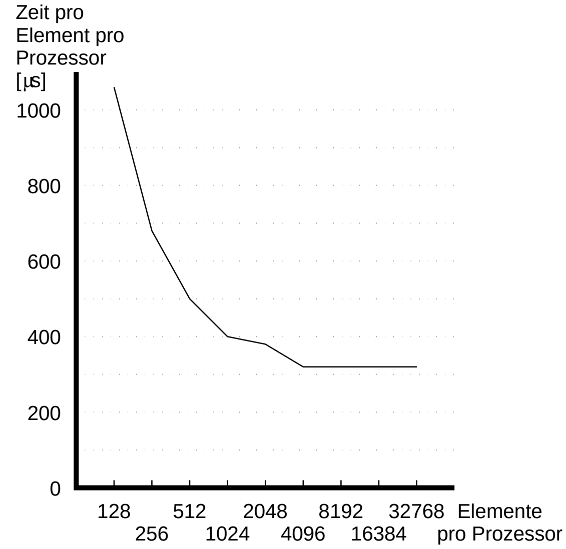


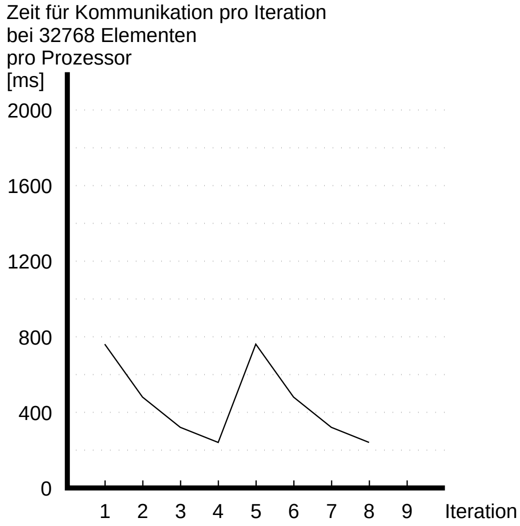


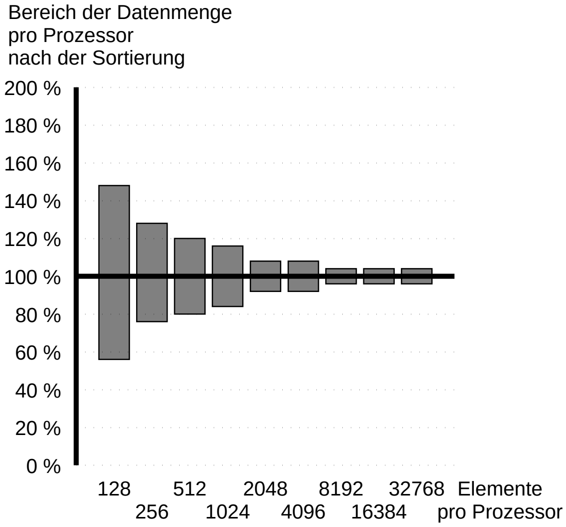
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	929	585	449	371	345	312	313	311	322
Gesamtzeit [s]	0.12	0.15	0.23	0.38	0.71	1.28	2.57	5.11	10.58
Zeit für Kommunikation [ms]	114	131	180	269	448	699	1299	2327	4591
Zeit für lokale Sortierung [ms]	18	30	69	141	314	670	1524	3181	6906
Minimale Anzahl Elemente	71	188	420	905	1835	3784	7869	15871	31905
Maximale Anzahl Elemente	182	337	618	1161	2232	4330	8767	16945	33500
Anzahl Datenpakete	2048	2048	2048	2048	3511	5705	9988	18303	34775
Gesamtweglänge aller Datenpaket	7680	7680	7680	7680	13300	21521	37530	68717	130452
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

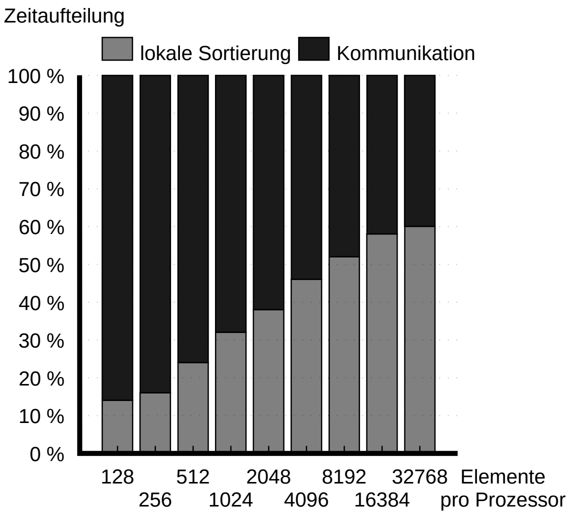
Datenblatt 51

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





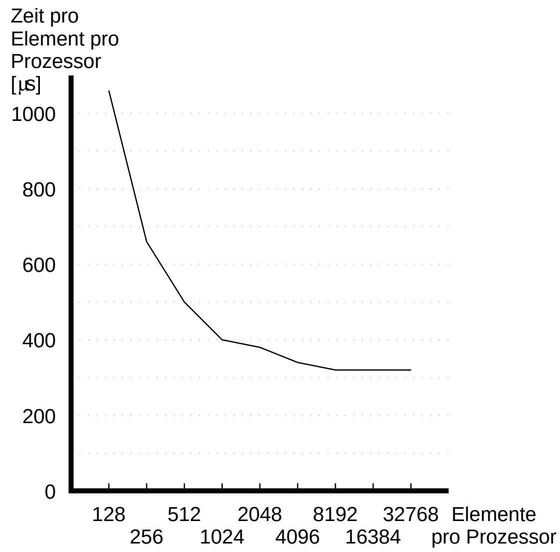


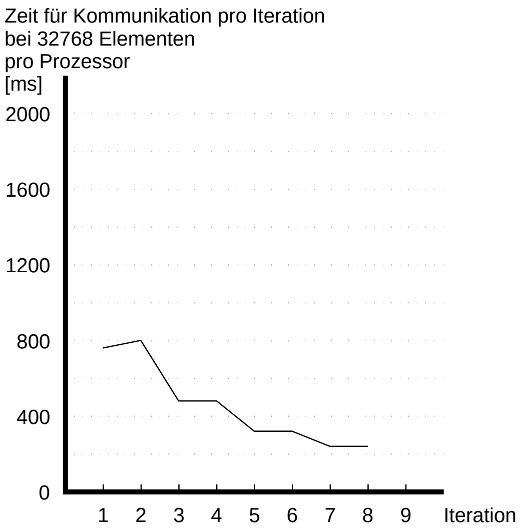


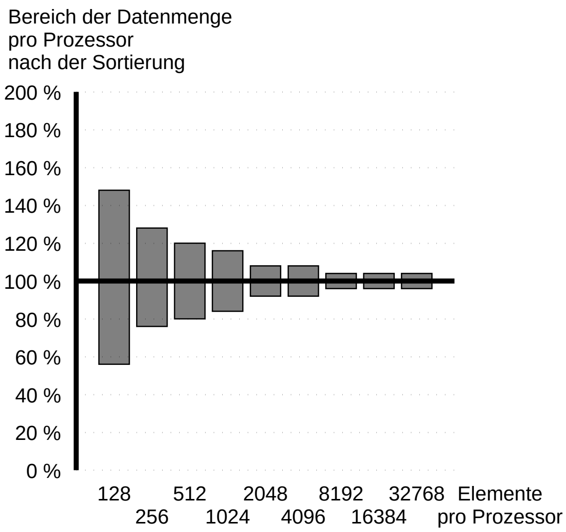
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1061	683	507	410	377	328	321	315	324
Gesamtzeit [s]	0.14	0.18	0.26	0.42	0.77	1.35	2.63	5.17	10.64
Zeit für Kommunikation [ms]	130	160	210	309	509	762	1360	2392	4702
Zeit für lokale Sortierung [ms]	19	30	69	143	319	670	1527	3181	6904
Minimale Anzahl Elemente	71	196	401	878	1871	3840	7872	15832	31917
Maximale Anzahl Elemente	189	331	622	1174	2225	4346	8593	16952	33505
Anzahl Datenpakete	2048	2048	2048	2048	3541	5750	9979	18310	34794
Gesamtweglänge aller Datenpaket	7680	7680	7680	7680	13497	21666	37487	68712	130486
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

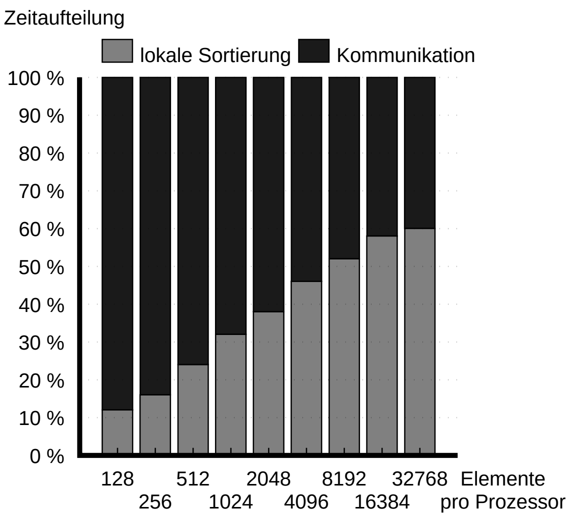
Datenblatt 52

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





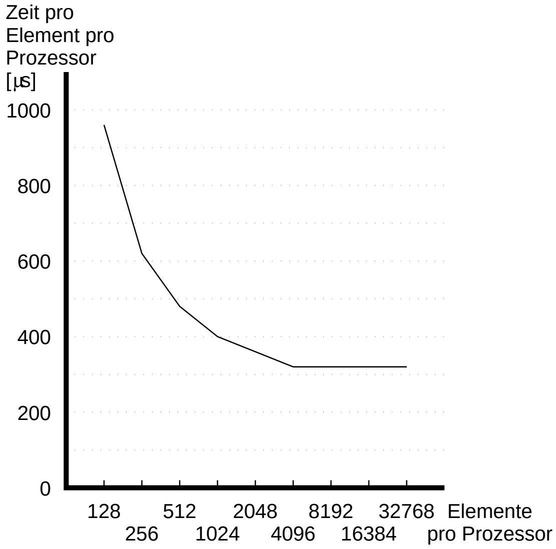


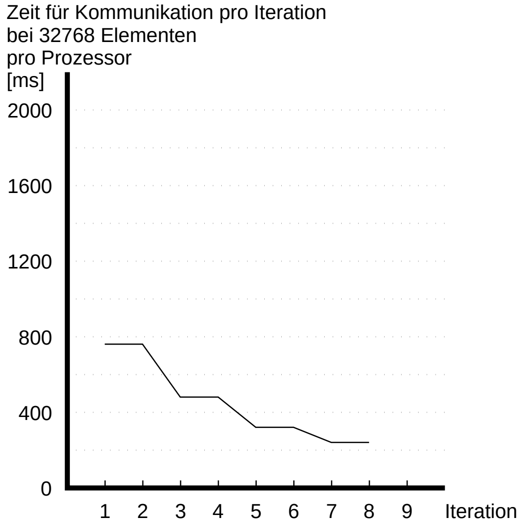


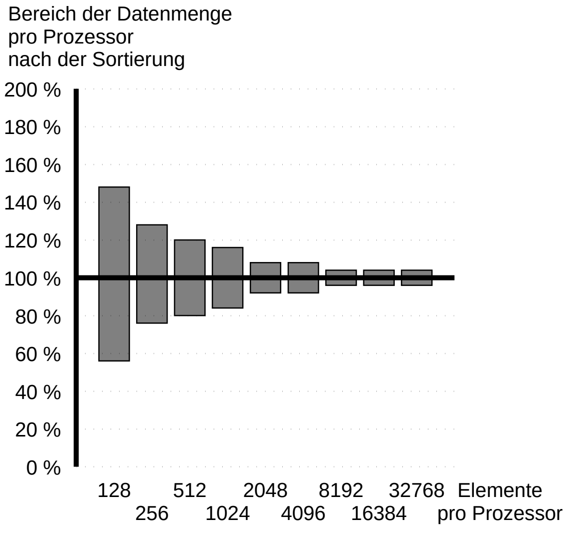
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1061	664	507	410	385	333	323	316	325
Gesamtzeit [s]	0.14	0.17	0.26	0.42	0.79	1.37	2.65	5.19	10.66
Zeit für Kommunikation [ms]	130	150	210	310	530	780	1379	2406	4720
Zeit für lokale Sortierung [ms]	19	30	69	140	319	670	1525	3181	6907
Minimale Anzahl Elemente	71	196	409	879	1871	3840	7872	15849	31912
Maximale Anzahl Elemente	189	331	622	1174	2225	4346	8607	16952	33489
Anzahl Datenpakete	2048	2048	2048	2048	3542	5750	9978	18310	34793
Gesamtweglänge aller Datenpaket	8800	8800	8800	8800	15568	25004	43182	78878	149563
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

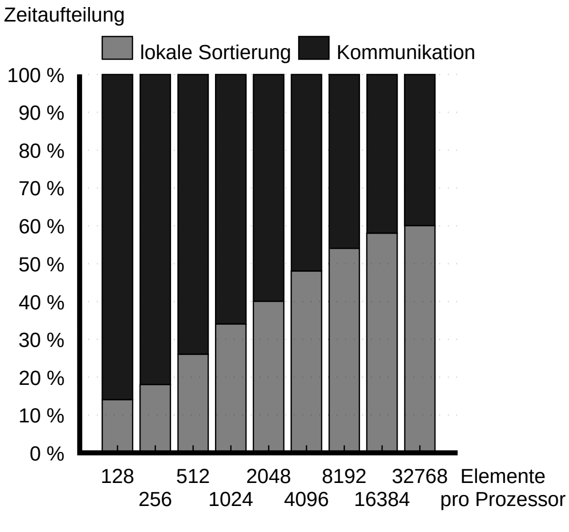
Datenblatt 53

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	H-Numerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





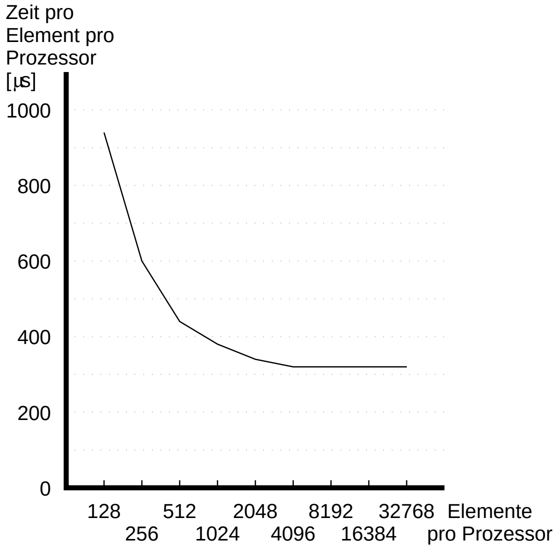


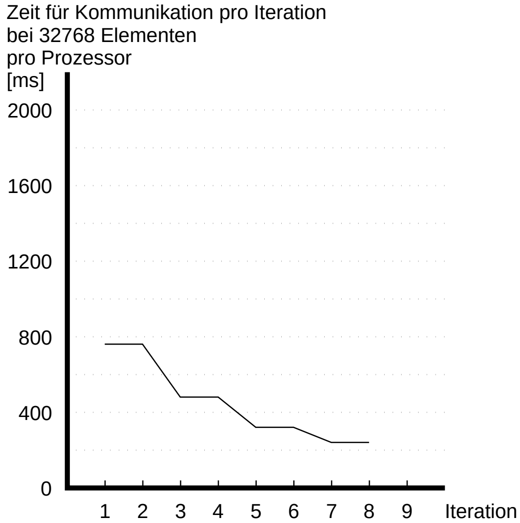


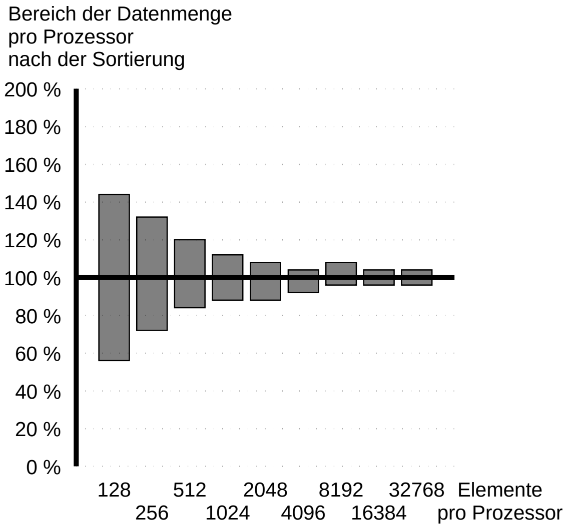
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	952	624	472	396	368	325	319	314	324
Gesamtzeit [s]	0.12	0.16	0.24	0.41	0.75	1.34	2.62	5.15	10.63
Zeit für Kommunikation [ms]	120	140	199	290	497	750	1345	2374	4689
Zeit für lokale Sortierung [ms]	19	30	69	145	318	670	1524	3180	6906
Minimale Anzahl Elemente	71	196	409	877	1871	3840	7872	15828	31912
Maximale Anzahl Elemente	189	331	622	1174	2225	4346	8608	16952	33488
Anzahl Datenpakete	2048	2048	2048	2048	3542	5750	9978	18311	34793
Gesamtweglänge aller Datenpaket	7680	7680	7680	7680	13596	21808	37704	68845	130525
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

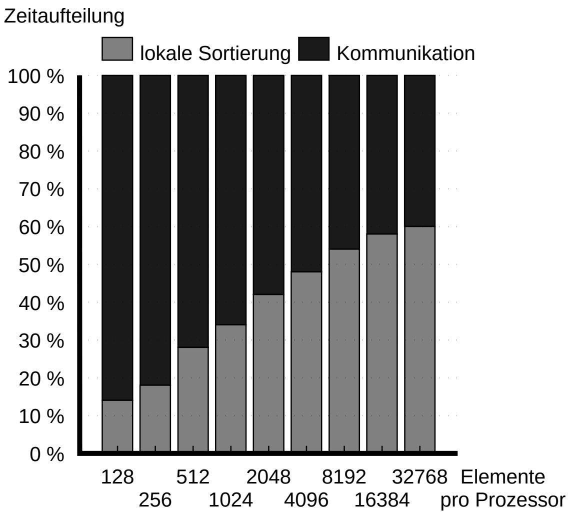
Datenblatt 54

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Z-Numerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





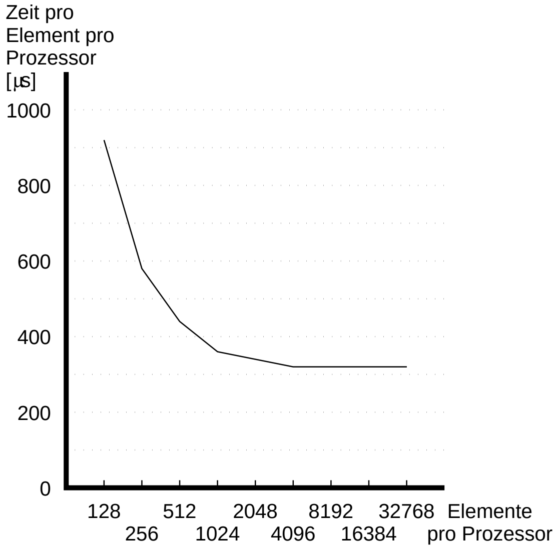


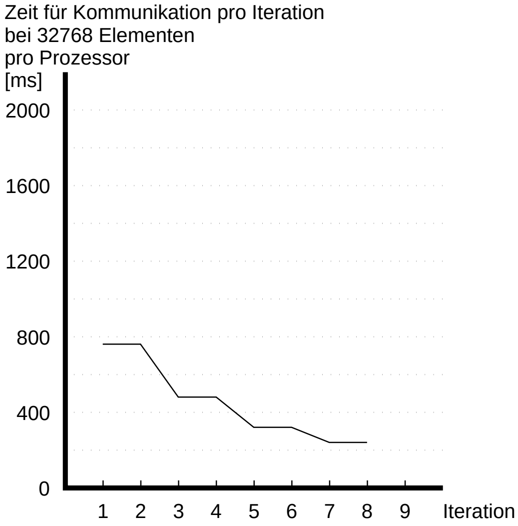


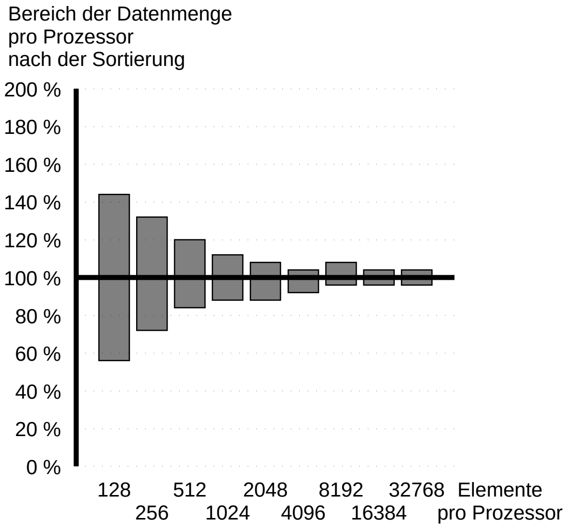
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	937	600	449	374	346	313	313	311	323
Gesamtzeit [s]	0.12	0.15	0.23	0.38	0.71	1.29	2.57	5.11	10.59
Zeit für Kommunikation [ms]	110	130	180	270	449	700	1299	2329	4649
Zeit für lokale Sortierung [ms]	19	30	69	143	318	670	1524	3181	6905
Minimale Anzahl Elemente	71	188	420	905	1835	3784	7869	15871	31905
Maximale Anzahl Elemente	182	337	618	1161	2232	4330	8767	16945	33500
Anzahl Datenpakete	2048	2048	2048	2048	3511	5704	9989	18304	34775
Gesamtweglänge aller Datenpaket	7936	7936	7936	7936	13832	22299	38897	71078	134816
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

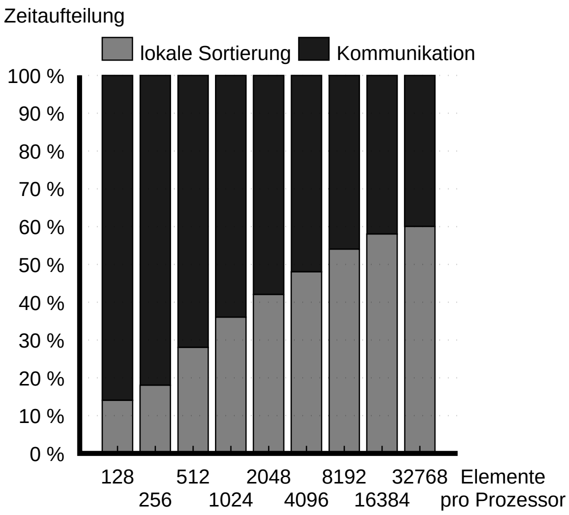
Datenblatt 55

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	1	**Prozessornumerierung**	Shuffle-Numerierung
Pivotstrategie	Median der Mediane		
reduzierte Kommunikation	Ja	**exakte Halbierung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





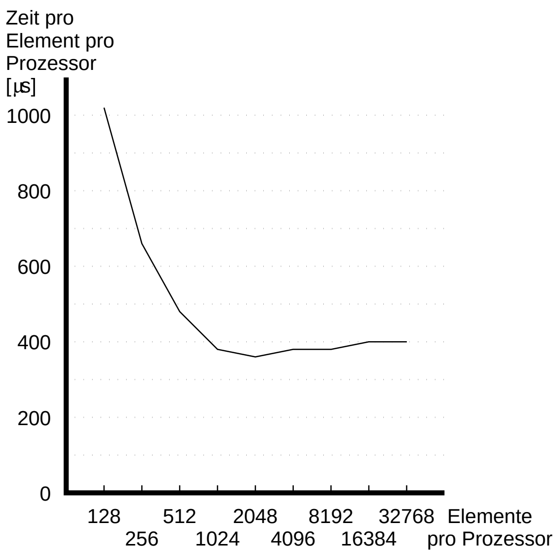


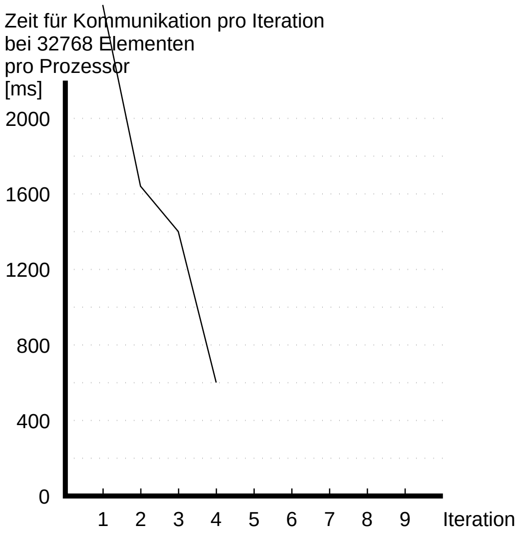


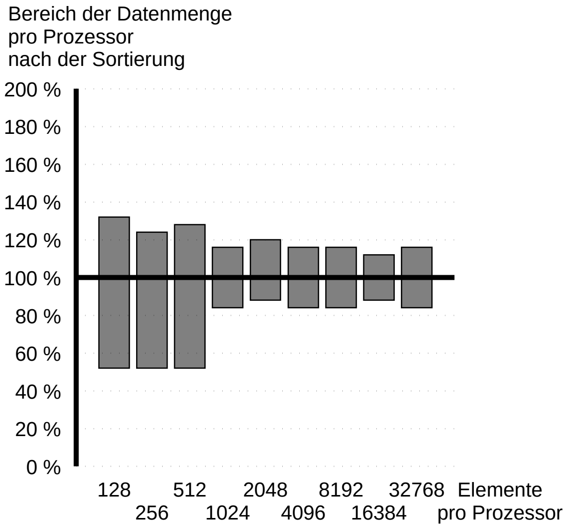
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	913	585	449	365	342	312	313	311	322
Gesamtzeit [s]	0.12	0.15	0.23	0.37	0.70	1.28	2.57	5.10	10.59
Zeit für Kommunikation [ms]	110	130	180	260	446	694	1294	2322	4604
Zeit für lokale Sortierung [ms]	19	30	69	141	319	670	1524	3182	6908
Minimale Anzahl Elemente	71	188	420	905	1835	3784	7869	15871	31905
Maximale Anzahl Elemente	182	337	618	1161	2232	4330	8767	16945	33500
Anzahl Datenpakete	2048	2048	2048	2048	3511	5705	9988	18303	34775
Gesamtweglänge aller Datenpaket	7680	7680	7680	7680	13397	21586	37652	68797	130471
Durchschn. Iterationstiefe	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00	8.00

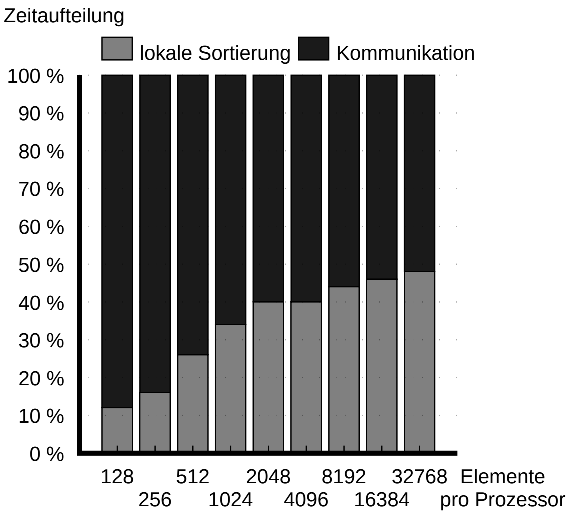
Datenblatt 56

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Nein	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





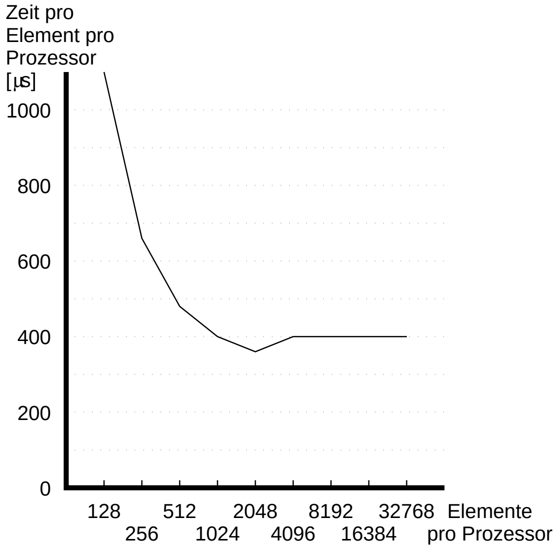


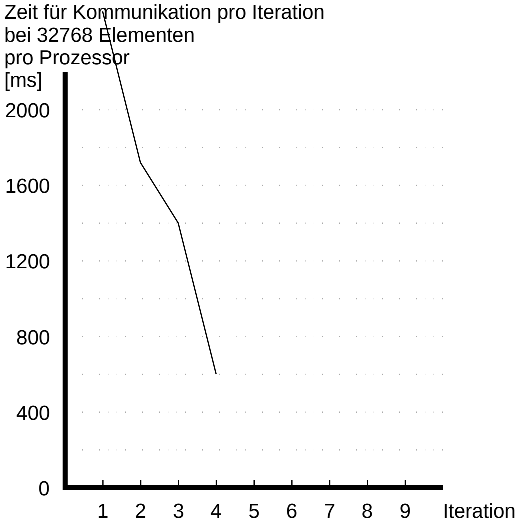


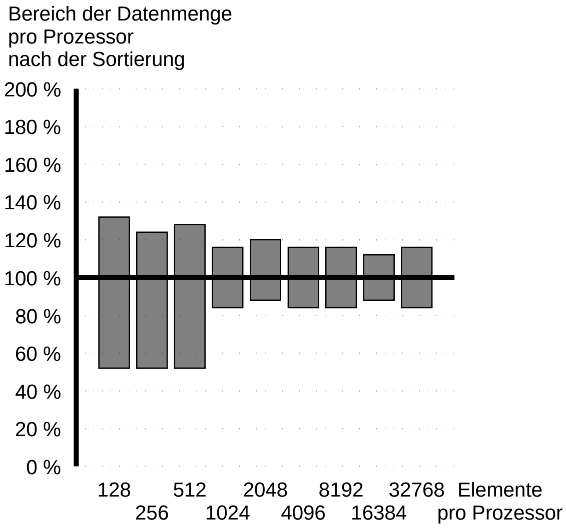
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1015	660	488	378	351	383	388	392	403
Gesamtzeit [s]	0.13	0.17	0.25	0.39	0.72	1.57	3.18	6.44	13.23
Zeit für Kommunikation [ms]	130	150	200	270	462	989	1905	3663	7287
Zeit für lokale Sortierung [ms]	17	30	69	142	319	670	1523	3183	6904
Minimale Anzahl Elemente	66	129	276	842	1781	3446	7020	14126	28169
Maximale Anzahl Elemente	171	322	653	1174	2454	4798	9660	18418	37511
Anzahl Datenpakete	4763	4775	4772	4778	4779	7209	11591	20042	36735
Gesamtweglänge aller Datenpaket	28711	28362	26824	26561	26578	39142	62509	107223	195858
Durchschn. Iterationstiefe	4.02	4.02	4.01	4.00	4.00	4.00	4.00	4.00	4.00

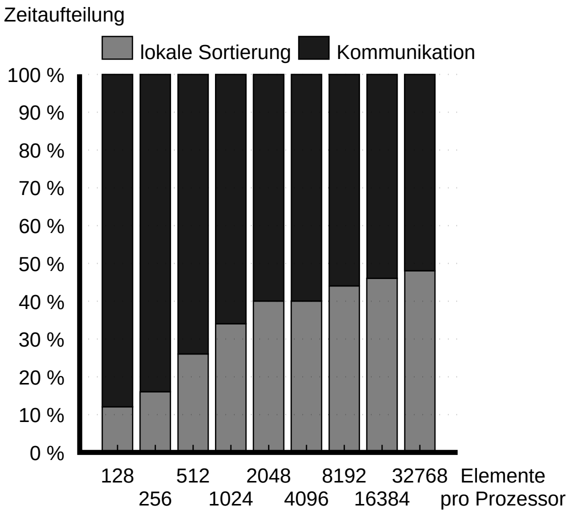
Datenblatt 57

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Nein	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





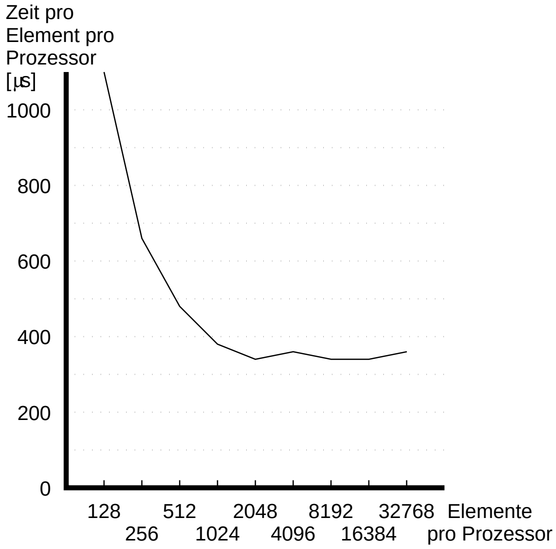


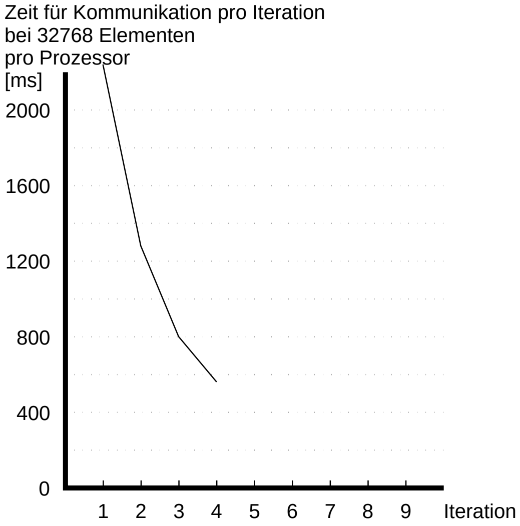


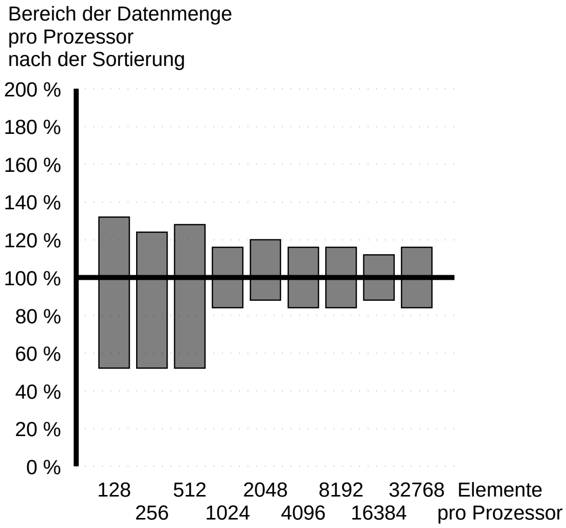
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1093	667	488	390	360	391	392	393	403
Gesamtzeit [s]	0.14	0.17	0.25	0.40	0.74	1.61	3.22	6.45	13.25
Zeit für Kommunikation [ms]	130	154	203	287	479	1017	1952	3665	7306
Zeit für lokale Sortierung [ms]	19	30	69	142	318	670	1526	3180	6906
Minimale Anzahl Elemente	66	129	276	842	1781	3446	7020	14126	28169
Maximale Anzahl Elemente	171	322	653	1174	2454	4798	9660	18418	37511
Anzahl Datenpakete	4763	4775	4772	4778	4779	7209	11591	20042	36735
Gesamtweglänge aller Datenpaket	27709	27851	27899	27973	27966	41442	66234	113528	207353
Durchschn. Iterationstiefe	4.02	4.02	4.01	4.00	4.00	4.00	4.00	4.00	4.00

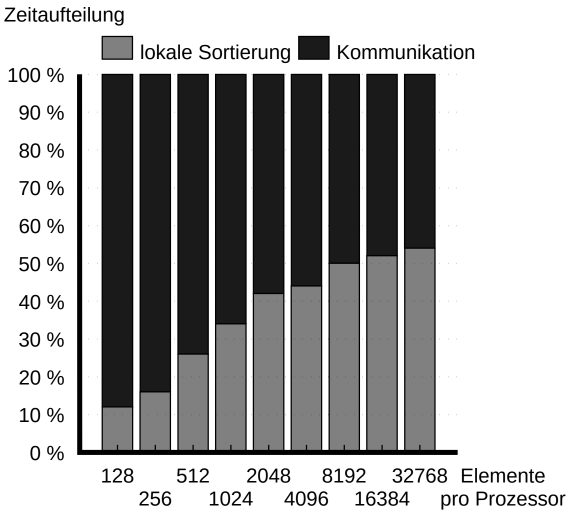
Datenblatt 58

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Nein	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





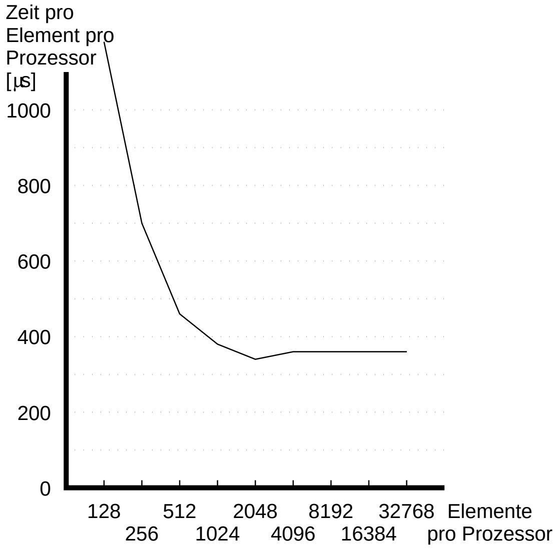


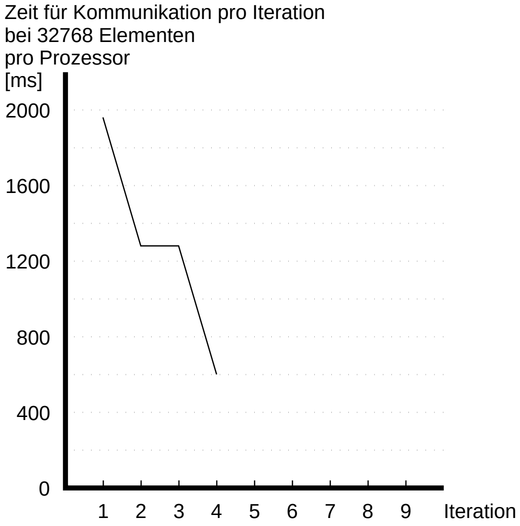


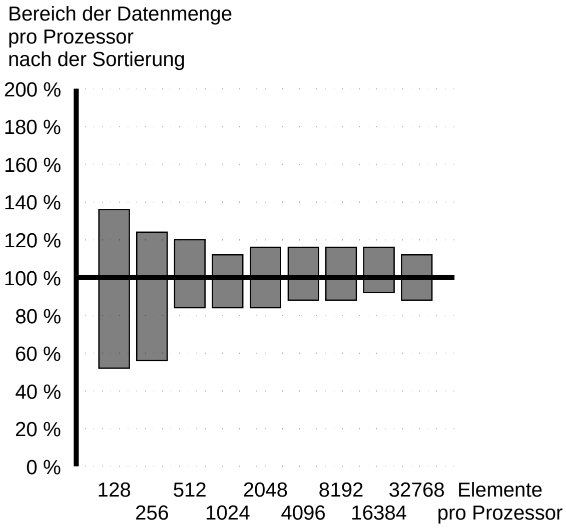
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1093	667	488	376	340	355	347	350	361
Gesamtzeit [s]	0.14	0.17	0.25	0.39	0.70	1.46	2.85	5.74	11.86
Zeit für Kommunikation [ms]	138	154	200	272	438	869	1571	2963	5916
Zeit für lokale Sortierung [ms]	19	30	69	140	315	670	1523	3182	6902
Minimale Anzahl Elemente	66	129	276	842	1781	3446	7020	14126	28169
Maximale Anzahl Elemente	171	322	653	1174	2454	4798	9660	18418	37511
Anzahl Datenpakete	4763	4775	4772	4778	4779	7209	11591	20042	36735
Gesamtweglänge aller Datenpaket	21809	21863	21493	21501	21468	31948	50800	87116	158737
Durchschn. Iterationstiefe	4.02	4.02	4.01	4.00	4.00	4.00	4.00	4.00	4.00

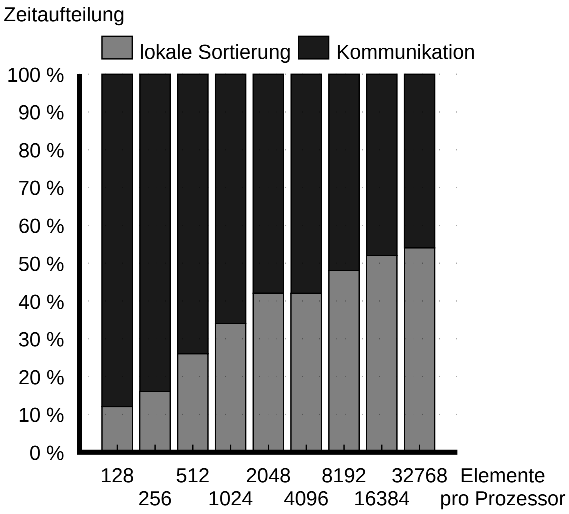
Datenblatt 59

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





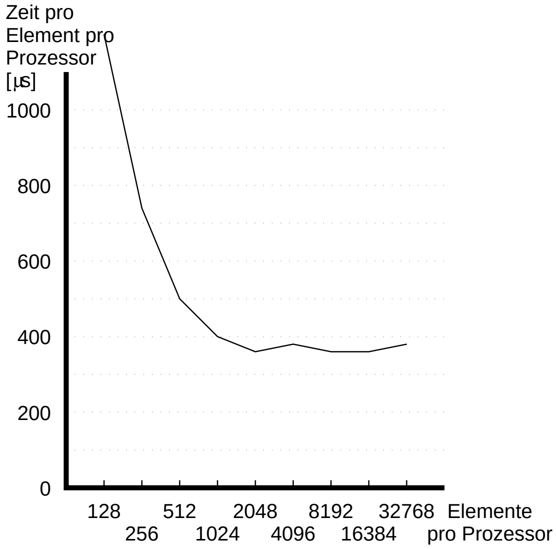


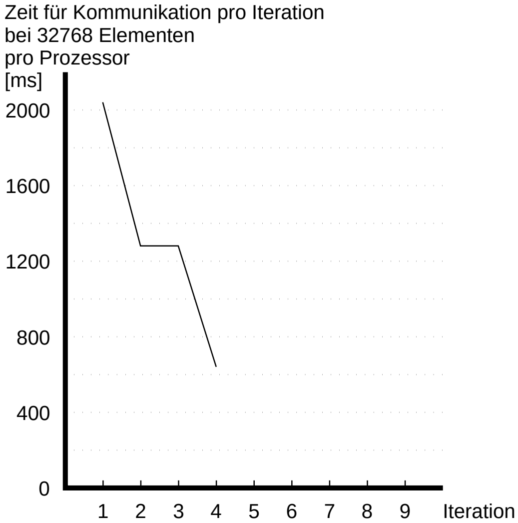


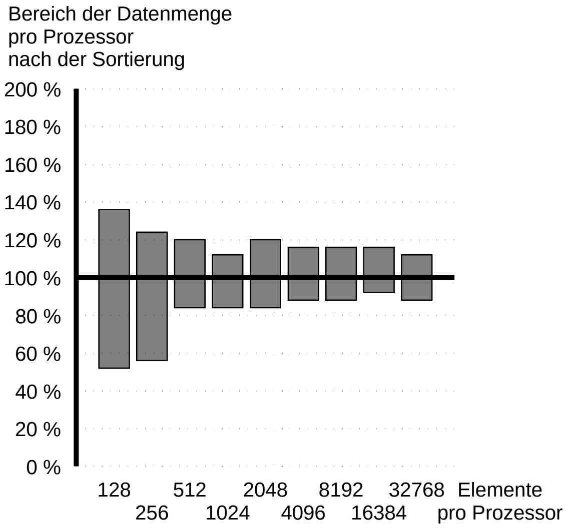
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1171	703	462	379	343	361	355	354	368
Gesamtzeit [s]	0.15	0.18	0.24	0.39	0.70	1.48	2.91	5.81	12.09
Zeit für Kommunikation [ms]	146	169	190	270	446	898	1640	3025	6151
Zeit für lokale Sortierung [ms]	19	30	70	142	319	670	1523	3182	6903
Minimale Anzahl Elemente	69	141	424	876	1761	3650	7166	14826	29307
Maximale Anzahl Elemente	172	321	610	1166	2437	4741	9634	19071	37283
Anzahl Datenpakete	3731	3747	3746	3750	3754	5548	8810	15217	27739
Gesamtweglänge aller Datenpaket	26602	25864	24516	24535	24540	35220	55398	94537	171852
Durchschn. Iterationstiefe	4.02	4.01	4.00	4.00	4.00	4.00	4.00	4.00	4.00

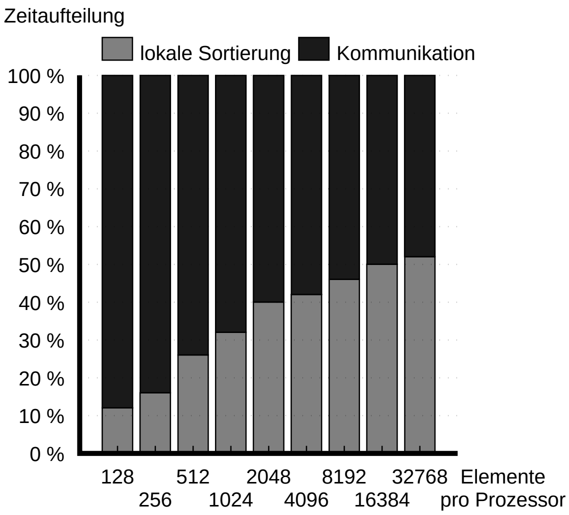
Datenblatt 60

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





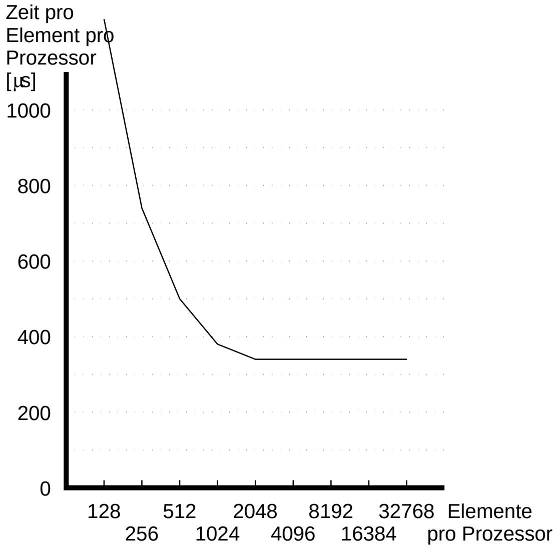


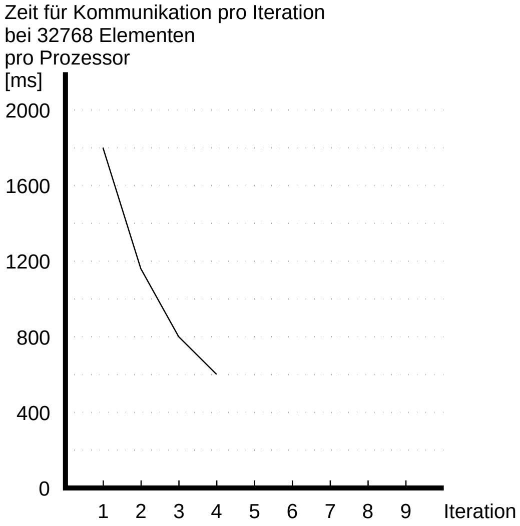


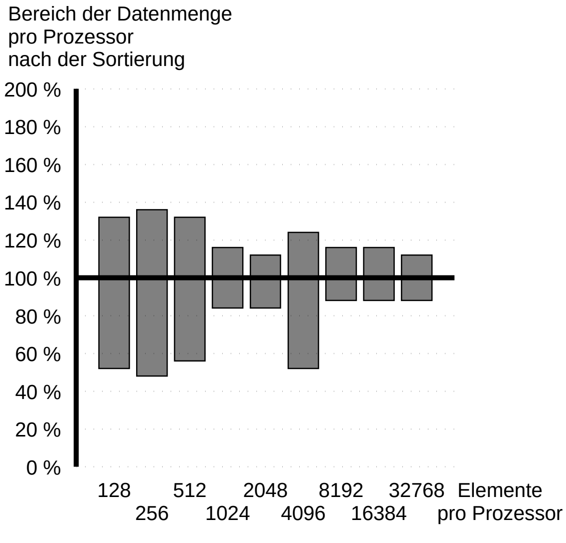
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1202	734	507	400	366	376	367	363	372
Gesamtzeit [s]	0.15	0.19	0.26	0.41	0.75	1.54	3.01	5.97	12.22
Zeit für Kommunikation [ms]	150	170	210	300	489	959	1734	3187	6280
Zeit für lokale Sortierung [ms]	19	30	70	141	319	670	1526	3182	6904
Minimale Anzahl Elemente	69	141	424	876	1761	3650	7166	14826	29307
Maximale Anzahl Elemente	172	321	610	1166	2437	4741	9634	19071	37283
Anzahl Datenpakete	3731	3747	3746	3750	3754	5548	8810	15217	27739
Gesamtweglänge aller Datenpaket	25918	26159	26158	26161	26176	37765	59927	102296	185772
Durchschn. Iterationstiefe	4.02	4.01	4.00	4.00	4.00	4.00	4.00	4.00	4.00

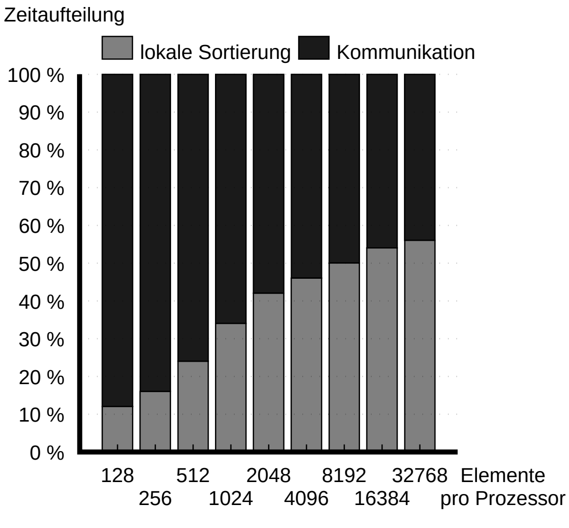
Datenblatt 61

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Nein





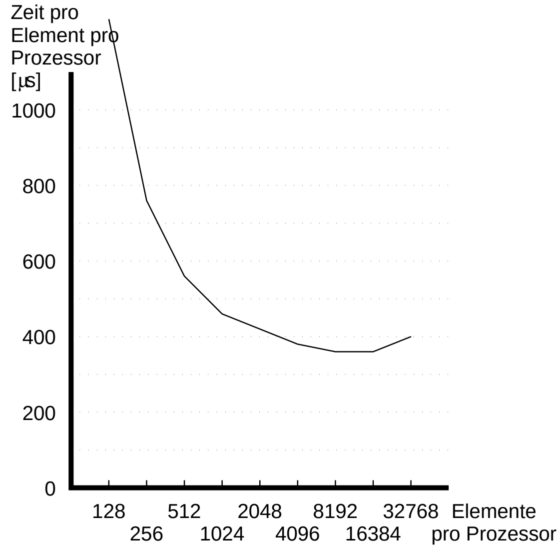


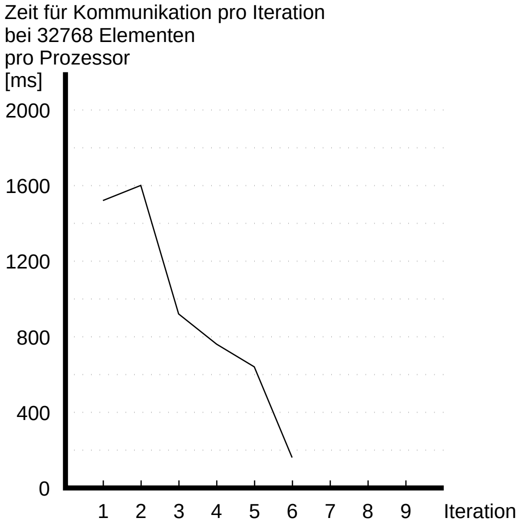


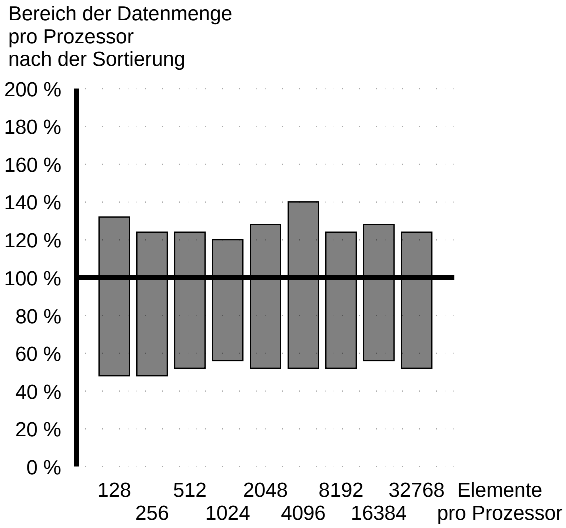
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1241	734	505	379	340	342	337	336	344
Gesamtzeit [s]	0.16	0.19	0.26	0.39	0.70	1.41	2.76	5.52	11.29
Zeit für Kommunikation [ms]	150	170	210	278	437	817	1490	2742	5351
Zeit für lokale Sortierung [ms]	19	30	69	144	319	670	1526	3181	6902
Minimale Anzahl Elemente	66	122	278	865	1741	2131	7106	14309	29258
Maximale Anzahl Elemente	171	352	674	1183	2280	5107	9464	19046	36816
Anzahl Datenpakete	3732	3738	3747	3752	3752	5500	8767	15201	27696
Gesamtweglänge aller Datenpaket	22044	21932	21738	21764	21760	31635	49423	84613	152934
Durchschn. Iterationstiefe	4.03	4.02	4.01	4.00	4.00	4.01	4.00	4.00	4.00

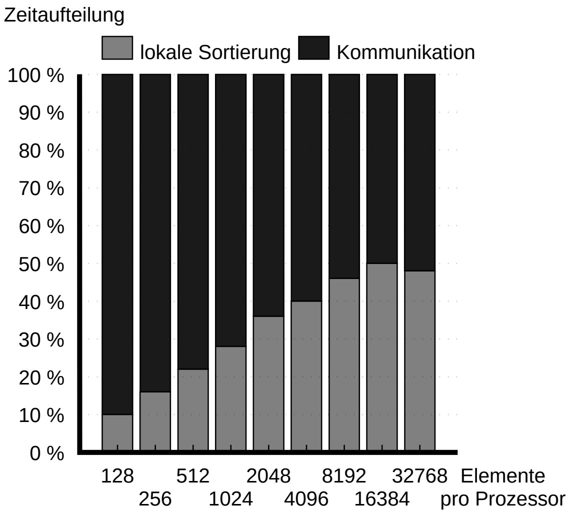
Datenblatt 62

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	2	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





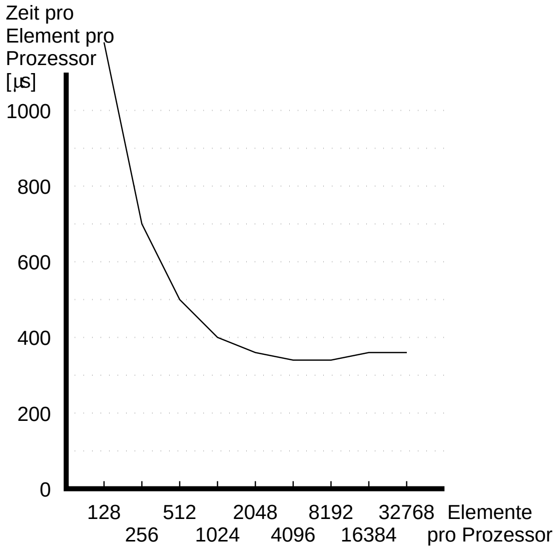


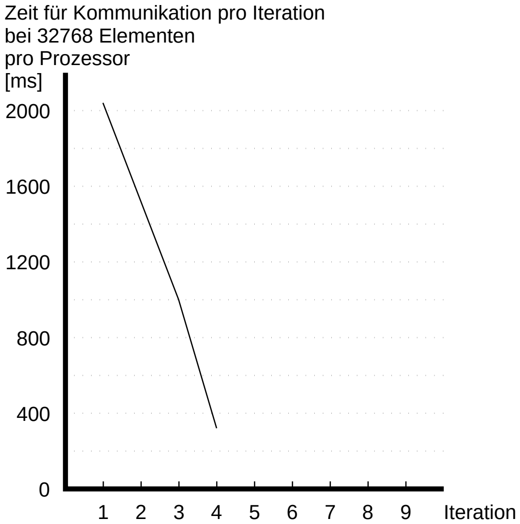


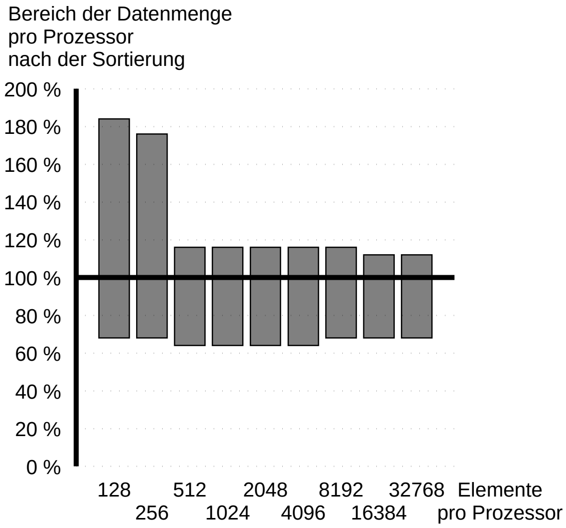
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1249	757	552	459	416	380	365	369	407
Gesamtzeit [s]	0.16	0.19	0.28	0.47	0.85	1.56	3.00	6.07	13.354
Zeit für Kommunikation [ms]	160	180	239	359	591	979	1721	3288	7416
Zeit für lokale Sortierung [ms]	19	30	69	140	319	670	15.25	3183	6903
Minimale Anzahl Elemente	60	118	259	554	1085	2194	4227	8957	17358
Maximale Anzahl Elemente	170	320	633	1231	2609	5811	10069	20707	40508
Anzahl Datenpakete	3484	3491	3494	3502	3502	5693	8709	15852	29705
Gesamtweglänge aller Datenpaket	21484	21595	21529	21540	21632	33754	51104	92914	172440
Durchschn. Iterationstiefe	5.10	5.10	5.10	5.10	5.10	5.11	5.10	5.10	5.10

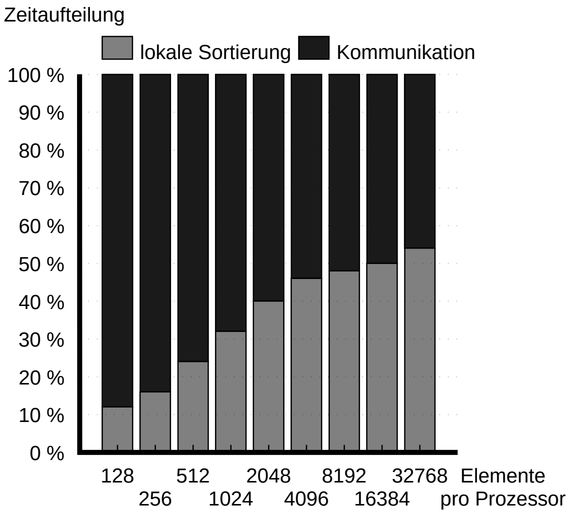
Datenblatt 63

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	4	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





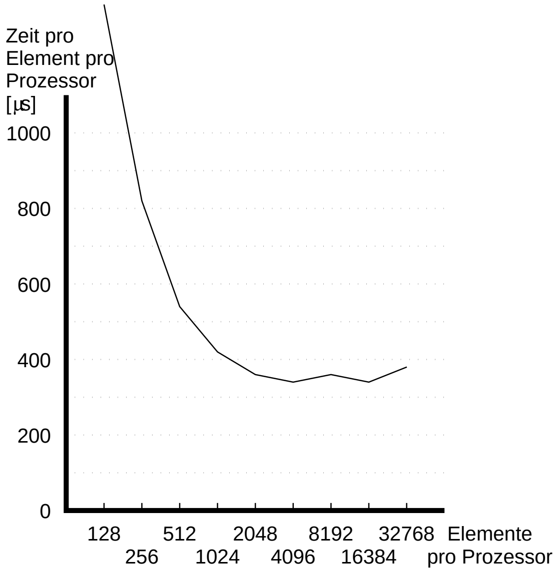


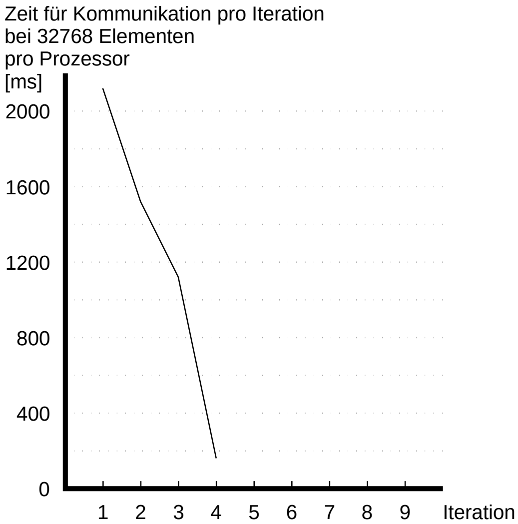


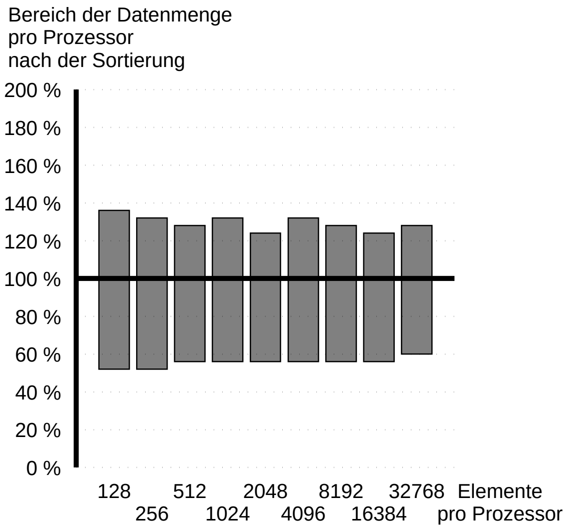
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1186	710	507	409	353	332	349	357	362
Gesamtzeit [s]	0.15	0.18	0.26	0.42	0.72	1.36	2.86	5.86	11.89
Zeit für Kommunikation [ms]	150	170	210	307	465	779	1588	3084	5948
Zeit für lokale Sortierung [ms]	19	30	69	141	319	670	1525	3182	6902
Minimale Anzahl Elemente	87	169	336	672	1344	2649	5503	11093	21926
Maximale Anzahl Elemente	235	453	592	1168	2348	4690	9662	18286	36537
Anzahl Datenpakete	3937	3938	3950	3954	4151	4422	7876	14582	26251
Gesamtweglänge aller Datenpaket	26395	26306	26520	26631	26902	27042	47941	89107	157514
Durchschn. Iterationstiefe	3.99	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00

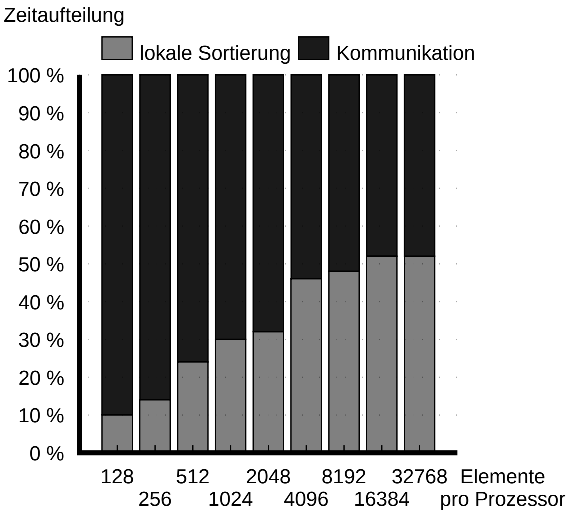
Datenblatt 64

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	5	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





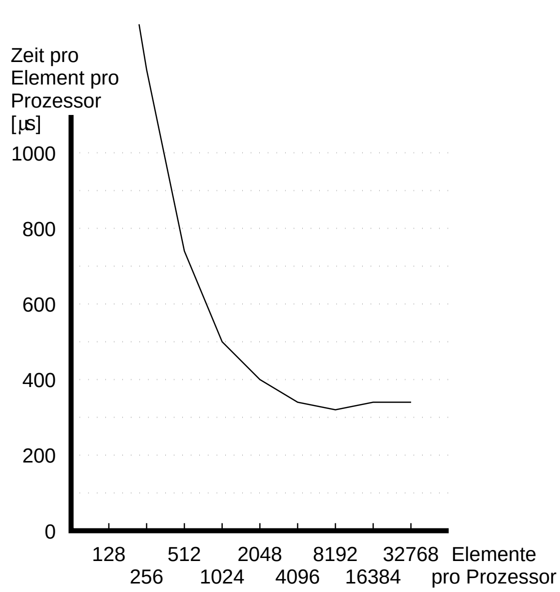


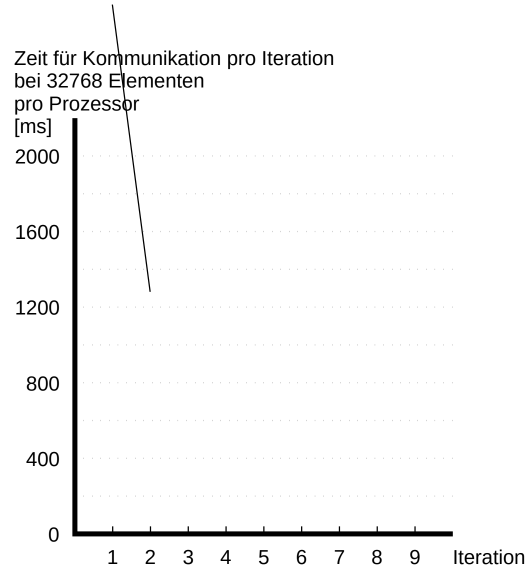


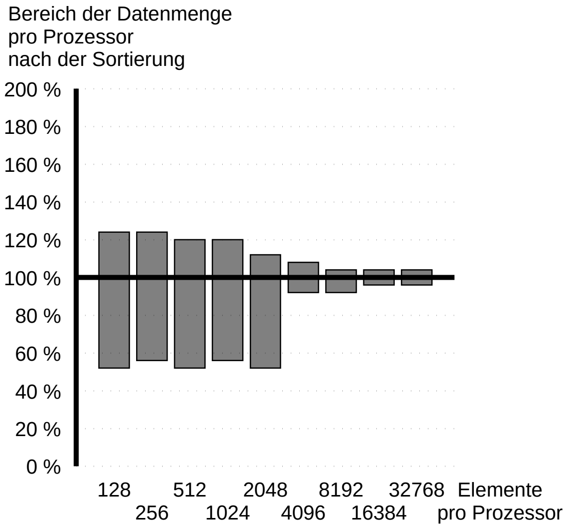
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1343	816	542	427	366	337	364	349	376
Gesamtzeit [s]	0.17	0.21	0.28	0.44	0.75	1.38	2.99	5.73	12.34
Zeit für Kommunikation [ms]	170	190	229	321	487	798	1717	2954	6404
Zeit für lokale Sortierung [ms]	19	30	70	141	319	671	1524	3180	6903
Minimale Anzahl Elemente	64	130	285	592	1172	2370	4673	9498	19027
Maximale Anzahl Elemente	172	336	657	1334	2568	5331	10341	20398	41323
Anzahl Datenpakete	4405	4416	4417	4422	4429	4510	9198	12511	24042
Gesamtweglänge aller Datenpaket	29116	29272	29280	29285	29302	29508	52672	79505	153174
Durchschn. Iterationstiefe	3.31	3.31	3.31	3.31	3.31	3.31	3.31	3.31	3.31

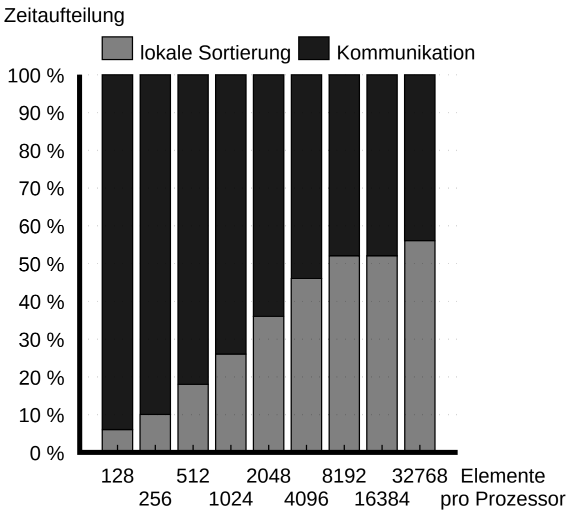
Datenblatt 65

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	15	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Nein
exakte Partnerprozessoren	Nein	**Datentauschinvertierung**	Ja





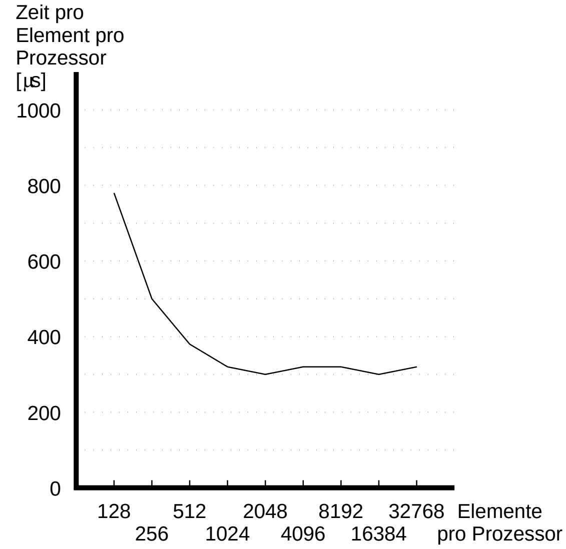


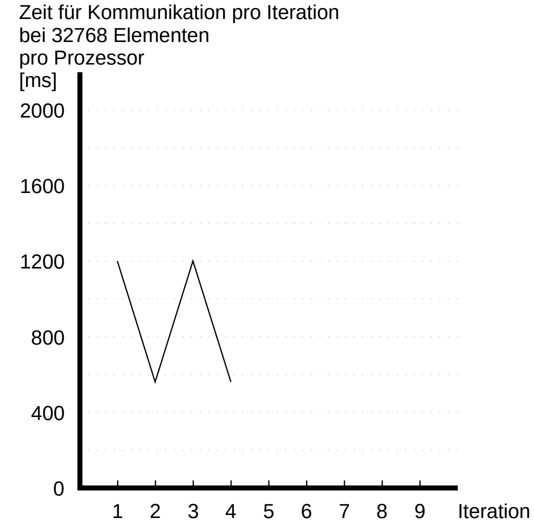


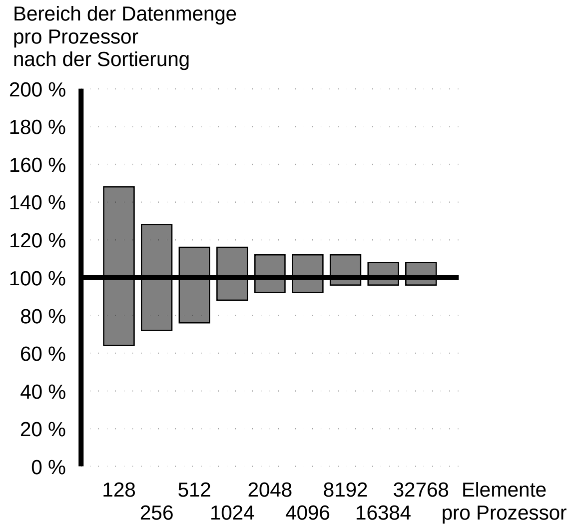
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	2265	1217	749	507	399	337	326	349	348
Gesamtzeit [s]	0.29	0.31	0.38	0.52	0.82	1.38	2.68	5.72	11.43
Zeit für Kommunikation [ms]	289	300	340	406	557	801	1402	2945	5489
Zeit für lokale Sortierung [ms]	19	30	70	143	319	671	1525	3181	6905
Minimale Anzahl Elemente	65	145	272	552	1092	3850	7482	15498	31534
Maximale Anzahl Elemente	161	322	614	1224	2298	4540	8661	17146	34022
Anzahl Datenpakete	7813	7870	7879	7903	7906	7919	7919	13544	21820
Gesamtweglänge aller Datenpaket	56017	56059	56359	56257	56423	56274	56291	95028	153226
Durchschn. Iterationstiefe	2.05	2.02	2.02	2.01	2.01	2.00	2.00	2.00	2.00

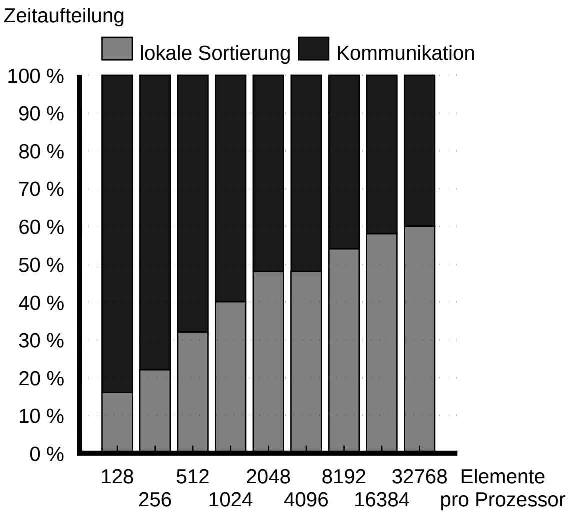
Datenblatt 66

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Zeilennumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





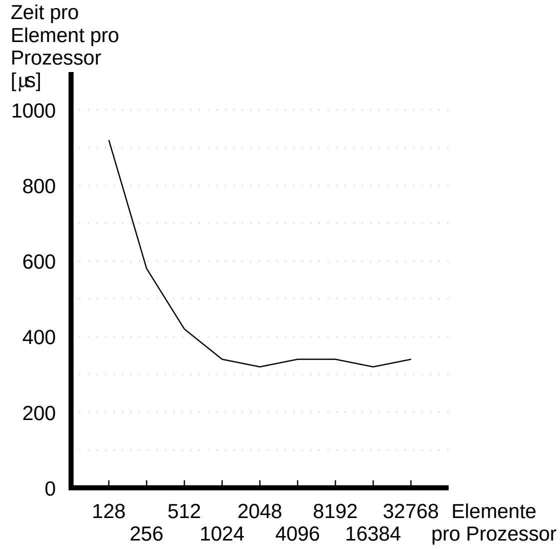


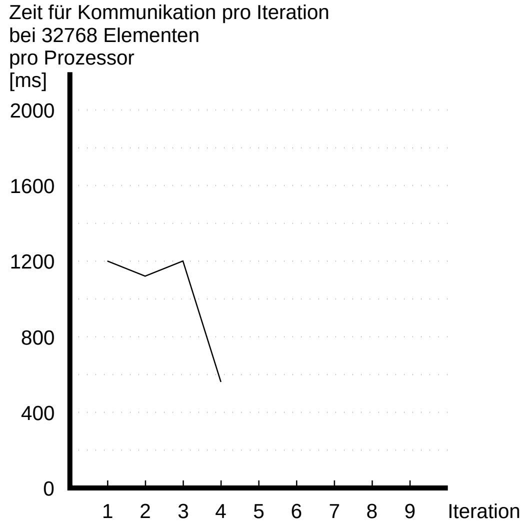


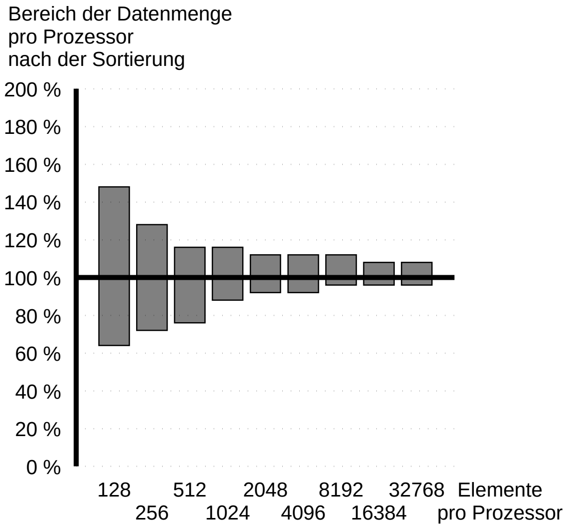
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	781	507	386	312	292	314	313	310	320
Gesamtzeit [s]	0.10	0.13	0.20	0.32	0.60	1.29	2.57	5.08	10.50
Zeit für Kommunikation [ms]	18	110	150	210	340	707	1298	2301	4505
Zeit für lokale Sortierung [ms]	100	30	69	142	314	670	1526	3181	6904
Minimale Anzahl Elemente	81	181	388	913	1884	3818	7781	15866	31960
Maximale Anzahl Elemente	190	330	603	1188	2248	4394	8711	16982	33560
Anzahl Datenpakete	4096	4096	4096	4096	4096	7019	11421	19919	36600
Gesamtweglänge aller Datenpaket	12800	12800	12800	12800	12800	22121	35928	62385	114410
Durchschn. Iterationstiefe	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00

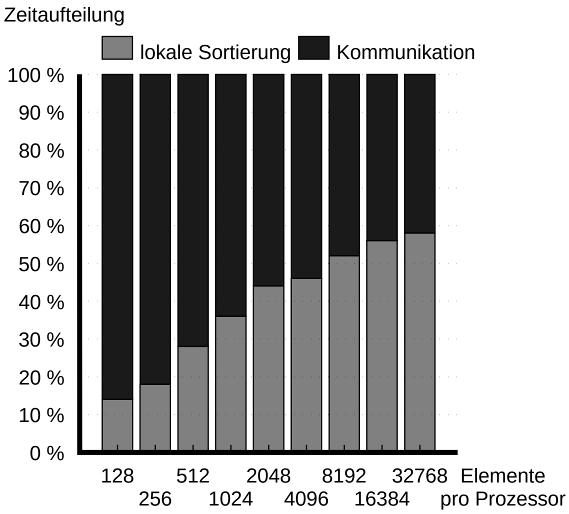
Datenblatt 67

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Schlangennumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





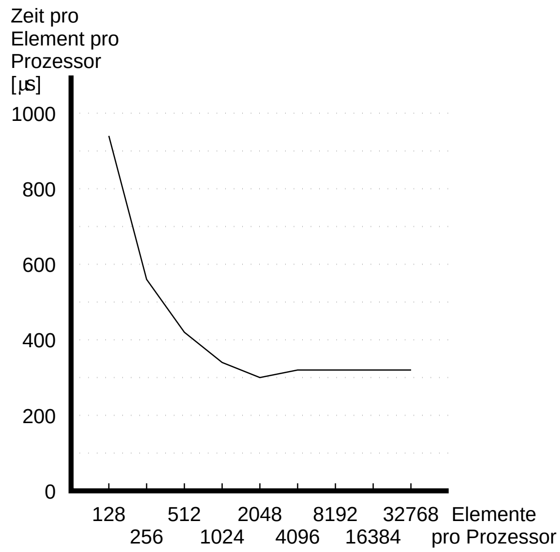


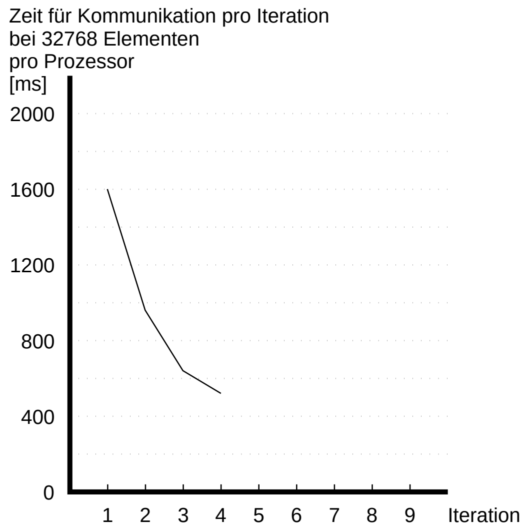


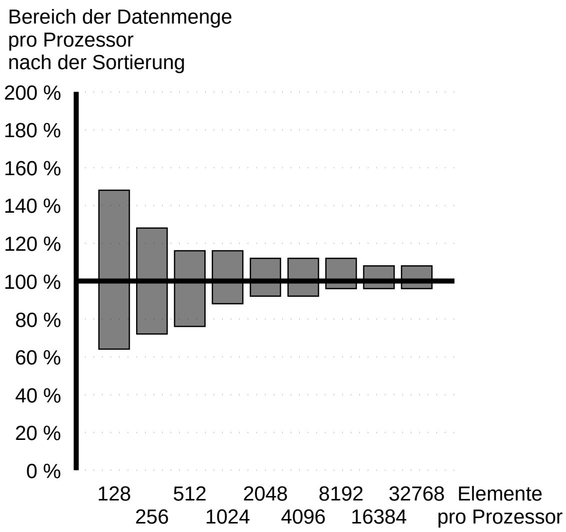
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	921	585	429	341	317	339	332	327	336
Gesamtzeit [s]	0.12	0.15	0.22	0.35	0.65	1.39	2.73	5.38	11.04
Zeit für Kommunikation [ms]	110	130	170	240	390	803	1451	2600	5105
Zeit für lokale Sortierung [ms]	19	30	69	140	319	670	1523	3180	6904
Minimale Anzahl Elemente	81	181	388	913	1884	3818	7776	15866	31972
Maximale Anzahl Elemente	190	330	603	1186	2248	4394	8711	16982	33560
Anzahl Datenpakete	4096	4096	4096	4096	4096	7020	11423	19917	36601
Gesamtweglänge aller Datenpaket	16896	16896	16896	16896	16896	29050	47350	82292	151079
Durchschn. Iterationstiefe	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00

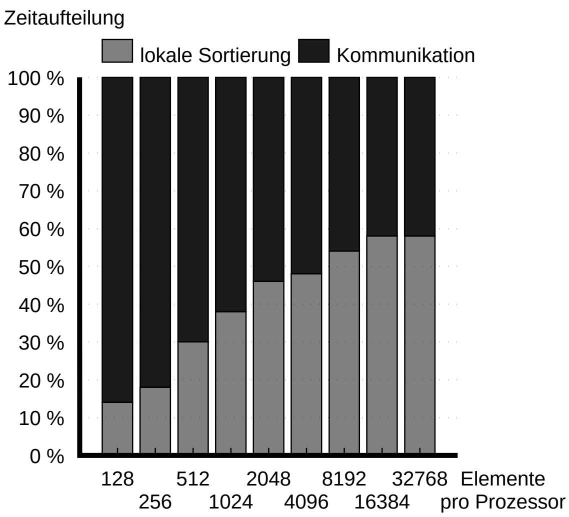
Datenblatt 68

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Hilbertnumerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Ja





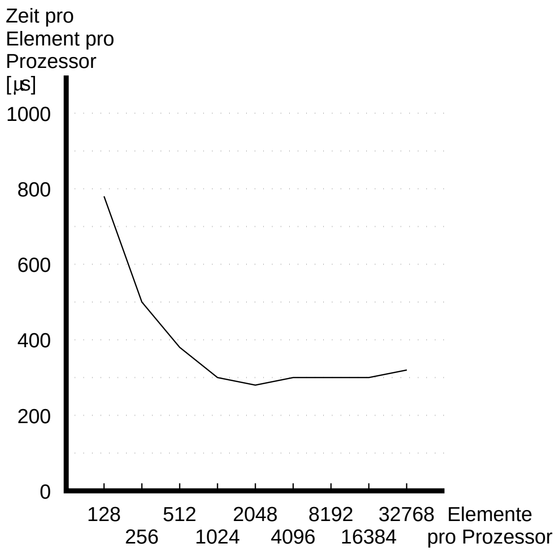


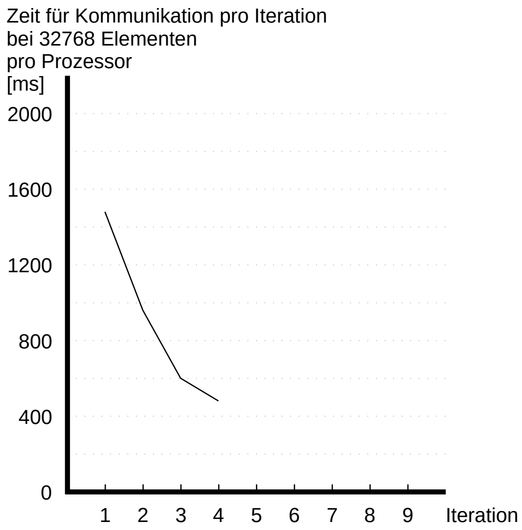


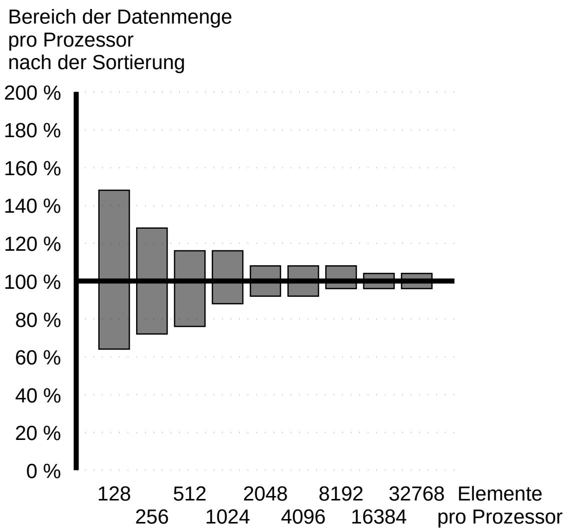
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	937	569	419	332	299	313	312	311	325
Gesamtzeit [s]	0.12	0.15	0.21	0.34	0.61	1.29	2.57	5.11	10.69
Zeit für Kommunikation [ms]	120	130	170	229	360	703	1290	2335	4749
Zeit für lokale Sortierung [ms]	18	30	70	140	318	670	1523	3181	6905
Minimale Anzahl Elemente	81	181	388	913	1884	3818	7778	15866	31970
Maximale Anzahl Elemente	190	330	603	1188	2248	4394	8711	16982	33560
Anzahl Datenpakete	4096	4096	4096	4096	4096	7019	11422	19918	36600
Gesamtweglänge aller Datenpaket	17376	17376	17376	17376	17376	30469	49274	85123	155717
Durchschn. Iterationstiefe	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00

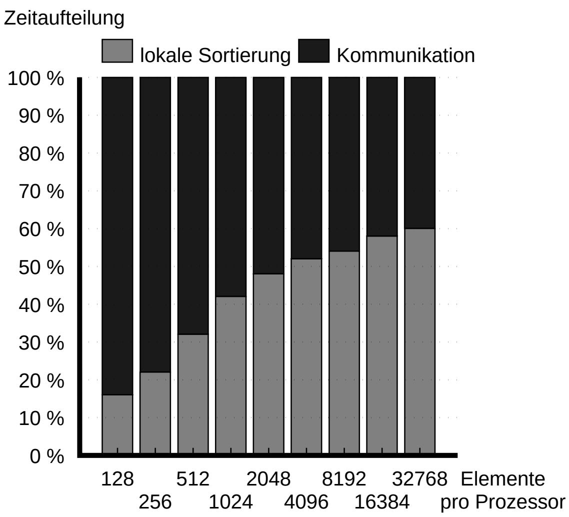
Datenblatt 69

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Shuffle-Numerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





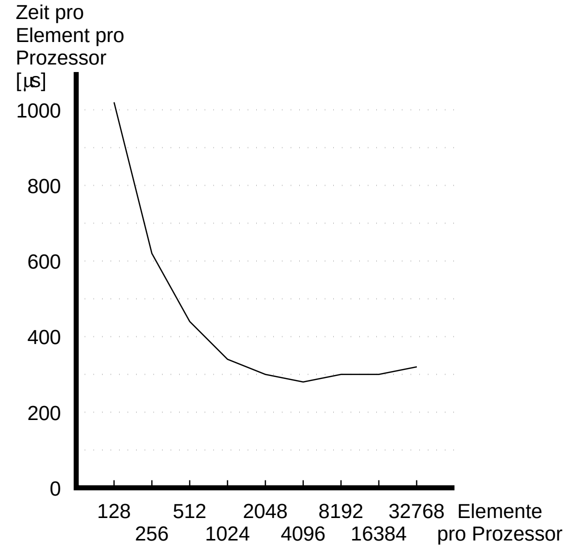


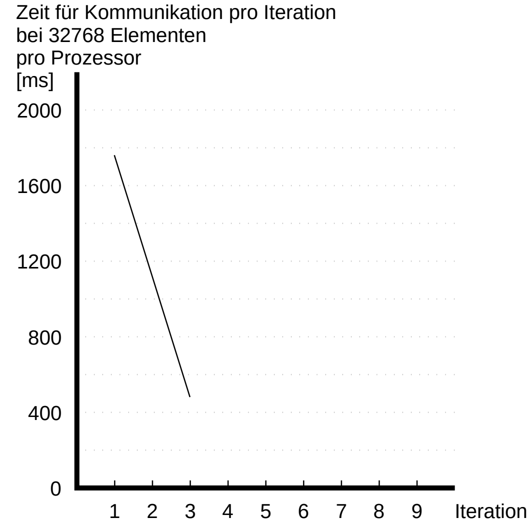


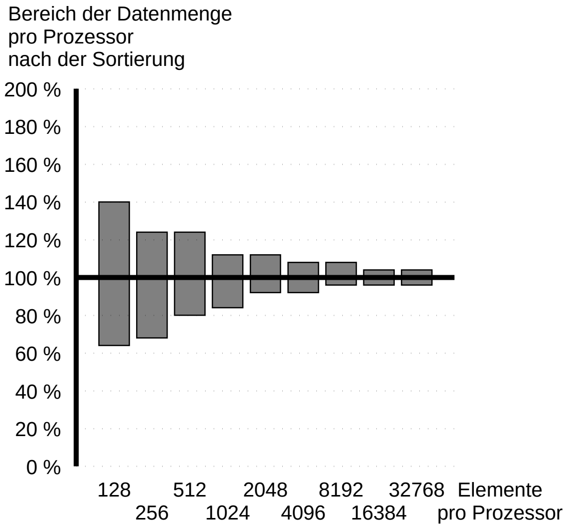
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	781	503	372	302	283	303	307	308	321
Gesamtzeit [s]	0.10	0.13	0.19	0.31	0.58	1.24	2.53	5.06	10.54
Zeit für Kommunikation [ms]	100	110	141	200	318	658	1250	2274	4555
Zeit für lokale Sortierung [ms]	19	30	69	140	317	670	1526	3181	6904
Minimale Anzahl Elemente	81	181	388	913	1884	3818	7780	15866	31959
Maximale Anzahl Elemente	190	330	603	1188	2248	4394	8711	16982	33560
Anzahl Datenpakete	4096	4096	4096	4096	4096	7019	11421	19919	36600
Gesamtweglänge aller Datenpaket	15360	15360	15360	15360	15360	26790	43506	75266	137601
Durchschn. Iterationstiefe	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00

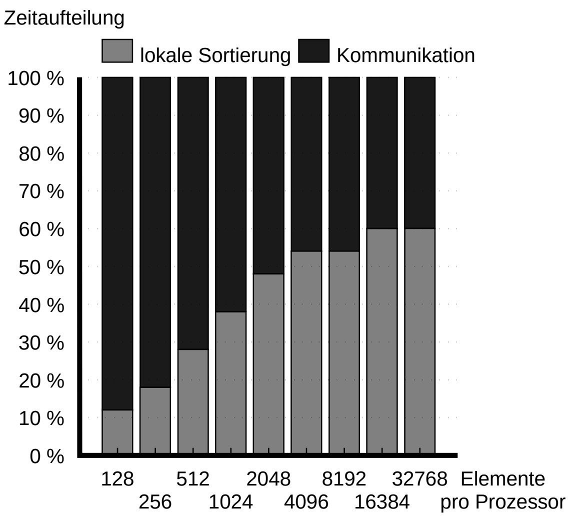
Datenblatt 70

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	7	**Prozessornumerierung**	Shuffle-Numerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





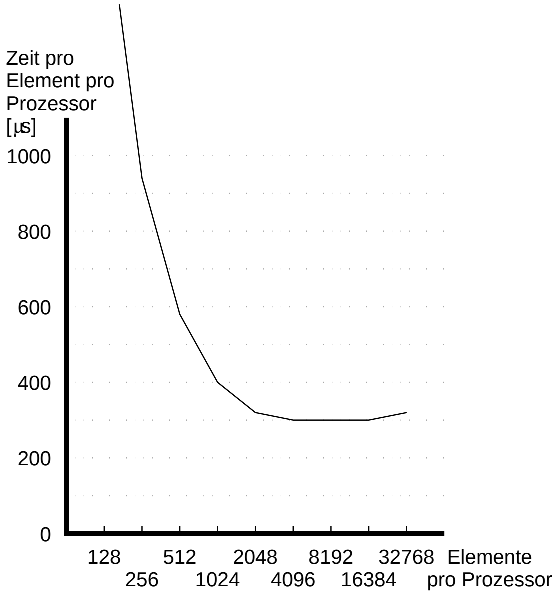


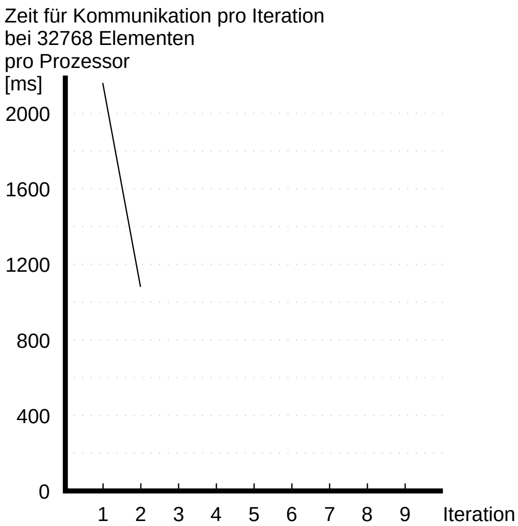


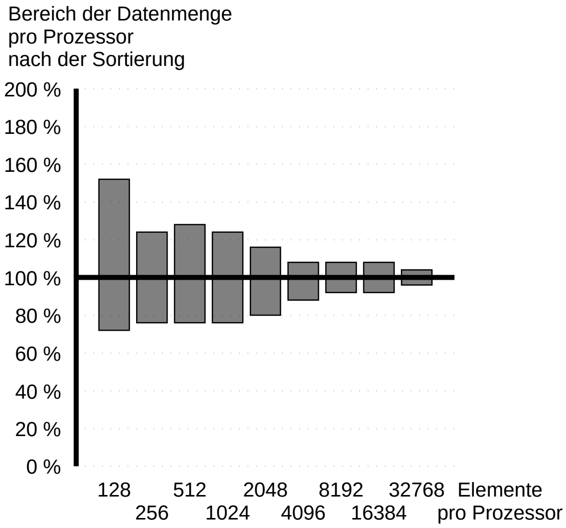
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1022	620	431	332	300	284	307	303	315
Gesamtzeit [s]	0.13	0.16	0.22	0.34	0.62	1.17	2.53	4.99	10.35
Zeit für Kommunikation [ms]	130	140	177	230	356	582	1250	2207	4414
Zeit für lokale Sortierung [ms]	19	30	69	140	319	670	1522	3181	6902
Minimale Anzahl Elemente	83	179	407	863	1848	3843	7801	15788	32010
Maximale Anzahl Elemente	178	319	629	1155	2262	4422	8688	16941	33477
Anzahl Datenpakete	5120	5120	5120	5120	5120	5816	9875	16445	29114
Gesamtweglänge aller Datenpaket	26624	26624	26624	26624	26624	27328	47330	77489	134747
Durchschn. Iterationstiefe	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00

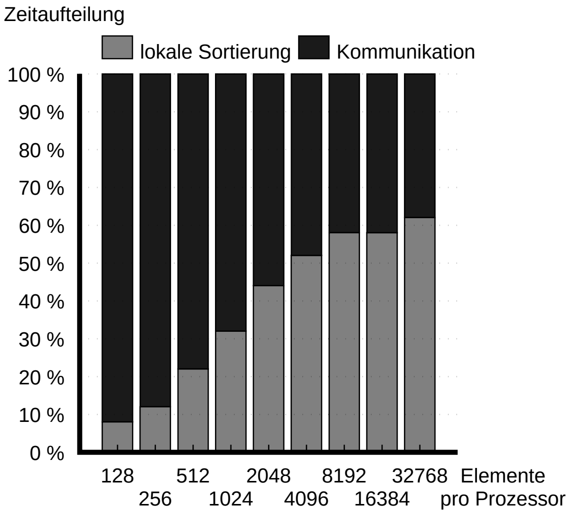
Datenblatt 71

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 256 Prozessoren		
Anzahl Pivotelemente	15	**Prozessornumerierung**	Shuffle-Numerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein





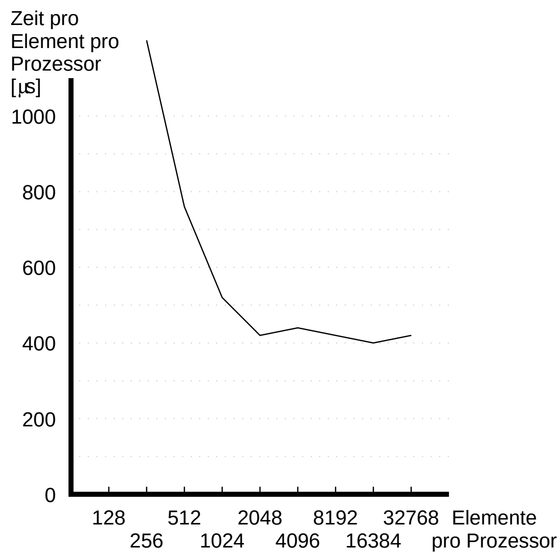


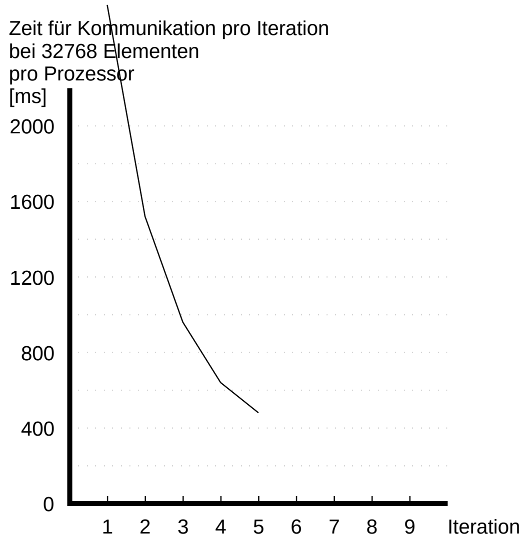


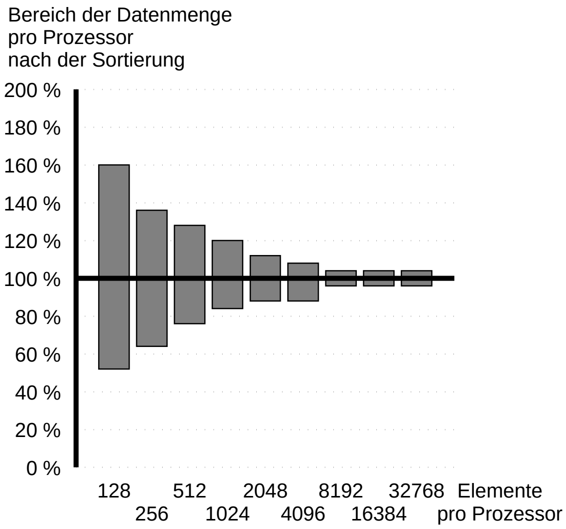
Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	1569	933	583	400	329	293	292	306	313
Gesamtzeit [s]	0.20	0.24	0.30	0.41	0.67	1.21	2.40	5.03	10.28
Zeit für Kommunikation [ms]	200	219	250	295	411	620	1125	2244	4341
Zeit für lokale Sortierung [ms]	19	30	69	142	319	670	1526	3181	6902
Minimale Anzahl Elemente	92	190	386	774	1637	3616	7473	15284	31667
Maximale Anzahl Elemente	193	314	658	1259	2404	4503	8893	17407	34381
Anzahl Datenpakete	8192	8192	8192	8192	8192	8192	8192	14034	22888
Gesamtweglänge aller Datenpaket	51200	51200	51200	51200	51200	51200	51200	89284	144688
Durchschn. Iterationstiefe	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00

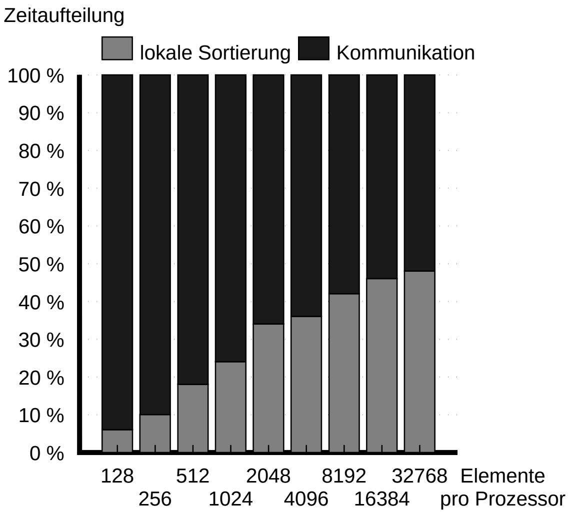
Datenblatt 72

Quicksort	mit lokaler Sortierung am Anfang		
Rechner	GCel mit 1024 Prozessoren		
Anzahl Pivotelemente	3	**Prozessornumerierung**	Shuffle-Numerierung
Pivotstrategie	Multi-Median der Pivotkandidaten		
reduzierte Kommunikation	Ja	**exakte Teilung**	Ja
exakte Partnerprozessoren	Ja	**Datentauschinvertierung**	Nein









Datenmenge	128	256	512	1024	2048	4096	8192	16384	32768
Zeit pro Element pro Prozessor [μs]	2093	1210	759	527	427	445	421	402	412
Gesamtzeit [s]	0.27	0.31	0.39	0.54	0.88	1.83	3.45	6.60	13.53
Zeit für Kommunikation [ms]	269	290	340	422	616	1247	2182	3819	7595
Zeit für lokale Sortierung [ms]	19	30	75	140	323	703	1526	3223	7123
Minimale Anzahl Elemente	66	165	392	878	1803	3644	7742	15581	31892
Maximale Anzahl Elemente	204	346	636	1221	2279	4457	8657	17036	33889
Anzahl Datenpakete	20480	20480	20480	20480	20480	34894	56920	99435	182799
Gesamtweglänge aller Datenpaket	126976	126976	126976	126976	126976	221504	359614	622514	1137188
Durchschn. Iterationstiefe	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00