technische universität
dortmund

Bachelorarbeit

Schnittmengenberechnung auf Moderner Hardware (Benchmarking)

Fabian Schulte
August 2014

Gutachter:
Prof. Dr. Jens Teubner
M. Sc. Sebastian Breß

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Datenbanken und
Informationssysteme (LS IV)
http://dbis.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

In diesem Kapitel wird eine kurze Einleitung in die Thematik dieser Bachelorarbeit gegeben. Dazu wird der Hintergrund vorgestellt und die Arbeit motiviert. Anschließend werden die Rahmenbedingungen dieser Arbeit erläutert und die Ziele der Arbeit herausgearbeitet. Abschließend wird auf die Struktur der Arbeit eingegangen.

1.1 Hintergrund

Um aus größeren Datenmengen Informationen zu extrahieren werden an Datenbanken Anfragen gestellt. Dabei ist die Berechnung von Schnittmengen eine zentrale Operation. Die Schnittmenge wird dabei aus zwei Tupelmengen erstellt, die sich aus zwei Teilanfragen ergeben haben. Die Bildung dieser Schnittmengen kann dabei typischerweise auf den Schnitt von zwei Listen von *record ids (RIDs)* zurückgeführt werden. Als Beispiel folgende SQL-Anfrage:

```
SELECT *
FROM R
WHERE R.a > 42 AND R.b > 17
```

Diese Anfrage kann ausgewertet werden, indem zwei Listen konstruiert werden, die die Tupel-Identifikatoren (RIDs) enthalten, welche die jeweilige Selektionsbedingung, also R.a $>$ 42 bzw. R.b $>$ 17, erfüllen. Die erste Liste enthält damit alle Einträge aus R, welche die Bedingung R.a $>$ 42 erfüllen, die zweite Liste alle Einträge aus R, die die Bedingung R.b $>$ 17 ebenfalls erfüllen. Die Schnittmenge dieser beiden Listen besteht aus der Menge an RIDs, deren korrespondierenden Einträge in R die Bedingungen R.a $>$ 42 und R.b $>$ 17 erfüllen. Im Bereich der OLAP-Systeme (OLAP steht für **O**n-**L**ine **A**nalytical **P**rocessing [Far11, S. 23]) und im Bereich des Information Retrieval findet sich das prinzipiell gleiche Problem. Statt wie im Bereich der Kern-Datenbanktechnologie auch auf die in klassischen Datenbanksystemen üblichen Listen von Tupel-Identifikatoren (*rid lists*) zu setzen, werden in OLAP-Systemen auch Bitvektoren eingesetzt und im Bereich des Information Retrieval

finden sich *posting lists* (mit Einträgen, die Dokumente referenzieren).
Diese in den unterschiedlichen Anwendungsbereichen spezifischen Darstellungen und Herangehensweise sollen und müssen in der Entwicklung des Benchmarks Beachtung finden. Dazu müssen die verschiedenen Anwendungsszenarien durch den Benchmark möglichst realistisch simuliert werden können. Nur so können die verschiedenen Techniken und die Implementierungen aus der parallelen Bachelorarbeit hinreichend bewertet werden.
Moderne Hardware ist Energie beschränkt („power wall“ [BBHS14]). Dies führt dazu, dass moderne Hardware immer heterogener wird. Computerprozessoren nutzen verschiedene Eigenschaften um die Performance zu steigern. Moderne CPUs nutzen mehrere Kerne, sowie die Möglichkeit mehrere Threads auf einem Kern auszuführen. Darüber hinaus werden SIMD-Befehle (**S**ingle **I**nstruction **M**ultiple **D**ata) unterstützt [KKS+08, S. 441].

1.2 Motivation

Die Berechnung von Schnittmengen ist im Query Processing eine zentrale Operation. Die effektive Ausnutzung von Fähigkeiten moderner Hardware soll Anfragen in Datenbanksystemen schneller machen. Eine effektivere Ausnutzung der Kapazitäten von moderner Hardware ist in allen Belangen ein lohnenswertes Ziel. Effizient ausgenutzte Hardware spart Kosten ein, denn eine Anschaffung neuerer Hardware kann so aufgeschoben werden. Außerdem folgt aus der effizienten Nutzung der zur Verfügung stehenden Hardware Kapazitäten für weitere Anfragen, die so zusätzlich bearbeitet werden können.

1.3 Rahmenbedingungen

Dieser Bachelorarbeit fällt die Aufgabe zu, einen Benchmark zu entwickeln, der eine Evaluation der verschiedenen Verfahren zur Schnittmengenberechnung ermöglicht. In einer parallel laufenden Bachelorarbeit werden verschiedene Verfahren zur Schnittmengenberechnung konkret implementiert. Dabei liegt der Fokus der Implementierung auf der bestmöglichen Nutzung der Eigenschaften moderner Hardware.

1.4 Ziele der Arbeit

Ziel der Arbeit ist die Entwicklung eines Benchmarks, mit dem sich Erkenntnisse zu den implementierten Schnittmengenberechnungsverfahren gewinnen lassen. Die gewonnenen Messergebnisse sollen dazu in Bezug zur moderner Hardware beurteilt werden. Darüber hinaus sollen Testszenarien angewandt werden, welche einen Bezug zum realen Query Processing haben.
Die Evaluation der gängigen Techniken auf modernen CPUs ist dabei nur ein erster Schritt auf dem Weg zur vollständigen Ausnutzung von allen Kapazitäten innerhalb von Compu-

tersystemen.
Alle Ergebnisse des Benchmarks zielen darauf ab, die in den verschiedenen Anwendungsbereichen spezifischen Darstellungen und Herangehensweisen zu evaluieren. Mit den Benchmark-Ergebnissen soll dann eine anwendugsbereichsübergreifende Bewertung der verschiedenen Verfahren erfolgen, denn möglicherweise ist der Einsatz von anwendugsbereichsspezifischen Herangehensweisen auch für die Nutzung innerhalb anderer Anwendungsbereiche sinnvoll.

1.5 Aufbau der Arbeit

Nach dieser Einleitung in meine Arbeit über das Thema des Benchmarkings von Schnittmengenberechnung auf moderner Hardware werden in Kapitel 2 die Grundlagen gelegt, welche für diese Arbeit nötig sind. Dazu wird kurz die Schnittmengenberechnung erklärt. Danach folgen Grundlagen zu Datenbanksystemen und Anfragen im Zusammenhang mit Datenbanken. Im darauf folgenden Abschnitt werden Kernzusammenhänge über die Speicherhierarchie in Computersystemen und weitere Punkte betrachtet, die Einfluss auf Datenbankanfragen bzw. die Schnittmengenberechnung haben. Im Anschluss werden die verschiedenen Verfahren zur Schnittmengenberechnung vorgestellt. Zuerst wird eine skalare Implementierung, welche auf den Einsatz von SIMD Befehlen setzt, instruiert, gefolgt von einem Schnittmengenberechnungsverfahren, welches auf die Nutzung von Bitmaps setzt. Zuletzt folgen die Schnittmengenberechnungsverfahren, welche auf den Einsatz von hashing setzen (Hashintersection und Radix-Cluster). Weiter geht es mit den Wahrscheinlichkeitsverteilungen. Dazu werden die Gleichverteilung, die Normalverteilung sowie die Zipf-Verteilung näher betrachtet. Das Kapitel über die Grundlagen wird dann noch Abgerundet von Grundlagen über Messungen und Experimente.
Auf das Kapitel über die Grundlagen folgt Kapitel 3 mit den Ansätzen des Benchmarks. In dem Kapitel geht es um die grundlegenden Gedanken, die zu der Implementierung des Benchmarks führen. Dazu werden die Schnittmengenberechnungsverfahren angesprochen, welche im Benchmark zur Verfügung stehen. Dabei handelt es sich um Implementierungen aus der parallel zu dieser Arbeit laufenden Bachelorarbeit. Des weiteren werden die Variablen im Benchmark näher betrachtet. Danach wird auf die Generierung von RIDs eingegangen. Dabei wird beschrieben, wie sich die RIDs in Abhängigkeit der gewählten Variablen und gemäß den möglichen Verteilungen generieren lassen. Als Verteilungen stehen die Gleichverteilung, die Normalverteilung sowie die Zipf-Verteilung zur Verfügung. Im Anschluss wird die Laufzeitmessung im Benchmark näher betrachtet und der Input für die Schnittmengenberechnungsalgorithmen erläutert. Es folgt die Angabe über den Output des Benchmarks und zuletzt eine Beschreibung über den Ablauf des Benchmarks.
Im Folgenden wird die Evaluation der Schnittmengenberechnungsverfahren unter Nutzung des Benchmarks erfolgen. Hierzu werden die verschiedenen Algorithmen einzeln evaluiert

und anschließend im Gesamten zueinander verglichen. Im Anschluss folgt die Bewertung der Algorithmen. Zuletzt werden die Erkenntnisse der Evaluation genutzt, um wünschenswerte Anforderungen an zukünftige Hardware abzuleiten.
Abschließend folgt in der Zusammenfassung ein Fazit der Arbeit. Ergänzt wird dies mit einem Ausblick auf mögliche zukünftige Themen und Ereignisse hinsichtlich der Implementierung von Schnittmengenberechnungsverfahren auf moderner Hardware.

Kapitel 2

Grundlagen

In diesem Kapitel wird ein Überblick über die Grundlagen gelegt. Die Grundlagen sollen damit für alle Leser ausgearbeitet sein, sodass jegliches Wissen für die weiteren Kapitel der Arbeit vorhanden ist.
Beginnend werden die Grundlagen zu Schnittmengen gelegt und dabei auf die grundsätzliche mathematische Mengenlehre kurz eingegangen. Es folgen Grundlagen zu Datenbanken und verschiedene Algorithmen zur Schnittmengenberechnung. Im Anschluss daran folgt der Abschnitt über Wahrscheinlichkeitsverteilungen. Schließlich folgen noch Grundlagen über die im Benchmark frei verwendbaren Variablen und grundlegendes zur Thematik „Messen“ bzw. dem experimentellen Vorgehen des Benchmarks.

2.1 Schnittmengen

Mengen sind Zusammenfassungen von einzelnen, unterscheidbaren Elementen vgl. [AS13, S. 9]. Mengen können gut als Venn-Diagramme visualisieren werden [LS97, S. 36]. Dabei werden die Mengen selbst als Kreis dargestellt, einzelne Elemente als Punkt. Alternativ kann man die Elemente auch mit einem x o.ä. kennzeichnen. In Abbildung 2.1 ist die Menge als schraffierter Kreis zu sehen. Der Rand des Kreises gehört hierbei zur Menge. Die Symbole ♠ und ♣ stellen beispielhafte Elemente dar. ♠ gehören dabei nicht zur Menge, sie sind nicht Element der Menge (mathematisch geschrieben: $♠ \notin Menge$). ♣ symbolisieren Elemente der Menge (mathematisch: $♣ \in Menge$). Die Menge enthält die mit ♣ markierten Elemente. Ein Spezialfall einer Menge ist die leere Menge, beschrieben durch das Zeichen $\emptyset$. Die leere Menge enthält kein Element.
Schnittmengen werden aus 2 oder mehr Mengen gebildet. Die Schnittmenge zweier Mengen enthält genau die Elemente, die sowohl in Menge M enthalten sind, wie auch in Menge N ($M \cap N = \{Menge\ der\ Elemnte\ x \mid x \in M \wedge x \in N\}$) [AS13, S. 12], [LS97, S. 26]. In Abbildung 2.2 ist die Schnittmenge als diagonal gekreuzte Fläche schraffiert. Es ist zu

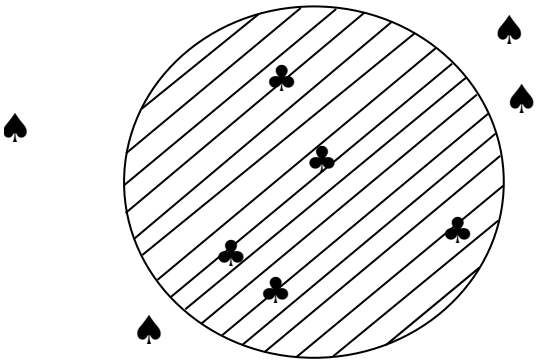

Abbildung 2.1: Schematische Darstellung einer Menge und einiger Elemente

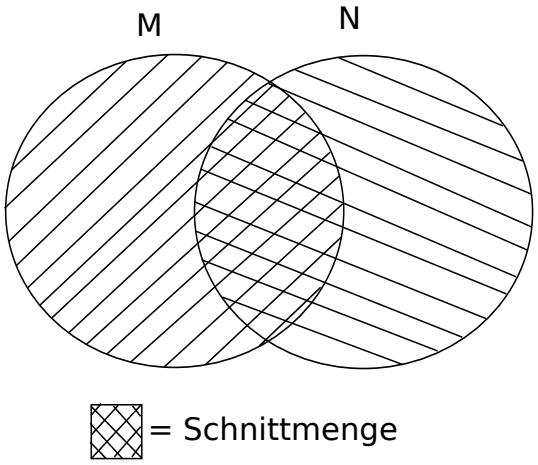

Abbildung 2.2: Schematische Darstellung zweier Mengen und deren Schnittmenge

erkennen, das Elemente in diesem Bereich sowohl $\in$ *Menge N* als auch in $\in$ *Menge M* liegen.

2.2 Datenbanken

Datenbanken dienen dem effizienten, widerspruchsfreien speichern und verwalten von elektronischen Daten [Sch07]. Das relationale Datenbankmodell ist der etablierte Standard für Datenbanken. Relationale Datenbanken basieren auf Relationen. Relationen sind dabei Beziehungen zwischen Elementen der Domäne der Attribute in der Relation. Dabei sind Relationen ausschließlich nur die Beziehungen, von denen klar ist, ob sie bestehen oder nicht. Daten innerhalb einer Relation in einer Datenbank stehen also in einer klaren Beziehung zueinander. Dieses strukturierte Speichern von Daten gemäß bestimmter Vorgaben ist es, was Datenbanken auszeichnet. Zu den Vorgaben gehört z. B., dass alle Einträge in eine Relation einen eindeutigen Schlüssel haben. Der Schlüssel darf sich nicht ändern und bezieht sich auf den Datensatz und nicht auf die Position in der Tabelle. Tabelle 2.1 zeigt Beispielhaft zwei Relationen, eine für Personen und eine für Städte. Relation „Personen“ umfasst dabei den Vor- und Nachnamen mit einem eindeutigen Schlüssel, genannt Personen-ID. Relation „Städte“ umfasst als Schlüssel die Postleitzahl (PLZ) sowie den Namen der Stadt und den Namen des zugehörigen Stadtteils. In einer dritten Relation könnte

Personen-ID	Nachname	Vorname
123	Mustermann	Max
234	Bond	James
345	Mustermann	Erika

PLZ	Stadt	Stadtteil
44149	Dortmund	Oespel
58454	Witten	Stockum
44227	Dortmund	Barop

Tabelle 2.1: Beispiel für Relation Person und Stadt

nun gespeichert werden, welche Person in welcher Stadt wohnt. Die Relation „Wohnort“ sähe dann wie folgt aus: Aus Tabelle 2.2 lässt sich nun z. B. entnehmen, dass die Person

Personen-ID	PLZ
123	44149
234	58454
345	44149

Tabelle 2.2: Beispiel für Relation Wohnort

mit der Personen-ID 2 in der Stadt mit der PLZ 58454 wohnt. Daraus folgt, dass James Bond in Witten Stockum lebt.
Existieren in einer Datenbank die Relation „Mitarbeiter“, die als Einträge alle Mitarbeiter einer Firma umfasst, und die Relation „Kunde“, die als Einträge alle Kunden der Firma umfasst, so ist es möglich, durch Schnittmengenberechnung herauszufinden, welche Mitarbeiter auch gleichzeitig Kunden ihrer Firma sind. Eine dazu passendes SQL-Statement würde lauten: SELECT * FROM Mitarbeiter INTERSECT SELECT * FROM Kunden. Damit SELECT * in dieser Anfrage funktioniert, muss das Schema der Relation „Mitarbeiter“ und das Schema der Relation „Kunde“ gleich sein.
Die Schnittmenge liefert auf die Frage, welcher Mitarbeiter auch Kunde der eigenen Firma ist, daher die richtige Antwort, da die Person sowohl als Mitarbeiter in der Relation „Mitarbeiter“ geführt wird, als auch in der Relation „Kunde“. Nur wenn die Person in beiden Relationen enthalten ist, liegt sie auch in der Schnittmenge der beiden Relationen.

2.3 Einfluss moderner Hardware auf Datenbankanfragen

In diesem Abschnitt wird kurz auf die Einflüsse der Architektur moderner Hardware auf Datenbankanfragen eingegangen. Das Verständnis der Kernzusammenhänge zwischen den einzelnen Komponenten moderner Hardware ermöglicht die Analyse von Messergebnissen im Zusammenhang.
In [MBK02] wird der Geschwindigkeitszuwachs von CPUs mit 70% pro Jahr, gegenüber 50% Geschwindigkeitszuwachs von DRAM (Hauptspeicher), als Hauptgrund ausgemacht, warum sich CPUs die meiste Zeit bei einer Anfrage im nicht-arbeitenden Zustand befinden. Während also die Geschwindigkeit der CPUs deutlich gewachsen ist, so ist der Speicherzugriff aus Sicht der CPU langsamer geworden. Die CPU bekommt also nicht so schnell Zugriff auf neue Daten, wie die CPU diese verarbeiten könnte. Um zu verdeutlichen, worin das Geschwindigkeitsproblem liegt, betrachtet man die Speicherhierarchie.
Innerhalb eines Computersystems sind verschiedene Speicher zu finden. Für die Speicher lässt sich eine Hierarchie angeben. Es gibt Speicher, auf den sehr schnell zugegriffen werden kann. Je schneller auf den Speicher zugegriffen werden kann, desto näher befindet er sich

an der CPU. Außerdem ist der CPU-nahe Speicher teurer und kleiner als der CPU-fernere Speicher. Abbildung 2.3 verdeutlicht die Hierarchie des Speichers. Würden alle Daten der

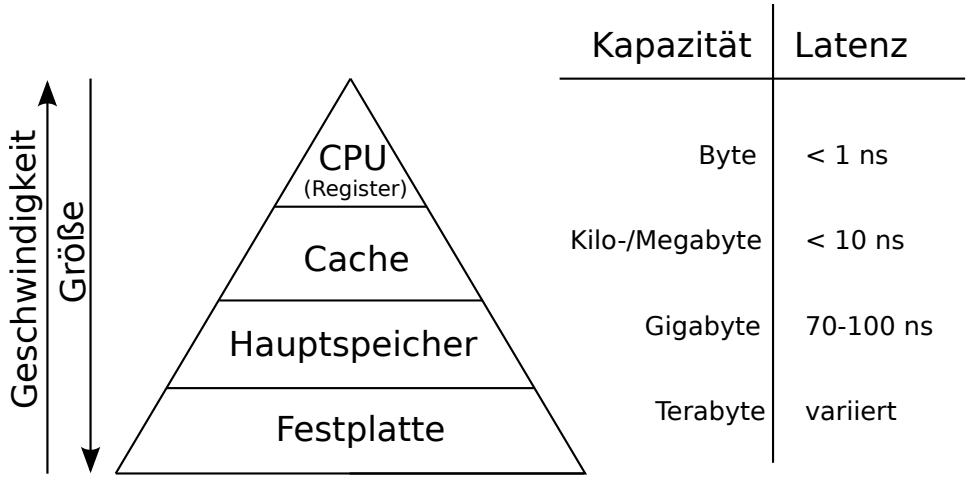

Abbildung 2.3: Hierarchie des Speichers in einem Computersystem

Datenbank in die Register der CPU passen, so wäre der Zugriff auf die Daten am schnellsten. Dazu ist der Speicher allerdings viel zu klein und auch zu teuer. Auch Cache-Speicher ist nur in einer geringen Größe vorhanden.
Für eine maximale Auslastung der CPU ist es daher nötig, die CPU schnell genug mit Daten zu Versorgen. Daraus ergibt sich nach [MBK02] die Forderung, Datenstrukturen und Algorithmen bezüglich Speicherzugriffen zu optimieren. Durch Optimierungen in diese Richtung könnten Datenbankanfragen schneller bearbeitet werden und das Potenzial moderner Hardware besser genutzt werden.

2.4 Verfahren zur Schnittmengenberechnung

Es werden verschiedene Verfahren zur Berechnung von Schnittmengen betrachtet. Die einzelnen Verfahren werden nachfolgend erörtert. Zuerst wird die skalare Schnittmengenberechnung betrachtet und unter Verwendung von SIMD-Befehlen erläutert. Nachfolgend wird auf die Bitmap-Schnittmengenberechnung eingegangen. In diesem Zusammenhang wird auch die WAH Kompression thematisiert. Es folgt noch die Schnittmengenberechnung auf Basis von Hash-Techniken und die Schnittmengenberechnung per Radix-Cluster. Alle zur Verfügung stehenden Schnittmengenberechnungsverfahren verarbeiten rid-Listen. Dabei steht rid für Record-ID und bezeichnet eine Zahl zur Identifikation eines Datenbankeintrags. Eine Record-ID ist eindeutig und eine ganzzahlige positive Zahl. Zu einer Record-ID gehört also nur immer ein Eintrag einer Relation einer Datenbank. Der Vergleich mit dem Schlüsselattribut eines Datenbankeintrags liegt nahe, eine Record-ID ist dieser auch durchaus ähnlich. Eine beispielhafte Relation könnte zwischen Mitarbeitern einer Firma und deren Wohnort wie in Tabelle 2.3 links aussehen. Dabei ist sowohl die

Personalnummer	PLZ
123	44149
234	58454
345	44149

Record-ID	Personalnummer	PLZ
1	123	44149
2	234	58454
3	345	44149

Tabelle 2.3: Beispiel für Relation Mitarbeiterwohnort, mit und ohne Record-IDs

Personalnummer Schlüssel der Relation Mitarbeiter, als auch die Postleitzahl Schlüssel für die Relation Wohnort. Mit Record-IDs sähe die Relation aus wie in Tabelle 2.3 rechts. Die rid-Liste wäre dann die Liste mit den Elementen 1, 2 und 3.

2.4.1 Skalare SIMD-Schnittmengenberechnung

Die skalare Schnittmengenberechnung mit Nutzung von SIMD Befehlssätzen basiert auf dem Paper „Fast Sorted-Set Intersection using SIMD Instructions“ von Benjamin Schlegel, Thomas Willhalm und Wolfgang Lehner ab [SWL11].
SIMD steht für **S**ingle **I**nstruction **M**ultiple **D**ata. SIMD unterstützt die schnelle Ausführung gleicher Befehle (Single Instruction) auf eine größere Menge eingehender Daten (Multiple Data).
Der in [SWL11] vorgestellte Algorithmus zur Schnittmengenberechnung setzt dabei auf die spekulative Ausführung. Das Hauptanliegen dabei ist gerade, mehr Vergleiche spekulativ voraus zu tätigen als Vergleiche von skalaren Algorithmen getätigt werden. Dabei wird auf *string and text processing new instructions* (STTNI) gesetzt. Diese sind Teil der Intel ® Streaming SIMD Extensions 4.2 (Intel ® SSE 4.2). Die Mengen zur Schnittmengenberechnung müssen dabei sortiert und frei von Duplikaten vorliegen. Diese Anforderung wird von rid-Listen erfüllt.
Skalare Schnittmengenberechnung errechnet aus zwei sortierten Mengen (A und B) die Schnittmenge. Dazu werden iterativ zwei Werte der Mengen A und B verglichen. Dabei wird mit den ersten Werten aus beiden Mengen gestartet. Sollte der Vergleich der beiden Werte ergeben, dass diese gleich sind, so wird der Wert in eine Ergebnismenge C geschrieben. Wenn die Werte von A und B gleich waren, so wird aus beiden Mengen jeweils der nächste Wert geladen und verglichen. Sollten die Werte von A und B nicht gleich sein, so wird nur aus der Menge der nächste Wert geladen, für den der aktuelle Wert kleiner ist als der aktuelle Wert aus der anderen Menge. Sind in Menge A sowie B keine weiteren Elemente mehr für einen Vergleich enthalten, so ist die Schnittmengenberechnung abgeschlossen. Abbildung 4.5 zeigt den Ablauf der skalaren Schnittmengenberechnung. Die durchgängigen Pfeile symbolisieren die Vergleiche, die Werte gefunden haben, die in die Schnittmenge einfließen. Gestrichelte Pfeile sind Vergleiche, die keinen korrespondierenden Wert gefunden haben. Die kleinen Zahlen an den Pfeilen zählen die Vergleichsoperationen in ihrer Durchführungsreihenfolge auf. Die berechnete Schnittmenge wäre nach dem Beispiel aus

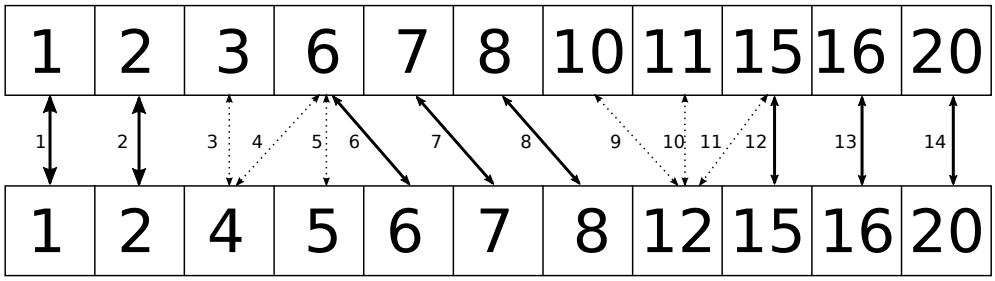

Abbildung 2.4: Schematische Abbildung des Ablaufes der Vergleichsoperationen während einer skalaren Schnittmengenberechnung

Abbildung 4.5 dann $\{1, 2, 6, 7, 8, 15, 16, 20\}$. Beide Beispielmengen enthalten 11 Elemente, 8 davon stimmten überein. Es wurden im Beispiel insgesamt 14 Vergleiche benötigt. Die Anzahl der Vergleiche betrüge im besten Fall $min\{l_A, l_B\}$ mit l_A Anzahl an Elementen in Menge A und l_B Anzahl an Elementen in Menge B. Im schlechtesten Falle betrüge die Anzal an Vergleichen $l_A + l_B - 1$ [SWL11, S. 3]. Durch die Nutzung von SIMD-Befehlen werden die Werte nicht mehr iterativ nacheinander verglichen, sondern Teile der Mengen komplett miteinander verglichen. Die Werte der Schnittmengen werden als Vektor hergenommen. Dann werden mit einem Befehl alle Werte dieser Vektoren miteinander Verglichen. Eine daraus resultierende Bitmaske indiziert die Werte, die in die Schnittmenge aufgenommen gehören. Abbildung 4.7 verdeutlicht dies. Die Bitmaske indiziert, dass der erste, der vierte,

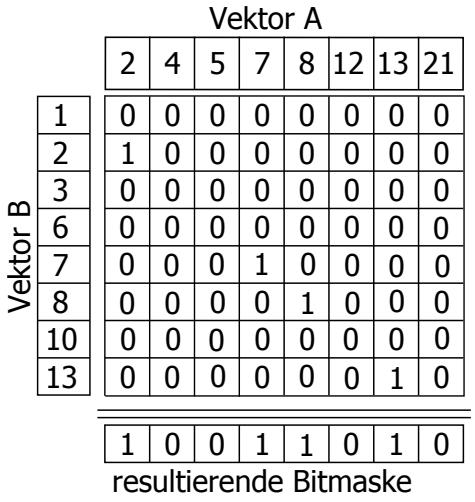

Abbildung 2.5: Vergleichsoperation mit SIMD-Befehl für Schnittmengenberechnung

der fünfte und der siebte Wert des Vektors A in die Schnittmenge gehört. Eine 1 wird nur dann gesetzt, wenn der Wert der Spalte (Vektor A) auch in einer Zeile (Vektor B) vorkommt. Die Anzahl der einzelnen Vergleiche in der Abbildung beträgt $8 * 8 = 64$ Vergleiche. Jedes Element des einen Vektors wird mit jedem Element des anderen Vektors verglichen.

2.4.2 Bitmap Intersection

Bitmap Intersection nutzt zur Schnittmengenberechnung Bitmaps. Bitmaps meinen dabei Bitmap Indizes, in dem jedes Bit zu einer RID korrespondiert [Joh99, S. 1]. Dadurch können Anfrageoperationen wie **and** und **or** direkt auf den Bitmaps ausgeführt werden. Die für jedes Bit auszuführende Operationen können sehr effizient auf moderner Hardware ausgeführt werden (↗ SIMD). Tabelle 2.4 zeigt ein Beispiel für eine Relation „Studenten“. Aus

Studenten

Matrikel-nummer	Nachname	Vorname	Studiengang	Semester	Geschlecht
12345	Mustermann	Max	Bachelor Informatik	5	m
23456	Doe	John	Bachelor Informatik	6	m
34567	Müller	Erika	Master Informatik	7	w
45678	Meier	Max	Master Informatik	7	m
56789	Smith	James	Bachelor Informatik	6	m

Tabelle 2.4: Beispiel für Relation Studenten

dieser würden dann die Bitmaps wie in Tabelle 2.5 folgen. Möchte man jetzt z. B. in einer

Studiengang

BSc Informatik	MSc Informatik
1	0
1	0
0	1
0	1
1	0

Semester

5	6	7
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0

Geschlecht

m	w
1	0
1	0
0	1
1	0
1	0

Tabelle 2.5: Beispiel Bitmap für Studiengang, Semester und Geschlecht

Anfrage herausfinden, welche männlichen Studenten im 6. Semester studieren, so müssen nur die Vektoren $Semester_6 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$ und $Geschlecht_m = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$ logisch verundet werden.

Der Ergebnisvektor wäre damit $Semester_6 \wedge Geschlecht_m = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \wedge \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$. Aus

dem Ergebnisvektor folgt, dass der zweite und der fünfte Eintrag der ursprünglichen Relation „Studenten“ männliche Studenten die im sechsten Semester studieren enthält. Dies wären der Student John Doe und der Student James Smith.
Auf diese Art und Weise funktioniert Bitmap Intersection. Damit Bitmap Intersection funktionieren kann, ist es nötig, die einzelnen Bitmaps vor zu halten bzw. zu erzeugen. Diese werden bei vielen unterschiedlichen Werten für einzelne Attribute der Relation sehr groß (auch Domäne genannt). Es werden viele 0-en in die Bitmap geschrieben. Kann das Attribut nur wenige Ausprägungen annehmen, so ergeben sich dementsprechend weniger Bitvektoren. Im Falle von nur 2 Werten reicht das Speichern eines Bitvektors aus, den der andere Bitvektor ist bloß der negierte Bitvektor des anderen. Abhängig von der Verteilung der Daten und der Anzahl an verschiedenen Ausprägungen der Relationsattribute können die Bitvektoren sehr viele 0-Einträge enthalten. Interessant sind aber eigentlich nur die 1-Werte. Da oftmals der Flaschenhals des gesamten Systems in der Bandbreite der Speicheranbindung an die CPU liegt ist, kann es sinnvoll sein die Vektoren zu komprimieren. Ein Verfahren zur Kompression von Bitmaps ist die WAH Kompression nach [WOS06, S. 2]. WAH steht für *W*ord-*A*ligned *H*ybrid. Die WAH Kompression ist ein auf Wörtern und nicht auf Byte basierende Kompression. Auf modernen Computern ist der Zugriff auf ein Byte in der Regel genauso Zeitaufwendig, wie der Zugriff auf ein Wort [HP07].
In den WAH komprimierten Daten gibt es zwei Arten von Wörtern. Einerseits gibt es *literal words*. Diese repräsentieren eine Bitmap 1 zu 1. Andererseits gibt es die *fill words*. Diese repräsentieren Bits der Bitmap, welche gleich sind und in der Bimap aufeinander folgen. Das erste Bit der Wörter dient der Auseinanderhaltung, so steht eine 0 für ein *literal word*, eine 1 für ein *fill word*. Das zweite Bit eines *fill words* repräsentiert das Füllbit, also gibt an, ob die folgenden Bits 0 oder 1 sind. Die restlichen Bits des Wortes repräsentieren die Länge des Füllens [WOS02]. Wenn ein Wort in einem Computer 32-bit lang ist, so muss die Länge des Füllens ein vielfaches von 31 Bit sein. Im Falle einer Länge von 64 Bit füllt ein *fill word* ein vielfaches von 63 Bit auf [WOS02]. Ein *literal word* repräsentiert $(Länge\ des\ Computer\ Worts) - 1$-viele Bits der Bitmap [WOS02].

2.4.3 Hashintersection

Hashintersection nutzt zur Schnittmengenberechnung eine Hash-Tabelle. Der Algorithmus kann in 2 Arbeitsphasen unterteilt werden. Der Algorithmus basiert auf den selben Grundgedanken wie Hash-Joins.
Zuerst wird von der kleineren Menge eine Hash-Tabelle angelegt (*build phase* oder auch *building phase*) [CBO14, S. 2], [Rah94, S. 351]. Danach wird geprüft, ob die Elemente der größeren Menge in der Hash-Tabelle liegen. Liegen diese in der Hash-Tabelle, so liegen diese in beiden Mengen vor, und können der zu berechnenden Schnittmenge hinzugefügt werden (*probe phase* oder *probing phase*) [CBO14, S. 2], [Rah94, S. 351]. Der Performance-

Vorteil bei Nutzung von Hashintersection liegt dabei darin, dass die Elemente der zweiten Relation schnell in der gehashten Menge gefunden werden können, sollten die Elementen in beiden Mengen vorkommen. Der Suchraum wird signifikant eingeschränkt [Rah94, S. 351]. Die Probe, ob ein Element vorhanden ist, ist in einer Hash-Tabelle weitaus schneller möglich als bei einem sequentiellen Scan der Menge. Außerdem funktioniert dieses Vorgehen auch ohne sortierte Daten als Input. Alle Werte der kleineren Menge werden über die Hash-Funktion in der Hash-Tabelle abgelegt und die Elemente der größeren Menge werden ebenfalls für den Lookup gehasht.
Vorteile der Hashintersection bestehen darin, dass keine Sortierung der Mengen nötig ist und das der Lookup nach korrespondierenden Werten schnell und effizient möglich ist. Nachteil gegenüber anderen Verfahren ist der Zuwachs an Aufwand durch das Erstellen der Hash-Tabelle und das Berechnen der Hash-Werte für die Elemente der größeren Menge.

2.4.4 Radix-Cluster Intersection

In [MBK02] wird der Radix-Cluster Algorithmus als Verbesserung der (partitionierenden) Hashintersection instruiert. Radix-Cluster zielen darauf ab, die zur Verfügung stehende Speicherbandbreite effizienter zu Nutzen und so den Flaschenhals der Speicherbandbreite zu umgehen. Optimierte Speicherzugriffe und optimierte Algorithmen führen zu einer höheren CPU Auslastung und sparen einiges an Kosten beim Speicherzugriffen ein. Die Performance mit Radix-Clustern wird nicht mehr von der Latenz des Speichers dominiert. Der Algorithmus zur Radix-Cluster Schnittmengenberechnung unterteilt die Schnittmengen in mehreren Schritten in H Cluster. Diese Unterteilung findet so lange statt, bis der Hash-Wert der Elemente nur noch vom niederwertigsten Bit abhängig ist. Ziel ist es, so oft Cluster zu bilden, bis eine effiziente Schnittmengenberechnung mit diesen einzelnen Clustern möglich ist. Im Vergleich zur Hashintersection findet vor der *build phase* und der *probe phase* noch die Phase der Partitionierung (Unterteilung in Cluster) statt.
Die Radix-Cluster Intersection verbindet die Vorteile der Hashintersection mit Optimierungen hinsichtlich dessen, das der Flaschenhals des Speicherzugriffs verringert wird. Eine effizientere Ausnutzung der Speicherbandbreite des Systems sollte Ausführungszeit der gesamten Schnittmengenberechnung verkürzen.

2.5 Wahrscheinlichkeitsverteilungen

So verschieden die Anwendungsgebiete der verschiedenen Datenbank-Techniken sind, so verschieden sind auch die Daten im System verteilt. Dabei lässt sich die Verteilung der Daten im System auf Wahrscheinlichkeitsverteilungen zurückführen. Wahrscheinlichkeitsverteilungen geben an, mit welcher Wahrscheinlichkeit ein Wert auftritt. Eine Wahrscheinlichkeitsfunktion liefert also zu jedem Wert die Wahrscheinlichkeit seines Auftretens gemäß

der Wahrscheinlichkeitsverteilung.
Wahrscheinlichkeitsverteilungen geben darüber Auskunft, mit welcher Wahrscheinlichkeit die jeweils möglichen Ereignisse eines Zufallsexperiments auftreten [KRES13, S. 121]. Im Zusammenhang mit dem Benchmark ist das Zufallsexperiment das zufällige Erstellen von Zahlen. Die Zufallszahl ergibt sich einfach aus dem Wert des auftretenden Ereignisses. Für die Ereignisse gilt, dass diese im Benchmark über einen Wertebereich angegeben sind, beispielsweise das Intervall von 0 bis 100. Dies bedeutet dann, dass die Zufallszahlen Werte im Bereich von einschließlich 0 bis einschließlich 100 annehmen können (Zufallszahlen $\in [0, 100]$). $[x, y]$ ist ein Intervall mit $x < y$ und steht für die Menge an Zahlen die größer oder gleich x sind und kleiner oder gleich y.
Zur Verdeutlichung noch als kurzes Beispiel der Münzwurf: Beim Münzwurf können die Ereignisse *Zahl* und *Kopf* auftreten. Die Wahrscheinlichkeit beträgt für beide üblicherweise $\frac{1}{2}$.
Für die Erstellung von Zufallszahlen im Rahmen des Benchmarks wird angestrebt, die Zufallszahlen gemäß verschiedener Wahrscheinlichkeitsverteilungen zu generieren. Umgesetzt werden sollen die Normalverteilung, die Gleichverteilung und die Verteilung nach dem Zipf'schen Gesetz (Zipf-Verteilung).
Bei Wahrscheinlichkeitsverteilungen unterscheidet man zwischen *diskreten* Wahrscheinlichkeitsverteilungen und *stetigen* Wahrscheinlichkeitsverteilungen. Bei diskrete Wahrscheinlichkeitsverteilungen können nur Wahrscheinlichkeiten von bestimmten, isolierten x-Werten annehmen. Ein diskrete Wahrscheinlichkeitsverteilung könnte zum Beispiel wiedergeben, wie hoch die Wahrscheinlichkeit dafür ist, dass 3 Studenten zu spät zur Vorlesung kommen. Die Wahrscheinlichkeit dafür, dass $3\frac{1}{2}$ Studenten zu spät kommen, kann hingegen nicht angegeben werden.
Stetige Wahrscheinlichkeitsverteilungen geben hingegen auch für Werte wie $x = 3,5$ oder $x = \frac{9}{32}$ die Wahrscheinlichkeit an.

2.5.1 Normalverteilung

Die Normalverteilung wird oft auch als Gauß-Kurve oder Gaußsche Glockenkurve benannt. Den Namen Glockenkurve trägt die Funktion aufgrund ihrer Form. Diese erinnert an eine Glocke, wie in der schematischen Abbildung 2.6 zu sehen. Normalverteilungen weisen einige charakteristische Eigenschaften auf. Normalverteilungen sind symmetrisch. Dadurch sind der Mittelwert, der Median und der Modus identisch und liegen in der Mitte der Verteilung. Sie teilen die Verteilung in zwei gleich große Hälften. Des weiteren liegen die meisten Werte einer Normalverteilung nah um den Mittelwert herum. Je weiter man sich dabei vom Mittelwert entfernt, desto weniger Werte findet man. Die letzte charakteristische Eigenschaft von Normalverteilungen ist, dass sich diese der x-Achse annähern, ohne sie jemals zu berühren [KRES13, S. 128]. Normalverteilungen sind durch zwei Größen ein-

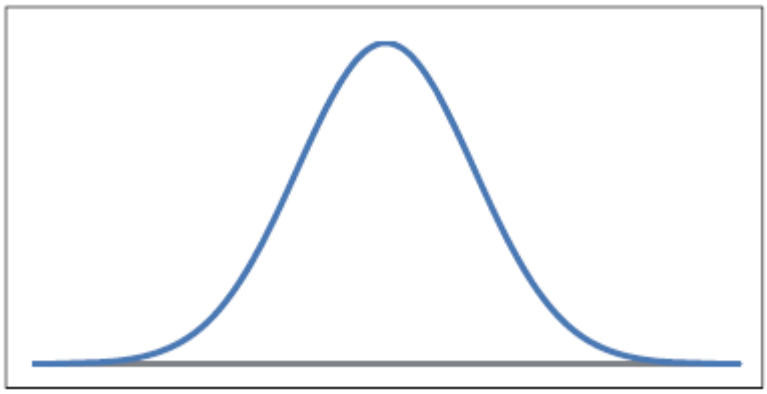

Abbildung 2.6: Schematische Abbildung einer Gauß-Kurve
Quelle: [KRES13, S. 127]

deutig bestimmt. Diese Größen sind der Mittelwert μ und die Standardabweichung σ. Es gibt überabzählbar viele verschiedene Normalverteilungen. In Abbildung 2.7 kann man die

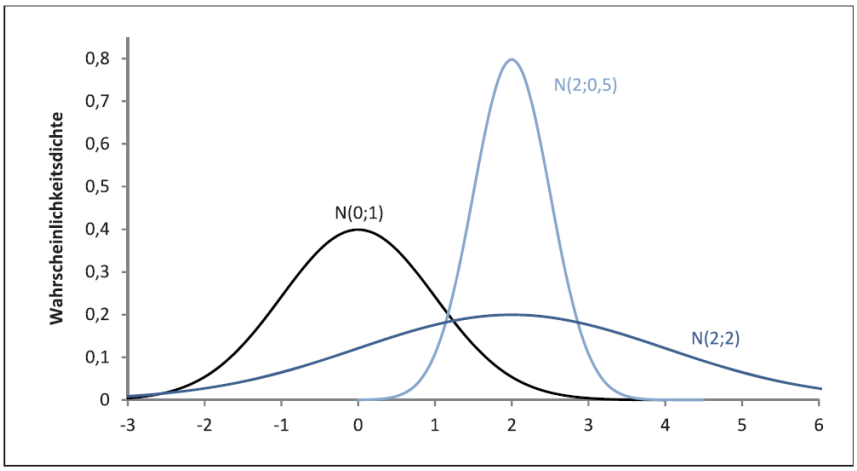

Abbildung 2.7: Gauß-Kurve mit verschiedenen Werten für μ und σ
Quelle: [KRES13, S. 127, Abb. 5-5]

Auswirkungen der Größen μ und σ erkennen. Die Schreibweise N(μ;σ) steht dabei für eine Normalverteilung mit Mittelwert μ und Standardabweichung σ. Es lässt sich gut erkennen, dass die Erhöhung des Werts für die Standardabweichung σ dazu führt, dass die Kurve „platt gedrückt“ wird. Der Wert für den Mittelwert μ hingegen verschiebt die Spitze der Kurve auf der x-Achse an die Stelle, an der gilt $x = \mu$.
Bei der Normalverteilung handelt es sich nicht um eine diskrete Verteilung, sondern um eine stetige Verteilung. Aus stetigen Verteilungen lässt sich nicht direkt die Wahrscheinlichkeit für ein einzelnes (diskretes) Ereignis ablesen [KRES13, S. 129].
Normalverteilte Zufallszahlen findet man zum Beispiel bei altersspezifischen Körpergewichten und Körpergrößen nach Geschlecht [KRES13, S. 133]. Eine Datenbank eines Kleidungsgeschäfts, die die Größe der Kunden enthält, kann so also normalverteilte Daten enthalten.

2.5.2 Gleichverteilung

Die Zufallszahlen für den Benchmark sollen gemäß der Verteilungen generiert werden. Bei der Gleichverteilung wird davon ausgegangen, dass die Wahrscheinlichkeit dafür, dass die Zufallszahl einen bestimmten Wert annimmt, immer gleich groß und damit gleich wahrscheinlich ist. Die Wahrscheinlichkeit P für das Ereignis (=Wert der Zufallsvariablen) x ergibt sich somit zu: $P(x) = \frac{1}{Anzahl\ an\ Experimenten}\ für\ \forall x \in\ der\ möglichen\ Ereignisse.$ Mathematisch ergibt sich für die Wahrscheinlichkeit: $P(X = x) = \frac{1}{n}\ für\ x \in \{1, \ldots, n\}$

2.5.3 Zipf-Verteilung

Die Zipf-Verteilung, nach George Kingsley Zipf leitet sich aus dem zipfschen Gesetzen ab. Zipf entwickelte als Linguistiker die zipfschen Gesetze, als er die Häufigkeit von Wörtern in unterschiedlichen Sprachen untersuchte. Dabei ordnete er die Wörter einer Sprache nach ihrer Häufigkeit an und erkannte, dass der Rang des Wortes einen Einfluss auf die tatsächliche Häufigkeit des Wortes hat [Cri09, S. 138 f.]. Für die deutsche Sprache ergibt sich folgende Tabelle für die 10 häufigsten Wörter:

Rang	1	2	3	4	5	6	7	8	9	10
Wort	der	die	und	in	den	von	zu	das	mit	sich

[Cri09, S. 138]

Dabei tritt das Wort „der“ ungefähr doppelt so häufig auf wie das Wort „die“ und dreimal so häufig wie das Wort „und“ [Cri09, S. 139]. Das Zipf'sche Gesetz ist ein empirisch bestimmtes Gesetz und besagt, dass die Häufigkeit eines Wortes gegeben ist durch $\frac{k}{r^\alpha}$, wobei r der Rang des Wortes ist [Cri09, S. 139]. Dabei ist k eine Konstante, die allein abhängig vom Wortschatz des Autors eines Textes ist [Cri09, S. 139]. k ergibt sich somit aus der Anzahl an möglichen Ausprägungen für Wörter bzw. Zufallszahlen. Der Exponent α wurde erst später hinzugefügt. Mit $\alpha = 1$ wird also die ursprüngliche Zipf-Verteilung beschrieben. Die Zipf-Verteilung ist eine diskrete Wahrscheinlichkeitsverteilung.
Es lassen sich nicht nur die Häufigkeiten von Wörtern auf eine Zipf-Verteilung zurückführen. Zipf zeigte auch, dass die Rangverteilung der Einwohnerzahlen von amerikanischen Städten dem Zipf'schen Gesetz folgt [Zip41]. Auch Zugriffe im Web folgen dem Zipf'schen Gesetz [CBC95]. Zipf-verteilte Daten finden sich also an vielen Stellen wieder. Zipf-verteilte Daten haben die Eigenschaft, dass einige Ausprägungen mit hoher Wahrscheinlichkeit auftreten, viele Ausprägungen aber mit einer sehr kleinen Wahrscheinlichkeit. Für die Ausprägungen mit kleiner Wahrscheinlichkeit gilt, dass die Wahrscheinlichkeit für das Vorkommen sehr gering ist, nimmt man aber die Ausprägungen mit kleinen Wahrscheinlichkeiten zusammen, so ist die Wahrscheinlichkeit sehr hoch, dass ein Wert solcher Ausprägung auftritt.

2.6 Vorgehen bei Messungen oder Benchmark als Experiment

Im Folgenden werden die Grundlagen für experimentelle Messungen mit Computersystemen gelegt.
Experimentum (lateinisch) bedeutet so viel wie Probe oder Versuch. [Sar90] definiert ein Experiment wie folgt:

> „Unter einem Experiment versteht man einen systematischen Beobachtungsvorgang, auf Grund dessen der Untersucher das jeweils interessierende Phänomen erzeugt sowie variiert und dabei gleichzeitig systematische und/oder unsystematische Störfaktoren durch hierfür geeignete Techniken ausschaltet bzw. kontrolliert.“ [Sar90]

Der Benchmark probiert unterschiedliche Mengen für die Schnittmengenberechnung aus und simuliert damit verschiedene Datenverteilungen. Dadurch sollen Erkenntnisse darüber gewonnen werden, welche Veränderung von welchen Parameter (unabhängige Variablen, s. u. in Abschnitt 3.2) welche Auswirkungen auf die Schnittmengenberechnung haben.
Alle Ergebnisse des Benchmarks sollen Reproduzierbar sein. Jeder, der möchte, soll dazu in der Lage sein, den Benchmark für sich zu einer anderen Zeit und unter anderen Rahmenbedingungen (bzgl. Hardware) zu wiederholen. Der Benchmark soll dabei sehr ähnliche Erkenntnisse liefern. Abbildung 2.8 zeigt die Phasen für die Durchführung eines solchen

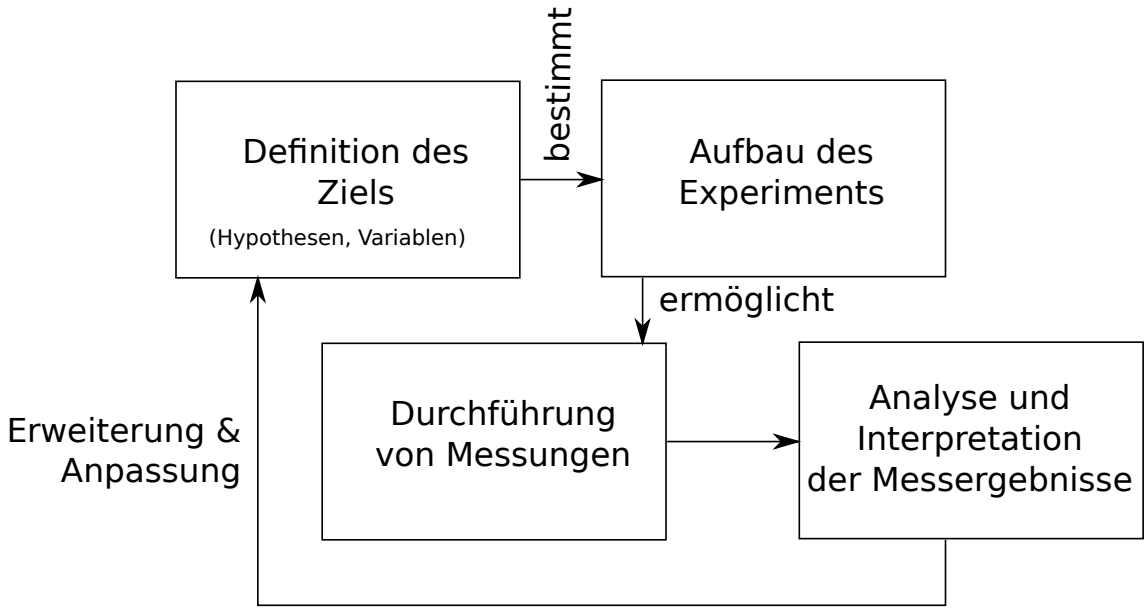

Abbildung 2.8: Phasen während eines Experiments

Experiments. Zuerst wird das Ziel, also was der Benchmark simulieren und messen soll, festgelegt. Dazu werden Hypothesen aufgestellt, die die Gedanken zu den abhängigen und

unabhängigen Variablen festhalten. Die Gedanken der Hypothesen sollten dabei auf Erwartungen über das Ergebnis des Benchmarks, unter dieser Variablenbelegung, basieren. Der Aufbau des Experiments ist der Benchmark und wird im Kapitel 3 näher betrachtet. Auch auf die Phase der Ausführung und der Erhebung von Daten aus dem Benchmark-Durchlauf wird in Kapitel 3 näher eingegangen. Die Analyse und Interpretation der Daten erfolgt dann in Kapitel 4.
Die Werte für die Variablen werden bei den verschiedenen Aufrufen in *abhängige* und *unabhängige Variablen* unterteilt.

2.6.1 Unabhängige Variablen

> „Die unabhängige Variable wird vom Experimentleiter absichtsvoll und geplant variiert, um eine Reaktion der abhängigen Variable zu bewirken.“ [KST09, S. 534]

Die unabhängigen Variablen sind die Variablenwerte, die verändert werden. Diese beeinflussen die abhängigen Variablen bzw. das Ergebnis der Messung.
Es empfiehlt sich, immer nur eine unabhängige Variable zu manipulieren, um die Auswirkung auf die abhängigen Variable besser zuordnen zu können. Ändert man mehrere unabhängige Variablen auf einmal, so kann man nicht mehr direkt den Einfluss einer unabhängigen auf eine abhängige Variable bzw. die abhängigen Variablen beobachten.

2.6.2 Abhängige Variablen

> „Die Reaktion der abhängigen Variable auf das geplante Variieren der unabhängigen Variablen wird beobachtet.“ [KST09, S. 534]

Abhängige Variablen sind von den unabhängigen Variablen abhängig. Eine Änderung der unabhängigen Variablen beeinflusst die abhängigen Variablen des Experiments. Die Veränderung der Selektivität von rid-Listen hat so zum Beispiel auch Einfluss auf die Trefferquote bei der Schnittmengenberechnung. Trefferquote meint hierbei die Quote, die angibt ob bei einem Vergleich im Prozess der Schnittmengenberechnung ein übereinstimmendes Paar aus den beiden Schnittmengen gefunden wurde oder nicht. Die Trefferquote der Schnittmengenberechnung ist also eine abhängige Variable.
Die Veränderung von unabhängigen Variablen sollte daher so gewählt werden, dass der Einfluss auf abhängige Variablen bedacht ist. Der Einfluss von unabhängigen Variablen auf abhängige Variablen ist nicht immer direkt zu erkennen. Wird dieser Einfluss nicht erkannt, so würde ein im Ergebnis der Messung erkannter Effekt fälschlicherweise nur der unabhängigen Variablen zu geschrieben, dabei zeichnet sich eigentlich die abhängige Variable für diesen Effekt verantwortlich. Konstruiert man Testmengen für die Schnittmengenberechnung nur nach der Selektivität, so weiß man im Voraus nicht, welche Trefferquote

die Schnittmengen aufweisen werden. Ein bei der Messung erkannter Effekt kann damit nicht der Selektivität zweifelsfrei zugeordnet werden, denn die Trefferquote änderte sich ebenfalls.

2.6.3 Störvariablen

> „Als Störvariable wird eine Variable bezeichnet, die den Einfluss der unabhängigen auf die abhängige Variable verfälscht.“ [KST09, S. 534]

Unter den Störvariablen fasst man Variablen zusammen, die Einfluss auf den Ausgang der Messung nehmen können. Diese stören also das Experiment. Störvariablen sind nicht unbedingt kontrollierbar, sollten aber wenn möglich weitestgehend ausgeschlossen werden. Auf die Messungen störend einwirken können, im Rahmen von experimentellen Messungen an einem Computersystem, z. B. Hintergrundprozesse oder System-Interrupts. Außerdem hat die Raumtemperatur einen direkten Einfluss auf die Kühlung des Computersystems. Die Temperatur der CPU beeinflusst dabei die Leistungsfähigkeit der CPU. Der Einfluss auf diese Störvariablen ist eher gering und schwer auszuschließen.

2.6.4 Median und arithmetisches Mittel

Um den Einfluss von Messfehlern und Abweichungen in der Messung möglichst gering zu halten wird die Schnittmengenberechnung gleich mehrfach durchgeführt. Aus den dann gemessenen Werten kann man entweder das arithmetische Mittel oder den Median nehmen. Das **arithmetische Mittel** ist der Durchschnitt. Ausgerechnet wird das arithmetische Mittel wie folgt: $\bar{x}_{\text{arithm}} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1+x_2+\cdots+x_n}{n}$.
Der **Median** hingegen erfordert der größe nach sortierte Werte und wählt aus den sortierten Werten den in der Mitte stehenden aus bzw. nimmt das arithmetische Mittel der beiden mittleren Werte. Mathematisch ergibt sich damit:

$$\bar{x}_{\text{med}} = \begin{cases} x_{\left(\frac{n+1}{2}\right)}, & n \text{ ungerade,} \\ \frac{1}{2}\left(x_{\left(\frac{n}{2}\right)} + x_{\left(\frac{n}{2}+1\right)}\right), & n \text{ gerade.} \end{cases}$$

Der Median hat gegenüber dem arithmetischen Mittel den Vorteil, dass Ausreißer in der Menge der Messergebnisse keine große Wirkung haben. Ein kurzes Beispiel: Die Zahlen sind $1, 2, 2, 3, 100$. Man sieht sofort, dass der Wert 100 deutlich größer ist als die anderen Werte. Es ergibt sich für den Median $\bar{x}_{\text{med}} = 2$, hingegen beträgt der Durchschnitt $\bar{x}_{\text{arithm}} = 21,6$.

2.6.5 Hypothesen

Hypothesen sind die schriftliche Fixierung von spezifischen Messszenarien. In einer Hypothese werden Erwartungen bzgl. der Auswirkungen von Änderungen der Variablen auf

den Ausgang des Experiments (Laufzeitmessung der Schnittmengenberechnungsverfahren) festgehalten. Die Hypothese umfasst begründete Überlegungen darüber, welchen Effekt eine Änderung einer oder mehrerer Variablen hat. Es empfiehlt sich möglichst nur eine Variable zu Ändern um damit Seiteneffekte auf abhängige Variablen zu vermeiden. Eine Änderung mehrerer Variablen in Kombination sollte erst dann geschehen, wenn der Einfluss einer Variablen ausreichend erkannt wurde. Einflüsse aus Wechselwirkung durch Änderung mehreren Variablen können möglicherweise schwerer erkannt werden.

2.7 Zusammenfassung

In diesem Kapitel wurden die Grundlagen für das Verständnis der weiteren Ausführungen dieser Arbeit gelegt. Von Grundbegriffen wie Schnittmenge und Datenbanken wurde über Einfluss moderner Hardware auf Datenbankanfragen hin zu verschiedenen Verfahren für die Berechnung von Schnittmengen und der Thematik der Wahrscheinlichkeitsverteilungen sowie der Thematik des Experiments alle relevanten Themen dargestellt.

Kapitel 3

Aufbau des Benchmarks

In diesem Kapitel soll es um die für den Benchmark verwendeten Ansätze gehen. Hierbei werden die wichtigsten grundlegenden Überlegungen und Gedanken abgehandelt. Zuerst werden die Variablen näher betrachtet. Dabei wird auf die unabhängigen und abhängigen Variablen näher eingegangen. Danach folgt die Generierung von Testdatenmengen. Dazu wird auf die Generierung gemäß Verteilung in Kombination mit Trefferquote und Selektivität näher eingegangen. Die Generierung von gleichverteilten, normalverteilten und Zipfverteilten RIDs wird dazu erläutert. Im Anschluss wird die Zeitmessung näher betrachtet. Dazu wird darauf eingegangen, was bei welchem Algorithmus gemessen wird. Außerdem wird erläutert, wie gemessen wird. Input und Output des Benchmarks werden danach eingehender betrachtet. Dabei wird auf die Parameter für den Aufruf des Benchmarks eingegangen, sowie auf den Output der vom Benchmark produziert wird (Messergebnisse und rid-Listen). Abschließend wird der Ablauf des Benchmarks näher betrachtet. Dabei geht es um die einzelnen Arbeitsschritte, welche der Benchmark umfasst.

3.1 Algorithmen zur Schnittmengenberechnung

In Abschnitt 2.4 wurden die verschiedenen Verfahren zur Schnittmengenberechnung bereits erläutert. Der Benchmark bietet für seine Messungen die Verwendung des SIMD-Verfahrens, des Radix-Cluster Algorithmus und der Bitmap Intersection mit und ohne WAH Kompression an. Die WAH Kompression wird mit einer Wort-Länge von 32 und 128 Bit betrachtet. Die Implementierungen stammen aus der parallel laufenden Bachelorarbeit. Außerdem ist es möglich, zur Schnittmengenberechnung auf die Standard-C++ Implementation für Schnittmengenberechnung, std::set_intersection, als Schnittmengenberechnungsmethode zurück zu greifen.

3.2 Variablen im Benchmark

Im Folgenden werden die zur Evaluation der verschiedenen Schnittmengenberechnungsverfahren unabhängigen Variablen näher betrachtet.
Alle Verfahren zur Schnittmengenberechnung schneiden zwei rid-Listen. Die Länge der rid-Liste, also die Anzahl an RIDs in einer rid-Liste, lässt sich verändern. Mehr Elemente in den Schnittmengen bedeuten mehr Aufwand, da zwischen mehr Elementen übereinstimmende Paare gefunden werden müssen. Außerdem sollen kleine mit großen Mengen geschnitten werden können, möglicherweise hat das Verhältnis der Schnittmengengröße zueinander einen negativen oder positiven Einfluss auf die Ausführungszeit der Schnittmengenberechnungsverfahren.
Die Angabe der Anzahl an Wiederholungen der Schnittmengenberechnung ist eher uninteressant. Eine größere Anzahl an Wiederholungen sollte das Messergebnis präzisieren und dem negativen Einfluss von Ausreißern entgegenwirken. Eine steigende Anzahl an Wiederholungen verlängert dabei die Gesamtausführungszeit des Benchmarks.
Außerdem kann die Selektivität beeinflusst werden. Selektivität bestimmt den Anteil der Datensätze, die die Selektionsbedingung erfüllen erfüllen. Eine Selektivität von 100% bedeutet also, dass kein Wert die Selektionsbedingung verletzt hat. Für rid-Listen bedeutet dies, dass die Liste durchgängig alle RIDs enthält. Ist die erste RID gleich 1 und die letzte ID gleich 10, so ist die rid-Liste gleich $1, 2, 3, 4, 5, 6, 7, 8, 10$. Bei einer Selektivität von 50% bedeutet dies, dass die Wahrscheinlichkeit dafür, dass die nächste RID geleich die vorherige ID+1 ist 50% beträgt. Es wird jeder zweite Wert für eine RID gsetzt. Die vorherige rid-Liste sähe dann z. B. so aus: $1, 3, 5, 7, 9$. Die Abbildung einer rid-Liste auf eine Bitmap ist dabei wesentlich von der Selektivität abhängig. Während bei einer Selektivität von 100% alle Bits der Bitmap auf 1 gesetzt werden, werden bei einer Selektivität von 0% alle auf 0 gesetzt. Eine Selektivität von 50% würde dazu führen, dass die Hälfte der Bits auf 1 gesetzt wird und die andere Hälfte auf 0. Die Selektivität hat dabei einen Einfluss auf die abhängige Variable der Trefferquote der Schnittmengenberechnung. Trefferquote der Schnittmengenberechnung meint die Anzahl der Elemente, die in der Schnittmenge liegen. Das Beispiel in Abbildung 3.1 zeigt den Zusammenhang von Selektivität und Trefferquote. Die drei rid-Listen sind alle nach Selektion aus einer Relation mit den Einträgen 1 bis 12 entstanden. Die rid-Listen 1 bis 3 haben jeweils eine Selektivität von 50% zu der ursprünglichen Relation mit den Record-IDs 1 bis 12. Obwohl alle Listen eine Selektivität von 50% aufweisen, beträgt die Trefferquote der Schnittmengenberechnung im Falle der Schnittmenge von rid-Liste 1 mit rid-Liste 2 0%. Im Falle der Schnittmenge von rid-Liste 2 mit rid-Liste 3 beträgt die Trefferquote 50%. Würde man rid-Liste 1, 2 oder 3 mit sich selber schneiden, so beträge die Trefferquote je 100%.
Die Trefferquote wird also direkt durch die Selektivität beeinflusst. Die Trefferquote ist somit abhängig von der Selektivität. Zu Beachten ist: Die Selektivität sagt im Gegenzug

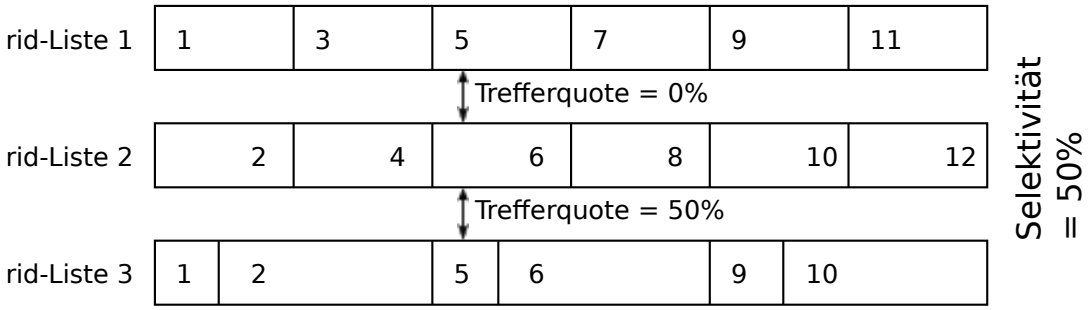

Abbildung 3.1: Beispiel für den Zusammenhang zwischen Selektivität und Trefferquote der Schnittmengenberechnung

nichts über die Trefferquote der Schnittmengenberechnung aus.
Einfluss auf die Trefferquote hat die Verteilung der Record-IDs in der rid-Liste gemäß der Wahrscheinlichkeitsverteilungen (siehe 2.5). Sind die Record-IDs gleichmäßig verteilt in der rid-Liste, so treten in relativ gleichmäßigen Abständen Lücken auf. Sind die Record-IDs hingegen einer Normalverteilung entsprechend in der rid-Liste, so sind in der Schnittmenge in einem Bereich sehr viele Record-IDs. Record-IDs mit weitaus höheren oder niedrigeren Werten kommen dann aber nicht bzw. kaum vor. Dies hat wiederum einen Einfluss auf die abhängige Variable der Trefferquote der Schnittmengenberechnung. Man stelle sich zwei rid-Listen vor, die eine hat Record-IDs aus dem unteren Bereich der möglichen Records, die andere rid-Liste hat nur Record-IDs aus dem oberen Bereich. Das Ergebnis ist, das die Trefferquote 0% beträgt, zumindest wenn die höchste Record-ID der ersten rid-Liste kleiner ist als die Record-ID der zweiten rid-Liste. Das sich Record-IDs nach einer Selektion um einen bestimmten Wert herum verstärkt auftreten könnte z. B. dann der Fall sein, wenn eine Relation gefüllt mit Einkäufen betrachtet wird. Wenn aus der Relation die Einkäufe heraus selektiert werden, bei denen Streusalz enthalten ist, so ist es durchaus Wahrscheinlich, dass sich viele Einträge mit Streusalz häufen. An Wintertagen mit dem ersten Schnee ist eine Anhäufung von Einkäufen mit Streusalz zu erwarten. Außerdem können sich Einkäufe mit bestimmten Artikeln auch auf Grund von Angeboten häufen, oder aber auch zu bestimmten Ereignissen. Das erste gute Wetter im Jahr führt so vielleicht zum Anstieg von Käufen mit Grillfleisch, zum Jahreswechsel steigt die Anzahl an Käufen mit Sekt an. Im Vergleich dazu ist es sehr wahrscheinlich, dass die Record-IDs von Einkäufen die Brot enthalten sehr gleichmäßig verteilt sind. Daraus ergeben sich Muster für die Verteilung der Record-IDs. Mal werden es wenig Record-IDs sein, gefolgt von einer kurzen starken Häufung von Record-IDs, mal werden die Record-IDs mit gleichmäßigen Abständen im Wechsel häufig und selten auftreten.
Bei der Wahl von Testmengen zur Schnittmengenberechnung sind also einige Variablen zu beachten. Die abhängigen Variablen müssen dabei immer Beachtet werden, falls eine unabhängige Variable geändert wird, welche Einfluss auf die abhängige Variable hat.

Die in Abschnitt 2.5 angesprochenen Parameter der Verteilungen bilden für den Benchmarks weitere unabhängige Variablen. Auch von diesen ist die Trefferquote der Schnittmengenberechnung abhängig. Die Verteilung von Record-IDs gemäß der Wahrscheinlichkeitsverteilungen zieht den wie oben angesprochenen Effekt bzgl. der Selektivität und damit der Trefferquote nach sich.

3.3 Generierung von Testmengen

Um die Schnittmengenberechnungsverfahren zu Benchmarken braucht es für eine Messung 2 Mengen. Bei der Generierung der Testmengen sollen Mengen generiert werden, mit denen gezielt bestimmte Effekte untersucht werden können. Die Generierung muss daher nach einigen Vorüberlegungen geschehen. Bei den Testmengen handelt es sich um Listen mit record-IDs. Daher wird statt Testmenge oder Menge auch der Begriff rid-Liste genutzt. Im Folgenden werden diese Begriffe synonym verwendet. In Abschnitt 3.2 wird bereits der Einfluss der rid-Listen auf die Trefferquote der Schnittmengenberechnung eingegangen. Einfluss auf die generierten Testmengen hat die Wahl der Verteilung, die mit dieser Verteilung einhergehenden Parameter, die Anzahl der zu erzeugenden Record-IDs sowie die Selektionsbedingung.
Zufallszahlen nach Verteilungen zu erzeugen ist leicht möglich, durch die Verwendung von Standardbibliotheken. Zufallszahlen, die eine Selektionsbedingung zu TRUE auswerten als RID zu verwenden stellt ebenfalls kein Problem da. Die Schwierigkeit in der Generierung von geeigneten Testmengen ist die Beachtung der Trefferquote. Wenn zwei rid-Listen nach Gleichverteilung mit einer Selektivität von jeweils 50% generiert werden, so beträgt die Wahrscheinlichkeit, dass eine RID in beiden Listen vorkommt $0,5 * 0,5 = 0,25$. Bei einer gleichen Anzahl an Elementen der beiden rid-Listen ergibt sich nach der Wahrscheinlichkeitsrechnung eine Trefferquote von 25%. Theoretisch kann es aber auch passieren, dass die rid-Listen gleich sind. Dies würde eine Trefferquote von 100% ergeben. Um also die Trefferquote während der Generierung von Testmengen beachten zu können, müssen die rid-Listen etwas aufwendiger konstruiert werden. Nachfolgende Abschnitte geben einen Einblick, wie die rid-Listen unter Beachtung von Selektivität, Art der Verteilung, Anzahl der RIDs und Trefferquote zwischen den Testmengen generiert werden.
Zu Beachten ist, dass die Algorithmen nur mit vorzeichenlosen Integer-Werten (unsigned int) für RIDs arbeiten. Daher ist es nur möglich, RIDs im Wertebereich von 0 bis $2^{32} = 4.294.967.296$ zu Erzeugen. Eine Selektivität von 0,1% bedeutet, dass nur jeder 1.000 Wert für eine RID gesetzt wird. Damit ergibt sich eine maximale Anzahl an 4.294.967 Werteausprägungen für RIDs. Mengen, die mehr RIDs enthalten sollen, sind bei dieser geringen Selektivität nicht möglich. Tabelle 3.1 gibt Auskunft, wie viele RIDs bei welcher Selektivität gewählt werden können.

Selektivität	gesetzte RID	Anzahl an möglichen RIDs
100%	jede	4.294.967.296
50%	jede 2.	2.147.483.648
10%	jede 10.	429.496.729
1%	jede 100.	42.949.672
0,1%	jede 1.000.	4.294.967
0,01%	jede 10.000.	429.496

Tabelle 3.1: Anzahl möglicher RID-Werte in Abhängigkeit von der Selektivität

3.3.1 Generierung von gleichverteilten Record-IDs

Sind die Daten in einer Datenbank gleichverteilt, so sind auch die daraus resultierenden RIDs in den Schnittmengen gleichverteilt. Gleichverteilte RIDs treten in gleichmäßigen Abständen innerhalb der rid-Liste auf. Die nicht vorkommenden RIDs sind dabei abhängig von der Selektivität und der Anzahl an RIDs die sich in der rid-Liste wiederfinden sollen. Eine Selektivität von 50% führt dazu, dass nur jede zweite RID gesetzt ist. Eine rid-Liste mit einer Selektivität von 50% könnte daher folgendermaßen aussehen: [1, 3, 5, 7, 9, ..., n*2], mit n =Anzahl an RIDs für die rid-Liste. Da nur jede zweite RID gesetzt ist, benötigt man für n RIDs $n * 2$ Werte für RIDs.
Ist die erste rid-Liste generiert, so kann eine zweite rid-Liste in Abhängigkeit von der ersten rid-Liste generiert werden. Einerseits wird die gewünschte Selektivität der zweiten rid-Liste beachtet, andererseits wird der gewünschten Trefferquote Rechnung getragen. Soll die Trefferquote 100% und die Anzahl der RIDs gleich sein, so bleibt nur die Möglichkeit, dass die zweite Liste die gleiche Liste ist wie die erste rid-Liste. Die Selektivität der zweiten Liste kommt dann nicht zum tragen. Ist die Anzahl der RIDs in den rid-Listen unterschiedlich, so kann nur aus Sicht der rid-Liste mit weniger RIDs eine Trefferquote von 100% erreicht werden. Die Liste mit weniger RIDs ist dann Teilmenge der Liste mit mehr RIDs. Sollen zwei RID-Listen mit jeweils n Elementen nach Gleichverteilung generiert werden, so wird zuerst die erste rid-Liste nur in Abhängigkeit der Anzahl an RIDs und der Selektivität konstruiert. Die zweite RID-Liste wird nun unter Angabe der Anzahl an RIDs für die zweite Liste, der Selektivität und der Trefferquote zur ersten Liste erstellt. Es werden $Trefferquote * Anzahl\ RIDs$-viele RIDs in die zweite RID-Liste aufgenommen, welche in dann in beiden RID-Listen, also auch in der Schnittmenge, liegen. Dazu werden die letzten $Trefferquote * Anzahl\ RIDs$-vielen RIDs der ersten Liste in die zweite Liste übernommen und die zweite Liste gemäß der Selektivitätsbedingung für die zweite Liste aufgefüllt, bis die Anzahl an RIDs für die zweite Liste erreicht ist. Sollte die Anzahl der RIDs der ersten Liste kleiner sein als $Trefferquote * Anzahl\ RIDs\ für\ zweite\ Liste$, so werden alle Elemente der ersten Liste in die zweite Liste übernommen und die Liste gemäß

der gewünschten Selektivitätsbedingung vervollständigt. Abbildung 3.2 verdeutlicht den Gedanken hinter der Konstruktion von gleichverteilten rid-Listen.

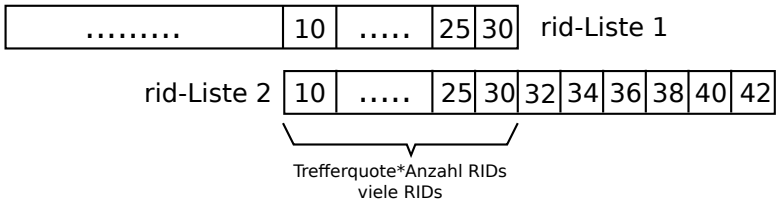

Abbildung 3.2: Skizzierte Grundgedanke für die Erstellung von zwei rid-Listen nach Gleichverteilung unter Berücksichtigung der Selektivität und der Trefferquote

3.3.2 Generierung von normalverteilten Record-IDs

Eine Normalverteilung ist durch die Werte μ und σ eindeutig bestimmt. Eine Normalverteilung zeichnet sich dadurch aus, dass die meisten Werte nah um den Mittelwert μ herum liegen. Der Grundgedanke in der Konstruktion einer rid-Liste mit normalverteilten RIDs ist in Abbildung 3.3 skizziert. Wert μ gibt im Grunde die Record-ID an, um welche

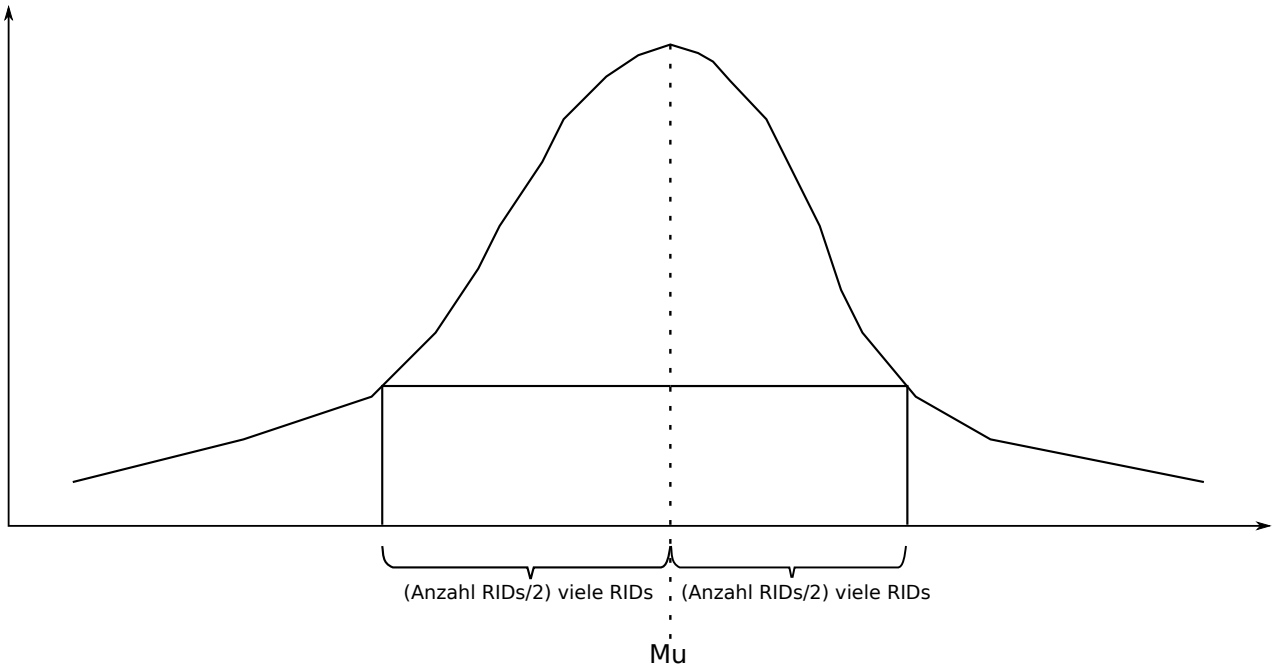

Abbildung 3.3: Skizze zur Idee hinter der Auswahl der RIDs für normalverteilte rid-Listen

herum die restlichen RIDs gesetzt werden um die gewünschte Anzahl an RIDs zu erhalten. Eine Liste von normalverteilten RIDs ist damit eine um den Wert μ liegende Liste mit gewünscht vielen RIDs. Wäre z. B. $\mu = 6$ und die Anzahl an RIDs beträge gleich 5, so ergäbe sich die Liste [4, 5, 6, 7, 8].
Die Generierung bzw. Konstruktion der zweiten rid-Liste erfolgt ähnlich der Konstruktion

der zweiten rid-Liste im Falle von gleichverteilten RIDs (s. o., Abschnitt 3.3.1). Es wird für die zweite rid-Liste wieder die gewünschte Trefferquote, sowie die Anzahl der RIDs für die zweite rid-Liste und der Wert für μ beachtet. Die RIDs im Falle von normalverteilten RIDs sind durchgängig ansteigende Werte für RIDs. Soll jetzt eine zweite rid-Liste unter Berücksichtigung der ersten rid-Liste generiert werden, so wird die zweite Liste gemäß der Trefferquote verschoben zur ersten Liste erstellt. Ist $\mu_{Liste2} < \mu_{Liste1}$, so wird, wenn möglich die zweite Liste von den Werten für die RIDs betrachtet, vor der ersten Liste erstellt. Um die zweite Liste vor der ersten erstellen zu können, muss der niedrigste RID-Wert der ersten Liste größer sein als die Anzahl an RIDs für Liste 2, falls die Trefferquote gleich 0% beträgt. Ist dies nicht der Fall, kann die zweite rid-Liste nicht vor der ersten erstellt werden, denn dort sind nicht genügend Werte für RIDs vorhanden, die noch nicht in Liste 1 verwendet wurden. Beträgt die Trefferquote nicht 0% oder 100%, so wird die zweite Liste mit $Trefferquote * Anzahl\ RIDs\ für\ Liste\ 2$-vielen RIDs erstellt, die sich auch in Liste 1 befinden. Dabei wird wenn möglich beachtet ob $\mu_{Liste2} < \mu_{Liste1}$ und ggf. versucht die zweite Liste „vor“ der ersten zu erstellen. Dies ist nur dann möglich, wenn noch genügend nicht benutzte RID-Werte vor Liste 1 zur Verfügung stehen. Beträgt die Trefferquote gleich 0%, so wird wenn möglich, die zweite rid-Liste unter Berücksichtigung ihres Wertes für μ, um μ herum konstruiert. Dies ist möglich, wenn der höchste so zu konstruierende RID-Wert kleiner ist, als der niedrigste RID-Wert der ersten Liste.

3.3.3 Generierung von Zipf-verteilten Record-IDs

Die aufwendigste Generierung von Zufallszahlen fällt der Generierung von Zipf-verteilten RIDs zu. Dazu werden zuerst einmal Zipf-verteilte Zufallszahlen generiert und im Anschluss wird ein Weg beschrieben, wie sich daraus dann Zipf-verteilte RIDs gewinnen lassen. Um der Verteilung nach dem Zipf'schen Gesetz gerecht zu werden, werden die Zufallszahlen innerhalb von 3 Hauptschritten erarbeitet. Zuerst wird die Wahrscheinlichkeit für die einzelnen Ränge errechnet, danach wird eine Kandidatenmenge gefüllt und aus dieser im Anschluss die konkreten Zufallszahlen gewonnen.
Im ersten wichtigen Teilschritt werden die Wahrscheinlichkeiten für die einzelnen Werte nach Rang gemäß zipf'schen Gesetz bestimmt. Diese sind abhängig von der Anzahl der Elemente und der Anzahl der möglichen Ausprägungen der Zufallszahlen, sowie dem Parameter α für die Zipf-Verteilung. Die Wahrscheinlichkeit für eine Ausprägung einer Zufallszahl beträgt $\frac{k}{r^{\alpha}}$. Dabei ist k die Konstante, die sich aus der Anzahl an möglichen Ausprägungen ergibt, r ergibt sich aus dem Rang der Ausprägung.
In der ersten Phase der Generierung von Zufallszahlen nach dem zipf'schen Gesetz werden die Wahrscheinlichkeiten ($p(n)$) für die einzelnen Ränge errechnet und in einen Vektor gespeichert. Der erste Eintrag des Vektor entspricht dabei der Wahrscheinlichkeit für die Ausprägung des Wertes des ersten Rangs, der zweite Wert für die Wahrscheinlichkeit der

Ausprägung des zweiten Rangs, usw.
Danach, in der zweiten Phase, wird eine Kandidatenmenge erstellt. Dies ist der Aufwendigste Schritt in der Generierung nach dem zipf'schen Gesetz. Dazu wird zuerst ein Vektor mit allen möglichen Ausprägungen, also allen möglichen Werten für die Zufallszahlen befüllt. Danach wird aus diesem Vektor ein zufälliger Wert bzw. eine Ausprägung gewählt. Diese wird dann gemäß der zuvor errechneten Wahrscheinlichkeit in einen anderen Vektor, die Kandidatenmenge, geschrieben. Die erste gewählte Ausprägung wird dabei $100 * p(1)$-mal in die Kandidatenmenge eingefügt. Die Kandidatenmenge ist dabei eine Multimenge, es sind also gleiche Elemente in der Menge möglich. $p(1)$ ist dabei die Wahrscheinlichkeit für den Wert von Rang 1. Es wird aus dem Vektor mit den möglichen Ausprägungen die zuvor gewählte Ausprägung entfernt. Die zweite zufällige Ausprägung wird im Anschluss zufällig aus dem Vektor mit den möglichen Ausprägungen gewählt. Die dann gewählte Ausprägung wird nun $100 * p(2)$-mal in die Kandidatenmenge eingefügt. Dabei ist $p(2)$ die Wahrscheinlichkeit für den Wert der Rang 2 innerhalb der Verteilung inne hat. Dies wird so häufig wiederholt, bis alle möglichen Ausprägungen für Zufallszahlen gemäß der Wahrscheinlichkeit ihres Ranges $*100$ in die Kandidatenmenge eingefügt worden sind. Damit ist die zweite Phase der Generierung von zipf-verteilten Zufallszahlen abgeschlossen und die dritten und letzte Phase beginnt.
In der dritten Phase werden jetzt die konkreten Zufallszahlen aus der Kandidatenmenge gezogen. Es wird zufällig die x-te Zahl aus der Kandidatenmenge gezogen. Diese ist dann die Zufallszahl für die Zufallszahlenmenge. Diese Ziehung wird so oft wiederholt, bis man so viele Elemente wie gewünscht hat.
Warum ergibt dieses Vorgehen Zipf-verteilte Zufallszahlen? Es ist so, dass während der Erstellung der Kandidatenmenge die einzelnen Ausprägungen zufällig einem Rang zugeordnet wurden und gemäß der Wahrscheinlichkeit des Rangs häufig in die Kandidatenmenge geschrieben wurden. Das Ziehen eines Wertes aus der Kandidatenmenge erfolgt gleichmäßig. Für die in der Kandidatenmenge häufiger auftretenden Ausprägungen ist dabei natürlich die Wahrscheinlichkeit aus der Kandidatenmenge gewählt zu werden höher als für die nicht so häufig in der Kandidatenmenge auftretenden Ausprägungen. Damit sind die daraus gezogenen Zufallszahlen, gemäß der Zipf-Verteilung, zipf-wahrscheinlich verteilt.
Um nun RIDs zu erhalten, die Zipf-verteilt sind, wird die Kandidatenmenge nicht mit einem zufälligen Wert $100 * p(1)$-mal gefüllt wird. Stattdessen wird die 1 $100 * p(1)$-mal der Kandidatenmenge hinzugefügt. Als nächstes wird dann die Zahl 2 $100 * p(2)$-mal der Kandidatenmenge hinzugefügt. Dies wird mit allen möglichen Werten wiederholt, bis die Kandidatenmenge gefüllt ist. Danach wird, wie zuvor auch, eine zufällige Zahl aus der Kandidatenmenge gezogen. Durch die Angabe eine Selektionsbedingung ($<$,$=$ oder $>$ als x) wird dann z. B. nur in dem Ziehungsdurchgang eine RID gesetzt, wenn die aus der Kandidatenmenge gezogenen Zufallszahl die Selektionsbedingung erfüllt. Die Wahrscheinlichkeit für das Ziehen einer 1 aus der Kandidatenmenge ergibt sich daher aus der Wahrscheinlich-

keit für die ranghöchste Ausprägung. Es werden solange Zahlen aus der Kandidatenmenge gezogen, bis so viele RIDs wie gewünscht gesetzt worden sind.
Die zweite Schnittmenge kann nicht in Abhängigkeit von der Trefferquoten zur ersten Schnittmengen generiert werden.

3.4 Laufzeitmessung

Um die Laufzeiten der Algorithmus zu messen und differenziert betrachten zu können, wird an mehreren Stellen innerhalb der Algorithmen, eine Zeitmessung durchgeführt. Eine Zeitmessung misst dabei die Ausführungszeit für den Code und die Methodenaufrufe zwischen dem Startpunkt und dem Endpunkt der Zeitmessung. Folgender Codeausschnitt enthält die für die Zeitmessung verwendeten Komponenten:

```
#include <chrono>

/* Code vor Messung
 * ...
 */

auto start = std::chrono::high_resolution_clock::now();
//Hier der Code, dessen Laufzeit gemessen wird
auto ende = std::chrono::high_resolution_clock::now();
int zeitInMillisekunden =
        std::chrono::duration_cast<std::chrono::milliseconds>
        (ende - start).count();
//Zeit gespeichert in zeitInMillisekunden

/* Code nach Messung
 * ...
 */
```

Es wird zur Zeitmessung *<chrono>* in das Programm eingebunden. Die Variable *start* bekommt den aktuellen Zeitpunkt vor Ausführung des Codeabschnitts, dessen Laufzeit gemessen werden soll, zugewiesen. Nach der Ausführung des für die Messung relevanten Codes wird der Variablen *ende* der aktuelle Zeitpunkt nach Ausführung des Codes zugewiesen. Aus der Differenz des späteren Zeitpunktes mit dem früheren Zeitpunkt ergibt sich die Laufzeit. Um die Laufzeit in Millisekunden zu bestimmen wird

```
std::chrono::duration_cast<std::chrono::milliseconds>
        (ende - start).count();
```

aufgerufen.
Der Benchmark nimmt die Laufzeiten für die Arbeitsabschnitte *Sortieren*, *Vorbereiten*, *Schnittmengenberechnung* und *Rückschreiben*. Die Rückschreib-Phase umfasst Arbeiten, die nach der Schnittmengenberechnung zu erledigen sind. Dies können Arbeiten sein, die eine gemachte Formatierung rückgängig machen oder ähnliches. Die einzelnen Laufzeiten für die verschiedenen Arbeitsschritte werden innerhalb des Benchmarks gespeichert und nach allen Wiederholungen der Messungen zur Ermittlung des Medians und zur Errechnung des arithmetischen Mittels verwendet.
Um die einzelnen Laufzeiten aus dem Algorithmen zu Gewinnen, wurden die Algorithmen soweit angepasst, dass ein Vektor für die einzelnen Laufzeiten übergeben werden kann. In diesem Vektor werden dann die einzelnen Teillaufzeiten abgespeichert.
Die Gesamtlaufzeit ergibt sich aus Vorbereitungsaufwand, der Schnittmengenberechnung an sich und dem Aufwand, der nach der Schnittmengenberechnung noch anfällt. Im Folgenden werde ich die einzelnen anfallenden Arbeiten der Algorithmen diesen Phasen zuordnen. Zur Vorbereitungsarbeit des SIMD-Algorithmus zählt die Formatierung der übergebenen rid-Listen. Die RIDs werden in höherwertige und niederwertige Bits unterteilt. Während des Arbeitsschritts der Schnittmengenberechnung wird dann mit den formatierten RIDs die Schnittmenge berechnet. Im Anschluss wird die Formatierung rückgängig gemacht (Rückschreib-Phase).
Der Radix-Cluster Algorithmus bildet in der Vorbereitungsphase die einzelnen Cluster. Danach folgt die Berechnung der Schnittmenge. Abschließende Arbeiten fallen nicht an. Hashintersection bildet in der Vorbereitungsphase die Hash-Tabelle der Schnittmengen. Danach wird die Schnittmenge berechnet, indem die RIDs der zweite Schnittmengen gehasht werden und in der zuvor gebildeten Hashtabelle auf eine korrespondierende RID geprüft wird. Für die Rückschreibphase steht ebenfalls keine Arbeit an.
In der Vorbereitungsphase der Bitmap Intersection Algorithmen (Bitmap ohne Komprimierung und WAH 32/128) wird aus der übergebenen rid-Liste die Bitmap erstellt und gegebenenfalls per WAH Kompression komprimiert. Die Berechnung der Schnittmenge findet im Anschluss auf den Bitmaps statt. In der Rückschreibphase wird die Ergebnisbitmap, die in der Berechnungsphase berechnet wurde, in eine rid-Liste umgewandelt.
In std::set_intersection kann nicht genauer gemessen werden. Daher wird jeglicher Aufwand, der von set_intersection anfällt als eine Laufzeit gemessen und zur Phase der Schnittmengenberechnung gezählt. Dadurch ist zwar keine genauere Differenzierung in die einzelnen Laufzeiten nach Arbeitsphasen möglich, aber ein Vergleich mit den anderen Algorithmen ist trotzdem möglich.

3.5 Input für die Intersection-Algorithmen

Um die Algorithmen hinsichtlich realer Systeme zu testen, sollte den Algorithmen zur Schnittmengenberechnung jeweils der optimale Input bereit gestellt werden. Bitmap-Intersection erfordert z. B. Bitmaps, der SIMD-Algorithmus hingegen arbeitet mit den sortierten rid-Listen als solches. Ein Vergleich zwischen Bitmap und SIMD-Algorithmus, bei gegebenem Input von sortierten rid-Listen, wäre unfair, wenn die Zeit für die Berechnung der Bitmap zur Ausführungszeit des Bitmap-Algorithmus hinzu gerechnet würde. Da die Algorithmen auch im Hinblick auf die in dem System verwendeten Datenstrukturen gewählt werden sollte, wird der Input gewählt, der der Repräsentation der Daten im realen System entspricht. Da der Benchmark nur rid-Listen zur Verfügung stellt, muss von den Bitmap-basierten Verfahren nur die Zeit für die Schnittmengenberechnung betrachtet werden. Sollte ein Algorithmus trotz nicht optimalen Input eine bessere Laufzeit erreichen, als ein Algorithmus mit optimalen Input, so wäre dies eine positive Erkenntnis. Als optimaler Input stehen demnach Bitmaps sortierten rid-Listen gegenüber. Einige Verfahren zur Schnittmengenberechnung (Hashintersection, Bitmap Intersection (ohne WAH Kompression) und Radix-Cluster Intersection) erfordern keine sortierten rid-Listen als Eingabe. Daher können die rid-Listen noch gemischt werden um auch die Ausführungszeit bei unsortierten Testmengen feststellen zu können. Der Aufwand für das Sortieren von rid-Listen kann bestimmt werden und zu der Gesamtausführungszeit hinzu addiert werden. Die Betrachtung von Ausführungszeiten auf sortierten Daten ist für Anfragen an Datenbanken interessant, die die Daten bereits sortiert in den Relationen vorliegen haben. Damit sind die Ergebnisse des Benchmarks somit unabhängig von der Effizienz des verwendeten Sortierverfahrens. Ergibt sich die Möglichkeit ein effizienteres Sortierverfahren zu verwenden, so kann die Ausführungszeit des effizienteren Sortierverfahrens zur Addition auf die gemessene Laufzeit verwendet werden. Weitere größere Operationen sind zur Vorbereitung der Testmengen nicht erforderlich.

Der SIMD-Algorithmus arbeitet nur mit aufsteigend sortierten rid-Listen, also Listen mit

Algorithmus	sortierte Daten	unsortierte Daten	Darstellungsform
SIMD	ja	nein	rid-Listen
Hashintersection	ja	ja	rid-Listen
Radixcluster Intersection	ja	ja	rid-Listen
Bitmap	ja	ja	Bitmaps
Bitmap mit WAH	ja	nein	Bitmaps

Tabelle 3.2: Input für die verschiedenen Algorithmen

aufsteigend sortierten nicht-negativen Ganzzahlen. Dabei ist es normalerweise so, dass rid-Listen in realen Systemen bereits sortiert vorliegen. Sortierte rid-Listen sind der optima-

le Input für die Schnittmengenberechnung mit dem SIMD-Algorithmus. Hashintersection und Radix-Cluster Intersection benötigen ebenfalls rid-Listen. Im Gegensatz zum SIMD-Algorithmus müssen diese nicht sortiert vorliegen.
Die Schnittmengenberechnung per Bitmap-Intersection arbeitet auf Bitmaps. Der Benchmark generiert rid-Listen, also müssen diese noch für die Bitmap-Intersection in Bitmaps konvertiert werden. Optimaler Input für die Schnittmengenberechnung per Bitmap-Intersection besteht demnach aus der Eingabe der Bitmaps oder von WAH komprimierten Bitmaps.

3.6 Output des Benchmarks

Der Benchmark produziert rid-Listen nach der Anzahl der Tupel, der Selektivität und der Quote an Tupeln in der Schnittmenge (Trefferquote). Diese rid-Listen können vom Algorithmus als Datei während der Ausführung des Benchmarks auf die Festplatte geschrieben werden. In den Dateien sind alle RIDs der rid-Listen fortlaufend untereinander enthalten. Die Ausgabe der rid-Listen als Datei ist optional und benötigt bei größeren rid-Listen einiges an Zeit und Speicherplatz.
Neben den rid-Listen sind die Ergebnisse der Laufzeitmessungen der Schnittmengenberechnungen der andere Output des Benchmarks. Es gibt gemessene Laufzeiten zur Sortierung, zur Vorbereitung des Algorithmus, zur Berechnung der Schnittmenge, sowie zu Arbeiten nach der Berechnung. Die einzelnen Laufzeiten zu den Arbeitsschritten werden per Leerzeichen getrennt voneinander in Dateien gespeichert. Die Datei mit dem Namen „*LaufzeitenDetailAlleMessungen.dat*“ enthält die verschiedenen Laufzeiten der einzelnen Messdurchläufe. Dabei steht die erste Zahl für den Durchlauf, also 1 bedeutet, es ist die erste Messung, 2 steht für die zweite Messung usw. Der zweite Zahlenwert steht beinhaltet die Summe der einzelnen Laufzeiten. An dritter Stelle steht die Laufzeit für die Sortierung der rid-Listen, also für die Laufzeit von std::sort(). Danach folgt als vierter Wert die Laufzeit für die Vorbereitung der Algorithmen. Welche Vorbereitungsarbeit für welchen Algorithmus anfällt, wurde bereits in Abschnitt 3.4 näher beschrieben. Auf den Laufzeitwert der Vorbereitung folgt der Laufzeitwert der Schnittmengenberechnung an fünfter Stelle. Zuletzt wird noch die Laufzeit für Arbeiten nach der Schnittmengenberechnung an sechster Stelle angefügt. Alle Werte sind per Leerzeichen separiert. Zu Begin steht innerhalb der Datei die Bedeutung der einzelnen Werte hinter # .
Die Datei „*LaufzeitDurchschnitt.dat*“ enthält das arithmetische Mittel aus den Laufzeitmessungen. Dabei wird die Summe der durchschnittlichen Laufzeiten gespeichert, gefolgt von der durchschnittlichen Laufzeit für das Sortieren der Liste, der durchschnittlichen Laufzeit für die Vorbereitungsarbeiten, der durchschnittlichen Laufzeit der Schnittmengenberechnung und der durchschnittlichen Laufzeit für die Arbeiten nach der Schnittmengenberechnung. Analog dazu enthält die Datei „*LaufzeitenMedian.dat*“ die Median-Werte

der Laufzeiten.
Um für die Bitmap-basierten Schnittmengenberechnungsverfahren vom optimalen Input auszugehen, betrachtet man aus dem Messergebnissen nur das Messergebnis für die Schnittmengenberechnung. Die Laufzeitmessung für die Konvertierung in eine Bitmap und zurück zu einer rid-Liste setzt man dazu einfach auf 0.
Jeglicher Output wird in einem Ordner mit dem Name „*benchmark_*“ gefolgt von dem zur Zeit der Ordnererstellung aktuellen Datum und Uhrzeit und den gewählten Parametern gespeichert. Der Ordner in dem der Output gesammelt wird heißt demnach z. B.: *benchmark_ 24.7.2014-11.57.27_ WH_ 25_ Algo_ 1_ Tup_ 128000_ 128000_ TQ _ 1.0 _ Sel _ 1.0 _ TypV_ 1_ 1_ Mu_ Alpha_ 1.0_ 1.0_ k_ 100_ 100_ SelArt _ 1_ 1_ SelWert _ 2 _ 2 _ 1 _ 1_ 0_ 1.*

3.7 Ablauf des Benchmarks

Im Folgenden Abschnitt wird der Ablauf des Benchmarks erklärt. Hierzu wird auf die einzelnen Arbeitsabschnitte eingegangen, die der Benchmark erledigt.
Zu Beginn des Benchmarks werden die Eingaben geparst, die per Programmaufruf übergeben werden können. Werden keine Werte übergeben, so werden die standardmäßig implementierten Werte genommen. Der Aufruf

```
./benchmark 25 1 128000000 128000000 1.0 1.0 1 1 1.0 1.0 100 100
 1 1 2 2 0 1 0 1
```

startet dabei mit den Benchmark mit 25 Wiederholungen der Messung. Außerdem ist für das Schnittmengenverfahren 1 gewählt, dies steht für die SIMD-Schnittmengenberechnung. Die Zahlen an 3. und 4. Stelle (128000000) stehen für die Anzahl an RIDs in der ersten und der zweiten rid-Liste. Danach folgt die Angabe der Trefferquote und der Selektivität. Für beides wurde 1.0, also 100% angegeben. Die darauffolgenden 1en stehen für die Art der Verteilung der RIDs in der ersten und der zweiten rid-Liste.Die Werte an 9. und 10. Stelle stehen für den Mu-Wert, falls die Verteilungsart gleich 2 und damit Normalverteilt gewählt wurde. Bei Wahl der Zipf-Verteilung stehen diese für α. Anschließende 0en und 1en stehen für den Bool-Wert für das Speichern der rid-Listen auf der Festsplatte, für die Verwendung von mehreren Kernen bei der Schnittmengenberechnung, für die Messung des Sortieraufwands sowie für die Möglichkeit, unsortierte rid-Listen für die Schnittmengenberechnung zu verwenden. Möchte man ein Bitmap-basierendes Schnittmengenverfahren benchmarken, so wird analog

```
./benchmarkBitmap
```

aufgerufen. Die Parameter sind die gleichen, nur die Schnittmengenverfahren, welche hinter dem zweiter Parameter stehen, unterscheiden sich. Tabelle 3.3 gibt einen Überblick über

die Aufrufparameter des Benchmarks. Die Parameter umfassen die Variablen, die man im Aufruf beeinflussen kann. Damit lassen sich verschiedene Auswirkungen der Änderung einzelner Variablen auf die Ausführungszeiten untersuchen.
Nachdem die Eingangsparameter geparst wurden, werden die rid-Listen anhand der eingegebenen Werte erstellt. Die Generierung von Testdaten wurde schon eingehend in Abschnitt 3.3 erläutert. Die rid-Listen werden nach der Erstellung sortiert. Danach wird der Ordner für den Output des Benchmarks erstellt. Im Anschluss werden, falls gewünscht (↗Parameter 17), die rid-Listen als Datei auf der Festplatte gespeichert. Im Anschluss werden die rid-Listen auf Wunsch per Aufruf von random_shuffle gemischt. Nach diesen Arbeiten beginnt der Abschnitt des Messens. Die Messungen werden gemäß der gewünschten Anzahl an Messungen wiederholt (↗Parameter 1). Zuerst wird der Aufwand für das Sortieren gemessen, falls dies gewünscht war. Dafür werden die Listen per random_shuffle gemischt und die Zeit für das Sortieren im Anschluss gemessen. Als Sortierverfahren wird auf std::sort zurückgegriffen. Im Anschluss daran werden die rid-Listen, welche in intern in einem std::vector verwaltet werden, in ein plain array kopiert. Diese Arbeitskopien werden den Algorithmen später übergeben. Darauf folgt die eigentliche Schnittmengenberechnung. Im Anschluss werden die Messergebnisse der Schnittmengenberechnung vor der nächsten Wiederholung gespeichert. Sind alle Wiederholungen abgeschlossen, so werden die ermittelten Messergebnisse in verschiedenen Dateien gespeichert, siehe auch Abschnitt 3.6. Im Anschluss wird nur noch etwas Speicher wieder freigegeben und die Ausführung des Benchmarks terminiert.

3.8 Zusammenfassung

In diesem Kapitel wurde der Aufbau des Benchmarks hinreichend beschrieben. Dazu wurden zuerst die zur Verfügung stehenden Algorithmen aufgezählt. Es folgt eine Übersicht über die Variablen im Benchmark. Im Anschluss daran wurde die Generierung von Testmengen beschrieben. Dabei wurde erklärt, wie man anhand der Wahrscheinlichkeitsverteilungen rid-Listen generiert, die diesen Wahrscheinlichkeitsverteilungen entsprechen. Im Anschluss wurde thematisiert, wie die Laufzeit gemessen wird und welchen Output der Benchmark generiert. Außerdem wird erläutert, welchen Input der Benchmark annimmt. Abschließend wurde der Ablauf des Benchmarks erörtert.

Nr.	mögliche Werte	Beschreibung
1	1-∞	Anzahl der Messwiederholungen
2	1 - 4 und 1 - 3	siehe unten, Tabelle 3.4
3	1 - ∞	Anzahl der RIDs für die erste Schnittmenge
4	1 - ∞	Anzahl der RIDs für die zweite Schnittmenge
5	0 - 1.0	Trefferquote, bei 0 ist die Schnittmenge leer
6	0.01 - 1.0	Selektivität, siehe dazu auch Tabelle 3.1
7	1,2,3	Typ der Verteilung für erste Schnittmenge, 1=Gleichverteilung, 2=Normalverteilung, 3=Zipf
8	1,2,3	Typ der Verteilung für zweiten Schnittmenge, 1=Gleichverteilung, 2=Normalverteilung, 3=Zipf
9	1 - Anzahl RIDs erste Menge	Wenn Normalverteilte RIDs, dann werden um diesen Wert die RIDs gesetzt, bei Zipf-Verteilung ist dies =α
10	1 - Anzahl RIDs erste Menge	Wenn Normalverteilte RIDs, dann werden um diesen Wert die RIDs gesetzt, bei Zipf-Verteilung ist dies =α
11	1 - ∞	k für Zipf-Verteilung, 1. Schnittmenge, siehe auch Abschnitt 3.3.3
12	1 - ∞	k für Zipf-Verteilung, 2. Schnittmenge
13	1 - ∞	Selektionsart für 1. Schnittmenge: 1=„<“, 2=„=“, 3=„>“ als Selektionswert für 1. Schnittmenge
14	1 - ∞	Selektionsart für 2. Schnittmenge
15	1 - ∞	Selektionswert für 1. Schnittmenge
16	1 - ∞	Selektionswert für 2. Schnittmenge
17	0 oder 1	Bool, ob rid-Listen auf Festplatte gespeichert werden sollen (1=speichern)
18	0 oder 1	Bool, Multicore (falls möglich, 1=mehrere Kerne)
19	0 oder 1	Bool, Zeitaufwand für Sortieren messen (1=speichern)
20	0 oder 1	Bool, ob unsortierte rid-Listen (falls möglich) verwendet werden sollen (1=sortierte Listen)

Tabelle 3.3: Input für die verschiedenen Algorithmen

Nr.	Algorithmus ./benchmark
1	SIMD
2	Hashintersection
3	Radix-Cluster Intersection
4	std::set_intersection

Nr.	Algorithmus ./benchmarkBitmap
1	Bitmap
2	Bitmap mit WAH 128
3	Bitmap mit WAH 32

Tabelle 3.4: Schnittmengenberechnungsverfahren

Kapitel 4

Evaluierung

In diesem Kapitel geht es um die Evaluation der Schnittmengenberechnungsverfahren. Zur Evaluation der verschiedenen Algorithmen wird der Benchmark eingesetzt. Zuerst sollen Erkenntnisse zu den einzelnen Algorithmen gewonnen werden. Mit den Ergebnissen zu den einzelnen Verfahren wird dann ein Vergleich untereinander angestrebt. Außerdem folgt eine Bewertung der Algorithmen hinsichtlich möglichen Einsatzzwecken in realen Systemen.
Im Folgenden wird WAH 128 für die Bitmap Intersection mit WAH Komprimierung (Wortlänge 128) und WAH 32 für die Bitmap Intersection mit WAH Komprimierung und einer Wortlänge von 32 bit synonym verwendet. Statt Radix-Cluster Intersection wird auch einfach nur Radix Verwendung finden. Ebenso steht SIMD für die SIMD-Schnittmengenberechnung. Das Schnittmengenberechnungsverfahren aus der C++-Standardbibliothek (std::set_intersection) wird im Folgenden unter anderem auch als Set Intersection, Standardalgorithmus oder Standard Intersection benannt.
Die verschiedenen Schnittmengenberechnungsverfahren werden dabei in der ersten Einzelbetrachtung bezüglich ihrer verwendeten Techniken zusammengefasst. Der Radix Algorithmus setzt auf das Clustern der RIDs, der SIMD Algorithmus und die C++-Standardschnittmengenberechnung arbeiten skalar auf RIDs. Die Technik hinter dem Bitmap-Verfahren unterscheidet sich nur darin, ob Kompression eingesetzt wird oder nicht.
Bis auf std::set_intersection stammen alle verwendeten Implementierungen der Schnittmengenberechnungsverfahren aus der parallel laufenden Bachelorarbeit, siehe dazu auch Abschnitt 1.3.

4.1 Testsystem

Alle der folgenden Messungen sind unter dem Betriebssystem Linux Mint 17 Cinnamon 64-bit ausgeführt worden. Die Linux-Kernel Version ist 3.13.0-24-generic. Als Prozessor kam ein Intel© Xeon© E3-1230 v3 @ 3.30GHz zum Einsatz. Die Größe des Arbeitsspeichers

beträgt 7,7 GiB. Als Compiler wurde der gcc-Compiler in Version 4.8.2 (Ubuntu 4.8.2-19ubuntu1) verwendet.

4.2 Ein erster Test

Im Folgenden wird eine erste Hypothese aufgestellt und überprüft. Dabei handelt es sich um einen ersten einfachen Vergleich des Laufzeitverhaltens in Bezug zur Größe der Schnittmengen. Dieser erste Test soll darüber hinaus auch als ein erstes einleitendes Beispiel für Testfälle der Evaluation dienen und in den Umgang mit Hypothesen einführen.
Hypothese: Alle Verfahren zur Schnittmengenberechnung verhalten sich in ihrer Laufzeit linear bei linearem Wachstum der Testmengen, wenn Selektivität und die Trefferquote bei den Testmengen konstant bleiben.
Konstruktion geeigneter Testmengen: Bei Testmengen mit einer Selektivität von 100% und einer Trefferquote für die Schnittmengenberechnung von 100% ergeben sich gemäß der Gleichverteilung zwei rid-Listen mit durchgängigen Werten für die Record-IDs. Die Listen sind von der Art [1, 2,. . . , n] für eine Anzahl von n-Elementen für die Testmengen.
Überlegungen: Sämtliche Arbeiten der verschiedenen Algorithmen ist gleichmäßig. Die Arbeit steigt dabei linear an, da die Größe der rid-Listen linear ansteigt. Für die Ausführungszeit wird somit auch ein linearer Anstieg erwartet. Seiteneffekte, die die Ausführungszeit verlängern könnten sind nicht zu erwarten.
Ergebnisse zur Hypothese: Die Ausführungszeiten wurden für die Algorithmen SIMD Intersection, Radix-Cluster Intersection und Bitmap Intersection gemessen. Die Messungen für Bitmap wurden ohne Kompression (Bitmap) und mit WAH Kompression (Bitmap WAH 128 und Bitmap WAH 32) durchgeführt. Der Aufwand für das Sortieren von Daten wurde nicht mit einbezogen. Die Laufzeit für die Bitmap-Algorithmen umfasst die Erstellung einer Bitmap sowie die Rückkonvertierung der Ergebnisbitmap zu RIDs. In Abbildung 4.1

RIDs in Schnittmengen	SIMD	Radix	Set Intersection	Bitmap	WAH 128	WAH32
32.000.000	81	272	33	68	58	62
64.000.000	153	518	64	128	118	124
96.000.000	220	687	100	189	177	190
128.000.000	288	858	134	246	236	253

Tabelle 4.1: Median der Messwerte. Alle Messwerte in Millisekunden.

erkennt man das lineare Laufzeitverhalten der Algorithmen. Auf der x-Achse ist die Anzahl der Elemente in den beiden Schnittmengen aufgetragen. Die Laufzeit in Millisekunden ist auf der y-Achse aufgetragen. Die Annahme eines linearen Laufzeitverhaltens bei linearem

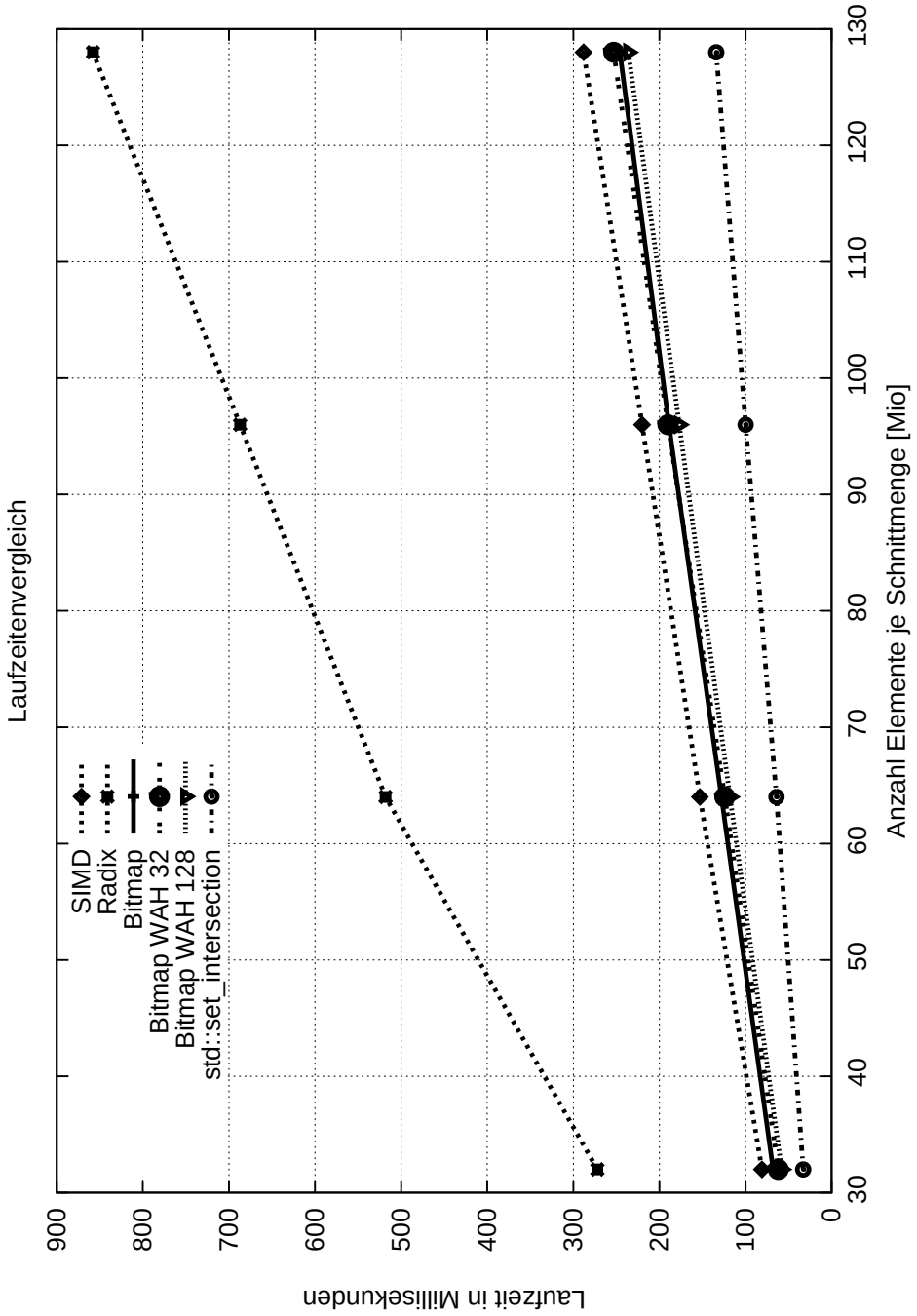

Abbildung 4.1: Laufzeitvergleich mit ansteigender Anzahl an Elementen in beiden Schnittmengen.

Anstieg der Elemente der Schnittmengen kann damit bestätigt werden. Die Hypothese ist anzunehmen
Aus diesem Vergleich lässt sich eine erste Tendenz der Performance der Algorithmen gewinnen. Ohne die Einberechnung des Sortieraufwands übersteigt die Laufzeit der Radix-Cluster Intersection bei 128 Millionen RIDs pro Schnittmenge, die Laufzeit des SIMD Intersections-Algorithmus um den Faktor 2,92. Außerdem ist zu erkennen, dass die Bitmap-basierten Verfahren nah beieinander liegen.

4.3 Radix-Cluster Algorithmus

Nach dem ersten Beispiel für eine Messung wird nun im Folgenden der Radix-Cluster Algorithmus und seine Performance näher untersucht. Dazu werden die Anzahl der Elemente in den rid-Listen variiert und ebenfalls die Selektivität und die Trefferquote variiert. Dazu wird noch eine Untersuchung der Auswirkung der Verwendung von unsortierten rid-Listen statt sortierten rid-Listen durchgeführt.

4.3.1 Evaluation des Radix-Cluster Algorithmus bei sortiertem und unsortiertem Input

Im ersten Test wird untersucht, ob die Verwendung von unsortierten rid-Listen, an Stelle von sortierten rid-Listen, als Eingabe, Einfluss auf die Berechnungszeit der Schnittmenge hat.
Hypothese: Die Verwendung von unsortierten rid-Listen als Eingabe für den Radix-Cluster Algorithmus anstatt der Eingabe von sortierten Listen haben negativen Einfluss auf die Gesamtlaufzeit des Algorithmus.
Konstruktion geeigneter Testmengen: Die Testmengen werden mit den gleicher Selektivität und Trefferquote generiert. Für diese Hypothese wird eine Selektivität von 100% gewählt, genauso für die Trefferquote. Die Anzahl der RIDs ist in beiden Schnittmengen gleich groß. Dabei wird mit 32, 64, 96 und 128 Millionen RIDs gemessen. Die RIDs werden gemäß der Gleichverteilung generiert.
Überlegungen: Die Ausführungszeit des Radix-Cluster Algorithmus verlängert sich, wenn statt sortierter rid-Listen unsortierte rid-Listen als Input dienen, weil beim Bilden von Clustern (Initialisierungsphase) die RIDs in unsortierter Reihenfolge auftreten. Da beim clustern die RIDs nach ihren einzelnen Bits geclustert werden, resultiert aus der unsortierte Reihenfolge der RIDs , dass die RIDs in der Regel in verschiedene Cluster unterteilt werden. Im Falle von sortierten rid-Listen wird öfters in das gleiche Cluster geclustert, dies ist effizienter.
Ergebnisse zur Hypothese: Im Anhang in Tabelle A.1 sind die Messergebnisse der Laufzeitmessung zu dieser ersten Hypothese zu finden. In Abbildung 4.2 sind die Messerwerte

visualisiert. Es ist zu erkennen, dass die Gesamtlaufzeit bei unsortiertem Input die Laufzeit

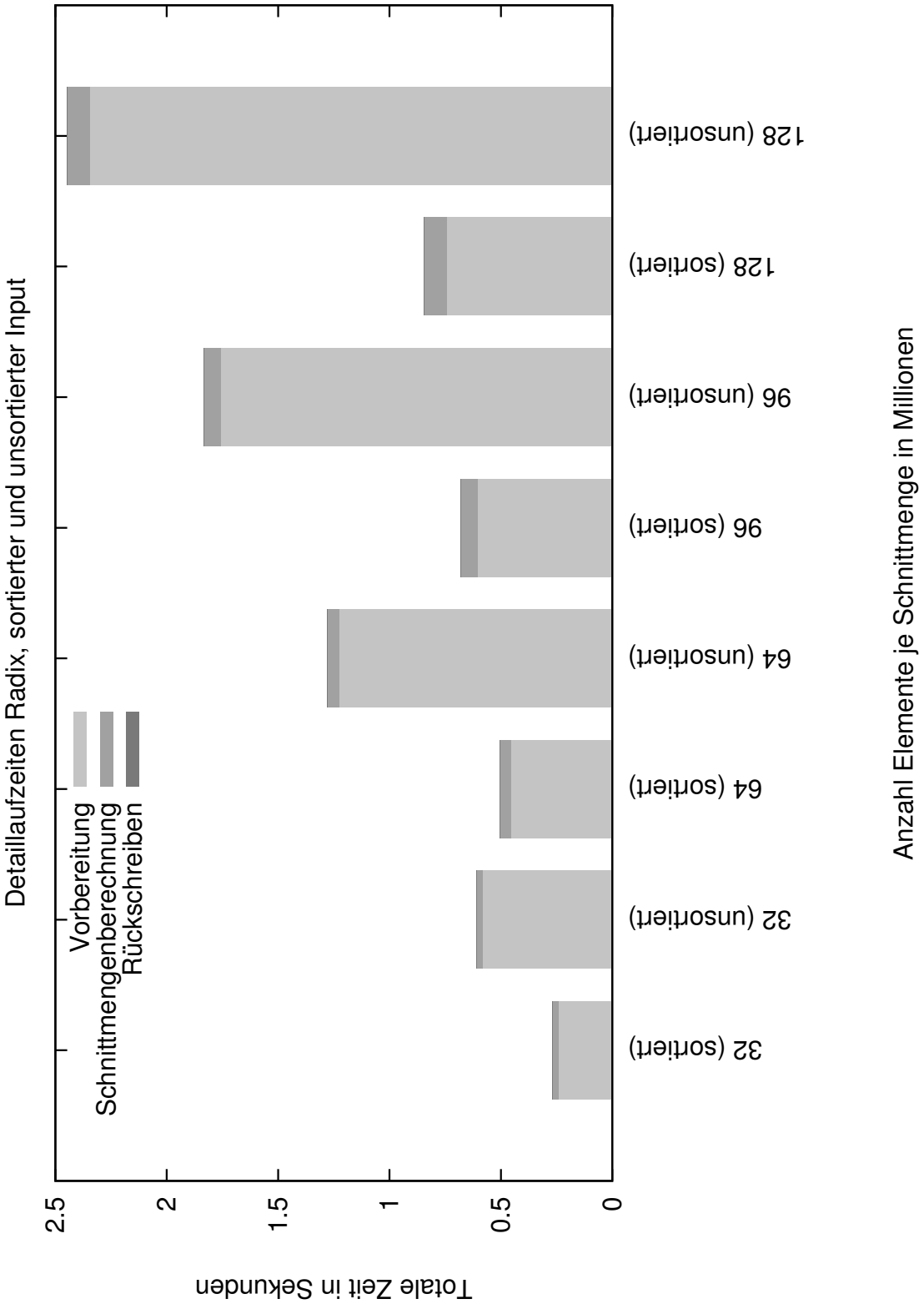

Abbildung 4.2: Laufzeitvergleich mit sortierten und unsortierten rid-Listen.

bei sortierten Input deutlich übersteigt (Faktor 2,26, 2,53, 2,69 und 2,89). Jeglicher Laufzeitzuwachs ist in der Vorbereitungsphase zu finden. Die Berechnungszeit ist unverändert zwischen sortiertem und unsortiertem Input. Da der Algorithmus nach Vorbereitung der Daten auf gleichen Grundlagen, gemeint sind die Cluster, operiert, wäre ein Anstieg der Berechnungszeit sehr verwunderlich. Die oben genannte Hypothese kann daher angenommen werden. Tabelle 4.2 enthält Messwerte zur Laufzeit des C++-Standardsortierverfahrens.

Elemente je rid-Liste	Sortieraufwand für 2 rid-Listen [ms]
32.000.000	4667
64.000.000	9656
96.000.000	14778
128.000.000	20037

Tabelle 4.2: Sortieraufwand für 2 rid-Listen in Millisekunden.

Die Zeitmessung gilt für das Sortieren von je zwei rid-Listen mit der angegebenen Anzahl an Elementen. Es ist zu erkennen, dass Aufwand für das Sortieren der rid-Listen die Differenz der Laufzeit zwischen sortierten und unsortierten rid-Listen deutlich übersteigt (siehe A.1). Es ist also Effizienter, den Radix-Cluster Algorithmus auf unsortierte rid-Listen arbeiten zu lassen, statt die Listen vorher zu sortieren.

4.3.2 Auswirkungen der Selektivität auf die Laufzeit des Radix-Cluster Algorithmus

Nachdem das Laufzeitverhalten des Radix Algorithmus bei sortiertem und unsortiertem Input untersucht wurde, wird in diesem zweiten Test das Laufzeitverhalten dieses Schnittmengenberechnungsverfahrens hinsichtlich der Veränderung der Selektivität untersucht.
Hypothese: Die Verringerung der Selektivität bei ansonsten gleichen Parametern führt zu einer längeren Gesamtlaufzeit des Radix Algorithmus.
Konstruktion geeigneter Testmengen: Es werden normalverteilte rid-Listen mit je 4.294.967 RIDs pro Schnittmenge verwendet. Die Trefferquote beträgt 100%. Die Selektivität wird variiert. Da die niedrigste Selektivität 0,1% beträgt, ist die Anzahl der RIDs auf 4.294.967 beschränkt.
Überlegungen: Das Laufzeitverhalten des Radix Schnittmengenberechnungsverfahrens müsste sich aus den selben Gründen wie schon im Test aus Abschnitt 4.3.1 verschlechtern. Statt unsortierten Listen, liegen die RIDs bei sinkender Selektivität weiter auseinander. Dies führt dazu, dass beim Clustern nicht in das geleiche Cluster geclustert werden kann. Die Vorbereitungsphase wird aufwendiger.
Ergebnisse zur Hypothese: Die Messergebnisse finden sich im Anhang in Tabelle A.2 und visualisiert in Abbildung 4.3 in diesem Abschnitt. Es ist zu erkennen, dass die Laufzeit der Vorbereitungsphase wie erwartet zunimmt. Die Ausführungszeit der Schnittmengenbe-

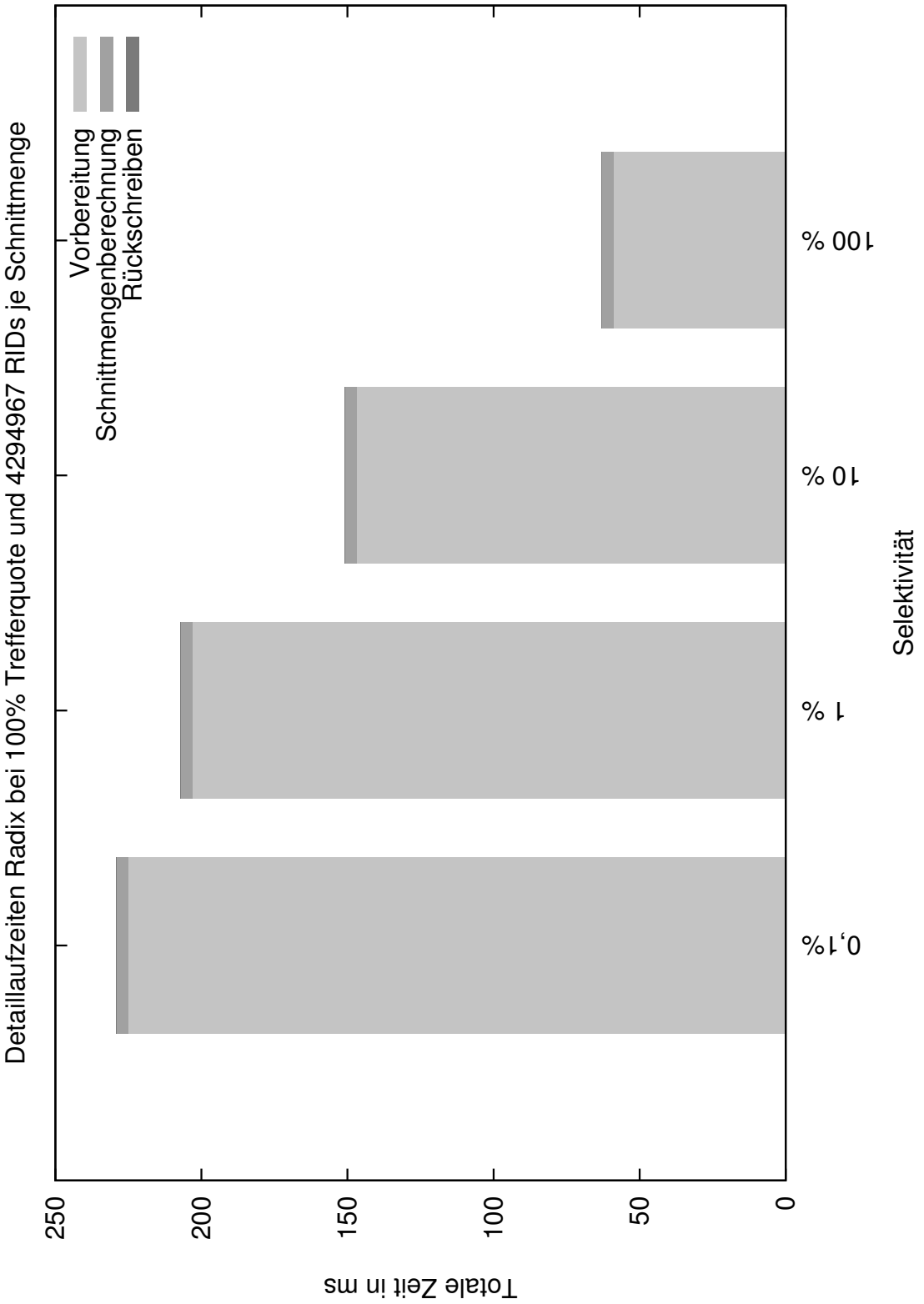

Abbildung 4.3: Laufzeitvergleich des Radix-Cluster Algorithmus bei variierenden Selektivitätswerten.

rechnung bleibt dabei konstant. Die oben genannte Hypothese kann daher angenommen werden. Sind die RIDs in kurzen Abständen hintereinander gesetzt, wie dies bei einer hohen Selektivität (maximal 100%) der Fall ist, hat dies einen positiven Effekt auf die Laufzeit des Algorithmus. Die Ausführungszeit ist bei einer Selektivität von 0,1% um den Faktor 3,63 länger als bei einer Selektivität von 100%. Dies ist ein deutlicher Performance-Unterschied. Auch normalverteilte RIDs kommen dem Radix-Cluster Algorithmus entgegen, den normalverteilte RIDs sind aufeinander folgende RIDs.

4.3.3 Auswirkungen der Trefferquote auf die Laufzeit des Radix-Cluster Algorithmus

Im Folgenden wird getestet wie sich die Variation der Trefferquote auf die Laufzeit des Radix Schnittmengenberechnungsverfahren auswirkt.
Hypothese: Die Variation der Trefferquote hat keinerlei Auswirkungen auf die Laufzeit des Radix-Cluster Algorithmus, wenn dabei die Anzahl der RIDs und die Selektivität unverändert bleiben.
Konstruktion geeigneter Testmengen: Es werden Testmengen mit jeweils 128 Millionen RIDs generiert. Dabei sind die RIDs normalverteilt und die Selektivität beträgt 100%. Die Trefferquote wird variiert. Die rid-Listen sind sortiert.
Überlegungen: Die Selektivität und die Anzahl der RIDs der verschiedenen Schnittmengen ändert sich nicht. Durch diese Faktoren ist keine Auswirkung auf die Laufzeit zu erwarten. Einzig die Trefferquote wird variiert. Es wird nicht erwartet, dass die Trefferquote einen Einfluss auf die Laufzeit des Radix Algorithmus hat.
Ergebnisse zur Hypothese: Die Messergebnisse finden sich im Anhang in Tabelle A.3 und visualisiert in Abbildung 4.4 in diesem Abschnitt. Die zuvor aufgestellte Hypothese bezüglich der Auswirkungen der Trefferquote auf die Laufzeit des Radix Algorithmus kann nicht angenommen werden. Es ist zu Erkennen, dass der Anstieg der Trefferquote die Laufzeit der Schnittmengenberechnung steigen lässt. Eine Erhöhung der Trefferquote um 25 Prozentpunkte führt zu einer ungefähren Verdopplung der Laufzeit für die Schnittmengenberechnung. Interessant ist, dass eine niedrige Trefferquote nicht automatisch für eine niedrige Gesamtlaufzeit spricht. Die Laufzeitmessung mit der Trefferquote von 50% führte zum niedrigsten Messwert. Die übrigen Laufzeitmessungen variieren nur in einem Rahmen von 23 Millisekunden. Den größten Unterschied in der Laufzeit macht bei 50% Trefferquote die Vorbereitungs- bzw. Initialisierungsphase aus. Hier operiert der Radix Cluster Algorithmus deutlich schneller als unter den anderen Trefferquoten. Auch bei Trefferquoten von 75 und 100% ist die Initialisierungsphase in ihrer Ausführungszeit deutlich kürzer als bei 25%, doch wird dieser Zeitvorsprung in der Phase der Schnittmengenberechnung wieder aufgebraucht. Eine Erklärung für diese Laufzeitunterschiede kann nur in der Bildung der Cluster liegen. Anscheinend sind die Cluster für die Schnittmengenberechnung

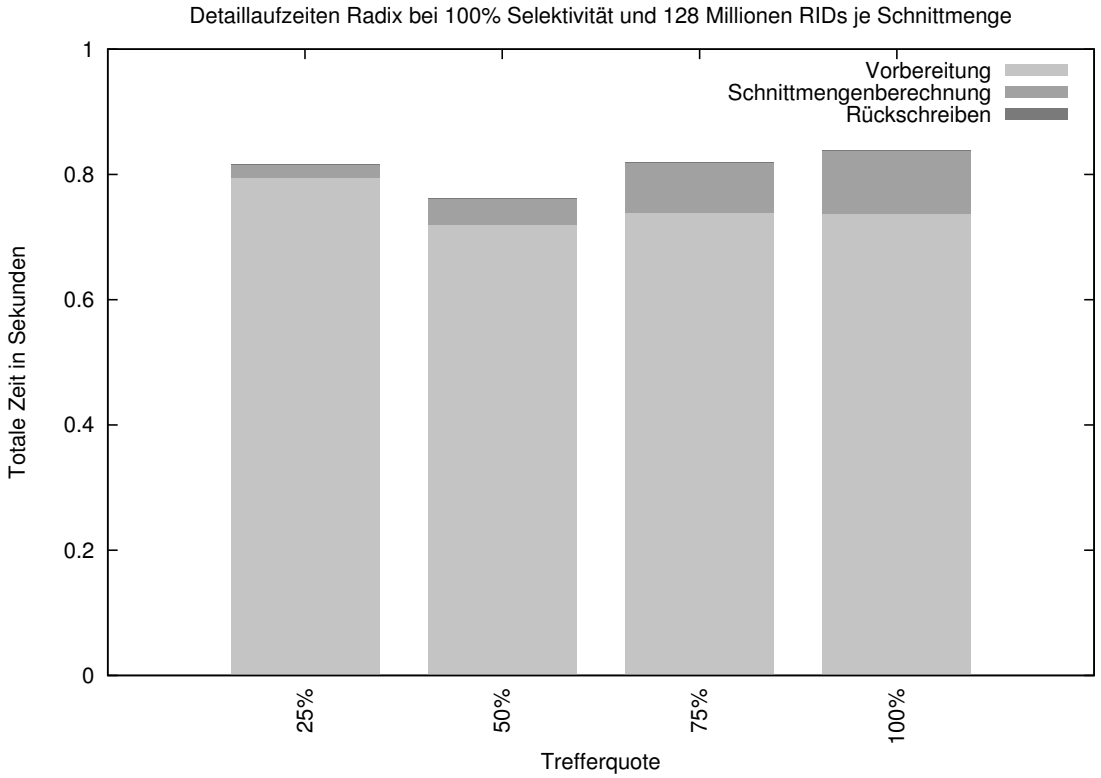

Abbildung 4.4: Laufzeitvergleich bei variierender Trefferquote.

bei niedrigerer Trefferquote effizienter als bei höheren Trefferquoten. Das Clustering in der Vorbereitungsphase hingegen ist bei geringer Trefferquote anscheinend aufwendiger als bei hohen Trefferquoten.

4.3.4 Zusammenfassung

In den voran gegangenen drei Tests ließen sich einige Erkenntnisse zum Radix-Cluster Algorithmus gewinnen. Im ersten Test wurde die Auswirkung von sortierten und unsortierten rid-Listen auf die Laufzeit des Algorithmus untersucht. Dabei war zu erkennen, dass der Radix Algorithmus auf sortiertem Input deutlich schneller operiert als auf unsortiertem Input. Die Laufzeit von unsortierten Input überstieg die Laufzeit bei sortiertem Input um den Faktor 2,26 bis 2,89. Der Aufwand des C++-Standardsortierverfahrens würde den Performance-Vorteil des Radix Algorithmus auf sortieren rid-Listen gegenüber unsortierten rid-Listen dennoch übersteigen und ist daher nicht empfehlenswert (siehe Sortieraufwand Tabelle 4.2). Danach wurde im zweiten Test die Selektivität variiert. Daraus folgte die Erkenntnis, dass das Radix Schnittmengenberechnungsverfahren bei rid-Listen mit hohem Selektivitätswert besser operiert als auf rid-Listen mit niedriger Selektivität. Der Unterschied der Laufzeit fußte hierbei auf einer verlängerten Initialisierungsphase, der Unterschied in der Laufzeit zwischen 0,1% und 100% war dabei mit dem Faktor 3,63 recht deutlich. Zu-

letzt, im dritten Test, wurde dann die Auswirkung der Trefferquote auf die Laufzeit des Algorithmus untersucht. Aus der Auswertung der Messergebnisse folgte die Erkenntnis, dass eine Trefferquote von 50% die Laufzeit positiv im Vergleich zu einer Trefferquote von 25, 75 oder 100% beeinflussen kann. Als optimale Einsatzzwecke für den Radix-Cluster Algorithmus haben sich damit sortierte rid-Listen mit hoher Selektivität und einer Trefferquote von 50% herausgestellt. Die Verwendung des C++-Standardsortierverfahrens sollte dem direkten Einsatz des Radix Algorithmus auf unsortierten Daten dennoch nicht vorgezogen werden.

4.4 SIMD-Algorithmus

In folgendem Abschnitt wird der SIMD-Algorithmus näher betrachtet. Dabei wird zu Vergleichszwecken auch immer die Standard-C++-Schnittmengenberechnung (std::set_intersection) herangezogen. Beide Algorithmen arbeiten auf unkomprimierten Integern. Diese Integer sind die RIDs. Der SIMD Algortihmus basiert dabei auf den Ausführungen in [SWL11].

4.4.1 Variation der Selektivität

Ein erster Test, analog zu dem Test in 4.3.1, ist für den SIMD Algorithmus und den Standardalgorithmus nicht möglich. Daher werden die beiden Schnittmengenberechnungsverfahren im ersten Test hinsichtlich der Auswirkung der Selektivität auf die Gesamtlaufzeit untersucht.

Hypothese: Eine höhere Selektivität führt zu einer bessern, also kürzeren, Gesamtlaufzeit der Algorithmen.

Konstruktion geeigneter Testmengen: Es werden Testmengen mit gleichverteilten RIDs bei einer Trefferquote von 100% generiert. Dabei wird die Selektivität die Stufen 0,1%, 1%, 10% und 100% annehmen. Die Anzahl der RIDs ist aufgrund der niedrigsten Selektivität auf 4.294.967 RIDs je rid-Liste begrenzt.

Überlegungen: Da eine höhere Selektivität dazu führt, dass die RID-Werte näher beieinander liegen, operieren die skalaren Algorithmen effizienter.

Ergebnisse zur Hypothese: Im Anhang in Tabelle A.4 sind die Messergebnisse der Laufzeitmessung zu dieser ersten Hypothese zu finden. In den Abbildungen 4.5 und 4.6 sind die Messerwerte visualisiert. Die Messergebnisse sind konstant, für beide Verfahren zur Schnittmengenberechnung über alle Selektivitäten hinweg. Aufgrund der Verwendung von „nur“ 4.294.967 RIDs je Schnittmengen, ist davon auszugehen, dass die Unterschiede der Rundung zum Opfer gefallen sind. Die Betrachtung des arithmetischen Mittels der Messergebnisse zeigte, für steigende Selektivitätswerte, kürzere durchschnittliche Laufzeiten. In Abbildung 4.5 ist die Laufzeit der Algorithmen detailliert geplottet worden. Für das Standard Intersection-Vefahren ist keine Unterteilung der Gesamtlaufzeit zu den Phasen

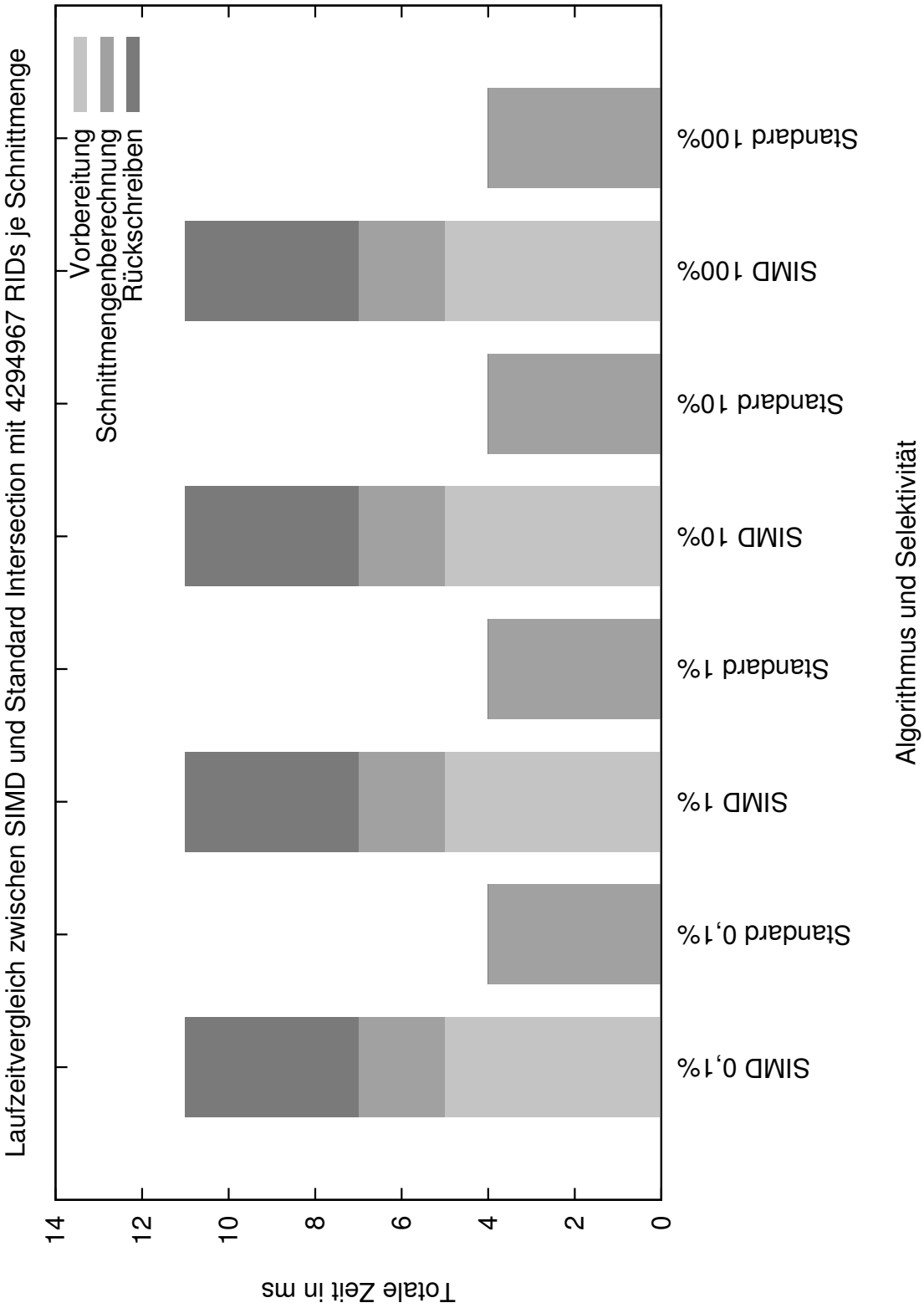

Abbildung 4.5: Laufzeitenvergleich des SIMD-Algortihmus mit dem Standard-C++-Intersection-Verfahren bei variierender Selektivität.

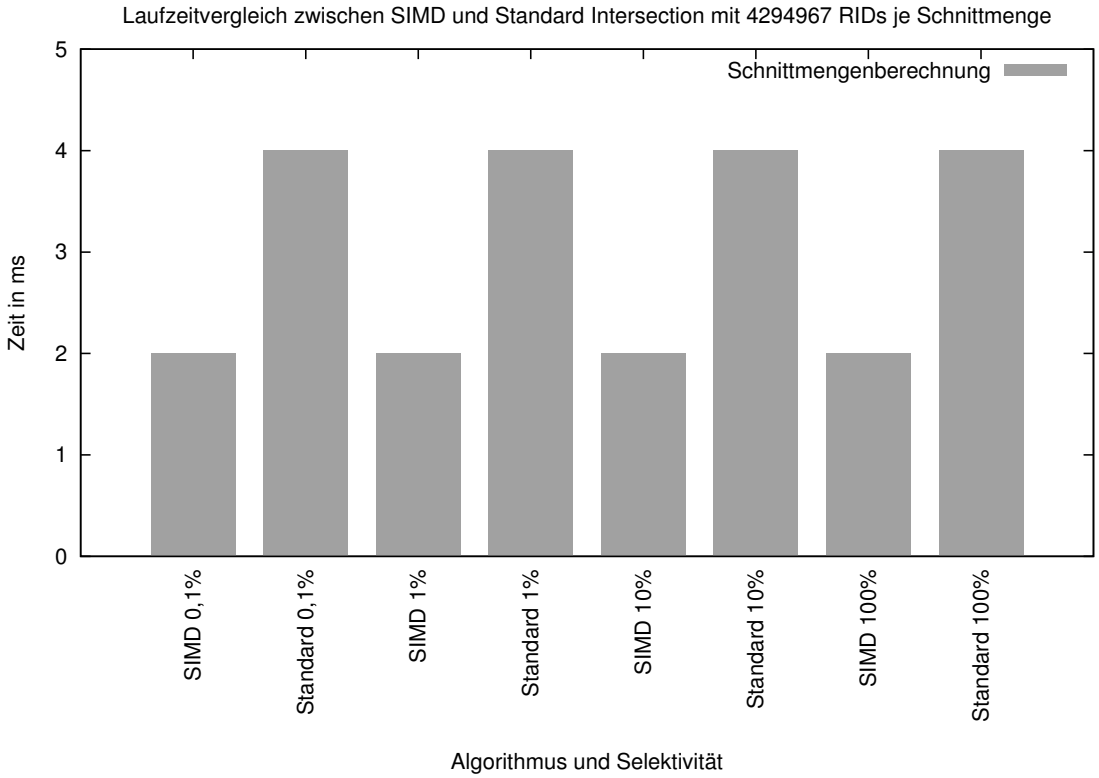

Abbildung 4.6: Vergleich der Berechnungszeit der Schnittmenge bei variierender Selektivität.

der Vorbereitung, der Schnittmengenberechnung und der Rückschreibarbeit möglich. In Abbildung 4.6 sind die reinen Zeiten der Schnittmengenberechnung nebeneinander aufgetragen. Es ist zu erkennen, dass der SIMD-Algorithmus die Schnittmenge um dem Faktor 2 schneller berechnet als die Standard Intersection. Betrachtet man die Algorithmen im Gesamten, so ist der SIMD Algorithmus deutlich langsamer in seiner Gesamtlaufzeit als das Standard Verfahren zur Schnittmengenberechnung. Während der SIMD Algorithmus mit den Vor- und Nacharbeiten auf eine Gesamtlaufzeit von 11 Millisekunden kommt, so kommt std::set_intersection auf eine Laufzeit von lediglich 4 Millisekunden (Faktor 2,75). Die oben aufgestellte Hypothese kann bei den vorliegenden Messergebnissen nicht angenommen werden. Die Hypothese kann allerdings auch nicht abgelehnt werden, denn die Messergebnisse wiedersprechen der Hypothese keineswegs. Die betrachtung der durchschnittlichen Laufzeit gegenüber den Medianwerten stützte die obige Hypothese. Die Hypothese müsste nur dann abgelehnt werden, wenn die Laufzeit bei höherer Selektivität steigen würde. Da dies nicht der Fall ist, kann die Hypothese nicht abgelehnt werden.

4.4.2 Variation der Trefferquote

In dem zweiten Test wird nun die Trefferquote variiert. Die Selektivität und die Anzahl der RIDs bleibt konstant.

Hypothese: Eine hohe Trefferquote wirkt sich negativ auf die Gesamtlaufzeit der Schnittmengenberechnung aus, wenn die Selektivität und die Anzahl an RIDs unverändert bleiben.

Konstruktion geeigneter Testmengen: Es werden Testmengen mit gleichverteilten RIDs konstruiert, die 128 Millionen Elemente enthalten und deren Selektivität immer 100% beträgt.

Überlegungen: Eine niedrigere Trefferquote führt dazu, dass die rid-Liste versetzt zu einander generiert werden. Dadurch muss der Anfang der ersten rid-Liste und das Ende der zweiten rid-Liste gar nicht mehr aktiv miteinander Verglichen werden. Für die Berechnung der Schnittmenge ist daher nur ein Teilstück, also eine viel kleiner Liste als die ursprüngliche, von Bedeutung. Außerdem sinkt die Anzahl der Treffer und damit die Anzahl an RIDs, welche in das Ergebnis geschrieben werden müssen. Dies verringert die Arbeit in der Rückschreib-Phase.

Ergebnisse zur Hypothese: Im Anhang in Tabelle A.5 sind die Messergebnisse der Laufzeitmessung zu dieser zweiten Hypothese zu finden. In Abbildung 4.7 sind die Messerwerte visualisiert. In Abbildung 4.5 ist gut zu erkennen, dass die Laufzeit des Algorithmus mit

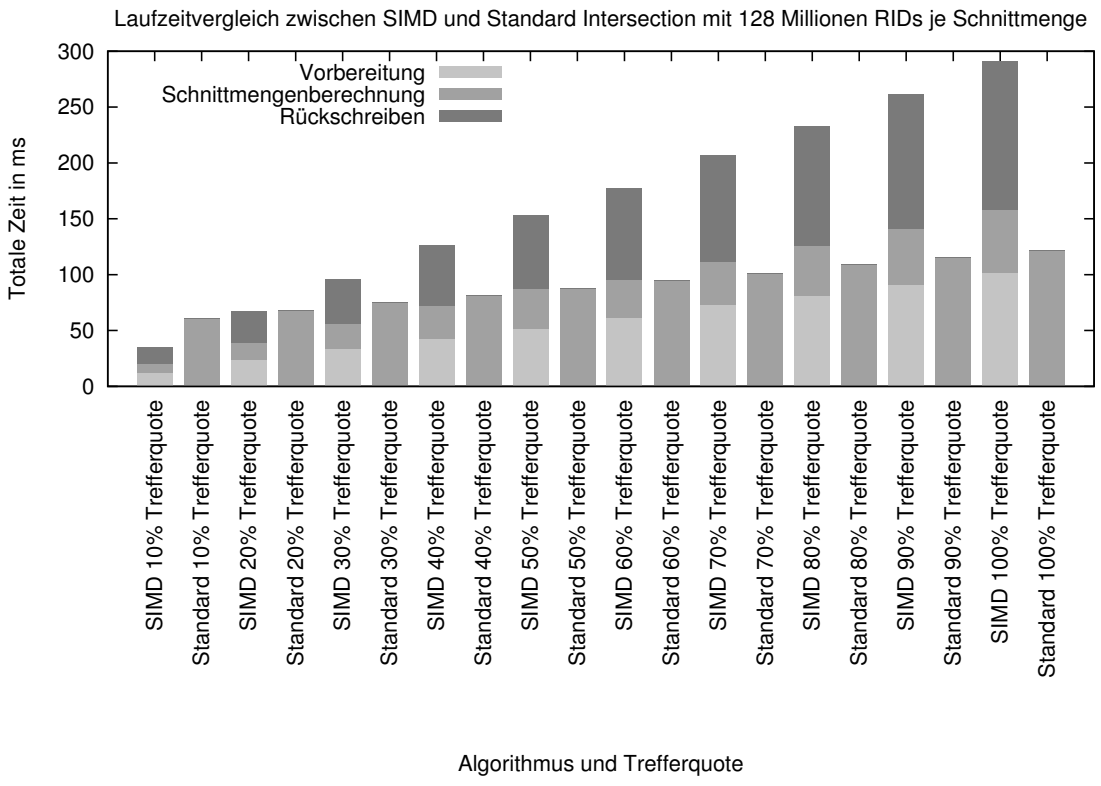

Abbildung 4.7: Detaillierter Vergleich der Laufzeit zwischen SIMD und set_intersection bei variierender Trefferquote.

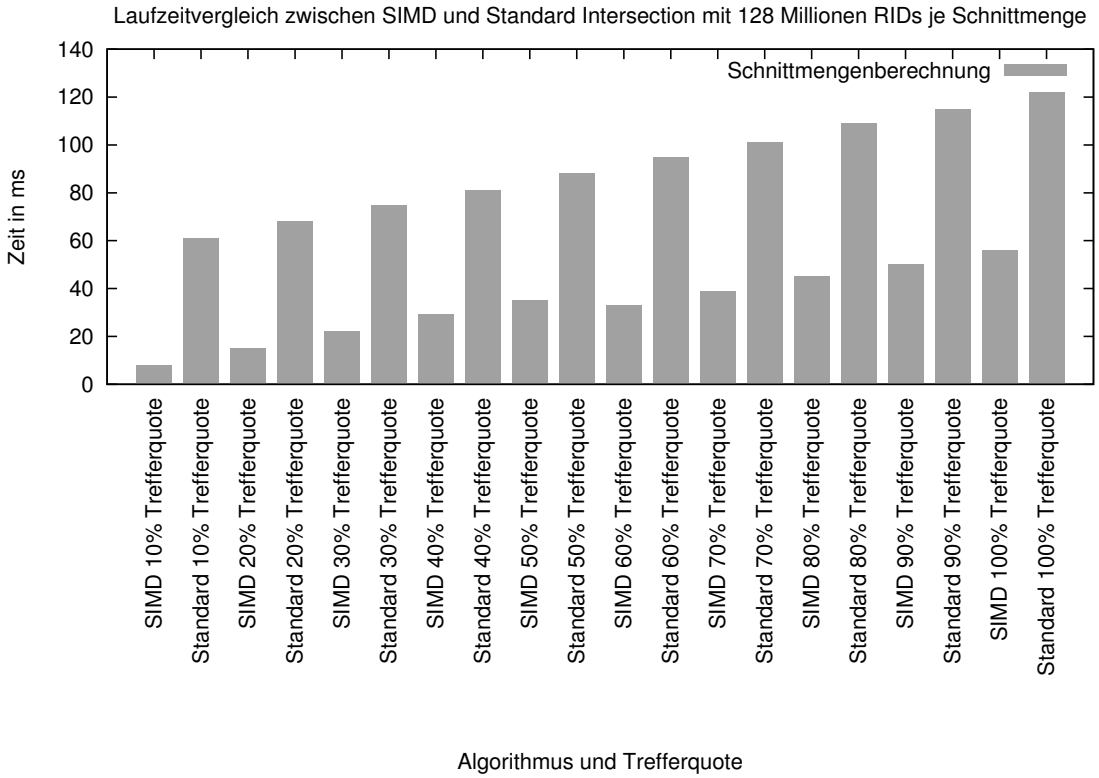

Abbildung 4.8: Vergleich der Berechnungszeit zwischen SIMD und set_intersection bei variierender Trefferquote.

steigender Trefferquote linear an Laufzeit zunimmt. In allen Messphasen ist ein Anstieg der Laufzeiten zu erkennen. Der Anstieg der Laufzeit für Rückschreibearbeiten ist dabei am stärksten ausgeprägt. Die Gesamtlaufzeit des SIMD Algorithmus steigt von 10% zu 100% um den Faktor 8,31 an. Der Zeitaufwand für die Vorbereitung stieg um den Faktor 8,5, der Aufwand für die Berechnung der Schnittmenge um 7 und die Laufzeit für Rückschreibearbeiten stieg um den Faktor 8,87 an. Der Anstieg in der Vorbereitungsphase und in der Rückschreibphase ist also deutlich stärker ausgeprägt als der Anstieg der Berechnungszeit. Die Laufzeit des set_intersection-Verfahrens steigt mit höherer Trefferquote ebenfalls an. Die Laufzeit stieg dabei zwischen einer Trefferquote von 10% und 100% um den Faktor 2 an. Der Anstieg um den Faktor 2 ist dabei deutlich geringer als der Anstieg der Laufzeit für den SIMD-Algortihmus, welcher 8,31 beträgt.
Betrachtet man die reine Zeit zur Schnittmengenberechnung des SIMD-Algorithmus, so liegt die Zeit für den SIMD-Algorithmus deutlich unter der der Standard Schnittmengenberechnung. Gerade bei 10%-Trefferquote benötigt das Standardverfahren das 7,625-fache an Zeit und bei 100%-Trefferquote immer noch das 2,18-fache an Zeit im Vergleich zum SIMD Verfahren.
Eine niedrigere Selektivität kommt beiden hier gegenübergestellten Verfahren bezüglich der Laufzeit entgegen. Die oben aufgestellte Hypothese kann angenommen werden.

4.4.3 Verwendung von Zipf-verteilten RIDs

Da die Generierung der gleichverteilten Daten RID-Werte in regelmäßigen Abständen produziert, liegt die Vermutung nahe, dass der Standard-C++-Algorithmus aufgrund von Branch Prediction die besseren Laufzeit gegenüber dem SIMD-Algorithmus erreicht. Aus diesem Grund wir nun in diesem dritten Test die Laufzeit der beiden Schnittmengenberechnungsverfahren bei Zipf-verteilten RIDs überprüft. Für k wurde 100 gewählt und für α 1. Als Selektionsbedingung wurde von kleiner 2 bis kleiner 6, sowie von größer 2 bis größer 5 variiert. Die Trefferquote lässt sich bei der Generierung von Zipf-verteilten Zufallszahlen nicht beeinflussen. Die Ergebnisse der Laufzeitmessung finden sich in Tabelle A.6 im Anhang. In Abbildung 4.9 sind die Messungen visualisiert. Die Laufzeit des SIMD Algorithmus

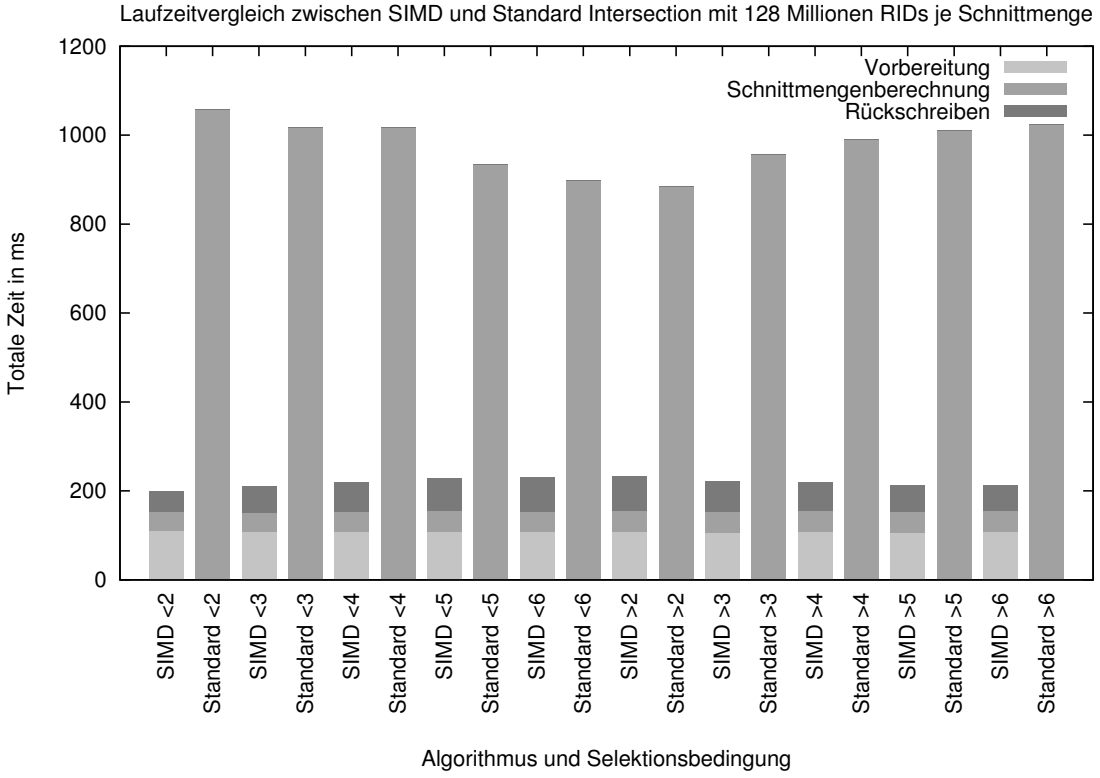

Abbildung 4.9: Vergleich der Berechnungszeit zwischen SIMD und set_intersection bei Zipf-verteilten RIDs.

variiert bei 128 Millionen RIDs je Schnittmenge zwischen 199 und 233 Millisekunden. Das Standard-Schnittmengenverfahren hingegen zwischen 884 und 1058 Millisekunden. Im Falle der Selektion „es werden RIDs gesetzt wenn 1 gezogen wird“ (<2) übersteigt die Laufzeit des Standard-Algorithmus den SIMD Algorithmus um den Faktor 5,32. Im schlechtesten Fall (<6) übersteigt die Laufzeit des Standard-Verfahrens die SIMD-Schnittmengenberechnung immer noch um den Faktor 3,79. Außerdem unterstreicht der Test die Erkenntnisse aus dem ersten SIMD Test. Die Selektivität scheint wenn nur einen geringen Einfluss auf die

Laufzeit der Algorithmen zu haben. Bei der Konstruktion von rid-Listen mit $< n$ gegenüber $< n + 1$ mit $n = \{2, \ldots, 6\}$ ist es so, dass weniger Lücken (nicht gesetzte RIDs) in der rid-Liste entstehen. Dies vergrößert die Chance auf mehr Elemente in der Schnittmenge. Die Ergebnisse aus Test 2 zeigten, dass eine höhere Trefferquote einen negativen Einfluss auf die Laufzeit der Algorithmen hat. Dies würde sich zumindest auch für den SIMD-Algorithmus aus den Messergebnissen folgern lassen. Für die Standard-Intersection allerdings nicht.
Der SIMD Algorithmus ist dem Standard-C++-Schnittmengenberechnungsverfahren also deutlich vor raus, wenn keine Branch Prediction greifen kann.

4.4.4 Zusammenfassung

In den beiden Tests in Abschnitt 4.4 stellte sich heraus, dass die Trefferquote einen deutlichen Einfluss auf die Laufzeit der Algorithmen hat. Eine niedrige Trefferquote ist absolut von Vorteil. Das Variieren der Selektivität hat nach den Messergebnissen keinen Einfluss auf die Laufzeit der Algorithmen. Die reine Zeit zur Berechnung der Schnittmenge liegt für den SIMD-Algorithmus stets unter der Zeit der Standard Intersection.
Vergleicht man die Messergebnisse mit den Ergebnissen der experimentellen Evaluation aus [SWL11], so geht aus Abbildung 8 in [SWL11] hervor, dass die Selektivität keinen Einfluss auf die Laufzeit der Schnittmengenberechnung zwischen 0% und 90% hat. Lediglich zwischen 90 und 100% ist eine leichte Abnahme der Laufzeit zu erkennen. Auch ist zu erkennen, dass der Zeitvorsprung des SIMD Algorithmus gegenüber den anderen Implementationen mit steigender Selektivität abnimmt. Dieser Effekt konnte nicht bestätigt werden. Die Selektivität hat demnach keine großen negativen Auswirkungen auf die Performance des SIMD Algorithmus und auch nicht auf die Standard Intersection. Ein Anstieg der Trefferquote führt allerdings zu schlechterer Performance. Die Verwendung des SIMD Algorithmus oder der Standard Intersection empfiehlt sich daher eher bei Datensätzen mit geringer Trefferquote.

Kritische Einschätzung

[SWL11] misst explizit in allen Experimenten nur die Ausführungszeit für die Schnittmengenberechnung („In all experiments, we measured the wall-clock time for the intersection process only.“ [SWL11, S. 6]). Die unumgänglichen Vor- und Nacharbeiten finden allerdings keinerlei Beachtung. Davon unangetastet bleibt der Performancevorteil des SIMD Verfahrens in der Berechnungsphase, doch nützt ein Geschwindigkeitsvorteil in der Phase der Schnittmengenberechnung alleine nichts. Sollte es möglich sein, die Vor- und Nacharbeiten des SIMD-Algorithmus zu umgehen, z. B. indem die Datenstruktur des Datenbanksystems für SIMD angepasst wird, so ist der SIMD-Algorithmus den anderen skalaren Implementationen vorzuziehen. Ein Intel-Befehl der es möglich macht, direkt mehr

Bits zu vergleichen, sodass die Aufteilung in höhere und niedrige Bits weg fallen könnte, würde ebenfalls den Weg für den Einsatz der SIMD Schnittmengenberechnung ebnen. Trotz des Aufwandes in den Phasen vor und nach der Schnittmengenberechnung zeigt der SIMD-Algorithmus in Test 3 eine deutlich kürzere Berechnungszeit als die Standard-C++-Schnittmengenberechnung. In Fällen in den keine Branch Prediction greifen kann ist der SIMD-Algorithmus definitiv eine gute Wahl eines skalaren Algorithmus.

4.5 Bitmap

In diesem Abschnitt werden die Bitmap-basierten Schnittmengenberechnungsverfahren evaluiert. Einerseits wird die Bitmap-Intersection ohne Kompression Bitmap, Bitmap Algorithmus oder Bitmap Intersection genannt. Andererseits wird der Einsatz der WAH-kompression mit 128 Bit und 32 Bit untersucht. Diese Verfahren zusammen mit der eigentlichen Bitmap Intersection werden als WAH128 und WAH32 oder ähnlich bezeichnet werden.
Die folgenden Tests sollen die drei Bitmap-basierten Schnittmengenberechnungsverfahren gegenüberstellen. Dabei soll herausgefunden werden, unter welchen Bedingungen sich der Einsatz welcher Kompression lohnt. Außerdem soll erkannt werden, unter welchen Gesichtspunkten die Bitmap-basierten Verfahren besonders effizient oder auch ineffizient arbeiten.

4.5.1 Laufzeitverhalten bei sortiertem Input gegenüber unsortiertem Input

Da die Bitmap Intersection ohne Kompression sowohl mit sortierten rid-Listen wie auch mit unsortierten rid-Listen arbeitet, wird in diesem ersten Test zu den Bitmap-basierten Algorithmen die Auswirkung auf die Gesamtlaufzeit, bezüglich sortiertem gegenüber unsortiertem Input, untersucht. Der Test gestaltet sich analog zu dem Test in Abschnitt 4.3.1.
Hypothese: Unsortierte rid-Listen als Input wirken sich negativ auf die Gesamtlaufzeit der Bitmap Intersection. im Vergleich zu sortierten rid-Listen, aus.
Konstruktion geeigneter Testmengen: Es werden normalverteilte RIDs mit einer Selektivität und einer Trefferquote von 100% generiert. Beide Schnittmengen enthalten jeweils gleich viele Elemente. Die Anzahl der Elemente variiert zwischen 32, 64, 96 und 128 Millionen RIDs pro Schnittmenge.
Überlegungen: Dadurch, dass die rid-Listen nicht sortiert sind, ist die Konstruktion der Bitmap aufwendiger. Bei sortieren rid-Listen ist die Erstellung der Bitmap innerhalb eines iterativen Laufs über die rid-Liste möglich. Ein Laufzeitzuwachs von sortiertem zu unsortiertem Input wird nicht in der Berechnungsphase erwartet.
Ergebnisse zur Hypothese: Im Anhang in Tabelle A.7 sind die Messergebnisse der Laufzeitmessung zu dieser ersten Hypothese zu finden. In Abbildung 4.10 sind die Messerwerte

visualisiert. Die oben aufgestellte Hypothese kann angenommen werden. Es ist eindeutig zu

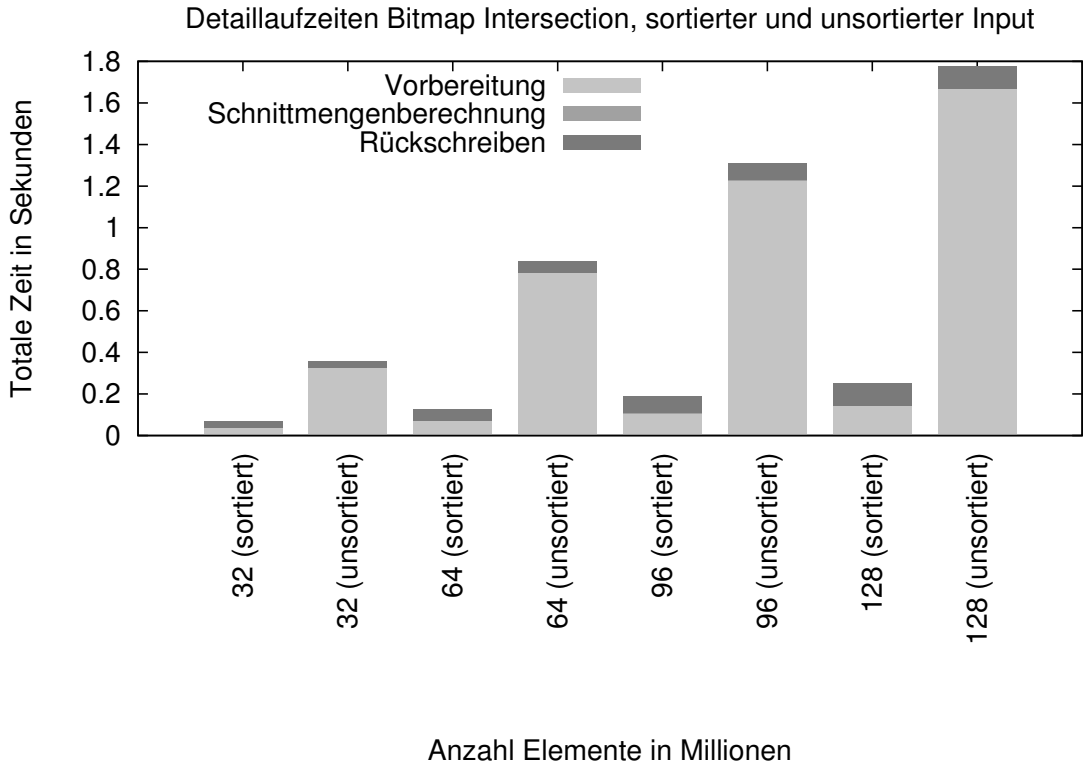

Abbildung 4.10: Detaillierter Vergleich der Laufzeiten des Bitmap Algorithmus bei sortierten und unsortierten rid-Listen als Input.

erkennen, dass der Aufwand der Initialisierungsphase zwischen sortierten und unsortierten Daten deutlich ansteigt. Im Falle von 128 Millionen RIDs je Schnittmenge ist die Laufzeit der Vorbereitungsphase um den Faktor 11,74 höher. Die Zeit der Schnittmengenberechnung unterscheidet sich zwischen sortierten und unsortierten Daten nicht. Auch der Aufwand in der Rückschreibphase, in der die Ergebnisbitmap wieder in eine rid-Liste umgewandelt wird, variiert fast nicht zwischen sortierten und unsortierten Input.
Aufällig ist, dass die Vorbereitungsphase einen Großteil der Gesamtlaufzeit ausmacht. Die Laufzeit der Schnittmengenberechnung an sich ist hingegen sehr gering. Die Berechnungszeit trägt nur in einem so kleinen Rahmen zur Gesamtlaufzeit bei, dass diese in Abbildung 4.10 nicht zu erkennen ist. Bei 128 Millionen RIDs je Schnittmenge beträgt diese lediglich 3 ms. Im Vergleich dazu, benötigt zum Beispiel der SIMD-Algorithmus für die konkrete Berechnung der Schnittmenge 56 Millisekunden (vgl. Abschnitt 4.4.2). Die Laufzeit bei unsortierten Daten ist dennoch kürzer als die Laufzeit bei sortierten RIDs addiert mit dem Zeitaufwand für das Sortieren der rid-Listen nach dem C++-Standardsortierverfahrens. Der Zeitaufwand für das Sortieren von zwei Schnittmengen mit 32 Millionen RIDs übersteigt die Gesamtlaufzeit der Bitmap Intersection bei 128 Millionen unsortierten RIDs je Schnittmenge. Der Sortieraufwand findet sich in Tabelle 4.2.

4.5.2 Laufzeitvergleich bei verschiedenen Selektivitätswerten

In diesem zweiten Test soll die Selektivität der Schnittmengen variiert werden. Dabei sollen Erkenntnisse darüber gewonnen werden, wie sich die Selektivität auf die Schnittmengenberechnung auswirkt.

Hypothese: Eine niedrige Selektivität wirkt sich negativ auf die Gesamtlaufzeit der Bitmap-basierten Schnittmengenberechnungsverfahren aus.

Konstruktion geeigneter Testmengen: Es werden gleichverteilte RIDs für die Schnittmengen generiert. Die Trefferquote der beiden Mengen beträgt 100%, die Selektivität variiert zwischen 0,1% und 100%. Die Anzahl der Elemente je Schnittmenge beträgt 4.294.967.

Überlegungen: Eine negative Auswirkung auf die Gesamtlaufzeit bei abnehmender Selektivität wird erwartet, da die nicht gesetzten RIDs in der Bitmap zunehmen. Dadurch steigt die Größe der Bitmap an. Dies führt zu einer längeren Verarbeitungszeit in der Vorbereitungsphase und in der Rückschreibphase. Außerdem werden mehr Vergleiche in der Phase der Schnittmengenberechnung nötig.

Ergebnisse zur Hypothese: Im Anhang in Tabelle A.8 sind die Messergebnisse der Laufzeitmessung zu dieser zweiten Hypothese zu finden. In Abbildung 4.11 und Abbildung 4.12 sind die Messerwerte visualisiert. Auffällig ist, dass die Verfahren ohne Kompression in den

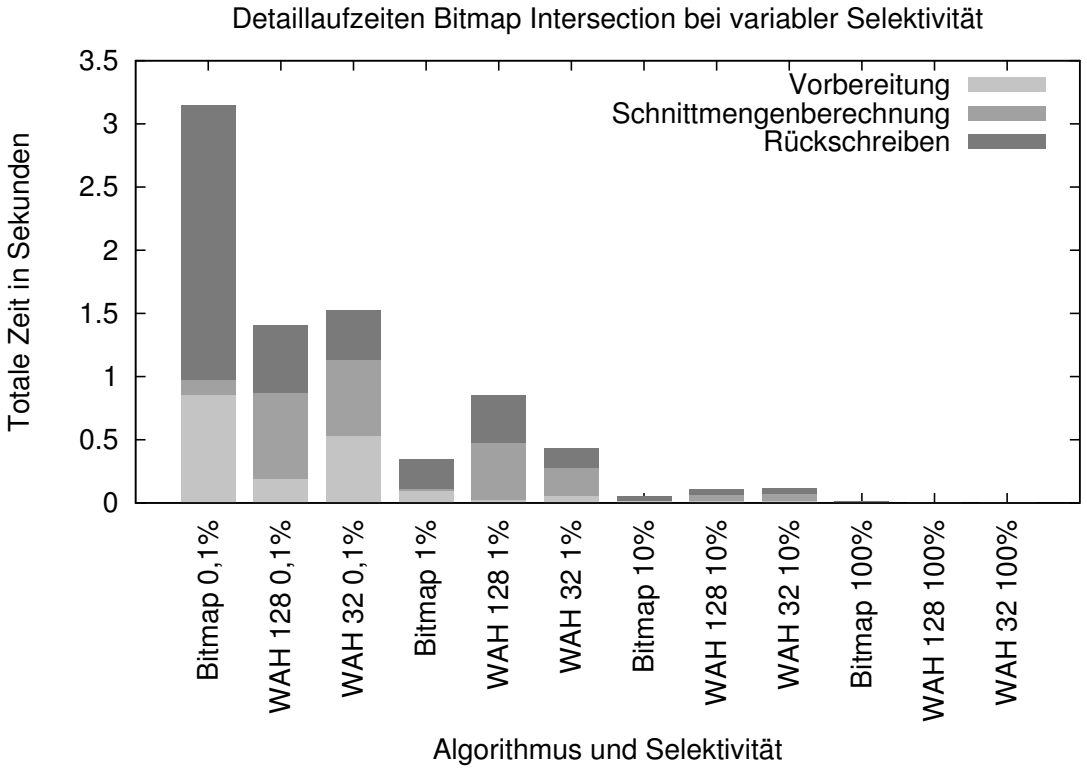

Abbildung 4.11: Detaillierter Vergleich der Laufzeiten der Bitmap Algorithmen bei variierender Selektivität.

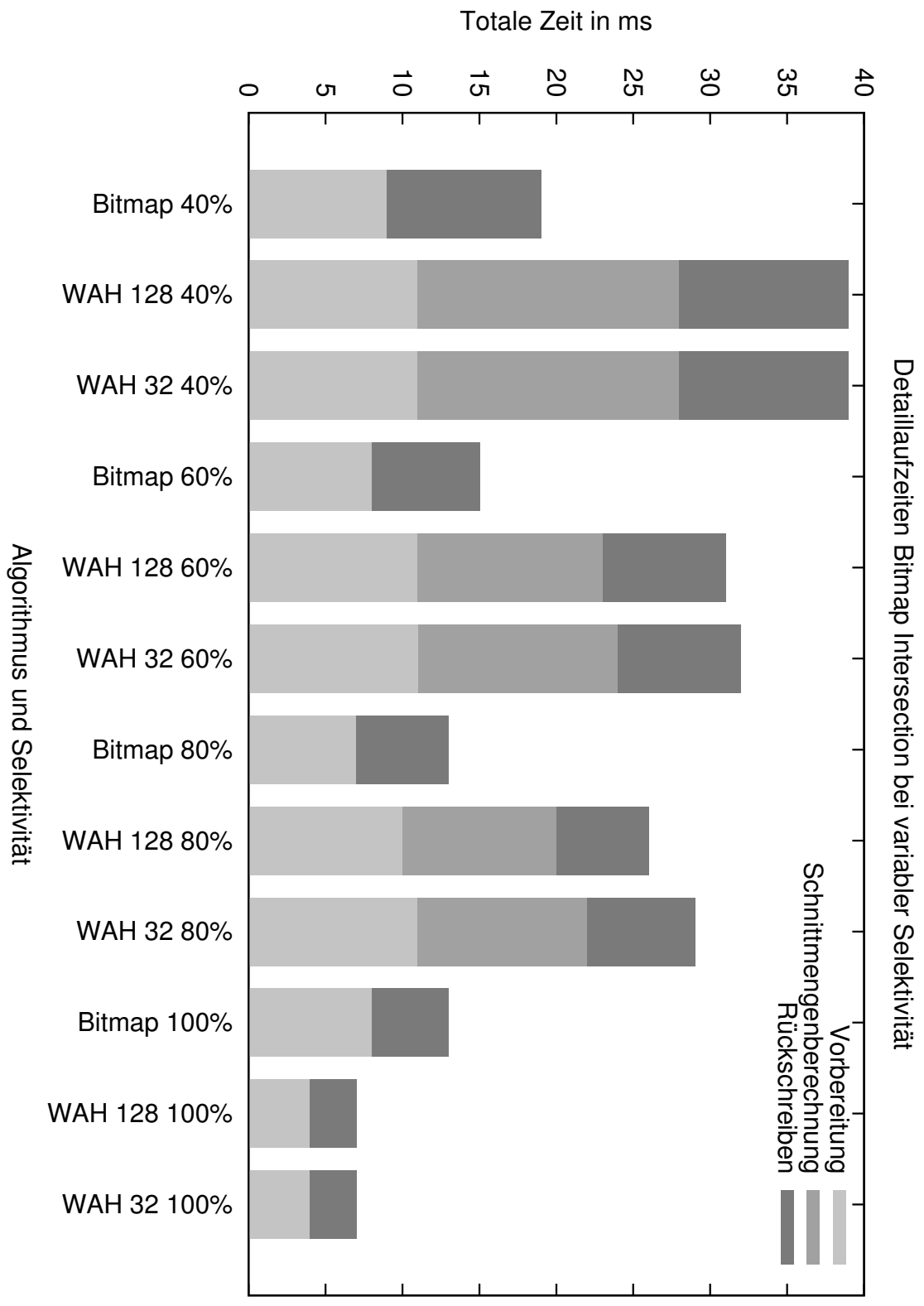

Abbildung 4.12: Detaillierter Vergleich der Laufzeiten der Bitmap Algorithmen bei variierender Selektivität.

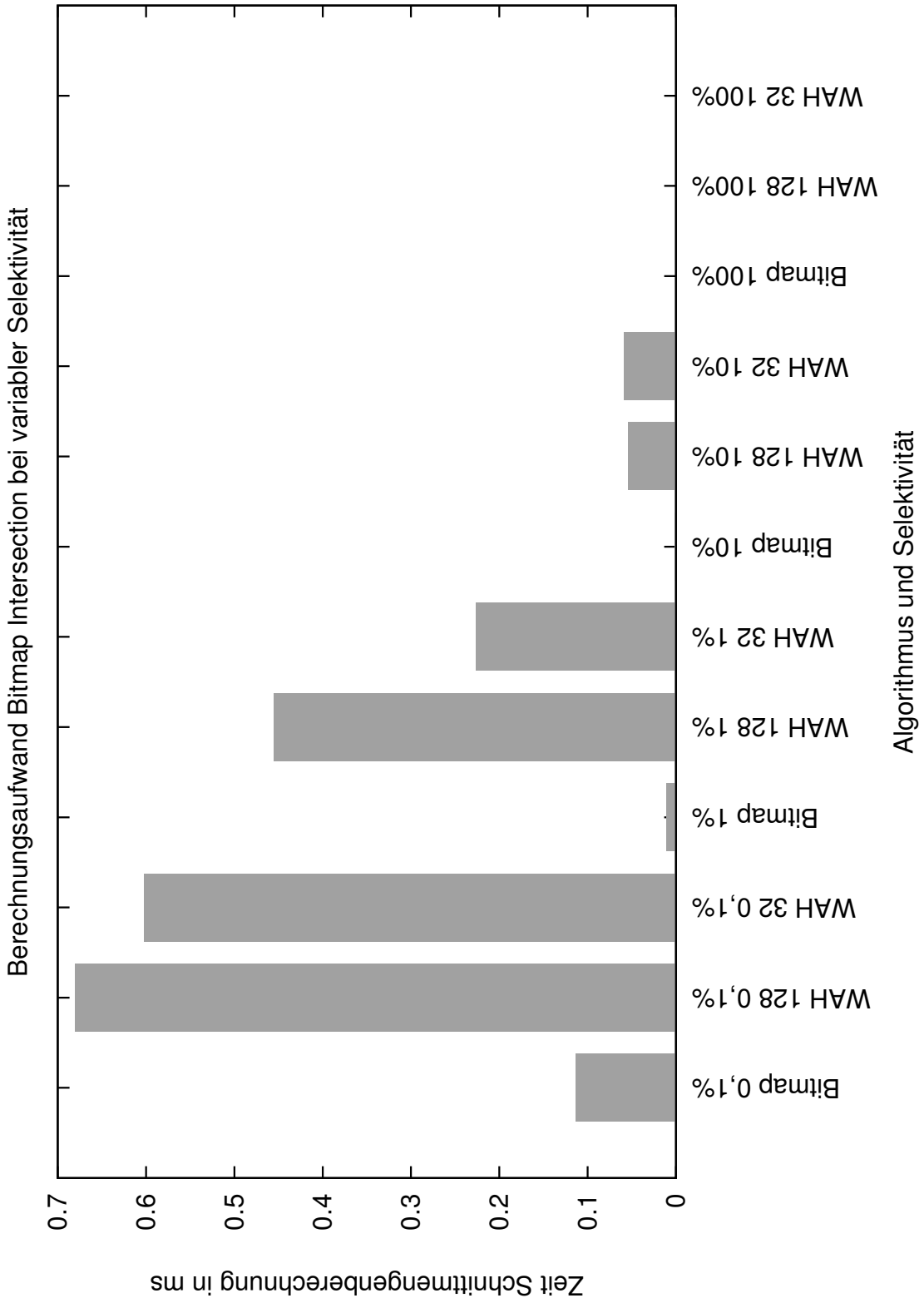

Abbildung 4.13: Detaillierter Vergleich der Berechnungszeit der Bitmap Algorithmen bei variierender Selektivität.

meisten Fällen eine deutlich längere Ausführungszeit aufweisen als die Bitmap Intersection ohne Kompression. Nur bei 100% Selektivität und bei einer Selektivität von 0,1% war die gemessene Performance mit Kompression besser. Bei einer Selektivität von 0,1% waren die Verfahren mit Kompression sogar deutlich besser in der Ausführungszeit. Bitmap Intersection ohne Kompression ist um den Faktor 2,24 (zu WAH128) bzw. um den Faktor 2,06 (zu WAH32) langsamer. Betrachtet man nur die Zeit für die Berechnung der Schnittmenge, so ist diese für Bitmap Intersection ohne Kompression immer geringer als die Berechnungszeit bei Intersection mit Kompression, siehe Abbildung 4.13. Festhalten kann man an dieser Stelle schon einmal, dass die Bitmap Intersection alleine von der Berechnungszeit der Schnittmenge einen Laufzeitvorteil gegenüber den Verfahren mit Kompression hat. Andererseits ist die Verwendung des WAH-Verfahrens bei einer Selektivität von 0,1% fast unumgänglich. Eine mehr als doppelt so lange Laufzeit des Bitmap Verfahrens ist ein sehr deutlicher Performanznachteil. Die Vorteile der Nutzung von Kompression bei Selektivitäten von 100% und geringen Selektivitäten (um 0,1%) liegt an der Kompression an sich. Im Falle von 100% ist jedes Bit gesetzt, daher effizient ein Fill-Word mit dem Wert 1 für alle Bits setzen, die in ihrer Summe um ein vielfaches ihrer Wort-Länge (128 oder 32 Bit) kleiner sind als die Gesamtanzahl an Bits. Auch bei kleinen Selektivitäten kann wieder effizient komprimiert werden, da die Größen der Lücken, also die Anzahl der aufeinander folgenden 0-Bits, zunimmt.
In Test 5 (Abschnitt 4.5.4) wird die Auswirkung von geringen Selektivitätswerten, zwischen 0,1% und 1%, auf die Laufzeit der Verfahren noch einmal näher betrachtet.

4.5.3 Laufzeitvergleich bei verschiedenen Trefferquoten

In diesem dritten Test zu den Bitmap-basierten Schnittmengenberechnungsverfahren soll die Trefferquote der Schnittmengen variiert werden. Dabei sollen Erkenntnisse darüber gewonnen werden, wie sich die Trefferquote auf die Laufzeit der Schnittmengenberechnung auswirkt.
Hypothese: Eine hohe Trefferquote wirkt sich negativ auf die Laufzeit der Bitmap-basierten Algorithmen aus bei ansonsten konstanter Selektivität und konstanter Anzahl an Elementen je Schnittmenge.
Konstruktion geeigneter Testmengen: Es werden Schnittmengen mit je 128 Millionen gleichverteilter RIDs generiert. Die Selektivität liegt bei 100%. Die Trefferquote zwischen den Schnittmengen variiert von 10 bis 100%.
Überlegungen: Da eine höhere Trefferquote eine größere Menge an Treffer nach sich zieht, welche aus der Bitmap zurück zu normalen RIDs konvertiert werden müssen, erhöht sich die Laufzeit bei steigender Trefferquote.
Ergebnisse zur Hypothese: Im Anhang in Tabelle A.9 befinden sich die Messergebnisse zu dieser Hypothese. In Abbildung 4.14 sind die Messerwerte visualisiert. Die zuvor

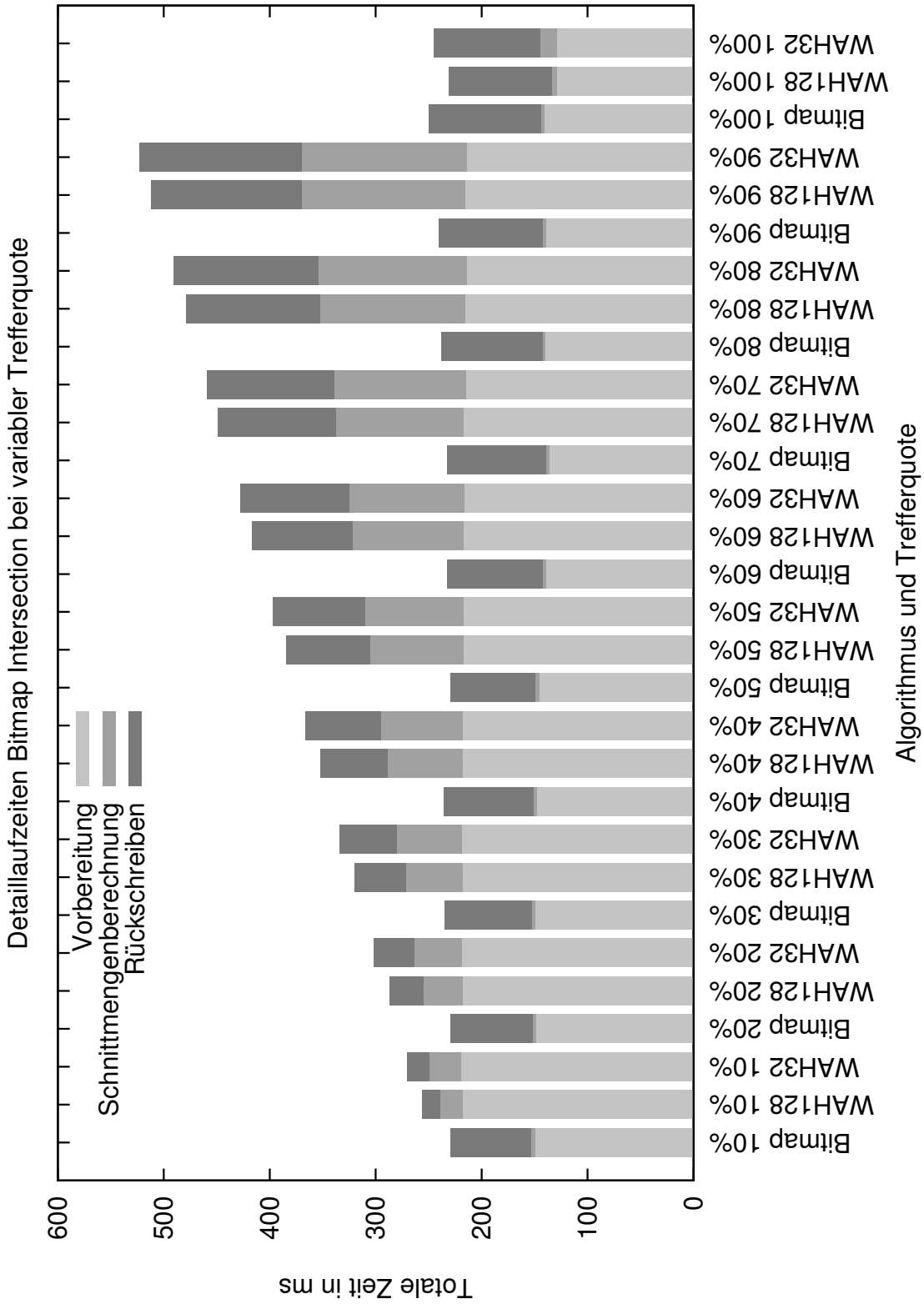

Abbildung 4.14: Detaillierter Vergleich der Laufzeiten der Bitmap Algorithmen bei variierender Trefferquote.

aufgestellte Hypothese kann nicht angenommen werden. Die Bitmap Intersection ohne Kompression zeigt sich unabhängig von der Trefferquote konstant. Bei einer Trefferquote von 100% haben die Schnittmengenberechnungsverfahren mit Kompression die insgesamt niedrigste Gesamtlaufzeit. Im Bereich von 10-90% Trefferquote gelten hingegen die Annahmen der Hypothese. Die Laufzeiten nehmen mit höheren Trefferquoten zu. Dabei wächst vor allem der Aufwand in der Rückschreibphase stark an (um den Faktor 5,71 (WAH128) und 4,76 (WAH32) von 10 zu 90%). Auch die Zeitaufwendung in der Berechnungsphase steigt analog zur Rückschreibphase an. Die Vorbereitungszeit hingegen bleibt unverändert konstant, bis auf den Fall, dass die Trefferquote 100% beträgt. Die Schnittmengenberechnungen mit Kompression waren nur im Falle der 100%-igen Trefferquote schneller in ihrer Gesamtlaufzeit als der Bitmap Algorithmus ohne Einsatz von Kompression. Die Laufzeit von WAH 128 überstieg die Laufzeit des nicht komprimierten Bitmap Verfahrens um den Faktor 2,13 (WAH 32 um 2,18).
Interessant ist, dass der nicht-komprimierte Bitmap Algorithmus unverändert konstante Laufzeiten trotz unterschiedlicher Trefferquote aufweist. Noch überraschender ist, dass die Schnittmengenberechnungen mit Kompression im Falle von 100% deutlich kürzere Laufzeiten aufzeigen als bei anderen Trefferquoten aus höheren Bereichen. Worin die kürzere Laufzeit bei 100%-iger Trefferquote für die Verfahren mit Kompression begründet liegen mag erschließt sich mir nicht. Aus diesem Grund wird die Auffälligkeit bezüglich der drastisch kürzeren Laufzeiten der Verfahren mit Kompression, bei 100% Trefferquote gegenüber 90% Trefferquote, im nachfolgenden Test näher betrachtet.
Es lässt sich festhalten, dass die Trefferquote auf den Bitmap Algorithmus keine merklichen negativen Auswirkungen hat. Das Verfahren lässt sich demnach bei jeglichen Trefferquoten einsetzen. Die Performance der Verfahren mit WAH Kompression wird dagegen deutlich negativ durch die steigende Trefferquote beeinflusst. Im Bereich von niedrigen Trefferquoten hält sich der Performanznachteil aber durch aus in Grenzen.
Es zeigte sich drastische Laufzeitverkürzungen für die Verfahren mit Kompression als die Trefferquote von 90 auf 100% erhöht wurde. In Abbildung 4.15 ist zu erkennen, dass der deutliche Performanz-Gewinn der Verfahren mit Kompression erst bei 100% einsetzt. Die Laufzeit verbessert sich bei den Verfahren mit WAH Kompression von 98% Trefferquote auf 100% Trefferquote um den Faktor 2,33 (WAH128) und 2,24 (WAH32). Vor allem die zur Berechnung der Schnittmenge benötigte Zeit sinkt deutlich. So werden statt 168 Millisekunden (WAH128 bei 98% Trefferquote) nur noch 5 Millisekunden (WAH128, Trefferquote 100%) benötigt. Mit der 32 Bit WAH Kompression fiel die benötigte Zeit von 169 Millisekunden auf 16 Millisekunden.

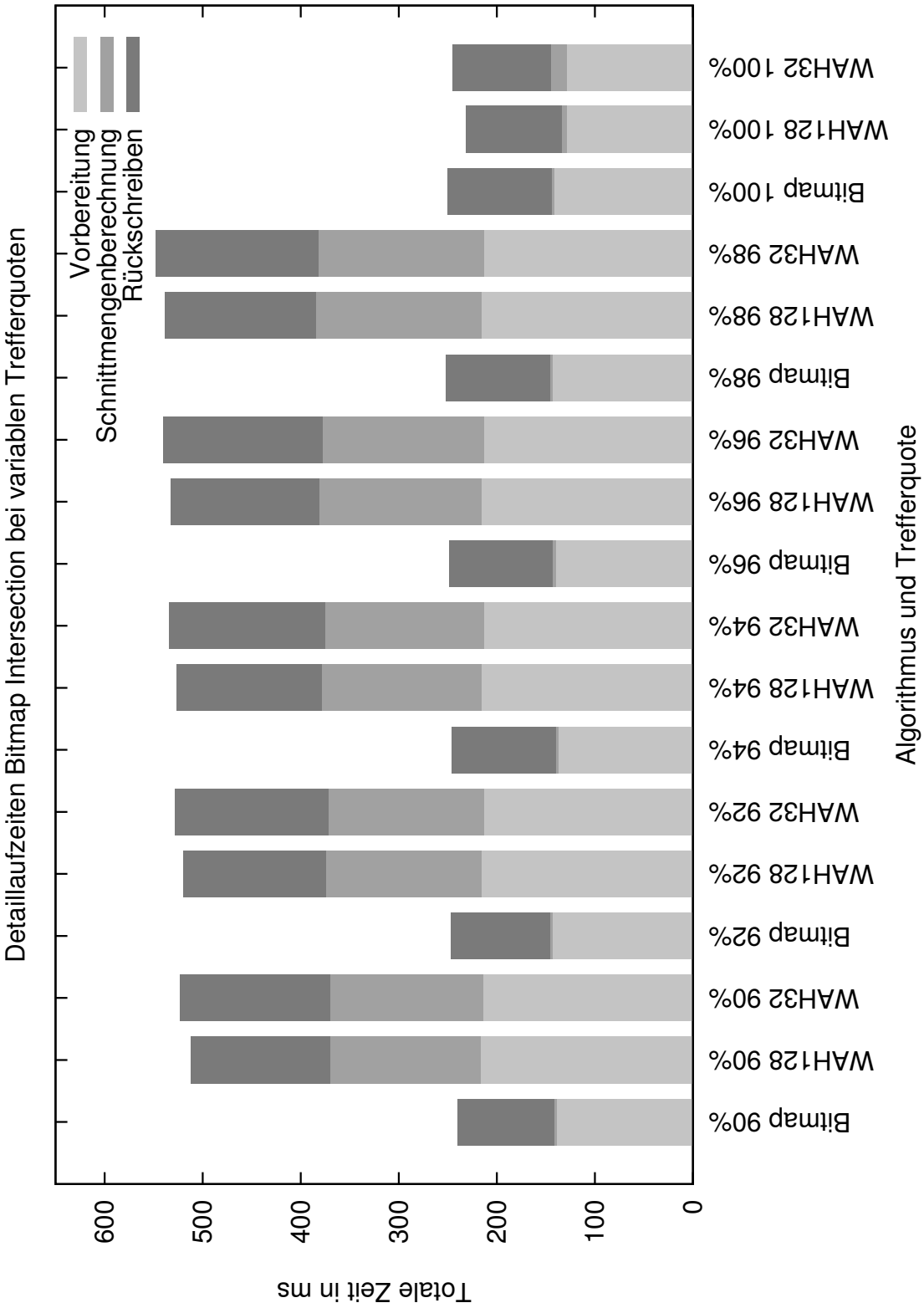

Abbildung 4.15: Detaillierter Vergleich der Laufzeiten der Bitmap Algorithmen bei variierender Trefferquote. Messergebnisse Tabelle A.10 im Anhang.

4.5.4 Variation der Selektivität bei niedrigen Selektivitätswerten

In diesem Test soll der Selektivitätsbereich näher untersucht werden, indem die Bitmap Verfahren mit Kompression dem Verfahren ohne Kompression überlegen sind. In Abbildung 4.11 zeigte sich, dass die Verfahren mit Kompression bei einer Selektivität von 0,1% deutlich kürzere Laufzeiten hatten, als bei 1% Selektivität. In diesem Test sollen nun Erkenntnisse über den Selektivitätsbereich zwischen 0,1% und 1% gewonnen werden. Dazu werden Testmengen wie in Test 2 generiert, nur mit anderen Selektivitätswerten. In Abbil-

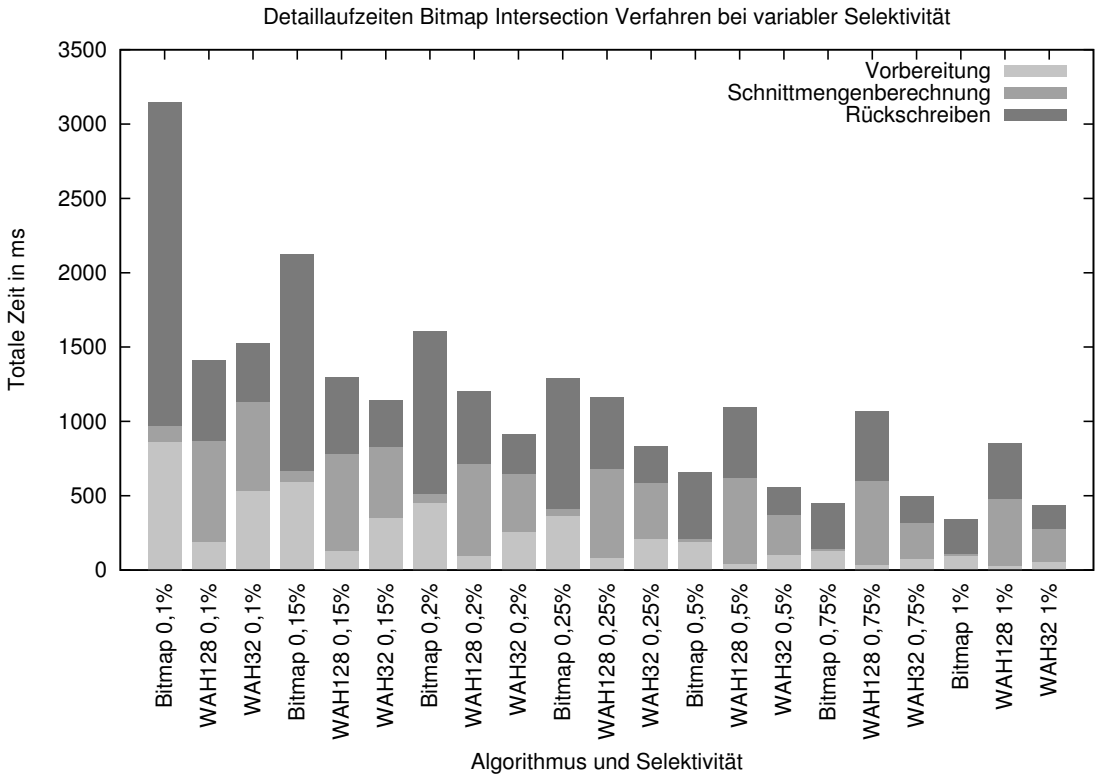

Abbildung 4.16: Detaillierter Vergleich der Laufzeiten der Bitmap Intersection ohne Kompression bei variierender Selektivität. Messergebnisse Tabelle A.11 im Anhang.

dung 4.16 sind die Laufzeiten bei kleinen variierenden Selektivitätswerten abgebildet. Nur bei der Selektivität von 0,1% hat WAH128 eine kürzere Laufzeit als WAH32. Das nicht komprimierte Bitmap Schnittmengenberechnungsverfahren ist ab 0,75% Selektivität den Verfahren mit Kompression von der Laufzeit her vorzuziehen. In Abbildung 4.17 ist nur die Laufzeit in Abhängigkeit von der variierenden Selektivität aufgetragen. Schon zwischen 0,1% und 0,2% Selektivität sinkt die Gesamtlaufzeit um den Faktor 1,96 ab. Der Einsatz der Bitmap Intersection empfiehlt sich also keinesfalls für niedrige Selektivitäten. Es ist mit einem extremen Anstieg der Gesamtlaufzeiten zu rechnen. Die WAH Kompression sollte daher zum Einsatz kommen.
Zusammen mit den Messergebnissen aus dem zweiten Test (Abschnitt 4.5.2) zeigt sich,

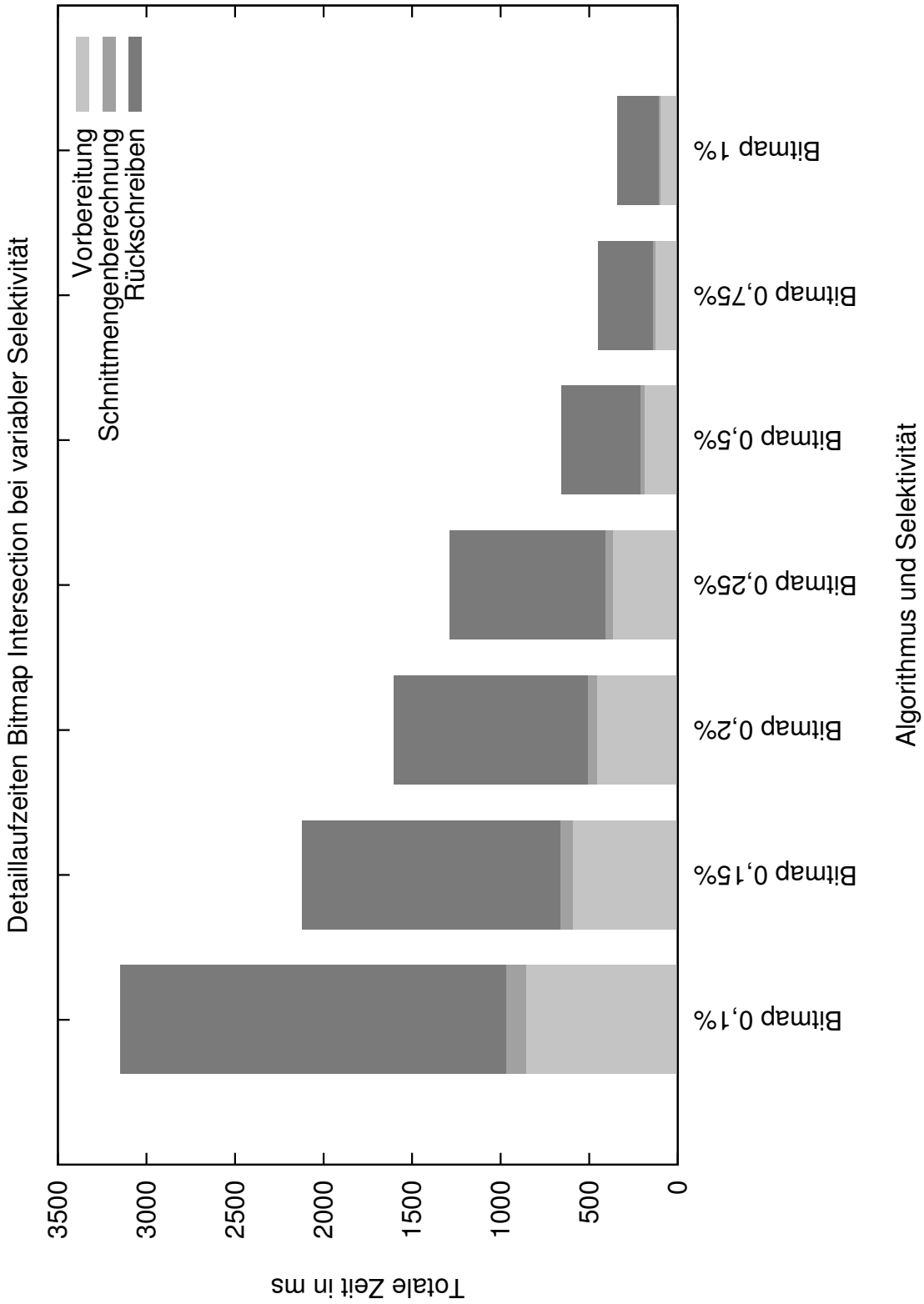

Abbildung 4.17: Detaillierter Vergleich der Laufzeiten der Bitmap Algorithmen bei variierender Trefferquote. Messergebnisse Tabelle A.11 im Anhang.

dass die WAH Kompression für Selektivitäten kleiner als 0,75% zum Einsatz kommen sollte. Dabei sollte im Bereich einer Selektivität größer als 0,1% auf die 32 Bit WAH Kompression gesetzt werden. Unter 0,1% empfiehlt sich sogar die Nutzung der 128 Bit WAH Kompression.

4.5.5 Umfassender Vergleich der Bitmap-basierten Schnittmengenberechnungsverfahren

In diesem Test soll herausgefunden werden, welcher Bitmap Algorithmus unter welcher Parameter-Kombination am effektivsten zu seien scheint. Dazu werden die Erkenntnisse aus Test 1 (↗ 4.5.1 und aus Test 2 (↗ 4.5.2) kombiniert. Es wird nun die Trefferquoten bei verschiedenen Selektivitäten, nicht nur bei einer Selektivität von 100%, variiert. Alle Messergebnisse finden sich dazu in Tabelle 4.3.
Für die Bitmap Intersection kann man aus Abbildung A.1 folgern, dass das Laufzeitverhalten bezüglich der Selektivität sich bei allen Trefferquoten gleich verhält. Dies war zu erwarten, da die Trefferquote nach den Ergebnisses aus der Messung zu Test 3 konstante Ausführungszeiten für die Schnittmengenberechnung nach dem nicht-komprimierten Bitmap Verfahren führte. Die von der Selektivität abhängigen Entwicklungen der Laufzeit gelten bei allen Trefferquoten. Abbildung A.2 zeigt noch einmal deutlicher das Laufzeitverhalten im niedrigen Selektionsbereich. Für Bitmap ohne Kompression gelten demnach die Erkenntnisse aus Test 2 und 3 auch in Kombination. Interessant ist das Laufzeitverhalten bei niedriger Selektivität bei der Verwendung der 128 Bit WAH Kompression. Abbildung 4.18 visualisiert dies. Die Messergebnisse bei der Selektivität von 0,5% und 1% zeigen, dass in diesem Bereich die Trefferquote einen starken Einfluss auf die Laufzeit hat. Eine niedrige Trefferquote verkürzt die Laufzeit deutlich. Bei 1% Selektivität übersteigt die Laufzeit bei 100% Trefferquote die Laufzeit bei 0% Trefferquote um den Faktor 15. Im Bereich von 0,5% um den Faktor 10,4 und bei 0,1% Sekektivität immerhin noch um den Faktor 3,03. Für WAH32 und Bitmap Intersection ohne Kompression ist dies nicht in dem Maße der Fall. So gelten für WAH32 nur die Faktoren 2,97, 1,94 und 1,33. Dies ist in Abbildung A.5 veranschaulicht. In Tabelle 4.3 sind die 3 Verfahren im Vergleich bei unterschiedlichen Selektivitäten und Trefferquoten abgetragen. Die jeweils kürzeste Laufzeit für die Kombination aus Trefferquote und Selektivität ist dabei grau hinterlegt. Anhand der Unregelmäßigkeit in der Hinterlegung der kürzesten Laufzeiten der Algorithmen erkennt man, dass es nicht einfach ist, das optimale Verfahren zu bekannter Trefferquote und bekannter Selektivität zu wählen. Die einfachste Aussage, die gemacht werden kann ist, dass bei einer Trefferquote von 0% das Schnittmengenberechnungsverfahren mit der 128 Bit Komprimierung zum Einsatz kommen sollte. Unabhängig von der Trefferquote sollte bei einer Selektivität von 0,1% und kleiner, ebenfalls auf WAH128 zurück gegriffen werden. Bei einer Selektivität von 0,5% sollte man auf WAH32 setzen, unabhängig von der Trefferquote.

Trefferquote	Verfahren	Selektivität 0,1%	0,5%	1%	10%	50%	100%
0%	Bitmap	3139	760	390	50	17	12
0%	WAH128	464	105	57	14	11	7
0%	WAH32	1148	286	145	22	12	7
25%	Bitmap	3145	715	370	50	18	11
25%	WAH128	629	351	254	37	17	9
25%	WAH32	1209	346	214	46	19	9
50%	Bitmap	3144	698	360	49	17	11
50%	WAH128	891	600	454	59	22	12
50%	WAH32	1315	415	287	68	24	13
75%	Bitmap	3142	677	350	51	15	12
75%	WAH128	1150	849	654	82	28	15
75%	WAH32	1419	487	364	90	30	15
100%	Bitmap	3152	655	341	50	15	14
100%	WAH128	1409	1096	854	106	34	7
100%	WAH32	1525	555	431	113	36	7

Tabelle 4.3: Medianwerte der Laufzeitmessungen zum 6. Bitmap Test. Die jeweils kürzeste Laufzeit bei vorliegender Trefferquote zu Selektivität ist grau hinterlegt.

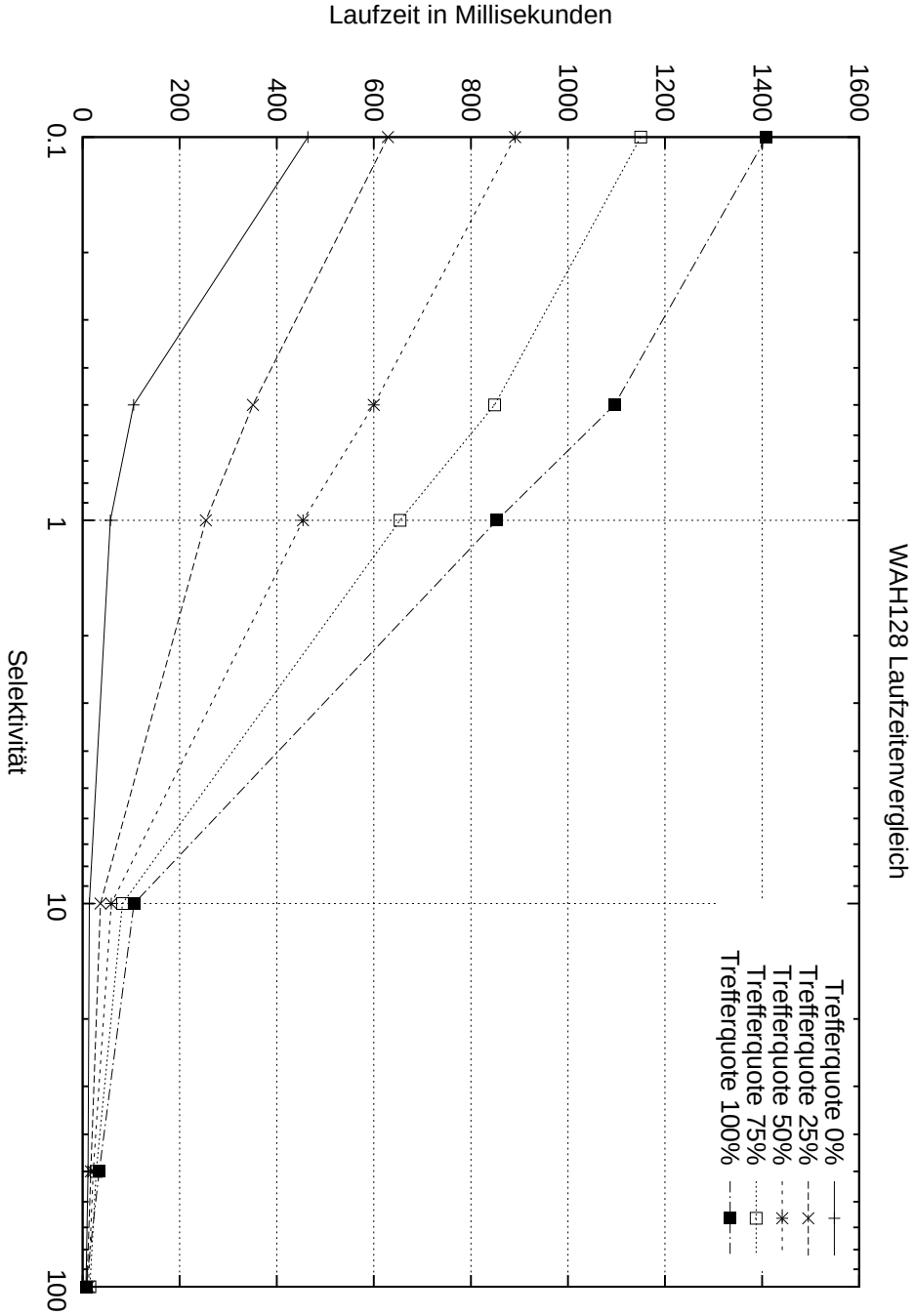

Abbildung 4.18: Vergleich der Laufzeiten von WAH128 bei variierender Selektivität und verschiedenen Trefferquoten. (Logarithmische x-Achse)

Auch bei einer Selektivität von 1% ist WAH32 dem Verfahren ohne Kompression vorzuziehen, solange die Trefferquote 75% nicht übersteigt. Zwischen 50% und 75% liegt der Punkt, an dem das Verfahren ohne Kompression eine kürzere Laufzeit aufweist als WAH32. Im Bereich mit einer Selektivität größer als 10% lieferte das kompressionslose Bitmap Intersection Verfahren die besten Laufzeiten. Nur bei sehr hohen oder niedrigen Trefferquoten und einer Selektivität von 100% empfahlen sich WAH128 und WAH32 gegenüber der normalen Bitmap Intersection.

4.5.6 Zusammenfassung

Eine allgemeingültige Empfehlung für ein Bitmap-basiertes Schnittmengenverfahren nach Trefferquote und Selektivität kann nicht einfach so getroffen werden. Tabelle 4.3 verdeutlicht dies gut. Trotzdem konnten einige Erkenntnisse über die Bitmap Algorithmen in diesem Abschnitt gewonnen werden.
Zuerst lässt sich feststellen, dass der Bitmap Algorithmus ohne WAH Kompression eine von der Trefferquote unabhängige Laufzeit bietet. In Fällen von kleinen Selektivitäten sollte komprimierungslose Bitmap Intersection allerdings nicht genutzt werden. Auffällige Performanz-Einbrüche waren die Folge. In Fällen, in dem die Trefferquote unbekannt ist, bietet sich daher das kompressionslose Bitmap Verfahren an. Interessant ist die Tabellenzelle für 50% Trefferquote und 1% Selektivität. Bei niedrigerer Selektivität empfiehlt sich der Einsatz von WAH Kompression, bei höherer Selektivität nur dann, wenn auch die Trefferquote abnimmt. Steigt die Trefferquote an und sinkt die Selektivität nicht ab, so ist der Einsatz von WAH Kompression ebenfalls nicht notwendig. Bei gleichbleibender Trefferquote und steigender Selektivität liegt der Einsatz des kompressionslosen Bitmap-Schnittmengenberechnungsverfahren ebenfalls nah.
Im Falle von einer 100%-igen Selektivität kann ruhig WAH Kompression zum Einsatz kommen, allerdings ist dies nicht unbedingt erforderlich, wenn man die reine Differenz der Laufzeiten betrachtet. Anders, wenn die Trefferquote 0% oder 100% beträgt. In den Fällen sollte WAH Kompression auf jeden Fall zum Einsatz kommen. Es zeigten sich Laufzeitverkürzungen um ca. den Faktor 2.
Bei niedriger Selektivität von 0,1% zeigte sich, dass auf jeden Fall WAH128 genutzt werden sollte. Hier wurde eine erhebliche Verkürzung der Laufzeit gegenüber Bitmap Intersection gemessen und eine nicht unerhebliche gegenüber WAH32. Nicht zum Einsatz kommen sollte die WAH128 Komprimierung dagegen, wenn die Selektivität im Bereich zwischen 0,1% und 1% liegt. In Abbildung 4.18 zeigt sich der drastische Anstieg der Laufzeit in diesem Selektivitätsbereich bei steigender Trefferquote für WAH128. Es ist daher auch schwer eine gewissen Wortlänge für die WAH Kompression zu empfehlen. Bei der Wahl der Länge der WAH-Wörter sollte bedacht werden, dass bei WAH128 dieser starke Anstieg der Laufzeiten zu beobachten war (Abbildung 4.18). Die WAH-Wortlänge sollte daher auch abhängig von

Trefferquote und Selektivität gewählt werden.
Insgesamt empfiehlt sich der Einsatz von WAH Kompression gerade bei niedriger Trefferquote und bei niedriger Selektivität. Auch für den Sonderfall von 100% Trefferquote und 100% Selektivität zeigte der Einsatz von WAH Kompression gute Laufzeitergebnisse.
Da nur die Schnittmenge für *unsigned int*-Werte berechnet wurde, war kein Einsatz von größeren Datenmengen möglich. Gerade dann, wenn die Größe der Daten zunimmt sollte die WAH-Kompression ihre Stärken ausspielen und auch erst voll entfalten können. Alle Daten für die Schnittmengenberechnung lagen im Arbeitsspeicher vor. Eine Übertragung von der Festplatte fand nicht statt. Bei dieser, von der Speicherbandbreite noch langsamer angeschlossenen Verbindung, wäre eine Versorgung des Prozessors mit möglichst vielen Daten in kurzer Zeit zu wünschen. Die WAH Kompression sollte die Bandbreite der Anbindung für mehr Daten pro Zeiteinheit nutzen können, da die Daten komprimiert auf der Festplatte vorliegen. Der Mehraufwand der Dekomprimierung würde vermutlich durch die Zeiteinsparung für die Datenübertragung ausgeglichen bzw. es wäre so, dass die Gesamtzeit der Schnittmengenberechnung mit dem Laden der Daten von der Festplatte kürzer mit Kompression wäre als ohne Kompression.

4.6 Vergleich aller Algorithmen

Im Folgenden werden alle Algorithmen, die zuvor einzeln betrachtet worden sind, untereinander verglichen.

4.6.1 Vergleich bei unsortierten Daten

Aus den vorangegangenen Tests in Abschnitt 4.3.1 und 4.5.1 zeigte sich, dass die Algorithmen mit unsortierten Input eine kürzere Laufzeit aufweisen gegenüber sortiertem Input addiert mit dem Zeitaufwand für das Sortieren der rid-Listen. Um zu erkennen, welches Vorgehen bei unsortiertem Input am besten gewählt wird, werden die Messergebnisse an dieser Stelle einmal zusammengeführt. Für die Arbeit auf sortierten rid-Listen wurde der Zeitaufwand für das Sortieren von zwei Schnittmengen mit dem Standard-C++-Sortierverfahren gemessen und für die Operation auf sortierten rid-Listen hinzu addiert. In Abbildung 4.19 ist zu erkennen, dass die Laufzeit des Radix Algorithmus die Laufzeit des Bitmap Schnittmengenverfahrens in allen Fällen übersteigt. Dabei ist mit einer Selektivität von 100% noch der beste Fall aus sich des Radix Algorithmus gewählt worden. Dieser zeigte bei hoher Selektivität die beste Performanz (Abschnitt 4.3.2). Die Trefferquote beträgt in diesem Vergleich ebenfalls 100%, doch diese beeinflusst die beiden Schnittmengenberechnungsverfahren nicht deutlich negativ (Abschnitt 4.3.3 und Abschnitt 4.5.3). Es ist zu erkennen, dass die Laufzeit für die Ausführung von std::sort() die Gesamtlaufzeiten der Algorithmen deutlich dominiert. Bei unsortierten Daten sollte aus dieser Sicht der Bitmap Algorithmus gewählt werden. Auf den Einsatz des Standard-C++-Sortierverfahrens sollte

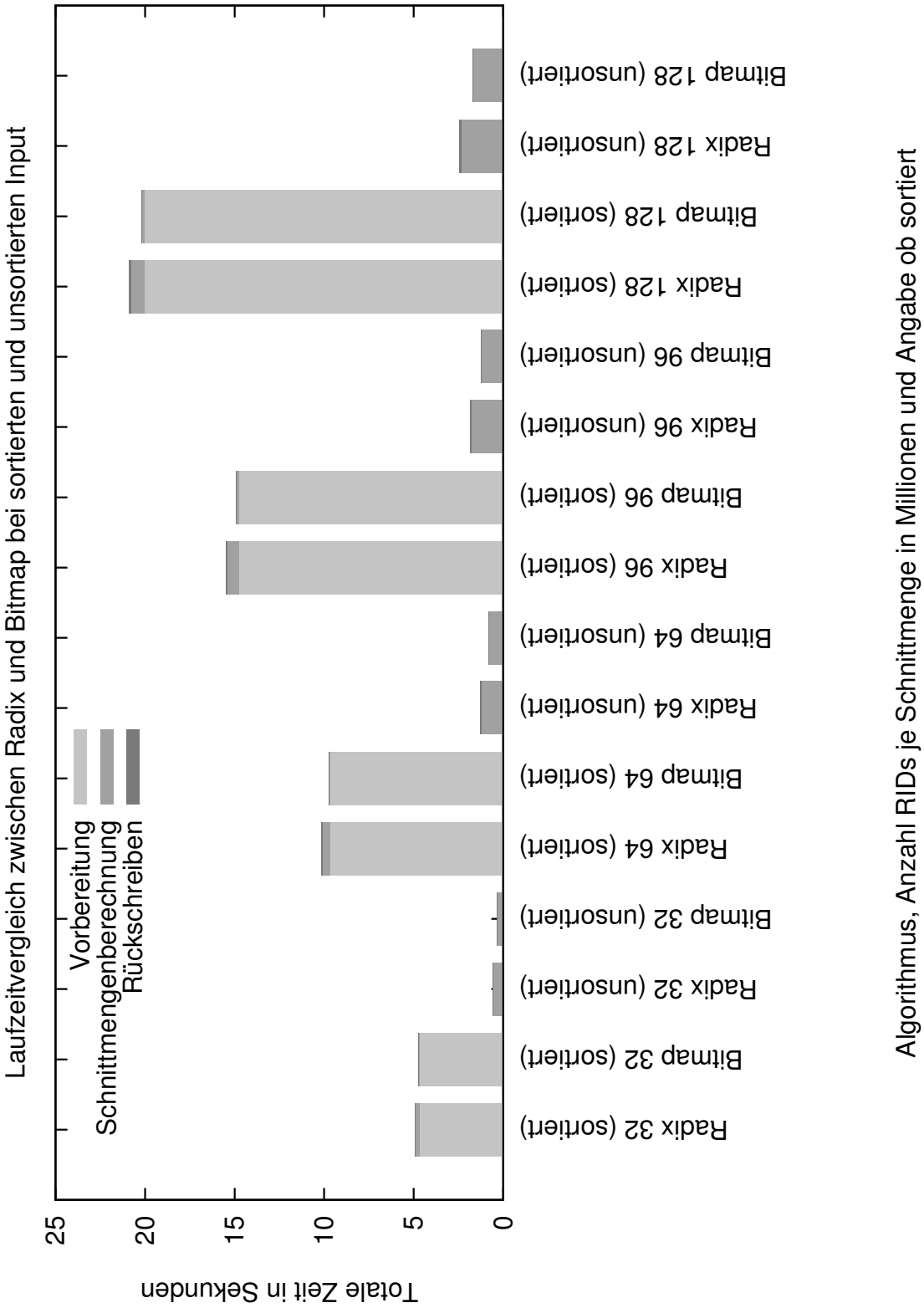

Abbildung 4.19: Vergleich der Laufzeiten von Radix und Bitmap Intersection bei sortierten und unsortierten Input. Messdaten in Tabelle A.12 im Anhang.

dabei verzichtet werden. Dies zeigt Abbildung 4.20 auch noch einmal deutlich. Auffällig

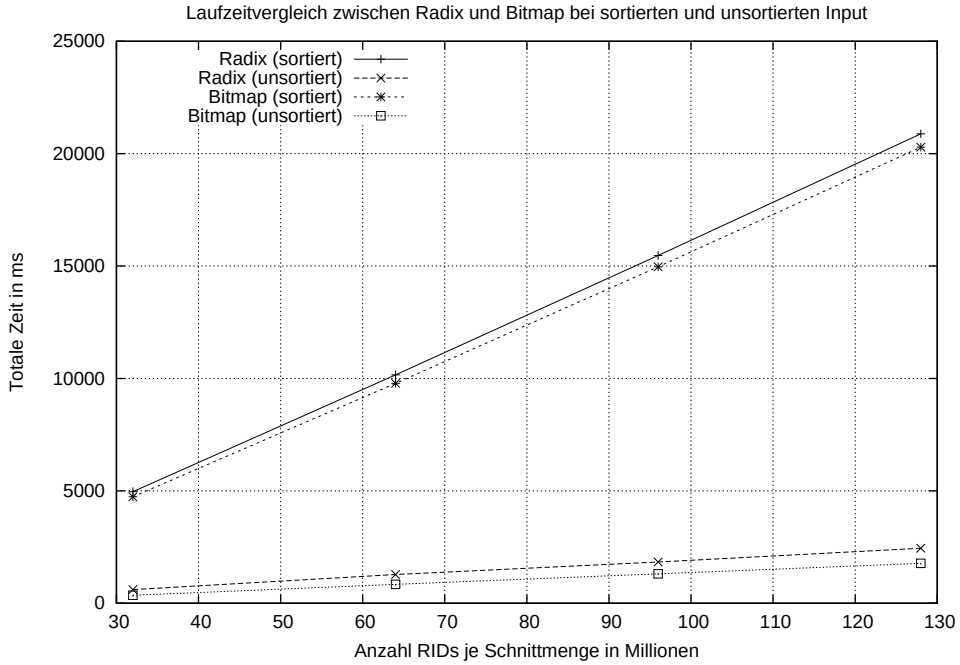

Abbildung 4.20: Vergleich der Laufzeiten von Radix und Bitmap Intersection bei sortierten und unsortierten Input. Messdaten in Tabelle A.12 im Anhang.

ist, dass die Laufzeit mit steigender Anzahl an Elementen bei rid-Listen als Input, die per std:.sort() sortiert wurden, deutlich stärken ansteigt als bei unsortierten rid-Listen als Input. Bei großen Schnittmengen sollte daher erst recht auf den Einsatz von std::sort() verzichtet werden.

4.6.2 Vergleich bei sortierten Daten

Im vorherigen Abschnitt konnte gezeigt werden, dass die Algorithmen aus unsortierten rid-Listen schneller die Schnittmenge berechnen, als aus sortierten rid-Listen wenn sich dafür noch der Sortieraufwand des C++-Standardsortierverfahrens zur Gesamtausführungszeit hinzu addiert. In diesem Abschnitt sollen jetzt die Algorithmen bei Eingabe von sortierten rid-Listen miteinander verglichen werden. Dazu wird auf die Testergebnisse aus den vorherigen Tests zurückgegriffen und diese, Falls nötig, ergänzt.
Aus Abschnitt 4.3, 4.4 und 4.5 sind die unterschiedlichen Stärken und Schwächen der Algorithmen bereits herausgearbeitet worden. Der Radix-Cluster Algorithmus zeigte für hohe Selektivitäten bessere Laufzeiten. Die Trefferquote hatte für 50% den besten Einfluss auf die Gesamtlaufzeit. Allerdings fielen die Laufzeitunterschiede eher gering aus. Für den

SIMD-Algorithmus ergab sich als beste Voraussetzung eine niedrige Trefferquote . Bei den Bitmap-basierten Verfahren zeigte sich, dass der jeweils von der Performanz beste Algorithmus nicht allgemein gewählt werden kann. Bei unterschiedlichen Kombinationen aus Selektivität und Trefferquote zeigten sich die 3 Schnittmengenberechnungsverfahren unterschiedlich performant. Bei niedriger Selektivität von 0,1% nahm die Laufzeit des Bitmap Algorithmus ohne WAH Kompression z. B. deutlich zu.
In diesem Abschnitt wird davon ausgegangen, dass die rid-Listen sortiert vorliegen. Aufwand für das Sortieren des Inputs fällt demnach nicht an. Es wird die Trefferquote zwischen 0% und 100% variiert, bei Selektivitätswerten für die rid-Listen von 0,1%-100%. Es wird immer der Radix-Cluster Algorithmus mit dem SIMD verglichen. Dazu kommt das Bimap-basierte Verfahren, welches sich nach dem Messwerten aus Abschnitt 4.5.5 als bestes für diese Kombination aus Trefferquote und Selektivität angeboten hat.
Es ist anzumerken, dass die hier verwendeten Laufzeiten für die Bitmap-basierten Verfahren die Erstellung der Bitmap und die nach der Bestimmung der Schnittmenge anstehende Rückrechnung des Ergebnis-Bitvektors zu Integer-Werten enthält. Die Bitmap-basierten Schnittmengenberechnungsverfahren operieren also nicht bei optimalen Input.
Alle in Tabelle A.12 und 4.5 gemessenen Werte sind die Medianwerte der Zeitmessung des Benchmarks bei sortierten Input. Die Schnittmengen enthielten dabei je Schnittmenge 4.294.967 gleichverteilte RIDs. Aus Tabelle A.12 folgt, dass im Falle der Selektivität

		Selektivität					
Trefferquote	Verfahren	0,1%	0,5%	1%	10%	50%	100%
0%	Radix	233	218	217	73	255	272
0%	Bitmap	464 [2]	105 [2]	57 [2]	14 [2]	11 [2]	7 [2]
25%	Radix	229	211	208	94	79	89
25%	Bitmap	629 [2]	346 [3]	214 [3]	37 [2]	17 [2]	9 [2,3]
50%	Radix	229	210	210	160	81	64
50%	Bitmap	891 [2]	415 [3]	287 [3]	49 [1]	17 [1]	11 [1]
75%	Radix	230	212	211	156	78	60
75%	Bitmap	1150 [2]	487 [3]	350 [1]	51 [1]	15 [1]	12 [1]
100%	Radix	231	212	209	150	73	63
100%	Bitmap	1409 [2]	555 [3]	341 [1]	50 [1]	15 [1]	7 [2,3]

Tabelle 4.4: Medianwerte der Laufzeitmessungen, Vergleich des Radix Algorithmus mit den Bitmap-basierten Verfahren. Die jeweils kürzeste Laufzeit bei vorliegender Trefferquote zu Selektivität ist grau hinterlegt. [1]=Bitmap Algorithmus, [2]=WAH128 und [3]=WAH32

von 0,1% der Radix-Cluster Algorithmus eine kürzere Laufzeit aufweist als das Bitmap-basierte Verfahren. Dabei zeigte das Verfahren mit Nutzung der 128 Bit WAH Kompression die besten Laufzeiten im Vergleich zu den anderen Bitmap-basierten Verfahren. Liegt die

Selektivität über 0,5% und die Trefferquote bei 0%, so zeigten sich die Bitmap-basierten Algorithmen performanter als der Radix-Cluster Algorithmus. Bei allen anderen Trefferquoten wurden für den Radix-Cluster Algorithmus bei Selektivitäten kleiner als 10% kürzere Laufzeiten gemessen, gegenüber den Bitmap-Algorithmen. Bei Selektivitäten größer gleich 10% zeigten sich die Bitmap-basierten Verfahren performanter im Vergleich zum Radix Verfahren. Tabelle 4.5 enthält die Messwerte zum SIMD-Algorithmus. Die Laufzeiten

	Selektivität					
Trefferquote	0,1%	0,5%	1%	10%	50%	100%
25%	8	3	4	3	4	3
50%	10	5	6	6	5	6
75%	11	9	9	8	9	8
100%	15	11	12	13	12	14

Tabelle 4.5: Medianwerte der Laufzeitmessungen des SIMD-Algorithmus. Mit „-“ markierte Werte konnten auf Grund eines Speicherzugriffsfehlers nicht gemessen werden.

des SIMD-Algorithmus fallen in diesen Messungen kürzer aus als die des Radix-Cluster Algorithmus oder der Laufzeiten der Bitmap-basierten Verfahren. Nur im Falle von 100% Trefferquote und 100% Selektivität zeigte sich der Bitmap Algorithmus mit Kompression schneller als der SIMD Algorithmus.

4.7 Bewertung der Algorithmen

In diesem Abschnitt sollen die Algorithmen bezüglich ihrer Performanz in verschiedenen Szenarien bewertet werden. Dabei soll der Einsatzbereich für die jeweiligen Algorithmen abgesteckt werden.
Zuerst kann man aus dieser Evaluation die Erkenntnis ziehen, dass der Einsatz von Schnittmengenberechnungsverfahren, die mit unsortierten rid-Listen als Input arbeiten können, dem Einsatz eines externen Sortierverfahrens mit anschließender Schnittmengenberechnung mit dem Algorithmus vorzuziehen sind. Der Aufwand für das Sortieren der rid-Listen übersteigt den Zuwachs an Laufzeit zwischen sortierten und unsortierten Input bei weitem. Der Einsatz eines viel effizienteren Sortierverfahrens könnte diese Erkenntnis kippen, doch müsste das angewendete Sortierverfahren schon sehr effizient sein.
Des weiteren zeigt sich, dass sich die Bitmap-basierten Algorithmen, trotz der vorherigen Erstellung der Bitmaps und der, nach der Schnittmengenberechnung erfolgten, Rückumwandlung der Ergebnisbitmap in Integer-Werten, gerade gegenüber dem Radix-Cluster Algorithmus nicht verstecken müssen.
Tabelle 4.5 zeigte die kurzen Laufzeiten des SIMD-Algorithmus. Die Performanz des SIMD-Algorithmus hat in meinen Messungen alle anderen Algorithmen unterboten.

Die Bitmap-basierten Schnittmengenberechnungsverfahren haben dann einen Nachteil, wenn die Werte der RIDs sich stark Unterscheiden. Eine große Domäne führt zu größeren Bitmaps, auch wenn kaum Werte zwischen dem ersten Bit der Bitmap und dem letzten Bit der Bitmap gesetzt sein sollte. Als Beispiel sei eine rid-Liste mit den Werten 1 und 4 Milliarden gefüllt, die andere rid-Liste ebenfalls. Der SIMD Algorithmus muss in diesem Fall nur zwei RIDs je Schnittmenge miteinander vergleichen. Die Bitmap-basierten Algorithmen erhalten aber einen Vektor mit 4 Milliarden Bits. Dies ergibt umgerechnet ca. 477 Megabyte an Daten je Schnittmenge. Beachtet man, dass die Speicherbandbreite als limitierender Faktor angesehen wird (vgl. [MBK02, BBHS14]), so muss gerade in Fällen von wenigen Elementen in den Schnittmengen bei sehr großen Domänen auf den Einsatz von Bitmap-basierten Schnittmengenberechnungsverfahren verzichtet werden. Liegt eine hingegen eine hohe Selektivität vor und eine eher kleine Dömäne, so zeichnen sich die Bitmap-basierten Verfahren gegenüber den anderen betrachteten Schnittmengenberechnungsverfahren aus, denn in den Fällen müssen auch dementsprechend viele RIDs für die Schnittmengenberechnung übertragen werden. Liegen zudem, in Fällen von hoher Selektivität bei vergleichsweise kleiner Domäne, die Daten bereits als Bitvektoren innerhalb des Datenbanksystems vor, so wird der Einsatz von nicht Bitmap-basierten Verfahren nur zur Verschlechterung der Performanz der Schnittmengenberechnung führen.
Die Wahl des „richtigen“ Schnittmengenberechnungsverfahrens kann nicht pauschal mit der Empfehlung eines Schnittmengenberechnungsverfahrens beantwortet werden. Einfluss auf die Wahl des Schnittmengenberechnungsverfahrens haben die Selektivität der Schnittmengen, die Trefferquote, die Domäne der RIDs sowie die Speicherstruktur der Daten innerhalb des Datenbanksystems (Speicherstruktur meint, ob die Daten als Integer-Werte aus der Datenbank gewonnen werden oder ob die Daten als Bitvektoren gespeichert sind). Außerdem beeinflusst die Annahme darüber, ob die rid-Listen sortiert oder unsortiert sind die Wahl des Schnittmengenberechnungsverfahrens. In der Regel sind rid-Listen aber bereits sortiert bzw., lassen sich sortiert aus der Datenbank, im Zusammenhang mit der Ausführung einer Schnittmengenberechnungsanfrage, heraus selektieren.

4.8 Konsequenzen für zukünftige Hardware

In der Evaluation der Schnittmengenberechnungsverfahren zeigt sich, dass der Einsatz von SIMD und AVX Befehlssätzen zu guten Leistungen der Algorithmen führt. In unserem Fall setzt der SIMD Algorithmus einen Befehl zu einem Voll-Vergleich von jeweils 8 16-Bit großen Werten ein. Der Bitmap Algorithmus setzt einen AVX Befehl ein, der die Verundung von je 256 Bits ermöglicht. Gerade für den SIMD Algorithmus wäre es förderlich, wenn ein Voll-Vergleich von 32-Bit großen Werten möglich wäre. Dadurch würde die im SIMD Algorithmus in der Vorbereitungsphase stattfindende Unterteilung der Eingabewerte nach den höherwertigen Bits hinfällig. Vorteilhaft wäre natürlich auch, wenn mehr als jeweils

8 Werte auf einmal verglichen werden könnten. Eine andere wünschenswerte Entwicklung bezüglich zukünftiger Hardware wäre die Erhöhung der Speicherbandbreite und damit das Durchbrechen der „Speicher Wand“ vgl. [BBHS14]. Die Erhöhung der Speicherbandbreite würde es ermöglichen, den Prozessor schneller mit Daten zu Versorgen und damit einfacher eine volle Auslastung zu erreichen.
Der Einsatz von GPUs (*g*raphics *p*rocessing *u*nits), APUs (*a*ccelerated *p*rocessing *u*nits oder auch coupled CPU-GPU), FPGAs (*f*ield-*p*rogrammable *g*ate *a*rrays) oder MICs (*m*any integrated core-Architektur) sollte ebenfalls weiterhin näher in Betracht gezogen werden. In Hinblick dessen, sei auf [BBS14] verwiesen.
Gerade die unterschiedlichen Fähigkeiten von CPU, GPU, FPGA, MIC und APU ermöglichen die Schnittmengenberechnung unter verschiedensten Hardware-Voraussetzungen.

4.9 Zusammenfassung

In diesem Kapitel wurden die verschiedenen Schnittmengenverfahren zuerst einzeln evaluiert. Dazu wurde zwischen dem Radix, dem SIMD und dem C++-Standardschnittmengenberechnungsverfahren sowie den Bitmap-basierten Verfahren unterschieden. Die Algorithmen wurden in dieser Zusammenstellung zuerst untereinander bewertet. Dabei wurde der Einfluss der Trefferquote und der Selektivität auf die Laufzeit der Algorithmen betrachtet. Anschließend folgte die Bewertung der Schnittmengenberechnungsverfahren untereinander, wobei auf die Messergebnisse aus den vorherigen Abschnitten dieses Kapitels zurückgegriffen wurde. Abschließend kann für kein Schnittmengenberechnungsverfahren eine generelle Empfehlung ausgesprochen werden. Zu verschieden ist die Performanz der Algorithmen unter den verschiedensten Bedingungen.

Kapitel 5

Zusammenfassung

Im Verlauf dieser Arbeit wurden die Grundlagen für die Evaluation der Schnittmengenberechnung auf moderner Hardware gelegt. Dies ermöglichte die Einführung in den in dieser Bachelorarbeit entwickelten Benchmark. Durch Nutzung des Benchmarks konnten die aus der parallel laufenden Bachelorarbeit implementierten Schnittmengenberechnungsverfahren evaluiert werden.
In der Evaluierung zeigte sich schnell, dass es kein triviales Unterfangen ist, den richtigen Algorithmus zur Schnittmengenberechnung zu wählen. Bei der Wahl des Algorithmus ist neben der Datenstruktur des Datenbanksystems (Bitmaps oder rid-Listen) auch die Verteilung der Daten zu beachten. Auf die Laufzeit der Schnittmengenberechnung haben anschließend sowohl die Selektivität der Schnittmengen, wie auch die Anzahl an Elementen in der Schnittmenge, einen stellenweise erheblichen Einfluss. Gerade die Wahl des richtigen Bitmap-basierten Schnittmengenberechnungsverfahren gestaltete sich schwierig. Mal empfiehlt sich der Einsatz von Kompression (gerade bei niedriger Selektivität), in anderen Fällen war der Einsatz der Kompression für eine deutlich längere Laufzeit verantwortlich.
Erfreulich war die Performanz des skalaren SIMD-Algorithmus. Die Nutzung der SIMD-Befehlssätze schaffte eine effiziente Implementierung. Das SIMD Verfahren zeigt trotz ihres eigentlich eher einfachen Grundgedankens gute Laufzeiten im Vergleich zu den komplexeren Algorithmen (Bitmap und Radix-Cluster Algorithmus).
Auffällig gut war die Gesamtlaufzeit bei unsortierten Input unter Verwendung des Radix-Cluster Algorithmus oder des Bitmap-Verfahrens ohne WAH Kompression.
In dieser Arbeit wurde nur die Implementierung von einigen Algorithmen auf CPUs untersucht. [BBHS14, HLH13, MTA09] zeigen, dass es noch jede Menge andere Architekturen gibt und in dem Bereich der effizienten Implementierung von Datenbankoperationen noch ein großes Forschungsfeld offen steht. In dieser Hinsicht freue ich mich auch, bald das Paper [BBS14] lesen zu können.
Sowohl die Hardware-nahe Implementierung, als auch die Entwicklung zukünftiger Hardware, bietet Potenziale für effizientere und effektivere Systeme für Datenbankoperationen.

In der Nutzung von SIMD und AVX Befehlssätzen steckt eine Menge Potenzial hinsichtlich einer effizienten Ausnutzung von CPU Ressourcen.
Während dieser Arbeit habe ich viel über das Potenzial moderner CPU-Architekturen erfahren und auch die Tücken der Evaluation kennen gelernt.

An dieser Stelle möchte im mich recht herzlich für die umfangreiche Betreuung, Anleitung und konstruktive Kritik durch Herrn Prof. Dr. Jens Teubner und Herrn M. Sc. Sebastian Breß bedanken.

Anhang A

Weitere Informationen

A.1 Kapitel 4

Elemente in Millionen	Gesamt	Vorbereitung	Schnittmengenberechnung
32 (sortiert)	269	243	26
32 (unsortiert)	608	582	26
Differenz	339	339	0
64 (sortiert)	505	454	51
64 (unsortiert)	1277	1226	51
Differenz	772	772	0
96 (sortiert)	682	605	76
96 (unsortiert)	1833	1757	76
Differenz	1151	1151	0
128 (sortiert)	847	745	101
128 (unsortiert)	2446	2344	101
Differenz	1599	1599	0

Tabelle A.1: Medianwerte der Laufzeitmessungen zum 1. Radix Test

Selektivität	Gesamt	Vorbereitung	Schnittmengenberechnung
0,1%	229	225	4
1%	207	203	4
10%	151	147	4
100%	63	59	4

Tabelle A.2: Medianwerte der Laufzeitmessungen zum 2. Radix Test

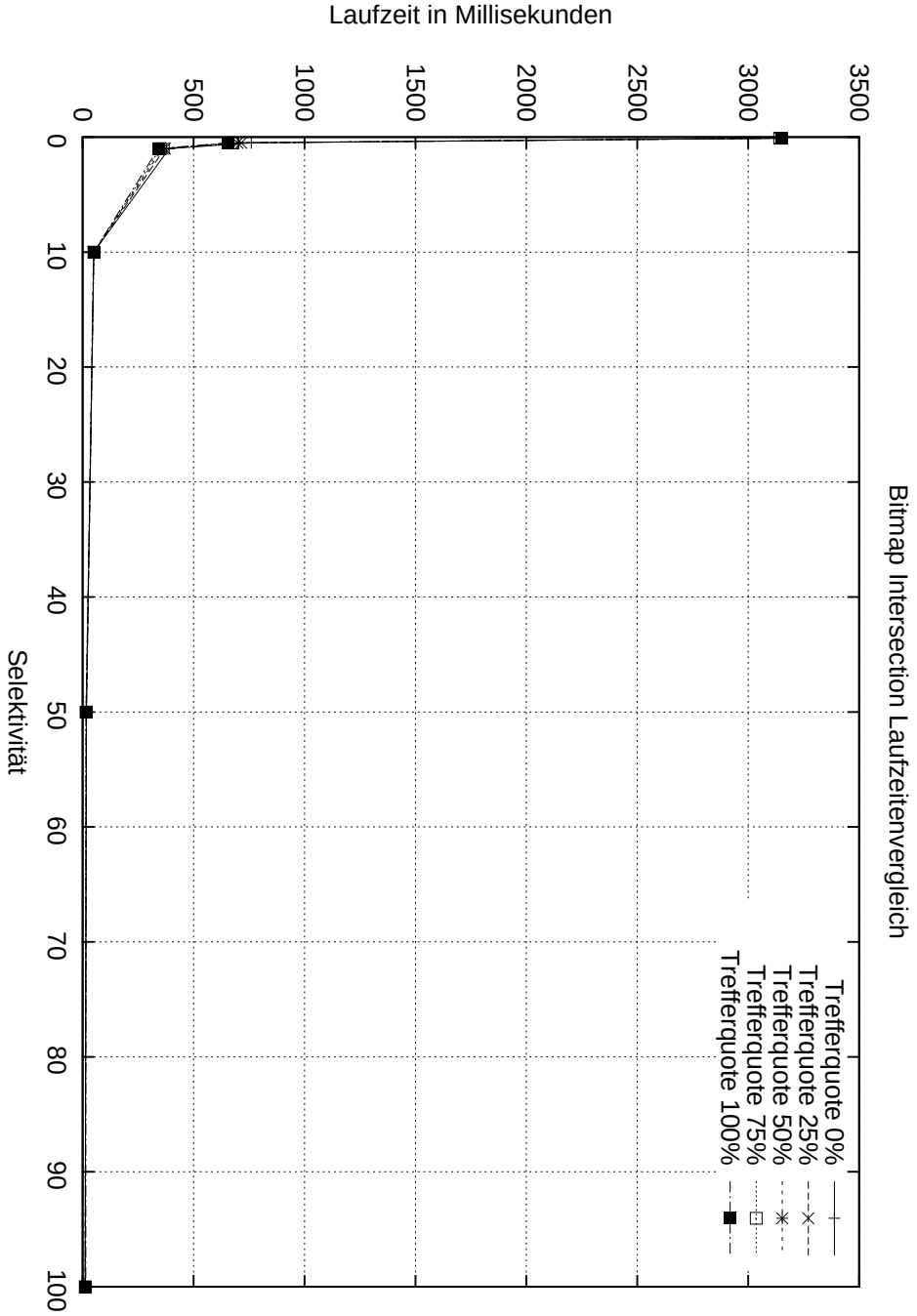

Abbildung A.1: Vergleich der Laufzeiten der Bitmap Intersection bei variierender Selektivität und verschiedenen Trefferquoten.

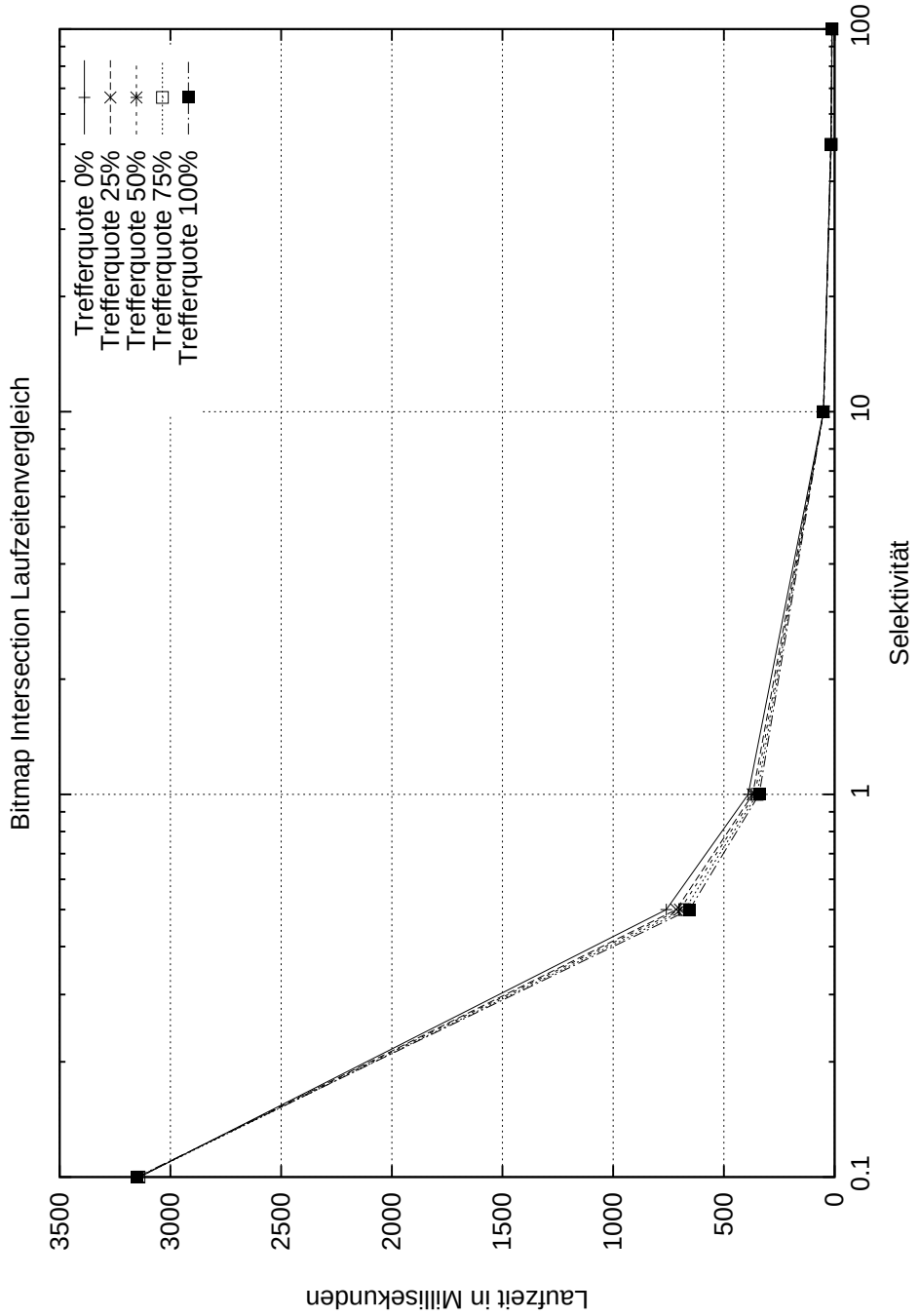

Abbildung A.2: Vergleich der Laufzeiten der Bitmap Intersection bei variierender Selektivität und verschiedenen Trefferquoten. (Logarithmische x-Achse)

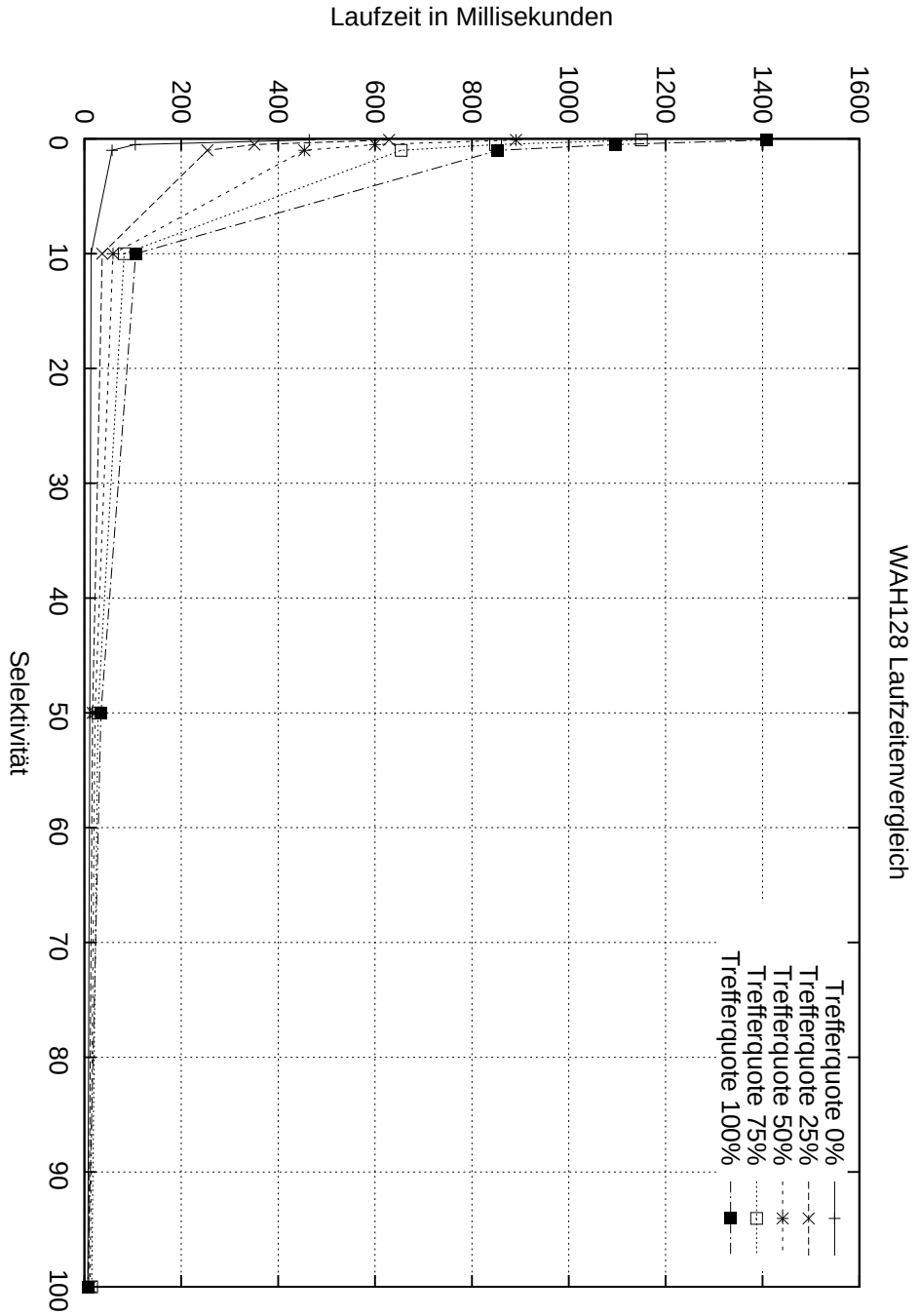

Abbildung A.3: Vergleich der Laufzeiten von WAH128 bei variierender Selektivität und verschiedenen Trefferquoten.

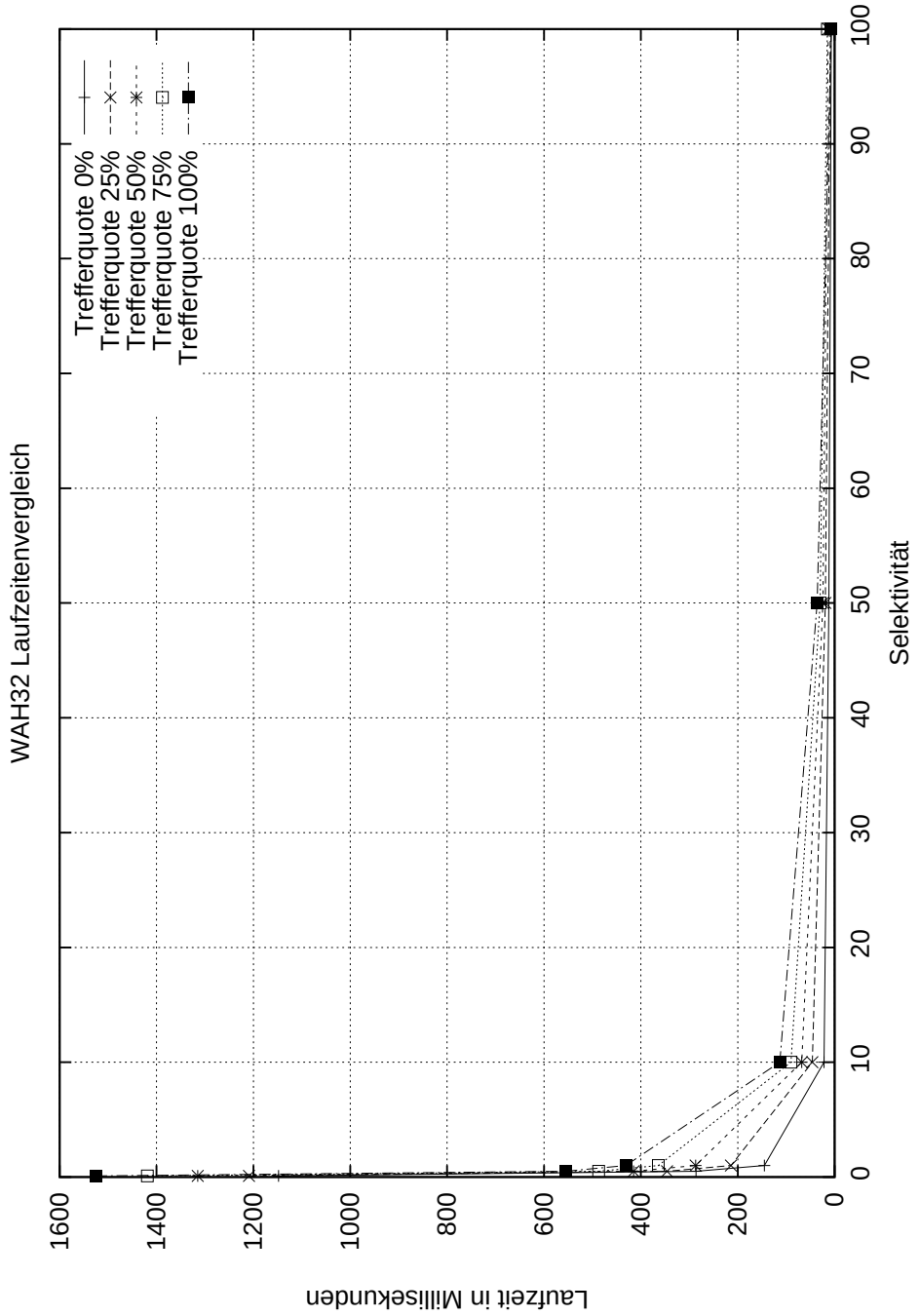

Abbildung A.4: Vergleich der Laufzeiten von WAH32 bei variierender Selektivität und verschiedenen Trefferquoten.

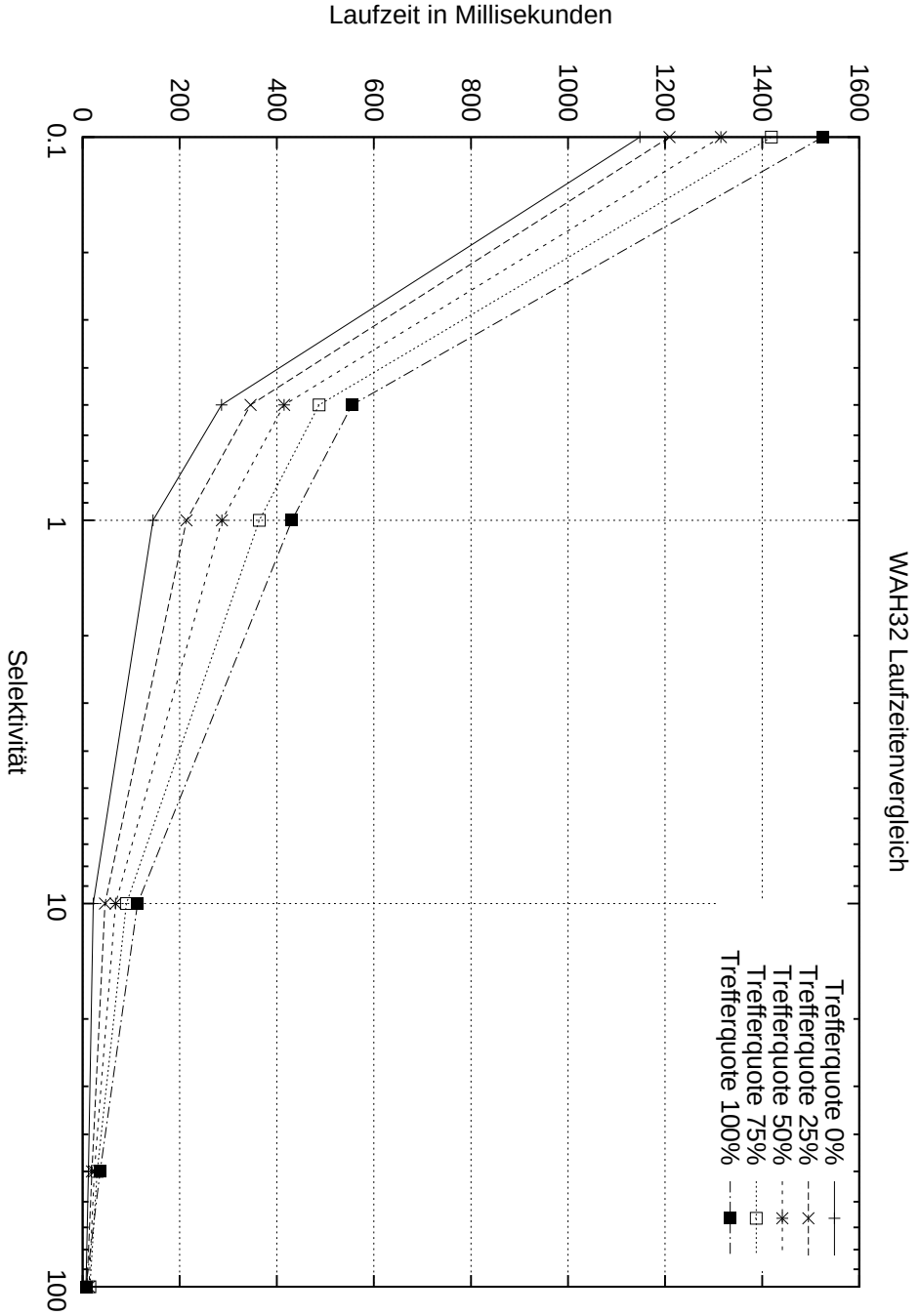

Abbildung A.5: Vergleich der Laufzeiten von WAH32 bei variierender Selektivität und verschiedenen Trefferquoten. (Logarithmische x-Achse)

Trefferquote	Gesamt	Vorbereitung	Schnittmengenberechnung
25%	816	795	21
50%	762	720	42
75%	820	738	81
100%	839	737	101

Tabelle A.3: Medianwerte der Laufzeitmessungen zum 3. Radix Test

Selektivität	Gesamt	Vorbereitung	Berechnung	Rückschreiben
SIMD 0,1%	11	5	2	4
Standard 0,1%	4	0	4	0
SIMD 1%	11	5	2	4
Standard 1%	4	0	4	0
SIMD 10%	11	5	2	4
Standard 10%	4	0	4	0
SIMD 100%	11	5	2	4
Standard 100%	4	0	4	0

Tabelle A.4: Medianwerte der Laufzeitmessungen zum 1. SIMD Test

Trefferquote	Gesamt	Vorbereitung	Berechnung	Rückschreiben
SIMD 10% Trefferquote	35	12	8	15
Standard 10% Trefferquote	61	0	61	0
SIMD 20% Trefferquote	67	24	15	28
Standard 20% Trefferquote	68	0	68	0
SIMD 30% Trefferquote	96	34	22	40
Standard 30% Trefferquote	75	0	75	0
SIMD 40% Trefferquote	126	43	29	54
Standard 40% Trefferquote	81	0	81	0
SIMD 50% Trefferquote	153	52	35	66
Standard 50% Trefferquote	88	0	88	0
SIMD 60% Trefferquote	177	62	33	82
Standard 60% Trefferquote	95	0	95	0
SIMD 70% Trefferquote	207	73	39	95
Standard 70% Trefferquote	101	0	101	0
SIMD 80% Trefferquote	233	81	45	107
Standard 80% Trefferquote	109	0	109	0
SIMD 90% Trefferquote	261	91	50	120
Standard 90% Trefferquote	115	0	115	0
SIMD 100% Trefferquote	291	102	56	133
Standard 100% Trefferquote	122	0	122	0

Tabelle A.5: Medianwerte der Laufzeitmessungen zum 2. SIMD Test

Algorithmus und Selektion	Gesamt	Vorbereitung	Berechnung	Rückschreiben
SIMD <2	199	110	44	45
Standard <2	1058	0	1058	0
SIMD <3	211	108	44	59
Standard <3	1017	0	1017	0
SIMD <4	220	107	46	67
Standard <4	1017	0	1017	0
SIMD <5	227	108	47	72
Standard <5	933	0	933	0
SIMD <6	231	109	46	76
Standard <6	897	0	897	0
SIMD >2	233	107	49	77
Standard >2	884	0	884	0
SIMD >3	221	107	45	69
Standard >3	957	0	957	0
SIMD >4	219	108	47	64
Standard >4	991	0	991	0
SIMD >5	212	107	45	60
Standard >5	1011	0	1011	0
SIMD >6	212	109	45	58
Standard >6	1025	0	1025	0

Tabelle A.6: Medianwerte der Laufzeitmessungen zum 3. SIMD Test

Elemente in Millionen	Gesamt	Vorbereitung	Berechnung	Rückschreiben
32 (sortiert)	68	39	0	29
32 (unsortiert)	355	326	0	29
64 (sortiert)	126	71	1	54
64 (unsortiert)	837	782	1	54
96 (sortiert)	189	106	2	81
96 (unsortiert)	1307	1226	2	79
128 (sortiert)	253	142	3	108
128 (unsortiert)	1775	1667	3	105

Tabelle A.7: Medianwerte der Laufzeitmessungen zum 1. Bitmap Test

Algorithmus und Selektivität	Gesamt	Vorbereitung	Berechnung	Rückschreiben
Bitmap 0,1%	3147	860	113	2174
WAH 128 0,1%	1408	191	680	537
WAH 32 0,1%	1527	534	602	391
Bitmap 1%	343	98	11	234
WAH 128 1%	853	26	455	372
WAH 32 1%	437	55	226	156
Bitmap 10%	49	17	1	31
WAH 128 10%	105	12	54	39
WAH 32 10%	113	15	59	39
Bitmap 40%	19	9	0	10
WAH 128 40%	39	11	17	11
WAH 32 40%	39	11	17	11
Bitmap 60%	15	8	0	7
WAH 128 60%	31	11	12	8
WAH 32 60%	32	11	13	8
Bitmap 80%	13	7	0	6
WAH 128 80%	26	10	10	6
WAH 32 80%	29	11	11	7
Bitmap 100%	13	8	0	5
WAH 128 100%	7	4	0	3
WAH 32 100%	7	4	0	3

Tabelle A.8: Medianwerte der Laufzeitmessungen zum 2. Bitmap Test

Algorithmus und Trefferquote	Gesamt	Vorbereitung	Berechnung	Rückschreiben
Bitmap 10%	229	150	3	76
WAH128 10%	256	218	21	17
WAH32 10%	270	220	29	21
Bitmap 20%	229	149	3	77
WAH128 20%	287	218	37	32
WAH32 20%	302	219	45	38
Bitmap 30%	235	150	3	82
WAH128 30%	320	218	54	48
WAH32 30%	334	219	61	54
Bitmap 40%	235	148	3	84
WAH128 40%	352	218	71	63
WAH32 40%	366	218	77	71
Bitmap 50%	229	146	3	80
WAH128 50%	384	217	88	79
WAH32 50%	397	217	93	87
Bitmap 60%	232	139	3	90
WAH128 60%	417	217	105	95
WAH32 60%	428	216	109	103
Bitmap 70%	232	136	3	93
WAH128 70%	449	217	121	111
WAH32 70%	459	215	124	120
Bitmap 80%	238	140	3	95
WAH128 80%	479	216	137	126
WAH32 80%	490	214	140	136
Bitmap 90%	240	139	3	98
WAH128 90%	512	216	154	142
WAH32 90%	523	214	156	153
Bitmap 100%	250	141	3	106
WAH128 100%	231	129	5	97
WAH32 100%	245	129	16	100

Tabelle A.9: Medianwerte der Laufzeitmessungen zum 3. Bitmap Test

Algorithmus und Trefferquote	Gesamt	Vorbereitung	Berechnung	Rückschreiben
Bitmap 90%	240	139	3	98
WAH128 90%	512	216	154	142
WAH32 90%	523	214	156	153
Bitmap 92%	247	143	3	101
WAH128 92%	519	216	158	145
WAH32 92%	528	213	159	156
Bitmap 94%	246	137	3	106
WAH128 94%	526	216	162	148
WAH32 94%	534	213	162	159
Bitmap 96%	248	140	3	105
WAH128 96%	532	216	165	151
WAH32 96%	540	213	165	162
Bitmap 98%	252	143	3	106
WAH128 98%	538	216	168	154
WAH32 98%	548	213	169	166
Bitmap 100%	250	141	3	106
WAH128 100%	231	129	5	97
WAH32 100%	245	129	16	100

Tabelle A.10: Medianwerte der Laufzeitmessungen zum 4. Bitmap Test

Algorithmus und Selektivität	Gesamt	Vorbereitung	Berechnung	Rückschreiben
Bitmap 0,1%	3147	860	113	2174
WAH128 0,1%	1408	191	680	537
WAH32 0,1%	1527	534	602	391
Bitmap 0,15%	2120	595	71	1454
WAH128 0,15%	1296	130	652	514
WAH32 0,15%	1146	352	479	315
Bitmap 0,2%	1604	457	53	1094
WAH128 0,2%	1205	99	619	487
WAH32 0,2%	910	262	384	264
Bitmap 0,25%	1286	367	43	876
WAH128 0,25%	1159	81	600	478
WAH32 0,25%	830	207	381	242
Bitmap 0,5%	655	189	22	444
WAH128 0,5%	1097	46	579	472
WAH32 0,5%	557	102	273	182
Bitmap 0,75%	447	128	14	305
WAH128 0,75%	1065	33	568	464
WAH32 0,75%	494	73	241	180
Bitmap 1%	343	98	11	234
WAH128 1%	853	26	455	372
WAH32 1%	437	55	226	156

Tabelle A.11: Medianwerte der Laufzeitmessungen zum 5. Bitmap Test

Fall und Elemente je Schnittmenge in Millionen	Gesamt	Sortieraufwand	Vorbereitung	Berechnung	Rückschreiben
Radix 32 (sortiert)	4936	4667	243	26	0
Bitmap 32 (sortiert)	4735	4667	39	0	29
Radix 32 (unsortiert)	608	0	582	26	0
Bitmap 32 (unsortiert)	355	0	326	0	29
Radix 64 (sortiert)	10161	9656	454	51	0
Bitmap 64 (sortiert)	9782	9656	71	1	54
Radix 64 (unsortiert)	1277	0	1226	51	0
Bitmap 64 (unsortiert)	837	0	782	1	54
Radix 96 (sortiert)	15469	14778	605	76	0
Bitmap 96 (sortiert)	14967	14778	106	2	81
Radix 96 (unsortiert)	1833	0	1757	76	0
Bitmap 96 (unsortiert)	1307	0	1226	2	79
Radix 128 (sortiert)	20883	20037	745	101	0
Bitmap 128 (sortiert)	20290	20037	142	3	108
Radix 128 (unsortiert)	2446	0	2344	101	0
Bitmap 128 (unsortiert)	1775	0	1667	3	105

Tabelle A.12: Medianwerte der Laufzeitmessungen zum Vergleich zwischen Radix und Bitmap bei sortierten und unsortierten Input.

Abbildungsverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[AS13] Auer, B. und F. Seitz: *Grundkurs Wirtschaftsmathematik: Prüfungsrelevantes Wissen - Praxisnahe Aufgaben - Komplette Lösungswege*. Springer Fachmedien Wiesbaden, 2013.

[BBHS14] Broneske, David, Sebastian Bress, Max Heimel und Gunter Saake: *Toward Hardware-Sensitive Database Operations*. In: *Proceedings of the 17th International Conference on Extending Database Technology (EDBT)*, Seiten 229–234. OpenProceedings.org, 2014.

[BBS14] Broneske, David, Sebastian Bress und Gunter Saake: *Database Scan Variants on Modern CPUs: A Performance Study*. In: *Proceedings of the 2nd International Workshop on In-Memory Data Management and Analytics (IMDM)*, 2014.

[CBC95] Cunha, Carlos R, Azer Bestavros und Mark E Crovella: *Characteristics of WWW client-based traces*. Technischer Bericht, Boston University Computer Science Department, 1995.

[CBO14] Cagri Balkesen, Jens Teubner, Gustavo Alonso und M. Tamer Öszu: *Main-Memory Hash Joins on Modern Processor Architectures*. 2014.

[Cri09] Crilly, Tony: *50 Schlüsselideen Mathematik*. Spektrum Akademischer Verlag, 2009.

[Far11] Farkisch, Kiumars: *Data-Warehouse-Systeme kompakt: Aufbau, Architektur, Grundfunktionen*. Springe Berlin Heidelberg, 2011.

[HLH13] He, Jiong, Mian Lu und Bingsheng He: *Revisiting Co-processing for Hash Joins on the Coupled CPU-GPU Architecture*. Proc. VLDB Endow., 6(10):889–900, August 2013.

[HP07] Hennessy, J.L. und D.A. Patterson: *Computer Architecture: A Quantitative Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2007.

[Joh99] Johnson, Theodore: *Performance measurements of compressed bitmap indices.* In: *Proceedings of the 25th International Conference on Very Large Data Bases*, Seiten 278–289. Morgan Kaufmann Publishers Inc., 1999.

[KKS+08] Kumar, Sanjeev, Daehyun Kim, Mikhail Smelyanskiy, Yen-Kuang Chen, Jatin Chhugani, Christopher J. Hughes, Changkyu Kim, Victor W. Lee und Anthony D. Nguyen: *Atomic Vector Operations on Chip Multiprocessors.* In: *Computer Architecture, 2008. ISCA '08. 35th International Symposium*, Seiten 441–452, Juni 2008.

[KRES13] Kuckartz, Udo, Stefan Rädiker, Thomas Ebert und Julia Schehl: *Statistik: Eine verständliche Einführung.* VS Verlag für Sozialwissenschaften, 2., überarb. Aufl. 2013 Auflage, 2013.

[KST09] Kühl, S., P. Strodtholz und A. Taffertshofer: *Handbuch Methoden der Organisationsforschung: Quantitative und Qualitative Methoden.* VS Verlag für Sozialwissenschaften, 2009.

[LS97] Lehmann, Ingmar und Wolfgang Schulz: *Mengen - Relationen - Funktionen: eine anschauliche Einführung.* Mathematik-ABC für das Lehramt. Vieweg+Teubner Verlag, 1997.

[MBK02] Manegold, Stefan, Peter Boncz und Martin Kersten: *Optimizing main-memory join on modern hardware.* Knowledge and Data Engineering, IEEE Transactions on, 14(4):709–730, 2002.

[MTA09] Mueller, Rene, Jens Teubner und Gustavo Alonso: *Data Processing on FPGAs.* Proc. VLDB Endow., 2(1):910–921, August 2009.

[Rah94] Rahm, Erhard: *Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung.* Addison-Wesley, 1994.

[Sar90] Sarris, Viktor: *Methodologische Grundlagen der Experimentalpsychologie. 1. Erkenntnisgewinnung und Methodik der experimentellen Psychologie.* UTB.: Grosse Reihe. Reinhardt Verlag, 1990.

[Sch07] Schubert, Matthias: *Datenbanken: Theorie, Entwurf und Programmierung Relationaler Datenbanken.* Lehrbuch : Informatik. Teubner Verlag, 2007.

[SWL11] Schlegel, Benjamin, Thomas Willhalm und Wolfgang Lehner: *Fast Sorted-Set Intersection using SIMD Instructions.* In: *ADMS@ VLDB*, Seiten 1–8, 2011.

[WOS02] Wu, Kesheng, E.J. Otoo und A Shoshani: *Compressing bitmap indexes for faster search operations.* In: *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, Seiten 99–108, 2002.

[WOS06] Wu, Kesheng, Ekow J. Otoo und Arie Shoshani: *Optimizing Bitmap Indices with Efficient Compression.* ACM Trans. Database Syst., 31(1):1–38, März 2006.

[Zip41] Zipf, George Kingsley: *National unity and disunity; the nation as a bio-social organism.* Bloomington, Ind., The Principia press, inc., 1941.

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift