technische universität
dortmund

Masterarbeit

Effiziente String-Verarbeitung in Datenbankanfragen auf hochgradig paralleler Hardware

Florian Lüdiger
Juni 2019

Gutachter:
Prof. Dr. Jens Teubner
Henning Funke

Technische Universität Dortmund
Fakultät für Informatik
Datenbanken und Informationssysteme (LS-6)
http://dbis.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

1.2 Aufbau der Arbeit

Kapitel 2

Grundlagen des verwendeten Verfahrens

2.1 Grundlagen zu Grafikkarten und deren Programmierung

2.2 Das Lane-Refill-Verfahren

Kapitel 3

Einfacher, paralleler String-Vergleich

Für die Evaluation des Lane-Refill-Verfahrens für die Verarbeitung von String-Daten wird zunächst ein einfacher String-Vergleich auf einer GPU untersucht. Ein Vergleich auf Gleichheit ist dabei die einfachste Variante von String-Verarbeitung, die vom Lane-Refill profitieren könnte. Diese Untersuchung wird dabei helfen, zu erfahren, ob die Anwendung des Lane-Refill-Verfahrens bei String-Daten allgemein Potenzial dafür bietet, den Durchsatz entsprechender Anwendungen zu erhöhen.

Zunächst wird dazu ein String-Vergleich mittels der CUDA Schnittstelle ohne spezielle Optimierungen implementiert, um einen Vergleich mit der optimierten Version durchführen zu können. Anschließend werden potenzielle Schwachstellen analysiert und daraus Ansatzpunkte für Optimierungen durch das Auffüllen leer gelaufener Lanes erarbeitet. Daraufhin wird eine Umsetzung vorgestellt, die diese Optimierungen enthält und somit durch die Verwendung geeigneter Workloads mit dem naiven Verfahren verglichen werden kann. Schließlich werden die erhaltenen Messergebnisse vorgestellt und analysiert werden.

3.1 Naive Umsetzung des Gleichheitstests

Als Basis für die Untersuchung wird zunächst der Gleichheitstest für Strings naiv, also ohne tiefgehende Optimierungen umgesetzt. Dies gibt Gelegenheit dazu, die Programmierung einfacher Algorithmen mithilfe der CUDA Schnittstelle für Grafikkarten darzustellen. Da die Analyse im Rahmen dieser Arbeit innerhalb einer Pipelining-Umgebung durchgeführt werden, lassen sich hier außerdem einige Besonderheiten der Implementierung erläutern.

```
__global__
void naiveKernel(
int *char_offset,          // indices of the first letter of every string
char *data_content,        // concatenated list of compare strings
char *search_string,       // string that will be searched for
int search_length,         // length of the search string
int line_count,            // number of lines in the data set
int *number_of_matches) { // return value for the number of matches

  // global index of the current thread,
  // used as the iterator in this case
  unsigned loop_var = ((blockIdx.x * blockDim.x) + threadIdx.x);

  // offset for the next element to be computed
  unsigned step = (blockDim.x * gridDim.x);

  bool active = true;
  bool flush_pipeline = 0;

  while(!flush_pipeline) {
    // element index must not be higher than line count
    active = loop_var < line_count;

    // break computation when every line is finished and therefore inactive
    flush_pipeline = !__ballot_sync(ALL_LANES, active);

    data_length = char_offset[loop_var+1] - char_offset[loop_var] - 1;

    // if the lengths of the strings don't match,
    // the string can be discarded immediately
    if (active && data_length != search_length)
      active = false;

    int search_id = 0;

    // iterate over both strings till the end
    // or until a non-matching character has been found
    while(__any_sync(0xFFFFFFFF, active) && search_id < search_length) {

      int data_id = search_id + char_offset[loop_var];

      // when strings don't match, inactivate the lane
      if (active && data_content[data_id] != search_string[search_id])
        active = false;

      search_id++;
    }
```

```
48
49      // when comparison finishes without being inactivated,
50      // a match has been found
51      if (active)
52      atomicAdd(number_of_matches, 1);
53
54      loop_var += step;
55    }
56  }
```

Listing 3.1: Naive Implementierung des String-Vergleichs

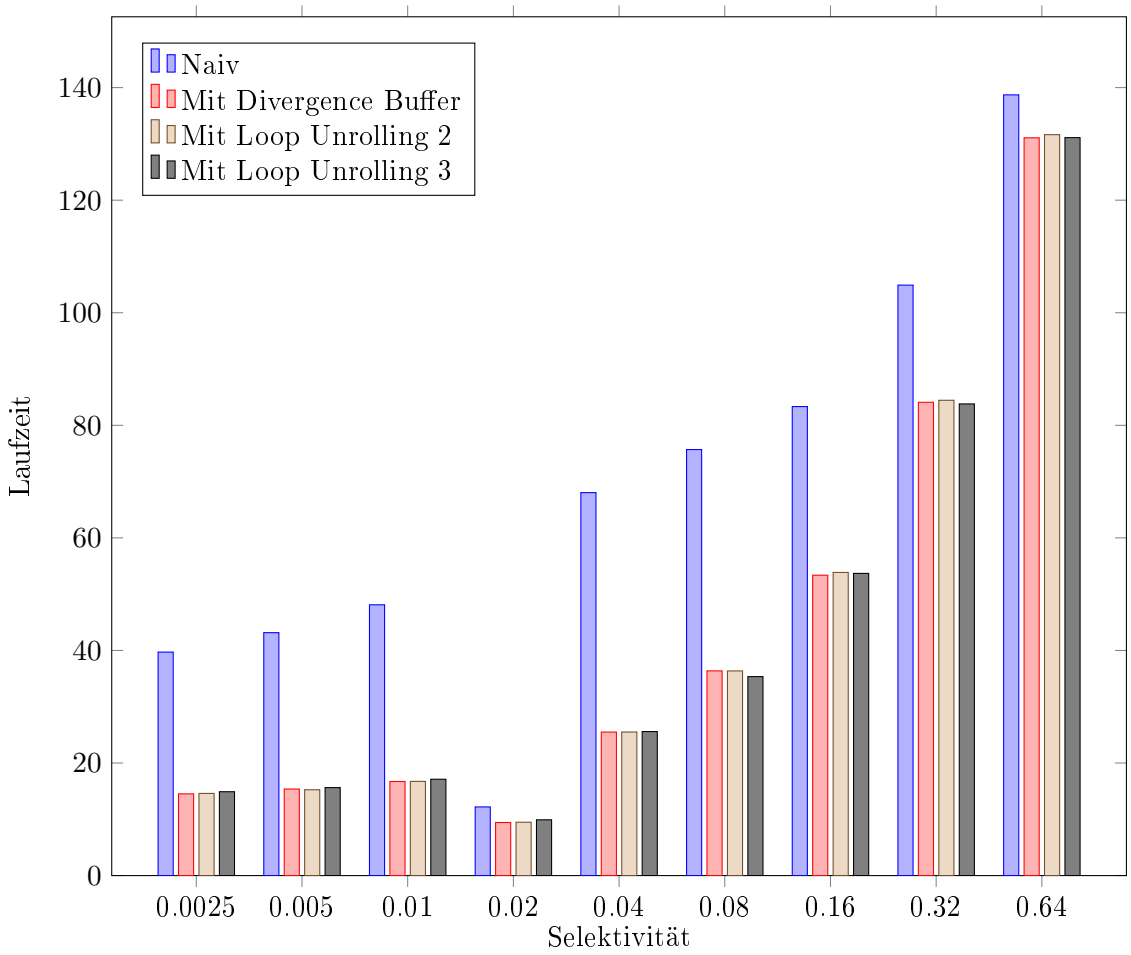
3.2 Ansatzpunkte für Lane Refill

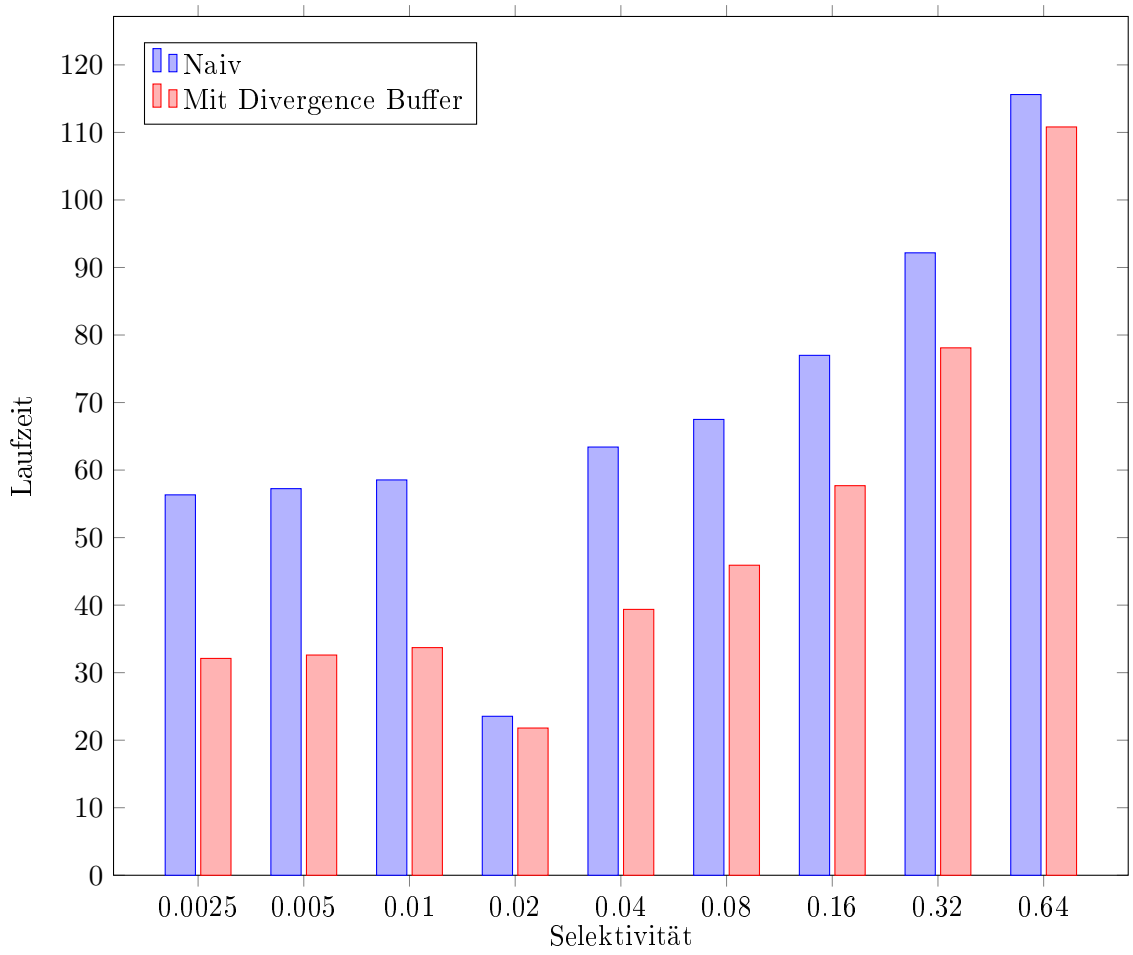
3.3 Umsetzung mit Lane Refill

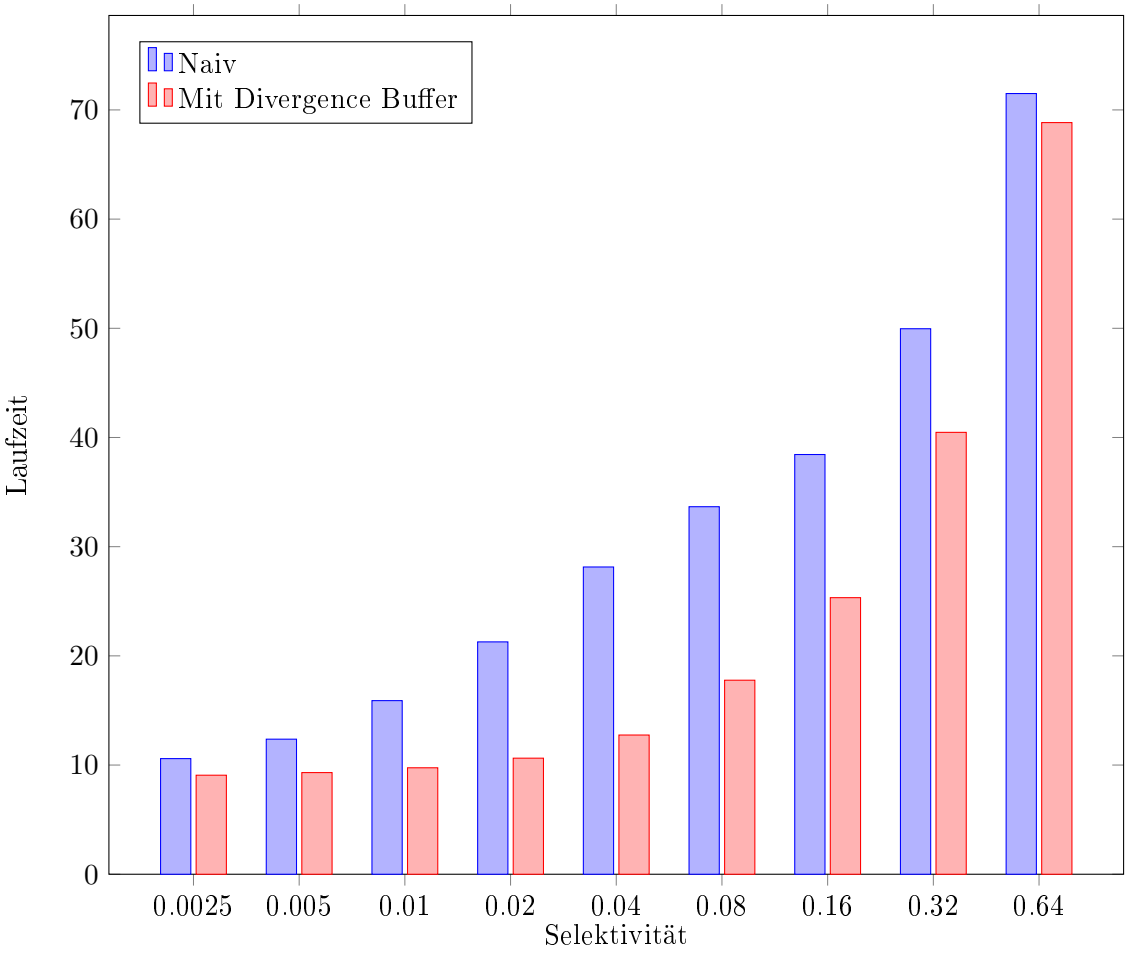
3.4 Präfixtest als alternativer Workload

3.5 Verwendete Workloads und deren Merkmale

3.6 Vorstellung der Messergebnisse




Naiv
Mit Divergence Buffer
Laufzeit
0
10
20
30
40
50
60
70
80
90
100
110
120
0.0025
0.005
0.01
0.02
0.04
0.08
0.16
0.32
0.64
Selektivität



3.7 Diskussion der Ergebnisse

Kapitel 4

Paralleler Musterabgleich mit regulären Ausdrücken

Kapitel 5

Ergebnis und Fazit

Anhang A

Weitere Informationen

Abbildungsverzeichnis

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 27. Januar 2019

Florian Lüdiger