Massively-Parallel Lossless Data Decompression

Seminar Supercomputer

Sebastian Hanisch

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2022-07-12

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

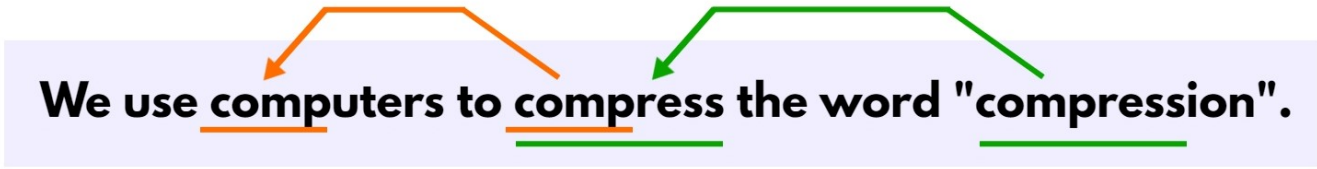
informatik
die zukunft

Gliederung (Agenda)

Einleitung und Motivation

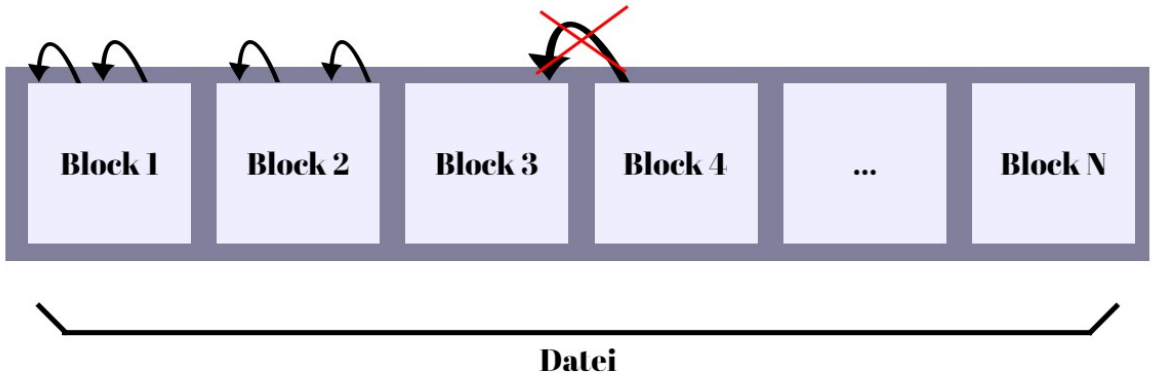
- Kompression in Big Data
  - Exponentiell mehr Daten
  - Hohe Speicherkosten
- Meiste Forschung in schnellere Kompression
- Wiederholte Zugriffe -> schnelle Dekompression
- Antwortzeit reduzieren
- Parallelismus der GPU ausnutzen

LZ77 Kompression



- 2 Byte Pointer
  - Pointer: <Sprung, Länge>
  - <12Bit, 4Bit> == <0-4095, 0-15>

Datenblöcke



- Inter-Block Parallelismus möglich
- Ziel: Intra-Block Parallelismus

GPU Architektur - NVIDIA GTX 1080 Ti

- 28 Streaming Multiprocessors (SM)
- 128 CUDA Kerne pro SM
- 2048 Threads in flight pro SM
- 32 Threads pro Warp
- SIMT - Single Instruction Multiple Threads

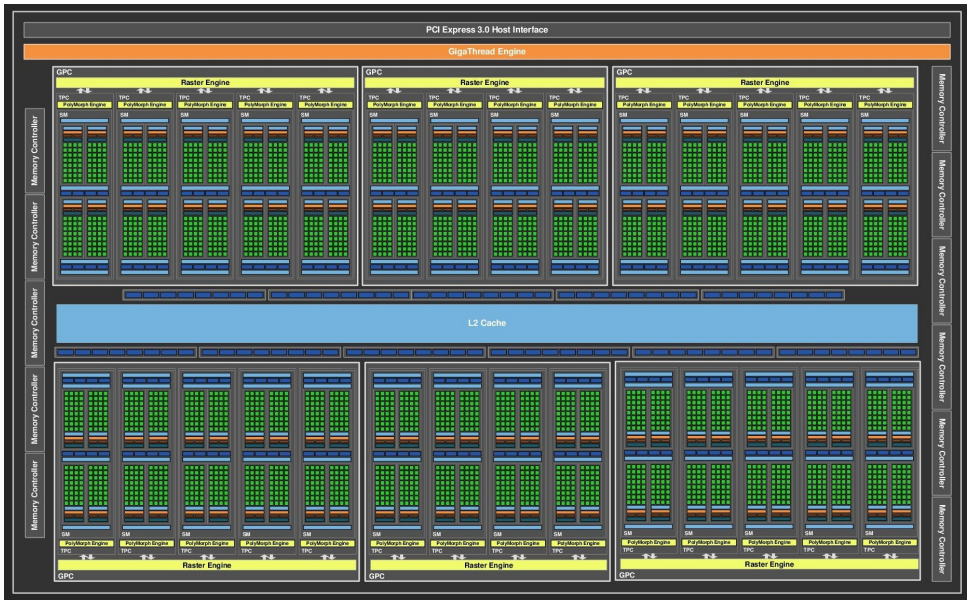


Abbildung: GTX 1080 Ti Architektur:
https://bilder.pcwelt.de/4073108_original.jpg

1) GPU Dekompressions-Algorithmus

- Huffman Dekodierung
  - Ein Thread pro Sub-Block
  - Lookup-Tabellen anstatt Zweige
  - Keine Abweichungen im Warp
- LZ77 Dekompression
  - 1 Daten-Block pro Warp
  - Buchstabenkette + Pointer = Sequenz

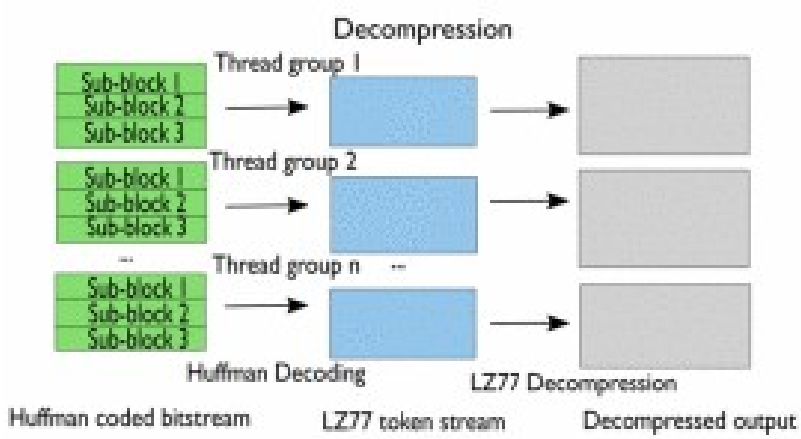
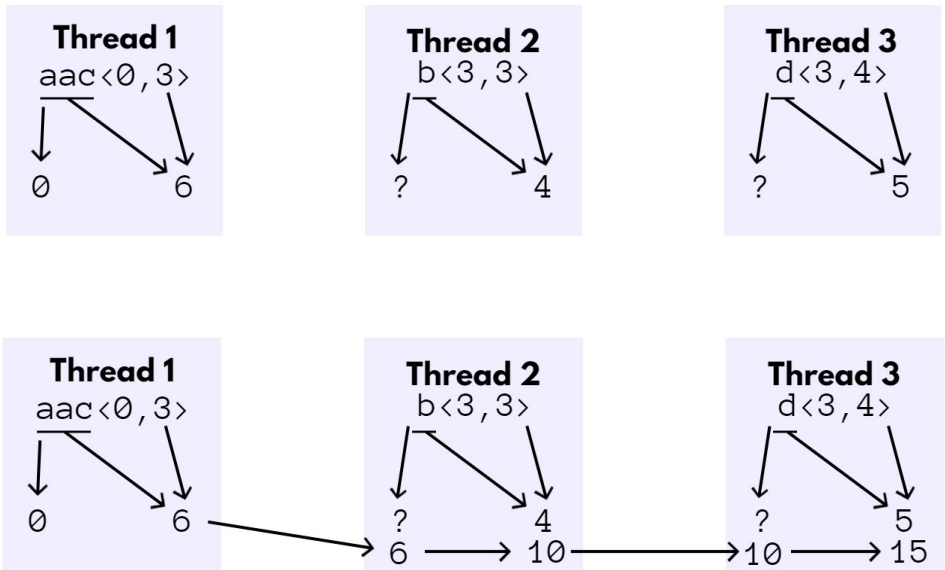


Abbildung: DOI: 10.1109/ICPP.2016.35 Fig. 1, 2

Speicherort über Warp Shuffling finden

- Sequenz wird Thread zugeordnet



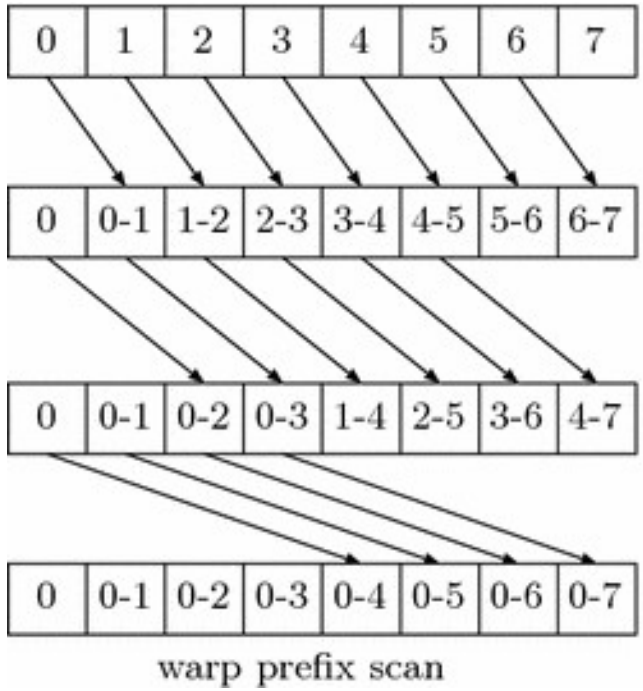
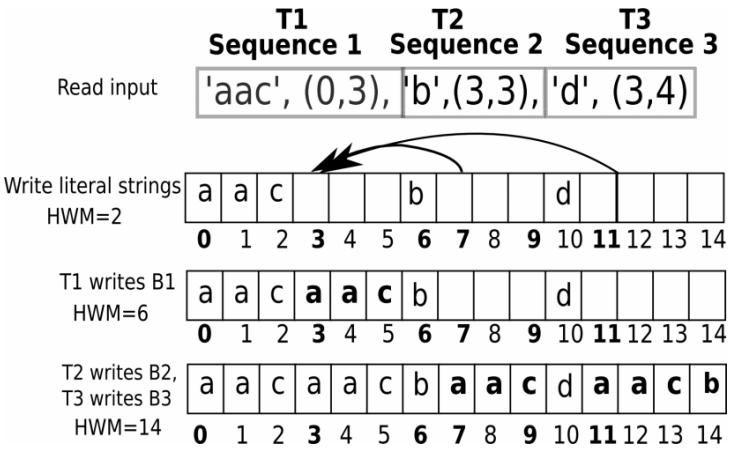


Abbildung: Warp Shuffling Prefix Scan: https://www.researchgate.net/figure/Illustrating-warp-prefix-scan-and-diagonal-arrangement-for_fig2_322536257

Multi-Round Resolution (MRR) Algorithmus

- Um Pointer aufzulösen -> MRR ausführen

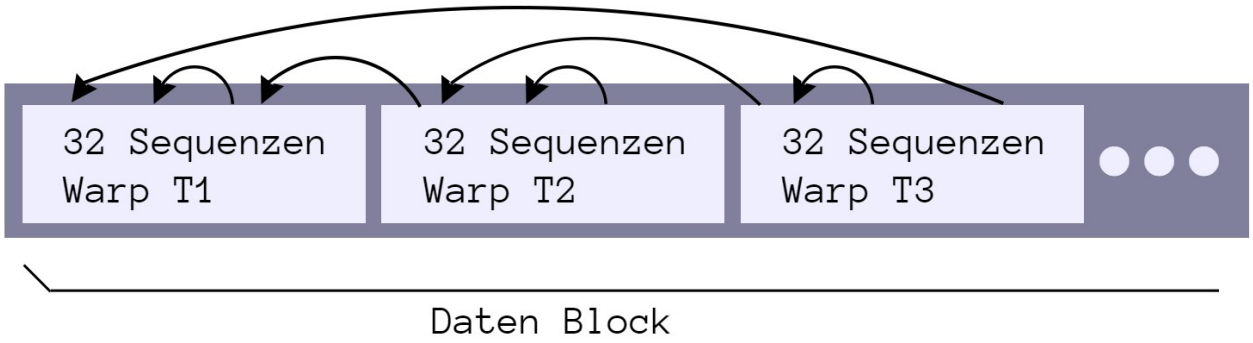



```
1:  function MRR(HWM, read_pos, write_pos, length)
2:      pending ← true          ▷ thread has not written any output
3:      do
4:          if pending and read_pos+length≤HWM then
5:              copy length bytes from read_pos to write_pos
6:              pending ← false
7:          end if
8:          votes ← ballot(pending)
9:          last_writer ← count_leading_zero_bits(votes)
10:         HWM ← shfl(write_pos+length, last_writer)
11:     while votes> 0              ▷ Repeat until all threads done
12:     return HWM
13: end function
```

Abbildung: DOI: 10.1109/ICPP.2016.35 Fig. 2, 3

	Thread 1	Thread 2	Thread 3	Thread 4
pending	False	False	False	True
votes	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1
last_writer	3	3	3	3
shfl	shfl(5,3)	shfl(9,3)	shfl(14,3)	shfl(23,3)
HWM	14	14	14	14

Abhängigkeiten in Datenblöcke



- Abhängigkeiten in Warp zu Zeit T -> MRR Algorithmus
- Abhängigkeiten aufgelöst -> nächste 32 Sequenzen

2) Abhängigkeits Elimination (DE) Dependency Elimination

- Ziel: Keine Abhängigkeiten in einem Warp
- Kompressionsrate vs. Dekompressions-Geschwindigkeit

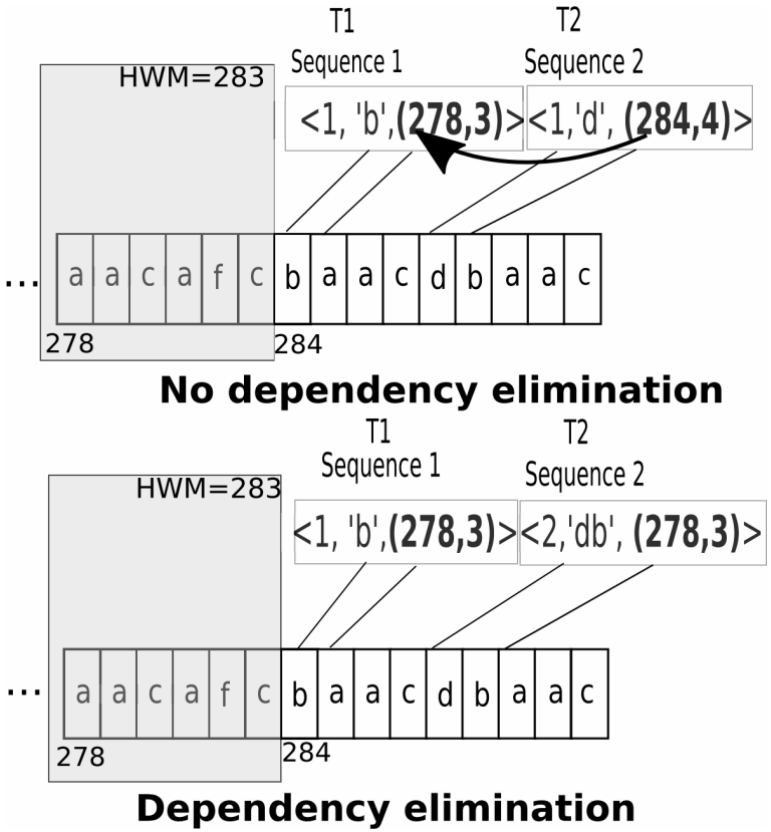


Abbildung: DOI: 10.1109/ICPP.2016.35 Fig. 5

Testdaten und Hardware

- Datensätze
  - 1GB XML Datei des Englischen Wikipedia – Gzip 3,09:1
  - 0.77 GB Matrix Market Datei – Gzip 4,99:1
- Hardware
  - CPU: E5-2620 v2 CPU, 2x6 Kern mit 24 Hardware Threads
  - GPU: NVIDIA Tesla K40 mit 2880 CUDA Kernen (1440 Warps)

Einfluss von Abhängigkeiten auf Performanz

- SC = Sequenzielles Kopieren (ohne intra-Block Parallelisierung)
- DE mindestens 5x schneller als SC

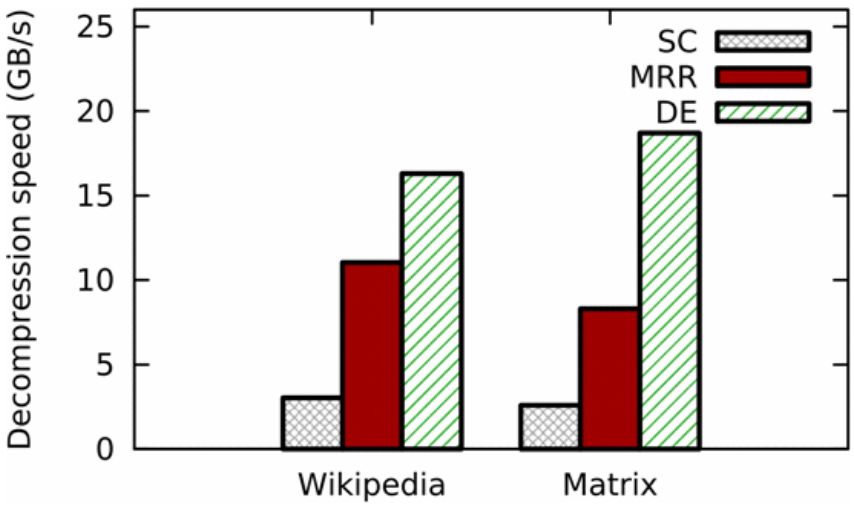


Abbildung: DOI: 10.1109/ICPP.2016.35 Fig. 6

Einfluss von DE auf Kompressionsrate und Geschwindigkeit

- Kompressionsrate und Geschwindigkeit: LZ4 vs. LZ4 + DE
  - Geschwindigkeitsverlust: max. 13%
  - Verlust Kompressionsrate: max. 19%

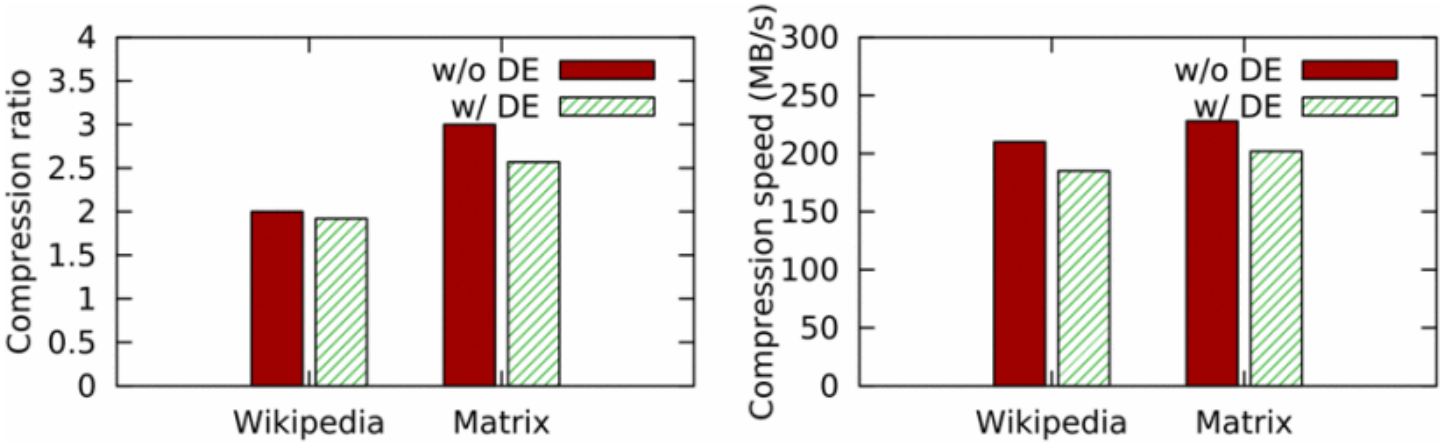


Abbildung: DOI: 10.1109/ICPP.2016.35 Fig. 7

GPU vs. Multi-core CPU Performanz

- Schnellste in ihren Klassen; gleiche Kompressionsrate
- Gompresso/Byte
  - Geschwindigkeit wichtiger als Kompressionsrate (DE)
  - 1,35x schneller als LZ4
- Gompresso/Bit
  - Kompressionsrate wichtiger als Geschwindigkeit (MRR)
  - 2x schneller als zlib

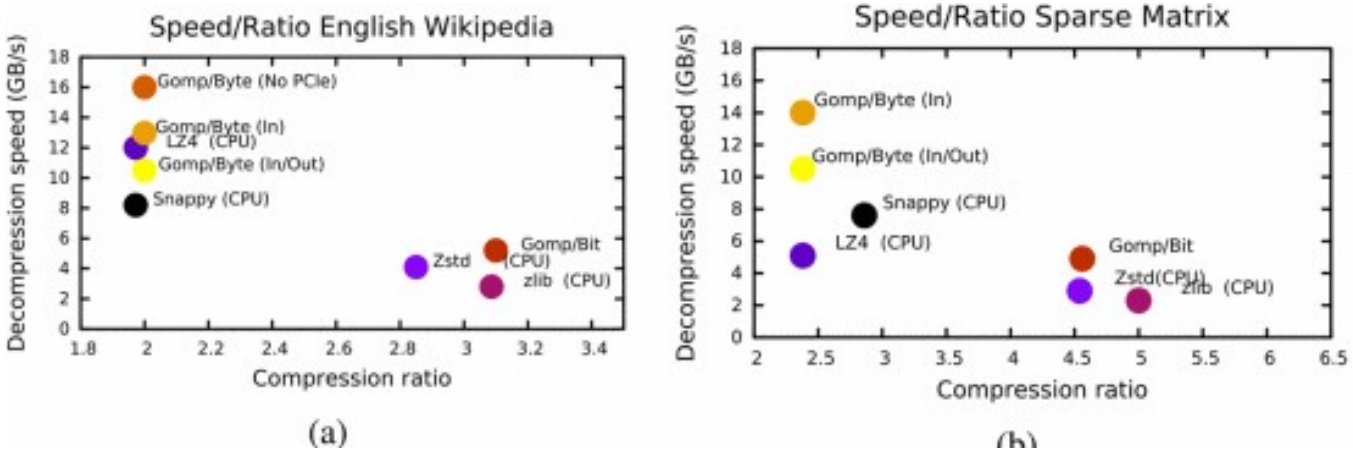


Abbildung: DOI: 10.1109/ICPP.2016.35 Fig. 8

Literatur I

[20120] IIT Kharagpur July 2018. Lecture 16: Warp Scheduling and Divergence. 07 2020.

[Com13] Computerphile. Elegant Compression in Text (The LZ 77 Method) - Computerphile. 01 2013.

[Dev14] Google Developers. The LZ77 Compression Family (Ep 2, Compressor Head). 05 2014.

[ea16] Evangelia Sitaridi et al. Massively-Parallel Lossless Data Decompression. *International Conference on Parallel Processing (ICPP)*, (45):242–247, 08 2016.

[LLC] Epic Games Tools LLC. Introducing Oodle Kraken!

Literatur II

[Lui14] Justin Luitjens. Faster Parallel Reductions on Kepler. 02 2014.

[Mel20] Chris Mellor. Enterprise SSDs cost ten times more than nearline disk drives. 08 2020.

[Pla20] PlayStation. The Road to PS5. 03 2020.

[Uda15] Udacity. Thread Blocks And GPU Hardware - Intro to Parallel Programming. 02 2015.

[Uni] Cornell University. Introduction to GPGPU and CUDA Programming: SIMT and Warp.

[Yan18] Shi Yan. Some CUDA concepts explained. 02 2018.