

# ERA Praktikum - Entropie (A406) - Gruppe 233

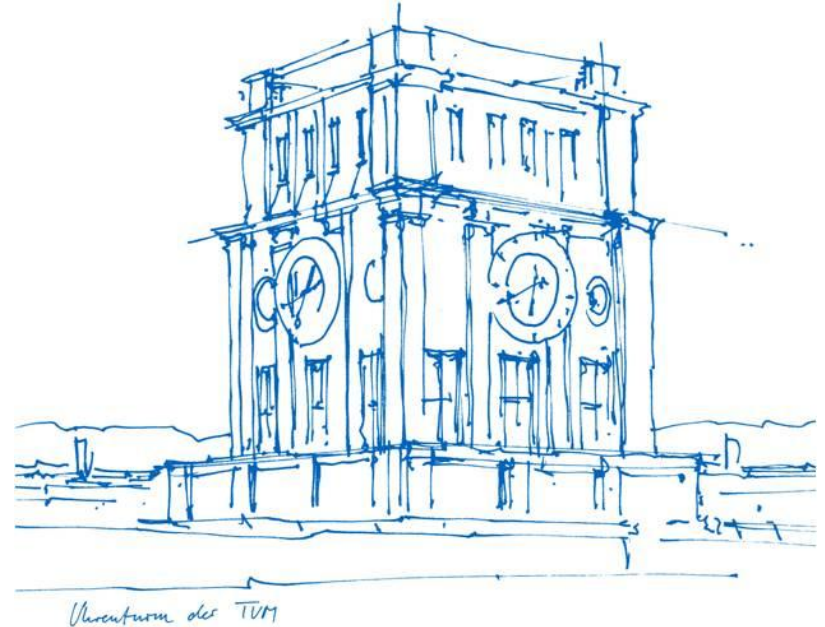
Fikret Ardal | Denis Paluca | Mert Corumlu

Technische Universität München

Fakultät für Informatik

Lehrstuhl für Rechnerarchitektur und Parallele Systeme

München, Juli 2021



# Inhalt

1. Einführung
2. Lösungsansatz
3. Genauigkeit
4. Performanz
5. Zusammenfassung

# Einführung

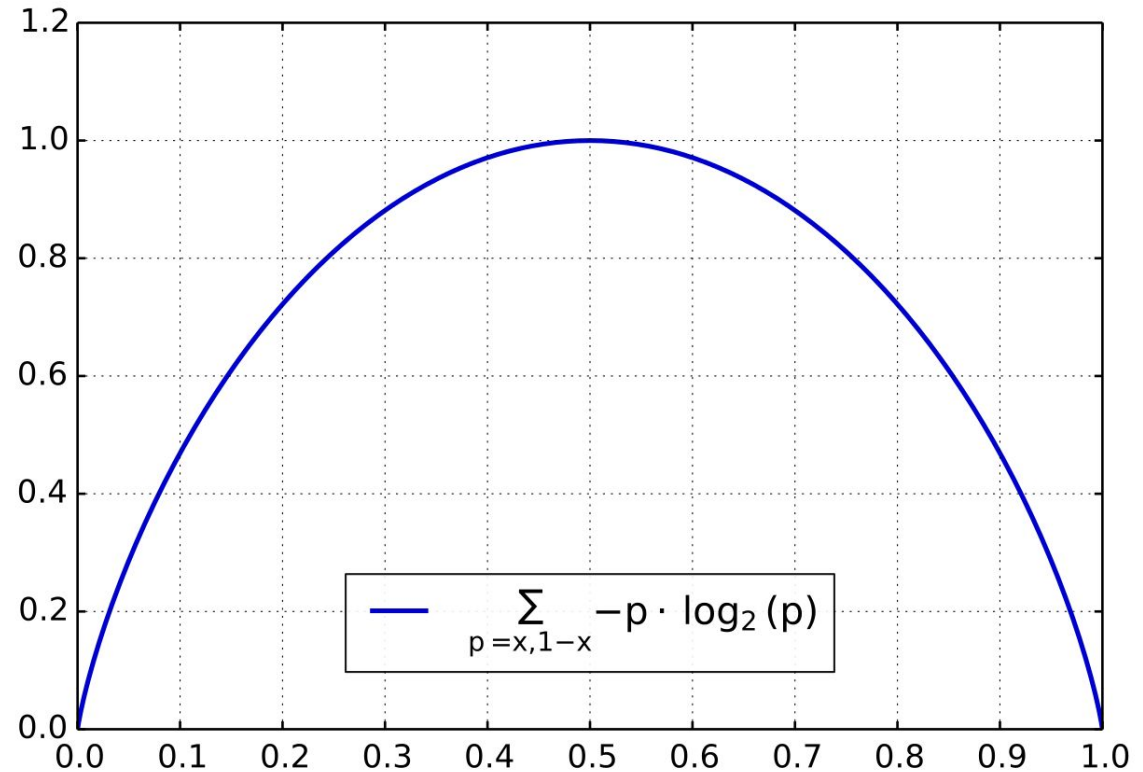
- Entropie

$$H(X) = - \sum_{x \in X} P(X = x) \cdot \log_2(P(X = x))$$

$$0 \leq H(X) \leq \log_2(|X|)$$

# Einführung

## Entropie eines Münzwurfs



# Einführung

Eingabe

<b>Outcome of die roll</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>
<b>Probability</b>	1/6	1/6	1/6	1/6	1/6	1/6

$$0 \leq p(x) \leq 1$$

$$\sum p(x) \approx 1.0$$

# Lösungsansatz

## Hauptprobleme bei der Entropiefunktion

1. Rundungsfehler -> Kahan's Summe
2. Logarithmusfunktion

$$H(X) = - \sum_{x \in X} P(X = x) \log_2(P(X = x))$$

# Lösungsansatz

## Logarithmusfunktion

- Approximation als Polynom
  - ARTANH Approximation
  - Remez Algorithmus
- Lookup-Tabelle
- Approximation als Polynom mit Hilfe einer Lookup-Tabelle
  - Glibc

# Lösungsansatz

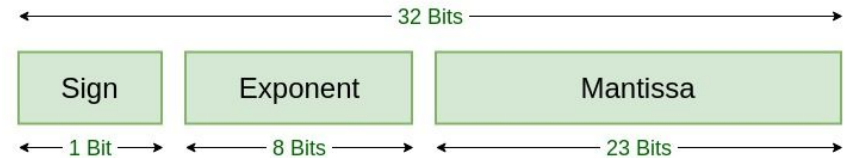
## Extrahieren des Exponenten

$$x = 2^k \cdot z$$

$$\log_2(x) = k + \log_2(z)$$

$$\log_2(x) = k + \ln(z)/\ln(2)$$

- Bei normalisierte Gleitkommazahlen liegt  $z$  zwischen 1 und 2
- Nur die Approximation der Logarithmusfunktion zwischen 1 und 2 notwendig





# Lösungsansatz

## ARTANH Approximation

$$\ln(z) = 2 \cdot \operatorname{artanh} \left( \frac{z-1}{z+1} \right) = 2 \left( \sum_{k=0}^{\infty} \frac{1}{2k+1} \cdot \left( \frac{z-1}{z+1} \right)^{2k+1} \right)$$

$$S_n = 2 \left( \sum_{k=0}^n \frac{1}{2k+1} \cdot \left( \frac{z-1}{z+1} \right)^{2k+1} \right)$$

- Wenn die Zahl eine Zweierpotenz ist, liefert dies genaues Ergebnis

# Lösungsansatz

## REMEZ Algorithmus

- Mini-Max Approximationsalgorithmus für bestimmtes Intervall

Das Algorithmus liefert für  $\log_2(z)$  im Intervall  $[1,2)$ :

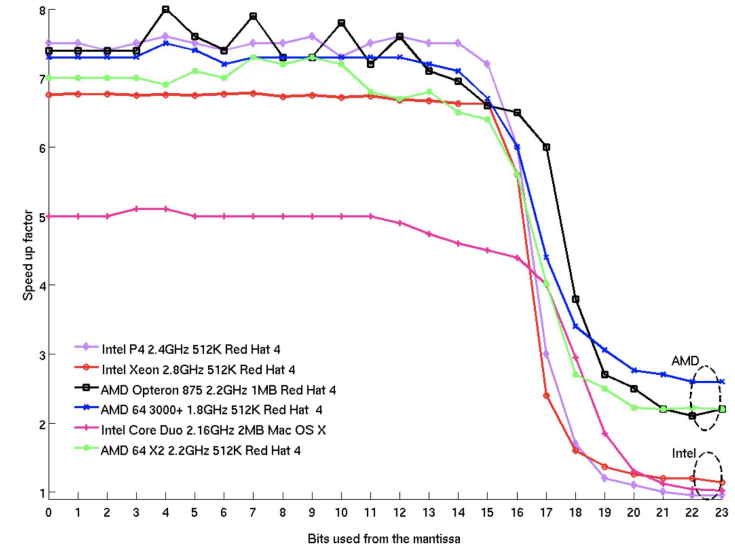
$$P_2(z) = -0.344845 \cdot z^2 + 2.024658 \cdot z - 1.674873$$

$$P_4(z) = -0.081616 \cdot z^4 + 0.645142 \cdot z^3 - 2.120675 \cdot z^2 + 4.070091 \cdot z - 2.512854$$

# Lösungsansatz

## Lookup Tabelle

- Speicherung der  $\log_2(z)$  Werte statt rechnen
- Insgesamt  $2^{23}$  Einträge notwendig für 100% Genauigkeit
- Wir benutzen trotzdem  $2^{16}$  Einträge
- Trade-off zwischen Genauigkeit und Performanz
- 100% Genauigkeit bei Zweierpotenzen



# Lösungsansatz

## Glibc Methode

$$\log_2(x) = k + \log_2(z/c) + \log_2(c)$$

- Mittels z findet man in einer Lookup-tabelle c und  $\log_2(c)$
- $z/c$  ist sehr nahe an 1
- $\log_2(z/c)$  wird mittels Taylor-Reihe approximiert  $\ln(x + 1) \approx x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4}$
- 100% Genauigkeit bei Zweierpotenzen

# Lösungsansatz

## Denormalen Zahlen

- Bei den denormalen Gleitkommazahlen liegt  $z$  zwischen 0 und 1.
- Unsere Methoden funktionieren im Intervall  $[1,2)$

$$\log_2(x \cdot 2^{23} / 2^{23}) = \log_2(x \cdot 2^{23}) - \log_2(2^{23}) = \log_2(y) - 23$$

- Wir multiplizieren die Zahl mit  $2^{23}$  und subtrahieren 23 von dem Exponent
- Wir führen die normale Schritte der Logarithmusfunktion für  $y$

# Genauigkeit

$$H(X) = - \sum_{x \in X} P(X = x) \cdot \log_2(P(X = x))$$

$$\text{FehlerAbsMax} = \text{Max}\{ | \log_2(x) - \log_2_{\text{approx}}(x) | \}$$

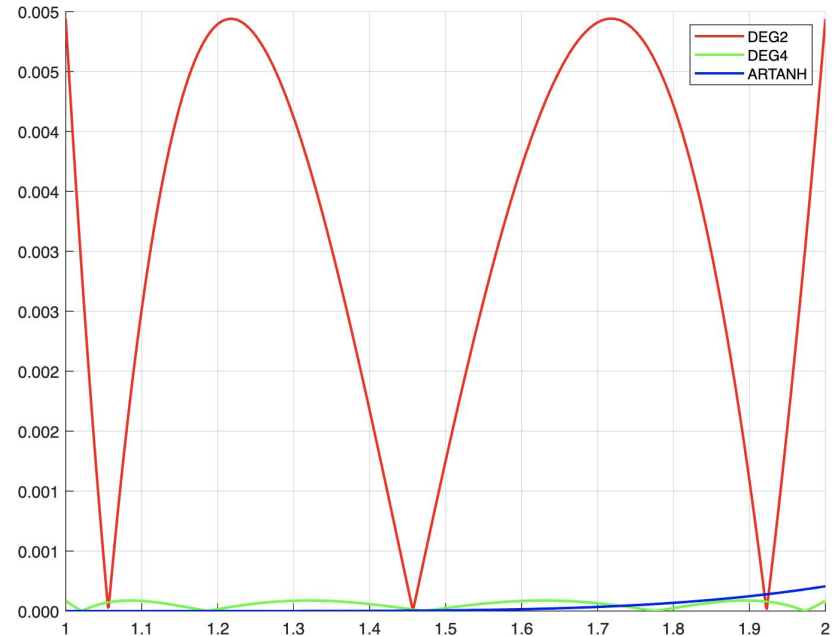
# Genauigkeit

## Maximale Absolute Fehler

- DEG2: 4,491  $\times 10^{-3}$
- DEG4: 8,752644  $\times 10^{-5}$
- ARTANH: 2,063996  $\times 10^{-4}$

## Durchschnittlicher Fehler

- Lookup-Tabelle: 6.55  $\times 10^{-6}$



# Genauigkeit

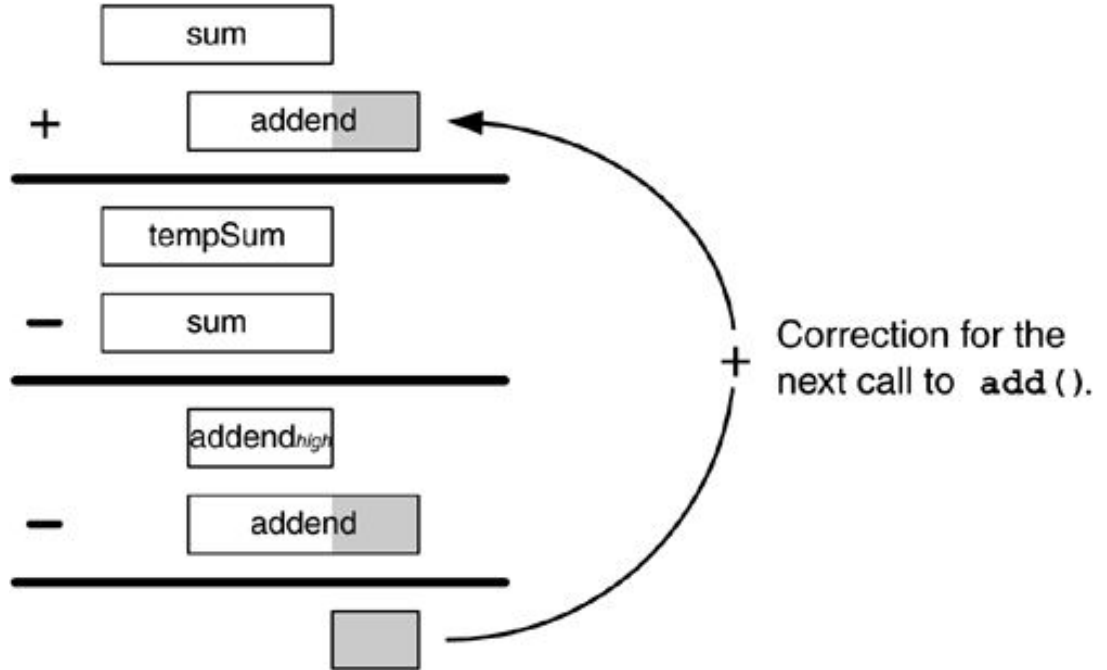
## Rundungsfehler

- Sequentielle Addition
- Zahlen werden auf Zwischensumme addiert
- Zwischensumme wird immer größer
- Stellen der nächste Zahl werden ignoriert



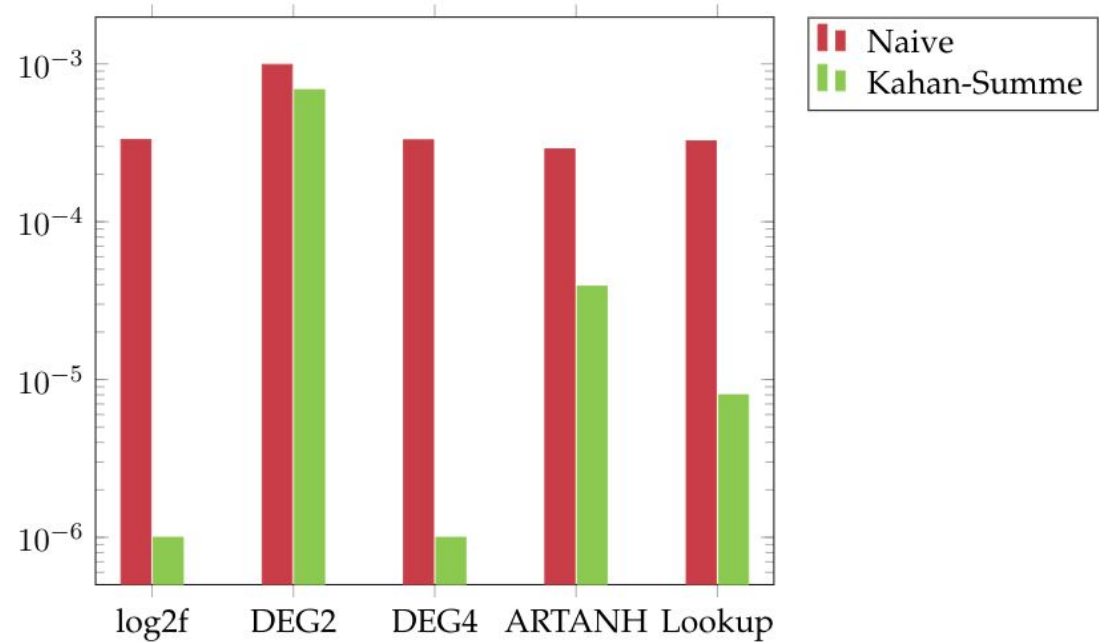
# Genauigkeit

## Kahan-Summe-Algorithmus



# Genauigkeit

Entropiefehler Größenordnung



# Performanz

Optimierungen unter 5 Hauptüberschriften :

- SIMD
- Speicher Allokation
- Ersparen von Branches
- ABI Verstoß

# Performanz

## SIMD

- Die Entropiefunktion ist leicht zu vektorisieren.
- Spezielle vektorisierte Logarithmusfunktionen.
- Maximale Speed-up von 4.
- Auch Lookup-Table Ansatz ist vektorisierbar.
- Speedup kleiner als Approximations.

# Performanz

## Speicher Allokation

- Wir stellen uns Sicher, dass die Länge der Entropiefunktion übergebene Float-Array immer ein Vielfaches 4 ist.
- Der Rest wird mit Nullen erfüllt.
- Damit keine skalare Schritte in SIMD-Funktion benötigt.
- Speicheralkotation mit *aligned\_alloc()*
- *movaps* Statt *movups*

Beispiel für [0.1, 0.2, 0.3, 0.4, 0.5]

Allokation im Heap	0x...01c	0	}	Zusätzliche Allokation
	0x...018	0		
	0x...014	0		
	0x...010	0.5		
	0x...00c	0.4		
	0x...008	0.3		
	0x...004	0.2		
	0x...000	0.1		

Startadresse →

# Performanz

## Ersparen von Branches

- Auslassen einige Randfälle
- NaN, +- Infinity, 0, negative Zahlen
- Die Korrektheit der Zahl wird in der Entropiefunktion überprüft
- Damit sparen wir einige Instruktionen

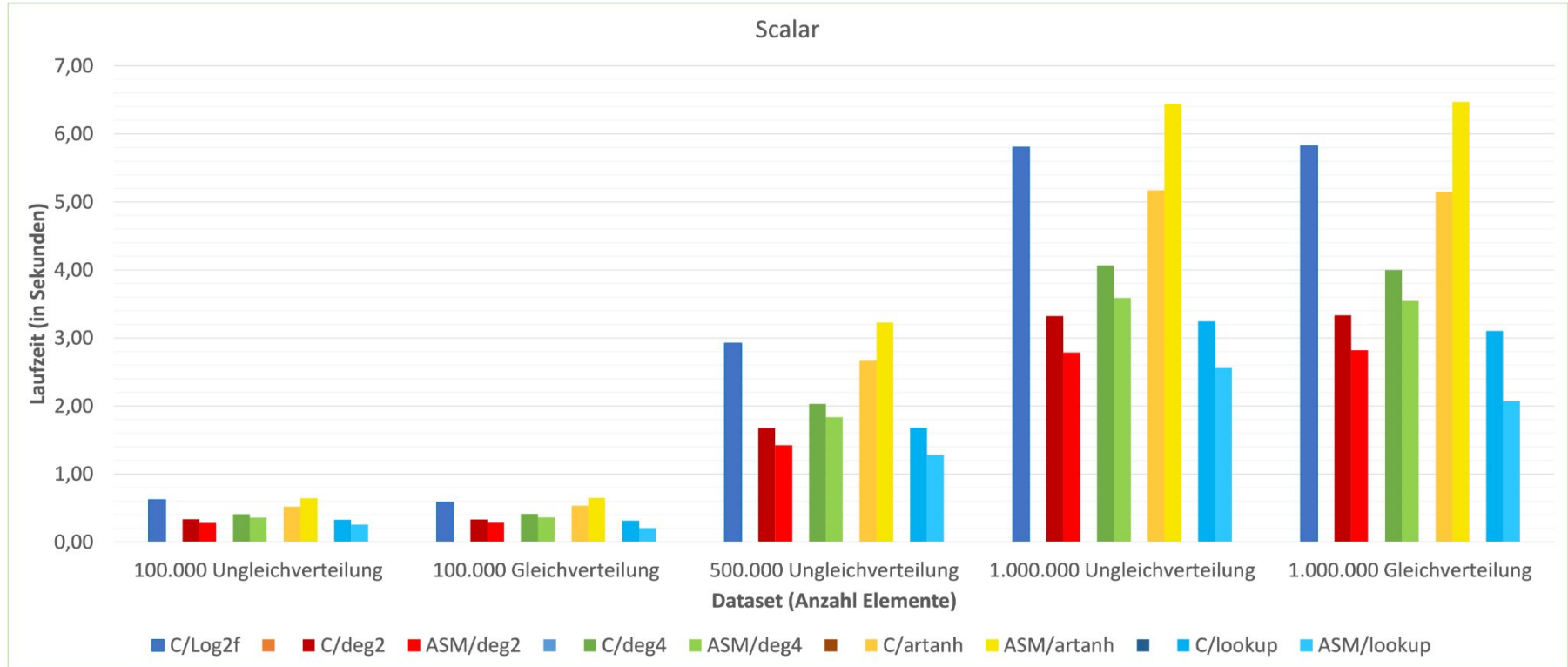
$$P(X = x) \cdot \log_2(P(X = x))$$

# Performanz

## ABI Verstoß

- Wir wissen genau welche Registern in Logarithmusfunktion verändert wird
- Caller-Saved Register werden nicht in Entropiefunktion gesichert
- Kein Stack benötigt

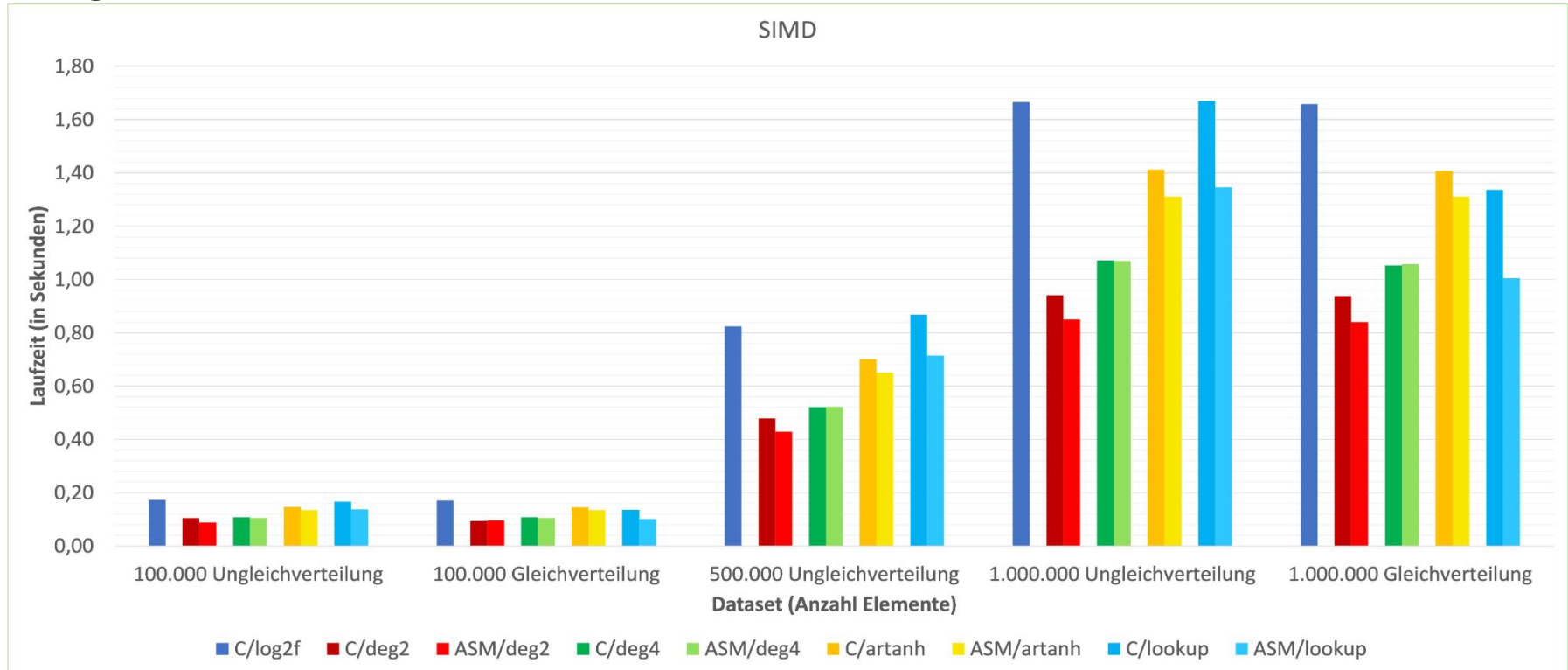
# Vergleich unter verschiedenen skalaren Implementationen



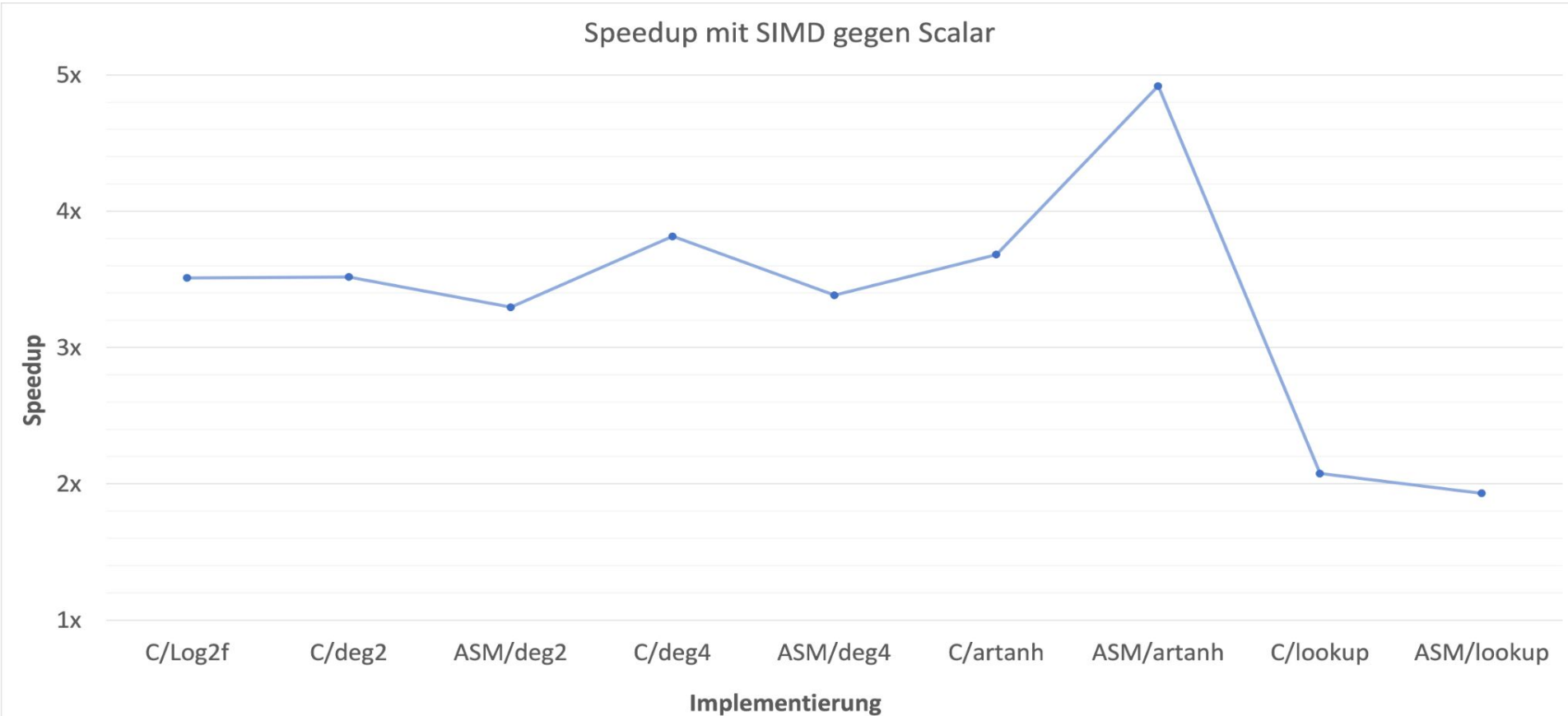
1000 Iterationen mit gcc 11.2.0 -O3 auf i7-10750H @ 2.6 GHz im Arch Linux kompiliert



# Vergleich unter verschiedenen SIMD Implementationen



1000 Iterationen mit gcc 11.2.0 -O3 auf i7-10750H @ 2.6 GHz im Arch Linux kompiliert



# Beobachtungen

- Das Programm konnte mittels Assembly weiter optimiert werden
- Alle Implementierungen (inklusive glibc) wurde erfolgreich vektorisiert
- Durch SIMD ist eine Speedup von 3,5x möglich (außer naive Lookup Tabelle)
- Bei der Lookup Tabelle ist die Speedup durch SIMD maximal 2x wegen der notwendigen Speicherzugriffe
- Die Laufzeit von Gleichverteilung ist besser als Ungleichverteilung wegen der Cache-Misses

# Zusammenfassung

- Die Entropiefunktion wurde mit unterschiedlichen Algorithmen erfolgreich implementiert
- Das Umsetzen von SIMD Prinzip führt zu offensichtlicher Steigerung von der Leistung
- Kahans Summe-Algorithmus reduziert deutlich den Rundungsfehler bei der Summe von Fließkommazahlen
- Grundsätzlich ist die DEG4 Implementation zu benutzen

# Danke für Ihre Aufmerksamkeit!

Haben Sie noch Fragen?

