Hochleistungsrechnen - Übungsblatt 5
Lösung: POSIX Threads

Moritz Heift, Björn Wege

15. November 2025

1 Aufgabe 1: Datenaufteilung

1.1 Teil 1.1: Generische Formeln

1.1.1 Grundlegende Überlegungen

Gegeben seien:

- $i \in \mathbb{N}_0$ Interlines
- $T \in \mathbb{N}$ Threads
- Thread-Index $t \in \{0, 1, \ldots, T-1\}$

Die Matrix hat folgende Eigenschaften:

- **Matrixgröße**: Bei i Interlines hat die Matrix die Größe $(N+1) \times (N+1)$ mit

$$N = i \cdot 8 + 9 - 1 = 8i + 8$$

- **Randzeilen**: Die erste Zeile (Index 0) und letzte Zeile (Index N) sind Randzeilen
- **Zu berechnende Zeilen**: Nur die inneren Zeilen mit Indizes 1 bis $N-1$ müssen berechnet werden
- **Anzahl zu berechnender Zeilen**:

$$\text{rows} = N - 1 = 8i + 7$$

1.1.2 Formeln für Thread t (mit $t \in \{0, 1, \ldots, T-1\}$)

Wir berechnen zuerst:

$$\begin{aligned} N &= 8i + 8 \\ \text{rows} &= N - 1 = 8i + 7 \quad \text{(zu berechnende Zeilen)} \end{aligned}$$

1. Anzahl der zu berechnenden Zeilen für Thread t:

$$\text{rows}(t) = \begin{cases} \left\lfloor \frac{\text{rows}}{T} \right\rfloor + 1 & \text{falls } t < (\text{rows} \bmod T) \\ \left\lfloor \frac{\text{rows}}{T} \right\rfloor & \text{sonst} \end{cases}$$

2. Startzeilen-Index (inklusiv) für Thread t:

$$\text{start}(t) = 1 + t \cdot \left\lfloor \frac{\text{rows}}{T} \right\rfloor + \min(t, \text{rows} \bmod T)$$

3. Endzeilen-Index (inklusiv) für Thread t:

$$\text{end}(t) = \text{start}(t) + \text{rows}(t) - 1$$

1.2 Teil 1.2: Berechnungen für konkrete Konfigurationen

1.2.1 Konfiguration 1: $i = 0$, $T = 5$

Mit $i = 0$ ergibt sich:

$$\begin{aligned} N &= 8 \cdot 0 + 8 = 8 \\ \text{Matrixgröße:} \quad & 9 \times 9 \\ \text{rows} &= N - 1 = 7 \quad \text{(zu berechnende Zeilen)} \\ \left\lfloor \frac{7}{5} \right\rfloor &= 1 \\ 7 \bmod 5 &= 2 \end{aligned}$$

Die ersten 2 Threads (Thread 0 und 1) bekommen jeweils 2 Zeilen, die restlichen Threads (2, 3, 4) bekommen jeweils 1 Zeile.

Thread t	Anzahl Zeilen	Startzeile	Endzeile
0	2	1	2
1	2	3	4
2	1	5	5
3	1	6	6
4	1	7	7

Tabelle 1: Aufteilung für Konfiguration 1 ($i = 0, T = 5$)

1.2.2 Konfiguration 2: $i = 1$, $T = 5$

Mit $i = 1$ ergibt sich:

$$\begin{aligned} N &= 8 \cdot 1 + 8 = 16 \\ \text{Matrixgröße:} \quad & 17 \times 17 \\ \text{rows} &= N - 1 = 15 \quad \text{(zu berechnende Zeilen)} \\ \left\lfloor \frac{15}{5} \right\rfloor &= 3 \\ 15 \bmod 5 &= 0 \end{aligned}$$

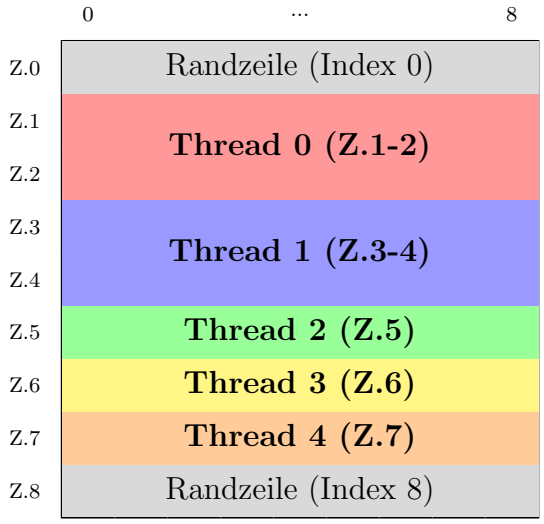
Da $15 \bmod 5 = 0$, bekommen alle 5 Threads jeweils genau 3 Zeilen.

Thread t	Anzahl Zeilen	Startzeile	Endzeile
0	3	1	3
1	3	4	6
2	3	7	9
3	3	10	12
4	3	13	15

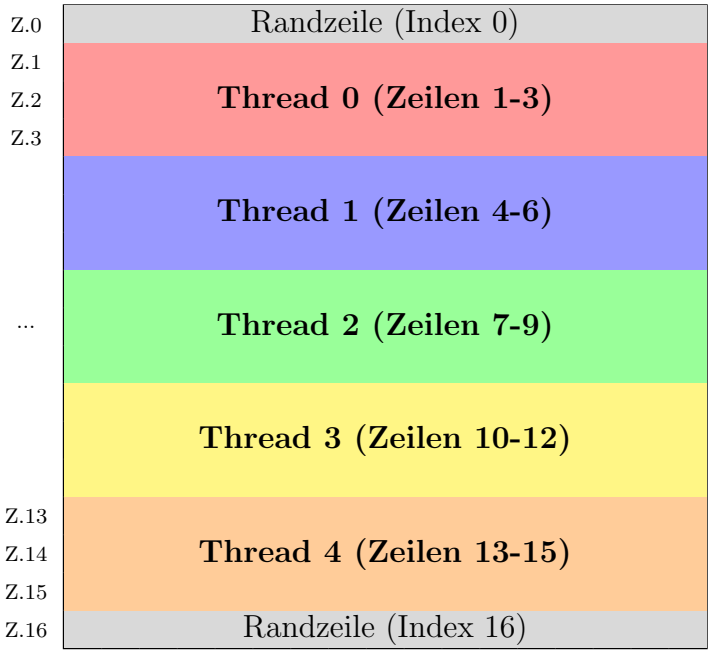
Tabelle 2: Aufteilung für Konfiguration 2 ($i = 1, T = 5$)

1.3 Teil 1.3: Visualisierung der Datenaufteilung

1.3.1 Konfiguration 1: $i = 0$, $T = 5$ (Matrix 9×9)



1.3.2 Konfiguration 2: $i = 1$, $T = 5$ (Matrix 17×17)



2 Aufgabe 3: Leistungsanalyse

2.1 Messaufbau

Hardware:

- Rechenknoten:west
- CPU: 12 Cores

Testparameter:

- Interlines: 512
- Störfunktion: $f(x, y) = 2\pi^2 \sin(\pi x) \sin(\pi y)$
- Methode: Jacobi
- Iterationen: 900
- Messungen: Je 3 Wiederholungen pro Konfiguration

2.2 Messergebnisse

Threads	**Lauf 1 (s)**	**Lauf 2 (s)**	**Lauf 3 (s)**	**Mittel (s)**	**Speedup**
Seq.	371.23	370.45	371.89	371.19	1.00
1	371.91	370.07	371.58	371.19	1.00
2	185.41	184.34	185.57	185.11	2.00
3	125.96	125.58	125.11	125.55	2.96
4	93.85	92.81	91.87	92.84	4.00
5	74.37	73.36	74.24	73.99	5.02
6	62.81	61.28	62.36	62.15	5.97
7	54.79	53.81	53.88	54.16	6.85
8	47.45	47.74	45.82	47.00	7.90
9	42.30	42.48	41.62	42.13	8.81
10	38.29	37.32	38.63	38.08	9.75
11	35.23	34.02	34.03	34.43	10.78
12	32.54	32.25	32.74	32.51	11.42

Tabelle 3: Laufzeitmessungen für verschiedene Thread-Anzahlen (512 Interlines, 900 Iterationen)

2.3 Visualisierung der Ergebnisse

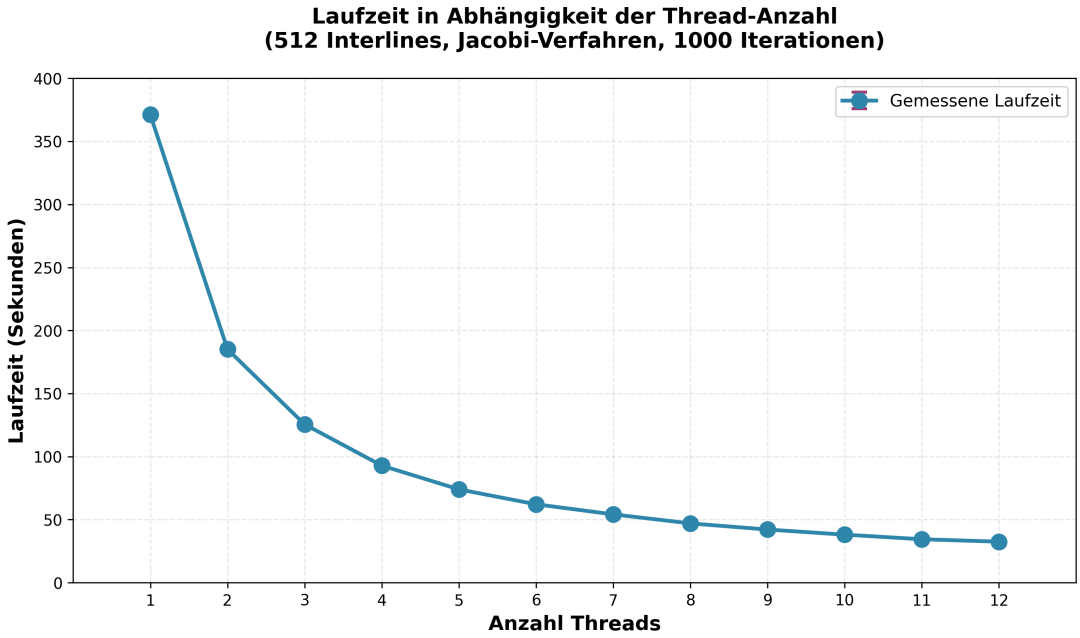


Abbildung 1: Laufzeit in Abhängigkeit der Thread-Anzahl

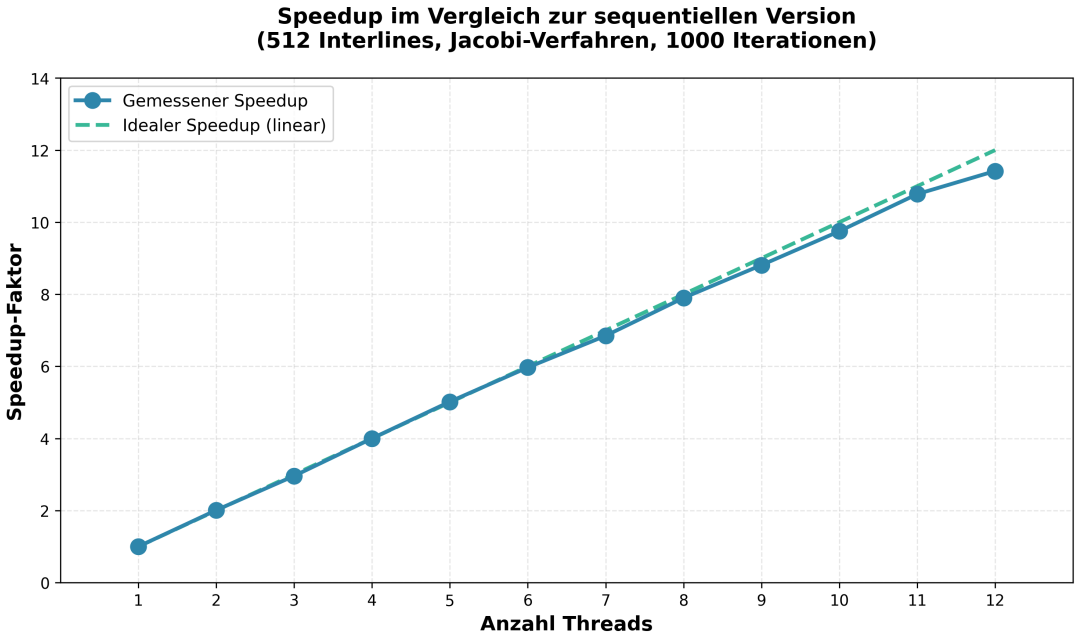


Abbildung 2: Speedup im Vergleich zur Version mit 1 Thread

2.4 Interpretation der Ergebnisse

Nach Auswertung der Ergebnisse der Tabelle und der Diagramme ist zu erkennen, dass mit zunehmender Anzahl der Threads, die benötigte Rechenzeit linear abnimmt. Allerdings ist der Speedup nicht vollständig linear, da ein gewisser Anzeil des Programmes nicht parallelisierbar ist und nicht zwischen den Threads aufgeteilt werden kann. Es zeigt sich, dass das Nutzen von Threads zu einer erheblichen Verkürzung der benötigten Rechenzeit führt, wenn es Rechenoperationen, wie in unserem Fall beim Jakobi Verfahren gibt, welche unabhängig berechnetr werden können.