1 Speedup of partdiff-par with MPI

1.1 Weak Scaling

Config	Time
1-1-100	37,1077
2-1-141	36,3078
4-2-200	36,9010
8-4-282	43,0961
16-4-400	37,6982
24-4-490	38,1505
32-5-566	41,8157
64-8-800	37,1526

(a) Jacobi

Config	Time
1-1-100	6,6506
2-1-141	12,2550
4-2-200	14,0042
8-4-282	14,7659
16-4-400	15,1066
24-4-490	14,2191
32-5-566	13,8884
64-8-800	15,4313

(b) Gauss-Seidel

Tabelle 1: Vergleich des Weak Scalings vom Jacobi- und dem Gauss-Seidel-Algorithmus

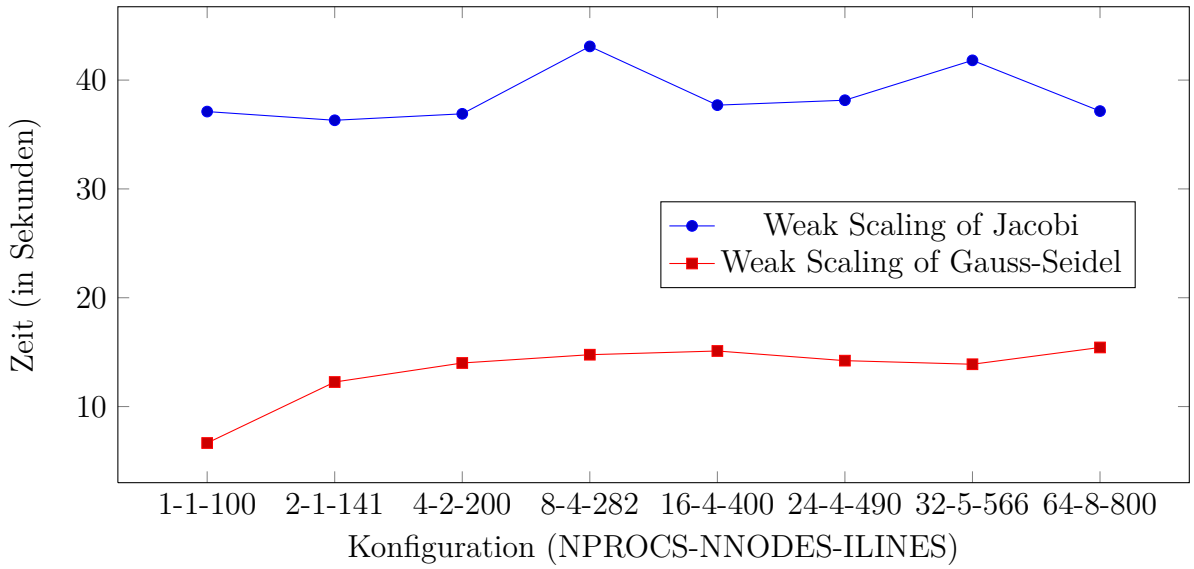

Abbildung 1: Vergleich des Weak Scalings vom Jacobi- und dem Gauss-Seidel-Algorithmus (mit $f(x, y) = 2\pi^2 \cdot sin(\pi x) \cdot sin(\pi y)$, 1500 Iterationen)

Die Wahl der Interlines in Bezug auf die Prozessorzahl erfolgt so, dass die Anzahl der zu berechnenden Matrixelemente pro Prozessor konstant bleiben. Da mit Erhöhung der Interlines die Anzahl der zu berechnenden Matrixelemente quadratisch steigt, ist diese Abhängigkeit nicht linear, d.h. bei Vervierfachung der Prozessorzahl wird die Anzahl der Interlines nur verdoppelt.
Man erkennt an den Plots, dass sowohl für Jacobi als auch Gauss-Seidel das Programm gut skaliert, da die Laufzeiten sich mit steigender Prozessorzahl kaum verlängern.

1.2 Strong Scaling

Config	Time	Speedup
12-1-960	187,8698	1,0000
24-2-960	94,1511	1,9954
48-4-960	47,4776	3,9570
96-8-960	24,0691	7,8054
120-10-960	19,1146	9,8286
180-10-960	16,7746	11,1997
240-10-960	13,7518	13,6615

(a) Jacobi

Config	Time	Speedup
12-1-960	68,5427	1,0000
24-2-960	34,8779	1,9652
48-4-960	18,6613	3,6730
96-8-960	9,9336	6,9000
120-10-960	8,5087	8,0556
180-10-960	6,7498	10,1548
240-10-960	5,4249	12,6348

(b) Gauss-Seidel

Tabelle 2: Vergleich des Strong Scalings vom Jacobi- und dem Gauss-Seidel-Algorithmus

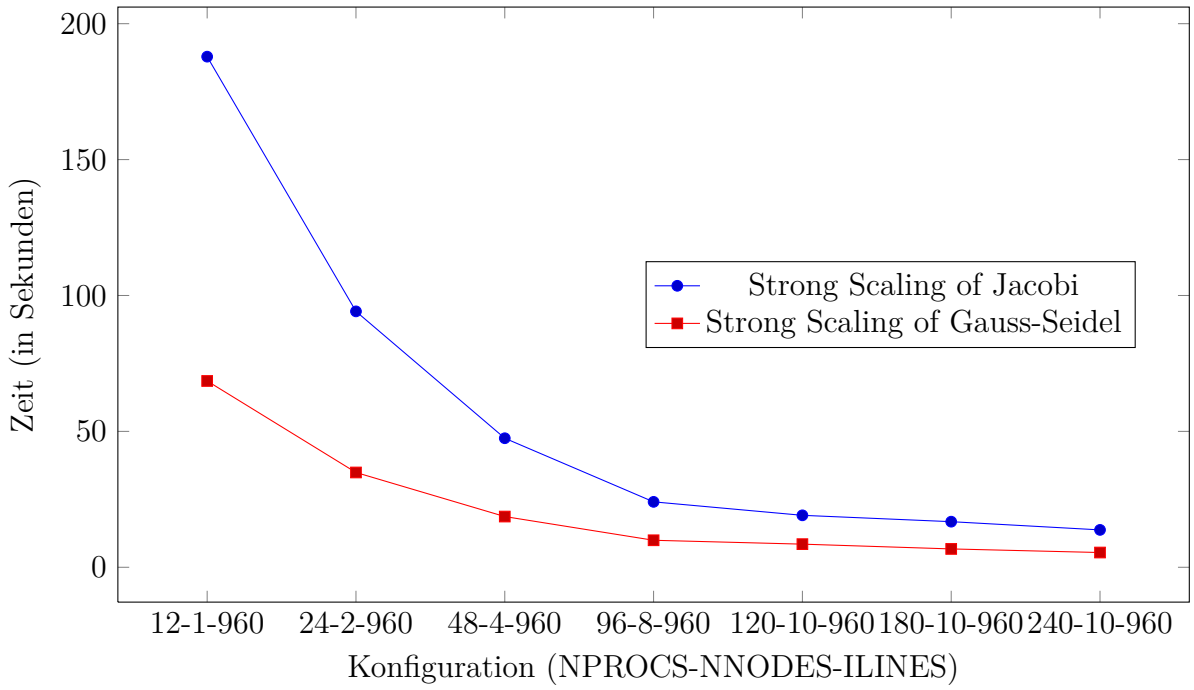

Abbildung 2: Vergleich des Strong Scalings vom Jacobi- und dem Gauss-Seidel-Algorithmus (mit $f(x, y) = 2\pi^2 \cdot sin(\pi x) \cdot sin(\pi y)$, 1000 Iterationen)

Bei beiden Algorithmen fällt mit steigender Prozesszahl die relative Steigerung des Speedups immer geringer aus.Dies wird wahrscheinlich durch den erhöhten Kommunikationsaufwand bedingt.
Der Speedup ist beim Jacobi-Algorithmus generell höher als beim Gauss-Seidel-Algorithmus. Dies könnte daran liegen, dass beim Gauss-Seidel-Algorithmus die Sinuswerte nur einmal gecacht werden und beim Jacobi-Algorithmus dies immer wieder neu geschieht.

1.3 Kommunikation und Teilnutzung der Knoten

Config	Time
10-1-200	19,6636
10-2-200	16,2858
10-3-200	16,9745
10-4-200	17,7834
10-6-200	20,2841
10-8-200	19,8398
10-10-200	20,6386

(a) Jacobi

Config	Time
10-1-200	33,8843
10-2-200	44,2525
10-3-200	41,6688
10-4-200	38,2670
10-6-200	40,0782
10-8-200	42,5476
10-10-200	36,9319

(b) Gauss-Seidel

Tabelle 3: Vergleich der Kommunikation und Teilnutzung der Knoten vom Jacobi- und dem Gauss-Seidel-Algorithmus

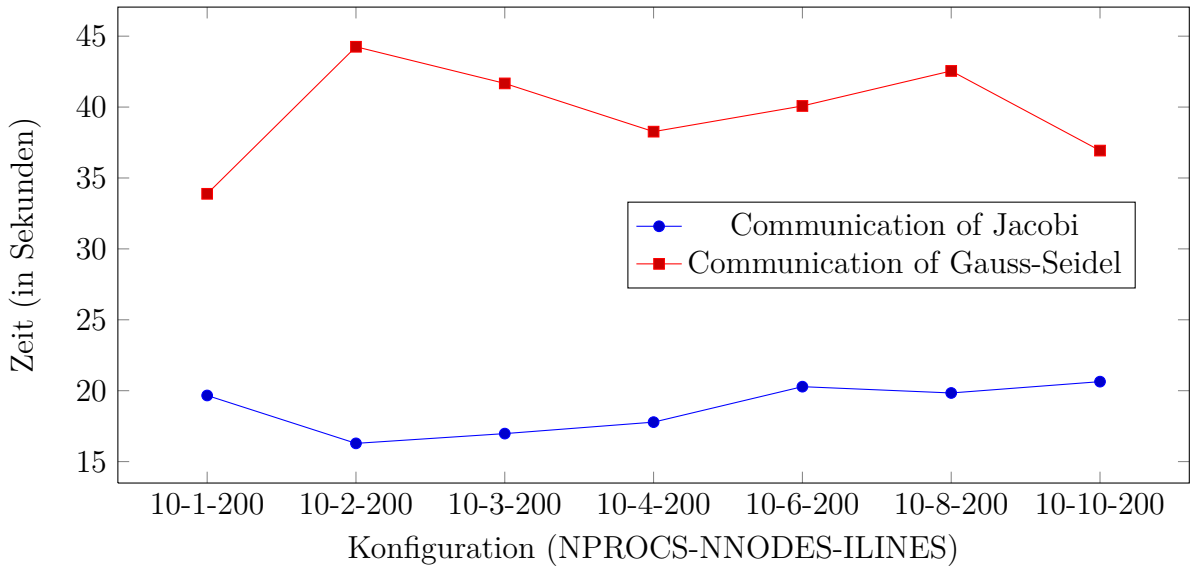

Abbildung 3: Vergleich der Kommunikation und Teilnutzung der Knoten vom Jacobi- und dem Gauss-Seidel-Algorithmus
(mit $f(x, y) = 0$, Genauigkeit: $3{,}3504 \cdot 10^{-5}$)

Beim Gauss-Seidel-Algorithmus fällt eindeutig auf, dass dieser schneller ist, wenn er nur auf einem Knoten mit mehreren Prozessen läuft. Beim Jacobi-Algorithmus hingegen spielt die Netzwerklatenz zwischen den verschiedenen Knoten nicht so eine große Rolle. Dieser wird sogar schneller, wenn die zehn Prozesse statt auf einem Knoten auf zwei Knoten laufen. Wahrscheinlich wird dies dadurch bedingt, dass durch die Halbierung des Matrixberechnungsaufwandes bei Verteilung der zehn Prozesse auf zwei Knoten die Berechnung so viel schneller abläuft, dass die Netzwerklatenz nicht mehr so stark ins Gewicht fehlt. Generell ist die Laufzeit für die mehrknotigen Konfigurationen leicht schwankend.