Lösungen zu Aufgabenblatt 10

Merlin Koglin, Timon Vosberg, Merlin Steuer

Abgabe: 16. Januar 2016

Inhaltsverzeichnis

Zusammenfassung

Es wurden die Leistungswerte und Laufzeiten der partdeff-par Implementierung getestet, welche von der Gruppe Koglin, Vosberg, Steuer zu Aufgabenblatt 9 abgegeben wurde. Es wurden außerdem zwei komplette Durchläufe zur Verifikation der Ergebnisse durchgeführt, daher sind in jedem Schaubild zwei Verläufe abgebildet.

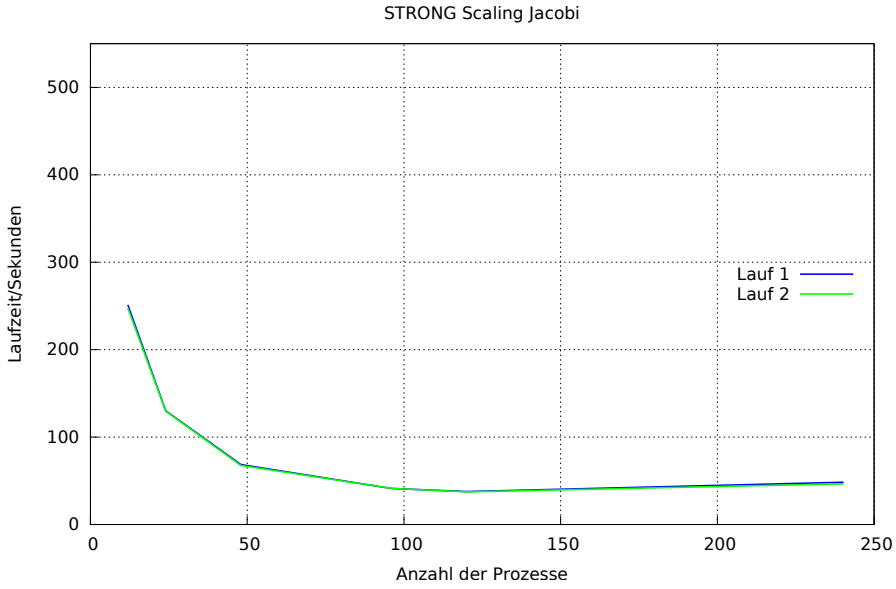
Eine erste allgemeine Betrachtung zeigt, dass unsere Jacobi-Implementierung deutlich schneller ist als die Gauß-Seidel Implementierung, welche etwas doppelt so lange Laufzeiten hat.

1 Strong Scaling

1.1 Jacobi

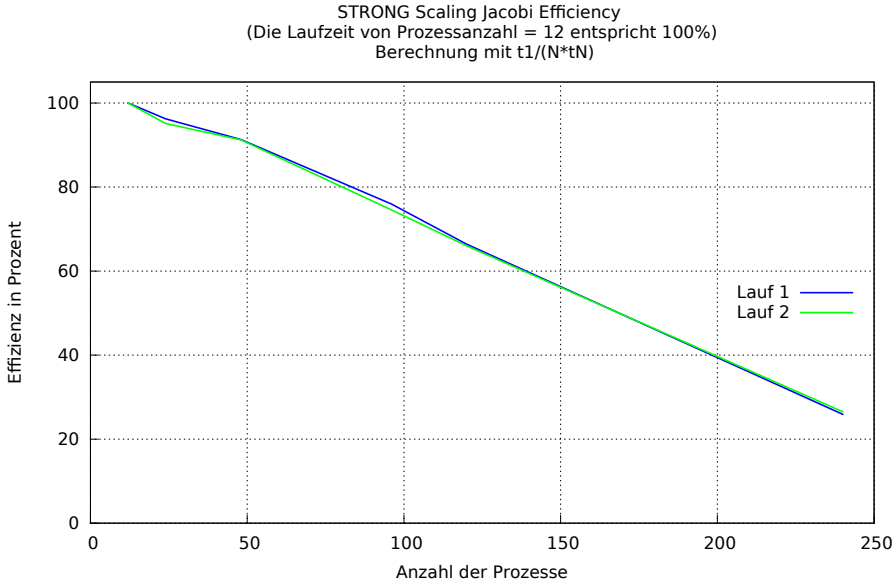
Anzahl der Prozesse	Laufzeit in s	Laufzeit in s (2. Durchlauf)
12	250.4	246.9
24	130.1	129.8
48	68.6	67.6
96	41.2	41.4
120	37.7	37.4
240	48.3	46.6

Wobei die Anzahl der genutzten Knoten stets $\frac{AnzahlProzesse}{12}$ entspricht. Auch bei 240 Prozessen wurden jedoch 10 Knoten verwendet. Weiterhin wurden in jedem Lauf 960 Interlines verwendet.



Grundsätzlich ist hier bei steigender Prozesszahl ein schöner antiproportionaler Verlauf zu betrachten. Auch ein Blick in die Tabelle zeigt, dass unser Programm von 12-48 Prozessen gut skaliert, dies jedoch bei weiter steigender Prozesszahl abnimmt. Bei der Verdoppelung der Prozesse von 120 auf 240 gibt es gar eine Steigerung der Laufzeit um ca. 25%. Hier steht die doppelt so große Prozesszahl der deutlich vermehrten Kommunikation entgegen, dies scheint auch den Vorteil des Hyperthreading zu verringern.

Bestätigt wird diese Beobachtung durch die Betrachtung der Effizienz:

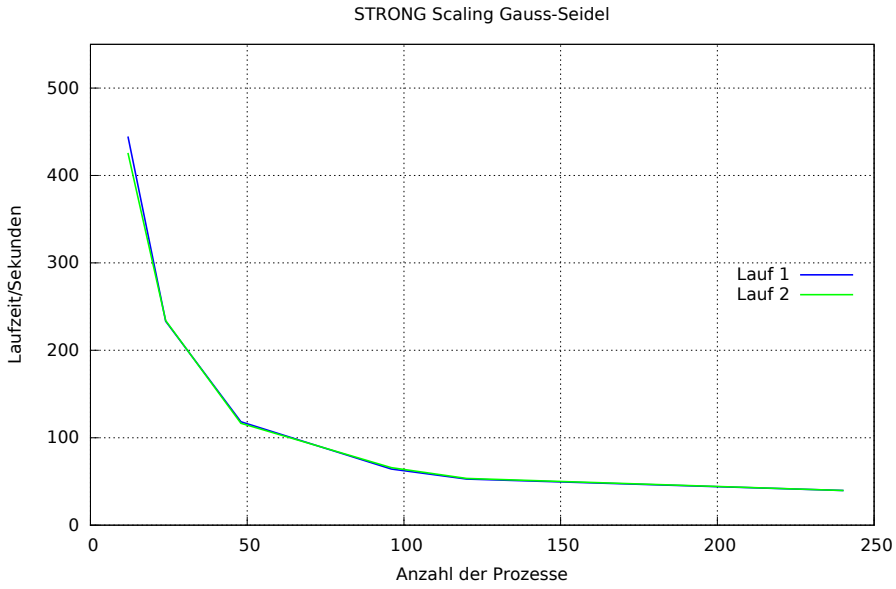


Die Effizienz der Jacobi-Implementation sinkt nahezu linear mit steigender Prozesszahl.

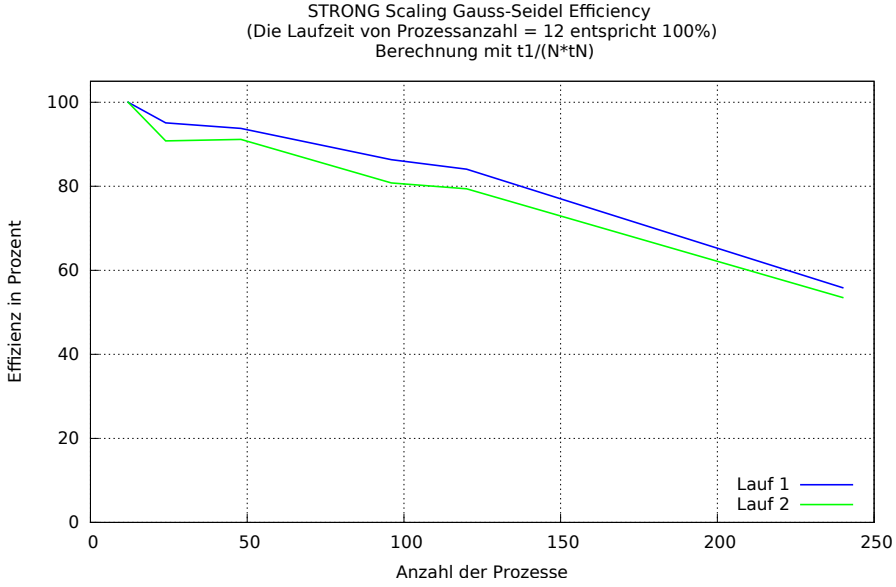
1.2 Gauß-Seidel

Anzahl der Prozesse	Laufzeit in s	Laufzeit in s (2. Durchlauf)
12	443.7	424.9
24	233.3	234.0
48	118.3	116.5
96	64.2	65.8
120	52.8	53.5
240	39.7	39.7

Wobei die Anzahl der genutzten Knoten stets $\frac{AnzahlProzesse}{12}$ entspricht. Auch bei 240 Prozessen wurden jedoch 10 Knoten verwendet. Weiterhin wurden in jedem Lauf 960 Interlines verwendet.



Anders als die Implementation des Jacobi-Verfahrens skaliert das Gauß-Seidel-Verfahren auch beim letzten Verdoppelungsschritt von 120 auf 240 Prozesse. Allgemein ist hier ein besseres Skalierungsverhalten bis in größere Prozesszahlen zu beobachten.

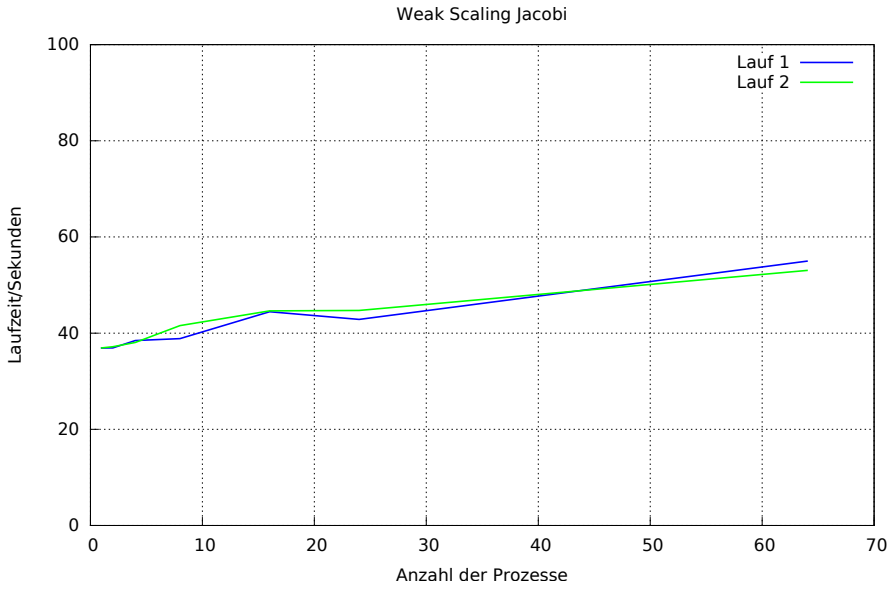


Auch hier betrachten wir durch die Faktoren wie nicht genutzte Wartezeiten in der Pipeline oder Latenzen in der Kommunikation einen nahezu linearen Abfall der Effizienz. Jedoch ist dieser nicht so stark wie beim Jacobi-Verfahren, entsprechend ist das Gauß-Seidel-Verfahren im *strong scaling* effizienter implementiert.

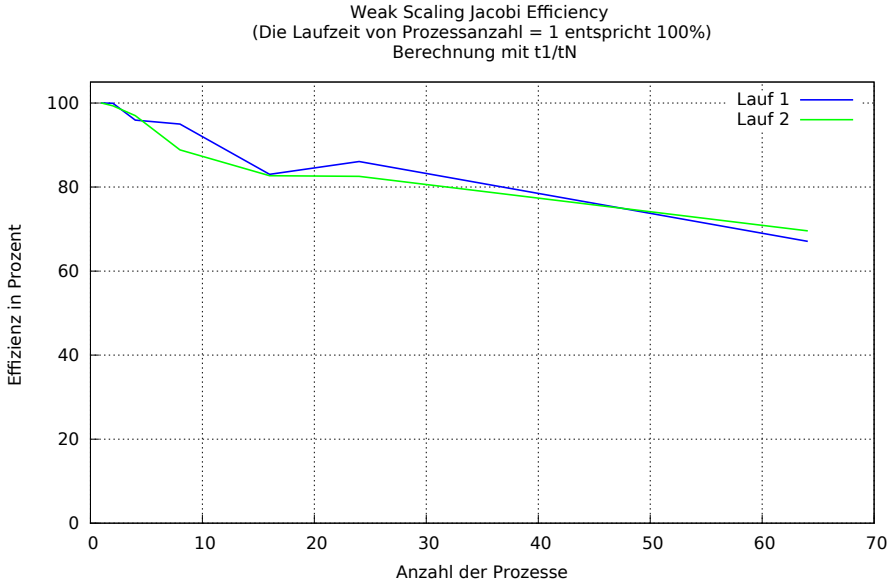
2 Weak Scaling

2.1 Jacobi

Prozesse	Knoten	Interlines	Laufzeit in s	Laufzeit in s (2. Durchlauf)
1	1	100	36.9	36.9
2	1	141	36.9	37.2
4	2	200	38.5	38.1
8	4	282	38.9	41.6
16	4	400	44.4	44.6
24	4	490	42.9	44.7
64	8	800	55.0	53.0



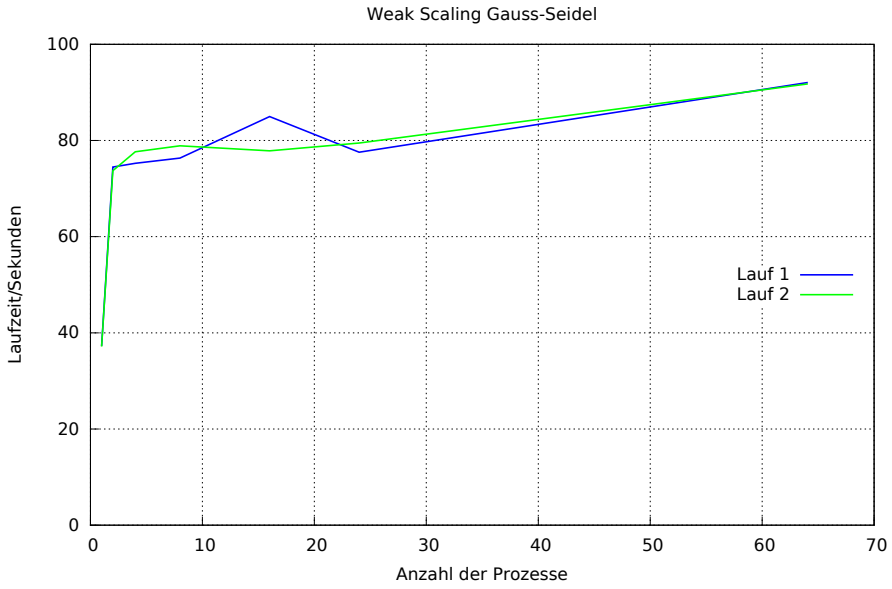
Die Implementation des Jacobi-Verfahrens schlägt sich im Weak-Scaling gut. Bei steigender Prozesszahl und proportionaler Vergrößerung des zu berechnenden Problems steigt die Berechnungszeit nur leicht nahezu linear an (ca. Faktor 1.5 zwischen kleinstem und größtem Problem).



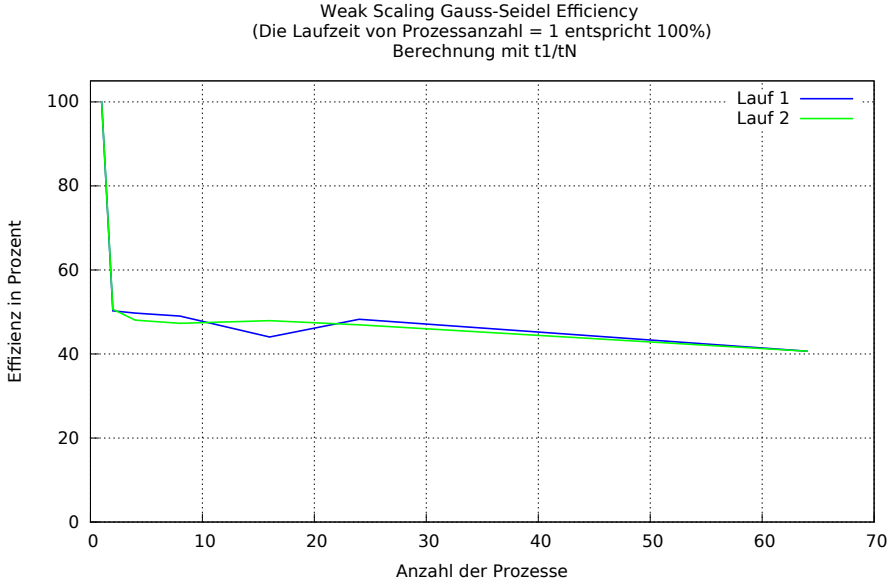
Wie auch schon in obigem Schaubild ersichtlich ist die Implementierung des *weak scaling* im Jacobi-Verfahren effizienter als im *strong scaling*. Zwar verliert man in kleineren Prozesszahlen etwas Effizienz, später jedoch sinkt diese deutlich flacher ab als beim *strong scaling*.

2.2 Gauß-Seidel

Prozesse	Knoten	Interlines	Laufzeit in s	Laufzeit in s (2. Durchlauf)
1	1	100	37.4	37.3
2	1	141	74.5	73.7
4	2	200	75.3	77.6
8	4	282	76.35	78.9
16	4	400	85.0	77.9
24	4	490	77.6	79.5
64	8	800	92.0	91.8



Anders als im Jacobi-Verfahren skaliert die Implementation des Gauß-Seidel-Verfahrens nicht sonderlich gut. Bereits im ersten Schritt (verdoppelung der Problemgröße bei zwei Prozessen auf einem Knoten) beobachten wir eine Verdoppelung der Laufzeit. Dies wird dadurch begründet sein, dass bei einem Prozess keinerlei Kommunikation statt findet, wobei bei zwei Prozessen die beiden Prozesse jeweils aufeinander warten müssen, bis sie weiter berechnen können, hierdurch entstehen natürlich Verzögerungen. Im weiteren Verlauf ist die Implementierung jedoch relativ stabil, lediglich bei sehr großen Problemgrößen bzw. Prozesszahlen steigt die Laufzeit wieder an.



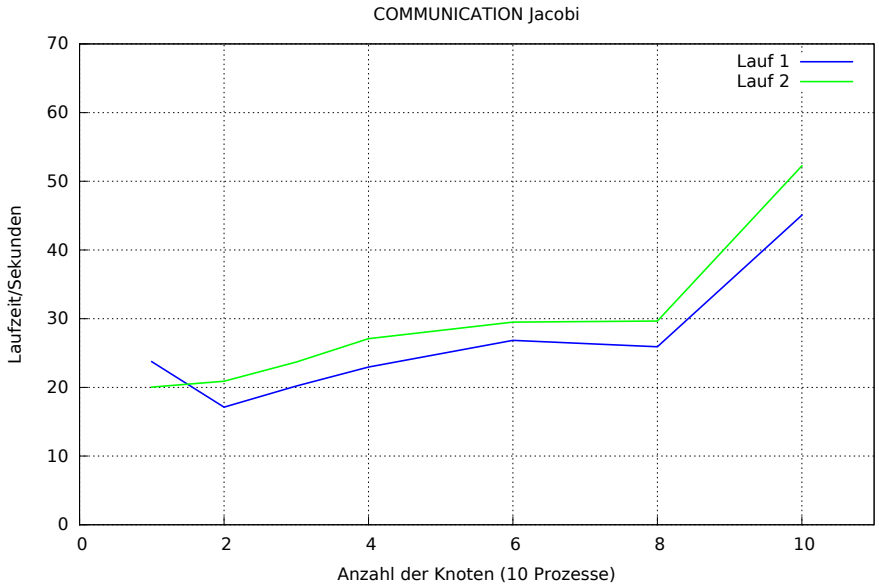
Obige Ergebnisse spiegeln sich auch schön hier wieder, ab 2 Prozessen ist die Effizienz nahezu konstant bei 40-50%.

3 Kommunikation und Teilnutzung der Knoten

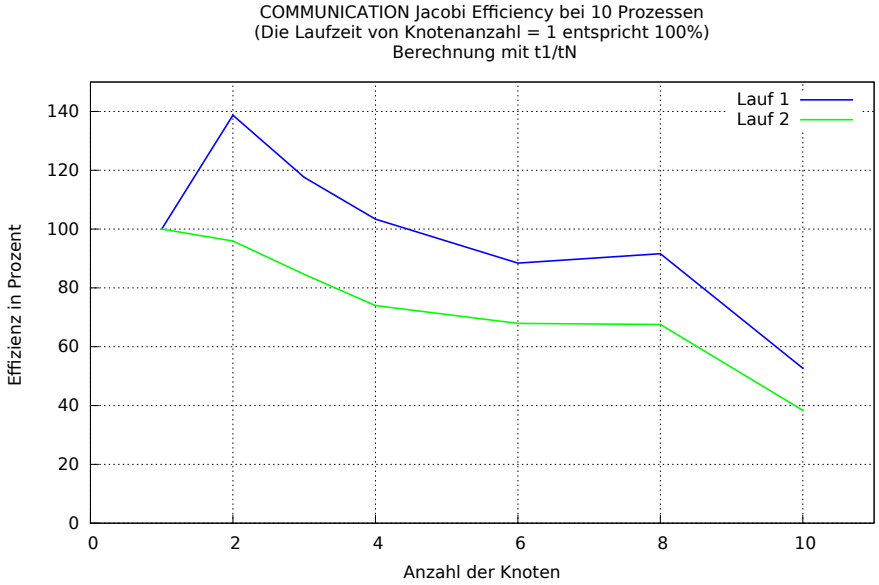
3.1 Jacobi

Knoten (je 10 Proz.)	Laufzeit in s	Laufzeit in s (2. Durchlauf)
1	23.7	20.0
2	17.1	20.9
3	20.2	23.7
4	23.0	27.1
6	26.8	29.5
8	25.9	29.7
10	45.1	52.3

Mit je 200 Interlines.



Da wir in dieser Problemstellung eine fixe Anzahl an Prozessen haben, nämlich zehn, und die Anzahl der verwendeten Knoten stets erhöhen steigt mit jeder Erhöhung der Knotenzahl die Zeit, welche für die Kommunikation benötigt wird drastisch an, denn Kommunikation zwischen zwei Rechnern ist deutlich langsamer als zwischen zwei Prozessen auf einem Knoten. Zwar steigt die Laufzeit nicht proportional zur Anzahl der Prozesse, jedoch ist ein nahezu lineares Wachstum zu betrachten.

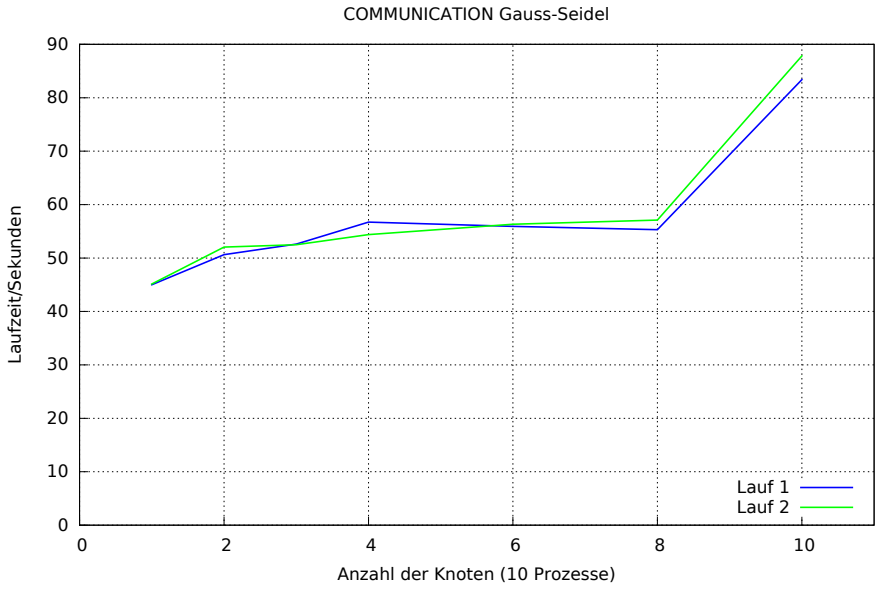


Besser als in der Laufzeit-Darstellung ist hier die starke Abweichung des ersten Laufs zu betrachten, welche zunächst sogar Effizienz von über 100% erreicht. Oben ist dies ebenfalls zu beobachten, denn die Laufzeit sinkt unter die der Durchführung mit 1 Prozess. Wirklich erklären können wir uns diesen Effekt jedoch nicht, zumal der zweite Durchlauf sich wie erwartet mit stets sinkender Effizienz verhält.

3.2 Gauß-Seidel

Knoten (je 10 Proz.)	Laufzeit in s	Laufzeit in s (2. Durchlauf)
1	45.0	45.1
2	50.6	52.4
3	52.6	52.5
4	56.7	54.4
6	55.9	56.3
8	55.3	57.1
10	83.4	87.8

Mit je 200 Interlines.



Das Gauß-Seidel Verfahren scheint hier jedoch von der Implementierung zu profitieren. Die Laufzeit steigt im Verlauf nicht allzu stark an, lediglich wenn nur 1 Prozess auf jedem Knoten läuft betrachten wir noch mal einen starken Sprung in Laufzeit und Effizienz nach oben bzw. nach unten. Womöglich liegt dies daran, dass die Kommunikations-Latenz gegenüber den Verzögerungen in der Pipeline nicht mehr so stark ins Gewicht fällt.