Übungsblatt 10

Wir führten unsere Messungen mit unseren Implementationen von Gauß-Seidel und Jakobi durch. Bei Jakobi verwendeten wir die hybride Variante. Dabei haben wir leider vergessen den Parameter „number“ , der für die zu verwendende Threadanzahl von OMP zuständig ist richtig zu übergeben. Dies führt dazu, dass OMP bestimmt wie viele Threads verwendet werden (Standardanzahl), weshalb unsere Jakobi Messungen nicht sehr aussagekräftig sind. Leider ist uns dieser Fehler sehr spät aufgefallen, weshalb wir keine Chance mehr hatten die Skripte noch einmal mit der korrigierten Version durchlaufen zu lassen.

Anmerkung:
Bei unserem Jakobi-Verfahren sollen die Zahlen in der Matrix für ≥ 8 Prozesse nicht mehr gestimmt haben, wir haben uns aber entschieden trotzdem unsere Variante zu verwenden. (Wir haben diese selbst noch einmal getestet aber bei uns kamen korrekte Ergebnisse raus).

Im Folgenden wird die Standardabweichung durch:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}}$$

mit n = 3 berechnet.

(Wir haben 3 Mal gemessen (n = 3) und der Erwartungswert (μ) ist das arithmetische Mittel der 3 Messungen).

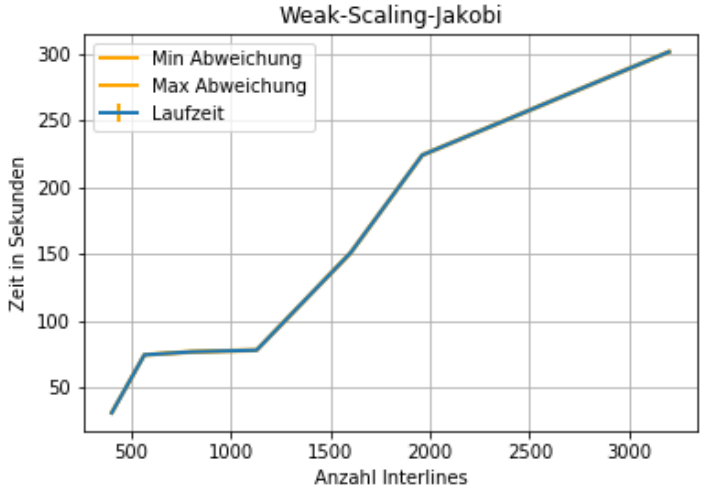
Weak Scaling

Frage: *Wie erklären Sie sich die Wahl der Interlines in Bezug auf die Prozesszahl?*

Antwort: Wir verdoppeln bei jedem Aufruf die Prozesszahl, daher muss sich für das Weak-Scaling auch die Anzahl der zu bearbeitenden Einträge in der Matrix verdoppeln, diese ist ca. quadratisch die Anzahl der Interlines, weshalb man diese entsprechend wählen muss.

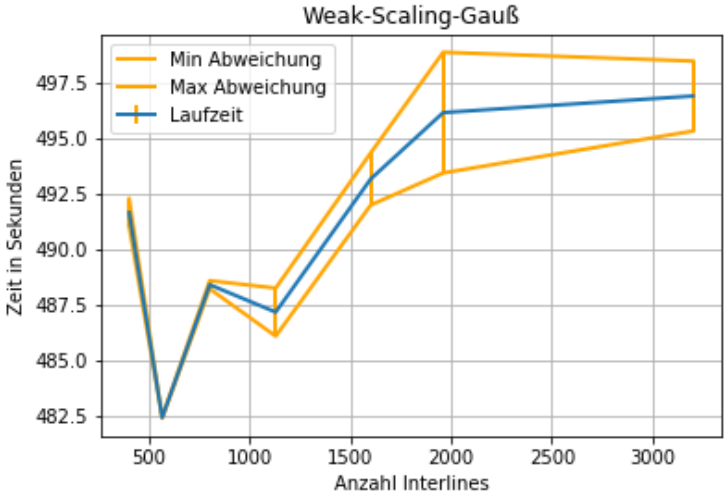
Wir haben uns beim Weak Scaling dafür entschieden, die benötigte Zeit anstelle des Speedups anzugeben, da das Ergebnis im Idealfall (wie in etwas bei unserem Gauß-Seidel) immer die von gleicher Dauer ist oder der Speedup sich konstant verhält. Dies ermöglichte uns die Varianz schöner darstellen zu können.

Desweiteren haben wir auf der x-Achse nur die Anzahl der Prozesse angegeben, aber die Größe der Matrix wächst analog mit. D.h. wenn sich die Prozesszahl verdoppelt, verdoppelt sich implizit auch die Anzahl der Einträge der Matrix. [Wir wollten keine Tupel (#Prozesszahl, #Interlines) auf der x-Achse angeben und entschieden uns daher für diese Variante).



Beim Jacobi-Verfahren variierten unsere Werte zu stark um die relativ kleinen Standardabweichung sehen zu können.

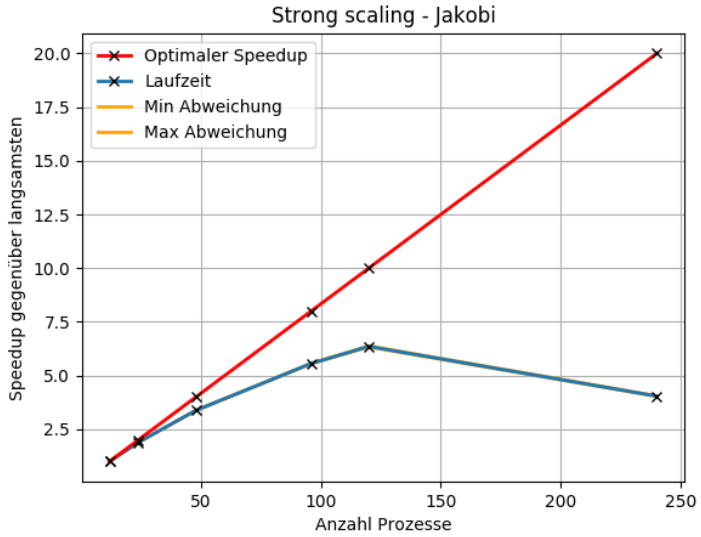
In diesen Messungen, ist neben einer Konstanten Verlangsamung, durch MPI, vermutlich noch unsere OMP Implementierung ein signifikanter Faktor. Mit fast jeder neuen Messung wird die Core-Anzahl pro CPU erhöht. Außerdem belegt OMP die Default-Thread-Menge an Cores. Nun können sich die Prozesse und Threads von OMP gegenseitig ausbremsen, was in dieser Grafik zu einer Verlangsamung führt.
Es ist nicht auszuschließen, dass bei einer derart großen Verlangsamung auch andere Implementationsfehler eine Rolle spielen.



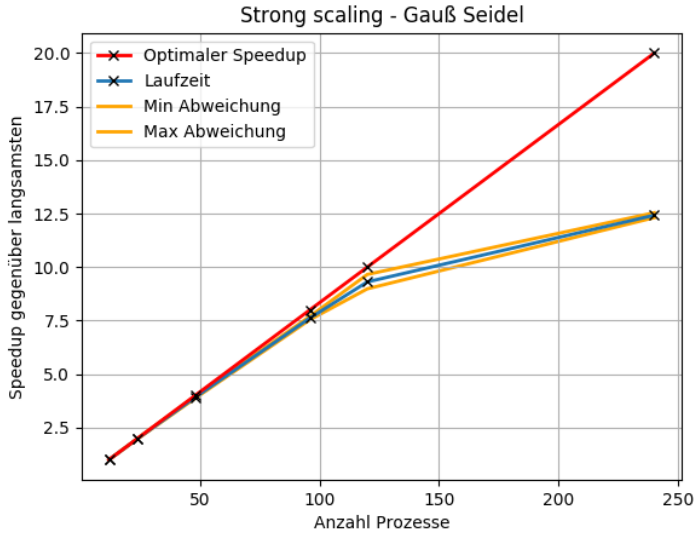
Beim Gauß-Seidel-Verfahren haben wir nur einen kleinen Zeitausschnitt in dem sich unsere Werte bewegen dargestellt, damit die Varianz besser ersichtlich ist. Es fällt auf, dass die Zeiten hier so skalieren, dass die Effizienz bei einer Anpassung der Problemgröße an die Prozesszahl nahezu konstant bleibt. Wobei für große Matrizen (und dementsprechend mehr Prozesse) ein leichter Anstieg zu beobachten ist, den man auf mehr MPI-Kommunikation zurückführen kann.
Es ist uns unbekannt, warum die Messung mit 400 Interlines konstant langsamer ist.

Strong Scaling

Beim Strong Scaling verwendeten wir im Folgenden unseren langsamsten parallelen Aufruf (dieser verwendet die wenigsten Prozesse) als Vergleichswert um den Speedup zu berechnen, anstelle des seriellen Aufrufs mit nur einem Prozess.



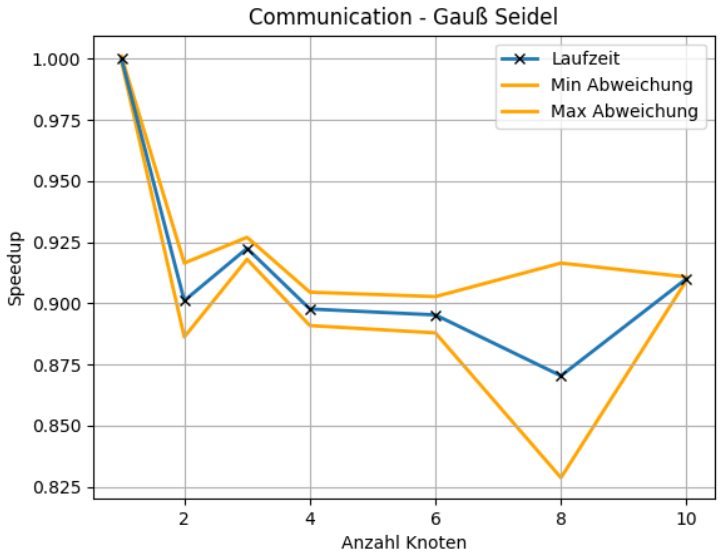
Hier ist eine Steigung der Speedup-Kurve gut sichtbar. Allerdings bricht diese bei der letzten Messung nach unten ab. Dies lässt sich dadurch begründen, dass die Prozesse zu wenig zu tun haben (zu wenig Zeilen pro Prozess) und mehr Zeit mit warten auf MPI-Syncs verbringen. Die Kurve für Jakobi steigt aber nicht so sehr wie der optimale Speedup, da wir bei unserer Implementierung ein Allreduce benutzen und daher alle Prozesse aufeinander warten müssen.



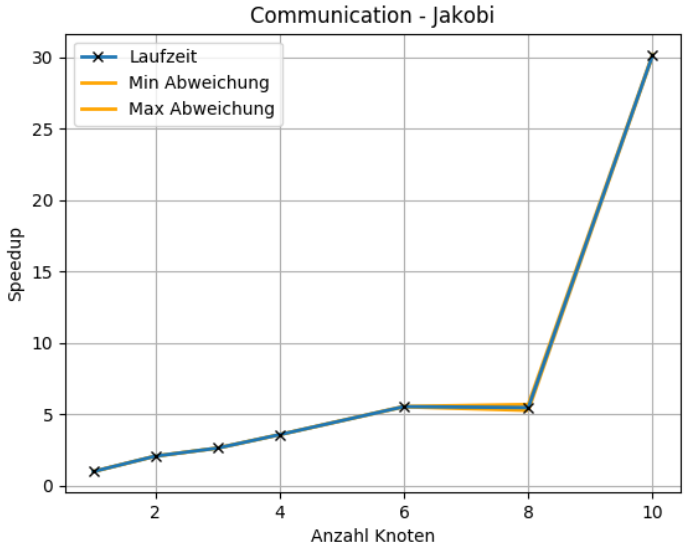
Bei unseren Gauß-Seidel-Messungen befindet sich die Kurve sehr viel näher am optimalen Speedup. Dies lässt sich dadurch erklären, dass hier die einzelnen Prozesse jeweils nur mit ihren Nachbarn kommunizieren und auf diese warten müssen. In der letzten Messung bricht auch hier die Kurve aus dem gleichem Grund wie beim Jakobi-Verfahren ab.

Kommunikation und Teilnutzung der Knoten

Bei diesen Messungen bleibt die Aufteilung der Matrix pro Prozess immer gleich. Es gibt insgesamt 10 Prozesse. Nun werden diese 10 Prozesse schrittweise auf 10 Knoten verteilt. Es ist zu erwarten, dass die nun nicht mehr die knoteninterne MPI-Kommunikation über das Netzwerk die Messungen verlangsamt.



Bei der Gauß-Seidel Messung ist der Messunterschied zwischen der ersten und zweiten Messung aufgrund des Wechsels von knoteninterner MPI-Kommunikation (1. Messung, da man hier nur einen Knoten nutzt) zu globaler Netzwerkkommunikation (2. Messung) ziemlich groß. Von da an wird das Netzwerk benutzt und der Speedup bleibt ungefähr gleich langsam.



Entgegen unserer Erwartung, steigt der Speedup an.
Dies liegt daran, dass wir eine hybride Version von OpenMP benutzt haben.
Diese versucht eine optimale Thread Anzahl zu belegen, welche im schlimmsten Fall 24 (Core-Anzahl Partition West) Threads benutzt. 10 Prozesse mit jeweils 24 Threads bringen keinen großen Speedup,

da sie sich gegenseitig blockieren (Pseudoparallelität). In der letzten Messung allerdings hat ein Prozess alle Cores für sich allein und kann 24 Threads echt parallel nutzen. Wir erhalten einen großen Speedup.