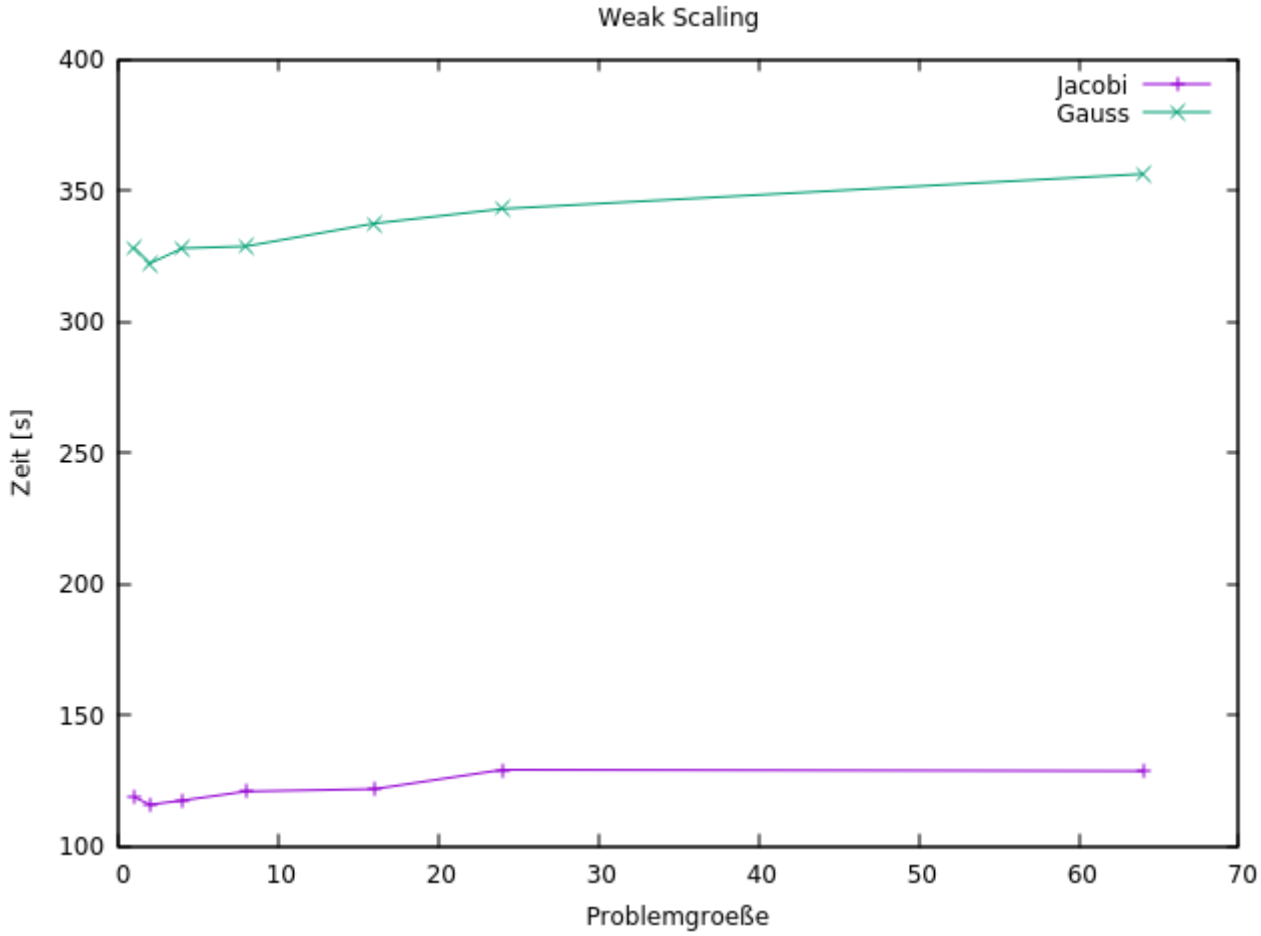
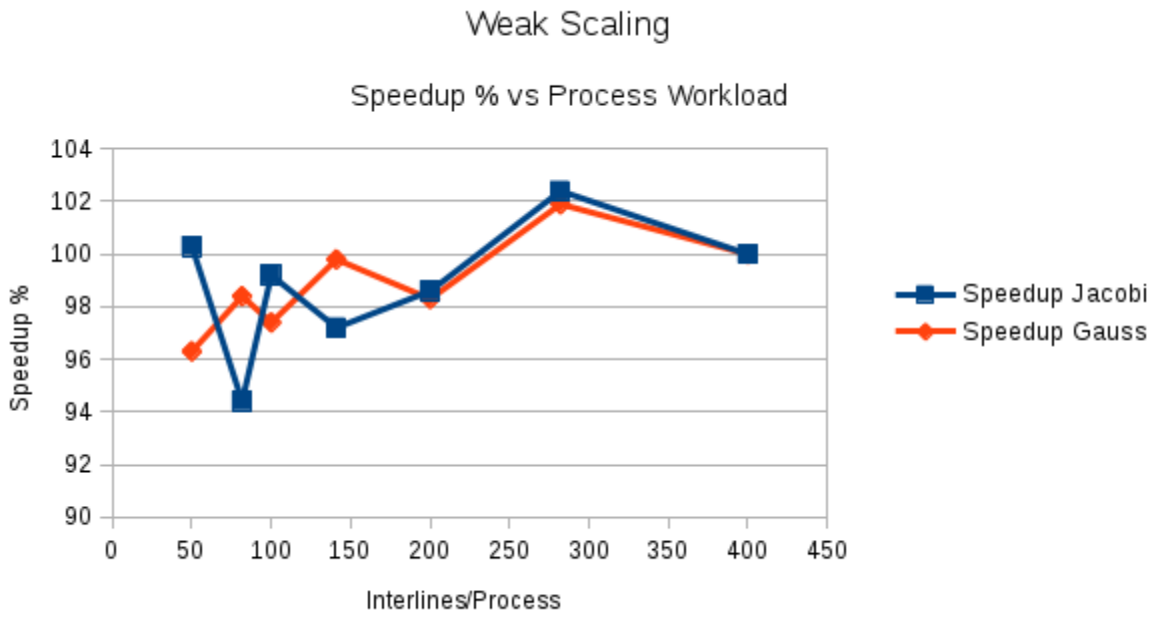
Weidmann, Steinke, Hansen

Leistungsevaluation paralleles Numerikprogramm

1. Weak Scaling

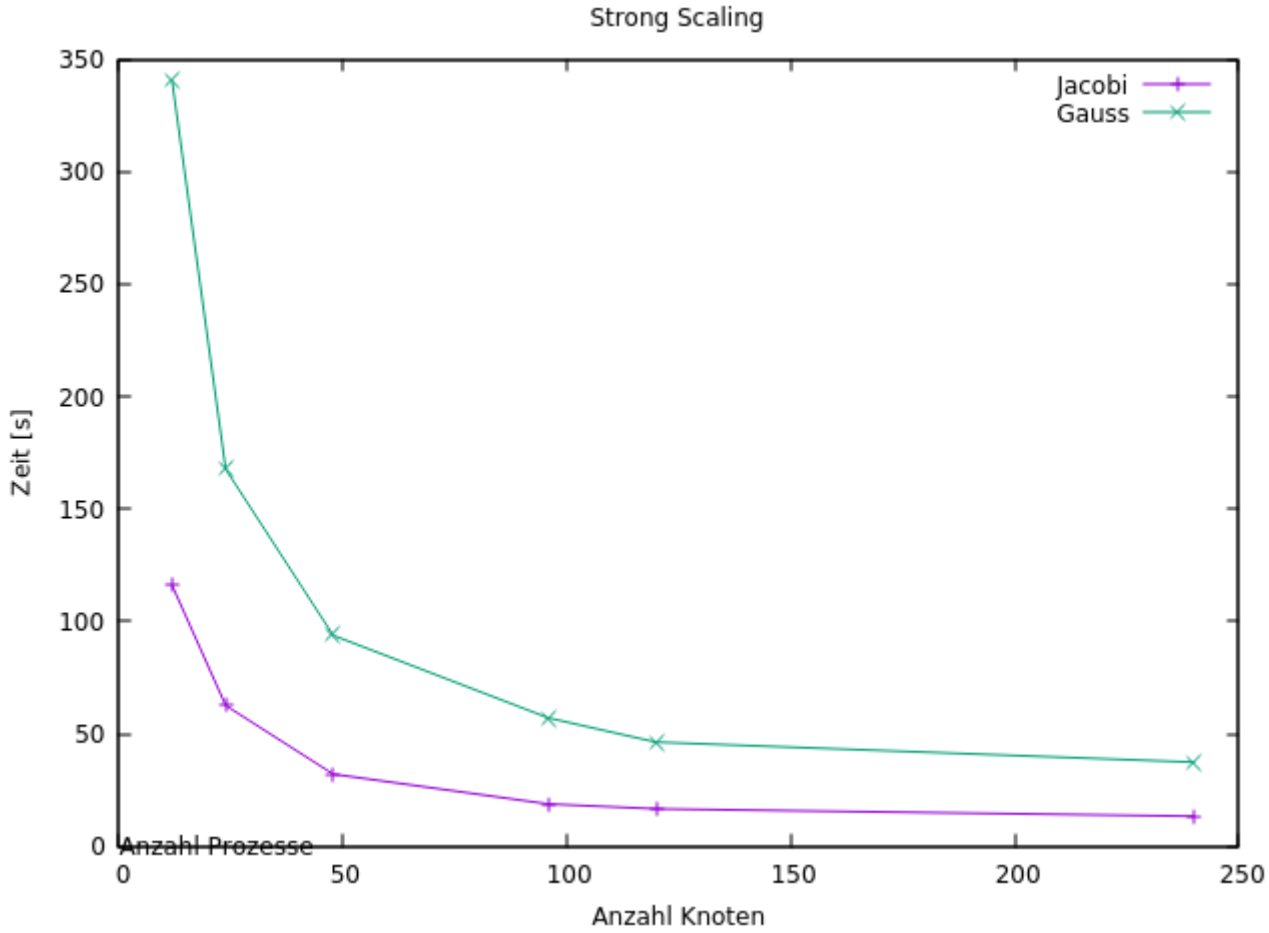


Hier ist die Berechnungszeit der Anzahl eingesetzter Prozesse (1 – 64) graphisch gegenübergestellt. Wie intuitiv erwartet bleibt die Ausführungszeit des Programms bei wachsender Problemgröße realtiv konstant, solange die Menge eingesetzter Prozesse proportional ansteigt. Wobei zu bedenken ist, dass mehr Prozesse auch den Kommunikationsanteil an der Rechenzeit je nach Algorithmus nach oben treiben – und das nicht unbedingt linear.

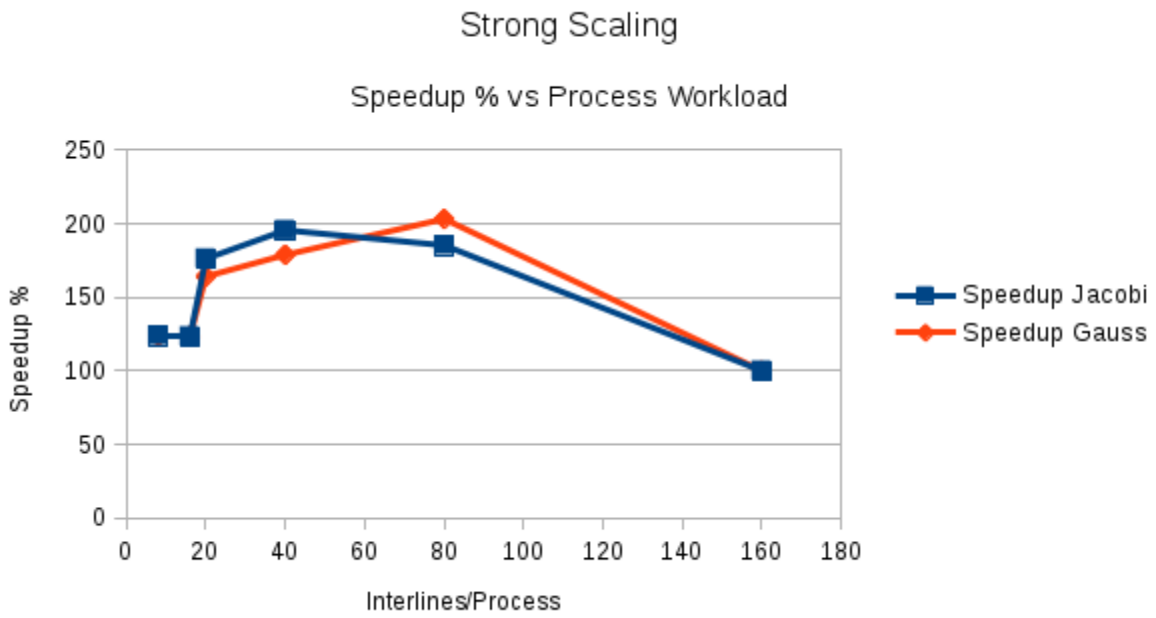


Der Speedup der Rechenzeit bleibt selbst bei sinkender Arbeitslast pro Prozess in etwa bei 100%. Bemerkenswert ist der ‘Ausreißer’ bei 81.6 Interlines pro Prozess im Jacobi-Verfahren. Ansonsten zeigt sich ein leichter Abwärtstrend der Zeitverbesserung mit steigender Prozesszahl, was auf den höheren Kommunikationsaufwand hinweist. Die Verwendung von Weak-Scaling ist somit eher für speicheraufwändige Berechnungen brauchbar, um ein Problem z.B. mit erhöhter Auflösung in etwa gleicher Zeit zu lösen.

Strong Scaling

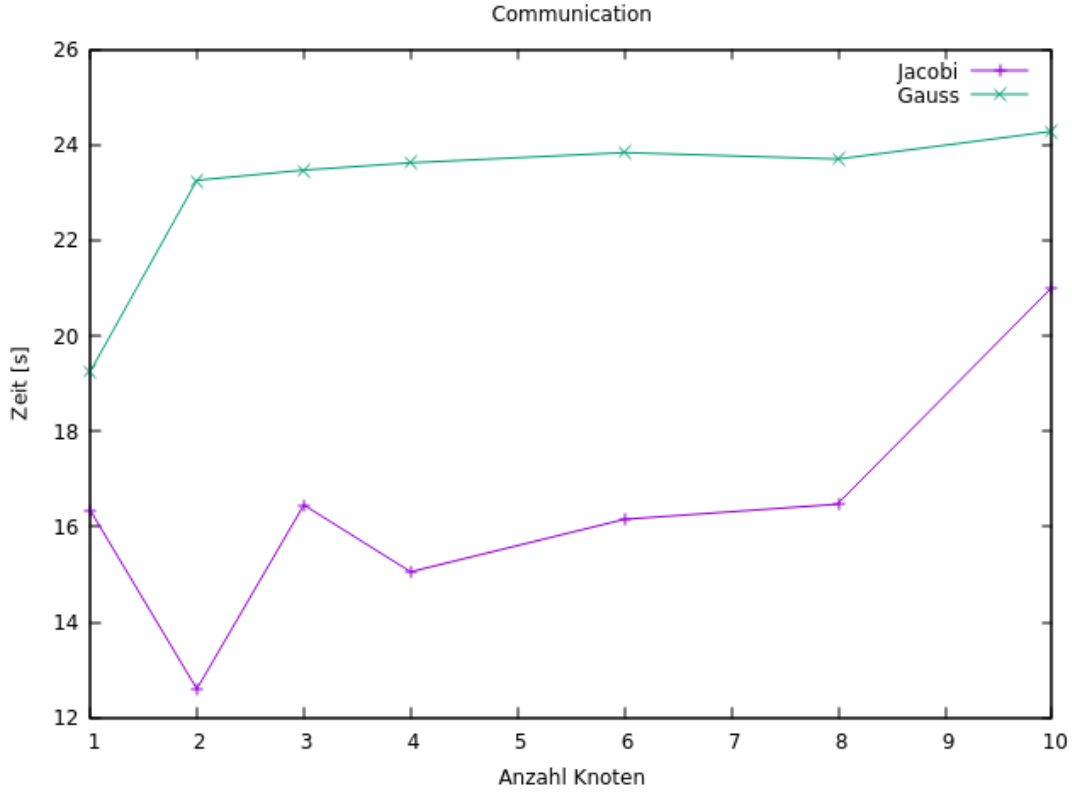


Hier zeigt sich die enorme verringerung der Rechenzeit eines Problems durch den Einsatz von mehr Prozessen (12 – 240). Die Kurven können weitestgehend als Bruchfunktion angenähert werden, was bedeutet, dass sie sich asymtotisch an Null annähern – mit quadratisch abnehmender Verbesserung. Somit müsste für ein Problem bestimmter größe ein Kompromiss zwischen Prozessbeschleunigung und Hardware-/Stromaufwendung gefunden werden.



Dieser Kompromiss dürfte durch die Betrachtung der prozentualen Beschleunigung über duch parallelisierung geminderte Prozesslast auffindbar sein. So ist hier ein scharfer Rückgang der Zeitverbesserung ab ca. 100 Prozessen sichtbar, unterschreitet der Speedup 100% beschäftigt man das Rechnersystem nur noch mit zusätzlichen MPI-Messages, ohne die Berechnung schneller voranzutreiben. Das kurze Plateau am anfang des Graphen zeigt kaum eine Verbesserung der Rechenzeit, obwohl die Prozessanzahl verdoppelt wurde – allerding bei gleicher Knotenanzahl, unter der annahme das CPUs mit ca. 10 Kernen verwendet werden bedeutet dies, dass man idealerweise mit einem Prozess pro Core Arbeiten sollte um effizient zu bleiben.

Kommunikation und Teilnutzung der Knoten



Die Pipeline des Gaussschen Algorithmus ist scheinbar resistenter gegenüber der Aufteilung auf viele Nodes, die Architektur der Pipeline erlaubt hier wohl eine höhere Asynchronität. Im Vergleich zeigt das Jacobi-Verfahren eine stetige Verschlimmerung mit wachsender inter-Node Kommunikation, vermutlich da am Ende einer Iteration auf alle Prozesse gewartet werden muss.