

Blatt 11

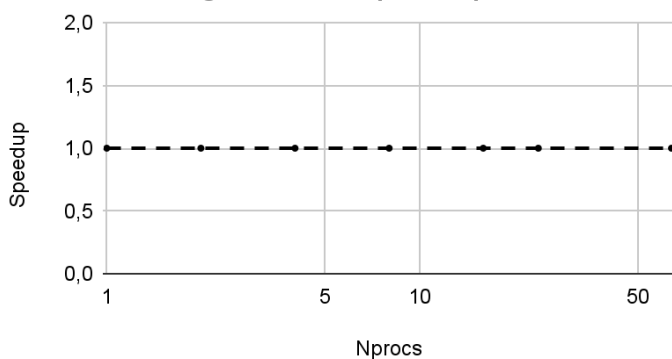
Abgabegruppe: SchusterPetersGurungPawelczyk

Aufgabe 1)

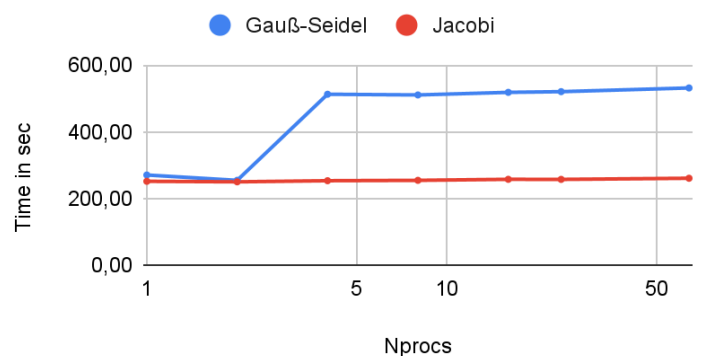
Leistungsevaluation des parallelen PDE-Lösers

| 1.1 Weak Scaling | | | | | |
|------------------|--------|--------|---------|---------|--|
| NPROCS | NNODES | ILINES | TIME GS | TIME JA | |
| 1 | 1 | 400 | 270,53 | 251,61 | |
| 2 | 1 | 564 | 254,50 | 250,02 | |
| 4 | 2 | 800 | 512,84 | 253,39 | |
| 8 | 4 | 1128 | 510,73 | 254,50 | |
| 16 | 4 | 1600 | 518,51 | 257,58 | |
| 24 | 4 | 1960 | 520,59 | 257,35 | |
| 64 | 8 | 3200 | 531,74 | 260,75 | |

Weak Scaling: Idealer Speedup



Weak Scaling GS Laufzeit



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

Die Wahl der Interlines bestimmt die Anzahl der zu berechnenden Zellen in der (Teil-)Matrix. Die Interlines wurden nun so gewählt, dass die Problemgröße proportional zu der Anzahl der Prozesse wächst. D.h. der Aufwand pro Prozess bleibt idealerweise gleich.

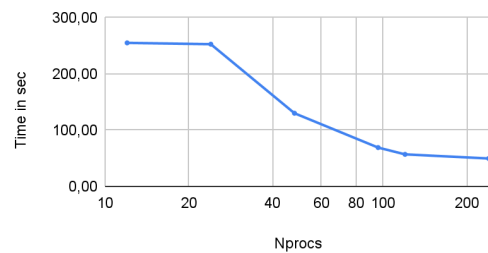
=> $(8 * \text{Interlines} + 9)^2 / \text{nprocs}$ bleibt ungefähr gleich für jede Konfiguration

Frage: Wie sähe die Speedupkurve bei idealem Weak Scaling aus?

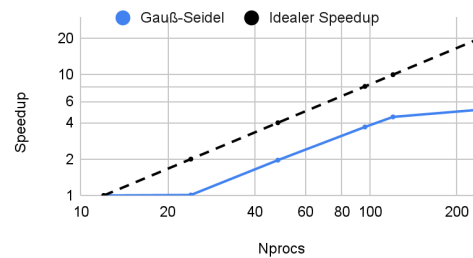
Im idealen Weak Scaling würde die Speedupkurve ein linearer Graph sein mit Steigung 0 und einem Speedup von 1, d.h. eine horizontale Linie.

| 1.2 Strong Scaling GS | | | | | | |
|-----------------------|--------|--------|--------|-------------|--------------|--|
| NPROCS | NNODES | ILINES | TIME | Speedup | Effizienz | |
| 12 | 1 | 1920 | 254,97 | 1 | 1 | |
| 24 | 2 | 1920 | 252,63 | 1,009273704 | 0,5046368518 | |
| 48 | 4 | 1920 | 130,03 | 1,960825947 | 0,4902064868 | |
| 96 | 8 | 1920 | 69,07 | 3,691328015 | 0,4614160019 | |
| 120 | 10 | 1920 | 57,02 | 4,471663104 | 0,4471663104 | |
| 240 | 10 | 1920 | 49,68 | 5,131906971 | 0,2565953486 | |

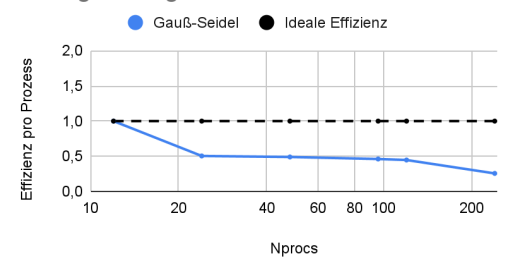
Strong Scaling GS Laufzeit



Strong Scaling GS Speedup

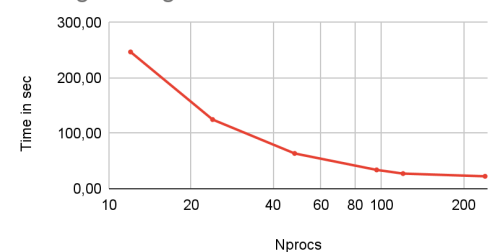


Strong Scaling GS Effizienz

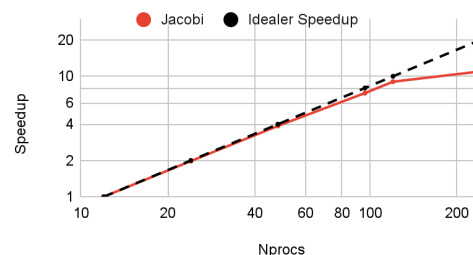


| 1.2 Strong Scaling JA | | | | | | |
|-----------------------|--------|--------|--------|-------------|--------------|--|
| NPROCS | NNODES | ILINES | TIME | Speedup | Effizienz | |
| 12 | 1 | 1920 | 246,84 | 1 | 1 | |
| 24 | 2 | 1920 | 124,86 | 1,976934166 | 0,9884670831 | |
| 48 | 4 | 1920 | 63,73 | 3,873215126 | 0,9683037816 | |
| 96 | 8 | 1920 | 34,02 | 7,255731922 | 0,9069664903 | |
| 120 | 10 | 1920 | 27,37 | 9,01863354 | 0,901863354 | |
| 240 | 10 | 1920 | 22,49 | 10,97554469 | 0,5487772343 | |

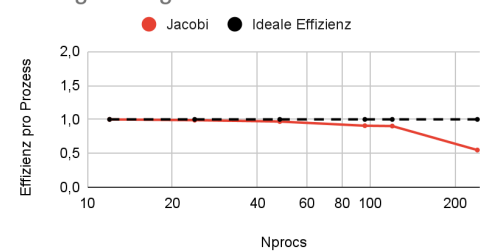
Strong Scaling JA Laufzeit



Strong Scaling JA Speedup



Strong Scaling JA Effizienz

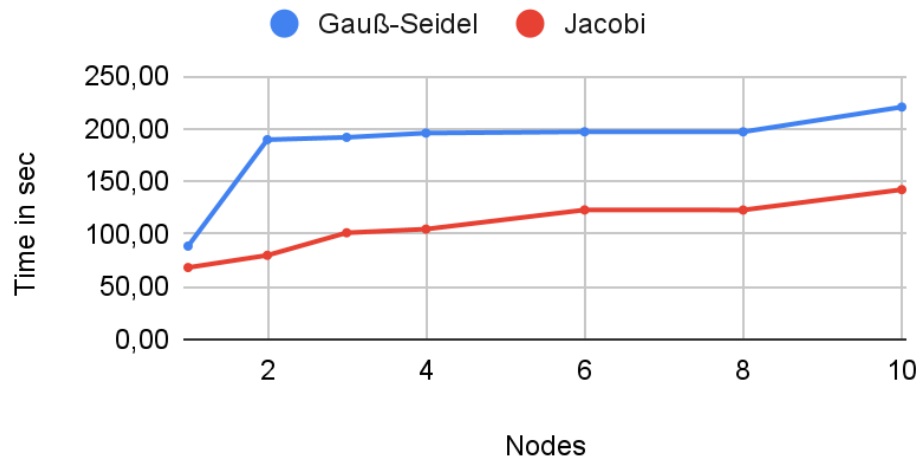


Frage: Wie sähe die Speedupkurve bzw. Effizienzkurve bei idealem Strong Scaling aus?

Eine ideale Speedupkurve würde proportional zu der Anzahl der Prozesse steigen, also ein linearer Graph. Eine ideale Effizienzkurve hingegen würde ein linearer Graph mit Steigung 0 sein, der an Effizienz(nprocs) = 1 liegt für jede Anzahl von Prozessen

| 1.3 Communication | | | | | |
|-------------------|--------|--------|---------|---------|--|
| NPROCS | NNODES | ILINES | TIME GS | TIME JA | |
| 10 | 1 | 200 | 88,74 | 68,38 | |
| 10 | 2 | 200 | 189,83 | 79,99 | |
| 10 | 3 | 200 | 192,04 | 101,40 | |
| 10 | 4 | 200 | 196,11 | 104,85 | |
| 10 | 6 | 200 | 197,31 | 123,08 | |
| 10 | 8 | 200 | 197,27 | 123,00 | |
| 10 | 10 | 200 | 220,83 | 142,36 | |

Communication



Frage: Gibt es Auffälligkeiten? Wie erklären Sie sich diese?

Im Gegensatz zum Strong bzw. Weak Scaling wurde die Problemgröße bzw. die Anzahl der Prozesse zum parallelen Rechnen nicht erhöht. Dennoch ist die Laufzeit gestiegen, was anhand der Auswertung auf die Anzahl der Rechnerknoten zurückzuführen ist. Erklären lässt sich dies über den erhöhten Kommunikationsaufwand zwischen Knoten (Latenz, Bandbreite, Overhead)

Aufgabe 2)

Manuelles Profiling

```
[Rank 0] MPI_Bcast: called at: 0.000055 ended at: 0.000069, duration: 0.000014
[Rank 1] MPI_Bcast: called at: 0.000001 ended at: 0.000074, duration: 0.000073
[Rank 0] MPI_Issend: called at 1.347843
[Rank 0] MPI_Issend: called at 1.348158
[Rank 0] MPI_Waitall: called at: 1.348164 ended at: 1.348214, duration: 0.000050
[Rank 1] MPI_Recv: called at: 0.535468 ended at: 1.348168, duration: 0.812699
[Rank 1] MPI_Recv: called at: 1.348198 ended at: 1.348203, duration: 0.000005
[Rank 1] MPI_Issend: called at 1.348546
[Rank 0] MPI_Recv: called at: 2.138935 ended at: 2.138998, duration: 0.000064
[Rank 0] MPI_Issend: called at 2.139272
[Rank 0] MPI_Issend: called at 2.139474
[Rank 0] MPI_Waitall: called at: 2.139479 ended at: 2.145357, duration: 0.005877
[Rank 1] MPI_Waitall: called at: 2.145228 ended at: 2.145271, duration: 0.000043
[Rank 1] MPI_Recv: called at: 2.145308 ended at: 2.145343, duration: 0.000035
[Rank 1] MPI_Recv: called at: 2.145349 ended at: 2.145352, duration: 0.000002
[Rank 1] MPI_Issend: called at 2.145629
[Rank 0] MPI_Recv: called at: 2.967905 ended at: 2.967959, duration: 0.000053
[Rank 1] MPI_Waitall: called at: 2.848501 ended at: 2.967975, duration: 0.119474
[Rank 0] MPI_Issend: called at 2.968202
[Rank 0] MPI_Issend: called at 2.968458
[Rank 0] MPI_Waitall: called at: 2.968464 ended at: 2.968491, duration: 0.000027
[Rank 1] MPI_Recv: called at: 2.968005 ended at: 2.968474, duration: 0.000469
[Rank 1] MPI_Recv: called at: 2.968485 ended at: 2.968489, duration: 0.000004
[Rank 1] MPI_Waitall: called at: 3.778351 ended at: 3.778352, duration: 0.000002
```

| Zeit | Rang 0 | Rang 1 |
|-----------|----------------------------------|--------------------------------------|
| ~ 0.0000s | MPI_Bcast | MPI_Bcast |
| 0.5355s | | MPI_Recv (A) |
| 1.3478s | MPI_Issend | |
| 1.3481s | MPI_Issend, MPI_Waitall (B) | MPI_Recv (A), MPI_Recv (in/out) |
| 1.3482s | MPI_Waitall (B) | |
| 1.3485s | | MPI_Issend |
| 2.1389s | MPI_Recv (in/out) | |
| 2.1392s | MPI_Issend | |
| 2.1394s | MPI_Issend, MPI_Waitall (C) | |
| 2.1452s | | MPI_Waitall (in/out) |
| 2.1453s | MPI_Waitall (C) | MPI_Recv (in/out), MPI_Recv (in/out) |
| 2.1456s | | MPI_Issend |
| 2.8485s | | MPI_Waitall (D) |
| 2.9679s | MPI_Recv (in/out) | MPI_Waitall (D) |
| 2.9680s | | MPI_Recv (E) |
| 2.9682s | MPI_Issend | |
| 2.9684s | MPI_Issend, MPI_Waitall (in/out) | MPI_Recv (E), MPI_Recv (in/out) |
| 3.7783s | | MPI_Waitall (in/out) |



Vergleichen Sie den Vampir-Trace mit Ihrem Sequenz-Diagramm.

Frage: Welche Gemeinsamkeiten und Unterschiede können Sie feststellen?

Wenn wir die Vampir-Ausgabe und das manuelle Profiling gegenüberstellen, erkennen wir direkt Gemeinsamkeiten in der Kommunikationsdauer. Zuerst ist ein auffällig langer MPI_Recv von Prozess 1 zusehen, da dieser warten muss bis Prozess 0 ein ganze Iteration auf seiner Matrix Allokierung durchgeführt. Dies dauerte bei beiden Läufen ca. 0.8 Sekunden. Die MPI_Waitall von Prozess 0 sind in Vampir kaum erkennbar: sie liefen extrem kurz. Das liegt daran, dass Prozess 1 schon in dem MPI_Recv Block ist. Prozess 1 schickt nach Berechnung der ersten Zeile diese hoch zu Prozess 0. Dies läuft wieder extrem kurz ($6.4 \cdot 10^{-5}$ Sekunden). Die Kommunikation läuft hier wie im Schema zu Blatt 9 weiter. Im Allgemeinen ist der Programmlauf bei beidem gleich.

Unterschiede sind lediglich in der Abstraktion und der Erschließbarkeit der Daten zu finden. Bei der Vampir Ausgabe erkennt man eindeutig die Kommunikationsrichtung, aber bei den Prints wird das nicht sofort eindeutig und man muss gewandter in der Systemstruktur des vorliegenden Programms sein.