Aufgabe 1)

Als Referenz für Zeitmessungen diente das Programm aus dem Material, mit dem Original Makefile compiliert und wie folgt aufgerufen (1 = Threads, 2 = Jacobi-Verfahren, 512 = interlines, 2 = mit Störfunktion, 1 = Abbruch hinreichender Genauigkeit bzw. 2 = Abbruch nach 256 Schritten):

```
srun time ./partdiff-seq 1 2 512 2 1 1e-6
```

Laufzeit bei Abbruch mit Genauigkeit 10^{-6} t_G = 0.582670 s,

```
srun time ./partdiff-seq 1 2 512 2 1 1e-7
slurmd[west1]: *** STEP 71394.0 CANCELLED AT 2014-11-13T03:58:52 DUE TO
    TIME LIMIT ***
srun: error: west1: task 0: Terminated
```

Laufzeit bei 10^{-7} t_G = > 6 STunden, Abbruch!

```
srun time ./partdiff-seq 1 2 512 2 2 256
```

Laufzeit bei Abbruch nach 256 Schritten t_S = 168.396228 s. Die Tests mit Parallelisierung wurden mit dem Abbruch-Kriterium 256 Schritte gemacht:

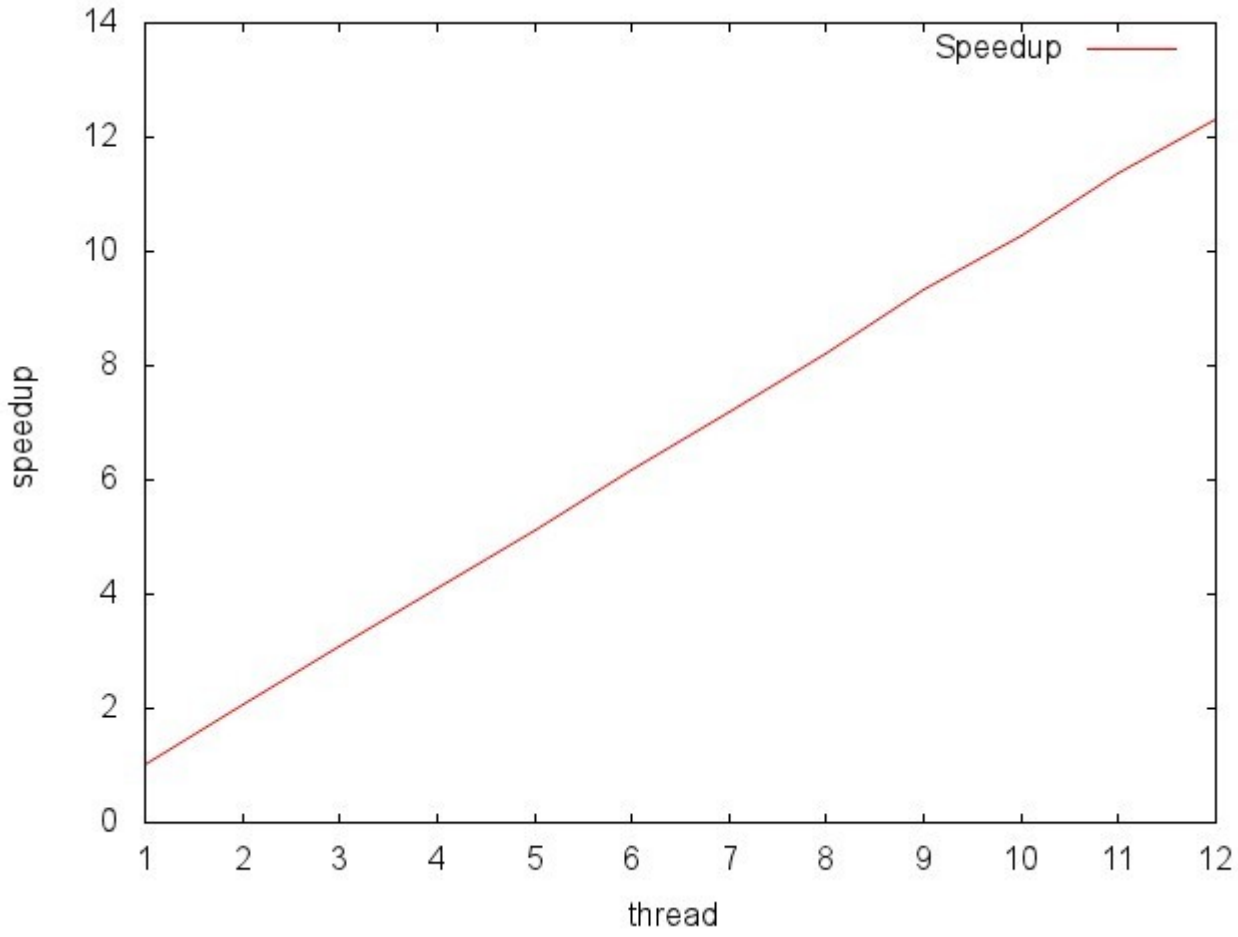
```
srun time ./partdiff-openmp-zeilen   1 2 512 2 2 256
srun time ./partdiff-openmp-spalten  1 2 512 2 2 256
srun time ./partdiff-openmp-element  1 2 512 2 2 256
```

Laufzeit Zeilenweise: t_S = 9.494860 s Erreichtes Speedup: **17,74**, Spaltenweise: t_S = 9.513828 s Erreichtes Speedup: **17,70**, Elementweise: t_S = 8.417721 s Erreichtes Speedup: **20,01**. Fazit: Parallelisierung bringt was!

Aufgabe 2)

Mit 2048 Iterationen /partdiff-openmp-zeilen 1 2 512 2 2 2048

Threads	Runtime	Mean	Speedup
12	1m46.121s	106.89	12.31
11	1m55.665s	115.66	11.37
10	2m7.767s	127.80	10.29
9	2m21.104s	141.10	9.32
8	2m40.478s	160.48	8.20
7	3m2.770s	182.77	7.19
6	3m33.357s	213.16	6.17
5	4m15.539s	255.54	5.14
4	5m19.556s	319.55	4.11
3	7m4.386s	424.40	3.10
2	10m36.117s	636.12	2.06
1	21m55.645s	1315.64	1.00

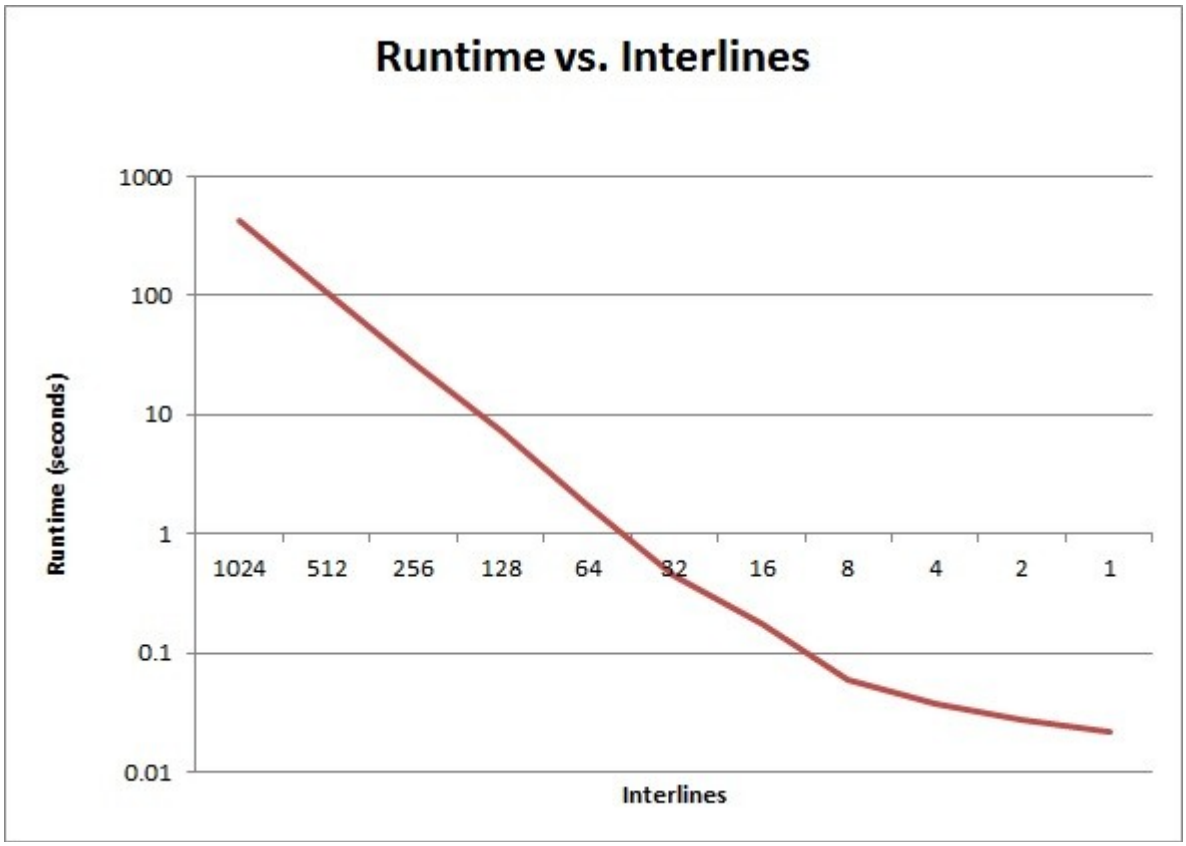


Wir konnten mit unserem Programm einen linearen Speedup unseres Programms bei vergrößerung der Threadanzahl erreichen. Der Speedup liegt sogar etwas größer als die Anzahl an Threads. Das Programm profitiert dabei vom verkleinern der Arbeitspakete pro Thread. Dies geht solange gut, bis die Kommunikation wesentlich mehr Rechenzeit benötigt, als das Rechnen selbst

time ./partdiff-openmp-zeilen 1 2 1024 2 2 2048

Interlines	Runtime
1024	7m7.696s =427.696
512	1m46.496s=106.496
256	0m27.314s
128	0m7.309s
64	0m1.714s
32	0m0.451s
16	0m0.179s
8	0m0.059s

4	0m0.037s
2	0m0.027s
1	0m0.022s



Die Laufzeit des Programms erhöht sich mit jeder Verdopplung der Interlines um etwa eine halbe 10er Potenz in der Laufzeit. Unterhalb von 8 Interlines scheint dies nicht zu gelten. Die Differenzen in der Laufzeit zwischen 1-8 Interlines müssen nicht durch das Problem gegeben sein, sondern können z.B. I/O bedingt sein. Bei den kleinen Problemen könnte der Overhead durch die Ausführung mit 12 Threads zudem ein Problem darstellen. Ob auch wirklich alle Threads hier gearbeitet haben ist nicht wirklich nachvollziehbar.

Aufgabe 3)

time ./partdiff-openmp-element 1 2 512 2 2 265

Static,1	0m14.289s
Static,2	0m14.326s
Static,4	0m14.403s
Static,16	0m14.321s
Dynamic,1	0m14.297s
Dynamic,4	0m14.537s
Guided	0m14.189s

time ./partdiff-openmp-zeilen 1 2 512 2 2 265

Static,1	0m14.009s
Static,2	0m13.992s
Static,4	0m14.285s
Static,16	0m14.016s
Dynamic,1	0m14.005s
Dynamic,4	0m14.076s
Guided	0m14.042s

Static scheint ein klein wenig schneller zu sein als die beiden Alternativen, die Unterschiede liegen aber im Bereich der Ungenauigkeit. Dies kann auf einem System, welches noch andere Aufgaben bewältigt anders aussehen! Unter den Umständen wie sie sich hier darstellen ist es zunächst egal, welche Variante benutzt wird. Getestet wurde auf einem System (west1), welches über genügend Leistungsreserven verfügt, sodass die Threads jeweils auf einer eigenen CPU laufen können und nicht durch andere Prozesse oder das Betriebssystem unterbrochen werden. Hat man jedoch ein System, in welchem andere Prozesse einige CPUs blockieren oder die Berechnungen unterschiedlich lange dauern (aufgrund von Eingabedaten/Komplexität) wird die Auswahl des Sceduling Verfahrens wichtig.