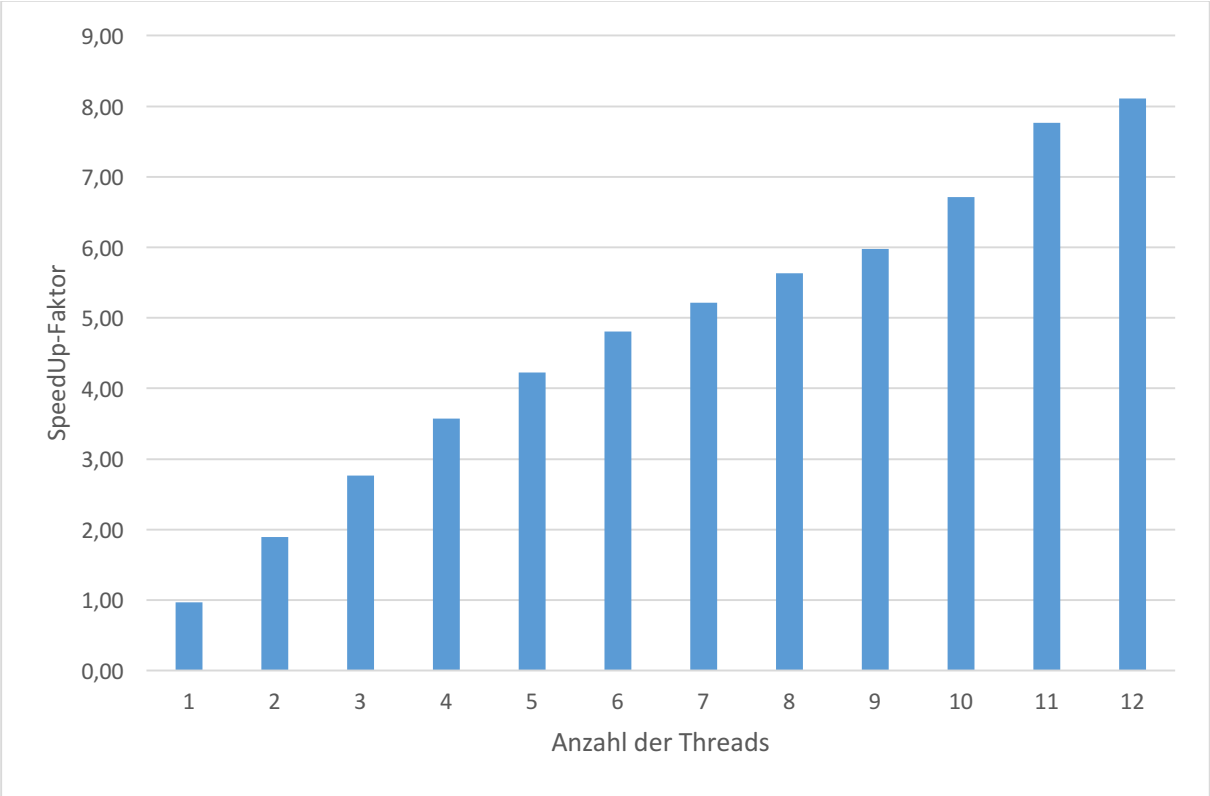
Leistungsanalyse

Alle Messungen wurden mit folgendem Aufruf auf dem Knoten west1 ausgeführt:

```
Aufruf:   ./partdiff-posix x 2 512 2 2 1024
```

Messreihe:

Laufzeit des Programmes bei verschiedenen Thread-Anzahlen:
(Verwendete SpeedUp-Formel: Zeit Sequentiell / Zeit POSIX)



Es fällt auf, dass diese Version mit einem Thread etwas langsamer läuft als mit der sequentiellen Version. Das liegt wahrscheinlich daran, dass der Aufwand für das Thread-Management und das Verwenden von Mutex mehr Aufwand erfordert. Dieser Mehraufwand schlägt sich dann bei einem Thread auf die Laufzeit in negativer Form nieder. Das ändert sich aber bei einem Lauf mit zwei Threads, denn dort halbiert sich die Laufzeit gegenüber dem Sequentiellen Programm nahezu. Im Folgenden kann man einen fast gleichmäßig steigenden SpeedUp feststellen, der aber mit der Zunahme der Threads etwas abnimmt. Mit fünf Threads ist das Programm 4,22-mal schneller und mit acht Threads 5,64-mal schneller. Erst im Sprung von 10 auf 11 Threads sieht man erneut einen großen Sprung beim SpeedUp-Faktor von 6,71 auf 7,77. Dieser Effekt hält aber nicht an, denn bei dem Sprung auf 12 Threads steigt der Faktor nur um 0,44. Mit 12 Threads erhalten wir also einen SpeedUp-Faktor von 8,11 und erreichen somit die in der Aufgabe erforderten Faktor 8.

Tabelle: Mittelwerte und SpeedUp-Faktor

Programm-Version	Mittelwert (s)	SpeedUp-Faktor
Sequentiell	572,822	
POSIX 1 Thread	593,477	0,97
POSIX 2 Threads	303,325	1,89
POSIX 3 Threads	207,116	2,77
POSIX 4 Threads	160,464	3,57
POSIX 5 Threads	135,672	4,22
POSIX 6 Threads	119,182	4,81
POSIX 7 Threads	109,877	5,21
POSIX 8 Threads	101,663	5,63
POSIX 9 Threads	95,823	5,98
POSIX 10 Threads	85,387	6,71
POSIX 11 Threads	73,756	7,77
POSIX 12 Threads	70,625	8,11

Messdaten:

```
Sequentiell:
Lauf 1: 571.641 s
Lauf 2: 571.735 s
Lauf 3: 575.089 s

1 Thread:
Lauf 1: 593.502 s
Lauf 2: 593.297 s
Lauf 3: 593.631 s

2 Thread(s):
Lauf 1: 303.616 s
Lauf 2: 303.189 s
Lauf 3: 303.169 s

3 Thread(s):
Lauf 1: 206.964 s
Lauf 2: 207.127 s
Lauf 3: 207.258 s

4 Thread(s):
Lauf 1: 160.457 s
Lauf 2: 160.647 s
Lauf 3: 160.288 s

5 Thread(s):
Lauf 1: 136.550 s
Lauf 2: 135.309 s
Lauf 3: 135.158 s
```

```
6 Thread(s):
Lauf 1: 119.271 s
Lauf 2: 119.180 s
Lauf 3: 119.096 s

7 Thread(s):
Lauf 1: 109.758 s
Lauf 2: 109.296 s
Lauf 3: 110.578 s

8 Thread(s):
Lauf 1: 101.659 s
Lauf 2: 101.612 s
Lauf 3: 101.719 s

9 Thread(s):
Lauf 1: 96.318 s
Lauf 2: 95.100 s
Lauf 3: 96.052 s

10 Thread(s):
Lauf 1: 84.428 s
Lauf 2: 85.199 s
Lauf 3: 86.534 s

11 Thread(s):
Lauf 1: 73.915 s
Lauf 2: 74.602 s
Lauf 3: 72.751 s

12 Thread(s):
Lauf 1: 70.509 s
Lauf 2: 70.175 s
Lauf 3: 71.191 s
```

Output Sequentielles Programm:

```
============================================================
Program for calculation of partial differential equations.
============================================================
(c) Dr. Thomas Ludwig, TU München.
    Thomas A. Zochler, TU München.
    Andreas C. Schmidt, TU München.
============================================================

Berechnungszeit:    571.605921 s
Speicherbedarf:     257.126236 MiB
Berechnungsmethode: Jacobi
Interlines:         512
Stoerfunktion:      f(x,y) = 2pi^2*sin(pi*x)sin(pi*y)
Terminierung:       Anzahl der Iterationen
Anzahl Iterationen: 1024
Norm des Fehlers:   2.929037e-07

Matrix:
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 0.0001 0.0001 0.0001 0.0001 0.0001 0.0000 0.0000
 0.0000 0.0001 0.0001 0.0002 0.0002 0.0002 0.0001 0.0001 0.0000
 0.0000 0.0001 0.0002 0.0003 0.0003 0.0003 0.0002 0.0001 0.0000
 0.0000 0.0001 0.0002 0.0003 0.0003 0.0003 0.0002 0.0001 0.0000
 0.0000 0.0001 0.0002 0.0003 0.0003 0.0003 0.0002 0.0001 0.0000
 0.0000 0.0001 0.0001 0.0002 0.0002 0.0002 0.0001 0.0001 0.0000
 0.0000 0.0000 0.0001 0.0001 0.0001 0.0001 0.0001 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Output POSIX Programm mit 12 Threads:

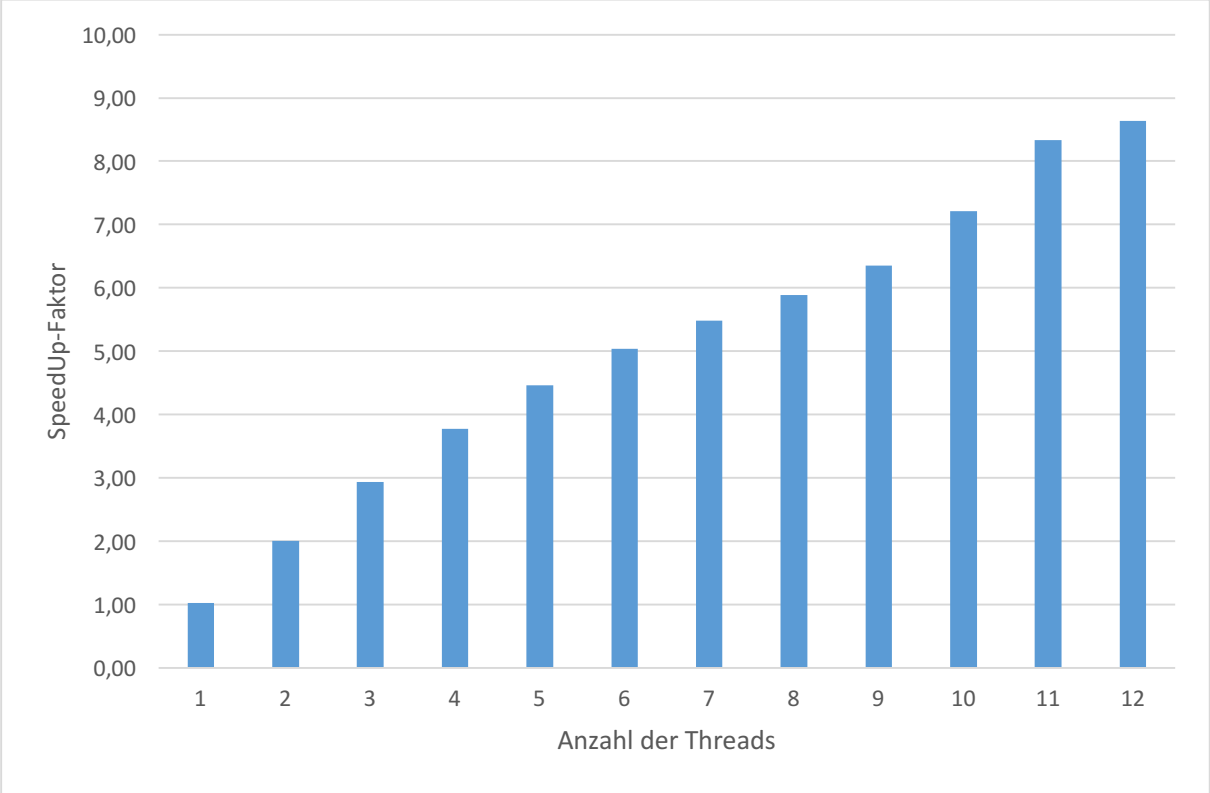
```
============================================================
Program for calculation of partial differential equations.
============================================================
(c) Dr. Thomas Ludwig, TU München.
    Thomas A. Zochler, TU München.
    Andreas C. Schmidt, TU München.
============================================================

Berechnungszeit:    70.097131 s
Speicherbedarf:     257.126236 MiB
Berechnungsmethode: Jacobi
Interlines:         512
Stoerfunktion:      f(x,y) = 2pi^2*sin(pi*x)sin(pi*y)
Terminierung:       Anzahl der Iterationen
Anzahl Iterationen: 1024
Norm des Fehlers:   2.929037e-07

Matrix:
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 0.0001 0.0001 0.0001 0.0001 0.0001 0.0000 0.0000
 0.0000 0.0001 0.0001 0.0002 0.0002 0.0002 0.0001 0.0001 0.0000
 0.0000 0.0001 0.0002 0.0003 0.0003 0.0003 0.0002 0.0001 0.0000
 0.0000 0.0001 0.0002 0.0003 0.0003 0.0003 0.0002 0.0001 0.0000
 0.0000 0.0001 0.0002 0.0003 0.0003 0.0003 0.0002 0.0001 0.0000
 0.0000 0.0001 0.0001 0.0002 0.0002 0.0002 0.0001 0.0001 0.0000
 0.0000 0.0000 0.0001 0.0001 0.0001 0.0001 0.0001 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Messreihe ohne Mutex:

Wir hatten auch eine POSIX-Version ohne Mutex, dabei haben wir das maxresiduum aller Threads am Ende zusammengeführt und ausgewertet (pardiff-posix.c Zeile: 374). In dieser Version hatte jeder Thread seine eigene maxresiduum-Variable und es wurde nicht die Adresse der eigentlichen Variable übergeben. Diese Version haben wir mit dem gleichen Aufruf und auf dem gleichen Knoten laufen lassen.

Dabei erhielten wir folgende Informationen:

Laufzeit des Programmes bei verschiedenen Thread-Anzahlen (ohne Mutex):
(Verwendete SpeedUp-Formel: Zeit Sequentiell / Zeit POSIX)



Es fällt auf, dass die POSIX-Version, anders als bei der Version mit Mutex, mit einem Thread bereits schneller läuft als die sequentielle Version. Danach verläuft der SpeedUp-Zuwachs etwa genau gleich, jedoch ist die Version ohne Mutex immer ein wenig schneller. Auch der große SpeedUp-Sprung von 10 auf 11 und der geringere Sprung von 11 auf 12 Threads ist bei dieser Messreihe wiederzufinden. Dass diese Version schneller ist, liegt wahrscheinlich daran, dass die einzelnen Threads nicht darauf warten müssen bis das maxresiduum freigegeben ist. Bei der Version ohne Mutex muss kein Thread warten, sondern es werden die einzelnen maxresiduum der Threads am Ende ausgewertet.

Tabelle: Mittelwerte und SpeedUp-Faktor:

Programm-Version	Mittelwert (s)	SpeedUp-Faktor
Sequentiell	574,494	
POSIX 1 Thread	562,827	1,02
POSIX 2 Threads	286,404	2,01
POSIX 3 Threads	196,084	2,93
POSIX 4 Threads	152,335	3,77
POSIX 5 Threads	128,643	4,47
POSIX 6 Threads	114,005	5,04
POSIX 7 Threads	104,884	5,48
POSIX 8 Threads	97,640	5,88
POSIX 9 Threads	90,446	6,35
POSIX 10 Threads	79,689	7,21
POSIX 11 Threads	68,952	8,33
POSIX 12 Threads	66,508	8,64