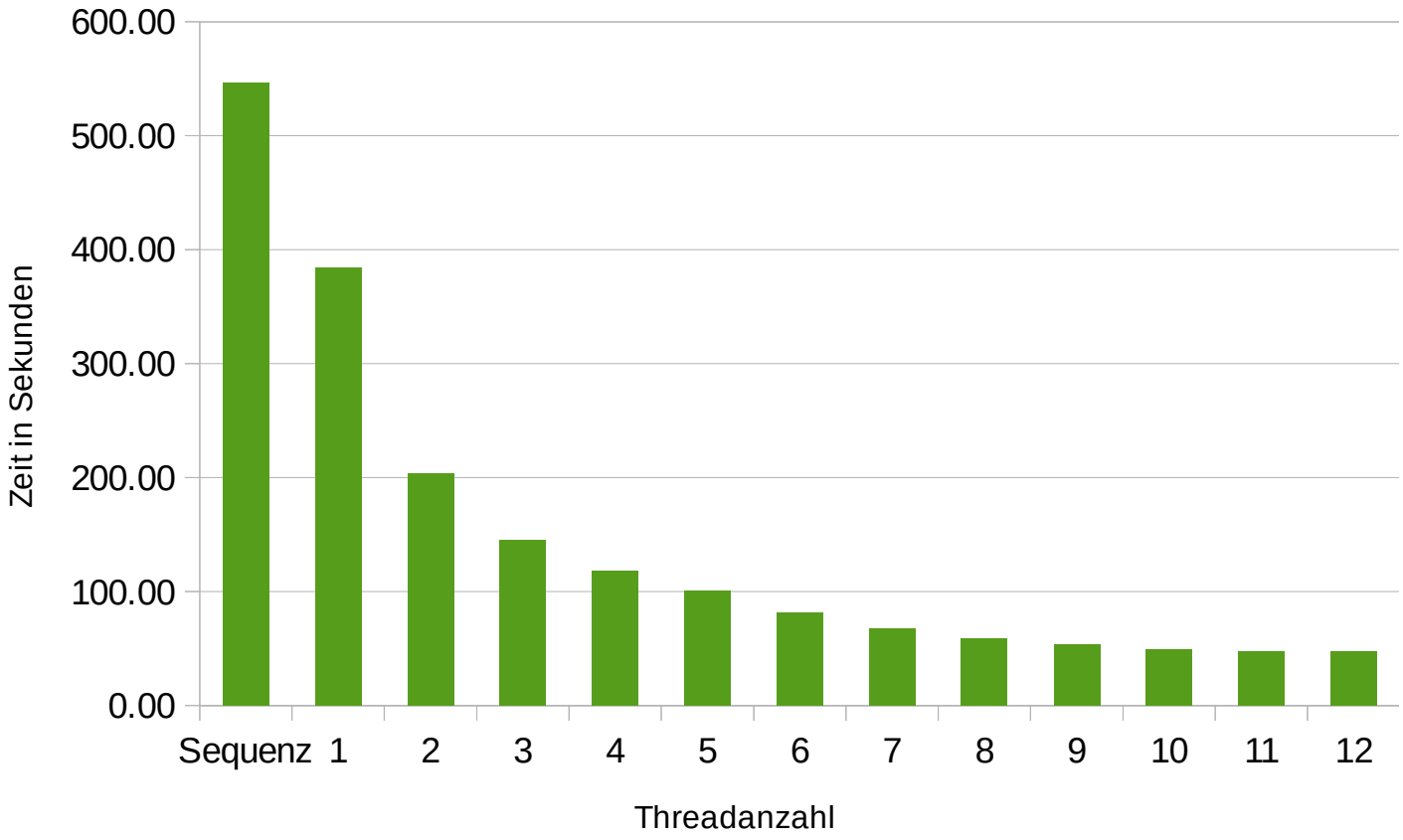
Leistungsanalyse

Um das Programm zu testen haben wir es mit verschiedenen Threadanzahlen auf verschiedenen Knoten des Clusters mit folgenden Parametern aufgerufen:

Threads: 1-12
Methode Nr: 2
Interlines: 512
Funktion Nr: 2
Termination Nr: 2
Iterationen: 2000

Die Ergebnisse haben wir Tabelliert wobei Messung 1 auf Knoten 8, Messung 2 auf Knoten 9 und Messung 3 auf Knoten 11 ausgeführt wurden.

Threads	Messung 1	Messung 2	Messung 3	Mittel
Sequenz	545.64	547.01	548.38	547.01
1	384.19	384.19	384.21	384.20
2	204.02	204.35	204.02	204.13
3	145.54	145.41	145.72	145.56
4	118.83	118.23	117.91	118.32
5	100.28	100.32	101.36	100.65
6	81.31	82.58	81.96	81.95
7	67.71	68.60	67.30	67.87
8	58.65	59.38	59.31	59.11
9	53.47	54.75	53.72	53.98
10	49.77	49.98	49.91	49.89
11	47.80	48.15	47.92	47.96
12	48.04	47.31	47.52	47.62



Der Unterschied zwischen dem Originalprogramm und der Posixversion mit nur einem Thread lässt sich durch die Benutzung des Compilerflags -Ofast erklären. Der Speedup von einem auf zwei Threads ist der Größte, was wir erwarteten da die last ungefähr durch 2 geteilt werden kann. Natürlich sollte es nicht genau 2 sein sondern weniger, da wir einen Overhead erzeugt haben. Gegen Ende des Graphen sieht man, dass der Leistungszuwachs nachlässt, auch das erwarten wir da der Unterschied von einem Faktor 11 auf einen Faktor 12 nicht so groß ist wie der eines Faktor 1 auf einen Faktor 2. Allerdings flacht die Kurve wesentlich schneller ab als Beispielsweise die openMP Parallelisierung. Das Zusammensammeln der Daten nachdem jeder Thread mit der Bearbeitung fertig ist skaliert in unserem Fall mit der Threadzahl, in unserem Fall sogar so stark, dass in einer Messung (Messung 1 mit 11 und 12 Threads) ein Aufruf mit mehr Threads länger dauert als einer mit weniger. In unserm Code wird beispielsweiße das Residuum in jedem Thread berechnet und in ein Array abgelegt, über welches am Ende eines Iterationsschrittes gelaufen werden muss um das Maxresiduum zu identifizieren, was sonst mit nur einer Variable geschieht. Hier scheinen die Speicherzugriffe ausschlaggebend zu sein, denn kommentieren wir die Berechnung des residuums aus gewinnen wir fast 2% an Leistung, wobei dies im Sequentiellem vernachlässigbar ist.