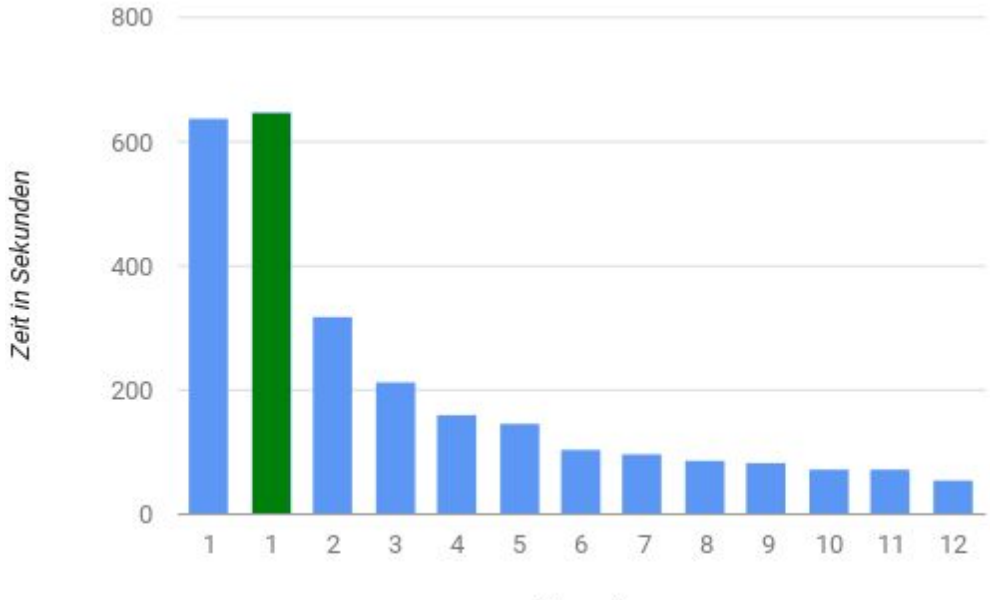
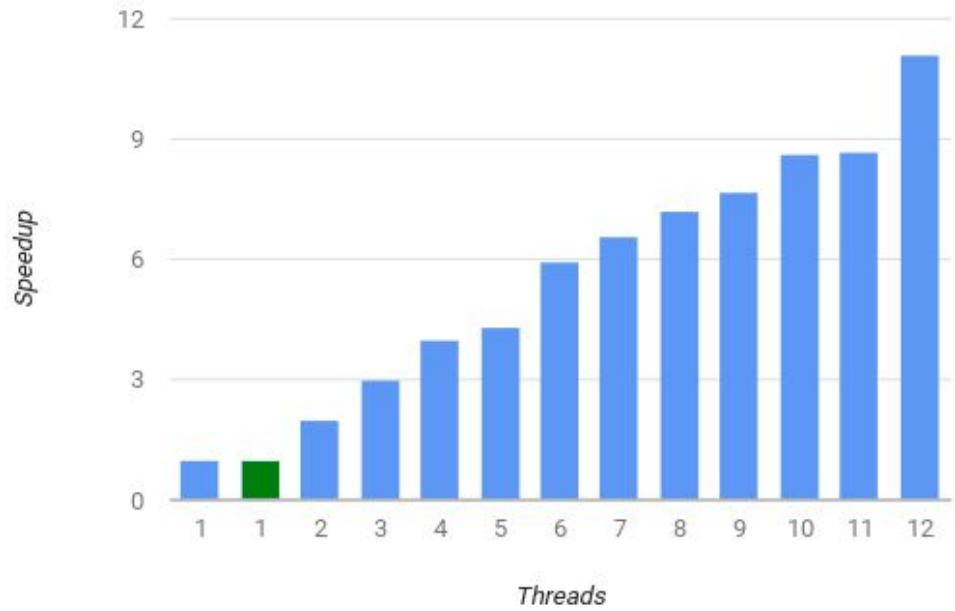
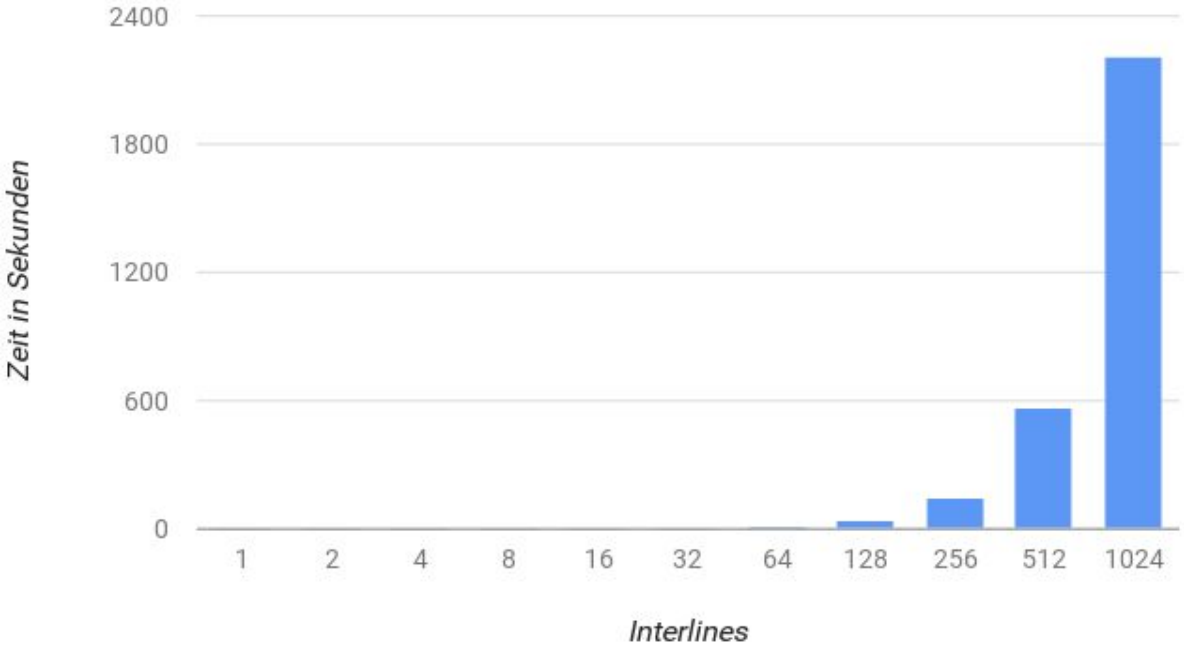
Messung 1:





Der erste Graph zeigt die Laufzeit in Sekunden in Abhängigkeit von der Anzahl der Threads. Die Laufzeit des sequentiellen Programms ist grün markiert. Der zweite Graph zeigt den erreichten Speedup bei einer bestimmten Anzahl von Threads. Bei einem Thread ist ist die Laufzeit des sequentiellen Programms und des OpenMP Programms nahezu identisch, da bei nur einem Thread OpenMP nicht parallelisieren kann. Insgesamt nimmt der Speedup mit wachsender Anzahl von Threads linear zu.

Messung 2:



Der Graph visualisiert die benötigte Zeit für die Berechnungen von Interlines in Schritten von 2^x Interlines mit x von 0-10. Der Zeitaufwand für die Berechnungen wächst dabei exponentiell.
Dies liegt daran, dass die Breite und Höhe der Matrix jeweils interlines*8*9 lang ist. Deshalb vergrößert sich die Anzahl der zu berechnenden Gitterpunkte in der Matrix exponentiell, wenn die Anzahl der Interlines vergrößert wird. Dementsprechend wächst auch der Zeitaufwand exponentiell.

Bonus 1 Datenaufteilung:

Zeilenweise	Spaltenweise	Elementweise
65,94s	161,22s	115,16s

Die Zeilenweise Aufteilung der Daten ist die schnellste Art die Ergebnisse zu berechnen. Dies wird dadurch verursacht, dass bei dieser Vorgehensweise die Cache Misses reduziert werden. Da beim einlesen die Daten nebeneinander im Speicher liegen wird der benachbarte Speicher beim einlesen mit in den Cache des jeweiligen Knoten geladen. Auf die Daten wird bei der Verarbeitung ebenfalls Zeilenweise zugegriffen und somit reduziert sich die Anzahl der Cache Misses. Bei einer Spaltenweise Aufteilung liegt aus diesem Grund die Anzahl der Cache Misses deutlich höher. In der Mitte vom Zeitaufwand liegt die Elementweise Aufteilung, da auch die Cache Misses zwischen der den anderen beiden

Vorgehensweisen liegt. Hier kann es zufällig sein, dass nacheinander auf Zeileweise benachbarte Elemente zugegriffen wird oder aber auch Spaltenweise.