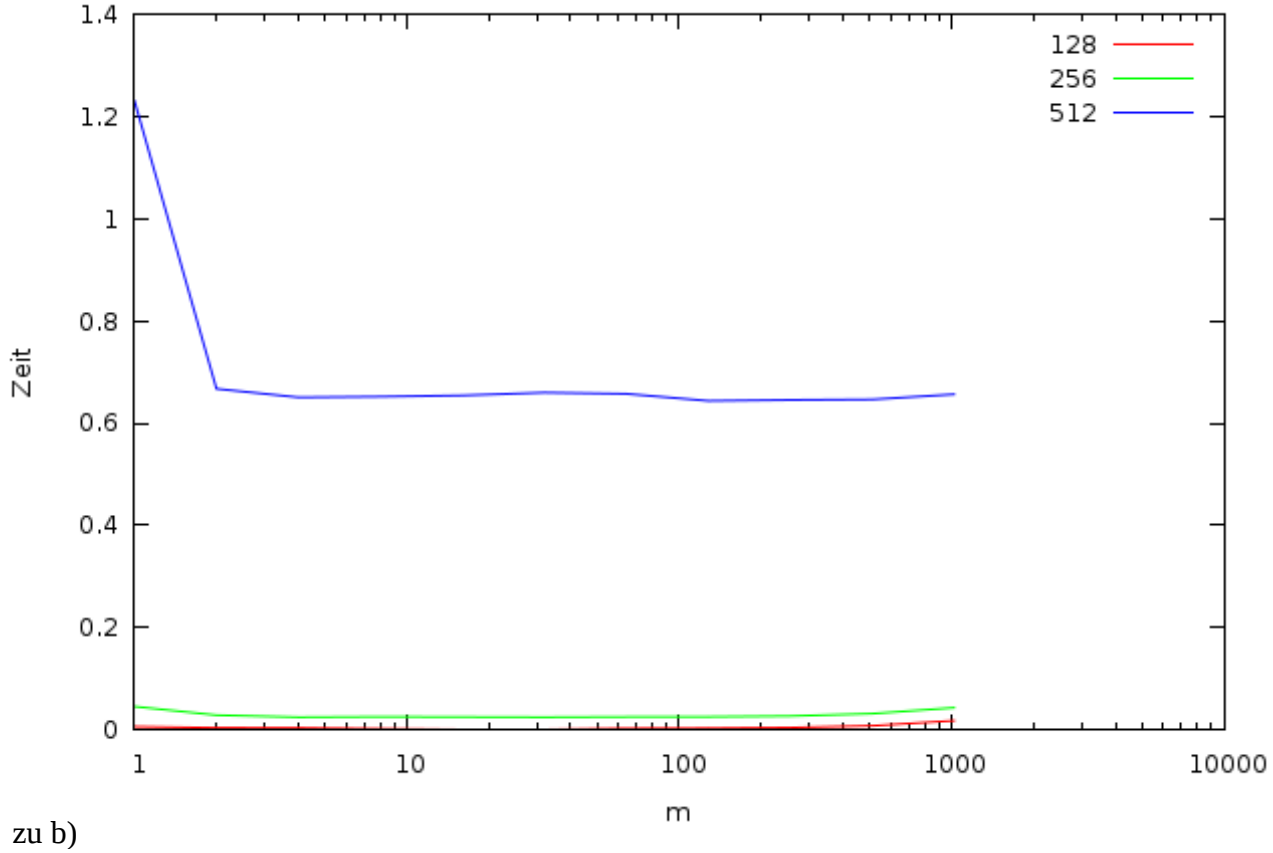
Übung 6
d = Tiefe des Baumes
Knotengrad: 3
Gesamtzahl der Verbindungen: d*2^d
Netzwerkdurchmesser: 2d
Bisektionsbreite: 2^d

Übung 7
Jeder Prozess hat eine lokale Variable *mine*, die er am Anfang auf 0 setzt und zwei öffentliche Variablen *choosing*, die mit 0 initialisiert wird und jeder Prozess der den kritischen Abschnitt(CS) betreten möchte diese auf 1 setzt, und *number*, die vorerst auf 0 gesetzt ist. Nun vergleicht der Prozess, der in den CS möchte seine eigene *mine*-Variable mit den öffentlichen *number*-Variablen und setzt *mine* auf *number* des Prozesses k wenn *number*[k] > *mine* ist ansonsten bleibt *mine* unberührt. Daraufhin setzt der Prozess die eigene *number*-Variable auf *mine*+1, sodass der aktuelle Prozess im Vergleich zu den anderen Prozessen den größten Wert in *number* hat und setzt seine *choosing*-Variable auf 0.
Die nun folgende For-Schleife ist unterteilt in 2 While-Schleifen. Die erste While-Schleife dient als Barrier, da hier alle Prozesse so lange blockieren bis alle anderen Prozesse das Entry-Protokoll durchlaufen und damit ihre *choosing*-Variable auf 0 gesetzt haben oder noch gar nicht in den kritischen Abschnitt wollen (das Entry-Protokoll noch gar nicht erreicht haben).
Die zweite While-Schleife sorgt dafür, dass immer nur ein Prozess weiterlaufen kann indem eine Reihenfolge festgelegt wird, die auf dem Inhalt der *number*-Variablen beruht. Da jedem Prozess eine unterschiedliche Nummer in der *number*-Variablen zugeteilt wurde(sofern nicht 2 Prozesse gleichzeitig eine Nummer gezogen haben > siehe Fall3), kann nun überprüft werden ob der jeweilige Prozess das Eingangsprotokoll durchlaufen und eine Nummer „gezogen“ hat (*number*[k] ! = 0), der jeweilige Prozess der Prozess mit der kleinsten *number*-Variablen ist oder aber der Prozess die selbe *number*-Variable hat, aber eine höhere Prozesspriorität hat (k<i).
Hat daraufhin der Prozess die Bedingungen erfüllt und darf in den CS, blockieren alle anderen und nachdem er im CS fertig ist verlässt er diesen indem er die *number*-Variable zurücksetzt (=0).

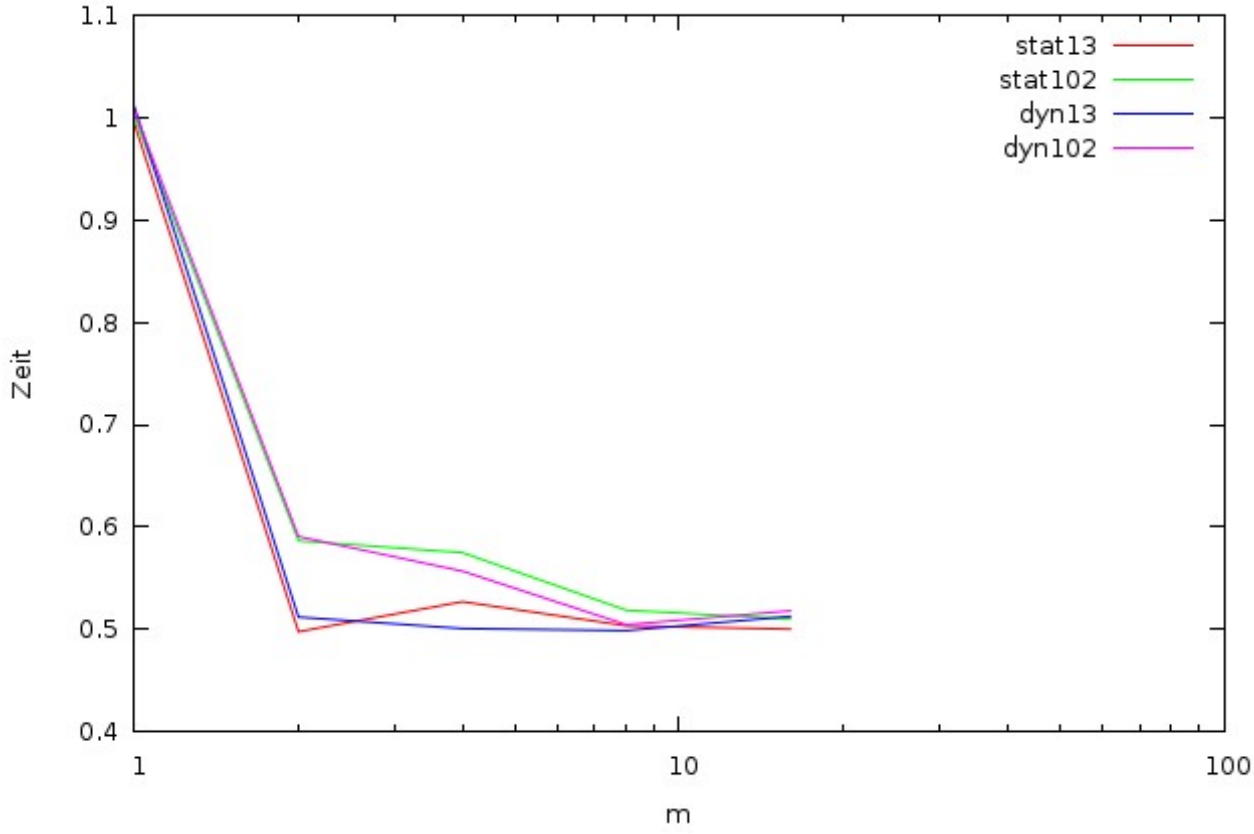
Übung8
a)
Eine höhere Anzahl an Threads beschleunigt die Programmabarbeitung so lange bis die Kosten für das Erzeugen und Wechseln der Threads teurer ist als die eigentliche Berechnung des Programms, was etwa bei 2-4 Threads der Fall ist.

b)
Der Unterschied zwischen static und dynamic Scheduling ist kaum auszumachen. Lediglich die unterschiedliche Chunksize lässt sich bis etwa 10 Threads erkennen und sorgt bei der Größe 102 für einen Performance-Anstieg gegenüber der Größe 13.

zu a)


1.4
1.2
1
0.8
0.6
0.4
0.2
0
Zeit
1
10
100
1000
10000
m
128
256
512

zu b)


1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
Zeit
1
10
100
m
stat13
stat102
dyn13
dyn102