Aufgabe 1

Frage:
„When run with one thread per logical core, threads from Listing 3.1 display their numbers randomly, while those from Listing 3.2 show them in the same order. Explain why“

Antwort:
Da wir bei der Berechnung von Fibonacci-Zahlen (es wird `zahl + thread_num` verwendet) die unterschiedlichen Threads unterschiedliche Werte verarbeiten, dauert die Berechnung für „frühere“ Threads weniger, weil sie kleinere Zahlen berechnen müssen und somit schneller fertig werden. Dadurch wird auch die Reihenfolge der Ausgabe eingehalten.

Aufgabe 2

Frage:
„What do you think, does the code snippet in Listing 3.10 use the cores efficiently, why yes, why no. Consider also the variable size in your argumentation.“

Antwort:
Falls die Matrixgröße (plane) nicht gleichmäßig unter den verfügbaren bzw. festgelegten Threads verteilt werden kann, kann das sein, dass manche Threads unter- oder überlastet werden. Um die Last dynamisch zu verteilen, könnte `schedule(dynamic)` verwendet werden. Dadurch könnten Arbeitsschritte den Threads flexibel zugeteilt werden. So wird eine bessere Nutzung der Kerne erreicht.

Aufgabe 3

Frage:
„..take your parallelized version ... and modify it based on the insights from Example 3.5“

Antwort:
Ich habe den Code so modifiziert, dass jeder Thread eigenen Seed bekommt, der an Thread-ID gekoppelt ist. Da jeder Thread eindeitige THread-ID hat, wird dadurch sichergestellt, dass jeder seinen eigenen unabhängigen Zufallsgenerator hat. Für Zufallszahlgeneration wurde rnd-Funktion aus Listing 3.18 übernommen. Im Verglecih zur alten Version hat sich Performance verbessert - 0.116s statt 3.689s