Parallel Fibonacci

Diese Aufgabe aus dem letzten Jahr soll dazu dienen, das im PA6 und in der Vorlesung eingeführte Konzept des Threadings nochmal zu verfestigen:

Die Fibonaccifolge wird wir folgt berechnet:

$$f_n = \begin{cases} 0 : n = 0 \\ 1 : n = 1 \\ f_{n-1} + f_{n-2} : n > 1 \end{cases}$$

Schreiben Sie eine von `Thread` abgeleitete Klasse `ParallelFib`, die ...

- eine statische Methode `fibSeq` hat, welche f_n rekursiv berechnet (ohne Threads),
- die Attribute `int n` und `int result` hat,
- im Konstruktor den Wert `n` speichert,
- in ihrer `run`-Methode `result` auf f_n setzt, falls $n = 0$ oder $n = 1$
- in ihrer `run`-Methode zwei neue `Fib`-Objekte für f_{n-1} und f_{n-2} erzeugt, diese parallel ausführt und das Ergebnis in `result` speichert,
- eine statische Methode `fib(int n)` hat, welche ein `Fib`-Objekt konstruiert, es sequentiell ausführt und das Ergebnis `result` zurückgibt.

Schreiben Sie zusätzlich eine `main`-Methode, welche die Laufzeiten von `fib(n)` und `fibSeq(n)` mittels `System.currentTimeMillis()` vergleicht. Stellen Sie sicher, dass beide Implementierungen die gleichen (korrekten) Werte liefern.

Lassen Sie *vor der eigentlichen Messung* `fib(40)` und `fibSeq(40)` einmal laufen, um Latenzen beim Threadstart entgegen zu wirken und dem JIT die Möglichkeit zu geben, beide Methoden zu optimieren.

Fügen Sie ein `depth`-Attribut in Ihrer `ParallelFib`-Klasse ein, das die aktuelle Rekursionstiefe mitzählt und modifizieren Sie die `run`-Methode so, dass `fibSeq()` verwendet wird, sobald die Rekursionstiefe über 6 liegt (wie viele Threads werden maximal gestartet?).

Vergleichen Sie die Laufzeiten von `fibSeq(n)` und `fib(n)` für verschiedene $\mathtt{n} \in \{1, \ldots, 45\}$ (vorzugsweise auf einem System mit mehr als einer CPU). Ab welchem **n** ist die parallele Version schneller as die sequentielle? Warum ist die parallele Version nicht *immer* schneller?