Parallele Programmierung

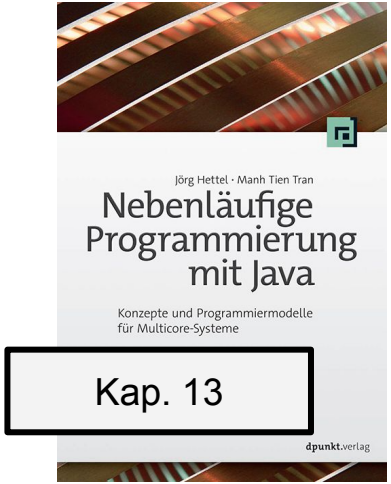
11. ForkJoin-Framework

Organisatorisches

- Anmeldung zur Prüfung?

Überblick

- ForkJoin-Framework
  - Kontrollfluss
  - Rekursion
- Programmiermodell
  - `ForkJoinPool` und “Work-Stealing”
  - `ForkJoinTask`
  - unterschiedliche Ausführungsmodi
- Kleine Beispiele (Array-Verarbeitung)
  - `RecursiveAction` (Filter)
  - `RecursiveTask` (Summieren)

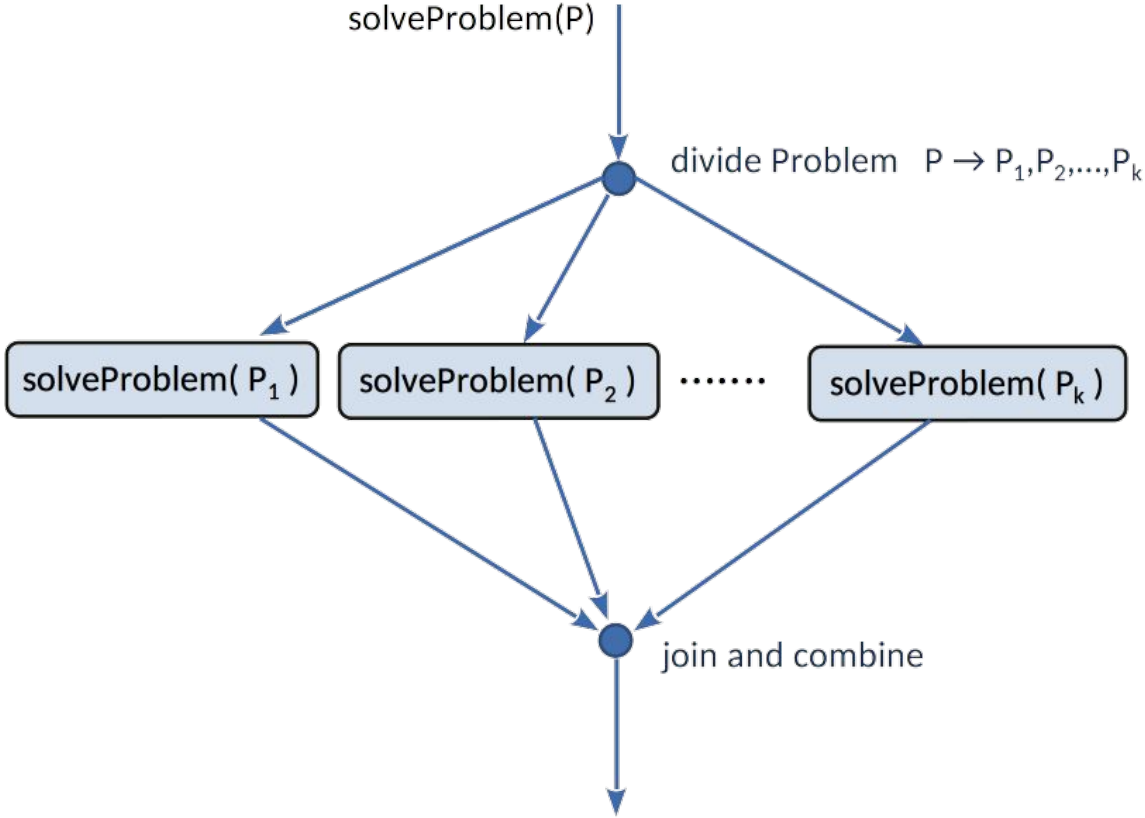


ForkJoin Framework

Kontrollfluss des ForkJoin-Patterns

Folie mit Anmerkungen

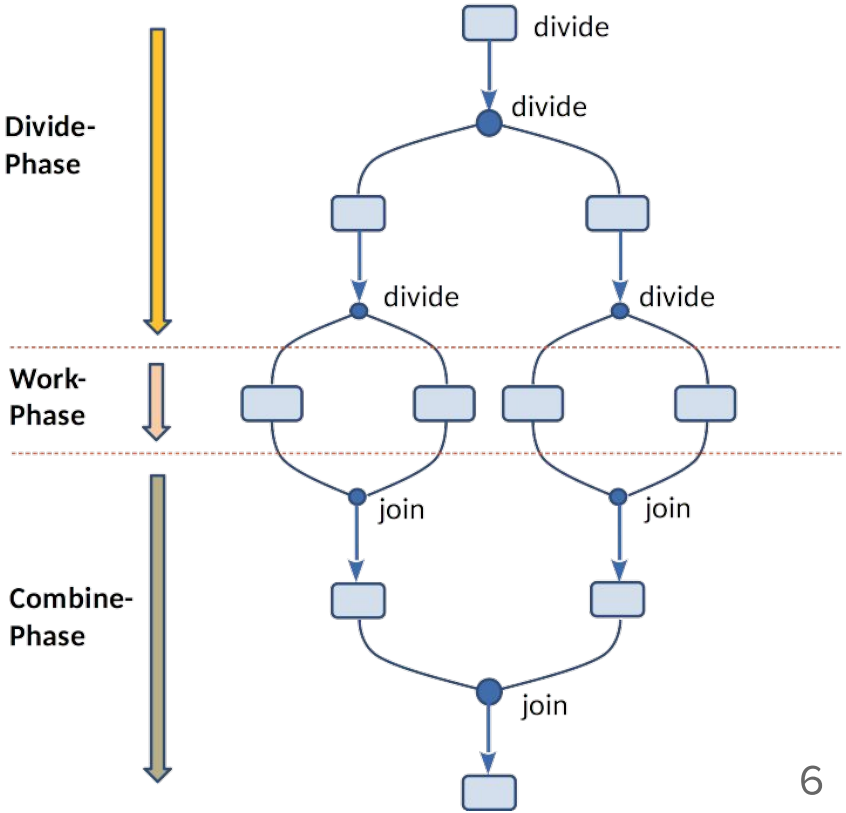
Kontrollfluss wird an einer Stelle in mehrere nebenläufige Flüsse aufgeteilt (“fork”), die an einer späteren Stelle alle wieder vereint (“join”) werden



Bildquelle: Hettel & Tran 2016

Rekursive Verwendung

Besonders geeignet für rekursive Algorithmen (“Divide and Conquer”):
In der ersten Phase wird das Problem immer wieder zerkleinert (*Divide-Phase*).
Ist eine entsprechende Problemgröße erreicht, werden die Teilaufgaben gelöst (*Work-Phase*) und anschließend das Ergebnis zusammengesetzt (*Combine-Phase*).



Bildquelle: Hettel & Tran 2016

Programmier-modell

ForkJoinPool und “Work-Stealing”

Folie mit Anmerkungen

`ForkJoinPool` (seit Java 7) implementiert `ExecutorService`:

- Standard-Pool bereits instanziiert: `ForkJoinPool.commonPool()` (seit Java 8)
- Using the common pool normally reduces resource usage (its threads are slowly reclaimed during periods of non-use, and reinstated upon subsequent use).
- Parallelität wird im Konstruktor vorgegeben, Default-Wert: `Runtime.getRuntime().availableProcessors()`; Maximum: 32767
- Eigener Scheduler: Alle Threads im Pool versuchen anstehende Subtasks zu finden und zu übernehmen, die von anderen aktiven Tasks erzeugt wurden (“Work-Stealing”).
- Effiziente Verarbeitung, wenn die meisten Tasks Subtasks erzeugen.

Folie mit Anmerkungen

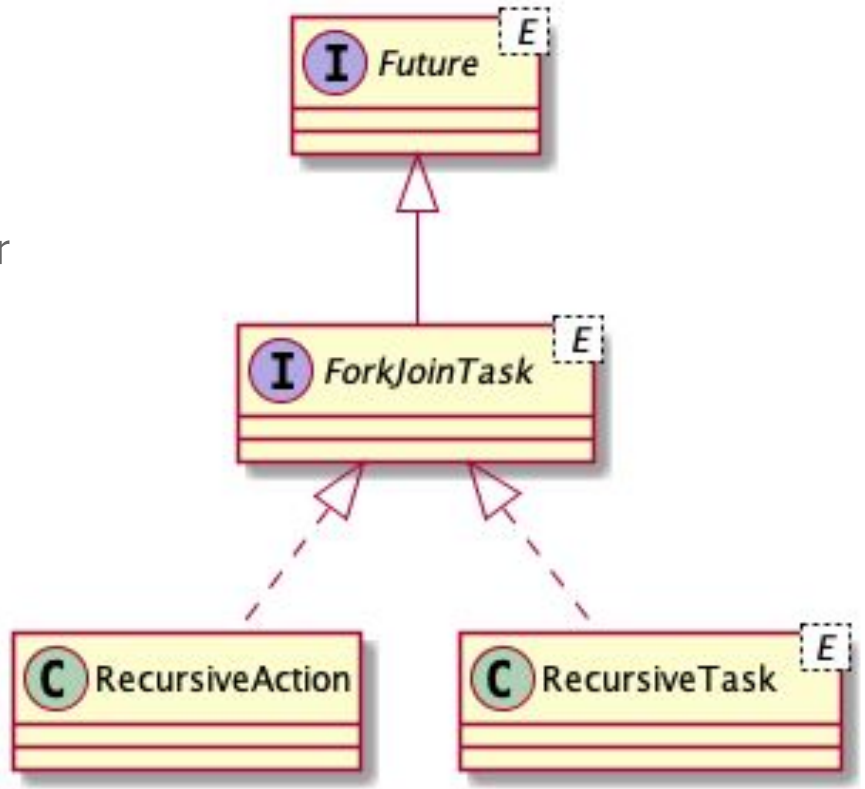
Programmiermodell

`ForkJoinTask<E>`: leichtgewichtiges `Future<E>` (soll nicht blockieren)

`RecursiveAction`: Kein Rückgabewert (nur Seiteneffekt, z.B. bei *in-place* Sortierung)

`RecursiveTask<E>`: Funktion mit Rückgabewert (wird z.B. vom Über-Task mit `join()` geholt und in Ergebnis eingebaut)

Achtung: *Fehler im Buch: RecursiveAction ist kein generischer Typ (`extends ForkJoinTask<Void>`).*



Bildquelle: eigene Darstellung

Folie mit Anmerkungen

Ausführungsmodell

	Call from non-fork/join clients	Call from within fork/join computations
Arrange async execution	`execute(ForkJoinTask)`	`ForkJoinTask.fork()`
Await and obtain result	`invoke(ForkJoinTask)`	`ForkJoinTask.invoke()`
Arrange exec and obtain Future	`submit(ForkJoinTask)`	`ForkJoinTask.fork()` (`ForkJoinTasks` *are* Futures)

Bildquelle: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ForkJoinPool.html

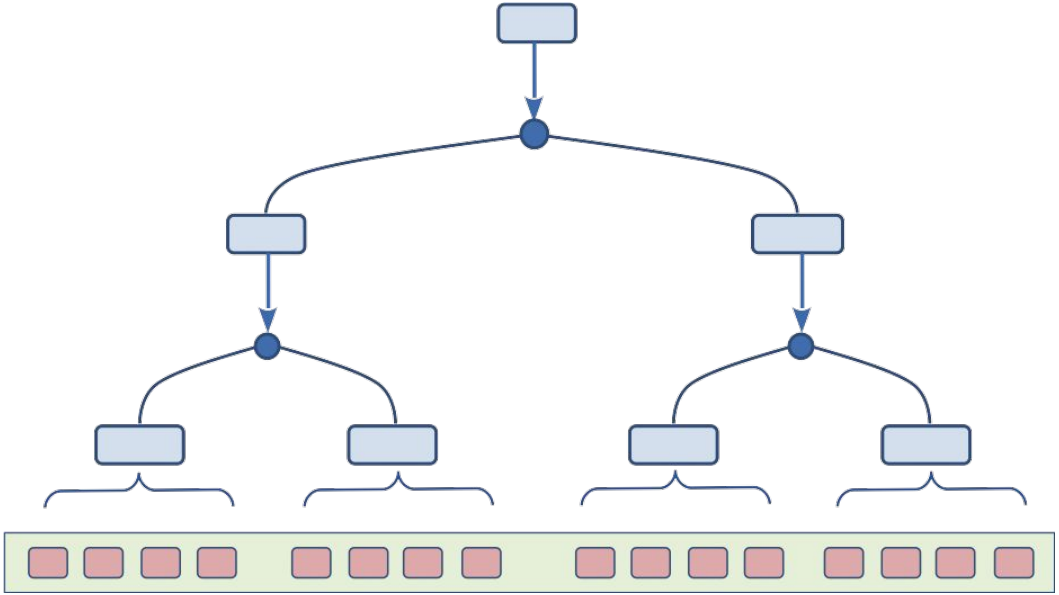
Kleine Beispiele

Array-Verarbeitung (RecursiveAction)

Das ForkJoin-Framework eignet sich besonders gut zur Implementierung von Array-Algorithmen:
z.B. *in-place Filter*: Alle Werte eines Arrays mit `MAX` überschreiben, die `>` `MAX` sind.

1) initialisieren (16 Elemente)
2) Array halbieren, bis <= 4 Elementen
3) Filtern

Wie oft *fork*? Und bei 32 Elementen?



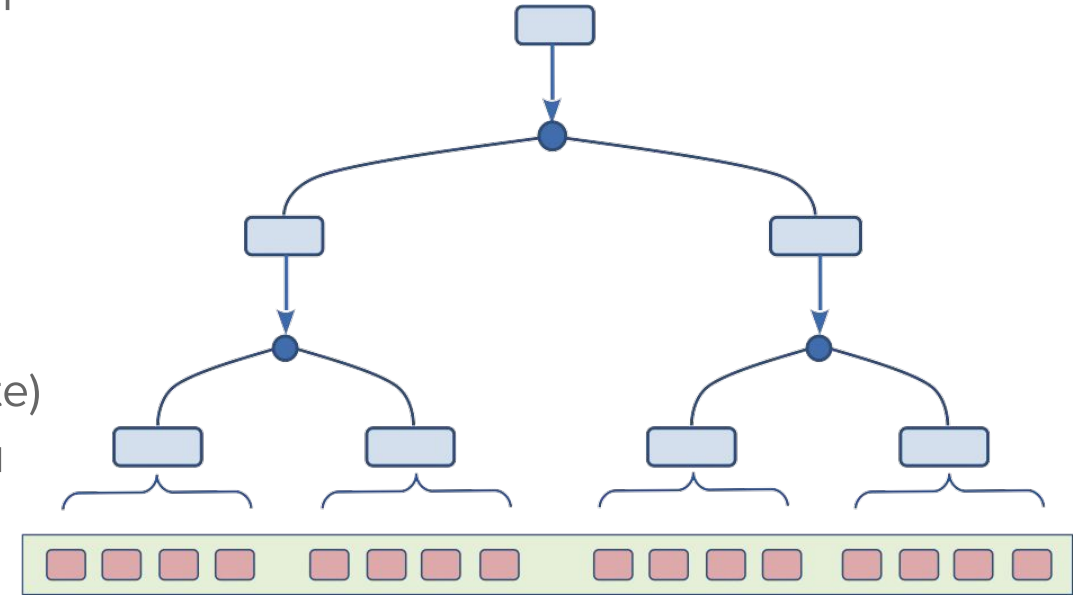
Bildquelle: Hettel & Tran 2016

Array-Verarbeitung (RecursiveAction)

Das ForkJoin-Framework eignet sich besonders gut zur Implementierung von Array-Algorithmen:
z.B. *in-place Filter*: Alle Werte eines Arrays mit `MAX` überschreiben, die `>` `MAX` sind.

1) initialisieren (ARRAY_LEN Elemente)
2) Array halbieren, bis <= SLICE_LEN Elementen
3) Filtern

Wie oft *fork*? Und bei 32 Elementen?



Bildquelle: Hettel & Tran 2016

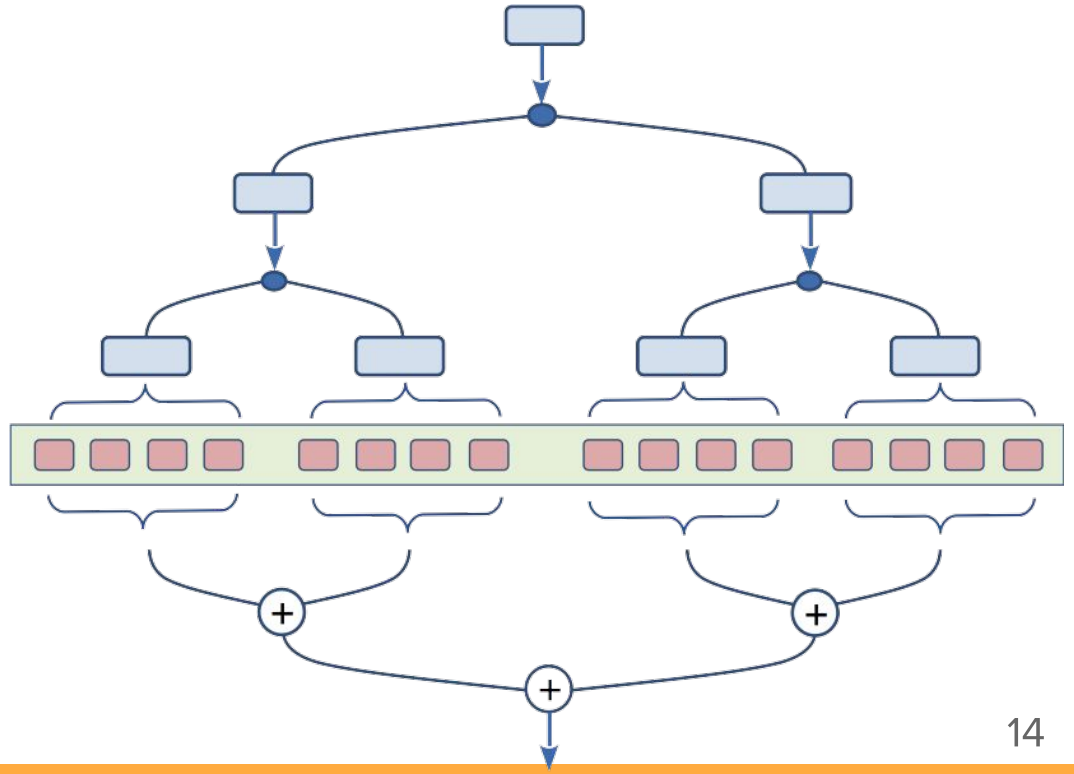
Array-Verarbeitung (RecursiveTask)

Summenbildung (*Reducer)*: Alle Werte eines Arrays summieren.

1) initialisieren (`ARRAY_LEN` Elemente)
2) Array halbieren, bis <= `SLICE_LEN` Elem.
3) summieren (Hälften oder <= `SLICE_LEN` Elemente)

Laufzeituntersuchung:

- UV: `SLICE_LEN`, `ARRAY_LEN`
- AV: Dauer in ms



Bildquelle: Hettel & Tran 2016