Parallele Programmierung

Evolution of Concurrency Support in Java
Intro to Stream Processing (Demo)

Hermann Lehner

1996: Threads, Locks, Synchronized Blocks

- Java 1.0 Release
- Java threads are mapped to heavy weight OS threads
- Each object in Java can act as a reentrant lock
- “Critical Sections” with synchronized blocks

- Concurrency on the lowest level, comparable to assembler programming. Very difficult to get right. Completely manual control.

- Manual fork/join of threads compares to using ‘goto’ statements in imperative programming

2004: Concurrency Framework (Executors…)

- Java 5.0 Release
- On top of the existing low-level concurrency tools (threads, locks…)
  - Executor service to conveniently run tasks concurrently (and potentially parallel)
  - Explicit Lock classes (Reentrant Locks, Read-Write Locks, …)
  - Toolkit of atomic operations (AtomicInteger, …)
- Much easier to write concurrent code
- System takes over the job of distributing work packages to available threads.
- Doesn’t allow Divide and Conquer Parallelism

2011: ForkJoin, More Data Structures

- Java 7 Release
- Finally, good support for Divide & Conquer Parallelism
- Decoupling from Java Threads. ForkJoin Tasks are very lightweight. Usage of available CPU is optimized by framework (after warm-up)
- ThreadLocalRandom allows to efficiently generate random numbers per thread (otherwise bottleneck)

- ForkJoin is still not easy to use, a lot of manual management of decomposition into tasks and collecting results

2014ff: Java Stream API,

- Huge step towards parallelizing computation
- Take away the burden on manually managing the decomposition into tasks and collecting results.
- Functional style programming
- CompletableFuture (async programming)

- If it works and performs, fine. Otherwise: A lot of magic happens behind the scenes, very difficult to debug

Running Example: Sum of Elements in List

→ IDE

Stream API Introduction

The Oracle Documentation provides a nice introduction to Streams!

Also, modern IDES can automatically replace wordy train wrecks in elegant lambdas/functional style.

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/stream/package-summary.html

Exploring the Java Stream API (+ Lambdas)

→ IDE