The Java Fork-Join Pool Framework

(Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand how the Java fork-join framework processes tasks in parallel
- Recognize the structure & functionality of the fork-join framework
The Structure & Functionality of the Fork-Join Framework
ForkJoinPool implements the ExecutorService interface

```java
public class ForkJoinPool
extends AbstractExecutorService

An ExecutorService for running ForkJoinTasks. A ForkJoinPool provides the entry point for submissions from non-ForkJoinTask clients, as well as management and monitoring operations.

A ForkJoinPool differs from other kinds of ExecutorService mainly by virtue of employing work-stealing: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when most tasks spawn other subtasks (as do most ForkJoinTasks), as well as when many small tasks are submitted to the pool from external clients. Especially when setting asyncMode to true in constructors, ForkJoinPools may also be appropriate for use with event-style tasks that are never joined.

A static commonPool() is available and appropriate for most applications. The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool. Using the common pool normally reduces resource usage (its threads are slowly reclaimed during periods of non-use, and reinstated upon subsequent use).
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html)
The Structure & Functionality of the Fork-Join Framework

• ForkJoinPool implements the ExecutorService interface
  • This interface is the basis for Java Executor framework subclasses

See docs.oracle.com/javase/tutorial/essential/concurrency/executors.html
ForkJoinPool implements the ExecutorService interface

- This interface is the basis for Java Executor framework subclasses

- Other implementations of Executor Service execute runnables or callables
• ForkJoinPool implements the ExecutorService interface
  • This interface is the basis for Java Executor framework subclasses
  • Other implementations of Executor Service execute runnables or callables
• In contrast, the ForkJoinPool executes ForkJoinTasks
The Structure & Functionality of the Fork-Join Framework

- ForkJoinPool implements the ExecutorService interface
  - This interface is the basis for Java Executor framework subclasses
- Other implementations of Executor Service execute runnables or callables
- In contrast, the ForkJoinPool executes ForkJoinTasks

It can also execute runnables & callables, but that’s not its main purpose
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask associates a chunk of data along with a computation on that data.

```java
public abstract class ForkJoinTask<V>
    extends Object
    implements Future<V>, Serializable

Abstract base class for tasks that run within a ForkJoinPool. A ForkJoinTask is a thread-like entity that is much lighter weight than a normal thread. Huge numbers of tasks and subtasks may be hosted by a small number of actual threads in a ForkJoinPool, at the price of some usage limitations.

A "main" ForkJoinTask begins execution when it is explicitly submitted to a ForkJoinPool, or, if not already engaged in a ForkJoin computation, commenced in the ForkJoinPool.commonPool() via fork(), invoke(), or related methods. Once started, it will usually in turn start other subtasks. As indicated by the name of this class, many programs using ForkJoinTask employ only methods fork() and join(), or derivatives such as invokeAll. However, this class also provides a number of other methods that can come into play in advanced usages, as well as extension mechanics that allow support of new forms of fork/join processing.
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask associates a chunk of data along with a computation on that data
- This enables fine-grained data parallelism

See [www.dre.Vanderbilt.edu/~schmidt/PDF/DataParallelismInJava.pdf](http://www.dre.Vanderbilt.edu/~schmidt/PDF/DataParallelismInJava.pdf)

```java
public abstract class ForkJoinTask<V>
extends Object
implements Future<V>, Serializable

Abstract base class for tasks that run within a ForkJoinPool. A ForkJoinTask is a thread-like entity that is much lighter weight than a normal thread. Huge numbers of tasks and subtasks may be hosted by a small number of actual threads in a ForkJoinPool, at the price of some usage limitations.

A "main" ForkJoinTask begins execution when it is explicitly submitted to a ForkJoinPool, or, if not already engaged in a ForkJoin computation, commenced in the ForkJoinPool.commonPool() via fork(), invoke(), or related methods. Once started, it will usually in turn start other subtasks. As indicated by the name of this class, many programs using ForkJoinTask employ only methods fork() and join(), or derivatives such as invokeAll. However, this class also provides a number of other methods that can come into play in advanced usages, as well as extension mechanics that allow support of new forms of fork/join processing.
```
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask is lighter weight than a Java thread

  *e.g., it doesn’t maintain its own run-time stack*
A ForkJoinTask is lighter weight than a Java thread
A large # of ForkJoinTasks can thus run in a small # of worker threads in a fork-join pool
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask is lighter weight than a Java thread
- A large # of ForkJoinTasks can thus run in a small # of worker threads in a fork-join pool

See blog.jamesdbloom.com/JVMInternals.html
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask has two methods that control parallel processing/merging:

  
  **ForkJoinTask**
  
  fork() – Arranges to asynchronously execute this task in the appropriate pool
  
  join() – Returns result of computation when it is done

**Parent ForkJoinTask**

join()  

fork()  

fork()

**Child ForkJoinTasks**

join()  

join()
A ForkJoinTask has two methods that control parallel processing/merging:

- **fork()** - Arranges to asynchronously execute this task in the appropriate pool.
- **join()** - Returns result of computation when it is done.

ForkJoinTask<T> fork() – Arranges to asynchronously execute this task in the appropriate pool.

V join() – Returns result of computation when it is done.

fork() is akin to a lightweight version of Thread.start()
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask has two methods that control parallel processing/merging:
  - `fork()` does not run the task immediately, but instead places it on a work queue.
  - `join()` returns the result of the computation when it is done.

<table>
<thead>
<tr>
<th>ForkJoinTask &lt;T&gt;</th>
<th>fork() – Arranges to asynchronously execute this task in the appropriate pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>join() – Returns result of computation when it is done</td>
</tr>
</tbody>
</table>

The diagram illustrates the parent and child ForkJoinTasks, with `fork()` methods arranging asynchronous execution and `join()` methods waiting for results.
A ForkJoinTask has two methods that control parallel processing/merging:

- **ForkJoinTask**
  - `fork()` – Arranges to asynchronously execute this task in the appropriate pool
  - `join()` – Returns result of computation when it is done

**Diagram:**

- **Parent ForkJoinTask**
  - `join()`
  - `fork()`
- **Child ForkJoinTasks**
  - `fork()`
  - `join()`
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask has two methods that control parallel processing/merging:
  - `fork()` - Arranges to asynchronously execute this task in the appropriate pool
  - `join()` - Returns result of computation when it is done

- Unlike `Thread.join()`, `ForkJoinTask.join()` doesn’t simply block the calling thread
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask has two methods that control parallel processing/merging:
  - `fork()` – Arranges to asynchronously execute this task in the appropriate pool
  - `join()` – Returns result of computation when it is done

- Unlike `Thread.join()`, `ForkJoinTask.join()` doesn’t simply block the calling thread.
- It uses a worker thread to run tasks.

“Collaborative Jiffy Lube” model of processing!
The Structure & Functionality of the Fork-Join Framework

- A ForkJoinTask has two methods that control parallel processing/merging:
  - `fork()` – Arranges to asynchronously execute this task in the appropriate pool
  - `join()` – Returns result of computation when it is done

<table>
<thead>
<tr>
<th>ForkJoinTask <code>&lt;T&gt;</code></th>
<th>fork()</th>
<th>A ForkJoinTask has two methods that control parallel processing/merging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent ForkJoinTask</td>
<td>fork()</td>
<td>Parent ForkJoinTask</td>
</tr>
<tr>
<td>Child ForkJoinTasks</td>
<td>join()</td>
<td>Child ForkJoinTasks</td>
</tr>
<tr>
<td>join()</td>
<td>join()</td>
<td>join()</td>
</tr>
</tbody>
</table>

- Unlike `Thread.join()`, `ForkJoinTask.join()` doesn’t simply block the calling thread
- It uses a worker thread to run tasks
- When a worker thread encounters a `join()` it processes other tasks until it notices the target sub-task is done
The Structure & Functionality of the Fork-Join Framework

- Programs don’t use ForkJoinTask directly
The Structure & Functionality of the Fork-Join Framework

- Programs don’t use ForkJoinTask directly... but instead extend a subclass & override its compute() hook method.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-tree.html
The Structure & Functionality of the Fork-Join Framework

- Programs don’t use `ForkJoinTask` directly... but instead extend a subclass & override its `compute()` hook method, e.g.

  **RecursiveAction**
  
  - Use for computations that do not return results

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/RecursiveAction.html](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RecursiveAction.html)
The Structure & Functionality of the Fork-Join Framework

- Programs don’t use ForkJoinTask directly…but instead extend a subclass & override its compute() hook method, e.g.
  - RecursiveAction
  - RecursiveTask
    - Use for computations that do return results

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RecursiveTask.html
Programs don’t use ForkJoinTask directly… but instead extend a subclass & override its compute() hook method, e.g.

- RecursiveAction
- RecursiveTask
- CountedCompleter
  - Used for computations in which completed actions trigger other actions

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CountedCompleter.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CountedCompleter.html)
Programs don’t use ForkJoinTask directly... but instead extend a subclass & override its compute() hook method, e.g.

- **RecursiveAction**
- **RecursiveTask**
- **CountedCompleter**

These classes aren’t functional interfaces, so lambda expressions can’t be used.

The Java 8 parallel streams framework provides a functional API to the ForkJoinPool.
The Structure & Functionality of the Fork-Join Framework

- ForkJoinPool enables non-ForkJoinTask clients to process ForkJoinTasks

<table>
<thead>
<tr>
<th>void</th>
<th><code>execute(ForkJoinTask&lt;T&gt;)</code> – Arrange async execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td><code>invoke(ForkJoinTask&lt;T&gt;)</code> – Performs the given task, returning its result upon completion</td>
</tr>
<tr>
<td>ForkJoinTask&lt;T&gt;</td>
<td><code>submit(ForkJoinTask&lt;T&gt;)</code> – Submits a ForkJoinTask for execution, returns a future</td>
</tr>
</tbody>
</table>

We’ll discuss these methods later in part 3 of this lesson.
The Structure & Functionality of the Fork-Join Framework

- ForkJoinPool enables non-ForkJoinTask clients to process ForkJoinTasks
- Clients insert new tasks onto a shared queued used to feed “work-stealing” queues managed by worker threads

See en.wikipedia.org/wiki/Work_stealing
The Structure & Functionality of the Fork-Join Framework

- ForkJoinPool enables non-ForkJoinTask clients to process ForkJoinTasks
  - Clients insert new tasks onto a shared queued used to feed “work-stealing” queues managed by worker threads
  - The goal of “work-stealing” is to maximize processor core utilization

See docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html
The Structure & Functionality of the Fork-Join Framework

- There are (intentionally) few “knobs” that can control a fork-join pool

```java
<<Java Class>>
ForkJoinPool

- ForkJoinPool()
- ForkJoinPool(int)
- ForkJoinPool(int,ForkJoinWorkerThreadFactory,UncaughtExceptionHandler,boolean)
- commonPool():ForkJoinPool
- invoke(ForkJoinTask<T>)
- execute(ForkJoinTask<?>):void
- execute(Runnable):void
- submit(ForkJoinTask<T>):ForkJoinTask<T>
- submit(Callable<T>):ForkJoinTask<T>
- submit(Runnable,T):ForkJoinTask<T>
- submit(Runnable):ForkJoinTask<?>
- invokeAll(Collection<Callable<T>>):List<Future<T>>
- shutdown():void
- shutdownNow():List<Runnable>
- isTerminated():boolean
- isTerminating():boolean
- isShutdown():boolean
- awaitTermination(long,TimeUnit):boolean
```

See [www.youtube.com/watch?v=sq0MX3fHkro](http://www.youtube.com/watch?v=sq0MX3fHkro)
The Structure & Functionality of the Fork-Join Framework

- There are (intentionally) few “knobs” that can control a fork-join pool
- Contrast with the ThreadPoolExecutor framework

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html
The Structure & Functionality of the Fork-Join Framework

- There are (intentionally) few “knobs” that can control a fork-join pool
- Contrast with the ThreadPoolExecutor framework, e.g.
  - corePool size
  - maxPool size
  - workQueue
  - keepAliveTime
  - threadFactory
  - rejectedExecutionHandler

See dzone.com/articles/a-deep-dive-into-the-java-executor-service
The Structure & Functionality of the Fork-Join Framework

- There are (intentionally) few “knobs” that can control a fork-join pool
  - Contrast with the ThreadPoolExecutor framework
- However, you can configure the size of the common fork-join pool

```java
System.setProperty(
    "java.util.concurrent+ForkJoinPool.common+parallelism",
    10);
```

# of desired threads

See Part 7 of this lesson for details
End of the Java Fork-Join Pool Framework (Part 2)