# Fork/Join

The Whole GPars Team <gpars-developers@googlegroups.com>

Version 2.0, 2017-02-01

# Table of Contents

To read this topic in the PDF format, please click here.

**Fork/Join** or `Divide and Conquer` is a very powerful abstraction to solve hierarchical problems

# The Abstraction

When talking about hierarchical problems, think about quick sort, merge sort, file system or general tree navigation and such.

- *Fork/Join* algorithms essentially splits a problem at hand into several smaller sub-problems and recursively applies the same algorithm to each of the sub-problems.
- Once the sub-problem is small enough, it is solved directly.
- The solutions of all sub-problems are combined to solve their parent problem, which, in turn, helps solve its own parent problem.

> ❗ The **JSR-166y** library leaves a couple of rough edges, which can hurt you

---

The **JSR-166y** library solves *Fork/Join* orchestration pretty nicely for us, but leaves a couple of rough edges, which can hurt you, if you don’t pay attention enough. You still deal with threads, pools and synchronization barriers.

---

# The GPars Abstraction Convenience Layer

**GPars** can hide the complexities of dealing with threads, pools, barriers and RecursiveActions from you, yet let you leverage the powerful *Fork/Join* implementation in *jsr166y*.

*ForkJoin Sample*

```
import static groovyx.gpars.GParsPool.runForkJoin
import static groovyx.gpars.GParsPool.withPool

//feel free to experiment with the number of fork/join threads in the pool
withPool(1) {pool ->
    println """Number of files: ${
        runForkJoin(new File("./src")) {file ->
            long count = 0
            file.eachFile {
                if (it.isDirectory()) {
                    println "Forking a child task for $it"
                    forkOffChild(it)       //fork a child task
                } else {
                    count++
                }
            }
            return count + (childrenResults.sum(0))
            //use results of children tasks to calculate and store own result
        }
    }"""
}
```

# *Fork/Join* Saves Your Resources

*Fork/Join* operations can be safely run with small number of threads thanks to internally using the TaskBarrier class to synchronize the threads.

While a thread is blocked inside an algorithm waiting for its sub-problems to be calculated, the thread is silently returned to the pool to take on any of the available sub-problems from the task queue and process them. Although the algorithm creates as many tasks as there are sub-directories and tasks wait for the sub-directory tasks to complete, as few as one thread is enough to keep the computation going and eventually calculate a valid result.

If you&#8217;d like to know more, check out <a href="./guide/#<em>fork_join_2">the _Fork/Join</em> section of the User Guide</a>.