GPars - Groovy Parallel Systems

Russell Winder

Version 1.0, 2015-10-01

Table of Contents

GPars - The Groovy concurrency and parallelism framework for Groovy and Java code

The **GPars** framework offers Java developers intuitive and safe ways to handle Java or Groovy tasks concurrently. Leveraging the enormous flexibility of the Groovy programing language and building on proven Java technologies, we aim to make concurrent programming for multi-core hardware intuitive, robust and enjoyable.

withPool

```
withPool {
    def selfPortraits = images
        .findAllParallel{it.contains me}
        .collectParallel{it.resize()}
}
```

GPars is a multi-paradigm concurrency framework, offering several mutually cooperating high-level concurrency abstractions, such as *Dataflow operators*, *Promises*, *CSP*, *Actors*, *Asynchronous Functions*, *Agents* and *Parallel Collections*.

GPars - 'coz concurrency is Groovy

Why people like GPars

Hear those who are using **GPars** already, check out the User Voices.

The traditional thread-based concurrency model built into Java doesn't match well with the natural human sense for parallelism. While this was not a problem at times, when the level of parallelism in software was low and concurrency offered only limited benefits compared to sequential code, nowadays, with the number of cores on a single main-stream chip doubling almost every year, sequential code quickly looses ground and fails to compete in performance and hardware utilization with concurrent code.

Inevitably, for concurrent programming to be effective, the mental models of concurrent systems interactions that people create in their heads have to respect the nature of human brains more than the wires on the chips. Luckily, such abstractions have been around for several decades, used at universities, in telephone switches, the super-computing industry and some other inherently concurrent domains. The current challenge for **GPars** is to bring these abstractions up to the mainstream software developers to help us solve our practical daily issues.

User Guide

Please refer to the User Guide for an extensive coverage of **GPars** abstractions. You may also like a few Demos to get a taste of what's in here for you.

The framework provides straightforward Java or Groovy-based APIs to declare, which parts of the code should be performed in parallel. Collections can have their elements processed concurrently, closures can be turned into composable asynchronous functions and run in the background on your behalf, mutable data can be protected by agents or software transactional memory.

For the common scenario that one or multiple results are calculated concurrently but need to be processed as soon as they are available, **GPars** makes it a breeze to correctly model this with Dataflow. Dataflow variables and channels gives you a handy abstraction of single-assignment multiple-read data elements, while dataflow operators let you build efficient concurrent data-processing networks.

DataFlows

```
final SyncDataflowQueue channel = new SyncDataflowQueue()

def producer = task {
    (1..30).each {
        channel << it  //writing to a channel
        println "Just sent $it"
    }
}

def consumer = task {
    while (true) {
        sleep 500  //simulating a slow consumer
        final Object msg = channel.val
        println "Received $msg"
    }
}

producer.join()
```

The concept of Actors as an approach to organizing concurrent activities has recently gained new popularity (thanks to the Scala, Erlang, and other programming languages).

GPars implements this concept for Java and Groovy developers. With actors support you can quickly create several independent Actors, which consume messages passed to them and communicate with other actors by sending them messages. You then build your solution by combining these actors into a communication network.

Let the fun begin!

Main Areas

- Dataflow concurrency
- Actor programming model
- CSP
- Agent — an thread-safe reference to mutable state
- Concurrent collection processing
- Composable asynchronous functions
- Fork/Join

Project's main priorities

- Good and clean design
- Elegant Java and Groovy APIs
- Flexibility through meta-programming
- Application-level solutions that scale with number of cores

Fast Track

If you want to start experimenting with **GPars** right away, use our Fast Track to get up and running within minutes.

- Groovy Fast Track
- Java Fast Track
- Grails and Griffon Fast Track

What people say about GPars

Check out the User Voices to hear the opinions of people walking here before you.

Licencing
