Inversion in Programming Language Design [*]

The Encore Way

Dave Clarke
Tobias Wrigstad
Uppsala University

Nobuko Yoshida
Imperial College London

Frank S. de Boer
Einar Broch Johnsen
CWI / University of Oslo

Abstract

Programming language research benefits from the injection of new (and possibly ludicrous) ideas. In this extended abstract, we overview the approach taken in the design of the new parallel, object-oriented programming language Encore — inverting the defaults taken by standard (popular, modern, . . .) programming languages that are wrong in the advent of multicore.

Inversion: Concurrent-by-default

Encore programs are built from active objects by default, and the default class kind is active. Active objects, or actors, logically encapsulate a thread of control and communicate with the outside world using message sends returning futures. Thus, Encore programs are concurrent and messages can be processed in parallel in different objects. Encore uses cooperative scheduling: messages are either processed in full before control moves to another object, or an object voluntarily gives up control, for example, because it is waiting on a future result from elsewhere in the system. The key to safe concurrency by default is *isolation-by-default.*

Inversion: (Data) Parallel-by-default

Parallelising compilers work hard to extract parallelism from sequential programs. Such compilers succeed only in limited domains, relying on strict assumptions. By making (data) parallelism the default, the Encore compiler and run-time can focus on exploiting parallelism in the best way. Encore makes parallel data types—so-called `Par<T>` types—one of the default data types, built into the compiler, and supported by specialised syntax.

Inversion: Data-race-free-by-default

While data-race freedom comes with active object isolation, Encore aims to support multiple ways for concurrent tasks to share state. The isolation model of concurrent objects is bad for solving problems that involve the parallel processing of a single data structure (such as an array or matrix). Encore employs a substructural type system that allows the splitting of data structures into subparts that can be operated on in isolation and later merged. Thus, while objects are technically shared, concurrent operations involving writes are guaranteed not to be visible to anyone but the mutator.

Inversion: Asynchronous-by-default

Synchronisation can kill both the performance and scalability of parallel programs. To support asynchrony, Encore provides asynchronous method calls by default. Methods return a future and futures can be operated on asynchronously using future chaining. Futures also integrate nicely with `Par<T>` types.

[*] Partly funded by the EU project FP7-612985 UpScale: From Inherent Concurrency to Massive Parallelism through Type-based Optimisations.

Inversion: Linearity-by-default

Mainstream OOPLs allow objects to be aliased by default, despite the fact that most objects are never aliased from the heap. By adopting linearity-by-default, Encore objects enjoy several properties that come with alias-freedom: objects can be transferred to other active objects without breaking isolation; data-race freedom comes for free; and objects can be reclassified (including promoted to actors). Encore supports several ways of overriding this default: actor-local references allow aliasing but capture an object within an actor; safe references allow sharing objects across actors using concurrency control mechanisms (locks, transactions, etc.); or immutability preserves data-race freedom.

Inversion: Immutable-by-default

Parallelism has driven programming languages and frameworks towards immutable objects as immutability simplifies sharing. Immutability allows objects to be copied transparently allowing actor's heaps to be isolated and simplifying garbage collection by not requiring actors to stop in order to start collecting. Shared mutable state, on the other hand, can be more efficient, for example, in terms of memory. In Encore, default-immutability of objects can be overridden, and several safe alternatives to immutability exist, such as actor-local objects, and linear objects.

Inversion: Local-by-default

Modern mainstream object-oriented languages rely on garbage collection to manage memory. This often places memory management outside of the program's control, making memory-efficient programming difficult. While moving compacting GC and thread-local frontiers generally give good locality of reference for many problems, Encore seeks to enable programmers to better manage data locality. To that end, we leverage ownership-types like features to group objects in *object-local heaps*, where representation objects of interest are managed by a specific memory manager that understands locality and layout properties of the data structure in question. This may entail splitting objects into different pools to improve memory efficiency of certain operations, or simply rearranging objects in specific ways during garbage collection.

Inversion: Multi-object-by-default

Object-oriented designs readily express multitudinous associations between objects, but these associations are serialised into lists when implemented in object-oriented languages. Encore offers *big variables* or *bigvars* to express such connections directly by allowing multi-valued fields. Writing to such a field amounts to adding data to a concurrent collection, reading is a filter operation, but general parallel operations of bigvars come for free. Bigvars gel well with `Par<T>`types.