A Type and Effect System for Parallelization

Manuel Geffken

April 19, 2012

As general purpose multicore processors are becoming ubiquitous programmers want to make use of this latent processing power to meet the requirements of today's applications. This gives rise to a strong demand for parallel programming to exploit the application's inherent potential for task parallelism.

In parallel programming one typically tries to achieve determinism by avoiding data races. However, in the presence of shared mutable state and aliasing avoiding data races is difficult to achieve because the programmer is often unaware of the exact nature of the side effects caused by a function. Thus, inherent task parallelism is both hard to spot and utilize in practice.

A typical approach to avoiding race conditions is a type system statically enforcing the uniqueness of distinct memory regions[1] by prohibiting cross-region aliasing.

We define a type and effect system in the context of JVM-based languages that allows to reason about a method's effects on its reachable heap in the presence of subtyping, method overriding and virtual method calls without requiring a whole program analysis. We do not impose additional restrictions on the programmer beyond the definition of effect annotations. In particular, we require no explicit definition of statically checked unique memory regions.

In short, our system allows method effect annotations expressing the following properties in terms of its reachable heap:

- Access permission contracts (APCs) describe read and write access to fields of objects.
- Abstract heap effects specify how the method changes heap aliasing.
- Abstract return effects approximate the method's return value regarding the method's reachable heap.

These effect annotions use sets of access paths to identify objects on the method's reachable heap. All access paths refer to the heap at method invokation time. They start with a method's formal parameter followed by a sequence of field names. We use type-based abstraction for newly heap-allocated storage. The effect annotations are designed to be composable accross method calls. If the object trees refered to in the APCs are guaranteed to be alias-free, we can statically determine tasks with race-free data access.

However, even incomplete information from a may-alias analysis might turn out to be sufficient to guarantee data race freedom in many cases as only certain classes of aliasing regarding the access paths need to be excluded. Furthermore, the alias-analysis-friendly nature of Java-like languages and the additional information from the effect annotations can be exploited to maximize the preciseness we get out of such an analysis. Where it seems promising, gaps in the analysis result can be filled with selected runtime alias tests.

We hope that we can exploit a reasonable amount of the application's inherent potential for task parallelism by providing the type system with exhaustive information about a method's effects only, rather than also limiting the allowed aliasing between data structures, which imposes limitations on all parts of the program refering to these data structures.

[1]Robert L. Bocchino, Jr., Vikram S. Adve, Danny Dig, Sarita V. Adve, Stephen Heumann, Rakesh Komuravelli, Jeffrey Overbey, Patrick Simmons, Hyojin Sung, and Mohsen Vakilian. A type and effect system for deterministic parallel Java. In ACM SIGPLAN Conf. on Object-Oriented Prog., Systs., Langs., and Apps. (OOPSLA), pages 97-116, New York, NY, USA, 2009. ACM Press.