Foundations of Persistent Programming

Azalea Raad

Context

Storage hardware is traditionally divided into two categories: *fast*, byte-addressable memory (e.g. DRAM) that is *volatile* (it loses its contents after a crash, e.g. a power loss); and *slow*, block-addressable storage (e.g. hard disks) that is *persistent* (its contents survive crashes). However, advances in cutting-edge non-volatile memory (NVM) render this dichotomy obsolete by combining the strengths of both: NVM provides fast, byte-addressable access as with DRAM, while ensuring persistence across crashes as with hard disks.

As of 2019, NVM (a.k.a. persistent memory) has become widely available with commercial products such as Intel Optane. This has already led to a surge in NVM research in the last few years and will inevitably lead to substantial changes in software and its engineering. Specifically, to enable programming on top of NVM, major hardware vendors such as Intel and Arm have already added architectural support for NVM programming. However, given the rapid emergence of the NVM technology, high-level support for persistent programming in mainstream languages such as C/C++ is currently lacking.

Project

The aim of this project is to develop rigorous theoretical and practical foundations for the cutting-edge paradigm of persistent programming in the context of the emerging non-volatile memory technology. The project has both theoretical and practical components.

On the theoretical side, the aim is to extend the C language to support persistent primitives, to devise a formal semantic model describing their persistency behaviour, and to develop a verified compilation scheme from this C extension to existing ARMv8 and Intel-x86 architectures.

On the practical side, the aim is to develop high-level support for persistent programming through designing and implementing verified persistent libraries such as persistent transactions and data structures.

Several Relevant Publications

- The persistency semantics of the ARMv8 architecture (2019)
- The persistency semantics of the Intel-x86 architecture (2020)
- Verifying persistent programs (2020)