JFP **28**, e6, 2 pages, 2018.
doi:10.1017/S0956796818000060

Editorial for the Special Issue on Parallel and Concurrent Functional Programming

Functional languages are uniquely suited to providing programmers with a programming model for parallel and concurrent computing. This is reflected in the wide range of work that is currently underway, both on parallel and concurrent functional languages, as well as on bringing functional language features to other programming languages. This has resulted in a rapidly growing number of practical applications. The *Journal of Functional Programming* decided to dedicate a special issue to this field to showcase the state of the art in how functional languages and functional concepts currently assist programmers with the task of managing the challenges of creating parallel and concurrent systems.

The common theme of the majority of the papers that were submitted and accepted for publication is the variety of aspects of scheduling for parallel and concurrent systems.

In *Transparent Fault Tolerance for Scalable Functional Computation*, Robert Stewart, Meier and Trinder address the often neglected issue of reliability and fault tolerance in this context. To this aim, the authors extend the domain-specific language HdpH, a Haskell EDSL for parallel and distributed programming, with support for reliable scheduling. In the paper, the authors present a formal semantics of HdpH-RS, on which they base the validation of the fault-tolerant distributed scheduling algorithm using a model checker.

Sivaramakrishnan, Harris, Marlow and Jones present a concurrency substrate design for the Glasgow Haskell Compiler (GHC) in *Composable Scheduler Activations for Haskell*. These abstractions allow application programmers to write customised schedulers as ordinary libraries, without compromising any of the existing features in GHC. They show that this approach integrates seamlessly with the existing runtime system features for concurrency support and that the performance is comparable to the default scheduler in GHC.

Implicit parallel languages offer a convenient, high-level programming model, as they leave scheduling decisions to the compiler and runtime system. Deciding on the scheduling strategy in such a context is a complex problem, which is addressed by Acar, Charguéraud and Rainey in *Oracle-Guided Scheduling for Controlling Granularity in Implicitly Parallel Languages*.

In the Haskell dialect Eden, parallelism and scheduling are implemented via skeletons. In *Skeleton Composition Versus Stable Process Systems in Eden*, Dieterle, Horstmeyer, Loogen and Berthold discuss the trade-offs between programmer convenience and program efficiency when deciding whether to use complex monolithic skeletons or composable, simpler ones using a range of case studies.

In *A Language for Hierarchical Data Parallel Design-Space Exploration on GPUs*, Svensson, Newton and Sheeran present case studies on how to use Obsidian,

a Haskell EDSL for GPU programming. Obsidian supports *hierachical data parallelism*, a restricted form of nested data parallelism, and aims at allowing programmers to abstract over details of the GPU architecture, while still providing sufficient means for fine tuning to achieve performance close to hand optimised GPU programs.

As editors of the special issue, we would like to thank all the authors who responded to the call for papers. We would also like to extend our gratitude to the reviewers for their expertise, time and effort, and to Matthias Felleisen, Jeremy Gibbons and the JFP editorial office for their support.

Gabriele Keller
University of New South Wales
gabriele.keller@unsw.edu.au

Fritz Henglein
University of Copenhagen
henglein@diku.dk