Prof. Ryan Newton — SICSA Distinguished Visitor
11–20 June 2014

Ryan R. Newton, Assistant Professor at Indiana University Bloomington, visited Scotland between 11th and 20th June 2014. He was hosted by Prof. Phil Trinder and Dr. Patrick Maier at the University of Glasgow. The main purpose of the visit was to explore the potential for collaboration between Ryan Newton's research group at Indiana and the parallel Haskell research community in Scotland.

During his stay, Prof. Ryan Newton visited four different SICSA universities for a day each, engaging in discussions about parallel programming language research with academic and research staff. He also gave three seminar talks and participated in a hackathon. Details about these activities are listed below; abstracts of the talks can be found at the end of this report.

- **Design Frontiers in Parallel Languages: The Role of Determinism.** Seminar at the University of St Andrews, 12th June, hosted by Dr. Edwin Brady.
- **Evolving the Parallel Haskell Ecosystem.** Hackathon at Heriot-Watt University, 16th June, hosted by Dr. Hans-Wolfgang Loidl.
- **Stream-processing for Functional Programmers.** Keynote at the Scottish Programming Languages Seminar, 18th June, at the University of Glasgow.
- **Design Frontiers in Parallel Languages: The Role of Determinism.** Seminar at the University of Edinburgh, 20th June, hosted by Prof. Murray Cole.

The hackathon on 16th June, attracting 9 participants from Heriot-Watt and Glasgow, was a unique opportunity to discuss in depth future collaboration on parallel programming language implementation and infrastructure. Among the topics discussed were the following; further topics may be gleaned from the whiteboard photos[1] on the hackathon web page.

- A common abstract API for parallel Haskell DSLs, with the ultimate goal of building composable parallel DSLs.
- A library of parallel data structures for Haskell, similar to parallel collections in Scala.
- Sharing of benchmarks between systems. This effort aligns with the upcoming 3rd phase of the SICSA multicore challenge.
- Infrastructure required for nightly regression benchmark runs, and sharing the necessary software stack.
- Techniques to implement/improve distributed work stealing.

Overall, this visit was very productive, and will shape the work carried out in the parallel Haskell research groups at Glasgow and Heriot-Watt. The discussed exchanges of APIs, libraries and benchmarks offer concrete opportunities for collaboration, and the upcoming 3rd phase of the multicore challenge provides a good opportunity to compare the results.

[1] `http://www.dcs.gla.ac.uk/~pmaier/Hackathon-2014-June-16/`

Design Frontiers in Parallel Languages: The Role of Determinism (seminar)

Constraints can be a source of inspiration; their role in creative art forms is well-recognized, with poetry as the quintessential example. We argue that the requirement of determinism can play the same role in the design of parallel programming languages. This talk describes a series of design explorations that begin with determinism as the constraint, introduce the concept of monotonically-changing concurrent data structures (LVars), and end in some interesting places — flirting with the boundaries to yield quasideterminism, and revealing synergies between parallel effects, such as cancelation and memoization, when used in a deterministic context.

Our goal is for guaranteed-deterministic parallel programming to be practical and efficient for a wide range of applications. One challenge is simply to integrate the known forms of deterministic-by-construction parallelism, which we overview in this talk: Kahn process networks, pure data-parallelism, single assignment languages, functional programming, and type-effect systems that enforce limited access to state by threads. My group, together with many others around the world, are developing libraries such as LVish and Accelerate that add these capabilities to the programming language Haskell. It is early days yet, but it is already possible to build programs that mix concurrent, lock-free data structures, blocking data-flow, callbacks, and GPU-based data-parallelism, without ever compromising determinism or referential transparency.

Stream-processing for Functional Programmers (seminar)

Functional programming and stream-processing have shared history — from early work on dataflow architectures, VAL, and SISAL, to Haskell's use of stream-based IO (before monads) or the modern-day resurgence of Haskell stream libraries (iteratees, pipes, conduit). These days, "streaming" can mean a lot of things; StreamIt, based on synchronous-dataflow, has totally ordered streams and will not duplicate stateful stream processors, whereas Apache Storm makes the opposite decisions. The first part of this talk will overview this broad landscape.

We argue that the degree of dynamism (e.g. in data-rates and stream topologies) is the major axis along which various stream technologies are differentiated. In the second part of this talk, we describe our past and ongoing work on navigating this spectrum, by developing technologies that leverage regularities where they occur, but tolerate dynamism. We have studied profile-driven program partitioning, and other compilation topics, and our current thrust for developing stream DSLs overlaps heavily with work on data-parallel DSLs (e.g. Accelerate).