CAMBRIDGE

Commercial Uses of Functional Programming Scribe Report

SCHOLARONE™
Manuscripts

1

Commercial Users of Functional Programming Scribe Report

ANIL MADHAVAPEDDY

Computer Laboratory, University of Cambridge

15 JJ Thomson Avenue, Cambridge CB3 0FD, UK

YARON MINSKY,

Jane Street Capital

1 New York Plaza, New York NY, USA

MARIUS ERIKSEN,

Twitter, Inc.

795 Folsom St., Suite 600

San Francisco, CA 94107, USA.

1 Overview

Commercial Users of Functional Programming (CUFP) is a yearly workshop that is aimed at the community of software developers who use functional programming in real-world settings. This scribe report covers the talks that were delivered in the 2011 workshop, which was held in association with ICFP in Tokyo. The goal of the report is to give the reader a rough sense of what went on, rather than to reproduce the full details of the talks. Videos and slides from all the talks are available online at `http://cufp.org`.

2 Keynote: Pragmatic Haskell

Lennart Augustsson from Standard Charter gave the keynote address, relating his long-time use of Haskell and Haskell-derived languages in commercial settings. Augustsson's experience with Haskell dates back to its inception: he authored the first Haskell compiler, `hbc`, which remains competitive with the Glasgow Haskell Compiler to this day. Lennart subsequently developed several Haskell variants that were tailored for his needs.

The first of these was pH, the parallel Haskell compiler (Aditya *et al.*, 1995). pH sought to exploit the implicit parallelism in a Haskell program by combining the Haskell syntax and type system with the evaluation strategy of the `id` parallel programming language (Arvind *et al.*, 1978). pH introduced several concepts which are still used in modern systems, such as the `MVar` abstraction.

Lennart then brought his Haskell experience over to the field of hardware design. The Bluespec hardware description language is a full compile-time implementation of Haskell, outputting Verilog for hardware synthesis. Bluespec significantly raised the level of abstraction for hardware design, and is still available commercially today (Rishiyur & Arvind, 2008). An interesting aspect of Bluespec was the incorporation of numbers and arithmetic in its type system.

2   *Anil Madhavapeddy, Yaron Minsky and Marius Eriksen*

Lennart is currently working in the banking industry, providing in-house technology for traders and quants. Mu is Lennart's latest Haskell dialect designed specifically for this use. Most traders and quants are chiefly interested in data transformation and processing, and do not necessarily care about effectful operations. Mu provides a simplified dialect of Haskell catering to these needs. Interestingly, Mu eschews Haskell's lazyness by being a strict language. A number of pragmatic design decisions were made: most performance sensitive code reuses C++ from in-house libraries, and strings in Mu are not lists of characters. Furthermore, recursion is provided only optionally (and only 6% of their modules turn this on). Mu provides a fine-grained IO monad, allowing for both "IO" and "O" (output only). In order to attain easy interprocess communication, all values in Mu are serializable.

Mu is a true Haskell dialect in that code written in Mu may be compiled with a Haskell compiler. As with its language and libraries, the Mu compiler is a work of pragmatic design. All compiler transformations are done assuming it is operating on terminating code, and the compiler uses LLVM (Lattner & Adve, 2004) for its backend.

Standard Chartered's Mu codebase is of significant size and lives within a library for quants written in a combination of C++, Haskell and Mu itself. Language interoperability is key; all parts of the system can easily be invoked from Excel, C#, Java or any other component.

Their experience with strict semantics has been positive. Particularly useful is the ease of obtaining meaningful stack traces, tracking resource usage, debugging and exception propagation. The chief downside of strict semantics, in their experience, is the increased difficulty of modular composition.

Lennart noted that Excel has a more expressive effect system than Haskell and provides an effective front-end to composing Mu modules. He demonstrated coding Mu within Excel live on stage, via an interactive and interpreted version.

The audience asked about integration between the Bluespec hardware description language and Mu, since both require code that statically terminates. However, Lennart replied that "if you want to design hardware you need to have some clue about what you are doing".

3 Cryptol: Theorem-Based Derivation of an AES Implementation

John Launchbury from Galois then described Cryptol, their declarative specification language for cryptographic protocols (Erkök & Matthews, 2008). Galois is now using Cryptol to derive a highly efficient implementation of AES targeted at FPGAs.

Cryptol is a first-order functional language with size-type declarations. Recursion is available via stream equations. The only sequentialization comes from data dependencies, making it possible to efficiently evaluate this language on FPGAs. Cryptography is also a natural match for FPGAs, and Cryptol makes it easier to experiment with several implementations. Cryptol has a `theorem` keyword to express properties such as the inverse relationship between encryption and decryption operations. A tool similar to QuickCheck (Claessen & Hughes, 2000) is used to generate test cases to ensure that these theorems hold.

The derivation of the design was done be a series of stepwise refinements to the original specification, all of which were written in Cryptol. The theorem-based testing system was used as a way of gaining confidence that each stage in the refinement was solid. The end re-

Commercial Users of Functional Programming 2011   3

sult was a chip that was competitive with top-notch manually constructed implementations, but that the designer could have far more confidence in.

4 Erlang: Large-Scale Discrete Event Simulation

Olivier Boudeville from EDF described the use of Erlang for building large-scale simulations. Sim Diasca (Simulation of Discrete Systems of All Scales) is a system implemented in Erlang for building large scale discrete simulations of the kind that are used for simulating Smart Energy Grids and other large-scale industry projects.

Sim Diasca was—as its name implies—designed to meet extraordinary scalability requirements, supporting upwards of 50 million parallel instances of complex models. It prescribes no fixed topology and uses Erlang's actor mechanism for communication between nodes. The main role of the generic engine is scheduling. This scheduler enforces causality, reproducibility of the model simulation and some forms of ergodicity.

As typical language choices for this domain are C++ and CORBA, the authors wrote a macro package, "Wooper", in order to aid development and provide a more familiar environment by providing object-oriented primitives on top of Erlang.

Boudeville characterized their choice of Erlang and functional programming as "massively positive". In particular, Erlang's support for distributing computation coupled with more suitable language constructs made complex algorithms significantly easier to express. The team did encounter resistance making such an esoteric language choice. In particular, hiring developers was harder.

5 Erlang: Testing Safety Critical Automotive Components

The next talk by Thomas Arts, who is CTO of Quviq, a company he co-founded with John Hughes. Quviq is devoted to doing automated testing of software using QuickCheck and Erlang. The talk covered the application of Erlang and QuickCheck to the testing of automotive components. The average modern car has over 64 computers that are networked ("the brakes might need to know how fast the car is going!"). Manufacturers use components from a variety of vendors and so interoperability testing takes on critical importance.

The industry standard for automotive components is AUTOSAR.[1] Initially, the AUTOSAR consortium outsourced its testing, and over thirty person-years were spent on writing manual tests for component interoperability. The result was disastrous and unsuitable for use. The chief reason for this is that AUTOSAR is highly configurable, with thousands of standardised parameters that are difficult to express manually. Furthermore, this means that meaningful tests must be parameterized on the configurations.

The solution came in the form of Erlang and QuickCheck. Quiviq's AUTOSAR testing produced far better test coverage, tailored the the manufacturer's particular configuration. Furthermore, the terseness of Erlang and QuickCheck reduced the code size of the tests by at least an order of magnitude.

---

[1] http://www.autosar.org/

4 *Anil Madhavapeddy, Yaron Minsky and Marius Eriksen*

6 OCaml: Mobile HTML5 Development

We moved on from testing and simulation to the world of functional web development in Japan. Keigo Imai delivered an entertaining talk about the consultancy (IPTL) he works for, where 45% of their annual sales are from functional programming projects.

Their typical sales story goes like this: customers specify their choice of programming language and an impossibly tight deadline. IPTL then propose using their existing OCaml codebase to deliver the solution, but within the required deadline and with much less risk. IPTL continually educate their customers about the benefits of rapid development using functional languages.

Keigo demonstrated an example project: a foreign exchange chart application that works on iPhone and Android, dynamically drawn using an HTML5 canvas. It was written in OCaml and translated to Javascript using the OCamlJS compiler.[2] In their experience, static typing was very helpful for web development. In particular typing of DOM elements avoided many runtime errors, especially in the complex `<canvas>` tag. Higher level language features also facilitate the asynchronous programming style prevalent in web apps. Their Web SQL database API is wrapped in a monad and bound using continuation passing style, allowing for ease of programming without having to create explicit callbacks chains. OcamlJS allows for inlining Javascript, which was useful when more low-level control was required.

Performance has been fine with OCamlJS, and no bugs have been reported yet on its output (to quote Keigo, "it is written in our miraculous super OCaml technique!"). An audience member asked if codesize (from the generated Javascript) is a problem, especially in view of this being a mobile project. Keigo replied that they haven't had any problems with this.

7 Scala: Large Scale Internet Services at Twitter

Twitter is a social network for sharing short text messages. It is growing in popularity extremely fast, and some backend components have to serve in excess of 10^6 queries per second. Steve Jenson and Wilhelm Bierbaum described how many of Twitter's infrastructure components are implemented in Scala. The JVM is the preferred virtual machine due to its maturity and performance, and Scala provides a much better type system and functional programming features than code in Java directly.

All Twitter client HTTP requests are answered by a reverse proxy called TFE, which routes requests using the Finagle distributed RPC system.[3] Request streams are passed through filters that transform and apply processing functions on them. The process is afforded the full facilities of the JVM, including the use of pre-existing libraries written in Java.

The use of the JVM means that many existing debugging and profiling tools (e.g. Yourkit, JStat, VisualVM) can be used, but the name mangling and anonymous functions in Scala occasionally introduce obscure results. The use of immutable values result in a lot of

---

[2] `https://github.com/jaked/ocamljs`
[3] `http://twitter.github.com/finagle/`

Commercial Users of Functional Programming 2011   5

pressure on the garbage collector. When developing the high-volume interactive services, it is not uncommon to spend a significant amount of time understanding and tuning garbage collection.

Twitter has also been hiring many programmers who have never used Scala before. While many of them are familiar with the concepts underlying functional programming, the syntax of Scala can often be subtle, and the use of implicits are often difficult for beginners.

8 F#: Mobile Applications using WebSharper

Adam Granicz from Intellifactory began by pointing out that the market for mobile applications is massive. It is projected that in 2011, the phone market will have grown to 3.7 billion users, with 18% of them owning smartphones capable of running applications. However, smartphone platforms are a heterogenous bunch: The current major platforms (Android, iPhone, Windows Mobile) differ in both programming style and language choice.

Ideally, mobile applications could be developed using higher level abstractions, support compilations to multiple targets, and make use of desktop and cloud resources when available. The switch to proprietary and divergent APIs on different devices is a step backwards.

Javascript is becoming the intermediate language that connects these devices together, and Windows 8 is even promoting it for building native desktop applications. Intellifactory built the WebSharper programming framework in F# that outputs Javascript that works with all of these diverse devices.

With WebSharper, all of the server and client code is written in F#, and compiles to a complete standalone web application. The F# interfaces make good use of the language's facilities: Type-safe URLs helps prevent common errors, and "sitelets" and "formlets" that are composable abstractions for fragments of websites.

9 Haskell: A Real Time Programming Project in Real Time

Gregory Wright from Alcatel-Lucent Bell Labs described a project that used Haskell to build the core of a real-time communication system. It was built by the GreenTouch consortium, an organization of equipment vendors, service providers, and research institutes to show a new antenna technology that reduces the energy used by wireless networks. The goal of the project was to demonstrate an algorithm that reduces radio transmission power as the number of base station antenna elements is increased, and is capable of scaling to antennas with thousands of elements.

An antenna can focus on a user and calibrate the amount of data sent over the link. As a result of this, the power used to go to an individual handset can be adjusted with respect to a target rate. This was initially simulated, and then later run on real hardware as a soft realtime system.

In total, the project lasted 14 real days. As a result of the timescale, it used very vanilla Haskell (e.g. arrays weren't unboxed), with no voodoo to improve memory behaviour, or strictness annotations. The STM library was used, as well as the DSP library from Hackage.

6   *Anil Madhavapeddy, Yaron Minsky and Marius Eriksen*

Despite this simplicity, the project was a big success. An important factor was that they always had a working system, with upgrades staged well. Haskell provided a high degree of safety from "crashing and burning".

Furthermore, the nature of the project was iteself very compatible with the core ideas of Haskell. The core algorithms implemented were all "dataflow" like, using lazyness quite effectively.

10 Erlang/OCaml: Disco, a MapReduce Platform

Prashanth Mundkur from Nokia Research in Palo Alto talked about the Disco Project. When Nokia started evaluating systems for distributed data processing, the most popular solution was Hadoop. However, this was a "massive pile of Java software" that was very complex to configure and operate. An Erlang hacker at Nokia wondered how hard it would be to implement a simpler alternative.

The first Disco prototype took a few weeks to implement. The core coordination components are implemented in Erlang making use of its runtime facilities for distributed computation. Python is the primary language used to write the data processing scripts (i.e. the "mappers" and "reducers"). OCaml has recently been added as a more strongly typed data processing alternative language.

The Erlang environment provides quite a few useful tools that made Disco easier to implement. In particular, a shell to invoke remote procedure calls on hosts in the cluster and `fprof` to collect profiling information in real-time. The dynamic typing did result in some hard to find bugs, and the `dialyzer` static analysis tool is now used extensively on the codebase (but is not mandatory) (Sagonas, 2007).

Disco is open-source software, and available from `http://discoproject.org`.

11 mzScheme: Functional DSLs for Game Development

Dan Liebgold from Naughty Dog Software in Santa Monica then came on stage with the first gaming related talk at CUFP. They produce the popular Uncharted game series for the Playstation, which is famous for its complex and interactive scripted scenes. Dan described modern game development as a major production effort where, roughly, artists produce data and programmers produce code.

Naughty Dog has a history of using various Lisp dialects to handle the code and data in a unified way. But when making the jump from the Playstation 2 to the Playstation 3, they decided that maintaining a custom Lisp-based game development system was too costly, and instead dedicated their efforts to rebuilding the tools, engine, and game in C++ and assembly language.

This decision left no scripting system for gameplay and, more importantly, no system for creating DSLs and the oodles of glue data that is typically required to develop a major video game. There was no off-the-shelf scripting system that fit the stringent memory requirements in a Playstation 3, and no language that would allow rapid DSL creation that fit into the existing tool chain.

With a bit of naivety, a penchant for the Scheme language, and a passion for functional programming techniques, the team dove in and put together a system to fill in the gaps! Dan

reported that the results have been very good, but not without issues. In particular, garbage collector performance sometime led to manual tuning being required, and build environment integration was tricky. Syntax transformations and error reporting led to confusion with new programmers too.

On the other hand, the functional nature of the system was a big win, as it allowed them to flexibly distill game data down to just the right form to embed into the resource-constrained run-time environment. The final result is a system where programmers, artists, animators, and designers are productively programming directly in an S-expression Scheme-like language. Dan closed his talk by wowing the audience with the trailer for the game, which has now been released and is garnering extremely positive reviews.

12 OCaml: The Acunu Storage Platform

Tom Wilkie from Acunu in London presented their Castle storage system to optimise big data storage and retrieval. Castle has two halves: a management and storage stack. The management stack is written in OCaml and handles control requests (eg. snapshots, backups, rollbacks, coordination). The storage stack implements an efficient disk-backed indexed key-value store. While developing their storage stack, Acunu needed very rapid prototyping in order to explore new techniques and iterate on ideas. Given that issues in storage systems tend to surface only after a certain amount of data or request volume, the prototypes also needed to be reasonably performant.

However, although OCaml was effective for implementing the core algorithm, they spent a lot of time on the support code to serialise data to and from disk, and deal with threading. When the filesystem codebase was moved to Java, performance increased by 6x times. The audience suggested a few techniques to speed up the OCaml code, such as the Lwt threading library and Bitstring syntax extension, and increasing the number of I/O threads used. Tom acknowledged these could help, but that their availability should be promoted and documented better.

13 Conclusion

This year's CUFP workshop covered a broad set of functional languages—Scala, Haskell, Scheme, OCaml, Erlang and F# were all well represented, along with a nascent but growing interest in more formal tools such as Coq and Isabelle. We also ran a successful 2-day tutorial series on using Scala, OCaml, Haskell and F# for problem solving in scientific computing, web programming, parallel programming and cloud computing.

The diversity of industries where functional programming is represented is also encouraging. Our speakers worked in the financial industry, big data processing, safety critical systems in automobiles and energy systems, the real-time and mobile Internet, and for the first time, the gaming industry. We look forward to continuing the conference series in Copenhagen next year!

8 *Anil Madhavapeddy, Yaron Minsky and Marius Eriksen*

References

Aditya, Shail, Arvind, Maessen, Jan-Willem, Augustsson, Lennart, & Nikhil, Rishiyur S. (1995). Semantics of ph: A parallel dialect of haskell. *Pages 35–49 of: in proceedings of the haskell workshop (at fpca 1995).*

Arvind, Gostelow, K.P, & Plouffe, W.E. 1978 (Dec). *An asynchronous programming language for a large multiprocessor machine.* Tech. rept. TR114a. UC Irvine.

Claessen, Koen, & Hughes, John. (2000). Quickcheck: a lightweight tool for random testing of haskell programs. *Pages 268–279 of: Proceedings of the fifth acm sigplan international conference on functional programming.* ICFP '00. New York, NY, USA: ACM.

Erkök, Levent, & Matthews, John. (2008). Pragmatic equivalence and safety checking in Cryptol. *Pages 73–82 of: Proceedings of the 3rd workshop on programming languages meets program verification.* PLPV '09. New York, NY, USA: ACM.

Lattner, Chris, & Adve, Vikram. (2004). LLVM: A compilation framework for lifelong program analysis & transformation. *Pages 75– of: Proceedings of the international symposium on code generation and optimization: feedback-directed and runtime optimization.* CGO '04. Washington, DC, USA: IEEE Computer Society.

Rishiyur, Nikhil S., & Arvind. (2008). What is Bluespec? *Sigda newsletter*, **38**(December), 1–1.

Sagonas, Konstantinos. (2007). Detecting defects in Erlang programs using static analysis. *Pages 37–37 of: Proceedings of the 9th acm sigplan international conference on principles and practice of declarative programming.* PPDP '07. New York, NY, USA: ACM.