M.Sc. Thesis
Master of Science in Engineering

DTU Fotonik
Department of Photonics Engineering

Exploring the use of purely functional programming languages for offloading of mobile computations

Danish title: Undersøgelse af brug af rent funktionelle programmeringssprog til offloading af mobile beregninger

Christian Kjær Laustsen
s124324 @ DTU
20176018 @ KAIST

Kongens Lyngby 2017

Thesis Project Description

This thesis topic is about exploring, as the title implies, the usage of purely functional programming languages for offloading of mobile computations, with the main goal of achieving energy savings. The work will focus on an investigation of tackling some of the commonly found problems in current mobile code offloading approaches, such as a) handling state and b) knowing which parts can are offloadable and which are not (e.q. UI code is not). This will be done by looking at what purely functional programming languages bring to the table, exploring several approaches to solve the domain, along with describing their pros and cons, and finally further developing and implementing one of the approaches to show the feasibility.

The work will be done in collaboration with my supervisors Henrik Lehrmann Christiansen (DTU) and Sung-Ju Lee (KAIST) and finally an external contact from, Ivan Perez (Keera Studios[1]), which has expertise in Haskell on mobile platforms.

As such, the preliminary outline of the work will look like,

1. Related work
2. Exploring different approaches (expanded in the section 0.1)
3. Settle on the most optimal approach to continue with
4. Implementation of selected approach
5. Evaluation of the implementation both technical and practical
6. Discussion and further work based on the evaluation

A much more detailed discussion will be presented in the following section, justifying the project.

0.1 Justification

While there exist several approaches to offloading of computations from mobile devices, handling state still is one of the bigger problems that existing solutions encounter. We see this in [Cue+10] with the development of MAUI (2010), and it gets recognized again by different authors in [Ben+17] (2017), mentioning "Ever since

[1] http://keera.co.uk/

MAUI, state transfer has been recognized as the biggest challenge in the development of code offloading frameworks." This is, amongst other things, caused by the fact that so far the programming languages used to design these frameworks have all been based on the principles of allowing side-effects (I/O, networking, shared state) everywhere in the program without limitations - so-called impure code. In other words, it becomes hard to impossible to guarantee that the code you want to offload has no effects elsewhere in the system, without complex analysis of the code itself. Therefore these systems impose certain restrictions, as mentioned in [JFF13][2], where UI updates, local I/O and alterations of shared state is not allowed.

Digressing a bit, there are ways to circumvent some of the problems with shared state. [Ben+17] tries to tackle this by focusing on long-running background tasks, by basically taking over the Service class in Android. Another approach is found in [Lin14] where they introduce the idea of having a bidirectional transfer of memory (although they only did unidirectional: from the client to cloud). The former doesn't really go far enough, while the latter incurs quite a complexity overhead, so the problem space still craves a less complex solution.

Going back to the topic at hand, Haskell [Has17] separates itself here by being a *pure functional programming language*. In other words, things such I/O and other effects are modelled in the type system and you cannot by accident throw a `print` call into a normal function without indicating that it is doing I/O[3]. This means that a function that prints a string does not simply have the type `print :: String` but `print :: IO String` signifying that I/O is being performed. Changing global state is also considered a side-effect.

This means that we can know from a functions type signature if it is doing I/O or not. We can use this knowledge to know with certainty if a method can be considered for offloading or if it **must** remain on the mobile device. Furthermore, because purity implies referential transparency, we can cache the results of various function calls (in fact, GHC, the de facto Haskell compiler, does this already for certain things). It also becomes easier to transfer a functions "state", since in Haskell this corresponds directly to the functions input arguments, for pure functions.

As such, several approaches of varying complexity are up for investigation, in no particular order,

1. Since the GHC runtime is a beast of its own, one could go about trying to extend it to perform all of the necessary operations for offloading. Extra information could be added at compile time, such as annotations that the runtime could read, and then the runtime could have the profiling component in it to decide whether to offload or not, along with handling the networking needed.

[2] Found in table 1 at page 5, for the interested reader

[3] A bit of a brief explanation, more detail can be read at [Mil14]

2. Instead of relying on the runtime, which would mean an alteration of GHC, and to avoid the costly implications that has (playing catch-up with upstream changes etc.), one could find inspiration in Facebook's Haxl system[Mar+14] (or specifically ApplicativeDo), which shares many of the ideas that would help energy savings, but also describes a way to, with very little effort, make the compiler do the tedious work of optimizing for performance (in the form of parallelism) without specifically incorporating it into the logic of the application itself. Haxl itself, in brief terms, allows for easy batching, caching and optimization of calls to external systems, such as databases without the programmer making this batching explicit. As such, several approaches can be investigated.

3. Alternatively, for a more automated approach, a compiler extension could be developed, which would add the needed machinery without the developer having to do anything.

4. Another idea is to make a source code pre-processor for the Haskell Language (e.g. using haskell-src-exts), that transforms the pure functions into blocking network calls that offload the code depending on the network conditions.

5. We can also, much more like the framework idea, use Template Haskell to generate the additional machinery involved to offload.

Furthermore, it could be interesting looking into dependent types to help specify conditions for when code can be offloaded or not.

0.2 Tie in with DTUs learning objectives

For thoroughness, the learning objectives of a graduate at the MSc Eng programme at DTU are included here,

1. can identify and reflect on technical scientific issues and understand the interaction between the various components that make up an issue

2. can, on the basis of a clear academic profile, apply elements of current research at international level to develop ideas and solve problems

3. masters technical scientific methodologies, theories and tools, and has the capacity to take a holistic view of and delimit a complex, open issue, see it in a broader academic and societal perspective and, on this basis, propose a variety of possible actions

4. can, via analysis and modelling, develop relevant models, systems and processes for solving technological problems

5. can communicate and mediate research-based knowledge both orally and in writing

6. is familiar with and can seek out leading international research within his/her specialist area.

7. can work independently and reflect on own learning, academic development and specialization

8. masters technical problem-solving at a high level through project work, and has the capacity to work with and manage all phases of a project - including preparation of timetables, design, solution and documentation

Going through these points rather quickly, I believe 1 and 2 should be clear already from section 0.1, where different current and older research is being presented, along with identifying the pain points which creates the foundation of the research topic of the thesis. As for 3 and 4, this will be demonstrated later on in the exploration of the solutions and in the implementation and evaluation of these.

Besides already having taken contact to external people, and being familiar with several of the researchers doing work in the different topics, I am is personally involved in the Haskell community which features several prominent figures of high level research in Programming Language Theory, Type Theory, Parallelism and Concurrency, Program Correctness and much more. This should help identify the relevant research that is needed to cover 6.

With regards to 7, most of the work will be done independently, while of course utilizing the expertise of both my supervisors and the larger communities in the different topics involved.

Finally, 5 and 8 remains to be seen until the evaluation of the final thesis and defense of this, but little worry is placed here as of now.

This concludes the thesis project description.

Bibliography

[Ben+17] Jose I. Benedetto et al. "Rethinking the Mobile Code Offloading Paradigm: From Concept to Practice". In: *Proceedings - 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2017* (2017), pages 63–67. DOI: 10.1109/MOBILESoft.2017.20.

[Cue+10] Eduardo Cuervo et al. "MAUI: Making Smartphones Last Longer with Code Offload". In: *MobiSys'10* 17 (2010), pages 49–62. ISSN: 10058885. DOI: 10.1145/1814433.1814441. URL: http://dl.acm.org/citation.cfm?id=1814441.

[Has17] Haskell-lang.org. *Haskell: An advanced purely-functional programming language*. 2017. URL: https://haskell-lang.org/ (visited on September 27, 2017).

[JFF13] Lei Jiao, Roy Friedman, and Xiaoming Fu. "Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities". In: *Future Network and Mobile Summit (FutureNetworkSummit)* (2013), pages 1–11. URL: http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=6633526.

[Lin14] Chit-kwan Lin. "Mobile App Acceleration via Fine-Grain Offloading to the Cloud". In: *6th USENIX Workshop on Hot Topics in Cloud Computing* (2014).

[Mar+14] Simon Marlow et al. "There is no Fork: an Abstraction for Efficient , Concurrent , and Concise Data Access". In: *ICFP '14 Proceedings of the 19th ACM SIGPLAN international conference on Functional programming* Section 9 (2014), pages 325–337. ISSN: 15232867. DOI: 10.1145/2628136.2628144.

[Mil14] Bartosz Milewski. *3. Pure Functions, Laziness, I/O, and Monads*. 2014. URL: https://www.schoolofhaskell.com/school/starting-with-haskell/basics-of-haskell/3-pure-functions-laziness-io (visited on September 27, 2017).