Ziria: Language for Rapid Prototyping of Wireless PHY

Gordon Stewart [1]
Princeton
gstew5@gmail.com

Mahanth Gowda [1]
UIUC
gowda2@illinois.edu

Geoffrey Mainland [2]
Drexel
mainland@cs.drexel.edu

Božidar Radunović
Microsoft Research
bozidar@microsoft.com

Dimitrios Vytiniotis
Microsoft Research
dimitris@microsoft.com

Doug Patterson
Microsoft
dougpatt@microsoft.com

ABSTRACT

Software-defined radios (SDR) have the potential to bring major innovation in wireless networking design. However, their impact so far has been limited due to complex programming tools. Most of the existing tools are either too slow to achieve the full line speeds of contemporary wireless PHYs or are too complex to master. In this demo we present our novel SDR programming environment called Ziria. Ziria consists of a novel programming language and an optimizing compiler. The compiler is able to synthesize very efficient SDR code from high-level PHY descriptions written in Ziria language. To illustrate its potential, we present the design of an LTE-like PHY layer in Ziria. We run it on the Sora SDR platform and demonstrate on a test-bed that it is able to operate in real-time.

1. INTRODUCTION

The past few years have witnessed tremendous innovation in the design and implementation of wireless protocols, both in industry and academia ([7, 4, 2]). Much of the work has occurred at the physical (PHY) layer of the protocol stack, which manages the translation between radio hardware signals and protocol packets. The numerous new signal processing algorithms and novel coding schemes that have resulted—many of which were first implemented using software-defined radio (SDR) platforms—have greatly increased the efficiency of existing radio communication channels.

However, SDRs have not yet reached the mainstream of wireless research. The main impediment is implementation complexity. Wireless PHY algorithms have to be very fast and efficient to meet line rate requirements. Software has to process tens of millions of complex samples per second, which is equivalent to a processing rate of close to a gigabit per second. To cope with these speeds a programmer has to have a good understanding of the underlying SDR hardware architecture and the effects of her design choices on performance.

[1]This work was done during an internship with Microsoft Research.
[2]Part of the work was done while the author was with Microsoft Research.

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.
ACM 978-1-4503-2836-4/14/08.
http://dx.doi.org/10.1145/2619239.2631427.

In order to design wireless algorithms, an SDR programmer has to have a profound understanding of signal processing and wireless networking, and in order to efficiently implement them she has to be an expert in computer architecture and system design. An average graduate student finds it quite hard to master all these areas, which is why only very few research groups have found enough resources to pursue this line of research.

To illustrate this complexity we consider two popular SDR designs. The first example is FPGA-based platforms, such as Warp [5] and Lyrtech [6]. A typical state-of-the-art SDR programming tool for FPGA platforms is based on Matlab, Simulink and an FPGA compiler. One of the main benefits of this environment is a rich set of DSP bulding blocks (such as basic fixed-point arithmetics, FFT, encoders, decoders, etc.). Further, being integrated in Matlab, this environment provides a programmer with powerful tools for debugging input and output signals, and for simulating different real-world effects using readily available libraries. However, the programming model is essentially the same as hardware design in VHDL or Verilog. A programmer has to have a fundamental understanding of how FPGAs operate. She has to be aware of different propagation delays and their effects on the system performance. One misplaced delay element can easily lead to a week of debugging!

Recently, several SDR platforms, such as GnuRadio [3] and Sora [8], allow programmers to design SDR code on general-purpose CPU processors. The hardware part of the platform performs analog-to-digital and digital-to-analog conversion of the baseband signals and supplies these signals to a CPU, where a programmer can further process it as desired.

Signal processing on a CPU provides great flexibility and permits rapid experimentation, but often at the price of decreased performance. Writing fast, real-time PHY code on a CPU can be equally, if not more complex than programming FPGAs. In fact, most of the recent academic works on SDR do not even attempt to process signals in real-time. They only use SDR platforms to capture signals. Post-processing is later done offline at much smaller processing rates than required by PHY. This approach clearly precludes any real-world networking experiments on SDR platforms.

One of the first CPU-based SDR platforms that has managed to process wireless baseband signals at a line rate is Sora [8]. It has demonstrated interoperability with commodity WiFi cards. But this performance comes with increased complexity. The code for Sora has to be written in C/C++ and a programmer needs to manually optimize the code to achieve the required speeds. The Sora project has put substantial effort into designing a C++ template library that eases the programming pain. However, the use of templates imposes several limitations, in particular on state sharing and dynamic reconfigurability of the dataflow graphs. It is worth noting that GnuRadio has a similar architecture based on C++ templates and similar limitations. We next overview the architecture of Ziria and we explain how it addresses these issues.

2. ZIRIA

Ziria is a new wireless programming platform. It consists of the Ziria language and an optimizing compiler. Ziria has a 2-layer design. The lower layer is an imperative language. It can be seen as a mix of C and Matlab code, with features of the two selectively choosen to guarantee efficient compilation. The higher layer is the language used to specify and stage stream processors.

The Ziria optimizing compiler consists of two parts. It has a front-end which parses a Ziria program and stores it in an abstract representation, and then performs several optimization passes on it. It also has a back-end which compiles the optimized abstract representation into a low-level execution model optimized for a target architecture. Our current back-end targets a low-level execution model in C for the Sora SDR platform, but it can easily be extended to other platforms, such as GnuRadio.

There are several benefits of this approach when compared to the existing approaches:

- *Dynamic staging of the control graph:* Ziria language for staging data flows allows for a dynamic reconfiguration of the control graph. This reconfiguration is very flexible, so for example a downstream component can reconfigure an upstream component or any other part of the graphs (for example using parameter passing and the repeat construct). This is in contrast with Sora and GnuRadio C++ templates that allow limited downstream reconfigurability using the multiplexing block.

- *No need for shared state:* Ziria dynamic reconfigurability and parameter passing between blocks eliminate the need for share states. Instead, a state is passed explicitly between components and all the dependencies are checked at the compile time. This is in contrast with Sora and GnuRadio which heavily rely on shared states. One example of a negative consequence is that an unitialized state cannot be detected by the compiler and causes program misbehaviour.

- *Code optimization:* Ziria compiler's optimization phase can optimize jointly the imperative part of the language and the data-flow graph. It can merge data-flow blocks, change input and output widths, inline expressions and unroll loops. One example of a useful code optimization are look-up tables. Ziria compiler can automatically identify blocks of code that operate on small bit inputs (e.g. scrambler and encoder) and convert them into lookup tables, which execute much faster on a CPU than bit operations. All this is in contrast to Sora and GnuRadio C++ template libraries where C compiler does not have an explicit understanding of the control flow graph and can perform far fewer optimizations. Lookup tables have to be implemented and tuned manually.

Overall, Ziria code is much more concise. For example, an implementation of a WiFi scrambler in Sora takes 90 lines of C++ code. The same code can be implemented in Ziria in 13 lines. It is easier to understand and has very similar performance to the hand-tuned Sora code.

3. DEMO DESCRIPTION

In the demo we present a Ziria implementation of an LTE-like PHY layer. The demo consists of two Sora SDR nodes. One node transmits a video transmission over the air and the other node receives it and plays it on the screen. All PHY code is developed entirely in Ziria. A real-time video transmission is a proof-of-concept of the real-time performance of Ziria.

Ziria is released as an open-source project. Further information about the project as well as a link to the code, examples and documentation can be found on [1].

4. REFERENCES

[1] Ziria. `http://research.microsoft.com/en-us/projects/ziria/`.

[2] T. Bansal et al. Symphony: Cooperative packet recovery over the wired backbone in enterprise WLANs. In *MOBICOM*, 2013.

[3] E. Blossom. GNURadio: tools for exploring the radio frequency spectrum. *Linux Journal*, 2004(122):4, 2004.

[4] T. Li et al. CRMA: Collision-resistant multiple access. In *MOBICOM*, 2011.

[5] P. Murphy, A. Sabharwal, and B. Aazhang. Design of WARP: a wireless open-access research platform. In *ESPC*, 2006.

[6] Nutaq. Zeptosdr. `http://nutaq.com/en/products/zeptosdr`.

[7] S. Sen, R. R. Choudhury, and S. Nelakuditi. No time to countdown: Migrating backoff to the frequency domain. In *MOBICOM*, 2011.

[8] K. Tan et al. Sora: High performance software radio using general purpose multi-core processors, 2009.