Coati Artifact README

By: Emily Ruppel and Brandon Lucia

1 OVERVIEW

This document explains the artifacts for PLDI 2019 paper submission #9: "Transactional Concurrency Control for Intermittent, Energy-Harvesting Computing Systems".

This work targets energy harvesting devices (e.g. low power sensors), so substantial custom hardware is required to reproduce the conclusions drawn in the paper. To enable artifact reviewers to feasibly evaluate the system, we have set up all of the necessary hardware to accurately measure application performance on a server in our lab as shown in Figure 1. The server can be controlled remotely via ssh.

Note: Please communicate with the other artifact reviewers to ensure that only one reviewer is accessing the server at a time. There is only one sensor board and measurement setup and it cannot accomodate multiple users.

This document provides instructions for accessing the server and executing the scripts that automate the programming and testing of applications on a real energy harvesting sensor board. Labeled diagrams of the setup and the hardware can be found in the artifact submission of hotcrp.

Please keep in mind that the artifact reviewer will be manipulating *real* hardware. This introduces the possibility for "loose wire" issues and undefined behavior if hardware is accessed without care. The reviewers should contact the authors immediately if hardware is unresponsive and only access hardware using the provided scripts.

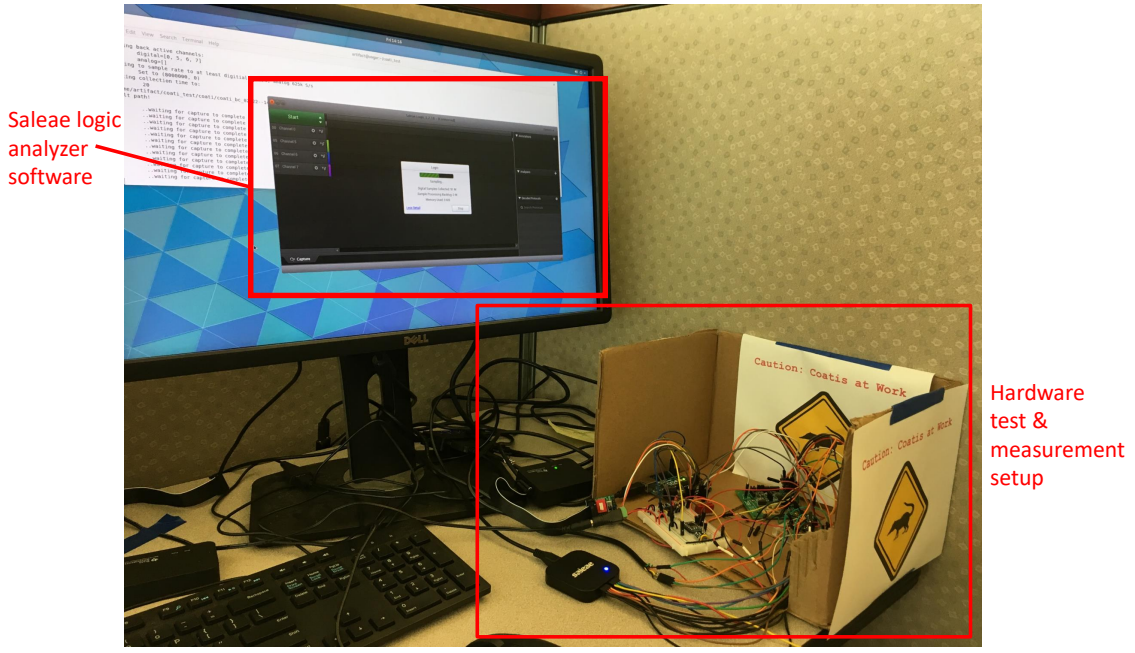

Fig. 1. The setup includes a server and a hardware setup. The server is physically connected to the hardware and it hosts the logic analyzer software that different scripts control.

2 GETTING STARTED

2.1 Connect to the Server

Login credentials: user 'xxx' password 'xxx'

Host Name: xxx

Use SSH to connect to the server and enter the password when prompted. E.g. on Linux:

```
ssh xxx@xxx
```

2.2 Environment Setup

The directory `coati_test` contains all of the application code, runtime code, test scripts and expected output of the applications. The necessary dependencies are all installed on the server and the bld directories within each app should be configured, but run the following commands to ensure that they are configured properly:

```
cd ~/coati\_test
make apps/coati/bld/gcc
make apps/buffi/bld/gcc
make apps/alpaca/bld/alpaca
```

Alpaca uses an LLVM pass to transform code so that it is safe for intermittent execution [2], to ensure that the pass if built, run the following commands:

```
cd ~/coati\_test/tools/alpaca/LLVM
mkdir -p build
cd build
cmake ..
make
```

2.3 Testing Applications

The hardware setup is somewhat brittle, please use only the scripts provided in the `coati_test/scripts` directory to test the applictions. These scripts will build, flash, and run all of the applications presented in the paper's evaluation, report the runtime, the percent of events captured, and the diff of the output with the expected output.

- `run_alpaca_apps.sh`
- `run_coati_apps.sh`
- `run_buffi_apps.sh`

Each of the scripts takes exactly one argument to specify whether the tests should be run using continuous power or harvested energy:

```
arg1 (bool) : 1 - test on continuous power, 0 - test on harvested energy
```

For example, from the `coati_test'` direcotry the following command runs all of the coati apps on continuous power:

```
./scripts/run_coati_apps.sh 1
```

And this command runs the app on harvested energy:

```
./scripts/run_coati_apps.sh 0
```

3 DETAILED EXPLANATIONS

3.1 Test Directory

The directory `coati_test` contains all of the application code, runtime code, test scripts and expected output of the applications. The hierarchy of `coati_test` as it pertains to this evaluation is as follows:

```
+ Makefile : contains all of the configuration setttings
+ apps     : directory with application code for different system configurations
+--+ alpaca   : applications from the paper's eval that will be built with alpaca
   +--+ src  : contains source code for the different apps
   +--+ bld
      +--+ alpaca : this subdirectory is built by maker and contains the final
                    executable
+--+ buffi  : contains the same test applications written for buffi
   +--+ src  : contains source code for the different apps
   +--+ bld
      +--+ gcc : this subdirectory is built by maker and contains the final
                 executable. Coati and buffi use gcc to compile, not a
                 specialized pass
+--+ coati  : contains the same test applications written for coati
   +--+ src  : contains source code for the different apps
   +--+ bld
      +--+ gcc : this subdirectory is built by maker and contains the final
                 executable
+ ext      : libraries required by the application
+--+ libcoatigcc : directory with source code for coati and buffi
+ tools    : contains maker code for building apps
+--+ maker : custom build system handles compiling app dependences
+--+ alpaca : contains the runtime code and LLVM pass that comprise alpaca
+ scripts  : all scripts for building, programming and evaluating applications
+ expected\_outputs : the correct output for each of the test apps
```

3.2 Hardware

The following is an explanation of the hardware that the server communicates with directly:

Capybara board: The energy harvesting platform that all of the applications will run on. It contains an MSP43FR5994 microcontroller and a variety of low peripheral devices (IMU, magnetometer, gesture detector, BLE radio chip) as well as an energy harvesting power system [1].

UART to USB adapter: captures print statements produced by Capybara and relays them to the serial port /dev/ttyUSB0 on the server

MSP-FET MSP MCU Programmer: A programmer that will flash the executable onto the Capybara MCU. It communicates over serial ports /dev/ttyACM0 and /dev/ttyACM1

Arduino: An extra MCU for controlling the magnetic relay that electrically isolates the Capybara from the MSP-FET when Capybara is running on harvested energy and providing the gpio pulses that trigger events in the test apps. It communicates on serial port /dev/ttyACM2

Saleae Logic analyzer: A digital logic analyzer that records a trace of gpio pulses and produced by Capybara while applications are running.

Figure 2 shows the hardware setup and the different connections between each of the components.

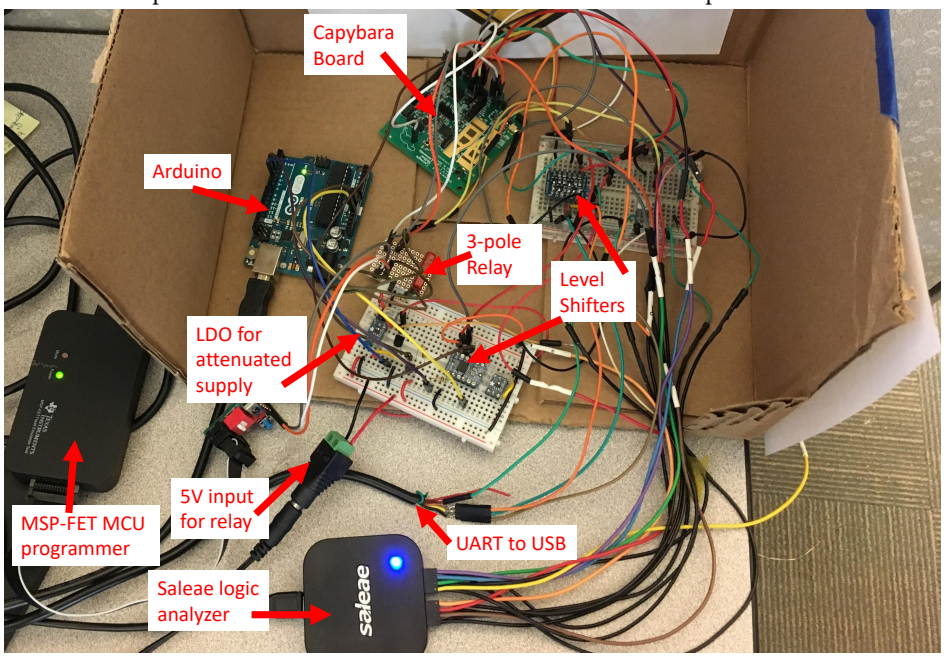

Fig. 2. The major components of the hardware setup are shown. The Capybara is programmed by the MSP-FET and it receives input from the Arduino. The UART to USB adapter and Saleae report data produced by the Capybara. The level shifters and relay isolate the Capybara when it is running on harvested energy from the highly attenuated power supply.

3.3 Detailed Testing

Time to failure using alpaca: The paper demonstrates that the Alpaca programming model cannot be used for applications that have interrupt service routines that modify memory shared with synchronous tasks. When run_alpaca_apps.sh runs, it will check if the application execution encountered and error, if an error is encountered, instead of runtime it reports time to failure.

To observe the failures that occur on harvestd energy, as shown in Table 3 of the paper, run:

```
./scripts/run_alpaca_apps.sh 0
```

Comparing buffi and coati: The paper demonstrated that the split-phase event strategy employed by Coati is more efficient than the full buffering used by Buffi. To easily compare the runtime difference on continuous power (as seen in Figure 12) run the following code. It produces a pdf of a graph that the reviewer can transfer to their machine to view:

```
./scripts/run\_coati\_apps.sh 1
./scripts/run\_buffi_apps.sh 1
python scripts/compare\_cont_pwrd_runtimes.py
scp xxx@xxx:/home/xxx/scripts/outputs/cont\_pwrd_runtimes.pdf .
```

Similarly, to compare the runtime on harvested energy (as seen in Figure 14), run:

```
./scripts/run\_coati_apps.sh 0
./scripts/run\_buffi_apps.sh 0
python scripts/compare\_harv_pwr_runtimes.py
scp xxx@xxx:/home/xxx/scripts/outputs/harv\_pwr_runtimes.pdf .
```

REFERENCES

[1] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18)*. ACM, New York, NY, USA, 767–781. https://doi.org/10.1145/3173162.3173210

[2] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent Execution without Checkpoints. In *Proceedings of the 2017 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM.