Research Progress

We start with an abstract of our project roughly based on the original proposal, since the core research idea did not change. We then discuss the progress of five specific goals which we investigated during the year with noticeable advances.

Abstract

Effective use of low-power standby will play a fundamental role in energy efficiency for the expected 50 billion IoT devices by 2020. This research project aims to address the software challenges, as observed by the International Energy Agency, towards an energy-efficient IoT: to ensure that devices employ the lowest possible modes of standby, and that devices remain in longest possible periods of standby.

Given the projected scale of the IoT and the role of power standby controlled by software towards energy efficiency, this research project has the following goals:

- Address energy efficiency through extensive use of standby in software.
- Target constrained embedded microcontroller architectures that form the IoT.
- Provide standby mechanisms at the programming language level that scale to all applications.
- Support transparent/non-intrusive standby software mechanisms that reduce barriers of development.

The bulk of our research is based on the design of Céu, a new reactive programming language, which we have been working for the past 9 years. The language is grounded on the synchronous concurrency model, which trades power for reliability and has a simpler model of time that suits most requirements of IoT applications. In this model, all reactions to the external world are guaranteed to be computed in bounded time, ensuring that applications always reach an idle state amenable to standby mode.

Specific Goals - Original Proposal

In the original proposal, we enumerated three specific goals with associated risks and expected contributions. In the mid-term report, we included two other goals which we are working on. We now also include an estimated completeness of each goal to give an idea of the overall progress.

Goal 1 - IoT Hardware Infrastructure

- **Risks:** very low (1/5)
- **Expected contribution:** (minor scientific contributions)

- **Completeness:** 75%

We need an energy-efficient IoT hardware infrastructure that allows our software infrastructure to control it and take advantage of low-power modes of operation.

We adopted third-party Arduino hardware platforms with builtin low power support and did not expect much risks associated with this choice. We also selected IoT peripherals (e.g., radio transceivers and real-time clocks) that can operate in sleep modes to consume less energy.

Using this infrastructure, the automatic software mechanisms from our proposal can now control the hardware to reach almost 100% energy efficient without compromising the functionalities of the applications.

We want even more control over the hardware components and intend to prototype a customizable and affordable low-power development board as a side project. Last year, we submitted this idea to the main Brazilian scientific funding agency (*CNPq - Universal Call*), but did not receive the grant. The proposal is attached as the file "cnpq-universal-18.pdf".

Goal 2 - IoT Software Infrastructure

- **Risks:** high (4/5)
- **Expected contribution:** An energy-aware programming language that integrates with the surrounding physical environment.
- **Completeness:** 80%

The main challenge we confronted this year was on extending the design of the programming language Céu to use standby effectively:

- A programmer writes **any** IoT application in Céu unaware of standby or energy constraints.
- The compiler of Céu generates a program that puts the IoT device to sleep automatically and in the most efficient standby mode for as long as possible.

We designed Céu from the very beginning to be promptly reactive to the surrounding environment. Basically, whenever a stimulus occurs (e.g., a sensor detects a change, or a radio message arrives), the program must be ready to react to that input. Céu guarantees that reactions always execute in finite time so that upcoming inputs are also readily handled.

A key insight is that if the period between subsequent stimuli is much higher than the associated reaction times, then the application can be in standby most of the time. As an example, if a radio message arrives every second and the time to process each message is in the order of a few milliseconds, then the device can sleep almost 100% of the time. Follows the general approach we employed to materialize this insight:

- Enforce programs to reach idle states after each reaction
- Put the device in standby mode whenever the program is idle
- Only awake programs from hardware interrupts on external stimuli

We described this approach in two work-in-progress papers, which we presented in June at LCTES'18 (Languages, Compilers, and Tools for Embedded Systems) and in November at REBLS'18 (Reactive and Event-Based Languages and Systems). The papers are attached as the files "lctes-wip-18.pdf" and "rebls-wip-18.pdf".

In the first paper, which targets embedded system researchers, we describe the design at a higher level, first discussing why current languages cannot provide automatic standby for applications, and then why and how we can extend Céu to understand which level of standby to adopt after each reaction and how to awake only from hardware interrupts.

In the second paper, which targets reactive language designers, we described the technical details of interrupt service routines in Céu. For instance, we detailed how programmers can map an input from the environment (e.g., a button press) to a reaction in the program. We also discussed the safety threads due to concurrency and how Céu can detect race conditions at compile time in order to prevent them.

We consider that we reached an effective design regarding energy efficiency, but we still need to polish the implementation of Céu to make it ready for a comprehensive quantitative evaluation to be described next.

Goal 3 - IoT Applications

- **Risks:** medium (3/5)
- **Expected contribution:** A realistic energy-efficient IoT programming alternative to Arduino.
- **Completeness:** 30%

In order to evaluate the gains in energy efficiency with the proposed software infrastructure, we need to evaluate the consumption of realistic applications.

In the first year, we focused on basic support for radio communication, which plays a fundamental role in the IoT: we developed SPI and RF drivers on top of our new infrastructure and also a basic ad-hoc communication protocol. Then, we measured the energy consumption of a number of applications to compare our system with the original Arduino:

```
================================================================
App         Arduino      Céu          Description
----------------------------------------------------------------
Empty         3.7         0.002       empty program
Blink         6.0         3.1         periodic LED blinking
Sensor       11.4         7.7         sensors reads
Radio        19.5        15.8/3.0     periodic RF communication
```

```
Protocol    19.6        15.9        active mesh RF protocol
================================================================
```

We did not advance much on these experiments after the mid-term report, so the numbers above are the same. The first case illustrates how an empty program in Arduino remains 100% of the time awake, while an equivalent in Céu sleeps in the most efficient standby mode. The blinking example in Céu relies on timer standby, which is not the most efficient, but still consumes less energy than its Arduino counterpart. The radio example in Céu can turn off the radio periodically, showing significant savings while in the awaiting state (3.0 mA). The protocol example illustrates a scenario for a busy node in a mesh network, which keeps the radio on 100% of the time, being unable to save much energy.

As we already mentioned on the mid-term report, the results are preliminary and should be taken with a grain of salt. We stll did not evaluate complex applications, which will demand substantial work until we consider a full publication comprising our three original goals.

Goal 4 - Training and Education

- **Risks:** low (2/5)
- **Expected contribution:** A framework for early computer science education centered around the IoT.
- **Completeness:** 20%

One of the difficulties we encountered early in the project was to hire students with basic skills on embedded systems and IoT. Since we are proposing a new programming approach for the IoT, the difficulties are even more remarkable.

In May, a student got a 3-month scholarship from the "Google Summer of Code" initiative and started to develop a new tool "Céu-Maker" targeting students with no programming background. In October, we submitted a project to encourage more girls to pursue STEM careers (*CNPq - Girls on Sciences, Engineering, and Computing*). The project was recommended but we did not receive the grant. The proposal is attached as the file "cnpq-meninas-18.pdf". As a follow up, we plan to offer a 3-week course open for early undergraduate students on each semester, starting this year.

In September, we introduced a new course on embedded systems in our University targeting graduate and late undergraduate students. The goal is to expose more technical aspects related to our research and instigate students seeking for dissertation topics.

Goal 5 - Formal Proofs

- **Risks:** very high (5/5)

- **Expected contribution:** An automated mathematical proof that reactions are deterministic, memory bounded, and terminate.
- **Completeness:** 20%
- rewrite Céu from scracth
- aposta em LH
- Anny
  - education
- Joao
  - games
- Anna
  - IoT / energy

not originally

rewrite from scratch use automatic proofs - more reliable - extensible

- focus on 3,5
  - higher risks and outcomes
  - others for the team we will build
    - important for bandwidth
  - goal 3 is too demanding of new ppl
- prove?
- Provide standby mechanisms at the programming language level that scale to all applications.
- Support transparent/non-intrusive standby software mechanisms that reduce barriers of development.
- no trained
- more clear what is expected

Team and Partners

- What individuals have worked on the project and what other organizations have been involved as partners

Carrier Impact

- Carrier Impact/Comments/Concerns (short paragraph)
- Life-time project
- PEL

Next Steps

- The next steps for the following three years in your project
- syntax and vocabulary
- semantics
- only myself

  Nevertheless, the novel support for ISRs would complete the whole development cycle, from the application down to the hardware, without operating system support.
- All software uses progr. languages
- Turing awards
- Big enterprises (Images)
  - Microsoft: VB, C#, ...
  - Google: Go
  - Apple: Swift
  - Mozilla: Rust
  - Sun/Oracle: Java
  - none of them are looking at energy