Energy Efficiency for IoT Software in the Large

Francisco Sant'Anna

francisco@ime.uerj.br

www.ceu-lang.org

About us...

- Research areas:
  - Programming Languages
  - Concurrent & Distributed Systems
- Programming Language “Céu”
  - Embedded platforms
  - IoT & Wireless Sensor Networks
  - Games
  - ~10 years of research

Energy Efficiency for IoT

- 15 billion “traditional” network-connected devices in 2015 (e.g., mobile phones & smart TVs).
- 75 billion by 2025 with the IoT (e.g., smart bulbs & fitness wearables). [2016]
- **Most energy consumed in standby mode.**
- Network standby is one of the six fronts on IEA/G20's Energy Efficiency Action Plan
    - https://www.iea-4e.org/projects/g20

Our Research Goals

1. Address energy efficiency through rigorous use of **standby**.

2. Target **constrained** embedded architectures that form the IoT.

3. Provide standby mechanisms at the **programming language** level that scale to all applications.

4. Support **transparent**/non-intrusive standby mechanisms that reduce barriers of adoption.

Research Non-Goals

- Adaptive Computing
  - QoS (e.g., resolution, frame rate, accuracy)
  - Behavior (e.g., switch UI, disable functionalities)
  - *Don't take advantage of standby modes (goal 1)*
- Energy-Aware Network Protocols
  - Low-power listening
  - Small periods of duty cycles
  - *Only for networked parts, not automatic (goals 3 & 4)*
- Complex Hardware Architectures
  - Microprocessors, MMU, OS-based, Smartphones
  - *Not constrained embedded platforms (goal 2)*

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
- Infer deepest sleeping mode
- Put device to sleep
- Only awake from interrupts

A Motivating Example

Every second, read sensor value and broadcast it.

- Arduino

```
while (1) {
  delay(1000);
  int v = analogRead();
  radioWrite(v);
}
```

- Céu

```
loop do
  await 1s;
  int v = await AnalogRead();
  await RadioWrite(v);
end
```

Let’s add concurrency…

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
  if (radioAvail())
    break;
  uint32_t cur = millis();
  if (cur > prv+1000) {
    prv = cur;
    int v = analogRead();
    radioWrite(v);
  }
}
```

```
while (1) {
  delay(1000);
  int v = analogRead();
  radioWrite(v);
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
  if (radioAvail())
    break;
  uint32_t cur = millis();
  if (cur > prv+1000) {
    prv = cur;
    int v = analogRead();
    radioWrite(v);
  }
}
```

- Céu

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

Standby Considerations

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
  if (radioAvail())
    break;
  uint32_t cur = millis();
  if (cur > prv+1000) {
    prv = cur;
    int v = analogRead();
    radioWrite(v);
  }
}
```

- Céu

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

Céu in a Nutshell

- Reactive
  - environment in control: *events*
- Imperative
  - sequences, loops, assignments
- Concurrent
  - multiple lines of execution: *trails*
- Synchronous
  - trails synchronize at each external event
  - **trails are always awaiting**
- Deterministic
  - always yields the same outcome for a given timeline

Project In Practice...

- Hardware infrastructure
  - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
  - Implement an energy-aware runtime for Céu
  - Rewrite device drivers in Céu (timers, ADC, Radio)
- Applications
  - Rewrite existing IoT applications in Céu
    - Time to rewrite
    - Coding “aesthetics”
    - Energy consumption

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
output void ADC_REQUEST do
    ADMUX = 0x40|(A0&0x07);
    bitSet(ADCSRA, ADIE);
    bitSet(ADCSRA, ADSC);
end
```

```
input ADC_vect_num do
  bitClear(ADCSRA, ADIE);
  emit ADC_DONE(ADC);
end
```

```
code AnalogRead (void) -> int
do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
output void ADC_REQUEST do
    ADMUX = 0x40|(A0&0x07);
    bitSet(ADCSRA, ADIE);
    bitSet(ADCSRA, ADSC);
end
```

```
input ADC_vect_num do
  bitClear(ADCSRA, ADIE);
  emit ADC_DONE(ADC);
end
```

```
void pm_sleep (void) {
  if (PM_GET(PM_TIMER1)) {
      sleep_1(<...>)
    } else if (PM_GET(PM_ADC)) {
      sleep_2(<...>);
    } else {
      sleep_3(<...>);
    }
  }
}
```

```
code AnalogRead (void) -> int
do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
output void ADC_REQUEST do
    ADMUX = 0x40|(A0&0x07);
    bitSet(ADCSRA, ADIE);
    bitSet(ADCSRA, ADSC);
end
```

```
input ADC_vect_num do
  bitClear(ADCSRA, ADIE);
  emit ADC_DONE(ADC);
end
```

```
void pm_sleep (void) {
  if (PM_GET(PM_TIMER1)) {
      sleep_1(<...>)
    } else if (PM_GET(PM_ADC)) {
      sleep_2(<...>);
    } else {
      sleep_3(<...>);
    }
  }
}
```

```
code AnalogRead (void) -> int
do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

General Approach
(standby, constrained, programming language, transparent)

- Enforce idle states of execution
  - Céu enforces a reactive model of execution
- Infer deepest sleeping mode
  - Céu has a semantics amenable to analysis
- Put device to sleep
  - Céu has an energy-aware runtime
- Only awake from interrupts
  - Céu provides interrupt service routines (ISRs)

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

Publications

- SenSys'13, ACM Embedded Networked Sensor Systems
  - *Safe System-level Concurrency on Resource-Constrained Nodes*
- TECS'17, ACM Transactions on Embedded Computing Systems
  - *The Design and Implementation of the Synchronous Language Céu*

- LCTES'18, ACM Conference on Languages, Compilers, and Tools for Embedded Systems
  - *A Memory-Bounded, Deterministic and Terminating Semantics for the Synchronous Programming Language Céu*
- LCTES'18, ACM Conference on Languages, Compilers, and Tools for Embedded Systems
  - *WIP: Transparent Standby for Low-Power, Resource-Constrained Embedded Systems*

Energy Efficiency for IoT Software in the Large

Francisco Sant'Anna

francisco@ime.uerj.br

www.ceu-lang.org