Lax: Driver Interfaces
for Approximate Sensor Device Access

Phillip Stanley-Marbell and Martin Rinard
psm@mit.edu

HotOS XV, Kartause Ittingen, Switzerland.
**Lax: Driver Interfaces**
for Approximate Sensor Device Access

Phillip Stanley-Marbell and Martin Rinard
psm@mit.edu

ARM Cortex M3 Microcontroller
Accelerometer IC
Bluetooth Low-Energy IC

(Source: Fitbit)
(Source: ifitxit.com)

HotOS XV, Kartause Ittingen, Switzerland.
Power Breakdown for a Wearable Sensor System

- Magnetometer sensor: 0.436 mW
- Pressure sensor: 0.010 mW
- Accelerometer sensor: 0.540 mW
- Gyroscope sensor: 18.300 mW
- Infrared/color sensor: 0.528 mW
- Humidity sensor: 0.528 mW
- Bluetooth Low-Energy Radio: 0.312 mW
- Processor: 0.408 mW
- 20x20 pixel OLED display: 0.235 mW

(Note: sectors scaled logarithmically due to large range of values)
Sensors/Displays Dominate Power Dissipation in Wearables

To improve system-wide energy-efficiency, focus on the dominant fraction

(Note: sectors scaled logarithmically due to large range of values)
Reducing Power Dissipation of Sensor Devices

What happens?

Errors (per $10^3$ Readings)

A

TI TMP006
IR Temperature Sensor

B

ST L3G4200D Gyroscope Sensor
Reducing Power Dissipation of Sensor Devices

A

TI TMP006
IR Temperature Sensor

B

ST L3G4200D Gyroscope Sensor
Reducing Power Dissipation of Sensor Devices

**A**

TI TMP006
IR Temperature Sensor

**B**

ST L3G4200D Gyroscope Sensor
How Can We Exploit What We Have Just Seen?

(access error rate versus power dissipation tradeoffs)

We can trade tolerance to errors for performance and power of sensors

Our objective: provide a **principled way to tradeoff error for energy** in systems

```plaintext
sample = sensor_read(sensorName, tolerableError);
```
How Can We Exploit What We Have Just Seen?

(access error rate versus power dissipation tradeoffs)

We can trade tolerance to errors for performance and power of sensors.

Our objective: provide a **principled way to tradeoff error for energy** in systems.

```python
sample = sensor_read(sensorName);
```
How Can We Exploit this Property in Real Systems?

HW Measurements

Domain-Specific Language

Runtime System

(Sources: Fitbit and ifixit.com)
How Can We Exploit this Property in Real Systems?

1. Characterize **HW error properties** (sometimes possible from datasheets)

HW Measurements

\[ \lambda x. x \]

Domain-Specific Language

Runtime System

(Sources: Fitbit and ifixit.com)
How Can We Exploit this Property in Real Systems?

1. Characterize **HW error properties** (sometimes possible from datasheets)
2. Capture **permitted error behavior** in Lax language (per application/domain)

**HW Measurements**

**Domain-Specific Language**

**(Sources: Fitbit and ifixit.com)**
How Can We Exploit this Property in Real Systems?

1. Characterize **HW error properties** (sometimes possible from datasheets)
2. Capture **permitted error behavior** in Lax language (per application/domain)
3. **Tools** to exploit HW properties modulo permitted behavior, per application
How Can We Exploit this Property in Real Systems?

1. Characterize **HW error properties** (sometimes possible from datasheets)
2. Capture **permitted error behavior** in Lax language (per application/domain)
3. **Tools** to exploit HW properties modulo permitted behavior, per application
4. **Runtime system** to use information generated from tools to control HW

(Sources: Fitbit and ifixit.com)
An Example

sensor gyroSourceXaxis @ 1.6V = {
    provide (loss) {
        occurs: likelihood < 25 in 1000 readings;
    }
}
...

tolerance gyroTolerance = {
    require (loss) {
        occurs : likelihood < 100 in 1000 readings;
    }
}

Example Usage:

/* Use Lax to achieve lowest power for required accuracy. */
sampleC = lax_sensor_read(gyroSourceX, gyroTolerance);
An Example

Example Usage:

/* Use Lax to achieve lowest power for required accuracy. */
sampleC = lax_sensor_read(gyroSourceX, gyroTolerance);
Great, But How Does it Affect Real End-to-End Applications?

![Diagram](image)

- **x-component analysis**
- **y-component analysis**
- **z-component analysis**

**Pedometer / Step Counting System**

1. **Maximum-Activity Axis Selection**
2. **Low-Pass Filter**
3. **Extremal Value Marking**

**Step Count**

---

Great, But How Does it Affect Real End-to-End Applications?

Unperturbed accelerometer data‡ for one axis

Induced error (erasure) for 5% of samples

Where We Are Today with **Lax**

We are pursuing both the language and hardware facets of Lax

Evolving language design (e.g., temporal distributions for tolerable error)

Statically determining appropriate runtime settings from Lax descriptions

Other hardware techniques to expose new tradeoffs to Lax

**Custom HW to explore using Lax in a “real” system**

**Why:** usage experience, multiple sensors for ground truth, etc.

- 2x Pressure sensors
- 3x Accelerometers
- 2x Gyroscopes
- 2x Magnetometers
- 2x Infrared/light sensors
- 2x Humidity sensors
- 1x Bluetooth LE radio

Hardware support for sensor voltage control

+ More...
The Big Picture: What Lax Enables

- A practical way to reduce power dissipation in sensor-driven systems.
  Provides a **new dimension for tradeoffs**: error (accuracy / precision / reliability) in addition to traditional metrics of power and performance.
The Big Picture: What Lax Enables

A practical way to reduce power dissipation in sensor-driven systems. Provides a new dimension for tradeoffs: error (accuracy / precision / reliability) in addition to traditional metrics of power and performance.

Opportunities for new research directions:

For language / compiler research
- Types for error properties (there is already some prior work)?
- Determine required accuracy/reliability from program source?

For systems research
- OS as an error-varying programmable VM?
- Scheduling to align error-alike accesses?
- Should OS manage error (like power, time)?
Opportunities for new research directions:

- For language / compiler research
  ① Types for error properties (perhaps, build on linear dependent types)?
  ② Estimate required accuracy/reliability from program source?

- For systems research
  ① OS as an error-varying programmable VM?
  ② Scheduling to align error-alike accesses?
  ③ Should OS manage error (like power, time)?