Toward Energy-Aware Programming for Unmanned Aerial Vehicles

Yu David Liu
SUNY Binghamton
Binghamton, NY 13902, USA
Email: davidL@binghamton.edu

Lukasz Ziarek
SUNY Buffalo
Buffalo, NY 14260, USA
Email: lziarek@buffalo.edu

***Abstract*—Unmanned Aerial Vehicles (UAVs) are an emerging platform with promising applications in merchandise delivery, geographical surveillance, disaster management, to name a few. Energy consumption is a first-class concern in UAV design, as the majority of today's lightweight UAVs are either battery-powered or solar-powered. In this talk, we describe a first step toward developing sustainable tasks and applications, through a sustainability-aware programming level to the domain of UAV software development.**

I. Introduction

Within the family of cyber-physical systems, Unmanned Aerial Vehicles (UAVs) have received significant attention in recent years with their unique combination of features in flight control, robotic control, and embedded systems. As the majority of today's UAVs are battery-powered — with some solar-powered with limited electricity generation — a crucial concern that impact the future development of UAVs is energy efficiency. Unlike other computer systems where energy efficiency only matters in the context of electricity cost, a drain of electricity source in UAVs often entail severe consequences, from damaging the device itself, to posing safety hazards to the ground it covers. In existing UAV systems, energy and power management is primitive, often based on the human operator's imprecise estimation on the consumption needed for the flight mission. Today's application development is also primitive, often in the forms of low-level embedded system programming.

In this paper, we consider how to make flight tasks energy-aware at development time. Specifically, we rely on a recently designed programming language for maintaining sustainability, Eco [12]. In a nutshell, sustainability is the ability of a program to "stay on energy budget," neither yielding deficit nor surplus. At the first glance, the goal here may appear counter-intuitive: wouldn't it be more desirable to yield as much surplus as possible? Ideally, yes, but free lunch is over. The most effective energy management strategies are known to be trade-offs (*e.g.*, between energy and quality of service). To balance the trade-off, an energy-aware program should neither squander nor skimp.

To motivate our project of supporting sustainable UAVs, let us consider two examples:

> *Example I: Sustainable image processing*: Imagine a surveillance UAV application that continuously transmits images to its ground station. Data transmission through wireless datalinks — especially for data such as images — consumes significant energy. To be more energy-aware, a sustainable program may be able to automatically and adaptively adjust image sampling precision rates based on energy budget.

and

> *Example II: Sustainable Circle Navigation*: circle navigation is a fundamental navigation mode of UAVs, important such as when the flight is in the stand-by mode or waiting for landing. Circle navigation is generally approximated by a navigation trajectory over a many-sided polygon. To be more energy-aware, a sustainable program may be able to automatically and adaptively adjust how many sides should be used for the polygon-based approximation.

In the next sections, we describe how these two Examples can be supported through two flavors of sustainable programming, the *task-centric* sustainable programming and the *data-centric* sustainable programming.

II. Task-Centric Sustainable Programming for UAVs

Inspired by economics and urban planning, Eco models sustainability as a form of supply and demand matching. A sustainable program consistently maintains the equilibrium between supply and demand. Let us now revisit the image processing example for UAVs, and the sample Eco program can be found in Figure 1(A).

Under this programming model, each task subject to sustainability management can be enclosed as a code block — which we call a *sustainable block* — preceded by the keyword **sustainable**, between Line 8 and Line 12 in Fig. 1(A). The block is followed by two *characterizer* constructs:

- **supply** characterization, indicating an application-specific "energy budget" for executing the enclosed sustainable block. In the example, the declaration says that the batch transmission of the images in total should not exceed 0.2% of the remaining UAV battery.
- **demand** characterization, serving as an application-specific "progress estimator" for executing the enclosed sustainable block. The characterizer consists of two parts,

```
1  modes { lo <: hi };
2  mcase<float> precision = {lo: 0.7; hi: 0.9995 };
3  Image buf [IMGS];
4
5  Image sample (Image i, float p) { ... }
6  void transmit(Image i) { ... }
7  void process() {
8    sustainable {
9      for (int num = 0; num < IMGS; num++) {
10       Image img = sample(buf[num], precision);
11       transmit(img);
12     }
13     supply (0.002 * battery)
14     demand (IMGS) -> (IMGS - num);
15   }
16 }
```

(A)

```
1  modes { lo <: hi };
2  mcase<float> radian_delta = {lo: 30; hi: 15 };
3  float centerx, centery, x, y, radius;
4  int radian_accumulate = 0
5  under sustainable
6     supply (0.002 * battery)
7     demand (360) -> (360-*);
8  ;
9
10 void fly_circle_next_step() {
11     x = nextposx(x, radian_delta);
12     y = nextposy(y, radian_delta);
13     radian_accumulate += radian_delta;
14     fly_to_xy(x, y);
15   }
16 }
```

(B)

Fig. 1. (A) shows a task-centric sustainability model and (B) shows s data-centric sustainability model.

connected by the symbol `->`: the *demand sum* denoting how much work in total, and the *demand gauge* denoting how much work is left. In the example, the overall work is abstractly represented by `IMG` — there are `IMG` number of images to process in total — and during the execution of the sustainable block, the remaining work is characterized by expression `IMG - num`, the number of images to be processed.

Based on the two characterizations, the language runtime automatically monitors the changes in supply and demand as the sustainable block is executed. Upon significant imbalance between remaining supply and demand, the program runtime reacts adaptively to "re-stabilize" the equilibrium between supply and demand. The adaptive control support is assisted by a language abstraction called *first-class mode cases*. First-class mode cases are an encapsulation of alternative values/behaviors based on modes into a first-class citizen in the programming language. For instance, an image sampling precision rate can be defined as {`lo: 0.7; hi: 0.9995`}, and a variable holding such a value represents the intuition fact that the precision is 70% if the program is executing under the `lo` mode, or 99.95% if executing under the `hi` mode. As first-class values, mode cases can be assigned to variables, stored on heaps, or passed around through function boundaries.

In the presence of *deficit* (*i.e.*, when remaining demand far exceeds remaining supply), the "operational mode" of the sustainable block execution is lowered — such as switching from `hi` to `lo` in the example — and all variables holding first-class mode cases will be *de facto* destructed to the value for that "operational mode." The same strategy can work analogously for *surplus* (*i.e.*, when remaining supply far exceeds remaining demand). Since the monitoring of the equilibrium is continuous throughout the block execution, the operational mode — and hence the destruction of the mode cases — can be adaptively switched, allowing the program to display energy-adaptive behaviors.

Overall, the simple design here reflects the "human-in-the-loop" philosophy for energy management: the programmer provides application-specific knowledge to guide the system runtime as characterizations, and the system runtime saves the programmer from the worries of considering when to adapt, what to adapt to and how to maintain sustainability.

III. Data-Centric Sustainable Programming for UAVs

The discussion so far focuses on the sustainability of executing a task, in the form of a block of program fragment, which we term *task-centric sustainability management*. We are further interested in supporting sustainability management through a *data-centric* view, which we describe through *Example II*, whose sample code is shown in Fig. 1(B).

Compared with Fig. 1(A), but with one important difference, the sustainability characterizers are not associated with a block of code, but instead a (potentially global) variable. In contrast with the task-centric abstraction, the data-centric design here allows sustainability management to span through multiple function invocations. In this example, the demand characterizer says that the overall demand is that the accumulated radian of the circle navigation (represented by variable `radian_accumulate`) is 360 degrees, whereas the remaining demand during the program execution is characterized by `360-*` — where `*` refers to the variable `radian_accumulate` itself. On the high level, this is aligned with our intuition that the remaining demand is the remaining radian to cover during a circle navigation. With such declarations, the language runtime will automatically balance the equilibrium whenever the `radian_accumulate` is updated. For example, if it discovers the navigation uses up 0.1% of the battery (half the specified budget) but has only covered 90 degrees of the radian, the operational mode of the program needs to be lowered, which in turn implies the change in radian for each polygon-based approximation (maintained by first-class mode case `radian_delta`) will be increased. Data-centric sustainability management works particularly well for event-based applications, common for UAV systems. Under such scenarios, each event handler may lead to very small energy consumption, but the accumulated effect is large and should be managed.

IV. A Research Roadmap

This position paper reports the early stage of our effort to support sustainable programming to UAVs. The task-centric

sustainable programming model is currently available in the Eco compiler, whereas the data-centric sustainable programming model is a planned extension. We plan to port the Eco compiler to Raspberry Pi [9], a lightweight platform with successful deployment on Paparazzi-driven UAVs [7]. The Eco compiler supports source-to-source translation to Java. We have successfully ported JVM to run on Raspberry Pi.

An important engineering decision for programming model design is whether the solutions will be a set of new APIs, or in the form of a new compiler. The API-based solution arguably requires fewer changes in programmer habit, and the compiler-based approach enables opportunities for enforcing the correct usage of the new abstractions (through program analyses). We plan to take a two-step approach: we first develop APIs to capture the semantic novelty of our proposed ideas, and then develop compiler extensions whose target code generation uses the previously developed APIs. Following this two-step approach, the gap in development between API and compiler is small: only parsing and semantic analysis. We now outline some technical challenges in programming model design.

a) Task I: A Unified Design for Sustainability Management: The proposed sustainable programming abstractions can either be task-centric or data-centric, which leads to a number of research issues our project will investigate into:

- How to build task-centric and data-centric sustainability management in one language, so that there is no feature intervention when the programmer uses both flavors of programming in the same application?
- Do we allow two task-centric sustainability management to be nested, *i.e.*, a long-running task may subject to an energy budget whereas a portion of its execution subjects to another energy budget? If nesting is supported, what is the most intuitive design for supporting adaptive behaviors?
- When two different memory locations are each subject to the data-centric sustainability management, how do they coordinate on adaptive control?
- Do we allow the demand characterization to be specified over *a set of* of variables, *i.e.*, placing multiple variables under the same sustainability management?

b) Task II: Asynchronous Support: In UAV applications, tasks are often created asynchronously, and handled in a separate thread to improve reactiveness and real-time support. This programming idiom poses a threat to sustainability management: task creation alone does not accurately characterize the demand. For example, when an execution sequence reaches a point of asynchronous task creation, the program counter will immediately move on to the continuation. Without additional support, the language runtime to manage the equilibrium between supply and demand may erroneously deem the demand of the created task being negligible (and hence its impact on supply negligible).

c) Task III: Empirical Studies: Finally, we plan to conduct a series of empirical studies to "chart" the adaptation and energy efficiency design space of UAVs. We plan to answer key research questions such as:

- How many types of parameters can be dynamic adjusted to support adaptive control?
- What is the impact of adaptation for the quality of the results?
- What is the impact of adaptation to energy consumption?
- What are the common energy budget needs, and what are the common patterns for adaptive budgets?
- What are the common patterns of demand change?

The knowledge gained from the study will not only help us understand UAV software systems better, but also delineate the design and optimization space of designing adaptive and energy-efficient UAV systems.

V. Related Work

Designing new programming techniques in support of energy awareness is a relatively new area. Data-flow processing paths may alternate based on energy availability in Eon [11]. User-specific QoS may be calibrated in Green [1]. ET [5] provides a typed programming model for phase-based and mode-based energy management. Green Streams [2] optimize energy efficiency through a data rate-based analysis over a stream programming model. Ent [3] combines static typing and dynamic typing to improve the proactiveness and adaptiveness of energy management. More broadly, approximate programming is a promising technique for streamlining software-hardware coordination and improving resource usage efficiency, often with consequences on energy savings. Examples include EnerJ [10], Rely [4], Chisel [6], and FlexJava [8]. Among related work, Eco is the only programming model with first-class abstractions for sustainability. We are unaware of any energy-aware programming models designed for UAVs.

VI. Acknowledgments

We thank Anthony Canino for useful discussions. This project is supported by US NSF CNS-1512992, CNS-1513006, CNS-1405614, and partially by NSF CCF-1526205.

References

[1] Baek, W., and Chilimbi, T. M. Green: a framework for supporting energy-conscious programming using controlled approximation. In *PLDI'10* (2010), pp. 198–209.

[2] Bartenstein, T., and Liu, Y. D. Green streams for data-intensive software. In *ICSE'13* (2013), pp. 532–541.

[3] Canino, A., and Liu, Y. D. Proactive and adaptive energy-aware programming with mixed typechecking. In *PLDI'17* (2017).

[4] Carbin, M., Misailovic, S., and Rinard, M. Verifying quantitative reliability for programs that execute on unreliable hardware. In *OOPSLA '13* (2013), pp. 33–52.

[5] Cohen, M., Zhu, H. S., Emgin, S. E., and Liu, Y. D. Energy types. In *OOPSLA '12* (October 2012).

[6] Misailovic, S., Carbin, M., Achour, S., Qi, Z., and Rinard, M. C. Chisel: reliability- and accuracy-aware optimization of approximate computational kernels. In *OOPSLA'14* (2014), pp. 309–328.

[7] Paparazzi installation guide: Raspberrypi, https://wiki.paparazziuav.org/wiki/Installation/RaspberryPi.

[8] Park, J., Esmaeilzadeh, H., Zhang, X., Naik, M., and Harris, W. Flexjava: Language support for safe and modular approximate programming. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (2015), ESEC/FSE 2015, pp. 745–757.

[9] Raspberry pi, https://www.raspberrypi.org/.

[10] SAMPSON, A., DIETL, W., FORTUNA, E., GNANAPRAGASAM, D., CEZE, L., AND GROSSMAN, D. EnerJ: Approximate data types for safe and general low-power computation. In *PLDI'11* (June 2011).

[11] SORBER, J., KOSTADINOV, A., GARBER, M., BRENNAN, M., CORNER, M. D., AND BERGER, E. D. Eon: a language and runtime system for perpetual systems. In *SenSys* (2007), pp. 161–174.

[12] ZHU, H. S., LIN, C., AND LIU, Y. D. A programming model for sustainable software. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1* (2015), pp. 767–777.