GAINSPRINT: COORDINATING CONTROLLERS THROUGH ERROR

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES DIVISION
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
WILL KONG

CHICAGO, ILLINOIS
DECEMBER 2019
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>iv</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>v</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vi</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 MOTIVATION</td>
<td>3</td>
</tr>
<tr>
<td>3 RELATED WORK</td>
<td>5</td>
</tr>
<tr>
<td>3.1 Accuracy-aware applications</td>
<td>5</td>
</tr>
<tr>
<td>3.2 Mobile/embedded systems</td>
<td>5</td>
</tr>
<tr>
<td>3.3 Control coordination methods</td>
<td>6</td>
</tr>
<tr>
<td>4 GAINSPLIT</td>
<td>7</td>
</tr>
<tr>
<td>4.1 Notation</td>
<td>7</td>
</tr>
<tr>
<td>4.2 Control theory</td>
<td>7</td>
</tr>
<tr>
<td>4.3 The GainSplit framework</td>
<td>8</td>
</tr>
<tr>
<td>4.4 The GainSplit interface</td>
<td>10</td>
</tr>
<tr>
<td>5 EVALUATION</td>
<td>12</td>
</tr>
<tr>
<td>5.1 Experimental setup</td>
<td>12</td>
</tr>
<tr>
<td>5.2 Results</td>
<td>13</td>
</tr>
<tr>
<td>6 FUTURE WORK</td>
<td>18</td>
</tr>
<tr>
<td>7 CONCLUSION</td>
<td>19</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>20</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

4.1 GainSplit system diagram .............................................. 9
5.1 Metrics for the three methods across time .............................. 14
5.2 Power results .......................................................... 15
5.3 Performance results ..................................................... 15
5.4 Accuracy results ......................................................... 16
5.5 Average of results across all targets ................................. 16
LIST OF TABLES

4.1 Notation ........................................ 7
5.1 Controller configurations ...................... 12
ACKNOWLEDGMENTS

This project could not have been possible without the patient guidance of my advisor, Hank Hoffmann. I’d also like to thank my labmates, peers, and friends Anne Farrell, Ivana Marincic, and Saeid Barati for all of the advice and sanity checks they’ve given me over the course of this project and over the years.
ABSTRACT

Self-adaptation is a valuable property of computing systems, enabling them to reconfigure themselves to respond to dynamic events while meeting operational requirements. Control theory has become a popular framework for implementing adaptive systems because it provides a framework for formally reasoning about the dynamic behavior of such systems, but it poses a new challenge: how to coordinate the behaviors of independent adaptive systems. Several advanced control methods have been proposed to address this problem including supervisory control systems, structured singular value controllers, and cascading controllers. Unfortunately, all of these approaches require that all control designers are aware of all other controllers that may exist at design time, which is infeasible as they cannot anticipate the environments their controllers will run under.

To overcome the limitations of design time coordination, we propose GainSplit, an interface that allows for the coordination of independent controllers without coordination at design time by controlling the gain of a feature common to all feedback controllers, the error signal. The key insight is that for any combination of controllers, we can set their error gain values such that they produce the desired target values. We implement GainSplit on an Android system, with two independent controllers controlling a single application. We show that this method allows us to coordinate independent controllers to meet run-time goals while removing the need for design time coordination.
CHAPTER 1
INTRODUCTION

For almost two decades, self-adaptation has been recognized as a valuable property of computing systems, enabling them to reconfigure themselves to meet operational requirements in reaction to unpredictable dynamic events [13, 15]. Control theory has proven to be a popular methodology for implementing such adaptive systems because it provides a framework for formally reasoning about the controlled system’s dynamic behavior [8, 9, 10, 12, 17, 20, 21, 23]. Nonetheless, the control of computing systems can be difficult due to the innumerable combinations of hardware, applications, and user decisions. As adaptive computing systems based on control theory become increasingly popular, a new challenge emerges: how to coordinate the actions of independent adaptive systems.

Prior approaches to the problem have allowed for the stable coordination of computing system resources at various levels of abstraction [8, 12, 14, 17, 20, 21, 22]. However, these approaches require assumptions about the rest of the system to be made at the time of design in order to function. For computer systems, this approach is clearly infeasible as operating systems, hardware, and applications are often developed separately, and their developers cannot anticipate all of the possibilities that the modularity of computing systems allows for. Due to these factors, it is difficult to design controllers that can coordinate with other controllers at run-time without making strong assumptions about the controlled system.

GainSplit differs from previous approaches by not requiring the coordination of controller designs while still allowing for a system to be controlled to meet real-time targets. GainSplit achieves this by abstracting away the specifics of any particular controller down to the one feature all closed-loop feedback controllers have: the error signal. By setting the gain for the error signal of a controller, we are able to control the range of actions that the controller can take. In doing so, GainSplit allows us to coordinate the behaviors of
independent controllers to meet our overall power targets.

We implement this system with a navigation app on a OnePlus 2 phone running Android 6.0. The navigation app is controlled directly by two separate controllers: a system level dynamic voltage and frequency scaling (DVFS) controller and an application level GPS polling frequency controller, which have their error gain values set in real-time by the gain control interface. We compare GainSplit to two other approaches to demonstrate that GainSplit can control the system it is deployed on. We compare GainSplit to uncoordinated controllers to demonstrate that GainSplit avoids the oscillation of uncoordinated controllers, which indicates that GainSplit is able to control the system. We also compare GainSplit to CoAdapt [8], an approach that has been shown to provide stable system control in order to meet goals, to show that GainSplit performs at least as well as another state-of-the-art solution without the design assumptions necessary for the former.

This paper makes the following contributions: 1) we propose the coordination of independent feedback controllers using the error signal, which removes the need for design time coordination and 2) we evaluate GainSplit, an implementation of this concept.
CHAPTER 2

MOTIVATION

While control theoretic adaptive systems hold the promise of saving energy, improving performance, and providing benefits to computing systems in general, there are a few hurdles in the way of their implementation on live systems. In this chapter, we walk through some of the issues preventing control theoretic adaptive systems from being usable on live systems which include: 1) the difficulties in designing controllers themselves, 2) the bespoke nature of control systems, and 3) a lack of a coordinating framework for independent control systems.

The difficulties in designing controllers comes from many sources but to name a few: figuring out which programs benefit from making tradeoffs, finding actuators that can change the dimensions desired (e.g. power, performance, accuracy), generating the correct model of the system to be controlled, etc. While there are ways of making these individual tasks easier, the efforts made to automate the generation of controllers would hopefully solve them outright [5, 17].

Similarly, it is a nontrivial task to design controllers that are portable between different hardware systems or between different applications. Some of the proposed solutions to this problem include platform agnostic frameworks like POET [11] and the use of machine learning frameworks like LEO [19] and CALOREE [18] to approximate tradeoff spaces for new applications using data from known applications.

This paper proposes a solution for the third problem, the issue of coordinating separate control systems. As far as we are aware, all of the currently proposed approaches to this problem require some level of coordination between controller designers or some assumptions to be made about the settings the controllers are running under. Yukta requires the hand-tuning of several parameters at the time of controller design [20]. CoAdapt requires that a somewhat arbitrary decision be made in order to decide which controller
becomes the lead and which becomes the subordinate [8]. Similarly, Spectr requires user input in order to decide the weights that prioritize one controller over another [21]. These types of approaches are infeasible due to the innumerable combinations of hardware, software, and operating system that exist, meaning that assumptions made in one context are likely to be largely invalid in another. This points to the need for an approach that works in real-time like the three above, but does not require assumptions to be made on the part of the controller designer.
CHAPTER 3
RELATED WORK

3.1 Accuracy-aware applications

Accuracy-aware applications make the tradeoff between computational accuracy and power savings, improved performance, or other improvements [1, 2, 4, 6, 9]. As the concept of accuracy can vary from application to application, there are many approaches to accuracy-aware applications. MCDNN utilizes models with varying accuracies (among other techniques) in order to reduce battery strain on mobile devices [6]. MEANTIME is a framework that processes user input through a governor that then generates a schedule and interacts with a controller to meet deadlines [4]. The diversity in application-specific controllers introduces a need for controller coordination methods to ensure that accuracy-aware applications do not interact poorly with other adaptive systems.

3.2 Mobile/embedded systems

The limited energy resources and short-lived tasks of mobile/embedded systems [16] makes the dynamic reconfiguration of resources extremely useful. Much work has been done on systems with similar hardware architectures and/or operating systems to the system that we evaluate GainSplit on [2, 4, 10, 11, 16]. Drowsy implements a kernel-level power management state that utilizes the minimal task and device component set necessary to fulfill tasks, leaving all other tasks frozen and all other components in a low-power mode [16]. We use the Bard framework, a portable control and optimization framework, to implement the application and system level controllers as well as the GainSplit interface [10].
3.3 Control coordination methods

Control coordination methods vary widely but tend to rely on some set of heuristics along with user input in order to function [8, 12, 14, 17, 20, 21, 22]. Yukta is a coordination method that uses structured singular value (SSV) controllers that read signals from each other in order to coordinate their behaviors [20]. Kadjo et al. generate a monolithic state-space multiple-input multiple-output (MIMO) controller in order to coordinate CPU and GPU control in their system-on-a-chip (SoC) [12]. SPECTR utilizes supervisory control theory (SCT) and some automated synthesis in order to create a state-machine based supervisor that governs the behaviors of lower level controllers. We directly compare Gain-Split to CoAdapt, which coordinates controllers with different dimensions of control (e.g. power, performance, accuracy) by cascading the controllers such that the lead controller modifies the baseline values for a subordinate controller [8].
CHAPTER 4
GAINSPRINT

In this chapter, we describe how GainSplit coordinates independent controllers to meet system-wide power targets. We begin by explaining the notation that we will be using throughout this chapter. We then go through the details of the GainSplit framework.

4.1 Notation

Table 4.1 summarizes the notation that will be used throughout this chapter. We use the subscript \( x \) to refer to the different controllers within the system. \( x \) is 1 when the symbol is referring to the DVFS controller, 2 when the symbol is referring to the GPS controller, and GS when referring to the GainSplit controller.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_x(t) )</td>
<td>configuration selected by controller ( x )</td>
</tr>
<tr>
<td>( C_x )</td>
<td>configuration set</td>
</tr>
<tr>
<td>( e_x(t) )</td>
<td>controller error, equivalent to ( g(t) - f_x(t) )</td>
</tr>
<tr>
<td>( f_x(t) )</td>
<td>feedback</td>
</tr>
<tr>
<td>( g(t) )</td>
<td>target</td>
</tr>
<tr>
<td>( G_x(t + 1) )</td>
<td>controller error gain</td>
</tr>
<tr>
<td>( n )</td>
<td>time interval multiplier for the GainSplit controller</td>
</tr>
<tr>
<td>( N )</td>
<td>natural numbers</td>
</tr>
<tr>
<td>( T )</td>
<td>time interval between controller actions</td>
</tr>
<tr>
<td>( u_x(t + 1) )</td>
<td>control signal</td>
</tr>
</tbody>
</table>

4.2 Control theory

Control theory provides formal guarantees for the response of a dynamic system being controlled. This allows computing systems to meet real-time power goals while maxi-
mizing performance, accuracy, or other benefits [7]. The controllers that we use are based on the Bard framework [10] but are not a contribution of this paper. As GainSplit only relies upon a controller having an error signal $e_x(t)$, a property common to all (feedback) controllers, it should be possible to generalize GainSplit to most controller frameworks.

4.3 The GainSplit framework

We can describe the controllers in the system through a set of equations. Bard [10] implements simple proportional integral (PI) controllers, so we describe them using the canonical PI form. These controllers effectively become the actuators for the GainSplit controller.

$$
\begin{align*}
    u_1(t + 1) &= u_1(t) - e_1(t) \\
    u_2(t + 1) &= u_2(t) - e_2(t)
\end{align*}
$$

These controllers are then modified to expose their error signals so that GainSplit can control them through their respective gain values. GainSplit itself is also built on top of a Bard controller, so we represent it the same way as the others. The system as a whole is illustrated in Fig 4.1. Each controller in the system has a configuration set $C_x$ that we collect by sweeping over the range of possible configurations and aggregating the results. This is specific to Bard [10] and frameworks like it, and there are other methods of profiling systems such as the min/max sweep method described by Maggio et al. [17]. The GainSplit controller sweeps over a range of potential gain values, which range from $[0, 1]$. Although this range can be extended arbitrarily, this decision is made only at the level of GainSplit (effectively the OS level) and therefore does not necessitate coordination at the time of controller design. One important property to note here is that the ability for a gain value to be set to 0 effectively allows the GainSplit controller to freeze the other controllers, which means that they stop acting. This can be useful when a subset of the
overall set of controllers is able to meet the target without the frozen controllers.

\[
\begin{align*}
    u_1(t+1) &= u_1(t) - G_1(t) \cdot e_1(t) \\
    u_2(t+1) &= u_2(t) - G_2(t) \cdot e_2(t) \\
    u_{gs}(t+1) &= u_{gs}(t) - e_{gs}(t)
\end{align*}
\] (4.2)

Figure 4.1: GainSplit system diagram

After each time interval \( T \), the DVFS and GPS controllers each update their control signals and select a configuration \( c_x(t) \) out of their configuration sets for their actuator. Each actuator remains in the configuration selected by the controller for \( T \) time units. In some cases, the two controllers will update at the exact same time (e.g. if the time interval is based on discrete iterations) but they can be out of sync with each other if they are both operating in continuous time.
Everyone $nT$ time units, the GainSplit controller updates its control signal, where $n > 1$ and $n \in \mathbb{N}$. $n$ is selected to guarantee that the controllers respond to changes in gain before the next change is made. $n$ is selected somewhat arbitrarily, but similar to the gain range selected in the configuration sweeping process, this decision is made only at the level of GainSplit and does not necessitate coordination at the time of controller design. After the GainSplit controller updates its control signal, it selects the appropriate set of gain values from its configuration set and sets the error gain values of the other controllers. This process repeats indefinitely.

### 4.4 The GainSplit Interface

The GainSplit interface consists of two components: 1) functions that expose the error signals of controllers that we are interested in coordinating the behavior of and 2) a feedback controller that monitors the values that we are interested in and calculates the correct gain values to send to the controllers in the system in order to meet a target.

The functions that expose the error signals can be any that allow for the modification of the error signal by a gain value prior to the calculation of the input signals to the application(s) or piece(s) of hardware that the controllers control. This could be implemented in the form of a function that takes in a multiplier that is applied to the error signal.

```c
void calculate_input_signal(double gain) {
    double input_signal = current_input_signal - (gain * error)
}
```

The feedback controller needs to be a closed-loop feedback controller that works at the operating system level so that it can coordinate both application and hardware level controllers. The role of the gain controller is to calculate the correct gain values such that the application and hardware level controllers meet the target. As mentioned in Section
4.2, this should be generalizable to most controller frameworks as long as those frameworks use closed-loop control. In most systems where self-adaptation/feedback is critical, closed-loop control is the norm and so GainSplit should work with those frameworks.
CHAPTER 5
EVALUATION

5.1 Experimental setup

We evaluate GainSplit using a navigation application on a OnePlus 2 smartphone running Android 6.0. The application is a JavaScript based navigation application running in an Android WebView, with modifications to allow for GPS traces to be replayed and for performance measurements to be made. The OnePlus 2 contains a Qualcomm Snapdragon 810 system-on-chip, which has an eight-core ARM big.LITTLE processor that is split between four 1.77GHz big cores and four 1.56GHz LITTLE cores. In our evaluation, we use two controllers: an application level controller that modifies the GPS polling frequency and a system level controller that modifies the DVFS of the processor. The application level controller controls the tradeoff between power and location update frequency while the system level controller controls the tradeoff between power and rendering performance. The controller implementations are modifications of Bard, a portable implementation of a closed-loop feedback controller [10]. The change made to Bard that makes GainSplit possible is the exposure of Bard’s error signal, which allows GainSplit to make gain modifications to it. The two controllers use distinct configuration sets (See Table 5.1) generated independently of one another by running the application with various application and systems level resources.

In our evaluation, we compare GainSplit to two other methods: 1) an uncoordinated

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Settings</th>
<th>Speedup</th>
<th>Powerup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU core speeds</td>
<td>121</td>
<td>1.56</td>
<td>1.25</td>
</tr>
<tr>
<td>GPS polling frequencies</td>
<td>5</td>
<td>15</td>
<td>1.26</td>
</tr>
</tbody>
</table>
pair of the application and system level controllers described above and 2) the same pair of controllers using the CoAdapt framework [8] as their coordination method. These control systems are given various power goals covering the range of possible power values of the system as a whole and their excess power usage, rendering performance, and GPS position update times are measured against each other over.

5.2 Results

We compare GainSplit with the uncoordinated controllers and CoAdapt on three metrics: power, performance, and accuracy. For our power metric, we take the mean percentage of deviations from the target caused by overshoot. While undershoot can occur, we only concern ourselves with overshoot because it violates the power cap set by our target. To measure performance, we look at the mean percentage of frames rendered below 60 frames per second. This is the suggested rendering speed in the Android developer documentation due to slower rendering speeds being perceived as stuttering by users [3]. Our accuracy metric is the mean of GPS position update times across a given trial. We chose this metric because delays in GPS update times can cause users of navigation applications to navigate incorrectly. We examine these metrics both in aggregate across trials and over time for individual trials. In the time series data, we are checking for oscillations in power, performance, and accuracy as oscillations in those metrics would indicate that the method in question is not controlling the system. In the aggregate data, we are checking for comparable performance between our two coordinated methods.

To get these metrics, we run trials that consist of running the navigation application while playing back a GPS trace to simulate movement. We run multiple trials with the three control methods using varying power targets to compare how the three methods behave with different goals.

The time series data for the three different methods shows what we would expect:
that the uncoordinated controllers are unable to control the system because they oscillate around the power target, but that CoAdapt and GainSplit do not oscillate. Fig. 5.1 shows that while CoAdapt and GainSplit remain fairly stable over time, the uncoordinated controllers start to oscillate as time passes.

![Figure 5.1: Metrics for the three methods across time](image)

If we look at the breakdown of the three metrics separated by power targets, the picture becomes a bit more clear. Fig. 5.2 shows that the power usage for all three methods is fairly similar. However, in the performance and accuracy results shown in Figs. 5.3 and 5.4, the uncoordinated controllers perform relatively worse as the power target increases. This indicates that the oscillation of the uncoordinated controllers that we saw in Fig. 5.1 results in hovering around an overall lower power range with the result being their overall worse performance and accuracy numbers. CoAdapt and GainSplit appear to have
fairly similar results although it is a bit hard to compare them in this form.

The aggregate of the results for our three metrics is given in Fig. 5.5. They confirm the overall trends that we found in examining the behaviors of the three methods with
varying power targets. Comparing the rendering performance and the GPS update times we can see that while they all have similar excess power values, the uncoordinated controllers perform worse on average than CoAdapt or GainSplit. We can also see that the
results of CoAdapt and GainSplit are extremely similar, which suggests that GainSplit is a viable alternative to CoAdapt.

In our evaluation, we have found that GainSplit does not oscillate like the uncoordinated controllers and has similar results to CoAdapt. These results demonstrate that GainSplit: 1) can control systems using the error signal while 2) remaining competitive when compared to a similar state-of-the-art method. Taken as a whole, they show that it is possible to coordinate the behaviors of separate controllers without the necessity of coordination at the time of controller design.
CHAPTER 6

FUTURE WORK

While we have demonstrated that coordinating self-adaptive systems through their error signals is viable, more work is necessary to develop the idea further.

First, we will need to demonstrate the viability of GainSplit with more applications. We chose a navigation application due to the ubiquity of GPS-based applications on mobile devices. Due to the difficulty of controlling systems that interact with physical environments and the noisiness of GPS-based applications in general [2], we are confident that GainSplit will be able to control self-adaptive systems while running relatively simpler applications (e.g. video encoding/decoding). Nonetheless, this needs to be shown in practice.

Similarly, we will need to demonstrate the viability of GainSplit with various feedback controller frameworks to demonstrate its generalizability.

Following this, we would want to demonstrate the viability of GainSplit at scale. This would entail running GainSplit with two or more self-adaptive applications running simultaneously alongside at least one hardware level controller, running GainSplit with two or more hardware level controllers with a self-adaptive application, etc.

Finally, it would be useful to create a method to handle changes in computing environments (e.g. the user closing an application, how to handle new applications, etc).
CHAPTER 7
CONCLUSION

GainSplit is a control theoretic controller coordination framework that can coordinate the resources across different levels of a computing system to meet real-time goals. Unlike prior proposed frameworks, GainSplit does not require any coordination at design time. Our results demonstrate that GainSplit can coordinate the self-adaptive systems of a real-time computing environment as well as a state-of-the-art method, making it a viable alternative. We believe that this makes it an important contribution towards the goal of implementing control theoretic systems in practice.
REFERENCES


