Time Dilation and Contraction for Programmable Analog Devices with Jaunt

Time Dilation and Contraction for Programmable Analog Devices with Jaunt

Sara Achour
MIT EECS & CSAIL
sachour@csail.mit.edu

Martin Rinard
MIT EECS & CSAIL
rinard@lcs.mit.edu

Abstract

Programmable analog devices are a powerful new computing substrate that are especially appropriate for performing computationally intensive simulations of neuromorphic and cytomorphic models. Current state of the art techniques for configuring analog devices to simulate dynamical systems do not consider the current and voltage operating ranges of analog device components or the sampling limitations of the digital interface of the device.

We present Jaunt, a new solver that scales the values that configure the analog device to ensure the resulting analog computation executes within the operating constraints of the device, preserves the recoverable dynamics of the original simulation, and executes slowly enough to observe these dynamics at the sampled digital outputs. Our results show that, on a set of benchmark biological simulations, 1) unscaled configurations produce incorrect simulations because they violate the operating ranges of the device and 2) Jaunt delivers scaled configurations that respect the operating ranges to produce correct simulations with observable dynamics.

CCS Concepts • **Computer systems organization → Analog computers**; • **Hardware → Emerging languages and compilers**; • **Applied computing** → *Systems biology*; • **Software and its engineering** → Domain specific languages;

Keywords Compilers; Analog Computing

ACM Reference Format:
Sara Achour and Martin Rinard. 2018. Time Dilation and Contraction for Programmable Analog Devices with Jaunt. In *ASPLOS '18: 2018 Architectural Support for Programming Languages and Operating Systems, March 24–28, 2018, Williamsburg, VA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3173162.3173179

1 Introduction

Programmable analog devices provide a powerful new substrate for performing computationally intensive simulations of neuromorphic and cytomorphic models [5, 9, 10, 26, 28, 29, 31, 32, 36]. To simulate a model, the model parameters, state, and transient variables map to voltages and currents in the analog device.

The biological dynamical systems these programmable analog devices target are used for optimizing medical doses, predicting diseases, and understanding biological phenomena such as cellular pathways [19, 30]. To make these simulations tractable on digital systems, these models often oversimplify the dynamics of the biological system [7, 13], which reduces the fidelity of the model and renders the simulation less faithful to the system it is designed to capture [13, 34]. Because the simulation runs on a digital platform, time is discretized and systems with fast and slow dynamics (such as many biological systems) are prone to instability unless the time step is small, which can significantly extend the simulation time.

Analog devices, in contrast, perform simulations in the continuous time domain, which negates may of the time scale issues from which discrete solvers suffer. Analog devices can also efficiently implement complex, domain-specific basis functions and deliver significant energy consumption benefits [32, 35]. Analog devices therefore have the potential to productively support more sophisticated stochastic simulations of biological phenomena that currently do not perform well on digital hardware [35].

In recent years, there have been advances in compiler techniques that target reconfigurable analog devices devices [2, 3, 16, 22]. The Arco compiler, for example, takes as input a specification of a reconfigurable analog device and a specification of a dynamical system and produces an analog device configuration that simulates the dynamical system [2]. Arco treats the available analog components as idealized computational blocks that implement abstract algebraic functions. But analog components are physical hardware components with limitations that a compiler must consider when configuring the analog device:

- **Operating Ranges:** In our target analog devices, physical properties such as voltage and current have *operating ranges* in the form of minimum and maximum values within which the voltage or current must fall for the device to operate properly. If these operating ranges are violated, the component may become damaged or fail to perform the computation to specification.
- **Digital-Analog Interface:** Although the simulation executes continuously, the digital to analog converters and analog to digital converters through which the simulation is configured and observed are clocked. If the analog simulation executes too quickly, the dynamics may not be observable in the sampled digital output stream collected from the digital to analog converters.

We present Jaunt, a solver which starts with an initial unscaled analog device configuration, then scales the values that configure the device to ensure that the resulting scaled simulation is:

- **Physically Realizable**: Each of the signals in the configured analog device stays within the operating range of the analog component that carries the signal.
- **Recoverable**: The original simulation can be recovered by scaling the magnitudes and times recorded at the sampled digital outputs by derived scaling factors.
- **Observable**: The simulation executes slowly enough to observe the transient dynamics in the sampled digital output streams.

Because all of these properties interact, Jaunt formulates the problem of obtaining a physically realizable, recoverable, and observable scaled simulation as a constraint satisfaction problem (specifically, a geometric program). The solution to this problem

ASPLOS '18, March 24–28, 2018, Williamsburg, VA, USA

ACM ISBN 978-1-4503-4911-6/18/03.
https://doi.org/10.1145/3173162.3173179

delivers *scaling factors* that, when applied to the digital inputs and outputs of the analog device, deliver a physically realizable, recoverable, and observable simulation. Explicitly including time in the constraint problem formulation enables Jaunt to appropriately contract and dilate the speed of the analog simulation to satisfy the constraints. In particular, dilating or contracting time can influence the physical realizability and recoverability of the simulation by inducing cascading changes to other device configuration values and analog signals. And explicitly including time in the constraint problem formulation enables Jaunt to control the observability of the simulation and optimize for desired simulation durations.

Implementation and Results: We implement Jaunt in concert with a modified version of the Arco compiler [2]. Arco delivers analog device configurations that simulate the specified dynamical system assuming an idealized analog device with no operating range constraints. Jaunt delivers scaling factors that ensure that the scaled configuration respects specified observability and analog device operating range constraints while preserving the recoverability of the simulation. Our experimental results show that, for our set of benchmark biological systems running on our target reconfigurable analog device, the unscaled Arco configurations often violate the operating range constraints to produce incorrect simulations. The Jaunt scaled configurations, in contrast, successfully deliver correct simulations with reasonable combined Arco/Jaunt compilation and simulation times.

Contributions: We claim the following contributions:

- **Basic Approach**: We present a new approach for scaling the values supplied to a configured analog device so that the resulting scaled, configured analog device operates within the operating constraints of the hardware. The approach ensures the original dynamical system simulation can be recovered from the scaled execution by multiplying the samples read from the analog device by appropriate constant factors. Using this approach, we are able to dilate or contract simulation time on the analog device to satisfy observability and simulation time constraints.
- **Geometric Programming Problem Generator**: We present a constraint generation algorithm that ensures that any satisfying scaling transform renders the configuration physically realizable and recoverable. We formulate the constraint problem as a geometric programming problem, which is then converted to a convex optimization problem and solved using a convex solver. We are able to optimally compute the scaling transforms that result in the fastest and slowest simulations and definitively determine when no Jaunt scaling factors can make the configured analog device physically realizable.
- **Optimizations**: We present a variable aggregation optimization for reducing the number of variables in the geometric programming problem and a pruning optimization for improving the efficacy of the configuration search algorithm. The pruning optimization removes search paths that produce configurations that cannot be made physically realizable.
- **Results**: We present experimental results that characterize the effectiveness of Jaunt in finding scaling factors that deliver physically realizable, recoverable, and observable scaled configurations.

To deliver these contributions, Jaunt works with an analog hardware specification language that includes operating range specifications and a dynamical system specification language that includes speed and sampling constraints over the scaled simulation. To successfully map the computation onto the analog device and respect the inherent underlying hardware operating range and sampling frequency constraints, Jaunt manipulates the values of state and transient variables written to and read from the digital interface of the analog device and the speed with which the device simulates the dynamical system.

$$
\begin{array}{rcll}
E & = & 6800 - ES & E \in [0, 6800] \\
S & = & 4400 - ES & S \in [0, 4400] \\
\partial ES/\partial t & = & 0.0001 \cdot E \cdot S - 0.01 \cdot ES & ES \in [0, 4400] \\
ES_0 & = & 0 &
\end{array}
$$

Figure 1. Enzyme-Substrate Dynamical System

2 Example

Consider an enzyme-substrate reaction in which an enzyme E bonds to a substrate S to yield the compound *ES*, where *ES* may dissociate into *E* and *S*. The compound *ES* is formed at the rate k_f (0.01) and dissociates at the rate k_r (0.0001):

$$E + S \rightarrow_{k_f} ES \qquad\qquad ES \rightarrow_{k_r} E + S$$

We model this chemical reaction using a dynamical system with one differential equation and two first-order equations, where the state variables *E*, *S*, and *ES* correspond to the number of molecules of each compound available at any point of time.

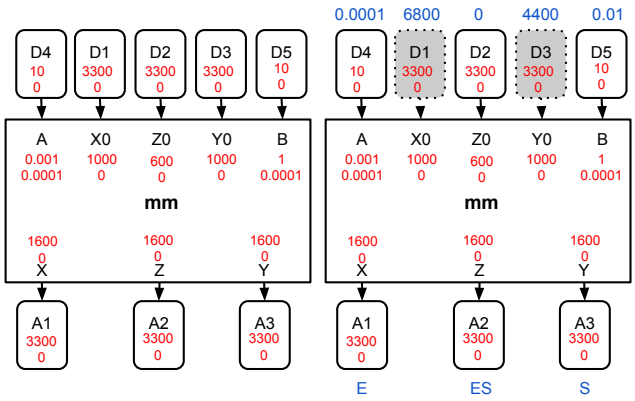
Figure 1 presents the dynamical system that models the enzyme-substrate reaction, including the dynamic range of each quantity. Here 6800 is the total amount of *E* and 4400 is the total amount of *S* available. We use physical properties of the chemical reaction to derive the dynamic range of each state variable:

- **Physical Quantity**: Each state variable represents a physical quantity and is therefore greater than or equal to 0.
- **Conservation of Mass**: The maximum value of each state variable is limited by the principle of conservation of mass. The amount of *E* and *S* cannot exceed the starting amounts (6800 and 4400 molecules). Because *ES* is formed from *E* and *S*, the amount of *ES* cannot exceed 4400 molecules, the amount of the limiting reagent (*S*).

2.1 Initial Configuration of the Analog Device

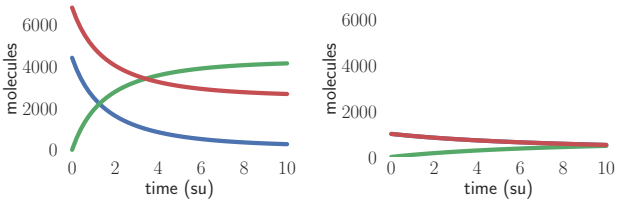
Figure 2a presents a reconfigurable analog device. The components of this device are inspired by existing hardware [9, 10, 28, 32, 36]. The device contains an `mm` computational block, three voltage DACs (`D1`-`D3`), two current DACs (`D4` and `D5`), and three voltage ADCs (`A1`-`A3`). The `mm` block interface contains analog input ports `A`, `X0`, `Y0`, `Z0`, and `B` (top edge of `mm` block) and analog output ports `X`, `Z`, and `Y` (bottom edge of `mm` block). Each DAC, ADC, and port in the `mm` block has an operating range (red text) consisting of an upper bound on the signal (top number) and a lower bound on the signal (bottom number). As long as the signal at each port falls within the port's operating range, the `mm` block implements the following dynamics:

$$
\begin{array}{ll}
X = X0 - Z & Z = \int (A \cdot X \cdot Y - B \cdot Z)\ \frac{\partial Z}{\partial t} \\
Y = Y0 - Z & Z_0 = Z0
\end{array}
$$



(a) Analog Device **(b)** Configured Analog Device

Figure 2. Configured Analog Device



(a) Reference Simulation **(b)** Analog Device Simulation

Figure 3. Simulation on Configured Analog Device

Given a specification of the reconfigurable analog device and a specification of the enzyme-substrate dynamical system, the Arco compiler [2] generates the analog device configuration presented in Figure 2b. In this configuration, the starting amounts of E (6800), S (4400), ES (0), and the values of k_f (0.01) and k_r (0.0001), are presented to the circuit via input DACs `D1`–`D5`. The resulting configured `mm` block simulates the reaction dynamics to produce the values of S, ES, and E on ADCs `A1`–`A3`. These ADCs are sampled to obtain the sequence of values that S, ES, and E take on over time as the device simulates the reaction.

Figure 3b presents the resulting simulation on the configured analog hardware (Figure 3a presents the correct reference simulation). Both figures plot the amount of E (red line), S (blue line), and ES (green line) as a function of time. The simulation is executed on the configured analog device by powering on the device and sampling the ADCs.

Because the Arco compiler does not take the operating ranges of the analog device into account when generating the configuration, some of the ports violate their operating ranges and the resulting simulation (Figure 3b) deviates significantly from the reference simulation (Figure 3a). In particular, the constants 6800 and 4400 supplied to `D1` and `D3` (shaded blocks in Figure 2b) exceed 1000 mV, the maximum allowable voltage for ports $X0$ and $Y0$ in the `mm` component, and the resulting anomalies propagate through the circuit to produce an incorrect simulation.

2.2 Jaunt

Given a configuration of the analog device (such as the configuration presented in Figure 2a), Jaunt finds a set of *port scaling factors* and a *time scaling factor* that together ensure the simulation is:

- **Physically Realizable**: After scaling the values at the (digital) input ports by their port scaling factors, all ports in the configured device execute within their operating ranges.
- **Recoverable**: Scaling the signals recorded at the digital output ports by the their port and time scaling factors correctly inverts the scaling transform to recover the original simulation.

In the example, Jaunt controls the signals in the analog device in two ways:

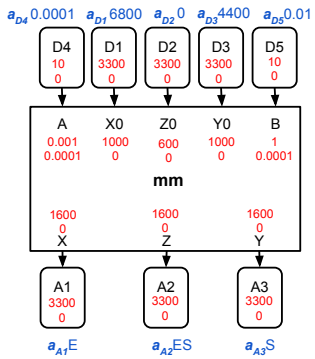
- **Scaling State Variables**: Jaunt scales the initial values of the state variables in the differential equation, which increases or decreases the magnitudes of the state variables in the resulting simulation. In this example, Jaunt scales the initial values E_0 (6800) on `D1`, ES_0 (0) on `D2`, and S_0 (4400) on `D3`.
- **Scaling Time**: Jaunt scales the speed of the differential equation to obtain a faster or slower simulation. Changing the simulation speed causes corresponding changes to the rate constants in the differential equation, in this example k_f (0.0001 on `D4`) and k_r (0.01 on `D5`).

Jaunt formulates the problem of finding a set of port scaling factors and a time scaling factor that render the simulation physically realizable and recoverable as a constraint satisfaction problem. Specifically, Jaunt generates constraints that 1) ensure the signals in the simulation remain within their corresponding port ranges and 2) deliver ADC scaling factors that recover the original simulation. Figure 4a shows DAC scaling factors $a_{D1} \ldots a_{D5}$ and ADC scaling factors $a_{A1} \ldots a_{A3}$, (the time scaling factor τ and scaling factors a_p for the internal analog ports p are not shown). Figures 4b, 4c, and 4d present relevant operating range, connection, and factor constraints. Jaunt also supports sampling constraints (which ensure that the simulation executes slowly enough for the transient dynamics to be observable in the sampled ADC outputs) and speed constraints (which limit how slowly the device executes the simulation).

Together, these constraints, along with an objective function over one or more of the scaling factors, form a *geometric program* [6, 24], a type of optimization problem that can be converted to a convex optimization problem and solved using an off-the-shelf solver. The variables in the geometric program are the port scaling factors a_p and the time scaling factor τ. The solution to the geometric program produces port and time scaling factors that Jaunt then uses to transform the analog device configuration.

Operating Range Constraints: Figure 4c presents constraints that ensure the dynamic range of each scaled signal falls within the operating range of the port that carries that signal. Jaunt generates these constraints by computing the upper and lower bounds of the signal in the original (unscaled) configuration, then imposing the constraint that the scaled upper and lower bounds of the signal must fall within the operating range of the port that carries that signal.

In our example, each voltage DAC is able to generate voltages between 0 and 3300 millivolts. Therefore, the scaled values $a_{D1} \cdot 6800$, $a_{D2} \cdot 0$ and $a_{D3} \cdot 4400$ must fall within the operating range $[0, 3300]$ for the scaled simulation to be physically realizable. From the dynamical system specification, we know that the dynamic range of E (which is carried on the analog `Y` port of the `mm` block)



(a) Scaling Transform

$$
\begin{aligned}
a_{D4} &= a_A & a_{D5} &= a_B \\
a_{D1} &= a_{X0} & a_{D2} &= a_{Z0} \\
a_{D3} &= a_{Y0} & a_{A1} &= a_X \\
a_{A2} &= a_Z & a_{A3} &= a_Y
\end{aligned}
$$

(b) Connection Constraints

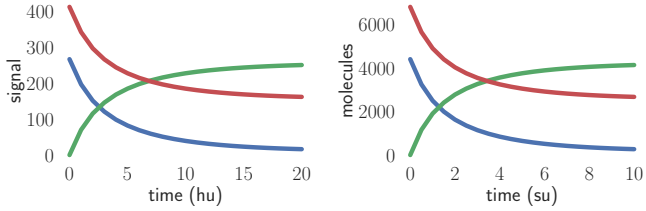
$$
\begin{aligned}
a_{D4} \cdot 0.01 &\in [0,10] \\
a_{D5} \cdot 0.0001 &\in [0,10] \\
a_{D1} \cdot 6800 &\in [0,3300] \\
a_{D2} \cdot 0 &\in [0,3300] \\
a_{D3} \cdot 4400 &\in [0,3300] \\
a_B \cdot 0.01 &\in [0.0001,1] \\
a_A \cdot 0.0001 &\in [0.0001,0.01] \\
a_{X0} \cdot 6800 &\in [0,1000] \\
a_{Z0} \cdot 0 &\in [0,600] \\
a_{Y0} \cdot 4400 &\in [0,1000] \\
a_X \cdot [0, 6800] &\subseteq [0,1600] \\
a_Z \cdot [0, 4400] &\subseteq [0,1600] \\
a_Y \cdot [0, 4400] &\subseteq [0,1600] \\
a_{A1} \cdot [0, 6800] &\subseteq [0,3300] \\
a_{A2} \cdot [0, 4400] &\subseteq [0,3300] \\
a_{A3} \cdot [0, 4400] &\subseteq [0,3300]
\end{aligned}
$$

(c) Operating Range Constraints

$$
\begin{aligned}
a_{X0} &= a_Z & a_X &= a_Z \\
a_{Y0} &= a_Z & a_Y &= a_Z \\
& a_A \cdot a_X \cdot a_Y = a_B \cdot a_Z & & \\
a_Z &= a_B \cdot a_Z / \tau & a_{Z0} &= a_B \cdot a_Z / \tau
\end{aligned}
$$

(d) Factoring Constraints

Figure 4. Scaled Configuration and Scaling Transform Constraints



(a) Scaled Signal Simulation

(b) Recovered Simulation

Figure 5. Simulation on Scaled Configured Analog Device

falls within $[0, 4400]$ molecules. Because the operating range of the Y port is $[0, 1600]$ millivolts, the scaled dynamic range $a_Y \cdot [0, 4400]$ of E must fall within $[0, 1600]$.

Connection Constraints: Because two connected ports carry the same signal, the scaling factors of any two connected ports must be the same. Figure 4b presents the connection constraints in our example.

Factor Constraints: Figure 4d presents constraints that ensure the scaling transform can be factored out of each output of the mm block. These constraints ensure the original simulation can be recovered from the scaled execution by inverting the scaling factors at each ADC. Jaunt derives the factor constraints by analyzing the propagation of the scaling factors through the (potentially nonlinear, nonconvex) dynamics of the device. The dynamics of the mm block are as follows:

$$
\begin{aligned}
X &= X0 - Z & Y &= Y0 - Z \\
Z &= \int (A \cdot X \cdot Y - B \cdot Z)\ \frac{\partial Z}{\partial t} & & \\
Z_0 &= Z0 & &
\end{aligned}
$$

Applying the scaling factors produces the following scaled dynamics:

$$
\begin{aligned}
a_X X &= a_{X0} X0 - a_Z Z & a_Y Y &= a_{Y0} Y0 - a_Z Z \\
a_Z Z &= \int (a_A A \cdot a_X X \cdot a_Y \cdot Y - a_B B \cdot a_Z \cdot Z)\ \frac{\partial Z}{\tau \partial t} & & \\
a_Z Z_0 &= a_{Z0} Z0 & &
\end{aligned}
$$

The scaled dynamics, which scale ∂t by τ (here ∂t is measured in units of hardware time), execute the differential equation at speed τ: with the time scaling factor τ applied, integrating over one unit of hardware time integrates over τ units of simulation time.

The factor constraints in Figure 4d capture several key properties required to correctly recover the original simulation from the scaled execution. The constraints $a_{X0} = a_Z$, $a_{Y0} = a_Z$, and $a_A \cdot a_X \cdot a_Y = a_B \cdot a_Z$ ensure that the operands of sums or differences have identical scaling factors. The constraints $a_X = a_Z$, $a_Y = a_Z$, and $a_Z = a_B \cdot a_Z / \tau$ ensure that the scaling factor for each port matches the scaling factor of the expression that defines the behavior of the signal carried on that port. The constraint $a_{Z0} = a_B \cdot a_Z / \tau$ ensures that the scaling factor of the initial value of a state variable matches the scaling factor of the expression that defines subsequent values of that state variable.

Together, these constraints ensure that it is possible to recover the original dynamics from the scaled dynamics by factoring out a single number from each equation that defines the scaled dynamics to recover the equations in the original unscaled dynamics. Specifically, because $a_{X0} = a_Z$, $a_X = a_Z$, $a_{Y0} = a_Z$, and $a_Y = a_Z$:

$$
\begin{aligned}
[a_X X = a_{X0} X0 - a_Z Z] &\quad \rightarrow \quad a_Z [X = X0 - Z] \\
[a_Y Y = a_{Y0} Y0 - a_Y Z] &\quad \rightarrow \quad a_Z [X = Y0 - Z]
\end{aligned}
$$

Because $a_Z = a_A \cdot a_X \cdot a_Y / \tau = a_B \cdot a_Z / \tau$:

$$
\begin{aligned}
&[a_Z Z = \int (a_A A \cdot a_X X \cdot a_Y \cdot Y - a_B B \cdot a_Z \cdot Z)\ \frac{\partial Z}{\tau \partial t}] \rightarrow \\
&a_Z [Z = \int (A \cdot X \cdot Y - B \cdot Z)\ \frac{\partial Z}{\partial t}]
\end{aligned}
$$

And because $a_Z = a_{Z0}$:

$$
[a_Z Z_0 = a_{Z0} Z0] \quad \rightarrow \quad a_Z [Z_0 = Z0]
$$

From the factored dynamics, it is evident that ports X, Y, and Z have the same scaling factor, specifically a_Z, where $a_Z = a_A \cdot a_X \cdot a_Y / \tau$. We can rewrite the scaling expression $a_Z = a_A \cdot a_X \cdot a_Y / \tau$ as $a_Z = a_A \cdot a_Z \cdot a_Z / \tau$, which implies $a_A = \tau / a_Z$. Because $a_B \cdot a_Z / \tau = a_Z$, $a_B = \tau$.

Objective Function: Jaunt supports objective functions that minimize τ, which slows down the simulation to maximize the sampled observability of the transient dynamics, or maximize τ, which speeds up the simulation to minimize the execution time. We can analytically compute bounds on the time scaling factor τ in our example as follows:

- **Lower Bound on** τ: The time scaling factor τ must be greater than 0.01 since $a_B = \tau$ and $a_B \geq 0.01$. We derive the lower bound on τ by applying the derived factor constraint $a_B = \tau$ to the range constraint $a_B \cdot 0.01 \in [0.0001, 1]$ ($\tau \cdot 0.01 \geq 0.0001$).
- **Upper Bound on** τ: The time scaling factor τ must be less than 14.7 since $a_A = \tau / a_Z$, $a_Z \leq 0.147$ and $a_A \leq 100$. We derive the upper bound on τ by applying the constraint $a_A = \tau / a_Z$ to the $a_A \leq 100$ constraint ($\tau / a_Z \leq 100$), then choosing the largest value of a_Z and solving for tau ($\tau / 0.147 \leq 100$). We derive $a_A \leq 100$ from the range constraint $a_A \cdot 0.0001 \in [0.0001, 0.01]$ ($a_A \cdot 0.0001 \leq 0.01$). We derive the upper bound of a_Z by applying the factor constraint $a_{X0} = a_Z$ to the range constraint $a_{X0} \cdot 6800 \in [0, 1000]$ ($a_Z 6800 \leq 1000$).

Jaunt is able to match these analytical bounds experimentally to compute scaling factors for any value of τ between 0.01 and 14.7 (the solution space is convex). Jaunt is therefore able to scale the

original Arco configuration to execute the simulation anywhere between 100 times slower ($\tau = 0.01$) to 14.7 times faster ($\tau = 14.7$), while still delivering scaling factors that ensure that the simulation is physically realizable and recoverable.

2.3 Scaled Configuration

Figure 5a presents the analog signals observed at each ADC with the Jaunt scaling transform applied to each DAC. For this simulation, the developer specified a sampling constraint ($\tau \leq 0.5$) that requires the time between ADC samples to be at most 0.5 simulation units (the ADCs are sampled once per hardware time unit). With this constraint, Jaunt scales time by a factor of 0.5x ($\tau = 0.5$) and the initial quantities of E, S, and ES by a factor of 0.06 (a_Z). To preserve the dynamics of the simulation, Jaunt also scales k_f by 8.28x (a_A) and k_r by 0.5x (a_B). The resulting scaled simulation executes within the operating constraints of the device.

Figure 5b presents the simulation recovered from the sampled signals by dividing each sample read from ADCs `A1-A3` by 0.06 (a_Z) and multiplying the hardware time of each sample by the time scaling factor ($\tau = 0.5$). The resulting recovered simulation matches the reference simulation in Figure 3a.

The chosen scaling transform executes the simulation on the device 2x *slower* ($\tau = 0.5$) than a real-time simulation ($\tau = 1$). At this speed, the transient dynamics are observable in the sampled signals, the simulation executes for 20 hardware time units (hu) instead of 10 hu, and takes 0.04 seconds instead of 0.02 seconds of wall clock time to complete (1 hu is 2 ms of wall clock time).

Without a sampling constraint, the simulation can execute 14.7x faster and finish in 0.68 hardware time units, or 0.0009 seconds of wall clock time. At this speed, the simulation executes too quickly for the ADCs to record the transient dynamics, but the final values of E, S, and ES (the steady state dynamics) remain observable.

3 Background

Arco and Jaunt, working together, produce configurations for a reconfigurable analog device. Each configuration models a specified dynamical system. The configured analog device, when powered on, executes a simulation of the dynamical system.

Analog Device Specification: The analog device contains input ports IP and output ports OP, with $HP = IP \cup OP$ the ports in the device. The signal at each output port $op \in OP$ is a (potentially nonlinear, nonconvex) function of the signals at the ports $hp \in HP$. The algebraic port expression $e_{hp} = R(op) \in E_{HP}$ defines the signal at output port $op \in OP$. To ensure the dynamics hold, the signal for each port hp must fall within its specified operating range interval $\langle x, y\rangle = I(hp)$, where $\langle x, y\rangle \in L$ and $I : HP \to L$. The analog device is reconfigurable and has a set $HZ \subseteq OP \times IP$ of available programmable connections between input ports and output ports.

The analog device functions as a co-processor. The main computer provides values to the analog device through digital input ports DI attached to digital-to-analog converters (DACs), then records the simulation by sampling digital output ports DO attached to analog-to-digital converters (ADCs). $DP = DI \cup DO$ is the set of digital ports.

The DACs and ADCs operate synchronously in accordance to a clock. Although the simulation runs in real time, the sampling rates of the DACs and ADCs affect the granularity of the sampled data points in the simulation. The function $G : DP \to \mathbb{R}_+$ maps each digital ports $dp \in DP$ to its time between samples.

	hardware specification
$\mathbb{R}_+ = \{x \in \mathbb{R} \mid x > 0\}$	positive real numbers.
$L = \{\langle x, y\rangle \in \mathbb{R}^2 \mid x \leq y\}$	an interval of two real numbers
$ip \in IP$	input ports in hardware specifiation
$op \in OP$	ouput ports in hardware specification
$hp \in HP = IP \cup OP$	hardware ports in specification
$dp \in DP \subseteq HP$	hardware ports in the digital interface
$di \in DI \subseteq IP$	digital inputs to DACs
$do \in DO \subseteq OP$	digital outputs from ADCs
$HZ \subseteq OP \times IP$	available connections
$e_{hp} \in E_{HP}$	expressions over hardware ports.
$R : OP \to E_{HP}$	maps output ports to port dynamics
$G : DP \to \mathbb{R}_+$	maps digital ports to sample periods
$I : HP \to L$	maps hardware ports to operating ranges
	dynamical system specification
$iv \in IV$	input variables
$ov \in OV$	output variables
$v \in V = IV \cup OV$	variables
$e_v \in E_V$	expressions over variables
$J : V \to L$	maps variables to dynamic ranges.
τ_{min}	minimum speed
λ_{max}	maximum time between samples.
	analog configuration
$p \in P \subseteq HP$	hardware ports used in the configuration
$z \in Z \subseteq HZ$	hardware connections used in configuration
$e_p \in E_P \subseteq E_{HP}$	expressions over ports in configuration
$F : P \to E_V$	expression each port models
$Q = \langle P, Z, F\rangle$	analog device configuration
	geometric program
$a_p \in A$	port scaling factors
$\tau \in A$	time scaling factor (simulation speed)
$m \in M$	monomials composed of scaling factors.
$s \in S$	posynomials composed of scaling factors.
$c \in C$	geometric program constraints.
s_{opt}	posynomial of scaling factors to minimize
$X = \langle s_{opt}, C\rangle$	geometric program.

Table 1. Notation Quick Reference

Dynamical System Specification: The dynamical system specification includes input variables IV and output variables OV, with $V = IV \cup OV$ the set of variables in the dynamical system. The value of each output variable $ov \in OV$ is a (potentially nonlinear, nonconvex) function over the variables $v \in V$ defined by an expression $e_v \in E_V$. At any point in the simulation, the value of each variable $v \in V$ falls within the interval $\langle x, y\rangle = J(v)$, where $\langle x, y\rangle \in L$ and $J : V \to L$. The dynamical system specification may also include a minimum speed τ_{min} and maximum time between samples λ_{max}.

Analog Device Configuration: An analog device configuration $Q = \langle Z, P, F\rangle$ includes connections $Z \subseteq HZ$, active ports $P \subseteq HP$, and a function $F : P \to E_V$ that maps each port $p \in P$ to the expression $e_v \in E_V$ that defines the value of the signal carried on the port.

Interaction Between Arco and Jaunt: Given a dynamical system, Arco generates a sequence of candidate unscaled configurations that map the dynamical system onto the device. Jaunt processes each candidate unscaled configuration in turn until it finds a configuration and scaling factors that deliver a scaled, physically realizable, and recoverable simulation. Jaunt is implemented as an extension to the Arco compiler and accepts an analog device configuration,

a dynamical system specification with sampling and speed constraints, and an analog device specification with operating ranges and sampling rates.

4 The Jaunt Solver

Given $\langle R, G, I\rangle$ from the hardware specification, $\langle J, \tau_{min}, \lambda_{max}\rangle$ from the dynamical system specification, and an analog device configuration $Q = \langle Z, P, F\rangle$, Jaunt finds a positive scaling factor for each port $p \in P$ and a simulation time scaling factor such that the configuration is physically realizable and recoverable.

4.1 Geometric Program

Jaunt formulates the problem of finding a physically realizable, recoverable scaled configuration as a geometric program, which consists of an objective function s_{opt} and constraints C over the geometric program variables A. There is a geometric program variable $a_p \in A$ for each port $p \in P$ and the time scaling geometric program variable $\tau \in A$. C may contain inequality constraints $m_i \leq 1 \in C$ and equality constraints $s_j = 1 \in C$:

$$\begin{gathered} minimize\ s_{opt} \\ m_i \leq 1, i = 1, ..., n \\ s_j = 1, j = 1, ..., m \end{gathered}$$

Here the m_i are *monomials* of the following form, where $c_i \in \mathbb{R}_+$ (positive real numbers), $a_p, \tau \in A$, and $x_{i,p}, x_{i,\tau} \in \mathbb{R}$:

$$m_i = c_i \tau^{x_{i,\tau}} \prod_{p \in P} a_p^{x_{i,p}}$$

Monomials are closed under multiplication and exponentiation to a real number, and are therefore closed under division by extension. Any positive nonzero constant is a monomial. The s_j are *posynomials* (sums of monomials) of the following form:

$$s_i = \sum_i m_i = \sum_i c_i \tau^{x_{i,\tau}} \prod_{p \in P} a_p^{x_{i,p}}$$

The variables in a geometric program (i.e., $a_p, \tau \in A$) take on only positive values. The geometric program solver computes a binding that maps each geometric program variable to a numerical value $x \in \mathbb{R}_+$.

4.2 Generated Geometric Program

Jaunt generates the constraints C in the geometric program by traversing the ports P and connections Z in the analog configuration and generating appropriate constraints for each port or connection:

- **Connection Constraints**: These constraints ensure that the scaling factors for connected input and output ports are the same. Jaunt generates, for each connection $\langle op, ip\rangle \in Z$, the connection constraint $a_{ip}/a_{op} = 1$, which ensures that the scaling factors a_{ip} and a_{op} of the connected input and output ports ip and op are the same. The generated connection constraints are:

$$\bigcup_{\langle op, ip\rangle \in Z} \{a_{ip}/a_{op} = 1\}$$

- **Operating Range Constraints**: These constraints ensure that the signal at each port $p \in P$ falls within p's operating range. The relevant ranges are the operating range $\langle x, y\rangle = I(p)$ for the port p and the dynamic range $\langle x', y'\rangle = \mathbb{I}(F(p))$ for the dynamical system value that appears on the port p. The constraints $x \leq a_p \cdot x'$ and $a_p \cdot y' \leq y$, where a_p is the scaling factor for port p, ensure that the scaled signal dynamics fall within the operating range of the port. The generated operating range constraints are:

$$\bigcup_{p \in P} \mathbb{L}(x, a_p, x') \cup \mathbb{U}(y', a_p, y)$$

where $\langle x, y\rangle = I(p)$ and $\langle x', y'\rangle = \mathbb{I}(F(p))$. The lower bound function $\mathbb{L}(x, a_p, x')$ (Figure 6) generates constraints that ensure that $x \leq a_p \cdot x'$. The upper bound function $\mathbb{U}(y', a_p, y)$ (Figure 6) generates constraints that ensure that $a_p \cdot y' \leq y$. The interval function $\mathbb{I}(F(p))$ (Figure 8) computes the bounds $\langle x', y'\rangle$ for the dynamical system value that appears on port p.
Note that if $x > 0$ and $x' < 0$ or $y < 0$ and $y' > 0$, the range constraints cannot be satisfied. In this case $\mathbb{L}$ or $\mathbb{U}$ generates a *false* constraint and Jaunt does not find a physically realizable configuration.

- **Factor Constraints**: These constraints ensure that the scaled dynamics of each output port op is equivalent to the original dynamics scaled by a constant value. Jaunt derives a set of constraints C required to factor out a monomial expression m of scaling factors from the scaled dynamics of the output port $R(op)$. Jaunt also enforces the constraint $m/a_{op} = 1$, which ensures the scaling factor for the output port a_{op} equals m, the monomial of scaling factors factored out of the dynamics. The set of factor constraints is:

$$\bigcup_{op \in P \cap OP} \{m/a_{op} = 1\} \cup C$$

where $\langle m, C\rangle = \mathbb{F}(R(op))$ generates constraints that ensure the scaled signal at the output port op equals $m \cdot R(op)$.

- **Speed Constraints**: All digital output ports are subject to speed constraints that ensure that τ, the speed of the scaled signal, is greater than or equal to the minimum speed τ_{min}. The following generated speed constraint enforces this property:

$$\tau_{min}/\tau \leq 1$$

- **Sampling Constraints**: All digital output ports $do \in P$ and nonconstant digital input ports $di \in P$ are subject to sampling constraints that ensure the time between samples of the scaled signal is less than or equal to the maximum allowed time between samples λ_{max}. The following generated sampling constraints enforce this property:

$$\bigcup_{dp \in DP \cap P} \{\tau \cdot G(dp)/\lambda_{max} \leq 1\}$$

where $\tau \cdot G(dp)$ is the time between samples, in simulation time.

Objective Function: Jaunt uses the objective function s_{opt} to focus its selection of scaling factors. The objective function is an arbitrary posynomial, which enables Jaunt to support a range of scaling factor selection criteria. And because the geometric program is reduced to a convex optimization problem, Jaunt delivers optimal solutions for each objective function. The current Jaunt objective functions support optimization criteria related to the execution time and dynamic ranges of the signals in the simulation:

- **Minimum/Maximum Speed**: The objective function $s_{opt} = \tau$ minimizes the simulation speed. The objective function $s_{opt} = 1/\tau$ maximizes the simulation speed. These objective functions

are useful for minimizing the simulation time and enabling developers to understand the range of feasible simulation times.

- **Balanced Speed**: The objective function $s_{opt} = \tau + 1/\tau$ produces a balanced speed. This objective function is useful for pruning partial configurations early in the search process and is the default Jaunt objective function.
- **Minimum/Maximum Scaling Factor**: The objective function $s_{opt} = a_p$ minimizes the scaling factor for port p. The objective function $s_{opt} = 1/a_p$ maximizes this scaling factor. These objective functions are useful for exploring how small or large Jaunt can make the dynamic range of the scaled signal carried on p.
- **Balanced Scaling Factors:** The objective function $\sum_{do \in DO \cap P} 1/a_{do}$ maximizes the scaling factors (and therefore the dynamic ranges) of the observable outputs. This objective function is useful for maximizing the dynamic ranges, and therefore the binary precision, of the observable outputs.

This range of implemented objective functions highlights the flexibility of formulating the configuration scaling problem as a geometric program. This approach enables Jaunt to support more sophisticated optimization strategies, for example hierarchical strategies that first maximize and fix the speed within a desired range, then maximize the scaling factors applied to select ports while preserving the speed constraints.

4.2.1 Upper and Lower Bound Functions ($\mathbb{L}$ and $\mathbb{U}$)

Figure 6 presents the lower bound function $\mathbb{L}$, which accepts a value $x' \in \mathbb{R}$ that is scaled by some monomial m' and a minimum value $x \in \mathbb{R}$, and generates a geometric constraint that enforces $x \leq m' \cdot x'$:

- $x \cdot x' > 0$: if x and x' have the same sign, then the constraint $x \leq m' \cdot x'$ can be rewritten as $x/(m' \cdot x') \leq 1$, where x/x' is a positive nonzero number.
- $x \leq 0 \wedge x' \geq 0$: if x' is greater than x, and x and x' do not have the same sign, then the constraint $x \leq m' \cdot x'$ is true ($C = \emptyset$) for all m', since $m' \cdot x'$ is always positive and x is always negative.
- $x > 0 \wedge x' < 0$: if x' is less than x and x and x' do not have the same sign, then the constraint $x \leq m' \cdot x'$ is never true ($C = \{false\}$) for all m', since $m' \cdot x'$ is always negative and x is always positive.

The upper bound function $\mathbb{U}$ accepts a value $y' \in \mathbb{R}$ that is scaled by some monomial m' and a maximum value $y \in \mathbb{R}$ and generates a geometric constraint that enforces $m' \cdot y' \leq y$. The upper bound function $\mathbb{U}$ enforces $m' \cdot y' \leq y$ using the same technique as $\mathbb{L}$.

4.2.2 Factor Function ($\mathbb{F}$)

Figure 7 presents the factor function $\mathbb{F}$, which accepts a port expression e_p and generates a set of constraints C that make it possible to factor out the monomial m from e_p:

- $\mathbb{F}(x)$: The constant value x cannot be scaled, and therefore has the scaling factor 1.
- $\mathbb{F}(p)$: The port p is scaled by a_p, the scaling factor associated with p.

For the remaining rules, we introduce subexpressions e_p and e'_p that are scaled by monomials m and m' provided constraints C and C' hold. The scaled expressions are therefore $m \cdot (e_p)$ and $m' \cdot (e'_p)$.

$$
\begin{array}{rcll}
\mathbb{L}(x,\, m',\, x') & = & \{x/(m'x') \leq 1\} & \text{if } x \cdot x' > 0 \\
 & & \{false\} & \text{if } x > 0 \wedge x' < 0 \\
 & & \emptyset & \text{if } x \leq 0 \wedge x' \geq 0 \\
\mathbb{U}(y',\, m',\, y) & = & \{m'y'/y \leq 1\} & \text{if } y \cdot y' > 0 \\
 & & \{false\} & \text{if } y' > 0 \wedge y < 0 \\
 & & \emptyset & \text{if } y' \leq 0 \wedge y \geq 0
\end{array}
$$

Figure 6. $\mathbb{L}(x, m', x') = C$ and $\mathbb{U}(y', m', y) = C$ functions

$$
\begin{array}{rcl}
\mathbb{F}(x) & = & \langle \emptyset,\, 1 \rangle \\
\mathbb{F}(p) & = & \langle \emptyset,\, a_p \rangle \\
 & & \\
\mathbb{F}(e_p \times e'_p) & = & \langle C \cup C',\, m \times m' \rangle \\
\mathbb{F}(e_p \div e'_p) & = & \langle C \cup C',\, m/m' \rangle \\
\mathbb{F}(e_p + e'_p) & = & \langle C \cup C' \cup \{m/m' = 1\},\, m' \rangle \\
\mathbb{F}(e_p - e'_p) & = & \langle C \cup C' \cup \{m/m' = 1\},\, m' \rangle \\
\mathbb{F}(e_p^{e'_p}) & = & \langle C \cup C' \cup \{m' = 1\},\, m^x \rangle \\
 & & \text{if } \langle x,\, x \rangle = \mathbb{I}(\mathbb{E}(e'_p)) \\
 & & \langle C \cup C' \cup \{m = 1\},\, 1 \rangle \\
 & & \text{otherwise} \\
\mathbb{F}(\int e_p \partial p/\partial t\ p_0 = e'_p) & = & \langle C \cup C' \cup \{(m/\tau)/m' = 1\},\, m/\tau \rangle \\
\multicolumn{3}{c}{\text{where } \langle C,\, m \rangle = \mathbb{F}(e_p)\ ,\ \langle C',\, m' \rangle = \mathbb{F}(e'_p)}
\end{array}
$$

Figure 7. $\mathbb{F}(e_p) = \langle C, m \rangle$ function

$$
\begin{array}{rcl}
\mathbb{I}(x) & = & \langle x,\, x \rangle \\
\mathbb{I}(v) & = & J(v) \\
 & & \\
\mathbb{I}(e_v + e'_v) & = & \mathbb{I}(e_v) + \mathbb{I}(e'_v) \\
\mathbb{I}(e_v - e'_v) & = & \mathbb{I}(e_v) - \mathbb{I}(e'_v) \\
\mathbb{I}(e_v \times e'_v) & = & \mathbb{I}(e_v) \times \mathbb{I}(e'_v) \\
\mathbb{I}(e_v \div e'_v) & = & \mathbb{I}(e_v) \div \mathbb{I}(e'_v) \\
\mathbb{I}(e_v^{e'_v}) & = & \mathbb{I}(e_v)^{\mathbb{I}(e'_v)} \\
\mathbb{I}(\int e_v \partial v/\partial t\ v_0 = e'_v) & = & J(v)
\end{array}
$$

Figure 8. $\mathbb{I}(e_v) = l$ function

- $\mathbb{F}(e_p \cdot e'_p)$ *and* $F(e_p/e'_p)$: The monomial $m \cdot m'$ can be factored out of the scaled expression $m \cdot e_p \cdot m' \cdot e'_p$. The monomial m/m' can be factored out of the scaled expression $m \cdot e_p/(m' \cdot e'_p)$.
- $\mathbb{F}(e_p + e'_p)$ *and* $\mathbb{F}(e_p - e'_p)$: The monomial m can be factored out of a scaled expression $m \cdot e_p + m' \cdot e'_p$ or $m \cdot e_p - m' \cdot e'_p$ provided $m = m'$.
- $\mathbb{F}(e_p^{e'_p})$: If e'_p evaluates to a constant x, the scaling factor m^x can be factored out of the scaled expression $(m \cdot e_p)^{m' \cdot x}$ provided $m' = 1$. If e_p is not a constant value, the expression cannot be scaled and therefore has a scaling factor of 1. For the scaling factor to be 1, m must also be 1.
- $\mathbb{F}(\int e_p \partial p/\partial t\ p_0 = e'_p)$: The scaling factor m/τ can be factored out of the scaled expression $\int m \cdot e_p \partial p/\tau \partial t\ p_0 = m' \cdot e_{p'}$ provided the scaling factor of the initial condition m' equals the scaling factor of the differential equation dynamics m/τ, where the differential equation executes at speed τ.

4.2.3 Interval Function ($\mathbb{I}$)

Figure 8 presents the interval function $\mathbb{I}$, which accepts a dynamical system variable expression e_v and computes the interval l that contains the expression.

- $\mathbb{I}(x)$: The interval of a constant value is $\langle x, x \rangle$.
- $\mathbb{I}(v)$: The interval of a variable v is the interval defined in the dynamical system specification $J(v)$.

For the remaining rules, we introduce subexpressions e_v and e'_v with intervals $\langle x, y \rangle = \mathbb{I}(e_v)$ and $\langle x', y' \rangle = \mathbb{I}(e'_v)$.

- $\mathbb{I}(e_v \odot e'_v)$: We use interval arithmetic to define the interval of an expression $e_v \odot e'_v$, where $\odot \in \{+, -, \times, \div, /, \hat{}\}$ (addition, subtraction, multiplication, division, or exponentiation). For example, $\mathbb{I}(e_v - e'_v)$ is $\langle x - y', y - x' \rangle$, where $\langle x, y \rangle = \mathbb{I}(e_v)$ and $\langle x', y' \rangle = \mathbb{I}(e'_v)$, since the smallest possible value is the lower bound of e_v minus the upper bound of e'_v and the largest possible value is the upper bound of e_v minus the lower bound of e'_v. $\mathbb{I}(e_v + e'_v)$ is $\langle x + x', y + y' \rangle$ where $\langle x, y \rangle = \mathbb{I}(e_v)$ and $\langle x', y' \rangle = \mathbb{I}(e'_v)$.
- $\mathbb{I}(\int e_v \partial v / \partial t \wedge v_0 = e'_v)$: The interval of the integral of the expression e_v is defined as $l = J(v)$ in the dynamical system specification. Jaunt additionally verifies that $\langle x', y' \rangle \in l$.

4.2.4 Evaluation Function ($\mathbb{E}$)

Figure 9 presents the evaluation function $\mathbb{E}$, which accepts a port expression e_p and produces a dynamical system variable expression e_v by applying the port mapping F.

- $\mathbb{E}(x)$: The constant x evaluates to x.
- $\mathbb{E}(p)$: The port p evaluates to its mapped variable expression $e_v = F(p)$.
- $\mathbb{E}(e_p \odot e'_p)$: The evaluation function recursively evaluates e_p and e'_p to e_v and e'_v, then constructs the expression $e_v \odot e'_v$. Here $\odot \in \{+, -, \times, \div, \hat{}\}$.
- $\mathbb{E}(\int e_p \partial p / \partial t \; p_0 = e'_p)$: The evaluation function recursively evaluates the differential equation expression e_p and initial condition e'_p and retrieves $v = F(p)$, the variable mapped to p to construct the evaluated integral $\int e_v \partial v / \partial t \; v_0 = e'_v$. We note that $v = F(p)$ always exists because if $p \in P$ is a port whose behavior is defined by a differential equation, Arco always maps some variable $v \in V$ to p.

4.3 Implementation and Optimizations

Jaunt builds the geometric program using the geometric programming kit API (`gpkit`) [8]. The `gpkit` library translates the geometric program to a convex optimization program and then solves the convex optimization program using `cvxopt`. Jaunt uses `gpkit` to produce a set of scaling factors if a solution exists:

- **Found Scaling Transform**: If Jaunt successfully finds a transformation, Arco emits the analog configuration along with the scaling transformation found by Arco. By default Jaunt emits the minimum, maximum, and balanced speed scaling transforms.
- **No Scaling Transform**: If Jaunt fails to find a transformation, Arco discards the analog configuration.

Jaunt implements several optimizations to reduce the complexity of the generated geometric program and improve the efficacy of the Arco compiler's search algorithm.

Metavariables: Jaunt aggregates equivalent scaling factors into a single aggregate scaling factor. This optimization reduces the number of variables in the geometric program and improves the performance of the geometric program solver.

Search Path Pruning: Arco generates configurations by exploring a search space of partially completed configurations. Arco can reduce the size of the explored search space by querying Jaunt to determine if the partial configuration associated with the current search path is physically realizable. If not, all configurations derived from the partial configuration are also physically unrealizable and Arco prunes the search path from the search. This optimization eliminates unproductive regions of the search space and helps Arco find successful configurations more quickly.

$$
\begin{array}{rcl}
\mathbb{E}(x) & = & x \\
\mathbb{E}(p) & = & F(p) \\
\\
\mathbb{E}(e_p + e'_p) & = & \mathbb{E}(e_p) + \mathbb{E}(e'_p) \\
\mathbb{E}(e_p - e'_p) & = & \mathbb{E}(e_p) - \mathbb{E}(e'_p) \\
\mathbb{E}(e_p \times e'_p) & = & \mathbb{E}(e_p) \times \mathbb{E}(e'_p) \\
\mathbb{E}(e_p \div e'_p) & = & \mathbb{E}(e_p) \div \mathbb{E}(e'_p) \\
\mathbb{E}(e_p^{e'_p}) & = & \mathbb{E}(e_p)^{\mathbb{E}(e'_p)} \\
\mathbb{E}(\int e_p \partial p / \partial t \; p_0 = e'_p) & = & \int \mathbb{E}(e_p) \partial v / \partial t \; v_0 = \mathbb{E}(e'_p) \\
 & & \text{where } v = F(p)
\end{array}
$$

Figure 9. $\mathbb{E}(e_p) = e_v$ function

4.4 Design Decisions

Geometric Programming Formulation: We chose to formulate the constraint problem as a geometric programming problem because it can be translated into a convex optimization problem and supports constraints over monomials, which cover all of the Jaunt scaling expressions. This formulation does exclude three kinds of scaling patterns. Including these scaling transforms would produce a nonconvex optimization problem and would not, in our experience, significantly improve Jaunt's ability to produce successful scaling transforms.

- **Exponentiation**: We exclude scaling transforms on exponentials (e_p^x) that simultaneously scale the base and the exponent of the expression (monomials do not include variables in exponents). Supporting these transforms would complicate the constraint problem and would not confer significant benefits because, in practice, the base and exponent rarely, if ever, need to both be scaled.
- **Negative Scaling Factors**: Negative scaling factors are useful only for 1) mapping negative values onto components with positive operating ranges or 2) mapping positive values onto components with negative operating ranges. An earlier version of Jaunt supported negative scaling factors. We eliminated this support because 1) it significantly simplified the formulation and solution of the constraint problem and 2) we found negative scaling factors to be of little to no use in practice.
- **Scaling by Zero**: We exclude transforms that scale signals by zero, since any nonzero signal scaled by zero cannot be recovered.

Scaling Analog Configurations: When designing Jaunt, we chose to compute the scaling transform starting from an analog configuration instead of directly searching for a physically realizable configuration. The analog configuration synthesis problem is a discrete, combinatorial problem that Arco solves by searching combinations of configured blocks and connections to include in the configuration. The problem of computing a scaling transform is a numerical optimization problem over a continuous space. Solving these two problems separately enabled us to use different algorithms as appropriate for these two different kinds of problems.

benchmark	parameters	functions	diffeqs	sample (λ_{max})	time	description
smol	4	2	1	0.5 su	10 su	simplified Michaelis-Menten reaction in terms of mols.
sconc	4	2	1	1 su	25 su	simplified Michaelis-Menten reaction in terms of concentrations.
mmrxn	5	0	4	80 su	6000 su	Michaelis-Menten equation reaction[25]
gtoggle	9	3	2	0.1 su	10 su	genetic toggle switch in E.coli [15]
repri	7	0	9	20 su	1000 su	synthetic oscillatory network of transcriptional regulators [12]
bont	8	1	5	5 su	250 su	paralysis of skeletal muscles from botulinum neurotoxin A [20]
epor	3	1	6	20 su	100 su	information processing at the erythropoletin receptor [4]

Table 2. Benchmark Characteristics. Minimum speed τ_{min} for all benchmarks is 0.0001x.

block	description	circuit	relation	input operating ranges
idac	current input	DAC	$Z_I = X_D$	I∈ [0,10] uA,clk=0.1 hu
vdac	voltage input	DAC	$Z_V = X_D$	V∈ [0,3300] mV,clk=0.1 hu
vadc	voltage output	ADC	$Z_D = X_V$	V∈ [0,3300] mV,clk=1 hu
iadd	current adder	wires	$O_I = A_I + B_I - C_I - C_I$	I ∈ [0,10] μA
vadd.deriv	voltage adder	$\sum$ amp+capacitor	$\partial Z_V/\partial t = A_V + B_V - C_V - D_I \cdot Z_V$, $Z_0 = D0_V$	I ∈ [0,10] μA, V ∈ [0,3300] mV
vadd.fxn	voltage adder	$\sum$ amp	$Z_V = [A_V + B_V - C_V] \cdot 0.25$	V ∈ [0,3300]
vgain	voltage to current	opamp	$Z_I = X_V K_I$	V ∈ [0,500] mV, K_I ∈ [0,10] μA
itovc	current to voltage	opamp	$Z_V = X_I K_V$	I ∈ [0,10] μA, K_V ∈ [0,100] mV
itov	current to voltage	opamp	$Z_V = X_I K_V$	I ∈ [0,0.1] μA, K_V ∈ [0,330] mV
vtoi	voltage to current	opamp	$Z_I = X_V / K_V$	X_V ∈ [0,3300] mV, K_V ∈ [330,3300] mV
ihill	hill function	logic circuit	$S_I = X_I[S_I^{n_V}/[S_I{}^{n_V} + K_I^{n_V}]]$	K_I ∈ [0.5,10] μA,I ∈ [0,10] μA, V ∈ [1,3] mV
switch	genetic switch	logic circuit	$O_I = M_I/[S_I/K_I + 1]^{n_V/500}$	I ∈ [0,10] mA,n_V ∈ [500,3300] mV
igenebind	gene binding	logic circuit	$P_I = X_I \cdot 1/[K_I \cdot T_I + 1]$	I ∈ [0,10] μA
mm	michaelis menten	logic circuit	$X_V = XT_V - Z_V$	XT_V, YT_V ∈ [0.0001,1000] mV
			$Y_V = YT_V - Z_V$	A_I ∈ [0.0001,0.01] μA, B_I ∈ [0.0001,1] μA
			$\partial Z_V/\partial t = A_I \cdot X_V \cdot Y_V - B_I \cdot Z_V$, $Z_0 = Z0_V$	$Z0_V$ ∈ [0,600] mV

Table 3. Components and Operational Ranges in Analog Device [9, 10, 28, 32, 36]. 1 hardware time unit (hu) = 2 ms wall clock time.

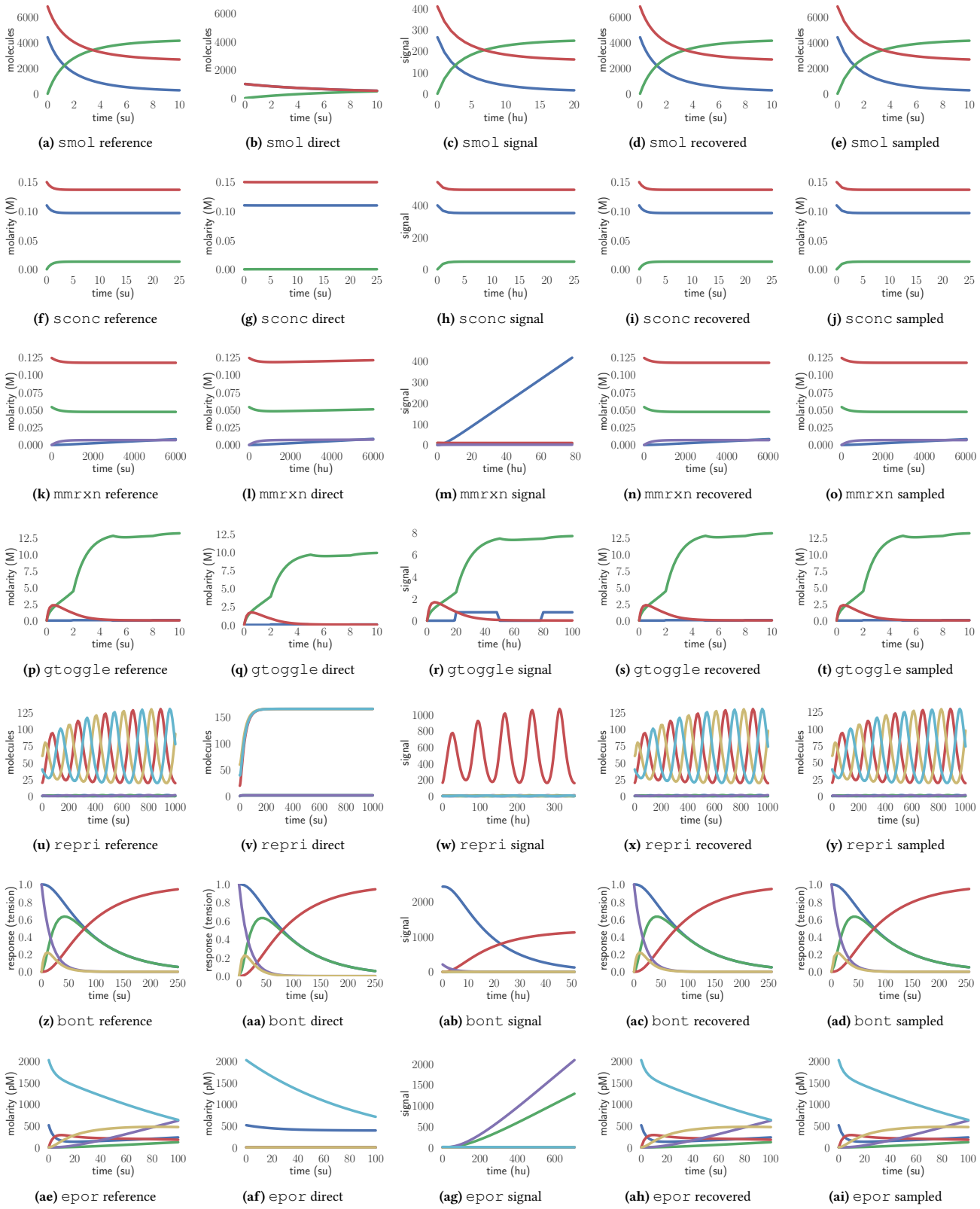
5 Experimental Results

We present experimental results for Jaunt on a set of benchmark biological simulations. Each benchmark is a dynamical system selected from a set of published artifacts included with a peer-reviewed biology paper. We derived the ranges of inputs and state variables from conservation properties present in systems of chemical reactions. Table 2 presents characteristics of these benchmarks:

- **smmrxn**: The simplified Michaelis-Menten benchmark models a reaction $E + S \leftrightharpoons ES$ [25]. The dynamical system has 4 parameters, 2 functions, and 1 differential equation. The system takes no inputs and produces the quantity of E,S, and ES as outputs. We evaluate two versions of the simplified Michaelis-Menten reaction, `smol`, which represents the quantities as molecules, and `sconc`, which represents the quantities as concentrations. The `smol` and `sconc` simulations execute to steady state (in 10 simulation units and 25 simulation units respectively).
- **mmrxn**: The Michaelis-Menten benchmark models the reaction $E + S \leftrightharpoons ES \rightarrow P$ [25]. The dynamical system has 5 parameters and 4 differential equations. The system has no inputs and produces the concentrations of E, S, ES, and P as outputs. The `mmrxn` simulation executes until an significant amount of P accumulates (6000 simulation units).
- **gtoggle**: The genetic toggle switch is a gene regulatory network found in E. coli with two repressible promoters [15]. The dynamical system has 9 parameters, 3 functions, and 2 differential equations. The system takes IPTG as an input and emits the activity levels of U and V as outputs. We evaluate `gtoggle` by introducing IPTG at simulation unit 2 and withdrawing it at simulation unit 5. The `gtoggle` simulation executes to steady state (10 su).
- **repri**: The LacI-tetR-Cl transcriptional reprissilator is an oscillating network composed of genes that are not involved in maintaining a biological clock [12]. The dynamical system has 7 parameters and 9 differential equations. The `repri` simulation executes to oscillating steady state (1000 simulation units).
- **bont**: The bont benchmark models the onset of skeletal muscle paralysis brought on by the botulinum neurotoxin [20]. The dynamical system has 8 parameters, 1 function, and 5 differential equations. The `bont` simulation executes for 250 simulation units, the length of time reported in the relevant paper [20].
- **epor**: The `epor` benchmark models erythropoietin receptor activation (EpoR) in response to a fixed ligand input [4]. The dynamical system has 3 parameters, 1 function, and 6 differential equations. The `epor` simulation executes for 100 simulation units, the length of time reported in the relevant paper [4].

5.1 Analog Device

The benchmarks execute on a hardware model composed of components commonly used in analog devices that target biological computations [9, 10, 28, 32, 36]. Table 3 presents the components and operational ranges of the simulated analog device. The simulations saturate signals that exceed the maximum operating constraint and clamp signals that fall below the minimum operating constraint.



(a) `smol` reference (b) `smol` direct (c) `smol` signal (d) `smol` recovered (e) `smol` sampled

(f) `sconc` reference (g) `sconc` direct (h) `sconc` signal (i) `sconc` recovered (j) `sconc` sampled

(k) `mmrxn` reference (l) `mmrxn` direct (m) `mmrxn` signal (n) `mmrxn` recovered (o) `mmrxn` sampled

(p) `gtoggle` reference (q) `gtoggle` direct (r) `gtoggle` signal (s) `gtoggle` recovered (t) `gtoggle` sampled

(u) `repri` reference (v) `repri` direct (w) `repri` signal (x) `repri` recovered (y) `repri` sampled

(z) `bont` reference (aa) `bont` direct (ab) `bont` signal (ac) `bont` recovered (ad) `bont` sampled

(ae) `epor` reference (af) `epor` direct (ag) `epor` signal (ah) `epor` recovered (ai) `epor` sampled

Figure 10. Reaction Dynamics

5.2 Evaluation Methodology

We run Jaunt on each of the benchmark applications, selecting the first Arco configuration for which Jaunt can find a physically realizable and recoverable simulation. We configure Jaunt to use the objective function $s_{opt} = 1/\tau$ that produces the fastest scaled simulation. To ensure the adequate observability of the transient dynamics, we enforced the sampling constraints from Table 2. We execute three simulations for each benchmark:

- **Reference Execution**: Executes the dynamical system simulation using a standard differential equation simulator.
- **Direct Execution**: Executes the unscaled Arco configuration.
- **Jaunt Execution:** Executes the Arco configuration with the Jaunt scaling transform applied.

We implement each execution using Simulink [21]. Following standard practice in the field, we consider a simulation accurate if it produces the same result as the reference simulation [32, 36].

5.3 Results

Figure 10 presents graphs that summarize the execution results. Each graph plots executed relevant values of state variables from the benchmark dynamical system measured in millivolts and milliamps as a function of time. We plot selected dynamical system state variables of interest to a biologist, using guidelines from the `BioModels` database when available [1]. We present five graphs for each benchmark:

- **Reference:** Plots values from the standard differential equation solver. Each graph plots the values of the selected state variables measured in the units of the specified dynamical system as a function of time measured in the simulation time units (su).
- **Direct:** Plots values from the direct execution. Each graph plots, as a function of time, values of the selected state variables measured at the analog ports that carry these values. The values are measured in the units of the analog hardware signals (mV or μA). Time is measured in hardware time units. Because Arco directly maps the simulation onto the analog hardware, the simulation units match the hardware units.
  Our instrumentation of these executions indicates that all executions except `bont` violate the device operating ranges. These violations cause the direct executions to diverge (in most cases significantly) from the reference executions.
- **Signal:** Plots values from the scaled Jaunt execution measured at the analog ports that carry the values. The graphs plot the values measured in hardware signal units as a function of hardware time units. Because of the applied scaling transformation, the time and state variable values are scaled with respect to the reference simulation.
  Our instrumentation of these simulations indicates that they all respect the device operating ranges.
- **Recovered:** Plots values from the scaled Jaunt execution measured at the analog ports that carry the values after recovering the simulation by inverting the scaling transformation. Each graph plots the values of the selected state variables measured in the units of the specified dynamical system as a function of time measured in the simulation time units. The reference and recovered plots correctly match for all benchmarks.
- **Sampled:** Plots values from the scaled Jaunt execution sampled at the digital ADC ports that carry the values out of the device after recovering the simulation by inverting the scaling transformation. Each graph plots the values of the selected state variables measured in the units of the specified dynamical system as a function of time measured in the simulation time units.
  The reference and recovered plots match for all benchmarks, indicating that the scaled executions are executed slowly enough to adequately observe the dynamics in the sampled signal.

Direct Executions: All of the direct executions except `bont` deviate from their reference execution. We attribute these deviations to the fact that the direct mapping drives the device outside its operating range. Some of these operating range violations occur at observable variables mapped to external ports and are straightforward to detect. In other cases, however, the operating range violations occur at saturated internal ports in the analog device. In this case, the observable variables can be well within their operating ranges while the execution itself is operating well outside the operating range of the device. In the absence of a specialized debugging interface, which analog devices typically do not provide, it can be difficult to pinpoint or even detect the presence of these operating range violations. By automating the process of scaling the configuration to fit within the operating range, Jaunt eliminates this potential source of error. We next discuss specific operating range violations that occur in our benchmark direct executions:

- **smol**: The starting concentrations of E (6800) and S (4000) exceed the maximum acceptable voltage of 1000mV at the `XT` and `YT` ports. This causes the dynamics of E (red line) and S (blue line) to look the same.
- **sconc**: The parameter 1 exceeds the maximum acceptable current of 0.01mA at the $K1$ port of the `mm` block. This causes ES (green) to be break down much faster than it is formed, eliminating any changes in concentration in E (red line), ES (green line) and P (blue line).
- **mmrxn**: The parameter 317.05 falls below the minimum acceptable voltage of 330 mV at the `K` port of two different `vtoi` blocks. In the resulting execution, E (red line), S (green line), and ES (purple line) begin to increase around 1000 hardware units instead of remaining constant.
- **gtoggle**: The parameters 15.6 and 13.32 exceed the maximum current of 10mA at the `X` port of two different `ihill` blocks. This triggers the amount of U (green line) to saturate prematurely at 10 molar in response to the introduction of IPTG (blue line).
- **bont**: The directly mapped configuration respects the operating range of the device.
- **repri**: The LacLp, clp and TetRp signals ([0, 140]) exceed the maximum current of 10 μA at the `S` port and the parameter 20 exceeds the maximum current of 10 μA of the `K` port in three distinct `ihill` blocks. These operating range violations eliminate the oscillations from the dynamics.
- **epor**: The EpoR signal ([0,516]) exceeds the maximum voltage of 100 mV of port `X` in two distinct `vtoic` blocks. The Epo signal ([0,2031]) exceeds the maximum current of 10 μA of the `Z` port of the `vgain` block. These violations alter the transient dynamics of the system.

Scaled Executions: As highlighted in the signal plots, the applied scaling transforms often apply different scaling factors to different ports and scale time to maximize the speed of the simulation while still ensuring the scaled configuration is physically realizable, recoverable, and observable. The `smol` and `sconc` benchmarks scale the

bmark	min τ	max τ	slow (hu)	fast (hu)	slow (s)	fast (s)
smol	0.010x	0.500x	1000 hu	20 hu	2s	0.004s
sconc	0.0001x	1.000x	25000 hu	25 hu	50s	0.05s
mmrxn	0.0001x	77.482x	$6 \cdot 10^6$ hu	77 hu	3h20m	0.155s
gtoggle	0.099x	0.100x	100 hu	100 hu	0.2s	0.2s
repri	0.0001x	2.839x	10^6 hu	352 hu	32m40s	0.704s
bont	0.0001x	5.000x	$25 \cdot 10^4$ hu	50 hu	8m20s	0.1s
epor	0.0001x	0.142x	10^6 hu	704 hu	3m20s	1.408s

Table 4. Jaunt Time Scaling Factors.

output signals uniformly. The `mmrxn`, `gtoggle`, `repri`, `bont`, and `epor` benchmarks scale each output signal by a different scaling factor.

- **smol**: All the output signals in the `smol` benchmark (Figure 10c) are scaled uniformly by 0.060x. Time is scaled by 0.50x.
- **sconc**: All the output signals in the `sconc` benchmark (Figure 10h) are scaled by 3643.68x. The simulation executes in real time ($\tau = 1$).
- **mmrxn**: The output signals in the `mmrxn` benchmark (Figure 10m) are all scaled by different amounts: E (red line) by 80.64x, S (green line) by 164.88x, ES by 167.57x (purple line), and P by 50291.40x (blue line). Time is scaled by 77.48x. We note the state variable P, which is scaled the most, has the smallest dynamic range and is most visible in the plot.
- **gtoggle**: All the input and output signals in the `gtoggle` benchmark (Figure 10r) are scaled by different amounts: the input $IPTG$ by 12.74 (blue line), U by 0.58x (green line), and V by 0.72x (red line). The simulation time is scaled by 0.10x. We note the $IPTG$ signal has the smallest dynamic range and the largest scaling factor.
- **repri**: All the output signals of the `repri` benchmark (Figure 10w) are scaled differently: most notably the $LacLp$,clp, and $TetRp$ variables (red, cyan, and yellow lines) are scaled by 8.25x, 0.065x and 0.064x respectively. The clm, $LacLm$ and $TetRm$ variables (blue, purple, and green) are scaled by 2.98x, 7.05x and 6.51x respectively. The simulation time is scaled by 2.84x.
- **bont**: All the output signals of the `bont` benchmark (Figure 10ab) are scaled differently: most notably $tension$ and $lytic$ (green and yellow) are scaled by 2428.06x and 1183.00x respectively and dominate the figure. The bnd, $free$ and $transloc$ variables (blue, red, and purple respectively) are scaled by 8.47x, 212.12x and 9.49x. The simulation time is scaled by 5.00x.
- **epor**: All the output signals of the `epor` benchmark (Figure 10ag) are scaled differently: most notably, the $dEpoe$ and $dEpoi$ variables (purple and green) are scaled by 3.36x and 10.65x respectively. The Epo and $EpoR$ variables (cyan and blue) are both scaled by 0.0049x and the $EpoEpoRi$ and $EpoEpor$ variables (yellow and red) are scaled by 0.011x and 0.023x. The simulation time is scaled by 0.14x.

Simulation Speed: Table 4 presents the simulation speeds for the optimal fastest and slowest simulations for each benchmark, and the corresponding simulation time in hardware time and wall clock time. The `smol`, `sconc`, `gtoggle`, and `bont` benchmarks attain the maximum possible simulation speed possible subject to each benchmark's sampling constraints. The `mmrxn`, `repri`, and `bont` benchmarks support executing the simulation 3.84x-77.48x faster than a real time execution ($\tau = 1$). The fastest physically feasible simulations complete in 20 hardware units (0.004s) to 704 hardware units (1.408s). The slowest physically feasible simulations complete in $6 \cdot 10^6$ hardware units (3 hours, 20 minutes) to 100 hu (0.2 s).

variable	min scale	max scale	balopt
mmrxn			
τ	0.020	77.48	69.88
E	20.82	80.64	80.64
S	20.82	185.19	185.19
P	$< 10^{-6}$	61111.11	61107.74
ES	0.11	185.19	185.19
gtoggle			
τ	0.000033	0.10	0.10
U	0.50	0.64	0.64
V	0.50	0.75	0.75
IPTG	0.033	16.66	16.66
repri			
τ	0.0033	2.84	0.16
LacLp	8.25	23.57	23.57
TetRp	0.025	0.071	0.071
clp	0.025	0.071	0.071
LacLm	0.0082	110.00	109.99
clm	0.0000010	3.33	3.33
TetRm	0.000025	110.00	109.99
bont			
τ	0.00083	5.00	2.09
bnd	$< 10^{-6}$	10	10.00
tension	$< 10^{-6}$	3299.99	3299.26
lytic	$< 10^{-6}$	3299.99	3299.27
transloc	$< 10^{-6}$	10	10.00
free	$< 10^{-6}$	329.99	329.99
epor			
τ	0.00033	0.14	0.14
EpoR	0.00010	0.0049	0.0049
dEpoi	$< 10^{-6}$	13.20	13.19
EpoEpoR	$< 10^{-6}$	0.030	0.030
EporEporI	$< 10^{-6}$	0.019	0.019
Epo	0.00010	0.0049	0.0049
dEpoe	$< 10^{-6}$	4.13	4.12

Table 5. Jaunt Output Variable Scaling Factors.

5.4 Dynamic Range Analysis

We next present results that use the port scaling factor objective functions (Section 4.2) to explore Jaunt's ability to minimize and maximize the dynamic ranges of output variables. For each benchmark, we execute the geometric programs used to generate the scaling transforms in Section 5.3 with the port scaling objective functions and with a minimum speed constraint that ensures the simulation finishes in ten minutes. Table 5 presents results from solving the geometric program with the following optimization functions:

- **Minimize/Maximize Individual Output Variables**: We solve the geometric program twice for each output variable, once to find the minimum scaling factor (column `min scale`) for that variable and once to find the maximum scaling factor (column `max scale`). The differences between the minimum and maximum possible scaling factors for each output variable highlight the substantial ranges that Jaunt makes available for the scaling factors for each individual output variable.

bmark	exec	total (s)	solver (s)	probs	succ	fail	# vars
smol	std	18.18	0.573	14	11	3	4
	nmt	25.51	7.32	16	10	6	25
sconc	std	17.14	0.833	17	14	3	4
	nmt	21.93	8.665	12	10	2	25
mmrxn	std	150.97	6.26	50	50	0	20
	nmt	152.33	23.38	50	50	0	76
gtoggle	std	199.00	4.74	56	56	0	15
	nmt	227.66	45.84	56	54	2	63
repri	std	384.27	12.06	95	87	8	34
	nmt	458.61	95.89	106	95	11	129
bont	std	112.20	4.13	56	49	7	18
	nmt	126.41	21.65	58	49	9	81
epor	std	295.90	9.42	96	81	15	25
	nmt	334.48	71.82	81	76	5	122

Table 6. Jaunt Execution times.

- **Maximize All Output Variables**: We solve the geometric program once for each benchmark, using the balanced scaling factor objective function to (conceptually) maximize all of the scaling factors together (column `balopt`). The results show that, for all benchmarks, it is possible to obtain scaled configurations in which each scaling factor attains close to its maximum possible value. For the `mmrxn`, `gentoggle`, and `epor` benchmarks, Jaunt is also able to obtain close to (within 10% of) the maximum simulation speed. The `bont` and `repri` benchmarks execute more slowly — `bont` executes 2.09x faster than real time (maximum speed is 5x faster) and `repri` benchmark executes 6.37x slower than real time (maximum speed is 2.84x faster).

5.5 Jaunt Optimizations

We analyze the performance of the Jaunt compiler and the effect of different optimizations. We execute two trials for each benchmark:

- **std**: Jaunt prunes partial configurations during the Arco search process and merges equivalent scaling factors into metavariables.
- **nmt**: Jaunt prunes partial configurations during the Arco search process but does not perform the optimization that merges equivalent scaling factors into metavariables.

Table 6 presents the time required for Arco and Jaunt, working together, to generate a physically realizable, recoverable scaled configuration. It also presents the total amount of time taken by the combined Arco/Jaunt compiler in seconds (`total`) and the time spent in solving geometric programming problems in seconds (`solver`).

Jaunt generates and attempts to solve one geometric programming problem for each partial or complete Arco configuration. Table 6 presents the total number of geometric programming problems that Jaunt generates and attempts to solve (`probs`), the number of geometric programming problems that produce physically realizable scaling transforms (`succ`), and the number of geometric programming problems that do not have a solution (`fail`). The `# vars` column reports the number of variables in the geometric programming problem that produces the final scaled configuration.
Metavariable Support: The `nmt` (no metavariable) trials spend substantially more time in the geometric solver than the `std` trials (7.32 to 95.89 seconds vs. 0.573 to 12.06 seconds). This difference reflects the impact of the metavariable optimization, which substantially reduces the number of variables in the generated geometric problems (25 to 129 variables vs. 4 to 34 variables). The reduced geometric program solution times translate into reduced overall compilation times (21.93 to 458.61 seconds vs. 17.14 to 384.27 seconds), highlighting the effectiveness of the metavariable optimization.
Pruning Search Paths: Both the `std` and `nmt` executions perform the configuration search with pruning enabled. With pruning disabled, Jaunt is unable to find physically realizable configurations for `epor`, `bont`, and `repri`.

6 Related Work

Historically, simple analog circuits with manually developed mappings have been used to perform numerical computation and perform dynamical system simulations [11, 23, 27, 33]. Jaunt automates the process of deriving a mapping that respects the physical operating constraints of the underlying analog device.

Analog computing has recently reemerged as a computing substrate within the hardware research community. New modern analog devices often contain domain specific computational blocks designed closely model physical phenomena from the domain of interest [5, 9, 10, 26, 28, 29, 32, 36]. By manipulating the simulation speed, Jaunt derives mappings that respect the operating constraints of these modern analog hardware devices.

Other work in synthesizing analog circuits has focused on transistor level techniques to help hardware designers create specialized logic circuits [3, 16, 22]. Researchers have also leveraged analog accelerators such as neural network accelerators to approximate digital subcomputations written in imperative languages [14, 31]. In contrast, the Jaunt solver accurately transforms the dynamical system specification to ensure that the simulation respects the physical limitations of the computational blocks on the device.

The Jaunt solver uses interval analysis to reason about the range of subexpressions comprised of dynamical system variables and subexpressions comprised of hardware port properties. Interval analysis has a long history in fields such as electrical engineering, control theory and robotics [17, 18]. The Jaunt solver automates interval propagation techniques in the constraint generation process and when inferring operating ranges.

7 Conclusion

Programmable analog devices are a powerful new computing substrate for neuromorphic and cytomorphic models [5, 9, 10, 26, 28, 29, 31, 32, 36]. Jaunt automates the complex process of deriving a mapping of the dynamical system onto the analog hardware that respects the device's physical operating constraints, a key step in enabling the use of this promising new class of devices.

8 Acknowledgements

This research was supported by AFRL/DARPA contract FA8650-15-C-7564.

References

[1] Biomodel artifact database. https://www.ebi.ac.uk/biomodels-main.

[2] Sara Achour, Rahul Sarpeshkar, and Martin C Rinard. Configuration synthesis for programmable analog devices with arco. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 177–193. ACM, 2016.

[3] Kurt Antreich, J Eckmueller, Helmut Graeb, Michael Pronath, E Schenkel, R Schwencker, and S Zizala. Wicked: Analog circuit synthesis incorporating mismatch. In *Custom Integrated Circuits Conference, 2000. CICC. Proceedings of the IEEE 2000*, pages 511–514. IEEE, 2000.

[4] Verena Becker, Marcel Schilling, Julie Bachmann, Ute Baumann, Andreas Raue, Thomas Maiwald, Jens Timmer, and Ursula Klingmüller. Covering a broad dynamic range: information processing at the erythropoietin receptor. *Science*, 328(5984):1404–1408, 2010.

[5] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Shobhit Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul Merolla, Kwabena Boahen, et al. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[6] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. A tutorial on geometric programming. *Optimization and engineering*, 8(1):67, 2007.

[7] Donald G Buerk. Can we model nitric oxide biotransport? a survey of mathematical models for a simple diatomic molecule with surprisingly complex biological activities. *Annual review of biomedical engineering*, 3(1):109–143, 2001.

[8] Edward Burnell and Warren Hoburg. Gpkit software for geometric programming. https://github.com/convexengineering/gpkit, 2017. Version 0.6.0.

[9] G.E.R. Cowan, R.C. Melville, and Y. Tsividis. A VLSI analog computer/digital computer accelerator. *Solid-State Circuits, IEEE Journal of*, 41(1):42–53, Jan 2006.

[10] Ramiz Daniel, Sung Sik Woo, Lorenzo Turicchia, and Rahul Sarpeshkar. Analog transistor models of bacterial genetic circuits. In *Biomedical Circuits and Systems Conference (BioCAS), 2011 IEEE*, pages 333–336. IEEE, 2011.

[11] J.L. Douce and H. Wilson. The automatic synthesis of control systems with constraints. *Mathematics and Computers in Simulation*, 7(1):18 – 22, 1965.

[12] Michael B Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[13] Jason Ernst and Manolis Kellis. ChromHMM: automating chromatin-state discovery and characterization. *Nature Methods*, 9(3):215–6, mar 2012.

[14] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460. IEEE Computer Society, 2012.

[15] Timothy S Gardner, Charles R Cantor, and James J Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342, 2000.

[16] Ramesh Harjani, L Richard Carley, et al. Oasys: A framework for analog circuit synthesis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 8(12):1247–1266, 1989.

[17] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer London, 2012.

[18] L.V. Kolev. *Interval Methods for Circuit Analysis*. Advanced series on circuits and systems. World Scientific, 1993.

[19] E Lalonde, A S Ishkanian, J Sykes, M Fraser, H Ross-Adams, N Erho, M J Dunning, S Halim, A D Lamb, N C Moon, G Zafarana, A Y Warren, X Meng, J Thoms, M R Grzadkowski, A Berlin, C L Have, V R Ramnarine, C Q Yao, C A Malloff, L L Lam, H Xie, N J Harding, D Y Mak, K C Chu, L C Chong, D H Sendorek, C P'ng, C C Collins, J A Squire, I Jurisica, C Cooper, R Eeles, M Pintilie, A Dal Pra, E Davicioni, W L Lam, M Milosevic, D E Neal, T van der Kwast, P C Boutros, and R G Bristow. Tumour genomic and microenvironmental heterogeneity for integrated prediction of 5-year biochemical recurrence of prostate cancer: a retrospective cohort study. *Lancet Oncol*, 15(13):1521–1532, 2014.

[20] Frank J Lebeda, Michael Adler, Keith Erickson, and Yaroslav Chushak. Onset dynamics of type a botulinum neurotoxin-induced paralysis. *Journal of pharmacokinetics and pharmacodynamics*, 35(3):251, 2008.

[21] MATLAB. *R2015b*. The MathWorks Inc., Natick, Massachusetts, 2015.

[22] E.S. Ochotta, R.A. Rutenbar, and L.R. Carley. Synthesis of high-performance analog circuits in ASTRX/OBLX. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 15(3):273–294, Mar 1996.

[23] Yakup Paker and Stephen H. Unger. {ADAC} — a programmed direct analog computer. *Mathematics and Computers in Simulation*, 9(1):16 – 23, 1967.

[24] Alberto Paoluzzi, Valerio Pascucci, Michele Vicentino, Claudio Baldazzi, and Simone Portuesi. *Geometric Programming*, pages 51–93. John Wiley & Sons, Ltd, 2005.

[25] Denise R. Ferrier PhD. *Biochemistry (Lippincott Illustrated Reviews Series)*. LWW, 2013.

[26] Sylvain Saighi, Yannick Bornat, Jean Tomas, Gwendal Le Masson, and Sylvie Renaud. A library of analog operators based on the Hodgkin-Huxley formalism for the design of tunable, real-time, silicon neurons. *Biomedical Circuits and Systems, IEEE Transactions on*, 5(1):3–19, 2011.

[27] Sams. Arrangement and scaling of equations. *Mathematics and Computers in Simulation*, 6(3):179 – 182, 1964.

[28] Rahul Sarpeshkar. *Ultra Low Power Bioelectronics: Fundamentals, Biomedical Applications, and Bio-Inspired Systems*. Cambridge University Press, 2010.

[29] Christian Schneider and Howard Card. Analog CMOS synaptic learning circuits adapted from invertebrate biology. *Circuits and Systems, IEEE Transactions on*, 38(12):1430–1438, 1991.

[30] Richard I Sherwood, Tatsunori Hashimoto, Charles W O'Donnell, Sophia Lewis, Amira a Barkal, John Peter van Hoff, Vivek Karun, Tommi Jaakkola, and David K Gifford. Discovery of directional and nondirectional pioneer transcription factors by modeling DNase profile magnitude and shape. *Nature Biotechnology*, 32(2):171–8, mar 2014.

[31] Renée St Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger. General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Computer Architecture News*, 42(3):505–516, 2014.

[32] Jonathan J. Y. Teo, Sung Sik Woo, and Rahul Sarpeshkar. Synthetic biology: A unifying view and review using analog circuits. *IEEE Trans. Biomed. Circuits and Systems*, 9(4):453–474, 2015.

[33] Rajko Tomovic. Proceedings of the international association for analog computation method of iteration and analog computation. *Mathematics and Computers in Simulation*, 1(2):60 – 63, 1958.

[34] Herbert Weiner. The illusion of simplicity: the medical model revisited. *The American journal of psychiatry*, 1978.

[35] Sung Sik Woo et al. *Fast simulation of stochastic biochemical reaction networks on cytomorphic chips*. PhD thesis, Massachusetts Institute of Technology, 2016.

[36] Sung Sik Woo, Jaewook Kim, and Rahul Sarpeshkar. A cytomorphic chip for quantitative modeling of fundamental bio-molecular circuits. *IEEE Trans. Biomed. Circuits and Systems*, 9(4):527–542, 2015.