Evaluation of an Analog Accelerator for Linear Algebra

Yipeng Huang*, Ning Guo†, Mingoo Seok†, Yannis Tsividis†, and Simha Sethumadhavan*
*Department of Computer Science, †Department of Electrical Engineering
Columbia University
New York, NY, USA
{*yipeng@cs., ng2364@, mgseok@ee., tsividis@ee., simha@cs.*}*columbia.edu*

***Abstract*—Due to the end of supply voltage scaling and the increasing percentage of dark silicon in modern integrated circuits, researchers are looking for new scalable ways to get useful computation from existing silicon technology. In this paper we present a reconfigurable analog accelerator for solving systems of linear equations. Commonly perceived downsides of analog computing, such as low precision and accuracy, limited problem sizes, and difficulty in programming are all compensated for using methods we discuss. Based on a prototyped analog accelerator chip we compare the performance and energy consumption of the analog solver against an efficient digital algorithm running on a CPU, and find that the analog accelerator approach may be an order of magnitude faster and provide one third energy savings, depending on the accelerator design. Due to the speed and efficiency of linear algebra algorithms running on digital computers, an analog accelerator that matches digital performance needs a large silicon footprint. Finally, we conclude that problem classes outside of systems of linear equations may hold more promise for analog acceleration.**

***Keywords*-accelerator architectures; analog-digital integrated circuits; analog computers; linear algebra.**

I. Introduction

In anticipation of the post-Moore's-law era of computing, there has been a scramble to either discover devices which can replace CMOS transistors, or to otherwise find ways to harness performance and efficiency from existing silicon technologies. Analog computing has been touted as one approach to address these challenges without the need for novel device technologies. For those familiar with the principles and history of analog computing, the expectations regarding analog computing's promise range from extremely positive to extremely negative.

Analog computing has many alluring properties: broadly, analog computing abandons digital representation of numbers, and also abandons step-by-step operation typical in modern computing. Much of research throughout computer architecture is in the line of breaking historical abstractions which hold back performance and efficiency of computers. Among the remaining abstractions yet to be broken are binary representation and discrete-time operation. In this regard analog computing may unleash untapped uses for existing CMOS technology. Arguments against analog computing include hardware design difficulty, low precision, limited scalability, programming difficulty, and even low performance improvements.

Analog computing effectively solves nonlinear ordinary differential equations, which frequently appear in cyber-physical systems workloads, with higher performance and efficiency compared to digital systems [1]–[4]. The analog, continuous-time output of analog computing is especially suited for embedded systems applications where actuators can use such results directly.

This work explores using analog computing in commodity mainstream computing systems. The main difference between the embedded application and this work is that, here we strive to use analog computing for solving a class of problems typically handled in digital computing (specifically, systems of linear equations as opposed to differential equations). Furthermore, we consider how analog computing can be leveraged if the outputs cannot be directly fed to actuators and have to be processed further digitally.

Towards this goal we present an architecture that allows analog computing results to be safely used with conventional architectures. The architecture is envisioned as an accelerator-style architecture with the digital processor acting as a host and the analog accelerator acting as a peripheral. Our architecture provides the digital host the ability to configure, control, and capture data from an analog accelerator, and to be able to react when problems occur in the course of analog computation. The choice of problem that we solve, system of linear equations, sets an extremely high bar for analog computing to challenge as the importance of this class of problems has led to highly efficient techniques in modern digital computing.

This work is a study of analog computing in the context of modern mainstream computer architecture. Using physical timing, power, and area measurements given by Guo *et. al.* [3], [4], we build a model that predicts the properties of larger scale analog accelerators. The perceived downsides of analog computing, such as low precision, and inability to sample intermediate results at high frequency, can be overcome. For instance, we find that precision of the results obtained from analog computing can be increased arbitrarily irrespective of the resolution of the analog-to-digital converter, and that there is also a way to divide large workloads into pieces that enables solution on limited analog hardware.

```
Data: time, steps, $a$, $b$, $u_{init}$
Result: evolution of $u$ over time in steps
$stepSize \leftarrow time \div steps$;
$u \leftarrow u_{init}$;
for $step \leftarrow 0$; $step < steps$; $step \leftarrow step + 1$ do
    $\delta \leftarrow a \times u + b$;
    $u \leftarrow u + stepSize \times \delta$;
end
```

Algorithm 1: Euler's method for Equation 1

As such programmability challenges can be overcome with proper support for exceptions and the ability to decompose and map problems.

On performance and energy metrics, we find that with high analog bandwidth, analog acceleration can potentially have 10× faster solution time and 33% lower energy consumption compared to a digital general-purpose processor. However, our analysis finds that high bandwidth in analog computers comes with high area cost, severely limiting the problem sizes that can be solved on an analog accelerator at a time. This ultimately limits the benefit of analog computing to this class of problems.

While our findings do not make a strong case for using analog accelerators for this important workload, our experience from using analog circuitry to accelerate digital computation provides guidance for future analog workloads and architectures.

The rest of the paper is organized as follows: Section II gives background on analog computation and the challenges that must be overcome in analog acceleration. Section III-A and III-B presents the microarchitecture and architecture for an analog accelerator. Section IV discusses problems and algorithms in scientific computing and how they can be solved using analog acceleration. Section V compares analog and digital computing solving sparse, structured grid problems. We analyze our empirical results and review perceptions about analog computing in Section VI. Section VII discusses related directions in analog computing. Section VIII concludes.

II. Analog Computing Background

This section is a tutorial on analog computation. Analog computing works by solving systems of ordinary differential equations (ODEs). We can also solve other types of problems by transforming them into ODEs. We discuss the main challenges that need to be overcome to use an analog accelerator in conjunction with a digital computer, on modern workloads.

A. Solutions to ODEs: Digital and Analog

Analog computers solve ODEs, which state the time derivatives of variables as functions of the variables. A simple linear first-order ODE has the form:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = au + b \tag{1}$$

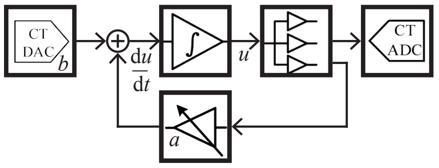

Figure 1. Mapping the ODE in Equation 1 to an analog accelerator. At the core an analog computing circuit is an integrator (the block labeled '$\int$'), which stores and gives as output a variable u, which is replicated to two branches using a fanout block. On one branch u is read out by an ADC. On the other branch u is multiplied with a coefficient a and added with a constant value b generated by a DAC. The function is fed back to the input of the integrator as $\frac{\mathrm{d}u}{\mathrm{d}t}$. The integrator charges to u_{init} and is released. The time-varying waveform for the variable is the ODE solution.

u is the time-varying variable we are solving for, a is a known coefficient, and b is a known constant bias. ODEs are specified with known initial conditions such as $u(0) = u_{init}$.

One class of numerical algorithms for solving ODEs are *explicit methods*, such as Euler's method shown in Algorithm 1. In explicit methods the time derivative $\frac{\mathrm{d}u}{\mathrm{d}t}$ is calculated for the present time instance, multiplied by a small time step, and added to u.

Analog computing does the same but in continuous time, using an infinitesimally small time period. Analog computers set up a datapath composed of units that perform multiplication, summation, and integration, as shown in Figure 1. The chip starts computation by releasing the integrator, allowing its output u to deviate from its initial value. The variable u can be the analog result of computation, or can be converted to digital by an analog-to-digital converter. Later, we will show a circuit for a system of ODEs solving for multiple variables at once.

B. Challenges in Analog Acceleration

While analog computing works well as an ODE solver for embedded systems, it is less clear how analog acceleration can be used in digital computing. Here we discuss the challenges in using analog acceleration in digital architectures.

Analog-to-digital conversion limits precision: In our example the time-varying variable u represents useful computation results that can directly control motors and actuators in embedded systems. An analog accelerator must convert the output to high precision digital numbers for use in the digital host. However, there is a trade-off between ADC sampling frequency and resolution, so in this work we use only the steady-state result of analog computing, which is easier to sample at high resolution. The highest precision ADCs still fall short of the precision in floating point numbers. To mitigate this, we discuss a technique to build up the precision of results in Section IV-A.

High hardware cost limits scalability: A physical integrator block is needed for every variable in the ODE. Furthermore, the analog datapath is fixed during computation and operates in continuous time, so there is no way to dynamically load variables from and store variables to main memory. Modern workloads routinely need thousands of integrators, exceeding area constraints of realistic analog accelerators. Large-scale problems must be decomposed into subproblems that can be solved in the analog accelerator. We discuss how sparse systems of linear equations can be decomposed in Section IV-B.

Different problems need significant reprogramming: Analog computing literature of the 1960s abounds with application-specific techniques for simulation and engineering design. However, digital computing has advanced significantly since the decline of analog computing, and now offers more flexibility and reliability, even if analog computing techniques offer high performance and efficiency. For analog acceleration to succeed, it must be able to accelerate a core kernel which is used extensively, without significant reprogramming to support new problems.

In order to address these architectural challenges, in this work we explore using analog computing to support solving sparse linear equations, which commonly arise in solving differential equations. We use the prototype analog accelerator presented in [3], [4] to validate the approach, and to serve as a basis for quantifying the performance, area cost, and energy efficiency of analog accelerators. We emphasize that this analog accelerator is designed primarily as an ODE solver, and is therefore not representative of an analog accelerator designed as a system of linear equations solver.

III. A Prototype Analog Accelerator

In this section we describe the microarchitecture of our analog accelerator and its hardware/software interface.

A. Analog Accelerator: Microarchitecture

Our research group recently prototyped an analog chip in 65nm CMOS technology [3], [4], shown in Figures 2 and 3. The accelerator consists of analog functional units connected with a crossbar. Each chip is organized as four macroblocks, each macroblock consisting of one analog input from off-chip, two multipliers, one integrator, two current-copying fanout blocks, and one analog output to off-chip. Two macroblocks share use of an 8-bit ADC, an 8-bit DAC, and a nonlinear function lookup table (256-deep, 8-bit continuous-time SRAM [5]). The chip also includes an interface to receive commands from the main digital processor. In the prototype these commands are received over an interface implementing an SPI protocol.

In our analog accelerator, electrical currents represent variables. Fanout current mirrors allow copying variables by replicating values onto different branches. To sum variables, currents are added together by joining branches. Eight multipliers allow variable-variable and constant-variable multiplication. The variables can also be subjected to arbitrary

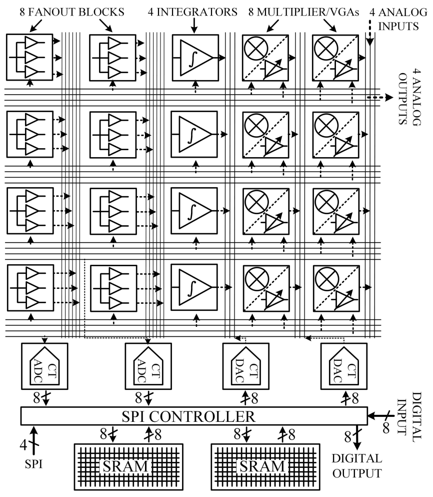

Figure 2. Chip layout diagram reproduced from [3], [4] showing rows of analog, mixed-signal, and digital components, along with crossbar interconnect. Each of the four rows of analog components are logically organized as a macroblock. "CT" refers to continuous time. SRAMs are used as lookup tables for nonlinear functions.

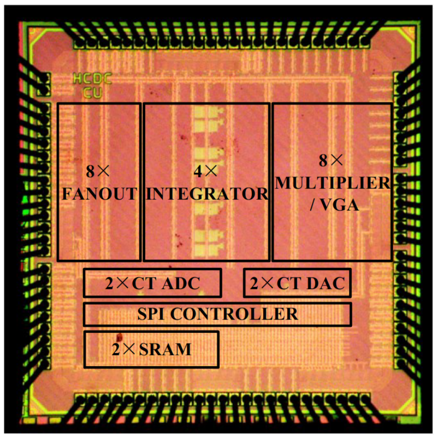

Figure 3. Die photo reproduced from [3], [4] of analog computer chip fabricated in 65 nm showing major components. "VGAs" are variable-gain amplifiers. Die area is 3.8 mm^2.

nonlinear functions, such as sine, signum, and sigmoid with the SRAM lookup table.

Overflow detection is done using analog voltage comparators to detect values exceeding the safe range. We compare a reference value (usually the maximum or minimum allowed values) to the signal carrying the variable. When a value exceeds the safe range an exception bit is set in a latch whose value can be read out during exception checking.

B. Analog Accelerator: Architecture

The analog accelerator acts as a peripheral to a digital host processor, which provides a configuration for the analog accelerator, performs calibration, controls computation, and reads out the output values. Table I summarizes the essential system calls and corresponding instructions for the analog

Instruction type	Instruction	Parameters	Description
Control	init		Find calibration codes for all function units
Config	setConn	source analog interface, destination analog interface	Create an analog current connection between the analog interfaces of two units
Config	setIntInitial	pointer to integrator, initial condition	Set integrator to have ODE initial condition value represented by the float value
Config	setMulGain	pointer to multiplier, gain	Set multiplier to have gain represented by the float value
Config	setFunction	pointer to lookup table, pointer to nonlinear function	Set lookup table to have nonlinear function represented by function pointer
Config	setDacConstant	pointer to DAC, constant bias	Set DAC to generate constant additive bias value represented by the float value
Config	setTimeout	timeout clock cycles	Set timer so analog computation, once started, stops after predetermined amount of time
Config	cfgCommit		Finish configuration and write any new configuration changes to chip registers
Control	execStart		Start analog computation by letting integrators deviate from their initial condition value
Control	execStop		Stop analog computation by holding integrators at their present value
Data input	setAnaInputEn	pointer to analog input	Open up chip's analog input channel, so outside stimulus can alter computation results
Data input	writeParallel	unsigned char data	Write to chip's digital input a value, which can be used by DAC or lookup table
Data output	readSerial	character array	Read from chip to character pointer the outputs of ADCs
Data output	analogAvg	pointer to ADC, number of samples	Record the digital output value of an ADC from multiple samples
Exception	readExp	character array	Read from chip to character pointer the exception vector indicating which analog units exceeded their operating range

Table I
ANALOG ACCELERATOR INSTRUCTION SET ARCHITECTURE

accelerator; we walk through how to use the instructions in the steps below.

Calibration: Before using the analog accelerator, the analog circuitry must first be calibrated. Numerical errors in analog computing come from three types non-ideal behaviors.

1) offset bias: a constant additive shift in values,
2) gain error: an error in how much values are multiplied,
3) nonlinearity: the possibility that the DC transfer characteristic has a non-constant slope.

The effect of these non-ideal behaviors varies between function units due to process variations. We use small DACs in each block to compensate for the first two sources of error by shifting signals and adjusting gains. These DACs are controlled by registers, whose contents are set during calibration by the digital host. The settings vary across different copies of the analog accelerator chip, but remain constant during accelerator operation and between solving different problems. When an analog unit is calibrated, its inputs and outputs are connected to DACs and ADCs; then, the digital processor uses binary search to find the settings that give the most ideal behavior. The third source of error, nonlinearity, is kept under control via overflow exception detection, which we discuss later.

Configuration: Before computation is offloaded to the accelerator, the programmer maps out the connections between analog units, along with settings of the units, and sends it to the analog accelerator using the configuration instructions. This configuration bitstream is written to digital registers on the analog accelerator. These digital registers contain only static configuration, akin to the program, and no dynamic computational data.

Computation: The architecture interface has instructions which control the start and stop of integration, which signify the beginning and end of analog computation.

Exceptions: A key aspect of the analog accelerator compared to prior analog computing designs is its ability to report exceptions. After computation is done, the chip can report if any exceptions occurred during analog computation. All analog hardware designs have a range of inputs where the output is linearly related to the input. Exceeding this range leads to clipping of the output, similar to overflow of digital number representations. The integrators and ADCs detect when their inputs exceed the linear input range, and these exceptions are reported to the digital host. At the same time, the host also observes if the dynamic range is not fully used, which may result in low precision. When such exceptions occur the original problem is scaled to fit in the dynamic range of the analog accelerator and computation is reattempted.

IV. ANALOG ACCELERATION FOR LINEAR ALGEBRA

In this section we discuss systems of linear equations, and how digital and analog computers solve them. Then, we discuss how systems of linear equations are used throughout scientific computing.

A. Systems of Linear Equations

Solving systems of linear equations entails finding an unknown solution vector $\mathbf{u}$ that satisfies $A\mathbf{u} = \mathbf{b}$, where A

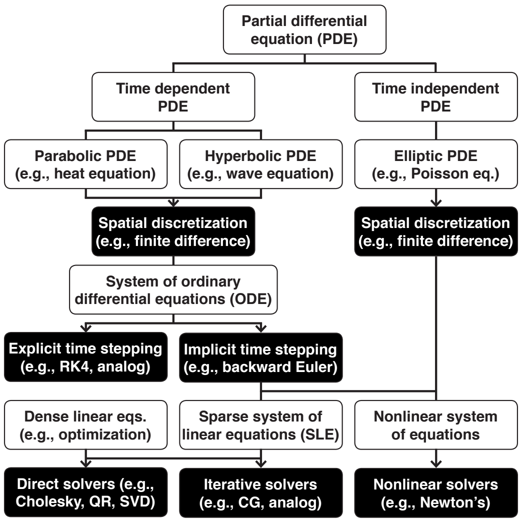
Partial differential equation (PDE)
Time dependent PDE
Time independent PDE
Parabolic PDE (e.g., heat equation)
Hyperbolic PDE (e.g., wave equation)
Elliptic PDE (e.g., Poisson eq.)
Spatial discretization (e.g., finite difference)
Spatial discretization (e.g., finite difference)
System of ordinary differential equations (ODE)
Explicit time stepping (e.g., RK4, analog)
Implicit time stepping (e.g., backward Euler)
Dense linear eqs. (e.g., optimization)
Sparse system of linear equations (SLE)
Nonlinear system of equations
Direct solvers (e.g., Cholesky, QR, SVD)
Iterative solvers (e.g., CG, analog)
Nonlinear solvers (e.g., Newton's)

Figure 4. Taxonomy of some classes of problems in scientific computation. Physical phenomena are described as partial differential equations. PDEs are solved by applying appropriate space and time discretizations, converting the continuous problem format into discrete node variables, interrelated by systems of algebraic equations. The dark boxes show steps to convert or solve problems. One way analog acceleration supports scientific computation is by acting as explicit solvers for systems of ODEs [3], [4]. This paper focuses on using analog computing as an iterative solver for sparse linear equations.

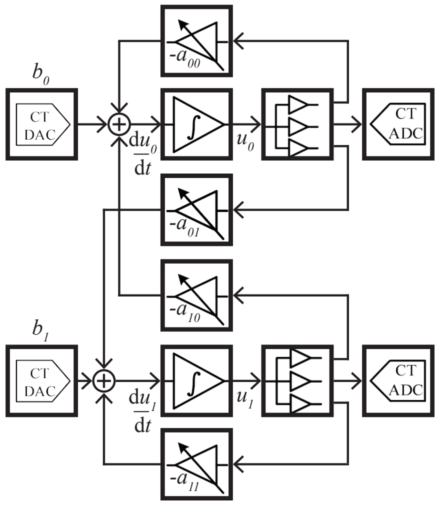

Figure 5. Solving a system of two linear equations with two unknowns in an analog accelerator.

is a matrix of known coefficients and $\mathbf{b}$ is a vector of known constant biases. Linear algebra algorithms that solve these problems include sparse matrix, dense matrix, structured, and unstructured grid algorithms, and are the bulk of the Berkeley Dwarfs taxonomy [6]. As shown in the bottom layer of Figure 4, scientific computation workloads mostly solve *sparse* linear algebra problems, where variables are loosely interconnected. Machine learning and optimization problems frequently solve *dense* linear algebra problems, where variables have all-to-all connectivity. Performance and efficiency gains in solving linear algebra problems would be highly beneficial.

Linear algebra techniques are categorized as direct and iterative solvers. *Direct solvers* focus on factoring the matrix, resulting in algorithms that assign correct values to the solution one element at a time. Notable direct solvers include Cholesky decomposition and Gaussian elimination. The literature on analog computing points out that analog computers are not suitable for direct linear algebra approaches [7]. On the other hand, *iterative solvers* start at an initial guess $\mathbf{u_{init}}$; the entire solution evolves step-by-step toward the correct answer according to an algorithm until the solution stops changing and is accurate at $\mathbf{u_{final}}$. Even if an iterative solver is stopped short of full convergence, the intermediate solution still approximately satisfies the original system of linear equations. Notable iterative solvers include conjugate gradients (CG) and steepest gradient descent.

In analog computing, we can imagine the iterative solver taking smaller steps more frequently, until it is taking infinitesimally small steps in continuous time. In continuous-time gradient descent, the time derivative of the solution vector is set to be the gradient pointing in the direction of the correct answer, resulting in the system of ODEs: $\frac{d\mathbf{u}}{dt} = \mathbf{b} - A\mathbf{u}(t)$. For example, a simple two-variable system of linear equations would be solved using the system of ODEs:

$$\frac{d}{dt}\begin{bmatrix} u_0(t) \\ u_1(t) \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} - \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}\begin{bmatrix} u_0(t) \\ u_1(t) \end{bmatrix} \tag{2}$$

This ODE can be mapped to analog hardware as shown in Figure 5. As $\mathbf{u}(t)$ evolves, the derivative approaches zero so long as A is a positive definite matrix. When the derivative becomes zero, the steady state value of $\mathbf{u}(t)$ satisfies the system of linear equations, and can be read out using ADCs. These techniques were used in early analog computers [8]–[14], and have been recently explored in small scale experiments with analog computation [15]–[18].

In contrast to solving time-varying ODEs, here the analog accelerator's ADCs only have to sample the value of the stable output $\mathbf{u_{final}}$, which means that sampling frequency is not a concern. If even higher precision is needed, more significant digits can can be obtained from the analog result by solving more times, each time setting $\mathbf{b}$ to be the residual, and scaling the problem up as necessary to fully use the dynamic range of the analog hardware. This procedure is shown in Algorithm 2. The longer sampling period, combined with Algorithm 2, mitigates concerns regarding ADC precision described in Section II-B.

Finally we note that imprecise solutions from analog acceleration are still useful in multigrid partial differential equation solvers. In multigrid PDE solvers, the overall PDE is converted to several linear algebra problems with

```
Data: $A$, $\mathbf{b}$
Result: $u_{precise}$ with high precision
$\mathbf{u_{precise}} \leftarrow \mathbf{0}$;
$\mathbf{residual} \leftarrow \mathbf{b}$;
while $||\mathbf{residual}|| > tolerance$ do
    analog accelerator solves $A\mathbf{u_{final}} = \mathbf{residual}$;
    $\mathbf{u_{precise}} \leftarrow \mathbf{u_{precise}} + \mathbf{u_{final}}$;
    $\mathbf{residual} \leftarrow \mathbf{b} - A\mathbf{u_{precise}}$;
end
```

Algorithm 2: Building precision in analog result

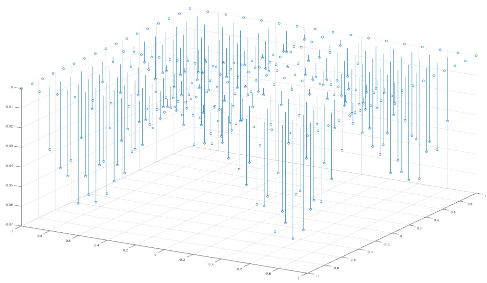

Figure 6. An example elliptic PDE. The continuously varying field has been discretized into node variables which are solved using linear algebra.

varying spatial resolution. Lower-resolution subproblems are quickly solved and fed to high-resolution subproblems, aiding the high-resolution problem to converge faster. The linear algebra subproblems can be solved approximately. Overall accuracy of the solution is guaranteed by repeating the multigrid algorithm. Because perfect convergence is not required, less stable, inaccurate, low precision techniques, such as analog acceleration, may also be used to support multigrid.

B. Linear Algebra for Elliptic Partial Differential Equations

As shown in Figure 4, elliptic partial differential equations are a fundamental class of PDEs. This class of problems is important in physical field simulations, such as fluid dynamics and solid mechanics. While such problems are not intrinsically difficult to solve, they dominate scientific computation workloads, and can take significant of computing power to solve when the problems have high dimensionality, high spatial resolution, and when they must be solved to high accuracy and precision.

The 2D Poisson elliptic PDE has the form $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = b(x, y)$. The continuous spatial partial derivatives indicate that $u(x, y)$ varies continuously over 2D space. The problem is discretized into $L \times L$ grid points, converting the continuous field into node variables. For example, using a 3×3 grid on the unit square:

u_0	u_1	u_2
u_3	u_4	u_5
u_6	u_7	u_8

would result in nine node variables in the vector $\mathbf{u}$ which are interrelated according to the system of linear equations:

$$A\mathbf{u} = \mathbf{b}$$

$$A = \frac{1}{\frac{1}{3^2}} \begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{bmatrix}$$

$$\mathbf{u} = [u_0, u_1, \ldots, u_8]^\top, \mathbf{b} = [b_0, b_1, \ldots, b_8]^\top$$

The coefficients in the matrix are a result of using a second-order central finite difference stencil. The coefficient value of 9 in front of A emerges because we discretized the 2D unit square into thirds on each side. Notice A is sparse, meaning that most coefficients are zero, a result from the fact cells are only related to itself, and to its four neighbors.

In practice, physics simulations using PDEs have millions of grid points in the vector $\mathbf{u}$, far larger than the problem sizes that can fit in an analog accelerator. Both digital and analog techniques would subdivide the large grid size problem into smaller linear problems. For example, the 3×3 2D problem can be solved as a set of three independent 1D subproblems:

$$A_s \mathbf{u_s} = \mathbf{b_s}$$

$$A_s = \frac{1}{\frac{1}{h^2}} \begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix}, \mathbf{u_s} = \begin{bmatrix} u_{s0} \\ u_{s1} \\ u_{s2} \end{bmatrix}, \mathbf{b_s} = \begin{bmatrix} b_{s0} \\ b_{s1} \\ b_{s2} \end{bmatrix},$$

This decomposition temporarily ignores the coefficients that connect the 1D problems into a 2D problem. The subproblems can be solved separately on multiple accelerators, or multiple runs of the same accelerator.

Solving the system of equations as block matrices only ensures that the solution vector $\mathbf{u_s}$ is correct for the subproblem. To get overall convergence across the entire problem, the set of subproblems would be solved several times, using a larger iteration across the subproblems. Typically, the larger iteration is an iterative method operating on *vectors*, and do not have as strong convergence properties as iterative methods on individual numbers. Therefore, it is still desirable to ensure the block matrices are large, so that more of the problem is solved using the efficient lower level solver.

Using this domain decomposition technique, in conjunction to accuracy boosting and a multigrid algorithm we can use the analog accelerator to calculate an elliptic PDE solution as shown in Figure 6. These divide-and-conquer and approximate computing techniques in solving PDEs mitigates concerns regarding precision and scalability described in Section II-B.

V. Methodology and Evaluation

In this section we compare analog and digital computation in terms of performance, hardware area, and energy consumption, using 2D Poisson PDEs as an example problem. We take into account accuracy, problem size, and the bandwidth of the analog accelerator design.

Accuracy: We compare the analog accelerator and the digital algorithm running on a CPU at equal solution accuracy, measured as the error in the solution. This is done by stopping the numerical iteration in the digital version well short of machine epsilon provided by high-precision digital floating point numbers. The stopping criterion is when no element in the output vector $\mathbf{u}$ changes by more than $1/256$ of full scale. This is equivalent to the amount of precision in the output vector that can be obtained from one run of the analog accelerator.

Problem size: We vary the total number of grid points in the problem $N = L^2$, where L is the number of increments on one side. We compare analog and digital solutions for grids of up to 2048 points. The relatively small problem size is due to the lack of dense data storage for analog variables in analog accelerators. As a case in point, the fabricated prototype chip has only four integrators to hold variables. Nonetheless, the chip establishes confidence in using the analog blocks for computation. Using the validated schematics we build circuit simulations in Cadence® Virtuoso®, in order to extrapolate the area and energy consumption of larger scale analog accelerators.

Bandwidth: The most important parameter in the analog accelerator design is the analog components' bandwidth. Increasing the bandwidth of the analog circuit design proportionally decreases the solution time, but also increases area and energy consumption. We do a design space exploration of analog accelerators with different bandwidths.

A. Analog and Digital Computation Time

We compare the time it takes for the analog accelerator and a digital algorithm to solve a 2D Poisson PDE.

In digital computing, the PDE can be solved using many linear algebra algorithms. Figure 7 establishes that conjugate gradients (CG) has the best convergence rate among classical iterative methods. The CG algorithm is implemented using stencils to capture the sparse structure of the matrix, without having to allocate memory for the full matrix, and avoiding iterating through the rows and columns of the matrix. We measure the computation time on single threaded code, running on an Intel Xeon X5550, clocked at 2.67 GHz. The problem sizes we tackle are smaller than 2048 total grid points, so the program data is entirely resident in the first level cache.

Figure 8 shows that the prototype analog design would have parity in terms of computing speed once it reaches a size of roughly 650 integrators. In Section VI-D we give a theoretical model why the analog computer's solution time

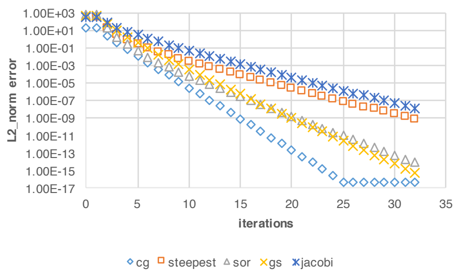

Figure 7. Comparison of the convergence rate for a Poisson equation. The L2-norm of the error is plotted against the number of numerical iterations. The numerical algorithms are conjugate gradients, steepest descent, successive over-relaxation, Gauss-Seidel, and Jacobi iterations. We see CG converges to a solution limited by the precision of double precision floating point numbers the quickest. The problem is discretized using finite differences with 16 points over three dimensions, for a total of 4096 grid points. Boundary condition $u(x, y, z) = 1.0$ for the plane $x = 0$, $u(x, y, z) = 0.0$ otherwise.

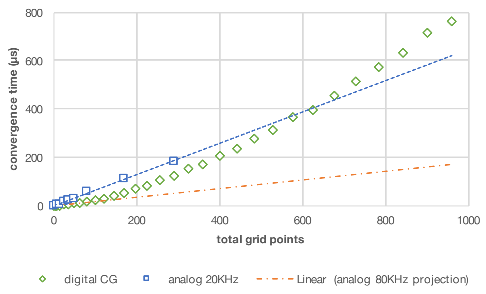

Figure 8. Comparison of time taken to converge to equivalent precision, for an analog accelerator and a CPU. The time needed to converge is plotted against the total number of grid points $N = L^2$. The convergence time for an analog solution is measured from simulations of larger analog accelerator circuits based on the prototyped hardware. We give the projected the solution time for an 80 KHz bandwidth analog accelerator design. The convergence time for the digital comparison is the software runtime on a single CPU core.

scales linearly with respect to the problem size, measured in grid points. An analog accelerator with 650 integrators occupies about 150 mm^2, accounting for integrators, multipliers, current mirrors, DACs, and ADCs; this is smaller than desktop CPU die sizes.

B. Choice of ADC Precision and Analog Bandwidth

In this section we explore the timing, area, and energy costs of high-bandwidth analog accelerators equipped with higher-resolution ADCs.

Choice of ADC resolution: The ADC conversion resolution is a limiting factor in the effectiveness of analog acceleration, as discussed in Sections II-B and IV-A. The prototype analog accelerator is equipped with 8-bit ADCs, which limits the precision that the digital CG algorithm has to achieve for an equivalent result. We assume the model analog accelerator has 12-bit ADCs, which increases the accuracy of the analog acceleration result, forcing the CG

Unit type	Power	Core power fraction	Area	Core area fraction
integrator	28 μW	80%	0.040 mm^2	40%
fanout	37 μW	80%	0.015 mm^2	33%
multiplier	49 μW	80%	0.050 mm^2	47%
ADC	54 μW	50%	0.054 mm^2	83%
DAC	4.6 μW	100%	0.022 mm^2	61%

Table II
SUMMARY OF ANALOG CHIP COMPONENTS TAKEN FROM [3], [4].

comparison to run for more iterations to achieve the same level of accuracy.

Choice of analog bandwidth: The prototype chip is designed as an ODE solver for embedded systems, with a relatively low bandwidth of 20 KHz, a design that ensures that the prototype chip accurately solves for time-dependent solutions in ODEs. The reason that high bandwidth is not used when solving ODE dynamics is that high bandwidth designs are more sensitive to parasitic effects, which degrade the solution's accuracy. However, the small bandwidth of the prototype makes it unrepresentative of an analog accelerator designed to solve time-independent algebraic equations, where accuracy degradation in time-dependent behavior has no impact on the final steady state output.

Power and area scaling: We scale up the bandwidth of the model, within reason, to up to 1.3 MHz to explore the performance, area, and energy traits of a high bandwidth design. We assume an analog accelerator with bandwidth multiplied by a factor of α has higher power and area consumption in the core analog circuits, by a factor of α.

For power/bandwidth, we observe that analog circuits operate faster when the internal node voltages representing variables change faster. As such, we need larger currents to charge and discharge the node capacitances in the signal paths carrying variables. A derivation shows that:

- (node voltage change) = (charge change) / capacitance
- (charge change) = time * (charging current)
  = (charging current) / frequency
- (node voltage change)
  = (charging current) / (frequency * capacitance)

We hold the capacitance fixed to the capacitance of the prototype's design—this is a conservative decision: careful design may permit smaller choices of capacitance. From this derivation we see the bandwidth, represented here as frequency, is linearly related to charging current, which is finally linearly related to the power consumption.

For area/bandwidth, we observe that the transistor aspect ratio W/L has to increase to increase the current, and therefore bandwidth, of the design. L is kept at a minimum dictated by the technology node, leaving bandwidth to be linearly dependent on W. Thus we estimate area increasing linearly with bandwidth. The assumption on area scaling is conservative; higher bandwidth may be obtained for less than proportional increase in area.

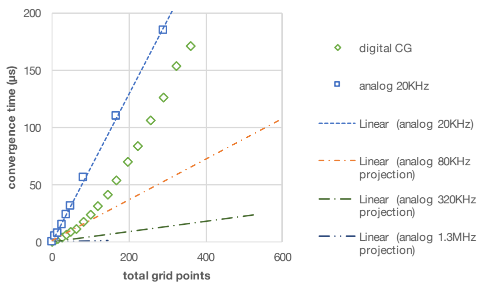

Figure 9. Comparison of time taken to converge to equivalent precision, for high bandwidth analog accelerators and a digital CPU. The time needed to converge is plotted against the total number of grid points $N = L^2$. We give the projected solution time for 80 KHz, 320 KHz, and 1.3 MHz analog accelerator designs. The high bandwidth designs have increasing area cost. In this plot the 320 KHz and 1.3 MHz designs hit the size of 600 mm^2, the size of the largest GPUs, so the projections are cut short. The convergence time for digital is the software runtime on a single CPU core.

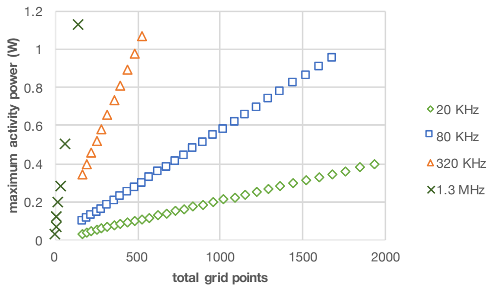

Figure 10. The power consumption of analog accelerators as a function of number of grid points it can simultaneously solve. The 20 KHz design is the prototyped analog accelerator. Higher bandwidth designs are projections from the prototype.

Table II shows the area and power consumption of the components of the prototype analog accelerator chip. The core power and area fraction show the fraction of each block that form the analog signal path. The area and power for core components that touch the analog variables scale up and down for different bandwidth designs. Not all area and power consumption of the blocks of the prototype design are involved in the analog signal path, and do not need to scale up for higher bandwidth designs. The non-core transistors and nets not involved in analog computation include calibration and testing circuits, and registers. The physical power and area measurements from the prototype analog accelerator provides a basis for projections, using this scaling model, summarized in Figures 10 and 11.

We compare the solution energy of analog and digital solvers in Figure 12. Using an estimate of 225 pJ for every floating point multiply-add operation in GPUs [19], we derive the amount of energy needed for GPUs to compute the solution to equivalent accuracy as the analog accelerator. The conjugate gradient algorithm uses a sustained 20 clock cycles per numerical iteration per row element. The comparison assumes identical transfer cost of data from

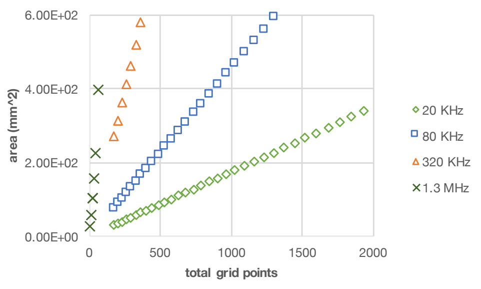

Figure 11. The area of analog accelerators as a function of number of grid points it can simultaneously solve.

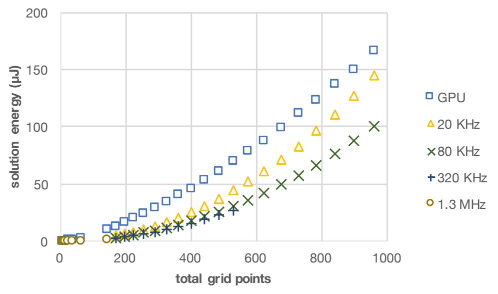

Figure 12. The energy needed to solve 2D problems of varying number of total grid points, for different analog accelerator designs, compared against a GPU running CG. The 80 KHz design shows some energy savings relative to the GPU. High bandwidth analog accelerators are quickly limited by its large chip area cost and cannot solve problems with many grid points. Furthermore, because not all power and area is spent on the analog critical path, efficiency gains cease after bandwidth reaches 80 KHz.

main memory to the accelerator versus the CPU: the energy needed to transfer data to and from memory is not modeled, due to the relatively small problem sizes, allowing the program data to be entirely cache resident.

While increasing the analog bandwidth directly improves solution speed, high bandwidth designs are limited in the number of variables that can fit in $600\ \mathrm{mm}^2$, the size of the largest GPU dies. As bandwidth increases, a higher fraction of area and power consumption becomes directly involved in analog computation, resulting in a more energy-efficient design. Once almost all of the power consumption is directly involved in analog computation, increasing bandwidth results in a proportional increase in power and decrease in computation time, so the efficiency gains do not increase after bandwidth reaches $80\ \mathrm{KHz}$. We conclude that analog accelerators need as high bandwidth as area permits for high speed solution. Analog acceleration may offer some efficiency gains for linear algebra, but not by a significant factor.

VI. Analysis of Results

In this section we analyze our experimental results and re-examine claims regarding the perceived benefits of analog computing. We investigate how algorithms, continuous-time computation, continuous-value representation, and choice of the targeted problem each impact the performance and efficiency of analog and digital techniques.

A. *Continuous-Time: Advantages*

In analog computation variables evolve continuously: they are multiplied and summed in continuous time. No power-hungry clock signal is needed to synchronize operations. We observe these low-power traits in the analog accelerator: even in the designs that fill a $600\ \mathrm{mm}^2$ die size, the analog accelerator uses about 0.7 W in the base prototype design and about 1.0 W in the design with $320\ \mathrm{KHz}$ bandwidth. The projected analog power figures are significantly below the TDP of clocked digital designs of equal area.

Continuous-time analog computation is also implicitly asynchronous. The outputs of functional units respond, to an extent limited by bandwidth, immediately to their inputs. For example, a non-zero input to an integrator causes change to the integrator's stored variable, and that change is immediately reflected in its output for other units to consume. The variables in integrators and on datapaths all evolve until the system is at steady state, distinct from how only groups of variables change at a time in discrete-time computation.

In discrete-time iterative linear algebra algorithms, the solution vector changes in steps, and each step is characterized by a step size. The step size affects the algorithm's efficiency, and requires many cycles to calculate. In CG, for example, the step size is calculated from the gradient magnitude and takes up half of the multiplication operations in each step. In the analog accelerator, the step size is reduced to an infinitesimal value, sidestepping the notion of discrete step sizes. Instead, numerical convergence in the analog accelerator is limited only by the bandwidth of the analog components.

B. *Continuous-Time: Disadvantages*

While discrete-time evolution has drawbacks, it permits algorithms to intelligently select a step size, which has advantages in solving systems of linear equations. Both solvers are performing iterative numerical algorithms, but the digital program runs conjugate gradients, the most efficient and sophisticated of the classical iterative algorithms. In CG, each step size is chosen, taking into account the gradient magnitude of the present point, along with the history of step sizes. With these additional calculations, CG avoids taking redundant steps, accelerating toward the answer when the error is large, slowing when close to convergence [20]. Note in Figure 7, the CG method has the steepest slope on a log-linear chart, more efficient than any other method presented.

In contrast, the analog accelerator has a limited variety of iterative algorithms it can carry out. In using the analog accelerator for linear algebra, the rate of convergence is limited by the bandwidth of the design, so the convergence rate within a time interval cannot be arbitrarily large. Therefore, the numerical iteration in the analog accelerator is akin to fixed-step size relaxation or steepest descent. While we can consider the analog accelerator as doing continuous-time steepest descent, taking many infinitesimal steps in continuous time, doing many iterations of a poor algorithm is in this case no match for a better algorithm. Efficient discrete-time algorithms such as CG, multigrid, and spectral methods were known to researchers by the 1950s. Analog computers remained in use in the 1960s to solve steepest descent due to their better immediate performance relative to early digital computers.

C. Continuous-Value: Advantages

Changing the value of a digital binary number affects many bits. For example, sweeping an 8-bit unsigned integer from 0 to 255 needs 502 binary inversions. Using more economical Gray coding, 255 inversions are still needed to sweep the range of an 8-bit integer. In general, the amount of charge needed for binary arithmetic is an exponential function of precision. To worsen the case for digital, real variables are usually encoded in floating point, which are costlier per operation. The logarithmically encoded exponent portion of floating point variables makes adding and subtracting variables complicated.

Analog computing is economical because real values are encoded in physical attributes, such as electrical current. The amount of energy needed to change the value of a variable is proportional to the size of the change in value. The precision of an analog variable is only limited by its signal to noise ratio. In effect, a single wire can capture many bits of information. Finally, no special hardware is needed to sum and subtract analog values encoded as current. The analog crossbars can sum values by simply joining branches.

D. Continuous-Value: Disadvantages

Despite its efficiency, continuous-value representation in the analog accelerator has drawbacks when used to assist digital computing. While the computation taking place inside the accelerator takes place at high precision, ADC conversion of the results is not so favorable. Each time the analog accelerator runs to solve an equation, the digital host only obtains as many bits of precision as the ADC conversion. At the levels of ADC precision we consider, $8 - 12$ bits, the digital algorithm takes only a few iterations to reach the same level of precision. On the other hand, while operation on floating points is costly, the digital algorithm can continue operating on the same set of data until precision is limited by the precision of floating point numbers.

Furthermore, floating point numbers are more able to represent variables with high dynamic range. In contrast, the problem's coefficients and constants must fit in the range of gain provided by multipliers and the output range of DACs. In order to multiply and add large numbers, the analog accelerator must use a procedure that scales down multiplication coefficients and added constants, but extends the amount of the time it takes to solve a problem (see inset).

Scaling the dynamic range of equation variables into that of the circuit: any system of linear equations of the form $A\mathbf{u} = \mathbf{b}$, with arbitrarily large magnitude coefficients in the A matrix, $\mathbf{b}$ vector, and solution $\mathbf{u}$, can be scaled to fit in the dynamic range of the analog computer. The solution is found using the convergent system of ODEs $\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{b} - A\mathbf{u}(t)$, where A is positive definite, subject to an initial condition on $\mathbf{u}(0) = \mathbf{u_0}$. The closed form solution for $\mathbf{u}(t)$, at some instant of time t, is:

$$\mathbf{u}(t) = A^{-1}\mathbf{b} + \mathbf{c}\mathrm{e}^{-At}$$

$$\mathbf{c} = \mathbf{u_0} - A^{-1}\mathbf{b}$$

Where e^{At} is the matrix exponential. When we use the analog accelerator as a linear algebra solver, the system is solved when:

$$\mathrm{e}^{-At} = \mathbf{0}$$

Now, suppose A has some element with value sg that exceeds the maximum gain g that the multipliers can give as coefficients. We can scale down the magnitude of A and instead program into the analog accelerator the matrix $A_s = \frac{A}{s}$ that has gains that are in the acceptable range. For the closed form equations to hold:

$$\mathbf{u}(t) = A_s^{-1}\frac{\mathbf{b}}{s} + \mathbf{c}\mathrm{e}^{-A_s st}$$

$$\mathbf{c} = \mathbf{u_0} - A_s^{-1}\frac{\mathbf{b}}{s}$$

We see that the result $\mathbf{u}(t)$ remains unchanged so long as we also scale down $\mathbf{b}$ by s, and scale up time t by a factor s. That is, given limited bandwidth in the system, we have restricted the dynamic range in A by extending the time it takes for the ODE to simulate. This is referred in the literature as value and time scaling; correct selection of scaling parameters can be challenging when using analog computers [7], [8], [10], [21], [22].

For example, when the two dimensional Poisson equation, defined on the unit square, is discretized with L increments to a side into system of linear equations, the absolute value of the elements inside the coefficients matrix increases in proportion to L^2. In order to map these matrices with larger magnitude coefficients into the dynamic range of the multipliers, we must scale down the elements of the matrix by L^2. In exchange, the analog computer requires more time, proportional to L^2, in order to solve the equation.

This ability for analog computers to trade dynamic range in variables by extending the computation time is a useful trick. But in comparison to computing on floating point numbers which have much higher dynamic range, this need to scale variables is a burdensome trade off.

E. Dimensionality

In the 2D Poisson elliptic equation example, we solved system of linear equations with coefficient matrices that result from discretization of two-dimensional space. These matrices have a sparse pentadiagonal form, meaning coefficients are non-zero along only five diagonals of the matrix. We now explore the scaling trends for 1D, 2D, 3D sparse matrices, as shown in Table III.

In the 2D example, analog acceleration follows a favorable scaling trend compared to CG, but the energy scaling favors

	Grid points	Analog			Conjugate gradients		
		HW cost	Conv. time	Energy=HW×time	Convergence steps	Time per step	Time and energy
1D	$N = L$	$N = L$ integrators	$N = L$	$N^2 = L^2$	$N = L$	$N = L$	$N^2 = L^2$
2D	$N = L^2$	$N = L^2$ integrators	$N = L^2$	$N^2 = L^4$	$N^{0.5} = L$	$N = L^2$	$N^{1.5} = L^3$
3D	$N = L^3$	$N = L^3$ integrators	$N = L^3$	$N^2 = L^6$	weak dependence	$N = L^3$	$N = L^3$

Table III
TIME, AREA, AND ENERGY TRENDS FOR ANALOG ACCELERATION AND CONJUGATE GRADIENTS, FOR DIFFERENT TYPES OF CONNECTIVITY BETWEEN VARIABLES, WHICH AFFECTS THE A MATRIX. N DENOTES THE NUMBER OF VARIABLES. L DENOTES THE NUMBER OF INCREMENTS PER DIMENSION.

CG. The overall effect there is a range of number of variables being solved where analog possibly wins in both speed and energy consumption.

In 3D problems, analog acceleration is not feasible, due to comparable scaling of solution speed and unfavorable scaling of energy consumption. Changing the dimensionality of the problem from 2D to 3D poses no significant challenges to a software algorithm. For each node value, the stencil will request node values in neighbors in all three dimensions. The node values for neighbors in the highest order dimension will be least recently used, and will have the least data locality. Compared to 2D problems, 3D problems have a larger data cache footprint, and an increase in the cache access stride length.

Analog computing, on the other hand, faces greater challenges in creating a hardware mapping for 3D problems on a 2D chip, due to difficulty in laying out the integrators in a way that balances and minimizes the analog interconnects.

F. Targeted Problem Class

Efficient linear algebra algorithms form the heart of modern continuous math workloads. These linear algebra algorithms operate on discrete-value variables which evolve in discrete time steps, which are approximations of the real dynamics of the physical world. Our objective was to apply analog computing to sparse systems, a fundamental kernel found in applications, with the hypothesis that a continuous model would yield benefits. Nonetheless, the intense demand for efficient linear algebra has led to powerful digital algorithms, optimized to run well on digital hardware, that make discrete approximations worthwhile, making the baseline in this study difficult to beat.

The analog accelerator is fundamentally an ODE dynamics simulator, meaning useful computational results are in the dynamic output waveform. As discussed in Section II-B, the dynamic output waveform has uses in embedded systems, where analog computation results are directly useful in driving actuators. In this paper we consider analog acceleration for digital computers, which potentially limits useful computation results to the steady state output, which can be precisely measured with slow, high-precision ADCs.

Analog acceleration may have greater benefits in other domains of continuous math, such as solving *nonlinear* PDEs. As shown in Figure 4, the solution of nonlinear PDEs proceeds in the same way as in the linear case, typically with discretization into nonlinear ODEs, then using implicit solvers that require solving systems of algebraic equations at each time step. The key difference is these are now *nonlinear* systems of equations, requiring Newton-Raphson method-based iterative solvers. These iterative solvers have continuous time formulations, which again involve solving ODEs of the form $\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = f(\mathbf{u}(\mathbf{t}))$. It is within our near future work to investigate how analog techniques can solve nonlinear problems, which can be vexing for digital algorithms.

VII. RELATED WORK

Analog electronic computers were used in the 1950s and 1960s for scientific simulations, including problems in optimization, ODEs, and PDEs [8]–[10], [21], [23]. By 1968 attention shifted to hybrid analog-digital computers, which introduced digital computers to provide capacious memory and ability to do discrete-time algorithms [11]–[14]. In the years since, digital computers, which provided the convenience and noise margin of binary variable encoding, capacious memory, and versatile numerical algorithms, eliminated analog computing from general use.

The development of analog and hybrid computers ran in parallel with the development of *digital differential analyzers*, which were digital numerical processors where variables were encoded in binary and evolved in discrete time. The digital units in DDAs were connected in the same topology of an analog computer, according to the differential equation being solved [7]. These designs faced difficulties in number dynamic range and scaling, which led to the development of extended resolution and floating-point variants of DDAs [22], [24]. These area-intensive function units were used in a time-multiplexed fashion, previewing the development of modern floating-point pipelines [25].

Computing using analog signals is resurgent in architecture research, due to challenges in IC scaling that limit the power dissipated by digital circuits [26], [27]. Analog computing circuits can be instantiated alongside digital circuits to handle workloads from a direct, numerical standpoint [1]–[5], [18], [28], [29], or to handle workloads developed in the field of neuromorphic computing.

In neuromorphic computing, broad classes of applications using both discrete and continuous variables are mapped to a variety of neural network topologies [30]–[32]. The neural networks can in turn be simulated as software, or can be simulated using resistive analog networks and A/D conversion, giving efficient multiplication and nonlinear lookups [33]. In neuroscience, analog circuits have been investigated for modeling ODEs that describe non-trivial neurons.

We draw distinction between our approach to analog acceleration and that of neuromorphic computing. Most important, we do *not* use training to get a network topology and weights that solve a given problem. No prior knowledge of the solution or training set of solutions is required. The analog acceleration technique presented here is a *procedural* approach to solving problems: there is a predefined way to convert a system of linear equations under study into an analog accelerator configuration.

Second, the analog computation presented in our work thrives on the possibility of connecting outputs of integrators to their inputs. This is in contrast to most neuromorphic computing approaches, which use cellular neural networks, autoencoders, and multilayer perceptrons, which are purely feedforward networks. The full crossbar between analog components allow any topology, including cycles, between components. In neural network terminology such topologies are recurrent neural networks and Hopfield networks, and represent the most powerful class of networks.

Among other emerging architectures, the quantum algorithm for linear systems of equations allows *characterization* of the solution vector in order of log$N$ time, where N is the size of the solution vector [34]. The solution is not readily measurable, but the algorithm has uses in algorithms that rely on linear algebra [35].

VIII. Conclusions

In this work we discussed how analog computing is different from digital computing in these key aspects: the variables evolve continuously in time; the variables have a continuous range of values; and the algorithm executed on the hardware is distinct. We presented an architecture for using solutions given by an analog accelerator, and we discussed methods to control downsides of analog computing such as limited problem size, limited dynamic range, and limited precision.

We used analog acceleration to solve systems of linear equations, an attractive and important application for this novel hardware design. Among linear algebra problems with 2D connectivity, analog acceleration may have both $10\times$ higher performance and 33% lower energy consumption, within a range of problem sizes. Notably, the obvious approach to improving analog accelerator performance, by increasing the analog circuit bandwidth, provides speedups, and can offer solutions requiring less energy, but has high area costs.

We recognize the performance increases and energy savings are not as drastic as one expects when using a fundamentally different computing model than digital, synchronous computing. This is primarily due to highly efficient algorithms for digital computers which are the dominant factor in comparing solver systems, implemented as either digital or analog hardware. Other numerical subroutines, such as those used in finding solutions to nonlinear systems of equations, present greater challenges to existing algorithms, and may show promise for analog computing.

Acknowledgment

This work is supported by NSF award CNS-1239134 and a fellowship from the Alfred P. Sloan Foundation.

References

[1] G. Cowan, R. Melville, and Y. Tsividis, “A VLSI analog computer/math co-processor for a digital computer,” in *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, Feb 2005, pp. 82–586 Vol. 1.

[2] ——, “A VLSI analog computer/digital computer accelerator,” *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 1, pp. 42–53, Jan 2006.

[3] N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis, “Continuous-time hybrid computation with programmable nonlinearities,” in *European Solid-State Circuits Conference (ESSCIRC), ESSCIRC 2015 - 41st*, Sept 2015, pp. 279–282.

[4] ——, “Energy-efficient hybrid analog/digital approximate computation in continuous time,” *Solid-State Circuits, IEEE Journal of*, vol. 51, Jul 2016, to be published.

[5] B. Schell and Y. Tsividis, “A clockless ADC/DSP/DAC system with activity-dependent power dissipation and no aliasing,” in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, Feb 2008, pp. 550–635.

[6] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, “The landscape of parallel computing research: A view from Berkeley,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[7] B. Ulmann, *Analog Computing*. Oldenbourg Wissenschaftsverlag, 2013.

[8] W. Karplus and W. Soroka, *Analog Methods: Computation and Simulation*, ser. McGraw-Hill series in engineering sciences. McGraw-Hill, 1959.

[9] A. Jackson, *Analog computation*. McGraw-Hill, 1960.

[10] S. Fifer, *Analogue Computation: Theory, Techniques, and Applications*, ser. Analogue Computation: Theory, Techniques, and Applications. McGraw-Hill, 1961, no. v. 3.

[11] G. Bekey and W. Karplus, *Hybrid computation*. Wiley, 1968.

[12] W. Chen and L. P. McNamee, "Iterative solution of large-scale systems by hybrid techniques," *IEEE Transactions on Computers*, vol. C-19, no. 10, pp. 879–889, Oct 1970.

[13] W. J. Karplus and R. Russell, "Increasing digital computer efficiency with the aid of error-correcting analog subroutines," *Computers, IEEE Transactions on*, vol. C-20, no. 8, pp. 831–837, Aug 1971.

[14] G. Korn and T. Korn, *Electronic analog and hybrid computers*. McGraw-Hill, 1972.

[15] C. C. Douglas, J. Mandel, and W. L. Miranker, "Fast hybrid solution of algebraic systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 11, no. 6, pp. 1073–1086, 1990. [Online]. Available: http://dx.doi.org/10.1137/0911060

[16] Y. Zhang, "Revisit the analog computer and gradient-based neural system for matrix inversion," in *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterranean Conference on Control and Automation*, June 2005, pp. 1411–1416.

[17] Y. Zhang and S. S. Ge, "Design and analysis of a general recurrent neural network model for time-varying matrix inversion," *Neural Networks, IEEE Transactions on*, vol. 16, no. 6, pp. 1477–1490, Nov 2005.

[18] S. Sethumadhavan, R. Roberts, and Y. Tsividis, "A case for hybrid discrete-continuous architectures," *IEEE Comput. Archit. Lett.*, vol. 11, no. 1, pp. 1–4, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1109/L-CA.2011.22

[19] S. Keckler, W. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *Micro, IEEE*, vol. 31, no. 5, pp. 7–17, Sept 2011.

[20] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

[21] B. Wilkins, *Analogue and iterative methods in computation, simulation, and control*, ser. Modern electrical studies. Chapman and Hall, 1970.

[22] R. B. McGhee and R. N. Nilsen, "The extended resolution digital differential analyzer: A new computing structure for solving differential equations," *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 1–9, Jan 1970.

[23] W. Karplus, *Analog simulation: solution of field problems*, ser. McGraw-Hill series in information processing and computers. McGraw-Hill, 1958.

[24] J. L. Elshoff and P. T. Hulina, "The binary floating point digital differential analyzer," in *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference*, ser. AFIPS '70 (Fall). New York, NY, USA: ACM, 1970, pp. 369–376. [Online]. Available: http://doi.acm.org/10.1145/1478462.1478516

[25] G. Hannington and D. G. Whitehead, "A floating-point multiplexed DDA system," *IEEE Transactions on Computers*, vol. C-25, no. 11, pp. 1074–1077, Nov 1976.

[26] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 365–376. [Online]. Available: http://doi.acm.org/10.1145/2000064.2000108

[27] M. B. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *Design Automation Conference*, 2012.

[28] E. H. Lee and S. S. Wong, "A 2.5GHz 7.7TOPS/W switched-capacitor matrix multiplier with co-designed local memory in 40nm," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 418–419.

[29] S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, R. Wunderlich, S. Nease, and S. Ramakrishnan, "A programmable and configurable mixed-mode FPAA SoC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–9, 2016.

[30] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. Lipasti, A. Nere, S. Qiu, M. Sebag, and O. Temam, "BenchNN: On the broad potential application scope of hardware neural network accelerators," in *Proceedings of the 2012 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 36–45. [Online]. Available: http://dx.doi.org/10.1109/IISWC.2012.6402898

[31] S. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. Basilico, S. Carpin, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. Kusnitz, T. Wong, W. Risk, E. McQuinn, T. Nayak, R. Singh, and D. Modha, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, Aug 2013, pp. 1–10.

[32] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014, pp. 609–622.

[33] R. St. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 505–516, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2678373.2665746

[34] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Phys. Rev. Lett.*, vol. 103, p. 150502, Oct 2009. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.103.150502

[35] S. Aaronson, "Read the fine print," *Nature Physics*, vol. 11, no. 4, pp. 291–293, 2015.