# Hybrid Analog-Digital Solution of Nonlinear Partial Differential Equations

Yipeng Huang
Columbia University
yipeng@cs.columbia.edu

Ning Guo
Columbia University
ng2364@columbia.edu

Mingoo Seok
Columbia University
mgseok@ee.columbia.edu

Yannis Tsividis
Columbia University
tsividis@ee.columbia.edu

Kyle Mandli
Columbia University
kyle.mandli@columbia.edu

Simha Sethumadhavan
Columbia University
simha@cs.columbia.edu

## ABSTRACT

We tackle the important problem class of solving nonlinear partial differential equations. While nonlinear PDEs are typically solved in high-performance supercomputers, they are increasingly used in graphics and embedded systems, where efficiency is important.

We use a hybrid analog-digital computer architecture to solve nonlinear PDEs that draws on the strengths of each model of computation and avoids their weaknesses. A weakness of digital methods for solving nonlinear PDEs is they may not converge unless a good initial guess is used to seed the solution. A weakness of analog is it cannot produce high accuracy results. In our hybrid method we seed the digital solver with a high-quality guess from the analog side.

With a physically prototyped analog accelerator, we use this hybrid analog-digital method to solve the two-dimensional viscous Burgers' equation —an important and representative PDE. For large grid sizes and nonlinear problem parameters, the hybrid method reduces the solution time by 5.7×, and reduces energy consumption by 11.6×, compared to a baseline solver running on a GPU.

## CCS CONCEPTS

• **Computer systems organization** → **Analog computers**; *Heterogeneous (hybrid) systems*; • **Hardware** → *Application specific integrated circuits*; • **Mathematics of computing** → **Nonlinear equations**; Partial differential equations;

## KEYWORDS

analog, accelerator, nonlinear, Newton's method


*MICRO-50, October 14–18, 2017, Cambridge, MA, USA*

ACM ISBN 978-1-4503-4952-9/17/10…$15.00
https://doi.org/10.1145/3123939.3124550

**ACM Reference format:**
Yipeng Huang, Ning Guo, Mingoo Seok, Yannis Tsividis, Kyle Mandli, and Simha Sethumadhavan. 2017. Hybrid Analog-Digital Solution of Nonlinear Partial Differential Equations. In *Proceedings of MICRO-50, Cambridge, MA, USA, October 14–18, 2017,* 14 pages.
https://doi.org/10.1145/3123939.3124550

## 1 INTRODUCTION

Emerging microscopic robots require the use of powerful mathematical models to simulate the physical world. Problems such as fluid dynamics and optimal control, once considered supercomputing workloads, are now needed in autonomous mobile robots where energy budgets are limited. These problems are phrased as nonlinear partial differential equations (PDEs). One unconventional computing approach is to use analog electronic circuits as differential equations solvers. Such an approach forgoes two fundamental abstractions in digital computing: the binary number representation of data, and the step-by-step operation of hardware and programs. By relaxing these abstractions which hold back performance and efficiency, the analog model provides a better fit for these challenging workloads. In this paper we explore the benefits of using a programmable analog accelerator as a domain specific accelerator for nonlinear PDEs.

The keys to benefiting from accelerator offloading are twofold: first, there must be a dominant kernel that consumes a significant portion of the application execution time. Second, there should be a simple way of refactoring existing programs to take advantage of new accelerators. A workload characterization of some engineering PDE solvers reveals they spend a large fraction of their runtime in solving systems of algebraic equations (Table 1). Thus, these programs can benefit from an accelerator for solving systems of algebraic equations. In fact, these core kernels are often offloaded to GPUs, and in our prior work, we examined an accelerator for solving *linear* systems of algebraic equations [22, 23]. In contrast, in this work we present an accelerator for solving *nonlinear* systems of algebraic equations, a more challenging problem class.

Compared to linear equations, nonlinear systems of equations are challenging for two reasons: first, to be able to find a solution, nonlinear numerical solvers need a good initial guess. This is more critical than in linear solvers not only to avoid redundant work but to even achieve convergence. As a

| Discipline | Problem description | Representative solver | Solving approach | Dominant kernel | Dominant kernel time |
|---|---|---|---|---|---|
| Fluid dynamics | 3D transonic transient laminar viscous flow | SPEC CPU2006 410.bwaves (test) | finite difference discretization with implicit time stepping on the compressible, viscous Navier-Stokes equations | Bi-CGstab | 76.7 + 11.7% |
| Magneto-hydrodynamics | 2D Hartmann problem | OpenFOAM | finite difference discretization on incompressible, viscous Navier-Stokes equation, coupled with Maxwell's equations | preconditioned conjugate gradients | 45.8% |
| Fluid dynamics | lid-driven cavity flow | OpenFOAM | finite volume discretization on incompressible, viscous Navier-Stokes equations | preconditioned conjugate gradients | 13.1% |
| Engineering mechanics | Cook's membrane | deal.II | finite element discretization with nonlinear spring forces | Solving Helmholtz PDE with preconditioned SOR and CG | 15.3% |

**Table 1: Function profile of nonlinear PDE solvers which would be the envisioned targets for analog acceleration. Linear and nonlinear algebra is the dominant kernel in all solvers. The equation solving proportion is higher for structured grids such as finite difference. Irregular memory accesses shift computation time away from equation solving for less structured grids such as finite volume and finite elements.**

rough analogy, if solving linear equations is like navigating an orderly city grid, solving nonlinear equations is like hiking in the mountains. In the latter case there is a higher chance of getting lost, and thus starting close to the solution spot is critical. Second, numerical solvers for nonlinear equations rely on a careful choice of the step sizes they take toward the solution. To further our analogy, this is as if nonlinear solvers need to frequently check the map, never traveling too far in any one direction, while linear solvers can speed to their solutions in just a few turns. These difficulties entail more work in solving nonlinear systems of equations.

Our proposed analog accelerator for solving nonlinear systems of equations has three major benefits: first, the analog units in the accelerator naturally implement nonlinear functions, reducing the amount of work compared to a digital accelerator. Second, the accelerator uses a different algorithm for solution finding, one that operates in continuously in time with no notion of step-by-step operation as required by digital hardware. In our analogy it allows the nonlinear solver to always know which direction to proceed, without pausing to check the map. Third, we use a method of initial solution guessing uniquely suited for the analog accelerator called homotopy continuation, which in effect maps an orderly city grid to the (nonlinear) wilderness, making it easier to find initial guesses.

Only using the analog accelerator is not without problems. While the strengths of the analog accelerator are its speed, its efficiency, and its ability to naturally support nonlinearity, it gives only approximate results and does not scale to large problem sizes. On the other hand, digital offload accelerators require lots of tuning on numerical parameters such as step sizes and initial guesses, but can give high precision results and handle large problem sizes. In this paper we combine the strengths of both approaches without complicating programming. We propose a program partitioning where the traditional, digital methods are used to break the nonlinear PDEs into subproblems that can be solved on an analog accelerator approximately. These analog approximate solutions are then seeded into the digital algorithm to obtain an accurate solution.

Using measurements from a system of physically prototyped analog accelerator chips, we show we can get approximate results (with lower accuracy—within about 5% of the fully accurate digital solution) nearly 100× faster; and obtain fully accurate, higher precision results with 5.7× speed improvements by using the analog results as good initial guesses for the digital solver (compared to a GPU). The extremely low power dissipation of analog circuits results in an energy consumption reduction of 11.7×. The lower accuracy results may be suited for graphics or motor control applications while the higher accuracy results can be used for accelerating innermost kernels of scientific computations.

The rest of this paper is organized as follows: Sections 2, 3 review scalar and vector nonlinear equations. We discuss the pitfalls of the Newton method for solving nonlinear equations; we show how a continuous model of computation avoids them. In Section 4 the theme of collaboration between analog and digital computers turns to the digital discretization of PDEs which are then solved in analog. Section 5 details the programming model, architecture, and microarchitecture of a prototype analog accelerator for solving nonlinear PDEs, and provides measured results in analog solution accuracy. Section 6 evaluates the merits of analog acceleration for nonlinear PDEs, including the use of larger accelerators, using the analog solution as a seed, compared against a GPU baseline. Section 7 discusses how our techniques generalize to other PDE problems and Section 8 discusses prior work in analog computing.

# 2 TUTORIAL: SCALAR NONLINEAR ROOT-FINDING

This section is a review of digital methods for solving nonlinear equations and a tutorial on doing the same using an analog accelerator. We highlight pitfalls of the digital method which are avoided in the analog computational model.

## 2.1 Digital classical and damped Newton's

Digital algorithms for nonlinear equations must have a good initial guess to the solution, or else it must spend a lot of time to find the right solutions. In order to understand this tradeoff, let's first review the problem statement.

Solving nonlinear equations entails finding a floating-point value $u$ that satisfies the nonlinear function $f(u) = 0$. The solution $u$ is called a *root* of $f$. For example, the equation

$$f(u) = u^3 - 1 = 0 \tag{1}$$

has one real-valued root $u = 1$ and two complex-valued roots.

To get these roots numerically, the Newton method starts with an initial guess $u_0$ and iterates through multiple guesses $u_p$ according to the recurrence relation:

$$u_{p+1} = u_p - \frac{f(u_p)}{f'(u_p)} = u_p - \frac{u_p^3 - 1}{3u_p^2}$$

A downside of Newton's method is it is sensitive to the initial guess of what the roots should be. When we plot on the imaginary plane the root to which Newton's method converges against the choice of the initial guesses, the resulting picture is fractal: the regions of the plot are intertwined in complex patterns, indicating a small change in the initial guess for Newton's method can lead to different conclusions [25]. This is because Newton's method updates its guess of the solution in discrete steps [37].

One way to reduce the classical Newton method's sensitivity to initial guesses is to increase the the computation time. We do this by using relaxed or damped steps, where the full step size is diminished to a fraction $h$ between 0 and 1:

$$u_{p+1} = u_p - h\frac{f(u_p)}{f'(u_p)}$$

By reducing the step size the guesses are more likely to stay in the same convergence basin. The pictures plotting the final solutions against initial conditions become less complex as the convergence basins grow in size and become contiguous [25]. In effect, the damped Newton method decreases the algorithm's sensitivity to initial conditions, at the cost of having to run the algorithm for more iterations. In practice it is difficult to choose the correct step size.

## 2.2 Analog continuous Newton's method

Now, we show how a continuous-time analog accelerator offers a more natural and reliable way to solve $f(u) = 0$ by avoiding the problem of finding a correct step size. We test the scheme on a prototype chip we will cover in detail in Section 5.

We take damped Newton's methods to the logical extreme and shrink the step size $h$ to infinitesimally small, and take infinitely many steps of the resulting *continuous* Newton's method, which should be minimally sensitive to the choice of the initial conditions [21, 29, 31, 34]. In fact, the continuous Newton method could be considered the natural way of solving nonlinear equations. It is stated concisely as an ordinary differential equation (ODE).

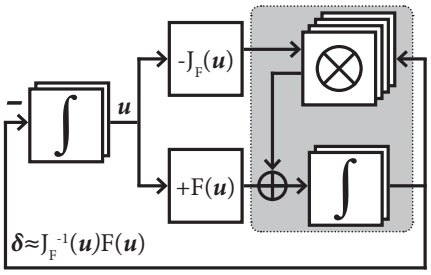


**Figure 1: Analog circuit for continuous Newton's method. Clockwise from the center left, the major analog function units: integrators for holding the present guess of $u$, a block for evaluating the Jacobian (the derivative $f'(u)$ in the scalar case), a block (shaded) for finding the Jacobian inverse (the quotient $f(u)/f'(u)$ in the scalar case) using gradient descent, and a block for evaluating the nonlinear function. Numbers are represented as analog current and voltage. Physically, integrators are capacitors. Digital-to-analog converters (DACs) generate constant values. Joining wires sums numbers by summing currents. The circuit values change continuously in time, with no clock cycles or steps.**

Digital computers cannot directly solve ODEs and instead approximate them using numerical integration. For example, the damped Newton method is an Euler's method approximation of the continuous Newton method ODE. More sophisticated Newton's method solvers use better numerical integration, but those improved algorithms quickly become complex and costly.

Analog accelerators on the other hand directly solve the continuous Newton method's ODE description. Let's walk through how this is done as it underpins the techniques used in the rest of this paper.

**Analog implementation:** Figure 1 shows an analog circuit that operates in continuous time, implementing the continuous Newton method. We use the integrators at the left side of the circuit to store the analog value of the real and imaginary parts of $u(t)$ as functions of time. The integrators take as their input the value $\frac{du}{dt}$, the rate of change of $u$ at any moment in time. $u(t)$ is then fed to analog hardware that multiplies and sums values to create the derivative $f'(u)$ and the function $f(u)$. Complex number multiplication is done by cross multiplying the real and imaginary parts appropriately.

Next, we must find the quotient between these values, $\frac{f(u)}{f'(u)}$. The quotient is calculated in analog hardware using negative feedback, using continuous gradient descent, a technique explored in detail in [22, 23].

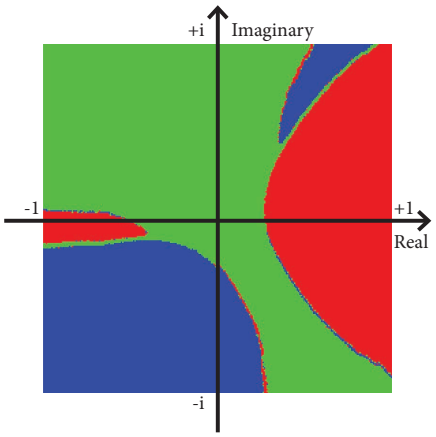


**Figure 2: The results of continuous Newton's method running on an analog accelerator prototype chip solving Equation 1. The colors encode which of the three cubic roots the chip returns, plotted on the imaginary plane indicating the initial conditions. Each of the $256 \times 256$ pixels is one run of the chip. The convergence basins are more contiguous compared to those in classical or damped Newton methods.**

The quotient is negated and fed to the inputs of the $u(t)$ integrators as $\frac{du}{dt}$, the rate of change of $u$. The values change continuously, with no notion of clock cycles or time steps, so the whole system is described as an ODE—the ODE for continuous Newton's. When the continuous Newton method converges, the inputs to the integrators tend toward zero, so the output of the integrators are steady, and at that point we can measure the output using analog-to-digital converters.

**Analog accelerator result:** Figure 2 shows the chip is able to return all of the three roots. Which root it converges to depends on the choice of the initial condition. The picture is simple and the convergence basins are contiguous compared to the pictures generated by classical and damped Newton's method [25], implying small changes in the initial condition are less likely to cause changes in the final solution. Using the analog accelerator it becomes easier to explore the effect of the initial guess.

## 3 MOTIVATION: NONLINEAR SYSTEMS OF EQUATIONS

In this section we discuss how analog and digital models of computing can work together, drawing on strengths and avoiding weaknesses of both. These ideas are important in understanding why hybrid analog-digital computing is useful for solving nonlinear PDEs.

### 3.1 Nonlinear systems: digital challenges

A weakness of the digital discrete-time model of computation becomes clear when we use the damped Newton method for solving nonlinear systems of equations. These problems have multiple unknown variables, unlike the previous section's root finding example which had one unknown. As a result the algorithm must find correct initial guesses for all of the unknowns, and solve a matrix equation in each step of the algorithm. These tasks are are inefficient when we are limited to using step-by-step digital computation.

**Finding the Jacobian and its inverse:** The Newton method requires finding the Jacobian matrix and solving a linear algebra problem involving the Jacobian. These tasks take the most time in nonlinear PDE solvers, as confirmed in the software profiles in Table 1.

First, let's discuss why the Jacobian matrix appears. Solving multidimensional nonlinear systems of equations entails finding $\vec{u}$, a $d$-dimensional vector satisfying $F(\vec{u}) = \vec{0}$. Just like in the scalar case, we need $F'(\vec{u_p})$, the derivative of $F$ with respect to $\vec{u}$ at the present guess $\vec{u_p}$. But unlike the scalar case, in multi-variable calculus this derivative is the Jacobian matrix $J_F(\vec{u})$, which is defined as:

$$F'(\vec{u}) = J_F(\vec{u}) =$$
$$\begin{bmatrix} \frac{\partial F_0}{\partial u_0}(u) & \frac{\partial F_0}{\partial u_1}(u) & \dots & \frac{\partial F_0}{\partial u_{d-1}}(u) \\ \frac{\partial F_1}{\partial u_0}(u) & \frac{\partial F_1}{\partial u_1}(u) & \dots & \frac{\partial F_1}{\partial u_{d-1}}(u) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_{d-1}}{\partial u_0}(u) & \frac{\partial F_{d-1}}{\partial u_1}(u) & \dots & \frac{\partial F_{d-1}}{\partial u_{d-1}}(u) \end{bmatrix}$$

Each row of the Jacobian corresponds to each element of $F(\vec{u})$, while each column differentiates $F(\vec{u})$ against each component of $\vec{u}$.

Next, let's discuss why linear algebra is involved. In the scalar example, we could simply find the quotient between the function and the derivative by doing scalar division. Now, the derivative is a matrix, and matrix division is not defined; so instead of doing division we multiply by the Jacobian matrix's inverse. The Newton method is then:

$$\vec{u_{p+1}} = \vec{u_p} - \vec{\delta_p}$$
$$\text{where,} \vec{\delta_p} = J_F^{-1}(\vec{u_p})F(\vec{u_p})$$

In practice we find the unknown $\vec{\delta_p}$ from the known $J_F(\vec{u_p})$ and $F(\vec{u_p})$ by solving the linear system of equations

$$J_F(\vec{u_p})\vec{\delta_p} = F(\vec{u_p})$$

So in each step of Newton's method we have to solve a linear algebra problem, and these subroutines becomes costly as problem sizes grow. Accelerating these subroutines with approximation techniques or dedicated hardware would be one way to speed up the overall algorithm.

**Uncertainty in the number of solutions and the effect of initial conditions:** Another challenge in solving nonlinear systems of equations in digital computers is it's difficult to know if any solutions, or how many solutions, there should be. Incorrect guesses at the beginning of the algorithm may prevent us from finding the right solutions.

It's difficult to visualize where the roots are located for nonlinear systems of equations. The problem asks us to find intersections of nonlinear surfaces that could have arbitrary shapes. This is in contrast to the simpler problem of finding the root of a scalar nonlinear function, which we can easily

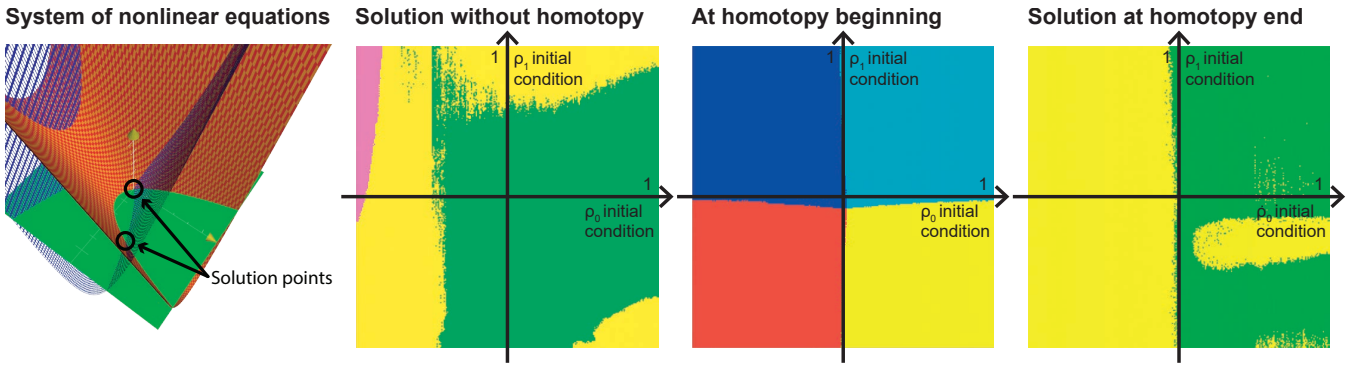


**Figure 3: Far left: Visualization of Equation 2. The two equations are surfaces (blue mesh and red checkerboard) formed by parabolas swept along straight lines. The root finding problem entails finding where two surfaces intersect at the $z = 0$ (solid green) plane. The RHS constants in the equations shift the surfaces up and down, so there can zero or several such solutions. Center left: Continuous Newton's without homotopy. Colors indicate the roots found by the chip, plotted against the initial conditions. Two solutions (green and yellow) are roots of Equation 2. The pink region is a set of initial conditions where Newton's method returns a wrong result. Clearly, the initial conditions strongly impacts the Newton method result. Center right: The initial state for a homotopy process. The chip settles on the four roots $(\rho_0, \rho_1) = (\pm 1, \pm 1)$ of Equation 3. The chip then solves an ODE to smoothly guide this initial state to the final state. Far right: Final result of the homotopy method. The chip returns two roots for Equation 2. Compared to naive Newton's method, all choices of initial conditions in the homotopy method lead to one correct solution or another.**

visualize in a 2D plane, showing the relationship between the nonlinear function and the function's unknown parameter. With that picture it was straightforward to locate the solutions for scalar problems.

As a concrete example, let's solve a coupled system of equations:

$$\begin{cases} \rho_0^2 + \rho_0 + \rho_1 = \text{RHS}_0 \\ \rho_1^2 + \rho_1 - \rho_0 = \text{RHS}_1 \end{cases} \tag{2}$$

This type of coupled system of equations may arise from solving a one-dimensional semilinear PDE problem on two grid points. The nonlinear term where the variables are squared indicate for example a reaction process.

We can visualize this coupled system of equations in 3D space shown in the leftmost panel of Figure 3, which shows that depending on the constant RHS coefficients, there may be 0, 1, 2, 4, or infinitely many solutions. Whether the Newton method converges to one of these solutions, and which one it ends up at, depends on the initial conditions to the algorithm. Wrong choices would make the algorithm incorrect or inefficient.

## 3.2 Nonlinear systems: analog homotopy

A strength of the analog continuous-time model of computation is we can naturally evaluate the nonlinear function and the Jacobian matrix by multiplying and summing analog signals. Then we can solve the Jacobian matrix equation and do the Newton method faster and more efficiently than in digital. We do so using continuous gradient descent and continuous Newton's method, which are continuous-time algorithms with no counterpart in digital computing.

To further illustrate the advantages of the analog computational model, here we try another continuous algorithm, homotopy continuation, which makes it easier to pick initial conditions for solving nonlinear systems of equations [3, 14, 30, 32].

In homotopy methods, we smoothly connect a simple problem with obvious initial conditions to the hard one we would like to solve. We would devise a simple root-finding problem $S(\vec{\rho}) = 0$, representing a trivial system of equations:

$$S(\vec{\rho}) = \begin{cases} \rho_0^2 - 1 = 0 \\ \rho_1^2 - 1 = 0 \end{cases} \tag{3}$$

We know that this system's four roots are $(\rho_0, \rho_1) = (\pm 1, \pm 1)$. Then, we denote a harder nonlinear system, such as Equation 2, as $H(\vec{\rho}) = 0$. The example hard nonlinear system has as many as four non-degenerate roots, but we do not know what initial conditions to set to get those solutions.

With the hard system and simple system in hand, we construct a joint system characterized by a homotopy parameter $\lambda$ that controls the system's degree of nonlinearity and solution difficulty:

$$(1 - \lambda)S(\vec{\rho}) + \lambda H(\vec{\rho}) = 0$$

We would start Newton's method with $\lambda = 0$ and $\vec{\rho}$ set to be one of the known roots, satisfying the simple system $S(\vec{\rho}) = 0$. Then, we smoothly guide the simple system to the hard system by incrementing $\lambda$ until $\lambda = 1$. At each moment while incrementing $\lambda$, we perform a Newton method inner loop so $\vec{\rho}$ remains the correct root for the combined system.

The end result is we have smoothly mapped each of the unknown roots of the hard system to the known roots of the

simple system. By exploring the roots of the simple system we explore the roots of the difficult problem. Homotopy methods are an appealing extension of Newton's methods, except for the fact the homotopy continuation is again an ODE in disguise [9, 10, 29, 33], and therefore costly to approximate in a digital computer.

We can instead solve this ODE on our analog accelerator prototype chip. The results are shown in Figure 3. The results show that analog accelerators can perform more advanced global Newton methods, in addition to the continuous Newton method, adding to our repertoire of continuous algorithms for analog accelerators.

### 3.3 Approximate analog & precise digital

An ideal solving system should use analog methods where digital ones are weak, while keeping all the convenient aspects of digital computing. The digital methods allow use of binary floating point numbers, which have higher precision and accuracy, but encounter problems in selecting a Newton step size and an initial condition. The analog methods have more reliability, but the computational results have low accuracy and precision. In prior work researchers have tried various ways to combine analog and digital computing. We review some ways below. This work extends and is distinct from those techniques.

Analog approximate solutions can be used to seed high-precision Newton solvers, by providing a good initial guess from which the Newton method immediately enters the region of quadratic convergence. For example, in prior work where analog computers served as direct physical models, Cowan *et al.* used an analog co-processor to solve a periodic nonlinear ODE directly, and the sampled low-precision analog trajectory assists a high-precision digital solver, helping it converge [11, 12]. This work achieves a similar effect, with an important distinction our analog accelerator performs an abstracted continuous algorithm instead of solving a physical model directly. Our approach more readily supports existing solvers that invoke solving nonlinear systems of equations as an underlying kernel.

In digital approximation approaches, numerical methods can first use single-precision floating point numbers with cheaper operations, allowing longer vectors to reside in local caches, before finishing off with double precision [4, 5, 8, 28]. The analog acceleration techniques in this paper can extend those methods due to its fundamental energy efficiency in the low bit precision regime [13].

This paper so far considers analog accelerator support for nonlinear algebra. The continuous-time analog model of computation supports the uniquely more reliable continuous Newton's and homotopy methods, which are less sensitive to choices of step size and initial conditions. Such an approach has not been done in prior analog work, and is incompatible with conventional discrete-time digital accelerators.

In the next section we combine the strengths of analog and digital computing another way. We discuss how a digital computer can break down large problems to make use of an analog accelerator, which is fast and efficient for limited problem sizes. We will return to using analog approximations to help high-precision digital in Section 6.

| Reynolds number | Mach number | Viscosity | Effect of diffusion | Dominant PDE character | Nonlinearity |
|---|---|---|---|---|---|
| Large | High | Low | Small | First-order, advective (hyperbolic PDE) | Quasilinear |
| Small | Low | High | Large | Second-order, diffusive (parabolic PDE) | Semilinear |

**Table 2: Effect of Reynolds number on Burgers' and Navier-Stokes equations. Larger Reynolds numbers result in more nonlinear and difficult problems.**

## 4 NONLINEAR PDES & DISCRETIZATION

PDEs describe the relationship of variables in terms of their derivatives, and are thus an important model for the natural world, which is also described using real numbers in continuous space. In this section we convert PDEs into the systems of nonlinear equations that have been the focus of this paper thus far.

### 4.1 The viscous Burgers' equation

In our effort to benchmark our analog accelerator as a nonlinear systems of equations solver, we must first choose an illustrative source of a nonlinear equation. Specifically the rest of this paper focuses on the viscous Burgers' equation, a nonlinear PDE. The Burgers' equation is the subset which asserts momentum is conserved in the Navier-Stokes equations for modeling fluids [16].

The viscous Burgers' equation has the form:

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} - \frac{1}{\mathrm{Re}}\nabla^2 \vec{u} = \mathrm{RHS} \quad (4)$$

$$\begin{cases} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \frac{1}{\mathrm{Re}}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = \mathrm{RHS}_0 \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \frac{1}{\mathrm{Re}}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) = \mathrm{RHS}_1 \end{cases} \quad (5)$$

The center of attention is on the pair of vector-valued variables $\vec{u} = (u, v)$, where $u$ represents the x-velocity field and $v$ represents the y-velocity field of a fluid in 2-dimensional space. The PDE is nonlinear because the partial derivatives of $u$ and $v$ have coefficients depending on $u$ and $v$ themselves.

We are drawn to this example problem in part because only one parameter needs to be selected; examples with more parameters may obscure the evaluation. Furthermore, different choices of that parameter causes the viscous Burgers' equation to behave similarly to a variety of PDEs, allowing us to generalize our techniques in this paper to other classes of PDEs, which we discuss in Section 7. That parameter is the Reynolds number, Re, a dimensionless coefficient which controls the behavior of the the Burgers' equation and the incompressible Navier-Stokes equations, as shown in Table 2.

Higher Reynolds numbers result in more nonlinear systems of equations, and increases the difficulty of solving this PDE.

### 4.2 Space discretization

With an example nonlinear PDE in hand, in the next two subsections we summarize how nonlinear PDEs are converted to nonlinear systems of equations. We advocate for doing these discretization steps in digital, where there are a wide variety of advanced techniques. The analog accelerator supports the variety of techniques by focusing on the inner kernel of solving the system of equations.

First we have to handle the fact the PDEs describe continuous fields in space. We do space discretization to convert the continuous fields into a grid of node variables. This is necessary because digital machines and our analog accelerator represent variables as scalars, which capture a value at one place in space as a function of time. In this paper we use a central finite difference method for simplicity. Analog accelerators can nonetheless help solve the nonlinear systems of equations generated by some other space discretization schemes, which we discuss in Section 7.

Applying space discretization to a PDE results in a system of ODEs, which is discrete in space but continuous in time. We handle the equations' time evolution next.

### 4.3 Time stepping

With the PDE spatially discretized into a system of ODEs, we can tackle the time derivative in several ways.

The first approach is to solve the ODEs directly in an analog computer, which then becomes the "method of lines" approach used in earlier hybrid computers [6, 15, 24, 26, 27, 36, 38]. Applying those techniques to support existing modern PDE solvers would require some way to generate and measure analog waveforms at both high precision and frequency, which are difficult to have simultaneously in DACs and ADCs.

So instead of solving the ODEs directly using the analog accelerator (as was typical in previous hybrid computing work), we will let the digital host do time stepping as well as space discretization. This approach allows the analog accelerator to work inside modern PDE solvers where these types of discretization are standard practice.

In this paper we use Crank-Nicolson, an implicit method which offers second-order accuracy, to decompose the 2D Burgers' equation into a nonlinear system of equations. Section 7 discusses how our approach generalizes to other time stepping schemes.

### 4.4 Viscous Burgers' PDE discretization

Using the techniques in the past two sections, we take Equation 4, the 2D viscous Burgers' equation, and apply second-order central finite difference and second-order Crank-Nicolson time stepping. Then, we make isotropic assumptions about the relative size of the space and time grid points to simplify the problem. We choose values for $\Delta t$, $\Delta x$, and $\Delta y$ so these coefficients are eliminated, for the sake of clarity.

The resulting stencil consists of a nonlinear system of equations and its Jacobian, which can be found in the literature [16, pg.172]. These two sets of mathematical expressions are what we need to program into the analog accelerator circuit shown in Figure 1. Different types of PDEs and discretization schemes will result in different nonlinear systems of equations and their Jacobians, which similarly can be set up inside the analog accelerator.

## 5 ANALOG ACCELERATOR SOLUTION OF NONLINEAR PDES

So far, we've shown how an analog accelerator can solve nonlinear systems of equations, using the continuous Newton method. We have also shown how solving nonlinear PDEs is converted into solving nonlinear systems of equations.

Now, we bring these ideas together: we discuss how the 2D viscous Burgers' equation is solved in a prototype analog accelerator. We will show the programming model and architecture interface of a reconfigurable analog accelerator. We test the approach using a physically prototyped analog accelerator chip, for a small $2 \times 2$ grid size due to prototype size constraints, and present measured accuracy results.

### 5.1 Programming and data interface

The analog accelerator has a digital interface for configuration, transmitting data, and programming.

A digital host processor prepares the analog accelerator for equation solving by configuring the chip so the analog signals in the chip represent the nonlinear system of equations $F(\vec{u})$ and the Jacobian matrix $J_F(\vec{u})$. Then, these mathematical expressions are connected according to Figure 1 so the signals evolve according to the continuous Newton method. This way of setting up the analog accelerator is distinct from prior work in analog computing where differential equations had to be directly mapped and programmed to analog computers.

The data transmission costs for the analog accelerator would be the same as a digital accelerator device such as a GPU or a node-attached FPGA. This is because the configuration of the analog accelerator remains the same when solving for different instances of the same kind of PDE. Once the connectivity between analog components is set, the digital host sends digital codes for equation constants and coefficients to be set by DACs. The analog accelerator solves the equation, and the digital host retrieves values measured by the ADCs. Only new problem parameters and results need to be transmitted between analog accelerator runs.

We program the accelerator using object-oriented C++, a style of programming that improves code reuse and minimizes errors when programming the analog accelerator. A code sample is given in Figure 4. Each analog subcomponent can be instantiated on the analog accelerator and tested individually. The programming scheme allows us to apply software and digital hardware engineering techniques to analog components, including unit testing, randomized validation, and incremental bringup of larger systems.

```
/*initialize and calibrate analog accelerator fabric*/
Fabric * fabric = new Fabric();
fabric->calibrate();

/*create top-level data structure representing 2D Burgers' equation analog node variables*/
/*upon instantiation, node variables get allocation of analog hardware to implement the needed analog datapath*/
cells = new NewtonTile[8] {
    /*cell variables take as parameters an initial condition, various Burgers' equation coefficients & settings*/
    NewtonTile ( fabric->chips[0].tiles[0], 1.0, 128, 128, 5.0, 0.0, 0.0, true, true, true ),
    ...
};

/*connect exposed analog interfaces together to form continuous Newton method circuit for 2D Burgers' equation*/
parallelConnect ( &cells[0], &cells[1] );
...

/*additional connections export variables between analog accelerator chips, off chip, and into ADCs*/
Fabric::Chip::Connection ( cells[0].u_out_chip, fabric->chips[0].tiles[0].slices[0].chipOutput->in0 ).setConn();
...

/*change analog parameters such as initial conditions, coefficients, and constants for different problems*/
for (unsigned char cellIndx = 0; cellIndx < 8; cellIndx++) {
    cells[cellIndx].setUCoeffParallel ( coeff_parallel[cellIndx] );
    cells[cellIndx].setRHS ( rhs[cellIndx] );
    cells[cellIndx].setDynamicRange ( dynamic_range );
}

/*underlying above high-level calls, analog accelerator is changing the analog parameters of subcomponents:*/
slice.muls[0].setGain ( 1.0 / dynamic_range ); // coefficients realized by multipliers
slice.dac->setConstant( jaco_coeff );          // constant biases provided by digital-to-analog converters
slice.integrator->setInitial(initial);         // integrator initial conditions for Newton initial guesses

/*commit the analog accelerator config and parameters; release the integrators to start continuous Newton's*/
fabric->cfgCommit();
fabric->execStart();

/*measure final analog value using ADCs, restore integrators and prepare for next set of parameters*/
newton_u[0][0] = fabric->chips[0].tiles[3].slices[3].chipOutput->analogAvg(REPS);
...
fabric->execStop();

/*destroying objects representing analog variables frees the analog hardware for other calculations*/
delete[] cells;
```

**Figure 4: Analog accelerator object-oriented C++ code sample.**

| Component | Nonlinear function | Jacobian matrix | Quotient feedback loop | Newton method feedback loop |
|---|---|---|---|---|
| integrator | 0 | 0 | 1 | 1 |
| fanout | 2 | 0 | 3 | 3 |
| multiplier | 4 | 3 | 1 | 0 |
| DAC | 3 | 1 | 0 | 0 |
| tile input | 4 | 4 | 0 | 0 |
| tile output | 4 | 0 | 4 | 3 |
| total area ($mm^2$) | .30 | .17 | .14 | .09 |
| total power ($\mu W$) | 284 | 152 | 188 | 139 |

**Table 3: Summary of analog chip component use for each PDE variable with area and power model from [18, 19, 22, 23].**

## 5.2 Board and chip hardware mapping

We use a circuit board with two analog accelerator chips to solve the 2D Burgers' equation. One analog accelerator chip stores and computes on $\vec{u}$, the x-velocity field, and the other does the same for $\vec{v}$, the y-velocity field. The interaction between these two fields is sparse, so they can be connected via circuit board-level connections. The characteristic analog bandwidth of the accelerator chips is kept low enough so the propagation time for data and control signals across board-level connections does not matter.

As shown in Figure 5, each analog accelerator chip contains four identical tiles. In this example each tile is in charge of one scalar element in $\vec{u}$ or $\vec{v}$. Within each tile, the analog function units are connected together to form the nonlinear equations and the Jacobian matrix. In the Burgers' equation the expressions are polynomials, which can be built using multipliers and summers. Table 3 shows the hardware needed to implement each mathematical component.

## 5.3 Dynamic range of values and scaling

The full dynamic range of the PDE problem variables must scale down to fit in the dynamic range of the analog hardware. The details of how to scale depend on the nonlinear PDE's type of nonlinear function. In the Burgers' equation, the nonlinear function is a quadratic polynomial. So, if the variables

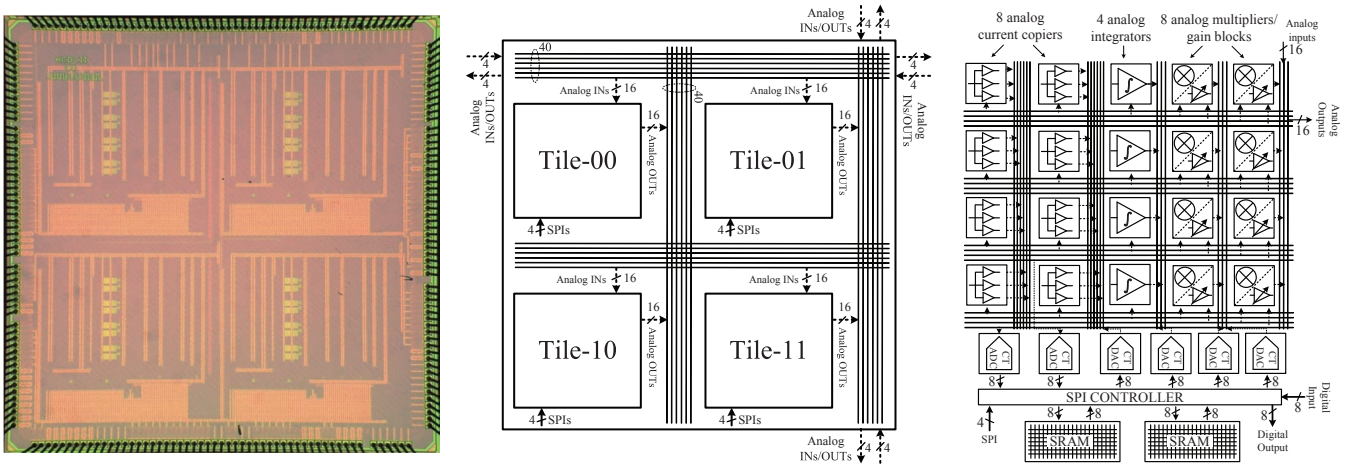


**Figure 5: Left: Microphotograph of an analog accelerator, measuring $3.7mm \times 3.9mm$, fabricated in a TSMC 65nm process. Center: Architecture diagram of an analog accelerator designed to test scalable multi-chip integration and calibration of large analog accelerators. The chip contains four tiles, each an instance of the microarchitecture presented in [18, 19, 22, 23]. Connectivity between tiles and between chips is tree-like with sparse connectivity, matching the neighbor-to-neighbor connection pattern for PDEs. The orientation of the analog inputs and outputs is designed for multiple-chip board-level integration. Right: Diagram of an analog accelerator tile containing 4 integrators. Other components include multipliers, current mirrors, ADCs, and DACs. A programmable crossbar enables all-to-all connectivity within each tile, matching the connection patterns needed to realize a variety of polynomial functions and Jacobians.**

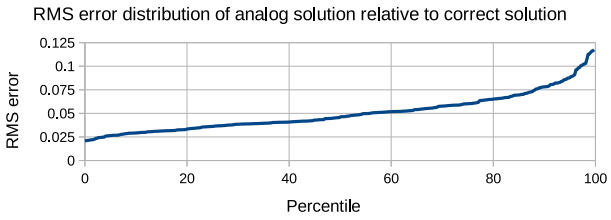


**Figure 6: Distribution of analog solution error for 400 randomly generated problems.**

$\vec{u}$ and $\vec{v}$ are scaled by $\frac{1}{s}$, the system of equations should be scaled by $\frac{1}{s^2}$. To make sure the terms in the nonlinear polynomial stay in correct proportion, any coefficients on linear terms of $\vec{u}$ and $\vec{v}$ should also be scaled by $\frac{1}{s}$. Scaling can be applied to nonlinear PDEs with polynomial nonlinearities, which excludes some PDEs with transcendental nonlinearities of theoretical interest, but fortunately includes many physically meaningful PDEs.

### 5.4 Analog accelerator accuracy results

We use the analog accelerator to solve 400 sets of nonlinear equations that would be generated from a 2D Burgers' equation stencil. The constants in the nonlinear system of equations are randomly chosen between a dynamic range of -3.0 and 3.0. The constants and the solution vector are then scaled to fit in the analog accelerator's dynamic range.

We define the error between the analog solution and the digital solution as:

$$\sqrt{\frac{\sum_N (u_a - u_d)^2}{N}} \tag{6}$$

Where $N$ is the number of elements in the analog and digital solutions. Figure 6 shows the distribution of the errors for the 400 trials. The total RMS error for the 400 trials is 5.38%.

The limited accuracy of the analog accelerator is due to several reasons. One is limited ADC resolution. Another is process variation and transistor mismatch, which we control by calibrating all components on the analog datapath, though the calibration precision is itself limited by DAC precision.

The analog accelerator solutions can be used where lower accuracy results are useful, or as a seed for a digital solver.

## 6 DESIGN SPACE EXPLORATION OF SCALED-UP ACCELERATORS

So far, the case studies of Sections 2.2, 3.2 and the $2 \times 2$ 2D Burgers' equations showcase the unique properties of the continuous Newton method and validate its implementation on an analog accelerator. In this section, we quantify the benefit of larger scale analog accelerators in terms of performance and efficiency improvements. Then, we show the approximate analog solution can provide a better initial guess to greatly speed up precise digital solvers.

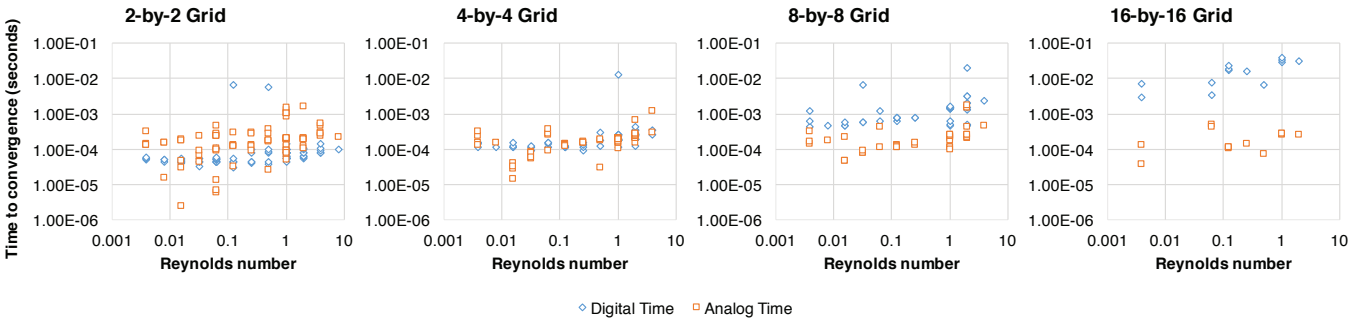


**Figure 7: Time to convergence for digital and analog solvers.**

| 2D Burgers' solver size | Chip area ($mm^2$) | Power use (mW) |
|---|---|---|
| $1 \times 1$ | 1.38 | 1.53 |
| $2 \times 2$ | 5.50 | 6.10 |
| $4 \times 4$ | 22.02 | 24.42 |
| $8 \times 8$ | 88.06 | 97.66 |
| $16 \times 16$ | 352.36 | 390.66 |

**Table 4: Area and power model for scaled-up analog accelerators. Power consumption is peak power; as the continuous Newton method approaches convergence the circuit activity and power consumption decreases. While the analog chip area is large, power consumption is extremely low.**

## 6.1 Performance vs. accelerator size

As the problem size increases, a digital Newton method solver takes more iterations to converge and give a solution. On the other hand, a scaled-up analog accelerator takes a relatively constant amount of time to converge, as long as the scaled-up design is feasible. Area constraints on the analog accelerator limit us to solving grid sizes as large as $16 \times 16$, corresponding to a large nonlinear system of equations with a $512 \times 512$ sparse Jacobian matrix.

**Problem setup:** We solve randomly generated 2D Burgers' equations with grid sizes of $2 \times 2$, $4 \times 4$, $8 \times 8$, and $16 \times 16$. The initial and boundary conditions for the problems are again randomly chosen within the dynamic range of the analog accelerator.

First, we get the correct solution using a golden-model Newton method solver taking small steps to generate an accurate solution. The golden-model solution is certified to satisfy the nonlinear system of equations.

Then, we use both a baseline digital solver taking moderate step sizes and a simulated experimental analog accelerator to solve the same problem. We compare the baseline digital and experimental analog solvers at equal, relatively low, accuracy. Both the baseline digital solver and the simulated analog solver are stopped when their error metric defined in Equation 6 reaches 5.38%, the value we measured from the analog accelerator chip.

**The baseline digital solver** is a parallelized damped Newton solver, implemented as a vectorized, 16-threaded OpenMP program running on two Intel(R) Xeon(R) X5550 CPUs running at 2.67GHz. The digital solver initially uses a damping parameter of 1.0. If the solution does not converge, it reduces the damping parameter by half until convergence is possible, at the cost of a long time-to-convergence. We give the digital solver the advantage counting only the time spent using the correct damping parameter, even though in practice this damping parameter is found via trial-and-error.

**The simulated scaled-up analog accelerator** models the variables in the analog accelerator as it solves the nonlinear problem. The model is built on the Odeint ODE solver library [2]. The time it takes for the continuous Newton ODE to reach a stable value corresponds to the reaction time of the analog circuit, which is in turn the solution time for the analog accelerator. The predicted solution time of the $2 \times 2$ analog accelerator is normalized to match the measured solution time of the physical analog accelerator. With this setup we can model analog accelerators larger than the one we physically prototyped.

**The dimensions and power consumption of the analog accelerator** are extrapolated for larger problem sizes. Table 4 shows that an analog accelerator for $16 \times 16$ problems is roughly the same size as CPU dies, while power density is about $400\times$ lower. Evidently, area costs constrain the problem sizes analog accelerators can solve directly. Fortunately, analog accelerators have unique strengths in extremely low power density and fault-tolerance, thereby avoiding constraints on digital die sizes such as heat and yield [35]. These unique strengths of analog accelerators may permit die sizes and stacking techniques otherwise impossible in digital designs. For now we limit ourselves to $16 \times 16$ problems.

We assume the analog accelerator generates a result with the same error metric as measured in the physical prototype chip, which would rely on the demonstrated calibration techniques to control for process variation and mismatches. The additional noise sources in scaled-up chips would have to be controlled at the expense of greater power dissipation.

Figure 7 shows the solution times for digital and analog accelerators. The axes, both in logarithmic scale, are the solution time in seconds plotted against the choice of Reynolds number for the problem.

Higher Reynolds number problems are more difficult to solve in both analog and digital. At high Reynolds numbers,

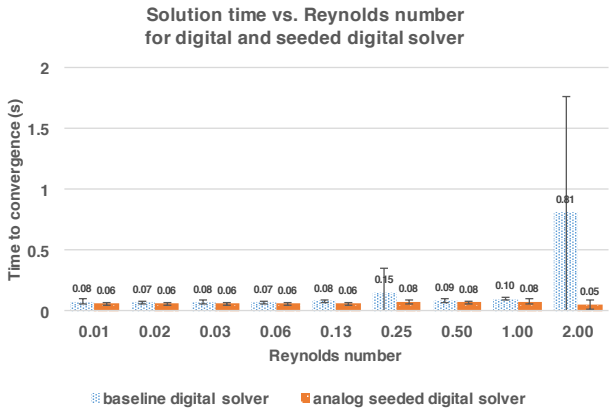


**Figure 8: Time to convergence for digital and seeded digital solvers to double-precision floating point epsilon precision.**

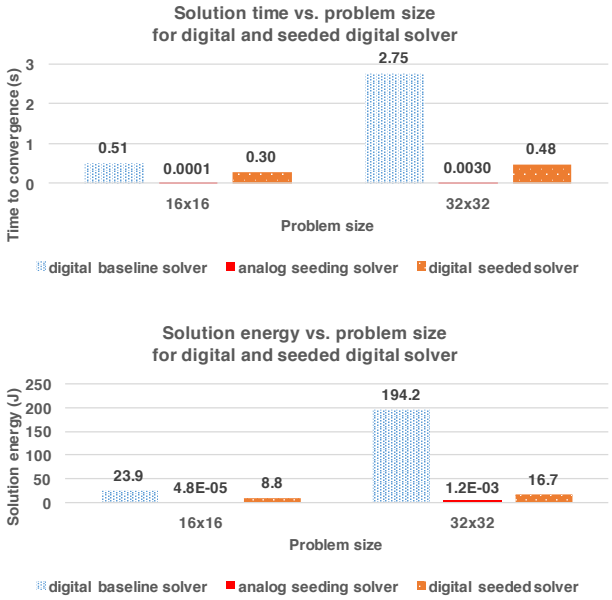


**Figure 9: Time and energy for solution for digital and seeded digital solvers running on a GPU.**

the PDE becomes more nonlinear and hyperbolic in character. The elements on the diagonal of the Jacobian diminish with higher Reynolds numbers, increasing the chance the Jacobian becomes singular in the process of solving the equation. In these situations, the baseline digital solver would then have to use many smaller-sized steps to get a solution. The data points become more sparse as the Reynolds numbers and problem sizes increase because fewer of the randomly generated problems have a correct solution. Even higher Reynolds number problems exist and have solutions, but would need different choices to be made during PDE discretization. We are limited by the spatial grid size and time step size we chose in our discretization scheme.

Looking across the different problem sizes, we see that the $4 \times 4$ problem has the analog and digital solving in roughly the same time. The digital solution time increases with each quadrupling of the problem size, while the analog accelerator solution time remains the same. The data show the $16 \times 16$ analog accelerator for solving nonlinear systems of equations may have $100\times$ faster solution time compared to a purely digital approach, and at much lower power dissipation.

## 6.2 Analog approx. as digital initial guess

We take the findings from the previous experiment and use the largest modeled analog accelerator design, capable of approximately solving $16 \times 16$ 2D Burgers' equations. We consider the benefits using such a chip to seed a digital solver that solves large problems at high precision.

For a given problem, the analog continuous Newton solver more reliably finds a solution, as discussed in Sections 2.2, 3.2, and does so in negligible time compared to the digital solver for the same accuracy, according to Figure 7. The analog solution is set as the initial condition for a seeded digital solver, which is then immediately in the quadratic convergence region for the Newton method. The digital solver carries on and terminates when the error metric is the smallest value representable in double-precision floating point numbers.

Figure 9 shows the solution time of a baseline digital solver compared to a seeded digital solver which benefits from the low-precision solution of an analog accelerator. The average solution time over 16 trials for both is plotted against various choices of Reynolds number for the problem, which influences the nonlinearity of the problem. The error bars represent the standard deviation of the solution times.

In relatively easier problems with low Reynolds number, the analog solver saves the digital solver a few steps. As the Reynolds number approaches 2.0, the baseline digital solver running the damped Newton method is forced to take smaller steps, causing the algorithm to run longer with greater variance in the solution time. On the other hand the analog seed saves the digital solver from having to use damped steps, greatly decreasing the digital solver's solution time.

## 6.3 Scaling to larger problems on GPUs

We now consider yet larger scale problems that potentially are solved using GPUs, and estimate how much energy is saved when an analog accelerator assists a GPU. The problem setup here is the 2D Burgers' equation with $\mathrm{Re} = 2.0$, at which point Newton's method may have poor convergence.

A common approach for solving larger nonlinear systems of equations is to offload the linear algebra inner loop of each Newton step to a GPU. For our **baseline digital solver** we offload work to a QR factorization solver, provided in the Nvidia cuSolver GPU sparse linear algebra library, running on an Nvidia GTX 1070 GPU. First, we certify the problem sizes are large enough to fully exercise the parallelism offered

by the GPU. For the $16 \times 16$ 2D Burgers' equation, the GPU program profiler nvProf reports the top three subroutines, accounting for 80% of the GPU runtime, use on average 90% of the GPU multiprocessors. When the problem size increases to $32 \times 32$, resulting in a $2048 \times 2048$ sparse Jacobian, the average multiprocessor activity increases to 95%.

The **analog seeding solver** needs a way to divide and conquer the larger systems of nonlinear equations, as our analog accelerator model is limited to solving $16 \times 16$ problems due to area constraints. We use red-black nonlinear Gauss-Seidel to split the $32 \times 32$ problems to fit. The Gauss-Seidel algorithm is in its most basic form an iterative algorithm for linear algebra, but it can also be used here for nonlinear problems [17, pg.291]. Digital architectures use the same decomposition and parallelization techniques to make subproblems fit in CPU caches or split work among nodes [20, pg.I-9], so the penalty of decomposition is not unique to the analog accelerator approach. The analog accelerator solves subproblems generated by nonlinear Gauss-Seidel several times until the Gauss-Seidel loop converges, and that solution is fed to the GPU as the initial condition.

Figure 9 shows seeding the GPU decreases the solution time for $32 \times 32$ Burgers' equations by 5.7×, and the energy by 11.6×. Not accounting for transfer costs between the analog accelerator, GPU, and CPU, the time and energy spent in the analog hardware is negligible compared to that spent in the digital solvers. Time and energy saved in these iterations would be significant, as Newton iterations are the innermost and dominant kernel in PDE solvers such as those in Table 1.

## 7 EXTENSIONS FOR OTHER PDES

Now we discuss whether our techniques can be extended to other varieties of nonlinear PDEs and solving methods. In this evaluation we have been focusing on a canonical nonlinear PDE, the quasilinear viscous Burgers' equation, defined on a two-dimensional grid. Our proposed way of using analog acceleration can be extended to other problems depending on the PDE properties and solver choices.

**Nonlinear PDE class:** We demonstrate solving a quasilinear PDE, which is a superset of semilinear PDEs and is generally more difficult to solve. Beyond quasilinear PDEs are fully-nonlinear PDEs, which permit the partial differential operators themselves be part of nonlinear functions. Fully-nonlinear PDEs are not generally solved using space and time discretization, so they are outside the scope of our investigation.

**Type of nonlinearity:** We demonstrate solving the Burgers' equation, which has a polynomial function as its source of nonlinearity. These polynomial functions can be calculated using multipliers and summers in the analog accelerator. Occasionally, nonlinear PDEs have transcendental nonlinear functions such as $e^u$ and $\sin(u)$. These transcendental equations would require analog nonlinear function generators. Transcendental nonlinear functions cause problems for analog accelerators because there is no clear way to scale problem variables to fit in the analog accelerator dynamic range.

**Dimensionality:** We demonstrate solving a two dimensional problem, which is more difficult to solve than one-dimensional ones. But most physical models are done in at least three-dimensional space. When multiple interacting physical laws appear in the model, the additional state variables can be thought as adding yet more dimensions to the problem. Solving higher-dimensional problems with analog acceleration would increase area consumption, and make the chip- and board-level routing of analog signals complicated. We note, however, all practical PDE solvers decouple the problem dimensions and solve the problem in one or two dimensions at a time, permitting the use of analog acceleration.

**Space discretization scheme:** There are many ways to do space discretization, which vary depending on whether the grid is regularly spaced, and on what the node variables represent. We solve the Burgers' equation using central finite difference, which features second-order accuracy. Higher-order finite difference schemes are more accurate and efficient, at the cost of having larger stencils, thereby requiring a larger accelerator. More advanced discretization schemes such as finite volume and finite elements are important in fluid and solid mechanics solvers. Those methods use unstructured grids, which on digital computers shift the bottleneck away from solving systems of equations and into generating the stencil. Analog accelerators offer only fixed stencils unless they are reconfigured frequently, so they would work poorly with unstructured grids.

**Time stepping scheme:** In this paper we use Crank-Nicolson time stepping, an implicit time stepping scheme with second-order accuracy suitable for time-dependent parabolic equations such as the viscous Burgers' equation. Higher-order time stepping methods allow larger step sizes to be taken, at the cost of putting more unknown variables at play in the systems of equations, thereby requiring a larger accelerator. Beyond parabolic PDEs, time-dependent PDEs also include hyperbolic PDEs. Those are often solved using explicit time-stepping, where there is no need to solve systems of algebraic equations and are therefore outside the scope of this paper.

## 8 RELATED WORK

Analog computers were used in early computing history as direct models for physical systems: one would describe an interesting physical system as a differential equation, then solve that equation on an analog computer [15, 24, 26, 27, 36]. Our research group revisited those techniques and revived analog computing in modern integrated circuits to solve various differential equations [11, 12, 18, 19, 22, 23]; the efforts are summarized in Table 5. Unfortunately, directly mapping differential equations to analog hardware limits us to solving problem sizes that can fit in the hardware, and provides solutions with accuracy limited by the analog circuit. Making matters more difficult, the analog computational model provides limited choices on how to break down the PDE and map equation variables to hardware. By using analog computers directly for differential equations we risk reinventing PDE solving algorithms discovered in the digital

| | DE types | Problem abstraction | Programming model | Analog-digital interaction | Relevant $\mu$arch. features |
|---|---|---|---|---|---|
| **This work** | Nonlinear parabolic PDEs | Supports Newton solver and homotopy continuation inside digital solvers | User configures nonlinear function and Jacobian for Newton solver | Digital decomposition using red-black Gauss-Seidel; analog solution seeds digital Newton | Multi-chip integration; enhanced calibration for all analog blocks |
| **[22, 23]** | Linear elliptic PDEs | Supports sparse linear algebra inside digital solvers | User provides linear equation coefficients and constants | Digital decomposition using multigrid; analog solves recursively on linear equation residual | Automatic calibration for all analog blocks; continuous-time ADC, lookup table, DACs; impl. in 65nm CMOS |
| **[18, 19]** | Nonlinear system of ODEs | Direct mapping of ODE to analog hardware | User configures analog datapath for ODE | Digital provides continuous-time lookup for nonlinear functions | |
| **[11, 12]** | Nonlinear ODEs, linear parabolic, stochastic PDEs | Direct mapping of ODE or PDE to analog hardware | User configures analog datapath for ODE or PDE | Analog solution seeds digital Newton | Calibration only for integrators; impl. in 250nm CMOS |

**Table 5: Summary of recent work in physically prototyped analog accelerators for differential equations.**

era. We need an analog-digital program partitioning where analog kernels support existing digital solvers.

Toward that goal, we observed modern scientific computation is founded on algebraic equations, not ODEs. In effort to adapt analog acceleration to conventional digital architectures, in prior work we evaluated the merits of using an analog accelerator for linear algebra [22, 23]. The performance and efficiency gains in that work were limited due to the following reasons: first, the continuous gradient descent algorithm in the analog accelerator was inefficient to compared to optimal digital algorithms such as conjugate gradients. Second, high area costs for analog functional units meant too little computation work was done per problem instance for a given analog accelerator silicon area. So even though an analog accelerator for linear algebra would be broadly useful, we established in that prior work the gains from an analog approach to linear algebra would not be worth the overheads of using an unconventional computational model.

## 9 CONCLUSION

This paper demonstrates how hybrid analog-digital computing can be used to accelerate solving nonlinear systems of equations and partial differential equations. Because the analog model of computing uses continuous-value variables and evolves in continuous time, we are able to use the continuous Newton method and homotopy continuation for solving nonlinear equations. In contrast to the linear algebra case study in prior work [22, 23], the continuous Newton method used in this paper is competitive with the digital alternative, for two reasons: first, the analog continuous-time algorithm is less sensitive to the choice of algorithm step sizes and initial conditions, while the prototypical digital algorithm for nonlinear equations needs careful tuning for these parameters. Second, the continuous Newton method repeatedly invokes the continuous gradient descent subcircuit (see Figure 1) as an analog "subroutine" for linear algebra, resulting in higher computational intensity than the linear algebra case study, effectively pulling a greater proportion of PDE solving work into the analog domain.

We tested our ideas on a multi-chip system of physically prototyped analog accelerators. The approximate analog solutions, when used in a divide-and-conquer scheme to break down large problems, are shown to accelerate the Newton method inside a precise digital nonlinear PDE solver. Using a simulated model of an analog accelerator that fits in 350mm$^2$, we predict such an accelerator could reduce solution times for the innermost Newton method loops on a GPU by 5.7×, and reduce energy consumption by 11.6×. The insight relayed by this paper is we get tangible performance and efficiency improvements by seeking out problems where digital stumbles and analog can succeed.

The missing analog-digital program partitioning for analog accelerators may be *continuous algorithms*. These algorithms are continuous-time analogs of iterative numerical methods that are workhorses of scientific computing [1, 7, 9, 10]. Continuous algorithms include continuous gradient descent for linear algebra, continuous Newton's and homotopy continuation for nonlinear equations, and others for problems such as eigenanalysis and linear programming. Because the digital counterparts to continuous algorithms are iterative numerical methods, they are amenable to the approximation technique we use in this paper where an analog approximation serves to replace or reduce the iterations done in a digital algorithm. As we look to the analog model of computation to overcome limitations in digital computing, continuous algorithms point the way to additional analog kernels.

## 10 ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1239134, the Defense Advanced Research Projects Agency (DARPA) under Contract No. D16PC00089, and an Alfred P. Sloan Foundation Fellowship.

## REFERENCES


[1] Pierre-Antoine Absil. 2006. Continuous-time Systems that Solve Computational Problems. *IJUC* 2 (2006), 291–304.

[2] Karsten Ahnert and Mario Mulansky. 2011. Odeint - Solving Ordinary Differential Equations in C++. *AIP Conference Proceedings* 1389, 1 (2011), 1586–1589. https://doi.org/10.1063/1.3637934

[3] E.L. Allgower and K. Georg. 2012. *Numerical Continuation Methods: An Introduction*. Springer Berlin Heidelberg. https://books.google.com/books?id=0-TtCAAAQBAJ

[4] Hartwig Anzt, Vincent Heuveline, and Björn Rocker. 2011. An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations. In *Proceedings of the 9th International Conference on High Performance Computing for Computational Science (VECPAR'10)*. Springer-Verlag, Berlin, Heidelberg, 58–70. http://dl.acm.org/citation.cfm?id=1964238.1964248

[5] Marc Baboulin, Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Julie Langou, Julien Langou, Piotr Luszczek, and Stanimire Tomov. 2009. Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications* 180, 12 (12 2009), 2526–2533. https://doi.org/10.1016/j.cpc.2008.11.005

[6] G.A. Bekey and W.J. Karplus. 1968. *Hybrid computation*. Wiley.

[7] Olivier Bournez and Manuel L. Campagnolo. 2008. *A Survey on Continuous Time Computations*. Springer New York, New York, NY, 383–423. https://doi.org/10.1007/978-0-387-68546-5_17

[8] Alfredo Buttari, Jack Dongarra, Julie Langou, Julien Langou, Piotr Luszczek, and Jakub Kurzak. 2007. Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems. *Int. J. High Perform. Comput. Appl.* 21, 4 (Nov. 2007), 457–466. https://doi.org/10.1177/1094342007084026

[9] Moody T. Chu. 1988. On the Continuous Realization of Iterative Processes. *SIAM Rev.* 30, 3 (1988), 375–387. https://doi.org/10.1137/1030090 arXiv:http://dx.doi.org/10.1137/1030090

[10] Moody T. Chu. 1994. A list of matrix flows with applications. In *in Hamiltonian and Gradients Flows, Algorithms and Control*. 87–97.

[11] G.E.R. Cowan, R.C. Melville, and Y. Tsividis. 2005. 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*. 82–586 Vol. 1. https://doi.org/10.1109/ISSCC.2005.1493879

[12] G.E.R. Cowan, R.C. Melville, and Y. Tsividis. 2006. A VLSI analog computer/digital computer accelerator. *Solid-State Circuits, IEEE Journal of* 41, 1 (Jan 2006), 42–53. https://doi.org/10.1109/JSSC.2005.858618

[13] E. P. DeBenedictis. 2016. Computational Complexity and New Computing Approaches. *Computer* 49, 12 (Dec 2016), 76–79. https://doi.org/10.1109/MC.2016.353

[14] Peter Deuflhard. 2011. *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*. Springer Publishing Company, Incorporated.

[15] S. Fifer. 1961. *Analogue Computation: Theory, Techniques, and Applications*. Number v. 3 in Analogue Computation: Theory, Techniques, and Applications. McGraw-Hill.

[16] C. Fletcher. 1991. *Computational Techniques for Fluid Dynamics 1*. Springer Berlin Heidelberg. https://books.google.com/books?id=KplQ9VP--cwC

[17] W. Gautschi. 2011. *Numerical Analysis*. Birkhäuser Boston. https://books.google.com/books?id=-fgjJF9yAIwC

[18] N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis. 2015. Continuous-time hybrid computation with programmable nonlinearities. In *European Solid-State Circuits Conference (ESSCIRC), ESSCIRC 2015 - 41st*. 279–282. https://doi.org/10.1109/ESSCIRC.2015.7313881

[19] N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis. 2016. Energy-Efficient Hybrid Analog/Digital Approximate Computation in Continuous Time. *IEEE Journal of Solid-State Circuits* 51, 7 (July 2016), 1514–1524. https://doi.org/10.1109/JSSC.2016.2543729

[20] John L. Hennessy and David A. Patterson. 2011. *Computer Architecture, Fifth Edition: A Quantitative Approach* (5th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[21] Steven M. Hetzler. 1997. A Continuous Version of Newton's Method. *The College Mathematics Journal* 28, 5 (1997), 348–351. http://www.jstor.org/stable/2687062

[22] Y. Huang, N. Guo, M. Seok, Y. Tsividis, and S. Sethumadhavan. 2016. Evaluation of an Analog Accelerator for Linear Algebra. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 570–582. https://doi.org/10.1109/ISCA.2016.56

[23] Y. Huang, N. Guo, M. Seok, Y. Tsividis, and S. Sethumadhavan. 2017. Analog Computing in a Modern Context: A Linear Algebra Accelerator Case Study. *IEEE Micro* 37, 3 (2017), 30–38. https://doi.org/10.1109/MM.2017.55

[24] A.S. Jackson. 1960. *Analog computation*. McGraw-Hill.

[25] Jon Jacobsen, Owen Lewis, and Bradley Tennis. 2007. Approximations of continuous Newton's method: an extension of Cayley's problem. *Electronic Journal of Differential Equations (EJDE) [electronic only]* 2007 (2007), 163–173. http://eudml.org/doc/127842

[26] W.J. Karplus. 1958. *Analog simulation: solution of field problems*. McGraw-Hill.

[27] W.J. Karplus and W.W. Soroka. 1959. *Analog Methods: Computation and Simulation*. McGraw-Hill.

[28] S. Leyffer, S. M. Wild, M. Fagan, M. Snir, K. Palem, K. Yoshii, and H. Finkel. 2016. Doing Moore with Less – Leapfrogging Moore's Law with Inexactness for Supercomputing. *ArXiv e-prints* (Oct. 2016). arXiv:cs.OH/1610.02606

[29] F. Milano. 2009. Continuous Newton's Method for Power Flow Analysis. *IEEE Transactions on Power Systems* 24, 1 (Feb 2009), 50–57. https://doi.org/10.1109/TPWRS.2008.2004820

[30] A. Morgan. 1987. *Solving polynomial systems using continuation for engineering and scientific problems*. Prentice-Hall. https://books.google.com/books?id=E3amAAAAIAAJ

[31] J. W. Neuberger. 2007. The Continuous Newton's Method, Inverse Functions, and Nash-Moser. *The American Mathematical Monthly* 114, 5 (2007), 432–437. http://www.jstor.org/stable/27642222

[32] J. Ortega and W. Rheinboldt. 2000. *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898719468 arXiv:http://epubs.siam.org/doi/pdf/10.1137/1.9780898719468

[33] F. Rothganger, C. D. James, and J. B. Aimone. 2016. Computing with dynamical systems. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 1–3. https://doi.org/10.1109/ICRC.2016.7738701

[34] Dietmar Saupe. 1989. *Discrete Versus Continuous Newton's Method: A Case Study*. Springer Netherlands, Dordrecht, 59–80. https://doi.org/10.1007/978-94-009-2281-5_2

[35] J. Schemmel, J. Fieres, and K. Meier. 2008. Wafer-scale integration of analog neural networks. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 431–438. https://doi.org/10.1109/IJCNN.2008.4633828

[36] R. Vichnevetsky. 1968. Analog/hybrid solution of partial differential equations in the nuclear industry. *SIMULATION* 11, 6 (1968), 269–281. https://doi.org/10.1177/003754976801100605 arXiv:http://sim.sagepub.com/content/11/6/269.full.pdf+html

[37] James A. Walsh. 1995. The Dynamics of Newton's Method for Cubic Polynomials. *The College Mathematics Journal* 26, 1 (1995), 22–28. http://www.jstor.org/stable/2687287

[38] B.R. Wilkins. 1970. *Analogue and iterative methods in computation, simulation, and control*. Chapman and Hall.