Hybrid Analog-Digital Co-Processing for Scientific Computation

Yipeng Huang

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2018

ABSTRACT

Hybrid Analog-Digital Co-Processing for Scientific Computation

Yipeng Huang

In the past 10 years computer architecture research has moved to more heterogeneity and less adherence to conventional abstractions. Scientists and engineers hold an unshakable belief that computing holds keys to unlocking humanity's Grand Challenges. Acting on that belief they have looked deeper into computer architecture to find specialized support for their applications. Likewise, computer architects have looked deeper into circuits and devices in search of untapped performance and efficiency. The lines between computer architecture layers—applications, algorithms, architectures, microarchitectures, circuits and devices—have blurred. Against this backdrop, a menagerie of computer architectures are on the horizon, ones that forgo basic assumptions about computer hardware, and require new thinking of how such hardware supports problems and algorithms.

This thesis is about revisiting hybrid analog-digital computing in support of diverse modern workloads. Hybrid computing had extensive applications in early computing history, and has been revisited for small-scale applications in embedded systems. But architectural support for using hybrid computing in modern workloads, at scale and with high accuracy solutions, has been lacking.

I demonstrate solving a variety of scientific computing problems, including stochastic ODEs, partial differential equations, linear algebra, and nonlinear systems of equations, as case studies in hybrid computing. I solve these problems on a system of multiple prototype analog accelerator chips built by a team at Columbia University. On that team I made contributions toward programming the chips, building the digital interface, and validating the chips' functionality. The analog accelerator chip is intended for use in conjunction with a conventional digital host computer.

The appeal and motivation for using an analog accelerator is efficiency and performance, but it comes with limitations in accuracy and problem sizes that we have to work around.

The first problem is how to do problems in this unconventional computation model. Scientific

computing phrases problems as differential equations and algebraic equations. Differential equations are a continuous view of the world, while algebraic equations are a discrete one. Prior work in analog computing mostly focused on differential equations; algebraic equations played a minor role in prior work in analog computing. The secret to using the analog accelerator to support modern workloads on conventional computers is that these two viewpoints are interchangeable. The algebraic equations that underlie most workloads can be solved as differential equations, and differential equations are naturally solvable in the analog accelerator chip. A hybrid analog-digital computer architecture can focus on solving linear and nonlinear algebra problems to support many workloads.

The second problem is how to get accurate solutions using hybrid analog-digital computing. The reason that the analog computation model gives less accurate solutions is it gives up representing numbers as digital binary numbers, and instead uses the full range of analog voltage and current to represent real numbers. Prior work has established that encoding data in analog signals gives an energy efficiency advantage as long as the analog data precision is limited. While the analog accelerator alone may be useful for energy-constrained applications where inputs and outputs are imprecise, we are more interested in using analog in conjunction with digital for precise solutions. This thesis gives novel insight that the trick to do so is to solve nonlinear problems where low-precision guesses are useful for conventional digital algorithms.

The third problem is how to solve large problems using hybrid analog-digital computing. The reason the analog computation model can't handle large problems is it gives up step-by-step discrete-time operation, instead allowing variables to evolve smoothly in continuous time. To make that happen the analog accelerator works by chaining hardware for mathematical operations end-to-end. During computation analog data flows through the hardware with no overheads in control logic and memory accesses. The downside is then the needed hardware size grows alongside problem sizes. While scientific computing researchers have for a long time split large problems into smaller subproblems to fit in digital computer constraints, this thesis is a first attempt to consider these divide-and-conquer algorithms as an essential tool in using the analog model of computation.

As we enter the post-Moore's law era of computing, unconventional architectures will offer specialized models of computation that uniquely support specific problem types. Two prominent examples are deep neural networks and quantum computers. Recent trends in computer science research show these unconventional architectures will soon have broad adoption. In this thesis I show another specialized, unconventional architecture is to use analog accelerators to solve problems in scientific computing. Computer architecture researchers will discover other important models of computation in the future. This thesis is an example of the discovery process, implementation, and evaluation of how an unconventional architecture supports specialized workloads.

Table of Contents

Bibliography **142**

List of Figures

List of Tables

Acknowledgements

I would like to thank my thesis advisor Prof. Simha Sethumadhavan for shaping my research personality from my undergrad senior year. When I asked him for a recommendation letter for grad school, he said he teaches his PhD students one area of knowledge and expects his students to teach him one in return. I appreciate the freedom and trust he has granted me to take my research in new directions, and for being patient even as I rushed to meet deadlines. By now I'm not sure what that one thing he taught me was, but if I had to boil it down to one it is how to be ever curious and ambitious. During my time with Simha I went from being a narrowly-focused student for whom things were either done or not, to being who I am today who embraces open-minded curiosity and setting challenging goals that are just out of reach. These are formative lessons I will appreciate going forward.

I thank chief among my collaborators Dr. Ning Guo whose indomitable ethic and careful work on the prototype chips laid a solid foundation for my research. This thesis would not have been possible without him.

I thank Prof. Margaret Martonosi of Princeton University for her time to serve on my thesis committee. I thank my collaborators Prof. Yannis Tsividis, Prof. Mingoo Seok, and Prof. Kyle Mandli, for the time and energy they vested in my research work. I have learned from them to be rigorous in my research, and that there are always new tricks to try in new areas of engineering and mathematics.

I thank my professors and teachers over the many years of school, especially Prof. Martha Kim, for the many times she has coached me to tell the real story and meaning of research, not just the facts, by flipping the whole paper or presentation upside down. I would also like to thank Prof. Luca Carloni, Prof. Steven Nowick, Prof. Steven Edwards, Prof. Peter Allen, Prof. Janet

Kayfetz, and Prof. Dan Rubenstein for their collaboration, supervision, and for their teaching that has inspired me to pursue my PhD studies. I must also thank my teachers from before Columbia University, including Mr. Rice, Mr. Peck, Mr. James, Ms. Chen, and Ms. Lin for their dedication to their students.

I am most grateful for and indebted to my family. First to my parents, who have supported me through my education, always nurturing my scholarship since before I was aware, and for encouraging me to pursue my interests as long as I did so with conviction. And to my siblings Eric and Charlotte, who as older siblings have always been my role models. And finally to my fiancée Catherine, who has shared the best and worst moments of this whole journey, and for getting me to wear collared shirts instead of T-shirts I get for free, and generally making me a more highly functioning human being.

To my parents.

Part I

Introduction

Chapter 1

Digital and Analog Accelerators

Improvements in computer architecture performance and efficiency have thus far been driven by three broad principles: scaling, parallelism, and specialization. Around 2005, the first of those principles faltered with the end of Dennard scaling, leading to the forecast then that increased core-level parallelism could make up for shortfalls in frequency scaling. Except difficulties in parallel programming and power-wall constraints have foreshortened the reign of the multicore era as well: instead of 1000s of cores in consumer CPUs, we have around a dozen. Hardware specialization using digital accelerators has been an alternative to scaling and parallelism for driving performance and efficiency. Specialization using digital accelerators offers several advantages over general-purpose CPUs, but it also suffers from limitations that I will discuss in the rest of this introduction. Because of these limitations, computer architecture researchers must consider extensions or alternatives to digital accelerators.

This thesis is about my team's experience and findings from building and using analog accelerators in a hybrid analog-digital architecture. We can think of analog accelerators as extensions to digital accelerators: both types of accelerators use specialization to provide greater performance and efficiency relative to the general-purpose alternative. At the same time, analog accelerators use a fundamentally different model of computation, so we should consider them outside of the conventional computer architecture principles of scaling, parallelism and specialization. In this introduction chapter I will first discuss analog accelerators as extensions to digital accelerators; later, I will discuss analog accelerators as an alternative model of computation.

Digital accelerators provide higher performance and efficiency through several mechanisms, including reducing overheads in control logic by specializing on a small set of algorithms, extracting greater parallelism from the workload, exploiting a memory hierarchy design tailored to the problem, and customizing the representation of data to the inputs. Analog accelerators carry forward many of these ideas.

1.1 Specialization in application- and domain-specific digital and analog accelerators

Specialization using accelerators has been a commercially successful idea. But many digital accelerators thus far have been application-specific accelerators, which have limited applications outside of a few obvious intensive applications. A more sustainable approach to building accelerators is to build domain-specific accelerators, which are specialized hardware that target a whole class of problems. Designing domain-specific accelerators needs more clever insight into workloads in order to map them into hardware, so relatively few domain-specific accelerators have been proven to work. As I show in this thesis, analog accelerators are an important type of domain-specific accelerator, suited for scientific computation workloads.

The first idea in digital accelerators is to specialize the control logic so the accelerator hardware supports a limited set of operations or algorithms. Doing so reduces the overhead costs of fetching and decoding instructions, ensuring that the hardware spends a greater proportion of time and energy in doing logical operations that actually carry out the algorithm [70, 140].

Many commercially realized accelerators belong to application-specific accelerators. Application-specific accelerators replace a well-defined innermost software loop or function call, and offer relatively little flexibility to accommodate different algorithms, so they serve only to accelerate some extremely intensive workload kernels such as AES encryption, multimedia codecs, or cryptocurrency hashing. Relatively speaking, application-specific accelerators are easier to build and use, and so they have been commercially successful for those well-defined kernels [97]. But eventually computer architecture researchers will run out of obvious target kernels for hardware acceleration, and each additional accelerator costs silicon area that could be used for general-purpose cores or other accelerators.

A more sustainable approach to architecture specialization is to build domain-specific accelerators. In domain-specific accelerators, the digital accelerator serves to tackle a whole well-defined class of problems in lieu of software. Domain-specific digital accelerators have the flexibility to reconfigure pipelines to implement different algorithms to support more problems. For example, database query accelerators can handle different query plans [182, 113], and neural network accelerators likewise can handle different neural network topologies [55, 31].

Prior work in analog computing, along with the work presented in this thesis, show that analog accelerators are an important domain-specific accelerator design. For example, the prototype analog accelerator developed at Columbia University comprises a few types of hardware units for multiplication, addition, and integration. By reconfiguring how these operations are pipelined, the analog accelerator can solve many types of problems in scientific computing, including differential equations and algebraic equations.

The case for domain-specific accelerators may be stronger than application-specific ones, as they can tackle a whole class of important problems; however, capturing a broader workload using domain-specific accelerators comes with overhead costs in area, performance, and efficiency, and striking a good balance between workload breadth and associated costs is difficult. So unless computer architecture researchers keep finding clever domain-specific accelerators for more classes of workloads, the accelerator specialization era of computer architecture may be limited, just like the multicore and frequency scaling eras before it.

1.2 Data-level parallelism in digital and analog accelerators

Another reason specialized hardware accelerators offer higher performance and efficiency relative to general-purpose approaches is greater parallelism. Specialized accelerators can make stronger assumptions about the structure and regularity of the workloads they support. As a result, accelerators can have wider single-instruction multiple-data operations (SIMD) compared to general-purpose designs. Prominent examples include GPGPUs, along with wide matrix-vector multiplication operations common in multimedia and neural network accelerators.

The amount of data-level parallelism that a digital accelerator can achieve is limited compared to analog accelerators. We can understand this point at two conceptual levels: one in terms of digital vs. analog matrix-vector multiplications, and the other in terms of what the data-level parallel operations are actually doing in the context of the overall algorithm.

Analog accelerators are able to realize much wider matrix-vector multiplication than digital accelerators. For example, an analog accelerator design can use 1,024 transistors and memristors to multiply a 1,024 element matrix against an input vector [79, 159]. While on the other hand a digital accelerator may need three times as many transistors just to multiply two 8-bit binary numbers [147]. As I will discuss in later in this introduction in Section 3.3, floating gate transistor amplifiers and emerging devices such as memristors have been useful for building wide matrix-vector multipliers. While those analog accelerator designs are not the focus of this thesis, those analog accelerator designs have been successful at accelerating neural network classifiers.

The other aspect of comparing data-level parallelism in digital and analog accelerators concerns what the parallel operations actually do, in the context of the overall algorithm. In the scientific computing case studies that are the focus of this thesis, SIMD operations in digital architectures take place in the innermost loop of numerical methods, which are multiple levels of iterations below the iteration that actually advances the physical model of interest. In some analog accelerator approaches for scientific computation, the parallel operations take place right at the level of the entire physical model, simultaneously advancing multiple spatial cells of the physical model in time. So in effect the parallel operations in an analog accelerator in those use cases span a deeper cross section of the workload.

1.3 In-memory computation in digital and analog accelerators

Some digital accelerator designs focus on specializing the memory hierarchy for a workload, by either providing specialized cache designs, by increasing the volume of memory available to the accelerator, or by bringing computation into the memory itself. In all these designs intermediate computational results for algorithms still have to be explicitly taken from or stored into memory, whether that memory is as far as DRAM main memory or as local as pipeline registers.

In the analog accelerator alternative, intermediate computational results are not stored into registers in the conventional sense altogether. Rather, as I will discuss in Section 2.1, intermediate computational results in an analog accelerator are merely analog circuit transient signals, dispersed throughout the analog accelerator circuit.

1.4 Approximate computing in digital and analog accelerators

Finally, an important trick in digital accelerators is to customize the representation of data to best match the workload. For example, a digital accelerator can make stronger assumptions on the precision needed for a given workload, and increase the amount of exploitable data-level parallelism by decreasing the precision of data and operations. Analog accelerators offer a logical extension to this approach by encoding data as analog variables, which I discuss in detail in Section 2.2.

Digital accelerators draw on the above four mechanisms—specialization, parallelization, increasing memory bandwidth, and approximation—to deliver higher performance and efficiency for specific workloads than general-purpose approaches. Nonetheless, the fundamental model of computation remains the same in CPUs, GPUs, and even field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) accelerators. That is, these digital architectures all operate step-by-step on binary digital numbers.

The potential in analog accelerators and hybrid analog-digital is not just that they extend ideas in digital accelerators; rather, their potential is in they offer different abstractions, which I make clear next in these introductory chapters. With those different abstractions, hybrid analog-digital computing may go beyond scaling, parallelism, and specialization in driving architecture improvements.

Chapter 2

Hybrid Analog-Digital Co-Processing: Definition & Motivation

As we approach the limits of silicon scaling, it behooves us to reexamine fundamental assumptions of modern computing, even well-served ones, to see if they are hindering performance and efficiency. I discuss in this thesis our research group's experience in using and building *analog accelerators* that break two such fundamental assumptions in modern computing: First, analog accelerators encode numbers as *analog variables*, which take the full range of circuit voltage and current, in contrast to *digital binary variables*, which is how data is encoded in all conventional digital computers. Second, analog accelerators update their values continuously, in *continuous time*, in contrast to operating step by step in *discrete time* as is done in all conventional digital computers.

Moving to a continuous time analog model of computation puts analog acceleration closer to the way biological brains work. Even though digital computers have made orders magnitudes of improvement in performance and energy efficiency in roughly the past half century, the biological brain is still the epitome of an energy efficient computer. According to Sarpeshkar, the human brain delivers a computational performance of 3.6×10^{15} to 1.0×10^{16} approximate floating point operations per second, on a power budget of 12 to 15 Watts, for an energy efficiency of 3×10^{14} FLOPS/W [147]. On the other hand, a Intel Westmere CPU from around 2011 has an energy efficiency of 5.88×10^{8} FLOPS/W [95]. The best commercial digital processors are at best around 1/1,000,000 as energy efficient in terms of FLOPS/W compared to the biological brain. Writing in 1990, Mead predicted this six orders of magnitude difference between best digital designs and biological brains [124]: according to his estimates, $100\times$ of the shortfall in digital processors can be covered by improving computation locality, by switching to designs that exploit data and thread level parallelism such as SIMD and GPUs, and by switching to specialized accelerators implemented in FPGAs and ASICs. The remaining $10,000\times$ shortfall, according to Mead, is due to costs in doing operations with digital numbers instead of analog signals as in the brain. The energy efficiency benefits of continuous time analog computation is highly problem-dependent: low estimates are around $1,000\times$ [73] while high estimates are around $100,000\times$ [149].

Therefore, the motivation for investigating analog computation is to extract the benefit of using analog variables and continuous time operation. Researchers have written extensively about the energy efficiency of using analog variables for approximate computing. Comparatively little has been said about the benefits of continuous time operation. Below, I discuss in Section 2.1 the benefits we observed of using a continuous-time analog accelerator in the context of solving scientific computing problems. Then, in Section 2.2 I summarize prior work discussing the energy efficiency of analog encodings and analog operators.

Despite the potential benefits of analog continuous time computation, computer architecture researchers have held mixed opinions about analog architectures, due to several reasons:

1. Continuous time computation supports only one type of operation—solving ordinary differential equations (ODEs)—and therefore appear to have limited applications.
2. Analog variables are more efficient than digital variables only for low bit precision, and therefore analog solutions are inaccurate and imprecise.
3. Analog continuous time accelerators have high hardware costs in storing analog variables and analog / digital conversion, and therefore support only limited problem sizes.

These are significant criticisms against analog acceleration, and we have endeavored to find problems, solutions methods, and analog architecture designs that mitigate these downsides. Sections 3.1, 3.2, and 3.3 discuss insights in problem types, algorithms, architecture, and circuit devices that address the above downsides of analog acceleration.

2.1 Continuous-time asynchronous signaling

The performance and efficiency benefit of using an analog accelerator relative to a conventional digital computer comes from two distinct levels: explicit data-graph execution and continuous-time operation. The first advantage is common between analog and digital accelerators (as I discussed in Section 1.1), while the second advantage is unique to analog accelerators.

Accelerating long iterations with explicit data-graph execution

In both analog and digital accelerators, explicit data-graph execution (EDGE) reduces the overhead costs of general-purpose computation by setting up hardware circuits for predefined tasks prior to computation. An EDGE architecture chains hardware functional units end-to-end so when data arrives, signals representing intermediate results flow from one unit to the next, with no overheads in fetching and decoding instructions, and furthermore no fetching and writing back data between iterations.

In digital architectures, application-specific EDGE architectures provide roughly $100\times$ to $1,000\times$ improvement relative to general purpose CPUs in speed or energy consumption, depending on which is more important [70, 140, 171]. In analog accelerators however the advantage of EDGE architectures is less clear-cut, because an analog accelerator cannot compete directly with a conventional digital computer in terms of problem size and solution accuracy and precision.

Implementing continuous algorithms with continuous-time operation

While analog and digital accelerators both use explicit data-graph execution to deliver higher performance and efficiency relative to general purpose CPUs, analog accelerators do so in continuous time giving it a distinct advantage relative to conventional hardware. The continuous-time operation of analog accelerators eliminates overheads caused by step-by-step operation in conventional digital computers. Furthermore, we can realize in analog accelerators a class of *continuous algorithms* useful in scientific computation which have no equivalent in discrete-time digital computers.

Step-by-step operation is an important abstraction that has allowed us to design scalable digital hardware. In a conventional digital computer, variables are transferred between digital registers synchronously, coordinated by a clock signal. Digital data is only valid on clock edges in digital hardware, thus avoiding timing concerns about signal phase shift and propagation time, making it easier to coordinate movement of data in digital hardware. These assumptions are even more important because they hide the internal timing behavior of hardware submodules, and enable techniques such as Moore machines and latency insensitive design [26] for easily composing hardware.

The clock signal now has itself become a limitation in terms of efficiency and performance for digital designs. In terms of efficiency, the clock signal is often the most power-hungry signal in chip designs. In terms of performance, the clock frequency ultimately limits how fast algorithms return solutions.

Because computers have for a long time operated step-by-step in a discrete-time synchronous model of computation, researchers have had to phrase algorithms also as step-by-step operations, where variables change by definite increments at discrete time points. If we can break free of that fundamental abstraction then we can consider *continuous algorithms*, where variables change by infinitesimal amounts continuously.

Doing continuous algorithms phrased as ODEs in continuous-time analog accelerators has advantages, particularly in areas of scientific computation I discuss in this thesis:

1. In Chapter 8 we follow in the footsteps of prior work, and solve stochastic ordinary differential equations using a continuous time analog accelerator. The analog accelerator naturally solves ODEs, giving solutions $3\times$ faster than an equivalent digital solver giving approximate solutions. Furthermore the analog accelerator uses continuous time analog noise, instead of having to generate and process pseudorandom numbers as in a digital computer, resulting in the analog solver being overall $32\times$ faster than the purely digital approach, for equal accuracy.
2. In Chapter 10 we solve linear algebra problems in an analog accelerator by doing steepest gradient descent, a version of an iterative numerical linear algebra method turned into an ODE. In a typical iterative numerical linear algebra method such as conjugate gradients, half of the multiplications in each step are just for finding the right step size. On the other hand an analog accelerator does not need to find step sizes as the solution evolves continuously toward the solution point.
3. In Chapter 11 we solve nonlinear systems of equations in an analog accelerator by carrying out the continuous Newton method, also an iterative numerical method turned into an ODE.

The classical digital algorithm for solving nonlinear systems of equations has a difficult time finding correct step sizes, to the extent most of the initial algorithm steps may not move guesses toward the correct solution at all. An analog accelerator uses the more reliable continuous-time versions of the algorithm to come up with good approximate guesses, which the digital computer then refines. For larger, more nonlinear problems, the hybrid analog-digital solver has a performance improvement of 5.7× and energy savings of 11.6×, relative to a GPU without the help of an analog accelerator.

The continuous-time model of computation allows us to expand the definition of algorithms beyond step-by-step sequences that give a solution, to include dynamical systems which evolve from some initial state to a final state that represent a solution. In terms of hardware microarchitecture and architectures, the continuous time model eliminates clocked operation and the von Neumann architecture organization of hardware.

2.2 Continuous analog value encoding

The most commonly cited advantage of continuous time analog computation is the energy efficiency of using analog variables, albeit for low-precision solutions only. The reason using analog encodings is efficient is because an analog encoding packs more data on a single wire, and because many analog operations are cheaper compared to digital. While that case for analog is intuitive and appealing, a quantitative comparison between analog and digital encodings show analog is more efficient than digital only for low-precision data. Furthermore no practical error-correction is possible on continuous time analog variables, making purely analog encodings less appealing for large scale systems.

Dense analog data encoding and simple analog operations

Analog encodings are relatively more efficient than digital encodings in terms of how to represent data, how to represent changes in data, and how to do simple operations on data.

In terms of representing data, a single wire in an analog accelerator can represent the full range of values at any moment, unlike a digital representation that would need a parallel multi-wire bus. In terms of representing a change in data, sweeping an analog value from one extreme value to the other extreme value simply means changing the voltage or current that encodes the value; on the other hand, sweeping a digitally represented 8-bit unsigned integer from 0 to 255 needs 502 binary inversions, while a more economical Gray encoding still needs 255 inversions.

Furthermore, in terms of doing simple operations on analog and digital data, many mathematical operations for analog data are more efficient than their digital counterparts. For example, a single wire can add two analog numbers together using Kirchhoff's current law, but about 240 transistors are needed to add two 8-bit digital integers [147]. The contrast is even more stark for multiplication: 4 to 8 transistors can multiply two analog numbers together in a Gilbert cell, but as many 3000 transistors are needed to multiply two 8-bit digital integers (for a maximum speed logic design) [147].

Digital precision doubles by increasing bits while analog precisions doubles by doubling signal-to-noise ratio

The actual relative costs of analog and digital data representation and operations depends on the data precision.

While an analog multiplier may seem cheaper and more efficient than an 8-bit digital multiplier in terms of transistor count, comparing the area and power consumption of the two is not so straightforward. The few transistors in the analog multiplier have to be large enough in size, such that when their sizes vary due to imperfections during manufacturing, the analog multiplication is still correct to 8-bit precision. The transistors in the digital multiplier in contrast can be all minimally sized; even when their sizes vary due to manufacturing imperfection, we can still unambiguously interpret the multiplier output bits as 0 or 1.

The case for analog becomes much less favorable for more precise computation. Suppose we now want to compute on 16-bit digital numbers, in effort to either increase dynamic range or precision. In digital, 16-bit addition becomes 2× as expensive as 8-bit addition [150], while multiplication is 4× as expensive. Each additional digital bit doubles the equivalent analog signal-to-noise ratio (SNR): the way to understand this is adding a single bit of digital range entails doubling the analog signal, and adding a single bit of digital precision entails halving the analog noise [148]. Doubling the SNR in an optimal, well-designed analog circuit entails doubling area and doubling power (to control thermal noise) [147]. In all, 16-bit analog addition alone becomes 256$\times$ as expensive compared to 8-bit in terms of area and power [150]!

The crossover point in bit precision after which analog is less efficient compared to digital representation is 8-bit, according to Sarpeshkar [147], or 12-bit according to Hasler provided efficient means to calibrate away process variations [73].

Digital has error correction but continuous-time analog does not

Digital computers have ways to detect and correct errors in binary numbers when noise causes error in the data. Digital error correction works by transmitting more bits than the actual data conveyed. The receiver can detect errors and infer the most likely intended correct data. This works because the correct data are attractive states that pull small errors back to the correct data. In contrast, in analog circuits any variation in analog variables becomes an error in the data, and unfortunately there is no easy way to do error detection or correction on analog variables.

One approach to get error correction in analog signals is to interleave analog and digital encoding of data [147, 150]. In those systems, analog data is frequently quantized by ADCs and regenerated using DACs. This ensures analog data is frequently pulled back to a finite number of valid, digital data. This controls for imprecision but is costly.

Another approach offers analog error correction but only for discrete-time data [112, 80]. In those approaches, the set of valid analog data are made to be a small subset of all the analog data that may be transmitted over a channel. Some highly nonlinear transformations make small changes in the intended data become big changes in the transmitted data, so the receiver can detect errors and find the intended data. Encoding and decoding takes time, so this only works for discrete-time, sampled analog data.

We are interested in computing on analog data, in continuous time, so neither analog error correction approaches I discussed above can cheaply eliminate noise.

Because the analog style of computation breaks the two foundational assumptions in digital computing, the rest of the design of the computer changes as well. I summarize in Table 2.1 the major ways in which analog and digital computers are different. Table 2.1 serves as a definition of the analog style of computing, relative to the dominant, conventional, digital style. From the kinds

	ANALOG	**DIGITAL**
Numerical method primitives	**Ordinary differential equations:** Analog computer techniques for physics simulations, developed in the 1950s and 1960s, rephrase the original problem as systems of ODEs. ODEs can readily be mapped on analog computing hardware.	**Linear algebra:** Modern digital computer techniques for physics simulations, on the other hand, rephrase the problem as linear algebra [36, 169]. Linear algebra algorithms are optimized to run well on modern hardware, and serve as a high-performance and efficient platform for complex simulations.
Algorithms	**Iterative, approximate:** Analog solvers are similar to iterative numerical methods for linear algebra, such as conjugate gradients and successive over-relaxation. These numerical methods take an initial guess at the correct solution vector, and let it evolve into a better approximation of the correct solution. The intermediate results of iterative numerical methods are approximations of the solution vector.	**Direct, exact:** The step-by-step operation of digital computers allows use of direct numerical linear algebra methods, such as Cholesky decomposition and QR decomposition. These numerical methods factor the linear algebra equation, and obtain solutions for components of the solution vector one component at a time. The intermediate results of direct methods are incomplete: some components of the solution vector are unsolved until the algorithm ends.
Programming model	**Declarative programming:** programming is done by defining connectivity between hardware components and by setting constraints on variables. This programming model has the advantages of being a better analogy for modeling dynamical systems. E.g., SystemVerilog, Simulink.	**Imperative programming:** programming is done by step-by-step instructions for algorithms. This programming model matches mainstream digital hardware, where operations happen step-by-step on data stored in registers, cache, and memory. E.g., C++.
Interconnect	**Circuit switched:** connectivity between hardware is set before computation in analog hardware, and remain fixed during the continuous-time evolution of any one variable. Circuit switched interconnects eliminate the overhead of decoding instructions and setting datapaths during execution.	**Packet switched:** Connectivity between hardware can vary when a program runs. The step-by-step operation allows pieces of data to be routed on a changing datapath. Packet switched interconnects incur an overhead in dynamically routing data, but allows time-multiplexing on wires to carry more variables.
Signaling	**Continuous time, asynchronous:** the signals evolve in timesteps that are infinitesimally small.	**Discrete time, synchronous:** the signals evolve in finite timesteps, which need synchronization with a clock to have correct semantics.
Data representation	**Continuous valued:** analog voltage, current, or spike probability.	**Discrete valued:** binary integers for integers, IEEE floating point for real numbers.
Physical devices	**Transistors as amplifiers:** computation takes place, even without a full swing in the signal between positive and negative supply voltage values. Many physical processes outside of MOS transistors can be used.	**Transistors as switches:** is becoming less favorable, and binary operations cost a fixed size charge per inversion.

Table 2.1: Comparison of analog and digital computing stacks.

of problems the two styles of computing can do, down to how the hardware encodes and moves data, analog and digital computers are profoundly different.

Chapter 3

Hybrid Analog-Digital Co-Processing: Challenges & Mitigations

Even though analog value representation is tangibly more efficient than digital for low-precision computation, giving up the basic abstraction of using binary numbers limits analog acceleration to give only low accuracy solutions. Likewise, even though continuous-time operation potentially offers high performance due to new types of continuous algorithms, the fact that analog accelerators are an EDGE architecture means hardware costs grow in conjunction with problem sizes. Moreover, there is the question whether analog acceleration is useful for any modern workloads. The next three sections discuss how my research work and related work in analog accelerators address the above limitations in the analog model of computation.

3.1 Expanding workload breadth and depth: analog problems and algorithms in Berkeley Dwarfs

Because analog accelerators operate in continuous time on analog variables, we model their behavior as differential equations. That was useful when early computer researchers set out to model physical phenomena using computers, as they could map mathematical descriptions for the natural world as differential equations in early analog computers. We revisit this style of mapping problems to analog accelerator hardware in Part III. Modern algorithms and computers however tackle a much broader variety of problems, and only a fraction of those workloads have anything to do with differential equations.

At first glance analog accelerators would seem useless for modern problems phrased in the language of discrete variables and operations. For example, a vast majority of modern problems ultimately are linear algebra problems. The most familiar way to solve linear algebra problems for most students is Gaussian elimination, which belongs to an important class of *direct numerical linear algebra algorithms* that solve the linear algebra problems row-by-row, element-by-element. An analog accelerator has no hope in solving linear algebra problems in such a style as the operations happen in discrete steps.

The breakthrough is analog accelerators can tackle problems, such as linear algebra, using continuous iterative numerical algorithms. In *iterative numerical linear algebra algorithms*, we start at an initial guess for the whole solution vector, and incrementally take better guesses for the whole solution, until each step changes sufficiently little indicating we've found the solution [161, 169, 96, 134]. In general iterative algorithms are becoming more important than direct ones because researchers are interested in getting approximate solutions by taking fewer steps [48]. Analog acceleration can step in and do iterative numerical algorithms in continuous time as ODEs, an idea we explore in Part IV.

My own research work, in addition to recent work by other researchers, has significantly expanded the breadth and depth of workloads that analog accelerators can handle. Following the outline laid out in prior work [158] we use the Berkeley Dwarfs [6] as a taxonomy of problems and algorithms analog accelerators can tackle. While the Dwarfs taxonomy is by now over a decade old, it remains a useful map for problems and algorithms. I further categorize the Berkeley Dwarfs as continuous math or discrete math, depending on whether the algorithms operate on real numbers.

Analog accelerator applications in continuous mathematics

Problems and algorithms in this category operate on real numbers, which naturally match the analog value encoding in analog accelerators. We can map these problems to analog accelerators if we can map the relationship between variables into connectivity inside analog hardware.

Sparse matrix

Sparse matrix linear algebra problems often come from physics simulations phrased as partial differential equations (PDEs). The sparsity of the matrix reflects the fact that state variables in any given space depend only on state variables in nearby space.

I discuss solving several types of sparse matrix problems in this thesis, including ordinary differential equations in Chapter 8; I also discuss solving parabolic and elliptic PDEs, by solving linear algebra problems in Chapter 10, and by solving nonlinear systems of equations in Chapter 11.

Most of the prior work in analog computing has been in solving sparse matrix problems. For example, prior work by Cowan et al. in solving some types of partial differential equations belong to this class of applications [40, 41]. In other work researchers have built analog accelerators where the PDE spatial domain is continuous, in addition to having continuous time and continuous value representation [128, 129, 127].

Dense matrix

Dense matrix linear algebra problems often come from optimization problems. In optimization problems variables can be arbitrarily interrelated, unlike sparse matrix PDE problems. Dense matrix problems also are kernels for physics problems converted in some ways that lead to all-to-all variable connectivity, such as in N-body problems (where variables interact over long range), or as a result of spectral domain methods (which convert large sparse problems to small dense ones).

In related work other researchers have explored solving in analog accelerators optimization problems including linear programming [58, 23] and quadratic programming [35, 172].

MapReduce & Monte Carlo

Monte Carlo problems use random variables to model properties not fully described by deterministic models. The computation result comes from solving a large ensemble of solutions, each with a different sample of a random distribution.

I discuss solving stochastic differential equations (SDEs) in Chapter 8, exploring how to generate analog noise, and practical details of using analog noise as a random input for SDEs. In related work other researchers have used other physical processes for generating and shaping random distributions for Markov problems [176, 177]

Structured grid

Structured grid problems again come from physics simulations phrased as PDEs. Structured grid solvers split the space and time variables in PDEs into orderly grids and intervals. A notable example is the multigrid method for solving elliptic PDEs, which split PDEs into a hierarchy of coarse and fine grids. In the multigrid method the PDE is solved approximately at every level of discretization resolution, and the coarse solutions serve as initial guesses for fine solutions.

I explore in Chapter 10 using analog accelerators for structured grid problems, by approximately solving linear algebra problems to aid a multigrid solver.

Unstructured grid

Unstructured grid problems also come from physics simulations phrased as PDEs. But unlike structured grid solvers, unstructured grid solvers arbitrarily change the size and shape of space cells to suit the problem geometry. Examples include finite volume and finite element discretization schemes. Because the variables are much less ordered, unstructured grid solvers use memory pointers for bookkeeping where state variables are stored in memory. In these methods resolving memory locations consumes more computation time, making analog accelerator support for unstructured grids difficult.

Spectral methods

Spectral methods are an important transformation in physics simulations. They convert sparsely interrelated variables into a smaller but equivalent problem with all-to-all connectivity. In related work other researchers have devised analog accelerators for spectral methods such as discrete Fourier transforms [78].

N-body

N-body problems are physics problems where variables interrelate over long distances, such as force field simulations for astronomy and molecular dynamics [64, 10]. These problems have many discrete operations to deal with spatial geometry, and have dense matrices which are costly to solve in analog.

Analog accelerator applications in mixed continuous-discrete mathematics

The next category of problems and algorithms primarily operate on discrete math structures such as graphs and trees, though sometimes these algorithms have inner loops operating on real values. Analog computing may help with the inner loops.

Dynamic programming

Dynamic programming is broadly a memoization technique that breaks down problems into optimal subproblems. The algorithms need random access to memory to jump to whichever subproblem is currently the most optimal, so in general these algorithms are difficult to map to analog accelerators.

In related work other researchers have built continuous-time asynchronous circuits for solving a prototypical dynamic programming problem, sequence alignment and matching [116, 117, 118, 115, 119].

Graphical methods

In related work other researchers have proposed accelerators for Bayesian networks [106, 107]. It remains an open research question whether analog accelerators can tackle more advanced dynamic programming / graphical method problems, such as the Viterbi algorithm, hidden Markov models, and Hamilton-Jacobi-Bellman equations for optimal control.

Backtrack and branch-and-bound

Backtrack and branch-and-bound problems include mixed continuous-discrete mathematics problems such as mixed-integer linear programming problems and linear complementarity problems. They can also be purely discrete, as in the case of Boolean satisfiability problems. Notably, in related work other researchers have explored solving Boolean satisfiability—an NP-complete problem whose all known solution algorithms have worst case exponential complexity—using Boltzmann machines [15, 17, 16], stochastic [37] or continuous-time analog [53, 130] models of computation.

Analog accelerator applications in discrete mathematics

The final category of problems and algorithms operate on discrete math structures. They can be state-full, such as *graph traversal* and *finite state machines*, or stateless, such as *combinational logic*. In general these have no loops operating on real values, so an analog solution may be elusive.

Graph traversal

In related work other researchers have shown analog accelerators can sort a set of real numbers, albeit at at higher complexity cost compared to digital techniques. Specifically an analog accelerator sorts numbers as a side effect of doing the QR algorithm, an eigenanalysis algorithm [23, 1, 14, 142, 74].

3.2 Refining solution accuracy and precision: analog approximations as digital seeds

As established in Section 2.2, analog accelerators are only comparatively efficient for low bit-precision solutions. Indeed, such a constraint rules out broad classes of analog applications I just described in Section 3.1: in many problems the proposed analog algorithms only "work" under specific assumptions about the needed analog solution accuracy. Any tightening of specifications for accuracy quickly break the analog approach as the analog method has no way to give higher accuracy solutions.

With such accuracy constraints in mind, analog acceleration must only be useful in two types of architecture designs: the first is an analog-only approach to using analog accelerators, and the other is to use a hybrid analog-digital architecture where digital refines approximate analog solutions.

Conventional wisdom: analog architectures for real-world inputs and outputs

We can imagine using a purely analog architecture in some cyber-physical systems settings, where accuracy requirements are low throughout the system. Such devices take real-world low-precision sensor inputs and give low-precision actuator outputs that have effect on their physical surroundings. In such settings the high precision granted by digital computers is unnecessary, as the data that the algorithms operate on are imprecise and approximate. Moreover such deeply embedded devices have extremely constrained power and energy budgets. For these types of data input and workloads, energy-efficient analog computing is useful for giving approximate results, within tight power and energy constraints.

My viewpoint: digital refinement of approximate analog solutions

In this thesis I advocate for using analog acceleration in problems where approximations are extremely useful. Such an approach has also been advocated by Sarpeshkar, who writes that analog acceleration must be used in conjunction with a conventional digital system in order control for noise and get high precision [148, 147]. Cowan et al. has also demonstrated using analog approximate solutions for an ODE help speed up a digital ODE solver [40, 41].

In this thesis we demonstrate a more general framework for using analog approximations in precise digital solvers. Specifically Part IV explores using analog acceleration as an approximate solver for algebraic equations. In Chapter 10 I solve in an analog accelerator linear algebra problems, which are useful in PDE solvers (such as the multigrid method) that need only approximate linear algebra solutions. In Chapter 11 I solve in an analog accelerator nonlinear systems of equations, which are useful in conventional digital solvers that benefit from a good albeit imprecise initial solution seed.

3.3 Growing problem sizes: digital problem decomposition and analog emerging devices

As established in Section 2.1, analog accelerators directly map problem variables into circuit signals and hardware operators, and therefore can only handle portions of the overall problem at once. Again, area and problem size constraints rule out broad classes of analog applications. Many proposed analog problems and workloads make assumptions about the maximum problem size, and the analog method has no way to scale to larger problems.

Two approaches offer a path to scalable analog accelerator applications. One is to use emerging dense analog circuit devices and limit ourselves to problems that can use those devices. The other approach is to use a hybrid analog-digital architecture where digital breaks down large problems for acceleration in analog.

Conventional wisdom: analog devices for parallel matrix-vector multiplication

A prominent recent direction is in using emerging analog devices as wide single-instruction multiple-data (SIMD) operators, for parallel matrix-vector multiplication. While such an approach limits the applications to workloads where matrix-vector multiplication is the dominant kernel, such workloads include many types of Berkeley Dwarfs including dense matrix, structured grid, spectral methods, and N-body methods. Furthermore such approaches surpass existing SIMD CPU and GPU operations [178, 18, 105] because new analog circuit devices give highly parallel multiply accumulate operations.

The competing analog devices for matrix-vector multiplication include floating gate analog amplifiers and memristors. Both devices provide low-precision approximate coefficient-variable multiplication, and in both cases the analog accelerator calibrates coefficients just prior to computation.

Hasler estimates the computational energy efficiency of floating gate analog amplifier arrays at 1×10^{10} FLOPS/W. The case for floating gate analog amplifier arrays is that they can be realized in existing CMOS process technology; they use transistors as amplifiers and not as switches. The floating gates act as miniature capacitors that hold charge indefinitely and set the gain for the amplifiers. The floating gates give area-efficient calibration, raising the effective output precision of analog accelerators up to 12-bits [155, 63, 72, 73].

Shafiee demonstrated a memristor architecture for matrix-vector multiplication with a computational energy efficiency of 3.8×10^{11} FLOPS/W [79, 159]. Unlike transistors, memristors are fabricated alongside metal layers and are less mature.

These analog circuit devices are not mutually exclusive; both are viable approaches in the urgent search for new devices, for addressing the end of Dennard's scaling [46] and Moore's scaling in its present sense [137, 44, 54, 165, 38].

My viewpoint: digital divide-and-conquer for analog subproblems

In this thesis I advocate for using analog acceleration in problems where a conventional digital computer and existing algorithms can divide-and-conquer large problems. Traditionally divide-and-conquer algorithms are useful in digital computers to increase task-level and thread-level parallelism, but here they are useful for getting subproblems that fit in an analog accelerator. Specifically Chapter 8 explores how Monte-Carlo methods map problems into an ensemble of independent experiments, each of which we can do separately in analog accelerators. Chapter 10 explores using a multigrid method to divide-and-conquer PDEs into subproblems that fit in analog accelerators. Chapter 11 likewise explores using red-black Gauss-Seidel to divide-and-conquer nonlinear PDEs. In general this approach works where domain decomposition methods break PDEs into multiple domains, so subproblems on each can be solved independently [22, 100, 167, 61].

Chapter 4

History of and Related Work in Analog Co-Processing

4.1 History of analog co-processing

This section gives pointers to the immense amount of prior work in analog electronic computing.

Analog electronic computers were used in the 1950s and 1960s for scientific simulations, including problems in optimization, ODEs, and PDEs [91, 92, 85, 57, 58, 180, 25]. Starting in 1962 attention shifted to hybrid analog-digital computers, which combined analog and digital computers to provide capacious memory and ability to do discrete-time algorithms [103, 93, 11, 173, 30, 94, 102, 144, 120, 49]. 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*, a digital version of analog computers. DDAs were built and connected like analog computers, but they replaced analog integrators with digital counters. These digital counters were wired to have some simple behavior to realize differential equation solving methods. While DDAs shared with analog computers the types of problems they solve, the algorithms they run, and programming methods, DDAs differed from analog computers in that DDA variables were encoded in binary and evolved in discrete time [60, 170]. These designs faced difficulties in number dynamic range and scaling, which led to the development of extended resolution and floating-point variants of DDAs [123, 52, 136]. These area-intensive function units were used in a time-multiplexed fashion, previewing the development of modern floating-point pipelines [71].

4.2 Recent related work in analog co-processing

Computing using analog signals is resurgent in architecture research, due to challenges in IC scaling that limit the power dissipated by digital circuits [54, 165]. The vast majority of recent computer architecture research in analog computing and analog accelerators has been in neuromorphic computing. Nonetheless, there has been some recent related work in analog accelerators for scientific computation, which is the focus of this thesis.

The differences among the work done by various research groups have been in the design choices for the proposed analog accelerators. These design choices are:

1. How values are stored (how to build the soma)
2. How values are communicated (how to build the axon)
3. How multiplication is done (how to build the dendrite)
4. Topology of neuron connections
5. The target application
6. How the envisioned accelerator is integrated with a conventional host computer

Here I review recent related work along these dimensions.

Choice of value storage and communication

Researchers have been exploring encoding data as analog current and voltage for computation. The analog value representation is useful in neural network accelerators [98, 147, 101, 154, 163, 159, 110], and also in analog computers intended for differential equations [40, 41]. A notable example of using analog values to represent variables involves using electric fields to represent a whole vector of variables at once [128, 129, 127]. While using analog variables is the most novel approach and offers the most promise, it is also true that designing and building fully analog electronic circuits is challenging.

Because analog circuit design is difficult with existing design tools, various projects have explored ways to mimic analog signal encodings using easier-to-design conventional digital hardware. One alternative to a fully analog value representation is to use digital pulses or signal spikes to imitate analog values. Such a signal encoding is similar to how biological neurons communicate, and is used in neuromoprhic computing research [157, 125]. Another approach to mimic analog value encoding is to use pulse-width modulation to imitate analog current while using digital hardware [162, 121]. Yet another approach is to use continuous-time asynchronous digital hardware for computation [153].

Choice of multiplier implementation

A key difference among the various related work in analog computing is how the proposed hardware realizes analog multiplication. The main difference is in whether the hardware can multiply two *time-dependent* variables in the course of computation; or, in contrast, if the hardware can only multiply variables with a constant coefficient. The former, variable-variable multiplication, is a useful operation but has high hardware area costs. The latter, constant coefficient multiplication, is area efficient but restricts the types of problems the hardware can solve.

Variable-variable multiplication was a requirement for prior work in analog and hybrid computing from the mid-20th century. The requirement is due to the need for multiplying two time-dependent variables when solving nonlinear differential equations. Modern integrated circuits can multiply two variables using Gilbert cells, which allow us to tackle a broader class of problems, but are unfortunately more costly compared to constant coefficient multipliers I discuss in the next paragraph. The recent prototypes at Columbia University all use Gilbert cells for multiplication [40, 41, 66, 67].

Constant coefficient multiplication is all that is needed for many neural network problems, where the dominant math operation is matrix-vector multiplication. In that setting, the neural network synapse weights are already determined from training the network. Those synapse weights are matrix coefficients for the analog hardware multipliers. The hardware would configure and calibrate the coefficients before running the neural network, then multiply the matrix with incoming vectors during computation.

Recent work explores two ways to realize constant coefficient multiplication: using floating gate transistors [155, 63], and using programmable resistors [32] or memristors [79, 159]. The key difference between these two coefficient multiplication devices is in how chip fabricators make the devices. Floating gate transistors are made alongside other transistors in front-end-of-line processes, while memristors are made alongside metal layers in back-end-of-line processes [181]. These two approaches are area-efficient and have different tradeoffs in terms of the accuracy of the coefficients and how quickly they can be reprogrammed.

Choice of network topology and target applications

In neuromorphic computing, neural networks serve as pattern recognizers and classifiers for problem areas such as image processing and computer vision [98, 56, 55, 50, 110], natural language processing [155, 63, 55], and machine learning [101, 29, 31, 111, 159]. In all these settings, the dominant mathematical operation is matrix-vector multiplication. The applications of software and hardware neural networks are not limited to computer vision; neural networks have found use as general-purpose function approximations for floating-point workloads [28, 163]. A comprehensive survey of neuromorphic computing is compiled in [156].

I draw distinction between how this thesis envisions analog acceleration and analog neuromorphic computing. Most important, I 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 in this thesis is a *procedural* approach to solving problems: there is a predefined way to convert a scientific computation problem under study into an analog accelerator configuration.

The analog computation presented in this 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. In thesis and in the Columbia University prototype analog accelerator analog components are connected via a crossbar, allowing any topology, including loops, between components. In neural network terminology such topologies are recurrent neural networks [145] and Hopfield networks, and represent the most powerful class of networks.

Choice of granularity of accelerator design

Analog neural network accelerators support digital host computers at various granularities. Fine-grained accelerators propose to integrate analog neural network accelerators at the pipeline level. At this fine-grained level of integration, the analog accelerator acts as analog arithmetic operators. Examples of tightly integrated analog accelerators include Neuflow [56] and NPUs [163]. Coarser grained accelerators such as the dot-product engine [79] perform arithmetic operations (matrix-vector multiplication) on whole vectors.

Large scale neural network accelerators such as IBM's TrueNorth [55] and DaDianNao [31] accelerate the computation for whole neural networks, only interfacing with the digital host through main memory. Finally, analog neural network designs such as RedEye [110] work between analog sensors and the digital host computer. In that setting, the RedEye accelerator acts as a image feature detector and serves to reduce the amount of data going from sensors to the digital host.

Part II

Columbia University Prototype Analog Accelerator Architecture

I worked on two versions of prototype analog accelerator chips as part of a team at Columbia University. Thorough documentation of how to use the prototypes is in the latest version of the Columbia Hybrid Computer User's Guide [65]. See also the design documentation for the programming model, instruction set architecture, and microarchitecture organization [84]. Circuit design details of the prototypes are in recent papers by Ning Guo et al. [66, 67]. The prototypes are successors to an earlier design built by Glenn Cowan et al. [40, 41].

This part discusses how a conventional digital host computer interfaces with and uses the analog accelerator chips. I will discuss the programming, architecture interface, and microarchitectural organization of the prototype analog accelerator chips. This part then concludes with measurements of the analog operational characteristics of the analog chip components.

The architectural design and the analog operational characteristics will allow us to map differential equations onto the chip. The remaining parts of this thesis explores how the analog accelerator chip solves differential equations and algebraic equations.

Chapter 5

Analog Accelerator Programming & Architecture

Programming the analog accelerator involves work at several levels: 1. The user of the analog accelerator system has to decompose the problem down to a *numerical primitive*, such as a differential or an algebraic equation. 2. The conventional digital host computer then has to configure the analog accelerator to solve the numerical primitive. 3. The system has to handle operational concerns such as calibrating the analog units, starting and stopping the solution, gathering and returning data, and reporting if any errors happened. In this chapter I discuss these concerns while discussing the programming model and architecture design of the analog accelerator.

5.1 Analog accelerator numerical primitives programming

As I discussed in Section 7.2, scientists and engineers state many problems in simulating physical systems, optimization, and control theory as partial differential equations. Computational numerical methods solve these differential equations in various ways. Eventually the problems become one of two types: solving systems of ordinary differential equations, or solving systems of algebraic equations. Now, whether a particular PDE solving algorithm is ultimately solving the PDE by one way or the other depends on the granularity at which we're inspecting the algorithm. I will make clear in Part III and Part IV that we can solve ODEs by solving algebraic equations, and even vice versa. For now, I talk about how we program the analog accelerator, depending on whether we are

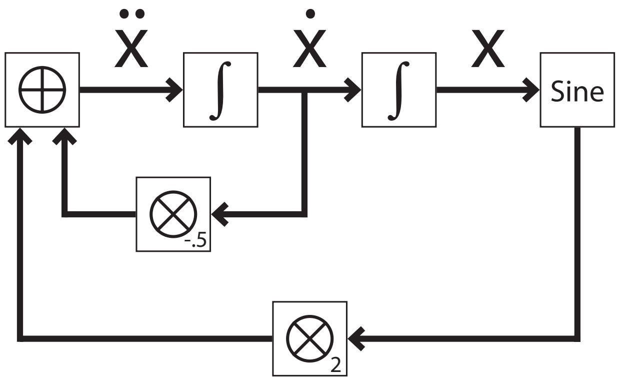


Figure 5.1: Analog accelerator block diagram for the ODE $\frac{d^2x}{dt^2} = -0.5\frac{dx}{dt} + 2\sin(x)$ (voltage mode).

solving differential equations or algebraic equations.

Programming analog accelerators for ordinary differential equations

In work I did with graduate students Matthew Maycock and Kenneth Harvey, we developed a compiler that converts ODE syntax to analog accelerator API code for solving that ODE. The process involves a front-end lexer and parser that converts ODE syntax into a graph that represents connections between multipliers and integrators. For example an ODE such as

$$\frac{d^2x}{dt^2} = -0.5\frac{dx}{dt} + 2\sin(x)$$

becomes the graph in Figure 5.1. The idealized graph at this stage, consisting of multipliers and integrators, has variables that evolve according to the given ODE. Using this graph we would like to generate analog accelerator API code for setting up and running the analog accelerator as an ODE solver.

To do so the compiler needs to take into account the hardware constraints of the analog accelerator. These constraints include the number of hardware resources such as integrators and multipliers, gain limits on multipliers, and dynamic range limits on the integrators. The graph also has to handle whether the analog accelerator uses *voltage signals* or *current signals*: if the analog circuit uses voltage signals (as in Figure 5.1), the circuit can easily copy values onto different branches, but needs an adder circuit to sum values. If the analog circuit uses current signals (as in Figure 5.2), the circuit needs a *fanout block* to copy values onto different branches, but can easily

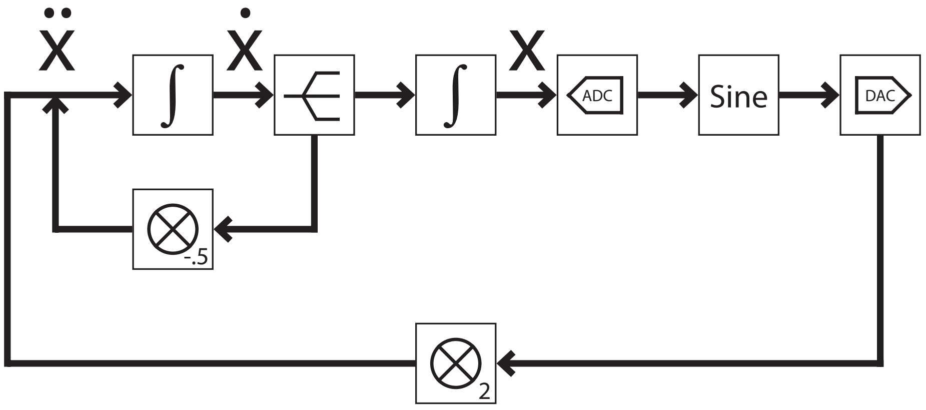


Figure 5.2: Analog accelerator block diagram for the ODE $\frac{d^2x}{dt^2} = -0.5\frac{dx}{dt} + 2\sin(x)$ (current mode).

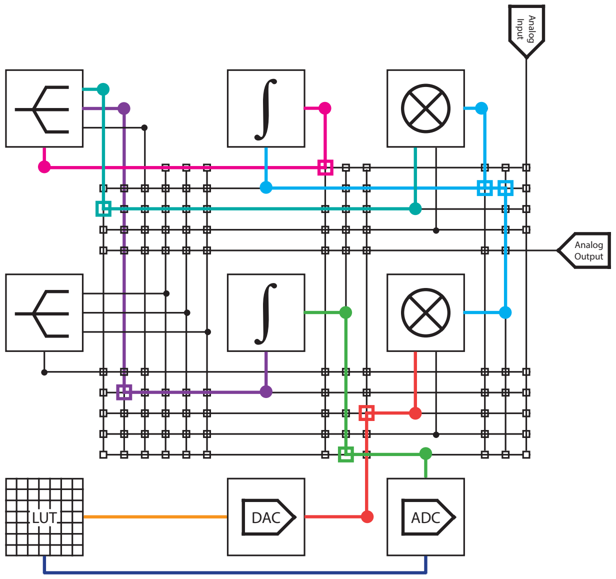


Figure 5.3: Programming the analog accelerator crossbar network and subcomponents to realize an ODE solver.

join wires to sum values.

Once the compiler has modified the graph of multipliers and integrators into a form that the analog accelerator can realize, the compiler generates analog accelerator API code to create the connections between blocks and to program parameters into the blocks (Figure 5.3). Additional connections feed analog values into analog-to-digital converters for measurement.

Challenges: While we had success with this compiler for converting simple ODEs into an analog accelerator, we faced numerous challenges that prevented this compiler from being useful:

1. Many different graphs can realize the same ODE. For example, we can symbolically refactor the ODE by hand, which in turn means multiplying by coefficients earlier or later in the circuit. The compiler should have an ability to pick among several graphs the best one for an ODE.
2. While different graphs ideally realize the same ODE, in analog hardware they may not result in the same solutions as some graphs have values that go out of bounds in some branches.
3. Practical issues in using the analog accelerator such as calibration and starting and stopping the analog accelerator are not well handled in the compiler framework.
4. When the ODE is large with many variables, the ability to split the ODE and solve the large problems as subproblems becomes important, and this compiler framework does not easily handle such issues.

Related work: Recent work by Achour et al. demonstrated a compiler for converting ODEs to analog computer connectivity graphs [2]. Their compiler focuses on differential equations in biological and chemical dynamical systems. At its core, it is a symbolic logic solver that converts between equivalent differential equation descriptions. It allows the user to get from an equation purely describing differential equations, to an equation describing hardware connectivity. While the compiler work by Achour et al. tackles the first two challenges listed above, the last two challenges remain.

Work by Pyle and Thangavel partially tackle the first three programming issues discussed above by using a genetic algorithm learning framework to try several different analog accelerator configurations [139, 166], in a narrower area of analog function generation.

Open problems: A high level language for analog computation must have functionality for measurement, calibration, and specifying testing and bring-up procedures. To get an analog solver to work, we have to connect some parts together, test and calibrate those subcomponents, and then connect more parts together. The connectivity between analog components and parameters of the components is just one part of the overall program. Languages for specifying ODEs and compilers that generate connectivity graphs for hardware components (such as our own and Achour's ODE compilers) may help with setting up the blocks, but doesn't help with the rest of the process for calibrating and getting them to work.

A high level language for analog computation must also express repetition and hierarchy, both necessary for large systems. In dynamical systems such as solving PDEs, the large system of ODEs is highly regular. The language needs some kind of support for telling the compiler that such structure exists, otherwise the compiler will take too much time to search and stumble into a configuration that works.

Programming analog accelerators for algebraic equations

I advocate for using analog accelerators as algebraic equation solvers. In this type of programming, the digital host computer calls the analog accelerator through a well-defined function call. The function call has the same signature as a software solver for linear or nonlinear system of equations. Doing so has a number of benefits compared to solving whole ODEs in an analog accelerator:

1. Programming the analog accelerator becomes a narrower, better defined task of programming mathematical expressions into hardware. For example, programming a linear algebra problem only involves programming the matrix coefficients and constants in the equation right hand side. Likewise, programming a nonlinear system of equations only involves programming the nonlinear polynomial expression for the nonlinear function, along with the Jacobian matrix. Both cases are simpler compared to programming a whole ODE graph.
2. The steps for bringing up, calibrating, configuring, running, and getting data from an analog accelerator become more clearly defined.
3. The computation results from the analog accelerator are reduced to a single vector of the variable values that satisfies the algebraic equation.

To set up an analog accelerator for algebraic equations, I program the accelerator using object-oriented C++, a style of programming that improves code reuse and minimizes errors when programming the analog accelerator. In our programming model, C++ classes represent mathematical expressions such as one nonlinear equation or one row of a Jacobian matrix. Each class exposes only the analog interfaces that need to be connected with other submodules. The classes also offer functions that change parameters such as initial conditions, coefficients, and constants. When a class object is instantiated, the instantiating program gives the object an allocation of analog hardware to physically implement the needed analog datapath. Then, the object constructor writes

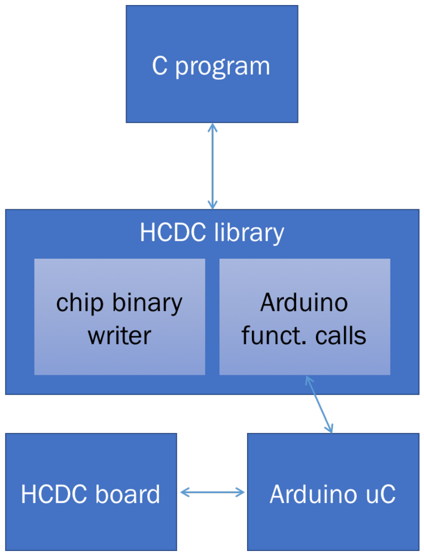


Figure 5.4: System diagram for user program, analog accelerator library, microcontroller, and analog accelerator chip.

a stream of bits to the analog accelerator setting up the object. Destroying the object likewise frees the analog resources to participate in other calculations. A concrete example will be given in Section 11.5.

5.2 Analog accelerator instruction set architecture

This section discusses how a digital host computer configures the prototype analog accelerator to solve differential or algebraic equations. The instruction set architecture interface for the analog accelerator handles operational concerns such as calibrating the analog units, starting and stopping the solution, gathering and returning data, and reporting if any errors happened. Table 5.1 summarizes the essential system calls and corresponding instructions for the analog accelerator; we walk through how to use the instructions in the steps below.

Calibration

Before we use the analog accelerator we first have to calibrate the analog circuitry. That is because analog circuits provide limited accuracy compared to binary ones, in which values are unambiguously interpreted as 0 or 1. Analog hardware on the other hand uses the full range of values.

Subtle variations in analog hardware due to process and temperature variation lead to undesirable variations in the computation result.

We identify three main sources of inaccuracy in analog hardware: gain error, offset error, and nonlinearity.

1. **Offset bias:** a constant additive shift in values,
2. **Gain error:** errors in either constant coefficient-variable or variable-variable multiplication,
3. **Nonlinearity:** the possibility that the DC transfer characteristic has a non-constant slope.

The amount 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, occurs when changes in inputs result in disproportionate changes in outputs. Typically this happens when analog values exceed the range in which the circuit's behavior is mostly linear, resulting in clipping of the output, akin to overflow of digital number representations. This type of error is kept under control via overflow exception detection, which we discuss later.

Configuration

Following calibration, the digital host computer software maps out the connections between analog units, along with settings of the units, and sends the configuration 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 improvement in our 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.

Observability

A few important node voltages, such as those setting the bias point for amplifiers, on the chip can be exposed to the chip pins. These can be checked from software to ensure these nodes are within tolerance, and certify the chip and configuration is free of major defects.

Instruction type	**Instruction**	**Parameters**	**Description**
Control	allZero		Reset all datapaths and shut off all function units
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	brkConn	source analog interface, destination analog interface	Break the 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	setDacSource	fuStruct * dac, bool external, bool lut0, bool lut1	Set DAC to respond to digital input from digital chip interface, or from lookup tables
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 the outputs of ADCs from chip to a memory location
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 5.1: Analog accelerator instruction set architecture.

Chapter 6

Analog Accelerator Microarchitecture & Characterization

In this chapter I briefly describe the Columbia University prototype analog accelerator chips, covering the necessary information to discuss its applications in the rest of this thesis. The second part of this chapter is a characterization of the analog subcomponents of the analog accelerator.

6.1 Analog accelerator physical prototype microarchitecture

In this section I describe the microarchitecture organization of our analog accelerator.

Our research group recently prototyped multiple versions of analog accelerator chips in 65nm CMOS technology [66, 67], shown in Figure 6.1. The physical prototypes validate the analog circuits' functionality and allows physical measurement of component area and energy. Additionally, the chips allow rapid prototyping of accelerator algorithms.

Microarchitecture hierarchical organization: The analog accelerator comprises four identical tiles connected with a global crossbar. Each tile contains analog functional units connected with a local crossbar. Each tile is then subdivided into four identical slices. At each slice's disposal are an analog input from off-chip, two multipliers, one integrator, two current-copying fanout blocks, and one analog output to off-chip (Figure 6.2). Two slices share use of an 8-bit ADC, an 8-bit DAC, and a nonlinear function lookup table (256-deep, 8-bit continuous-time SRAM [153]).

Storing and integrating variables: At the heart of analog computers are integrator function

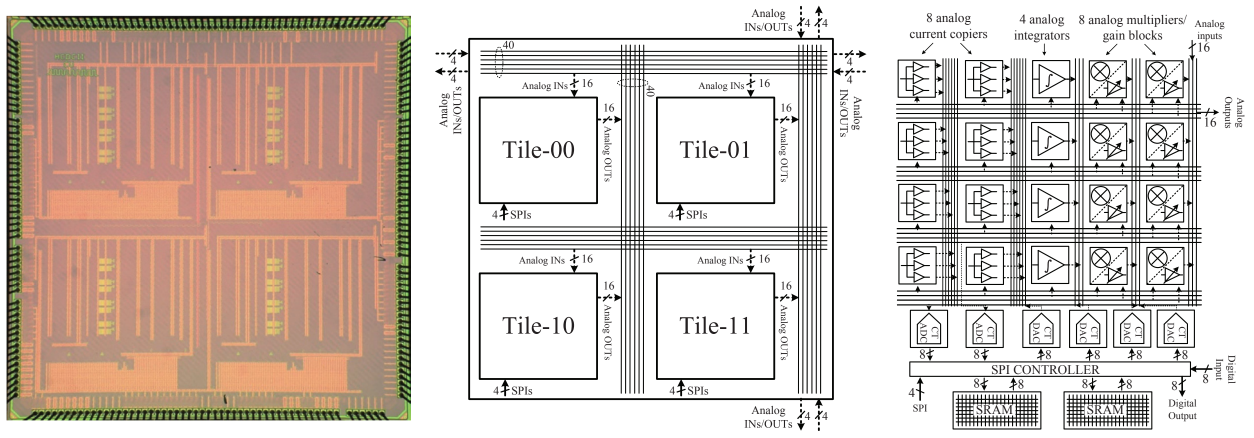


Figure 6.1: **Left:** Microphotograph of a Columbia University prototype analog accelerator chip, measuring $3.7mm \times 3.9mm$, fabricated in a TSMC 65nm process. **Center:** Architecture diagram of an analog accelerator designed to test scalable multi-chip integration and calibration of large analog accelerators. The chip contains four tiles, each an instance of the microarchitecture presented in [66, 67]. 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 differential and algebraic equations.

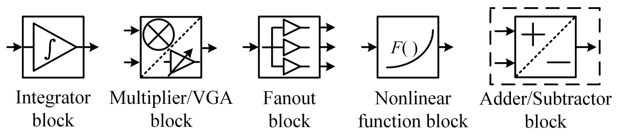


Figure 6.2: Analog accelerator subcomponents.

units: physically they are capacitors that output a continuous range of values. A collection of integrators can represent a vector of variables, $\vec{u}$. The input to integrators represent the time derivative of the variables, $\frac{\mathrm{d}\vec{u}}{\mathrm{d}t}$. These variables can be multiplied and summed, the results of which can be taken as output, or fed back to the inputs of integrators, resulting in the ordinary differential equation (ODE) $\frac{\mathrm{d}\vec{u}}{\mathrm{d}t} = f(\vec{u})$, where f is a polynomial function of $\vec{u}$.

Summing, copying, multiplying variables: In our analog accelerator, electrical currents represent problem variables, which may be added, multiplied, integrated, and subjected to arbitrary nonlinear functions. Fanout current mirrors allow the analog circuit to copy variables by replicating values onto different branches. To sum variables, currents are added together by joining branches. Gilbert cell multipliers allow variable-variable and constant-variable multiplication.

While constant-variable multiplication is enough for many linear ODEs and linear algebra problems, variable-variable multiplication is a requirement for nonlinear ODEs.

Nonlinear functions: The analog variables in the chip can be transformed by arbitrary non-linear functions, such as sine, cosine, and signum, with the help of continuous-time digital circuitry [153]. The circuit transfers the analog variable to a binary representation using a clockless ADC, so the variable can index into a clockless lookup table implementing the nonlinear function; the function output is restored to analog representation with a clockless digital-to-analog converter (DAC). The clock-free nature of this structure avoids aliasing problems associated with discrete time sampling of the data; moreover it minimizes the power needed to create arbitrary nonlinear functions.

Overflow detection on variables: 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.

Digital interface: The chip also includes an interface to receive commands from a conventional digital host processor. In the prototype these commands are received over an interface implementing an SPI protocol.

6.2 Analog accelerator analog subcomponent characterization

This section is a characterization of the analog components of the prototype analog accelerator chip. The goal is to certify that the chip can operate with input signals with frequency up to 20KHz, as designed. The components include the global crossbar, tile crossbar, the integrators, the fanout blocks, and the multipliers.

This work was done with supervision from colleagues at Sendyne Corp., during my internship there. I attained these measurements using two ways to generate inputs to the chip and measure outputs from the chip. The first way is to use DACs on the Arduino microcontroller for function generation and the microcontroller ADCs for measurement. Since the function generation and measurement must work at high frequency with precise timing, I used microcontroller timing interrupts and direct access to microcontroller registers to get these results. The second way is using

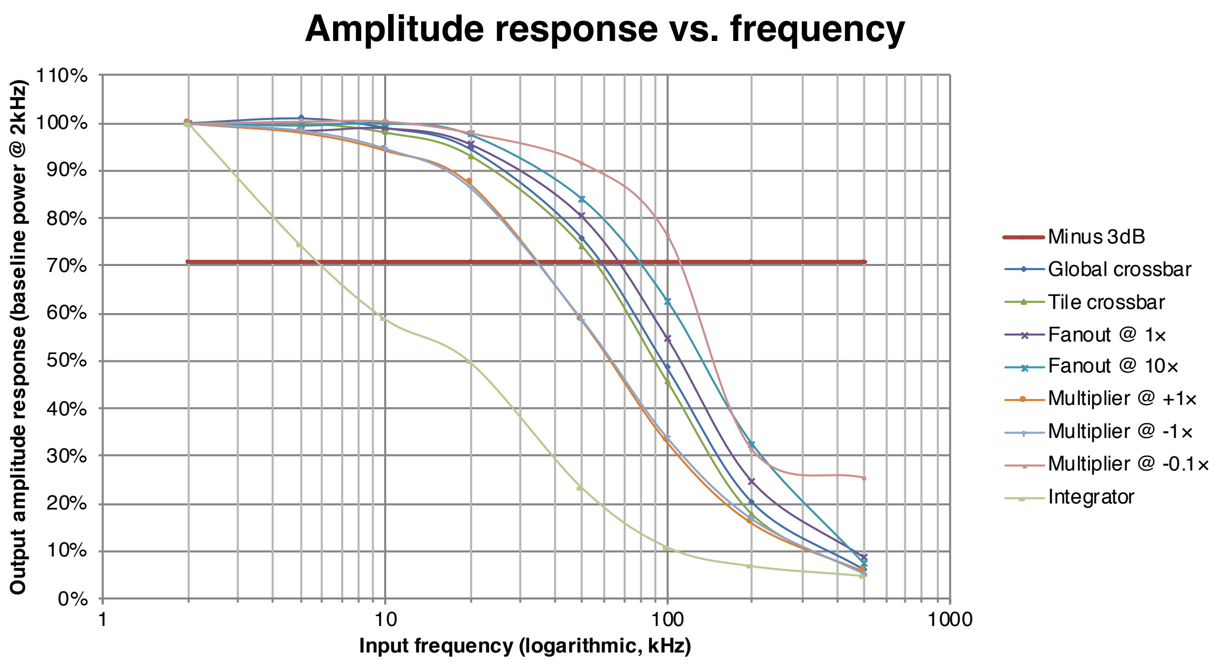


Figure 6.3: Analog component amplitude frequency response (function generator input, oscilloscope output).

a function generator to generate chip inputs and an oscilloscope to measure outputs. The results using the two measurement methods match well.

The first measurement is to find how much the analog components attenuate input signals. Figures 6.3 and 6.4 shows the output amplitude is within 85% accurate when the input is as high as 20KHz. Figure 6.5 is a zoomed in view of the same measurement.

The second measurement shown in Figure 6.6 is to find how much signal propagation time impacts accuracy, in the form of phase shift.

The third measurement in Table 6.1 measures how much noise is in each component's output signal. Section 8.2 will exploit this noise for solving stochastic differential equations. Altogether, the attenuation of the analog signals, phase shift, and noise all contribute to the error of the analog accelerator output.

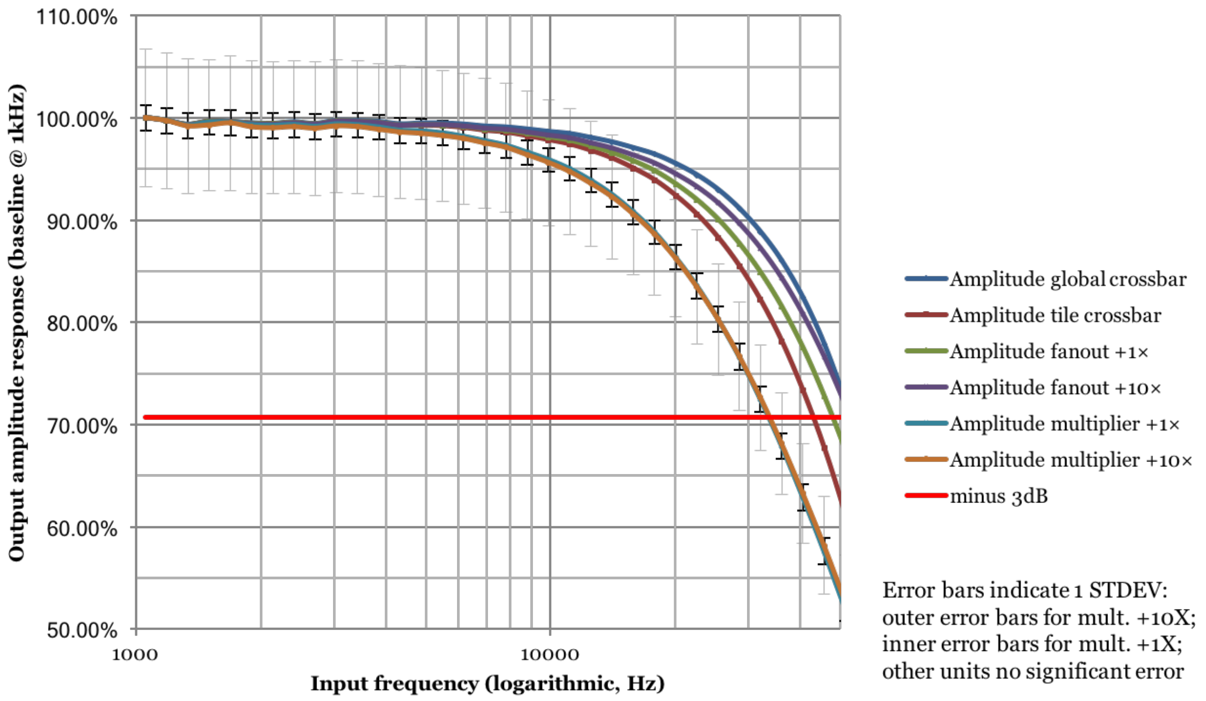


Figure 6.4: Analog component amplitude frequency response (DAC generated input, ADC acquired output). The error bars show the variance in multiplier gain, even after calibration and configuration. Measurements for integrators only available in Figure 6.3.

component	**noise RMS (mV)**
chip output only	0.857
global crossbar	3.179
tile crossbar	3.079
fanout 1X	3.022
fanout 10X	3.615
multiplier 1X	3.006
multiplier 0.5X	2.001
multiplier 10X	9.104

Table 6.1: Analog component noise measurement. These RMS noise figures are measured using a microcontroller DAC to generate a DC signal, then a microcontroller ADC measures the variation about the mean output voltage.

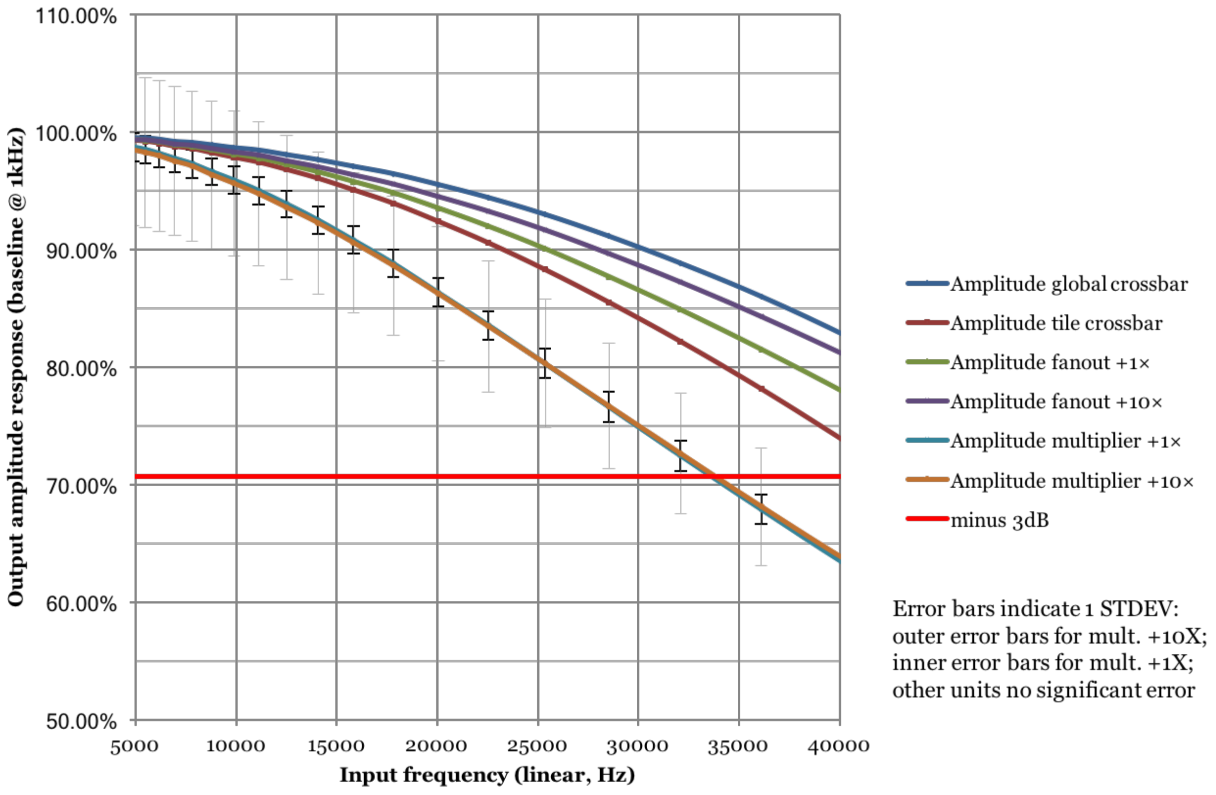


Figure 6.5: Analog component amplitude frequency response (zoomed in view). Measurements for integrators only available in Figure 6.3.

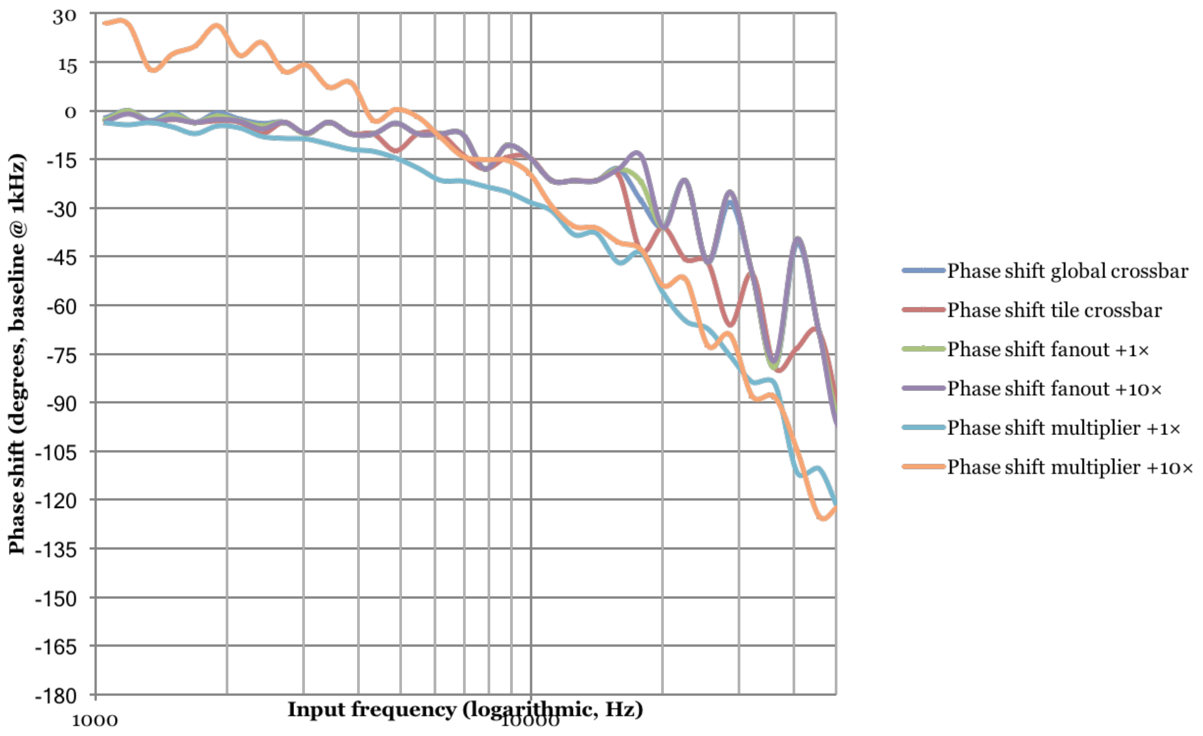


Figure 6.6: Analog component phase shift frequency response (DAC generated input, ADC acquired output). Phase shift estimated from timing of first zero crossing.

Part III

Analog-Digital Co-Processing for Solving Differential Equations

Chapter 7

Partial Differential Equations

This thesis focuses the application of analog-digital co-processing for applications in scientific computing. An important class of problems in this application domain is solving partial differential equations (PDEs) and ordinary differential equations (ODEs) [39, 6].

Solving PDEs is an increasingly important workload as they give natural and accurate models for the physical world. Researchers use PDEs to model water waves, combustion, and plasma physics, all of which belong to the area of computational fluid dynamics (CFD) and its extensions. Researchers also use nonlinear PDEs in solid mechanics for modeling structures as nonlinear springs in finite element models. Currently, these problems are tackled by computers ranging in size from supercomputer systems to mobile devices, usually on networked CPUs and GPUs [18, 178].

Though solving PDEs was once considered a supercomputing workload, they are now needed in autonomous mobile robots where energy budgets are limited. For example in optimal control theory, the optimal control path is the solution of Euler Lagrange PDEs. A mobile robot capable of solving these types of equations would be able to make more informed and optimal decisions navigating the physical world [141, 12, 99].

7.1 Taxonomy of PDEs

PDEs are classified according to their dimensionality, order, discriminant, nonlinearity, and the type of nonlinearity [122, 164]. The classification of a PDE according to these dimensions is important for finding the right way to solve the PDE.

Taxonomy: PDE dimensionality & order

The dimension of a PDE says how many time and space dimensions are involved. For example, the material derivative equation:

$$\frac{D\rho}{Dt} \equiv \frac{\partial \rho}{\partial t} + \vec{u} \cdot \nabla \rho = 0 \tag{7.1}$$

may be used in one-dimensional space, giving the following equation:

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} = 0$$

or it may be used in two-dimensional space, giving the following equation:

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + v\frac{\partial \rho}{\partial y} = 0 \tag{7.2}$$

These equations state that the density, ρ, at a point in space changes in time if and only if there is an opposite total change of density in its spatial neighbors.

The order of a PDE refers to the highest partial derivative in the equation. There are no second- or higher derivatives in the above examples, so they are first order equations. We will look at PDEs of first, second, and third orders. Some first-order PDEs include:

- **Gradient on scalar variable**

$$\nabla \rho = 0$$
$$\frac{\partial \rho}{\partial x}\vec{i} + \frac{\partial \rho}{\partial y}\vec{j} = 0$$

- **Divergence on vector variable**

$$\text{div}\vec{u} = 0$$
$$\nabla \cdot \vec{u} = 0$$
$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

- **The material derivative equation**

$$\begin{aligned}
\frac{\mathrm{D}\rho}{\mathrm{D}t} &= 0\\
\frac{\partial\rho}{\partial t} + \vec{u}\cdot\nabla\rho &= 0\\
\frac{\partial\rho}{\partial t} + u\frac{\partial\rho}{\partial x} + v\frac{\partial\rho}{\partial y} &= 0
\end{aligned}$$

- **The Navier-Stokes continuity equation (assuming incompressible fluid), also known as the advection equation, which describes conservation of mass**

$$\begin{aligned}
&\frac{\partial\rho}{\partial t} + \mathrm{div}(\rho\vec{u}) = 0\\
&\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\vec{u}) = 0\\
&\frac{\partial\rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0\\
&\frac{\partial\rho}{\partial t} + (\rho\frac{\partial u}{\partial x} + u\frac{\partial\rho}{\partial x}) + (\rho\frac{\partial v}{\partial y} + v\frac{\partial\rho}{\partial y}) = 0\\
&\frac{\partial\rho}{\partial t} + (u\frac{\partial\rho}{\partial x} + v\frac{\partial\rho}{\partial y}) + (\rho\frac{\partial u}{\partial x} + \rho\frac{\partial v}{\partial y}) = 0\\
&\frac{\partial\rho}{\partial t} + \vec{u}\cdot\nabla\rho + \rho\nabla\cdot\vec{u} = 0\\
&\frac{\mathrm{D}\rho}{\mathrm{D}t} + \rho\nabla\cdot\vec{u} = 0
\end{aligned}$$

Taxonomy: second-order PDE classification

PDEs of second order and higher can be classified as elliptic, parabolic, and hyperbolic according to their discriminant. The prototypical parabolic PDE is the heat equation, which describes a system that has diffusion. A two-dimensional heat equation PDE is expanded out in equivalent notations as follows:

$$\frac{\partial u}{\partial t} - \Delta u = 0 \tag{7.3}$$

$$\frac{\partial u}{\partial t} - \nabla^2 u = 0 \tag{7.4}$$

$$\frac{\partial u}{\partial t} - (\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = 0 \tag{7.5}$$

The second-order partial derivative makes this a second-order PDE, while the choice of signs on the terms makes it parabolic. Roughly speaking, parabolic PDEs have time-dependent behavior and evolve toward a steady state.

1. **Elliptic Helmholtz equation**

$$\Delta\rho + k^2\rho = f$$
$$\nabla^2\rho + k^2\rho = f$$
$$\frac{\partial^2\rho}{\partial x^2} + \frac{\partial^2\rho}{\partial y^2} + k^2\rho = f(x,y)$$

2. **Parabolic heat (diffusion) equation**

$$\begin{aligned}
\frac{\partial\rho}{\partial t} - c\Delta\rho &= 0\\
\frac{\partial\rho}{\partial t} - c\nabla^2\rho &= 0\\
\frac{\partial\rho}{\partial t} - c(\frac{\partial^2\rho}{\partial x^2} + \frac{\partial^2\rho}{\partial y^2}) &= 0
\end{aligned}$$

3. **Hyperbolic wave equation**

$$\begin{aligned}
\frac{\partial^2\rho}{\partial t^2} - c^2\Delta\rho &= 0\\
\frac{\partial^2\rho}{\partial t^2} - c^2\nabla^2\rho &= 0\\
\frac{\partial^2\rho}{\partial t^2} - c^2(\frac{\partial^2\rho}{\partial x^2} + \frac{\partial^2\rho}{\partial y^2}) &= 0
\end{aligned}$$

Taxonomy: semilinear, quasilinear, and fully-nonlinear

The example PDEs so far have been linear. For nonlinear PDEs, the analysis on the order of the equation and the analysis on its discriminant is the same as in the linear cases. Nonlinear PDEs are classified by how nonlinear they are. This is determined by whether and how the nonlinear terms appear in the highest-order partial derivatives.

Semilinear PDEs

Semilinear PDEs are the least nonlinear among nonlinear PDEs and most well-behaved. A PDE is semilinear if it is still linear in the highest order partial differential operator. In other words, the PDEs' coefficients on the highest order partial derivatives are independent of the main variable (e.g., u) and its partial derivatives. The nonlinear functions appear only in lower-order partial derivative terms. Examples of semilinear PDEs include:

- **Liouville equation:** a semilinear elliptic equation with a transcendental nonlinear term.

$$\begin{aligned}\Delta\rho + e^{\rho} &= 0\\ \nabla^2\rho + e^{\rho} &= 0\\ \frac{\partial^2\rho}{\partial x^2} + \frac{\partial^2\rho}{\partial y^2} + e^{\rho} &= 0\end{aligned}$$

- **Elliptic sine-Gordon equation:** a semilinear elliptic equation with a transcendental nonlinear term.

$$\begin{aligned}\Delta\rho - \sin(\rho) &= 0\\ \nabla^2\rho - \sin(\rho) &= 0\\ \frac{\partial^2\rho}{\partial x^2} + \frac{\partial^2\rho}{\partial y^2} - \sin(\rho) &= 0\end{aligned}$$

- **Reaction-diffusion equation:** adds a nonlinear term to the second-order diffusion equation.

ρ is the concentration of different species.

$$\begin{aligned}
\frac{\partial \rho}{\partial t} - \Delta\rho - \rho(1-\rho) &= 0\\
\frac{\partial \rho}{\partial t} - \nabla^2\rho - \rho(1-\rho) &= 0\\
\frac{\partial \rho}{\partial t} - \frac{\partial^2 \rho}{\partial x^2} - \rho(1-\rho) &= 0\\
\frac{\partial \rho}{\partial t} - \frac{\partial^2 \rho}{\partial x^2} - \frac{\partial^2 \rho}{\partial y^2} - \rho(1-\rho) &= 0
\end{aligned}$$

- **Korteweg-de Vries equation:** this classic third-order equation exhibits soliton behavior.

$$\frac{\partial \rho}{\partial t} + \frac{\partial^3 \rho}{\partial x^3} + \rho\frac{\partial \rho}{\partial x} = 0$$

Quasilinear PDEs

Quasilinear PDEs are more nonlinear than semilinear ones. A PDE is quasilinear if its coefficients on the highest order partial derivatives are linearly dependent on the main variable (e.g., u) and its partial derivatives. The nonlinear functions on u and its partial derivatives appear in lower-order partial derivative terms.

Quasilinear PDEs appear in fluid dynamics equations: they form the core of the Navier-Stokes momentum equations. An example of a quasilinear PDE is the inviscid Burgers' equation:

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u}\cdot\nabla)\vec{u} = \text{RHS} \tag{7.6}$$

$$\begin{cases}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} &= \text{RHS}\\
\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} &= \text{RHS}
\end{cases} \tag{7.7}$$

This equation is similar to Equation 7.2, the material derivative equation, except here the problem variable is the velocity field vector $\vec{u}$, instead of the density scalar variable ρ.

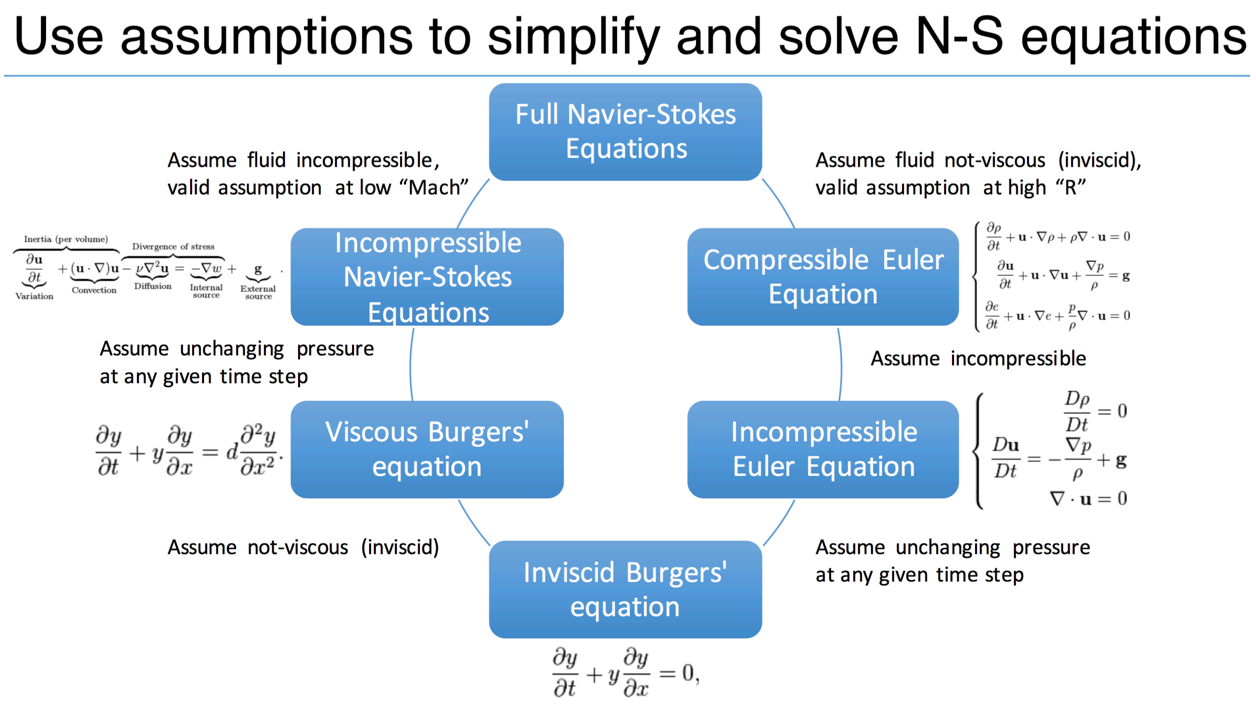

Use assumptions to simplify and solve N-S equations
Full Navier-Stokes Equations
Assume fluid incompressible, valid assumption at low “Mach”
Assume fluid not-viscous (inviscid), valid assumption at high “R”
$\overbrace{\underbrace{\frac{\partial \mathbf{u}}{\partial t}}_{\text{Variation}} + \underbrace{(\mathbf{u}\cdot\nabla)\mathbf{u}}_{\text{Convection}}}^{\text{Inertia (per volume)}} \overbrace{- \underbrace{\nu\nabla^2\mathbf{u}}_{\text{Diffusion}} = \underbrace{-\nabla w}_{\text{Internal source}}}^{\text{Divergence of stress}} + \underbrace{\mathbf{g}}_{\text{External source}}.$
Incompressible Navier-Stokes Equations
Compressible Euler Equation
$\begin{cases} \frac{\partial \rho}{\partial t} + \mathbf{u}\cdot\nabla\rho + \rho\nabla\cdot\mathbf{u} = 0 \\ \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} + \frac{\nabla p}{\rho} = \mathbf{g} \\ \frac{\partial e}{\partial t} + \mathbf{u}\cdot\nabla e + \frac{p}{\rho}\nabla\cdot\mathbf{u} = 0 \end{cases}$
Assume unchanging pressure at any given time step
Assume incompressible
$\frac{\partial y}{\partial t} + y\frac{\partial y}{\partial x} = d\frac{\partial^2 y}{\partial x^2}.$
Viscous Burgers' equation
Incompressible Euler Equation
$\begin{cases} \frac{D\rho}{Dt} = 0 \\ \frac{D\mathbf{u}}{Dt} = -\frac{\nabla p}{\rho} + \mathbf{g} \\ \nabla\cdot\mathbf{u} = 0 \end{cases}$
Assume not-viscous (inviscid)
Assume unchanging pressure at any given time step
Inviscid Burgers' equation
$\frac{\partial y}{\partial t} + y\frac{\partial y}{\partial x} = 0,$

Figure 7.1: Relationship between simplifications of the Navier-Stokes momentum equations.

Fully nonlinear PDEs

A PDE is fully nonlinear if its coefficients on the highest order partial derivatives are nonlinearly dependent on u and its partial derivatives. An example is the Eikonal equation from optics:

$$(\frac{\partial u}{\partial x})^2 + (\frac{\partial u}{\partial y})^2 = c^2$$

Navier-Stokes equations

After our short primer on nonlinear PDEs, we now see that these simple PDE variants lead to PDEs used in physical models. An important example of nonlinear PDEs are the Navier-Stokes equations in fluid dynamics, which we introduce here to motivate the nonlinear PDEs we explore in detail in the rest of this thesis. The full set of equations results in a large nonlinear set of equations which need to be solved simultaneously. Navier-Stokes equations have been solved using GPUs [178] and with cellular neural networks [104].

In practice the equations can be simplified depending on assumptions about the flow. When heat transfer is not included in the model, the conservation of energy equations are ignored, and we only need to solve for the conservation of mass and momentum across a flow field. The **conservation of mass** is described by the continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0$$

So long as the dynamic range of speeds in the system is not large (low Mach number), the fluid is assumed incompressible. The incompressibility of the fluid is represented mathematically by saying the divergence of $\vec{u}$ is zero:

$$\nabla \cdot \vec{u} = 0$$

Substituting the above equation into the continuity equation gives us another way to state incompressibility in terms of Equation 7.1, the material derivative equation:

$$\frac{D\rho}{Dt} \equiv \frac{\partial \rho}{\partial t} + \vec{u} \cdot \nabla \rho = 0$$

When incompressibility is assumed the **conservation of momentum** equation has a relatively simple form:

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

With these assumptions, the equation consists of a nonlinear advection/convection component, a linear diffusion component, and a nonlinear scalar pressure component.

7.2 Solution steps for PDEs

Solution steps: space discretization

Typically PDEs describe state variables as continuous functions of both time and space, while ODEs state variables as functions of time or space. As shown in Figure 7.2, PDEs are classified as time-independent or time-dependent equations.

In both conventional digital algorithms, and in the analog acceleration techniques we explore, state variables are discretized in space into a set of discrete variables that are functions of time. Major space discretization methods include:

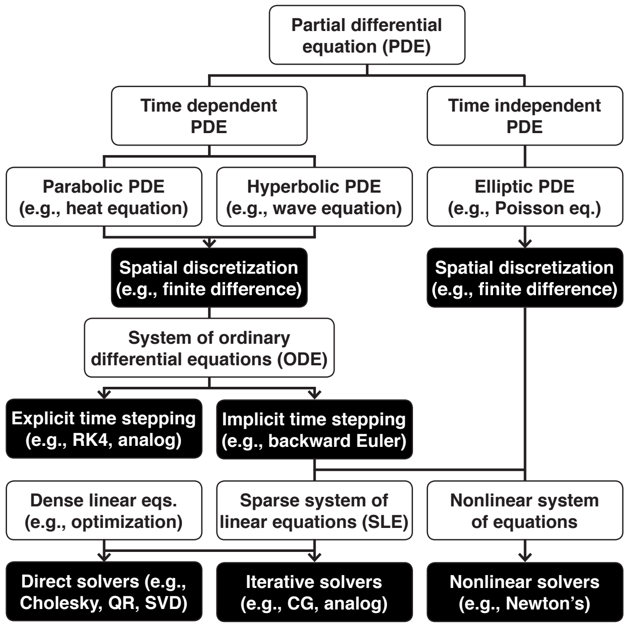

Partial differential equation (PDE)
Time dependent PDE
Time independent PDE
Parabolic PDE (e.g., heat equation)
Hyperbolic PDE (e.g., wave equation)
Elliptic PDE (e.g., Poisson eq.)
Spatial discretization (e.g., finite difference)
Spatial discretization (e.g., finite difference)
System of ordinary differential 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 7.2: 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. Analog accelerators assists these problems from several directions.

1. Finite difference
2. Finite volume
3. Finite element
4. Spectral method[1]

Here we limit discussion to finite difference and finite volume methods.

In most finite difference and finite volume approaches, time-dependent PDEs become systems of ODEs after spatial discretization; for example, the canonical parabolic heat equation becomes a convergent system of ODEs, and the canonical hyperbolic wave equation becomes an oscillating one. We can focus on studying solving ODEs after we have discretized PDEs in space.

Solution steps: time stepping

ODE solvers are split between explicit and implicit time-stepping methods; both methods have mappings onto analog accelerators. Different time stepping methods are summarized in Table 7.1.

[1]Recent work in optical analog computing focuses on performing the Fourier transform.

Explicit / implicit	**Timestepping order**	**Target problems**	**Possible methods**
Explicit methods	First-order methods	ODEs	Forward Euler
		parabolic PDEs	FTCS (forward time central space)
		hyperbolic PDEs	Lax-Friedrichs, Upwind
	Second-order methods	ODEs	Heun's method, Leapfrog
		hyperbolic	Lax-Wendroff, MacCormack
	Higher-order methods	ODEs	Explicit RK, Adams-Bashforth
Implicit methods	First-order methods	ODEs	Backward Euler
		parabolic PDEs	BTCS (backward time central space)
	Second-order methods	ODEs	Trapezoidal rule
		parabolic PDEs	Crank-Nicolson, ADI
	Higher-order methods	ODEs	Implicit RK, Adams-Moulton

Table 7.1: Comparison of finite difference time stepping methods.

Explicit time stepping approximates the time derivative of u using the present and past guesses of u and increments u one step at a time. Explicit methods are suitable for some hyperbolic PDEs. All practical discretized PDEs result in systems of ODEs that are *stiff*, meaning that they force explicit solvers to take many small time steps to solve and therefore are computationally inefficient. Furthermore, explicit methods, whether discrete time or continuous time, lack a notion of error checking: small errors accumulate and the solution drifts away from the true solution. In order to increase the time stepping step size, implicit methods are often used.

Implicit time stepping, on the other hand, solves a system of linear equations to determine the next state u, enforcing the next state is in agreement with the system's partial derivatives with respect to time. Since the next u is unknown, we need to solve a system of algebraic equations. Implicit solvers effectively solve stiff systems; furthermore, the existence of efficient linear algebra solvers have led many PDEs and ODEs to be solved using implicit solvers.

7.3 Analog-digital co-processing for PDEs

In this section I talk about various design options for how PDEs are solved on either digital or analog hardware.

The fully continuous mathematical description of PDEs go through various types of discretization. In a fully digital, conventional approach, space, time, and variables are all discrete. On the other hand, fully analog approaches could include physical models such as wind tunnels, or continuous-space extended analog computers, where space, time, and variables are continuous [128, 129, 127]. Between the two extremes is a continuum of techniques.

	DE types	**Problem abstraction**	**Programming model**	**Analog-digital interaction**	**Relevant microarchitectural features**
[81]	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
[82, 83]	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; implementation in 65nm CMOS
[66, 67]	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	
[40, 41]	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; implementation in 250nm CMOS

Table 7.2: Summary of recent work in physically prototyped analog accelerators for differential equations.

Analog computers of the mid-20th century usually discretize space while keeping time and variables continuous. The analog computers of that era acted as explicit integrators for ODEs, which then supported PDE solving techniques such as method of lines and shooting methods [91, 93, 173]. Recent work in analog computing using integrated circuits has also explored this approach to solve linear parabolic PDEs [40, 41] and nonlinear ODEs [66, 67].

Following this logic, it is tempting to move toward a fully analog model of computation for modern workloads. But such a model faces two important challenges. First, analog accelerator hardware has high silicon area costs. Second, analog acceleration provides only limited accuracy. Because of these two important limiting factors, we must apply analog acceleration in a broader digital, discrete-time framework, using the digital computer for its high-precision operations and dense memory.

In this work, the solving methods have the PDEs discretized in space and time by the digital host. Part III uses the analog accelerator in the same way as prior work, for ODE solving techniques such as method of lines and shooting methods.

However, modern scientific computation is founded on algebraic equations, not ODEs. In effort to adapt analog acceleration to conventional digital architectures, we explored using analog acceleration for linear algebra [82, 83] and nonlinear algebra [81]. Starting in Part IV, we use the continuous time feature of our analog accelerator in a seemingly limited sense, to do steepest descent or the continuous Newton's method for algebraic equations. As I will show, even limited applications of continuous-time computation are fruitful, allowing us to replace temperamental discrete-time algorithms with effective algorithms otherwise impossible in discrete-time hardware.

Chapter 8

Analog Co-Processing for Stochastic Differential Equations

In this chapter I explore using our analog accelerator for solving stochastic differential equations (SDEs)[1]. SDEs are a combination of a deterministic model in the form of a differential equation, and a stochastic noise component in the model. Typically scientists find the deterministic part analytically while the stochastic part is an attempt to capture effects not accounted for in the model.

The motivation for using an analog accelerator for this class of problem is twofold:

1. **The analog accelerator excels at integrating differential equations, the task that dominates the solving time for SDEs.** Numerical methods for SDEs use Monte Carlo techniques, which entails running many parallel and independent solutions of the differential equation, each subject to a different stochastic noise input [77, 174, 151, 51]. Because the solver needs to solve a statistically significant ensemble of solutions, having a more efficient method for solving differential equations would be advantageous.
2. **In an analog solution to an SDE, the analog circuit can use analog noise from natural sources for the stochastic input to the equation.** On the other hand a numerical method running on a conventional digital computer needs to dedicate some time

[1] I did the work in this chapter during my time at Sendyne Corp.; they have permitted me to include this writeup in this dissertation.

to create pseudorandom numbers, and then reshape the distribution of those numbers to fit the distribution needed by the SDE.

8.1 The Black-Scholes stochastic differential equation

The famous Black-Scholes equation refers to two different stochastic differential equations: the first and simpler equation is the Black-Scholes stochastic ordinary differential equation for modeling stock prices, and the second more complex equation is the Black-Scholes-Merton partial differential equation for finding prices for options.

This chapter focuses on solving the Black-Scholes stochastic ODE, the solutions for which are needed in the PDE model for pricing options.

The Black-Scholes stochastic ODE has the form:

$$\begin{aligned} dX(t) &= \lambda X(t)dt + \sigma X(t)dW(t) \\ X(0) &= X_0 \end{aligned} \tag{8.1}$$

where $X(t)$ is the price of a stock as a function of time, X_0 is the initial price, λ is a parameter called drift, σ is a parameter called volatility, and $W(t)$ is the standard Wiener process, which I will define in Section 8.3.

Intuitively, the Black-Scholes stochastic ODE captures known effects and unknown effects on the price of a stock. In the long run, we can deterministically model the price of a stock as an exponential growth or decay process. In the short run, the fluctuations in the price of a stock are unpredictable, but are proportional to the current price of the stock.

The Black-Scholes stochastic PDE is useful for setting the price of a financial derivative or option, optimized so that it minimizes the risk due to the fact we cannot predict stock prices. The type of deriviative or option we are interested in pricing decides how we would define and solve the Black-Scholes stochastic PDE. Financial regulations in different markets permit different types of options.

Some types of options, such as barrier, American, and Asian options, are *path-dependent*, meaning the price of the option depends on the price of its underlying stock as a function of time. Barrier options for example must have a positive stock price at all times, otherwise the option expires. When

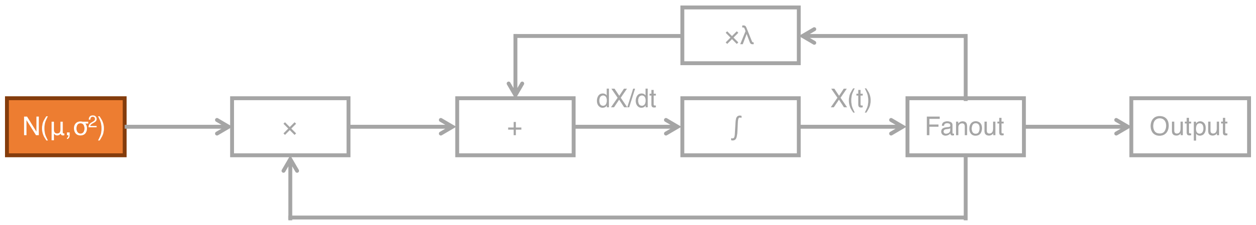


Figure 8.1: Analog accelerator setup for solving the Black-Scholes equation, with the Gaussian white noise source highlighted.

we set the price of these types of options we must use Monte-Carlo simulation to generate many price trajectories for the underlying stock. The work in this chapter envisions using the analog accelerator to assist in this type of calculation.

On the other hand more basic types of options such as European options are *path-independent*, meaning the option prices do not depend on actual stock price trajectories. In pricing these options we can use the closed form solution for the Black-Scholes stochastic ODE for stocks, which takes the form of the lognormal distribution:

$$X(t) = X_0 e^{(\lambda - \frac{\sigma^2}{2})t + \sigma W(t)} \tag{8.2}$$

We will be using this closed form solution to check the accuracy of the analog accelerator solution.

8.2 Analog Black-Scholes bringup: Gaussian white noise

The type of noise plays a major role in solving SDEs. The Black-Scholes stochastic ODE uses Gaussian white noise as its stochastic input. An analog accelerator solving the Black-Scholes stochastic ODE can obtain this noise from several sources (Figure 8.1): One option is to use Johnson-Nyquist thermal noise from resistors as a source of white noise [148, 114]. The analog noise may need some additional processing such as amplification and bandpass filtering to be useful in an analog SDE solver. Another option is to revert to using pseudorandom digital codes to generate analog noise. In this chapter I discuss the method and merits of using each of these noise sources.

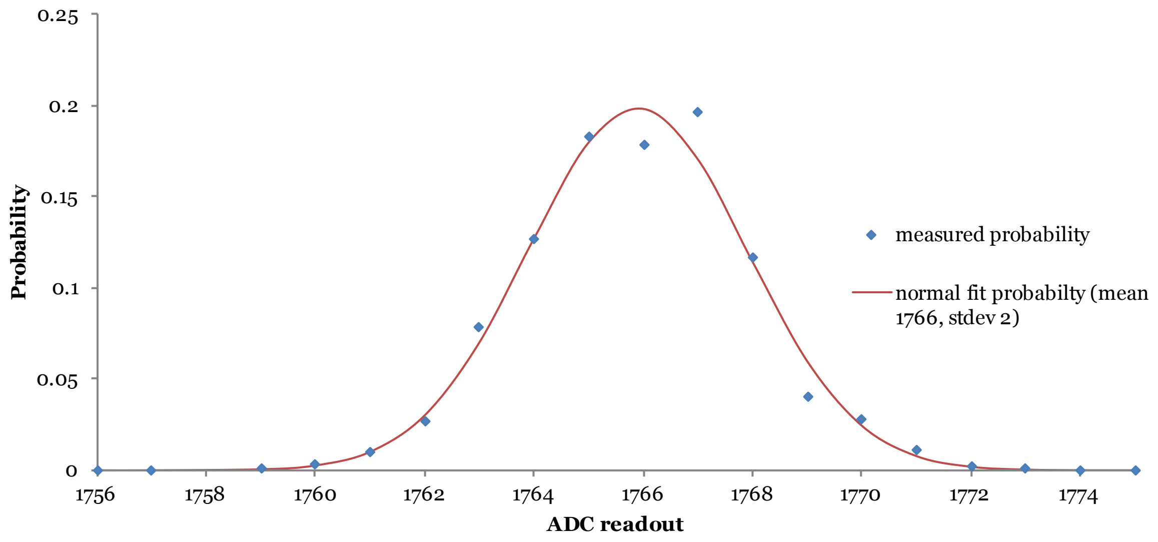


Figure 8.2: The noise I feed to the analog accelerator is normally distributed. I use microcontroller ADCs to measure the number of observations above and below the mean voltage, μ, due to noise, over 50K data points. Some ADC readouts have excessive or deficient counts due to ADC nonlinearity; no reason to believe this is a property of the noise itself.

Analog noise

Generating high-amplitude, high-power analog noise is actually a bit tricky. Dedicated programmable analog noise generators are available for industrial uses[2]. At the same time, our prototype analog accelerator also naturally generates analog noise as a side effect of rescaling signals using resistor ladders (see Table 6.1). Here I discuss practical considerations using the analog accelerator's noise sources as random input for solving SDEs.

The analog noise needs to satisfy two properties for it to be suitable in solving SDEs. From a time-domain perspective, the noise must be Gaussian normally distributed. From a frequency-domain perspective, the noise must have constant power spectral density. I consider these two properties below.

Analog noise: Gaussian distribution

Figures 8.2 and 8.3 show the analog accelerator noise is Gaussian normally distributed around a mean, μ. The setup for generating this noise has the microcontroller DACs generating some constant DC signal, which passes through a board-level scaling circuit (consisting of a resistor

[2]NoiseCom, noisecom.com

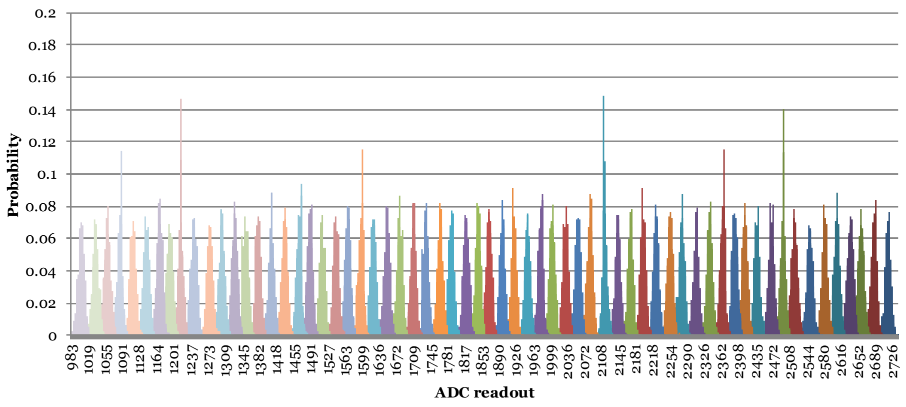


Figure 8.3: Microcontroller DACs can control the mean, μ, of the input Gaussian noise while the analog accelerator multiplier controls variance, σ^2. In this plot I swept the microcontroller DAC codes over its output range to change noise mean. I use the chip multipliers to amplify the noise to increase the noise variance. Each Gaussian distribution is plotted separately. ADC nonlinearities result in excess of counts at some values, but the peaks are consistent across different runs.

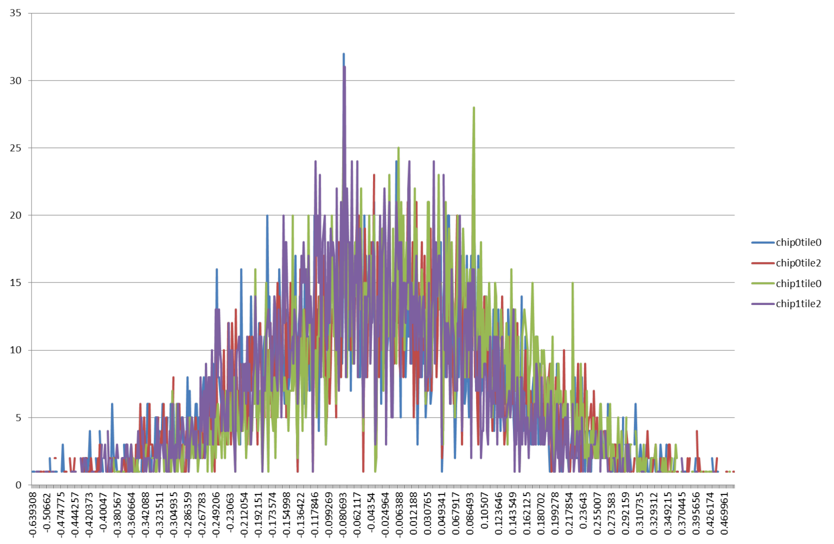


Figure 8.4: We can calibrate multiple copies of noise to have identical distributions, even as the noise sources are mutually independent. The analog accelerator can calibrate the noise standard deviation, σ to within 5%.

ladder), that then flows into an analog input channel of the analog accelerator chip. The current flowing through resistors generates Johnson-Nyquist thermal noise. The DAC codes control the mean of the Gaussian white noise, while multipliers inside the analog accelerator chip control the variance, σ^2. With control over these two variables, we can calibrate several independent noise sources to have the same distribution, as shown in Figure 8.4.

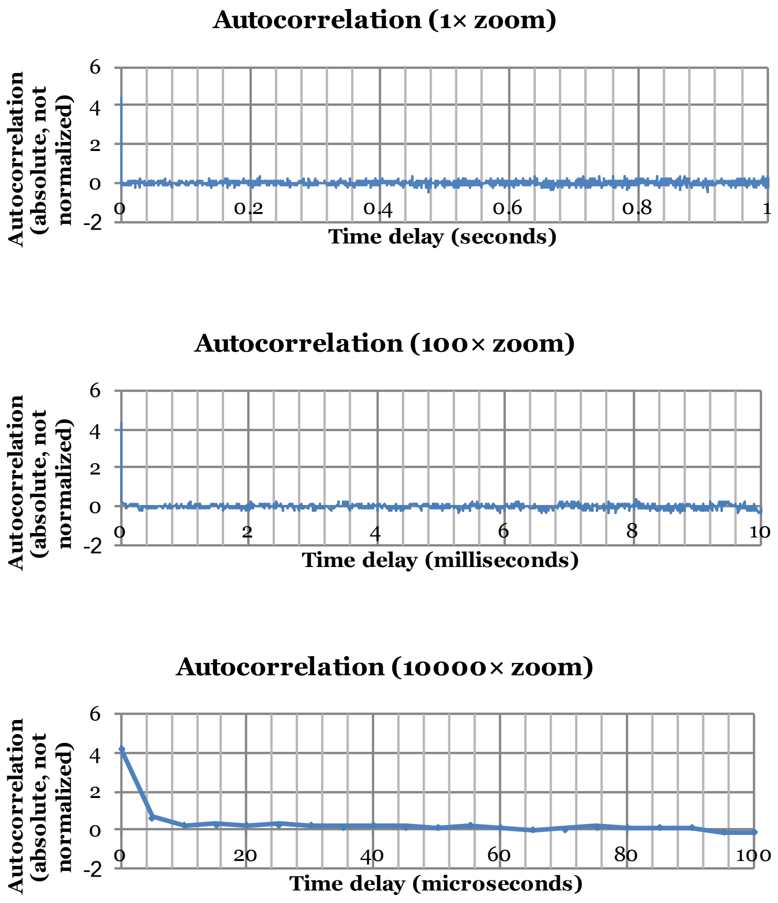


Figure 8.5: Analog accelerator input noise autocorrelation. The autocorrelation period of the analog input noise is less than 10 microseconds, equivalent to white noise up to 100 KHz (beyond the analog accelerator's design frequency). These measurements were done with microcontroller ADCs, operating at 200 kS/s, over 3200 samples, and matches a FFT measurement done with an oscilloscope.

Analog noise: constant power spectral density

The second property the analog noise needs for use in solving SDEs is the noise should be white noise. White noise refers to noise with constant power spectral density. We can check the frequency-domain plot using an oscilloscope's spectrum analyzer to see if an analog noise source is white noise.

We can also check if an analog noise is white noise by checking if its autocorrelation plot is a Dirac delta function, as shown in Figure 8.5. A Dirac delta autocorrelation plot indicates that a signal is not at all correlated with delayed versions of itself, no matter how much the signal is delayed. Plainly speaking, it means knowing the value of a signal at any moment gives us no

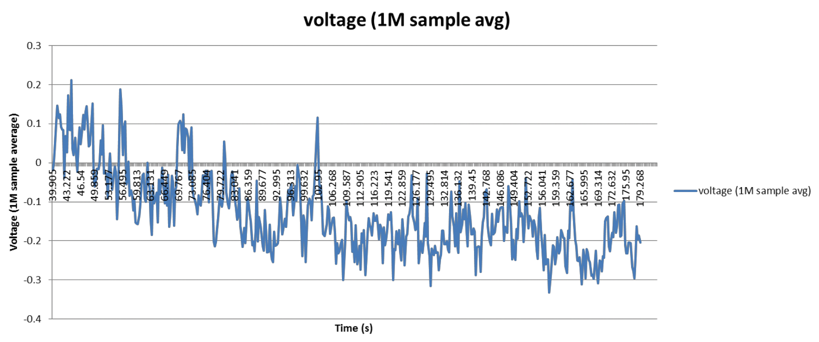


Figure 8.6: Analog noise DC drift. The mean of the noise drifts significantly over the time scale of seconds to minutes, even after calibration.

	Low frequency limitations	**High frequency limitations**
Black-Scholes stochastic ODE model limitations	Market prevailing interest rate measurable and modeled deterministically	Trading day or market clock ticket is minimum meaningful time interval
Digital numerical method limitations	Long-run model parameters are deterministic	Numerical integration and models have non-zero time step size
Analog electronic circuit limitations	Environmental variables, such as temperature and RF interference, introduce DC drift	Analog components, such as integrators and ADCs, have finite bandwidth due to parasitic capacitance
White noise limitations	At lowest frequencies flicker (1/f, pink) noise dominates	At highest frequencies shot noise (Poisson process) dominates

Table 8.1: Rationale why low-frequency and high-frequency components of noise are both unattainable and unneeded for solving SDEs. In our experiments using band-limited noise is sufficient.

information about the value of the noise at any other point in time. The equivalence between constant power spectral density and Dirac delta autocorrelation is a result of the Wiener-Khinchin theorem [138].

Filtered analog noise

I faced a significant challenge in using the analog accelerator noise source for solving SDEs in the form of DC drift. Here I explore using a bandpass filter to limit the spectrum of the analog white noise, in order to use the simple noise source for solving SDEs.

While the calibrated analog accelerator noise source appeared close to ideal on first glance, Figure 8.6 shows the mean of the noise drifts significantly over the course of minutes. While white noise does have low-frequency components that cause the mean to drift, this drift was coming from

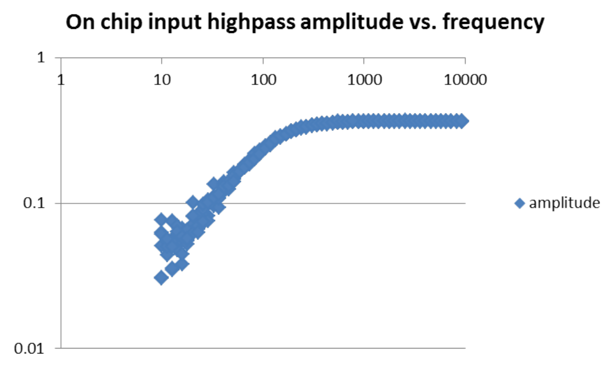


Figure 8.7: Analog frequency response for on-chip high-pass filter. Ideally we want to shift the low frequency pole as low as possible because the DC offset drift is on the scale of seconds to minutes. Attenuating the high-pass filter feedback loop shifts the low frequency pole lower, subject to the limit that the feedback signal cannot be weaker than the noise floor.

environmental effects such as temperature. Such a large, low frequency, and uncontrollable noise component overwhelms the white noise signal at low frequencies and cannot be used for solving SDEs.

A few techniques can eliminate DC mean drift. These include:

1. **Subtract two independent noise sources both subject to the same DC drift.** This is in fact the approach Intel takes for generating true random numbers in their microarchitectures [89]. In our case however some of the DC drift was itself independent for different channels, so subtracting noise sources did not cancel the DC drift.
2. **Calibrate frequently to eliminate the DC drift.** While this works, calibration takes too much time for this to be practical.
3. **Use a high-pass filter to eliminate DC components.** This is the technique I explored most extensively, though limitations on this technique eventually motivated using a digital pseudorandom number source instead.

While SDEs such as the Black-Scholes stochastic ODE specify the noise source should be white noise, in practice both the mathematical model and the numerical methods for solving SDEs do not need to use perfect white noise. Likewise, our analog circuit approach and the noise sources the analog accelerator has access to do not support perfect white noise either. I summarize in Table 8.1 the rationale for why band-limited noise is sufficient for solving SDEs.

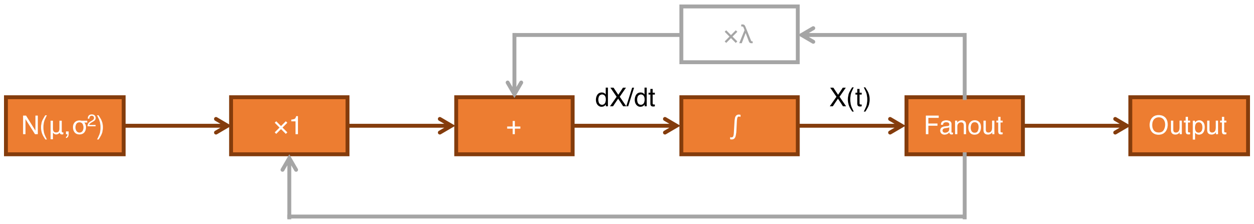


Figure 8.8: Standard Wiener process / Brownian motion.

Figure 8.7 shows the amplitude response of a high-pass filter we can build in the analog accelerator to filter out the DC drift in the analog noise source. The resulting filtered noise is still Gaussian normally distributed as required. From a frequency domain perspective, its autocorrelation function is a Dirac delta around the origin, implying that it is white noise at high frequencies. At the low frequency end, the DC components are cut out, so there is no longer any drift in the noise mean.

Digital noise

For the purposes of the experiments in this chapter, I used noise generated digitally by feeding a DAC with a pseudorandom number sequence. I do this by drawing digital uniformly distributed pseudorandom numbers in the digital host computer. The Box-Muller transform allows us to convert the uniformly distributed number sequence to a Gaussian normally distributed one [138, 151, 174].

Using digitally generated noise avoids downsides of using purely analog noise such as low amplitude and DC drift due to environmental variables, but comes at a cost of more work in the conventional digital host computer. An ideal analog accelerator for stochastic differential equations should have analog circuits to create high amplitude, low-drift analog noise.

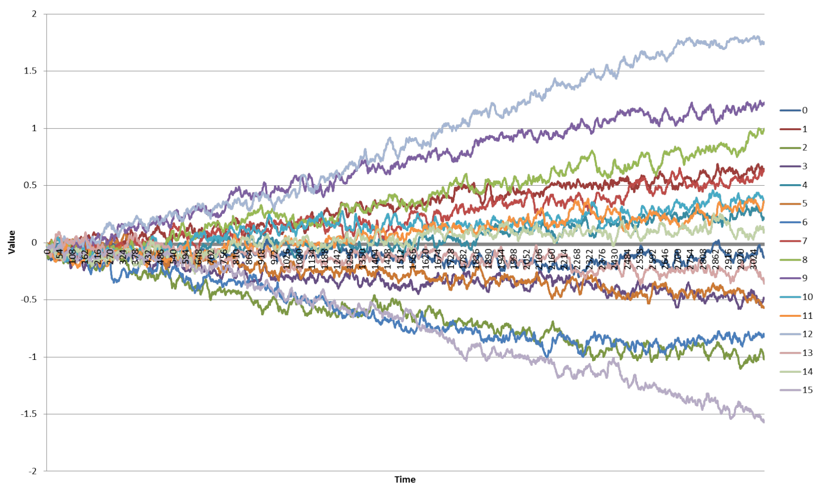


Figure 8.9: Sixteen different trajectories using each analog accelerator integrator integrating white noise. I plot voltage against time, out to 200ms, equivalent to over 20K autocorrelation periods.

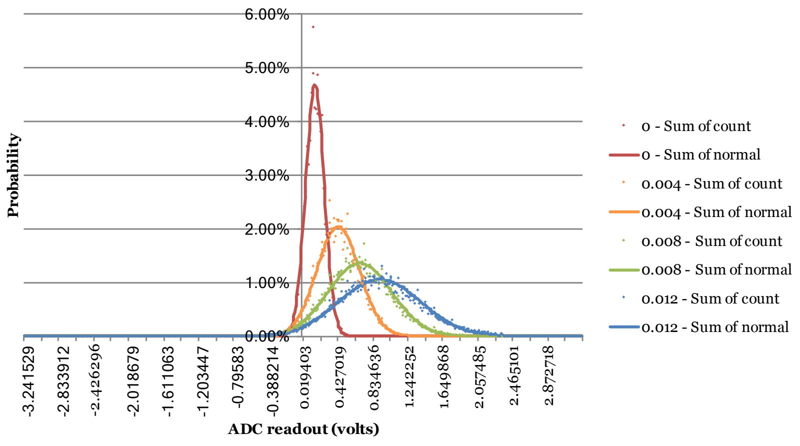


Figure 8.10: Probability density functions of 65K Brownian motion paths at intervals of 4 milliseconds. The circuit setup is that shown in Figure 8.8. I plot the number of measured values (probability) at four time points against voltage. The rightward drift of mean of the histograms is due to integrator input offsets caused by environmental effects. The plot is what we expect for stochastic diffusion: the drift is linearly proportional to time t, the standard deviation σ is proportional to $\sqrt{t}$, and the variance σ^2 is proportional to t.

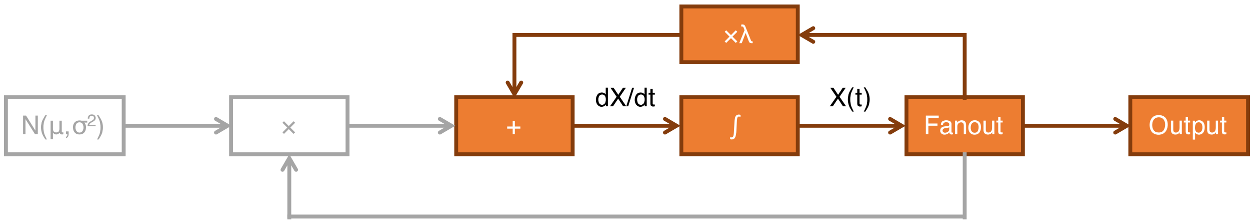


Figure 8.11: Exponential growth process.

8.3 Analog Black-Scholes bringup: standard Wiener process / Brownian motion

The standard Wiener processes, also known as Brownian motion, is the time integral of white noise [114, 174, 151, 146]:

$$W(t) = \int_0^t \frac{dW(\tau)}{d\tau} d\tau$$

Figure 8.8 shows an analog accelerator configuration that integrates the white noise source, described in the previous section, to create signal trajectories belonging to the standard Wiener process, shown in Figure 8.9.

It is important to establish correct analog accelerator solutions for the standard Wiener process because the solutions are a component of the Black-Scholes stochastic ODE. For example, Figure 8.10 shows the effect of one of the sources of error tackled in the previous section. Environmental effects such as temperature changes disturb the mean value of the white noise source, which in turn cause histograms for the standard Wiener process to drift. Because of these environmental sources of error, I had to use filtered analog noise or digital noise to get correct standard Wiener process solutions.

8.4 Analog Black-Scholes bringup: exponential growth process

After taking care of the stochastic parts of the Black-Scholes stochastic ODE, this section focuses on solving in the analog accelerator the deterministic part of the equation, an exponential growth process. Solving this deterministic part of the SDE has its own set of challenges, specifically in the calibration of the analog components and in scaling the SDE parameters for solution in analog. Accurate calibration and intelligent scaling is important for getting correct solutions to the Black-

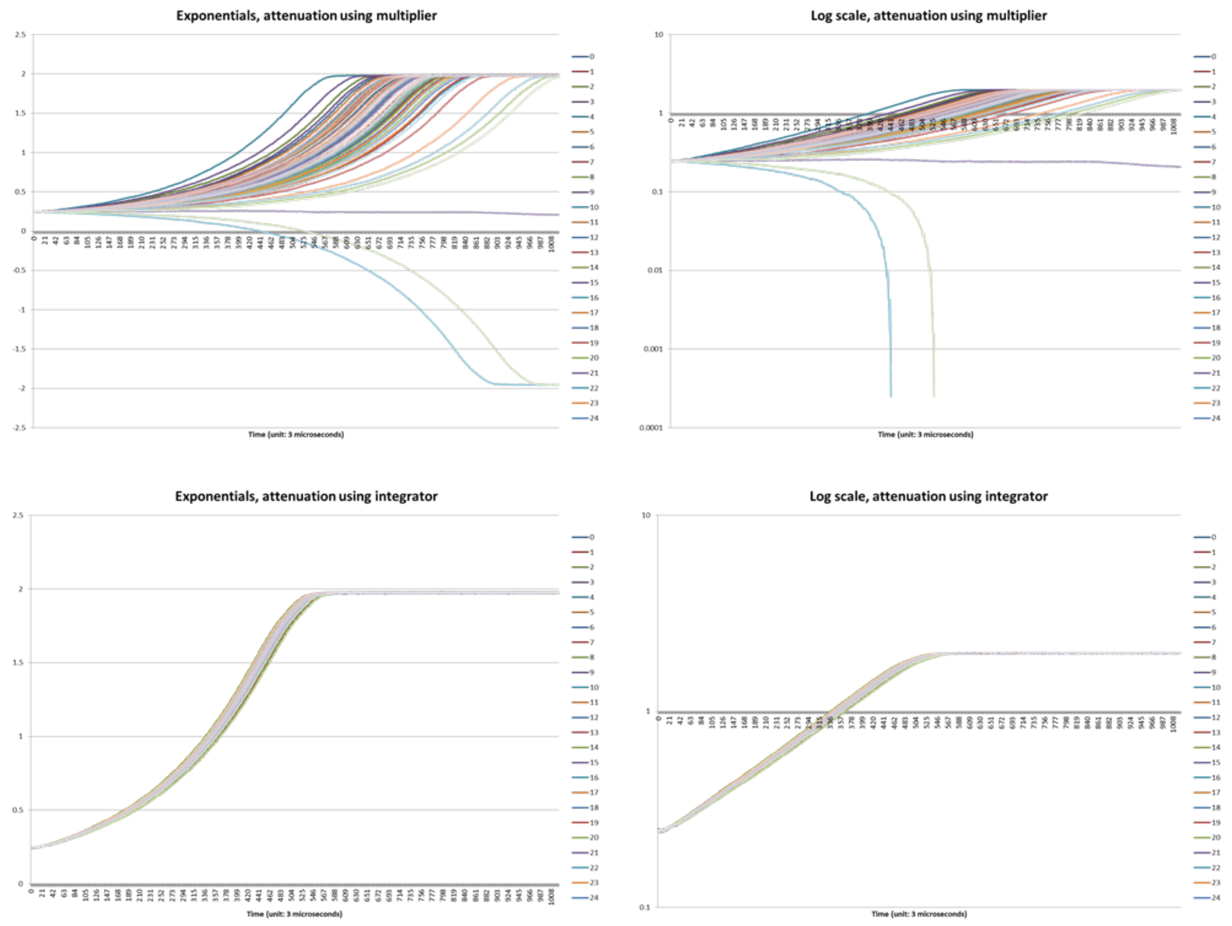


Figure 8.12: Analog accelerator solutions for the exponential growth curve in Black-Scholes stochastic ODE. The two rows are a comparison showing the impact of using integrator input attenuation for controlling noise. In top row the feedback attenuation is done using the multiplier. In bottom row the feedback attenuation is done using both the multiplier and the integrator input. In both cases the feedback gain is $\lambda = \frac{1}{80}$, but the exponential growth trajectories are much more accurate in the bottom row. The right column plots the exponential growth curves on log-linear axes so the time constant can be calculated from the slope. This time constant is 8.3 microseconds, corresponding to 120 KHz. The integrators are set to an initial condition of $X_0 = 0.25$.

Scholes stochastic ODE using an analog accelerator.

The circuit setup for this test shown in Figure 8.11 is simple. We connect the integrator output to a fanout. One branch of the fanout goes through a multiplier, set to a small positive gain λ (e.g., $+\frac{1}{80}$). That small positive feedback is connected back to the input of the integrator. The integrator is set to some small positive initial condition X_0 (e.g., $\frac{1}{16}$). The positive feedback gain and the initial condition are both set to small values in order to get a long integration time for useful computation. We observe the expected exponential using a second branch of the fanout, connected to the integrator output.

Precise calibration and control of the analog accelerator is important for getting accurate solutions to this part of the Black-Scholes stochastic ODE. Four types of errors contribute to inaccurate solutions for the exponential trajectories. I discuss the relative importance of these sources of errors and mitigation techniques below:

1. **Variation in the multiplier gain:** We would like to calibrate the multiplier to realize a minuscule gain factor λ, in order to elongate the useful time duration for solving the Black-Scholes stochastic ODE. That is because a low gain factor decreases the growth rate of the exponential growth process and delays the time when the integrators become saturated. But in practice accurate calibration of the multipliers for tiny gain factors (e.g., $\lambda = \frac{1}{160}$) is difficult. A gain factor of $\lambda = \frac{1}{80}$ balances the need for a small factor and precise calibration for that factor. The bottom right subplot of Figure 8.12 plots an ensemble of exponential growth trajectories on a log-linear chart; the slope of the trajectories indicate the multiplier gain factor. The plot shows little variation in the realized multiplier gains for $\lambda = \frac{1}{80}$.
2. **Variation in the integrator initial condition:** We would also like to calibrate the integrators to start off with a minuscule initial condition X_0, also in order to elongate the useful time duration for solving the Black-Scholes stochastic ODE. While precise calibration of the integrator initial condition is easier than that for multiplier gains, we should avoid too small of an initial condition because of noise, which is another source of error I discuss below. The bottom subplots of Figure 8.12 show little variation in the calibrated integrator initial conditions.
3. **Jitter in the integrator start time:** Very little variation here; the integrator start times are consistent to within 1 microsecond.
4. **Noise at the integrator input and multiplier output:** Once the above sources of error are minimized, stochastic noise becomes the biggest contributor of error in the deterministic exponential growth curve. Specifically, the noise at the input of the integrator is most problematic at the beginning of the exponential growth curve, when its amplitude is comparable to the actual signal at the beginning of the integration. In other words at the beginning of integration the integrator is mostly integrating noise, before the exponential signal catches

up and becomes dominant. Worse, we cannot rearrange the definition of the Black-Scholes stochastic ODE to make use of noise at the integrator inputs, as the Black-Scholes stochastic ODE needs the noise amplitude to be proportional to the present value of x.

The trick to reducing the integrator's sensitivity to noise is to turn on the attenuation feature of the integrator inputs. The attenuation setting of the integrator input makes the integrator expect a -20uA to +20uA signal instead of the usual -2uA to 2uA signal. With attenuation on, the integrator effectively implements a $\times 0.1$ signal gain internally. Then, because the integrator is also participating in elongating the exponential curve time constant, we can use a slightly bigger (less attenuated) gain in the feedback amplifier. So now the positive feedback signal is not as feeble compared to the background noise. The better signal to noise ratio at the integrator input leads to more consistent exponential curves.

Figure 8.12 shows the impact of using integrator input attenuation for controlling noise. In the "before" setup (top row), the feedback attenuation is done using the multiplier (gain=1/80). In the "after" setup (bottom row), the feedback attenuation is done using both the multiplier (gain=1/8) and the integrator input (gain=1/10). In both cases the feedback gain is 1/80, but the exponential growth trajectories are much more accurate in the bottom row.

Scaling problem variables in the Black-Scholes stochastic ODE to match analog accelerator variables. An important challenge for us is how to logically scale problem variables for this type of problem. In the overall end-to-end analog accelerator solution for the Black-Scholes stochastic ODE, we need to scale the problem variables so the integration time period matches the time period granted by the exponential growth process discussed above. We would set the positive feedback gain and the initial condition as small as possible in order to delay integrator saturation, in order to get a long integration time that fits more autocorrelation periods and ADC samples.

Usually, we can rescale problems variables to analog variables when problems are linear. In the Black-Scholes stochastic ODE, the deterministic part is just an exponential growth process, which is a linear system. We can easily rescale the deterministic part. The stochastic part of the Black-Scholes stochastic ODE on the other hand is nonlinear, so scaling is more difficult.

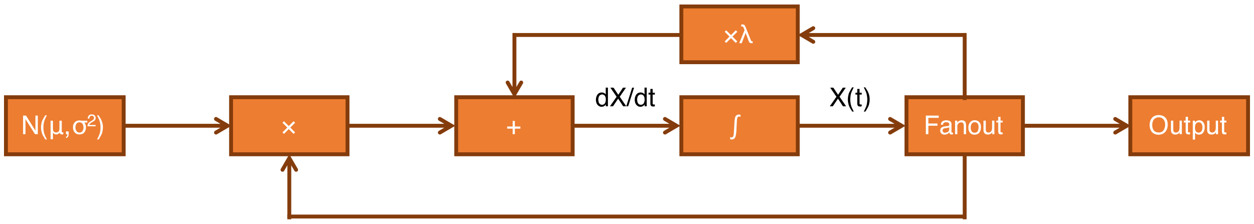


Figure 8.13: Black-Scholes stochastic ordinary differential equation.

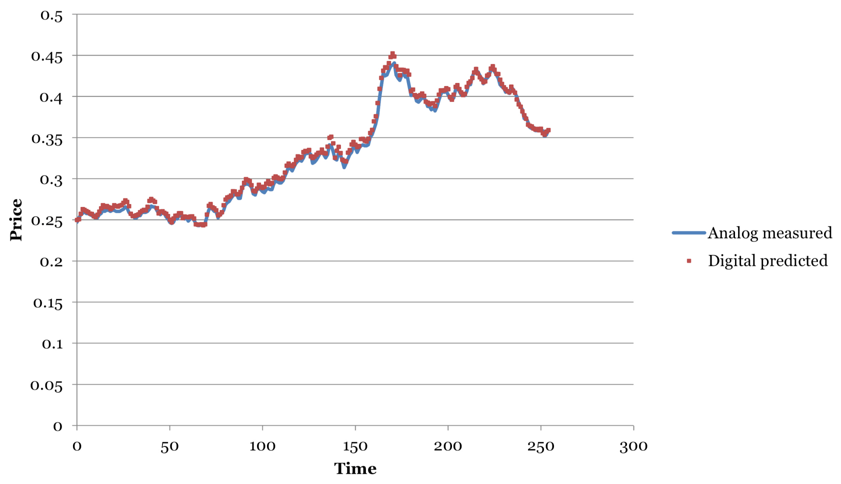


Figure 8.14: Example analog and digital solutions for Black-Scholes SDE. Both solutions use the same random number sequence.

8.5 Convergence & time for analog and digital Black-Scholes

In this section I use the analog accelerator to solve the full Black-Scholes stochastic ODE, and compare the accuracy and performance against a digital solver. Figure 8.13 shows the analog accelerator configuration for solving the Black-Scholes ODE, with all subcomponents I've covered in prior sections enabled.

A conventional digital computer can use several techniques for solving the Black-Scholes ODE. One option is to use the closed form solution Equation 8.2, which gives the solution for X at any time, given a predetermined input for the standard Wiener process. Here we are more interested in Monte-Carlo numerical methods that will give us the full trajectory of $X(t)$, since pricing models for many types of options need this trajectory. The basic Monte-Carlo method for SDEs include the stochastic Euler-Maruyama and higher order Milstein methods [174, 151, 146]. Figure 8.14 shows one example solution for the Black-Scholes SDE using a random number sequence for both

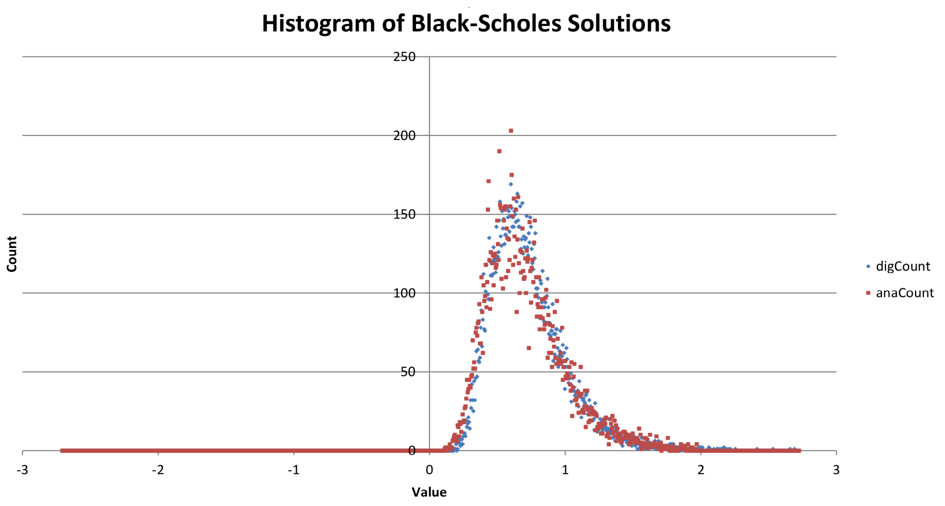


Figure 8.15: The distribution of digital and analog final solutions match well. Here, the histogram is generated from an ensemble size of 16K. This plot uses the following parameters, which I have established in previous experiments to be the optimal settings to get the analog solver to work well: initial condition $X_0 = 0.25$, exponential positive feedback gain $\lambda = 0.0125$, standard deviation of noise $\sigma = 0.039563$.

the digital and analog solvers.

In evaluating the accuracy of Monte-Carlo method solutions there is a difference between *strong convergence* and *weak convergence*. As we invest more computation steps and time in the Monte-Carlo method, the solutions improve in two ways: strong convergence is the improvement of each trajectory given by the Monte-Carlo method, measured as the decrease in the mean of the errors for each solutions; on the other hand weak convergence is the improvement of average of all the trajectories as an ensemble, measured as the decrease in the error of the mean for all the solutions.

Figure 8.14 for instance shows the effect of strong convergence of the digital solver in comparison to an analog solution. If the digital solver takes fewer time steps with wider interval between the steps, the digital solver would track the analog solution less accurately. The mean of error for digital ODE solutions improves as the step sizes decrease, at the cost of taking more computation time.

Figures 8.15 and 8.16 shows the effect of weak convergence of the digital and analog solvers. The mean of the distribution converges to the expected mean, as the ensemble size grows. If we keep increasing the solvers' ensemble size, the error of the mean solutions will keep decreasing; the trend is bounded by a linear frontier on a log-log plot.

Figures 8.17 and 8.18 compare the time cost of random number generation, the digital solution,

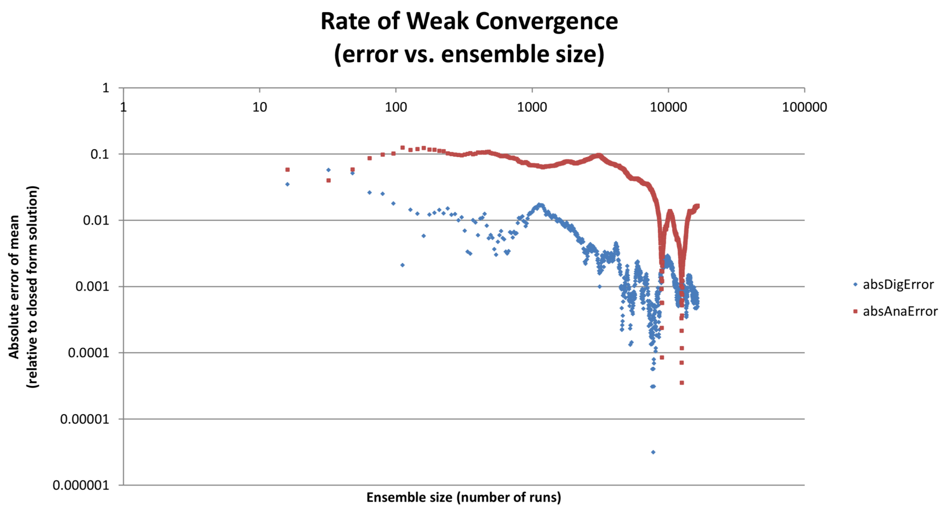


Figure 8.16: The accuracy of analog and digital solutions, in terms of the weak convergence of the solution as ensemble size grows. As expected, the digital solver steadily converges to the solution. For the analog solver, the precision of the calibration seems to ultimately set a limit on how accurate the solution can be. Here, we are calibrating every 1000 runs, so for the 16K final ensemble size we had recalibrated 16 times. The recalibration has two purposes: one is to account for any drift in the analog system; the other is to remove any systematic positive or negative bias in the solution.

the calibration routine, and the analog solution. Random number generation takes a lot of time because the basic method of converting from a uniform random number to a normally distributed random number is costly. A high quality analog noise source would provide Gaussian noise at zero time cost, making the analog solution much faster than the digital approach.

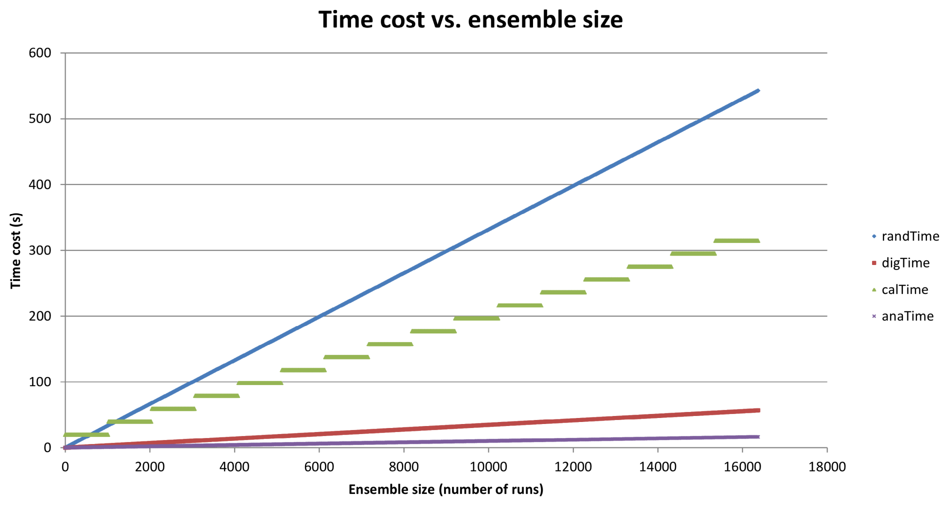


Figure 8.17: The time cost of random number generation, the digital solution, the calibration routine, and the analog solution; all of these grow as the ensemble size grows. Notably, the calibration cost grows only when we recalibrate the analog solver, so it is a step function.

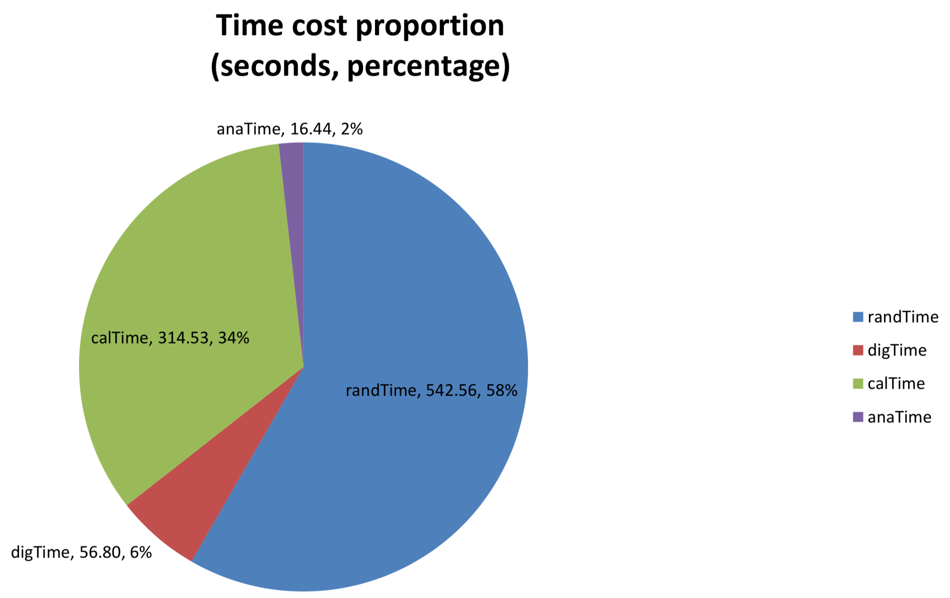


Figure 8.18: Pie chart of where computation time is spent in random number generation, digital solution, analog calibration, analog solution. The analog solution time includes the time to convert data between the analog chip and the microcontroller. The calibration time for the analog solver grows or shrinks depending on how often calibration is done.

Part IV

Analog-Digital Co-Processing for Solving Algebraic Equations

Chapter 9

Analog-Digital Co-Processing for Solving Algebraic Equations

In this part we explore using analog accelerators to assist scientific computation workloads by solving algebraic equations. The motivation for doing so is that modern scientific computation is built upon on solving linear and nonlinear algebraic equations. Workload profiles of scientific computation benchmarks show that solving applications go through various steps to transform differential equations problems into systems of algebraic equations. Solving these systems of equations becomes the dominant kernel and takes the most computing time.

Using analog accelerators to solve algebraic equations has several advantages, compared to using them to solve differential equations. One advantage is it decreases the engineering effort of converting existing problems, algorithms, and source code to use analog accelerators. The second advantage is that it draws on the complementary strengths of analog and digital architectures.

9.1 Algebraic equations dominate software profiles of equations, solvers, libraries

A survey of scientific computation literature shows that solving linear and nonlinear algebraic equations is the most important numerical primitive [36, 6, 62, 48]. A workload characterization of some engineering PDE solvers reveals that they spend a large fraction of their runtime in solving

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 [9]	finite element discretization with nonlinear spring forces	Solving Helmholtz PDE with preconditioned SOR and CG	15.3%

Table 9.1: Function profile of 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. Finite volume and finite elements are less structured, and the resulting less regular memory accesses shift computation time away from solving systems of equations. We profiled these applications at runtime using Valgrind KCachegrind, gperftools, OProfile, and GNU gprof to identify the subroutines relevant to hybrid analog-digital co-processing.

systems of algebraic equations (Table 9.1).

The reason solving algebraic equations is so important in differential equations problems for scientific simulations is due to three major reasons, spanning three conceptual levels: the physical model, the numerical algorithm, and hardware support for solvers.

Importance of algebraic equations: physical model

Scientific computation workloads solve algebraic equations as a way to capture large dynamic range in a physical model. The dynamic range can come from wide ranges of scales in variable values, and in space and time dimensions.

A concrete example shows up in the classical way of solving incompressible Navier-Stokes problems [59, 160]. There, Navier-Stokes solvers have to solve a system of linear equations (for a Poisson elliptic equation) describing the fluid's pressure field. The algebraic equations are keeping track of the pressure field, the evolution of which happens much quicker than the other variables in the model—while the fluid velocity field ripples slowly through the modeled space according to a hyperbolic wave equation, the incompressibility assumption on the fluid means pressure changes propagate through the modeled space in a figurative blink of an eye. So, the pressure field is *steady* within each time step. The standard way to solve steady elliptic PDEs is to use multigrid and iterative numerical linear algebra solvers. Because the pressure field needs updating every timestep of the velocity field, solving for the pressure field ends up taking the most time to compute.

The reason iterative numerical linear algebra solvers dominate scientific computation software profiles becomes clearer, when we also consider the physical meaning of the pseudo-time steps in the iterative solvers. An incompressible Navier-Stokes solver has to correctly solve for the quickly-evolving pressure field, in addition to the slowly-evolving velocity field. Therefore, the real rate at which the simulation can advance is the rate at which the solver figures out the solutions to the pressure field. As a result, the algebraic solvers for the pressure field end up consuming most of the computation time.

Importance of algebraic equations: numerical algorithm

Scientific computation workloads solve algebraic equations also as a way to decompose large problems to improve available parallelism and subproblem locality. Domain decomposition methods rephrase differential equation problems as solving algebraic equations, in order to break apart large problems into several subproblems, while at the same time getting the correct result for the overall problem [100, 167, 61].

The alternating directions implicit (ADI) operator splitting method for solving the 3D Navier-Stokes equations gives us an example of the approach. A 3D differential equation stencil (x,y,z dimensions) causes additional problems compared to a 2D stencil (x,y dimensions) for conventional digital architectures. The variables for neighboring cells in the z dimension have the least locality when problem variables are stored in multi-dimensional arrays. As a result of the 3D stencil, the cache access stride length for the workload becomes longer, potentially impacting performance if the working set exceeds the cache size. A common way work around this problem when tackling 3D Navier-Stokes equations is to split the problem into 1-dimensional slices at a time, to increase the problem variable locality. As a result, we get the added benefit of an increased number of parallel subproblems in software, which can exploit hardware thread-level parallelism. The domain decomposition method that ensures the overall solution is correct has overhead iterations and costs in decomposing the problem, but those costs are overcome by the performance benefit of increased parallelism.

Importance of algebraic equations: hardware support

Finally, good support for solving linear algebra problems from digital hardware architectures is another reason scientific computation workloads focus on solving algebraic equations. Differential equations solvers are tuned to extract performance from the thread-level, data-level, and instruction-level parallelism modern digital architectures offer. Domain decomposition methods allow extracting thread-level parallelism, while numerical linear algebra methods extract data- and instruction-level parallelism. Numerical linear algebra libraries for GPUs provide even higher performance support for linear algebra. As a result of the strong support for linear algebra from hardware, scientific computation research has gravitated toward rephrasing problems as algebraic equations.

9.2 Solving algebraic equations as the interface between analog accelerator and digital host

Due to the above reasons, solving algebraic equations is the most important kernel in scientific computation. Solving algebraic equations in a hybrid analog-digital solver system would support many PDE solvers, while needing little rework of existing digital solvers. The problem kernel of solving algebraic equations serves as an analog-digital program partitioning where existing software for scientific computation ends, and where our new hardware model of computation steps in.

The approach in this part contrast with the approach in Part III. In Part III, we directly mapped differential equations problems onto the dynamics of analog and digital accelerators, also phrased as differential equations. Unfortunately, directly mapping differential equations to hardware limits us to solving problem sizes that can fit in the hardware, and provides solutions with accuracy limited by the hardware's accuracy. Making matters more difficult, the analog computational model provides limited choices on how to break down the PDE and map equation variables to hardware. That's rather restricting compared to the variety of discretization methods used for PDE solvers from several disciplines (Table 9.1).

The methods in Part III potentially offer high performance and efficiency rewards, but at the same time they run the risk of abandoning or reinventing PDE solving algorithms discovered in the digital era. The methods in this part on the other hand keep existing differential equations solver intact, foregoing some performance and efficiency benefits of a completely new approach, with the goal of making analog accelerator immediately useful in more situations.

9.3 Complementary strengths of hybrid analog-digital solvers for algebraic equations

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 such as GPUs 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 part of this thesis 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 PDE problems 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.

In the remaining chapters of this part I discuss our findings in using our analog accelerators to tackle linear and nonlinear algebraic equations.

Chapter 10

Analog-Digital Co-Processing for Linear Algebra

This chapter uses the programmable analog accelerator for solving systems of linear equations. In work published in [82, 83], our team compared the analog solver's performance and energy consumption against an efficient digital algorithm running on a general-purpose processor. The analog approach may have 10$\times$ better performance than digital methods, while spending $\frac{1}{3}$ less energy, for certain designs of the analog accelerator and certain problem sizes.

We found analog co-processors for linear algebra must offer high *analog bandwidth* in order to speed up convergence of the circuit. Providing this analog bandwidth in the circuit consumes silicon area and increases power consumption. These limitations ultimately limit the performance and efficiency benefits of analog co-processing for linear problems.

The potential benefit of analog co-processing for linear systems is further limited in comparison to the best digital solvers. Digital algorithms such as conjugate gradients are optimal, and are difficult to beat using analog co-processors, no matter the intrinsic speed and efficiency of analog hardware. Finally, we conclude that problem classes outside of systems of linear equations could hold more promise for analog acceleration.

Nonetheless, the ability to solve linear algebra problems in analog becomes a useful "inner loop" for other problems we solve in analog co-processors, such as the nonlinear systems of equations in Chapter 11.

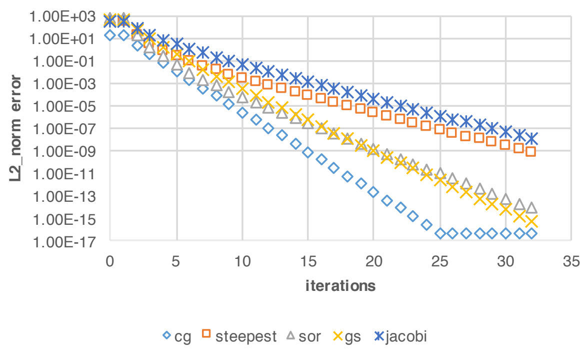


Figure 10.1: Comparison of iterative numerical linear algebra algorithms for solving a Poisson equation. The problem is discretized using finite differences with 16 points over three dimensions, for a total of 4096 grid points. Boundary condition $u(x, y, z) = 1.0$ for the plane $x = 0$, $u(x, y, z) = 0.0$ otherwise.
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.

10.1 Importance of linear algebra

Solving systems of linear equations is the single most important numerical primitive in continuous mathematics [36, 6, 48]. Many modern scientific computing and big data problems are converted to linear algebra problems. Just to give a few examples, these problems include optimization problems (such as linearly constrained quadratic programming) [90, 133, 20], finite difference elliptic PDEs (such as the Poisson equation) [18, 22], and support vector machines. 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]. Analog acceleration would be extremely useful if it can tackle linear algebra and serve as a foundation for problems in many domains.

10.2 Digital iterative numerical methods for linear algebra

As a quick review of linear algebra, the problem entails finding a value for $\vec{x}$ that satisfies the equation $A\vec{x} = \vec{b}$, where A is a known matrix of coefficients and $\vec{b}$ on the right-hand side is a known vector of constants or biases. No closed form solution exists if the size of matrix A is 4×4 or greater, so digital computers resort to numerical linear algebra methods.

Linear algebra algorithms are categorized as direct and iterative solvers [169, 138]. *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.

On the other hand, *iterative solvers* start at an initial guess $\vec{u_{init}}$; then, the entire solution evolves step-by-step toward the correct answer according to an algorithm until the solution stops changing and is accurate at $\vec{u_{final}}$. Even if an iterative solver is stopped short of full convergence, the intermediate solution still approximately satisfies the original system of linear equations. Classical iterative numerical linear algebra methods include Jacobi, Gauss-Seidel, successive over-relaxation, and conjugate gradient methods [161]. Figure 10.1 establishes that conjugate gradients (CG) has the best convergence rate among classical iterative methods. Efficient iterative methods such as the conjugate gradient method are increasingly important because intermediate guess vectors are a good approximation of the correct solution [48].

An important property of linear algebra problems is *conditioning*, which controls the difficulty of solving the linear algebra problem. Intuitively, ill-conditioning arises when there is a large dynamic range in the values of variables (technically, the eigenvalues of the system). When a numerical linear algebra method tackles an ill-conditioned problem, it becomes more likely that in the course of solving the problem, a divide-by-zero error occurs.

Ill-conditioning is a symptom of imperfect modeling of problems or physical systems. For example, a scientist or a mathematician may write down a model equation without a clear idea of which aspects of the model are dominant. In such situations it is difficult for numerical integrators to operate, because no choice of a time step size permits both accurate calculation of the gradient and rapid simulation of the system. Ill-conditioning is a problem in machine learning as well. Overfitting in regression is the result of directly using the stiff solution. The model has been forced to treat relatively irrelevant control variables as significant, resulting in a poor regression result. When ML researchers do regularization, they are actually conditioning their matrix.

Iterative linear algebra algorithms introduce a concept called *step size* in order to handle ill-conditioned problems. The step size controls how much the solution vector changes in each algorithm step. The step size affects the algorithm's efficiency and requires many processor cycles to calculate. In the conjugate gradient method, for example, the step size is calculated from previous step sizes and the gradient magnitude, and this calculation takes up half of the multiplication

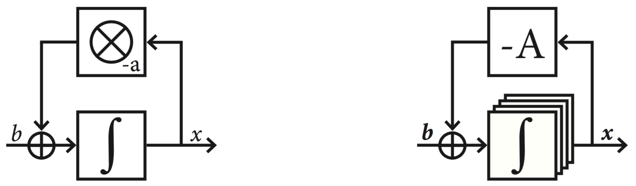


Figure 10.2: How feedback in an analog computer circuit can be used to implement scalar division (left) and solving for the solution vector of a linear algebra problem (right).

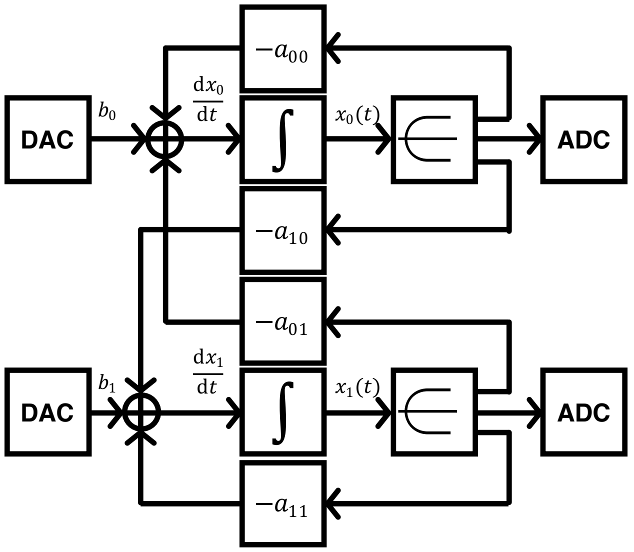


Figure 10.3: Schematic of an analog accelerator for solving $A\vec{x} = \vec{b}$, a linear system of two equations with two unknown variables. Matrix A is a known matrix of coefficients realized using multipliers; x is an unknown vector contained in integrators; b is a known vector of biases generated by digital-to-analog converters (DACs). Signals are encoded as analog current and are copied using current mirror fan-out blocks. The solver converges if matrix A is positive definite, usually true for the problems we discuss.

operations in each conjugate gradient step [161].

10.3 Analog continuous steepest descent for linear algebra

We can also use iterative algorithms in an analog accelerator to solve systems of linear equations. The critical idea is to use the transient behavior of an analog resistor-capacitor circuit, and use that circuit to compute the intermediate results of an iterative numerical method. Essentially, we draw an equivalence between iterative numerical methods in applied math, algorithms in computer science, and transient circuit dynamics in electrical engineering.

The key difference in doing iterative algorithms in analog hardware is the that guess vector is updated using infinitesimally small steps, over infinitely many iterations. This continuous trajectory from the original guess vector to the correct solution is an ordinary differential equation (ODE), which states the change in a set of variables is a function of the variables' present value. We would then apply the methods discussed in Section 5.1 and Part III to solve the ODE.

Let's walk through an example of an analog accelerator configuration for solving an ODE that in turn solves a system of linear equations. At the analog accelerator's heart are integrators, which contain the present guess of the solution vector represented as an analog signal evolving as a function of time (see Figures 10.2, 10.3). The accelerator performs operations on this solution vector by feeding the vector through a linear network of multiplier and summation units. Digital-to-analog converters (DACs) provide constant coefficients and biases. Using these function units, the hardware creates a linear function of the solution vector, which is fed back to the inputs of the integrators. In this fully formed circuit, the solution vector's time derivative is a linear function of the solution vector itself.

The analog accelerator charges the integrators to initial condition values representing the iterative method's initial guess. The accelerator starts computation by releasing the integrators, allowing its output to deviate from its initial value. Then, the variables contained in the integrators converge on the correct solution vector that satisfies the system of linear equations. When the analog variables are steady, the accelerator samples them using analog-to-digital converters (ADCs).

These techniques were used in early analog computers [30, 94, 102] and have recently been explored in small-scale experiments with analog computation [120, 49, 183, 184].

10.4 Mitigation of analog linear algebra disadvantages

This linear algebra use case for analog accelerators encounters several drawbacks of analog computing, including limited accuracy, precision, and scalability. We demonstrated mitigations for each of these problems in the context of solving linear algebra, although the techniques we discuss apply to other styles of analog computer architecture.

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

Algorithm 1: Building precision in analog result

Improve sampling precision by focusing on analog steady state

High-frequency and high-precision analog-to-digital conversion is costly. So, instead of trying to capture the time-dependent analog waveform, we use the analog accelerator as a linear algebra solver by solving a convergent ODE. In contrast to solving time-varying ODEs, here the analog accelerator's ADCs only have to sample the value of the stable output $\vec{u_{final}}$, which means that sampling frequency is not a concern. When the analog accelerator outputs are steady, ADCs can sample the solutions with higher-precision.

Even then, high-precision ADCs still fall short of the precision in floating-point numbers. The digital host gets higher-precision results by running the analog accelerator multiple times. The digital host computer finds the residual error in the solution, and then sets up the analog accelerator to solve a new problem, focusing on the residual. Each problem has smaller-magnitude variables than previous runs, which lets the software scale up the variables to fit the dynamic range of the analog hardware. Iterating between analog and digital hardware a few times results in a more precise result than using the analog hardware alone. This procedure is shown in Algorithm 1.

Tackle larger problems by accelerating sparse linear algebra subproblems

Modern workloads routinely need thousands of variables, corresponding to as many analog integrators in the accelerator, exceeding the area constraints of realistic analog accelerators. Furthermore, the analog datapath is fixed during continuous time operation, so there is no way to dynamically load variables from and store variables to main memory.

Analog accelerators can solve large-scale sparse linear algebra problems by accelerating the solv-

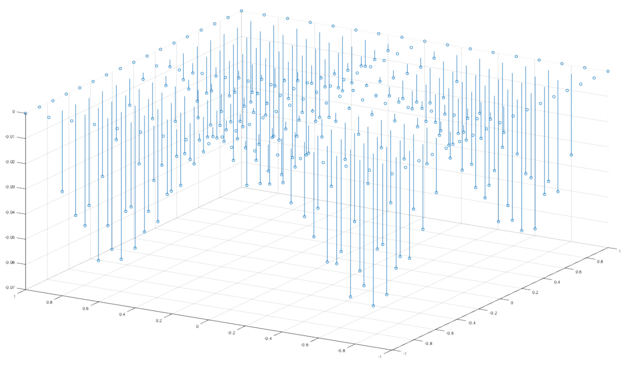


Figure 10.4: An example of the solution to an elliptic PDE. The continuously varying field has been discretized into node variables which are solved using linear algebra.

ing of smaller subproblems. This lets analog accelerators solve problems containing more variables than the number of integrators in the analog accelerator.

In such a scheme, the analog accelerator finds the correct solution for a subproblem. To get overall convergence across the entire problem, the set of subproblems would be solved several times, using an outer loop iterating across the subproblems.

As a specific example, we walk through the process of splitting a 2D Poisson elliptic PDE, defined on the unit square ($\Omega = [0,1]^2$). A 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. Its solution, with appropriate discretization, may look like the example in Figure 10.4. 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 $\vec{u}$ which are interrelated according to the system of linear equations:

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

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

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

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 that cells are only related to themselves, and to their four neighbors.

In practice, physics simulations using PDEs have millions of grid points in the vector $\vec{u}$, 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:

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

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

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 $\vec{u_s}$ 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 con-

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 10.1: Summary of analog chip components taken from [66, 67].

vergence properties as iterative methods on individual numbers. Therefore, it is still desirable to ensure the block matrices captured in the analog accelerator are large, so that more of the problem is solved using the efficient lower-level solver [62, 22].

Handle indefinite matrices by multiplying by the matrix transpose

A problem that both discrete and continuous gradient descent methods face is if A is an indefinite matrix. Indefinite matrices do not have global maxima or minima in the solution set: the solution set is convex in some dimensions but concave in others. We observed that if the input matrix is indefinite, the output does not converge and the integrators saturate. To counter this, we substitute A with $(A^T)A$. This can be fixed by multiplying the input matrix by its transpose. In practice this is what discrete gradient descent solvers do on digital computers. We can also do this in the CPU or with analog multipliers.

10.5 Design space exploration of high-bandwidth analog accelerators for linear algebra

We compare the analog accelerator and digital approaches in terms of performance, hardware area, and energy consumption, while varying the number of problem variables and the choice of analog accelerator component bandwidth, a measure of how quickly the analog circuit responds to changes.

Power and area model

Using physical timing, power, and area measurements recorded by Ning Guo and colleagues [66, 67] and summarized in Table 10.1, we built a model that predicts the properties of larger-scale analog accelerators. In Table 10.1, "core power fraction" and "core area fraction" show the power and

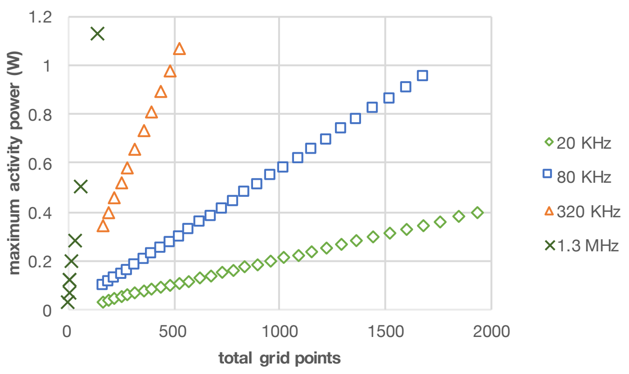


Figure 10.5: Power versus analog accelerator size for various bandwidth choices. We observe that analog circuits operate faster when the internal node voltages representing variables change more quickly. We hold the capacitance fixed to the capacitance of the prototype's design, and use larger currents that draw more power to charge and discharge the node capacitances in the signal paths carrying variables.

area of each block that forms the analog signal path. The core area and power scale up and down for different analog bandwidth designs. Not all area and power consumption of the blocks of the prototype design are involved in the analog signal path. The area and power consumption of such subcircuits do not need to scale up for higher bandwidth designs. The noncore transistors and nets not involved in analog computation include calibration and testing circuits and registers. We explore how different bandwidth choices influence analog accelerator performance and efficiency.

Analog bandwidth model

The prototype chip has a relatively low analog 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 prototype's low bandwidth makes it unrepresentative of an analog accelerator designed to solve time-independent algebraic equations, in which accuracy degradation in time-dependent behavior has no impact on the final steady state output. We scale up the model's bandwidth, within reason, up to 1.3 MHz.

Increasing the bandwidth of the analog circuit design proportionally decreases the solution time, but also increases area and energy consumption. As Figures 10.5 and 10.6 show, 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.

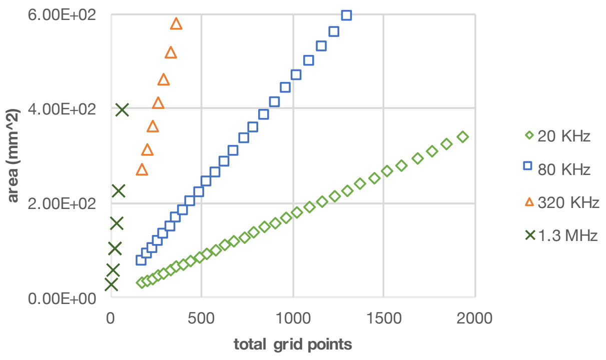


Figure 10.6: Area versus analog accelerator size for various bandwidth choices. We observe that the transistor aspect ratio W/L must increase to increase the current, and therefore bandwidth, of the design. L is kept at a minimum dictated by the technology node, leaving bandwidth to be linearly dependent 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.

The projected analog power figures are significantly below the thermal design power of clocked digital designs of equal area. Even in the designs that fill a 600 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 KHz of bandwidth.

10.6 Sparse linear algebra case study

We use as our test case a sparse system of linear equations derived from a multigrid elliptic partial differential equation (PDE) solver. In multigrid PDE solvers, the overall PDE is converted to several linear algebra problems with 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 iterating the multigrid algorithm [22, 18]. Because perfect convergence is not required, less stable, inaccurate, and low-precision techniques, such as analog acceleration, can support multigrid.

In our case, we compare the analog accelerator designs to a conjugate gradient algorithm running on a CPU, solving to equal (relatively low) solution precision, equivalent to the precision obtained from one run of the analog accelerator equipped with high-resolution ADCs. On the digital side, the numerical iteration stops short of the machine precision provided by high-precision digital

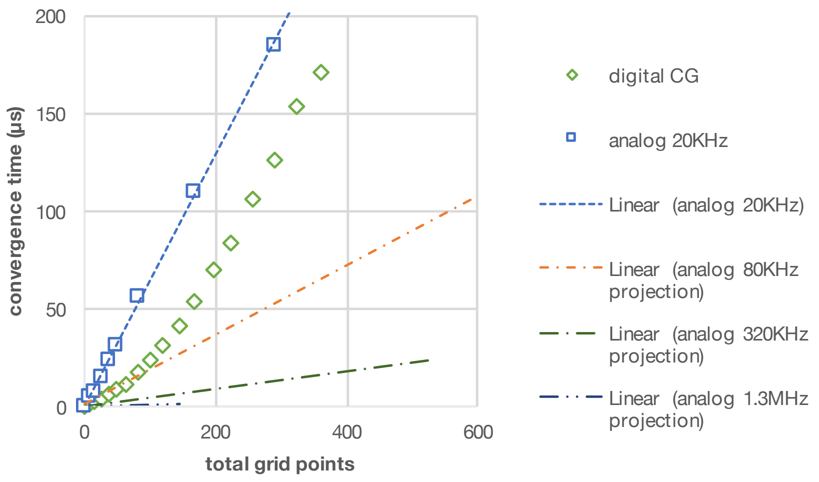


Figure 10.7: Comparison of time taken to converge to equivalent precision, for high-bandwidth analog accelerators and a digital CPU. The time needed to converge is plotted against the linear algebra problem vector size. We give the projected solution time for 80-KHz, 320-KHz, and 1.3-MHz analog accelerator 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.

floating-point numbers.

Analog and digital linear algebra performance comparison

As Figure 10.7 shows, we found that an optimal analog accelerator design that balances performance and the number of integrators should have components with an analog bandwidth of approximately 320 KHz. With our bandwidth model, high-bandwidth analog computers come with high area cost, quickly reaching the area of the largest CPU or GPU dies. On performance and energy metrics, we find that with 400 integrators operating at 320 KHz of analog bandwidth, analog acceleration can potentially have a 10-times faster solution time.

Analog and digital linear algebra energy comparison

We compare the solution energy of analog and digital solvers in Figure 10.8. Using our analog bandwidth model for power, this design corresponds to 33 percent lower energy consumption compared to a digital general-purpose processor.

Using an estimate of 225 pJ for every floating point multiply-add operation in GPUs [95], we

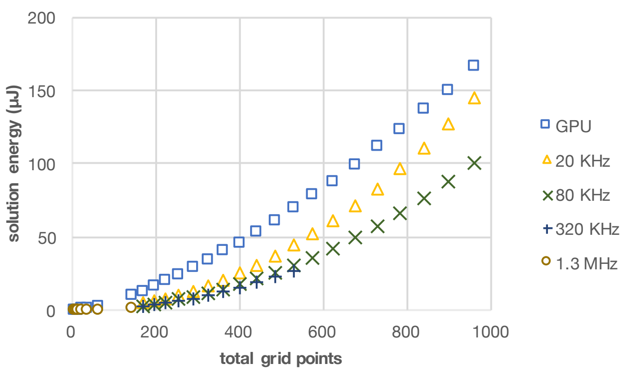


Figure 10.8: The energy needed to solve 2D problems of varying number of total grid points, for different analog accelerator designs, compared against a GPU running CG. The 80 KHz design shows some energy savings relative to the GPU. High bandwidth analog accelerators are quickly limited by their large 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.

derive the amount of energy needed for GPUs to compute the solution to equivalent accuracy as that of the analog accelerator. As bandwidth increases, a higher fraction of area and power consumption becomes directly involved in analog computation— calibration, testing, and digital circuits become a smaller fraction of area and power costs— resulting in a more energy-efficient design. Once almost all 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 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.

10.7 Challenges and pitfalls of analog linear algebra

Effect of variable dynamic range on analog performance and efficiency

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

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

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

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

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

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

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:

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

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

We see that the result $\vec{u}(t)$ remains unchanged so long as we also scale down $\vec{b}$ 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 [13, 69, 27].

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 on page 98).

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 ma-

		Analog			**Conjugate gradients**		
	Grid points	HW cost	Conv. time	Energy = HW $\times$ time	Conv. 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	$N = L^3$	$N = L^3$
Dense	N	N^2 multipliers	?	?	N	N^2	N^3

Table 10.2: 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.

trices 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. In the case of this comparison between analog and digital solvers, the analog accelerator's need to do value and time scaling is a performance penalty.

Effect of problem dimensionality on analog performance and efficiency

In the 2D Poisson elliptic equation example, we solved systems 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 10.2.

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

10.8 Summary

The performance increases and energy savings for linear algebra are not as drastic as one expects when using a domain-specific accelerator built on a fundamentally different computing model than digital, synchronous computing. The reason for this shortfall is twofold.

First, the high area cost of high-bandwidth analog components limits the problem sizes that can fit in the accelerator, and therefore limits the analog performance advantage.

Second, the extreme importance of linear algebra problems has also led to intense research in optimal algorithms and hardware support. Although discrete-time operation has drawbacks, it permits algorithms to intelligently select a step size, which has advantages in solving systems of linear equations. Both the analog and digital solvers perform iterative numerical algorithms, but the digital program runs the conjugate gradient method, the most efficient and sophisticated of the classical iterative algorithms. In the conjugate gradient method, each step size is chosen, considering the gradient magnitude of the present point, along with the history of step sizes. With these additional calculations, the conjugate gradient method avoids taking redundant steps, accelerating toward the answer when the error is large and slowing when close to convergence. Note in Figure 10.1, 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 fewer iterative algorithms it can carry out. In using the analog accelerator for linear algebra, the design's bandwidth limits the convergence rate, so the convergence rate within a time interval cannot be arbitrarily large. Therefore, the numerical iteration in the analog accelerator is akin to relaxation or steepest descent at a constant rate. Although 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 conjugate gradient and multigrid have been known to researchers since 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.

Changing the basic abstractions in computer architecture could change what types of problems are solvable. Interesting physical phenomena are usually continuous-time, analog, nonlinear, and often stochastic, so the computer architectures and mathematical abstractions for simulating these processes should also be continuous-time and analog. Although analog acceleration has limited benefits for solving linear algebra, I show in the next that section analog acceleration holds promise in problem classes such as nonlinear systems, in which digital algorithms and hardware architectures have been less successful.

Chapter 11

Analog-Digital Co-Processing for Solving Nonlinear Systems of Equations

This chapter explores architectural ideas that allow us to successfully use a analog accelerator, here in the context of solving nonlinear systems of equations [81]. We use the approximate solution from an analog accelerator to help a precise digital nonlinear equation solver running on a GPU. For larger, more nonlinear problems, the hybrid analog-digital solver has a performance improvement of 5.7× and energy savings of 11.6×, relative to a GPU without the help of an analog accelerator.

The favorable findings contrast with the last chapter which found analog accelerators would have limited benefits in solving linear problems, due to prohibitively high analog silicon area costs and due to stiff competition from efficient digital computer algorithms for linear algebra [82, 83]. Here, analog acceleration redeems itself in nonlinear problems, which pose no special challenge in analog but are tricky in digital because the prototypical digital algorithms for nonlinear equations are unreliable.

The difference in how digital *linear* solvers vs. digital *nonlinear* solvers spend computation time is a key factor why analog acceleration had limited impact in helping with linear equations, while significantly helping with nonlinear problems. Digital *linear* solvers give the most significant digits of the solution quickly but take time to give the least significant digits, and unfortunately analog acceleration cannot help to give precise solutions. On the other hand, digital *nonlinear* solvers have a hard time getting a rough-guess solution but polishing a good guess to high precision is cheap. A hybrid analog-digital system combines the analog solver for cheap approximate solutions and the digital solver to obtain high precision.

An analog accelerator has a unique advantage in solving nonlinear equations because it works in continuous time, without steps. In an analog accelerator, nonlinear circuit blocks such as multipliers and adders can evaluate the products and sums for Taylor polynomials, which are useful for finding nonlinear functions and derivatives. As a result, analog accelerators always have up-to-date estimates of nonlinear functions and derivatives. That contrasts with digital, discrete-time systems which must pretend the problem is linear at each step.

Hybrid analog-digital system

Once we set up a way to give approximate solutions to nonlinear systems of equations using an analog accelerator, we had to evaluate its usefulness in workloads that a conventional digital computer would handle. The requirements for a hybrid analog-digital approach include high accuracy and precision in the solution and the ability to handle large problem sizes.

To get high accuracy solutions, we use the analog accelerator in an analog-digital solver system where approximate and low-precision analog solutions are good initial seeds for a digital solver. This scheme is fruitful because a naïve Newton method solver with a poor initial guess spends most of its iterations trying to find the general area of the correct solution. The results in this chapter confirm a digital solver incurs higher time costs as a problem becomes more nonlinear due to this initial search phase. On the other hand, an analog approximate solution allows the hybrid system to fast forward through this phase. Once the hybrid system is in the general area of the solution, the digital solver quickly refines the solution to high precision. The result is the hybrid system solves increasingly nonlinear problems with no significant increase in solution time.

To handle large problem sizes, the digital solver divides nonlinear PDE problems into nonlinear systems of equations problems that can fit in the analog accelerator. Since the analog accelerator focuses only on solving nonlinear systems of equations, the existing PDE space and time discretization techniques stay the same, reducing the amount of reprogramming needed to use analog acceleration. If the nonlinear systems of equations are still too big to fit in the analog accelerator, we use red-black nonlinear Gauss-Seidel to further split the problems.

In this chapter, we make projections on the performance, power, and area of a tiled-out analog accelerator, up to the point where the analog accelerator size matches that of the largest commercial digital chips. Our comparisons and results assume that as the largest possible analog accelerator. We speculate that given the low power consumption and signal fault tolerance of analog chips, it is possible to build larger analog chips and stack them in ways that are impossible with digital chips.

Physical prototype implementation

We tested our ideas on a two-chip system of our physically prototyped analog accelerators. The connectivity for analog signals between chips and between tiles is sparse to match the sparse connectivity of PDEs. Within each tile, the connectivity between analog components is all-to-all to create a variety of nonlinear polynomial functions and their derivatives, giving support for different nonlinear PDEs.

The two-chip system allowed us to test 2D Burgers' equations on a 2×2 grid. We then extrapolated the solution times for analog chips capable of solving larger problems. When an analog accelerator chip capable of solving 16×16 2D Burgers' equations generates approximate solutions to help a GPU running the Newton method, the solution time of the GPU decreases by 5.7× and the energy consumption decreases by 11.6×. These savings are significant since they reduce the costs of the innermost and most intensive kernels of nonlinear PDE solvers.

11.1 Importance and difficulty of solving nonlinear systems of equations

Scientific computing workloads increasingly rely on nonlinear equations to accurately model the real world. A recent informal survey of applied math literature[1] found Newton methods for nonlinear equations to be the most mentioned algorithm, surprisingly topping other numerical stalwarts like

[1]https://nickhigham.wordpress.com/2016/03/29/the-top-10-algorithms-in-applied-mathematics/

matrix factorization, eigenanalysis, Monte-Carlo methods, and FFT. For comparison, an earlier ranking of algorithms from the turn of the century did not mention nonlinear problems at all [36].

The importance of Newton methods may surprise computer architects, since scientific computing workload profiles show sparse linear algebra is by far the most important kernel. Accordingly, we tune architectures such as GPUs for linear algebra [18, 105]. The reality is scientific computing workloads oftentimes call linear algebra subroutines from nonlinear equation solvers. Therefore, improvements for nonlinear solvers would reduce the number of calls to linear algebra solvers altogether. Unfortunately, the software behavior of nonlinear solvers is less regular, making it difficult to devise conventional accelerators for nonlinear problems.

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 [96, 134, 47]. This is more critical than in linear solvers not only to avoid redundant work but to even achieve convergence. As a 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 algorithm iterations (turns on the city grid in my analogy). 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 (using analog multipliers, adders, and lookup tables), reducing the amount of work compared to a digital accelerator. Second, the accelerator uses a different algorithm for solution finding, one that operates continuously in time with no notion of step-by-step operation as required by digital hardware. In our analogy, such an algorithm working in continuous time 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.

11.2 Tutorial: scalar nonlinear root-finding

To solve nonlinear equations in the analog accelerator, we studied how modern digital computers solve nonlinear equations using algorithms. The prototypical digital numerical method for nonlinear equations is the Newton method, which is an iterative method that makes successive guesses at the solution vector with decreasing error until it converges [96, 134, 47].

In practice, the Newton method in a digital computer does not always give a correct result, and needs fine tuning of the algorithm in two aspects. First, the choice of the iterative method's step size for updating the guess is important, as the algorithm needs to often reevaluate the nonlinear function and its derivative. Second, the initial guess to the algorithm needs to be close to the correct solution or else the algorithm does not converge.

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.

Digital classical and damped Newton's

Digital algorithms for nonlinear equations must have a good initial guess to the solution, or else they 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

$$f(u) = u^3 - 1 = 0 \tag{11.1}$$

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 complex 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. This is because Newton's method updates its guess of the solution in discrete steps [152, 175, 86].

One way to reduce the classical Newton method's sensitivity to initial guesses is to use relaxed or damped steps, where the full step size is diminished to a fraction h between 0 and 1, at the cost of increasing the computation time.

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

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 [152, 175, 86]. 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.

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 our prototype analog accelerator chip.

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 [152, 76, 86, 132, 126]. In fact, the continuous Newton method could be considered the natural way of solving nonlinear equations. The continuous Newton method is stated concisely as an ordinary differential equation (ODE).

Digital computers cannot directly solve ODEs and instead approximate them using numerical integration for solving ODEs. For example, the damped Newton method is an Euler's method approximation of the continuous Newton method ODE, and classical Newton's method is an Euler

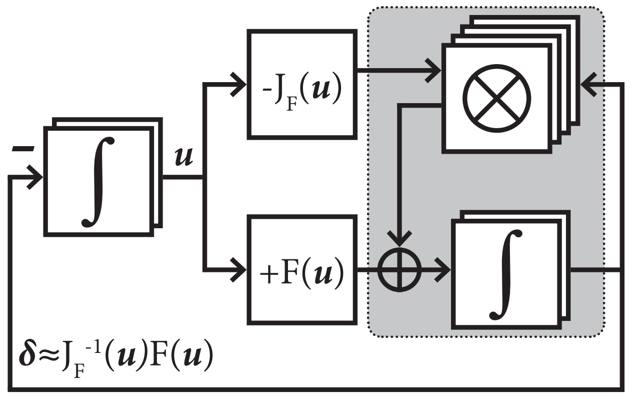


Figure 11.1: Analog circuit for continuous Newton's method. Clockwise from the center left, the major analog function units: integrators for holding the present guess of u, a block for evaluating the Jacobian (the derivative $f'(u)$ in the scalar case), a block (shaded) for finding the Jacobian inverse (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.

approximation with step size of 1. More sophisticated Newton's method solvers use better numerical integration (such as Runge-Kutta or backward differentiation formulas), 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 chapter.

Analog implementation

Figure 11.1 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 $\frac{\mathrm{d}u}{\mathrm{d}t}$, 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

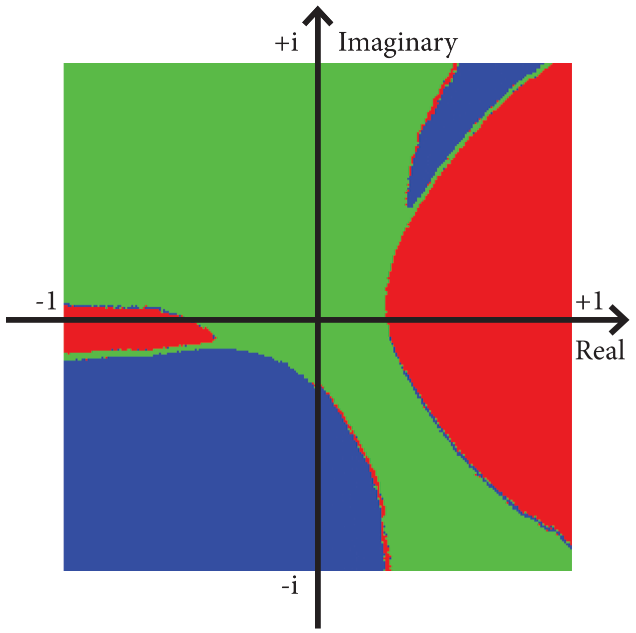


Figure 11.2: The results of continuous Newton's method running on an analog accelerator prototype chip solving Equation 11.1. The colors encode which of the three cubic roots the chip returns, plotted on the complex plane indicating the initial conditions. Each of the 256×256 pixels is one run of the chip. The convergence basins are more contiguous compared to those in classical or damped Newton methods.

in detail in the previous chapter [82, 83].

The quotient is negated and fed to the inputs of the $u(t)$ integrators as $\frac{\mathrm{d}u}{\mathrm{d}t}$, 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:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\frac{f(u)}{f'(u)}$$

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 11.2 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 [86], 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.

Doing the Newton method in continuous-time on an analog accelerator has unique advantages. First, the algorithm always has an up-to-date evaluation of the nonlinear function and its derivative, and since the algorithm runs in continuous time, we do not run into problems having to select a step size. Second, we show in the next section an improvement to the basic Newton's method called homotopy continuation which allows the analog accelerator to select initial guesses more easily.

11.3 Motivation: nonlinear systems of equations

In this section I 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.

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 9.1.

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 $\vec{u}$.

As was the case with just one variable, Newton's method may fail to converge to the correct solution, and requires a damping parameter h to ensure convergence. The damping parameter h smooths out the convergence basins, so that the initial conditions have less of a chaotic effect on the final solution. The approach succeeds so long as the Jacobian is always non-singular, meaning that the determinant of the Jacobian never falls to zero [126].

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. In fact, later in this chapter we will be comparing the analog approach against a digital solver where a

GPU solves th internal linear algebra problem.

In addition to the work solving for $\vec{\delta_p}$, another part of the difficulty in solving nonlinear systems of equations is finding the nonlinear function and the Jacobian matrix on each iteration. In particular, in large size problems the Jacobian is a large sparse matrix, and it becomes prohibitively costly to store the entire Jacobian in memory just to pass it to a linear algebra subroutine. In practice, the elements Jacobian is evaluated just-in-time by passing the linear algebra subroutine a function object that returns Jacobian elements as needed.

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 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:

$$\begin{cases} \rho_0^2 + \rho_0 + \rho_1 = \mathrm{RHS}_0 \\ \rho_1^2 + \rho_1 - \rho_0 = \mathrm{RHS}_1 \end{cases} \tag{11.2}$$

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 11.3, 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

Various **quasi-Newton methods** simplify the Newton method by avoiding finding the Jacobian matrix and/or its inverse. These techniques are not applicable to arbitrary nonlinear systems of equations such as the ones encountered in nonlinear PDEs. Instead, the following techniques are used in *nonlinear least squares* and *nonlinear optimization* problems, where one wants to find the maximum or minimum values of a scalar-valued function $g(\vec{u})$. The set of $\vec{u}$ which maximize or minimize $g(\vec{u})$ are roots of the gradient, in which case

$$F(\vec{u}) = \nabla g(\vec{u})$$

Only when we can make these assumptions about $F(\vec{u})$, can the following quasi-Newton methods be used [96, 134, 47, 138]. For solving nonlinear systems of equations descending from nonlinear PDEs, Newton's method remains the main algorithm.
I list these quasi-Newton methods as background:

- **Gauss-Newton:** Newton's method on the normal equations, useful when there are more unknowns or equations than the other.
- **Levenberg-Marquardt:** Gauss-Newton but with damping on the diagonal elements of the Hessian. Avoids causing the system of equations to become singular, which may result in overfitting.
- **Broyden's:** extension of secant method to multidimensional equations.
- **BFGS:** a way of updating the Hessian inverse without calculating the inverse.

make the algorithm incorrect or inefficient.

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 algorithms that work in continuous time, and have 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 [131, 4, 143].

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:

$$S(\vec{\rho}) = \begin{cases} \rho_0^2 - 1 = 0 \\ \rho_1^2 - 1 = 0 \end{cases} \tag{11.3}$$

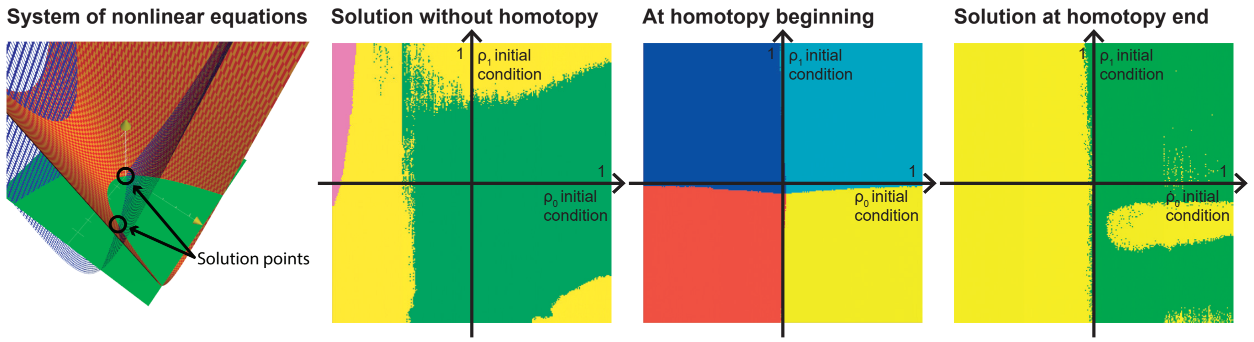


Figure 11.3: **Far left:** Visualization of Equation 11.2. The two equations are surfaces (blue mesh and red checkerboard) formed by parabolas swept along straight lines. The root finding problem entails finding where two surfaces intersect at the $z = 0$ (solid green) plane. The RHS constants in the equations shift the surfaces up and down, so there can zero or several such solutions. In the next three panels we solve without and using homotopy continuation. **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 11.2. 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 11.3. 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 11.2. Compared to naive Newton's method, all choices of initial conditions in the homotopy method lead to one correct solution or another. The convergence basins are more contiguous, indicating less sensitivity to the initial guess.

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 11.2, 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 λ 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 λ until $\lambda = 1$. At each moment while incrementing λ, 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 [33, 34, 126, 143], 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 11.3. 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.

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 [40, 41]. 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 [24, 108, 8, 5]. The analog acceleration techniques in this paper can extend those methods due to its fundamental energy efficiency in the low bit precision regime [147, 43].

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 11.1: Effect of Reynolds number on Burgers' and Navier-Stokes equations. Larger Reynolds numbers result in more nonlinear and difficult problems. When the Reynolds number is large (a high Mach, inviscid system), the effect of diffusion is low, and the PDEs are dominated by first-order advection effects, and the system is quasilinear. When the Reynolds number is small (a low Mach, viscous system), the effect of diffusion is large, and the PDEs are more second-order parabolic in character, and the system is semilinear.

This chapter 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 attempted 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 11.6.

11.4 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 chapter thus far.

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 chapter focuses on the viscous Burgers' equation, a nonlinear PDE. The Burgers' equation is the subset of the Navier-Stokes equations for modeling fluids asserting that momentum is conserved [59, 160, 179] . It is the core nonlinear heart of the Navier-Stokes equations.

The viscous Burgers' equation has the form:

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} - \frac{1}{\mathrm{Re}}\nabla^2 \vec{u} = \mathrm{RHS} \tag{11.4}$$

$$\begin{cases} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \frac{1}{\mathrm{Re}}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = \mathrm{RHS}_0 \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \frac{1}{\mathrm{Re}}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) = \mathrm{RHS}_1 \end{cases} \tag{11.5}$$

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. This equation has the characteristics of Equation 7.6, but with an additional second-order viscosity term, making the equation parabolic and similar to Equation 7.3.

We are interested in 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 chapter to other classes of PDEs, which I discuss in Section 11.7. 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 11.1. Higher Reynolds numbers result in more nonlinear systems of equations, and increases the difficulty of solving this PDE.

Space discretization

With an example nonlinear PDE in hand, in the next two subsections I 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 that 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

$$
\begin{aligned}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \frac{1}{\mathrm{Re}}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = RHS \\
\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \frac{1}{\mathrm{Re}}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) = RHS \\
\frac{\partial u_{i,j}}{\partial t} + u_{i,j}\frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + v_{i,j}\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} - \frac{1}{\mathrm{Re}}\Big(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}\Big) = 0 \\
\frac{\partial v_{i,j}}{\partial t} + u_{i,j}\frac{v_{i+1,j} - v_{i-1,j}}{2\Delta x} + v_{i,j}\frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y} - \frac{1}{\mathrm{Re}}\Big(\frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{\Delta x^2} + \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{\Delta y^2}\Big) = 0 \\
\frac{\partial u_{i,j}}{\partial t} = -u_{i,j}\frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} - v_{i,j}\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} + \frac{1}{\mathrm{Re}}\Big(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}\Big) \\
\frac{\partial v_{i,j}}{\partial t} = -u_{i,j}\frac{v_{i+1,j} - v_{i-1,j}}{2\Delta x} - v_{i,j}\frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y} + \frac{1}{\mathrm{Re}}\Big(\frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{\Delta x^2} + \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{\Delta y^2}\Big)
\end{aligned}
$$

Figure 11.4: Space discretization of a 2d viscous Burgers' nonlinear PDE.

value at one place in space as a function of time. In this chapter 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 I discuss in Section 11.7.

Applying space discretization to a PDE results in a system of ODEs, which is discrete in space but continuous in time. Figure 11.4 shows an example of doing so for the Burgers' equation. We handle the equations' time evolution next.

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 [91, 92, 85, 57, 173, 11, 180]. 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 chapter we use Crank-Nicolson, an implicit method which offers second-order accuracy, to decompose the 2D Burgers' equation into a nonlinear system of equations [42]. Section 11.7 discusses how our approach generalizes to other time stepping schemes.

Viscous Burgers' PDE discretization

Using the techniques in the past two sections, we take Equation 11.4, the 2D viscous Burgers' equation, and apply second-order central finite difference and second-order Crank-Nicolson time stepping [42]. Then, we make isotropic assumptions about the relative size of the space and time grid points to simplify the problem. We choose values for Δt, Δx, and Δy so these coefficients are eliminated, for the sake of clarity.

The resulting stencil consists of a nonlinear system of equations and its Jacobian, which is shown in Figure 11.5. The stencil for the general case can be found in the literature [59, pg.172]. These two sets of mathematical expressions are what we need to program into the analog accelerator circuit shown in Figure 11.1. 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.

11.5 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 2×2 grid size due to prototype size constraints, and present measured accuracy results.

Programming and data interface

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

Below, the nonlinear systems of equations of a viscous Burgers' nonlinear PDE. The unknown variables are grouped left of the equals sign while all constant boundary conditions are grouped on the right. The variables $\vec{u} = \{u, v\}$ are the x and y velocity components of a 2-dimensional, 2×2 grid. In these equations, superscripts are time steps, and subscripts are spatial grid points. In total the problem is a system of eight nonlinear equations and eight unknowns.

$$F(\vec{u}) =$$

$$\begin{cases}
+u_{0,0}^{n+1}(u_{1,0}^{n+1} + 1 + \frac{4}{\mathrm{Re}} - u_{-1,0}^{n+1}) + v_{0,0}^{n+1}(u_{0,1}^{n+1} - u_{0,-1}^{n+1}) - \frac{1}{\mathrm{Re}}(u_{1,0}^{n+1} + u_{-1,0}^{n+1} + u_{0,1}^{n+1} + u_{0,-1}^{n+1}) = -u_{0,0}^{n}(u_{1,0}^{n} - 1 + \frac{4}{\mathrm{Re}} - u_{-1,0}^{n}) - v_{0,0}^{n}(u_{0,1}^{n} - u_{0,-1}^{n}) + \frac{1}{\mathrm{Re}}(u_{1,0}^{n} + u_{-1,0}^{n} + u_{0,1}^{n} + u_{0,-1}^{n}) \\
+v_{0,0}^{n+1}(v_{0,1}^{n+1} + 1 + \frac{4}{\mathrm{Re}} - v_{0,-1}^{n+1}) + u_{0,0}^{n+1}(v_{1,0}^{n+1} - v_{-1,0}^{n+1}) - \frac{1}{\mathrm{Re}}(v_{1,0}^{n+1} + v_{-1,0}^{n+1} + v_{0,1}^{n+1} + v_{0,-1}^{n+1}) = -v_{0,0}^{n}(v_{0,1}^{n} - 1 + \frac{4}{\mathrm{Re}} - v_{0,-1}^{n}) - u_{0,0}^{n}(v_{1,0}^{n} - v_{-1,0}^{n}) + \frac{1}{\mathrm{Re}}(v_{1,0}^{n} + v_{-1,0}^{n} + v_{0,1}^{n} + v_{0,-1}^{n}) \\
+u_{1,0}^{n+1}(u_{2,0}^{n+1} + 1 + \frac{4}{\mathrm{Re}} - u_{0,0}^{n+1}) + v_{1,0}^{n+1}(u_{1,1}^{n+1} - u_{1,-1}^{n+1}) - \frac{1}{\mathrm{Re}}(u_{2,0}^{n+1} + u_{0,0}^{n+1} + u_{1,1}^{n+1} + u_{1,-1}^{n+1}) = -u_{1,0}^{n}(u_{2,0}^{n} - 1 + \frac{4}{\mathrm{Re}} - u_{0,0}^{n}) - v_{1,0}^{n}(u_{1,1}^{n} - u_{1,-1}^{n}) + \frac{1}{\mathrm{Re}}(u_{2,0}^{n} + u_{0,0}^{n} + u_{1,1}^{n} + u_{1,-1}^{n}) \\
+v_{1,0}^{n+1}(v_{1,1}^{n+1} + 1 + \frac{4}{\mathrm{Re}} - v_{1,-1}^{n+1}) + u_{1,0}^{n+1}(v_{2,0}^{n+1} - v_{0,0}^{n+1}) - \frac{1}{\mathrm{Re}}(v_{2,0}^{n+1} + v_{0,0}^{n+1} + v_{1,1}^{n+1} + v_{1,-1}^{n+1}) = -v_{1,0}^{n}(v_{1,1}^{n} - 1 + \frac{4}{\mathrm{Re}} - v_{1,-1}^{n}) - u_{1,0}^{n}(v_{2,0}^{n} - v_{0,0}^{n}) + \frac{1}{\mathrm{Re}}(v_{2,0}^{n} + v_{0,0}^{n} + v_{1,1}^{n} + v_{1,-1}^{n}) \\
+u_{0,1}^{n+1}(u_{1,1}^{n+1} + 1 + \frac{4}{\mathrm{Re}} - u_{-1,1}^{n+1}) + v_{0,1}^{n+1}(u_{0,2}^{n+1} - u_{0,0}^{n+1}) - \frac{1}{\mathrm{Re}}(u_{1,1}^{n+1} + u_{-1,1}^{n+1} + u_{0,2}^{n+1} + u_{0,0}^{n+1}) = -u_{0,1}^{n}(u_{1,1}^{n} - 1 + \frac{4}{\mathrm{Re}} - u_{-1,1}^{n}) - v_{0,1}^{n}(u_{0,2}^{n} - u_{0,0}^{n}) + \frac{1}{\mathrm{Re}}(u_{1,1}^{n} + u_{-1,1}^{n} + u_{0,2}^{n} + u_{0,0}^{n}) \\
+v_{0,1}^{n+1}(v_{0,2}^{n+1} + 1 + \frac{4}{\mathrm{Re}} - v_{0,0}^{n+1}) + u_{0,1}^{n+1}(v_{1,1}^{n+1} - v_{-1,1}^{n+1}) - \frac{1}{\mathrm{Re}}(v_{1,1}^{n+1} + v_{-1,1}^{n+1} + v_{0,2}^{n+1} + v_{0,0}^{n+1}) = -v_{0,1}^{n}(v_{0,2}^{n} - 1 + \frac{4}{\mathrm{Re}} - v_{0,0}^{n}) - u_{0,1}^{n}(v_{1,1}^{n} - v_{-1,1}^{n}) + \frac{1}{\mathrm{Re}}(v_{1,1}^{n} + v_{-1,1}^{n} + v_{0,2}^{n} + v_{0,0}^{n}) \\
+u_{1,1}^{n+1}(u_{2,1}^{n+1} + 1 + \frac{4}{\mathrm{Re}} - u_{0,1}^{n+1}) + v_{1,1}^{n+1}(u_{1,2}^{n+1} - u_{1,0}^{n+1}) - \frac{1}{\mathrm{Re}}(u_{2,1}^{n+1} + u_{0,1}^{n+1} + u_{1,2}^{n+1} + u_{1,0}^{n+1}) = -u_{1,1}^{n}(u_{2,1}^{n} - 1 + \frac{4}{\mathrm{Re}} - u_{0,1}^{n}) - v_{1,1}^{n}(u_{1,2}^{n} - u_{1,0}^{n}) + \frac{1}{\mathrm{Re}}(u_{2,1}^{n} + u_{0,1}^{n} + u_{1,2}^{n} + u_{1,0}^{n}) \\
+v_{1,1}^{n+1}(v_{1,2}^{n+1} + 1 + \frac{4}{\mathrm{Re}} - v_{1,0}^{n+1}) + u_{1,1}^{n+1}(v_{2,1}^{n+1} - v_{0,1}^{n+1}) - \frac{1}{\mathrm{Re}}(v_{2,1}^{n+1} + v_{0,1}^{n+1} + v_{1,2}^{n+1} + v_{1,0}^{n+1}) = -v_{1,1}^{n}(v_{1,2}^{n} - 1 + \frac{4}{\mathrm{Re}} - v_{1,0}^{n}) - u_{1,1}^{n}(v_{2,1}^{n} - v_{0,1}^{n}) + \frac{1}{\mathrm{Re}}(v_{2,1}^{n} + v_{0,1}^{n} + v_{1,2}^{n} + v_{1,0}^{n})
\end{cases}$$

Below, the Jacobian matrix of the 2D 2x2 viscous Burgers' nonlinear PDE. Each row of the Jacobian corresponds to the x-velocity or y-velocity at each grid point.

$$J_F(\vec{u}) =$$

$$\begin{bmatrix}
u_{1,0} + 1 + \frac{4}{\mathrm{Re}} - u_{-1,0} & u_{0,0} - \frac{1}{\mathrm{Re}} & v_{0,0} - \frac{1}{\mathrm{Re}} & 0 & u_{0,1} - u_{0,-1} & 0 & 0 & 0 \\
-u_{1,0} - \frac{1}{\mathrm{Re}} & u_{2,0} + 1 + \frac{4}{\mathrm{Re}} - u_{0,0} & 0 & v_{1,0} - \frac{1}{\mathrm{Re}} & 0 & u_{1,1} - u_{1,-1} & 0 & 0 \\
-v_{0,1} - \frac{1}{\mathrm{Re}} & 0 & u_{1,1} + 1 + \frac{4}{\mathrm{Re}} - u_{-1,1} & u_{0,1} - \frac{1}{\mathrm{Re}} & 0 & 0 & u_{0,2} - u_{0,0} & 0 \\
0 & -v_{1,1} - \frac{1}{\mathrm{Re}} & -u_{1,1} - \frac{1}{\mathrm{Re}} & u_{2,1} + 1 + \frac{4}{\mathrm{Re}} - u_{0,1} & 0 & 0 & 0 & u_{1,2} - u_{1,0} \\
v_{1,0} - v_{-1,0} & 0 & 0 & 0 & v_{0,1} + 1 + \frac{4}{\mathrm{Re}} - v_{0,-1} & u_{0,0} - \frac{1}{\mathrm{Re}} & v_{0,0} - \frac{1}{\mathrm{Re}} & 0 \\
0 & v_{2,0} - v_{0,0} & 0 & 0 & -u_{1,0} - \frac{1}{\mathrm{Re}} & v_{1,1} + 1 + \frac{4}{\mathrm{Re}} - v_{1,-1} & 0 & v_{1,0} - \frac{1}{\mathrm{Re}} \\
0 & 0 & v_{1,1} - v_{-1,1} & 0 & -v_{0,1} - \frac{1}{\mathrm{Re}} & 0 & v_{0,2} + 1 + \frac{4}{\mathrm{Re}} - v_{0,0} & u_{0,1} - \frac{1}{\mathrm{Re}} \\
0 & 0 & 0 & v_{2,1} - v_{0,1} & 0 & -v_{1,1} - \frac{1}{\mathrm{Re}} & -u_{1,1} - \frac{1}{\mathrm{Re}} & v_{1,2} + 1 + \frac{4}{\mathrm{Re}} - v_{1,0}
\end{bmatrix}$$

Figure 11.5: The nonlinear system of equations and Jacobian matrix of a 2d 2-by-2 grid point viscous Burgers' nonlinear PDE.

```
/*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 11.6: Analog accelerator object-oriented C++ code sample.

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 11.1 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 11.6. 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.

In our programming model, C++ classes represent components such as one nonlinear equation or one row of the Jacobian matrix. Each class exposes only the analog interfaces that need to be connected with other submodules. The classes also offer functions that change parameters such as initial conditions, coefficients, and constants. When a class object is instantiated, the instantiating program gives the object an allocation of analog hardware to physically implement the needed analog datapath. Then, the object constructor writes a stream of bits to the analog accelerator setting up the object. Destroying the object likewise frees the analog resources to participate in other calculations.

The object-oriented style of programming goes beyond writing a single nonlinear equation or one row of the Jacobian matrix. We create higher-level classes that represent one node variable. Yet higher-level classes that represent all of the x-velocity and all of the y-velocity variables $\vec{u}$ and $\vec{v}$. Finally, we connect the nonlinear system of equations, the Jacobian matrix, and the integrators storing the variables together to form a circuit for continuous Newton's method. Additional

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 (μW)	284	152	188	139

Table 11.2: Summary of analog chip component use for each PDE variable with area and power model from [66, 67, 82, 83].

connections between blocks can be made to export variables between analog accelerator chips, off chip, and into analog-to-digital converters.

Board and chip hardware mapping

We use a circuit board with two analog accelerator chips (each with 16 integrators) 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 6.1, 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 11.2 shows the hardware needed to implement each mathematical component.

This case study highlights the advantage of the Gilbert-cell implementation of multipliers compared to memristor multipliers. Re-programing memristors to realize different coefficients is time consuming as the coefficients need to be calibrated. Re-programing Gilbert-cell multipliers, on the other hand, has identical time cost as writing to a memory-mapped device, so our design choice of using Gilbert-cell multipliers permits solving different problems with new parameters without

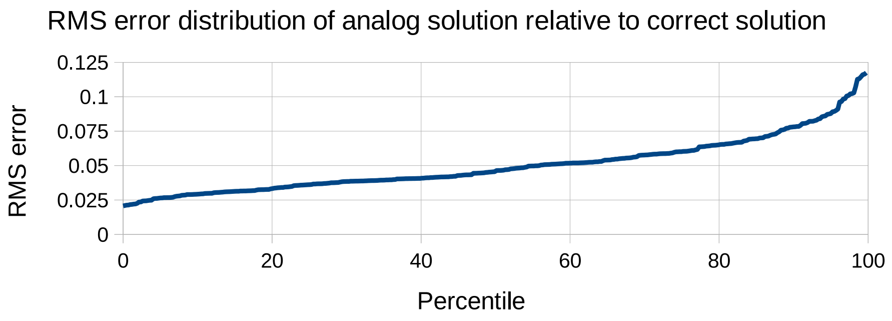


Figure 11.7: Distribution of analog solution error for 400 randomly generated problems.

taking time to recalibrate the multipliers.

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 $\vec{u}$ 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.

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:

$$\sqrt{\frac{\sum_N (u_a - u_d)^2}{N}} \tag{11.6}$$

Where N is the number of elements in the analog and digital solutions. Figure 11.7 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. This limited accuracy is potentially a shortcoming in using analog acceleration for solving nonlinear systems of equations. In the case of accelerating linear algebra, a higher-precision and higher-accuracy result can be obtained by repeatedly invoking the analog accelerator [82]. No such precision and accuracy-building scheme exists in nonlinear problems, due to the fact that the residual cannot be scaled back up to the dynamic range of the accelerator.

11.6 Design space exploration of scaled-up analog accelerators for nonlinear systems of equations

So far, the case studies of Sections 11.2, 11.3 and the 2×2 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.

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 16×16, corresponding to a large nonlinear system of equations with a 512×512 sparse Jacobian matrix.

Problem setup: We solve randomly generated 2D Burgers' equations with grid sizes of 2×2, 4×4, 8×8, and 16×16. The initial and (Dirichlet) boundary conditions for the problems are

2D Burgers' solver size	**Chip area** (mm^2)	**Power use** (mW)
1×1	1.4	1.5
2×2	5.5	6.1
4×4	22.0	24.4
8×8	88.1	97.7
16×16	352.4	390.7

Table 11.3: 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.

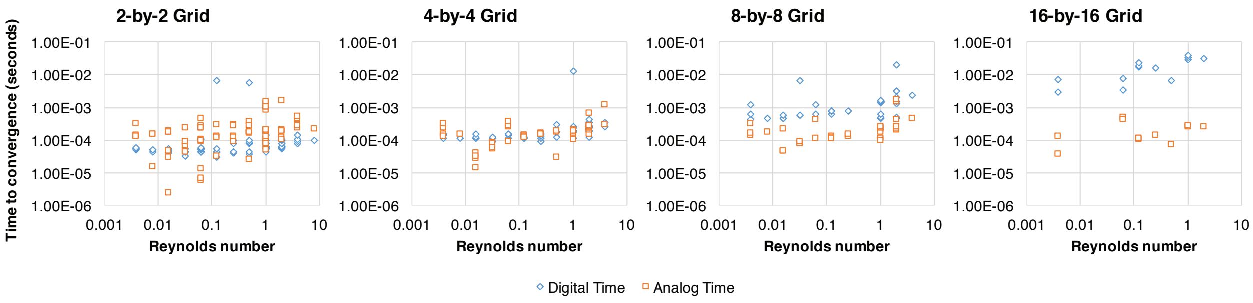


Figure 11.8: Time to convergence for digital and analog solvers.

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 11.6 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.

Detailed function profiling of the digital solver shows it spends at least 95% of its time in finding $\vec{\delta_p} = J_F^{-1}(\vec{u_p})F(\vec{u_p})$, using QR factorization, a numerical linear algebra algorithm. The remainder of the time is spent evaluating the function $F(\vec{u_p})$ and building the Jacobian matrix $J_F^{-1}(\vec{u_p})$.

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 [3]. 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 2×2 analog accelerator is normalized to match the measured solution time of the physical analog accelerator (with 200KHz integrator bandwidth). 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 11.3 shows that an analog accelerator for 16×16 problems is roughly the same size as CPU dies, while power density is about 400× 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 [154]. These unique strengths of analog accelerators may permit die sizes and stacking techniques otherwise impossible in digital designs. For now we limit ourselves to 16×16 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 11.8 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, the PDE becomes more nonlinear and hyperbolic in character. As shown in the definition of the Jacobian in Figure 11.5, 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

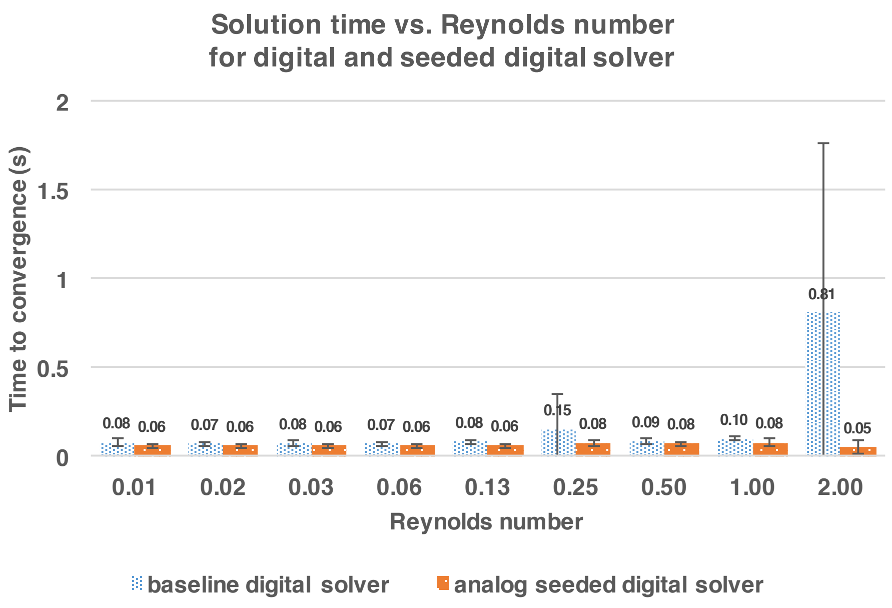


Figure 11.9: Time to convergence for digital and seeded digital solvers to double-precision floating point epsilon precision.

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 4×4 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 16×16 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.

Analog approximation as digital initial guess

We take the findings from the previous experiment and use the largest modeled analog accelerator design, capable of approximately solving 16×16 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 11.2, 11.3, and does so in negligible time compared to the digital solver for the same accuracy, according to Figure 11.8. 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.

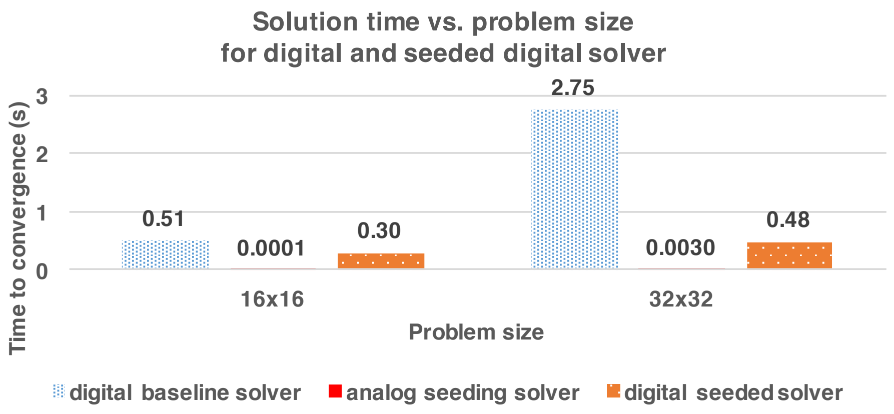
Figure 9 shows the solution time of a baseline digital solver compared to a seeded digital solver which benefits from the low-precision solution of an analog accelerator. The average solution time over 16 trials for both is plotted against various choices of Reynolds number for the problem, which influences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

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 16×16 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 32×32, resulting in a 2048×2048 sparse Jacobian, the average multiprocessor activity increases to 95%.



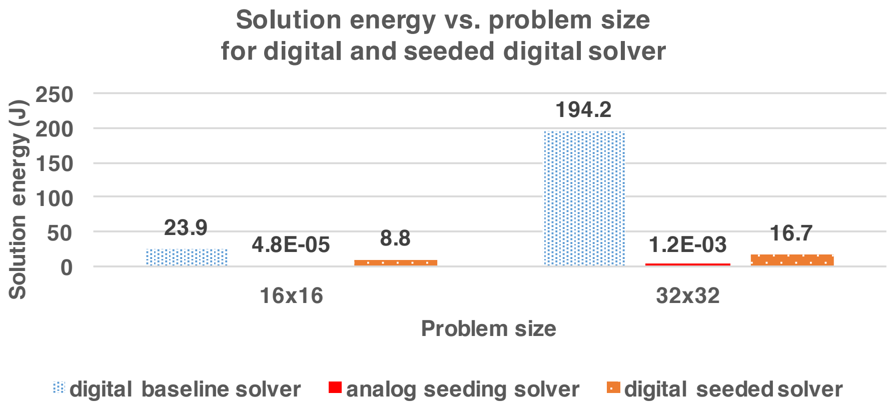


Figure 11.10: Time and energy for solution for digital and seeded digital solvers running on a GPU.

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 16×16 problems due to area constraints. In other words we need a way to solve a subset of a (potentially too large) nonlinear systems of equations. We use red-black nonlinear Gauss-Seidel to split the 32×32 problems to fit. In Gauss-Seidel iterations, we treat some unknowns as live and others as fixed. In any given iteration, we solve only for the live unknowns. Then, we reverse the roles of the live and fixed unknowns, solve again, and repeat until all values are unchanging. The Gauss-Seidel method is known to converge if the (Jacobian) matrix is diagonally dominant, which is true for sparse matrices coming from PDE problems. 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 [62, pg.291]. Digital architectures use the same decomposition and parallelization techniques to make subproblems fit in CPU caches or split work among nodes [75, 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 11.10 shows seeding the GPU decreases the solution time for 32×32 Burgers' equations by 5.7$\times$, and the energy by 11.6$\times$. 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 9.1.

11.7 Extensions for other PDEs

Now I 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. We chose to use central finite difference for space discretization and Crank-Nicolson for time stepping. The 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. The cause of the difference in difficulty is in semilinear PDEs the unknown variables appear in the Jacobian matrix only in the diagonal terms, while in quasilinear PDEs the unknowns appear in off-diagonal terms of the Jacobian matrix, making the Jacobian matrix more costly to generate.

Beyond quasilinear PDEs are fully-nonlinear PDEs, which permit the partial differential operators themselves be part of nonlinear functions. An example of a fully-nonlinear PDEs is the Eikonal equation in optics. 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 $\sin(u)$. Outside of polynomials, nonlinear functions can be transcendentals, which do not have a viable scaling scheme. 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. For example, the pressure and velocity fields in the Navier-Stokes equations are solved separately, yielding PDE subproblems with fewer dimensions. Furthermore, three-dimensional velocity field problems in Navier-Stokes solvers are decomposed into one-dimensional problems using techniques such as alternating directions implicit methods.

So solving two-dimensional PDEs in analog accelerators provides reasonable coverage of practical solvers.

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 chapter 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 chapter.

11.8 Summary

This chapter demonstrates how hybrid analog-digital computing can be used to accelerate solving nonlinear systems of equations and partial differential equations. We tested our ideas on a multi-chip system of our 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 chapter is we get tangible performance and efficiency improvements by seeking out problems where digital stumbles and analog can succeed.

Nonlinear is analog killer app

This chapter shows analog acceleration has unique advantages in tackling nonlinear problems. That is because the analog accelerator works in continuous time, so that nonlinear functions and derivatives are continuously reevaluated. In comparison, discrete time digital computers must pretend the problem is linear at each time step. If this linear assumption causes problems, the digital computer must invest more iterations and computation time until the linear approximation is good enough. Using an analog accelerator to solve the same nonlinear problem sidesteps these problems because the nonlinear behavior of the analog circuit better matches the nonlinear problem description.

How to do more problems types in analog accelerators

A challenge in using analog accelerator architectures is finding an interface to separate analog and digital computing models.

This thesis paves the way to finding more problems for the analog accelerator. We do so by converting iterative numerical methods that are workhorses of scientific computing into ODEs [33, 34, 1, 19], which we then solve in the analog accelerator. The critical idea is to use the transient behavior of an analog resistor-capacitor circuit, and use that circuit to compute the intermediate results of iterative numerical methods. Essentially, we draw an equivalence between iterative numerical methods in applied math, algorithms in computer science, and transient circuit dynamics in electrical engineering. These ODE *continuous algorithms* may be the missing analog-digital program partitioning for analog accelerators.

In this chapter, I give three examples of continuous ODEs that solve numerical problems: continuous gradient descent for linear algebra, along with continuous Newton's method and homotopy continuation, both for nonlinear algebra. These examples give us clues on how to find more problems for analog accelerators in the future. For example, iterative numerical methods for important problems such as eigenanalysis [45, 68] and linear programming [58, 23, 172] all have continuous time versions.

Doing iterative numerical methods in an analog accelerator has three advantages:

1. Analog accelerators work nicely in hybrid analog-digital architectures for iterative methods by giving cheap approximate solutions which a conventional digital computer can then refine. That is possible because iterative numerical methods all work by giving progressively more correct guesses for the problem solution.
2. Analog accelerators work in continuous time, without discrete steps, avoiding the choice of step sizes, which control how fast iterative numerical methods updates solution guesses. The choice of these step sizes is often difficult and needs fine tuning.
3. When we use analog accelerators to solve iterative numerical methods, the solution output of the analog accelerator is the final, converged output. Because the output is steady, we can sample the solution with high precision, making it easier to connect the analog accelerator with a digital computer.

How to do more work in an analog accelerator

This chapter shows how we can make the analog accelerator do more work, so the ratio of analog computation vs. analog / digital conversion is higher, making analog acceleration more worthwhile. The trick is to have an analog accelerator equivalent of inner loops. For example, we invoked an analog inner loop for linear algebra inside the analog Newton method solver (Figure 11.1). The Newton method solver is itself an inner loop for homotopy continuation (Section 11.3). We implement these inner loops by building subcircuits which converge faster than the overall circuit. Using this trick, we can nest other types of iterative numerical methods, in the same way digital algorithms compose different subroutines.

Part V

Conclusion

Chapter 12

Conclusion & Research Directions

12.1 Conclusion

This thesis revisited hybrid analog-digital computing in support of diverse modern workloads, such as those in the Berkeley Dwarfs [6] taxonomy (Section 3.1). My team and I demonstrated solving a variety of scientific computing problems, including stochastic ordinary differential equations (ODEs) (Chapter 8), partial differential equations (PDEs), linear algebra (Chapter 10), and nonlinear systems of equations (Chapter 11), in analog. We solved these problems on a system of multiple prototype analog accelerator chips built by a team at Columbia University (see Part II). On that team I made contributions toward programming the chips (Section 5.1), building the digital interface (Sections 5.2 and 6.1), and validating the chips' functionality (Section 6.2). The analog accelerator chip is intended for use in conjunction with a conventional digital host computer.

The appeal and motivation for using an analog accelerator is efficiency and performance, but it comes with limitations in accuracy and problem sizes that we have to work around.

The first problem is how to do problems in this unconventional computation model (Section 3.1). Scientific computing phrases problems as differential equations and algebraic equations. Differential equations are a continuous view of the world (Chapter 7), while algebraic equations are a discrete one (Chapter 9). The secret to using the analog accelerator for modern workloads is that these two viewpoints are interchangeable. Typically, applied mathematicians turn differential equations into algebraic ones, in part to match the dominant discrete digital model of computation. Less well appreciated is that the reverse is also true: linear algebra and nonlinear systems of equations can be solved as differential equations, such that the dynamics of the differential equations interpolates the intermediate values of step-by-step algorithms! Iterative numerical linear algebra (Section 10.2) and Newton's methods (Sections 11.2 and 11.3) algorithms for algebraic equations can be viewed as discrete approximations of ODEs. A hybrid analog-digital computer architecture can solve those ODEs, in turn solving the underlying linear and nonlinear algebra problems for many workloads.

The second problem is how to solve large problems using hybrid analog-digital computing (Section 3.3). The reason the analog computation model can't handle large problems is it gives up step-by-step discrete-time operation, instead allowing variables to evolve smoothly in continuous time. To make that happen the analog accelerator works by chaining hardware for mathematical operations end-to-end. During computation analog data flows through the hardware with no overheads in control logic and memory accesses (Section 2.1). The downside is then the needed hardware size grows alongside problem sizes. The trick to overcome this limitation is to focus on problems that have divide-and-conquer algorithms that break problems into smaller sizes. We demonstrate this trick for three problem types: 1. For stochastic ODEs, the solution statistics come from an ensemble of independent analog solutions, each using a different sample of analog noise (Section 8.5). 2. For linear elliptic PDEs, we use the multigrid method to break the problem into a hierarchy of coarse and fine grids. Then the analog accelerator approximately solves the PDE at every level of discretization resolution. The coarse solutions serve as initial guesses for fine solutions (Section 10.6). 3. For nonlinear parabolic PDEs, we use the red-black Gauss-Seidel method to likewise break a nonlinear system of equations into subproblems that fit in the analog accelerator. The digital host does a few iterations of the Gauss-Seidel outer loop to make sure the subproblem solutions are in agreement (Section 11.6).

The third problem is how to get accurate solutions using hybrid analog-digital computing (Section 3.2). The reason the analog computation model gives less accurate solutions is it gives up representing numbers as digital binary numbers, and instead uses the full range of analog voltage and current to represent real numbers. Encoding data in analog signals gives an energy efficiency advantage as long as the analog data precision is limited (Section 2.2). While the analog accelerator alone may be useful for energy-constrained applications where inputs and outputs are imprecise, we are more interested in using analog in conjunction with digital for precise solutions. The trick is to solve problems where low-precision guesses are useful for conventional digital algorithms. This trick works for linear algebra, where the analog accelerator finds an approximate solution, and the digital host finds the residual and rescales the problem for another round in analog (Section 10.4). The trick also works for nonlinear systems of equations, where an analog approximate solution is a good initial guess for a digital Newton's method solver (Section 11.5). Hybrid analog-digital computation for accurate solutions is possible for algebraic equations but not for differential ones, a strong indication that mine is the right approach.

12.2 Future research directions

Analog accelerator solutions for differential algebraic equations (DAEs) and integral equations are two potential applications for analog accelerators. Both are mathematical logical extensions beyond the differential equations and algebraic equations I focused on in this thesis. Furthermore, analog support for some problem and algorithm categories in the Berkeley Dwarfs taxonomy deserves further research.

Analog accelerator applications in differential algebraic equations

DAEs are systems of equations defined by a mix of differential equations and algebraic equations [135, 21, 7]. We can think of them as extremely stiff differential equations—so stiff, in fact, the Jacobian matrix interrelating derivatives and the variables is singular. As such, traditional methods for solving either differential equations or systems of algebraic equations are alone not enough to solve DAEs.

Several widespread engineering applications have to solve DAEs. One such example is robotic arm and leg kinematics [87]. When robotic limbs are in a fully extended position, the kinematics of the limb loses a degree of freedom, meaning that a change of input to one of the joints has no influence on the final position of a hand or foot. In that situation, the ODE describing the hand or foot position has a singular Jacobian matrix, causing the ODE to become a DAE. Trying to solve the DAE using regular ODE solvers will cause the controller to fail, and the limb may move incorrectly. Another domain of application for DAEs is the optimal control of induction motors [109].

Analog co-processing is potentially a more reliable way to solve stiff systems of ODEs, due to the lack of a notion of discrete time steps. Further research is needed to understand how DAEs can be mapped into analog accelerators.

Analog accelerator applications in Berkeley Dwarfs

Some problems and algorithms in the Berkeley Dwarfs taxonomy (Section 3.1) deserve research to see if analog accelerators are applicable.

This thesis explored analog accelerator applications in several areas of continuous mathematics, such as sparse and dense matrix (Part IV), Monte Carlo (Chapter 8), and structured and unstructured grid methods (Part IV). Other researchers have delved into analog spectral methods [78] while N-body seems better suited for digital accelerators.

Additional research is needed to find or rule out analog accelerator applications in important dynamic programming and graphical methods problems such as language processing (Viterbi algorithm and hidden Markov model) and optimal control. Analog accelerator applications in mostly discrete problems, such as Boolean satisfiability problems and sorting, deserve attention too. Even though those approaches are theoretical and outlandish (given the mismatch between discrete problems and continuous computation model), they may offer new directions for approximate solutions for discrete problems.

12.3 Broader view

In the past 10 years computer architecture research has moved to more heterogeneity and less adherence to conventional abstractions. Scientists and engineers hold an unshakable belief that computing holds keys to unlocking humanity's Grand Challenges. Acting on that belief they have looked deeper into computer architecture to find specialized support for their applications. Likewise, computer architects have looked deeper into circuits and devices in search of untapped performance and efficiency. The lines between computer architecture layers—applications, algorithms, architectures, microarchitectures, circuits and devices—have blurred. Against this backdrop, a menagerie of computer architectures are on the horizon, ones that forgo basic assumptions about computer hardware, and require new thinking of how such hardware supports problems and algorithms.

As we enter the post-Moore's law era of computing, unconventional architectures will offer specialized models of computation that uniquely support specific problem types [168, 88]. Two prominent examples are using deep neural networks to support pattern recognition, and using quantum computers for simulating quantum systems. In this thesis I show that another specialized, unconventional architecture is to use analog accelerators to solve problems in scientific computing. As recent computer architecture conference programs show, these unconventional architectures are now commercially relevant. Computer architecture researchers will discover other important models of computation in the future. The work in this chapter is an example of the discovery process, implementation, and evaluation of how an unconventional architecture supports a specialized workload.

Bibliography

[1] Absil, P.-A. Continuous-time systems that solve computational problems. *IJUC 2* (2006), 291–304.

[2] Achour, S., Sarpeshkar, R., and Rinard, M. C. Configuration synthesis for programmable analog devices with Arco. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2016), PLDI '16, ACM, pp. 177–193.

[3] Ahnert, K., and Mulansky, M. Odeint - solving ordinary differential equations in C++. *AIP Conference Proceedings 1389*, 1 (2011), 1586–1589.

[4] Allgower, E., and Georg, K. *Numerical Continuation Methods: An Introduction.* Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2012.

[5] Anzt, H., Heuveline, V., and Rocker, B. 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* (Berlin, Heidelberg, 2011), VECPAR'10, Springer-Verlag, pp. 58–70.

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

[7] Ascher, U. M., and Petzold, L. R. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, vol. 61. Siam, 1998.

[8] Baboulin, M., Buttari, A., Dongarra, J., Kurzak, J., Langou, J., Langou, J., Luszczek, P., and Tomov, S. Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications 180*, 12 (12 2009), 2526–2533.

[9] Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kanschat, G., Kronbichler, M., Maier, M., Turcksin, B., and Wells, D. The `deal.II` library, version 8.4. *Journal of Numerical Mathematics 24* (2016).

[10] Beatson, R., and Greengard, L. A short course on fast multipole methods.

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

[12] Betts, J. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, second ed. Society for Industrial and Applied Mathematics, 2010.

[13] Bingulac, S., Milovanovic, M., Lazarevic, B., and Lolic, B. A hybrid approach to automatic rescaling on general purpose analog computers. *Mathematics and Computers in Simulation 9*, 4 (1967), 188 – 194.

[14] Bloch, A. M., and Rojo, A. G. *Sorting: The Gauss Thermostat, the Toda Lattice and Double Bracket Equations,.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 35–48.

[15] Bojnordi, M. N., and Ipek, E. Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (March 2016), pp. 1–13.

[16] Bojnordi, M. N., and Ipek, E. Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *2017 Fifth Berkeley Symposium on Energy Efficient Electronic Systems Steep Transistors Workshop (E3S)* (Oct 2017), pp. 1–3.

[17] Bojnordi, M. N., and Ipek, E. The memristive Boltzmann machines. *IEEE Micro 37*, 3 (2017), 22–29.

[18] Bolz, J., Farmer, I., Grinspun, E., and Schröoder, P. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. *ACM Trans. Graph. 22*, 3 (July 2003), 917–924.

[19] Bournez, O., and Campagnolo, M. L. *A Survey on Continuous Time Computations.* Springer New York, New York, NY, 2008, pp. 383–423.

[20] Boyd, S., and Vandenberghe, L. *Convex Optimization.* Cambridge university press, 2004.

[21] Brenan, K., Campbell, S., and Petzold, L. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* Society for Industrial and Applied Mathematics, 1995.

[22] Briggs, W., Henson, V., and McCormick, S. *A Multigrid Tutorial*, second ed. Society for Industrial and Applied Mathematics, 2000.

[23] Brockett, R. W. Dynamical systems that sort lists, diagonalize matrices and solve linear programming problems. In *Proceedings of the 27th IEEE Conference on Decision and Control* (Dec 1988), pp. 799–803 vol.1.

[24] Buttari, A., Dongarra, J., Langou, J., Langou, J., Luszczek, P., and Kurzak, J. Mixed precision iterative refinement techniques for the solution of dense linear systems. *Int. J. High Perform. Comput. Appl. 21*, 4 (Nov. 2007), 457–466.

[25] Care, C. *From analogy-making to modelling: the history of analog computing as a modelling technology.* PhD thesis, University of Warwick, 2008.

[26] Carloni, L. P., McMillan, K. L., and Sangiovanni-Vincentelli, A. L. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20*, 9 (Sep 2001), 1059–1076.

[27] Celmer, J., Rouland, M., Aeronautics, U. S. N., Administration, S., and Center, G. S. F. *Automatic Analog Computer Scaling Using Digital Optimization Techniques.* NASA technical note. 1970.

[28] Chen, T., Chen, Y., Duranton, M., Guo, Q., Hashmi, A., Lipasti, M., Nere, A., Qiu, S., Sebag, M., and Temam, O. BenchNN: On the broad potential application scope of hardware neural network accelerators. In *Proceedings of the 2012 IEEE International Symposium on Workload Characterization (IISWC)* (Washington, DC, USA, 2012), IISWC '12, IEEE Computer Society, pp. 36–45.

[29] Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., and Temam, O. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 269–284.

[30] Chen, W., and McNamee, L. P. Iterative solution of large-scale systems by hybrid techniques. *IEEE Transactions on Computers C-19*, 10 (Oct 1970), 879–889.

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

[32] Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., and Xie, Y. PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (June 2016), pp. 27–39.

[33] Chu, M. T. On the continuous realization of iterative processes. *SIAM Review 30*, 3 (1988), 375–387.

[34] Chu, M. T. A list of matrix flows with applications. In *in Hamiltonian and Gradients Flows, Algorithms and Control* (1994), pp. 87–97.

[35] Chua, L., and Lin, G.-N. Nonlinear programming without computation. *IEEE Transactions on Circuits and Systems 31*, 2 (1984), 182–188.

[36] Cipra, B. A. The best of the 20th century: Editors name top 10 algorithms. *SIAM news 33*, 4, 1–2.

[37] Cockshott, P., Koltes, A., O'Donnell, J., Prosser, P., and Vanderbauwhede, W. A hardware relaxation paradigm for solving NP-hard problems. In *Proceedings of the 2008 International Conference on Visions of Computer Science: BCS International Academic Conference* (Swindon, UK, 2008), VoCS'08, BCS Learning & Development Ltd., pp. 75–86.

[38] Conte, T. M., DeBenedictis, E. P., Gargini, P. A., and Track, E. Rebooting computing: The road ahead. *Computer 50*, 1 (Jan 2017), 20–29.

[39] Council, N. R. *Fueling Innovation and Discovery: The Mathematical Sciences in the 21st Century.* The National Academies Press, Washington, DC, 2012.

[40] Cowan, G., Melville, R., and Tsividis, Y. 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.

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

[42] Crank, J., and Nicolson, P. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical Proceedings of the Cambridge Philosophical Society* (1947), vol. 43, Cambridge University Press, pp. 50–67.

[43] DeBenedictis, E. P. Computational complexity and new computing approaches. *Computer 49*, 12 (Dec 2016), 76–79.

[44] DeBenedictis, E. P. It's time to redefine Moore's law again. *Computer 50*, 2 (Feb 2017), 72–75.

[45] Deift, P., Nanda, T., and Tomei, C. Ordinary differential equations and the symmetric eigenvalue problem. *SIAM Journal on Numerical Analysis 20*, 1 (1983), 1–22.

[46] Dennard, R., Rideout, V., Bassous, E., and LeBlanc, A. Design of ion-implanted MOSFET's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of 9*, 5 (Oct 1974), 256–268.

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

[48] Dongarra, J., Heroux, M. A., and Luszczek, P. High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems. *The International Journal of High Performance Computing Applications 30*, 1 (2016), 3–10.

[49] Douglas, C. C., Mandel, J., and Miranker, W. L. Fast hybrid solution of algebraic systems. *SIAM Journal on Scientific and Statistical Computing 11*, 6 (1990), 1073–1086.

[50] Du, Z., Fasthuber, R., Chen, T., Ienne, P., Li, L., Luo, T., Feng, X., Chen, Y., and Temam, O. ShiDianNao: Shifting vision processing closer to the sensor. In *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on* (June 2015), pp. 92–104.

[51] Edwards, D. Numerical and analytic methods in option pricing.

[52] Elshoff, J. L., and Hulina, P. T. The binary floating point digital differential analyzer. In *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference* (New York, NY, USA, 1970), AFIPS '70 (Fall), ACM, pp. 369–376.

[53] Ercsey-Ravasz, M., and Toroczkai, Z. Optimization hardness as transient chaos in an analog approach to constraint satisfaction. *Nature Physics 7*, 12 (2011), 966.

[54] Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., and Burger, D. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2011), ISCA '11, ACM, pp. 365–376.

[55] Esser, S., Andreopoulos, A., Appuswamy, R., Datta, P., Barch, D., Amir, A., Arthur, J., Cassidy, A., Flickner, M., Merolla, P., Chandra, S., Basilico, N., Carpin, S., Zimmerman, T., Zee, F., Alvarez-Icaza, R., Kusnitz, J., Wong, T., Risk, W., McQuinn, E., Nayak, T., Singh, R., and Modha, D. 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.

[56] Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., and LeCun, Y. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on* (June 2011), pp. 109–116.

[57] Fifer, S. *Analogue Computation: Theory, Techniques, and Applications*. No. v. 3 in Analogue Computation: Theory, Techniques, and Applications. McGraw-Hill, 1961.

[58] Firth, A. W. O. Optimization problems: Solution by an analogue computer. *The Computer Journal 4*, 1 (1961), 68–72.

[59] Fletcher, C. *Computational Techniques for Fluid Dynamics 1*. Computational Techniques for Fluid Dynamics. Springer Berlin Heidelberg, 1991.

[60] Forbes, G. The simulation of partial differential equations on the digital differential analyzer. In *Proceedings of the ACM Annual Conference - Volume 2* (New York, NY, USA, 1972), ACM '72, ACM, pp. 860–866.

[61] Gander, M. J. 50 years of time parallel time integration. In *Multiple Shooting and Time Domain Decomposition Methods*. Springer, 2015, pp. 69–113.

[62] Gautschi, W. *Numerical Analysis*. SpringerLink : Bücher. Birkhäuser Boston, 2011.

[63] George, S., Kim, S., Shah, S., Hasler, J., Collins, M., Adil, F., Wunderlich, R., Nease, S., and Ramakrishnan, S. A programmable and configurable mixed-mode FPAA SoC. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems PP*, 99 (2016), 1–9.

[64] Greengard, L., and Rokhlin, V. A fast algorithm for particle simulations. *Journal of Computational Physics 73*, 2 (1987), 325–348.

[65] Guo, N., Huang, Y., and Lei, K. Columbia hybrid computer user's guide. Tech. rep., Columbia University, 2017.

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

[67] Guo, N., Huang, Y., Mai, T., Patil, S., Cao, C., Seok, M., Sethumadhavan, S., and Tsividis, Y. Energy-efficient hybrid analog/digital approximate computation in continuous time. *IEEE Journal of Solid-State Circuits 51*, 7 (July 2016), 1514–1524.

[68] Guseinov, G. S. A class of complex solutions to the finite Toda lattice. *Math. Comput. Model. 57*, 5 (Mar. 2013), 1190–1202.

[69] Hall, C. R., and Kahne, S. J. Automated scaling for hybrid computers. *IEEE Transactions on Computers C-18*, 5 (May 1969), 416–423.

[70] Hameed, R., Qadeer, W., Wachs, M., Azizi, O., Solomatnikov, A., Lee, B. C., Richardson, S., Kozyrakis, C., and Horowitz, M. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2010), ISCA '10, ACM, pp. 37–47.

[71] Hannington, G., and Whitehead, D. G. A floating-point multiplexed DDA system. *IEEE Transactions on Computers C-25*, 11 (Nov 1976), 1074–1077.

[72] Hasler, J. Opportunities in physical computing driven by analog realization. In *2016 IEEE International Conference on Rebooting Computing (ICRC)* (Oct 2016), pp. 1–8.

[73] Hasler, J. Starting framework for analog numerical analysis for energy-efficient computing. *Journal of Low Power Electronics and Applications 7*, 3 (2017).

[74] Helmke, U., Brockett, R., and Moore, J. *Optimization and Dynamical Systems.* Communications and Control Engineering. Springer London, 2012.

[75] Hennessy, J. L., and Patterson, D. A. *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[76] Hetzler, S. M. A continuous version of Newton's method. *The College Mathematics Journal 28*, 5 (1997), 348–351.

[77] Higham, D. J. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Review 43*, 3 (2001), 525–546.

[78] Hu, M., and Strachan, J. P. Accelerating discrete fourier transforms with dot-product engine. In *2016 IEEE International Conference on Rebooting Computing (ICRC)* (Oct 2016), pp. 1–5.

[79] Hu, M., Strachan, J. P., Li, Z., Grafals, E. M., Davila, N., Graves, C., Lam, S., Ge, N., Yang, J. J., and Williams, R. S. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE* (2016), IEEE, pp. 1–6.

[80] Hu, X., Ma, L. H., Ru, L., and Zhang, S. Analog error correction codes based on chaotic system: The 2-dimensional tent codes. In *2014 International Conference on Wireless Communication and Sensor Network* (Dec 2014), pp. 34–38.

[81] Huang, Y., Guo, N., Seok, M., Tsividis, Y., Mandli, K., and Sethumadhavan, S. Hybrid analog-digital solution of nonlinear partial differential equations. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2017), MICRO-50 '17, ACM, pp. 665–678.

[82] Huang, Y., Guo, N., Seok, M., Tsividis, Y., and Sethumadhavan, S. Evaluation of an analog accelerator for linear algebra. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Piscataway, NJ, USA, 2016), ISCA '16, IEEE Press, pp. 570–582.

[83] Huang, Y., Guo, N., Seok, M., Tsividis, Y., and Sethumadhavan, S. Analog computing in a modern context: A linear algebra accelerator case study. *IEEE Micro 37*, 3 (2017), 30–38.

[84] Huang, Y., and Sethumadhavan, S. Hybrid continuous-discrete computer: from ISA to microarchitecture. Tech. Rep. CUCS-029-14, Computer Science, Columbia University, 2013.

[85] Jackson, A. *Analog Computation.* McGraw-Hill, 1960.

[86] Jacobsen, J., Lewis, O., and Tennis, B. Approximations of continuous Newton's method: an extension of Cayley's problem. *Electronic Journal of Differential Equations (EJDE) [electronic only] 2007* (2007), 163–173.

[87] Jalon, J. G. d., and Bayo, E. *Kinematic and Dynamic Simulation of Multibody Systems: The Real Time Challenge.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1994.

[88] Joneckis, L., Koester, D., and Alspector, J. An initial look at alternative computing technologies for the intelligence community. Tech. rep., INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA, Jan 2014.

[89] Jun, B., and Kocher, P. The Intel random number generator.

[90] Karmarkar, N. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing* (1984), ACM, pp. 302–311.

[91] Karplus, W. *Analog Simulation: Solution of Field Problems.* McGraw-Hill series in information processing and computers. McGraw-Hill, 1958.

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

[93] Karplus, W. J. A hybrid computer technique for treating nonlinear partial differential equations. *IEEE Transactions on Electronic Computers*, 5 (1964), 597–605.

[94] Karplus, W. J., and Russell, R. Increasing digital computer efficiency with the aid of error-correcting analog subroutines. *Computers, IEEE Transactions on C-20*, 8 (Aug 1971), 831–837.

[95] Keckler, S., Dally, W., Khailany, B., Garland, M., and Glasco, D. GPUs and the future of parallel computing. *Micro, IEEE 31*, 5 (Sept 2011), 7–17.

[96] Kelley, C. *Iterative Methods for Linear and Nonlinear Equations.* Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, 1995.

[97] Khazraee, M., Gutierrez, L. V., Magaki, I., and Taylor, M. B. Specializing a planet's computation: ASIC clouds. *IEEE Micro 37*, 3 (2017), 62–69.

[98] Kinget, P., and Steyaert, M. S. J. A programmable analog cellular neural network CMOS chip for high speed image processing. *IEEE Journal of Solid-State Circuits 30*, 3 (Mar 1995), 235–243.

[99] Kirk, D. *Optimal Control Theory: An Introduction.* Dover Books on Electrical Engineering. Dover Publications, 2012.

[100] Knoll, D. A., and Keyes, D. E. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *Journal of Computational Physics 193*, 2 (2004), 357–397.

[101] Koosh, V. F. *Analog computation and learning in VLSI.* PhD thesis, California Institute of Technology, 2001.

[102] Korn, G., and Korn, T. *Electronic Analog and Hybrid Computers.* McGraw-Hill, 1972.

[103] Korn, G. A. The impact of hybrid analog-digital techniques on the analog-computer art. *Proceedings of the IRE 50*, 5 (1962), 1077–1086.

[104] Kozek, T., and Roska, T. A double time-scale CNN for solving two-dimensional Navier-Stokes equations. *International Journal of Circuit Theory and Applications 24*, 1 (1996), 49–55.

[105] Krüger, J., and Westermann, R. Linear algebra operators for GPU implementation of numerical algorithms. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 908–916.

[106] Kulkarni, S., Bhat, S., Khasanvis, S., and Moritz, C. A. Magneto-electric approximate computational circuits for bayesian inference. In *2017 IEEE International Conference on Rebooting Computing (ICRC)* (Nov 2017), pp. 1–8.

[107] Kulkarni, S., Bhat, S., and Moritz, C. A. Structure discovery for gene expression networks with emerging stochastic hardware. In *2017 IEEE International Conference on Rebooting Computing (ICRC)* (Nov 2017), pp. 1–4.

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

[109] Li, M., Chiasson, J., Bodson, M., and Tolbert, L. M. A differential-algebraic approach to speed estimation in an induction motor. *IEEE Transactions on Automatic Control 51*, 7 (July 2006), 1172–1177.

[110] LiKamWa, R., Hou, Y., Gao, Y., Polansky, M., and Zhong, L. RedEye: Analog ConvNet image sensor architecture for continuous mobile vision. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (June 2016), pp. 255–266.

[111] Liu, D., Chen, T., Liu, S., Zhou, J., Zhou, S., Teman, O., Feng, X., Zhou, X., and Chen, Y. Pudiannao: A polyvalent machine learning accelerator. *SIGPLAN Not. 50*, 4 (Mar. 2015), 369–381.

[112] Liu, Y., Li, J., and Xie, K. Efficient image transmission through analog error correction. In *Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on* (2011), IEEE, pp. 1–6.

[113] Lottarini, A., Edwards, S. A., Ross, K. A., and Kim, M. A. Network synthesis for database processing units. In *Proceedings of the 54th Annual Design Automation Conference 2017* (New York, NY, USA, 2017), DAC '17, ACM, pp. 90:1–90:6.

[114] MacDonald, D. *Noise and Fluctuations: An Introduction.* Dover books on physics. Dover Publications, 2006.

[115] Madhavan, A. *Abusing Hardware Race Conditions for High Throughput Energy Efficient Computation.* PhD thesis, 2016. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2017-02-27.

[116] Madhavan, A., Sherwood, T., and Strukov, D. Race logic: A hardware acceleration for dynamic programming algorithms. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Piscataway, NJ, USA, 2014), ISCA '14, IEEE Press, pp. 517–528.

[117] Madhavan, A., Sherwood, T., and Strukov, D. Race logic: Abusing hardware race conditions to perform useful computation. *IEEE Micro 35*, 3 (May 2015), 48–57.

[118] Madhavan, A., Sherwood, T., and Strukov, D. Energy efficient computation with asynchronous races. In *Proceedings of the 53rd Annual Design Automation Conference* (New York, NY, USA, 2016), DAC '16, ACM, pp. 108:1–108:6.

[119] Madhavan, A., Sherwood, T., and Strukov, D. A 4-mm2 180-nm-CMOS 15-giga-cell-updates-per-second DNA sequence alignment engine based on asynchronous race conditions. In *2017 IEEE Custom Integrated Circuits Conference (CICC)* (April 2017), pp. 1–4.

[120] Mandel, J., and Miranker, W. New techniques for fast hybrid solutions of systems of equations. *International Journal for Numerical Methods in Engineering 27*, 3 (1989), 455–467.

[121] Manohar, R. Comparing stochastic and deterministic computing. *Computer Architecture Letters PP*, 99 (2015), 1–1.

[122] McDonough, J. Lectures on computational numerical analysis of partial differential equations.

[123] McGhee, R. B., and Nilsen, R. N. The extended resolution digital differential analyzer: A new computing structure for solving differential equations. *IEEE Transactions on Computers C-19*, 1 (Jan 1970), 1–9.

[124] Mead, C. Neuromorphic electronic systems. *Proceedings of the IEEE 78*, 10 (1990), 1629–1636.

[125] Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., and Modha, D. S. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *2011 IEEE Custom Integrated Circuits Conference (CICC)* (Sept 2011), pp. 1–4.

[126] Milano, F. Continuous Newton's method for power flow analysis. *IEEE Transactions on Power Systems 24*, 1 (Feb 2009), 50–57.

[127] Mills, J. W. The nature of the Extended Analog Computer. *Physica D Nonlinear Phenomena 237* (July 2008), 1235–1256.

[128] Mills, J. W., Himebaugh, B., Allred, A., Bulwinkle, D., Deckard, N., Gopalakrishnan, N., Miller, J., Miller, T., Nagai, K., Nakamura, J., et al. Extended analog computers: A unifying paradigm for VLSI, plastic and colloidal computing systems.

[129] Mills, J. W., Parker, M., Himebaugh, B., Shue, C., Kopecky, B., and Weilemann, C. "empty space" computes: The evolution of an unconventional supercomputer. In *Proceedings of the 3rd Conference on Computing Frontiers* (New York, NY, USA, 2006), CF '06, ACM, pp. 115–126.

[130] Molnár, B., and Ercsey-Ravasz, M. Analog dynamics for solving max-SAT problems. In *2014 14th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)* (July 2014), pp. 1–2.

[131] Morgan, A. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems.* Prentice-Hall, 1987.

[132] Neuberger, J. W. The continuous Newton's method, inverse functions, and Nash-Moser. *The American Mathematical Monthly 114*, 5 (2007), 432–437.

[133] Nocedal, J., and Wright, S. *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer New York, 2000.

[134] Ortega, J., and Rheinboldt, W. *Iterative Solution of Nonlinear Equations in Several Variables.* Society for Industrial and Applied Mathematics, 2000.

[135] Petzold, L. Differential/algebraic equations are not ODEs. *SIAM Journal on Scientific and Statistical Computing 3*, 3 (1982), 367–384.

[136] Philokyprou, G., and Halatsis, C. Floating-point and multibit-increment digital-differential-analyser structures. *Electronics Letters 8* (October 1972), 531–532(1).

[137] Present, I. Cramming more components onto integrated circuits. *Readings in Computer Architecture 56* (2000).

[138] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3 ed. Cambridge University Press, New York, NY, USA, 2007.

[139] Pyle, S. D., Thangavel, V., Williams, S. M., and DeMara, R. F. Self-scaling evolution of analog computation circuits with digital accuracy refinement. In *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)* (June 2015), pp. 1–8.

[140] Qadeer, W., Hameed, R., Shacham, O., Venkatesan, P., Kozyrakis, C., and Horowitz, M. A. Convolution engine: Balancing efficiency & flexibility in specialized computing. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2013), ISCA '13, ACM, pp. 24–35.

[141] Rao, A. V. (preprint) aas 09-334 a survey of numerical methods for optimal control, 2009.

[142] Rojo, A. G., and Bloch, A. M. Nonholonomic double-bracket equations and the Gauss thermostat. *Phys. Rev. E 80* (Aug 2009), 025601.

[143] Rothganger, F., James, C. D., and Aimone, J. B. Computing with dynamical systems. In *2016 IEEE International Conference on Rebooting Computing (ICRC)* (Oct 2016), pp. 1–3.

[144] Rubin, A. I., and Mawson, J. B. Hybrid computation 1976 and its future. *Computer 9*, 7 (1976), 37–46.

[145] Rutishauser, U., and Douglas, R. J. State-dependent computation using coupled recurrent networks. *Neural Comput. 21*, 2 (Feb. 2009), 478–509.

[146] Sanz-Serna, J. M. *Markov Chain Monte Carlo and Numerical Differential Equations.* Springer International Publishing, Cham, 2014, pp. 39–88.

[147] Sarpeshkar, R. Analog versus digital: extrapolating from electronics to neurobiology. *Neural Computation 10*, 7 (1998), 1601–1638.

[148] Sarpeshkar, R., Delbruck, T., and Mead, C. A. White noise in MOS transistors and resistors. *IEEE Circuits and Devices Magazine 9*, 6 (1993), 23–29.

[149] Sarpeshkar, R., Lyon, R. F., and Mead, C. A low-power wide-dynamic-range analog VLSI cochlea. *Analog Integrated Circuits and Signal Processing 16*, 3 (Aug 1998), 245–274.

[150] Sarpeshkar, R., and O'Halloran, M. Scalable hybrid computation with spikes. *Neural Computation 14*, 9 (2002), 2003–2038.

[151] Sauer, T. *Numerical Solution of Stochastic Differential Equations in Finance.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 529–550.

[152] Saupe, D. *Discrete Versus Continuous Newton's Method: A Case Study.* Springer Netherlands, Dordrecht, 1989, pp. 59–80.

[153] Schell, B., and Tsividis, Y. 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.

[154] Schemmel, J., Fieres, J., and Meier, K. Wafer-scale integration of analog neural networks. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (June 2008), pp. 431–438.

[155] Schlottmann, C. R., Shapero, S., Nease, S., and Hasler, P. A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing. *IEEE Journal of Solid-State Circuits 47*, 9 (2012), 2174–2184.

[156] Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., and Plank, J. S. A survey of neuromorphic computing and neural networks in hardware. *CoRR abs/1705.06963* (2017).

[157] Seo, J.-s., Brezzo, B., Liu, Y., Parker, B. D., Esser, S. K., Montoye, R. K., Rajendran, B., Tierno, J. A., Chang, L., Modha, D. S., et al. A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE* (2011), IEEE, pp. 1–4.

[158] Sethumadhavan, S., Roberts, R., and Tsividis, Y. A case for hybrid discrete-continuous architectures. *IEEE Comput. Archit. Lett. 11*, 1 (Jan. 2012), 1–4.

[159] Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J. P., Hu, M., Williams, R. S., and Srikumar, V. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (June 2016), pp. 14–26.

[160] Shaw, C. *Using Computational Fluid Dynamics.* Prentice Hall, 1992.

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

[162] Srinivasan, M., and Bernard, G. A proposed mechanism for multiplication of neural signals. *Biological Cybernetics 21*, 4 (1976), 227–236.

[163] St. Amant, R., Yazdanbakhsh, A., Park, J., Thwaites, B., Esmaeilzadeh, H., Hassibi, A., Ceze, L., and Burger, D. General-purpose code acceleration with limited-precision analog computation. *SIGARCH Comput. Archit. News 42*, 3 (June 2014), 505–516.

[164] Tadmor, E. A review of numerical methods for nonlinear partial differential equations. *Bulletin of the American Mathematical Society 49*, 4 (2012), 507–554.

[165] Taylor, M. B. Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. In *Design Automation Conference* (2012).

[166] Thangavel, V., Song, Z. X., and Demara, R. F. Intrinsic evolution of truncated Puiseux series on a mixed-signal field-programmable soc. *IEEE Access 4* (2016), 2863–2872.

[167] Toselli, A., and Widlund, O. *Domain Decomposition Methods - Algorithms and Theory.* Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2004.

[168] Traub, J. F. A continuous model of computation. *arXiv Preprint Physics/0106045* (2001).

[169] Trefethen, L. N., and Bau III, D. *Numerical Linear Algebra*, vol. 50. SIAM, 1997.

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

[171] Venkatesh, G., Sampson, J., Goulding, N., Garcia, S., Bryksin, V., Lugo-Martinez, J., Swanson, S., and Taylor, M. B. Conservation cores: reducing the energy of mature computations. In *ASPLOS 2010: Architectural Support for Programming Languages and Operating Systems* (2010).

[172] Vichik, S., and Borrelli, F. Solving linear and quadratic programs with an analog circuit. *Computers & Chemical Engineering 70* (2014), 160–171.

[173] Vichnevetsky, R. Analog/hybrid solution of partial differential equations in the nuclear industry. *SIMULATION 11*, 6 (1968), 269–281.

[174] Vom Scheidt, J. Kloeden, p. e.; Platen, e., numerical solution of stochastic differential equations. Berlin etc., Springer-Verlag 1992. XXXVI, 632 pp., 85 figs., dm 118,oo. isbn 3-540-54062-8 (applications of mathematics 23). *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik 74*, 8 (1994), 332–332.

[175] Walsh, J. A. The dynamics of Newton's method for cubic polynomials. *The College Mathematics Journal 26*, 1 (1995), 22–28.

[176] Wang, S., Lebeck, A. R., and Dwyer, C. Nanoscale resonance energy transfer-based devices for probabilistic computing. *IEEE Micro 35*, 5 (Sept 2015), 72–84.

[177] Wang, S., Zhang, X., Li, Y., Bashizade, R., Yang, S., Dwyer, C., and Lebeck, A. R. Accelerating Markov random field inference using molecular optical Gibbs sampling units. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (June 2016), pp. 558–569.

[178] Wang, Y., Baboulin, M., Rupp, K., Le Maître, O., and Fraigneau, Y. Solving 3D incompressible Navier-Stokes equations on hybrid CPU/GPU systems. In *Proceedings of the High Performance Computing Symposium* (San Diego, CA, USA, 2014), HPC '14, Society for Computer Simulation International, pp. 12:1–12:8.

[179] Whitham, G. B. *Linear and Nonlinear Waves*, vol. 42. John Wiley & Sons, 2011.

[180] Wilkins, B. *Analogue and Iterative Methods in Computation, Simulation, and Control.* Modern electrical studies. Chapman and Hall, 1970.

[181] Wong, H.-S. P., Lee, H.-Y., Yu, S., Chen, Y.-S., Wu, Y., Chen, P.-S., Lee, B., Chen, F. T., and Tsai, M.-J. Metal–oxide RRAM. *Proceedings of the IEEE 100*, 6 (2012), 1951–1970.

[182] Wu, L., Lottarini, A., Paine, T. K., Kim, M. A., and Ross, K. A. Q100: The architecture and design of a database processing unit. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 255–268.

[183] Zhang, Y. 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.

[184] Zhang, Y., and Ge, S. S. Design and analysis of a general recurrent neural network model for time-varying matrix inversion. *Neural Networks, IEEE Transactions on 16*, 6 (Nov 2005), 1477–1490.