Analog Computing in a Modern Context: A Linear Algebra Accelerator Case Study

This article presents a programmable analog accelerator for solving systems of linear equations. The authors compensate for commonly perceived downsides of analog computing. They compare the analog solver's performance and energy consumption against an efficient digital algorithm running on a general-purpose processor. Finally, they conclude that problem classes outside of systems of linear equations could hold more promise for analog acceleration.

Yipeng Huang
Ning Guo
Mingoo Seok
Yannis Tsividis
Simha Sethumadhavan
Columbia University

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. An analog accelerator discussed in this article breaks two fundamental assumptions in modern computing: in contrast to using digital binary numbers, an analog accelerator encodes numbers using the full range of circuit voltage and current. Furthermore, in contrast to operating step by step on clocked hardware, an analog accelerator updates its values continuously. These different hardware assumptions can provide substantial gains but would need different abstractions and cross-layer optimizations to support various modern workloads. We draw inspiration from an immense amount of prior work in analog electronic computing (see the sidebar, "Related Work in Analog Computing").

To support modern workloads in the digital era, we observed that modern scientific computing and big data problems are converted to linear algebra problems. To maximize analog acceleration's usefulness, we explored whether analog accelerators are effective at solving systems of linear equations, the single most important numerical primitive in continuous mathematics.

For readers not familiar with linear algebra, systems of linear equations are often solved using iterative numerical linear algebra methods, which start with an initial guess for the entire solution vector and update the solution vector over iterations of the algorithm, each step further minimizing the difference between the guess and the correct solution.[1]

0272-1732/17/$33.00 © 2017 IEEE

Related Work in Analog Computing

Analog computers of the mid-20th century were widely used to solve scientific computing problems, described as ordinary differential equations (ODEs). Analog computers would solve those ODEs by setting up analog electronic circuits, whose time-dependent voltage and current were described by corresponding ODEs. The analog computers therefore were computational analogies of physical models.

Our group revisited this model of analog computing for solving nonlinear ODEs, which frequently appear in cyber-physical systems workloads, with higher performance and efficiency compared to digital systems.[1,2] The analog, continuous-time output of analog computing is especially suited for embedded systems applications in which sensor inputs are analog and actuators can use such results directly. The question for this article is whether analog acceleration can help conventional workloads in which inputs and outputs are digital.

Modern scientific computation and big data workloads are phrased as linear algebra problems. In this article, our analog accelerator solves an ODE that does steepest descent, in turn solving a linear algebra problem. Such a solving method belongs to a broad class of ODEs that can solve other numerical problems, including nonlinear systems of equations.[3,4] These ODEs point to other ways analog accelerators can support modern workloads.

We draw a distinction between our approach to analog acceleration and that of using analog circuits to build neural networks.[5,6] Most importantly, we do not use training to get a network topology and weights that solve a given problem. No prior knowledge of the solution or training set of solutions is required. The analog acceleration technique presented in this article is a procedural approach to solving problems: there is a predefined way to convert a system of linear equations under study into an analog accelerator configuration.

References

1. G. Cowan, R. Melville, and Y. Tsividis, "A VLSI Analog Computer/Digital Computer Accelerator," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, 2006, pp. 42–53.
2. N. Guo et al., "Energy-Efficient Hybrid Analog/Digital Approximate Computation in Continuous Time," *IEEE J. Solid-State Circuits*, vol. 51, no. 7, 2016, pp. 1514–1524.
3. M.T. Chu, "On the Continuous Realization of Iterative Processes," *SIAM Rev.*, vol. 30, no. 3, 1988, pp. 375–387.
4. O. Bournez and M.L. Campagnolo, *A Survey on Continuous Time Computations*, Springer, 2008, pp. 383–423.
5. R. LiKamWa et al., "RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision," *SIGARCH Computer Architecture News*, vol. 44, no. 3, 2016, pp. 255–266.
6. A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *SIGARCH Computer Architecture News*, vol. 44, no. 3, 2016, pp. 14–26.

Efficient iterative methods such as the conjugate gradient method are increasingly important because intermediate guess vectors are a good approximation of the correct solution.

In discrete-time-iterative linear algebra algorithms, the solution vector changes in steps, and each step is characterized by a step size. The step size affects the algorithm's efficiency and requires many 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 operations in each conjugate gradient step.

In an analog accelerator, systems of linear equations can also be thought of as solved via an iterative algorithm, with an important distinction that the 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 that the change in a set of variables is a function of the variables' present value. We can naturally solve ODEs using an analog accelerator.

We give an example of an analog accelerator 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 Figure 1). We perform operations on this solution vector by feeding the vector through multiplier and summation units. Digital-to-analog converters (DACs) provide constant coefficients and biases. Using these function units, we create a linear function of the solution vector, which is

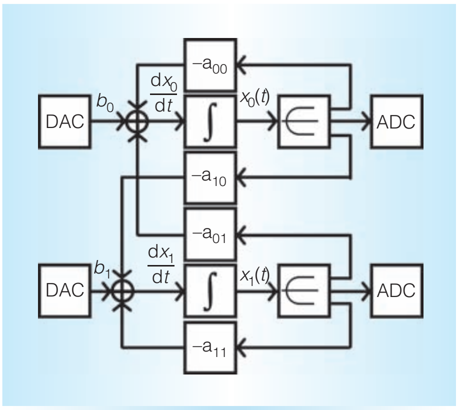

Figure 1. Schematic of an analog accelerator for solving $A\mathbf{x} = \mathbf{b}$, a linear system of two equations with two unknown variables. Matrix A is a known matrix of coefficients realized using multipliers; $\mathbf{x}$ is an unknown vector contained in integrators; $\mathbf{b}$ is a known 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, which is usually true for the problems we discuss.

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 integrators are charged to an initial condition representing the iterative method's initial guess. The accelerator starts computation by releasing the integrator, allowing its output to deviate from its initial value. 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, we sample them using analog-to-digital converters (ADCs).

These techniques were used in early analog computers[2–4] and have recently been explored in small-scale experiments with analog computation.[5,6]

Analog Linear Algebra Advantages

Solving linear algebra problems using ODEs on an analog accelerator has several potential advantages compared to using a discrete-time algorithm on a digital general-purpose or special-purpose system.

Explicit Data-Graph Execution Architecture

The analog accelerator uses an explicit dataflow graph in which the sequence of operations on data is realized by connecting functional units end to end. During computation, analog signals representing intermediate results flow from one unit to the next, so there are no overheads in fetching and decoding instructions, and there are no accesses to digital memory. The former is a benefit of digital accelerators, too, but the latter is a unique benefit of the analog computational model.

Continuous Time Speed

The analog accelerator hardware and algorithm both operate in continuous time. The values contained in the integrators are continuously being updated, and the update rate is not limited by a finite clock frequency, which is the limiting factor in discrete-time hardware. Furthermore, a continuous-time ODE solution has no concern about the correct step size to take to update the solution vector, in contrast to discrete-time iterative algorithms, in which computing the correct step size represents most operations needed per algorithm iteration. In these regards, the analog accelerator is potentially faster than discrete-time architectures. Finally, no power-hungry clock signal is needed to synchronize operations.

Continuous Value Efficiency

The analog accelerator solves the system of linear equations using real numbers encoded in voltage and current, so each wire can represent the full range of values in the analog accelerator. In contrast, changing the value of a digital binary number affects many bits: sweeping an 8-bit unsigned integer from 0 to 255 needs 502 binary inversions, whereas a more economical Gray encoding still needs 255 inversions. Furthermore, multiplication, addition, and integration are all comparatively straightforward on analog variables compared to digital ones. This contrasts with floating-point arithmetic, in which the logarithmically encoded exponent portion of digital floating-point variables makes it complicated to add and subtract variables. In these regards, analog encoding is potentially more efficient than digital, binary encodings.

Table 1. Analog accelerator instruction set architecture.

Instruction type	Instruction	Parameters	Instruction effect
Control	Initialize	—	Find input and output offset and gain calibration settings for all function units.
Configuration	Set connection	Source, destination	Set a crossbar switch to create an analog current connection between two analog interfaces.
Configuration	Set initial condition	Pointer to an integrator, initial condition value	Charge integrator capacitors to have ODE initial condition value.
Configuration	Set multiplier gain	Pointer to a multiplier, gain value	Amplify values by constant coefficient gain.
Configuration	Set constant offset	Pointer to a DAC, offset value	Add a constant bias to values.
Configuration	Set timeout time	Number of digital controller clock cycles	Stop analog computation after specified time once started.
Configuration	Configuration commit	—	Finish configuration and write any new changes to chip registers.
Control	Execution start	—	Start analog computation by letting integrators deviate from initial conditions.
Control	Execution stop	—	Stop analog computation by holding integrators at their present value.
Data input	Enable analog input	Pointer to chip analog input channel	Open chip analog input channel, allowing multiple chips to participate in computation.
Data output	Read analog value	Pointer to an ADC, memory location to store result	Read analog computation results from ADCs and store values.
Exception	Read exceptions	Memory location to store result	Read the exception vector indicating whether analog units exceeded their valid range.

*ADC: analog-to-digital converter; DAC: digital-to-analog converter; ODE: ordinary differential equation.

Analog Accelerator Architecture

The analog accelerator acts as a peripheral to a digital host processor. The analog accelerator interface accepts an accelerator configuration, which entails the connectivity between function units, multiplier gains, DAC constants, and integrator initial conditions. Additionally, the interface allows calibration, computation control, and reading of output values, and reporting exceptions. Table 1 summarizes the analog accelerator's essential system calls and corresponding instructions.

Analog Accelerator Physical Prototype

We tested analog acceleration for linear algebra using a prototype reconfigurable analog accelerator silicon chip implemented in 65-nm CMOS technology (see Figures 2 and 3). The accelerator comprises four integrators, plus accompanying DACs, multipliers, and ADCs connected using crossbar switches. In our analog accelerator, electrical currents represent variables. Fan-out 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. Eight multipliers allow variable-variable and constant-variable multiplication.

The physical prototype validates the analog circuits' functionality and allows physical measurement of component area and energy. Additionally, the chip allows rapid prototyping of accelerator algorithms.

Using physical timing, power, and area measurements recorded by Ning Guo and colleagues[7] and summarized in Table 2, we built a model that predicts the properties of larger-scale analog accelerators. In Table 2, "analog core power" and "analog core area" show the power and area of each block that forms the analog signal path. The noncore transistors and nets not involved in analog computation include calibration and testing circuits and registers. The core area and power

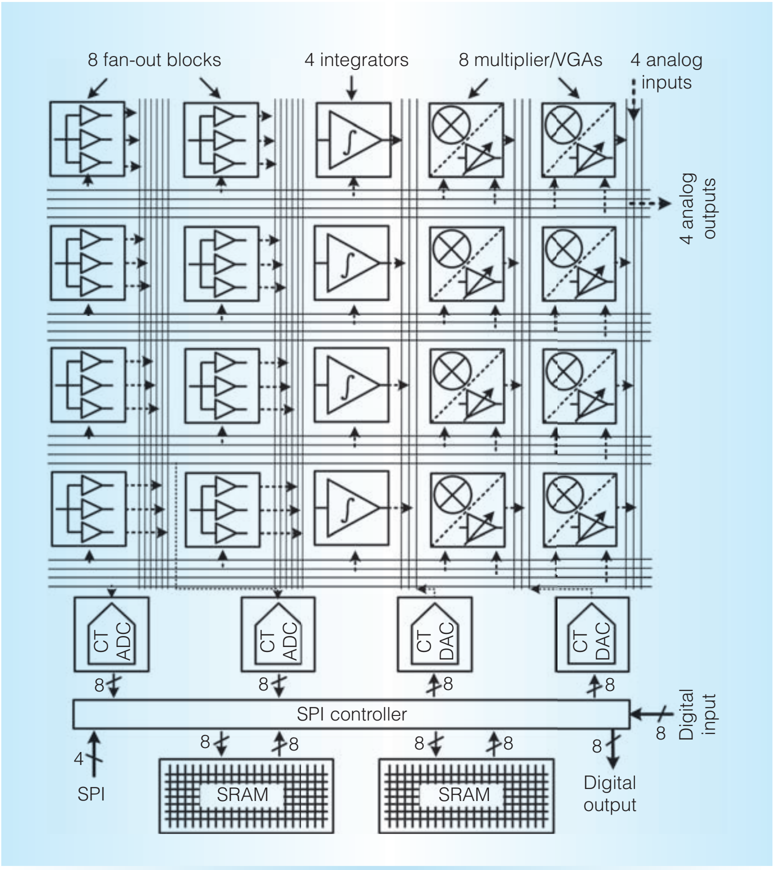

Figure 2. Analog accelerator architecture diagram, showing rows of analog, mixed-signal, and digital components, along with crossbar interconnect.[7] "CT" refers to continuous time. Static RAMs (SRAMs) are used as lookup tables for nonlinear functions (not used for the purposes of this article).

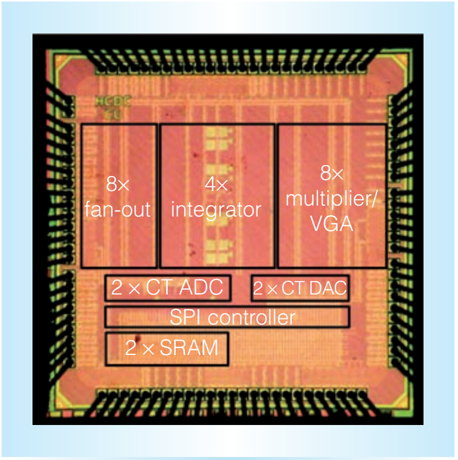

Figure 3. Die photo of an analog accelerator chip fabricated in 65-nm CMOS technology, showing major components.[7] "VGAs" are variable-gain amplifiers. The die area is 3.8 mm^2.

scale up and down for different analog bandwidth designs. We explore how different bandwidth choices influence analog accelerator performance and efficiency.

Mitigation of Analog Linear Algebra Disadvantages

We encountered several drawbacks of analog computing, including limited accuracy, precision, and scalability. We tackled each of these problems in the context of solving linear algebra, although the techniques we discuss apply to other styles of analog computer architecture.

Improve Accuracy Using Calibration and Analog Exceptions

Analog circuits provide limited accuracy compared to binary ones, in which values are unambiguously interpreted as 0 or 1. Analog hardware 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. Gain and offset errors refer to inaccurate results in multiplication and summation, which can be calibrated away using additional DACs that adjust circuit parameters to shift signals and adjust gains. These DACs are controlled by registers, whose contents are set using binary search during calibration by the digital host. The settings vary across different copies of functional units and accelerator chips, but remain constant during accelerator operation.

Nonlinearity errors occur when changes in inputs result in disproportionate changes in outputs, and 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. At the same time, the host observes if the dynamic range is not fully used, which could result in low precision. When either exception type occurs, the original problem is rescaled to fit in the dynamic range of the analog accelerator, and computation is reattempted.

The combination of widespread calibration and exception checking ensures that the

Table 2. Summary of analog accelerator components.

Unit type	Analog core power	Total unit power	Analog core area	Total unit area
Integrator	22 μW	28 μW	0.016 mm^2	0.040 mm^2
Fan-out	30 μW	37 μW	0.005 mm^2	0.015 mm^2
Multiplier	39 μW	49 μW	0.024 mm^2	0.050 mm^2
ADC	27 μW	54 μW	0.049 mm^2	0.054 mm^2
DAC	4.6 μW	4.6 μW	0.013 mm^2	0.022 mm^2

analog solution's accuracy is within the sampling resolution of ADCs.

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. When the analog accelerator outputs are steady, we can sample the solutions once with higher-precision ADCs.

Even then, high-precision ADCs still fall short of the precision in floating-point numbers. Even though the analog variables are themselves highly precise, sampling the variables using ADCs can result in only 8 to 12 bits of precision. We get higher-precision results by running the analog accelerator multiple times. We use the digital host computer to find the residual error in the solution, and we set up the analog accelerator to solve a new problem, focusing on the residual. Each problem has smaller-magnitude variables than previous runs, which lets us scale up the variables to fit the dynamic range of the analog hardware. We can iterate between analog and digital hardware a few times to get a more precise result than using the analog hardware alone.

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 solving 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. Typically, the larger iteration is an iterative method operating on vectors, which do not have as strong convergence properties as iterative methods do 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.

Evaluation

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.

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. However, the prototype's small 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

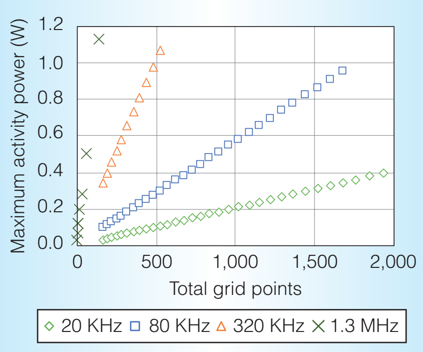

Figure 4. Power versus analog accelerator size for various bandwidth choices. We observe that analog circuits operate faster when the internal node voltages representing variables change more quickly. We hold the capacitance fixed to the capacitance of the prototype's design, and use larger currents that draw more power to charge and discharge the node capacitances in the signal paths carrying variables.

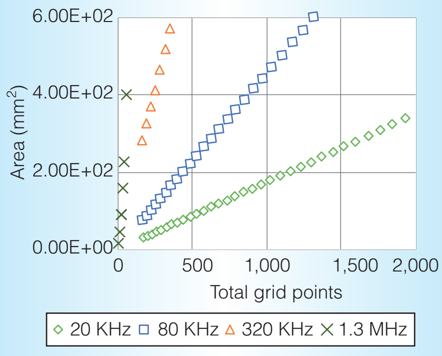

Figure 5. Area versus analog accelerator size for various bandwidth choices. We observe that the transistor aspect ratio *W*/*L* must increase to increase the current, and therefore bandwidth, of the design. *L* is kept at a minimum dictated by the technology node, leaving bandwidth to be linearly dependent on *W*. Thus, we estimate area increasing linearly with bandwidth.

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 4 and 5 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.

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.

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. 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 floating-point numbers.

The conjugate gradient algorithm uses a sustained 20 clock cycles per numerical iteration per row element. The comparison assumes identical transfer cost of data from main memory to the accelerator versus the CPU: the energy needed to transfer data to and from memory is not modeled, due to the relatively small problem sizes, allowing the program data to be entirely cache resident.

As Figure 6 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; using our analog bandwidth model for power, this design corresponds to 33 percent lower energy consumption compared to a digital general-purpose processor.

We recognize that the performance increases and energy savings 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.

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 fixed-step size relaxation or steepest descent. 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.

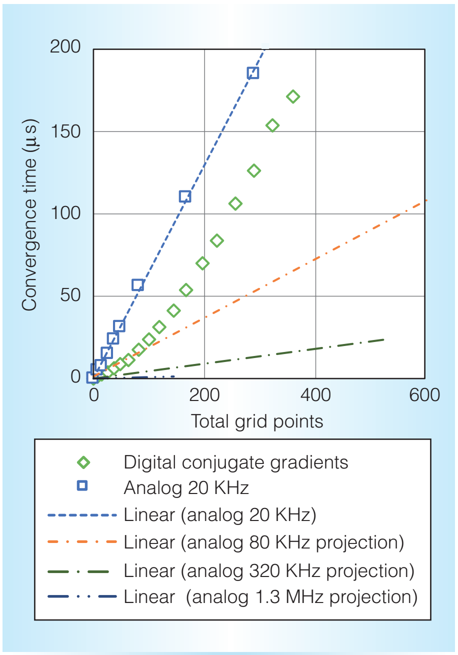

Figure 6. Comparison of time taken to converge to equivalent precision, for high-bandwidth analog accelerators and a digital CPU. The time needed to converge is plotted against the linear algebra problem vector size. We give the projected solution time for 80-KHz, 320-KHz, and 1.3-MHz analog accelerator 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.

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, analog acceleration holds promise in problem classes such as nonlinear systems, in which digital algorithms and hardware architectures have been less successful. In this regard, this article could be the first in a line of work redefining what problems are tractable and should be pursued for analog computing. MICRO

Acknowledgments

This work is supported by NSF award CNS-1239134 and a fellowship from the Alfred P. Sloan Foundation. This article is based on our ISCA 2016 paper.[8]

References

1. W.H. Press et al., *Numerical Recipes: The Art of Scientific Computing*, 3rd ed., Cambridge Univ. Press, 2007.
2. W. Chen and L.P. McNamee, "Iterative Solution of Large-Scale Systems by Hybrid Techniques," *IEEE Trans. Computers*, vol. C-19, no. 10, 1970, pp. 879–889.
3. W.J. Karplus and R. Russell, "Increasing Digital Computer Efficiency with the Aid of Error-Correcting Analog Subroutines," *IEEE Trans. Computers*, vol. C-20, no. 8, 1971, pp. 831–837.
4. G. Korn and T. Korn, *Electronic Analog and Hybrid Computers*, McGraw-Hill, 1972.
5. C.C. Douglas, J. Mandel, and W.L. Miranker, "Fast Hybrid Solution of Algebraic Systems," *SIAM J. Scientific and Statistical Computing*, vol. 11, no. 6, 1990, pp. 1073–1086.
6. Y. Zhang and S.S. Ge, "Design and Analysis of a General Recurrent Neural Network Model for Time-Varying Matrix Inversion," *IEEE Trans. Neural Networks*, vol. 16, no. 6, 2005, pp. 1477–1490.
7. N. Guo et al., "Energy-Efficient Hybrid Analog/Digital Approximate Computation in Continuous Time," *IEEE J. Solid-State Circuits*, vol. 51, no. 7, 2016, pp. 1514–1524.
8. Y. Huang et al., "Evaluation of an Analog Accelerator for Linear Algebra," *Proc. ACM/IEEE 43rd Ann. Int'l Symp. Computer Architecture* (ISCA), 2016, pp. 570–582.

Yipeng Huang is a PhD candidate in the Computer Architecture and Security Technologies Lab at Columbia University. His research interests include applications of analog computing and benchmarking of robotic systems. Huang has an MPhil in computer science from Columbia University. He is a member of the IEEE Computer Society and ACM SIGARCH. Contact him at yipeng@cs.columbia.edu.

Ning Guo is a hardware engineer at Cognescent. His research interests include continuous-time analog/hybrid computing and energy-efficient approximate computing. Guo received a PhD in electrical engineering from Columbia University, where he performed the work for this article. Contact him at ng2364@columbia.edu.

Mingoo Seok is an assistant professor in the Department of Electrical Engineering at Columbia University. His research interests include low-power, adaptive, and cognitive VLSI systems design. Seok received a PhD in electrical engineering from the University of Michigan, Ann Arbor. He has received an NSF CAREER award and is a member of IEEE. Contact him at mgseok@ee.columbia.edu.

Yannis Tsividis is the Edwin Howard Armstrong Professor of Electrical Engineering at Columbia University. His research interests include analog and hybrid analog/digital integrated circuit design for computation and signal processing. Tsividis received a PhD in electrical engineering from the University of California, Berkeley. He is a Life Fellow of IEEE. Contact him at tsividis@ee.columbia.edu.

Simha Sethumadhavan is an associate professor in the Department of Computer Science at Columbia University. His research interests include computer architecture and computer security. Sethumadhavan received a PhD in computer science from the University of Texas at Austin. He has received an Alfred P. Sloan fellowship and an NSF CAREER award. Contact him at simha@cs.columbia.edu.