IN-MEMORY COMPUTING

Programming memristor arrays with arbitrarily high precision for analog computing

Wenhao Song[1,2], Mingyi Rao[2], Yunning Li[3], Can Li[3], Ye Zhuo[1], Fuxi Cai[2], Mingche Wu[2], Wenbo Yin[2], Zongze Li[2], Qiang Wei[2], Sangsoo Lee[2], Hengfang Zhu[2], Lei Gong[2], Mark Barnell[4], Qing Wu[4], Peter A. Beerel[1], Mike Shuo-Wei Chen[1], Ning Ge[2]*, Miao Hu[2]*, Qiangfei Xia[2,3]*, J. Joshua Yang[1,2,3]*

In-memory computing represents an effective method for modeling complex physical systems that are typically challenging for conventional computing architectures but has been hindered by issues such as reading noise and writing variability that restrict scalability, accuracy, and precision in high-performance computations. We propose and demonstrate a circuit architecture and programming protocol that converts the analog computing result to digital at the last step and enables low-precision analog devices to perform high-precision computing. We use a weighted sum of multiple devices to represent one number, in which subsequently programmed devices are used to compensate for preceding programming errors. With a memristor system-on-chip, we experimentally demonstrate high-precision solutions for multiple scientific computing tasks while maintaining a substantial power efficiency advantage over conventional digital approaches.

Many complex physical systems can be described by coupled nonlinear equations that must be analyzed simultaneously at multiple spatiotemporal scales. However, these systems are often too complicated for analytical techniques, and direct numerical computation is hindered by the "curse of dimensionality," which requires exponentially increasing resources as the size of the problem increases. These systems range from nanoscale problems in material modeling to large-scale problems in climate science. Although the need for accurate and high-performance computing solutions is growing, traditional von Neumann computing architectures are reaching their limit in terms of speed, energy consumption, and infrastructure.

A promising alternative is in-memory computing that circumvents the memory-processor bottleneck inherent to von Neumann architectures. In-memory computing in crossbars can execute a large vector-matrix multiplication (VMM) in the analog domain within one computing cycle [O(1) time complexity] by exploiting Ohm's law and Kirchhoff's current law $I_o = G^T V_i$, where V_i is the input voltage vector, G^T is the transposed conductance matrix, and I_o is the output current vector from the crossbar. In the digital domain, such computation requires N^2 multiplication and additions, in which N is the vector size (*1*). To achieve efficient in-memory computing, various emerging devices (*2*), such as floating gate transistors (*3–5*), phase-change (*6–8*), ferroelectric (*9–12*), magnetic (*13*, *14*), organic (*15*), and metal oxide (*16–19*) switching materials, have been studied intensively to enable the parallel computation of matrix operations in nonvolatile memory crossbars. However, technical challenges (*20–25*), such as reading noises and writing variabilities (caused by device-to-device inhomogeneities), have limited the scalability and precision required by many applications, such as high-performance scientific computing and in situ training for neural networks.

Thousands of conductance levels have recently been achieved by eliminating the reading noise issue in individual memristors (*26*). Still, practical numerical problems often require solutions with single ($2^{23} \approx 10^7$ levels, or ~10^{-7} error) or double precision ($2^{52} \approx 10^{15}$ levels, or ~10^{-15} error). Accordingly, analog devices have been primarily used for applications without high-precision requirements, such as machine learning (*8*, *27–31*), randomness-based processing like stochastic computing (*32–34*), and hardware security (*35–37*). To achieve high-precision solutions, innovations in architecture and algorithms, codesigned with analog devices, must be made.

Some theoretical (*38*) and experimental studies (*39*) have used analog arrays to generate a low-precision estimate and then resort to an integrated high-precision digital solver for refinement to produce the required high-precision solutions. Recent efforts in solving high-precision numerical problems used memristors as binary or low-precision cells. They relied on complicated peripheral circuit design (*40*) or intensive software processing (*41*) techniques with frequent quantization operations to obtain error-free results, which substantially reduced energy and area efficiency because of increased costs for analog-to-digital converters (ADCs) and other postprocessing.

In this work, we propose and demonstrate a new circuit architecture and programming protocol that can efficiently represent high-precision numbers using multiple relatively low-precision analog devices, such as memristors, with a greatly reduced overhead in circuitry, energy, and latency compared with existing quantization approaches. As proof-of-principle demonstrations, we have experimentally solved both static and time-evolving partial differential equations (PDEs), including Laplace and Poisson equations, Navier-Stokes (N-S) equations, and magnetohydrodynamics (MHD) problems, as well as adaptive filters like recursive least square (RLS) filters, with memristor crossbar arrays playing various critical roles in the solver. We have achieved high-precision solutions up to 10^{-15} precision on a fully integrated memristor system-on-chip (SoC) while maintaining a substantial power efficiency advantage over conventional digital PDE solvers.

High precision obtained with low-precision devices

Figure 1 shows the simplified schematics of a traditional crossbar architecture with a bit-slicing approach and our proposed true analog architecture and its programming algorithm that can achieve arbitrarily high precisions by overcoming the issues of device writing accuracy and variability. Peripheral input-output registers and digital-to-analog converters (DACs) are omitted in the schematic for simplicity.

Traditional in-memory computing architecture (*31*, *40*, *42*, *43*) follows the same paradigm as digital circuits, which is not error tolerant. In digital circuits design, each number is represented by multiple bits. During multiplication, each bit in the multiplier is multiplied with each bit in the multiplicand to get many partial products. So, a multiplier circuit has a group of bit shifters and adders that add the partial products. Such an approach is inherited and expanded to arrays in traditional in-memory computing architecture, forming the so-called bit-slicing approach. In that approach, the memory devices are bound into very limited (usually binary) predetermined states, with the weight matrix and input vectors sliced into multiple bit planes (only weight planes are shown in Fig. 1, and the input planes are omitted for simplicity), and partial VMMs are performed in those bit planes to obtain many partial products. Those partial products are then quantized and combined using additional digital circuitry such as ADCs, shifters, and adders to get the full VMM product. That is as complex as the digital multiplier, if not more. On the algorithm side, the same computing algorithm optimized for digital computing is typically used. Therefore, the accuracy of the VMM result relies heavily on the programming accuracy of each cell in the array. This approach forces analog devices to behave like digital devices, negating the advantages of analog devices and analog computing.

[1]Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA, USA. [2]TetraMem Inc., Fremont, CA, USA. [3]Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA. [4]Air Force Research Lab, Information Directorate, Rome, NY, USA.
*Corresponding author. Email: glenn.ge@tetramem.com (N.G.); miao.hu@tetramem.com (M.H.); qxia@umass.edu (Q.X.); jjoshuay@usc.edu (J.J.Y.)

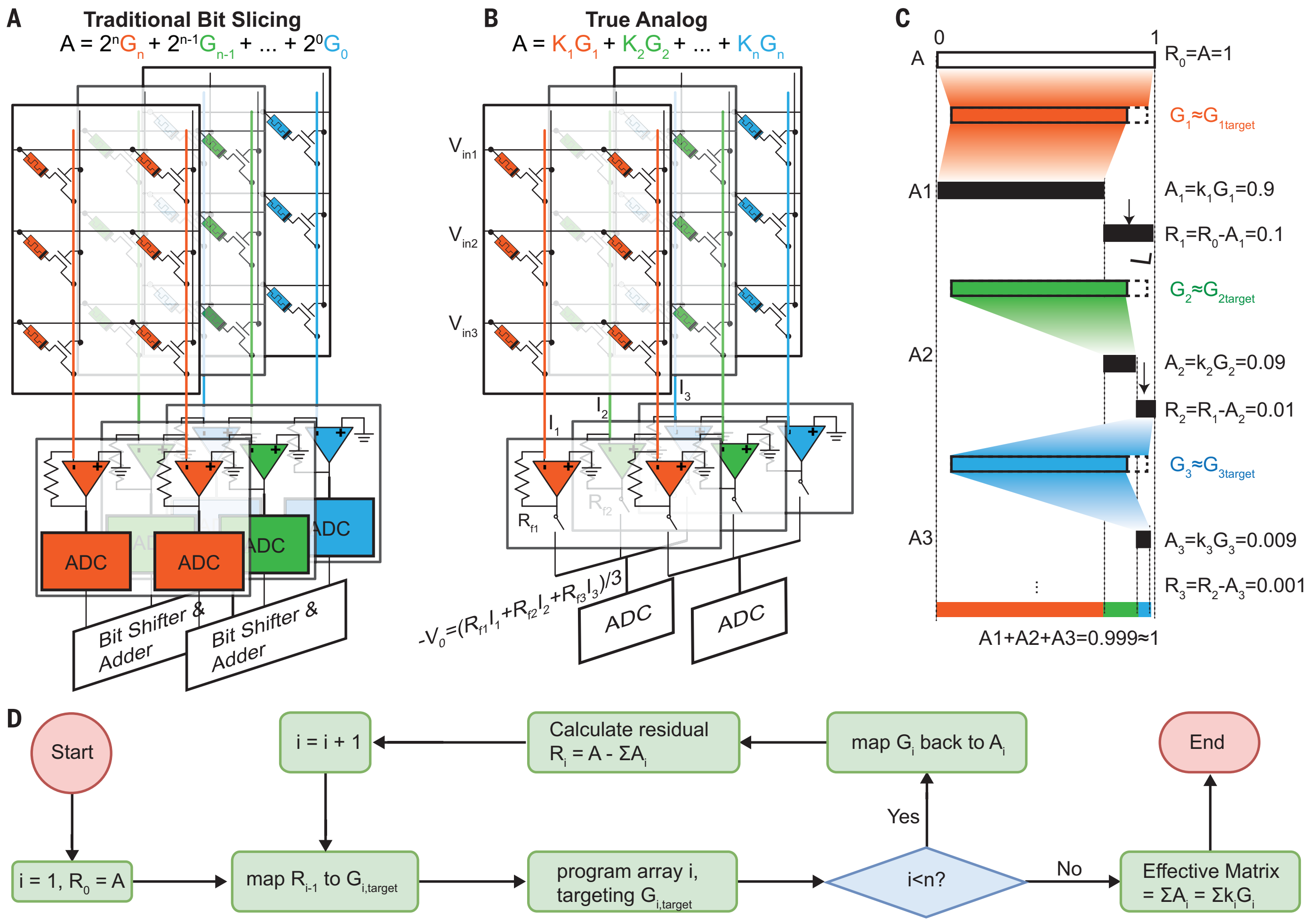


Fig. 1. Comparison of abitrary precision programming and traditional crossbar arrays. (**A**) Traditional crossbar arrays with ADCs and additional postprocessing circuits. (**B**) Proposed arbitrary precision programming circuit with shared ADCs. (**C**) Example of programming a numerical value $A = 1$ into multiple memristor devices step by step. Red, green, and blue represent memristive devices in the first, second, and third subarray. (**D**) Flowchart of the arbitrarily high-precision programming algorithm.

Instead, the proposed true analog approach tries to complete the computation in the analog domain as much as possible. It only converts the computing result to digital at the last step. In practice, owing to yield issues and device-to-device variations, or to keep programming iterations and time reasonable, some devices are less accurately programmed than others and may not meet the predetermined criteria of the traditional bit-slicing approach, resulting in quantization error. In our analog approach, we also use the weighted sum of multiple devices to represent one number but use the subsequently programmed devices to compensate for the conductance error of the previously programmed devices. Such compensation is done by dynamic mapping between the residual value (error) and the device conductance in the proposed algorithm (Fig. 1D, detailed in supplementary text). As shown in Fig. 1B, multiple crossbar subarrays were used for the multistage compensation. The subarrays can be physically placed horizontally, vertically, or three-dimensionally (3D) stacked without substantially changing the algorithm.

A simple example of writing a number a = 1 into three combined devices is drawn in Fig. 1C. In this example, the first device ended up with a 10% programming error, either by one-shot programming or by a read-verify feedback programming method (*26*) with a few programming cycles, as it was programmed to be 0.9 instead of 1. After reading the programming result of this first device, we can program a second device to compensate for this error. The second device likely also had a 10% programming error and was programmed to 0.9, for example. A weight of less than 1 (e.g., 0.1) was used for the second device to ensure the error scales down, with which the second device represented 0.9 × 0.1 = 0.09. Therefore, the combined value of those two devices became 0.9 + 0.1 × 0.9 = 0.99, successfully reducing the total error to only 1% by only two sequential programming operations. Similarly, adding a third device further reduced the total error to 0.1%.

More rigorously, to write any target numerical matrix A, $R_0 = A$ is considered as the initial residual and mapped to a conductance matrix $G_{1,\mathrm{target}}$ within the programmable conductance range of the memristive devices. Because A can have both positive and negative elements whereas the conductance is a physical quantity related to the device, which must be positive and within a certain dynamic range, a mapping method that can map both positive and negative values to a given positive range is needed. One approach is to perform a scaling followed by a shifting so that all elements are shifted into the positive range. This linear mapping method $G^{T}_{1,\mathrm{target}} = K_1R_0 + B_1$ was chosen in our work, where K is a diagonal matrix for scaling, and B serves as a global offset matrix where the elements within the same row are identical so that the entire column of memristors is shifted by the same value. Because there is a transpose operation in the formula $I_o = G^{T} V_i$, the columns of the physical matrix G correspond to the rows of the mathematical matrix A (and

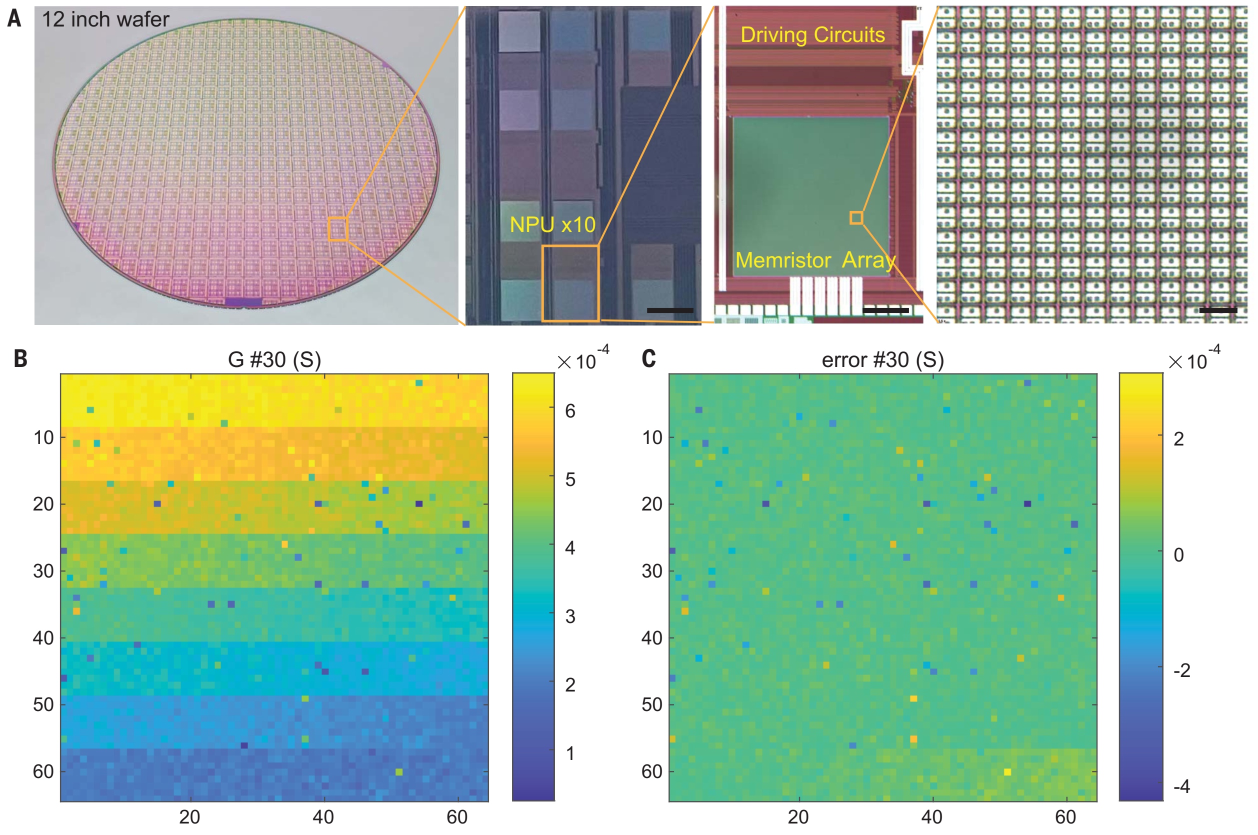


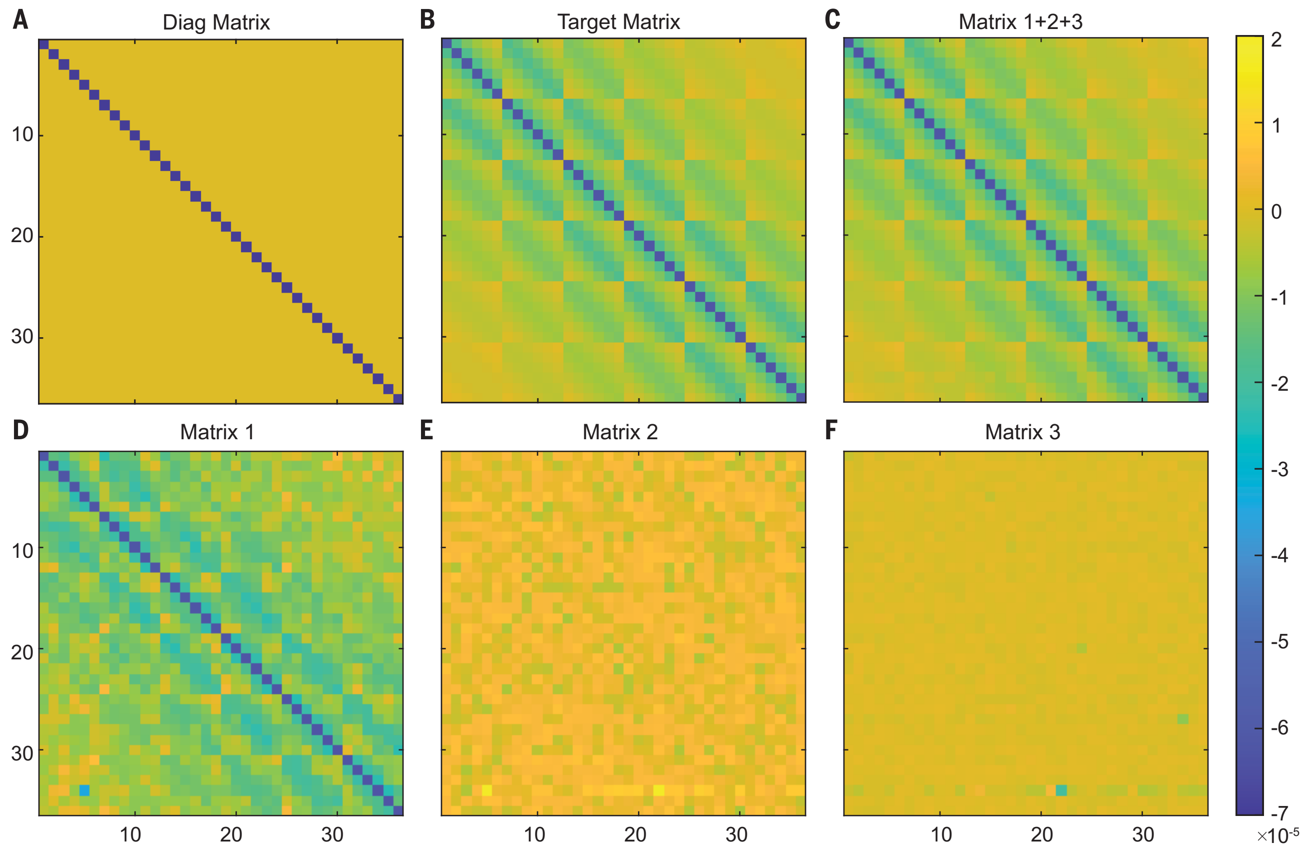
Fig. 2. Photos and programmability of the memristor crossbar array on the SoC. (**A**) Optical images of the wafer and SoC under test. Each chip has ten 256 × 256 1T1R crossbar arrays. Scale bar (from left to right): 2 mm, 500 μm, 10 μm. (**B**) The final conductance map of a 64 × 64 region after 30 programming cycles. (**C**) The absolute error map after 30 programming cycles.

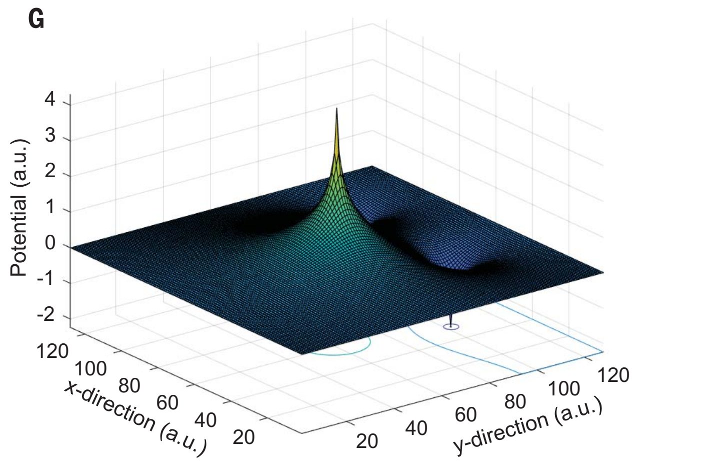
R). K and B are chosen in such a way that each column of $G_{i,\text{target}}$ is mapped to the full conductance range of the memristor. An alternative to this linear mapping method would be using the differential pairs (*42*, *44*), which utilizes two devices to represent one number, and the number is proportional to the difference of the conductances of the two devices. But this approach would double the number of memristors used. Both approaches are compatible with the proposed programming algorithm, but to avoid introducing unnecessary complexity in our description, we assume all numbers are positive in the schematic in Fig. 1.

The first memristor subarray was then programmed targeting $G_{1,\text{target}}$ using fast one-shot programming or the write-verify approach. Because of the analog nature of devices, there was always a programming error between the target $G_{1,\text{target}}$ and the programmed array G_1. G_1 was inversely mapped to the numerical matrix A_1, and the residual matrix $R_1 = A - A_1$ was set as the programming target for the next subarray. Similarly, the second subarray was programmed to G_2, which was converted to A_2, and the residual $R_2 = A - (A_1 + A_2) = R_1 - A_2$ became the target of the third subarray.

One of the key advantages of this approach over the traditional bit-slicing technique is that the scaling factor K in the linear mapping is dynamically calculated and adaptive to the programming performance of each column of the subarray instead of a predetermined value. (See figs. S6 and S7 for comparison with the bit-slicing approach.) This allows faster convergence when the programming error is small and guarantees the largest residual of each column to be monotonically decreasing (converging) when the programming error is large. That is because even if a device in the next subarray is stuck and extremely far away from the target, it cannot be further than the difference between G_{on} and G_{off}, thus its residual cannot be larger than the previous largest residual. As the scaling factors, K, or the weights for subsequent subarrays decrease, the remaining error would also decrease and converge toward zero. Because the actual programming result is read and considered when calculating the next residual, the accuracy can be guaranteed. And with the help of the shrinking scaling factors, the effective precision of the whole array can exceed the device programming precision. In principle, an arbitrarily high precision can be achieved by using more and more subarrays. Such scaling factor granularity was chosen as a balance between using one scaler for the whole subarray and individual scaling factors for every device in the subarray. Using a global scaling factor for the whole subarray would make the entire subarray less effective and cumbered by even a single inaccurate device, and maintaining individual scaling factors for every device would require too much extra computation and could not be implemented in hardware efficiently.

The proposed mapping mechanism could be conveniently implemented in hardware by programming the feedback resistor R_f in the existing trans-impedance amplifier circuits and adding a last row at the subarray for the offset B. The overhead for dynamic scaling factor calculation was negligible because it was only needed once for each subarray and required no additional array reading operation. This calculation had the same complexity as calculating the programming voltage amplitudes for one cycle in the write-verify programming, which usually takes multiple cycles. During the programming of each subarray, the corresponding switch was turned on, and switches of other subarrays were turned off. Both the traditional scheme and the proposed scheme





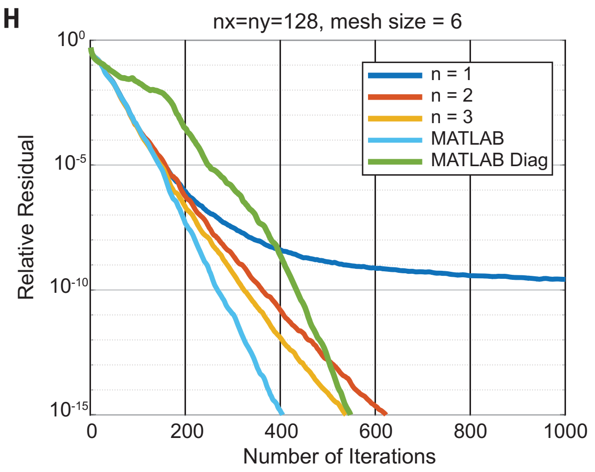


Fig. 3. Experimental results on Poisson solver with arbitrary precision programming with three arrays. (**A**) The classical diagonal preconditioner matrix. (**B**) The target Green's function preconditioner matrix. (**C**) The summation of all three numerical matrices $A_1 + A_2 + A_3$. (**D**) The first subarray, programmed and mapped to A_1. (**E**) The second subarray, programmed and mapped to A_2. (**F**) The third subarray, programmed and mapped to A_3. (**G**) Correct solution of a 128 × 128 Poisson equation example by the hardware solver using all three subarrays on the SoC. (**H**) The residual of the solution over iterations of different settings. n = 1,2,3 results are experimentally obtained with the SoC, whereas MATLAB and MATLAB diagonal results are obtained with software solvers.

require high-resolution ADCs for accurate VMM operations. By dynamically calculating the weight, i.e., the scaling factor implemented by R_f, the proposed scheme also efficiently utilizes the existing ADC precisions to achieve faster and more error-tolerant matrix programming. More hardware accuracy limitations are discussed in figs. S8 and S9.

Such a high-precision programming method enabled high-precision full vector-matrix multiplications. When the input voltages were applied to the rows, switches of all subarrays were turned on simultaneously, and the output currents of all subarrays were naturally weighted and summed together to obtain the total VMM result, which was then sent to the ADCs for a final digitization. The entire VMM process was analog, and the result was only digitized at the last step. Multiple subarrays can share one ADC, which saves a substantial portion of this area- and power-consuming component (*40*, *45*, *46*), as well as other postprocessing digital circuits such as bit shifters and adders for partial products in the traditional approach.

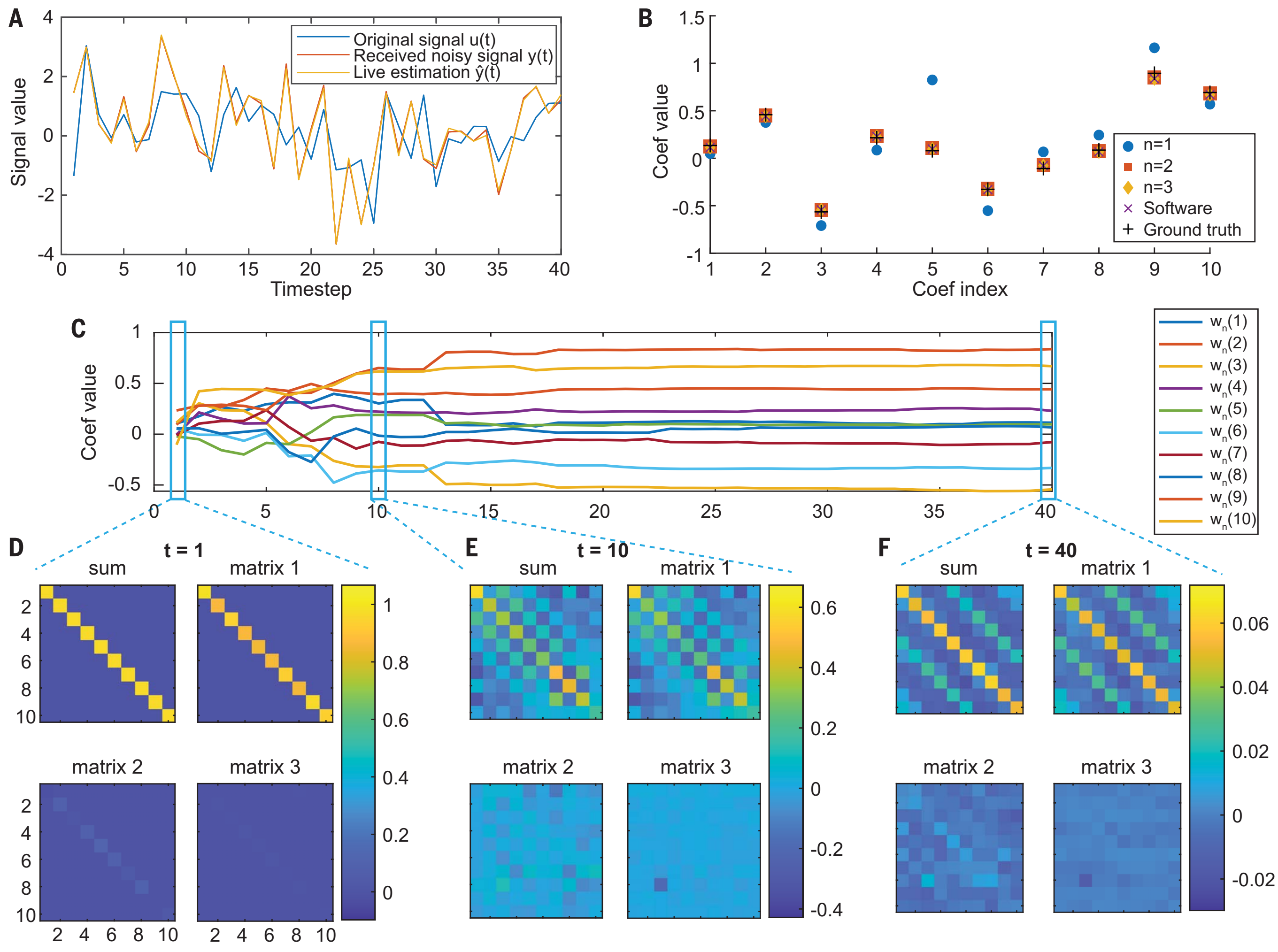


Fig. 4. Hardware recursive least-square filter with the arbitrary precision programming. (**A**) The randomly generated original signal $u(t)$ and the received noisy signal $y(t)$ passing an echoey channel, and the live estimation of the noise-free signal $\hat{y}(t)$. (**B**) The estimated coefficients of the channel with the ground truth. Experiments using n = 1~3 subarrays are performed. With two or more subarrays used, the hardware result is almost identical to the software estimation. (**C**) The history of the coefficient estimations within 40 timesteps. Each line represents one coefficient of the channel. (**D**) The covariance matrix and the numerical matrix of the three subarrays programmed at t = 1. (**E**) The covariance matrix and the numerical matrix of the three subarrays programmed at t = 10. (**F**) The covariance matrix and the numerical matrix of the three subarrays programmed at t = 40.

Moreover, the time and energy overheads incurred in the input preprocessing circuit of the bit-slicing approach can be eliminated by our approach as well.

Experimental demonstration of high-precision solvers

We use the proposed architecture and algorithm to solve PDEs as an experimental verification. The experiments were conducted on two memristor platforms, i.e., a non–fully integrated system featuring lab-made memristors with relatively larger device variations and a fully integrated SoC chip featuring fab-made memristors with improved homogeneities. The former consisted of a 128 × 64 one transistor–one resistor (1T1R) memristor crossbar array and printed circuit board (PCB) driving circuits (see materials and methods and figs. S1 to S3). The latter was a newly developed analog in-memory computing accelerator SoC with 10 neural processing units (NPUs). Each NPU had a 256 × 256 memristor array fabricated in a commercial foundry with much better yield and uniformity than the lab memristors (see materials and methods and figs. S4 and S5). Our experiments on these two platforms verified that the proposed approach worked well for both cases with large or small device variances, and the SoC chip exhibited an especially encouraging performance. Photographs of the unpacked SoC are shown in Fig. 2A. Each memristive cell could be programmed in an analog fashion within the range from 30 to 700 μS by controlling the gate voltage of the transistor in the 1T1R cell. A 64 × 64 region was programmed in a write-and-verify manner to a multilevel pattern within 30 programming cycles (Fig. 2B). A few conspicuous devices were not written to the target because of the device-to-device variability and limited programming cycles (Fig. 2C). Those devices, if not compensated later, would greatly affect the vector-matrix multiplication accuracy and prevent the PDE solver from convergence.

The effectiveness of our arbitrary precision programming method was first verified by using the on-chip crossbar as a VMM core in the high-precision PDE solver. The solver used the preconditioned conjugate gradient (PCG) algorithm. Compared with the vanilla conjugate gradient (CG) method, PCG used a preconditioning matrix to improve the condition number of the system to be solved and thus improve the convergence speed (*47*) (see materials and methods). In hardware, a high-precision VMM core enabled new optimizations

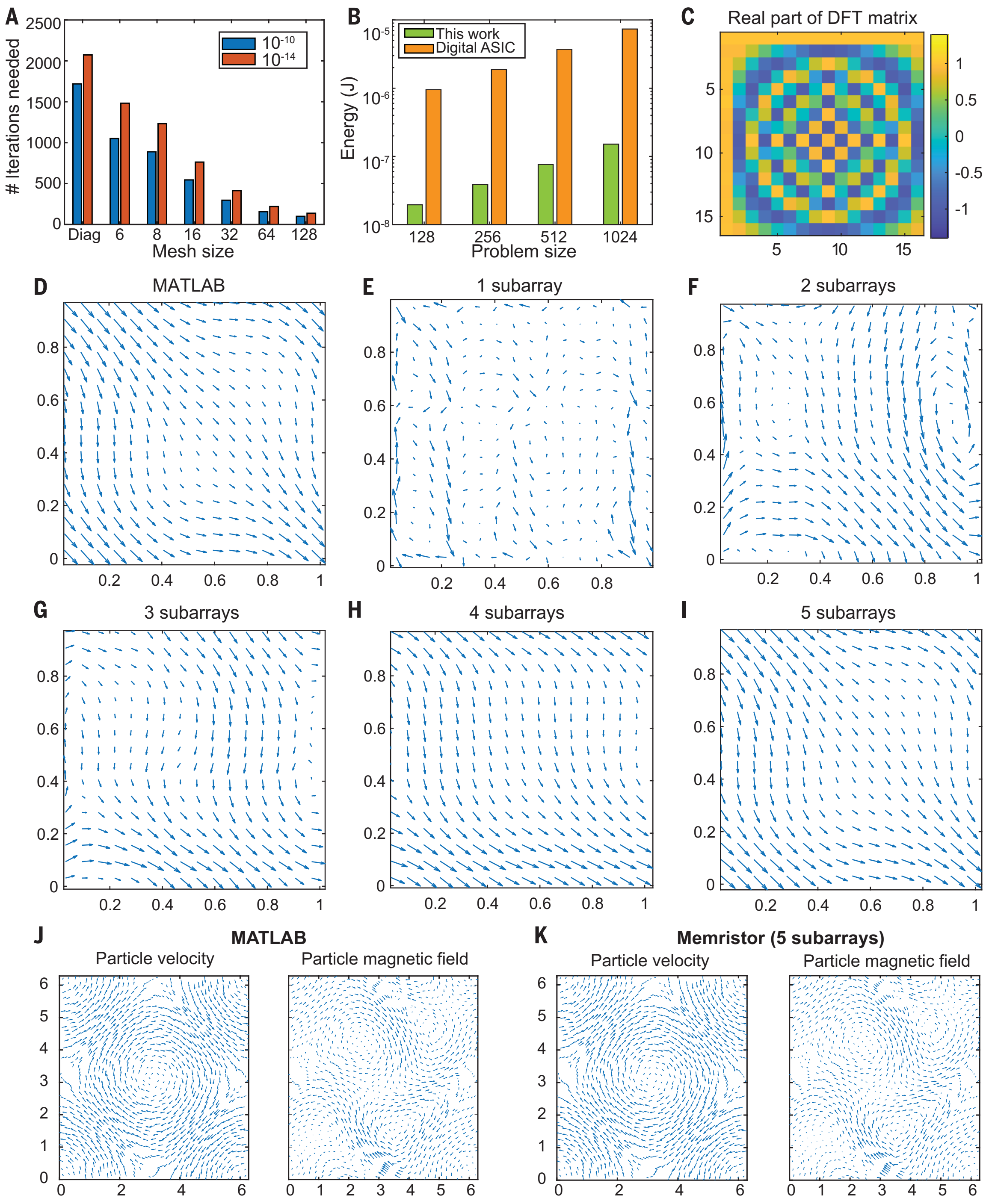


Fig. 5. Scalability, efficiency, and more applications. (**A**) The number of iterations for convergence reduces when the mesh size increases. (**B**) Energy consumption compared with an ASIC design running at approximately the same speed. (**C**) Real part of the DFT matrix (n = 16) used in the N-S equation solver. (**D**) Solved velocity field of a N-S equation at t = 4.8 s solved by MATLAB. (**E** to **I**) Solved velocity field of a N-S equation at t = 4.8 s solved by memristor simulation using one to five subarrays. (**J**) The solved velocity and magnetic flux density field of an MHD problem at t = 2 by MATLAB. (**K**) The solved velocity and magnetic flux density field of an MHD problem at t = 2 by memristor simulation using five subarrays. Each arrow in the quiver plot (D) to (K) shows the direction and relative magnitude of the described field.

that were previously not feasible. As an example of our software-hardware codesign advantage, this VMM core allowed us to use the more efficient Green's function of the problem as the preconditioner (*48*, *49*) for numerous problems, instead of the classical Jacobi (diagonal) preconditioner that has been widely used (*41*, *50*) in digital solvers because of its computational simplicity for digital computers. Using this more efficient but complex preconditioner allowed us to converge faster than digital solvers with simpler preconditioners. To accommodate multiple subarrays in our chip, the problem with a size of $n_x \times n_y$ was downsized into a rougher mesh with a size of $j_x \times j_y$ when performing the hardware preconditioning.

As an example, we solved a Poisson equation with $n_x = n_y = 128$ grids by downsampling it into a $j_x = j_y = 6$ mesh in the preconditioning process (movie S1). In this hardware preconditioning step, the input matrix was flattened as a 1×36 vector to multiply with the flattened 2D physical preconditioner matrix. Hence, the size of the preconditioner matrix needed was 36×36 per subarray. A traditional diagonal preconditioner for the Poisson problem is shown in Fig. 3A. The Green's function for the Poisson equation was calculated explicitly and reshaped to 2D for hardware VMM (Fig. 3B). Up to three subarrays were experimentally programmed to the target Green's function preconditioner matrix for hardware VMM in the PCG algorithm, so the total number of physical devices used is 108×36. The results obtained with the non–fully integrated platform are shown in fig. S10, while the SoC platform generated much improved results as shown in Fig. 3. The programmed subarrays A_1 to A_3 are shown in Fig. 3, D to F, and the effective matrix (Fig. 3C) was the summation of A_1 to A_3. Compared with the traditional single matrix approach (Fig. 3D), Fig. 3C was much closer to the target in Fig. 3B. Additional details on this mapping process can be found in figs. S11 and S12.

We solved the same initial condition of one source and two sinks with different numbers of subarrays enabled in the VMM operation to see the differences in the solution obtained. With only the first subarray of lab-made memristors (non–fully integrated platform), the preconditioner could not be effectively reconstructed in hardware. Thus, the solution did not converge correctly (fig. S10G), which revealed the subpar performance of a normal memristor crossbar in scientific computing without using the approach proposed in this study. When using two or more subarrays, the obtained solution converged to the correct value (Fig. 3G). Compared with similar previous work (*41*) that achieved 2.7% mean absolute error in hardware VMM, the precision of the solution improved enormously as more subarrays were used for VMM, and up to 10^{-15} precision was obtained with three subarrays within 600 iterations (Fig. 3H). Using more subarrays would bring the residual curve closer to the theoretical curve of using Green's function preconditioner, ultimately converging to the theoretical curve. Using a diagonal preconditioner was slower than using Green's function preconditioner because it contained less information and was less effective. Because of the experimental reading variation, there were slight variations on the residual curve for each run, which did not change the above general observations. We did not need to use techniques such as time multiplexing, as all subarrays could compute simultaneously, greatly improving the throughput.

The PCG algorithm uses a fixed matrix, and we have gone a step further by using a changing matrix in the hardware recursive least squares (RLS) filter application (see materials and methods). Suppose a signal $x(n)$ is transmitted over an echoey, noisy channel. It will be received as $y(n) = \sum_{k=0}^{q} w_n(k)x(n-k) + v(n)$, where $v(n)$ represents additive noise. The RLS filter can be used to estimate the channel coefficients w_n and recover the noise-free version of the received signal, $\hat{y}(n)$ (Fig. 4A). At each timestep, it uses the last step's estimation to update the next step. The crossbar array served as the covariance matrix, which was critical in updating the Kalman gain and estimating the coefficients. In this example problem, the noisy window was assumed to be $t = 10$; thus, 10 coefficients were estimated, and the covariance matrix was 10×10. The experiments were done with one subarray, two subarrays, and three subarrays. Their corresponding estimated coefficients are shown in Fig. 4B, along with the software estimation and the ground truth that was randomly generated when setting up the problem. One subarray was not sufficient to accurately estimate the channel coefficients, but two or three subarrays substantially improved the result that overlapped with the software estimation. The updating history of coefficients is shown in Fig. 4C, with covariance matrices on three typical timesteps shown in Fig. 4, D to F. The covariance matrix was updated in hardware in each timestep. It was observed that it changed from the initial diagonal matrix to a checkerboard-like shape in the middle and finally changed to a banded matrix when the estimation became stable. This verified the capability and stability of our programming scheme for dynamic matrices that change during computation.

Given the limited physical crossbar size of our current chip, extended configurations for solving larger equations and other types of PDEs were verified in simulation. Up to five subarrays were programmed in simulations using our noisy memristor model calibrated with lab-made devices and used in the PDE solver to solve the Poison equation. A large writing tolerance of 60 μS was assumed in the writing process. The five matrices and the summed effective matrix were visualized in fig. S13. Even with such a considerable programming error, the effective summed matrix closely approximated the target matrix as more arrays contributed to the summation.

As for the scalability, increasing the mesh size reduced the iterations needed to achieve a specific solution precision, and the numbers of iterations required to obtain 10^{-10} and 10^{-14} precision on a 512×512 problem were listed in Fig. 5A (and table S1 for more problem sizes). We also compared the energy performance with a highly optimized digital system with an application-specific integrated circuit (ASIC), which exhibited an energy efficiency of 7.02 tera-operations per second per watt (*51*) and a latency of 10.4 ns, almost the same speed as our system. We obtained nearly two orders of magnitude energy advantage over the digital system (Fig. 5B and supplementary text).

The proposed programming methods proved more valuable when solving complicated time-evolving problems such as N-S equations and MHD problems. When solving those equations, a subsequent timestep needed to be calculated based on the result of the previous timesteps, so even tiny errors in the previous step could accumulate and propagate, making a high precision critical for each timestep. Our approach was a general programming approach that can serve not only as the preconditioners but also any other matrices, for instance, the discrete Fourier transform (DFT) matrices (Fig. 5C). As an example, we solved N-S equations in a simulation of the motion of fluids over time using spectral method and $n = 1\sim5$ subarrays (movie S2). The simulated velocity field (Fig. 5, E to I) became closer to the MATLAB solver (Fig. 5D) as more subarrays were used. In each timestep, the solution was transformed to the spectral space by multiplying with the DFT matrix and transformed back to the physical space using the inverse DFT after the pressure and diffusion effect were applied in the frequency domain. The input vectors and the DFT matrix were divided into real and imaginary parts to process complex number multiplication.

One advantage of such spectral methods is that they can achieve high accuracy with relatively few grid points. The Fourier transform and inverse Fourier transform were highly parallelizable but could be computationally expensive, which is a perfect example to be solved by the hardware VMM using memristive crossbars. As a last example, we solved complicated MHD problems in which the fluid flow and magnetic fields were coupled together, by exploiting both our hardware FFT technique in solving the N-S subproblem and hardware PCG technique in solving the pressure and magnetic field pressure. The simulated fields with five subarrays nearly perfectly matched our MATLAB solver within 100 timesteps (Fig. 5, J and K).

Conclusions

We have demonstrated an innovative circuit architecture and programming protocol that can efficiently program inaccurate analog devices with arbitrarily high precision within the limit of digital peripheral circuits. This method enables us to execute PDE solvers with high precision, energy efficiency, and throughput. Beyond in-memory computing architecture, it also opens doors for computational applications previously considered infeasible for emerging analog memories, such as scientific computing, neural network training, and complex physical system modeling. The accuracy of analog computing was limited not solely by writing accuracy, but also by reading accuracy. Although our proposed programming method overcame the limitation of writing accuracy caused by the device's writing pulse response variations and a small number of stuck or poorly conditioned devices, hardware computing accuracy was still bound by device reading variation, and peripheral circuit elements like ADC precision. With recent advancements in memristor programming (*26*), the reading variations can be substantially mitigated, making the proposed solver more effective. This codesign approach enables the use of low-precision analog devices for high-precision computing, considerably broadening the application scope of analog computing. Experimental demonstrations with software equivalent precision and a higher efficiency achieved in a large-scale memristor SoC fully integrated in a standard foundry further proved its real-world applicability and readiness for commercialization.

REFERENCES AND NOTES

1. M. Hu *et al*., in *Proceedings of the 53rd Annual Design Automation Conference* (ACM, 2016), pp. 1–6, https://dl.acm.org/doi/10.1145/2897937.2898010.
2. F. Zangeneh-Nejad, D. L. Sounas, A. Alù, R. Fleury, *Nat. Rev. Mater.* **6**, 207–225 (2020).
3. S. Ramakrishnan, J. Hasler, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **22**, 353–361 (2014).
4. L. Danial *et al*., *Nat. Electron.* **2**, 596–605 (2019).
5. E. J. Fuller *et al*., *Science* **364**, 570–574 (2019).
6. G. W. Burr *et al*., *IEEE Trans. Electron Dev.* **62**, 3498–3507 (2015).
7. I. Boybat *et al*., *Nat. Commun.* **9**, 2514 (2018).
8. V. Joshi *et al*., *Nat. Commun.* **11**, 2473 (2020).
9. A. Aziz *et al*., *"Computing with ferroelectric FETs: Devices, models, systems, and applications" in 2018 Design, Automation & Test in Europe Conference & Exhibition* (IEEE, 2018), pp. 1289–1298; https://ieeexplore.ieee.org/document/8342213/.
10. A. I. Khan, A. Keshavarzi, S. Datta, *Nat. Electron.* **3**, 588–597 (2020).
11. R. Berdan *et al*., *Nat. Electron.* **3**, 259–266 (2020).
12. X. Niu, B. Tian, Q. Zhu, B. Dkhil, C. Duan, *Appl. Phys. Rev.* **9**, 021309 (2022).
13. M. Romera *et al*., *Nature* **563**, 230–234 (2018).
14. S. Jung *et al*., *Nature* **601**, 211–216 (2022).
15. Y. van de Burgt *et al*., *Nat. Mater.* **16**, 414–418 (2017).
16. D. B. Strukov, G. S. Snider, D. R. Stewart, R. S. Williams, *Nature* **453**, 80–83 (2008).
17. J. J. Yang, D. B. Strukov, D. R. Stewart, *Nat. Nanotechnol.* **8**, 13–24 (2013).
18. W. Woods, C. Teuscher, *"Approximate vector matrix multiplication implementations for neuromorphic applications using memristive crossbars" in 2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)* (IEEE, 2017), pp. 103–108; https://ieeexplore.ieee.org/document/8053729/.
19. G. Zhou *et al*., *Adv. Electron. Mater.* **8**, 2101127 (2022).
20. Z. Fahimi, M. R. Mahmoodi, M. Klachko, H. Nili, D. B. Strukov, *IEEE Trans. Circuits Syst. I Regul. Pap.* **68**, 4090–4101 (2021).
21. Y. Kim *et al*., *Front. Nanotechnol.* **4**, 1008266 (2022).
22. A. Amirsoleimani *et al*., *Adv. Intell. Syst.* **2**, 2000115 (2020).
23. M. R. Mahmoodi, A. F. Vincent, H. Nili, D. B. Strukov, *IEEE Trans. Nanotechnol.* **19**, 429–435 (2020).
24. C. Bengel *et al*., *Neuromorph. Comput. Eng.* **2**, 034001 (2022).
25. G. Choe, A. Lu, S. Yu, *IEEE Electron Device Lett.* **43**, 304–307 (2022).
26. M. Rao *et al*., *Nature* **615**, 823–829 (2023).
27. T. Gokmen, Y. Vlasov, *Front. Neurosci.* **10**, 333 (2016).
28. Z. Wang *et al*., *Nat. Electron.* **1**, 137–145 (2018).
29. S. Choi *et al*., *Nat. Mater.* **17**, 335–340 (2018).
30. S. Kumar, X. Wang, J. P. Strachan, Y. Yang, W. D. Lu, *Nat. Rev. Mater.* **7**, 575–591 (2022).
31. H. Jiang, W. Li, S. Huang, S. Yu, *"A 40nm Analog-Input ADC-Free Compute-in-Memory RRAM Macro with Pulse-Width Modulation between Sub-arrays" in 2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)* (IEEE, 2022), pp. 266–267, https://ieeexplore.ieee.org/document/9830211/.
32. F. Cai *et al*., Harnessing Intrinsic Noise in Memristor Hopfield Neural Networks for Combinatorial Optimization. arXiv:1903.11194 [cs.ET] (2019).
33. S. Misra *et al*., *Adv. Mater.* **35**, e2204569 (2022).
34. M. Riahi Alam, M. H. Najafi, N. Taherinejad, M. Imani, R. Gottumukkala, *IEEE Trans. Circuits Syst. II Express Briefs* **69**, 2423–2427 (2022).
35. H. Jiang *et al*., *Nat. Commun.* **8**, 882 (2017).
36. H. M. Ibrahim, H. Abunahla, B. Mohammad, H. AlKhzaimi, *Sci. Rep.* **12**, 8633 (2022).
37. S. Larimian, M. R. Mahmoodi, D. B. Strukov, *IEEE Trans. Electron Dev.* **69**, 1816–1822 (2022).
38. I. Richter *et al*., "Memristive accelerator for extreme scale linear solvers" in *Government Microcircuit Applications & Critical Technology Conf. (GOMACTech)* (2015); https://www.hajim.rochester.edu/ece/sites/friedman/papers/GOMAC_15.pdf.
39. M. Le Gallo *et al*., *Nat. Electron.* **1**, 246–253 (2018).
40. Q. Huo *et al*., *Nat. Electron.* **5**, 469–477 (2022).
41. M. A. Zidan *et al*., *Nat. Electron.* **1**, 411–420 (2018).
42. W. Wan *et al*., *Nature* **608**, 504–512 (2022).
43. Y.-L. Zheng, W.-Y. Yang, Y.-S. Chen, D.-H. Han, *IEEE Trans. Comput. Aided Des. Integrated Circ. Syst.* **42**, 740–753 (2023).
44. C. Li *et al*., *Nat. Electron.* **1**, 52–59 (2018).
45. J. Langenegger *et al*., *Nat. Nanotechnol.* **18**, 479–485 (2023).
46. Y. Luo *et al*., *IEEE J. Emerg. Sel. Top. Circuits Syst.* **12**, 445–457 (2022).
47. R. Barrett *et al*., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (Society for Industrial and Applied Mathematics, 1994; https://epubs.siam.org/doi/book/10.1137/1.9781611971538).
48. D. Loghin, "Green's functions for preconditioning," thesis, University of Oxford (1999).
49. T. Ichimura *et al*., *"A Fast Scalable Iterative Implicit Solver with Green's function-based Neural Networks" in 2020 IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)* (IEEE, 2020), pp. 61–68; https://ieeexplore.ieee.org/document/9308819/.
50. J. Vandenplas, M. P. L. Calus, H. Eding, C. Vuik, *Genet. Sel. Evol.* **51**, 30 (2019).
51. P. M. Sheridan *et al*., *Nat. Nanotechnol.* **12**, 784–789 (2017).

ACKNOWLEDGMENTS

The authors thank B. Perot of UMass Amherst and L. Wu of MIT for providing example problems. **Funding:** TetraMem (015542-00001); Army Research Office (grant no. W911NF2120128 and W911NF1810268); Air Force Office of Scientific Research through the Multidisciplinary University Research Initiative program under contract no. FA9550-19-1-0213; US Air Force Research Laboratory (FA8750-22-1-0501); National Science Foundation under contract no. 2023752 and 2036359. **Author contributions:** Conceptualization: W.S., J.J.Y.; Methodology: W.S., J.J.Y.; Resources: J.J.Y., M.R., Y.L., C.L., N.G., M.H., Q.X., P.A.B., M.S.W.C., M.W., W.Y., Z.L., H.Z., Q.W., S.L., L.G.; Software: W.S., Y.L., C.L., F.C.; Investigation: W.S., M.R.; Visualization: W.S., Y.Z., M.R.; Funding acquisition: J.J.Y., M.B., Q.W.; Project administration: J.J.Y.; Supervision: J.J.Y.; Writing – original draft: W.S., M.R., J.J.Y.; Writing – review and editing: W.S., M.R., J.J.Y. **Competing interests:** J.J.Y. and Q.X. are cofounders and paid consultants of TetraMem. J.J.Y. and W.S. have filed a provisional US patent under 35 USC 111(b) titled Memristor array for analog computing. **Data and materials availability:** All data are available in the main text or the supplementary materials.

SUPPLEMENTARY MATERIALS

science.org/doi/10.1126/science.adi9405
Materials and Methods
Supplementary Text
Figs. S1 to S16
Table S1
Movies S1 to S3

Submitted 26 May 2023; accepted 28 December 2023
10.1126/science.adi9405