nature electronics

ARTICLES
https://doi.org/10.1038/s41928-018-0100-6

A general memristor-based partial differential equation solver

Mohammed A. Zidan[1,3], YeonJoo Jeong[1,3], Jihang Lee[1], Bing Chen[1,2], Shuo Huang[1], Mark J. Kushner[1] and Wei D. Lu[1]*

Memristive devices have been extensively studied for data-intensive tasks such as artificial neural networks. These types of computing tasks are considered to be 'soft' as they can tolerate low computing precision without suffering from performance degradation. However, 'hard' computing tasks, which require high precision and accurate solutions, dominate many applications and are difficult to implement with memristors because the devices normally offer low native precision and suffer from high device variability. Here we report a complete memristor-based hardware and software system that can perform high-precision computing tasks, making memristor-based in-memory computing approaches attractive for general high-performance computing environments. We experimentally implement a numerical partial differential equation solver using a tantalum oxide memristor crossbar system, which we use to solve static and time-evolving problems. We also illustrate the practical capabilities of our memristive hardware by using it to simulate an argon plasma reactor.

Numerical computations, such as solving partial differential equations (PDEs), are ubiquitous in scientific research and engineering[1–4], as are many other tasks that involve simulation, prediction and optimization, such as weather forecasting[5] and economics[6]. Analytical PDE solutions are rare, and the vast majority of systems of PDEs are solved (or integrated) using numerical methods that are computationally expensive, involving iterative vector–matrix operations with massive amounts of data. In fact, the most powerful supercomputers are normally built to carry out such large-scale numerical computation tasks[1,7]. The limit in efficiency can be traced to the classical von Neumann computing architecture with separate memory and processing units[8,9], which is the same root cause that affects the hardware performance of other data-centric applications[9–11]. Compared with soft computing tasks such as artificial neural networks, solving PDEs can be considerably more difficult because it typically requires high precision during operations to ensure convergence and an accurate solution. Additionally, the matrices used are typically extremely large, magnifying the hardware challenges.

Recent advances in emerging devices such as memristors[12–17] offer promising alternatives to the classical computing architecture. As a memory device, often termed resistive random-access memory (RRAM), a memristor stores data with its different resistance values: for example, '0' may be represented by a high-resistance state and '1' may be represented by a low-resistance state[2–20]. Programming the device between the resistance states is achieved with a write/erase voltage above a threshold, and data readout is achieved with a read voltage below the threshold. Besides data storage, the device acts as a two-terminal switch that directly modulates the current passing through it, based on the resistance values[14–17]. Hence, memristors can be used to physically perform analogue multiplication in-memory, where the current that passes through the device is equal to the applied voltage multiplied by the stored conductance value, without having to retrieve data from a separate memory[17]. With this approach, common 'multiply and accumulate' operations can be achieved by arranging the devices in a crossbar structure, where the output current of a column represents the dot product between the input voltage vector and the device conductance vector associated with the column, following Ohm's law and Kirchhoff's current law[15–17,21–25].

The co-location of memory and logic, and the high parallelism offered by the crossbar structure (in which vector–matrix multiplication can be carried out in a single readout operation), have generated strong interest in memristor-based computing systems[15–17,21–28]. So far, the focus has been on tasks such as artificial neural networks[15,16,23–28], which typically aim to obtain an approximate or qualitative solution and can thus tolerate limited precision and device variabilities[29–31]. This is not the case for numerical computational tasks such as solving PDE problems, where high-precision and accurate solutions are mandatory, making it more challenging to implement these computing tasks in memristor-based hardware. For example, a well-designed memristor device may provide around 64 different resistance levels[21,32], which is equivalent to six binary bits. However, practical numerical tasks may require up to 64 bits (2^{64} levels) of precision. Additionally, solving PDEs normally involves working with very large matrices that are neither practical nor efficient to fit in a single memristor crossbar. Recent theoretical[33] and experimental studies[34] have used memristor (including phase-change memory) arrays to generate an initial, low-precision guess (seed), and rely on an integrated high-precision digital solver to produce the required high-precision solutions from the seed solution. Such types of accelerators are certainly beneficial, as they reduce the number of iterations required by the digital solver. Determining whether memristor-based hardware can be used to directly perform high-precision computing tasks, however, will enable a better understanding of how broadly memristive hardware can be applied. Such knowledge will help pave the way to build more general memristor-based computing platforms, instead of special-purpose accelerators.

[1]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, USA. [2]Present address: College of Electronic Engineering and Information Science, Zhejiang University, Hangzhou, China. [3]These authors contributed equally to this work: Mohammed A. Zidan, YeonJoo Jeong. *e-mail: wluee@eecs.umich.edu

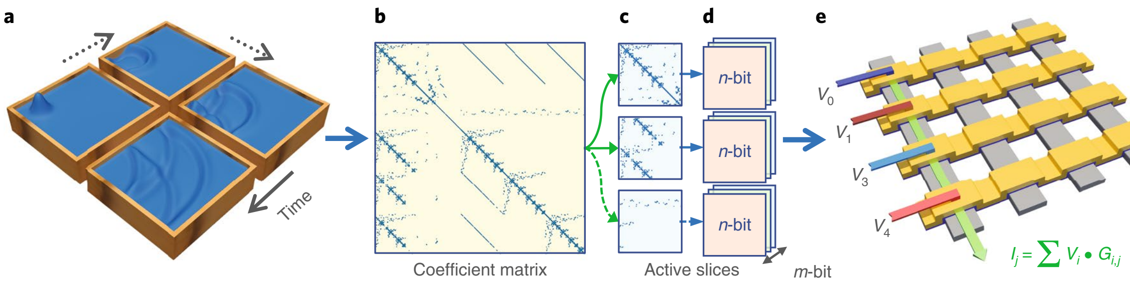

Fig. 1 | High-precision PDE solver based on memristor crossbar. a, An example of a time-evolving system showing a water wave inside a pool at four different time instances. **b**, An example sparse coefficient matrix used in numerically solving PDEs. **c**, The coefficient matrix is sliced into equal-sized patches, and numerical calculations are performed only for the active (non-zero) slices. **d**, Each high-precision (m-bit) active slice is mapped into multiple arrays, each representing a portion (n-bit) of the desired precision. **e**, The values of the elements in the n-bit slice are mapped as conductance values in a memristor crossbar of the same size, and vector–matrix multiplication operations of the slice are performed by supplying the input vector as voltage pulses to the rows and reading out the current outputs at the columns.

In this Article, we present a memristor-based in-memory computing system, including a complete hardware and software package, that can effectively address the concerns of limited device precision and crossbar size. We experimentally solve static and time-evolving problems using the memristor crossbar-based hardware system, with results comparable to those obtained from digital solvers. We further verify the system's performance in a real-world task, where the memristor-based PDE solver is used as part of the workflow in a comprehensive plasma-hydrodynamics simulator package for the type of plasma systems used for plasma etching. We achieved reliable results comparable to conventional digital PDE solvers, with improvements in power efficiency and throughput.

High-precision memristor computing system

A system of PDEs describes the relationship between multiple variables and their partial derivatives. Typically, a system of PDE is solved numerically by discretizing space (and/or time) into grid points such that the partial derivatives at one point can be reduced into combinations of the variable values at several neighbouring grid points. Afterwards, the problem is mapped to matrix form, with the numerical coefficients representing linearized operators between variables at neighbouring grid points. The resulting coefficient matrix can be very large but is typically sparse. This process is performed during the initial problem formulation stage, using techniques such as finite-difference, finite-element or finite-volume methods (see Methods). Iterative methods are then used to estimate the variable values at the grid points through the coefficient matrix and the system's boundary conditions (Fig. 1a). These operations can be performed through a series of vector–matrix operations that we aim to compute in memristor crossbars.

For practical systems, the coefficient matrix can be very large: for example, a 2D system with a 100×100 grid will result in a coefficient matrix with $(10^4)^2=10^8$ elements. However, the coefficient matrix is also typically sparse, with only a very small fraction of non-zero elements, as shown in Fig. 1b (see examples in Methods). This makes it difficult and inefficient to map the coefficient matrix into a single memristor array. By taking advantage of the sparsity, we can divide the matrix into equally sized slices and map only the active slices (the ones containing non-zero elements) into memristor crossbars, as shown in Fig. 1c. By doing so, practical crossbar sizes, for example 16×16 or 32×32, can be used to map the active slices, while greatly improving the hardware utilization. Because during vector–matrix operations all devices are selected, device nonlinearity will not play an important role for such small crossbar arrays[17]. Using smaller crossbar arrays also minimizes parasitic effects due to series resistance, sneak currents and imperfect virtual grounds[23,35], thus making it feasible to build practical hardware systems using passive crossbar arrays—that is, without selectors.

We also show that the low native precision of memristor devices can be extended through the use of multiple crossbars, where each crossbar represents a given number of bits (Fig. 1d). This precision expansion approach is similar to the techniques used in digital circuits, where binary (two-level) physical values, such as capacitor voltages in a dynamic random-access memory, are used as the basis of high-precision computing systems. Similar approaches have also been proposed to improve the effective precision (for example, effective bitwidth of the weights) in memristor-based neural networks[24–28]. Assuming that a memristor can natively support a number l of resistance levels, the goal is thus to perform high-precision arithmetic operations using base-l numbers, analogous to the use of base-2 numbers in digital circuits. At the single crossbar level, analogue vector–matrix multiplications are performed directly between an input vector, represented by the voltage pulses applied to the rows, and the coefficient matrix elements, represented by the memristor conductance values, as shown in Fig. 1e. By summing results from the partial products from these base-l operations, the desired output for the extended precision can then be obtained (see Methods).

We experimentally implemented the proposed approach in a complete hardware and software package, as shown in Fig. 2a. The hardware is based on a Ta_2O_{5-x} memristor crossbar array, which is wire-bonded and integrated onto a printed circuit board (PCB) to carry out the coefficient matrix storage and vector–matrix multiplication tasks, shown in Fig. 2b. The software, based on Python, performs system-level operations including matrix slicing, mapping the problem to the hardware using the precision-extension task and controlling the iterative processes (see Methods and Supplementary Fig. 1). The software also provides an interface between the hardware and the end user for data input/output (see Supplementary Fig. 2 for the program interface).

A scanning electron microscopy (SEM) image of the as-fabricated memristor crossbar is shown in Fig. 2c. For reliable array operation, the initial forming voltage V_{form} required to establish switching behaviour should be low, as a high V_{form} can lead to high voltage drop on devices sharing the same electrode, even under common protective voltage schemes (that is, for the $V/3$ scheme, the voltage on these half-selected devices is about $(1/3)V_{\mathrm{form}}$)[36], leading to damage of already-formed devices in the same array. The Ta_2O_{5-x}-based memristor devices using a thin (3.5 nm) Ta_2O_{5-x} switching layer offer low forming voltage (1.25 V) and stable switching characteristics, as shown in Fig. 2d. The forming, set and reset voltages measured from multiple (26) devices are shown in Fig. 2e, obtained using a pulse programming method through the test board. The

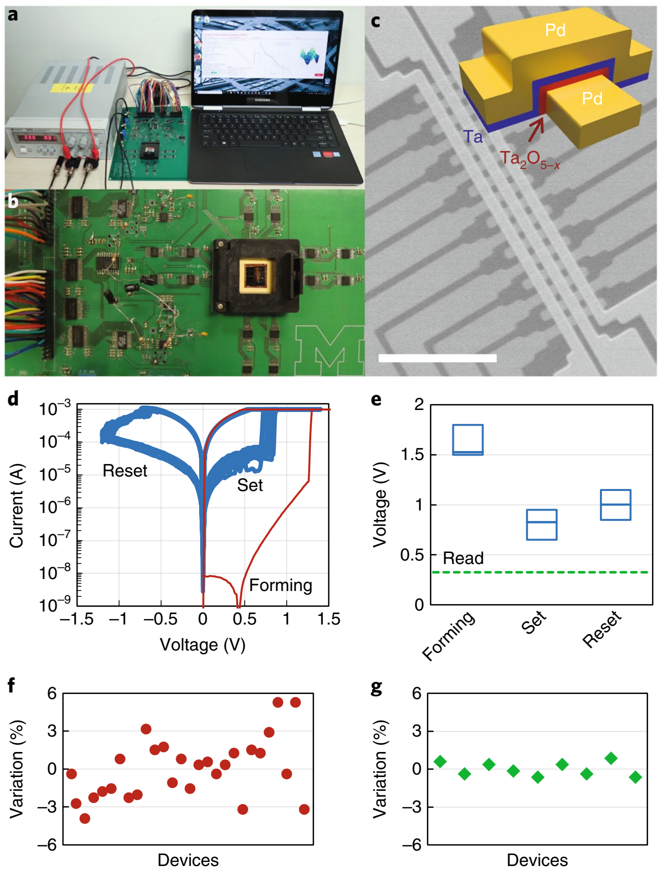

Fig. 2 | Hardware set-up and device measurement. a, A photograph of the memristor-based PDE solver system in operation, showing the hardware set-up controlled by a software package. **b**, A photograph of the test board with the memristor crossbar chip mounted on it. **c**, SEM image of the as-fabricated 16 × 3 crossbar array. Inset: device structure with a thin Ta_2O_{5-x} resistive switching film sandwiched by a Ta top electrode and a Pd bottom electrode. Scale bar, 25 µm. **d**, Current–voltage (d.c.) measurements showing the original forming process and 10 subsequent set/reset processes. A low forming voltage and reliable resistive switching can be obtained. **e**, Pulse measurement results from multiple devices (26 cells), showing narrow distributions ($\sigma < 0.1$ V) for forming, set and reset voltages. **f**, Variation of device conductance obtained from pre-determined programming conditions without any feedback mechanism. A 5.3% conductance variation is achieved. **g**, With the write–verify approach (see Methods), the conductance variation is reduced to 0.85%, making it possible to implement the PDE solver in memristor-based hardware.

forming voltage V_{form} is less than $3V_{set}$, ensuring that high voltage will not be applied to the half-selected devices. Moreover, the devices show narrow distributions ($\sigma < 0.1$ V) for forming, set and reset voltages, making these devices well suited for passive crossbar array operations.

However, the inherent stochastic ion migration processes in the set and reset stages lead to sizable device variability[37,38], as shown in Fig. 2f. Without any feedback mechanism during programming, a cell-to-cell variation of 5.3% may occur, limiting the native precision of a single device to four bits. Lower device variability can be obtained by using a write–verify feedback method[39,40] (see Methods and Supplementary Fig. 3), as shown in Fig. 2g, leading to a cell-to-cell variation of <1%. Combined with the precision-extension technique discussed above, this device system is successfully used to experimentally demonstrate the proposed high-precision PDE solver system.

Poisson's equation example

The first test using the memristor-based system was static problems. This class of equations describes spatial relationships between the variables at a steady-state condition. Examples include elliptic PDE systems such as Laplace's ($u_{xx} + u_{yy} = 0$) and Poisson's ($u_{xx} + u_{yy} = f(x, y)$) equations[41]. Typically, elliptic and other systems of PDEs can be numerically formulated as solving an $A \cdot \mathbf{X} = \mathbf{B}$ problem, where $\mathbf{X}$ is the unknown vector to be solved, A is the coefficient matrix, and $\mathbf{B}$ is a constant vector containing the boundary conditions. While such problems can be solved using several numerical techniques, here we adopted the Jacobi method[42] as it can be directly mapped to the memristor crossbar hardware system using entirely iterative vector–matrix operations (see Methods).

Specifically, at iteration k, a new estimate of the unknown vector $\mathbf{X}$ is computed for the next iteration $k + 1$ as:

$$\mathbf{X}^{(k+1)} = \mathbf{C} - R \cdot \mathbf{X}^{(k)} \tag{1}$$

where R is a modified coefficient matrix with the diagonal elements removed, and $\mathbf{C}$ is a constant vector that includes the boundary values. Equation (1) can be implemented in a crossbar array by mapping R and $\mathbf{C}$ to the crossbar with numerical values represented by the memristor device conductances, as shown in Fig. 3a. By applying $\mathbf{X}^{(k)}$ to the input rows of this crossbar as voltage pulses, the output currents collected at the columns represent the new estimated value of $\mathbf{X}^{(k+1)}$. The process is then repeated iteratively by feeding $\mathbf{X}^{(k+1)}$ to the system as the next input until the desired accuracy is achieved.

Based on this approach, we experimentally solved a Poisson's equation test case at 16-bit precision using our hardware set-up. The problem is defined as:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2\sin(x)\cos(y) \tag{2}$$

The expected solution of the problem is shown in Fig. 3b, along with the boundary conditions shown in Fig. 3c. Equation (2) is then converted to the matrix form using a five-point numerical stencil, through the finite-difference method (see Methods). Here we used a uniform grid, which typically results in a symmetric matrix with the non-zero elements along the penta-diagonal directions. In this example, only four elements along any row are non-zero after removal of the diagonal elements following the Jacobi method. Specifically, when dividing the matrix into 3 × 3 slices (if the number of grid points is a multiple of 3), only four different patterns are needed. We thus divided the coefficient matrix into 3 × 3 slices and wrote the four patterns into a 16 × 3 array (Supplementary Fig. 4). Time multiplexing is then used to obtain the vector–matrix products from the crossbar output for different slices sharing the same pattern. The different partial products are then summed through the board to obtain the final output. Time multiplexing is not required in general, as parallel processing of the slices can be obtained if a larger crossbar hardware system can be built.

Using the proposed precision-extension approach any target precision can potentially be achieved in the hardware system. Here, 16-bit precision is needed for the input and output vectors to achieve convergence and correct solutions. With the proposed approach, the Poisson's equation was iteratively solved using the memristor-based system at 16-bit precision. We used a simple coarse-to-fine grid approach to improve the numerical convergence speed, where we started with a 3 × 3 grid and ended with a 30 × 30 grid (over the same area) after ten system iterations. After each iteration, the solution is updated and the grid size is increased, where the coarse grid solution acts as an initial approximation for the next finer grid.

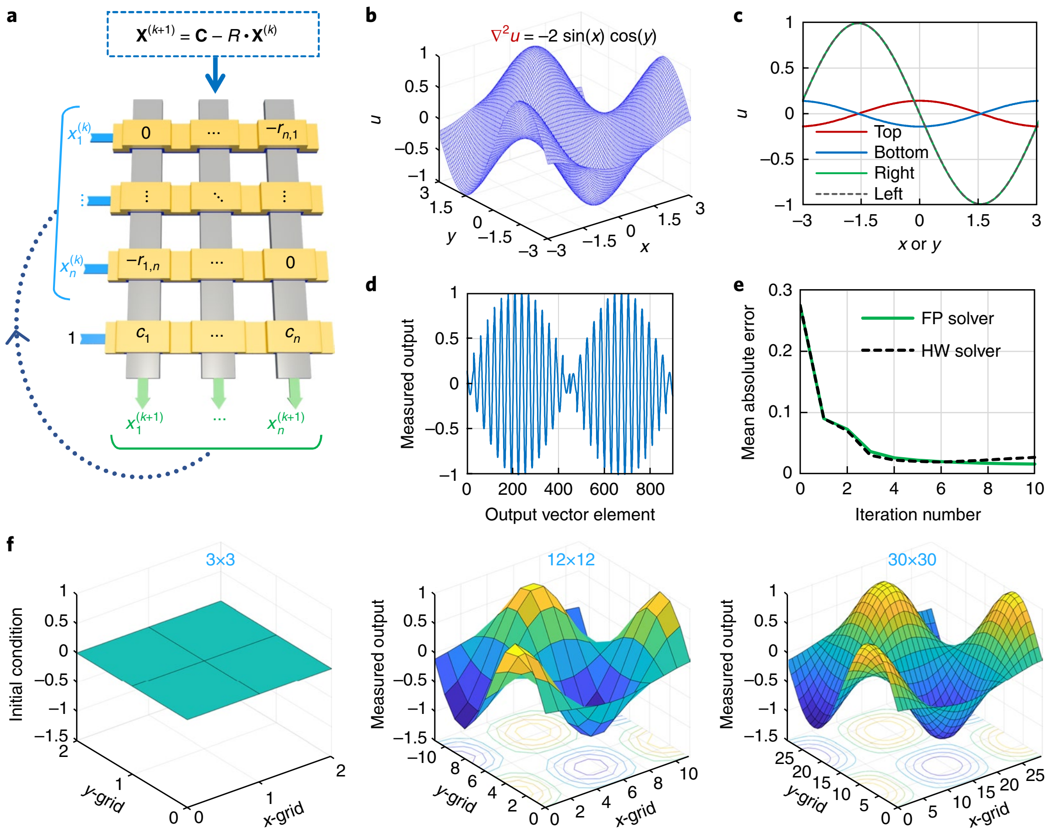

Fig. 3 | Experimental demonstration of solving a Poisson's equation. a, Mapping the Jacobi method (used to iteratively solve Poisson's PDE) to a memristor crossbar-based system. A single crossbar is shown for illustration purpose. The solution is iteratively computed by applying the vector $\mathbf{X}^{(k)}$ as voltage pulses to the rows of the crossbar and collecting the output currents at the columns which represent the numerical value of $\mathbf{X}^{(k+1)}$. **b**, Poisson's equation used as a test example, and a 3D plot of the intended solution. **c**, The boundary conditions used in the example, measured at the four edges of the system. **d**, Final measured output from the memristor-based PDE solver hardware, for the 900 grid points in the 30×30 mesh. **e**, Evolution of the mean absolute error for the memristor-based (hardware (HW)) solver and a floating-point (FP) solver, measured against the exact numerical solution. A coarse-to-fine grid technique was used during the iterations, where the system started with a 3×3 grid and ended with a 30×30 grid. **f**, Reconstructions of the initial condition (with a 3×3 grid), and the measured outputs at iteration numbers 4 and 10, for grid sizes of 12×12 and 30×30, respectively.

These grids generate coefficient matrices of sizes ranging from 81 elements to 8.1×10^5 elements, while the number of active non-zero slices ranges from 7 to 1.42×10^3.

The output of the memristor crossbar system for the final 30×30 grid points is shown in Fig. 3d. The measured output was compared with the exact solution obtained using the inverse matrix technique for the same equation, and the mean absolute error (MAE) was measured and plotted against the iteration number, as shown in Fig. 3e. For comparison, results obtained from a standard floating-point solver are also plotted. The results show that both solutions converge at roughly the same rate with similar error figures. Ten iterations were enough for the hardware system to achieve an MAE below 2.7% compared with the exact solutions. Figure 3f shows three-dimensional (3D) reconstructions of the experimentally obtained solution from the memristor-based solver, at different iteration numbers. The solution after 10 iterations shows an excellent match with the expected solution. The small differences in the results obtained from the memristor-based solver and the floating-point solver in Fig. 3e are due to the device variability in the hardware system, as the precision-extension technique was only applied to the input vectors in this experiment. By applying the precision-extension technique to also minimize the device variation effects, a precise match between the memristor-based solver and the floating-point solver can be obtained (see Methods and Supplementary Figs. 5 and 6).

Time-evolving problem example

The second set of PDEs that we tested using the memristor-based hardware system are time-evolving problems. In this case, the PDE includes partial derivatives with respect to time along with other variables. Typically, numerical methods such as finite difference are used to map the equation to the matrix form[42], and the new state of the system is computed in an iterative manner. At the hardware level, this process is again reduced into a series of vector–matrix operations, where the output of the crossbar array at one time frame is used as the input for the next iteration, as shown in Fig. 4a.

As an example, we experimentally solved a 2D wave equation using the memristor-based set-up. The equation is:

$$\frac{\partial^2 u}{\partial t^2} = \theta^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \zeta\frac{\partial u}{\partial t} \qquad (3)$$

where u is the wave amplitude, θ is the wave speed constant and ζ is a decay (damping) constant. This equation represents a classical physics description of the propagation of 2D waves, and can be used to visualize a shallow water surface in a computationally inexpensive manner[43].

We solved the wave equation in a 60×60 grid, with a wave speed constant of $\sqrt{0.37}$, a decay constant of 2.5×10^{-2}, spatial

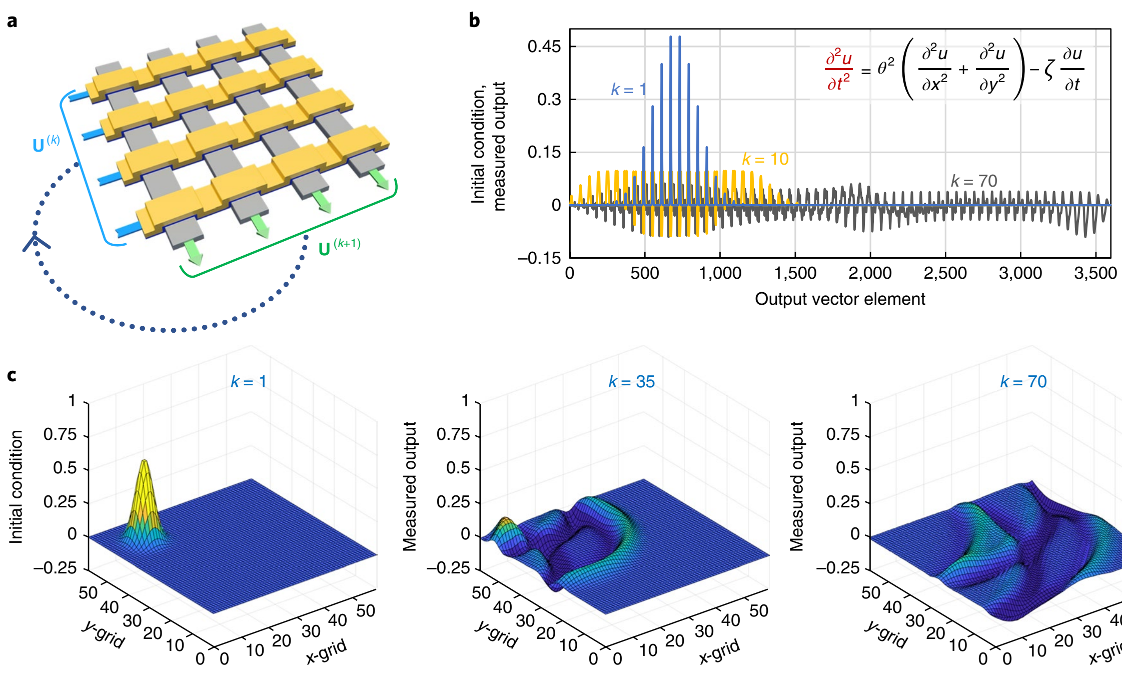

Fig. 4 | Experimental demonstration of solving a damped 2D wave equation. **a**, A general approach to solve time-evolving problems using memristor crossbar arrays. The output currents represent a new estimate for the next time step. **b**, The initial condition (at $k=1$) and the measured outputs of the 10th and 70th iterations for a damped wave equation PDE test case (inset). The test problem is iteratively solved in a 60×60 grid. **c**, Reconstructions in 3D of the initial condition, showing a droplet touching the water surface with a Gaussian-shaped surface profile, and the measured outputs at iteration numbers 35 and 70, where the z-axis represents the water surface level.

steps $h_x = h_y = 0.1$ and time step $\Delta t = 0.1$. Using the finite-difference method, equation (3) is re-written as:

$$\mathbf{U}^{(k+1)} = \alpha_1\mathbf{U}^{(k)} + \alpha_2\mathbf{U}^{(k-1)} + \alpha_3(A\cdot\mathbf{U}^{(k)}) \quad (4)$$

where $\mathbf{U}$ is the targeted solution vector, k is the iteration number, α_1, α_2, α_3 are constants based on θ, ζ, h_x, h_y and Δt, and A is the coefficients matrix (see Methods). Using a five-point stencil to generate the coefficient matrix, the matrix A contains 1.3×10^7 elements, but less than 0.14% of the elements are non-zero. After removing the diagonal elements, the sparse coefficient matrix is divided into 3×3 patches with a total of 5,840 active slices using the software package. These slices follow four different patterns which are then mapped directly to four 3×3 crossbars in the 16×3 crossbar (see Supplementary Fig. 4). Similar to the static problem example, time-multiplexing was used to perform vector-matrix operations on the 3×3 crossbars for slices sharing the same pattern.

As an initial condition, we set $\mathbf{U}^{(1)}$ to be a Gaussian shape representing a droplet touching the water surface. The water droplet initiates the 2D wave, and iterative operations were performed through the memristor-based system to solve the evolution of the water wave. The input and output vectors are encoded as 16-bit numbers. Precision-extension techniques were applied to both the input vectors and the devices to reduce error propagation in the time-evolving iterations (see Methods and Supplementary Fig. 6). We ran the process for 70 successive iterations to solve the wave propagation through the water pool and its reflection from the pool edges. The initial input vector to the system at $k=1$ and the measured outputs at $k=10$ and 70 are shown in Fig. 4b. The 3D reconstructions of the solution from the experimentally measured output of the memristor-based hardware system are in Fig. 4c, showing a snapshot of the wave propagation at different times, and verify the system's ability to solve this time-evolving problem. More examples of the solutions (Supplementary Fig. 7 and Supplementary Video) are in the online Supplementary Materials.

Plasma reactor simulation

Finally, we tested the approach by inserting the memristor-based PDE solver into the workflow of a plasma-hydrodynamics simulator. The specific problems involved simulation of a plasma sustained in argon gas of the type commonly used in the semiconductor industry for etching and deposition. A schematic of an inductively coupled plasma (ICP) reactor system is shown in Fig. 5a (see Supplementary Note 1). Numerical PDE solvers provide the core functions of industry-standard simulators for such systems, such as the hybrid plasma equipment model (HPEM)[44] (see Methods). In this particular implementation, the simulation goes through a hierarchy of outer loops that provide densities of charged particles, charges on surfaces and voltage boundary conditions on metal surfaces. These parameters are produced by solving a set of fluid-dynamics-like equations. At the innermost loop of the simulator is the solution of Poisson's equation for the electric potential at the future time $t+\Delta t$. The particular implementation of Poisson's equation is semi-implicit where charge densities at time t are augmented by predictions of charge densities at $t+\Delta t$. The solution for the electric potential (and electric field) is then used to update the fluid dynamics equations to obtain new charged particle distributions, and the process is then repeated for each time step. Here, we directly replaced the standard floating-point solver used within the HPEM package to solve the Poison's equation (the DSLUCS subroutine[45]; see Methods) with our memristor solver module, as shown in Fig. 5b. Care was taken to make the process fully transparent to the user. The entire structure of the HPEM package remained the same, with the only change being which subroutine is called to solve the system of PDEs—the call to DSLUCS or the call to the interface to the memristor-based solver.

The ICP system was simulated using a 94×52 grid mesh and a coefficient matrix having 2.4×10^7 elements. Within the inner loop, Poisson's equation for electric potential is formulated as an $A\cdot\mathbf{X}=\mathbf{B}$

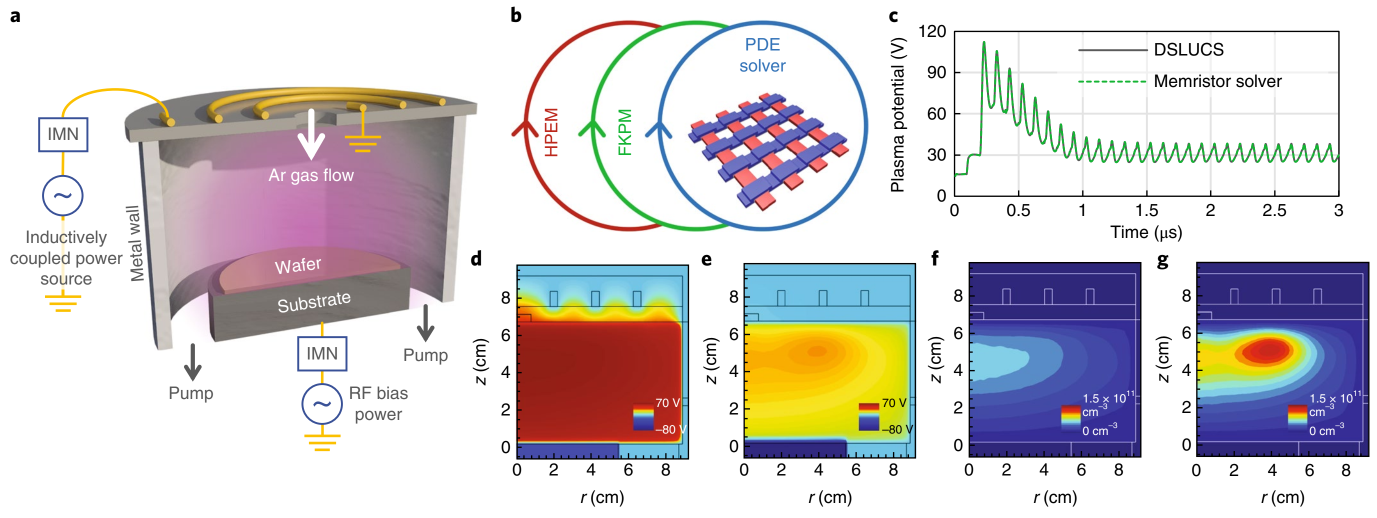

Fig. 5 | Argon plasma reactor simulation. a, A 3D illustration representing the plasma reactor chamber to be simulated. IMN, impedance-matching network. **b**, Schematic of the HPEM package. The HPEM iteratively calls a fluid kinetics Poisson module (FKPM) that in turn calls a subroutine to compute the numerical solutions of Poisson's equations. **c**, Simulation results of the plasma reactor showing the plasma potential versus time during the plasma ignition process, for results obtained using the standard floating-point solver (DSLUCS) and the memristor-based solver . The system is solved in a 94 × 52 mesh, representing only the right half of the chamber owing to its symmetry. The argon gas pressure is 20 mTorr, the power delivered by the antenna is 150 W and the radiofrequency (RF) bias on the substrate has a 100 V amplitude. **d**,**e**, Contour maps of the computed plasma potential using the memristor solver at $t = 0.5$ μs (**d**) and $t = 3$ μs (**e**). r is the radius of the reactor chamber. **f**,**g**, Contour maps of the computed electron density using the memristor solver at = 0.5 μs (**f**) and $t = 3$ μs (**g**).

problem. As such, a similar approach to the static problem example was used. Note although the mesh in the HPEM is structured, the crossbar method is also applicable to unstructured meshes using finite-element[42] or finite-volume methods[46]. The structure of the mesh and method of discretization of the PDEs may change the format of the A matrix but does not impact the solver's operation. To make certain that the crossbar approach is general enough to address all such possibilities, we divided the matrix into 32 × 32-sized slices assuming no common patterns in the coefficient matrix (see examples of an unstructured coefficient matrix in Supplementary Fig. 8). Each slice was then treated independently to perform local vector–matrix operations. A 64-bit precision was implemented in the memristor solver subroutine, by mapping every active slice to multiple memristor crossbars through the precision-extension technique. After solving the Poisson's equation at each time step, the solutions were provided to the HPEM package, the coefficient matrix updated, and the processed was repeated. This process was performed in an automated manner, with the memristor solver fully integrated into the HPEM package as a standard subroutine.

Owing to the large matrix size required to address the plasma transport, the problem was solved through simulation using a device model that represents actual device parameters, unlike the previous two examples which were solved experimentally using physical memristor crossbars. The device simulator incorporates matrix slicing and precision-extension techniques, while accounting for device non-idealities and other circuit details. The simulation results obtained from HPEM using the crossbar-based simulator for solutions of Poisson's equation are shown in Fig. 5c. The time evolution of the plasma potential inside the ICP reactor from the crossbar simulator are compared to the results of an otherwise identical calculation using the standard floating-point solver based on the DSLUCS subroutine, also shown in Fig. 5c. The memristor-based solver produces results match well with those obtained using the double-precision (64-bit) floating-point DSLUCS solver.

Results obtained using the memristor-based subroutine, showing the time evolution of the plasma potential and the electron density during the initial 3 μs of the plasma ignition, are plotted in Fig. 5c–g. The simulation clearly captured the initial quick rise of the plasma potential and the stabilization to a quasi-steady state after 1 μs (Fig. 5c). The oscillation in the plasma potential with a period of 0.1 μs results from the application of the 10 MHz RF bias on the substrate. Snapshots of the plasma potential and the electron density during the initial ignition ($t = 0.5$ μs) and after the plasma stabilization ($t = 3$ μs) are shown in Fig. 5d–g. These snapshots reveal the evolution of the plasma potential and the electron density. Specifically, the electron density has a steady-state distribution that is maximum at approximately half-radius towards the top of the chamber, as a result of the boundary conditions and polarization of the electric field produced by the antenna. It is in this region that the inductively coupled electric field from the antenna is maximum, producing a torus-shaped region of power deposition. Results from HPEM, such as those shown in Fig. 5 that reveal the electron density distribution and plasma potential, have been extensively used by the semiconductor industry to optimize plasma tool design and help develop critical etching and deposition processes. The ability of the memristor-based system to produce accurate simulation results confirm the potential of the proposed system to mitigate device-level limitations and provide efficient numerical computing hardware systems for complex real-world applications.

Conclusions

We have demonstrated that memristor-based in-memory computing systems can be used to process tasks requiring high precision and accurate solutions, beyond what has been demonstrated for soft computing tasks in which high device variability may be tolerated. Despite the limited native precision offered by the devices, architecture-level optimizations such as the precision-extension techniques proposed here can effectively lead to computations achieving 64-bit accuracy. We experimentally demonstrated a high-precision memristor crossbar-based PDE solver. Using a tantalum oxide memristor crossbar, we solved elliptic and hyperbolic PDE equations representing static and time-evolving systems, for the widely used Poisson's equation and a classical water wave propagation problem, respectively. We further incorporated the crossbar-based PDE solver in a large-scale simulation package, HPEM, and used the package to simulate a real-world system: plasma evolution in an ICP reactor with accuracies matching those obtained from floating-pointing solvers.

Our studies showed that challenges including device variability, limited equivalent precision and limited on/off ratio can be successfully addressed even for high-precision computing tasks. Additionally, the precision-extension approach used in our work allows the system to dynamically adjust the system precision as needed, and thus can efficiently allocate hardware resources depending on the task requirement on hand. As a result, a common hardware platform may be used to process different tasks, including both soft computing and hard computing problems. We believe that such demonstrations, showing that memristor crossbars can be used to directly solve high-precision computing tasks (instead of playing a supporting role to a digital system), broaden the appeal of memristor-based hardware systems and pave the way for the development of more general-purpose, memristor-based computing systems[17].

We anticipate a fully integrated computing system based on arrays of memristor crossbars monolithically integrated on complementary metal-oxide-semiconductor supporting circuitry[26] (Supplementary Notes 1 and Supplementary Fig. 9) can offer a scalable computing system with very high processing speeds and power efficiency, owing to its ability to natively compute information in-memory and to its high level of parallelism. Our analysis, after accounting for the power consumption in the crossbar and peripheral circuitry, shows that the proposed memristor-based in-memory computing system can considerably outperform existing and emerging approaches including application-specific integrated circuits (Supplementary Notes 2 and Supplementary Fig. 10), and suggest the proposed memristor-based computing hardware system is well positioned for soft as well as hard data-intensive computing tasks now and in the future.

Methods

Partial differential equation. A PDE is any equation with a function of multiple variables and their partial derivatives. Let u be a function with independent variables $t, x_1, \ldots, x_n$, defined as:

$$u = u(t, x_1, x_2, \ldots, x_n) \quad (5)$$

A general PDE of u has the form:

$$f\left(t, x_1, \ldots, x_n, u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial t^2}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_n}, \ldots\right) = 0 \quad (6)$$

The order of equation (6) is determined by the order of the highest partial derivative appearing in the equation.

Software package CPHS. A Python-based software package provides system-level functions to the hardware solver and controls the hardware operation. The program, Crossbar PDE Hardware Solver (CPHS), performs high-level operations such as coefficient matrix slicing and precision-extension. These system-level operations transform the process of solving a PDE problem into a set of matrix operations that are in turn computed in the memristor crossbar hardware. Specifically, CPHS provides control signals to the system's PCB to manage modules such as analogue-to-digital converters (ADCs), digital-to-analogue converters (DACs) and multiplexers to send signals to and collect data from the memristor crossbar. The results (typically partial products) collected from the hardware set-up are then used to reconstruct the final output of the system. A graphical user interface for CPHS was developed to ease the design and analysis processes (see Supplementary Fig. 1 and 2).

CPHS also supports modules based on exact and iterative floating-point methods, as well as simulations using realistic memristor device models for comparison and analysis of the system performance. CPHS enables users to readily update device parameters such as the crossbar size and the number of bits per device, as well as other parameters in the circuitry.

Memristor array fabrication and characterization. A Ta_2O_{5-x}-based memristor crossbar array was used to implement the PDE solver in hardware. Starting from a SiO_2/Si substrate, the bottom electrode (Pd, 50 nm thick) was patterned by photolithography, electron-beam evaporation and lift-off processes. The 3.5-nm-thick Ta_2O_{5-x} switching layer was deposited by RF sputtering, using low power (30 W) and a slow deposition rate (about 1.1 Å min^{-1}). The top Ta (40 nm) electrode was deposited by photolithography and d.c. sputtering (100 W), followed by electron-beam evaporation of a passivation layer (Pd 25 nm + Au 75 nm) and lift-off processes. Finally, a reactive ion etching process (SF_6/Ar) was performed to expose the bottom contacts. Characterization of the Ta_2O_{5-x}-based memristors was performed using either a Keithley semiconductor parameter analyser tool (for stand-alone cells), or a custom-built system including a test board (for crossbar array measurements and PDE solver implementation).

Test board system. A custom-built PCB test board was developed to measure the memristor crossbar array and implement the PDE solver along with the software package CPHS. The board can send and retrieve signals from up to 32 rows and 32 columns. Pulse signals needed for set/reset and read operations are generated from four independent DACs, two of which (DAC0 and DAC1) are used for rows and two (DAC2, DAC3) for columns. The signals are delivered to the selected rows or columns through switching matrices on the board. During read (computing) operation, the output node (typically a column) is connected to a sensing circuit and the output current is converted to a voltage signal and processed by an ADC. The board is controlled by a mixed code based on Python and C.

Write–verify technique. To reduce the memristor device variability, we used a write–verify technique to write and update the coefficient values in the crossbar. Specifically, each write operation is based on a sequence of write–read pulse pairs, each pair including a programming (set or reset) pulse and a subsequent read pulse (0.3 V, 100 µs) for verification purpose. By comparing the read current with a target value of the cell, the programming pulse in the next pair in the sequence is determined: that is, set (reset) for conductance increase (decrease). Each set (reset) pulse has a fixed duration (1 µs) but gradually increasing voltage levels to achieve the desired conductance value (0.75 V to 0.9 V for set, −0.85 V to −1.05 V for reset). When the conductance reaches within a pre-determined range of the target value (for example 1%), the write operation is considered complete. Supplementary Fig. 3 shows a flow chart for the write–verify process. In the experimental implementation, the initial write sequence (with the device in the high-resistance state) typically requires 100 write–verify pairs on average, while updating the coefficients later on typically requires around 10 write–verify pairs in a write sequence.

Precision-extension technique. Precision extension is a system-level technique we proposed to improve the effective precision of memristor-based hardware. In this case, a high precision can be provided through multiple devices together, each of which stores a portion of the required bitwidth. The same technique is also used to implement the required precision of the input and output analogue data to and from the crossbar array, by representing a high-precision data using multiple splits.

The approach here is to treat the data in a base-l number system, where l is the number of levels represented by a single digit. For example, operations of 12-bit numbers can be processed in a physical system based on six-bit devices: thus, 12-bit vectors $\mathbf{X}$ and $\mathbf{Y}$ can be represented as:

$$\mathbf{X} = \begin{bmatrix} (a_1, a_0)_l \\ (b_1, b_0)_l \\ (c_1, c_0)_l \end{bmatrix} = (\mathbf{X}_1, \mathbf{X}_0)_l \quad (7)$$

$$\mathbf{Y} = \begin{bmatrix} (d_1, d_0)_l \\ (e_1, e_0)_l \\ (f_1, f_0)_l \end{bmatrix} = (\mathbf{Y}_1, \mathbf{Y}_0)_l \quad (8)$$

where $\mathbf{X}_1, \mathbf{X}_0, \mathbf{Y}_1, \mathbf{Y}_0$ are six-bit vectors. A dot product between the two vectors is performed as:

$$\mathbf{X} \cdot \mathbf{Y} = (\mathbf{X}_0 \cdot \mathbf{Y}_0) + l\,(\mathbf{X}_0 \cdot \mathbf{Y}_1) + l\,(\mathbf{X}_1 \cdot \mathbf{Y}_0) + l^2\,(\mathbf{X}_1 \cdot \mathbf{Y}_1) \quad (9)$$

where l and l^2 denote single and double digit shifts. Each partial dot product is computed as at the (native) single digit level:

$$\mathbf{X}_i \cdot \mathbf{Y}_j = a_i d_j + b_i e_j + c_i f_j \quad (10)$$

Equation (10) can be directly executed in a memristor crossbar by encoding $\{a_i, b_i, c_i\}$ as the input voltages applied to different rows and $\{d_j, e_j, f_j\}$ as the memristors' conductance values along a column. The results of the partial products are summed together according to equation (9) to obtain the full dot product result. Other arithmetic operations with the extended precision can be performed similarly by splitting the high-precision operations into partial operations. Similar techniques to improve the precision of memristor-based hardware systems have been proposed for neural network type of applications[24–28].

To properly implement the precision-extension technique, the partial products need to be quantized before the digit shift operations, otherwise noise (error) in the analogue output due to device variation becomes amplified during the digit shift and reduces the precision of the final output. Fortunately, the quantization operation can be readily implemented in the existing circuitry through the ADC circuitry that is already used in the hardware to quantize analogue outputs to digital values. In a typical ADC operation, any analogue value within quantization thresholds is represented by the same quantized value, thus preventing the noise in the analogue signal from being amplified during the shift operation (see Supplementary Fig. 6), and enable the shift operations to perform properly.

Specifically, the required number of ADC bits is determined by the sum of the input bits, the stored coefficient (weight) bits, and the number of non-zero inputs per column, as defined in[24]:

$$b_{\mathrm{ADC}} = b_{\mathrm{i}} + b_{\mathrm{d}} + \log_2 \omega \tag{11}$$

where b_{i} is the number of bits of the input, b_{d} is the equivalent number of bits of each memristor device, and ω is the number of non-zero inputs per column, although this bitwidth requirement may be reduced slightly through architecture optimizations[24]. By designing ADCs with the right number of bits based on equation (11) (instead of pursuing as many bits as possible), the ADC area and power consumption overhead can be minimized while also providing the desirable quantization effect to minimize error propagation during the shift operations. Indeed, our simulations and experiments verify that combining the ADC quantization effect with the precision-extension technique can reliably lead to high-precision operation (for example, 64 bits) of the memristor-based hardware, even though the native device precision is much lower, for example 4-bit. Additionally, a normalization (scaling) step can be introduced to improve the dynamic range of the fixed-point solver. In this approach, a global exponent field is used for all values, and the radix point location for all elements in the input vector can be shifted together to improve the dynamic range if needed. The normalization can be performed at either the initialization stage or during iterations, depending on the precision requirement of the problem.

Finite-difference method. The finite-difference method (FDM) is a numerical technique to solve partial (or ordinary) differential equations by discretizing the partial derivatives using difference equations. Given a set of regularly distributed grid points of fixed separation, the FDM approximates the partial derivatives at each point using its neighbouring values based on Taylor's theorem[42]. For instance, a first derivative can be approximated in a 1D system using the nearest neighbour points as:

$$\text{Forward}: \frac{\mathrm{d}u}{\mathrm{d}x} \approx \frac{u_{i+1}-u_i}{h} \tag{12}$$

$$\text{Backward}: \frac{\mathrm{d}u}{\mathrm{d}x} \approx \frac{u_i-u_{i-1}}{h} \tag{13}$$

$$\text{Central}: \frac{\mathrm{d}u}{\mathrm{d}x} \approx \frac{u_{i+1}-u_{i-1}}{2h} \tag{14}$$

where h is the spacing between two grid points, i is the index of the desired grid point, and $i-1$ and $i+1$ are the indices of the neighbouring points. The forward, backward and central approximations depend on the direction equation used. Higher-accuracy approximations can be achieved by using more neighbouring points. The technique is not limited to fixed mesh spacing, although we use fixed meshing spacing here for simplicity.

In this study, we used a central approximation to represent the second-order derivatives of the elliptic and hyperbolic PDEs that are mapped to the hardware system. For example, a second-order partial derivative with respect to variable x for a 2D system in Cartesian coordinates is:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2} \tag{15}$$

where i is the x-axis grid index, j is the y-axis grid index and h is the distance between two neighbouring grid points along the x- or y-axis. Hence, partial derivatives at a point can be expressed by values at its neighbouring points. The pre-factors (coefficients) in the expressions are summarized as coefficient tables for a given order and accuracy during the discretization process.

For instance, the gradient relation in a 2D space can be described using central difference as:

$$\nabla u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i+1,j}+u_{i,j+1}-4u_{i,j}+u_{i-1,j}+u_{i,j-1}}{h^2} \tag{16}$$

where h is the distance between two neighbouring grid points along the x-axis or the y-axis. Here, each point is related to its four neighbouring points using coefficients of 1,1,–4,1,1. Such relations can be graphically represented using stencils, where the size of a stencil is defined by the number of points each point is related to (see Supplementary Fig. 11). Hence, equation (16) is described using a five-point stencil.

Finally, the discretized equation is mapped into a coefficient matrix, where each row in the coefficient matrix describes the relation between a point and the other points in the grid. The dimension of the (square) coefficient matrix, that is, the number of elements per row, is equal to the total number of grid points in the mesh and can thus be very large. However, the number of non-zero elements per row equals the stencil size (for non-edge grid points), resulting in highly sparse coefficient matrices.

Jacobi method. The Jacobi iterative method was used to solve elliptic PDE systems. The Jacobi method is a numerical technique used to solve diagonally dominant linear systems, $A \cdot \mathbf{X} = \mathbf{B}$, where:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \text{ and } \mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{17}$$

The iterative solution is obtained from:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j\neq i} a_{ij} x_j^{(k)}}{a_{ii}}, i = 1, 2, \ldots, n \tag{18}$$

Equation (17) can be written in a matrix form as:

$$\mathbf{X}^{(k+1)} = D^{-1}(\mathbf{B} - M \cdot \mathbf{X}^{(k)}) \tag{19}$$

where D is a matrix with only the diagonal elements of A, and M contains the remaining elements of A. In the case where the diagonal elements are equal, which is common, equation (19) is simplified as:

$$\mathbf{X}^{(k+1)} = \frac{1}{d}(\mathbf{B} - M \cdot \mathbf{X}^{(k)}) \tag{20}$$

where d is the common diagonal element.

We note that any scientific or engineering system that includes transport phenomena or reaction chemistry (such as fluid dynamics or radiation transport) that is modelled for chemical engineering, combustion, fluid mechanics, solid state physics and nuclear engineering, in addition to plasma physics, is of a diagonally dominant matrix. Thus, the vast majority of the PDE problems are diagonally dominant and can be solved by the proposed method in this work. It should be noted, however, that diagonally dominant coefficient matrices do not have to be structured. As shown in Supplementary Fig. 8a,b, an unstructured matrix can be divided into slices that are diagonally dominant and mapped into the crossbar system.

Poisson's equation mapping. The Poisson's equation described in equation (2) is mapped to the crossbar hardware system by first approximating it using central FDM and a common 2D five-point stencil as

$$\frac{u_{i+1,j}+u_{i,j+1}-4u_{i,j}+u_{i-1,j}+u_{i,j-1}}{h^2} = -2\sin(x_i)\cos(y_i) \tag{21}$$

where i is the x-axis grid index, j is the y-axis grid index and h is the distance between two neighbouring grid points along the x-axis or y-axis. In an example where the problem's domain is discretized into a 3×3 grid (excluding boundary points), equation (21) is transformed into a matrix form as:

$$\begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \cdot \begin{pmatrix} u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{2,0} \\ u_{2,1} \\ u_{2,2} \end{pmatrix} = \begin{bmatrix} h^2 f(x_0,y_0) - b_{-1,0} - b_{0,-1} \\ h^2 f(x_0,y_1) - b_{-1,1} \\ h^2 f(x_0,y_2) - b_{-1,2} - b_{0,3} \\ h^2 f(x_1,y_0) - b_{1,-1} \\ h^2 f(x_1,y_1) \\ h^2 f(x_1,y_2) - b_{1,3} \\ h^2 f(x_2,y_0) - b_{2,-1} - b_{3,0} \\ h^2 f(x_2,y_1) - b_{3,1} \\ h^2 f(x_2,y_2) - b_{2,3} - b_{3,2} \end{bmatrix} \tag{22}$$

$$\begin{bmatrix} u_{0,0}\\ u_{0,1}\\ u_{0,2}\\ u_{1,0}\\ u_{1,1}\\ u_{1,2}\\ u_{2,1}\\ u_{2,2}\\ u_{1,2} \end{bmatrix}^{(k+1)} = \mathbf{C} + \begin{bmatrix} 0 & 1/4 & 0 & 1/4 & 0 & 0 & 0 & 0 & 0\\ 1/4 & 0 & 1/4 & 0 & 1/4 & 0 & 0 & 0 & 0\\ 0 & 1/4 & 0 & 0 & 0 & 1/4 & 0 & 0 & 0\\ 1/4 & 0 & 0 & 0 & 1/4 & 0 & 1/4 & 0 & 0\\ 0 & 1/4 & 0 & 1/4 & 0 & 1/4 & 0 & 1/4 & 0\\ 0 & 0 & 1/4 & 0 & 1/4 & 0 & 0 & 0 & 1/4\\ 0 & 0 & 0 & 1/4 & 0 & 0 & 0 & 1/4 & 0\\ 0 & 0 & 0 & 0 & 1/4 & 0 & 1/4 & 0 & 1/4\\ 0 & 0 & 0 & 0 & 0 & 1/4 & 0 & 1/4 & 0 \end{bmatrix} \cdot \begin{bmatrix} u_{0,0}\\ u_{0,1}\\ u_{0,2}\\ u_{1,0}\\ u_{1,1}\\ u_{1,2}\\ u_{2,1}\\ u_{2,2}\\ u_{1,2} \end{bmatrix}^{(k)} \quad (23)$$

where $f(x_i, y_j) = \sin(x_i)\cos(y_j)$, and $b_{i,j}$ are the boundary values of the system.

Similar matrices can be obtained for larger grids. It should be noted that the sparsity of the coefficient matrix is a function of the mesh size, with larger meshes leading to more sparse matrices, since the matrix dimension is determined by the total number of grid points, while the number of non-zero elements equals the fixed (small) stencil size. A typical stencil size is 5 for second-order 2D PDEs, while the matrix dimension can be very large (for example, with 900 elements in a row for a small 30×30 grid), leading to very sparse matrices. Equation (22) is in the form of $A\cdot\mathbf{X}=\mathbf{B}$, and can be solved using the Jacobi method as:

where the constant vector $\mathbf{C}$ equals $-\mathbf{B}/4$ in this example. Afterwards, the system is mapped to the memristor crossbar hardware and solved as described in the main text.

Two-dimensional wave PDE mapping. To map the 2D wave system described in equation (3) to the hardware set-up, we first discretized it following FDM as:

$$\frac{u_{i,j}^{(t+1)} - 2u_{i,j}^{(t)} + u_{i,j}^{(t-1)}}{(\Delta t)^2} = \theta^2\left(\frac{u_{i+1,j}^{(t)} + u_{i,j+1}^{(t)} - 4u_{i,j}^{(t)} + u_{i-1,j}^{(t)} + u_{i,j-1}^{(t)}}{h^2}\right) - \zeta\left(\frac{u_{i,j}^{(t)} - u_{i,j}^{(t-1)}}{\Delta t}\right) \quad (24)$$

where u represents the wave height, i is the x-axis grid index, j is the y-axis grid index, h is the distance between two neighbouring grid points along the x-axis or y-axis, t is the time index, Δt is the numerical time step, θ is the wave speed and ζ is the decay (damping) constant. Here, central FDM is used for the second-order partial derivatives, and backward FDM is used for the first-order partial derivative. Equation (24) can be re-written in a five-point stencil format as:

$$u_{i,j}^{(t+1)} = (2-\zeta\Delta t)u_{i,j}^{(t)} + (\zeta\Delta t-1)u_{i,j}^{(t-1)} + \left(\frac{\theta\Delta t}{h}\right)^2 (u_{i+1,j}^{(t)} + u_{i,j+1}^{(t)} - 4u_{i,j}^{(t)} + u_{i-1,j}^{(t)} + u_{i,j-1}^{(t)}) \quad (25)$$

and mapped to a matrix form as (shown for a 3×3 grid for illustration purposes):

$$\begin{bmatrix} u_{0,0}\\ u_{0,1}\\ u_{0,2}\\ u_{1,0}\\ u_{1,1}\\ u_{1,2}\\ u_{2,1}\\ u_{2,2}\\ u_{1,2} \end{bmatrix}^{(t+1)} = \alpha_1 \begin{bmatrix} u_{0,0}\\ u_{0,1}\\ u_{0,2}\\ u_{1,0}\\ u_{1,1}\\ u_{1,2}\\ u_{2,1}\\ u_{2,2}\\ u_{1,2} \end{bmatrix}^{(t)} + \alpha_2 \begin{bmatrix} u_{0,0}\\ u_{0,1}\\ u_{0,2}\\ u_{1,0}\\ u_{1,1}\\ u_{1,2}\\ u_{2,1}\\ u_{2,2}\\ u_{1,2} \end{bmatrix}^{(t-1)} + \alpha_3 \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0\\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0\\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1\\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0\\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \cdot \begin{bmatrix} u_{0,0}\\ u_{0,1}\\ u_{0,2}\\ u_{1,0}\\ u_{1,1}\\ u_{1,2}\\ u_{2,1}\\ u_{2,2}\\ u_{1,2} \end{bmatrix}^{(t)} \quad (26)$$

where the constants $\alpha_1 = 2-\zeta\Delta t$, $\alpha_2 = \zeta\Delta t-1$ and $\alpha_3 = (\theta\Delta t/h)^2$. Similar to the Poisson's equation case, the maximum number of non-zero elements in each row equals the stencil size (5), while the row length equals the total number of grid points. This results in highly sparse coefficient matrices.

Hybrid plasma equipment model simulator. The HPEM[44] is a 2D fluid-kinetic hydrodynamics simulation package used for the investigation of low-pressure (1 mTorr to 1 Torr) and low-temperature (T_e < tens of eV) plasma processing reactors. The HPEM has a hierarchical structure in which different modules addressing different physical phenomena are sequentially executed in an iterative manner (see Supplementary Fig. 12). In this study, the major modules used are the electromagnetics module (EMM), the electron energy transport module (EETM) and the FKPM. The EMM solves the inductively coupled electromagnetic fields based on the applied azimuthal currents in the ICP coils using a frequency domain solution of Maxwell's equations. The EETM solves the spatially dependent electron energy distributions using an electron Monte Carlo simulation, which then provides electron impact source functions for inelastic collisions and electron transport coefficients. In the FKPM, owing to the tight coupling of electrostatic fields to the densities of charged particles, the Poisson's equation is solved self-consistently with the continuity, momentum and energy equations for all heavy particles. An iteration in HPEM consists a complete cycle through these modules which sequentially receive and provide data between them. In the FKPM, given the charge distribution in the plasma region and the boundary conditions (d.c. bias and applied voltage waveform on the electrodes and charges accumulated on the surface of the dielectrics), the 2D Poisson's equations are solved using two approaches, the DSLUCS[45] and the memristor crossbar-based PDE solver, for the present electrostatic potentials.

Integrating the memristor PDE solver with HPEM. The HPEM uses a finite-volume method to generate coefficient matrices for Poisson's equation in the FKPM module. The matrices are then handed to the memristor PDE solver. The matrices are sliced into 32×32 patches by the solver's software, with all diagonal elements removed to prepare it for the numerical Jacobi method processes. Precision-extension techniques are used to implement the matrix operations at 32 bits. Afterwards, the results are supplied to the HPEM code for the next operation. To represent positive and negative coefficients, two matrices, representing respectively the positive and negative numbers, were used, with precision extension applied to each of them separately.

DSLUCS subroutine. DSLUCS[45] is a Fortran language subroutine used to solve linear $A\cdot\mathbf{X}=\mathbf{B}$ systems using the biconjugate gradient squared method with Incomplete LU decomposition preconditioning technique. The DSLUCS is used by the HPEM software as the default $A\cdot\mathbf{X}=\mathbf{B}$ numerical solver.

Mean absolute error. We calculated the error between the computed numerical result and the exact solution using the MAE, which is defined as:

$$\mathrm{MAE} = \frac{\sum_{j=0}^{n-1} |x_j^{\mathrm{E}} - x_j^{\mathrm{N}}|}{n} \quad (27)$$

where x^{E} is the exact solution of the problem, x^{N} is the numerically computed solution either using the memristor PDE solver or a floating-point solver, and n is the number of grid points.

Data availability. The data that support the plots within this paper and other findings of this study are available from the corresponding author upon reasonable request.

Received: 26 January 2018; Accepted: 14 June 2018;
Published online: 13 July 2018

References

1. Simon, H., Zacharia, T. & Stevens, R. *Modeling and Simulation at the Exascale for Energy and the Environment* (Department of Energy Technical Report, 2007).
2. Palmer, T. Build imprecise supercomputers. *Nature* **526**, 32–33 (2015).
3. Aage, N., Andreassen, E., Lazarov, B. S. & Sigmund, O. Giga-voxel computational morphogenesis for structural design. *Nature* **550**, 84–86 (2017).
4. Altrock, P. M., Liu, L. L. & Michor, F. The mathematics of cancer: integrating quantitative models. *Nat. Rev. Cancer* **15**, 730–745 (2015).
5. Bauer, P., Thorpe, A. & Brunet, G. The quiet revolution of numerical weather prediction. *Nature* **525**, 47–55 (2015).
6. Achdou, Y., Buera, F. J., Lasry, J.-M., Lions, P.-L. & Moll, B. Partial differential equation models in macroeconomics. *Philos. Trans. R. Soc. A* **372**, 20130397 (2014).
7. Dongarra, J. J. et al. The International Exascale Software Project roadmap. *Int. J. High. Perform. Comput.* **25**, 3–60 (2011).
8. Nair, R. Evolution of memory architecture. *Proc. IEEE* **103**, 1331–1345 (2015).
9. Kogge, P. et al. *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems* (DARPA, 2008).
10. Nair, R. et al. Active memory cube: a processing-in-memory architecture for exascale systems. *IBM J. Res Dev.* **59**, 1–7 (2015).
11. Jeddeloh, J. & Keeth, B. Hybrid memory cube new DRAM architecture increases density and performance. In *Proc. IEEE Symposium on VLSI Technology (VLSIT)* 87–88 (IEEE, 2012).
12. Strukov, D. B., Snider, G. S., Stewart, D. R. & Williams, R. S. The missing memristor found. *Nature* **453**, 80–83 (2008).

13. Yang, J. J., Strukov, D. B. & Stewart, D. R. Memristive devices for computing. *Nat. Nanotech.* **8**, 13–24 (2013).
14. Wong, H.-S. P. et al. Metal–oxide RRAM. *Proc. IEEE* **100**, 1951–1970 (2012).
15. Prezioso, M. et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015).
16. Sheridan, P. et al. Sparse coding with memristor networks. *Nat. Nanotech.* **12**, 784–789 (2017).
17. Zidan, M. A., Strachan, J. P. & Lu, W. D. The future of electronics based on memristive systems. *Nat. Electron.* **1**, 22–29 (2018).
18. Ielmini, D. Modeling the universal set/reset characteristics of bipolar RRAM by field- and temperature-driven filament growth. *IEEE Trans. Electron Devices* **58**, 4309–4317 (2011).
19. Kim, K.-H. et al. A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications. *Nano Lett.* **12**, 389–395 (2011).
20. Waser, R. & Aono, M. Nanoionics-based resistive switching memories. *Nat. Mater.* **6**, 833–840 (2007).
21. Li, C. et al. Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* **1**, 52–59 (2018).
22. Feinberg, B., Vengalam, U., Whitehair, N., Wang, S. & Ipek, E. Enabling scientific computing on memristive accelerators. In *ACM/IEEE Int. Symp. on Computer Architecture* (ACM/IEEE, 2018).
23. Hu, M. et al. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *ACM/EDAC/IEEE Design Automation Conf.* 1–6 (ACM/EDAC/IEEE, 2016).
24. Shafiee, A. et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ACM/IEEE Ann. Int. Symp. on Computer Architecture* 14–26 (ACM/IEEE, 2016).
25. Chi, P. et al. PRIME: a novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *ACM/IEEE Ann. Int. Symp. on Computer Architecture* 27–39 (ACM/IEEE, 2016).
26. Zidan, M. A. et al. Field-programmable crossbar array (FPCA) for reconfigurable computing. *IEEE Trans. Multi-Scale Comput. Syst.* https://doi.org/10.1109/TMSCS.2017.2721160 (2017).
27. Song, L., Qian, X., Li, H. & Chen, Y. PipeLayer: a pipelined ReRAM-based accelerator for deep learning. *IEEE Int. Symp. on High Performance Computer Architecture* 541–552 (IEEE, 2017).
28. Bojnordi, M. N. & Ipek, E. Memristive Boltzmann machine: a hardware accelerator for combinatorial optimization and deep learning. *IEEE Int. Symp. on High Performance Computer Architecture* 1–13 (IEEE, 2016).
29. Zidan, M. A., Chen, A., Indiveri, G. & Lu, W. D. Memristive computing devices and applications. *J. Electroceram.* **39**, 4–20 (2017).
30. Neftci, E., Pedroni, B. U., Joshi, S., Al-Shedivat, M. & Cauwenberghs, G. Stochastic synapses enable efficient brain-inspired learning machines. *Front. Neurosci.* **10**, 241 (2016).
31. Yu, S. et al. Scaling-up resistive synaptic arrays for neuro-inspired architecture: challenges and prospect. In *IEEE Int. Electron Devices Meeting* 17.3.1–17.3.4 (IEEE, 2015).
32. Alibart, F., Gao, L., Hoskins, B. D. & Strukov, D. B. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* **23**, 075201 (2012).
33. Richter, I. et al. Memristive accelerator for extreme scale linear solvers. In *Government Microcircuit Applications & Critical Technology Conf. (GOMACTech)* (2015).
34. Gallo, M. L. et al. Mixed-precision in-memory computing. *Nat. Electron.* **1**, 246–253 (2018).
35. Jeong, Y., Zidan, M. A. & Lu, W. D. Parasitic effect analysis in memristor array-based neuromorphic systems. *IEEE Trans. Nanotechnol.* **17**, 184–193 (2018).
36. Choi, S., Shin, J. H., Lee, J., Sheridan, P. & Lu, W. D. Experimental demonstration of feature extraction and dimensionality reduction using memristor networks. *Nano Lett.* **17**, 3113–3118 (2017).
37. Guan, X., Yu, S. & Wong, H.-S. P. On the switching parameter variation of metal-oxide RRAM—Part I: Physical modeling and simulation methodology. *IEEE Trans. Electron Devices* **59**, 1172–1182 (2012).
38. Jo, S. H., Kim, K.-H. & Lu, W. Programmable resistance switching in nanoscale two-terminal devices. *Nano Lett.* **9**, 496–500 (2008).
39. Alibart, F., Gao, L., Hoskins, B. D. & Strukov, D. B. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* **23**, 075201 (2012).
40. Kim, K. M. et al. Voltage divider effect for the improvement of variability and endurance of TaO_x memristor. *Sci. Rep.* **6**, 20085 (2016).
41. Gilbarg, D. & Trudinger, N. S. *Elliptic Partial Differential Equations of Second Order* (Springer, Berlin, 2015).
42. Ames, W. F. *Numerical Methods for Partial Differential Equations* (Academic, New York, 2014).
43. Nishidate, Y. & Nikishkov, G. P. Fast water animation using the wave equation with damping. *Int. Conf. on Computational Science* 232–239 (Springer, 2005).
44. Kushner, M. J. Hybrid modelling of low temperature plasmas for fundamental investigations and equipment design. *J. Phys. D* **42**, 194013 (2009).
45. *SLAP Sparse Matrix Library* (accessed 6 Jan 2017); http://www.netlib.org/
46. Eymard, R., Gallouët, T. & Herbin, R. in *Handbook of Numerical Analysis* (eds Ciarlet, P. G. & Lions, J. L.) 713–1018 (Elsevier, 2000).

Acknowledgements
We acknowledge inspiring discussions with Z. Zhang, J. Moon and T. Chen. This work was support by the Defense Advanced Research Projects Agency (DARPA) through award HR0011-17-2-0018 and by the National Science Foundation (NSF) through grant CCF-1617315.

Author Contributions
M.A.Z. and W.D.L. conceived the project and constructed the research frame. M.A.Z., Y.J., J.L. and B.C. prepared the memristor arrays and built the hardware and software package. M.A.Z. and Y.J. performed the hardware measurements. M.A.Z, Y.J., S.H., M.J.K. and W.D.L. analysed the experimental data and simulation results. W.D.L. directed the project. All authors discussed the results and implications and commented on the manuscript at all stages.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary information is available for this paper at https://doi.org/10.1038/s41928-018-0100-6.

Reprints and permissions information is available at www.nature.com/reprints.

Correspondence and requests for materials should be addressed to W.D.L.

Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.