Eliminating Iterations of Iterative Methods: Solving Large-Scale Sparse Linear System in $O(1)$ with RRAM-based In-Memory Accelerator

Tao Song, Xiaoming Chen*, Yinhe Han*

Institute of Computing Technology, Chinese Academy of Sciences; University of Chinese Academy of Sciences

*Corresponding authors: {chenxiaoming, yinhes}@ict.ac.cn

ABSTRACT

The sparse linear solver is an important component in lots of scientific computing applications. For large-scale sparse linear systems, general-purpose processors such as CPUs and GPUs are facing challenges of high time complexity and massive data movements between processors and main memories. This work utilizes the ability of in-situ analog computing of RRAMs and builds an RRAM-based accelerator for iterative linear solvers. We first propose a basic principle of mapping iterative solvers onto RRAM-based crossbar arrays. The proposed principle eliminates not only the iterations but also the convergence condition. Based on the principle, we propose a scalable architecture that can solve large-scale sparse matrices in $O(1)$ time complexity. Compared with a massively parallel iterative solver on GPU, our accelerator shows 100× higher performance and 1000× energy reduction. If the solution obtained by our accelerator is used as the seed for a further refinement on GPU, about 35% of the solving time and energy consumption can be saved compared with a pure GPU solving process.

CCS CONCEPTS

• **Hardware** → **Analog and mixed-signal circuits; Emerging architectures.**

KEYWORDS

Resistive random-access memory (RRAM); sparse matrix; iterative method; in-memory computing

ACM Reference Format:

T. Song, X. Chen, Y. Han. 2021. Eliminating Iterations of Iterative Methods: Solving Large-Scale Sparse Linear System in $O(1)$ with RRAM-based In-Memory Accelerator. In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21), June 22–25, 2021, Virtual Event, USA*. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3453688.3461510>

1 INTRODUCTION

Solving large-scale sparse linear systems ($Ax=b$) is the core component of numerous practical applications, especially in scientific computing. Sparse linear systems are usually solved by direct methods [5] or iterative methods [12] on general-purpose processors

such as CPUs or GPUs. The time complexity is quite high, typically ranging from $O(N^2)$ to $O(N^3)$. As the problem scale continuously grows, conventional computers are suffering from not only the increasing computational time, but also the increasing data movements between processors and memories, known as the memory wall bottleneck [10], due to the separated processors and memories. Moreover, for sparse matrices, the memory access patterns are quite irregular, leading to a high cache miss rate on CPUs or GPUs, which further aggravates the memory wall bottleneck.

The “computing-in-memory” paradigm is one of the promising solutions to alleviate the memory wall bottleneck. Recently, there emerge some nonvolatile devices that support in-memory computing. Among them, RRAMs [20] have the ability of analog computing so they may have potentials in numerical computing. An RRAM is a two-terminal device which can store its resistance when powered off and change its resistance by applying proper current stimuli. An RRAM-based crossbar array can complete an analog matrix-vector multiplication (MVM) in $O(1)$ time complexity, where the matrix values are represented by the conductance of the RRAMs and the vector is the input, as illustrated in Fig. 1. This feature has widely been studied for neural network acceleration (e.g., [2, 3, 13, 19]), which shows outstanding performance and energy efficiency compared with conventional digital processing.

Recently, there are a few studies utilizing RRAMs to solve linear systems. Refs. [6, 7, 11, 21] proposed accelerating linear iterative solvers by utilizing RRAM-based crossbar arrays, in which the hardware computes an MVM in each iteration. The intermediate results of each iteration, which are analog signals, need to be stored. Digital storage is adopted by these approaches so digital-to-analog (D2A) and analog-to-digital (A2D) conversions are needed in each iteration, resulting in high latency and energy cost. Differently, Refs. [15, 16] proposed an RRAM-based accelerator that can solve a linear system in one step like a direct method, and thus, eliminates intermediate D2A and A2D conversions. The principle of the approach is different from the RRAM-based acceleration of MVMs. The solution is obtained in the direct form of $x = -A^{-1}b$. However, it has two major drawbacks. First, it cannot handle large-scale sparse linear systems which are indispensable in practical applications. Second, the operational amplifiers (OpAmps) in the design work in the open-loop state which may lead to unstable situations.

In this paper, we investigate how to use RRAMs to build energy-efficient accelerators to solve large-scale sparse linear systems. We provide detailed principle and architecture design for solving sparse linear systems in $O(1)$ time complexity. The contributions of the paper are summarized as follows.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8393-6/21/06...\$15.00

<https://doi.org/10.1145/3453688.3461510>

Figure 1: MVM in $O(1)$ by an RRAM-based crossbar array.

- We propose a basic principle of mapping iterative solvers onto RRAM-based crossbar arrays. Our principle not only breaks the convention that an iterative solver needs a number of iterations, but also eliminates the convergence condition of iterative methods. Although we map a mathematical iterative method onto hardware, the solution is produced in $O(1)$ time complexity without any iterations.
- Based on the principle, we propose a scalable architecture that can solve a large-scale sparse linear system in one step. A mapping algorithm is introduced to map sparse linear systems to the architecture.
- Compared with the existing accelerators for iterative methods [6, 7, 11, 21], our approach is a significant advance due to the elimination of iterations and convergence condition. Compared with [15, 16] which also solve linear systems in one step, our architecture has a superiority of supporting large-scale sparse linear systems.

2 PRINCIPLE OF $O(1)$ IN-MEMORY ACCELERATOR FOR ITERATIVE METHODS

2.1 General Form of Classical Iterative Methods

A classical iterative method for $Ax = b$ generates a sequence of approximate solutions $\{x^{(k)}\}$ that converges to the precise solution $A^{-1}b$, by way of splitting A into $A = M - N$ and converting the original system to the fixed-point problem $Mx = Nx + b$, giving rise to the iterative process [12]

$$Mx^{(k+1)} = Nx^{(k)} + b. \quad (1)$$

Equation (1) can be converted to

$$x^{(k+1)} = Bx^{(k)} + g \quad (2)$$

where $B = M^{-1}N$ and $g = M^{-1}b$. Both B and g are fixed during iterations. The convergence condition is that the spectral radius of B is less than 1 [12].

Take the well-known Jacobi and Gauss-Seidel iterative methods as examples. In the former, M is the diagonal of A (denoted by D) and $N = D - A$. In the latter, $M = D + L$ and $N = -U$, where U and L are the strictly upper and lower triangular parts of A , respectively. Then $B = -(D+L)^{-1}U$ and $g = (D+L)^{-1}b$. As $D+L$ is a triangular matrix, computing its inverse is trivial.

The general form of classical iterative methods, (2), also defines a unified interface for mapping classical iterative methods onto RRAM-based crossbar arrays. As long as an iterative method can be expressed as the form of (2) (regardless of whether it converges or not), it can be mapped onto RRAM-based crossbar arrays. Typical well-known methods include the Jacobi method, the Gauss-Seidel method, and the successive over relaxation method.

Figure 2: Principle illustration of RRAM-based $O(1)$ accelerator for iterative methods.

2.2 Principle Overview

Theoretically, an $N \times N$ matrix can be mapped onto an $N \times N$ RRAM-based crossbar array. After tuning the conductance of the RRAMs to the target values, it is effortless to accomplish an MVM in $O(1)$ by applying a voltage vector V_{WL} to the word lines (WLs) and obtaining results from the bit lines (BLs) [8], according to the Kirchhoff's law (see Fig. 1), i.e.,

$$I_{BL} = G^T V_{WL} \quad (3)$$

where G is the conductance matrix of the crossbar array.

Fig. 2 illustrates the proposed principle of RRAM-based $O(1)$ accelerator for iterative methods, which can also be regarded as an accelerator for small-scale linear systems. It is composed of two major parts: two RRAM-based crossbar arrays and the peripheral analog adders. Since the conductance of an RRAM device can only represent a positive value, we need two crossbar arrays to represent positive and negative values, respectively. The two crossbar arrays are connected through the left-side word lines (LWLs) and the right-side word lines (RWLs) with analog inverters inserted between them. According to (3), the two crossbar arrays compute the currents of the left-side bit lines (LBLs) and the right-side bit lines (RBLs), respectively. The corresponding LBLs and RBLs are connected to the common BLs, which are linked to the peripheral circuits. The I-V converters and analog adders convert the BL currents to voltages and also add vector g to the converted voltages. The voltage values of V_g 's in the analog adders are provided by digital-to-analog converters (DACs). We directly connect the outputs to the inputs. This is the key for our principle to eliminate both the iterations and the convergence condition of iterative methods, which will be further discussed in Section 2.4. The outputs of the analog adders correspond to the solution, and may be read out in virtue of analog-to-digital converters (ADCs).

2.3 Mapping Methodology

2.3.1 Mapping Iterative Matrix B . The iterative matrix B is mapped to the conductance of the two crossbar arrays shown in Fig. 2. Matrix B is split into $B^+ + B^-$ where B^+ and B^- respectively contain the positive and negative elements of B . B^+ and B^- are mapped to the two crossbars, respectively.

We should chiefly guarantee that the entire circuit is converged in the DC mode according to the Kirchhoff's law, and the solution shows sufficient precision. Therefore, the mapping should be linear,

meaning that the conductance of an RRAM should be proportional to the absolute value of the corresponding matrix element. However, the resistance of RRAM devices has an upper limit and a lower limit, namely, the high-resistance state (HRS) and the low-resistance state (LRS). In practice, it is impossible to precisely map matrix $\mathbf{B}$, not only because of the limited range of the RRAM resistance, but also due to device variations, programming errors, etc.

For zero elements in $\mathbf{B}$, the mapping is straightforward — we just use the HRS of RRAMs to represent them approximately. For nonzero elements, the value range of the elements may not be in the coverage of the RRAM conductance range. To make them matched, a scaling factor α is multiplied to the elements of $\mathbf{B}$. Since big elements typically have large impacts on the accuracy, we select α such that the biggest absolute value of the elements does not exceed $\frac{1}{R_{LRS}}$, which is the LRS conductance. In other words, α is selected to satisfy

$$\alpha \cdot \max_{1 \leq i, j \leq N} |B_{ij}| \leq \frac{1}{R_{LRS}}. \quad (4)$$

For example, if $R_{LRS} = 10K\Omega$ and the maximum absolute value is 0.6, we can select $\alpha = 10^{-4}$. After α is decided, any $|B_{ij}|$ is mapped to a resistance value $\frac{1}{\alpha|B_{ij}|}$. If $\frac{1}{\alpha|B_{ij}|} > R_{HRS}$, then we just map it to R_{HRS} .

The scaling factor and the resistance in the analog adders actually affect each other. An analog adder is mainly composed of two OpAmps that both operate in the negative-feedback state. The output voltage of OpAmp ① in the j th column is

$$V_{OpAmp1,j} = -R_{f1} I_{BL,j} \quad (5)$$

where $I_{BL,j}$ is the current of the j th BL. The output voltage of OpAmp ② in the j th column is

$$\begin{aligned} V_{OpAmp2,j} &= \frac{R_{f2}}{R_1} (V_{g,j} - V_{OpAmp1,j}) \\ &= \frac{R_{f1}R_{f2}}{R_1} I_{BL,j} + \frac{R_{f2}}{R_1} V_{g,j}. \end{aligned} \quad (6)$$

The first and second terms on the right-hand side of (6) correspond to $\mathbf{B}\mathbf{x}^{(k)}$ and $\mathbf{g}$ in (2), respectively. Recall that a scaling factor α is used when mapping the iterative matrix $\mathbf{B}$, which equivalently scales $I_{BL,j}$ by α . To make the two terms on the right-hand side of (6) consistently scaled, their coefficients should be identical and equal to 1, i.e.,

$$\frac{\alpha R_{f1} R_{f2}}{R_1} = \frac{R_{f2}}{R_1} = 1. \quad (7)$$

This implies that $\alpha = \frac{1}{R_{f1}}$ and $R_1 = R_{f2}$.

2.3.2 Mapping Vector $\mathbf{g}$. Vector $\mathbf{g}$ is mapped to the voltage sources (i.e., V_g 's) in the analog adders. The voltages are provided by DACs. In practice, the values of $\mathbf{g}$ may exceed the output voltage range of DACs. So a scaling factor β is also needed for vector $\mathbf{g}$. Different from scaling matrix $\mathbf{B}$, scaling vector $\mathbf{g}$ has no impact on the circuit. Scaling $\mathbf{g}$ is equivalent to scaling the RHS vector $\mathbf{b}$ on the software side. After the solution is obtained by our accelerator, we should scale the solution by $\frac{1}{\beta}$ to obtain the correct solution.

2.4 Proof and Analysis

In Section 2.1 we have presented the iterative formula (2) and how to complete an iterative method through repeating it. Superficially we should use a clock to actualize iterations, which involves a difficult problem: analog storage. Digital storage brings high cost since ADCs and DACs are needed.

To avoid the aforementioned overhead, in our methodology, the outputs are directly connected to the inputs. Here we prove that the DC response of the circuit is the solution of the given linear system. We ignore those RRAM devices which are programmed into HRS to approximate zeros. According the analysis in Section 2.3, the impacts of the scaling factors have been removed so we also ignore the scaling factors. According to the Kirchhoff's law, (5) can be converted to

$$V_{OpAmp1,j} = - \sum_{i=1}^N (g_{ij}^+ V_{LWL,i} + g_{ij}^- V_{RWL,i}) \quad (8)$$

where g_{ij}^+ and g_{ij}^- represent the conductance at the intersections of row i and column j of the two crossbars, respectively (g_{ij}^+ and g_{ij}^- are both positive), and $V_{LWL,i}$ and $V_{RWL,i}$ are respectively the voltages of the i th LWL and i th RWL. Since $-V_{RWL,i} = V_{LWL,i} = x_i$ and one of g_{ij}^+ and g_{ij}^- equals to $|B_{ji}|$ and the other is 0, (8) can be simplified to

$$V_{OpAmp1,j} = - \sum_{i=1}^N B_{ji} x_i \quad (9)$$

which is another form of (3). Equation (6) becomes

$$V_{OpAmp2,j} = -V_{OpAmp1,j} + V_{g,j} = \sum_{i=1}^N B_{ji} x_i + g_j. \quad (10)$$

Since $V_{OpAmp2,j} = x_j$ (the outputs are directly connected to the inputs), (10) becomes

$$x_j = \sum_{i=1}^N B_{ji} x_i + g_j \quad (11)$$

or

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{g} \quad (12)$$

in a matrix form. Obviously, the DC response of the circuit is the solution of (12), which is also the solution of (2) if (2) is convergent. This proves the correctness of our methodology.

It is conceivable to get rid of “iterations” by directly connecting inputs and outputs, and in turn, making the whole circuit a closed loop. As a result, the circuit can converge to the steady state instantly, mimicking the iterative process. As a result, the time complexity of the iterations is immensely scaled down from $O(K \times NZ)$ to $O(1)$, where NZ is the number of nonzeros in a sparse matrix and K is the number of iterations. $O(NZ)$ is eliminated by the inherent parallelism of RRAM-based crossbar arrays and $O(K)$ is eliminated by the one-step response. It is worth emphasizing that the DC response of the circuit does not rely on the convergence condition of the iterative form (2). We have experimentally verified this statement by HSPICE simulations (see Section 4.2). This is understandable because a closed-loop circuit can automatically stabilize at its steady state, if there exists one. In fact, the circuit behavior corresponds to (12) (direct form) instead of (2) (iterative form). Besides the elimination of iterations, this is another advantage of our methodology compared with the accelerators for iterative methods [6, 7, 11, 21], which are “real simulations” of iterative methods and apparently subject to the convergence condition.

3 SCALABLE ARCHITECTURE FOR LARGE-SCALE SPARSE LINEAR SYSTEMS

Summing up the discussion above, the accelerator for illustration purpose shown in Fig. 2 could efficiently accomplish dense linear systems without an ancillary procedure. One can simply connect

Figure 3: Architecture for large-scale sparse linear systems.

dense crossbar arrays together to solve larger dense problems and realize a maximum degree of parallelism. However, most matrices in practical applications are sparse. Directly mapping a sparse matrix on dense RRAM crossbar arrays is infeasible, as zero elements will not be accessed, resulting in high redundancy in both resource utilization and power consumption. Here we introduce a novel RRAM-based architecture to handle large-scale sparse linear systems and also propose a mapping strategy for the proposed architecture. The architecture for sparse linear systems is designed following three rules. First, it should also complete a linear system in $O(1)$ time complexity by a closed-loop response. Second, the hardware resources should chiefly be used for nonzero elements. Third, considering that nonzero elements can arbitrarily be distributed in a sparse matrix, the architecture should be reconfigurable to adapt to the nonzero distribution.

3.1 Architecture

Fig. 3 shows the architecture for large-scale sparse linear systems. It is constructed with a set of crossbar clusters, peripheral adder units, ADCs and DACs, and a block controller. We organize RRAM crossbar arrays in a two-level structure. The top level contains 4 crossbar clusters and each of which is composed of a set of identical-size crossbar blocks and routing units. Basically, we partition the sparse iterative matrix B into sub-matrices and map the nonzero sub-matrices to the crossbar blocks. Empty sub-matrices are not mapped to hardware.

Crossbar clusters. There are 4 crossbar clusters. A cluster contains a series of independent RRAM crossbar blocks with identical size. The block sizes in different clusters are different, which are 64×64 , 128×128 , 256×256 and 512×512 . There are a set of global WLs and global BLs (not shown in Fig. 3) shared by all clusters. An $S \times S$ block is constituted by two $S \times S$ RRAM crossbar arrays (for positive and negative elements) and S I-V converters (i.e., the OpAmp ① shown in Fig. 2) connected to the local BLs in the block. The blocks are used to map the nonzero sub-matrices in the sparse matrix B . Normally we need more small blocks than large blocks to maximize resource utilization. However, smaller blocks have larger overhead so we merge adjacent small blocks into larger blocks when possible. The mapping strategy will be introduced in the next subsection.

Figure 4: (a) Schematic of WL routing units. (b) Schematic of BL routing units and an adder unit.

Adder units. They are used to add vector b to the global BL voltages. An adder unit is the OpAmp ② with its peripheral resistors, shown in Fig. 2, which is also shown in Fig. 4b.

Routing units. To realize the reconfigurability to adapt to arbitrary nonzero element distribution in sparse matrices, the connections between the crossbar blocks and the peripheral circuits are implemented by routing units. Basically, an $S \times S$ block can be mapped to any position whose horizontal and vertical coordinates are both aligned at S . The WL and BL routing units are built to achieve this goal. Fig. 4 shows the schematics of the routing units. We use transmission gates (TGs) to build BL routing units. The i -th adder unit can be connected to the $(nS+i)$ -th column (n is an integer) in the $S \times S$ cluster. TGs have ON-resistance so they will degrade the accuracy of the result. However, since the ON-resistance is much smaller than R_1 in the adder units, the impact is tiny. For the WL routing units, using TGs will cause voltage drop on the local WLs so we instead use voltage followers with a TG at the inputs. The structure of WL routing units is similar to that of BL routing units. If a WL routing unit needs to be turned off, we just turn off both the TG and OpAmp.

Block controller. It is used to control the sub-matrix mapping, namely, the connectivity of the routing units. A set of registers is used to store the configuration bits of the TGs.

DACs and ADCs. They perform D2A and A2D conversions before and after the accelerator operation, respectively. There is no intermediate D2A or A2D conversion.

3.2 Sparse Matrix Mapping Algorithm

The purpose of this step is to partition the iterative matrix B ($N \times N$) into proper-size sub-matrices such that all sub-matrices that contain nonzero elements can be mapped to crossbar blocks of the corresponding size. We apply a search algorithm to find out all valid blocks that can cover all nonzero elements, and expand them recursively to form larger blocks. Fig. 5 shows the mapping algorithm with an illustration.

The finest granularity for matrix partitioning is 64×64 . We first create a search area B' whose size is $N' \times N'$ where N' is aligned at 64 ($N' = \lceil \frac{N+63}{64} \rceil \times 64$). The search area B' is partitioned into sub-areas of size 64×64 which are aligned at 64. Considering that $N' \times N'$ is larger than $N \times N$ when N is not dividable by 64, matrix B can be moved in the larger search area B' . There are $(N' - N) \times (N' - N)$ possibilities of the position of B in B' . For each possible position of B , we try to merge small non-empty sub-areas of B' (containing at least one nonzero element at the corresponding position of B)

Figure 5: (a) Mapping algorithm. (b) Illustration of mapping algorithm.

Table 1: Parameters of ADCs, DACs and OpAmps.

	Power (μ W)	Delay (μ s)	Precision (bits)
DAC [9]	210	10	14-16
ADC [17]	1.9	1	10
OpAmp	~ 28	0.0004	N/A

to form larger sub-areas when possible, until the largest sub-area size is 512×512 . For example, if 4 adjacent 64×64 non-empty sub-areas can form a 128×128 sub-area and the resulting sub-area is aligned at 128, the merge is valid. The merge scheme with the least non-empty sub-areas is selected as the mapping, where the formed non-empty sub-areas in B' correspond to the non-empty sub-matrices in matrix B .

After the mapping procedure, we have transformed the sparse iterative matrix B into a set of non-empty sub-matrices, thus each one can be mapped to a crossbar block with its connections routed by the block controller. As a result, the sparse accelerator still holds the equivalent structure and function of the circuit shown in Fig. 2, and thus, returns the solution after the time of DC response.

4 SIMULATION RESULTS

The proposed accelerator is simulated with HSPICE. We implement the Jacobi iterative method on the accelerator. The RRAM resistance range is from $5K\Omega$ to $5M\Omega$ [1, 14, 18]. Eight practical sparse matrices from the SuiteSparse Matrix Collection [4] are used as the test cases. Their dimensions are from 7102 to 40806. The maximum problem size that can be handled by our accelerator is only restricted by the hardware resources of the architecture. In our simulations, however, due to the large memory usage of HSPICE, we were unable to simulate larger matrices. The parameters of ADCs, DACs and OpAmps are listed in Table 1, where the parameters of OpAmps are from our HSPICE simulations of a 3-stage OpAmp (working in the negative-feedback state) designed in 45nm technology. The accelerator is compared with a Jacobi iterative sparse solver on an NVIDIA Tesla P40 GPU. The GPU solver is built using the cuSPARSE and cuBLAS libraries. The convergence tolerance of the GPU solver is 10^{-6} . We use the command `nvidia-smi` to measure the runtime power of the GPU.

4.1 Accuracy Upper Bound

Since our accelerator is an analog circuit, it suffers from the inherent noise. We first evaluate the highest accuracy that can be achieved by our accelerator. We simulate the illustration circuit shown in Fig. 2 for four randomly-generated dense matrices. The root-mean-square-error (RMSE) between the obtained solution by the circuit and the precise solution is calculated. We consider an ideal case

Figure 6: RMSE evaluation (for illustration circuit in Fig. 2).

Figure 7: (a) Solution converges to DC response on accelerator. (b) Solution diverges by a GPU iterative solver.

where RRAMs have no resistance variation and a non-ideal case where RRAMs have 20% ($1-\sigma$ value) variations in the resistance. The ideal case has little practical significance but it tells the accuracy upper bound that our accelerator can achieve. Fig. 6 shows the results. The ideal case can achieve an accuracy level of 10^{-4} . The error increases with the increase of the problem size. When RRAMs have variations, rendering a more practical situation, the RMSE is at the magnitude of 10^{-2} . The RMSE decreases with the increase of the problem size. This is because the positive and negative random variations of the RRAM resistance tend to be canceled by each other for large crossbar arrays. For the scalable architecture for sparse matrices, the RMSE will obviously be larger.

4.2 Convergence

To understand how the solution gets convergence on our accelerator and to prove that our accelerator is not subject to the convergence condition, we perform a transient simulation on a linear system for which classical iterative methods diverge. A 10fF capacitor is added to each WL and each BL to act as the parasitic capacitance. The first 4 components of the solution are shown in Fig. 7, which clearly shows that the hardware converges (to the DC response) just in a few picoseconds, while the GPU-based iterative solver diverges.

4.3 Comparison with GPU Solver

Here we show the simulation results of the 8 practical sparse matrices on the proposed scalable accelerator.

4.3.1 Accuracy. Fig. 8 shows the RMSE of the solution of the 8 sparse matrices solved by the scalable accelerator. For the ideal case without any variation in the RRAM resistance, the RMSE is at the magnitude of 10^{-2} , while for the non-ideal case with variations in the RRAM resistance, the RMSE is several times larger and can reach the magnitude of 10^{-1} . For practical applications, solutions with 10^{-1} RMSE are usually insufficient. However, the solution obtained by our accelerator can be used as the initial seed for a further

Figure 8: RMSE evaluation.

Figure 9: Performance comparison between accelerator and GPU.

Figure 10: Performance comparison between pure GPU solving and hybrid GPU refinement.

refinement on GPU. Since the seed is close to the precise solution, the refinement process needs less iterations to get convergence. We will show the related results in the following contents.

4.3.2 Performance. We first compare our accelerator with the GPU solver, as shown in Fig. 9. For these problems, the number of iterations to get convergence ranges from 3 to 39 while the running time ranges from 0.26ms to 2.31ms on GPU. Our accelerator gets the solution in $O(1)$ time complexity, and thus, the latency is constant and independent with the problem size. Since the voltages of V_g 's are provided by DACs and the result needs to be converted to digital signals by ADCs, the total latency is dominated by the latency of ADCs and DACs. According to Table 1, the total latency of our accelerator is less than $12\mu s$. As a result, the speedup is $100\times$ on average. Comparing the constant latency of our accelerator and the $O(K \times NZ)$ complexity of software solvers, our accelerator can obviously get higher speedups for larger problems.

We then evaluate the overall performance if the solution obtained by our accelerator is used as the initial seed for a further refinement on GPU. For comparison, we also run a full solving process on GPU starting from a zero seed. The hybrid refinement method can reduce 1 to 7 iterations compared with the full solving process. As shown in Fig. 10, the hybrid refinement method can reduce about 35% of the solving time on average, due to the reduced iterations.

4.3.3 Energy. Based on the performance data (Fig. 9), the measured GPU runtime power (around 70W) and the simulated accelerator power ($<20W$ for the tested cases), Fig. 11 compares the energy consumption. Our accelerator reduces the energy by $1000\times$ on average compared with the GPU solver. If the low-precision solution obtained by our accelerator is used as the seed for a refinement on GPU, the total energy is reduced by about 35% on average, according to the performance data shown in Fig. 10, as in the hybrid refinement approach most of the energy is consumed by GPU.

Figure 11: Energy comparison between accelerator and GPU.

To understand the energy breakdown, OpAmps and DACs consume more than 99% of the total energy, while the energy consumption of the other components is negligible.

5 CONCLUSIONS

A novel method to solve large-scale sparse linear systems by utilizing RRAM-based crossbar arrays is proposed. The key feature of the approach is the $O(1)$ time complexity by eliminating iterations of iterative methods and the elimination of the convergence condition. The simulation results demonstrate the more superior performance and energy efficiency of RRAM-based systems than conventional massively parallel GPUs. We believe that RRAM-based accelerators for scientific computing applications have practical significance, especially for the cases which can tolerate approximations.

ACKNOWLEDGEMENTS

This work was supported by National Key R&D Program of China (2018YFA0701500), by Key Research Program of Frontier Sciences, Chinese Academy of Sciences (ZDBS-LY-JSC012), by Strategic Priority Research Program of Chinese Academy of Sciences (XDB44000000), by National Natural Science Foundation of China (61804155 and 61834006), by the Youth Innovation Promotion Association CAS, and by Beijing Academy of Artificial Intelligence (BAAI).

REFERENCES

- [1] H. Chang et al. 2011. Physical mechanism of HfO₂-based bipolar resistive random access memory. In *VLSI-TSA*. 1–2.
- [2] M. Cheng et al. 2019. TIME: A Training-in-Memory Architecture for RRAM-Based Deep Neural Networks. *IEEE TCAD* 38, 5 (May 2019), 834–847.
- [3] P. Chi et al. 2016. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *ISCA*. 27–39.
- [4] Tim Davis. [n.d.]. SuiteSparse Matrix Collection. <https://sparse.tamu.edu/>
- [5] Timothy Davis. 2006. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics.
- [6] B. Feinberg et al. 2018. Enabling Scientific Computing on Memristive Accelerators. In *ISCA*. 367–382.
- [7] Manuel Le Gallo et al. 2018. Mixed-precision in-memory computing. *Nature Electronics* 1, 4 (2018), 246–253.
- [8] B. Li et al. 2015. RRAM-Based Analog Approximate Computing. *IEEE TCAD* 34, 12 (Dec 2015), 1905–1917.
- [9] Maxim Integrated Products, Inc. [n.d.]. MAX5214/MAX5216: 14-/16-Bit, Low-Power, Buffered Output, Rail-to-Rail DACs with SPI Interface. <https://datasheets.maximintegrated.com/en/ds/MAX5214-MAX5216.pdf>
- [10] D. Patterson et al. 1997. A case for intelligent RAM. *IEEE Micro* 17, 2 (March 1997), 34–44.
- [11] Isaac Richter et al. 2015. *Memristive Accelerator for Extreme Scale Linear Solvers*. Technical Report. University of Rochester. <http://www2.ece.rochester.edu/~xiguo/gomac15.pdf>
- [12] Y. Saad. 1996. *Iterative Methods for Sparse Linear Systems*. PWS Pub. Co.
- [13] A. Shafiee et al. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *ISCA*. 14–26.
- [14] S. Sheu et al. 2009. A 5ns fast write multi-level non-volatile 1 K bits RRAM memory with advance write scheme. In *VLSIC*. 82–83.
- [15] Z. Sun et al. 2019. Fast Solution of Linear Systems with Analog Resistive Switching Memory (RRAM). In *ICRC*. 1–5.
- [16] Zhong Sun et al. 2019. Solving matrix equations in one step with cross-point resistive arrays. *PNAS* 116, 10 (2019), 4123–4128.
- [17] M. van Elzakker et al. 2010. A 10-bit Charge-Redistribution ADC Consuming 1.9μW at 1 MS/s. *IEEE JSSC* 45, 5 (May 2010), 1007–1015.
- [18] H.-P. Wong et al. 2012. Metal-Oxide RRAM. *Proc. IEEE* 100, 6 (June 2012), 1951–1970.
- [19] Peng Yao et al. 2017. Face classification using electronic synapses. *Nature Communications volume 8* (2017).
- [20] S. Yu. 2016. *Resistive Random Access Memory (RRAM): From Devices to Array Architectures*. San Rafael, California.
- [21] Mohammed A. Zidan et al. 2018. A general memristor-based partial differential equation solver. *Nature Electronics* 1, 7 (2018), 411–420.