Received 2 March 2020; revised 26 April 2020; accepted 11 May 2020.
Date of publication 15 May 2020; date of current version 6 July 2020.

Digital Object Identifier 10.1109/JXCDC.2020.2995123

Memristor-Based Analog Recursive Computation Circuit for Linear Programming Optimization

LIUTING SHANG (Student Member, IEEE), MUHAMMAD ADIL (Student Member, IEEE), RAMTIN MADANI (Member, IEEE), and CHENYUN PAN (Member, IEEE)

Department of Electrical Engineering, The University of Texas at Arlington, Arlington, TX 76019 USA

CORRESPONDING AUTHOR: L. SHANG (liuting.shang@mavs.uta.edu)

ABSTRACT Linear programming optimization is central to engineering designs, logistics management, and decision-making in every sector of the economy. Traditional hardware using CPU and GPU platforms for this purpose is severely limited by the scaling of the transistor technology. In this article, we design an analog in-memory computation circuit for accelerating linear programming optimization problems. The scheme includes a memristive crossbar array and analog peripheral circuits that do not require DAC/ADC between each algorithm iteration. In addition, several key parameters related to nonideal device characteristics and interconnect parasitics are discussed for providing practical guidelines. Furthermore, three design schemes are proposed to alleviate the computation error caused by the interconnect resistance for a large-scale crossbar array implementation. Optimal design parameters are quantified under a given number of array size and memristive resistance. Finally, the proposed hardware accelerator and error mitigation techniques are applied to six real-world power system optimization problems. The results show that the average error of generator power and the overall cost is less than 3%. It is demonstrated that the proposed accelerator achieves area, delay, and energy consumption reductions of $\sim$151×, $\sim$33×, and $\sim$21×, respectively, compared with the CMOS digital circuits at the 16-nm technology node for a 1000 × 1000 array with 6-bit precision.

INDEX TERMS Accuracy, delay, energy, in-memory computation, linear programming optimization, memristor, performance.

I. INTRODUCTION

A RECENT surge of research on data-driven applications, such as nonconvex optimization and machine learning, has raised a great demand for energy-efficient and high-performance computation hardware. Among various applications, the optimization algorithm plays an important role in a wide range of fields, including engineering designs, logistics management, and decision-making [1]. These applications are extremely computationally intensive for a large problem size with considerable iterations. With the traditional CMOS technology approaching the end of Moore's law scaling and the growing challenge of the Von-Neumann memory bottleneck [2], conventional CPU- and GPU-based computing platforms cannot keep up with the ever-increasingly complex scenarios of real-world optimization applications. Novel computing paradigms, such as the "compute by physics" [3], [4], i.e., compute in resistive networks through Ohm's law and Kirchhoff's current law, have regained much attention from researchers. Benefited from the emerging nonvolatile memristor technologies [5], [6], "compute by physics" can significantly improve the energy efficiency and alleviate the memory bottleneck by utilizing the analog computation paradigm and in-memory computation architecture.

Recently, a lot of works have presented memristor-based in-memory computation applications, such as the dot product engine, recursive neural network, and linear system solver [7]–[10]. One previous work [9] has also proposed a memristor-based crossbar architecture for solving convex optimization problems. However, it uses an analytical approach to evaluate the algorithm without the consideration of actual circuit design, assuming that a system of linear equations can be solved with unique solutions. As suggested in this work, the nonideality of memristor parameters, transistor variation, and, most importantly, the interconnect resistance at small technology nodes could impose substantial impacts on the application-level accuracy. Therefore, it is imperative to consider key device and interconnect parameters to understand the true value of a large-scale analog-based computing circuit for solving optimization problems. Regarding the optimization algorithm, traditional optimization techniques

run serial streams of calculations to obtain an optimal solution for a problem, in such a way that only one operation is done at a time and the outcome of the previous step is needed for the next one. As a result, these algorithms perform poorly when handling large problems and cannot leverage analog computing architectures. Although several memristor-crossbar-based recursive circuits [11]–[14] that have comparatively simple structure with parallel computing capability have been developed for Hopfield neural networks and eigenvector solver, there are many differences between these works and the proposed work in terms of optimization algorithms. None of these works targets for constrained optimizations; the design purpose, delay/error source, and the rule/complexity of the recursive process are very different; and the circuit-level design for the nonconvex optimization algorithm is still absent. To enable efficient real-time execution of such an optimization algorithm on a hardware solution, a wide range of low-complexity algorithms has gained popularity in recent years [15]–[17].

In this article, we adopt a first-order method that divides the computational task of each algorithmic step into several smaller ones, which can be carried out in parallel [15], [18]. This feature of the proposed approach offers unprecedented scalability in solving the broad class of real-world optimization problems. Compared with the standard alternating direction method of multipliers (ADMM) approach adopted by previous work [9], the method in this article requires a smaller size of crossbar array, which improves the overall accuracy as well as the delay and energy dissipation. In addition, the optimization method can be easily extended to solve a more complex second-order cone program (SOCP). The high level of parallelism also makes the proposed algorithm fit well with the proposed analog computing circuits. Furthermore, the proposed optimization algorithm only requires simple operations during each iteration, such as the absolute, summation, and vector-matrix multiplication, which significantly reduces the complexity of the peripheral circuits. We are, in part, motivated by the application of this work to mixed-integer linear programming (MILP) that is central to the operation of the electricity markets. Solving MILP problems using state-of-the-art branch-and-bound algorithms may involve a large number of linear programs with similar structures. Hence, it is beneficial to leverage analog hardware for this purpose.

On the hardware design side, with complete analog peripheral components, the proposed analog computing platform does not require ADC/DAC between each iteration, which could save up to 50% energy in the application based on in-memory dot-product engine [19]. In addition, three solutions are proposed to reduce the impact of nonideal wire resistance and alleviate the accuracy degradation. Furthermore, we develop a fast and accurate circuit-level simulation framework to perform a large design space exploration with the consideration of circuit designing parameters and process variation of mismatched memristors/transistors in the analog peripheral circuits. The results presented are generic and can be applied to other in-memory computing applications based on the crossbar architecture, such as deep neural network accelerators. Finally, the proposed schemes are applied to six real-world power system optimization problems. The major contributions of this work are highlighted in the following.

1) We design an analog in-memory computation circuit platform to accelerate large-scale recursive linear programming algorithms for solving constrained optimization problems with a complete analog peripheral circuit. To the best of our knowledge, this is the first full circuit implementation for this type of optimization problem.
2) We develop a fast and accurate simulation framework for performing large-scale design space explorations to address key design tradeoffs among device-, circuit-, and system-level parameters.
3) Three solutions are proposed to alleviate the accuracy degradation due to the wire resistance existed in the proposed circuit as well as in the generic crossbar-based in-memory computing circuits.
4) The results from this work can provide valuable insights to material researchers and technologists to understand the requirement and better design analog integrated circuit and devices for memristor-based in-memory computation applications.

II. LINEAR PROGRAMMING OPTIMIZATION PROBLEMS

In this section, we introduce the optimization algorithm that can be efficiently implemented with the proposed analog computing platform in Section III. This article is concerned with the class of linear programming optimization problems of the form

$$\text{minimize } c^\top x \tag{1}$$
$$\text{s.t. } Ax = b \tag{2}$$
$$x \geq 0 \tag{3}$$

where the vector $x \in \mathbb{R}^n$ represents the unknown decision variable, and $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$ are given. The goal is to determine an optimal solution $x^{\text{opt}} \in \mathbb{R}^n$ that minimizes the linear objective function (1) while satisfying the affine equality constraint (2) and inequality constraint (3).

In order to solve the problem (1)–(3) on analog platforms, we adopt a computationally cheap first-order algorithm from [15] and [18]. The algorithm is outlined as follows:

$$s \leftarrow \mathbf{0}_n \tag{4}$$
$$l \leftarrow +\infty \tag{5}$$

while $l \geq$ tolerance

$$q \leftarrow |s| \tag{6}$$
$$r \leftarrow Mq \tag{7}$$
$$s \leftarrow \frac{s}{2} - \frac{r}{2} + h \tag{8}$$
$$l \leftarrow \|2d - s - r\|_2 \tag{9}$$
$$x \leftarrow \frac{s+q}{2} \tag{10}$$

where $M \triangleq 2A^\dagger A - I_{n\times n}$ and $h \triangleq A^\dagger b - (\eta/2)(c - Mc)$. In the above mentioned algorithm, we represent an error, and once it is below a certain tolerance level, then the convergence is achieved. This convergence process requires up to 300 iterations in the cases that we are going to set as examples. In the worst situation, each iteration requires $60 \times [1560, 1560]$ vector-matrix multiplication and thousands of extra sum, multiply, and absolute operations. Here, we use n, i.e., the size of matrix M, to denote the problem size in the rest of this article.

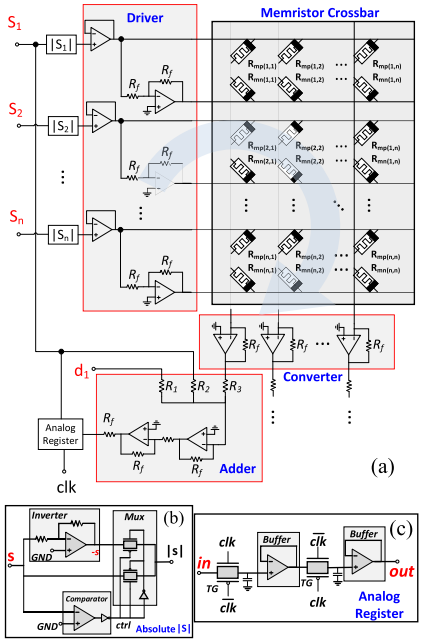

FIGURE 1. (a) Overall schematic of the proposed circuit. (b) Absolute module. (c) Analog Register.

Variables "s" and "q" in the abovementioned algorithm account for $x + \lambda$ and $x - \lambda$, respectively, where λ is the dual vector associated with the inequality constraint (3). The surrogate variables s and q in the proposed Douglas–Rachford (DR) framework allow us to consider a memristor crossbar of size $n \times n$ to perform (7). Alternatively, the standard ADMM approach requires a crossbar of size $(n + m) \times (n + m)$, which is not practical for the application targeted in this work. In addition, the proposed approach relies on an absolute value operator as opposed to projection into a nonnegative orthant. The proposed DR approach can be readily applied to a broader class of convex/nonconvex optimization problems via substitution of (6) with other mappings. It should be noted that as an alternative to the off-line calculation of matrix M, we can adapt the approach proposed in work [20] to further streamline the process.

III. CIRCUIT SCHEME AND MODELING FRAMEWORK

In this section, we propose a hardware accelerator and aim to significantly speed up the recursive computation. Because of the slow SPICE circuit simulation for a large problem size, we develop a modeling framework to efficiently explore a large crossbar array size, including peripheral circuits with a minimum accuracy penalty.

A. PROPOSED RECURSIVE CIRCUIT SCHEME

The scheme includes a memristive crossbar-based dot product engine for the core vector-matrix multiplication and peripheral circuits for the other operations, including the addition and absolute function. As shown in Fig. 1(a), the memristive crossbar-based vector-matrix dot product engine consists of the memristive crossbar array, drivers, and current-to-voltage converters. We adopt the CMOS high-performance Arizona State University Predictive Technology Model at the 16-nm technology node to design all components and interconnect in the circuit.

In the memristive crossbar array, the multilevel memristors are used to store signed weight in the matrix M in (7). We linearly map the weight values to the conductance of two memristors. Each weight value corresponds to two memristors, which presents positive and negative values, respectively. When one memristor stores the magnitude of a positive or negative value, the other memristor is set to the cutoff state to approximate the disconnection. The input voltages provided by the driver module present the vector q shown in (6). To multiply the input vector q with the matrix M stored in memristor array, we pass input voltages that present s in (6) through the analog absolute module. Then, the positive signals are connected to the memristors that store positive values, and an inverted voltage is generated after the absolute and connected to the memristors storing negative values. Consequently, the overall current flowing to the output of each column is the sum of the product of all input voltages and memristor conductance. Once the signed multiplication is achieved, the output current is converted to a voltage by using an inverting amplifier. Next, the summation in (10) is achieved by a three-input analog inverting summing amplifier. The intermediate results at the end of each iteration are stored in capacitor-based analog registers. During the next clock cycle, the stored voltages are passed as input voltages to the next iteration. The initial voltages on the capacitors at the first iteration are set to be d in (8). Note that due to the limitation of the highest available memristance, we tune the memristance to compensate for the leakage current flow through the cutoff memristors during the programming stage. The memristance is tuned based on the following equations:

$$R_p = \frac{R_{p0} \times R_{\max}}{R_{p0} + R_{\max}}, \quad R_n = R_{\max}, \text{ if } W > 0 \tag{11}$$

$$R_n = \frac{R_{n0} \times R_{\max}}{R_{n0} + R_{\max}}, \quad R_p = R_{\max}, \text{ if } W < 0 \tag{12}$$

where W presents the weight to be stored, R_{p0} and R_{n0} are the corresponding memristance that is mapped into "positive" and "negative" values without tuning, respectively, R_p and R_n are the memristance updated by (11) and (12), respectively, and $R_{\min}$ and $R_{\max}$ are the minimum and maximum memristor resistances, respectively. Fig. 1(b) and (c) shows the schematic of the analog absolute module and analog register module. The op-amp design follows standard differential-input single-ended output seven-transistor operational amplifiers with a supply voltage of 1.2 V.

B. MODELING FRAMEWORK

The accurate circuit-level behavior can be precisely simulated by the SPICE simulator. However, running SPICE simulations is very time-consuming, especially for large-scale circuit applications, which makes it impractical to perform a large design space exploration with multiple design parameters across device and circuit levels, especially some parameters require a large number of resampling. Therefore, we create a fast and accurate numerical modeling framework to emulate the behavior of the proposed circuit without the need for detailed SPICE simulations. Such a framework is able to evaluate circuit variation, calculation error of the algorithm, component delay, area, and energy. Here, the error is calculated for the whole linear programming optimization

problem, and it takes iterations, transient delay, matrix multiplication, and nonideal circuit components, such as transistors and op-amps into account. The final output of the circuits is the vector s in (8), which is shown as s_1 to s_n in Fig. 1(a). All the errors in this article present the average difference between s calculated from the circuit and the ideal s. The definition of error is expressed as

$$\text{Error}\,(\%) = 100\% \times \sum_{i=1}^{n} \frac{|s_{\text{circuit}}(i) - s_{\text{ideal}}(i)|}{s_{\text{ideal}}(i) \times n} \qquad (13)$$

where $s_{\text{circuit}}(i)$ is the simulated output of the ith path of the proposed circuit, $s_{\text{ideal}}(i)$ is the ith value in ideal s vector, and n is the crossbar array size.

The modeling framework includes two parts, namely, the current/voltage calculation for the crossbar array and input–output voltage transfer characteristic modeling for peripheral circuits with op-amp circuit modules.

In terms of the crossbar array, we adopt a modeling framework to simulate the node voltages by creating a conductance matrix based on Ohm's law and Kirchhoff's current law [21]. All wire resistance and memristance are included during the construction of the conductance matrix, and we assume that each output node is ideal virtual grounded. With given input voltages and memristance, the output current to the converter module can be calculated efficiently. We also validate the numerical results with standard circuit SPICE simulation results, and a negligible difference is observed. For the peripheral voltage transfer characteristic modeling, we obtain fitting functions by sweeping the input voltages and gathering the output voltages of the converter, adder, register, and absolute modules. Furthermore, we include the impact of process variation, such as the transistor mismatch, by generating a sufficient fitting database. To validate the modeling approach, we simulate a 100 × 100 crossbar array under the 3σ transistor width variation of 30% of the feature size F.

According to Fig. 2(a), the transient s (iterated parameter) generated from the modeling method and SPICE has a similar magnitude of error from ideal curves. From the comparison results shown in Fig. 2(b), the error distribution of the modeling method matches well with accurate SPICE simulation results. More details about the source of the error for the framework itself are discussed in the Supplementary Material. The framework achieves ∼1000× speed improvement compared with SPICE simulation when generating Fig. 2. Similar to the transistor variation, the fitting object of circuit delay is also integrated into the framework by taking interconnect resistance and capacitance, memristor resistance, and op-amp intrinsic delay into account. Based on the delay, the corresponding energy dissipation is calculated, which is dominated by the peripheral op-amp circuits. An example of delay and energy per iteration is shown in Table 1, indicating that the framework can well capture the performance of delay and energy.

IV. SIMULATION RESULTS, ANALYSES, AND SOLUTIONS

A. IMPACT FROM NONIDEAL PARAMETERS AND PROCESS

Due to the nature of the analog computation, it is critical to quantify the impact of nonideal parameters on the overall computation error. In this section, we investigated the interconnect-, transistor-, and memristor-related parameters, such as conductance levels, minimum memristance, ON/OFF ratio, and memristance drift/variation. The metric of the computational error is calculated by taking the geometric mean of the difference between the circuit simulation outputs and ideal outputs. Note that most of the results presented in this section are applicable to a broad field of crossbar-based in-memory computation circuits using memristor devices.

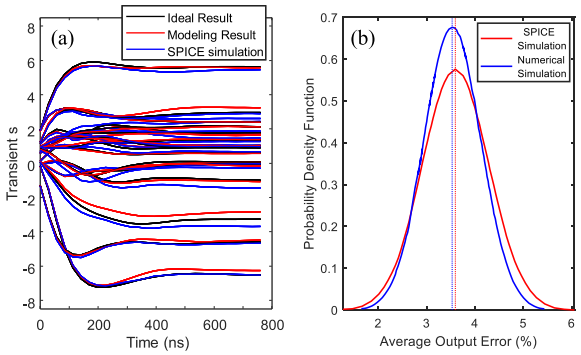

FIGURE 2. **(a) Simulated transient s curves based on the modeling framework for a 100 × 100 array size, where s is the iterated parameter defined in (4). (b) Comparison of the error probability density function of the proposed modeling framework and a full SPICE circuit simulation under the 3σ transistor width variation of 30% of the feature size F (16 nm) for a 100 × 100 crossbar array.**

TABLE 1. **Comparison of delay and energy per iteration between the results from the proposed framework and SPICE simulation.**

Results	Proposed Framework	SPICE Simulation
Delay per Iteration (ns)	5.78	5.21
Energy per Iteration (pJ)	32.77	29.54

1) IMPACT OF INTERCONNECT RESISTANCE

As the technology scales down, the interconnect resistance increases significantly due to the reduced cross section area and the size effect [22]. For a large crossbar array, the long interconnect leads to a substantial voltage drop and lowers the actual voltage across the memristor, which causes the output current of the crossbar array to be smaller than the expected value.

To quantify the impact of interconnect resistance, Fig. 3 shows that the error increases considerably with the increase of interconnect wire resistance. In addition, a large ON/OFF ratio of the memristor device increases the average memristance, which helps the circuit to tolerate a larger interconnect resistance. In this article, we fix the ON-memristance as 100 KΩ to efficiently compare the effect of changing average memristance with the ON/OFF ratio.

2) IMPACT OF TRANSISTOR VARIATION

For all analog circuits, the process variation due to the transistor mismatch is a major concern. Therefore, we quantify the impact of transistor width variation on the computation error, as shown in Fig. 4. We vary the size of the crossbar array, and the influence of transistor width variation decreased with the increase in the array size. It can be observed that the

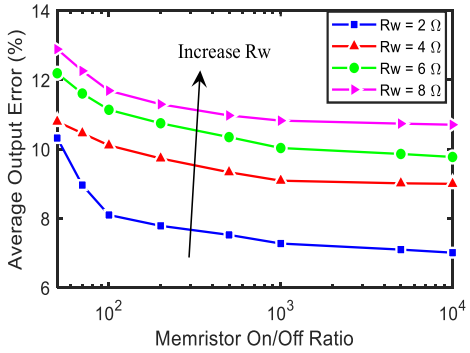

FIGURE 3. **Relationship between error and memristor ON/OFF ratio for different interconnect segment resistances between neighboring memristors for a crossbar array size of 1000 x 1000.**

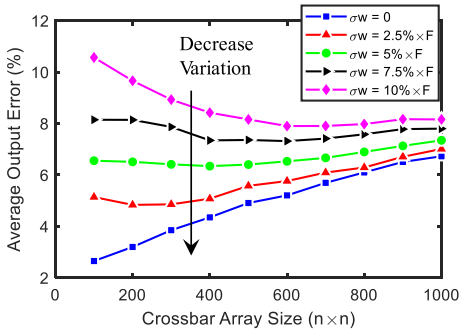

FIGURE 4. **Relationship between error and transistor width variation for a crossbar array size of 1000 x 1000.**

process variation has a smaller impact on a larger array. That is because the impact of transistor variation on op-amp can be almost canceled by each other when the array size is large enough.

3) IMPACT OF MEMRISTOR PARAMETERS

Computation error caused by resistance/conductance variation has been well discussed in other works [23]–[25], showing that the programming process and inherent device manufacturing error are both determine the device performance. In Fig. 5, device-to-device and cycle-to-cycle conductance variation magnitude varies with a standard deviation up to $\sigma/w = 20\%$ of the normal memconductance value. For a small problem size of 50×50 shown in the inset of Fig. 5, the error significantly increases with the increase of memristor variation. For example, the error almost doubles when a 20% device-to-device variation is considered compared with the baseline case without the variation. However, this trend does not apply to large array size. For example, the error only increases less than 8% with the same variation for the array size of 1000×1000. This is because when the crossbar array size is large enough, the output errors caused by memristor variation are averaged during the summing among hundreds of branch currents in the same column in a crossbar array. On the other side, the levels of the conductance of the memristor dominated the calculation error, and the curve becomes flat when the memristor has more than 64 levels.

The conclusion from Figs. 4 and 5 reveals a potential variation tolerance of the analog dot product engine, indicating that some methods for reducing device variation,

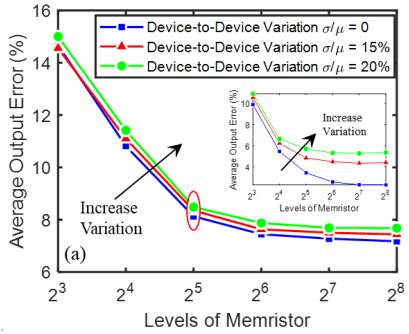

FIGURE 5. **Relationship between error and memristor levels and memconductance variation for a crossbar array size of 1000 x 1000. The inset is a comparison under an array size of 50 x 50. Conductance variation means the standard deviation of normal distribution σ. Levels of memristor mean how many levels of conductance can be achieved in a memristor. Each point in the figure is the average error of 150 samples.**

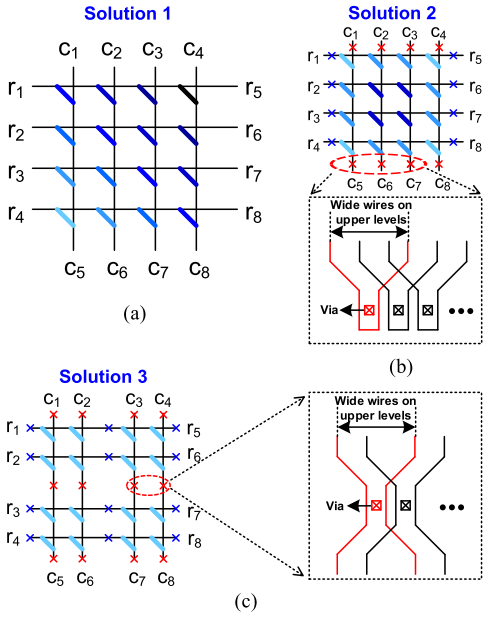

FIGURE 6. **Three proposed solutions for alleviating the interconnect wire resistance problem, where colored bar indicates memristor. (a) Solution 1 scales down the memristance to compensate for the interconnect resistance. (b) Solution 2 adds wider interconnects on upper layers to connect nodes on two sides of the array. (c) Solution 3 adds wider wires to connect two sides as well as internal nodes of the crossbar array.**

such as a complex auxiliary circuit and repetitive programming/correcting, may not be cost-effective for solving large-scale linear optimization problems.

B. SOLUTIONS FOR MITIGATING THE IMPACT OF WIRE RESISTANCE

Due to the large impact of the interconnect resistance, we propose three solutions in this section to alleviate the accuracy degradation, as shown in Fig. 6. In Solution 1, we propose to scale down the memristor weights during the programming stage to compensate for the wire resistance based on the physical position of the memristor. In Fig. 6(a), the impact of the voltage drop due to the wire resistance on each memristor is presented by the color. A darker bar indicates a longer

distance from input to output nodes and results in a larger error. In Solution 1, we set the memristor with a darker color to a smaller memristance by a scaling factor, which is proportional to the distance between the input and output nodes. As a result, the output current can be increased to compensate for the interconnect resistance. Note that Solution 1 cannot fully compensate for the interconnect wire resistance, but it helps to reduce the overall error in general, and the scaling coefficient is dependent on the application.

In Solution 2, we propose to add wide interconnects ($10 \times F$) above and underneath the row and column interconnects, respectively (for example between c_1–c_5 or r_1–r_5). Such interconnect gives a resistance per unit length of 2.65 Ω/μm based on the interconnect resistance model presented in [26], which is $\sim 40\times$ smaller compared with the minimum wire width of $2\times F$. As can be observed in Fig. 6(b), the worst case voltage drop due to the interconnect resistance exists is in the center of the crossbar array. By combining Solutions 1 and 2, the error due to the interconnect can be effectively reduced for most array sizes that are smaller than 1000×1000.

For solving even larger optimization problems with large array sizes, Solutions 1 and 2 may still be unable to sufficiently compensate worst case voltage drops in the center of the array. Therefore, we propose Solution 3 to use wide wires not only to connect two sides of the array but also to connect nodes (red crosses) inside the array so that the crossbar array can be divided into several smaller blocks, as shown in Fig. 6(c). A tradeoff exists in terms of the number of blocks, which is due to the fact that each connection to the node inside the array requires a via and a segment of narrowed wire [as shown in Fig. 6(b) and (c)], which induces a resistance overhead.

Note that there is an existing method that mitigating the wire resistance problem by calibrating the memconductance [27]. This calibration method can reduce the impact of the wire resistance substantially. However, it relies on a very complex procedure that requires fitting functions and weight computations with multiple full-size vector-matrix multiplications, which means that the result of the accurate vector-matrix multiplication comes at the cost of a heavy computation burden, especially when the weight needs to be frequently updated. Such complex weight calculations would need CPU and cannot be easily implemented by specialized hardware in many applications, such as the resource/energy-constrained edge-computing or the IoT systems. Regarding the low-cost calibration algorithm proposed in this article, i.e., the memristance scaling (Solution 1), the calculation of a scaled weight is based on its distance from the input and output of the crossbar array, as shown in Fig. 6(a). Since this scaling method is based on the physical position, it does not require complex calculations and can be implemented by the peripheral circuit.

One overhead of the proposed Solutions 2 and 3 is the additional fabrication cost due to the extra metal layers. However, the additional cost is relatively small because the wires on the upper layers are much wider than those minimum width wires at the bottom. The other overhead of the proposed solution is the extra area due to the added vias. The area overhead can be expressed as $n_b \times 0.4\%$, where n_b is the number of blocks. The last overhead of the proposed methods comes from the extra parasite capacitor, which slightly increases the delay and energy based on our simulation results. The increment of delay is $<4\%$ when ON-memristance is 100 KΩ, the ON/OFF ratio is 10^3, and the number of blocks is 10.

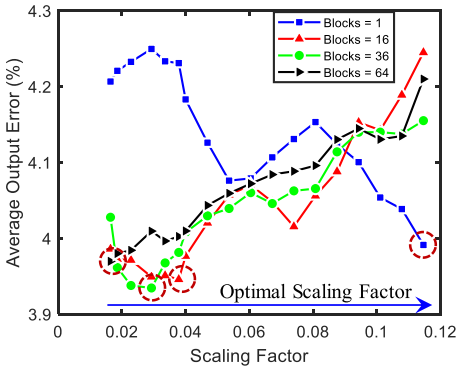

FIGURE 7. Relationship between error and scaling factor for different numbers of blocks for a crossbar array size of 1000 x 1000.

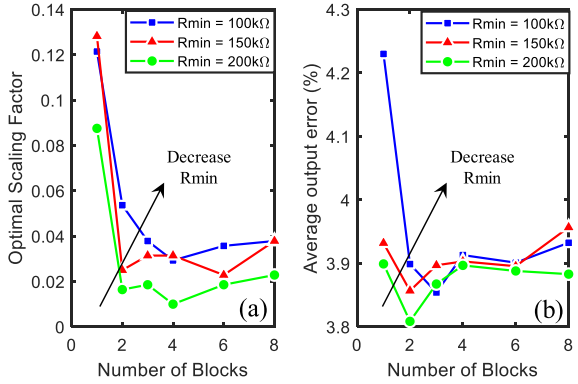

FIGURE 8. (a) Optimal scaling factor to achieve optimal performance versus different numbers of blocks for different minimum memristances for a crossbar array size of 1000 x 1000. (b) Optimal error versus different numbers of blocks for different minimum memristances for a crossbar array size of 1000 x 1000.

Similar to Solution 2, Solution 3 can be applied along with Solution 1. To verify the benefit of the combination, we explore how the number of blocks interacts with the memristance scaling factor. In Fig. 7, as the number of blocks increases, the optimal memristance scaling factor decreases to reach the minimum error because each block contains shorter interconnect wire. Meanwhile, the best number of blocks and scaling factors is also determined by the characteristics of the memristor device, such as the minimum resistance or ON/OFF ratio, because a larger memristor resistance helps to reduce the impact from the interconnect resistance. As a result, the array using memristors with a smaller minimum memristance prefers to have a large number of blocks and the scaling factor. In Fig. 8(a), the result shows that lower $R_{\min}$ requires a larger scaling factor to achieve the best error even with the help of Solution 3. Similarly, Fig. 8(b) shows that if we decrease $R_{\min}$, the circuit still needs to increase the number of blocks to achieve the best error even Solutions 1 and 3 are applied at the same time.

As shown in Fig. 9, when the number of blocks is one, circuits with different wire resistances have a comparative

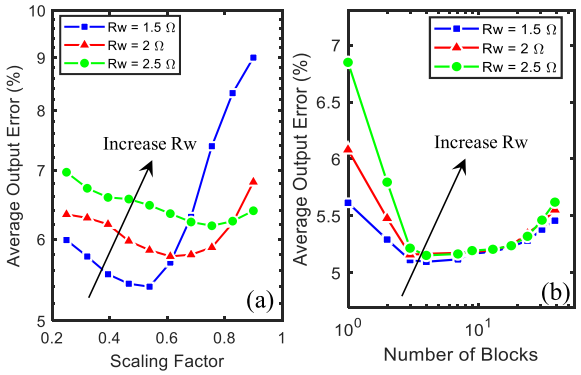

FIGURE 9. **(a) Relationship between error and scaling factor for different interconnect resistances for a crossbar array size of 1000 × 1000. (b) Relationship between error and the number of blocks for different interconnect resistances for a crossbar array size of 1000 × 1000.**

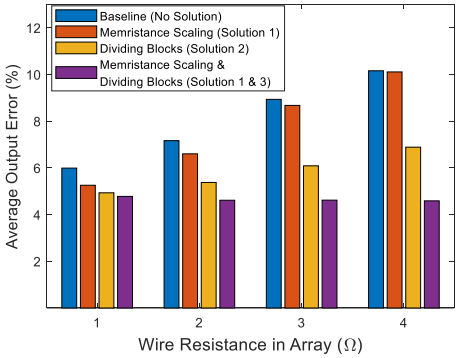

FIGURE 10. **Optimal error versus wire segment resistance using different solutions with the optimal number of blocks and scaling factors for an array size of 1000 × 1000.**

large error difference. With increasing the number of blocks by using Solution 3, the accuracy improves significantly, especially for the case when the interconnect resistance is relatively large. At the optimal number of blocks, the minimum errors tend to be consistent, indicating that the impact of wire resistance is small, and the majority of the error comes from the peripheral circuits. If we continually add blocks, the circuit error increases again due to the overhead of vertical vias resistance connecting to wide interconnects above or underneath the array.

In Fig. 9(a), it reveals that if the resistance of interconnect increased, the circuit requires a larger scaling factor to reach the smallest error, and the increased error cannot be fully compensated by this method. As shown in Fig. 9(b), by dividing the array into blocks, Solution 3 can always reduce the error to a similar level, and the error ramps up again if we keep increasing the number of blocks.

Results shown earlier quantify the impact of the minimum memristance, interconnect resistance, number of blocks, and scaling factor on the trend of the overall accuracy of the optimization algorithm. To compare the effectiveness of three solutions for a large-scale problem, we perform an exhaustive search to find the optimal parameters to achieve the minimum error, as shown in Fig. 10. Compared with the baseline case without applying the solution, using memristor resistance scaling (Solution 1) helps to reduce the error for an array

TABLE 2. **Performance benchmarking between proposed circuit and CMOS ASIC design.**

Parameters	Proposed Circuit	CMOS ASIC
Computation Error (%)	4.60	4.45
Area (mm^2)	0.007	1.057
Delay per Iteration (ns)	25.41	847.98
Energy per Iteration (nJ)	4.52	93.75

with relatively small interconnect resistance. As the wire resistance increases, the scaling method shows a limited error reduction. Regarding Solution 2, adding interconnect connections above/underneath the array achieves a better error reduction compared with Solution 1, especially when the interconnection resistance is large. By combining Solution 1 and adding extra connection via inside the array (Solution 3), we achieve the best error reduction, and up to 50% of the error is eliminated under a relatively large interconnect resistance. The majority of the remaining error comes from the peripheral circuit.

C. PERFORMANCE BENCHMARKING

In this section, we perform a performance benchmarking between the proposed hardware accelerator and the CMOS ASIC-type accelerator in terms of the computation accuracy, delay, and energy dissipation. Here, we consider an optimization problem size of 1000 × 1000. For the proposed analog computing circuit, we assume that a minimum memristance of 100 KΩ, an interconnect segment resistance of 2 Ω, and both Solution 1 and 3 are applied with an optimal memristance scaling factor. The CMOS circuit performance is evaluated based on the beyond-CMOS benchmarking methodology [28]. Due to the SRAM area and the complex multiplier structure, the layout area of the CMOS scheme is much larger than the proposed analog computing circuit. We determine the SRAM read time and area according to a state-of-the-art work [29]. Since multiple digital array multipliers and Kogge–Stone carry look-ahead adders have been used in parallel to accelerate the vector-matrix operation, the SRAM access time dominates the overall delay for the CMOS circuits, and the leakage energy dominates the overall energy dissipation. More details about the ASIC-type digital implementation and the delay and energy breakdown charts are provided in Section II in the Supplementary Material. To achieve a similar computation error of the proposed accelerator, the CMOS ASIC accelerator is designed based on a 6-bit precision. From the comparison shown in Table 2, the proposed analog accelerator provides ∼151×, ∼33×, and ∼21× reductions in terms of the area, delay, and energy dissipation per iteration, respectively. The proposed computing hardware is benefitted by the compact structure of the crossbar array, leading to a significant area saving. The major advantages of energy and delay come from the fast and efficient vector-matrix operation as well as the avoided ADC/DAC usage.

V. CASE STUDY FOR A REAL-WORLD APPLICATION AND COMPARISON

A. CASE STUDY ON POWER SYSTEM OPTIMIZATION

In this section, we consider the well-known problem of DCOPF as a case study to evaluate the merits of the proposed

hardware accelerator in Section III [30]. Consider a power grid with N nodes, L lines, and G generators. The goal is to determine the level of commitment by each generator and the flow of power throughout the network to meet the demand for power as economically as possible. This problem can be formulated as follows:

$$\text{minimize } o^\top p \tag{14}$$

$$\text{s.t. } H^\top p - Y\theta = d \tag{15}$$

$$\left|\vec{Y}\theta + \vec{f}\right| \le f^{\max} \tag{16}$$

$$p^{\min} \le p \le p^{\max} \tag{17}$$

$$e_{\text{ref}}^\top \theta = 0 \tag{18}$$

where $p \in \mathbb{R}^G$ is the vector of power injections by generators across the network and $o \in \mathbb{R}^G$ is the vector of cost coefficients.

The lower and upper limits of generator powers are enforced via constraint (17), where $p^{\min}, p^{\max} \in \mathbb{R}^G$ are given. The incidence matrix $H \in \{0,1\}^{G\times N}$ contains locations of generators, i.e., $H_{gn} = 1$ if and only if the generator g is located at node n. In addition, $Y \in \mathbb{R}^{N\times N}$ denotes the network susceptance matrix, and θ and $d \in \mathbb{R}^N$ represent the vectors of nodal voltage angles and nodal demand values, respectively. The nodal power balance (15) enforces the conservation of power at each node. In addition, $\vec{Y} \in \mathbb{R}^{L\times N}$ denotes the network branch susceptance matrix, and the vector $\vec{f} \in \mathbb{R}^L$ accounts for the effect of transformers and phase shifters. The constraint (16) enforces the thermal limits of transmission lines, where $f^{\max} \in \mathbb{R}^L$ is given. Finally, one of the nodes is marked as the reference node, at which the voltage angle is equal to zero. This is enforced via constraint (18), where all of the elements of $e_{\text{ref}} \in \mathbb{R}^N$ are equal to zero except the element corresponding to the reference bus that is equal to 1. To evaluate the proposed approach in solving real-world DCOPF problems, we have used data from the MATPOWER package [31].

B. OPTIMIZED SOLUTION FOR APPLICATION

Based on the findings from Section IV, we select a certain set of numbers of blocks and scaling factors, as well as minimum memristance, ON/OFF ratio, and device variation to simulate six real-world data set. One of the transient simulation results of generator power is shown in Fig. 11; it can be observed that the power of each generator from the circuit simulation matches well with the ideal power.

By implementing the proposed solutions, the optimal numbers of blocks and scaling factors are determined in Fig. 12. When the problem size is comparatively small (cases 1–4, smaller than 300×300), the block dividing technique (Solution 3) is not required. However, when the problem size is larger, such as cases 5 and 6, with array size 716×716 and 1560×1560, respectively, using multiple blocks helps to lower the overall error. This observation is consistent with the conclusion from Section IV. As shown in Fig. 12, by using the optimal scaling factor and number of blocks, the overall cost from the circuit simulation matches well with the ideal cost, and the average error of the generator power is controlled under 3% (the definition is similar to the (13) by replacing s with the power).

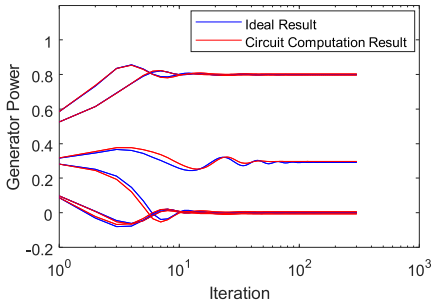

FIGURE 11. Comparison between the ideal power of generators and the simulated power based on the proposed circuits. Here, the number of memristor levels is 128, and the 3σ variation of the transistor width is 5% of the feature size F.

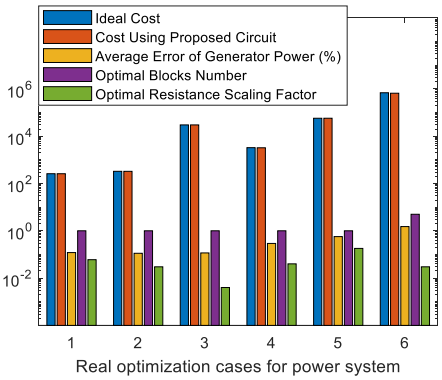

FIGURE 12. Ideal cost, cost calculated by the circuit, and the average error of generator power that can be achieved from case 1 to case 6.

VI. CONCLUSION

In this article, a memristor crossbar array-based analog in-memory computation circuit for accelerating large-scale linear programming recursive optimization problems is proposed. To accelerate simulation, a numerical modeling framework is developed to fast and accurately capture circuit-level simulations. Key device- and circuit-level parameters are investigated, including memristor variation, levels, minimum memristance, memristor ON/OFF ratio, transistor width variation, and interconnect wire resistance, to optimize accuracy, delay, energy, and area. To minimize the accuracy degradation, three solutions based on memristance scaling and interconnect routing are proposed. The results from simulations demonstrate that the proposed solutions significantly reduce errors due to the wire resistance. After applying the solution, the remaining error is mainly determined by the number of memristor levels, transistor variation, and array size. Finally, we apply the proposed framework for solving optimization programs for real-world power systems. Based on the optimal results, the proposed hardware accelerator provides $\sim$151×, $\sim$33×, and $\sim$21× reductions than CMOS benchmark in terms of the area, delay, and energy dissipation, respectively.

REFERENCES

[1] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[2] N. Lorente and C. Joachim, Eds., *Architecture and Design of Molecule Logic Gates and Atom Circuits: Proceedings of the 2nd AtMol European Workshop*. Springer, 2013.

[3] H. Kobayashi, J. L. White, and A. A. Abidi, "An active resistor network for Gaussian filtering of images," *IEEE J. Solid-State Circuits*, vol. 26, no. 5, pp. 738–748, May 1991.

[4] G. Liebmann, "Solution of partial differential equations with a resistance network analogue," *Brit. J. Appl. Phys.*, vol. 1, no. 4, p. 92, 1950.

[5] X. Zhu *et al.*, "Observation of conductance quantization in oxide-based resistive switching memory," *Adv. Mater.*, vol. 24, no. 29, pp. 3941–3946, Aug. 2012.

[6] Z. Wang *et al.*, "Engineering incremental resistive switching in TaO_x based memristors for brain-inspired computing," *Nanoscale*, vol. 8, no. 29, pp. 14015–14022, 2016.

[7] M. Hu *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Adv. Mater.*, vol. 30, no. 9, Mar. 2018, Art. no. 1705914.

[8] Y. Kim, Y. Zhang, and P. Li, "A digital neuromorphic VLSI architecture with memristor crossbar synaptic array for machine learning," in *Proc. IEEE Int. SOC Conf.*, Sep. 2012, pp. 328–333.

[9] S. Liu, Y. Wang, M. Fardad, and P. K. Varshney, "A memristor-based optimization framework for artificial intelligence applications," *IEEE Circuits Syst. Mag.*, vol. 18, no. 1, pp. 29–44, Feb. 2018.

[10] E. Giacomin, T. Greenberg-Toledo, S. Kvatinsky, and P.-E. Gaillardon, "A robust digital RRAM-based convolutional block for low-power image processing and learning applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 2, pp. 643–654, Feb. 2019.

[11] Y. Zhou *et al.*, "Associative memory for image recovery with a high? Performance memristor array," *Adv. Funct. Mater.*, vol. 29, no. 30, 2019, Art. no. 1900155.

[12] M. R. Mahmoodi, M. Prezioso, and D. B. Strukov, "Versatile stochastic dot product circuits based on nonvolatile memories for high performance neurocomputing and neurooptimization," *Nature Commun.*, vol. 10, no. 1, pp. 1–10, Dec. 2019.

[13] F. Cai *et al.*, "Harnessing intrinsic noise in memristor hopfield neural networks for combinatorial optimization," 2019, *arXiv:1903.11194*. [Online]. Available: http://arxiv.org/abs/1903.11194

[14] Z. Sun, E. Ambrosi, G. Pedretti, A. Bricalli, and D. Ielmini, "In-memory PageRank accelerator with a cross-point array of resistive memories," *IEEE Trans. Electron Devices*, vol. 67, no. 4, pp. 1466–1470, Apr. 2020.

[15] S. Boyd, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2010.

[16] S. Wang and N. Shroff, "A new alternating direction method for linear programming," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1480–1488.

[17] R. Madani, A. Kalbat, and J. Lavaei, "A low-complexity parallelizable numerical algorithm for sparse semidefinite programming," *IEEE Trans. Control Netw. Syst.*, vol. 5, no. 4, pp. 1898–1909, Dec. 2018.

[18] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 2011.

[19] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, Oct. 2016.

[20] Z. Sun, G. Pedretti, E. Ambrosi, A. Bricalli, W. Wang, and D. Ielmini, "Solving matrix equations in one step with cross-point resistive arrays," *Proc. Nat. Acad. Sci. USA*, vol. 116, no. 10, pp. 4123–4128, Mar. 2019.

[21] A. Chen, "A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics," *IEEE Trans. Electron Devices*, vol. 60, no. 4, pp. 1318–1326, Apr. 2013.

[22] A. Nieuwoudt and Y. Massoud, "Evaluating the impact of resistance in carbon nanotube bundles for VLSI interconnect using diameter-dependent modeling techniques," *IEEE Trans. Electron Devices*, vol. 53, no. 10, pp. 2460–2466, Oct. 2006.

[23] H. Wang, H. Li, and R. E. Pino, "Memristor-based synapse design and training scheme for neuromorphic computing architecture," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jun. 2012, pp. 1–5.

[24] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: Variation-aware training for memristor X-bar," in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, 2015, pp. 1–6.

[25] A. Grossi *et al.*, "Experimental investigation of 4-kb RRAM arrays programming conditions suitable for TCAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, pp. 2599–2607, Dec. 2018.

[26] C. Pan and A. Naeemi, "Interconnect design and benchmarking for charge-based beyond-CMOS device proposals," *IEEE Electron Device Lett.*, vol. 37, no. 4, pp. 508–511, Apr. 2016.

[27] F. Zhang and M. Hu, "Mitigate parasitic resistance in resistive crossbar-based convolutional neural networks," 2019, *arXiv:1912.08716*. [Online]. Available: http://arxiv.org/abs/1912.08716

[28] C. Pan and A. Naeemi, "Beyond-CMOS device and interconnect technology benchmarking based on a fast cross-layer optimization methodology," *ECS Trans.*, vol. 72, no. 3, p. 93, 2016.

[29] H. Fujiwara *et al.*, "A 64 kb 16 nm asynchronous disturb current free 2-port SRAM with PMOS pass-gates for FinFET technologies," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 1–3.

[30] R. Madani, J. Lavaei, and R. Baldick, "Constraint screening for security analysis of power networks," *IEEE Trans. Power Syst.*, vol. 32, no. 3, pp. 1828–1838, May 2017.

[31] R. D. Zimmerman and C. E. Murillo-Sanchez. (2019). *MATPOWER (Version 7.0)*. [Online]. Available: https://matpower.org