Nano Communication Networks 18 (2018) 62–71

Contents lists available at ScienceDirect

Nano Communication Networks

journal homepage: www.elsevier.com/locate/nanocomnet

Review article

A low-computation-complexity, energy-efficient, and high-performance linear program solver based on primal–dual interior point method using memristor crossbars

Ruizhe Cai *, Ao Ren, Sucheta Soundarajan, Yanzhi Wang

College of Engineering and Computer Science, Syracuse University, Syracuse, NY, 13210, United States

ARTICLE INFO

Article history:
Received 12 June 2017
Received in revised form 2 October 2017
Accepted 14 January 2018
Available online 31 January 2018

Keywords:
Memristor
Memristor crossbar
Linear programming
Primal–dual interior point method

ABSTRACT

Linear programming is required in a wide variety of application including routing, scheduling, and various optimization problems. The primal–dual interior point (PDIP) method is state-of-the-art algorithm for solving linear programs, and can be decomposed to matrix–vector multiplication and solving systems of linear equations, both of which can be conducted by the emerging memristor crossbar technique in $O(1)$ time complexity in the analog domain. This work is the first to apply memristor crossbar for linear program solving based on the PDIP method, which has been reformulated for memristor crossbars to compute in the analog domain. The proposed linear program solver can overcome limitations of memristor crossbars such as supporting only non-negative coefficients, and has been extended for higher scalability. The proposed solver is iterative and achieves $O(N)$ computation complexity in each iteration. Experimental results demonstrate that reliable performance with high accuracy can be achieved under process variations.

Contents

1. Introduction

Linear programs are common in a wide variety of applications, including routing, scheduling, and other optimization problems.

* Corresponding author.
E-mail addresses: rcai100@syr.edu (R. Cai), aren@syr.edu (A. Ren), susounda@syr.edu (S. Soundarajan), ywang393@syr.edu (Y. Wang).

https://doi.org/10.1016/j.nancom.2018.01.001
1878-7789/© 2018 Elsevier B.V. All rights reserved.

Interior point methods are a popular class of algorithms for solving linear programs. Unlike the well-known simplex algorithm [1], which traverses vertices of the feasible region to find the optimal solution, interior point methods trace a path through the interior of the feasible region. The primal–dual interior point (PDIP) method uses the gap between the current solutions of the primal linear program and its dual in order to determine the path to follow within the feasible region. In each iteration, the algorithm involves calculating matrix–vector product and solving systems of linear equations. The emerging memristor crossbar technology can be potentially utilized to achieve significant speed-ups due to its significant benefits in matrix operations.

Memristor was predicted as the fourth circuit element nearly half a century ago [2]. It has been investigated for many years for its switching behavior in memory design field and analog computing field [3–5]. Non-volatility, low power consumption, and excellent scalability are some of its promising features. More importantly, its capability to record historical resistance makes it unique, and has resulted in heightened interests over the last several years. A crossbar structure of memristor devices (i.e. a memristor crossbar) can be utilized to perform matrix–vector multiplication and solve systems of linear equations in the analog domain in $O(1)$ time complexity [6–8]. Such advantages in matrix operations make it ideal candidate for implementing the state-of-the-art PDIP method for solving linear programs given its high usage of matrix–vector multiplication and solving linear systems. Moreover, experimental results suggest that the effect of process variations of memristor devices can be significantly mitigated by the inherent noise tolerance of the iterative PDIP algorithm.

Although promising, multiple challenges need to be overcome when applying memristor devices for linear program solving. Since the memristor crossbar performs matrix operations in the analog domain, we need to formulate the whole PDIP algorithm using memristor crossbar in the analog domain in order to avoid the significant overhead of D/A and A/D conversions. Moreover, some limitations of memristor crossbars (e.g., only non-negative matrix coefficients and square matrices when solving a system of linear equations can be supported) need to be properly addressed.

To the best of our knowledge, this paper provides a comprehensive algorithm-hardware framework on memristor crossbar for linear program solving. The PDIP method is reformulated for memristor crossbar and analog computations. The proposed solver can effectively deal with matrices containing negative numbers, and has been extended for linear program solving with higher scalability that can overcome size limitations of the memristor crossbar structure. The proposed solver achieves pseudo-$O(N)$ computation complexity, i.e., $O(N)$ complexity in each iteration, which is a significant improvement compared with the software-based PDIP method of $O(N^3)$. Experimental results demonstrate that the performance of proposed implementation is reliable with less than 4% inaccuracy on average under 10% process variations. Based on our estimation, the proposed solver could lead to an average of 80× improvements in speed and 270× reduction in energy consumption [9].

The rest of this paper is organized as follows: Section 2 describes the background of linear program solving methods and memristor devices. Section 3 presents a memristor crossbar based linear program solver as well as extensions to large-scale applications. Experimental results and discussions are provided in Section 4. Finally we conclude in Section 5.

2. Background

2.1. Existing methods for linear programming

The simplex method of Dantzig was the first efficient algorithm for solving linear programming problems, and is still popular today [10]. The simplex algorithm considers the feasible region of the linear program (i.e., the space of points satisfying all constraints), which is a polytope. The algorithm begins at one vertex of the polytope, and moves from vertex to vertex in such a way as to increase the value of the objective function. The simplex algorithm is extremely efficient in practice, but has exponential running time in the worst case [11].

Interior point methods for solving linear programs were developed in response to this inefficiency. Unlike the simplex algorithm, which moves from vertex to vertex of the feasible region, interior point methods traverse the interior of this region. Karmarkar's projective method was the first interior point algorithm that was both polynomial time in the worst case as well as fast in practice [11]. This method first begins at an interior point within the feasible region. It next applies a projective transformation so that the current interior point is the center of the projective space, and then moves in the direction of steepest descent. This is repeated until convergence.

The primal–dual interior point method uses the above technique, but incorporates information from the dual of the problem. Every linear program has a dual program, with the property that when the primal linear program has an optimal solution, the dual linear program also has the same optimal solution, and these two solutions are equal. The primal–dual interior point method exploits this property by simultaneously solving both the primal linear program as well as its dual, and steadily decreasing the duality gap (i.e., the difference between the value of the current solution to the primal and the current solution to the dual).

2.2. Memristor

Introduced as the fourth element of circuit by L.O. Chua in 1971, founded by HP labs in 2008 [2,3], memristor have shown its advantage and necessity with its memristive features. A memristor could memorize its most recent resistance that can be altered from excitation with voltage greater than a threshold. Given its memristive property, memristor can be used to design non-volatile memory. Furthermore, memristor crossbar structure has much advantage for matrix–vector multiplication.

When Maxwell published his famous theory unified electricity and magnetism, it was made clear that each of three fundamental circuit elements(capacitor, resistor, inductor) is a consequences of differential relation between two of four circuit variables(voltage, current, charge, flux). However, these 4 variables lead to 6 possible combinations. With 3 defined elements, current being the derivative of charge and flux as the integral of voltage, Chua noticed that one relation remained undefined. In 1971, Chua proposed a fourth element bridging the relation between charge and flux, for the sake of completeness. The new element, memristor is defined as:

$$M\left(q\right)=\frac{d\Phi}{dq} \tag{1}$$

which is equal to:

$$M\left(q\left(t\right)\right)=\frac{\frac{d\Phi}{dt}}{\frac{dq}{dt}}=\frac{V\left(t\right)}{I\left(t\right)} \tag{2}$$

The above equation looks like a nonlinear version of ohm's law. It was generalized to memristive systems by Chua in 1976 and can be written as:

$$V(t)=M(x,t)\cdot I(t) \tag{3}$$

where memristance is a time dependent and state dependent variable. This property of time dependence provides the memory of the system.

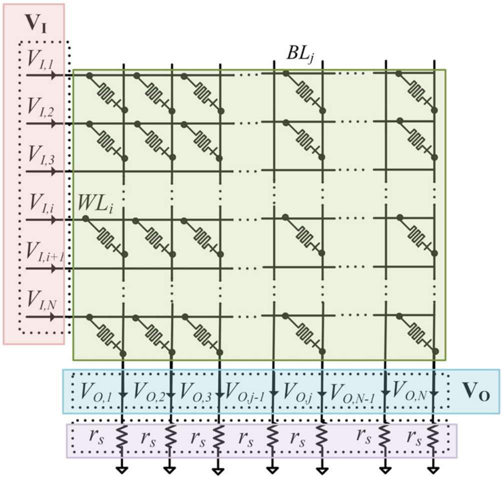


Fig. 1. Structure of memristor crossbar.

In 2008, HP labs announced that they have developed memristor based on titanium dioxide thin film and modeled the newly discovered element as follow:

$$M(q(t)) = R_{OFF} \cdot (1 - \frac{\mu_\upsilon \cdot R_{ON}}{D^2} q(t)) \tag{4}$$

R_{OFF} represents high resistance state, R_{ON} represents low resistance state, D represents film thickness and μ_υ represents mobility if dopants [12–15].

2.3. Memristor crossbar

As mentioned above, the state of a memristor will change when certain voltage higher than the threshold voltage, i.e., $|V_m| > |V_{th}|$, is applied at its two terminals for a small time period. Otherwise, the memristor behaves like a resistor. Such memristive property makes it an ideal candidate for non-volatile memory and matrix computations [6,7].

With its high degree of parallelism, the memristor crossbar array is attractive for matrix computations (which can often be performed with $O(1)$ time complexity). A typical structure of an $N \times N$ memristor crossbar is shown in Fig. 1, in which a memristor is connected between each pair of horizontal word-line (**WL**) and vertical bit-line (**BL**). This structure could provide large number of signal connections within a small footprint. In addition, it is capable of reprogramming each memristor to different resistance states by properly applying biasing voltages at its two terminals [8,16,17].

For multiplications, a vector of input voltages $\mathbf{V_I}$ is applied on **WL**s and the current through each **BL** can be collected by measuring the voltage across resistor R_s with conductance of g_s. Assume that the memristor at the connection between WL_i and BL_j has a conductance of $g_{(i,j)}$. Then the output voltages are represented by $\mathbf{V_O} = \mathbf{C} \cdot \mathbf{V_I}$, where the connection matrix $\mathbf{C}$ is constructed by a programmed crossbar array, which transfers the input vector $\mathbf{V_I}$ to the output vector $\mathbf{V_O}$. $\mathbf{C}$ is determined by the conductance of memristors as follows:

$$\mathbf{C} = \mathbf{D} \cdot \mathbf{G}^{\mathrm{T}} = diag(d_1, \ldots, d_N) \cdot \begin{bmatrix} g_{1,1} & \cdots & g_{1,N} \\ \vdots & \ddots & \vdots \\ g_{N,1} & \cdots & g_{N,N} \end{bmatrix} \tag{5}$$

where $d_i = 1/(g_s + \sum_{k=1}^{N} g_{k,i})$.

In reverse, the memristor crossbar structure can also be used to solve a linear system of equations, by mapping the linear equations to the memristor crossbar structure. A voltage vector $\mathbf{V_O}$ is applied on each R_s of **BL**, so the current flowing through each **BL** can be approximated as $I_{o,j} = g_s V_{o,j}$. On the other hand, current $I_{o,j}$ through BL$_j$ can also be calculated as $I_{o,j} = \sum_j V_{I,i} g_{i,j}$. Hence, for each BL$_j$, equation $\frac{1}{g_s}\sum_j V_{I,i} g_{i,j} = V_{o,j}$ is mapped. Therefore, the system of linear equations $\mathbf{C} \cdot \mathbf{V_I} = \mathbf{V_O}$ is mapped to the memristor crossbar structure, and solution $\mathbf{V_I}$ can be determined by measuring voltages on the **WL**s. Please note that, elements of matrix **C** should be non-negative in order to be mapped to memristor crossbar, because resistance cannot reach negative values. It is worth mentioning that the matrix calculation process with the memristor crossbar just has a negligible effect on memristance of each memristor, because the time period that current go through a memristor is short enough during the calculation process.

It is proved in [8] that a fast and simple approximation can be adopted for mapping above matrix onto the memristor crossbar (g_{max} is the largest value in **G**). Therefore for matrix–vector multiplication $\boldsymbol{Ax = b}$, and $\mathbf{b} = g_s \mathbf{V_O}$; for the solution of linear system $\boldsymbol{Ax = b}$, and $\mathbf{x} = \frac{g_s}{g_{max}} \mathbf{V_I}$.

3. Memristor crossbar-based solver for linear programs

We present a memristor crossbar based linear program solver based on the PDIP algorithm, which overcomes hardware limitations of memristor crossbar while taking its advantages on matrix operations. The presented solver could handle the vastly used matrix operations in PDIP algorithm efficiently with significantly reduced computation complexity (to pseudo-O(N)), power consumption, and latency. Moreover, the proposed solver can deal with matrices containing negative numbers that cannot be directly mapped on to memristor crossbars. In addition, we introduce an extension for linear program solving with higher scalability that can overcome size limitations of the memristor crossbar structure.

This section is organized in five parts: The PDIP algorithm is discussed in part 3.1; The proposed memristor crossbar-based linear program solver is introduced in part 3.2; Part 3.3 discusses writing coefficients in memristor crossbar, and the proposed solutions for representing and computing large-scale matrices are introduced in part 3.4. Part 3.5 investigates computation complexity of proposed memristor crossbar-based solvers.

3.1. The primal–dual interior point (PDIP) method for solving linear programs

Linear programs or linear programming problems [18] are problems that can be expressed as:

Maximize $\mathbf{c^T x}$ subject to: $\mathbf{Ax} \preceq \mathbf{b}(\mathbf{A} \in \mathbb{R}^{m \times n}), \mathbf{x} \succeq \mathbf{0}$

where $\mathbf{Ax} \preceq \mathbf{b}$ means that every element in the vector $\mathbf{Ax}$ is smaller than or equal to the corresponding element in vector $\mathbf{b}$. Every linear program can be converted into a symmetrical dual problem:

Minimize $\mathbf{b^T y}$ subject to: $\mathbf{A^T y} \succeq \mathbf{c}(\mathbf{A} \in \mathbb{R}^{m \times n}), \mathbf{y} \succeq \mathbf{0}$

By Introducing two additional variables, inequality constraints can be transformed into equality constraints. The above problem can be reformulated as follows [19]:

Maximize $\mathbf{c^T x}$ subject to:

$$\mathbf{Ax} + \mathbf{w} = \mathbf{b} \quad \mathbf{x}, \mathbf{w} \succeq \mathbf{0} \tag{6a}$$

and its dual:

Minimize $\mathbf{b^T y}$ subject to:

$$\mathbf{A^T y} + \mathbf{z} = \mathbf{c} \quad \mathbf{y}, \mathbf{z} \succeq \mathbf{0} \tag{6b}$$

with complementary conditions:

$$\forall i \in \mathbb{N} \cap [1, n] \wedge \forall j \in \mathbb{N} \cap [1, m] : x_i z_i = 0, \;\; y_j w_j = 0$$

which can be represented using the following matrix notations:

$$\mathbf{XZ_e} = \mathbf{0}\ ,\ \mathbf{YW_e} = \mathbf{0} \tag{6c}$$

In the above equation, uppercase notations are utilized to denote diagonal matrices, e.g.,

$$\mathbf{X} = diag\left(x_1,\ \ldots,\ x_n\right),$$

where $\mathbf{x} = [x_1,\ \ldots,\ x_n]^{\mathbf{T}}$, and the subscript e stands for the reverse operation, that is

$$\mathbf{X_e} = [\mathbf{X}_{11},\ \ldots,\ \mathbf{X_{ii}},\ \ldots,\ \mathbf{X_{nn}}]^{\mathbf{T}}.$$

Due to nonlinearity characteristics in (6c), the above problem is difficult to solve directly. The interior point algorithm [18,19] is introduced to solve this problem effectively. In this algorithm, $\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$, $\mathbf{z}$ are initialized as arbitrary vectors and updated iteratively until Eqs. (6a)–(6c) are (sufficiently) satisfied. In each iteration, a set of vectors $\Delta\mathbf{x}$, $\Delta\mathbf{y}$, $\Delta\mathbf{w}$, $\Delta\mathbf{z}$, which are referred to as step direction vectors, are derived from solving the following system of equations:

$$\mathbf{A}\left(\mathbf{x} + \Delta\mathbf{x}\right) + \left(\mathbf{w} + \Delta\mathbf{w}\right) = \mathbf{b} \tag{7a}$$

$$\mathbf{A}^{\mathbf{T}}\left(\mathbf{y} + \Delta\mathbf{y}\right) - \left(\mathbf{z} + \Delta\mathbf{z}\right) = \mathbf{c} \tag{7b}$$

$$\left(\mathbf{X} + \Delta\mathbf{X}\right)\left(\mathbf{Z} + \Delta\mathbf{Z}\right)_{\mathbf{e}} = \boldsymbol{\mu} \tag{7c}$$

$$\left(\mathbf{Y} + \Delta\mathbf{Y}\right)\left(\mathbf{W} + \Delta\mathbf{W}\right)_{\mathbf{e}} = \boldsymbol{\mu} \tag{7d}$$

where $\boldsymbol{\mu}$ is a small value vector, values of all of its elements are equal to μ. μ is a important parameter to guarantee that every step of iteration does follow the correct path to the optimal solution. If chosen too large, then the sequence could converge to the center of feasible region. However, a too small μ could force algorithm jam into the boundary of feasible region. It has been suggested that:

$$\mu = \delta\frac{\mathbf{z^Tx} + \mathbf{y^Tw}}{n+m} \tag{8}$$

where δ is a number between zero and one

Since $\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$, $\mathbf{z}$ are nonnegative vectors, the complementary conditions in (6c) are replaced with $\boldsymbol{\mu}$-complementary conditions (7c) and (7d). Ignoring the second-order elements in (7c) and (7d), the above system of equations can be represented as a system of linear equations of $\Delta\mathbf{x}$, $\Delta\mathbf{y}$, $\Delta\mathbf{w}$, $\Delta\mathbf{z}$, denoted by:

$$\mathbf{A}\Delta\mathbf{x} + \Delta\mathbf{w} = \mathbf{b} - \mathbf{Ax} - \mathbf{w} \tag{9a}$$

$$\mathbf{A^T}\Delta\mathbf{y} - \Delta\mathbf{z} = \mathbf{c} -\ \mathbf{A^T y} + \mathbf{z} \tag{9b}$$

$$\mathbf{Z}\Delta\mathbf{x} + \mathbf{X}\Delta\mathbf{z} = \boldsymbol{\mu} - \mathbf{XZ_e} \tag{9c}$$

$$\mathbf{W}\Delta\mathbf{y} + \mathbf{Y}\Delta\mathbf{w} = \boldsymbol{\mu} - \mathbf{YW_e} \tag{9d}$$

The unknown vectors $\Delta\mathbf{x}$, $\Delta\mathbf{y}$, $\Delta\mathbf{w}$, $\Delta\mathbf{z}$, or step directions, can be solved from solving the system of linear equations (9a)–(9d) and applied to update $\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$, $\mathbf{z}$. For each iteration, $\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$, $\mathbf{z}$ are updated with step directions determined from solving (9a)–(9d). In order to guarantee the positive property of every primal and dual variable, a step length parameter θ is used to limit the impact of step directions.

$$\mathbf{x} = \mathbf{x} + \theta\Delta\mathbf{x} \tag{10a}$$

$$\mathbf{y} = \mathbf{y} + \theta\Delta\mathbf{y} \tag{10b}$$

$$\mathbf{w} = \mathbf{w} + \theta\Delta\mathbf{w} \tag{10c}$$

$$\mathbf{z} = \mathbf{z} + \theta\Delta\mathbf{z} \tag{10d}$$

and θ must satisfy $\theta\ \le\ \max_{i,j}\left(-\frac{\Delta x_j}{x_j},\ -\frac{\Delta y_i}{y_i},\ -\frac{\Delta w_j}{w_j},\ -\frac{\Delta z_j}{z_j}\right)$. In addition, solution of (10a)–(10d) are determine under the assumption that the step length direction is equal to one, hence, $\theta\ \le\ 1$. Therefore, we have:

$$\theta = r\cdot\min(\max_{i,j}\left(-\frac{\Delta x_j}{x_j},\ -\frac{\Delta y_i}{y_i},\ -\frac{\Delta w_j}{w_j},-\frac{\Delta z_j}{z_j}\right)^{-1},1) \tag{11}$$

where r, a parameter whose value is less than but close to 1, is introduced to guarantee strict inequality. Above steps are repeated until primal infeasibility ($\mathbf{Ax}+\mathbf{w}-\mathbf{b}$), dual infeasibility ($\mathbf{A^Ty}-\mathbf{z}-\mathbf{c}$) and duality gap ($\mathbf{z^Tx} + \mathbf{y^Tw}$) are small enough. It is proven that unbound dual indicates primal being infeasible and vice versa, therefore, constraints are infeasible if the element with the largest absolute value in $\mathbf{x}$, $\mathbf{y}$ is greater than a certain enough large number, such property could applies to each iteration.

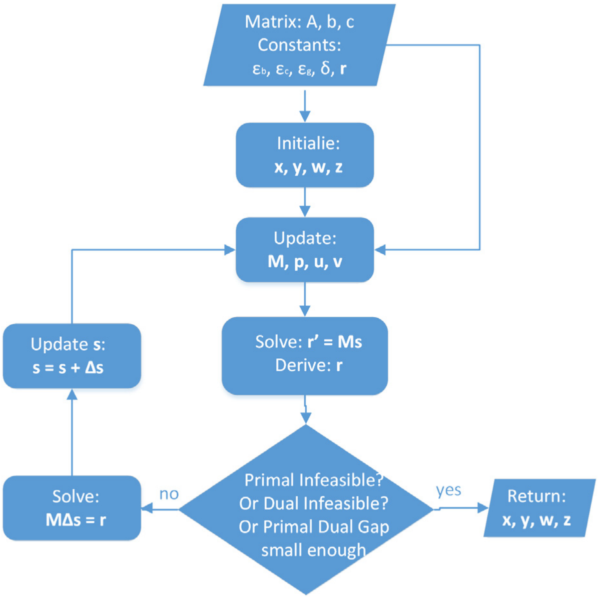


Fig. 2. Control flow graph for proposed memristor crossbar-based PDIP linear program solver.

3.2. Memristor crossbar-based linear program solver using PDIP algorithm

The memristor crossbar array structure has high potential for implementing PDIP algorithms due to its advantages in matrix operations. However, the memristor crossbar array structure has some limitations, which necessitate the adjustment of PDIP algorithm for effective memristor crossbar based implementations. Since the matrix elements are represented as non-negative memristance values in the memristor crossbar, a novel mechanism is required for representing negative matrix coefficients. In addition, the linear system to be solved should have a square coefficients matrix. Next, we propose a memristor crossbar based linear program solver using PDIP algorithm through effectively resolving the above mentioned issues. (See Fig. 2.)

For facilitating memristor-based implementations, linear equations in (9a)–(9b) can be rewritten as a linear system with $2(n+m)$ variables, as shown in (12):

$$\begin{bmatrix} \mathbf{A} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{A^T} & \mathbf{0} & -\mathbf{I} \\ \mathbf{Z} & \mathbf{0} & \mathbf{0} & \mathbf{X} \\ \mathbf{0} & \mathbf{W} & \mathbf{Y} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{y} \\ \Delta\mathbf{w} \\ \Delta\mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{b} - \mathbf{Ax} - \mathbf{w} \\ \mathbf{c} -\ \mathbf{A^Ty} + \mathbf{z} \\ \boldsymbol{\mu} - \mathbf{XZ_e} \\ \boldsymbol{\mu} - \mathbf{YW_e} \end{bmatrix} \tag{12}$$

where **I** represents the identity matrix with diagonal values equal to 1.

In order to make the matrix representable in memristor crossbar structure, new variables have to be introduced to eliminate negative elements. Consider a linear system $\mathbf{Ax} = \mathbf{b}$, in which $\mathbf{A_{i,j}}$ is negative element. It can be transformed into a non-negative matrix by introducing a compensation variable $x_c = -x_j$. Hence, the above linear system is equivalent to:

$$\begin{bmatrix} \mathbf{A}_{1,1} & \dots & \mathbf{A}_{1,\mathbf{j}} & \dots & \mathbf{A}_{1,\mathbf{n}} & 0 \\ \vdots & \dots & \dots & \dots & \vdots & 0 \\ \mathbf{A}_{\mathbf{i},1} & \dots & 0 & \dots & \mathbf{A}_{\mathbf{i},\mathbf{n}} & -\mathbf{A}_{\mathbf{i},\mathbf{j}} \\ \vdots & \dots & \dots & \dots & \vdots & 0 \\ \mathbf{A}_{\mathbf{n},1} & \dots & \mathbf{A}_{\mathbf{n},\mathbf{j}} & \dots & \mathbf{A}_{\mathbf{n},\mathbf{n}} & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x_j} \\ \vdots \\ \mathbf{x_n} \\ \mathbf{x_c} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b_j} \\ \vdots \\ \mathbf{b_n} \\ 0 \end{bmatrix} \tag{13}$$

As shown in Eq. (12), the matrix on left hand-side consists of a sub-matrix $-\mathbf{I}$ introduced by Δz in Eq. (7b). A new variable vector, $\Delta\mathbf{v} = -\Delta\mathbf{z}$, has to be introduced. Besides, a compensation variable vector $\Delta\mathbf{u} = -\Delta\mathbf{w}$ is required for maintaining a square matrix. In addition, **A** and $\mathbf{A^T}$ are the only matrices that may contain negative elements. Processes like Eq. (13) are needed to eliminate all negative elements in **A** and $\mathbf{A^T}$. Therefore the left hand side of the linear system to be mapped can be written as:

$$\begin{bmatrix} \mathbf{A}' & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \text{ or } \mathbf{A}'' \\ \mathbf{0} & \mathbf{A}^{\mathbf{T}'} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \text{ or } \mathbf{A}^{\mathbf{T}''} \\ \mathbf{Z} & \mathbf{0} & \mathbf{0} & \mathbf{X} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W} & \mathbf{Y} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} \text{ or } \mathbf{A}^{\mathbf{I}} & \mathbf{0} \text{ or } \mathbf{A}^{\mathbf{TI}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \text{ or } \mathbf{I} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{y} \\ \Delta\mathbf{w} \\ \Delta\mathbf{z} \\ \Delta\mathbf{u} \\ \Delta\mathbf{v} \\ \Delta\mathbf{p} \end{bmatrix} \tag{14a}$$

where $\Delta\mathbf{p}$ comprises $\Delta p_i = \begin{cases} -\Delta x_j & \text{if } A_{\alpha,j} < 0 \text{ for some } \alpha \\ -\Delta y_k & \text{if } A^T_{\beta,k} < 0 \text{ for some } \beta \end{cases}$. $\mathbf{A}'$ and $\mathbf{A}^{\mathbf{T}'}$ are matrices that change the negative elements in **A** and $\mathbf{A^T}$ to zero. $\mathbf{A}''$ and $\mathbf{A}^{\mathbf{T}''}$ are matrices whose elements are the absolute values of negative elements in **A** and $\mathbf{A^T}$. $\mathbf{A^I}$ and $\mathbf{A^{TI}}$ are matrices consisting of 1 and 0's. Locations of 1's depend on the locations of negative elements in **A** and $\mathbf{A^T}$ (please refer to Eq. (13) as an example).

The equation with above left hand size can be denoted as:

$$\mathbf{M}\Delta\mathbf{s} = \mathbf{r} \tag{14b}$$

where **M** can be implemented and variable vector $\Delta\mathbf{s}$ can be derived using memristor crossbar.

In the PDIP algorithm, once $\Delta\mathbf{x}$, $\Delta\mathbf{y}$, $\Delta\mathbf{w}$, $\Delta\mathbf{z}$ (all in the derived vector $\Delta\mathbf{s}$) are derived, we will update x, y, w, z, which can be performed using summing amplifiers. We will further update the left-hand side matrix **M** and the right hand-side vector r of Eq. (14b). Updating matrix **M** is relatively straightforward since we only need to update **X**, **Y**, **Z**, and **W** in **M**, using the memristor writing technology as shall be discussed in part C. On the other hand, r can be viewed as the difference of two vectors:

$$\mathbf{r} = \begin{bmatrix} \mathbf{b} - \mathbf{Ax} - \mathbf{w} \\ \mathbf{c} - \mathbf{A^T y} + \mathbf{z} \\ \mu - \mathbf{XZ_e} \\ \mu - \mathbf{YW_e} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \\ \mu \\ \mu \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{Ax} + \mathbf{w} \\ \mathbf{A^T y} - \mathbf{z} \\ \mathbf{XZ_e} \\ \mathbf{YW_e} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{15a}$$

The subtraction could be implemented using summing amplifiers [12]. Next, we will discuss the calculation of the last vector in Eq. (14a). Note that

$$\mathbf{M} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{w} \\ \mathbf{z} \\ \mathbf{u} \\ \mathbf{v} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{Ax} + \mathbf{w} \\ \mathbf{A^T y} - \mathbf{z} \\ 2\mathbf{XZ_e} \\ 2\mathbf{YW_e} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{15b}$$

where $\mathbf{u} = -\mathbf{w}$, $\mathbf{v} = -\mathbf{z}$, and **p** consists of elements whose value are negative of some elements of **x** and **y**, depending on the location of negative elements in **A** and $\mathbf{A^T}$. The result of Eq. (15b) is only slightly different from the last vector in Eq. (15a) on the 3rd and 4th elements. Since the matrix–vector product in memristor crossbar is represented as voltage, we can first calculate (15b) by performing matrix–vector multiplication using the updated memristor crossbar **M**, and then acquire the last vector in Eq. (15a), using a simple dividing-by-2 procedure on corresponding elements. **r** can be updated accordingly.

Unlike PDIP method under ideal conditions, implementation using memristor crossbar could suffer from hardware process variation, which could alter actual resistance of each memristor (will be explained briefly in chapter 4). In some cases, process variation could severely affect constraints and feasible region accordingly. Thus, a more robust feasibility detection technique is required to guarantee an optimal solution is given. In addition to primal–dual unbound check, a constraints check should be given to the solution to determine its feasibility. Theoretically, all optimal solutions should satisfy $\mathbf{Ax} \leq \mathbf{b}$. However, consider impact form process variation, there is no strict guarantee that above equation should stand. Therefore, condition $\mathbf{Ax} \leq \alpha\mathbf{b}$ is used to check constraints satisfaction at the end of the algorithm, where α is a parameter close but greater than 1.

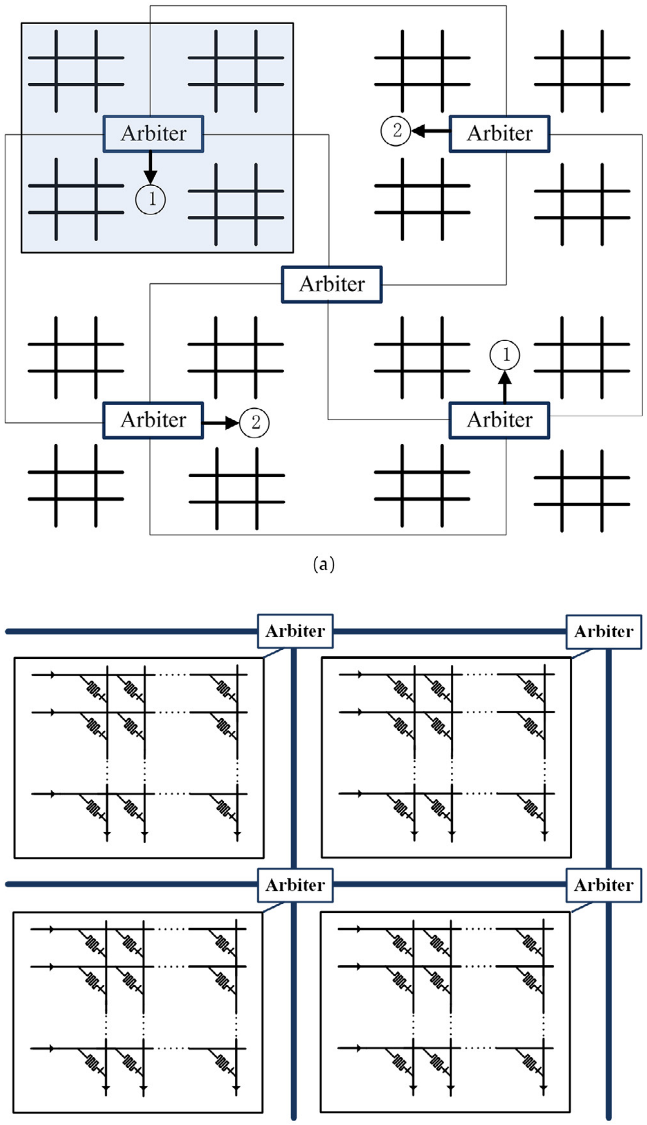
Our proposed memristor crossbar-based linear program solver is summarized as follows:

Algorithm 1: Memristor Cross-bar Linear Pram Solver

```
Input: Matrix M, vectors b, c, constants $\epsilon_b, \epsilon_c, \epsilon_g, \delta, \theta$
Output: Vectors x, y, z, w
while $\mathbf{Ax} + \mathbf{w} - \mathbf{b} > \epsilon_b$ or $\mathbf{A^T y} + \mathbf{z} - \mathbf{c} > \epsilon_c$ or $\mathbf{z^T x} + \mathbf{y^T w} > \epsilon_g$ do
    Update coefficient matrix M in matrix crossbar based on A, x, y, z, w.
    Derive $r$ using memristor cross-bar.
    Solve $\mathbf{M}\Delta\mathbf{s} = \mathbf{r}$ using memristor crossbar.
    Update $\mathbf{s} = \mathbf{s} + \theta\Delta\mathbf{s}$.
    Update $\mu$.
end
Return x, y, z, w.
```

3.3. *Writing coefficients in matrices*

The analog computation requires that memristor arrays (e.g., matrix M in (11b)) be programmed prior to execution (solving linear program), and be updated in each iteration during execution. Modifying the resistance of a memristor device can be achieved by applying V_{dd} or $-V_{dd}$ (satisfying $|V_{dd}| > |V_{th}|$) to two terminals of the memristor device [8,16,17]. In a memristor crossbar, the voltage difference V_{dd} is applied on the corresponding WL and BL that are connected to the target memristor device, whereas other WLs and BLs are biased by $V_{dd}/2$, which will have negligible effect on other memristor devices since $|V_{dd}/2| < |V_{th}|$ [2,17]. Programming a memristor device to a specific resistance is achieved by adjusting the amplitude and width of the write pulse (or the total number of write pulse spikes) [8,17]. The writing circuits of memristor crossbars and corresponding controlling circuits will be CMOS based.



(a)

(b)

Fig. 3. NoC structure for large scale computation.

3.4. Supporting large-scale matrices in solving linear programs

A memristor crossbar has limitation on its size due to manufacturing and performance considerations [20], which can potentially limit its scalability for large-scale and high-data rate applications. In order to overcome this shortcoming, motivated by [20], we adopt analog network-on-chip (NoC) communication structures that effectively coordinate multiple memristor crossbars for supporting large-scale applications. Data transfers within this NoC structure maintain analog form and are managed by the NoC arbiters.

Fig. 3(a) and (b) illustrate two potential analog NoC structures for multiple memristor crossbars. Fig. 3(a) is a hierarchical structure of memristor crossbars, in which four crossbar arrays are grouped and controlled by one arbiter, and four such groups again form a higher-level group controlled by a higher-level arbiter. Fig. 3(b) is a mesh network-based structure of memristor crossbars, which resembles the mesh network-based NoC structure in multi-core systems [20]. Analog buffer and switches [21] will be utilized (in the arbiters) for the proper operation of this structure. The controller of NoC structure will be implemented in CMOS circuits. The NoC structure in Fig. 3(a) will adopt a centralized controller whereas that in Fig. 3(b) could employ a distributed controller similar to mesh network-based NoC in multi-core systems [20].

In addition to the NoC structure, we also present a memristor-based linear program solver with enhanced scalability. They key motivation is to use an iterative process to reduce the required size of matrix M in (14b), thereby improving scalability. More specifically, we treat Eqs. (9a)–(9b) as two systems of linear equations:

$$\begin{bmatrix}\mathbf{A} & \mathbf{0}\\ \mathbf{0} & \mathbf{A}^{\mathbf{T}}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{y}\end{bmatrix}=\begin{bmatrix}\mathbf{b}-\mathbf{Ax}-\mathbf{w}\\ \mathbf{c}-\ \mathbf{A}^{\mathbf{T}}\mathbf{y}+\mathbf{z}\end{bmatrix} \tag{16a}$$

$$\begin{bmatrix}\mathbf{X} & \mathbf{0}\\ \mathbf{0} & \mathbf{Y}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{z}\\ \Delta\mathbf{w}\end{bmatrix}=\begin{bmatrix}\boldsymbol{\mu}-\mathbf{XZ_e}\\ \boldsymbol{\mu}-\mathbf{YW_e}\end{bmatrix} \tag{16b}$$

Unlike (14a) which solves all step direction vectors (i.e., $\Delta\mathbf{x}$, $\Delta\mathbf{y}$, $\Delta\mathbf{w}$, $\Delta\mathbf{z}$) as one linear system, the proposed iterative algorithm for large-scale operations updates the step directions in an iterative approach. While updating step directions for vector $\mathbf{x}$, $\mathbf{y}$, vectors $\mathbf{w}$, $\mathbf{z}$ are assumed to be fixed so that we only need to solve Eq. (14a) using memristor crossbar. After updating $\mathbf{x}$, $\mathbf{y}$, we derive the step directions for vectors $\mathbf{w}$, $\mathbf{z}$ by solving (14b) using memristor crossbar.

However, the coefficient matrix in (16a) is singular if $\mathbf{A}$ is not a square matrix, that is, Eq. (16a) has no solution. In order to make (16a) solvable, part of the zero elements needs to be transformed to nonzero elements while causing limited impact to solution. Hence, following change is made to (16a).

$$\begin{bmatrix}\mathbf{A} & \mathbf{R_U}\\ \mathbf{R_L} & \mathbf{A}^{\mathbf{T}}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{y}\end{bmatrix}=\begin{bmatrix}\mathbf{b}-\mathbf{Ax}-\mathbf{w}\\ \mathbf{c}-\ \mathbf{A}^{\mathbf{T}}\mathbf{y}+\mathbf{z}\end{bmatrix} \tag{16c}$$

where $\mathbf{R_U}$ is a matrix whose upper right m by m sub-matrix is a zero matrix and $\mathbf{R_L}$ is a matrix whose lower left n by n sub-matrix is a zero matrix while the values of the rest of their elements are very small. Above transformation could change the linear system represented by previous equations, however, based on our experiments, minor changes infused in coefficient (e.g. process variation), in very rare cases, could largely affect the accuracy of final optimal value. Given limited size of in $\mathbf{R_U}$ and $\mathbf{R_L}$ and their small constructing values, impact from introduction above two balancing matrices is slight. (See Fig. 4.)

If $n > m$, $\mathbf{R_L}$ is used to replace the submatrix containing lower left zero elements, and if $m > n$, $\mathbf{R_U}$ is used to replace submatrix containing upper right zero elements. Process alike Eq. (9) is still needed after this step; therefore, the left hand side of Eq. (16a) is transformed into

$$\begin{bmatrix}\mathbf{A}' & \mathbf{R_U} & \mathbf{0}\text{ or }\mathbf{A}''\\ \mathbf{R_L} & \mathbf{A}^{\mathbf{T}'} & \mathbf{0}\text{ or }\mathbf{A}^{\mathbf{T}''}\\ \mathbf{0}\text{ or }\mathbf{A}^{\mathbf{I}} & \mathbf{0}\text{ or }\mathbf{A}^{\mathbf{TI}} & \mathbf{0}\text{ or }\mathbf{I}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{y}\\ \Delta\mathbf{p}\end{bmatrix} \tag{16d}$$

On the other hand, the right hand-side vectors of Eqs. (16a) and (16b) can be calculated as:

$$\begin{bmatrix}\mathbf{b}-\mathbf{Ax}-\mathbf{w}\\ \mathbf{c}-\ \mathbf{A}^{\mathbf{T}}\mathbf{y}+\mathbf{z}\\ \mathbf{0}\end{bmatrix}=\begin{bmatrix}\mathbf{b}-\mathbf{w}\\ \mathbf{c}+\mathbf{z}\\ \mathbf{0}\end{bmatrix} - \begin{bmatrix}\mathbf{A}' & \mathbf{0} & \mathbf{0}\text{ or }\mathbf{A}''\\ \mathbf{0} & \mathbf{A}^{\mathbf{T}'} & \mathbf{0}\text{ or }\mathbf{A}^{\mathbf{T}''}\\ \mathbf{0}\text{ or }\mathbf{A}^{\mathbf{I}} & \mathbf{0}\text{ or }\mathbf{A}^{\mathbf{TI}} & \mathbf{0}\text{ or }\mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\\ \mathbf{p}\end{bmatrix} \tag{17a}$$

$$\begin{bmatrix}\boldsymbol{\mu}-\mathbf{XZ_e}\\ \boldsymbol{\mu}-\mathbf{YW_e}\end{bmatrix}=\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}\end{bmatrix}-\begin{bmatrix}\mathbf{X} & \mathbf{0}\\ \mathbf{0} & \mathbf{Y}\end{bmatrix}\begin{bmatrix}\mathbf{z}\\ \mathbf{w}\end{bmatrix} \tag{17b}$$

Updating $\boldsymbol{\mu}$ shall still follow Eq. (8). θ, on the other hand, were found to be better to be constant to guarantee convergence.

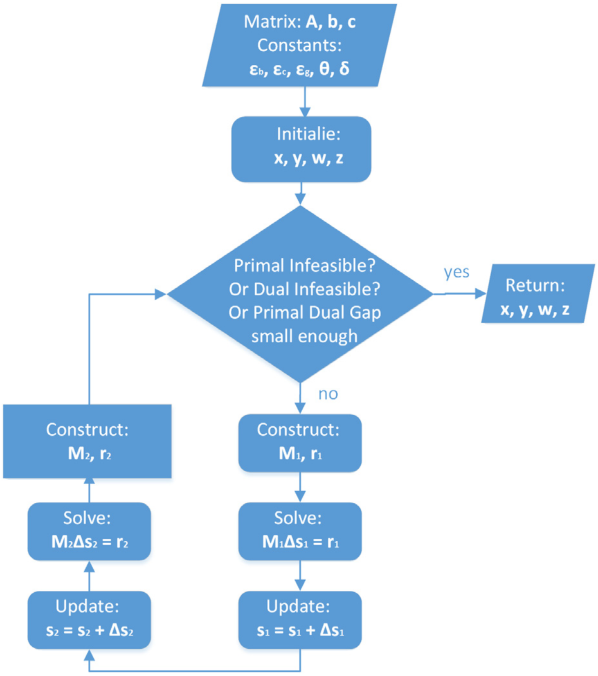


Fig. 4. Control flow graph for proposed memristor crossbar-based linear program solver for large-scale operations.

Even though a constant θ may not strict restraints primal or dual solution from being negative in some cases, optimal results can still approach acceptable accuracy.

Details of the proposed iterative linear program solver for enhancing scalability are described as below:

Algorithm 2: Memristor Crossbar Linear Program Solver for Large-Scale Operations

```
Input: Matrix $\mathbf{M}$, vectors $\mathbf{b}$, $\mathbf{c}$, constants $\epsilon_b, \epsilon_c, \epsilon_g, \delta, \theta$
Output: Vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w}$
Initialize $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w}$ with an arbitrary guess.
while $\mathbf{Ax} + \mathbf{w} - \mathbf{b} > \epsilon_b$ or $\mathbf{A^T y} + \mathbf{z} - \mathbf{c} > \epsilon_c$ or $\mathbf{z^T x} + \mathbf{y^T w} > \epsilon_g$ do
    Update coefficient matrix $\mathbf{M_1}$ in matrix crossbar based on $\mathbf{A}$, $\mathbf{x}$, $\mathbf{y}$.
    Calculate vector $\mathbf{r_1}$ based on $\mathbf{M_1}$ and $\mathbf{s_1}$ using memristor crossbar where $\mathbf{s_1} = [\mathbf{x}, \mathbf{y}, \mathbf{p}]^\mathbf{T}$.
    Solve $\mathbf{M_1} \Delta \mathbf{s_1} = \mathbf{r_1}$ using memristor crossbar.
    Update $\mathbf{s_1} = \mathbf{s_1} + \theta \Delta \mathbf{s_1}$.
    Update coefficient matrix $\mathbf{M_2}$ in matrix crossbar based on $\mathbf{x}$, $\mathbf{y}$.
    Calculate vector $\mathbf{r_2}$ based on $\mathbf{M_2}$ and $\mathbf{s_2}$ using memristor crossbar where $\mathbf{s_2} = [\mathbf{z}, \mathbf{w}]^\mathbf{T}$.
    Solve $\mathbf{M_2} \Delta \mathbf{s_2} = \mathbf{r_2}$ using memristor crossbar.
    Update $\mathbf{s_2} = \mathbf{s_2} + \theta \Delta \mathbf{s_2}$.
    Update $\mu$.
end
Return $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w}$.
```

3.5. *Algorithm complexity comparison*

Given the fact that iteration-exiting conditions are same in software-based PDIP algorithm and the proposed memristor crossbar-based solver, the difference in iteration times is minimal. For each iteration step in software-based PDIP algorithm, a set of $2(n+m)$ equations needs to be solved. Solving such linear system could require $O\left(N^3\right)$ time complexity with direct method such as Gaussian Elimination method or LU-Decomposition, and $O\left(N^2\right)$ for each iteration by using iterative method such as Gauss–Seidel method ($N = n + m$). For the proposed solver, complexity for updating X, Y, W, Z in matrix $\mathbf{M}$ is $O(N)$ (please note that matrices $\mathbf{A}$ and $\mathbf{A^T}$ do not need updating), and solving linear system in Eq. (14a) only costs $O(1)$ time complexity. That is, for each iteration the complexity for memristor crossbar-based linear program solver is $O(N)$, while software-based PDIP algorithm could cost at least pseudo-$O\left(N^2\right)$. As for memristor crossbar-based linear program solver for large-scale applications, complexity for updating X, Y in matrix Eq. (16b) is $O(N)$, and complexities for solving (16a) and (16b) on memristor crossbar are both $O(1)$. Hence, the time complexity for memristor crossbar-based linear program solver for large-scale applications is also $O(N)$ for each iteration step, and the overall time complexity is pseudo-$O(N)$.

Please note that the above analysis only applies for the iterative solution of linear programs. On the other hand, the initialization time complexity is $O\left(N^2\right)$ for dense matrices, and will be lower for sparse matrices that are common in linear programs.

4. Experiments and analysis

4.1. *Hardware process variation*

Under ideal condition, proposed two implementations shall provide accurate and optimized solution. However, hardware process variation could alter actual mem-resistance from designated mem-resistance, thus affect computation complexity and accuracy. Because the impact of process variations is too complex to be expressed by a mathematical closed-form solution, we model it as a uniform distribution with a maximum range.

The actual matrix represented by memristor crossbar is

$$\mathbf{M}' = \mathbf{M} + \mathbf{M} \circ (var \cdot \mathbf{Rd}) \tag{18}$$

where var is the maximum variation percentage, usually range from 5% to 20% [22], and $\mathbf{Rd}$ is a matrix whose elements are random number whose absolute values are less than one. All voltage inputs and outputs are stored with 8-bit precision.

4.2. *Experiments setup*

Experiments were given in Matlab 2015a a on a PC (Intel i7-6700, 16 GB RAM, Windows 10 Pro). Since size of the matrix could affect results, linear problems with different number of constraints were tested. The number of constraints varies from 256 to 1024 exponentially while the number of variables is one third of the number of constraints. 100 randomly generated feasible tests and 100 randomly generated infeasible tests were given for both two implementations under no process variation, up to 5% process variation, up to 10% process variation and up to 20% process variation. Results in optimal value and latency were compared to the results obtained from using Matlab linprog function. Following aspects were taken into consideration: Relative error, number of iterations, and number of iterations for detecting infeasibility, speed and energy efficiency.

4.3. *Accuracy*

Under ideal condition, matrix operations on memristor crossbar-based design should be accurate given Kirchhoff's law [11]. However, as mentioned above, due to process variations, the actual memristance matrix of a memristor crossbar may be different from the theoretical values. Optimal values are used to compare with results from Matlab function to calculate relative errors. Accuracy tests results are shown in Fig. 5.

For tested process variation the inaccuracy range is 0.2% to 9.9% for Memristor Crossbar-based Linear Program Solver and 0.8% to

8.5% for Memristor Crossbar-based Linear Program Solver for Large Scale Operation. Inaccuracy decreases with increasing of numbers of constraints. Both implementations have shown reliable and accurate performance. Even for up to 20% process variation, relative error can be as low as 1%.

Relative error being immune to even up to 20% process variation is a surprising result. To investigate this, we tested Matlab linprog function with matrices with process variation. To our surprise, relative error is similar to what we get from PDIP solver simulation. It can be concluded that, linear program are not affected by process variation too much, the larger the size, the less impact process variation could result.

Another observation is that memristor crossbar-based linear program solver for large-scale operations is very reliable in terms of accuracy. However, due to the constant step length and non-singular system by introducing noise, implementation for large-scale operations may fail to converge in some rare cases. We believe that some singular matrices induced by process variations in the intermediate steps may cause such steep drop. While memristance is altered under the impact of process variation, its mapping matrix might be changed from a non-singular matrix to closer to a singular matrix (with determinant equal to 0), which could lead to zero solution or less accurate solution for the linear system. Since the coefficient size is relatively small, it could be more easily affected by some elements change and turn into a singular matrix.

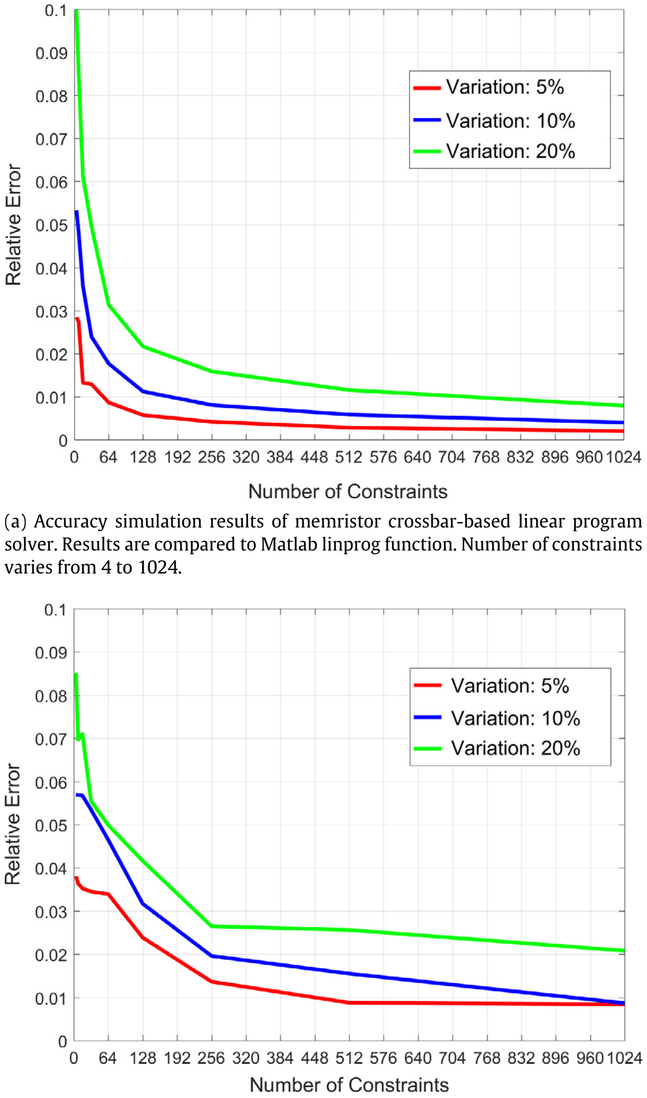
Apart from singular matrix, matrix whose determinant is close to zero could be more vulnerable to process variations. Recall that each unknown in the solution of a linear system can be formulated as the division between determinants of a sub-matrix of coefficient matrix and coefficient matrix according to the Crammer's rule; the solution is inversely proportional to the determinant of coefficient matrix. Hence, matrices whose determinant values are close to zero could lead to massive change in values of solution under the impact of process variation. The accuracy for above two circumstances could be easily affected by process variation.

However, based on our randomly generated experiments, the above two circumstances are not common, and are very rare for large-scale matrices. Besides, based on our experiments, solver for large-scale operations could always converge if a checking scheme is added, that is, solve the problem again if fail to converge. Since process variation differs from each time of writing, actual matrix represented by memristor crossbar is not same as the previous time, thereby, could guarantee convergence.

4.4. Estimated computation latency and energy efficiency

Our experiments based on memristor model from [23] show significant improvement in speed of memristor crossbar based implementation. For instance, the measured delay for Matlab linprog if the number of constraints is 1024 is 6.23 s; the estimated delay for memristor-crossbar based solver is 239 ms with 20% process variation, 195 ms with 10% process variation, 155 ms with 5% variation and, though under ideal condition, 78 ms if no process variation. That would be at least 26×improvements in speed. Improvements for infeasibility detection is more significant, an infeasible system with 1024 constraints could cost Matlab linprog around 30 s to detect while estimated delay on memristor crossbar-based solver is 265 ms with 20% process variation, which is at least 113×speed up.

Our estimation is based on (i) actual simulation results iteration times for convergence, and (ii) the amount of coefficients updating in each iteration is 2.7N(the number of variables is one third of the number of constraints) where N is the number of optimization constraints. A maximum of 110×estimated improvement in speed is achieved compared with PDIP algorithm implemented in MATLAB executed on an Intel I7 server (when the number of constraints is 1024). This significant improvement is because of reduction in computational complexity and speedup due to dedicated hardware implementation. Detailed comparison results are shown in Fig. 6.



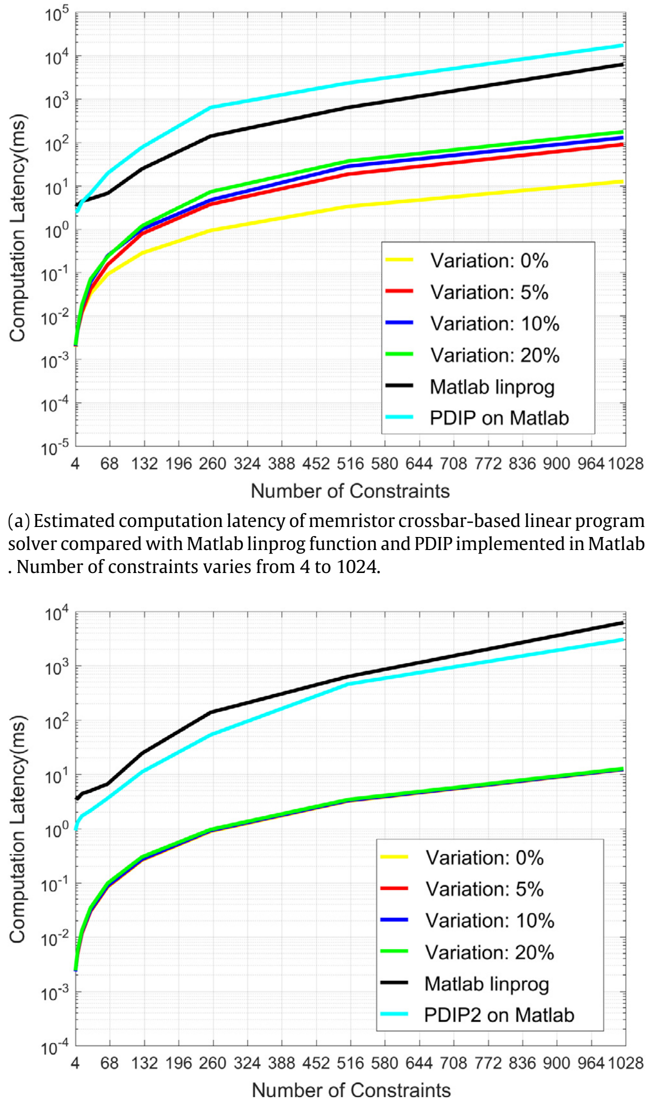
(a) Accuracy simulation results of memristor crossbar-based linear program solver. Results are compared to Matlab linprog function. Number of constraints varies from 4 to 1024.

(b) Accuracy simulation results of memristor crossbar-based linear program solver for large scale operations. Results are compared to Matlab linprog function. Number of constraints varies from 4 to 1024.

Fig. 5. Simulated accuracy.

For proposed memristor crossbar-based linear program solver for large scale operations, speed up over Matlab linprog is also vast. A linear program with 1024 constraints can be solved by proposed structure in less than 80 ms(with 20% process variation) while it could cost Matlab linprog function 6234 ms to derive a solution. However, unlike memristor crossbar-based linear program solver, estimated computation latency does not increase with process variation increment. It can be explained by its constant step length and number of iterations tests results.

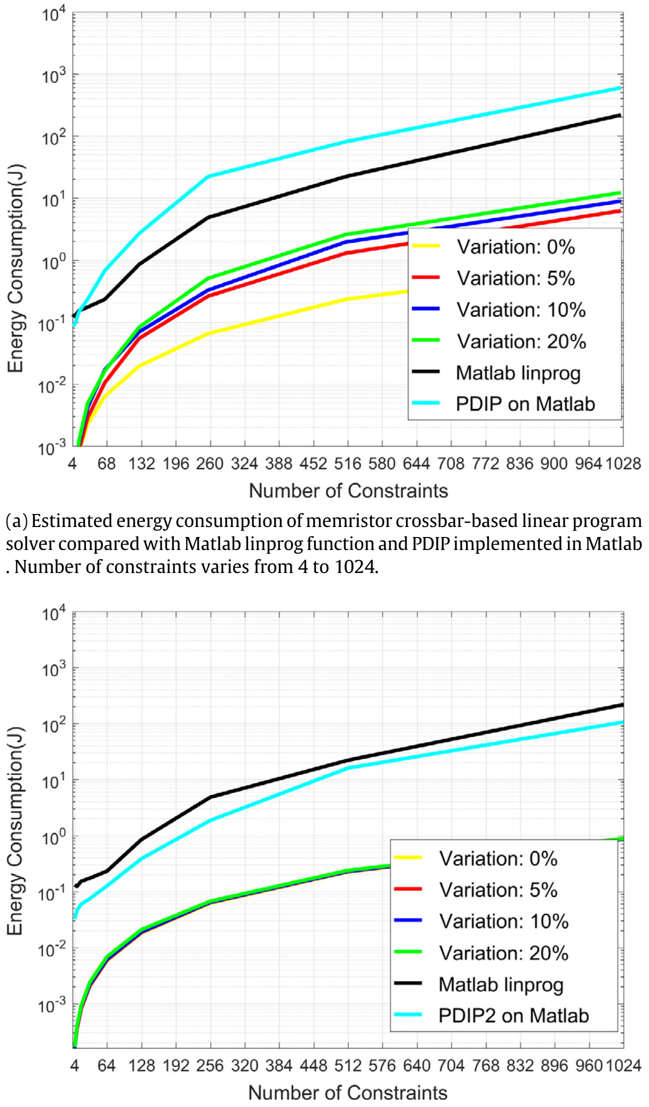
As for energy efficiency, our experiments based on memristor model from [23] show significant improvement in energy efficiency of memristor crossbar based implementation. For instance, the estimated energy consumption for Matlab linprog if the number of constraints is 1024 is 218.1 J; the estimated delay for memristor-crossbar based solver is 12.1 J with 20% process variation, 8.9 J with 10% process variation, 6.2 J with 5% variation



(a) Estimated computation latency of memristor crossbar-based linear program solver compared with Matlab linprog function and PDIP implemented in Matlab . Number of constraints varies from 4 to 1024.

(b) Estimated computation latency of memristor crossbar-based linear program solver for large scale operations. compared with Matlab linprog function. Number of constraints varies from 4 to 1024.

Fig. 6. Estimated computation latency.



(a) Estimated energy consumption of memristor crossbar-based linear program solver compared with Matlab linprog function and PDIP implemented in Matlab . Number of constraints varies from 4 to 1024.

(b) Estimated energy consumption of memristor crossbar-based linear program solver for large scale operations compared with Matlab linprog function. Number of constraints varies from 4 to 1024.

Fig. 7. Estimated energy consumption.

and, though under ideal condition, 0.9 J if no process variation. That would be at least 24×improvements in energy efficiency. Improvements for infeasibility detection is more significant, an infeasible system with 1024 constraints could cost Matlab linprog around 1023.1 J to detect while estimated delay on memristor crossbar-based solver is 10.9 J even with 20% process variation, which is at least 113×speed up.

An average of 30×estimated improvement in energy efficiency is achieved compared with MATLAB linprog executed on an Intel I7 server (when the number of constraints is 1024). The improvement is even larger on proposed implementation for large-scale, an average of 273×. This significant improvement is because of reduction in computational complexity and speedup due to dedicated hardware implementation. Detailed comparison results are shown in Fig. 7.

4.5. Results analysis

Based on our numerous experiments, proposed memristor crossbar-based linear program solver and memristor crossbar-based linear program solver for large scale operations are capable of solving linear programs in a much smaller amount of time while consume less energy. Especially for some large scale infeasible linear problems, which could be time consuming for Matlab linprog function, our proposed implementations could detect infeasibility much faster. Even though process variation could affect number of iterations, overall speedup is still huge thanks to speed advantages of memristor crossbar in matrix–vector operations.

Accuracy tests show that even with up to 20% process variation, proposed memristor crossbar-based implementations can always give a reliable optimal solution. Memristor crossbar-based linear program solver for large scale operations may not guarantee a positive solution due to constant step length, which may also causing converge failure, but it can generate an acceptable optimal value. Convergence failures can be eliminated by a double checking scheme, which solve the problem for a second time if solver indicates infeasible.

In brief, proposed memristor crossbar-based linear program solver and memristor crossbar-based linear program solver for large scale operations are verified to be reliable in accuracy, speed and energy efficiency.

5. Conclusion

This paper described the design of memristor crossbar-based linear program solver using primal–dual interior point algorithm. Two implementations using memristor crossbar have been presented for effectively trading-off between hardware complexity and computing speed. We also presented extension schemes to large-scale applications. Experimental results demonstrate reliable performance with high accuracy and high efficiency.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1637559.

References

[1] K.G. Murty, Linear programming, 1983.

[2] L. Chua, Memristor-the missing circuit element, IEEE Trans. Circuit Theory 18 (5) (1971) 507–519.

[3] D.B. Strukov, G.S. Snider, D.R. Stewart, R.S. Williams, The missing memristor found, Nature 453 (7191) (2008) 80–83.

[4] D.R. Lamb, P.C. Rundle, A non-filamentary switching action in thermally grown silicon dioxide films, Br. J. Appl. Phys. 18 (1) (1967) 29. URL http://stacks.iop.org/0508-3443/18/i=1/a=306.

[5] M. Hu, Y. Wang, Q. Qiu, Y. Chen, H. Li, The stochastic modeling of TiO 2 memristor and its usage in neuromorphic system design, in: Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific, IEEE, 2014, pp. 831–836.

[6] S.H. Jo, T. Chang, I. Ebong, B.B. Bhadviya, P. Mazumder, W. Lu, Nanoscale memristor device as synapse in neuromorphic systems, Nano Lett. 10 (4) (2010) 1297–1301 PMID: 20192230. http://dx.doi.org/10.1021/nl904092h.

[7] M.D. Ventra, Y.V. Pershin, L.O. Chua, Circuit elements with memory: Memristors, memcapacitors, and meminductors, Proc. IEEE 97 (10) (2009) 1717–1724. http://dx.doi.org/10.1109/JPROC.2009.2021077.

[8] M. Hu, H. Li, Y. Chen, Q. Wu, G.S. Rose, Bsb training scheme implementation on memristor-based circuit, in: 2013 IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA, 2013, pp. 80–87. http://dx.doi.org/10.1109/CISDA.2013.6595431.

[9] R. Cai, A Low-Computation-Complexity, Energy-Efficient, and High-Performance Linear Program Solver Using Memristor Crossbars, 2016.

[10] G.B. Dantzig, M.N. Thapa, Linear Programming 1: Introduction, Springer Science & Business Media, 2006.

[11] Á. Rák, G. Cserey, Macromodeling of the memristor in SPICE, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 29 (4) (2010) 632–636.

[12] B. Liu, Y. Chen, B. Wysocki, T. Huang, The circuit realization of a neuromorphic computing system with memristor-based synapse design, in: Proceedings of the 19th International Conference on Neural Information Processing - Volume Part I, ICONIP'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 357–365. http://dx.doi.org/10.1007/978-3-642-34475-6.

[13] D.B. Strukov, G.S. Snider, D.R. Stewart, R.S. Williams, The missing memristor found, Nature 453 (7191) (2008) 80–83. http://dx.doi.org/10.1038/nature06932.

[14] Q. Xia, W. Robinett, M.W. Cumbie, N. Banerjee, T.J. Cardinali, J.J. Yang, W. Wu, X. Li, W.M. Tong, D.B. Strukov, G.S. Snider, G. Medeiros-Ribeiro, R.S. Williams, MemristorCMOS hybrid integrated circuits for reconfigurable logic, Nano Lett. 9 (10) (2009) 3640–3645 PMID: 19722537. http://dx.doi.org/10.1021/nl901874j.

[15] J. Liang, S. Yeh, S.S. Wong, H.-S.P. Wong, Effect of wordline/bitline scaling on the performance, energy consumption, and reliability of cross-point memory array, J. Emerg. Technol. Comput. Syst. 9 (1) (2013) 9:1–9:14. http://dx.doi.org/10.1145/2422094.2422103.

[16] A. Heittmann, T.G. Noll, Limits of writing multivalued resistances in passive nanoelectronic crossbars used in neuromorphic circuits, in: Proceedings of the Great Lakes Symposium on VLSI, GLSVLSI '12, ACM, New York, NY, USA, 2012, pp. 227–232. http://dx.doi.org/10.1145/2206781.2206836.

[17] D. Kadetotad, Z. Xu, A. Mohanty, P.Y. Chen, B. Lin, J. Ye, S. Vrudhula, S. Yu, Y. Cao, J.s. Seo, Neurophysics-inspired parallel architecture with resistive crosspoint array for dictionary learning, in: 2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings, 2014, pp. 536–539. http://dx.doi.org/10.1109/BioCAS.2014.6981781.

[18] R.O. Ferguson, L.F. Sargent, Linear Programming, McGraw-Hill, 1958.

[19] R.V. Robert, Linear programming: Foundations and extensions, 1996.

[20] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, J. Yang, RENO: A high-efficient reconfigurable neuromorphic computing accelerator design, in: 2015 52nd ACM/EDAC/IEEE Design Automation Conference, DAC, 2015, pp. 1–6. http://dx.doi.org/10.1145/2744769.2744900.

[21] J. Steensgaard, Bootstrapped low-voltage analog switches, in: Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on, vol. 2, 1999, pp. 29–32. http://dx.doi.org/10.1109/ISCAS.1999.780611.

[22] M. Hu, H. Li, Y. Chen, X. Wang, R.E. Pino, Geometry variations analysis of TiO2 thin-film and spintronic memristors, in: Proceedings of the 16th Asia and South Pacific Design Automation Conference, ASPDAC '11, IEEE Press, Piscataway, NJ, USA, 2011, pp. 25–30. http://dl.acm.org/citation.cfm?id=1950815.1950820.

[23] C. Yakopcic, T.M. Taha, R. Hasan, Hybrid crossbar architecture for a memristor based memory, in: NAECON 2014 - IEEE National Aerospace and Electronics Conference, 2014, pp. 237–242. http://dx.doi.org/10.1109/NAECON.2014.7045809.

Ruizhe Cai received the B.S. degree in Integrated Circuit Design and Integrated System from Dalian University of Technology, Dalian, Liaoning, China, in 2014, and the M.S. degree in Computer Engineering from Syracuse University, Syracuse, NY, USA, in 2016. He was a research student in Communication and Computer Engineering at Tokyo Institute of Technology, Tokyo, Japan between 2013 to 2014. He is currently a Ph.D. student in Computer Engineering in Syracuse University, Syracuse, NY, USA. His research interests include neuromorphic computing, deep neural network acceleration, and low power design.

Ao Ren received the B.S. degree in Integrated Circuit Design and Integrated System from Dalian University of Technology, Dalian, Liaoning, China, in 2013, and the M.S. degree in Computer Engineering from Syracuse University, Syracuse, NY, USA, in 2015.

He is currently a Ph.D. student in Computer Engineering in Syracuse University, Syracuse, NY, USA. His research interests include neuromorphic computing, deep neural network acceleration, and low power design.

Sucheta Soundarajan was born in Columbus, Ohio, United States in 1984. She earned her BS in Mathematics and Computer/Information Science from The Ohio State University in Columbus, Ohio, United States in 2005, and received her Ph.D. in Computer Science from Cornell University in Ithaca, New York, United States in 2013.

She was a postdoctoral associate in the Department of Computer Science at Rutgers University in New Brunswick, New Jersey from 2013–2015. Since 2015, she has been an Assistant Professor in the Department of Electrical Engineering and Computer Science at Syracuse University in Syracuse, New York. Her research interests include algorithm development and data mining, with a focus on complex networks and large matrices.

She is a member of the ACM and SIAM.

Yanzhi Wang is currently an assistant professor at Syracuse University, starting from August 2015. He received B.S. degree from Tsinghua University in 2009 and Ph.D. degree from University of Southern California in 2014, under supervision of Prof. Massoud Pedram. His research interests include neuromorphic computing, energy-efficient deep learning systems, deep reinforcement learning, embedded systems and wearable devices, etc. He has received best paper awards from International Symposium on Low Power Electronics Design 2014, International Symposium on VLSI Designs 2014, top paper award from IEEE Cloud Computing Conference 2014, and best paper award and best student presentation award from ICASSP 2017. He has two popular papers in IEEE Trans. on CAD. He has received multiple best paper nominations from ACM Great Lakes Symposium on VLSI, IEEE Trans. on CAD, and Asia and South Pacific Design Automation Conference., and International Symposium on Low Power Electronics Design.