A Fast and Effective Memristor-Based Method for Finding Approximate Eigenvalues and Eigenvectors of Non-Negative Matrices

Chenghong Wang, Zeinab S. Jalali, Caiwen Ding, Yanzhi Wang and Sucheta Soundarajan
Department of Electrical Engineering and Computer Science
Syracuse University - Syracuse, NY
{cwang132, zsaghati, cading, ywang393, susounda} @syr.edu

***Abstract*—Throughout many scientific and engineering fields, including control theory, quantum mechanics, advanced dynamics, and network theory, a great many important applications rely on the spectral decomposition of matrices. Traditional methods such as the power iteration method, Jacobi eigenvalue method, and QR decomposition are commonly used to compute the eigenvalues and eigenvectors of a square and symmetric matrix. However, these methods suffer from certain drawbacks: in particular, the power iteration method can only find the leading eigen-pair (i.e., the largest eigenvalue and its corresponding eigenvector), while the Jacobi and QR decomposition methods face significant performance limitations when facing with large scale matrices. Typically, even producing approximate eigen-pairs of a general square matrix requires at least $O(N^3)$ time complexity, where N is the number of rows of the matrix.**

In this work, we exploit the newly developed memristor technology to propose a low-complexity, scalable memristor-based method for deriving a set of dominant eigenvalues and eigenvectors for real symmetric non-negative matrices. The time complexity for our proposed algorithm is $O(\frac{N^2}{\Delta})$ (where Δ governs the accuracy). We present experimental studies to simulate the memristor-supporting algorithm, with results demonstrating that the average error for our method is within 4%, while its performance is up to 1.78X better than traditional methods.

I. INTRODUCTION

As Moore's law is reaching its physical limits, the benefits obtained by technology scaling on planar CMOS will diminish. On the other hand, since we have entered the era of big data, the demands of lower latency and higher computing speed of integrated circuits have been consistently increasing.

The memristor is a newly developed electronic device that, among other important features, allows for hardware-level implementation of extremely fast matrix-vector multiplications and solving of systems of linear equations, and outperforms FPGAs and GPUs in many cases and applications [1]–[4]. Specifically, initializing memristor in real-time to represent a matrix can be potentially done in $O(N)$ time complexity using the proposed parallel writing scheme in [5], and once initialized, matrix-vector multiplication itself can be done in $O(1)$ time. The memristor technology has already proven to be of great value in areas such as deep learning.

In this work, we present the first memristor-based technique for fast and accurate eigen-pair finding. The eigen-pair finding problem is vital to many problems within the science and engineering fields. This task is generally fulfilled using classical methods like the power iteration [6], Jacobi eigenvalue method [7], and QR decomposition [8]. These methods have two shortcomings: 1) They have relatively high computational complexity (at least $O(N^3)$), and 2) They lack flexibility: they either find only the largest eigen-pair or all eigen-pairs. Some applications, like spectral clustering and principal component analysis, require a set of eigenvalues and their eigenvectors. In other words, finding the largest eigen-pair is not enough, but finding all eigen-pairs is more than necessary. One solution is to apply deflation techniques in combination with the power iteration method to find a set of leading eigen-pairs, but this technique incurs additional complexity [9].

To address these problems, we propose a low-complexity, scalable, and flexible memristor-based sweeping method for deriving approximate eigenvalues and eigenvectors for real symmetric non-negative matrices. With this fast and accurate eigen-pair finding method, one can immediately scale up the matrix application algorithms. This type of matrix is important in different areas including social network analysis, where non-negative, symmetric matrices represent adjacency matrices of undirected graphs. The proposed eigen-pair finding method is fast and flexible, allowing the user to find as many or as few leading eigen-pairs as desired. Our contributions are summarized as follows:

- We introduce a novel memristor crossbar-based sweeping algorithm for finding all (or a specified number of leading) eigen-pairs for real symmetric non-negative matrices with complexity $O(\frac{N^2}{\Delta})$, where Δ governs accuracy.
- We discuss in details how one can implement the sweeping algorithm on a memristor crossbar.
- We conduct experimental studies to simulate memristor supported hardware and evaluate our proposed method on the simulation platform. Simulation results indicate a large performance gain with low accuracy loss.

II. BACKGROUND & RELATED WORK

In this section, we first describe background related to the previous work on matrix eigen-pair finding, and then describe the background of memristor-based matrix operations.

A. Eigen Decomposition

Finding a matrix's leading eigenvalue and corresponding eigenvector is often done via the power iteration method. To use the power iteration method for finding more eigenvalues,

one must perform a series of matrix deflation operations [9] in each round, which removes the leading eigen-pair and thus allows the power iteration method to find the next eigen-pair.

The Lanczos method is another option, which can be used to find larger quantities of eigenvalues and eigenvectors [10]. The Jacobi, Rayleigh quotient iteration [11] and the QR decomposition methods [12], [13] are also commonly used to find eigenvalues and eigenvectors, and, unlike the power iteration and Lanczos method, find all eigenvalues simultaneously. However, the computational complexities for these techniques can be large: Using the power iteration and Lanczos method to find the largest pair of eigenvalue and eigenvectors requires $O(N^2)$ complexity, and when used with deflation to produce a set of eigen-pairs, the complexity is increased to $O(mN^3)$, where m is the number of required eigenvalues. The QR decomposition requires $O(N^2)$ and $6N^3+O(N^2)$ complexity to produce all eigenvalues and all eigen-pairs, respectively, and the Jacobi method and Rayleigh quotient iteration also have cubic computational complexity.

B. Memristors and Crossbar Arrays

Initially proposed by Leon Chua in 1971, and created by HP Labs in 2008, a memristor is an electrical component with the capacity to 'remember' the historical profile of excitations on the device. Specifically, the state (memristance) of a memristor will change when voltage higher than a threshold voltage, i.e., $|V_m| > |V_{th}|$, is applied at its terminals. Otherwise, the memristor behaves like a resistor, with features such as non-volatility, low-power, high density, and excellent scalability [14], [15]. In this work, we take advantage of the ability to connect memristors together into a crossbar array. By applying voltages at different input locations of this array and observing the output voltages, we can efficiently calculate matrix-vector products in the analog domain. Specifically, once the crossbar memristances are set, matrix-vector multiplication and linear equation solving can be performed in $O(1)$ time complexity [1], [2], [16], [17]. Using the parallel writing scheme in [5], the memristor can potentially be set in $O(N)$ time. Thus, multiplication and linear equation solving are fast even if the matrix must be initialized; and if the matrix has been initialized earlier, these operations can be done in constant time. Memristors have been critical to research on neural networks (e.g., neuromorphic computing) [18], [19], as well as other areas like pattern recognition [20], text recognition [21], random number generation [22], and the dot-product engine [23]. In general, memristor-based algorithms for the above applications are evaluated using simulation or emulation, due to limited access to memristor technology.

III. PROPOSED METHOD

In this section, we first introduce the design of our proposed sweeping algorithm, then discuss how to accelerate our method by implementing it on a memristor crossbar. We end with a discussion of the time complexity of our proposed method.

A. Sweeping Based Eigen-pair Finding Method

Consider an N-by-N non-negative, non-singular, symmetric matrix $\mathbf{M}$. We begin with the following observation:

Observation 1: Suppose $\mathbf{b}$ is a vector of random numbers in the range [0, 1]. Define $\mathbf{x}_\beta$ as the solution to $(\mathbf{M}-\beta\mathbf{I})\mathbf{x}_\beta = \mathbf{b}$. Then if λ is an eigenvalue of $\mathbf{M}$, as β approaches λ, $||\mathbf{x}_\beta||_\infty$ approaches ∞. Thus, if we define a small step-size Δ, then if we initialize β to be an upper bound on the largest eigenvalue of $\mathbf{M}$ and gradually decrement β by ϵ to 'sweep' across the range of potential eigenvalues, then by repeatedly solving the problem $(\mathbf{M}-\beta\mathbf{I})\mathbf{x}_\beta = \mathbf{b}$. and examining $||\mathbf{x}_\beta||_\infty$, we can identify the eigenvalues of $\mathbf{M}$.

Figure 2 depicts this relationship on a a toy 10×10 random symmetric non-negative matrix, where β is decremented by 0.01 in each step. The value of $||\mathbf{x}_\beta||_\infty$ varies with β, and forms peaks at certain locations. When we examine the locations of these peaks, we find that every peak (and corresponding β) corresponds to an eigenvalue of the target matrix. We observed this property over a large set of random and real matrices. Thus, if we have the upper bound for matrix $\mathbf{M}$'s eigenvalue distribution and the number of desired eigenvalues, and find the β values such that $max(\mathbf{x}_\beta)$ forms a peak, then these β's are approximate eigenvalues of $\mathbf{M}$.

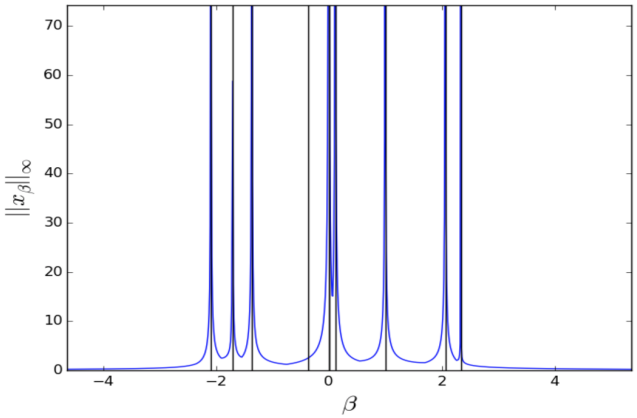

Fig. 1. $max(\mathbf{x}_\epsilon)$ vs. ϵ on a non-negative matrix. The peaks of the curve occur at values of ϵ corresponding to eigenvalues of the matrix. Black lines indicate true eigenvalues.

Algorithm: This observation naturally leads to a sweeping algorithm to approximately identify all or a set of leading eigenvalues of a matrix $\mathbf{M}$, as presented in Algorithm 1. This algorithm requires four input parameters: a non-negative square symmetric matrix $\mathbf{M}$; the searching interval $[L, U]$, which contains all eigenvalues of matrix $\mathbf{M}$; the step decrement value Δ[1], and the number k of eigenvalues desired.

To set L , we follow Gershgorin's Circle Theorem [24], which finds L by taking the sum of the absolute values of the off-diagonal elements in each row, and adding the diagonal element in that row to that sum[2]. U can be found equivalently by negating $\mathbf{M}$.

After setting all necessary parameters, the sweeping algorithm searches for eigenvalues within the interval in the range $[L, U]$. First, the algorithm creates a random positive vector $\mathbf{b}$, and then performs a series of iterations to locate the eigenvalues. The algorithm first steps with $\beta = U$, and

[1]According to our experimental studies, we suggest a Δ around 0.01 to 0.05.

[2]Note that because we consider only non-negative matrices, this value is simply the sum of the row.

Algorithm 1 Sweeping-Based Eigenvalues Search Method

Input:
Matrix, $\mathbf{M}$
Sweeping Interval, $[L, U]$
Fixed Increment, Δ
Number of Eigenvalues required, N
Output:
Set of Eigenvalues, E.

$\beta \leftarrow U$
Solve the Linear Equation $(\mathbf{M} - \beta\mathbf{I})\mathbf{x} = \mathbf{b}$
$EigNum \leftarrow 0, x_0 \leftarrow -1, x_1 \leftarrow -1$
$x_0 \leftarrow max(\mathbf{x})$
while $EigNum \leq N$ and $\beta > L$ **do**
  $\beta \leftarrow \beta - \Delta$.
  Solve Linear Equation $(\mathbf{M} - \beta\mathbf{I})\mathbf{x} = \mathbf{b}$
  $x' \leftarrow max(\mathbf{x})$
  if $x_0 \neq -1$ and $x_1 \neq -1$ and $x_0 \leq x'$ and $x' \geq x_1$ **then**
    $N \leftarrow N - 1,\ E \leftarrow E \cup \{\beta\}$
    $x_0 \leftarrow x_1,\ x_1 \leftarrow x'$
  end if
end while
return E;

in each iteration: (i) decreases β by the fixed decrement Δ, (ii) solves the problem $(\mathbf{M} - \beta\mathbf{I})\mathbf{x}_\beta = \mathbf{b}$, and (iii) records $||\mathbf{x}_\beta||_\infty$ as x'_β. The algorithm then finds those x'_β values where $x'_{\beta-\Delta} \leq x'_\beta$ and $x'_\beta \geq x'_{\beta+\Delta}$ (in other words, the local peaks). This procedure continues until k such values have been obtained, and the resulting eigenvalues are stored within the set E. Because β begins at U and is decremented until it reaches L, these are the leading eigenvalues. The eigenvalues detected by this method are approximate, not exact, with the error depending on step size Δ. As Δ decreases, the accuracy and running time increase. For instance, the running time of finding eigen-pairs with accuracy 0.01 is five times slower than finding eigen-pairs with accuracy 0.05.

To find the corresponding eigenvectors, we adopt the *Inverse Iteration* method [25]. We create a random vector $\mathbf{v}_0$, and repeat the following until convergence:

- Define $\mathbf{w}_{i+1}$ to be the solution to $(\mathbf{M} - \lambda'\mathbf{I})\mathbf{w}_{i+1} = \mathbf{z}_i$.
- Set $\mathbf{z}_{i+1} = \frac{\mathbf{w}_{i+1}}{||\mathbf{w}_{i+1}||_\infty}$. The infinity norm is simply the largest absolute value of an element in the vector.

In this way, we obtain the set of leading eigenvalues and corresponding eigenvectors. The divergence of maximum components near eigenvalues can be proven through application of Cramer's Rule, but the scope and space constraints of this paper prevent us from presenting the proof here.

B. Implementation on the Memristor Crossbar

The sweeping method requires solving the linear equation $(\mathbf{M} - \beta\mathbf{I})\mathbf{x} = \mathbf{b}$ at each iteration. This can easily be implemented using a memristor crossbar involving two operations: 1) Mapping matrix $(\mathbf{M} - \beta\mathbf{I})$ and vector $\mathbf{b}$ onto memristor crossbar and 2) solving the linear equation $(\mathbf{M} - \beta\mathbf{I})\mathbf{x} = \mathbf{b}$ using memristor crossbar. We map the matrix using the parallel writing scheme described in [5] in $O(N)$ time, and once mapped, the crossbar solves the linear equation in $O(1)$ time.

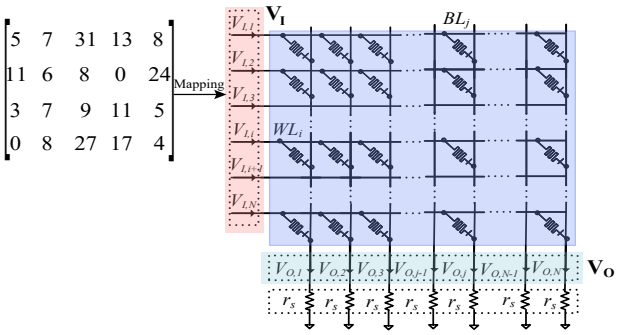

Fig. 2. The memristor crossbar structure and an example of matrix mapping

Here, we show how these two operations can be implemented using a memristor crossbar.

A typical $N \times N$ memristor crossbar is illustrated in Fig. 2. A memristor is connected between each pair of horizontal *word-lines* (WL) and vertical *bit-lines* (BL). This structure can be implemented with a small footprint, and each memristor can be re-programmed to different resistance states by applying biasing voltages at its two terminals [2], [16]. To perform matrix-vector multiplications, we apply a vector of input voltages $\mathbf{V_I}$ on the WLs and collect the current through each BL by measuring the voltage across resistor R_s with conductance of g_s. Suppose 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 $\mathbf{V_O} = \mathbf{C} \times \mathbf{V_I}$. Here, each coefficient $\mathbf{C}_{ij}$ of matrix $\mathbf{C}$ is (approximately) proportional to the conductance $g_{i,j}$ [1], [4]. Here, $\mathbf{C}$ is represented by the conductance of memristor.

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

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

Previous work [26] has demonstrated that we can use a fast and simple approximation $g_{i,j} = c_{i,j} \cdot g_{max}$ to map the above matrix onto a memristor crossbar, in which g_{max} is the largest value of $\mathbf{G}$. Hence, when we want to calculate matrix-vector multiplication $\mathbf{y} = \mathbf{M}\mathbf{x}$, we can set $\mathbf{M} = g_{max}\mathbf{C}$ and $\mathbf{y} = g_s\mathbf{V_O}$, and the solution is: $\mathbf{x} = g_s/g_{max}\mathbf{V_I}$.

In the opposite direction, the memristor crossbar structure can solve a linear system of equations, as required by our algorithm [5]. A voltage vector $\mathbf{V_O}$ is applied on each R_s of BL, so the current flowing through each BL is approximated as $I_{o,j} = g_s V_{o,j}$. The current $I_{o,j}$ through BL_j can also be calculated as $I_{o,j} = \sum_j V_{I,i} g_{i,j}$. For each BL_j, the equation $\frac{1}{g_s}\sum_j V_{I,i} g_{i,j} = V_{o,j}$ is mapped onto the crossbar. The solution $\mathbf{V_I}$ can be found by measuring voltages on the WLs.

One challenge in implementing the sweeping algorithm using memristor crossbars is that a memristor crossbar only allows square matrices with nonnegative entries. By assumption, matrix $\mathbf{M}$ is a square matrix with nonnegative elements; however, matrix $(\mathbf{M} - \beta\mathbf{I})$ may have negative elements on the diagonal, depending on the value of β. Thus, inherent hardware limitations prevent the direct mapping of this matrix into the memristor crossbar. To overcome this problem, if the

input matrix contains negative elements, the linear equation $(\mathbf{M}-\beta\mathbf{I})\mathbf{x}=\mathbf{b}$ will be converted to another equation using the technique in [27]. If $\mathbf{A}=(\mathbf{M}-\beta\mathbf{I})$, the conversion is as follows (from Eqn. (2) to Eqn. (3)):

$$\begin{bmatrix} A_{1,1} & \cdots & A_{1,N} \\ \vdots & \ddots & \vdots \\ A_{N,1} & \cdots & A_{N,N} \end{bmatrix} \times \begin{bmatrix} X_1 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix} \quad (2)$$

$$\begin{bmatrix} 0 & \cdots & A_{1,N} & -A_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ A_{N,1} & \cdots & 0 & 0 & \cdots & -A_{N,N} \\ 1 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 1 & 0 & \cdots & 1 \end{bmatrix} \times \begin{bmatrix} X_1 \\ \vdots \\ X_N \\ -X_1 \\ \vdots \\ -X_N \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (3)$$

Note that currently, existing memristor crossbars are quite small (e.g., 1024 x 1024). However, as with any new hardware devices, this size is likely to increase in the future. Moreover, the proposed method is intended as a core algorithm upon which other algorithms can be built (e.g., a divide-and-conquer version of this method) for larger-scale applications.

C. Complexity Analysis

The sweeping algorithm consists of three stages: (1) Calculate the upper bound for performing eigenvalue searching; (2) Perform the sweeping based eigenvalue-finding algorithm; (3) Calculate corresponding eigenvectors using the inverse method. We make the following assumptions:

- The input matrix $\mathbf{M} \in \mathbf{R}^{N\times N}$ is a full-rank real matrix with non-negative entries in the range $[0,1]$[3].
- The upper bound is set to be $1+max(D)$, and the lower bound to $-1-max(D)$, where $D=\{d_1,d_2,...,d_n\}$ indicates the sum of each rows of matrix.
- The step increment is set to Δ.

For these three stages, we have the following analysis:
(1) Upper and Lower Bound Calculation: To calculate the upper and lower bounds of the eigenvalues of matrix $\mathbf{M} \in \mathbf{R}^{N\times N}$ with the Gershgorin Circle Theorem, we need to find the sum of each row, which takes $O(N^2)$ time, and identify the largest sum from all results, which takes $O(N)$ time. The total computation time is thus $O(N^2)$.
(2) Sweeping to Find Eigenvalues: The length of the sweeping interval is $O(D)$. Because we assumed that the elements in $\mathbf{M}$ are in the range $[0,1]$, we have $D \leq N$. The algorithm searches in space $[-D-1, D+1]$, with step size Δ, and hence the total number of iterations is given by $\frac{2(1+max(D))}{\Delta} \sim O(\frac{N}{\Delta})$. In each iteration, the time complexity of mapping $\mathbf{M}$ to crossbar in $O(N)$, and once mapped, the crossbar solves the linear equation in $O(1)$ [5]. Thus, the total complexity is $O(\frac{N^2}{\Delta})$.
(3) Eigenvector Calculation: Once k eigenvalues have been found, the sweeping method uses the inverse iteration method to approximate their corresponding eigenvectors. This requires $b\cdot(O(N)+O(N)) \sim O(b\cdot N)$ complexity, where b is the number of rounds before convergence (usually 2-4).

Based on this analysis, the complexity for the whole process is $O(N^2)+O(\frac{\alpha\cdot N^2}{\Delta})+O(b\cdot N) \sim O(\frac{N^2}{\Delta})$, a significant improvement compared to existing eigen-finding methods. Note that the complexity is governed by Δ, which controls the tradeoff between accuracy and running time.

[3]If the matrix has values outside this range, it can be normalized with elapsed time $O(N^2)$, which does not affect the overall complexity analysis.

D. Algorithmic Improvements

There are numerous improvements that can be made to this algorithm, such as finding repeated eigen-pairs, dynamically selecting the step decrement value Δ, and using divide-and-conquer techniques [28] to handle matrices too large to fit on the crossbar. Due to space and scope limitations, we focus this paper on the core sweeping algorithm and implementation details, and present algorithmic improvements separately.

IV. Experimental Studies

We conduct three sets of experimental studies to evaluate the proposed method: (i) experiments to measure the accuracy of the identified eigen-pairs, (ii) running time experiments and (iii) energy efficiency comparison experiments.

A. Experiment Setup

In our first set of experiments, we evaluate our sweeping algorithm on synthetic matrices of sizes 1000×1000, 5000×5000, and 10000×10000, and three real world network adjacency matrices (Dolphins, Facebook, Robots [29]). We compare the results obtained by our proposed method to those produced by MATLAB's eigen-pair functions. Separately, we introduce random matrix and vector errors to demonstrate that our sweeping algorithm is not sensitive to potential imprecisions that may occur in memristor devices. In each round, a random vector (of the same size as $\mathbf{x}$) and a random matrix (of the same size as $\mathbf{M}$) with elements in the range of $\{-\epsilon,+\epsilon\}$ are added to $\mathbf{x}$ and $(\mathbf{M}-\epsilon\mathbf{I})$, respectively.

Second, we compare the running time and energy consumption rates between our sweeping algorithm and the QR decomposition, where both algorithms have been boosted using memristor (we show only results for the QR decomposition, because this was the best standard method that we considered).

Third, we compare the energy consumption rate of the sweeping algorithm with and without memristor boosting. We design a memristor simulator to evaluate the efficiency of the proposed methods. The Matlab-based memristor crossbar simulator is designed based on real memristor model (a fabricated 8 nm × 8 nm memristive device demonstrating fast switching property of around 10 ns and more than 20,000 successful operations) as proposed in [30], with power consumption per switch of 3 Nano-watts. We run our experiments on a server with Intel i7 6700HQ, 2.6 GHz CPU, 48G DDR4 memory, and 512G SSD hardware. We use three evaluation metrics:

- VAL_{err}: The mean error between the eigenvalues detected by the sweeping algorithm and the corresponding actual eigenvalues. The inaccuracy for one specific sweeping-produced eigenvalue λ and its corresponding actual value λ' is calculated as $\frac{|\lambda-\lambda'|}{\lambda'}$
- VEC_{err}: The average error in the eigenvectors found by the sweeping method. The error is calculated by $\frac{||\mathbf{v}-\mathbf{v}'||}{||\mathbf{v}||}$, where $\mathbf{v}'$ is an eigenvector calculated by the sweeping algorithm and $\mathbf{v}$ is the corresponding actual eigenvector.
- $Energy\ Performance$: The total energy consumption calculated when executing the algorithms on our simulation platform. The energy is calculated by $E=P_{cpu}*$

TABLE I
ACCURACY RESULTS FOR SWEEPING ALGORITHM BEFORE AND AFTER PERTURBATION– RANDOM MATRICES

Random Dataset	Measure	Percentage of Calculated Eigen-Pairs							
		5%	**10%**	**15%**	**20%**	**25%**	**30%**	**50%**	**100%**
1000×1000	VAL_{err}	5.0E-4	6.2E-4	6.7E-4	6.9E-4	7.3E-4	7.9E-4	1.0E-3	4.4E-3
	$VAL_{err(p)}$	4.8E-4	6.0E-4	6.7E-4	6.9E-4	7.3E-4	7.9E-4	1.0E-3	4.4E-3
	VEC_{err}	8.8E-3	3.5E-2	2.2E-2	1.7E-2	1.7E-2	1.4E-2	9.7E-3	8.6E-3
	$VEC_{err(p)}$	1.8E-4	3.1E-2	2.6E-2	2.0E-2	1.7E-2	1.4E-2	9.6E-3	8.8E-3
5000×5000	VAL_{err}	1.0E-4	1.4E-4	2.8E-4	3.7E-4	4.1E-4	4.5E-4	9.8E-4	1.2E-3
	$VAL_{err(p)}$	1.1E-4	1.4E-4	2.8E-4	3.6E-4	4.1E-4	4.5E-4	9.7E-4	1.2E-3
	VEC_{err}	1.1E-6	2.1E-2	1.0E-2	7.0E-3	1.0E-2	1.4E-2	5.5E-2	4.8E-2
	$VEC_{err(p)}$	8.0E-7	8.9E-2	4.5E-2	2.8E-2	2.3E-2	2.0E-2	5.4E-2	5.5E-2
10000×10000	VAL_{err}	1.3E-4	1.3E-4	1.7E-4	2.2E-4	2.6E-4	3.1E-4	4.7E-4	5.3E-2
	$VAL_{err(p)}$	1.2E-4	1.3E-4	1.6E-4	2.2E-4	2.6E-4	3.1E-4	4.7E-4	5.4E-2
	VEC_{err}	5.0E-4	5.0E-4	1.1E-2	1.1E-2	1.1E-2	2.0E-2	1.3E-1	1.4E-1
	$VEC_{err(p)}$	5.3E-4	9.0E-4	1.1E-2	1.1E-2	1.0E-2	2.4E-2	1.3E-1	1.7E-1

TABLE II
ACCURACY RESULTS FOR SWEEPING ALGORITHM BEFORE AND AFTER PERTURBATION – REAL WORLD DATASETS

Real World Dataset	Measure	Percentage of Calculated Eigen-Pairs							
		5%	**10%**	**15%**	**20%**	25%	**30%**	50%	**100%**
dolphins	VAL_{err}	3.4E-4	6.3E-4	6.1E-4	5.6E-4	1.1E-3	2.6E-3	6.4E-3	1.4E-2
	$VAL_{err(p)}$	7.6E-4	6.6E-4	5.8E-4	9.4E-4	1.5E-3	2.4E-3	6.4E-3	1.4E-2
	VEC_{err}	6.7E-7	4.0E-7	3.5E-7	1.9E-6	1.8E-6	1.8E-6	3.0E-3	1.8E-3
	$VEC_{err(p)}$	1.1E-6	6.4E-6	5.4E-7	3.5E-6	3.2E-6	3.1E-6	3.4E-3	3.3E-3
robots	VAL_{err}	6.6E-4	1.1E-3	1.2E-3	1.2E-3	1.1E-3	1.2E-3	1.2E-3	5.3E-3
	$VAL_{err(p)}$	5.7E-4	1.1E-3	1.1E-3	1.2E-3	1.1E-3	1.2E-3	1.2E-3	5.3E-3
	VEC_{err}	2.3E-6	1.0E-6	1.2E-5	3.4E-2	3.4E-2	3.8E-2	4.0E-2	8.2E-2
	$VEC_{err(p)}$	1.9E-6	1.8E-6	9.1E-7	3.9E-2	3.8E-2	4.3E-2	4.2E-2	7.5E-2
Facebook	VAL_{err}	2.1E-4	4.0E-4	4.0E-4	3.9E-4	3.9E-4	4.1E-4	4.4E-4	4.5E-4
	$VAL_{err(p)}$	1.7E-4	3.9E-4	3.9E-4	3.9E-4	3.9E-4	4.1E-4	4.4E-4	4.5E-4
	VEC_{err}	2.3E-2	1.0E-2	4.1E-2	3.7E-2	1.86E-1	1.65E-1	1.73E-1	1.76E-1
	$VEC_{err(p)}$	2.4E-2	1.8E-2	4.7E-2	4.2E-2	1.95E-1	1.80E-1	1.78E-1	1.78E-1

$T_{cpu} + P_{mem} * T_{mem}$ where P_{cpu} is the power specification mentioned in [31] for selected CPU and P_{mem} is the power consumption for executing computations on the 8 nm × 8 nm memristive device [30]. T_{cpu} and T_{mem} are the elapsed times for software and memristor.

B. Results and Discussion

We present our results in three parts: accuracy experiments (both with and without hardware inaccuracies), performance (running time) results, and energy consumption results.

1) Accuracy Experiments: The results of our accuracy experiments are depicted in Table I and Table II, where VAL_{err} (VEC_{err}) and $VAL_{err(p)}$ ($VEC_{err(p)}$) represent, respectively, the accuracy results for the detected eigenvalues and eigenvectors found by the sweeping method, without and with simulated perturbation errors. Table I shows results for different sizes of synthetic matrices, while Table II presents results for real world datasets. Our results show that:

First, our sweeping algorithm produces a set of leading eigen-pairs with high accuracy. The average error over all eigenvalues for all testing datasets is around 1%, and the average error for producing all eigenvectors is approximately 7%. The accuracy for computed eigen-pairs is above 90%, which is acceptable for many applications, such as PCA or spectral clustering, that use eigen-pairs for further computing.

Second, we observe that the sweeping method is not sensitive to perturbations, indicating that memristor imprecisions do not have strong effect on its accuracy.

Third, the fewer leading eigenvalues we require, the higher the accuracy. For example, on a random 1000×1000 matrix, when we use the sweeping algorithm to produce 5% of leading eigenvalues (i.e., top-50 eigenvalues), the eigenvalue inaccuracy is 0.0005. However, when we identify the 25% leading eigenvalues (i.e., top 250 eigenvalues), this error almost doubles. The inaccuracy reaches 0.0044 when we detect all eigenvalues. This occurs because if Δ is too large, it may skip over eigenvalues. Eigenvalues tend to concentrate near the central part of the distribution, resulting in higher inaccuracy. However, many applications such as PCA and spectral clustering only require a set of leading eigen-pairs.

2) Performance Experiments: The results of our second experiment are shown in Figure 3, where the x-axis indicates the matrix size and the y-axis shows the total running time. Here, we set $\Delta = 0.05$. The text in the plots shows the approximate accuracy of the algorithms for different tests. The running time of our proposed method shows a significant improvement over the QR decomposition, especially for larger matrix sizes. For example, on a 10000x10000 matrix, the elapsed time for sweeping method is 22.39s, while the elapsed time for QR decomposition is 77.53s, a 3.4X improvement. On average, the sweeping method is 1.78X faster than QR decomposition method. The sweeping method also shows a higher accuracy. The sweeping method is faster than the QR decomposition, while matching or beating its accuracy.

3) Energy Consumption Experiments: Results for the third experiment are listed in Table III, where the total energy consumption for software-based sweeping algorithm and the memristor boosted sweeping method in each test rounds are shown. In all cases, the memristor boosted algorithm demonstrates significant energy reduction as compared with software implementation. For instance, the total energy consumptions for software sweeping algorithm with matrix sizes 5000 and 10000 are around 55 and 165 times of that of the memristor boosted one. The significant reduction in energy consumption

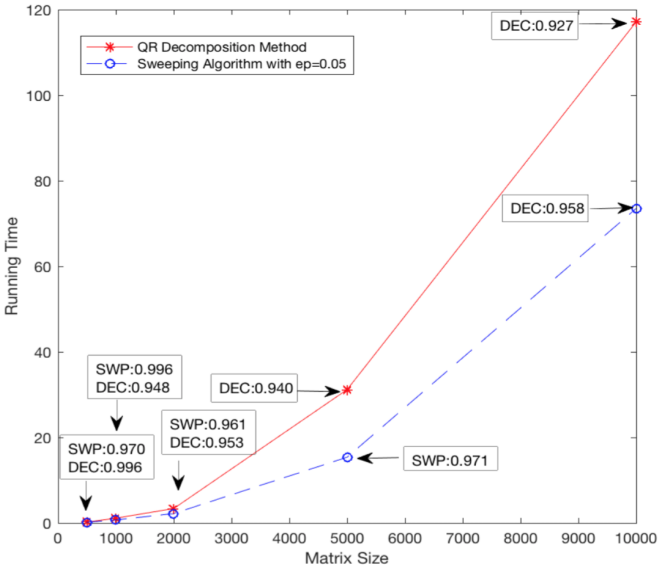

Fig. 3. Elapsed Time Comparison between the proposed Sweeping Algorithm and the QR Decomposition based Method

TABLE III
ENERGY PERFORMANCE COMPARISON

Matrix Size	with memristor (J)	without memristor (J)
$\mathbf{500 \times 500}$	4.60	24.74
$\mathbf{1000 \times 1000}$	15.41	189.75
$\mathbf{2000 \times 2000}$	68.46	1942.38
$\mathbf{5000 \times 5000}$	385.02	21091.93
$\mathbf{10000 \times 1000}$	1799.94	298431.12

indicates that the proposed method is suitable for implementation, and so is suitable for a variety of research fields.

V. CONCLUSION AND FUTURE WORK

We have proposed a novel sweeping algorithm for finding eigen-pairs of a non-negative square matrix. Our proposed method runs in $O(\frac{N^2}{\Delta})$ time.Additionally, our algorithm is flexible, allowing the user to find only the desired number of eigen-pairs. We demonstrated that our method can precisely find sets of eigen-pairs with very low imprecisions, with a very low elapsed time as compared to other methods and significant energy consumption reduction as compared with software methods. Experimental Results show that our proposed method can achieve up to a 165$\times$ energy saving across matrix sizes.

REFERENCES

[1] M. Hu, H. Li, Q. Wu, and G. Rose, "Hardware realization of neuromorphic bsb model with memristor crossbar network," in *IEEE Design Automation Conference (DAC)*, 2012, pp. 554–559.

[2] D. Kadetotad, Z. Xu, A. Mohanty, P.-Y. Chen, B. Lin, J. Ye, S. Vrudhula, S. Yu, Y. Cao, and J.-s. Seo, "Neurophysics-inspired parallel architecture with resistive crosspoint array for dictionary learning," in *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE*. IEEE, 2014, pp. 536–539.

[3] M. Sharad, D. Fan, and K. Roy, "Ultra low power associative computing with spin neurons and resistive crossbar memory," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–6.

[4] S. Yu and Y. Cao, "On-chip sparse learning with resistive cross-point array architecture," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 195–197.

[5] I. Richter, K. Pas, X. Guo, R. Patel, J. Liu, E. Ipek, and E. G. Friedman, "Memristive accelerator for extreme scale linear solvers," Tech. rep. University of Rochester, 2015. u rl: http://www. ece. rochester. edu/users/friedman/papers/GOMAC_15. pdf, Tech. Rep., 2015.

[6] B. N. Parlett, *The symmetric eigenvalue problem*. SIAM, 1998.

[7] G. L. Sleijpen and H. A. Van der Vorst, "A jacobi–davidson iteration method for linear eigenvalue problems," *Siam Review*, vol. 42, no. 2, pp. 267–293, 2000.

[8] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*. Springer Science & Business Media, 2010, vol. 37.

[9] K.-J. Bathe and E. L. Wilson, *Numerical methods in finite element analysis*. Prentice-Hall Englewood Cliffs, NJ, 1976, vol. 197.

[10] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

[11] B. N. Parlett, "The rayleigh quotient iteration and some generalizations for nonnormal matrices," *Mathematics of Computation*, vol. 28, no. 127, pp. 679–693, 1974.

[12] H. Rutishauser, "The jacobi method for real symmetric matrices," *Numerische Mathematik*, vol. 9, no. 1, pp. 1–10, 1966.

[13] C. R. Goodall, "13 computation using the qr decomposition," *Handbook of statistics*, vol. 9, pp. 467–508, 1993.

[14] B. Liu, Y. Chen, B. Wysocki, and T. Huang, "The circuit realization of a neuromorphic computing system with memristor-based synapse design," in *Neural Information Processing*. Springer, 2012, pp. 357–365.

[15] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[16] A. Heittmann and 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*. ACM, 2012, pp. 227–232.

[17] G. Yuan, C. Ding, R. Cai, X. Ma, Z. Zhao, A. Ren, B. Yuan, and Y. Wang, "Memristor crossbar-based ultra-efficient next-generation baseband processors," in *Circuits and Systems (MWSCAS), 2017 IEEE 60th International Midwest Symposium on*. IEEE, 2017, pp. 1121–1124.

[18] A. Thomas, "Memristor-based neural networks," *Journal of Physics D: Applied Physics*, vol. 46, no. 9, p. 093001, 2013.

[19] R. Hasan and T. M. Taha, "Enabling back propagation training of memristor crossbar neuromorphic processors," in *Neural Networks (IJCNN), 2014 International Joint Conference on*. IEEE, 2014, pp. 21–28.

[20] C. Yakopcic and T. M. Taha, "Energy efficient perceptron pattern recognition using segmented memristor crossbar arrays," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–8.

[21] Q. Qiu, Z. Li, K. Ahmed, W. Liu, S. F. Habib, H. H. Li, and M. Hu, "A neuromorphic architecture for context aware text image recognition," *Journal of Signal Processing Systems*, vol. 84, no. 3, pp. 355–369, 2016.

[22] H. Jiang, D. Belkin, S. E. Savel'ev, S. Lin, Z. Wang, Y. Li, S. Joshi, R. Midya, C. Li, M. Rao *et al.*, "A novel true random number generator based on a stochastic diffusive memristor," *Nature communications*, vol. 8, no. 1, p. 882, 2017.

[23] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication," in *Proceedings of the 53rd annual design automation conference*. ACM, 2016, p. 19.

[24] H. E. Bell, "Gershgorin's theorem and the zeros of polynomials," *The American Mathematical Monthly*, vol. 72, no. 3, pp. 292–295, 1965.

[25] J. W. Demmel, *Applied numerical linear algebra*. SIAM, 1997.

[26] M. Hu, H. Li, Y. Chen, Q. Wu, and G. S. Rose, "Bsb training scheme implementation on memristor-based circuit," in *Computational Intelligence for Security and Defense Applications (CISDA), 2013 IEEE Symposium on*. IEEE, 2013, pp. 80–87.

[27] R. Cai, A. Ren, Y. Wang, S. Soundarajan, Q. Qiu, B. Yuan, and P. Bogdan, "A low-computation-complexity, energy-efficient, and high-performance linear program solver using memristor crossbars," in *System-on-Chip Conference (SOCC), 2016 29th IEEE International*. IEEE, 2016, pp. 317–322.

[28] M. Gu and S. C. Eisenstat, "A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem," *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 1, pp. 172–191, 1995.

[29] J. Leskovec and A. Krevl, "{SNAP Datasets}:{Stanford} large network dataset collection," 2015.

[30] S. Pi, P. Lin, and Q. Xia, "Cross point arrays of 8 nm$\times$ 8 nm memristive devices fabricated with nanoimprint lithography," *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena*, vol. 31, no. 6, p. 06FA02, 2013.

[31] https://ark.intel.com/products/88967/Intel-Core-i7-6700HQ-Processor-6M-Cache-up-to-3_50-GHz.