Algorithm-Hardware Co-Optimization of Memristor-Based Framework for Solving SOCP and Homogeneous QCQP Problems

Ao Ren	Syracuse University
Sijia Liu	University of Michigan
Ruizhe Cai	Syracuse University
Wujie Wen	Florida International University
Pramod K. Varshney	Syracuse University
Yanzhi Wang	Syracuse University

Outline

1. Motivation
2. Background
   - Memristor & Memristor Crossbar
   - Convex Optimization
   - SOCP
   - QCQP
3. Memristor Crossbar-Based Framework For Solving Convex Optimization Problems
4. Experiments & Analysis
5. Conclusion

1. Motivation

- The power of convex optimization has been shown extensively in various applications such as signal process, communications, smart grid, machine learning, circuit design, and other applications.

- However, in the era of data deluge, software-based optimization solvers suffer from limited scalability in high-dimensional data regimes.

- Therefore, a new technique or a new solver is imperative to conquer these limitations.

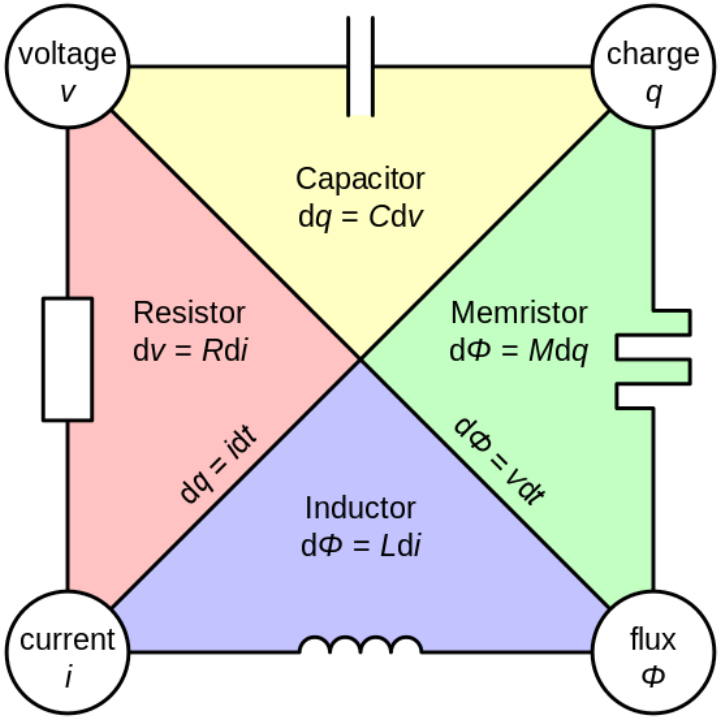
- The recently invented memristor crossbar can potentially resolve those limitations efficiently.

2. Background

- Memristor & Memristor Crossbar
- Convex Optimization
- Second-Order Cone Programming (SOCP)
- Homogeneous Quadratically Constrained Quadratic Programming (QCQP)

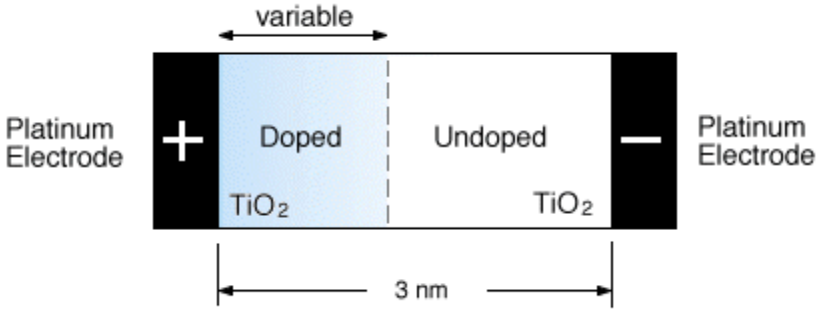
2.1. Memristor

- Introduced as the forth circuit element by L. Chua in 1970s.



2.1. Memristor

- Found by HP in 2008.
- Could “memorize” its most recent history.
- Resistance can be altered by applying different voltage

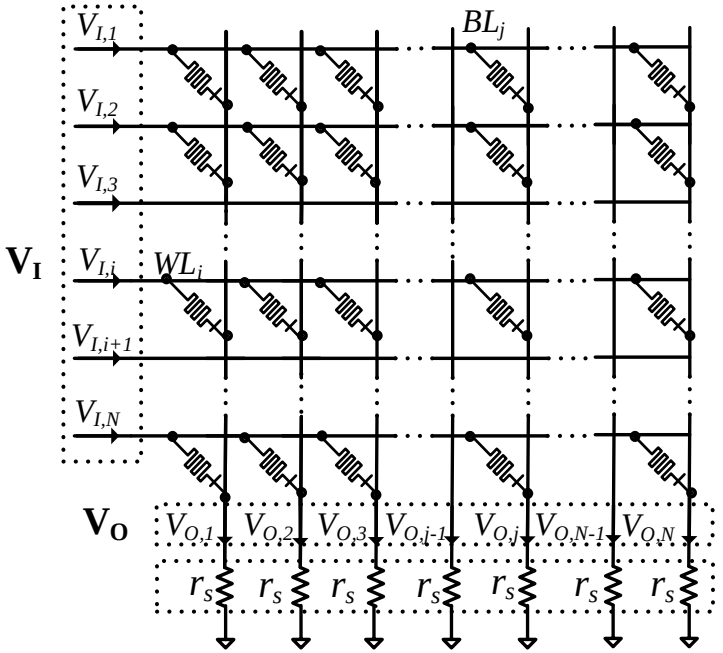


2.2 Memristor Crossbar

- Computing in analog domain
- Matrix-vector multiplication and solving system of linear equations in O(1) time complexity

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

$$\mathbf{V_O} = \mathbf{C} \cdot \mathbf{V_I}$$



2.3 Convex Optimization

- Convex optimization is a research field aiming to find the optimal solution for the problem of minimizing a convex objective function and subjecting to convex constraints.

- Its standard form consists of three parts:
  - an objective function which must be a convex function
  - a set of inequality constraints which must be convex as well
  - a set of equality constraints which must remain affine

2.3 Convex Optimization

- Therefore, a convex minimization problem is written as:

$$\begin{aligned} &\text{minimize} && f_0(\boldsymbol{x}) \\ &\text{subject to:} && f_i(\boldsymbol{x}) \leq 0 \ (i = 1, \ldots, m), \\ & && h_i(\boldsymbol{x}) = 0 \ (i = 1, \ldots, p) \end{aligned}$$

where the optimization variables $\boldsymbol{x} \in \mathbb{R}^n$, and $f_0, f_1, \ldots, f_m$ are convex functions: $\mathbb{R}^n \to \mathbb{R}$.

- A convex function can be written as:

$$f_i(\theta x + (1-\theta)y) \leq \theta f_i(x) + (1-\theta) f_i(y)$$

where $0 \leq \theta \leq 1$.

2.4 Second-Order Cone Programming (SOCP)

- SOCP is a kind of essential convex programs to minimize a linear function over a set of linear constraints and the product of second-order cones.

- It has wide applications in resource allocation in wireless communication networks, high-performance computing, smart grid, etc.

2.4 Second-Order Cone Programming (SOCP)

- An SOCP can be formulated as: :

$$\begin{aligned} &\text{minimize} && \boldsymbol{c}^T \boldsymbol{x} \\ &\text{subject to:} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ & && \left\| \boldsymbol{x}_{1:(n-1)} \right\|_2 \leq x_n, \end{aligned}$$

where $\boldsymbol{x}$ is the optimization variable, $\boldsymbol{x}_{1:(n-1)}$ is the vector that consists of the first (n-1) entries of $\boldsymbol{x}$, x_n is the n-th entry of $\boldsymbol{x}$. The last constraint represents a second-order cone in $\mathbb{R}^n$.

2.5 Homogeneous Quadratically Constrained Quadratic Programming (QCQP)

- If the objective function and the inequality constraints are convex quadratic, then it is called a quadratically constrained quadratic problem (QCQP).

- A QCQP is homogeneous if all quadratic functions have no linear term.

- Homogenous QCQPs were commonly used to address the problems of resource management in signal processing, such as optimal power allocation for linear coherent estimation and optimal spectrum sharing in MIMO cognitive radio networks.

2.5 Homogeneous Quadratically Constrained Quadratic Programming (QCQP)

- A homogeneous QCQP problem has the form as below:

$$\begin{aligned} &\text{minimize} && \boldsymbol{x}^T \boldsymbol{P}_0 \boldsymbol{x} \\ &\text{subject to:} && \boldsymbol{x}^T \boldsymbol{P}_i \boldsymbol{x} \leq \boldsymbol{r}_i \ (i = 1, \dots, m), \\ & && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned}$$

it can be converted to an SOCP problem.

3. Memristor Crossbar-Based Framework For Solving Convex Optimization Problems

- Alternating Direction Method of Multipliers (ADMM)
- Solving SOCP Using Memristor Crossbar via ADMM
- Memristor Conductance Matrix Mapping and Negative Elements Eliminating
- Procedure of Memristor Crossbar-Based Framework for Solving SOCP Problems
- Computational Complexity Analysis
- Memristor-Based Framework for Solving Homogeneous QCQP Problems

3.1 Alternating Direction Method of Multipliers (ADMM)

- The major advantage of ADMM is that it can split the original problem into a set of problems of solving linear systems.

- ADMM solves convex problems of the form:

$$\text{minimize } f(\boldsymbol{x}) + g(\boldsymbol{y}) \text{ subject to: } \boldsymbol{x} = \boldsymbol{y}$$

where *f* and *g* may be non-smooth or take infinite values to encode implicit constraints.

3.1 Alternating Direction Method of Multipliers (ADMM)

- ADMM is an iterative method. Its k-th iteration is:

$$\boldsymbol{x}^{(k+1)} = \arg\min_{\boldsymbol{x}}\left(f(\boldsymbol{x}) + g(\boldsymbol{y}) + (\rho/2)||\boldsymbol{x} - \boldsymbol{y}^{(k)} + (1/\rho)\boldsymbol{\mu}^{(k)}||_2^2\right)$$

$$\boldsymbol{y}^{(k+1)} = \arg\min_{\boldsymbol{y}}\left(f(\boldsymbol{x}) + g(\boldsymbol{y}) + (\rho/2)||\boldsymbol{x}^{(k+1)} - \mathbf{y} + \left(1/\rho\right)\boldsymbol{\mu}^{(k)}||_2^2\right)$$

$$\boldsymbol{\mu}^{(k+1)} = \boldsymbol{\mu}^{(k)} + \rho(\boldsymbol{x}^{(k+1)} - \boldsymbol{y}^{(k+1)})$$

where $\rho > 0$ is the step size parameter, and $\boldsymbol{\mu}$ is the dual variable associated with the constraint $x = y$.

3.2 Solving SOCP Using Memristor Crossbar via ADMM

$$\begin{bmatrix} \mathbf{I}_{n\times n} & \boldsymbol{A}^{\boldsymbol{T}} \\ \boldsymbol{A} & \mathbf{0}_{m\times m} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}^{(k)} \\ \boldsymbol{b} \end{bmatrix} \tag{1}$$

$$\boldsymbol{u}^{(k)} \equiv \boldsymbol{y}^{(k)} - (1/\rho)(\boldsymbol{\mu}^{(k)} + \boldsymbol{c}) \tag{2}$$

$$\boldsymbol{y}^{(k+1)} = \begin{cases} \mathbf{0}_n & \left\|\boldsymbol{w}^{(k)}\right\|_2 \le -s \\ \boldsymbol{v}^{(k)} & \left\|\boldsymbol{w}^{(k)}\right\|_2 \le s \\ \left(\frac{1}{2}\right)\left(1 + \frac{s}{\|\boldsymbol{w}\|_2}\right)[\boldsymbol{w}^T, \|\boldsymbol{w}\|_2]^T & \left\|\boldsymbol{w}^{(k)}\right\|_2 \ge |s| \end{cases} \tag{3}$$

$$\boldsymbol{v}^{(k)} = \left[\left(\boldsymbol{w}^{(k)}\right)^T, s\right]^T \tag{4}$$

$$\boldsymbol{v}^{(k)} \equiv \boldsymbol{x}^{(k+1)} + (1/\rho)\boldsymbol{\mu}^{(k)} \tag{5}$$

$$\boldsymbol{\mu}^{(k+1)} = \boldsymbol{\mu}^{(k)} + \rho(\boldsymbol{x}^{(k+1)} - \boldsymbol{y}^{(k+1)}) \tag{6}$$

By iteratively solving equations (1)-(6), the SOCP problems can be solved.

3.3 Memristor Conductance Matrix Mapping and Negative Elements Eliminating

- For using a memristor crossbar to represent $\boldsymbol{C} = \begin{bmatrix} \mathbf{I}_{n\times n} & \boldsymbol{A}^{\boldsymbol{T}} \\ \boldsymbol{A} & \mathbf{0}_{m\times m} \end{bmatrix}$

Equation $\mathbf{C} = \boldsymbol{D} \cdot \boldsymbol{G}^{T} = diag(d_1, \ldots, d_N) \cdot \begin{bmatrix} g_{1,1} & \cdots & g_{1,N} \\ \vdots & \ddots & \vdots \\ g_{N,1} & \cdots & g_{N,N} \end{bmatrix}^{T}$ may be used to convert $\boldsymbol{C}$ to the memristor conductance matrix $\boldsymbol{G}$.

- However, it is over-complicated to use above equation to perform the mapping. Hence, we adopt a fast approximation: $g_{i,j} = c_{i,j} \cdot g_{max}$, where g_{max} is the maximum conductance among memristors in the memristor crossbar.

$$\boldsymbol{G} = g_{max} \cdot \begin{bmatrix} \mathbf{I}_{n\times n} & \boldsymbol{A}^{\boldsymbol{T}} \\ \boldsymbol{A} & \mathbf{0}_{m\times m} \end{bmatrix}$$

3.3 Memristor Conductance Matrix Mapping and Negative Elements Eliminating

- Since the memristance value cannot be a negative number, effective techniques are necessary to eliminate these negative elements in matrix A.

- Consider a linear system $\boldsymbol{Ax} = \boldsymbol{b}$, and suppose that $a_{i,j}$ is a negative element in $\boldsymbol{A}$. The equation in the *i*-th row:

$$a_{i,1}x_1 + \cdots + a_{i,j}x_j + \cdots + a_{i,n}x_n = b_i$$

is equivalent to:

$$\begin{cases} a_{i,1}x_1 + \cdots + 0 \cdot x_j + \cdots + a_{i,n}x_n + \left(-a_{i,j}\right)z_j = b_i \\ x_j + z_j = 0 \end{cases}$$

3.3 Memristor Conductance Matrix Mapping and Negative Elements Eliminating

- Hence, a negative element can be eliminated by setting it to zero and introducing one more row and one more column. Thus, the linear system $\boldsymbol{Ax} = \boldsymbol{b}$ can be written as:

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,j} & \cdots & a_{1,n} & 0 \\ \vdots & \cdots & \cdots & \cdots & \vdots & 0 \\ a_{i,1} & \cdots & 0 & \cdots & a_{i,n} & -a_{i,j} \\ \vdots & \cdots & \cdots & \cdots & \vdots & 0 \\ a_{n,1} & \cdots & a_{n,j} & \cdots & a_{n,n} & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_j \\ \vdots \\ x_n \\ z_j \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_j \\ \vdots \\ b_n \\ 0 \end{bmatrix}$$

3.3 Memristor Conductance Matrix Mapping and Negative Elements Eliminating

- After applying the above technique to $\boldsymbol{C}$, the linear system can be reformulated as $\boldsymbol{M} \cdot \boldsymbol{s} = \boldsymbol{r}$:

$$\begin{bmatrix} \boldsymbol{I}_{n\times n} & \boldsymbol{A}^{\boldsymbol{T'}}_{n\times m} & \boldsymbol{0}\ or\ \boldsymbol{A}^{\boldsymbol{T''}}_{n\times 2k} \\ \boldsymbol{A}'_{m\times n} & \boldsymbol{0}_{m\times m} & \boldsymbol{0}\ or\ \boldsymbol{A}''_{m\times 2k} \\ \boldsymbol{A}^{\boldsymbol{I}}_{2k\times n} & \boldsymbol{A}^{\boldsymbol{TI}}_{2k\times m} & \boldsymbol{0}\ or\ \boldsymbol{I}_{2k\times 2k} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \\ \boldsymbol{z} \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}^{(k)} \\ \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix}$$

- The memristor conductance matrix of $\boldsymbol{C'}$ is set by: $\boldsymbol{G'} = g_{max} \cdot \boldsymbol{M}$

3.4 Procedure of Memristor Crossbar-Based Framework for Solving SOCP Problems

Detailed Procedure of the Proposed Memristor Crossbar-Based Framework for Solving SOCP Problems

Input: Matrix $\boldsymbol{A}$, vectors $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{u}^{(0)}$, $\boldsymbol{\mu}^{(0)}$, constant ε, ρ, k.

Output: Vector x, y

Initialize $\boldsymbol{x}$, $\boldsymbol{y}$ with arbitrary elements, $k = 0$.

Construct matrix $\boldsymbol{M}$ and vector $\boldsymbol{r}$ in (22) based on $\boldsymbol{A}$.

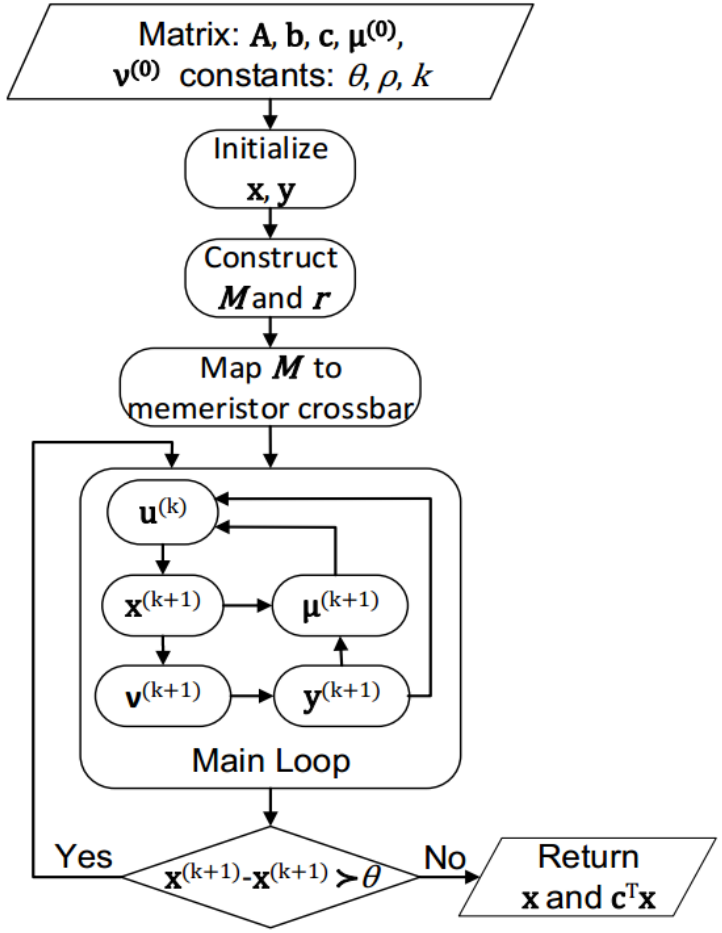
Map $\boldsymbol{M}$ to memristor crossbar according to $\boldsymbol{G'} = g_{max} \cdot \boldsymbol{M}$.

do:

1) Solve $\boldsymbol{x}^{(k+1)}$: read the solution $\boldsymbol{V_I}$ from the memristor crossbar, then according to (24), $\boldsymbol{x}^{(k+1)} = \frac{g_s}{g_{max}} \boldsymbol{V_I}_{(1:n)}$.
2) Calculate $\boldsymbol{v}^{(k)} \equiv \boldsymbol{x}^{(k+1)} + (1/\rho)\boldsymbol{\mu}^{(k)}$ using summing amplifier, and then construct $\boldsymbol{w}^{(k)}$ and s according to $\boldsymbol{v}^{(k)} = \left[\left(\boldsymbol{w}^{(k)}\right)^T, s\right]^T$.
3) Calculate $\boldsymbol{y}^{(k+1)}$ in (15) by calculating $\boldsymbol{w}^{(k)}$ with peripheral circuits.
4) Update $\boldsymbol{\mu}^{(k+1)}$ using summing amplifier according to (7c).
5) Update $\boldsymbol{u}^{(k+1)}$ using summing amplifier: $\boldsymbol{u}^{(k+1)} = \boldsymbol{y}^{(k+1)} - (1/\rho)(\boldsymbol{\mu}^{(k+1)} + \boldsymbol{c})$.
6) $k = k + 1$.

While $\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \succ \varepsilon$

Return $\boldsymbol{x}$, $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$.



3.5 Computational Complexity Analysis

- The algorithm-hardware co-optimization of memristor-based framework proposed in this paper is an iterative solution framework.
- In each iteration, the complexity of solving $\boldsymbol{M} \cdot \boldsymbol{s} = \boldsymbol{r}$ with the memristor crossbar is O(1) and that of calculating $\boldsymbol{y}^{(k+1)}$ with peripheral circuits is O(N).
- Hence the framework presents an overall solution complexity of pseudo-O(N), or O(MN) if M represents the number of iterations, which is a significant improvement compared with the state-of-the-art software-based solution of O($N^{3.5}$) - O(N^4).
- The complexity of initialization of the matrix in the memristor crossbar is O(N^2), or lower for sparse matrices which are very common in (large-scale) optimization problems.

3.6 Memristor-Based Framework for Solving Homogeneous QCQP Problems

- According to the eigenvalue decomposition of $\boldsymbol{P}_i$, we have $\boldsymbol{Q}_i$ such that $\boldsymbol{P}_i = \boldsymbol{Q}_i^T \boldsymbol{Q}_i$. Upon defining: $\boldsymbol{z}_i = \begin{bmatrix} \boldsymbol{Q}_i \boldsymbol{x} \\ \sqrt{\boldsymbol{r}_i} \end{bmatrix} \in \mathbb{R}^{n+1}$,

The homogeneous QCQP problem can be expressed as a convex program with second-order cone constraints:

$$
\begin{aligned}
\text{minimize} \quad & \boldsymbol{x}^T \boldsymbol{P}_0 \boldsymbol{x} \\
\text{subject to:} \quad & \|[\boldsymbol{z}_i]_{1:n}\|_2 \leq [\boldsymbol{z}_i]_{n+1} \ (i = 1, \dots, m) \\
& \boldsymbol{z}_i - \boldsymbol{C}_i \boldsymbol{x} = \boldsymbol{d}_i \ (i = 1, \dots, m) \\
& \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},
\end{aligned}
$$

where the optimization variables are $\boldsymbol{x}$ and $\boldsymbol{z}_i$, and $C_i = \left[Q_i^T, 0\right]^T$, $d_i = [0^T, \sqrt{\boldsymbol{r}_i}]^T$.

4. Experiments & Analysis

- Experiment Methods
- Experiment Results of Solving SOCP problems
- Experiment Results of Solving Homogeneous QCQP problems

4.1 Experiment Methods

- All input matrices are randomly generated using *sprandn* provided by Matlab to generate sparse matrices, since the constraint matrix of an SOCP or homogeneous QCQP problem is sparse in general.

- These inputs are firstly sent to the CVX tool, which is a well-known and widely recognized software solver for convex optimizations.

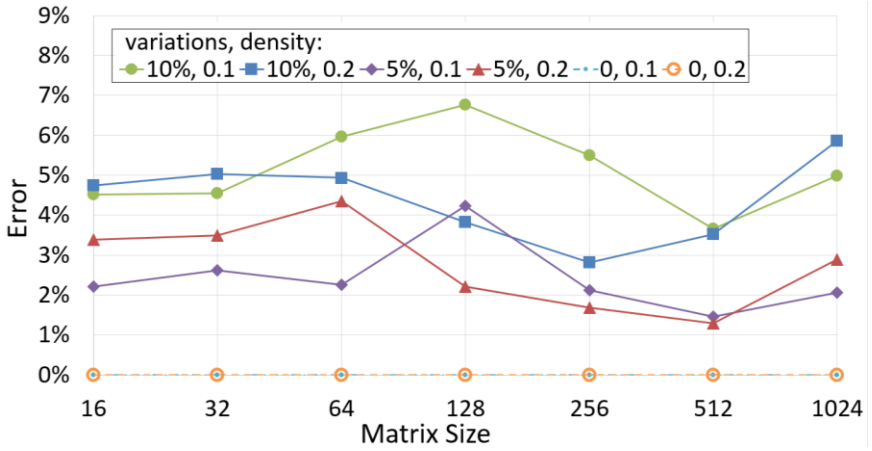
- After the randomly generated problems are verified to be feasible and bounded, our solver is then utilized to solve the problems.

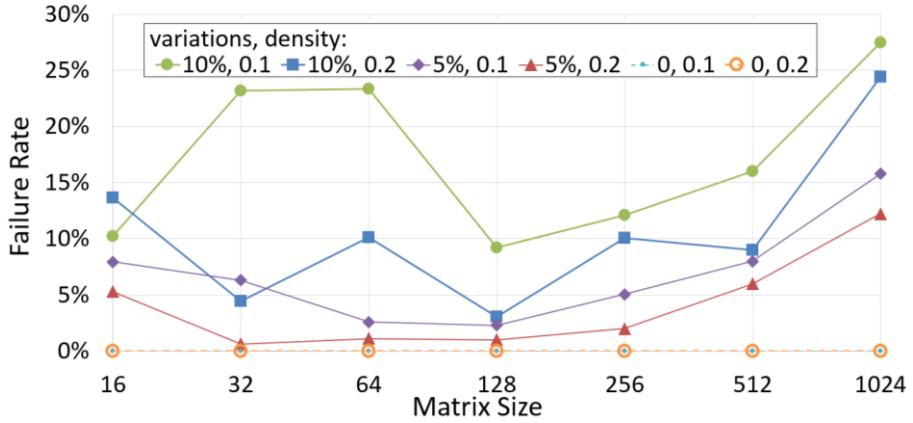
- Element writing inaccuracies, random process variations, and other variations are considered and added to the randomly generated matrices.

4.2 Experiment Results of Solving SOCP problems

- The condenser the input matrix ***A*** is, the more reliabilities our hardware-based solution framework can guarantee.

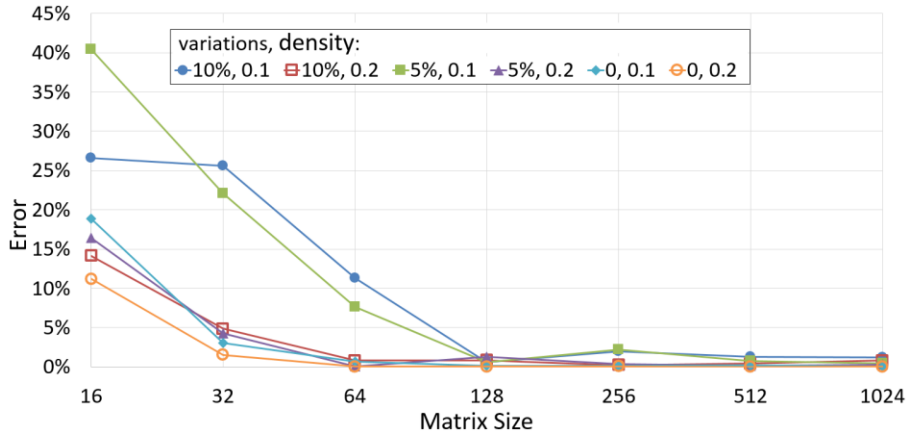
- Basically, our algorithm can achieve high accuracy (95%) and success rates (85%) when the process variations are restricted below 5%.

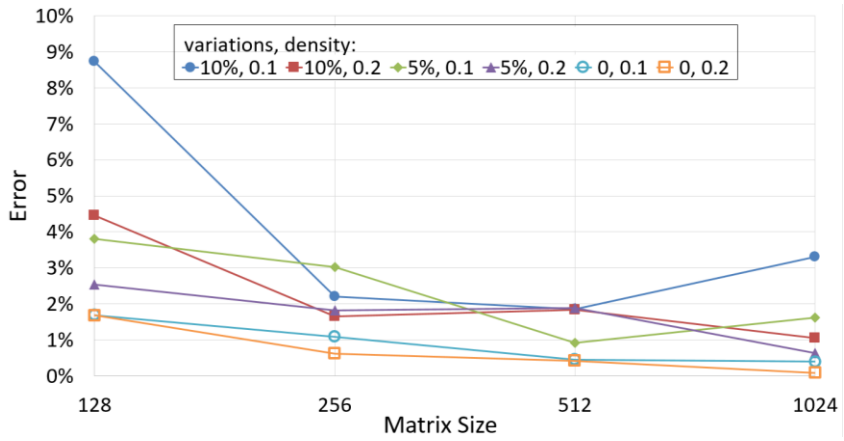




4.3 Experiment Results of Solving Homogeneous QCQP problems

- Even though we significantly reduce the number of iterations by relaxing the tolerance level, up to 96% accuracy can be obtained.
- The impact of process variations is quite limited here and no one failure is found during the experiment process.





5. Conclusion

- A framework for solving SOCP and homogeneous QCQP problems is proposed in this work.
- The overall time complexity of solving SOCP and homogeneous QCQP problems are both pseudo-O(N).
- A maximum of 1.57×10^5X estimated improvement in speed is achieved compared with the CVX tool executed on an Intel I7 server.
- Higher than 94% accuracy can be achieved when solving SOCP problems.
- Higher than 96% accuracy when solving homogeneous QCQP problems.

Thank You !