SUPPLEMENTARY MATERIALS A

The codes provided here will be updated in the Github repository from the link below:
https://github.com/yjnoori/Phase-Change-Memory

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %**Welcome to Noori's memristor array simulator**%
              %**Selector + Memristor arrays**%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% The function line can be used with the supplied code for running
% functions
% this is useful for running the code in parallel processors using the
% Matlab Parallel Computing toolbox
%
% function [Measurement_V] = Code_Measured_Vvsn_if(n_if)
%
% Setting up the resistances for the matrix array
% file = matfile('Random100x100bits')
% R = file.R

R(1:100,1:100) = 0

% Assigning the input values
% Choosing the position of the read cell
Read_row = 1
Read_coloumn = 1
% Assigning the bit value of the read cell
R(Read_row, Read_coloumn) = 0
% Assigning the resistance of R_low and R_high for 1 and 0 respectively
R_h = 1000000
R_l = 10000
% Assigning sense resistor value
R_Sense = 1000
% Assigning word and bit lines resistance
R_WL = 5
R_BL = 5

error = 2
% Choosing the accuracy of the simulated results
pass = 0.0001

% Assigning selector input parameters
T = 300
I_S = 10^-12
n_if = 1.5
V_T =  1.3806 * 10^-23 * T / (1.602*10^-19)

m = size (R,1);
n = size (R,2);

% Assigning the sense resistors for WL and BL
R_S_WL1(1:m) = 10
R_S_WL1(Read_row) = 10
R_S_BL1(1:n) = 10
R_S_BL1(Read_coloumn) = R_Sense
R_S_WL2(1:m) = 10^8
R_S_BL2(1:n) = 10^8
```

```
% Assigning applied voltages at the WLs and BLs
V_APP_WL1(1:m) = 0.5
V_APP_WL1(Read_row) = 1
V_APP_BL1(1:n) = 0.5
V_APP_BL1(Read_coloumn) = 0
V_APP_WL2(1:m) = 0
V_APP_WL2(Read_row) = 0
V_APP_BL2(1:n) = 0
V_APP_BL2(Read_coloumn) = 0

%-------------------------------------------------

% Assigning initial voltage guess
parfor i = 1:m
        for j = 1:n
            V(i,j) = V_APP_WL1(i) - V_APP_BL1(j)
        end
end
ct = 0

% A while loop is used to keep iterating the code until a desired accuracy
% is achieved
while error > pass

    % Calculating the array resistances
    parfor i = 1:m
        for j = 1:n
            if R(i,j) == 0
                    I_first =
(n_if*V_T/R_h)*lambertw(((I_S*R_h)/(n_if*V_T))*exp(V(i,j)/(n_if*V_T)))
                    I_second =
(n_if*V_T/R_h)*lambertw(((I_S*R_h)/(n_if*V_T))*exp((V(i,j)+0.01)/(n_if*V_T)
))
                    RR(i,j) = 0.01/(I_second-I_first)

            else
                    I_first =
(n_if*V_T/R_l)*lambertw(((I_S*R_l)/(n_if*V_T))*exp(V(i,j)/(n_if*V_T)))
                    I_second =
(n_if*V_T/R_l)*lambertw(((I_S*R_l)/(n_if*V_T))*exp((V(i,j)+0.01)/(n_if*V_T)
))
                    RR(i,j) = 0.01/(I_second-I_first)

            end
        end
    end


% Define the matrix A

    tv = cell(1,m);
    parfor i = 1:m
        A = sparse((zeros(n,n)))
        for j = 1:n

            if j == 1

               A(j,j) = (1/(R_S_WL1(i)))+1/(RR(i,j))+1/R_WL
```

```
            A(j,j+1) = -1/R_WL

        elseif j == n
            A(j,j) = (1/(R_S_WL2(i)))+1/(RR(i,j))+1/R_WL
            A(j,j-1) = -1/R_WL


        else
            A(j,j-1) = -1/R_WL;
            A(j,j) = (1/RR(i,j))+(2/R_WL)
            A(j,j+1) = -1/R_WL

        end

    end
    tv{i} = A;
end
AA = blkdiag(tv{:})
tv = cell(1,m);

% Define the matrix B
parfor i = 1:m
    B = sparse((zeros(n,n)))
    for j = 1:n

        B(j,j) = -1/(RR(i,j))

    end

    tv{i} = B;
end
BB = blkdiag(tv{:})

% Define the matrix C

tv = cell(1,n);
parfor j = 1:n
    C = sparse(zeros(n,m*n))
    for i = 1:m

        C(i,n*(i-1)+j) = 1/(RR(i,j))

    end

    tv{j} = C;
end
CC = vertcat(tv{:})

% Define the matrix D

tv = cell(1,n);
parfor j = 1:n
    D = sparse((zeros(n,m*n)))
    for i = 1:m

        if i == 1

            D(i,j) = (-1/R_S_BL1(j))+(-1/R_BL)+(-1/RR(i,j))
```

```
            D(i, n*i+j) = 1/R_BL

        elseif i == m

            D(i,n*(i-2)+j) = 1/R_BL
            D(i, n*(i-1)+j) = (-1/R_S_BL2(j))+(-1/RR(i,j))+(-1/R_BL)

        else

          D(i,n*(i-2)+j) = 1/R_BL
          D(i, n*(i-1)+j) = (-1/R_BL)+(-1/RR(i,j))+(-1/R_BL)
          D(i, n*i+j) = 1/R_BL

        end

    end

    tv{j} = D;
end
DD = vertcat(tv{:})

% Define the coloumn vector EW

tv = cell(1,m);
parfor i = 1:m
    EW = sparse((zeros(n,1)))
    for j = 1:n

        if j == 1

            EW(j,1) = V_APP_WL1(i)/R_S_WL1(i)

        elseif j == n

            EW(j,1) = V_APP_WL2(i)/R_S_WL2(i)

        else
            EW(j,1) = 0
        end

    end
    tv{i} = EW;
end
EWEW = vertcat(tv{:})

% Define the coloumn vector EB

tv = cell(1,n);
parfor j = 1:n
    EB = sparse(zeros(m,1))
    for i = 1:m

        if i == 1

            EB(i,1) = -V_APP_BL1(j)/R_S_BL1(j)

        elseif i == m
```

```
                EB(i,1) = -V_APP_BL2(j)/R_S_BL2(j)

            else
                EB(i,1) = 0
            end

        end
        tv{j} = EB;
    end
    EBEB = vertcat(tv{:})


    % Calculating the V coloumn vector [VV]
    VV = [AA   BB
         CC   DD]  \  [EWEW
                       EBEB]

    %Reshaping the voltage vector into a square matrix of bitline and word
line
    %voltages

    V_WL_sq = sparse((reshape(VV(1:m*n),[m,n])).')
    V_BL_sq = sparse((reshape(VV(((m*n)+1):(2*m*n)),[m,n])).')


    %Calculate the currents throughout the array
    II = sparse(zeros(m,n))
    PP = sparse(zeros(m,n))

    parfor i = 1:m
        I = sparse((zeros(1,n)))
        P = sparse((zeros(1,n)))
        for j = 1:n
                % Calculating the cells' current using the node voltages
obtained
                % previously
                I(j) = (V_WL_sq(i,j) - V_BL_sq(i,j))/RR(i,j)

                % Calculating the cells' power
                P(j) = I(j)*(V_WL_sq(i,j) - V_BL_sq(i,j))

        end
        II(i,:) = I
        PP(i,:) = P
    end
    V_temp = abs((V_WL_sq - V_BL_sq)-V)
    error = max(V_temp(:))
    V = V_WL_sq - V_BL_sq
  ct = ct +1
end
    % Calculating the percentage of the current and power in the matrix.
    Per_II = (II/max(II(:)))*100
    Per_PP = (PP/max(PP(:)))*100
    imagesc(PP)

    % Calculating the voltage at sense resistor
```

```
    Measurement_V = (sum(II(1:n, Read_coloumn))) * R_Sense
    Measurement_V = full(Measurement_V)

    % Calculating the resistance of the selected cell
    Measured_RR = (V_APP_WL1(Read_row)-Measurement_V)/(sum(II(1:n,
Read_coloumn)))
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %**Welcome to Noori's memristor array simulator**%
              %**Memristor devices arrays**%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function [Measurement_V] = Code_Measured_RvsI_S(I_S)
% %
% file = matfile('Random10x10bits')
% R = file.R

R(1:100,1:100) = 0

R_h = 1000000
R_l = 10000
Read_row = 1
Read_coloumn = 1
R(Read_row, Read_coloumn) = 0
R_Sense = 1000
R_WL = 1
R_BL = 1

m = size (R,1);
n = size (R,2);

error = 2
pass = 0.0001

R_S_WL1(1:m) = 10
R_S_WL1(Read_row) = 10
R_S_BL1(1:n) = 10
R_S_BL1(Read_coloumn) = R_Sense
R_S_WL2(1:m) = 10^8
R_S_BL2(1:n) = 10^8

V_APP_WL1(1:m) = 0.5
V_APP_WL1(Read_row) = 1
V_APP_BL1(1:n) = 0.5
V_APP_BL1(Read_coloumn) = 0
V_APP_WL2(1:m) = 0
V_APP_WL2(Read_row) = 0
V_APP_BL2(1:n) = 0
V_APP_BL2(Read_coloumn) = 0

%-------------------------------------------------

parfor i = 1:m
        for j = 1:n
            V(i,j) = V_APP_WL1(i) - V_APP_BL1(j)
        end
end
ct = 0

while error > pass

    parfor i = 1:m
        for j = 1:n
            if R(i,j) == 0
```

```
                RR(i,j) = R_h

        else

                RR(i,j) = R_l

        end
    end
end


%Define A

tv = cell(1,m);
parfor i = 1:m
    A = sparse((zeros(n,n)))
    for j = 1:n

        if j == 1

           A(j,j) = (1/(R_S_WL1(i)))+1/(RR(i,j))+1/R_WL
           A(j,j+1) = -1/R_WL

        elseif j == n
           A(j,j) = (1/(R_S_WL2(i)))+1/(RR(i,j))+1/R_WL
           A(j,j-1) = -1/R_WL


        else
           A(j,j-1) = -1/R_WL;
           A(j,j) = (1/RR(i,j))+(2/R_WL)
           A(j,j+1) = -1/R_WL

        end

    end
    tv{i} = A;
end
AA = blkdiag(tv{:})

%Define B

tv = cell(1,m);
parfor i = 1:m
    B = sparse((zeros(n,n)))
    for j = 1:n

        B(j,j) = -1/(RR(i,j))

    end

    tv{i} = B;
end
BB = blkdiag(tv{:})

%Define C
```

```
tv = cell(1,n);
parfor j = 1:n
    C = sparse(zeros(n,m*n))
    for i = 1:m

        C(i,n*(i-1)+j) = 1/(RR(i,j))

    end

    tv{j} = C;
end
CC = vertcat(tv{:})

%Define D, might need double checking

tv = cell(1,n);
parfor j = 1:n
    D = sparse((zeros(n,m*n)))
    for i = 1:m

        if i == 1

           D(i,j) = (-1/R_S_BL1(j))+(-1/R_BL)+(-1/RR(i,j))
           D(i, n*i+j) = 1/R_BL

        elseif i == m

           D(i,n*(i-2)+j) = 1/R_BL
           D(i, n*(i-1)+j) = (-1/R_S_BL2(j))+(-1/RR(i,j))+(-1/R_BL)

        else

          D(i,n*(i-2)+j) = 1/R_BL
          D(i, n*(i-1)+j) = (-1/R_BL)+(-1/RR(i,j))+(-1/R_BL)
          D(i, n*i+j) = 1/R_BL

        end

    end

    tv{j} = D;
end
DD = vertcat(tv{:})

%Define EW

tv = cell(1,m);
parfor i = 1:m
    EW = sparse((zeros(n,1)))
    for j = 1:n

        if j == 1

            EW(j,1) = V_APP_WL1(i)/R_S_WL1(i)

        elseif j == n
```

```
            EW(j,1) = V_APP_WL2(i)/R_S_WL2(i)

        else
            EW(j,1) = 0
        end

    end
    tv{i} = EW;
end
EWEW = vertcat(tv{:})

%Define EB

tv = cell(1,n);
parfor j = 1:n
    EB = sparse(zeros(m,1))
    for i = 1:m

        if i == 1

            EB(i,1) = -V_APP_BL1(j)/R_S_BL1(j)

        elseif i == m

            EB(i,1) = -V_APP_BL2(j)/R_S_BL2(j)

        else
            EB(i,1) = 0
        end

    end
    tv{j} = EB;
end
EBEB = vertcat(tv{:})


VV = [AA   BB
     CC   DD]  \  [EWEW
                   EBEB]

%Reshaping the voltage vector into a square matrix of bitline and word
line
%voltages

V_WL_sq = sparse((reshape(VV(1:m*n),[m,n])).')
V_BL_sq = sparse((reshape(VV(((m*n)+1):(2*m*n)),[m,n])).')


%Calculate the currents throughout the array
II = sparse(zeros(m,n))
PP = sparse(zeros(m,n))

parfor i = 1:m
    I = sparse((zeros(1,n)))
    P = sparse((zeros(1,n)))
    for j = 1:n
```

```
                I(j) = (V_WL_sq(i,j) - V_BL_sq(i,j))/RR(i,j)

                P(j) = I(j)*(V_WL_sq(i,j) - V_BL_sq(i,j))


        end
        II(i,:) = I
        PP(i,:) = P
    end
    V_temp = abs((V_WL_sq - V_BL_sq)-V)
    error = max(V_temp(:))
    V = V_WL_sq - V_BL_sq
  ct = ct +1
end

    Per_II = (II/max(II(:)))*100
    Per_PP = (PP/max(PP(:)))*100
    imagesc(PP)

    %Measured voltage at sense resistor
    Measurement_V = (sum(II(1:n, Read_coloumn))) * R_Sense
    Measurement_V = full(Measurement_V)

    Measured_RR = 1/(sum(II(1:n, Read_coloumn)))
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %**Welcome to Noori's memristor array simulator**%
           %**non-linear devices only arrays**%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% function [Measurement_V] = Code_Measured_RvsI_S(I_S)

R(1:100,1:100) = 0

m = size (R,1);
n = size (R,2);


error = 2
pass = 0.0001


T = 300
V_T =  1.3806 * 10^-23 * T / (1.602*10^-19)
I_S_h = 10^-14
I_S_l = 10^-13
n_if_h = 1.8
n_if_l = 1.5

Read_row = 1
Read_coloumn = 1
R_Sense = 100
R_WL = 2
R_BL = 2

R_S_WL1(1:m) = 10
R_S_WL1(Read_row) = 10
R_S_BL1(1:n) = 10
R_S_BL1(Read_coloumn) = R_Sense
R_S_WL2(1:m) = 10^8
R_S_BL2(1:n) = 10^8

V_APP_WL1(1:m) = 0.5
V_APP_WL1(Read_row) = 1
V_APP_BL1(1:n) = 0.5
V_APP_BL1(Read_coloumn) = 0
V_APP_WL2(1:m) = 0
V_APP_WL2(Read_row) = 0
V_APP_BL2(1:n) = 0
V_APP_BL2(Read_coloumn) = 0

%-------------------------------------------------

parfor i = 1:m
        for j = 1:n
            V(i,j) = V_APP_WL1(i)
        end
end



ct = 0
```

```
while error > pass

    parfor i = 1:m
        for j = 1:n
            if R(i,j) == 0
            RR(i,j) = 1/((I_S_h/(n_if_h*V_T))*exp(V(i,j)/(n_if_h*V_T)))
            else
            RR(i,j) = 1/((I_S_l/(n_if_l*V_T))*exp(V(i,j)/(n_if_l*V_T)))
            end
        end
    end


    %Define A

    tv = cell(1,m);
    parfor i = 1:m
        A = sparse((zeros(n,n)))
        for j = 1:n

            if j == 1

               A(j,j) = (1/(R_S_WL1(i)))+1/(RR(i,j))+1/R_WL
               A(j,j+1) = -1/R_WL

            elseif j == n
               A(j,j) = (1/(R_S_WL2(i)))+1/(RR(i,j))+1/R_WL
               A(j,j-1) = -1/R_WL


            else
               A(j,j-1) = -1/R_WL;
               A(j,j) = (1/RR(i,j))+(2/R_WL)
               A(j,j+1) = -1/R_WL

            end

        end
        tv{i} = A;
    end
    AA = blkdiag(tv{:})

    %Define B

    tv = cell(1,m);
    parfor i = 1:m
        B = sparse((zeros(n,n)))
        for j = 1:n

            B(j,j) = -1/(RR(i,j))

        end

        tv{i} = B;
    end
    BB = blkdiag(tv{:})
```

```
%Define C

tv = cell(1,n);
parfor j = 1:n
    C = sparse(zeros(n,m*n))
    for i = 1:m

        C(i,n*(i-1)+j) = 1/(RR(i,j))

    end

    tv{j} = C;
end
CC = vertcat(tv{:})

%Define D

tv = cell(1,n);
parfor j = 1:n
    D = sparse((zeros(n,m*n)))
    for i = 1:m

        if i == 1

           D(i,j) = (-1/R_S_BL1(j))+(-1/R_BL)+(-1/RR(i,j))
           D(i, n*i+j) = 1/R_BL

        elseif i == m

           D(i,n*(i-2)+j) = 1/R_BL
           D(i, n*(i-1)+j) = (-1/R_S_BL2(j))+(-1/RR(i,j))+(-1/R_BL)

        else

          D(i,n*(i-2)+j) = 1/R_BL
          D(i, n*(i-1)+j) = (-1/R_BL)+(-1/RR(i,j))+(-1/R_BL)
          D(i, n*i+j) = 1/R_BL

        end

    end

    tv{j} = D;
end
DD = vertcat(tv{:})

%Define EW

tv = cell(1,m);
parfor i = 1:m
    EW = sparse((zeros(n,1)))
    for j = 1:n

        if j == 1

            EW(j,1) = V_APP_WL1(i)/R_S_WL1(i)
```

```
            elseif j == n

                EW(j,1) = V_APP_WL2(i)/R_S_WL2(i)

            else
                EW(j,1) = 0
            end

        end
        tv{i} = EW;
    end
    EWEW = vertcat(tv{:})

    %Define EB

    tv = cell(1,n);
    parfor j = 1:n
        EB = sparse(zeros(m,1))
        for i = 1:m

            if i == 1

                EB(i,1) = -V_APP_BL1(j)/R_S_BL1(j)

            elseif i == m

                EB(i,1) = -V_APP_BL2(j)/R_S_BL2(j)

            else
                EB(i,1) = 0
            end

        end
        tv{j} = EB;
    end
    EBEB = vertcat(tv{:})


    VV = [AA   BB
         CC   DD]  \  [EWEW
                       EBEB]

    %Reshaping the voltage vector into a square matrix of bitline and word
line
    %voltages

    V_WL_sq = sparse((reshape(VV(1:m*n),[m,n])).')
    V_BL_sq = sparse((reshape(VV(((m*n)+1):(2*m*n)),[m,n])).')


    %Calculate the currents throughout the array
    II = sparse(zeros(m,n))
    PP = sparse(zeros(m,n))

    parfor i = 1:m
        I = sparse((zeros(1,n)))
        P = sparse((zeros(1,n)))
        for j = 1:n
```

```
                I(j) = (V_WL_sq(i,j) - V_BL_sq(i,j))/RR(i,j)

                P(j) = I(j)*(V_WL_sq(i,j) - V_BL_sq(i,j))


        end
        II(i,:) = I
        PP(i,:) = P
    end
    V_temp = abs((V_WL_sq - V_BL_sq)-V)
    error = max(V_temp(:))
    V = V_WL_sq - V_BL_sq
    ct = ct + 1
end

    Per_II = (II/max(II(:)))*100
    Per_PP = (PP/max(PP(:)))*100
    imagesc(PP)

    %Measured voltage at sense resistor
    Measurement_V = 0
    for i = 1:m

        Measurement_V = Measurement_V + ((II(i,Read_coloumn))/(((m-
i)*R_BL)+R_Sense))

    end
    Measurement_V = Measurement_V * R_Sense
```