SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Flash Memory based Computing-In-Memory System to Solve Partial Differential Equations

Yang FENG[1], Fei WANG[1], Xuepeng ZHAN[1], Yuan LI[1] & Jiezhi CHEN[1*]

[1]*School of Information Science and Engineering, Shandong University, Qingdao* 266237*, China*

Appendix A Iterative method

$$A=\begin{bmatrix} -4 & 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 1 & \ddots & \ddots & & & & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & & & & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & & & 1 \\ 0 & & & \ddots & \ddots & \ddots & & & 0 \\ 1 & & & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & & & & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & & & & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & 1 & 0 & \dots & 0 & 1 & -4 \end{bmatrix}$$

$$\boldsymbol{A}\cdot\boldsymbol{X}=\boldsymbol{b}$$
$$\boldsymbol{X}^{(k+1)}=\boldsymbol{D}^{-1}((\boldsymbol{A}-\boldsymbol{D})\cdot\boldsymbol{X}^{(k)}-\boldsymbol{b})$$
$$\boldsymbol{X}^{(k+1)}=\boldsymbol{M}\cdot\boldsymbol{X}^{(k)}-\boldsymbol{N}$$
$$\boldsymbol{M}=\boldsymbol{D}^{-1}(\boldsymbol{A}-\boldsymbol{D})$$

$$D=\begin{bmatrix} -4 & 0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & -4 & & & & & & & \vdots \\ \vdots & & -4 & & & & & & \vdots \\ \vdots & & & -4 & & & & & \vdots \\ \vdots & & & & \ddots & & & & \vdots \\ \vdots & & & & & -4 & & & \vdots \\ \vdots & & & & & & -4 & & \vdots \\ \vdots & & & & & & & -4 & 0 \\ 0 & \cdots & \dots & \dots & \dots & \dots & \dots & 0 & -4 \end{bmatrix} \quad M=\begin{bmatrix} 0 & 1/4 & 0 & \dots & 0 & 1/4 & 0 & \dots & 0 \\ 1/4 & & \ddots & & & & \ddots & & \vdots \\ 0 & \ddots & & \ddots & & & & \ddots & 0 \\ \vdots & & \ddots & & \ddots & & & & 1/4 \\ 0 & & & \ddots & & \ddots & & & 0 \\ 1/4 & & & & \ddots & & \ddots & & \vdots \\ 0 & \ddots & & & & \ddots & & \ddots & 0 \\ \vdots & & \ddots & & & & \ddots & & 1/4 \\ 0 & \cdots & 0 & 1/4 & 0 & \dots & 0 & 1/4 & 0 \end{bmatrix}$$

Figure S1 The explicit form of the coefficient matrix has been shown in the figure as matrix M, which only has only four diagonals with nonzero values. This coefficient matrix is transformed from the matrix A processed by the FDM method.

In most cases, we can solve numerical differential equations with the finite difference scheme by formulating equations as $A\cdot x=b$ for matrix solutions, where A represents the coefficient matrix, X the unknown vector $X\,(x_1,x_2,x_3,\cdots,x_{n-1},x_n)$ to be solved, and b the constant vector including boundary conditions. The iteration of the solution for the original equation is constructed by the Jacobi iteration method, which can be written as,

$$X^{(k+1)}=D^{-1}\left((A-D)\cdot X^{(k)}-b\right)$$

where D is the matrix only contains the diagonal elements of A, and equation can be simplified as:

$$X^{(k+1)}=M\cdot X^{(k)}-N$$

This equation defines the iterative relationship of the unknown vector X and the coefficient matrix M, which is implemented in the flash-memory-based hardware system by mapping X with numerical values of pulse durations and M to the cell array with numerical values of the V_{th}-adjustable channel conductance at a constant V_g. By inputting the $X^{(k)}$ obtained in the last iteration into the next iteration in the form of pulse time, the $X^{(k+1)}$ is obtained. The process is then repeated iteratively by feeding $X^{(k+1)}$ to the system as the next input vector until the desired accuracy is achieved.

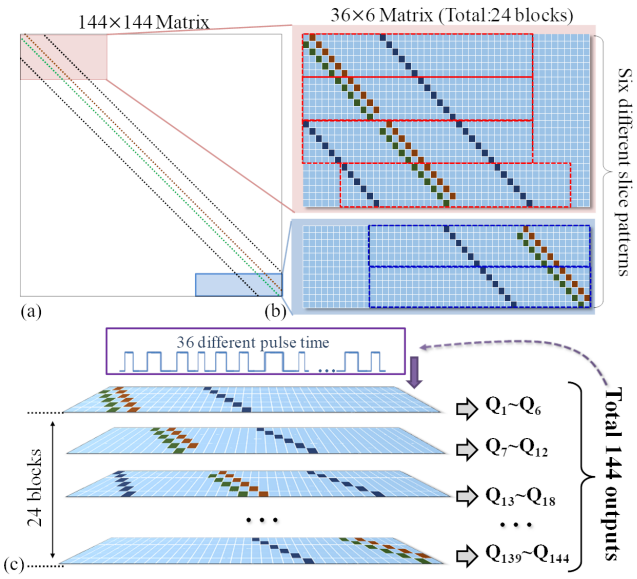

Figure S2 (a) A 144x144 coefficient matrix (from a 12x12 grid) generated by the FDM method. (b) The way we divide the blocks that leads to totally six different slice patterns by enlarging the selected area of (a). (c) Illustrations of mapping slices intercepted from the coefficient matrix into the flash memory arrays. The 144x144 coefficient matrix is sliced into 24 slices with 36 inputs and 6 outputs.

Appendix B Matrix partition method 1

The first scheme is to slice the big matrix into 36x6 matrix slices with 36 inputs and 6 outputs for each matrix, and only intercepted active slices are considered (the slices containing non-zero elements). Thus, the hardware utilization can be largely improved. The 36x6 matrix slices can be mapped onto the 36x6 flash memory arrays. The 36 different pulse time are regarded as input (with the same pulse voltage), and the pulse time value is determined by which column it takes in 144 columns when the coefficient matrix is partitioned. As a result of the 36x6 matrix structure, we obtain 6 outputs represented by the amount of charge. After integrating all the outputs in 24 blocks, there are totally 144 outputs, which can be considered as the next input pulse time after A/D conversion.

Appendix C Matrix partition method 2

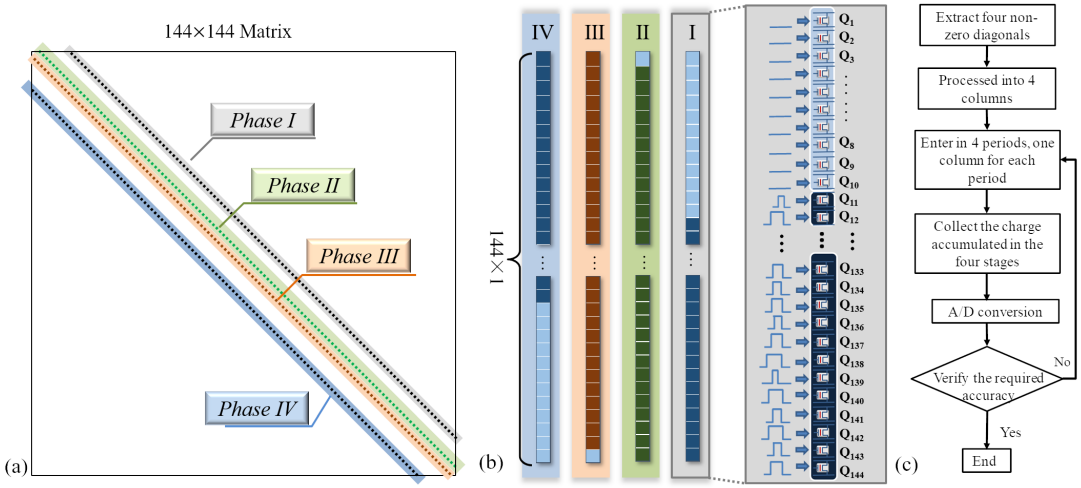

Figure S3 (a) A 144x144 coefficient matrix (from a 12x12 grid) generated by the FDM method. (b) Four different slice patterns of the matrix mapped onto 144x1 flash memory arrays through time multiplexing, where Phase-I 144x1 array implementation is shown in detail. (c) Flow chart of 144x1 matrix iteration process.

* Corresponding author (email: chen.jiezhi@sdu.edu.cn)

Considering that the Jacobi iteration matrix only have nonzero values in the four diagonals, we can extract the four diagonals and convert them to four columns containing nonzero values, as shown in Figure C1(a). To make use of the most of the array, we construct 144x1 matrix with 144 inputs and 144 outputs. The different pulse time is inputted in each of the four periods, and each period enters one of the four columns. After four periods, the accumulated results acquired from the integrators are the output of this iteration. The flow chart (Figure C1(c)) shows the iteration process.

Appendix D Meausured output

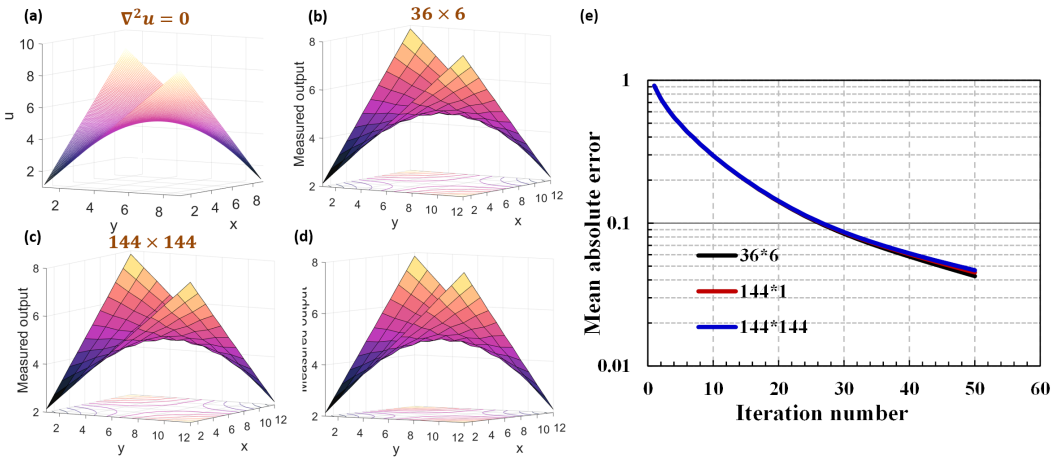

Figure S4 (a) Expected solution of the partial differential equations. (b) Measured output obtained from 36x6 slices after 50 iterations.(c) Measured output obtained from original matrix after 50 iterations. (the slight difference in value is caused by the rough matrix partition.).(d) Measured output obtained from 144x1 matrix after 50 iterations.(e) Evolution of the mean absolute errors (MAE) during the iterations of solving the Laplace's equation by the flash-memory-based PDE solver. Different matrix partitioning methods are compared and the reference values are from the exact numerical solutions by software.

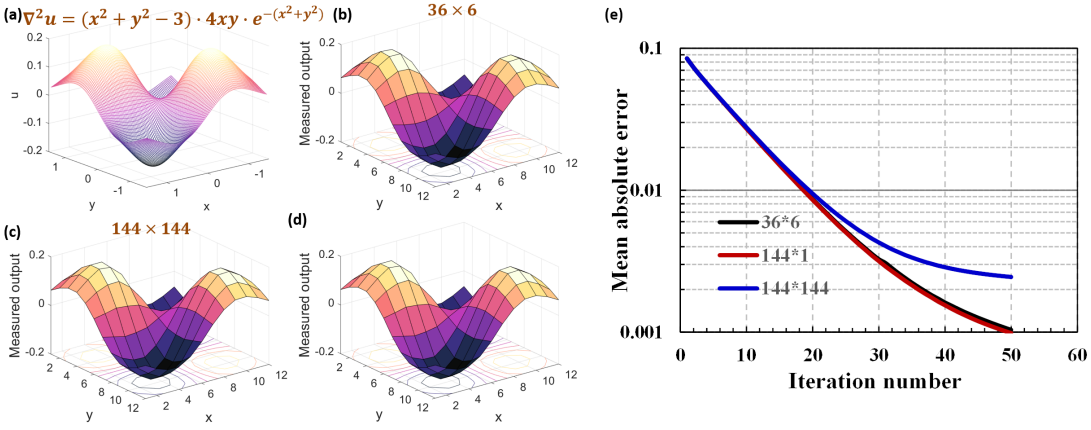

Figure S5 (a) Expected solution of the partial differential equations. (b) Measured output obtained from 36x6 slices after 50 iterations. (c) Measured output obtained from original matrix after 50 iterations.(d) Measured output obtained from 144x1 matrix after 50 iterations. (e) Different matrix partitioning methods are compared and the reference values are from the exact numerical solutions by software.