micromachines

Article

An Efficient and Robust Partial Differential Equation Solver by Flash-Based Computing in Memory

Yueran Qi [1,†], Yang Feng [1,†], Jixuan Wu [1,*], Zhaohui Sun [1], Maoying Bai [1], Chengcheng Wang [1], Hai Wang [1], Xuepeng Zhan [1], Junyu Zhang [2], Jing Liu [3] and Jiezhi Chen [1,*]

[1] School of Information Science and Engineering, Shandong University, Qingdao 266237, China; 202132729@mail.sdu.edu.cn (Y.Q.); feng.yang@mail.sdu.edu.cn (Y.F.); sunzhaohui@mail.sdu.edu.cn (Z.S.); maoy.bai@mail.sdu.edu.cn (M.B.); 202132732@mail.sdu.edu.cn (C.W.); 201800272037@mail.sdu.edu.cn (H.W.)
[2] Neumem Co., Ltd., Hefei 241060, China
[3] Key Laboratory of Microelectronic Devices and Integrated Technology, Institute of Microelectronics of Chinese Academy of Sciences, Beijing 100029, China
* Correspondence: jixuanwu@sdu.edu.cn (J.W.); chen.jiezhi@sdu.edu.cn (J.C.)
† These authors contributed equally to this work.

Abstract: Flash memory-based computing-in-memory (CIM) architectures have gained popularity due to their remarkable performance in various computation tasks of data processing, including machine learning, neuron networks, and scientific calculations. Especially in the partial differential equation (PDE) solver that has been widely utilized in scientific calculations, high accuracy, processing speed, and low power consumption are the key requirements. This work proposes a novel flash memory-based PDE solver to implement PDE with high accuracy, low power consumption, and fast iterative convergence. Moreover, considering the increasing current noise in nanoscale devices, we investigate the robustness against the noise in the proposed PDE solver. The results show that the noise tolerance limit of the solver can reach more than five times that of the conventional Jacobi CIM solver. Overall, the proposed flash memory-based PDE solver offers a promising solution for scientific calculations that require high accuracy, low power consumption, and good noise immunity, which could help to develop flash-based general computing.

Keywords: computing in memory; flash memory; partial differential equations

Citation: Qi, Y.; Feng, Y.; Wu, J.; Sun, Z.; Bai, M.; Wang, C.; Wang, H.; Zhan, X.; Zhang, J.; Liu, J.; et al. An Efficient and Robust Partial Differential Equation Solver by Flash-Based Computing in Memory. *Micromachines* **2023**, *14*, 901. https://doi.org/10.3390/mi14050901

Academic Editor: Zhongrui Wang

Received: 30 March 2023
Revised: 17 April 2023
Accepted: 20 April 2023
Published: 22 April 2023

1. Introduction

To break the bottleneck of traditional von Neumann architecture, computing-in-memory (CIM) architectures have become a budding technology that can effectively reduce the overhead caused by frequent data transmission between the memory unit and the processing unit [1,2]. So far, most CIM is concentrated in the field of neural networks, such as the edge detection of moving objects by implementing a convolutional neural network (CNN) based on three-dimensional memristors [3], MNIST digit recognition realized on a flash-based deep neural network [4], and an artificial olfactory inference system based on memristive devices [5], while the applications in scientific computing are rarely studied. Scientific computing, such as solving linear equations and partial differential equations (PDE), is a crucial aspect of applied science, social science, and engineering [6–8]. Different from neural networks, to implement scientific calculations like the PDE solver, the accuracy and processing speed should be strictly controlled.

In previous work, a high-precision PDE solver in sliced small memristor crossbar arrays was proposed by adopting the Jacobi iterative method and the precision-extension technique [9]; for low-power calculations, mixed-precision architectures, such as the linear equation solver [10] and the PDE solver [11], were proposed by processing high-precision calculations in ALU units and low-precision calculations in CIM units. To enhance the parallel processing speed of large matrix calculations, flash-based CIM as the PDE solver

was proposed, and the high accuracy required to solve time-dependent PDE was demonstrated [12]. Additionally, a 32-bit floating-point (FP) CIM architecture was realized on a NOR flash CIM as a solution for general-purpose computation tasks [13]. However, all the above studies used difference pairs or two arrays to handle positive and negative values separately in the calculation. While this method could be effective in achieving the desired outcome, it may also result in greater power consumption.

Different from the mathematical methods to solve PDE, the method that can be utilized in CIM-based PDE solvers should take the hardware factors into account, such as the properties of CIM devices, arrays, and peri-circuits. In these considerations, the Jacobi iterative method was commonly adopted in previous studies because it is simple for the design of CIM operations [14], by which the matrix weights are fixed without the necessity for real-time data updating. However, a large number of iterations by the Jacobi iterative method led to large power consumption and a serious time delay in computing tasks. More importantly, with more iterations, the computing errors caused by the instability of devices and arrays accumulate and seriously affect the final result [15–17]. Thereby, for future applications of CIM chips, it is strongly required to optimize the CIM-oriented iteration method and perform the co-optimizations of hardware (the CIM array) and soft strategies.

In this paper, a novel PDE solver for the flash-based CIM array is proposed by adopting the second refinement of the Jacobi (SRJ) iterative method [18], which can reduce the iteration number effectively and optimize the calculation process with improved accuracy. Based on 55 nm flash memory technology, the abilities to solve PDE are demonstrated in different grid situations. In comparison to the standard Jacobi iterative method, faster convergence speed, better anti-noise ability, and lower power consumption are achieved, as well as improved accuracy. Moreover, the pre-processing approach is proposed to save unnecessary array area waste and further suppress the total power consumption.

2. Materials and Methods

2.1. Architecture

It is widely acknowledged that data processing has become increasingly demanding in practical applications. The conventional von Neumann architecture results in a significant amount of energy being expended on data transmission between the memory unit and the calculation unit, rather than on computing itself. This can be particularly problematic for certain applications that contain lots of matrix–vector multiplication, such as solving partial differential equations. In such cases, CIM architecture can offer a solution by reducing unnecessary power consumption. Specifically, all the operation processes in this work are littered with large-scale, high-precision multiplication and cumulation (MAC) that we can implement via a one-shot read operation on a flash memory array that has mature technology with capabilities of high bit-density and robust reliability in ultra-large arrays. Figure 1a,b show the stacking structure diagram and the TEM image of one flash memory cell of a 55 nm technology node. Depending on the current characteristics in the saturation region, as shown in Figure 1c, each cell can be considered a variable conductance that can be modulated by the settled gate bias. The entire schematic of the flash memory array as a multiplier is also shown in Figure 1d. In the matrix to process multiplications, the iteration matrix is mapped as the matrix of threshold voltage (V_{th}) in the flash memory array, while the vectors are designed by the applied pulses, with pulse durations proportional to the values at a fixed amplitude. In this work, the input (for example, $x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}$) is applied to the word-line (WL), then the multiplication result ($x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_n^{(k+1)}$) is achieved at the source side by integrating the current to obtain the amount of charge according to Ohm's law and Kirchhoff's current law. Thereby, we can implement the vector–matrix multiplications with peri-circuits, as presented in Figure 2a.

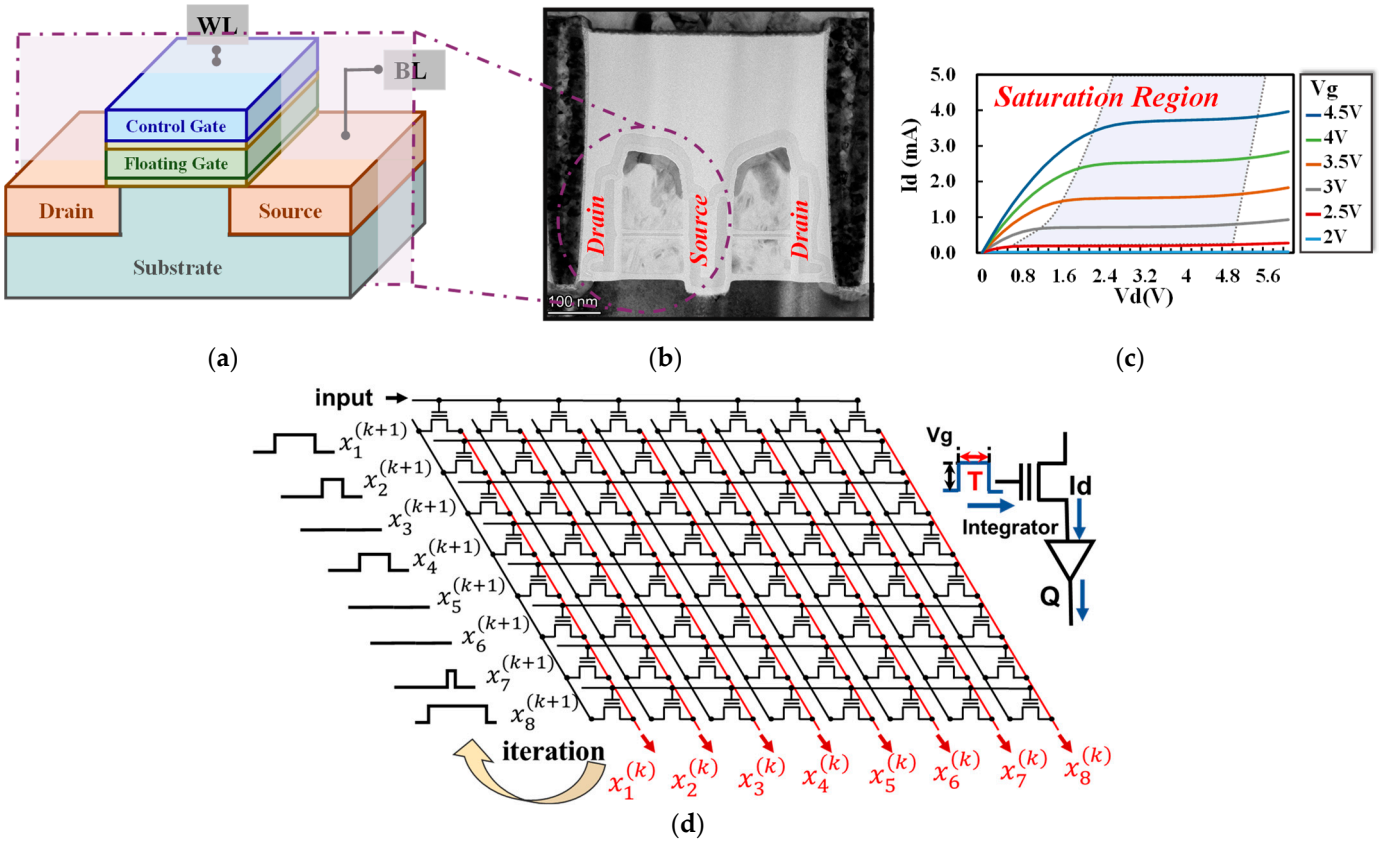


Figure 1. (**a**) Schematic of the structure of a transistor, (**b**) TEM image of the cell's cross-section, and (**c**) the output characteristic curves are also shown for reference; (**d**) the flash-based PDE solver, input as pulses with variable duration and output as the charge we collected.

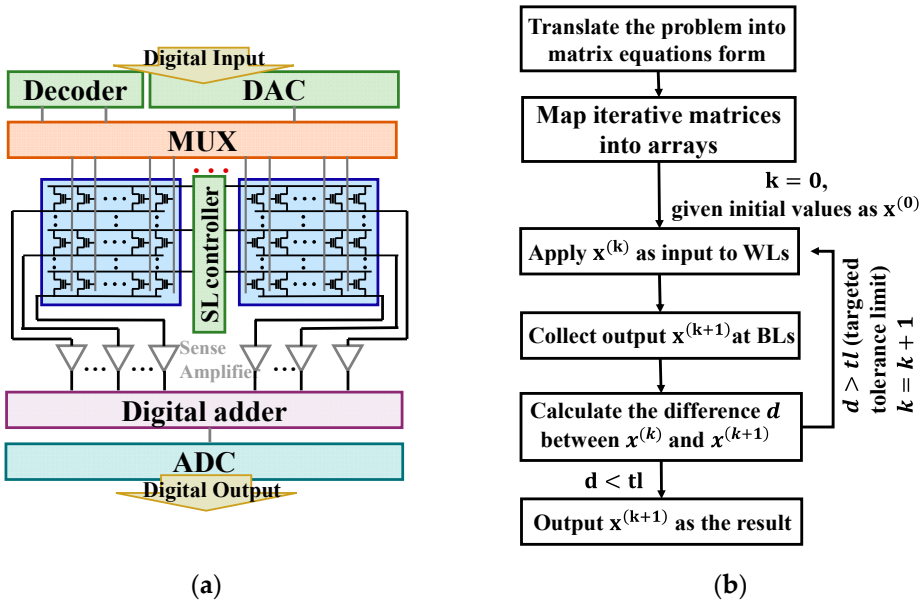


Figure 2. (**a**) Schematic of the system design of the solver; (**b**) the flow chart of the whole solution process.

PDE contains multivariate unknown functions and their partial derivatives of several orders. The Poisson equation is a particular kind of PDE and its main form is shown in Equation (1) [19]. The Poisson Equation (2) exemplified in this work is solved by the finite difference method (FDM), whose main principle is to make a direct difference

approximation to the differential term in the equation, translating the differential equation into the linear equation that is easier to be solved.

$$\nabla^2 u = f \tag{1}$$

$$\begin{cases} \nabla^2 u = -2\pi^2 \sin \pi x \sin \pi y, \\ u(x,0) = u(x,2) = 0, 0 \le x \le 2, \\ u(0,y) = u(2,y) = 0, 0 \le y \le 2. \end{cases} \tag{2}$$

To solve PDEs, the first step of the FDM method is to transform the solution domain into discrete grid blocks to obtain the differential approximation of each discrete point. After changing the two-dimensional second-order partial derivatives of x and y into FDM form, it is easy to derive the five-point difference format of the Laplace operator term as

$$\nabla^2 u = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{h^2} \tag{3}$$

where h is the step size we selected in the x-axis direction, i is the x-axis index of the current grid point, and $i-1$ and $i+1$ are the indexes of adjacent grid points. It is the same of j as the y-axis index. Similarly, we choose the same step size of h in the y-axis direction.

To facilitate understanding and determining the function values $u_{i,j}$ of each grid point, we convert the five-point difference format of all inner nodes into a system of difference equations. Then, the left side of Poisson Equation (1), discretized by 3×3 grid points as an example, can be transformed into grid form, as shown in Figure 3a.

Then, we can obtain its matrix equation form as $Ax = b$,

$$A = \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix},$$

$$b = \begin{bmatrix} h^2 f_{1,1} + g_{1,0} + g_{0,1} \\ h^2 f_{2,1} + g_{2,0} \\ h^2 f_{3,1} + g_{3,0} + g_{4,1} \\ h^2 f_{1,2} + g_{0,2} \\ h^2 f_{2,2} \\ h^2 f_{3,2} + g_{4,2} \\ h^2 f_{1,3} + g_{0,3} + g_{1,4} \\ h^2 f_{2,3} + g_{2,4} \\ h^2 f_{3,3} + g_{3,4} + g_{4,3} \end{bmatrix}, x = \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ u_{1,3} \\ u_{2,3} \\ u_{3,3} \end{bmatrix}. \tag{4}$$

where f is the value of the function on the right side of the equation, g is the boundary value given by the boundary conditions. Since we need to obtain the final exact solution by iteration, $x^{(k+1)}$, obtained from the output value in the previous iteration, is used as the new input in the next iteration. During the process of repeating the output as the next input, the iteration is carried on. When the difference between the result of the previous iteration and the result of the current iteration is below a set value which we call tolerance (tl), the iterative action cuts off. In this case, the output of the last iteration is the final result we obtain. The entire process is described in Figure 2b. In the following studies, we use two different iterative methods for comparisons of their efficiency in obtaining the final exact solution, the standard Jacobi iterative method and the SRJ iterative method.

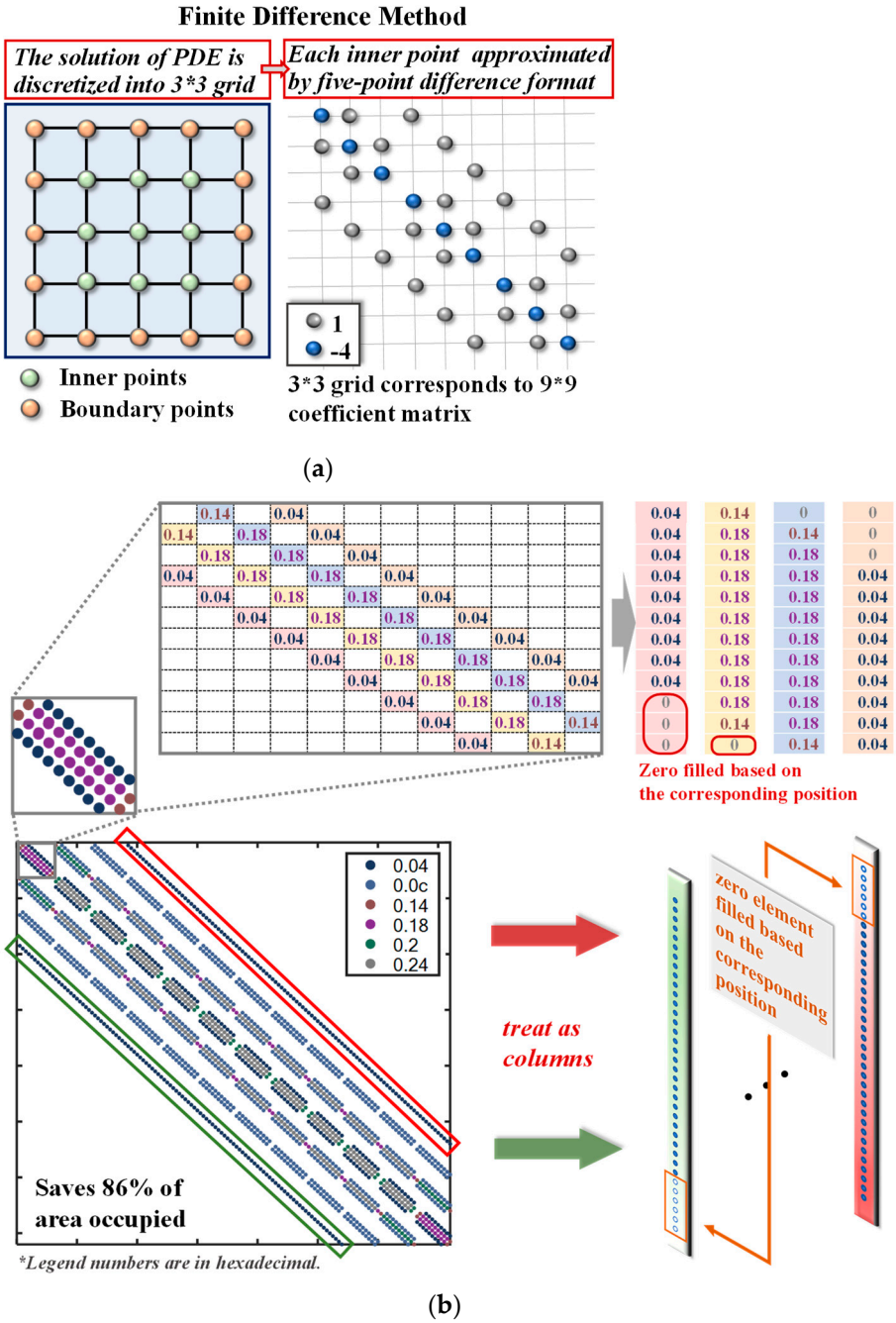


Figure 3. (**a**) The FDM flow (3 × 3 grid) and the point map obtained; (**b**) an example with hexadecimal legend numbers of the SRJ iteration matrix in the case of 12 × 12 grid FDM. Top: the zoomed upper left corner of the matrix is an example to show how the mapping works. Bottom: the first diagonal and the last diagonal chosen as a demonstration are mapped into the array.

2.2. SRJ Iterative Method

The standard Jacobi iterative method is a generally used method with the format (5) to solve linear equations in the form of $A \cdot x = b$,

$$x^{(k+1)} = B_J x^{(k)} + f_J \tag{5}$$

where $B_J = D^{-1}(L + U)$, $f_J = D^{-1}b$, D is a diagonal matrix with a non-zero diagonal; L and U are the upper triangular matrix and lower triangular matrix of A, respectively; and $A = D - L - U$. $x^{(k)}$ is the k^{th} the iterative result, while $x^{(k+1)}$ is the $(k+1)^{th}$.

In this work, considering the convergence speed, we use a new modified algorithm of Jacobi instead: the second-refinement of Jacobi (SRJ) iterative method. The iteration format of SRJ is

$$x^{(k+1)} = \left[D^{-1}(L+U)\right]^3 x^{(k)} + \left[I + D^{-1}(L+U) + (D^{-1}(L+U))^2\right] D^{-1}b \tag{6}$$

and the simplified form can be expressed as

$$x^{(k+1)} = {B_J}^3 x^{(k)} + \left(I + B_J + {B_J}^2\right) f_J \tag{7}$$

where I is an n-dimensional identity matrix.

To accelerate the iterative process, we can use the result $x^{(k+1)}$ twice in Formula (5) to obtain the final optimized iterative Formula (7), then the results of three Jacobi iterations can be achieved in a single SRJ iteration process. It is critical for CIM-based PDE solvers that the increase in the number of iterations inevitably leads to an increase in power consumption. In order to reduce the power consumption, the tradeoff between accuracy and the number of iterations needs to be taken into consideration to ensure the optimal CIM PDE solver solution with low power consumption and latency.

2.3. Pre-Processing

Although the values of the iteration matrix are all positive, there is a tendency that intermediate results during the iteration, that is, the input of the next iteration, could have negative values, which is contrary to the requirement that input values cannot be negative due to the physical significance of flash memory. To fix this problem, we normalize the input value to a positive value with the form of (8) instead of using two different flash arrays to handle negative and positive values, respectively. This can reduce the array area to half. The pre-processing part is implemented in the software and applied to the arrays by DAC operation to make sure that all values in the array are positive to match the physical requirements of the flash memory.

$$x_{new} = \frac{x_{original} - x_{min}}{x_{max} - x_{min}} \tag{8}$$

Then, the original output result can be easily obtained by calculating the output result gained by pre-processing via the inverse normalization formula (9),

$$y_{original} = (x_{max} - x_{min}) \cdot y_{new} - D \cdot x_{min} \tag{9}$$

where y_{new} is the output after pre-processing, $y_{original}$ is the correct result that should have been obtained, and D is the matrix value stored in the array. Additionally, the inverse normalized output, that is, the actual solution $y_{original}$ during this iteration, can be reconstructed after ADC transforms the output of integrators into digital form. Then, if the iteration does not converge, that is the solution does not reach a stable state, the solution would be reapplied as the input of the next iteration to achieve the final result.

2.4. Mapping Method

Since the SRJ iterative matrices and Jacobi iterative method are both characterized by their large and sparse identity, the zero elements predominate in the matrix. Therefore, we can use a partition method as indicated in Figure 3b to map the matrix to a flash memory array rather than mapping all elements with large amounts of zero elements that will cause excessive area waste. To pull all the non-zero diagonals into columns and fill in the zeros above or below, according to the corresponding positional relationship, this method maximizes array utilization while reducing array area and prevents most invalid zero weights in the flash array. Therefore, in large-scale computing with high-sparsity data,

adapting this mapping method can significantly improve system performance and ensure that the memory array operates efficiently and effectively.

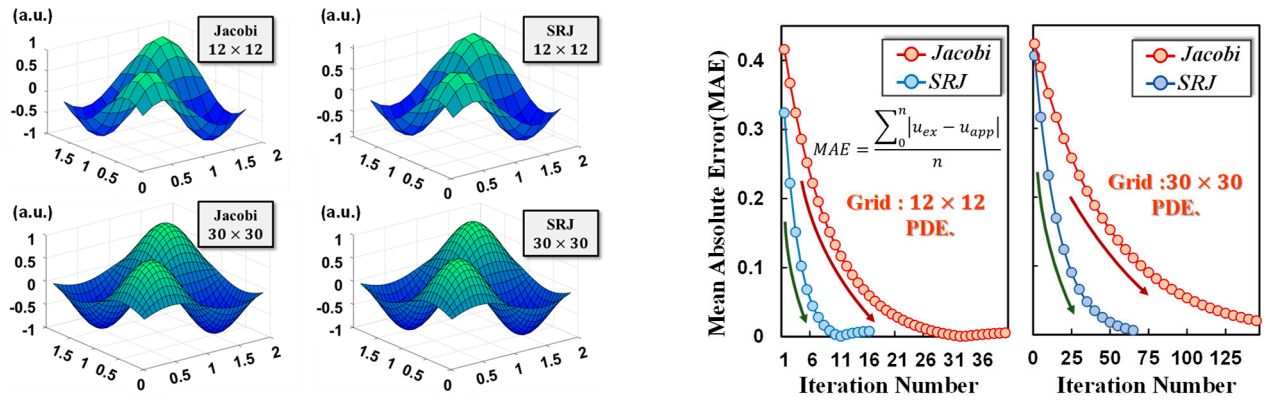
3. Results

Based on the aforementioned optimization methods, we studied the performances of the Jacobi method, which has been widely applied in previous studies, and the new SRJ iterative methods in solving PDEs in 32-bit fixed-point precision. Here, the flash model we used to simulate is the commercial 55 nm NOR flash, which has a native 4-bit precision. To overcome this limitation, the final solution is computed by digital adders according to the precision-extension technique [9]. To verify the influence of the grid number on the accuracy, 12×12 is used as the number of grid points to solve the equation by two different iterative methods. The tolerance (tl) that is described as the iteration break condition in Figure 2b is 1×10^{-3} in the simulation. The results obtained by Jacobi and SRJ are plotted in the top part of Figure 4a. As a comparison, we selected denser grid points (30×30) to obtain more accurate results, and the results are plotted in the bottom part of Figure 4a. It is noticed that the iteration number by SRJ is just one-third of that by Jacobi, which can bring a reduction in power consumption.

In addition to the aforementioned statement, we proceeded to perform a comparative analysis of the two methods, focusing on three key aspects: precision, robustness performance to noise, and power dissipation. The results are presented in the following section.

3.1. Accuracy Analysis

The mean absolute error (MAE) values of each iteration are recorded during the solving process and made into trend graphs, as shown in Figure 4b. The exact solution of this equation is given by the MATLAB internal solver. It is revealed by the trend graph that the MAE value of the SRJ decreases extremely fast at the beginning of the iteration, regardless of whether the case is a 12×12 grid or a 30×30 grid. These phenomena indicate that the SRJ solver converges rapidly in all cases, demonstrating the efficiency of the SRJ method in achieving accurate results with fewer iterations. This means that much more power consumption can be effective by adopting the SRJ method for scientific computing tasks. It is noticed that the precision of the Jacobi solver decreases dramatically when the grid mesh becomes dense, even more so than the results of the SRJ solver in a rough grid situation, contrary to the assumption that a finer mesh should be closer to the exact solution. Nevertheless, the SRJ solver fits well with the idea regardless of the case, showing that the MAE value approaches zero as the mesh becomes finer. Thus, we can conclude that the SRJ solver can perform better with denser grid conditions; in other words, it will be better when the target solution is closer to the true value.



(**a**) (**b**)

Figure 4. *Cont.*

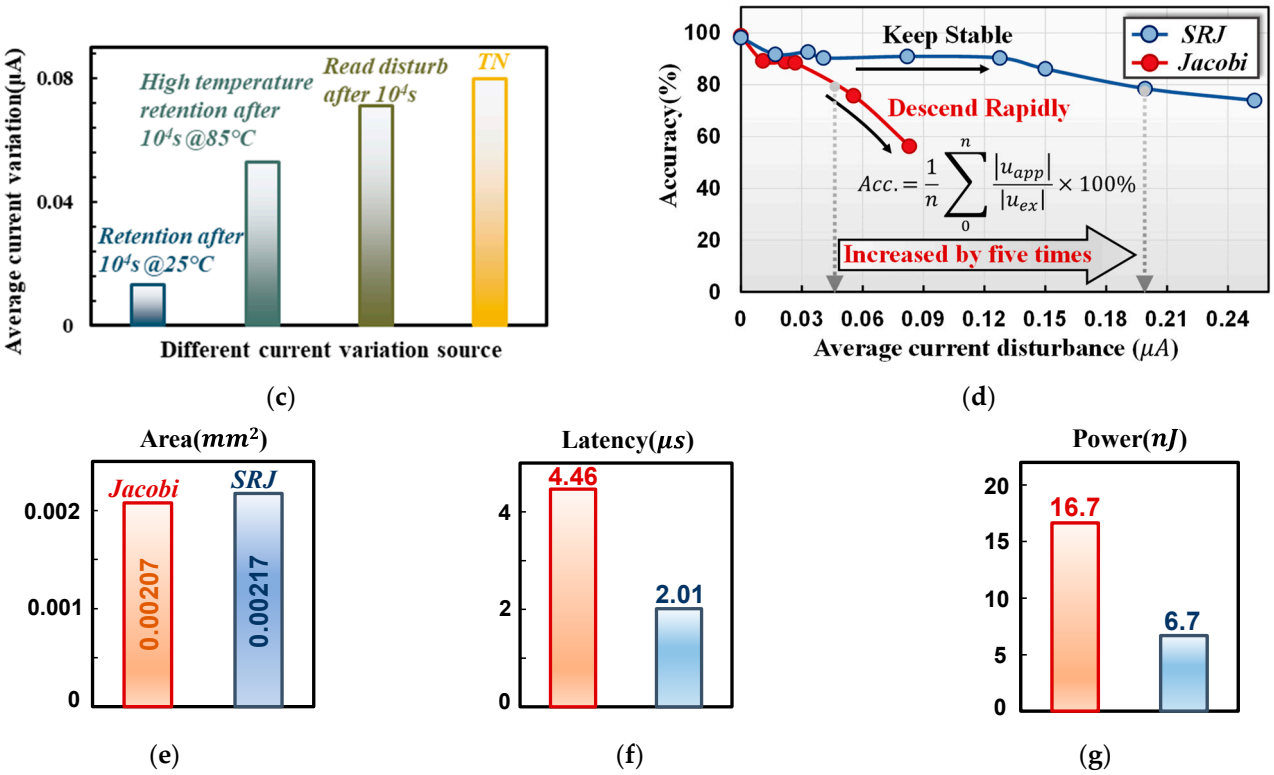


Figure 4. (**a**) Image of the approximate solution of Poisson Equation (2) from the flash memory solver using the Jacobi iterative method (**left top**) and the SRJ iterative method (**right top**) with a 12 × 12 grid; and the Jacobi iterative method (**left bottom**) and the SRJ iterative method (**right bottom**) with a 30 × 30 grid. (**b**) Trend graphs of the MAE values in solving (**a**) PDEs with 12 × 12 grid and (**b**) PDEs with 30 × 30 grid. The orange/blue stands for Jacobi/SRJ, respectively. u_{ex} in the formula is the exact result by MATLAB, and u_{app} is the approximate iterative solution. The iteration termination condition for both methods is the tolerance of 1×10^{-3}. (**c**) The current summary of current variations in different tests. (**d**) The variation of accuracy with the average current disturbance increasing. (**e**) Estimated area, (**f**) latency, and (**g**) power by Jacobi CIM and SRJ CIM when solving Poisson's equation with a 12 × 12 grid.

3.2. Robustness Performance to Noise

The current variation is the key factor determining computing accuracy, which is affected by external conditions and devices. To evaluate the impact of current variations, aspects of reality that affect flash memory reliability, including retention, read disturb, and telegraph noise (TN), are summarized in Figure 4c. The average current drifts after 1×10^4 s of retention both at 25 °C and 85 °C are smaller than read disturbance and TN. However, the current variations remain under control, indicating the capability of flash memory to construct the large array system.

To explore the stability of the architecture, the effects of the current disturbance on the accuracy are tested. Figure 4d shows the comparison of the Jacobi solver and the SRJ solver in a 12 × 12 grid condition. When the average current disturbance reaches 0.04 μA, the accuracy of the Jacobi solver dropped down to 80%, which is fatal for functions that need a high accuracy, such as solving PDEs, while at the same time, the SRJ solver still performs well even at 0.2 μA, which is five times the tolerance limit of the Jacobi solver, and the accuracy is maintained above 80%. It is noticed that current variations remain in a controlled range without accuracy loss, according to Figure 4d. Thus, the SRJ solver can achieve a better robustness compared to the traditional Jacobi solver.

3.3. Power Dissipation

To analyze the power dissipation of the CIM architecture, the iteration number and precision of this PDE solver architecture are evaluated by making a comparison between two iterative methods. The simulation results show that only 16 iterations are needed by the SRJ method when processing PDE with a 12×12 grid, which is much lower than the 40 iterations by the Jacobi method. Similar results are also obtained in the PDE with a 30×30 grid. The iteration number of the SRJ method and the Jacobi method are 67 and 147, respectively. The MAE of the SRJ architecture with a 30×30 grid is 0.005, much smaller than the MAE of the Jacobi architecture (0.019). More nonzero values in the SRJ method make the array power dissipation of the SRJ architecture larger than the Jacobi architecture, which is 30 pJ and 15 pJ, respectively, in one PDE solution with a 12×12 grid.

Moreover, the power dissipation containing peripheral circuits must also be considered [20]. The traditional peripheral circuit of the PDE solver can be directly used in the proposed PDE solver without any modification, and the specific peripheral circuit parameters refer to [21]. The switch matrix controls multiple word lines (WLs) and bit lines (BLs), and the read circuit is used to convert analog current to digital output. An adder and subtractor are also needed to realize the entire computing process. Similar to other CIM applications, write pulses much shorter than traditional write operation are required to tune the memory states with higher accuracy. The results of peripheral circuits in Figure 4e–g are simulated results based on the NeuroSim simulation tool [21], which can be utilized for the estimations of recognition accuracy and power consumption [22,23]. As shown in Figure 4e–g, although the area of the array and peripheral circuit in the SRJ architecture is slightly larger than the Jacobi architecture because the iteration matrix of the SRJ method is somewhat larger, the simulation results shown in Figure 4e confirm that the latency of the Jacobi method is over two times higher than that of the SRJ method when solving a Poisson equation. This provides evidence that the SRJ mapping scheme exhibits exceptional performance, with an operating speed that is more than twice as fast as that of the Jacobi method. Simultaneously, the power consumption can also be well suppressed by the reduction of iteration numbers.

4. Discussion

So far, several PDE solvers has been proposed with the CIM architecture for efficient computing. Compared with the previously proposed method to solve the matrix equation in one step by sacrificing the accuracy [24], our architecture accelerates computation processes with the premise of maintaining calculation accuracy. Especially for the high-precision required by PDE solvers, accuracy and reliability are the most important concerns. While taking the feasibility for large-matrix processing into account, the noise immunity, array size, power dissipation, as well as the convergence speed are all important and co-optimizations are strongly required. Therefore, the proposed flash-based PDE solver could provide a feasible approach because flash memory is a mature technology with capabilities of high bit density in ultra-large arrays and robust reliability that can carry out large-scale high-precision computing tasks. By adopting flash technology, we need to design the processing approach for the triple-terminal cell and optimize the operation schemes to the unique properties and array designs of flash memory, as described in Figures 1 and 2. Moreover, in this work, we propose to solve PDEs by using the SRJ method, which makes a distinction between our work and other related work. Table 1 shows the comparison between our work and other prior CIM solvers. Specifically, the algorithm in [9,13,25] is the Jacobi method, in [8,11] it is the residual method, and in [26] it is the fourth order Runge–Kutta method. Different algorithms lead to different results. As can be seen, the proposed PDE solver can maintain a high accuracy and low residual norm errors together with a low power dissipation.

Table 1. Comparison with the state of the art.

	CIM Device	Power Dissipation	Accuracy	Residual Norm Error	Latency
This Work	NOR Flash (55 nm)	**6.7 nJ**	**98.78**%	$\mathbf{2 \times 10^{-8}}$	**2.01 µs**
Gallo [10]	PCM (90 nm)	**<1 − 100 fJ** per device	—	$\mathbf{\sim 1 \times 10^{-6}}$	**<100 ns**
Zidan [9]	RRAM	—	**>97.3**%	—	**1 µs**
Chen [25]	SRAM (180 nm)	—	—	—	**90 ns**
Feng [13]	NOR Flash (65 nm)	—	**100**%	—	—
Ensan [26]	RRAM (65 nm)	—	**97**%	—	**25 ns**
Yang [11]	RRAM (22-nm) &CPU/GPU	**30 mJ**	—	$\mathbf{8 \times 10^{-15}}$	—

5. Conclusions

To sum up, we propose a flash-based CIM architecture as the PDE solver by adopting the SRJ iterative method. Due to the fast operation speed and large storage capacity of flash memory, high-speed data processing and high-precision computing results can be achieved in a large-scale flash memory array. Moreover, an efficient iterative algorithm, SRJ, is proposed to minimize iteration times and accelerate the solving process. Compared with the standard PDE solver using the Jacobi iterative method, this architecture can largely suppress power consumption with much improved calculation accuracy and current noise tolerance. The influence of the array size (grid points) on the computing accuracy is also investigated, showing the ability of the SRJ architecture to achieve high-accuracy solutions for fine grids. Furthermore, the analysis of peripheral circuits in the system indicates that co-optimizations of CIM hardware (the memory cells and the array) with the soft architectures are the key to developing general-purpose CIM applications. This work provides a feasible flash-based CIM that has great potential to overcome the limitations of traditional von Neumann architectures and enable a high-precision and high-performance PDE solver. This could help pave the way for the further development of CIM in general computing.

Author Contributions: The work presented here was completed in collaboration between all authors. Conceptualization, Y.Q. and Y.F.; methodology, Y.Q.; software, Y.Q. and Y.F.; validation, Y.Q., Y.F. and H.W.; formal analysis, Y.Q.; investigation, Y.F., J.W. and C.W.; resources, Y.F., Z.S. and M.B.; writing—original draft, Y.Q. and Y.F.; writing—review and editing, J.C.; supervision, J.C.; project administration, J.C.; funding acquisition, J.W., X.Z., J.Z., J.L. and J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (Nos. 62034006, 92264201, 91964105), the Natural Science Foundation of Shandong Province (Nos. ZR2020JQ28, ZR2020KF016), and the Program of Qilu Young Scholars of Shandong University.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nair, R. Evolution of Memory Architecture. *Proc. IEEE* **2015**, *103*, 1331–1345. [CrossRef]
2. Yu, S.; Sun, X.; Peng, X.; Huang, S. Compute-in-memory with emerging nonvolatile-memories: Challenges and prospects. In Proceedings of the Custom Integrated Circuits Conference (CICC), Boston, MA, USA, 22 March 2020.
3. Lin, P.; Li, C.; Wang, Z.; Li, Y.; Jiang, H.; Song, W.; Rao, M.; Zhuo, Y.; Upadhyay, N.K.; Barnell, M.; et al. Three-dimensional memristor circuits as complex neural networks. *Nat. Electron.* **2020**, *3*, 225–232. [CrossRef]
4. Xiang, Y.C.; Huang, P.; Zhou, Z.; Han, R.Z.; Jiang, Y.N.; Shu, Q.M.; Su, Z.Q.; Liu, Y.B.; Liu, X.Y.; Kang, J.F. Analog Deep Neural Network Based on NOR Flash Computing Array for High Speed/Energy Efficiency Computation. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019.

5. Wang, T.; Huang, H.; Wang, X.; Guo, X. An artificial olfactory inference system based on memristive devices. *InfoMat* **2021**, *3*, 804–813. [CrossRef]
6. Wang, H.; Yamamoto, N. Using A Partial Differential Equation with Google Mobility Data to Predict COVID-19 in Arizona. *Math. Biosci. Eng.* **2020**, *17*, 4891–4904. [CrossRef] [PubMed]
7. Dumitrescu, F.; McCaskey, J.; Hagen, G.; Jansen, R.; Morris, D.; Papenbrock, T.; Pooser, C.; Dean, J.; Lougovski, P. Cloud quantum computing of an atomic nucleus. *Phys. Rev. Lett.* **2018**, *120*, 210501. [CrossRef] [PubMed]
8. Al Asaad, B.; Erascu, M. A Tool for Fake News Detection. In Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 20 September 2018.
9. Zidan, M.A.; Jeong, Y.; Lee, J.; Chen, B.; Huang, S.; Kushner, M.J.; Lu, W.D. A general memristor-based partial differential equation solver. *Nat. Electron.* **2018**, *1*, 411–420. [CrossRef]
10. Gallo, M.L.; Sebastian, A.; Mathis, R.; Manica, M.; Giefers, H.; Tuma, T.; Bekas, C.; Curioni, A.; Eleftheriou, E. Mixed-Precision In-Memory Computing. *Nat. Electron.* **2018**, *1*, 246–253. [CrossRef]
11. Yang, H.; Huang, P.; Zhou, Z.; Zhang, Y.; Han, R.; Liu, X.; Kang, J. Mixed-Precision Partial Differential Equation Solver Design Based on Nonvolatile Memory. *IEEE Trans. Electron Devices* **2022**, *69*, 3708–3715. [CrossRef]
12. Feng, Y.; Zhan, X.; Chen, J. Flash Memory based Computing-In-Memory to Solve Time-dependent Partial Differential Equations. In Proceedings of the IEEE Silicon Nanoelectronics Workshop (SNW), Honolulu, HI, USA, 13–14 June 2020.
13. Feng, Y.; Chen, B.; Liu, J.; Sun, Z.; Hu, H.; Zhang, J.; Zhan, X.; Chen, J. Design-Technology Co-Optimizations (DTCO) for General-Purpose Computing In-Memory Based on 55nm NOR Flash Technology. In Proceedings of the IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, USA, 11–16 December 2021.
14. Ames, W.F. *Numerical Methods for Partial Differential Equations*, 3rd ed.; Academic Press: San Diego, CA, USA, 2014.
15. Fantini, P.; Ghetti, A.; Marinoni, A.; Ghidini, G.; Visconti, A.; Marmiroli, A. Giant Random Telegraph Signals in Nanoscale Floating-Gate Devices. *IEEE Trans. Electron Devices* **2007**, *28*, 1114–1116. [CrossRef]
16. Wang, R.; Guo, S.; Ren, P.; Luo, M.; Zou, J.; Huang, R. Too noisy at the nanoscale?—The rise of random telegraph noise (RTN) in devices and circuits. In Proceedings of the IEEE International Nanoelectronics Conference (INEC), Chengdu, China, 9–11 May 2016.
17. Spinelli, A.S.; Malavena, G.; Lacaita, A.L.; Monzio, C. Compagnoni. Random Telegraph Noise in 3D NAND Flash Memories. *Micromachines* **2021**, *12*, 703. [CrossRef] [PubMed]
18. Eneyew, T.K.; Awgichew, G.; Haile, E.; Abie, G.D. Second Refinement of Jacobi Iterative Method for Solving Linear System of Equations. *IJCSAM* **2019**, *5*, 41–47. [CrossRef]
19. Gilbarg, D.; Trudinger, N. *Elliptic Partial Differential Equations of Second Order*; Springer: Berlin/Heidelberg, Germany, 1977; Volume 224.
20. Chen, P.Y.; Peng, X.; Yu, S. NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 3067–3080. [CrossRef]
21. Chen, P.Y.; Peng, X.; Yu, S. NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures. In Proceedings of the International Electron Devices Meeting (IEDM), San Francisco, CA, USA, 2–6 December 2017.
22. Choi, Y.; Oh, S.; Qian, C. Vertical organic synapse expandable to 3D crossbar array. *Nat. Commun.* **2020**, *11*, 1–9. [CrossRef] [PubMed]
23. Kazemi, A.; Rajaei, R.; Ni, K.; Datta, S.; Niemier, M.; Hu, X.S. A Hybrid FeMFET-CMOS Analog Synapse Circuit for Neural Network Training and Inference. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020.
24. Sun, Z.; Pedretti, G.; Ambrosi, E. Solving matrix equations in one step with cross-point resistive arrays. *Proc. Natl. Acad. Sci. USA* **2019**, *116*, 4123–4128. [CrossRef] [PubMed]
25. Chen, T.; Botimer, J.; Chou, T.; Zhang, Z. A 1.87-mm 2 56.9-GOPS Accelerator for Solving Partial Differential Equations. *IEEE J. Solid-State Circuits* **2020**, *55*, 1709–1718. [CrossRef]
26. Ensan, S.S.; Ghosh, S. ReLOPE: Resistive RAM-Based Linear First-Order Partial Differential Equation Solver. *IEEE Trans. VLSI Syst.* **2021**, *29*, 237–241. [CrossRef]