A 1.87-mm^2 56.9-GOPS Accelerator for Solving Partial Differential Equations

Thomas Chen, *Student Member, IEEE*, Jacob Botimer, *Student Member, IEEE*, Teyuh Chou, *Student Member, IEEE*, and Zhengya Zhang, *Senior Member, IEEE*

***Abstract*— Solving partial differential equations (PDEs) require high-precision numerical iterations that are demanding in both computation and memory. We apply the multigrid method with a hybrid layer update to reduce iterations and improve speed, and to transform both fine and coarse grids to a residual form to reduce the precision requirement. The reduced precision enables the mapping of a high-precision PDE solver on SRAMs that perform low-precision parallel multiply-accumulates (MACs) in memory, reducing both energy and area. We employ a delay-locked loop to generate well-controlled unit pulses for driving word lines and a dual-ramp single-slope analog-to-digital converter (ADC) to convert bitline outputs. The design is prototyped in a 1.87-mm^2 180-nm test chip made of four 320 × 64 MAC SRAMs, each supporting 128× parallel 5 b × 5 b MACs with 32 5-b output ADCs and consuming 16.6 mW at 200 MHz. The test chip is demonstrated to reach an error tolerance of 10^{-8} in solving PDEs at a grid update rate of 1.38-G entries/s.**

***Index Terms*— Accelerator, numerical solver, partial differential equation (PDE), precision optimization, process in memory (PIM).**

I. Introduction

MANY physical phenomena, such as heat and fluid dynamics, are described by partial differential equations (PDEs). Most PDEs are solved numerically, by first quantizing the solution space in a grid and then applying iterative methods to refine the solution to a desired error tolerance [1].

High-precision PDE solutions require fine grids and high numerical precision, leading to a significant amount of data that need to be processed, moved, and stored. Moreover, a PDE solver commonly requires tens of thousands of iterations to converge. For example, solving a 2-D Poisson equation using a 128 × 128 grid on a graphics processing unit in a floating point is estimated to take 15 mJ/iteration. To shrink the error tolerance from 10^{-4} to 10^{-7} costs at least 320 J!

High-performance digital PDE solvers have been proposed [2], [3], but they still require high-bandwidth DRAM access to sustain the massive number of parallel, high-precision compute. Analog computers [4], [5] were proposed to accelerate PDE solvers by approximately compute to reduce the IO requirement and mixed-signal approach to speedup core computations. However, analog computers require a large area, limiting the parallelism and efficiency in supporting large PDE problems.

To enable a more efficient and practical PDE accelerator design, we apply two algorithm approaches: 1) we adopt a multigrid method that divides the PDE solver to a fine-grid compute and a coarse-grid compute and iterates between the two to accelerate convergence by 5–10× on average and 2) we transform both fine-grid compute and coarse-grid compute to a residual form, lowering the precision to 5 b for a low-error tolerance below 10^{-8}.

Even with faster convergence and a much reduced precision, the implementation cost can still be high using a conventional digital approach. Recently, process in memory (PIM) has been proposed as a new technique that computes directly on a large array of data in place, within memory [6], [7]. SRAM-based PIM relies on level- and/or width-modulating word lines (WLs) of the SRAM array to encode multipliers and activate multiple WLs in parallel [6]–[9]. The SRAM cells' currents in discharging the bitlines (BLs) represent the products, and the current on each BL represents the sum of products. Alternatively, BLs can be modulated to encode multipliers, and BLs are joined to produce the sum of products [10]. By partly eliminating data movement cost and providing a high degree of parallelism, PIM holds the potential of achieving both high performance and efficiency in tasks that involve parallel multiply–accumulate (MAC) operations such as classification and neural networks. Prior work has demonstrated PIM in SRAM that achieved 633.4 pJ/classification [6] and 1.2 nJ/classification [8], translating to 1.17 pJ/op where an op is defined as a MAC operation.

Current SRAM-based PIM designs are limited by SRAM's binary storage and the overhead of multi-bit analog-to-digital (A/D) conversion. Some designs support only binary multiplicands [6], [9], [10], and some choose binary outputs [6], [9]. To reduce the number of A/D converters (ADCs), some designs are tailored to computations in a cone structure that requires only one or a small number of ADCs at the final output [7], [8]. These approaches are not applicable to a PDE solver that requires iterative multi-bit operations and solutions to be updated in every iteration.

In this article, we combine the multigrid, residual algorithm approach with the optimal mapping on 5-b MAC SRAMs to produce a high-performance and efficient PDE

Manuscript received July 28, 2019; revised October 20, 2019 and December 18, 2019; accepted December 18, 2019. Date of publication January 15, 2020; date of current version May 27, 2020. This article was approved by Associate Editor Vivek De. This work was supported in part by DARPA under Grant HR0011-17-2-0018 and in part by NSF under Grant GRFP DGE 1256260. *(Corresponding author: Thomas Chen.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: tcchen@umich.edu; botimerj@umich.edu; teyuh@umich.edu; zhengya@umich.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2019.2963591

solver accelerator [11]. A MAC SRAM supports 5 b $\times$ 5 b MACs with full-bandwidth 5-b outputs to support a PDE solver. We design a delay-locked loop (DLL)-based 5-b driver that produces WL pulses down to 1/8 of a clock period with PVT tolerance. The WL pulses are level-modulated to match 5-b multiplicands stored in SRAM. We employ a dual-ramp single-slope (DRSS) ADC [12] that employs a coarse ramp followed by a fine ramp to increase conversion speed and minimize area. A 1.87-mm^2 180-nm chip consisting of four 320 $\times$ 64 MAC SRAMs is demonstrated at 200 MHz, each providing 1.38-G MAC/s and 32 5-b ADCs at a power consumption of 16.6 mW.

The target application of this article is an embedded PDE solver to accelerate desktop-class scientific applications that require high-performance, high energy efficiency, and high compute density (low cost per function). Such applications often require the use of compute clusters. We aim to provide a more power-efficient, high-performance accelerator solution to this class of problems. The product volume for such applications is not nearly as high as consumer electronics to justify the use of state-of-the-art technology nodes. 180-nm technology can be a suitable starting point.

Compared to the previous publication [11], we added algorithmic reformulation, mapping of a PDE solver system on the low-precision PIM circuits, full characterization of the circuit components, and comparison with state-of-the-art PDE solver chips. Compared to the previous PDE solvers in silicon [4], [5], we demonstrate that our approach enabled improvements of two orders of magnitude in performance and energy efficiency and four orders of magnitude in compute density. These improvements cannot be achieved by circuit design alone. The key is in the co-optimization of the system design with the circuits.

The rest of this article is organized as follows. Section II introduces the algorithm background, Section III explains the algorithm reformulation, and Section IV explains the mapping and dataflow. The prototype architecture is presented in Section V, followed by the implementation of MAC operations in Section VI and memory readout in Section VII. Section VIII presents the results, and Section IX concludes this article.

II. Numerical PDE Solver by Finite Difference Method (FDM)

We use the solution to 2-D Poisson's equation, shown in (1), to explain the PDE solver design. Poisson's equation is widely used in practical applications [13]

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = b \tag{1}$$

where $b(x, y)$ is given, boundary conditions are specified on the perimeter of the domain, and the solution $u(x, y)$ is sought.

Most PDE problems do not have analytical solutions. Instead, numerical approaches using grid discretization is popular. For the 2-D Poisson's equation above, the FDM can be applied to convert u and b into an $M \times N$ grid of step size Δx and Δy along x and y, as shown in Fig. 1(a) [14]. The discretization results in a system of MN equations

$$\frac{u_{i-1,j} + u_{i+1,j} - 2u_{i,j}}{\Delta x^2} + \frac{u_{i,j-1} + u_{i,j+1} - 2u_{i,j}}{\Delta y^2} = b_{i,j}$$
$$2 \le i \le M-1, 2 \le j \le N-1 \tag{2}$$

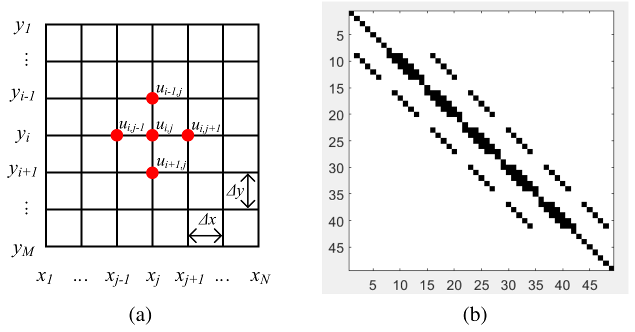


Fig. 1. (a) Illustration of a 2-D finite difference grid. (b) Rendition of matrix **A** for a 7 $\times$ 7 grid.

where $u_{i-1,j}$ is the value of u at grid position $(i-1, j)$, $u_{i+1,j}$ is the value of u at $(i+1, j)$, and so on. $u_{x,y}$ is known on the boundaries of the grid and unknown in the interior of the grid. The u values can be put in an $MN \times 1$ vector $\mathbf{u}$, and similarly, the b values are put in an $MN \times 1$ vector $\mathbf{b}$. The system of equations (2) can be written as

$$\mathbf{Au} = \mathbf{b} \tag{3}$$

where $\mathbf{A}$ is an $MN \times MN$ matrix that stores the weights of u's in (2). For a sufficiently large grid, $\mathbf{A}$ is highly sparse. A rendition of matrix $\mathbf{A}$ for a 7 $\times$ 7 grid is shown in Fig. 1(b), where black dots indicate nonzero entries.

A. Jacobi Iterations

Starting from the boundary conditions and initial guess of interior points of u, the system of equations (2) can be solved iteratively by the Jacobi method

$$\begin{aligned} u_{i,j}^{(n+1)} &= \frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)}\left(u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)}\right) \\ &\quad + \frac{\Delta y^2}{2(\Delta x^2 + \Delta y^2)}\left(u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)}\right) \\ &\quad - \frac{\Delta x^2 \Delta y^2}{2(\Delta x^2 + \Delta y^2)} b_{i,j} \\ &= s_1\left(u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)}\right) + s_2\left(u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)}\right) + c_{i,j} \\ &\quad 2 \le i \le M-1, 2 \le j \le N-1 \end{aligned} \tag{4}$$

where the superscript (n) in $u_{i,j}^{(n)}$ indicates the value of $u_{i,j}$ in iteration n. Since Δx and Δy are known and $b_{i,j}$ does not change each iteration, s_1, s_2, and $c_{i,j}$ can be pre-computed. Each Jacobi iteration updates all $(M-2)(N-2)$ interior u values based on their values in the previous iteration. Using FDM, each $u_{i,j}$ update requires its neighboring four points, $\{u_{i-1,j}, u_{i+1,j}, u_{i,j-1}, u_{i,j+1}\}$, called a four-point stencil, as illustrated in Fig. 1(a). In the matrix form, the Jacobi iterations can be described as

$$\mathbf{u}^{(n+1)} = \mathbf{T}\mathbf{u}^{(n)} + \mathbf{c} \tag{5}$$

where $\mathbf{T}$ is an $MN \times MN$ matrix that store the stencil weights. $\mathbf{T}$ essentially contains the off-diagonal entries of $\mathbf{A}$, and it is also highly sparse for a sufficiently large grid. Higher order PDEs use higher order grids, but the same formulation applies.

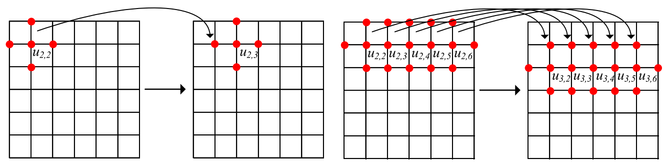

$u_{2,2}$
$u_{2,3}$
$u_{2,2}$ $u_{2,3}$ $u_{2,4}$ $u_{2,5}$ $u_{2,6}$
$u_{3,2}$ $u_{3,3}$ $u_{3,4}$ $u_{3,5}$ $u_{3,6}$
(a)
(b)

Fig. 2. PDE iterative methods with sequential updates. (a) Gauss–Seidel update. (b) Hybrid layer update.

B. Hybrid Layer Update Method

Using the Jacobi method, the next iteration u values are computed based entirely on the u values from the current iteration. Thus, the next iteration cannot start until the current iteration is complete and saved in a buffer memory. To reach a faster convergence and possibly remove the buffer, the Gauss–Seidel method computes the next iteration u values based on the latest available u values.

As shown in Fig. 2(a), the Gauss–Seidel method updates $u_{i,j}$ one by one, for example, from left to right and then top to bottom. The updated values are immediately applied in computing the neighboring $u_{i,j}$ value. By always using the latest available values, the Gauss–Seidel method converges faster than the Jacobi method. The Gauss–Seidel method also removes the buffer, but it introduces data dependence and limits parallelism.

To speedup convergence without sacrificing all the parallelism, we employ a hybrid layer update method [15], [16]. As shown in Fig. 2(b), the grid is divided into layers. A layer of u values can be updated in parallel without data dependence. Updated u values from one layer are used in computing the updates for the next layer. For a large grid, plenty of parallelism is available in one layer to enable parallel processing. The layer-to-layer sequential update also provides a faster convergence. The hybrid layer update method uses a layer buffer, smaller than a block buffer needed for the complete Jacobi method.

III. Algorithm Reformulation for Lower Precision

The baseline PDE solver applies a single grid. To reach an accurate solution, a fine grid of fine step sizes is required. To further speed up convergence, we adopt a multigrid method with our proposed complete residual reformulation to enable aggressive precision reduction to simplify the computation.

A. Complete Residual Approach

To further speed up convergence, the multigrid method introduces an $m \times n$ ($m < M$, $n < N$) coarse grid in addition to the fine grid [14]. Coarse-grid vertices represent a local region of grid values. By interleaving coarse-grid with fine-grid iterations, convergence is accelerated, thanks to faster propagation. A coarse grid reduces the computation workload, for example, using a 2×2 downsampled coarse grid reduces the workload by 75%.

As illustrated in Fig. 3(a), the residual $\mathbf{r}$ is obtained after a round of fine-grid iterations and restricted to $\mathbf{r}^*$. After transitioning to the coarse grid, we can solve for $\mathbf{e}^*$ in $\mathbf{A}^*\mathbf{e}^* = \mathbf{r}^*$. Note that the $mn \times mn$ matrix $\mathbf{A}^*$ is the downsampled version of $\mathbf{A}$. After a round of coarse-grid iterations, $\mathbf{e}^*$ is interpolated to $\mathbf{e}$, and then it is used to update $\mathbf{u}$ to start the next round of fine-grid iterations. The restriction and interpolation are commonly done by pooling and averaging.

Because coarse-grid compute operates on errors $\mathbf{e}^*$ and residuals $\mathbf{r}^*$, that is, the small differences between consecutive iterations, the precision of the errors, and the residuals can be relaxed. Realizing the potential benefit of the residual approach, we present the complete residual approach by extending the residual approach to fine-grid compute, as shown in Fig. 3(b), so that the fine-grid iterations also work on errors $\mathbf{e}$ and residuals $\mathbf{r}$.

Note that in Fig. 3(b), after a round of fine-grid or coarse-grid iterations, $\mathbf{u}$ is updated, and then the updated $\mathbf{u}$ is used to compute the residuals. Since $\mathbf{u}$ is in full precision, computing the residuals requires costly multiplications. To avoid full-precision multiplications, we apply an equivalent form of residual computation based on the low-precision $\mathbf{e}$ and $\mathbf{e}^*$. In this way, all full-precision multiplications are eliminated, as shown in Fig. 3(c).

B. Evaluation of Algorithm Improvement

In Fig. 4, we compare the average convergence speed of solving 2-D Poisson's equation using the Jacobi method and error tolerance of 10^{-7}. A 32-b-quantized 127×127 grid requires 74k iterations to reach convergence. If the single grid is replaced by multigrids, that is, a 127×127 grid and a 64×64 grid, the convergence is shortened by more than $12\times$ to 6k iterations.

The complete residual approach allows the precision to be aggressively reduced to allow a shorter bit width but at the cost of slower convergence. For example, when the complete residual computation is quantized to 8 b, the latency increases by 33%. Further quantizing to 5 and 4 b results in $2.1\times$ and $2.3\times$ latency increase, respectively. The optimal bit width is between 4 and 8 b. Reducing the bit width below 4 b slows down the convergence significantly and ceases to be practical. We choose 5 b for this article to provide margin to avoid the steep degradation in accuracy (or increase in latency to meet a given accuracy).

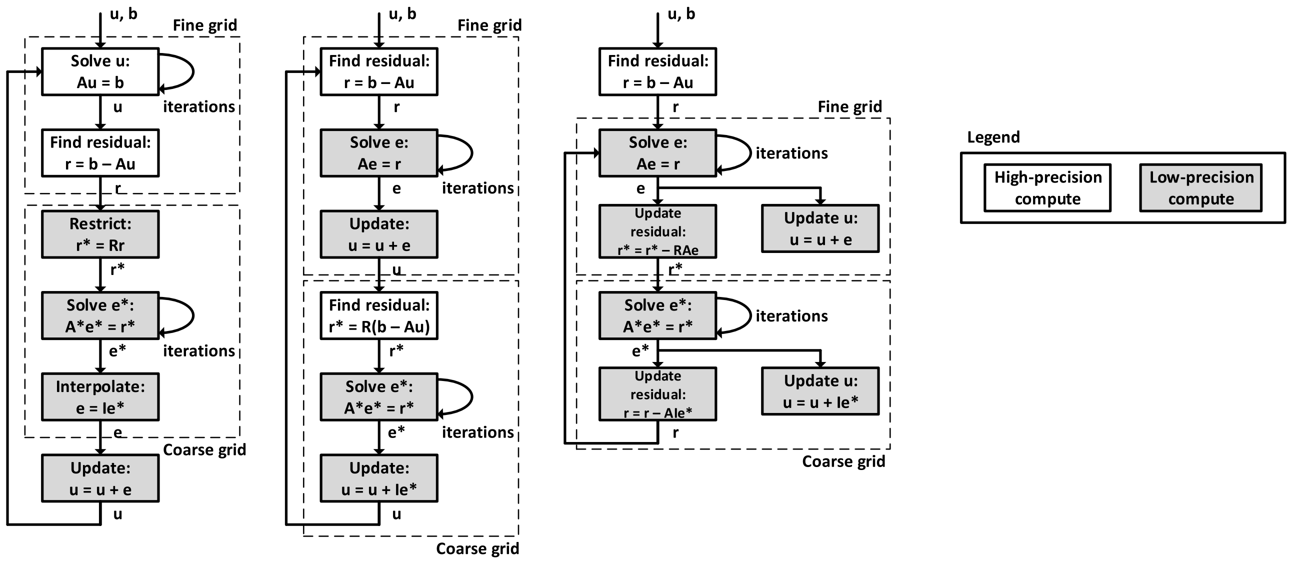
The results in Fig. 4 were obtained using the Jacobi method. The latency can be reduced further by 31% using the hybrid layer update method.

IV. Mapping of PDE Solver and Dataflow

Following the complete residual approach shown in Fig. 3(c), the core computation is for solving $\mathbf{A}\mathbf{e} = \mathbf{r}$ in the fine grid and $\mathbf{A}^*\mathbf{e}^* = \mathbf{r}^*$ in the coarse grid. Both $\mathbf{A}\mathbf{e} = \mathbf{r}$ and $\mathbf{A}^*\mathbf{e}^* = \mathbf{r}^*$ are solved by iterations

$$\begin{aligned} \mathbf{e}^{(n+1)} &= \mathbf{T}\mathbf{e}^{(n)} + \mathbf{c} \\ \mathbf{e}^{*(n+1)} &= \mathbf{T}^*\mathbf{e}^{*(n)} + \mathbf{c}^* \end{aligned} \tag{6}$$

where $\mathbf{e}^{(n+1)}$ and $\mathbf{e}^{(n)}$ are $MN \times 1$ vectors, $\mathbf{T}$ is an $MN \times MN$ matrix, $\mathbf{e}^{*(n+1)}$ and $\mathbf{e}^{*(n)}$ are $mn \times 1$ vectors, and $\mathbf{T}^*$ is an $mn \times mn$ matrix. The matrix-vector products, $\mathbf{T}\mathbf{e}^{(n)}$ and $\mathbf{T}^*\mathbf{e}^{*(n)}$ take the most computation resources.



(a) (b) (c)

Fig. 3. Residual approaches. (a) Standard residual approach applied to fine grid only. (b) Complete residual approach applied to both fine and coarse grids. (c) Reformulated complete residual approach without high-precision multiplication.

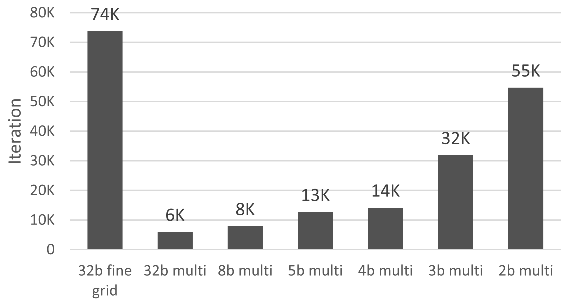


Fig. 4. Latency of solving 2-D Poisson's equation to reach an error tolerance of 10^{-7} using a 127 × 127 single grid and 32-, 8-, 5-, 4-, 3-, and 2-b-quantized multigrids (a 127 × 127 fine grid combined with a 64 × 64 coarse grid).

The complete residual approach allows us to quantize the computation to 5 b, making it possible to perform the computation using MAC SRAM. Since the stencil matrices **T** and **T*** are highly sparse, it is wasteful to be stored in SRAM. Instead, we store the errors **e** and **e*** in SRAM with each 5-b value stored in five cells in consecutive rows, and 5-b stencil weights are applied as WL pulses to the SRAM. The MAC outputs are the updated errors, which are converted to 5-b digital values.

In the direct mapping, a $5M \times N$ SRAM array can be used to store up to an $M \times N$ grid, and the **e** or **e*** values are stored in the SRAM based on their grid locations. The mapping is illustrated in Fig. 5(a). Note that a row in Fig. 5(a) represents five consecutive rows in memory as each 5-b operand spans five rows. To avoid confusion, we will use "group" to refer to a group of five rows that store 5-b operands.

As an example, to update $e_{1,1}$, $\{e_{1,0}, e_{1,2}, e_{0,1}, e_{2,1}\}$ need to be read and multiplied by their respective stencil weights, and then the partial sums are added. However, the four operands $\{e_{1,0}, e_{1,2}, e_{0,1}, e_{2,1}\}$ are not located on the same WL or BL; thus, it is impossible to add the partial sums in PIM. The direct mapping is also incompatible with the hybrid layer update method due to BL access conflicts that prevent **e** values to be computed in parallel.

A. MAC SRAM Organization and Mapping

To use PIM, the operands need to be aligned in memory. We create a rotation mapping to transform Fig. 5(a) and (b): group 0 stays in place, group 1 is right rotated by 1, group i is right rotated by i, and so on. The rotation allows the operands to be aligned. For example, activating groups 0 and 1 with the respective stencil weights enables the parallel summing of pairs of partial sums for updating $e_{1,i}$, $2 \leq i \leq M - 1$.

With rotation mapping, the operands from the odd-numbered columns are read and the results are written to the even-numbered columns, and vice versa. Therefore, we split the odd and even columns and store them in separate even and odd SRAM arrays, as shown in Fig. 5(c). Each array provides two ports: one port for read to perform MAC operations and another port for write back. The odd and even arrays run in parallel. The read output of the odd array is written back to the even array, and vice versa.

After the rotation mapping, one four-point stencil is separated into two halves. For example, the stencil $\{e_{1,0}, e_{1,2}, e_{0,1}, e_{2,1}\}$ stored in the odd SRAM array needs to be separated to two banks: one that stores $\{e_{1,0}, e_{0,1}\}$ and the other that stores $\{e_{1,2}, e_{2,1}\}$, as shown in Fig. 5(d). The partial sums of the two halves are summed on two BLs after activating groups 0 and 1 for the left bank and groups 1 and 2 for the right bank.

To sum up, an $M \times N$ grid is stored in two SRAM arrays, each consisting of two banks of size $5M \times N/4$. The complete update of a layer requires two steps across the two SRAM arrays. For example, the update of layer 0 is done in two steps across the two SRAM arrays: 1) groups 0 and 1 are activated on the left bank, whereas groups 1 and 2 are activated on the

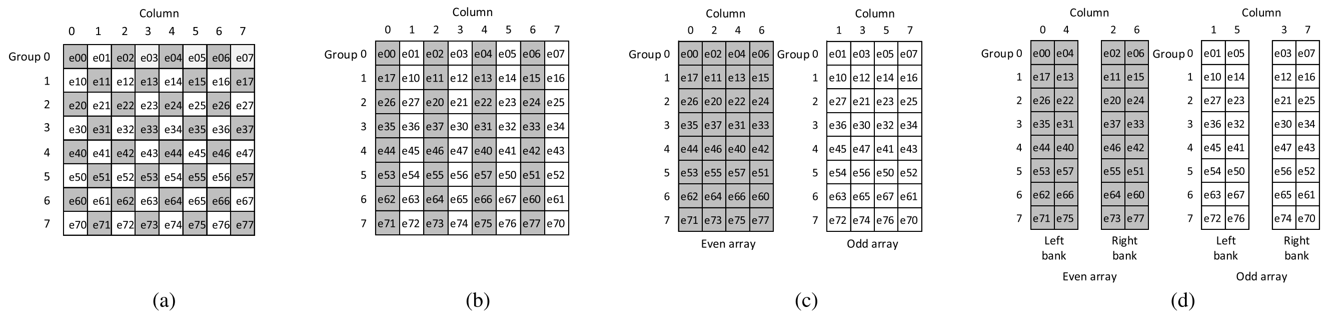


Fig. 5. PDE mapping on MAC SRAM. (a) Direct mapping. (b) Rotation mapping. (c) Array splitting. (d) Bank splitting.

right bank. The respective BLs from the two banks are joined to complete the summing, and 2) groups 1 and 2 are activated on the left bank, whereas groups 0 and 1 are activated on the right bank. The respective BLs are then joined to complete the summing.

The rotation mapping and the array and bank splitting offer a number of advantages: 1) the memory is nearly fully utilized to support stencil computation without duplicate storage; 2) simple and regular BL muxing and control; and 3) compatible with the hybrid layer update method. One possible drawback of the approach is the lower array efficiency due to the splitting into smaller SRAM arrays and banks. However, for a sufficiently large grid size that is common in the most demanding applications, the loss of efficiency due to banking is minimized.

We also note that in a typical PIM design, all rows of the memory are activated at the same time to unleash the full parallelism [6], [9], but it is at the cost of reduced BL precision. Our approach activates a subset of rows of the memory, that is, ten rows. It sacrifices the performance but also reduces the BL precision and simplifies the ADC design.

B. Dataflow Mapping

The PDE solver chip architecture is designed to implement the dataflow shown in Fig. 3(c). The dataflow's two main parts, fine-grid and coarse-grid computes, each consists of three steps: iterations, residual update, and solution update, as shown in Fig. 3(c). Each step is mapped to a module shown in Fig. 6. The iteration module leverages MAC SRAM to perform iterative layer update in 5 b to solve for the errors, $\mathbf{e}$ or $\mathbf{e}^*$. After a round of iterations, the residual update module is called to apply the errors in updating the residuals, $\mathbf{r}$ or $\mathbf{r}^*$, and the solution update module is called to accumulate 5-b errors to update the full-precision $\mathbf{u}$ values in memory. Fine-grid compute and coarse-grid compute are done by the same hardware, with coarse grid utilizing a fraction of the hardware.

V. PROTOTYPE ARCHITECTURE

A prototype PDE solver chip is designed in a 180-nm CMOS following the architecture in Fig. 7. The iteration module consists of four MAC SRAM arrays. A pair of arrays is used together as the even and odd arrays to support the rotation mapping. In the prototype design, each array consists of

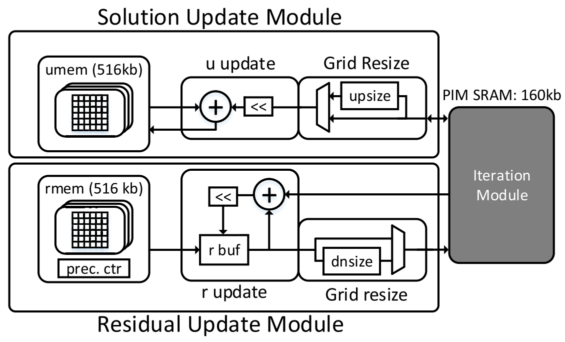


Fig. 6. Top-level architecture of PDE solver.

320×64 8T SRAM cells that are completed with peripherals. The 8T SRAM cell used in this chip is custom designed following normal design rules. The cell area of the custom 8T SRAM cell is 2.63× the size of a standard 6T SRAM cell in the 180-nm technology. If we used "push rule" (common in memory designs), the custom 8T SRAM cell area can be shrunk by 20% or 2.10× the size of a standard 6T SRAM cell. Though the density of our custom 8T SRAM is lower, the MAC SRAM is used for both storage and compute rather than storage alone, and memory data movement is eliminated and no separate MAC circuits are needed.

The PDE iteration module can be used to compute two independent grids of up to 64×128 (5-b grid values), or they can be joined to support a grid of up to 128×128 (5-b grid values). The precision is configurable from 1 b to 5 b. A separate memory is used to store the offsets $\mathbf{c}$. Offset subtraction is done at the output of each array. A buffer is added to store the updated solutions for writing back to the neighboring even or odd array.

A 320×64 MAC SRAM array is internally split into two 320×32 banks to be used as the left and right banks, as shown in Fig. 8. The MAC SRAM array occupies 0.467 mm^2 in 180-nm CMOS and is clocked at 200 MHz. It provides two ports: a single read/write port for normal memory access and a group read port for MAC operations. In the MAC mode, up to 20 WLs (i.e., four 5-b groups, two for each bank) are selected in parallel by the group decoder. A 5-b width-modulation of WL is controlled by a DLL, and 5-b level-modulation is done via current mirrors. Select muxes allow the analog summation of partial sums from the two banks. The 32 merged BLs are digitized by 32 5-b ADCs.

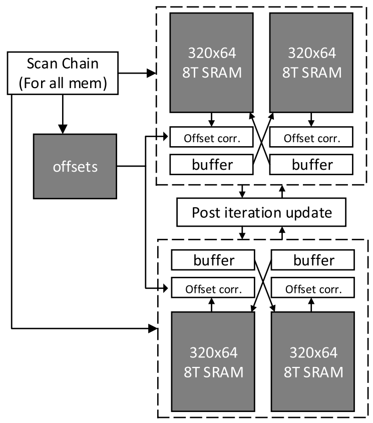


Fig. 7. Architectural sketch of PDE iteration module.

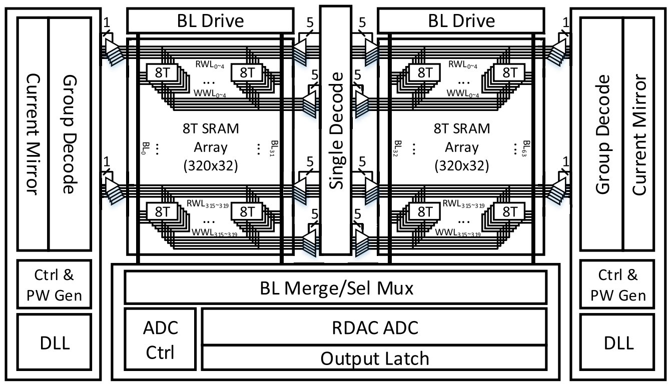


Fig. 8. Block diagram of MAC SRAM.

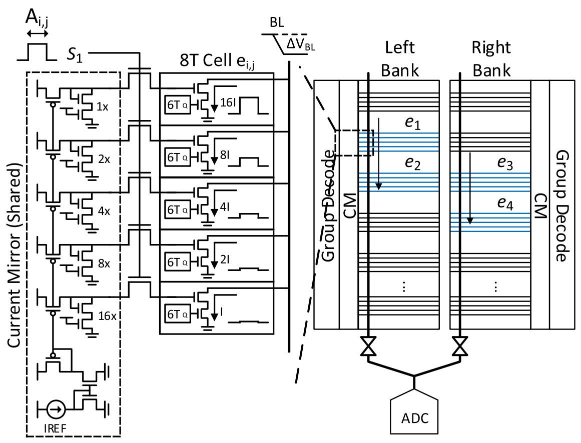


Fig. 9. Illustration of group read for MAC operations.

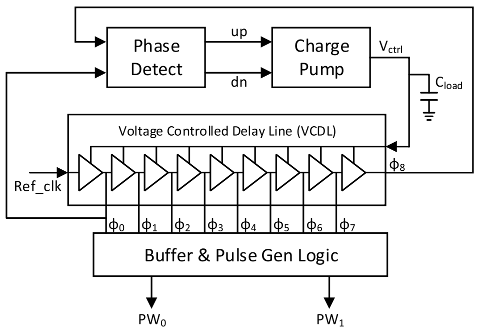


Fig. 10. DLL design for WL pulse generation.

VI. Group Read and WL Pulse Generation

The group read mode is illustrated in Fig. 9, showing four stencil entries (s_1, s_2, s_3, s_4) applied to four error vectors (e_1, e_2, e_3, e_4) stored in 20 rows of the SRAM banks for MAC operations. The MAC operations are conducted in groups in the following manner: 1) the group decoder turns on the access to a group of five SRAM rows; 2) the WL pulsewidth (PW) is selected by a 5-b stencil entry; 3) current mirrors generate the WL voltage needed to provide $1\times$, $2\times$, $4\times$, $8\times$, and $16\times$ unit cell current for the analog readout of 5-b error values; and 4) the products between the stencil entry and the error values are accumulated on the BLs. Up to four groups are activated at the same time to enable 128 5 b $\times$ 5 b MACs in parallel in the MAC SRAM.

If we use the 5-ns clock period as the unit PW, a 5-b WL pulse will take 32 clock cycles or 160 ns. To improve performance, we use 625 ps as the unit PW, so a 5-b WL pulse only takes 20 ns. To generate fine and well-controlled WL pulses, we design a DLL to subdivide a 5-ns clock period to a 625-ps unit PW using an eight-stage voltage-controlled delay line in a control loop and a pulse generator logic, as shown in Fig. 10. The phases are continuously adjusted by tracking the 200-MHz reference clock using the phase detector, and errors are corrected by the control voltage of the delay line.

We allocate up to 200-mV BL swing to represent the readout of one group. With all four groups activating at the same time, the BL swings up to 800 mV, from 1.8 V down to 1.0 V. The swing is limited by the one-stage pre-amplifier of the ADC. The pre-amplifier performs offset cancellation at its output for a lower area and power, but the input common-mode range is more limited. Since 800-mV BL swing is digitize by a 5-b ADC, an LSB step is 25 mV. The process variation is evaluated by Monte Carlo simulations for reading a group of 5-b operands. Variations were added to the memory array, the current mirrors, and the WL drivers in this simulation. To obtain the maximum absolute variation, we used the maximum PW of 19.375 ns (31-unit PW). The BL voltage and its standard deviation are shown in Fig. 11. The standard deviation of the BL voltage is shown to be limited to 18 mV.

The DLL occupies 1500 μm^2 in 180 nm and consumes 950 μW. The differential nonlinearity (DNL) and the integral nonlinearity (INL) for the DLL are evaluated for all process corners. The DLL provides a maximum INL of sub-0.15-unit PW, as shown in Fig. 12. The closed-loop pulse generation is more robust than an open-loop approach [10].

VII. BL Readout

The BL ADC needs to be compact and energy efficient to avoid becoming a bottleneck of the design. Therefore, flash or SAR architectures are excluded. Instead, we choose a ramp ADC that consists of a ramp reference and a counter shared by all columns, and a single comparator and latch per column. The ramp architecture minimizes the area and energy, but a 5-b conversion requires 32 time steps.

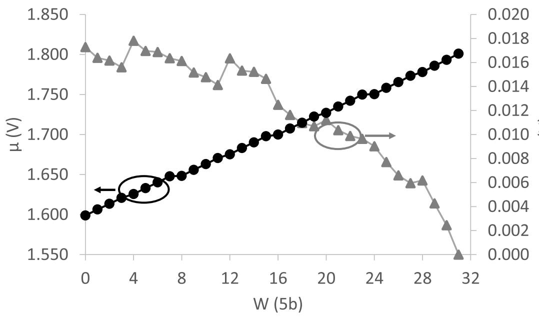


Fig. 11. Statistics of the Monte Carlo simulations of BL voltage for a multiplication operation (the left axis is for the average BL voltage, and the right axis is for the standard deviation of the BL voltage. The dotted curve shows the average BL voltage when a pulse representing the maximum input of 31 is multiplied by the weight values stored in memory, and the triangled curve shows the standard deviation of the BL voltage).

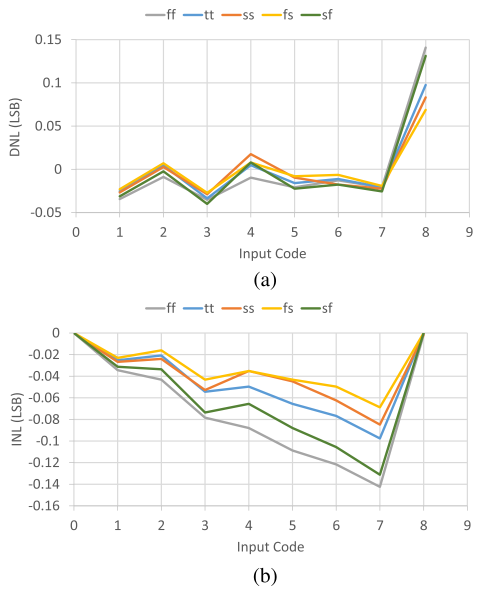


Fig. 12. (a) DNL and (b) INL of WL pulses generated by DLL.

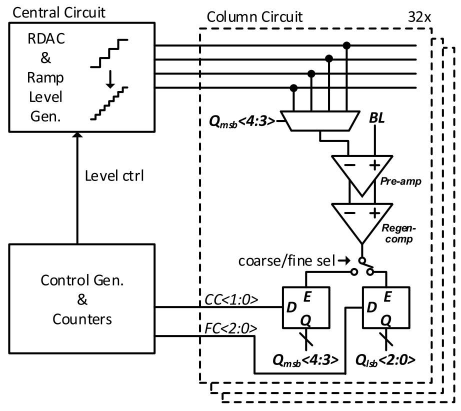


Fig. 13. 5-b DRSS ADC design for BL readout.

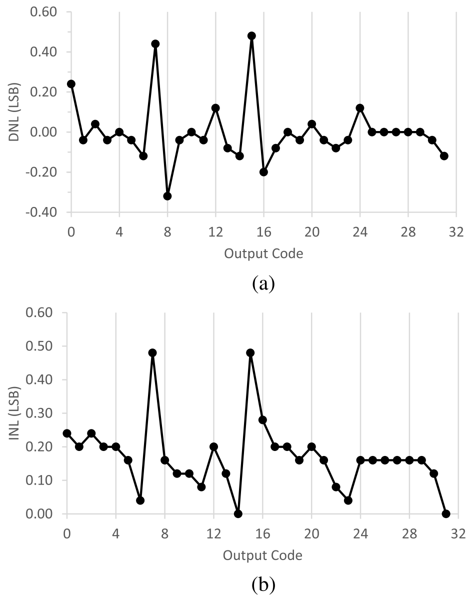


Fig. 14. ADC (a) DNL and (b) INL.

We adopt a compact DRSS ADC architecture [12] that applies a 2-b coarse-ramp comparison followed by a 3-b fine-ramp comparison, as shown in Fig. 13. The BL voltage is first compared with the 2-b coarse ramp to obtain the 2-b MSB, which then selects one of four 3-b fine ramps for comparison to obtain the 3-b LSB. The dual-ramp approach performs 5-b comparison in $2^2 + 2^3 = 12$ time steps, faster than a serial conversion architecture [10].

In implementing DRSS ADCs, a central circuit is shared by 32 columns, and it generates two ramps by a resistive DAC and a controller that steps through the two conversion phases. A compact column circuit consists of a pre-amplifier followed by a regenerative comparator and latches.

The column circuit measures only 350 μm $\times$ 110 μm. The 32 ADCs in a MAC SRAM occupy 0.044 mm^2, and the conversion costs 8.91 mW at 200 MHz. The DNL of the ADC is kept below 0.45 b and the INL of the ADC is within 0.5 b, as shown in Fig. 14.

VIII. Results and Discussion

A 180-nm 11.0-mm^2 PDE solver test chip was fabricated and tested. The chip consists of a PDE solver and BIST circuits, as shown in Fig. 15. The four MAC SRAMs in the PDE solver core each takes 570 μm $\times$ 820 μm and dissipates 16.6 mW when performing group read at 200 MHz and room temperature.

The ADC, DLL, and decoder/current mirror account for 62%, 12%, and 9% of the power consumption shown in Fig. 16. When running Jacobi and the hybrid layer update iterations, the 5-b multigrid PDE solver reaches an error tolerance of 10^{-8} while speeding up convergence by 6$\times$ and 8$\times$, respectively, over the baseline 32-b single-grid implementation, as shown in Fig. 17.

The 200-MHz MAC SRAM completes 128 5 b $\times$ 5 b MAC operations in 18 clock cycles (four-cycle WL pulse, one-cycle BL propagation, 12-cycle ADC, and one-cycle latching). With four MAC SRAMs, the PDE solver chip performs 512 5 b $\times$ 5 b MAC operations every 18 clock cycles. Following [10] that counts an operation at each active SRAM cell as two OPs, the performance and energy of each MAC SRAM

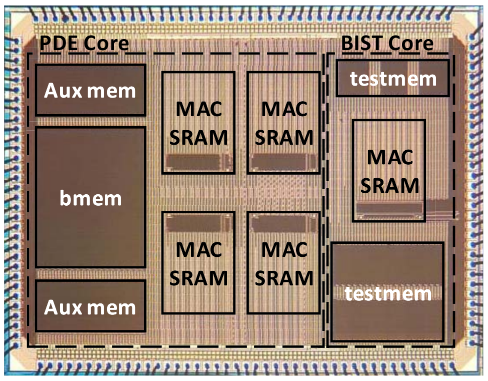


Fig. 15. Microphotograph of PDE solver test chip.

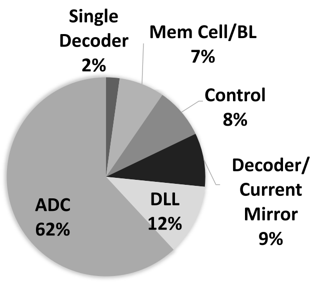


Fig. 16. Breakdown of MAC SRAM power.

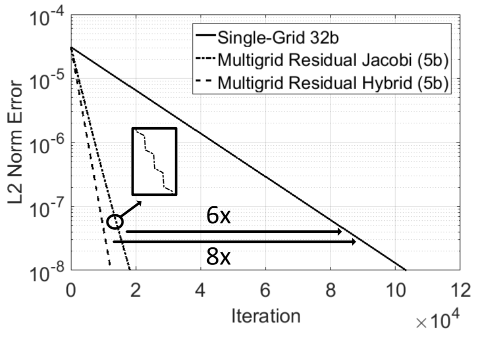


Fig. 17. Convergence of 5-b multigrid (5-b fine grid: 127 × 127, 5-b coarse grid: 64 × 64) using Jacobi and hybrid layer update iterations compared to single-grid baseline implementation (32-b fine grid: 127 × 127). Results are based on solving 2-D Poisson's equation.

are 14.2 GOPS and 857 GOPS/W, respectively. At a lower precision, the performance and energy efficiency can be more than double, as shown in Fig. 18.

For reference, we synthesized a digital ASIC in the same 180-nm CMOS technology. The digital ASIC is designed to match our chip in terms of computational capability needed to perform the same task. The digital ASIC consists of 16 KB of SRAM (compiled 6T SRAM macros) to store the grid values, 128 multipliers to compute 128 products in parallel (match the parallelism of our chip), an adder tree to sum the products, and the necessary buffers and pipeline registers. The synthesized ASIC has a clock frequency of 90.9 MHz. It occupies 22.137 mm^2 and consumes 3.392 W. The ASIC is pipelined to complete one set of 128 MACs in six cycles.

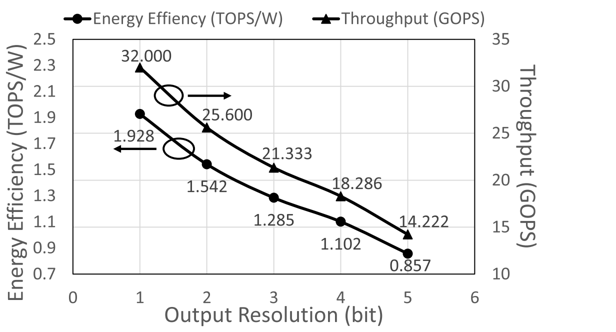


Fig. 18. Measured performance and energy.

TABLE I
COMPARISON WITH PRIOR PDE ACCELERATORS (UNNORMALIZED)

	Cowan [4]	Guo [5]	This work
Technology	250nm	65nm	180nm
Core voltage	2.5V	1.2V	1.8V
Core area (mm^2)	100	3.8	1.868
# Active PEs	400[a]	26[a]	512[b]
Compute precision	8b	16b	5b
Core frequency	20kHz	25kHz	200MHz
Grid Update Rate (MEntries/s)	1.6[c]	0.04[c]	1380
Power (mW)	300	1.2	66.4
Efficiency (MEntries/s/W)	0.005	0.033	20.8
Density (MEntries/s/mm^2)	0.016	0.01	739

[a]1 PE represents an analog operation block
[b]1 PE represents a 5b multiplier unit for MAC operation
[c]Assume outputs are continuously transferred at full bandwidth.

The compute density of the ASIC is nearly 35× lower than the MAC SRAM chip.

A. Comparisons of PDE Solver Chips

This design is the first PIM that targets solving PDEs. Prior PIM designs do not meet the requirements of the PDE solver due to limited multiplicand precision [6], [9], [10], limited ADC resolution [6], [9], or limited number of ADCs [7], [8].

In Table I, this article is compared with recently published silicon implementations of accelerators for solving PDEs: an analog computer accelerator [4] and a hybrid computing unit [5]. This article is the first to use PIM in PDE applications. It is also the first among hardware accelerators to use a multi-grid residual approach to reduce the core precision requirement to 5 b. An intrinsic benefit of PIM is less data movement, compared to the other work that relies on frequent accesses of external DRAMs. Optimized toward dense, low-precision compute, this article achieves a grid update rate of 1.38-G entries/s and an energy efficiency of 20.8-M entries/s/W. The overhead of loading from buffers and unloading to buffers in each iteration, restriction, and interpolation between fine-grid compute and coarse-grid compute, and the incomplete utilization of the hardware in coarse-grid compute haveall been accounted for in the measurements.

Compared to [4] in 250 nm, our 180-nm design's performance, performance per watt, and performance per unit silicon area all are more competitive by over two orders of magnitude. Similarly, compared to [5] in 65 nm, our 180-nm design's performance, performance per watt, and performance per unit silicon area are also more competitive by over two orders of magnitude. Cowan *et al.* [4] used a 8-bit precision, and Guo *et al.* [5] used a 16-bit precision. We opted for an optimized multi-grid residual approach to reduce the precision of our hardware to 5 bits, but our multi-grid 5-bit residual design could reach the same accuracy of a 32-bit precision, single-grid design in 5× fewer iterations, as shown in Fig. 4.

B. Technology Scaling, Porting, and Voltage Scaling

To support 5-b readout, combining four groups, and a sufficient variation tolerance, our design requires an 800-mV voltage headroom. 180-nm technology is a good fit for this design. With the 180-nm technology, we show that our hardware achieves two orders of magnitude higher energy efficiency and four orders of magnitude higher compute density than the 65-nm competitor [5] for the same application. This result demonstrates that implementation in an older technology can be competitive if it is used in a suitable way. If we scale our design to more advanced technology with a smaller voltage headroom and larger variations, we will need to modify the design to reduce the number of groups to combine (lower parallelism) and a possibility to use multiple bitlines to provide the required precision (lower compute density). Since more advanced technology is denser, faster, and lower power, it can offset the sacrifice in parallelism and compute density.

Mixed-signal circuit scaling with respect to improved technology nodes is a much more complex function than digital circuit scaling. This is especially true in a high-precision ADC, which is limited by thermal noise [17]. However, for a low-precision ADC, such as our 5-b DRSS ADC, the design scales with technology. There are two factors that contribute to the scalability: 1) about 2/3 of the ADC used in this article comprises digital circuit elements (switches, logic gates, and flip-flops), which scale with technology, and 2) while not inherently digital, other elements such as comparators and amplifiers gain implicit benefits such as less internal capacitive loading, which can help to improve the overall efficiency.

We did two trial designs in 65 nm with halved BL swing. Design A (this work) requires 800-mV BL swing, and the two 65-nm trial designs, Designs B and C, each uses only 400-mV BL swing due to the reduced voltage headroom.

Due to the reduced BL swing, Design B only offers 4 b of resolution. As shown in Fig. 4, 4-b resolution is still sufficient for the PDE solver. The drawbacks are the reduced design margin and 10% extra iterations to reach convergence. We performed a trial design to estimate the area and power of SRAM, ADC, and peripheral circuits. The SRAM design references [8], and the ADC design references [10]. The results show that the 65-nm Design B achieves 10.6× higher compute density and 9.2× higher energy efficiency compared to this article, demonstrating that the design is scalable.

Design C also uses a 400-mV BL swing, but it relaxes the design constraints by quantizing BL output to only 3 b. 3 b is insufficient for the PDE solver, so we use twice as many cells to provide an effective 6-b resolution. The 3-b design relaxes all the analog and peripheral component designs at the cost of a larger area. We performed a trial design to estimate the area and power of SRAM, ADC, and peripheral circuits. The results show that the 65-nm Design C still achieves 4.7× higher compute density and 5.3× higher energy efficiency compared to this article, demonstrating that the design is scalable.

To port this design to another technology node, the MAC SRAM module needs to be custom designed. However, the memory array is regular and can be instantiated. It may be possible to reuse a commercially available standard 8T SRAM array but with replaced peripheral circuitry. The peripheral circuitry, including current mirror, DLL, and ADC are custom, but once one unit is designed, it can be instantiated. Other than the MAC SRAM, the remaining parts of the PDE solver can be digitally synthesized. To sum up, some effort is required to port the custom parts of the design, but a hierarchical, reuse approach will significantly simplify the porting process.

Lastly, we note that PIM is analog compute, and it cannot be voltage-scaled easily. This also applies to analog compute in general. However, if the bit precision is low, analog computation is more advantageous because of higher compute density. PIM also removes the data movement cost, an added bonus. If the required precision is high, conventional digital implementation is more advantageous in terms of energy and accuracy, and it can be voltage scaled to save power. Therefore, there is hardly a reason to use analog compute for high precision.

IX. Conclusion

Numerical PDE solvers require high-precision, iterative, and memory-intensive computation. In this article, we adopt a residual form of the multigrid method to reduce the precision requirement and a hybrid layer update to reduce the computation time while providing sufficient parallelism.

The resulting PDE solver design is mapped to a 5-b SRAM-based PIM system that consists of an iteration module, a solution update module, and a residual update module. Quantized grid values are mapped to SRAM following a rotation mapping method for high storage utilization and efficient parallel computation. Four 320 × 64 SRAMs perform parallel 5 b × 5 b MAC operations, with 5-b WL-level modulation and 5-b PW modulation. Each MAC SRAM output is digitized by 32 5-b DRSS ADCs.

The PDE solver is prototyped in a 11-mm^2 180-nm test chip. The chip is measured to achieve a grid update rate of 1.38-G entries/s at 200 MHz at a power consumption of 66.4 mW. Compared to previously published PDE solver accelerator chips, this article demonstrates two orders of magnitude improvement in energy efficiency and four orders of magnitude improvement in compute density without technology normalization. The results show the promise of using PIM in numerical PDE solver applications.

Acknowledgment

The authors would like to thank Dr. S. Song, Prof. W. Lu, and Prof. M. Kushner for their advice.

REFERENCES

[1] R. S. Varga, *Matrix Iterative Analysis*, vol. 27, 2nd ed. Springer, 2000.

[2] Y. Huang, N. Guo, M. Seok, Y. Tsividis, and S. Sethumadhavan, “Evaluation of an analog accelerator for linear algebra,” in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, vol. 44, no. 3, pp. 570–582.

[3] J. Kung, Y. Long, D. Kim, and S. Mukhopadhyay, “A programmable hardware accelerator for simulating dynamical systems,” in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 403–415.

[4] G. Cowan, R. Melville, and Y. Tsividis, “A VLSI analog computer/digital computer accelerator,” *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 42–53, Jan. 2006.

[5] N. Guo *et al.*, “Energy–efficient hybrid analog/digital approximate computation in continuous time,” *IEEE J. Solid-State Circuits*, vol. 51, no. 7, pp. 1514–1524, Jul. 2016.

[6] J. Zhang, Z. Wang, and N. Verma, “In–memory computation of a machine–learning classifier in a standard 6T SRAM array,” *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[7] M. Kang, S. Gonugondla, A. Patil, and N. Shanbhag, “A 481pJ/decision 3.4M decision/s multifunctional deep in-memory inference processor using standard 6T SRAM array,” 2016, *arXiv:1610.07501*. [Online]. Available: https://arxiv.org/abs/1610.07501

[8] S. K. Gonugondla, M. Kang, and N. Shanbhag, “A 42pJ/decision 3.12 TOPS/W robust in-memory machine learning classifier with on-chip training,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 490–492.

[9] W.-S. Khwa *et al.*, “A 65 nm 4 Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary dnn edge processors,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 496–498.

[10] A. Biswas and A. P. Chandrakasan, “Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 488–490.

[11] T. Chen, J. Botimer, T. Chou, and Z. Zhang, “An SRAM–based accelerator for solving partial differential equations,” in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2019, pp. 1–4.

[12] M. F. Snoeij, P. Donegan, A. J. Theuwissen, K. A. Makinwa, and J. H. Huijsing, “A CMOS image sensor with a column-level multiple-ramp single-slope ADC,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2007, pp. 506–618.

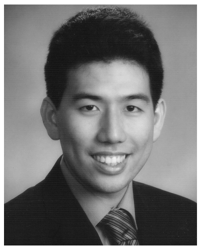
[13] P. Pérez, M. Gangnet, and A. Blake, “Poisson image editing,” *ACM Trans. Graph. (TOG)*, vol. 22, no. 3, p. 313, Jul. 2003.

[14] Y. Saad, *Iterative Methods for Sparse Linear Systems*, vol. 82. Philadelphia, PA, USA: SIAM, 2003.

[15] Y. Xu, “Hybrid Jacobian and Gauss–Seidel proximal block coordinate update methods for linearly constrained convex programming,” *SIAM J. Optim.*, vol. 28, no. 1, pp. 646–670, Jan. 2018.

[16] D. M. Young, *Iterative Solution of Large Linear Systems*. Amsterdam, The Netherlands: Elsevier, 2014.

[17] B. Murmann, “A/D converter trends: Power dissipation, scaling and digitally assisted architectures,” in *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2008, pp. 105–112.



Thomas Chen (Student Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2013, 2015, and 2019, respectively.

He did an internship with Circuits Research Laboratory, Intel Corporation, in 2015. His research interests are high-speed and low-power VLSI circuits and systems.

Dr. Chen received the Rackham Merit Fellowship from the University of Michigan in 2013 and the NSF Graduate Research Fellowship in 2015.

Jacob Botimer (Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2016 and 2019, respectively, where he is currently pursuing the Ph.D. degree in electrical engineering.

He did internships with Power Electronics Group, Texas Instruments, Dallas, TX, USA, in 2015 and 2016. In 2019, he joined the start-up company MemryX, Ann Arbor, where he has been working on computer architecture and circuit design for hardware accelerators. His research interests focus on in-memory computing and 2.5-D integration.

Teyuh Chou (Student Member, IEEE) received the B.S. degree in electrical engineering from National Central University, Taoyuan City, Taiwan, in 2013, and the M.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2016. She is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Michigan, Ann Arbor, MI, USA.

Her current research interests include nanoscale neuromorphic computing systems, RRAM-based hardware neural networks, efficient machine-learning accelerators, and memory-centric architectures.

Zhengya Zhang (Senior Member, IEEE) received the B.A.Sc. degree in computer engineering from the University of Waterloo in 2003 and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley (UC Berkeley), Berkeley, CA, USA, in 2005 and 2009, respectively.

He has been a Faculty Member with the University of Michigan, Ann Arbor, MI, USA, since 2009, where he is currently an Associate Professor with the Department of Electrical Engineering and Computer Science. His current research interests include low-power and high-performance VLSI circuits and systems for computing, communications, and signal processing.

Dr. Zhang was a recipient of the David J. Sakrison Memorial Prize from UC Berkeley in 2009, the National Science Foundation CAREER Award in 2011, the Intel Early Career Faculty Award in 2013, and the University of Michigan College of Engineering Neil Van Eenam Memorial Award in 2019. He serves on the Technical Program Committees of the Symposium on VLSI Circuits and the IEEE Custom Integrated Circuits Conference (CICC) since 2018. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS from 2013 to 2015 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS–PART II: EXPRESS BRIEFS from 2014 to 2015. He has been an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS since 2015.