Solving Sparse Linear Systems via Flexible GMRES with In-Memory Analog Preconditioning

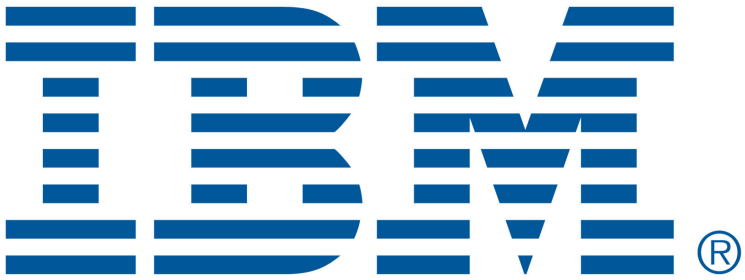
Vassilis Kalantzis at al.

September 2023

EPrint ID: 2023.6

IBM Research
Thomas J. Watson Research Center

Preprints available from:
https://researcher.watson.ibm.com/researcher/view.php?person=ibm-vkal



Solving Sparse Linear Systems via Flexible GMRES with In-Memory Analog Preconditioning

Vasileios Kalantzis, Mark S. Squillante, Chai Wah Wu, Anshul Gupta,
Shashanka Ubaru, Tayfun Gokmen, and Lior Horesh
MIT-IBM Watson AI Lab & IBM Research
Email: {vkal,Shahanka.Ubaru}@ibm.com, {mss,cwwu,anshul,tgokmen,lhoresh}@us.ibm.com

***Abstract*—Analog arrays of non-volatile crossbars leverage physics to compute approximate matrix-vector multiplications in a rapid, in-memory fashion. In this paper we consider exploiting this technology to precondition the Generalized Minimum Residual iterative solver (GMRES). Since the preconditioner must be applied through matrix-vector multiplication, approximate inverse preconditioners are a natural fit. At the same time, the errors introduced by the analog hardware render an iteration matrix that changes from one iteration to another. To remedy this, we propose to combine analog approximate inverse preconditioning with a flexible GMRES algorithm that naturally incorporates variations of the preconditioner into its model. The benefit of our approach is that the analog circuit is much simpler than correcting the errors at the hardware level. Our experiments with a simulator for analog hardware show that such an analog-flexible scheme can lead to fast convergence.**

***Index Terms*—Analog hardware, preconditioning, flexible GMRES, sparse linear systems**

I. Introduction

The fast iterative solution of sparse systems of linear algebraic equations is one of the most common computational tasks encountered in computational science and engineering (1; 2). These linear systems are typically solved by a preconditioned Krylov subspace iterative solver (3). For non-Hermitian sparse systems, a commonly used Krylov subspace iterative solver is the preconditioned Generalized Minimum Residual (PGMRES) algorithm (4).

Traditionally, the solution of sparse linear systems is computed in double precision and the wall-clock time required to compute an approximate solution is a function of the convergence rate of the iterative process as well as how fast digital hardware can execute common basic linear algebraic operations, such as matrix-vector multiplications (MVMs), vector additions, and dot products. Over the past decades, the execution time of these operations keeps reducing as a by-product of digital microprocessors becoming faster through an increase in the number of transistors, in accordance with Moore's law (5) and a reduction in the size of transistors. The advent of machine learning, and deep learning in particular, led to significant hardware advances in low-precision arithmetic, starting in 2016 with the release of the Tesla P100 Graphics Processing Unit (GPU) accelerator from NVIDIA that delivered up to 21.2 Tera Floating-Point Operations per second (TFLOPS) of half-precision arithmetic, which represents a $4\times$ improvement over double-precision. The A100 GPU widened the gap between 16-bit and 64-bit arithmetic, delivering up to 624 TFLOPS of half-precision arithmetic, which represents an approximately $60\times$ improvement over double-precision arithmetic. In particular, most recent technological advances can deliver up to $10-100\times$ of Giga FLOPS (GFLOPS) even for sparse matrices (6). The speed-up of the MVM kernel has revived interest in the use of sparse approximate inverse preconditioners, such as preconditioners that approximate the matrix inverse directly and are applied through MVM; see, e.g., (7; 8) and most recently (9; 10; 11; 12; 13; 14).

Since approximate inverse preconditioners are applied through MVM, exploiting computational architectures that speed-up the execution of MVM can lead to significant reductions in the wall-clock time of preconditioned iterative linear system solvers. While GPUs have led to drastic speed-ups in the execution of MVMs as noted above, their design is based on the von Neumann architectural model. Thus, the performance of the MVM kernel, both in terms of execution time and energy consumption, is inherently limited by the latency occurring during the transfer of data from the random access memory. An alternative paradigm suggested in (15) is to apply approximate inverse preconditioners through in-memory computing devices based on non-volative analog crossbar arrays (16; 17). These devices can achieve high degrees of concurrency with low energy consumption by mapping matrices onto arrays of memristive elements capable of storing information and executing simple operations such as a multiply-and-add via Ohm's and Kirchhoff's laws. Such hardware devices are becoming more available (18).

In this paper we focus on preconditioning the GMRES algorithm by executing MVM with approximate inverse preconditioners through *analog crossbar arrays*. Preconditioning the GMRES algorithm through analog crossbar arrays has been previously explored in (19) where it was suggested to apply a domain decomposition overlapping preconditioner through in-situ matrix inversion. To enhance the accuracy of the preconditioning step due to analog noise, the authors considered a bit-slicing technique in conjunction with a circuit compensation approach. Similarly, analog hardware with precision enhancement has been exploited in the context of numerical PDEs (20). Other related work includes inner-outer iteration schemes for dense systems, where analog arrays were used for the inner solver while iterative refinement was used as an outer solver in the digital space (21; 22). The work presented in this

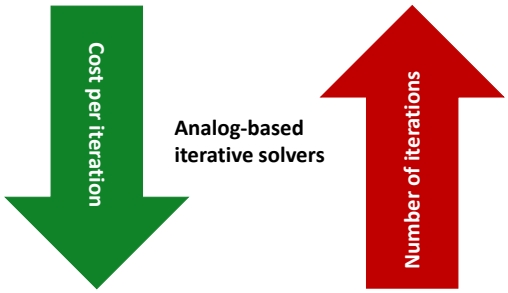


Fig. 1: *Trade-off of hybrid digital-analog algorithms.*

paper is quite different in the sense that we do not consider any additional hardware other than standard crossbar arrays. Instead, our proposed approach exploits this analog hardware and mitigates the analog noise and errors by considering a variant of GMRES, known as Flexible GMRES (23), which allows the preconditioner to vary from one iteration to another. The stochastic analog noise, which leads to a different error at each iteration, can be then understood as a perturbation of the original preconditioner mapped to analog hardware. Therefore, our approach addresses the inaccuracies stemming from the analog hardware at an algorithmic level, at the expense of storing a few additional vectors in digital memory. While these inaccuracies can lead to slower convergence, each preconditioned iteration can be performed much faster, as depicted in Figure 1.

II. Flexible GMRES

One major limitation of Krylov subspace iterative solvers, such as PGMRES, is that the Krylov subspace must be built using the same matrix operator. As a result, PGMRES with analog crossbar arrays is generally impossible unless additional analog circuitry is added to increase the accuracy of analog-based MVM. In this paper, instead of increasing hardware complexity to tame the inexactness of the analog hardware, we consider a variant of PGMRES that allows us to fuse the inexactness from the analog computations directly to the iterative computational model.

This approach is known as *Flexible GMRES* (FGMRES) (23), and was originally developed as a variant of PGMRES that allows the preconditioner to vary from one iteration to another. While in our case we always aim to apply the same preconditioner M, the random errors from the analog device can be treated as variations of the preconditioner using a backward-error argument. FGMRES is similar to the PGMRES algorithm with the exception that the fixed preconditioner M is replaced by the preconditioner $M_j \in \mathbb{R}^{n\times n}$ during the j-th iteration. The preconditioner M_j can be either a perturbation of M, e.g., a lower precision representation of M, or a completely different preconditioner. When $M_j = M,\ j = 1,\dots,m$, FGMRES is mathematically equivalent to PGMRES with right preconditioning. A detailed description of the FGMRES algorithm is listed in Algorithm 1.

Similarly to PGMRES, the inner iteration loop of FGMRES is divided into two parts: a) the Arnoldi process (Steps $3-8$); and b) the computation of the approximate solution x_m (Step 9). Recall now that PGMRES computes $x_m = MV_m y_m$, and thus expresses x_m as a linear combination of the vectors $Mv_j,\ j = 1,\dots,m$. When M is constant, the formation of x_m only requires applying M to the vector $V_m y_m$. In contrast to PGMRES, FGMRES bypasses the preconditioned Krylov subspace and computes an approximate solution $x_m \in x_0 + \texttt{range}(Z_m)$ where the columns of the matrix $Z_m = [z_1,\dots,z_m]$ with $z_j = M_j v_j$ replace the Krylov subspace directions of PGMRES. While FGMRES does not produce a Krylov subspace, it still satisfies the following Krylov-like relation:

$$AZ_m = V_{m+1}H_{m+1,m},$$

where $V_{m+1} = [v_1,\dots,v_m,v_{m+1}]$ is orthogonal, and the matrix $H_{m+1,m} = V_{m+1}^H AZ_m$ is upper-Hessenberg. Notice that while FGMRES generates the next preconditioned Krylov vector as $z_j = M_j v_j$, its definition in fact allows $z_1,\dots,z_m$ to be chosen even entirely randomly, as long as Z_m has full rank.

Let now $x_m = x_0 + Z_m y_m$ where $y_m \in \mathbb{R}^m$. The corresponding residual r_m satisfies

$$\begin{aligned} r_m = r_0 - Ax_m &= r_0 - AZ_m y_m \\ &= \|r_0\|v_1 - V_{m+1}H_{m+1,m}y_m \\ &= V_{m+1}(e_1\|r_0\| - H_{m+1,m}y_m). \end{aligned}$$

Recalling that V_{m+1} is orthogonal, we have

$$\|r_0 - Ax_m\| = \big\|(e_1\|r_0\| - H_{m+1,m}y_m)\big\|.$$

The coefficient vector $y_m = \arg\min_{y\in\mathbb{R}^m} \big\|(e_1\|r_0\| - H_{m+1,m}y)\big\|$ is computed such that $x_m = x_0 + Z_m y_m$ satisfies $x_m = \underset{z\in x_0+\text{range}(Z_m)}{\arg\min} \|r_0 - Az\|$.

Algorithm 1 Flexible GMRES (FGMRES)

1: **input**: $A \in \mathbb{R}^{n\times n}$; $b \in \mathbb{R}^n$; $x_0 \in \mathbb{R}^n$; $t_{\text{ol}} \in \mathbb{R}$; $m \in \mathbb{N}$. Set $e_1 = [1,0,\dots,0]^\top \in \mathbb{R}^m$.
2: Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|$, and set $v_1 = r_0/\beta$, $Z = 0$
3: **for** $j = 1$ **to** m **do**
4: Compute $z_j = M_j v_j$ and augment $Z = [Z, z_j]$
5: Compute $w = Az_j$
6: For $i = 1,\dots,j$: $\begin{cases} h_{i,j} = w^\top v_i \\ w = w - h_{i,j}v_i \end{cases}$
7: Set $h_{j+1,j} = \|w\|$ and $v_{j+1} = w/h_{j+1,j}$
8: **end for**
9: Solve $y_m = \underset{y\in\mathbb{R}^m}{\arg\min} \|\beta e_1 - H_{m+1,m}y\|$
10: Compute $x_m = x_0 + Z_m y_m$
11: If $\|r_m\| \le t_{\text{ol}}\|r_0\|$, exit; else, restart from Step 2 with $x_0 = x_m$

III. FGMRES with In-Memory Analog Preconditioning

A. Approximate inverse preconditioners

The convergence rate of PGMRES depends on the spectrum of the preconditioned iteration matrix AM, where fast convergence is typically ensured when the eigenvalues of AM are clustered around 1. In most cases, the default choice for the preconditioner is to compute an incomplete LU (ILU) factorization $A \approx LU$ and set $M = (LU)^{-1}$ (3). The matrices $L \in \mathbb{R}^{n\times n}$ and $U \in \mathbb{R}^{n\times n}$ are lower and upper triangular, respectively. ILU preconditioners can be quite efficient as general-purpose preconditioners, e.g., when A is diagonally dominant. However, for indefinite or non-diagonally dominant matrices, the norm of the matrices L^{-1} and U^{-1} can be quite large and/or encounter zero pivots. In addition, the sequential nature of triangular substitutions generally leads to poor computational performance in parallel computing environments.

Since analog hardware is a highly efficient execution platform for MVMs, it is appropriate to consider preconditioners M that can be applied via MVM. This can be achieved by setting up M such that it is a direct approximation of the matrix inverse A^{-1}. From an optimization perspective, such a preconditioner can be obtained by solving the optimization problem $\arg\min_{M\in\mathbb{R}^{n\times n}} \|I - AM\|_F^2$ up to a sparsity constraint (24; 25; 26; 27; 28). In contrast to incomplete factorizations, approximate inverse preconditioners are generally more robust than ILU for indefinite problems. On the other hand, the inverse of a sparse matrix is generally dense, and thus an efficient approximate inverse preconditioner M might be considerably more dense than A. As a result, the application of M via von Neumann systems introduces higher computational cost compared to ILU preconditioners. However, as we discuss next, once copied to analog hardware, the sparsity pattern of M does not have major effects on the execution of an analog MVM. This makes analog crossbar array hardware highly appealing for applying approximate inverse preconditioners.

B. Hybrid digital-analog architecture

The enormous success and applicability of deep neural networks over the last decade has revolutionized numerical computing and led to the development of scientific hardware that can tremendously accelerate important linear algebra kernels of neural network training, such as MVM, by taking advantage of reduced precision arithmetic (29). For example, a batched MVM (i.e., multiple MVMs bundled together) executing on an NVIDIA A100 GPU can achieve up to a few hundred TFLOPS. Nonetheless, GPUs are based on the von Neumann computer model, and thus they store the matrix data on separate units equipped with memory capacity, e.g., dynamic Random Access Memory. As a result, their performance for sparse to moderately dense non-batched MVM, as encountered in Krylov subspace iterative solvers such as GMRES, is generally much lower, e.g., up to a few hundred GFLOPS (6).

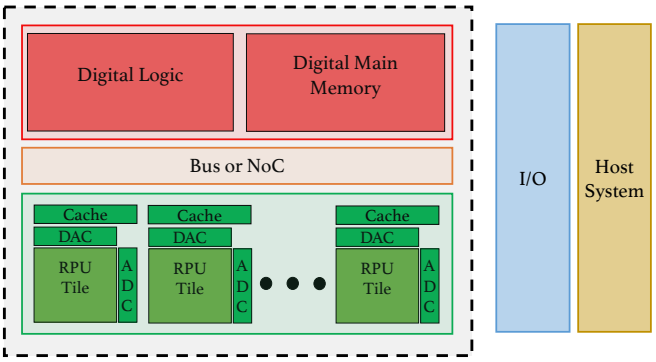


Fig. 2: *A hybrid digital-analog architecture consisting of a host system, an I/O bus, and an accelerator consisting of digital logic and one or more analog crossbar arrays of resistive processing units (RPUs).*

A memristor is an electronic device that remembers the amount of charge which previously flowed through it and can store numerical values via conductance in a non-volatile manner (30). When organized into a 2D crossbar array formation, the current at each cross point is equal to the product between the input voltage and stored conductance (Ohm's law), while the total current across each column is equal to the sum of the currents at each cross point (Kirchhoff's current law). The computation of the current at each cross point is performed in parallel, and sensing the accumulated current across each column is equivalent to computing an in-memory MVM. The time to perform this task is independent of the matrix size, thus providing a highly efficient approach to compute MVMs. Figure 2 depicts a hybrid digital-analog architecture that can be exploited to compute MVMs. The system consists of a host system, e.g., CPU, connected to an analog accelerator through an I/O bus. The accelerator is equipped with some digital logic to perform low-complexity operations and control the input/output of data to/from one or more tiles of analog crossbar arrays. When multiple tiles are available, the matrix M can be either distributed across tiles (e.g., if it is too big to fit on a single tile) or replicated among several tiles. A third option is to take advantage of additional tiles to perform bit-slicing that increases the accuracy of analog-based MVM, i.e., see (19; 31).

C. Modeling MVMs on analog crossbar hardware

We describe next a general procedure for computing MVM $y = Mr$ on analog crossbar array hardware. Without loss of generality, let us assume that the matrix M fits on a single analog crossbar array consisting of n rows and columns of conductors with a memristive element at each row-column intersection. The conductance of these elements can be set, reset or updated in a electrically programmable, non-volatile manner. The preconditioner can be copied to the crossbar array by scaling and mapping the nonzero matrix entries $M[i,j]$ (i.e., the (i,j)-th entry of the matrix M) to the conductance G_{ij} of the memristor at the intersection of row i and column j.

The MVM operation $y = Mr$ can be performed by sending pulses of V_{in} volts along the columns of the crossbar array such that the length t_j of the pulse along column j is proportional to the jth entry $r[j]$ of vector r, with suitable normalization. By Ohm's Law, this contributes a current equal to $V_{in}G_{ij}$ on the conductor corresponding to row i for a duration t_j. Following Kirchhoff's Current Law, the currents along each row accumulate and can be integrated over the time period equivalent to the maximum pulse length using capacitors, yielding a charge proportional to $\sum_{j=1}^{n} M[i,j]r[j]$, which in turn is proportional to $y[i]$. This integrated value can be recovered as a digital quantity via an analog-to-digital converter (ADC), and yields an approximation $\widehat{y}[i]$ of $y[i]$.

The above procedure involves multiple sources of non-deterministic noise so that the output $\widehat{y} \in \mathbb{R}^n$ is an approximation of y. Writing M to the crossbar array incurs multiplicative and additive write noise terms $N^{Wm} \in \mathbb{R}^{n\times n}$ and $N^{Wa} \in \mathbb{R}^{n\times n}$, respectively; and the actual conductance values at the crosspoints of the array is described by $\widehat{M} = M \odot (I + N^{Wm}) + N^{Wa}$, where $\odot$ denotes element-wise multiplication. Similarly, digital-to-analog conversion (DAC) of the vector r into voltage pulses suffers from multiplicative and additive input noise terms $N^{Im} \in \mathbb{R}^n$ and $N^{Ia} \in \mathbb{R}^n$, respectively. As a result, the matrix $\widehat{M}$ is effectively multiplied by a perturbed version of r given by $\widehat{r} = r\odot(\mathbf{1}+N^{Im})+N^{Ia}$, where $\mathbf{1}$ is a vector of all ones.

A characteristic equation to describe the output $\widehat{y}$ of an analog MVM Mr can be written as in (15):

$$\widehat{y} = \widehat{M}\widehat{r} \odot (\mathbf{1} + N^{Om}) + N^{Oa}, \tag{15}$$

where $N^{Om} \in \mathbb{R}^n$ and $N^{Oa} \in \mathbb{R}^n$ denote the multiplicative and additive components of the output noise, respectively. These components reflect the inherent inaccuracies in the multiplication based on circuit laws and current integration, as well as the loss of precision in the ADC conversion of the result vector. In addition to the various noises, another source of error in analog MVM is that the total charge from integrating the current along any row is bounded because it cannot exceed the capacity of the corresponding capacitor.

The stochastic noise characteristics of an analog crossbar array generally depend on its physical realization (21; 32), however such considerations are beyond the scope of this paper. Nonetheless, the behavior of analog arrays is generally captured by our model and our simulator listed in Section IV. The analog MVM computation $\widehat{y} = \widehat{M}\widehat{r}$ can be performed in near-constant time and, accounting for the $O(n)$ I/O cost for loading operands and reading the results, can achieve an $O(n)$ speedup in practice over its digital counterpart for a dense M. Therefore, a linear system solver that can tolerate analog MVM errors, can also lead to substantial reductions in the wall-clock time of the solver, even if it requires more iterations. This trend becomes more pronounced as the approximate inverse preconditioner M gets denser.

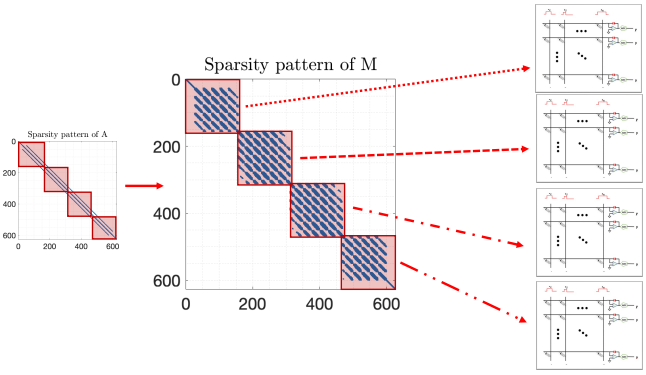


Fig. 3: *Mapping a block-Jacobi approximate inverse preconditioner to four separate analog crossbar arrays.*

D. Analog block-Jacobi preconditioning

Current technological considerations limit the maximum size of analog crossbar arrays to a few thousand rows and columns, which can be several orders of magnitude less than the size n of current practical matrix problems. More specifically, if we denote the size of a crossbar array by $\tau \in \mathbb{N}$, and assume that each matrix entry is copied to exactly one crossbar memory element, then storing the preconditioner M requires $\left\lceil \frac{n}{\tau} \right\rceil^2$ arrays, which becomes increasingly impractical as n increases. In addition, several of the entries of M can be zero, thus leading to a waste of hardware. Hence, as a practical alternative, we focus on *block-Jacobi preconditioners*, which reduce the analog hardware requirements by an integer factor $p \in \mathbb{N}$, where p denotes the level of coarsening of the block-Jacobi preconditioner. A diagram of our analog block-Jacobi preconditioning is shown in Figure 3 for the case $p = 4$.

The application of the block-Jacobi preconditioner $M = \text{blkdiag}\left(\left[M_{[1]}, \ldots, M_{[p]}\right]\right)$ onto the vector $r = \text{blkdiag}\left(\left[r_{[1]}, \ldots, r_{[p]}\right]\right)$ is equal to

$$Mr = \begin{bmatrix} M_{[1]}r_{[1]} \\ \vdots \\ M_{[p]}r_{[p]} \end{bmatrix},$$

where $M_{[j]}$ is stored at the jth analog crossbar array and each MVM $M_{[j]}r_{[j]}$ can be computed independently of each other. In addition to in-memory execution, a second advantage of analog block-Jacobi preconditioning is that the cost to perform an MVM with $M_{[j]}$ does not depend on its sparsity. Thus, the execution time of the MVM $M_{[j]}r_{[j]}$ is approximately identical among the p different arrays, leading to nearly optimal load balancing.

IV. Numerical Experiments

In this section we illustrate the behavior of FGMRES preconditioned via analog crossbar arrays on two typical model problems stemming from PDE discretizations. Our experiments were conducted in a Matlab environment (version R2020b) on a single core of a 2.3 GHz 8-Core Intel i9 machine equipped with 64 GB of system memory.

We used a Matlab version of the publicly available simulator (33) with a PyTorch interface for emulating the noise, timing, and energy characteristics of an analog crossbar array. The simulator models all sources of analog noise outlined in Section III-C as scaled Gaussian processes. Using Matlab notation, the components of the matrix write noise were modeled as $\texttt{randn}(\cdot) \times \texttt{5.0e}-\texttt{3}$, and those of the input and output noises were both modeled as $\texttt{randn}(\cdot) \times \texttt{1.0e}-\texttt{2}$; these are the default settings in the simulator based on currently realizable analog hardware (34). The number of bits used in the ADC and DAC was set to 7 and 9, respectively.

Throughout this section we consider the following schemes:

1) **GMRES**: non-preconditioned GMRES.
2) **PGMRES**: right-preconditioned GMRES. We consider preconditioning by approximate inverses ("**PGMRES + AI**") and ILU(0) ("**PGMRES + ILU**").
3) **IGMRES**: inexact, right-preconditioned GMRES. The approximate inverse preconditioner is applied through analog hardware.
4) **FGMRES**: flexible variant of GMRES, preconditioned by an approximate inverse through MVMs on analog hardware.
5) **FGMRES + Ric**(m_{inner}): inner-outer flexible variant of GMRES, preconditioned by $m_{\text{inner}} \in \mathbb{Z}$ iterations of Richardson iteration. Each iteration of the inner solver is preconditioned by an approximate inverse through MVMs on analog hardware, i.e., as described in (15).

Unless mentioned otherwise, the initial approximation for all GMRES variants will be set to zero and the restart cycle will be set to $m = 20$. For all GMRES variants we assume that convergence is established when the residual norm associated with the approximate solution x_m satisfies $\|b - Ax_m\| \leq \texttt{t}_{\text{ol}}\|b\|$, $\texttt{t}_{\text{ol}} \in \mathbb{R}$. By default, we set $\texttt{t}_{\text{ol}} = \texttt{1.0e}-\texttt{8}$. Table I lists the default parameters used throughout our experiments to simulate the analog hardware and construct the approximate inverse preconditioner via the SPAI algorithm (25).

TABLE I: *Default parameters.*

Module	Parameter	Value
Analog device	N^W	$\texttt{5.0e}-\texttt{3}$
Analog device	N^I	$\texttt{1.0e}-\texttt{2}$
Analog device	N^O	$\texttt{1.0e}-\texttt{2}$
GMRES	$\texttt{t}_{\text{ol}}$	$\texttt{1.0e}-\texttt{8}$
GMRES	$\texttt{m}_{\text{it}}$	250
GMRES	m	20
Approximate inverse	$\texttt{nnz}_{\text{AI}}$	50
Approximate inverse	$\texttt{tol}_{\text{AI}}$	$\texttt{5.0e}-\texttt{2}$

We consider Finite Difference discretizations of Poisson's equation with Dirichlet boundary conditions ("fd2d" and "fd3d") on two dimentions (2D) and three dimensions (3D):

$$-\Delta \mathbf{u} - c\mathbf{u} = f \text{ in } \Omega,$$
$$\mathbf{u} = 0 \text{ in } \partial\Omega,$$

where $\Omega = (0,1)^2$ (2D) and $\Omega = (0,1)^3$ (3D), and $\partial\Omega$ denotes the boundary of the domain. We assume 5-point (2D) and 7-point (3D) centered discretizations on a regular mesh. In particular, for the 2D case we consider a 50×50 grid with $c = 0.1$, leading to a discretized Laplacian of size $n = 2500$. For the 3D case, we assume a $10 \times 10 \times 10$ grid and $c = 0.8$, leading to a discretized Laplacian of size $n = 1000$.

Figure 4 plots the relative residual norm curves for all the schemes considered, where each application of the preconditioned Richardson iteration performs $m_{\text{inner}} = 4$ inner iterations. Moreover, for PGMRES, we considered both approximate inverse and ILU(0) preconditioners. Every experiment is performed three times, each time with a different coarsening $p \in \{1, 2, 4\}$. A few remarks are as follows. First, the convergence of IGMRES is greatly impacted by restarting due to the inexact residual computation at the beginning of each cycle. Second, FGMRES without Richardson acceleration typically converges slower than PGMRES (recall Figure 1). Nonetheless, the former requires no digital FLOPS during the application of the preconditioner, and thus it is more appropriate to be compared against GMRES and IGMRES. In this context, observe that the convergence of FGMRES is not affected by restarting, i.e., FGMRES incorporates analog noise directly into its model. Finally, FGMRES + Ric leads to the fastest convergence overall, especially for larger p.

Figure 5 plots the relative residual norm achieved by FGMRES + Ric with $m_{\texttt{inner}} \in \{0, 1, 2, 3, 4\}$. Note that the choice "Ric(0)" corresponds to block-Jacobi preconditioning with no inner iteration, i.e., FGMRES. In summary, increasing the number of inner Richardson iterations in FGMRES enhances convergence, especially for smaller values of p.

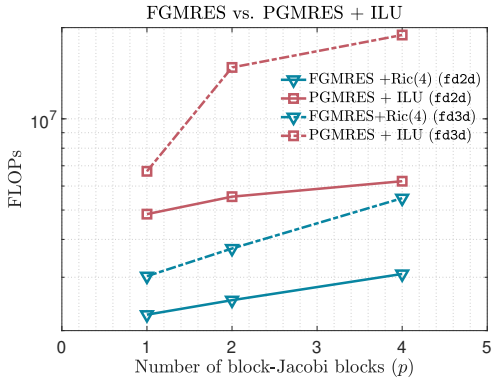


Fig. 6: *Number of FLOPS required by the FGMRES and PGMRES + ILU preconditioner for the matrix problems "fd2d" (solid) and "fd3d" (dashed).*

Figure 6 plots the number of FLOPS required by FGMRES + Ric with $m_{inner} = 4$ compared to PGMRES + ILU, for the matrix problems "fd2d" (solid) and "fd3d" (dashed). For FGMRES + Ric, the block-Jacobi approximate inverse was generated with $\texttt{nnz}_{\text{AI}} = 150$ and $\texttt{tol}_{\text{AI}} = \texttt{1.0e}-\texttt{2}$. For both problems, FGMRES + Ric required $2\times$ to $4\times$ fewer FLOPS than PGMRES + ILU. Moreover, FGMRES + Ric avoids the non-scalable triangular substitutions required by ILU.

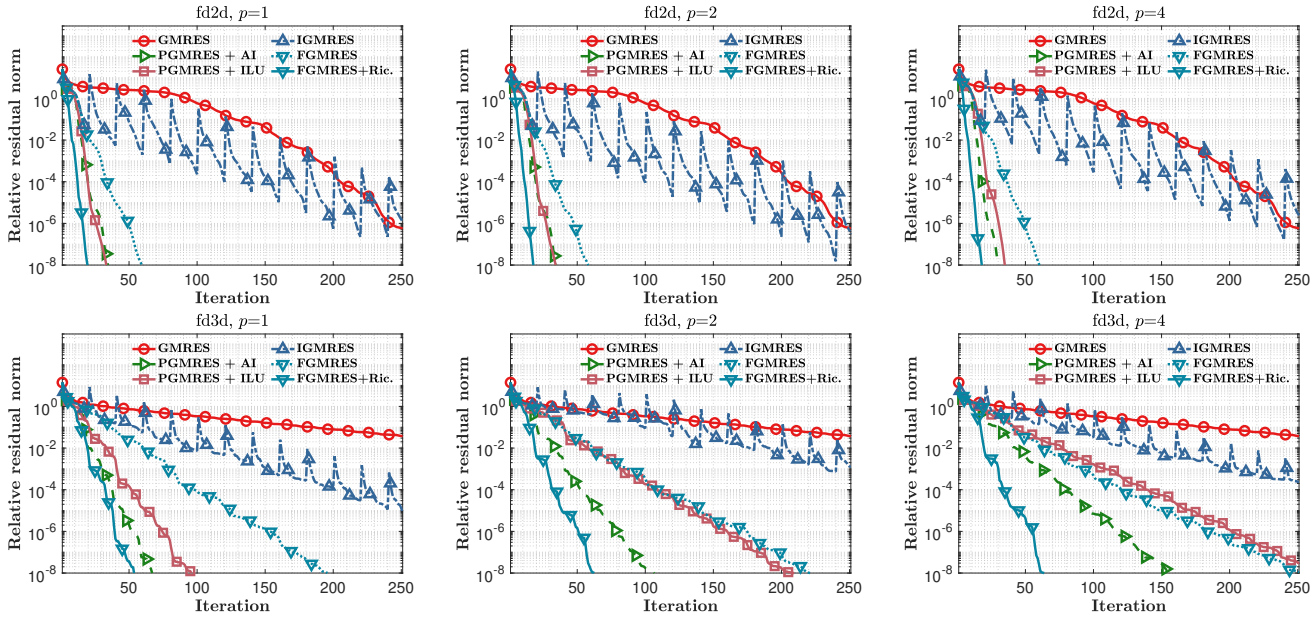


Fig. 4: *Convergence plots of various GMRES variants and block-Jacobi coarsenings.*

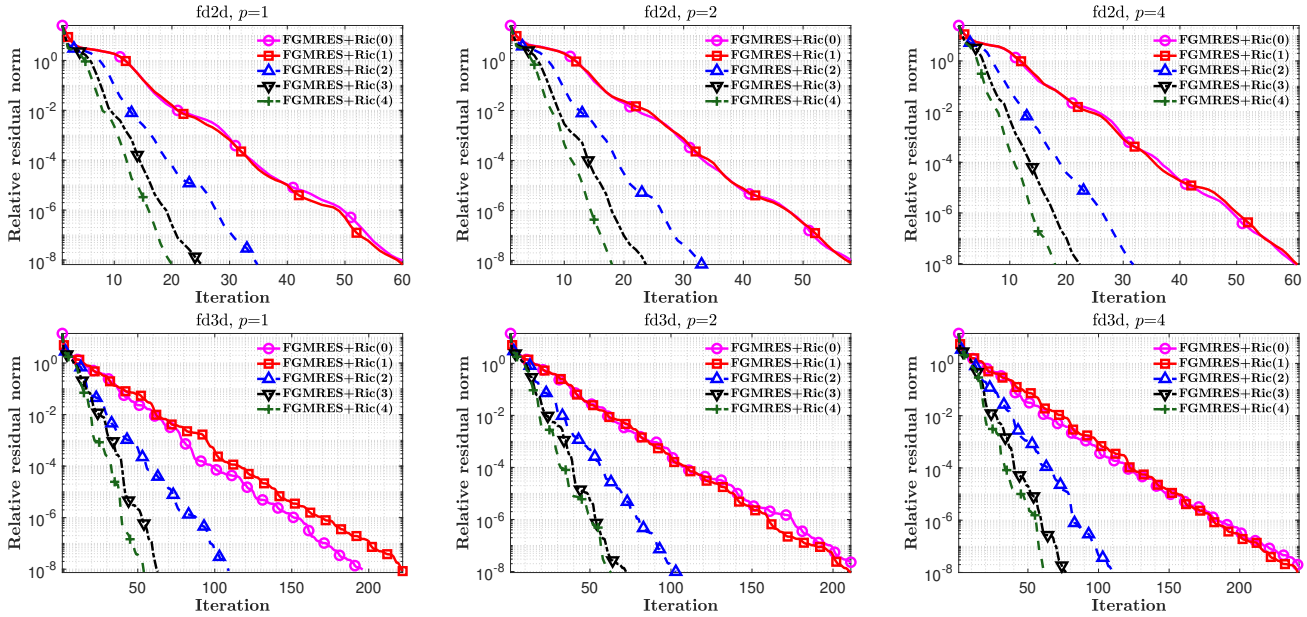


Fig. 5: *Convergence plots of FGMRES preconditioned by Richardson iteration for the matrix problems "fd2d" and "fd3d".*

V. Conclusion

This paper considered leveraging analog crossbar hardware as an in-memory accelerator to apply approximate inverse preconditioners to Krylov iterative methods. Analog crossbar hardware can perform MVMs extremely fast, leading to a rapid application of sparse approximate inverse preconditioners, irrespectively of their non-zero sparsity pattern. On the other hand, analog crossbar hardware introduces stochastic errors, and thus the application of sparse approximate inverses can be quite inaccurate. One option to remedy this is by using additional hardware to compensate for circuit non-idealities (see, e.g., (19)). Instead, the idea put forth in this paper is to try and remedy this issue at an algorithmic level by using flexible Krylov algorithms, in particular FGMRES, that naturally incorporate inexactness in their model. Numerical results with simulated hardware on two model problems verified the robustness of the proposed scheme. As part of future work, we plan to perform a detailed theoretical and experimental analysis of the behavior of FGMRES on general sparse problems using the IBM Analog Hardware Acceleration Kit (33), a (CUDA-capable) C++ simulator that allows for simulating a wide range of analog devices and crossbar configurations.

References

[1] G. Strang, *Computational Science and Engineering*. Wellesley-Cambridge Press, 2007.

[2] T. Xu, V. Kalantzis, R. Li, Y. Xi, G. Dillon, Y. Saad, "pargemslr: A parallel multilevel schur complement low-rank preconditioning and solution package for general sparse matrices," *Par. Comp.*, vol. 113, p. 102956, 2022.

[3] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[4] Y. Saad, M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comp.*, 7(3): 856–869, 1986.

[5] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[6] H. Anzt, Y. M. Tsai, A. Abdelfattah, T. Cojean, J. Dongarra, "Evaluating the performance of NVIDIA's A100 ampere GPU for sparse and batched computations," in *IEEE/ACM Perf. Model., Benchmark. Sim. High Perf. Comp. Sys. (PMBS)*, pp. 26–38, 2020.

[7] M. Ament, G. Knittel, D. Weiskopf, W. Strasser, "A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-gpu platform," in *Euromicro Conf. Par., Dist. and Net. Proc.*, pp. 583–592, 2010.

[8] M. M. Dehnavi, D. M. Fernandez, J.-L. Gaudiot, D. D. Giannacopoulos, "Parallel sparse approximate inverse preconditioning on graphic processing units," *IEEE TPDS*, 24(9): 1852–1862, 2012.

[9] G. Isotton, C. Janna, M. Bernaschi, "A GPU-accelerated adaptive fsai preconditioner for massively parallel simulations," *Int. J. HPCA*, p. 10943420211017188, 2021.

[10] F. Göbel, T. Grützmacher, T. Ribizel, H. Anzt, "Mixed precision incomplete and factorized sparse approximate inverse preconditioning on gpus," in *Euro. Conf. Par. Proc.*, pp. 550–564, 2021.

[11] M. Bernaschi, M. Carrozzo, A. Franceschini, C. Janna, "A dynamic pattern factored sparse approximate inverse preconditioner on graphics processing units," *SIAM J. Sci. Comp.*, 41(3): C139–C160, 2019.

[12] J. Gao, Q. Chen, G. He, "A thread-adaptive sparse approximate inverse preconditioning algorithm on multi-gpus," *Par. Comp.*, vol. 101, p. 102724, 2021.

[13] H. Anzt, J. Dongarra, G. Flegar, N. J. Higham, and E. S. Quintana-Ortí, "Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers," *Con. Comp.: Prac. Exp.*, 31(6): e4460, 2019.

[14] H. Anzt, M. Gates, J. Dongarra, M. Kreutzer, G. Wellein, M. Köhler, "Preconditioned Krylov solvers on GPUs," *Par. Comp.*, vol. 68, pp. 32–44, 2017.

[15] V. Kalantzis, A. Gupta, L. Horesh, T. Nowicki, M. S. Squillante, C. W. Wu, T. Gokmen, H. Avron, "Solving sparse linear systems with approximate inverse preconditioners on analog devices," *IEEE HPEC*, pp. 1–7, 2021.

[16] Q. Xia, J. J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature Mat.*, 18(4):309–323, 2019.

[17] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nano.*, 15(7): 529–544, 2020.

[18] S. Ambrogio and others, "An analog-AI chip for energy-efficient speech recognition and transcription," *Nature*, 620: 768-775, 2023.

[19] B. Feinberg, R. Wong, T. P. Xiao, C. H. Bennett, J. N. Rohan, E. G. Boman, M. J. Marinella, S. Agarwal, E. Ipek, "An analog preconditioner for solving linear systems," in *IEEE HPCA*, pp. 761–774, 2021.

[20] M. A. Zidan, Y. Jeong, J. Lee, B. Chen, S. Huang, M. J. Kushner, W. D. Lu, "A general memristor-based partial differential equation solver," *Nature Elec.*, 1(7): 411–420, 2018.

[21] M. Le Gallo, A. Sebastian, R. Mathis, M. Manica, H. Giefers, T. Tuma, C. Bekas, A. Curioni, E. Eleftheriou, "Mixed-precision in-memory computing," *Nature Elec.*, 1(4): 246–253, 2018.

[22] I. Richter, K. Pas, X. Guo, R. Patel, J. Liu, E. Ipek, E. G. Friedman, "Memristive accelerator for extreme scale linear solvers," *GOMACTech*, 2015.

[23] Y. Saad, "A flexible inner-outer preconditioned GMRES algorithm," *SIAM J. Sci. Comp.*, 14(2): 461–469, 1993.

[24] M. W. Benson, "Iterative solution of large scale linear systems," Ph.D. dissertation, 1973.

[25] M. J. Grote, T. Huckle, "Parallel preconditioning with sparse approximate inverses," *SIAM J. Sci. Comp.*, 18(3): 838–853, 1997.

[26] M. Benzi, M. Tuma, "A comparative study of sparse approximate inverse preconditioners," *Appl. Num. Math*, 30(2-3): 305–340, 1999.

[27] E. Chow, "A priori sparsity patterns for parallel sparse approximate inverse preconditioners," *SIAM J. Sci. Comp.*, 21(5): 1804–1822, 2000.

[28] ——, "Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns," *Int. J. High Perf. Comp. Appl.*, 15(1): 56–74, 2001.

[29] I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*. MIT press, 2016.

[30] L. Chua, "Memristor-the missing circuit element," *IEEE Trans. Circuit Theory*, 18(5): 507–519, 1971.

[31] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, E. Ipek, "Enabling scientific computing on memristive accelerators," *ISCA*, pp. 367–382, 2018.

[32] W. Haensch, T. Gokmen, and R. Puri, "The next generation of deep learning hardware: Analog computing," *Proc. IEEE*, vol. 107, pp. 108–122, 2019.

[33] M. J. Rasch, D. Moreda, T. Gokmen, M. L. Gallo, F. Carta, C. Goldberg, K. E. Maghraoui, A. Sebastian, and V. Narayanan, "A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays," *arXiv preprint arXiv:2104.02184*, 2021.

[34] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: Design considerations," *Frontiers Neuro.*, vol. 10, 2016.