Solving Sparse Linear Systems with Approximate Inverse Preconditioners on Analog Devices

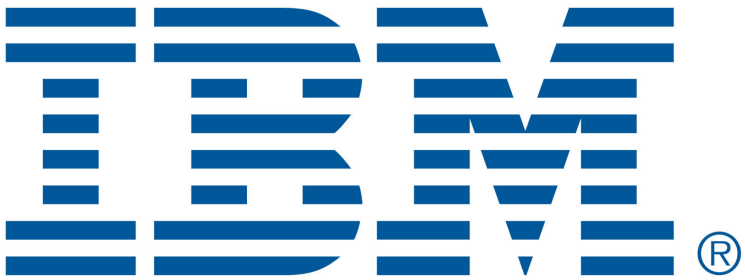
Vassilis Kalantzis, Anshul Gupta, Lior Horesh, Thomas Nowicki, Mark. S Squillante, Chai C. Wu, Tayfun Gokmen, and Haim Avron

November 2021

EPrint ID: 2021.4

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 with approximate inverse preconditioners on analog devices

Vasileios Kalantzis*, Anshul Gupta†, Lior Horesh†, Tomasz Nowicki†,
Mark S. Squillante†, Chai Wah Wu†, Tayfun Gokmen†, and Haim Avron‡
*,†IBM Research, Thomas J. Watson Research Center ‡Tel Aviv University
Email: *vkal@ibm.com, †{anshul,lhoresh,tnowicki,mss,cwwu,tgokmen}@us.ibm.com, ‡haimav@tauex.tau.ac.il

***Abstract*—Sparse linear system solvers are computationally expensive kernels that lie at the heart of numerous applications. This paper proposes a preconditioning framework that combines approximate inverses with stationary iterations to substantially reduce the time and energy requirements of this task by utilizing a hybrid architecture that combines conventional digital microprocessors with analog crossbar array accelerators. Our analysis and experiments with a simulator for analog hardware show that an order of magnitude speedup is readily attainable despite the noise in analog computations.**

***Index Terms*—Richardson iteration, approximate inverse preconditioners, analog crossbar arrays**

I. Introduction

The iterative solution of sparse linear systems [1] is a fundamental task across many applications in science, engineering, and optimization. For decades, progress in speeding up preconditioned iterative solvers was primarily driven by constructing more effective preconditioning algorithms and via steady microprocessor performance gains. The last decade saw the advent of fast power-efficient low-precision hardware, such as Graphics Processing Units (GPUs), which gave rise to new opportunities to accelerate sparse iterative solvers [2]–[8]. While these advances have led to remarkable performance gains, conventional CMOS-based digital microprocessors possess inherent scaling and power dissipation limitations. It is therefore imperative that we explore alternative hardware architectures to address current and future challenges of sparse iterative solvers. One such alternative paradigm is based on analog devices configured as crossbar arrays of non-volatile memory. These devices can achieve high degrees of parallelism 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. In particular, such devices can perform Matrix-Vector Multiplication (MVM) in near constant time independent of the number of nonzero entries in the operand matrices [9], [10]. Indeed, considerable computational speed-ups have been achieved with analog crossbar arrays [11]–[19], mostly for machine learning applications that can tolerate the noise and lower precision of these devices.

Analog designs with enhanced precision have been proposed [20], [21] to meet the accuracy demands of scientific applications, but the improvement in precision comes at the cost of increased hardware complexity and energy consumption. Memristive hardware with precision enhancement has been exploited in the context of Jacobi iterations to solve linear systems stemming from partial differential equations (PDEs) [22], and as an autonomous linear system solver [23] for small dense systems. The latter approach was used to apply overlapping block-Jacobi preconditioners in the Generalized Minimal RESidual (GMRES) iterative solver [20] with bit-slicing for increasing precision. Inner-outer iterations [24]–[26] have been considered 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.

The main contribution of this paper is a flexible preconditioning framework for inexpensively solving a large class of sparse linear systems by effectively utilizing simple analog crossbar arrays without precision-enhancing extensions. Our approach combines flexible iterative solvers and approximate inverse preconditioners on a hybrid architecture consisting of a conventional digital processor and an analog crossbar array accelerator. We analytically and experimentally show that this combination can solve certain sparse linear systems at a fraction of the cost of solving them on digital processors. Although the relative advantage of analog hardware grows with the size of the linear systems, we demonstrate that an order of magnitude speedup is achievable even for systems with just a few hundred equations.

While our method can be used to provide a fast stand-alone sparse solver, it also has exciting applications in the context of the upcoming exascale platforms [27]. Extreme-scale solvers are likely to be highly composable [28] and hierarchical [29] with multiple levels of nested algorithmic components. A solver like the one proposed in this paper, running on analog accelerator-equipped nodes of a massively parallel platform, is an ideal fast and low-power candidate for a local subdomain or coarse-grid solver to tackle the local components of a much larger sparse linear system.

II. Linear Algebra on Analog Hardware

We begin by briefly describing our architecture model and its key properties. Figure 1 illustrates a hybrid digital-analog architecture that combines a conventional microprocessor system with an analog crossbar array for performing MVM. The crossbar consists of rows and columns of conductors with a memristive element at each row-column intersection [30]. The conductance of these elements can be set, reset, or updated in a non-volatile manner. Matrix operations are performed by

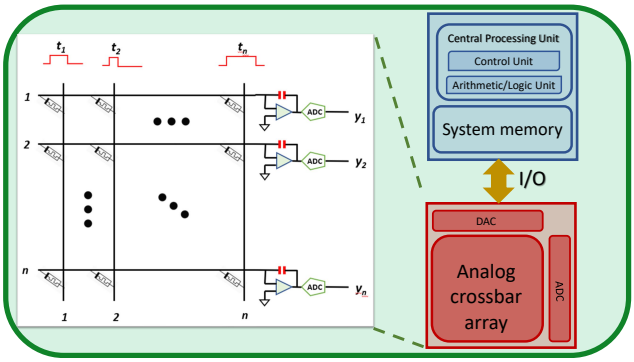


Fig. 1: *A hybrid digital-analog architecture consisting of a CPU and a memristive crossbar array for multiplying an $n \times n$ matrix M with a vector r.*

mapping the $n \times n$ operand matrix M onto the crossbar array, where the conductance G_{ij} at the intersection of row i and column j represents the value $M[i, j]$ (i.e., the (i, j)-th entry of the matrix M) after a suitable scaling.

The MVM operation $y = Mr$ can be emulated by sending pulses of V_{in} volts along the columns of the crossbar 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. Following 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 Kirchoff'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 $\widehat{y}[i]$ to $y[i]$.

The above procedure involves multiple sources of nondeterministic noise so that the output $\widehat{y} \in \mathbb{R}^n$ is equivalent to a low-precision approximation of y. Writing M to the crossbar array incurs multiplicative and additive write noises $N^{Wm} \in \mathbb{R}^{n\times n}$ and $N^{Wa} \in \mathbb{R}^{n\times n}$, respectively, and the actual conductance values at the crosspoints of the array reflect $\widehat{M} = M \odot (I + N^{Wm}) + N^{Wa}$, 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 noises $N^{Im} \in \mathbb{R}^n$ 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

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

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 inexactness 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. Equation (1) can be simplified as

$$\widehat{y} = (M + E)r + \widehat{N}^{Oa}, \tag{2}$$

where $\widehat{N}^{Oa}$ is the overall effective additive noise that captures all the lower order noise terms from Equation (1) and the nondetermistic error matrix E is given by

$$E = M \odot (N^{Wm} + \mathbf{1} \otimes N^{Im} + N^{Om} \otimes \mathbf{1}) + N^{Wa}; \tag{3}$$

here $\otimes$ denotes outer product. We provide additional technical details in [31].

The exact characteristics and magnitudes of the noises depend on the physical implementation [12], [25] of the analog array and the materials used therein, the details of which are beyond the scope of this paper but they are generally captured by our model. Regardless of the implementation, the noises are always device dependent and stochastic. 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 overcome the analog MVM error, even if it requires many more MVM operations than its digital counterpart, can substantially reduce the cost of the solution.

III. An Iterative Solver Based on Memristive Approximate Inverse Preconditioning

The goal of preconditioning is to transform an $n \times n$ system of linear equations $Ax = b$ to enable an iterative procedure to compute a sufficient approximation of x in fewer steps. Algebraically, preconditioning results in solving the system $MAx = Mb$, where $M \in \mathbb{R}^{n\times n}$ is an approximation of A^{-1}. Since A^{-1} is dense in general, most popular general-purpose preconditioners are applied implicitly; e.g., the ILU preconditioner generates a lower (upper) triangular matrix L (U) such that $A \approx LU$. The preconditioner $M = U^{-1}L^{-1}$ is then applied by forward/backward substitution [1].

A. Approximate inverse matrices as preconditioners

Although the inverse of a sparse matrix is dense in general, a significant fraction of the entries in the inverse matrix often have small magnitudes [32], [33]. Approximate inverse preconditioners [34]–[36] exploit this fact and seek to compute an approximation M to A^{-1} that can be applied via an MVM operation. Computing M can be framed as an optimization problem that seeks to minimize $||I - MA||_F$. One popular method for computing M is by the Sparse Approximate Inverse (SPAI) technique [36], which updates the sparsity pattern of M dynamically by repeatedly choosing new profitable indices for each column of M that lead to an improved approximation of A^{-1}. The quality of the approximation is controlled by computing the norm of the residual $I(:, j) - AM(:, j)$ after each column update. The jth column of M has at most $\texttt{nnz}_{\text{AI}}$ nonzero entries and satisfies $\|AM(:, j) - I(:, j)\|_F \leq \texttt{tol}_{\text{AI}}$, for a given threshold tolerance $\texttt{tol}_{\text{AI}} \in \mathbb{R}$.

An effective approximate inverse preconditioner M can still be considerably denser than the coefficient matrix A. Therefore, preconditioning must lead to a drastic convergence improvement for a reduction in the overall wall-clock time. Applying approximate inverse preconditioners as proposed in

this paper has the advantage that the computation time and energy consumption of preconditioning would be much lower on an analog crossbar array than on digital hardware, even at low precisions. Since MVM with M can be performed in near-constant time on an analog array regardless of the number of nonzeros in M, denser approximate inverses may be utilized without increasing the associated costs.

While analog crossbar arrays can speed up the application of the approximate inverse preconditioner, the accuracy of the application step is constrained by the various nondeterministic noises in the analog hardware discussed in Section II. This restricts the use of popular Krylov subspace approaches such as GMRES [37], which require a deterministic preconditioner. Although flexible variants [38] exist, in which the preconditioner can vary from one step to another, these variants generally require the orthonormalization of large subspaces and perform a relatively larger number of Floating Point Operations (FLOPs) outside the preconditioning step. Therefore, in this paper, we consider stationary iterative schemes.

B. Preconditioned Richardson iteration

Stationary methods approximate the solution of a linear system $Ax = b$ by iteratively applying an update of the form $x_i = Dx_{i-1} + d$, where $D \in \mathbb{R}^{n\times n}$ and $d \in \mathbb{R}^n$ are fixed. Preconditioned Richardson iteration can be defined as

$$\begin{aligned} \alpha MAx &= \alpha Mb, \qquad \alpha \in \mathbb{R}, \\ x &= x + \alpha Mb - \alpha MAx \\ &= (I - \alpha MA)x + \alpha Mb, \\ x_i &= (I - \alpha MA)x_{i-1} + \alpha Mb, \end{aligned}$$

which is identical to the form $x_i = Dx_{i-1} + d$, with $D = I - \alpha MA$ and $d = \alpha Mb$ [39]. The approximate solution x_i produced by preconditioned Richardson iteration during the ith iteration satisfies

$$\begin{aligned} x_i - x &= x_{i-1} - x - \alpha MAx_{i-1} + \alpha MAx \\ &= (I - \alpha MA)(x_{i-1} - x), \text{ and} \\ \|x_i - x\| &\leq (\|I - \alpha MA\|)^i \|x_0 - x\|, \end{aligned}$$

where the latter inequality holds for any vector norm and the corresponding induced matrix norm. A similar bound can also be shown for the residual vector $r_i = b - Ax_i$; i.e.,

$$\|r_i\| \leq (\|I - \alpha AM\|)^i \|r_0\|.$$

The above inequalities show that the sequence $\lim_{i\to\infty} x_i$ will converge to the true solution x as long as the preconditioner M and scaling scalar α are chosen such that $\|I-\alpha MA\| < 1$. In fact, the limit of the iterative process will converge to x as long as the spectral radius $\rho(I-\alpha MA)$ of the matrix $I-\alpha MA$ is strictly less than one. Nonetheless, we use the spectral norm since $\rho(I - \alpha MA) \leq \|I - \alpha MA\|$. Moreover, denoting the eigenvalues of the matrix MA by $|\lambda_1| \leq \ldots \leq |\lambda_n|$, we need to pick α such that $|1-\alpha\lambda_n| < 1$, i.e., we need $0 < \alpha < \frac{2\lambda_n^*}{|\lambda_n|^2}$. Unless mentioned otherwise, we will assume $\alpha = 1$.

The above analysis remains valid even in the case where M varies between iterations, which outlines the flexible nature of preconditioned Richardson iteration, and stationary iterative schemes in general.

C. Preconditioning through memristive hardware

Algorithm 1 summarizes our hybrid Richardson iterations preconditioned by applying an approximate inverse through analog crossbar hardware. The iterative procedure terminates when either the maximum number of iterations m_{it} is reached or the relative residual norm drops below a given threshold tolerance $t_{\text{ol}} \in \mathbb{R}$. Each iteration of Algorithm 1 requires a sparse MVM with the coefficient matrix A (Line 5), two "scalar alpha times x plus y" (AXPY) [40] operations (Lines 5 and 7), a vector norm computation (Line 6), and an MVM with the approximate inverse preconditioner M (Line 7) computed in analog hardware. Therefore, of the $3n+2(\texttt{nnz}(A)+\texttt{nnz}(M))$ total FLOPs per iteration, $2\texttt{nnz}(M)$ can be performed in $O(n)$ time on the analog hardware.

Algorithm 1 Richardson iterations with approximate inverse preconditioning on hybrid hardware

1: **input**: $A \in \mathbb{R}^{n\times n}$; $b, x_0 \in \mathbb{R}^n$; $t_{\text{ol}} \in \mathbb{R}$; $m_{\text{it}} \in \mathbb{N}$.
2: Construct approximate inverse preconditioner $M \in \mathbb{R}^{n\times n}$;
3: Write M to the analog crossbar array;
4: **for** $i = 0$ **to** $m_{\text{it}} - 1$ **do**
5: $\quad r_i = b - Ax_i$; // Digital MVM and AXPY
6: $\quad$ If ($\|r_i\| \leq t_{\text{ol}}\|b\|$) exit;
7: $\quad x_{i+1} = x_i + Mr_i$; // Analog MVM, digital AXPY
8: **end for**
9: **return** x_i;

In practice, the hybrid implementation is likely to require more iterations to reach the same residual norm as the digital implementation since the analog hardware leads to a less accurate application of the preconditioner M. In particular, excluding the time spent on constructing M and ADC-DAC conversions, the optimal computational speedup achieved by the hybrid implementation is bounded by

$$\mathcal{S}_{\text{tot}} = \frac{m_{\text{d}}}{m_{\text{h}}}\left(1 + \frac{2\texttt{nnz}(M)}{3n + 2\texttt{nnz}(A)}\right),$$

where m_{d} and m_{h} denote the number of digital and hybrid preconditioned Richardson iterations, respectively.

D. The effects of device noise

Since the MVM between the preconditioner M and r_i is computed through an analog crossbar array, the inherent inaccuracies of the analog device lead to the update

$$x_{i+1} = x_i + (M + E)r_i + \widehat{N}^{Oa},$$

where E and $\widehat{N}^{Oa}$ vary stochastically and nondeterministically between iterations. The error norm $\|x_{i+1} - x\|$ then satisfies the relationships

$$\|x_{i+1} - x\| \geq \|\widehat{N}^{Oa}\|$$

and

$$\|x_{i+1} - x\| \leq \|I - (M + E)A\|\|x_i - x\| + \|\widehat{N}^{Oa}\|.$$

In the following, we disregard conversion errors (i.e., $\widehat{N}^{Oa}$) since the dominant source of error is the inaccuracy during the copy of matrix M to the analog crossbar array. Then, the norm of the error $\|x_{i+1} - x\|$ will be smaller than $\|x_i - x\|$ as long as $\|I - (M + E)A\| < 1$, for which we have

$$\|I - (M + E)A\| \leq \|I - MA\| + \|E\|\|A\|.$$

Under the assumption that the spectral norm of the matrix $I - MA$ is less than one, i.e., $\|I - MA\| < 1$, a sufficient condition for the spectral norm of the matrix $I - (M + E)A$ to be less than one is then given by

$$\|E\| < \frac{1 - \|I - MA\|}{\|A\|}.$$

Additionally, under the condition $M = A^{-1} + \Delta$, we have $\|I-(M+E)A\| < 1$ as long as $\|E\| < \|A\|^{-1} - \|M - A^{-1}\|$.

Consistent with the original deterministic analysis for iterative refinement by Moler [41] from a digital perspective, but adapted to Richardson iteration with noisy analog devices, a sufficiently small $\|E\|$ in a stochastic sense guarantees the convergence of the preconditioned Richardson iteration process. In particular, we need to have $\|E\| < \frac{1-\|I-MA\|}{\|A\|}$ hold in a sufficiently large number of iterations. To understand and ensure when these conditions will hold, we establish explicit bounds for $\|E\|$ in terms of the statistical characteristics of the elements of the matrix E. We omit the details here and refer the interested reader to [31].

IV. Simulation Experiments

Our experiments were conducted in a Matlab environment (version R2020b) with 64-bit arithmetic on a single core of a 2.3 GHz 8-Core Intel i9 machine with 64 GB of system memory. We used a Matlab version of the publicly available simulator [42] 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 II 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 [16]. The number of bits used in the ADC and DAC was set to 7 and 9, respectively. Table I shows the default parameters used throughout our experiments to simulate the analog device and to construct the approximate inverse preconditioner.

TABLE I: *Default parameters.*

Module	Parameter	Value
Analog device	N^W	$5.0\text{e}-3$
Analog device	N^I	$1.0\text{e}-2$
Analog device	N^O	$1.0\text{e}-2$
Richardson it.	t_{ol}	$1.0\text{e}-5$
Richardson it.	m_{it}	50
Approximate inverse	$\texttt{nnz}_{\text{AI}}$	$40 \times \texttt{nnz}(A)$
Approximate inverse	$\texttt{tol}_{\text{AI}}$	$5.0\text{e}-2$

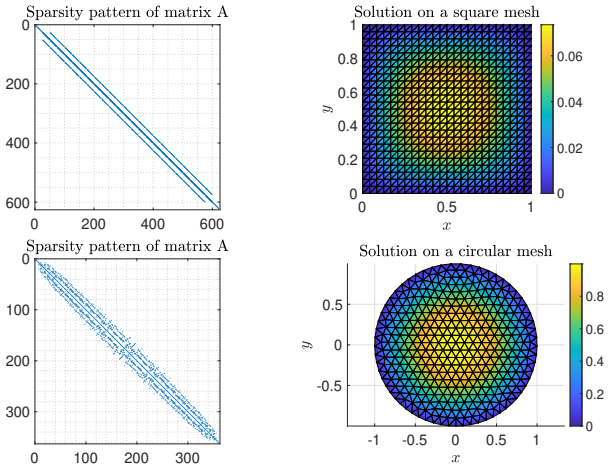


Fig. 2: *Left: Sparsity patterns of the discretized Laplacian for the square mesh* $\Omega := [0,1]^2$ *(top) and the circular mesh* $\Omega := \{x^2 + y^2 = 1\}$ *(bottom). Right: Surface plots of the solutions* $u(x, y)$.

Our test problems originate from discretizations of model PDEs. The first two matrices are derived from a Finite Element (FE) discretization of Poisson's equation on $\Omega \subset \mathbb{R}^2$ with homogeneous Dirichlet boundary conditions

$$-\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u(x, y) = f(x, y), \quad u|_{\partial\Omega} = 0, \qquad (4)$$

where ∂ denotes the partial derivative with respect to an independent variable and $f(x, y)$ denotes the load (or source) vector. We consider two different domains: a square domain $\Omega := [0, 1]^2$ and a circular domain $\Omega = \{[x, y] \in \mathbb{R}^2 \mid x^2 + y^2 = 1\}$ of radius one centered at the origin. The Laplace operator $\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u(x, y)$ is discretized by linear finite elements while the load vector is set equal to the constant source function $f(x, y) \equiv 1$. Figure 2 plots the sparsity patterns of the discretized Laplacian matrices A and the surfaces of the solutions $u(x, y)$ for the square ($n = 625$) and circular ($n = 362$) meshes. Our third test matrix stems from a Finite Difference (FD) discretization of the Laplace operator in the unit cube with homogeneous Dirichlet boundary conditions and $n = 8^3$. The average number of nonzero entries per row in the discretized sparse matrices A and in the corresponding preconditioners M are listed in Table II, which shows that the fill-in factor (i.e., the ratio $\text{nnz}(M)/\text{nnz}(A)$) ranges from 14 to 24. Our test matrices are relatively small because we are limited by the speed of the Matlab version of the simulator for the analog crossbar arrays; however, our results are meaningful because the relative advantage of analog hardware, in general, only increases with larger matrices.

In our experiments, we mainly compare the following iterative solvers: (a) Richardson iterations without preconditioning; (b) Richardson iterations with approximate inverse preconditioning applied in the IEEE 754 standard (denoted by "Richardson+d-ai"); and (c) Richardson iterations with approximate inverse preconditioning applied through simulated analog hardware (denoted by "Richardson+h-ai"). We observe from the results in Table II that standard Richardson iterations

TABLE II: *Comparison of Richardson iterations with digital (M_d) and hybrid (M_h) preconditioning; m, m_d and m_h denote the number of non-preconditioned, digital, and hybrid preconditioned Richardson iterations, respectively; κ denotes the condition number of A.*

Problem	n	κ	$\frac{\mathrm{nnz}(A)}{n}$	$\frac{\mathrm{nnz}(M)}{n}$	$\rho(I-A)$	$\rho(I-M_\mathrm{d}A)$	$\rho(I-M_\mathrm{h}A)$	m	m_d	m_h	FLOPs(M_d)	FLOPs(M_h)
FE Square	625	$3.4\mathrm{e}+2$	4.4	93.5	0.99	0.75	0.75	Failed	41	44	$5.0\mathrm{e}+6$	$3.1\mathrm{e}+5$
FE Circular	362	$2.0\mathrm{e}+2$	5.6	86.6	0.97	0.55	0.55	Failed	21	23	$1.4\mathrm{e}+6$	$1.1\mathrm{e}+5$
FD 3D	512	$6.2\mathrm{e}+1$	6.2	81.1	0.75	0.17	0.54	Failed	7	16	$6.3\mathrm{e}+5$	$1.2\mathrm{e}+5$

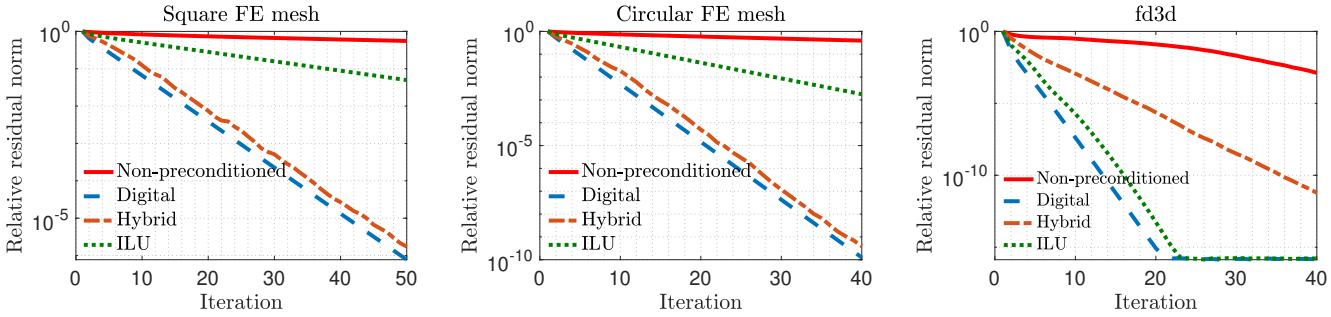


Fig. 3: *Relative residual norm achieved by Richardson iterations without and with digital/hybrid approximate inverse preconditioning.*

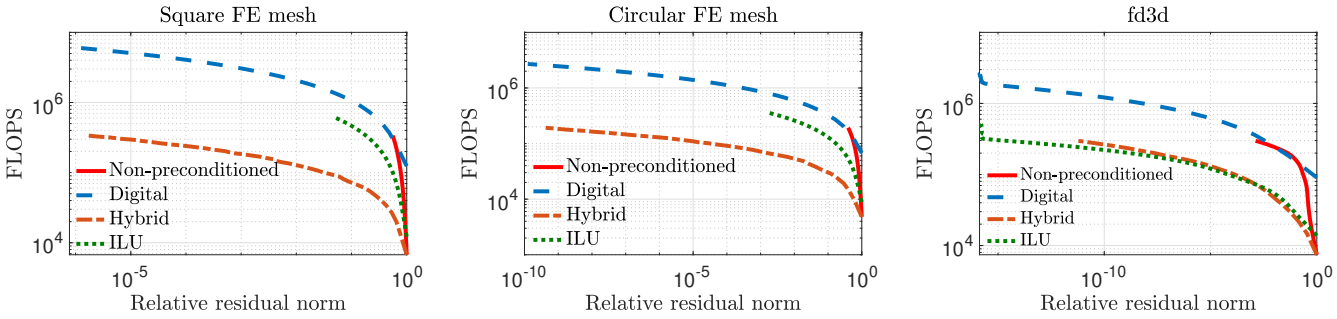


Fig. 4: *FLOPs as a function of the achieved relative residual norm for each of the three problems in Table II.*

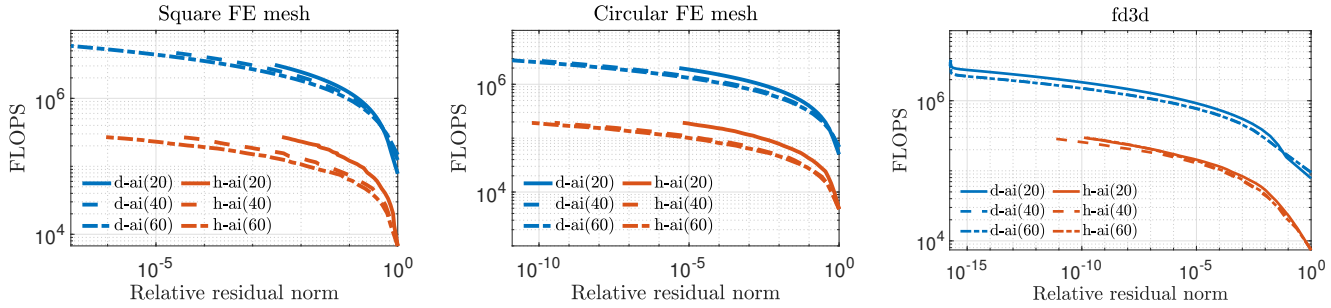


Fig. 5: *FLOPs required to reach a certain accuracy by standard and preconditioned Richardson iterations as the maximum number of nonzero entries in M varies according to* $\mathtt{nnz}_\mathrm{AI} = 20 \times \mathtt{nnz}(A)$, $40 \times \mathtt{nnz}(A)$, *and* $\mathtt{nnz}_\mathrm{AI} = 60 \times \mathtt{nnz}(A)$.

fail to converge within $m_\mathrm{it} = 50$ iterations for all three test problems, but converge rapidly with approximate inverse preconditioning. This is not surprising since the spectral radius of the matrix $I - A$, i.e., $\rho(I - A)$, is close to one in all cases. The convergence behavior of the two preconditioned variants is almost identical for the FE matrices, but hybrid preconditioned Richardson iteration requires more than twice the number iterations compared to the digital variant for the FD matrix. Overall, the hybrid preconditioned Richardson requires $5\times$ to $10\times$ fewer digital FLOPs compared to the purely digital version.

Figure 3 plots the relative residual norm after each Richardson iteration both without preconditioning and with the various preconditioners considered in this paper. As expected, hybrid preconditioning requires more iterations; however, this trade-off is highly beneficial because analog hardware allows for a rapid application of the preconditioner. This is evident in Figure 4 which shows that, on an average, hybrid preconditioning requires substantially fewer digital FLOPs to converge. For reference, we have also included Richardson iterations with an ILU(0) preconditioner [1] applied in the IEEE 754 standard.

Figure 5 plots the number of digital FLOPs versus the relative residual norm for $\mathtt{nnz}_\mathrm{AI}$ values of $20 \times \mathtt{nnz}(A)$, $40 \times \mathtt{nnz}(A)$, and $60 \times \mathtt{nnz}(A)$. A denser preconditioner

speeds up the convergence rate at the cost of additional FLOPs per iteration in the digital implementation. For the square FE mesh, the hybrid implementation yields a greater improvement in FLOPs compared to its digital counterpart as the preconditioner is made denser because the time to apply M on the analog device does not increase with nnz(M). However, this trend is not observed for the other matrices and diminishes with increasing preconditioner density because the analog noises limit the accuracy of the MVM operation Mr even if M is highly accurate. As a result, the number of iterations does not always decline as nnz(M) increases.

Figures 3–5 show that Richardson iterations preconditioned with approximate inverses on the hybrid architecture outperform preconditioned Richardson iterations on the digital architecture by greater margins as the condition number (listed in Table II) of A grows larger. Figure 5 also shows that our method can use denser preconditioners more effectively for matrices with higher condition numbers. These trends bode well for the potential use of our proposed preconditioning framework on a hybrid architecture in real-life problems that are likely to be larger and more ill-conditioned.

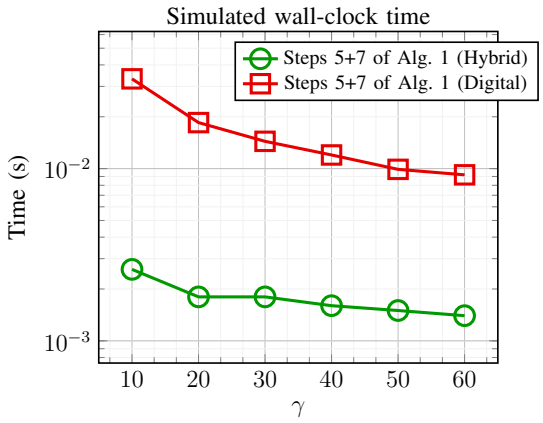


Fig. 6: *Simulated wall-clock time of various steps of Algorithm 1 as a function of the upper limit γ on the number of nonzeros per row.*

Figure 6 plots the simulated wall-clock time of the computations in the key steps of Algorithm 1 for the FE square matrix for various values of the fill-in factor γ on hypothetical digital and hybrid platforms based on the current state of hardware technology. The hybrid variant is significantly faster for all values of γ considered, and especially for smaller values when the preconditioner is less effective and convergence is slow. These times do not include the cost of data movement. For reference, the simulated cost of loading M on the analog crossbar array is about 3×10^{-3} seconds. This is roughly of the same order of magnitude as the cost of transferring a matrix of this size, which is only 625×625, between L3 cache and a typical CPU or a GPU. The current state of analog hardware technology is capable of realizing crossbar arrays of sizes up to 4000×4000 [16], which can be further combined to accommodate matrices of larges sizes that are even more favorable for the hybrid platform. Since M is written to the analog device only once, we expect the gap between the digital and hybrid variants to be higher in practice, and increase for larger and more ill-conditioned/slower-convergeing problems, especially when solving for multiple right-hand sides [43], [44]. Note that we omit the cost of constructing the preconditioner M here since, in practice, this step is relatively fast due the ample availability of parallelism [35], [36].

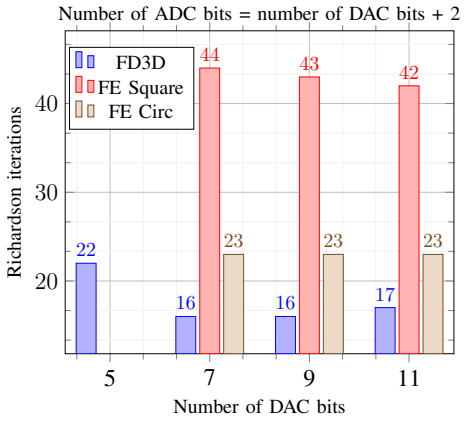


Fig. 7: *Number of iterations required by hybrid preconditioned Richardson as the number of DAC/ADC bits vary. For the 5-bit DAC, no convergence was obtained after 100 iterations for the FE matrices.*

Some hardware parameters, such as the number of DAC and ADC bits, can have a profound impact on the time and energy consumption of the analog device, and therefore on its overall performance. We vary the number of DAC and ADC bits and plot the results in Figure 7 to justify our choice of 7 DAC bits and 9 ADC bits. Fewer bits tend to lead to divergence, while more bits result in increased time and energy consumption with little or no improvement in convergence, as the accuracy of the analog computations is limited by other sources of noise.

V. Concluding Remarks

With the slowing of Moore's law [45] for digital microprocessors, there has been considerable interest in exploring alternatives, such as analog computing, for speeding up computationally expensive kernels. While simple analog crossbar arrays have been shown to be effective in many applications involving dense matrices, it has been challenging to address sparse matrix problems, such as solving sparse linear systems, because they do not naturally map to dense crossbar arrays and generally have a low tolerance for the stochastic errors pervasive in analog computing. This paper proposes, analyzes, and experimentally evaluates a preconditioning framework that exploits inexpensive MVM on analog arrays to substantially reduce the time and energy required for solving an important practical class of sparse linear systems. These and similar efforts are critical for meeting the continually growing computational demands of various applications of sparse solvers and for preparing these solvers for the fast but energy-constrained embedded and exascale [27] systems of the future.

Acknowledgments

We would like to thank Malte Rasch and Shashanka Ubaru for helpful discussions and input. that substantially contributed to the content of this paper.

REFERENCES

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

[2] D. Bertaccini and S. Filippone, "Sparse approximate inverse preconditioners on high performance GPU platforms," *Computers & Mathematics with Applications*, vol. 71, no. 3, pp. 693–711, 2016.

[3] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, M. Gates, T. Grützmacher, N. J. Higham, S. Li *et al.*, "A survey of numerical methods utilizing mixed precision arithmetic," *arXiv preprint arXiv:2007.06674*, 2020.

[4] K. L. Oo and A. Vogel, "Accelerating geometric multigrid preconditioning with half-precision arithmetic on GPUs," *arXiv preprint arXiv:2007.07539*, 2020.

[5] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, "Accelerating scientific computations with mixed precision algorithms," *Computer Physics Communications*, vol. 180, no. 12, pp. 2526–2533, 2009.

[6] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, "Harnessing GPU tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers," in *SC18: Int. Conf. for High Perf. Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 603–613.

[7] H. Anzt, M. Gates, J. Dongarra, M. Kreutzer, G. Wellein, and M. Köhler, "Preconditioned krylov solvers on GPUs," *Parallel Computing*, vol. 68, pp. 32–44, 2017.

[8] A. Haidar, H. Bayraktar, S. Tomov, J. Dongarra, and N. J. Higham, "Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems," *Proceedings of the Royal Society A*, vol. 476, no. 2243, p. 20200110, 2020.

[9] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[10] L. Xia, P. Gu, B. Li, T. Tang, X. Yin, W. Huangfu, S. Yu, Y. Cao, Y. Wang, and H. Yang, "Technological exploration of rram crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 3–19, 2016.

[11] A. Sebastian, M. L. G. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, pp. 529–544, 2020.

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

[13] M. J. Rasch, T. Gokmen, M. Rigotti, and W. Haensch, "RAPA-ConvNets: Modified convolutional networks for accelerated training on architectures with analog arrays," *Frontiers in Neuroscience*, vol. 13, p. 753, 2019.

[14] S. Ambrogio *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, pp. 60–67, 2018.

[15] A. Fumarola, P. Narayanan, L. L. Sanches, S. Sidler, J. Jang, and K. Moon, "Accelerating machine learning with non-volatile memory: exploring device and circuit tradeoffs," in *IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–8.

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

[17] G. W. Burr, R. M. Shelby, A. Sebastian, S. Kim, and S. Sidler, "Neuromorphic computing using non-volatile memory," *Advances in Physics*, vol. 2, pp. 89–124, 2016.

[18] M. N. Bojnordi and E. Ipek, "Memristive Boltzmann machine: A hardware accelerator for combined optimization and deep learning," in *Int. Symp. on High Performance Computer Architecture (HPCA)*, 2016.

[19] A. Shafice *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[20] B. Feinberg, R. Wong, T. P. Xiao, C. H. Bennett, J. N. Rohan, E. G. Boman, M. J. Marinella, S. Agarwal, and E. Ipek, "An analog preconditioner for solving linear systems," in *2021 IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*. IEEE, pp. 761–774.

[21] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *International Symposium on Computer Architecture (ISCA)*, 2018, pp. 367–382.

[22] M. A. Zidan, Y. Jeong, J. Lee, B. Chen, S. Huang, M. J. Kushner, and W. D. Lu, "A general memristor-based partial differential equation solver," *Nature Electronics*, vol. 1, no. 7, pp. 411–420, 2018.

[23] Z. Sun, G. Pedretti, E. Ambrosi, A. Bricalli, W. Wang, and D. Ielmini, "Solving matrix equations in one step with cross-point resistive arrays," *Proceedings of the National Academy of Sciences*, vol. 116, no. 10, pp. 4123–4128, 2019.

[24] I. Richter, K. Pas, X. Guo, R. Patel, J. Liu, E. Ipek, and E. G. Friedman, "Memristive accelerator for extreme scale linear solvers," in *Government Microcircuit Applications & Critical Technology Conference (GOMACTech)*, 2015.

[25] M. Le Gallo, A. Sebastian, R. Mathis, M. Manica, H. Giefers, T. Tuma, C. Bekas, A. Curioni, and E. Eleftheriou, "Mixed-precision in-memory computing," *Nature Electronics*, vol. 1, no. 4, pp. 246–253, 2018.

[26] Z. Zhu, A. B. Klein, G. Li, and S. Pang, "Fixed-point iterative linear inverse solver with extended precision," *arXiv preprint arXiv:2105.02106*, 2021.

[27] H. Antz *et al.*, "Preparing sparse solvers for exascale computing," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 20190053, 2020.

[28] J. Brown, M. G. Knepley, D. A. May, L. C. McInnes, and B. Smith, "Composable linear solvers for multiphysics," in *11th International Symposium on Parallel and Distributed Computing*, 2012, pp. 55–62.

[29] L. C. McInnes, B. Smith, H. Zhang, and R. T. Mills, "Hierarchical and nested Krylov methods for extreme-scale computing," *Parallel Computing*, vol. 40, no. 1, pp. 17–31, 2014.

[30] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[31] V. Kalantzis, A. Gupta, L. Horesh, T. Nowicki, M. S. Squillante, and C. W. Wu, "Solving sparse linear systems with approximate inverse preconditioners on analog devices," *arXiv preprint arXiv:2107.06973*, 2021.

[32] M. Benzi and G. H. Golub, "Bounds for the entries of matrix functions with applications to preconditioning," *BIT Numerical Mathematics*, vol. 39, no. 3, pp. 417–438, 1999.

[33] S. Demko, W. F. Moss, and P. W. Smith, "Decay rates for inverses of band matrices," *Mathematics of computation*, vol. 43, no. 168, pp. 491–499, 1984.

[34] M. Benzi and M. Tuma, "A comparative study of sparse approximate inverse preconditioners," *Applied Numerical Mathematics*, vol. 30, no. 2-3, pp. 305–340, 1999.

[35] E. Chow, "Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns," *The International Journal of High Performance Computing Applications*, vol. 15, no. 1, pp. 56–74, 2001.

[36] M. J. Grote and T. Huckle, "Parallel preconditioning with sparse approximate inverses," *SIAM Journal on Scientific Computing*, vol. 18, no. 3, pp. 838–853, 1997.

[37] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

[38] Y. Saad, "A flexible inner-outer preconditioned GMRES algorithm," *SIAM J. on Scientific Computing*, vol. 14, no. 2, pp. 461–469, 1993.

[39] L. F. Richardson, "IX. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 210, no. 459-470, pp. 307–357, 1911.

[40] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for Fortran usage," *ACM Transactions on Mathematical Software*, vol. 5, no. 3, pp. 308–323, 1979.

[41] C. Moler, "Iterative refinement in floating point," *Annals of the ACM*, vol. 14, no. 2, pp. 316–321, 1967.

[42] 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.

[43] V. Kalantzis, C. Bekas, A. Curioni, and E. Gallopoulos, "Accelerating data uncertainty quantification by solving linear systems with multiple right-hand sides," *Numer. Algorithms*, vol. 62, no. 4, pp. 637–653, 2013.

[44] V. Kalantzis *et al.*, "A scalable iterative dense linear system solver for multiple right-hand sides in data analytics," *Parallel Comput.*, vol. 74, pp. 136–153, 2018.

[45] J. Shalf, "The future of computing beyond Moore's law," *Phil. Trans. R. Soc. A.*, vol. 378, no. 20190061, 2020.