# Memristive Accelerator for Extreme Scale Linear Solvers

***Isaac Richter[1], Kamil Pas[1], Xiaochen Guo[1], Ravi Patel[1], Ji Liu[2], Engin Ipek[1,2], and Eby G. Friedman[1]***

[1]Department of Electrical and Computer Engineering [2]Department of Computer Science

University of Rochester
Rochester, New York, 14627, USA
Email Contact: ipek@cs.rochester.edu

**Abstract:** *Scientists model physical phenomena by means of computer simulations that typically require iteratively solving large systems of linear equations. We discuss a novel method of solving these systems by exploiting recently developed memristor technology. The proposed approach results in a $1500\times$ improvement in computational runtime, and an $8.5\times$ reduction in energy as compared with existing solvers.*

**Keywords:** computer architecture; supercomputing; VLSI circuits; architecture-circuit interactions.

## Introduction

Computer models of physical systems are a vital part of modern scientific and engineering research and development. Large scale computational models of the Earth's weather, climate, and geological activity; models of nuclear weapons; astronomical models of galaxies; and even macroeconomic models require immense computing resources. These simulations run on thousands of processors for several months at a time, utilizing tens or hundreds of millions of CPU hours before reaching a solution [1]. A 2010 report on exascale computing by the U.S. Department of Energy concludes that "computational modeling, simulation, prediction, and control at exascale offer the prospect of transformative progress in energy, national security, the environment, and our economy, and for fundamental scientific questions." [2] The same report finds that "making the transition to exascale poses numerous unavoidable scientific and technological challenges," while a 2008 report by the U.S. National Academy of Engineering identifies "engineering the tools of scientific discovery" [3] as one of 14 grand challenges in engineering for the 21st century. Complex physical models are often described in the form of systems of partial differential equations. The most common way to solve these systems is to discretize these expressions by transforming the continuous differential equations into discrete difference equations that serve as an approximation of the original system. The resulting difference equations are written as a large, sparse linear system, which is subsequently solved using iterative techniques [4] such as conjugate gradient, successive over-relaxation, or adaptive multi-grid. These iterative solvers first estimate the solution, and proceed to improve upon that estimate in a sequence of steps. Each successive step generates a closer approximation than the previous step, ultimately producing an answer sufficiently close to the exact solution. The process of modeling the physical world using numerical methods is illustrated in Figure 1.

*Key Idea:* To obtain the currents and voltages in a large resistive network, the network can be converted, using Kirchhoff's laws, into a system of linear equations. Given known resistor values and voltage or current inputs, solving the system provides the remaining unknowns. The reverse is also true: given a system of linear equations, it is possible to conceive of a resistor network that provides the solution to that linear system. Unfortunately, constructing physical networks with accurate programmable resistances requires substantial area. Utilizing memristors in a crosspoint array to solve a system of linear expressions is proposed here. Memristors offer the ability to construct a dense, continuously programmable, and reasonably accurate resistor network. The network produces a fast and accurate solution as an initial starting point for a (conventional) digital solver.

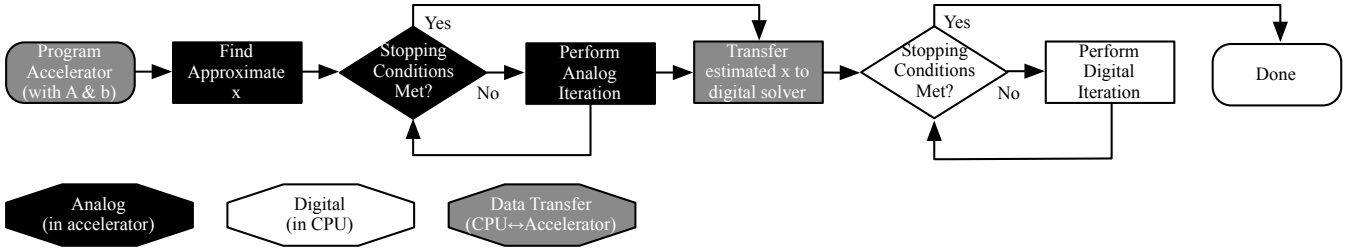


**Figure 1:** Flowchart for operating an accelerator alongside existing digital methods to solve linear systems. The steps performed by both the conventional and the proposed approaches are marked in gray. Steps exclusive to the approach proposed here are marked in black. In the figure, A, x, and b correspond, respectively, to the matrix and vectors in the equation $A\vec{x} = \vec{b}$.

The proximity of this initial estimate to the final answer has a profound effect on the number of iterations that are required to achieve convergence. As a result, using the output from the accelerator as an initial guess in an iterative solver generally results in convergence after significantly fewer iterations.

### System Overview

An analog accelerator for solving linear systems is proposed here. The accelerator is a discrete module that connects to an existing bus (e.g., DDRx or PCIe), and communicates with a conventional iterative solver running on a CPU. Results from the accelerator seed the iterative solver to reduce the number of iterations. The accelerator itself can also take the seed and run an iterative method in hardware using a memristor-based digital matrix-vector multiplication capability.

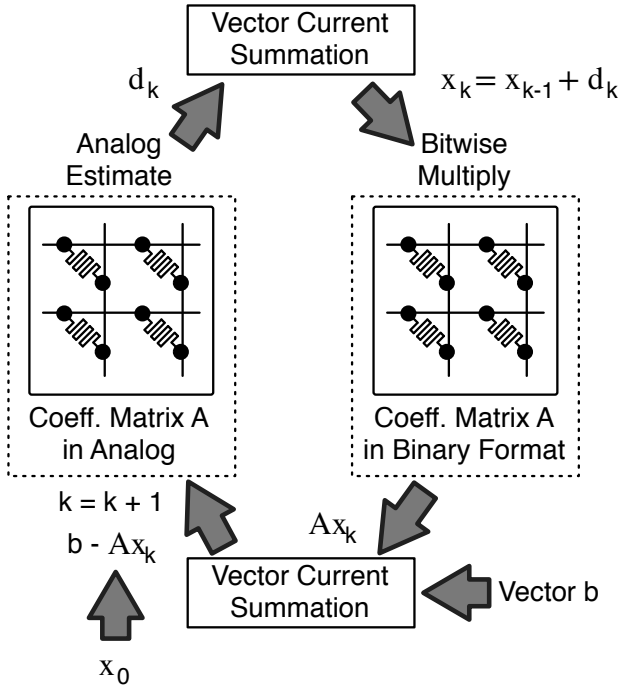


**Figure 2:** Overview of proposed accelerator.

The accelerator consists of a programmable resistive crosspoint array, as well as drivers, sensors, and control logic. The interface between the array and the rest of the system supports memory mapped DMA transfers to and from the accelerator. A control interface starts and interrupts calculations and determines status. Due to the slow speed of writes in a crosspoint array (443 MB/s [5]), incoming data are buffered and back pressure applied via the data bus to the DMA controller. The process in which the data flows between the accelerator and a conventional linear solver is illustrated in Figure 1.

*Integration with existing solvers:* Solvers for large, sparse linear systems typically rely on iterative methods. These iterative solvers generate exact results (for a given residual), but may require months of processing time on a modern supercomputer cluster [6]. By utilizing an analog solver, an approximate solution is produced in significantly less time and with substantially less energy. This approximate solution is used as an initial estimate in a conventional digital solver, thereby eliminating many iterations and reducing computational time.

The accelerator operates in two modes. In the *multiplication mode*, the accelerator multiplies matrix $A$ with vector $\vec{v}$ to obtain $A\vec{v}$. To achieve sufficiently high accuracy in the multiplication mode, the computations are performed digitally, where the memristors are used as digital switches. In the *analog estimation mode*, the accelerator estimates the vector $\vec{x}$ in $A\vec{x} = \vec{b}$. In this mode, the memristors are treated as analog devices that can be programmed over a continuous range, which for tantalum oxide devices is between 70 Ω and 4 kΩ [7].

*Operating Principles:* The solver initiates the computation by loading matrix $A$ and the right hand side (RHS) vector $\vec{b}$ into an accelerator. The accelerator operates in the estimation mode to return an approximate solution $\vec{x}_0$ close to the exact solution $\vec{x}^\star$, in which the inaccuracies in $\vec{x}_0$ are due to the quantization errors $\delta$ on $A$, source errors in $\vec{b}$, and sensing errors on $\vec{x}$. If $\vec{x}_0$ satisfies accuracy requirements, it is immediately returned. Otherwise, the estimate is refined on the accelerator and the improved version is returned. Either way, the solver reaches the stopping criterion earlier due to the initial estimate.

Among the sources of error that cause inaccuracies in the initial estimate, the quantization error $\delta$ is constant after programming the matrix $A$. In contrast, the source and sensing errors are due to thermal and voltage noise, and hence fluctuate. Two copies of the coefficient matrix $A$ are programmed into the memristive arrays: for the analog estimation mode, memristors are treated as analog storage, whereas the multiplication mode requires each bit of the coefficients to be programmed as a binary state. The accelerator iteratively refines the initial estimate $\vec{x}_0$ by leveraging both the analog estimation and the multiplication modes to compensate for quantization error $\delta$ in $A$, and to minimize the source and sensing errors, as shown in Algorithm 1. At every refinement iteration, the accelerator computes the residual of the previous iteration and estimates

**Algorithm 1** Analog iterative refinement ($A$ is the coefficient matrix, $\delta$ is the quantization error matrix of $A$, $\vec{b}$ is the right-hand side vector, and $\vec{x}_0$ is an initial estimate of the solution).

**function** INTERATIVEANALOGREFINE($A, \delta, \vec{b}, \vec{x}_0$)
  $k \leftarrow 0$
  **do**
    $k \leftarrow k + 1$
    $\vec{r}_{k-1} \leftarrow \vec{b} -$ BitwiseMultiply$(A, \vec{x}_{k-1})$
    $\vec{d}_k \leftarrow$ AnalogEstimate$(A + \delta, \vec{r}_{k-1})$
    $\vec{x}_k \leftarrow \vec{x}_{k-1} + \vec{d}_k$
  **while** $\|\vec{d}_k\| / \|\vec{x}_k\| >$ Threshold
  **return** $\vec{x}_k$

the solution of the linear system $(A + \delta)\vec{d_k} = \vec{r}_{k-1}$, where $\vec{d_k}$ compensates for the error on $\vec{x}_{k-1}$. The new estimate $\vec{x}_k$ is produced by adding $\vec{d_k}$ to $\vec{x}_{k-1}$. Note that in the analog estimation mode, the source and sensing errors are proportional, respectively, to the RHS vector $\vec{r}$ and the estimate $\vec{d}$, since the RHS vector is scaled to match the representation range of the current source.

*Data Transfer to/from the Accelerator:* The crosspoint array is initialized by communicating the contents of the matrix $A$ into the accelerator via memory mapped writes. Upon receipt of the data, the programming circuit sets the resistance of the crosspoint cells. To avoid the performance penalty of individually programming each cell, bulk-erase and bulk-load methods can be used to reduce programming time. Once the array is programmed, the accelerator is configured for either the multiplication or the estimation mode of operation. The voltage and current sources are programmed based on the contents of $\vec{x}$ and $\vec{b}$. Similar to the initial matrix, these data are sent to the accelerator via the memory mapped interface. Once programming is complete, the results of the resistive network are converted by a set of analog-to-digital converters (ADCs) into binary data. A control/status channel is used by the driver to poll for readiness, or to receive an interrupt when the output data are available for reading.

*Integrated Circuit Topology:* Due to circuit-level challenges when programming large memristive crosspoint arrays, the crosspoint arrays are limited to 1,000 rows by 1,000 columns [8]. Analog crossbar circuits combine multiple crosspoint arrays, analog drivers, and sensors (see Figure 3). This approach also leverages sparsity to block the matrices, requiring only the non-zero blocks to be programmed into the memristor arrays. Higher memory utilization is therefore achieved, making the system more likely to fit within a single accelerator.

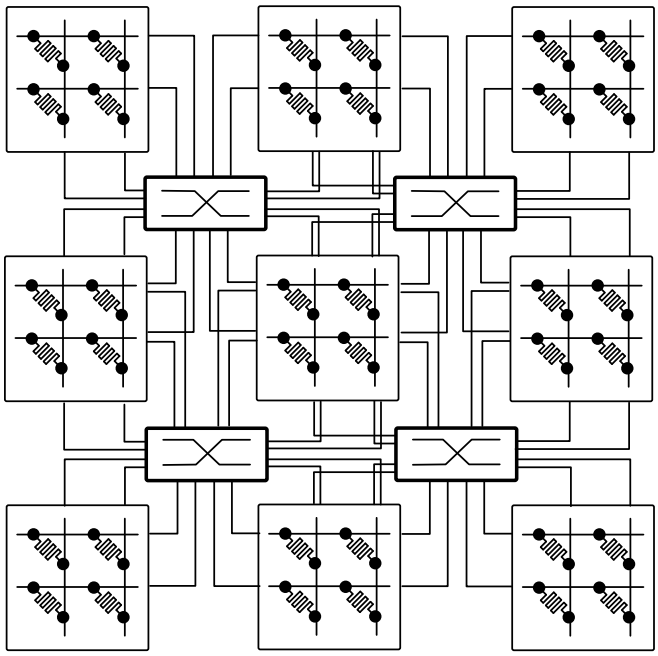

**Figure 3:** A memristive system containing multiple blocks communicating through programmable interconnect.

## Results

Initial results on the performance and energy characteristics of this proposed memristor-based linear accelerator are presented in this section.

*Circuit Performance:* A test case using $1{,}000 \times 1{,}000$ arrays has been simulated to determine the computational speedup. Assuming a GDDR5 memory bandwidth of 176 Gbps, a CMOS GPU requires approximately 786 $\mu$s for every iteration[1]. With the proposed approach, an analog array requires 6.2 $\mu$s per iteration—a greater than two orders of magnitude speedup. For a modern GPU accelerator with a 225 W power budget and a peak performance of 5 Tflops/s [9], the energy consumption to perform this computation is 622 mJ. Analog computation of the same application with the proposed accelerator array expends 73 mJ, 11.7% of the total energy of the GPU-based computation. The bulk of the energy savings is due to the reduced number of iterations in the analog array as compared to the number of iterations needed in a completely digital system. The delay for the computation is 55.4 $\mu$s—a $1559\times$ speedup in GPU solution time.

*Capacity:* Large scale scientific applications involve sparse matrices with hundreds of millions of rows and columns. Supercomputers with millions of cores map and solve problems of this magnitude. In contrast, the analog approach proposed here utilizes memristor crosspoint arrays that reduce system complexity to several integrated circuits. Consider, for example, a CPU on a compute node with a 150 Watt power budget. In the same power envelope, more than 16 arrays can be supported on a single die. Assuming a 22 nm CMOS technology and a die size of $5 \text{ mm} \times 5 \text{ mm}$, the area occupied by the crosspoint array is 0.07 $\text{mm}^2$, less than 0.3% of the die area, and capable of handling 32 million entries on a single integrated circuit. Assuming that the remaining 99.4% of the die area is dedicated to peripheral and interface circuitry, 32,000 circuits would be required to compute a problem with one hundred million rows and one hundred million columns. For comparison, a modern high performance supercomputer typically utilizes hundreds of thousands to millions of cores [2].

*Convergence:* The performance of the proposed hardware has been assessed against a random synthetically generated symmetric positive definite test matrix with 1,000 rows and

[1] Based on a Nvidia Tesla K10 limited by 320 GBps peak memory bandwidth using two channels

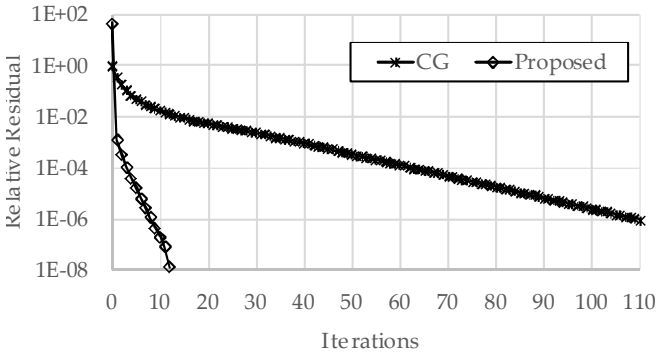


**Figure 4:** Residual reached at each iteration for synthetic matrices solved via conventional floating point CG and via analog iterative refinement

$62.8 \times 10^6$ non-zero entries[2]. For the purposes of this assessment, the analog hardware is assumed to have 13-bit digitization (in the ADCs and DACs). The drivers are assumed to exhibit 5% variance, and the sensors are assumed to exhibit a 20 $\mu$V RMS sensing noise for a 5 $\mu$s sensing time [10]. The memristors are programmed to within 1% of the requested value. Although 1% tolerance of the programmable resistors may seem high, recent results have shown controllability of memristive resistance to within a few percent [7]. This capability is expected to improve as memristor technology matures. For the digital bit-plane fixed-point operation used by the matrix-vector multiplication, 32-bit capability is assumed.

Results of the analog accelerator are compared to those results obtained by running the conjugate gradient method (CG) on conventional digital floating point hardware. CG requires one matrix vector multiplication per iteration. CG can be run on the accelerator using the fixed-point matrix-vector mode. Depending upon the input data, the memristor-accelerated CG converges to the same residual as a conventional floating point CG with 10% or fewer additional iterations.

Running CG on the accelerator, however, does not exploit the capability of the accelerator to generate solution vector estimates. The analog iterative refinement algorithm uses one matrix vector multiplication and one analog solution estimation per iteration. Analog iterative refinement is unsuitable for conventional hardware due to the time required to approximately solve a linear system during each iteration. On the accelerator, however, generating the approximate solution is fast. The iterations are therefore not prohibitively lengthy.

As shown in Figure 4, analog iterative refinement can reach a target residual with 14$\times$ fewer iterations than conventional CG. Given the parallel operation of the arrays, each iteration is performed rapidly. The test matrix only needs eight iterations of analog iterative refinement to reach the same residual that requires 110 iterations of CG. This example corresponds to 73.3 mJ and 55 $\mu$s for the proposed hardware as compared with 622 mJ and 86 ms on a GPU. This result is an improvement of 1500$\times$ in time, and 8.5$\times$ in energy.

[2]The test matrix is generated from $A = preA \times preA^T + I_m$, where $preA$ is a random sparse matrix with 100 non-zeros per row. The values for $preA$ are sampled from a standard normal distribution.

## Conclusions

Analog accelerators provide an interesting alternative to traditional digital math. A resistive crosspoint array can provide an approximate solution as a seed to digital iterative methods, resulting in as much as a 14$\times$ reduction in the number of iterations. This reduction in the number of iterations and the proposed architecture directly translates to an 8.5$\times$ energy savings and a 1500$\times$ reduction in runtime, supporting more detailed simulation experiments that can generate new insights into numerous scientific disciplines.

## Acknowledgments

This work was supported by NSF awards 1217418, 1054179, and 1329374. Xiaochen Guo is supported by an IBM fellowship.

## References

1. M. Boylan-Kolchin, "Cosmology: A Virtual Universe," *Nature*, Vol. 509, No. 7499, pp. 170–171, May 2014.
2. US Department of Energy, *The Opportunities and Challenges of Exascale Computing*, 2010.
3. National Academy of Engineering, *Grand Challenges for Engineering*, 2008.
4. Y. Saad, *Iterative Methods for Sparse Linear Systems: Second Edition*, Society for Industrial and Applied Mathematics, 2003.
5. A. Kawahara, *et al.*, "An 8Mb Multi-Layered Cross-Point ReRAM Macro with 443MB/s Write Throughput," *Proceedings of the International Solid-State Circuits Conference*, pp. 432–434, February 2012.
6. E. Gibney, "Model Universe recreates evolution of the cosmos," *Nature*, May 2014.
7. F. Miao, *et al.*, "Continuous Electrical Tuning of the Chemical Composition of $TaO_x$-Based Memristors," *ACS Nano*, Vol. 6, No. 3, pp. 2312–2318, February 2012.
8. J. Liang, S. Yeh, S. S. Wong, and H.-S. P. Wong, "Effect of Wordline/Bitline Scaling on the Performance, Energy Consumption, and Reliability of Cross-Point Memory Array," *ACM Journal on Emerging Technologies in Computing Systems*, Vol. 9, No. 1, pp. 1–14, February 2013.
9. Nvidia, *Tesla K10 GPU Accelerator Board Specification*, November 2012.
10. M. B. Leslie and R. J. Baker, "Noise-Shaping Sense Amplifier for MRAM Cross-Point Arrays," *IEEE Journal of Solid-State Circuits*, Vol. 41, No. 3, pp. 699–704, March 2006.