# **Acamar**: A Dynamically Reconfigurable Scientific Computing **A**ccelerator for Robust **C**onvergence **a**nd **M**inim**a**l **R**esource Utilization

Ubaid Bakhtiar, Helya Hosseini and Bahar Asgari
*University of Maryland-College Park*
Email: {ubaidb, helia, bahar}@umd.edu

***Abstract*—Although modern supercomputers are capable of delivering Exaflops now, they do not always achieve their peak performance. For instance, even today's high-end supercomputers achieve only less than 5% of their peak FLOPS when running HPCG, a benchmark designed to represent real-world scientific computing programs. To improve the efficiency of the key kernels in scientific computing, such as those used in solving partial differential equations, computer architects have begun to expand the applications of domain-specific architectures (DSAs) to scientific computing. However, DSAs that often have a *fixed* design are not likely to be practical solutions, as one specialized solution cannot fit all the *diverse* scientific computing workloads, making them less effective. The challenges of hardware inefficiency in today's supercomputers and the ineffectiveness of DSAs are further exacerbated by *sparsity*, a key characteristic of scientific computing workloads. While prior studies have proposed DSA solutions for sparse computations, they too are static and not adaptable to variations in the patterns and levels of sparsity across different scientific workloads. To address these challenges and target not only the diversity of computations in such workloads but also variations in sparsity, we propose Acamar, a dynamically reconfigurable accelerator. Acamar is adaptable to various solvers across different workloads and dynamically optimizes the trade-off between resource utilization and latency for sparse computations. The adaptable design also enables selecting a solver that guarantees convergence. We evaluate Acamar based on its Vitis HLS implementation on Xilinx Alveo u55c. Our experiments show a resource utilization and latency improvement up to 3.5× and 6× as well as improved performance efficiency and achieved throughput over a static design and Nvidia GTX 1650 Super.**

## I. Introduction

Scientific computing has long served as a catalyst for groundbreaking advancements in computer systems. The significance of scientific computing problems lies in their direct applicability to real-world scenarios. From simulating fluid dynamics and electromagnetics to tackling combinatorial challenges and aeronautical calculus, diverse domains have fueled the need for high-speed computational capabilities. For years, supercomputers equipped with CPUs and more recently GPUs have been used for running scientific computing workloads. While the peak performance of modern supercomputers have now reached to Exaflops, they still cannot utilize their full capability when they run scientific workloads. For instance, Fugaku and Frontier, the two top-ranked supercomputers based on High Performance Conjugate Gradients (HPCG) [21], [69] ranking in 2023, achieve 3.6% and 1.1% of their peak performance when running this benchmark. These numbers suggest that the extreme inefficient utilization of the potential of computing resources is more alarming in the post Moore's Law era – when translating technology scaling to performance scaling is more challenging than before.

During the past decade, after witnessing the success of domain-specific architectures (DSAs) for machine learning and artificial intelligence, computer architects have expanded the reach of DSAs to scientific computing by proposing various techniques and utilizing a range of technologies such as analog computing [16], [20], [24], processing-in-memory [8], [9], [65], wafer-scale processors [29], [52], and FPGAs [10], [14], [28], [70]. Before undertaking the development of optimized computing solutions, understanding the intricacies of scientific phenomena and their numerous parameters is essential. As mathematics offers means of simplifying complex scientific problems into more manageable algebraic forms, computer architects have primarily concentrated on accelerating and optimizing these relatively simplified representations of the problem, leveraging the power of mathematics to streamline their approach. However, sparse matrices of various sizes are inherent in these mathematical simplifications. For instance, partial differential equations (PDEs) employed in engineering applications undergo discretization through techniques such as finite element method (FEM) and finite difference method (FDM). Discretization transforms the original problem into a standard linear algebra form represented as $Ax = b$, where $A$ is a highly sparse coefficient matrix, $b$ is a constant vector, and $x$ is the sought-after solution. Similar to PDEs, numerous other scientific computing problems can also be transformed into the $Ax = b$ representation, making this simplified algebraic form a cornerstone of accelerated scientific computing research.

To solve $Ax = b$ several contributions have been made by both the mathematicians [11], [13], [54]–[58], [66], [74] and computer scientists, wherein the main role of the former is to develop sophisticated yet accurate mathematical algorithms (a.k.a. solvers) and the latter is responsible for their accelerated and efficient implementation on computer systems. The primary kernel in these solvers is sparse matrix-vector multiplication (SpMV). General-purpose hardware such as CPU and GPU provides solutions to $Ax = b$ problems, but they often lack efficiency in terms of resource utilization.

The suboptimal performance is primarily attributed to the sparse and unpredictable structure of the coefficient matrix $A$ resulting in poor performance of the SpMV kernel. To address these challenges, various application-specific architectures have been proposed [9], [20], [38], [47], [65]. These architectures aim to efficiently handle sparse coefficient matrices, leading to accelerated sparse computations (i.e. SpMV) and, as a result, faster implementation of various algorithms. However, we have identified two major limitations in state-of-the-art accelerators: **(i)** lack of generality **(ii)** static design. These accelerators operate under the assumption that any $Ax = b$ problem can converge to a solution using a specific solver regardless of the structural properties of the coefficient matrix $A$. Practically, matrix A needs to have a specific structure for it to converge to a solution using the underlying solver. Additionally, these accelerators employ static accelerator designs that cannot adapt based on the specific structure of matrix $A$, resulting in inefficient resource utilization in the SpMV kernel.

These challenges necessitate an efficient hardware solution capable of achieving solution convergence for any structure of coefficient matrix A while dynamically adapting to maximize resource utilization. To deal with these challenges, *we propose Acamar*[1]*, a dynamically reconfigurable FPGA-based design that has the ability to not only reconfigure the FPGA fabric to different solvers but also reconfigure the sparse computational unit based on the structural properties of coefficient matrix A*. To achieve this, Acamar can seamlessly switch between three widely used iterative method solvers: Jacobi iterative method (JB), conjugate gradient (CG), and bi-conjugate gradient-stabilized (BiCG-STAB). Moreover, Acamar can reconfigure the sparse computational unit, to ensure enhanced resource utilization. Our accelerator also offers a multi-level iterative decision chain to minimize the reconfiguration rate so as to incur less reconfiguration overhead. Acamar is an FPGA-based design owing to the ability of FPGAs to offer partial dynamic reconfiguration. We have modeled Acamar based on its Vitis HLS implementation on Xilinx Alveo u55c. Our experiments have shown an improvement in resource utilization over a static design and Nvidia GTX 1650 Super GPU.

To the best of our knowledge this is the first work to leverage the partial dynamic reconfiguration ability of FPGAs for the acceleration of scientific problems. In summary, this paper contributes the following:

- **Robust Convergence.** We show that a solver might not converge to a solution based on the properties of the coefficient matrix $A$, necessitating the need for a robust convergence accelerator.
- **Dynamic Fine-Grained Reconfiguration.** We propose a reconfigurable accelerator that has the ability to switch between solvers as well as optimize the sparse computations by fine-grained reconfiguration of the SpMV unit based on the structure of the coefficient matrix.
- **Optimization.** To optimize the reconfiguration rate of SpMV unit and reduce the reconfiguration overhead, we propose an inexpensive yet effective multi-stage iterative decision chain.

[1]Acamar /'ækəmaːr/ is a binary star system in the constellation of Eridanus.

## II. System of Linear Equations

This section first introduces the use cases of the systems of linear equations, $Ax = b$, in scientific computing, and then reviews the various solvers used in these applications, summarizing their key features, major computations and main drawbacks in their hardware deployment.

### A. Representing Problems in Ax=b

Scientific computing problems are typically simplified into a concise $Ax = b$ representation, where $b$ represents a constant vector and $x$ denotes the sought-after solution vector. The pivotal component of this equation is the coefficient matrix $A$, which commonly exhibits high sparsity and encapsulates the parameters of the underlying scientific application. Several streams of scientific computing problems can be mapped in the $Ax = b$ form. Here we touch upon some of them:

- **PDE Solvers.** Majority of the engineering problems boil down to multi-dimensional equations with multiple unknown variables and their partial derivatives. These equations are called PDEs. PDEs provide a mathematical framework for understanding phenomena such as heat transfer, fluid dynamics, electromagnetism, and wave propagation. They are used to model the behavior of structural components, electrical circuits, and power systems. Owing to their manifestation in diverse fields, a notable volume of research work exist on their efficient solutions [35], [63], [66]. One of the most widely used methods to solve PDEs is their reduction to $Ax = b$ form via discretization of the independent and dependent variable domains with a certain grid size.
- **Optimization Problems.** These problems play a crucial role across various disciplines because of their ability to efficiently allocate resources, minimize costs, maximize efficiency, and find optimal solutions to complex decision-making problems. Some optimization problems include linear programming, network flow optimization, and linear regression. They optimize a linear objective function subject to linear equality and/or inequality constraints. The constraints can often represented in terms of $Ax = b$ form, where $x$ is the vector of decision variables.
- **Graph Theory.** Graphs are data structures used to represent relationship between objects or entities in various applications. One of the classical applications of graph theory is the place and route problem of electrical circuits. Several aspects of graph theory can be encapsulated in the $Ax = b$ form. For example, the spanning tree problems can be modeled as $Ax = b$ problems to represent the constraints imposed by the spanning tree. In spectral graph theory, properties of the Laplacian matrix are studied to understand the behavior of the graph. The eigenvalues and eigenvectors of the Laplacian matrix can be used to

solve $Ax = b$ systems, where $A$ is the Laplacian matrix and $b$ is a vector representing graph constraints.

### B. Solving Ax=b

Now that the importance of $Ax = b$ representation is established, this section discusses the methods to solve it. There are two primary categories of $Ax = b$ solvers:

**Direct Methods.** To solve the equation $Ax = b$, these methods rely on decomposing the coefficient matrix $A$ into simpler representations to find the solution vector $x$. These methods guarantee an exact solution but can be computationally expensive, especially for large and sparse matrices.

Some common direct methods include LU decomposition, Cholesky decomposition, and QR decomposition. These methods decompose $A$ into factors such as lower triangular matrices, upper triangular matrices, or orthogonal matrices, which can then be used to efficiently solve the system of equations.

However, because of their computational cost, direct methods are not always practical for scientific applications that involve large and sparse matrices.

**Iterative Methods.** These methods start with an initial guess of $x$ and iteratively update the solution vector until a convergence criterion is met. Rather than pursuing an exact solution, these methods rely on finding the approximate solution within a certain threshold by changing the search direction every iteration. Iterative methods exhibit favorable convergence properties, often outperforming direct methods in terms of computational efficiency and memory usage. Some frequently-used iterative solvers include the Jacobi iterative method (JB) and the Gauss-Seidel iterative method, which are relatively simple yet effective. Additionally, there are more sophisticated iterative solvers based on Krylov subspace methods, such as conjugate gradient (CG), bi-conjugate gradient stabilized (BiCG-STAB), and general methods of residuals (GMRES).

**Algorithm 1** Jacobi Iterative Method (JB)

given $b := resultVector$
initialize $x_0 := 0$
$L := lowerTriangular(A)$
$U := UpperTriangular(A)$
$D^{-1} := inverseDiagonal(A)$
$T := D^{-1}(L + U)$
$c := D^{-1}.b$
**for** $j = 0, 1, \ldots,$ until convergence **do**
    $x_{j+1} := c - T.x_j$
**end for**

**Algorithm 2** Conjugate Gradient (CG)

given $b := resultVector$
Compute $r_0 := b - Ax_0, p_0 := r_0$
**for** $j = 0, 1, \ldots,$ until convergence **do**
    $\alpha_j := (r_j, r_j)/(Ap_j, p_j)$
    $x_{j+1} := x_j + \alpha_j p_j$
    $r_{j+1} := r_j - \alpha_j A p_j$
    $\beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$
    $p_{j+1} := r_{j+1} + \beta_j p_j$
**end for**

**Algorithm 3** Bi-Conjugate Gradient-Stabilized (BiCG-STAB)

given $b := resultVector$
Compute $r_0 := b - Ax_0$; $r_0^*$ arbitrary;
$p_0 := r_0$.
**for** $j = 0, 1, \ldots,$ until convergence **do**
    $\alpha_j := \left(\frac{r_j, r_0^*}{Ap_j, r_0^*}\right)$
    $s_j := r_j - \alpha_j A p_j$
    $\omega_j := \left(\frac{As_j, s_j}{As_j, As_j}\right)$
    $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
    $r_{j+1} := s_j - \omega_j A s_j$
    $\beta_j := \left(\frac{r_{j+1}, r_0^*}{r_j, r_0^*}\right) \times \frac{\alpha_j}{\omega_j}$
    $p_{j+1} := r_{j+1} + \beta_j (p_j - \omega_j A p_j)$
**end for**

In this paper, our focus is exclusively on iterative methods, particularly the JB, CG, and BiCG-STAB, the algorithms of which are shown respectively in Algorithm 1, 2, and 3. JB Method (Algorithm 1) is a straightforward iterative approach that discretizes a PDE into a linear system and then solves it. It updates each variable independently based on the current estimates of the other variables, using a process of iteration until convergence. Since the JB method updates each variable independently, it is simple but can be slow to converge, particularly for large systems. Similarly, CG (Algorithm 2) also begins by discretizing the PDE into a linear system, specifically targeting symmetric and positive-definite matrices. Unlike the JB method, CG uses conjugate directions to accelerate convergence. This method is more sophisticated and generally converges faster than JB by effectively reducing the error along successive orthogonal directions. Finally, BiCG-STAB (Algorithm 3) extends the CG method to handle non-symmetric linear systems, which are common in more complex scientific problems. It combines the approaches of Bi-Conjugate Gradient and additional stabilization techniques to enhance convergence. This method seeks to retain the fast convergence properties of CG while addressing the oscillatory convergence issues that may arise in non-symmetric systems, unlike the more stable but slower JB method.

## III. Challenges & Motivation

The combination of our three selected iterative solvers, JB, CG, and BiCG-STAB, collectively offers a robust solution space that addresses a vast majority of situations encountered

**TABLE I:** Structural requirements on coefficient matrix $A$ for convergence.

| Solver | Convergence Criteria | Solver | Convergence Criteria |
|---|---|---|---|
| Jacobi | Strictly Diagonally Dominant | Gauss-Seidel | Strictly Diagonally Dominant |
| Successive Over Relaxation | Symmetric, Positive Definite | CG | Symmetric, Positive Definite |
| Preconditioned CG | Negative Definite | Conjugate Residual | Hermition |
| BiCG | Non-symmetric | BiCG-Stablized | Non-symmetric |
| Two Sided Lanczos | Non-symmetric | General Method of Residual | Symmetric and Non-symmetric, Positive Definite |
| Concus, Golub and Widlund | Nearly symmetric, Positive Definite | | |

**TABLE II:** Examples of solvers diverging (×) and converging (✓) for coefficient matrices from SuiteSparse collection [12].

| ID | Dataset | DIM | Sparsity% | JB | CG | BiCG STAB | Acamar |
|---|---|---|---|---|---|---|---|
| 2C | 2cubes_sphere | 101K | 0.016 | × | ✓ | ✓ | ✓ |
| Of | offshore | 259K | 0.0063 | × | ✓ | ✓ | ✓ |
| Wi | windtunnel_evap3d | 40K | 0.1426 | ✓ | × | ✓ | ✓ |
| If | ifiss_mat | 96K | 0.0388 | × | × | ✓ | ✓ |
| Wa | wang3 | 177K | $8.3\times10^{-5}$ | ✓ | ✓ | ✓ | ✓ |
| Fe | fe_rotor | 99K | $5.6\times10^{-6}$ | ✓ | × | × | ✓ |
| Eb | epb3 | 84K | 0.0065 | ✓ | × | ✓ | ✓ |
| Qa | qa8fm | 66K | 0.038 | × | ✓ | ✓ | ✓ |
| Th | thermomech_TC | 711K | 0.0068 | × | ✓ | ✓ | ✓ |
| Bc | bcircuit | 375K | $4.8\times10^{-5}$ | × | ✓ | × | ✓ |
| Sd | sd2010 | 88K | $5.2\times10^{-5}$ | ✓ | × | × | ✓ |
| Li | light_in_tissue | 29K | 0.0474 | ✓ | ✓ | ✓ | ✓ |
| Po | poisson3Db | 85K | 0.032 | ✓ | ✓ | ✓ | ✓ |
| Cr | crystm03 | 583K | 0.0957 | × | ✓ | ✓ | ✓ |
| At | atmosmodm | 1.4M | 0.0005 | ✓ | ✓ | ✓ | ✓ |
| Mo | mono_500Hz | 169K | 0.0175 | ✓ | ✓ | ✓ | ✓ |
| Ct | cti | 16K | $1.8\times10^{-4}$ | ✓ | × | × | ✓ |
| Ns | ns3Da | 1.67M | $7.2\times10^{-7}$ | × | × | ✓ | ✓ |
| Fi | finan512 | 74K | 0.0107 | ✓ | ✓ | ✓ | ✓ |
| G2 | G2_circuit | 150K | $2.8\times10^{-5}$ | ✓ | ✓ | ✓ | ✓ |
| Ga | GaAsH6 | 3.3M | $5.3\times10^{-8}$ | × | ✓ | ✓ | ✓ |
| Si | Si343H6 | 5.1M | 0.016 | × | ✓ | ✓ | ✓ |
| To | torso2 | 1M | $1.1\times10^{-5}$ | ✓ | ✓ | ✓ | ✓ |
| Ci | cit-HepPh | 27K | $1.9\times10^{-5}$ | ✓ | × | × | ✓ |
| Tf | Trefethen_20000 | 20K | 0.0014 | × | ✓ | ✓ | ✓ |

in scientific computing, such as large coefficient matrices, random sparsity patterns, and non-uniform numerical patterns. However, the fact that covering such a diverse range of scientific computing problems do indeed rely on various solutions, challenges the optimizations for efficiency.

Additionally, while these solvers are accurate, their applicability and performance vary based on the structural properties of the *sparse* coefficient matrix A. Such variations make today's scientific hardware accelerators less effective and efficient, highlighting the need for a dynamically reconfigurable accelerator that can adapt to diverse requirements of various inputs without sacrificing efficiency. To clarify these key points, this section first briefly reviews the recent advancements in hardware specialization solutions for solving $Ax = b$, their limitations, and then delves deeply into the specific challenges targeted by this paper.

### A. Hardware Specialization & Their Limitations

Several specialized hardware accelerators have been proposed to optimize the solution of $Ax = b$ problems. For instance, in [16], the system of linear equations has been solved using BiCG-STAB solver on memristive accelerators. However, this work focuses on near-memory computations and termination of higher bits of floating-point computations. Alrescha [5] proposes a hardware-software co-design approach to accelerate the sparse computations in the HPCG benchmark by mitigating the data dependencies but it focuses on CG solvers only. FDMAX [38] is a recent accelerator that proposes reconfigurable grid sizes while solving the PDEs using Jacobi/Hybrid iterative solver. However, FDMAX fails to discuss the matrix form of Jacobi iterative method and applicablity of their proposed system on other solvers. [46] and [9] propose analog-based micro-architectural optimizations to reduce the data movement. However, their design is static, limited to a single solver and do no offer any sort of reconfiguration.

As summarized above, the state-of-the-art accelerators rely on pre-computational optimizations to efficiently execute the solvers. They, however, do not use the runtime performance information of solvers to tune the available hardware and they are also limited by the fixed choice of iterative solvers due to their static design. In other words, these accelerators work based on two assumptions; first, their choice of iterative solver is suitable for all types of coefficient matrices, which is not a realistic assumption; and second, the SpMV implementation will result in the most efficient performance for a given dataset, which is also not always correct as variable sparsity in the co-efficient matrix results in resource inefficiency. The following section and Table II provide more details on why these assumptions might not always be accurate.

### B. Targeted Challenges

**Solution Divergence.** The applicability of iterative solvers varies depending on the characteristics of the coefficient matrices. The solvers may fail to achieve optimal performance or even diverge, limiting the applicability of a given solver on different datasets. In other words, a solver may perform well for symmetric and diagonally dominant coefficient matrices, but exhibit poor performance when applied to a negative semi-definite coefficient matrix. In addition to their performance, a solver only guarantee convergence if its required properties are met. For instance, solvers specifically designed to handle non-symmetric or ill-conditioned matrices may diverge for matrices with properties otherwise. As a result, the choice of iterative solvers for particular coefficient matrices depends on the structural properties of the coefficient matrix. The associated conditions with our selected solvers are as follows

- **JB:** The coefficient matrix must be strictly diagonally dominant, that is, for all rows, the absolute sum of off-diagonal values must be less than the absolute diagonal value.

$$\forall i, \sum_{j \neq i} |A_{ij}| < |A_{ii}| \tag{1}$$

  where $A_{ij}$ represents the values in the $i^{th}$ row and $j^{th}$ column of coefficient matrix $A$

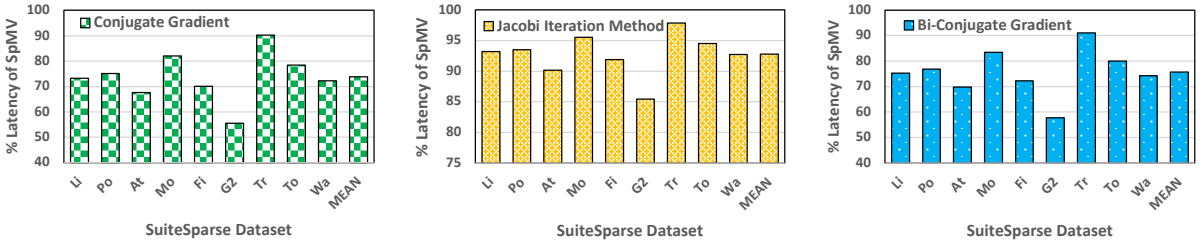


**Fig. 1: Latency of SpMV for different SuiteSparse datasets–** In the FPGA implementation of CG, JB, and BiCG-STAB solvers, SpMV consumes most of the time, making it the most expensive kernel.

- **CG:** The coefficient matrix must be symmetric and positive definite.

$$A^T = A \quad \textit{symmetric} \tag{2}$$

$$\forall\ eigenvalue(A) = \lambda > 0 \quad \textit{positive definite} \tag{3}$$

- **BiCG-STAB:** This is suitable for non-symmetric coefficient matrices, that is,

$$A^T \neq A \quad \textit{non-symmetric} \tag{4}$$

Apart from these solvers, Table I outlines the required structural properties of the coefficient matrix $A$ for some other iterative methods. We have verified these conditions across our dataset collection from the SuiteSparse matrices and summarized the convergence results for our selected three solvers in Table II. This table shows that no single solver can guarantee convergence across all datasets because of the structural properties of the coefficient matrix $A$. As a result, static DSAs for a particular solver can only be a practical solution for a limited set of workloads. Moreover, adding various DSAs to a system to accelerate a wide range of problems is neither an efficient nor a scalable solution. Therefore, as the last column of Table II highlights, our solution, Acamar (details in Section IV), not only guarantees convergence through the capability of executing different solvers but also achieves efficiency by leveraging reconfigurable computing. Acamar reconfigures hardware resources to adapt to the most suitable solver for a given input.

**Resource Underutilization.** Even if an ideal reconfigurable system can be smoothly reconfigured to specifically implement the computational requirements of different solvers, it still cannot achieve the best resource utilization, because of the *sparsity*, which manifests in SpMV shown in blue in Algorithms 1, 2, and 3. The source of resource underutilization (R.U) in SpMV is the uneven distribution of the number of non-zero (NNZ) values among the rows of a sparse matrix as governed by Equation 5. Ideally, for each row, the number of resources allocated or unroll factor (architectural details about unroll factor in Section IV-B) should be related to the number of non-zeros per row (NNZ/row) by Equation 6:

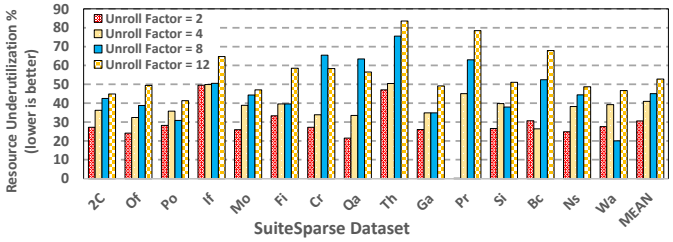


**Fig. 2: Resource Underutilization in baseline SpMV for SuiteSparse datasets –** The resource utilization changes with unroll factor. The sparse structure of matrix $A$ does not allow for one fixed unroll factor that will result in most optimal resource utilization in all cases.

$$\text{R.U} = \begin{cases} 1 - \dfrac{\text{unroll factor} - \text{mod}(NNZ/row, \text{unroll factor})}{\text{unroll factor}} \\ \qquad \text{if } NNZ/row \geq \text{unroll factor; } or \\ \dfrac{\text{unroll factor} - NNZ/row}{\text{unroll factor}} \\ \qquad \text{if } NNZ/row < \text{unroll factor} \end{cases} \tag{5}$$

$$\text{unroll factor} \quad | \quad \text{mod}\,(\text{NNZ/row}, \text{unroll factor}) = 0 \tag{6}$$

As shown in Figure 2, an FPGA implementation of the SpMV operation on a set of matrices from the SuiteSparse collection can lead to suboptimal resource utilization, which varies based on the resources allocated. Although a glance at the algorithms may indicate that the majority of operations involve dense vector computations (i.e. codes not marked in blue), a significant portion of *computational time* is still dedicated to SpMV operations, as illustrated in Figure 1.

Therefore, the inherent sparsity in SpMV operations makes these dominant computations cost ineffective in terms of resource utilization. This not only underscores the need to enhance the resource efficiency of SpMV computations within the solver but, more importantly, emphasizes that R.U must be co-optimized with execution time to achieve optimal performance, which is one of the two main goals of Acamar with details in the following section.

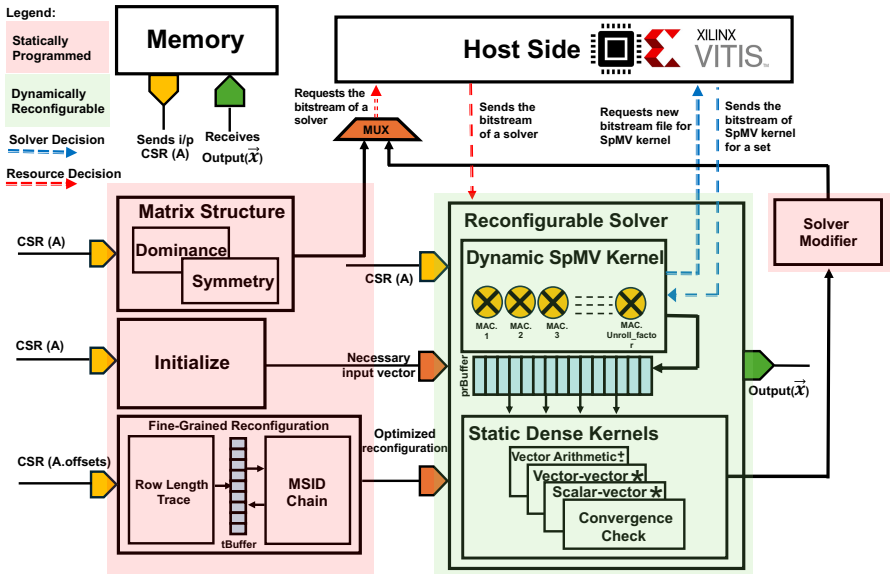


**Fig. 3: Architecture of Acamar–** It consists of static (in pink) and dynamic units (in cyan), wherein static units decides the reconfiguration events and signals the host to dynamically reconfigure the FPGA fabric.

## IV. Acamar

### A. Key Insight & Overview of the Solution

The reason for the challenges mentioned above is the static design that was prevalent in previous DSAs that do not monitor the runtime performance and reconfigure themselves at runtime. To fill this research gap, we propose Acamar, a dynamically reconfigurable accelerator that has the ability to reconfigure itself and adapt to the changes that not only promise convergence for different sparse matrix structural scenarios but also tune the amount of allocated hardware resources to the underutilized sparse computational unit, SpMV. In other words, *the key insight of Acamar is to enable a flexible accelerator that is dynamically reconfigured to match the computations required by the solver suitable for a given coefficient matrix offering a robust convergence for diverse datasets*. Additionally, Acamar features fine-grained tuning of the SpMV unit based on the pattern of coefficient matrix sparsity. To incur a minimum reconfiguration cost, Acamar also introduces a cost effective multi-stage iterative decision (MSID) chain to minimize the reconfiguration rate.

Figure 3 shows the high level overview of Acamar. As the coefficient matrix $A$ is highly sparse, it is compressed in the Compressed Sparse Row (CSR) format. The Matrix Structure unit examines the structural properties, that is, diagonal dominance and symmetry, of coefficient matrix $A$ and signals the host side to configure the dynamically reconfigurable fabric with an appropriate solver based on its findings. The Initialize unit executes the instructions preceding the start of the solver loop, as depicted in Algorithm 1, 2, and 3. The Fine-Grained Reconfiguration unit reads the offsets of compressed matrix $A$ and makes the reconfiguration decisions about the Dynamic SpMV kernel in the Reconfigurable Solver unit. It has a Row Length Trace unit that generates the average length trace of coefficient matrix rows and stores it in tBuffer. It communicates this information with the MSID unit to apply multi-stage iterative decision chain optimization, aiming to minimize the required number of reconfigurations of the Dynamic SpMV Kernel. These three units have no dependencies and run concurrently. The initialized vectors and optimized reconfiguration rate for SpMV kernel is related to the Reconfigurable Solver, which starts execution until convergence or divergence occurs. In case of divergence, the Solver Modifier unit is triggered and selects the solver with whom the Reconfigurable Solver unit should be reconfigured.

### B. Architecture & Analyzers

Acamar has two fundamental architectural categories based on how they map on FPGA fabric. **(i)** *Statically Programmed* **(ii)** *Dynamically Reconfigurable*. As its name suggests, the statically programmed components are fixed and programmed only once by the host side on the FPGA fabric. The dynamically reconfigurable side has the ability to reconfigure itself on different granularity levels. The decision about the reconfiguration is communicated to the host by the statically programmed components. One of the main goals of Acamar is to introduce dynamic reconfigurablity. To achieve this Acamar has to make decisions regarding when to initiate reconfiguration during execution and deciding the specific adjustments to be made to the dynamically reconfigurable fabric. The decisions are made using two levels of decision loops as shown in Figure 3. The Solver Decision loop, shown in red, depicts the decision about reconfiguration of the Reconfigurable Solver unit to one of the three solvers. The Resource Decision loop in blue represents the fine-grained reconfiguration of the Dynamic SpMV Kernel unit. It allocates the optimal number of resources to the Dynamic SpMV kernel based on the output from the Fine-Grained Reconfiguration unit.

**Matrix Structure Unit.** This unit is responsible for deciding the most suitable solver for the given problem. It checks the structural properties of the coefficient matrix and decides the solver with which the host side must reconfigure the

**Algorithm 4** Multi-Stage Iterative Decision Chain Algorithm

1: $rOpt$; ▷ Number of MSID-Chain stages
2: $tolernce$; ▷ Reconfiguration threshold
3: $SamplingRate$; ▷ Number of sets
4: **for** $t = 1$ to $rOpt$ **do**
5: **for** $i = 0$ to $j - 1$ **do**
6: $tBuffer^t[i] \leftarrow tBuffer^{t-1}[i]$
7: **end for**
8: **for** $i = j$ to $SamplingRate$ **do**
9: $diff \leftarrow 0$
10: $diff \leftarrow \left| \frac{tBuffer^{t-1}[k]}{tBuffer^{t-1}[k-1]} - 1 \right|$
11: **if** $diff \leq tolerance$ **then**
12: $tBuffer^t[k] \leftarrow tBuffer^{t-1}[k-1]$
13: **else**
14: $tBuffer^t[k] \leftarrow tBuffer^{t-1}[k]$
15: **end if**
16: **end for**
17: **end for**

Reconfigurable Solver unit. As discussed in Table I, there are some conditions that a coefficient matrix must satisfy to converge to a solution. The selection of our solvers, that is, JB, CG, and BiCG-STAB, is contingent upon the necessary conditions discussed in Section III-B for their convergence.

The Matrix Structure unit only checks the symmetry and diagonal dominance properties. The diagonal dominance and symmetry require trivial logic but the computational cost of finding eigenvalues is a sophisticated task and has its fair share of research on its acceleration. Therefore for CG, Acamar only checks the symmetry property. Our experiments have shown that this property was enough to ensure that, for the most part, the solution will converge for CG. As Acamar takes input in CSR format, verifying symmetry presents a challenge. To address this, the Matrix Structure unit converts the CSR format to a Compressed Sparse Column (CSC) format and compares them. If the CSC format matches the CSR format, the matrix $A$ is considered symmetric; otherwise, non-symmetric.

**Fine-Grained Reconfiguration Unit.** This unit is responsible for deciding the optimal number of allocated resources to the sparse computation units in the solvers. It enables the fine-grained tuning of the Dynamic SpMV Kernel unit to ensure maximum resource utilization. To approximate the optimal allocation of resources, a dynamically reconfigurable system is required. In an ideal scenario, the number of allocated resources should change for each row but that will be a computationally expensive task. The *Row Length Trace* unit solves this challenge by finding the optimal required resources for a certain set of rows of $A$. In our FPGA-based implementation, the unroll factor serves as a parameter to adjust resource allocation. The optimal unroll factor is calculated by averaging the size of each row (NNZ/row) within each set, given in Equation 7.

$$\text{Optimal Unroll Factor} = \frac{\sum \text{NNZ/row}}{\text{Set Size}} \quad \forall sets \in A \tag{7}$$

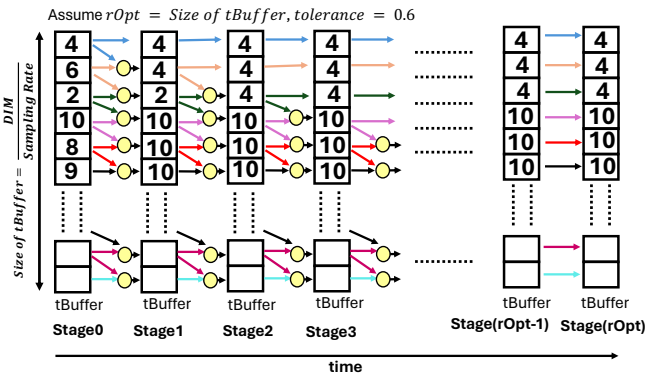


**Fig. 4: MSID-Chain–** tBuffer holds the unroll factors (4, 6, 2, 10 ...) of each set of rows. The yellow circle refers to the computations shown in Algorithm 4 line no. 10–14.

$$\text{Set Size} = \frac{\text{\#. of Rows in A}}{\text{Sampling Rate}} \tag{8}$$

where *Sampling Rate* is the parameter given by the host and defines the number of sets in coefficient matrix A.

However, this approach can lead to unique unroll factors for each set of rows, necessitating reconfiguration of the SpMV unit for each set. This can have a detrimental impact on the overall system latency. Additionally, the optimal unroll factor may vary slightly between sets, resulting in only marginal improvements. To mitigate these challenges and optimize the reconfiguration rate of the SpMV unit, we propose a multi-stage iterative decision chain implemented by the *MSID Chain* unit. Algorithm 4 shows the algorithm of MSID Chain.

MSID Chain unit calculates the normalized difference between successive optimal unroll factors of each set of rows. If the difference falls below a certain tolerance level, the MSID Chain unit maintains the previous unroll factor; otherwise, it retains the current one as shown in lines 10-14 of Algorithm 4. Figure 4 illustrates an example of how the multi-stage iterative decision chain works. The number of stages (rOpt) is a parameter and is equal to the size of tBuffer, tolerance is kept to be 0.6. Initially, the number of times we needed to reconfigure were five for the shown unroll factors, and post-optimization they were reduced to only two reconfiguration events. However, that can alter the resource utilization. Our experiments have shown that in most of cases, resource utilization is improving (details in Section VI).

**Initialize Unit.** It starts implementing the required pre-loop operations of the solver. In the case of CG and BiCG-STAB, one of these operations is the SpMV kernel, which is implemented as a static unit. As the Initialize unit runs only once, to avoid the reconfiguration latency, Acamar does not reconfigure the SpMV unit in the initialize unit and continues with an unoptimized variant of the SpMV unit.

**Reconfigurable Solver Unit.** Based on the output of Matrix Structure, the host will configure the Reconfigurable Solver to one of the three solvers, that is, either JB, CG, or BiCG-STAB. These solvers have dense and sparse computations. For the dense kernels, such as vector-vector multiplication and scalar-vector multiplication, we are implementing their most

optimized HLS design. The dense kernel units will not be reconfigured on the runtime, as they are not the main source of resource underutilization. The SpMV kernel, however, is a sparse kernel and it must be reconfigured to optimize the resource utilization. The host will reconfigure the Dynamic SpMV Kernel unit in the Reconfigurable Solver based on the optimized configurations obtained from the Fine-Grained Reconfiguration. The host may or may not reconfigure the Dynamic SpMV Kernel for each set. The output of the Dynamic SpMV kernel will be temporarily stored in the prBuffer and forwarded to other dense kernels once SpMV finishes. The Reconfigurable Solver unit runs until either the solution converges or diverges. If the solution diverges, the Solver unit interrupts and triggers the Solver Modifier unit. In case of convergence, it forwards the output vector $x$ to the memory.

**Solver Modifier Unit.** As mentioned previously, we are only verifying one out of two convergence conditions for the CG solver. Consequently, divergence may occur in certain datasets. To ensure convergence nonetheless, the Solver Modifier unit intervenes by transitioning the solver to an alternative option from the available choices. Additionally, the Solver Modifier unit triggers the Initialize unit to reset and resend the values to the Reconfigurable Solver unit. The working mechanism of the Solver Modifier is straightforward and runs by assigning the solver whose corresponding bit is low in a temporary register.

## V. Experimental Setup

### A. Simulation Infrastructure

We model Acamar based on Xilinx Alveo u55c FPGA implementation. The hardware has been defined in Xilinx Vitis HLS and optimized using the HLS pragmas. We implement the most optimized design of static units for both Acamar as well as the static baseline. The SpMV unit is optimized dynamically on runtime using the partial reconfigurablity feature available in Alveo u55c. To extend our design space exploration, we use a cycle-level simulator that takes the performance numbers from the HLS co-simulation and runs exhaustive experiments while changing different parameters of Acamar.

### B. Computing Precision

Scientific computing problems demand high precise calculations to minimize residual errors, a crucial factor in achieving accurate results. However, higher precision comes at the cost of increased computational resources, particularly in floating-point arithmetic. Typically, scientific computing accelerators employ either 32-bit or 64-bit floating-point precision for calculations [5], [9], [16], [38], [46], [47]. In Acamar, we use 32-bit numbers for the floating point computations. The convergence criteria are kept fixed in all the solvers so as to achieve convergence within a threshold of $10^{-5}$. Acamar has the ability to reconfigure the FPGA fabric with a new solver in case of divergence. Acamar gives each solver a setup time before it begins checking for the divergence. The setup time increases with the problem size. In Acamar the problem size is fixed to $4096 \times 4096$ and hence, the setup time is 200 iterations for each solver.

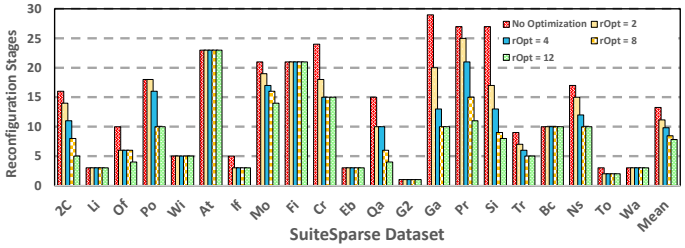


**Fig. 5: Reconfiguration rate for different MSID Chain stages –** It becomes almost constant after $rOpt = 8$, making it a suitable reconfiguration rate for the experiments

### C. Datasets

For evaluations, we use the SuiteSparse [12] matrix data collection. Table II shows the datasets we are using for our evaluation. These matrices exhibit diverse structural properties; they consist of a combination of strictly diagonally dominant, symmetric, and non-symmetric matrices, making them suitable for evaluating our dynamically reconfigurable system. Acamar processes the matrices in $4096 \times 4096$ chunks.

### D. Hardware Configurations

The parameters discussed below affect the performance of Acamar. We run experiments with a combination of possible configurations and discuss their effects in Section VI.

**Sampling Rate:** It is the number of sets of rows in coefficient matrix $A$. It defines the number of rows per set, that is, Set Size, using the following equation:

$$\text{Set Size} = \frac{\text{\#. of Rows in A}}{\text{Sampling Rate}} \tag{9}$$

A larger sampling set means a smaller set size and more fine-tuning of the unroll factor. We have discussed the implication of using different sampling rates in Section VII-B. We observe that a sampling rate of 32 works well in almost all the cases. Hence, we choose $SamplingRate = 32$ to compare Acamar with the baselines.

**rOpt:** It defines the number of stages in the multi-stage iterative decision chain. 0 implies no optimization, 1 implies a single-stage chain, and so on. The number of stages directly affects the reconfiguration rate of the Dynamic SpMV Kernel. More stages imply more iterative decisions and therefore a reduction in the reconfiguration rate. However, Figure 5 shows that the change in reconfiguration rate becomes almost constant after $rOpt = 8$. Hence, for the sake of comparison with baseline, we are choosing $rOpt = 8$. Tuning MSID Chain stages can have implications on resource utilization and latency. More details are discussed in Section VII-A.

**tolerance:** It tunes the tolerance level of a multi-stage iterative chain. A number greater than $0.5$ signifies a more tolerable system that can result in a smaller reconfiguration rate but possible wasted resources. For the sake of our experiments, we have kept it equal to 0.15

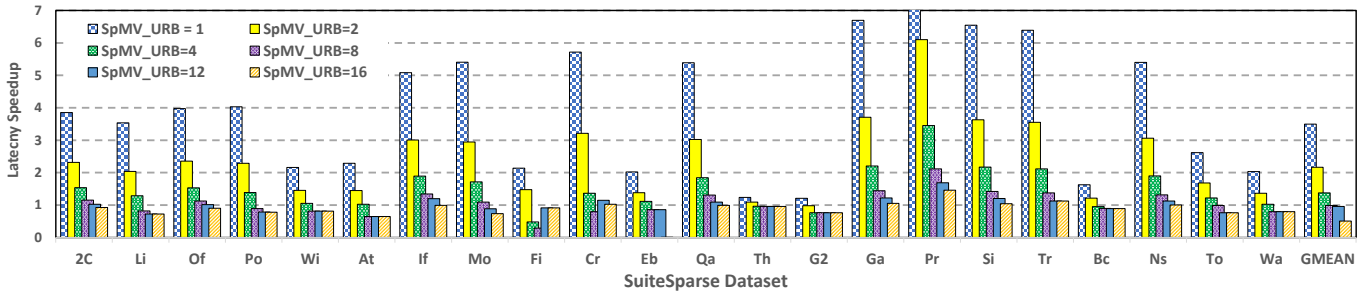


**Fig. 6: Latency speedup of Acamar over static design for different SuiteSparse datasets–** Acamar latency speedup reduces as the allocated resources to the baseline increases and becomes constant after a while due to insufficient NNZ per row to take advantage of surplus resources in the baseline. (*GMEAN* refers to Geometric Mean).

**SpMV_URB:** For the baselines, the SpMV unit is static and must have a fixed number of allocated resources or in other words, a fixed unroll factor. SpMV_URB assigns the unroll factor of the baseline SpMV unit. The performance of Acamar against varying SpMV_URB have been discussed in detail in Sections VI-A and VI-B.

### *E. Baselines*

To the best of our knowledge, Acamar is the first FPGA-based accelerator capable of dynamic partial reconfiguration for scientific computing problems. For a fair comparison, we refrained from comparing Acamar to other state-of-the-art accelerators tailored for scientific problems. Instead, we compare it to a static design that incorporates the same optimized static units as Acamar, as well as a static configuration of the SpMV unit, while including all optimizations for a concise evaluation. We run our baseline across all three iterative solvers and, in cases where convergence is achieved, we assess the performance for each solver. To analyze the the effectiveness of fine-grained resource allocation in SpMV kernel, we implement SpMV in cuSparse library using the Nvidia open-source code [1] and test it on Nvidia GTX 1650 Super running on Cuda v11.6. We used Nvidia Nsight toolkit to run GPU evaluation. In the evaluation section, we detail the resource utilization, latency, achieved throughput, performance efficiency, and area saving of Acamar. Since resource allocation on FPGAs is controlled by the unroll factor, we use these terms interchangeably throughout the evaluation section.

## VI. Evaluations and Results

The goal of Acamar is to guarantee convergence for different coefficient matrices while promising optimal resource utilization. Table II shows that our system offers robust convergence for the cases where a static iterative solver system would fail. In this section, the main focus is on the performance evaluation. As discussed in Section III-B, the SpMV kernel stands out as the most demanding kernel of iterative solvers (Figure 1). In Acamar, we aim to allocate the optimal number of resources to the SpMV kernel so that we can minimize resource underutilization via reconfiguration.

### *A. Speedup*

Figure 6 shows the latency speedup of Acamar against different resource allocations in the baseline. For the baseline, we assume the same solver that is being used in Acamar. We observe a substantial improvement in latency upto $11.61\times$ if we consider an un-optimized implementation of SpMV w.r.t latency on the FPGA, that is, the allocated resources (unroll factor) are equal to 1 (only a single MAC unit). However, as we increase the number of allocated resources to the SpMV unit, the improvement starts diminishing with marginal changes in speedup for $SpMV\_URB > 16$. $SpMV\_URB = 1$ implies that the baseline is optimized solely for resource utilization. There will be only 1 MAC unit in the SpMV kernel and it will run for every non-zero value, resulting in 0% resource underutilization. However, in terms of latency, this is the worst-case scenario. In this case, there will be no opportunity for parallelism and the baseline will suffer from high computational time costs. Acamar, on the other hand, decides the best unroll factor on the fly with the aid of Dynamic SpMV kernel and Fine-Grained Reconfiguration units. It will allocate different numbers of resources for a given set of rows. This promises a substantial improvement in latency against sub-optimal baseline w.r.t latency.

For $SpMV\_URB$ greater than 16, the latency improvements become marginal and almost constant. The reason is the inability of the baseline SpMV kernel to take advantage of extra allocated resources. The additional resources are wasted as there are no more non-zero values in a specific row, leading to diminishing returns in latency improvement.

It is worth mentioning that for the baseline, we are optimistically choosing the solver that offers convergence for the given dataset. Practically, it is quite possible that a solver diverges for a dataset as shown with some examples in Table II. The divergence not only leads to false or no solution but also results in unbounded execution time.

### *B. SpMV Resource Utilization*

Resource utilization is related to the sparse computations happening in the solver, related to the number of MAC units that are busy in the sparse kernel. As the only sparse kernel in our choice of solvers is SpMV, as shown in blue in Algorithm 1, 2, and 3, we discuss improvements in SpMV

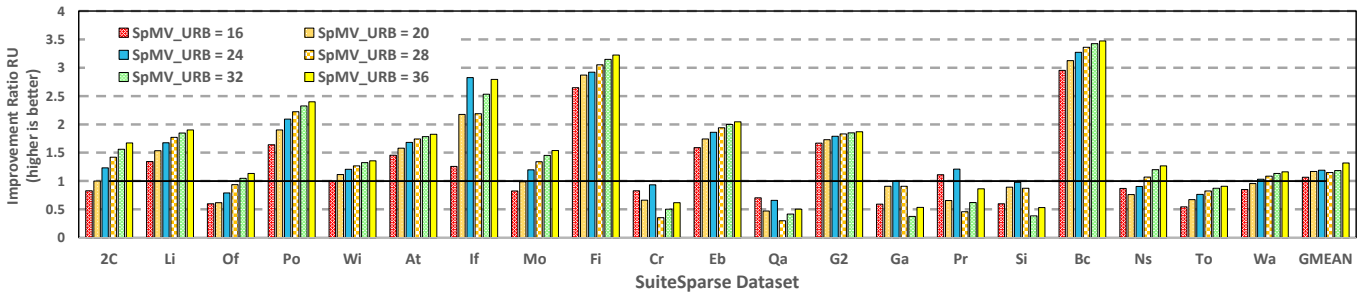


**Fig. 7: Improvement ratio in resource underutilization in Acamar for different SuiteSparse datasets (higher is better)–** Resource underutilization improvement becomes significant due to sub-optimal resources allocation in the baseline.

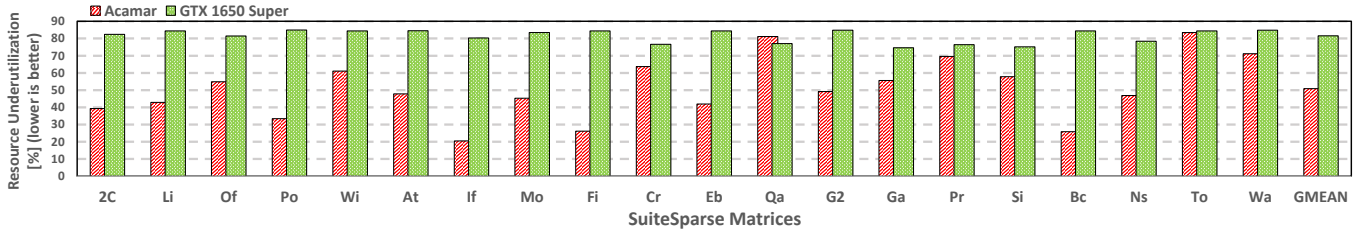


**Fig. 8: Resource underutilization in Acamar and Nvidia GTX 1650 Super (lower is better)–** GPU fails to achieve high compute units occupancy due to sparse non-zero values, resulting in high underutilization.

kernel resource utilization in this section. The resource underutilization in the baseline is susceptible to increase as the number of allocated resources increases. This is due to the insufficient number of non-zeros per row. If the allocated resources exceed the number of non-zeros in a row, the SpMV unit suffers from resource underutilization, leading to sub-optimal performance. Acamar, informed by the number of non-zeros per set of rows makes a deliberate decision about the allocated resources. This results in resource utilization improvement in SpMV of upto $3\times$ as shown in Figure 7.

For most of the cases, the improvement is increasing if we increase the allocated resources in the baseline. This signifies the pros of using a dynamic solution to decide the number of allocated resources. However, when the baseline allocates a very small number of resources to the SpMV kernel, we do not observe substantial resource utilization improvement as we rely on the average of NNZ/row in a given set of rows to decide the best unroll factor. In these cases, the baseline assumes the number of resources conservatively to keep the resource underutilization at the minimum but ends up paying the cost in terms of computational latency. Acamar on the other hand, aims to strike a balance between latency and resource utilization with its fine-grained reconfiguration solution.

Figure 8 shows the resource underutilization in Nvidia 1650 Super GPU and Acamar. Each SM in GPU house multiple thread blocks, that act as computing units. However, it is observed that the computing units are not fully utilized and the resources are highly underutilized. On the other hand, Acamar resource underutilization is very small as compared to the GPU, making it a more reliable choice. On average Acamar is underutilized 50% compared to 81% underutilized GPU.

### C. Achieved Throughput

Effective use of compute resources leads to improved achieved throughput. Figure 9 shows the percentage of peak throughput achieved by Acamar and the baselines. On average Acamar achieves 70% throughput of the peak throughput and can go up to 83%. In the top graph, Acamar informed by dynamic decision about the unroll factor achieves better compute throughput as compared to the fixed unroll factor baseline. However, in some cases, e.g. in Pr and Cr workloads, Acamar does not always yield better throughput consumption. This is because of the highly random nature of the sparse matrices, making our MSID-chain render slightly sub-optimal unroll factors. In the bottom graph, Acamar is compared with 1650 Super GPU. GPU achieves very low of the peak throughput because it has a large number of compute units but only few of them are utilized in the SpMV operation.

### D. Performance Efficiency

We define *performance efficiency* as the number of floating point operations (FLOPS) per square millimeter area of FPGA fabric. Performance efficiency helps us to analyze the pros of utilizing the resources effectively. Greater performance efficiency means increased number of FLOPS achieved alongside a smaller area. The dynamic architecture of Acamar allows the FPGA to harness more arithmetic operations while sparing more resources (area) to execute another application or kernel that is sharing the same FPGA fabric. Figure 10 shows the FLOPS per square millimeter area of Acamar vs. fixed unroll factor baselines. On average Acamar achieves 720 $GLOPS/mm^2$ performance efficiency. For some cases in Ga, Pr, Si workloads, it is less than the baseline owing to the same reason mentioned in VI-C Increased performance efficiency means more area saving in Acamar. On average

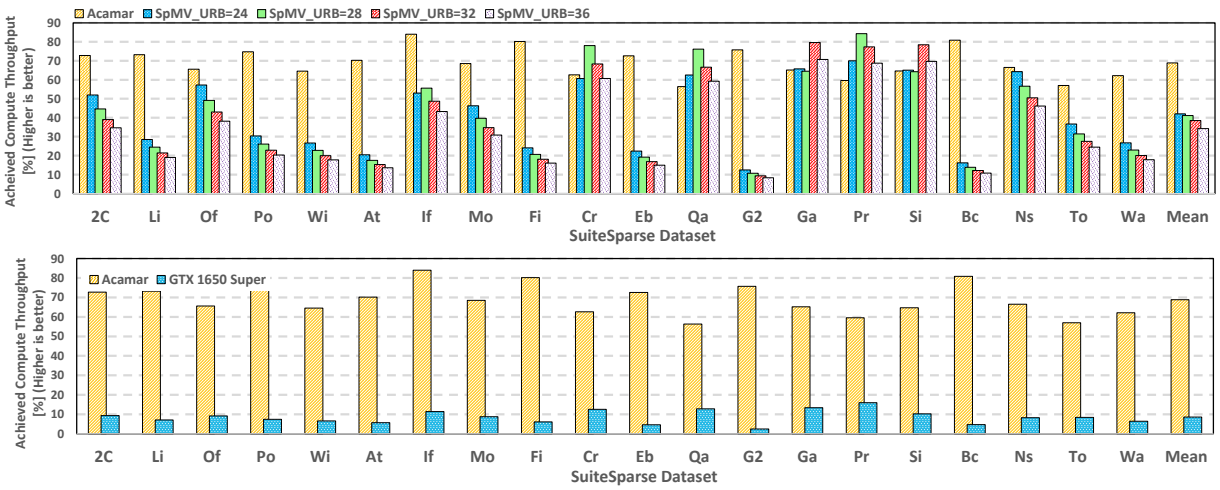


**Fig. 9: Achieved compute throughput as a percentage of peak throughput (higher is better)– Top.** Acamar vs. Static design. **Bottom.** Acamar vs. Nvidia GTX 1650 Super.

Acamar is $2\times$ more area efficient than a static design. This gives more area for the deployment and production of a co-running application on the same FPGA. It is worth mentioning that the performance efficiency and area saving evaluation is done only with a static SpMV design while excluding GPUs because the partial reconfiguration is a concept limited to FPGAs only and the comparison with GPU would not have been sound and fair.

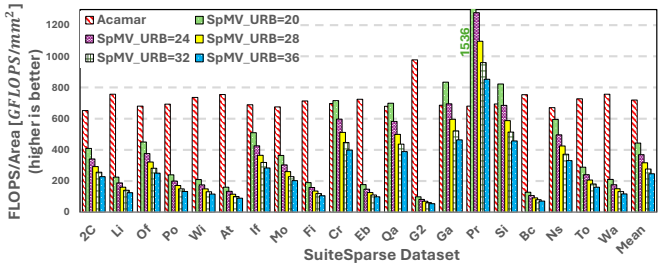


**Fig. 10: Performance Efficiency of Acamar and static baseline design (higher is better)–** Acamar gives higher throughput per unit area, leaving more area of the FPGA fabric to be configured for another application.

## VII. Design Space Exploration

In this section, we evaluate the effects of changing the parameters of Acamar. The two parameters being discussed in this section are MSID Chain stages and Sampling Rate.

### A. Effect of MSID Chain Stages

The main objective of the MSID Chain unit is to minimize the number of times the Dynamic SpMV Kernel unit is reconfigured. As shown in Figure 5, the MSID Chain successfully reduces the reconfiguration rate. However, this can have potential repercussions on the performance of Acamar. There are two situations that can happen;

- First, assume there are 8 non-zeros in a row and the optimal number of initial unroll factors for its set of rows is 4. According to Equation 5 that means 0% resource underutilization but if the MSID chain changes it to 10, this signifies a loss of

$$\frac{10-8}{10} \times 100 = 20\% \text{ resource underutilization} \quad (10)$$

  But the advantage is the increase in available parallelism and consequently, latency improvement.
- The second case is when the resource utilization improves. For example, if there are 6 non-zero values in a row of a set and the initial unroll factor is 7, signifying 14% resource underutilization. Now, the MSID Chain changes the unroll factor to 3. According to Equation 5, this will improve the resource underutilization from 14% to

$$(1-\frac{3-mod(6,3)}{3})\times 100 = 0\% \text{ resource underutilization} \quad (11)$$

  This improvement is at the expense of lost parallelism and Acamar incurs more latency.

In an ideal scenario, we want the MSID Chain to not significantly affect the performance of a solver. Therefore, the only notable advantage of MSID Chain will be reflected in the reduced reconfiguration time and not in the SpMV kernel itself. Figure 11 shows the change in resource underutilization and latency of the Dynamic SpMV Kernel unit as the number of stages (rOpt_stage) changes. For almost all the cases, we observe the same results (Li, Wi, Fi, C etc) with very small changes in some cases (Pr, Ns, 2C). These changes could be due to one of the above-mentioned cases and may go in our favour or against us. Such a uniform behavior is good for Acamar as it ensures that the system does not lean towards either latency or resource utilization and maintains a balance between both of the metrics.

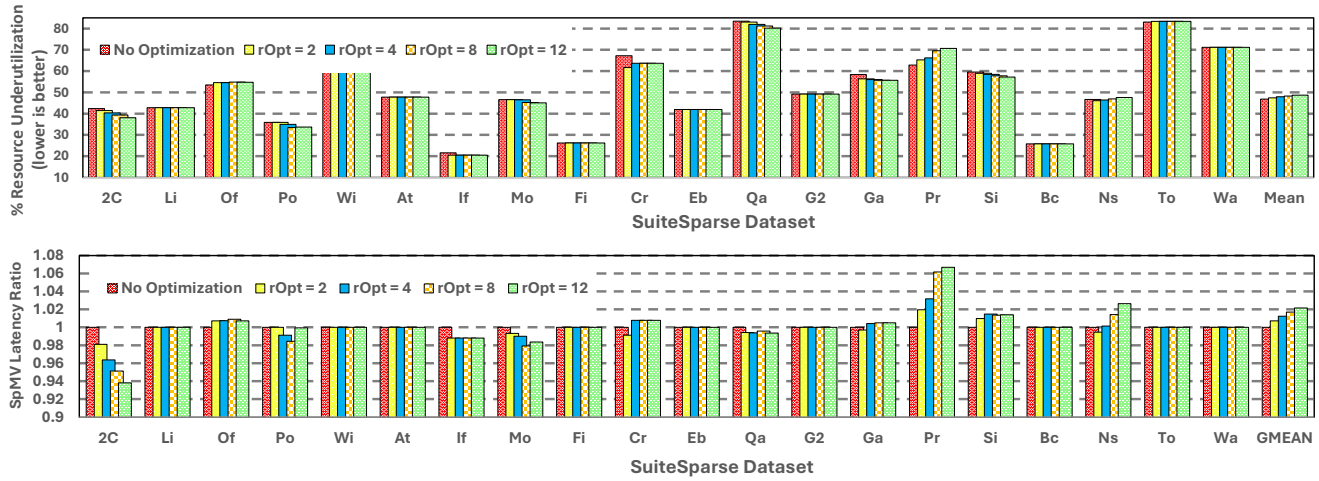


**Fig. 11: Resource underutilization and change in SpMV latency for different MSID Chain stages –** Both the resource underutilization and SpMV latency remains almost constant post optimization, making them naive to rOpt changes.

### *B. Effect of Sampling Rate*

Sampling Rate defines the maximum number of times Dynamic SpMV Kernel is reconfigured. It is inversely related to the set size according to the Equation 9. It changes the granularity levels of reconfiguration and directly affects the performance of the system. A bigger sampling rate signifies a small set size, allowing for better analysis of the rows in one set. This can result in resource utilization improvement. However, it can result in more reconfiguration instances, making it expensive in terms of latency. For instance, if the problem size is $4096 \times 4096$ and the sampling rate is $4096$, that means the host will reconfigure the Dynamic SpMV Kernel $4096$ times resulting in the reconfigured SpMV kernel that will guarantee $100\%$ resource utilization. However, this is not practical as the time spent in reconfiguration will increase and impact the overall execution of the system adversely. Figure 12 shows the effect of changing sampling rate on resource underutilization after the MSID Chain optimization. We can observe that with the increasing sampling rate the resource underutilization is decreasing. To strike a fair balance between reconfiguration latency and resource underutilization, we fix sampling rate to 32 in our experiments.

## VIII. Discussions

### *A. Reconfiguration Time*

Modern FPGAs feature dynamic partial reconfiguration to allow a part of the design to modify itself while the rest of the design is running. This makes a system more robust to different exceptions and situations that can occur on the runtime. It also allows the designer to use the same FPGA fabric for diverse applications. AMD Xilinx uses Dynamic Function eXchange (DFX) to refer to dynamic partial reconfiguration. In this work, we work with Xilinx Alveo u55c which is equipped with Virtex UltraScale+ FPGA. Xilinx Ultrascale+ family allows reconfiguration for all of the FPGA resources including clocking networks, gigabit transceivers, I/O ports, as well as basic computing resources, that is, lookup tables, flip flops, and digital signal processors. It also allows Nested DFX to reconfigure a module within a reconfigurable module. Acamar uses Nested DFX that allows the reconfiguration of the Dynamic SpMV Kernel unit within the Reconfigurable Solver unit. An important consideration in dynamically reconfigured designs is the time spent in reconfiguration which depends on the interface used for partial bitstream loading and transfer, which in our targeted platform is Xilinx Hardware Internal Configuration Access Port (ICAP) core that allows bitstream transfer and reconfiguration. It runs at a frequency of 200MHz and offers a reconfiguration speed of 6.4Gb/s. The reconfiguration time is directly related to the bitstream size. To ensure that Acamar incurs the same or less latency as the baseline, it is crucial to keep the reconfiguration latency within specific bounds. Figure 13 shows the bounds within which the reconfiguration must be done. It is worth mentioning that the main goal of Acamar was to offer robust convergence and improve resource utilization. The latency considerations, while significant, are a secondary outcome of our work and its relevance may vary depending on the specific use case.

### *B. An Overview of Broader Related Studies*

In addition to the DSAs previously discussed, which are more related to our work, this section presents a comprehensive overview of past efforts in this field to clarify Acamar's position within the broader context.

***DSAs for Scientific Computing & SpMV.*** The enhancement of scientific computation initially emphasized GPU-based acceleration of PDE solvers [6], [44], [48]. Following the rise of DSAs in AI and neural networks, their potential benefits for scientific computing were recognized. Since 2016, various DSAs such as analog computing, hybrid solutions for nonlinear PDEs, cellular nonlinear networks, memristive systems, and processing-in-memory technologies have been developed to tackle complex scientific computations [8], [9], [16], [20], [24], [29], [37], [46], [52], [65]. FPGAs have also been employed to expedite algorithms like 2D and 3D FDTD, enhancing memory hierarchy optimization [10], [14], [28], [70].

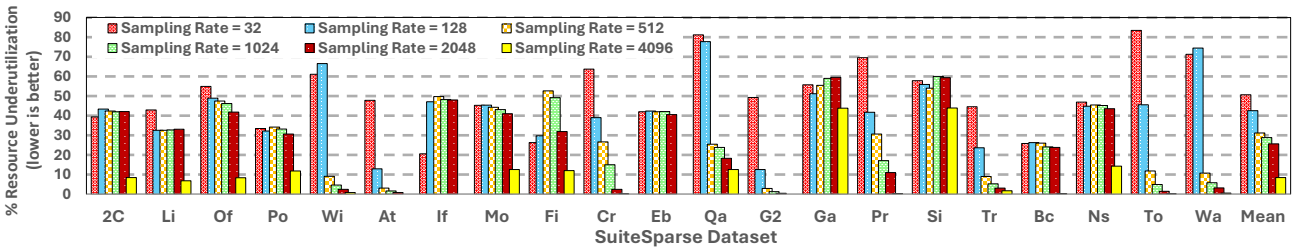


**Fig. 12: Resource underutilization for different sampling rates –** Increasing the sampling rate results in finer granularity of reconfiguration, improving resource utilization but at the expense of increased reconfiguration latency.

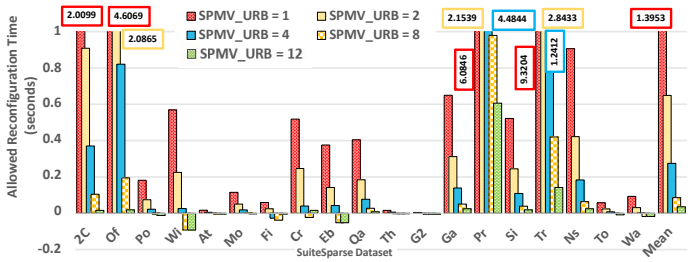


**Fig. 13: Allowed reconfiguration time –** The reconfiguration must be done within these bounds to ensure latency equal to or less than the baseline.

Beyond solving PDEs, DSAs have facilitated improvements in matrix inversion processes, such as the LU decomposition method on systolic arrays and blocked approaches on chips for enhanced parallelism [2], [4], [23], [27], [34], [72]. In addition to such scientific-computing-specific DSAs, several hardware accelerators have been proposed to improve the performance of SpMV that can be used for accelerating scientific computing as well but they are static designs, too [3], [17], [30], [39], [41], [49], [51], [59], [60], [64].

***Reconfigurable DSAs and Hardware Search.*** Reconfigurable accelerators, embracing a wide spectrum of domains, employ techniques such as data-centric parallel computing and dataflow architectures [22], [53], [62], [67], [75], [76]. They utilize dynamic partial reconfigurability in FPGAs to adapt DNN accelerators flexibly [26]. Enhancing DSA flexibility involves techniques ranging from mapping problems to hardware, using ML for resource assignment, to flexible interconnection networks [7], [31]–[33], [40], [50], [61], [62], [77], [78]. Machine learning, especially reinforcement learning, has been instrumental in resource management for DNNs and in improving specific tasks like prefetching [7], [32]. Innovations also include scalable reconfigurable accelerators and architectural explorations for efficient network-on-chip designs [31], [40], [50], [78]. Recent efforts aim to co-explore neural architectures and ASIC designs for multiple tasks, evaluating various dataflows and acceleration features for their efficiency in speed and energy consumption [68], [71], [73].

***Software-based Optimizations for Scientific Computing.*** The common theme of recent software-based sparse accelerators for scientific computing revolves around memory-related GPU optimizations. For instance, CUDA-based GPU parallelization optimizes 3D finite difference computations by leveraging data parallelism and efficient memory access patterns, achieving high throughput and scalability across multiple GPUs [45]. [19] employs specialized algorithms and advanced memory management strategies in GPU implementations of PDE models. [42] advocates for the use of domain-specific languages like UFL to tailor FEM implementations for GPUs and multicore CPUs, optimizing algorithms and data structures. [18] develops efficient GPU-based algorithms for the entire FEM pipeline, including optimized local element information generation and parallel linear system solving with algebraic multigrid preconditioning. [25] optimizes memory arrangement for GPU-accelerated finite element simulations, emphasizing efficient data mapping and memory access strategies. [43] proposes a matrix-free GPU implementation of FGFEA, exploiting grid regularity and on-chip memory utilization to minimize memory transactions and maximize parallel computational throughput. [15] introduces a GPU-based approach for efficient sparse linear systems generation in electromagnetics, utilizing parallel matrix assembly to exploit GPU computational power. [36] evaluates scalable FEM algorithms on multi-core CPUs and GPUs, focusing on optimized mesh partitioning and memory access patterns to minimize data conflicts. These papers showcase a range of software-based meomry optimizations tailored to enhance across various scientific computing applications.

## IX. Conclusions

This paper proposed Acamar, an innovative FPGA-based dynamically reconfigurable accelerator for diverse scientific computing workloads. Through dynamic reconfiguration, Acamar transcends the constraints of static design and lack of generality, providing tailored solutions for varying structures of coefficient matrices. The capacity to seamlessly transition between solvers including JB, CG, and BiCG-STAB, coupled with the reconfigurability of the SpMV unit, ensures robust convergence and optimized performance. Acamar's MSID chain further enhances the efficiency by minimizing reconfiguration overhead. This unique approach not only optimizes resource utilization but also paves the way for a new era of DSAs that adapt in real-time to the demands of scientific problems, marking a significant leap forward from the inefficiencies observed in current supercomputing practices.

## Acknowledgment

We gratefully acknowledge the support of US Department of Energy (DoE) under the ASCR ECRP, Award DE-SC0024079.

## REFERENCES


[1] "Nvidia cusparse library samples," [Accessed: June-24th-2024]. [Online]. Available: https://github.com/NVIDIA/CUDALibrarySamples/tree/master/cuSPARSE/spmv_csr

[2] J. Arias-García, R. P. Jacobi, C. H. Llanos, and M. Ayala-Rincón, "A suitable fpga implementation of floating-point matrix inversion based on gauss-jordan elimination," in 2011 vii southern conference on programmable logic (SPL). IEEE, 2011, pp. 263–268.

[3] B. Asgari, R. Hadidi, J. Dierberger, C. Steinichen, A. Marfatia, and H. Kim, "Copernicus: Characterizing the performance implications of compression formats used in sparse workloads," in IISWC. IEEE, 2021, pp. 1–12.

[4] B. Asgari, R. Hadidi, N. S. Ghaleshahi, and H. Kim, "Pisces: Power-aware implementation of slam by customizing efficient sparse algebra," in DAC. ACM, 2020, p. 233.

[5] B. Asgari, R. Hadidi, T. Krishna, H. Kim, and S. Yalamanchili, "Alrescha: A lightweight reconfigurable sparse-computation accelerator," in 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2020, pp. 249–260.

[6] A. Balevic, L. Rockstroh, A. Tausendfreund, S. Patzelt, G. Goch, and S. Simon, "Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose gpus," in 2008 11th IEEE International Conference on Computational Science and Engineering. IEEE, 2008, pp. 327–334.

[7] R. Bera, K. Kanellopoulos, A. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A customizable hardware prefetching framework using online reinforcement learning," in MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021, pp. 1121–1137.

[8] T. Chen, J. Botimer, T. Chou, and Z. Zhang, "An sram-based accelerator for solving partial differential equations," in 2019 IEEE Custom Integrated Circuits Conference (CICC). IEEE, 2019, pp. 1–4.

[9] T. Chen, J. Botimer, T. Chou, and Z. Zhang, "A 1.87-mm 2 56.9-gops accelerator for solving partial differential equations," IEEE Journal of Solid-State Circuits, vol. 55, no. 6, pp. 1709–1718, 2020.

[10] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport, "An fpga implementation of the two-dimensional finite-difference time-domain (fdtd) algorithm," in Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, 2004, pp. 213–222.

[11] E. Chow and Y. Saad, "Approximate inverse preconditioners via sparse-sparse iterations," SIAM Journal on Scientific Computing, vol. 19, no. 3, pp. 995–1023, 1998.

[12] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," ACM Transactions on Mathematical Software (TOMS), vol. 38, no. 1, p. 1, 2011.

[13] J. Douglas and J. P. Wang, "An absolutely stabilized finite element method for the stokes problem," Mathematics of computation, vol. 52, no. 186, pp. 495–508, 1989.

[14] J. P. Durbano and F. E. Ortiz, "Fpga-based acceleration of the 3d finite-difference time-domain method," in 12th Annual IEEE symposium on field-programmable custom computing machines. IEEE, 2004, pp. 156–163.

[15] A. Dziekonski, P. Sypek, A. Lamecki, and M. Mrozowski, "Finite element matrix generation on a gpu," Progress In Electromagnetics Research, vol. 128, pp. 249–265, 2012.

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

[17] S. Feng, X. He, K.-Y. Chen, L. Ke, X. Zhang, D. Blaauw, T. Mudge, and R. Dreslinski, "Menda: a near-memory multi-way merge solution for sparse transposition and dataflows," in Proceedings of the 49th Annual International Symposium on Computer Architecture, 2022, pp. 245–258.

[18] Z. Fu, T. J. Lewis, R. M. Kirby, and R. T. Whitaker, "Architecting the finite element method pipeline for the gpu," Journal of computational and applied mathematics, vol. 257, pp. 195–211, 2014.

[19] M. Giles, E. László, I. Reguly, J. Appleyard, and J. Demouth, "Gpu implementation of finite difference solvers," in 2014 Seventh Workshop on High Performance Computational Finance. IEEE, 2014, pp. 1–8.

[20] N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis, "Energy-efficient hybrid analog/digital approximate computation in continuous time," IEEE Journal of Solid-State Circuits, vol. 51, no. 7, pp. 1514–1524, 2016.

[21] M. A. Heroux, J. Dongarra, and P. Luszczek, "Hpcg benchmark technical specification," Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2013.

[22] R. Hojabr, A. Sedaghati, A. Sharifian, A. Khonsari, and A. Shriraman, "Spaghetti: Streaming accelerators for highly sparse gemm on fpgas," in 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2021, pp. 84–96.

[23] B. Holanda, R. Pimentel, J. Barbosa, R. Camarotti, A. Silva-Filho, L. Joao, V. Souza, J. Ferraz, and M. Lima, "An fpga-based accelerator to speed-up matrix multiplication of floating point operations," in 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum. IEEE, 2011, pp. 306–309.

[24] Y. Huang, N. Guo, M. Seok, Y. Tsividis, K. Mandli, and S. Sethumadhavan, "Hybrid analog-digital solution of nonlinear partial differential equations," in MICRO. IEEE, 2017, pp. 665–678.

[25] P. Huthwaite, "Accelerated finite element elastodynamic simulations using the gpu," Journal of Computational Physics, vol. 257, pp. 687–707, 2014.

[26] H. Irmak, D. Ziener, and N. Alachiotis, "Increasing flexibility of fpga-based cnn accelerators with dynamic partial reconfiguration," in 2021 31st International Conference on Field-Programmable Logic and Applications (FPL). IEEE, 2021, pp. 306–311.

[27] A. Irturk, B. Benson, S. Mirzaei, and R. Kastner, "An fpga design space exploration tool for matrix inversion architectures," in 2008 Symposium on Application Specific Processors. IEEE, 2008, pp. 42–47.

[28] Y. Ishigaki, Y. Tomioka, T. Shibata, and H. Kitazawa, "An fpga implementation of 3d numerical simulations on a 2d simd array processor," in 2015 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2015, pp. 938–941.

[29] M. Jacquelin, M. Araya-Polo, and J. Meng, "Scalable distributed high-order stencil computations," in 2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society, 2022, pp. 411–423.

[30] A. K. Jain, H. Omidian, H. Fraisse, M. Benipal, L. Liu, and D. Gaitonde, "A domain-specific architecture for accelerating sparse matrix vector multiplication on fpgas," in 2020 30th International conference on field-programmable logic and applications (FPL). IEEE, 2020, pp. 127–132.

[31] Y. Kan, M. Wu, R. Zhang, and Y. Nakashima, "Mugra: A scalable multi-grained reconfigurable accelerator powered by elastic neural network," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 69, no. 1, pp. 258–271, 2021.

[32] S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2020, pp. 622–636.

[33] S.-C. Kao and T. Krishana, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in 2020 Internation Conference on Computer Aided Design. ACM, 2020, pp. 1–9.

[34] M. Karkooti, J. R. Cavallaro, and C. Dick, "Fpga implementation of matrix inversion using qrd-rls algorithm," in Asilomar Conference on Signals, Systems, and Computers, 2005.

[35] V. S. K. Kokkiligadda, V. Naikoti, G. S. Patkotwar, S. L. Sabat, and R. Peesapati, "Fpga-based hardware accelerator for matrix inversion," SN Computer Science, vol. 4, no. 2, p. 147, 2023.

[36] S. Kopysov, A. Novikov, N. Nedozhogin, and V. Rychkov, "Scalability of parallel finite element algorithms on multi-core platforms," in IOP Conference Series: Materials Science and Engineering, vol. 158, no. 1. IOP Publishing, 2016, p. 012055.

[37] J. Kung, Y. Long, D. Kim, and S. Mukhopadhyay, "A programmable hardware accelerator for simulating dynamical systems," ACM SIGARCH Computer Architecture News, vol. 45, no. 2, pp. 403–415, 2017.

[38] J. Li, Y. Zhang, H. Zheng, and K. Wang, "Fdmax: An elastic accelerator architecture for solving partial differential equations," in Proceedings of the 50th Annual International Symposium on Computer Architecture, 2023, pp. 1–12.

[39] S. Li, D. Liu, and W. Liu, "Optimized data reuse via reordering for sparse matrix-vector multiplication on fpgas," in 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD). IEEE, 2021, pp. 1–9.

[40] T.-R. Lin, D. Penney, M. Pedram, and L. Chen, "A deep reinforcement learning framework for architectural exploration: A routerless noc case

study," in 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2020, pp. 99–110.

[41] B. Liu and D. Liu, "Towards high-bandwidth-utilization spmv on fpgas via partial vector duplication," in Proceedings of the 28th Asia and South Pacific Design Automation Conference (ASP-DAC), 2023, pp. 33–38.

[42] G. R. Markall, D. A. Ham, and P. H. Kelly, "Towards generating optimised finite element solvers for gpus from high-level specifications," Procedia Computer Science, vol. 1, no. 1, pp. 1815–1823, 2010.

[43] J. Martínez-Frutos and D. Herrero-Pérez, "Efficient matrix-free gpu implementation of fixed grid finite element analysis," Finite Elements in Analysis and Design, vol. 104, pp. 61–71, 2015.

[44] D. Michéa and D. Komatitsch, "Accelerating a three-dimensional finite-difference wave propagation code using gpu graphics cards," Geophysical Journal International, vol. 182, no. 1, pp. 389–402, 2010.

[45] P. Micikevicius, "3d finite difference computation on gpus using cuda," in Proceedings of 2nd workshop on general purpose processing on graphics processing units, 2009, pp. 79–84.

[46] J. Mu and B. Kim, "29.2 a 21× 21 dynamic-precision bit-serial computing graph accelerator for solving partial differential equations using finite difference method," in 2021 IEEE International Solid-State Circuits Conference (ISSCC), vol. 64. IEEE, 2021, pp. 406–408.

[47] J. Mu, C. Yu, T. T.-H. Kim, and B. Kim, "A scalable bit-serial computing hardware accelerator for solving 2d/3d partial differential equations using finite difference method," in ESSCIRC 2022-IEEE 48th European Solid State Circuits Conference (ESSCIRC). IEEE, 2022, pp. 353–356.

[48] T. Okimura, T. Sasayama, N. Takahashi, and S. Ikuno, "Parallelization of finite element analysis of nonlinear magnetic fields using gpu," IEEE transactions on magnetics, vol. 49, no. 5, pp. 1557–1560, 2013.

[49] A. Parravicini, F. Sgherzi, and M. D. Santambrogio, "A reduced-precision streaming spmv architecture for personalized pagerank on fpga," in Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASP-DAC), 2021, pp. 378–383.

[50] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishana, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in 2020 IEEE Internation Sumposium on High Performance Computer Architecture. IEEE, 2020, pp. 58–70.

[51] D. Ramchandani, B. Asgari, and H. Kim, "Spica: Exploring fpga optimizations to enable an efficient spmv implementation for computations at edge," in 2023 IEEE International Conference on Edge Computing and Communications (EDGE). IEEE, 2023, pp. 36–42.

[52] K. Rocki, D. Van Essendelft, I. Sharapov, R. Schreiber, M. Morrison, V. Kibardin, A. Portnoy, J. F. Dietiker, M. Syamlal, and M. James, "Fast stencil-code computation on a wafer-scale processor," in SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020, pp. 1–14.

[53] A. Rucker, M. Vilim, T. Zhao, Y. Zhang, R. Prabhakar, and K. Olukotun, "Capstan: A vector rda for sparsity," in 2021 Annural IEEE/ACM International Symposium on Microarchitecture. IEEE/ACM, 2021, pp. 1022–1035.

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

[55] Y. Saad, "The lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems," SIAM Journal on Numerical Analysis, vol. 19, no. 3, pp. 485–506, 1982.

[56] Y. Saad, Iterative methods for sparse linear systems. siam, 2003, vol. 82.

[57] Y. Saad, "Filtered conjugate residual-type algorithms with applications," SIAM Journal on Matrix Analysis and Applications, vol. 28, no. 3, pp. 845–870, 2006.

[58] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h, "A deflated version of the conjugate gradient algorithm," SIAM Journal on Scientific Computing, vol. 21, no. 5, pp. 1909–1926, 2000.

[59] F. Sadi, L. Fileggi, and F. Franchetti, "Algorithm and hardware co-optimized solution for large spmv problems," in 2017 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 2017, pp. 1–7.

[60] F. Sadi, J. Sweeney, T. M. Low, J. C. Hoe, L. Pileggi, and F. Franchetti, "Efficient spmv operation for large and highly sparse matrices using scalable multi-way merge parallelization," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2019, pp. 347–358.

[61] A. Samajdar, J. M. Joseph, M. Denton, and T. Krishna, "Airchitect: Learning custom architecture design and mapping space," arXiv preprint arXiv:2108.08295, 2021.

[62] A. Samajdar, E. Qin, M. Pellauer, and T. Krishna, "Self adaptive reconfigurable arrays (sara) learning flexible gemm accelerator configuration and mapping-space using ml," in Proceedings of the 59th ACM/IEEE Design Automation Conference, 2022, pp. 583–588.

[63] Z. Shi, Q. He, and Y. Liu, "Accelerating parallel jacobi method for matrix eigenvalue computation in doa estimation algorithm," IEEE Transactions on Vehicular Technology, vol. 69, no. 6, pp. 6275–6285, 2020.

[64] B. Sigurbergsson, T. Hogervorst, T. D. Qiu, and R. Nane, "Sparstition: a partitioning scheme for large-scale sparse matrix vector multiplication on fpga," in 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), vol. 2160. IEEE, 2019, pp. 51–58.

[65] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gómez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "Nero: A near high-bandwidth memory stencil accelerator for weather prediction modeling," in 2020 30th International Conference on Field-Programmable Logic and Applications (FPL). IEEE, 2020, pp. 9–17.

[66] G. L. Sleijpen and H. A. Van der Vorst, "A jacobi–davidson iteration method for linear eigenvalue problems," SIAM review, vol. 42, no. 2, pp. 267–293, 2000.

[67] C. Tan, C. Xie, T. Geng, A. Marquez, A. Tumeo, K. Barker, and A. Li, "Arena: Asynchronous reconfigurable accelerator ring to enable data-centric parallel computing," IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 12, pp. 2880–2892, 2021.

[68] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han, "Searching efficient 3d architectures with sparse point-voxel convolution," in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII. Springer, 2020, pp. 685–702.

[69] Top500. (2023) Top500 hpcg ranking. [Online]. Available: https://www.top500.org/lists/hpcg/2023/11/#:~:text=HPCG%20Release&text=This%20score%20is%20meant%20to,%2DPFlop%2Fs%20and%20No.

[70] H. M. Waidyasooriya and M. Hariyama, "Fpga-based deep-pipelined architecture for fdtd acceleration using opencl," in 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS). IEEE, 2016, pp. 1–6.

[71] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An analytical approach to sparse tensor accelerator modeling," in 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2022, pp. 1377–1395.

[72] Y. Xu, D. Li, Y. Xi, J. Lan, and T. Jiang, "An improved predictive controller on the fpga by hardware matrix inversion," IEEE Transactions on Industrial Electronics, vol. 65, no. 9, pp. 7395–7405, 2018.

[73] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, and Y. Shi, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in 2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE, 2020, pp. 1–6.

[74] X. I. Yang and R. Mittal, "Acceleration of the jacobi iterative method by factors exceeding 100 using scheduled relaxation," Journal of Computational Physics, vol. 274, pp. 695–708, 2014.

[75] W. You and C. Wu, "A reconfigurable accelerator for sparse convolutional neural networks," in Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, pp. 119–119.

[76] J. J. Zhang, N. B. Agostini, S. Song, C. Tan, A. Limaye, V. Amatya, J. Manzano, M. Minutoli, V. G. Castellana, A. Tumeo et al., "Towards automatic and agile ai/ml accelerator design with end-to-end synthesis," in 2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 2021, pp. 218–225.

[77] Z. Zhao, H. Kwon, S. Kuhar, W. Sheng, Z. Mao, and T. Krishna, "mrna: Enabling efficient mapping space exploration for a reconfiguration neural accelerator," in 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2019, pp. 282–292.

[78] H. Zheng, K. Wang, and A. Louri, "Adapt-noc: A flexible network-on-chip design for heterogeneous manycore architectures," in 2021 IEEE

international symposium on high-performance computer architecture (HPCA). IEEE, 2021, pp. 723–735.