Zoom Out: Abstractions for Efficient Radar Algorithms on COTS architectures

Tze Meng Low, Yuejie Chi, James Hoe, Swarun Kumar, Akarsh Prabhakara,
Laixi Shi, Upasana Sridhar, Nicholai Tukanov, Chengyue Wang, Yuchen Wu
Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA

2022 IEEE International Symposium on Phased Array Systems & Technology (PAST) | 978-1-6654-4166-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/PAST49659.2022.9975097

***Abstract*—The advent of machine learning has resulted in the rapid development of machine learning accelerators that are capable of computing tensor operations efficiently. Specifically, these accelerators compute matrix-matrix multiplication, a key routine in linear algebra libraries and machine learning. While using the accelerators would result in high performance radar signal processing, the algorithms used often require significant redesign in order to efficiently map them on to existing machine-learning hardware. In this paper, we show that higher levels of abstraction facilitate the efficient mapping of array algorithms onto commercial-off-the-shelf (COTS) machine learning hardware that results in higher performance in terms of execution time and/or throughput. Furthermore, similar levels of abstraction can be used to design efficient implementations of ML algorithms for radar processing, resulting in improved radar capabilities.**

***Index Terms*—Super-resolution, machine-learning, high performance computing, efficiency, performance**

I. Introduction

Phase array algorithms and machine learning models are often written using a high level language such as Matlab [1], Tensorflow [2] or Julia [3]. The interfaces of these high-level frameworks are often designed to resemble mathematical statements. This alignment between implementation and mathematical description enables scientists and developers to convert an algorithm to executable code with relative ease. Moreover, these frameworks capture the operations that need to be performed as opposed to capturing hardware implementation details; resulting in portable code that can run on different platforms. However, the performance attained with these frameworks are highly dependent on 1) the way the implementation is written, 2) the mapping from the input language to the high-performance library routines underlying these frameworks, and 3) the operations supported by the frameworks.

Very often, array algorithms are described as operations that are performed on individual steering vectors, array elements, and data frames. This level of description often means that each individual operation does not contain sufficient computation in order for them to be mapped efficiently on modern commercial-off-the-shelf (COTS) architecture such as CPUs, GPUs and FPGAs. As a result, implementations based on such description often results in poor utilization of the available compute resources leading to low performance such as long execution time and/or low throughput. COTS architectures with specialized hardware accelerators that target machine learning workloads require even more computation in order to achieve a high utilization of the compute resources. This means that the low level at which algorithms are described becomes an even bigger obstacle for attaining performance.

In this paper, we show that it is necessary to describe the desired computation at a higher level of abstraction than currently provided by existing languages/frameworks in order to efficiently map array workloads onto modern COTS architectures. Leveraging insights from multiple decades of high performance domain-specific library designs and program synthesis, we discuss extensions to the Tensorflow language that facilitate the description of array and ML computational pipelines. Specifically, using a classical array algorithm (MUSIC) and a machine learning for signal processing, we demonstrate how different techniques more commonly applied on scientific workload can provide significant speedup for array processing workload through the use of a higher level of abstraction.

This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) TRIAD program under Agreement No. HR00112190099. The content, views and conclusions presented in this document are those of the authors and do not necessarily reflect the position or the policy of the sponsoring agencies, or the U.S. Government.

II. Abstractions Needed For Performance

We motivate the need for higher level of abstractions using an example array processing workload, Direction of Arrival (DoA) estimation. Specifically, we use the Multiple Signal Classification algorithm, referred to as MUSIC [4] to compute the estimate. Below, we detail the computational steps in MUSIC, and present the insights that could be used given information at increasing levels of abstraction.

A. Overview of MUSIC

Mathematically, the MUSIC algorithm can be described with the following computational pipeline [5], [6]:

1) *Compute Correlation Matrix, R.* The first step of the MUSIC algorithm is to compute the correlation matrix, R based on the received radar signals.

$$R = \frac{1}{K}\sum_{i=1}^{K} x_i x_i^H, \tag{1}$$

where K is the number of antenna elements and x_i is the received signal vector.

2) *Finding Basis for Noise Subspace, Q.* The MUSIC algorithm makes the assumption that the noise subspace is orthogonal to the signal subspace. As such, the eigenvector of R is computed, and the vectors corresponding to the n smallest eigen-values are extracted and multiplied to form the noise subspace Q.

$$R \rightarrow [UU_n]\Lambda[UU_n]^H \tag{2}$$
$$Q = U_n U_n^H$$

3) *Finding Output Power Spectrum, $P(\phi)$.*

$$P(\phi) = \frac{1}{a^H(\phi)Qa(\phi)} \tag{3}$$

Notice from the above description that the computations of the correlation matrix R and the power at individual angles, $P(\phi)'s$, are described in terms of single array elements, and individual steering vectors. This low level of abstraction, when implemented as is, results in the use of many memory-bound routines which means that computational resources are often idle while data is moved from source to the compute resources.

B. Raising the level of abstractions

We discuss different levels of abstractions that are applicable to the MUSIC algorithm, and the different optimizations made available by the abstractions.

1) Beyond vector operations: A key observation made by the linear algebra community is that vector-based, memory-bound operations can be transformed into compute-bounded operations by casting them in terms of matrix-matrix multiplication. This insight underlies the shift from Level 1 and Level 2 Basic Linear Algebra Subprograms (BLAS) [7], [8] to the Level 3 BLAS [9]. Using the same approach, the computation of the correlation matrix can be reorganized into

$$R = \frac{1}{K} XX^H,$$

where X is a matrix created by collating the received vectors x_i's as columns of X in the following manner:

$$X = (x_0|x_1|\ldots|x_K).$$

We use the fact that R is the sum of the outer products, to reshape it into matrix multiplication, a Level 3 BLAS operation. While the number of floating point operations remain the same, the reorganization allow us to more efficiently hide the cost of moving data, while also utilizing existing compute resources more effectively. On modern COTS architectures with specialized matrix-matrix multiplication accelerators, the reformulation from vector to matrix operations also allows these specialized accelerators to be used.

The computation of the power spectrum can also be expressed in terms of matrix operations. First, we group all the steering vectors a_i together to form a steering matrix A.

$$A = (a_0|a_1|\ldots|a_K).$$

This steering matrix is then applied to the noise subspace.

$$P(\phi) = \frac{1}{diag(A^H Q A)}$$

Note the use of the the $diag(.)$ operator to pick out the correct elements of the application of the steering matrix.

2) Exploiting Matrix Properties: Once computations on vectors have been reorganized into computation on matrices, various matrix properties can be exploited to gain better performance. For instance, computing correlation matrices of various snapshots is a common operation in many array algorithms. This means that, in addition to recognizing that the computation is a matrix-matrix multiplication, we can exploit the observation that correlation matrix R is Hermitian. This means means that only (either the upper or lower) half the elements of R need to be computed. A similar observation can be made for computing Q. However, a drawback of this method of computation is that subsequent operations that use matrix R (or Q) needs to be cognizant of the fact only half the elements in the matrix are available. However, as we illustrate in the next subsection, information across different computation operations could be used to reduce the cost of computations.

3) Optimizing across computational blocks: Beyond optimizing within a computational block, raising the level of abstraction across computational blocks allows for even more optimization opportunities. We look back at the power spectrum calculation. The computation of Q performed in the second block in the MUSIC algorithm can be folded into Equation II-B1 as follows:

$$B = A^H U_n, \tag{4}$$
$$P(\phi) = \frac{1}{diag(BB^H)} \tag{5}$$

Notice that BB^H is a familiar pattern, and the symmetry available in the problem can be used to compute half the outputs. The diagonal elements are the final result of the power calculation.

Block diagrams for describing the MUSIC algorithm at different levels of abstraction are shown in Figure 1.

C. Performance Impacts of Different Levels of Abstractions

In Figure 2, we compared the performance of the MUSIC algorithm implemented by exploiting optimizations using different levels of abstraction and restructuring. We report performance as the number of radar array elements increases along the x-axis. We also show 2 different sampling sizes, 512 and 4096 samples. All 4 implementations presented are written using the Tensorflow framework and the SciPy [10] library as an interface to BLAS functions.

Performance is reported as speedup over a baseline implementation written as faithfully as possible using the mathematical operations previously described. Other implementations directly call specialized high performance library routines used by the Tensorflow framework.

The main takeaway from the plot is that as the level of abstraction increases, more aggressive optimizations can be performed, and thus faster implementations can be attained. Note that the eigen-decomposition function used in Tensorflow requires a full matrix as input. When the input sizes are

978-1-6654-4166-7/22/$31.00 ©2022 IEEE

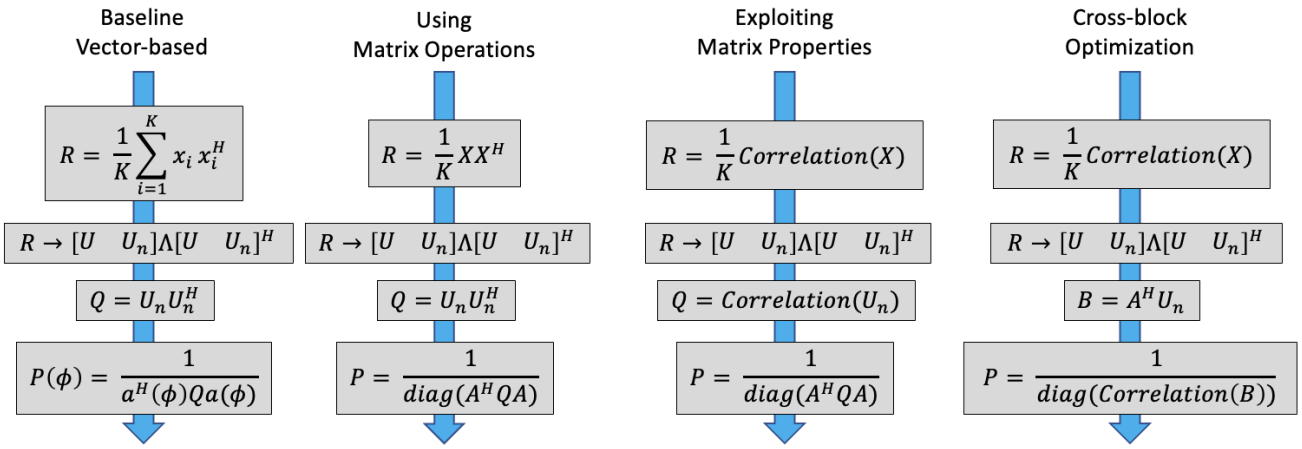

Fig. 1. Computation blocks that describe the MUSIC algorithm at different levels of abstractions. As we raise the level of abstraction (from left-to-right), we increase the amount of information that is encapsulated within each computational block. The information captured includes details (e.g. symmetry of that goes beyond the mathematics).

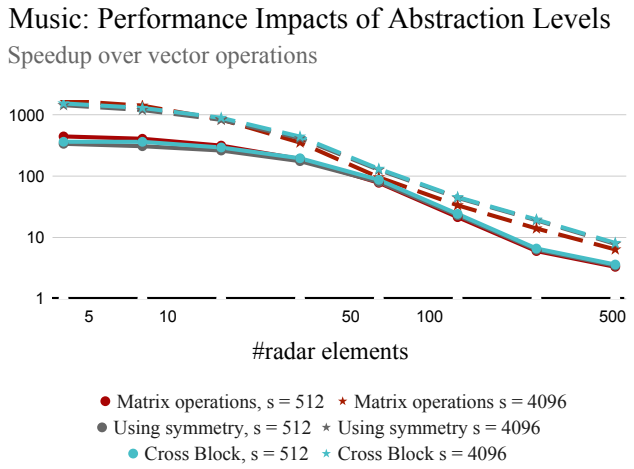

Fig. 2. Performance of MUSIC algorithm implementations for different radar sizes and sampling rates s. Implementations that use a higher level of abstraction perform better, especially for the radar arrays on the higher end of the spectrum.

small, the cost of making a triangular matrix (as part of exploiting symmetry) into a full matrix out-weighs the benefit of exploiting symmetry. As such, the implementation that exploits symmetry is less performant for small input sizes.

III. Abstraction for Intelligent Radar

In recent years, machine learning is increasingly being used in the context of phased array and signal processing algorithms [11]–[13]. In this section, we discuss abstractions that are required to improve performance of machine learning for radar processing algorithms on COTS architectures using the LISTA model to perform super-resolution to obtain higher angular accuracy.

A. Super-resolution with the LISTA Model

The problem of performing object detection can be formulated as a sparse coding problem using the LASSO formulation:

$$\min_x \frac{1}{2}\|y - Ax\|_2^2 + \|x\|_1, \tag{6}$$

where $y \in C^M$ denote the received signals from the antennas, $A \in C^{M \times N}$ is the specific sensing matrix such that $M < N$, $x \in C^N$ represents the object detection vector that is to be recovered.

An approach for solving Equation 6 is to use the Iterative Shrinkage-Thresholding Algorithm (ISTA) which computes the following expression at the k^{th} iteration:

$$x^{k+1} = \eta_{\frac{1}{L}}\left(x^k + \frac{1}{L}A^T(y - Ax^k)\right),$$

where η is the soft-thresholding function and L is the largest eigenvalue of $A^H A$. A drawback of using ISTA is that it can take a large number of iterations, and consequently a long time in order to get accurate results.

It has been recognized that individual iterations from an iterative approach can be turned into separate layers of a feed-forward network [14], [15]. This forms the basis of the LISTA model, and it has been shown that the LISTA model often requires a significantly smaller number of iterations to get the same accuracy as the ISTA approach. Here, the expression computed in each iteration of ISTA is translated into a single feed-forward layer that computes

$$x^{t+1} \leftarrow \eta_{\theta^t}\left(W_1 y + W_2 x^t\right), \qquad \forall t = 1, 2, \cdots, T, \tag{7}$$

leading to a T-layers feed-forward neural network. Here, (θ^t, W_1, W_2) denote the model parameters in the t-th layer, $\eta_\theta^t(\cdot)$ denotes the soft-threshold function with learned parameter θ^t, and x^t is the output of the t-th layer ($x^0 = 0$). $W_1 = \frac{1}{L}A^T$ and $W_2 = I - \frac{1}{L}A^H A$, are trained weight

978-1-6654-4166-7/22/$31.00 ©2022 IEEE

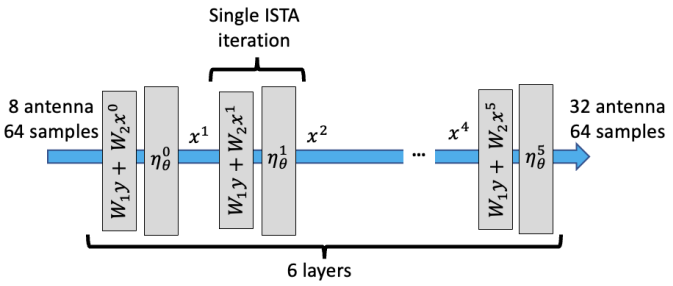

Fig. 3. Diagram of the LISTA model. For our discussion, we considered a model, consisting of 6 layers, that has been trained for an input of 8 antenna elements, 64 samples, and produces an output that is equivalent of 32 antenna elements, 64 samples.

matrices. A pictorial representation of the LISTA model is shown in Figure 3, and the performance of the LISTA model for different synthetic input data and comparison with other approaches is shown in Figure 4.

1) Adapting LISTA for array processing: Within the context of radar processing, the inputs to the LISTA model are the samples from the antenna elements. This requires adapting the LISTA model to handle complex inputs. However, machine learning with complex arithmetic is not a data format supported by commonly used machine learning frameworks such as Tensorflow[1]. A common approach taken by many machine learning frameworks is to perform complex arithmetic manually. This means that the real and imaginary components of the inputs and outputs are first separated, and then computed using separate real-arithmetic operations. For instance, for a complex matrix-vector multiplication $c = Ab$, the components are computed as

$$c_r = A_r b_r - A_i b_i$$
$$c_i = A_r b_i + A_i b_r$$

Where the variable subscripts r and i represent the real and imaginary components respectively. In the case of LISTA, the 2 matrix-vector products in Equation 7, are rewritten as 8 matrix-vector products of the corresponding components. We adopt this approach as our baseline LISTA implementations.

B. Raising the level of abstractions

1) Native Complex machine learning layers: While mathematically correct, implementing a complex operation (e.g. matrix-matrix multiplication) as multiple real routines is usually slower on modern COTS architecture. This is because there are often high performance complex arithmetic routines that can be leveraged for better performance. As such, we extended Tensorflow through their customization interface to create a specialized machine learning layer for computing with complex arithmetic. Following this extension, our implementation performs the 2 matrix-vector multiplication on complex values, in alignment with Equation 7.

[1]It should be noted that Tensorflow supports complex arithmetic, but the machine learning module in Tensorflow does not support machine learning with complex arithmetic.

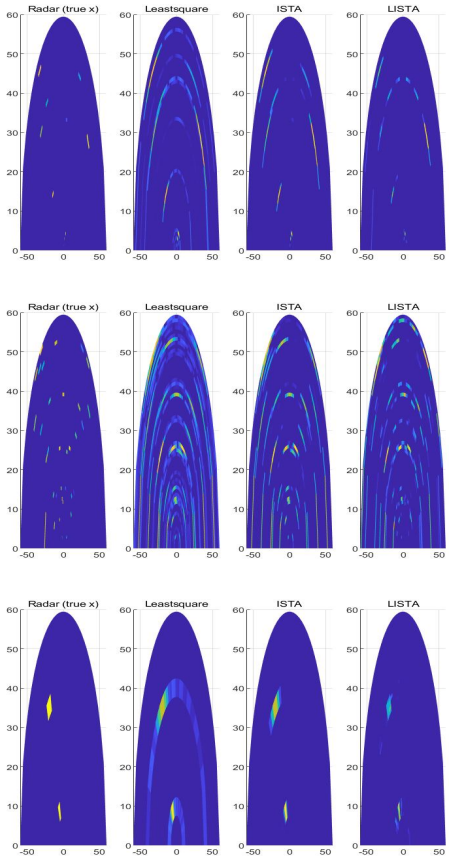

Fig. 4. Comparison of super resolution results for different methods using synthetic data created using different sparsity ratios. The ground truth data (left-most) for the top and middle row are generated assuming 32 antenna elements where 0.5% and 2% of the values are non-zeros. The bottom row has a sparsity of 1% but the points are generated in clusters/blocks. The ground truth was then sub-sampled to simulate an 8 antenna elements input. In all cases, the LISTA model was able to recover original the radar signal more accurately than the two other approaches.

2) Beyond vector-operations: The LISTA model in the previous subsection describes the computation on a single frame worth of radar data, where the entire frame is treated as a single vector y in Equation 7. Similar to the optimization for the MUSIC algorithm, we can group multiple frames worth of data and cast the matrix-vector operations into matrix-matrix operations, i.e.

$$X^{t+1} \leftarrow \eta_{\theta^t}(W_1 Y + W_2 X^t), \forall t = 1, 2, ..., T,$$

where the columns of Y are received signals for a data frame.

3) Optimizing across computational blocks: Equation 7 gives us an end-to-end picture of the computation of the LISTA model. Note that the intermediate result, $W_1 y$, is the same for all layers from 1 to T. We recognize this reuse of the intermediate value and reorganize the computation as:

$$\begin{aligned} z &= W_1 y \\ x^{t+1} &\leftarrow \eta_{\theta^t}\left(z + W_2 x^t\right), \qquad \forall t = 1, 2, \cdots, T, \end{aligned}$$

Using this method, we only have to compute z once.

978-1-6654-4166-7/22/$31.00 ©2022 IEEE

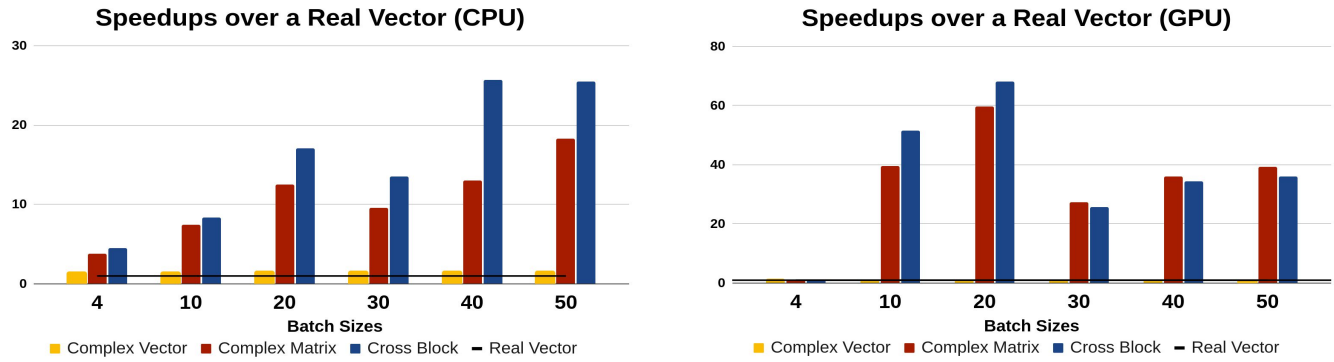

Fig. 5. Performance comparisons of different implementations of the LISTA model with 2x angular resolution. The baseline uses real vector computations to emulate complex. The 3 levels of abstraction we show are, 1) complex vector operations per frame, 2) complex matrix operations over the entire batch and 3) complex matrices with a reused intermediate output. For the CPU implementations on the left, each level of abstraction yields progressively higher speedup over the baseline. The GPU implementations on the right show a similar trend, however, for larger sizes, performing the redundant computation is faster than reusing intermediates

C. LISTA Performance with different levels of abstraction

Figure 5 shows the impact of designing LISTA implementations given information at different levels of abstraction. All our implementations are written in C++ because Tensorflow's machine learning framework does not easily support complex operations. It should be noted that our baseline code is our best attempt at staying faithful to the interfaces Tensorflow offers. This captures the impact of the low-level of abstraction, while avoiding overheads due to Python's runtime. Performance is reported as speedup over the baseline implementation that implements LISTA using only real arithmetic computations (labelled Real Vector). Each frame of radar data is 8×64, or 512 elements. We vary the number of batches to see the impact of grouping them together as matrix-matrix multiplication.

1) CPU performance: The CPU results are obtained on an Intel Rocket Lake Core i9-11900K. Switching to 2 native complex matrix-vector multiplications instead of using 8 real arithmetic multiplications yielded a small benefit over the baseline implementation. The bulk of the performance improvement is attained by computing the LISTA model for multiple data frames simultaneously. Even for a small number of frames (4), a speedup of 3.7x, and up to 18.2x for a larger number of frames (50) can be attained. By avoiding re-computing $W_1 Y$ we gain a small benefit when the number of frames is small but as the number of frames increases, the performance benefit of this optimization increases significantly.

2) GPU performance: The GPU implementations were run on an AMD Vega 64. We use the same 3 styles of implementation as the CPU. Using a native complex matrix-vector multiplication is almost the same as using the emulated real matrix-vector multiplication. This may be because the size of each frame (512 elements) is too small to utilize the GPU well. In fact, when the batch size is 4, even grouping the batches together is not enough to efficiently use the GPU. For this case, all 4 implementations are nearly identical.

For the larger batch sizes greater than 4, casting the computation as matrix operations is highly effective, attaining a speedup between 27 and 59 times over the real matrix-vector implementation. The optimization to avoid re-computation reduced the number of operations performed in each computational block. For batch sizes 10 and 20, this results in a higher speedup (up to 68x over the real matrix-vector). However, for batch sizes 30 - 50, avoiding re-computation is up to 8% slower than the complex matrix-matrix implementation that performs the re-computations. This may be due to the reduced number of operations per computational block. Such problem-specific trade-offs are enabled by higher level abstractions and warrant further study.

IV. Related Work

In this section, we review related works for improving the performance from high level abstractions.

A. Domain Specific Languages and Frameworks

Domain specific languages such as Tensorflow [16], Numpy [17] and SciPy [10] encapsulate common functionalities used in particular domains. These high level descriptions of operations allow high-level developers to be more productive by giving them the ability to program in terms they are very familiar with. However, the level of abstraction at which domain specific languages are described may make it difficult for a lower level library or compiler to produce performance code. This work highlights that in order to get performance improvements, it is necessary to go above and beyond the natural way by which computation in the domain is described. For example, batching of multiple vector operations into operations involved matrices is often not performed. This is because these optimization opportunities are often left for developers, who are domain experts but non-performance experts, to identify and exploit.

B. High Performance Libraries

High performance libraries that support commonly used interfaces such as the Basic Linear Algebra Subprograms (BLAS) and Linear Alegbra package (LAPACK) [18] provide

978-1-6654-4166-7/22/$31.00 ©2022 IEEE

efficient implementations for specific architectures. However, these often rely on the developer being well-versed with specific routines within these libraries. Performance is reliant on the choice of the appropriate routine. While some high performance libraries are used in many other domain-specific frameworks, the mappings from domain specific routines to library routines often do not exploit specialized routines in the libraries.

V. Conclusion

In this paper, we discussed and demonstrated that raising the level of abstraction is necessary in order to efficiently map array algorithms onto commercial-off-the-shelf (COTS) architectures. This higher level of abstraction is needed to 1) increase the number of computations per computational block, 2) capture numerical properties of the matrices/tensors that can be utilized for performance optimization, and 3) perform higher level optimizations that cut across traditional computational blocks.

Through two array algorithms (one classical, and one ML-based), we demonstrate the benefits of using higher level descriptions. We show that algorithms implemented using a higher level of abstraction often run significantly faster than baseline implementations written faithfully based on the mathematical description of the algorithm. In addition to higher performance, a higher level of abstraction provides the flexibility of changing the underlying implementation to best address the problem size and COTS architecture at hand.

The abstractions and optimizations presented are not specific to the algorithms presented in this paper. Many of the abstractions (e.g. correlation, and applying of weights to steering vectors) are also used in radar algorithms such as the Space-Time Adaptive Processing [19]. This suggests that a high-level language based on the presented abstractions could be designed, and the optimizations enabled by the abstractions could be automatically applied using a domain-specific compiler build using the Multi-Level intermediate language (MLIR) [20] framework.

References

[1] C. Moler, J. Little, and S. Bangert, *Pro-Matlab, User's Guide*, The Mathworks, Inc., 1987.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[3] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman, "Julia: A fast dynamic language for technical computing," *arXiv preprint arXiv:1209.5145*, 2012.

[4] R. Schmidt, "Multiple emitter location and signal parameter estimation," *IEEE transactions on antennas and propagation*, vol. 34, no. 3, pp. 276–280, 1986.

[5] N. A. Baig and M. B. Malik, "Comparison of direction of arrival (doa) estimation techniques for closely spaced targets," *International journal of future computer and communication*, vol. 2, no. 6, p. 654, 2013.

[6] E. Gentilho, P. R. Scalassara, and T. Abrão, "Direction-of-arrival estimation methods: A performance-complexity tradeoff perspective," *J. Signal Process. Syst.*, vol. 92, no. 2, p. 239–256, feb 2020. [Online]. Available: https://doi.org/10.1007/s11265-019-01467-4

[7] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for Fortran usage," *ACM Trans. Math. Soft.*, vol. 5, no. 3, pp. 308–323, Sept. 1979.

[8] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, "An extended set of FORTRAN basic linear algebra subprograms," *ACM Trans. Math. Soft.*, vol. 14, no. 1, pp. 1–17, March 1988.

[9] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, "A set of level 3 basic linear algebra subprograms," *ACM Trans. Math. Soft.*, vol. 16, no. 1, pp. 1–17, March 1990.

[10] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[11] W. Liu, "Super resolution doa estimation based on deep neural network," *Scientific Reports*, vol. 10, no. 1, p. 19859, 2020. [Online]. Available: https://doi.org/10.1038/s41598-020-76608-y

[12] S. Chakrabarty and E. A. Habets, "Broadband doa estimation using convolutional neural networks trained with noise signals," in *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2017, pp. 136–140.

[13] P. Lang, X. Fu, M. Martorella, J. Dong, R. Qin, X. Meng, and M. Xie, "A comprehensive survey of machine learning applied to radar signal processing," *arXiv preprint arXiv:2009.13702*, 2020.

[14] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proceedings of the 27th international conference on international conference on machine learning*, 2010, pp. 399–406.

[15] P. Sprechmann, A. M. Bronstein, and G. Sapiro, "Learning efficient sparse and low rank models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1821–1833, 2015.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[18] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, "LAPACK: A portable linear algebra library for high-performance computers," in *Proceedings Supercomputing '90*. Los Alamitos, California: IEEE Computer Society Press, 1990, pp. 2–11.

[19] J. Ward, "Space-time adaptive processing for airborne radar," in *1995 International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, 1995, pp. 2809–2812 vol.5.

[20] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14.

978-1-6654-4166-7/22/$31.00 ©2022 IEEE