Dynamic Prediction of Parallel CA-CFAR Processing Performance on Large Datasets

Marcello G. Costa, Renato Machado and Fabio M. Bayer

Abstract— **The constant false alarm rate (CFAR) detection theory maintains good decision performance over statistical interference. For imagery applications, this class of detectors increases the processing complexity due to the element-wise matrix multiplication performed for each pixel resolution. Hence, the largest collected dataset implies computational float-point operations (FLOPs) with squared growth order, which end up compromising its performance under real-time requirements and power consumption constraints. To make CFAR processing a workable solution for general-purpose devices, enabling it for several emergent embedded technologies, this article presents a dynamic random forest regression to predict the parallel processing performance on cell-average CFAR. In this method, the distributed data parallelism performance through CPU and GPU cores can be dynamically predicted over time, enabling evaluation of convergence to higher energy efficiency and lower runtime processing. As a result, for large FLOPs demand over an energy budget, the obtained predicted discharge consumption slopes over processing runtime demonstrate higher efficiency on GPU cores and their remaining capacity for processing and consumption.**

Keywords— **CA-CFAR, runtime processing, energy saving, parallel processing, dynamic prediction.**

I. Introduction

Matrix multiplication with non-trivial implementation achieves the time complexity of $\mathcal{O}(n^{\log_2 7})$ with the classical Strassen algorithm [1]. In computational systems, the effect of this complexity can be measured in float-point operations (FLOPs), which consists of the number of multiplications and summations required in matrix-vector operations for the input data-set size (n) [2]-[3]. From the last decade, high-performance computing demands for machine learning and data science have stimulated efforts in the development of optimization kernels for algebra libraries like the Basic Linear Algebra Subprograms (BLAS) and the Linear Algebra PACKage (LAPACK), to increase their computational speed [4]. At the same time, improvements in the hardware parallelism architectures, based on multi-core central processing unit (CPU) and graphics processing unit (GPU) have determined the demand for power computation in software-based applications running in General Purpose devices, enabling real-time and low-consumption requirements.

Particularly, in signal processing applications, element-wise matrix multiplication (Hadamard product) is a common operator for image filtering, where each item or cell in a matrix is multiplied point-to-point with cells of another matrix to produce an output matrix of the same size. In spite of lower complexity, $\mathcal{O}(n^2)$, compared to matrix-matrix products and matrix inversion operations, element-wise matrix multiplication employed in image processing usually involves very large input dimensions, requiring a design of efficient and low-complexity algorithms to process the required number of floating-point operations. The constant false alarm ratio (CFAR) test in radar target detection is a high-parallelized application also subject to optimizations, on which adaptive processing maintains the optimal detection performance. For homogeneous clutter statistics, the cell-average CFAR (CA-CFAR), defined in [5], presents the simplest solution to decide on target presence, involving element-wise matrix multiplication computed on a kernel of data by pixels following for fewer cost operations like sums and comparisons.

In such processing, higher-resolution acquisition sensors provide complexity growth and make this problem difficult to treat under computational restrictions. Some authors have explored the reduction of processing operations [6] but with the reduction in the parallelism power in hardware implementation. Strictly considering the hardware implementation for parallel processing, [7], [8], [9], and [10] show solutions to improve the runtime and the power consumption as the most relevant point in comparative performances. However, the available studies do not introduce any predictive model for hardware efforts over a period of time, where the processing and power demands may suffer variations. In this sense, time series forecasting is a dynamic prediction method to anticipate the system's performance analysis. The class of well-experimented autoregressive methods, such as autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) is widely applied [11], but such approaches are not effective under large datasets or robust to outliers of weak signal measurements. Such restrictions can be overcome by dynamic regression models based on decision trees, such as random forests (RF) [12], where efficient solutions for accuracy and computational complexity for time series have been obtained [13]-[14].

Under these circumstances, in this work, the parallelism for CA-CFAR filtering includes arithmetic throughput and energy consumption as a restriction. A time series forecasting is built in a dynamical random forest regression, where the training

Marcello G. Costa, Laboratório de Ciências Espaciais de Santa Maria (LACESM), UFSM/INPE, Santa Maria-RS, Brazil, e-mail: marcello.costa.br@ieee.org; Renato Machado, Departamento de Engenharia Eletrônica e Computação/Telecomunicações, Instituto Tecnológico de Aeronautica (ITA), São José dos Campos-SP, Brazil, and Fabio M. Bayer, Departamento de Estatística and LACESM, Universidade Federal de Santa Maria (UFSM), Santa Maria-RS, Brazil, e-mail: bayer@ufsm.br. This work was partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq, grant number 150158/2023-8) and by the Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS, grant number 23/2551-0000121-7), Brazil.

and prediction steps are dynamically adjusted to ensure a high level of accuracy in online operation. The main contributions of this article are: (i) computational complexity and power consumption time series prediction for CFAR processing with parallelism; (ii) trade-off analysis for distributed data parallelism in the growth of element-wise matrix multiplication on GPU and general-purpose CPU; and (iii) dynamic updating method on statistical learning for minimum MSE.

The remainder of this article is organized as follows. Section II presents the optimal detection based on CA-CFAR and its complexity. In Section III, a parallel approach to computing the matrices multiplication on CA-CFAR is adjusted into a predictive dynamic model. Section IV shows experimental results with specific CPU and GPU systems. Finally, Section V presents the conclusions.

II. CA-CFAR Algorithm and Complexity Analysis

The target presence in a homogeneous background can be detected by using a hypothesis test, such that the background statistics are modified under the alternative hypothesis $\mathcal{H}_1$, and the null hypothesis $\mathcal{H}_0$ determines the observations y as part of the background distribution. For this purpose, a widely used detector is the likelihood ratio (LR) test [15], where the LR statistics are tested as

$$\Lambda(y) = \frac{p(y|H_1)}{p(y|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \gamma, \tag{1}$$

where the threshold γ is a constant value from the null distribution of $\Lambda(y)$, $p(y|H_1)$ and $p(y|H_0)$ are the probability density functions of y under H_1 and H_0, respectively.

The detection performance in the LR test depends on the level of signal separation achieved (under H_1), influenced by the background model approximation. Usually, homogeneous clutter is characterized by Gaussian, exponential, log-normal, $\mathcal{K}$, gamma, and $\mathcal{G}_A^0$ distributions [16]. In addition, the CFAR filtering provides the threshold adaptation, γ_k, for expected local interference through the dataset size $K \in \{1, \ldots, k\}$. Therefore, the maximum probability of detection (P_D) for a constant probability of a false alarm (P_{FA}) is ensured. In the CA-CFAR algorithm, this interference is homogeneous, and its power estimation is computed in terms of the average of the surroundings test cell and a multiplier α for false alarm probability that attenuates or amplifies the detection threshold [17], i.e.,

$$\alpha = N_C(P_{FA}^{-\frac{1}{N_C}} - 1), \tag{2}$$

where N_C is the number of neighboring cells.

The CFAR property is established by maintaining a constant false alarm according to the N_C. Fig. 1 shows the CA-CFAR structure, where the unit to be tested is the Cell Under Test (CUT), the cells used for estimation of the interference are Reference Cells (RC), and the cells between the CUT and the RC are the Guard Cells (GC). CFAR window shifts through the range-Doppler data and tests each cell for a target. A target is detected if the signal strength of CUT is greater than clutter strength.

The computational complexity is summarized for a data matrix having N_R and N_D cells along the range and Doppler axes, respectively, where the naive technique is given by $\mathcal{O}(N_R N_D N_{RC} N_{GC})$. In terms of FLOPs, there is one element-wise product $N_{RC} \odot N_{GC}$ for each $N_R N_D$ pixels, resulting in $N_R N_D \left(\frac{1}{2}N^2 + \frac{1}{2}N\right)$ FLOPs [4], [6], where N is due to the equal dimension size of the reference cells N_{RC} and guard cells N_{GC}, that depends of the target resolution.

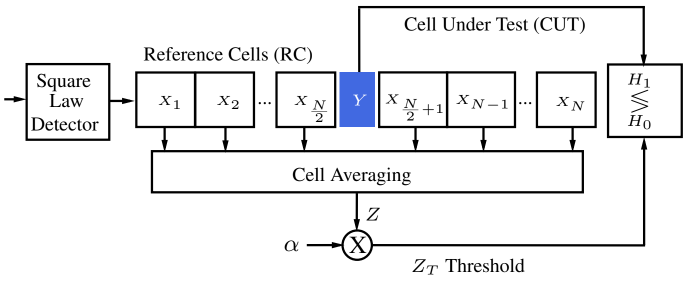


Fig. 1: The flow of a typical 2D CA-CFAR algorithm.

III. An Energy-Efficient Parallelized Approach

The high runtime processing in CA-CFAR is due to element-wise matrix multiplication to compute the average background statistics. An effective filtering performance over the naive product depends on the parallelism in hardware. Since then, the conventional serial computation across the CPU cores is strongly dependent on power computation and memory access with current technology. In parallel processing the computation is distributed to simultaneous execution by a set of cores, reducing drastically the runtime, when the computations are potentially parallelized.

A. Distributed Data Parallel

Among existing solutions for parallelism, distributed data-parallel (DDP) is a dominant strategy due to its minimally intrusive nature and ability to speed up some processing on large datasets. This technique replicates the model on every computational resource to generate gradients independently and then communicates those gradients at each iteration to keep model replicas consistent [4]. `PyTorch` data parallelism is a package distribution that offers several tools to facilitate distributed processing, including the `DistributedDataParallel` module, being appropriated to start sub-tasks in distributed cores at the same time [18]. The sub-tasks are managed by `AllReduce` operation, which collectively applies a given arithmetic operation to input tensors (input data) from all processes and returns the same result tensor to each participant. The number of processes for distribution through the cores includes an overhead, which for small datasets the parallel operation cannot be efficient. However, for large datasets, this overhead is negligible.

In terms of the computational performance, the computations and memory accesses are bounded as a function of arithmetic intensity in FLOPs/bytes combined with the arithmetic throughput, in FLOPs per second (FLOPS), which describes the expected runtime performance and system occupancy demands for parallelism structure. Note that the involved power consumption is a hardware-dependent variable of system architecture, where parallelism has a strong influence.

B. Dynamic Model for Energy Prediction

Dynamic predictions are usually built into time series to forecast future values over a period of time. Large datasets, as provided for high sampling data, and weak signal measurements, which are subject to outliers and non-linear relationships, represent a limitation for well-experimented time series forecasting methods. An attractive time series modeling has been implemented on RF learning method [13], being robust to outlying observations and more efficient with respect to computational complexity, than other techniques.

RF is an ensemble of unpruned classification or regression trees created using bootstrap samples of the training data and random feature selection in the decision tree induction. Prediction is done by aggregation with a weighted vote (in classification) or weighted average (in regression) of the ensemble outputs. The method builds an extensive collection of de-correlated and randomized regression trees leading to a dissimilarity measure between the observations [19]. Suppose that we have a sequence $\{\mathbf{X}_t, Y_t\}_{t\in\mathbb{Z}}$, where $Y_t \in \mathbb{R}$ is the response (target) and $\mathbf{X}_t \in \mathbb{R}^p$ is a set of p regressors (features), such that

$$Y_t = f(\mathbf{X}_t) + \epsilon_t, \tag{3}$$

where ϵ_t is stochastic error such that $\mathbb{E}[\epsilon_t] = 0, \forall t$. The purpose of random forests is to estimate, by only observing a dataset $\mathcal{D}_n = ((\mathbf{X}_1, Y_1)), \ldots, (\mathbf{X}_n, Y_n)$, the regression function

$$f(\mathbf{x}_t) = \mathbb{E}[Y_t|\mathbf{X}_t = \mathbf{x}_t], \forall \mathbf{x}_t \in \mathbb{R}^p. \tag{4}$$

The regression involves the following recursion: (1) From the training data, draw randomly a number of observations with bootstrap per tree with or without replacement; (2) For each bootstrap sample, grow a tree with the best split among a randomly selected subset of m_{try} descriptors. The tree is grown to the maximum size; (3) Repeat the above steps until a sufficiently large number of outputs. When $m_{\text{try}} = p$, i.e., the best split at each node is selected among all descriptors.

The adaptation of the RF for the time series is defined through the autoregressive model over the regression structures in the following way

$$Y_t = f(\mathbf{X}_t, Y_{t-1}, Y_{t-2}, \ldots, Y_{t-p}) + \epsilon_t, \tag{5}$$

where the arbitrary time step t is determined by using a combination of the p last observations. The simplest method for time series RF is based on non-overlapping block bootstrap (NBB), where the dataset is transformed into supervised learning using this sliding-window representation. In this method, the bootstrap set $\mathcal{D}_n^*$ is then obtained by drawing K blocks, $(B_1^*, \ldots, B_k^*)$, uniformly with replacement in the collection of non-overlapping blocks $(B_b)_{1\leq b\leq B}$ for a suitably chosen k, where ℓ_n block size and $B \geq 1$ the greatest integer such that $\ell_n B \leq n$.

$$B_b = (Y_{(b-1)\ell_n+1}, \ldots, Y_{b\ell_n}), \tag{6}$$

where $b = 1, \ldots, B$.

In the proposed model for energy prediction, the set of observable variables is the combined hardware efforts based on the power consumption, in Watts (W), and the arithmetic throughput, in FLOPS, demanded by the CPU or GPU. The parallelism distribution determines the costs of energy and runtime to perform all the required FLOPs for CFAR processing. The continuous hardware measurements feed the time series predictor, where the RF structure is split into training and testing data for each region separately, reproducing the NBB method. This modeling is extended to adjust the region size for the range of accurate prediction, as defined as follows:

Proposition 1 (Block Adaptive Random Forest time series): Let a response variable $Y = (Y_1, \ldots, Y_n)$ be split in K consecutive blocks of length ℓ. Then, we can rewrite $Y = (B_1, \ldots, B_k)$ with $B_b = (Y_{(b-1)\ell+1}, \ldots, Y_{b\ell})$ non-overlapping blocks, being each block subject to bootstrap set $\mathcal{D}_n^*$. If ℓ is separated in training length, ℓ_{tr}, and testing length, ℓ_{ts}, we can obtain the prediction $\hat{Y}$ for the next $(Y_{(b-1)\ell_{tr}+1}, \ldots, Y_{b\ell_{tr}})$ samples. For each training sliding-window, the block length ℓ_{tr} and ℓ_{ts} are determined under the prediction performance, i.e,

$$\overbrace{(Y_{(k-1)\ell+1}, \ldots, Y_{k\ell})}^{\ell=\ell_{tr}} = \hat{Y} \approx \overbrace{(Y_{(k-1)\ell+1}, \ldots, Y_{k\ell})}^{\ell=\ell_{ts}} = Y \tag{7}$$

such that

$$\begin{cases} \ell_{tr}^0 \to \text{(initial block length)} & \text{if } (Y-\hat{Y})^2 \leq \delta \\ \ell_{tr}^* \to \text{(block length increment)} & \text{otherwise.} \end{cases} . \tag{7a}$$

Given the initial training block length ℓ_{tr}^0, the time series prediction dynamically draws the block length for training and testing according to the previous performance rule, where a predefined mean square error (MSE) is established with δ. $\square$

The block diagram in Fig. 2 extends the Proposition 1 to predict the energy consumption with different distributed parallelism processes. The details for CA-CFAR processing demands and its dynamic operation are described as follows.

1) The DDP selects the core elements to break the input CA-CFAR matrix product processing. Then, the time series forecasting is fed with the hardware measurements, where the power consumption, in terms of battery life, is evaluated in the runtime to process all the FLOPs demand.
2) By considering an energy budget, the initial block length setup ℓ_{tr} and ℓ_{ts} are adjusted under reduced battery discharge (in percentage), which depends on the parallelism setup and processing demand. Therefore, the collected data in this interval are used to predict the next discharge window.
    - If the predicted energy $\hat{E}_t$ compared to the update E_t observations obeys the MSE criterion, the initial block length is kept;
    - Otherwise, the NBB is balanced by increasing the training length, ℓ_{tr}.

The online prediction advances through the incoming measurement samples for a relatively large dataset, where the remaining energy budget decays in a minimum state of charge. In such a prediction, parallelism efficiency can be evaluated

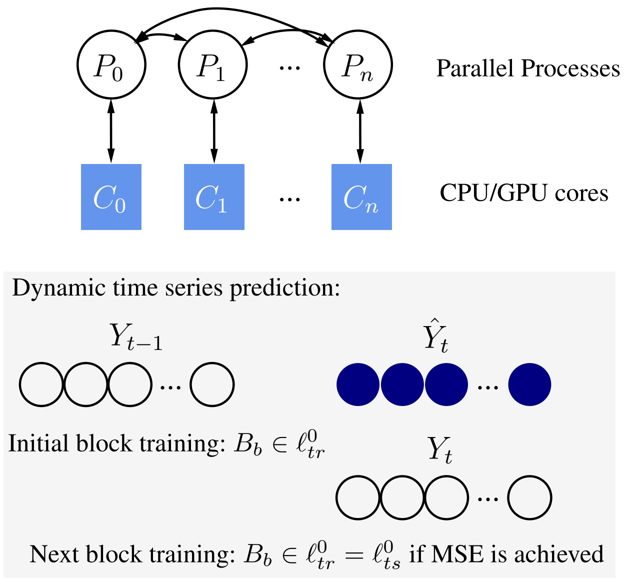

P_0
P_1
P_n
Parallel Processes
C_0
C_1
C_n
CPU/GPU cores
Dynamic time series prediction:
Y_{t-1}
$\hat{Y}_t$
Initial block training: $B_b \in \ell^0_{tr}$
Y_t
Next block training: $B_b \in \ell^0_{tr} = \ell^0_{ts}$ if MSE is achieved

Fig. 2: Distributed parallel processing for multicores and the dynamic RF regression prediction.

quantitatively to extend the battery life under the best computational performance.

IV. Experimental Results

To provide an effective CA-CFAR implementation on general-purpose devices, the computation of element-wise matrices multiplication used in such filtering process was parallelized with `Pytorch` package, where parallel sub-processes are applied to distribute the overall computation. Firstly, we employed the parallel approach through multi-core CPU[1], then in a graphic card GPU[2]. The hardware performance measurements are performed using the Linux-based monitor tools `htop`, `powerstat`, and `nvidia-smi perf` in which the collected 1 second samples of CPU/GPU cores usage, memory access, and power consumption are made available for prediction. In the experimental campaigns, we selected different numbers of CPU threads $(1, 12, 24)$ to verify the achieved runtime and the consumption, as the input dataset was made large. This experiment was replied to through the dedicated $(64, 512)$ GPU cores, where the comparative parallelism performance was evaluated.

For a given energy budget[3], we can build a time series prediction subject to available cores in parallel processing to estimate the remaining battery charge over the FLOPS throughput. Then, the time series in Proposition 1 is used for forecasting the collected energy consumption and FLOPS as a function of the number of cores in parallelism. In the static prediction on RF, using all dataset, the FLOPS is selected as

[1]The CPU is based on a high-performance processor Intel Core i9-12900KF, 16-Core, 24-threads, with the clock of 3.2 GHz and 32 GB DDR5 of memory

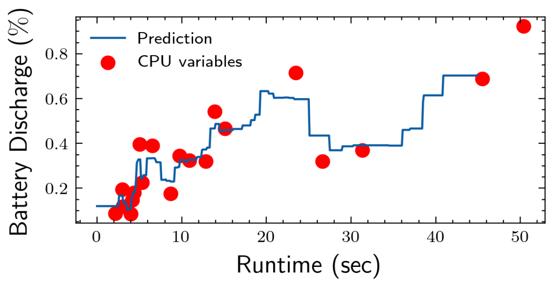
[2]Graphic card GPU NVIDIA GeForce GTX 1630, 512 shading units (cores), 32 texture mapping units, and 16 ROPs, with the clock of 1740 MHz and 4 GB GDDR6 of memory

[3]For reference we consider a Lithium-ion battery for a laptop, model H-HT03XL with 3615 mAh (41.7 Wh) and 11.55 V.

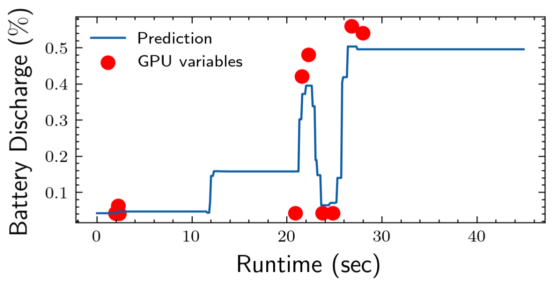
TABLE I: Random Forest Regression performance by input features over time.

Input features over time	*Random Forest Regression*		
	R-Squared	MSE	MAE
CPU Cores, FLOPs	0.9978	87.2629	6.2015
Battery (%), CPU cores	0.9852	52.2315	5.1524
Battery (%), CPU cores, FLOPs	0.9437	104.3046	7.5771
GPU Cores, FLOPs	1	20.4178	4.5062
Battery (%), GPU cores	1	11.2621	3.3171
Battery (%), GPU cores, FLOPs	1	11.4413	3.2661

Note: We present the performance for runtime prediction in RF regression by different features input in the reduced energy budget evaluation.



(a)



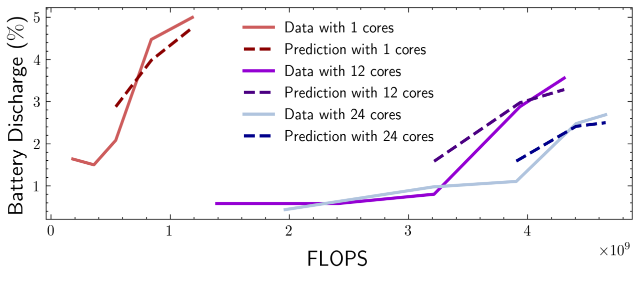
(b)

Fig. 3: Random Forest Regression performance using all related features over time: (a)using CPU cores, and (b) using GPU cores.

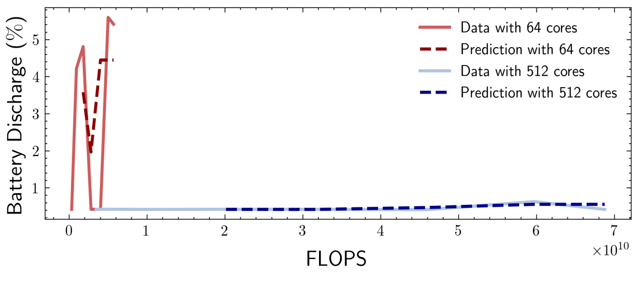
the time reference and target variable, and the features list includes the battery discharge and cores amount (for CPU and GPU technologies). The results of regression in terms of R-Squared (R^2), MSE, and mean absolute error (MAE) using the function `RandomForest` in R language is summarized in Table I illustrated in Fig. 3.

For the time series approach, the initial training step considers the first 5% of battery discharge over minimum parallelism, where the length of blocks for training and testing are adjusted for the target MSE. In such a situation, we can predict the energy consumption for the next 3% of discharge from actual 1%. By moving to the next prediction block, we can dynamically predict the energy consumption for a wider range of discharge percentages, since the MSE criterion is obeyed. This process is continuously repeated to achieve the limit for FLOPs demand in CA-CFAR matrix multiplication or the battery discharge is full.

The results presented in Fig. 4a give us a separate analysis



(a) DDP with CPU



(b) DDP with GPU

Fig. 4: Prediction of parallelism performance of CA-CFAR FLOPS with dynamic time series through 5% of battery discharge.

of parallelism by CPU threads, where the limits of $N_r N_d$ pixels for $N \times N$ matrix element-wise multiplications are computed. As observed, if more cores are employed in the parallel processing, the convergence to the energy efficiency is achieved and the largest dataset can be processed by CA-CFAR in the shortest runtime. In Fig. 4b, the prediction was extended to GPU cores, where the parallel robustness was achieved through the 512 cores. In this scenario, the power to process the matrices at lower energy consumption was $8\times$ better than the maximum threads in the CPU system. Furthermore, the dedicated operations performed in GPU with multiple cores are efficient in repeating the $N \times N$ multiplications. Since the $N \times N$ matrices depend on the target size, typically its dimension is tractable for GPU cores, here we considered $N = 30$.

The discharge slops shown in Fig. 4 are dynamic predictions that use online hardware measurement samples under reduced budget space, the throughput FLOPS jointly with the discharge of the battery is able to forecast the remaining discharge progress, given a specified prediction error over the demands for CFAR processing. As expected, the longer prediction space can be adjusted as more training elements become available.

V. Conclusions

The squared growth of complexity on element-wise matrix multiplication in CA-CFAR processing could be effectively performed under a distributed data-parallel approach. The FLOPs demand by increasing dataset input was balanced over CPU and GPU multicores architectures, achieving effective runtime and lower energy costs. Under parallelism, demanded FLOPS for execution and their energy consumption were effectively predicted by dynamic RF regression, where the large input dataset or outliers were not limitations. In this prediction, the expected hardware efforts performance is a dynamic process that determines the maximum dataset size that the parallelism structure supports to perform over an energy budget. The tests performed with the time series predictor under a sample of battery discharge (5%), 256 GPU cores were $8\times$ most effective than high-performance CPU 24 cores, where 3% were predicted at lower MSE. This prediction can be extended to estimate the remaining capacity for processing, given a dataset demand.

References

[1] V. STRASSEN, “Gaussian elimination is not optimal.,” *Numerische Mathematik*, vol. 13, pp. 354–356, 1969.

[2] P. J. Freire, S. Srivallapanondh, A. Napoli, J. E. Prilepsky, and S. K. Turitsyn, “Computational complexity evaluation of neural network applications in signal processing,” *arXiv:2206.12191 [cs]*, jun. 2022.

[3] D. Watkins, *Fundamentals of Matrix Computations*. Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts, Wiley, 2004.

[4] A. Sankaran, N. Alashti, C. Psarras, and P. Bientinesi, “Benchmarking the linear algebra awareness of tensorflow and pytorch,” in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 924–933, IEEE Computer Society, 2022.

[5] H. M. Finn, “Adaptive detection mode with threshold control as a function of spatially sampled clutter level estimates,” *RCA Rev*, vol. 29, pp. 414–465, 1968.

[6] G. G. Acosta and S. A. Villar, “Accumulated CA–CFAR process in 2-D for online object detection from sidescan sonar data,” *IEEE Journal of Oceanic Engineering*, vol. 40, no. 3, pp. 558–569, 2015.

[7] F. Nar, O. E. Okman, A. Özgür, and M. Çetin, “RmSAT-CFAR: Fast and accurate target detection in radar images,” *SoftwareX*, vol. 8, pp. 39–42, 2018.

[8] C. J. Venter, H. Grobler, and K. A. AlMalki, “Implementation of the CA-CFAR algorithm for pulsed-doppler radar on a GPU architecture,” in *2011 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, pp. 1–6, 2011.

[9] Y. Sim, J. Heo, Y. Jung, S. Lee, and Y. Jung, “FPGA implementation of efficient CFAR algorithm for radar systems,” *Sensors*, vol. 23, no. 2, 2023.

[10] N. Brown, “Exploring the versal AI engines for accelerating stencil-based atmospheric advection simulation,” in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ACM, feb 2023.

[11] G. E. P. Box and G. C. Tiao, “Intervention analysis with applications to economic and environmental problems,” *Journal of the American Statistical Association*, vol. 70, no. 349, pp. 70–79, 1975.

[12] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.

[13] B. Goehry, “Random forests for time-dependent processes,” *ESAIM: Probability and Statistics*, vol. 24, pp. 801–826, 2020.

[14] A. Pankratz, *Forecasting with Dynamic Regression Models*. Wiley Series in Probability and Statistics, Wiley, 2012.

[15] S. M. Kay, “Statistical signal processing: Detection theory,” *Prentice Hall*, vol. 146, p. 222, 1998.

[16] G. Gao, “Statistical modeling of SAR images: A survey,” *Sensors*, vol. 10, no. 1, pp. 775–795, 2010.

[17] M. A. Richards, *Fundamentals of Radar Signal Processing*. New York, NY, USA: McGraw-Hill Professional, 2005.

[18] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, “Pytorch distributed: Experiences on accelerating data parallel training,” *arXiv:2006.15704 [cs]*, jun. 2020.

[19] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston, “Random forest: A classification and regression tool for compound classification and qsar modeling,” *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 6, pp. 1947–1958, 2003.