Advances in Military Technology
Vol. 20, No. 2, 2025, pp. 565-576
ISSN 1802-2308, eISSN 2533-4123
DOI 10.3849/aimt.01994

Considerations for Delay and Sum Beamformer on a Multi-Core Processor

M. Prakash Narayanan[1,2*], G. Vijay Gopal[1], and R. Rajesh[1]

[1] *Naval Physical & Oceanographic Laboratory, 682021, Kerala, India*
[2] *Cochin University of Science & Technology, Kerala, India*

The manuscript was received on 29 April 2025 and was accepted after revision for publication as an original research paper on 11 December 2025.

Abstract:

This paper presents a real-time, scalable beamformer solution utilizing Intel multicore processors for a Passive Surveillance Sonar (PSS) system. With larger arrays being developed to address the complexities of the ocean environment, the demand to handle high-bandwidth data in the beamformer has become essential. The time-domain delay-and-sum beamformer is analyzed for a cylindrical array, and the best configuration is selected for a simplified realization. The beamformer is demanding in terms of both computation and memory. Considerations for developing an optimized implementation on a multicore machine are discussed, along with the realization of a scalable solution. The beamformer is evaluated for performance with arrays of varying complexities and successfully meets real-time requirements. Finally, a solution for a PSS with a panoramic 450-beam configuration, which has evolved based on this concept, demonstrates the capabilities of the proposed approach.

Keywords:

sonar beamformer, multicore processing, sonar signal processing

1 Introduction

Noisy oceans, together with increasingly silent submarine designs, have led to the development of larger and more complex sonar arrays [1]. This trend demands that the beamformer handle massive amounts of data. With today's multicore processors and high-speed interfaces, compact and cost-effective systems are now within reach.

This work focuses on the design of a parallel and pipelined solution for a Passive Surveillance Sonar (PSS) beamformer – scalable to a 7 600-element array – using readily available hardware that can be configured for different array sizes. With 10 G Ethernet, data aggregation into a processor-based solution has become more feasible,

* *Corresponding author: Naval Physical and Oceanographic Laboratory, 682021, Kerala, India. Phone: +91 484 257 25 06, Fax +91 484 2424858, E-mail: prakashnm.npol@gov.in. ORCID 0000-0002-9167-3360.*

enabling the implementation of a parallel/pipelined approach on Intel multicore processors.

1.1 Literature Survey

Several approaches to realizing beamformers for sonar and radar systems are available in the literature. In [2], a combination of FPGA (Field-Programmable Gate Array) and DSP (Digital Signal Processor) is proposed for a multibeam sonar. Data acquisition and packet processing are performed on the FPGA, while the DSP executes beamformer processing for an 8×8 planar array with 90×90 beams. An analysis of implementation aspects for achieving real-time performance in active sonar is presented. However, scalability is limited by FPGA I/Os, and the system has been shown to scale only up to a 12×12 array. A radar processing solution using DSP with a Serial Rapid I/O interconnect is discussed in [3], where careful interprocessor communication design is emphasized to maximize computational throughput. An FPGA-based high-frequency sonar is described in [4], in which a 360-element array forming 360 beams is implemented. In [5, 6], real-time sonar beamformer implementations on distributed UltraSPARC computers are presented. Specifically, [6] describes a two-stage sonar beamformer realized on 12-processor UltraSPARC workstations. Using process networks, the authors emphasize software portability and configurability. With MPI (Message Passing Interface) [5], three different implementation optimizations are explored, achieving 50 % efficiency compared to a straightforward design (with 91 and 181 beams). A GPU-based scalable beamformer is presented in [7], using a 50-node Cray system with 10-core Intel Xeon processors and Nvidia Tesla K20X GPUs, highlighting heterogeneous computing. The design achieves 40 % of CPU and 51 % of GPU performance. Beamformer optimization aspects on Intel-class multicore processors are discussed in our previous works [8, 9]. This work extends those results to beamformer processing solutions for high-bandwidth passive sonar systems, which can be scaled, reconfigured, and realized on readily available multicore processors.

A scalable, reconfigurable design based on Intel multicore processors is proposed. The development focuses on a computationally efficient beamformer for a 7 600-element array forming 450 beams, with each beam computed using 3 200 sensors. A complexity analysis of the beamformer is presented to guide the selection of an optimized solution. By leveraging the characteristics of the cylindrical array, the beamformer is reorganized to simplify computations and enable a parallel/pipelined architecture, resulting in a scalable and reconfigurable design. Testing confirms both the scalability and reconfigurability of the sonar beamformer.

Section 2 introduces the considered time-domain beamformer algorithms, compares computational complexity, and identifies the one which is most suitable for solving the problem, along with details of the sonar hardware and real-time requirements. Section 3 presents implementation considerations of the beamformer on a multi-core processor. Section 4 discusses the results, and the conclusion is presented in Section 5.

2 Beamformer Analysis

2.1 Sonar Array Architecture

A cylindrical array with 40 elements per stave (in the vertical direction) and 190 elements along the bearing is considered, resulting in a 7 600-element array. For beamform-

ing, 80 elements along the bearing are utilized (i.e., $80 \times 40 = 3\,200$ sensors per beam). The system requirement is to form 150 beams along the bearing and 3 beams in the vertical direction. This results in a total of 450 beams, each computed using 3 200 sensors.

2.2 Delay and Sum Beamformer

This study considers the delay-and-sum (DAS) beamformer in both the time domain (TD) and the frequency domain (FD). Passive Surveillance Sonar (PSS) is realized after analyzing these algorithms, since a larger number of sensors and beams are involved.

If $s_i(n)$ is the signal received at i^{th} sensor of the array, p_i is the position vector of the i^{th} sensor, and w_i is the weight applied, then the beamformer output can be written as

$$b(n,\varphi,\theta) = \sum_{i=0}^{N-1} w_i s_i \left[n - \tau_i(\varphi,\theta) \right] \tag{1}$$

where

$$\begin{aligned} \tau_i(\varphi,\theta) &= p_i \frac{\boldsymbol{u}}{c} \\ &= \frac{1}{c}\left(p_{xi}\cos\theta\cos\varphi + p_{yi}\cos\theta\sin\varphi + p_{zi}\sin\theta \right) \end{aligned} \tag{2}$$

is the delay incurred by the signal at the i^{th} sensor; $p_i = (p_{xi}, p_{yi}, p_{zi})$, is the position vector; $\boldsymbol{u}$ is the unit vector in the signal direction of arrival with bearing and elevation angles (φ, θ), and c is the velocity of sound propagation. Eq. (1) gives the delay-and-sum TD beamformer (TDBF) output in a specified direction (φ, θ). The frequency domain representation of the beamformer is

$$B(k,\varphi,\theta) = \sum_{i=0}^{N-1} w_i\, S_i(k)\, e^{-j2\pi f_k \tau_i(\varphi,\theta)} \tag{3}$$

where $f_k = k f_s / K$, K is the frequency bin and k frequency index.

The exponential term in Eq. (3) gives the frequency-dependent phase rotation given to the signal $s_i(n)$ for a specific beam direction. $S_i(k)$ is the frequency domain representation of the signal of i^{th} sensor found using the efficient FFT algorithm. To get the beam time series $b(n, \varphi, \theta)$, take the inverse FFT of the beam output, which is the same as that found by Eq. (1). To generate a correct output, an adequate overlap is recommended.

TDBF is an inherently broadband method and is effective for short pulses as well as transient signals. It is possible to handle broadband signals in the frequency domain; however, it is computationally expensive. Time-delay computation in TDBF is valid for all the frequencies in the band of interest. One needs to calculate the phase values for each frequency bin, apply them, and then perform the inverse transformation to obtain the time signal in FDBF.

TDBF requires signal oversampling for the time delays necessary for beamforming, thereby demanding more computational resources and memory [10]. Oversampling, when replaced by interpolation schemes, makes the memory requirements similar to those of FDBF. In FDBF, the signal can be sampled at the Nyquist rate, and precise beamforming can be performed by applying the appropriate phase shift.

2.3 Computational Complexity Analysis

This section works out the Computational Complexity (CC) of each of the beamformers. There are 3 200 ($N_\vartheta \times N_\varphi = 40 \times 80$) elements used for computing the beam in a specified direction, where N_θ and N_φ are the sensors in the elevation and bearing for the beam. It is computationally expensive to compute a beamformer with all the elements taken together ($N = 3\,200$) for each of the $N_B = 450$ beams ($N_E = 3$ in elevation and $N_A = 150$ in bearing). To reduce the computations, we combine the N_θ elements in staves to form $N_E = 3$ vertical beams for each of the $Nx = 190$ staves, called the vertical beamformer (VBF). Horizontal beamformer (HBF) forms N_A beams for each of the vertical beams, taking N_φ elements (VBF output).

Now the computational complexity will be split into VBF and HBF. For finer time delay shifts in the time-domain BF, an interpolation filter of length $N_F = 6$ is used, instead of oversampling by $N_I = 10$ times. Oversampling increases the memory and bandwidth requirements. This filter will compute all the required time delays for the sensor data once it is fetched and can be reused for all the beams for which the sensor might be used. Similarly, FFT in FDBF is computed only once per individual sensor and reused wherever needed, eliminating re-computations.

The TDBF complexity can be calculated as follows. The number of multiplications (N_m) and number of additions (Na) for Eq.1 are as follows:

$$N_m = N \cdot N_F + N = N\,(N_F + 1) \tag{4}$$

$$N_a = N(N_F - 1) + N - 1 = N \cdot N_F - 1 \approx N \cdot N_F \tag{5}$$

CC per beam is the sum of Eq. (4) and Eq. (5).

$$\mathrm{CC}_{\mathrm{beam}} = N\,(2\,N_F + 1) \tag{6}$$

Hence CC for TDBF for all beams Eq. (6) to by multiplied by the number of beams.

$$\mathrm{CC}_{\mathrm{TD}} = N_B \cdot N\,(2N_F + 1) \tag{7}$$

Now for the TDBF realized as VBF+HBF, we will be calculating the computational complexity. In Eq. (7), the beamformer is calculated once, and the interpolation is carried out every time. If the interpolation operation is done only once and the beamformer is split into VBF + HBF, there is scope for a reduction in computation. If each sensor is interpolated N_I times, N_θ elements contributing to a beam, and the number of beams N_E in the vertical direction, N_x staves, the computational complexity can be calculated as:

$$\mathrm{CC}_{\mathrm{TDOS}} = 2N_X \cdot N_E \cdot N_\theta \cdot N_I + N_I \cdot N_\theta \left(2N_F - 1\right) + N_B \cdot N_\theta \tag{8}$$

The computational complexity of FDBF has two parts. FFT computation for all sensors, which are common across all beams, requires $N_S N_2 K \log_2(N_K)$ multiplications and $N_S N_K \log_2(N_K)$ additions. These are complex number multiplications and additions; real multiplications will be 4 times, and real additions will be 3 times the above. Therefore, the combined CC for the FFT of all N_S sensors is

$$\mathrm{CC}_{\mathrm{FFT}} = 4\,N_S\,(N_K / 2)\log_2(N_K) + 3\,N_S\,N_K \log_2(N_K) = 5\,N_S\,N_K \log_2(N_K) \tag{9}$$

FDBF involves phase-shifting the $N_K/2$ FFT bins with complex phase shift values. This, together with the window function, needs to be taken. Each beam has N sensors, and N_B beams are formed. This takes $N_B 2NN_K/2 = N_B N N_K$ multiplications and

$N_B NN_K/2-1=N_B NN_K/2$ additions. These are complex operations. Each complex multiplication takes 4 real multiplications and 2 additions. Each complex addition is 2 real additions. Hence, the CC for phase shift and add of FDBF (CC_{MA}) is $(4 + 2)\cdot N_B\cdot N\cdot N_K + 2\cdot N_B\cdot N\cdot(N_K/2) = 6\cdot N_B\cdot N\cdot N_K + N_B\cdot N\cdot N_K$. That is to complete the FBDF the CC is as follows.

$$\mathrm{CC}_{\mathrm{MA}} = 7N_B N N_K \tag{10}$$

Combining Eq. (9) and Eq. (10), we get the CC for FDBF as given below:

$$\mathrm{CC}_{FDO} = 7N_B N N_K + 5N_S N_K \log_2\left(N_K\right) \tag{11}$$

FDBF can be calculated as VBF + HBF, in which case the CC can be computed as follows. N_E beams formed with N_θ sensors each, with N_x staves. Using Eq. (10) CC for VBF will be $7N_xN_EN_\theta N_K$. HBF: N_B beams are formed with N_φ sensors contributing to a beam.

Using Eq. (10), CC of HBF will be $7N_BN_\varphi$ and the CC will be the sum of VBF, HBF and the FFT for all sensors.

$$\mathrm{CC}_{\mathrm{FDOS}} = 7N_K\left(N_X N_E N_\theta + N_B N_\theta\right) + 5N_S N_K \log_2\left(N_K\right) \tag{12}$$

Tab. 1 summarizes the computational complexity (Floating Point Operations) for the beamformers discussed. The CC for FFT-based methods are for epoch, whereas the TD methods are per point. Compared to FDBF, TDBF is simpler in terms of computational requirements, and both give the same output/performance for a broadband system. Fig. 1 shows the comparison of all beamformer types in Tab. 1 for different numbers of sensors used in the beamformer. The numbers in Fig. 1 are normalized to CC per second and are hence comparable. It is observed that the TDBF with VBF followed by HBF gives the best performance in all these cases. Hence, this paper will further pursue TDBF with VBF and HBF as separate stages.

Tab. 1 Summary of computational complexity of different beamformer configurations

Method and Reference equation	Computational Complexity (Number of Operations)	Details
TD, Eq. (7)	$N_BN(2N_F+1)$	TD BF
TDOS, Eq. (8)	$2N_xN_EN_\theta N_I + N_IN_\theta(2N_F-1)+N_BN_\varphi$	TD BF with VBF & HBF, interpolation only once
FDO, Eq. (11)	$7N_BNN_K + 5N_SN_Klog_2(N_K)$	FD BF with FFT once
FDOS, Eq. (12)	$7N_K(N_xN_EN_\theta + N_BN_\varphi) + 5N_SN_Klog_2(N_K)$	FD BF with FFT once and VBF + HBF

2.4 A Scalable Hardware Architecture

Fig. 2 shows the proposed system architecture. The low-noise amplifier (LNA), followed by an amplifier to provide the required gain for the analog-to-digital converter (ADC), is realized on a modular printed circuit board (PCB). Each printed circuit board (PCB) has up to 32 channels, with an FPGA sending the digitized data as a UDP (User Datagram Protocol) packet over a 10G Ethernet. A synchronization signal is used to synchronize all ADCs (Fig. 2). The signal processor is proposed on a multicore Xeon processor. Once the high-bandwidth data from the signal conditioning circuits is taken and synchronized, the signal processing function will be implemented in the Xeon processor.

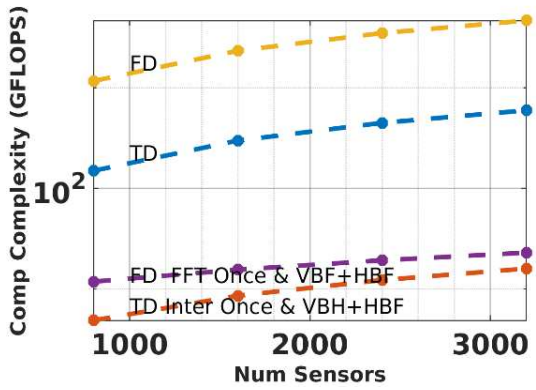

Fig. 1 Comparison of computational complexity by varying the number of sensors used in the beamformer for different cases given in Tab. 1 *(GFLOPS for* 450 beams*)*

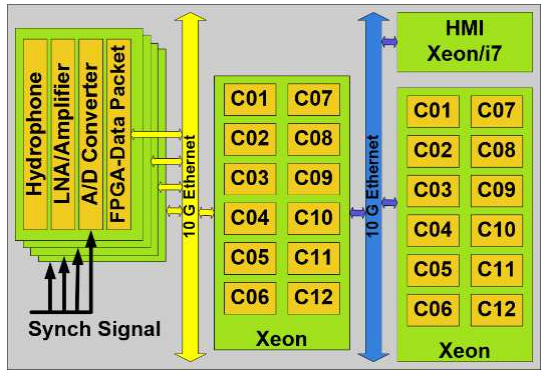

Fig. 2 PSS configuration with scalable signal conditioning, processing, and industry standard interfaces

Processing Elements: The proposed processing hardware is based on Intel processors. Both i7 and Xeon are candidates; Xeon has more cores on a die and larger cache memory. The solution is developed on a Xeon server-class machine; however, it can easily be ported to any Intel machine.

Data Telemetry: A 10G Ethernet is proposed as the data telemetry backbone, considering higher-bandwidth data. Ethernet is an industry standard and is readily available. To enable easy scaling of the system, we propose using two networks.

Real-time Constraints: A packet is generated every t_p = 500 us, with a sampling time of 32 kHz. Batch processing of 2 048 samples takes, t_E = 64 ms. Therefore, the real-time constraint (processing time) for the signal processor is

$$t_{\text{proc}} = 64 \text{ ms} \tag{13}$$

A Xeon D 16-core computer is used for processing, with the clock frequency pinned to 1.5 GHz. Xeon has a theoretical peak performance of $1.5 \text{ GHz} \times 32 \times 16 = 820$ GFLOPS, 32 being the single-precision FLOP/Hz. This implies that the beamformer for the array of 7.6 k elements can be carried out in a Xeon-based processing solution. However, the solution needs to account for the high amount of data handled in the processor through a 10 G Ethernet interface.

3 Realization of Beamformer on Multicore Processor

This section deals with software considerations for the utilization of the Xeon processor architecture. The application is coded using lightweight pthreads on the Linux OS. To maximize performance, computationally intensive processing schemes are coded using Intel Performance Primitives (IPP). The combination of pthreads and IPP is used to maximally utilize the computational resources without compromising scalability. Effective utilization of the cache maximizes beamformer performance. Analysis of these considerations for optimizing the beamformer is presented in [7, 8, 10]. OpenMP automates the threading schemes; however, an earlier study has shown that a pthread-based implementation is better than OpenMP without compromising the scalability of the application [8, 10].

3.1 Processor Architecture Considerations

The microarchitecture of Intel has a superscalar pipeline, SIMD instructions, and out-of-order execution, along with branch prediction and speculative execution. This is enabled through the hardware and compiler of the Intel processors. The most important feature of the Xeon D, of interest in signal processing applications, is the AVX-2 vector units. This vector unit can perform a Fused Multiply-Add (FMA) operation, which specifically performs a multiply-and-accumulate operation. The superscalar architecture exploits Instruction-Level Parallelism (ILP) to the maximum, with the Xeon D, which is based on the Broadwell family, featuring a 14-stage pipeline. This enables the processor to perform branch prediction, out-of-order execution, and speculative execution. This is managed by the hardware at runtime, and Intel's Dynamic Scheduling Technology handles this operation. The dynamic scheduler checks dependencies of the instructions, takes necessary actions to rename registers, places them in the queue in the order of operand availability, tracks data dependencies for out-of-order execution, and manages the in-order retirement of instructions to ensure consistent results. This maximizes CPU utilization, parallelism, and hides the latency in execution.

The Xeon D 1 500 class processors have L1, L2, and L3 caches. The Broadwell family L1 cache is dedicated to each core in the processor. L1 is a low-latency memory and has a few tens of kB/core available exclusively (32 kB each for instruction and data per core), which varies depending on the specific processor. Mid-level cache (L2) is also dedicated to individual cores, but with slightly more latency than L1, and larger in size (256 kB/core). Low-Level Cache (LLC or L3), with higher latency, is shared by all processor cores. This is larger than L1 and L2 (usually 1.2 to 2.5 MB/core) and is faster than the off-chip DDR. The memory hierarchy is designed to get the maximum performance out of the computational capabilities of the cores and the associated AVX engines. In a shared memory architecture realized using a Xeon processor, inter-core communication can easily be achieved using the L3 cache, as this is common to all cores. Beamformer data partitioning critically affects the performance of the code in each processor. Hence, careful application design is mandated for high-performance real-time systems.

Memory alignment to 32-byte boundaries for AVX and 64-byte boundaries for AVX-2 improves the speed of access and vectorization. An inline assembler, though the least portable option, offers complete control over vectorization. Intrinsics consist of a collection of data types and internal compiler functions that directly correspond to processor instructions, with vector registers being allocated by the compiler. Using

extensions such as Intel Array Notation, Intel ISPC, Apple Swift SIMD, and libraries including C++17 SIMD types improves execution speed. Intel compilers provide ease of use and high code portability. Signal processing algorithms such as FIR filters and FFTs require multiply-and-accumulate operations. The AVX-2 hardware of the Xeon D processor provides Fused Multiply Accumulate (FMA) instructions for floating-point (double and single precision) and integer numbers. Depending on the precision, the SIMD engine can perform these operations on multiple data points simultaneously, providing efficient code vectorization. A careful analysis of the algorithm can reveal opportunities for vectorized processing of signal-processing functions [11]. Vectorization is essential for achieving better performance in beamforming.

3.2 Beamformer Algorithm

VBF followed by HBF reduces computational complexity (Section 2.3). Separate threads are created to carry out the vertical beams in six data sets, partitioned without overlap. VBF outputs are separately assigned to three threads and do not require any data overlap. Splitting the beamformer into VBF and HBF removes the requirement for sensor data overlap, reducing data movement from memory. Additional beams, if required, can be formed by scaling the system. This also improves data availability in the cache, accelerating execution and enhancing computational efficiency. A previous study [12] shows a 70% increase in the Fused Multiply-Add (FMA) operations used in beamformer computation.

3.3 Threading Options

The HBF and VBF are coded with OpenMP and pthread threads to compare efficiencies. The OpenMP version schedules the beamformer threads automatically, which is an easy way to realize parallel code from a user perspective. However, in the pthreaded version, users must split the beams (and hence computations) to balance the load across processing cores. The beams are scheduled in two different ways (Fig. 3) to study the effect of memory access. In Fig. 3b, alternate beams are given to different threads, effectively making memory accesses to consecutive locations by different threads. This can lead to much more memory access than the second scheme. In Fig. 3a, consecutive threads are given a batch of beams to be processed. This makes, as seen from the results, the memory access more ordered for the cache controller.

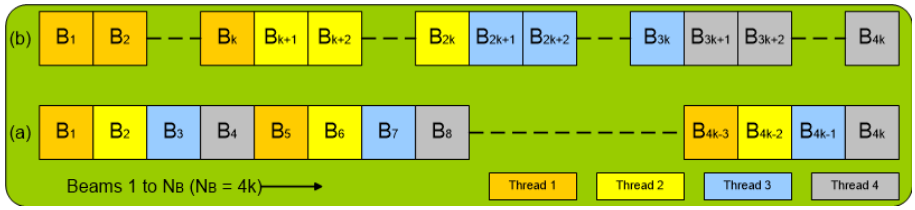

Fig. 3 Beam partitioning across threads in pthread scheme.

4 Results and Analysis

To study beamformer performance with OpenMP and pthread, a single-threaded version without any parallelization is used as a benchmark (Fig. 4). The pthreaded version is configured with both options shown in Fig. 3 for the beam computational task parti-

tioning. OpenMP is configured to manage the workload by itself, with a maximum limit on core usage specified to make it comparable with the pthread version.

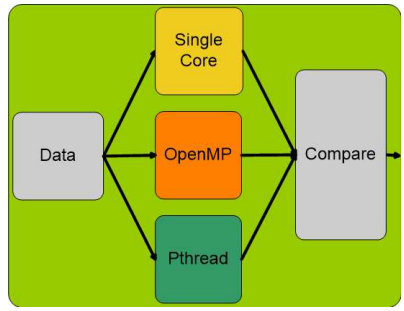

Fig. 4 Test setup for evaluation of threading schemes

Fig. 5 gives the performance figures for different data block lengths (512, 1 024, 2 048, 4 096, 8 192) and threading schemes. There is an increase in performance with-block length. Numbers in brackets on the x-axis (p, q) indicate the number of cores used (p) and the beam stride (q). An increase in block length could lead to data starvation. The multi-threaded beamformer running on four cores performs better, as multiple cores are utilized. With simpler coding, OpenMP closely follows the custom pthreads. More effort in fine-tuning increases utilization, as seen in pthread versions with better GFLOPS. This implies that the different levels of memory are better utilized with block processing, which is separated in shared memory. The pthreaded version with beams processed in groups performs better.

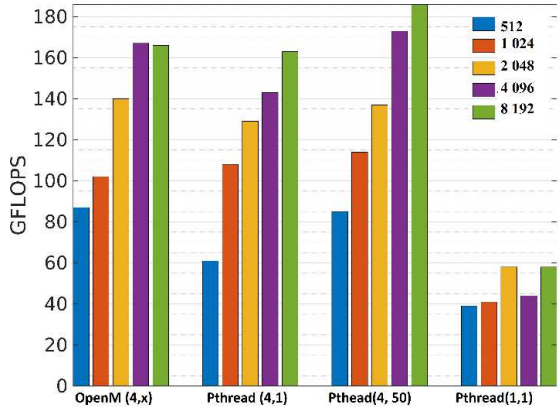

Fig. 5 Performance figures for the different threading schemes

The array configuration discussed in Section 2.1 is used to study the beamformer's real-time behavior and scalability. The system scalability is demonstrated for the beamformer configured for two array complexities: a 1 520-element array and a 7 600-element array. Both cases are run on a workstation (WS)-class Xeon Gold machine with a theoretical performance of 96 GFLOPS/core and an embedded Xeon-based single-board computer (SBC) with a theoretical performance of 48 GFLOPS/core. The timings are tabulated (Tab. 2) with the beamformer configured with 2-, 4-, and 6-point interpolators. The beamformer complexity increases with the interpolator. Similarly, the workstation-class machine is superior, with AVX-512 per-

forming better than the SBC. Processor timers are used to measure the timings. The system is running Linux OS, and hence, variability in the execution timings of threads is expected.

The timings in Tab. 2 are averaged over a large number of samples taken with the beamformer receiving input data continuously. A Linux-based OS is used, which can introduce variance in the performance timings. For one of the cases in Tab. 2, a histogram is plotted in Fig. 6. As observed, the timings exhibit a spread caused by the operating-system-based system. In Section 2.4, the real-time constraints required for the system are introduced. In Eq. (13), the time for completion of the beamformer task is calculated as 64 ms. The results show that, for all considered beamformer and array complexities, the system satisfies the real-time requirements.

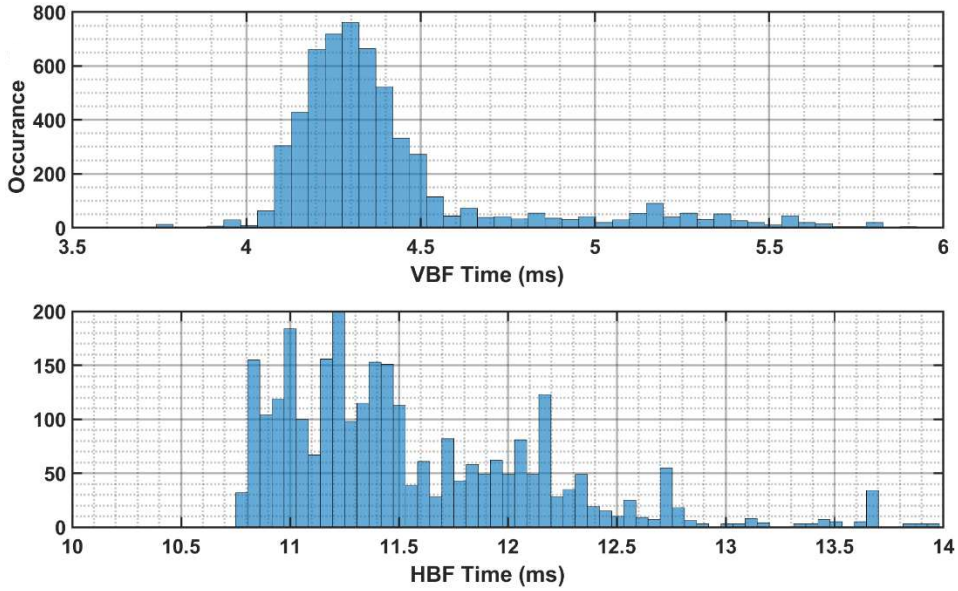

Fig. 6 Histogram of VBF, HBF timings for Case I in Tab. 2

Tab. 2 Timings of different array and beam-former complexities in multiple hardware platforms

H/W	INTRP	VBF	HBF
Case I (1 520) SBC	2	4.1	21.6
	4	4.4	25.3
	6	4.9	33.2
Case I (1 520) WS	2	4.4	11.7
	4	4.9	14.2
	6	4.4	14.0
Case II (7 600) SBC	2	18.1	21.7
	4	20.5	26.6
	6	23.8	37.0
Case II (7 600) WS	2	17.2	13.2
	4	18.0	15.2
	6	17.6	16.2

5 Conclusion

This work presents a comprehensive design method for real-time processing of a time-domain DAS beamformer with a large number of sensors, implemented on multicore processors. The computational complexity of the algorithms is analyzed for different cylindrical array configurations. Factors influencing beamformer performance on multicore architectures are discussed, and the effects of various threading schemes are evaluated. The results show that a pthread-based implementation delivers superior performance. The beamformer meets real-time requirements in two scenarios: arrays with 7 600 and 1 520 elements, each forming 450 panoramic beams. These findings demonstrate the system's scalability across arrays of different sizes and beamformer complexities.

Acknowledgement

Sincere thanks to the Director, NPOL, for granting permission to publish this work, which is carried out as part of the research done with the Department of Electronics, Cochin University of Science and Technology (CUSAT), Kochi, India.

References

[1] BELL, T.G. *Probing the Ocean for Submarines*. Bloomington: Peninsula Publishing, 2011. ISBN 0-932146-26-0.

[2] TIAN, H., S. GUO, P. ZHAO, M. GONG and C. SHEN. Design and Implementation of a Real-Time Multi-Beam Sonar System Based on FPGA and DSP. *Sensors*, 2021, **21**(4), 1425. DOI 10.3390/s21041425.

[3] KLILOU, A., S. BELKOUCH, P. ELLEAUME, P.L. GALL, F. BOURZEIX and M.M. HASSANI. Real-Time Parallel Implementation of Pulse-Doppler Radar Signal Processing Chain on a Massively Parallel Machine Based on Multi-Core DSP and a Serial RapidIO Interconnect. *EURASIP Journal on Advances in Signal Processing*, 2014, **2014**, 161. DOI 10.1186/1687-6180-2014-161.

[4] WANG, J. and K. LIU. High Frequency Active Sonar Real-time Signal Processing System Based on FPGA. In: *IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*. Qingdao: IEEE, 2018. DOI 10.1109/ICSPCC.2018.8567799.

[5] GEORGE, A.D., J. MARKWELL and R. FOGARTY. Real-Time Sonar Beamform-Ing on High-Performance Distributed Computers. *Parallel Computing*, 2000, **26**(9), pp. 1231-1252. DOI 10.1016/S0167-8191(00)00037-5.

[6] ALLEN, G.E. and B.L. EVANS. Real Time Sonar Beamforming on Workstations Using Process Networks and POSIX Threads. *IEEE Transactions on Signal Processing*, 2000, **48**(3), pp. 921-926. DOI 10.1109/78.824694.

[7] SAROFEEN, C and P. GILLETT. A High Performance Parallel and Heterogeneous Approach to Narrowband Beamforming. *IEEE Transactions on Parallel and DIstributed Systems*, 2016, **27**(8), pp. 2196-2207. DOI 10.1109/TPDS.2015.2494038.

[8] NARAYANAN, M.P., G.V. GOPAL and R. RAJESH. Accelerating Performance of a Real-Time 2D Delay-Sum Beamformer on General Purpose Processors. In: *IEEE International Symposium on Ocean Technology (SYMPOL).* Kochi: IEEE, 2021. DOI 10.1109/SYMPOL53555.2021.9689448.

[9] NARAYANAN, M.P., G.V. GOPAL and R. RAJESH. Design and Implementation Considerations for a 3D Beamformer on State-of-the-Art Multicore Processors. In: *IEEE OCEANS 2022*. Chennai: IEEE, 2022, pp. 1-7. DOI 10.1109/OCEANSChennai45887.2022.9775501.

[10] JAECKEL, O. Strengths and Weaknesses of Calculating Beamforming in the Time Domain. In: *1st Berlin Beamforming Conference*. Berlin: BeBeC, 2006.

[11] MARTINEZ-NIETO, D. et al. Digital Signal Processing on Intel Architecture. *Intel Technology Journal,* 2009, **13**(1), pp. 122-145. ISSN 1535-864X.

[12] NARAYANAN, M.P., G. VIJAYGOPAL, and R. RAJESH. A High-Performance Parallel Approach to Delay Sum Beamformer in a Homogeneous Multicore Environment. *Defense Science Journal*, 2024, **74**(5), pp. 755-762. DOI 10.14429/dsj.74.19505.