PARALLEL ALGORITHMS FOR ADAPTIVE MATCHED-FIELD PROCESSING ON DISTRIBUTED ARRAY SYSTEMS

KILSEOK CHO, ALAN D. GEORGE, AND RAJ SUBRAMANIYAN
High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering, University of Florida
P.O. Box 116200, Gainesville, FL 32611-6200

KEONWOOK KIM
High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering, Florida A&M University and Florida State University
2525 Pottsdamer Street, Tallahassee, FL 32310-6046

Received (to be inserted
Revised by Publisher)

Matched-field processing (MFP) localizes sources more accurately than plane-wave beamforming by employing full-wave acoustic propagation models for the cluttered ocean environment. The minimum variance distortionless response MFP (MVDR-MFP) algorithm incorporates the MVDR technique into the MFP algorithm to enhance beamforming performance. Such an adaptive MFP algorithm involves intensive computational and memory requirements due to its complex acoustic model and environmental adaptation. The real-time implementation of adaptive MFP algorithms for large surveillance areas presents a serious computational challenge where high-performance embedded computing and parallel processing may be required to meet real-time constraints. In this paper, three parallel algorithms based on domain decomposition techniques are presented for the MVDR-MFP algorithm on distributed array systems. The parallel performance factors in terms of execution times, communication times, parallel efficiencies, and memory capacities are examined on three potential distributed systems including two types of digital signal processor arrays and a cluster of personal computers. The performance results demonstrate that these parallel algorithms provide a feasible solution for real-time, scalable, and cost-effective adaptive beamforming on embedded, distributed array systems.

1. Introduction

In underwater acoustic signal processing, the classical beamforming approaches based upon plane-wave signals with homogeneous signal field are not the best to localize a source in typical ocean environments. The simple plane-wave model is a poor approximation of the signal received by a large array of sensors from a distant source due to refraction and multipath effects in the ocean. Over the last 20 years, full-wave beamforming based on more accurate models of the acoustic propagation structure has been extensively researched to enhance the limited performance of plane-wave beamforming. In particular, matched-field processing (MFP) originally proposed by Bucker[1] and first implemented by Fizell[2] focuses on acoustic propagation modeling of the ocean waveguide with signal processing algorithms. The MFP algorithm locates acoustic sources more precisely than plane-wave beamforming methods by using a full-wave acoustic propagation model instead of a simple plane-wave acoustic propagation model for the ocean.

The performance of a beamforming algorithm is largely dependent on the accuracy of its acoustic propagation model. The MFP algorithm requires intensive computational and memory capacities to

implement an accurate acoustic propagation model of the ocean, and hence suffers from the complexity of its acoustic propagation model. The MFP algorithm is also sensitive to environmental mismatches such as incorrect models for ocean waveguide and acoustic sources. Recent research on MFP algorithms has concentrated on exploiting robust adaptive algorithms to overcome environmental mismatches and constructing fast processing techniques necessary to meet the substantial computational requirements of real-world applications.

A considerable amount of work has been done to improve the performance of MFP algorithms by employing more accurate acoustic propagation models and more adaptive algorithms in order to overcome mismatching environmental and signal characteristics. The main approaches that have been studied to enhance the performance and robustness of MFP algorithms are reviewed by Vaccaro.[3] Specifically, the minimum variance distortionless response (MVDR) algorithm[4] has several features suitable to the MFP algorithm such as good sidelobe suppression and modest tolerance to mismatch which other adaptive methods lack.[5] The adaptive MFP algorithms, which combine the MVDR technique with the MFP algorithm, were extensively used to improve the performance and robustness of beamforming by employing environmental perturbation constraints and dimension reduction on simulated and field data.[6-10] The MVDR-MFP algorithm is used as our baseline for adaptive MFP algorithms.

Computationally, adaptive MFP algorithms are highly complex because they must compute the replica field from a complex acoustic propagation model over dense grid points and correlate the computed field with the observed field adaptively at each grid point. Despite the ever-increasing performance levels of single-processor computers, the real-time implementation of adaptive MFP algorithms for many hydrophones and broadband frequencies still presents a serious computational challenge. In order to satisfy these ever increasing computational trends, recent research has emphasized two approaches. One approach is to develop fast and efficient algorithms by reducing the complexity and exploiting fast mathematical techniques. The other approach is to develop parallel processing techniques for distributed sensor array systems.

As for the first approach, Ozard et al.[11] presented the Nearest Neighbors technique for selecting replica to reduce the computational search within large ocean areas. By applying this technique to the Bartlett beamformer, a speed improvement was obtained with a loss in detection performance. Cox et al.[12] described a variant of beam-space processing that has a dimensionality reduction method to diminish computational requirements by using only the beams that contain significant signal energy. A fast MFP algorithm based on the FFT technique developed by Aravindan et al.[13] reduces the computational demands at the cost of source peak in a shallow water channel. However, as might be anticipated, the performance of most of these algorithms is reduced in comparison to original beamforming algorithms.

The other approach to reduce computational cost is to decompose the computational load and memory space of beamforming algorithms over multiple processors in parallel computer systems. Several parallel algorithms and systolic array architectures have been described for adaptive beamforming algorithms.[14, 15] Despite the improvement of throughput, these algorithms are closely coupled to the systolic array architecture implemented with VLSI techniques and cannot be efficiently applied on modern general-purpose distributed systems due to high interprocessor communication requirements. Parallel beamforming algorithms have also been studied for split-aperture conventional beamforming (SA-CBF)[16], adaptive minimum variance distortionless response beamforming,[17, 18] and conventional matched-field processing (CMFP)[19] algorithms. In the aforementioned studies, different parallel algorithms based on control and domain decomposition for beamforming are introduced and scalability of parallel performance on distributed systems is examined.

The significant computational and memory requirements of the MVDR-MFP algorithm make imperative the development and use of parallel processing techniques on real-time sonar array systems. In this paper, three parallel algorithms suitable for the MVDR-MFP algorithm are developed to meet the computational challenges. The parallel algorithms have been developed in the light of providing a feasible solution for real-time and cost-effective implementation on embedded, distributed DSP systems. The first is the frequency decomposition (FD) algorithm that exploits data parallelism between processing tasks of different frequency bins. The FD algorithm statically allocates the processing of different subsets of frequency bins to different processors by using the concurrency between beamforming tasks of different frequency bins. The second algorithm is the section decomposition (SD) that is based on data parallelism in the processing of grid points. The SD algorithm distributes the processing of different subsets of grid points into different processing nodes by using output domain decomposition. The third decomposition algorithm is a variant of the iteration decomposition method known as the iteration-frequency decomposition (IFD) scheme. The IFD algorithm distributes processing of successive iterations into different processors by round-robin scheduling of beamforming iterations. Within a given iteration, processing of different sets of frequency bins is partitioned across multiple stages of execution to balance the load. The three parallel algorithms are statically scheduled to reduce interprocessor communication, synchronization, and other parallel management overheads that otherwise deteriorate the performance of parallel algorithms on distributed systems.

This paper focuses on examining and analyzing performance tradeoffs of the parallel algorithms on a cluster of personal computers (PCs) and two arrays of digital signal processors (DSPs). The testbeds are used to investigate the parallel performance on three different types of processors, communication networks, and system architectures. The performance characteristics are analyzed in terms of execution times, communication times, memory requirements, and parallel efficiencies of the sequential and parallel algorithms.

Section 2 presents background on the MVDR-MFP algorithm and Section 3 describes the computational tasks of the sequential algorithm and beamforming outputs of CMFP and MVDR-MFP. In Section 4, the configurations and features of three parallel algorithms for the MVDR-MFP are explained, and the communication characteristics of the parallel algorithms are also examined. Section 5 describes three experimental testbeds and their software environments. Section 6 experimentally analyzes the performance results of the parallel algorithms. Finally, Section 7 presents conclusions and directions for future research.

2. Overview of the MVDR-MFP Algorithm

The MFP algorithm localizes a source in the ocean by matching the field measured by a sensor array with the replica field derived from a full-wave acoustic propagation model over a range of all expected source positions. The MVDR algorithm originally introduced by Capon[4] is an adaptive array processing technique that adjusts linear weighting of sensors to minimize variance of output while maintaining the unity gain constraint for the direction of interest. The MVDR-MFP algorithm incorporates the MVDR approach into the MFP algorithm to enhance beamforming performance. Thus, MVDR-MFP is a high-resolution adaptive MFP algorithm that adaptively constructs an optimum weighting vector as in MVDR and uses a more accurate acoustic propagation model of the ocean waveguide as in MFP.

The replica field of the MVDR-MFP algorithm is generated from an acoustic propagation model based on refractive and multipath effects of the ocean waveguide. For precise source localization, the acoustic propagation model must be accurate otherwise the performance of the MVDR-MFP algorithm will be degraded due to the inaccuracy of the modeling assumptions. The ocean is modeled as a waveguide and

the source signal is assumed to be the point source solution to the wave equation. The wave equation for a source is given by

$$\nabla^2 p(r,z) + \frac{\omega^2}{c^2(r,z)} p(r,z) = \frac{-\delta(r-r_s)\delta(z-z_s)}{r} \tag{2.1}$$

where ∇^2 is the Laplacian operator, *p(r,z)* and *c(r,z)* represent the pressure and sound speed respectively at range *r* and depth *z*, ω is the angular frequency of the source location at r_s and z_s, and $\delta(x\text{-}x_0)$ is the delta function whose amplitude is unity at x_0.[20] The pressure field at position (r,z) from an acoustic source at position (r_s,z_s) is obtained by solving Eq. (2.1). Numerous acoustic models have been proposed to solve the wave equation. The complexity and solution of the wave equation depends on the model applied. In this paper, the normal mode model is employed as the acoustic propagation model of the ocean because it provides an accurate and computationally efficient propagation model for MFP applications. The normal mode method expresses the acoustic pressure field in terms of a normal mode expansion and then solves for the eigenvalues and eigenfunctions of the wave equation. The acoustic pressure field at position (r,z) is the weighted sum of the contribution from each mode and can be calculated from

$$p(r,z) \approx \frac{i}{\sqrt{8\pi r}} e^{\frac{-i\pi}{4}} \sum_{m=1}^{\infty} \Psi_m(z_s)\Psi_m(z) \frac{e^{ik_m r}}{\sqrt{k_m}} \tag{2.2}$$

where eigenfunction $\Psi_m(z)$ and eigenvalue k^2_m are the mode shape function and horizontal wave number for mode *m*, respectively.[21] The KRAKEN normal mode program developed by Porter[22, 23] is employed herein to predict acoustic propagation in the ocean. The KRAKEN program, which is one of the most widely used models, efficiently and accurately calculates horizontal wave numbers and mode shape functions for a given acoustic ocean environment.

The MVDR processor is categorized as an adaptive beamformer with linear constraints as given by

$$w^H p = g \tag{2.3}$$

where *w* is the weight vector of the adaptive beamformer, *p* is the replica vector, and *H* denotes complex-conjugate transposition. By imposing this constraint, it is ensured that signals from the steering position of interest are passed with gain *g* (which equals unity in the case of MVDR) while the output power contributed by interference signals from other positions is minimized using a minimum mean-square criterion. Therefore, assuming that *p* is normalized, we obtain the weight vector by solving

$$S = w_{mvdr}(r,z)^H R w_{mvdr}(r,z) \quad \text{subject to} \quad w_{mvdr}(r,z)^H \bar{p}(r,z) = 1 \tag{2.4}$$

where w_{mvdr} is the weight vector of the MVDR-MFP beamformer, *R* represents the cross-spectral matrix (CSM) expressed in Eq. (2.5), and $\bar{p}(r,z)$ is the normalized replica vector obtained from Eq. (2.6). The CSM is the covariance of the frequency-domain input data of an array as given by

$$R = E[xx^H] \tag{2.5}$$

where $E[]$ is the expectation operation and *x* is a frequency-domain input data vector. The replica vector is normalized to have unit magnitude as follows:

$$\bar{p}(r,z) = \frac{p(r,z)}{|p(r,z)|} \tag{2.6}$$

Solving Eq. (2.4) by the method of Lagrange multipliers, the weight vector is derived as:

$$w_{mvdr}(r,z) = \frac{R^{-1}\overline{p}(r,z)}{\overline{p}(r,z)^{H} R^{-1}\overline{p}(r,z)} \tag{2.7}$$

Finally, by substituting the w_{mvdr} calculated from Eq. (2.7) into Eq. (2.4), the output power for a steering position (r, z) is obtained as

$$S_{mvdr}(r,z) = \frac{1}{\overline{p}(r,z)^{H} R^{-1}\overline{p}(r,z)} \tag{2.8}$$

where $S_{mvdr}(r, z)$ is the detection factor at range r and depth z that presents the likeliness of detection for a given data set. The MVDR-MFP algorithm finds source locations accurately by calculating the weighting vectors adaptively based on the sample data from Eq. (2.8).

3. Sequential MVDR-MFP Algorithm

The computational tasks of the MVDR-MFP algorithm include Fast Fourier Transform (FFT), Sequential Regression (SER) inversion, steering, and broadband averaging. The incoming time-domain data is transformed into frequency-domain by the FFT stage and then suitable frequency bins are selected from the FFT output for broadband processing. The inverse CSM is directly updated from the FFT output by the matrix inversion lemma in the SER inversion stage. The output power of a narrowband frequency is calculated by steering the array for all possible grid points and the average output power for the selected frequency set is obtained in the broadband averaging stage. The computational block diagram is depicted in Fig. 1 where $x(k)$ represents the frequency-domain input data vector, R^{-1} is the inverse CSM matrix, S_{mvdr} is the MVDR-MFP beamforming output for narrowband frequency given in Eq. (2.8), and S_{av} denotes the MVDR-MFP beamforming output for broadband frequency obtained from Eq. (3.2). The steering stage is performed as many times as the number of grid points to steer an array for all possible locations of sources under consideration. The SER inversion and steering stages are repeated for the number of selected frequency bins for broadband processing. Finally, the outermost loop including all stages of the MVDR-MFP algorithm is performed once for every iteration obtaining one snapshot of input data from an array of sensors.

The FFT stage transforms the input sample data received by the array of sensors from time-domain to frequency-domain. The computational complexity of the FFT task implemented by the radix-2 butterfly method is generally $O(N\log N)$ in terms of data length. However, the FFT stage in MVDR-MFP involves a complexity of $O(N)$ in terms of the number of sensor nodes because the FFT process is simply replicated for each sensor node and the FFT data length is fixed. Therefore, the computation complexity of the FFT stage is linearly increased as the number of sensor nodes is increased.

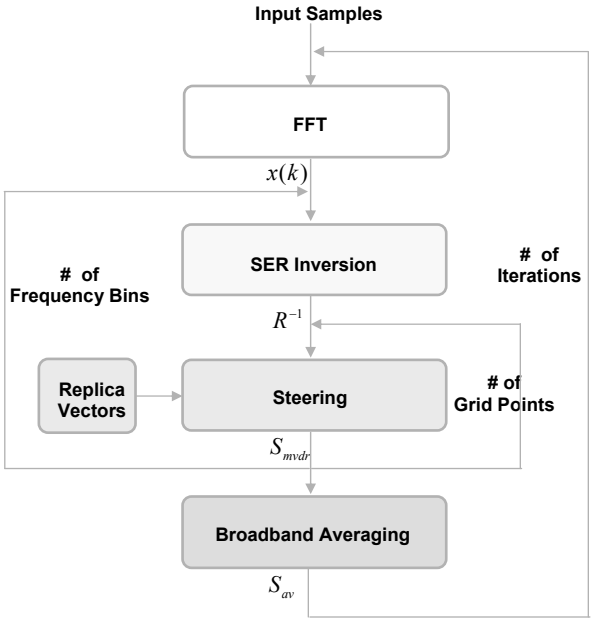


Fig. 1. Computational block diagram of the sequential MVDR-MFP algorithm.

The SER inversion stage calculates the inverse CSM matrix. The computational intensity of matrix inversion increases with the size of matrix but there exist numerous inversion techniques to reduce computation time. For example, Gauss-Jordan Elimination (JDE), LU decomposition and sequential regression (SER) methods are widely used in beamforming applications. The choice of a suitable inversion technique for beamforming applications affects the execution time of beamforming algorithms as well as determines the applicability of different parallelization methods. Sinha at el.[18] implemented the above three inversion techniques to study the trade-offs in terms of computation time and memory requirements. They suggested that the sequential regression method is the most computationally efficient but has a relatively large memory requirement. The computational complexity of the SER method is $O(N^2)$ while the complexities of both JDE and LU methods are $O(N^3)$ with regard to size of the matrix. The SER method is used as the CSM inversion method here because it is one of the most computationally efficient methods and does not need to calculate CSM. In the SER technique, the inverse CSM is initialized with non-zero values and updated during every subsequent iteration as shown in the following matrix inversion equation,

$$\hat{R}_{i+1}^{-1} = \frac{1}{\alpha}\hat{R}_i^{-1} - \left(\frac{1-\alpha}{\alpha}\right)\left(\frac{\hat{R}_i^{-1} x_{i+1} x_{i+1}^H \hat{R}_i^{-1}}{\alpha + (1-\alpha) x_{i+1}^H \hat{R}_i^{-1} x_{i+1}}\right) \tag{3.1}$$

where α is the forgetting factor that determines the weight of the previous inverse CSM estimate relative to the new input data vector, $\hat{R}_i^{-1}$ is the inverse CSM estimate of the i^{th} iteration, and x_i is the frequency-domain input data vector of the i^{th} iteration.

The steering stage is responsible for steering an array and calculating the output power for every steering position. The steering stage calculates the output power from the inverse CSM estimate and the replica vectors over the grid points where sources are likely to be present. The complexity of the steering

stage for the narrowband MVDR-MFP beamformer is $O(RDN^2)$ because the steering loop has quadratic computational complexity in terms of the number of nodes N and is executed as many times as the number of range R and depth D grid points, as shown in Fig. 1. Along with calculation of output power as given by Eq. (2.8), the steering stage includes the replica vector generation task that is invariant to input data. Once replica vectors are calculated from environmental and system parameters, they are not varied until an environmental change is encountered. The computational procedure of Eq. (2.2) for replica vector generation is described in detail by Porter.[22] The replica vector generation task has high-order complexity due to its computationally intensive procedures such as eigenvalue computation. However, as with the FFT stage complexity, the complexity of the replica vector generation is linear with respect to the number of nodes since each sensor node is required to generate the replica vectors for entire range and depth points separately. Despite the low complexity of the task, the computational burden presented by the replica vector generation task is very significant due to a substantial scalar factor. Among all the computational tasks in this experiment, the steering stage is the most computationally intensive.

Broadband averaging is required to calculate the average beamformer output for multiple selected frequency bins. The broadband MVDR-MFP beamformer output is given by

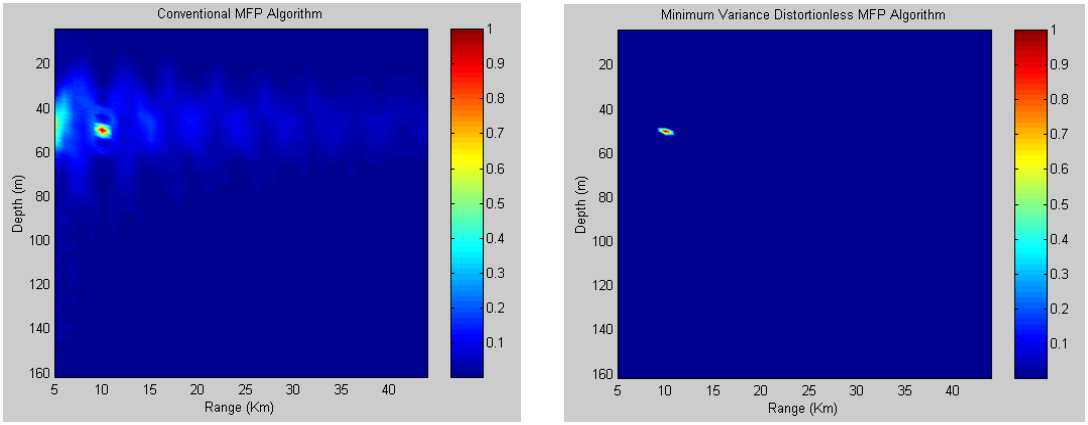
$$S_{av}(r,z) = \frac{1}{B}\sum_{k=1}^{B} S_{mvdr}(f_i, r, z) \tag{3.2}$$

where B is the number of frequency bins selected from the FFT output, f_i represents the i^{th} frequency bin, and $S_{mvdr}(f_i,r,z)$ is the narrowband output power of the i^{th} frequency bin. The narrowband processing including the SER inversion and steering stages for each individual frequency bin is performed as many times as the number of the selected frequency bins for broadband processing. Hence, as the number of frequency bins is increased, the MVDR-MFP beamformer has to compute an increased number of narrowband beamforming results. This broadband processing vastly increases the computation time of the MVDR-MFP algorithm. By averaging the narrowband beamformer outputs over selected frequency bins, interference signals in the sidelobe are smoothed and signals near the main lobe are enhanced because the positions of sidelobes are generally frequency dependent whereas the location of main lobe remains constant. Thus, the detection probability for narrowband beamforming can be enhanced by broadband averaging over multiple frequency bins.

To build a baseline for the three parallel MVDR-MFP algorithms, two implementation models are considered; a minimum-memory (MM) model and a minimum-computation (MC) model. The MM model requires less memory capacity but more computation time by calculating temporary values on the fly as needed during execution. On the contrary, the MC model requires more memory space and less computation time since those values are precalculated, stored in memory, and reused when needed.

The final beamformer output is commonly depicted as an ambiguity surface indicating the likeliness of target detections. In the ambiguity surface, peak positions denote the locations that are most likely to be targets. To compare the detection capabilities of the CMFP and MVDR-MFP algorithms, the ambiguity surfaces for a source at 10Km in range and 50m in depth are shown in Fig. 2a and Fig. 2b, respectively. In this experiment, the pressure field data was generated for a point source having 32 frequency bins with 1Hz across from 200Hz to 231Hz from a vertical array, which contains 32 hydrophones spaced at 4m apart from 10m to 70m in depth. The noise component for each hydrophone has a Gaussian distribution with zero mean and signal-to-noise ratio (SNR) of 10dB. The ambiguity surfaces are computed from 5 to 44Km in range at 1Km intervals, and from 2m to 160m in depth, at 2m intervals. From the two beamforming results, it is seen that the MVDR-MFP beamformer localizes the source more precisely than the CMFP as well as

adequately suppressing sidelobes, since it calculates optimum weight vectors adaptively for every given sample data set.



(a) (b)

Fig. 2. Ambiguity surfaces of CMFP (a) and MVDR-MFP (b).

4. Parallel Algorithms for MVDR-MFP

As described in the previous section, the MVDR-MFP algorithm with a large number of sensor nodes, a wide range of frequency bands, and dense grid points requires both significant computational and memory capacities. When beamforming algorithms are implemented on a real-time sonar system, the challenges exceed the capabilities of single-processor computers including conventional DSP processors. Parallel processing is a solution to execute the MVDR-MFP algorithm in real-time, to achieve cost effectiveness, and to provide dependability on distributed array systems. The parallel algorithms are designed to execute on distributed DSP and PC array systems, which consist of smart processing nodes connected through communication networks. A smart node has its own processing power, the hydrophone to collect input data as well as the communication assist that generates outgoing messages and handles incoming messages.

Parallel algorithms need to distribute workload evenly across processors and reduce the communication, synchronization, and other parallelization overhead to improve performance. When a parallel algorithm breaks sequential tasks into a collection of small tasks to be distributed over available processors, finer-grained strategies impose more parallelization overhead to synchronize and communicate more frequently between processors but may achieve more balanced workload distribution than coarse-grained decomposition methods. Additionally, limited concurrency is the most fundamental problem in achieving the necessary speedup through parallelism per Amdahl's law.[24] In this work, three parallel algorithms for adaptive MFP are developed using domain decomposition methods to exploit more concurrency and lower communication overhead in distributed array systems. The following sections describe the three parallel algorithms: frequency decomposition, section decomposition, and iteration-frequency decomposition.

4.1. Frequency decomposition algorithm

The MVDR-MFP algorithm has concurrency between beamforming operations of different frequency bins since processing of one frequency bin can be executed in parallel with another frequency bin. Frequency

decomposition (FD) statically allocates processing for different subsets of frequency bins within a given beamforming iteration to distinct processors. FD is an input domain decomposition technique that exploits data parallelism between processing tasks of different frequency bins. Thus, several processing nodes in a distributed array system perform the same beamforming operation simultaneously on different data sets for their assigned frequency bins as in a single-instruction multiple-data (SIMD) machine. The FD method can be easily implemented for the MVDR-MFP algorithm without large parallelization overhead.

In a distributed array system, the FFT operation is an inherently distributed implementation since each node performs the FFT operation only for data from the sensor attached to its own node. Thus, the FFT output data from each node must be transmitted to other processors for a subsequent beamforming stage. Each node is only responsible for performing the partial beamforming output for a subset of narrowband frequency bins, and therefore each node must share its beamforming output of all grid points to calculate the average of the total beamforming output for all broadband frequency bins. FD requires two all-to-all communications per iteration where one all-to-all communication is needed to distribute the FFT data and the other to collect the narrowband beamforming output. This communication requirement results in a substantial communication load on a distributed array system. To lessen the communication load, a data packing method is employed to combine the partial beamforming output of the previous iteration and the FFT data of the current iteration as one data packet. As a result, the combined data packet is transmitted once per iteration after the FFT operation of the current iteration. This data packing decreases the communication cost by eliminating the overhead of initiating communications twice per iteration. However, the FD algorithm still needs a significantly large communication message size due to the partial beamforming results. The partial beamforming outputs of all range and depth grid points on the ambiguity surface impose a substantially large communication message on each node because the number of grid points, 3200 in the case of 40 range and 80 depth points, is extremely large for better beamforming resolution.

As shown in Fig. 3, each node performs an FFT operation on the input data received from its own sensor for the current iteration. Each node performs data packing for the FFT output data from the current iteration and the partial beamforming output from the previous iteration, and then executes an all-to-all communication for this data packet. Upon reception of the data packet, each node separates the FFT data and the partial beamforming output from the data packet. Each node obtains the final beamforming output from the previous iteration by averaging the beamforming output for broadband frequencies. In the 3-node array configuration, the FFT data from the current iteration is decomposed into three nodes. The first subset of frequency bins are assigned to node 1, the second subset of frequency bins to node 2, and so on. Each node performs the SER inversion and steers the sensor array to get a narrowband beamforming output over all grid points. The narrowband beamforming output is added to the running sum to obtain the partial beamforming outputs for the assigned frequency bins per node. The above beamforming procedures are repeated for the next iteration. As the number of nodes is increased, the message size of each communication per node is fixed since the partial beamforming outputs over all the grid points and all the FFT output data for the selected frequency bins are broadcasted to other nodes in the FD algorithm. In the block diagram, result latency is defined as the delay between the collection of input data and the computation of the final output. The result latency in the FD algorithm can be slightly increased by the data packing. The data packing can generate more delay time since each node has to wait for the FFT output data from the current iteration to be calculated and then packed with the partial beamforming from the previous iteration.

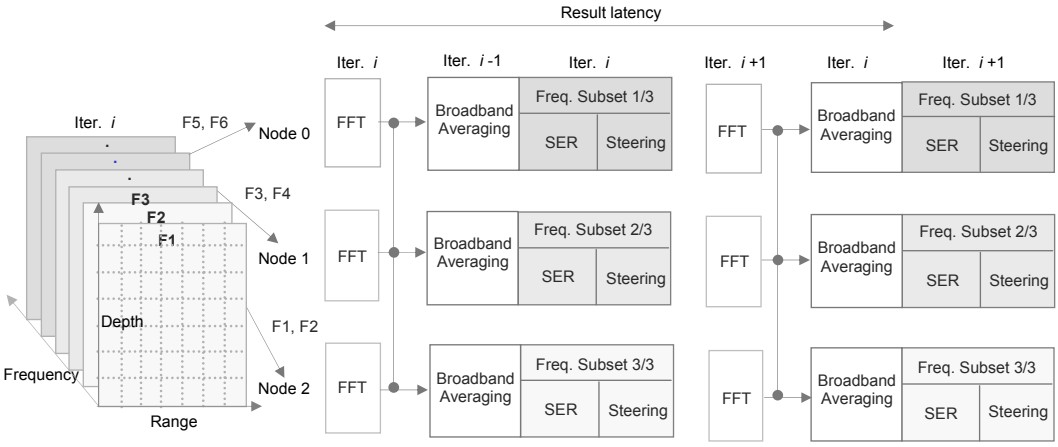


Fig. 3. Block diagram for frequency decomposition in a 3-node array configuration with 6 frequency bins.

4.2. Section decomposition algorithm

The second parallel algorithm features section decomposition (SD) based on data parallelism in the beamforming processing tasks of grid points. The SD algorithm distributes processing of different subsets of grid points within a given beamforming iteration into different processing nodes in the output domain. Unlike both the FD and IFD parallel algorithms, the SD algorithm has a sequential dependency between the SER inversion stage and the steering stage because full spectrums of the inverse CSM are required for the calculation of the beamforming output for a single grid point. Therefore, the SER inversion stage is excluded from the parallel implementation in the SD algorithm. This sequential fraction is not expected to cause a serious computational problem in SD since the computational cost of the SER stage for a given system and problem size is not significant compared with the steering stage. In the SD algorithm, each node requires two all-to-all communications as in FD. To lessen this communication load, the data packing technique is used in the SD algorithm as well, resulting in one all-to-all communication per iteration. As the number of nodes is increased, the message size of each communication for the SD algorithm is decreased because the number of beamforming outputs for the fixed grid points is divided across multiple nodes whereas the message size for the FD algorithm is fixed as described in the previous section.

A block diagram for the SD method is shown in Fig. 4. In the SD algorithm, each node performs the FFT task, the data packing operation, and one all-to-all communication between nodes as in the FD algorithm. Each node executes the SER inversion task to obtain the inverse CSM from the current iteration. After that, in the 3-node array configuration, the overall grid points from the current iteration are decomposed into three nodes. The first subset of grid points are allocated to node 1, the second subset of grid points to node 2, and so on. For its assigned subset of grid points, each node performs the steering task to obtain partial beamforming outputs for all the selected frequency bins and then calculates the average of the beamforming outputs of broadband frequencies. The above beamforming processes are repeated for the next iteration. In the SD algorithm, it is expected that the communication latency will be lower than that of the FD algorithm because the message size for each communication is decreased with increased system size.

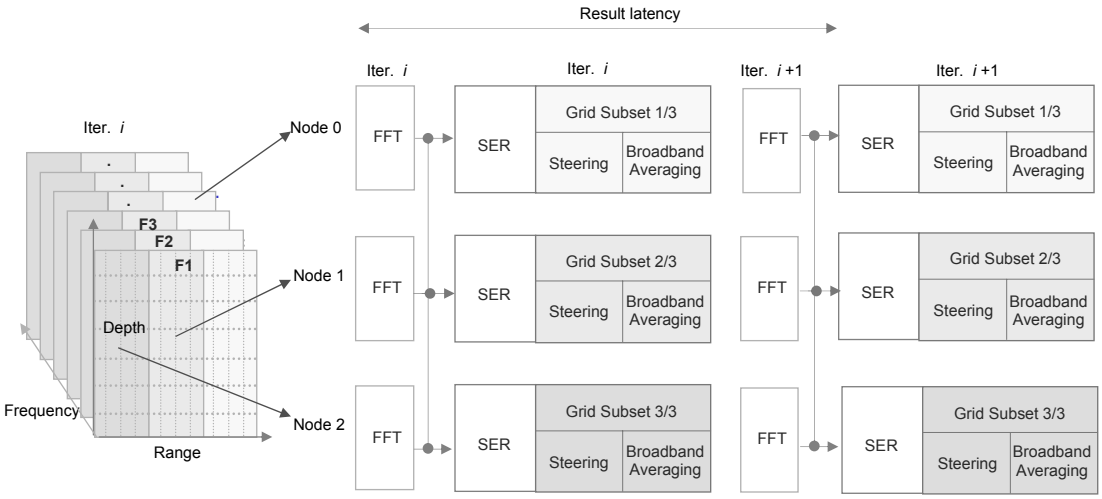


Fig. 4. Block diagram for section decomposition in a 3-node array.

4.3. Iteration-frequency decomposition algorithm

The third decomposition technique is a hybrid of frequency and iteration decompositions known as the iteration-frequency decomposition (IFD). The IFD algorithm exploits parallelism between beamforming tasks of successive iterations and frequency bins. IFD distributes the processing of successive iterations into distinct processors by round-robin scheduling of beamforming iterations. Within an assigned iteration for each node, the processing of different sets of frequency bins is also partitioned across multiple stages of execution for load balance. While one node is performing the beamforming task of the assigned frequency bin for every execution stage, all the other nodes are simultaneously working on their beamforming iterations. At the beginning of the iteration, each node computes the FFT of the data collected from its own sensor and then sends the FFT results to only one node before the beamforming operation of the current iteration begins. The IFD algorithm needs one all-to-one communication per iteration and the smallest message size since each node is required to transmit only one FFT data set to the only node that is responsible for processing the current iteration.

As depicted in Fig. 5, each node performs an FFT operation on the input data after each stage and sends the FFT result to the node that is to perform the current iteration by round-robin scheduling. Then the node partitions the beamforming tasks of different frequency bins into stages (i.e., three in this case) within the assigned iteration. The first subset of frequency bins is processed in the first stage, the second subset of frequency bins in the second stage, and so on. Within an assigned iteration, each node executes the SER inversion and steering tasks for the frequency bins assigned in a stage, and obtains the final beamforming output by executing broadband averaging in the final stage. The above beamforming operations are repeated for the next iteration. As shown in Fig. 5, each node performs all the beamforming tasks for its assigned iteration in the same way as a single node executes all beamforming tasks for every iteration in the sequential algorithm. The IFD algorithm includes some communication and stage overhead. As a result, the result latency of the IFD algorithm is slightly higher than that of the sequential algorithm.

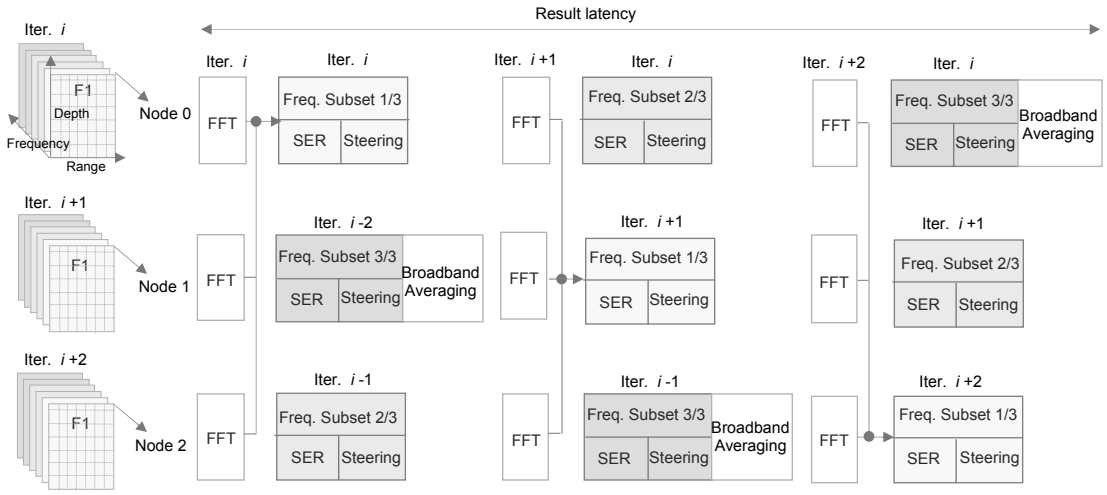


Fig. 5. Block diagram for iteration-frequency decomposition in a 3-node array

4.4. Computational complexity and communication

In this section, we analyze computational complexities, communication patterns, and message sizes for the parallel algorithms presented in previous sections. The computational complexities for the sequential and three parallel algorithms are compared in Table 1 where N is the number of nodes, B is the number of frequency bins, R is the number of range grid points, and D is the number of depth grid points. When broadband averaging is implemented for this research, a running sum of the beamforming outputs for different frequency bins is maintained as soon as a narrowband beamformer output is computed in the steering stage. The final sum for every grid point is only divided by the number of frequency bins B in the broadband averaging stage. The broadband averaging is partially calculated in the steering stage to reduce necessary memory, and its execution time is relatively small compared to other stages. Thus, the complexity and execution time of the broadband averaging is included in that of the steering stage.

As described in Section 3, in the sequential algorithm with a fixed number of data points per sensor, the FFT stage for all sensor nodes requires a computational complexity of $O(N)$. The inverse CSM using the SER algorithm involves a complexity of $O(BN^2)$, and the steering stage has a complexity of $O(BRDN^2)$. Total computational complexity of each algorithm is the sum of the computational complexities of all stages and denoted as the most dominant term among the highest-order terms of the equation.

Table 1. Computational complexities of the sequential and parallel algorithms.

	Sequential	**FD**	**SD**	**IFD**
FFT	$O(N)$	$O(1)$	$O(1)$	$O(1)$
SER	$O(BN^2)$	$O(BN)$	$O(BN^2)$	$O(BN)$
Steering & Broadband averaging	$O(BRDN^2)$	$O(BRDN)$	$O(BRDN)$	$O(BRDN)$
Total	$O(BRDN^2)$	$O(BRDN)$	$O(BN^2 + BRDN)$	$O(BRDN)$

The computational complexities of all stages of the three parallel algorithms except the SER stage in the SD algorithm are reduced by a factor of N as compared to the sequential algorithm by distributing computational loads into N processors. In the SD algorithm, the computational complexity of the SER stage is not decreased since this stage is not parallelized due to the dependency between computational tasks. The execution times of parallel algorithms might ideally be expected to be N times faster than the sequential algorithm. However, it is anticipated that parallel performance will be degraded in comparison to the ideal since there are sources of overhead such as interprocessor communication, synchronization, and unbalanced workload distribution.

The communication patterns and message sizes of the three parallel algorithms are shown in Table 2. Here the number of frequency bins used is 32, the number of grid range and depth points is 40 and 80, respectively, and the number of sensor nodes is N. The communication scheme for both FD and SD algorithms is an all-to-all communication that requires $N(N\text{-}1)$ *send/receive* unicast communications, but the IFD algorithm needs an all-to-one communication requiring N-1 *send/receive* unicast communications per iteration. As the number of sensor nodes is increased, the message sizes of each communication for the FD and IFD algorithms are fixed for the above problem size, but that of the SD algorithm decreases as in Table 2. From these communication characteristics of the parallel algorithms, it is expected that FD will impose the highest communication time, and IFD the lowest communication time.

Table 2. Communication patterns and message sizes for three parallel algorithms.

	FD	**SD**	**IFD**
Pattern	All-to-All communication	All-to-All communication	All-to-One communication
Number of communications per iteration	$N(N\text{-}1)$ send/receive communications	$N(N\text{-}1)$ send/receive communications	N-1 send/receive communications
Message size per communication	(# of frequency bins × 2 + # of grid points) × # of bytes per float variable (4) = 13056 bytes	(# of frequency bins × 8) + (# grid points × 4 ÷ # of nodes) = 256 + (12800 ÷ N) bytes	# of frequency bins × 8 = 256 bytes

5. Experimental testbeds

The performance of the parallel algorithms is analyzed on three distributed platforms; two types of DSP arrays and a PC cluster. The testbeds have similar configurations with multiple processing units connected by loosely coupled communication links. The testbeds contain distinct types of processing units and communication mechanisms. The two DSP arrays use two different types of DSP devices from Analog Devices, and the PC cluster machines have a general-purpose processor as their CPU. All the testbeds use the message passing interface (MPI)[25] as the middleware to communicate and synchronize between processors in the distributed environment. The hardware and software characteristics of both DSP arrays and the PC cluster are summarized in Table 3 and described in the following sections.

Table 3. Characteristics of the three experimental testbeds.

Testbeds	Processor	Link Speed	Memory	Topology	OS	MPI
PC cluster	400 MHz Celeron	100 Mbps	64MB	Switched Star	Linux 2.2.16	MPI/Pro 1.5
ADSP-21602 DSP array	40 MHz ADSP-21062	320 Mbps	Internal: 256KB External: 3MB	Ring	N/A	MPI-SHARC 21602
ADSP-21160 DSP array	80 MHz ADSP-21160	640 Mbps	Internal: 512KB External: 512KB	Ring	N/A	MPI-SHARC 21160

5.1. ADSP-21062 DSP array

The first testbed is composed of eight Bittware Blacktip-EX DSP boards[26] connected to one another with link ports, as shown in Fig. 6a. Each board includes a single 40MHz ADSP-21062 Super Harvard ARChitecture (SHARC) processor[27] from Analog Devices, 3MB external memory with zero wait-state access time, and interfaces for external I/O devices. The ADSP-21062 SHARC processor consists of a 256KB internal memory, multiple link ports, and direct memory access (DMA) controllers, etc. Each link port consisting of four bi-directional data and two bi-directional handshaking lines can operate at twice the clock rate of the processor, achieving a peak throughput of 320Mbps. In this experiment, the link ports and DMA channels are employed to provide high-speed, low-overhead communication between DSP boards. The Blacktip-EX DSP boards contain two link ports with external connectors to communicate with other boards. The link ports are dedicated to transmit and receive channels separately to eliminate the need for external routing or external hardware. This configuration allows the boards to be arranged in a uni-directional ring topology. Although a ring does not provide communication scalability compared to other topologies, its simple routing and low hardware complexity make it a natural choice for this system.

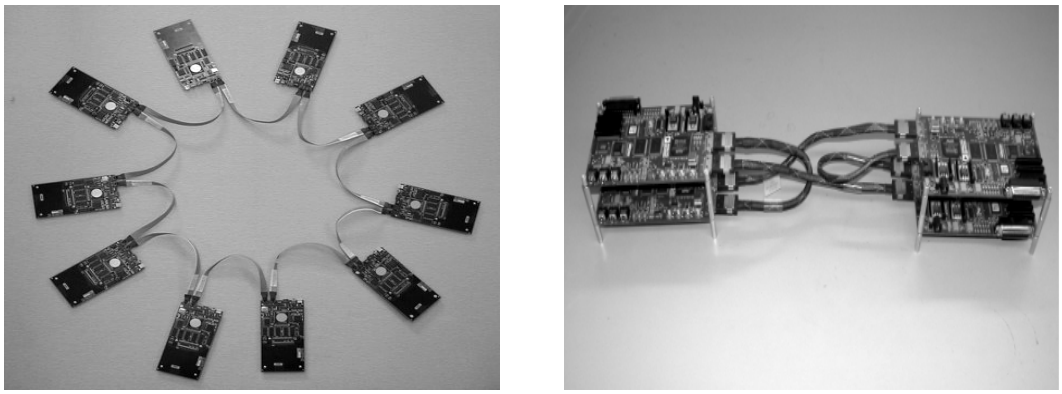
The MPI-SHARC network service was developed and optimized for this particular architecture by Kohout and George[28] to provide MPI functionality for the ADSP-21062 DSP array. Although the MPI-SHARC is a subset of the full MPI specification, the functionality and syntax are identical to the MPI for common distributed systems. The most common functions used in parallel applications were included in the design. This network service provides highly efficient collective communications since certain message-passing functions are optimized for this particular DSP architecture and communication topology.

5.2. ADSP-21160 DSP array

The second testbed consists of four EZ-KIT Lite DSP boards[29] connected to one another like the Blacktip-EX DSP boards, as shown in Fig. 6b. Each EZ-KIT board includes a single 80MHz ADSP-21160 SHARC processor[30] from Analog Devices, 512KB external memory with one wait-state access time, as well as external I/O devices. Each link port, which is composed of eight bi-directional data and two bi-directional handshaking lines, can operate at the same clock rate as the processor. It thereby provides a data transfer rate of 640Mbps, twice as fast as the ADSP-21062 processor. Similar to the Blacktip-EX DSP board, the link ports and the DMA channels on this EZ-KIT Lite board are also dedicated to produce efficient communications with the ring topology between DSP boards.

In this research, the MPI-SHARC network service for the ADSP-21062 processor has been extended to provide the MPI functions for the ADSP-21160 DSP array. The functionality and syntax of the MPI-

SHARC for ADSP-21160 are identical to the MPI-SHARC network service for ADSP-21062. This network service provides highly efficient collective communications since certain message-passing functions are designed for optimized performance in the ADSP-21160 processor architecture.



(a) (b)

Fig. 6. Experimental configurations of ADSP-21062 DSP array (a) and ADSP-21160 DSP array (b).

5.3. PC cluster

The third testbed is a Linux-based cluster of 32 PCs where each node consists of a 400MHz Intel Celeron processor with 64MB of memory. The interconnection fabric between computers is 100Mbps switched Fast Ethernet. The MPI/Pro V1.5 middleware from MPI Software Technology is used as the message-passing and synchronization layer.

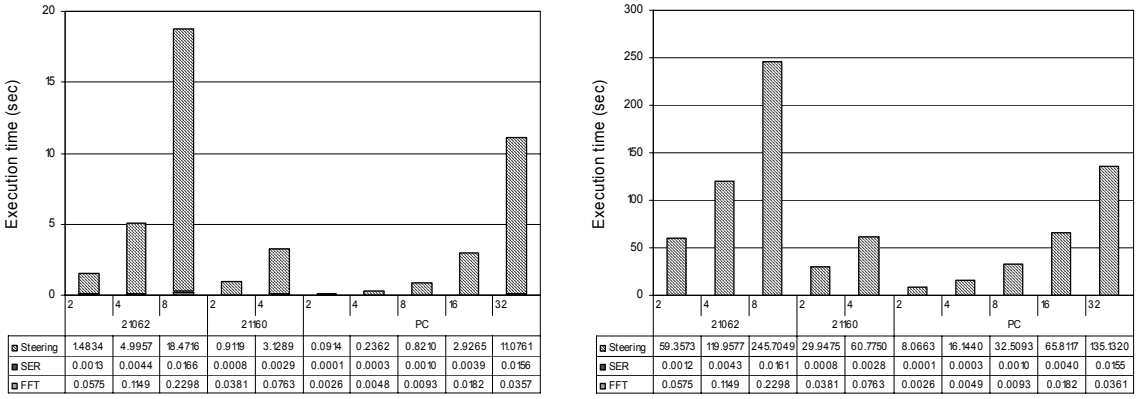
6. Performance Analysis of the MVDR-MFP Algorithm

The performance of the parallel MVDR-MFP algorithms is experimentally analyzed on the testbeds explained in the previous section. The sequential MVDR-MFP algorithm used as a baseline for comparing parallel performance is implemented with the MM and MC models. The system and problem parameters used in this experiment are 32 frequency bins, 40 grid points in range, 80 grid points in depth, and up to 32 sensor nodes. In this section, parallel performance factors are analyzed in terms of execution times, communication times, data memory requirements, scaled speedup, parallel efficiency, and result latency to demonstrate the performance effects of the parallel algorithms presented in the previous section. In these experiments, the average number of CPU clock cycles per iteration was measured for several hundreds of iterations to measure the execution and communication times for one beamforming iteration more accurately. The execution and communication times were finally calculated by multiplying the average number of CPU clock cycles by the CPU clock period for the testbed in use.

6.1. Sequential execution time

The execution time for the sequential MVDR-MFP algorithm is measured on a single processing unit in each testbed. Fig. 7a and 7b illustrate the results of sequential execution times for the MC and MM models on the three testbeds, respectively. The execution time results are examined up to 32 nodes in the PC

cluster, eight nodes in the ADSP-21062 DSP array, and four nodes in the ADSP-21160 DSP array because of their processing node limitations.



	21062			21160		PC				
Steering	1.4834	4.9957	18.4716	0.9119	3.1289	0.0914	0.2362	0.8210	2.9265	11.0761
SER	0.0013	0.0044	0.0166	0.0008	0.0029	0.0001	0.0003	0.0010	0.0039	0.0156
FFT	0.0575	0.1149	0.2298	0.0381	0.0763	0.0026	0.0048	0.0093	0.0182	0.0357

(a)

	21062			21160		PC				
Steering	59.3573	119.9577	245.7049	29.9475	60.7750	8.0663	16.1440	32.5093	65.8117	135.1320
SER	0.0012	0.0043	0.0161	0.0008	0.0028	0.0001	0.0003	0.0010	0.0040	0.0155
FFT	0.0575	0.1149	0.2298	0.0381	0.0763	0.0026	0.0049	0.0093	0.0182	0.0361

(b)

Fig. 7. Sequential execution time per iteration vs. system size for MC model (a) and MM model (b) on three testbeds.

The execution times of the sequential algorithms in the two DSP arrays are higher than those of the PC cluster because the clock rate of their processing units is much slower than their counterparts. The clock rate and architecture of the processors, and the software environments used in the three testbeds, are factors that affect the sequential execution time. For instance, although the clock rate of the ADSP-21062 is twice as slow as the ADSP-21160, the sequential execution time on the ADSP-21062 DSP board is not exactly twice as much as that of the ADSP-21160 DSP board. One reason is that there is no clock cycle delay to access the external memory on the ADSP-21062 board (0 wait state) whereas there is one clock cycle delay on the ADSP-21160 board (1 wait state).

The execution time results show that the steering stage is the most computationally dominant stage because the complexity of the steering stage is much higher than that of other stages due to the substantial number of grid points. As the number of nodes is increased, the sequential execution time in the MC model increases more rapidly than in the MM model due to the complexity of the steering stage. The steering stage includes the replica vector generation task in the MM model and excludes it in the MC model. The replica vector generation task has linear complexity in terms of the number of nodes but requires the most intensive computation of the tasks in the steering stage due to the substantial scalar operations for a large number of grid points as described in Section 3. Thus, the steering stage reveals quadratic complexity in the MC model but linear complexity in the MM model since the replica vector generation task overshadows that of other tasks in the steering stage. The sequential execution times in the MM model are much higher than in the MC model due to the inherent computational characteristics of the MC model.

6.2. Parallel execution time

The execution times of the parallel algorithms on the two DSP arrays and the PC cluster are illustrated in Fig. 8. Since the complexity of parallel algorithms is reduced by a factor of N compared to the sequential execution times as in Table 1, the parallel execution times for both the MM and MC models slowly increase as the system size is increased. The increase in parallel execution times for the MM model is slower than that for the MC model because linear growth of the sequential execution time created by the increased number of sensor nodes for the MM model is evenly distributed across multiple processors by the

parallel algorithms. However, the parallel execution times for the MC model are much lower than those for the MM model as in the sequential execution times since the MC model emphasizes efficient computation.

The parallel execution times for the ADSP-21062 DSP array, the ADSP-21160 DSP array, and the PC cluster are depicted in Fig. 8a, 8b, and 8c, respectively. The steering stage is still the most computationally dominant as in the sequential execution time even though its complexity is reduced in the parallel algorithms. The execution time results of each parallel algorithm are about the same as their counterparts on each DSP array since the communication and parallelization overheads created by each parallel algorithm are low for the given problem sizes on both the DSP arrays. The execution times of the three parallel algorithms on the PC cluster follow the same trends as those on the two DSP arrays but for the communication times. The FD algorithm handles a larger message size than both SD and IFD algorithms. Thus, communication times of the FD algorithm increase rapidly with increased system size as a result of ineffective communication on the PC cluster.

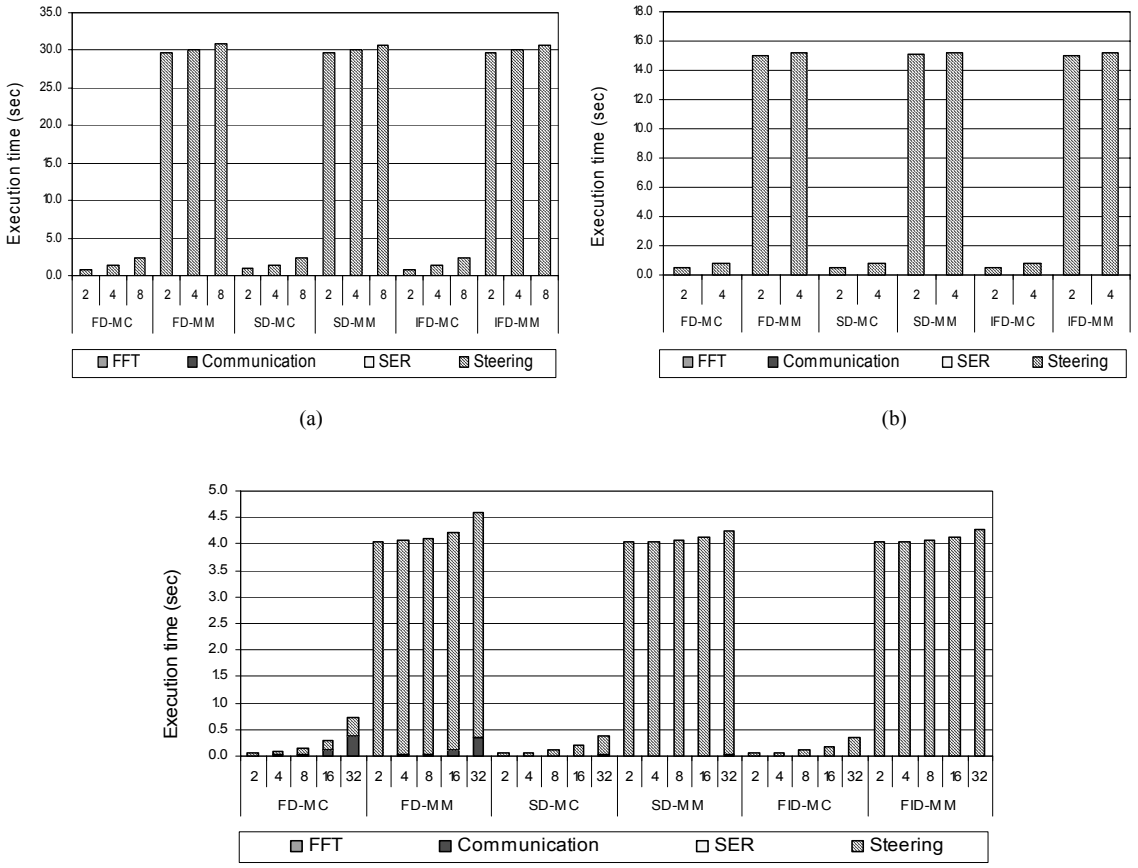


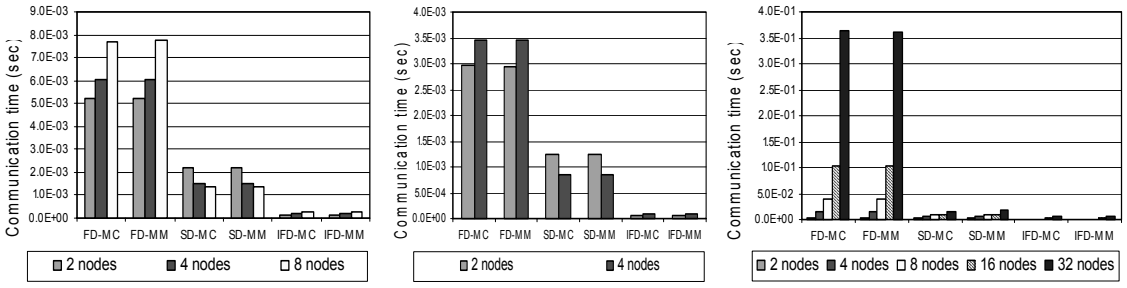
Fig. 8. Parallel execution time per iteration vs. system size on ADSP-21062 array (a), ADSP-21160 array (b), and PC cluster (c).

6.3. Communication time

Parallel execution time consists of two portions, computation time and communication time. The communication time is one of the most important factors affecting the performance of parallel algorithms

on distributed systems. Therefore, communication times for the three parallel algorithms are compared on the two DSP arrays and the PC cluster.

As shown in Fig. 9, the communication times for the FD algorithm for all testbeds are higher than those for the other parallel algorithms because of its large message size and all-to-all communication pattern. By contrast, the IFD algorithm provides the most efficient communication due to its small message size and communication frequency. On the two DSP arrays, the communication times of FD and IFD are steadily increasing with system size but those of the SD are decreasing because the effect of decreasing message size for the communication time outweighs the effect of increasing communication frequency in the SD algorithm. If the message size exceeds the maximum payload size in the MPI-SHARC network services, the message is divided into multiple packets and then sent to destination nodes. As a result, a larger message size incurs more communication time compared to more frequent communications due to the overhead in the processor's internal memory movement when using *Allgather* communication.[28] The communication time for the ADSP-21160 DSP array is about half that for the ADSP-21062 DSP array since the ADSP-21160 processor provides a communication network that is twice as fast as the ADSP-21062 processor. As the number of nodes increases, the communication times for the three parallel algorithms on the PC cluster increase rapidly unlike the two DSP arrays. Since MPI services on the two DSP arrays provide a form of hardware broadcast communication, the two DSP arrays can handle the collective communication more efficiently than the PC cluster. The greater communication time of FD compared to other parallel algorithms may result in degrading parallel performance.



(a) (b) (c)

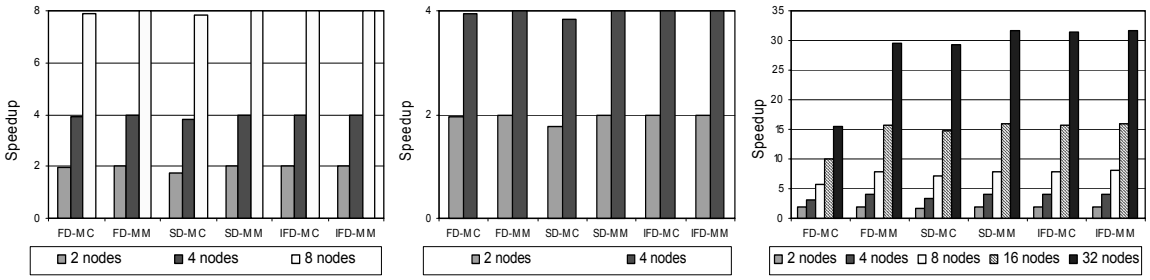
Fig. 9. Communication time per iteration vs. system size on ADSP-21062 array (a), ADSP-21160 array (b), and PC cluster (c).

6.4. Scaled speedup and parallel efficiency

Overall system performance is analyzed next in terms of scaled speedup and parallel efficiency as depicted in Figs. 10 and 11. Scaled speedup is defined as the ratio of sequential execution time to parallel execution time. This performance metric takes into account the fact that the problem size is also scaled upwards as the number of processing nodes is increased since each node has its own sensor. Parallel efficiency is the ratio of scaled speedup to ideal speedup, which is equal to the number of processors.

Fig. 10 shows the scaled speedup of the parallel algorithms measured on the three testbeds. As the results indicate, all three algorithms exhibit promising speedup with increase in system and problem size. The relative speedup results for all the algorithms on both DSP arrays are better than their counterparts on the PC cluster due to smaller communication-to-computation ratios. In both DSP arrays, the communication-to-computation ratios are smaller because they have more efficient communication channels and slower processors. The scaled speedup of the FD-MC algorithm on the PC cluster is lower than it is for the other algorithms since the FD-MC algorithm has more communication overhead and less

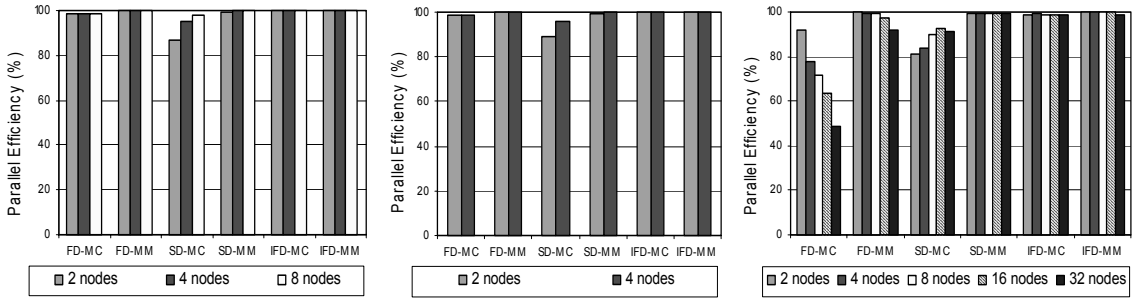
computation time than the other parallel algorithms. The IFD algorithm shows marginally better speedup than the other algorithms on the three testbeds because IFD has lower communication overhead due to its smaller message size and all-to-one communication pattern.



(a) (b) (c)

Fig. 10. Scaled speedup vs. system size on ADSP-21062 array (a), ADSP-21160 array (b), and PC cluster (c).

The parallel efficiencies for the two DSP arrays and the PC cluster are shown in Fig. 11a, 11b, and 11c, respectively. The results on the two DSP arrays show a promising parallel efficiency of between 86% and 99% for the given system sizes. However, the results on the PC cluster exhibit a lower parallel efficiency than those on the two DSP arrays since the PC cluster has less efficient communication channels and faster processors. Specifically, the parallel efficiency of FD-MC on the PC cluster rapidly decreases from 91% for two nodes to 48% for 32 nodes because the communication-to-computation ratio rapidly increases. The SD-MC algorithm on both DSP arrays shows a lower parallel efficiency than the other algorithms because the ratio of the execution time for the parallelization overheads including non-parallelized SER stage to the total execution time is relatively high in the case of smaller nodes. The parallel efficiency of the SD algorithm increases with increased system size since the overhead ratio and the communication time are decreased. The IFD algorithm achieves marginally better parallel efficiency than either FD or SD as a result of its lower communication overhead.



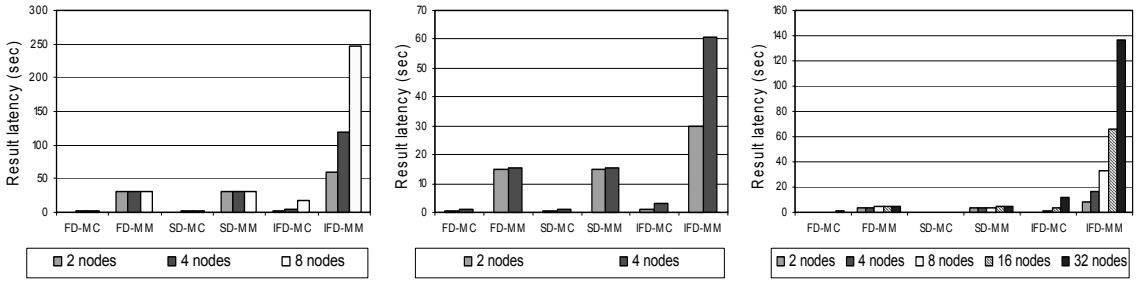
(a) (b) (c)

Fig. 11. Parallel efficiency vs. system size on ADSP-21062 array (a), ADSP-21160 array (b), and PC cluster (c).

6.5. Result latency

The result latency on two DSP arrays and the PC cluster is depicted in Fig. 12. This demonstrates that the result latency for IFD is much higher than for the other algorithms because each node in IFD is responsible for the entire processing of the assigned iteration as in the sequential algorithm. This higher result latency

is one of the drawbacks of the IFD algorithm. By contrast, the FD and SD algorithms reduce their result latency by distributing the computations of beamforming tasks over multiple processing nodes. The MC model shows far smaller result latency than the MM model due to its efficient computation as set forth in Section 3.



(a) (b) (c)

Fig. 12. Result latency vs. system size on ADSP-21062 array (a), ADSP-21160 array (b), and PC cluster (c).

6.6. Data memory requirement

Fig. 13 shows data memory requirements for the sequential and three parallel algorithms with the MM and MC models. Considering the sequential algorithm, MM needs much smaller memory capacity than MC because of its memory efficiency. For all the parallel algorithms with the exception of IFD-MC, their memory requirements are considerably reduced by distributing the necessary data across multiple processing nodes for the MC model and by using more computation time for the MM model. By contrast, IFD-MC requires the largest memory space since it needs to preserve a full set of replica vectors for the assigned iteration as in the sequential algorithm. Hence these algorithms except IFD-MC may be used on systems with memory-constrained processing nodes.

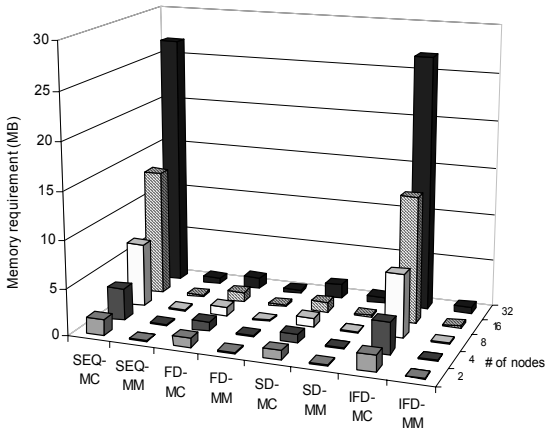


Fig. 13. Data memory requirements per node for FD, SD, and IFD algorithms.

7. Conclusions

Three parallel algorithms are developed in this paper to meet the substantial computational and memory requirements of the MVDR-MFP algorithm on real-time sonar array systems. They are based on domain decomposition methods and statically scheduled to reduce interprocessor communication and other parallelization overheads that would otherwise deteriorate performance on distributed systems. This paper focuses on performance analysis of the parallel algorithms as implemented on three potential platforms; two types of DSP arrays and a PC cluster. These testbeds are used to analyze the comparative parallel performance of three different types of processors, communication networks, and system architectures. The performance of the parallel algorithms is examined in terms of computation and communication time, scaled speedup, parallel efficiency, result latency, and memory requirement for the three different testbeds.

The three parallel algorithms exhibit promising parallel performance with increased system and problem size. All parallel algorithms with exception of the FD-MC algorithm on the PC cluster achieve between 81% and 99% parallel efficiency up to a system size of 32 nodes on the PC cluster, eight nodes on the ADSP-21062 DSP array, and four nodes on the ADSP-21160 DSP array. On the contrary, the parallel efficiency of FD-MC on the PC cluster rapidly decreases from 91% for two nodes to 48% for 32 nodes due to its substantial communication overhead. The relative speedup results of all algorithms on both DSP arrays are better than their counterparts on the PC cluster because the DSP arrays have more efficient communication channels and slower processors. For all the parallel algorithms except the IFD-MC algorithm, memory requirements are considerably reduced by distributing the necessary data across multiple processing nodes in the MC model or by using more computation time in the MM model.

With the FD algorithm, promising parallel performance is obtained with increase in system size on both DSP arrays, but not on the PC cluster due to its inefficient communication network. The FD algorithm requires a relatively small memory requirement and longer communication time than the other algorithms. Thus, the FD algorithm would be preferable for systems with a scalable communication network and a severe memory constraint such as some embedded, distributed DSP systems. The SD algorithm has better parallel performance and more efficient communication as system size increases, while at the same time requiring smaller memory capacity. Hence, the SD algorithm would be a suitable choice for systems with a larger problem size and severe memory constraints. The IFD algorithm achieves marginally better parallel efficiency than either FD or SD as a result of its lower communication overhead. However, this improvement comes at the cost of higher memory requirements and result latency. The IFD algorithm would be efficient for systems with sufficient memory for each processing unit and a relatively slow communication network. Given the increasing demands for computationally sophisticated forms of processing and data models projected for future sonar systems, these parallel algorithms provide a feasible solution for real-time and cost-effective implementation of beamforming algorithms on embedded and distributed array systems.

Further research can proceed in several directions. The parallel techniques presented in this paper can be extended to more advanced forms of beamforming algorithms such as adaptive MFP algorithms which are robust to environmental mismatch. Further work on fault-tolerant algorithms and architectures for MFP beamforming will be required in the presence of sensor element failure or processing element failure to satisfy the need for increasing dependability in distributed sonar array systems. Finally, power and energy models need to be built to analyze the power efficiency of adaptive MFP algorithms on distributed DSP systems to extend the mission time of systems operating in severe environments.

Acknowledgements

We acknowledge and appreciate the support provided by the Office of Naval Research on Grant N00014-99-10278. Special thanks go to Byung Il Koh, Jeong-Hae Han, and Seokjoo Lee in the HCS Lab at the University of Florida for their assistance with useful suggestions. This work was also made possible by donations of equipment by Analog Devices and MPI/Pro software from MPI Software Technologies, Inc.

References

1. H. P. Bucker, "Use of calculated sound fields and matched-field detection to locate sound sources in shallow water," *J. Acoust. Soc. Amer.* **59** (2), 368-373 (1976).
2. R. G. Fizell and S. C. Wales, "Source localization range and depth in an Arctic environment," *J. Acoust. Soc. Amer.* **78** (5), (1985).
3. *R. J. Vaccaro*, "The past, present, and future of underwater acoustic signal processing," *IEEE Signal Processing Magazine*, 21-51 (1998).
4. J. Capon, "High-resolution frequency-wavenumber spectrum analysis," *Proc. of the IEEE* **57** (8), 1408-1419 (1969).
5. A. B. Baggeroer, W. A. Kuperman, and P. N. Mikhalevsky, "An overview of matched field methods in ocean acoustics," *J. of Oceanic Eng.* **18** (4), 401-423 (1993).
6. A. B. Baggeroer, W. A. Kuperman, and Henrik Schmidt, "Matched field processing: Source localization in correlated noise as an optimum parameter estimation problem," *J. Acoust. Soc. Am.* **83** (2), 571-587 (1988).
7. J. M. Ozard and G. H. Brooke, "Improving performance for matched field processing with a minimum variance beamformer," *J. Acoust. Soc. Am.* **91** (1), 141-150 (1992).
8. Feuillade, W. A. Kinney, and D.R. Delbalzo, "Shallow water matched-field localization off Panama city, Florida," *J. Acoust. Soc. Am.* **88** (1), 423-433 (1990).
9. Byrne, R. T. Brent, C. Feuillade, and D. R. Delbazo, "A stable data-adaptive method for matched-field array processing in acoustic waveguide," *J. Acoust. Soc. Am.* **87** (6), 2493-2592 (1990).
10. J. L. Krolik and W. S. Hodgkiss, "Matched field localization in an uncertain environment using constraints based on sound-speed perturbations," *Proc. IEEE Oceans '91 Ocean Technologies and Opportunities in the Pacific for the '90s*, 771-778 (1991).
11. J. M. Ozard, M. J. Wilmut, D. G. Berryman, and P. Z. Zakarauskas, "Speed-up performance of a nearest-neighbors algorithm for matched-field processing with a vertical line array," *Proc. Oceans '93 Engineering in Harmony with Ocean*, 86-90 (1993).
12. H. Cox, R. M. Zeskind, and M. Myers, "A subarray approaches to matched-field processing," *J. Acoust. Soc. Amer.* **87** (1), 168-178 (1990).
13. S. Abravindan, N. Ramachandran, and P. S. Naidu, "Fast matched field processing," *IEEE J. of Oceanic Eng.* **18** (1), 1-5 (1993).
14. F. Vanpoucke and M. Moonen, "Systolic robust adaptive beamforming with an adjustable constraint," *IEEE Trans. on Aerospace and Electronic Systems* **31** (2), 658-669 (1995).
15. E. T. Tang, K. J. R. Liu, and S. A. Tretter, "Optimal weight extraction for adaptive beamforming using systolic arrays," *IEEE Trans. on Aerospace and Electronic Systems* **30**(2), 367-384 (1994).
16. A. George and K. Kim, "Parallel algorithms for split-aperture conventional beamforming," *J. of Computational Acoustics* **7**(4), 225-244 (1999).
17. A. George, J. Garcia, K. Kim, and P. Sinha, "Distributed parallel processing techniques for adaptive sonar beamforming," *J. of Computational Acoustics*, **10** (1), 1-23 (2002).

18. P. Sinha, A. George, and K. Kim, "Parallel algorithms for robust broadband MVDR beamforming," *J. of Computational Acoustics* **10** (1), 69-96 (2002).
19. K. Kim, "Parallel algorithms for distributed *in-situ* array beamforming," Ph. D dissertation, Univ. of Florida, (2001).
20. United States, Office of Scientific Research and Development, National Defense Committee, "Physics of sound in the sea," Dept. of the Navy, Headquarters Naval Material Command, Washington, 8-40 (1969).
21. M. Porter, "Acoustic models and sonar systems," *J. of Oceanic Engineering*, **18** (4), 425-437(1993).
22. M. Porter, "The KRAKEN normal mode program," SCALANT Memorandum, SM-245, September (1991).
23. M. Porter, "KRAKEN normal mode program," fttp://oalib.saic.com/pub/oalib/demo/MatMod.ps, (1997).
24. Culler and J. P. Singh, "Parallel computer architecture: A hardware/software approach," Morgan Kaufman Publishers Inc. San Francisco, CA, (1998).
25. Message Passing Interface Forum, "MPI: a message-passing interface standard," Technical Report CS-94-230, Computer Science Department. Univ. of Tennessee, April (1994).
26. Bittware Research Systems, "User's Guide: Blacktip-EX, Bittware Research Systems," Concord, NH, (1997).
27. Computer Products Division, "ADSP-2106x SHARC user's manual 2nd edition," Analog Devices Inc., Norwood, MA, (1997).
28. J. Kohout and A. George, "A high-performance network service for parallel computing on distributed DSP systems," *Parallel Computing*, accepted and in press.
29. Digital Signal Processing Division, "User's guide: ADSP-21160 EZ-KIT Lite: Hardware Rev. 2.2," Analog Devices Inc., Norwood, MA, (2000).
30. Digital Signal Processing Division, "ADSP-21160 SHARC DSP Hardware Reference 1st edition," Analog Devices Inc., Norwood, MA, (1999).