Quiet submarine threats and high clutter in the littoral undersea environment increase the processing demands on beamforming arrays, particularly for applications which require in-array autonomous operation. Whereas traditional single-aperture beamforming approaches may falter, the Split-Aperture Conventional Beamforming (SA-CBF) algorithm can be used to meet stringent requirements for more precise bearing estimation. Moreover, by coupling each transducer node with a microprocessor, parallel processing of the split-aperture beamformer on a distributed system can glean advantages in execution speed, fault tolerance, scalability, and cost. In this paper, parallel algorithms for SA-CBF are introduced using coarse-grained and medium-grained forms of decomposition. Performance results from parallel and sequential algorithms are presented using a distributed system testbed comprised of a cluster of workstations connected by a high-speed network. The execution times, parallel efficiencies, and memory requirements of each parallel algorithm are presented and analyzed. The results of these analyses demonstrate that parallel in-array processing holds the potential to meet the needs of future advanced sonar beamforming algorithms in a scalable fashion.

1. Introduction

Sonar beamforming is a class of array processing that optimizes an array gain in a direction of interest to detect the movement of hulls and propellers in water, which create signals over a wide frequency band. The determination of the direction of arrival relies on the detection of the time delay of the signal between sensors. Incoming signals are steered by complex-number vectors. If the beamformer is properly steered to an incoming signal, the multi-channel input signals will be amplified coherently, maximizing power in the beamformed output; otherwise, the output of the beamformer is attenuated to some degree. Thus, peak points in the beamforming output indicate directions of arrival for sources.

Performance of a beamformer depends on several factors. One of the important elements of concern is node configuration. Previous research has shown that uniform sensor spacing is not the best choice from the point of view of minimizing bearing error.\textsuperscript{1,2} Cramer-Rao lower bound (CRLB) analysis, which is used to set an absolute lower bound on the bearing error, determines the optimum positioning for the sensors of a linear array in order to obtain optimum estimates for direction of arrival. By using split-aperture conventional beamforming (SA-CBF) on linear arrays, the lower bound for bearing error as estimated by CRLB analysis can be approached, yielding more improved bearing estimation than single-aperture arrays.

As advancements in acoustics and signal processing continue to result in beamforming algorithms better able to cope with quiet sources and cluttered environments, the computational requirements of the algorithms also rise, in some cases at a pace exceeding that of conventional processor performance. Moreover, as the number of sensors increases, so too does the problem size associated with these algorithms. To implement modern beamforming algorithms in real-time, considerable processing power
is necessary to cope with these demands. A beamformer based on a single front-end processor may prove insufficient as these computational demands increase; thus, a number of parallel approaches have been proposed in order to overcome the limits of a single high-end processor in beamforming applications. Several projects from the Naval Undersea Warfare Center (NUWC) involve the development of real-time sonar systems by exploiting massive parallelism. The delay-and-sum beamformer has been mapped by Salinas and Bernecky to a MasPar Single Instruction Multiple Data (SIMD) architecture. The MasPar machine was also used by Dwyer to develop a real-time active beamforming system. Zvara built a similar system on the Connection Machine CM 200 architecture. Other work has concentrated on a variety of adaptive algorithms on such parallel systems as systolic arrays, SIMD multiprocessors, and DSP multicomputers. The work presented in this paper extends this knowledge base of parallel beamforming, particularly with respect to split-aperture sonar processing for in-array beamforming. An in-depth performance analysis of the stages of a sequential version of SA-CBF is shown in order to examine the sequential bottlenecks inherent in the system. In addition, new parallel algorithms for SA-CBF are designed for use with intelligent distributed processing arrays, and their performance is analyzed.

Most of the computations in beamforming consist of vector and matrix operations with complex numbers. The regularity in the patterns of these calculations simplifies the parallelization of the algorithms. Two parallel versions of SA-CBF have been developed: iteration-decomposition and angle-decomposition techniques. Iteration decomposition, which is a form of control parallelism, is a coarse-grained scheduling algorithm. An iteration is defined as one complete loop through the beamforming algorithm. A virtual front-end processor collects the input data set from each sensor. This virtual front-end then proceeds to execute a complete beamforming algorithm independently of the operation of the other nodes. Other processors are concurrently executing the beamforming algorithm with different data sets collected at different times. Angle decomposition, a form of data parallelism, is a medium-grained scheduling algorithm in which different steering angle jobs for the same data set are assigned to different processors. Application of these methods to the SA-CBF algorithm extends the original development of algorithms for control and domain decomposition in traditional single-aperture beamforming.

A theoretical background of SA-CBF is presented in Section 2 with a focus on digital signal processing. A sequential version of the SA-CBF algorithm is given in Section 3. Two different memory models of the sequential algorithm are shown to determine a baseline for the parallel algorithms. In Section 4, two parallel SA-CBF algorithms are presented. In Section 5, the performance of the parallel SA-CBF algorithms, in terms of execution time, speedup, efficiency, and memory requirements are shown. Finally, a summary of the strengths and weaknesses of the algorithms and a discussion of future research are presented in Section 6.

2. Overview of Split-Aperture Beamforming

SA-CBF is based on single-aperture conventional beamforming in the frequency domain. The beamforming array is logically divided into two sub-arrays. Each sub-array independently performs conventional frequency-domain beamforming using replica vectors on its own data. The two sub-array beamforming outputs are cross-correlated to detect the time delay of the signal for each steering angle. The cross-correlated data, with knowledge of the steering angles and several other parameters, will map the final beamforming output.

Unlike the single-aperture beamforming algorithm, the SA-CBF algorithm does not need to steer at every individual desired angle in the steering stage. The cross-correlation creates some redundant information between the adjacent sub-array steering angles. By cross-correlating to time delays slightly offset from the sub-array steering delays, each sub-array steering angle can be used to generate a range of the time delay plot. The discrete cross-correlation function is defined in Eq. (2.1):
\[ c_{xy}(n) = \frac{1}{N} \sum_{i=0}^{N-1} x(i)y(i+n) \quad \text{Fourier Transform} \quad \Leftrightarrow \quad C_{xy}(k) = X(k)Y(k)^* \quad (2.1) \]

\[ n = 0, 1, ..., N - 1 \quad k = 0, 1, ..., 2N - 2 \]

where vectors \( x \) and \( y \) are the sub-array beamforming outputs, \( N \) is the number of samples in \( x \) and \( y \), and operator * indicates complex conjugation. We are only interested in the small number of angles or time delays in the cross-correlation adjacent to the beamforming angle of the sub-array.

Before the inverse Fourier transform is applied to obtain the cross-correlation as a function of time delay, the Smoothed Coherent Transform (SCOT) is used to prefilter the cross-correlation. The spectral whitening obtained by SCOT results in an improved signal-to-interference ratio and an enhancement of the correlation between the two sub-arrays. Additional advantages of SCOT, as well as other prefiltering techniques, can be found in Ferguson.12 The SCOT weighting function is given by Eq. (2.2).

\[ |W(k)|^2 = \frac{1}{\sqrt{|X(k)|^2|Y(k)|^2}} \quad (2.2) \]

SCOT is accomplished either by taking the instantaneous magnitude of the cross-correlation in frequency or a running average of the magnitude.

As mentioned previously, each beam formed by the sub-arrays can be used to calculate multiple points in plotting the output bearing. The process by which this increase in resolution is accomplished is called \( \tau \)-interpolation. For each output angle, \( \tau \)-interpolation works with only the two cross-correlation results that are nearest to the desired angle. Raised-cosine weights are used to calculate the beamforming output for the interpolated angle from a linear combination of the two adjacent cross-correlation values. Because we need only a limited range of cross-correlation values to calculate the final beamforming output, the inverse discrete Fourier transform is then performed using Eq. (2.3) rather than the conventional FFT algorithm. This method will decrease computational cost of the inverse Fourier transform stage.

\[ c(\tau_j) = \frac{1}{N_{\text{FFT}}} \sum_{m=M_1}^{M_2} 2 \text{Re}[\tilde{C}_m e^{j2\pi (m-1)\Delta f_j}] \quad (2.3) \]

In Eq. (2.3), \( M_1 \) and \( M_2 \) are the minimum and maximum frequency bin numbers in which we are interested, \( \tilde{C}_m \) is the weighted and normalized frequency-domain cross-correlation, \( \Delta f \) is the frequency resolution of the FFT, \( \tau \) is the time delay between phase centers, and \( \text{Re}(\cdot) \) is the function that returns the real value of a complex number.

The two nearest cross-correlation vectors, subscripted as \( c_L(\tau_j) \) and \( c_R(\tau_j) \), are used to evaluate each interpolated output angle via \( \tau \)-interpolation as defined in Eq. (2.4) and Eq. (2.5).

\[ c(\theta_{\text{out}}) = h_L c_L(\tau_j) + h_R c_R(\tau_j) \quad (2.4) \]

\[ h_L = \frac{1}{2} \left[ 1 + \cos \pi \left( \frac{\theta_{\text{out}} - \theta_L}{\theta_L - \theta_R} \right) \right] \quad ; \quad h_R = 1 - h_L \quad (2.5) \]

In these equations, \( \theta_L \) and \( \theta_R \) are the sub-array beamformed angles to the left and right, respectively, of the output interpolated angle \( \theta_{\text{out}} \), and \( c_L(\tau_j) \) and \( c_R(\tau_j) \) are cross-correlation values for angles \( \theta_L \) and \( \theta_R \), respectively. The final output of the SA-CBF beamformer finds \( c(\theta_{\text{out}}) \) versus each interpolated \( \theta_{\text{out}} \). Further information on the design of the sequential SA-CBF algorithm can be found in Machell10 and the experimental evaluation of SA-CBF can be found in Stergiopoulos11. To consolidate the above
processes, Fig. 1a shows the block diagram of the SA-CBF algorithm, and Fig. 1b is a sample output of the SA-CBF.

![Block diagram of the SA-CBF algorithm](image)

Fig. 1. Block diagram of the sequential SA-CBF algorithm for 8 nodes (a), and a sample output of the SA-CBF for 8 nodes with a source at $\theta = 30^\circ$ (b). Arrows in the block diagram indicate data stream vectors.

It is often highly desirable to increase the number of input nodes and the number of steering angles in a beamforming system. By increasing the number of input sensors, the beamformer resolves finer angles because the main lobe of the beam pattern narrows and the side lobes of the beam pattern decrease. Fig. 2a, which is constructed using a polynomial representation of a linear array$^{13}$, shows this result. Even if there are enough nodes to obtain a sharp beam pattern, the number of steering angles remains an important factor in producing high-resolution beamforming output. The number of steering angles decides the number of output points of the beamformer; thus, a small number of steering angles may cause a blurry beamforming output.

![Normalized beam pattern and number of multiplication operations](image)

Fig. 2. Normalized beam pattern of the SA-CBF as a function of the number of nodes and $\sin(\theta)$ (a), and the required number of multiplication operations as a function of the number of steering angles and the number of nodes (b).
As both of these parameters increase, the number of multiplication operations required to generate beamforming output is increased rapidly, as shown in Fig. 2b. According to this figure, powerful processing is essential to generate high-resolution beamforming output with acceptable latency and throughput. In cases where current technology cannot provide sufficient real-time performance in a single processor, a trade-off will be required between response time and resolution. The scalable performance of parallel processing will help to overcome limits imposed by a single front-end processor.

3. Performance Analysis of Sequential Split-Aperture Beamformers

The SA-CBF algorithm previously discussed can be split into five distinct stages: FFT, Steering, Sub-array Summation, Cross-correlation/SCOT, and Inverse Fourier Transform/Interpolation. High-level pseudo-code for SA-CBF is shown in Fig. 3. Up to and including the sub-array summation, SA-CBF is a frequency-domain conventional beamforming algorithm except that there are two phase centers. The succeeding stages improve the visual resolution and bearing estimation of the beamformer output using DSP techniques.

\begin{verbatim}
Do FFT for every incoming signal vector;
For j=1, number of steering angles
    Steering(j);
    Sub-array summation(j);
    Cross-correlation and SCOT(j);
End
For k=1, number of output angles
    Inverse Fourier transform and interpolation(k);
End
\end{verbatim}

Fig. 3. Pseudo-code for the sequential SA-CBF algorithm.

In building a baseline for the parallel SA-CBF algorithms, two different sequential implementations were created, one using a minimum-memory model and the other using a minimum-calculation model. The minimum-memory model tries to save memory by doing more calculations, and the minimum-calculation model saves redundant calculations by using more memory. To illustrate the trade-off involved in selecting one model over the other, consider the steering vectors and the inverse Fourier transform basis. Under normal operation, these vectors are not subject to change from iteration to iteration. In the minimum-memory model, these space-consuming vectors are computed on the fly for every iteration. However, in the minimum-calculation model, the vectors are already calculated in an initial phase and saved into special memory locations to access easily whenever needed without recalculation. Thus, the minimum-calculation model compromises memory space for faster execution time. In the event of node failures, the minimum-calculation model will be forced to recalculate all values in these vectors. Node failure increases the distance between nodes so new steering vectors need to be established based on new parameters. A new inverse Fourier transform basis is also necessary when the processing frequency bins are changed. By contrast, the minimum-memory model encounters significantly fewer disturbances in the event of failures since it would have recalculated all values in any case.
Due to the recalculation in the steering and inverse Fourier transform stages, execution times of these stages are dramatically increased for the minimum-memory model. Conversely, required memory space for these stages is significantly smaller than that required in the minimum-calculation model. Experiments were conducted to examine the execution time and memory requirements of the two sequential models. The platform used was a SPARCstation-20 workstation with an 85MHz SuperSPARC-II processor and 64MB of memory, and running Solaris 2.5. The experimental results in Fig. 4 show that execution time of the minimum-calculation model is five times less than that of the minimum-memory model with twice the memory required. The execution time and memory requirement of the FFT stage are the same for both models due to the fact that the same FFT algorithm is used in both. The sub-array summation and cross-correlation/SCOT stages are also unaffected by the model since these stages have no vectors that are the same from iteration to iteration.

The model selected for the baseline of the performance analysis of parallel algorithms depends on the focus of the study. In the next section, to estimate attainable speedup with each of the parallel programs, execution time is the most important factor to be measured. Therefore, the minimum-calculation model is preferred as the sequential baseline. All parallel algorithms are implemented with the same minimum-calculation model, and it is assumed that each processor has sufficient memory to hold the program and all data.

4. Parallel Algorithms for Split-Aperture Beamformers

The parallel algorithms in this paper were designed to operate in conjunction with a distributed, parallel, sonar system architecture. This architecture is composed of intelligent nodes connected by a network. Each of the smart nodes, comprised of a hydrophone and a microprocessor, has its own processing power as well as requisite data collection and communication capability. By using such a distributed array architecture, the algorithmic workload is distributed and cost is reduced due to the elimination of an
expensive front-end processor. Such an architecture ties the degree of parallelism (DOP) to the number of physical nodes in the target system. Of course, an increase in the number of nodes will increase the amount of input data and thus the problem size.

The best performance of a parallelized task is achieved by minimizing processor stalling and communication overhead between processors. With a homogeneous cluster of processors, dividing tasks evenly among processors serves to maximize performance by reducing these hazards. While task-based parallelism is possible with the SA-CBF algorithm (via assigning steering to one node, sub-array summation to another node, etc.), the workload would not be homogeneous and would result in degraded performance. Fig. 5a shows this unbalanced workload amongst the various sequential tasks and serves as a justification for not using task-based parallelism.

The SA-CBF algorithm computes many vector and matrix operations using nested loops. Therefore, we can partition iterations of the outer loop across the processors to create a balanced workload for each processor. When loop partitioning is applied to the parallel SA-CBF algorithm, special attention is required because there are two external loops that are repeated: number of steering angles and number of output angles, as shown in Fig. 3. These loops run separately within the SA-CBF algorithm but their information is tightly coupled and must be used together to generate the final beamforming output. If we parallelize only the first loop and not the second, which includes the inverse Fourier transform and interpolation stages, severe performance slowdown would result as the number of processors increases. Fig. 5b shows how loop partitioning of the stages in the first loop results in the second loop becoming a bottleneck. As the number of processors increases, the execution time of each stage decreases linearly except that of the IFT/Interpolation stage since this stage is not parallelized. Since the total execution time of the beamformer in this figure is found by summing the execution times of the stages, it becomes
clear that the execution time of the IFT/Interpolation stage will become increasingly dominant as the number of processors increases. Of course, according to Amdahl’s law, a small number of sequential operations can significantly limit the speedup achievable by a parallel program. Thus, in this case the sequential bottleneck caused by the IFT/Interpolation stage limits the speedup to no more than 5 for an 8-node configuration or 9 for a 32-node configuration. Though parallelization of the IFT/Interpolation stage is nontrivial due to strong dependencies with previous stages, it will significantly increase the efficiency of the overall algorithm.

The two parallel algorithms presented in the rest of this section make use of loop partitioning in two different ways. Furthermore, these algorithms parallelize the second loop (i.e., the IFT/Interpolation stage) to achieve better efficiency. The next two subsections present an overview of the two parallel algorithms, followed by performance results in Section 5.

### 4.1. Iteration-decomposition method

The first decomposition method involves the partitioning of iterations, the solutions of a complete beamform cycle. Iteration decomposition is a technique whereby multiple beamforming iterations, each operating on a different set of array input samples, are overlapped in execution by pipelining. The algorithm follows the tradition of overlapped concurrent execution pipelining, where one operation does not need to be completed before the next operation is started. The beamforming task for a given sample set is associated with a single node in the parallel system. Other nodes work concurrently on other iterations. Pipelining is achieved by allowing nodes to collect new data from the sensors and begin a new iteration before the current iteration is completed. At the beginning of each iteration, all nodes stop processing the beamforming iterations assigned them just long enough to execute the FFT on their own newly collected samples and send the results to the node assigned the new iteration. Once this data has been sent, all nodes resume the processing of their respective iterations. Using this pipelining procedure, there are as many iterations currently being computed as there are processors in the system, each at a different stage of completion. A block diagram illustrating this algorithm in operation on a 4-node array is shown in Fig. 6.

![Block diagram of the iteration-decomposition algorithm in a 4-node configuration. Solid arrows indicate inter-processor, all-to-one communication and shaded boxes represent independent complete beamforming cycles (i.e. iterations).](image)

Individual beamforming is separated by inter-processor communication stages and each processor takes responsibility for a different beamforming job. A number of difficulties exist for iteration decomposition. First, since every partial job is synchronized at the communication points, an unbalanced processing load can develop across nodes, which may lead to processor stalling. Second,
each iteration of the beamforming algorithm must be completed by a node before its pipeline cycle is complete so as to avoid collision between jobs. Therefore, to maximize the performance of the iteration-decomposition algorithm, the beamforming jobs should be evenly segmented by the number of processors. The pseudo-code illustrating the basic algorithm followed by each processor is shown in Fig. 7.

For t=1, Total number of iterations
   For j=1, number of processors  //beamforming is divided by # of processors
      index=(j+my_rank)%(number of processors);  //my_rank is node number
      Do FFT for their own node data;
      Communicate with other nodes;
      For k=start_steering_angle(index), end_steering_angle(index)
         Steering(k);
         Sub-array summation(k);
         Cross-correlation and SCOT(k);
      End
      For i=start_output_angle(index), end_output_angle(index)
         Inverse Fourier transform and interpolation(i);
      End
   End
End

Fig. 7. Pseudo-code for the iteration-decomposition algorithm.

Each processor calculates an index based on its node number, the current job number, and the number of nodes. This index is used to access arrays which tell the node from which point in its iteration it must continue after executing the FFT and communicating new data and at which point it must again pause in order to begin another new iteration. Specifically, arrays containing the starting steering angle and ending steering angle for a computation block instruct the node on how to partition the first of the two loops. Arrays containing the starting output angle and ending output angle are used to decompose the second loop. Upon completion of this procedure, a new iteration is started, and each node calculates a new portion of the necessary steering and output angles for its iteration. Managing these arrays requires a nontrivial amount of overhead, but such overhead is a necessary part of the correct operation of the pipelining method.

4.2. Angle-decomposition method

The second parallel algorithm decomposes SA-CBF using a medium-grained approach in which the internals of a complete beamforming iteration are segmented. The angle-decomposition algorithm distributes processing load by decomposing the domain, the steering angles. Each node calculates the SA-CBF results for a certain number of desired steering directions from the same sample set. Before doing so, all participating nodes must have a copy of the data from all other nodes. After completing this all-to-all communication, each node computes different beamforming angles for the same data. This algorithm introduces considerably more communication than the iteration-decomposition algorithm, and the interconnection scheme between processors will have a more significant effect on performance. The communication requirements are further studied in George. A block diagram illustrating this algorithm in operation on a 4-node array is shown in Fig. 8.
Fig. 8. Block diagram of the angle-decomposition algorithm in a 4-node configuration. Solid arrows indicate inter-processor communication and shaded boxes represent independent beamforming jobs.

For the purposes of decomposing the two loops, four variables that indicate beginning and ending steering and output angles are calculated in an initial phase. These four variables serve much the same purpose as the arrays in the iteration-decomposition method, though for angle decomposition these values are a function only of the node number. The steering direction and the output angle on which a node is to begin computing are determined by that node’s relative location from a virtual front-end node. The number of steering directions a node is to compute is based on dividing the total number of desired steering directions by the number of nodes. The number of output angles per node is also derived from the steering direction information. Fig. 9 shows the pseudo-code for this approach. After a node is finished computing the results for its steering directions, it must communicate them to a specially designated node for final collection. This collection node can be fixed or, to provide fault tolerance, free-floating perhaps via round-robin scheduling. Although this algorithm involves a more complex communication mechanism best served with a broadcasting network, it does not require the additional overhead necessary to manage pipelining as does the iteration-decomposition method.

```
Calculate angle information based on the node number
For t=1, total iteration number
    Do FFT for their own node data;
    Communicate with other nodes;
    For k=my_start_steering_angle, my_end_steering_angle
        Steering(k);
        Sub-array summation(k);
        Cross-correlation and SCOT(k);
    End
    For i=my_start_output_angle, my_end_output_angle
        Inverse Fourier transform and interpolation(i);
    End
    Collect result from other nodes;
End
```

Fig. 9. Pseudo-code for the angle-decomposition algorithm.
5. Performance Analysis of Split-Aperture Beamformers

In order to understand the strengths and weaknesses of the two parallel algorithms, their performance characteristics were measured on a physical multicomputer testbed. The results of these experiments are presented in this section. The testbed used consists of a cluster of SPARCstation-20 workstations connected by a 155-Mbps (OC-3c) Asynchronous Transfer Mode (ATM) network.

The algorithms were implemented via message-passing parallel programs written in C-MPI (Message-Passing Interface)\textsuperscript{15}. MPI is a library of functions and macros for message-passing communication and synchronization that can be used in C/C++ and FORTRAN programs. In the program code, we call a time-check function at the beginning of a stage and save the return value. After each stage we call the function again, and subtract the earlier return value from the new return value. The difference is the execution time of the stage. In order to obtain reasonable and reliable results, all parallel and sequential experiments were performed 500 times and execution times were averaged.

The first experiment involves the execution of the sequential SA-CBF algorithm on a single workstation, where the number of sensors is varied to study the effects of problem size. The results are shown in Fig. 10. Execution times of the FFT, steering, and sub-array summation stages increase linearly with an increase in sensors. The number of sensors determines the number of data stream vectors in Fig. 1a; therefore, the processing load of these stages is greater with increased numbers of sensors. Conversely, the number of data stream vectors remains constant after the sub-array summation stage. No matter how many data stream vectors enter the sub-array summation stage, the number of output data stream vectors is always two. Furthermore, after interpolation, only one data stream vector is left. Thus, the execution times of Xcorr/SCOT and IFT/Interpolation stages remain fixed as the number of sensors is increased.

![Fig. 10. Average execution time per iteration as a function of array size for the sequential SA-CBF algorithm with 500 iterations on the SPARC20/ATM cluster.](image)

For each of the two parallel algorithms, Fig. 11 shows average execution times for three system sizes: 4, 6, and 8 nodes. For iteration decomposition, the execution time shown represents the effective execution time and not the result latency. As was previously shown for a 4-node configuration, the results for a given beamforming cycle are output after four pipeline stages. In fact, as is typical of all
pipelines due to the overhead incurred, this result latency is longer than the total execution time of the sequential algorithm. Instead, the figure plots the effective execution time, which represents the amount of time between outputs from successive iterations once the pipeline has filled. In the angle-decomposition case, we can measure the execution time directly because there is no overlap of beamforming jobs.

Fig. 11. Average execution time per iteration as a function of array size for the parallel SA-CBF algorithms with 500 iterations on the SPARC20/ATM cluster

As seen in Fig. 11, the execution times of the FFT, steering, and sub-array summation stages do not change significantly as the number of nodes increases. As mentioned earlier, the distributed sonar architecture uses smart nodes at each input sensor; therefore, the number of data stream vectors is identical to the number of processors. The additional workload caused by increasing the number of nodes is evenly distributed across the processors. Therefore, the number of nodes does not influence the execution time of these stages. However, the execution times for the Xcorr/SCOT and IFT/Interpolation stages decrease as the number of nodes and processors increase. In these stages, each processor does less work as the number of nodes increases because the workload from a fixed number of data stream vectors is divided among the processors. In spite of a growing problem size, the overall computation times (i.e. execution time minus communication time) of both decomposition algorithms decline as the number of nodes increases.

In this experiment, communication time is defined as the time spent in communication function calls such as MPI_Send and MPI_Recv. The communication pattern can be shown with the graphical profiling program Upshot. A sample Upshot output is shown in Fig. 12, which displays a portion of the profiling log collected from both parallel algorithms running on four nodes in the testbed. In this figure, communication blocks are represented by rectangles defined in the legend, and computation blocks are represented by the horizontal lines between successive communication blocks. Iteration decomposition uses a fairly simple communication pattern but angle decomposition communicates an all-to-all message at the initial phase of each beamforming job. Communication time of iteration decomposition and collection time of angle decomposition have little contribution to total execution time. However, with angle decomposition, the size of the first data communication of each iteration increases rapidly with the number of nodes. With this increase in communication comes an increase in
the MPI overhead, an increase in network contention, and poorer performance, which eventually comes
to dominate the total execution time. Clearly, the relatively small amount of communication in iteration
decomposition is an advantage for that algorithm. Fig. 11 indicates that total execution time of iteration
decomposition decreases and total execution time of angle decomposition increases with an increasing
number of nodes.

Fig. 12. Upshot profiles for both of the parallel SA-CBF algorithms with 4 nodes. The upper snapshot shows a profile from the
iteration decomposition, while the lower snapshot shows a profile from the angle decomposition.

Fig. 13a shows the scaled speedup of the two decomposition methods over the testbed cluster of
SPARCstation-20 workstations. The baseline for comparison is the sequential SA-CBF algorithm
running on one workstation of the same testbed. Since the algorithms incur additional workload as
sensors are added, the plots show scaled speedup. Fig. 13b displays scaled efficiency which is defined
as scaled speedup divided by the number of processors used. Together these figures show the
pronounced effect of the communication overhead in the angle-decomposition method, whereas iteration
decomposition exhibits near-linear scaling. Improvement of the performance of angle decomposition
may be achieved by employing more complex network architectures, such as broadcast-efficient
networks, and by implementing more robust communication pipelining. Unfortunately, these methods
may by impractical to implement on a distributed sonar array topology with limited communication
capabilities.
In Section 3, we chose the minimum-calculation model as the baseline for this investigation. In this model, the majority of the memory requirement arises from the steering stage, as shown in Fig. 4. Iteration decomposition requires the full amount of steering-vector and steered-signal storage because each processor implements a whole beamforming task for an incoming data set. By contrast, angle decomposition needs only part of the memory space for steering since individual processors generate only part of the beamforming result for a given data set. For both the sequential algorithm and iteration decomposition, the demands for memory space for the steering stage grow linearly when the number of nodes is increased, as shown in Fig. 14. However, little change is observed for angle decomposition. These results illustrate the significant trade-off of execution time versus memory requirements. Despite the fact that iteration decomposition shows considerably better performance than angle decomposition, angle decomposition may be preferred when memory requirements are stringent.

Fig. 13. Scaled speedup (a) and scaled efficiency (b) as a function of the number of processors for the iteration-decomposition and angle-decomposition algorithms.

Fig. 14. Memory requirement of the steering stage as a function of the number of processors for both parallel algorithms. The memory requirements for the sequential algorithm are comparable to those of iteration decomposition.
6. Conclusion

The iteration-decomposition algorithm distributes its job in time space with overlap between processors, and the angle-decomposition algorithm parallelizes its job in processor space. The iteration-decomposition algorithm for SA-CBF shows more than 80 percent scaled efficiency. However, as the number of nodes is increased, the performance of the angle-decomposition method begins to degrade due to inefficiency in the communication stages. In fact, the communication time is the most significant difference between the two parallel algorithms, whereas little difference is observed in computation times. Of the two algorithms, iteration decomposition would be the better choice in architectures with enough memory for each processor to accommodate the large memory requirements. Furthermore, due to the limited amount of communication in iteration decomposition, it would also be well suited for architectures with a low-performance network. By contrast, for a system with a higher-performance network but restricted on memory capacity, angle decomposition may be the better choice. Because angle decomposition uses an all-to-all communication in each iteration, an architecture using an efficient broadcast network would also improve its performance relative to iteration decomposition.

The parallel beamforming techniques described in this paper present many opportunities for increased performance, reliability, and flexibility in a distributed parallel sonar array. These parallel methods provide considerable speedup with multiple nodes, thus enabling previously impractical algorithms to be implemented in real time. Furthermore, the fault tolerance of the sonar architecture can be increased by taking advantage of the distributed nature of these parallel algorithms and avoiding single points of failure. Future work will involve parallelizing more intricate computations, including the advanced matrix manipulations needed for adaptive beamforming algorithms and matched-field processing. With the additional complexities of these algorithms over conventional algorithms, parallel and distributed systems and software such as those described in this paper will be necessary to provide sufficient performance. The decompositions examined here not only provide important parallelization of the split-aperture algorithm, but also can serve as the foundation for these more complicated sonar signal processing algorithms.

Acknowledgements

The support provided by the Office of Naval Research on grant N00014-98-0188 is acknowledged and appreciated.

References