

# PARALLEL SUBSPACE PROJECTION BEAMFORMING FOR AUTONOMOUS, PASSIVE SONAR SIGNAL PROCESSING

KEONWOOK KIM and ALAN D. GEORGE

*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*

Received 25 Aug 2001

Revised 8 July 2002

Adaptive techniques can be applied to improve performance of a beamformer in a cluttered environment. The sequential implementation of an adaptive beamformer, for many sensors and over a wide band of frequencies, presents a serious computational challenge. By coupling each transducer node with a microprocessor, in-situ parallel processing applied to an adaptive beamformer on a distributed system can glean advantages in execution speed, fault tolerance, scalability, and cost. In this paper, parallel algorithms for Subspace Projection Beamforming (SPB), using QR decomposition on distributed systems, are introduced for in-situ signal processing. Performance results from parallel and sequential algorithms are presented using a distributed system testbed comprised of a cluster of computers connected by a 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-situ processing holds the potential to meet the needs of future advanced beamforming algorithms in a scalable fashion.

## 1. Introduction

The conventional beamforming (CBF) algorithm is quite vulnerable to the noise of incoming data due to the substantial power of the sidelobes in the beampattern known as “spectral leakage.” Frequently, the CBF output in the look direction can be contaminated by unwanted signals that come from directions other than the look direction. Since the level of sidelobes and the width of the mainlobe are inversely proportional to the number of nodes, the CBF algorithm generates a very wide and smooth output with a limited number of nodes and, as a result, a large number of nodes is required for the CBF to resolve the direction of arrival (DOA) with precise accuracy. Adaptive beamforming (ABF) techniques improve the performance of the beamformer by maximizing the signal and minimizing noise power of steered direction output. With certain constraints, the ABF algorithm optimizes the cost function for better beamforming output with narrower and sharper peaks. These improvements generally require extra manipulations to compute the data-dependent weights. Many ABF algorithms have been proposed in the literature<sup>1-4</sup>, but this paper focuses on subspace beamforming algorithms such as Multiple Signal Classification (MUSIC), as presented by Schmidt.<sup>5</sup>

Subspace beamforming algorithms exploit properties of eigenstructures to provide a solution to an underlying estimation problem for a given observed process. The significant attention given to the subspace approach in the literature is primarily due to the introduction of the MUSIC algorithm. The performance improvement of the MUSIC algorithm was so significant that it became an alternative to most existing methods.<sup>6</sup> Stoica<sup>7</sup> mathematically analyzed the performance and derived the Cramer-Rao Bound (CRB) of the MUSIC method.

The MUSIC beamformer requires eigendecomposition of the Cross-Spectral Matrix (CSM) to separate the signal and noise subspaces. One method for eigendecomposition is Singular Value Decomposition (SVD); however, the computation of the SVD is intensive to perform. To implement the MUSIC beamforming algorithm 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, several approaches to parallelization have been proposed. Hsiao<sup>8</sup> proposed parallel SVD algorithms using a systolic array architecture known as the “Coordinate Rotation Digital Computer” (CORDIC). Robertson<sup>9</sup> built a MUSIC beamforming algorithm for a pipelined systolic array. Both approaches use iterative methods to compute eigenvalues and eigenvectors of the input data. In a parallel implementation, iterative methods are not desirable since the convergence rate of the algorithm depends on the data characteristics and the distribution of the singular values. This unpredictable workload may develop into an unbalanced workload between processors and lead to performance degradation of the parallel algorithms. For that reason, both algorithms use a strategy of stopping after a predetermined number of iterations based on some experimentation. In a scalable system, however, where the number of sensors is changed by fault or task requirement, the altered problem size may affect the predetermined number of iterations. Instead of using the SVD algorithm, Smith<sup>10</sup> proposed SPB using row-zeroing QR decomposition, which has fixed computational load.

The work presented in this paper extends the SPB algorithm for parallel in-situ processing on a linear processor array connected by a network. Parallel experiments were performed on a Linux PC cluster, which follows the Beowulf style<sup>11</sup> of parallel computing clusters. Performance analysis of the computational stages of a sequential version of SPB is shown in order to examine the sequential bottlenecks inherent in the system. In addition, novel parallel algorithms for SPB are designed for use with 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 SPB have been developed: iteration decomposition and frequency decomposition. 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, then proceeds to execute a complete beamforming algorithm independently of the operation of the other nodes. Other processors concurrently execute the beamforming algorithm with different data sets collected at different times. Frequency decomposition, a form of data parallelism, is also a coarse-grained scheduling algorithm in which different frequency jobs for the same data set are assigned to different processors. Application of these methods to the SPB algorithm extends the original development of parallel algorithms for control and domain decomposition in Conventional Beamforming (CBF)<sup>12</sup>, Split-Aperture Conventional Beamforming (SA-CBF)<sup>13</sup>, adaptive sonar beamforming<sup>14</sup>, and robust broadband Minimum Variance Distortionless Response (MVDR) beamforming<sup>15</sup>.

A theoretical background of SPB is presented in Section 2 with a focus on digital signal processing. A sequential version of the SPB algorithm is given in Section 3. In Section 4, two parallel SPB algorithms are presented. In Section 5, the performance of the parallel SPB algorithms, in terms of execution time, speedup, efficiency, result latency, and memory requirements, is examined. 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 Subspace Projection Beamforming

The SPB is based on the subspace beamforming method, which uses the orthogonal property between signal and noise subspaces. The SPB decomposes the CSM into subspaces via row-zeroing QR

decomposition. Each column of the orthogonal matrix  $Q$  contains DOA information. The columns corresponding to the target locations are signal subspace, otherwise the columns are noise subspace. The reciprocal values of the steered noise subspace are the output of the SPB algorithm. Thus, peak points in the beamforming output indicate DOA of sources.

Unlike the MUSIC algorithm, the SPB algorithm has a fixed amount of computation because the algorithm estimates rank and subspace by QR decomposition. The conventional SVD algorithm for the MUSIC estimator requires three steps to compute eigenvectors and eigenvalues. The input data matrix, the CSM, is defined as

$$C = E\{x \cdot x^H\} \quad (2.1)$$

where  $x$  is a vector of Fourier components near the desired frequency,  $H$  denotes complex conjugation and transposition, and  $E\{\}$  is the expectation operation. The first step for the SVD algorithm of the CSM, a complex Hermitian matrix, is to reduce the CSM to a complex tridiagonal form by Householder matrices. Next, using a complex diagonal matrix, the complex tridiagonal matrix transforms into a real tridiagonal matrix. Finally, by an implicit QL algorithm<sup>16</sup> or QR algorithm, the tridiagonal matrix converges to a diagonal matrix. These diagonal entries of the final matrix are eigenvalues of the CSM, and the product of all transformation matrices is the eigenvector matrix. In the final stage, the QR or QL algorithm is an iterative method with a stop criterion; therefore, the computational load for the SVD is usually unknown.

Smith<sup>10</sup> designed the SPB algorithm and showed that the subspace, estimated by row-zeroing QR decomposition, is almost identical to the subspace from SVD when the signal and noise eigenvalues are well separated. A row-zeroing method for QR decomposition was used because small elements on the leading diagonal of the upper triangular matrix  $R$  produce corruption subspaces. Therefore, the  $R$  matrix becomes an unreliable indicator of further rank-deficiency. The decomposition avoids the problem by zeroing  $R$  rows, which have a small difference between previous subspaces or a small leading diagonal element. The main focus of the algorithm is to estimate rank and signal subspace by row-zeroing QR decomposition, as shown by:

$$\Sigma_S \Sigma_S^H C \approx Q_S Q_S^H C \quad (2.2)$$

where  $\Sigma_S$  is a signal column subspace from SVD and  $Q_S$  is columns of the  $Q$  matrix, which corresponds to the signal subspace.

In this realization, we assume that the number of signals is known, and the number of nodes is larger than the number of signals. Generally, the number of signals is determined directly by evaluating the eigenvalues of CSM or, in the SPB algorithm, the number can be obtained by assessing the diagonal elements of  $R$  matrix.

For simplicity, QR decomposition is realized by the Householder method in this SPB implementation. The Householder method is easily extended to row-zeroing QR decomposition. Also, the QR decomposition is equivalent to, and provides an efficient implementation of, the Gram-Schmidt Orthogonalization (GSO) process, which is used by the original SPB algorithm. In addition, the number of multiplications required to compute  $R$  in the QR decomposition using the Householder method is about half the multiplication count of the Givens QR method.<sup>17</sup>

The basic idea and implementation of the Householder method are explained by several references such as Golub<sup>18</sup>. The QR decomposition based on the Householder method entails  $N-1$  reflections, which is the matrix multiplication between the Householder matrix and the input matrix, to annihilate subdiagonal elements of the  $N \times N$  CSM matrix. Each matrix multiplication zeros out subdiagonal elements of the  $k$ th column of the CSM by multiplying the CSM with  $H_k$ , the Householder matrix. Since the last column of the CSM does not have subdiagonal elements, no reflection is applied. After  $N-1$

reflections, the CSM is the upper triangular matrix  $R$  and multiplication of all reflection matrices,  $H_k$  is the orthogonal matrix  $Q$  as shown in Eq. (2.3).

$$\begin{aligned} H_{N-1} \cdots H_2 H_1 C &= R \\ Q^H C = R &\Rightarrow C = QR \end{aligned} \quad (2.3)$$

Multiplying an elementary reflector  $H_k$  is equivalent to reflecting a matrix across the plane perpendicular to the  $w_k$  vector. For a complex number matrix, modification is necessary for the  $w_k$  vector, as shown in Eq. (2.4).

$$w_k = \frac{1}{\sqrt{2r(r + |c_{k,k}|)}} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c_{k,k} - \sigma r \\ c_{k,k+1} \\ \vdots \\ c_{k,n} \end{bmatrix} \quad \text{where } r = \sqrt{|c_{k,k}|^2 + |c_{k,k+1}|^2 + \cdots + |c_{k,n}|^2} \quad (2.4)$$

In this equation,  $w_k$  is a vector for the Householder matrix  $H_k$ , which annihilates the  $k$ th column of the CSM, and  $c_{i,j}$  represents the element at the  $i$ th row and  $j$ th column of the CSM. The parameter  $\sigma$  is  $c_{k,k} / |c_{k,k}|$  if  $c_{k,k}$  is nonzero and 1 otherwise. Based on the  $w_k$  vector, the Householder matrix is of the form:

$$H_k = I - 2w_k w_k^H \quad (2.5)$$

The CSM can now be considered as the QR decomposition rewritten in the form, Eq. (2.6)

$$C = QR = [q_1 \ q_2 \ q_3 \ \cdots \ q_n] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix}, \quad Q \Rightarrow [Q_S \ Q_N] \quad (2.6)$$

The orthogonal subspace can be divided into signal and noise subspace by comparing diagonal entries of the  $R$  matrix. Provided the matrix  $Q_N$  consists of a different subset of columns of  $Q$ , which exclude signal subspace columns, then the SPB output can be written as

$$P_{SPB}(\theta) = \frac{1}{s^H(\theta) Q_N Q_N^H s(\theta)} \quad (2.7)$$

where  $s(\theta)$  is the steering vector corresponding to look direction  $\theta$ . The noise subspace spanned by the specific columns of the  $Q$  matrix is orthogonal to the signal direction; hence, the multiplication with the steering vector corresponding to the signal direction makes the denominator terms decrease. Consequently, the output of the SPB algorithm produces prominent values near source locations.

To illustrate the above processes, Fig. 1a shows the block diagram of the SPB algorithm, and Fig. 1b is a sample output of SPB and MUSIC algorithms for comparison. As the original SPB paper<sup>10</sup> showed, the results illustrate that both beamforming algorithms generate approximately identical outputs at high Signal-to-Noise Ratio (SNR) scenarios. In the case shown, both algorithms use the same estimated CSM matrix and the SNR is 25dB with white gaussian noise.

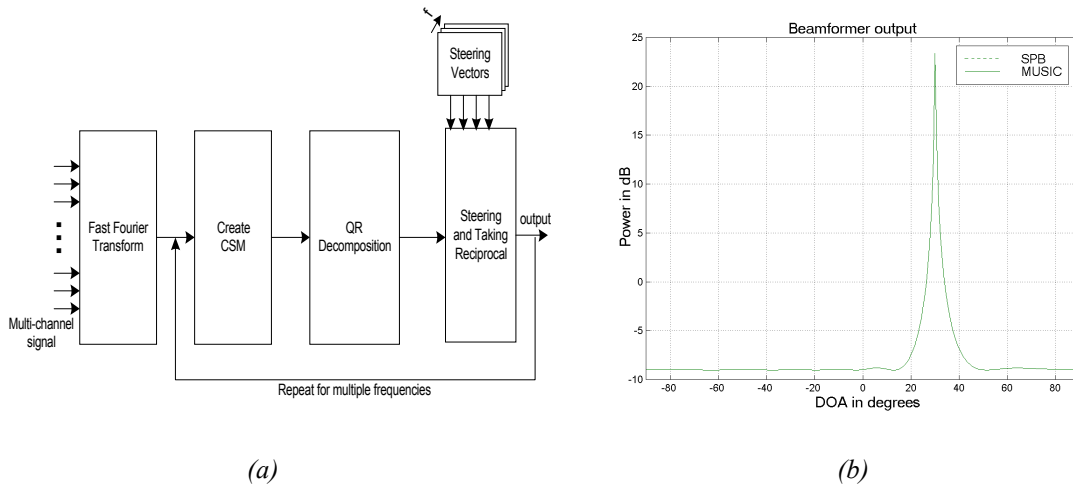


Fig. 1. Block diagram of the sequential SPB algorithm (a), and a sample output of the SPB and MUSIC for an eight-node array with a source at  $\theta = 30^\circ$  (b)

The number of emitters we can detect is restricted by the number of input nodes for subspace beamforming algorithms because of the limitation of signal subspace dimensionality. It is desirable to increase the number of input nodes and the processing frequencies in a beamforming system, which generally increases the performance from a statistical perspective, for instance CRB, statistical efficiency<sup>7</sup>, and so forth. Even if there are enough nodes to obtain a high-quality beamforming output, the number of frequency bins remains an important factor for a high-performance beamformer. For instance, beamformer post-processing often requires an augmented number of processing frequencies because of target classification (signature) and detection likelihood.

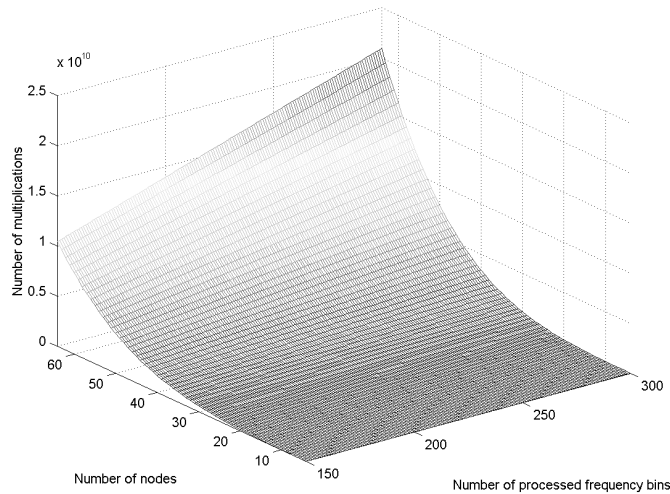


Fig. 2. The required number of multiplication operations as a function of nodes and processed frequency bins

As both of these parameters increase, the number of multiplication operations required to generate the beamforming output increases rapidly, as shown in Fig. 2. Due to the *QR decomposition* stage, the number of multiplications is more sensitive to the number of nodes. This result is expected, since the computational complexity of QR decomposition by the Householder method is  $O(N^3)$ . A more detailed analysis of sequential SPB complexity will be presented in the next section. According to Fig. 2, significant processing power is essential to generate high-performance 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 is required between response time and performance. The scalable performance provided by parallel processing will help to overcome limits imposed by a single front-end processor.

### 3. Analysis of the Sequential Subspace Projection Beamformer

The SPB algorithm consists of four separate stages: *FFT*, *CSM*, *QR decomposition*, and *steering*. Each stage has a distinctive function and computational complexity. The *FFT* stage transforms data from time domain to frequency domain by multiplying with a complex number basis. This domain transformation allows implementation of time shifting in the *steering* stage by complex number multiplication. The computation complexity of each node's FFT is  $O(N \log_2 N)$  in terms of data length; however, in terms of the number of sensor nodes, the complexity is  $O(N)$ . The creation of the CSM involves an infinite length of computation, which is not feasible in real situations. The CSM is estimated from the input streaming data and updated at each iteration with an exponential average (where  $\alpha$  is the forgetting factor) shown below:

$$\hat{C}(n+1) = \alpha \hat{C}(n) + (1-\alpha) \{x(n+1) \cdot x(n+1)^H\} \quad (3.1)$$

In Eq. (3.1), the expectation is computed by performing the exponential average, which is an estimate of the CSM, assuming that it is stationary. The computational complexity of the *CSM* stage is  $O(N^2)$  due to the vector-vector multiplication, where  $N$  is the number of nodes. The next stage is to extract orthogonal signal and noise subspaces from the CSM by the QR decomposition. The QR decomposition by the Householder method entails  $N-1$  time reflections to annihilate subdiagonal elements of the CSM. Each reflection creates a Householder matrix, which multiplies with the CSM in consecutive fashion. The *QR decomposition* stage is the most intensive stage in the SPB algorithm with  $O(N^3)$  complexity. The final stage, *steering*, computes the output for each direction with complex number vectors. Although the number of steering angles is often considerable, the computational complexity of the *steering* stage is  $O(N^2)$ . Overall, the computational complexity of the SPB algorithm is bounded by the most intensive stage, *QR decomposition*, therefore the sequential complexity is  $O(N^3)$ .

In the sequential SPB algorithm, the number of nodes and number of processed frequency bins play important roles in the computational complexity perspective. However, the number of steering angles is considerably less significant, since the number of steering angles affects only computation at the *steering* stage. The computational pattern of SPB for multiple frequency bins is identical for each frequency except frequency selection in the *FFT* stage. For the SPB output, therefore, the beamforming process is repeated for the number of frequency bins. The complexity, which depends on the number of frequency bins, is only  $O(N)$  because of this simple replication processing. Fig. 2 confirms this observation by increasing linearly in terms of the number of processed frequency bins. However, the exponential growth of multiplication represents high order complexity for the number of nodes. The *CSM* stage and *steering* stage have the same complexity, but the scalar factors for these complexities are different. Fig. 3 shows that the required multiplications for both stages vary between one another within this range. However, the numbers from the *CSM* stage and *steering* stage increase in a quadratic fashion.

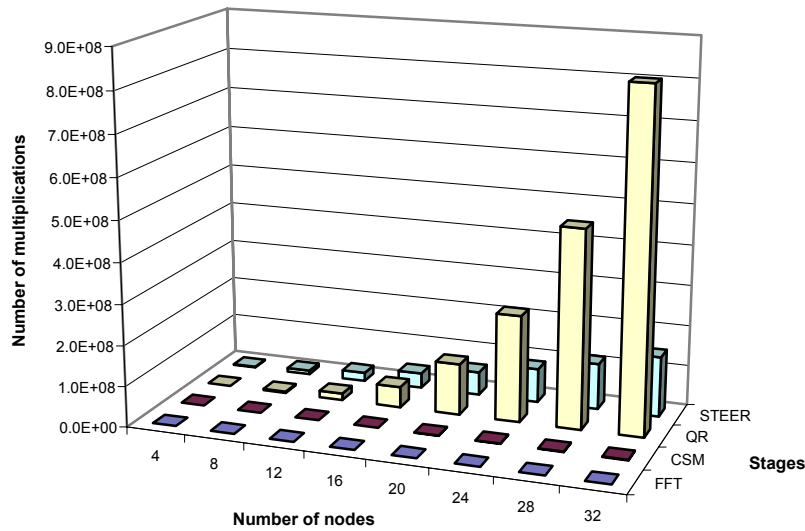


Fig. 3. Number of multiplications for each stage with 192 processed frequency bins and 2048 FFT length

To estimate attainable speedup with each of the parallel programs, execution time is the most important factor to be measured. Therefore, a minimum-calculation model<sup>13</sup>, which eliminates redundant computation by using more memory, is selected as the sequential baseline. The sequential SPB algorithm is optimized for computational speed by precalculating the steering vectors and storing in memory. The parallel algorithms are also implemented with the same minimum-calculation model, and it is assumed that each processor has sufficient memory to store the program and data.

#### 4. Parallel Algorithms for Subspace Projection Beamformers

In this paper, parallel algorithms are designed to execute on a sonar array constructed as a distributed system architecture. The system architecture consists of intelligent nodes connected by a network, as shown in Fig 4. The elimination of an expensive front-end processor achieves cost effectiveness and improved fault tolerance by distributing the workload over all the nodes. Each intelligent node has its own processing power, as well as requisite data collection and communication capability. Therefore, an augmentation in the number of nodes in the sonar array will increase processing power, as well as the problem size. Such an architecture ties the degree of parallelism (DOP) to the number of physical nodes in the target system.

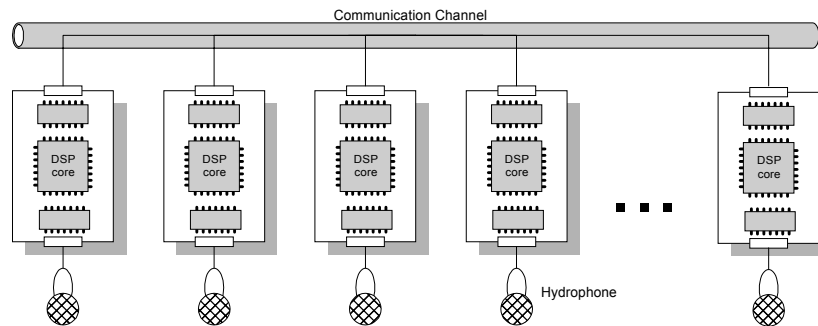


Fig. 4. Distributed architecture model

To investigate the computational performance of the parallel algorithms, it is desirable to use a flexible testbed. In these experiments, the testbed is a Linux PC cluster, which follows the Beowulf style<sup>11</sup> of parallel computing clusters. This cluster has been chosen as a parallel processing testbed on which to analyze the speedups obtained by the parallel algorithms, which subsequently can be ported to a DSP-based system. Although most DSP processors have lower clock speeds than PC microprocessors, they attain performance comparable to general-purpose processors running at much higher clock speeds because of their unique architectural features, such as fast multiply-accumulate, multiple-access memory architecture, specialized addressing modes, etc. These salient features support fast execution of the repetitive, numerically intensive tasks of signal processing algorithms.

Efficient parallelization comes from minimizing processor stalling, as well as low communication overhead between processors. With a homogeneous cluster of processors, dividing tasks evenly among the processors serves to maximize parallel performance by reducing these drawbacks. While task-based parallelism is possible with the SPB algorithm (i.e. by assigning steering to one node, QR decomposition to another node, and so forth), the workload would not be homogeneous and would result in degraded parallel performance. Fig. 3 shows this unbalanced workload among the various sequential tasks and serves as a justification for not using task-based parallelism.

The two parallel algorithms presented in this section make use of decomposition in two different domains, iteration and frequency. 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 distribution of iterations, that is, the solutions of a complete beamforming 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 uses 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, and other nodes work concurrently on other iterations. Pipelining is achieved by allowing nodes to collect new data from the sensors at all nodes and begin a new iteration before the current iteration is completed. At the beginning of each iteration, all nodes stop processing the beamforming algorithm, execute the FFT on their own newly collected samples, and send the results to the other nodes. Once these data have 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 three-node array is shown in Fig. 5.

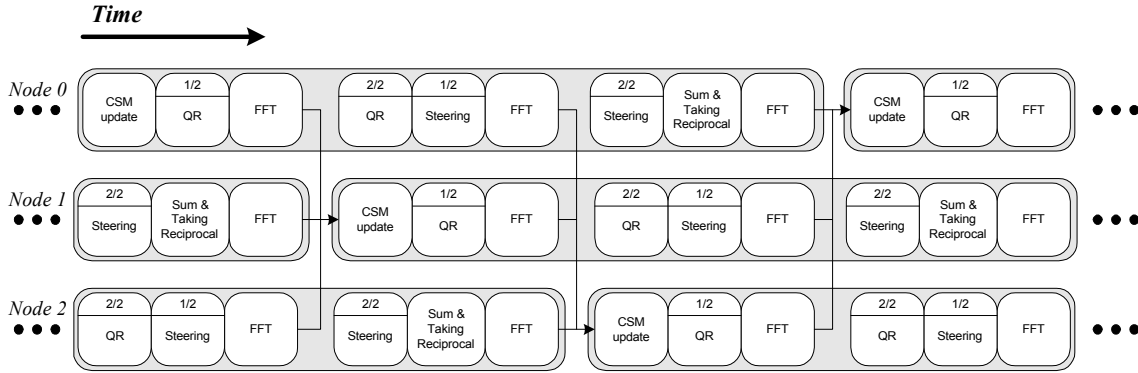


Fig. 5. Block diagram of the iteration decomposition algorithm in a three-node configuration. Solid arrows indicate interprocessor, all-to-one communication, and shaded boxes represent independent complete beamforming cycles. The index in the upper portion of some boxes shows the workload index (e.g. where “1/2” signifies the first of two workload elements).

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 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.

For even distribution of workload, iteration decomposition divides the outermost loop of the sequential algorithm across the set of processors. Each divided loop includes all computational stages, therefore the intermediate result from each computational stage is forwarded to generate a fraction of the final result in each pipelined stage. If one of the computational stages cannot produce the intermediate result, extra pipelining is required to overlap the execution.<sup>14</sup> Unlike SVD and matrix inversion, the intermediate result of QR decomposition can be obtained as the computation is progressing. In the middle of the *QR decomposition* stage, one column of the  $Q$  matrix and one column of the  $R$  matrix can be obtained by using the matrix reflection. Then, the column of the  $Q$  matrix from the *QR decomposition* stage is evaluated and steered in the following *steering* stage. Consequently, the workload distribution in the iteration decomposition is based on the columns of the  $Q$  matrix.

In each pipelined stage, one reflection of the QR decomposition is executed except the final pipelined stage. The  $(N-1)$ th reflection creates the last two columns of the  $Q$  and  $R$  matrices, and there is no necessity for another reflection. So, the DOP for the QR decomposition is not the same as the number of nodes. However, in the final pipelined stage of each iteration, the results of each steered column are added and inverted for the final SPB output.

Since the noise subspace of the  $Q$  matrix is only required for SPB computation, each column of this matrix is evaluated to determine into which of two different subspaces it belongs. If the leading diagonal values of the  $R$  matrix are less than a threshold value, then the corresponding column of the  $Q$  matrix belongs to the noise subspace. Otherwise, columns are included in the signal subspace. The columns of the noise subspace are steered at the next stage, and the signal subspace columns are discarded. The number of *steering* stages in iteration decomposition is the same as the number of columns in the noise subspace. The *CSM* stage is performed only at the first pipelined stage; however, the computational load offered by this stage is not significant compared with other stages, as shown in Fig. 3. Therefore, this slight imbalance is not expected to trigger serious computational bottlenecks in iteration decomposition.

#### 4.2. Frequency decomposition method

The second decomposable space of the SPB algorithm is the frequency domain. The SPB algorithm generates multiple frequency results by independent computation between frequency bins. The only differences in computation occur in the frequency selection and steering vectors after the *FFT* stage. The other stages are identical from frequency to frequency with different frequency samples. Therefore, the frequency decomposition algorithm distributes the processing load by decomposing the frequency bins. Each node calculates the SPB results for a certain number of desired frequency bins from the same sample set. Before the start of processing, all participating nodes must have a copy of the proper frequency data from all other nodes. After completing this all-to-all communication, each node computes beamforming for a different frequency from the same data set. A block diagram illustrating this algorithm in operation on a three-node array is shown in Fig. 6.

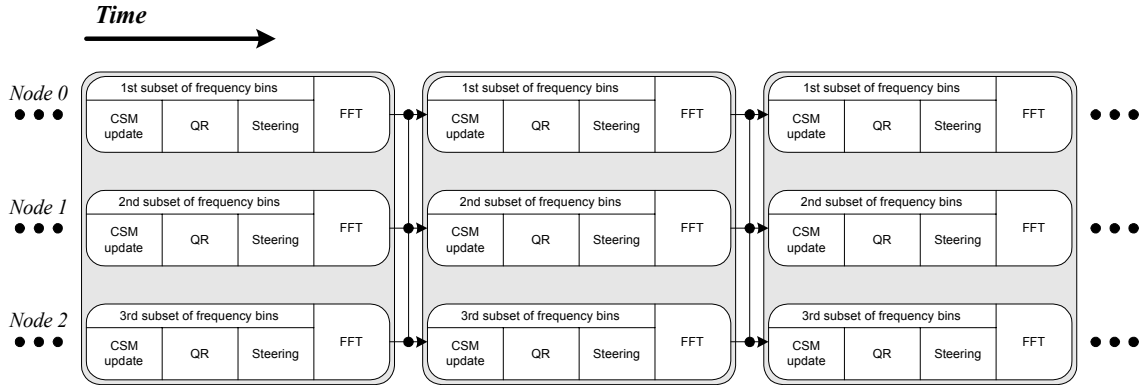


Fig. 6. Block diagram of the frequency decomposition algorithm in a three-node configuration. Solid arrows indicate interprocessor, all-to-all communication and shaded boxes represent independent complete beamforming cycles.

The communication requirement of the frequency decomposition is considerably higher than that of the iteration decomposition algorithm due to the all-to-all communication to distribute the data and all-to-one communication to collect results. The second communication is necessary to gather results because each node has partial results of wideband output following the frequency decomposition beamforming computation. The complexity and the number of instances of communication per iteration increase the total execution time of the parallel algorithm. To lower the impact of communication in the frequency decomposition method, data packing is used where nodes combine the result data of the previous iteration and the new data for the current iteration as one packet. Data packing eliminates the need to perform an all-to-one communication at the collecting stage between each pipeline stage. The beamforming results of all frequency bins are available at every node after all-to-all communication. For example, the result of the first beamforming iteration is sent by the next instance of communication in Fig. 6. The use of data packing makes the granularity of the parallel algorithm more coarse. Due to the overhead in setting up communication in a distributed-array system, sending small amounts of data results in a high overhead to payload ratio, hence it is desirable to combine multiple data packets together to amortize the overhead and reduce the effective execution time. This parallel algorithm involves a more complex communication mechanism best served with a broadcasting network. However, it does not require the additional overhead necessary to manage pipelining, as does the iteration decomposition method.

## 5. Performance Analysis of Subspace Projection Beamformers

In this section, the two parallel algorithms presented are evaluated on a distributed system testbed to analyze the parallel performance. The target testbed of this experiment is a cluster of 32 Linux PCs where each node contains a 400MHz Intel Celeron processor and 64MB of memory. The communication channel between the computers is provided by switched Fast Ethernet.

The algorithms were implemented via message-passing parallel programs written in C with the Message-Passing Interface (MPI)<sup>19</sup>. 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, a time-check function is invoked at the beginning of a stage that stores time-stamp information with clock-cycle resolution. After each stage, the function is invoked again and the earlier value is subtracted from the new return value, where the difference is the number of clock cycles required in the execution of the stage. In order to obtain reliable results, all experiments were performed for 900 iterations, and execution times were averaged.

### 5.1. Execution time of sequential algorithm

The first experiment involves the execution of the sequential SPB algorithm on a single computer, where the number of sensors is varied to study the effects of problem size. The basic beamforming parameters of this experiment and the sequential execution times are shown in Fig. 7. Execution times of the *FFT*, *CSM*, *QR decomposition*, and *steering* stages are observed to increase correspondingly with an increase in the number of sensor nodes. As expected, the execution time of the *QR decomposition* stage increases rapidly because the complexity of the stage is the most intensive in the SPB algorithm. However, the *CSM* stage shows less computation time than the *FFT* stage even though the *CSM* stage has a higher computational complexity. The reason for this phenomenon is that, for this problem size, the scalar factor of the *FFT* complexity is significantly bigger than that of the *CSM* with a fixed FFT length, 2048. Eventually, the computation time of the *CSM* stage will exceed the *FFT* time if the problem size is increased further.

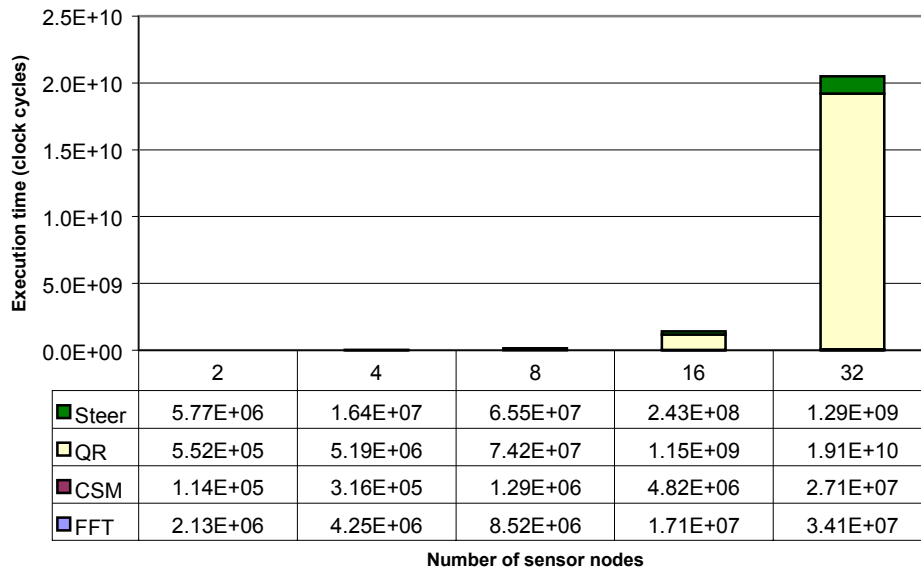


Fig. 7. Average execution time per iteration as a function of array size for the sequential SPB algorithm with 900 iterations on the testbed. (181 steering angles, 2048 FFT length and 192 processed frequency bins)

### 5.2. Execution time of parallel algorithms

Fig. 8 illustrates the average parallel execution times for five different problem and system sizes. The execution time measured from both decomposition methods is the effective execution time, which represents the amount of time between outputs from successive iterations once the pipeline has filled.

The execution time of the *FFT* stage does not change significantly as the number of nodes increases. As mentioned earlier, the parallel computing model of this implementation uses intelligent nodes at each input sensor; therefore, the number of data stream vectors is identical to the number of processors. This linear processor model decreases the degree of the complexity polynomial by one in terms of number of nodes. For example, the FFT complexity in this implementation will be  $O(1)$  from  $O(N)$ ; hence, the additional workload caused by increasing the number of nodes is evenly distributed across the processors in this case. As a result, the number of nodes does not influence the execution time of this stage. The execution times for the *CSM*, *QR*, and *steering* stages increase in the fashion of  $O(N)$ ,  $O(N^2)$  and  $O(N)$ , respectively.

Total computation time, which excludes communication time, shows that the iteration decomposition method requires more computation than frequency decomposition. To manage the pipelined execution, the iteration decomposition involves overhead, which allows the sequential workload to be distributed evenly over pipeline stages. In addition to the pipeline overhead, the slight imbalance of workload, as explained in Section 4.1, also increases the effective execution time of iteration decomposition.

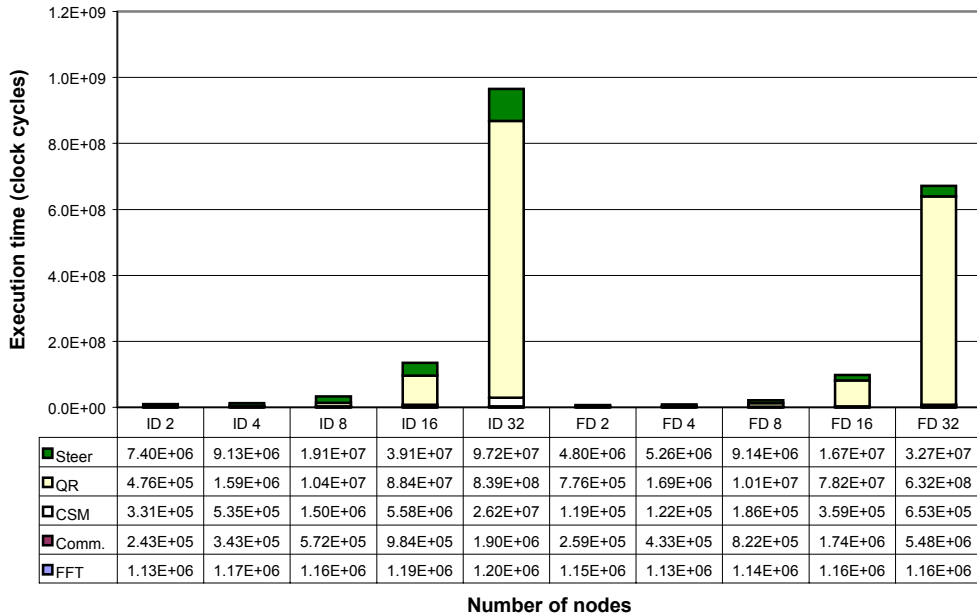


Fig. 8. Average parallel execution time as a function of array size for the parallel SPB algorithms (ID is iteration decomposition and FD is frequency decomposition) with 900 iterations on the Linux PC cluster

### 5.3. Computation vs. communication

The communication time in this parallel experiment is defined as the time spent in communication function calls of MPI. The iteration decomposition uses a simple all-to-one communication but frequency decomposition communicates an all-to-all message at the initial phase of each beamforming

iteration. All nodes require approximately the same amount of time for all-to-all communication, but in all-to-one communication, only one node requires greater communication time for receiving data from other nodes. Since every partial job is synchronized at the communication points, receiving time is considered as communication time for iteration decomposition.

Compared to frequency decomposition, the communication time of iteration decomposition has a smaller contribution to the total execution time. Due to the increased communication complexity in frequency decomposition, the time of the data communication increases rapidly with the number of nodes as evidenced in Fig. 9a. With this increase in communication comes an increase in the MPI overhead, an increase in network contention, and poorer performance, which eventually deteriorates the parallel performance as the problem and system size increase. Obviously, the relatively small amount of communication in iteration decomposition is an advantage.

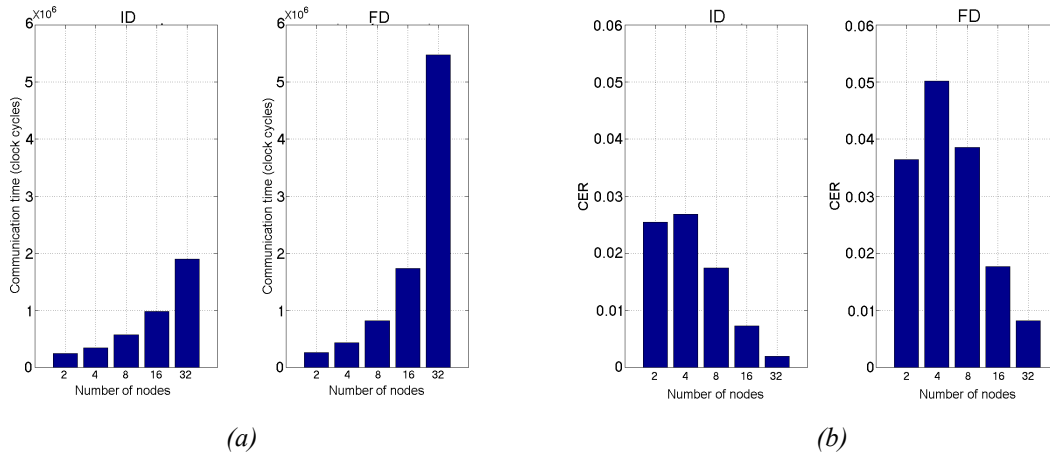


Fig. 9. Average communication time per iteration (a) and communication to execution time ratio (b)

The parallelization of the sequential algorithm inevitably requires interprocessor communication, which cannot be divided into any domain. By contrast, it is possible to distribute the computation portion of the algorithm into processors or time space. According to Amdahl's law, a small number of sequential operations can significantly limit the speedup achievable by a parallel program. Thus, the bottleneck caused by the communication limits the speedup if the communication to parallel execution time ratio is significant. To analyze the parallel algorithm from this viewpoint, Fig. 9b plots the communication to execution time ratio (CER). If the computation portion of the sequential algorithm is effectively partitioned, a smaller CER is desirable because less CER indicates the less communication time for the parallel algorithm. With a small number of nodes, the increased CER may cause performance degradation of both the parallel algorithms. Overall, the CER of both decompositions decreases as the number of nodes increases because of computational complexity; hence, we expect the parallel algorithms for SPB to be scalable in this viewpoint.

#### 5.4. Scaled speedup and parallel efficiency

Fig. 10a shows the scaled speedup of the two decomposition methods on the testbed cluster of Linux PCs. The baseline for comparison is the sequential SPB algorithm running on one PC of the same testbed. Since the algorithms incur additional workload as sensors are added, the plots show scaled speedup. Fig. 10b displays scaled efficiency, which is defined as scaled speedup divided by the number of processors used. The parallel efficiency for both decomposition techniques demonstrates an increasing trend with a different offset value. For iteration decomposition, the parallel efficiency starts

from 45% and levels off at 67%. The efficiency of iteration decomposition is bounded by the pipeline overhead and imbalance of the parallel algorithm rather than by interprocessor communication. The communication time of iteration decomposition is not considerable, as seen from the small CER values; therefore, the contribution of the communication time is negligible. However, in this case, the bottleneck caused by overhead and imbalance limits the speedup to no more than 12 for a 16-node configuration and 23 for a 32-node configuration.

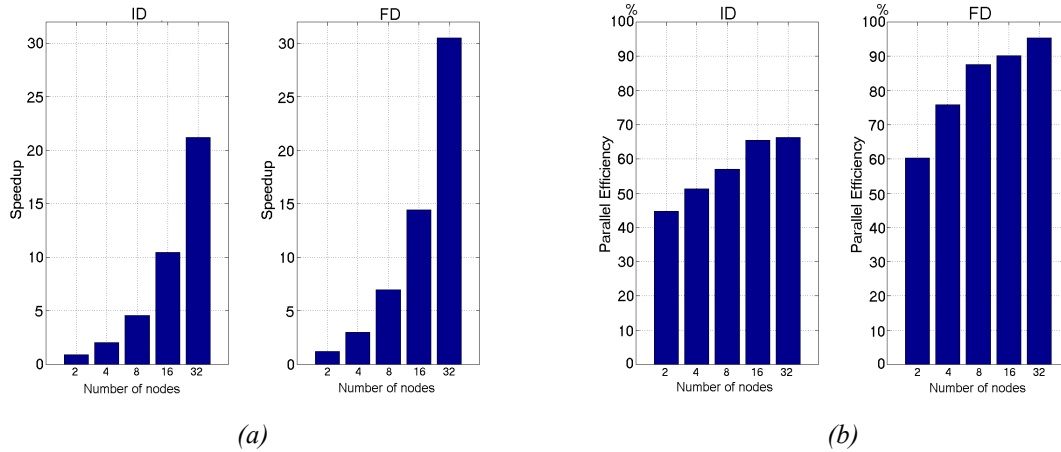


Fig. 10. Scaled speedup (a) and scaled efficiency (b) as a function of the number of processors for the iteration decomposition and frequency decomposition algorithms

For frequency decomposition, the parallel efficiency increases with problem and system size to a greater extent than did iteration decomposition. As mentioned earlier, frequency decomposition efficiently distributes sequential workload with less overhead, but the communication time of the all-to-all scheme compensates for the execution time with a small number of nodes. The advantage of efficient partitioning is not fully realized by the complicated communication up to a certain range of the problem and system size. For the two-node configuration, the parallel efficiency of frequency decomposition is the worst because of the significant communication time. As the number of nodes increases, the computational part becomes more dominant, and the parallel efficiency approaches the ideal. However, it is expected that, as the problem and system sizes continue to increase, eventually communication will become a serious performance bottleneck given the nature of its  $O(N^2)$  communication complexity.

### 5.5. Result latency

Due to the pipelined property between nodes, the results of iteration decomposition for a given beamforming cycle are obtained after  $N$  stages. As is typical of all pipelines, due to the overhead incurred, the result latency of iteration decomposition is greater than the total execution time of the sequential algorithm. For frequency decomposition, the communication is overlapped across iterations by the data packing technique; therefore, the result latency of this parallel algorithm is the effective execution time plus a small amount of overhead. In this case, after the data are transformed, the first communication takes place to distribute data to all nodes. The generated beamforming results from each node are delivered by the next communication. In addition to the effective execution time, the result latency of frequency decomposition requires one more FFT and communication time. Fig. 11 shows result latency of the beamforming algorithms. As expected, frequency decomposition experiences short latency but iteration decomposition suffers from longer latency than the sequential algorithm.

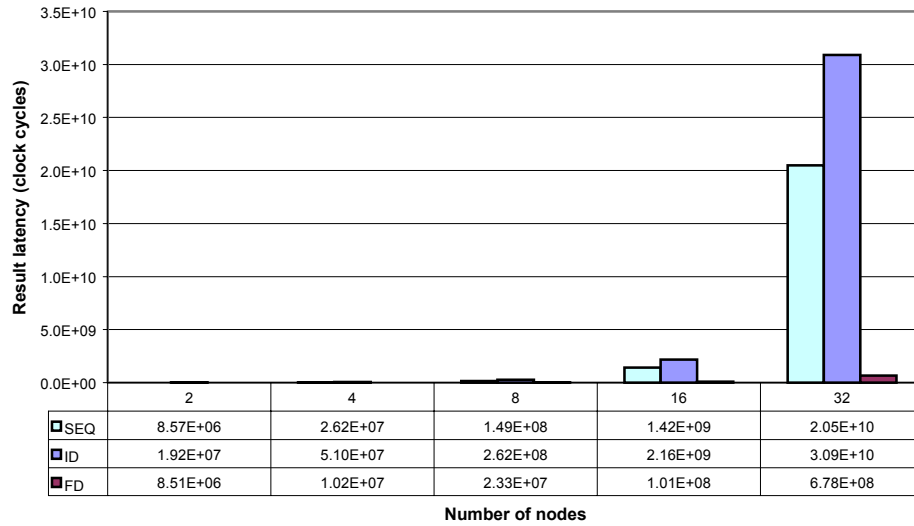


Fig. 11. Average result latency time per beamforming job as a function of array size for the sequential and parallel SPB algorithms with 900 iterations on the Linux PC cluster

### 5.6. Data memory capacity

For faster execution, most of the invariant parameters are stored in memory space. In this configuration, the majority of the memory requirements associated with the sequential and parallel SPB algorithms arises from the *steering* stage. Iteration decomposition requires the full amount of steering vectors because each processor implements a whole beamforming task for an incoming data set. By contrast, frequency 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 as the number of nodes is increased, as shown in Fig. 12. However, little change is observed for frequency decomposition.

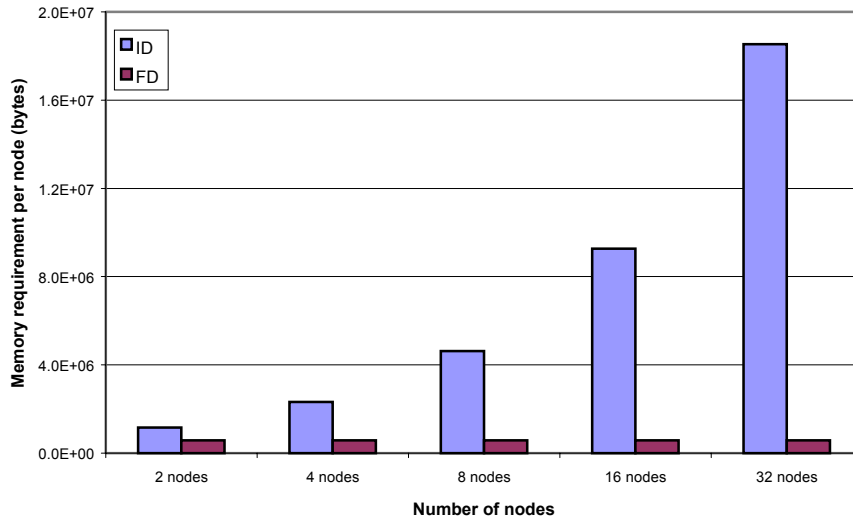


Fig. 12. Memory requirement of the *steering* stage as a function of nodes for both parallel algorithms. The memory requirements for the sequential algorithm are comparable to those of iteration decomposition.

## 6. Conclusions

In this paper, two novel parallel algorithms are developed and analyzed for the computationally intensive SPB beamforming algorithm. In the iteration decomposition, the sequential workload is divided into the number of processor stages and these stages are overlapped in execution by pipelining. Each processor in the frequency decomposition calculates the beamforming results for a subset of desired frequency bins from the same data set. For the workload distribution, the iteration decomposition uses the column and row of the sequential matrix computation and the frequency decomposition is based on the entire matrix computation for the certain set of the frequency bins. Another major difference between the two parallel algorithms is the communication pattern, which is all-to-one for the iteration decomposition and all-to-all for the frequency decomposition. The performance of these parallel algorithms is investigated as aspects of communication and computation such as execution time, scalable speedup, parallel efficiency, result latency, and memory requirements.

Overall, both the parallel algorithms achieve scalable parallel performance with increasing scaled efficiency. On the testbed, it was observed that parallel efficiency increased by almost 22% and 35% from a 2-node to a 32-node system with iteration decomposition and frequency decomposition, respectively. Due to the cubic computational complexity in the sequential algorithm, the partitioned computation in the parallel SPB algorithms occupies most of the parallel execution time as the number of nodes increases. Thus, the overhead caused by parallel implementation is concealed by these computation times and parallel efficiency of both algorithms exhibits better performance for larger system size. Of the two algorithms, frequency decomposition is the better overall choice since it shows better scalable performance with a smaller memory requirement and shorter result latency as ratio of system size. Furthermore, the parallel performance of frequency decomposition may be further improved by using a high-speed network with efficient broadcast capability. By contrast, due to the limited amount of communication, iteration decomposition may be well suited for architectures with a low-performance network, or when problem and system sizes begin to make communication the dominant factor in execution time.

The parallel beamforming techniques described in this paper present many opportunities for increased performance, reliability, and flexibility in a distributed parallel sonar array. Future work will involve parallelizing more intricate beamforming algorithms, including matched-field processing and match-field tracking. Furthermore, the fault tolerance of the sonar architecture will be studied to take advantages of the distributed nature of the parallel architecture.

### Acknowledgments

The support provided by the Office of Naval Research on Grant N00014-99-1-0278 is acknowledged and appreciated. Special thanks go to Thomas Phipps from the Applied Research Lab at the University of Texas at Austin for his insight and many useful suggestions. This work was also made possible by equipment donations from Nortel Networks.

### References

1. A. O. Steinhardt and B. D. Van Veen, "Adaptive beamforming," *International J. of Adaptive Control and Signal Processing*, **3**, 253-281 (1989).
2. L. Castedo and A. R. Figueiras-Vidal, "An adaptive beamforming technique based on cyclostationary signal properties," *IEEE Trans. on Signal Processing*, **43** (7), 1637-1650 (1995).
3. M. Zhang and M. H. Er, "Robust adaptive beamforming for broadband arrays," *Circuits, Systems, and Signal Processing*, **16** (2), 207-216 (1997).
4. J. Krolik and D. Swingler, "Multiple broad-band source location using steered covariance matrices," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, **37** (10), 1481-1494 (1989).
5. R. Schmidt, "A signal subspace approach to multiple emitter location and spectral estimation," Ph.D. dissertation, Stanford Univ., (1981).
6. H. Krim and M. Viberg, "Two decades of array signal processing research," *IEEE Signal Processing Magazine*, pp. 67-94, July, (1996).
7. P. Stoica and A. Nehorai, "MUSIC, Maximum Likelihood, and Cramer-Rao Bound," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, **37** (5), 720-741 (1989).
8. S. Hsiao and J. Delosme, "Parallel singular value decomposition of complex matrices using multidimensional CORDIC algorithms," *IEEE Trans. on Signal Processing*, **44** (3), 685-697 (1996).
9. W. Robertson and W. J. Phillips, "A system of systolic modules for the MUSIC algorithm," *IEEE Trans. on Signal Processing*, **39** (11), 2524-2534 (1991).
10. M. J. Smith and I. K. Proudler, "A one sided algorithm for subspace projection beamforming," *Proc. SPIE: "Advanced signal processing algorithms, architectures and implementations VI,"* pp. 100-111, Denver, Colorado, Aug. 4-9, (1996).
11. T. Sterling, D. Becker, D. Savarese, et al. "BEOWULF: A Parallel Workstation for Scientific Computation," *Proc. of 1995 International Conf. on Parallel Processing (ICPP)*, Vol. 1, pp. 11-14, Aug., (1995).
12. A. George, J. Markwell, and R. Fogarty, "Real-Time Sonar Beamforming on High-Performance Distributed Computers," *Parallel Computing*, **26** (10), 1231-1252 (2000).
13. A. George and K. Kim, "Parallel algorithms for split-aperture conventional beamforming," *Journal of Computational Acoustics*, **7** (4), 225-244 (1999).
14. A. George, J. Garcia, K. Kim, and P. Sinha, "Distributed Parallel Processing Techniques for Adaptive Sonar Beamforming," *Journal of Computational Acoustics*, **10** (1), 1-23 (2002).
15. P. Sinha, A. George, and K. Kim, "Parallel Algorithms for Robust Broadband MVDR Beamforming," *Journal of Computational Acoustics*, **10** (1), 69-96 (2002).
16. R. S. Martin and J. H. Wilkinson, "The implicit QL algorithm," *Numer. Math.*, **12**, 377-383 (1968).
17. C. G. Cullen, "An introduction to numerical linear algebra," PWS Publishing Company, Boston, p. 134, (1994).
18. G. H. Golub and C. F. Van Loan, "Matrix computations 3<sup>rd</sup> edition," The Johns Hopkins University Press, Baltimore and London, p. 224, (1996).
19. Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," Technical Report CS-94-230, Computer Science Dept., Univ. of Tennessee, April (1994).