Space-Time Adaptive Processing on the Mesh Synchronous Processor

Janice Onanian McMahon

This article examines the suitability of the Mesh synchronous processor (MeshSP) architecture for a class of radar signal processing algorithms known as space-time adaptive processing (STAP), which is an important but computationally demanding technique for mitigating clutter and jamming as seen by an airborne radar. The high computational requirements of STAP algorithms, combined with the need for programming flexibility, have motivated Lincoln Laboratory to investigate the application of commercially available massively parallel processors to the STAP problem. These processors must be sufficiently flexible to accommodate different STAP architectures and algorithms, and must be scalable over a wide parameter space to support the requirements of different radar systems.

The MeshSP offers high peak-signal-processing performance in a small form factor, which makes it attractive for airborne radar environments. An algorithm implementation must be reasonably efficient to take advantage of this performance. The implementation efficiency depends on four factors: the computational details of the algorithm, the chosen decomposition of the algorithm into constituent parts capable of parallel execution, the subsequent mapping of these parallel components onto the processor, and the underlying suitability of the architecture of massively parallel processors. This article describes performance models and methods of estimating MeshSP efficiency on representative STAP algorithms in order to assess the potential use of MeshSP in airborne STAP applications.

The performance of airborne surveillance radars can be severely degraded by environmental clutter, especially in overland or coastal regions, and by hostile jamming sources. Faced with a potentially formidable interference environment, radar engineers have developed advanced signal processing techniques to improve radar performance. One such technique, known as space-time adaptive processing, or STAP, is based on the idea of designing a two-dimensional (space and time) filter that maximizes the output signal-to-interference-plus-noise ratio, thereby selectively nulling clutter and jamming returns while at the same time retaining the target signal. The filter is formed by simultaneously combining the signals received on multiple elements of an antenna array and multiple pulse-repetition intervals of a coherent processing interval. The antenna elements provide the filter's spatial dimension and the pulse-repetition intervals provide the temporal dimension. This combination can also be seen as a beamforming operation, in which a beam is formed with chosen spatial and frequency directions matched to the target signal, and possessing nulls in the directions and frequencies of adaptively sensed interferers. The beam-

forming weights are computed from radar returns containing information on the spatial and temporal characteristics of the interference.

STAP offers the potential to improve airborne radar performance in several ways. It can increase detection of low-velocity targets by better suppression of mainlobe clutter, improve detection of small cross-section targets that might otherwise be obscured by sidelobe clutter, and improve detection in combined clutter and jamming environments. Moreover, STAP is inherently robust to radar system errors, and provides a means to handle nonstationary interference.

The benefits of STAP, however, come with a high computational cost. A STAP signal processor computes a set of adaptive filter weights by solving in real time a system of linear equations of size NM, where N and M are the number of spatial and temporal degrees of freedom, respectively, in the filter. The solution for each set of weights requires on the order of $(NM)^3$ operations. For the fully adaptive approach, in which a separate set of weights is applied to all the N_c antenna elements and N_d pulse-repetition intervals per coherent processing interval, values of N_c and N_d in the range of ten to two hundred are typical. As the airborne radar scans a volume of space and searches for targets throughout a range of possible target velocities, coherent processing intervals from fifty milliseconds to a full second in duration are common. Given all these multiplicative factors, a STAP processor must be able to reformulate and solve the adaptive weight computation problem an enormous number of times. The net result of such high demand on the processor can be a requirement for computational throughputs on the order of hundreds of billions of floating-point operations per second, with execution speeds of fractions of a second.

Although the fundamentals of STAP were first pioneered by L.E. Brennan and I.S. Reed in the early 1970s [1], STAP has become practical as a real-time technique only with the recent advent of high-performance digital signal processors. Even so, fully adaptive STAP is still beyond the reach of state-of-the-art processor technology. Consequently, much of the current research work on STAP has focused on the development of algorithms that decompose the fully adaptive problem into reduced-dimension adaptive problems capable of being implemented in real time on reasonably sized processors.

A radar processor operating in real time must reliably produce the correct output within a prescribed time limit, or *latency*. Consequently, sustained performance on key STAP kernel computations and communication patterns is a primary consideration in assessing processor suitability. We must also be able to install and operate a STAP processor on an aircraft, where the availability of space and other resources is limited. Thus processing performance per unit size, power, and weight are important metrics. In addition, algorithm development for STAP systems is an active area of research. To accommodate new algorithms and modes of operations, STAP processors must be programmable. Furthermore, STAP is applicable to a wide range of radars, so scalability of processor performance in terms of both machine size and problem size is an important consideration.

The substantial computational requirements of STAP, coupled with the need for future growth and scalability, have led Lincoln Laboratory to investigate the suitability of massively parallel processors (MPP) for this application. An MPP is a computer system with a large number of processing nodes (hundreds to thousands) interconnected by a dedicated communication network. MPPs are based on high-performance, low-cost commodity microprocessors; large commodity memory chips; and high-bandwidth, low-latency networks. They have supplanted vector supercomputers as the best high-performance machines commercially available.

Lincoln Laboratory has conducted a performance analysis of a representative STAP processing kernel on a commercially available programmable parallel architecture called the Mesh synchronous processor, or MeshSP. The MeshSP provides state-of-the-art performance, compact size, reduced weight, and low power consumption, which makes it a potentially suitable solution to our computational problem. It is capable of 7.7-billion floating-point operations per second (Gflops/sec) computational throughput on a board 7×13 inches in size, while consuming only a hundred watts of power. These excellent attributes have motivated us to look closely at the MeshSP to determine its potential as a real-time STAP processor.

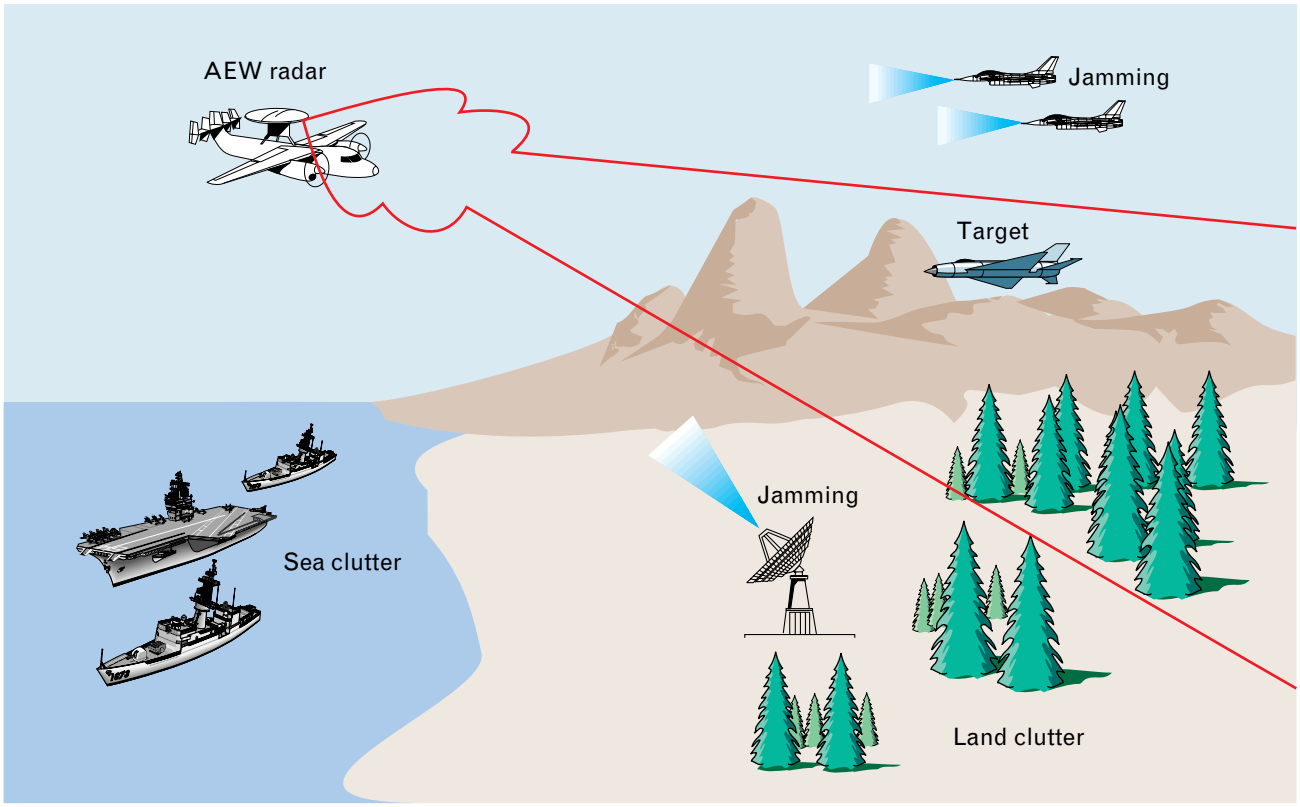


FIGURE 1. The airborne early-warning (AEW) radar environment. The airborne radar must be able to provide long-range target detection in the presence of environmental clutter and hostile jamming.

The approach we applied in the performance analysis was to develop a parameterized model for estimating latencies of computation and communication in terms of problem complexity and machine size. If we apply the model to various radar systems and processor configurations, we can determine a configuration that meets real-time requirements, and also analyze the scalability of the configuration for future changes in algorithm development [2].

In this article we review the fundamentals of STAP and describe a representative algorithm—the higher-order post-Doppler (HOPD) STAP algorithm and its key computational kernels. We follow with an overview of MPPs and a description of the MeshSP architecture. Then we present mappings onto the MeshSP of the HOPD algorithm for two representative radars, and subsequently analyze the performance and scalability of the MeshSP. We discuss the results and show that the MeshSP has strong potential as an airborne STAP processor. We conclude by indicating future directions for STAP research.

Space-Time Adaptive Processing

Figure 1 illustrates the elements of the airborne early-warning (AEW) radar environment. The AEW surveillance radar must detect increasingly smaller targets in a background of heavy interference. The two main sources of interference are environmental clutter and hostile jamming. The motion of the airborne radar platform spreads the clutter in Doppler frequency; clutter from a specific point on the ground has a Doppler frequency that depends on the angle of the clutter position relative to the heading of the platform. Noise-like jamming (or *barrage noise jamming*) from discrete sources (both in the air and on the ground) is localized in angle but spread over all Doppler frequencies. Detecting a target by enhancing radar performance in this environment of interference is the ultimate problem we must solve.

The goal of STAP is straightforward—suppress the interference and detect the target [3]. Figure 2 illustrates an example of the signal-to-noise ratio (SNR)

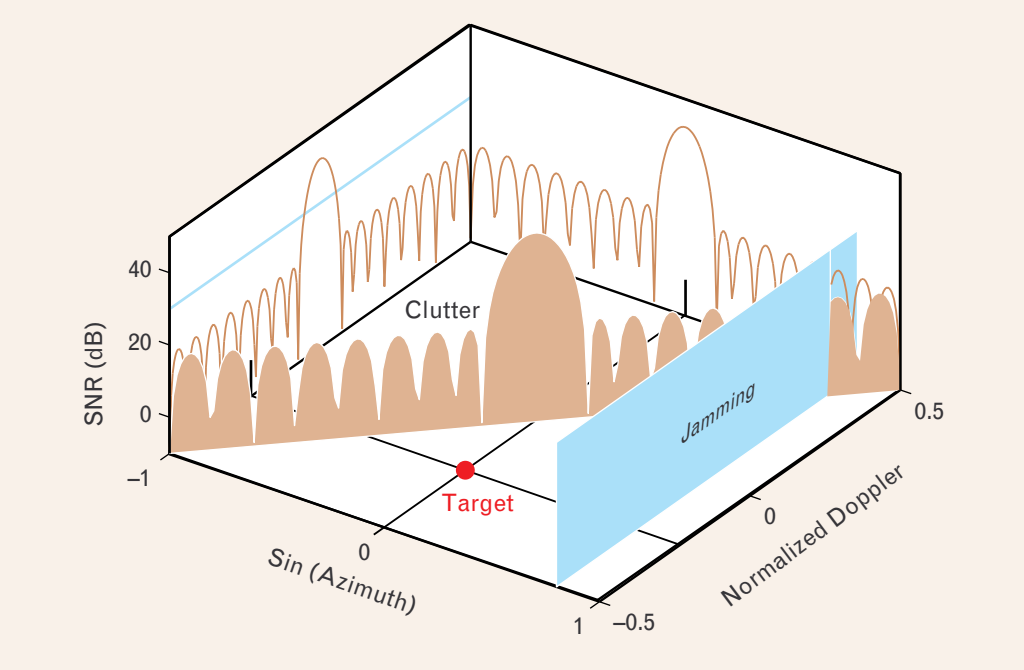


FIGURE 2. The AEW interference scenario. The problem is to detect the target by enhancing radar performance in this environment of interference. This plot shows the signal-to-noise ratio (SNR) resulting from clutter and a single jamming signal, as a function of angle and Doppler frequency. The plot also shows the view of the clutter characteristic from the perspective of azimuth for a given Doppler frequency, and the view of the clutter from the perspective of Doppler frequency for a given azimuth. These views indicate that the problem is two-dimensional in nature because filtering must be performed in each dimension.

that results from clutter and a single jamming signal, as a function of angle and Doppler frequency. Clutter from all angles lies on the clutter ridge shown in the figure, whereas the jamming signal from one angle appears in all Doppler frequencies. To suppress the interference and detect the target, the AEW surveillance radar must have high gain at the target angle and Doppler frequency, as well as deep nulls along the clutter and jamming lines.

A space-time adaptive processor combines receive beamforming in the spatial dimension and Doppler filtering in the temporal dimension to achieve the specific filter response shown in Figure 3 for a particular target angle and Doppler frequency. The STAP processor applies many of these filters, each covering a different target angle and velocity, to detect targets within the range of interest.

The process of electronically steering the radar receiver in different directions is called *phased-array beamforming*. Beamforming algorithms involve the application of weights to samples in a signal processing system. Weight application is computed as a dot product between weight vectors and sample vectors, where the vectors span the radar channels (these channels are either independent antenna receivers or elements within a single large antenna). In a conventional nonadaptive beamforming algorithm the weights are a fixed function of the look direction. In an adaptive beamforming algorithm the weights are computed from the input training data and the beam steering vectors. Figure 4 shows the STAP processing typically performed in one radar coherent processing interval, which consists of L range gates, M pulse-repetition intervals, and N antenna elements.

The optimal adaptive weight vector $\mathbf{w}$ for a given steering vector $\mathbf{s}$ is related to the interference covariance matrix $\mathbf{R}$ through the relationship $\mathbf{Rw} = \mathbf{s}$. The covariance matrix $\mathbf{R}$, which is unknown *a priori*, is defined as $\mathbf{R} = E\{\mathbf{x}\mathbf{x}^H\}$, where $\mathbf{x}$ is the space-time sample vector. We therefore perform the adaptation

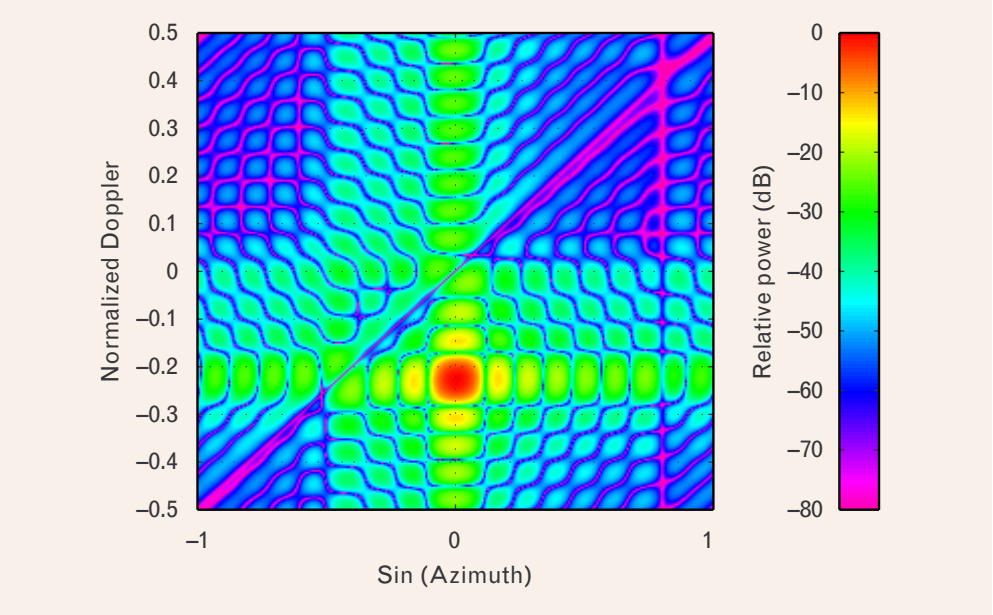


FIGURE 3. Space-time adaptive filter response. This filter combines beamforming in the spatial dimension and Doppler filtering in the temporal dimension to achieve a response for a particular target angle and Doppler frequency. These data show deep nulls along the clutter ridge and jamming line shown in Figure 2, with a high response at the target location.

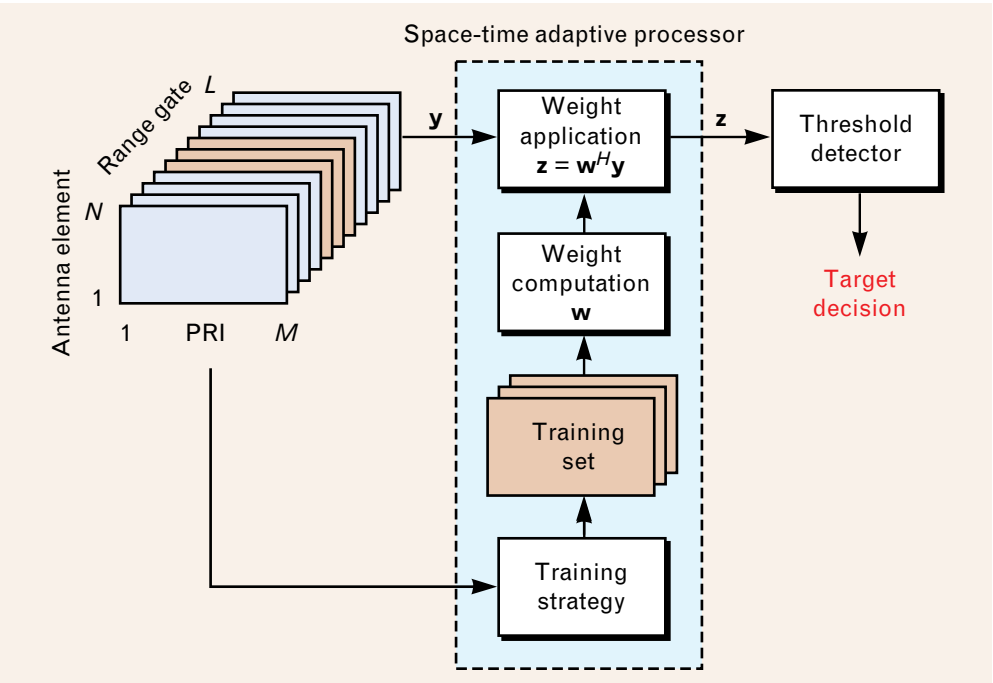


FIGURE 4. The space-time adaptive processing (STAP) typically performed in one radar coherent processing interval, which consists of L range gates, M pulse-repetition intervals (PRI), and N antenna elements. A subset of samples is chosen from the input data cube according to a training strategy. The resulting training set is used to compute the weights that are applied to the entire data cube. The final detection phase identifies the location of the targets in the data cube.

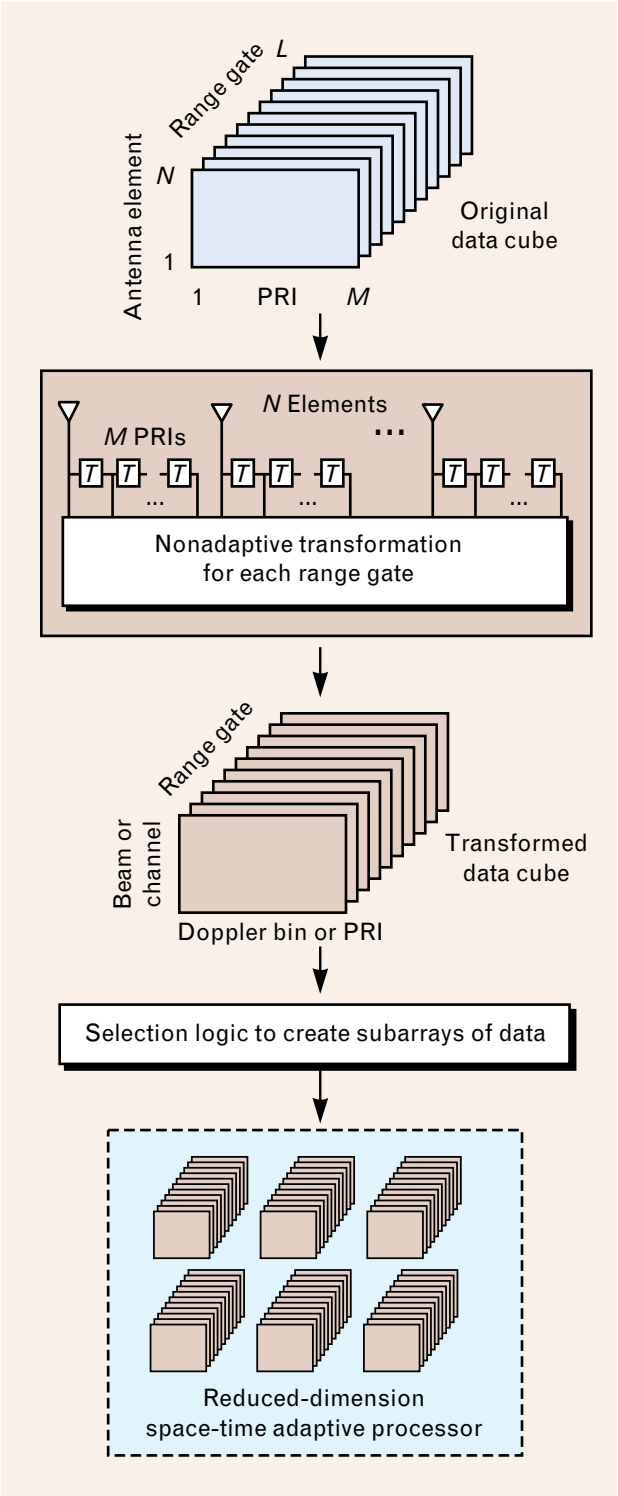


FIGURE 5. The generic partially adaptive STAP architecture. By transforming the original data cube into subarrays, we break down the prohibitively large problem of a fully adaptive STAP algorithm into a number of smaller and more computationally tractable partially adaptive problems.

by forming an estimate of $\mathbf{R}$ from a set of radar data called the training set. Specifically, the covariance matrix $\mathbf{R}$ is estimated as $\overline{\mathbf{R}} = \mathbf{X}^H\mathbf{X}$, where the training-set matrix $\mathbf{X}$ is a subset of the input data. The process of computing the adaptive weight vector $\mathbf{w}$ from the estimated covariance matrix $\overline{\mathbf{R}}$ is called *sample matrix inversion*. These methods include direct matrix inversion after explicitly forming $\overline{\mathbf{R}}$, as well as factorization approaches that compute the Cholesky decomposition of $\overline{\mathbf{R}}$ via QR decomposition of $\mathbf{X}$ [4]. For reasons of numerical stability as well as computational complexity, the factorization approach is the more viable.

Specifically, we compute $\mathbf{X} = \mathbf{QA}$, where $\mathbf{A}$ is upper triangular. Since $\mathbf{Q}$ is an orthogonal, unitary matrix, $\mathbf{Q}^H\mathbf{Q} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Therefore, $\overline{\mathbf{R}} = \mathbf{X}^H\mathbf{X}$ is equivalent to $\overline{\mathbf{R}} = \mathbf{A}^H\mathbf{Q}^H\mathbf{QA} = \mathbf{A}^H\mathbf{A}$. Since $\mathbf{A}$ is triangular, $\mathbf{w}$ is easily computed from the two backsolve operations, $\mathbf{A}^H\mathbf{u} = \mathbf{s}$ and $\mathbf{Aw} = \mathbf{u}$, where $\mathbf{s}$ is the target response corresponding to a particular hypothesized angle and Doppler frequency, and $\mathbf{u}$ is an intermediate computation vector. A different $\mathbf{s}$ is used for each member of the STAP filter bank. The beamforming operation is a matrix-vector multiplication, $\mathbf{z} = \mathbf{w}^H\mathbf{y}$, where $\mathbf{y}$ is the input data for a specific range gate and $\mathbf{z}$ is the output data in that beam.

In fully adaptive STAP algorithms, in which a separate adaptive weight is applied to all pulse-repetition intervals as well as all channels, the covariance matrix $\mathbf{R}$ has dimension $N_cN_d \times N_cN_d$, where N_c is the number of array elements and N_d is the number of pulse-repetition intervals per coherent processing interval. For typical radar systems, the product N_cN_d can vary from several hundreds to tens of thousands. For a variety of reasons, not the least of which is the large amount of computational power required for fully adaptive STAP, partially adaptive STAP is a more attractive candidate for implementation in an actual processing system.

In a partially adaptive STAP algorithm, the prohibitively large problem of a fully adaptive STAP algorithm is broken down into a number of independent, smaller, and more computationally tractable adaptive problems while achieving near-optimum performance. Figure 5 shows the generic partially adaptive STAP architecture. The first step in the par-

tially adaptive algorithm is nonadaptive filtering of the input signal data to reduce the dimensionality of the problem. The weight vector is computed by applying sample matrix inversion to the final reduced-dimension data. Once the input data are transformed, and bins and beams (or channels and pulse-repetition intervals) are selected to span the target and interference subspaces, multiple separate adaptive sample-matrix-inversion problems are solved, one for each Doppler frequency bin or pulse-repetition interval, across either antenna elements or beams, depending on the domain of the adaptation. Therefore, there is a natural inherent parallelism in partially adaptive STAP algorithms, which forms the first step in parallelizing the STAP problem.

The initial nonadaptive filtering can be either a transformation into the frequency domain (for example, by performing a fast Fourier transform, or FFT, over pulses in each channel) or a transformation into beam space (for example, by performing nonadaptive beamforming in each pulse). We can perform both spatial and temporal transformations, if desired, or we can eliminate nonadaptive filtering altogether. The nonadaptive filtering determines the domain (frequency or time, element or beam) in which adaptive weight computation occurs, and thereby serves as a convenient means of classifying STAP algorithms, as shown in Figure 6. Each quadrant in this figure shows a box representing the data domain for a single range gate after a different type of nonadaptive transform. For example, the upper right quadrant represents pulse-repetition-interval data that have been transformed into Doppler space. Thus each sample is a radar return for a specific Doppler frequency and receiver element. The lower-left quadrant represents element data that have been transformed into beam space. Thus each sample is a radar return for a specific pulse-repetition interval and look direction. Each quadrant has a variety of particular reduced-dimension algorithms based on filtering and subspace selection.

The STAP kernel used in this analysis is adaptive in the frequency domain, and therefore falls into the element-space post-Doppler quadrant of the taxonomy. A Doppler filter-bank FFT is applied to signals from each element. Low-sidelobe Doppler filtering effectively localizes competing clutter in angle, reducing the extent of clutter that must be adaptively canceled. The adaptivity occurs over all elements and a number of Doppler bins. The number of Doppler bins is a parameter of the element-space post-Doppler algorithm. When the number of Doppler bins is two or greater, the algorithm is in the class of HOPD variants of STAP. Factored post-Doppler algorithms perform spatial adaptation in a single bin and are therefore, in the strict sense, not adaptive in the temporal dimension.

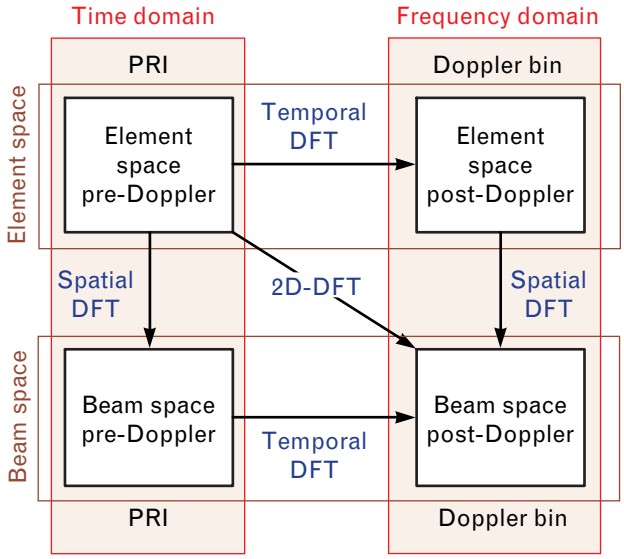


FIGURE 6. The four types of STAP algorithm transformations. Each quadrant represents a different domain in which STAP can occur. Transitions from one domain to another require the specific discrete Fourier transforms (DFT) shown in blue.

Figure 7 illustrates the STAP algorithm functional flow. The input to the kernel is a three-dimensional data cube consisting of sample data (range gates) from some number of radar channels, for some number of Doppler bins. The range gates are divided into range segments. A sample matrix, or training set, is formed for each Doppler bin and range segment by selecting a subset of the range gates from all channels for that Doppler bin and the two adjacent Doppler bins. The two-dimensional data from the three Doppler bins are concatenated to form the two-dimensional matrix used for sample matrix inversion. The algorithm to upper-triangularize the sample matrix is based on Householder transformations [5]. Figure 8

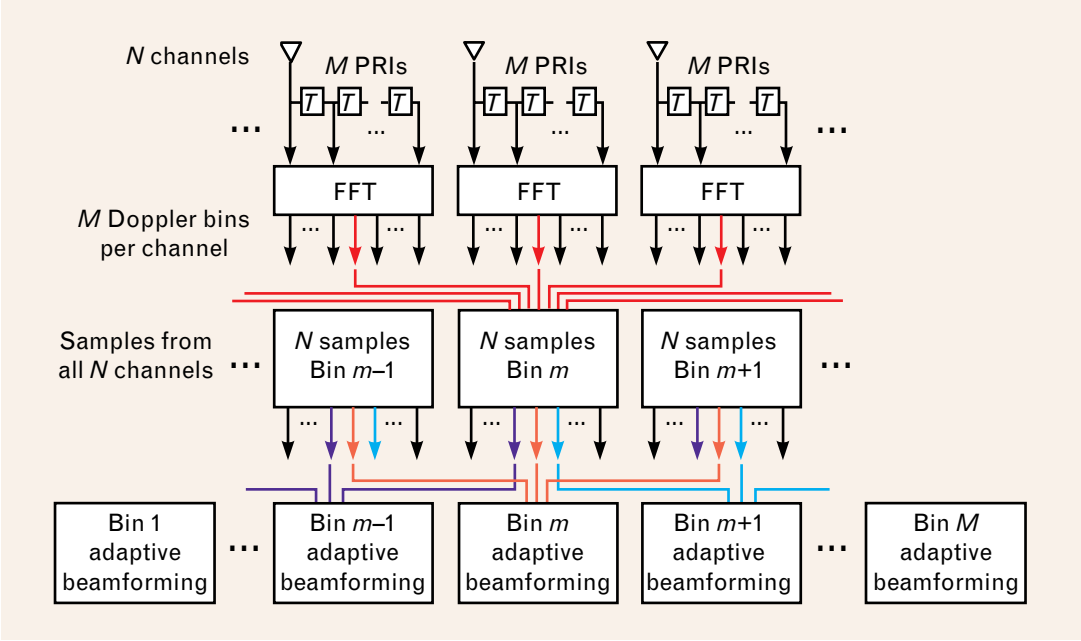


FIGURE 7. Functional flow for the higher-order post-Doppler (HOPD) algorithm. The fast Fourier transform (FFT) is performed over channels and PRIs in each range gate to set frequency-domain data for each Doppler bin. The adaptive beamforming in each Doppler bin requires post-FFT data from the two adjacent Doppler bins for all range gates.

illustrates the partitioning of the input data cube and the transformation of the input data into a two-dimensional sample matrix.

Performance of the adaptive beamforming algorithm varies with the size of each training-set sample matrix. Performance is particularly sensitive to the number of adaptive weights (i.e., the number of columns in the sample matrix). This parameter is called the *degrees of freedom*. Performance is also sensitive to the size of the training set (i.e., the number of samples selected for training). Training-set size is quantified via the *sample ratio*, which is the ratio of columns to rows in the sample matrix. Typically, to achieve decent algorithm performance, the sample ratio must be two or greater to provide adequate target detection and to null clutter and jamming. These parameters, as well as radar-specific parameters, are used to define the scalability of the algorithm mapping.

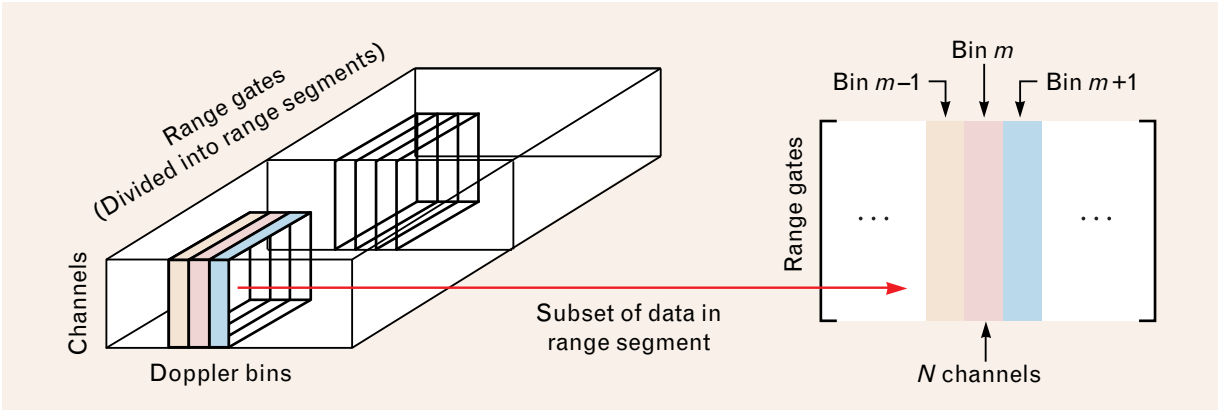


FIGURE 8. Transformation of a selected subset of data from the input data cube into a 2D training-set matrix used for sample matrix inversion. The 2D training-set matrix for each Doppler bin and range segment is formed by concatenating all the channels from adjacent Doppler bins and selecting a subset of the range gates.

MeshSP Architecture

The MeshSP is a single-instruction multiple-data (SIMD) architecture originally developed at Lincoln Laboratory and now marketed by Integrated Computing Engines, Inc., of Waltham, Massachusetts. It is comprised of an array of processors connected via a two-dimensional or three-dimensional nearest-neighbor mesh [6]. Its current realization incorporates a single monolithic processor element, the Analog Devices ADSP-21060 SHARC integrated circuit, which was developed jointly by Analog Devices and Lincoln Laboratory. The 21060 is based on a 21000 core but adds communication and memory to form a monolithic processor element that offers significant advantages in processing density per unit of size, weight, and power, if available data memory is sufficient. Each processor element permits 120 Mflops/sec peak performance and has 512 KB of internal memory (SRAM), six interprocessor communication ports, each capable of 40 MB/sec peak throughput, and two input/output ports, each capable of 5 MB/sec peak throughput. The 21060 does not have a cache; the on-chip internal memory contains the data for critical computations and is managed by the application programmer. This solution provides maximum performance in a deterministic manner. The 21060 is not an instrinsically SIMD processor in the strict sense, since it contains its own instruction-execution engine, and can be utilized in multiple-instruction multiple-data (MIMD) parallel processors as well.

The MeshSP includes an array of processor elements, a master processor, an input/output module, and a host computer. The program is executed by the master processor and the instructions are broadcast to the processor-element array, or slaves, for parallel execution. The hardware can support limited MIMD operation when code segments are loaded into the processor-element on-chip internal memory. Figure 9 illustrates the MeshSP architecture.

The MeshSP processing array has two-dimensional toroidal connectivity via its interprocessor communi-

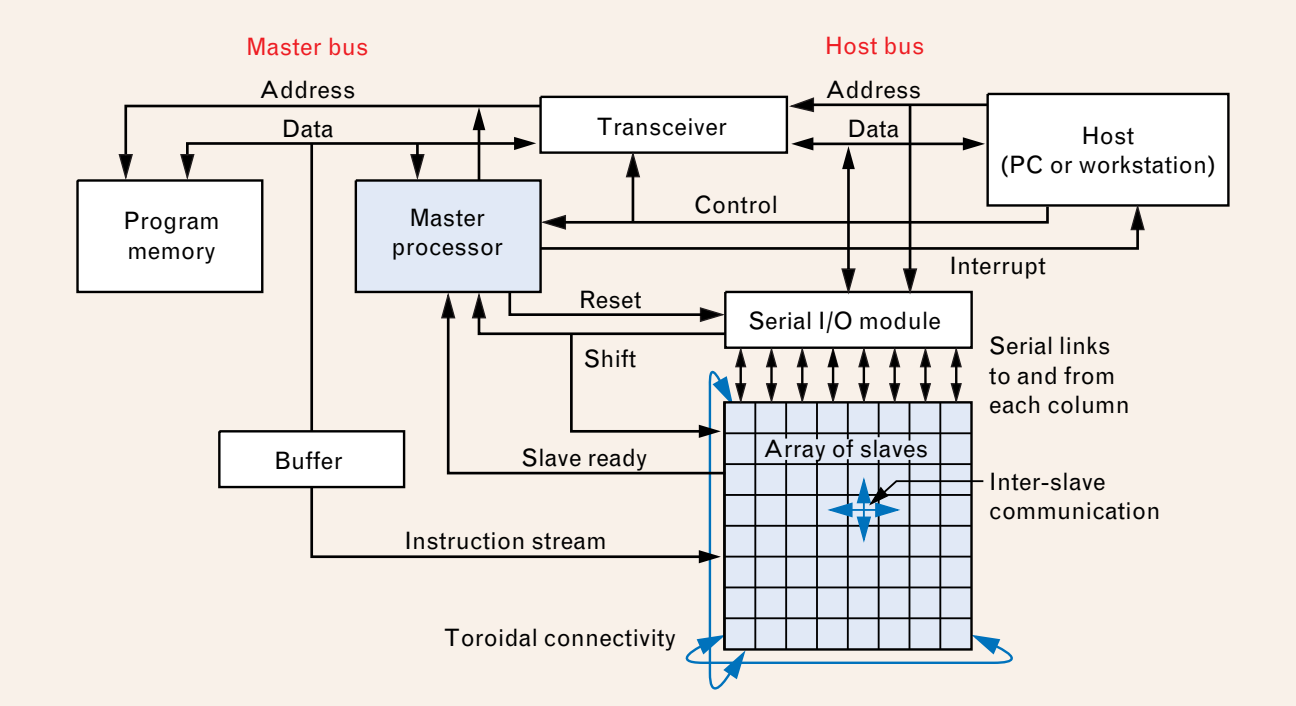


FIGURE 9. MeshSP architecture. Most of the processing power is in the array of processor elements, or slaves, which is a two-dimensional mesh of SHARC processors with four-way nearest-neighbor toroidal communication links. Each processor executes the same program, which is downloaded by the master SHARC processor from program memory. The host processor is a personal computer or workstation that controls the MeshSP and provides external communication for operators. Data input/output (I/O) for the array of slaves is achieved by the serial I/O module.

cation links. Each processor has six input ports and six output ports. In the current configuration, four input ports and four output ports are devoted to four-way nearest-neighbor interconnections. The extra two ports are used for fault tolerance by connecting them to the next nearest neighbor to the right and left. Interprocessor communication in the MeshSP is asynchronous to computation and can occur in parallel as long as no data conflicts occur.

Latency of a data transfer between processors is proportional to the sum of the vertical and horizontal distances between them in processor coordinates. In mapping an algorithm, or equivalently, a dataset, onto the processor array, data locality is of primary importance. Locality in the context of massively parallel processing means that data used in a computation are mapped to the same or physically close processors in order to reduce the cost of interprocessor communication.

Figure 10 shows a MeshSP processing board, which is 7 × 13 inches in size and dissipates approximately one hundred watts of power. The board contains sixty-four SHARC chips (thirty-two per side) and is capable of 7.7-Gflops/sec peak performance. The MeshSP architecture is scalable and can therefore reside on a multiple number of processing boards. The high processing density per unit of size, power, and weight of the MeshSP architecture is highly desirable in an airborne platform, provided acceptable levels of efficiency can be sustained. This high density is more readily achievable in SIMD processors than in MIMD processors because only one global program memory is needed, from which instructions are broadcast to each processor.

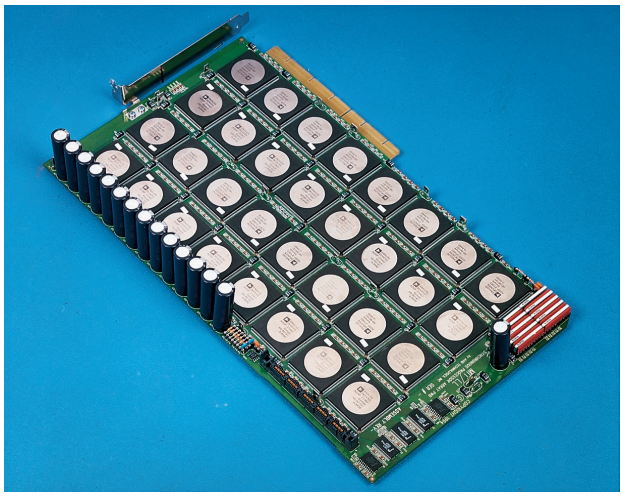


FIGURE 10. MeshSP processing board. Each processor element is an Analog Devices SHARC integrated circuit developed jointly by Analog Devices and Lincoln Laboratory. Each processing board is two sided, and contains thirty-two SHARC processor elements per side; this board is capable of 7.7-Gflops/sec peak performance while dissipating only one hundred watts of power.

Algorithm Mapping

The first step in algorithm design is to map the problem topologically onto the processor architecture. The mapping problem involves assigning data and computations in the algorithm to one or more specific processors in the processor array. Computations assigned to multiple nodes in a parallel processor benefit from increased computational bandwidth. However, a price is often paid in the lower efficiencies that result from necessary communication between nodes. The exact trade-off between computational power and parallel efficiency depends on the balance of computation and communication that results from a particular mapping on a particular architecture. The end-to-end latency, or execution time, of the algorithm must meet real-time requirements of the radar; this fact establishes a minimum number of nodes on which the algorithm must be mapped. To maximize efficiency, the assignment of data to processors must take into account the advantages of data locality so that overall communication is minimized. In addition, assignment of computations to processors must insure that there is adequate load balancing; i.e., the computational load is evenly distributed between individual processors so that overall processor utilization is maximized.

In addition to algorithm execution time, three metrics are used in benchmarking parallel-processor performance in this study. These metrics—execution speedup, parallel efficiency, and ratio of communication-to-computation latency—each quantify the performance of a parallel algorithm. The metrics are used to evaluate the suitability of the MeshSP architecture for the HOPD algorithm, and are described in the section entitled "Analytical Results."

Because each Doppler bin and range segment has

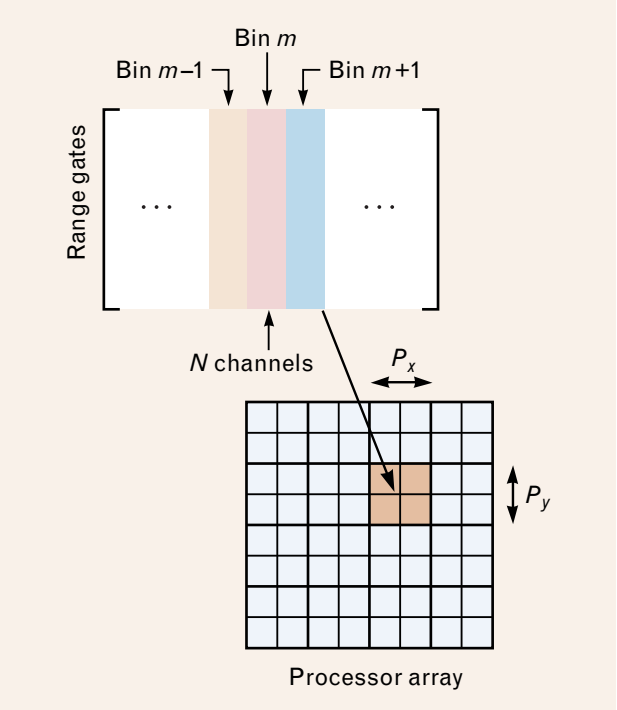


FIGURE 11. The global mapping approach. The two-dimensional training-set matrix for each Doppler bin m is mapped to a region of multiple processors in the full two-dimensional processor array. By increasing the number of processors allocated to each Doppler bin, we can improve overall execution time.

its own sample-matrix-inversion subproblem, the global mapping approach for the HOPD algorithm is to map each sample-matrix-inversion subproblem to a subarray of the full MeshSP processor-element array, where the size of the subarray is P_x by P_y processors. Figure 11 illustrates the global mapping approach. Because each subproblem after the initial data sharing is independent, scalability and performance of the HOPD algorithm as a whole are equivalent to those of an individual subproblem. For the remainder of this article we examine only subproblem performance; the number of subproblems affects only the total number of processors required.

Two mappings of training-set matrices to subarrays of the full MeshSP processor-element array were simulated: *block* and *cyclic* [7]. Each training set is an $M \times N$ matrix; therefore, each processor has $R = M/P_y$ rows and $C = N/P_x$ columns. In the block mapping, adjacent processors in the two-dimensional mesh contain adjacent blocks of multiple data elements in the two-dimensional matrix. In the cyclic mapping, each processor is assigned every P_x-th element on the x dimension and every P_y-th element on the y dimension. Adjacent processors in the two-dimensional mesh contain adjacent single data elements in the two-dimensional matrix, with wraparound at the processor boundaries. Figure 12 illustrates the differences between block mapping and cyclic mapping. In general, the cyclic mapping yields better load balancing between processors because the data are spread over more processors. For the same reason, the block mapping generally yields better (i.e., lower) communication-to-computation ratios.

Distributed Algorithm Operation

In the section on space-time adaptive processing we stated that the computation of the adaptive weight vector $\mathbf{w}$ from the estimated covariance matrix $\overline{\mathbf{R}}$ could be done most effectively through QR decomposition of the training-set matrix. In the HOPD algorithm, the method for computing $\mathbf{A} = \mathbf{QR}$ is based on the technique of Householder reflections [4]. A two-by-two orthogonal matrix $\mathbf{Q}$ is a reflection if it has the form

$$\mathbf{Q} = \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix}.$$

If $\mathbf{y} = \mathbf{Q}^T\mathbf{x} = \mathbf{Q}\mathbf{x}$, then $\mathbf{y}$ is obtained by reflecting the vector $\mathbf{x}$ across the line defined by

$$S = \text{span}\left\{\begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2) \end{bmatrix}\right\}.$$

Reflections can be used to introduce zeros in a vector by properly choosing the reflection plane. A Householder reflection is an n-by-n matrix $\mathbf{P}$ of the form

$$\mathbf{P} = \mathbf{I} - 2\mathbf{v}\mathbf{v}^T/(\mathbf{v}^T\mathbf{v}),$$

where $\mathbf{v} \in \Re^n$ is called the Householder vector. When a vector $\mathbf{x}$ is multiplied by $\mathbf{P}$, it is reflected in the hyperplane $\text{span}\{\mathbf{v}\}^{\perp}$. Householder reflections are symmetric and orthogonal, and can be used to set selected components of a vector to zero. A succession of Householder reflections applied to the columns of matrix $\mathbf{A}$ can introduce zeros below the main diago-

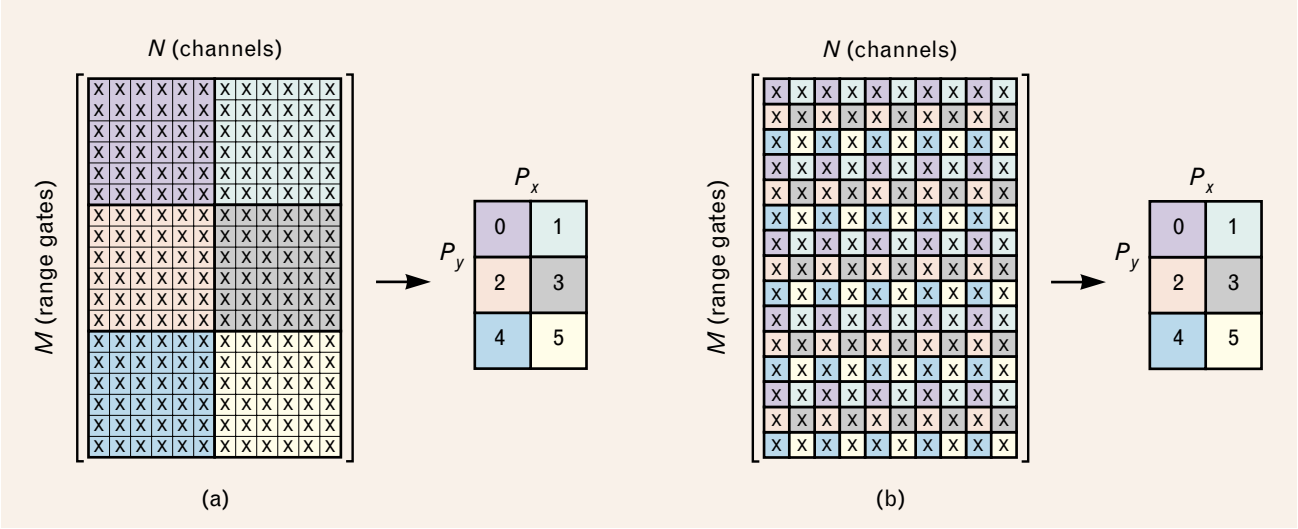


FIGURE 12. Block mapping and cyclic mapping. Each $M \times N$ matrix of training-set data is mapped onto a $P_x \times P_y$ array of processors such that each processor is assigned $R = M/P_y$ rows and $C = N/P_x$ columns of the matrix. (a) In the block mapping, adjacent processors in the two-dimensional mesh contain adjacent blocks of multiple data elements in the two-dimensional matrix. (b) In the cyclic mapping, each processor is assigned every P_x-th element on the x dimension and every P_y-th element on the y dimension. Adjacent processors in the two-dimensional mesh contain adjacent single data elements in the two-dimensional matrix, with wraparound at the processor boundaries. In general, the block mapping yields better communication-to-computation ratios, while the cyclic mapping yields better load balancing between processors.

nal, yielding an upper-triangular matrix **R**. A Householder update of a matrix **A** involves a matrix-vector multiplication followed by an outer-product update, and therefore never entails the explicit formulation of the Householder matrix. The algorithm for the kth Householder update of the sample matrix is

$$\mathbf{Q}_k = \mathbf{I} - \frac{2\mathbf{v}\mathbf{v}^H}{\mathbf{v}^H\mathbf{v}} \Rightarrow$$

$$\mathbf{Q}_k\mathbf{A} = \left(\mathbf{I} - \frac{2\mathbf{v}\mathbf{v}^H}{\mathbf{v}^H\mathbf{v}}\right)\mathbf{A} = \mathbf{A} + \mathbf{v}\mathbf{w}^H ,$$

where, if $\mathbf{a}_k$ is a column vector of data below the diagonal of matrix **A**, then

$$\mathbf{v} = \mathbf{a}_k + \mathrm{sgn}[\mathbf{a}_k(1)]\,|\mathbf{a}_k|\,\mathbf{e}_1$$

$$\mathbf{w} = \beta\mathbf{A}_k^H\mathbf{v}$$

$$\beta = -\frac{2}{\mathbf{v}^H\mathbf{v}}.$$

In this algorithm the matrix $\mathbf{A}_k$ is the submatrix of **A** for the kth iteration. Table 1 lists the steps in the distributed Householder decomposition of the matrix **A**. Table 2 outlines the steps in the distributed backsolve and weight-application operation.

As the Householder algorithm iterates over columns in the input matrix, the active part (i.e., the elements of the matrix that are involved in the current update) consists of the elements below and to the right of the main diagonal. Since only part of the matrix is active in each iteration, load balancing is an issue. Figure 13 illustrates the different load-balancing characteristics of the Householder algorithm for the two different mappings. In the block mapping some processors become inactive earlier in the sequence of iterations, whereas in the cyclic mapping all processors remain active until the final iteration. Because of this characteristic of the mappings, cyclic mapping has better load balancing.

Performance Model

Table 3 shows the equations for estimating the computation and communication costs for each step in the HOPD algorithm. A reduction operation over P

processors in which K numbers are added over the P processors involves an initial step to add the elements within each processor, then $\log_2 P$ stages to add partial sums between processors. The ith stage involves an addition and a communication of distance $2i-1$. On the MeshSP, the result is repeated in each of the P processors.

The number of steps to broadcast K numbers over P processors is proportional to KP. If pipelining can be used, then the number of steps is proportional to $K+P$. On the MeshSP there is a three-cycle overhead penalty for pipelined broadcasts; therefore, the number of steps is proportional to $K+3P$. The values R_j and C_j are the number of rows and columns, respectively, per processor during iteration j. The values P_j^x and P_j^y are the respective number of processors active during iteration j in the x and y dimensions. These four values vary with different mappings for the matrix decomposition. Table 4 lists the values for the two mappings in this study, and Table 5 lists the communication costs for the backsolve operations, where $P_y' = N/R$ is the number of processors in the y dimension active in the backsolve operations, and $R' = N/P_y$ is the number of backsolve rows per processor.

Analytical Results

Two different sets of parameter values based on two different radar systems were used in this study. Table 6 lists the parameters and their values for each case.

The resource requirements for memory and input/output are defined for a given application by using a data cube of size $N_d \times N_r \times N_{dof}$, where N_d is the number of Doppler bins, N_r is the number of range gates, and N_{dof} is the number of degrees of freedom, to allow for sharing of adjacent Doppler bins. The number of processors required for the entire processor array is $N_d \times N_s \times P_x \times P_y$, where N_s is the number of range segments, since each Doppler bin and range segment has its own training set. The number of data elements per processor element is the data cube size divided by the total number of processors; the memory required per processor element is twice that

Table 1. Distributed Householder Decomposition

Given a complex matrix $\mathbf{A}$*, apply a sequence of Householder transformation matrices* $\mathbf{Q}_k$ *to upper-triangularize* $\mathbf{A}$*. Each Householder matrix* $\mathbf{Q}_k$ *zeros a column of* $\mathbf{A}$ *below the diagonal.*

For each column $\mathbf{a}_k$ of $\mathbf{A}$ ($k = 1$ to N), compute the update of $\mathbf{A}$ that represents $\mathbf{Q}_k\mathbf{A}$:

- Square the elements in that column, then perform a reduction (addition) over the appropriate column of processors, and take the square root to compute $|\mathbf{a}_k|$.
- Update the appropriate value in the appropriate processor to compute $\mathbf{v}(1) = \mathbf{a}_k(1) + \text{sgn}[\mathbf{a}(1)]\ |\mathbf{a}|$.
- Multiply the elements in that column by their complex conjugates, then perform a reduction (addition), division, and multiplication to compute $\beta = -2/(\mathbf{v}^H\mathbf{v})$.
- Broadcast β from the appropriate processor to the right in the processor array.
- Broadcast $\mathbf{v}$ from all processors in the appropriate column to the right in the processor array.
- In all processors to the right and down from the selected processor, compute $\mathbf{w} = \beta\mathbf{a}_k\mathbf{v}$, then perform a reduction over those columns of processors to compute $\mathbf{w} = \beta\mathbf{a}_k^H\mathbf{v}$.
- Update by setting $\mathbf{a}_k = \mathbf{a}_k + \mathbf{v}\,\mathbf{w}^*$ for all elements of $\mathbf{A}$.

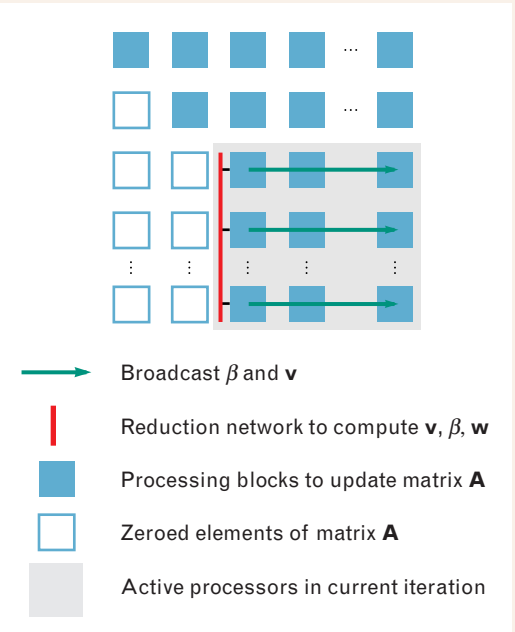


Table 2. Distributed Backsolve and Weight Application

Distributed backsolve.

First backsolve, for ($j = 1$ to N), to solve $\mathbf{A}^H\mathbf{u} = \mathbf{s}$ for vector $\mathbf{u}$:

- Divide to get $\mathbf{u}(j)$; broadcast $\mathbf{u}(j)$ right along the processor row.
- Multiply $\mathbf{u}(j)$ by $\mathbf{a}(j,j)$ and subtract from $\mathbf{v}(j)$ for all values in all processors in rest of current row.
- Send updated $\mathbf{v}(j)$ down to next row.

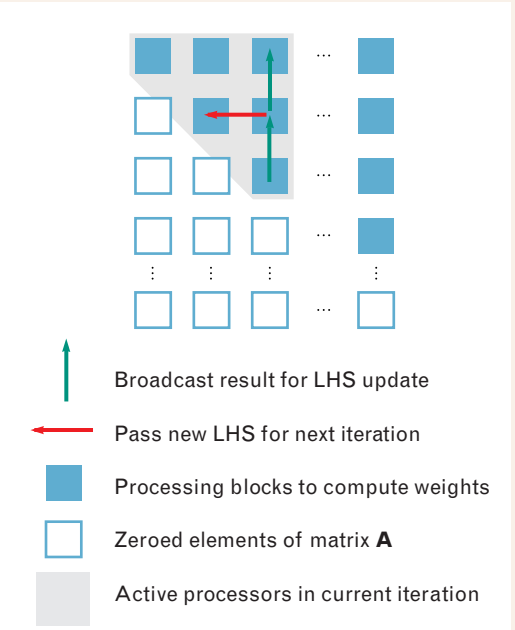
Second backsolve, for ($j = N$ to 1), to solve $\mathbf{Aw} = \mathbf{u}$ for the weight vector $\mathbf{w}$:

- Divide to get $\mathbf{w}(j)$; broadcast $\mathbf{w}(j)$ up along the processor column.
- Multiply $\mathbf{w}(j)$ by $\mathbf{a}(j,j)$ and subtract from $\mathbf{u}(j)$ for all values in all processors in rest of current column.
- Send updated $\mathbf{u}(j)$ left to next column.

Distributed weight application.

For each range gate per processor ($I = 1$ to N_r/P_y):

- Multiply and add all elements within each processor.
- Perform reduction (addition) over rows of processors.



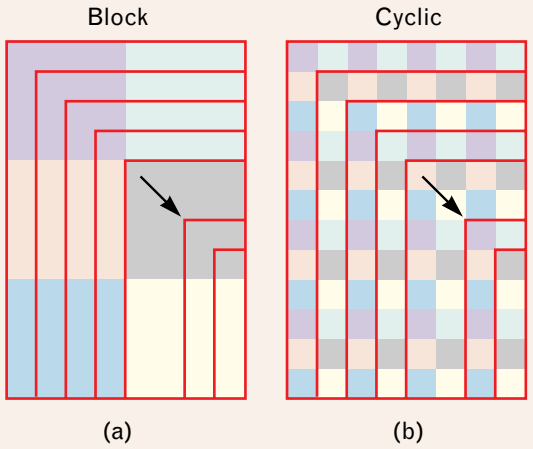


FIGURE 13. Householder-algorithm load-balancing characteristics for block mapping and cyclic mapping. (a) Block mapping produces better communication-to-computation ratios, while (b) cyclic mapping produces better load balancing between processors because the data are spread over more processors.

value to reflect double buffering of the data cube (scaled by eight for eight-byte complex data). Input/output throughput required per processor element is the number of data elements per processor element (times eight) divided by the length of a coherent processing interval.

Table 7 lists the performance metrics that were derived in this study. The peak values for MeshSP computational throughput and communication bandwidth are inserted in the equations with an estimated efficiency of 25% in both processor-element performance and processor-element intercommunication. The single processor-element efficiency has a significant effect on overall processor efficiency and therefore must be measured on actual MeshSP hardware.

Architecture Performance Analysis

The performance model described in the previous section was exercised by using two different radar

Table 3. Total Computation and Communication Costs*

Function	*Computation*	*Communication*
Householder decomposition:		
Compute **v**	$4R_j + \log_2 P_j^y + 1$	$4(P_j^y - 1)$
Compute β	$4R_j + \log_2 P_j^y + 1$	$4(P_j^y - 1)$
Compute **w**	$C_j(8R_j + 2\log_2 P_j^y + 2)$	$8(P_j^y - 1)$
Compute **a**	$8R_j C_j$	$8R_j + 12P_j^y$
Backsolve	$N_v(22 + 8C_j + 8R_j')$	$N_v Z$
Weight application	$N_v N_r(8C + 2\log_2 P_x)/(P_y N_s)$	$N_v \times N_r \times 8(P_x - 1)/P_y N_s$

* R_j is number of rows per processor, C_j is number of columns per processor, P_j^x is number of x processors active, and P_j^y is number of y processors active during iteration j of the Householder algorithm. See Tables 4 through 6 for the mapping-specific parameters (R_j', Z) and radar parameters (N_v, N_r, N_s).

Table 4. Mapping-Specific Parameters

*Value**	*Block*	*Cyclic*
R_j	if $j < M - R$ then R else $M - j$	$R - \text{floor}(j/P_y)$
C_j	if $j < N - C$ then C else $N - j$	$C - \text{floor}(j/P_x)$
P_j^x	$P_x - \text{ceiling}((j+1)/C)$	if $j < N - P_x$ then $P_x - 1$ else $N - j - 1$
P_j^y	P_y	P_y
R_j'	if $j < N - R$ then R else $N - j$	$R - \text{floor}(j/P_y)$
$P_j^{y'}$	$P_y - \text{ceiling}((j+1)/R)$	if $j < N - P_y$ then $P_y - 1$ else $N - j - 1$

* R_j' is the number of rows per processor during iteration j of the backsolve algorithm; $P_j^{y'}$ is the number of y processors active during iteration j of the backsolve algorithm.

Table 5. Backsolve Costs for Block Mapping and Cyclic Mapping*

Block
$$Z = 8\left(\sum_j P_j^x + \sum_j P_j^{y'} + (P_y' - 1) + (P_x - 1)\right)$$

Cyclic
$$Z = 8\left(\sum_j P_j^x + \sum_j P_j^y + (P_y - 1)(2R' - 1) + (P_x - 1)(2C - 1)\right)$$

* $P_y' = N/R$ is the number of y processors active in the backsolve; $R' = N/P_y$ is the number of rows per processor.

Table 6. Radar Application Definition

Parameter	*Symbol*	*Radar 1*	*Radar 2*
Number of channels	N_c	48	18
Number of Doppler bins	N_d	128	16
Number of range gates	N_r	1250	16,384
Number of segments	N_s	2	4
Degrees of freedom	N_{dof}	144	54
Training sample ratio (M/N)	k	2	3
Number of beams	N_v	2	3
Length of coherent processing interval (sec)	T_{cpi}	0.5	0.06

Table 7. Performance Metrics

Definition	*Value*
Processing latency	$T_p = N_{\text{ops}}/(120 \times 0.25)$
Communication latency	$T_c = N_{\text{bytes}}/(40 \times 0.25)$
Total latency	$T = T_p + T_c$
Communication-to-computation ratio	$R = T_c/T_p$
Speedup	$S = T_1/T_N$
Efficiency	$E = T_1/(N \times T_N)$

applications (Radar 1 and Radar 2) and two different mappings (block and cyclic). Radar 1 and Radar 2 parameter sets are based on actual radar systems with low and high instantaneous bandwidths, respectively. For each case, we derived performance measures for a variety of processor configurations. By plotting processor configuration on one axis and performance metric on the other axis, we can show scalability of the algorithm mapping to the MeshSP.

Radar 1 Performance Results

Figure 14 shows the variation in estimated total latency for each mapping of the HOPD algorithm with the Radar 1 radar parameters. These graphs show that real-time requirements can be met with four processors per sample-matrix-inversion problem with each mapping. For 128 Doppler bins and two range segments, a total of 1024 processors for all Doppler bins and range segments is required. The cyclic mapping gives greater flexibility in mapping choice to meet real-time requirements for latency, since two points in processor space that do not meet real-time requirements for block mapping do meet real-time requirements for cyclic mapping. This fact occurs because of the better load balancing of the cyclic mapping.

Figure 15 shows a breakdown of latency for each of the three operations in the algorithm. Algorithm execution time is clearly dominated by the Householder QR decomposition. Execution time for the cyclic mapping is approximately half that of the block mapping because of the better load balancing of the cyclic mapping. In the Householder QR decomposition,

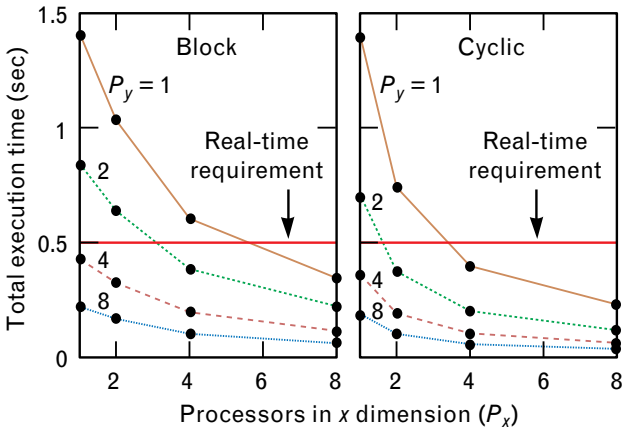


FIGURE 14. Radar 1 real-time processing. Real-time requirements can be met with a minimum of four processors for either mapping.

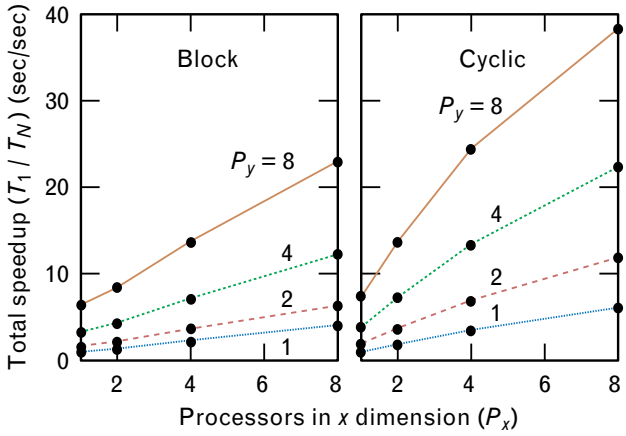


FIGURE 16. Radar 1 speedup. Speedup is better with the cyclic mapping because of the better load balancing among processors.

algorithm activity progresses to the lower right corner of the matrix. Thus, with the block mapping, processors in the upper left corner of each processor subarray become inactive in the beginning stages of the algorithm. With the cyclic mapping, all processors stay active for more iterations of the algorithm.

Figure 16 shows speedup and efficiency of the entire Radar 1 application. Speedup is defined as the ratio of execution time with one processor (T_1) to execution time with N processors (T_N). Once again we see that the cyclic mapping is more efficient because of better load balancing between processors. We also determined that parallel efficiency, as defined in Table 7, is high for lower numbers of processors, i.e., four or eight processor elements per problem.

Figure 17 shows the communication-to-computation ratio for the entire Radar 1 application. These graphs show that the cyclic algorithm, although it has a better execution time because of better load balancing, has a worse communication-to-computation ratio than the block mapping. This result matches expectations, since better load balancing means spreading data over a higher average number of processors per iteration. The increased number of processors lowers latency, but increases the communication burden in the distributed algorithm.

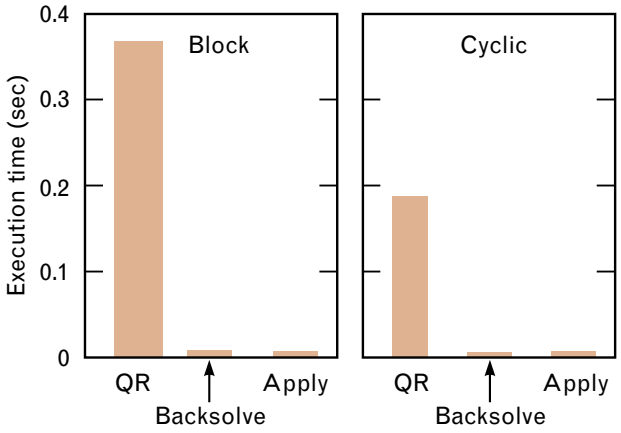


FIGURE 15. Radar 1 mapping analysis. The majority of algorithm execution time is spent in the Householder QR decomposition. Execution time with the cyclic mapping is lower because of better load balancing.

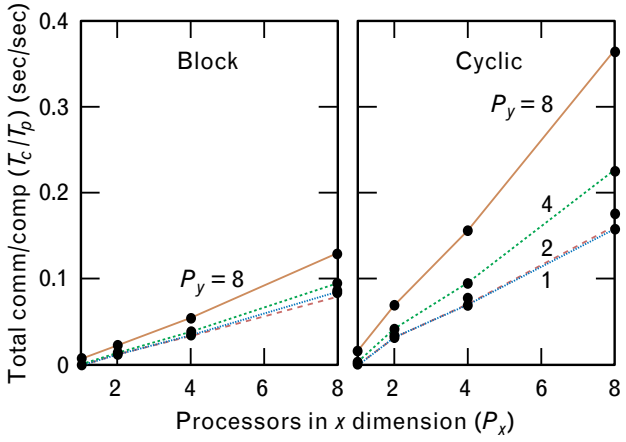


FIGURE 17. Radar 1 communication-to-computation ratio. This ratio is higher with the cyclic mapping because data computation is distributed over more processors, which requires greater interprocessor communication.

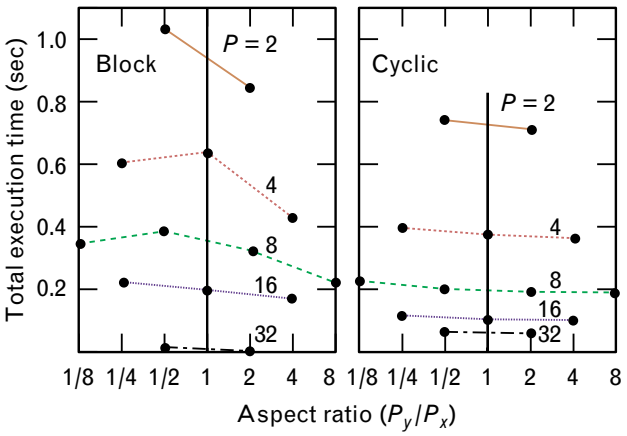


FIGURE 18. Radar 1 aspect ratio. In general, execution time is lower when P_y is greater than P_x for both the block mapping and the cyclic mapping.

Figure 18 shows the effect of adding processors in the *x* dimension versus adding processors in the *y* dimension. In each graph, as we move to the right along the *x*-axis, the aspect ratio of *y* processors to *x* processors increases; if the ratio is greater than one, then there are more *y* processors than *x* processors. Execution time decreases slightly as we increase *y* processors relative to *x* processors, slightly favoring *y*-rectangular processor arrays. Because reduction versus broadcast operations vary in execution time in proportion to log(*N*) versus *N*, this result again meets expectations.

Figure 19 shows memory and input/output requirements for different processor configurations. We can see that the required memory just fits into that provided by the four processors per matrix decomposition. Input/output requirements are well within the capabilities of the MeshSP.

The performance results for Radar 1, on the basis of the performance model, indicate that we can meet real-time requirements for Radar 1 with approximately sixteen MeshSP processor boards. The results show that the Radar 1 application scales reasonably on the MeshSP architecture, and suggest a preference for cyclic mapping and *y*-rectangular processor aspect ratio.

Radar 2 Performance Results

Figure 20 shows the variation in estimated latency for each mapping of the HOPD algorithm with the Radar 2 radar parameters. As shown in Table 6, the "shape" of the Radar 2 radar application is different from that of the Radar 1 problem; Radar 2 has many range gates and smaller degrees of freedom, whereas Radar 1 has fewer range gates and larger degrees of freedom.

Figure 21 shows a breakdown of latency for each operation in the algorithm. Because the Radar 2 case has many more range gates, algorithm execution time is dominated by the weight application (the application of weights to the entire data cube). Although the cyclic mapping still outperforms block mapping for the Householder matrix decomposition, it is equiva-

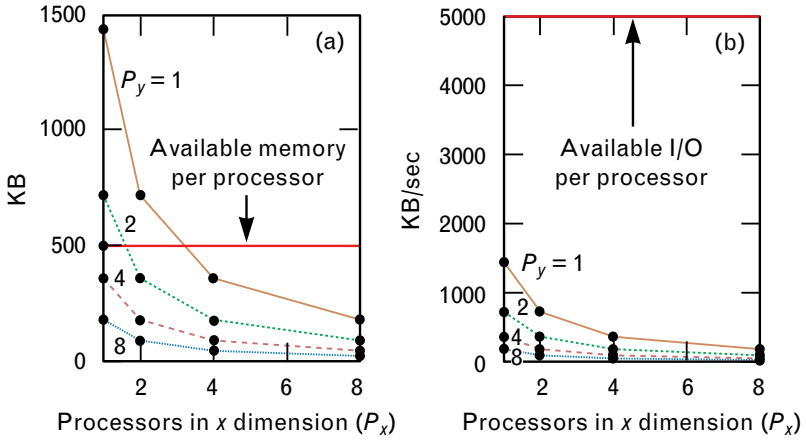


FIGURE 19. (a) Radar 1 required memory per processor and (b) required input/output per processor. Radar 1 processing fits into available memory limits with a minimum of four processors, and within input/output limits for all processor sizes.

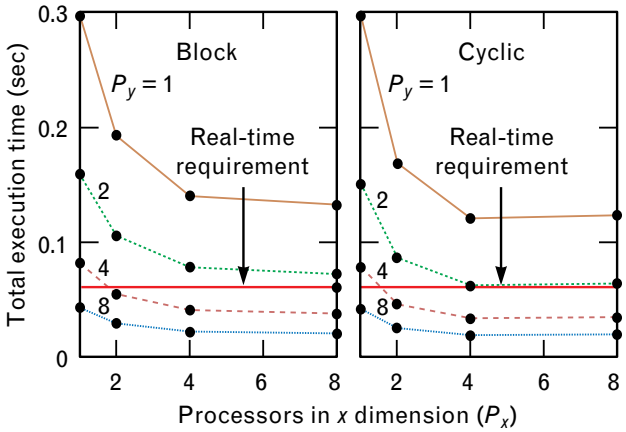


FIGURE 20. Radar 2 real-time processing. Real-time requirements can be met with a minimum of eight processors for either mapping.

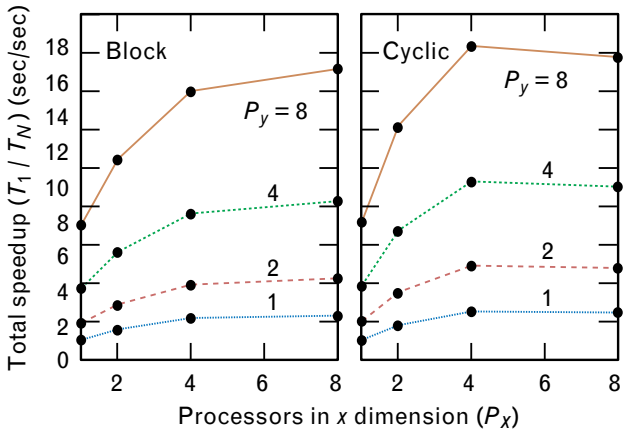


FIGURE 22. Radar 2 speedup. The total speedup, which is approximately the same for both mappings, increases only marginally for values of P_x greater than four.

lent to the block mapping for the weight application. This result meets expectation, since reduction operations as used in the weight application simply add N numbers distributed over P processors, irrespective of the mapping.

Figure 22 shows speedup and efficiency of the entire Radar 2 application. In this case, the cyclic mapping is only slightly more efficient than the block mapping, since weight application dominates execution time and is mapping-insensitive. Again, we see higher efficiency for lower numbers of processors.

Figure 23 shows the communication-to-computation ratio for the entire Radar 2 application. Again, the cyclic algorithm has a communication burden that is only slightly higher because of the distribution over more processors, since weight application dominates. We can also see that adding y processors does not increase communication significantly. This result also meets expectation, because the weight application communicates only in the x direction. We would therefore also expect that this algorithm would strongly favor distribution over processors in the y dimension over distribution over processors in the x dimension. This result can be seen in Figure 24. Lastly, Figure 25 shows the memory and input/output requirements for different processor configurations. We

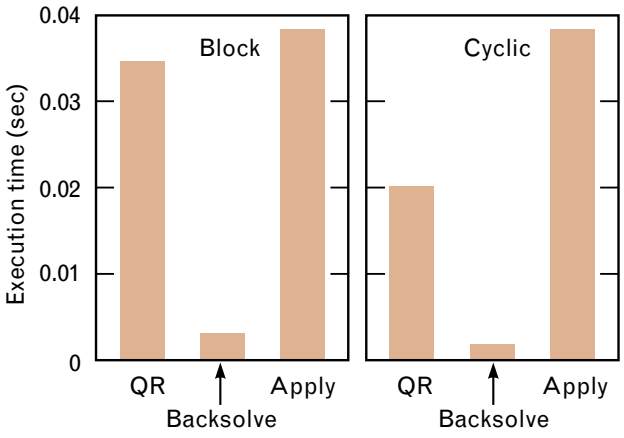


FIGURE 21. Radar 2 mapping analysis. Unlike Radar 1, the majority of algorithm execution time for Radar 2 is spent in the weight application, which is not influenced by the mapping.

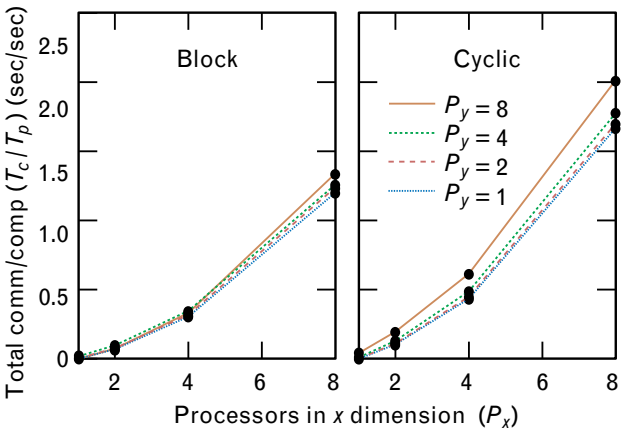


FIGURE 23. Radar 2 communication-to-computation ratio. This ratio is higher with the cyclic mapping; it varies only slightly with changes in P_y because the weight application communicates only in the x dimension.

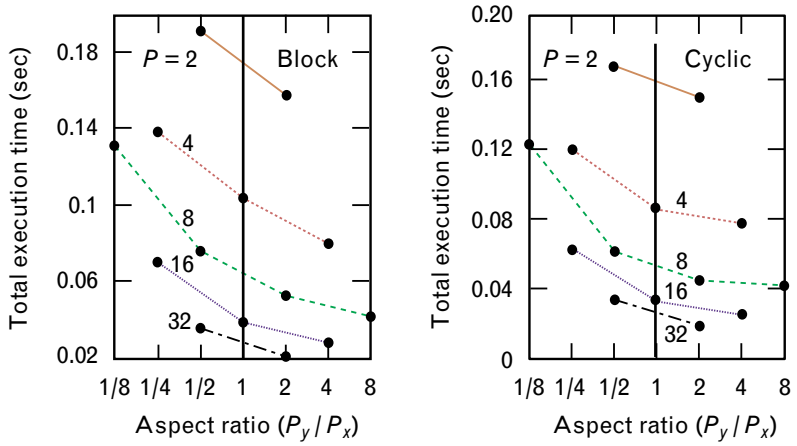


FIGURE 24. Radar 2 aspect ratio. Execution time is always lower when P_y is greater than P_x for both the block mapping and the cyclic mapping.

can see that the required memory just fits into that provided by the eight processors per matrix decomposition, but the fit is very tight. For the Radar 2 case, as with the Radar 1 case, input/output requirements are also within capabilities of the MeshSP.

These graphs show that real-time requirements can be met with eight processors per sample-matrix-inversion problem with each mapping, for a total of 512 processors for all sixteen Doppler bins and four range segments. The cyclic mapping gives slightly more flexibility in mapping choice, but not as much as with the Radar 1 radar parameters. These graphs also show that adding processors in the *x* dimension after a certain point does not increase performance significantly, whereas adding processors in the *y* dimension does increase performance significantly.

From the performance results for Radar 2 based on the performance model, we can meet real-time requirements for Radar 2 with eight MeshSP processor boards. The results show that the Radar 2 application scales reasonably on the MeshSP architecture, and suggest a preference for cyclic mapping and *y*-rectangular processor aspect ratio.

Future Work

With this analysis as a basis, the next step is to implement and optimize the algorithm in order to verify the performance model here and to derive the exact measured efficiency of a representative STAP algorithm on the MeshSP architecture. Another area for

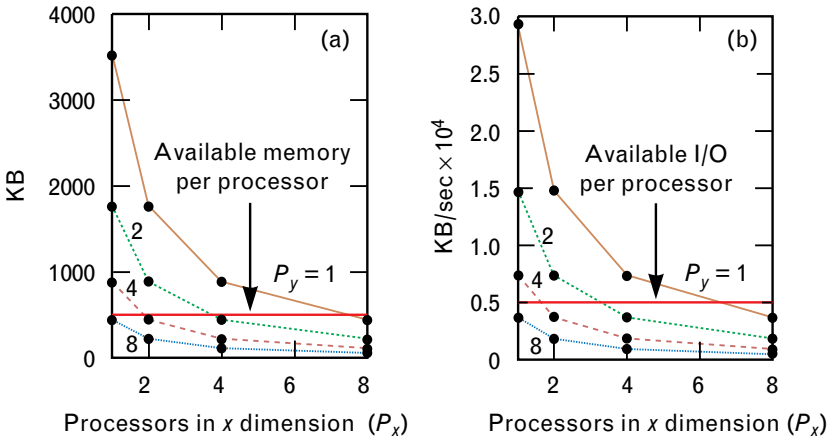


FIGURE 25. (a) Radar 2 required memory per processor and (b) required input/output per processor. Radar 2 processing fits into available memory and input/output limits with a minimum of eight processors.

future work is that of algorithm optimization, which was not addressed here. Other issues that must be solved for a feasible system solution are the input/output and radar-interface issues.

In actuality, the overall processor efficiency is the product of the parallel efficiency and the single-processor efficiency, which was assumed in this study to be 25%. More accurate estimates of overall processor efficiency must utilize single-processor efficiencies derived from measurements of execution times on actual processors. These estimated measurements must be made in order to derive actual efficiencies.

Other STAP processing stages, such as channel equalization, pulse compression, Doppler filtering, and detection, must be included in the processing stream for analysis. In these stages, the parallelism is usually on a different dimension from that of the parallelism in the HOPD algorithm presented here. Since the optimal mapping for each stage is potentially different, an analysis must be done of the additional communication costs caused by interstage remappings. There is a trade-off between optimal mappings for each stage with interstage remappings and suboptimal mappings for each stage with lower interstage remapping costs. The remapping step often involves a major rearrangement of the input data, commonly referred to as *corner turn*. Corner-turn operations involve all nodes in the parallel processor and therefore require high levels of sustained interprocessor communication bandwidth. The ability of an architecture to implement remappings must be analyzed and understood in the context of STAP algorithms and their associated implementation tradeoffs. Future versions of the performance model presented here will be enlarged to include this concept.

Conclusion

This article has demonstrated an approach to measuring performance and scalability of a demanding kernel in STAP algorithms on a commercially available massively parallel processor called the MeshSP. For each radar system evaluated, we can meet real-time requirements with 1024 processors or fewer (eight or sixteen individual processor boards), with available on-chip memory, and in a form factor that is not unreasonable for an airborne platform. The algorithms scale well on the architecture, and we have been able to compare different mappings of data to processors in order to determine which provides the best performance. We have also been able to suggest processor aspect ratios that best suit the algorithm.

On the basis of the parallel efficiencies attained in this study, SIMD seems to be a good architectural match for the single STAP computational kernel presented here, both in scalability and in delivered processing power per unit size, weight, and power. Follow-on work that must be performed in order to fully understand the match between the SIMD architecture and STAP algorithms includes enlarging the model to include the other stages of STAP computations as well as obtaining measured efficiencies on actual MeshSP hardware.

Acknowledgments

This work was originally performed with Ken Teitelbaum, who brought valuable insight and contributed significant direction to this project [2]. The author is also indebted to Robert Bond for his important contributions and feedback. The STAP material presented in this article is summarized from an original report by James Ward [3], where a more detailed treatment can be found.

REFERENCES

1. L.E. Brennan and I.S. Reed, "Theory of Adaptive Radar," *IEEE Trans. Aerosp. Electron. Syst.* **9** (2), 1973, pp. 237–252.
2. J.O. McMahon and K. Teitelbaum, "Space-Time Adaptive Processing on the Mesh Synchronous Processor," *10th Int. Parallel Processing Symp., Honolulu, Hawaii, 15–19 Apr. 1996*, pp. 734–740.
3. J. Ward, "Space-Time Adaptive Processing for Airborne Radar," *Technical Report 1015*, MIT Lincoln Laboratory, DTIC #AD-A293032 (13 Dec. 1994).
4. G.H. Golub and C.F. Van Loan, *Matrix Computations,* 2nd ed. (Johns Hopkins, Baltimore, 1989).
5. C.M. Rader and A.O. Steinhardt, "Hyperbolic Householder Transformations," *IEEE Trans. Acoustics, Speech Signal Process.* **34** (6), 1986, pp. 1589–1602.
6. I.H. Gilbert and W.S. Farmer, "The Mesh Synchronous Processor MeshSP™," *Technical Report 1004*, MIT Lincoln Laboratory, DTIC #AD-A289875 (14 Dec. 1994).
7. C. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steele, Jr., and M.E. Zosel, *The High-Performance FORTRAN Handbook* (MIT, Cambridge, Mass. 1994).

JANICE ONANIAN MCMAHON is a staff member in the Digital Radar Technology group. Her research interests are in the application of massively parallel processing technology to advanced signal processing applications. She received an S.B. degree in computer science and engineering, and an S.M. degree in electrical engineering and computer science, both from MIT. Before joining the staff of Lincoln Laboratory in 1995 she worked at MasPar Computer Corporation.