Application of Parallel Processors to Real-Time Sensor Array Processing[1]

David R. Martinez
MIT Lincoln Laboratory
dmartinez@ll.mit.edu

Abstract

Historically, most radar sensor array processing has been implemented using dedicated and specialized processing systems. This approach was necessary because the algorithm computation requirements were several orders of magnitude higher than any commercial supercomputer could provide in an acceptable size, weight and power. Most recently, with the interest by the military to employ commercial-off-the-shelf (COTS) technology, and the rapid advances in computation throughput by COTS supercomputers, we are able to meet the present algorithm requirements with COTS massively parallel processors (MPPs). However, there are still many issues to be overcome to get the most out of a parallel processor for our application. Some of the important issues are the balance between high computation throughput and total exchange communication over the processor network, message passing or shared memory access with high communication throughput for small message sizes, and the scalability of these machines to more demanding algorithm and application requirements. In this paper we present the application requirements, review the important computational kernels, and conclude with our experiences in evaluating representative MPPs.

1. Introduction

The next advances in electronically phased-array radars will be in the implementation of adaptive signal processing algorithms [1],[2]. Many of today's military assets implement very conventional signal processing techniques. As the need for battlefield superiority increases, many of the present military platforms will undergo major upgrades in COTS computing technology. Supercomputers will play a very important role in these upgrades because of the cost-to-performance ratio advantage provided by COTS parallel processors relative to an all custom solution, and also because of the flexibility provided by a programmable system in being able to download different types of sensor array processing algorithms. There will still be sections in the processing chain that can not be economically solved with COTS programmable processors. In these instances, we will employ COTS interfaces but the processing will be performed using very large scale integration (VLSI) circuits, designed for application-specific functions.

Sensor array processing demands very high computation throughput in real-time. The example used throughout this paper describes representative requirements of space-time adaptive processing. The typical computation throughput requirement ranges from 100 to 1000 billion operations per second (GOPS). This operation throughput results from the aggregate of operations performed for digital filtering, fast Fourier transforms, adaptive matrix inversion and matrix multiplies on multiple sensor data channels. One can reach these levels of throughput by concatenating a large number of processors in a pipeline fashion. The problem with this approach is that the demands are not just in computation throughput, but also in latencies, on the order of a few seconds. This application is well matched to parallel processors to meet both the computation and latency requirements.

Parallel computers, organized with a number of low-cost, off-the-shelf microprocessors, offer a very attractive solution. However, there are a number of challenges faced by parallel computers to reach the high computation rates demanded by the application. One important challenge is the ability to break up the application problem into enough degrees of parallelism (DOP) such that the processing tasks can be performed concurrently [3].

Another important challenge for these parallel computers is the ability to do a total exchange of the data within a minimum latency and a maximum bandwidth. This all-to-all communication is particularly important

[1] This work is sponsored by DARPA/ETO, under Air Force Contract F19628-95-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Air Force.

for sensor array processing because the sequence of operations must be performed on multiple dimensions of the input sensor data. For example, digital data filtering is normally performed on a channel-by-channel independent basis. However, the matrix inversion is performed by combining multiple sensor channels. Therefore, a large bisection bandwidth, balanced with the total computational power of the machine, is needed to reach the requisite performance.

In addition to raw power in processing and communication, we are very interested in standards for communication and signal processing software libraries. The message passing interface standard (MPI) [4] was a significant step forward in formulating a set of communication libraries that the parallel processing community could implement to maintain portability across machines, that comply with a distributed memory architecture. More recently, a number of computer manufacturers have put forward another standard, referred to as OpenMP, perhaps better suited to shared memory architectures [5]. We are also starting to see efforts in standards for signal processing software libraries targeted at real-time implementations. The Defense Advanced Research Projects Agency (DARPA) has started an effort to standardize vector signal and image processing (VSIP) library functions [6]. The VSIP standardization has been principally focused, so far, on functions operating on a single microprocessor. The communication standards, like MPI or OpenMP, are mostly limited to nonreal-time applications. DARPA has also started to extend MPI into real-time [7]. These standards are very important to maintain portability in legacy code, and therefore to minimize the software investment.

In this paper we describe the application of supercomputing systems to the very challenging problem of real-time sensor array processing, and present our experiences in prototyping a class of MPP systems. The remainder of the paper is organized as follows. Section 2 presents an overview of the sensor array processing algorithms. In Section 3, we formulate the architecture best suited for our application, and also address the CPUs and interconnection networks used in typical implementations. Section 4 describes experiences in benchmarking a class of sensor array processing algorithms on different types of parallel processor systems. Section 5 summarizes our conclusions.

2. Sensor array processing algorithms

The computational requirements in radar sensor array processing arise from the need to cancel unwanted interference and improve the information signal-to-noise ratio [8]. The mainbeam clutter (ground clutter coming through the mainbeam of the sensor antenna) and the sidelobe clutter (ground clutter interfering with the signal of interest through the antenna sidelobes) are typically 50 to 60 dB higher than the signal. Therefore, the digital processing algorithms must be able to null the interfering signals and gain in signal relative to residual noise to a point that a small target can be detected and tracked. Many sets of adaptive antenna weights (ranging from 100 to 1000) must be computed in real-time. Each adaptive computation requires a matrix inversion within a latency of less than milliseconds to seconds.

However, prior to the computation of the adaptive weights, the input sensor data must be conditioned to meet the interference cancellation requirements stated earlier (50–60 dB). The data conditioning is performed by filtering the data using a combination of finite or infinite impulse response filters (FIR and IIR filters) and fast Fourier transform (FFT). These computations must be completed in milliseconds. The aggregate in operations, for data conditioning, adaptive weight computation and application, imposes computation throughputs ranging from 100 to 1000 GOPS/sec, within a latency of less than a few seconds. Parallel processors are well matched to these requirements. In the next sections, we address each processing stage and the key computational kernels in more detail.

2.1 Signal processing functions

Figure 1 illustrates the order of magnitude in data and computation throughput requirements, in parametric form, at each processing stage. This signal processing flow is typical of space-time adaptive processing. Data arrive from the ADCs at a rate commensurate with the output sampling rate and the number of channels. For example, for an ADC with a sampling rate of 100 million samples per second (two bytes per sample), and the number of channels ranging from 10 to 100, we are faced with an input data rate in the range of 2 to 20 billion bytes per second (GBytes/sec).

This large data rate must be processed first through a set of FIR filters. Data filtering is necessary, as the first step in the processing flow, to convert the incoming real data to complex in-phase and quadrature samples, and to pre-condition the data for subsequent processing. The FIR filtering, in computation throughput, is proportional to the output data bandwidth, the number of sensor channels and the length of the filter (number of taps). The output bandwidth is commonly a factor less than the input data rate. A typical decimation factor ranges from 2 to 4. Therefore, the data output from the FIR filtering ranges from 500 MBytes/sec to 10 GBytes/sec. For this example with tens to hundreds of channels, and a number of taps equal to 256, we can reach a computation throughput ranging from tens to hundreds of sustained GOPS/sec. Clearly for the upper range in computation

(TERAOPS/sec), it is most effective, in size, weight, power, and cost, to employ dedicated VLSI, constructed to perform very specific functions. However, as the processing capability increases for off-the-shelf programmable microprocessors, we can bring more and more COTS parallel processing towards the front-end of the processing flow.

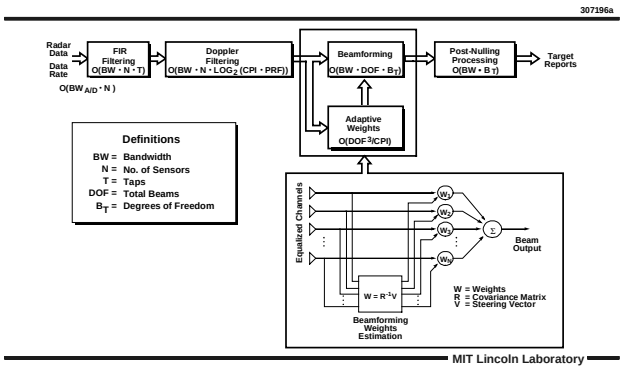

Figure 1. Signal processor system complexity.

Doppler filtering follows FIR filtering, as shown in Figure 1. The data throughput is on the same order of magnitude as the output of the FIR filtering function (500 MBytes/sec to 10 GBytes/sec). The computation throughput is about one to two orders of magnitude less than FIR filtering, because of the FFT being proportional to Log_2 of the number of pulses transmitted by the sensor system (CPI x PRF). Therefore, the computation throughput at the Doppler filtering stage ranges from one to tens of sustained GOPS/sec. However, in contrast to the FIR filtering function where we normally operate in the range dimension, for Doppler filtering we have to operate on a cumulative set of radar pulses. If these operations are mapped to parallel processors, we are required to perform either an in-place corner turn or a global corner turn. The choice depends on whether there is enough memory at the nodes containing each sensor channel to facilitate an in-place data transpose. Further discussion related to algorithm mapping to a parallel architecture is presented in the next subsection.

After completing the data pre-conditioning through the FIR and Doppler filtering operations, we are now ready to perform the jamming and ground clutter interference suppression. As shown in Figure 1, there are two fundamental steps. We first compute adaptive weights and then apply the weights to the data to form output beams.

The computing of the adaptive weights consists of inverting a covariance matrix R constraint to the steering vectors V. The computation of the adaptive weights is proportional to DOF^3; where DOF represents the number of rows in the matrix. For the sensor array to adapt to the changing environment in real-time, many sets of adaptive weights must be computed concurrently over a time commensurate with a coherent processing interval (CPI). This time interval ranges from tens to hundreds of milliseconds. The data input rate into the adaptive weight computation stage depends on the number of samples used to train the weights. The training samples (i.e., columns in a training data matrix) range from 2xDOF to 5xDOF. The computation throughput, because of the DOF^3 growth, can be on the order of tens to hundreds of sustained GOPS/sec. The parallel processor peak throughput will range from 10 to over 1000 GOPS/sec, depending on the algorithm parameters.

The adaptive weights must then be applied to the sensor data to cancel the jamming and clutter interference. This operation is often referred to as beamforming. Typically, there is a data word length growth, from the ADC 14 bits to 32-bit words by the time the adaptive weights are computed and applied. This word length growth is due to the signal-to-noise ratio gain achieved through the front-end Doppler filtering. The data bandwidth, into the adaptive beamformer, ranges from 1 GBytes/sec to 20 GBytes/sec. The computation throughput for the adaptive weight application is proportional to the data bandwidth, the DOF and the number of beams to form (B_T). For the parameters chosen earlier, 1 GBytes/sec to 20 GBytes/sec, 100 DOF, and a few beams, the computation throughput at this stage can range from hundreds to thousands of sustained GOPS/sec.

A very important requirement of the parallel processor for effectively implementing both the weight computation and application is the ability to do a global corner turn. This total exchange transformation stresses the bisection bandwidth of the machine. Techniques discussed by Sundar, et al. [9] are suited for the global corner turn. Ideally, we desire to hide this communication time relative to the total latency available for the computation. We also desire a parallel processor that has a bisection bandwidth that grows at a constant rate as its available computational throughput is increased. Teitelbaum [10] presented a thorough analysis of bisection bandwidth for a class of processors with a crossbar interconnect topology.

The last step, shown in Figure 1, is referred to as post-nulling processing. At this stage the data input has been decimated to the few beams of interest. The data throughput is reduced to less than a few GBytes/sec. The computation throughput is proportional to the input bandwidth and the number of beams of interest. The computation throughput is in the range of a few GOPS/sec. For this last processing stage, COTS MPPs can easily meet the computational requirements.

However, the software programming is more complex than earlier processing stages.

In a modern adaptive signal processor we begin with a very large data input rate (GBytes/sec) and high computation throughput (TERAOPS/sec). We enhanced the signal of interest over the background noise by applying advanced signal processing techniques. The resulting output data rate is on the order of hundreds of MBytes/sec. However, the data throughput, computation throughput, and interconnect network bandwidth, for the front-end digital filtering, adaptive weight computation and application, can stress the capability of the most advanced supercomputer systems. The parallel processor chosen, for embedded array signal processing applications, will also need to meet stringent ruggedization and form factor requirements.

2.2 Algorithm mapping and computational kernels

The sensor array processing algorithms can be mapped to a parallel processor by exploiting several DOP available in the data. McMahon [11] presented several examples of mapping a class of adaptive sensor array processing algorithms in to an embedded signal processor. McMahon demonstrated the required balancing between the DOP available in the data and the available computation and memory at the node. The DOP available in the data vary significantly from application to application. Unfortunately, the algorithm mapping today is a task done by hand since there are no automated tools to effect this mapping directly with little intervention by the user.

In the digital filtering stage, the data arrive from the ADC on a channel-by-channel parallel format and all of the filtering is done in either a range swath or a set of pulses. Therefore, one can subdivide the computational tasks in a channel parallel form by distributing the data to multiple nodes; each node contains all of the range and pulse data for that channel. The key computational kernels to be performed are FIR or IIR filters and/or FFTs. If we take, for example, typical communication and computation throughputs, 2000 MBytes/sec and 100 GOPS/sec, respectively, the data can be broadcasted to a set of nodes to operate on each respective channel in parallel. For example, a system with 48 nodes requires each node to hold several MBytes of data (for a few hundreds of milliseconds in latency) and to sustain a computation throughput of 2 GOPS/sec. In today's supercomputers, over 64 MBytes per node is sufficient to store data, the application program, and the local operating system and library functions. The problem is in computation. Today a single CPU is not able to reach a sustained floating point throughput of 1 GOPS/sec. Therefore, depending on the delivered efficiency[2] from the node, we are forced to split the computational kernels across several microprocessors. This problem is exacerbated when we are required to meet the upper end of the data and computation requirements (several GBytes/sec and TERAOPS/sec). It is important to remember that at this stage of the processing, the maximum latency allowed is on the order of tens to hundreds of milliseconds (a radar coherent processing interval, or CPI) to meet real-time.

Once we completed the digital filtering, we must do a global corner turn. In real-time we must do a global corner turn commensurate with the real-time data throughput. For example, over a time interval of 100 milliseconds, the total data exchange would be 100 MBytes (for the low end of the data rate, 1 GBytes/sec). If we want to distribute these data across, for example, 96 nodes, we would need to transfer a message of about 1 MByte per node. This small size message begins to stress the communication network to a point where the startup latency can dominate the achievable communication throughput. Thus, we need high bisection bandwidths (on the order of several GBytes/sec), but with the ability to achieve a sustainable bandwidth close to the maximum hardware communication throughput for small message sizes. As we increase the data throughput to 20 GBytes/sec, we also demand more computation and higher bisection bandwidth. The ability of a processor to scale with a balanced architecture between bisection bandwidth and aggregate computational speed is very desirable for sensor array processing. The work by Zhang and Xu [12] presents techniques for quantifying the scalability of a multiprocessor system.

In the adaptive beamforming stage (including both weight computation and application) for each Doppler, we compute a set of adaptive weights and apply the weights to the same Doppler data. Thus, at this stage of processing the operations are Doppler parallel. Let us take the example with the low end of the data communication and computation throughputs (1 GBytes/sec and 100 GOPS/sec, respectively). We assume a time interval of 100 milliseconds. The number of available processing nodes is assumed to be 96 (same as the DOP available in the data). Typical memory size of greater than 64 MBytes at the node in today's supercomputers is sufficient to hold the distributed sensor data and other ancillary data. The problem is again with the computation per node. For this example each node must perform about 1 GOPS/sec of sustained floating point throughput. Therefore we are forced to divide the problem into a level of medium-grain

[2] Efficiency is defined as the ratio of delivered processing throughput over the peak throughput.

partitioning across several microprocessors. For the upper end of the communication and computation requirements, we would be forced to go into a fine-grain mapping of the algorithm across multiple microprocessors. Today's parallel processors are dismal in the implementation of fine-grain partitioning, resulting in efficiencies of less than 10%.

The key computational kernels at the adaptive beamforming stage are matrix inversions for the adaptive weight computation, and matrix multiplies for the weight application. QR factorization techniques are often employed in the matrix inversion. The efficiency achieved for a QR factorization kernel varies depending on the specific QR algorithm used and the particular microprocessor (RISC or DSP based).

3. Application of parallel processing systems

The supercomputing industry is undergoing a significant resurgence in the development of parallel processing systems. We are beginning to see architectures derived from commodity microprocessors and off-the-shelf DRAM chips. Companies like IBM, SGI/Cray, and HP/Convex are putting significant effort into systems with the capability to grow from the "laptop to teraflops." Other important players in the parallel processor market are Sun Microsystems and COMPAQ computers. One main distinguishing difference among these systems is in their high speed interconnection networks.

The future military upgrades will benefit from the availability of COTS parallel processor systems, and the implicit cost advantages over more custom solutions. Some of these systems are able to meet the low end of the requirements matrix. They are far from meeting several GBytes/sec and tens of TERAOPS/sec of sustained throughput requirements, discussed in the previous sections, within an affordable form factor. In this section, we address the architecture, classes of CPU, and interconnect topology best suited for our application.

3.1 Processor architecture taxonomy

A complete analysis of processor architectures is beyond the scope of this paper. Instead we like to adopt the terminology and taxonomy presented by Kumar [13], then proceed to identify the characteristics, within this taxonomy, best matched to our sensor array processing problem. Kumar classifies the taxonomy of a parallel architecture by separating the different elements of an architecture into the following attributes:

- Control mechanism
- Address-space organization
- Processor granularity
- Interconnection networks

As for the processor control mechanism, we can opt for a single instruction stream, multiple data stream (SIMD) architecture, or choose a multiple instruction stream, multiple data stream (MIMD) architecture. For the sensor array processing application, the architecture best matched to the problem is MIMD. There are several reasons for this choice. First, even though the application is highly data parallel, typical SIMD processors have very anemic microprocessor engines resulting in a need for thousands of microprocessors to meet the latency requirements. Processor efficiencies catastrophically drop when the problem is partitioned among those many processors since the data set does not have that many degrees of parallelism (thousands). The second important reason for selecting a MIMD architecture is the ability to implement a different processing function in one part of the parallel processor while concurrently performing a separate processing stage on another section of the processor. For example, we could allocate a set of processors to perform adaptive beamforming on earlier data, while another set of processors are performing digital filtering on more recent incoming data.

The best address-space organization for our application is not as easy a choice as the selection of the control mechanism. There are fundamentally two classes of address-space organizations: a message-passing architecture and shared-address-space organization. Message-passing architectures require that the programmers manage the sending and receiving of messages across the network. In contrast, the shared-address-space architecture provides the programmers with the hardware mechanism for accessing variables across a single memory address space; the memory might be physically located within a cluster of memory banks, or physically distributed at each of the processor nodes.

In the case of memory physically distributed at each of the processor nodes, when the time to access a remote memory location is more than the time to access a local memory location (typically the case), the architecture is referred to as nonuniform memory access (NUMA). If in addition to NUMA, the parallel processor also supports cache coherence to make sure that copies of the same variables, present in different remotely located caches, are updated as modifications are made by the microprocessors, then the architecture is categorized as cache coherent-NUMA (CC-NUMA). Programmers tend to favor a CC-NUMA architecture over a message passing architecture, because of the ease in programming. The best architecture for the sensor array processing problem is one with the lowest startup latency for small messages and the highest communication bandwidth. This way we would devote most of the

sensor timeline to computation, instead of spending a major portion of this timeline communicating data.

A MIMD control mechanism is found in several of today's MPPs using a message-passing organization (e.g., Mercury, Sky, CSPI, or Intel Paragon computers), or with a shared-address-space organization (e.g., SGI/CRAY or HP/Convex computers). Examples of CC-NUMA architectures are the SGI/CRAY Origin 2000 or the HP/Convex Exemplar.

3.2 Classes of microprocessors

Microprocessors fall into the class of either complex instruction set computer (CISC) like the Intel Pentium microprocessor, the reduced instruction set computer (RISC) like the DEC Alpha microprocessor, or digital signal processors (DSP) like the Analog Devices SHARC microprocessor. For the last several years, all of these microprocessors have enjoyed the advances in integrated circuits, resulting from reduction in lithography feature size, as predicted by Gordon Moore of Intel (known as Moore's law). Figure 2 illustrates a trend in microprocessor evolution [14],[15]. Yu predicts that by the year 2006, we will have microprocessors with over 350 million transistors operating at 4 GHz clock speeds. The limiting factor will be memory speed in and out of the microprocessor. This limitation is one of the reasons why more of the microprocessor transistors are being dedicated to on-chip memory as shown in Figure 2.

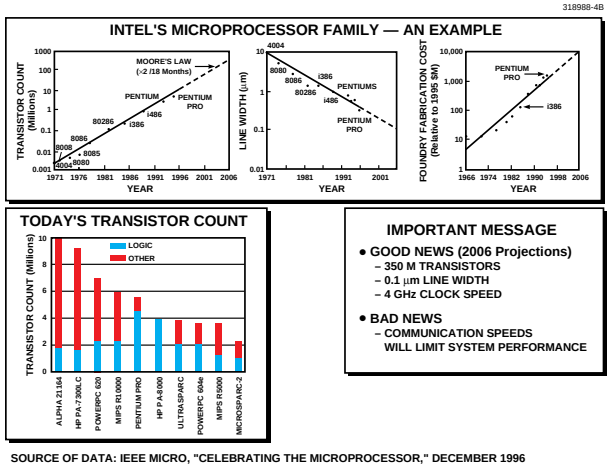

Figure 2. Trends in microprocessor evolution.

We can arrange these commodity microprocessors in different configurations supported by the interconnection network (discussed later). The granularity of the programming implementation will depend on how efficiently we can concurrently utilize several microprocessors. For example, if the computational kernel (FIR, FFT, or matrix inversion) is partitioned such that for every one or few instructions a data word is communicated in the network, from and to other microprocessors, then this program partitioning is referred to as fine-grain. On the other hand, if the computational kernel is partitioned such that many program instructions are executed before a data word is accessed, from one to another microprocessor, then this program partitioning is referred to as course-grain. Categorically, we have found that approaching coarse-grain algorithm partitioning is the preferred choice for the sensor array processing problem. This result is a direct outcome of a degrading network communication bandwidth as the message size is decreased. For a small message size, the communication is dominated by startup latencies. Therefore, we prefer very few but powerful microprocessors per node. Based on the typical sensor parameters presented earlier, microprocessors with greater than 1 GOPS/sec of sustained floating point throughput at the node.

3.3 High speed interconnection networks

As alluded to by Kumar, message-passing address-space organization or shared-address-space memory architectures can be interconnected via different types of interconnection networks.

The most appropriate network topology must result in a low latency (low startup cost) and high communication bandwidth for a small message size. In some of our prototype demonstrations, we require to move KBytes of data across hundreds of MBytes/sec links. The performance of the interconnection network will be influenced by not just the startup hardware latency or the communication bandwidth over a hardware link, but also by the communication functions used to set up and transfer a message (software latency). In several of our measured results, we have been unable to achieve greater than 50% of the available total peak bandwidth for a few KBytes in data size. This result is one example showing why the sensor array processing algorithms are best matched to a coarse-grain partitioning, with limited communication across the network.

4. Benchmark results

For the last six years, MIT Lincoln Laboratory has been engaged in benchmarking several representative classes of parallel processor architectures. We concluded that an architecture based on a MIMD control mechanism resulted in the highest levels of multiprocessor efficiency ranging from 21% to 43%. The efficiency varied depending on how much assembly code was used in the implementation of compute intensive functions.

Another important lesson learned was the demonstration that a medium to coarse-grain mapping was more suitable to our application than a fine-grain mapping approach. From these results we also concluded that a 25% parallel processor efficiency was acceptable as an overall figure of merit. The total effort of implementing the benchmark suite on a single parallel processor system ranged from 12 to 15 man-months. Another benchmark suite containing a similar set of algorithms for evaluating embedded processors have been developed by MITRE [16].

5. Conclusions

The application of supercomputers to a real-time sensor array processing problem is very demanding. We require tens of GBytes/sec of data I/O and similar order of magnitude in bisection bandwidth. The computation throughput ranges from hundreds of GOPS/sec to over several TERAOPS/sec, depending on the specific algorithm parameters. Memory is not as severe a requirement as communication and computation, because we distribute the data across tens to hundreds of nodes.

Fortunately, the sensor array processing problem has several degrees of parallelism inherent in a multi-dimensional data. We exploit these DOP to achieve concurrency across the processor architecture. This feature makes the application very well matched to a parallel processor system to meet the throughput within the sensor's allowed latency (milliseconds to seconds).

Through many benchmark experiments, we have found that a MIMD architecture is best matched to the application. The software programmers also favor a CC-NUMA class of memory organization because of the ease in programming. The main challenge for the interconnection network is in the ability to sustain a large percentage of peak bandwidth for a message size less than 1 MByte. Today, the low communication bandwidth leads us to a coarse-grain algorithm partitioning. Finally, the ideal processor architecture for the application would have powerful microprocessors at the node capable of sustaining over 1 billion floating point operations per second.

References

[1] A. Farina, *Antenna-Based Signal Processing Techniques for Radar Systems*, Artech House, Inc., 1992.

[2] D. Martinez, F. Lee, and M. Davis, "Space-Time Adaptive Technology Applied to Airborne Early Warning Radars," 41st Annual Tri-Service Radar Symposium, June 27–29, 1995.

[3] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., 1993.

[4] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, 1994.

[5] OpenMP: A Proposed Industry Standard API for Shared Memory Programming, October 1997, http://www.openmp.org.

[6] D. Schwartz, "VSIP: A Standard API for Vector Signal and Image Processing," Proceedings of the High Performance Embedded Computing Workshop, Lincoln Laboratory, Massachusetts Institute of Technology, September 17–18, 1997, http://www.vsipl.org.

[7] A. Skjellum and A. Kanevsky, "Design and Development of the Real-Time Message Passing Interface (MPI/RT) Standard," Proceedings of the High Performance Embedded Computing Workshop, Lincoln Laboratory, Massachusetts Institute of Technology, September 17–18, 1997, http://www.mpirt.org.

[8] J. Ward, "Space-Time Adaptive Processing for Airborne Radar," Lincoln Laboratory Technical Report #1015, 13 December 1994.

[9] N. Sundar, D. Jayasimha, D. Panda, and P. Sadayappan, "Hybrid Algorithms for Complete Exchange in 2D Meshes," Proceedings of the 10th ACM International Conference on Supercomputing, May 25–28, 1996.

[10] K. Teitelbaum, "Scalability of Crossbar Tree Architectures," Proceedings of the High Performance Embedded Computing Workshop, Lincoln Laboratory, Massachusetts Institute of Technology, September 17–18, 1997.

[11] J. McMahon and M. Sexton, "Mapping Analysis of Advanced Processing on a Parallel Architecture," Proceedings of the High Performance Embedded Computing Workshop, Lincoln Laboratory, Massachusetts Institute of Technology, September 17–18, 1997.

[12] X. Zhang and Z. Xu, "Multiprocessor Scalability Predictions Through Detailed Program Execution Analysis," Proceedings of the 9th ACM International Conference on Supercomputing, July 3–7, 1995.

[13] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, The Benjamin/Cummings Publishing Company, Inc., 1994.

[14] M. Slater, "The Microprocessor Today," IEEEMicro, Vol. 16, No. 6, December 1996.

[15] A. Yu, " The Future of Microprocessors," IEEEMicro, Vol. 16, No. 6, December 1996.

[16] C. Brown, M. Flanzbaum, R. Games, and J. Ramsdell, "Real-Time Embedded High Performance Computing: Application Benchmarks," MTR 94B0000145, MITRE, 1994.