# Parallel Digital Signal Processing of Phased Array Radar in Real-time Environments

Pangfeng Liu
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
pangfeng@csie.ntu.edu.tw

Yu-Lin Su
Phased Array Radar Section, Electronic System Division
Chung Shan Institute of Science and Technology
Long-Tan Tao-Yuan, Taiwan, R.O.C.

Yi-Lin Chu Shinn-Liang Chen Cheng-Hsin Chang
Department of Computer Science and Information Engineering
National Chung Cheng University
Chia-Yi, Taiwan, R.O.C.

**Abstract**

This paper describes efficient parallel implementations of digital signal processing in real-time phased array radar systems. A phased array radar system processes a great amount of data in order to respond to the ever-changing combat environment, therefore it is vital that the system is able to process the input data and produce the results in limited time budget. As a result it is inevitable that parallel processing must be adopted into digital signal processing for performance improvements.

Our implementations combine two types of parallelism – data parallelism and task parallelism. The combination proves to be effective in achieving good performance. We also introduce a layered software architecture to ensures software portability and reusability among different platforms, e.g., ordinary PC clusters and real-time multiprocessing systems.

We conduct experiments to verify the performance and software portability and reusability of our implementations. The implementation platforms include a 8-node PC cluster and a Pentek 4290 ReadyFlow system. From the timing data we identify essential factors in parallel implementations. The characteristics of individual radar function modules are identified and will be used as guidance for future implementations. The communication overhead is recognized as the major obstacle, and the role of granularity is carefully studied and discussed through the experiments.

## 1 Introduction

This paper describes parallel implementations of digital signal processing modules used in phased array radar systems. A phased array radar system needs to process a great amount of data in order to make keen judgment and respond to the ever-changing combat environment, and it is vital that the system is able to process the input data and produce the results in

limited time budget. As the rapid development of multi-processing digital signal processing environments dramatically increases the raw computing power of real-time systems, it is inevitable that we must adopt parallel processing into the field of digital signal processing for performance improvements. In this paper we propose practical parallel processing techniques to reduce the execution time of digital signal processing function modules used in phased array radars, and as a result we optimize the overall execution time of the radar operation functions. This not only improves performance but also provides possibilities for more flexibility in real-time scheduling.

## 1.1 A Phased Array Radar System

A typical radar system detects targets by sending out a microwave beam and waits for its reflection. A radar system can compute various parameters of the detected target, including speed, range, and direction, by the collection and processing of the beam reflection data. A traditional radar system has an antenna that directs beams to specified directions by mechanical means. On the other hand, a phased array radar has an array of antenna, and the direction of the antenna can be steered electronically by adjusting the phase of these antenna, i.e., a *beam steering controller*, controls the phase difference of individual antenna so that their energy can concentrate to a specific direction. As a result a phased array radar system can switch its direction to respond to the mission requirements much faster than a traditional radar. As the development of better digital signal processing (DSP) chips, nowadays a typical phased array radar system can support different mission modes simultaneously, including search, track, and missile guidance [1, 2, 8, 11, 12].

As the technology in computer hardware and software develops rapidly, a phased array radar system could no longer be designed just by connecting a collection of complex hardware components together – the ability to substitute one component for another is equally important. This can be verified by the phenomenon that current radar systems consist of commercially-off-the-shelf components [4, 5], which can be replaced by new technology on a component-wise basis. As a result a radar system can be divided into several functional units, and each of them performs a dedicated function. On the other hand, since the functionality is modulized, and the processing speed of processors is increasing rapidly, the function of a hardware component can now be replaced by a software module. This *component-oriented* design philosophy has become the mainstream of modern DSP systems. For example, the Rapid prototype of Application Specific Signal Process (RASSP) [14], which is sponsored by the U.S. Department of Defense, describes the four component-oriented engineering processes, with the goal to reduce development costs and expenses. This framework not only increases the flexibility of radar resource usage, but also creates new possibilities in radar designs. All the signal processing components, e.g., pulse Doppler processing and movement target indication, can be implemented as software modules, and assembled together into DSP radar functions with great flexibility and reasonable costs. When new DSP technology becomes available, the software modules can be ported into the new platform of better technology easily and efficiently.

### 1.1.1 Major Components of A Phased Array Radar

The operations of a phased array radar system can be described as follows (refer to figure 1 for an illustration [2, 11, 13, 18]). The *radar control computer* gives commands to the *signal processing computer* to perform required operations. The commands are then given to the *beam steering controller* for a specific direction, and target-acquiring microwave beams are transmitted from the antenna, which also collects the reflection. The received analog data are processed and converted into digital format by the *analog signal processor*, then placed into the *input buffer unit*. The data will be transmitted via the *data interconnection network* into *vector signal processor* for digital signal processing. The results are then feedback to the signal processing computer, and then to the radar control computer [3].

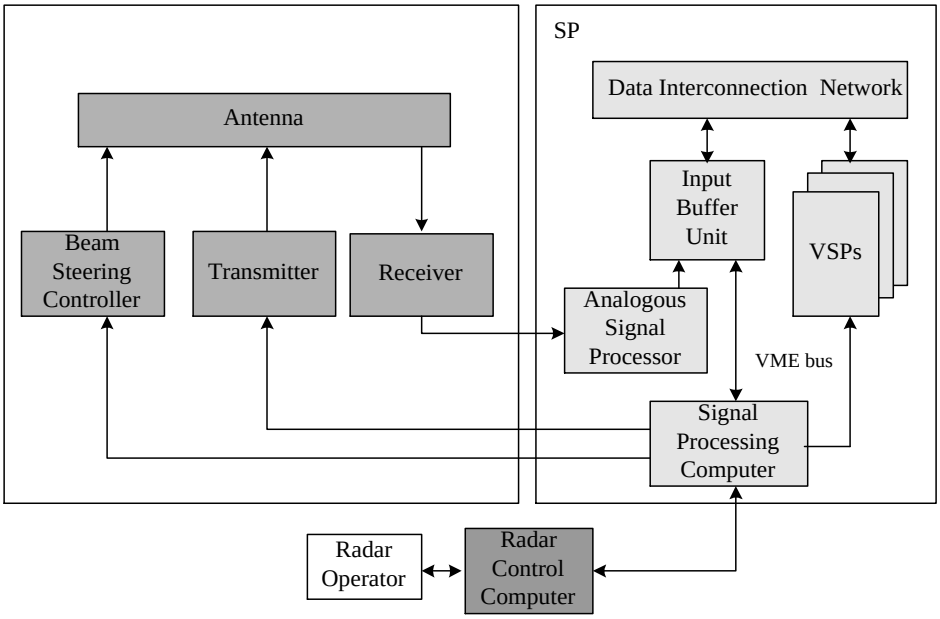



Figure 1: The architecture of a component-based DSP system.

The following is the detailed function description of the major functional units in a phased array radar.

**Beam Steering Controller** The beam steering controller determines the direction a phased array radar searches for targets.

**Radar Control Computer** The radar control computer controls the operations of a radar system by monitoring the system resources and scheduling tasks to be performed, e.g., search, track and missile guidance.

**Signal Processing Computer** The signal processing computer receives the commands from the radar control computer and relays them to transmission/receiving modules. It also collects the targets recognized by the vector signal processors back to the radar control computer.

**Analog Signal Processor** The analog signal processor processes the analog signal from the antenna and feed them to DSP modules.

**Input Buffer Unit** The input data are queued in input buffer unit for DSP processing. The extra storage provide scheduling flexibility for the radar control computer.

**Vector Signal Processor** The vector signal processor is responsible for computing all DSP functions. A vector signal processor usually has several DSP chips, which are connected by a multi-port shared memory.

### 1.1.2 The Data Channels

A phased array radar system usually has two communication channels – a control bus for control signals and a data interconnection channel for DSP data. The control bus (e.g. VME bus) relays commands among signal processing computer, input buffer unit, and vector signal processor. The data connection network are usually bidirectional and non-blocking to provide high bandwidth to transfer data between input buffer unit and vector signal processor.

## 1.2 Basic Operation Mode

A radar system can operate in several basic modes, and it can switch among these basic modes as required by the circumstance. When no target is acquired, the radar operates in *search mode* and scans for possible targets. Three search modes are classified according to the possible targets range – the *horizon search*, *normal search*, and *long range search*. If the radar locates a target, it switches to *track mode* and tries to confirm the finding. The track mode is further classified into *normal track* and *precision track*, depending on the precision required. Also a radar can operate in *missile track mode* to detect and control a missile, or *missile acquisition mode* to track and confirm the movement of a missile. The operation modes are described as follows [2].

**Search Mode**

**Horizontal Search** A horizontal search locates the targets within short range and with a lower angle.

**Normal Search** A normal search locates targets within short range but with a higher angle, which could be missed by the horizontal search.

**Long Range Search** To locate long range targets that could be missed by the horizontal and normal search.

**Track Mode**

**Missile Track** This is usually a part of a defense system that tracks and control missiles fired by friendly forces.

**Normal Track** For targets that do not require precision tracking.

**Precision Track** For target that have specific tracking precision requirements.

**Track Confirmation** When a radar detects a target, it will send out a series of track confirmation beams to ensure the finding. If the result is positive, the radar switches to track mode and start tracking.

**Missile Acquisition** After a missile is fired by friendly forces, the radar system sends out a series of missile acquisition beams to locate the missile. If the acquisition is positive, the radar switches to missile tracking mode.

## 1.3 An Parallel Implementation Framework

This paper describes an efficient parallel implementation framework for digital signal processing in real-time phased array radar systems. Our implementations combine two types of parallelism – data parallelism and task parallelism. The combination proves to be effective in achieving good performance. We also introduce a layered software architecture to ensures software portability and reusability among different platforms, e.g., ordinary PC clusters and real-time multiprocessing systems.

We conduct experiments to verify the performance and software portability and reusability of our implementations. The implementation platforms include a 8-node PC cluster and a Pentek 4290 ReadyFlow system. From the timing data we identify essential factors in parallel implementations. The characteristics of individual radar function modules are identified and will be used as guidance for future implementations. The communication overhead is recognized as the major obstacle, and the role of granularity is carefully studied and discussed through the experiments.

The rest of the paper is organized as follows. Section 2 will describes the basic building blocks in radar functions. Section 3 describes various issues of the parallel implementations. Section 4 describes the experimental results, and section 5 concludes with possible future works.

# 2 Basic Radar Function Modules

In the previous section we have described the basic operation modes. A task prescribed by an operation mode can be divided into logically independent subtasks. In other words, a radar operation consists of basic radar function modules with clearly defined functionality. See figure 2 for the functional diagram for a special search operation – the *clear search*. Each function module must perform prescribed operation on its input data for a radar to operate correctly. In addition, if we can implement these function modules efficiently, the overall radar operation is efficient. That is, if we can improve the performance of each function module by parallelization, the performance can be improved.

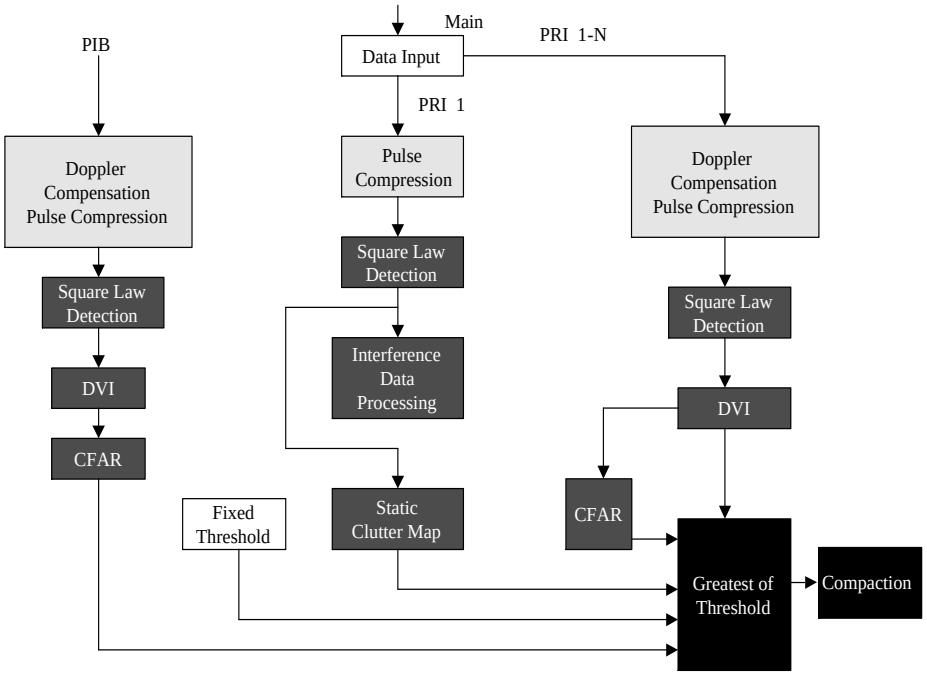



Figure 2: The functional diagram of a clear search.

## 2.1 Data Format

A radar system processes data in the number of *range cells*. That is, the input data consists of an array in which each cell represents the data collected from that range, i.e., the number of range cell multiplied by the distance resolution is the distance the radar can detect targets. A radar system collects data from two channels – $\Sigma$ channel and *pulse interference blanking channel*, or PIB channel. Each channel is further divided into I and Q channels. Each channel consists of range cells, and each cell is represented by an integer. In our simulation we use 1028 range cells with 8 bit integers.

## 2.2 Doppler Compensation

The Doppler compensation adjusts the data received from different Doppler channels, and classified the signals according to their speeds for further processing [17]. The data in different range cells ($f_i(m,k)$) are multiplied by different adjustors for phase compensation, and the results are placed into $DC_i(m,k,l)$. The adjustors are determined by the range cell index and the channel the data is taken from.

$$DC_i(m,k,l) = f_i(m,k) \times e^{jk\Delta t\omega_d(l)} \tag{1}$$

$$\omega_d(l) = 2\pi \cdot \frac{2f}{C} \cdot \Delta v \cdot (l-1) + \omega_{d0} \tag{2}$$

- $m = \Sigma$,PIB
- $k = 1, \ldots,$ NRC (number of range cells)
- $l = 1, \ldots,$ NDC (number of Doppler channels)
- $i = 1, \ldots,$ NPRI (number of pulse repetition interval)
- $c = 3 \times 10^8$m/sec

## 2.3 Pulse Compression

Pulse compression decodes the data from Doppler compensation [17]. The input data from the antenna are encoded to be free from the corruption due to electronic counter measures from the foe. In pulse compression the data from Doppler compensation will be decoded by a *reference code*, one pulse after another. The decoding is to compute the correlation between the reference code and the data ($DC_i(m,k,l)$), and the result is placed into $PC_i(m,k,l)$ [17]. Since the correlation is a very time-consuming, it is vital to parallelize the pulse compression efficiently in order to improve performance.

$$PC_i(m,k,l) = \sum_{x=1}^{L} DC_i(m,x+k,l) \times \text{ReferenceCode}(x) \tag{3}$$

$m$, $k$, $l$ and $i$ are defined as before, and $L$ is the length of the reference code.

## 2.4 Square Law Detection

The square law detection increases the signal-to-noise ratio [17] by computing the square of the magnitude of the complex number with real part $Q_i(m,k,l)$ and imaginary part $I_i(m,k,l)$ from the same range cell. The result is placed into $SLD_i(m,k,l)$.

$$SLD_i(m,k,l) = (I_i(m,k,l))^2 + (Q_i(m,k,l))^2 \tag{4}$$

## 2.5 Digital Video Integration

Digital video integration sums the data for the same pulse repetition interval to increase signal-to-noise ratio [17]. The result is placed into $DVI(m,k,l)$.

$$DVI(m,k,l) = \sum_{i=1}^{NDVI} SLD_i(m,k,l) \tag{5}$$

## 2.6 Constant False Alarm Rate

The constant false alarm rate (CFAR) operation helps reducing background noises [16]. The computation includes two summations over two sample areas on both sides of the target cell, and a maximum is chosen between these two sample areas to ensure that the stronger signal is selected. The CFAR computation consists of three steps.

1. For a given range cell, locate two windows that are CFD range cells away from the given range cell and of size CFS cells. CFD and CFS are parameters used in CFAR operation.

2. Compute $L_w(m,k,l)$ and $R_w(m,k,l)$ for the two windows. They are scaled summation of the data in these two windows.

$$L_w(m,k,l) = \sum_{k-CFD-CFS+1}^{k-CFD} \frac{DVI(m,x,l)}{CFAT} \tag{6}$$

$$R_w(m,k,l) = \sum_{k+CFD+CFS-1}^{k+CFD} \frac{DVI(m,x,l)}{CFAT} \tag{7}$$

3. Compute the scaled maximum between $L_w(m,k,l)$ and $R_w(m,k,l)$.

$$CFAR(m,k,l) = CFTC \times (\max(L_w(m,k,l), R_w(m,k,l)) \tag{8}$$

## 2.7 Static Clutter Map

A clutter map provides background signal signatures so that a radar system can filter out their effects. A radar system uses different clutter maps for different operations. For example, in clear search a radar uses a static clutter map to filter out stationary and low speed targets. On the other hand, for pulse Doppler search the radar will use dynamic clutter map to filter out high speed targets. For the entire region scanned by the radar, a map is maintained to record the background noise, in which each element is called a *clutter map cell*. Note that usually the clutter map cell does not have a one-to-one mapping with the range cells. This map is initialized when the radar system starts operation, after that the clutter map cells need to be constantly updated to maintain their accuracy, i.e., the radar will use the scanned data (range cells) from the beam reflection to update its clutter map.

1. For every clutter map cell, we compute the maximum value in those range cells that are within one range cell from it.
2. Update the clutter map cell as the a linear combination of its old value and the maximum found in the previous step.
3. The clutter map value that will be used in later stages is computed from the values stored in the map.

## 2.8 Greatest of Threshold Detection

The greatest of threshold detection operation identifies possible targets for each range cell. The detection algorithm, described in figure 4, uses a threshold that is equal to the maximum of $CFAR(s,k)$, the static clutter map value, and a fixed threshold.

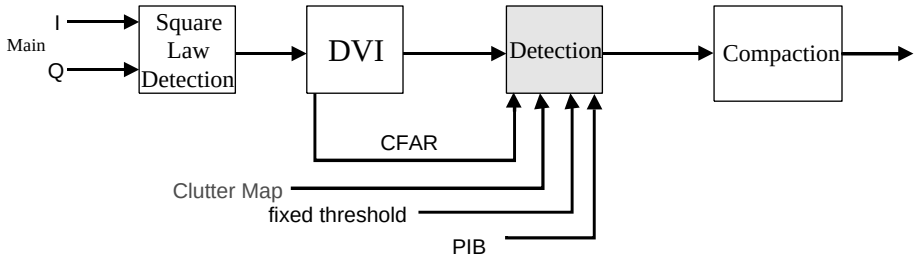



Figure 3: Functional diagram of GTD.

if $(DVI(PIB, k) > DVI(\Sigma, k))$ then
  return false
else if (DVI$(S, k)$ >threshold) then
    return true
  else return false

Figure 4: The detection algorithm of GTD.

### 2.9 Compaction

A phased array radar system can search and track targets in several regions simultaneously. When the system operates for a period of time and would like to report the possible targets, the results have to be compacted. The compaction operation will prepare the data and report the channel and range cells that have the possible targets.

## 3 DSP Parallelization

This section describes our parallelization strategy. First we introduce our parallelization framework and the software architecture. Then we introduce the implementation details of each operation described in section 2.

### 3.1 Hierarchical Software Architecture

A high performance radar system demands a real-time environment for predictable time limits in both computation and data communication, as a result it requires parallel processing to provide scalable aggregated raw computing power. In order to facilitate the portability among different real-time system environments and data transmission application interface (API), and maintain the software reusability, we use a layered approach in designing our software architecture, which is illustrated in figure 5.

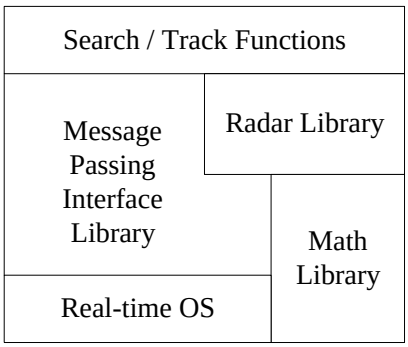



Figure 5: The hierarchical software architecture.

**Mathematical Library** The mathematical library provides bottom-level computing building blocks for our framework. The library includes all the basic vector operations, where the data in the vectors can be a real or a complex number.

**Radar Library** The radar library provides all the basic functions for digital signal processing. All the functions described in section 2 are implemented in this library.

**Message Passing Interface Library** To increase portability and software reusability, we design our data communication library based on the popular Message Passing Interface (MPI) [9, 10]. This approach not only ensures that we can reuse most of the communication codes, which takes up the bulk of implementing a parallel program, but also guarantees that we can test our implementation on general purpose computing platform other than real-time systems, including networks of workstations (NOW), PC clusters. or even shared memory parallel computers.

**Search/Track Functions** The search/track functions define the top level API for a radar system. These functions use radar functions for computation, and message passing interface library for data communication.

## 3.2 Parallelization Strategy

We describe two parallelization strategies – *data parallelism* [9, 15] and *task parallelism* [15] in this section.

### 3.2.1 Data Parallelism

The idea of data parallelism is to apply the same operation on a batch of data to achieve parallelism. The data is partitioned into different *computation domains*, and according to the relative speed of different processors each processor computes the answer for each region assigned to it. Since the data volume per processor is reduced, the overall execution time is also reduced.

### 3.2.2 Task Parallelism

The task parallelism divides the entire computation into subtasks, which by themselves are independent logical units of computation, and assigns each task to a processor for execution. Although the tasks are logically separated, but they might be temporally dependent, i.e., one task may start only after another has completed. In such a case task synchronization is necessary. The parallelism is achieved by having more than one processor working on temporally independent subtasks.

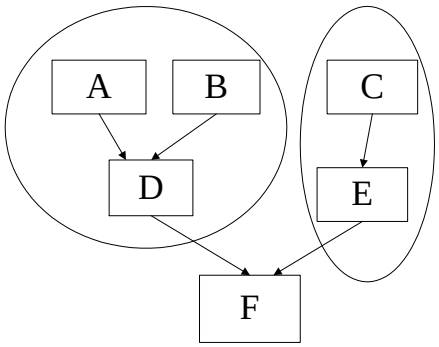



Figure 6: An example of task parallelism among two processors.

A simple example of task parallelism is illustrated in figure 6. Each node represents a computation task and each edge represents temporal dependency. In this example we assume that there are two processors and computation $A$, $B$ and $D$ are assigned to one processor, and $C$ and $E$ are assigned to the other. If the times for the two sets of the computation are roughly the same, the entire computation can be almost twice as fast.

### 3.3 Evaluation

We observed that there are two key factors in achieving high parallel performance – the network performance and the granularity, i.e., the amount of data assigned to a processor [15]. For any parallel implementation inter-processor communication is inevitable, therefore the efficiency of network communication is essential, especially when the data volume is large. The communication can happen in various places. For example, to ensure that each processor has the correct data to start with, the data from the previous stage need to be transfer to the processor in charge of the next stage of the computation.

Data parallelism should be used in situations where the data granularity is sufficiently large so that the performance gained from data parallelism is not taken away by the extra communication overheads. That is, frequent data transmission of small volume will not be efficient. Instead data should be packed into large volume for transmission.

Data parallelism should be used when the logically independent subtask can be clearly identified from the computation. That is, the computation consists of computational units that can be isolated and implemented individually. When the granularity is too small to achieve efficient data parallelism, it is a good idea to use task parallelism since the granularity can increase, and it is more likely that the parallelism is better achieved by having multiple processor running different subtasks simultaneously, rather than enforcing the data parallelism, which could result in inefficient small granularity.

### 3.4 Implementation Strategy

We use different parallel strategy at different layer of our software hierarchy.

**Functional Modules** We apply data parallelism on basic radar function modules. The current system we are using have four DSP processors, therefore we implement the basic radar functions with data parallelism, in order to fully explore the computing power. When the computation in these functional modules is heavy, or the input data is sufficiently large, we adopt the data parallelism for better parallel efficiency.

**Radar Operation Modes** We apply task parallelism to resolve the data dependency among the functional modules that constitute a radar operation mode, e.g., clear search in figure 2. The rationale is that these functional units are clearly logically independent and all we need to explore is the possibility of running multiple tasks at the same time.

### 3.5 Implementation Details

This section describes the implementation details of all functional modules. The main focus is whether the computation requires any inter-processor communication.

**Pulse Compression** The pulse compression computes correlation between the range cell data and the reference code. Since the range cell window may overlap with the computation domain of different processors, data communication is necessary. Our implementation will collect the boundary data first then starts calculating according to equation 3.

**Constant False Alarm Rate** The constant false alarm rate (CFAR) operation requires the computation of two windows from the current range cell. The windows may be in the domains of other processors, hence inter-processor communication is required.

**Static Clutter Map** The static clutter map requires constant updating. Recall that in the first step of the update algorithm neighboring cells have to be referenced, and inter-processor communication is required.

Figure 7 summarizes the requirement for inter-processor communication for all basic radar functional modules.

| Functions | | Is Data Scattering needed between processors before executing the function? |
|---|---|---|
| Doppler Compensation | | No |
| Pulse Compression | | Yes |
| Square Law Detection | | No |
| Digital Video Integration | | No |
| Constant False Alarm Rate | | Yes |
| Static Clutter Map | Step 1 | Yes |
| | Step 2 | No |
| | Step 3 | No |
| Greatest of Threshold Detection | | No |
| Compaction | | No |

Figure 7: A summary table for inter-processor communication.

# 4 Experimental Results

This section describes the experimental results from our implementations.

## 4.1 Experiment Environment

The experiments are conducted on two different platforms – a PC cluster and a Pentek 4200 ReadyFlow system. Further details of these tow implementation can be found in [7] and [6].

## 4.2 PC cluster

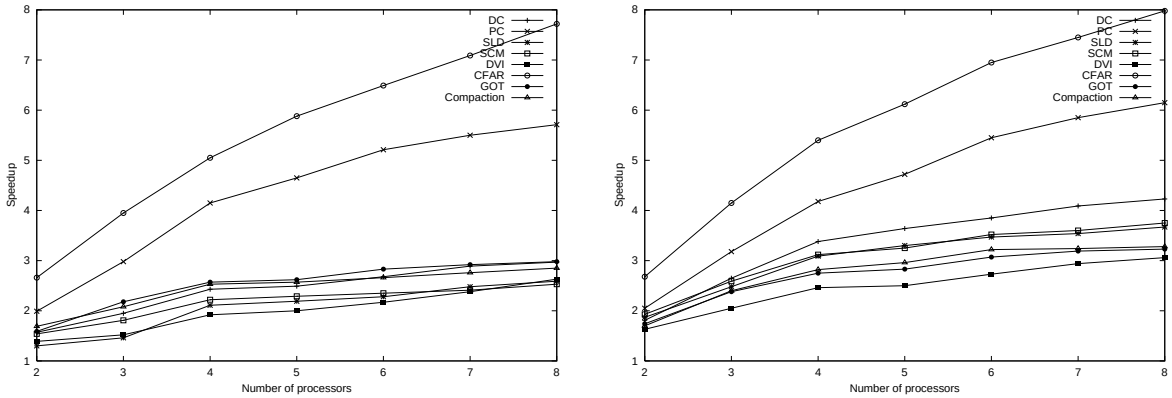
**Processor:** 8 Intel Pentium-Pro PCs, each has 256 Mega bytes of memory.

**Connection:** Fast Ethernet connects the 8 processors.

**Operating System:** All PCs use FreeBSD 3.4 operating system.

**Message Passing Library:** We use MPICH version 1.1.2 as the communication library.

**Compiler:** The program is compiled with GCC 2.95.2.




1024 range cells 3072 range cells

Figure 8: The speedups of different radar function modules with 1024 and 3072 range cells.

We summarize our observations as follows.

- **Our parallel implementation of DSP function dose improve performance but for some cases, the scalability is limited.**

  Observer figure 8, which illustrates the speedups versus the number of processors for 1024 range cells. We notice that we can improve the performance of all of the radar function modules by adding processors. However, when we add processors the performance gain will not scale accordingly. This is because the granularity decreases and the amount of communication increases. Therefore it is crucial to determine the most efficient number of processor for a given task – not only because this could spare more processors for other useful work, but also that more processors do not improve performance significantly.

  We observe the same phenomenon when the number of range cells is 3072, as in figure 8.

- **The network communication delay is the major overhead.**

  Figure 9 illustrates the effect of communication overheads. If we simply performance pulse compression, the speedup scales well. However, when we add communication routine to ensure that each processor has all the data it requires, the speedup deteriorates dramatically.

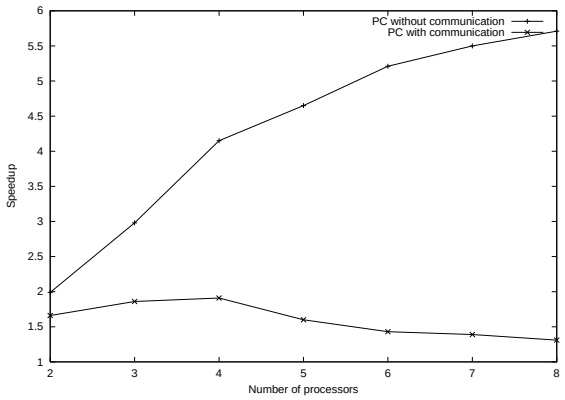



Figure 9: The pulse compression speedup comparison between with and without communication overheads.

- **A sufficiently large granularity is essential.**

  Consider figure 10 and we observe that in both PC and CFAR computation the speedup improves when the granularity increases. For example, in figure 10 we observe that the speedup is 1.91 when 4 processors are used for 1024 range cells. When the problem size increases to 2048 the speedup increases to 2.35. Similar results can be observed in figure 10 for CFAR.

  The amount of computation can also affect the choice of granularity. Observe figure 12 and we notice that despite with the same granularity, the PC has better speedup than CFAR, which is better than SCM. The reason is that PC has the largest amount of computation among them, hence the highest computation-to-communication ratio (figure 11). As a result it could maintain reasonable speedup even for small granularity.

- **Each radar function module has different computation/communication characteristics.**

  Different radar function modules can be characterized differently. For example, PC can achieve very good performance with 4 processors from figure 10. On the other hand, CFAR must use large problem size to achieve reasonable performance when communication overhead is included.

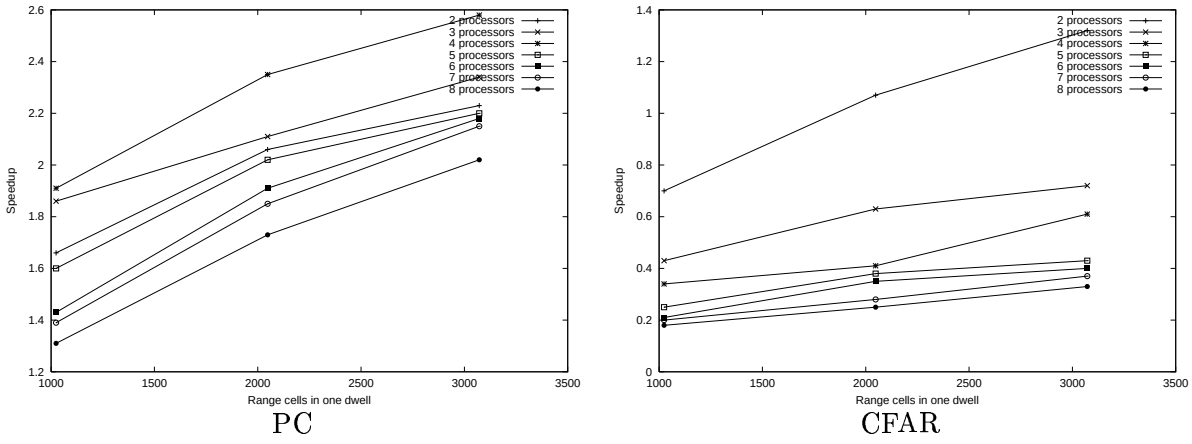



Figure 10: Speedups of CFAR and PC operations under different problem sizes.

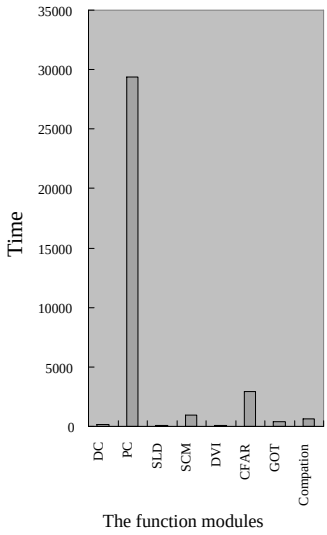



Figure 11: The total amount of time in each radar function module.

### 4.3 Pentek 4290 ReadyFlow System

We also conduct experiments on a Pentek 4290 ReadyFlow system. The system can be characterized as follows.

**Processor:** 4 TMS 320C6000 DSP chips running at 200MHz.

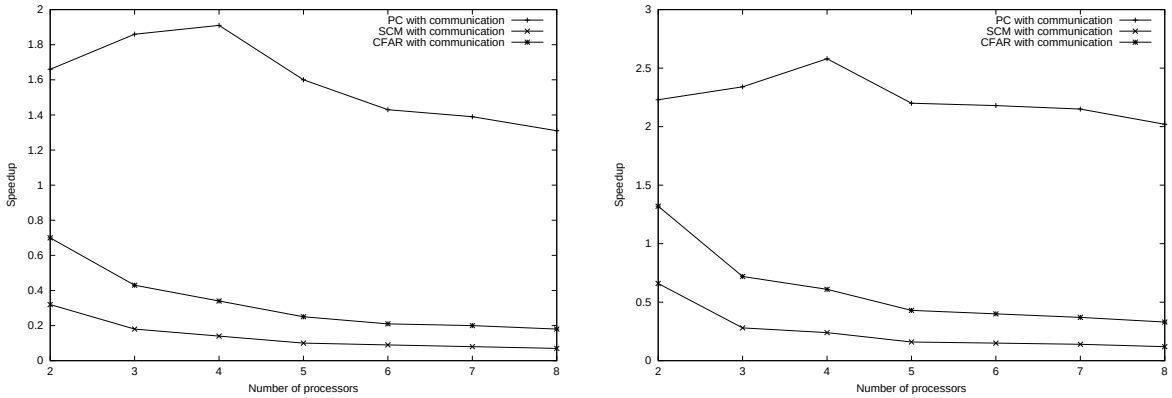
**Memory:** 16M SDRAM.

**Connection:** 64K bidirectional FIFO for interprocessor communication.

**Operating System:** Virtuoso 4.1.

**Message Passing Library:** The code is programming in Virtual Single processor (VSP) model.

The main obstacle in running our code in Pentek 4290 ReadyFlow is that it does not support MPI API. As a result we implemented an MPI interface on top of Virtuoso operating system functions. This wrapper makes heavy use of Virtuoso system objects, including semaphore, resource, and mailbox. The details of this implementation can be found in [6]. The MPI functions implemented are list below.




1024 range cells 3072 range cells

Figure 12: Speedup comparison among PC, SCM, and CFAR when the input size is 1024 and 3072.

**MPI_Send:** A processor sends a message to another processor.

**MPI_Recv:** A processor receives a message from another processor.

**MPI_Bcast:** A processor broadcasts a message to all the other processors.

**MPI_Gather:** A processor collects data from all of the other processors.

**MPI_Scatter:** A processor distributes different data blocks to all of the other processors.

**MPI_Barrier:** A barrier synchronization for all processors.

The timing results are summarized in Table 1.

| Function | DC | PC | SLD | DVI | CFAR |
|---|---|---|---|---|---|
| Time (sec) | 0.023 | 0.041 | 0.028 | 0.033 | 0.036 |
| Function | SCM | GTD | compaction | data transmission | clear search |
| Time (sec) | 0.071 | 0.041 | 0.036 | 0.036 | 2.625 |

Table 1: Timing results on the Pentek 4290 ReadyFlow system.

# 5 Conclusion

This paper describes an efficient implementation framework for parallel digital signal processing in real-time multiprocessing environments. The combination of data parallelism and task parallelism proves to be effective in achieving good performance. The layered software architecture ensures portability and software reusability among different platforms.

We conduct experiments to verify the implementations, in both performance and software portability and reusability. The implementation platforms include a 8-node PC cluster and a Pentek 4290 ReadyFlow system. From the timing data we identify essential factors in parallel implementations. The characteristics of individual radar function modules are identified and are used as guidance for future implementations. The communication overhead is recognized as the major obstacle, and the role of granularity is carefully studied and understood through the experiments.

We are still interested in improving the performance of our parallel implementation. One possibility is to use assembly, or use libraries that are built with assembly to increase

performance. Despite that this will hinder portability, we argue that as long as the high-level library, such as the search/track functions are kept intact, the investment is worthwhile. The best candidate to use assembly implementation is the low level routines, like those in the mathematical library. We believe this approach will strike a nice balance between the efficiency and portability.


### Acknowledge

The authors thank the generous support of The Chung Shan Institute of Science and Technology. We also thank Dr. Cheng Chang and Dr. Tai-Chong Wang for helpful discussion.


## References


[1] A. Barbato and P. Giustiniani. An improved scheduling algorithm for a naval phased array radar. In *ALENIA Defence System*, Italy.

[2] R. A. Baugh. *Computer Control of Modern Radars*. RCA M&SR-Moorestown Library, 1973.

[3] C. Chang and C. Chung. A real-time scheduler in a programmable radar signal processor. 1997.

[4] C. Chang and T.-C. Wang. Use object-oriented paradigm to design a programmable radar digital signal processor. In *Third Workshop on Object-Oriented Technology and Applications (OOTA' 97)*, September 1997.

[5] C. Chang and T.-C. Wang. Component-oriented digital signal processor. pages 1263–1269, June 1999.

[6] S. Chen. Parallel processing of digital signals in phased array radars on virtuoso, 2001. master thesis, National Chung Cheng University.

[7] Y. Chu. Parallel digital signals processing for phased array radars, 2000. master thesis, National Chung Cheng University.

[8] R. Filippi and S. Pardini. An example of resources management in a multifunctional rotating phased array radar. In *Real-Time Management of Adaptive Radar Systems, IEE Colloquium*, pages 2/1–2/3, 1990.

[9] I. Foster. *Designing and Building Parallel Software Engineering*, chapter 1, pages 17–18. Addison-Wesley Publishing Company, Inc., 1995.

[10] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi: a message passing interface standard. Technical report, Argonne National Laboratory and Mississippi State University.

[11] A.G. Huizing and A.A.F. Bloemen. An efficient scheduling algorithm for a multifunction radar. In *IEEE International Radar Conference*, pages 359–364, 1996.

[12] G. V. Keuk and S. S. Blackman. On phased-array radar tracking and parameter control. In *IEEE Transactions*, pages 186–194, Jan 1993.

[13] R.L. Nevin and F.W. Schatz. An/apg-67 multimode radar development. In *IEEE International Radar Conference*, pages 1–8, 1985.

[14] The America Ministry of National Defence. Rapid prototyping of application specific signal processors (rassp). http://eto.sysplan.com/ETO/RASSP.

[15] P. S. Pacheco. *Parallel Programming with MPI*, chapter 5, 10, pages 82–83,217–218. Morgan Kaufmann Publishers, Inc., 1997.

[16] M. I. Skolnik. *Introduction to Radar Systems*, chapter 10, pages 392–393. McGraw-Hill Book Company,Inc., 2nd edition, 1980.

[17] M. I. Skolnik. *Radar Handbook*, chapter 10, pages 7–8. McGraw-Hill Book Company,Inc., 2nd edition, 1990.

[18] D. Stromberg and P. Grahn. Scheduling of tasks in phased array radar. In *IEEE International Radar Conference*, pages 318–321, 1996.