

2. Y. Chung, and V. K. Prasanna, "Image Feature Extraction on Coarse-grain Parallel Machines," submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, 1996.
3. J. C. Curlander and R. N. McDonough, Synthetic Aperture Radar-Systems and Signal Processing, Wiley, 1991.
4. T. Einstein, "Realtime Synthetic Aperture Radar Processing on the RACE(R) Multicomputer," [http://www.mc.com/ap3/ap3\\_01.html](http://www.mc.com/ap3/ap3_01.html), 1996.
5. I. Foster, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison-Wesley Co., 1995.
6. R. A. Games, "Benchmarking Methodology for Real-Time Embedded Scalable High Performance Computing," MITRE Technical Report MTR 96B0000010, March 1996.
7. W. Gropp, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, MIT Press, 1994.
8. Intel, "At the Jet Propulsion Laboratory: Spaceborne Radar Image Processing on the Paragon Supercomputer," <http://www.ssd.intel.com/success/sar.html>, 1996.
9. Y. W. Lim, P. B. Bhat, and V. K. Prasanna, "Efficient Data Remapping Algorithms for Embedded Signal Processing Applications", 10th HPCS '96, Ottawa, Canada, 1996.
10. P. G. Meisl and M. R. Ito, "Parallel Synthetic Aperture Radar Processing on Workstation Networks," International Parallel Processing Symposium, 1996.
11. Sandia National Laboratories, "Synthetic Aperture Radar", <http://www.sandia.gov/RADAR/sar.html>, 1996.
12. R. Schotter, "Digital Realtime SAR Processor for C- and X-band Applications," Proceedings of IGRASS '86, Zurich, September 8-11, 1986.
13. S. N. Madsen and J. Dall, "Processing of the Danish C-band SAR Data," Proceedings of IGARSS '90, Washington, D.C., May 20-24, 1990.
14. C.-L. Wang, "High Performance Computing for Vision on Distributed Memory Machines," Ph.D. Thesis, University of Southern California, August 1995.
15. C.-L. Wang, P. B. Bhat, and V. K. Prasanna, "High-Performance Computing for Vision," Proceedings of the IEEE, Vol. 84, No.7, July, pp. 931-946, 1996.
16. C. R. Yerkes, and E. Webster, "Implementation of  $\omega$ -k synthetic aperture radar imaging algorithm on a massively parallel supercomputer," SPIE Proceedings Vol. 2230, pp. 171-178, Orlando, FL, April 4-8, 1994.

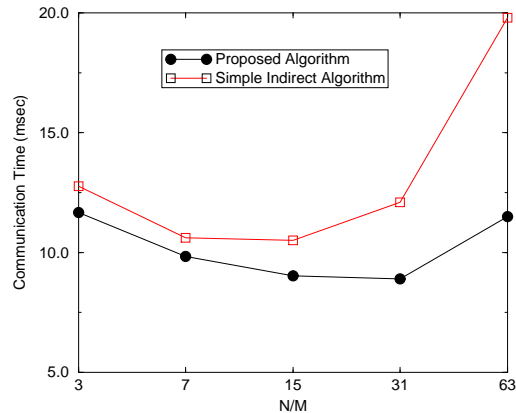


Figure 9: Comparison of the simple indirect algorithm and the proposed algorithm

application, the application can be divided into several stages and the computational requirements of the stages vary. Thus, in each stage, different number of processors are used. The  $M$ -to- $N$  communication algorithm is used between any two consecutive stages. Such processing arises in applications such as ATR, SAR, STAP, and Sonar.

### Acknowledgements

This research work was supported in part by NSF under contract CCR-9317301 and in part by DoD High Performance Computing Modernization Office, and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-2-0016. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The experimental results were obtained at the Maui High Performance Computing Center (MHPCC) and at the San Diego Supercomputer Center (SDSC).

### References

1. G. Aloisio and M.A. Bochicchio, "The Use of PVM with Workstation Clusters for Distributed SAR Processing," Proceedings of HPCN Europe '95, Milan, Italy, May 3-5 1995.

the second group, respectively. The previous algorithm needed  $MN$  communication steps. By using the proposed algorithms, the number of processors required to perform  $1K \times 1K$  FFT was reduced by up to 50%. The maximum data size that could be processed was also increased by using our algorithm.

The proposed  $M$ -to- $N$  communication algorithms can be used in applications that have heterogeneous computational characteristics. In such an

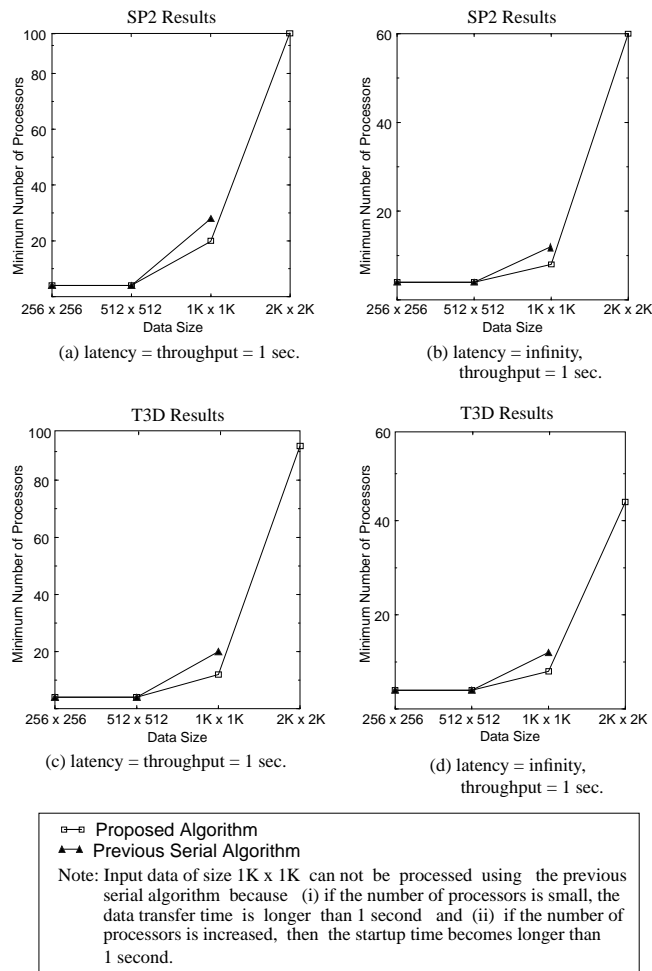


Figure 8: Minimum number of processors needed for FFT

shows that the proposed communication algorithm reduces the communication time significantly. The reduced communication time is mainly due to the reduction in the startup time.

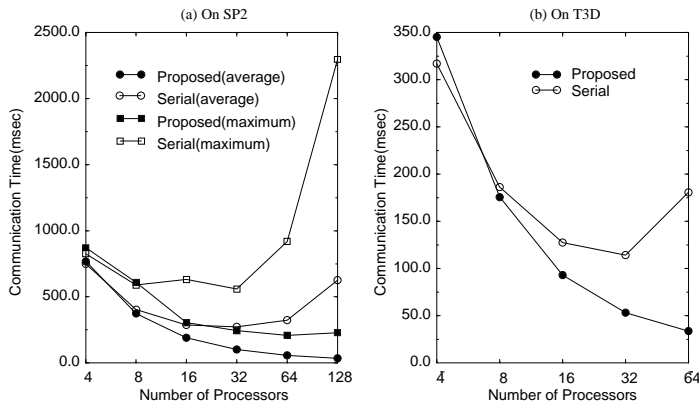


Figure 7: Communication time for processing  $1K \times 1K$  data

In addition to the SAR implementation, 2D FFT was also implemented. Each data sample was 4 bytes in this implementation. The results are shown in Figure 8.

A comparison of the performance of the previous and the proposed algorithms for  $M$ -to- $N$  communication is shown. In Figure 9, simple indirect algorithm and the proposed algorithm are compared and in Table 2, the `MPI_Alltoall` and `MPI_Scatter` primitives and the proposed algorithm are compared.

Table 2: Comparison of the performance of the MPI primitives and the proposed algorithm on SP2

Data size	Proposed algorithm	<code>MPI_Alltoall</code>	<code>MPI_Scatter</code>
4 Bytes	0.43 msec	0.75 msec	1.21 msec
4 KBytes	23.1 msec	44.5 msec	78.8 msec

## 6 Conclusion

In this paper, we implemented the SAR processing based on MITRE benchmark guidelines on the SP2 and the T3D. The implementation used software task pipeline which consisted of two groups of processors. Using our communication algorithms, the number of communication steps was reduced to as low as  $\lg(M + N)$ , where  $M$  and  $N$  are the number of processors in the first and

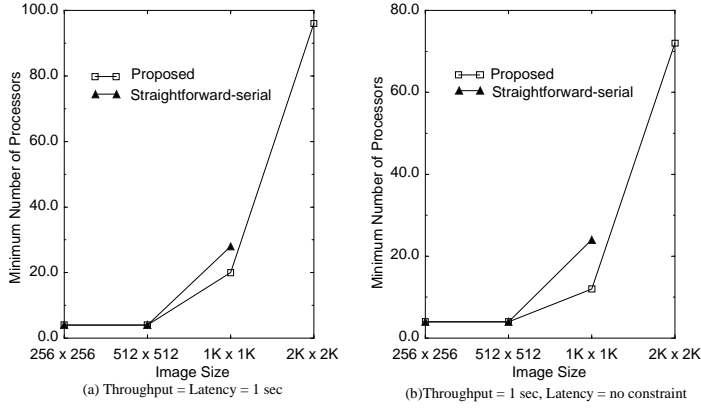


Figure 5: Minimum number of processors needed for SAR processing on SP2

algorithm is the increased maximum data size that can be processed to meet the specified real time constraint. For example, when the data size is  $2K \times 2K$ , it was not possible to process it in 1 sec using the previous straightforward algorithm on the SP2 (see Figure 5 (a)). Using the earlier straightforward approach, when the number of processors is small, the data transfer time is longer than 1 sec, and when the number of processors is large, the total startup time is longer than 1 sec. Thus, in any case, the data could not be processed.

To analyze the communication time, it was measured separately. Figure 7

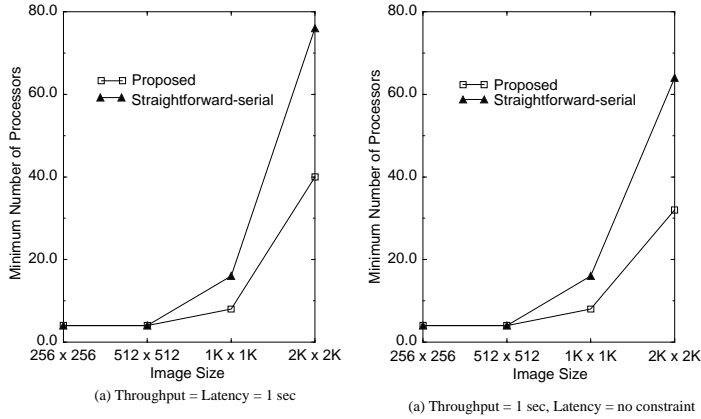


Figure 6: Minimum number of processors needed for SAR processing on T3D

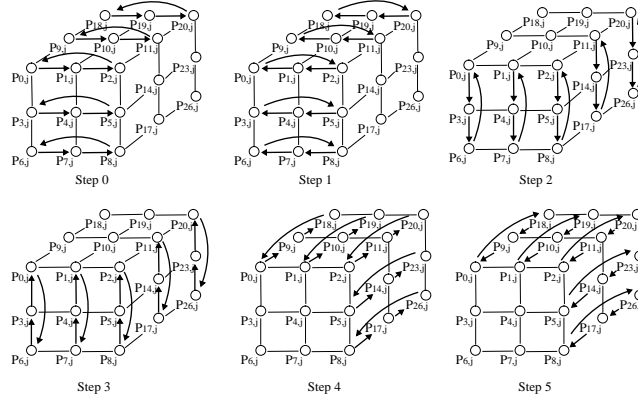


Figure 4: An example of *All-to-All* communication ( $M = 27, k=3,$  and  $n=3$ )

The experiments were performed on the IBM SP2 at the Maui High Performance Computing Center (MHPC) and on the Cray T3D at the San Diego Supercomputer Center (SDSC). The communication routines were implemented using the Message Passing Interface (MPI) standard. The image sizes used in the experiments were  $8K(\text{azimuth}) \times 1K(\text{range})$  and  $8K \times 8K$ . Since one byte was used to represent each pixel, the total data transferred between the processors was 8MB and 64MB, respectively. The time for performing the FDC and the time for communication were measured. Based on the MITRE benchmark evaluation guidelines<sup>6</sup>, the programs were executed 1000 times and the maximum execution time was used. The MITRE benchmark evaluation guidelines were designed to provide a methodology to assess the performance of HPC platforms for real-time embedded applications<sup>6</sup>. These use the *design-to-specification* methodology. In this method, both the timing and the functional specifications are given. The minimum size of a machine that can process a given problem is determined. Our experiments were performed based on the timing specification for real-time FFT. The specification consists of two cases as follows.

Case 1: Period = latency = 1 second.

Case 2: Period = 1 second, no restriction on latency.

The period is the time interval between successive input matrices to the machine. The latency is the elapsed time between the data input to the machine and the corresponding output.

Figure 5 and Figure 6 show the minimum number of processors needed to perform SAR processing for the above two cases. The results show that the proposed algorithm reduces the required number of processors by up to 50%. The reduction is possible due to the reduced communication time using the proposed algorithm. Another advantage of SAR processing using the proposed

$$T_{comm} = \lg(N/M + 1)T_s + B(M + N)\tau_d\left(\frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{N/M + 1}\right) \quad (1)$$

$$= \lg(N/M + 1)T_s + BN\tau_d. \quad (2)$$

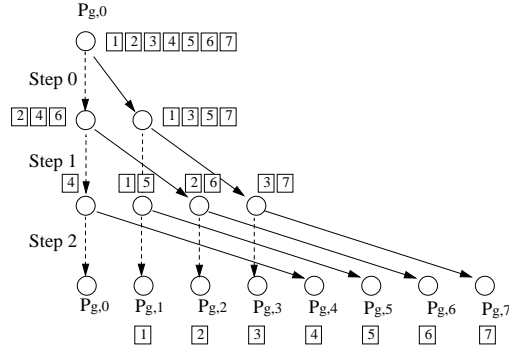


Figure 3: An illustration of the scheduling of the first stage in the  $i$ -th ( $0 \leq i \leq M - 1$ ) subgroup when  $N/M = 7$

In the second stage, an *all-to-all* communication is performed. A destination processor  $P_{i,j}$  ( $i = 0, 1, \dots, M - 1, j = 1, \dots, N/M$ ) needs to exchange data among  $P_{k,j}$  ( $0 \leq k \leq M - 1, k \neq j$ ) using an *all-to-all* communication.

A general algorithm for the second stage is described here. The processors are arranged in a  $k$ -ary  $n$ -cube,  $2 \leq k \leq M$  and  $n = \lceil \lg_k M \rceil$ . The number of communication steps and the total communication time is determined by setting  $k$ . The communication schedule is performed in  $n(k-1)$  steps as follows: In the  $s$ -th ( $s = 0, 1, \dots, n(k-1) - 1$ ) step, a processor  $P_{i,j}$  ( $0 \leq i \leq M - 1, 1 \leq j \leq N/M$ ) sends data to  $P_{(i+k^m(s \bmod (k-1))) \bmod k^{m+1} + \lfloor i/k^{m+1} \rfloor k^{m+1}, j}$ . The amount of data transferred by each processor is  $MB/k$ . Thus, the total communication time is  $n(k-1)T_s + (Mn(k-1)B/k)\tau_d$ . An example of the schedule is shown in Figure 4, where  $k = 3$  and  $n = 3$ .

## 5 Experimental Results

The SAR signal processing was implemented using a FFT library and the proposed communication algorithms. In our implementation, the processors are partitioned into two groups: one group consisting of  $M$  processors performs FDC in the row dimension, and the other consisting of  $N$  processors performs FDC in the column dimension. The proposed algorithm using  $n=2$  was used for corner turn between the two groups. For the sake of comparison, the SAR signal processing was also implemented using a straightforward serial communication algorithm for corner turn.

example of this case is performing a permutation of data. During  $T_s + m\tau_d$  time, the sending processor is busy handling the communication. However, if there is a communication coprocessor and nonblocking send is used, the sending processor has to spend only the startup time to activate the communication coprocessor. After this initial startup time, the processor is free and can continue its computations, while the actual transfer of messages is handled by the communication coprocessor. Such a model has been used for algorithm design on distributed memory parallel systems. It has been called the General Purpose Distributed Memory(GDM) model. Similar models have been used by others<sup>2,5,9</sup>.

Our proposed general algorithm consists of two stages. Without loss of generality, we assume that  $M < N$ . If  $M > N$ , the roles of senders and receivers in the following algorithms are reversed. If  $M = N$ , it becomes *All-to-All* communication. In the first stage, we assume that  $N/M + 1 = 2^q$  for simplicity, where  $q$  is a positive integer. The  $N$  destination processors are partitioned into  $M$  subgroups. Let the source processor in the  $g$ -th subgroup ( $g=0,1, \dots, M-1$ ) be denoted as  $P_{g,0}$ . Let the  $i$ -th destination processor in the  $g$ -th group be denoted as  $P_{g,i}$  ( $i = 1, 2, \dots, N/M$ ). Let  $C(i, j, k)$  ( $0 \leq i \leq M - 1, 0 \leq j \leq M - 1, 1 \leq k \leq N/M$ ) be the data block from  $P_{i,0}$  to be sent to  $P_{j,k}$ . Let  $B = D/MN$ , where  $D$  is the total amount of data to be communicated.

Initially, the data is rearranged which is performed only once at the source processors to avoid data rearrangement during the communication. After the rearrangement, the data in the source processor  $P_{i,0}$  is redistributed to destination processors  $P_{i,k}$  ( $1 \leq k \leq N/M$ ) in  $\lg(N/M + 1)$  steps as follows (see Figure 3). In the  $s$ -th step,  $s = 0, 1, \dots, \lg(N/M + 1) - 1$ , there are  $2^s$  processors that can initiate the send operation in each subgroup. In the  $i$ -th subgroup,  $2^s$  processors  $P_{i,j}$  ( $j = 0, 1, \dots, 2^s - 1$ ) send data  $C(i, g, j + 2^{s+1}(1 + 2k))$  ( $g = 0, 1, \dots, M - 1, k = 0, 1, \dots, (N/M + 1)/2^{s+1} - 1$ ) to  $P_{i,j+2^{s+1}}$ . In Figure 3, an example communication schedule and the corresponding data movement are shown. In this figure, the number within a box represents the destination processor of a data block. After the  $s$ -th step, the number of processors that would have received the data is  $2^{s+1}$ . These processors can initiate a send operation in the next step. After  $\lg(N/M + 1)$  steps, in each subgroup, the data in the source processor will be distributed to all the destination processors in the group.

The communication time of the  $s$ -th step is  $T_s + B(N/M + 1)\tau_d/(2^{s+1})$  using the GDM model. Therefore, the total communication time  $T_{comm}$  for the first stage is:

be performed sequentially. A comparison of the performance of these methods and our proposed algorithm is presented in Section 5.

Another algorithm for  $M$ -to- $N$  communication is a simple indirect algorithm. In this algorithm,  $N$  destination processors are partitioned into  $\lceil N/M \rceil$  groups. In the  $s^{th}$  step,  $s = 0, 1, \dots, \lceil N/M \rceil - 1$ ,  $M$  source processors send data to  $M$  destination processors in the  $s^{th}$  group. After the destination processors receive the data, the processors perform an *all-to-all* communication among the processors in their group. The drawback of this algorithm is that the available network and processors are not fully used because only a few groups are involved in the communication. A comparison of the performance of this algorithm and our proposed algorithm is shown in Section 5.

Another method is a straightforward serial communication method reported in<sup>4</sup>. During the  $s^{th}$  step ( $s = 0, \dots, MN - 1$ ), the  $(\lceil s/M \rceil)^{th}$  source processor sends its  $(s \bmod M)^{th}$  block to the  $(s \bmod N)^{th}$  destination processor. In this algorithm, the number of communication steps is  $MN$ . This method is easy to program and there is no node and link contention. It uses very little available network bandwidth since only one path from a processor to another is used during each communication step. This results in large communication time.

#### 4 Proposed Communication Algorithms

In this section, we first introduce the General-purpose Distributed Memory (GDM) model<sup>15</sup>. Then, we propose a general efficient algorithm for  $M$ -to- $N$  communication using this model. Using our communication algorithms, the required number of processors can be reduced by as much as 50% in a SAR benchmark implementation. Also, our algorithm is suitable for using a software task pipeline which is an efficient architecture to obtain high throughput performance for signal processing applications.

An important component of our approach is the use of a realistic computational model of HPC platforms. It allows us to analyze the computation and communication overheads. The communication operation in such a system can be modeled using two parameters. One is the *startup time* which occurs in every message transfer. This includes the software and the communication protocol processing overheads. The other is the *unit transmission time* which is the cost of transferring a message of unit length through the network. Let  $T_s$  denote the startup time and  $\tau_d$  denote the unit data transmission time. Then, the time for sending a message containing  $m$  units of data from a processor to another is modeled as  $T_s + m\tau_d$ . This corresponds to the case when “blocking” send is used and no special hardware for communication is available. An

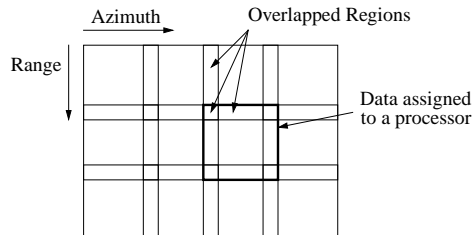


Figure 2: Overlapped block distribution of data for  $N=12$ .

limited scalability. Since the size of a block must be larger than the size of the filter, the maximum number of processors is limited by the ratio of the data size to the filter size.

In the second case<sup>16</sup>, there are two stages of computation. After each processor performs FDC in range dimension in the first stage, all of the data is redistributed among all the processors to perform the FDC in azimuth dimension. After the first stage, the data in a processor is partitioned into  $M$  blocks, where  $M$  is the number of processors. During step  $s$ ,  $0 \leq s \leq M - 1$ , processor  $j$  sends a data block to processor  $((j \oplus s) \bmod M)$ , where  $\oplus$  represents the binary exclusive-OR operation. The number of communication steps is  $M - 1$ . A completely connected topology is assumed. This algorithm gives good performance for *all-to-all* communication. However, the algorithm is not suitable for communication between adjacent stages in a software task pipeline in which adjacent stages have different number of processors.

In the third case, a software task pipeline is used. The processors are partitioned into two groups. One group consisting of  $M$  processors performs the FDC in range dimension, and the other consisting of  $N$  processors performs the FDC in column dimension. Since computational requirement for each group is different, the number of processors assigned to each group is different. Thus, in general,  $M \neq N$ . The data in a processor in the first group is partitioned into  $N$  blocks, and each data block is sent to a different processor in the second group. Therefore, *M-to-N* communication is required between the two stages.

A simple method for *M-to-N* communication is to use the *all-to-all* communication primitive on  $M + N$  nodes supplied by an MPI library<sup>7</sup>. This method is easy to program. However, the *all-to-all* primitive requires each of the processors to send the same amount of data to each destination processor. Thus, the destination processors need to send dummy data which results in low utilization of bandwidth. Another method is to use MPI Scatter primitive.  $M$  scatter groups are constructed; in the  $i^{\text{th}}$  group, the  $i^{\text{th}}$  source processor and  $N$  destination processors are included. Then, in each group, a scatter operation is performed. The drawback of method is that all the scatter operations must

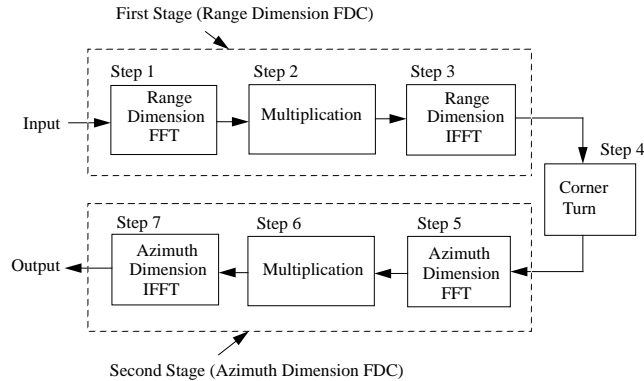


Figure 1: Block diagram of SAR signal processing  
 Table 1: Typical parameters of SAR signal processing

Parameter	Typical Value
Number of range samples	6840
Number of azimuth samples	4096
Block size	$8192 \times 8192$
Data rate	11.27 M samples/sec
Computational requirement(azimuth)	3.28GFLOPS
Computational requirement(range)	1.83GFLOPS
Total computational requirement	5.11GFLOPS

The number of samples in range dimension is fixed by parameters such as the pulse dispersion angle and the pulse incident angle to the ground. However, the number of samples in the azimuth dimension depends on the scanning time. Thus, the number of samples in the azimuth dimension can be arbitrarily long.

### 3 Previous Research

SAR processing consists of three main stages; range dimension FDC, corner turn, and azimuth dimension FDC. Based on how the input data is partitioned, parallel implementation of SAR processing can be classified into three categories. Each of them is briefly explained in the following.

In the first case, the data is partitioned into  $N$  overlapped blocks as shown in Figure 2. The boundary data of a block overlaps with up to four adjacent blocks. Each block is assigned to a processor which performs FDC using given filter coefficients. The overlapped region between adjacent blocks is sufficiently large so that no communication is required during the FDC computation. This method is efficient in terms of communication because it minimizes the communication cost during the computation. A disadvantage of this algorithm is

nication is also shown. In this algorithm, the number of communication steps is  $\lceil \lg(N/M + 1) \rceil + n(k - 1)$ , where  $k \geq 2$ ,  $n = \lceil \lg_k M \rceil$ . As an example, if  $M = 16$  and  $N = 48$ , the number of steps in the proposed algorithm is 6 whereas a serial communication algorithm takes 768 steps.

Our implementation was performed on the IBM SP2 and the Cray T3D using C and the Message Passing Interface (MPI) standard. The processing times were measured using the MITRE real-time benchmark evaluation guidelines<sup>6</sup>. Our implementation results shows that the required number of processors for a SAR benchmark was reduced by 50%. The reduction was possible due to the reduced communication time using the proposed algorithm. Also, the maximum data size that can be processed under a given throughput and latency constraints was increased.

The organization of this paper is as follows: The SAR model and its computational requirements are described in Section 2. Previous work in parallelizing SAR is discussed in Section 3. Section 4 describes the proposed communication algorithms. The analyses of the previous and the proposed algorithms are also presented here. Experimental results are presented in Section 5. Concluding remarks are made in Section 6.

## 2 The SAR Model and its Computational Requirements

In this section, we briefly explain the principles of SAR and the key signal processing operations to be performed<sup>3</sup>.

A vehicle carrying the antenna moves along the azimuth direction at a constant speed. A pulse is transmitted in the range direction. The echoed signal received by the antenna is sampled and stored. The sampled data correspond to a range value. Then, a new pulse is transmitted to obtain new range data. The new pulse corresponds to the next azimuth value. Therefore, the collected data is two dimensional data along the range and the azimuth dimensions.

A single point target appears in many samples in the SAR raw data. Therefore, the signals spread over many samples should be collected to one point. This can be accomplished by two-dimensional convolution. The convolution can be performed efficiently by Frequency Domain Convolution (FDC). The FDC consists of three parts: FFT of data set, multiplication of the transformed data and transformed filter coefficients, and Inverse FFT (IFFT) of the result of multiplication.

The basic SAR signal processing sequence is shown in Figure 1. The preprocessing, postprocessing and other steps that require relatively small amount of computations are not shown in this sequence.

The typical parameters of SAR signal processing are shown in Table 1<sup>3</sup>.

send and receive data. The overhead incurred for the initial setup to perform the communication is called *startup time* and the data transmission overhead is called *data transfer time*. Therefore, the time for point-to-point communication when there is no contention in the network can be modeled as  $T_s + D\tau_d$ , where  $T_s$  is startup time,  $\tau_d$  is data transfer time per byte, and  $D$  is the amount of data transferred in bytes<sup>15</sup>.  $T_s$  is usually much larger than  $\tau_d$ <sup>14</sup>.

SAR implementations can be classified into three cases based on how the input data is partitioned. In the first case, the SAR image data is partitioned into  $N$  blocks in chessboard style, where  $N$  is the number of processors. Each block is assigned to a distinct processor which performs the SAR processing.

In the second case, the SAR image data is first partitioned into  $N$  nonoverlapped horizontal strips. Each data strip has  $Y/N$  rows, where  $Y$  is the number of samples in a column. Each data strip is assigned to one of  $N$  processors which performs FDC in row dimension. Then, each horizontal data strip is partitioned into  $N$  blocks. The size of each block is  $Y/N \times X/N$ , where  $X$  is the number of samples in a row. Each block in a processor is sent to a distinct processor for subsequent processing. Therefore, *All-to-All* communication is needed.

In the third case, there are two groups of processors; one group consisting of  $M$  processors performs the FDC in row dimension, and the other consisting of  $N$  processors performs the FDC in column dimension. The data in a processor in the first group are partitioned into  $N$  blocks, and each of them is sent to a distinct processor in the second group. Thus, there are  $M$  source processors and  $N$  destination processors. Therefore, *M-to-N* communication is required between the two groups. A straightforward serial communication algorithm was reported in<sup>4</sup>. The number of communication steps was  $MN$ . Note that using the MPI *all-to-all* communication primitive is not efficient for this communication.

The *M-to-N* communication pattern is also required for implementing an application that consists of several stages and the computational requirements of the stages vary. In such an application, the processing can be performed on a software task pipeline. The software task pipeline consists of several stages, and each stage has a different number of processors. Therefore, *M-to-N* communication is required between any two consecutive stages. Such processing arises in applications such as Automatic Target Recognition (ATR), SAR, Space-Time Adaptive Processing (STAP), and Sonar.

The third case discussed above offers an ideal environment for implementing many high throughput signal processing applications including SAR. We present an efficient SAR implementation on HPC platforms to obtain high throughput performance. An efficient general algorithm for *M-to-N* commu-

# PARALLEL IMPLEMENTATION OF SYNTHETIC APERTURE RADAR ON HIGH PERFORMANCE COMPUTING PLATFORMS

JINWOO SUH, MONTE UNG, and VIKTOR K. PRASANNA

*Department of EE-Systems, EEB-200C*

*University of Southern California*

*Los Angeles, CA 90089-2562*

*Tel: (213) 740 - 4483, Fax: (213) 740 - 4418*

*E-mail: prasanna@usc.edu*

*URL: <http://ceng.usc.edu/~prasanna>*

In this paper, we show high throughput implementation of SAR on High Performance Computing (HPC) platforms. In our implementation, the processors are divided into two groups of size  $M$  and  $N$ . The first group consisting of  $M$  processors computes the FDC (Frequency Domain Convolution) in range dimension, and the second group of  $N$  processors computes the FDC in azimuth dimension.  $M$  and  $N$  are determined by the computational requirements of FDC in range and azimuth dimensions respectively. The key contribution of this paper is in the development of a general high-throughput  $M$ -to- $N$  communication algorithm.  $M$ -to- $N$  communication algorithm is a basic communication primitive used in many signal processing applications when a software task pipeline is employed to obtain high throughput performance. Our algorithm reduces the number of communication steps to  $\lg(N/M + 1) + n(k - 1)$ , where  $k \geq 2$  and  $n = \lceil \lg_k M \rceil$ . Implementation results on the IBM SP2 and the Cray T3D based on the MITRE real-time benchmarks are presented. The results show that, given an image of size  $1K \times 1K$ , the minimum number of processors required for processing the SAR benchmarks can be reduced by 50% by using the proposed communication algorithm.

## 1 Introduction

Synthetic Aperture Radar(SAR) is a radar system which produces high resolution images using signal processing techniques<sup>3</sup>. SAR signal processing has been implemented on special purpose architectures<sup>13,16</sup> and on HPC platforms<sup>1,4,8,10,11,12</sup>. Due to the high cost of the special purpose SAR processors, HPC platforms are becoming popular for SAR processing. The most time-consuming task in performing SAR signal processing is Frequency Domain Convolution (FDC). Thus, an efficient implementation of FDC is needed.

Another major problem in performing SAR signal processing on High Performance Computing (HPC) platforms is the cost of communication; partially processed data should be moved between processors for subsequent processing. To send data from a processor to another in a message passing environment, the two processors first need to set up a communication channel, and then