Simulation of the Communication Time for a Space-Time Adaptive Processing Algorithm on a Parallel Embedded System

Jack M. West and John K. Antonio

Department of Computer Science, P.O. Box 43104, Texas Tech University, Lubbock, TX 79409-3104
{west, antonio}@ttu.edu

Extended Abstract

The focus of this work involves the investigation of parallelization and performance improvement for a class of radar signal processing techniques known as space-time adaptive processing (STAP). STAP refers to an extension of adaptive antenna signal processing methods that operate on a set of radar returns gathered from multiple elements of an antenna array over a specified time interval. Because the signal returns are composed of range, pulse, and antenna-element samples, a three-dimensional (3-D) cube naturally represents STAP data. Typical STAP data cube processing requirements range from 10-100 giga floating point operations per second (Gflops). Imposed real-time deadlines for STAP applications restricts processing to parallel computers composed of numerous interconnected compute nodes (CNs). A CN has one or more processors connected to a block of shared memory.

Developing a solution to any problem on a parallel system is generally not a trivial task. The overall performance of many parallel systems is highly dependent upon network contention. In general, the mapping of data and the scheduling of communications impacts network contention of parallel architectures. The primary goals of many applications implemented on parallel architectures are to reduce latency and minimize interprocessor communication time (IPC) while maximizing throughput. It is indeed necessary to accomplish these objectives in STAP processing environments. In most STAP implementations, there are three phases of computations, one for each dimension of the data cube (i.e., range, pulse, and channel). To reduce computational latency, the processing at each phase must be distributed over multiple CNs using a single program multiple data (SPMD) approach. Additionally, prior to each processing phase, the data set must be partitioned in a fashion that attempts to equally distribute the computational load over the available CNs. Because each of the three phases process a different dimension of the data cube, the data must be redistributed to form contiguous vectors of the next dimension prior to the next processing phase. This redistribution of data or distributed "corner-turn" requires IPC. Minimizing the time required for interprocessor communication helps maximize STAP processing efficiency.

Driven by the need to solve complex real-time applications that require tremendous computational bandwidths such as STAP algorithms, commercial-off-the-shelf

(COTS) embedded high-performance computing systems that emphasize upward scalability have emerged in the parallel processing environment. In a message passing parallel system, CNs are connected with each other via a common data communication fabric or interconnection network. For the purposes of discussion and illustration, assume that a crossbar with six bidirectional channels is the building block for the interconnection network. Each of the six input/output channels is bidirectional, but may only be driven in one direction at a time. The versatility of the six-port crossbar allows for the interconnect to be configured into a number of different network topologies, including two-dimensional (2-D) and 3-D meshes, 2-D and 3-D rings, grids, and Clos networks. However, the most common configuration is a fat-tree, where the crossbars are connected in a parent-child fashion. In a fat-tree configuration, which is the configuration assumed in this paper, each crossbar has two parent ports and four child ports. The fat-tree architecture helps alleviate the problem of communication bottlenecks at high levels of the tree (present in conventional tree architectures) by increasing the number of effective parallel paths between CNs. Unfortunately, the addition of multiple paths between CNs increases the complexity of the communication pattern in applications such as STAP that involve data redistribution phases.

Additional complexity emerges when each CN is composed of more than one processor or compute element (CE) configured with the shared-memory address space of the CN. In a system with one CE per CN, the communication pattern during distributed corner-turn phases is very regular and well-understood (i.e., a matrix transpose operation implemented in parallel). However, the overall complexity of both the mapping and scheduling of communications increases in systems where the CNs contain more than one CE, for two reasons. First, the communication pattern can be less regular. Second, the message sizes are not uniform.

Two major challenges of implementing STAP algorithms on embedded high-performance systems are determining the best method for distributing the 3-D data set across CNs (i.e., the mapping strategy) and the scheduling of communication prior to each phase of computation. At each of the three phases of processing, data access is either vector-oriented along a data cube dimension or a plane-oriented combination of two data cube dimensions. During the processing at each phase, the contiguous vectors along the dimension of interest are distributed among the CNs for processing in parallel. Additionally, each CE may be responsible for processing one or more vectors of data during each phase. Before processing of the next phase can take place, the data must be redistributed among the available CNs to form contiguous vectors of the next dimension. Determining the optimal schedule of data transfers during phases of data repartitioning on a parallel system is a formidable task. The combination of these two factors, data mapping and communication scheduling, provides the key motivation for this work.

One approach to data set distribution in STAP applications is to partition the data cube into sub-cube bars (see Fig. 1). Each sub-cube bar is composed of partial data samples from two dimensions, while preserving one whole dimension of the data-cube. After performing the necessary computations on the current whole dimension, the data vectors must be redistributed to form contiguous sub-cube bars of the next dimension to be processed. By implementing a sub-cube bar partitioning scheme, IPC between processing stages is isolated to clusters of CNs and not the entire system

(i.e., the required data exchanges occur only between CNs in the same logical row or column).

To illustrate the impact of mapping, consider the two examples shown in Fig. 2 and Fig. 3. For these two examples, assume that the parallel system is composed of four CNs, with each having three CEs, and connected via one six-port crossbar (see Fig 4). Additionally, the number on each sub-cube bar indicates the processor to which the sub-cube bar is initially distributed for processing. Fig. 2 illustrates a mapping scheme where the sub-cube bars are raster-numbered along the pulse dimension. In contrast, the sub-cube bars are raster-numbered along the channel dimension in Fig. 3. As illustrated in the two examples, the initial mapping of the data prior to pulse compression affects the number of required communications during the data redistribution phase prior to Doppler filtering. In the case where the data cube is raster-numbered along the pulse dimension, six messages, totaling 20 units in size, must be transferred through the interconnection network. By implementing the mapping scheme in Fig. 3, the number of required data transfers increases to twelve, while the total message size expands to 36 units. For this small example, the initial mapping of the sub-cube bars greatly affects the communication overhead that occurs during phases of data repartitioning.

To illustrate the impact of scheduling communications during data repartitioning phases, consider the problem depicted in Fig. 5, which is the same problem as shown in Fig. 3. The left-hand portion of the figure shows the current location of the STAP data cube on the given processors after pulse compression. The data cube on the right-hand side of the figure illustrates the sub-cube bars of the data cube after repartitioning. The coloring scheme indicates the destination CN of the data for the data prior to the next processing phase. If any part of the sub-cube bar is a different color than its current processor color in the left-hand data cube, the data must be transferred to the corresponding colored destination node. In this example, the repartitioning phase involves transferring six data sets through the interconnection network. If the six messages were sequentially communicated (i.e., no parallel communication) through the network, the completion time (T_c) would be the sum of the length of each message, which totals 20 network cycles. If two or more messages could be sent through the network concurrently, then the value of T_c would be reduced (i.e., below 20).

Scheduling the communications for each of the six messages through the interconnection network greatly affects the overall performance (even for this small system consisting of only one crossbar). Fig. 6 shows the six messages, labeled A through F, in the outgoing first-in-first-out (FIFO) message queues of the source CNs. Each message's destination is indicated by its color code. The number in parenthesis by each message label represents the relative size of the message. The minimal achievable communication time is dependent upon the CN with the largest communication time of all outgoing and incoming messages. For this example, the minimum possible communication time is the sum of all outgoing and incoming messages on the CNs having two messages, which equals fourteen message units. The actual communication time, T_c, that would result from this example with the given message queue orderings (i.e., schedule) is 17 units. However, changing the ordering of the messages in the outgoing queues will yield an optimal schedule of

messages. The message queues in Fig. 7 are identical to those in Fig. 6 except the positions of messages C and F have been swapped in the outgoing queue. Swapping the ordering of the messages on the green CN allows for an increase in the number of messages that can be communicated in parallel. For this new ordering of queued messages, the actual completion time achieves the optimal completion time of fourteen units. The purpose of this example is to illustrate that the order (i.e., the schedule) in which the messages are queued for transmission can impact how much (if any) concurrent communication can occur. The method used to decompose and map the data onto the CNs will also impact the potential for concurrent communication.

The current research involves the design and implementation of a network simulator that will model the effects of data mapping and communication scheduling on the performance of a STAP algorithm on an embedded high-performance computing platform. The purpose of the simulator is not to optimally *solve* the data mapping and scheduling problems, but to *simulate* the different data mappings and schedules and resultant performance. Thus, the simulator models the effects associated with how the data is mapped onto CNs, composed of more that one CE, of an embedded parallel system, and how the data transfers are scheduled.

The network simulator is designed in an object-oriented paradigm and implemented in Java using Borland's JBuilder Professional version 1.0. Java was chosen over other programming languages because of its added benefits. First, Java code is portable. This feature allows the simulator to run on various platforms regardless of the architecture and operating system. Additionally, Java can be used to create both applications (i.e., a program that executes on a local computer) and applets (i.e., an application that is designed to be transmitted over the Internet and executed by a Java-compatible web browser). Third, Java source code is written entirely in an object-oriented paradigm, which is well-suited for the simulator's design. Fourth, Java provides built-in support for multithreaded programming. Finally, Java development tools, like Borland's JBuilder, provide a set of tools in the Abstract Window Toolkit (AWT) for visually designing and creating graphical user interfaces (GUIs) for applications or applets.

The simulator's functionality is encompassed by a friendly GUI. The main user interface of the simulator provides a facility for the user to enter the corresponding values of the three dimensions of a given STAP data cube and the number of CNs to allocate to processing the STAP data cube using an element-space post-Doppler heuristic and a sub-cube bar partitioning scheme. After providing the problem definition information, the user selects an initial mapping that includes a set of predefined mappings (e.g., raster-numbering along the pulse dimension, raster-numbering along the channel, etc.), a random mapping, or a user-definable customized mapping. Furthermore, the user selects the ordering of the messages in the outgoing queues from a predefined set of scheduling algorithms (e.g., short messages first, longest messages first, random, custom, etc). After providing the necessary input, the network simulator simulates the defined problem and produces the timing results from both phases of data repartitioning. The level of detail that the simulator models could be defined as a medium- to fine-grained simulation of the interconnection network. The simulator assumes the network is circuit switched, and the contention resolution scheme is based on a port number tie-breaking mechanism

to avoid deadlocks. In addition, the simulator incorporates a novel and efficient method of evaluating blocked messages within the interconnection network and queued messages waiting for transfer.

The simulator can be used as a tool for collecting and analyzing how the performance of a system is affected by changing the mapping and scheduling. If it is determined that mapping and/or scheduling choices have a significant impact on performance, then the simulator will serve as a basis for future research in determining the optimal mappings and communications.

Acknowledgements

This work was supported by Rome Laboratory under Grant No. F30602-96-1-0098 and Defense Advanced Research Projects Agency (DARPA) under Contract No. F30602-97-2-0297.

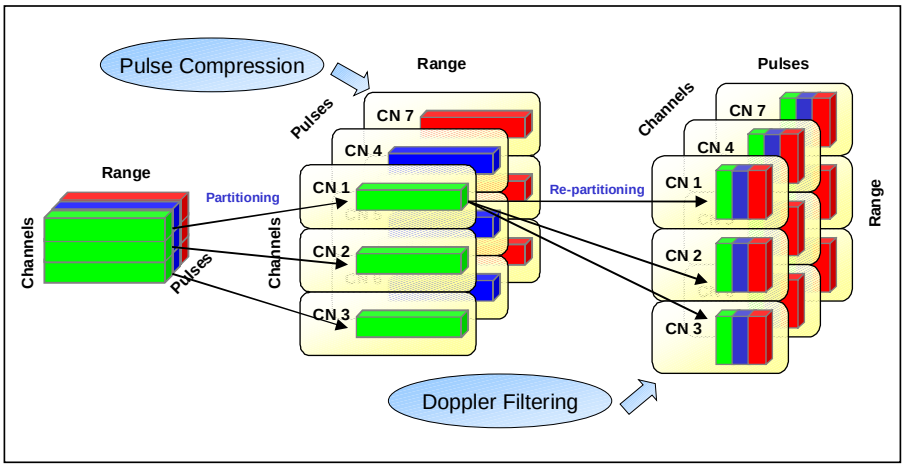

Fig. 1. STAP data cube partitioning by sub-cube bars
(This method of partitioning was first described in [1]).

[1] M. F. Skalabrin and T. H. Einstein, "STAP Processing on a Multicomputer: Distribution of 3-D Data Sets and Processor Allocation for Optimum Interprocessor Communication," *Proceedings of the Adaptive Sensor Array Processing (ASAP) Workshop*, March 1996.

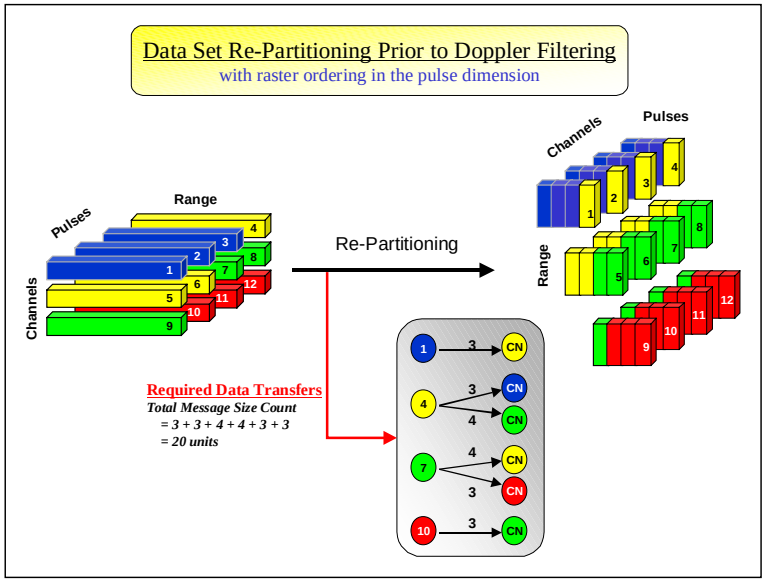

Fig. 2 Data set repartitioning with raster-numbering along the pulse dimension.

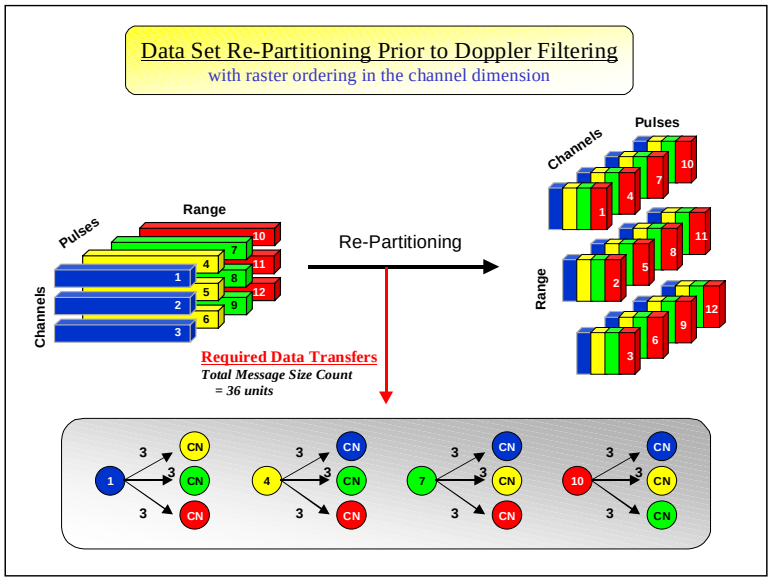

Fig. 3 Data set repartitioning with raster-numbering along the channel dimension.

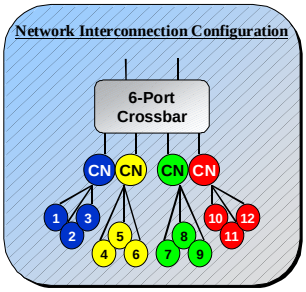

Fig. 4 An example configuration of a four CN (twelve CE) interconnection network.

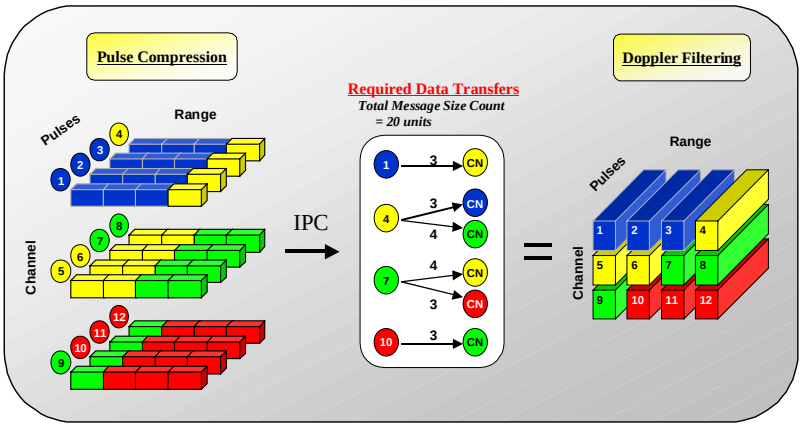

Fig. 5 An example of sub-cube bar repartitioning prior to Doppler filtering.

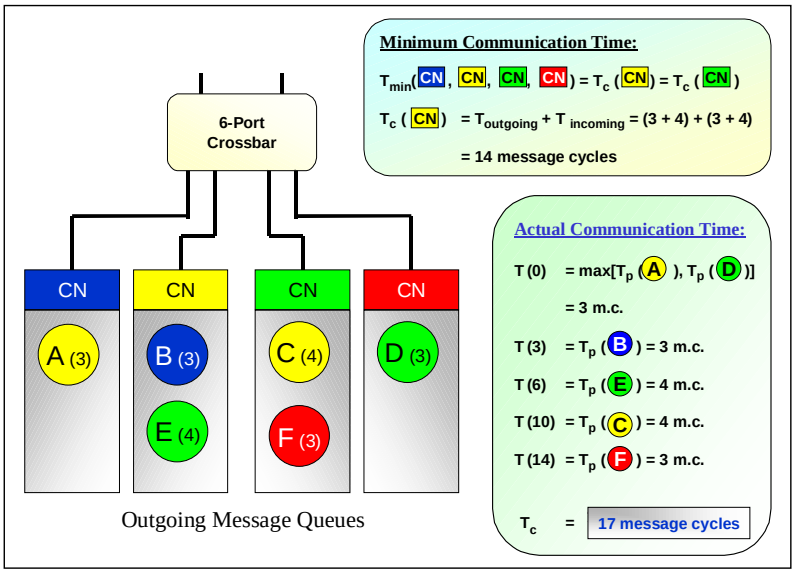

Fig. 6 A sub-optimal communication scheduling example.

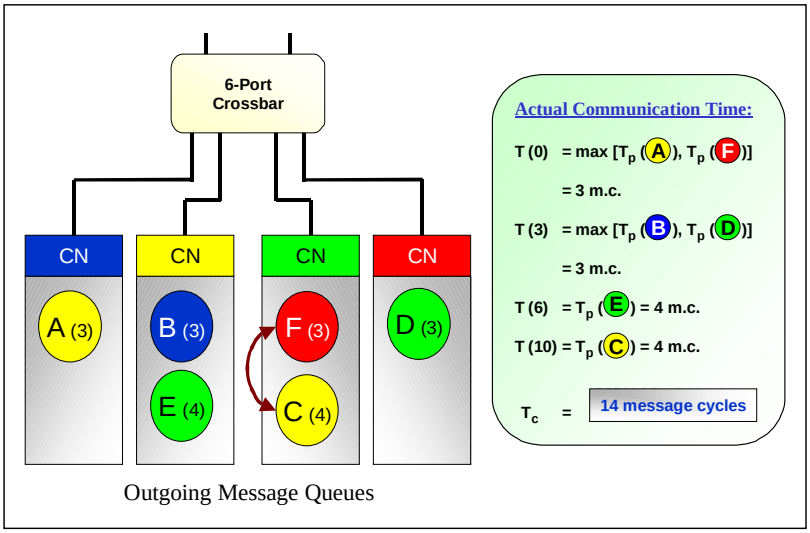

Fig. 7 An optimal communication scheduling example.