The Mapping Method for Parallel Processing of SAR Data

In-Pyo Hong*, Jae-Woo Joo*, Han-Kyu Park** *Regular Members*

ABSTRACT

It is essential design process to analyze processing method and set out top level HW configuration using main parameters before implementation of the SAR processor. This paper identifies the impact of the I/O and algorithm structure upon the parallel processing to be assessed and suggests the practical mapping method for parallel processing to the SAR data. Also, simulation is performed to the E-SAR processor to examine the usefulness of the method, and the results are analyzed and discussed.

Ⅰ. Introduction

Synthetic Aperture Radar(SAR) can provide all weather, high-resolution images of the earth over a wide area[1]. SAR data represent an important source of information for a large variety of scientists around the world[2]. The generation of SAR images from the raw radar data is computationally demanding, and depends on the radar platform and the chosen processing parameters[3].

The SAR processor is the primary image-generating component of the SAR system. A major difficulty in developing a SAR processor for a high resolution and large swath spaceborne imaging sensor is associated with the requirement for a very large amount of data memory in order to access the total information needed to generate a synthetic aperture and the very high speed arithmetic computation requirement. Work during initial design stage set out the processing algorithm definitions, performance analysis, and architecture of the SAR processor. These definitions and results constitute the starting point of this paper.

It is critical design process to analyze the mapping method for parallel processing of the SAR data before implementation of the SAR processor because the generation of SAR imagery involves an extensive amount of computation and data manipulation for the simple reason that the formation of each pixel involves the combination of data from many thousands of echoes[4].

This paper has identified the impact of the Input and Output(I/O) and algorithm structure upon parallel processing methods to be assessed. This assessment has led to a rationale which confirms the top level hardware(HW) configuration, defines the software(SW) and HW mapping method and which will be able to leads to a definition of the criteria against which a suitable HW system may be selected. The impact of the SAR processor algorithms upon the mapping method for parallel processing has been taken into account in this paper.

Also, it is assumed a spaceborne SAR processor for simulation and called Experimental -SAR(E-SAR) processor. In section Ⅱ, the echo processing flow through the E-SAR processor and the parameter values based on SAR processor design criteria are presented.

Ⅱ. E-SAR Processor

The Range-Doppler algorithm is selected for E-SAR processing because Range-Doppler algorithms are built around the fundamental benefits of the Range- Doppler domain[5]. Fig. 1 illustrates

* 국방과학연구소(hip7777@hanmail.net) ** 연세대학교 전기전자공학과
논문번호 : 010115-0524, 접수일자 : 2001년 5월 24일

the processing flow of it.

Fig. 2 shows functional structure of the E-SAR processor and illustrates that it can be conveniently subdivided into three functional segments. The Data Assembly Processor(DAP) takes as its input a stream of echo data records from the synchronized data. It checks the state of completion of the data in terms of its status as a sequence of radar echoes, and extracts and records the echo, replica, sample, and auxiliary radar data.

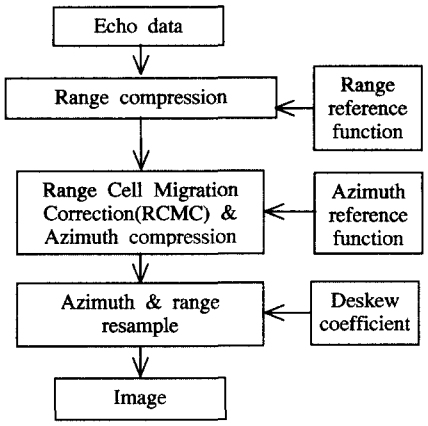

Fig. 1 The processing flow of the E-SAR processor

The completed echo data is passed to the Main Processor(MP) and the auxiliary data passed to the Support Processor(SP) where the processing parameters for the MP are derived. The SP performs those calculations that need to look at information on a scene-wide basis. The results are broken down into data sets applicable to independent data blocks. Each data block can then be processed without reference to the processing of any other block. This allows the MP, which handles most of the processing load, to be efficiently implemented on a parallel array processor. The SP is organized (by algorithm type and by SW unit) into two stages. One derives scene-wide processing parameters (from the data extracted by the DAP), the other derives the block-wide processing parameters required by the MP enabling it to generate and organize the image data according to the image quality requirements.

The parameters and requirements of the E-SAR processor used for simulation are shown at below Table 1.

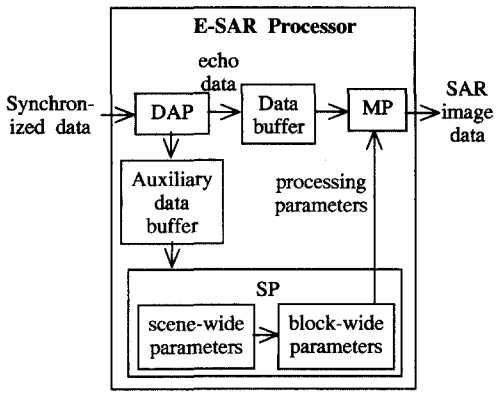

Fig. 2 Functional structure of the E-SAR processor

Table 1. The parameters and requirements of the E-SAR processor

Contents	Values	Unit	Remark
FLOP per sample	600	FLOP	Floating point operations
Required processing rate	2.3×10^9	FLOP/sec	
The number of CPU	23		
The mean data input rate	3	Mbytes/sec	for Single Look Detected(SLD)
The mean data output rate	20	Mbytes/sec	for SLD
The size of a scene	$10^4\times10^4$		The number of resolution cells

Ⅲ. Mapping Method for Parallel Processing & Simulation

3.1 Mapping Concept

The basic unit of SAR data processing is a scene. The types of processing tasks involved in SAR processor processing reveal the parallel processing concepts best suited to the situation. Top level tasks can be categorized as follows: tasks associated with scene-wide tasks associated with extracting and deriving data and processing

parameters which apply to the entire scene, block-wide tasks associated with deriving the parameters for processing discrete data blocks independently of one another, sample-by-sample computationally intensive tasks involved in processing independent data blocks to generate the image output, Man Machine Interface(MMI) tasks, and I/O tasks. In order to process an entire scene the scene-wide, block-wide, and sample-by-sample tasks must be implemented in sequence.

It has been assumed that we should achieve an effective FLOP rate of about 2.3×10^9 FLOP/sec in the sample-by-sample tasks and that in practice this would require about 23 CPU's working simultaneously. In this section we analyze the impact of I/O requirements and algorithm structure upon the parallel processing in order to seek opportunities for sub-division of the main processing task into independent parallel paths.

3.2 Impact of I/O and Algorithm Structure upon Parallel Processing

The mean data input rate is of the order of 3 Mbyte/sec and mean data output rate for an SLD product is of the order of 20 Mbyte/sec. The HW must provide sufficient disc controllers to permit aggregate I/O rates of 23 Mbyte/sec continuously while a scene is being processed in order to achieve the required throughput rate. Non-blocking I/O would be an advantage. The following features of the processing problem influence the processor configuration :

- Data I/O must be smooth, line-by-line read and write.
- Input data will be in echo window sequence.
- Output data will be in image line sequence.
- With the Range-Doppler algorithm we can compress in range line-by-line order.
- With the Range-Doppler algorithm block processing the azimuth compression is done in blocks of compressed range lines. It can only be initiated when sufficient range compressed lines have been accumulated.
- Therefore we require a RAM buffer in which range compressed data can be accumulated before azimuth compression can be initiated.
- Azimuth compression tasks take place on data blocks of along track extent equal to the range block dimension and of a different across track width.
- Each block can be processed independently with its own parameters.
- The output data from azimuth de-skew comes as incomplete range line segments as each block is processed (see Fig. 3).
- The output achieved data rate to disc would be lower for pseudo-random writing of range line segments as they are ready than for sequential output of complete segments.

Therefore to achieve the pseudo-continuous output write to disc we require an image assembly RAM buffer in which to accumulate output range line segments. Complete, in sequence, range lines would be written from this buffer to disc as a rolling operation.

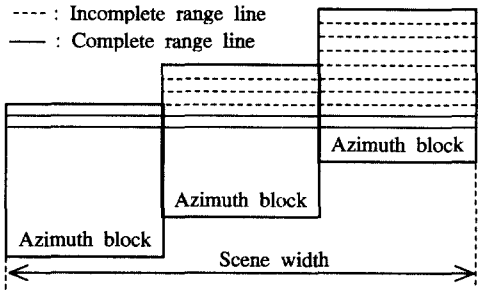

Fig. 3 Incomplete range lines from azimuth block processing

With all these features in mind, it is clear that the structure of the Range-Doppler algorithm in conjunction with making best use of I/O controller facilities indicates a requirement for main processing SW using two primary RAM buffers and two I/O buffers. These buffers can act as convenient multiple CPU system data transfer interfaces. Their position in the processing sequence therefore shapes the allocation of tasks to CPU's. The SW mapping must also be designed to respect these buffer and interface

positions.

Multiple CPU processing opportunities arise in at least two dimensions(see Fig. 4) :

●Range and azimuth processing may be paralleled using the corner turning buffer[5][6], *ctb*, as an inter-process data interface.

●Across track sub-swaths can be defined since independent block processing is possible. Within sub-swaths the *ctb* and *iab* can be partitioned across track allowing CPU's to work independently within sub-swaths.

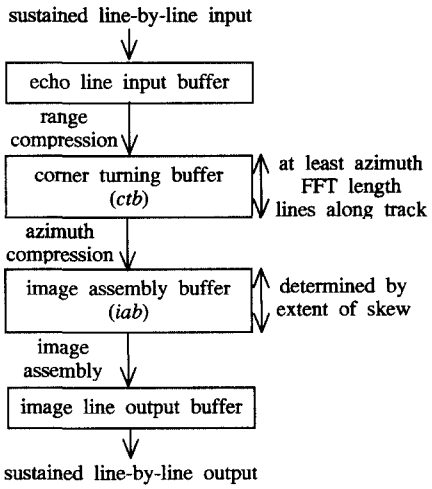

Fig. 4 Main processing primary RAM data buffers

3.3 The Principles of Multiple CPU Processing

The most efficient method is always to have each CPU executing code which operates independently on locally partitioned data. The SW design must be such that data transfer between CPU's is minimized. This has the consequence that as we add more processors the overall processing rate will increase in proportion to the total number of CPU's. Array processor architectures also support memory sharing by way of physical memory accessible to a number of CPU's. Whatever multiprocessor structure is chosen, the SW structure must reflect it by providing independently executable tasks at suitable points in the sequence. In other words the SW must be such that many copies of the same executable code can be spawned to run independently with their own parameters and data. In the case of an array processor each process can be directly associated with a CPU and its local RAM.

A process running on a CPU is able to free-run once initiated until its end point. Only occasional synchronization operations are required to co-ordinate memory sharing and data transfer & access operations. These inter-process communications are supported by the array processor operating system. The SW structure requires control layers to initiate and control the process flow.

3.4 Parallel Processing Method

A wide range of multiple CPU task configurations can be devised[7]. Some methods illustrating the principles are outlined in the following sub-sections. In each method advantage is taken of the opportunities for sub-division into independent tasks provided by the algorithm structure and by the echo data structure.

3.4.1 Method 1

The data is divided into independent sub-swaths. The corner turning buffer is partitioned into sub-swaths. Input communication is via the input line buffer. The image assembly buffer is common and accessed by all sub-swath processors. Each sub-swath uses 1 CPU for range and azimuth compression as a single process. Fig. 5 depicts the processing sequence for the four sub-swaths.

The penalties of this method are that, between sub-swaths : raw data must overlap by the range reference function length, range compressed data must overlap by range cell migration curvature allowance plus the range interpolator length.

Worthwhile gain in data processing rate by increasing the number of sub-swaths (each processed by one CPU) is limited to about 8 sub-swaths, since for a given echo line length the

greater the number of sub-swaths, the greater the proportion of range data overlap. With more than 8 sub-swaths adding more CPU's gives a diminishing return in terms of processing speed.

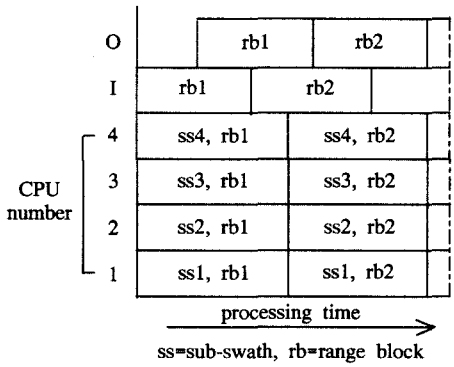

Fig. 5 Parallel processing in four sub-swaths

3.4.2 Method 2

This method adds potential for speed enhancement over method 1 by sub-dividing the processing of each sub-swath between range and azimuth compression tasks. The data and tasks are again partitioned by sub-swath. Within each sub-swath 1 CPU is dedicated to range compression and another is dedicated to azimuth processing simultaneously. Data from range compression is pipelined to the azimuth processors[8] via a partitioned corner turning buffer[6], as illustrated in Fig. 4.

The same penalties and limitations of multiple sub-swath processing apply as applied to method 1. It is clear from the idea illustrated in Fig. 6 that the highest pipeline efficiency is achieved if the task durations are balanced between range and azimuth processing. In fact the processing load imposed by range tasks is generally significantly lower than for azimuth leading to the range processors being be idle for significant periods of time. This method certainly increases processing speed by doubling the number of CPU's operating in each swath (for at least some of the time) but generally is not optimally efficient. In practice optimal efficiency may be less important than the gain in speed achieved.

Note also that in this method the SW structure must be different from that required in method 1 in that separate calls to range and azimuth processes must be available; thus the HW mapping influences the SW structure. Inter-process communication is via the partitioned *ctb* and *iab* data buffers.

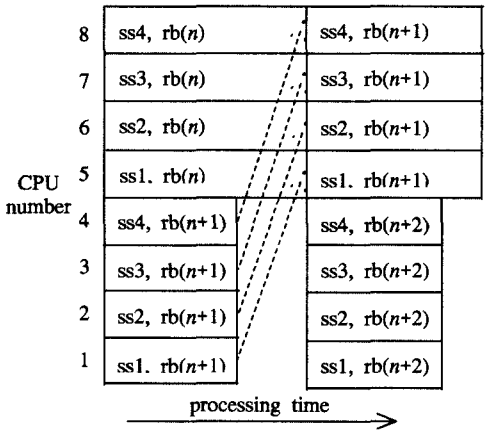

Fig. 6 Parallel processing in four sub-swaths

3.4.3 Method 3

This method is a further refinement of the ideas in method 2. In principle, the range and azimuth compression tasks are further subdivided among a greater number of CPU's :

- Data and tasks are partitioned by sub-swath.
- Range and azimuth tasks are separate (pipe-lined).
- In range the data and tasks are further subdivided into groups (along track) of range lines with 1 CPU dedicated to a sub-aperture of range lines.
- In azimuth, within each sub-swath, each CPU is dedicated to a smaller group of azimuth blocks than is associated with the sub-swath as a whole.

Thus at any time, once the processes are in dynamic equilibrium (with non blocking I/O), within a sub-swath the range compression CPU's work simultaneously on data from range block $n+1$ in a pipe-line with the azimuth compression CPU's which are working on data from range block n. sub-swaths are processed in parallel. Fig.

7 illustrates this method.

This method allows us to enhance efficiency by adjusting the proportion of CPU's dedicated to range processing in relation to those doing azimuth processing so that pipelining leads to shorter idle periods. It also allows us to use of a number of CPU's greater than $2\times$(*number of sub-swaths*).

There is an additional penalty introduced by the task subdivision in range: within a sub-swath the across track interpolation overlap must be handled by either processing the overlap data twice or by exchanging data between CPU's or by sharing memory between CPU's within the group. Each of these alternatives adds a little time overhead.

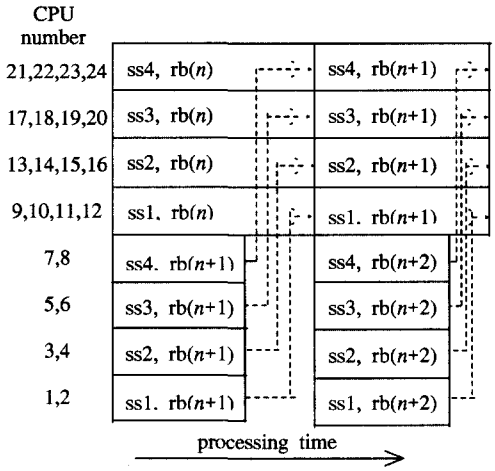

Fig. 7 *Range-azimuth pipeline in subdivided sub-swaths*

Ⅳ. The Result and Discussion

Scene-wide and block-wide tasks are executed in sequence and depend upon the data assembly tasks having been executed first. Whether or not the block-wide tasks are grouped together with the scene-wide tasks or are placed in the main sample-by-sample processing is a question of the balance between the volume of block-wide data output and processing speed. Optimal balance and operational speed can be achieved by grouping the block-wide tasks with the scene-wide with block data parameter transfer by RAM. The computing resources implied by this mapping presents the opportunity for pipelining the synchronized data, and assembly, support, and main processing tasks, as illustrated in Fig. 8. In this way best use may be made of the computing resources available.

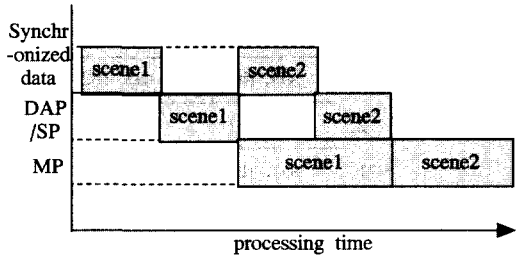

Fig. 8 Pipeline processing using separate computing resources

There may be idle times in the pipeline sequence for one or more resources. Best use of resources is made if the task load and resource are designed to give each resource a task of the same duration. The design concept must be first to satisfy the main processing throughput requirement, this uses the expensive HW, and then to design the resources of the support and synchronized data so that they never cause interruption to the main processing task.

The requirement for a multi-CPU MP has been identified. Best efficiency is always achieved when each CPU is working independently on a partitioned data blocks. A general purpose multi-CPU workstation, for example a SUN workstation, does not give a computational rate increase in linear proportion to the number of CPU's added primarily because of its methods of memory access and co-ordination. In addition multi-CPU workstations do not support the number of CPU's required by the E-SAR processor. A different HW and SW architecture is required for the MP. An array processor is a more efficient system for executing many similar tasks in parallel using partitioned data blocks.

However, an array processor requires a host

computer system to isolate it from the LAN and to co-ordinate its activities. Operationally we can take advantage of this situation since the DAP and SP tasks can in principle be executed on the host workstation. Once the host has initiated the array processor activities it can be used to process the parameters for the next scene and to handle the MMI activities of the current process. This facilitates operational pipelining. Then, the proper top level HW configuration is shown in Fig. 9.

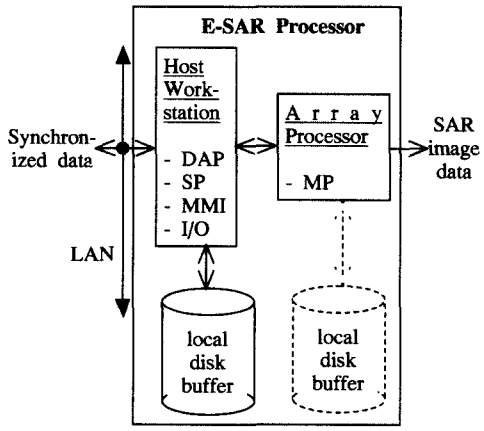

Fig. 9 Top level HW configuration

Many possible methods of parallel processing in array processor can be imagined. For E-SAR, requiring approximately 23 processors, one possible method based on method 3 may be appropriate : define data into 4 sub-swaths : with each sub-swath dedicate :

- 2 CPU's for range processing(partitioned data),
- 4 CPU's for azimuth processing(partitioned data),
- arranged as a range-azimuth pipeline(the range processing load ≅ 1/2 that of azimuth),
- thus, total : 24 CPU's in parallel.

This configuration meets the performance levels derived in this paper, using CPU's with performance at the level currently available. Data common to all processors, for example radar auxiliary data, universal constants, can be declared as read only so that each CPU takes a copy to its own local memory partition. I/O should be supported by non-blocking methods. 1 disc controller is required for input and at least 1 controller for output, served either by the array processor or host. Having first defined the MP throughput capability the host workstation resources can be defined so that neither of them constitutes a data flow bottleneck.

V. Conclusion

Most of the data handling in a SAR processor can be thought of as correlation of the received signal with a two-dimensional reference function. Thus, it is very essential to apply the mapping method for parallel processing of the SAR processor during design stage before implementation.

This paper identifies the impact of the I/O and algorithm structure upon parallel processing methods to be assessed. It describes work on the derivation of the mapping method and sets out the parallel processing procedure and configuration of the SAR processor. Also, simulation has been performed to the E-SAR processor. It suggests proper top level HW configuration and parallel processing method of that. HW performance improves significantly from year to year. No special HW is required and SW can be developed without dependence upon final HW selection, since no HW dependent protocols are required.

Therefore, our approach can provide a practical mapping method for parallel processing of the airborne as well as spaceborne SAR data processing.

REFERENCES

[1] Kuang Y. Liu, and Wayne E. Arens, Spacecraft on-board SAR image generation for Eos-type missions, *IEEE Trans. Geosci. Remote Sensing*, vol. 27, no. 2, pp. 184-192, March 1989.

[2] Ursula Benz, Klaus Strodl, and Alberto

Moreira, A comparison of several algorithms for SAR raw data compression, *IEEE Trans. Geosci. Remote Sensing*, vol. 33, no. 5, pp. 1266-1276, Sep. 1995.

[3] Einar-Arne Herland, A SAR processor for a GIS environment, in *Proc. Int. Geosci. Remote Sensing. Symp. IGARSS 91*, vol. 2, pp. 623-626, 1991.

[4] Charles Elachi, *Spaceborne Radar Remote Sensing:Applications and Techniques*. New York : IEEE Press, ch1, 4, 5, 1988.

[5] Floyd M. Henderson, and Anthony J. Lewis, *Manual of Remote Sensing, vol. 2 : Principles and Applications of Imaging Radar*. New York : John Wiley & Sons, Inc., ch. 2, 1998.

[6] Donald R. Wehner, *High Resolution Radar*. Norwood, MA, Artech House, Inc., ch 6, 1987.

[7] Dan I. Moldovan, *Parallel Processing : From Applications to Systems*. San Mateo, California : Morgan Kaufmann Publishers, Inc., ch 4, 5, 6, 1993.

[8] John C. Curlander, and Robert N. McDonough, *Synthetic Aperture Radar : Systems & Signal Processing*. New York : John Wiley & Sons, Inc., ch 9, appendix A, 1991.

홍 인 표(In-Pyo Hong) 정회원
한국통신학회 논문지 Vol. 26, No. 9B, 2001년 9월 참조

주 재 우(Jae-Woo Joo) 정회원
한국통신학회 논문지 Vol. 26, No. 9B, 2001년 9월 참조

박 한 규(Han-Kyu Park) 정회원
한국통신학회 논문지 Vol. 26, No. 9B, 2001년 9월 참조