A distributed approach to efficient time-domain SAR processing

A. Reigber[1], M. Jäger[1], A. Dietzsch[1], R. Hänsch[1], M. Weber[1], H. Przybyl[1] and P. Prats[2]

[1] Berlin University of Technology (TUB), Computer Vision and Remote Sensing Group,
Franklinstraße 28/29, Sekretariat FR3-1, D-10587 Berlin, Germany.
Tel. +49-30314-23276, Fax. +49-30314-21114, E-mail: anderl@cs.tu-berlin.de
[2] German Aerospace Center (DLR), Microwave and Radar Institute
P.O. Box 1116, D-82234 Wessling, Germany

Abstract— **This paper presents a distributed approach for time-domain focusing, which significantly enhances the overall efficiency by distributing the computational load across a (potentially large) number of networked computers. The system described includes the so-called master, responsible for pre-processing, the distribution of fragments of raw-data data and the collection of processed image fragments. Fragments of raw-data are passed to so-called slaves, any number of which can be connected to the master, which are responsible for the focusing itself. Master and slave actively communicate over the network to organise the entire process in a scalable manner. In this way, time-domain processing can be accelerated by a factor that is virtually linear in the number of participating slaves.**

This paper summarises the current status of software development, realised in a platform-independent way using the IDL and Java languages. Additionally, some preliminary evaluations of performance, scalability and the required network infrastructure are given. Some examples of SAR data, acquired by the airborne sensor E-SAR of DLR, and processed with the system described are shown.

I. Introduction

A number of advanced algorithms for focusing the raw-data acquired by synthetic aperture radar (SAR) sensors are described in the literature. For reasons of computational complexity, most established approaches focus the data in the one- or two-dimensional frequency domain. A popular representative of such FOURIER-domain based focusing algorithm is the extended chirp scaling (ECS) algorithm [1], which allows an accurate and phase preserving processing of spaceborne SAR data without the need for interpolation in correcting the range cell migration (RCM). However, in case of large squint angles above 15-20^o or very wide azimuth beam-widths, the approximations made in the algorithm are no longer valid.

On the other hand, wavenumber-domain, or ω-k processing algorithms [2], are free of approximations and commonly accepted to be an ideal solution of the SAR focusing problem in case of a straight sensor trajectory. This type of technique is based around an interpolation step in the wavenumber domain, the so-called STOLT-mapping, which allows one to focus data to very high levels of resolution, independent of the range and azimuth bandwidth. Nevertheless, this approach is only applicable to space-borne SAR data, since it does not include a high precision motion compensation step.

Since modern sensor technology is developing more and more towards higher resolution in range and azimuth, a need for very accurate focusing techniques of such wide-band data is evolving. An attractive alternative are time-domain based, exact SAR processing techniques, which can be regarded as the optimal solution to the SAR focusing problem. Time-domain approaches work independent of the system bandwidth and are able to take into account the influence of sensor motion, squint angle, terrain topography and non-equidistantly sampled apertures precisely. However, time-domain focusing is extremely expensive computationally and reasonable processing times can only be achieved by re-introducing certain approximations [3]. This circumstance is the reason why time-domain techniques have not yet been more widely employed, in spite of their obvious appeal regarding the high-quality focusing of SAR data.

This paper presents an efficient non-approximative time-domain processing approach, which distributes the computational burden over an arbitrary number of processing nodes in a network in order to achieve a significant decrease in processing time. The concept is similar to the one presented in [4], with the difference that the entire system has been implemented in IDL and Java languages. The platform-independence of both languages makes it possible to mix node computers with different architectures and varying raw processing power. Using IDL for the core signal processing components ensures easy maintainability and extensibility of the concept, while Java possess powerful network communication capabilities.

II. The distributed time-domain processor

A. General architecture

The distributed time-domain (DTD) SAR processor described in the following sections consists of a so-called master process and a variable number of client processes, typically running on several networked computers and communicating with the master. Both parts are composed of an IDL component and a Java component, i.e. there is an IDL-master and a Java-master and equally IDL-clients and Java-clients. The

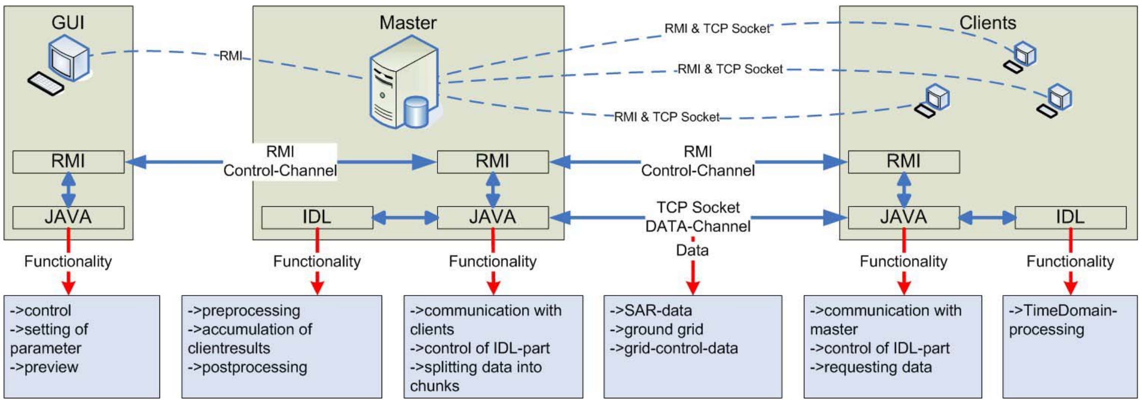


Fig. 1. Overview of the architecture of the distributed time-domain processor

IDL parts are responsible for the data handling and processing, and instantiate a corresponding Java objects via the IDL-Java bridge. The Java-clients are responsible for communication between Java and IDL, as well as between master and clients. They register with the Java-master using Java remote method invocation (RMI) [5]. Once the Java-clients are registered, the user can start the processing through a graphical user interface (GUI), which connects to the Java-master from any computer in the network. The GUI allows the user to set processing parameters and specify the location of files needed for processing, such as sensor parameters or the flight path.

After starting, the GUI sends a command to the Java-master to begin processing with the specified parameters. The Java-master then induces the IDL-master to begin pre-processing the data. When finished, the Java-master extracts the pre-processed raw-data and distributes packages consisting of rows of pre-processed data to the Java-clients, from where they are forwarded to the IDL counterparts. The IDL-clients then perform the actual time-domain processing of their data package. When a client has finished, it saves the processing result locally and asks the master for another data package to process. After the complete raw-data is processed, the Java-master accumulates the partial results of all clients and transfers them to the IDL-Master for post-processing and the creation of the final result. An overview of the general architecture of DTD is given in Fig. 1

B. *The processing kernel*

The processing kernel in the clients is a back-projection time-domain processor, which computes the contribution to the final result $v(x, y, z)$ at grid positions (x, y, z), that arise from a single range-focused range line of the raw-data supplied by the master process.

$$v(x, y, z) = u(r) * \frac{4\pi}{\lambda}\sqrt{(x - x_s)^2 + (y - y_s)^2 + (z - z_s)^2} \tag{1}$$

where (x_s, y_s, z_s) denotes the sensor position at which the respective line of raw-data was acquired, and $u(r)$ is the raw-data corrected for range-cell-migration in an interpolation step. From a single raw-data line, Eq. 1 has to be solved for all coordinates of the output grid that fall inside the length of the synthetic aperture around the azimuth position of raw-data line. In IDL, this operation can be vectorized such that only a single loop over the length of the synthetic aperture is required. Vectorization makes the performance of the processing kernel comparable to C/C++ code in terms of performance.

The output grid can be freely chosen and is independent of the sampling of the raw-data itself. Therefore, images can be focused in slant-range, as usual, or directly in ground-range coordinates or even considering topographic information (if available). In the latter case, clients locally accumulate not only the image result but also the received output grid coordinates and request only portions of the grid that are missing from the master. This strategy reduces the amount of network traffic and prevents the associated bottleneck.

C. *Master / Client communication*

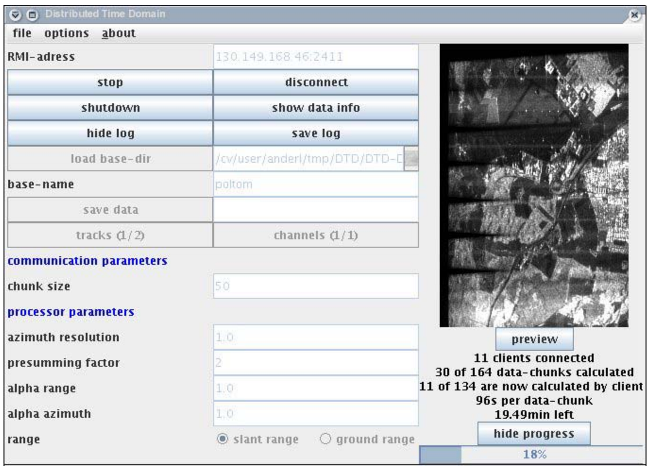
As mentioned above, both master and client consist of two parts: The processing part written in IDL and the (network) communication part written in Java. The IDL parts never interact directly among each other: all communication passes through the Java components of the system. The internal communication between the IDL and the Java part is accomplished with the help of the IDL-Java bridge that is integrated in the IDL distribution.

In doing so, three major problems must be overcome. Firstly, the IDL-Java bridge only allows one-way communication from IDL to Java, but not vice versa. Thus, IDL must wait and poll the Java part for events from the network. Depending on the event received, the appropriate Java method is called and its return value constitutes the message received. Secondly, the IDL-Java bridge is not aware of the IDL primitive *COMPLEX*. Hence, the flow of complex data between IDL and Java must be transformed to data understood by Java and the IDL-Java bridge. Thirdly, the available memory that can be used for Java applications is constrained by the Java Virtual Machine itself. That means that the available memory in the Java part is limited and that the transferred data must be split up to pass the IDL-Java bridge. A detailed description of this process is given in section II-D.

TABLE I

LIST OF ACHIEVED PROCESSING TIMES WITH DIFFERENT NUMBER OF CLIENTS.

No. of clients	processing time
1	290 min
2	155 min
4	80 min
8	41 min
12	27 min
16	20 min



RMI-adress: 130.149.168.46:2411

/cv/user/anderl/tmp/DTD/DTD-D

base-name: poltom

communication parameters

chunk size: 50

processor parameters

azimuth resolution: 1.0

presumming factor: 2

alpha range: 1.0

alpha azimuth: 1.0

range: [✓ slant range] ground range

11 clients connected
30 of 164 data-chunks calculated
11 of 134 are now calculated by client
96s per data-chunk
19.49min left

18%

Fig. 2. Screenshot of the Java-GUI that controls the DTD processor, which is, in this case, processing with 11 client nodes.

The Java parts of master and clients communicate by the means of Java RMI [5] and TCP/IP sockets. TCP/IP sockets are used to transfer large blocks of data between two endpoints in an efficient manner. Java RMI is utilized by the master to execute functions in the clients (e.g. start processing, initiate result delivery, etc.) and by the clients to trigger control functions in the master (e.g. initiate selection of new data fragments, sending status/error messages, etc.).

Once the master is set up, it starts an instance of the Java RMI registry, exports its RMI-object for the clients to bind to, and enters its initial state. The clients then have to be set up by configuring the address and port of the master's RMI registry[1]. When started, the clients automatically connect to the master, enter their initial state and wait for the processing to begin or immediately participate in an ongoing processing. Then, they request initial parameters and the first data fragment, which is transferred over the TCP/IP socket. After finishing one fragment, a new one is requested by contacting the master with a call to its Java RMI object. When all pieces are successfully processed, or the user hits the "preview" button in the GUI, the master contacts the clients and the completed portions of the processing result are collected. This the moment of most intense network usage, since all clients are requested to send their computed parts of the image back. To lower the impact on the master and the network, the clients are scheduled by the master and, thus, send back their results one after another. If the processing was interrupted by requesting a preview, the computation continues after the transfer, otherwise the clients return to their initial state and the master accumulates the final result before returning to its initial state as well. Upon cancellation, both the clients and the master discard any processed data and return to their initial states to wait for a new computation to start.

D. Distribution of data

As in every distributed processing approach, the data must be fragmented and distributed among the available client processes. In the ideal case, all clients are equally fast and the data can be statically split up into n pieces, where n denotes the number of clients. In general, where clients can join and disappear at any time, this strategy is not satisfactory, and the data must be dynamically partitioned. In this regard, it is important to note that the processing of one column of the raw data results in image data with m columns, where m denotes the length of the synthetic aperture. Thus, to avoid unnecessary fragmentation and communication overhead, the fragments distributed to each client should be chosen as closely located to one another as possible.

The algorithm divides the columns of the range-compressed raw data into chunks. Each chunk contains the same (configurable) amount of columns, except for the last one. They are used to make use of efficient transfers of several columns at once, reduce the aforementioned fragmentation and prevent inefficient single column transfers. For each client, the chunk surrounded by the largest amount of un-processed data is chosen as the client's starting point. The master then provides the client with chunks in the spatial neighborhood of the starting point. The results of processing a chunk of raw-data are accumulated locally by the clients until the end of the processing in order to avoid unnecessary communication overhead. If there is no chunk left on either side of the already processed chunks, a new starting point is selected using the criteria outlined above. This is repeated until all chunks (and thus all columns) are processed and the final image can be synthesized by the master.

III. EXPERIMENTAL RESULTS

Since the proposed processor is still in development, the assessment of processing quality is restricted to basic visual inspection. Fig. 3 depicts the range-compressed raw-data of the test-site, acquired by the airborne E-SAR sensor at L-band, as well as the focused SAR image after time-domain processing. The image appears well focused and no obvious processing artefacts are visible. It should be noted that for such data, with only 100MHz of range bandwidth and about 10° azimuth beamwidth, FOURIER-domain processors can also be expected to perform well.

Tab. I shows a comparison of processing times with different numbers of clients. The raw-data has a size of 2048x8192 pixels, the length of the processed synthetic aperture is about

[1]One might need to configure firewalls, if present, as well

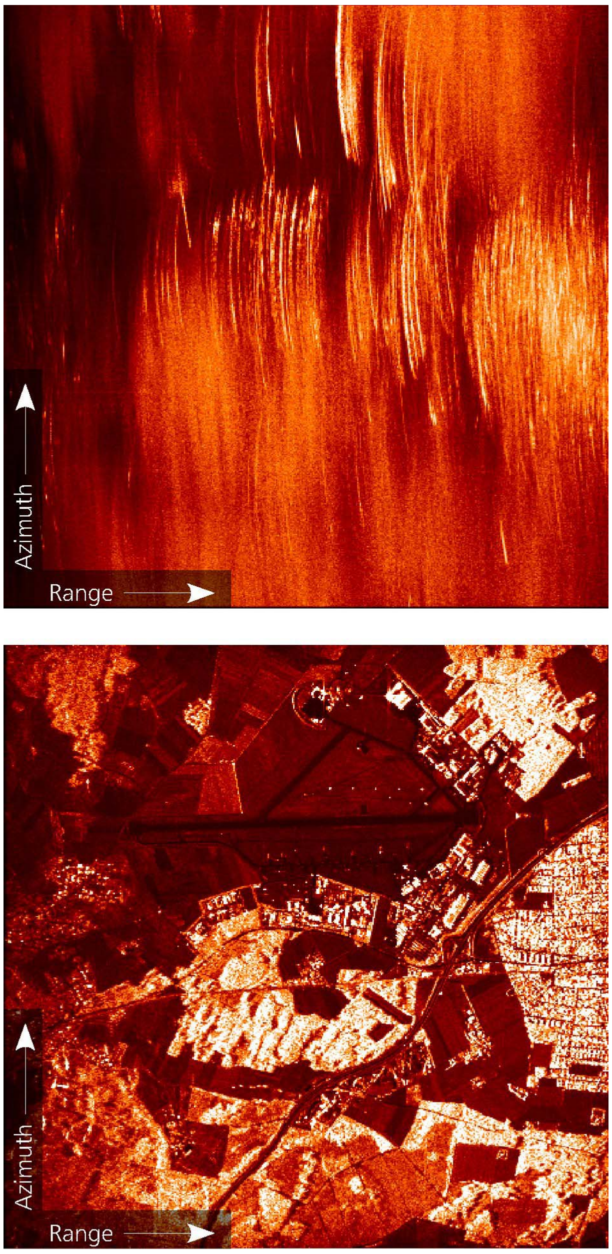


Fig. 3. Top: Raw-data after range compression. Bottom: Processed image result.

1500 pixels, corresponding to an azimuth resolution of 1.0m in this case. All computers are connected by standard 100MBit Ethernet. Unfortunately, not all computers involved have the same processing power, and were not completely idle apart from DTD processing. A heterogenous workstation cluster, containing Linux, Windows-XP and Windows-Vista clients was used, with CPU clock speeds ranging from 1.5 to 3.0GHz. Due to the diversity of the systems, the results in Tab. I give only a rough indication of the speed improvement achieved. The results seem to indicate that processing performance scales almost linearly with the number of clients.

IV. Summary & Conclusions

An innovative approach to time-domain SAR processing has been proposed, based on the distribution of the computational load over an arbitrary number of networked computers. In this way, it becomes possible to process SAR data in the time domain acceptably quickly, if a sufficiently large number of computation clients is available.

The proposed concept has several advantages. In contrast to conventional Fourier-domain processing approaches, no approximations are necessary and high-precision results can be achieved even in case of non-linear sensor trajectories, uneven sampling, strong ground topography, very wide azimuth beam-width or large squint angles. The proposed processor can be used in heterogeneous network clusters, i.e. it runs in all environments that support IDL and Java SE $\geq$ version 1.5 (testing was carried out under Mac OS X, Linux, Windows XP and Vista). Additionally, by using IDL and Java, the system is very easy to maintain and to extend. It has been shown that an almost linear scaling of the overall processing speed with the number of clients can be achieved. However, there are also some drawbacks in integrating IDL and Java, namely the high memory requirements of Java and the burden of the additional data conversion across the IDL-Java Bridge.

Future developments will focus on the operationalisation and generalisation of the proposed concept. In particular, the automatic discovery of master/client nodes, the optimal adaption to client speeds and a generalisation of the pre- and post-processing steps are among the improvements planned in the near future.

References

[1] A. Moreira, J. Mittermayer and R. Scheiber: "Extended Chirp Scaling Algorithm for Air- and Spaceborne SAR Data Processing in Stripmap and ScanSAR Imaging Modes", *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 34, No. 5, pp. 1123-1136, 1996.

[2] C. Cafforio, C. Prati and F. Rocca: "SAR Data Focusing Using Seismic Migration Techniques", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 27, No. 2, pp. 194-207, 1991.

[3] L. M. H. Ulander, H. Hellsten and G. Stenstrom: "Synthetic-aperture radar processing using fast factorized back-projection", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 39, No. 3, pp. 760-776, 2003.

[4] A. R. Brenner: "Distributed SAR processing in the time domain" *Proceedings of EUSAR 2002, Cologne, Germany*, pp. 573-576, 2002.

[5] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp (04/24/07)