Efficient parallel implementation of polarimetric synthetic aperture radar data processing

Sergio S. Martinez\textsuperscript{a}, Prashanth R. Marpu\textsuperscript{a} and Antonio J. Plaza\textsuperscript{b}

\textsuperscript{a}Masdar Institute of Science and Technology, PO Box 54224, Abu Dhabi, United Arab Emirates.

\textsuperscript{b}University of Extremadura, Caceres, Spain.

ABSTRACT

This work investigates the parallel implementation of polarimetric synthetic aperture radar (POLSAR) data processing chain. Such processing can be computationally expensive when large data sets are processed. However, the processing steps can be largely implemented in a high performance computing (HPC) environment. In this work, we studied different aspects of the computations involved in processing the POLSAR data and developed an efficient parallel scheme to achieve near-real time performance. The algorithm is implemented using message parsing interface (MPI) framework in this work, but it can be easily adapted for other parallel architectures such as general purpose graphics processing units (GPGPUs).

Keywords: HPC, MPI, Cluster of computers, POLSAR, H/alpha/A decomposition

1. INTRODUCTION

Lately, polarimetric synthetic aperture radar (POLSAR) technology has shown a lot of potential in several earth-observation applications as is evident through space-borne missions such as TerraSAR-X, RADARSAT-2, ALOS-PALSAR, etc. which are preceded by several airborne campaigns.\cite{successSpaceborne} The success of the space-borne missions means that a large volume of data is being made available. POLSAR data is characterized by a complex scattering matrix representing the polarization components of the backscatter, which is used to build the covariance and coherence matrices which are further processed to analyze the data. Computationally, processing such data is a huge challenge. In this work, we explore the possibility of implementing the POLSAR data processing chain of the standard target decomposition algorithm\cite{standardDecomposition} in a high performance computing environment to achieve near-real time processing of data. We first investigate the implementation in a cluster, based on message parsing interface (MPI),\cite{mpi} but the same scheme could be adapted for implementation on general purpose graphics processing units (GPGPUs).\cite{gpgpus} Such an implementation has a lot of potential in real-time on-board processing of data.

2. POLSAR DATA PROCESSING CHAIN FOR TARGET DECOMPOSITION

POLSAR data is often represented as a $3 \times 3$ complex coherence matrix ($T$) which embeds all the information pertaining to the different polarization combinations which in turn characterize the scattering behavior in a pixel. The multi-look coherency matrix for $n$ number of looks is given as:

$$\langle T \rangle = \frac{1}{n} \sum_{j=1}^{n} k_j k_j^T$$  \hspace{1cm} (1)

where, $k_j$ is the complex Pauli scattering vector of $j^{th}$ pixel. The basic observable in POLSAR is a complex scattering matrix with coefficients as follows:\cite{observablePolarsar}

Further author information:
Sergio Sanchez Martinez: E-mail: smartinez@masdar.ac.ae
Prashanth Reddy Marpu: E-mail: pmarpu@masdar.ac.ae
Antonio J. Plaza Miguel: E-mail: aplaza@unex.es
\[
S = \begin{bmatrix}
S_{HH} & S_{HV} \\
S_{VH} & S_{VV}
\end{bmatrix} \tag{2}
\]

A subscript HV for example corresponds to the scattering element of horizontal transmitting and vertical receiving polarizations. For unsupervised analysis of POLSAR data, the coherence matrix is decomposed using an appropriate method to characterize the scattering behavior at the surface. One of the most commonly used decomposition algorithms is the Entropy-Alpha-Anisotropy (H/\(\alpha\)/A) decomposition which can be used to cluster the POLSAR data into 16 classes.\(^2\) Calculating H, \(\alpha\) and A requires eigenvalue decomposition of the complex coherence matrix at every pixel. The target decomposition algorithm is framed by using the eigenvalues and eigenvectors of the multi-look coherency matrix.

\[
\langle T \rangle = \lambda_1 e_1 e_1^T + \lambda_2 e_2 e_2^T + \lambda_3 e_3 e_3^T \tag{3}
\]

\(\lambda_i\) and \(e_i\) are eigenvalues and eigenvectors respectively.

Entropy (H) can be defined as:

\[
H = -\sum_{i=1}^{3} P_i \log_2 P_i \tag{4}
\]

where

\[
P_i = \frac{\lambda_i}{\sum_{j=1}^{3} \lambda_j} \tag{5}
\]

The eigenvectors can be mathematically written as:

\[
e_i = e^{j\varphi_i} \begin{bmatrix}
cos\alpha_i \\
\sin\alpha_i \cos\beta_i e^{j\delta_i} \\
\sin\alpha_i \sin\beta_i e^{j\gamma_i}
\end{bmatrix} \tag{6}
\]

where, \(\alpha\) is the variation from surface scattering (0\(^\circ\)) to dipole scattering (45\(^\circ\)) to double bounce scattering (90\(^\circ\)), \(\beta\) is twice the polarization orientation angle, \(\delta\) is the phase difference between the \(S_{HH} + S_{VV}\) and \(S_{HH} - S_{VV}\) terms, \(\gamma\) is the phase difference between the \(S_{HH} + S_{VV}\) and \(S_{HV}\) terms, and \(\phi\) is the phase of the term \(S_{HH} + S_{VV}\). The average alpha angle is defined accordingly as,

\[
\bar{\alpha} = \sum_{i=1}^{l} P_i \alpha_i \tag{7}
\]

Anisotropy is defined as,

\[
A = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} \tag{8}
\]

The values of H and \(\alpha\) are used to define 9 classes indicating various types of scattering mechanisms. The different scattering mechanisms that can be classified are as follows:

- **Z9** : Low Entropy Surface Scatterers (\(H < 0.5; \alpha < 42.5\))
  
  Eg :- scattering surfaces like water, sea ice and very smooth land surfaces

- **Z8** : Low Entropy Dipole Scatterers (\(H < 0.5; 42.5 < \alpha < 47.5\))
  
  Eg :- isolated dipoles

- **Z7** : Low Entropy Multiple Scatterers (\(H < 0.5; \alpha > 47.5\))
  
  Eg :- isolated dielectric and metallic dihedral scatterers
- Z6: Medium Entropy Surface Scattering (0.5 < H < 0.9; α < 40)
  Eg: - oblate spheroidal scatterers (leafs and discs)
- Z5: Medium Entropy Vegetation Scattering (0.5 < H < 0.9; 40 < α < 50)
  Eg: - vegetated surfaces with anisotropic scatterers
- Z4: Medium Entropy Multiple Scattering (0.5 < H < 0.9; α > 50)
  Eg: - forestry, urban areas
- Z2: High Entropy Vegetation Scattering (H > 0.9; 40 < α < 52.5)
  Eg: - forest Canopies, highly anisotropic bushes
- Z1: High Entropy Multiple Scattering (H > 0.9; α > 52.5)
  Eg: - vegetation with well-developed branch and crown structure
- Z3 indicates high entropy surface scattering and is non-existent. Fig.1 shows the boundaries of various zones:

These classes can be further divided into 16 classes by setting a threshold of 0.5 for anisotropy (A).

3. PARALLEL IMPLEMENTATION

The processing chain involves estimation of the complex coherence matrix at every pixel; eigenvalue decomposition of the complex coherence matrix; calculation of H and α values; and clustering of pixels into 8 classes based on these values. The main computational aspect of the processing chain is the eigenvalue decomposition of the complex coherence matrix at every pixel. Since, the complex-coherence matrix is always a 3 × 3 matrix for POLSAR data, we simply use the direct formulas derived by solving the eigenvalue decomposition of a 3 × 3 matrix which reduces to solving a complex cubic polynomial. The pseudocode for the processing chain is shown in Fig.2. We simply divide the execution in p processes executed in parallel where each process deals with a part of the data consisting of approximately n/p pixels.
$n$ → number of pixels
$T_{9\times n}$ → Coherence matrix (each pixel has 9 elements)
$p$ → number of processes
$k$ → number of classes

1. parallel for $i=0$ to $p-1$ do
2. $T_i$ ← Read_Coherency($T$, $i$);
3. $i_{th}$eigvec, eigval ← Compute_Eigenv ($T_i$);
4. $H$, $\alpha_i$ ← Compute_H_alpha( eigvec, eigval);
5. end parallel for

Figure 2. Pseudocode of the parallel implementation of POLSAR target decomposition algorithm.

3.1 Read coherence matrix

The process of loading the matrix $T$ from hard disk to main memory is illustrated in Fig. 3. Arrow indicates how the elements are stored in memory. Values in same pixel are separated $n$ positions in memory. The coherence matrix is a Hermitian matrix, therefore we only need the elements of the upper triangular matrix to be stored. The three diagonal elements are real and the three off-diagonal elements are complex, so we have to store nine values in total to represent the coherence matrix.

Figure 3. Data representation of the coherence matrix in the memory.

We divide the data set into $p$ parts. Each process will read $n/p$ pixels from $T$ and store them in memory as shown in Fig. 4.

Figure 4. Illustration of memory accessed by each process.

3.2 Computation of Eigenvalues and Eigenvectors

As the computation in each pixel is independent from the rest, each process computes the eigenvalues and eigenvectors of all its pixels one by one. Each process will read a pixel from its subset, compute the eigenvalues and eigenvectors and store the results in main memory as shown in Fig. 5.
3.3 Computation of H, α and A
Again the computation of the variables of interest (H, α and A) for each pixel is independent from the rest. Thus, each process will use the eigenvalues and eigenvectors previously computed to compute these variables for all its pixels one by one. The results will be stored in main memory as shown in Fig. 6.

4. HARDWARE SPECIFICATIONS
We use the high performance computing cluster at Extremadura Research Centre for Advanced Technologies (CETA-Ciemat). The cluster is a FERMI cluster which uses 4 BullX R424 E2 (4 nodes per BullX), which is a total of 16 nodes. The specifications of each node are shown in Table 1. A total of 192 cores are available for processing.

| CPU       | 2 x Hexacores Core Intel Xeon 2.53 GHz |
| Memory    | 24 GB SDRAM DDR3 1333MHz ECC          |
| Hard Disk | 1 TB SATA                              |

Table 1. Hardware specification of a node in the CETA-Ciemat cluster.

For the implementation of CPU version we use one core of one node and for the MPI version we use up to 128 cores.

5. RESULTS
The efficiency of the parallel implementation is estimated in terms of the speed-up achieved in comparison with the processing using one core on a single node. The computational time of loading the data in the memory, eigenvalue decomposition, and computation of H, α and A values is recorded separately and speed-ups achieved at all these stages are estimated. To achieve a consistent comparison, all the reported times are calculated as averages of fifty executions of the same processes. To measure the time of a stage using more than one process, we keep the time stamp of all the processes at the beginning and after end, and then we compute the difference between the maximum and the minimum of all the recorded time stamps. The results of the time taken for loading the data in the cluster and the CPU for different configurations is shown in Fig. 7 it can be observed that the no speed up is achieved for small data sets. However, a significant speed-up is achieved for large datasets.

Fig. 8 and 9 show the execution time and speed-ups achieved for the computation of eigenvalues and eigenvectors, and the computation of H, and A respectively. The total execution time for the processing
chain and the speed-ups achieved are shown in Fig. 10. It can be clearly seen that a significant speed-up is achieved in the computation of the eigenvalue problem. The speed-ups are not necessarily equal to number of cores due to the fact that different processes start and end at different time stamps. It can be observed that near-real-time performance is achieved in the cluster even for large datasets. For a dataset of size 4096 × 4096 pixels, the estimated computation time for the entire processing chain is around 0.79 seconds.

6. CONCLUSION AND FUTURE WORK

In this work, we demonstrated that near-real-time performance is achieved for the parallel implementation of the target decomposition algorithm for processing polarimetric synthetic aperture radar data. The algorithm is implemented on parallel architecture using message parsing interface (MPI) framework. In the future, we would like to extend this work to implement the complex-wishart classifier which uses the target decomposition as the initial clustering to perform unsupervised classification of POLSAR data. Further, a GPGPU implementation is also possible with a simple adaptation of the MPI implementation.
ACKNOWLEDGMENTS

This work is supported by the research fund of Masdar Institute of Science and Technology, Abu Dhabi, UAE. We acknowledge the support of the computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF). CETA-CIEMAT belongs to CIEMAT and the Government of Spain.

REFERENCES