Development of Quicklook and Precision Synthetic Aperture Radar Processing System for UAV and Microsatellite Platform using Mobile Heterogeneous Computing

July 2016

Bambang Setiadi

Graduate School of Advanced Integration Science
CHIBA UNIVERSITY

（千葉大学審査学位論文）

Development of Quicklook and Precision Synthetic Aperture Radar Processing System for UAV and Microsatellite Platform using Mobile Heterogeneous Computing

July 2016

Bambang Setiadi

Graduate School of Advanced Integration Science
CHIBA UNIVERSITY

Abstract

Synthetic Aperture Rada (SAR) technology offers high resolution imaging technique for earth observation in all weather condition and all-day operation. SAR sensors usually mounted on spaceborne or airborne platforms and can work in different wavelengths and use various wave polarizations to accommodate diverse remote sensing applications. Nowadays, the availability of lightweight unmanned aerial vehicle (UAV) and microsatellite platforms offered lower cost and risk in SAR sensor development, deployment and operation. However, these lightweight platforms have significant limitations on the payloads size, weight, and power (SWAP).

SAR processing which converts raw data into the two-dimensional image is an important part of the SAR system. Due to SWAP limitations, the usage of compact embedded high-performance computing (HPC) for onboard SAR processing is getting a lot of attentions. However, not many have been focusing on using heterogeneous mobile computing as a feasible solution. In this thesis, we developed onboard quicklook and precision SAR processing systems for UAV and microsatellite using mobile heterogeneous computing platform consisting of an integrated mobile CPU and GPU.

The quicklook SAR processor is designed for L-band CP-SAR sensor onboard future GAIA-II microsatellite and able to generate low-resolution. It is based on modified spectral analysis (SPECAN) algorithm for monostatic stripmap mode SAR image formation. The precision SAR processor which based on Range-Doppler Algorithm (RDA) is intended for generating high-resolution images for L-band linearly polarized SAR sensor onboard the future JX-2 UAV.

To optimize the algorithms, we analyze the details of both algorithms to find parallelism potentials. For a given SAR raw data our model uses (1) data partitioning based on the number of samples in range and azimuth directions to balance the task workload on the underlying hardware (2) identification and separation of SAR processing tasks into coarse and fine-grained parallel tasks.

To evaluate the result, we have implemented the sequential version of both applications on single CPU core as the baseline. The quicklook implementation is validated using actual raw data from JERS-1 SAR sensor that has similar characteristics with GAIA II CP-SAR sensor, and the precision algorithm is validated using publicly available raw data from a UAV based stripmap mode LFMCW SAR sensor. We have evaluated our proposed approach using various size inputs number of samples and bins on a middle-end mobile heterogeneous CPU-GPU development kit. Implementation on an integrated quad-core CPU and 192 cores CPU showed that regardless of the size of range samples and azimuth bin, speedup up to 4.2 and 6.5x can be achieved for quicklook and precision application respectively.

Abstract

(in Japanese)

モバイル異種計算を用いた無人航空機及び小型衛星プラットフォームのクイックルックと精密合成開口レーダ信号処理システムの開発

ヨサファットマイクロ波リモートセンシング研究室は現在,マルチバンドの円偏波合成開口レーダ搭載航空機と小型衛星の研究開発を行っている。

無人航空機と小型衛星にセンサを搭載することで開発と運用中のリスクとコストを低減することができる。しかし、これらのプラットフォームにはサイズや重量やセンサーペイロードとパワー（SWAP）などの様々な制限がある。

SWAP に制限があるため、オンボード処理には従来の HPC を使用することが困難であり、代わりに DSP、ASIC や FPGA などの高性能組込みコンピューティングシステム（HPEC）を使うのが解決策となる。

モバイル異種計算プラットフォームは SAR 処理を加速する機会を持っている。

この論文では、マルチコア CPU と多くのコア GPU からなる移動異種の計算プラットフォームを使用して、無人航空機や小型衛星のアプリケーションのためのクイックルックと精密合成開口レーダ信号処理システムを開発した。

与えられた SAR 生データに対し、我々のパフォーマンスモデルは（1）CPU と GPU の性能特性を用い、（2）順次 SAR 処理タスクの識別と粗ときめの細かい並列タスクへの分離（3）レンジ方向のサンプル数とアジマスビン数を用い、基盤となるハードウェア上の処理負荷を分散させることができる。

クアッドコア CPU と 192 コア CPU 上での実装がレンジサンプルおよびアジマスビンのサイズに関係なく、クイックルックと精度アプリケーションに比べ、シーケンシャルのバージョンがそれぞれの 4.2 と 6.5 倍に高速化することが示された。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objectives

Josaphat Microwave Remote Sensing Laboratory is carrying out research and development of Circularly Polarized Synthetic Aperture Radar (CP-SAR) sensors in various electromagnetic wave bands (L,C and X band). The SAR sensors will be deployed on UAV and micro satellite platforms, because the development and operations of sensors in these platforms have lower cost, lower risk and agile compare to the conventional aircrafts and satellites. However, lightweight UAV and microsatellite have several major limitations related to the sensor payload: size, weight and power (SWAP) [1, 2].

Miniaturization of SAR sensor system is an answer to the challenge of SWAP on lightweight SAR platforms. Currently there are efforts to develop smaller SAR sensor subsystems including smaller antennas, radio frequency (RF) subsystem, signal generators and raw data processing. Onboard raw SAR sensor data processing (or on-board SAR processing) as one of key part of SAR sensor system is also significantly influenced by this challenge [3, 4, 5].

The SAR processing subsystems main objective is to convert raw data produced by sensors into two dimensional image. It is usually done digitally using high performance computing (HPC) system due to complex computations and large amount of data. Desktop computers with CPU and GPU, clusters of networked computers and super computers constitute the vast majority of hardware platforms used for processing raw SAR data. Due to SWAP limitations, it is difficult to use conventional HPC for SAR processing onboard UAV and microsatellite platforms. Consequently, making the usage of high performance embedded computing systems (HPEC) such as special purpose digital signal processors (DSPs), application-specific integrated circuits (ASIC), field programmable gate array (FPGA) and lightweight single board personal computers as suitable solution candidates for this problem [6, 7].

General purpose computing on graphics processing units (GPGPU) and heterogeneous computing has recently gained considerable attention in SAR processing domain. GPGPU is based on using the GPU as a co-processor accelerator to offload computationally intensive tasks from the CPU, whereas heterogeneous computing is a term that refer to the practice of using more than one type of processors or cores for computation. In this case GPGPU computing can also be referred to as heterogeneous CPU/GPU computing [6].

Despite the facts that heterogeneous computing platforms such as desktops and servers with CPU and GPU are widely available and already used for SAR processing, minor approach has been done to use the mobile version of the heterogeneous computing platform for onboard SAR processing. Mobile heterogeneous computing platforms which integrates energy efficient CPU and GPU, provides opportunity to accelerate SAR processing onboard lightweight platforms. However, mapping SAR processing algorithm to mobile heterogeneous platform involves many design decisions to make and is not a direct task [8, 9].

In this thesis we developed on-board quicklook and precision SAR processing systems for UAV and microsatellite applications using mobile heterogeneous computing platform consisting of a multicore CPU and many core GPU. The quicklook processing represents the case where low-resolution imagery is required especially during SAR data acquisition and raw data preview process. On the other hand, the precision SAR processing represents the case where producing high resolution imagery is the main users objective.

The quicklook processor application is designed for L-band CP-SAR sensor onboard future GAIA-II microsatellite and developed to produce low-resolution images with additional function of enabling rapid preview of CP-SAR imageries. It is based on modified spectral analysis (SPECAN) algorithm for monostatic stripmap mode SAR image formation.

The precision processor application is designed for L-band linearly polarized SAR sensor onboard the future JX-2 UAV and developed to produce high-resolution imageries for advanced applications such as interferometry and polarimetry. It is based on Range Doppler Algorithm which consists of compressions in range and azimuth direction with range cell migration correction process in between.

To optimize the algorithm on mobile heterogeneous platform, we use an offline profiling step to determine the performance of a systems CPU and GPUs with respect to SPECAN and RDA algorithms. For a given SAR raw data our performance model uses (1) the CPU and GPU performance characteristics (2) identification and separation of sequential SAR processing tasks into coarse and fine grained parallel tasks (3) the number of samples in range directions and number of azimuth bins to balance the processing workload on the underlying hardware. Speed-ups is achieved through series of assessment, parallelization, optimization and deployment (APOD) design cycle. The run-time partitioning scheme exploits task, data and pipeline parallelism by sharing the parallelizable task between the CPU and GPU and assigning non parallelizable computation on the CPU.

1.2 Contributions

The specific contributions of this research are summarised as follows:

- We have designed and developed quicklook SAR processing system based on modified SPECAN algorithm for use on-board future GAIA-II CP-SAR satellite space-borne platform.
- We have designed and developed precision SAR processing system based on Range Doppler algorithm for use on-board future JX-2 CP-SAR airborne platform.
- We have implemented and tested the quicklook SAR processing system based on modified SPECAN algorithm for use on-board future JX-2 CP-SAR airborne platform on mobile heterogeneous computing hardware as an effort to overcome the SWAP limitation.
- We have implemented and tested the quicklook SAR processing system based on Range-Doppler algorithm for use on-board future JX-2 CP-

SAR airborne platform on mobile heterogeneous computing hardware as an effort to overcome the SWAP limitation.

- We formulated a method to balance the load of SPECAN algorithm between CPU and GPU on mobile heterogeneous platform in order to optimize execution.
- We formulated a method to balance the load of RDA algorithm between CPU and GPU on mobile heterogeneous platform in order to optimize execution.

1.3 Outline

This thesis is organized as follows:

- In Chapter 2 provides the background information and context necessary to understand the material presented in the subsequent chapters. This includes a discussion of CP-SAR hardware development and SAR processing software development in JMRSL. We also present background information on SPECAN and Range Doppler Algorithms.
- In Chapter 3 presents the a modified SPECAN algorithm for satellite borne and RDA for airborne SAR sensor. Our SPECAN and RDA kernels for GPU are also presented.
- In Chapter 4 describe the experimental setup, performance model and dynamic partitioning for CPU and GPU system.
- In Chapter 5 presents the conclusion and further research.

Chapter 2

Background and Related Works

2.1 CP-SAR Development at JMRSL

Synthetic Aperture Radar (SAR) is a well-known remote sensing technology that offers the possibility of earth observation under all weather conditions with day and night time operation [10], [11]. Most of SAR sensors for earth remote sensing applications available today are linear polarized sensors which send and receive horizontal or vertical wave polarization. Josaphat Microwave Remote Sensing Laboratory (JMRSL) of Center for Environmental Remote Sensing, Chiba University, has been working on the development of circularly polarized SAR (CP-SAR) sensors to explore the usage of elliptical polarization wave for SAR applications. The objective of using elliptical polarization wave is to explore various possibilities offered by circularly polarized scattering phenomena. The future CP-SAR sensors systems will be deployed on a small UAV and a microsatellite.

Table 2.1 summarizes various basic experiments and applications of CP-SAR

Table 2.1: CP-SAR Objectives

Field	Items	Details
Basic SAR Experiment	Scattering mechanism of CP microwave	studying scattering mechanism from vegetation, cryosphere, soil and rocks, desert, etc.
	Interferometry	LP vs CP interferometric SAR, DEM generation
	Axial ratio image	Vegetation, geologic, cryosphere mapping
SAR Applications	Land cover mapping	Forest/non-forest area classification Paddy field/wet land extraction Mangrove area mapping Snow - iceberg detection
	Disaster monitoring	Earthquake, Volcano eruption, Flood, forest fire, landslides, etc.
	Cryosphere monitoring	Iceberg and glacier, Arctic routing
	Ocean monitoring	Oil spill, ocean wave

[2]. Although many of the applications listed in Table 2.1 require the deployment of satellite CP-SAR sensors, it would be a reasonable way to investigate the CP-SAR capability using an unmanned aerial vehicle (UAV) as a testing platform. Recently UAV has gained more popularity as a preferable choice for SAR sensor platform due to its properties such as relatively easier and agile mission deployment, lower implementation cost and lower risk when compared with conventional airborne or space-borne systems [1, 12].

Several concurrent efforts for the development of CP-SAR sensor systems are taking place at JMRSL: antenna, RF system, signal generator, data communications and SAR data processing. This thesis focused on a part of SAR data processing task: development of on-board SAR processing system for the UAV and microsatellite based sensor. The following sections describe summary of CP-SAR sensor system development at JMRSL, followed by literature research on the field of on-board SAR processing with concentration on the usage of heterogeneous computing platform.

2.1.1 Development of CP-SAR System for UAV

Figure 2.1 shows a small UAV called Josaphat laboratory experimental UAV (JX-2) which will become the platform for L-band SAR sensor system under development. During flight, it is capable of having a ground speed of approximately 30-40 m/s with operating altitude of 1-4 km and flight endurance up to 4 hours. The UAV has wing span of 6 m, body length of 4.75 m and total weight of 146 kg including 25 kg payload, i.e. the SAR sensor, ground positioning system (GPS), inertial measurement unit (IMU) and data communication link.

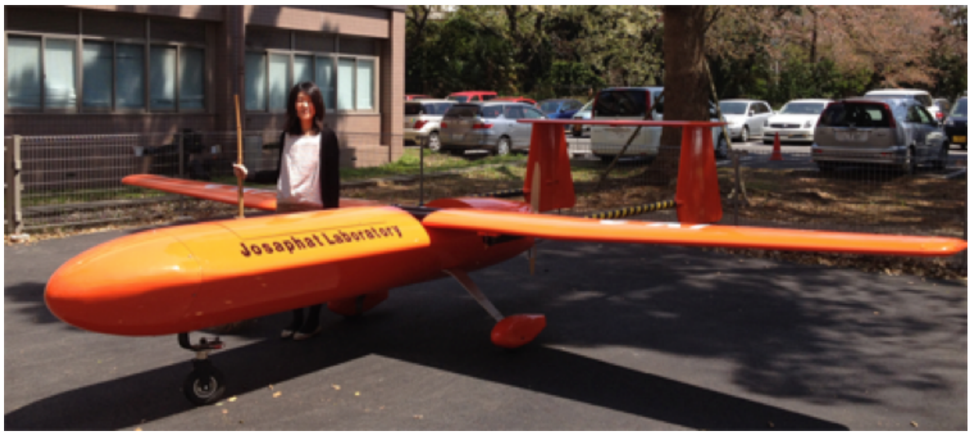


Figure 2.1: JX-2 Unmanned Aerial Vehicle (UAV)

The CP-SAR sensor system is composed of a chirp signal generator, RF Module, CP antenna, analog to digital converter/ data recorder, SAR processor, motion sensing module and data downlink module. The CP antenna is connected to the RF module, while the motion sensing module contains an inertial measurement unit (IMU) that receives position signals from GPS antenna. In order to communicate with the ground station, a data link antenna is connected to the data downlink module. Figure 2.3 shows the antenna developed by Yohandri et.al. [13].

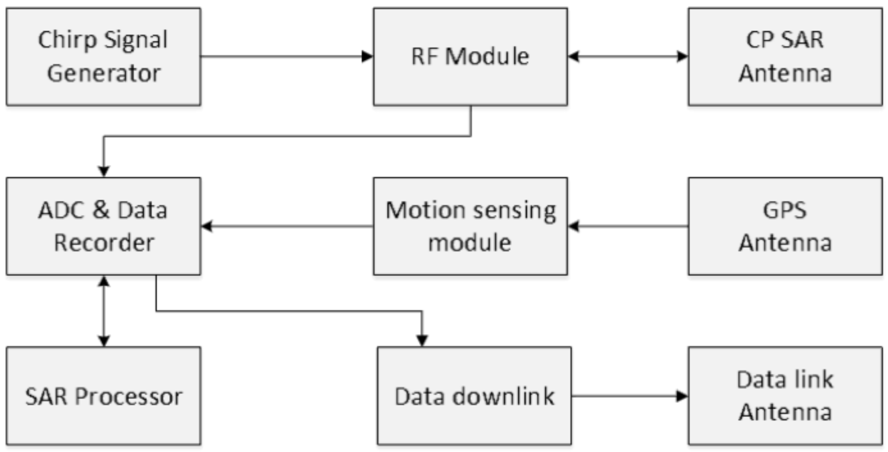


Figure 2.2: CP-SAR functional block diagram

The RF subsystem consists of a transmitter, receiver, local oscillator and low noise amplifier with total weight of approximately 10 kg. The transmitter mixes the 150 MHz bandwidth chirp up to 1.27 GHz for transmission. The receiver and local oscillator are used to mix the RF radar return from the antenna to an offset baseband and amplify it so that it can be sampled by the ADC subsystem.

The signal generator subsystem generates chirp signal as the input for RF subsystem. It is implemented using a field programmable gate array (FPGA) to enable rapid modification to the generated chirp signal. The chirp signal generator must be able to generate chirp signal with various configurations depending on the mission requirement such as flight height and platform speed.

The parameters of generated chirp signal that can be modified are: chirp bandwidth, pulse duration and pulse repetition frequency (PRF). Direct digital synthesizer (DDS) technique is used for the chirp signal generation process

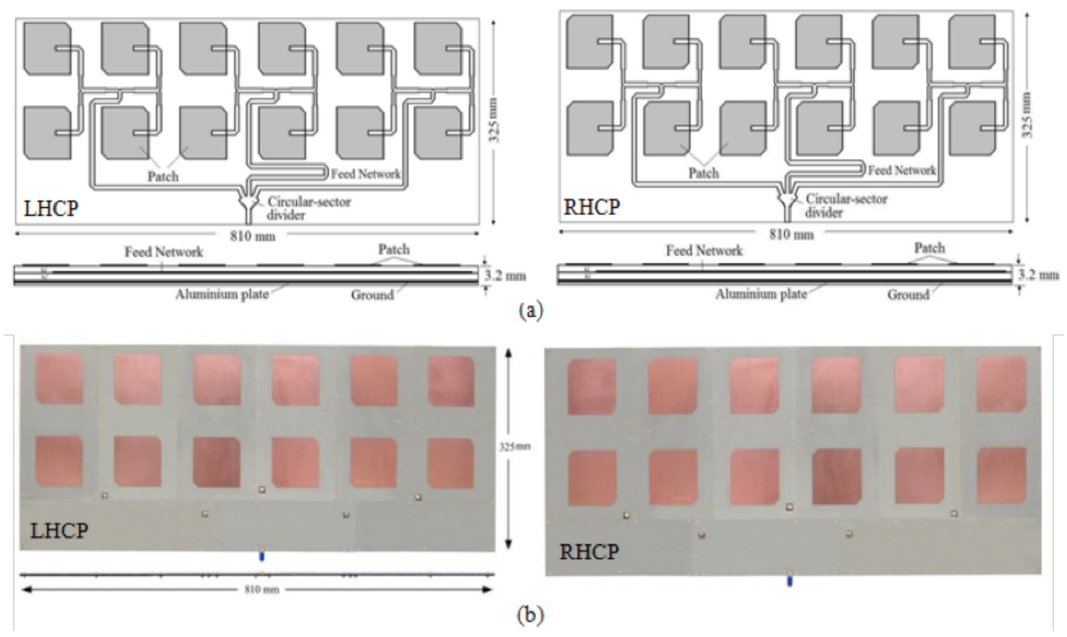


Figure 2.3: CP-SAR Antennas

because it has many advantages compare to its analog counterpart. Figure 2.4 shows an example of real and imaginary components of the chirp signal created by the signal generator subsystem developed by Suto et.al [14].

The analog to digital converter/data recorder subsystem handles the sampling and down-converting of signal received by the CP antenna. After down-converting to the baseband, the data are stored in a solid-state drive storage. This data is known as a raw SAR data or phase histories, and will be processed further using signal processing technique in order to produce a SAR image. shows the photographs of chirp generator subsystem, analog to digital converter (ADC) and data recording subsystem, and the SAR processor on a compact single PC board.

The processing was previously done using Range Doppler algorithm implemented in MATLAB scripting language in order to accelerate the development process and lower the learning curve. We also have implemented several other

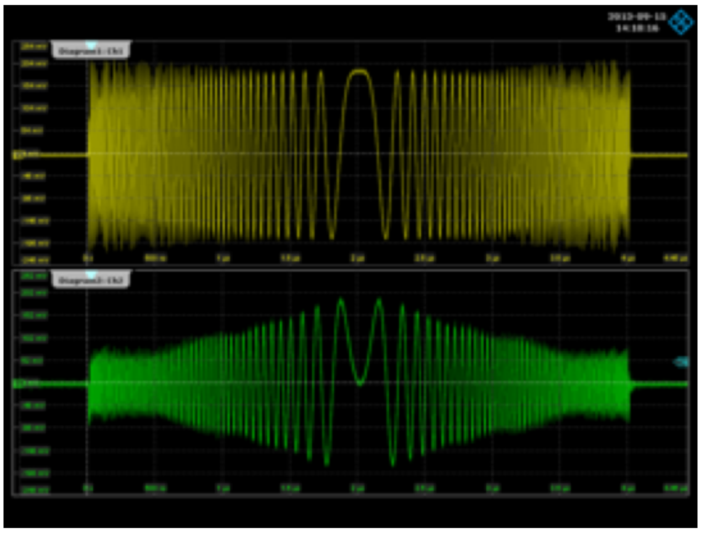


Figure 2.4: Chirp signals

algorithms such as Chirp Scaling and Spectral Analysis as part of the development [15, 16, 17, 18]. Details on the SAR processing is presented in section 2.1.3 respectively.

The processing of raw sensor data into full resolution image is usually carried out in the ground station because it involves complex computation and large amount of memory which requires high performance computing (HPC) facility. It is difficult to do onboard the platform due to size, weight and power constraints. But recent developments in computing has made this task suitable for desktop computing, although real-time processing of SAR data without HPC is still under intensive research [6].

2.1.2 Development of CP-SAR system for Microsatellite

As part of the development roadmap, besides using airborne platform, JMRSL will also deploy the L-band CP-SAR sensors on spaceborne platform which will

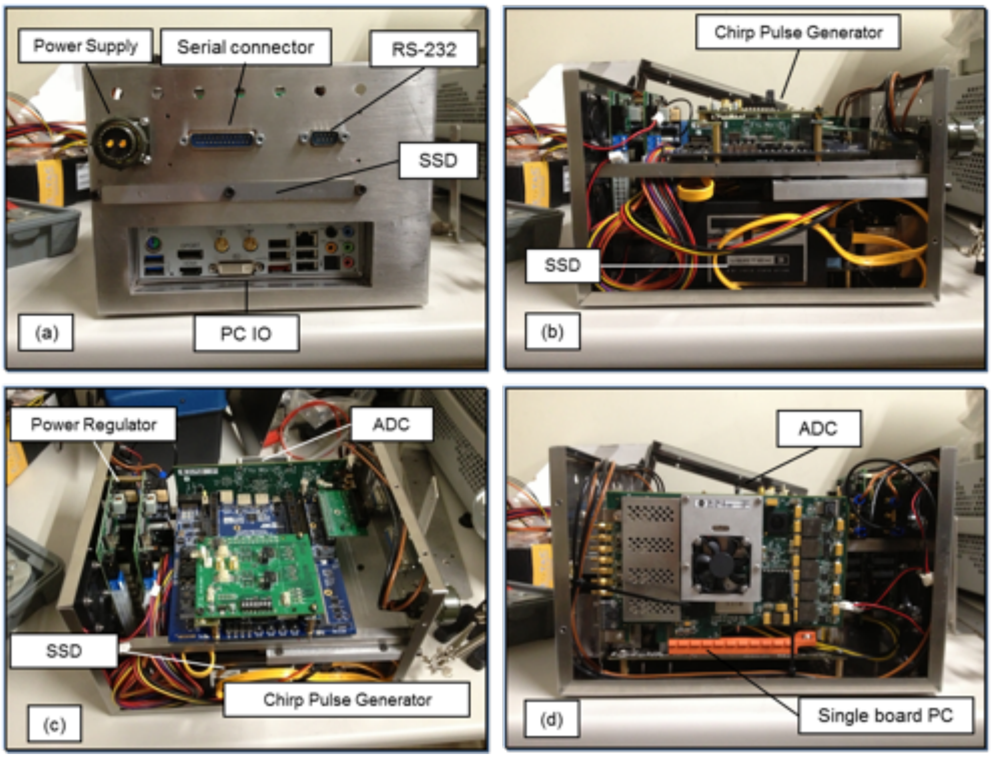


Figure 2.5: Analog to Digital Converter and Chirp Signal Generator Hardware (a) Front side view. (b) Left side of view (c) Top View (d) Right View

be a microsatellite called GAIA-II [19]. An illustration of the CP-SAR onboard the GAIA-II is depicted in Figure 2.6. CP-SAR sensor onboard the satellite is utilizing the elliptical wave propagation and scattering phenomenon by radiating and receiving the elliptically polarized wave, which includes circular and linear polarizations. Elliptical polarized sensor is considered not depend on the platform posture, and able to lower the effect of Faraday rotation in Ionosphere.

The SAR sensor transmits and receives L-band chirp pulses with PRF between 2,000 to 2,500 Hz. It is designed as a low cost, light, low power, low profile sensor that transmits and receives left-handed circular polarization (LHCP) and right-handed circular polarization (RHCP). Besides producing single look complex (SLC) image and amplitude image, this sensor can also generate var-

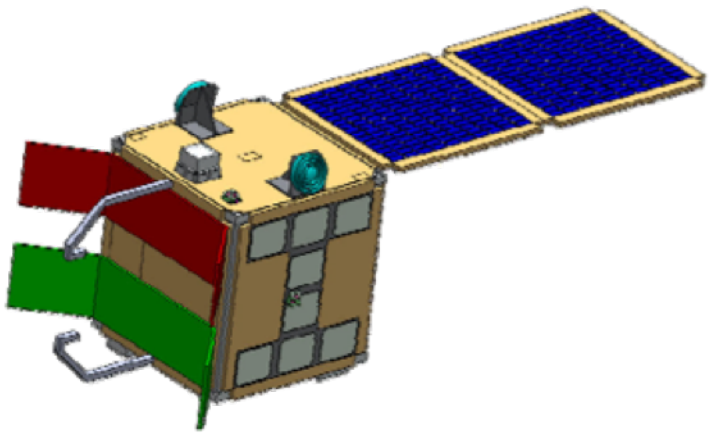


Figure 2.6: CP-SAR sensor onboard GAIA-II microsatellite

ious SAR images unique to elliptically polarized sensor e.g. axial ratio image (ARI), ellipticity and tilt angle [19, 20].

GAIA-II satellite will orbit between 500 to 700 km, with 50 km swath width and maximum spatial resolution of 30 m. The L-band (1.27 GHz) antenna size is 2.0 m in elevation and 5.0 in azimuth direction, with 97.6 inclination angle and 29.0 off nadir angle. The satellite has the form of 1m cube with maximum weight of 100 kg.

Table 2.2 summarizes the specification of CP-SAR onboard GAIA-II [20].

To withstand the space environment, Osanai et.al. has developed a 1.275 GHz Right Handed Circularly Polarized (RHCP) and Left Handed Circularly Polarized (LHCP) SAR antenna using Kevlar honeycomb core which have the advantage of lightweight and heat and shock resistance [21]. The antenna has the size of 267 mm by 267 mm with the radiating size of 101 mm by 101 mm, and bandwidth values for RHCP and LHCP of 90 MHz and 180 MHz respectively (Figure 2.7).

A functional block diagram of the CP-SAR system onboard GAIA II satellite and SAR processor on the ground station is shown in Figure 2.8. The space-

Table 2.2: CP-SAR Sensor Onboard GAIA II Specification

Parameter	Value
Altitude	500 - 700 km
Inclination angle	97.6°
Frequency	1.27 GHz
Polarization	LL, LR, RL, RR
Gain/Axial Ratio	>30 dBic / <3 dB
Off nadir angle	29°
Swath width	50 km
Spatial resolution	30 m
Peak power	90-300 W
Bandwidth	10 MHz
PRF	2000-2500 Hz
Platform size	1 m $\times$ 1 m $\times$ 1 m
Weight	100 kg
Antenna size	2.0 m x 5.0 m

borne SAR sensor system diagram is similar with the UAV based airborne sensor system. Both system have the same signal generator, RF, antenna, ADC & Data Recorder, Motion Sensing, GPS and communication modules.

The data downlink module will transmit received SAR signal to the ground station via data link antenna. The CP antenna is connected to the RF module, while the motion sensing module contains an inertial measurement unit (IMU) that receives position signals from GPS antenna. In order to communicate with the ground station, a data link antenna is connected to the data downlink module.

In current design, due to stricter size and power limitation the SAR processing subsystem is done on the ground station, consequently SAR processor is not installed onboard the satellite.

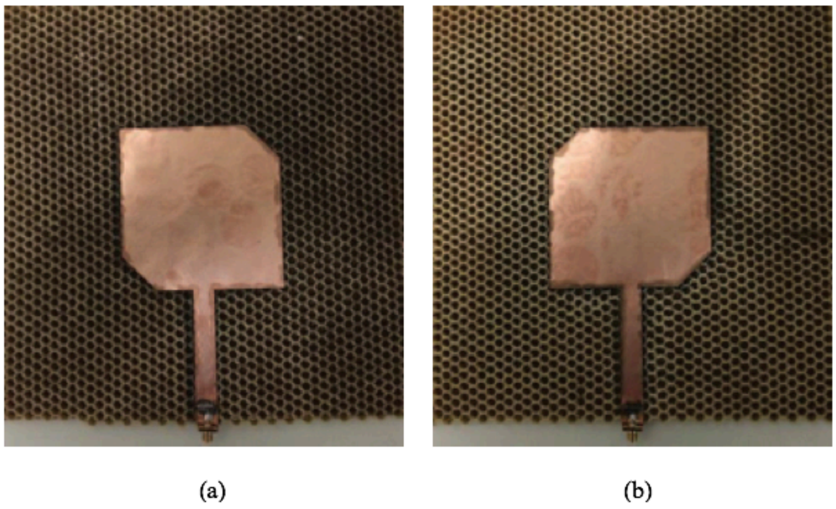


Figure 2.7: (a) RHCP and (b) LHCP Antenna for GAIA-II

2.1.3 Development of SAR Processing System

Besides the radar subsystem that generates, transmits, receives and records the backscattered signals, another important component of a SAR system is module that converts the SAR raw data into two dimensional images called SAR processor. Algorithm that is used for SAR processor is also known as SAR image formation algorithm. Currently there are various SAR processing algorithms available and some of the most commonly used are: Range Doppler, Chirp Scaling, SPECAN and Omega-K [11, 22, 23].

According to the types of processing, there are three types of processing can be performed to the raw data: survey processing (quicklook), Precision processing and ScanSAR processing. Different with precision processing that aim to produce high resolution image, quicklook processing objective is to produce reduced-resolution imagery of a full square scene with reasonable speed [24]. Although most SAR application requires high-resolution image, there are cases where low-resolution is also important, for example when browsing from large collection of raw data to find the area of interest.

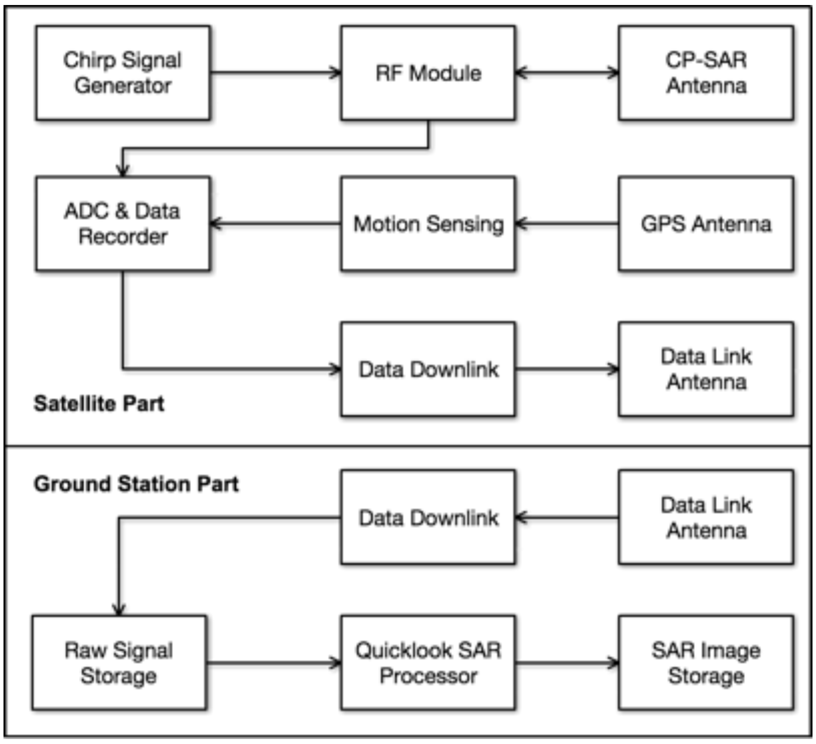


Figure 2.8: Functional block diagram for CP-SAR sensor onboard GAIA-II

Because of the complex computations requirements and large amount of raw data produced by SAR sensors, SAR processing is usually done digitally using high performance computing (HPC) system. Processing of raw SAR data is done using desktop computers with CPU and GPU, clusters of networked computers and super.

The development of SAR processor software in JMRSL was started with the development of JERS-1 SAR raw data processor based on Range Doppler Algorithm (RDA) using MATLAB programming language [17, 18]. MATLAB was used due to the availability of vast mathematical, signal processing and image processing libraries. It also has user friendly development environment that support the write, test and debug software development cycle.

The effort was continued by development of SAR processors using other algorithms i.e. Chirp Scaling and Spectral Analysis (SPECAN). To accelerate the processing for existing satellite borne SAR, there was also efforts to develop the GPU version of RDA and SPECAN algorithms [16, 17, 18, 25].

To support the spaceborne SAR development that will be mounted on GAIA-II microsatellite, a SPECAN algorithm based quicklook processor has also been designed and developed. This quick-look SAR processor is designed for L-band CP-SAR sensor onboard GAIA-II microsatellite, which will produce low-resolution images and enable rapid preview of CP-SAR images on commodity of the shelf computer system. The algorithm is based on modified spectral analysis (SPECAN) algorithm for SAR image formation. The result is validated using actual raw data from JERS-1 SAR sensor that has similar characteristics with GAIA-II CP SAR sensor [26].

Due to SWAP limitations, it is difficult to use conventional HPC for onboard processing, thus making the usage of high performance embedded computing systems (HPEC) such as special purpose digital signal processors, ASICS and FPGA as alternative solutions for this problem.

Besides the development of SAR processing system on the ground station, there also development of the onboard version of the system. The initial effort of this onboard processor development in JMRSL was marked by the development of FPGA based FFT computation for SAR sensor onboard Microsatellite [15]. Further development was continued by Panggabean et.al. with the development of A Single-on-Chip for Onboard SAR Imaging Processor Based on the LEON3 soft-core processor implemented on FPGA [27].

Despite the facts that heterogeneous computing platforms such as desktops and servers with CPU and GPU are widely available and already used for SAR processing, minor approach has been done to use mobile heterogeneous computing platform for onboard SAR processing. Mobile heterogeneous computing platforms which integrates CPU and GPU provides opportunity to accelerate SAR processing application. However, mapping the complex processing algorithm to mobile heterogeneous platform is not a direct task with many design decisions.

In this thesis we developed quicklook and precision SAR processing systems for UAV and microsatellite applications using mobile heterogeneous computing platform consisting of a multicore CPU and many core GPU. The quicklook processing represents the case where low-resolution imagery is required during SAR data acquisition and the precision processing represents the case where producing high resolution imagery is the main objective.

The quicklook processor application is designed for L-band CP-SAR sensor onboard future GAIA-II microsatellite and developed to produce low-resolution images with additional function of enabling rapid preview of CP-SAR imageries. It is based on modified spectral analysis (SPECAN) algorithm for monostatic stripmap mode SAR image formation. Details will be presented in the following chapters.

2.2 Parallelism

The trend of multi-core architectures has pushed the exploration and exploitation of its benefits by finding obvious and hidden parallelism inside computations. A software may have tasks that can inherently run in parallel or redesigned to do so. The main guidelines of parallelism is to run computational intensive tasks, which takes a lot of time, in parallel. More and more legacy application are being redesigned to exploit this possibility. However, The level of parallelism that can be exploited has limitations.

2.2.1 Speedup and Amdahl's Law

The speed of a program is the time it takes for a program to execute which can be measured using any increment of time. Speedup is the ratio between the time to run a program sequentially (on single processor) and the time to run a program in multiple processor:

$$S = \frac{T(1)}{T(j)}, \tag{2.1}$$

where $T(j)$ is the time it takes to run the program on j processors. Efficiency is speedup divided by the number of processors which is an important factor due to the cost of the additional processors.

The maximum speedup that a program can achieve when executed on multiple processor is dictated by Amdahl's law [28]. It states that, if s is the sequential part of the program and $1-s$ is the part that can be parallelized, the maximum expected speedup S when using P number of processors can be defined as:

$$S(P) = \frac{1}{s + \frac{1-s}{P}} \tag{2.2}$$

When the number of processor is unlimited, the maximum speedup is limited by the sequential part of the program. Which means that in order to get the best speedup, we have to parallelize the part of the program that takes most of the execution time.

However, Amdahl's Law is a generalized argument which has been re-evaluated over the years to adapt to current multicore era [29, 30]. Various models also have been introduced based on scalable computing where problem size can increase due to increasing computing power.

2.2.2 Types of Parallelism

According on how often the subtasks inside a computation need to synchronize or communicate with each other, parallelism inside an application or computation can be categorized into:

- Fine grained, if the subtasks must communicate many times per second. Fine-grained parallelism is typically in the form of loop level which can be done incrementally one loop at a time. It does not require deep knowledge of the code. To get decent speedup, a lot of loops have to be parallelize. And, finally has potentially many synchronization points at the end of each parallel loop.
- Coarse grained, if the subtasks do not communicate many times per second. Coarse grained parallelism make larger loops parallel at higher call-tree level potentially in-closing of many small loops. It has more parallel code at once but requires fewer synchronization points which can reduce overhead. However, coarse grained parallelism requires depeeder knowledge of the underlying code.
- Embarrassingly parallel, if the subtasks rarely communicate. Embarrassingly parallel applications are considered the easiest to parallelize.

2.2.3 Parallel Computation Models

Parallel computation models are needed to structure the design process of parallel programs. Conceptually parallel program can be modelled in one the following categories [31]:

- Farmer-Worker/Master-Slave: the workload can be distributed among algorithmically identical processes, each one taking different input arguments than the other. Usually, a Farmer thread or process is responsible into splitting and assigning work among the Worker threads/processes. The idle Worker threads are usually part of a thread pool, where the Farmer thread checks for available threads to assign work. When a Worker thread finishes its work, it becomes again part of the thread pool waiting for further assignments.
- Divide and conquer: the problem can be divided recursively in smaller problems until they become small enough to be fast to solve. Finally, all the partial solutions are combined to solve the initial problem. It is important to point out that each sub-problem is theoretically independent of the others, which makes concurrent or parallel execution feasible.
- Data parallelism: computation follows the data. The data are distributed in different computing nodes each one performing the same task (executing the same code) on its own data. When finished, the data are usually gathered on a single node forming the final solution.
- Task parallelism: in this model the basic tasks/function can be isolated, each one being responsible for solving a specific sub-problem. Each task acts on specific arguments providing specific outputs. The dependency

between these tasks can be represented using an acyclic directed graph. The way these tasks can be mapped onto a structure of processors is not trivial, thus two types of execution models can be concluded: data-driven and demanddriven. In the data-driven execution model, a task proceeds to execution when the input arguments are available, while in the demand-driven, execution of a task occurs when the provided data are required further in the pipeline.

- Bulk-synchronous: the problem is solved through an iterative process, until solution convergence. Every iteration consists of three distinct steps: a computation step, a communication step and finally a synchronisation step. Each process performs the computation steps asynchronously with respect to the other processes. When computations end, processes can exchange data if required. Finally, for consistency, a synchronisation step follows where all individual processes reach a certain point before proceeding further.
- Hybrid: combination of the above models.

2.3 Heterogeneous Computing

The need for specific domain computing has led to the development of hardware diversity on single system. Heterogeneous computing refers to systems that utilize more than one type of processors in order to gain better computing and energy efficiency. The objectives are to get the best performance from different types of processors, including specialized kinds of processor for specific tasks. It includes both serial and parallel processing and creates a challenge for application developer on how to map the available tasks into the best processing device available on the system. As an example, in a heterogeneous CPU and GPU computing system, GPU have graphics rendering capabilities and also able to perform mathematically intensive computations on very large data sets, while the CPUs can run the operating system and perform traditional serial tasks [32, 33, 34].

This section describes the most important hardware categories that can be found nowadays in computing infrastructures that can also be found inside a heterogeneous computing platform.

2.3.1 General Purpose Processors (GPP)

This category includes all conventional and general purpose computer architectures which usually comprise of one or more same CPU cores. GPP have shared-memory architectures with layers of cache hierarchy and also supports complex performance oriented hardware level optimisations such as branch prediction and dynamic execution. They are mostly used as stand-alone processors inside a computer, however in newer designs, have also been designed as high performance coprocessors, such as Knights Corner and Knights Landing architecture (brand name Xeon Phi) from Intel. This category of hardware targets most of the market spectrum, from mobile devices to servers, and are offered by many industrial vendors like Intel, AMD, IBM and ARM.

2.3.2 Digital Signal Processors

In modern applications, almost every application involved the processing of digital signals. To improve the speed of such applications a specialized DSP processors were designed in order to be able to manipulate digitals signals efficiently. A DSP processors architecture is different in many ways compare to other processors. For example, DSP processor has a signal processing oriented Instruction Set Architecture (ISA), which enables easier and faster implementation of digital signal processing applications.

As required from these algorithms they perform a significant amount of operations in parallel per work cycle. Such operations can be Multiply-accumulate (MAC) operations or Fast Fourier Transform (FFT) loops. Moreover, in order to provide multiple operations per cycle and low power at the same time, they operate in lower frequencies. Example companies designing DSP architectures are Texas Instruments, NXP and Freescale.

2.3.3 GPGPU

Besides graphics rendering, a graphics processing unit (GPU) can also perform general computations that would usually be conducted by the CPU and referred to as General Purpose GPU (GPGPU).

Nowadays GPGPUs are used for various high-powered CPU domain tasks such as physics calculations, encryption/decryption, scientific computations and the generation of cryptocurrencies. By remodeling the problem to suit graphics calculations, the massive parallelism capability of GPU can compete with the computation rate of most powerful CPUs for various parallel processing tasks.

The same graphics shader cores which render allow multiple pixels simultaneously can also be used process multiple data at the same time. Although a shader core is simpler than a CPU, a high-end GPU may consist of thousands of shader cores compare to a multicore CPU, which might only have eight or twelve cores.

After DirectX 10 included unified shaders in its shader core specifications for Windows Vista, focus on GPGPU computing has been increased. Development high-level programming languages and API to ease GPU programming has been taking places, especially by major GPU producers such as AMD and NVIDIA. Two major GPGPU APIs are OpenCL and CUDA.

To enhance the computational capabilities of GPU simple computing cores then added, making it ALU (arithmetic logic unit) heavy. GPGPU suitable for applications with high arithmetic intensity, large input data sets and minimal dependencies between data elements during processing. High arithmetic intensity means that the ratio value between numbers of operations and data transfer during operations is high.

GPGPU is designed to have a complex memory-hierarchy which consist of on-chip memories, with multiple cache layers, and off-chip memories. Currently most GPUs are as accelerators of the main CPU, however integrated graphics solutions also exist. For examples Intels new processor designs (Sandy Bridge, Ivy Bridge, Haswell) and ARMs Mali T60x architecture integrates GPUs and GPPs into a signle SoC and share the same physical memory space.

Example companies that provide GPUs are NVIDIA, AMD, Intel, ARM, Imagination Technologies and Qualcomm.

2.3.4 Field Programmable Gate Array (FPGA)

Hardware based solutions offers the best inherent performance in terms of execution speed, Unfortunately, designing application specific integreated circuit

(ASIC) for every application is time consuming and not flexible.

FPGAs consist of reconfigurable logic blocks connected with each other with programmable interconnects. Although the performance is lower compared to a custom made ASIC solution, but the reprogrammability makes it more flexible compare to ASIC. FPGAs offer high speed I/Os operation and high data buses size which make them appropriate for real-time or high volume applications.

Example companies that provide FPGAs are Achronix, Altera, Atmel and XILINX.

2.4 Parallel Programming Languages

2.4.1 OpenCL

Open Computing Language (OpenCL) is a programming framework for developing software that runs on platforms consisting of CPUs, GPUs, DSPs, FPGAs and other processors or hardware accelerators in a single platform [35].

OpenCL project was pioneered by Apple Inc. which then transfer it to non-profit Kronos Group since 2008. It is supported by major companies such as Intel, NVIDIA, ARM, AMD and Qualcomm. Each of these providers release their own OpenCL implementation for their hardware platform.

The OpenCL platform uses a model where single host connected to one or more compute devices. Each of this compute device consists of one or more compute units that made of one or more processing elements. Finally, each processing element executes code as Single Instruction Multiple Data (SIMD) or Single Program Multiple Data (SPMD).

An OpenCL program consists of the host and the device code. The host code is responsible for initialization, communication, and execution of the kernels among the compute devices. The device code is a kernel or a collection of kernels running on the compute devices written in a data/task parallel fashion. An index space with varying dimensionality from one to three is used to process the data. Each element of this index space is a work-item, and a collection of work-items is called work-group. Each workgroup is independent of any other one, and multiple work-groups can run in parallel. Thus, work-groups should not directly share data, but work-items of a work-group can communicate and synchronize [36].

2.4.2 CUDA

NVIDIA corp develop Common Device Unified Architecture (CUDA) to make it easier for programmer to use its GPUs. It was released in 2007 and the latest stable version is 7.5 in 2016. Its popularity among academia and industry has boost the implementation of various compute intensive applications to GPU platform.

CUDA has similar programming models with OpenCL, it assumes a single host with one or more GPU capable devices. The host is responsible for communications and synchronizations between CPU and GPU, and for the invocations of kernels that run on GPU. The host code is responsible for coordination

CUDA organize threads on GPU into blocks called threadblocks, and threadblocks are further grouped into grid. CUDA memory system has two types of physical address spaces: an off-chip DRAM and an on-chip memory. There are four different memory spaces: the global memory, the constant memory, the local memory and shared memory.

When targetting NVIDIA GPUs CUDA is a major solution because it has programmer friendly extension to existing programming language, making parallel applications the implementation simpler for those who already have a clear understanding of the programming model.

GPU programming optimizations may require deep understanding of both the application domain and the underlying hardware used which depends on the capabilities of each programmer. Furthermore, CUDA comes with pre-optimized libraries for different application domains, such as linear algebra (CUBLAS), signal processing (CUFFT), neural networks (CUDNN), image processing (NPP) and various other libraries.

2.4.3 OpenMP

OpenMP (Open Multi-Processing) is an application programming framework that supports shared memory multiprocessing programming in various programming languages, supporting different processor architectures and operating systems. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior that hides the implementation details from the programmers.

OpenMP uses the fork-join parallelization model, where a master thread is split into a pre-specified number of threads which share the tasks among them. Once forked, the threads run concurrently and depending on the nature of the parallelized application synchronization or communication between threads is required which controlled using pragmas.

2.4.4 MPI

Message Passing Interface (MPI) is a standardized and portable message-passing system to function on a wide variety of parallel computers. The message passing paradigm enabled programming of parallel applications on distributed machines with separate memory spaces.

Although initially targeted only for distributed memory architectures, due to the trend of network connected multi-core processors, MPI then also supported this distributed and shared memory (hybrid architectures). There exists different implementations of MPI such as MPICH, LAM-MPI and OpenMPI.

An MPI program starts with environment initialization step to create the communication domain between distributed machines. Each machine then performs its work and in parallel until finished then the MPI environment is terminated. During the parallel execution, machines can communicate using different message passing calls according to the required communication. Currently MPI supports various communication such as broadcast, gather/scatter and reduce.

2.5 Real-time Systems

Real-time systems are computing systems that have to react to input within strict timing constraints. The correctness of a real-time system is specified by the accuracy of the calculated output and also by meeting the required deadlines. To characterize a system as real-time, a clear understanding of the working environment and which influence its timing characteristics is needed. The system must operate in the same time scale with the environment, and guarantee that it will respond fast according to the environments evolution. Sometimes, such systems may be compromised when external, nonnatural, events disrupt the timing characteristics of the environment it operates in. False modeling of the environment timing characteristics can cause real-time systems to malfunction causing to the incorrect correlation between the systems time and the environments time. [37].

2.5.1 Types of Real-time Tasks

According to the criticality of a system's deadline, real-time systems can be divided into three categories [37]:

- Hard: the miss of a deadline causes catastrophic consequences to the system or the environment. An example of a hard real-time system is an airplanes motors and sensors control where deadline miss may cause catastrophic result from the feedback control loop.
- Firm: the missed deadline produces useless data for the system but does not damage the system or the environment. Miss deadline in a firm real-time system may lead to quality degradation, for examples tasks in signal processing domain, like video encoding/decoding or video streaming.
- Soft: the data produced after a deadline are still useful but lead to a degradation of the systems performance, for example in a graphic user interface of an application.

2.5.2 Features

A real-time system needs to have six fundamental properties to be able to host a critical application:

- Timeliness: able to guarantee that all the imposed deadlines are going to be met.
- Predictability: developed in a way where extensive analysis schemes can be applied to them, and varying behaviour over different scheduling techniques can be predicted.
- Efficiency: operates correctly on certain timing, power, area and computational power constraints.
- Robustness: behave correctly in extreme workload cased.
- Fault tolerance: able to tolerate permanent or transient faults and continue working correctly without compromise the systems operation.
- Maintainability: supports extensions or modifications to the system which can take place without much integration effort.

2.6 Related Works

To overcome the limited signal processing efficiency improvement, the most straightforward way to accelerate the SAR imaging is by using the parallel algorithm on high-performance computing (HPC) platforms. There are two kinds of methods can be applied to SAR imaging processing in HPC: the global shared memory system, and the distributed network of independent nodes [38].

Some researchers have implemented SAR imaging algorithms using special purpose DSP [39, 40, 41] and FPGA [42, 43, 41, 27] which can reach the level of real-time processing. However, the special purpose nature of the solutions makes the generality of this approach another challenging problem. There also exists several generic parallel implementations using CPU based utilizing OpenMP and MPI technologies [44, 45, 46], yet they are still have difficulties to achieve real-time processing performance.

In recent years, heterogeneous computing in Synthetic Aperture Radar processing has been a popular research topic. Clemente et.al. and Bisceglie et.al. demonstrate the potential of using GPU for SAR processing in context of Range Doppler Algorithm [47, 6]. The authors review the main computational features of a range-Doppler Synthetic Aperture Radar (SAR) algorithm, disclosing the degree of parallelism in SAR processing operations in context of GPU programming model.

Further parallelization of Range Doppler Algorithm on a single GPU is reviewed by authors in [17, 48, 49], followed review of the usage in multiple GPU [50]. GPU implementation of other processing algorithms also have been reviewed, for example SPECAN algorithm in [25], Chirp Scaling [18, 51], and Backprojection algorithm in [52, 53, 54].

Various existing research on GPU based methods for SAR imaging showed improvement of execution times for dozens of times and able to outperform traditional HPC methods [55, 56, 47]. However, several issues such as data transfer between CPU and GPU [57] and limited number of GPU devices and coordination in the usage of multiple GPU for computation [57, 56] still become major challenges. Although GPU computing can increase the efficiency of computation by many times using SIMD methods, they still cannot achieve the practical requirement for fast and real-time imaging.

Another factor that become a challenge in heterogeneous computing for SAR is that, often the computing power of the CPU is not optimized or sometime under utilized. On the typical case, CPU is only used for logical processing, controlling and input output tasks [57, 8]. To overcome this challenge, there are efforts to use multi-core parallelism on the CPU in order to gain processing efficiency [56]. However, the efficiency of parallel processing in CPU and GPU are at a different level, making the CPU contribution not as impressive as expected [8].

According to [8] there are three limiting factors in current heterogeneous CPU-GPU based SAR imaging:

- GPU memory size limitation. Due to small amount of GPU memory, typical SAR data with large size, can not fit into GPU memory which requires more complex processing and memory management technique.
- CPU power underutilization. Most CPU-GPU based SAR processing do not optimize the usage of CPU power and using CPU only for task coordination and IO related task.
- Lack of collaborative computing solution. There is a need to answer the scalability challenge of using both CPU and GPU computing resources in order to meet real-time imaging requirements.

In this work, we proposed to solve the problem of achieving near real-time SAR processing using quick-look and precision algorithm utilizing mobile heterogeneous computing platform. A recent study on the capability of GPU on mobile platform suggests that GPU embedded platforms have considerable potential for SAR processing tasks. This new emerging platform offers various advantage such as low power consumption, light weight and standard programming tools, which could open new possibilities of the embedded space applications in Synthetic Aperture Radar processing [58]. The previous study focus specifically on the porting existing based SAR application to the embedded GPU platform by maximize offloading task from CPU to GPU without considering how to best balance the load between the CPU and GPU. Design examples of utilizing GPUs to accelerate SAR processing computations can also be found in [59, 60, 61, 58, 49]. However to the best of our knowledge, there is no prior work that provides insight how the mobile CPU and GPU can balance the processing workload in context of SAR processing on quick-look and precision algorithm to achieve the best speed-ups.

Chapter 3

SAR Processing on Mobile Heterogeneous Platform

3.1 SPECAN on Mobile Heterogeneous Computing

This section presents the development of on-board quicklook SAR processing for the GAIA-II microsatellite platform. The software processes the SAR sensor raw data into intensity images for preview operation during satellite data acquisition mission. First, SPECAN algorithm for quicklook SAR processing is introduced, followed by discussion on the software implementations architecture for heterogeneous multicores. There are two versions implementations: sequential and heterogeneous CPU/GPU.

3.1.1 Requirements for the Quicklook Processor

The quicklook SAR processor on-board JX-2 UAV has the following objectives:

1. Processing the IQ signal acquired by the sensor into an amplitude image in near real-time fashion during the flight, and
2. The moderately low resolution image produced will be saved to disk and transmitted to the ground station for quick review of SAR image during a flight mission.
3. The processor is converting the raw data into image continuously as long as there is raw data in the buffer.

Because of the intensive computational process involved, the trade-off between the processing speed and the SAR image quality becomes a major concern. This works objective is to get a fast preview of SAR images during a mission, thus the processing speed becomes a priority over image quality. Which makes the usage of quicklook algorithm suitable for the purpose.

In terms of SAR processing, due to the similar geometry, the algorithm for processing linearly polarized SAR (LP-SAR) raw data can also be used to process CP-SAR data. In actual operation, a LP-SAR sensor will operate in full polarimetric mode: LL, LR, RL and RR polarization which can be divided into four channel ADC output. These facts made the volume of data produced by full polarimetric CP-SAR system four times as big as the linear polarization sensor.

The typical flow of a quicklook SAR processing is depicted in Figure 3.1. In general, the quicklook processing subsystem consists of three main process:

a raw data file buffer processor loop that reads raw signal and motion data file from the data recording subsystem, a process that convert the data using quicklook algorithm and a process that write the resulting image in common graphical format (e.g. jpg, bmp, png, gif) back to the storage in the data recording.

The processed SAR images in data recorder then will be transmitted to the ground system via the data downlink in order to give user a quick preview of the currently observed SAR area.

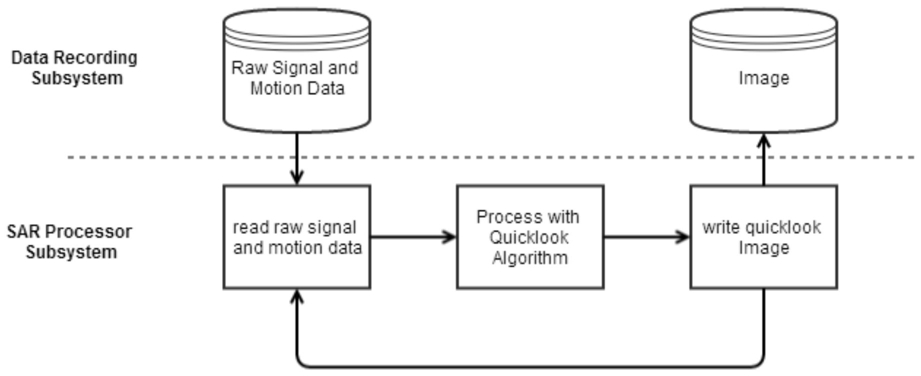

Data Recording Subsystem
Raw Signal and Motion Data
Image
SAR Processor Subsystem
read raw signal and motion data
Process with Quicklook Algorithm
write quicklook Image

Figure 3.1: General flow of the Quicklook processor

3.1.2 Modified SPECAN Algorithm

To produce the quicklook image specified by requirements in previous section, we have selected to implement the processor using Spectral Analysis (SPECAN) algorithm. SPECAN algorithm was invented in 1979 by MacDonald Dettwiler and Associates (MDA) to perform real-time processing on SEASAT SAR data. This algorithm provides efficient method of producing space borne SAR imagery of modest resolution, which makes it ideal for quicklook imaging [22]. Quicklook images produced by SPECAN algorithm showed degradation of the image quality. Recent work has been done to implement new methods of correcting the degradation while still focus on the main advantage of this algorithm which is efficiency in computation [62, 63].

Generally, SAR processing algorithm is divided into two main steps: processing in range and azimuth direction. Azimuth is the direction of platform flight while range is the direction perpendicular to azimuth. The original SPECAN algorithm depicted in Figure 3.2 consists of general processing in range and azimuth direction [22, 64].

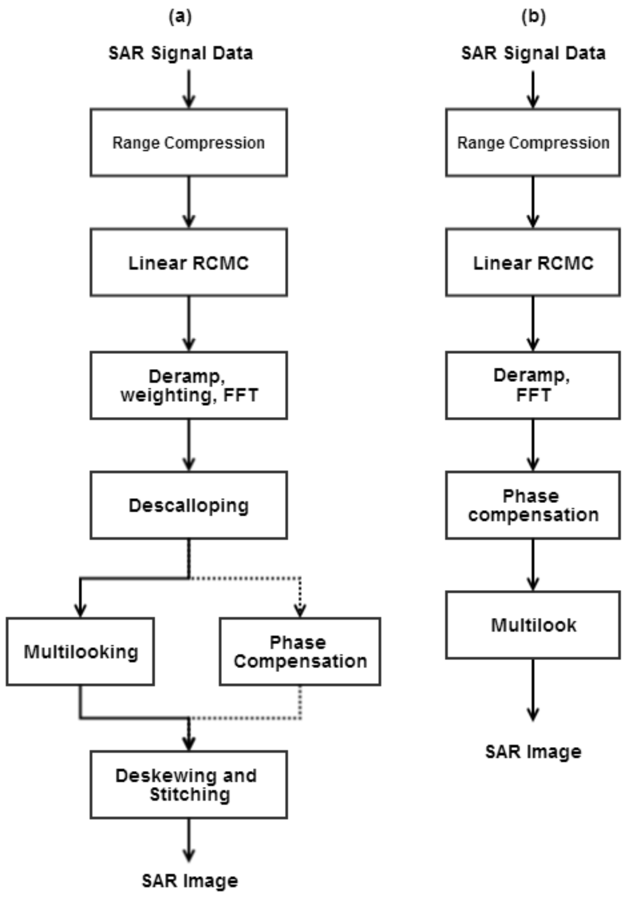


Figure 3.2: SPECAN algorithm processing flow. (a) Reference SPECAN algorithm (b) Implemented modified SPECAN algorithm

In SPECAN algorithm, range processing contains only pulse compression of received signals in the range direction, while azimuth processing consists of linear range cell migration correction (RCMC), deramping with weighting and FFT, descalloping, multilooking or phase compensation and deskewing with stitching operation. The range compression is done by pulse compression between the received signals in a range bin with the replica of transmitted signal. Using fast convolution technique, the range compression is carried out as multiplication in the frequency domain involving FFT and inverse FFT operation.

The azimuth processing steps are started with a linear RCMC operation to compensate the effect of range migration. After deramping and FFT operation that equals to time domain convolution, a descalloping operation takes place. Descalloping is done by applying functions which are inversely proportional to the predicted antenna gain pattern function to correct the scalloping effect due to different magnitude of each burst. If required, multilook processing must be done to generate a multilook image, but in the case of single look image, only phase compensation is done instead. The last stage is the step to correct the skewing effect in the result of each burst and a stitching process to combine all of the resulting SAR images into a single look image.

Figure 3.2 (b) shows the flow of modified SPECAN algorithm implemented in this work. The modified algorithm consists only of range compression, linear RCMC, deramping and FFT, phase compensation and multilook process. Range compression is implemented as an FFT operation to range samples in an azimuth gates with a range reference function, followed by multiplication operation and an IFFT operation on the result. A single range reference function is used for all azimuth gates. After range compression, RCMC is applied by shifting the range compressed data set at certain value for each range gates.

First the range curvature is calculated using Doppler centroid, chirp rate and chirp center value, followed by interpolation operation. The calculation is executed in the frequency domain, by applying a pair of FFT and inverse FFT operation.

Deramping calculation consists of a creation of complex matched filter using Doppler and PRF information, followed by matched filtering operation to the data set in range gates. Deramping is based on the stretch technique which allows linear manipulation of the time and bandwidth coordinates of a linear frequency modulated signal by mixing it with another signal of different frequency-time slope to slow down, speed up or time reverse the signal [62, 65].

Assuming that a linear frequency modulation (FM) signal has the following form:

$$s(t) = e^{j\pi K(t-\tau_r)^2}, -\frac{T_c}{2} \leq t \leq \frac{T_c}{2} \tag{3.1}$$

where K is the signal FM rate, T_c is pulse time extent, and is the time position of the zero frequency of the signal. The zero frequency of the target is the static phase point, the time at which the phase rate goes from positive to negative or vice versa. If the phase switch-over occurs in the middle of the signal time duration, the zero frequency will be at the same position as the center frequency.

The instantaneous frequency of the FM signal can be calculated using the first derivative of its phase, θ. From Equation 3.1, the phase, θ_s, of the transmitted signal is:

$$\begin{aligned} \theta_s(t) &= j\pi K(t-\tau_c)^2 \\ \omega_{inst} = \frac{d\theta_s(t)}{dt} &= 2\pi K(t-\tau_c) \end{aligned} \tag{3.2}$$

which shows that the instantaneous frequency, , is a linear function of time. Next, we apply the stretch technique. If the linear FM signal in Equation 3.1 is multiplied with another linear FM signal with different FM rate , and shifted with respect to $s(t)$, the product is a linear FM signal with FM rate equals to $(K + K1)$:

$$\begin{aligned} s_1(t) &= e^{j\pi K(t-\tau_r)^2}, -\frac{T_c}{2} \leq t \leq \frac{T_c}{2} \\ s(t)s_1(t) &= e^{j\pi(K+K_1)t^2} e^{-j\pi K_1(K\tau_c+K_1\tau_r)t} e^{j\pi(K\tau_c{}^2+K_1 tau_r{}^2)t} \end{aligned} \tag{3.3}$$

The first term in 3.3 is a linear FM signal with the FM rate of $(K + K1)$, the second term is a constant frequency signal, and the third is a constant complex number. In SPECAN algorithm, the same linear FM signal multiplication technique is also applied. The target signal is transformed from linear FM chirps to constant frequency signal using K as the value of K_1. This process creates a deramped signal, $d(t)$, such as

$$s_1(t) = e^{j\pi K(t-\tau_r)^2}, -\frac{T_c}{2} \leq t \leq \frac{T_c}{2} \tag{3.4}$$

The phase,$\theta_d(t)$, and instantaneous frequency, $\omega'_{inst}(t)$, of this signal are

$$\begin{aligned} \theta_d(t) &= \pi K\{2(\tau_r - \tau_c)t + (\tau_r{}^2 - \tau_c{}^2)\} \\ \omega'_{inst}(t) &= 2\pi K(\tau_r - \tau_c), \end{aligned} \tag{3.5}$$

where the instantaneous frequency does not vary with time.

In SPECAN algorithm, the matched filter is created using the FM rate of the transmitted pulse to generate the conjugate of the ideal received signal. It is then used to deramp the received data by a complex multiplication in the time domain. After deramping, each target will have a constant frequency value proportional to the time position of its zero Doppler frequency with respect to that of the reference function in Equation 3.2.

To narrow most of the energy of each target to a single frequency bin, a FFT operation is performed on this signal. The relative positions of the targets time locations will be detected at the correct positions relative to each other. Short length FFT blocks are placed along the deramped signal such that each targets trajectory is included in at least one FFT input vector [22].

Several important issues related to FFT operation after the deramping are: how to determine the position of the first FFT, how to calculate the length of FFT, how to select the correct FFT output, and where to put subsequent FFT operation. The length of the FFT operation is influenced by several factors such as the desired azimuth resolution, prevention of aliasing and computing efficiency [22]. Larger number of FFT point will cause heavier computational load.

The azimuth resolution obtained by an NFFT point of FFT can be formulated as

$$\rho_{az-CP} = \left(\frac{0.886 PRF \gamma_{w-az}}{N_{FFT} K_a} \right) v cos \theta_{sq} \tag{3.6}$$

where γ_{w-az} is the broadening factor of the weighting window, K_a is the azimuth FM rate, and θ_{sq} is the squint angle which in our case equals to 0. The length of FFT is governed by the PRF time: a mixed target will happen if it is longer than the PRF time. Therefore, the upper limit of the FFT length is the PRF value. The output sample spacing of the FFT output is calculated as one PRF in the frequency domain, which equals to $PRF/NFFT$ in frequency units; in time units, this corresponds to

$$\Delta_y = \frac{PRF}{N_{FFT}K_a}. \tag{3.7}$$

If the FFT size is not an efficient length (i.e., not in the power of 2), the value can be zero padded to a nearest better length. After this operation, the number of good points in the FFT output (N_{good}) can be calculated as

$$N_{good} = N_{FFT}\left(T_a - \frac{N_{FFT}}{PRF}\right)\frac{K_a}{PRF} \tag{3.8}$$

where T_a is the synthetic exposure time. The frequency of target after deramping (f_{tar_dr}) can be given as

$$f_{tar-d_r} = f_{\eta_c} + K_a\left(\eta'_{mid} - \eta'_{ramp_0}\right) \tag{3.9}$$

where f_{η_c} is the Doppler centroid frequency, η'_{mid} is time at the middle of target exposure, and η'_{ramp_0} is the time at which the reference function passes through zero frequency. Then, the position of the good FFT can be calculated by

$$k = \frac{N_{FFT}}{PRF}f_{tar-dr} + 1 \tag{3.10}$$

This indicates that the deramped frequency is at the k-th element, where the value k is between 1 and N_{FFT}.

After the valid pixel number is calculated, valid pixel value determines the size of final result of phase compensation. The phase compensation of azimuth deramped signal is done in the frequency domain. The phase compensation process is composed of an FFT operation, filter calculation and matched filtering operation between the filter and the range gate of the deramped signal dataset. Phase compensation also includes interpolation operation depending on the calculated image resolution. In the modified SPECAN algorithm, the descalloping process is skipped in order to avoid interpolation operation to reduce computation load. The last step of this quicklook processing is multilooking which will reduce the scalloping effect in the image. Multilooking process is carried out by averaging different parts of the signal spectrum to form processed SAR images.

3.1.3 Sequential Implementation

Figure 3.3 shows the flow chart of sequential implementation of SPECAN processor. The process takes three kind of input file, one configuration file, leader file and raw data file respectively. The configuration file contains the value related to the program execution parameter such as leader file name, raw data file name and processing result file name. Leader file and raw data file is actually part of a set of data from the SAR sensor. Leader file contains information related to the SAR data and sensor i.e. data set summary, map projection, platform position, attitude, radiometric, quality, spectra, ground control point and facility related data. The raw data file contains records related to file descriptor, signal data and processed data [66]

The implementation of SPECAN algorithm is as follows:

1. Before main processing, the program reads processing parameters from a configuration and leader and then calculate several processing parameters such as number of valid data in range and azimuth direction, Doppler

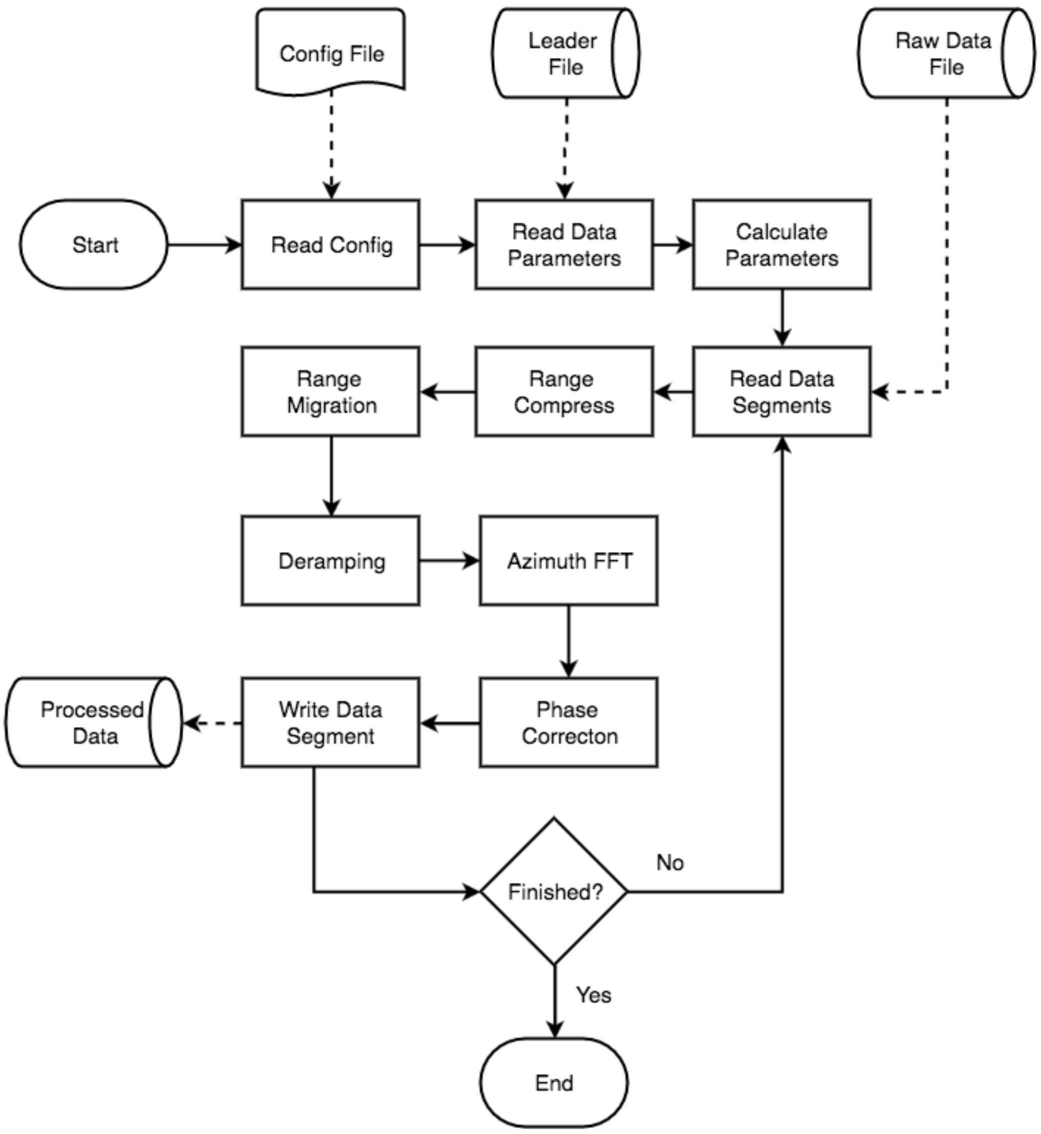


Figure 3.3: Flow chart of sequential SPECAN implementation

centroid and Doppler rate value which will be used further in the program.

2. The main processing of SPECAN algorithm consists of a loop that process a segment of raw data that loaded into a 2D array (matrix). The loop will be repeated as many as the number of segment and each one will result in a SPECAN processed image. This data partitioning technique is applied because it is not possible to load the whole data into

memory. After processing, all image segments will be combined into a bigger image.

3. The range compression process is implemented as element-wise complex number multiplication between a range reference function inside an array and every range lines in the current data segment. The result is a 2D range compressed data.

4. Range migration correction is applied to range compressed data inside the 2D array. In our implementation, the range migration is implemented using interpolation by means of sinc function.

5. Deramping process follows the range migration of 2D array data by first creating an azimuth chirp signal or azimuth reference function. The azimuth reference function then multiplied with the 2D array data to complete the deramping process. FFT operation to the deramped data in azimuth direction then applied to the data.

6. After azimuth FFT, a phase correction is applied using Doppler rate and frequency step data that we calculate in previous step.

7. The 2D array result of phase correction is then written to storage, and the program will continue processing the next available SAR data segment.

3.1.4 Heterogeneous CPU/GPU Implementation

Briefly, our parallel SPECAN algorithm is divided into eight steps depicted in Figure 3.3: read configuration file, read data parameters from leader file, parameter calculations, read segment of data/data partitioning, range compression, range migration, deramping, azimuth FFT, phase compensation and write data segment. Among those task, the I/O related task is the non-parallelizable while the rest of the task have potential to run in parallel.

We divide the process into steps carried out in CPU and in GPU, while trying as much as possible to offload the parallelizable work to the GPU. Two processes with tight relationship to input/output operations are handled by CPU i.e. reading raw data from file to memory and writing the results image blocks into file.

We also divide the process into steps according to its processing direction i.e. range direction and azimuth direction. Since the mobile heterogeneous platform usually does not have large amount of memory compare to the size of raw SAR data, a data partitioning strategy is needed in order to be able to distribute the work load between CPU and GPU.

In SPECAN processing, prior to range processing (range compress and range migration) and azimuth processing (deramping, azimuth FFT and phase compensation). A flow chart of the complete SPECAN algorithm implemented on the heterogeneous framework is given in Figure 3.4. The processing steps are as follows:

1. CPU divides the data along the azimuth direction
2. GPU executes range processing, which includes range compression and range migration.
3. CPU transposes the data from range processing result into format suitable for azimuth processing. After that, it divides the data along the range direction.

4. Azimuth processing is performed on the GPU, which includes deramping, azimuth FFT and phase compensation.

Using above approach, the coarse grained parallelism inside our SPECAN algorithm can be identified i.e. data partitioning, range processing, data transposition and azimuth processing.

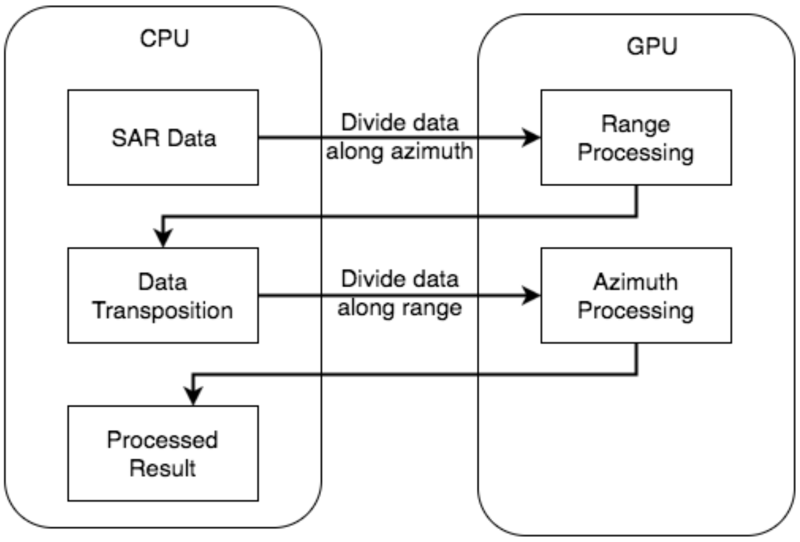


Figure 3.4: Flow Chart of RDA on Heterogeneous Platform

3.1.5 Data Partitioning

To distribute work to different processors, SAR data set must be partitioned into subsets where each processor performs essentially identical operations on each subset. There are a number of options that can be used to partitioning: (a) azimuth sub-swaths or strips (b) range sub-swaths or strips, and (c) sub-matrices. Each subset computed by launching a kernel on the GPU.

In current implementation the sub-matrices approach of partitioning which enables for creating most work as possible is selected. Matrices size is calculated using matched filter length size in either range or azimuth direction, which depend on the SAR system parameter. In order to optimize the execution time on CPU and GPU, the following principles are also considered:

1. To minimize idle resources, size of one the block should be equal to the size that can be process by CPU and GPU with the same latency value.
2. The size of block must be less than the size of GPU memory.
3. The length of the block must be selected with the value that optimize the FFT operation, which leads to the nearest 2^n value.

3.1.6 Range Compression

Range compression is implemented as an FFT operation to range samples in an azimuth gates, followed by multiplication with range reference function and IFFT of the result. A single range reference function is used for all azimuth gates. To optimize the operation, first all the azimuth gates are transformed using batched FFT operations, saved the result in GPU global memory, multiply the azimuth gates with range reference functions and batched IFFT of the result. The optimum number of parallel execution per blocks depends on the available GPU threads and memory, and limited by the available memory bandwidth. The global memory usage is optimized by coalesced read and writes operations [47].

3.1.7 Linear RCMC

Range Cell Migration Correction (RCMC) is completed by shifting the range compressed data set at certain value for each range gates. First the range curvature is calculated using Doppler centroid, chirp rate and chirp center value followed by interpolation operation. The calculation is executed in frequency domain, which involved a pair of FFT and IFFT operation. Parallel RCMC operation is implemented by assigning each range gate to one thread and use tiling method (by copying smaller chunks of data from GPU global memory to shared memory) to enable efficient kernel computations.

3.1.8 Deramping

Deramping process calculation consists of a creation of complex matched filter using Doppler and PRF information, followed by matched filtering operation to the data set in range gates. Similar to range compression, the optimum number of parallel execution per blocks also depend on the available GPU thread and memory, and limited by the available bandwidth. We also apply the coalesced read and write operation to optimize memory usage.

3.1.9 Phase Compensation

Before phase compensation operation, valid pixel calculation must be carried out. Valid pixel value determined the size of final result of phase compensation. The phase compensation of azimuth deramped signal is done in frequency domain, consists of an FFT operation, filter calculation and matched filtering operation between the filter and the range gate of the deramped signal dataset. Phase compensation also included interpolation operation depending on the calculated image resolution.

3.2 RDA on Mobile Heterogeneous Computing

This section presents the development of on-board precision SAR processor for the JX-2 UAV platform using Range Doppler Algorithm (RDA). First, the general processing flow of Range-Doppler algorithm is presented, followed by discussion on two implementations: sequential and heterogeneous CPU/GPU version.

3.2.1 Requirements for the Precision Processor

The precision SAR processor on-board JX-2 UAV has the following objectives:

1. Processing the IQ signal acquired by the sensor into an amplitude image during the SAR data acquisition flight.
2. The high resolution image produced will be saved to disk and transmitted to the ground station for preview or further processing.
3. The processor is converting the raw data into image continuously as long as there is raw data in the buffer.

3.2.2 Range-Doppler Algorithm

The Range-Doppler algorithm (RDA) is one of the oldest and the most commonly used algorithm for processing SAR data. The first successful application of this algorithm is for processing SEASAT-the first satellite borne SAR mission-data in 1976. RDA is computationally efficient and, for typical spaceborne imaging geometries and it is an accurate approximation to the exact SAR transfer function. Thus, the algorithm is phase-preserving and Single Look Complex (SLC) images formed with RDA can be used for advanced applications such as interferometry [67].

To achieve block processing efficiency RDA uses frequency domain operations in range and azimuth directions, while maintaining the simplicity of one-dimensional operations. It takes advantage of the approximate separability of processing range and azimuth directions, allowed large difference in time scales of the range and azimuth data, and by the use of range migration correction (RCMC) between the two one-dimensional operations in order to compensate the migration effect.

The RCMC operation is done in the range time and azimuth frequency domain to also gain block processing efficiency. This domain is known as ”range Doppler” domain, because azimuth frequency is the same with Doppler frequency. The algorithm is called range Doppler because the RCMC as the unique feature of this algorithm is executed in this domain.

A key feature of this algorithm lies in fact that the processing of energy from point targets, at the same range but different azimuth, is transformed to the same location in azimuth frequency domain. Therefore, correction of one target trajectory in this domain resulted in an effectively corrected family of target trajectories that have the same slant range of closest approach.

To achieve efficiency in implementation, the matched filter convolution process in range and azimuth directions are implemented as multiplication in frequency domain. RDA able to accommodate range variant matched filtering and RCMC operation with relative ease, and all operations are performed in one dimensional data arrays to achieve simplicity without losing efficiency.

Currently there are several variants of RDA available. For example, in order to handle data with a moderate amount squint, in 1984 NASA added a secondary range compression (SRC) step before RCMC is done. Using SRC the algorithm able to compensate the range and azimuth coupling of target's phase history and helps to remove phase distortions in case of SAR with squinted or large aperture datasets.

Figure 3.5 gives a block diagram of the basic RDA algorithm which suited for processing data from relatively small squint angle and aperture lengths antenna.

The following are the general steps of RDA:

1. Range compression is performed with fast convolution when the data are in the azimuth time domain. In other words, a range FFT is performed followed by a range matched filter multiply, and finally a range IFFT, to complete the range compression.

2. An azimuth FFT transform the data into range Doppler domain, where Doppler centroid estimation and most of the subsequent operations are performed.

3. RCMC, which is range time and azimuth frequency dependent, is performed in the range Doppler Domain, where a family of target trajectories at the same range are transformed into one single trajectories so that they now run parallel to the azimuth frequency axis.

4. Azimuth matched filtering can be conveniently performed as a frequency domain matched filter multiply at each range gate.

5. The final step in azimuth IFFT to transform the data back to the time domain, resulting in a compressed image. Detection and look summation can be done at this stage, if desired.

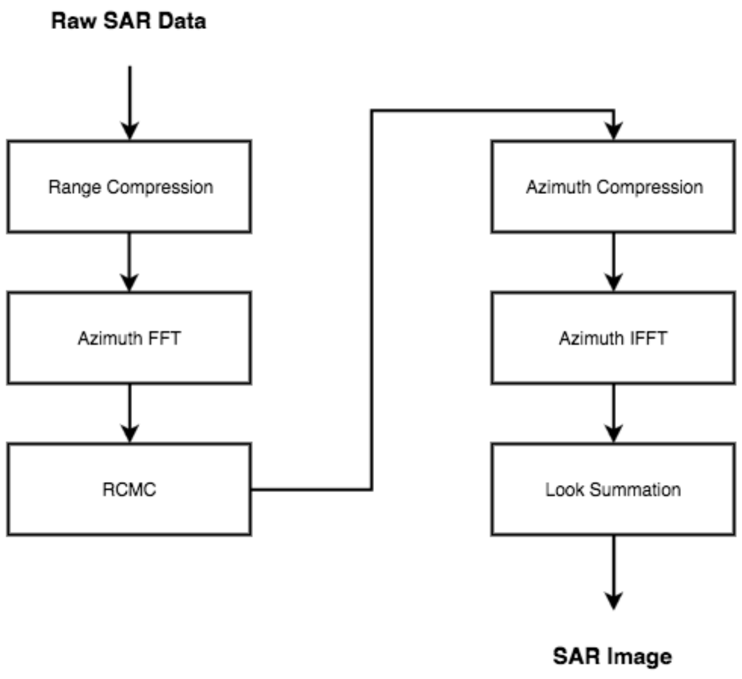


Figure 3.5: General steps of Range Doppler Algorithm

Data received from the radar system are referred to as signal data or raw data. The raw data are first demodulated to baseband, so that the nominal center frequency is zero. The demodulated radar signal, , received from a point target can be modelled as (3.1).

$$\begin{aligned} s_0(\theta,\eta) &= A_0\omega_r\left(\gamma-\frac{2R(\eta)}{c}\right)\omega_\sigma(\eta-\eta_c) \\ &\times e^{\left(-\frac{j4\pi f_0 R(\eta)}{c}\right)} e^{\left(-j4\pi K_r\left(\gamma-\frac{2R(\eta)}{c}\right)^2\right)} \end{aligned} \tag{3.11}$$

Where

A_0 = an arbitrary complex constant

γ = range time

η_c = azimuth time reference to closest approach

η = beam center offset time

$\omega_r(\tau)$ = range envelope (a rectangular function)

$\omega_\sigma(\tau)$= azimuth envelope (a sinc-squared function)

f_0 = radar center frequency

K_r = range chirp FM rate

$R(\eta)$ = instantaneous slant range

The two omega terms model the magnitudes of the range and azimuth signals, and are often neglected in the signal analysis. The instantaneous slant range,$R(\eta)$, is given by:

$$R(\eta) = \sqrt{(R_0^2 + V_0^2\eta^2)}, \tag{3.12}$$

where R_0 is the slant range closest approach.

For this target, the azimuth time, η, is referenced to zero Doppler. When multiple targets are considered, a common absolute time, η, is needed, such as η at the start of data acquisition.

In range compression, range FFT is performed followed by a range matched filter multiply. Range IFFT is then performed to the data after the range

Matched filter multiply, where the range matched filter phase is defined as

$$s_1(\tau,\eta) = IFFT_\tau(FFT(s_0(\tau,\eta)G(f_\tau)) \tag{3.13}$$

$$G(f_\tau) = e^{-j\pi K_\tau \tau^2} \tag{3.14}$$

Second step of RDA is azimuth FFT. In this stage, azimuth FFT is performed on the data, which transforms the data into range Doppler domain (τ, f_η).

$$s_2(\tau, f_\eta) = FFT(s_1(\tau,\eta)) \tag{3.15}$$

Third step of RDA is RCMC which performed by a range interpolation operation in the range Doppler domain. In RCMC, an interpolation is implemented based on the sinc function. The data are corrected by the amount given by RCM, which is defined in (2.6).

$$s_3(\tau, f_\eta) = RCMC(s_2(\tau, f_\eta)) \tag{3.16}$$

$$\Delta(f_\eta) = \frac{\lambda^2 R_0 f_\eta^2}{8V_t^2} \tag{3.17}$$

Azimuth Compression is the fourth step of RDA. In azimuth compression, azimuth matched filtering is performed on the data at each range gate in range Doppler domain. Azimuth matched filter phase, $H_{az}(f_\eta)$,is as defined in

$$s_4(\tau, f_\eta) = s_3(\tau, f_\eta)H_{az}(f_\eta) \tag{3.18}$$

$$H_{az}(f_\eta) = e^{\left(-j\pi \frac{f_{eta}^2}{K_a}\right)} \tag{3.19}$$

$$K_a \approx \frac{2V_\tau^2}{\lambda R_0} \tag{3.20}$$

RDA has advantage from other algorithms as it is a simple and basic algorithm that can be adapted to most SAR processing tasks. RDA also has one of the best trade-offs between accuracy and complexity. Many find it the easiest to understand and implement although its accuracy is not the highest. RDA can also be easily implemented in an efficient pipeline architecture where all operations are done in one dimension at a time.

One of the disadvantages of RDA is the restrictive beam-width and squint limitations. RDA also has high computation load as the interpolator used for RCMC is a time consuming operation.

3.2.3 Sequential Implementation

Figure 3.6 shows the flow chart of sequential implementation of RDA processor for UAV.

The implementation of RDA algorithm for UAV is as follows:

1. The program has three kinds of input: (1) program configuration (2) flight position data (3) raw data. After reading the application configuration data, the program reads the flight position and calculate data partitioning scheme.

2. The main loop of the program starts reading one segment of raw data at a time until all data in the file is processed.

3. After reading a segment of raw data, an interpolation step to generate missing data is required if the data is not uniformly sampled. An additional step of interpolating missing data from the raw data is needed if the data is not uniform in time. This step is required because RDA algorithm works on uniformly sampled data.

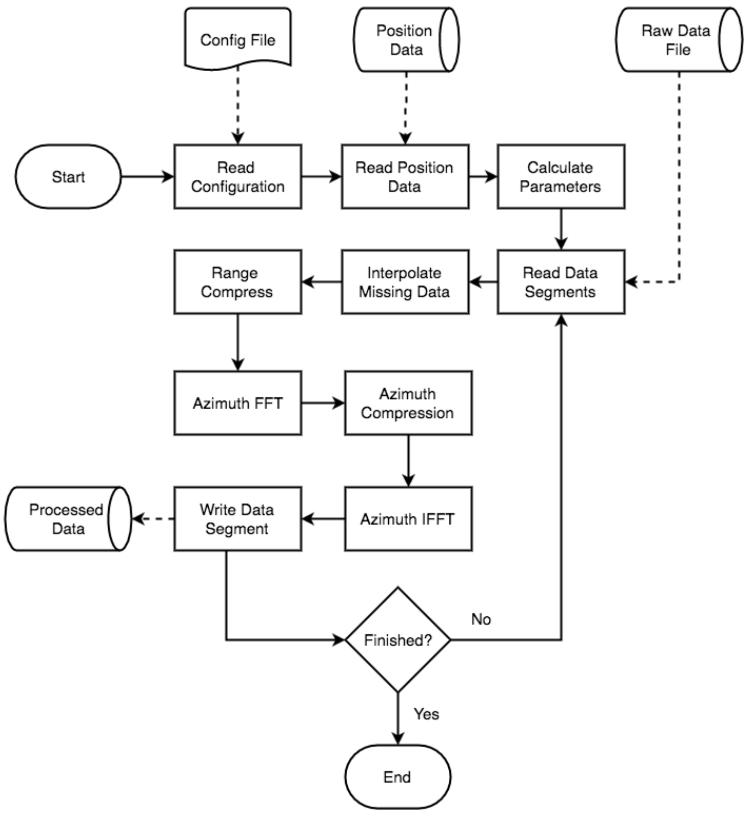


Figure 3.6: Range Doppler Algorithm

4. Range compression by frequency domain complex multiplication is carried out between range reference function and the raw data.

5. Azimuth FFT to convert the range compressed data to frequency domain.

6. Azimuth compression by frequency domain complex multiplication is carried out between azimuth reference function and Azimuth FFT data.

7. Azimuth IFFT to convert the data into time domain.

8. Write azimuth compressed data to disk buffer.

All the steps in the algorithms are done sequentially. As we can see from the flow chart description, the processing of each segment of raw data is independent to other segment. This means that there is an opportunity to accelerate the program by processing of each segment in parallel.

The details processing steps of each algorithms are described in Algorithm 1 to 5.

Algorithm 1: Interpolate Missing Data

```
Input: M1 /* raw signals with missing data */
Output: M2 /* raw signals with missing data filled */
initialization;
read the timestamp of each sample in M1;
for each missing timestamp do
    interpolate new sample array from the previous and next array;
    store array inside matrix M2;
end
return M2;
```

Algorithm 2: Range Compression

```
Input: M1, R1 /* raw data, range reference */
Output: M2 /* range compressed data */
initialization;
for each Vector V1 in M1 do
    Read one vector into R1 ;
    V2 = FFT(R1) * FFT(V1) ;
    Save V2 into M2 ;
end
return M2 ;
```

Evaluating each sub process inside the RDA algorithm, we found the following parallelism opportunities:

1. Missing data interpolation, we can parallelize the for loop that calculate each missing data.

2. Range compress and Azimuth compress, there are opportunities to parallelize the FFT operation for each range/azimuth samples and the vector multiplication operation as well.

3. Azimuth FFT and Azimuth IFFT, there is an opportunity to parallelize the FFT/IFFT operation for each azimuth samples. The implementation

Algorithm 3: Azimuth FFT

```
Input: M1 /* raw data */
Output: M2 /* raw data after azimuth fft */
initialization;
for each Vector V1 in M1 do
    Read one vector into R1 ;
    V2 = FFT(R1) ;
    Save V2 into M2 ;
end
return M2 ;
```

Algorithm 4: Azimuth Compression

```
Input: M1, R1 /* raw data, azimuth reference */
Output: M2 /* azimuth compressed data */
initialization;
for each Vector V1 in M1 do
    Read one vector into R1 ;
    V2 = FFT(R1) * FFT(V1) ;
    Save V2 into M2 ;
end
return M2 ;
```

of these parallelization opportunities are discussed in the next section.

3.2.4 Heterogeneous CPU/GPU implementation

The RDA processing can be categorized into two kind of processing: processing in the range direction and in azimuth direction. The Jetson TK1 mobile heterogeneous platform that was used in our experiments consists of a four cores CPU and 192 cores GPU. Compare to the size of raw SAR data plus the amount of memory needed during processing, the available 2GB of RAM is quite limited. Consequently, prior to the range processing (i.e. interpolate missing data and range compression) the large-scale data should be divided into smaller segments along the azimuth direction. And it should be divided

Algorithm 5: Azimuth IFFT

```
Input: M1 /* raw data                                      */
Output: M2 /* raw data after azimuth ifft                  */
initialization;
for each Vector V1 in M1 do
    Read one vector into R1 ;
    V2 = IFFT(R1) ;
    Save V2 into M2 ;
end
return M2 ;
```

into smaller segments along the range direction before azimuth processing (i.e. azimuth FFT, azimuth compression and azimuth IFFT).

A flow chart of the complete algorithm implemented on the heterogeneous framework is given in Figure 3.6. The SAR data is processed step by step as follows:

1. CPU divides the data along the azimuth direction
2. Range processing is executed on GPU, which includes missing data interpolation and range compression.
3. CPU transposes the data from the result of range processing into format for azimuth processing. After that, it divides the data along the range direction.
4. The azimuth processing is performed on the GPU, which includes (azimuth FFT, azimuth compression and azimuth IFFT).

Concluding from above approach, the coarse grained parallelism inside our RDA algorithm can be identified i.e. data partitioning, range processing, data transposition and azimuth processing, which are similar with the SPECAN processing algorithm. In above heterogeneous framework, the data processing

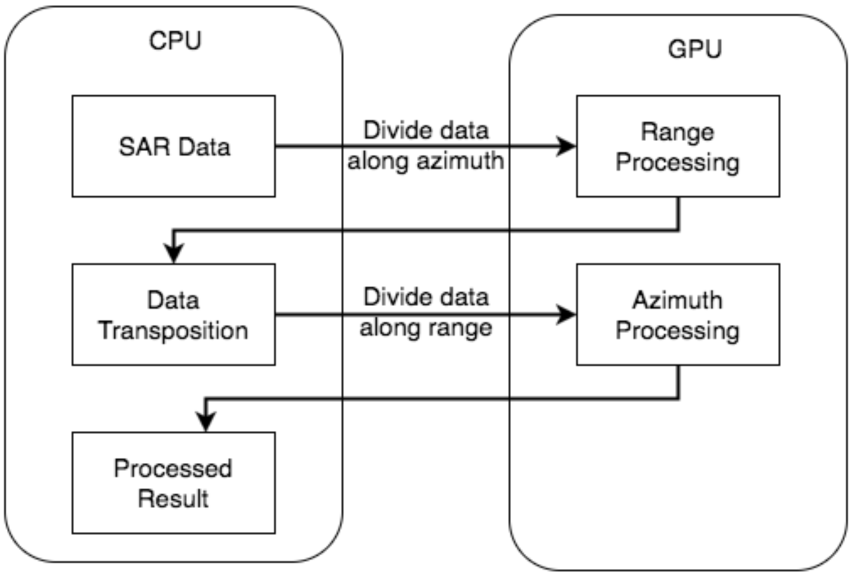


Figure 3.7: Flow Chart of RDA on Heterogeneous Platform

between CPU and GPU is executed sequentially, bringing the total processing time equal to the sum of processing time of CPU and the processing time of GPU.

In order to accelerate the above execution scheme, a pipelining processing approach between CPU and GPU based on data segment is proposed. We take the advantage of asynchronous GPU kernels in order to execute different task in CPU and GPU concurrently. This execution scenario is depicted in figure 3.8.

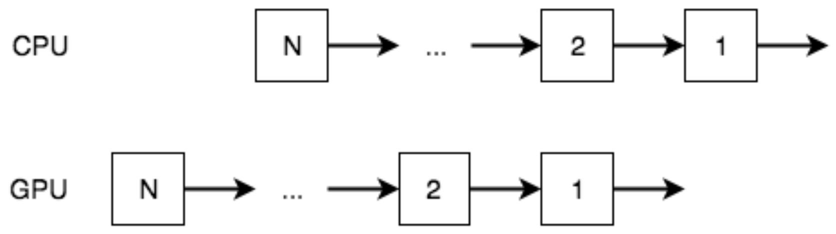


Figure 3.8: Parallel task execution scenario

Because the processing of each data segment is independent with others, the execution of one task on one data segment on CPU can be carried out in parallel with the execution of another task on GPU. For example, the CPU task no. 2 can be run at the same time with the GPU task no. 1. And the CPU task no N can be run at the same time with the GPU task no N-1.

Using above execution model, we can achieve the total execution time of data approximately equal to the total execution of GPU, leading to the reduction of total processing time.

Algorithm implementation on the GPU

The processing on GPU can be categorized into two types: range and azimuth processing. Referring to previous section, majority the algorithm basically consists of two mathematical operations:

1. FFT and IFFT, the forward and inverse Fast Fourier Transform which converts the data from time to frequency domain and vice versa. These operations are used in range compression, azimuth compression, azimuth FFT and azimuth IFFT.
2. Complex Vector multiplication, which used by the missing data interpolation, range compression and azimuth compression.

Both algorithms implementation on the GPU are as follows:

1. Vector multiplication, implemented as three level hierarchy of GPU threads: threads, blocks and grids. Threads are grouped into blocks and multiple blocks are grouped into grids. Each kernel function code will be executed by every threads, and the result are stored in device memory. Since each vector multiplication on SAR processing is independent with others, the

multiplication of each vector element can be executed by one thread, resulting a group of threads working on a single vector multiplication.

2. FFT and IFFT, one dimensional complex to complex FFT and IFFT operation are implemented using the available cuFFT library. The range and azimuth direction FFT/IFFT is implemented separately. To execute multiple FFT/IFFT operation at the same time, the batch mode operation of cuFFT is used. Using cuFFT library enable us to use parallel processing without having to develop our custom FFT implementation.

Using above approach we are able to identify the fine grained parallelism potential inside the RDA algorithm.

Algorithm Implementation on the CPU

In our implementation, the CPU handles the tasks of data partitioning into segments and arranging the direction of data in segments in range or azimuth depending on the processing direction requirement. The task of arranging data direction is implemented using transposition algorithm which is more time consuming compare to the data partitioning task. Because the transposition of one segment of data is not related to other data segment, the CPU can carry out this process using multithread which will reduce the processing time. On the other hand, although the data partitioning task is not a time consuming process, it plays an important role in the processing. Dividing data into unsuitable blocks of segment will increase the algorithm complexity. We use the following principles in order to partition the data into segments:

1. To minimize the idle resource, size of one the block should be equal to

the size that can be process by CPU and GPU with the same latency value.

2. The size of block must be less than the size of GPU memory.

3. The length of the block must be selected with the value that optimize the FFT operation, which leads to the nearest 2^n value.

Parallel CPU and GPU execution

After the CPU divide the large SAR data into different independent blocks, the CPU and GPU can process each blocks concurrently. In range processing some threads on CPU will transpose the (N+1)-th data block and instruct the GPU to process the Nth data block simultaneously. Other thread can instruct GPU to process the data block while waiting for (N+1)-th transposition. This scenario is depicted in Figure 3.9.

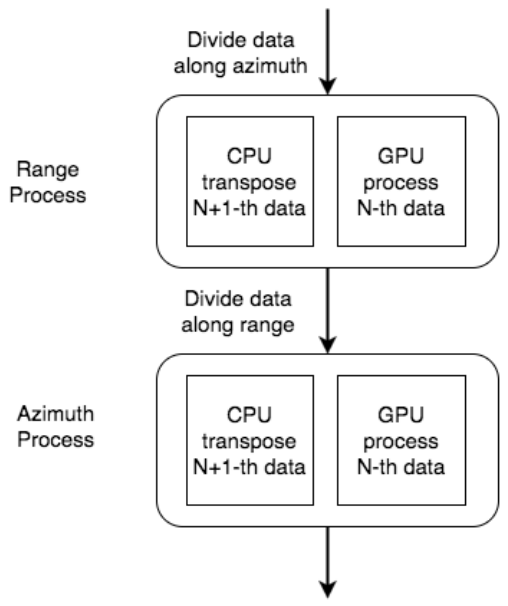


Figure 3.9: Flow Chart of Parallel CPU and GPU Threads

Chapter 4

Experimental Results

This chapter contains the results and evaluation of the developed on-board SAR processing software. Since actual CP-SAR data from UAV or Microsatellite based sensor are not available yet, there are two possible testing scenarios: using synthetic/simulated data generated with parameters from current system under test or using actual data from currently available system that have similar parameters with system under test. This thesis follows the second path of using actual data from currently available air-borne and space-borne sensor, in order to demonstrate the imaging capabilities of the software on actual complex SAR data.

4.1 Experimental Setup

In order to test the software implementations, we use Jetson TK1 mobile heterogeneous platform, a small (5 inches by 5 inches) board designed for development of embedded and mobile applications depicted in Figure 4.1. The

Table 4.1: Specification of evaluation hardware & software

Parameter	Value
Name	Jetson TK-1
CPU	4+1 cores
CPU architecture	Arm Cortex 15
GPU	NVidia Tegra K1
GPU architecture	Kepler
No of GPU Cores	192
GPU memory size	2 GB
Compute capability	3.2
OS	Ubuntu Linux 14.04
Kernel version	3.10.40-gdacac96

Jetson TK1 is powered by the Tegra K1, a mobile processor featuring a CUDA-capable GPU. It has a GK20A Kepler GPU with 192 cores and a quad-core ARM Cortex-A15 32bit CPU. Jetson has 2GB of system RAM and run Ubuntu Linux version 14.4. Table 4.1 shows the specification of the evaluation hardware and the physical appearance is depicted in Figure 4.1.

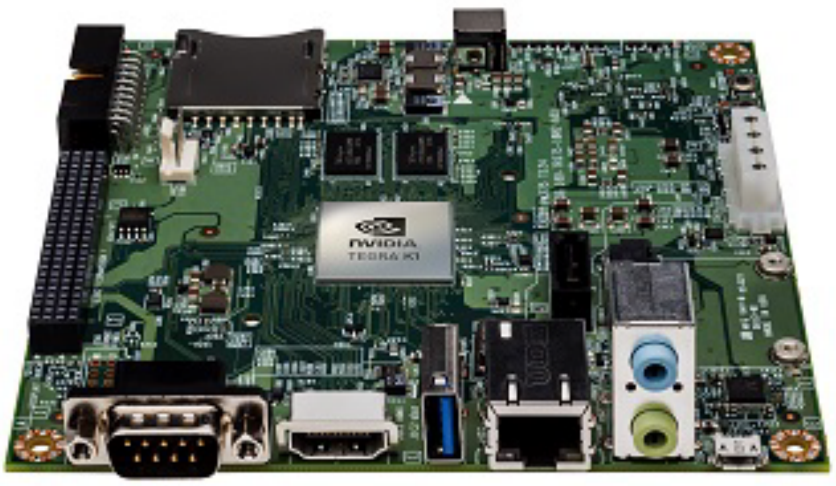


Figure 4.1: NVidia Jetson-TK1 development board hardware

Table 4.2: GAIA-II and JERS-I parameters comparison)

Parameter	JERS-I	GAIA-II
Frequency	1.275 GHz	1.270 GHz
Bandwidth	15 MHz	10 MHz
Off-nadir angle	54.90	29
Pulse length	35.00 μs	50.00 μs
Polarization	HH	LL, LR, RL, RR
Swath width	75 km	50 km

4.2 SPECAN Results

We have implemented two types of quicklook SPECAN processors:

1. Sequential CPU version (SSC)

2. Heterogeneous CPU and GPU version (SCG)

Several measurements have been performed in order to record and compare the processing speed of both quicklook SPECAN processor. The following sections discuss details of input data used and results of the program executions.

4.2.1 Input Data

Actual data from Japanese JERS-1 SAR satellite has been chosen to test the SPECAN algorithm due to its relatively similar system configuration and orbit geometry with GAIA-II. Table 4.2 shows the comparison between JERS-1 and GAIA-II configuration parameters.

JERS-1 has orbit altitude of 568 km compare to GAIA-II of 500 to 700 km and both has approximately the same center frequency. JERS-1s off nadir angle is

Table 4.3: Data set parameters)

Parameter	Value
Sensor name	JERS-I SAR
Frequency	1.275 GHz
Bandwidth	15 MHz
Observation date	1998/1/17
Off-nadir angle	54.90
PRF	1555.20 Hz
Pulse length	35.00 μs
Sampling rate	17.073 MHz
Range samples	6144
Azimuth lines	19904
Polarization	HH
Swath width	75 km
Data file size	242 mega Byte

wider (54.9) than GAIA-II (29) causing it to have wider swath-width (75 km compared to 50 km).

Table 4.3. list the specification of JERS-1 satellite raw data used for testing the SPECAN implementation.

The selected raw data set represents an area of approximately 75 km $\times$ 75 km, acquired over southern part of Bali Island, Indonesia. Figure 4.2 shows the intensity image of processed using Range Doppler algorithm.

4.2.2 Experimental Results

Table 4.4 shows the result of processing the whole sample data with sample size of 6144 in range and 19904 azimuth lines using Sequential CPU and the Heterogeneous on CPU/GPU. The value written is the average value of 100 times executions with minimum and maximum outliers value excluded.

Table 4.4: SPECAN processor program execution time

Program	**Latency**
Sequential CPU (SSC)	623.015 s
CPU/GPU (SCG)	95.848 s

Using values from Table 4.4, the speed-up value for SPECAN processor is:

$$SPECAN\ Speedup_{scg} = \frac{SSC\ Latency}{SCG\ Latency} = 6.5\ times \qquad (4.1)$$

4.3 RDA Results

4.3.1 Input Data

Actual raw data of microASAR UAV image from CASIE-09 has been chosen to test the RDA algorithm to its relatively similar flight geometry with JX-2. Table 4.5. list the comparison of of CASIE and JX-2 SAR platform.

Table 4.5: JX-II and CASIE parameters comparison)

Parameter	JX-II	CASIE
Frequency	1270 MHz	5428.76 MHz
Velocity	100 km/h	36-540 km/h
Altitude	1-4 km	1.5-3 km
Bandwidth	150 MHz (max)	80-200 MHz
PRF	1 kHz	7-14 kHz
Swath width	1000 m	300-2500 m

4.3.2 Experimental Results

Table 4.6, shows the result of processing the whole sample data with sample size of 3884 in range and 1702 azimuth lines using Sequential CPU and the Heterogeneous on CPU/GPU. The value written is the average value of 100 times executions with minimum and maximum outliers value excluded.

Table 4.6: RDA processor program execution time

Program	**Latency**
Sequential CPU (SSC)	65.730 s
CPU/GPU (SCG)	15.650 s

Using values from Table 4.6, the speed-up value for RDA processor is:

$$RDASpeedupscg = \frac{SSC\ Latency}{SCG\ Latency} = 4.2\ times \tag{4.2}$$

4.3.3 Double Precision vs Single Precision

In order to observe the effects of the usage of double precision and single precision floating point on the computation, we implemented the algorithm using both data types. Figure 4.6 shows the difference between single precision and double precision implementation.

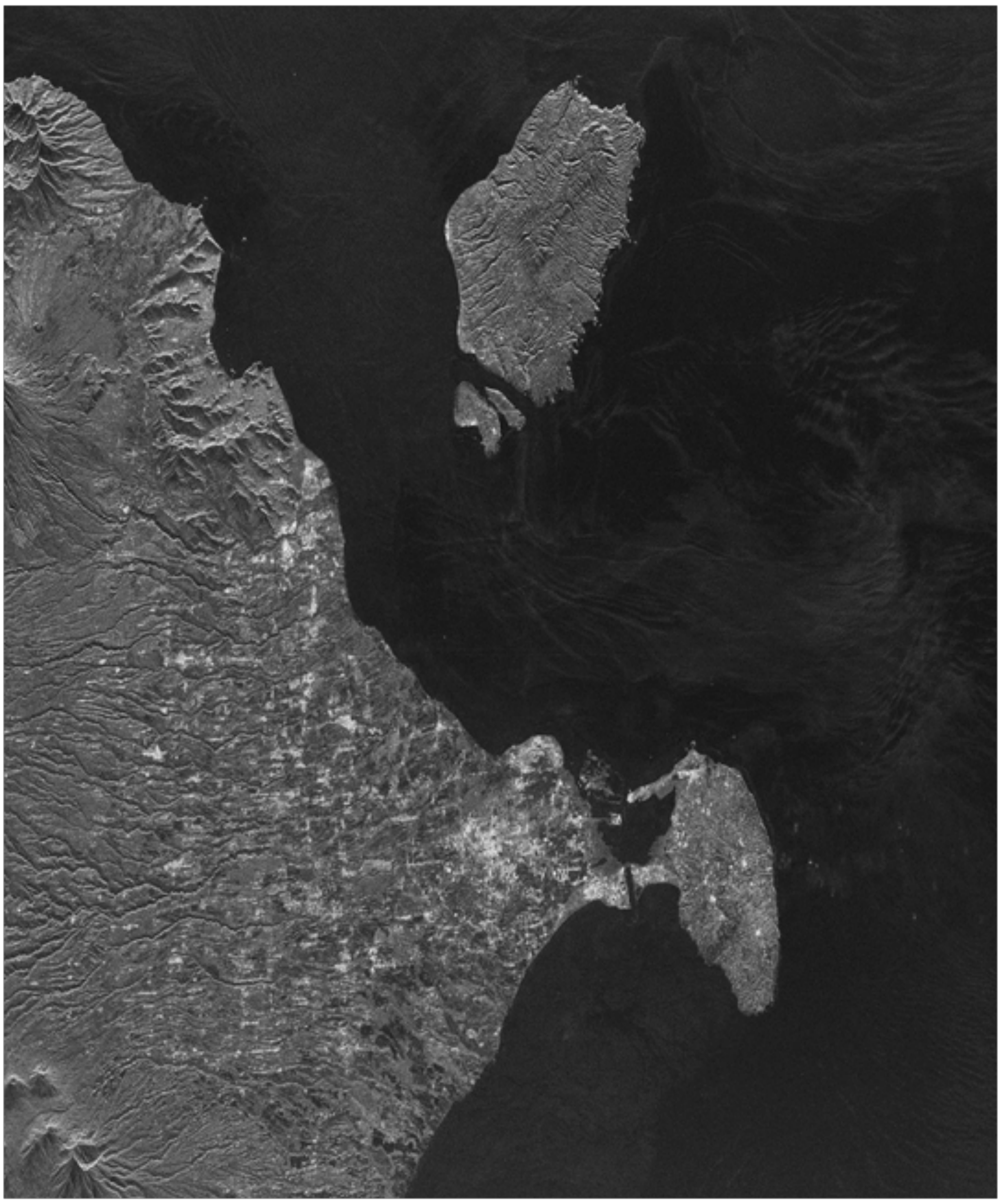
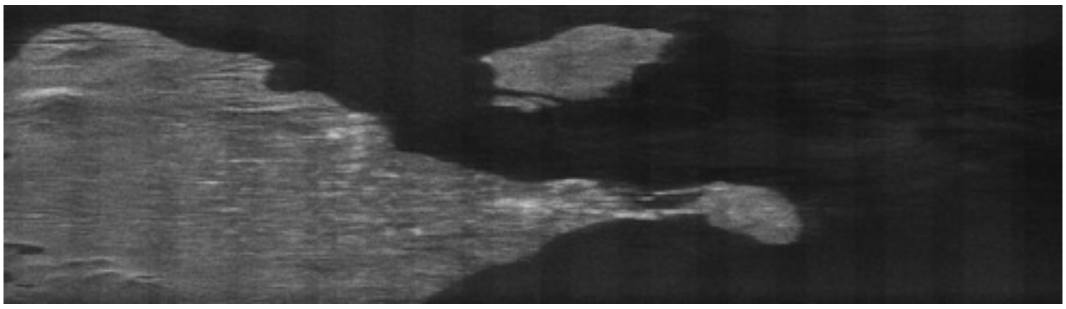
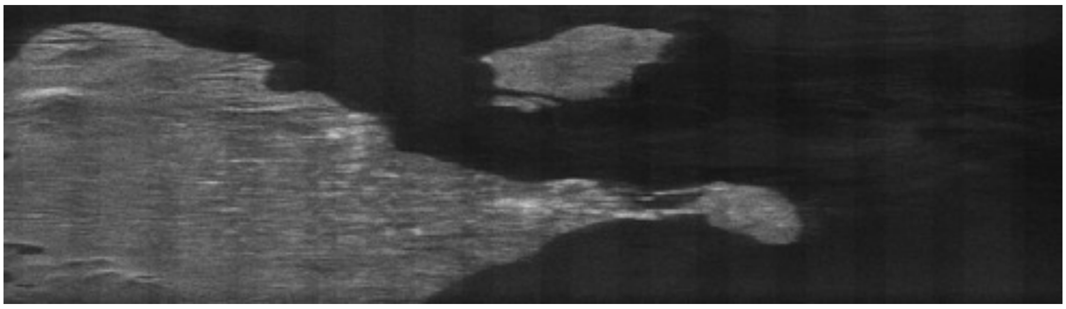


Figure 4.2: Intensity image of JERS-1 SAR sample data used for the experiment if processed with Range Doppler Algorithm (3 Looks)


(a) SPECAN sequential CPU


(b) SPECAN CPU/GPU

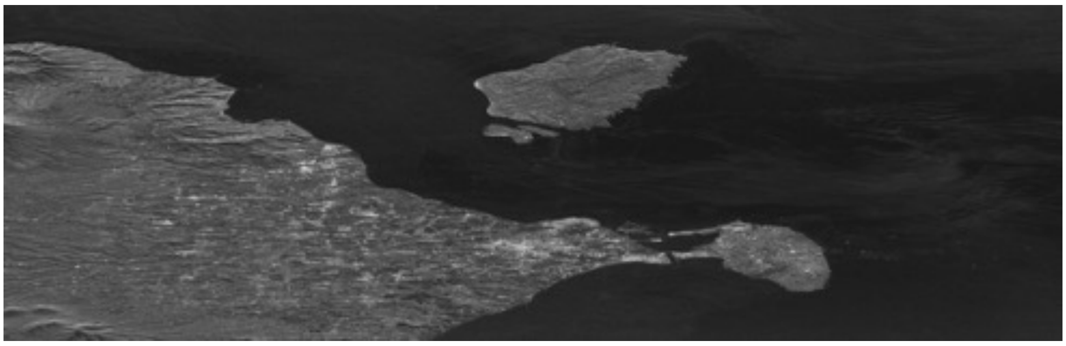

(c) RDA

Figure 4.3: Intensity image of JERS-I SAR sample data(all images in 1 look)

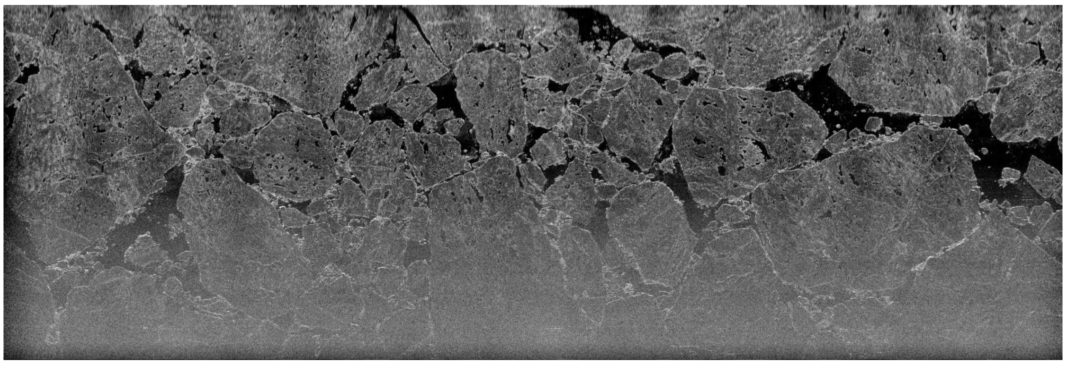
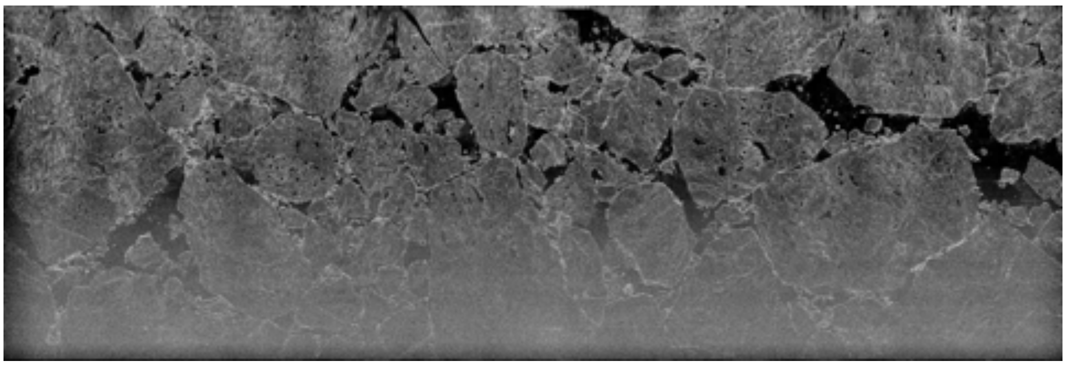
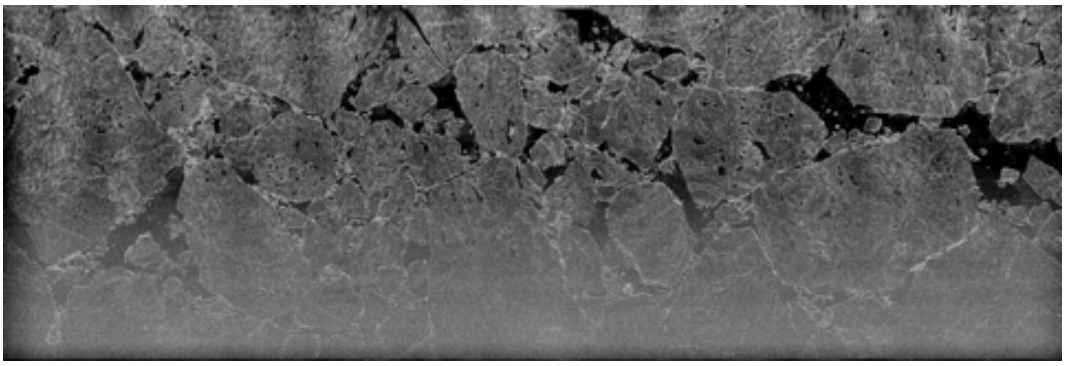


Figure 4.4: Sample Intensity image of microASAR from CASIE-09 processed raw SAR data processed with original Range Doppler Algorithm from the sample data.


(a) RDA sequential CPU


(b) RDA CPU/GPU

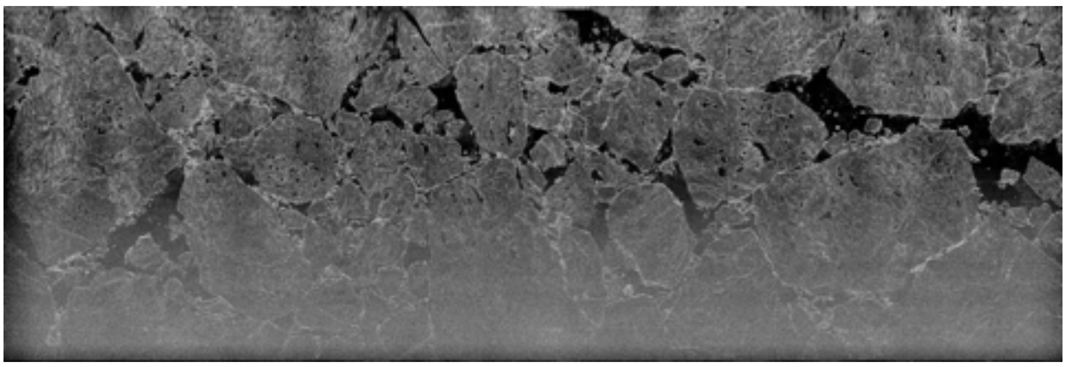

(c) Original RDA processor program included from sample data

Figure 4.5: Intensity image of CASIE-09 SAR sample data (all images in 1 look)

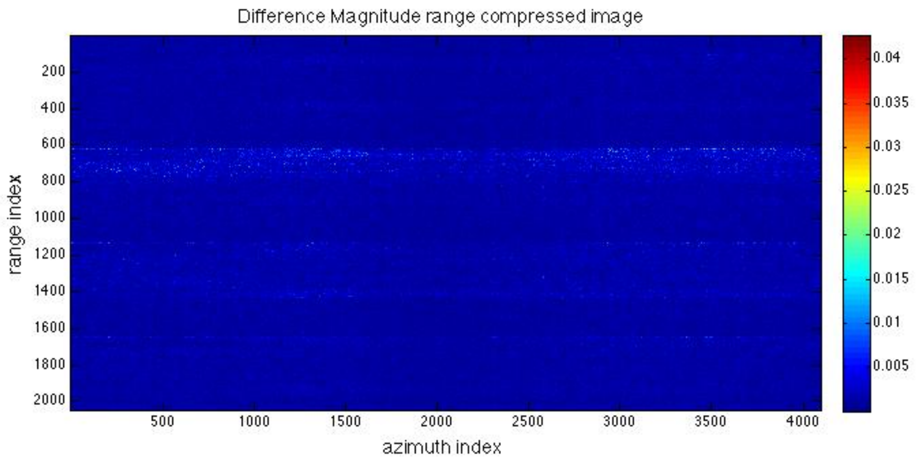


Figure 4.6: Difference of range compression image result using double and single precision.

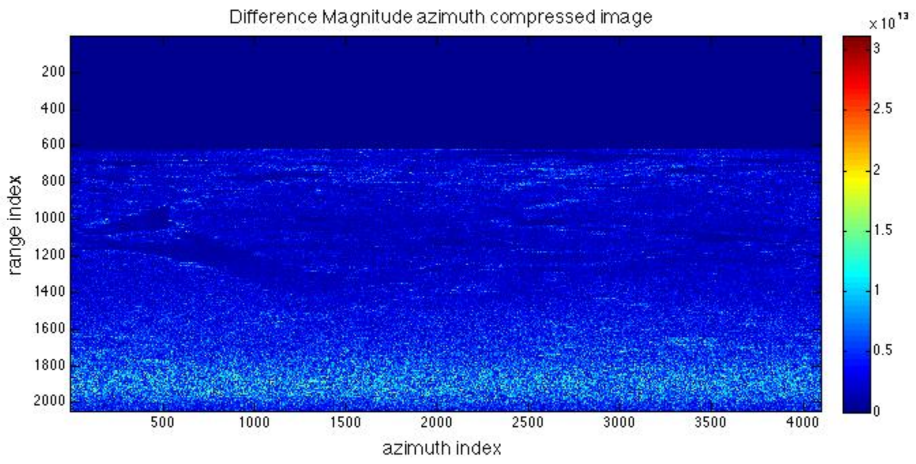


Figure 4.7: Difference of azimuth compression image result using double and single precision.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

In this thesis we have developed quicklook and precision SAR processing algorithm for mobile heterogeneous computing platform to overcome the SWAP problem on UAV and Microsatellite platform.

The quicklook SAR processor is designed for L-band CP-SAR sensor onboard future GAIA-II microsatellite and able to generate low-resolution. It is based on modified spectral analysis (SPECAN) algorithm for monostatic stripmap mode SAR image formation. The precision SAR processor which based on Range-Doppler Algorithm (RDA) is intended for generating high resolution images for L-band linearly polarized SAR sensor onboard the future JX-2 UAV.

To optimize the algorithms, we have analysed the details of both algorithms in order to find parallelism potentials. For a given SAR raw data our parallelism model uses (1) data partitioning based on the number of samples in range and azimuth directions to balance the task workload on the underlying hardware

(2) identification and separation of sequential SAR processing tasks into coarse and fine grained parallel tasks.

To evaluate the result, we have implemented the sequential version of both applications on single CPU core as the baseline. The quicklook implementation is validated using actual raw data from JERS-1 SAR sensor that has similar characteristics with GAIA II CP-SAR sensor and the precision algorithm is validated using publicly available raw data from a UAV based stripmap mode LFMCW SAR sensor. We have evaluated our proposed approach using various size inputs number of samples and bins on a middle-end mobile heterogeneous CPU-GPU development kit. Implementation on an integrated quad-core CPU and 192 cores CPU showed that regardless of the size of range samples and azimuth bin, speedup up to 4.2 and 6.5x over the sequential version can be achieved for quicklook and precision application respectively.

5.2 Contributions

This dissertation makes novel contributions to the field of synthetic aperture radar, particularly in regard to the SAR processing using quicklook and precision algorithm. These contributions are organized into two primary categories: implementation of quicklook using SPECAN algorithm on heterogeneous CPU/GPU architecture and implementation of precision algorithm and implementation of precision processing using Range-Doppler algorithm on heterogeneous CPU/GPU architecture.

The specific contributions of this research are summarised as follows:

1. We have designed and developed quicklook SAR processing system based

on modified SPECAN algorithm for use on-board future GAIA-II CP-SAR satellite space-borne platform.

2. We have designed and developed precision SAR processing system based on Range Doppler algorithm for use on-board future JX-2 CP-SAR air-borne platform.

3. We have implemented and tested the quicklook SAR processing system based on modified SPECAN algorithm for use on-board future JX-2 CP-SAR airborne platform on mobile heterogeneous computing hardware as an effort to overcome the SWAP limitation.

4. We have implemented and tested the quicklook SAR processing system based on Range-Doppler algorithm for use on-board future JX-2 CP-SAR airborne platform on mobile heterogeneous computing hardware as an effort to overcome the SWAP limitation.

5.3 Future Works

The research presented in this thesis may be extended in a variety of ways. Following is a list of some possible works:

1. Development of raw SAR data simulator using current UAV and Microsatellite parameters. In chapter 4, we have demonstrated the capability of the on-board processor for UAV to generate SAR image using raw data from actual UAV based sensor. However, the difference in sensor specifications and flight geometry in our future JX-2 platform havent been addressed. Although theoretically the developed processor will be able to work with minor modifications to its configuration, but in order

to fully calibrate the image quality, simulated data using exact sensor parameters are required.

2. Experiments on other mobile heterogeneous platforms. We strongly suggest to conduct experiments using other mobile heterogeneous platform in order to find out how our algorithms run on different hardware configuration.

3. SAR processing using multiple CPU/GPU hardware simultaneously. In order to handle the challenge of real-time processing of multiple polarization data produced by CP-SAR sensor, we propose the study on using one or multiple mobile heterogeneous computing hardware to handle each channel of CP-SAR output.

Bibliography

[1] V. C. Koo, Y. K. Chan, V. Gobi, M. Y. Chua, C. H. Lim, C.-S. Lim, C. C. Thum, T. S. Lim, Z. bin Ahmad, K. A. Mahmood, M. H. Bin Shahid, C. Y. Ang, W. Q. Tan, P. N. Tan, K. S. Yee, W. G. Cheaw, H. S. Boey, A. L. Choo, and B. C. Sew, "A New Unmanned Aerial Vehicle Synthetic Aperture Radar For Environmental Monitoring," *Progress In Electromagnetics Research*, vol. 122, pp. 245–268, 2012.

[2] J. T. Sri Sumantyo, "Circularly Polarized Synthetic Aperture Radar Onboard Unmanned Aerial Vehicle (CP-SAR UAV)," in *Autonomous Control Systems and Vehicles* (K. Nonami, M. Kartidjo, K.-J. Yoon, and A. Budiyono, eds.), pp. 175–192, Springer Japan, jul 2013.

[3] Z. Fang and J. Xia, "A miniature implementation of air-born SAR real-time processing," in *2009 2nd Asian-Pacific Conference on Synthetic Aperture Radar*, pp. 939–942, IEEE, oct 2009.

[4] J. F. Nouvel and O. R. du Plessis, "The ONERA compact SAR in Ka band," in *Synthetic Aperture Radar (EUSAR), 2008 7th European Conference on*, pp. 1–4, IEEE, 2011.

[5] V. C. Koo and J.T. Sri Sumantyo, *Development of a miniaturized -band UAVSAR.* [IEEE], 2012.

[6] M. di Bisceglie, M. Di Santo, C. Galdi, R. Lanari, and N. Ranaldo, "Synthetic Aperture Radar Processing with GPGPU," *IEEE Signal Processing Magazine*, vol. 27, pp. 69–78, mar 2010.

[7] A. Ahlander and B. Svensson, "Energy-Efficient Synthetic-Aperture Radar Processing on a Manycore Architecture," in *2013 42nd International Conference on Parallel Processing*, pp. 330–338, IEEE, oct 2013.

[8] F. Zhang, G. Li, W. Li, W. Hu, and Y. Hu, "Accelerating Spaceborne SAR Imaging Using Multiple CPU/GPU Deep Collaborative Computing.," *Sensors (Basel, Switzerland)*, vol. 16, p. 494, jan 2016.

[9] W. Sodsong, J. Hong, S. Chung, Y. Lim, S.-D. Kim, and B. Burgstaller, "Dynamic partitioning-based JPEG decompression on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 28, pp. 517–536, feb 2016.

[10] G. Franceschetti and R. Lanari, *Synthetic Aperture Radar Processing.* CRC Press, 1999.

[11] J. C. Curlander and R. N. McDonough, *Synthetic Aperture Radar: Systems and Signal Processing (Wiley Series in Remote Sensing and Image Processing).* Wiley-Interscience, 1991.

[12] M. Y. Chua and V. C. Koo, "FPGA-Based Chirp Generator for High Resolution UAV SAR," *Progress In Electromagnetics Research*, vol. 99, pp. 71–88, 2009.

[13] Y. Yohandri, V. Wissan, I. Firmansyah, J. Sri Sumantyo, H. Kuze, P. Rizki Akbar, and J. T. Sri Sumantyo, "Development of Circularly Polarized Array Antenna for Synthetic Aperture Radar Sensor Installed on

UAV," *Progress In Electromagnetics Research C*, vol. 19, no. January, pp. 119–133, 2011.

[14] K. Suto, J. Sri Sumantyo, C. W. Guey, and K. V. Chet, "FPGA Based Multiple Preset Chirp Pulse Generator For Synthetic Aperture Radar Onboard Unmanned Aerial Vehicle System," in *The 20th CEReS International Symposium, Symposium on Microsatellite for Remote Sensing 2013*, pp. 2–3, 2013.

[15] T. Hirata, K. Namba, H. Ito, B. Setiadi, and J. T. S. Sumantyo, "FFT computation FPGA for Microsatellite onboard Synthetic Aperture Radar," in *International Workshop on Synthetic Aperture Radar (IWSAR2009)*, 2009.

[16] Wumaier Muzapaer, Bambang Setiadi, and J.T. Sri Sumantyo, "Chirp Scaling Algorithm using Matlab for SAR Image Signal Processing," The 18th Remote Sensing Forum, The Society of Instrument and Control Engineers (SICE), pp.5-6, 28 February 2011 (Japan : Chiba)," in *The 18th Remote Sensing Forum, The Society of Instrument and Control Engineers (SICE)*, (Chiba), pp. 5–6, The Society of Instrument and Control Engineers (SICE), 2011.

[17] B. Setiadi, J. Sri Sumantyo, and H. Kuze, "Range doppler synthetic aperture radar signal processing on CUDA," in *the 48th (2010) Spring Conference, The Remote Sensing Society of Japan*, 2010.

[18] B. Setiadi, J. Sri Sumantyo, and H. Kuze, "GPU Based Chirp Scaling Algorithm for SAR Processing," in *IEICE Technical Report Vol. 111, No. 355*, (Tokyo), pp. 77–79, IEICE, 2011.

[19] J. T. S. Sumantyo, "Progress on development of synthetic aperture radar onboard UAV and microsatellite," in *2014 IEEE Geoscience and Remote Sensing Symposium*, pp. 1081–1084, IEEE, jul 2014.

[20] J. S. Sumantyo, "Development of Circularly Polarized Synthetic Aperture Radar onboard Microsatellite for Earth Diagnosis," in *IGARSS 2011. 2011 IEEE International Geoscience and Remote Sensing Symposium. Proceedings*, pp. 929–932, 2011.

[21] Y. Osanai, J. T. S. Sumantyo, and Z. Baharuddin, "Development of spaceborne antenna for circularly polarized SAR using Kevlar honeycomb core," in *2015 IEEE 5th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR)*, pp. 231–234, IEEE, sep 2015.

[22] I. G. Cumming and F. H. Wong, *Digital Signal Processing of Synthetic Aperture Radar Data: Algorithms and Implementation (Artech House Remote Sensing Library)*. Artech House, 2004.

[23] A. Hein, *Processing of SAR Data: Fundamentals, Signal Processing, Interferometry (Signals and Communication Technology)*. Berlin: Springer, 2004.

[24] A. Thompson, E. Cheung, and C. Chang, "Precision SAR processing for Radarsat," in *1995 International Geoscience and Remote Sensing Symposium, IGARSS '95. Quantitative Remote Sensing for Science and Applications*, vol. 3, pp. 2307–2309, IEEE, 1995.

[25] B. Setiadi, L. Bayuaji, J. Sri Sumantyo, and H. Kuze, "Parallel SPECAN Algorithm for SAR Processing on GPU," in *International Conference on Space, Aeronautical and Navigational Electronics (ICSANE 2012)*, pp. 61–65, 2012.

[26] B. Setiadi, Z. Baharuddin, G. F. Panggabean, H. Kuze, and J. T. S. Sumantyo, "Development of Quicklook Processor for Circularly Polarized Synthetic Aperture Radar onboard GAIA-II Microsatellite," 2015.

[27] G. Panggabean, B. Setiadi, and J. S. Sumantyo, "A Single-on-Chip for Onboard SAR Imaging Processor Based on the LEON3," in *The 11th International Conference on Intelligent Unmanned Systems*, 2015.

[28] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*, (New York, New York, USA), p. 483, ACM Press, 1967.

[29] M. D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," *Computer*, vol. 41, pp. 33–38, jul 2008.

[30] X.-H. Sun and Y. Chen, "Reevaluating Amdahl's law in the multicore era," *Journal of Parallel and Distributed Computing*, vol. 70, pp. 183–188, feb 2010.

[31] H. Sips, "Programming Languages for High Performance Computing," in *Aspects of Computational Science a Textbook on High Performance Computing* (A. van der Steen., ed.), pp. 125–194, 1995.

[32] I. Tartalja and V. Milutinovic, "A survey of heterogeneous computing: concepts and systems," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1127–1144, 1996.

[33] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, pp. 32–38, nov 2005.

[34] S. Mittal and J. S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques," *ACM Computing Surveys*, vol. 47, pp. 1–35, jul 2015.

[35] K. Karimi, N. G. Dickson, and F. Hamze, "A Performance Comparison of CUDA and OpenCL," p. 10, may 2010.

[36] B. R. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, "Chapter 2 - Introduction to OpenCL," in *Heterogeneous Computing with OpenCL*, pp. 15–38, 2013.

[37] G. Buttazzo, "Hard real-time computing systems: Predictable scheduling algorithms and applications," *Computers & Mathematics with Applications*, vol. 36, no. 3, p. 126, 1998.

[38] G. Pinitas, *Towards Real-time SAR*. PhD thesis, Delft University of Technology, jul 2014.

[39] P. Meisl, M. Ito, and I. Cumming, "Parallel processors for synthetic aperture radar imaging," in *Proceedings of the 1996 ICPP Workshop on Challenges for Parallel Processing*, pp. 124–131, IEEE Comput. Soc. Press, 1996.

[40] D. Wang and M. Ali, "Synthetic Aperture Radar on low power multi-core Digital Signal Processor," in *2012 IEEE Conference on High Performance Extreme Computing*, pp. 1–6, IEEE, sep 2012.

[41] S. K. Rethinagiri, O. Palomar, J. A. Moreno, O. Unsal, and A. Cristal, "An energy efficient hybrid FPGA-GPU based embedded platform to accelerate face recognition application," in *2015 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XVIII)*, pp. 1–3, IEEE, apr 2015.

[42] M. Pfitzner, S. Langemeyer, P. Pirsch, and H. Blume, "A flexible real-time SAR processing platform for high resolution airborne image generation," in *Proceedings of 2011 IEEE CIE International Conference on Radar*, pp. 26–29, IEEE, oct 2011.

[43] R. R. Hoare and D. Smetana, "Accelerating SAR processing on COTS FPGA hardware using C-to-gates design tools," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, IEEE, sep 2014.

[44] M. Cafaro, I. Epicoco, S. Fiore, D. Lezzi, S. Mocavero, and G. Aloisio, "Near real-time parallel processing and advanced data management of SAR images in grid environments," *Journal of Real-Time Image Processing*, vol. 4, pp. 219–227, aug 2009.

[45] G. Aloisio, M. Cafaro, I. Epicoco, and G. Quarta, "Teaching High Performance Computing Parallelizing a Real Computational Science Application," 2005.

[46] G. Li, F. Zhang, L. Ma, W. Hu, and W. Li, "Accelerating SAR imaging using vector extension on multi-core SIMD CPU," in *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 537–540, IEEE, jul 2015.

[47] C. Clemente, M. di Bisceglie, M. Di Santo, N. Ranaldo, and M. Spinelli, "Processing of synthetic Aperture Radar data with GPGPU," in *2009 IEEE Workshop on Signal Processing Systems*, pp. 309–314, IEEE, oct 2009.

[48] S. Zhao and R. Wang, "A GPU Based Range-Doppler Algorithm for SAR Imaging in OpenCL," in *2011 First International Conference on*

Instrumentation, Measurement, Computer, Communication and Control, pp. 224–227, IEEE, oct 2011.

[49] B. Liu, K. Wang, X. Liu, and W. Yu, “An Efficient SAR Processor Based on GPU via CUDA,” in *2009 2nd International Congress on Image and Signal Processing*, pp. 1–5, IEEE, oct 2009.

[50] X. Ning, C. Yeh, B. Zhou, W. Gao, and J. Yang, “Multiple-GPU Accelerated Range-Doppler Algorithm for Synthetic Aperture Radar Imaging,” *Cell*, pp. 698–701, 2011.

[51] R. Wang, “Processing of SAR Data based on the Heterogeneous Architecture of GPU and CPU,” in *IET International Radar Conference 2013*, pp. 0271–0271, Institution of Engineering and Technology, 2013.

[52] T. M. Benson, D. P. Campbell, and D. A. Cook, “Gigapixel spotlight synthetic aperture radar backprojection using clusters of GPUs and CUDA,” in *2012 IEEE Radar Conference*, pp. 0853–0858, IEEE, may 2012.

[53] A. Fasih and T. Hartley, “GPU-accelerated synthetic aperture radar backprojection in CUDA,” in *2010 IEEE Radar Conference*, pp. 1408–1413, IEEE, 2010.

[54] M. Duersch, “Backprojection for Synthetic Aperture Radar,” 2013.

[55] W. Chapman, S. Ranka, S. Sahni, M. Schmalz, U. K. Majumder, L. Moore, and B. Elton, “Parallel processing techniques for the processing of synthetic aperture radar data on GPUs,” in *2011 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pp. 17–22, IEEE, dec 2010.

[56] Z. Wu, Y. Liu, L. Zhang, N. Li, K. Du, and T. Balz, "Highly efficient synthetic aperture radar processing system for airborne sensors using CPU+GPU architecture," *Journal of Applied Remote Sensing*, vol. 9, p. 097293, apr 2015.

[57] Ming-cong Song, Y.-b. Liu, F.-j. Zhao, R. Wang, and H.-y. Li, "Processing of SAR Data based on the Heterogeneous Architecture of GPU and CPU," in *IET International Radar Conference 2013*, pp. 0271–0271, Institution of Engineering and Technology, 2013.

[58] M. Fatica and E. Phillips, "Synthetic Aperture Radar imaging on a CUDA-enabled mobile platform," pp. 1–5, 2014.

[59] L. Bin, W. Kaizhi, L. Xingzhao, and Y. Wenxian, "An Efficient Signal Processor of Synthetic Aperture Radar Based on GPU," in *EUSAR 2010*, pp. 1054–1057, 2010.

[60] E. Hayden, M. Schmalz, W. Chapman, S. Ranka, and S. Sahni, "Techniques for Mapping Synthetic Aperture Radar Processing Algorithms to Multi-GPU Clusters," in *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2012.

[61] O. Altun, S. Paker, and M. Kartal, "Realization of Interpolation-free Fast SAR Range-Doppler Algorithm Using Parallel Processing on GPU," pp. 998–1002, 2013.

[62] H. Hobooti, *Radiometric correction in range-SPECAN SAR processing.* PhD thesis, University of British Columbia, 1995.

[63] R. Lanari, S. Hensley, and P. Rosen, "Modified SPECAN algorithm for ScanSAR data processing," in *IGARSS '98. Sensing and Managing the*

Environment. 1998 IEEE International Geoscience and Remote Sensing. Symposium Proceedings. (Cat. No.98CH36174), vol. 2, pp. 636–638 vol.2, IEEE, 1998.

[64] A. Vidal-Pantaleoni and M. Ferrando, “On-board medium resolution SAR processing for fast image generation,” *International Journal of Remote Sensing*, vol. 25, pp. 161–176, jan 2004.

[65] W. Caputi, “Stretch: A Time-Transformation Technique,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-7, pp. 269–278, mar 1971.

[66] “CEOS SAR Data Products Format Standards,” tech. rep., 1989.

[67] J. Bennett and I. Cumming, “A Digital Processor for the Production of Seasat Synthetic Aperture Radar Imagery,” *LARS Symposia*, 1979.

Appendix A

Publications List

A.1 Peer reviewed journal papers

1. B. Setiadi, M.Z. Baharuddin, G. F. Panggabean, H. Kuze, and J. T. Sri Sumantyo, "Development of Quicklook Processor for Circularly Polarized Synthetic Aperture Radar onboard GAIA-II Microsatellite," Progress and Communication in Sciences, vol. 2, no. 2. pp. 3238, 08-Jan-2016.

2. B. Setiadi, G.F. Panggabean, J.T. Sri Sumantyo and V.C. Koo, "Development of Raw Data Processing System for JX-2 UAV Using Mobile Heterogeneous Computing" , Journal of Unmmaned System Technologies, July 23, 2016 - Accepted.

3. J. T. Sri Sumantyo, B. Setiadi, D. Perissin, M. Shimada, P. Mathieu, M. Urai, and H. Z. Abidin, Analysis of Coastal Sedimentation Impact to Jakarta Giant Sea Wall using PSI ALOS PALSAR, IEEE Geoscience and Remote Sensing Letters (GRSL), July 14, 2016 Accepted.

A.2 Conference papers

1. B. Setiadi, J.T. Sri Sumantyo, and H. Kuze, GPU Based Chirp Scaling Algorithm for SAR Processing, in IEICE Technical Report Vol. 111, No. 355, 2011, pp. 7779.

2. B. Setiadi, L. Bayuaji, J. T. Sri Sumantyo, and H. Kuze, Parallel SPECAN Algorithm for SAR Processing on GPU, in International Conference on Space, Aeronautical and Navigational Electronics (ICSANE 2012), 2012, pp. 6165.

3. J. T. S. Sumantyo, B. Setiadi, D. Perissin, S. Masanobu, P.-P. Mathieu, and M. Urai, Analysis of land deformation velocity using PSI ALOS PALSAR: Impact of coastal sedimentation to future Jakarta giant sea wall and waterfront city, in 2015 IEEE 5th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR), 2015, pp. 516521.

4. M. Z. Baharuddin, B. Setiadi, J. T. Sri Sumantyo, and H. Kuze, An experimental network analyzer based ISAR system for studying SAR fundamentals, in 2015 IEEE 5th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR), 2015, pp. 103107.

5. Heein Yang, Bambang Setiadi, Josaphat Tetuko Sri Sumantyo, and Jae-Hyun Kim, Image Quality Comparison of Linear Polarized and Circular Polarized SAR , in The 2nd Symposium on Microsatellites for Remote Sensing (SOMIRES 2014), 2014.

6. Dodi Sudiana, Rokhmatuloh, Mia Rizkinia, Ardiansyah, Rahmat Arief, Bambang Setiadi, Luhur Bayuaji, and Josaphat Tetuko Sri Sumantyo, Analysis of Land Deformation on Slope Area using PS InSAR. Case

Study: Bandung Area , in International Conference on Space, Aeronautical and Navigational Electronics 2013, 2013.

7. Nikhil Raj Poudyal, Ryutaro Tateishi, and Bambang Setiadi, Monitoring Glacier Flow Velocity By Sar Interferometry And TextureTracking Method Using Alos Palsar Data Around Mt. Everest Region, in The 34th Asian Conference on Remote Sensing 2013 , 2013.

8. Dodi Sudiana, Rokhmatuloh, Mia Rizkinia, Ardiansyah, Rahmat Arief, Bambang Setiadi, Luhur Bayuaji, and Josaphat Tetuko Sri Sumantyo, Analysis of Land Deformation on Slope Area using PS InSAR. Case Study: Malang Area, in The 2013 International Conferences on Geological, Geographical, Aerospaces and Earth Sciences (AeroEarth 2013), 2013.

9. Luhur Bayuaji, Bambang Setiadi, and Josaphat Tetuko Sri Sumantyo, Continous monitoring of Metropolitan city land deformation by DInSAR technique on L, C and X-band SAR data, case study: Jakarta city, Indonesia, in IEICE Technical Report Vol. 111, No. 355, 2011, pp. 4145.