UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DIGITAL SIGNAL PROCESSOR BASED REAL-TIME PHASED ARRAY RADAR BACKEND SYSTEM AND OPTIMIZATION ALGORITHMS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

XINING YU
Norman, Oklahoma
2017

Downloaded from SHAREOK. Persistent Handle URL: https://hdl.handle.net/11244/52417

DIGITAL SIGNAL PROCESSOR BASED REAL-TIME PHASED ARRAY RADAR BACKEND SYSTEM AND OPTIMIZATION ALGORITHMS

A DISSERTATION APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

_____
Dr. Yan Zhang, Chair

_____
Dr. Robert Palmer

_____
Dr. Nathan Goodman

_____
Dr. Joseph Havlicek

_____
Dr. S. Lakshmivarahan

© Copyright by XINING YU 2017
All Rights Reserved.

Acknowledgements

I extend deep thanks to my academic advisor, Professor Zhang Yan (Rockee), for his invaluable insight and guidance during my entire graduate study at the University of Oklahoma. In Chinese, there is an old saying—One day as a teacher, father for life. Dr. Zhang is a reasonable explanation of those words. He uses his thoughtful, kindness, and wisdom guiding his fresh-off-the-boat students when they arrived on this new continent. It would be my pleasure to know Dr. Zhang, and learn from him both in academic and daily life.

As members of the Intelligent Aerospace Radar Team in the Advanced Radar Research Center (ARRC), Pan Yu, Sudantha Perera, and Nepal Ramesh provided valuable insight in conversations about our research. Many thanks go out to the rest of my faculty committee: Robert Palmer, Joseph Havlicek, Nathan Goodman, and Sivaramakrishnan Lakshmivarahan. This research was funded by NOAA-NSSL and conducted in ARRC. The state-of-art equipment and comfort working environment provide by ARRC would also be a supportive factor that makes sure our research going well.

In the end, I would also like to recognize the support and encouragement of my parents and my wife. They are my beloved ones.

Table of Contents

1. Introduction 1

1.1. Introduction of Phased Array Radar 1

1.2. Challenge and Requirements of Multi-functional PAR Backend 4

1.3. Emerging Technologies for Digital Backend System 10

1.4. Comparison of DSP, FPGA, and GPGPU 13

1.5. Outline of Dissertation 15

2. Signal Processing Algorithms and Processing Chain 19

2.1. Introduction 19

2.2. Digital Beamforming 23

2.3. Pulse Compression 29

2.4. Doppler Processing and Data Corner Turn 31

2.5. Weather and Air-surveillance Data Products 32

2.5.1. Mean Velocity Estimation 32

2.5.2. Spectrum Width Estimation 33

2.5.3. Target Tracking 34

2.6. Advanced Algorithms 42

2.6.1. Model-Based Algorithms and System Optimizations 42

2.6.2. Compressive Sensing for Channel Data Rate Reduction 45

2.6.3. System Optimizations 50

2.6.4. Target Direction Estimation 62

3. System Architectures 69

3.1. Parallelization Model 72

3.2. Data Transportation and Backend Protocol 78
3.3. Processing Units 86
3.4. System Synchronization 89
3.4.1. General Calibration Procedure 89
3.4.2. Backend Synchronization 92
3.5. System Performance Evaluations 96
3.5.1. FFT performance 98
3.5.2. Weighted Dot Multiplication 101
3.5.3. Data Corner Turn 106
4. An Example System Implementation 109
4.1. Architecture Design Considerations 109
4.2. Vendor Selections 113
4.3. Processing Chain Implementation Details 115
4.4. Benchmark Results 118
4.5. Comparison with OpenCL 119
4.6. Summary 122
5. Summary and Future Plans 125
5.1. Considerations of Future Architecture Design 127
5.1.1. System fault-tolerance 128
5.1.2. Scalability 130
5.1.3. Cost 131
5.2. Future Works 132
5.2.1. Optimal task and communication scheduling 133

5.2.2. Cognitive Radar 134

6. References 137

Abbreviation 149

List of Tables

Table 2.1: Computational Complexity for Signal Processing Kernels 23

Table 2.2: Doppler filtering performance measured in GFLOPS per core 31

Table 2.3: Tracking simulation parameters 39

Table 2.4: Parameters used in the MUSIC 65

Table 2.5: Assumption parameters used in the DOA algorithms 66

Table 3.1: Typical COTS Interconnection Fabrics 81

Table 3.2: DSP core performance after mitigating cache misses 105

Table 3.3:Time consumption of corner turn for one beam 107

Table 4.1: Example of PAR system parameters 111

Table 4.2: Communication Data Rate per Stage 111

List of Figures

Figure 1.1: Simplified TR channel diagram for a PAR system 1

Figure 1.2: CPPAR demonstrator operated by OU-ARRC 4

Figure 1.3: Cost-effectiveness and power efficiency comparison 7

Figure 2.1: Canonical front-end and back-end architecture of an HPEC application 19

Figure 2.2: Overview of data cube processing chain in a general PAR 20

Figure 2.3 : Illustration of large-scale PAR overall software system diagram 22

Figure 2.4: Output SNR of the new beamforming method versus SNR of traditional beamforming method 28

Figure 2.5: Output SNR versus loading level 29

Figure 2.6: Reconstruction error vs SNR using CS algorithm 49

Figure 2.7: Computational time comparison of two CS algorithms on AMD Opteron 6128/MATLAB regarding to different degrees of signal sparsity 50

Figure 2.8: Initialization example for $Pam; ch$ 53

Figure 2.9: Simple example of three-channel receiver calibration results obtained by using EM self-calibration algorithm 56

Figure 2.10: Comparison of calibration results based on various initial conditions. (The dash line is the measurement value vs truth level. The solid line is the measured level vs calibration result from three different values of σ.) 57

Figure 2.11 Optimization Procedure 58

Figure 2.12: Comparison of calibration results based on various initial starting condition and optimum finding result 59

Figure 2.13: Calibration results after using optimum result finding procedure 59

Figure 2.14: Calibration result when noise variance=1 60

Figure 2.15: Calibration result when noise variance=2 61

Figure 2.16: Composite beam response for two signals at 10 and 0 degree. The dashed curves are the responses to the individual signals, and solid curve is the composite response 62

Figure 3.1: Top-level system digital array system concept 69

Figure 3.2: Illustration of the MTCA architecture in a PAR 71

Figure 3.3: (a) Partition input and output matrices into 2×2 submatrices. (b) A decomposition of matrix multiplication into four tasks based on (a) 72

Figure 3.4: Example of Parallel Quicksort 74

Figure 3.5: Ping-Pong Buffering Mechanism 74

Figure 3.6: Examples of round-robin and pipeline scheduling 75

Figure 3.7: Levels of parallelism 76

Figure 3.8: Typical interconnection fabric 79

Figure 3.9: MTCA backplane configuration 80

Figure 3.10: point to point connectivity between port 2 and 3 82

Figure 3.11: Typical PCI Express system topology 83

Figure 3.12: Data throughput experiment results in MTCA-based SRIO testbed 85

Figure 3.13: MTCA based processing unit 87

Figure 3.14: Simple example of a PU-based architecture 88

Figure 3.15: General system calibration procedure for DAR and the focus of this work 90

Figure 3.16: Measurement of radiation pattern from a phased array antenna at the near-field range 91
Figure 3.17: PU frontend AMC module architecture 93
Figure 3.18: Frontend in-chassis synchronization timing sequence 94
Figure 3.19: Example timing sequence of multi-chassis synchronization 95
Figure 3.20: TI C66x DSP Hierarchical Cache 97
Figure 3.21: FFT performance for different range gate numbers 100
Figure 3.22: L1D cache architecture 102
Figure 3.23: Weighted Dot Product Example 103
Figure 3.24: Memory Layout after two iterations 104
Figure 3.25: Computing performance of a DSP-core versus the number of range gates 105
Figure 3.26: An illustration of front-end data transmission strategy 107
Figure 4.1: System Network Topology 109
Figure 4.2: Clock Synchronization classification 112
Figure 4.3: An example of front-end processing platform 115
Figure 4.4: Speedup and efficiency of beamforming implementation 117
Figure 4.5: Real-time system timeline for the example backend system 118
Figure 4.6: Beamforming kernel performance using Open CL 120
Figure 4.7: Comparing OpenCL performance to manually optimized code for beamforming 121
Figure 4.8: Pulse compression performance using OpenCL (8192 range gates) 122

Figure 4.9: Comparing OpenCL performance to manually optimized code for pulse compression 122

Figure 5.1: The triple-duplex approach to redundancy 129

Figure 5.2: Data flowing graph in PAR frontend 131

Figure 5.3: Processing cycle of a cognitive radar 135

Abstract

This dissertation presents an implementation of multifunctional large-scale phased array radar based on the scalable DSP platform.

The challenge of building large-scale phased array radar backend is how to address the compute-intensive operations and high data throughput requirement in both front-end and backend in real-time. In most of the applications, FPGA or VLSI hardware are typically used to solve those difficulties. However, with the help of the fast development of IC industry, using a parallel set of high-performing programmable chips can be an alternative. We present a hybrid high-performance backend system by using DSP as the core computing device and MTCA as the system frame. Thus, the mapping techniques for the front and backend signal processing algorithm based on DSP are discussed in depth.

Beside high-efficiency computing device, the system architecture would be a major factor influencing the reliability and performance of the backend system. The reliability requires the system must incorporate the redundancy both in hardware and software. In this dissertation, we propose a parallel modular system based on MTCA chassis, which can be reliable, scalable, and fault-tolerant.

Finally, we present an example of high performance phased array radar backend, in which there is the number of 220 DSPs, achieving 7000 GFLOPS calculation from 768 channels. This example shows the potential of using the combination of DSP and MTCA as the computing platform for the future multi-functional large-scale phased array radar.

1. Introduction

1.1. Introduction of Phased Array Radar

Phased Array Radar (PAR) is an electronically scanned array with multiple numbers of antennas. Compared with traditional mechanical beam steering radar, PAR can generate a focused beam by applying a weight to each antenna and the beam direction can be steered by adjusting the weights. Figure 1.1 shows a simplified transmit-receive (TR) channel diagram for a modern PAR. The overall PAR system comprises three sections: a phased array antenna manifold, an RF front-end, and a processing backend. The array manifold contains a number of radiating elements, which can be different types of antennas. The shape of array manifold can be linear, planar, or conformal. The linear and

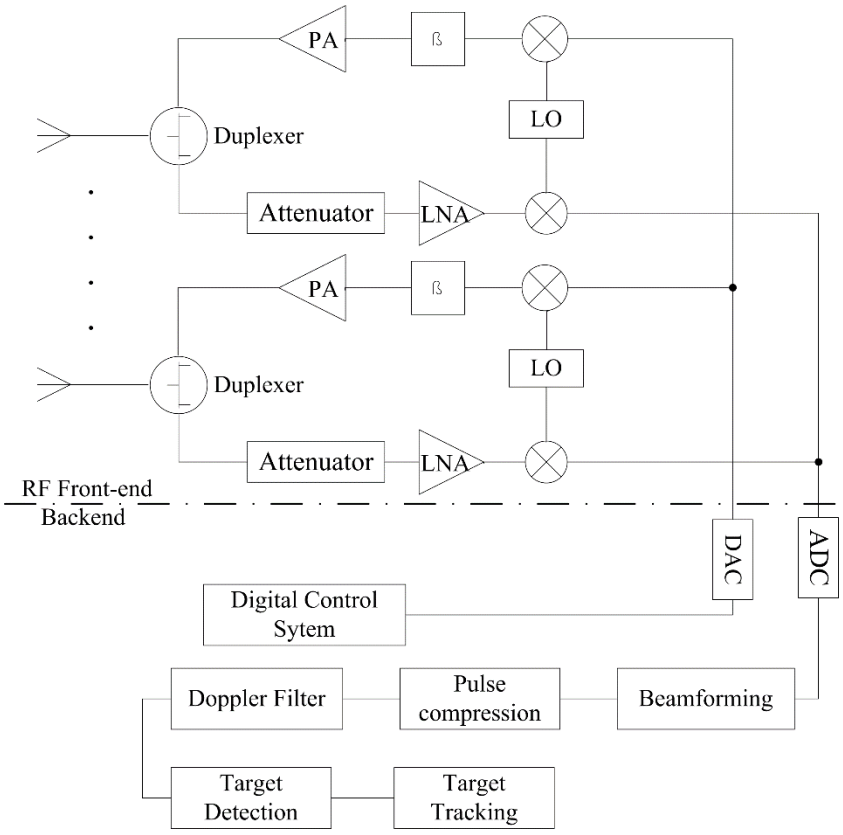


Figure 1.1: Simplified TR channel diagram for a PAR system

planar arrays have been widely used in military and civilian applications for many years, which is more mature than the conformal array. However, circular or cylindrical arrays have found many applications in communications, direction finding, missile guidance and recently in weather radars. RF front-end is responsible for the signal generating, signal transmitting and receiving, and up/down converting. In digital PAR systems, the radar pushes the backend system closer to the antennas. Such front-end systems are mixed-signal systems responsible for transmitting and receiving radio frequencies (RF), digital in-phase and quadrature (I/Q) sampling, and channel equalization that improves the quality of signals. An example application is the Space Fence test facility built by Lockheed Martin. The radar system in this facility is fully digital array composed by multiple numbers of front-end transmit-and-receive line-replaceable unit [1]. Meanwhile, digital PAR backend systems control the overall system, prepare to transmit waveforms, transform received data for use in a digital processor, and process data for other functions, including real-time calibration, beamforming, and target detection/tracking.

As the RF front-end becomes more digitalized and the increased performance of backend processing units, Multifunctional Phased Array Radar (MPAR) is more feasible by programmable RF and processing parallelization. This improvement makes possible to combine multiple types of radar in one unit, which is also a way to enhance the efficiency of spectrum utilization. For example, current U.S. government operates several unique types of radars that provide weather, air traffic control, and homeland defense missions. It is possible to reduce the total number of radars and spectrum footprints with a single network of MPAR, which could potentially save billions of dollars [2]. Moreover, the electronically scanned antennas can reduce the maintenance cost over mechanically

steered antennas by the absence of moving parts, and the radar system would be still functional even if 20% of the TR modules fail [3]. With the fast development of commercial wireless industry, various companies and agencies made their RF equipment and required them to work on a stand-alone spectrum to avoid the interference. Therefore the spectrum becomes crowded and makes the spectrum a highly-priced product in the market. For example, a 65 MHz of spectrum in L-band are sold by $45 billion in 2015 [4]. Although the defense or national weather radars have the privilege of using some specific spectrum, if government choose to move the working spectrum out of a crowded area, they can use the selling money to update their system without raising fund from other places. Thus, the MPAR would be a feasible, reliable and cost-effectiveness system.

Many types of research have begun in academia, industry, and government to identify technical challenges and risks, and demonstrate their technologies for needs from both weather and the airport surveillance. MIT Lincoln laboratory had a concept study for the requirement of the aircraft surveillance and weather observation [5], in which it purposed a planar PAR with four antenna faces. Each face contains roughly 20,000 elements with 10 Watts peak power. In 2015, MIT Lincoln lab had built a 10-panel prototype array with the dual-polarization capability to refine system requirements and to quantify performance for weather observations. This prototype has 640 elements with 3.5 kilo-Watts peak power at antennas, working at S-band [6]. As mentioned before, the problem of using planar array in the dual polarization application is E_v and E_h are skewed when beam is not perpendicular to the array face. At the meantime, the Cylindrical Polarimetric Phased Array (CPPAR) has recently been introduced for MPAR. In the University of Oklahoma, a demonstrator of the CPPAR is designed by Advanced Radar

Research Center (ARRC) to prove the concept of polarimetric measurements of actual weather and demonstrating a multi-functional PAR. CPPAR has 1824 elements separated into 96 columns. Each column has 80 Watts peak power, working at 2900 MHz. Figure 1.2 shows the picture of CPPAR currently operated by ARRC.



Figure 1.2: CPPAR demonstrator operated by OU-ARRC

1.2. Challenge and Requirements of Multi-functional PAR Backend

The concept of MPAR is associated with many technical challenges which remain to be solved. This work mainly focuses on the backend aspect. A canonical PAR processing platform contains a front-end component that performs basic array signal processing, which requires relative easy but a significant amount computing throughput. A more advanced backend performs knowledge-based processing requires complex operations but the relatively small amount of computing throughput. For example, [5] proposed a

400-channel PAR with 1 ms pulse repetition interval (PRI); assuming 8,192 range gates, each range sample uses 8 bytes length in memory. For each PRI, the throughput in the front-end can reach up to 5.24 GB/s. As the requirements for such data throughput are extraordinarily demanding, at present, such computing performance requires digital I/Q filtering to be mapped to a fixed set of gates, look-up tables, and Boolean operations on the field-programmable gate array (FPGA) or very-large-scale integration (VLSI) with the full-custom design [7]. After front-end processing, data are sent to the backend system, in which more computationally intensive functions are performed. Compared with FPGA or full-custom VLSI chips, programmable processing devices such as digital signal processors (DSPs) offer a high degree of flexibility, which allows designers to implement algorithms in a general-purpose language (e.g., C) in backend systems. For application in aerospace surveillance, target detection and tracking are thus performed in the backend. Target tracking algorithms, including the Kalman filter and its variants, predict future target speeds and positions by using Bayesian estimation [8], whose computational requirements vary according to the format and content of input data. Accordingly, detection and tracking functions require processors to be more capable of logic and data manipulation, as well as complex program flow control. Such features are different from those of baseline radar signal processors, in which the size of data involved dominates the throughput of processing [9]. As such, for tracking algorithms, a general purpose processor or Graphics Processor Unit (GPU)-based platform is more suitable than FPGA or DSP. In summary, in radar backend processing, *hybrid solutions* need to be developed that exploit the advantages of each type of processor units.

Normally, the hybrid backend system is based on a modular design concept, in which one or more processors are placed in one extension card. The modular architecture is scalable, which allows the sub-system to be upgraded with minimal impact on the overall system. However, the drawback of modular architecture is the communication requirement among the extension cards and the complexity of software design when the granularity of processing becomes small. The granularity of processing is defined according to the size of a processing assignment that forms a part of the entire task. Although finer granularity allows designers to attune the processing assignment, it also poses the disadvantage of increased communication overhead within each unit [10]. To balance computation load and real-time communication in one extension card, the ratio of the number of computation operations to communication bandwidth needs to be checked carefully. For example, in a 6678 Evaluation Module (Texas Instruments), which has eight C66xx DSP cores, contains 24 DSP cores and four ARM cores in a single board. Texas Instruments claims that each C66xx core has 16 Giga floating point operation per second (GFLOPS) at 1 GHz [11]. On this board, it has four-lane SRIO (Gen 2) link, which has a theoretical link speed up to 1,600 MB/s in NWrite mode; since the single-precision floating point format (IEEE 754) [12] occupies 4 bytes in memory, the SRIO link convoys 400 million floating point data per second. The ratio of computation to bandwidth is 40 [8], meaning that the core can perform up to 40 floating point operations for each piece of data that flows into the system without halting the core-to-core communication link. As such, when the ratio reaches 40, the processor achieves an optimal balance between real-time computing and communication. To achieve this optimization and efficiency,

technologies needed are optimized algorithms, computing resources optimization, and ensuring that I/O capacity reaches its peak level.

The key design tradeoffs depend on the constraints imposed by the radar system. In some applications, such as airborne radar, the size, weight, and power consumption (SWaP) is a constraint and requires the designer to address the SWaP challenges by balancing the performance and form factors. In contrast, the SWaP requirements are likely to be relaxed for ground-based applications, but the cost of the platform may need to be limited. As one of the tradeoffs, Figure 1.3 shows the comparison among different type of processor selections in terms of power consumption, cost, and computing power. Based on such comparisons, a designer may choose the proper type of processor to fulfill the processing task according to the requirement. For example, FPGA is more appropriate

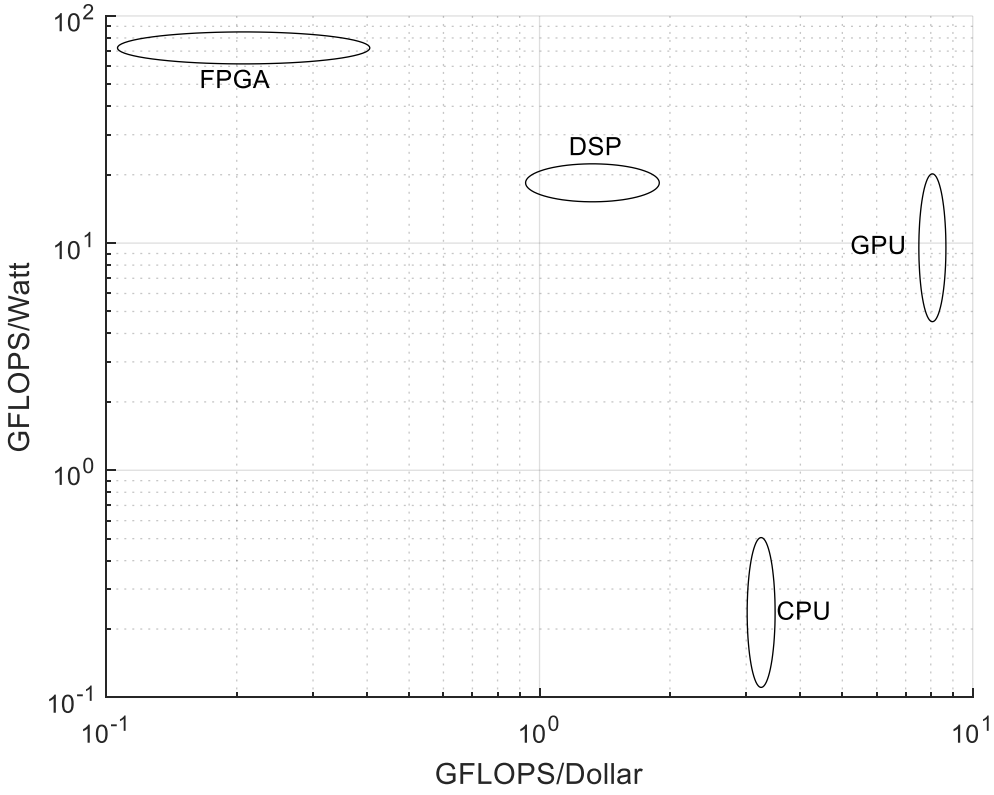


Figure 1.3: Cost-effectiveness and power efficiency comparison

for applications those require highest computation throughput at the lowest unit SWaP. On the other hand, CPU is suitable for the cost-sensitive scenarios. DSP or GPU can be a better choice when a balance between cost and performance is needed. Even though FPGA and ASIC are relatively difficult to be programmed than DSP and the cost is higher. However, when a large volume of devices are in demand and high computing intensity is required, FPGA and ASIC will be used in the most demanding portion of the system to keep the power consumption and form factor under control. For example, [5] mentions that each MPAR contains 1,200 channel and more than 200 MPAR are needed in the future. Based on this volume, the cost of using FPGA and ASIC would be dropped. So, in the final phase of the product, FPGA or ASIC would be a better choice.

Besides the high data throughput requirements for the backend of a PAR system, in the front-end, there are also issues related to antenna calibration, RF distortion, and multi-channel synchronization. Compared with PAR, reflector dish radars have mechanically steered the antenna to point the radar beam in a specific direction, so the characteristics of the beam are the same during scanning. However, the beam characteristics of a PAR change with the pointing directions as well as the performance of its transmit and receive elements. For dual-polarized PAR, antenna beams at each pointing direction need to be calibrated and monitored. Moreover, distributed array architecture, such as the distributed local oscillator (LO) in digital arrays, leads to small variations in signal response among different channels. Although various antenna calibration methods, such as peripheral fixed probes and near-field measurement, can help channel equalization, those procedures are all complex and need a clutter-free

environment. Once a PAR is deployed, re-calibration may be difficult, which brings the challenge of achieving stable system performance.

In practice, due to the nonlinear behaviors of RF hardware, the relative phase, and amplitude values deviate considerably during the time and among different channels. This distortion reduces the dynamic-range and downgrades the data quality, so the predicting, assessing and quantifying these effects would be necessary for calibrations. Several linearization techniques have been developed to inversely models the behavioral models of system's gain and phase characteristics, such as feedforward linearization [13], RF predistortion [14], Cartesian feedback [15], and digital predistortion. These methods require extra predistortion circuits and feedback from the output of RF system, which increase the complexity of RF front-end design and cost.

The transmitters, receivers, and other sequential circuits in PAR channels are synchronized by a Local Oscillator clock (LO). In other to synchronize LOs in different channels, a global clock signal is distributed so that the data acquired from multiple channels are correlated in time. A reliable clock network is required to deliver the clock signal to all the channel circuit components. In a multiple chassis PAR (especially for DAR) application, the interconnection clock distribution network is complex and the control of arrival times of the global clock at different LOs becomes difficult. If not properly controlled and monitored constantly, the clock skew can adversely affect the performance or even cause erratic operations of the systems.

The design of clock distribution network poses a formidable challenge of considering the variations in interconnect parameters. For example, the length of between the source of global clock to each LO may be varied, and the power supply noise on each

LO affects the clock jitter, which, in turn, affects the arrival time of the global clock. Those instabilities would make the overall system unreliable, which is one of the lessons learned from OU's first version CPPAR development. The most common synchronization solution is using Network Time Protocol, which synchronizes each client by using the UDP packets over Ethernet. The drawback of this solution is the low accuracy, ranging from 5 to 100 ms [12]. Another more accurate method is using the IEEE 1588 Precision Time Protocol (PTP) standard [13], which can achieve sub-microsecond synchronization [14]. However, to implement PTP, an extra dedicated hardware and software are needed, which would increase the complexity and cost of front-end system.

1.3. Emerging Technologies for Digital Backend System

The digitization of transmitting and receiving signals at the element level opens the door to new processing technologies for the phased array system. In the RF front end, the state-of-art Gallium nitride technology outperforms traditional CMOS power amplifiers in terms of high power density and smaller die areas [16]. With the fast development of integrated chip industry, the cost and size of chips are reduced, which makes the radar system smaller, more powerful, and affordable for the customers from the consumer electronics market. For example, in the automotive industry, the frequency modulation continuous wave radar has been widely utilized in the forward collision avoidance system and active cruise control system. Those mass productions would further bring down the cost of the radar and make the radar product more affordable.

In the backend processing platforms, a high-performance embedded computing (HPEC) platform contains microprocessors, network interconnection technologies such

as those of the Advanced Telecommunications Computing Architecture (ATCA) and Micro Telecom Computing Architecture (MTCA), and management software that allows more computing power to be packed into a system with smaller SWaP. Such designs achieve compatibility with industrial standards and reduce both the cost and duration of development. MTCA and ATCA contain groups of specifications that aim to provide an open, multi-vendor architecture that seeks to fulfill the requirements of a high throughput interconnection network, increase the feasibility of system upgrading and upscaling, and improve system reliability. In particular, MTCA specifies the standard use of an Advanced Mezzanine Card (AMC) to provide processing and input-output (I/O) functions on a high-performance switch fabric with a small form factor.

Within the backend system, the processing chipsets have also evolved rapidly. Not only the data throughput of widely-used FPGA and DSP are increased dramatically, but also a new type of processor has emerged as a new tool to accelerate radar signal processing tasks. Traditionally, GPU has been used as a special-purpose device whose function is to accelerate the graphics pipeline for the video games in the PC environment. With the fast development of GPU and its standardized application programming interfaces (API), such as OpenGL, DirectX, and CUDA, GPUs have moved beyond graphics applications to become powerful floating point processing units [17]. As GPU is a native hardware for floating point operations, many areas of study related to a significant computing throughput requirement, such as machine learning, computational biology [18], and computer vision, has begun to adopt newer signal processing algorithms to GPU. Moreover, GPUs offer good backward compatibility than DSP and FPGA.

Applying machine learning algorithms in radar signal processing, especially in the target recognition area, has become a new trend [19]. For example, cognitive radar [20] is designed to intelligently perceive, track, and classify the targets from the past experiments, which is realized by the Bayesian approach. The cognitive aspect is manifested in the form of the cognitive signal processing cycle, in which an adapt waveforms are generated to illuminate the non-stationary environment. Within this cycle, active target classifications are optimized based on prioritized system objectives, understanding of the observation environment, and other forms of prior knowledge. With each illumination, the system improves the understanding of surrounding area in response to collected data and other information [21]. The cognitive radar uses the scene analysis to develop an appropriate statistical model to describe the information content of received signal on clutter, targets, or other false alarms. For example, when there is a target moving on an ocean surface, the Doppler spectrum of clutter would be relatively smooth across a wide range of the spectrum, whereas the spectral of the target would be appeared as a line component [22]. Moreover, when the power level reflected from the target is small compared with clutter, the cognitive radar needs an enhancement to extract the target information from the clutter. Thus, three statistic models are developed to classify the different conditions: clutter-statistics described by the F-distribution $F_{2,2k}(z)$, where z is the power of spectrum and, k is the number of neighboring Doppler bins [23], target-pulse-clutter statistics described by scaled F-distribution $\frac{1}{\gamma}F_{2,2k}\left(\frac{z}{\gamma}\right)$, where γ is the power ratio of target to clutter, and target motion described by the Gaussian distribution .

1.4. Comparison of DSP, FPGA, and GPGPU

The signal processing tasks in the DAR can be implemented with different types of processors; each has its own benefits and limitations. FPGA has the advantage of low non-recurring expenses and reconfigurability with high throughput. Different from other types of general purpose processors, FPGA is programmable device based on user applications. In the FPGA, multiple logic blocks are connected with programmable interconnect point, in which the designer can implement algorithms by configuring logic blocks and routing the data traffic through interconnect points [24]. Since a designer can control the hardware structure implemented in FPGAs, the computation load and communication throughput may be balanced better than other General-Purpose Processors (GPPs). For example, the bus width and the processing speed is fixed in the GPP, so the performance of the processors may be compromised when the communication requirement is more stringent than computing requirements,

Before the proliferation of FPGA applications, DSP has been the primary choice for signal processors. Within DSP, multiple numbers of Multiply Accumulate Engines (MAC) are used for parallel processing. For example, a TI C66x core [11] contains one MAC, which can perform four single precision floating point multiplications and two single precision floating point additions in one clock cycle. Since a DSP operates on *instructions*, the programming mechanism can be a high-level language for fast deployment or assembly language for higher performance requirements. Those two choices provide the flexibility for the designers compared by using only one mechanism--HDL on FPGA. However, with hundreds of MACs, FPGA can be built into a more powerful parallel computing platform than DSP. Incorporating so much computing power

in one chip makes the power consumption in FPGA much higher than DSP. Moreover, FPGAs are usually more expensive than DSP in terms of GFLOPS per dollar. Thus, DSP would be a better choice for applications which are cost sensitive and have strict power budget.

As a GPP, a CPU is designed to follow general purpose instructions among different types of tasks and thus allow the advantage of programming flexibility and efficiency in flow control. However, since CPUs do not accommodate for a range of scientific calculations, GPUs can be used to support heavy processing loads. The combination of a CPU and GPU offers competitive levels of flow control and mathematical processing, which enable the radar backend system to perform sophisticated algorithms in real-time. With the increasingly friendly programming environment and standardized API, CPU-GPU becomes easy to be programmed and maintained, compared with FPGA and DSP. Moreover, CPU-GPU has a better performance than FPGA and DSP in term of GFLOPS per dollar. The drawback of the combination, however, is its limited bandwidth for handling data flow in and out of the system. CPUs and GPUs are designed for a server environment, in which Peripheral Component Interconnect Express (PCIe) can efficiently perform point-to-point for onboard communication. However, PCIe is not suitable for high throughput data communication among a large number of boards. If the throughput of the processing is dominated by the size of data involved, then the communication bottleneck downgrades the computing performance for a CPU–GPU combination.

Therefore, when signal processing algorithms have demanding communication bandwidth requirements, DSP and FPGA are better options, since both can provide

significant bandwidth for in-chassis communication by using SRIO while at once achieving high computing performance. FPGA is more capable than DSP of providing high throughput in real-time for a given device size and power. When the DSP cluster cannot achieve performance requirements, the FPGA cluster can be employed for critical-stage, real-time radar signal processing. However, such improved performance comes at the expense of limited flexibility in implementing complex algorithms [7]. If FPGA and DSP both meet application requirements, then DSP can be a more preferred option given its reduced cost and less complicated programmability. The CPU-GPU combination can be used in the applications, in which there is a significant computing load requirement but with a more flexible timeline and power consumption requirements.

1.5. Outline of Dissertation

This dissertation presents the method of realizing digital phased array radar functions in a scalable, compact, and power efficiency form factor by using commercial off-self products. The ways of implementing fundamental signal processing algorithms in real-time on DSP platforms are discussed. An HPEC platform for parallel backend processing is introduced as an example, and novel algorithms to self-calibrate the array system are investigated. Finally, an example of system implementation is elaborated to demonstrate the performance of our purposed HPEC solutions.

Chapter 2 presents the computational aspect of canonical radar signal processing algorithms and procedures, focusing on computational complexity, algorithm decomposition, and mapping of algorithms onto embedded hardware processors. A new self-calibration technique based on Expectation Maximization (EM) algorithm is studied. Typically, antenna calibration needs a controlled environment and additional hardware,

which is difficult to be performed in the field. The proposed self-calibration method would use EM algorithm to build up a Bayesian model to find the ground truth value based on thousands of observation data from the antenna. Moreover, compressive sensing is introduced to improve the range resolution. In this chapter, some of the research results are based on the past publication [25, 26].

Chapter 3 presents an efficient and scalable backend system architecture design for a large-scale PAR, which achieves high throughput and computing performance. The basic signal processing chain, including beamforming, pulse compression, and Doppler filtering are mapped to processing units in parallel for the real-time processing. More advanced adaptive processing algorithms can also be implemented on this HPEC testbed. Other radar applications, such as Synthetic Aperture Radar (SAR) can benefit greatly from this design as well. Our approach integrates multiple DSPs by using SRIO links as part of commercial-off-the-shelf (COTS) MTCA chassis. In a digital array radar (DAR) containing hundreds of channels, a highly accurate synchronization technique is critical to the system stability and performance since if transmitters or receivers are out of phase, a focused beam cannot be reliably formed and the SNR would be reduced. We developed a synchronization procedure with nano-seconds level accuracy to ensure the backend system is synchronized. Compared with other synchronization techniques, such as Network Time Protocol (NTP) and Precision Time Protocol (PTP), our method is more reliable, convenient, and accurate. In this chapter, some of the research results are based on the past publication [27].

Chapter 4 presents key benchmark results for radar processing algorithms to investigate the performance of the backend processing platform design, and proves that

the design can meet the real timeline requirements for a large-scale PAR containing 768 digital channels with 4,096 range gates. The system architecture allows adjusted processing power, which requires each processing board on the platform operates independently and be “hot swappable.” Moreover, the architecture needs to provide enough bandwidth for the large data communication among different computing units. Based on those considerations, we use a product from various vendors and integrate them with hybrid backplanes, which proves to enhance the computing performance through benchmark results. Lastly, benchmark performance results of using “bare-bone” parallelism method and standard libraries (such as OpenCL) are compared.

In the end, Chapter 5 would summarize the architecture design consideration for the multi-functional PAR, in which the designer must diligently research the solution and ensure a predictable degree of operational continuity during production hours. Without a good underlying infrastructure and with poor planning, a single hardware failure in the computing environment could affect the system ability to continue the service. Although there are many considerations to a PAR system, the following are some of the most important aspects:

- System Fault-tolerance: A fault-tolerant system has redundant hardware components inside to withstand hardware failure. When the system encounters a hardware failure, the application should remain operational or may be degraded, while the system is repaired.
- Scalability: A scalable system is one whose performance can be increased, or decreased, after adding or removing proportional hardware without changing the framework of the infrastructure components.

- Cost-effective: Using less money to make more outcomes is the goal that every system designer wants to achieve. By doing so, the designer should compare different types of technologies and chose the one that can benefit the system most.

2. Signal Processing Algorithms and Processing Chain

2.1. Introduction

With the development of the higher level of functionality and complexity in signal and image processing applications, the computing throughput in HPEC becomes more demanding with hard-real-time deadlines and stringent form factors. Figure 2.1 shows a top-level structure for a typical HPEC application. The HPEC can be divided into three parts: sensors, a front-end signal processing and a backend data processing. For the PAR application, the sensor could be patch antenna or reflect array [28, 29]. The front-end is to transmit or receive the signal to/from the outside by using the appropriate sensors, remove noise and interference from the signal, and extract the useful information from a large amount of received data. The purpose of the backend is to further refine and classify the information into different categories, convert the numerical information into readable, user-friendly data, and estimate the status of the future targets based on the current result.

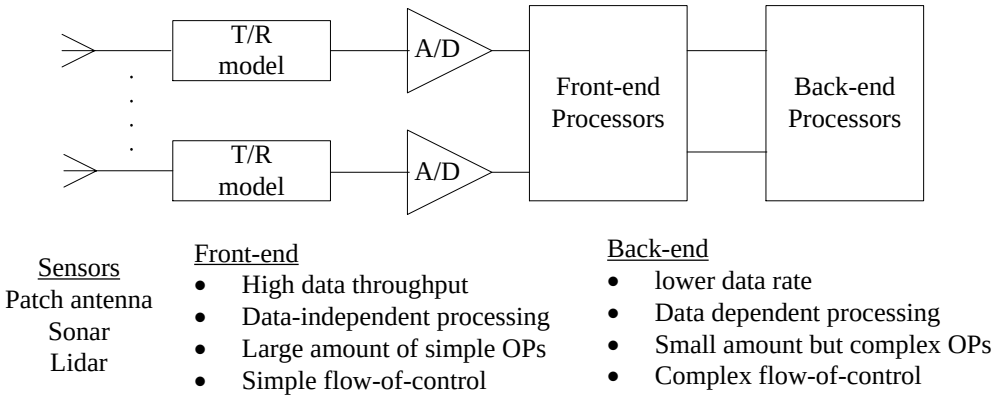


Figure 2.1: Canonical front-end and back-end architecture of an HPEC application

In the PAR application, multiple pulses are transmitted within one Coherent Pulse Interval (CPI). The pulses in the same CPI are phase-coherent, in which each pulse starts with the same phase and reflects back from targets with the relatively small differences in phase. The time interval between transmitted pulse is Pulse Repetition Interval (PRI), which determines the maximum unambiguous range. After the radar antenna receives the reflected signal, the signal would proceed through receivers that perform down-conversion and band-pass filtering, and input to the front-end and back-end processing platform, in which more complex signal processing tasks are performed and output the detection result to the users. In the front-end, three fundamental or general-purpose processing tasks would be conducted— beamforming, pulse compression, and Doppler filtering, and then the result would be feed to the backend platform, in which the types of processing tasks are based on the requirements of different applications. For example, in the weather radar processing, it focuses on the information related to volume targets and analysis the spectrum of the target velocities. In contrast, the aircraft surveillance is committed to giving a good estimation of the position and velocity of the aircraft targets.

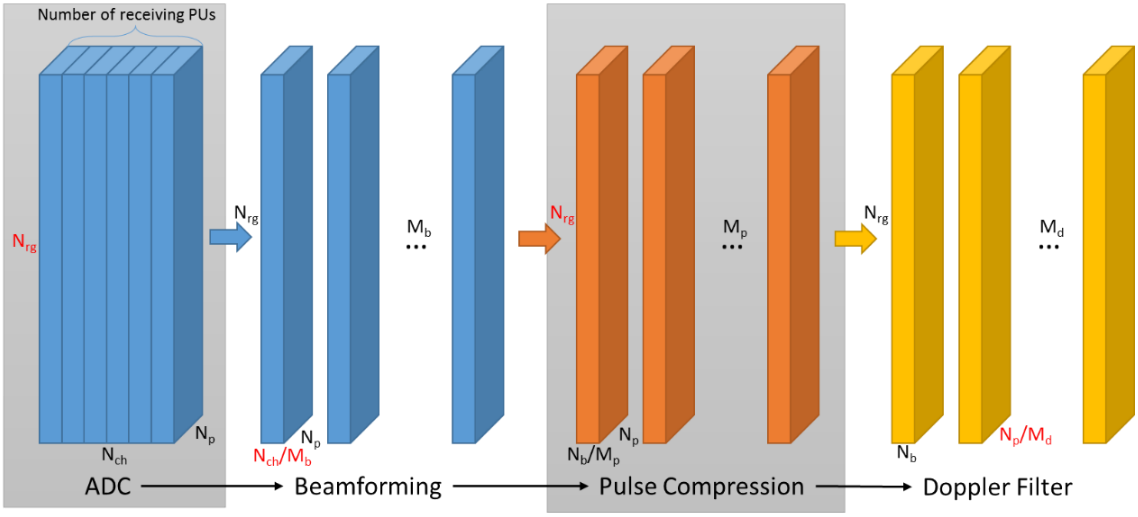


Figure 2.2: Overview of data cube processing chain in a general PAR

Figure 2.2 shows the *data cube* processing chain in the PAR front-end. The received data from the array manifold and front-end electronics are organized into three-dimensional data cubes, and N_{rg}, N_{ch}, N_p, and N_b represent the total number of range gates, channels, pulses, and beams, respectively. When any of those four numbers are red, the data are aligned in their corresponding dimensions. After the data cube is digitized by ADCs, the data is aligned in the dimension of range gate. Before doing further processing, the large data cube would be decomposed to several small portions for the purpose of parallel computing and re-align the data in the channel domain to facilitate the following beamforming processing. The beamforming stage transform spatial domain signal into beam-space domain, creating a set of focused beams. Another data corner-turn is applied at the output of beamforming to align the data in the range gate dimension. The pulse compression stage concentrates the signal energy spread over the entire transmitted waveform into a short pulse response to increase the SNR and sensitivity. A third data corner turn is performed to transform the data from channel to pulse dimension. The Doppler filter stage determine the radial velocity of targets relative to the radar array by applying FFT across the pulses within one CPI. At the output of the Doppler filter, the data cube has dimensions of number of range gates × the number of beams × the number of Doppler bins. After the Doppler processing, the data cube is converted into a data set containing the information about the position and velocity of the targets, from which the processing tasks in the backend can further extract information based on the requirements of applications by using more complex signal processing algorithms.

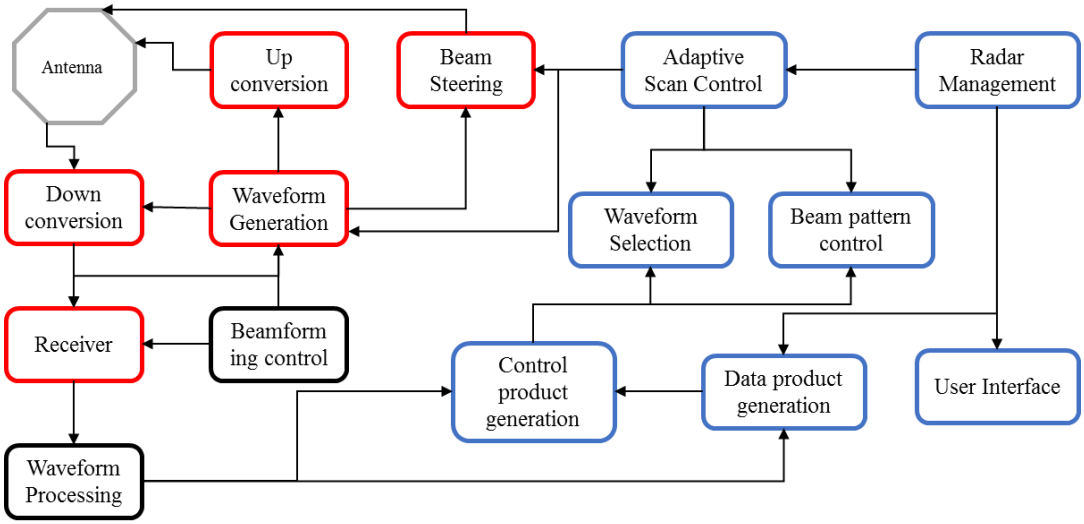


Figure 2.3 : Illustration of large-scale PAR overall software system diagram

Figure 2.3 shows the large-scale PAR overall software diagram, in which the red, black, and blue boxes represent the corresponding tasks performed by the FPGA, DSP, and GPGPU platforms respectively. The selection of different types of the processor is based on the requirements of processing tasks mentioned in Section 1.4. The data cube is formed in the FPGA platform, red boxes, and processed on the DSP platform, black boxes. This dissertation focuses on performing waveform processing and beamforming control task on DSP. The following sections in this chapter would be organized as: Section 2.2-2.4 introduce the canonical radar algorithm in the processing chain and study the computing complexities of these algorithms. After the signal processing chain, the data processing tasks including weather data product generation and target tracking are discussed in Section 2.5. In Section 2.6, several advanced algorithms are introduced, aiming to improve the performance of overall processing chain.

Before calculating the computing complexity of each algorithm in the following sections, it is necessary to list the computational complexity expressions for several fundamental *signal processing* kernels for both real and complex values in advance. The

discussion of the complexity of each processing kernel is beyond the scope of this dissertation; a comprehensive discussion can be found in [30]. The following sections would use the computational complexity listed in Table 2.1 as a reference. Moreover, we use Giga Floating-point Operations (GFLOPS) [31] as a metric to measure the benchmarks of digital PAR backend system performance.

Table 2.1: Computational Complexity for Signal Processing Kernels

Signal Processing Kernel	**Computational Complexity**	
	Real value	Complex value
Matrix Multiplication	$2mnp$	$2mnp$
Fast Fourier Transform	$\frac{5}{2}nlog_2n$	$5nlog_2n$
Forward or back substitution	n^2	$4n^2$
Eigen-decomposition	$9n^3$	$23n^3$
For the matrix multiplication, the matrices are of dimensions $\boldsymbol{m} \times \boldsymbol{n}$ and $\boldsymbol{n} \times \boldsymbol{p}$. For the FFT, the vector size is $\boldsymbol{n}$. The lower triangular matrix used in forward or back substitution is $\boldsymbol{n} \times \boldsymbol{n}$.		

2.2. Digital Beamforming

The procedure of beamforming is to convert the data from channel data (range gate) to beamspace, steer the radiating direction, suppress sidelobes and interferences by applying the beamformer weight, W_i, to received signal, Y_i, indicating in Equation (2.1), in which Θ is the beam pointing angle indicator and Ω is the total number of channels. The computation complexity of Equation (2.1) can be determined as follows: first, each complex multiplication requires four floating point multiplications and two floating point

additions, for a total of $4 + 2 = 6$ real flops. There are two complex additions for summing of each channel. Hence, for a complex beamforming, the complexity formula arrived at is $(6 + 2) \times \Omega \times N_{rg} = 8N_cN_{rg}$, where N_{rg} is the number of range gates. The above calculating complexity evaluates the throughput of beamforming at the time interval, T, as $\boldsymbol{(8N_cN_{rg})/T}\ FLOPS$.

$$Beam^{\Theta} = \sum_{i=1}^{\Omega} W_i^{\Theta} Y_i \tag{2.1}$$

In a large-scale PAR, there will be thousands of range gates and channels, which bring huge computing burden for a front-end computing platform. For example, in [5], each face of the array contains 20,000 channels, and suppose the number of range gates is 1,000 and pulse time interval is $one\ ms$. If all the elements are digitalized, the throughput of the beamforming would be 160 GFLOPS for a single beam forming. It is impossible for a single processor to handle such computation. So parallel computing is needed and it requires to separate the entire computing load to various computing nodes. Equation (2.2) and (2.3) shows an example of "systolic beamforming", by dividing the beamforming process into M parts. The entire data is divided equally and a portion assigned to each sub-beamformers, (i.e., computing node), in which the term $\sum_{i=1}^{C}\left(W_{jC+i}^{b} Y_{jC+i}\right)$ is calculated independently and C is the number of channel of each computing node can get.

$$\begin{aligned} Beam^{\Theta} = &\left[\sum_{i=1}^{C} W_i^{\Theta} Y_i\right] + \left[\sum_{i=1}^{C} W_{C+i}^{\Theta} Y_{C+i}\right] + \left[\sum_{i=1}^{C} W_{2C+i}^{\Theta} Y_{2C+i}\right] + \cdots \\ &+ \left[\sum_{i=1}^{C} W_{(M-1)C+i}^{\Theta} Y_{(M-1)C+i}\right] \end{aligned} \tag{2.2}$$

$$Beam^{\Theta} = \sum_{j=0}^{M-1} \left[\sum_{i=1}^{C} W_{jC+i}^{\Theta} Y_{jC+i} \right] \tag{2.3}$$

A PAR system may operate in an environment which is not stable and contain unwanted interference, so an adaptive beamforming weight is used to generate high gain in the beam steering direction and reject or minimize energy from other directions by adaptively adjusting the steering vector according to the interference environment. There are numerous methods for calculating the beamformer weights adaptively. The most standard solution is Wiener filter [32, 33], showing in Equation (2.4), in which W is a beamforming weight matrix; V is a matrix of column steering vectors; R is the covariance matrix of the received signal A. To achieve good performance, [34] suggests that the number of samples needs to be 2 to 5 times of channel number, N_{ch}. For example, if the system has N_{ch} number of channels, then a sample matrix, A has dimension of $N_{ch} \times 5N_{ch}$. Moreover, to desensitize the adaptive weight computation to perturbations [35], covariance matrix R needs to be appended by an extra loading matrix, Q. Though many studies suggest that the loading matrix takes the form of the covariance matrix of steering vector due to of the simplicity and effectiveness [36], the most widely-used method is *diagonal loading matrix*. Equation (2.5) shows a method for appending the diagonal loading matrix to covariance matrix with a constant *Loading Level*, σ. The diagonal loading matrix can be accommodated into Equation (2.4) by augmenting the sample matrix with an identity matrix with square root of desired loading level, as showing in Equation (2.6), in which B is the sample matrix appended with the loading matrix. To efficiently solve the weight vector W, [9] suggests using the Winer filter to avoid the calculating of the covariance matrix in Equation (2.6) by decomposing matrix

B into a lower triangular matrix L and an orthogonal matrix Q. The Equation (2.7) shows the simplified version of Equation (2.6) after applying LQ decomposition to matrix B.

$$W = R^{-1}V, in\ which\ \mathrm{R} = \mathrm{AA^H} \tag{2.4}$$

$$\bar{R} = R + Q = AA^H + \sigma I \tag{2.5}$$

$$W_1 = \bar{R}^{-1}V = (AA^H + \sigma I)^{-1}V = \left(\left[A|\sqrt{\delta}I\right]\left[A|\sqrt{\delta}I\right]^H\right)^{-1}V = (BB^H)^{-1}V \tag{2.6}$$

$$W_1 = (L^H)^{-1}(L^{-1}V) \tag{2.7}$$

The Wiener filter method of calculating adaptive beamforming weight contains three parts: LQ decomposition and two matrices backsolve. Suppose in Equation (2.6), B and V are the matrices of dimensions $m \times n$ and $m \times 1$. The complexity expressions for the LQ decomposition of matrix B is $8mn^2 - 8n^3/3$. As the dimension of L is the same as B, so two backsolves cost $4n^2$and $4m^2$ flops, respectively. Thus, in total, the complexity of weight calculation is

$$8mn^2 - 8n^3/3 + 4n^2 + 4m^2 \tag{2.8}$$

$$\begin{aligned} W_2 = (BB^H)^{-1}V &= \left(\left(A + \sqrt{\delta}I\right)\left(A + \sqrt{\delta}I\right)^H\right)^{-1}V \\ &= \left(AA^H + \sigma I + \sqrt{\delta}\hat{A}^H + \sqrt{\delta}\hat{A}\right)^{-1}V \end{aligned} \tag{2.9}$$

As mentioned above, the sample matrix A has the dimension of $N_{ch} \times 5N_{ch}$. In Equation (2.6), the symbol "|" represents that the matrix on the right side of the bar is appended to the end of the matrix on the left side. After diagonal loading matrix Q inserted into matrix A, the matrix B has the dimension of $N_{ch} \times 6N_{ch}$, which increases the complexity of computing the autocovariance matrix A. To reduce this additional computation load after diagonal loading, a new method is introduced here, named *direct appending loading level method*, by augmenting the loading matrix directly to the sample

matrix A, as showing in Equation (2.9). $\hat{A}$ is square matrix of the product from $A \times I$. By using this new method, the dimension of B is $N_{ch} \times 5N_{ch}$, maintaining the original size of sample matrix A. However, when we perform the direct appending method, two noise matrices $\sqrt{\delta}\hat{A}^H$ and $\sqrt{\delta}\hat{A}$ are brought into the covariance result, which may degrade the performance of beamforming. So, to investigate how much those two noise matrices may affect the performance of beamforming, we compared the SNR between original weight calculating method and our new method, as shown in Figure 2.4 and Figure 2.5, respectively. The signal power P_s, and the interference pulse plus noise power P_{i+n}, are given by

$$P_s = W^H R_s W \tag{2.10}$$

$$P_{i+n} = W^H R_{i+n} W \tag{2.11}$$

So, the SNR is equal to

$$\frac{P_s}{P_{i+n}} = \frac{W^H R_s W}{W^H R_{i+n} W} \tag{2.12}$$

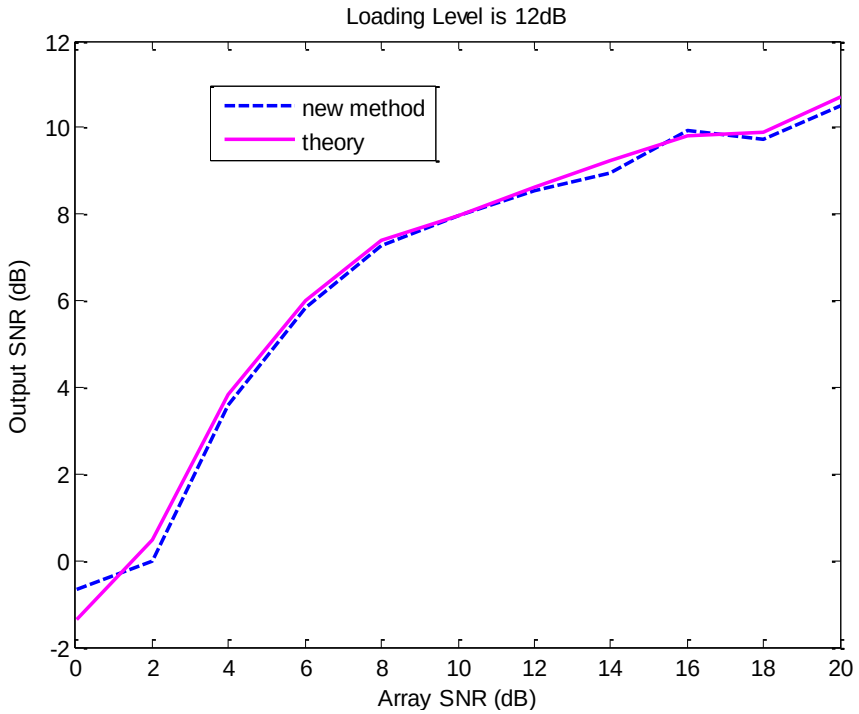


Figure 2.4: Output SNR of the new beamforming method versus SNR of traditional beamforming method

In the first simulation, we compared the performance of adaptive beamforming versus the array SNR. The loading level, LL is defined as $10\log(LL/\delta^2)$, in which δ is the standard deviation of zero mean white noise. From Figure 2.4, we observe that the performance of new method is closed to the theory one, which means the new method bring little impact to the SNR. Next, the influence of the loading level is shown in Figure 2.5. The maximum difference between two methods is 0.4 dB, which also indicates that the new method has similar performance as the theory's. Thus, the direct appending loading level method can be a good way to reduce the computing resource. Note that to implement the weight vector calculating for the adaptive beamforming, an extra processing node are required to receive the all the channel data, and then the weight can be distributed to each processing node.

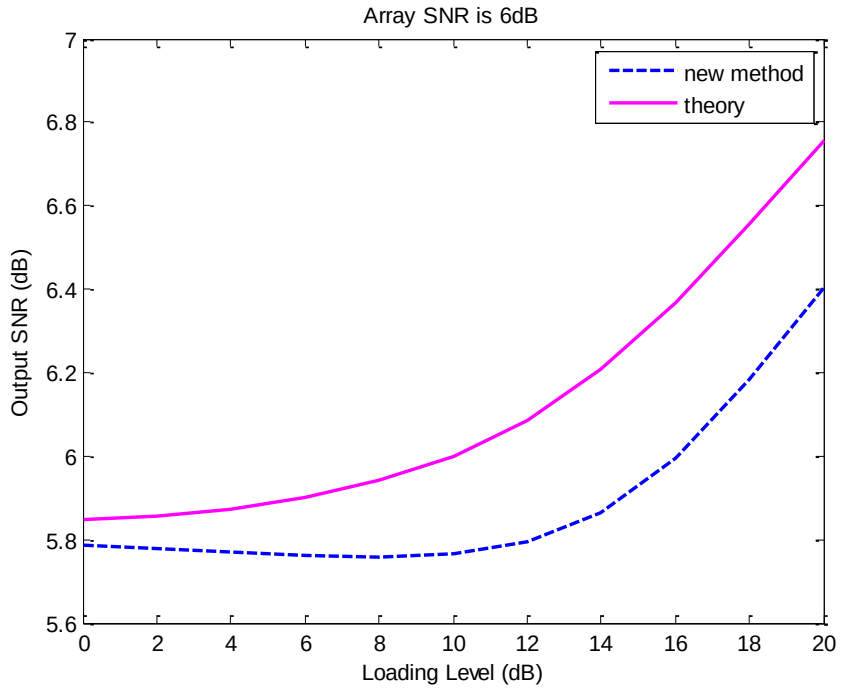


Figure 2.5: Output SNR versus loading level

2.3. Pulse Compression

After beamforming, pulse compression is for improving the signal-to-noise ratio and the range resolution. Pulse compression can be implemented by performing correlation of the return signal, $s[n]$, and a replica of the transmitted waveform, $x[n]$, which is equivalent to *matched filter* operation. By filtering the return signal, the energy of returned waveform would be aggregated into range gates and concentrates on the target ranges. Matched filter implementation converts the signal to the frequency domain, performs point-wise multiplies with transmitted waveform, and converts the result back to time domain [9], as shown in Equations (2.13)-(2.16). The length of FFT is chosen to be the first power of 2 greater than $N + L - 1$. For example, if $N = 2250$ and $L = 22$, which makes $N + L - 1 = 2271$, so the length of FFT should be 4098. In this situation, it requires to zero-pad $x[k]$ and $s[k]$ to the length of 4098, before converting into frequency

domain. Zero-padding increases computing complexity of the FFT in Equation (2.13) and (2.14), and makes Equation (2.15) to consume more computing resources. Values of N and L need to be selected to avoid unnecessary computation.

$$S[k] \overset{FFT}{\longleftarrow} s[n] \quad 0 \leq n \leq N \tag{2.13}$$

$$X[k] \overset{FFT}{\longleftarrow} x[n] \quad 0 \leq n \leq L \tag{2.14}$$

$$Y(k) = S[k]X[k] \; 0 \leq k \leq (N + L - 1) \tag{2.15}$$

$$y[n] \overset{IFFT}{\longleftarrow} Y[k] \quad 0 \leq n \leq (N+L-1) \tag{2.16}$$

From the above equations, the overall computing complexity of pulse compression depends on FFT, IFFT and point-wise vector multiplication. For radix-2 FFT, there are $log_2(N)$ butterfly computation stages. Each stage consists of $N/2$ butterflies, and each butterfly operation requires one complex multiplication, one complex addition, and one complex subtraction. Hence, the complexity of computing radix-2 FFT is:

$$C_{FFT} = (6 + 2 + 2) \times \frac{N}{2} \times \log_2 N = 5N \log_2 N \, flops.$$

As the IFFT has the same complexity of FFT, the throughput of the whole pulse compression in frequency domain is

$$\frac{2 \times C_{FFT} + NC_{mult}}{T} = \frac{10N \log_2 N + 6N}{T} FLOPS,$$

where N is the number of range gates after zero-padding, and C_{mult} is the complexity of point-wise complex multiply.

2.4. Doppler Processing and Data Corner Turn

The first objective of the Doppler processing is to mitigate the impacts of stationary or slow-moving clutter. The second objective is to measure the radial velocity of the targets by calculating the Doppler shift [37], from the Flourier transformation of data cube along the CPI dimension for each range bin. The throughput of an FFT-based basic Doppler filter is $C_{FFT}/T = 5N_p \log_2 N_p/T$ FLOPS, where N_p is the number of pulses in one CPI. The Doppler filtering performance is shown in Table 2.2.

Table 2.2: Doppler filtering performance measured in GFLOPS per core

Range Gates	**Pulses**				
	8	**16**	**32**	**64**	**128**
1024	0.7293	1.6036	2.6852	3.8543	4.2866
2048	0.7294	1.6000	2.6841	3.8543	4.2867
4096	0.7294	1.5999	2.6842	3.8544	4.2867
8192	0.7295	1.6000	2.6842	3.8544	4.2732

Compared with previous beamforming and pulse compression processing, Doppler processing requires less computing power. However, additional data transmission time is required before Doppler processing. As the output of the pulse compression is arranged along the range gate dimension, the output needs to undergo a corner turn before being handled by the Doppler filtering processors [38]. This two-dimensional corner turn operation is equivalent to a matrix transpose in the memory space. Using EDMA3 [39] on TI generic C66xx DSP, the data can be reorganized into the desired format without interfering the real-time computations in DSP core. So, the performance of Doppler processing can be performed without interference from data corner turn. The use of EDMA3 would be further discussed in further.

2.5. Weather and Air-surveillance Data Products

Section 2.2, 2.3, and 2.4 have illustrated the basic PAR signal processing algorithms, which are the building blocks of radar signal processing chain and foundations of other advanced signal processing algorithms. For example, in weather radar application, the mean Doppler velocity and spectrum width estimation takes outputs from beamforming and pulse compression. In the air-surveillance application, target detection and tracking processing also depend on the results of beamforming, pulse compression, and Doppler filtering. This section will illustrate the high-level, or *backend*, PAR data processing algorithms and discuss their complexities.

2.5.1. Mean Velocity Estimation

Mean Doppler velocity is the averaged velocity of the radar resolution volume. There are two methods to calculate the Doppler frequency shift: spectral processing and Pulse Pair Processor (PPP) [40]. In the spectral processing method, the first step is to calculate the periodogram [41] of signal along the CPI domain by using FFT as

$$S = \frac{|Z(f)|^2 T}{M}, in\ which\ Z(f) \overset{fft}{\leftarrow} s(n) \tag{2.17}$$

where M is the number of pulses, T is the Pulse Repetition Time (PRT), $s(n)$ is the samples along the CPI domain. Then the mean velocity is calculated by using Equation (2.18) [42], in which λ is the wavelength of transmitting wave, P is the total power in the periodogram, k_m is the index of the strongest Fourier coefficient, and i is the index of pulse. Most of computation load in mean velocity estimation is from Equation (2.17) and the summation part in Equation (2.18). As the $Z(f)$ is the result from Doppler filtering mentioned in Section 2.4, those results can be utilized twice in the mean velocity

estimation to save the computing resources. Therefore, the complexity of Equations (2.17)and (2.18) are $\mathbf{12}\boldsymbol{M} + \mathbf{7}\boldsymbol{M} = \mathbf{19}\boldsymbol{M}$ in combination.

$$v = \frac{\lambda}{2M}\left\{\frac{k_m}{T} + \frac{1}{PT}\sum_{k_m - M/2}^{k_m + M/2}(i - k_m)S[mod_M(k_m)]\right\} \tag{2.18}$$

PPP is another method to estimate the mean Doppler velocity by comparing the phase differences among various samples. A general method to calculate the phase difference is using the covariance approach [40]. The first step is to estimate the autocorrelation of the signal along the CPI domain with one sample lag, $n = 1$, as

$$R_n = \frac{1}{M}\sum_{i=0}^{M-1} s_i^* s_{i+n} \tag{2.19}$$

So, the mean velocity can be estimated as

$$v = \frac{\lambda}{4\pi T} arg(R_1) \tag{2.20}$$

where the argument of *R* calculates the phase of *R* in radians. Based on Equation (2.19) and (2.20), we can calculate the complexity of PPP is $6M + 2M = 8M$. Compared with the spectral processing, PPP method has the advantages of requiring less computing resources and having smaller velocity variance when there is no unique solution for SNR and spectrum widths [43].

2.5.2. Spectrum Width Estimation

The mean velocity estimation mentioned in the previous section represents the average speed of hydrometeors in one range gate. When there are turbulence or chaotic flow, hydrometeors within one resolution volume have vastly different radial velocities [44]. In this case, the mean velocity cannot represent the entire range of velocities within one range gate and may overlook the fast-changing weather phenomenon. The spectrum

width can depict the standard deviation of the velocity and represent random particle movements. Methods used to extract the spectrum width are usually based on autocovariance processing and spectral estimation.

The autocovariance processing method utilizes the autocorrelation of the signal at different lags to estimate the spectrum width. If the weather signal spectra closely follow a Gaussian shape, the estimated spectrum width, ${\sigma_v}^2$, is

$$ {\sigma_v}^2 = \frac{\lambda^2}{24(\pi T)^2} \ln \left| \frac{R_1}{R_2} \right| \tag{2.21} $$

where R_1 and R_2 are the autocorrelation based on Equation (2.19) [45]. The computing load of Equation (2.19) mainly comes from autocorrelation, so the complexity can be approximated as $\boldsymbol{6N_p + 6N_p = 12N_p}$.

Corresponding to the autocovariance method, spectrum width estimation by using spectral processing is given by

$$ {\sigma_v}^2 = \frac{\lambda^2}{4PT^2} \sum_{k_m - M/2}^{k_m + M/2} \left(\frac{i}{M} + \frac{2vT}{\lambda} \right)^2 S[mod_M(k_m)] \tag{2.22} $$

The complexity of Equation (2.22) is the same as Equation (2.18), which is $\boldsymbol{19N_p}$. Thus, the spectral processing requires more computing resources than autocorrelation method. Moreover, Equation (2.22) is a biased estimation due to the window effect in the FFT. In general, this bias is difficult to compute [43], thus in general, autocovariance method is superior than the spectral processing.

2.5.3. Target Tracking

For MPAR, target tracking (such as air-traffic tracking) is a crucial function. Once a radar receiver detects the presence of the targets and converts the detections into validated

measurements, the radar tracker initiates and estimates the target's future state, while integrating the new measurements into an existing track. Bayesian tracking or Bayesian recursively tracking is a method to treat the tracking problem from the perspective of Bayesian inference. It assumes that a likelihood function links the events observed in the current state to the future unknown. As such, if we specify a prior distribution of some targets, we can calculate the posterior distributions or future states with the help of likelihood function [23]. For example, if a surveillance radar records the previous speeds and locations of an air-vehicle, based on the currently measured speed and location, a likelihood function can be established, which is then used to predict the vehicle's future positions and velocities through different models.

The goal of multiple target tracking (MTT) is to estimate the states of multiple targets simultaneously [46]. Compared with single target tracking, MTT needs to determine which target generates each sensor response or whether the response is a false alarm. For most cases, we may not know the exact number of the objectives, which makes the MTT further complicated. Situations that the tracks of multiple targets are overlapped or intersected can lead to ambiguity of the data association process. Also, as the kinematic model of each target can vary, the transition functions that we used in a Kalman filter may not be suitable for all the scenarios.

For multiple target tracking, joint probabilistic data association (JPDA) and multiple hypothesis tracking (MHT) are the two classical Bayesian tracking methods. In all cases, both algorithms (JPDA and MHT) can provide reliable target tracking performance [46, 47]. MHT forms data association hypotheses by assigning probability 1 or 0 to a target, which is a *hard* association. JPDA relaxes this assumption by allowing

for the partial association. In the low SNR environments, JPDA and MHT are both capable of handling a high volume of clutter. However, MHT has a major disadvantage of requiring high computational complexity because the number of hypotheses grows exponentially over tracking time [48]. Although various methods have been developed to control the growth of hypotheses tree [49], JPDA is still more efficient and easier to be implemented. Also, when the detection probability is reduced for the weak target scenarios, MHT is more vulnerable than JPDA [50], since MHT is a single-scan algorithm compared to JPDA and depends heavily on the past scans. Hence, JPDA would be a better choice if there are no other specific requirements and use JPDA as an example to illustrate the computing complexity of tracking algorithm.

To track the targets, at first, we may build a model to represent the tracking system. Let $\boldsymbol{S}$ be the state space of a target dynamics, in which it contains various target information that can be utilized to locate and track the targets, such as the position, velocity, and acceleration of the targets. Thus, the targets in the space $\boldsymbol{S}$ can be represented as a vector containing kinematic parameters [8]. For a typical parametric approach, the classic Bayesian approach uses the dynamical motion and measurement equations shown in Equation (2.23) and (2.24) for the target tracking models:

$$y_k = A_{k-1}x_{k-1} + Q_{k-1} \tag{2.23}$$

$$y_k = H_k x_k + r_k \tag{2.24}$$

where x_k is the state vector for the target on the time step k, A_{k-1} defines the transition matrix of the dynamic model, y_k is the measurement vector on the time step k, Q_{k-1} is the process Gaussian random noise covariance matrix for the time step $k-1$, denoted as $Q_{k-1} \sim N(0, R_{k-1})$. H_k is the measurement matrix that converts the system state to the

measurement, and $r_k \sim N(0, R_k)$ is the sensor measurement noise vector [37]. Based on this model, we can represent the target moving in a two-dimensional space in Cartesian coordinate system as

$$\boldsymbol{x}_k = [x_k \; y_k \; \dot{x}_k \; \dot{y}_k]^T \tag{2.25}$$

in which $\dot{x}_k$ and $\dot{y}_k$ are the velocities of the target along the x and y coordinate observed by a Radar. By giving Equation (2.25), we can represent transition matrix A_{k-1} as Equation (2.26), in which $\Delta t = t_k - t_{k-1}$.

$$\mathrm{A_k} = \begin{bmatrix} 1 & 0 & \Delta\mathrm{t} & 0 \\ 0 & 1 & 0 & \Delta\mathrm{t} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.26}$$

For the nearly constant velocity motion model, the process noise covariance matrix is given by

$$Q_k = \delta^2 \begin{bmatrix} \Delta t^3/3 & \Delta t^2/2 & 0 & 0 \\ \Delta t^2/2 & \Delta t & 0 & 0 \\ 0 & 0 & \Delta t^3/3 & \Delta t^2/2 \\ 0 & 0 & \Delta t^2/2 & \Delta t \end{bmatrix} \tag{2.27}$$

The δ^2 is the design parameter for the system model error. Typically, this parameter is set to be greater than one half of the maximum acceleration of the target and less than the maximum acceleration [51]. The measurement matrix H_k is used to calculate the position of the target given the system state. In this case, the measurement matrix can be expressed as Equation (2.28).

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.28}$$

The system defined in (2.23) and (2.24) satisfies the Markov property, which means that the future state of this system is based solely on its present state. We can express this property in general as:

$$p(x_k|x_{1:k-1}, y_{1:k-1}) = p(x_k|x_{k-1}) \tag{2.29}$$

This also implies the fact that the past does not depend on the future state by given the present, which is the same concept as in Equation (2.29), in which $x_{k:T}$ represents the system states from current time step k up to future time step T.

$$p(x_k|x_{k:T}, y_{k:T}) = p(x_{k-1}|x_k) \tag{2.30}$$

For the measurement, which is the same as system state, the current measurement y_k is independent from the past measurement and system state. This property can be expressed in Equation (2.31).

$$p(y_k|x_{1:k}, y_{1:k-1}) = p(y_k|x_k) \tag{2.31}$$

The JPDA is an extended version of *Probabilistic Data Association Filter (PDAF).* PDAF is to set up a validation region at each sampling time. Among that possible validated measurement, the position of a target can be determined by calculating the data association function of each measurement [52]. Similar to Kalman filter, PDAF makes an estimation based on the past measurements and states. If the state and measurement equations are assumed to be linear, the update and predicting algorithm of PDAF can be based on Kalman filter. When the state or measurement equations are nonlinear, then PDAF can be based on Extended Kalman Filter (EKF) [50] or Particle Filters [8]. The algorithms of PDAF and JPDA discussed in this section are based on Kalman filter. To perform PDAF, a basic assumption that the posterior probability function for the system state is summarized approximately by a normally distributed Gaussian

$$p[x(k)|z^{k-1}] = N[x(k); \hat{x}(k|k-1), P(k|k-1)] \tag{2.32}$$

In contrast with previously discussed front-end processing algorithms, the throughput of the tracking algorithm occurs on a per-target or per-track basis. Since the number of

targets, tracks, and false alarms in the detection area is unknown, the computation requirements are nondeterministic. Moreover, compared with front-end processing tasks, which are all streamlined mathematic calculations, the back-end tracking algorithm involves more logical operations--stored, accessed, and updated target position overtime periods. So, it is usually difficult to estimate the computing resources needed for tracking algorithm without prior knowledge of the operational environment. To give a basic idea of the computational complexity, we calculate the throughput of the JPDA based on the sample parameters listed in Table 2.3.

Table 2.3: Tracking simulation parameters

Parameters	Description
T	Number of currently confirmed targets
M	Number of measurements including potential targets and false alarms
E	Number of potential tracks

Prediction

The first step of JPDA is to conduct the Kalman prediction from $k-1$ step to k, which is the same procedure as the single target Kalman filter, namely

$$\hat{x}(k|k-1) = A(k-1)\hat{x}(k-1|k-1) \tag{2.33}$$

$$\hat{z}(k|k-1) = H(k)\hat{x}(k|k-1) \tag{2.34}$$

$$P(k|k-1) = A(k-1)P(k-1|k-1)A(k-1)' + Q(k-1) \tag{2.35}$$

where $P(k-1|k-1)$ is the covariance matrix of the past system state. Based on it, the new innovation covariance matrix can be computed as

$$S(k) = H(k)P(k|k-1)H(k)' + R(k) \tag{2.36}$$

The computation complexities of Equation (2.33)-(2.36) are $32T$, $16T$, $272T$, and $132T$ separately. In total, there are $\mathbf{452T}$ **flops** computation in each scan.

Measurement Validation

In this step, an elliptical shape validation region is defined as Equation (2.37). The volume of this region is limited by the gate threshold parameter γ. Measurements that lie inside the gate γ are considered valid; those are outside are discard.

$$V(k,\gamma) = \{z: [z - \hat{z}(k|k-1)]' S(k)^{-1} [z - \hat{z}(k|k-1)] \leq \gamma\} \tag{2.37}$$

In this step, there is $22MT$ flops computation in each scan.

Data Association

The data association of the measurement $\theta_i(k)$ at time k of the target $z_i(k)$ is

$$P[\boldsymbol{\theta}(k)|z^k] = \frac{1}{c} P[Z(k)|\boldsymbol{\theta}(k), Z^{k-1}] P\{\boldsymbol{\theta}(k)\} \tag{2.38}$$

where c is the normalization constant. If we consider the entire measurements and suppose that all the measurements are lied in the validation area, the PDF on the left-hand side of Equation (2.38) can be written as

$$p[Z(k)|\boldsymbol{\theta}(k), Z^{k-1}] = \prod_{j=1}^{M} p[z_j(k)|\theta_{jt_j}(k), Z^{k-1}] \tag{2.39}$$

where θ_{jt_j} is the measurement j originated from target t that $j = 1, \cdots, M$ and $t = 0, \cdots, T$. The conditional PDF of the above equation is assumed to be a Gaussian distribution as:

$$p\left[z_j(k)\middle|\theta_{jt_j}(k), Z^{k-1}\right] = N[z_j(k);\ \hat{z}_{t_j}(k|k-1);\ S_{t_j}(k)] \tag{2.40}$$

The prior probability of an event $\theta(k)$, the second part of right side of Equation (2.38), can be obtained as

$$P[\boldsymbol{\theta}(k)] = \mu_F(\phi) \prod_{t=1}^{T} (P_D)^{\delta_t} (1 - P_D)^{1-\delta_t}, \tag{2.41}$$

where δ_t is the number of target that has been detected at time k, and $\mu_F(\phi)$ is the clutter density. Finally, after we combine (2.41) and (2.40) into (2.38), we have

$$P[\boldsymbol{\theta}(k)|z^k] = \mu_F(\phi) \prod_{j=1}^{M} N_{t_j}[z_j(k)] \prod_{t=1}^{T} (P_D)^{\delta_t} (1 - P_D)^{1-\delta_t} \tag{2.42}$$

The computation complexity of Equation (2.42) is $73ME$. The marginal association probabilities are obtained from the joint probabilities by summing over all the joint events, which has the complexity of MT. So, in total there are $73ME + MT$ in the data association step. The marginal association probabilities are obtained from the joint probabilities by summing over all the joint events.

$$\beta_{jt} \triangleq P\left[\theta_{jt} \middle| Z^k\right] = \sum_{\boldsymbol{\theta}} P[\boldsymbol{\theta}(k)|z^k] \hat{\omega}_{jt}(\boldsymbol{\theta}) \tag{2.43}$$

State Estimation

The state update equation of JPDA is the same as Kalman filter as

$$K(k) = P(k|k-1)H(k)^T S(k)^{-1} \tag{2.44}$$

$$x(k) = x(k|k-1) + K(k)v(k) \tag{2.45}$$

$$v(k) = \sum_{j=1}^{m_k} \beta_{jt}(k) v_i(k) \tag{2.46}$$

The computation complexity of Equation (2.44) and (2.45) are $109T$.

The error covariance associated with the updated state estimate is

$$P(k|k) = \beta_0(k)P(k|k-1) + [1 - \beta_0(k)]P^c(k|k) + \tilde{P}(k) \tag{2.47}$$

where

$$\tilde{P}(k) = K(k)\left[\sum_{j=1}^{M} \beta_{jt}(k) v_i(k) v'_i(k) - v(k)v'(k)\right] K'(k) \tag{2.48}$$

and

$$P^c(k|k) = [I - K(k)H(k)]P(k|k-1) \tag{2.49}$$

The computation complexity of Equation (2.48) and (2.47) is $(20M + 108)T$ and $88T$. So, there are $(20M + 196)T$ flops in the state estimation step. In total, the JPDA requires $452T + 22MT + 73ME + MT + 20MT + 108T = \mathbf{73ME} + (\mathbf{560} + \mathbf{43M})\mathbf{T}$ flops for one scan. To give a realistic example for a terminal aircraft surveillance tracking, we chose the parameters that $M = 30$, $T = 20$, and $E = 25$, and the radar would take 4.5 seconds for each scan. So, the throughput of JPDA tracking is around 20 KFLOPS. This workload is much lower than the "front-end" signal processing. Although the workload varies linearly with the number of tracks and targets, even a tenfold increase would only be a small fraction of the "front-end" processing computing complexity. If parallelism is required, the tracking algorithm can be easily implemented in a multithreading operating system, and each thread contains several hypothesis tracks to be estimated in each processor.

2.6. Advanced Algorithms

2.6.1. Model-Based Algorithms and System Optimizations

The primary signal-processing chain described in Section 2.1 has certain assumptions about the PAR system, these assumptions are not usually valid in realistic radars. For example, the estimation of computational loads assumes zero latency for data transportation and memory access. The radiation patterns of individual antenna elements

are usually considered to be isotropic, and there is no channel-to-channel and pulse to pulse errors for the beamforming algorithm, and there are no signal distortions in RF channels. In reality, however, these assumptions are not realistic, and the overall signal processing performance can be severely affected by these factors. In this section, we explore several advanced processing algorithms and real-time implementations targeting these issues. Many existing works have been reported on these topics [53, 54] while our focus is the channel data rate control, interference mitigation and optimized calibrations based on particular types of signal models (sparsity, covariance and nonlinearity distortions).

The Power Amplifier (PA) models can be divided into two class: physical model and empirical models [55]. The physical model handles the true RF modulated signal and conceives to process real excitations by using nonlinear models of the active devices to form an equivalent-circuit description, which requires deep knowledge and insight of the circuit layout. This method provides a high level of accuracy result and limited by the quality of the modeling of each component in PA. However, this benefit comes at the cost of high computational time and the need for a detailed description of each component by measurement of inspection.

When the design of PA is unknown or driving a PA equivalent circuit is not available, PA behavior model is preferred, which is a black box simulation based on the input and output behavioral observations. Thus it is used to simulate the PA behavior by employing low-pass equivalent PA models and thus processes only the complex-valued envelope signals at the PA input and output [56]. The accuracy of this behavior model highly depends on the adopted model structure and the excitation parameters. Based on

the memory effect of different PAs, the system-level model can be divided into two categories: memoryless and memory models. Memory effects are non-noise circuit characteristics, which is caused by the thermal constants of the active devices or components in the biasing network that has frequency-dependent behaviors. As the name suggests, the memoryless model represents the input of PA has instantaneously effect to the output. So, the observed AM-AM and AM-PM conversion constitute the PA's memoryless behavior. Commonly, two memoryless models are used: a polynomial function with complex coefficients, like cubic polynomial model as showing in Equation (2.50) and (2.51), where u is the normalized input voltage,

$$F_{AM-AM} = u - \frac{u^3}{3} \tag{2.50}$$

$$F_{AM-PM} = upper\ power\ limit - lower\ power\ limit \tag{2.51}$$

and the Saleh model [57] in Equation (2.52) and (2.53)

$$r_y\big(r_x(t)\big) = \frac{\alpha_r r_x(t)^2}{1 + \beta_r [r_x(t)]^2} \tag{2.52}$$

$$\phi_y\big(r_x(t)\big) = \frac{\alpha_\phi r_x(t)^2}{1 + \beta_\phi [r_x(t)]^2} \tag{2.53}$$

where $\mathrm{r_x}(t)$ represents the input envelope, the coefficients $\alpha_r, \beta_r, \alpha_\phi,$ and β_ϕ are fitting parameters for the measured PA's AM-AM and AM-PM characteristics, which can be extracted using a least squares approximation to minimize the relative error between the envelope measurements of the target PA and the values predicted by model.

The above-mentioned low-pass equivalent AM-AM and AM-PM memoryless models are frequency independent, which have reasonable accuracy when the narrowband signal drives the amplifiers. However, when the bandwidth of the input is

comparable to the inherent bandwidth of the amplifier, the response of each frequency in the PA is frequency-dependent. So, in the wide-band application, it is necessary to take both the nonlinear and memory effects into consideration when modeling a PA. A straightforward method to simulate the memory effects is sampling the bandwidth at all possible frequency points, so the variant in PA response for different frequencies can be found [58]. In some situations, the memory may have nonlinear memory effect, which can be seen as frequency-dependent nonlinear impulse responses [59, 60]. The idea of modeling nonlinear memory effect is to postulate that the gain and phase characteristics of PA do not merely depend on the instant input $\mathrm{r_x}(t)$ but also on a parameter $\mathrm{z}(t)$, where $z(t)$ is a function of historical input signal and physical characteristic causing memory effects within the amplifier [61]. Then the nonlinear memory effect can be modeled by self-heating of the active device or by a varying power supply as Equation (2.54).

$$y(t) = f[r_x(t), z(t)]r_x(t)e^{j\theta(t)} \tag{2.54}$$

2.6.2. Compressive Sensing for Channel Data Rate Reduction

A traditional coherent radar receive channel generates in-phase and quadrature data signals in either analog or digital forms. These signals are transported to the specialized signal processor units for pulse compression, detection, and tracking. With the extensive use of advanced waveforms, the bandwidth of the transmitting pulses can be quite large. Accordingly, based on Nyquist–Shannon sampling theorem, increasing ADC sampling speed may introduce a massive amount of data transactions. As an example, for a single dual-polarized channel, if the signal bandwidth is 20 MHz and ADC's resolution is 12 bits, the transmitting rate should be at least 960 Mbps per channel. For an envisioned

MPAR system with 200 dual-pol channels, the data rate at inputs of a beamformer can be higher than 192 Gbps. Even with advanced data link technologies today, this is still a tremendous challenge.

According to the compressive sampling concept, when the signal matrix is sparse, we can sample the radar signal incoherently at a much slower rate than the Nyquist sampling rate [62, 63], which may translate into saving of communication bandwidth, reduction of signal processors, and eventually lower costs. When we introduce the CS concept into array signal sampling, there are two specific issues we may pay attention to: (1) Robustness of signal recovery from noisy data, especially for the received signals before pulse compression. Indeed, CS processing can tolerate a proper level of noise. However, when noise power is comparable to the power of the signal, it may lead to errors or distortions in signal recovery. (2) The computing time and resources requirement of the signal recovery. The computational resources required by CS processing, and the additional latency that adds in the receiver chain, should not offset the benefits it brought in for data transportation bandwidth reduction.

Supposing a return signal reflecting from a target and sampled by the front-end with the Nyquist rate into a vector f with the length of N, there exists a sensing basis Φ, on which the projection of x in the front-end is a vector with the length of l. Since the signal is sampled lower than Nyquist rate, the received data needs to be reconstructed and projected onto representation basis Ψ. If the signal on basis Ψ has s non-zero coefficients, it said this signal is the s-sparse. In the representation basis Ψ, it makes the possible that the front-end system use fewer samples to reconstruct the signal without much loss by discarding the zero coefficients. If the condition satisfied that $s < l \ll N$

and the basis Φ and Ψ are uncorrelated, the under-sampled signal can be reconstructed back by using CS. Note that, as s increases, it becomes harder to sense and reconstruct the original signal [64]. The coherence between the basis Φ and Ψ is measured by

$$\mu(\Phi,\Psi) = \sqrt{n} \cdot \max_{1\le k,j\le n} |\langle \phi_k \psi_j \rangle| \tag{2.55}$$

in which μ is the incoherence property, n is the number of elements in the original signal, and k and j are indices of the basis functions. In other words, the sensing and representation basis should be concerned as low coherence pairs. For example, we may choose spike basis $\varphi_k(t) = \delta(t-k)$ as sensing matrix, and Fourier basis $\psi_j(t) = \sqrt{n}e^{-i2\pi jt/n}$ as representation basis [63]. To analyze the coherence between the sensing basis and representation basis, the restricted isometry property (RIP) is introduced. RIP characterizes isometry constant δ_{2s} of a matrix such that

$$(1-\delta_{2s})\|x_1 - x_2\|_{l2}^2 \le \|\Theta(x_1 - x_2)\|_{l2}^2 \le (1+\delta_{2s})\|x_1 - x_2\|_{l2}^2 \tag{2.56}$$

in which Θ is the reconstruction matrix, which is the product of Φ and Ψ. If δ_{2s} is sufficiently less than one, this implies that the all pairwise distance between s-spare signals, for any vector x_1 and x_2, can be well preserved in the measurement space. That means measurement matrix contains the sufficient information in signal of interest, and the majority part of signal can be reconstructed from the measurements.

When the basis Φ and Ψ satisfy RIP, the Equation (2.57) gives an accurate reconstruction of the undersampled signal by using L1-norm minimization.

$$\min_x \|x\|_{l_1} \; s.t. \; y = \Phi\Psi x \tag{2.57}$$

Different applications may have various requirements or limitations to use CS. In the communication system, it requires the CS algorithm for speedy spectrum sensing; in medical imaging processing, like magnetic resonance imaging, with benefits for patients'

economics, the scan time reduction is the thing researchers pay more attention to. In radar application, the SNR may be too small that the signal can be immersed within the noise; hence, robust signal recovery from noisy data is a crucial point for radar sampling. To exam the performance of CS algorithm when SNR is low, Figure 2.6 shows a comparison between reconstruction data and original signal (noise-free), and error compared to the original signal with noise. We can see that the CS can suppress noise levels when SNR is low. This is because the signal (pulse) is sparse, and the noise is widely spread the entire spectrum. As a result, the reconstruction process would ignore those small variations produced by the noise. From Figure 2.6 we can also notice that when SNR is larger than 6 dB, the reconstruction data have the similar result as original data with noise, which means the compressive sampling can be used in the radar application even the SNR is low. Besides that, CS can still perform under low SNR conditions. This may be because this signal (pulse) is sparse, and the noise is widely spread the entire spectrum. As a result, the reconstruction process would ignore those small variation produced by the noise. This noise reduction phenomenon had been proved in [65, 66].

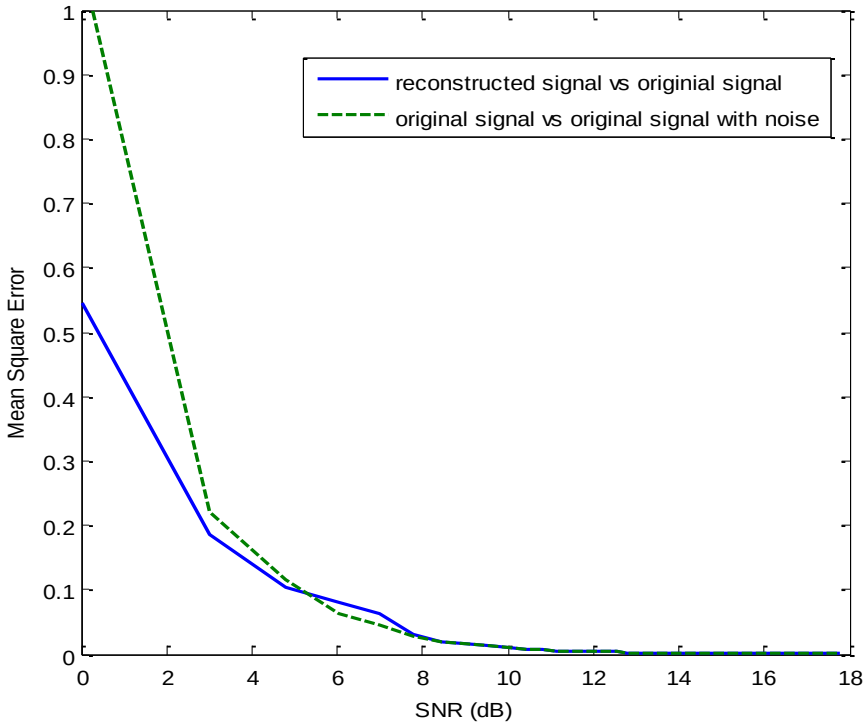


Figure 2.6: Reconstruction error vs SNR using CS algorithm

Another important aspect of CS implementation is the algorithm efficiency. There are so many reconstruction algorithms existing, such as Basis Pursuit [67], Matching Pursuit [68], and Message Passing [69]. Among those algorithms, the greedy iterative algorithm is easy to implement and has a high speed of signal recovery. It solves the reconstruction problem by finding an optimal result iteratively. Within the framework of greedy pursuing, we select the Orthogonal Matching Pursuits (OMP) [70] as our core compressive sensing algorithm. For a signal with length n and s sparsity, OMP can reliably recover this signal by using $O(s \log n)$ measurements. The complexity of OMP algorithm is $O(smn)$ is the number of measurements. Figure 2.7 shows the comparison between the OMP and Basis Pursuit, where $n = 600$, and $m = 4s$. It can be seen that OMP have better performance than the basis pursuit. However, when the signal is not sparse, the recovery becomes costly.

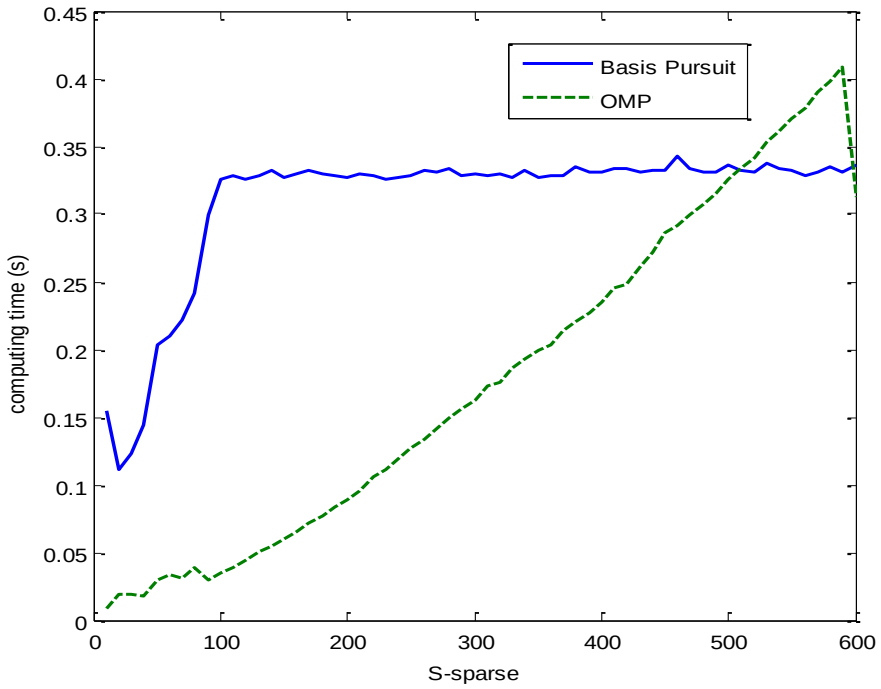


Figure 2.7: Computational time comparison of two CS algorithms on AMD Opteron 6128/MATLAB regarding to different degrees of signal sparsity

2.6.3. System Optimizations

RF hardware in radar receivers usually causes some of the undesired phase shifts and amplitude variations, and those noise and distortions contaminate measurements. For example, the transfer function of a power amplifier may not hold a constant gain for a broad range of input power levels, and different amplifiers may not share the same transfer function. So, this response inconstancy among channels would distort the shape of the antenna pattern and bring errors into the measurements. To alleviate the inconstancy response among channels, it is essential to calibrate the array by equalizing the phase and amplitude effects [71].

There are some existing methods to calibrate the PAR system channels such as near-field scanning probe [72], fixed peripheral probes [73], calibration lines [74], and

mutual coupling [75]. The first three calibration methods require a controlled environment by using specific calibration equipment, and procedures for performing those methods are complex and challenging to be conducted in the field. Furthermore, due to the reason of that the relative phase/amplitude shifts depend on frequency, temperature, and time, calibration should be repeated at various temperatures and frequencies. These variables increased the complexity of performing calibration once the radar is deployed. Compared with the first three methods, mutual coupling calibration can be done without extra equipment, but the calibration accuracy can be easily deteriorated when the environment contains near-field clutter. Moreover, for the reason of that setting the output power level too high would make transmitters saturate the receivers, the mutual coupling cannot perform full range power level calibration. As the PAR channels may have different gain values for different input power levels for a given frequency, the calibration result based on mutual coupling may not fully meet the actual operating requirements.

To make the PAR calibration reliable and feasible, we introduced a calibration procedure that allows the radar system to perform self-calibration in the field by using the Expectation-Maximization (EM) algorithm. Moreover, EM calibration does not require extra equipment or feedback line to do the calibration. A similar EM calibration method has been used in the [76]. Compared with mutual coupling calibration, EM method is more robust when the radar surrounding environment has clutter, and the calibration can be done during normal radar operations. EM algorithm is based on a probabilistic learning model by iteratively computing the maximum-likelihood estimates when the observations can be viewed as incomplete data [77]. Starting from an initial

assumption, each iteration involves two steps: expectation step (E-step) and a maximization step (M-step) [78]. In E-step, it finds a probability distribution over the unobserved variables given the known values for the current model; in M-step, it re-estimates the parameters of the current model to be those with maximum likelihood, under the assumption that the distribution found in E-step is correct. It can be shown that each iteration improves the likelihood and a local maximum can be reached [79].

By using EM algorithm calibration for PAR application, we derive the relationship between observed value and truth into a probabilistic model. Then, the algorithm would iteratively seek the maximum likelihood between the observed value and ground truth. The self-calibration procedure can be separated into two parts: amplitude calibration and phase calibration. For amplitude calibration, the received power level is directly calibrated by using EM algorithm. As mentioned in [80], the phase distortion follows the nonlinear model between input power level and output signal for a given frequency. This distortion can be estimated by comparing the phase differences between the signal leaked through the diplexer from the transceiver to the receiver and undistorted baseband signal [81]. After applying a range of power levels in the transmitter, the nonlinear phase distortion model can be applied. Note that the calibration procedures mentioned here are all based on the assumption that the RF system is working at a single frequency and the system is memoryless.

The first step of EM algorithm is assuming the initial probability distribution of observed power level, a, for each range gate by given true power level value, m, is a Gaussian distribution, defined as

$$P(a|m;ch) = \eta \times e^{\frac{-(a-m)^2}{2\sigma}} \tag{2.58}$$

in which η is the normalizer, ch is the channel indicator, σ is the variance. Note that each channel maintains its own distribution $P(a|m;ch)$, and is initialized as a distribution with the dimension of $a \times m$. An example of initialization is shown in Figure 2.8, in which the values of a and m are both in the range between 0 to 100. While we should note that the power level at each range gate is dependent with each other due to the sidelobe of the pulse compression results, if ignoring those correlations, we may assume that each range gate is conditionally independent of each other given the radar waveform and antenna beam pattern. Since what we are interested in is deriving the calibrated power levels in the RF channels rather than the truth RCS value of targets in each range gate, ignoring this spatial correlation is acceptable.

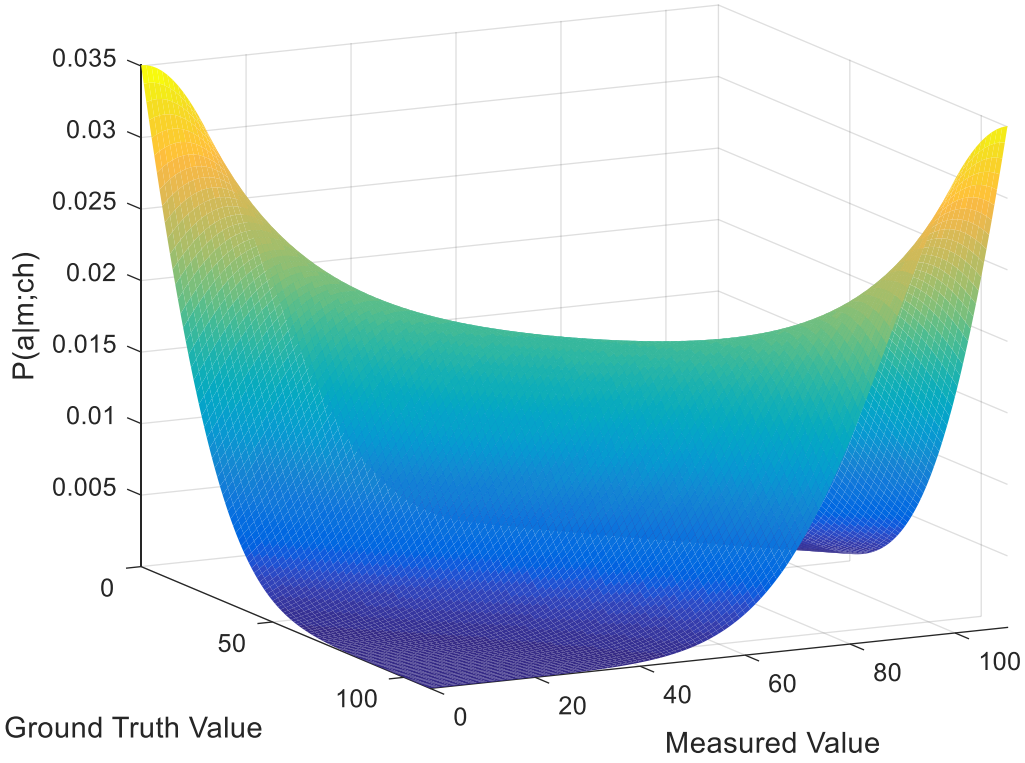


Figure 2.8: Initialization example for $P(a|m;ch)$

The goal of the amplitude calibration is to find the maximum likelihood of $P(a|m;ch)$ given truth power value m for each channel ch. Starting with the

initialization value in Equation (2.58), the algorithm alternates between updating $P(m = k; Rg)$ (E-step) and computing $P(a|m; ch)$ (M-step), which are described as follows.

Expectation step (E-step):

In E-Step, we calculate the probability of m for each range gate independently:

$$P(m = k; Rg) = \eta \prod_{ch=1}^{N} P(a|m = k; ch) \tag{2.59}$$

where η is the normalizer, k is the assumed ground truth power level, Rg is the index of range gate, a is the measured power level when the index of the range gate equals to Rg, and N is the total number of array channels.

Maximization step (M-step)

M step calculates the marginal probability of m by given a as

$$P(m|a; ch) = \sum_{Rg=1}^{M} P(m = k; Rg) \tag{2.60}$$

in which M is the total number of range gates. And then, we update Equation (2.58) by using Bayes' rule

$$P(a \,|\, m; ch) = \eta \times P(m|a; ch) \times P(a; ch) \tag{2.61}$$

in which $P(a; ch)$ is the probability of measured power amplitude from each channel. Equation (2.61) computes distribution of a for each channel given the distribution over ground truth power levels. After several iterations, the $P(a|m; ch)$ would converge and the maximum likelihood estimate between measured and the undistorted power level would be established.

To verify the performance of EM algorithm calibration, we simulated a simple three-channel receiver example with nonlinear distortion errors by using Matlab®. First, we generated a set of data, D, ranging between 0 and 100, representing the truth power levels. Note that we restrict the power levels to integers solely for the purpose of illustration, and decimal power levels can be used when more fidelity is required. Then, three different nonlinear transformations are applied to D, representing the output of three RF nonlinear systems--D_1, D_2, and, D_3. The three nonlinear transformation models used in this simulation are:

$$D_1 = 10\sqrt{m} \tag{2.62}$$

$$D_2 = \frac{m^2}{100} \tag{2.63}$$

$$D_3 = 14 \log_2 m \tag{2.64}$$

respectively, in which m are the undistorted power level of the reflected signal. We applied the EM algorithm for calibration to these three datasets. Figure 2.9 shows the calibration result, in which the solid lines represents the errors between the truth value and calibration result for each channel after using the EM algorithm. The dash lines represent the error between the nonlinearly-transformed datasets and the truth value. From the figure, we can see that the EM algorithm successfully predicts the trend of the three nonlinear transformations. However, this prediction contains some errors. The cause for these errors is that although the EM algorithm can increase the likelihood function between observed data and its truth value, this converges may be a local maximum of the observed data [78] depending on the initial value, which means the algorithm cannot guarantee a global optimum.

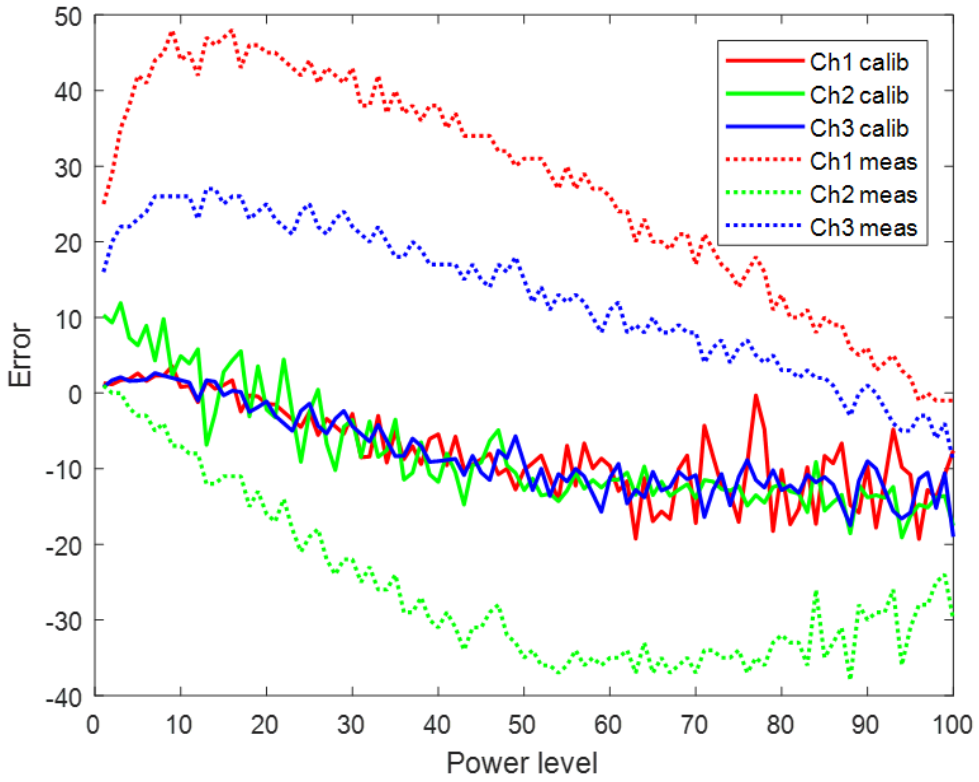


Figure 2.9: Simple example of three-channel receiver calibration results obtained by using EM self-calibration algorithm

One simple way to alleviate the problem is to set the initial value randomly and take the highest likelihood obtained as the global maximum [82]. Figure 2.10 shows the calibration result of data set D_1, as mentioned before, of using various initial conditions as an example, in which the solid curve represents what is the truth power level for each measured power level, and the dash curves is the calibration results. Among dash curves, there is an intersection point of each solid line. When the calibrated power level is on the left side of the intersection point, with the increasing value of σ, the calibration result has better matching with the truth value, however, the results become worse when σ grows too large. On the other hand, on the right side of intersection point, when the value of σ becomes larger, the curve has good matching between ground truth and measured value. So, we need an approach to select the best estimation based on different initial conditions

σ. Practically, since the truth value is unknown, we cannot directly tell which initial condition has better predictions than others. However, the measured power level differences among adjacent curves for a given calibrated power level indicates the prediction status (fitted or over-fitted compared with true values). For example, in Figure 2.10, when the calibrated power level is in the range of [1, 20], starting from the blue line, with the value of σ increased, the prediction status of each line changing from under-fitted to over-fitted compared with dash line. When σ is no larger than 529, the measured power level differences between each of three curves on the top of figure are around $\sqrt{\sigma} = 10$. In contrast, the measured power level differences between purple and yellow curves is smaller than 10. So, based on this observation, we can use the variation of the

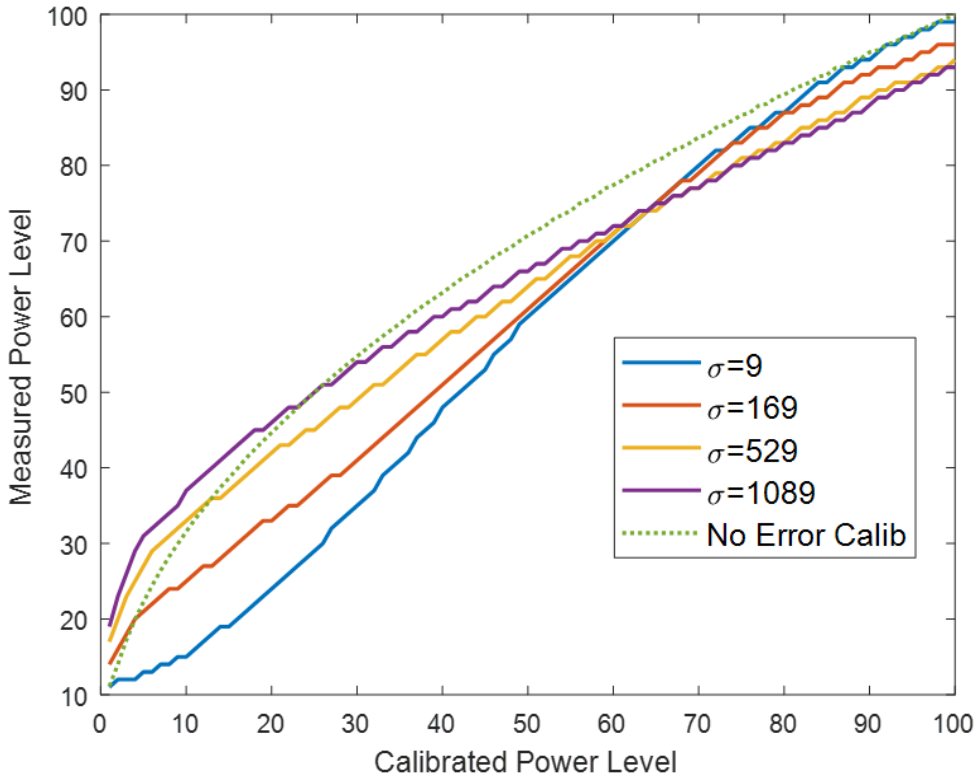


Figure 2.10: Comparison of calibration results based on various initial conditions. (The dash line is the measurement value vs truth level. The solid line is the measured level vs calibration result from three different values of σ.)

differences of the measured power level for each σ as an indicator of the prediction status (fitted or over-fitted). Moreover, there is an intersection point of four calibration curves. After this point, the increasing rate for each curve is inversed, which can help us to determine which curve should be picked. The optimum curve line finding procedure is described as in Figure 2.11, in which opt is the numbers of starting conditions, $RecPwr$ indicates the power level in each channel, $Curv(i,k)$ is the calibration result for each initial condition, $sigma(k)$ is the parameters for each initial condition.

```
Final = Curv(:,1);
for k=2:opt
        for i=1:RecPwr
                tmp = Curv(i,k)-Curve(i,k-1)
                If (tmp>sigma(k)-sigma(k-1)) or (tmp>diff_mem(i)*0.5)
                        Final(i) = Curv(i,k)
                        diff_mem(i) = tmp
                end
        end
end
```

Figure 2.11 Optimization Procedure

Figure 2.12 shows the optimum finding result (the dash-dot-plus-sign line) based on four different initial conditions (solid line), in which the optimal result shows a better estimation than other conditions. Figure 2.13 shows the calibration results after using the procedure of the optimum-result-finding, in which we can see that the improvement of predictions for three channels compared with curves in Figure 2.9. In this example, we use four different initial conditions. More initial conditions can be used to improve the optimum-finding-results at the cost of higher computing load.

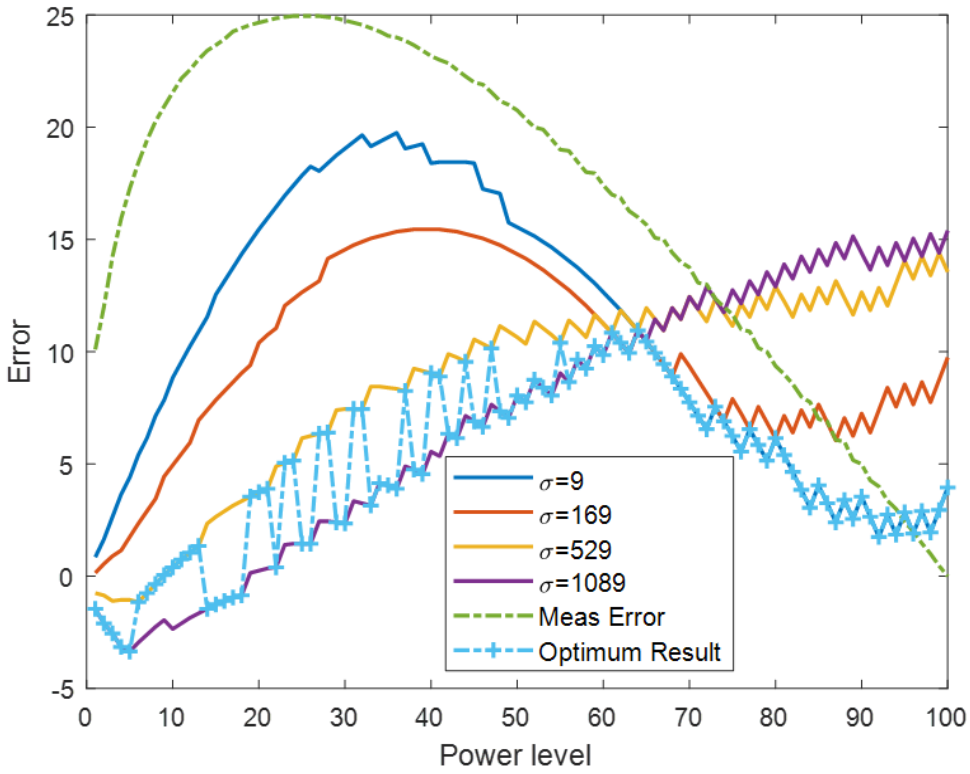


Figure 2.12: Comparison of calibration results based on various initial starting condition and optimum finding result

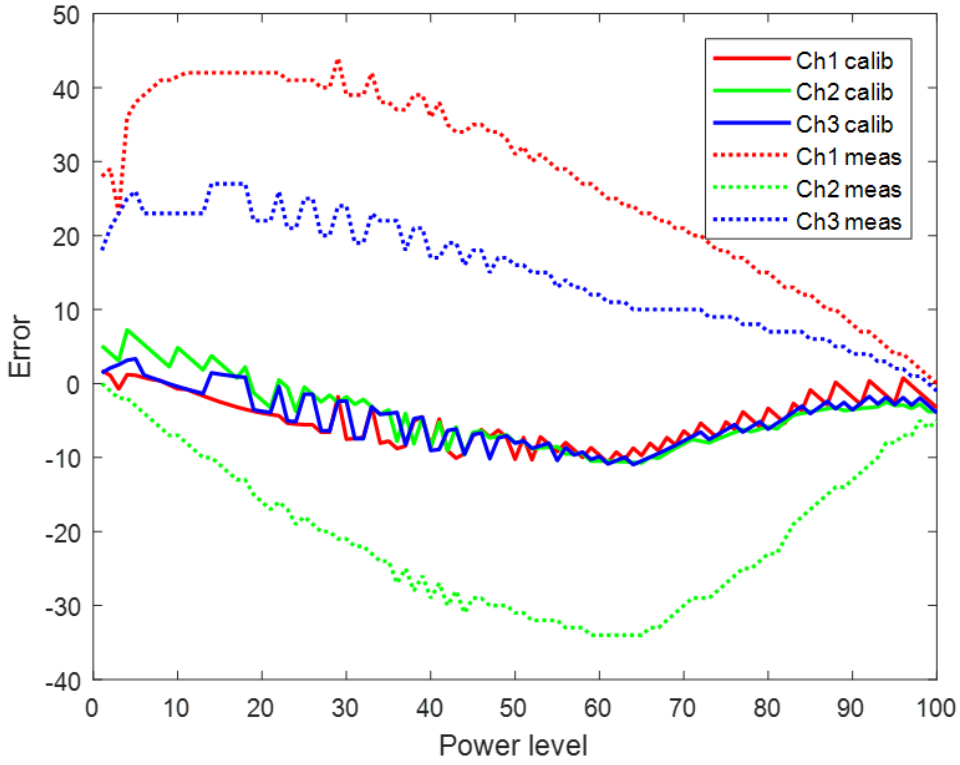


Figure 2.13: Calibration results after using optimum result finding procedure

In reality, since the noise would always be a factor to influence the condition of the signal, we apply the Gaussian noise to the three datasets, D_1, D_2, and, D_3, as mentioned before. Figure 2.14 and Figure 2.15 show the calibration results with two different noise power levels, from which we can see that the EM algorithm is robust enough to correctly predict the nonlinear transformation errors.

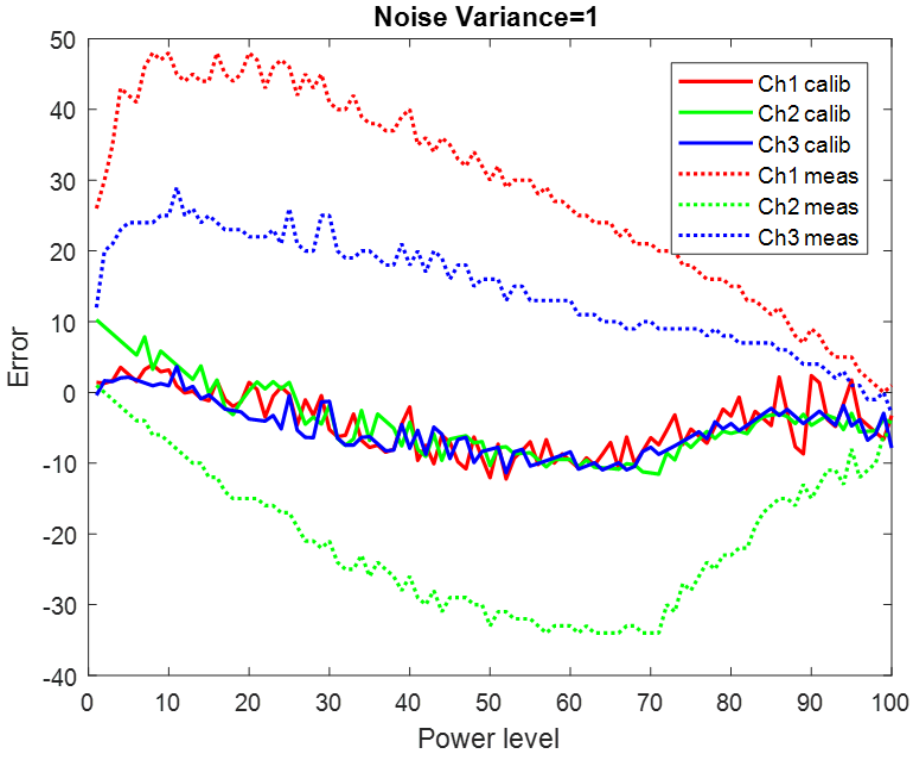


Figure 2.14: Calibration result when noise variance=1

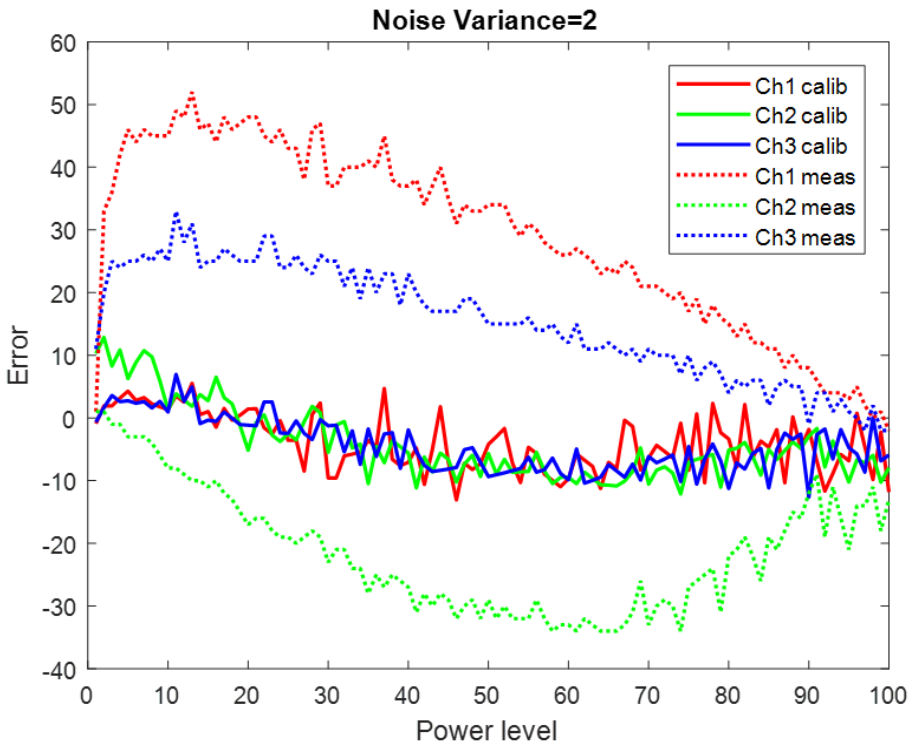


Figure 2.15: Calibration result when noise variance=2

2.6.4. Target Direction Estimation

One of the important missions for a multi-channel radar system or PAR is determining the direction (in azimuth and elevation) of a point target or a radiation source (such as interference source). The most common or easiest way to detect the location of the point targets is to perform the beamforming and find the peaks in corresponding azimuth or elevation degree. The angular resolution of the traditional beamforming is limited by the beamwidth of the antenna pattern. To reduce the angular resolution, some advanced algorithms can be utilized to overcome the antenna limitation and achieve super-resolution. In this section, the Direction of Arrival (DOA) estimation is investigated and provide the computing complexity for each advanced algorithm.

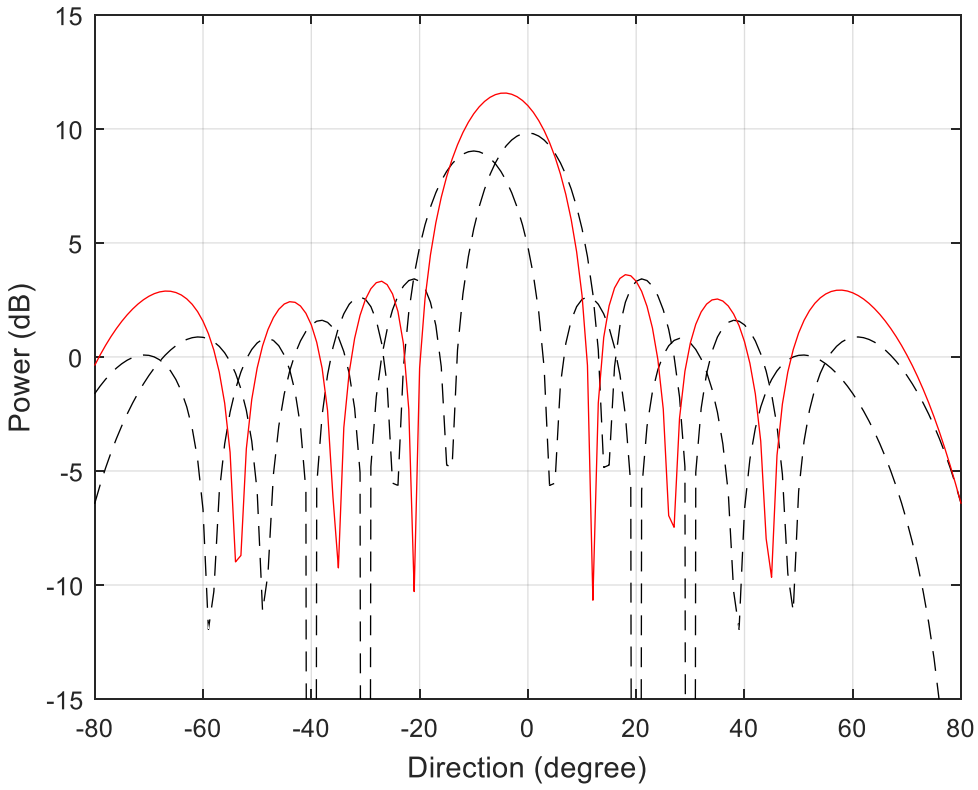


Figure 2.16: Composite beam response for two signals at 10 and 0 degree. The dashed curves are the responses to the individual signals, and solid curve is the composite response

Herein we assume the transmission medium is nondispersive so that the electrometric wave emitted from PAR propagates in straight lines, and the targets are in the far-field area of the array. Consequently, the radiation reflected from the targets impinging on the array is in the form of a sum of plane waves. Under those conditions, a coarse estimation of the DOA of a single target can be known from the phase differences applied to the beamforming weight factors, W_i, as shown in Equation (2.1). A more precise estimation is monopulse estimation [83], which is performed in principle by comparing the amplitude or phase from the sum-beam output and diff-beam output. If there is only one target present in the beamwidth, the monopulse estimation is unbiased and efficient (minimum variance) [84], and rather insensitive to measurement noise [85]. However, when return signal is a composite of multiple targets within one beamwidth, the beam response will have a single peak, as shown in Figure 2.16. The beamformer therefore would fail to resolve two targets, producing a single biased direction estimation. In this case, when the distance among each target is less than one beamwidth, other methods needed to be applied to obtain the truth DOA of each target. The ability to overcome the angular resolution of PAR, limited by its physical size of the aperture, is called super-resolution.

Super-resolution can be achieved by utilizing the multiple spatial samples of incoming wavefront, and some assumptions about the signal are needed to make at first. For example, if we know the incoming signal is the echo of less than N reflecting bodies, the DOA of each target can be estimated by the knowledge of that the signal space is orthogonal to the noise space. The resulting algorithm was called Multiple Signal Classification (MUSIC) [86]. Although the super-resolution algorithms can increase the

angular resolution, they are based on the prior knowledge of the status of targets from the conventional beamforming processing, which means that the conventional DOA estimation method, such as monopulse, should be conducted in the first stage, and then analyses the signal by using super resolution in detail [85]. Note that, for simplicity, the following discussion of super resolution algorithms deals only with single dimensional parameter, such as azimuth only DOA, and the narrow band signal of known frequency is assumed.

Many super-resolution algorithms have been developed including maximum likelihood [87] and maximum entropy method [88]. Although those algorithms have good performance, they consume considerable computing resources, especially when the target number is large [89]. Moreover, the signal model used by those two algorithms are biased and sensitive to parameter estimates [90]. Thus, research had been done to exploit the structure of data model and derive a new complete geometric solution for obtaining a reasonable approximate solution, which is named as MUSIC. MUSIC is an eigenvector projection procedure, in which the DOA is estimated by the fact that the signal space is orthogonal to the noise space. The system model used in PAR is based on the parameters listed in Table 2.4. The signal model is given as:

$$x = As + n, \tag{2.65}$$

where x is the M elements measured signal vector, s is the ground truth signal source vector with T elements, A is the steering vector with dimension of $M \times T$, and n is the noise vector. If the reflect signal from targets are modeled as stationary stochastic processes, they are assumed to uncorrelated with signal and possess a positive definite covariance matrix $R_s = ss^H$. Under this condition, the covariance matrix of measured

signal is given by $R_x = E\{xx^H\} = AR_sA^H + \sigma_0^2 I$, in which σ_0^2 is the noise power. Then, we perform the eigenvalue decomposition of the matrix R_x to get the eigenvector U and eigenvalue D. For $M > T$, it implies that in D the first number of T largest eigenvalues are corresponding to the reflected signal, the result of $M - T$ eigenvalues are from noise. So we can rewrite R_x as $R_x = U_sD_sU_s^H + U_nD_nU_n^H$, in which U_s and U_n are the signal subspace and noise subspace, and D_s and D_n are the diagonal matrix whose entries correspond to the eigenvalues associated with U_s and U_n. As the signal is uncorrelated with noise, so the DOA estimates are obtained by observing the peaks of the spatial function spectrum function $S(\theta)$ as

$$S(\theta) = \frac{1}{\mathrm{A}^H \times \mathrm{U_n}} \tag{2.66}$$

Table 2.4: Parameters used in the MUSIC

T	Target Number
N	number of range gate in the snapshot
M	number of antenna elements
D	number of directions in the detection area

To give a throughput estimation in MUSIC, we use the parameters listed in Table 2.4. There are $8N^3$, $23M^3$, and $8DM(M-T)+M$ flops in calculating R_x, eigenvalue decomposition of R_x, and the final spectrum function $S(\theta)$ respectively. In total, there are

$$8MN^2 + 23M^3 + 8DM(M-T) + M \tag{2.67}$$

complex number floating-point operations in MUSIC. To give an example, we use the parameters listed in Table 2.5 to see how much computing resources are needed by using

MUSIC. Based on Equation (2.70), the throughput will be around 320 *MFLOPS*. This workload does not seem large in this case, however, if we perform the 2-dimension search-for every elevation, the computational load would dramatically increase. An alternative algorithm, Estimation of Signal Parameters Via Rotational Invariance Techniques (ESPRIT), [90] can be utilized to reduce the computing requirements and achieve super-resolution DOA estimation at the cost of more number of antenna elements than MUSIC.

Table 2.5: Assumption parameters used in the DOA algorithms

T	5
N	1000
M	20
D	180
Single elevation volume scan time	500 ms

Unlike MUSIC, ESPRIT does not need the knowledge of array manifold nor searching over parameter space, which is computationally expensive. Like MUSIC, ESPRIT correctly exploits the underlying data model by using the knowledge of signal subspace and the noise subspace. Different from MUSIC, the ESPRIT exploits the displacement invariance of signal subspace induced by two identical subarrays X and Y, displaced from each other by distance d. The system model used in PAR is also based on the parameters listed in Table 2.5. The signal model is defined as

$$\begin{aligned} x &= As + n \\ y &= A\varphi s + n \end{aligned} \tag{2.68}$$

in which s is the $T \times N$ matrix impinging signals as observed by the subarrays X and Y, n is the noise vector, and φ is the subarray displacement vector. Each subarray contains $K = M - 1$ number of antenna elements, so the steering vector, A, is the matrix with dimension of $(M-1) \times T$. The total-least-square ESPRIT algorithm based on a covariance formation can be summarized as follows.

1. Obtain the receiving covariance matrix, from the measurement as

$$R_{xy} = [x \quad y] \times [x \quad y]^H \tag{2.69}$$

   The computation complexity in this step is $8K^2N$ flops.

2. Compute the eigenvectors of R_{xy} as shown in Equation (2.69). As the number of target is T, there are T eigenvectors associate with signal subspace, which means, for each subarray, it obtains the signal subspace, U_x and U_y, as a matrix with dimension of $K \times T$. In this step, the workload is $\mathbf{23} \times (\mathbf{2}\boldsymbol{K})^{\mathbf{3}} = \mathbf{204}\boldsymbol{K}^{\mathbf{3}}$ flops.

$$R_{xy} = \lambda U_{xy}, in\ which\ U_{xy} = [U_x\ U_y] \tag{2.70}$$

3. The invariance structure of the array implies the signal subspace from the aspect of entire array, U_s, can be decomposed into U_x and U_y as

$$U_s = \begin{bmatrix} U_x \\ U_y \end{bmatrix}. \tag{2.71}$$

4. Then we can compute the eigendecomposition of $U_s^H \times U_s$, as $U_s^H \times U_s = E\Lambda E^H$ and partition E into $T \times T$ submatrices, as

$$E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix} \tag{2.72}$$

   In this step, the matrix multiplication and the eigendecomposition would have $23 \times (2T)^3$ and $\mathbf{4} \times \boldsymbol{2T} \times \boldsymbol{K} \times \boldsymbol{2T}$ flops computation complexity separately.

5. Calculate the eigenvalues of $\Psi = -E_{12}E_{22}^{-1}$, as

$$\Psi = \lambda_k U, k = 1, \cdots, T \tag{2.73}$$

In this step, there are $\mathbf{11}\boldsymbol{T^3} + \mathbf{23}\boldsymbol{T^3}$ flops computation in total.

6. Estimate the $\theta_k = \sin^{-1}\{angle(\lambda_k)/\pi\}$ for DOA, and costs $\boldsymbol{6T}$ flops.

   In total, there are

$$204K^3 + 238T^3 + 6T + 32K^2N + 16T^2K \tag{2.74}$$

   flops in the ESPRIT.

Assume we take the same parameters for MUSIC in Table 2.5, the throughput of ESPRIT is $\mathbf{26}\ \boldsymbol{MFLOPS}$, which is 12 times less than the working load in MUSIC. The primary computational advantage of ESPRIT is from eliminating the search procedure happened in Equation (2.66), and directly produces signal parameters in terms of eigenvalue instead. This advantage would become even more pronounced in 2-dimensional DOA estimation, where the computational load grows linearly with dimension in ESPRIT, while for MUSIC it grows exponentially. On the other hand, MUSIC needs to know the array manifold, A, so it is sensitive to sensor position, gain and phase errors. In other words, MUSIC requires precise calibrations. The advantages of MUSIC method are that it is more accurate and stable in the term of SNR variations [91], and it can be extended for to arbitrary arrays of sensors. In contrast, ESPRIT only works for uniform arrays. In all, both methods can give high resolution DOA estimations for multiple targets. If the applications are more cost-sensitive, ESPRIT would be a better choice. If the operational environment is complex and has low SNR, it is better to use MUSIC procedure happened in Equation (2.66), and directly produces signal parameters in terms of eigenvalue instead.

3. System Architectures

Typically, an HPEC platform for PAR accommodates a computing environment [9] consisting of multiple parallel processors. To facilitate system upgrades and maintenance, the multiprocessor computing and interconnection topology should be flexible and modular, meaning that each processing endpoint in the backend system needs to be identical, and its responsibility entirely assumed or shared with other endpoints without interfering overall system operations. Moreover, the connection topology among each processing and I/O module should be flexible and capable of switching a significant amount of data from other endpoints. Figure 3.1 shows a top-level system description of a general large-scale array radar system. In receiving arrays, once data are collected from the array manifold, each transmits and receive module (TRM) downconverts the incoming I/Q streams in parallel. To support the throughput requirement, the receivers group I/Q data from each coherent pulse interval (CPI) and send grouped data for beamforming, pulse compression, and Doppler filtering. Beamforming and pulse compression are paired into pipelines, and the pairs process the data in a round-robin

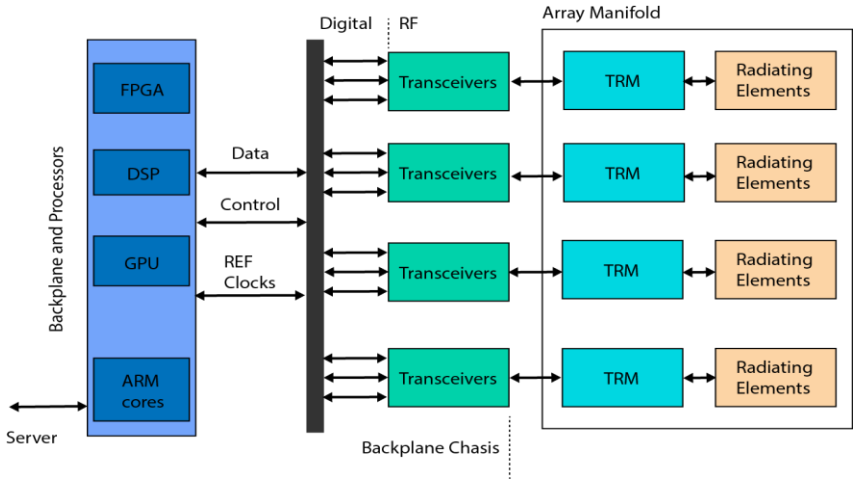


Figure 3.1: Top-level system digital array system concept

fashion. At each stage, data-parallel partitioning is used to mitigate the massive amount of computations into smaller, more manageable pieces.

Fundamental processing functions for PAR (e.g., beamforming, pulse compression, Doppler processing, and real-time calibration) require teras of operation per second for large-scale PAR applications [5]. Since such processing is executed on a channel-by-channel basis, the processing flow can be parallelized naturally. A typical scheme for parallelism involves assigning computation operations to multiple parallel processing elements (PE). In that sense, from the perspective of radar application, a data cube containing data from all range gates and pulses in a CPI is distributed across multiple PEs within at least one chassis. A good distribution strategy can ensure that systems not only achieve high computing efficiency but fulfill the requirements of modularity and flexibility as well. In particular, modularity permits growth in computing power by adding PEs and ensures that an efficient approach to development and system integration can be adopted by replicating a single PE [9]. The granularity of each PE is defined according to the size of a processing assignment that forms part of an entire task. Although finer granularity allows designers to attune the processing assignment, also poses the disadvantage of increased communication overhead within each PE [10].

As mentioned earlier, the features of a basic radar processing chain allow for independent and parallel processing task divisions. In pulse compression, for instance, the match filter operation in each channel along the range gate dimension can perform independently; as such, a large throughput radar processing task can be assigned to multiple processing units (PUs). Since each PU consists of identical PEs, the task would undergo further decomposition into smaller pieces for each PE, thereby allowing an

adjustable level of granularity that facilitates precise radar function mapping. At the same time, a centralized control unit is used for monitoring and scheduling distributed computing resources, as well as for managing lower-level modules. PU implementations based on the MTCA open standard can balance tradeoffs among processing power, I/O functions, and system management. In our implementation, each PU contains at least one chassis, each of which includes at least one MCH that provides central control and acts as a data-switching entity for all PEs, which could be an I/O module (e.g., RF transceiver) or a processing card. The MCH of each MTCA chassis could be connected with a system manager that supports the monitoring and configuration of the system-level setting and status of each PE by way of an IP interface. Within a single MTCA chassis, PEs exchanges data through the SRIO or PCIe fabric on the backplane, and the MCH is responsible for both switching and fabric management.

Figure 3.2 gives an example of using an MTCA chassis to implement the fundamental functions of radar signal processing. We will not go into details of each step since the purpose here is to illustrate the parallelism of the major steps, but a brief description of the system is in Chapter 4. Depending on the nature of data parallelism within each function, computing load is divided equally and a portion assigned to each

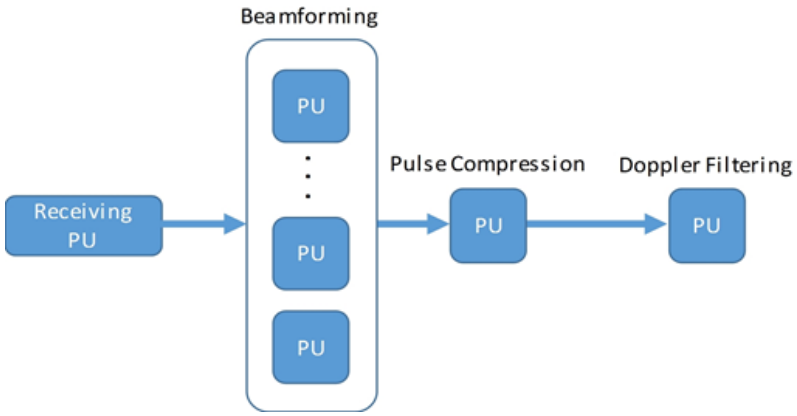


Figure 3.2: Illustration of the MTCA architecture in a PAR

PU. The computational capability is reconfigurable by adjusting the number of PUs, and for each processing function, a different PU can constitute at least one MTCA chassis with various types of PEs inserted into it, all according to specific needs. In the front, several PUs handle a tremendous amount of beamforming calculations, and by changing the number of PUs and PEs, the beamformer can be adjusted to accommodate different types and numbers of array channels. Since computing loads are smaller for pulse compression and Doppler filtering, assigning one PU for each function is sufficient in MPAR systems.

3.1. Parallelization Model

A parallelization model is typically a way of decomposing a signal processing algorithm into small portions, mapping each portion to the different processing unit, and reconstructing the results calculated from each portion. There are two fundamental types of parallelization: task and data parallelism. In data parallelism, the parallelization is accomplished by equally divided a data object into subjects, each of which is operated by a processing unit with a similar or identical computation. For example, matrices can be partitioned into blocks or submatrices, and matrix computation can be formulated regarding submatrices. The decomposition shown in Figure 3.3 is based on dividing the

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

(a)

$$\begin{aligned} &\text{Task 1: } C_{11} = A_{11}B_{11} + A_{12}B_{21} \\ &\text{Task 2: } C_{12} = A_{11}B_{12} + A_{12}B_{22} \\ &\text{Task 3: } C_{21} = A_{21}B_{11} + A_{22}B_{21} \\ &\text{Task 4: } C_{22} = A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

(b)

Figure 3.3: (a) Partition input and output matrices into 2×2 submatrices.
(b) A decomposition of matrix multiplication into four tasks based on (a)

output matrix C into four blocks and each of task computes one of these blocks independently. A and B are two $n \times n$ matrices, so each of submatrix in C is $n/2 \times n/2$. If each of processing endpoints has enough memory to store four of submatrices, it is easy to see that, with data decomposition, each task in Figure 3.3 does not need to communicate data between others, which is referred as embarrassingly parallel. However, if each endpoint has less memory to hold entire four submatrices, data parallel requires synchronization and communication among the parallel units.

Compared with data parallelism, the task parallelism looks for the independence among tasks, so that one algorithm can be divided into tasks and executed concurrently. An example of data sorting based on task parallelism is shown Figure 3.4 [10], in which Lines 11 and 12 indicates that the array is partitioned into two parts and each part can be solved recursively. Therefore, to parallelize the quicksort is to execute it initially on one processing unit, and then when the algorithm runs to line 11 and line 12, assign one of the subtasks to another processing unit. In this example, no communication is needed between tasks. However, in general, some tasks may use data produced by other tasks. Thus synchronization and communication may be needed in task parallelism schemes too.

In both data and task parallelism, tasks may need to exchange data with other tasks. This communication time can significantly impact the efficiency of parallel programming by requiring processors halt for the data. A suitable data communication pattern or strategy is a support of a stable and low latency HPEC system. When data exchanging is unavoidable, we can make the computations to be carried out in concert with communication. A ping-pong buffer mechanism is a simple technique that can make

```
1.  procedure QUICKSORT (A, q, r)
2.      if q<r
3.          x=A[q];
4.          s=q;
5.          for i=q+1:r
6.              if A[i]≤x
7.                  s=s+1;
8.                  swap(A[s],A[i]);
9.              end
10.         swap(A[q],A[s]);
11.         QUICKSORT (A,q,s);
12.         QUICKSORT (A,s+1,r);
13.     end
14. end
```

Figure 3.4: Example of Parallel Quicksort

the processor to do the computing job without waiting for the data. This technique requires extra data communication hardware, usually, Direct Memory Access (DMA), to prepare two sets of data buffers for all incoming and outgoing data streams as shown in Figure 3.5. While the DMA transfers the data into and out of the ping buffers, the processing core manipulates the data in the pong buffers. When both the processor and DMA finished the tasks, they switch the working buffers. By using the ping-pong buffer, processors activity can be distanced from memory fetching activity.

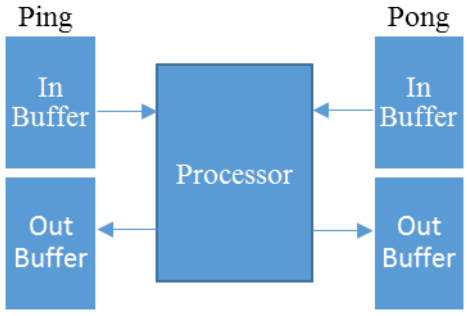
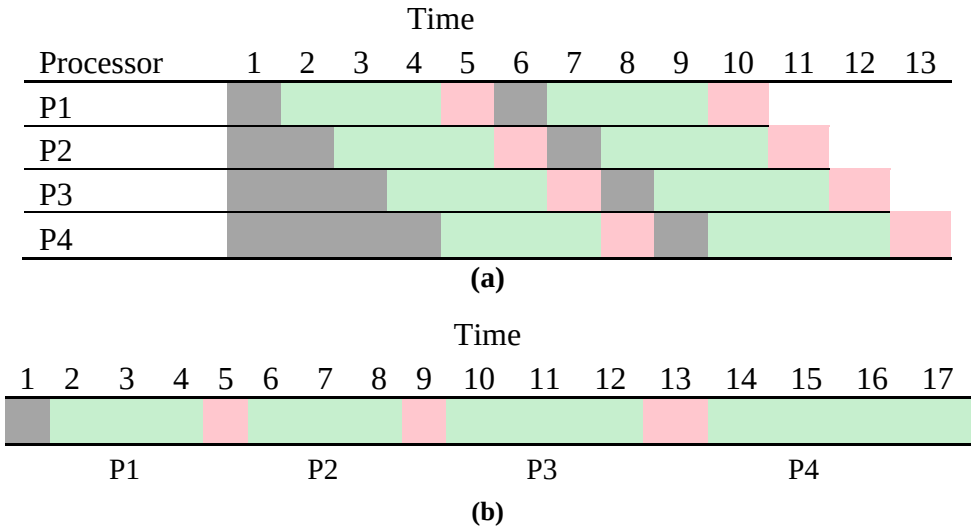


Figure 3.5: Ping-Pong Buffering Mechanism

Ping-pong buffer is used to “hide” the communication latency in the scale of the processing unit. In the large picture, to reduce the computing halt time among each parallel task two forms of scheduling can be employed: round-robin and pipeline mechanism. In the PAR application, those two methods are widely used in the front-end processing by exploiting the repetitive nature of the incoming data streams [9]. In a round-robin, as shown in Figure 3.6 (a), a data object is partitioned into four pieces, and each piece is dealt out to a free processor, which means P1 operates on an earlier data piece while a different set of parallel processors operate on a more recent data. As such, round-robin can be viewed as data-parallel parallelism. Once the P1 finished the processing of the earlier data, a new data set would be ready to be processed. The latency of the round-robin scheduling is the number of parallel processing point multiply by the initial waiting time for each processor.



The gray box represents processor is waiting for the data. The green box indicates the processor is working. The red box is the result is sending out.

Figure 3.6: Examples of round-robin and pipeline scheduling

Pipeline scheduling, as shown in Figure 3.6 (b), is to have each processor in an N-processor queue processing an entire data. By overlapping various tasks, pipelining

improves the overall processing throughput, which is quite suited to the PAR application. In the front-end, one processor could implement the baseband quadrature conversion, another pulse compression, and another beamforming, in which the output of the previous stage is the input of the following stage. In this case, each stage performs a distinct task, thus pipelining can be viewed as task parallelism. The latency of this approach is the depth of the pipeline, which means the time span between the data starting to flow in the pipeline and the last data flow out.

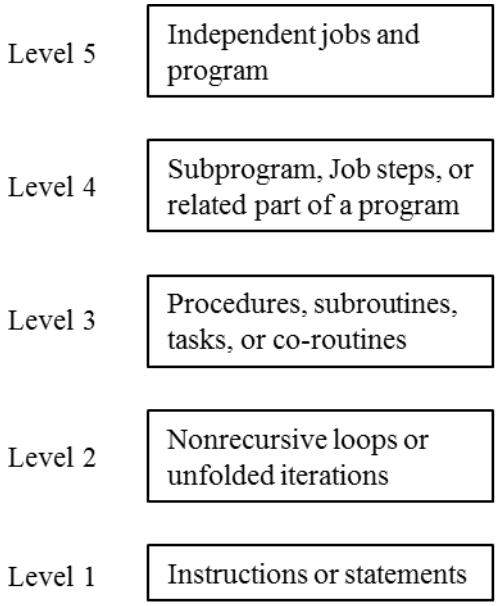


Figure 3.7: Levels of parallelism

Despite the forms of parallelism, the level of parallelism determines the granularity of the decomposition of the algorithm, from bit-level parallelism with a basic operation in processing core to sub-problems within the entire program. A decomposition into a large number of concurrent tasks is called *fine-grained*, and in contrast, a decomposition into a relatively small number of tasks is *coarse-grained*. [92] defines a five-level parallelism as shown in Figure 3.7. The lower the level, the higher degree of parallelism is achieved, while the communication overhead would be increased too. At

some point, the cost of communication will consume more time than the time saved by parallel implementation of a program. This communication overhead effectively limits the size and level of parallelism that may be productively employed. For evaluating the effectiveness of parallel algorithm implementations in parallel computing systems, *parallel speedup* and *parallel efficiency* are two important metrics. Speedup is a metric of latency improvement for a parallel algorithm compared with a serial algorithm distributed over M PUs, defined as:

$$S_M = T_S/T_P \tag{3.1}$$

In Equation (3.1), T_S and T_P are the latency of the serial algorithm and the parallel algorithm, respectively. Ideally, we expect $S_M = M$, or perfect speedup, although such is rarely achieved in practice. Instead, parallel efficiency is used to measure the performance of a parallel algorithm, defined as

$$E_M = S_M/M \tag{3.2}$$

E_M is usually less than 100%, since the parallel components need to spend time on data communication and synchronization [9], also known as *overhead*. In some cases, overhead is possible to overlap with computation time by using multiple buffering mechanisms. However, as the number of parallel computing nodes increases, the data size of each computing node lessens, meaning that the computing nodes would need to switch between processing and communication more often, thereby inevitably resulting in what is known as *method call overhead*. When the algorithm is distributed across more nodes, such overhead can preclude the benefit of using additional computing power. Parallel scheduling thus needs to minimize both communication and method call overhead.

3.2. Data Transportation and Backend Protocol

With more powerful and efficient processors, HPEC platforms can acquire significant computing power and meet scalable system requirements. However, HPEC performance is also limited by the availability of a commensurate high throughput interconnect network. Moreover, the scalability of communication fabric that providing achievable communication bandwidth to each processing node should grow along with the newly added multiple nodes. Since the communication overhead setback significantly impacts the efficiency of executing system functions, a proper implementation of the interconnection network among all processing nodes is critical to the performance the parallel processing chain. Two primary connection methods can be selected based on the distance of data transmission. For the long distance, chassis to chassis communication, Gigabit Ethernet or InfiniBand over copper of fiber cable would be a better choice. For a short distance board-to-board communication, in which the data are transmitted through the trace lines on the printed circuit board, the RapidIO and PCI Express are the two most common options. For both cases, they all employ multiple low-voltage differential signaling pairs, apply 8B/10B coding, and base on switched-serial interconnects; the differences are in the data packaging and routing strategy.

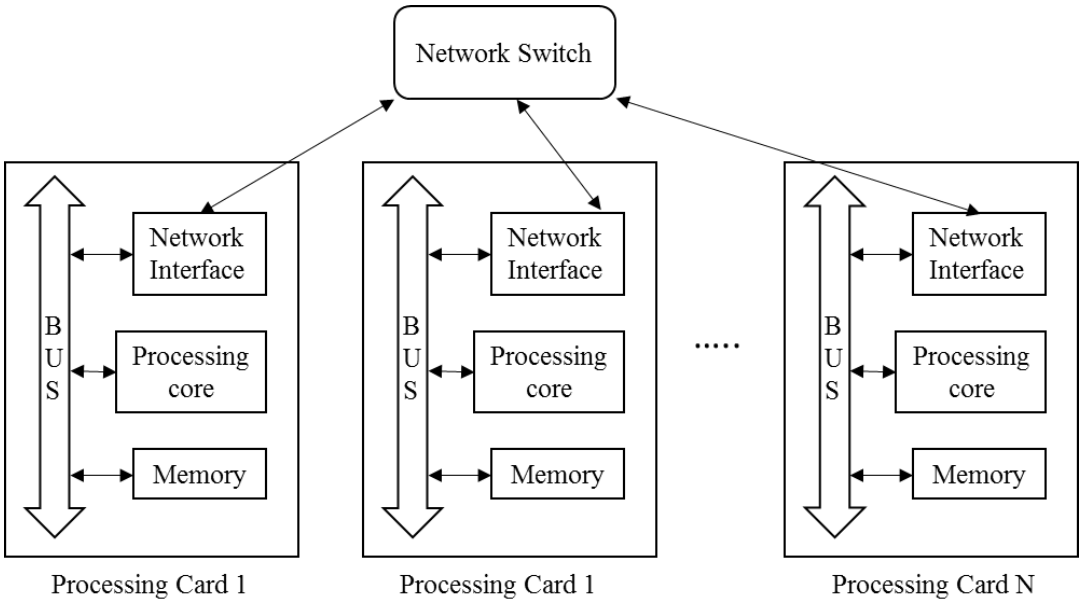


Figure 3.8: Typical interconnection fabric

Figure 3.8 shows a block diagram of a typical interconnection fabric, in which the communication is issued from the processing card and managed by a network switch. In the processing card, the network interface connects to the processing core and memory via the bus. On the network side, the network interface connects to the switch network through cables for out-of-chassis communication, or through copper traces on chassis backplane for board-to-board communication. Typically, multiple numbers of links are active concurrently in the network interface to increase the data throughput. The number of switches and the number of ports on each switch determine the scalability of the network, and the aggregate bandwidth across all of the paths defines the metric of network capacity.

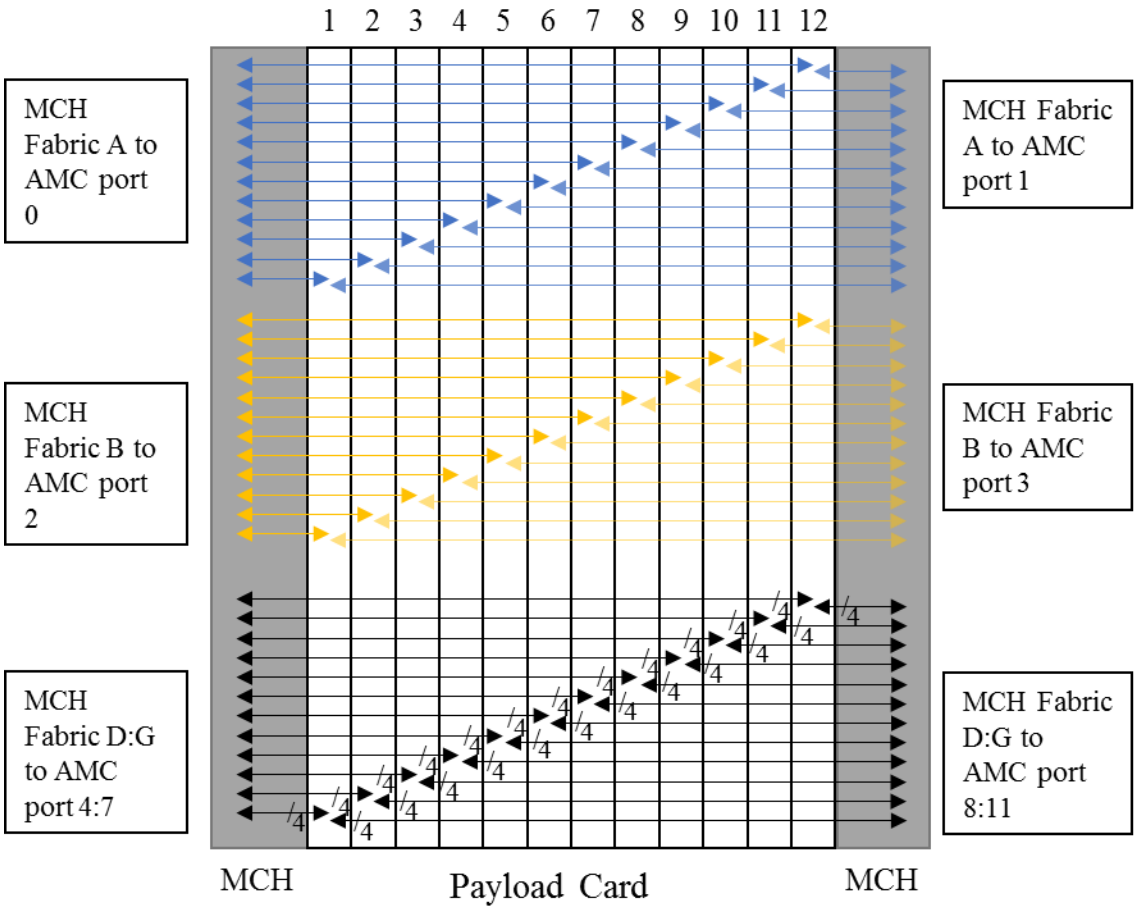


Figure 3.9: MTCA backplane configuration

As a concrete example, we consider the MTCA standard, which represents the latest attempt at increasing board to board communication on the backplane by leveraging the switched-serial interconnection fabrics. For the switch network, the MTCA supports multiple protocols, including Serial RapidIO, PCIe, SATA, and 10 Gigabit Ethernet (GbE). MTCA has two kinds of cards: standard AMC front model and MCH (MicroTCA Carrier Hub). AMC is a payload card which can be as a processing unit, or front-end data transmitting/receiving model, and can be inserted into the slots on the backplane of MTCA shelf. Each payload card can exchange the data through the high-speed differential trace lines on the backplane. MCH is responsible for the monitoring system status and provides data link switch for each payload card. To create larger, stable, and

redundant network, two MCHs can be used and connected in a dual-star configuration. A common backplane configuration for MTCA is 14-slots chassis with 12 payload cards and 2 MCH, as shown in Figure 3.9. Each MCH has an independent fabric network, providing the redundancy for the system. The fabric A for each MCH has one serial link, routed to ports 0 and 1 of AMCs, which is allocated for GbE in common. Fabric B has one serial link too and is allocated for protocol SATA through port 2 and 3 of AMCs. If the application requires a direct link between AMCs, the fabric B can be routed as inter-slot connections, allowing for one AMC directly communicate with another one without involving MCH as shown in Figure 3.10. Fabrics D to G use the four links to support data connectivity, known as fat pipes, which is usually used as PCIe or Serial RapidIO transmission.

Table 3.1: Typical COTS Interconnection Fabrics [93] [94]

	SRIO Gen 2	*SRIO Gen 3*	*PCIe Gen2*	*PCIe Gen3*	*10 GbE*
Signal pair	*1, 2, 4*	*1, 2, 4*	*1, 2, 4*	*1, 2, 4*	*4 (XAUI)*
Encoding	*8b/10b*	*64b/67b*	*8b/10b*	*128b/130b*	*8b/10b*
Channel	*~80-100 cm*	*~80-100 cm*	*~40-50 cm*	*~40-50 cm*	*100 m*
Bandwidth (4X)	*20 Gbps*	*40 Gbps*	*16 Gbps*	*32 Gbps*	*40 Gbps*
Latency	*sub μs*	*sub μs*	*sub μs*	*sub μs*	*tens of μs*

As mentioned before, multiple transmission protocols have been employed in HPECs. Much more often, people tend to think each protocol would be equal if they have similar peak bandwidth. However, each protocol is developed and optimized for different purposes of the application and types of processor. Typically, an interconnect will solve the problems in one application, but it would be less efficient if some conditions have

been changed. Table 3.1 presents the typical bandwidth and lane configurations for most widely used Commercial off-the-shelf (COTS) fabrics, in which the raw bandwidth is a fundamental parameter to determine the peak throughput of the system. However, the channel length, coding method, and latency would affect the efficiency of each protocol. Thus, when people choose the backend data protocol, the inherent protocol capabilities, supported topologies and latency should also be considered. The following use PCI Express, Serial RapidIO (SRIO), and 10 GbE as illustrative examples.

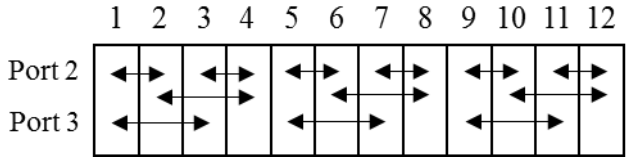


Figure 3.10: point to point connectivity between port 2 and 3

PCI Express, used as its name, is designed to make the host processor, usually CPU, to connect multiple numbers of peripheral devices. Subsequently, the topology of PCI Express is a hierarchy of buses with a single root complex, as shown in Figure 3.11. Although PCI Express switch uses the 32-bit or 64-bit device ID to forward the packets to the device or downstream switch, PCI Express specification does not support peer-to-peer communication. Implementing peer-to-peer connectivity requires researchers to create a new mechanism, such as “non-transparent bridge” [95], which could be exceedingly complex. In contrast, Ethernet and SRIO are routable protocols, in which more complex topology can be implemented. Note that both PCI Express and SRIO are designed for onboard or board-to-board communications, which shows low latency and higher data throughput compared to Ethernet. In summary, if the applications have clear hierarchy structure and no out-of-chassis communication, PCI Express would be a good choice for connectivity.

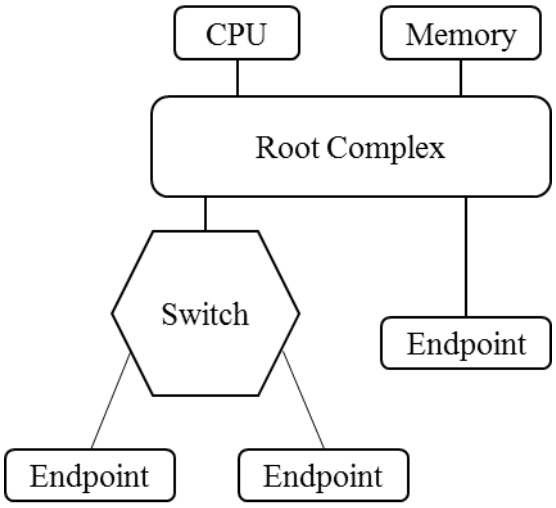


Figure 3.11: Typical PCI Express system topology

Ethernet is the oldest and the most widely used protocol compared with other two protocols. For the past 35 years, although Ethernet has been evolved to gain more bandwidth, the improvement has been steadily in recent years and is reaching its bottleneck. Ethernet has the advantage of long-distance transmission, which suits the chassis-to-chassis communication. Moreover, as the Ethernet has been used widely, the cost of Ethernet-related chips, boards, and interfaces is relatively low. Identical to SRIO, Ethernet is using the *routing table* to switch the packages. The drawback of Ethernet is the significant latency and communication overhead. Ethernet was originally designed for long distance transmission, so it requires a collection of protocols and related networking functionality to compensate the error brought from the noisy transmission environment. Those extra protections increased the overhead of Ethernet package and made the transmission inefficient. For example, for a 100 Bytes payload, the efficiency is only 60% if the UDP package is used. Using TCP package or smaller sizes of payload would be less efficient. The large overhead also increases the computing burden of Ethernet switch, leading to the problem of high power consumption and rising cost of

switch and latency. In summary, Ethernet is suitable for the long distance, high latency transmission applications, such as data center, in which the power consumption and the cost of Ethernet switch are not sensitive, and many sophisticated functionalities such as firewalls, prioritization, and virtual LAN can be utilized [93].

RapidIO is a reliable, efficient, and highly scalable protocol. Compared with PCIe, SRIO is designed to support both point-to-point and hierarchical models. The package routing in the SRIO is based on the device ID, and the switching decisions are merely based on source and destination ID. This feature allows SRIO to add new transaction types without changing the switch. Moreover, it demonstrates a better flow control mechanism than PCI Express and Ethernet. The flow of control in Ethernet is implemented in the software, which requires significant buffering capabilities to allow for retransmission [96]. PCI Express and RapidIO flow of control both offer a retry mechanism based on tracking credits inserted into packet headers in physical layer[97]. Also, SRIO also has logical layer flow control mechanisms by metering the admission of packets to the fabric, which is similar to the Ethernet flow control. Compared with Ethernet, logical level flow control in RapidIO is implemented by hardware, freeing precious processing core. RapidIO also includes a virtual output queue backpressure mechanism, which allows switches and endpoints to learn whether data transfer destinations are congested [96]. Given those characteristics, SRIO allows an architecture to strike a working balance between high-performance processors and the interconnection network.

In light of those considerations, we use SRIO as our backplane transmission protocol [98], and our current testbeds are based on SRIO Gen 2 backplanes. Each PE

has a four-lane port connected to an SRIO switch on the MCH. In our system, SRIO ports on the C6678 DSP support four different bandwidths: 1.25, 2.5, 3.125, and 5 Gb/s. Since SRIO bandwidth overhead is 20% in 8-bit/10-bit encoding, the theoretical effective data bandwidths are 1, 2, 2.5, and 4 Gb/s, respectively. In reality, SRIO performance can be affected by transfer type, the length of differential transmission lines, and the specific type of SRIO port connectors. To assess SRIO performance in our testbed, we conducted the following throughput experiments.

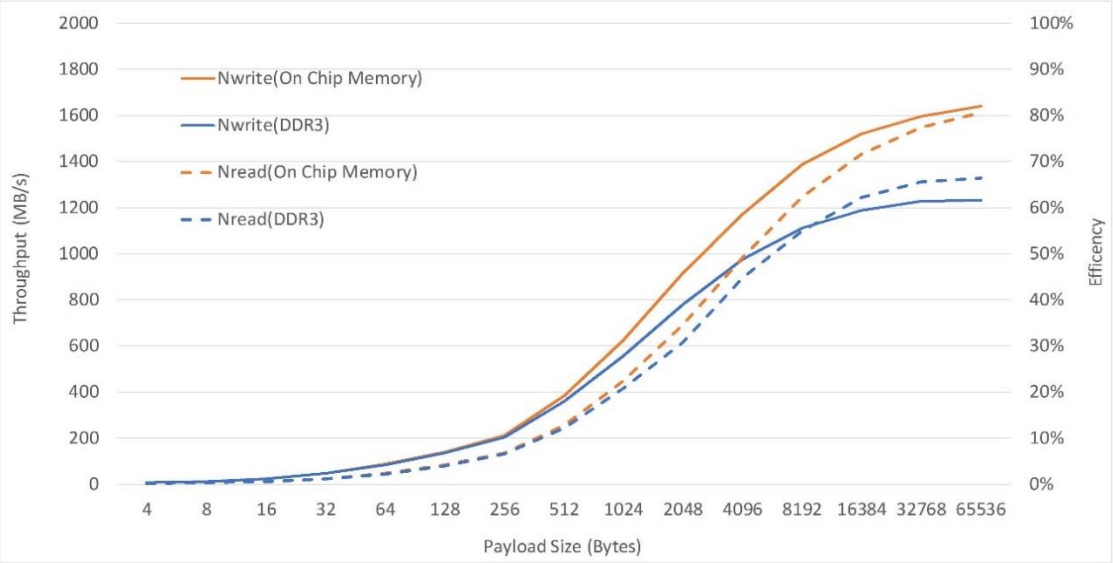


Figure 3.12: Data throughput experiment results in MTCA-based SRIO testbed

Figure 3.12 shows the performance of the SRIO link in our MTCA test environment by using NWrite and NRead packets in 5 Gb/s, four-lane mode. Performance is calculated by dividing the payload size by the elapsed transaction time from when the transmitter starts to program SRIO registers and the receiver has received the entire dataset. First, the performance of the SRIO link is enhanced along with larger payload sizes. Second, the closer the destination memory to the core, the better the performance

achieved with the SRIO link. Optimally, SRIO 4× mode can reach a speed of 1,640 MB/S, which is 82% of its theoretical link rate.

3.3. Processing Units

For an embedded signal processing unit, designers often have to balance various competing objects: development cost, performance, and time to the market. It is impossible to meet all the requirements at the same time. So, the designer needs to select a proper implementation technology, according to the constraints specific to the application. In this dissertation, we propose a low-cost experimental PAR computing platform. The capacity to fulfill the canonical radar processing for a scalable PAR is our main concern, and at the same time, we want to reduce the cost of the system. Those requirements confine the choice of the principal processing power for the most of the processing tasks should be in the area of COTS. Finally, we choose to use the MTCA to build the processing unit for the PAR front-end processing and part of backend processing. The processing units are the highly parallel homogeneous computation platform, in which multiple numbers of payload cards can be inserted. Each processing unit can be connected with others and form up a scalable computing system for PAR. Figure 3.13 shows the MTCA based processing unit, in which there are 12 slots for payload cards and two slots for the MCH. As mentioned in Section 3.2, each payload card can communicate with others via switch fabric on the backplane, and the MCH would provide the routing capability and system monitoring. The out-of-chassis communication can be done by using the high-speed ports on one or each of the payload cards.



Figure 3.13: MTCA based processing unit

In our test platform, there are two types of payload cards: RF transceiver module and DSP module. The RF transceiver module is AMC518 + FMC214 from VadaTech©, which has four receiving channels, two transmitting channels, and one Xilinx Zynq FPGA. The DSP module is the EVMK2H or EVM6678 from TI©. By using MTCA, RF transceiver, and DSP processing module, a scalable HPEC platform for PAR application can be built. An example showing in Figure 3.14 illustrates a simple front-end processing platform. After the return signal sampled by ADCs, FPGA on AMC518 would do the down conversion and basic digital filtering, and then the data would be transmitted through 4 lanes of SRIO to the DSP module via the backplane fabric. In the DSP, the data coming from all the FPGA would be combined and sent to the computing PU through Hyperlink. In the computing PU, DSP modules take the responsibility of performing canonical radar signal processing algorithms. Finally, the data would be transmitted to

the next stage through Hyperlink ports. In the computing PU, the number of DSP module is determined based on required computational loads. Moreover, each computing PU can be connected with others by way of the Hyperlink port. With this proposed PU, the computing power of HPEC can be scaled horizontally by connecting more MTCA chassis, and scaled vertically by adding more DSP or RF transceiver model in MTCA chassis.

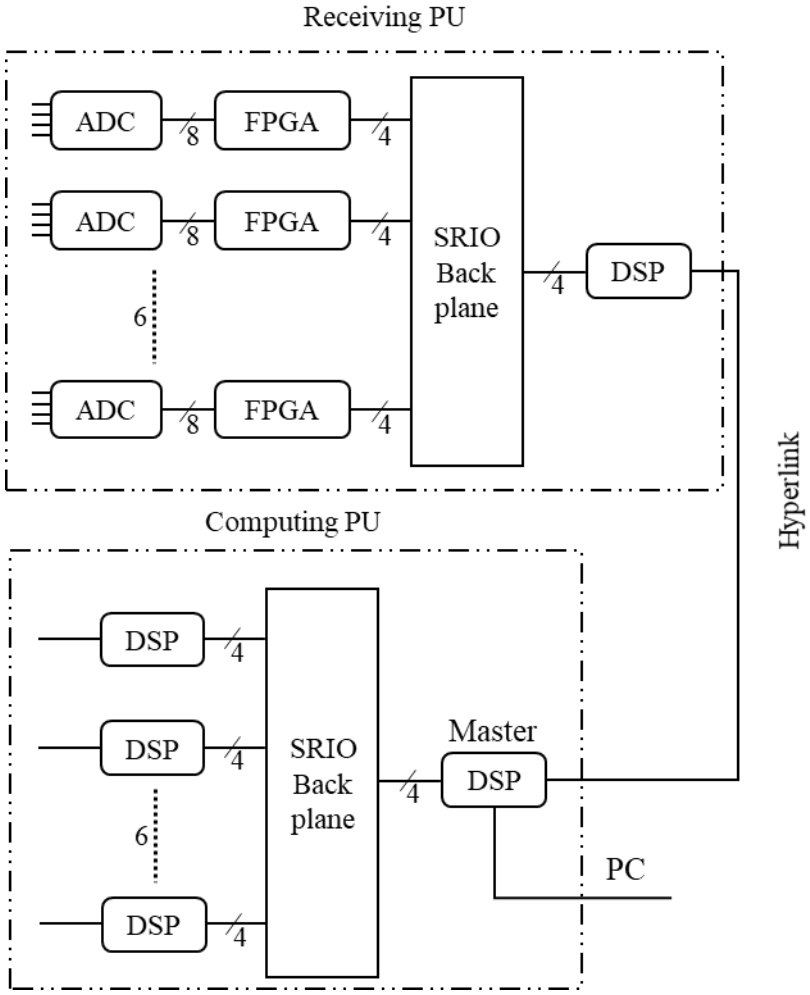


Figure 3.14: Simple example of a PU-based architecture

In each computing PU, the master DSP is also responsible for sending the commands from PC to other DSPs. Those commands would instruct each DSP the size of a data cube and other parameters related to the processing. After receiving those commands, DSP

would start to set up the dedicated memory region for the communication and computing, and configure the EDMA and interrupt registers. When the initialization is finished, DSP will wait for the command to start the processing. By using this method, the system has the flexibility of controlling the computing and communication load of each PU and makes the possibility of that progressively increasing the number of PU.

3.4. System Synchronization

3.4.1. General Calibration Procedure

Calibrating a fully digital PAR system is a complex procedure involving four general stages, as shown in Figure 3.15. During the first stage, transmit-receive chips in each array channel need to calibrate themselves regarding DC and frequency offsets, on-chip phase alignment, and local oscillator calibration. Those problems always relate to the issue of the signal integrity, power integrity, and electromagnetic compatibility, which is quite complicated and the most effects to solve them are based on experience and felt as "black magic'. So, the chance of the first failure is quite high, and multiple version of PCB design would be a common situation until the performance can be meet the requirements. During the second stage, subarrays containing fewer channels and radiating elements are aligned precisely in the chamber environment by way of near-field measurements, plane wave spectrum analysis, and far-field active element pattern characterizations. Note that for the small antennas, which width of radiators is smaller compared with the wavelength, λ, the near field region is a radius $r \ll \lambda$. While for the large antenna, the near field region is a radius $r = 2D^2/\lambda$, in which D is the aperture of antenna. In the second stage, the focus falls upon antenna elements, not the digital

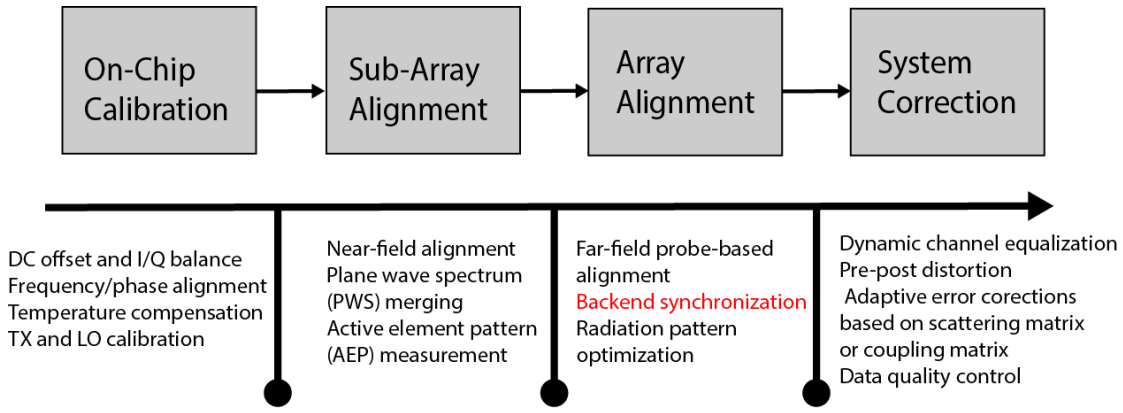


Figure 3.15: General system calibration procedure for DAR and the focus of this work

backend, and initial array weights for forming focused beams at the subarray level are estimated precisely. In the near filed measurements, as shown in Figure 3.16, it involves two calibrations: transmitting and receiving calibration. In transmitting calibration, a test probe scan across each antenna in the subarray to directly measure the phase and amplitude. In the receiving calibration, the measurements are the output directly from the ADC without mixing with the local oscillator, which would prevent the error brought from the local oscillator. The measured value by each antenna are normalized and compared. So, we can adjust the phase shifter and attenuators respectively and make each antenna have the same response function. This near-field measurement requires an automated precise probe control in an anechoic test chamber. Therefore, this method is an initial factory calibration rather than in-field calibration [71]. Plane wave spectrum analysis [99] is to acquire the far-field pattern of array by extracting the information for near-field measurement. Unlike the near-field calibration mentioned before, the data obtained in the plane wave spectrum analysis is by sampling the antenna pattern on a spherical surface. The near-field scanning can give much more information than the far-field, because many details can be computed by using near-field data but difficult to be measured in far-field [100]. By using the expansion coefficients and far-field terms for

the modes, the transformation between near-field and far-field can be build. The calibration is performed by using the information calculated from the near-field data.

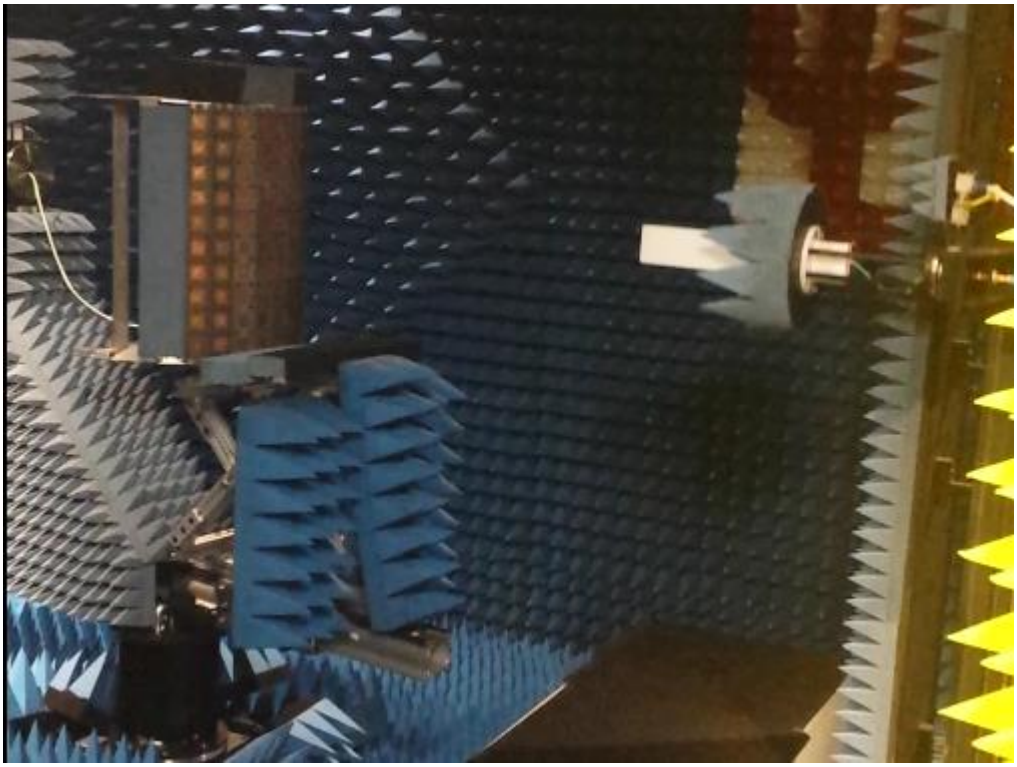


Figure 3.16: Measurement of radiation pattern from a PAR at the near-field range

In the third stage, far-field full array alignment is performed in either chamber or outdoor range environments. Similar to the near-field measurement, this stage requires a far-field probe in the loop of the alignment process and requires synchronization and alignments in the backend. We use a simple unit-by-unit approach to ensure that when each time a subarray is added it maximizes the coherent construction of the wavefront at each required beam-pointing direction. At first, only one subarray is excited, and then by adjusting the phase shifter, the phase offsets with the highest received power level from the test probe is recorded. Repeat this procedure for the rest of subarrays. In the end, the whole array would be calibrated. Note that these array-level weights are combined with chamber-derived initial weights from the second stage to optimize array radiation patterns

for all beam directions numerically. When multiple beams are formed at once, the procedure repeats for all beamspace configurations. Initial factory alignment is finished after this stage. As multiple numbers of the subarray are used, and each subarray has its own local oscillator, it is important to synchronize all the subarray unless the array pattern would be unstable and lose its focus. The aim of the final stage is to ensure the consistency of system performance after the system is shipped for the field deployment. In the field, the working condition would be changed regarding temperature, electronics drifting, and platform vibration, those perturbations would affect the characteristic of RF components, and a calibration is needed to offset those performance deviations. Based on internal sensor (i.e., calibration network) monitoring data, algorithms in the backend perform channel equalization and pre-post distortions, as well as correct system errors of deviations from the factory standard. The final step entails data quality control, which compares the obtained data product with analytical predictions to further correct biases at the data product level for desired pointing.

3.4.2. Backend Synchronization

Our study focuses only on backend synchronization during the third stage, a step necessary before parallel, multicore processing can be activated. Also, synchronized backend enables that reference clock signals in the front-end PU (and the RF chipsets such as AD9361/9371/9375 in the front-end PU) to be aligned through FPGA Mezzanine Card (FMC) interface. For the testbed architecture in Section 3.3, the front-end PU, referred to as simply “front-end” in this section, of the digital PAR systems includes a

number of array RF channels. In each channel, there is an integrated RF digital transceiver with an independent clock source in its digital section.

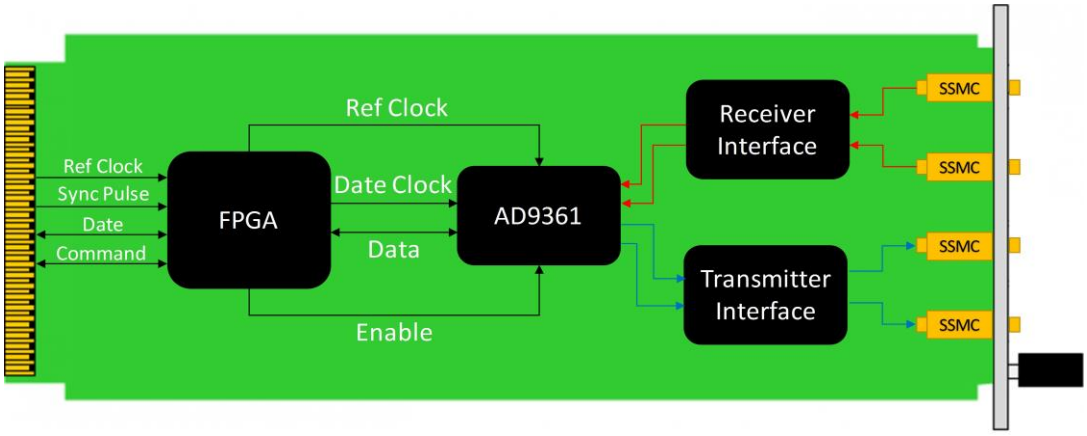


Figure 3.17: PU frontend AMC module architecture

Synchronization in this front-end system can be categorized according to either in-chassis or multi-chassis synchronization. In-chassis synchronization ensures that each front-end AMC in a chassis works synchronously with those in the other chassis. Figure 3.17 shows the architecture of a dual-channel front-end AMC module, which is based on an existing product from VadaTech. The Ref Clock and Sync Pulse in Figure 3.17 are radial fan-out by the MCH to each slot in the chassis, and each front-end AMC uses the Sync Pulse and Ref Clock to accomplish in-chassis synchronization. As an example, Figure 3.18 shows the timing sequence of synchronizing two front-end AMCs. Since commands from the remote PC server or other MTCA chassis may arrive at AMC 1 and AMC 2 at different times, transmitting or receiving synchronizations requires sharing the Sync Pulse between the AMCs. When AMCs acknowledge the command and detect the Sync Pulse, the FPGA triggers the AD9361 chip on both boards at the falling edge of the next Ref Clock cycle. By using that mechanism, multichannel signal acquisition and generation can be synchronized within a chassis. The accuracy of in-chassis

synchronization depends on how well the trace length is matched from an MCH to each AMC. If the trace length is fully matched, then synchronization will be tight.

For multi-chassis synchronization, the chief problem is so-called *clock skew*, which to overcome, requires a clock synchronization mechanism. The most common clock synchronization solution is Network Time Protocol (NTP), which synchronizes each client based on messaging with User Datagram Protocol [101]. However, NTP accuracy ranges from 5 to 100 ms, which is not precise enough for PAR application [102]. To get more accurate synchronization in the local area network, the IEEE 1588 Precision Time Protocol (PTP) standard [103] can provide sub-microsecond synchronization [104]. To implement PTP, the front-end chassis needs to be capable of packing or unpacking Ethernet packets, and additional dedicated hardware and software are required, which increase both the complexity and cost of the front-end subsystem. A better method of implementing multi-chassis synchronization would take advantage of GPS pulse per second (PPS), since by connecting each chassis to a GPS receiver, the MCHs can use PPS as a reference signal to generate the Ref Clock and Sync Pulse for in-chassis synchronization. Because the PPS signal among different MCHs is synchronized, the Ref Clock and Sync Pulse in each chassis is phase matched at any given time. However, when the GPS signal is inaccessible or lost, the front-end subsystem should be able to stay

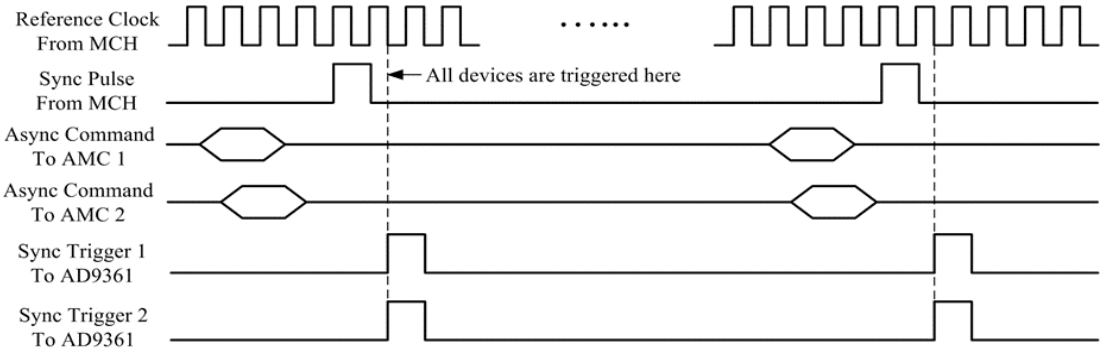


Figure 3.18: Frontend in-chassis synchronization timing sequence

synchronized by sharing the Sync Pulse from a common source, which could be an external chassis clock generator or a signal from one of the chassis. In both methods, the trace length to each MCH from the common Sync Pulse source can vary, thereby making propagation time delay of the Sync Pulse from each chassis differ. To address this issue, we need to know the delay time difference of each chassis compared with the reference (i.e., master) chassis. With that knowledge, all chassis can use the time difference as an offset to adjust the triggered time.

To implement that approach, we designed a clock counter to measure elapsed clock cycles between the Sync Pulse and the return Sync Beacon, the latter of which is transmitted only from antennas connected to the reference chassis. Since the beacon arrives at all antennas simultaneously, each front-end subsystem stops its counter at the same time. The time differences in delay can be obtained by subtracting the counter number from each slave chassis to the reference chassis. Figure 3.19 illustrates a model timing sequence after each chassis receives the Sync Pulse. At time T0, the reference chassis begins to transmit Sync Beacon and starts the counter. After two and a half clock cycles of propagation delay, the slave chassis launches the counter as well. At time T3, the Sync Beacon is received by both chassis, however, since the chassis detect the signal only at its rising edge, the reference chassis detects the signal at time T5 with counter

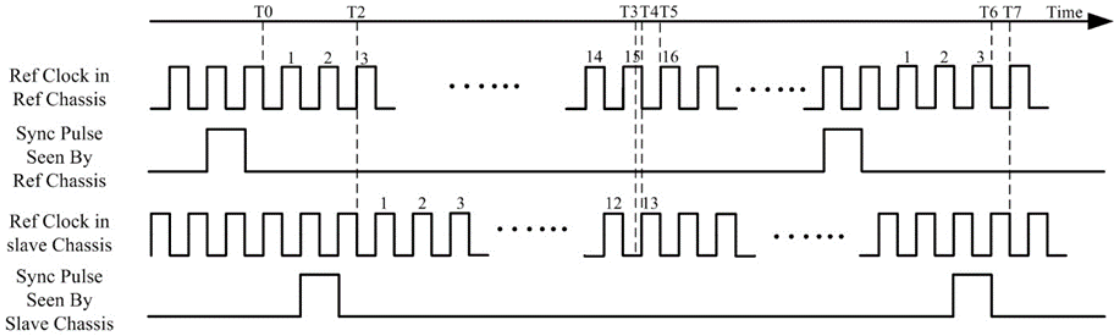


Figure 3.19: Example timing sequence of multi-chassis synchronization

number 16. By contrast, in the slave chassis, the counter stops at 13. In turn, when the Sync Pulse is received the next time, the reference chassis is delayed by three clock cycles and triggers AD9361 at time T6, whereas the slave chassis starts it at T7. In our example, T6 is not the same as T7. Such deviation arises because the clock phase angle between the two chassis is not identical. When this phase angle approaches 360 degrees, it is possible for the Sync Beacon to arrive when the rising edge of one clock has just passed, while the rising edge of the next clock cycle is still approaching. In the worst-case scenario, only one clock cycle synchronization error will occur, meaning that the accuracy of multi-chassis synchronization refers to the period of the reference clock. One way to enhance its accuracy is to reduce the period of the reference clock; however, the sampling speed of ADC confines the shortest period of the clock, because a front-end AMC cannot read new data in every clock cycle from ADC when AMC's reference clock frequency exceeds ADC's sampling speed. In our example, since the maximum data rate in AD9361 is 61.44 million samples per second, the inter-chassis synchronization accuracy without using the GPS signal is 16 ns.

3.5. System Performance Evaluations

In this section, we will demonstrate the processing capacity for FFT, vector multiplication, and data corner turn for each processing node. At first, we may introduce the principle of cache and locality, which is a fundamental knowledge of improving the efficiency of computing. From a signal processing application perspective, ideally, a larger and faster on-chip memory is better. However, the performance of processors has improved faster than the pace of memory. As a result, the high-speed and large size on-

chip memory is expensive, so only a small size high-speed on-chip memory is used, which causes the problem that the on-chip memory may not obtain enough data for the computing, so the processor needs to stall, waiting for the data to be cached. To solve this problem, the memory hierarchical can be used to reduce the cost and maintain the high computing efficiency, as shown in Figure 3.20. A fast but small size memory is placed beside the processing core, in which the access time is one clock cycle from processing core to Level 1 cache. This level memory are maintained by the cache controller, which could predict the processor's access pattern and pre-fetch the data from the external memory to the cache. The next lower memory levels are larger but slower than Level 1. Through this type of architecture, the average memory access time will be closer to the access time of the fastest memory rather than to the access time of the slowest memory [105].

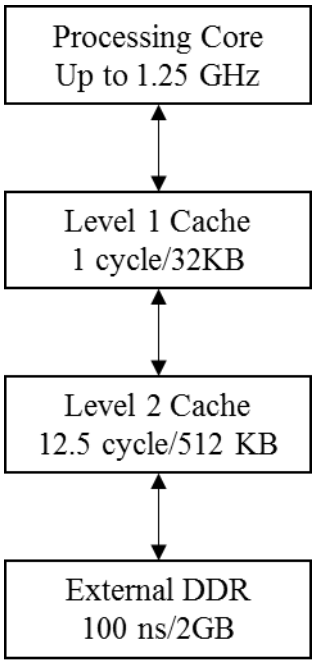


Figure 3.20: TI C66x DSP Hierarchical Cache

3.5.1. FFT performance

The Fourier transform is to transform the signal from time domain into frequency domain. This transformation can be compared as a prism separating the sunlight into the different colors (frequencies). In the digital system, to compute the Fourier transform the analog signal should be sampled at discrete intervals and then applied the discrete Fourier transform to the digitalized data. The direct expression for the computation of DFT is listed below:

$$X[k] = \sum_{n=0}^{N-1} x(n) W_N^{kn}, k = 0,1,\cdots,N-1 \tag{3.3}$$

in which $W_N^{kn} = e^{-j2kn\pi/N}$ is called the twiddle factor. The set of twiddle factors can be computed ahead of time and saved in the length of DFT is known. There are N^2 complex multiplications and $N(N-1)$ complex additions for an N-point DFT. So, there are $8N^2 - 2N$ complex operations. To reduce the number of computation, we may take advantage of twiddle factor by writing the DFT expression into a summation of the odd-number points, and even-number points, showing as:

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{nk} + \sum_{n=N/2}^{N-1} x[n] W_N^{nk}. \tag{3.4}$$

W_N^k can be factored out of the odd number part to get

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x\left[n + \frac{N}{2}\right] W_N^{\left(n+\frac{N}{2}\right)k} \tag{3.5}$$

Since $W_N^{kN/2} = e^{-jk\pi} = (-1)^k$, the above equation can be rewritten as

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] + (-1)^k x\left[n + \frac{N}{2}\right] \right\} W_N^{nk} \tag{3.6}$$

Equation (3.6) simply divides the DFT into two smaller DFTs, so this method is named as radix-2 FFT, in which the number of computations can be reduced to $5N \log_2 N$ operations. Another popular algorithm is the radix-4 FFT, which express Equation (3.3) as four summations, then divides it into four equations, as shown below:

$$\begin{aligned} X[k] = \sum_{n=0}^{\frac{N}{4}-1} \Big\{ & x[n] + (-j)^k x\left[n + \frac{N}{4}\right] + (-1)^k x\left[n + \frac{N}{2}\right] \\ & + (j)^k x\left[n + \frac{3N}{4}\right] \Big\} W_N^{nk} \end{aligned} \tag{3.7}$$

The radix-4 FFT combines two stages of a radix-2 FFT into one, so half as many stages are required. The computation load for radix-4 FFT is $4.25N \log_2 N$, which is 15% less than radix-2 FFT.

To arrive at a four-point DFT decomposition, since $W_N^4 = W_{N/4}$, Equation (3.4) can be written as four $N/4$ points DFTs, as

$$X[4k] = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] + x\left[n + \frac{N}{4}\right] + x\left[n + \frac{N}{2}\right] + x\left[n + \frac{3N}{4}\right] \right\} W_{\frac{N}{4}}^{nk} \tag{3.8}$$

$$X[4k+1] = W_N^n \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] - jx\left[n + \frac{N}{4}\right] - x\left[n + \frac{N}{2}\right] + jx\left[n + \frac{3N}{4}\right] \right\} W_{\frac{N}{4}}^{nk} \tag{3.9}$$

$$X[4k+2] = W_N^{2n} \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] - x\left[n + \frac{N}{4}\right] - x\left[n + \frac{N}{2}\right] - x\left[n + \frac{3N}{4}\right] \right\} W_{\frac{N}{4}}^{nk} \tag{3.10}$$

$$X[4k+3] = W_N^{3n} \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] + jx\left[n+\frac{N}{4}\right] - x\left[n+\frac{N}{2}\right] - jx\left[n+\frac{3N}{4}\right] \right\} W_{\frac{N}{4}}^{nk} \quad (3.11)$$

$X[4k]$, $X[4k+1]$, $X[4k+2]$, and $X[4k+3]$ are $N/4$ point DFTs. So, a DFT of length N has been factored into four DFTs of length $N/2$, in which each of $N/4$ point is a sum of a four input samples $x[n]$, $x[n+N/4]$, $x[n+N/2]$, and $x[n+3N/4]$, multiplied by either 1, -1, j, or $-j$. The sum is multiplied by a twiddle factor W_N^0, W_N^n, W_N^{2n}, or W_N^{3n}. The same factorization can be applied to each of these smaller DFTs, and so on, until the original DFTs has been factored into a four-point DFTs.

To implement computing algorithm on DSP, especially for FFT, we need to reduce cache misses and improve the commuting efficiency by aligning the data based on the computing sequence order. Ideally, researchers can arrange the data array and twiddle factor array in the computing sequence, however, usually, the incoming data order is fixed and additional memory management time would cost more than that saved from ordered sequence computing. In this case, only the twiddle factors, W_N^0, W_N^n, W_N^{2n},

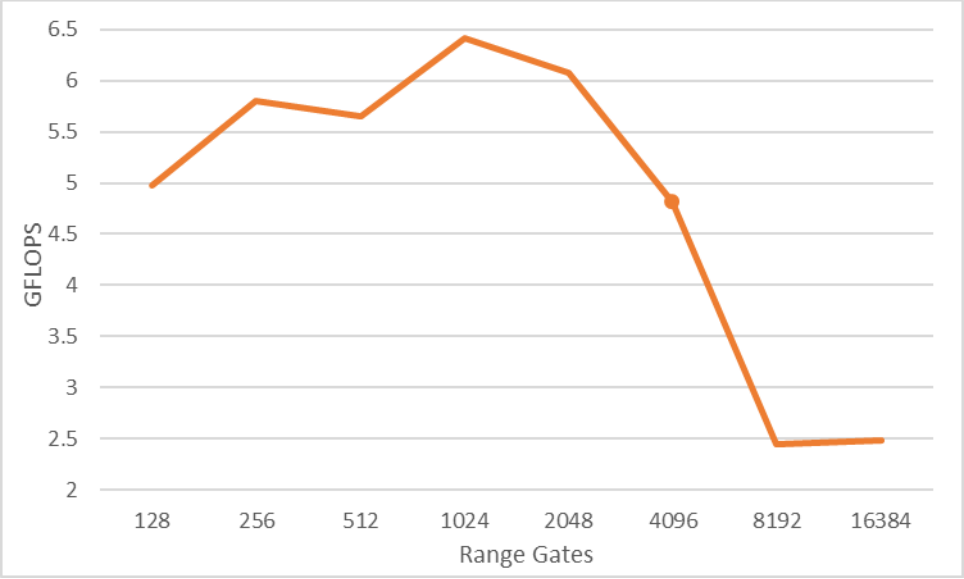


Figure 3.21: FFT performance for different range gate numbers

and W_N^{3n} are arranged to be contiguous. This eliminates the twiddle factors allocation separation within a butterfly. However, this implies that as the loop is traversed from one stage to another, a redundant version of the twiddle factor array is required. Hence the size of the twiddle factor array is increased to $2N$ compared with that the conventional FFT is of size $3N/4$.

The computation throughput of FFT measured on one C66xx core is in Figure 3.21, in which dot represents the maximum number of range gates that the DSP cache can hold. It is evident that the calculation performance would degrade dramatically when the data size is close to or over the cache size.

3.5.2. Weighted Dot Multiplication

Besides FFT, weight dot multiplication is another basic computing algorithm used in the signal processing. Compared with FFT, the weight dot multiplication does not involve the data manipulation in the butterfly network, which means the performance of the vector multiplication is highly depended on how to use the cache efficiently and reduce the computing stall. So, a good strategy for optimizing cache performance is a guarantee of a high throughput computing. There are two levels of cache optimization: application level and procedural level. The application level optimization is a high-level optimization procedure that the designer should make the flow of data continuously poured in/out of the on-chip memory by using DMA. Those on-chip memories, L1/L2 SRAM, are closer to the processing core, working as a buffer. Therefore, the computing stall time is reduced and throughput is increased. Moreover, the cache coherence in the on-chip memory is automatically maintained by the cache controller. This mechanism can increase the computing efficiency, compared with by using external memory as a buffer, in which

programmer should manually issuing L2 cache coherence operations. However, implementing the DMA buffer is time-consuming, for the rapid-prototyping applications, it would be easier to configure L1/L2 as cache and maintain the cache coherence manually.

The next level is the procedure optimization, in which data structures that are accessed by the algorithm are optimized to make use of cache memory efficiently. For the condition that the size of data is larger than the cache and the data would not be reused, the interleaving cache sets can improve the computing efficiency. The interleaving cache, or memory, is to spread the entire data evenly across several memory banks. Normally, this method is used to increase the throughput of memory by avoiding using the same memory bank repeatedly [106]. In the cache optimization, the interleaved cache is used to separate the buffer data into different cache sets. Before introducing interleaved cache, we should note that TI C66x DSP core uses the 2-way associative cache, which means

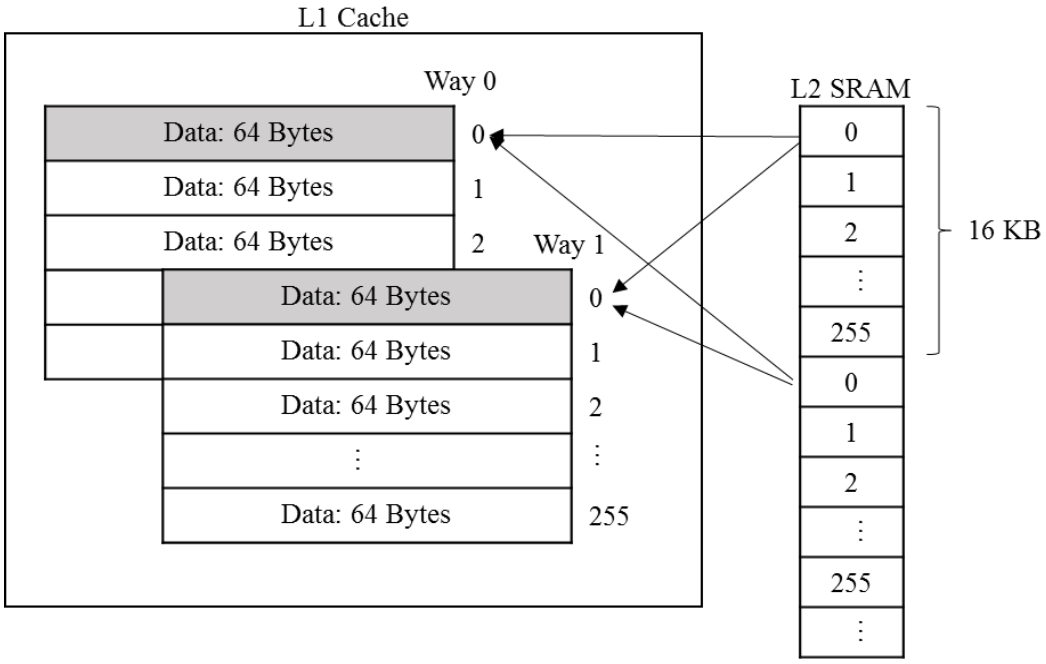


Figure 3.22: L1D cache architecture

```
for (i=0; i<N; i++)
      sum += w[i] * x[i] * h[i];
```

Figure 3.23: Weighted Dot Product

the DSP core have two cache ways to reduce the probability of conflict misses. As shown in Figure 3.22, each cache line contains 64 bytes data, and each set of a 2-way set-associative cache consists of two line from L2 SRAM, one line frame in way 0 and another line frame in way 1. A line in L2 SRAM still maps to one set, but now can be stored in either of the two line frames. The problem of this architecture is that if multiple data being used belong to the same set, the previously cached data would be evicted. For example, Figure 3.23 shows the codes of an N-element weighted-dot-product, in which the size of w, x, and h are associated with the same cache line in L1. So those three vectors cannot be cached at the same time. The solution of this problem is to allocate the data set contiguously in memory and pad arrays as to force an interleaved mapping to cache sets. Figure 3.24 shows the memory layout after first two iterations based on Figure 3.23. The pad reallocates the array h in the next set, thus avoiding the eviction the array w. As a result, all the three arrays can be in the cache.

Another technique used in the procedure level optimization is to split the entire data set and process one subset a time, which is referred as blocking or tilting. This method would increase the computing efficiency when cached data is reused. For example, in the beamforming, the weight vector is multiplied by with the array data, so in Figure 3.23, array x and w are used, and h is omitted. Thus, w is reused each time. In that sense, we can handle the data storage carefully to make sure the weight vector not be evicted before the next subset reuses it. As an example, suppose one DSP core forms 15 beams from 24 channels, and each channel contains 1024 range gates, so w and x are the

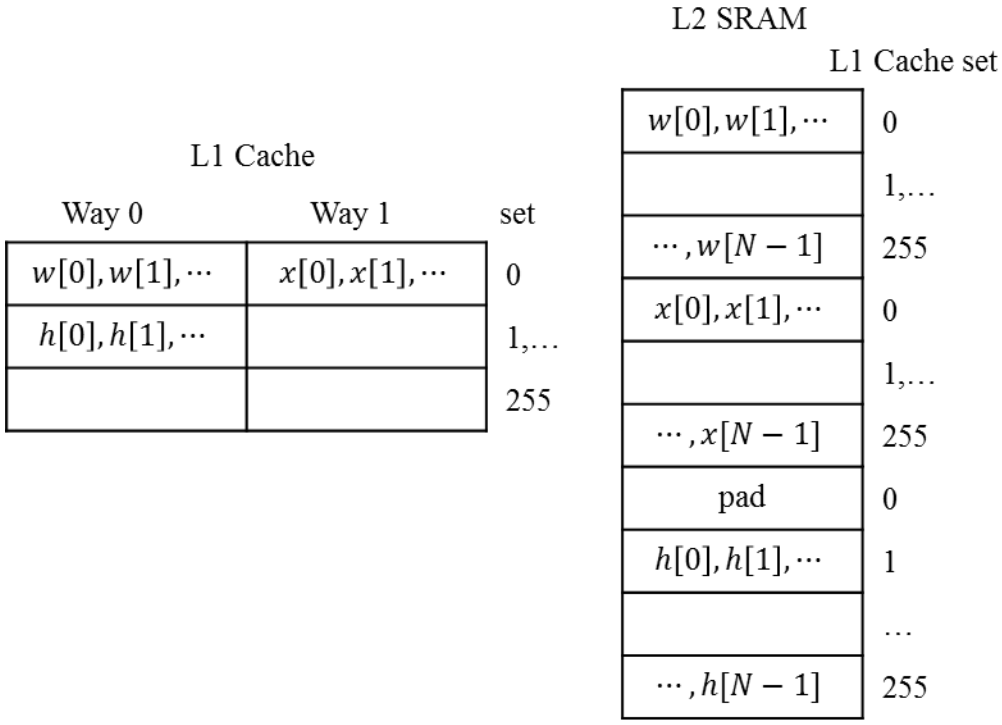


Figure 3.24: Memory Layout after two iterations

matrices of dimensions 24×15 and 24×1024, respectively, which are 3 KB and 192 KB in sizes. As the size of L1D cache is 32 KB, to allow the weight vectors and input matrix fitting into L1D cache, the data from 24 channels should be divided into 16 subsets. So, one large-size beamforming based on 1024 range gates is converted into 16 small size beamforming based on 640 range gates. For example, in Figure 3.25, when channel number equals to 16, if there are no cache misses, four cases should have the same number of GFLOPS. However, for the cases that the numbers of range gates equal to 128 and 256, the beamformer can outperform the cases that range gates are 512 and 1024. This variation is caused by cache miss. The markers in Figure 3.25 represent the maximum number of channels that the DSP cache memory can hold for a specific number of range gates. Before reaching each marker point, the performance improvement of each case is from using larger sizes vectors, which reduces the method-call-overhead.

However, after reaching the marker points, the benefit of using large sizes of the vectors is compromised by the cache misses. Table 3.2 shows the beamforming performance after utilizing the blocking, in which the performance of DSP core remains the same regardless the size of input data.

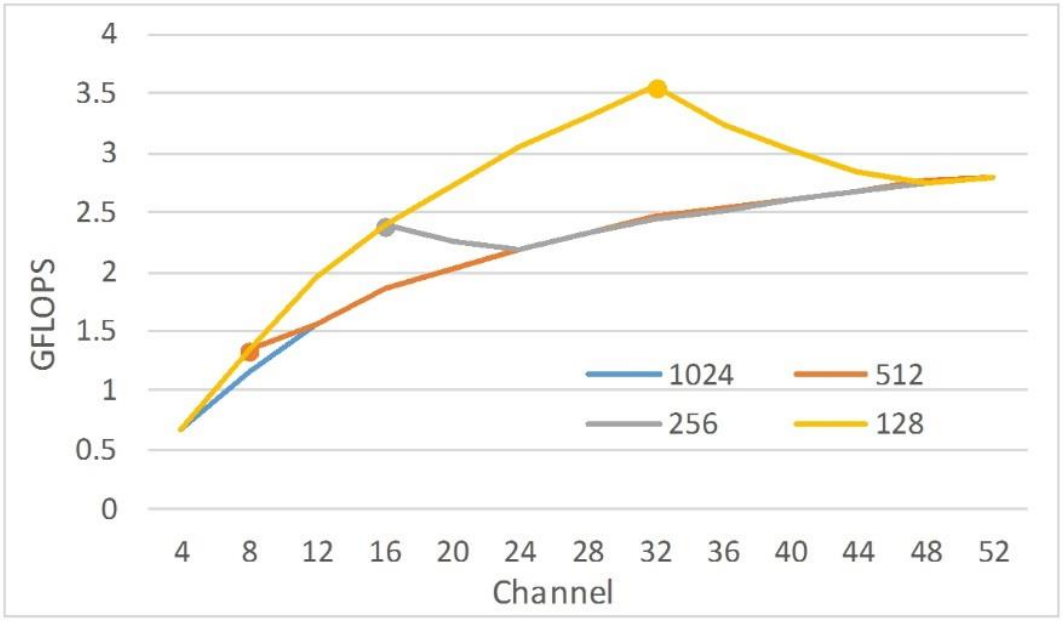


Figure 3.25: Computing performance of a DSP-core versus the number of range gates

Table 3.2: DSP core performance after mitigating cache misses

GFLOPS / Range Gates (Subsets) / Channel	1024 (16)	512 (8)	256 (4)	128 (2)
4	0.67	0.67	0.67	0.67
8	1.34	1.34	1.34	1.33
12	1.97	1.96	1.96	1.95
16	2.42	2.42	2.41	2.40
20	2.81	2.81	2.80	2.78
24	3.15	3.15	3.14	3.12
28	3.45	3.44	3.43	3.40
32	3.71	3.70	3.68	3.66
36	3.92	3.91	3.87	3.82
40	4.10	4.09	4.04	3.96
44	4.25	4.24	4.19	4.08
48	4.39	4.38	4.34	4.23
52	4.49	4.48	4.46	4.35

3.5.3. Data Corner Turn

In PAR, different processing operates on three-dimensional data in multiple stages. For the efficiency reasons, as we mentioned in Section 3.5.2, it is desirable to continuously align the data in the domain where the algorithm works on. Therefore, the alignment of the data needs to be turned from one dimension to another. This realignment is called "corner turn" in radar vernacular. This two-dimensional corner turn operation is equivalent to a matrix transpose in the memory space. An example situation where this might occur would be a Doppler filtering followed by a pulse compression [107]. Pulse compression and Doppler filtering process the data along the range and pulse domain separately, thus the two operations suggest different optimal data layouts. So, the corner turn transforms the layout of the data matrix to preserve data locality in the dimension being operated on. As the data comes in one dimension and is read in another, the amount of data control operations can be quite large and time-consuming. It is not a good choice to make the processing core to handle the corner turn. With the help of Enhance Direct Memory Access (EDMA3) [39] on TI C66x DSP, by pre-defining the procedure of corner turn, EDMA3 can reorganize the data into the desired format independently without interfering the real-time computations in DSP core.

EDMA3 is a co-processor, which can perform data transfers without processor core intervention. There are two components in EDMA3: DMA and Quick DMA (QDMA). DMA is configured to respond to the interrupts from EDMA event, processing core, and peripheral registers. It can be used for synchronizing the peripheral events and processing. For example, Figure 3.26 shows an illustration of front-end transmission plan. After the data are grouped and packed by FPGA, the data would send through the SRIO

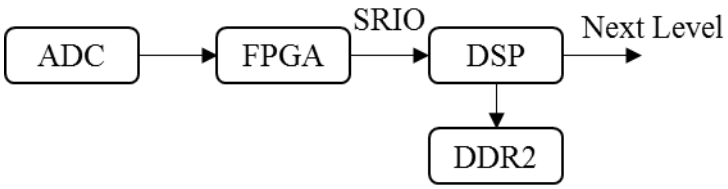


Figure 3.26: An illustration of front-end data transmission strategy

link to DSP and stored at DDR2. When the SRIO transmission is done, a sync event (interrupt) would be generated from SRIO to DMA, and the DMA would copy the data from DDR2 to the on-chip memory buffer independent of the processing core. Once the buffer is full, another interrupt would be sent from DMA to notify that the data is ready for use. Once the current processing is done and data is ready to be sent to next level. The core would trigger the pre-programmed DMA to transmit the data to external DDR2. When the out-of-buffer transmission is done, the DMA will trigger the SRIO peripheral to send the data outside. In this data in-and-out transmission, DSP core and DMA engine work independently, increasing the computing efficiency and data transmission throughput. Compared with DMA, QDMA is used for on-chip memory-to-memory data movement, which is easy to be programmed and triggered.

Table 3.3:Time consumption of corner turn for one beam

Time (μs) Pulses / Range Gate	8	16	32	64	128
1024	21	33	64	126	253
2048	40	66	130	254	502
4096	135	265	513	1011	2028
8192	528	1025	2033	4070	8107

Besides the basic sequential data transfer function, the DMA in TI C66x core offers the advanced index transfer for both source and destination addresses. For example, by properly programmed the registers, DMA can send the data separated for

every N bytes and stored them at the destination address separated for every M bytes, in which M and N could be any value between 0 and 0xFFFF. By using this feature, the data corner turn can be easily accomplished. Table 3.3 shows the performance of data corner turn by using EDMA3 under different conditions.

4. An Example System Implementation

4.1. Architecture Design Considerations

In the previous sections, the computational aspects of the front-end and backend processing algorithm have been explored, and various mapping strategies and the architecture of processing unit have been discussed. A low-cost HPEC system with scalability is now considered as the host platform for a large-scale PAR. Although there are no tight form-factor constraints compared to some applications, such as airborne radar, this platform imposes the requirements of showing the ability of scaled up and upgraded and flexibility of enhancing the signal processing algorithm in the future. Table 4.1 shows the parameters for an example PAR system. Based on those parameters, a complete implementation of the processing chain would be given in the following sections. First and for the most important, the network topology is the critical factor to affect both the

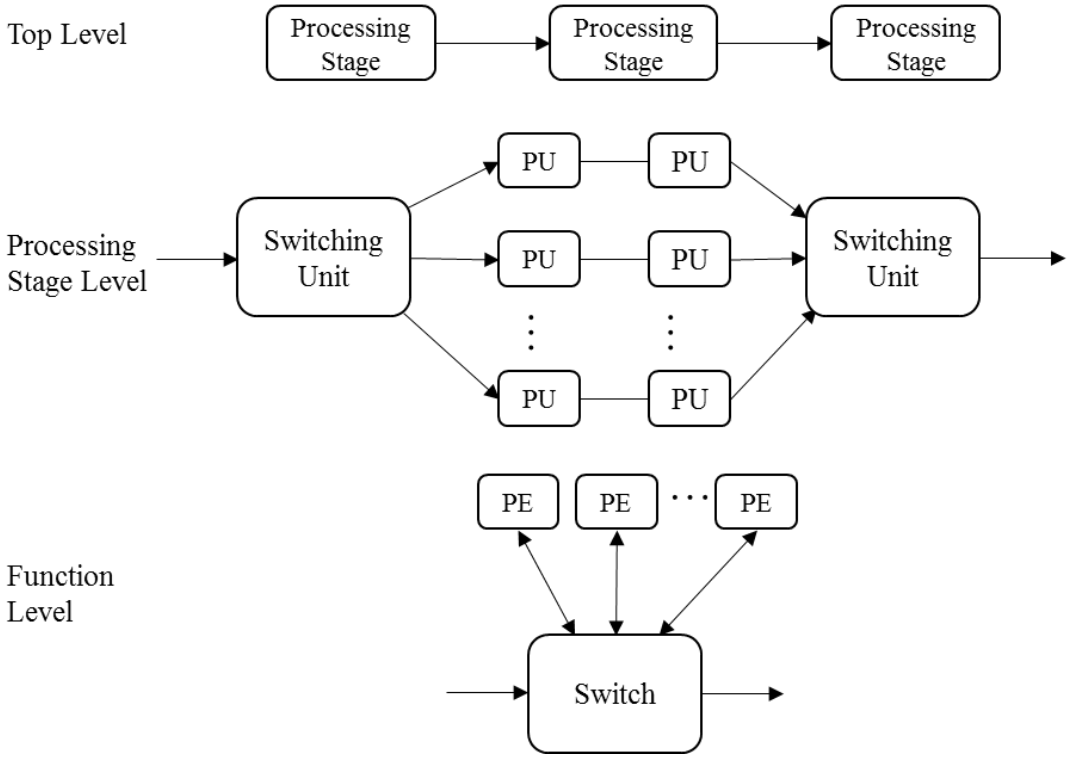


Figure 4.1: System Network Topology

system architecture and the bandwidth of data communication. Figure 4.1 shows the network topology for the various levels in the computing platform. In the top level, the data received from the antenna array form a three-dimension data cube, and each processing stages process one domain's data independently. Therefore, we can use the pipeline parallelism, in which the output of one processing stage is the input of the next. For the hundreds of the channel PAR application, the movement of data is as important as the processing. Table 4.2 gives an estimate of the communication bandwidth between the processing stages for the system based on the parameters from Table 4.1. Note that the time for the data corner turn between each stage should also be considered, which requires the network interface with high bi-directional bandwidth and the flexibility of routing data. To increase the transmission efficiency, in each processing stage, a switching unit is placed either at front or end, of each processing stage to combine or distribute data from previous or to the next stage. The switching unit in this level needs to buffer the data for the high-speed out-of-chassis communication, so it requires the unit has the ability of access large amount of data with low latency. The data in each processing stage would be separated into multiple PUs, and then the results are combined into switching unit. In the function level, the interconnection of multiple PEs via a dynamic switch network is built based on a multiport switch. At this point, the switch in the PU only needs to route the data between each PE or out of function level, so compared with the switching unit in the processing level, the switch in this level would handle smaller size data with more complex data routing requirements.

Table 4.1: Example of PAR system parameters

Parameters	Value
Range gates	4096
Pulses	128
Channels	768
Beams	264
PRI	$1\ ms$
CPI	$128\ ms$

Table 4.2: Communication Data Rate per Stage

Stage	Input (GBytes/s)
Beamforming	12.5
Pulses Compression	4.3
Doppler Filtering	4.3

In a PAR system, different processing unit maintain their own local clocks and those clocks have drifting errors. Even a tiny drift in each clock cycle, it will be magnified to a large error when hundreds clock cycle has passed. Hence, a continuous mechanism for synchronization is needed for the distributed computing system, so that the system operation can be coordinated. Figure 4.2 shows an illustration of typical synchronization method for distributed systems. The NTP has an approximate error in the range between 5 ms to 50 ms [108]. In the local network, numerous software clock synchronization algorithms have been analyzed and evaluated, such as [109], [110], and [111]. Those methods can achieve the accuracy in the range of several ms. By using dedicated

hardware, the software processing delay can be eliminated and the accuracy can be improved in the range of several μs, or to the best by using PTM-1588 and MTCA chassis the accuracy can reach to 50 ns [112]. For the PAR application, it requires a tight synchronization, so the system needs a dedicated hardware to maintain the synchronization by distributing a common clock to the multiple numbers of chassis, and the clock in each chassis can be synchronized based on this common clock source. However, it is always difficult to make the clock reaching each chassis at the same time for the reason of that there may be skew and uncertainly from routing delays along the physical signal wire. To get a better synchronization, the GPS signal is utilized to give a reference signal and another common clock source works as a trigger signal, as mentioned in the Section 3.4.2.

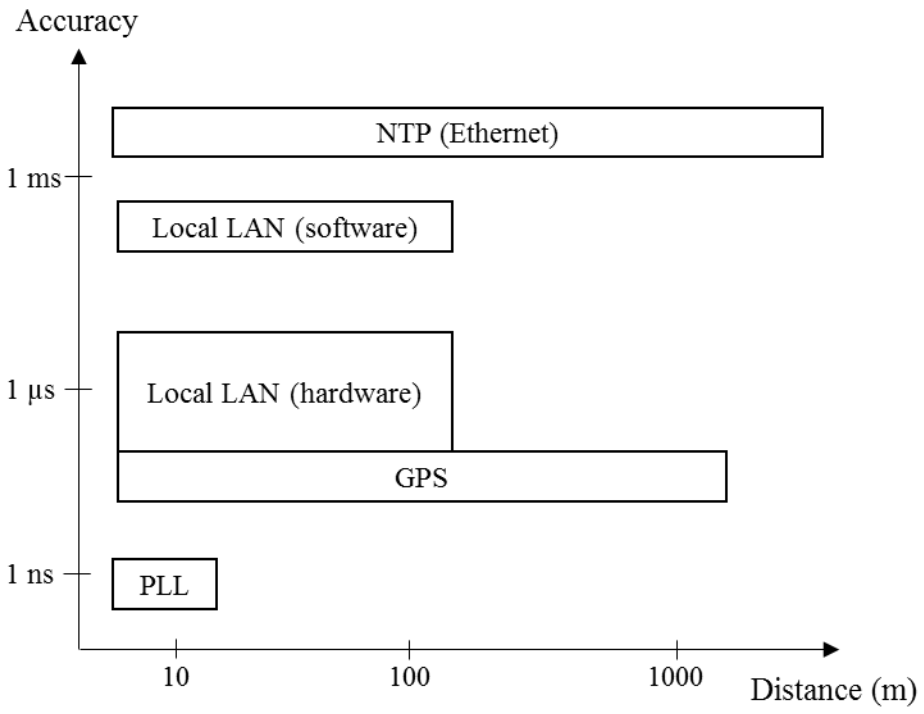


Figure 4.2: Clock Synchronization classification

In a distributed system, given a large number of processing node and chassis, in which each of them has multiple computational, I/O, and network components, failures would be commonplace. Therefore, the control system should monitor the health of each node, identify the failure parts, and quickly recover by using either automatic or out-of-band method. The control link should be implemented independently from the data network to increase the reliability of the system. An example of system level monitoring is the Intelligent Platform Management Interface (IPMI) in the MTCA, which provides the diagnostics information of each AMC (e.g., power supply, fan tray, and inventory information) to the shelf manager. Another important role of the control system is to allow the entire system to be scaled up and become increasingly distributed [113]. Since the fast development of electronic, it is inevitable to scale the system with newer and different vectors hardware, so the control system should be capable of handling the variation in hardware and software introduced by the upgrading the system. An example of this distributed monitoring software is Ganglia [113], which is an open-source project that shows the high levels of robustness and ease of management [114].

4.2. Vendor Selections

Choosing the products from the various vendor is one of the important processes to design the architecture of the system. Many industrial standards, such as MTCA, ATCA, and AMC, are defined the form factors, such as the capability, I/O bandwidth, and processing power, of products in different ways. So by using different vendor products to build a heterogeneous system, the platform can take advantages of those varieties, however, when parallelizing an algorithm in a multi-processor environment, it would be better to choose a homogeneous system for the less programming complexity heterogenous system

[115]. After the researchers decide the products for handling the computing, the system architecture to support those products can be fixed. Thus, a vendor selection is critical for the system architecture and the performance of the entire system.

In this research, we studied the products from three different vendors: TI, Vadtech, and Prodrive. Each vendor has their focus, so the usage of their products may be varied based on the purpose of applications. TI, one of major semiconductor companies [116], produces several multi-core processor lines, in which the most powerful one is the TI 66AK2H12. It has 8 C66x DSP cores and Quad Cortex-A15 cores with multiple types of high-speed I/O [11], and the corresponding evaluation module (EVM), 66AK2H, for under $1,000 [117]. This EVM has the advantage of better price-performance ratio, easy to purchase, and less leading time, compared to other equivalent products on the market. Moreover, the product support from is always reliable from TI then other small companies. Since the EVM is a reference design for the general purpose, the board tends to represent all the features on one board, so it does not optimize the size, power consumption, and performance. On the other hands, the third-party products aim to the market of the high performance and high-reliability applications. So, it has larger computing throughput than TI EVM. For example, AMC-TK2 manufactured by Prodrive is a full-size AMC that combines a Quad ARM Cortex-A15 cores with 24 C66x DSP cores [118], which is three times more processing power than TI EVM. Another advantage of the third-party products is their technology supports are more specific to the area of their customers. As the purpose of this study is to build up a prototype platform to verify the feasibility and functionality of the HPEC for large-scale PAR system, we choose the TI EVM board as our processing node. A systematic process for decision

support in evaluating and ranking various vendor products is still needed for the formal product [119].

4.3. Processing Chain Implementation Details

In Chapter 2 and 3, the focus is on computational complexity and algorithm decomposition for the baseline PAR signal processing. This section will show an example of a system-level large scale PAR computing platform design focusing on the front-end processing, and reveal some of the trade-offs when mapping the algorithms to HPEC system. This processing platform is not specific for a large-scale PAR system, rather, it can be a generalized purpose real-time HPEC processing platform for the multi-channel applications that require high throughputs, such as driver-assistance automotive [120], telecommunications [121], and biomedical imaging [122].

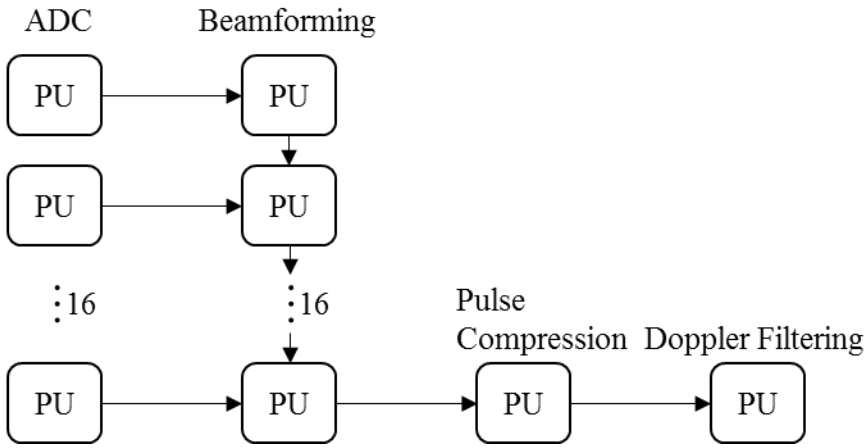


Figure 4.3: An example of front-end processing platform

Based on the parameters listed in Table 4.2, and the PU described in Section 3.3, we proposed a front-end processing platform as shown in Figure 4.3. The entire processing chain works as a pipeline and can be separated into four generic stages: AD conversion, beamforming, pulse compression, and Doppler filtering. In the ADC step,

each PU samples the signal from 48 channels, so in total 16 PUs will collect $48 \times 16 = 768$ channels data. After the data from each pulse is recorded, the receiving PUs would send the data to their counterpart beamforming PUs. In the beamforming, since each beamformer requires the data from all the antennas, the data routing between antennas and beamformer would be complex when the number of channels is large. To mitigate the complexity in data routing, as showing in Equation (2.2) and (2.3), the entire data is divided equally and a portion is assigned to each sub-beamformers, (i.e., computing node), in which the term $\sum_{i=1}^{C}\left(W_{jC+i}^{b}Y_{jC+i}\right)$ is calculated independently. A formed beam is generated by accumulating the results from each sub-beamformers. This method is named as *systolic beamforming* [123]. Based on Equation (2.2) and (2.3), the beamforming can be re-written to Equation (4.1),

$$\sum_{i=1}^{C} W_{mC+i}^{\Theta} Y_{mC+i} = \bigcup_{n=1}^{N}\left(\sum_{i=1}^{C} W_{i}^{(n-1)B+1} Y_{i}\right) \tag{4.1}$$

in which $N = 12$ is the number of PE in a PU, $C = 48$ is the number of channels obtained by each PU, $B = 22$ is the number of beams processed by each PE, Θ are the beam number indicator. In our implementation, the received data from total 768 channels are sent to 16 PUs, in which, as showing in Equation (4.1) each PE calculates the term $\sum_{i=1}^{C} W_{i}^{(\mathrm{n}-1)\mathrm{B}+1} Y_{i}$ forming number of B partial beams in parallel. After all the PEs finish the computing, the first PU starts to pass the result to its downstream neighbor, in which the received data are summed with its own and the results are send downstream. In turn, after the last PU combined the results from all the upstream PU, the entire number of 264 beams based on 768 channels are formed. In other words, each PE converts the data from 48 channels into partial of 22 beams. So, the output of one PU is the data matrix with 48

channels and 264 beams data, and this matrix would be given to the next PU in the systolic beamforming. In the end, the last PU would combine all the results from previous PU and form the entire 264 beams.

Following beamforming, the next step is the pulse compression, in which there are 12 PEs within one PU to process the 264 beams. So, each PE would do the pulse compression for 22 beams. After all the PE finish the computing, one PE would combine all the results from others through the backplane by using Serial RapidIO, corner turn the data along the pulse domain, and send them through Hyperlink cable to the Doppler filtering stage. Same as pulse compression PU, 12 PEs will perform Doppler filtering for 128 pulses. From the prospect of task parallelism, in the top-level, the processing chain works in the pipeline. In the lower-level, there are two parallelisms-the beamforming is systolic parallelism and the rest of the two processing stages work in the round-robin parallelism. In each PU, the parallelism is round-robin.

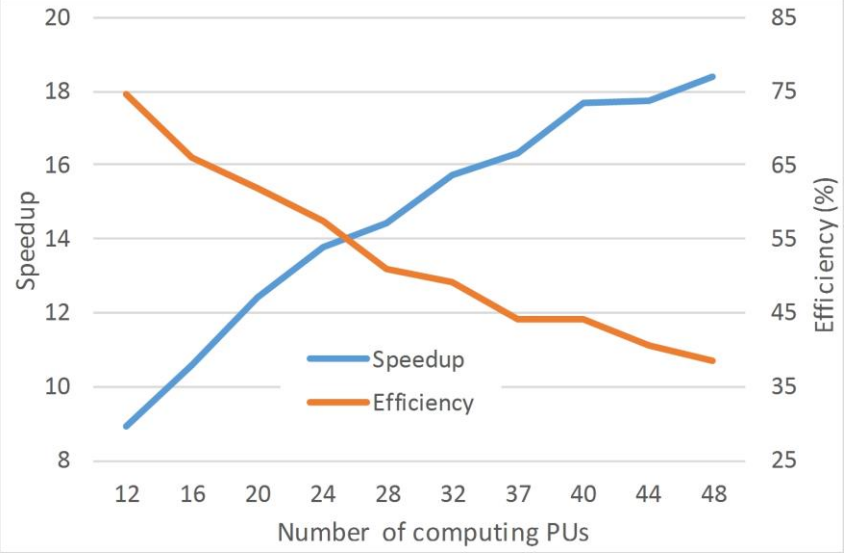


Figure 4.4: Speedup and efficiency of beamforming implementation

In Section 3.1, parallel speedup and parallel efficiency are introduced, which are two important metrics. Figure 4.4 shows this effect for a parallel implementation of the

beamforming, in which the speedup grows with the number of PUs, but the efficiency is degradation due to the reason of method call overhead. As for this reason, we need to seek a balance between the performance and effectiveness based on the system requirements. According to Figure 4.4, an optimal choice, for example, when the number of PU equals to 28, allows the system to achieve a good speedup while maintaining a reasonable level of efficiency. In our proposed system, we want to make full use of our equipment and achieve high efficiency, so the number of computing PU in the beamforming is 16.

4.4. Benchmark Results

In previous sections, a set of single-processor kernel benchmarks, such as FFT, weighted dot production, and data corner turn, have been given. This section gives a quantitative evaluation multiprocessor application benchmark. Figure 4.5 shows the time scheduling of the radar processing chain mentioned in Section 4.3 and parameters listed in Table 4.1.

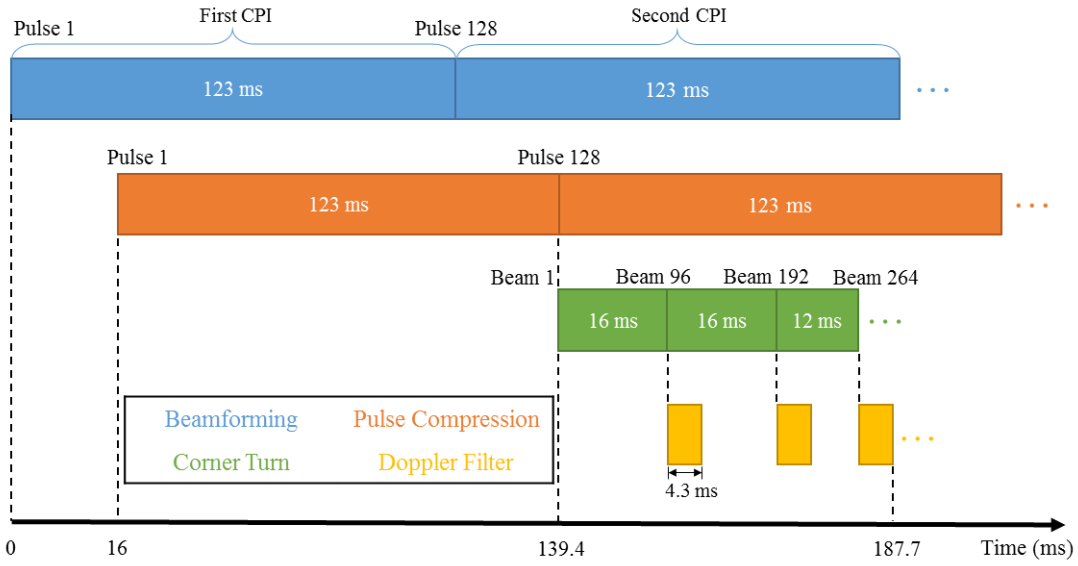


Figure 4.5: Real-time system timeline for the example backend system

The numbers of PU and PE are chosen as an example, which can be changed based on

the application requirements. This scheduling is a rigorous and realistic timeline including all the impacts of SRIO communication and memory access latency, and has been verified by real-time hardware running tests. After the first data sample by ADC, the data cube from the first pulse is formed and send to the beamforming stage, the parallel beamforming processors use $123\ ms$ to generate 264 beams for the each of 128 pulses in one CPI, in which $16\ ms$ is needed until the first pulse beamforming is done and reached to the pulse compression stage. In the pulse compression stage, the processing platform needs $123\ ms$ to do the pulse compression for the entire 128 pulses in one CPI. After the pulse compression, the data corner turn is required before Doppler filtering. As we mentioned before, EDMA on the DSP would do the data transformation independently from processing core, thus, the data corner turn is conducted in the pulse compression stage. In $16\ ms$, the first 96 beams from 128 pulses will be realigned in the CPI domain and sent out to the Doppler filter. In the end, the data cube would be processed through the Doppler filtering stage. In total, there are 192, 12, and 12 TI C6678 DSP cores involved for the beamforming, pulse compression, and Doppler filtering, respectively. And for each processing function, it achieves 6880 GFLOPS, 370 GFLOPS, and 140 GFLOPS real-time performance, respectively. The overall latency, depth of pipeline, for the backend system is 1.5 CPI or $187.7\ ms$.

4.5. Comparison with OpenCL

The previous section summarizes the approach of "manual task division and parallelization." Another option is using standard and automatic parallelization solutions. For example, *OpenCL* is a standard for parallel computing on heterogeneous devices. The

standard requires a host to dispatch tasks, or kernels, to devices which perform the computation. In the single cluster, one master processor is running the host system and dispatch the tasks to other slave processors. For systems with more than one cluster, OpenCL can dispatch different kernels to each cluster. When the kernel is dispatched, arrays must be copied from host memory to device memory. This communication adds significant *overhead* to computation time that increases linearly with buffer size.

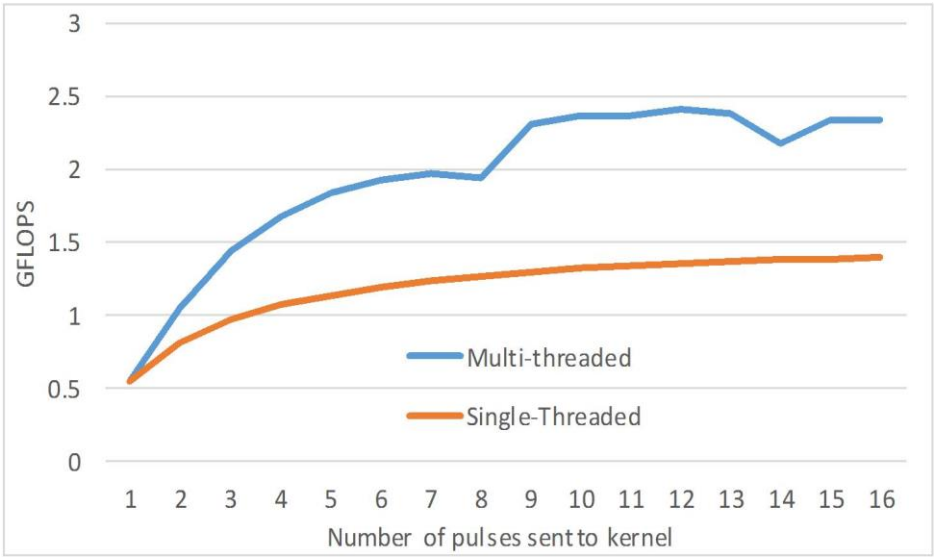


Figure 4.6: Beamforming kernel performance using Open CL

To leverage the performance of OpenCL, the TI 66AK2H14 is loaded with an embedded Linux kernel that contains the OpenCL drivers, in which the ARM core will dispatch the computing task to each DSP core. In the beamforming, the processing of each beam is allocated to its parallel processing thread for each DSP core. Figure 4.6 shows that as the number of beams sent to the kernel increases, the time it takes to process an individual beam decreases. Because of task dispatching communication overhead, the performance of the kernel increases logarithmically.

Comparing the performance of OpenCL implementation to the manually optimized scheme, the overhead of standard scheme can be seen more clearly in Figure 4.7. On

average, using OpenCL/MP results in a 33% average performance penalty in beamforming with a maximum penalty of 44.3% when performing beamforming from 48 channels.

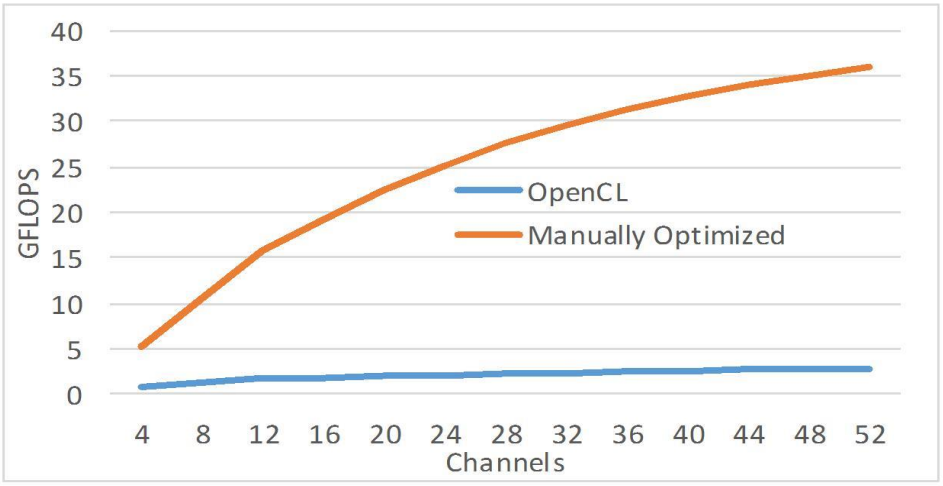


Figure 4.7: Comparing OpenCL performance to manually optimized code for beamforming

In the pulse compression, the performance of OpenCL implementation is shown in Figure 4.8. The comparison between OpenCL with the manually optimized codes is shown in Figure 4.9. As discussed previously in Section 3.5.1, FFT and IFFT require highly non-linear memory accesses. Thus it is essential to optimize the data fetching pattern manually. However, in the OpenCL, it does not provide the flexibility for a programmer to adjust the memory patch pattern, so the latency due to the non-linear accesses are compounded which results in severely degraded performance.

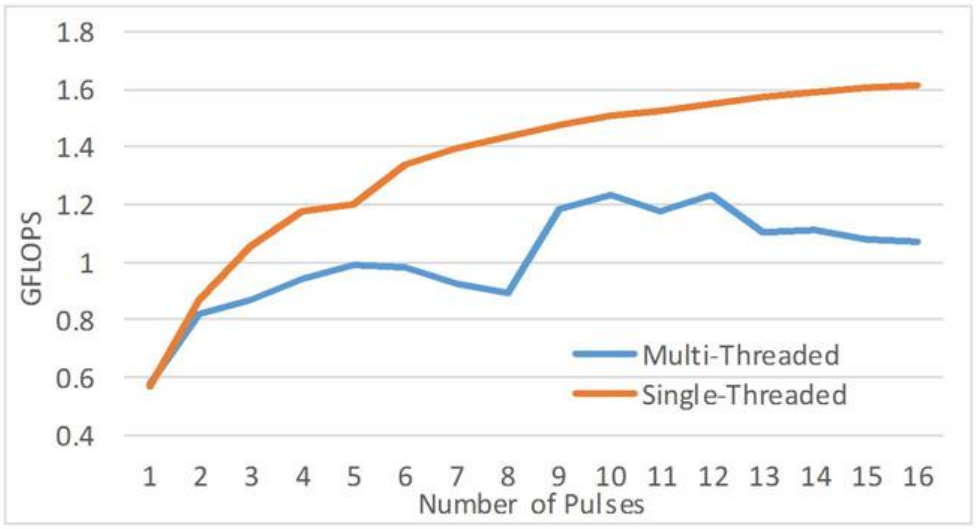


Figure 4.8: Pulse compression performance using OpenCL (8192 range gates)

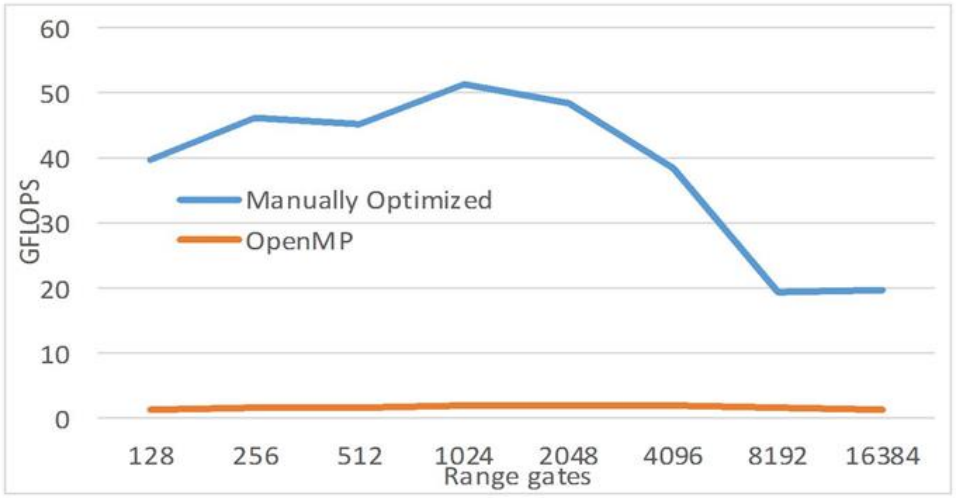


Figure 4.9: Comparing OpenCL performance to manually optimized code for pulse compression

4.6. Summary

In this chapter, considerations of the architecture of the PAR processing platforms are illustrated. In general, the architecture should reflect the data topology in each processing stage, provide enough communication bandwidth for high throughput computing, and maintain an accurate synchronization among each processing node. When designing platform for PAR based on COTS technologies, the choice of products from various

vendors requires a process of evaluating and ranking software and hardware. First, we need to evaluate the cost, time to market, and features that each COTS product has. Then, further analysis would estimate how much the long-term maintenance costs would be.

We present a development model of an efficient and scalable backend system for digital PAR based on Field-Programmable-RF channels, DSP core, and SRIO backplane. The architecture of model allows for real-time, synchronized and data-parallel radar signal processing. Moreover, the system is modularized for scalability and flexibility. Each PE in the system has a proper granularity to maintain a good balance between computation load and communication overheads.

Even for the *basic* radar processing operations studied in this work, teras-scale floating point operations are required in the MPAR/SENSR type backend system. For such requirements, using programmable software DSP that can be attuned to the processing assignment in parallel would be a good solution. The computational aspects of a 7400 GFLOPS throughput phased array backend system has been presented to *illustrate* the analysis of the basic radar processing tasks and the method of mapping those tasks to an MTCA chassis and DSP hardware. In our implementation of PAR backend system, the form-factor can be changed based on requirements of various systems. By changing the number of PUs, the total capacity of the system can be easily scaled. By changing the number of inputs for each PE, we can adjust the throughput performance of a PU. A carefully customized design of different processing stages in DSP core also helps to achieve the optimal performance regarding latency and efficiency. When we parallelize a candidate algorithm, there are two steps in the design process. First, the algorithm is decomposed into small components. Next, each algorithm component is assigned to

different processors for parallel execution. In the parallel computing, the communication overhead among parallel computing nodes is a key impact on the *parallel efficiency* of the system. Within each parallel processor, dividing the entire data cube into small subsets to avoid cache miss is also necessary when the size of input data is larger than the cache size of processors. For data communication links, the SRIO, HyperLink, and EDMA3 handle the data traffic between and/or within each DSP. By using SRIO, the data traffic among DSPs can be switched through the SRIO fabric controlled by an MCH of the MTCA chassis, which is more flexible than PCIe and efficient than Gigabit Ethernet. A unique advantage of our proposed method is utilizing EDMA3 and Ping-pong buffer mechanism, which helps the system to overlap the communication time with computing time and reduce the processing latency. OpenCL is a framework to control the parallelism in high level, in which the master kernel assigns the tasks to each slave kernels. Compared with the "bare-bone" method we developed, OpenCL is platform-independent and enables heterogeneous multicore software development, which leads to the drawback of less flexibility and efficiency to specific hardware.

5. Summary and Future Plans

In the previous chapters, the fundamental radar signal processing algorithms have been introduced, and the computing aspects of a large-scale PAR have been presented to illustrate the analysis and mapping of challenging algorithms onto computing devices. Moreover, the difficulty of calibrating RF system in PAR was focused on as a key design consideration. Indeed, the traditional antenna calibration procedures, such as near-field and far-field calibration, are the dominant methods. For many circumstances, however, the traditional calibration method is limited by the surrounding environment and the complexity of procedures. To make the calibration practical and easy to be performed in the field, we have proposed the EM algorithm, which calculates the probabilistic values between measurement results and ground truth values. In particular, if clutter is in the field-of-vew, the traditional calibration method is often too difficult and inaccurate to implement, while the EM method calibration can remain tolerant toward these clutters. On the other hand, the EM method can be performed while the radar is operating. This feature is especially important for which the calibration is expected to conduct from time to time as the parameters from outside environment, such as temperature and humidity, are changing with time.

Several advanced processing algorithms have been introduced in previous chapters, however, due to the time constraint, most of those advance signal processing algorithms are not implemented in hardware. The future work involves reformulating existing MATLAB codes so that they are suitable for HPEC processor architectures such as DSPs. For future algorithm development, the future work would investigate the identification and mitigation of ground clutters. The strong echoes from the ground clutter

contaminate the return signal and mask the weak weather signals. Currently, there are three mitigation methods: enhanced radar system design [124], clutter filtering [125], and post-processing in the backend [126]. By carefully planning the location of the radar system and selecting proper wavelengths, the clutter return can be reduced, but these techniques are limited by other factors. The second option is to apply a notch filter to cancel zero frequency signals in the Doppler spectrum. However, such method would fail for weather echo with zero Doppler frequency. Another option is based on postprocessing, which encompasses the traditional method by integrating radar moments data and their spatial texture, by using pulse-to-pulse cancellation, and by applying mathematical analysis and fuzzy logic synthesis to identify the clutter, as recent developments [127].

Scalability is another issue discussed in previous chapters. For the reasons that MTCA is a modular design that subdivides the system into smaller computing parts, so accommodating scalability in MTCA would be easy. In the software aspect, using DSP on the MTCA platform could facilitate software re-use during upgrading, compared with hardware-coded devices, such as FPGA. The nature of beamforming, pulse compression, and Doppler filtering algorithm is to perform calculations on each slice of data from the different dimensions of the data cube. This feature makes the computing on each slice independent, so it eases the difficulty of parallel partitioning in software. Moreover, when the design of the algorithm for each slice is changed in a way that increases its complexity, either spare computing unit can be exploited, or more computing unit can be added to the system. Since the computing load is equally divided in the current system, in the future, as more advanced chips are added into the system, a dynamic task scheduling and

assigning mechanism can be implemented, which allows the system assign the computing load for each node based on their capabilities.

5.1. Considerations of Future Architecture Design

When designing the backend computing platform for PAR application, many factors such as the communication scheduling patterns among processes, the interoperability between different types of process, the fault tolerance of the system, the scalability of the system, and cost-effectiveness, are needed to be considered. To a great extent, these factors depend on the performance of the entire system. Typically, the PAR system would employ the parallel programming model as a multiple-program multiple-data model, which requires the processors and associated peripherals have a robust and high throughput communication network. Based on the requirement of the radar application, the network hardware should be fault tolerant, which allows for the failed parts to be replaced while the system is still operational with little or no performance degradation. The problem of fault tolerance can be solved by making the system redundant both in hardware and software level. So, the capability of the scalability in the PAR backend not only implies that the system offers the potential for computing power growth but also ensures the system is high-availability. A cost-effective approach to design the backend would include usage of COTS technologies for both hardware and software. Although using COTS components can increase integration complexity has more risk in reliability compared with custom-designed products, the performance improvement by bringing state-of-the-art processor technology and software into the system would overcome these deficiencies.

5.1.1. System fault-tolerance

The role of fault tolerance is to make the system to tolerate the faults, maintain the critical operations, and recover from the failures. When the fault happens in hardware, the best method to keep the system running is to have the redundancy. There are three forms of redundancy: information redundancy, physical redundancy, and time redundancy [128]. The information redundancy is to use the extra information to allow fault detection, fault masking, or possibly fault tolerance. Usually, the information redundancy is to prevent the transmission error over a long distance or a noisy channel. For example, the checksum is a widely used method for purposes of error detection. Because of the high throughput in the PAR backend, even a low transmission error rate would bring many fault bits and cause the system to stall. When implementing the information redundancy, extra hardware or software computing power is usually required. Thus, selecting a proper coding technique would be necessary.

With the physical redundancy, one or more standby sparing hardware or software can bring a system back to full operation once the fault is detected. As the task switching between fault hardware and spares cannot be seamless, this time discontinuity may disrupt the system. To minimize the task switching time, a system can set up the hot standby sparing operating along with other online modules and prepare to take over at any time [128]. Opposed to hot standby sparing, a cold standby sparing is used when the application is not time sensitive, so the system has enough time to power up and synchronize the spare with other modules when the fault is detected. Compared with hot spares, the cold sparing can be power friendly and used in the condition that the power consumption is limited [129]. Figure 5.1 [128] shows an example of the triple-duplex

physical redundancy, which can perform error detection, error location, and system recovery. In this example, six modules are separated into three groups, forming up a triple modular redundancy system. The system output is determined by the majority vote from three groups. If one of the modules becomes faulty, the two-remaining can mask the error. In each group, it contains the combination of one online module and its duplication. The use of the duplication allows the faulty module be removed from the voting arrangement without interfering the system operations.

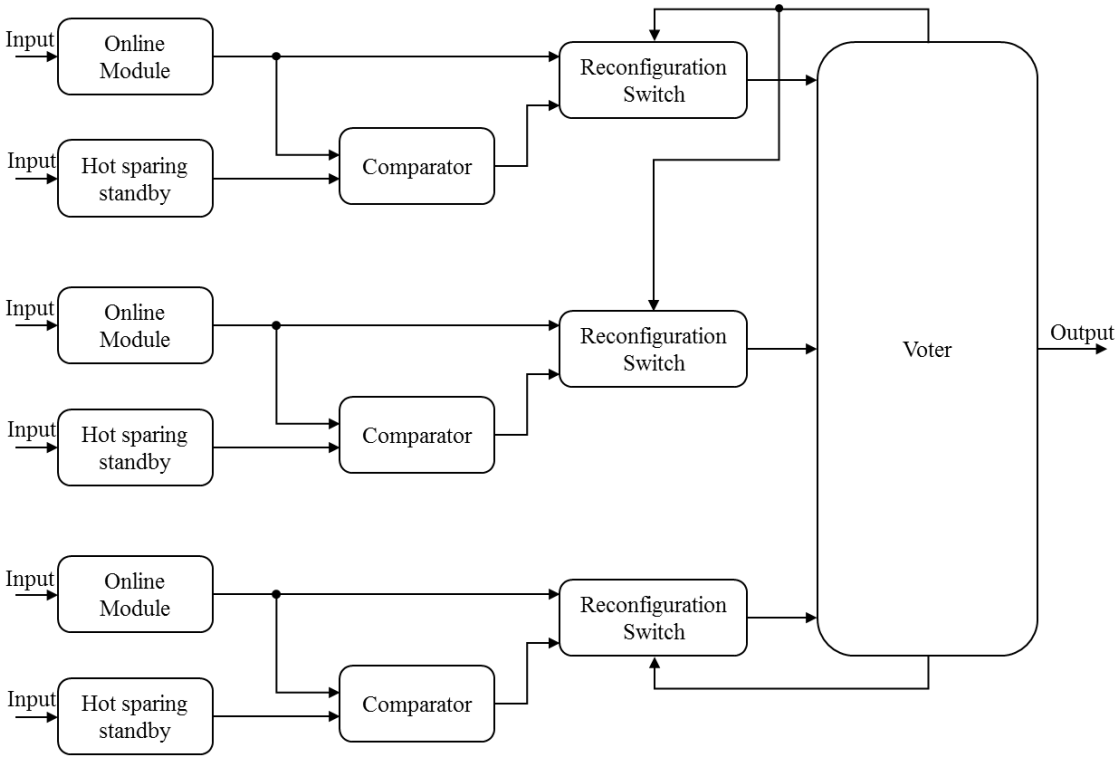


Figure 5.1: The triple-duplex approach to redundancy

In some applications, when the time constraint is not tight, errors can be detected by repeating the computation or transmission and comparing those results. If the discrepancy exists, the computation or transmission can be performed again to see if the conflict remains or disappear. This method is termed as time redundancy. Compared with information and physical redundancy, the time redundancy does not require extra

hardware to detect the fault. Thus the cost of implementing time redundancy is lower than the other two methods. However, the biggest problem of using the time redundancy is that for some applications it is unable to assure the system has the same data to be processed. Besides that, when there is a hardware error happened the time redundancy can detect the error neither. Thus, the time redundancy suits for the processes that the faults are transient or intermittent.

5.1.2. Scalability

The scalability of the PAR backend system allows handling additional of data from increasing number of channels or the computing burden from more complex signal processing algorithm by adding more hardware without suffering a noticeable increase in administrative complexity [130]. A good scalability indicates that the size of a problem can be efficiently extended to the increasing numbers of parallel processing elements. In the real world, as we expand the number of the computing elements, the cost of communication and synchronization among each element would increase, thus reducing the efficiency of parallel programs. Also, adding more hardware to the system would temper the reliability. Thus, at some point, when the number of nodes gets too large, the program cannot perform up to expectations.

As the communication is always the potential bottleneck for scalability of a parallel program, it is necessary for the system architecture represents the pattern of the data flow in the signal processing. For example, in the frontend processing, three fundamental processing tasks, beamforming, pulse compression, and Doppler processing, are conducted in the three-dimension data cube separately as shown in Figure 5.2, in which the data are gathered and separated multiple times. Based on this flow pattern, the

front-end architecture needs to be designed in a combination model of distributed memory and pipeline.

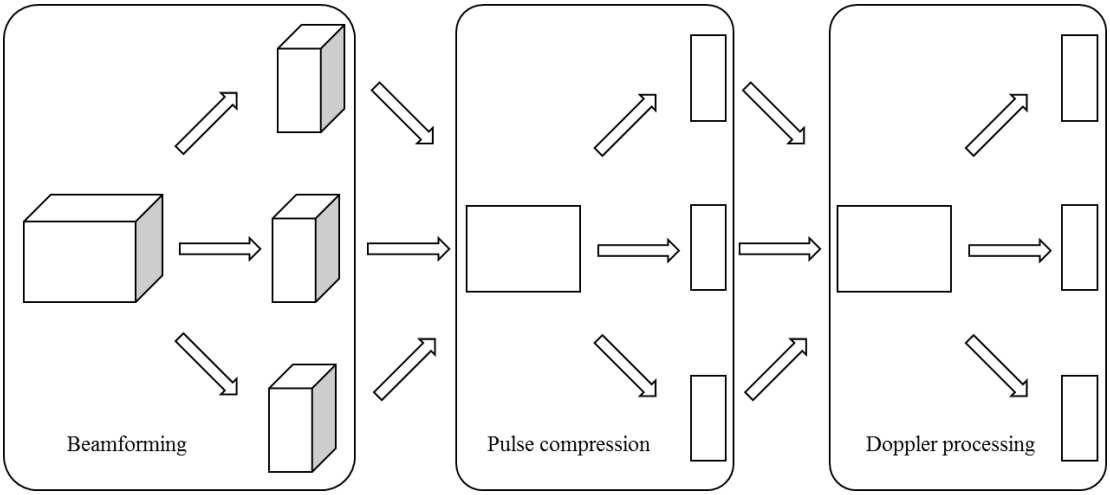


Figure 5.2: Data flowing graph in PAR frontend

Besides the data communication pattern, the synchronizing issue among the computing module should also be considered. As the number of computing node increased, the synchronization among each node becomes a matter that the latency of each synchronization trigger would vary due to physical distance and the hardware variations among modules. If the expansion of the computing node is within one chassis, it is easy to achieve synchronization by using a global clock source. However, when the system requires more chassis to solve additional processing tasks, the synchronization among each chassis requires extra synchronization hardware, such as GPS based phasor measurement units [131] and NTP as mentioned in Section 3.4.2.

5.1.3. Cost

When designing a PAR system, the developers need to balance various competing objects: development cost, production cost, and time to the market. During the development phase, researchers would determine the radar performance objectives based on the goal of the

application and evaluate the candidate algorithms. In the development period, using the reconfigurable computing device would provide flexibility than a dedicated computing device. In the production phase, since the algorithm suite and system parameters are fixed, a device with more computing power but with less or no flexibility to reconfigure can be used. Besides the development time, the time to the market also depends on the difficulty of changing the hardware structure of the development platform to the final product. The changes would be either larger system scale or more specific hardware. Ideally, it would be better to design a scalable system by using the easy-to-program device, such as DSP. There are two advantages of using DSP devices. First, as shown in Figure 1.3, the easier the device to be programmed, the more cost-efficient it is. Second, it can save time and cost to change the software and algorithm once they need to be upgraded. As the computing efficiency of the easy-to-program device is lower than the dedicated device, seeking the balance among the usage of different chips would be important, and in many cases, a hybrid system consisting of the dedicated and flexible programming devices would provide low expense and reconfigurability.

5.2. Future Works

While this dissertation has demonstrated a DSP based high-performance embedded PAR backend system, there are many remaining challenges. Future work includes further implementing the dynamic task and communication scheduling (hot-swapped), cognitive radar (knowledge-based computation), introducing GPU for solving finer-grained parallelism, and optimally scheduling communication and resources. For example, the hot-swapping feature requires the platform reschedule the tasks that had been assigned to the removed computing node to other nodes. When the new nodes are plugged in, the

platform should be able to schedule each task to the computational resource based on the current usage [132]. The hot-swapping feature can be implemented by optimally scheduling the communication and resources in the system. In the radar backend, tasks often have dynamic behaviors caused by the changing external conditions. For example, the number of tasks for target tracking is associated with the number of targets; a good dynamic runtime scheduler can assign the task to the available computing resources based on the system status. Another important future work is to incorporate knowledge base, which incorporates higher-level computations results into front-end sensor processing. The knowledge-based radar uses the prior knowledge of the environment, such as the location of roads, terrain, and types of ground, to perform 'intelligent' processing that avoids invalid assumptions about the environment [133]. Another example is cognitive radar [19], in which the optimal detection threshold based on the measured data is determined by using machine learning technique. The remainder of this section will give a discussion of the outline of the future work for the abovementioned research topics.

5.2.1. Optimal task and communication scheduling

The emergence of radar sensor network [129] and the blurring boundary between the front-end sensing and the backend detection system is both extending the need for the high-performance parallel computation. In the parallel computing platform, there are two views to improve the performance [134]. One is to develop and integrate more advanced hardware and software; the other focuses on the issue of scheduling. In this dissertation, the first method has been discussed and analyzed. So, in the future works, we should concentrate on developing an appropriate task scheduler for PAR backend. The scheduler is responsible for optimally scheduling the tasks to the available computation resources

across the entire networks. The tasks scheduling algorithms can be categorized as either *online* (dynamic) or *offline* (static) scheduling [135]. The offline scheduling has the complete knowledge of all the tasks before the scheduler begins planning their execution schedule. With the online scheduling, tasks arrive at the time that some other tasks are already running, where the scheduler must place the new task around the currently running tasks [136]. In the PAR system, especially in the front-end, the amount and types of running tasks are predictable so that the task scheduling can be guaranteed before execution. Nevertheless, online approaches do have a significant role in the tasks that the predicting is impossible, such as target tracking and super-resolution DOA estimation. Thus, for PAR system, a scheduler may, in essence, be offline but incorporate online scheduling that allows dynamic tasks to continue executing.

In a parallel computing system, the communication delays are significant and non-deterministic, so it gives the difficulties to calculate the useful worst-case delay times. Many algorithms have been proposed to represent the network and end-point contention [137, 138]. However, most of those algorithms employ idealized models of the target parallel system and a fully connected network. Future works need to emphasis on the monitoring the contention for communication resources in a real parallel system. Besides the communication issues, it has been proved that the problem of finding an optimal schedule for a set of tasks is NP-hard [139]. It is, therefore, necessary to plan ways of simplifying the problem and algorithms.

5.2.2. Cognitive Radar

In the age of big data and machine learning, researchers start to use the computers to explore the hidden information and linking information among the data. Following this

trend, researchers start to make the radar backend system adaptively to calculate the transmitter parameters based on the requests from different usage, and the backend signal processing is not only based on the current input but also use previously determined knowledge [19-21]. This way of thinking leads us to the block diagram of Figure 5.3, which depicts the processing cycle of a cognitive radar. A key step in the cognitive radar is to analyze the detection area and build up a knowledge database containing environment information, such as the characterization of radar clutter and the type of targets (continuous or sparse). Since the optimal transmitting waveform is task-dependent [140], based on the interests of different task and the knowledge of the operating environment, the waveform of the transmitter can be optimized to increase the SNR and enhance decision-making performance with defined hypotheses [140, 141].

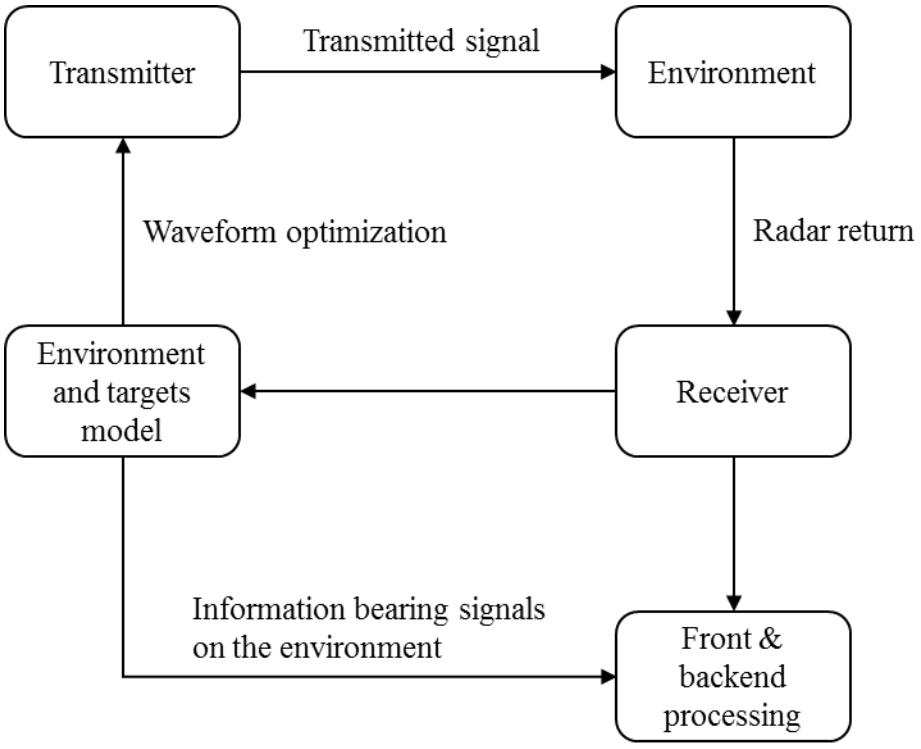


Figure 5.3: Processing cycle of a cognitive radar

Besides updating the transmitting waveform adaptively, other processing parameters can also be adaptively determined, such as the detection threshold in the CFAR [19], bearing angle in the STAP processing [142], and the sensing matrix in compressive sensing [141]. Those dynamic processing techniques are suitable for multifunctional radars, which perform various types of observation on a single platform. Thus, in the future, a multifunctional radar with cognitive sensing capabilities would afford unprecedented levels of intelligence.

6. References

[1] L. Martin, "Lockheed martin space fence radar prototype tracking orbiting objects," *Press Release,* vol. 8, 2012.

[2] J. Herd *et al.*, "Multifunction Phased Array Radar (MPAR) for aircraft and weather surveillance," in *2010 IEEE Radar Conference*, 2010, pp. 945-948.

[3] W. Benner, G. Torok, N. Gordner-Kalani, M. Batista-Carver, and T. Lee, "MPAR program overview and status," 2007.

[4] G. N. Thomas Gryta, "FCC Raises $44.9 Billion in U.S. Wireless Spectrum Sale," in *The Wall Street Journal*, ed, 2015.

[5] J. C. Mark Weber, James Flavin, Jeffrey Herd, Michael Vai, "Muti-function Phased Array Radar for U.S. Civil-Sector Surveillance Needs," presented at the AMS Conference, Albuquerque, NM, 2005.

[6] D. Conway *et al.*, "On the development of a tileable LRU for the nextgen surveillance and weather radar capability program," in *2013 IEEE International Symposium on Phased Array Systems and Technology*, 2013, pp. 490-493.

[7] D. Martinez, T. Moeller, and K. Teitelbaum, "Application of Reconfigurable Computing to a High Performance Front-End Radar Signal Processor," (in English), *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 28, no. 1-2, pp. 63-83, 2001/05/01 2001.

[8] L. D. Stone, R. L. Streit, T. L. Corwin, and K. L. Bell, *Bayesian Multiple Target Tracking*, Second ed. Norwood, MA: Artech House, 2013.

[9] D. R. Martinez, R. A. Bond, and M. M. Vai, *High performance embedded computing handbook : a systems perspective*. Boca Raton: Boca Raton : CRC Press, 2008.

[10] A. Grama, *Introduction to parallel computing*, 2nd ed.. ed. Harlow, England ; New York: Harlow, England ; New York : Addison-Wesley, 2003.

[11] T. Instruments, "Multicore DSP+ARM KeyStone II System-on-Chip (SoC)," 2013.

[12] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1-70, 2008.

[13] A. Katz, "Linearization: Reducing distortion in power amplifiers," *IEEE microwave magazine*, vol. 2, no. 4, pp. 37-49, 2001.

[14] S. Boumaiza, J. Li, M. Jaidane-Saidane, and F. M. Ghannouchi, "Adaptive digital/RF predistortion using a nonuniform LUT indexing function with built-in dependence on the amplifier nonlinearity," *IEEE Transactions on Microwave Theory and Techniques,* vol. 52, no. 12, pp. 2670-2677, 2004.

[15] J. L. Dawson and T. H. Lee, "Automatic phase alignment for a fully integrated Cartesian feedback power amplifier system," *IEEE Journal of Solid-State Circuits,* vol. 38, no. 12, pp. 2269-2279, 2003.

[16] J. S. Herd and M. D. Conway, "The Evolution to Modern Phased Array Architectures," *Proceedings of the IEEE,* vol. 104, no. 3, pp. 519-529, 2016.

[17] F. Kong, Y. R. Zhang, J. Cai, and R. D. Palmer, "Real-time radar signal processing using GPGPU (general-purpose graphic processing unit)," 2016, vol. 9829, pp. 982914-982914-7.

[18] J. L. Payne, N. A. Sinnott-Armstrong, and J. H. Moore, "Exploiting Graphics Processing Units for Computational Biology and Bioinformatics," *Interdisciplinary sciences, computational life sciences,* vol. 2, no. 3, pp. 213-220, 07/25 2010.

[19] J. Metcalf, S. D. Blunt, and B. Himed, "A machine learning approach to cognitive radar detection," in *2015 IEEE Radar Conference (RadarCon)*, 2015, pp. 1405-1411.

[20] S. Haykin, "Cognitive radar: a way of the future," *IEEE Signal Processing Magazine*, vol. 23, no. 1, pp. 30-40, 2006.

[21] J. R. Guerci, "COGNITIVE RADAR: THE NEXT RADAR WAVE?," *Microwave Journal,* vol. 54, no. 1, pp. 22-+, Jan 2011.

[22] F. Gini and M. Rangaswamy, *Knowledge based radar detection, tracking and classification*. John Wiley & Sons, 2008.

[23] R. Bakker, T. Kirubarajan, B. Currie, and S. Haykin, "Adaptive radar detection: A bayesian approach," in *Proc. EPSRC IEE Workshop Nonlinear Non-Gaussian Signal Processing*, 2002.

[24] "Introduction," in *Digital Signal Processing with Field Programmable Gate Arrays*Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1-52.

[25] X. Yu, R. Y. Zhang, M. Weber, and A. Zahari, "Compressive Sampling and Real-Time Data Transportation in MPAR Backend System," presented at the AMS, 2015.

[26] X. Yu, Y. Zhang, A. Patel, A. Zahrai, and M. Weber, "An Implementation of Real-Time Phased Array Radar Fundamental Functions on DSP-Focused, High Performance Embedded Computing Platform," in *Proc. of SPIE Vol*, vol. 9829, pp. 982913-1.

[27] X. Yu, Y. Zhang, A. Patel, A. Zahrai, and M. Weber, "An Implementation of Real-Time Phased Array Radar Fundamental Functions on a DSP-Focused, High-Performance, Embedded Computing Platform," *Aerospace*, vol. 3, no. 3, p. 28, 2016.

[28] Y. Pan, Y. R. Zhang, and X. Yu, "AX/Ku dual‐band reflectarray design with cosecant squared shaped beam," *Microwave and Optical Technology Letters*, vol. 56, no. 9, pp. 2028-2034, 2014.

[29] S. Perera, Y. Pan, Y. Zhang, X. Yu, D. Zrnic, and R. Doviak, "A fully reconfigurable polarimetric phased array antenna testbed," *International Journal of Antennas and Propagation*, vol. 2014, 2014.

[30] M. Arakawa, "Computational workloads for commonly used signal processing kernels," DTIC Document2006.

[31] S. F. Reddaway, P. Bruno, P. Rogina, and R. Pancoast, "Ultra-high performance, low-power, data parallel radar implementations," *Ieee Aerospace and Electronic Systems Magazine*, vol. 21, no. 4, pp. 3-7, Apr 2006.

[32] W. L. Melvin and J. A. Scheer, "Principles of Modern Radar, Volume 2 - Advanced Techniques," ed: Institution of Engineering and Technology, 2010.

[33] M. A. Richards, *Fundamentals of radar signal processing*, Second edition.. ed. New York : McGraw-Hill Education, 2014.

[34] J. Ward, "Space-Time Adaptive Processing for Airborne Radar Submitter," MIT Lincoln Laboratory Technical Report 10151994.

[35] B. D. Carlson, "Covariance matrix estimation errors and diagonal loading in adaptive arrays," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, no. 4, pp. 397-401, 1988.

[36] O. Besson and F. Vincent, "Performance analysis of beamformers using generalized loading of the covariance matrix in the presence of random steering vector errors," *IEEE Transactions on Signal Processing*, vol. 53, no. 2, pp. 452-459, 2005.

[37] M. A. Richards, J. A. Scheer, and W. A. Holm, *Priniciples of Modern Radar*. Edison, NJ: Scitech Publishing, 2010.

[38] A. Klilou, S. Belkouch, P. Elleaume, P. Le Gall, F. Bourzeix, and M. M. R. Hassani, "Real-time parallel implementation of Pulse-Doppler radar signal processing chain on a massively parallel machine based on multi-core DSP and Serial RapidIO interconnect," *EURASIP Journal on Advances in Signal Processing*, journal article vol. 2014, no. 1, pp. 1-22, 2014.

[39] T. Instruments, "Enhanced Direct Memory Access 3 (EDMA3) for KeyStone Devices User's Guide (Rev. B)," 2015 May.

[40] K. Miller and M. Rochwarger, "A covariance approach to spectral moment estimation," *IEEE Transactions on Information Theory*, vol. 18, no. 5, pp. 588-596, 1972.

[41] R. Passarelli, P. Romanik, S. Geotis, and A. Siggia, "Ground clutter rejection in the frequency domain(for radar meteorology applications)," in *Conference on Radar Meteorology, 20 th, Boston, MA*, 1981, pp. 295-300.

[42] D. S. Zrnic, "Estimation of Spectral Moments for Weather Echoes," *IEEE Transactions on Geoscience Electronics*, vol. 17, no. 4, pp. 113-128, 1979.

[43] R. J. Doviak and D. S. Zrnic, *Doppler Radar & Weather Observations*. Academic press, 2014.

[44] J. haby. *What is spectrum width*. Available: http://www.theweatherprediction.com/habyhints/245/

[45] R. Srivastava, A. Jameson, and P. Hildebrand, "Time-domain computation of mean and variance of Doppler spectra," *Journal of Applied Meteorology*, vol. 18, no. 2, pp. 189-194, 1979.

[46] M. de Feo, A. Graziano, R. Miglioli, and A. Farina, "IMMJPDA versus MHT and Kalman filter with NN correlation: performance comparison," *Radar, Sonar and Navigation, IEE Proceedings* -, vol. 144, no. 2, pp. 49-56, 1997.

[47] R. Danchick and G. E. Newnam, "Reformulating Reid's MHT method with generalised Murty K-best ranked linear assignment algorithm," *Radar, Sonar and Navigation, IEE Proceedings* -, vol. 153, no. 1, pp. 13-22, 2006.

[48] K. Panta, B.-N. Vo, S. Singh, and A. Doucet, "Probability hypothesis density filter versus multiple hypothesis tracking," in *Defense and Security*, 2004, pp. 284-295: International Society for Optics and Photonics.

[49] I. J. Cox and S. L. Hingorani, "An efficient implementation of Reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 18, no. 2, pp. 138-150, 1996.

[50] Y. Bar-Shalom, F. Daum, and J. Huang, "The probabilistic data association filter," *Control Systems, IEEE,* vol. 29, no. 6, pp. 82-100, 2009.

[51] W. D. Blair, "Design of nearly constant velocity track filters for tracking maneuvering targets," in *Information Fusion, 2008 11th International Conference on*, 2008, pp. 1-7.

[52] Y. Bar-Shalom, *Tracking and data association*. San Diego, Calif.: San Diego, Calif. : Academic Press, 1988.

[53] D. Schreurs, M. O'Droma, A. A. Goacher, and M. Gadringer, Eds. *RF Power Amplifier Behavioral Modeling* (The Cambridge RF and Microwave Engineering Series). Cambridge: Cambridge University Press, 2008.

[54] C. G. Tua, T. Pratt, and A. I. Zaghloul, "A Study of Interpulse Instability in Gallium Nitride Power Amplifiers in Multifunction Radars," *IEEE Transactions on Microwave Theory and Techniques,* vol. 64, no. 11, pp. 3732-3747, 2016.

[55] J. C. Pedro, J. C. Madaleno, and J. A. García, "Theoretical basis for extraction of mildly nonlinear behavioral models," *International Journal of RF and Microwave Computer‐Aided Engineering,* vol. 13, no. 1, pp. 40-53, 2003.

[56] L. B. Chipansky Freire, C. De Franca, and E. G. de Lima, "Low-pass equivalent behavioral modeling of RF power amplifiers using two independent real-valued feed-forward neural networks," *Progress In Electromagnetics Research C,* vol. 52, pp. 125-133, 2014.

[57] J. C. Pedro and S. A. Maas, "A comparative overview of microwave and wireless power-amplifier behavioral modeling approaches," *IEEE Transactions on Microwave Theory and Techniques,* vol. 53, no. 4, pp. 1150-1163, 2005.

[58] A. A. Saleh, "Frequency-independent and frequency-dependent nonlinear models of TWT amplifiers," *IEEE Transactions on communications,* vol. 29, no. 11, pp. 1715-1720, 1981.

[59] A. Soury, E. Ngoya, J.-M. Nebus, and T. Reveyrand, "Measurement based modeling of power amplifiers for reliable design of modern communication systems," in *Microwave Symposium Digest, 2003 IEEE MTT-S International*, 2003, vol. 2, pp. 795-798: IEEE.

[60] D. Mirri, F. Filicori, G. Iuculano, and G. Pasini, "A nonlinear dynamic model for performance analysis of large-signal amplifiers in communication systems," *IEEE Transactions on Instrumentation and Measurement,* vol. 53, no. 2, pp. 341-350, 2004.

[61] P. Draxler, I. Langmore, T. Hung, and P. Asbeck, "Time domain characterization of power amplifiers with memory effects," in *Microwave Symposium Digest, 2003 IEEE MTT-S International*, 2003, vol. 2, pp. 803-806: IEEE.

[62] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on information theory*, vol. 52, no. 2, pp. 489-509, 2006.

[63] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE signal processing magazine*, vol. 25, no. 2, pp. 21-30, 2008.

[64] S. Qaisar, R. M. Bilal, W. Iqbal, M. Naureen, and S. Lee, "Compressive sensing: From theory to applications, a survey," *Journal of Communications and networks*, vol. 15, no. 5, pp. 443-456, 2013.

[65] D. Wu, W.-P. Zhu, and M. Swamy, "A compressive sensing method for noise reduction of speech and audio signals," in *Circuits and Systems (MWSCAS), 2011 IEEE 54th International Midwest Symposium on*, 2011, pp. 1-4: IEEE.

[66] J. F. Gemmeke and B. Cranen, "Noise reduction through compressed sensing," 2008.

[67] D. L. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ1 minimization," *Proceedings of the National Academy of Sciences*, vol. 100, no. 5, pp. 2197-2202, 2003.

[68] J. Tropp and A. C. Gilbert, "Signal recovery from partial information via orthogonal matching pursuit," ed: Apr, 2005.

[69] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proceedings of the National Academy of Sciences*, vol. 106, no. 45, pp. 18914-18919, 2009.

[70] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on information theory*, vol. 53, no. 12, pp. 4655-4666, 2007.

[71] I. Şeker, "Calibration methods for phased array radars," 2013, vol. 8714, pp. 87140W-87140W-15.

[72] M. Scott, "SAMPSON MFR active phased array antenna," in *IEEE International Symposium on Phased Array Systems and Technology, 2003.*, 2003, pp. 119-123.

[73] H. Pawlak, A. Charaspreedalarp, and A. F. Jacob, "Experimental Investigation of an External Calibration Scheme for 30 GHz Circularly Polarized DBF Transmit Antenna Arrays," in *2006 European Microwave Conference*, 2006, pp. 764-767.

[74] K. M. Lee, R. S. Chu, and S. C. Liu, "A built-in performance monitoring/fault isolation and correction (PM/FIC) system for active phased array antennas," in *Proceedings of IEEE Antennas and Propagation Society International Symposium*, 1993, pp. 206-209 vol.1.

[75] C. Fulton and W. Chappell, "Calibration techniques for digital phased arrays," in *2009 IEEE International Conference on Microwaves, Communications, Antennas and Electronics Systems*, 2009, pp. 1-10.

[76] J. Levinson and S. Thrun, "Unsupervised calibration for multi-beam lasers," in *Experimental Robotics*, 2014, pp. 179-193: Springer.

[77] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1-38, 1977.

[78] C. B. Do and S. Batzoglou, "What is the expectation maximization algorithm?," *Nat Biotech*, 10.1038/nbt1406 vol. 26, no. 8, pp. 897-899, 08//print 2008.

[79] C. J. Wu, "On the convergence properties of the EM algorithm," *The Annals of statistics*, pp. 95-103, 1983.

[80] M. K. Nezami, "Fundamentals of power amplifier linearization using digital pre-distortion," *High Frequency Electronics*, vol. 54-59, 2004.

[81] B. Nutten, P. Amayenc, M. Chong, D. Hauser, F. Roux, and J. Testud, "The RONSARD Radars: A Versatile C-Band Dual Doppler Facility," *IEEE Transactions on Geoscience Electronics*, vol. 17, no. 4, pp. 281-288, 1979.

[82] Y. Wang and N. L. Zhang, "Severity of Local Maxima for the EM Algorithm: Experiences with Hierarchical Latent Class Models," presented at the Probabilistic Graphical Models, 2006. Available: http://dblp.uni-trier.de/db/conf/pgm/pgm2006.html#WangZ06

[83] W. Kederer and J. Detlefsen, "Direction of arrival (DOA) determination based on monopulse concepts," in *Microwave Conference, 2000 Asia-Pacific*, 2000, pp. 120-123: IEEE.

[84] M. J. Hinich and P. Shaman, "Parameter estimation for an r-dimensional plane wave observed with additive independent Gaussian errors," *The Annals of Mathematical Statistics*, pp. 153-169, 1972.

[85] U. Nickel, "Angular superresolution with phased array radar: a review of algorithms and operational constraints," in *IEE Proceedings F-Communications, Radar and Signal Processing*, 1987, vol. 134, no. 1, pp. 53-59: IET.

[86] J. Odendaal, E. Barnard, and C. Pistorius, "Two-dimensional superresolution radar imaging using the MUSIC algorithm," *IEEE Transactions on Antennas and Propagation,* vol. 42, no. 10, pp. 1386-1391, 1994.

[87] T. E. Tuncer and B. Friedlander, "1. Wireless Direction-Finding Fundamentals," in *Classical and Modern Direction-of-Arrival Estimation*: Elsevier.

[88] D. G. Childers and S. B. Kesler, *Modern spectrum analysis*. IEEE press New York, 1978.

[89] K. Agarwal and R. Macháň, "Multiple signal classification algorithm for super-resolution fluorescence microscopy," *Nature communications,* vol. 7, p. 13752, 2016.

[90] R. Roy and T. Kailath, "ESPRIT-estimation of signal parameters via rotational invariance techniques," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. 37, no. 7, pp. 984-995, 1989.

[91] O. A. Oumar, M. F. Siyau, and T. P. Sattar, "Comparison between MUSIC and ESPRIT direction of arrival estimation algorithms for wireless communication systems," in *The First International Conference on Future Generation Communication Technologies*, 2012, pp. 99-103.

[92] K. Hwang, *Advanced Computer Architecture: Parallelism,Scalability,Programmability*. McGraw-Hill Higher Education, 1992, p. 771.

[93] S. Fuller, "The opportunity for sub microsecond interconnects for processor connectivity," *Technology Comparisons, RapidIO®, retrieved online,* 2015.

[94] R. T. Association, "RapidIO, PCI express and Gigabit ethernet comparison," ed.

[95] J. Regula, "Using non-transparent bridging in PCI Express systems," *PLX Technology, Inc,* pp. 1-31, 2004.

[96] T. Barry Wood, "Backplane tutorial: RapidIO, PCIe and Ethernet," Available: http://www.eetimes.com/document.asp?doc_id=1275655

[97] S. H. Fuller, *RapidIO : the embedded system interconnect*. Chichester, England ; Hoboken, NJ: Chichester, England ; Hoboken, NJ : John Wiley & Sons, 2005.

[98] D. Bueno, C. Conger, A. D. George, I. Troxel, and A. Leko, "RapidIO for radar processing in advanced space systems," *ACM Trans. Embed. Comput. Syst,* vol. 7, 2007.

[99] J. C. Bennett and E. P. Schoessow, "Antenna near-field/far-field transformation using a plane-wave-synthesis technique," *Electrical Engineers, Proceedings of the Institution of,* vol. 125, no. 3, pp. 179-184, 1978.

[100] J. Appel-Hansen, E. S. Gillespie, T. G. Hickman, and J. D. Dyson, "Antenna measurements," in Electromagnetic Waves, Handbook of Antenna Design, Vol. 1: Institution of Engineering and Technology, 1982, pp. 584-694. [Online]. Available: http://digital-library.theiet.org/content/books/10.1049/pbew015f_ch8.

[101] N. P. (R&D). *What is NTP?* Available: http://www.ntp.org/ntpfaq/NTP-s-def.htm

[102] N. P. (R&D). *How accurate will my Clock be?* Available: http://www.ntp.org/ntpfaq/NTP-s-algo.htm

[103] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (1588-2008)*. USA: USA: IEEE, 2008, pp. 1-269.

[104] D. M. Anand, J. G. Fletcher, Y. Li-Baboud, and J. Moyne, "A practical implementation of distributed system control over an asynchronous Ethernet network using time stamped data," ed, 2010, pp. 515-520.

[105] D. TMS320C66x, "Cache User Guide," *Literature Number: SPRUGY8, Texas Instruments,* 2010.

[106] F. Baskett and A. J. Smith, "Interference in multiprocessor computer systems with interleaved memory," *Communications of the ACM,* vol. 19, no. 6, pp. 327-334, 1976.

[107] J. Lebak, "Polymorphous computing architectures (PCA) example application 4: Corner-turn," *External report, Oct,* 2001.

[108] P. Thambidurai, A. M. Finn, R. M. Kieckhafer, and C. J. Walter, "Clock synchronization in MAFT," in *Fault-Tolerant Computing, 1989. FTCS-19. Digest of Papers., Nineteenth International Symposium on*, 1989, pp. 142-149: IEEE.

[109] B. Simons, "An overview of clock synchronization," in *Fault-Tolerant Distributed Computing*: Springer, 1990, pp. 84-96.

[110] P. Ramanathan, K. G. Shin, and R. W. Butler, "Fault-tolerant clock synchronization in distributed systems," *Computer,* vol. 23, no. 10, pp. 33-42, 1990.

[111] Z. Yang and T. A. Marsland, "Annotated bibliography on global states and times in distributed systems," in *ACM SIGOPS Operating Systems Review*, 1993: Citeseer.

[112] G. Jabłoński *et al.*, "IEEE 1588 Time Synchronization Board in MTCA.4 Form Factor," *IEEE Transactions on Nuclear Science*, vol. 62, no. 3, pp. 919-924, 2015.

[113] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817-840, 2004.

[114] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler, "Wide area cluster monitoring with Ganglia," in *CLUSTER*, 2003, vol. 3, pp. 289-289.

[115] N. B. Jeremy Kepner, "Evaluating the Productivity of a Multicore Architecture," Available: https://www.ll.mit.edu/HPEC/agendas/proc08/Day3/47-Day3-Focus5-Kepner-abstract.pdf.

[116] (2017). *Leading semiconductor companies from 2014 to 2016*. Available: https://www.statista.com/statistics/283359/top-20-semiconductor-companies/

[117] T. Instruments. *66AK2H Evaluation Modules*. Available: http://www.ti.com/tool/evmk2h

[118] PRODRIVE. (2016). *AMC-TK2: ARM and DSP AMC*. Available: https://prodrive-technologies.com/products/arm-dsp-amc/

[119] H. Lin *et al.*, "Cots software selection process," in *Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS'07. Sixth International IEEE Conference on*, 2007, pp. 114-122: IEEE.

[120] W. Wong, "The Automotive Supercomputer," *Electronic Design*, Available: http://electronicdesign.com/embedded/automotive-supercomputer

[121] M. L. Brodie, "The promise of distributed computing and the challenges of legacy systems," in *British National Conference on Databases*, 1992, pp. 1-28: Springer.

[122] Y. M. Kadah, K. Z. Abd-Elmoniem, and A. A. Farag, "Parallel Computation in Medical Imaging Applications," *International Journal of Biomedical Imaging*, vol. 2011, p. 2, 2011, Art. no. 840181.

[123] H. T. Kung and C. E. leiserson, "Systolic Arrays (for VLSI)," in *Sparse Matrix Proceedings 1978*Philadelphia: SIAM, 1979, pp. 256-282.

[124] D. Mann, J. Evans, and M. Merritt, "Clutter suppression for low altitude wind shear detection by Doppler weather radars," in *23rd Conference on Radar Meteorology*, 1986.

[125] S. M. Torres and D. S. Zrnic, "Ground clutter canceling with a regression filter," *Journal of Atmospheric and Oceanic Technology*, vol. 16, no. 10, pp. 1364-1372, 1999.

[126] J. Ward, "Space-time adaptive processing for airborne radar," 1998.

[127] C. S. Morse, R. K. Goodrich, and L. B. Cornman, "The NIMA method for improved moment estimation from Doppler spectra," *Journal of atmospheric and oceanic technology*, vol. 19, no. 3, pp. 274-295, 2002.

[128] B. W. Johnson, "An introduction to the design and analysis of fault-tolerant systems," *Fault-tolerant computer system design*, vol. 1, pp. 1-84, 1996.

[129] J. Liang and Q. Liang, "Design and analysis of distributed radar sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 11, pp. 1926-1933, 2011.

[130] B. C. ord Neuman, "Scale in distributed systems," *ISI/USC*, 1994.

[131] K. E. Martin *et al.*, "IEEE standard for synchrophasors for power systems," *IEEE Transactions on Power Delivery*, vol. 13, no. 1, pp. 73-77, 1998.

[132] G. P. C. Tran, D.-I. Kang, and S. Crago, "Dynamic runtime optimizations for systems of heterogeneous architectures," in *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, 2014, pp. 1-6: IEEE.

[133] G. Schrader, "A KASSPER Real-Time Signal Processor Testbed," MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB2005.

[134] M. Joseph and A. Goswami, "Formal description of realtime systems: a review," *Information and software technology*, vol. 31, no. 2, pp. 67-76, 1989.

[135] A. Jacobs, N. Wulf, and A. D. George, "Task scheduling for reconfigurable systems in dynamic fault-rate environments," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, 2013, pp. 1-6: IEEE.

[136] J. r Sgall, "On-line scheduling| a survey," in *Dagstuhl Seminar on On-Line Algorithms (Schlo Dagstuhl (Wadern), Germany, June 24 {28, 1996), to appear in LNCS. Springer-Verlag, Berlin-Heidelberg-New York*, 1998.

[137] O. Beaumont, V. Boudet, and Y. Robert, "A realistic model and an efficient heuristic for scheduling with heterogeneous processors," in *Parallel and*

Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM, 2001, p. 14 pp: IEEE.

[138] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of parallel and Distributed Computing,* vol. 9, no. 2, pp. 138-153, 1990.

[139] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of discrete mathematics,* vol. 5, pp. 287-326, 1979.

[140] S. Haykin, Y. Xue, and T. N. Davidson, "Optimal waveform design for cognitive radar," in *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*, 2008, pp. 3-7: IEEE.

[141] J. Zhang, D. Zhu, and G. Zhang, "Adaptive compressed sensing radar oriented toward cognitive detection in dynamic sparse target scene," *IEEE Transactions on Signal Processing,* vol. 60, no. 4, pp. 1718-1729, 2012.

[142] J. R. Guerci, "Cognitive radar: A knowledge-aided fully adaptive approach," in *Radar Conference, 2010 IEEE*, 2010, pp. 1365-1370: IEEE.

Abbreviation

ADC	Analog-to-digital converter
AM	Amplitude modulation
AMC	Advanced Mezzanine Card
ARM	Advanced RISC Machines
ATCA	Advanced Telecommunications Computing Architecture
CFAR	Constant False Alarm Rate
CMOS	Complementary metal–oxide–semiconductor
COTS	Commercial off-the-shelf
CPI	Coherent Pulse Interval
CPPAR	Cylindrical Polarimetric Phased Array Radar
CPU	Central Processing Unit
CS	Compressive Sensing
DAR	Digital Array Radar
DFT	Digital Flourier Transform
DMA	Direct Memory Access
DOA	Direction of Arrival
DSP	Digital Signal Processing
EM	Expectation Maximization
ESPRIT	Estimation of Signal Parameters Via Rotational Invariance Techniques
EVM	Evaluation Module
FFT	Fast Flourier Transform
FPGA	field-programmable gate array

GFLOPS	Giga Floating Operation per Second
GPS	Global Position System
GPU	Graphic Processing Unit
HPEC	High Performance Embedded Computing
JPDA	Joint Probabilistic Data Association
MCH	MicroTCA Carrier Hub
MHT	Multiple Hypothesis Tracking
MPAR	Multifunctional Phased Array Radar
MTCA	Micro Telecom Computing Architecture
MTT	Multiple Target Tracking
MUSIC	Multiple Signal Classification
NP	Non-deterministic Polynomial-time
OMP	Orthogonal Matching Pursuits
PA	Power Amplifier
PAR	Phased Array Radar
PCI	Peripheral Component Interconnect
PDAF	Probabilistic Data Association Filter
PDF	Probability Density Function
PE	Processing Element
PM	Processing Module
PPS	Pulse per Second
PRI	Pule Repetition Interval
PTP	Precision Time Protocol

PU	Processing Unit
RCS	Radar Cross Section
RF	Radio Frequency
RIP	Restricted Isometry Property
SATA	Serial at Attachment
SENSR	Spectrum Efficient National Surveillance Radar
SNR	Signal Noise Ratio
SRIO	Serial Rapid IO
STAP	Space Time Adaptive Processing
TCP	Transmission Control Protocol
TR	Transmit and Receive
UDP	User Datagram Protocol
VLSI	Very Large-Scale Integration