NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

OPTIMIZING ENGINEERING TOOLS USING MODERN GROUND ARCHITECTURES

by

Ryan P. McArdle

December 2017

Thesis Advisor:	Marc Peters
Co-Advisor:	I.M. Ross

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis
4. TITLE AND SUBTITLE OPTIMIZING ENGINEERING TOOLS USING MODERN GROUND ARCHITECTURES		**5. FUNDING NUMBERS**
6. AUTHOR(S) Ryan P. McArdle		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.		**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Over the past decade, a deluge of large and complex datasets (aka "big data") has overwhelmed the scientific community. Traditional computing architectures were not capable of processing the data efficiently, or in some cases, could not process the data at all. Industry was forced to reexamine the existing data processing paradigm and develop innovative solutions to address the challenges. This thesis investigates how these modern computing architectures could be leveraged by industry and academia to improve the performance and capabilities of engineering tools. First, the effectiveness of MathWorks' Parallel Computing Toolkit is assessed when performing somewhat basic computations in MATLAB. Next, a more computationally intensive series of tests using synthetic aperture radar datasets is demonstrated using the MATLAB/Simulink Toolbox and Apache Spark, a powerful distributed processing framework. Finally, hyperspectral sensor datasets are processed using the MATLAB Hyperspectral Toolbox and machine learning libraries in Apache Spark to demonstrate the additional capabilities that modern computing architectures enable.

14. SUBJECT TERMS distributed processing, MATLAB, Apache Spark, ISR data, synthetic aperture RADAR, hyperspectral sensor, Amazon Web Services			**15. NUMBER OF PAGES** 81
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

OPTIMIZING ENGINEERING TOOLS USING MODERN GROUND ARCHITECTURES

Ryan P. McArdle
Civilian, The Boeing Company
B.S., Loyola Marymount University, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ASTRONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2017

Approved by:	Marc Peters Thesis Advisor
	I.M. Ross, Ph.D. Co-Advisor
	Garth V. Hobson, Ph.D. Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Over the past decade, a deluge of large and complex datasets (aka “big data”) has overwhelmed the scientific community. Traditional computing architectures were not capable of processing the data efficiently, or in some cases, could not process the data at all. Industry was forced to reexamine the existing data processing paradigm and develop innovative solutions to address the challenges. This thesis investigates how these modern computing architectures could be leveraged by industry and academia to improve the performance and capabilities of engineering tools. First, the effectiveness of MathWorks’ Parallel Computing Toolkit is assessed when performing somewhat basic computations in MATLAB. Next, a more computationally intensive series of tests using synthetic aperture radar datasets is demonstrated using the MATLAB/Simulink Toolbox and Apache Spark, a powerful distributed processing framework. Finally, hyperspectral sensor datasets are processed using the MATLAB Hyperspectral Toolbox and machine learning libraries in Apache Spark to demonstrate the additional capabilities that modern computing architectures enable.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

2D	Two-Dimensional
3D	Three-Dimensional
FFT	Fast Fourier Transformation
HPEC	High Performance Embedded Computing
IC ITE	Intelligence Community Information Technology Enterprise
MHT	MATLAB Hyperspectral Toolbox
NIST	National Institute of Standards and Technology
SAR	Synthetic Aperture RADAR
vCPU	Virtual Central Processing Unit

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank all of the Naval Postgraduate School faculty and staff that I had the pleasure of working with over my relatively short time in Monterey.

To my 591 cohorts: Thank you for welcoming a civilian contractor into your circle and allowing me to be part of the team.

To the Boeing Mission Processing Framework Team: Your innovations have changed the future of mission data processing.

To my two daughters, Ava and Cara: You can achieve anything you set your mind to. Never let anyone tell you otherwise.

Last (and certainly not least), to my wife, Jennifer: Thank you for supporting me and providing the encouragement needed to complete this adventure. Without your love and patience, none of this would have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Over the past decade, there has been a 1600% increase in the high-definition imagery collected from drones, satellites and in-situ data sensors [1]. This is just one example of the impact that "big data" has had on the scientific community. "Big data" is a collection of data sets so large and complex that traditional approaches to capturing, indexing, storing, exploiting, visualizing and analyzing have become increasingly difficult.

The big data paradigm requires new approaches to tasking, processing, exploitation and dissemination of the data. Traditional techniques using serial processing and dedicated hardware may not be sufficient to handle the four main characteristics "V"s of big data [1].

Volume—the scale of the collection

Variety—varying sources

Veracity—the uncertainty of disparate feeds

Velocity—speed of incoming streams

This influx of data led to the rise in on-demand, virtualized resources ("cloud computing"), such as Amazon Web Services, and provided a whole new resource for developers to use. In order to address these new challenges and to take advantage of virtualized hardware, frameworks designed specifically for distributed cluster computing became increasingly popular.

In a recent Q&A in Geospatial Intelligence Forum, National Reconnaissance Office (NRO) Director Ms. Betty Sapp discussed the agency's newly created Intelligence Community Information Technology Enterprise (IC ITE) [2]. Introduced in 2011 by Director of National Intelligence (DNI) James Clapper, IC ITE is the intelligence community's common IT platform. This infrastructure includes the capability for developers to request on-demand computing resources, therefore eliminating the need for an application to be hosted on dedicated hardware.

When asked how she saw the IC ITE changing the way the NRO conducts business, Ms. Sapp replied:

> The IC ITE is not only changing the way we run IT at NRO, but it is also changing how we utilize capabilities from other IC elements—that is, the IC ITE services. In short, IC ITE is going to be a huge enabler for NRO and allow us to expose much more of our data to analysts and users, much earlier and much more often than we had previously been able to do. It really is a game-changer for us. NRO is already putting metadata of collected imagery into the IC ITE cloud environment, making it more accessible to the IC. We have also started utilizing the hardware in the cloud architecture for development and test of some ground software systems. [3]

When Ms. Sapp was asked how she saw the NRO's ground infrastructure evolving in the years ahead, she responded:

> As the NRO space segment moves to increased persistence and diversity, the ground will use new innovative means to improve products, create new products, counter physical gaps in coverage and improve analytics, multi-INT opportunities, activity-based intelligence, object-based production and predictive models.
>
> The move to the cloud-based IC ITE will enable the NRO ground to continue to provide current capabilities and products while striving to improve ground resiliency through flexible, sensor agnostic apps and services hosted anywhere in the world. The NRO ground architecture will include a new enterprise collection orchestration (ECO) function to maximize and optimize collection opportunities, fully exploiting integrated intelligence alerts, providing a more automated tipping and cueing capability to enable collections that are relevant and utilizing all available sensors. The goal is to allow the role of the analyst to evolve from sifting through large amounts of data to working the actionable, relative data that is provided to them. [3]

B. OBJECTIVE

The objective of this thesis is to explore how common engineering tools used in academia and the aerospace industry can be adapted to take advantage of features provided by modern ground architectures. This will be accomplished by demonstrating various types of applications of different complexity and, in some cases, using on-demand cluster computing.

C. OVERVIEW

This thesis will explore a variety of tools commercially available (and in some cases, free) to data scientists and data analysts to improve performance and enhance capabilities. Chapter II provides insight into modern computing architectures, including techniques for processing large data sets in parallel. It also describes the paradigm shift from dedicated computing resources to virtualized, on-demand resources hosted on a cloud platform. The chapter ends with a description of powerful open-source software applications that have been optimized for cluster-computing environments. Chapter III describes some of the more common principles and terminology in parallel computing. Chapter IV describes in detail the two computing applications (MATLAB and Apache Spark) used in this thesis to demonstrate these concepts. Chapter V puts to use the principles and tools described in prior chapters by demonstrating how a variety of computational operations, ranging from simple, linear calculations to complex machine learning routines can be improved by these new methods.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

The complexity of big data has been a catalyst in the development of frameworks designed to take advantage of parallel processing and scalable computing resources. Some of these capabilities already exist in popular applications (such as Mathworks' MATLAB) and only require minor code modifications for the user to take advantage of them. Others might require a bit more specific domain expertise and setup, but the additional capabilities (not to mention the potential savings in time) may be worth the additional effort. Virtual, on-demand computing resources could also be utilized to scale up computing power to meet processing or mission needs.

A. PARALLEL COMPUTING

Parallel Computing practices can be utilized by data scientists and analysts to address multiple types of computational situations. MathWorks, the company responsible for the powerful number-crunching application MATLAB, breaks these situations down into three different use cases and provides insight into how each use could be used to optimize performance [4].

1. Parallel "For" Loops

Generally speaking, engineering tools are designed to execute multiple command instructions in a repetitive manner. In order to perform these tasks in an efficient manner, most modern coding languages provide a "for" loop construct that executes desired commands over a pre-determined number of iterations. These iterations are performed in a serial manner and will not begin until the prior iteration has completed. There are certain cases where performance could be improved dramatically if these iterations are run in parallel on one computer or on a cluster of computers [4]. These cases include:

Parameter Sweep Applications

Many iterations—a task requires a larger number of iterations, negatively impacting system performance

Long iterations—each iteration in performs a significant number of operations, taking a long time for the iteration to complete

Test Suites with Independent Segments

Applications may run a series of unrelated tasks, as long as there is no dependency on any other iterations

2. Executing Batch Jobs in Parallel

Tasks can be offloaded to worker nodes to be run as a batch job. This allows a client session to continue with regular execution while the worker node completes the assigned task. In some cases, this worker node could be run on the same machine or on a remote cluster virtually connected to the client [4].

3. Partitioning Large Data Sets

Data sets can be too large to fit into a single computer's memory, causing the management of this data to become unwieldy for analysis tools that are attempting to operate on this data set. Using parallel processing techniques, this data set can be distributed across multiple computers and each subset can be operated on by its assigned worker [4].

B. CLOUD COMPUTING

Traditionally, computing systems required dedicated resources and custom applications. It wasn't until recently that a whole paradigm of on-demand, virtual "cloud computing" resources was introduced to the public.

Cloud computing typically refers to a computer architecture that enables on-demand computing resources (networks, servers, storage, applications and services) [5]. This infrastructure should be rapidly provisioned (and subsequently released) with minimal effort or human interaction.

According to the National Institute of Standards and Technology (NIST), there are five essential characteristics of a cloud computing models, three types of service models and four deployment models [5], shown in Table 1.

Table 1. Cloud Computing Models. Source: [5].

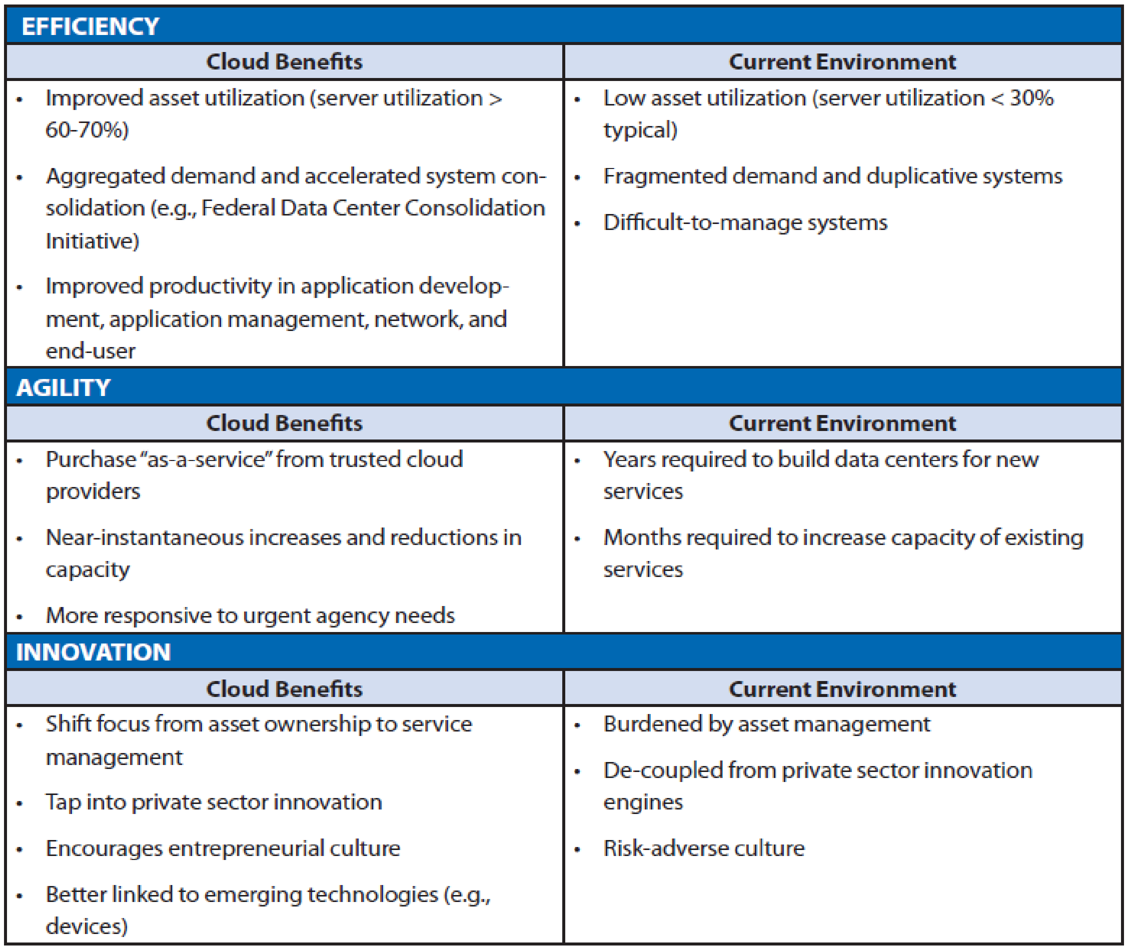
Characteristics	
On-demand self-service	A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
Broad network access	Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
Resource pooling	The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.
Rapid elasticity	Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

Measured service	Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.
Service Models	
Software as a Service (SaaS)	The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings.
Platform as a Service (PaaS)	The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.3 The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Service Models	
Infrastructure as a Service (IaaS)	The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).
Deployment Models	
Private cloud	The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.
Community cloud	The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises
Public cloud	The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider

Deployment Models	
Hybrid cloud	The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Utilizing a cloud infrastructure offers the user an opportunity to efficiently manage IT investments. If a user is developing a new program that requires high-performance computing, a cloud infrastructure can be easily procured without having to acquire dedicated hardware, allowing more a more rapid deployment. Additionally, there are no operations and maintenance costs, as there would be with dedicated hardware. According the federal government's Cloud Computing Strategy [6], the following benefits shown in Figure 1 can be realized with a cloud computing environment.



EFFICIENCY	
Cloud Benefits	**Current Environment**
• Improved asset utilization (server utilization > 60-70%) • Aggregated demand and accelerated system consolidation (e.g., Federal Data Center Consolidation Initiative) • Improved productivity in application development, application management, network, and end-user	• Low asset utilization (server utilization < 30% typical) • Fragmented demand and duplicative systems • Difficult-to-manage systems
AGILITY	
Cloud Benefits	**Current Environment**
• Purchase "as-a-service" from trusted cloud providers • Near-instantaneous increases and reductions in capacity • More responsive to urgent agency needs	• Years required to build data centers for new services • Months required to increase capacity of existing services
INNOVATION	
Cloud Benefits	**Current Environment**
• Shift focus from asset ownership to service management • Tap into private sector innovation • Encourages entrepreneurial culture • Better linked to emerging technologies (e.g., devices)	• Burdened by asset management • De-coupled from private sector innovation engines • Risk-adverse culture

Figure 1. Summary of Cloud Computing Benefits. Source: [6].

C. DISTRIBUTED PROCESSING FRAMEWORKS

The big data surge has also challenged commercial software developers to find creative ways to tap into the potential value of available data. Frameworks developed to scale horizontally as processing demands fluctuate and minimize the latency of the required processing began to surface to meet this need. Applications such as Hadoop MapReduce and Apache Spark provide the tools necessary to effectively navigate the seas of big data.

THIS PAGE INTENTIONALLY LEFT BLANK

III. COMPUTING COMPLEXITY AND PARALLEL PROCESSING THEORIES

A. AMDAHL'S LAW

The effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.

—Gene Amdahl [7]

Amdahl was a chief architect at IBM in the 1960s and is considered a pioneer of mainframe computing. He formulated "Amdahl's Law," which states the fundamental limitation of parallel computing [8]. "Amdahl's Law" asserts that the execution time (T_1) of a program falls into two categories: time spent doing non-parallelizable serial work (W_{SER}) and time spent doing parallelizable work (W_{PAR}). The execution time required for a program is described in Equation 3.1 as:

$$T = W_{SER} + W_{PAR} \tag{3.1}$$

where "T" is the total execution time, W_{SER} is the time required to perform the serial work and W_{PAR} is the time required to perform the parallel work.

With "p" number of workers to do the parallelizable work, the execution time can be expressed as shown in Equation 3.2:

$$T_p = W_{SER} + (W_{PAR} / p)\,. \tag{3.2}$$

Amdahl's Law is the ratio of the serial execution time to the execution time with "p" workers performing on the parallelizable work. Using Equations 3.1 and 3.2, this speedup ratio ("S_P") is defined in Equation 3.3.

$$S_p = \frac{W_{SER} + W_{PAR}}{W_{SER} + (W_{PAR} / p)} \tag{3.3}$$

Figure 2 is a visualization of Equation 3.3. As the number of workers ("p") increases, the time it takes to execute the parallelizable portion of the work decreases at the same rate.

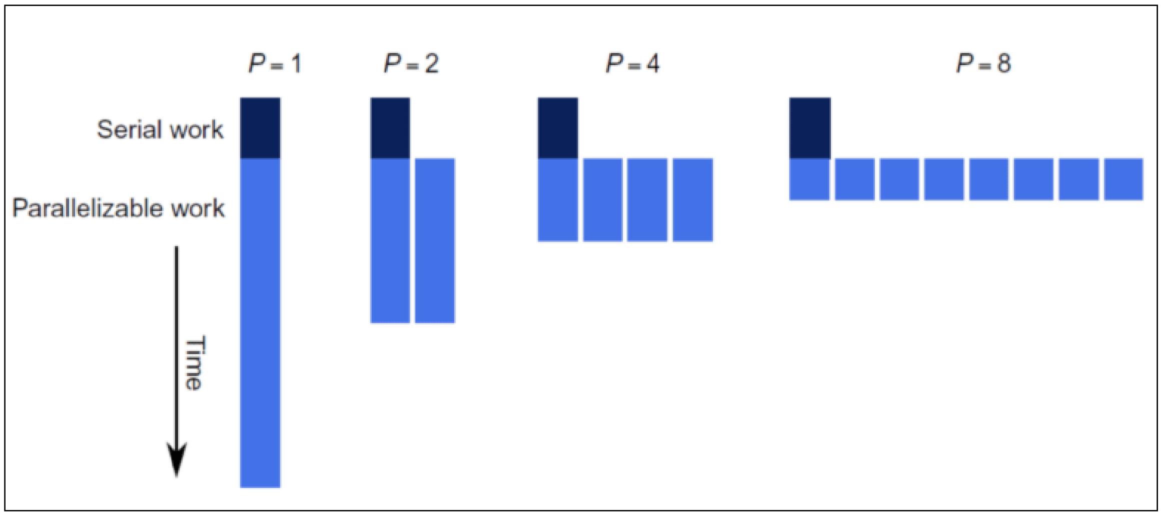


Figure 2. Visualization of Amdahl’s Law. Source: [7].

Another way to express the serial portion of the work is by considering it as a fraction of the total work (“f”)

$$W_{SER} = fT \tag{3.4}$$

Conversely, the parallelized portion of the work can also be described as a function of “f”:

$$W_{PAR} = (1-f)T \tag{3.5}$$

When Equations 3.4 and 3.5 are substituted into Equation 3.3, the equation for speedup can be defined as:

$$S_P = \frac{1}{(f + (1-f)/\mathrm{p})} \tag{3.6}$$

And as P goes toward infinity:

$$S_{\inf} = 1/f \tag{3.7}$$

Therefore, speedup is limited by the fraction of work that is not parallelizable, even with an infinite number of worker nodes to distribute the parallelizable work load [7]. For example, if 10% of the work is serial, then the maximum speedup is 10x.

B. GUSTAFSON-BARSIS' LAW

Computer scientist John Gustafson had a different view on the scalable computing problem and, based on the performance of programs at Sandia National Laboratories, believed that Amdahl's Law could be evaded [7].

> Speedup should be measured by scaling the problem to the number of processors, not by fixing the problem size.
>
> —John Gustafson [7]

Amdahl's law considers the problem as fixed and the computing resources as scalable. However, it does not take into account the continual improvements in computing technologies. As these technologies advance, the applications developed to exploit these new technologies mature in parallel. Gustafson believed that problem sizes grew as computers grew and the work required for the parallel portions of the problem grew at a much faster pace than the serial. The serial time remained the same, but would diminish as a fraction of the whole, as shown in Figure 3.

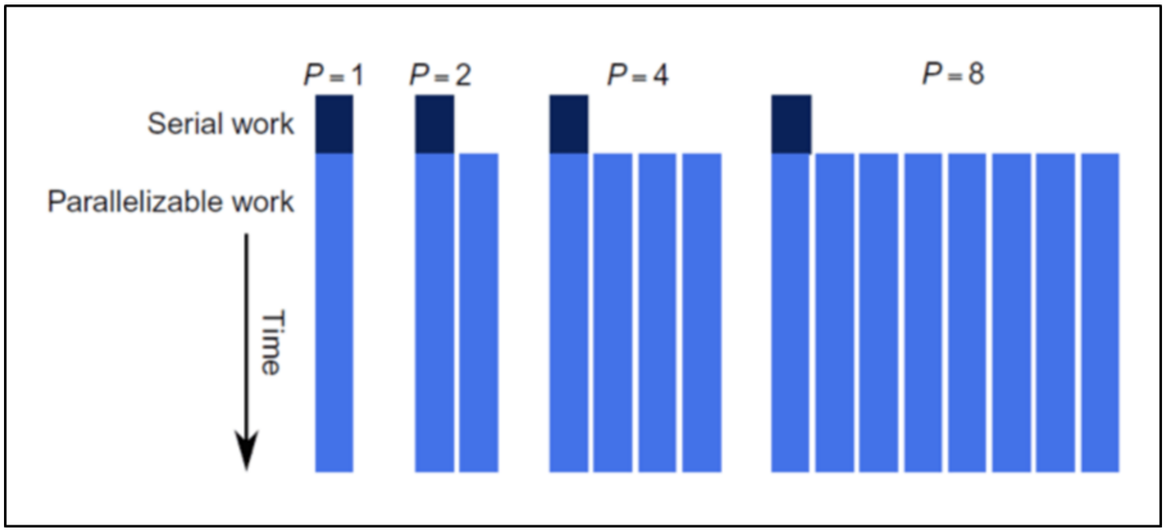


Figure 3. Visualization of Gustafson-Barsis' Law. Source: [7].

As Figure 3 shows, the serial portion becomes insignificant, the ability to take on new work without adding execution time grows at the same rate as the number of processors, achieving a linear speedup.

C. PARALLEL SCALABILITY

"Scalability" refers to the ability to be scaled up to meet demand through replication and distribution of work across a pool of workers [9]. "Strong scaling" occurs when the problem size is fixed and resources are added to proportionally improve performance. A program with strong scaling will typically see a linear speedup that is equal to the amount of processors available for the problem. Conversely, an application categorized as having weak scaling would not experience a change in speedup, regardless of how many resources are utilized.

D. COMPLEXITY AND "BIG-O" NOTATION

The complexity of a function is expressed using "Big-O" Notation and describes how fast a function grows or declines. This notation describes the worst-case scenario and can be used to size appropriate resources (e.g., memory or disk space) for a specific application. This notation is helpful in easily describing the general complexity of the computations involved in an application.

Suppose "f(x)" and "g(x)" are two functions defined on some subset of real numbers. It can be written that:

$$f(x) = O(g(x)) \tag{3.8}$$

if, and only if, there exists constants N and C such that $|f(x)| <= C|g(x)|$ for all x>N

For example, when considering some algorithm, the number of steps required to complete a function of size "n" can be given as the function T(n) = $9n^3 + 8n^2 - 14n + 9$. If the constants and the slower growing terms (e.g., quadratic and linear) are ignored, it can be said that "T(n) grows at the order of n^3" or "T(n) = $O(n^3)$" [10].

Table 2 is a list of common functions used when analyzing complexity:

Table 2. Big-Notation and Functions. Adapted from [10].

Notation	Name
O(1)	Constant
O(log(n))	Logarithmic
O(n)	Linear
O(n^2)	Quadratic
O(n^c)	Polynomial

This notation is useful when providing a quick “order of magnitude” estimation when designing the computing architecture to fit a specific problem.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. TOOLS

There are many commercial off-the-shelf products available to take advantage of modern computing architectures. Some of these products, such as MATLAB and its Parallel Computing Toolbox, require expensive software licenses that are not easily portable from system to system. These license requirements could make a system like MATLAB less desirable for developers working under budgetary constraints. The boom in large data processing in the commercial sector has led to the development of powerful applications to analyze large datasets, such as network log files, email content and GEOINT sensor collects. Some of these applications are free, open-source software (FOSS) that are viable alternatives to licensed software. These section will discuss solutions using both licensed software and FOSS.

A. MATLAB FOR PARALLEL PROCESSING

1. A Brief History of MATLAB Parallel Processing

MATLAB is arguably the most widely used complex computation software application suite used by the industry and academia. Started in 1984 by Cleve Molder, Jack Little and Steve Bangert, MathWorks was meant to address the need of technical computing brought on by the emerging personal computer [11]. MATLAB did not immediately embrace the concept of distributed processing since MATLAB’s memory model did not align with that of the parallel computing memory model and adaptation of the baseline code would require a significant effort. It was believed by the company in 1995 that only a small portion of its execution time could be automatically parallelized and there wouldn’t be the market for these tools [12]. The data environment began to change, however, and the need to evolve was largely due to: a) The MATLAB software suite had evolved dramatically, b) Microprocessors with multiple computational cores were now common, c) Memory structures became more sophisticated and d) Users now had increased access to computing clusters. MathWorks finally recognized the benefits of parallel computing and the resultant services are shown in Table 3.

Table 3. MATLAB Parallel Processing Tools and Services. Adapted from [13].

	Parallel Computing Toolbox	MATLAB Parallel Cloud	Computing Server for Amazon EC2	Computing Server Private Cloud
Maximum Workers	No limitation	16	256	No limitation
Hardware Resources	Desktop computer	MathWorks Cloud	Amazon EC2 Instances	Private cloud, other cloud services, on premise and ad-hoc clusters, and grids
First-Time Configuration Effort	None	A few clicks in MATLAB	Amazon EC2 sign up and set up followed by a few clicks in Cloud Center and MATLAB	Software installation followed by scheduler configuration
Time to Access Configured Solution	Instant	< 90 seconds	< 15 minutes	Solution dependent
Customization Options	None	None	Available through Cloud Center Options include cluster size, machine type, storage options	Options include multiple cluster configurations, storage types and schedulers
Licensing Model	Toolbox license	Self-serve On-demand license	On-demand, perpetual or term license	On-demand, perpetual or term license
Geographic Availability	Worldwide	United States and Canada	United States, Canada, and other select countries	Worldwide

2. MATLAB Parallel Toolbox

In order to address the need for parallel computing tools, MathWorks released its first version of its “Distributed Computing Toolbox” in 2005. This somewhat basic set of tools provided the user with tools for managing multiple, independent MATLAB jobs [13]. Over the years, subsequent releases have built on the prior release and now provide tools to support key operations such as parallel loops, batch processing, and detailed job management [12].

3. MATLAB Parallel Cloud

With a few mouse clicks (and a valid credit card, of course), MATLAB users can run their applications on the MATLAB Parallel Cloud, which provides an on-demand computing cluster of (up to) 16 virtual worker nodes. This provides the user with all the benefits of a larger cluster without the overhead costs of setting up and maintaining their own computing cluster. Key features of the MATLAB Parallel Cloud include; a) Ready-to-use instances running MATLAB workers, b) 16-core machines with 60GB RAM optimized for MATLAB computations, and c) "Pay as you go" pricing (~$4->$6 per hour, depending on user license) [13]:

4. MATLAB Distributed Computing Server for Amazon EC2

If the 16-core MATLAB Parallel Cloud does not provide the computing resources needed, MATLAB Distributed Computer Server for Amazon EC2 allows the user to run on a customizable cluster on Amazon's cloud [14]. MATLAB's Cloud Center provides a simple interface to configure a cluster for a specific need.

5. MATLAB Distributed Computing Server (Private Cloud)

If the resources provided by the Amazon EC2 cluster are still not sufficient to address performance or operational needs (e.g., security), MATLAB provides the option to configure personally owned computing clusters for use as a MATLAB distributed computing cluster. This allows the user to scale the number of worker nodes to fit their needs and administer security protocols as required. This environment will not be covered in this thesis.

6. MATLAB Data Processing Toolboxes

a. SAR Processing Toolbox

In addition to toolboxes for parallel computing, MATLAB also provides toolkits for efficiently processing specific types of datasets. One particular toolkit, the Synthetic Aperture RADAR (SAR) Processing Toolkit, is provided with the standard license [15]. These toolkits include algorithms designed to perform the complex transformations on SAR data in an efficient manner. They can also be paired with parallel processing constructs for batch processing of SAR data on a distributed cluster.

b. Open Source Toolboxes

The MATLAB user community is an alternative source for useful engineering toolkits. Developers can make their custom algorithms available to other MATLAB users for little to no cost. Developers are free to share their toolboxes through popular online repositories such as Github (www.github.com) and BitBucket (www.bitbucket.org).

A popular example of a free, open-source toolbox is the MATLAB Hyperspectral Toolbox (MHT) [16]. This toolbox is a collection of algorithms that process and exploit hyperspectral sensor data using MATLAB. Included in this set are routines that perform the more complex unsupervised learning operations on the datasets, thus enabling more efficient processing in more tactical timelines.

B. APACHE SPARK

While MATLAB is a powerful application and the Parallel Computing Toolbox harnesses the power of cluster computing, it does come with some disadvantages. One, in particular, are the fees associated with the various licenses required to run MATLAB in a distributed manner. For a (relatively) basic computing cluster containing one master node and sixteen worker nodes, the licensing costs quickly escalate to values outside of a cost-conscience developer's budget.

Table 4. MATLAB License Costs. Adapted from [17].

Basic Individual License	$2150
Parallel Computing Toolbox	$1000
Sub-Total:	$3150
Distributed Cluster License (minimum 16 worker)	16 x $341/worker = $5456
Total:	$8606

Fortunately for the budget-constrained developer, big data has driven a change in the paradigm from expensive customized solutions to FOSS-based solutions. Furthermore, in order to analyze data effectively, large datasets need to be processed interactively and cannot be constrained by long processing times. Apache Spark is a robust cluster computing platform designed to be both multi-purpose and fast [18]. Spark extends the popular MapReduce model, a programming model designed for processing large datasets in a distributed manner, by adding more capabilities such as streaming and interactive queries. The features of Apache Spark make it a popular framework for both data scientists and engineers. While both groups may have different use cases in mind, the general-purpose nature of Apache Spark makes it appealing to both types of users. The Apache Spark stack consists of five main capabilities.

Spark Core contains the more basic capabilities of Spark, including most of the "behind-the-scenes" orchestration for distributing processing [19]. The Spark Core also contains the Application Programming Interface (API) that defines the main program abstraction of Apache Spark, the resilient distributed dataset (RDD). An RDD is a collection of data items that can be distributed across multiple computing nodes so that the data can be processed in parallel [18].

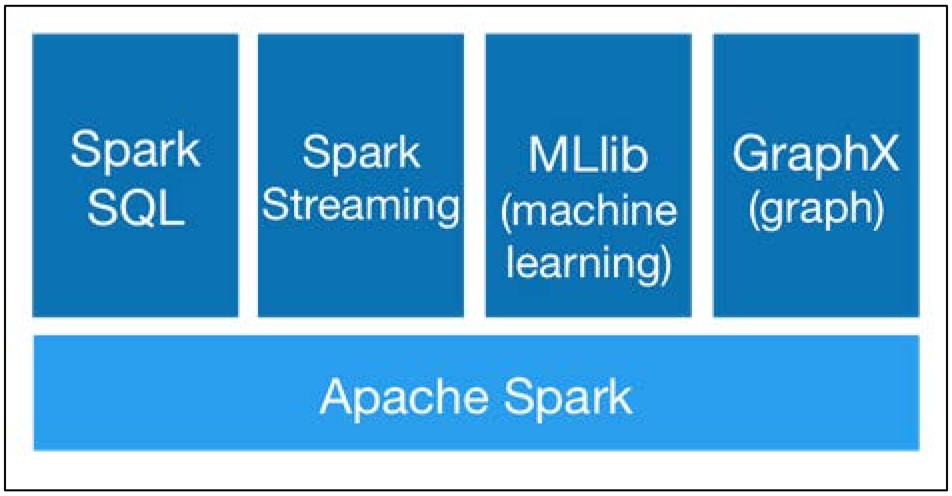


Figure 4. The Apache Spark Stack. Source: [18].

1. Spark SQL

Spark SQL is the component of Apache Spark that allows a developer to manipulate structured data [18]. It allows for use of a standard relational database system called Structured Query Language (SQL) and supports many of the more common data structures, including Hive tables, Parquet and JSON. Spark SQL provides the developer an ability to utilize all the data manipulations associated with RDDs using common, industry proven data structure constructs.

2. Spark Streaming

Apache Spark also provides the capability to process live streams of data through the Spark Streaming component [18]. This feature is useful for data streams that are continually generating content, such as web server log files, social media providers, or telemetry streams. Data streams can be manipulated using a similar API to the RDD API, allowing a developer to remain agnostic to whether the data is static or streaming, or resident in memory or on disk.

3. MLlib

Apache Spark also contains a machine learning library known as “MLlib.” This library provides several types of machine learning algorithms such as classification, regression, clustering and collaborative filtering [18]. The computationally heavy MLlib algorithms are designed to take advantage of parallelization by distributing the work across a cluster.

4. GraphX

GraphX is Apache Spark’s library for manipulating graphs and performing graph-parallel computations [18]. Just like Spark Streaming and Spark SQL, GraphX extends the Spark RDD API, allowing a developer to graphically display data residing in RDDs.

5. Cluster Managers

The power of Apache Spark is based on the ability to scale from a single node to as large of a cluster as you need (or have available) [18]. Spark can effectively operate over common cluster managers such as Hadoop YARN or Apache Mesos. The Apache Spark stack also includes its own cluster manager called the Standalone Scheduler.

This thesis will focus on the improvements offered by the Spark Core and MLlib machine learning libraries. These two components offer the most upfront capability without requiring too much specialized knowledge.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DEMONSTRATIONS

A. MATLAB DEMONSTRATIONS

While it may not be revolutionary to assert that parallel processing can reduce execution times of looped software code, the resources available and their ease of use are not often known. This section describes some basic techniques for developers in academia and industry to take advantage of parallel processing in MATLAB.

1. Basic “For” versus “Parfor” Loop (Linear Complexity)

The following example demonstrates the difference in how the “for” loop and “parfor” is executed in MATLAB. The regular “for” loop example evaluates the desired code serially, beginning with “i = 1” and incrementing the value of “x” by 1 until x=360 (one full circle). In this example, the function will return the cosine value of x and save the result in the array “y.” This operation would be considered of linear complexity (O(n)), due to the single, fundamental operation being performed.

Table 5. Regular “For” Loop Test Information

Regular “For” Loop	

Code:	for x= 0:360
	        y(x) = cos(x)
	end

Average Execution Time (50 runs):	0.003 seconds

Figure 5 demonstrates that the “for” loop construct executes the threads of a loop in a serial manner. The loop executes the called function for x = 0, then x = 1…until x=360 when the loop ends. This is an important characteristic of a “for” loop since, for some uses, the value of f(x) may be dependent on the value of f(x-1) and cannot be evaluated until execution of f(x-1). If this is the case, parallel processing may not be the best solution for a problem of linear complexity.

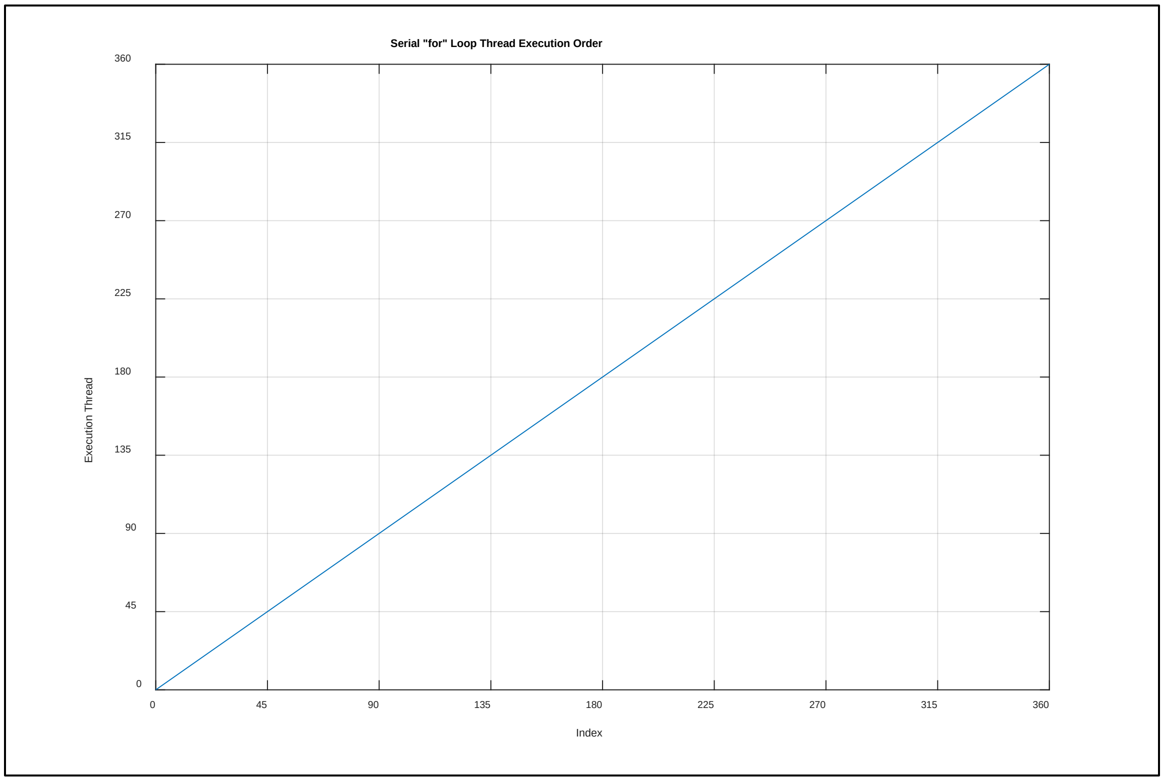


Figure 5. Serial “For” Loop Thread Execution Order

In order to evaluate the difference when using parallel processing, the “for” construct was replaced by the “parfor” construct and the same scenario can be evaluated using parallel processing techniques.

Table 6. Regular “Parfor” Loop Test Information

Parallel “parfor” Loop	

Code:	parfor x= 1:360
	        y(x) = cos(x)
	end

Average Execution Time:	0.0735 seconds

The average execution time for the “parfor” loop (0.0735 seconds) was much higher than that of the “for” loop (0.003 seconds). The “for” loop could execute almost 1000x as many times as the “parfor” loop and complete roughly in the same amount of time. While this may seem counterintuitive to the notion that parallelism is supposed to reduce the execution time, the behavior is expected for such a simple calculation. The higher execution time is due to the fact that use of "parfor" construct is meant for more complex computations. In order to coordinate the distribution of the data and algorithms to the worker nodes, there is some overhead processing prior to the beginning of the "parfor" loop. When the functions are relatively simple, this overhead cost far outweighs benefit of parallelism.

As part of the "pre-processing" done by the parallelism is the creation of a queue of tasks for each of the nodes. This will typically result in a non-serial order of execution (see Figure 6). In other words, iteration 25 of the parfor loop may complete before iteration 1 starts. Figure 6 shows the asynchronous order of execution of the prior run.

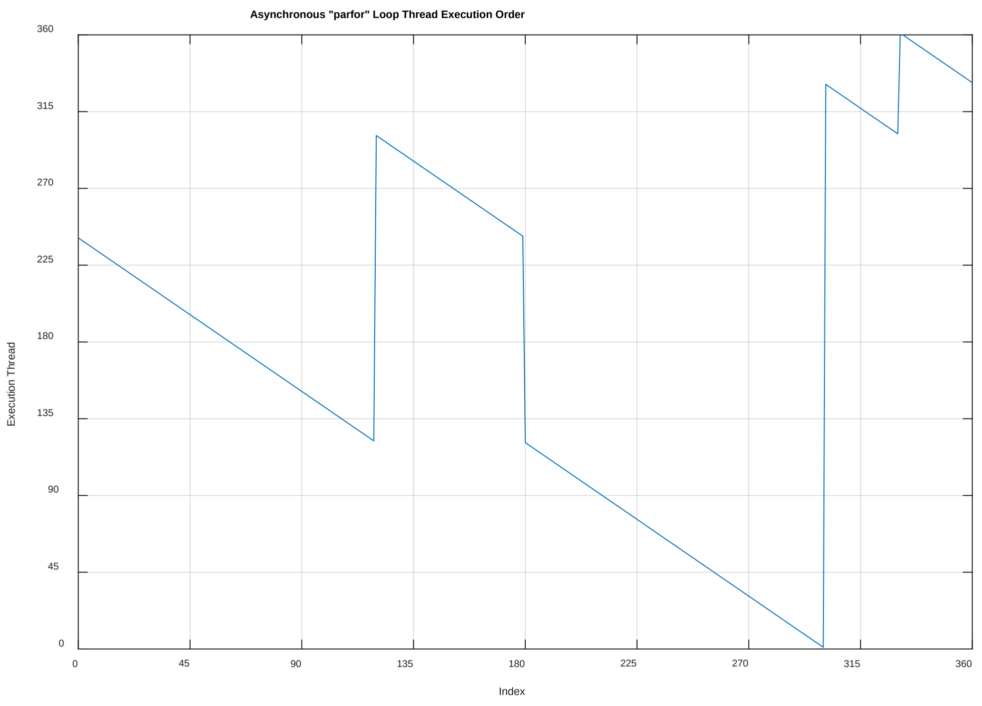


Figure 6. Asynchronous "Parfor" Loop Thread Execution Order

In this case, the “parfor” construct executor created an initial batch that began with thread 240 and worked linearly to 122. The second batch began at 301 and worked down to 240. The third began at 239 and finished at index 0, and subsequent batches perform in a similar manner. The execution pattern is not consistent across multiple runs of the same parfor loop. This information reinforces the idea that data that is dependent on prior states is not a good candidate for a “parfor” loop (see II.A.1.b).

2. Blackjack Benchmark (Distributed Processing)

To further characterize the performance of the parfor construct, MATLAB provides a Blackjack simulator that utilizes the “parfor” construct and benchmarks performance against the standard “for” loop. The benchmark repeatedly plays a game of blackjack in parallel using the same number of players and hands. The benchmark test utilizes an optional argument in the “parfor” construct that allows the developer to designate the number for workers to perform the test. This allows data to be collected from 2 through the maximum available number of workers.

For this section, the maximum number of available workers is 2, so only one data point will be collected and compared against the standard “for” loop performance. In later sections, the performance of larger clusters will be evaluated.

Using the 2 local worker nodes, the benchmark test was run 100 times and the median speed improvement was 1.37, which corresponds to an “efficiency” of 0.67 (Figure 7). Even with a minimal number of worker nodes, the benefits of parallelism are becoming more apparent than with a basic function call.

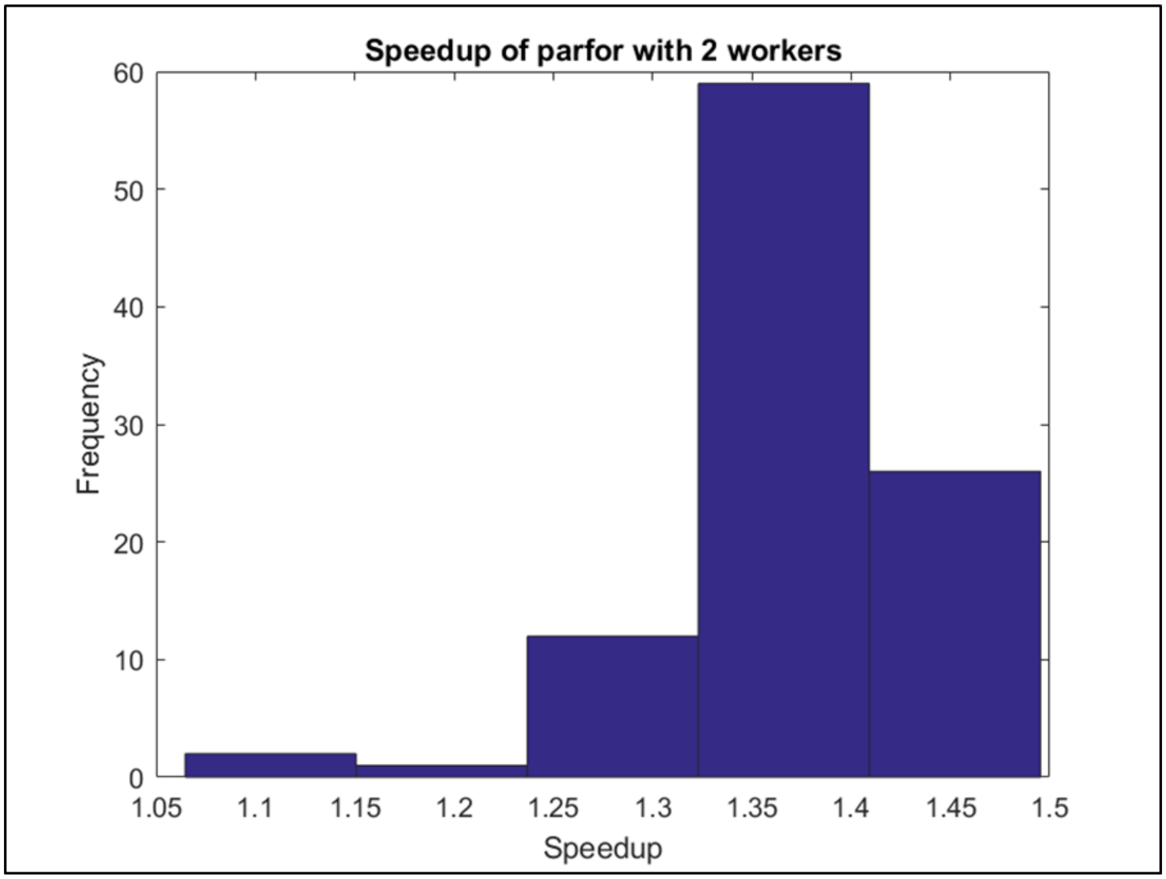


Figure 7. Blackjack Performance Using "Parfor" (Double Node)

Next, the same benchmark tests were run using the MATLAB Parallel Cloud with 16 worker nodes. Characterized by Gustafson-Barsis' Law, the ~2x speed improvement determined in the prior section with just 2 worker nodes, it should be expected that a ~16x improvement should be seen with 16 worker nodes.

The Blackjack benchmark test was run on the MATLAB Parallel Cloud and there was a much wider distribution of speed up values, ranging from ~5x→~20x with the highest frequency in the ~15x range. This is aligned with the expected 16x improvement over the single node execution times (Figure 8).

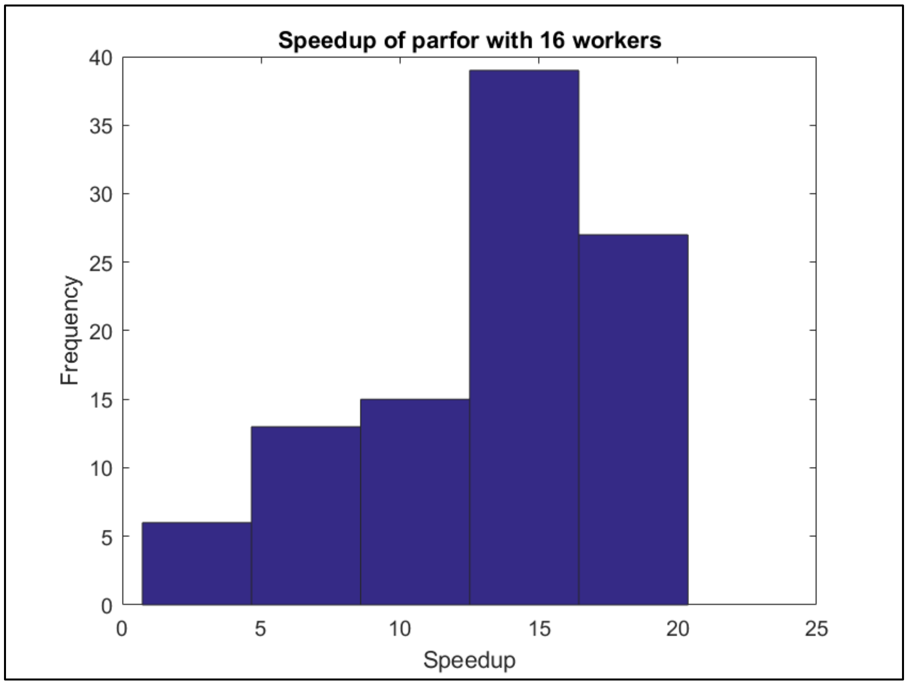


Figure 8. Blackjack Performance Using “Parfor” (16 Nodes)

3. Simulink Calls

The “parfor” construct can also be used to improve the execution times when making multiple calls to Simulink models. For example, the ubiquitious Simulink Proportional + Velocity Feedback loop model in Figure 9 can be invoked within a “parfor” loop.

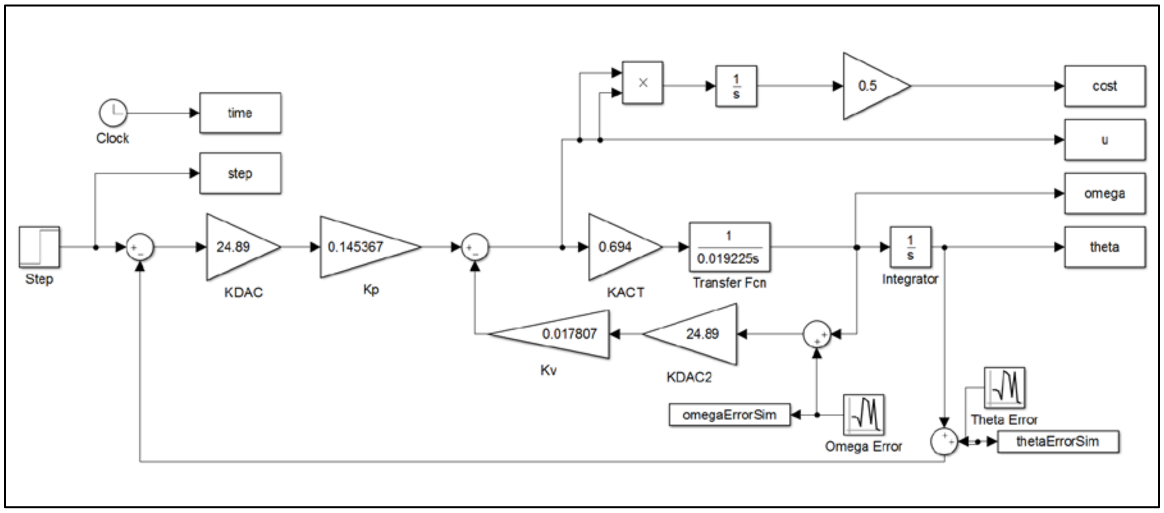


Figure 9. P+V Feedback Simulink Model

To benchmark the performance of a basic Simulink model, a series of loops with increasing numbers of runs (2,4,6,8,…256) was performed using the “for” loop. The same test was run using the “parfor” construct and the results of the two are shown in Figure 10.

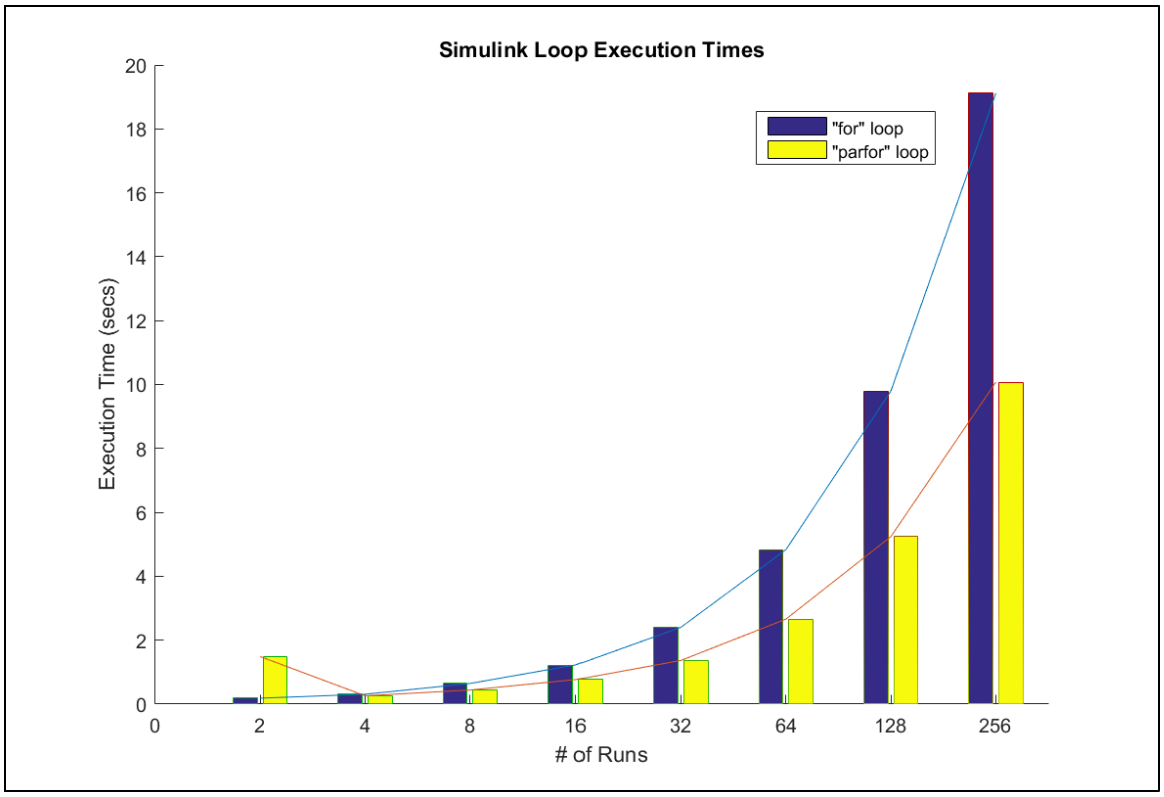


Figure 10. Execution Time of Simulink Runs (Single Node)

With the exception of the first run, execution times improved dramatically using the “parfor” construct and only continued to improve with more runs. In the final run (# of Runs = 256), the execution time was almost cut in half. In Figure 10, the first run (# of Runs =2) had a significantly larger execution time using the “parfor” construct. This is due to the initial orchestration required by the “parfor” construct and should be accounted for when deciding whether or not to design tools using the construct.

When run on the 16-node cluster, the Simulink model using a “for” loop averaged ~0.08 seconds per model run. This is similar to the performance of the “for” loop on the local cluster, with an average of ~0.075 seconds. These results are aligned with Gustafson-Barsis’ Law of distributed processing.

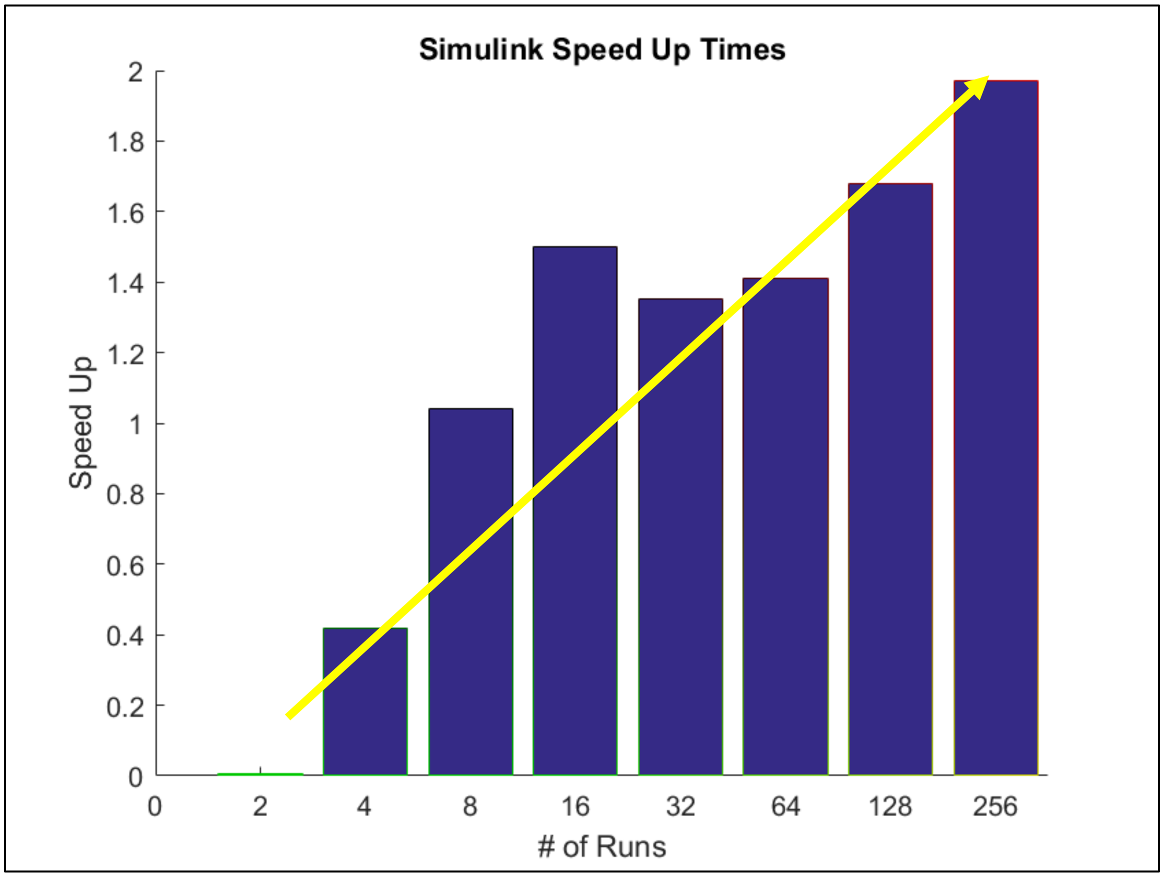


Figure 11. Speed up for Simulink Models Using Local Cluster

When using a larger cluster (>16 runs), the average runtime using the “parfor” construct is significantly lower than the runtimes of the regular “for” loop (Figure 12). When only performing a small number of iterations (<16 runs), however, the “for” loop is more efficient. This is, again, due to overhead orchestration associated with the “parfor” construct.

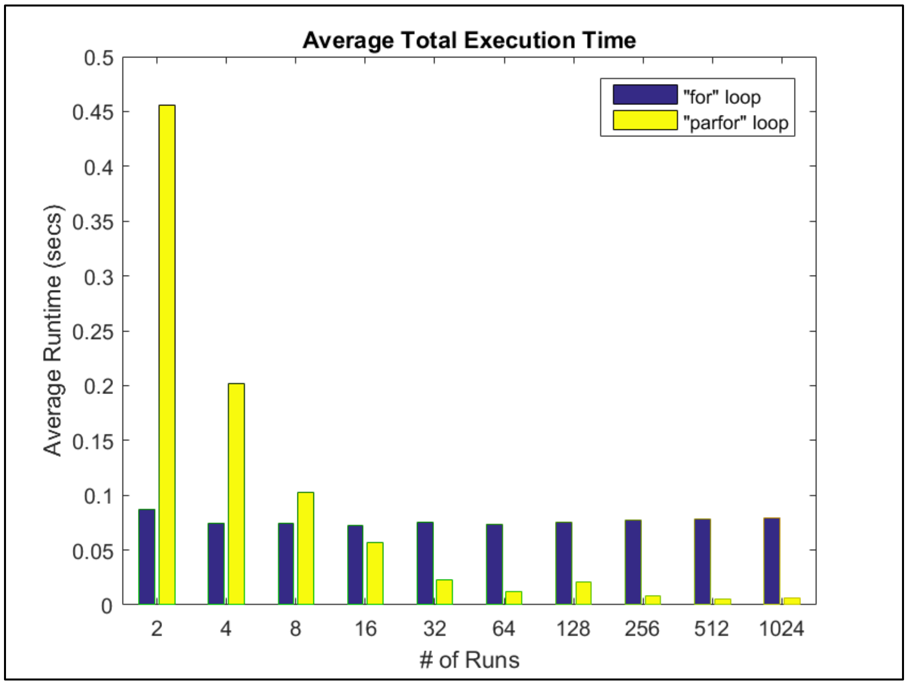


Figure 12. Average “for” versus “parfor” Loop Runtimes (16 Workers)

When looking at this data together on one graph (Figure 13), the speedup associated with the larger nodes aligns with Amdahl’s Law (Equation 3.7), although it may not be entirely predictable based solely on the number of workers. This variation is, again, associated with the overhead orchestration associated with worker nodes.

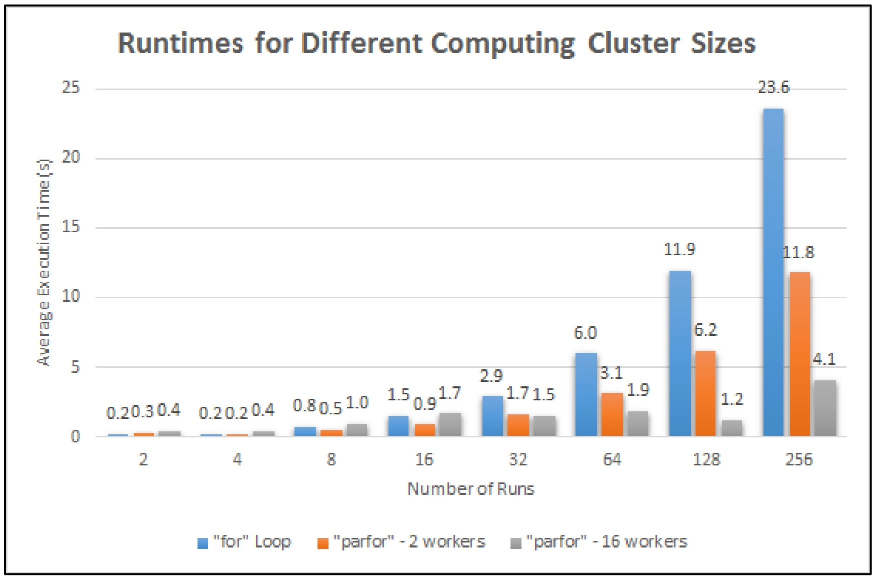


Figure 13. Runtimes for Different Computing Cluster Sizes

B. SAR MISSION DATA PROCESSING

The proliferation of overhead remote sensing has driven a need for innovative architectures to process their data. Raster data files, such as the data generated from a Synthetic Aperture RADAR sensor, are good candidates for integration into a distributed processing environment, as the files can be large and require complex operations for exploitation.

1. Brief History of SAR

Radar was developed in World War II and was designed for tracking of ships and aircraft. There were two key radar breakthroughs that enabled the generation of two-dimensional image of targets. First, in 1947, British engineer Dennis Gabor developed the principles of holography, which would be the early groundwork for waveform reconstruction theory [20]. Secondly, in 1951, Carl Wiley from Goodyear Aerospace combined the Doppler shift information from SAR returns and Gabor’s waveform reconstruction theory.

In the 1970s, radar imaging came into commercial use after the military SAR technology was released to the public [21]. Since then, commercial space-borne SAR platforms such as TerraSAR-X and RADARSAT have contributed to environmental planning, natural resource exploration, regional and urban development, catastrophe response and relief, and defense [22].

2. SAR Basics

Radar systems are an active remote sensing system in that it requires its own energy source to illuminate the target [23]. A radar system has three main functions:

Transmit a microwave signal toward a scene

Receive a portion of the transmitted signal as backscatter

Observe the strength and time delay of the returned signal

When the microwave signal hits the target, it is scattered in all directions with some of the signal returning back to the antenna. The amplitude of the returned signal depends on the irregularities in the target's surface (Figure 14).

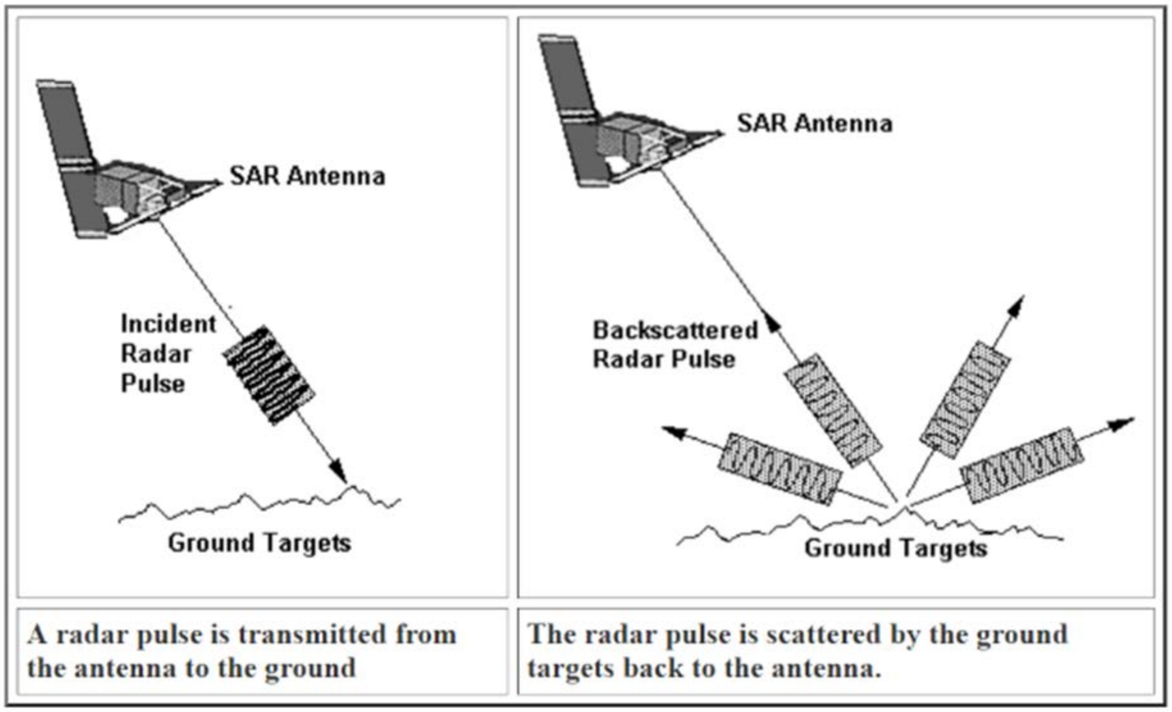

SAR Antenna
Incident
Radar
Pulse
Ground Targets
A radar pulse is transmitted from the antenna to the ground
SAR Antenna
Backscattered
Radar Pulse
Ground Targets
The radar pulse is scattered by the ground targets back to the antenna.

Figure 14. Transmit/Receive of a Spaceborne SAR Platform. Source: [23].

3. SAR Mission Data Processing

SAR data is comprised of large, multi-dimensional arrays of phase and amplitude data that require a series of complex transformations (e.g., digital filtering, domain transformations). The 2-Dimensional Fast Fourier Transform (2D FFT) is the most computationally complex algorithm required for SAR data processing. The original 2D data is split into rows and an FFT is performed on each row. The new data is then shuffled into columns and another FFT is performed on each column. Another factor that makes SAR data processing difficult is the fact that the data must be viewed as a contiguous block (see II.A.1.c). Large data files (100s of GBs) may exceed the computing resource available to most people. Techniques exist to split large files into smaller, more manageable, files; however, there are several negative side effects that impact the integrity of the data. This issue can be resolved using modern computing architectures and frameworks such as Apache Spark.

a. SAR Processing Using MATLAB/Simulink Toolbox

MATLAB provides a simplified SAR processing chain based on Massachusetts Institute of Technology (MIT) Lincoln Laboratory’s High-Performance Embedded Computing (HPEC) Challenge benchmark. The model uses simulated (however, realistic) SAR data, representing a 6x8 grid of reflectors placed on the ground that have been imaged by an aircraft flying directly overhead. The unprocessed SAR data does not produce any patterns that allow the human eye to infer what is being viewed and requires several steps of processing to produce an image that provides recognizable context (Figure 15).

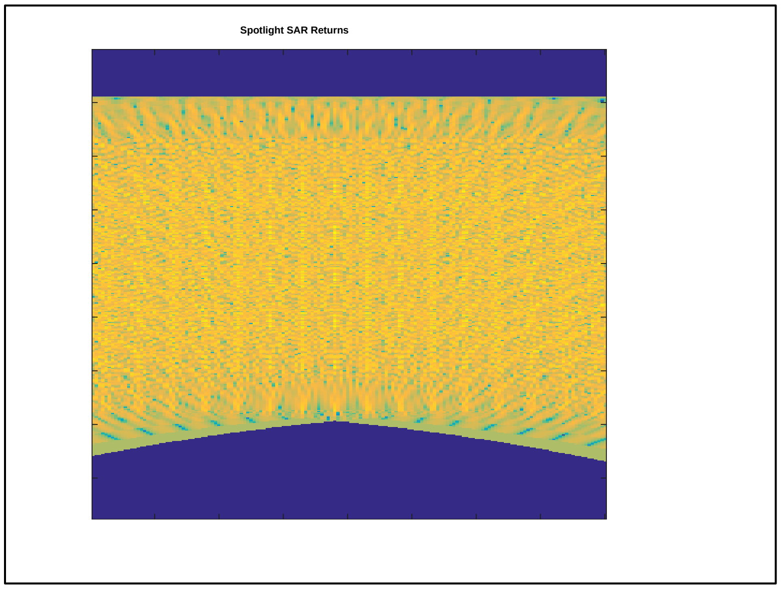


Figure 15. Raw SAR Data

In order to process the data to a viewable product, a series of operations must be chained together and performed on the data. There are many products that can be derived from SAR data and each product has several variations of the necessary workflow. For this model, a simplified version of the processing workflow is used containing the follow steps: a) Digital filtering and spotlight SAR processing, b) Two-dimensional matched filtering, and c) Polar to rectangular interpolation (Figure 16).

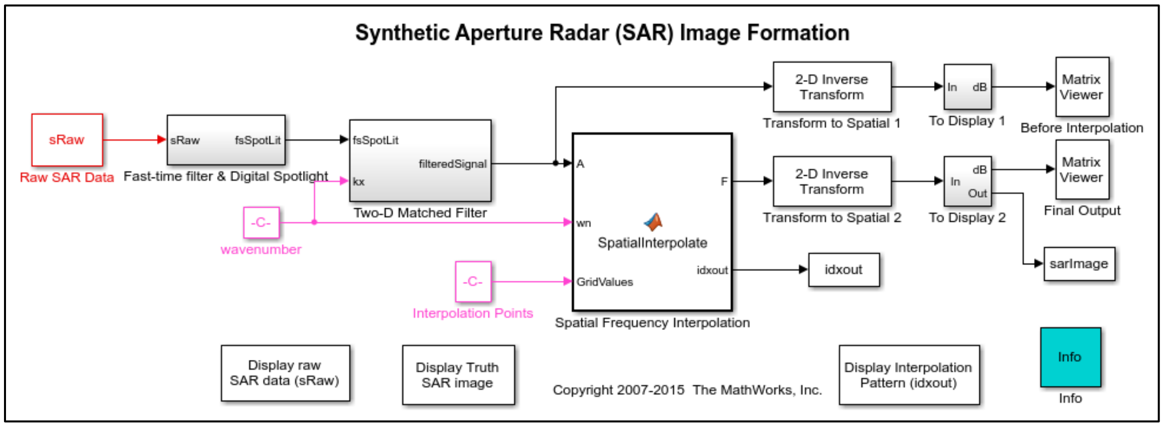


Figure 16. SAR Image Formation Simulink Model. Source: [15].

(1) Step 1: Digital Filtering and Spotlight SAR Processing

Three operations are performed in this step, two of which require the 2D FFT operation: 1) A Fast-Fourier Transformation (FFT) converts the return from each pulse, converts from the time domain into the frequency domain, and convolves with the expected return 2) A digital spotlighting step then focuses the return in the cross-range, and 3) the cross-range resolution is increased using a series of FFTs known as Bandwidth Expnsion [15].

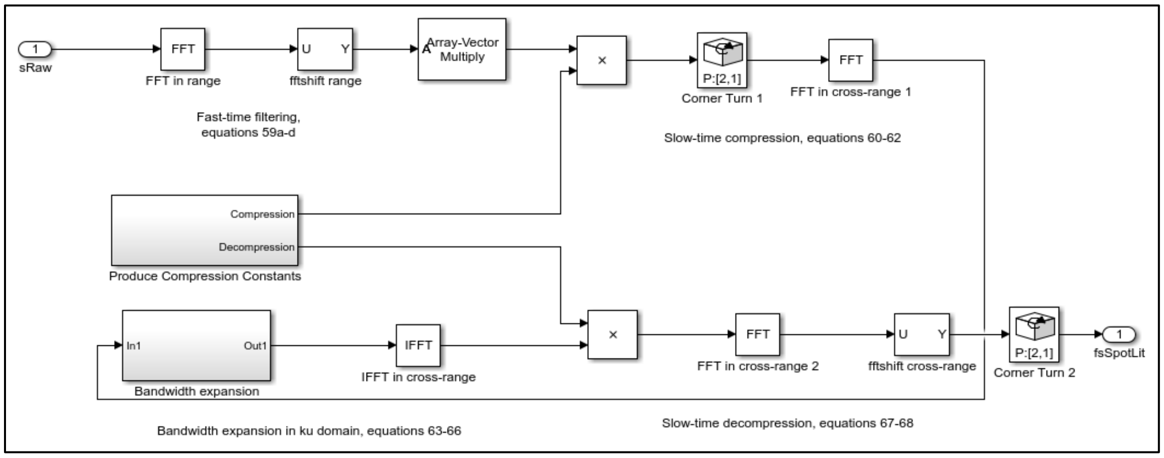


Figure 17. Digital Filtering and Spotlight SAR Processing. Source: [15].

(2) Step 2: Two-Dimensional Matched Filtering

The second step takes the output from Step 1 and convolves it with the impulse response of an ideal return. Since the data is in the frequency domain, this operation is done by multiplying the two together (See Figure 18) [15].

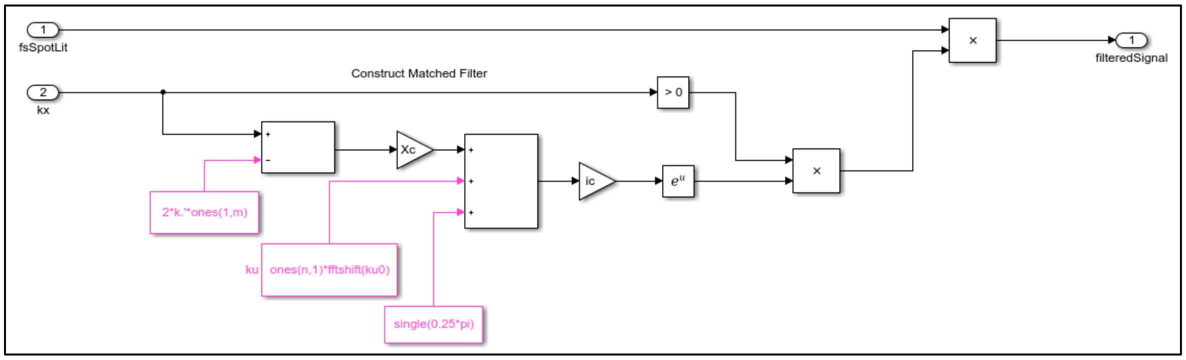


Figure 18. Two-Dimensional Matched Filtering. Source: [15].

(3) Step 3: Polar-to-Rectangular Interpolation

The third step performs a series of operations on the output of Step 2 to increase the range resolution of the data and transform the data back to the spatial domain. This includes a two-dimensional, inverse FFT to convert back to the spatial domain [15].

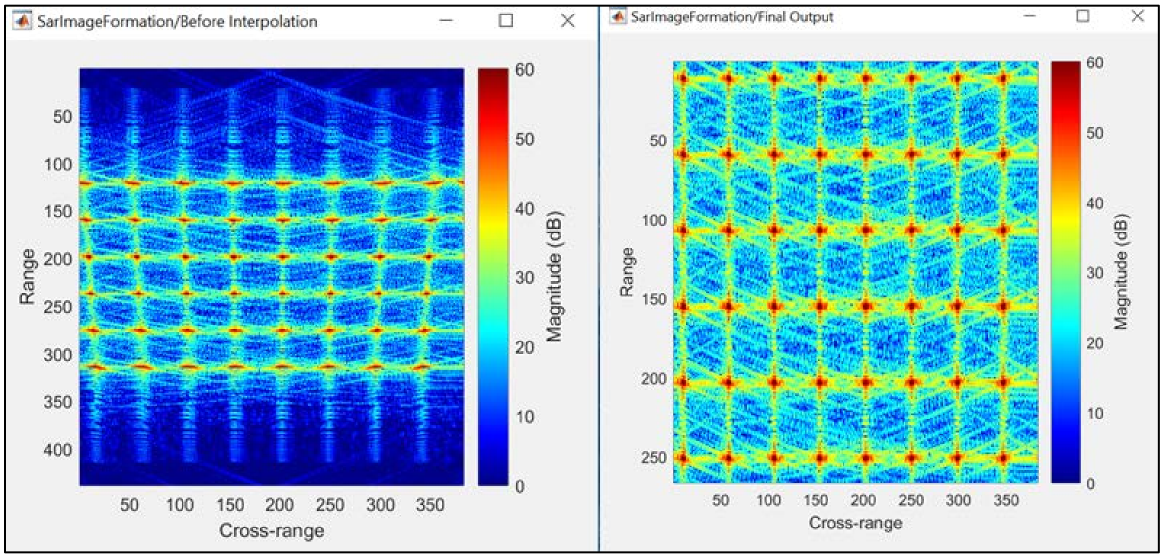

SarImageFormation/Before Interpolation
Range
Cross-range
Magnitude (dB)
SarImageFormation/Final Output
Range
Cross-range
Magnitude (dB)

Figure 19. SAR Image before/after Interpolation. Source: [15].

b. SAR Sensor Data Processing in Apache Spark

To address the challenges of processing large SAR data, a framework using Apache Spark on a scalable computing resource (such as Amazon Web Services) is a highly effective solution. Apache Spark is designed to process structured data such as computer log files and social media streams and distribute them into Resilient Distributed Datasets (RDDs) [24]. The algorithms to process SAR datasets require contextual information, awareness of surrounding elements of an array and the ability to efficiently reshape the 2D arrays of data. This can be done by dividing the arrays into a collection of RDDs, with each element containing a sub-block of the total array along with the context metadata to describe the array context. This metadata is based on the Department of Energy's Distributed Array Protocol, which defines the overlap with neighboring data blocks [25].

To demonstrate if Apache Spark could be used to efficiently process SAR data, the 2D FFT operation was first used to benchmark performance. A 2D FFT operation was performed against a variety of different sized 2D arrays representing structured SAR data. Resources of different computational capabilities were rented on Amazon Web Services (AWS) and the 2D FFT was performed on each of the instances [25]. Figure 20 shows the execution times of these runs.

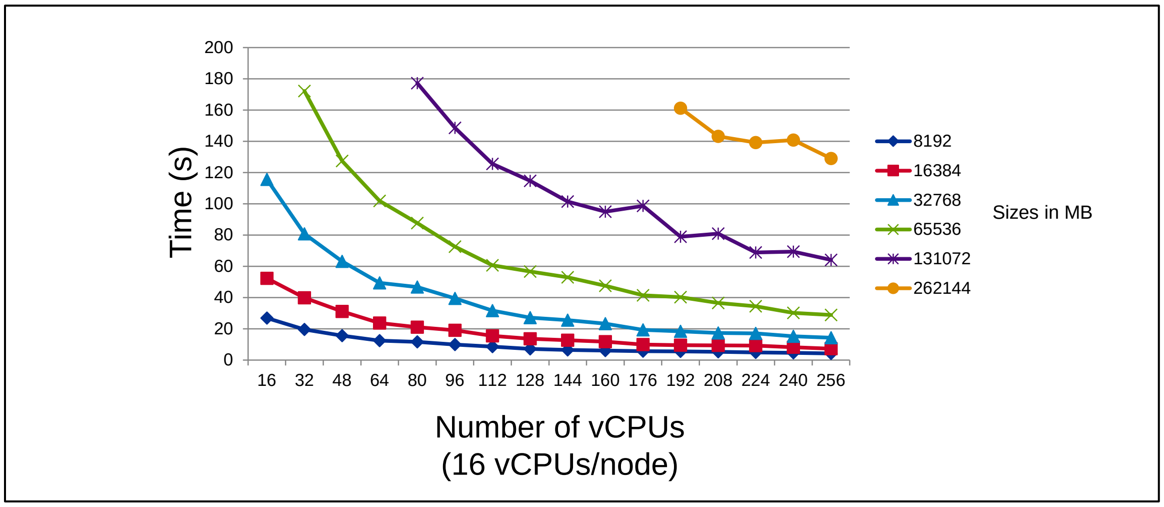


Figure 20. 2D FFT Performance on AWS Compute Cloud. Source: [25].

In Figure 20, each color-coded set of plotted data represents a data file of a given size (shown in legend). As expected, the processing time required for each dataset decreased as the number of vCPUs available increased. Consistent with Amdahl’s law, eventually, there is a point the graph becomes asymptotic and increasing the number of vCPUs does not reduce processing time, due to the serial work associated with the SAR processing. However, since the 2D FFT is highly leveraged in SAR processing, the work performed in parallel is reduced dramatically and could be optimized by applying enough vCPUs.

Now that the 2D FFT operation has been proven to work in a distributed manner, the full set of image formation operations was run against the SAR data. A finished SAR product would help validate the feasibility of SAR processing using Apache Spark. Using the operations in Figure 21, a full SAR dataset from AFRL’s GOTCHA challenge [26] was processed from raw data to a finished product using Apache Spark on a multi-node cluster.

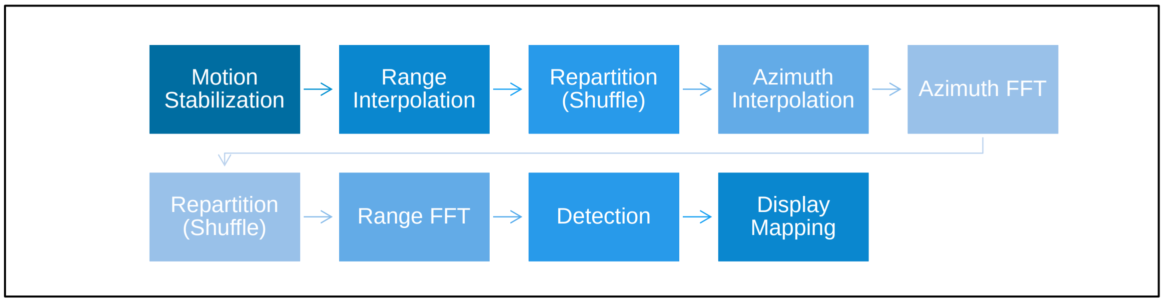


Figure 21. SAR Image Formation Chain. Source: [25].

The finished SAR product is shown in Figure 22. The image shows the Air Force Research Laboratory facility in Dayton, OH and key features of the scene (roads, buildings, vegetation) can easily interpretable by an analyst.

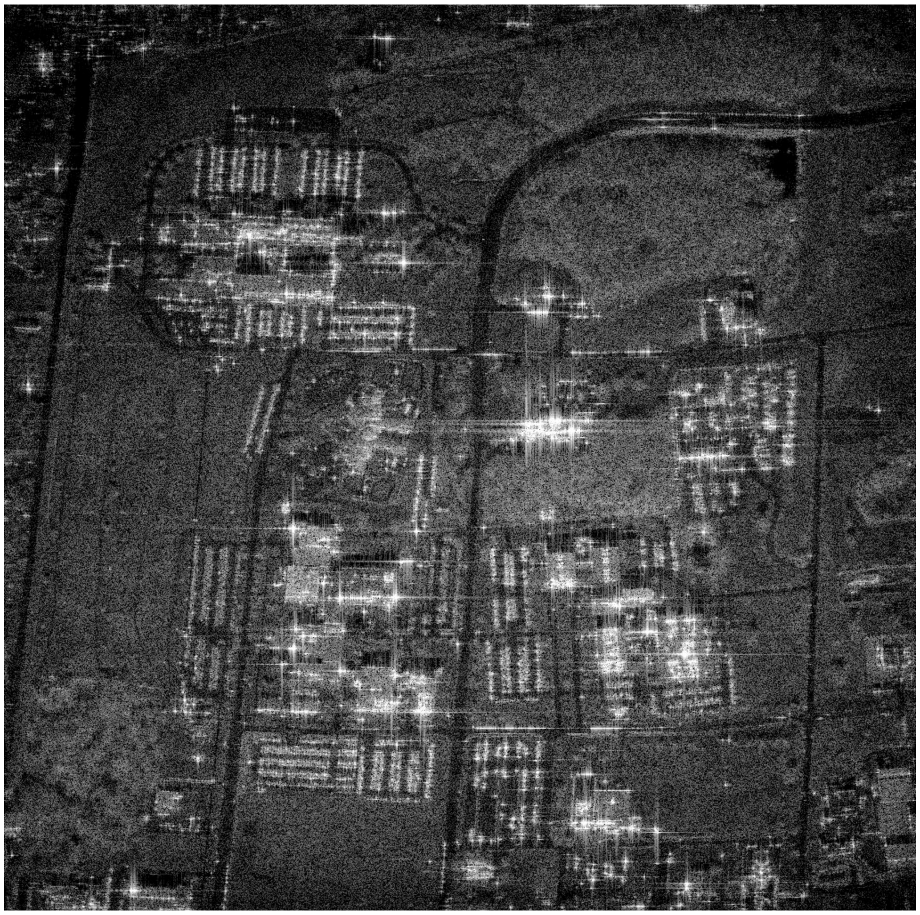


Figure 22. Finished SAR Product Using AFRL's GOTCHA Dataset

C. HYPERSPECTRAL MISSION DATA PROCESSING

Hyper-spectral datasets also consist of large spatial raster sensor data, but in this case, the dataset is three dimensional. Like SAR, hyper-spectral data has a two-dimensional spatial component; however, the hyper-spectral sensor typically collects the spatial data across multiple wavelengths, adding a third dimension to the dataset. This additional dimension adds another level of complexity when processing the data. Certain operations, such as k-means algorithms are of polynomial complexity ("O(n^3)"). To effectively process this data, use of modern computing architectures is not only an enhancement, it is a necessity if the finished products are needed in near-real time timelines.

1. Brief History of Hyperspectral Sensor Data

Introduced in the mid-80s, hyperspectral remote sensing, also known as “Imaging Spectroscopy,” is a relatively new technology. This platform is primarily used for detection and identification of minerals, vegetation and man-made materials [27]. Hyperspectral sensors detection individual absorption features associated with specific chemical nods in a material. Commercial hyperspectral sensors are currently hosted on airborne platforms and have recently begun transitioning to space-based platforms.

2. Hyperspectral Basics

Imaging spectroscopy has been in use by scientists and chemists to identify materials based on their composition [27]. Spectroscopy detects individual absorption features due to the specific bonds in solids, liquids and gasses. Advancements in the technology in the last few decades have allowed geologists to use the technology for the mapping of minerals on Earth.

Hyperspectral remote sensing combines imaging and spectroscopy into one platform [28]. The Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) is an optical sensor that provides imagery in 224 contiguous spectral bands containing wavelengths between 400 to 2500 nanometers [28]. The sensor is flown on an airborne platform and has been used to image areas over North America, Europe and South America. The main objective of AVRIS is identification, measurement and monitoring of constituents in the Earth’s surface and atmosphere using molecular absorption and particle scattering signatures [28]. An example of the “image cube” for an AVIRIS collect over Moffett Field is shown in Figure 23.

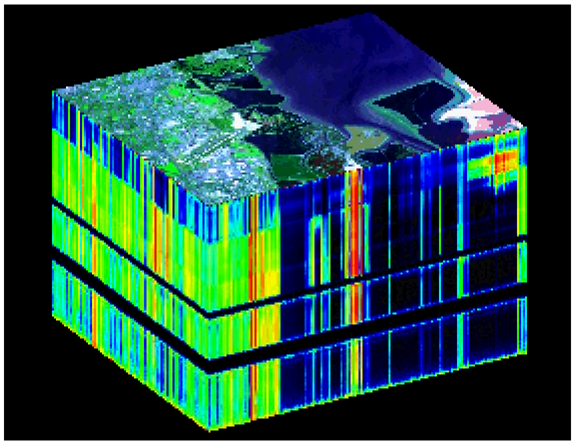


Figure 23. AVIRIS Hyperspectral Datacube of Moffet Field, California. Source [27].

The image cube demonstrates the potential volume of data returned by the platform. The top of the cube is a false-color image that accentuates the water features in the image [28]. The sides of the cubes are slices that show the edges of the top in all of the spectral channels. The tops of the cubes lie in the visible spectrum (wavelengths of 400 nanometers) and the bottoms reside in the infrared spectrum (~2500 nanometers). Hyperspectral is useful in that it can accentuate materials that may not be obvious to the naked eye. For example, in the top right corner of Figure 23, the return is a pink/red color due to the presence of red brine shrimp in the local evaporation ponds.

3. Hyperspectral Mission Data Processing

Due to the complexity of the dimensional nature of hyperspectral datasets, the techniques to process them are equally as computationally intense. In order to address this computing complexity, operations involving machine learning are required to process the datasets in a timely manner. Using unsupervised learning techniques such as "k-means clustering" can be utilized to reduce the problems from an "O(n^3)" complexity to a linear "O(n)".

a. *Hyperspectral Sensor Data Using MATLAB*

Using the open-source MATLAB Hyperspectral Toolbox (MHT), hyperspectral sensors can be processed relatively easily [16]. Another AVIRIS collection (“f970620t01p02_r03”) over the same region in 1997 is used to demonstrate capabilities provided in the MHT [28]. Figure 24 shows a grayscale representation of the region.

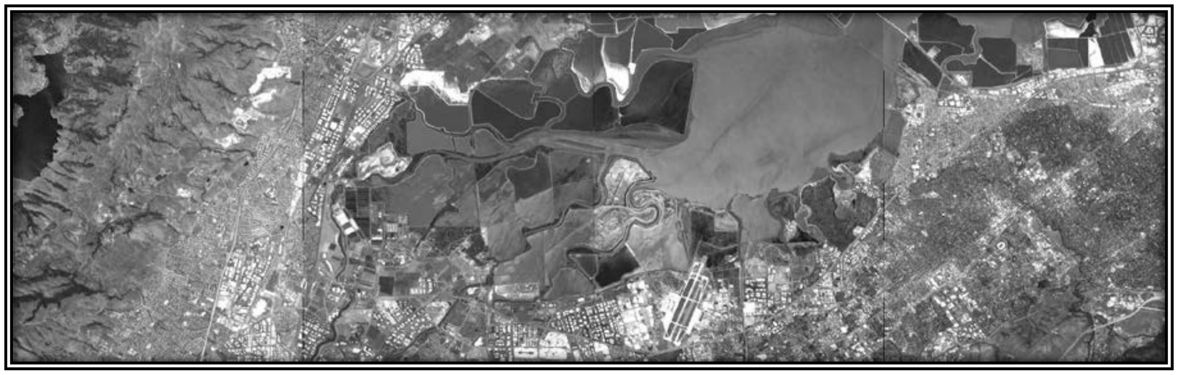


Figure 24. Grayscale Image of Moffett Field Scene Using MHT

Each band of the collection could be extracted and reviewed, if desired. Figure 25 is the top-left section of Figure 24 that shows the return of an arbitrary band (132) in gray-scale.

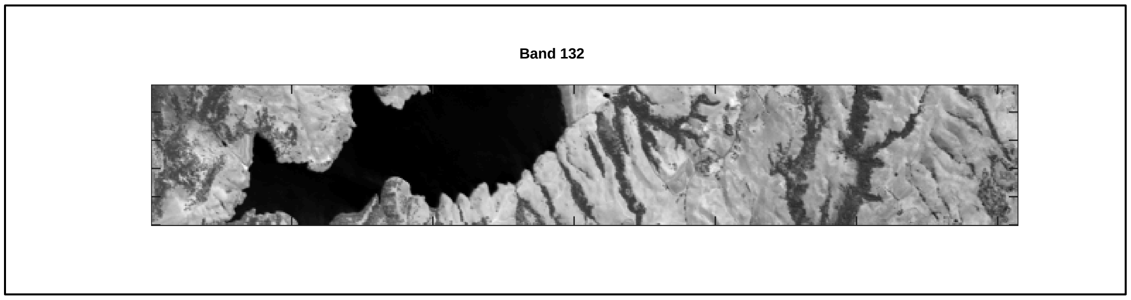


Figure 25. Band 132 of the 1997 Moffett Field Collect

Another algorithm is used to compute the scene's returns relative to the Normalized Difference Vegetation Index (NDVI) to determine where live green vegetation exists. A more positive value represents a presence of vegetation while a more negative number represents the absence.

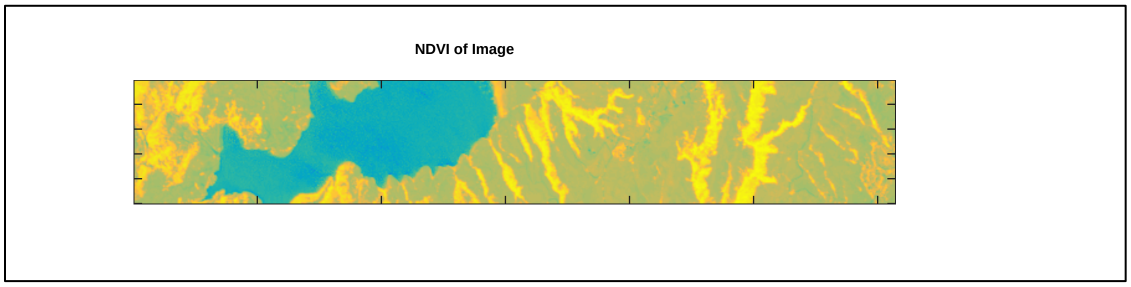


Figure 26. Normalized Difference Vegetation Index of 1997 Moffett Scene

Finally, to characterize the different materials in the scene, unsupervised learning algorithms are used on each band of data to calculate the fractional abundance of vegetation. The algorithm uses a "nearest neighbor" interpolation technique to estimate the abundance levels. The interpolation method is a relatively efficient algorithm computationally but not as accurate as bilinear or bi-cubic interpolation [29].

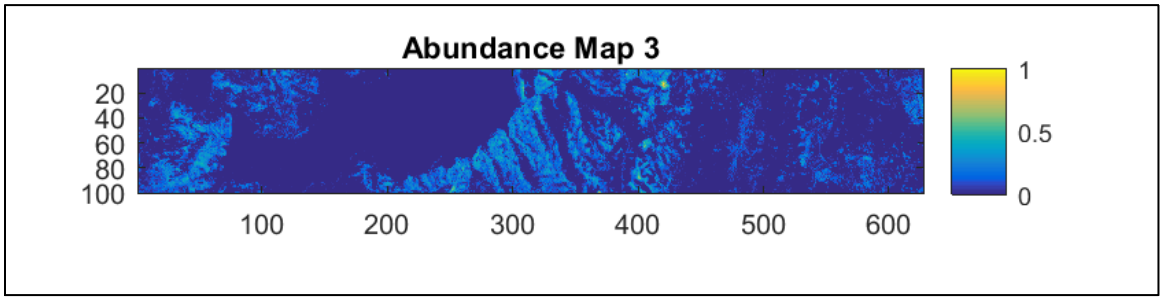


Figure 27. Abundance Map for Band 3 of 1997 Moffett Scene

b. Hyperspectral Sensor Data Using Apache Spark

The machine learning library (MLlib) that comes with Apache Spark provides developers with another method of the functions necessary to process the complex three-dimensional (3D) data arrays associated with hyperspectral data. Clustering is an unsupervised learning problem whereby subsets of entities are grouped with one another based on some form of similarity. The “k-means” algorithms is one of the more commonly used clustering algorithms used by machine learning workflows to associate data points into a predefined number of clusters [30].

Using the MLib machine learning library in Apache Spark, large 3D hyperspectral data cubes can be ingested into a virtual contiguous memory as a resilient distributed dataset. This allows the k-means algorithm to be performed on each band of the data in parallel.

As an example, a 300 row x 1300 column x 181 spectral band collect was taken over the Washington DC area. A k-means algorithm that was adapted to work with Apache Spark was applied to the data to bin their values. The resultant 2D color-coded map was generated, shown in Figure 28, along with a screen capture from Google Earth of the same scene, shown in

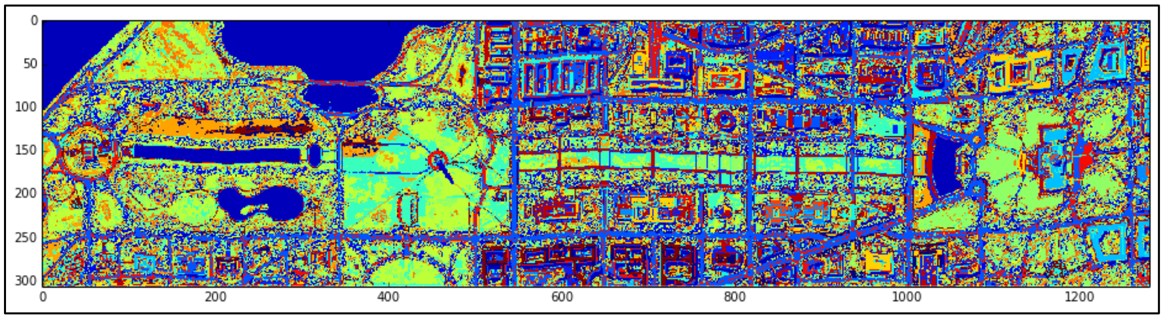


Figure 28. Two-Dimensional Map Derived from Hyperspectral Data Using Apache Spark

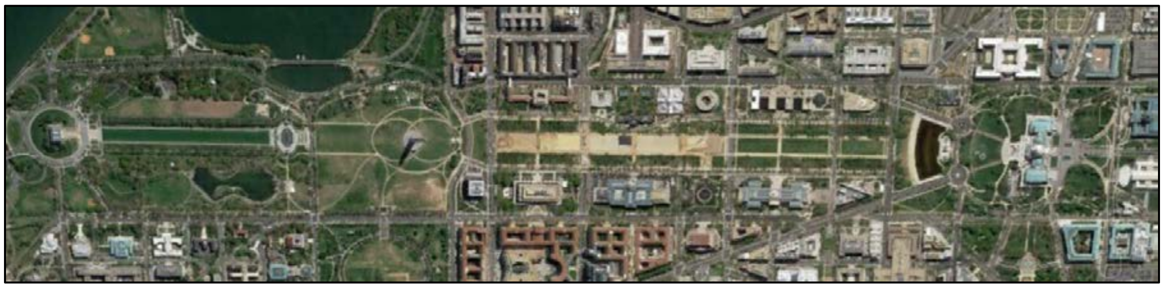


Figure 29. Google Earth Image of National Mall, Washington, DC. Adapted from [31].

The colors in Figure 28 represent different spectral returns. Each pixel in the image was analyzed and binned according to their characteristics. For example, pixels characterized as water appear (coincidentally) as blue in Figure 28. Several other features are highlighted using this method, including distressed foliage around the National Monument and patinated rooftops along the National Mall.

Apache Spark’s MLlib library proved to be a viable tool for processing the complex hyperspectral data. The features extracted through the k-means clustering algorithm increases the intelligence value of the hyperspectral sensor data.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

The work performed in this thesis validated that modern computing architecture and frameworks provide a rich set of tools to help data scientists and analysts navigate the challenges brought on by big data. MATLAB, popular in both academia and industry, provides several toolboxes to assist in parallel processing and simplification of normally complex calculations using machine learning. Additionally, third-party developers can assemble libraries of their specialized code and make them available to others through public repositories, as is the case with the MATLAB Hyperspectral Toolbox. Free, commercial-off-the-shelf (COTS) products, such as Apache Spark, provide a highly viable alternative to their expensive, specialized counterparts. In some cases, COTS products that are developed with distributed processing principles in mind provide elegant solutions to complex computing problems. By providing a framework that handles the orchestration of the distributed processing, data scientists and analysts can focus on the domain-specific areas of expertise. When these tools are run on scalable, on-demand computing resources, their performance can be improved dramatically by dividing the work among multiple worker nodes. From a cost perspective, savings can be realized through 1) on-demand rental of virtual computing resources, 2) horizontally scalable architectures all for efficient utilization of resources and 3) Free, open-source applications.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

A. BASIC “FOR” VERSUS “PARFOR” LOOP

```
function parforExamples(closePoolFlag)
close all; clear; clc
LoopNum = 50;
time = [1:1:LoopNum];
for index2 = 1:LoopNum
    clear y1;
    tic
    for x = 1:360
        y1(x) = cosd(x);
        %fprintf ('Executing thread: %d\n',x);
    end
    t1 = toc;
    time1(index2) = t1;
end
fprintf ('Average run time for "for" loop: %0.4f\n',mean(time1));
fprintf ('Total time to run %d times: %0.5f seconds\n',LoopNum,
LoopNum*mean(time1));
fprintf ('=========================================================\n')
figure ('Name','"for" loop example')
bar(y1)
ylabel('f(x)= cosd(x)')                    % label y-axis
xlabel('x (degs)')                         % label x-axis
title('Basic "for" Loop Demonstration')        % figure title
xlim([0 360])% xlimits 0->360

 poolobj = gcp('nocreate'); % If no pool, do not create new one.
if isempty(poolobj)
    parpool;
else
    poolsize = poolobj.NumWorkers;
end
for index2 = 1:LoopNum;
    clear y2;
    tic
    parfor x = 1:360
        y2(x) = cosd(x);
    end
    t2 = toc;
    time2(index2) = t2;
end
```

```
fprintf ('Average run time for "parfor" loop: %0.4f\n',mean(time2));
fprintf ('Total time to run %d times: %0.5f seconds\n',LoopNum,
LoopNum*mean(time2));
fprintf ('==========================================================\n')
figure ('Name','"parfor" loop example')
bar(y2)
ylabel('f(x)= cosd(x)')          % label y-axis
xlabel('x (degs)')               % label x-axis
xlim([0 360]);
title('Basic "parfor" Loop Demonstration')
figure; hold on;
title('Execution Times')
[AX1,H1,H2] = plotyy(time,time2,time,time1,'bar','bar');
legend ('"parfor" loop','"for" loop');
xlim ([0.5,50.5]);
H2.FaceColor = 'r';
set(AX1(1),'Xlim',[0 50]);
set(AX1(2),'Xlim',[0 50]);
xlabel('Time (sec)')
ylabel(AX1(1),'"parfor" loop execution time (secs)') % left y-axis
ylabel(AX1(2),'"for" loop execution time (secs)') % right y-axis

hold off;
% Print final stats
fprintf ('"for" loop Time: %0.4f\n',mean(time1));
fprintf ('"parfor" loop time: %0.4f\n',mean(time2));
fprintf ('Time difference: %0.4f\n',mean(time1)-mean(time2));
fprintf ('Percent improvement: %0.0f%%\n',(mean(time1)-
mean(time2))*100/mean(time1));
fprintf ('==========================================================\n')
```

B. BLACKJACK SIMULATOR

Source Code Available at:
https://www.mathworks.com/help/distcomp/examples/simple-benchmarking-of-parfor-using-blackjack.html

C. HYPERSPECTRAL COMPUTING TOOLKIT

Source Code available at:
https://github.com/isaacgerg/matlabHyperspectralToolbox

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] A. Carbonell, "GEOINT big data: Evolving how we understand the world through research and development" presented at Big Data Analytics and Applications for Defense, Intelligence and Homeland Security Symposium, Washington, DC, 2013.

[2] K. Quinn. (2013, Oct. 9). IC ITE moves forward. [Online]. Available: http://trajectorymagazine.com/got-geoint/item/1570-ic-ite-moves-forward.html

[3] B. J. Sapp, "Innovation architect: Increasing space persistence to improve performance," in *Geospatial Intelligence Forum,* March/April 2015, pp. 17-21.

[4] MathWorks. (n.d.). Key problems addressed by parallel computing. [Online]. Available: http://www.mathworks.com/help/distcomp/key-problems-addressed-by-parallel-computing.html?refresh=true. Accessed Nov. 18, 2015.

[5] P. Mell and T. Grace. (2015, Sept.). The NIST definition of cloud computing. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[6] V. Kundra, "Federal cloud computing strategy," The White House, Washington, DC, 2011.

[7] M. McCool, J. Reinders and A. Robison. (2013, Oct. 22). Amdahl's law vs. Gustafson-Barsis' law. [Online]. Available: http://www.drdobbs.com/parallel/amdahls-law-vs-gustafson-barsis-law/240162980. Accessed Nov. 13, 2016.

[8] Gene Amdahl. (2016, Nov. 10). *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Gene_Amdahl

[9] M.M. Falatah and O.A. Batarfi, "Cloud Scalability Considerations," *International Journal of Computer Science & Engineering Survey,* vol. 5, no. 4, 2014.

[10] R. Bell. (n.d). A beginner's guide to big O notation. [Online]. Available: https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/. Accessed Nov. 186 2016.

[11] C. Moler. (2004). The origins of MATLAB. [Online]. Available: http://www.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html. Accessed Nov. 20, 2015.

[12] C. Moler. (2007, June). Parallel MATLAB: Multiple processors and multiple cores. *The Mathworks News&Notes*. [Online]. Available: http://www.math works.com/tagteam/42682_91467v00_NNR_Cleve_US.pdf

[13] MathWorks, Inc. (2015). MATLAB parallel cloud. [Online]. Available: http://www.mathworks.com/products/parallel-computing/matlab-parallel-cloud/. Accessed Nov. 29, 2015.

[14] Mathworks, Inc. (2016). MATLAB distributed computing server for Amazon EC2. [Online]. Available: https://www.mathworks.com/products/parallel-computing/parallel-computing-on-the-cloud/distriben-ec2.html. Accessed Nov. 24, 2016.

[15] MathWorks. (2016). Sythetic Aperture Radar (SAR) Processing - MATLAB & Simulink Example. [Online]. Available: https://www.mathworks.com/help/dsp/examples/synthetic-aperture-radar-sar-processing.html

[16] I. Gerg. (2004, Jan. 30). Matlab hyperspectral toolbox. [Online]. Available: https://sourceforge.net/projects/matlabhyperspec/. Accessed Nov. 20, 2016.

[17] MathWorks. (2016). MathWorks pricing and licensing. [Online]. Available: https://www.mathworks.com/pricing-licensing.html

[18] Apache. (2016). Apache Spark: Lighting fast cluster computing. [Online]. Available: http://spark.apache.org. Accessed Oct. 29, 2016.

[19] H. Karau, A. Konwinski, P. Wendell and M. Zaharia, *Learning Spark*, 1st ed. O'Reilly Media, Inc, 2015.

[20] A. Moreira. (2013, July 1). Synthetic Aperture Radar (SAR): Principles and applications. Harokapia University. [Online]. Available: https://earth.esa.int/documents/10174/642943/6-LTC2013-SAR-Moreira.pdf

[21] M. Schlutz, "Synthetic aperture radar imaging simulated in MATLAB," M.S. thesis, California Polytechnic State University, San Luis Obispo, CA, 2009.

[22] Airbus Defence and Space. (2016). Satellite image gallery. [Online]. Available: http://www.intelligence-airbusds.com/en/5751-image-detail?img=1707&search=gallery&market=0&world=0&sensor=26&continent=0&keyword=#.WClrz_krLb0. Accessed Nov. 14, 2016.

[23] Center for Remote Imaging, Sensing & Processing. (2001). Principles of remote sensing. [Online]. Available: http://www.crisp.nus.edu.sg/~research/tutorial/mw.htm

[24] J. Daily, B. Granger, R. Grant, M. Ragan-Kelley, M. Kness, K. Smith and B. Spotz. (2014, Apr. 7). Distributed array protocol documentation. [Online]. Available: http://distributed-array-protocol.readthedocs.io/en/rel-0.10.0/

[25] Boeing Mission Framework & Analytics Team, "Mission processing framework (MPF) technical information," unpublished.

[26] United States Air Force. (n.d). GOTCHA volumetric SAR data set overview [Online]. Available: https://www.sdms.afrl.af.mil/index.php?collection=gotcha. Accessed Nov. 20, 2016.

[27] University of Texas at Austin, Center for Space Research. (n.d.). Hyperspectral remote system. [Online]. Available: http://www.csr.utexas.edu/projects/rs/hrs/hyper.html. Accessed Nov. 14, 2016.

[28] Jet Propulsion Laboratory. (2016, Sept. 01). AVIRIS - Airborne Visible Infrared Imaging Spectrometer. [Online]. Available: http://aviris.jpl.nasa.gov/. Accessed Nov. 18, 2016.

[29] Mathworks, Inc, (n.d.). Nearest neighbor, bilinear, and bicubic interpolation methods. [Online]. Available: https://www.mathworks.com/help/vision/ug/interpolation-methods.html. Accessed Nov. 20, 2016.

[30] Apache Spark (n.d.). Clustering - RDD-based API. [Online]. Available: http://spark.apache.org/docs/latest/mllib-clustering.html#k-means. Accessed Nov. 29, 2016.

[31] "Washington, DC." 38°53'22.56"N and 77° 1'58.27"W. Google Earth. Dec. 12, 2016. Feb 23, 2017.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California