

Master's Thesis:
Optimal Configuration of a Parallel Embedded System for
Synthetic Aperture Radar Processing

Jeffrey T. Muehring

Department of Computer Science
Texas Tech University

November 1997

Committee Members:

Dr. John K. Antonio (Chairperson)

Dr. W.J. Bryan Oldham

Dr. Milton L. Smith

ACKNOWLEDGMENTS

I would like to express my sincere respect and appreciation to my chairman, Dr. John K. Antonio, for his mentorship throughout the research process. He has been an invaluable instructor, advisor, and friend. I thank Dr. Milton L. Smith and Dr. W.J. Bryan Oldham for participating on my committee and for classes that I had the opportunity to take under them. Dr. Smith introduced me to mathematical programming and Dr. Oldham acquainted me with signal processing and the wonders of the Fourier Transform.

Completion of this thesis is due largely to the support of my coworkers in the High Performance Computing Lab. I thank Jack West, Nikhil Gupta, and Tim Osmulski for the countless hours of brainstorming and help in classes. In addition, a large portion of the chapter on the Mercury system is taken directly from Jack's thesis material.

This research was supported by Rome Laboratory under grant F30602-96-1-0098 and by the Defense Advanced Research Projects Agency (DARPA) under contract F30602-97-2-0297.

My faithful friends, Albert McSpadden and Norvell Spearman, have provided continual encouragement through my graduate work. I especially thank Albert for his aid in classes and Norvell for his help in \LaTeX .

Finally, I would like to pay tribute to some special people in my life. I thank

my loving wife Trina for her constant support and enthusiasm for whatever I am doing. This thesis would have greatly suffered without her careful editing and proofreading till the wee hours of the morning on many occasions. I thank my parents, Gene H. and Judy D. Muehring, and my sister, Deanna L. Muehring, for offering their love and sharing their wisdom with me throughout my life.

CONTENTS

ACKNOWLEDGMENTS	ii
ABSTRACT	vi
TABLES	vii
FIGURES	x
SYMBOLS	xi
ABBREVIATIONS	xiii
I INTRODUCTION	1
II PRINCIPLES OF SYNTHETIC APERTURE RADAR	4
2.1 Conventional Radar	5
2.2 Synthetic Aperture Radar	10
III THE MERCURY RACE SYSTEM	19
3.1 Mapping of SAR Processing onto the RACE System	28
3.2 Computational Framework	29
IV THE OPTIMIZATION PROBLEM	37
4.1 Mathematical Programming	38
4.2 Optimization Objectives	42
4.3 Hardware Configurability	43
4.3.1 Optimal Configuration Using Custom-Designed Boards	43
4.3.2 Optimal Configurations Using COTS	45
4.4 Architectural Models	46
4.4.1 Ideal Shared-Memory Model	46
4.4.2 CN-Constrained Model	47
4.5 Hardware Availability Constraints	49
4.6 Points of Reference: Nominal Configurations	50
4.7 Summary	51
V IDEAL SHARED-MEMORY MODEL	54
5.1 Minimization of Power	55
5.1.1 Optimal Mixed Card Type Configuration	56
5.1.2 Optimal Single Card Type Configuration	70
5.1.3 Nominal Mixed Card Type Configurations	74
5.1.4 Nominal Single Card Type Configurations	77
5.1.5 Summary of Power Minimization Models	79
5.2 Maximization of Velocity	79
5.2.1 Set Power with Variable Number of Cards	82
5.2.1.1 Optimal Mixed Card Type Configuration	83
5.2.1.2 Optimal Single Card Type Configuration	87
5.2.1.3 Nominal Mixed Card Type Configuration	89
5.2.1.4 Nominal Single Card Type Configuration	92

5.2.1.5	Comparison of Maximum Velocity Configurations	92
5.2.2	Configuration with Set Number of Cards	94
5.3	Minimization of Resolution	98
5.3.1	Optimal Mixed Card Type Configuration	101
5.3.2	Optimal Single Card Type Configuration	106
5.4	Conclusions	109
VI	CN-CONSTRAINED MODEL	112
6.1	Formulation	112
6.2	Computational Approach	120
6.3	Minimization of Power	127
6.3.1	Optimal Mixed Configuration	131
6.3.2	Nominal Mixed Configuration	141
6.3.3	Comparison of Optimal and Nominal Configurations	151
6.3.4	Effects of Integer Numbers of Cards	155
6.3.5	Comparison of CNCM and ISMM	162
6.4	Conclusions	163
VII	RANDOMLY-GENERATED SOLUTIONS	167
7.1	Solution Verification	167
7.2	Random Solutions as an Optimization Technique	172
VIII	CONCLUSIONS	180
	REFERENCES	186
	APPENDIX	188

ABSTRACT

Embedded systems often must adhere to strict size, weight and power (SWAP) constraints and yet provide tremendous computational throughput. Increasing the difficulty of this challenge, there is a trend to utilize commercial-off-the-shelf (COTS) components in the design of such systems to reduce both total cost and time to market. Employment of COTS components also promotes standardization and permits a more generalized approach to system evaluation and design than do systems designed at the application-specific-integrated-circuit (ASIC) level.

The computationally intensive application of synthetic aperture radar (SAR) is by nature a high-performance embedded application that lends itself to parallelization. Mercury Computer Systems' RACE multicomputer is the COTS computing platform under investigation. With the target software and hardware defined, a system performance model, in the context of SWAP, is developed based on mathematical programming. This work proposes an optimization technique using a combination of constrained nonlinear and integer programming.

TABLES

5.1	Comparison of minimum power configurations.	81
5.2	Comparison of maximum velocity configurations.	99
5.3	Comparison of set-hardware configurations for maximum velocity.	99
7.1	Summary of random solutions.	171
7.2	Comparison of optimal and random configuration power.	177
7.3	Ratios of random to optimal power.	177

FIGURES

2.1	Footprint of aerial radar.	6
2.2	Reference chirp starting at $t = 0$	8
2.3	Reference chirp centered at $t = 0$	9
2.4	Overlapping realbeam footprints.	11
2.5	Resultant narrow synthetic beams.	12
2.6	Example waveform of a convolution	14
3.1	The RACE multicomputer.	20
3.2	The RACEway six-port network chip.	22
3.3	The RACE multicomputer fat-tree interconnection network.	23
3.4	Message transfer between two CNs.	24
3.5	Parallelization of the matrix transpose operation.	30
3.6	Sectioned convolution.	32
5.1	Range FFT size for power minimization.	57
5.2	Range processor requirements.	57
5.3	Range memory requirements.	61
5.4	Optimal power consumption.	61
5.5	Optimal azimuth FFT size.	63
5.6	Optimal azimuth memory requirements.	63
5.7	Optimal azimuth processor requirements.	64
5.8	Optimal ratio of azimuth to range memory requirements.	64
5.9	Optimal ratio of azimuth to range processor requirements.	66
5.10	Optimal azimuth FFT kernel size.	66
5.11	Optimal azimuth section size.	68
5.12	Optimal ratio of azimuth section size to FFT size.	68
5.13	Optimal percentage of power usage by the S2T16B.	69
5.14	Optimal percentage of power usage by the S1D64B.	69
5.15	Optimal power consumption in S2T16B-only configuration.	71
5.16	Optimal power consumption in S1D64B-only configuration.	71
5.17	Power comparison of single card type configurations.	73
5.18	Binarized power comparison of single card type configurations.	73
5.19	Nominal power consumption.	76
5.20	Ratio of azimuth kernel size to optimal section size.	76
5.21	Comparison of nominal and optimal configurations.	78
5.22	Nominal power of S2T16B-only configuration.	78
5.23	Nominal power of S1D64B-only configuration.	80
5.24	Power comparison of six configurations at set velocity.	80
5.25	Power comparison of six configurations at set resolution.	81

5.26	Maximum velocity at fixed power and resolution.	85
5.27	FFT size of maximum velocity solutions.	85
5.28	Optimal section size for maximum velocity.	86
5.29	Optimal percentage power by S2T16B for maximum velocity.	86
5.30	Maximum velocity with S2T16B-only configuration.	88
5.31	Maximum velocity with S1D64B-only configuration.	88
5.32	Section size of S2T16B configuration.	90
5.33	Section size of S1D64B configuration.	90
5.34	Nominal velocity.	91
5.35	FFT size for nominal velocity.	91
5.36	Percent power by S2T16B for nominal velocity.	93
5.37	Nominal velocity with S2T16B-only configuration.	93
5.38	Nominal velocity with S1D64B-only configuration.	95
5.39	Comparison of velocities in fixed hardware configuration.	95
5.40	Optimal and nominal FFT and section sizes for the 5:5 system.	97
5.41	Optimal and nominal FFT and section sizes for the 7:2 system.	97
5.42	Initial minimum resolution solution.	103
5.43	Smoothed minimum resolution solution.	103
5.44	Optimal azimuth FFT size for minimum resolution.	105
5.45	Optimal azimuth section size for minimum resolution.	105
5.46	Azimuth kernel size for minimum resolution.	107
5.47	Initial resolution graph for S2T16B-only configuration.	107
5.48	Smoothed resolution graph for S2T16B-only configuration.	108
5.49	Initial resolution graph of S1D64B-only configuration.	108
5.50	Initial resolution graph of S1D64B-only configuration.	111
6.1	Minimal power for optimal mixed configuration.	133
6.2	System configurations for minimal power.	133
6.3	FFT size in optimal mixed configuration.	135
6.4	Section size in optimal mixed configuration.	135
6.5	Optimal ratio of kernel size to section size.	137
6.6	Azimuth memory requirements in optimal mixed configuration.	137
6.7	Azimuth processor requirements in optimal mixed configuration.	138
6.8	Optimal ratio of azimuth to range memory.	138
6.9	Optimal ratio of azimuth to range processors.	140
6.10	Optimal azimuth memory per processor in CNCM.	140
6.11	Optimal azimuth memory per processor in ISMM.	142
6.12	Optimal percentage power consumption by S2T16B.	142
6.13	Optimal percentage power consumption by S1D64B.	145
6.14	Nominal power in naive approach.	145
6.15	Configurations in naive approach.	146
6.16	Nominal azimuth memory per processor in naive approach.	146

6.17	Nominal power in sophisticated approach.	150
6.18	Configurations for the sophisticated approach.	150
6.19	Added processors in sophisticated approach.	152
6.20	Utilization in the sophisticated approach.	152
6.21	Ratio of naive to sophisticated approach power.	154
6.22	Ratio of optimal to naive nominal power.	154
6.23	Ratio of optimal to sophisticated nominal power.	159
6.24	Optimal power in integer card-number solution.	159
6.25	Ratio of integer card-number to real-value power.	161
6.26	Configurations of integer card-number solutions.	161
6.27	Ratio of CNCM to ISMM power.	166
7.1	Histogram of random solutions.	171
7.2	Optimal power by 100-sample random solution.	175
7.3	Optimal power by 1000-sample random solution.	175
7.4	Optimal power by 10000-sample random solution.	176
7.5	Optimal power by 100000-sample random solution.	176

SYMBOLS

A_{real}	length of antenna, i.e., the real aperture
A_{syn}	synthetic aperture
B	bandwidth
c	speed of light, i.e., 3×10^8 m/s
C_{max}	upper bound on C_1
C_1	number of S2T16Bs
C_2	number of S1D64Bs
C_T	daughtercard type
$ E $	number of evaluations
$ E_{\text{het}} $	number of evaluations in heterogeneous scenario
$ E_{\text{het,hom}} $	number of evaluations in heterogeneous-homogeneous scenario
$ E_{\text{hom}} $	number of evaluations in homogeneous scenario
F_a	azimuth FFT size
f_d	Doppler frequency
F_r	range FFT size
I	X or Y , i.e., either of two card configurations
I_a	number of azimuth processors on I
I_r	number of range processors on I
I_T	daughtercard type of I
k	integer constant primarily used as power of two in F_a
K_a	azimuth kernel size
K_r	range kernel size
M	total required memory
M_{CN}	memory per CN
M_{max}	maximum available memory per CN
M_a	required azimuth memory
M_d	memory per daughtercard
M_r	required range memory
N_d	number of different daughtercard types
N_I	number of required CNs of configuration I
N_X	number of required CNs of configuration X
N_Y	number of required CNs of configuration Y
P	total number of required processors
P_{act}	actual number of processors
P_{CN}	processors per CN
P_a	number of required azimuth processors
P_d	processors per daughtercard

P_r	number of required range processors
Q	sample rate
R	radar range
R_s	range swath
S_a	azimuth section size
S_r	range section size
t	time
T_e	elapsed time from transmission to reception of radar signal
U_p	processor utilization
v	velocity
X	card configuration in CNCM
X_a	number of azimuth processors on X
X_r	number of range processors on X
X_T	daughtercard type of X
Y	card configuration in CNCM
Y_a	number of azimuth processors on Y
Y_r	number of range processors on Y
Y_T	daughtercard type of Y
Z	objective function in optimization formulation
α_a	load unique to azimuth processing
α_r	load unique to range processing
β	power requirements per MB memory
δ	resolution (assuming equal range and azimuth resolutions)
δ_r	range resolution
δ_{real}	uncompressed azimuth resolution
κ	power requirements per processor
λ	wavelength
ϕ_a	azimuth load per sample
ϕ_r	range load per sample
Π	total power
Π_{CN}	power consumption per CN
Π_{prop}	proposed optimal power
Π_d	power consumption per daughtercard
π_T	total processors across all CN types
τ_c	compressed pulse width
τ_p	pulse duration

ABBREVIATIONS

ASIC	application-specific integrated circuit
CE	compute element
CN	compute node
CNCM	CN-constrained model
COTS	commercial-off-the-shelf
CPU	central processing unit
DRAM	dynamic random access memory
DSP	digital signal processor
FFT	Fast Fourier Transform
Gflops	gigaflops, i.e., billion floating point operations per second
I/O	input/output
ISMM	ideal shared-memory model
Mflops	megaflops, i.e., million floating point operations per second
PCI	peripheral component interconnect (bus)
PRI	pulse repetition interval
RISC	reduced-instruction-set-computing
SAR	synthetic aperture radar
SHARC	Super Harvard Architecture Computer
SNR	signal-to-noise ratio
SQP	Sequential Quadratic Programming
SWAP	size, weight and power
UAV	unmanned aerial vehicle
VLSI	very large scale integrated (circuit)
VME	VERSA-Model Eurocard (bus)

CHAPTER I

INTRODUCTION

Even as increasingly more computing power is available on ever decreasing areas of silicon, the processing requirements of modern applications often exceed the capabilities of individual processors. That is, regardless of the speed and memory of a system, there always will exist some application that pushes the envelope of imaginable computation. It is highly probable that this maxim will remain valid for all generations of computers to come. Out of this truth was born parallel processing.

When current technology cannot provide a single chip with adequate performance, it seems reasonable to assume that multiple chips might work in tandem to provide for the shortcomings of the single chip. However, apart from the fact that a vast number of computational tasks are not easily parallelizable, the physical requirements of multiple processors can pose critical difficulties in terms of size, weight, and power (SWAP). Such constraints especially hold true for embedded systems.

Synthetic aperture radar (SAR) data processing often belongs to this genre of problems that require both high-performance computing and adherence to tight SWAP constraints. Intensive computing results from the massive amount of information that is required to process a SAR image and SWAP constraints are

due to the nature of the host vehicles of such systems — often unmanned aerial vehicles (UAVs) or spaceborne orbiting satellites. Assuming the requirement of multiple processors and exploiting the well-defined parallelization of SAR processing, it is beneficial to determine the exact configuration of hardware and software that will optimize limited resources (i.e., SWAP). This work proposes two optimization models based on mathematical programming. The models are applied to a Mercury Computer Systems' RACE heterogeneous multicomputer [7], assumed to be onboard a tightly SWAP-constrained UAV, on which a SAR stripmap image processing algorithm is mapped across multiple computing elements.

This work begins with an overview of the background material. Chapter II briefly covers the principles of radar and synthetic aperture radar and the formulas that are most relevant to the processing of the data. Chapter III provides an overview of the Mercury RACE multicomputer and applies the processing techniques discussed in Chapter II to the Mercury RACE system. Chapter IV formulates the optimization problem in the context of mathematical programming and establishes a basis for applying it to the configuration of a Mercury RACE system. Chapter V introduces an ideal shared-memory model (ISMM) and investigates a representative sample of solutions using this model. Chapter VI introduces a more sophisticated and realistic approach, the CN-constrained model (CNCM). Comparison to the ISMM is conducted and the utility of the

ISMM as an approximator to the CNCM is investigated. Chapter VII explores the use of random configurations to both verify the solutions obtained from the models discussed and also possibly provide an alternative method of performing optimization. Chapter VIII concludes the work with a summary of the investigation and results.

CHAPTER II

PRINCIPLES OF SYNTHETIC APERTURE RADAR

Synthetic aperture radar (also known as synthetic array radar) is implemented in numerous systems for military, commercial, and scientific purposes. SAR's widespread use is due to its ability to produce photo-quality images with the use of radio waves. Uses include ground surveillance, terrain mapping, weather mapping, ocean current and ice floe tracking, and detection of earthquake faults. Because radio waves are relatively unaffected by poor weather and/or lighting, radar's performance remains constant in most conditions. In contrast to most optical techniques, as a ranging instrument radar can deliver true three-dimensional images. As discussed below, SAR distinguishes itself from conventional radar by its drastically reduced size requirements of the physical antenna in exchange for a substantial amount of postprocessing. A brief overview of basic radar and more specific SAR principles as is relevant to this research is given below. For a thorough treatment of basic radar, the reader is referred to books such as [6, 21, 22]. Synthetic aperture radar is covered in works such as [3, 5, 9, 12].

2.1 Conventional Radar

The fundamental principle of radar involves the detection of objects by the transmission and return of electromagnetic waves. When pulses are emitted from the radar transmitter, portions of the signals are returned (with significant attenuation in power) after colliding with objects in their path. Since electromagnetic waves travel at the speed of light, the range R of an object can be easily calculated by

$$R = \frac{cT_e}{2},$$

where c is the speed of light and T_e is the elapsed time from the transmission to the reception of the signal.

If the transmitter consisted simply of a point with no direction of the signal, the range information returned by an object would yield only the radius of the spherical surface on which the object resides, with the transmitter located at the center. However, transmitters typically direct the signal beam so as to sweep out a solid angle of the sphere. In the case of an airborne radar directed toward the ground, such as employed for terrain mapping or ground surveillance, the solid angle effectively becomes an elliptical area on the ground illuminated by electromagnetic waves, known as the radar's footprint (Fig. 2.1). This two-dimensional area is referred to in terms of range and azimuth, where

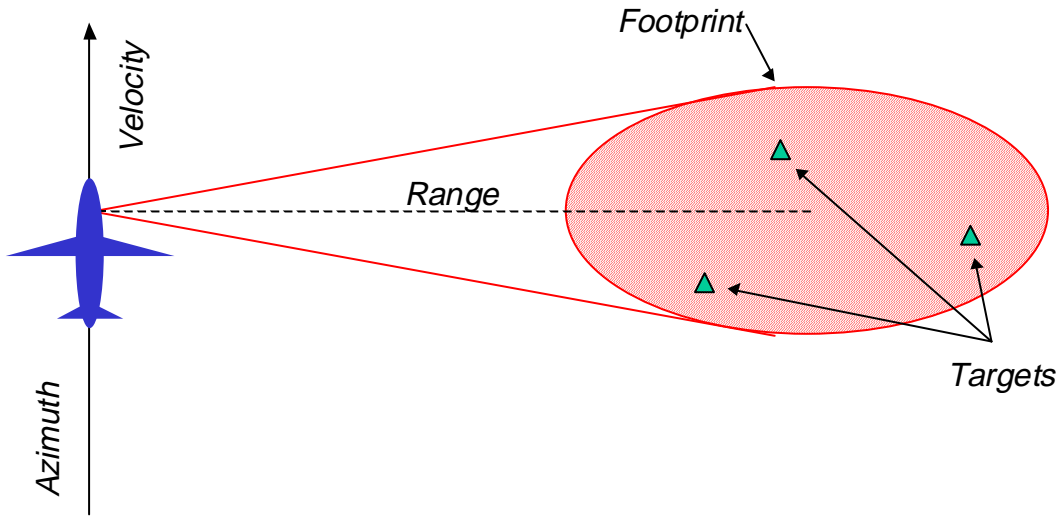


Fig. 2.1: Footprint of aerial radar.

the range dimension extends orthogonally from the aircraft and the azimuth dimension runs parallel to the aircraft's line of flight. The range swath R_s is the length of the footprint in the range dimension, and the width of the footprint in the azimuth dimension is the beamwidth at a given range. Although the beamwidth increases with range, typically it is treated as a constant, assuming an insignificant variance in the beamwidth from the bottom to the top of the range swath, at least at ranges of interest.

The radar resolution is the minimum distance between two distinguishable points on the ground. Resolutions for azimuth and range are individually calculated. However, physical parameters of the system are typically determined such that the resolutions in both dimensions are equal. Other factors, as discussed

below, determine the actual resolution for a given system. Distinction is made between a *simple radar*, which employs a minimum of signal processing, a *conventional radar*, which is mounted on a stationary platform, and finally a *synthetic aperture radar*.

Range resolution δ_R of a simple radar is affected by the transmission pulse. Directly proportional to the duration of the pulse τ_p , δ_R is defined by the following equation:

$$\delta_R = \frac{c\tau_p}{2}. \quad (2.1)$$

Therefore for fine resolution, τ_p must be small. However, a significant signal to noise ratio (SNR) in the returned signal must be maintained, requiring a high total power in the transmitted signal. A small τ_p and a set total power entails a very high burst of energy for fine resolutions, which is impractical for most systems.

To overcome this difficulty, a carrier frequency that varies with time is often applied to the pulse, known as analog linear frequency modulation. Physically, this pulse is represented by Fig. 2.2. Mathematically, however, it should be noted that each pulse is visualized as a signal with both positive and negative frequency components, centered at time $t = 0$ (Fig. 2.3). The resultant pulse is known as a chirp, and the rate with which the frequency varies is the chirp

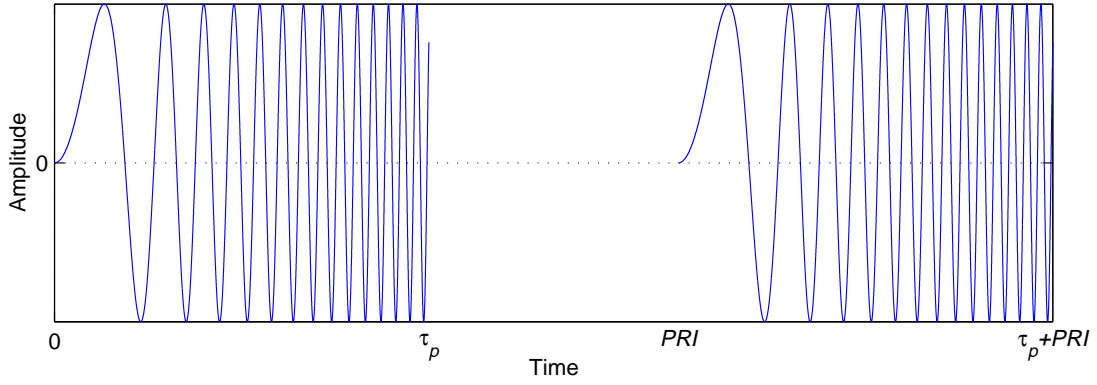


Fig. 2.2: Reference chirp starting at $t = 0$.

rate. With signal processing techniques, this method allows definition of the compressed pulse width τ_c in time as

$$\tau_c = \frac{1}{B},$$

where the bandwidth B of the pulse is the frequency differential between the lowest and highest frequencies of the carrier signal. A new equation for δ_R follows:

$$\begin{aligned} \delta_R &= \frac{c\tau_c}{2} \\ &= \frac{c}{2B}. \end{aligned} \tag{2.2}$$

The above equation for range resolution is greatly improved over the previous one employing τ_p because of the high bandwidths feasible in typical systems.

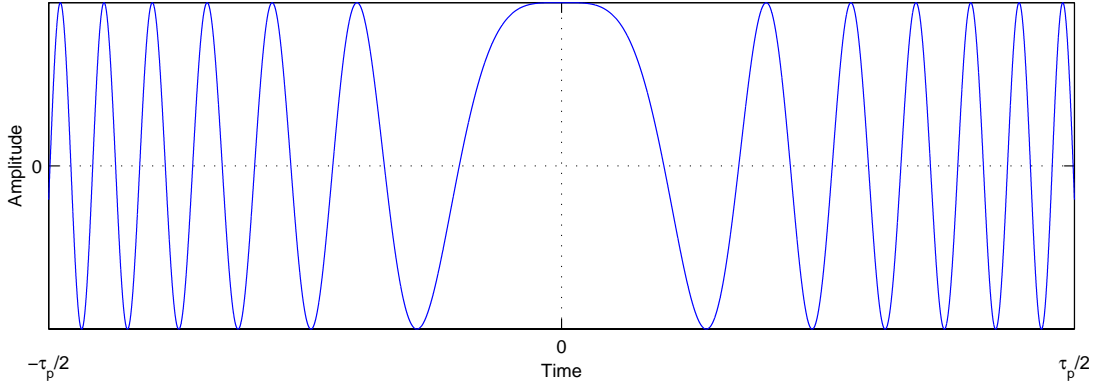


Fig. 2.3: Reference chirp centered at $t = 0$.

The carrier frequency is often in the gigahertz range, although the frequency range (i.e., B) is typically in megahertz.

With a conventional radar, azimuth (also known as cross-range) resolution δ_{real} is defined simply by the beamwidth. Most systems, however, cannot produce a sufficiently narrow beam at ranges of interest to provide acceptable resolution. Beamwidth depends upon the real antenna aperture A_{real} (length of the antenna) and is approximated as follows:

$$\delta_{\text{real}} \approx \frac{R\lambda}{A_{\text{real}}}, \quad (2.3)$$

where λ is the wavelength.

Fine azimuth resolution at a set range therefore can be obtained with a small wavelength or a large effective aperture. Different wavelengths are preferential

for different types of targets, so this variable may also be set to some degree. The effective aperture is extremely limited by the allowable size and weight of an antenna mountable on an aircraft, especially a UAV.

An intuitive method of increasing the effective aperture would be to line up several smaller antennas and average the returns. Such an array of antennas might be feasible for a ground radar system, but this provides no help in the case of airborne radar. This idea of an antenna array, however, leads to the discussion of SAR, which simulates an antenna array with only one antenna.

2.2 Synthetic Aperture Radar

SAR provides a method of obtaining high (fine) cross-range resolution with a small effective aperture. Assuming a fixed-angle radar and straight line of flight, the method (called stripmapping) exploits the inherent motion of an aircraft by utilizing many successive large-beamwidth footprints, each slightly offset in time by the interval between transmitted pulses, or pulse repetition interval *PRI* (Fig. 2.4). After processing, the resultant radar image has a resolution that is otherwise obtainable only with a beamwidth many times narrower than the real one (Fig. 2.5). The real antenna aperture then can be replaced by the synthetic aperture A_{syn} . Counterintuitively, A_{syn} can be as great as the width of the real radar footprint δ_{real} , creating an inverse relationship between real antenna length A_{real} and synthetic aperture length A_{syn} . This phenomenon is shown by

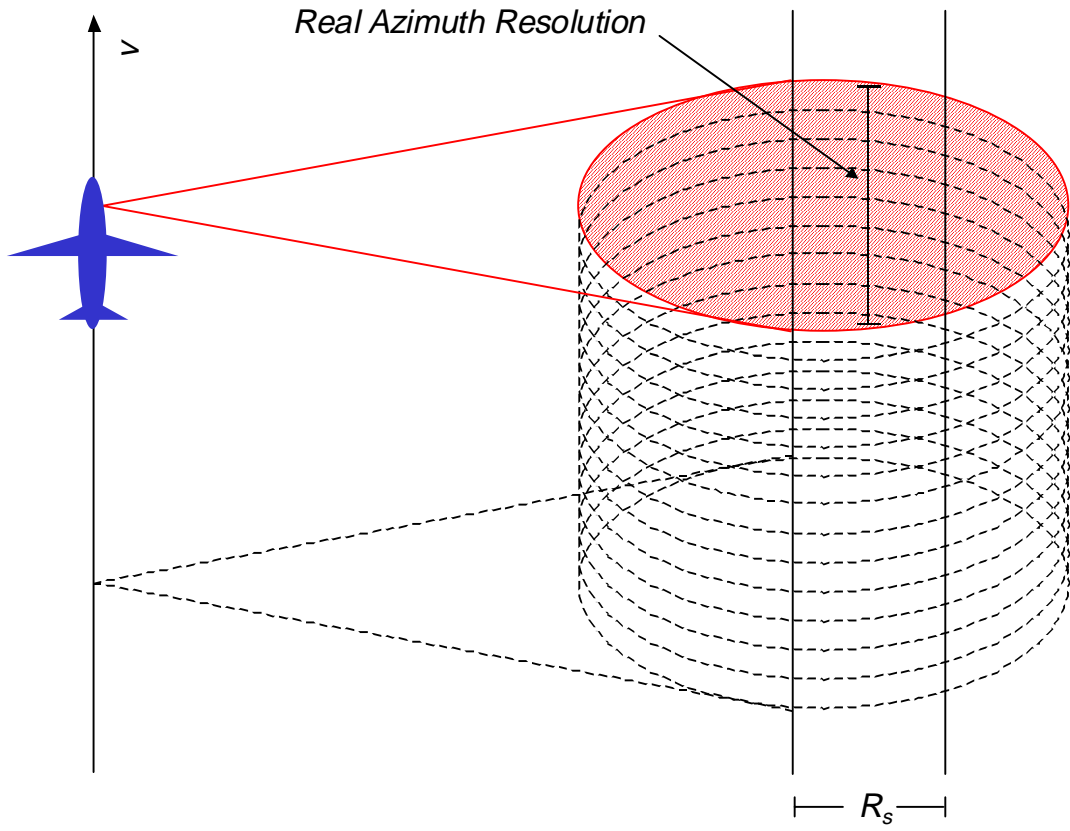


Fig. 2.4: Overlapping realbeam footprints.

substituting the equation for the real beamwidth δ_{real} (Eqn. 2.3) for A_{syn} in the corresponding equation for δ_{syn} :

$$\begin{aligned}
 \delta_{\text{syn}} &= \frac{R\lambda}{A_{\text{syn}}} \\
 &= \frac{R\lambda}{\frac{R\lambda}{A_{\text{real}}}} \\
 &= A_{\text{real}}.
 \end{aligned}$$

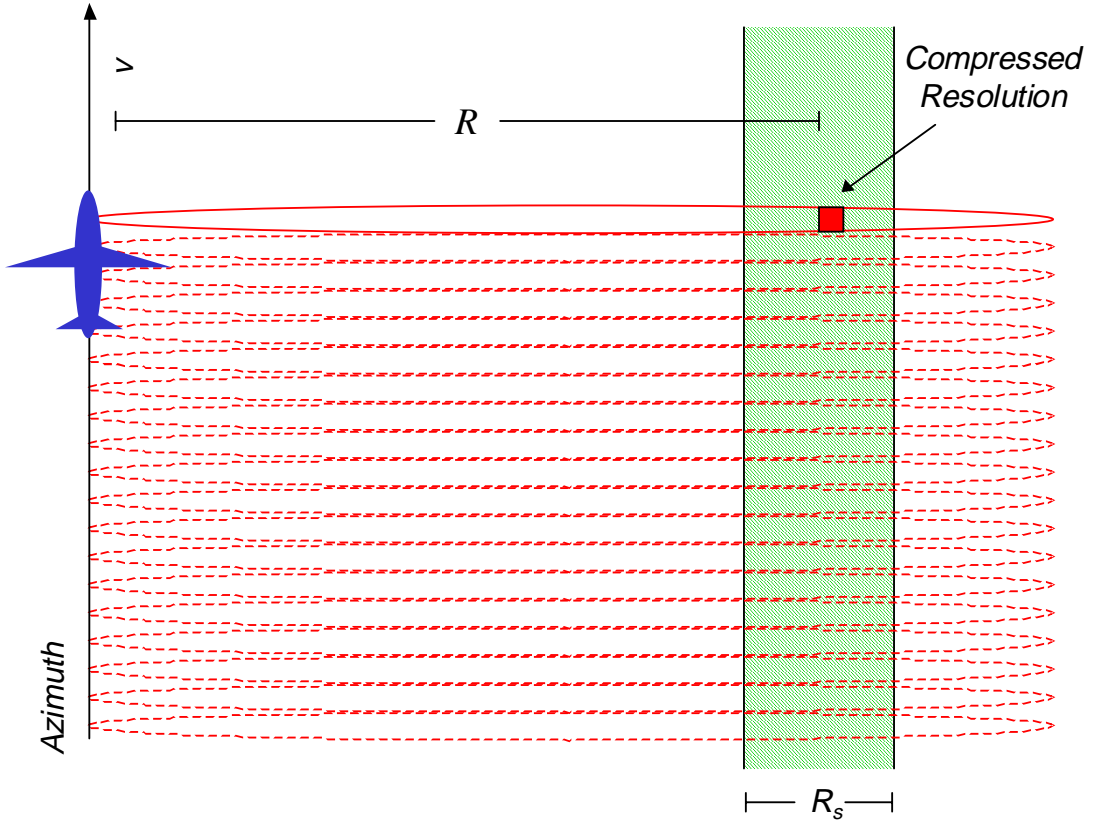


Fig. 2.5: Resultant narrow synthetic beams.

Because of increased phase sensitivity that occurs from processing the return signals of the synthetic aperture, an additional $\frac{1}{2}$ can be factored into the equation for δ_{real} [9], resulting in

$$\delta_{\text{syn}} = \frac{A_{\text{real}}}{2}. \quad (2.4)$$

Notice that the above equation for δ_{syn} involves only the antenna length A_{real} and is independent of range.

Processing of the returned signals in both the range and azimuth dimensions involves the use of matched filtering, a signal processing technique in which the returned signal is convolved with a reference signal to reduce noise and/or increase resolution. The convolution integral is defined as

$$g(t) = \int_{-\infty}^{\infty} h(\tau)f(t - \tau)d\tau, \quad (2.5)$$

where $g(t)$ is the output, $h(t)$ is the impulse response of the filter, and $f(t)$ is the input signal. For a matched filter, the reference function $h(t)$, also known as the convolution kernel, is essentially a mathematically manipulated version of the original signal. The resultant waveform from the convolution contains a “spike,” or mainlobe, representing the return for the compressed range pulse or compressed azimuth beamwidth (Fig. 2.2).

In azimuth processing, the matched filter effects the focusing of multiple real beamwidths to the compressed beamwidth by emulating the geometry of a radar receiver dish. Receiver dishes are classically parabolic in shape because of the property of parabolas in which lines parallel to the axis and incident on the parabola all converge at the focus. Furthermore, the lengths of all such line segments originating from a common range are equal. At ranges of interest, the return signal is approximated by a plane wave composed of parallel lines. SAR emulates a parabola, even though the synthetic array is a straight line,

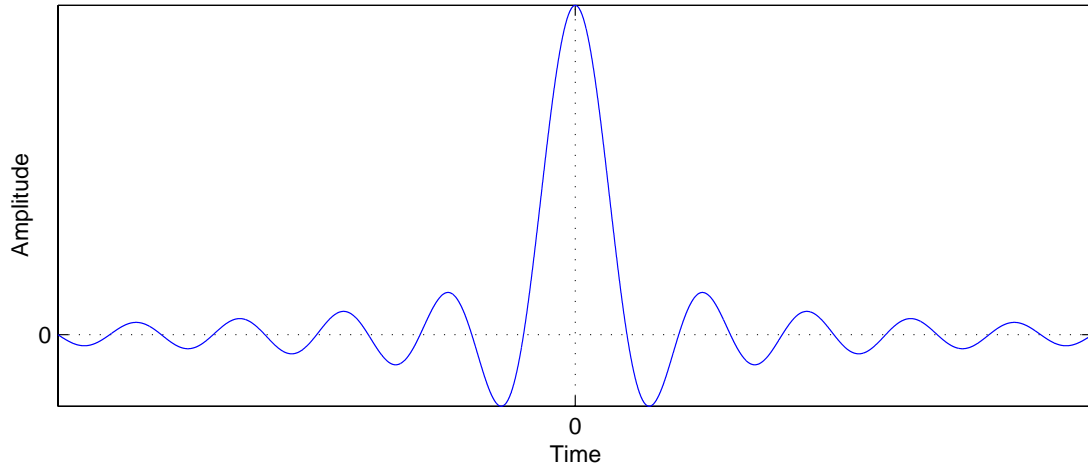


Fig. 2.6: Example waveform for the convolution of a return signal with reference signal.

by mathematically setting the time delays in the return signals that would have occurred if the array was parabolic in shape. This compensation is accomplished by the convolution of the return signal and the reference function. Without this adjustment, an antenna array is termed “unfocused.”

The convolution integral in Eqn. 2.5 is most efficiently implemented with the use of Fast Fourier Transforms (FFTs), where the Fourier Transform $F(\omega)$ of a function $f(t)$ is given by

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t} dt. \quad (2.6)$$

The inverse Fourier Transform is similarly expressed as

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega.$$

The FFT describes a family of efficient algorithms for computing the discrete form of Eqn. 2.6. This use of FFTs to perform convolution is known as fast convolution. Fast convolution exploits the Fourier transform property in which the product of two functions in the frequency domain is equivalent to their convolution in the time domain. The Fourier transform of Eqn. 2.5 is

$$G(\omega) = H(\omega)F(\omega), \tag{2.7}$$

where $H(\omega)$ and $F(\omega)$ represent the reference and received signals, respectively. Computationally, Eqn. 2.7 can be faster to perform on a digital computer than Eqn. 2.5. It should be noted that at the digital processing level, the operations are discrete, with the input function a sampled representation of the real returned signal. The discrete forms of Eqns. 2.5 and 2.7 are, respectively,

$$g[t] = \sum_{\tau=-\infty}^{\infty} h[\tau]f[t - \tau],$$

and

$$G[\omega] = H[\omega]F[\omega].$$

Similarly, the reference function is represented by a discrete number of sample points equal to a value known as the time-bandwidth product.

In the case of range processing, the time-bandwidth product depends strictly upon the original signal, where the time component is τ_p and bandwidth is B . This value also represents the compression ratio achieved by processing, as determined in Eqn. 2.2. Because the reference function in a matched filter convolution is called the kernel, this parameter will be referenced in this work as the range kernel size K_r , representing the number of discrete points in the kernel. A convenient form of this value is derived by taking the ratio of uncompressed range resolution (Eqn. 2.1) to the compressed range resolution, resulting in

$$K_r = \frac{c\tau_p}{2\delta_R}. \quad (2.8)$$

A brief discussion of Doppler effects precedes derivation of the azimuth reference function and time-bandwidth product. Doppler frequency shift describes the effect of relative motion between two objects on the reception of a signal by one object when transmitted or reflected by the other object. The shift

is inversely related to the rate of change of distance between the two objects. Given a fixed radar platform velocity, the distance from the aircraft to an object constantly changes within the synthetic aperture and thus so does the Doppler frequency. The frequency of the returned signal then will not be identical to that of the transmitted signal.

Taking time $t = 0$ to be the point at which the radar is directly perpendicular to the target in question, i.e., when the center of the real beam (and synthetic beam) footprint is upon the target, the formula for Doppler frequency f_d is approximated as follows:

$$f_d \approx \frac{2v^2t}{R\lambda} \quad (2.9)$$

where v is the velocity of the aircraft. When a target first appears in the real beam footprint, f_d is at its maximum because the rate of change in the azimuth dimension toward the target is at a maximum. When the object is in the center of the aperture, f_d is zero. Then, as the object leaves the footprint, f_d assumes its greatest negative value. Doppler frequency information, as well as signal intensity, from the received signal is stored for azimuth resolution processing.

The azimuth time-bandwidth product depends upon the range of Doppler frequency in the returned signal rather than on the bandwidth of the transmitted signal as in range compression. The frequency range is defined by Eqn. 2.9. The

time parameter is the duration of the synthetic aperture rather than that of the real aperture. As for range, the value also defines the azimuth compression ratio, describing the ratio of the real beamwidth to the synthetic beamwidth. Using Eqns. 2.3 and 2.4, azimuth kernel size K_a can be represented as follows:

$$\begin{aligned}
 K_a &= \frac{\delta_{\text{real}}}{\delta_{\text{syn}}} \\
 &= \frac{2R\lambda}{A_{\text{real}}^2} \\
 &= \frac{R\lambda}{2\delta_{\text{syn}}}.
 \end{aligned}$$

This chapter has presented an overview of the fundamental mathematics of SAR. The next chapter incorporates the equations derived in this chapter into the framework of a SAR processing system based on the Mercury RACE multicomputer.

CHAPTER III

THE MERCURY RACE SYSTEM

Following is an overview of the Mercury RACE multicomputer. This first section has been taken in part from J. West's introduction to the Mercury RACE system [24]. West is currently researching the Mercury RACE at the network level for the target application of space-time adaptive processing.

In recent years, Mercury Computer Systems, Inc. has emerged as one of the leaders in the development and manufacturing of high-performance embedded heterogeneous message-passing systems designed to address complex real-time applications requiring tremendous computational throughput. The computational requirement is often in the order of billions of floating point operations per second (Gflops). Mercury's RACE multicomputer provides a foundation for parallel systems and offers a set of building blocks that provide upward scalability. A high-level diagram of a typical RACE multicomputer is illustrated in Fig. 3.1. The system's primary components include DSPs, reduced-instruction-set-computing (RISC) processors, I/O ports, and a network interface all connected via the RACEway interconnection network.

The fundamental computing unit in the RACE system is the compute node (CN). A CN houses one or more compute elements (CEs) of the same type, either central processing units (CPUs) or digital signal processors (DSPs). Daughter-

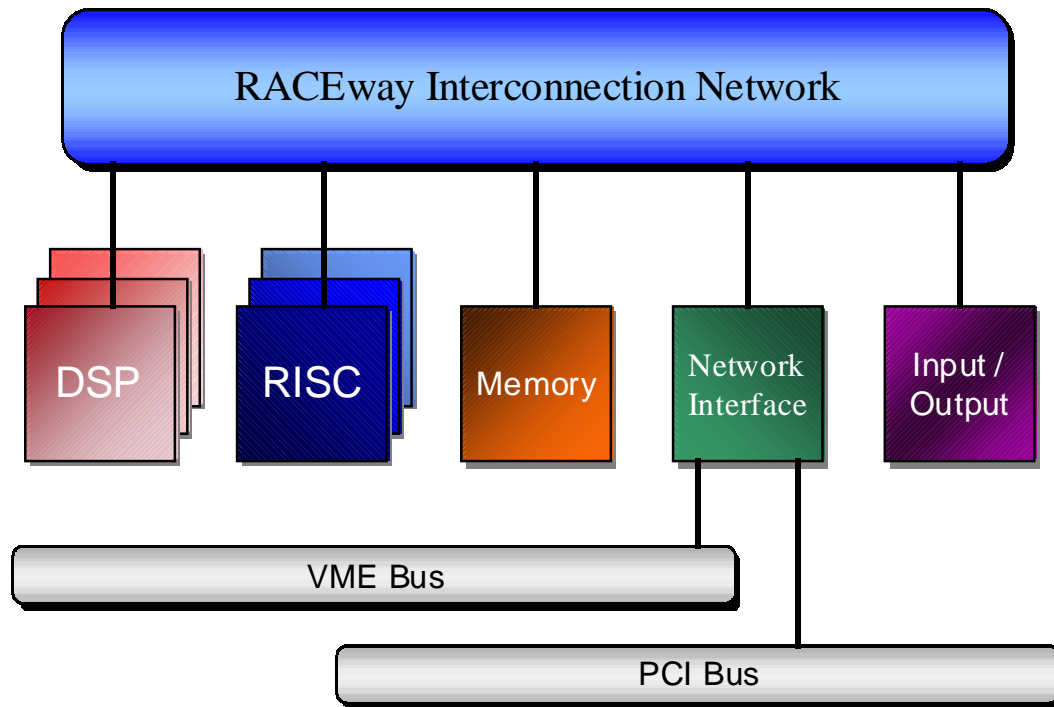


Fig. 3.1: The RACE multicomputer.

cards accommodate one or two CNs of the same type. Two or more daughter-cards reside on one mainboard. Communication between CNs, daughtercards, and mainboards takes place through the RACEway interconnect. Communication between processors residing on the same CN, however, is routed through the logic within the CN. This hardware logic also facilitates access to a shared memory block by all processors on the same CN, as well as accomodating remote memory accesses from other CNs.

The RACEway interconnection network is the framework used to provide

high-performance communications among the interconnected processors and devices. Each node in the multicomputer interfaces the network through the RACE network chip. The network chip (see Fig. 3.2) is a crossbar with six bidirectional channels consisting of 32 parallel data lines and eight control leads. Each crossbar transfers data synchronously at a clock rate of 40 megahertz (MHz). Each channel is bidirectional but is only driven in one direction at a time at 160 megabytes per second (MB/s) [15]. Among the six ports comprising a RACE crossbar, each switch can either interconnect any three port pairs, providing an aggregate bandwidth of 480 MB/s, or can cause data to be broadcasted to all or a subset of the remaining five ports [7].

The versatility of the RACE network chip allows the RACE multicomputer to be configured into a number of different network topologies. Possible network topologies include two-dimensional (2-D) and three-dimensional (3-D) meshes, 2-D and 3-D rings, grids, and Clos networks. However, the most common configuration is a fat-tree architecture (see Fig. 3.3). For a fat-tree configuration, the crossbar switches are connected in a parent-child arrangement. Each crossbar has two parent ports and four child ports (see Fig. 3.2). The crossbars of the RACE multicomputer are connected to form the branches of the fat tree. The compute nodes represent the leaves of the tree.

To route a message from one processor to another, the message goes up the tree, selecting one of the two parents as it goes, until it reaches a network chip

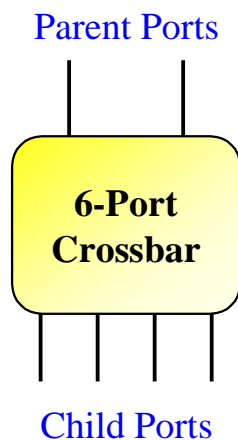


Fig. 3.2: The RACEway six-port network chip (derived from [15]).

that is a common ancestor of both the source and destination node [15]. After reaching the common ancestor network switch, the message travels down the fat tree to the destination compute node. Fig. 3.4 illustrates a message transfer between two CNs.

In conventional tree architectures, there is only one path between any pair of processors. One major problem associated with such conventional networks is that they suffer communication bottlenecks at higher levels in the tree. For example, when several compute nodes in the left subtree communicate with compute nodes in the right subtree, the root node must handle all the messages [14]. This problem can be partially alleviated by increasing the number of effective parallel paths between compute nodes. This type of modified tree architecture is referred to as a fat tree. Mercury’s RACE system is based on the fat-tree

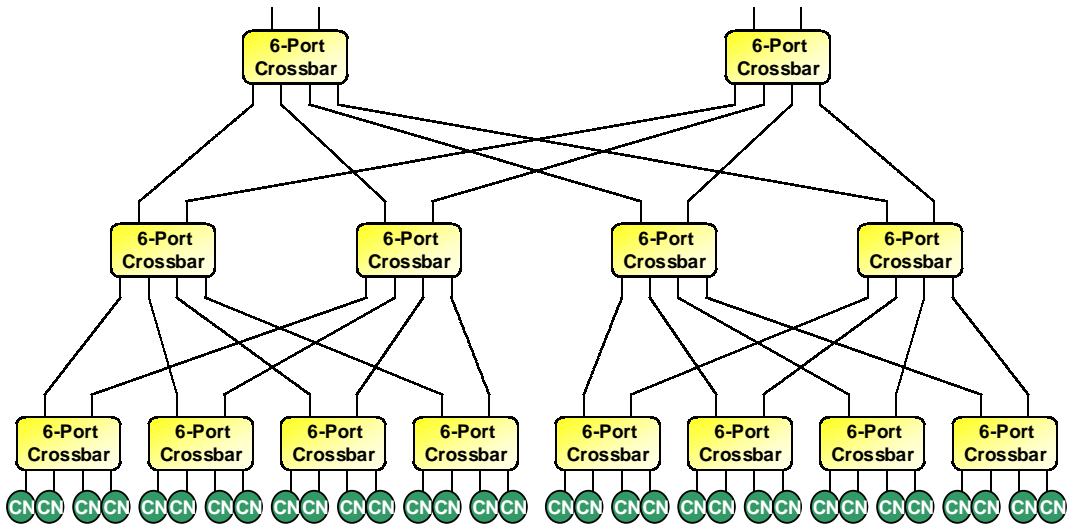


Fig. 3.3: The RACE multicomputer fat-tree interconnection network.

topology.

The RACE system is a circuit-switched network. In a circuit-switched network, a compute node establishes a path through the network. Once the compute node has been granted a path to the destination node, the path is occupied for the duration of the message transmission.

To send a message through Mercury's fat-tree network, the first step is to establish a path. To establish a path, a message header specifying a path is sent through the network along a given channel. The status of a channel is categorized as either free or occupied. The header makes as much progress as possible through the network until blocked. After a message header has been blocked, it waits until a free channel becomes available. When a free channel

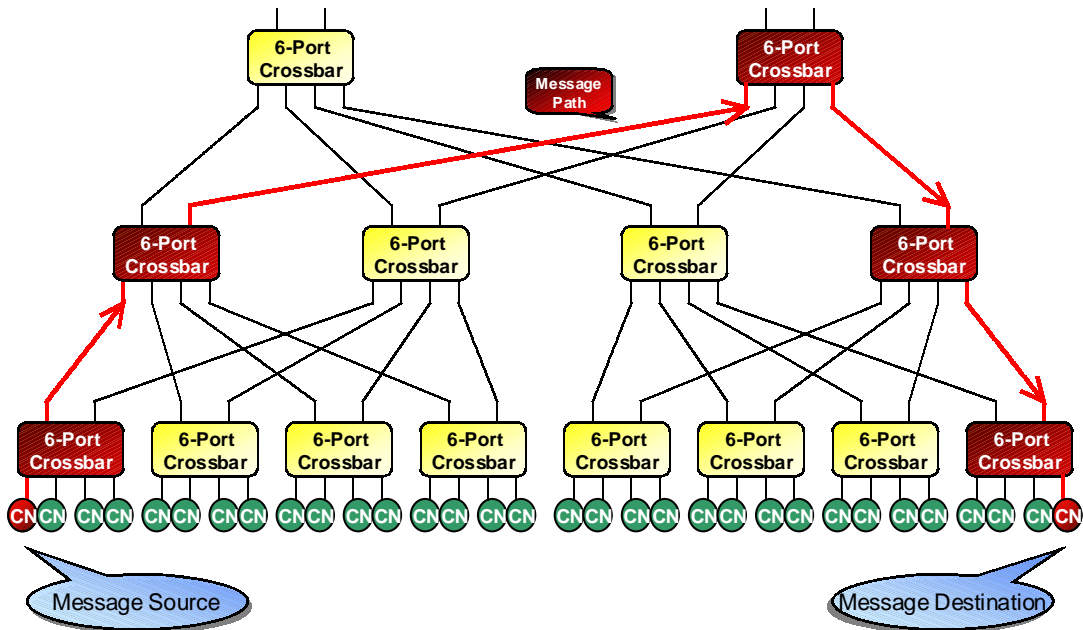


Fig. 3.4: Message transfer between two CNs.

matching the path specification (of the message header) becomes available, the channel is flagged as occupied, and the message header advances along that path. After establishing a path to the destination node, the message header sends an acknowledgment to the source along the allocated path. Upon receiving acknowledgment of a granted network path, the source node sends its message down the path in a pipelined fashion [15]. During the transmission of the last byte of data, the status of each occupied channel is set to free.

As stated above, the Mercury interconnection network under consideration is a fat-tree architecture comprised of multiple parallel paths. An interesting feature of the Mercury system is that it provides auto route path selection at

the crossbar level, which means the multiple paths in the RACEway network may be automatically and dynamically selected by the RACE network crossbars. For instance, if one path is currently occupied with a data transfer and another path matching the path specification is free, the free path is automatically selected by the crossbar logic [19]. Auto-route path selection frees the programmer from the details of path routing. In addition, processes that require high amounts of interprocessor communication, such as a distributed matrix transposition, benefit from adaptive routing [7].

In networks that take advantage of adaptive routing, some type of priority scheme is typically used to avoid deadlocks and guarantee that an application will meet tight real-time constraints. To facilitate the implementation of a priority scheme, each message header includes a priority number, ranging from zero to three. To understand the role of priorities, suppose a high priority message arrives at a crossbar, and all the outgoing channels matching the message's path specification are occupied by other messages. If a lower priority message occupies one of the channels that the higher priority message needs, the lower priority message is required to release the channel in the Mercury system [15]. The lower priority message is suspended by sending a "kill" signal backwards along the path to the source node. Data that was already in the path propagates down the pipeline to the destination node with the current byte releasing the channels as if it were the last byte of data in the message. After the path becomes

free, the higher priority message may gain access to the channel. The lower priority message resumes when a free channel becomes available. The processor-network hardware contains built-in facilities that handle the suspension and reestablishment of a killed message.

For messages contending for the same channel with the same priority, the incoming port number is the tie-breaking mechanism. Furthermore, messages coming from parents have a higher priority than messages from children, and messages coming from a higher numbered parent (or child) port number have a higher priority than messages originating from lower numbered ports. However, this is only a tie-breaking mechanism for messages arriving or blocked at the same crossbar, and it does not result in suspension of any message that has already been routed to the next switch [15].

With the network configured as a fat tree, the RACEway interconnection fabric provides very good scaling properties. In a p -processor system, the height h of the fat tree is $h = \lceil \log_4 p \rceil$. Thus, the network diameter D or maximum number of links traversed is $D = 2h - 1$. The bisection bandwidth B of a system, which is defined as the minimum number of edges (or channels) that have to be removed along a cut that partitions the network into two equal halves, assuming $p = 4^k$ processors, where k is an integer, is $B = 160\sqrt{p}$ MB/s. (Each channel in the RACEway system has a bandwidth of 160 MB/s [15].

The RACEway system may be configured as a heterogeneous multicomputer

composed of two or more different types of processors. The potential heterogeneity of the RACE multicomputer includes various possible configurations of i860, PowerPC, and Super Harvard Architecture Computer (SHARC) DSP processors. The SHARC DSP is ideally suited for embedded vector signal processing operations such as FFTs where physical size and power are at a premium or other similar algorithms that have a high ratio of data-to-computation. Furthermore, Analog Devices' 21060 SHARC processor provides more than twice the physical processor density of RISC-based CNs. In contrast, the PowerPC and i860, both RISC processors, are appropriate for executing scalar-type applications, with a low ratio of data to computation, generated by arbitrary compiled code.

Because this work focuses on optimization of the FFT-intensive operations involved in SAR processing, it is assumed that the system studied uniformly employs SHARC CNs. However, there exist different types of CNs even with the same type CE. At the time of this writing, the two standard SHARC-based daughtercards are the S2T16B and the S1D64B. The S2T16B implements two CNs for a total of six CEs and 32 MB of DRAM. The S1D64B implements only two CEs but 64 MB of DRAM, all contained within one CN. The power consumption of the S2T16B and the S1D64B are 12.2 and 9.6 watts, respectively. Clearly, both daughtercards have different characteristics, each with a different CE-to-memory ratio and power consumption penalty.

3.1 Mapping of SAR Processing onto the RACE System

The basic computational framework and mapping of CEs assumed here is the same as that described in [8]. The descriptions given in this section and the next represent an overview; for more details refer to [8].

CEs are divided into range and azimuth CEs. Every CE is dedicated exclusively to the processing of data either in the range or azimuth direction. Although it would be possible to investigate the utilization of individual CEs for the simultaneous processing of both range and azimuth data, only one fractional CE each for range and azimuth is potentially wasted. Consideration of the processing overhead associated with multitasking and the memory overhead of multiple programs quickly diminishes any benefit that might be obtained from such a configuration. Furthermore, [8] recommends availability of both memory and CEs above the calculated requirement to provide for flexibility and any contingencies. Any such excess resources are usually in excess of that associated with a single CE.

After radar returns have been sampled and converted to digital signals, samples are typically read into memory at a rate of 5–50 Msamples/s [8]. By visualizing memory as a two-dimensional grid, a row of memory contains the returns from a single radar pulse, whereas a column contains returns of different pulses from the same range. Memory is therefore sequentially filled a row at a time. When a sufficient number of rows have been filled, this data is processed by a

range CE. These blocks of data are sent to the range CEs in a round-robin fashion. After a number of range CEs have processed data, the conglomerate block of data is “corner-turned,” or matrix-transposed, and then sent to the azimuth CEs. Note that the number of range and azimuth CEs need not be the same. The matrix transposition of the data dictates that the azimuth CEs receive the range-processed rows as columns and the unprocessed columns of the azimuth direction as rows. Fig. 3.5 illustrates the communication in a matrix transposition. Note that although each range processor is responsible for several signal returns (set of pulses), each range processor only needs to hold one entire return in memory for computation before sending the result to the azimuth processors.

3.2 Computational Framework

As discussed earlier, SAR processing primarily involves convolution of the data with reference functions. For the sake of simplicity and without loss of significant performance (because of the relatively small requirement of range processing as compared to azimuth), it is assumed that the entire vector of range samples for a given pulse return is processed as a single section of data. The azimuth CEs perform similar operations on the data as the range CEs (i.e., fast convolution) but with one important difference: the length of the data stream in the azimuth direction is indefinite, whereas in the range direction it is of a fixed length. Therefore the data cannot be convolved as a single entity in the

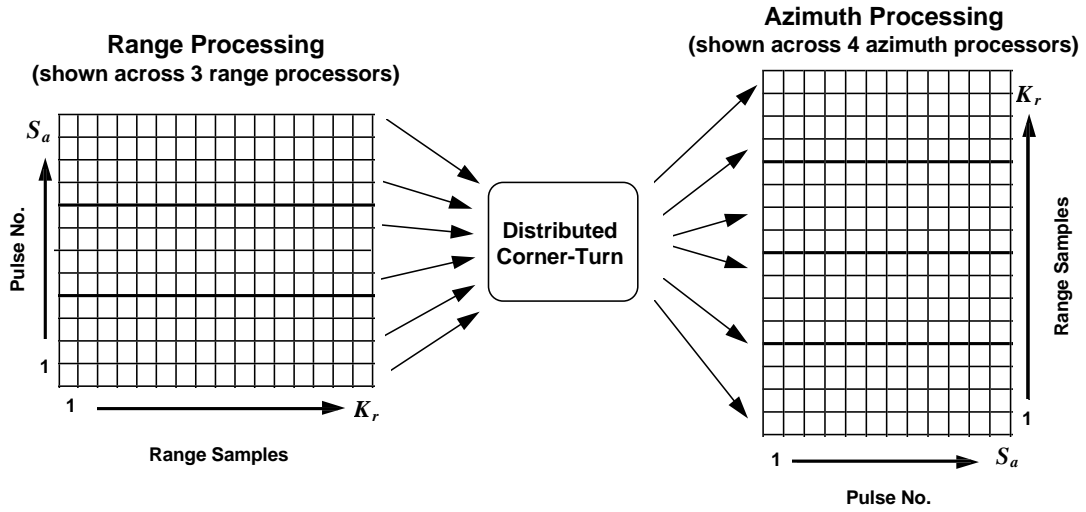


Fig. 3.5: Parallelization of the matrix transpose operation.

azimuth dimension.

Sectioned fast convolution [18] provides a method for processing data streams of indefinite length. For such a data stream, the data is divided into sections of arbitrary length. A section is then convolved with the prestored kernel as in the case of a regular fast convolution. (Note that this prestored kernel saves the time of taking the FFT of the transmitted signal each time, which ideally should be the same for each pulse. Furthermore, functions such as windowing and other filtering techniques can be included in this kernel and precalculated.) Overlapping the sections by an amount equal to the kernel size and performing fast convolutions on each overlapped section yields the same result as if the entire data stream were convolved at once. However, there is a price to be paid

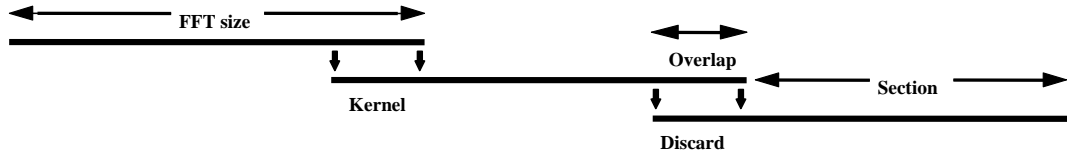
in computational efficiency for using this method. A portion (of length equal to the kernel size) of each convolution resultant must be discarded. Therefore computational efficiency decreases as the ratio of the section of new data to the kernel size decreases. Fig. 3.6 illustrates the principle of sectioned convolution.

Besides memory, another limiting factor to the size of the new data to be convolved is the $O(N \lg N)$ time complexity of the standard FFT algorithm. An important objective is to balance computational efficiency with memory requirements. For instance, selecting a section size that maximizes computational efficiency alone, without regard for concomitant memory requirements, may be unfavorable due to high power consumption by the memory. Accounting for this tradeoff is an important aspect of the model presented in this work.

A fast convolution consists of an N -point FFT, N complex multiply operations, and an N -point inverse-FFT, where N is the number of data points to be processed, including any overlap. The complexity of this computational load is therefore $L = O(N \lg N + N)$. The exact number of floating point operations generally depends on CE- and implementation-specific details. If SHARC CEs are assumed, the exact number of floating point operations is given by [8]:

$$L = 10N \lg N + 6N.$$

The computational load per sample is obtained by dividing L by the number of



Large Overlap/Section ratio \Rightarrow **Small azimuth memory, large number azimuth processors**
Small Overlap/Section ratio \Rightarrow **Large azimuth memory, small number azimuth processors**

Fig. 3.6: Sectioned convolution.

new data points processed, which reflects the efficiency of the calculation. For range processing this load per sample ϕ_r due to the fast convolution is given by

$$\phi_r = \frac{10F_r \lg F_r + 6F_r}{S_r},$$

where F_r is the FFT size for the range and S_r is the number of points in the range to be processed. These two values can differ because of the stipulation in the FFT algorithm that requires the FFT size to be a power of two (i.e., $F_r = 2^k$). Although this implies some inefficiency, it is usually still faster than using a direct convolution algorithm based on the exact sequence length.

The number of range points S_r is equal to the range swath R_s divided by the desired resolution δ (assuming $\delta_{\text{syn}} = \delta_R = \delta$). That is,

$$S_r = \frac{R_s}{\delta}. \quad (3.1)$$

Using this expression, the equation for ϕ_r becomes

$$\phi_r = \frac{\delta F_r (6 + 10 \lg F_r)}{R_s}.$$

Similarly, the azimuth processing load per sample due to the fast convolution is given by

$$\phi_a = \frac{F_a (6 + 10 \lg F_a)}{S_a},$$

where F_a is the azimuth FFT size and S_a is the azimuth section length.

To determine the number of CEs required for both range and azimuth processing, the total computational load must be derived. The fast convolution comprises the majority of the load. However, several other operations are also involved, including fix-to-float conversion, complex signal formation, motion compensation, magnituding, and the matrix transpose already mentioned [8]. It is important to realize that different operations can take different amounts of time, even if they are considered to be a “single floating point operation.” Therefore, calculating the total computational load requirement per data sample involves dividing the number of real operations per sample of each type by their respective tested throughputs for a given type of CE. This value multiplied by the sample rate yields the total number of CEs required.

Range and azimuth processing have unique load requirements in addition to the fast convolution load and are noted by the constants α_r and α_a , respectively. The required number of range CEs is then defined by

$$P_r = Q(\alpha_r + \frac{\phi_r}{\gamma}), \quad (3.2)$$

where Q is the sample rate and γ is the throughput in Mflops for a fast convolution based on the assumed CE type used. Similarly, the number of azimuth CEs required is given by

$$P_a = Q(\alpha_a + \frac{\phi_a}{\gamma}). \quad (3.3)$$

It can be shown that the sample rate is determined by the following equation [8]:

$$Q = \frac{vR_s}{\delta^2}.$$

If this expression is substituted for Q and the expressions for ϕ_r and ϕ_a are also

applied, then Eqns. 3.2 and 3.3 become

$$\begin{aligned}
 P_r &= \frac{v(6\delta F_r + \alpha_r \gamma R_s + 10\delta F_r \lg F_r)}{\gamma \delta^2} \\
 P_a &= \frac{v R_s (\alpha_a + \frac{F_a(6+10 \lg F_a)}{\gamma S_a})}{\delta^2}.
 \end{aligned} \tag{3.4}$$

The total memory required for range processing is a product of the number of range CEs P_r and the number of range samples S_r . This value represents the number of complex range samples that are stored in memory at a given instant, each complex sample consisting of 16 bytes. Therefore the total range memory required is

$$M_r = 16P_r S_r,$$

or equivalently,

$$M_r = \frac{16R_s v(6\delta F_r + \alpha_r \gamma R_s + 10\delta F_r \lg F_r)}{\gamma \delta^3}. \tag{3.5}$$

Azimuth memory requirements dominate total system memory, necessitating a double-buffer (for the matrix transpose operation) and an output image buffer, both of size $S_r(S_a + K_a)$. The double-buffer stores complex values; the output image buffer stores reals. Storing the data in the double buffer as complex

fixed-point values instead of complex reals reduces the double-buffer storage by 50%. Computation, however, is performed using reals to prevent a substantial compromise in precision. The total azimuth memory requirement in bytes is expressed as

$$M_a = 10S_r(S_a + K_a). \quad (3.6)$$

The value of K_a can be expressed in terms of basic parameters of the radar. Let λ be the wavelength of the radar. The value for K_a is derived in [8] to be:

$$K_a = \frac{\lambda R}{2\delta^2}. \quad (3.7)$$

Substituting this expression and $S_r = R_s/\delta$ into Eqn. 3.6 yields

$$M_a = \frac{5R_s(\lambda R + 2\delta^2 S_a)}{\delta^3}. \quad (3.8)$$

CHAPTER IV

THE OPTIMIZATION PROBLEM

The final equations derived above for P_r , P_a , M_r , and M_a , given by Eqns. 3.2, 3.4, 3.5 and 3.6, depend on several different types of basic system parameters. These basic parameters can be divided into four major categories:

- radar parameters: R (range), R_s (range swath), and λ (wavelength);
- application parameters: δ (desired resolution) and v (platform velocity);
- processor parameters: α_r , α_a , and γ ; and
- software parameter: S_a .

From Eqns. 3.2, 3.4, 3.5 and 3.6, it appears that there is also a dependence on the parameters F_r (range FFT size) and F_a (azimuth FFT size). However, recall that F_r and F_a are functions of S_r and $S_a + K_a$, respectively, and S_r and K_a can both be expressed in terms of basic radar and application parameters (see Eqns. 3.1 and 3.7).

Let the total processor requirement $P_r + P_a$ and the total memory requirement $M_r + M_a$ be denoted as P and M . Any variable will be represented as a function when it is a function of another variable that is to be implicitly or explicitly optimized in the problem. In the example of P and M , they will normally be

represented as $P(S_a)$ and $M(S_a)$ as a reminder that they are dependent on the optimization variable S_a .

Throughout this work, it is assumed that all radar and processor parameters are fixed. Depending on the optimization objective, one or more of the application or software parameters will be optimized, with the remaining parameters fixed or implicitly computed. However, in each case the optimal S_a must be determined. The problem then focuses on finding an optimal value for S_a and the effect of S_a on the resulting computing platform configuration and the optimization objective.

4.1 Mathematical Programming

Optimization of parameters can be achieved through a method known as mathematical programming. The expression that represents the variable or variables to be minimized is called the objective function, consistently designated in this work as Z . Additional inequality and equality relations comprise the set of constraints for the programming problem.

Many algorithms exist for different types of mathematical programming problems. The algorithm employed depends on the nature of the optimization. If possible, it is favorable to formulate the problem such that the solution space can be constructed by a set of linear equations. Such a problem falls into the category of linear programming. One common linear programming algorithm is

the Simplex Method. The Simplex Method merely examines vertices of the two-dimensional solution space constructed by the constraints until the minimum value is found, the linear nature of the solution space assuring that a minimal solution will fall on a vertex.

Often, however, nonlinear equations enter into the problem, as is the case in this work. Nonlinear programming requires more complex algorithms. MATLAB's **constr** function in the Optimization Toolbox is employed in this work to solve the nonlinear programming problems.

The **constr** function implements a Sequential Quadratic Programming (SQP) algorithm, which constitutes a family of the most efficient constrained optimization algorithms currently known. See [2] for specific information on the MATLAB implementation of the **constr** function and [10] for an overview of the SQP algorithm.

Like most optimization algorithms, the SQP algorithm requires that the solution space is convex to guarantee an optimal solution. That is, no local minima or maxima exist in the solution space to deceive the algorithm to settle for a suboptimal solution. In general, without an exhaustive search, which is not practical in the continuous domain, no algorithm can guarantee an optimal solution from a nonconvex solution space. Optimization algorithms operate on the basis of finite changes in the objective function value due to small changes in the optimization variables. When no more improvement can be obtained, within a set

tolerance, from various excitations of the optimization variables, the algorithm is done. If the optimization algorithm ventures into a local minimum or maximum, the conditions for algorithm completion are met even if the value of the solution is very bad relative to other points in the solution space.

Proving the convexity of a multidimensional problem can be very time consuming, without the guarantee of an eventual solution. Even if a problem can be proved to be nonconvex, optimal solutions still often are desired, without a way of restructuring the problem to be convex. One method of countering such problems in a nonconvex space is to either backtrack from a solution and go in a different direction from the one previously taken for a certain number of steps, even if the immediate results from such an action return an inferior objective function value. If the algorithm consistently returns to the original solution, it may be assumed with increasing confidence that the given solution is not a local minimum or maximum. This procedure resembles a Tabu search or simulated annealing algorithm, of which portions may be applied, with either deterministic or probabilistic methods of determining the next direction to try [4, 11, 17].

Another method of approaching nonconvex or possibly nonconvex problems is to solve the same problem many times, with each solution given a different initial guess. To provide the highest level of confidence that a resultant solution is good (optimal or near optimal), the range of initial guesses should be as disparate as is practically feasible. These initial guesses, consisting of values for one or

more optimization variables, can be deterministically or nondeterministically generated.

Unless explicitly stated, it is assumed that the problems optimized in the succeeding chapters are convex. Although convexity has not been formally proved for any case, it is considered adequate that widely ranging initial guesses affect only the speed of convergence of the optimization algorithm, not the final solution. Furthermore, obvious logic dictates that when plotted against linearly spaced independent variable values, the optimal solution surfaces would represent either nondecreasing or nonincreasing functions of those variables in each dimension. As this research illustrates, this trend holds true for all the problems investigated except where explicitly noted. See Section 5.3 for an example of a possibly nonconvex problem formulation.

Integer programming constitutes another type of mathematical programming. Integer programming imposes the stipulation that one or more of the variables to be optimized must be an integer, and can be applied to either linear or nonlinear programming. In general, efficient algorithms do not exist for integer programming, and solving such optimization problems can be computationally intensive (see [13] for a summary on integer programming techniques). Although different approaches exist in integer programming, essentially a problem must be solved using linear or nonlinear programming, the noninteger result rounded up or down, and the solution then recomputed based on the new values.

Because it is difficult to predict how the rounding of one variable will affect the other variables, testing of many permutations of the discrete variables may be required. In many cases, the optimal solution may not involve the floor or ceiling of the calculated optimal value for a given variable, especially in nonlinear problems with many variables.

4.2 Optimization Objectives

Several different optimization objectives are investigated in this work. An optimization objective can be categorized in two different dimensions. The first dimension concerns the SWAP constraint of interest. In UAV systems, all SWAP constraints are critical. However, in terms of computer processing, the most variable and controllable parameter is power. The weight of a computer system largely results from the chassis, which for UAVs is commonly custom designed. The size of the embedded system is a combination of volume of individual components and the geometry by which they are arranged and physically configured. Straightforwardness and generalized application in the calculation of power requirements lend power consumption to be the fundamental case of study in this research.

The second dimension of the objective function involves the parameter, or performance measure, to be optimized. The most obvious parameter to be optimized is power consumption. The minimization of power is indeed the primary

objective function under consideration for this work. However, within the framework of computable power consumption, the maximization of velocity and the minimization (making as fine as possible) of resolution are explored as objective functions, assuming a preset maximum power consumption level for the onboard computing system. Further constraints of weight and/or size could be added to the problem, but these are not investigated in this work because of the reasons given above.

4.3 Hardware Configurability

Basic guidelines about the hardware components available for system design determine which variables can be optimized. One such set of guidelines includes the option of custom designing each component. Such endeavors should result in very efficient and highly specialized components. Another guideline that is often applied is that widely available preexisting components of less specialized purpose must be employed in the design of the system. This latter assumption is the basis of this research. Both of these approaches are examined below.

4.3.1 Optimal Configuration Using Custom-Designed Boards

The custom design of a card entails the capability to produce a board with the desired ratio of memory to processing power. The objective function Z of

such an approach is given by the following:

$$\kappa P + \beta M, \quad (4.1)$$

where κ and β are constants that represent power requirements on a per processor (or per Mflop) and per byte of memory basis, respectively. Although the above optimization problem appears to be a simple linear programming problem (assuming the existence of constraints), even for this most basic formulation it must be noted that both P and M are functions of several variables. In particular, they are functions of the software parameter S_a . If the expressions for P and M are substituted into Eqn. 4.1, the following equivalent expression results:

$$\begin{aligned} & \kappa \left[\frac{v(6\delta F_r + \alpha_r \gamma R_s + 10\delta F_r \lg F_r)}{\gamma \delta^2} + \frac{v R_s (\alpha_a + \frac{F_a(6+10 \lg F_a)}{\gamma S_a})}{\delta^2} \right] \\ & + \beta \left[\frac{5 R_s (\lambda R + 2\delta^2 S_a)}{\delta^3} + \frac{16 R_s v (6\delta F_r + \alpha_r \gamma R_s + 10\delta F_r \lg F_r)}{\gamma \delta^3} \right]. \end{aligned} \quad (4.2)$$

Even if all variables except for S_a are fixed, Eqn. 4.2 is obviously nonlinear and requires a nonlinear programming algorithm such as **constr**. Determining a value of S_a that minimizes this function defines an optimal configuration, with a corresponding optimal number of P processors and M bytes of memory. Modeling total consumed power as described above is unrealistic in many cases because of the overhead in time and money involved in production of customized boards. In addition, the resultant system would be fairly inflexible because it

would have been optimized for a specific velocity and resolution. If either of these two application parameters changed at a later point in time or even if a radar parameter changed, the system would become inefficient at best and possibly obsolete. Any change thus would necessitate the design of a new board.

4.3.2 Optimal Configurations Using COTS

The disadvantages discussed above of custom designed boards often lead to the employment of commercially available boards, or commercial off-the-shelf (COTS) products, that contain differing numbers of processors and amounts of memory. The COTS components under consideration here are Mercury's S2T16B and the S1D64B daughtercards. Although systems composed of COTS components are theoretically never as efficient as custom designed boards because of their more generalized purpose, COTS systems often can be built to provide adequate performance for a fraction of the cost and in much less time. Furthermore, if any parameters change, the system can be quickly adapted to accommodate the changes merely by adding or removing cards and appropriately modifying the software. This work is premised on the assumption that COTS hardware must be used, specifically the Mercury RACE system and associated individual components.

With this assumption, the total power requirements of a system can be computed directly from the number of daughtercards employed. When appropriate,

the power consumption will be denoted by the symbol Π , usually with an appropriate subscript depending on the exact context. In all systems this research considers, power consumption is assumed to be equal to the product of the number of daughtercards of a given type employed and the power consumption for a single daughtercard of that type. The total power consumption of a system is the sum of all such products for each of the types of daughtercards.

4.4 Architectural Models

This research investigates in depth two models of processor-memory architectures. The first and simplest architecture assumes that all memory and processors that the daughtercards in the system contribute can be pooled and treated as a single entity. For reasons discussed below, this architectural assumption is termed as the ideal shared-memory model (ISMM). The second model treats each CN as a separate entity, and thus is designated as the CN-constrained model (CNCM).

4.4.1 Ideal Shared-Memory Model

The simplest and most intuitive approach to determine how many daughtercards are required to provide the necessary total memory M and processors P for correct system function involves taking the maximum of M and P divided by the

amount of memory or number of processors, respectively, contributed by a single daughtercard (discussed in Chapter V). In the case of the S2T16B, 32 MB of memory and six processors are contributed, or in the case of the S1D64B, 64 MB and two processors. Because the tasks involved in SAR processing are readily parallelizable, the assumption of pooled processors does not present a problem. However, the concept of pooled memory across daughtercards (or more specifically, CNs), can be dangerously naive, as discussed in the next subsection.

The above assumption provides a simplified model of a real system. Although the usefulness of such a model might be questioned, this research demonstrates that such a model can be used as a lower bound heuristic that yields adequately consistent results for the expected power consumption. Such a heuristic is useful when considering that the solution to the more realistic model discussed below involves a large amount of integer programming, resulting in a more complex problem formulation that is much more computationally intensive in terms of time. Furthermore, in the case that a different system is investigated for which memory can be effectively pooled, this model is valid. This model is investigated in Chapter V.

4.4.2 CN-Constrained Model

A more realistic model for the Mercury RACE system necessitates transition from the aggregate of daughtercards as the quantity of concern to the CN as the

fundamental hardware denominational unit in terms of memory and processors. In the ISMM, an optimal configuration of components may allow all processors on one daughtercard to be employed in computation, but each of which require memory of the daughtercard on which they reside and all the memory on several other daughtercards. In such a case, the active constraint on minimum power is memory, leaving the processors on most daughtercards idle. Although such a configuration is theoretically possible, such never occurs in practice because of the prohibitive costs in communication time involved with remote memory accesses from a processor on one CN to the DRAM of another CN. Remote memory accesses occur only during the distributed matrix transposition, which is relatively time consuming considering the amount of data handled. Note that for even two CNs on the same daughtercard to communicate, they must send requests through the crossbar, just as if they were on distinct daughtercards. Because of this fact, the CN is the fundamental unit of concern in this more realistic model.

To avoid remote memory accesses, a formulation must ensure that all processors employed on a single CN have adequate local memory to complete their tasks. Only configurations that abide by such constraints are considered in the solution. In addition, range and azimuth processors are treated as distinct since they operate based on distinct programs. That is, a processor is dedicated to either azimuth or range processing. However, there is no need for entire dedicated

range and azimuth CNs, since each processor on the CN still can operate from a different program. As discussed in depth in Chapter VI, integer programming is required to ensure integral solutions of range and azimuth processor assignments to CNs. Just for the two daughtercards under consideration, it will be shown that the number of processor assignment combinations that may need to be evaluated is in excess of 50, which translates to a 5000% increase in computation. Needless to say, in the case that there exist more than two types of daughtercards from which to select, quick solutions are not forthcoming due to the computational intensity.

4.5 Hardware Availability Constraints

The most general type of configuration assumes that there are multiple daughtercards from which to select. Furthermore, the optimization routine is given freedom to populate a system completely with only one type of card or to determine some mixture of the two card types. This case will be designated the mixed card type configuration.

If a constraint is added to the formulation such that the number of daughtercards of one type or the other must equal zero, this special case will be called the single card type configuration. Such a formulation represents a scenario in which a system will not tolerate multiple types of daughtercards or the scenario in which hardware has already been acquired and the hardware happens to con-

sist of all one type daughtercard. Furthermore, solutions to the single card type formulation provide insight into the characteristics and best usages of a card type.

4.6 Points of Reference: Nominal Configurations

It has been suggested that using a section size S_a equal to the azimuth kernel size K_a is a good heuristic for adequate system performance yet with moderate conservation of memory, which is usually the scarce resource [7]. Such a heuristic ensures the intuitively comfortable result of processing at least as much new data as old data with each convolution. However, as the research exemplifies, the optimal section size often deviates from this heuristic choice.

To illustrate the advantage of optimizing the software parameter S_a , comparisons are made of the optimal solutions to what will be termed nominal solutions, meaning that the above heuristic was employed in determining the section size instead of the optimization routine. However, even in such nominal configurations, there are often still other variables that may be optimized. If a system was designed without any forethought to optimization, basing configuration only on heuristics or the most obvious approach, it is true that no optimization of any sort might occur. In this case, even most of the nominal configurations presented in this work will outperform arbitrarily designed systems. However, giving the benefit of the doubt to system designers, and for the purpose of analyzing the

utilization of optimized section sizes, all other variables besides S_a are optimized even in the nominal configurations.

4.7 Summary

The next two chapters present the results of computed solutions for the ISMM and the CNCM, respectively. Because of the computational intensity involved with the CNCM and because of the greater insight into parameter interrelationships provided by the ISMM, more examples have been investigated in the ISMM. For the CNCM, more space is dedicated to examination of the formulation and its complexity, with fewer examples than the ISMM with the assumption that the ISMM provides a good heuristic for solutions. However, it must be noted that the CNCM provides, in addition to more realistic power consumption values, more detailed information for the actual implementation of a configuration. That is, whereas the ISMM only provides information on the number of daughtercards necessary for the entire system, the optimal azimuth section size, and the resultant power consumption, the CNCM also provides precise information on which processors are dedicated to range and azimuth and how much memory of the total local memory is allocated to each. This latter information is sufficient to design a system, whereas the designer still has several problems left, the solutions to which may be physically infeasible, to resolve with the ISMM results.

In addition to the two distinct models, within each model examples are generated by taking combinations of different optimization objectives and hardware availability scenarios. To illustrate the utilization of optimization, comparison is also made between optimal and nominal configurations.

All numerical results produced in this work are based on radar and processor parameters fixed at the following values: $R = 100000$ m, $R_s = 20000$ m, $\lambda = 0.03$ m, $\alpha_r = 0.3528$ Mflops, $\alpha_a = 0.9068$ Mflops, and $\gamma = 94$ Mflops. These values are derived from [8], representative of a real SAR system and experimental SHARC throughputs.

Most examples investigated in this work are represented by three-dimensional graphs with the x and y axes formed by the range of values for two independent variables and the z axis the optimal solution to the given optimization problem, respective of the two independent variables. In the case of power minimization, the two independent variables are resolution and velocity, consistently ranging from 0.5 m to 2.0 m and 50 m/s to 400 m/s, respectively. In the velocity maximization problem, power and resolution are the independent variables, with the same resolution range as noted above and a power range of 30 w to 100 w. Similarly, in the resolution minimization problem, power and velocity comprise the independent variable set with the same range of values for both variable as just noted above for the two other optimization problems. In each case, solution values on the z axis correspond to linear samplings on the x and y axes such

that 25 points of equal intervals on each axis are generated. As a result, each graph represents $25^2 = 625$ separate optimizations.

CHAPTER V

IDEAL SHARED-MEMORY MODEL

The first COTS model to be investigated considers the two Mercury RACE daughtercards but assumes no penalty for communication between CNs physically located on different cards. After calculating the system requirements, the optimal number of cards is found by dividing the system requirements by the processing and memory capacity of the cards. Such an approach ignores communication costs for remote memory accesses and treats the aggregate throughput of all the processors as a single entity. Although vendors often advertise total throughput and memory to characterize a system, real performance rarely can be modeled accurately based on such a proposition.

Research shows that the only significant flaw in this assumption in regards to SAR processing on the Mercury RACE is the communication costs for remote memory accesses. It is theoretically feasible that a problem would call for an optimal FFT size too large for a single processor to handle and thus would require a distributed FFT algorithm, which would change the entire model. However, test cases with real data show that this case does not occur within the scenarios studied in this research.

Therefore, although this approach is unrealistic for determining the real requirements of a Mercury RACE or similar system, this model reflects the per-

formance of an ideal shared-memory system (henceforth, this model is referred to as the ISMM). Furthermore, the fundamental nature of the problem and optimal solutions are illustrated more clearly with such a model. The addition of real constraints to ensure solutions of considerable fidelity tends to obscure lower level dynamics of the problem.

5.1 Minimization of Power

Minimization of power is the simplest of the optimization problems because of the variables that can be fixed before the optimization algorithm is invoked. All the range variables K_r , S_r , F_r , P_r , and M_r are functions of only radar, processor, and application parameters. Among the azimuth variables, only K_a can be fixed, a function of radar parameters and resolution. The other azimuth variables are dependent on the the section size S_a , which is an optimization variable, and therefore cannot be calculated statically outside the optimization algorithm.

Because the range variable values will remain the same for each of the power minimization problems, it is useful to examine them only once initially. F_r is a function of K_r and S_r , only assuming values equal to integer powers of two. K_r and S_r are linear functions of fixed radar parameters and resolution (Eqns. 2.8 and 3.1) and therefore will be planes that decrease as resolution becomes coarser. The graph of F_r is shown in Fig. 5.1. The figure illustrates F_r as a step function, assuming only the following values: 16384, 32768, and 65536. Note that F_r is

independent of velocity.

The number of processors and megabytes of memory necessary for range processing is represented in Figs. 5.2 and 5.3. Both these parameters can be considered as constants added to the azimuth processor and memory requirements. The average number of range processors required is 16.8, with a minimum of 0.85 and a maximum of 121.7. The plot of the required range memory displays a similar trend with an average of 7.2, a minimum of 0.14, and a maximum of 77.9 MB. It is observed that both these graphs exhibit rapid growth, increasing as resolution becomes finer. This observation is confirmed by Eqns. 3.2 and 3.5, which show δ^2 in the expression for P_r and δ^3 in the expression for M_r .

Because the azimuth variables are the only variables that can be optimized in the power minimization problem, unless otherwise specified, variables such as FFT kernel size, section size, and FFT size will refer to the azimuth parameters as opposed to the range parameters. These variables are discussed and graphed below.

5.1.1 Optimal Mixed Card Type Configuration

Let C_1 and C_2 denote the number of S2T16B and S1D64B cards utilized, respectively. Thus the function for total consumed power (in watts) is defined

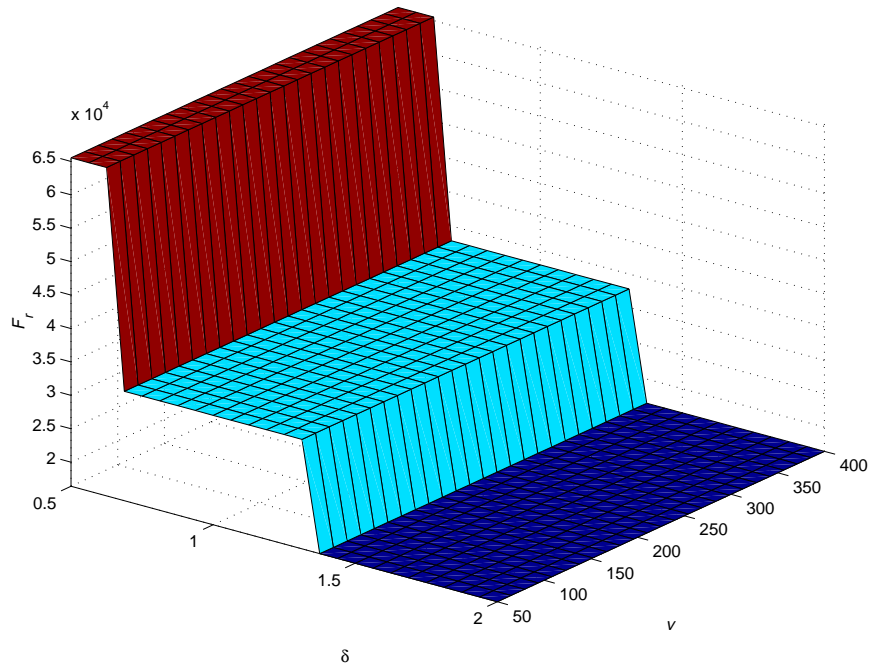


Fig. 5.1: Range FFT size for power minimization.

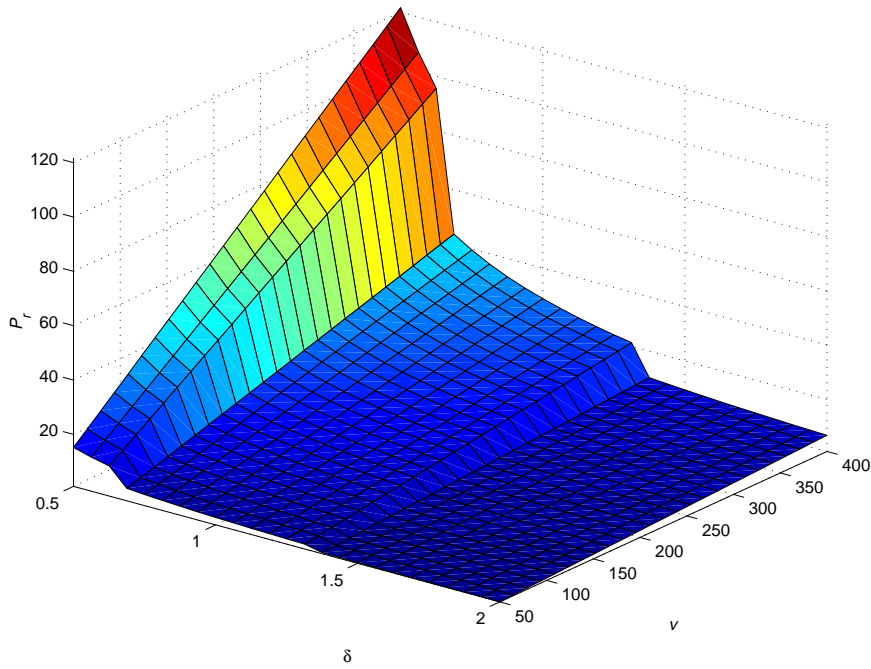


Fig. 5.2: Range processor requirements (in number of processors) for power minimization.

as

$$Z = 12.2C_1 + 9.6C_2, \quad (5.1)$$

reflecting the total power requirements of the two boards. Next, two required constraint equations naturally follow based on the values of $P(S_a)$ and $M(S_a)$:

$$6C_1 + 2C_2 \geq P(S_a) \quad (5.2)$$

$$32C_1 + 64C_2 \geq M(S_a). \quad (5.3)$$

These two constraint equations ensure that the total number of processors in the configuration is no less than the total number of required processors and the total amount of memory in the configuration is no less than the total amount of memory required. In this framework, values for the parameters C_1 and C_2 must be optimized in addition to the value of the parameter S_a . Although the parameter S_a does not explicitly appear in the objective function that is to be minimized (i.e., Z), its effect is implicit through the constraint equations. That is, the optimal values for C_1 and C_2 are contingent on some calculated value of S_a .

The only discontinuous portion in the formulation is due to the definition of F_a , which is a discontinuous function of S_a . (Recall that F_a is defined as the

smallest integer power of two that is greater than $S_a + K_a$.) This discontinuous function prevents the direct application of nonlinear programming. However, by selecting F_a as an integer power of two, and adding a constraint to ensure that $K_a + S_a$ is no greater than this selected value, the discontinuity can be removed. Thus, in addition to the constraints given by Eqns. 5.2 and 5.3, the following constraint equation is added:

$$K_a + S_a \leq F_a, \tag{5.4}$$

where the value of $F_a = 2^k$ is fixed (the value of K_a is known based on the values of the specified basic parameters). The discrete nature of F_a constitutes an integer programming problem and precludes the direct application of nonlinear programming techniques with F_a as an optimization variable. Thus, to ensure optimality it may be necessary to solve several constrained optimizations based on different feasible values for F_a . In practice, however, only a few values for F_a need to be tried: from the smallest feasible value (2^k where $k = \lceil \lg K_a \rceil$) up to the point at which the optimal value of S_a is such that $K_a + S_a < F_a$ (i.e., the constraint becomes inactive). The inactiveness of the constraint suggests that every value of F_a greater than that last tried would produce an increasingly worse solution. This phenomenon is true because if S_a does not inflate to its maximum value given the constraint (i.e., $F_a - K_a$), memory is in shortage and

will become more in shortage with every increment in the FFT size.

With the addition of constraints that ensure that optimization variables are greater than zero ($S_a \geq 1, C_1 \geq 0, C_2 \geq 0$), Eqns. 5.1–5.4 constitute the first constrained nonlinear and integer optimization problem solved in this work. Representative samples of the MATLAB code used to solve all the optimization problems and produce the data are included in the Appendix.

Fig. 5.4 represents the total power consumption of the ISMM for a range of resolution and velocity pairs. As would be expected, more power is required for higher velocities and finer resolutions. However, it is noted that resolution has a more dramatic effect on power consumption than does velocity. The graph is smooth except for several almost imperceptible ridges at resolution values of approximately 0.65 m, 0.91 m, 1.28 m, and 1.78 m. When the power graph is compared to the graph for F_a (Fig. 5.5), the cause of the anomalies becomes apparent. The ridges result from the discontinuous nature of F_a , as described above. For this set of resolution and velocity values, optimal F_a values range from 512 to 8192 points, corresponding to coarse and fine resolutions, respectively. This finding supports the observation that resolution requirements dominate system performance, and fine resolution demands high memory usage, which in turn drives power consumption high, even in the case of very low velocity. In this scenario, at fine resolutions, relatively inefficient data processing is being performed because memory is in shortage, entailing a surplus of processors.

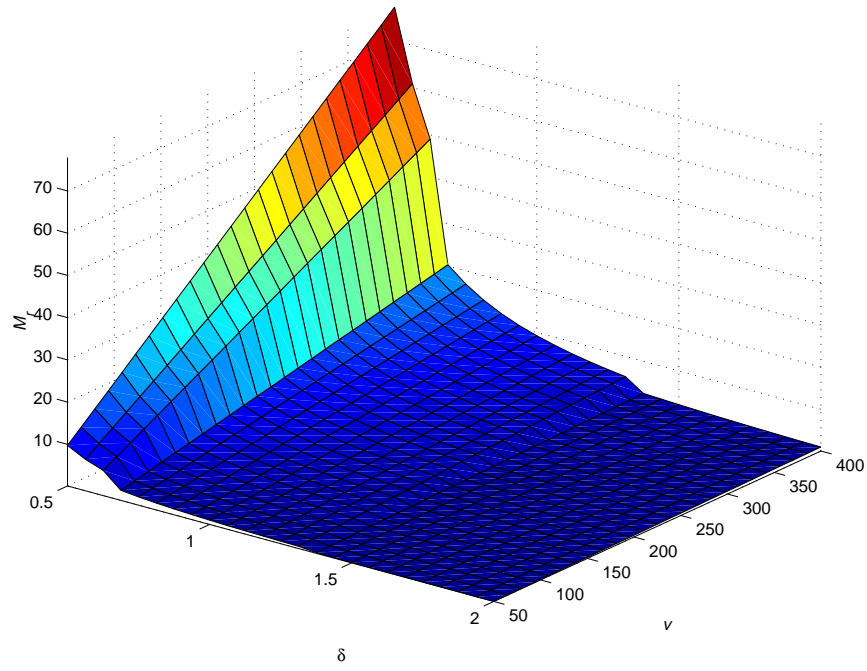


Fig. 5.3: Range memory requirements (in MB) for power minimization.

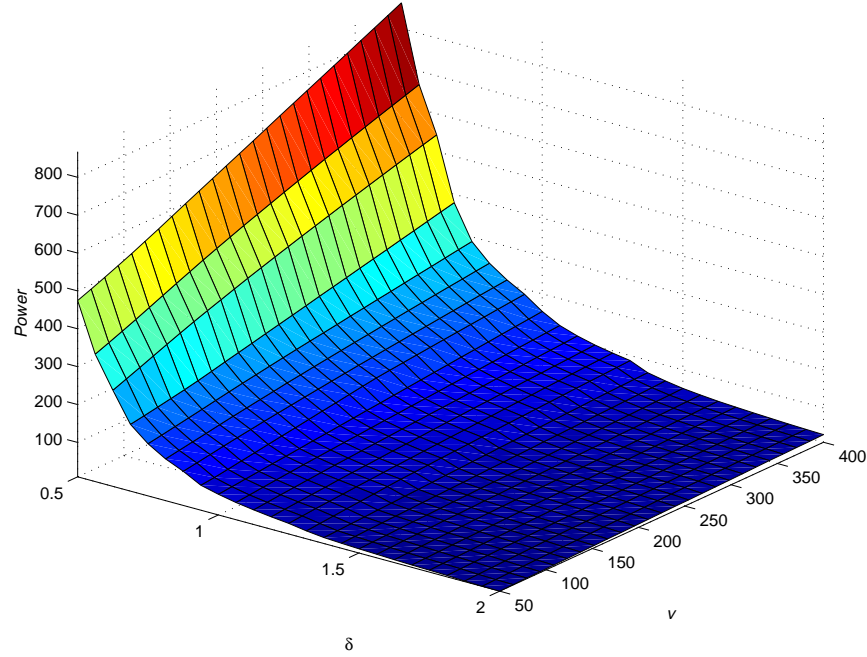


Fig. 5.4: Optimal power consumption.

Although most of the attention paid to explaining Fig. 5.4 will be in terms of the role of azimuth processing requirements, to understand all the intricacies of the graph the role of range processing also must be taken into consideration. As already noted, Figs. 5.3 and 5.2 illustrate the requirements of range processing. Similarly, Figs. 5.6 and 5.7 show the graphs of the azimuth memory and processor requirements. Note that for the power minimization model, although the range requirements remain constant for different configurations, the azimuth requirements change according to the optimally computed S_a . Analysis of the ratio of azimuth requirements to range requirements therefore is useful. Figs. 5.8 and 5.9 represent these ratios for memory and processors. It is obvious that the disparity between azimuth and range memory is much greater than that of processor requirements. The ratio of azimuth to range processor requirements varies from 0.9:1 to 7.2:1, the lower ratio entailing a larger range processor requirement than that for azimuth. However, the ratio of azimuth to range memory requirements varies from 59:1 to 648:1, a minimum disparity of almost sixty times the amount of required memory for azimuth than for range processing, even at the few points where more range processors are required than azimuth processors.

A general statement can therefore be made that azimuth requirements always dominate a power minimization configuration (for the given range of resolution, velocity, and radar parameters). Furthermore, every visible ripple in the power consumption graph of Fig. 5.4 can be accredited to discontinuities in azimuth

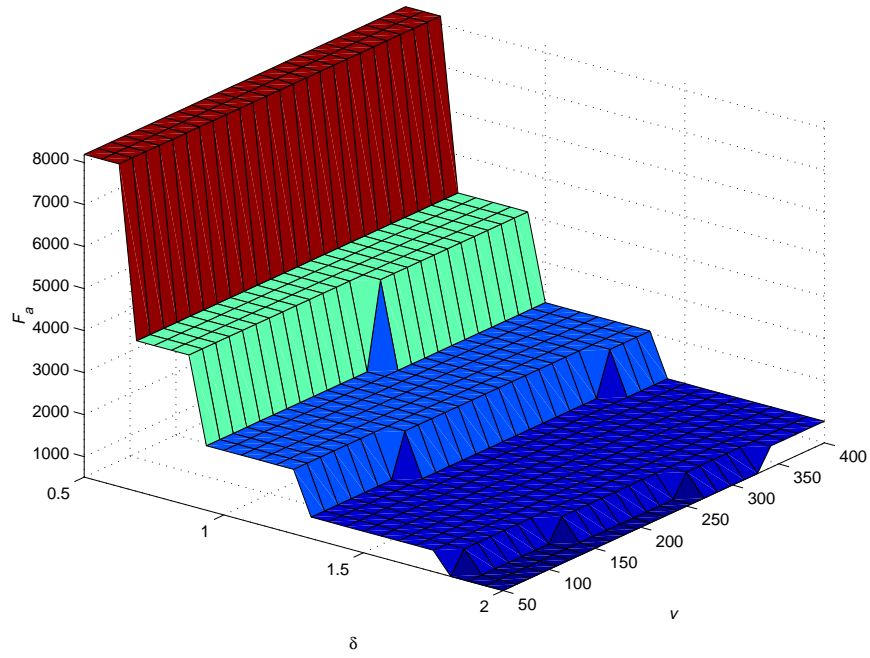


Fig. 5.5: Optimal azimuth FFT size for power minimization.

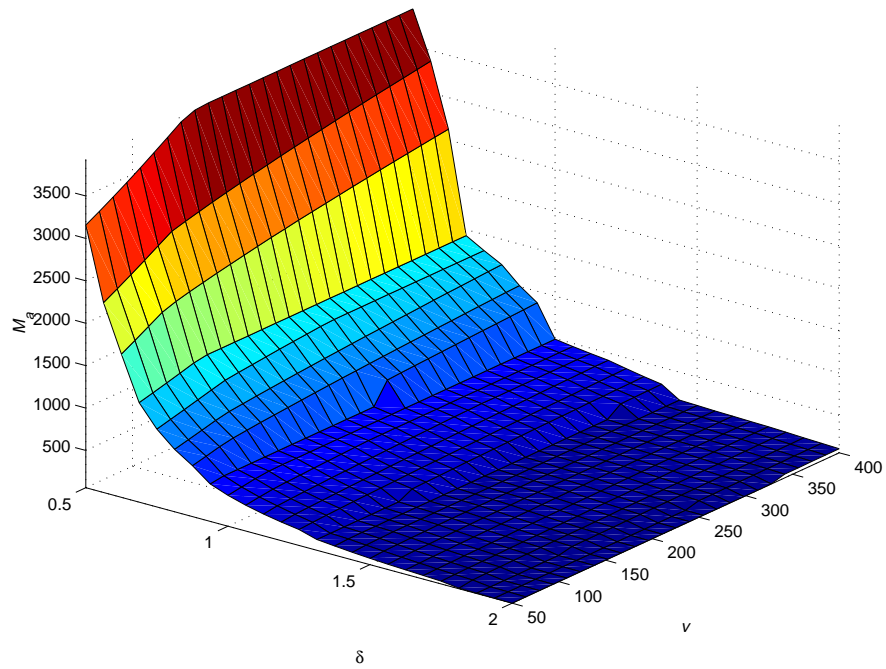


Fig. 5.6: Optimal azimuth memory requirements for power minimization.

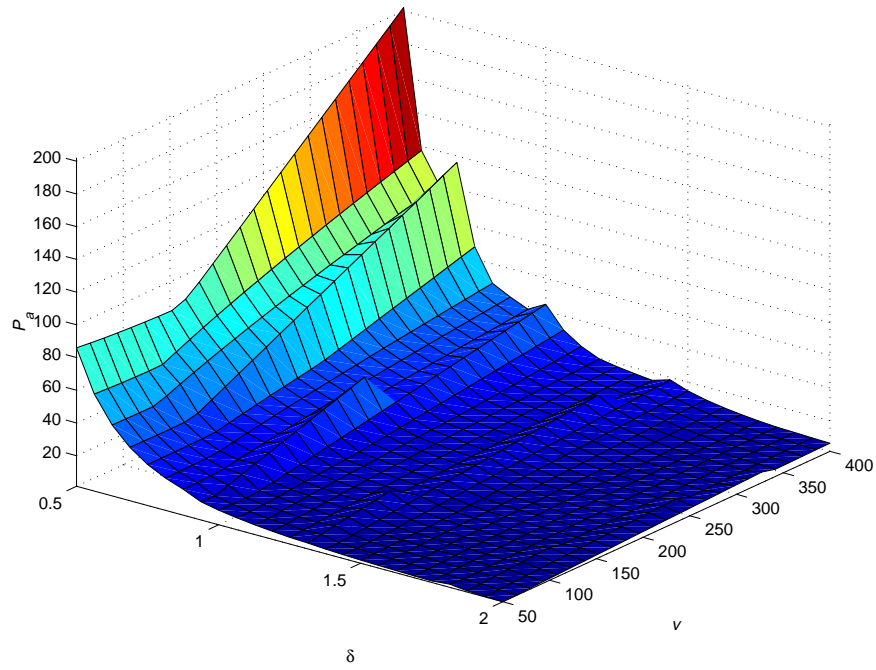


Fig. 5.7: Optimal azimuth processor requirements for power minimization.

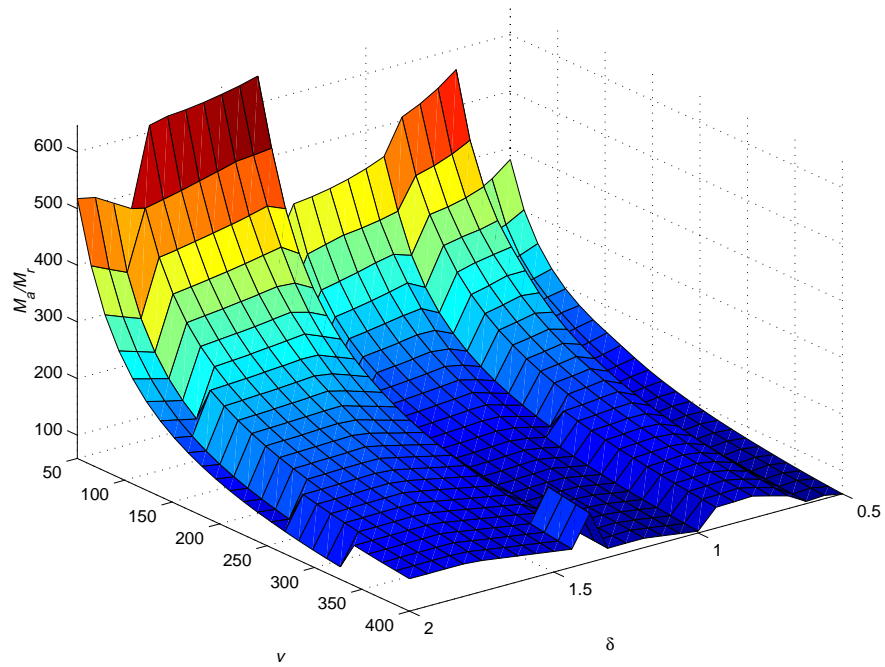


Fig. 5.8: Optimal ratio of azimuth to range memory requirements for power minimization.

requirements because the discontinuities in range requirements correspond spatially to discontinuities in azimuth requirements. It might seem that if the power graph is to be analyzed primarily in terms of azimuth requirements, then the power consumption by range requirements should be subtracted from the total power requirements before analysis. This method is implausible, however, because the power consumption can not be measured strictly by the product of the requirements and some constant representing the power per megabyte or power per processor, as was theorized in the custom-VLSI model of Section 4.3.1. Adherence to both the processor and memory constraints of Eqns. 5.2 and 5.3 leads to taking the maximum of the daughtercards required by both constraints to determine total power consumption. Consequently, power consumption by only range or azimuth processing has no meaning because optimization of the azimuth section size automatically seeks to utilize all available resources, which are dependent on the range requirements. Therefore, throughout the rest of the power minimization model, knowledge of the range requirements and the impact they have on total power requirements should be considered, but discussion of variables will be limited to the azimuth requirements because they are the values of optimization.

As is expected from Eqn. 3.7, the graph of K_a (Fig. 5.10) is completely smooth, increasing as resolution becomes finer, and independent of velocity. The graph of S_a (Fig. 5.11), however, is more interesting. S_a is nondecreasing in the

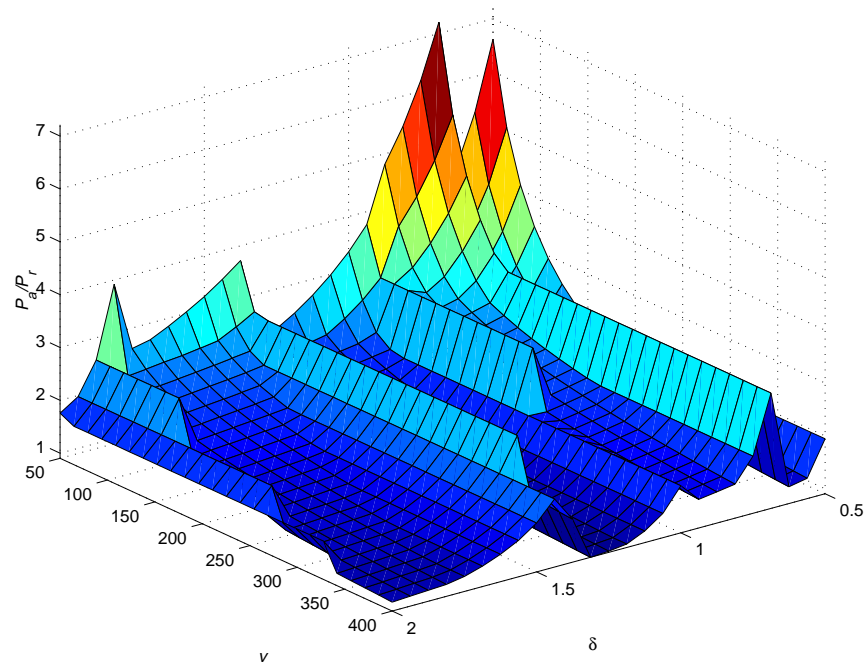


Fig. 5.9: Optimal ratio of azimuth to range processor requirements for power minimization.

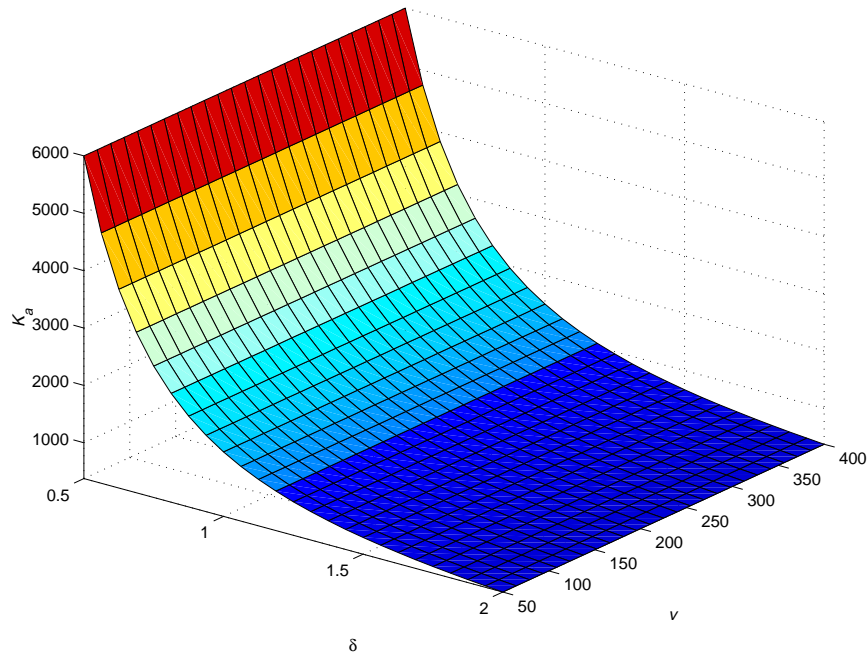


Fig. 5.10: Optimal azimuth FFT kernel size for power minimization.

velocity dimension, but undulates in the resolution dimension. This rippling effect results from the graph of F_a . The tiers of the F_a graph determine the discontinuities of the S_a graph. As resolution becomes finer, S_a also decreases to compensate for the additional memory that is required by the resolution. When the processing becomes too inefficient, the next value of F_a becomes optimal and evokes a corresponding increase in S_a . Resolution demands again necessitate reductions in S_a to save memory and utilize processors until the next value of F_a becomes optimal. However, notice that in Fig. 5.12, which represents the ratio of S_a to F_a , overall S_a gradually decreases as a proportion of F_a as resolution becomes finer. As a result of the gradual decrease in this ratio, there is a corresponding decrease in the computed optimal employment of the processor-rich (S2T16B) boards to the memory-rich boards (S1D64B). This trend is illustrated in Figs. 5.13 and 5.14.

Surprisingly, note that velocity seems to have a more dramatic effect on the card type utilization than does resolution. However, recall that the two card types vary by a factor of two in memory capacity but vary by a factor of three in processors. The undulations in both graphs again result from the discontinuities in the graph of F_a , but the effect of the F_a discontinuities is very transient, resulting in spikes that quickly return to the general shape of the graph.

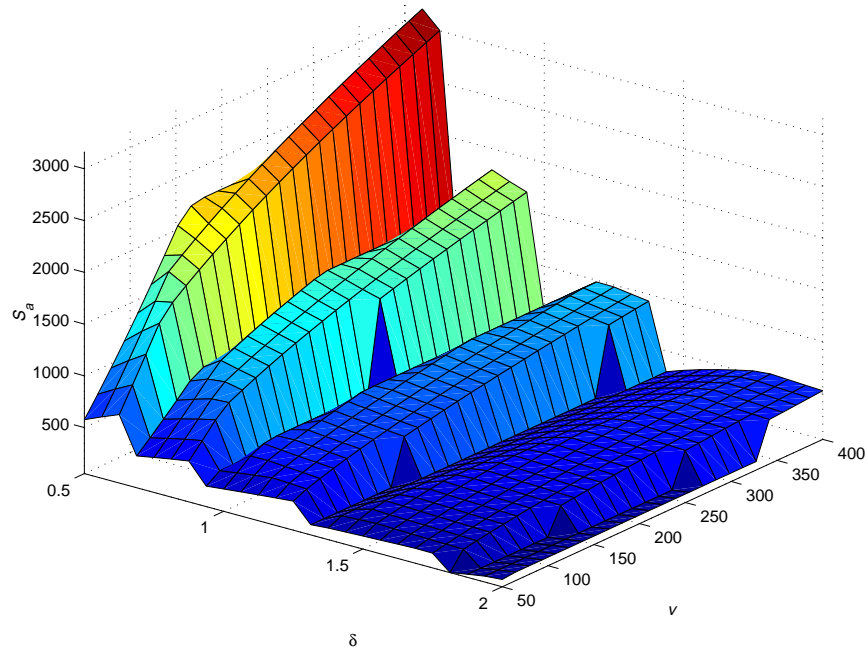


Fig. 5.11: Optimal azimuth section size for power minimization.

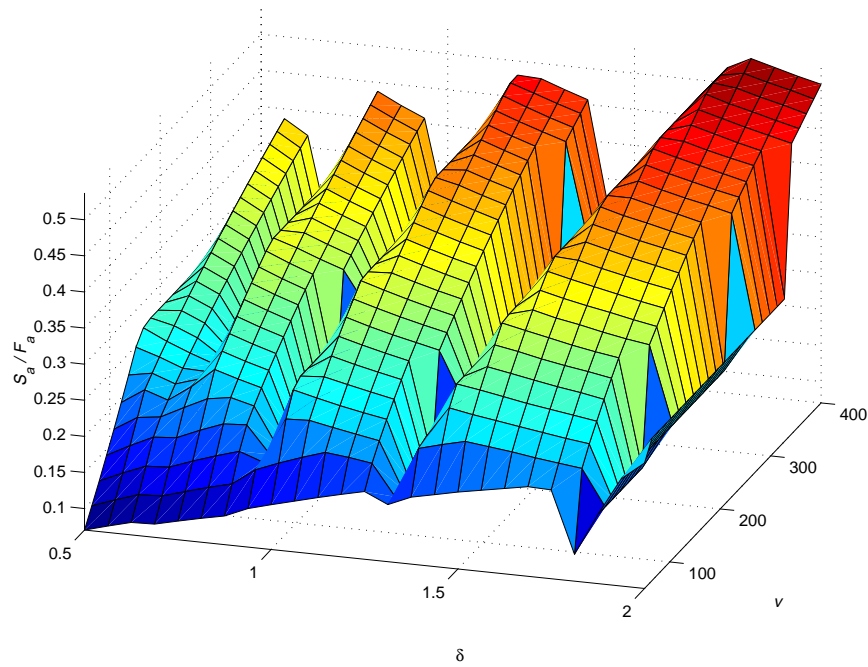


Fig. 5.12: Optimal ratio of azimuth section size to FFT size for power minimization.

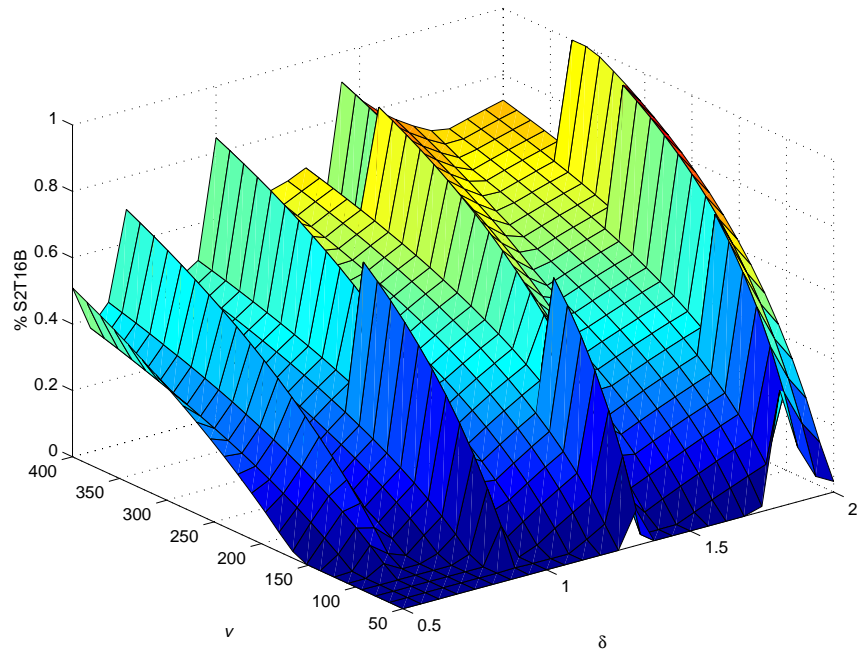


Fig. 5.13: Optimal percentage of power usage by the S2T16B for power minimization.

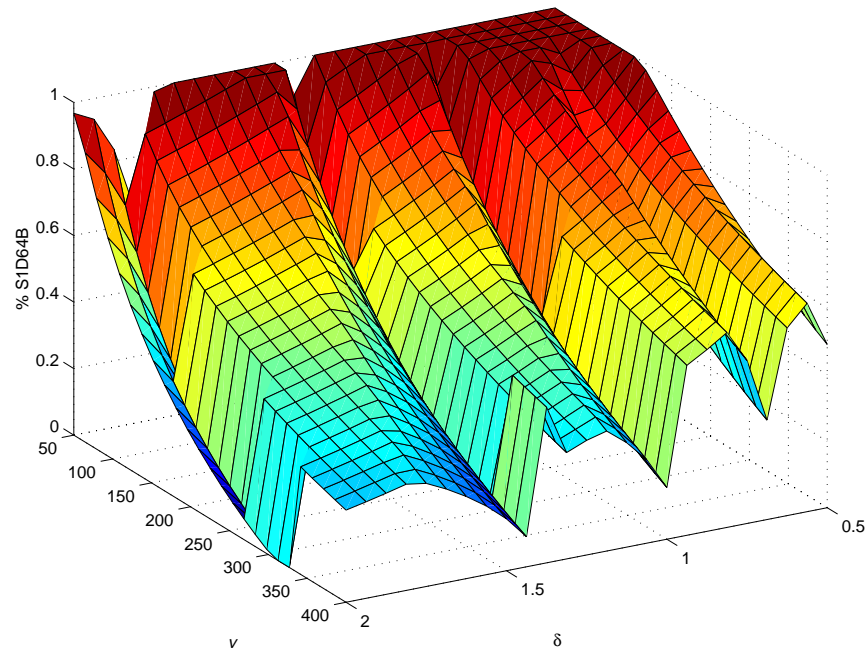


Fig. 5.14: Optimal percentage of power usage by the S1D64B.

5.1.2 Optimal Single Card Type Configuration

In the case that only one or the other daughtercard type is available for system configuration, the optimization problem is easily adapted to accommodate this tighter constraint. The generalized objective function becomes

$$Z = \Pi_d(C_T), \quad (5.5)$$

where Π_d denotes the power consumption per daughtercard as a function of the card type and C_T is the card type. Similarly, the constraint equations become:

$$CP_d(C_T) \geq P(S_a) \quad (5.6)$$

$$CM_d(C_T) \geq M(S_a), \quad (5.7)$$

where C is the number of cards employed, and P_d and M_d are the number of processors and amount of memory available as functions of the daughtercard type. All other constraints remain the same. Solving this problem for both card types produces the power consumption graphs of Figs. 5.15 and 5.16. Fig. 5.15 for the S2T16B is a much smoother graph than that of the S1D64B in Fig. 5.16.

There is a noncoincidental resemblance between Fig. 5.15 and the perfectly smooth curled plane of K_a (note that the graph of K_a is the same for every configuration involving optimal power with resolution and velocity fixed). As K_a increases, so does the card requirement. However, Fig. 5.16 depicts a less smooth

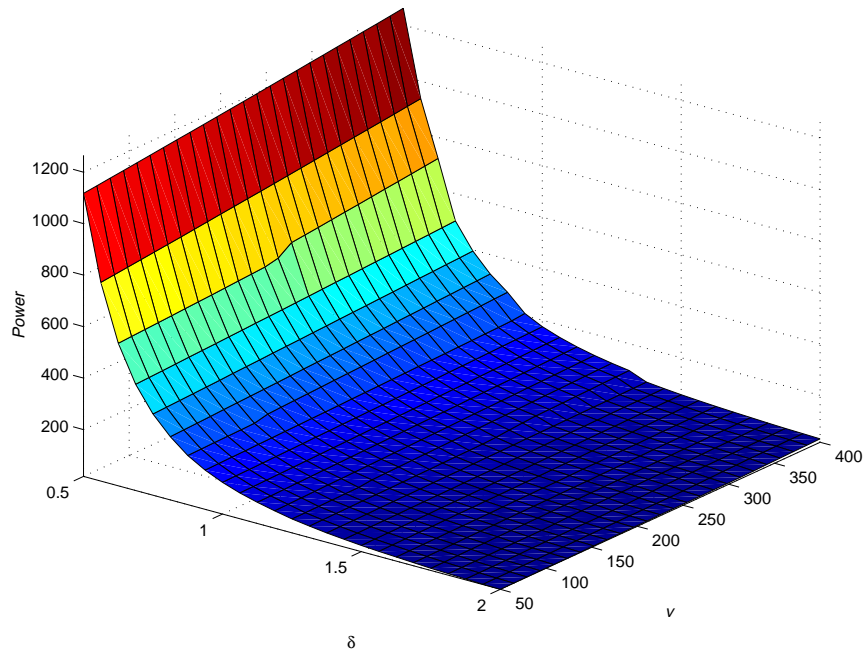


Fig. 5.15: Optimal power consumption in S2T16B-only configuration.

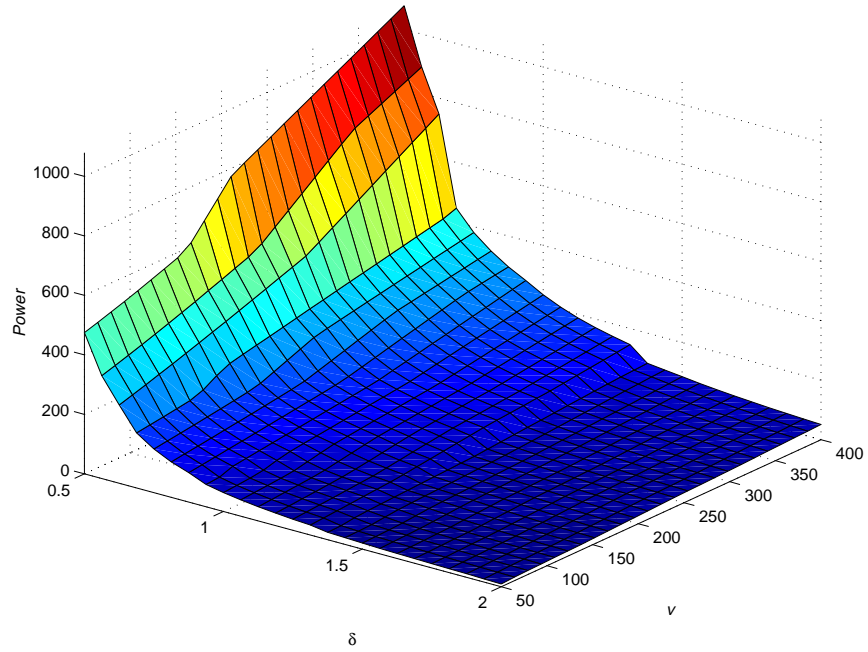


Fig. 5.16: Optimal power consumption in S1D64B-only configuration.

function. Obviously, the S1D64B configuration depends on more than just K_a . This difference is explained by an examination of the resource utilizations of both configurations. In both cases, there is a 100% processor utilization. The S2T16B similarly has an average 99.7% memory utilization. In contrast, average memory utilization in the S1D64B configuration was only 90.5%, seemingly low for an optimal solution. The low memory usage in the latter case is a consequence of the more extreme ratio of memory to processors in the S1D64B, having one-third as many processors but twice as much memory as the S2T16B. Thus, in regards to resource utilization, the processor-rich S2T16B is better suited for the range of resolution and velocity pairs in this investigation.

Resource utilization, however, was not the goal of the optimization problem. It seems reasonable to assume that efficient resource utilization would entail low power consumption, but that is not necessarily the case. As is observed from examining Figs. 5.15 and 5.16, peak power consumptions were 1265 and 1079 w for the S2T16B and S1D64B configurations, respectively. Similarly, average power consumption was 213.2 and 164.3 w. The S1D64B, with its poorer memory utilization, consumed an average 29.8 w less than the S2T16B configuration.

Such statistics can be misleading, however, if overgeneralized. If it is necessary to employ only one type of card in a system, the S1D64B is not necessarily a better choice. As seen by Figs. 5.17 and 5.18, there is a clear demarcation of the areas where each card type is most appropriate. Fig. 5.17 shows the per-

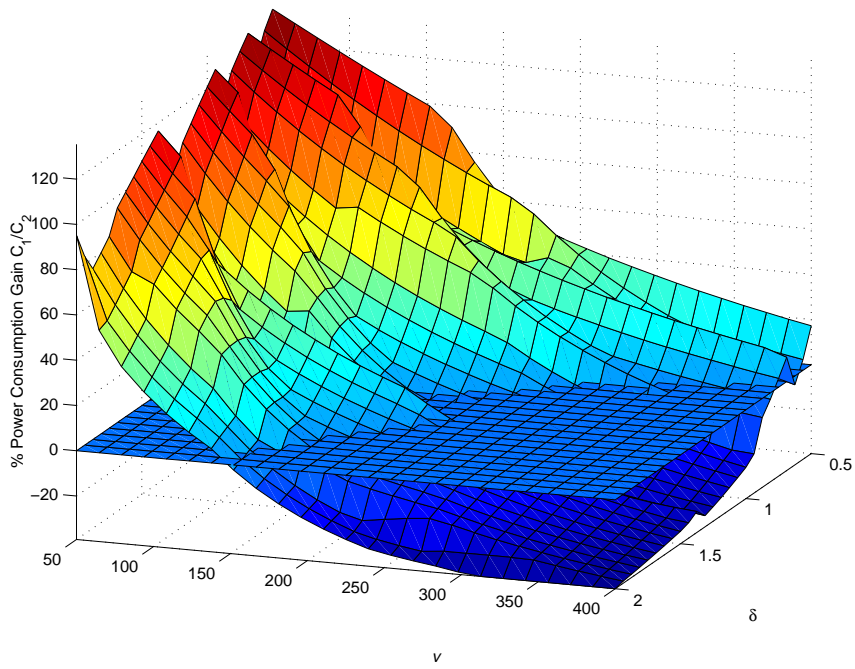


Fig. 5.17: Percentage power gain and loss of S2T16B-only over S1D64B-only configurations. Positive values therefore indicate that the S1D64B or the S2T16B is better-suited, respectively.

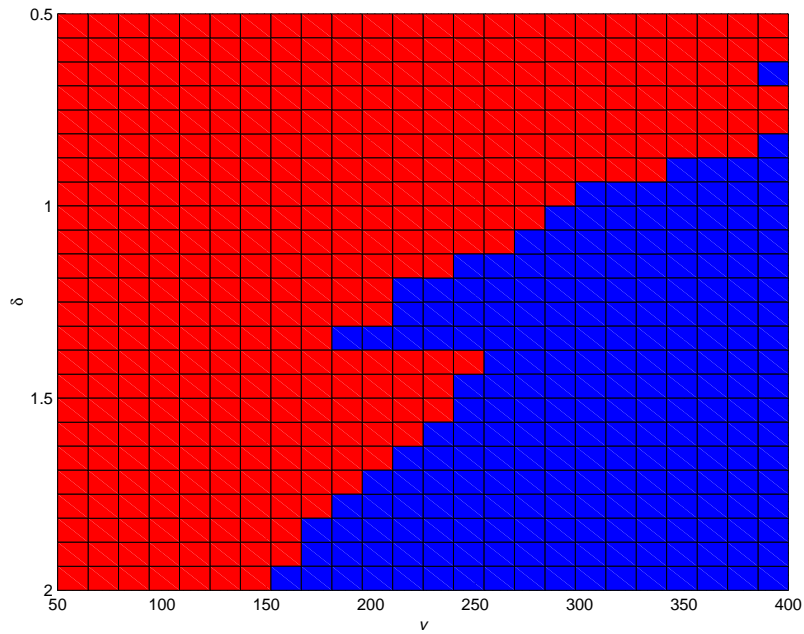


Fig. 5.18: Red and blue areas represent lowest power consumption either by S1D64B or S2T16B configurations, respectively.

cent gain in power consumption of employing the S2T16B over the S1D64B. The plane running through the graph marks zero percent gain. Everywhere above the plane therefore signifies that the S1D64B card is more efficient, noting a gain in power over the S1D64B. Similarly, areas of the graph below the plane denote better performance by the S2T16B. Fig. 5.18 represents the surface formed in Fig. 5.17 as a binary function, with blue denoting gains in power and red losses (improvements) in power. The S1D64B provides up to a 135% decrease in power and at worst consumes up to 39% more power. However, the S1D64B is better-suited for 59% of the cases considered.

Clearly, the required resolution and velocity determine which card is most appropriate in a single-card type system. The two extremes of the card type power consumptions occur at the extremes of the resolution and velocity graph. The S1D64B's advantage is most apparent at the highest performance scenario—where velocity is at a peak (400 m/s) and resolution is finest (0.5 m). Conversely, the S2T16B outperforms the S1D64B most drastically in the low performance scenario—where velocity is lowest (50 m/s) and resolution is coarsest (2 m).

5.1.3 Nominal Mixed Card Type Configurations

It has been suggested that using an azimuth section size (S_a) equal to the kernel size (K_a) is a good heuristic for adequate performance with moderate conservation of memory, which is usually the scarce resource [7]. The optimiza-

tion problem is simplified by removing S_a from the optimization variables and setting it equal to K_a . The third constraint (Eqn. 5.4) is also removed, resulting in only one meaningful value for F_a ($F_a = 2^{\lceil \lg(2K_a) \rceil}$). Fig. 5.21 graphs the power consumption of a system using the section size heuristic yet still optimizing the number of cards of each type. For the range of values tested in this investigation, it was found that for 91.5% of the cases, the optimal kernel to section size ratio is larger than the 1:1 ratio associated with the heuristic. Fig. 5.20 shows the ratio of the kernel size to the optimal section size. The average ratio in this scenario is 2.24 with a minimum of 0.72 and maximum of 10.42. Consequently, there is a substantial increase in power requirements of the nominal configuration. Adaptation of the number of cards of each type by the optimization routine keeps the increase from attaining the ten fold that might otherwise occur if the same card type ratio was maintained from the optimal to the nominal configuration. Nevertheless, Fig. 5.21 shows a power increase from 0–82.3% with an average of 19.4%. The 0% increase occurs where the optimal section size happens to be equal to the kernel size and the 82.4% increase intuitively occurs in the area where the optimal $K_a : S_a$ ratio is highest, where velocity is at a minimum and resolution finest.

The entire region where velocity is low exhibits extreme improvements for the optimal section size. At first glance, this seems to be a surprising result because resolution and memory requirements usually dominate power requirements. This

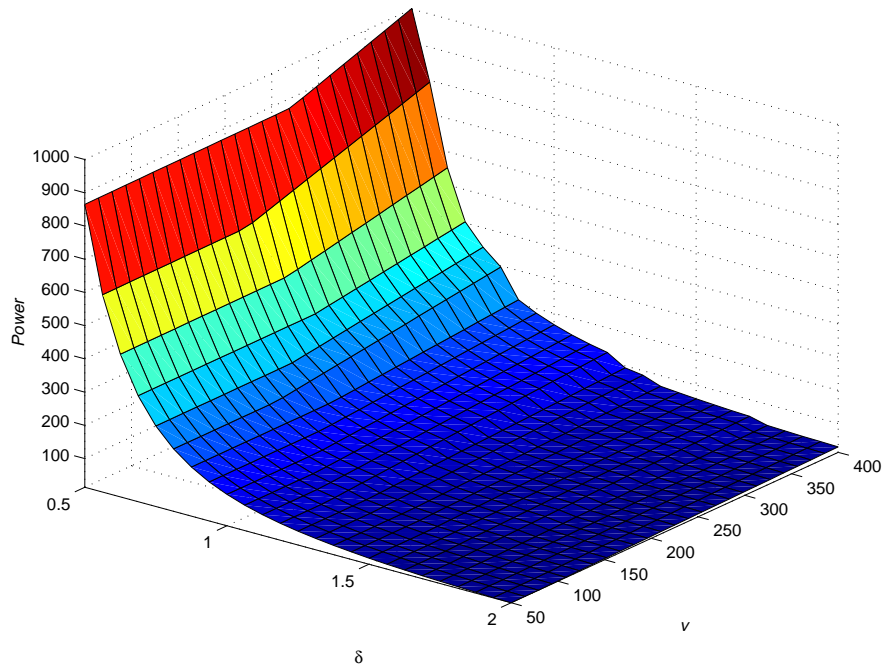


Fig. 5.19: Optimal card type configuration and nominal section size for power minimization.

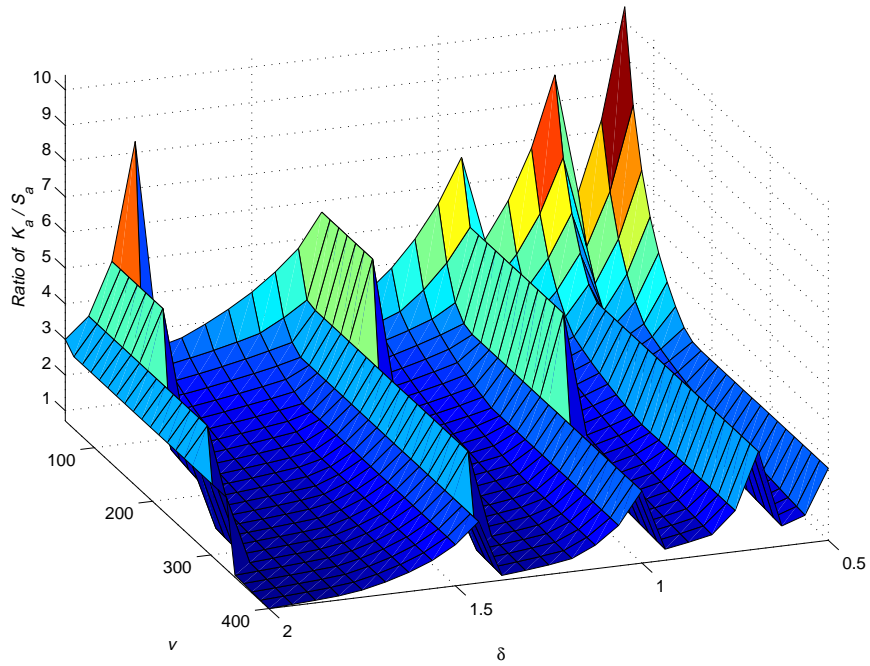


Fig. 5.20: Ratio of azimuth kernel size to optimal section size for power minimization.

rule remains true in this case also but indirectly. At lower velocities, processing power becomes less crucial. A nominal section size results in a surplus of memory. The optimal section size lowers the section size, resulting in less efficient processing but more efficient memory usage. The undulations in the surface of Fig. 5.21 correspond to the rippling nature of S_a (Fig. 5.11).

5.1.4 Nominal Single Card Type Configurations

To complete the comparison of optimal and nominal section sizes in both single and mixed card type configurations, nominal single card type configurations are now investigated. As expected, this configuration requires the highest power. Figs. 5.22 and 5.23 represent the power consumption graphs of the two single card type configurations. The S2T16B graph now follows K_a (Fig. 5.10) even more closely than its optimal section size counterpart. This resemblance is due to the total lack of configuration optimization. In the nominal mixed card type configuration, C_1 and C_2 are still optimization variables. In the single card type configuration, C is calculated by the following formula:

$$C = \max \left\{ \frac{P(K_a)}{P_d(C_T)}, \frac{M(K_a)}{M_d(C_T)} \right\}. \quad (5.8)$$

Therefore memory is always the active constraint for the memory-poor S2T16B configuration. That is, the right side of Eqn. 5.8 is always dominant. However,

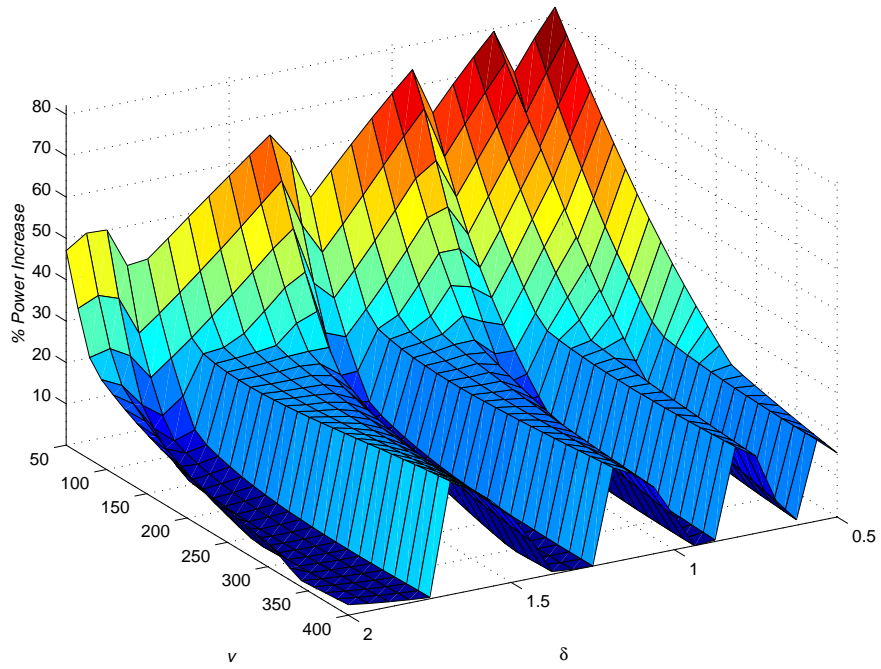


Fig. 5.21: Percentage increase in power of nominal section size over optimal section size configuration.

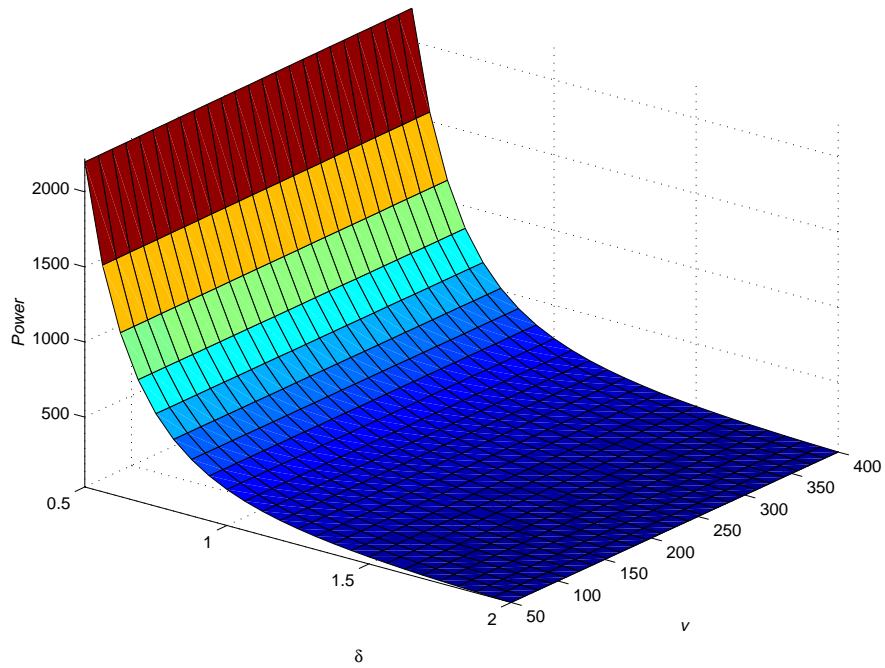


Fig. 5.22: Power consumption of nominal single card type configuration using the S2T16B.

Fig. 5.23 still exhibits sharp points. The memory-rich S1D64B configuration is still susceptible to both memory and processor constraints, being processor bound in 73.1% of the cases and memory bound the other 26.9% of the cases at low velocities.

5.1.5 Summary of Power Minimization Models

Fig. 5.24 compares the six possible configurations discussed so far. The velocity is fixed at 298 m/s. Fig. 5.25 similarly compares the six configurations but with the resolution fixed at 0.875 m. As expected, the optimal mixed configuration requires the least power for all values of resolution and velocity. Table 5.1 summarizes the comparison across all values.

5.2 Maximization of Velocity

All models presented thus far have considered the minimization of power consumption as the objective function. In this section the maximization of velocity v for a given system is investigated. It is assumed that the resolution δ and either available power or the number of daughtercards of each type are the independent variables.

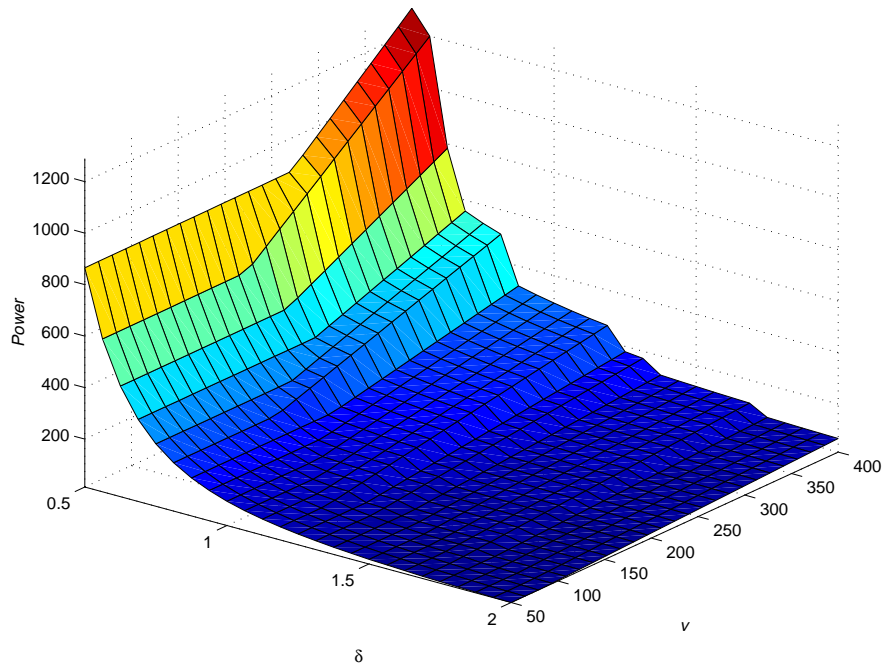


Fig. 5.23: Power consumption of nominal single card type configuration using the S1D64B.

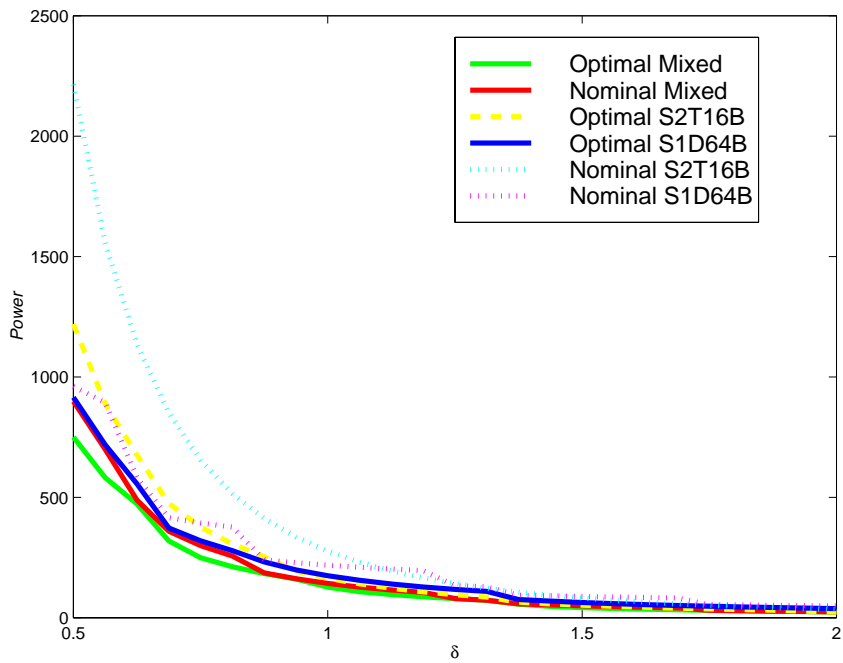


Fig. 5.24: Comparison of power consumption of six configurations with velocity fixed at 298 m/s.

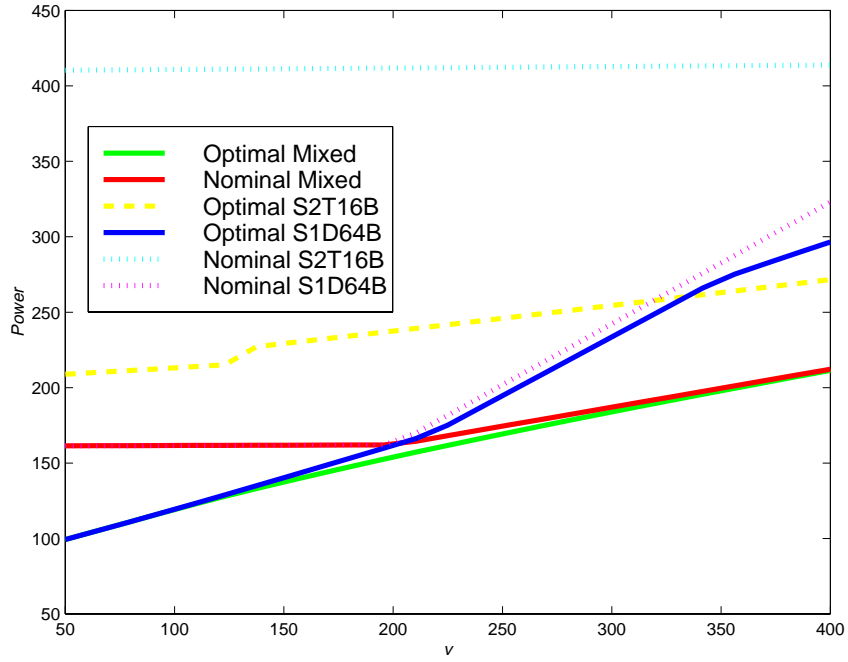


Fig. 5.25: Comparison of power consumption of six configurations with resolution fixed at 0.875 m.

Table 5.1: Comparison of configurations showing the minimum, maximum, and average power and the percent increase of each statistic over the power consumption of the optimal mixed configuration.

Configuration	Min.	% Inc.	Max.	% Inc.	Avg.	% Inc.
Optimal Mixed	9.192	–	867.6	–	135.5	–
Nominal Mixed	13.52	47.05	1002	15.49	166.8	23.13
Optimal S2T16B	17.96	95.37	1265	45.82	213.2	57.40
Optimal S1D64B	9.203	.1134	1079	24.41	164.3	21.28
Nominal S2T16B	34.36	273.8	2221	156.0	379.1	179.9
Nominal S1D64B	13.52	47.05	1289	48.61	206.3	52.30

5.2.1 Set Power with Variable Number of Cards

Fixing power still leaves the number of each card type to be maximized. The formulation for velocity maximization is very similar to that of power minimization. The objective function is simply the maximization of the following:

$$Z = v. \tag{5.9}$$

The constraint equations are also similar to the power minimization model except for the addition of a power constraint that is almost identical to the objective function in the power minimization model. This additional constraint is as follows:

$$\Pi \geq 12.2C_1 + 9.6C_2, \tag{5.10}$$

where Π represents the power allocated for the system. The optimization problem is slightly more complex than in the case of minimizing power because P_r , P_a , and M_r are all functions of v . Therefore both S_a and v are implicit in the constraint equations. Following the convention set forth, Eqns. 5.2 and 5.3

become

$$6C_1 + 2C_2 \geq P(S_a, v) \quad (5.11)$$

$$32C_1 + 64C_2 \geq M(S_a, v). \quad (5.12)$$

In addition, the following lower bound is added:

$$v \geq 0. \quad (5.13)$$

Eqn. 5.10 could be expressed as an equality constraint because fractional numbers of cards are not disallowed in this formulation. The inequality expression is left, however, because the optimization algorithm always finds a solution utilizing all available power. Furthermore, the inequality constraint is more correct in the generalized form of the problem if C_1 and C_2 are forced to be integers.

5.2.1.1 Optimal Mixed Card Type Configuration

The graph of the maximum attainable velocity given a set power and resolution is shown in Fig. 5.26. The expressed maximum velocity at high power and fine resolution is probably impracticably high for a real airborne UAV, but such speeds might be realistic for a spaceborne satellite, although other parameters in the radar system would probably change and necessarily the range R .

Fig. 5.27 illustrates the different geometry of the solution space in reference to the power minimization problem. The three plateaus are at FFT sizes of 1024, 2048, and 4096. About one third of the graph is missing (32.0%) because there was no feasible solution to the given power-resolution pair. The boundary of infeasibility in this scenario runs roughly on the line where resolution equals 1.0. Recalling that azimuth memory is defined by an expression with δ^3 in the denominator (Eqn. 3.8), the 1.0 resolution boundary is logical. Increasing the power range would provide at least some feasible solutions for all resolutions. However, the maximum attainable velocity at coarse resolutions would become unreasonably high for the given scenario because the velocity already exceeded 1800 m/s (over Mach 5) in Fig. 5.26. Note that the graph of K_a is the same as for the power minimization problem (Fig. 5.10). Fig. 5.28 shows the optimal section size. Each point of inflection corresponds to a jump in the FFT size.

The plot of the optimal S2T16B usage for maximum velocity is shown in Fig. 5.29. The plot for the S1D64B can be easily visualized by turning the graph upside down, or taking one minus the graph for the S2T16B. The high plateau in Fig. 5.29 corresponds to the low plateau in the graph of the FFT size (Fig. 5.27). When the optimal FFT size was low, implying a great quantity of processing, the processor-rich S2T16B became the exclusively ideal choice. Outside of this region, a mixture of the two cards was optimal, with the S2T16B usage generally increasing both as resolution became coarser and available power increased. For

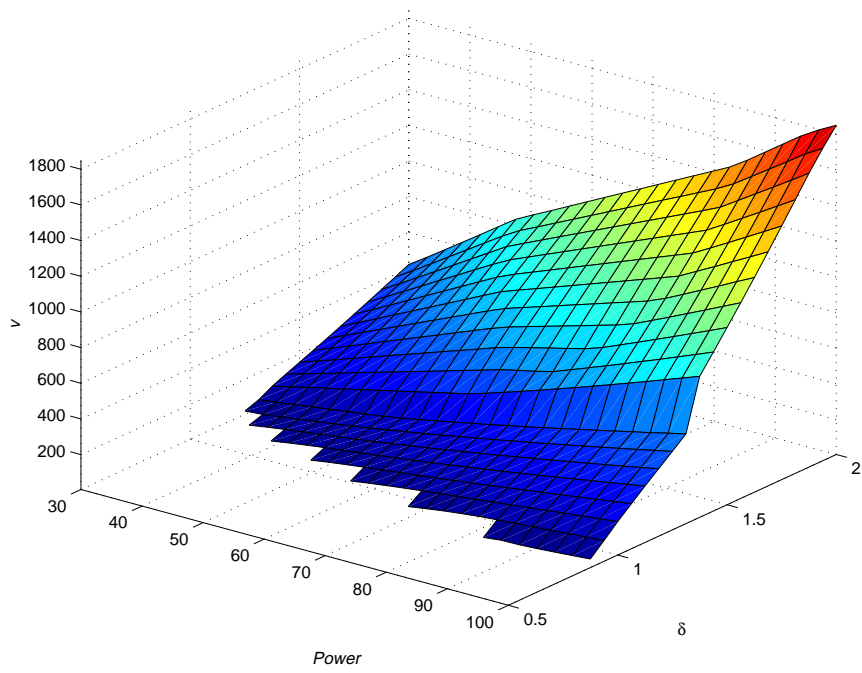


Fig. 5.26: Maximum velocity attainable at fixed power and resolution.

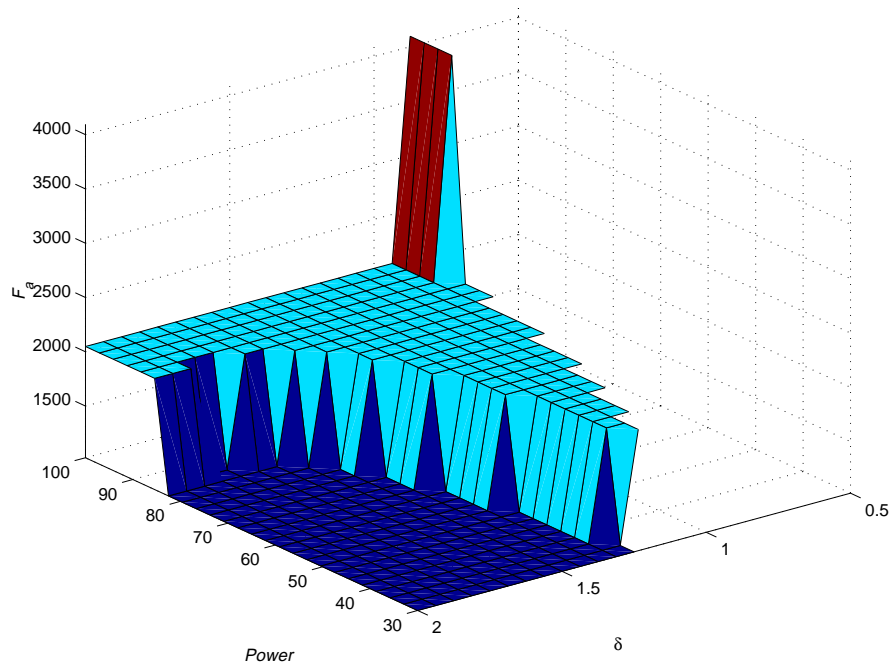


Fig. 5.27: FFT size of maximum velocity solutions.

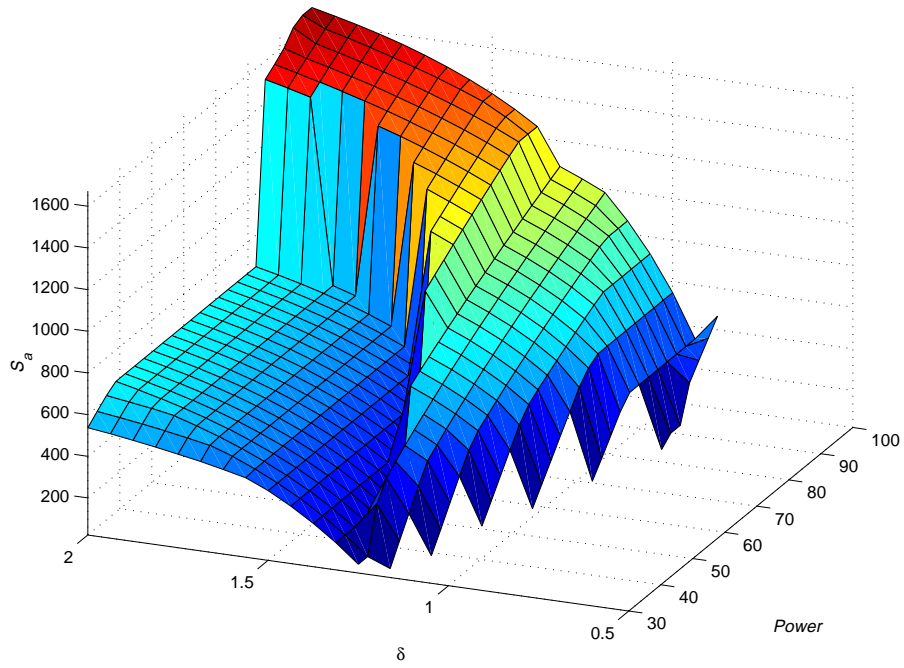


Fig. 5.28: Optimal section size for maximum velocity.

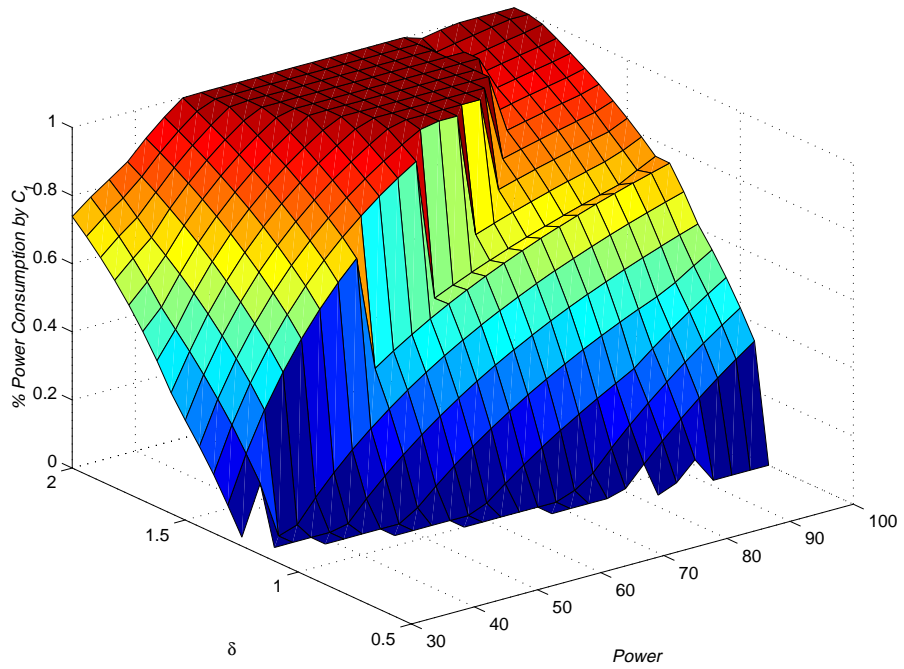


Fig. 5.29: Percentage power consumption by S2T16B in optimal mixed configuration.

this range of values, the S2T16B consumed an average 65.7% of the power.

5.2.1.2 Optimal Single Card Type Configuration

Observing that the S2T16B seems to be favored in the velocity maximization problem, the optimal single card type configurations are now investigated. Figs. 5.30 and 5.31 show the maximum velocities attainable using only the S2T16B or the S1D64B daughtercards. As expected, for feasible scenarios the S2T16B accommodates an average maximum velocity of 721 m/s compared to 286 m/s for the S1D64B. Thus, the S2T16B shows a 150% improvement over the S1D64B. However, the above statistic only considers the average across the feasible solutions for that card type. The S2T16B provides feasible solutions in only 45.9% of the power-resolution pairs, as compared to 68.0% for the S1D64B, which is the same percentage as attained by the optimal mixed configuration. This outcome results from the exclusive employment of the S1D64B in the fine resolution region by the mixed configuration. Although the S1D64B is not ideal in the majority of cases tested, it can always provide a feasible solution whenever the S2T16B can.

The FFT size employed in both single card type configurations follows the pattern expected by the respective memory-processor ratios of the daughtercards. Of the feasible solutions, the FFT size ranged from 512 to 2048 for the S2T16B and from 1024 to 4096 for the S1D64B. The graphs of the section size

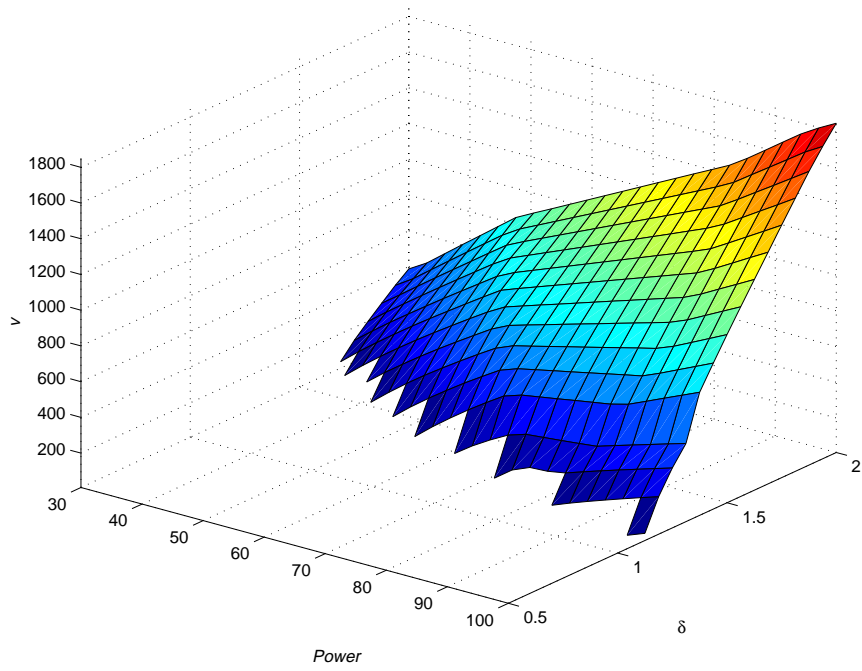


Fig. 5.30: Maximum velocity with S2T16B-only configuration.

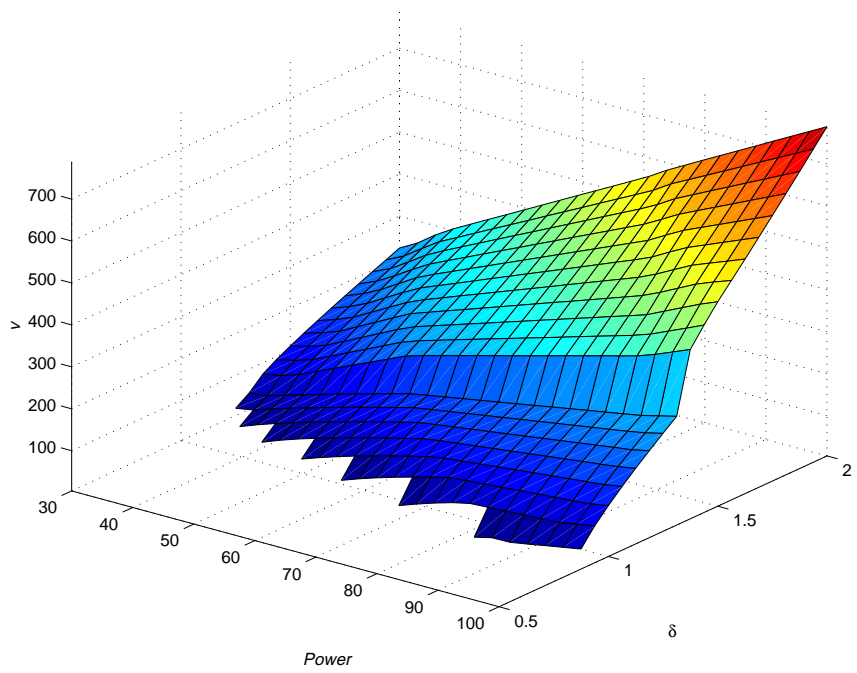


Fig. 5.31: Maximum velocity with S1D64B-only configuration.

for both daughtercards are shown in Figs. 5.32 and 5.33. Note that the graph for the S2T16B closely resembles a shifted and scaled version of the graph for the S1D64B. The shift would be in the resolution dimension by about 0.5 m and the scaling in the S_a dimension by one half. This phenomenon results from an active memory constraint in the optimization problem up to the point of feasibility for the S2T16B and an active processor constraint thereafter.

5.2.1.3 Nominal Mixed Card Type Configuration

The nominal section size with optimal card configuration problem evokes some interesting variable relationships. Fig. 5.34 graphs the maximum velocity attainable under this configuration. The points of discontinuity in the graph correspond to jumps in the FFT size, as illustrated in Fig. 5.35. The point of interest in these two graphs is that as the velocity increases, F_a decreases (note that Fig. 5.35 is reversed in regards to Fig. 5.34). It could be expected that maximizing velocity, being processor intensive, would call for large FFT sizes for efficient processing as in the optimal mixed configuration (Fig. 5.26). However, just the opposite is true in this case. The nominal section size forces the FFT size to be much smaller than is optimal because the section size is equivalent to the kernel size (Fig. 5.10), and the kernel size decreases as resolution becomes coarser. Compensation for this counterproductive section size trend is made by employing a larger percentage of the S2T16B card (Fig. 5.36). The processor

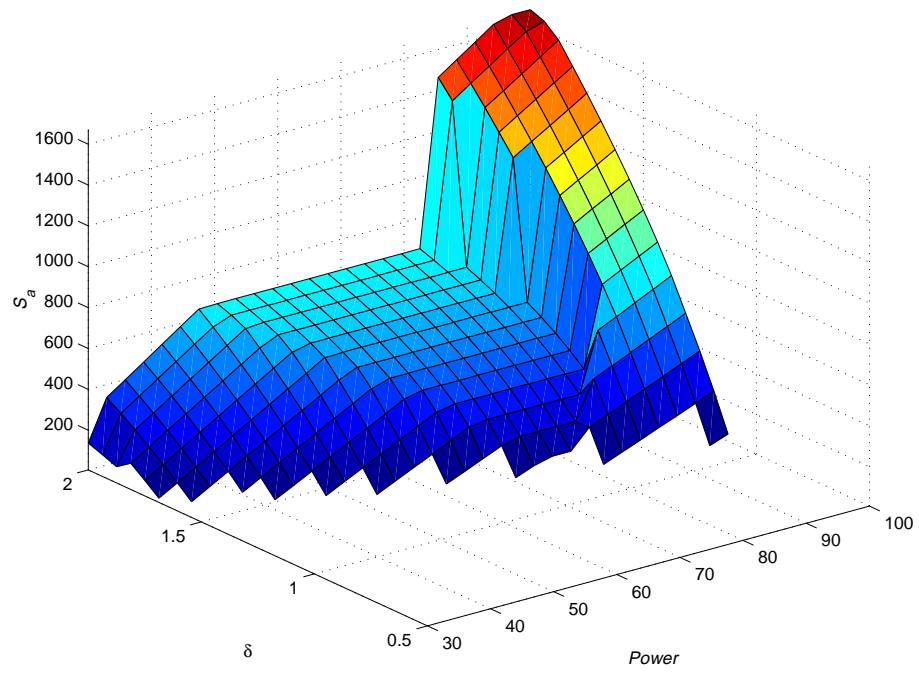


Fig. 5.32: Section size of S2T16B configuration in maximum velocity problem.

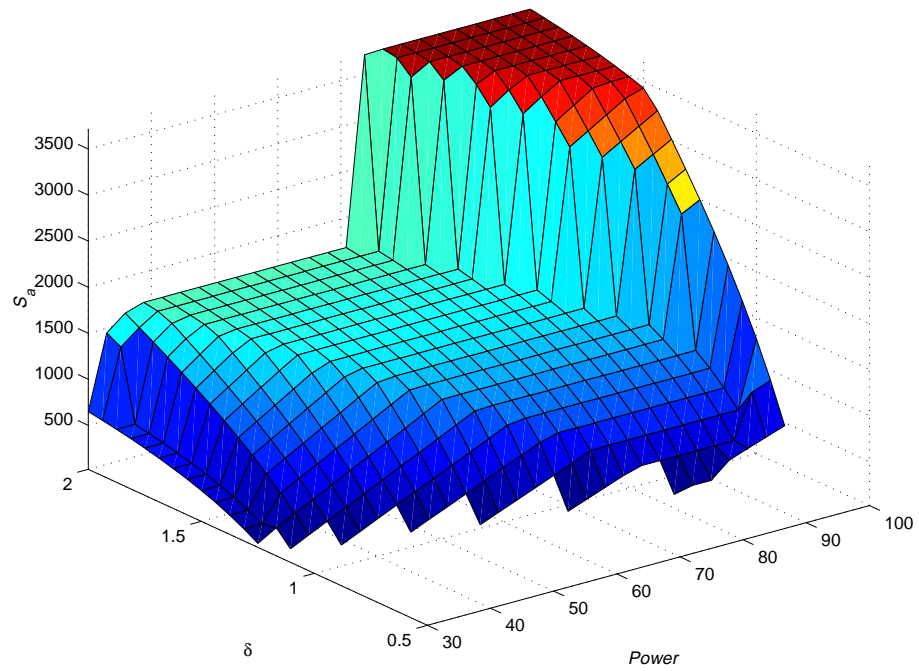


Fig. 5.33: Section size of S1D64B configuration in maximum velocity problem.

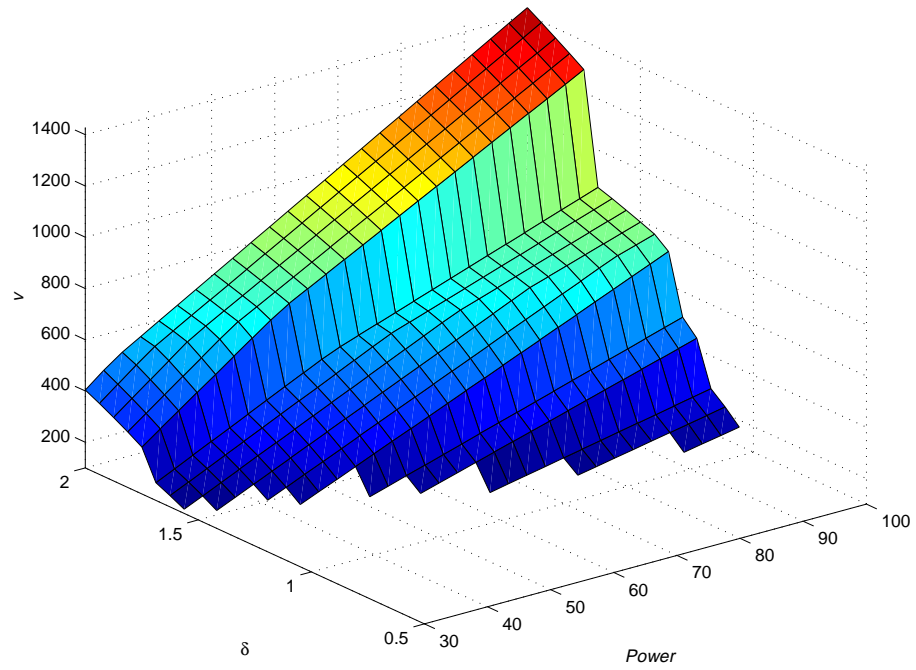


Fig. 5.34: Maximum velocity attainable in nominal mixed configuration.

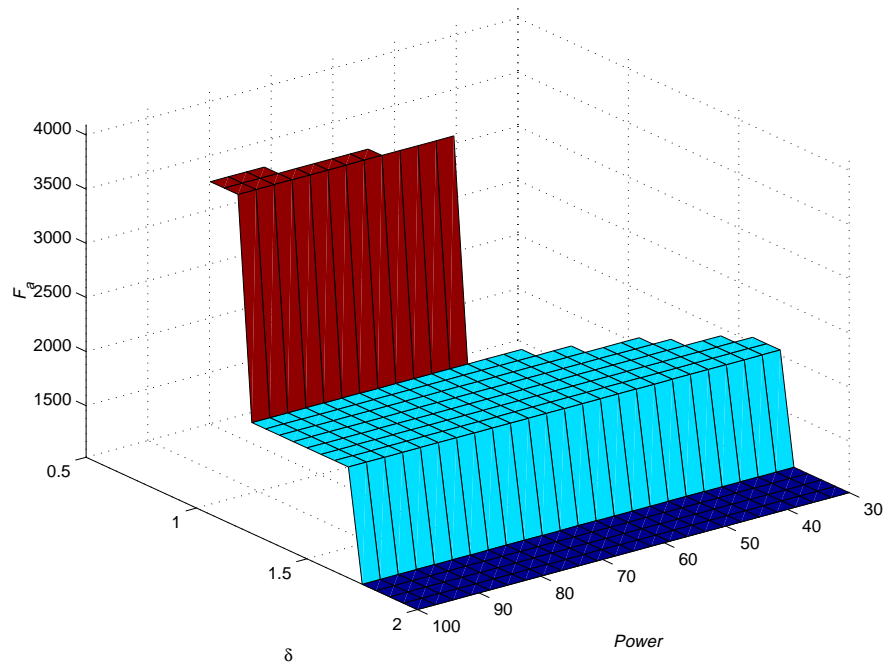


Fig. 5.35: FFT size for maximum velocity attainable in nominal mixed configuration.

rich and memory poor S2T16B card can afford to do rather inefficient processing with the small FFT size yet still provide higher velocities than could the S1D64B.

5.2.1.4 Nominal Single Card Type Configuration

Finally, the nominal single card type configurations are investigated. Figs. 5.37 and 5.38 depict the maximum velocities and feasibility regions for the nominal section size configurations of the two card types. The relationship between these two configurations is very similar to that of the optimal single card type configurations but with decreased velocities and regions of feasibility.

5.2.1.5 Comparison of Maximum Velocity Configurations

Table 5.2 compares the different configurations for the maximum velocity problem. Note that the minimum velocity statistic is not meaningful because each configuration theoretically at some point provides a maximum velocity of $0+\epsilon$, where ϵ is a very small number. However, from graph to graph the minimum velocity varies because the discrete sampling points disallow the occurrence of the real minimum velocity in each case. Although the percentage of area with feasible solutions statistic approaches 100% as the resolution and power approach infinity, the statistic is meaningful for the sampling space because these values are deemed as representative of values of a real system. Average velocity statistics

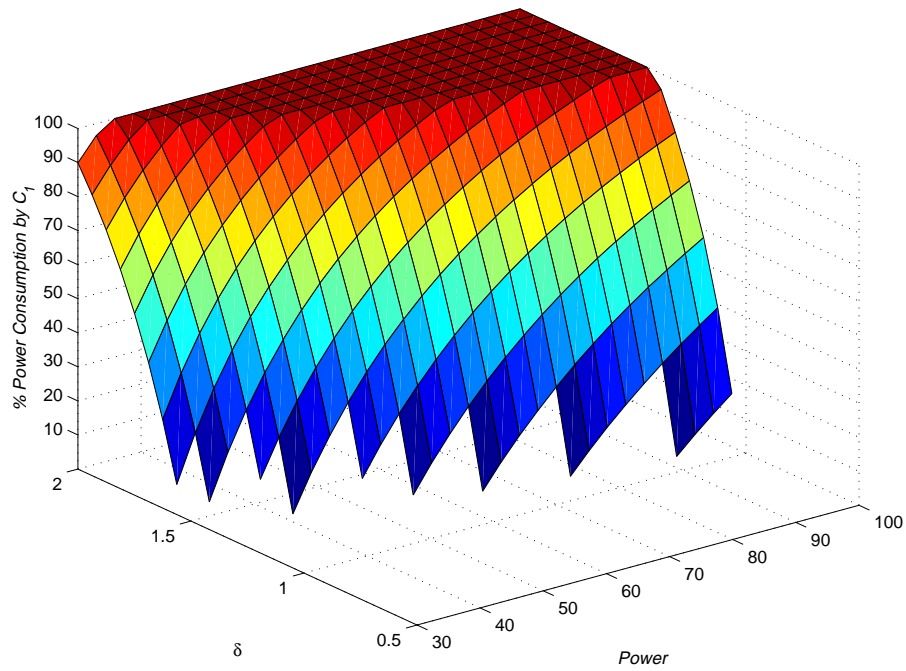


Fig. 5.36: Percent power consumption by S2T16B in nominal configuration for maximum velocity.

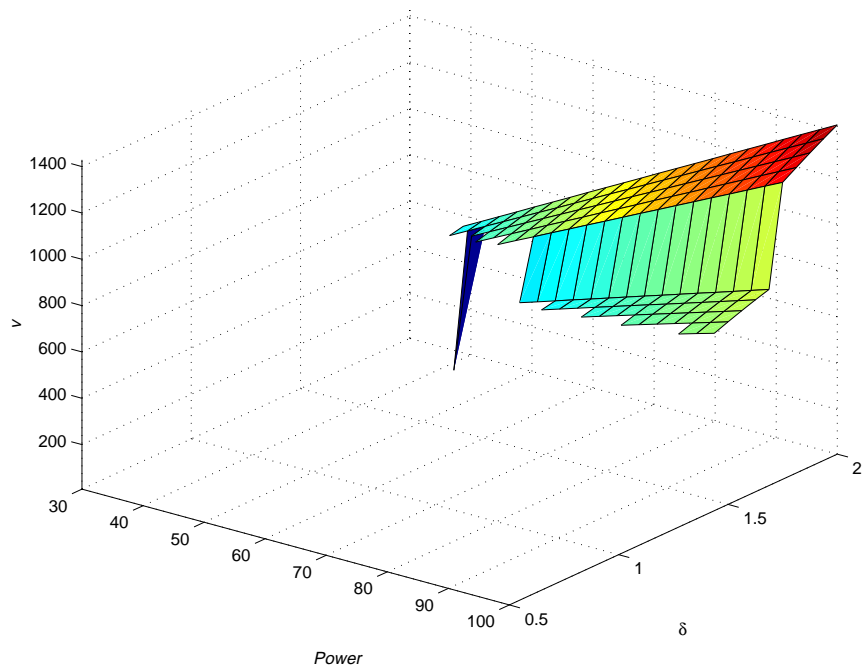


Fig. 5.37: Maximum velocity attainable in nominal S2T16B configuration. The lowest value represents a physically impractical velocity of 7.6 m/s.

are given both over the total area and over the feasible area only. The average velocity over the feasible area is not a valuable statistic alone in the design of a system, although it does provide insight into the performance of a configuration once the feasible solution boundary is crossed.

5.2.2 Configuration with Set Number of Cards

Constraining the problem further, the number of each card type is also fixed. Although this model is much simpler to optimize because there are two fewer optimization variables (C_1 and C_2), this model may represent a frequently occurring situation for a system engineer: The hardware is already decided, whether because it was the only option in purchasing or because it is being reused from a previous purpose, and now the software must be configured to make the system work at optimal performance. The power then is set ($\Pi = 12.2C_1 + 9.6C_2$) and the only variables left to optimize are v and S_a . The objective function and constraints remain the same as in the set power problem except for the omission of the power constraint (Eqn. 5.10).

Fig. 5.39 compares the optimal and nominal configurations of two different systems. The first system has five each of the two daughtercard types. The second type has seven of the S2T16B and two of the S1D64B. Note that the power consumption of both systems is slightly different: The 5:5 system requires 109.0 w and the 7:2 requires 104.6 w. The results were similar to those above

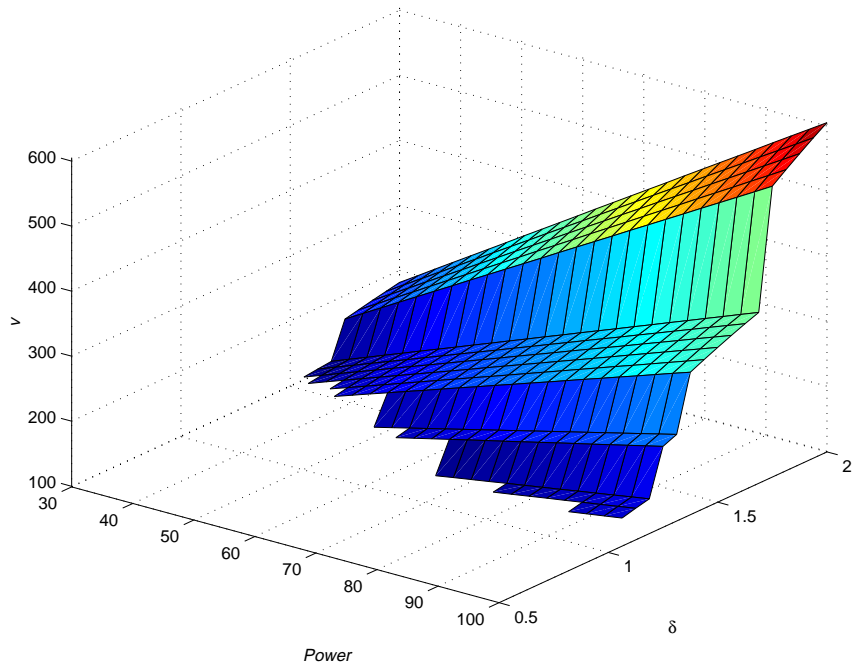


Fig. 5.38: Maximum velocity attainable in nominal S1D64B configuration.

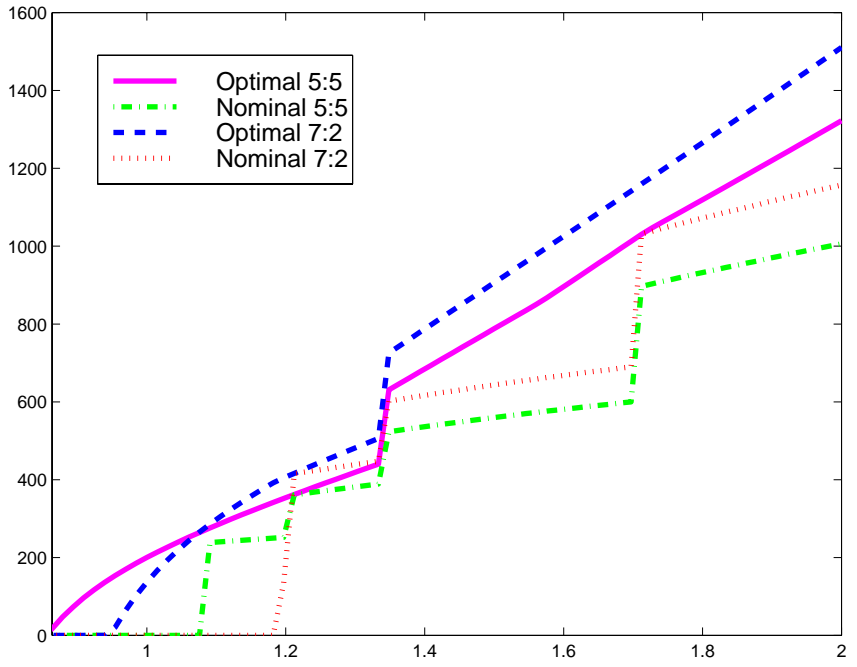


Fig. 5.39: Maximum velocity by nominal and optimal configurations in two systems: one having five of both type daughtercards and the other having seven S2T16Bs and two S1D64Bs.

of the fixed power but variable card-configuration model. As expected, the configuration with the greater proportion of S2T16B cards performed better at coarse resolution and provided fewer feasible solutions at fine resolutions.

A revealing point in the plot is where resolution is approximately 1.35 where there is a sharp point of discontinuity. Unlike the power minimization problem, the discontinuities do not result from jumps in the FFT size. Inspection of Figs. 5.40 and 5.41 shows no corresponding FFT size movement at $\delta = 1.35$ m. In the 7:2 configuration, F_a even remains at a constant 2048. Instead, the reason for the discontinuities in the maximum velocities for the optimal configurations is due to a jump in the range FFT size. The range FFT size has played an insignificant role in the optimization problem up to this point in the investigation. With the number of cards and resolution set, the fall of F_r from 32768 to 16384, caused by the increase in resolution coarseness, spurred a sharp increase in maximum velocity because an additional seven processors became available for azimuth processing. Recall that F_r is computed as the next power of two greater than the sum of S_r and K_r , both of which are functions of resolution and radar parameters, and are therefore not optimized in the maximum velocity problem. As a result, the effect of S_r is much more poignant in the present problem than in other problems.

Also note that a major disadvantage with the nominal section size heuristic in the maximum velocity problem is that as the section size optimally needs to

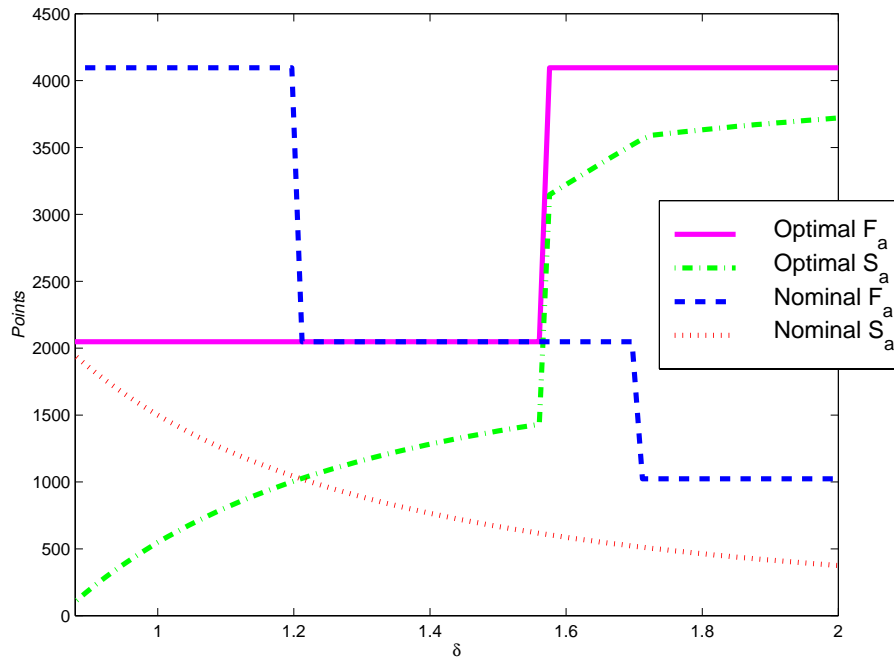


Fig. 5.40: Optimal and nominal FFT and section sizes for the 5:5 system.

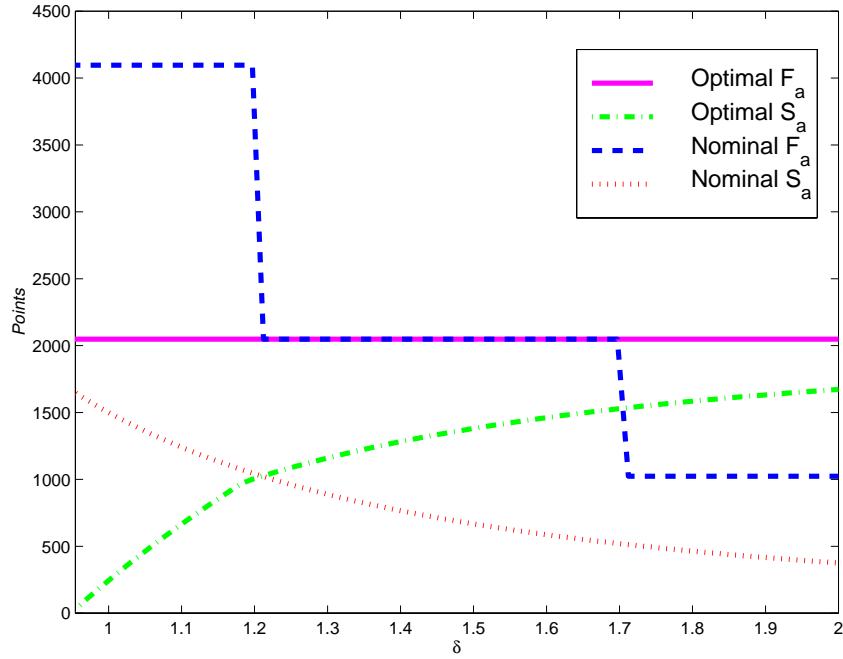


Fig. 5.41: Optimal and nominal FFT and section sizes for the 7:2 system.

be increasing as resolution becomes coarser, the decreasing K_a forces S_a to also decrease. As a result, the disparity between the optimal and nominal configurations increases as curves approach the right side of the plot where resolution becomes coarser.

Table 5.2.2 summarizes the two set hardware configurations discussed above. Dissimilar to the power minimization problem, where the optimal section size was usually much smaller than the kernel size, the optimal section size in the present case averages two to three times the nominal section size.

5.3 Minimization of Resolution

The minimization of resolution (i.e., making resolution finer) is the most computationally intensive of the optimization problems. Resolution must be known before any of the following expressions can be calculated: $K_r, S_r, P_r, M_r, K_a, P_a$, and M_a . The most troublesome of the above variables for formulation is K_a . Without a value for K_a when the optimization algorithm is entered, not even a lower bound for F_a can be determined. Recalling that $F_a = 2^k$, where $k = \lceil \lg(K_a + S_a) \rceil$, and because S_a is to be optimized, the first value of k usually tried is $k = \lceil \lg(K_a + 1) \rceil$. Without a value for either S_a or K_a , however, the above calculation for k becomes $k = \lceil \lg(1 + 1) \rceil = 1$.

Based on historical data, a slightly larger value for k can be initially injected into the optimization routine and successively higher values tried thereafter in

Table 5.2: Comparison of configurations showing the average velocity over the total sampling area ($\overline{v_t}$), maximum velocity, the average velocity over only the feasible solutions ($\overline{v_f}$), and percentage of area with feasible solutions and the percent increase or decrease of each statistic over that of the optimal mixed configuration.

Configuration	$\overline{v_t}$	% +	Max.	% -	$\overline{v_f}$	% +	% Feas.	% -
Optimal Mixed	382	-	1851	-	562	-	68.0	-
Nominal Mixed	309	19.0	1429	22.8	592	5.3	52.3	23.1
Optimal S2T16B	331	13.5	1851	0.0	592	28.2	45.9	32.5
Optimal S1D64B	194	49.2	789	57.5	286	-49.1	68.0	0.0
Nominal S2T16B	211	44.8	1429	22.8	882	56.8	24.0	64.7
Nominal S1D64B	141	63.1	605	67.3	271	-51.9	52.3	23.1

Table 5.3: Comparison of set hardware configurations: (1) five each of both cards and (2) seven S2T16Bs and two S1D64Bs. The table shows the minimum resolution at which a solution was feasible, the maximum velocity, the average velocities over the total range of resolutions ($\overline{v_t}$) and over only feasible resolutions ($\overline{v_f}$), and the average section size $\overline{S_a}$.

Configuration	Min δ	Max v	$\overline{v_t}$	$\overline{v_f}$	$\overline{S_a}$
Optimal 5:5	0.95	1511	516	679	1932
Nominal 5:5	1.20	1157	379	622	690
Optimal 7:2	0.86	1323	574	821	1215
Nominal 7:2	1.09	1006	415	768	629

the same manner as in the other problems. A better initial value could be offered for F_a if the lowest feasible value for resolution was calculated beforehand for which all the constraints are met, but that in itself is the optimization problem.

As a result of the above difficulty with a lack of an initial K_a , the first guesses at F_a tend to be very poor. MATLAB's **constr** function is not always robust enough to handle such poor guesses and in the course of calculating the best resolution for the range of F_a values tried, **constr** periodically “crashes” on infeasibly low F_a values, seemingly having entered into an infinite loop. To rectify such a situation, the program must be restarted at the point it failed, incrementing the k in F_a by one (only for that power-velocity pair).

Only several of the possible configurations for the resolution minimization problem are investigated here because of the computational intensity and strains on the robustness of the optimization routine for this problem. Furthermore, in some cases solution points are obviously aberrant from their surrounding values. Consequently, the absolute convexity of the solution space for resolution minimization is suspect. In each configuration investigated below, the initial solution surface is shown as for all the power minimization and velocity maximization problems. However, because of the aberrant solution points mentioned, where appropriate the aberrant points in the initial surfaces are smoothed using a moving average technique. This surface then is also presented.

The objective function for the resolution minimization problem, similar to

the velocity maximization problem, is to minimize

$$Z = \delta. \tag{5.14}$$

The constraints are revised to reflect the dependence on δ :

$$6C_1 + 2C_2 \geq P(F_a, S_a, \delta) \tag{5.15}$$

$$32C_1 + 64C_2 \geq M(S_a, \delta) \tag{5.16}$$

$$K_a(\delta) + S_a \leq F_a, \tag{5.17}$$

with the standard lower bounds:

$$C_1 \geq 0, C_2 \geq 0, S_a \geq 1, \delta > 0. \tag{5.18}$$

5.3.1 Optimal Mixed Card Type Configuration

The initial optimal solution graph for the resolution minimization problem is shown in Fig. 5.42. It would be expected that an optimal surface would be nonincreasing or nondecreasing along each dimension. That is, as velocity increases in one dimension for a set power, resolution should become coarser. Similarly, as power increases for a set velocity, resolution should become finer. Thus it is expected that the optimal solution surface is nondecreasing in the

power dimension and nonincreasing in the velocity dimension.

However, it is observed that there are aberrations from this expected characterization in Fig. 5.42. Checking the surface against the characterization described above, a total of twelve deviant points are found, although a cursory visual inspection of the graph reveals four prominent aberrations. For each of these nonoptimal solutions, it is found that the optimization routine employed a smaller FFT size than in the surrounding points. In some cases, forcing the optimization routine to solve for a higher FFT size results in the optimal solution. In other cases, the optimization routine cannot find the optimal solution without a very precise initial guess and an adjusted step size for the MATLAB function. It is also observed that for some deviant points, the surrounding area, although smooth, does not employ a constant FFT size. Rather, the FFT size oscillates between two values. This phenomenon could suggest a boundary area or even a nonconvex area in the solution space resulting in nonoptimal solutions. Supporting the possibility of nonconvexity, the optimization routine occasionally returns an “infeasible solution” message with some initial guesses.

Due to the time expenditure and unreliability of reoptimizing a particular point in the solution surface, as discussed above, a 3 x 3 neighborhood averaging mask was applied to apparently suboptimal solution points. The resultant surface of this smoothing technique is shown in Fig. 5.43. Note that the smoothing mask is not applied to the entire surface but only to the apparent points of de-

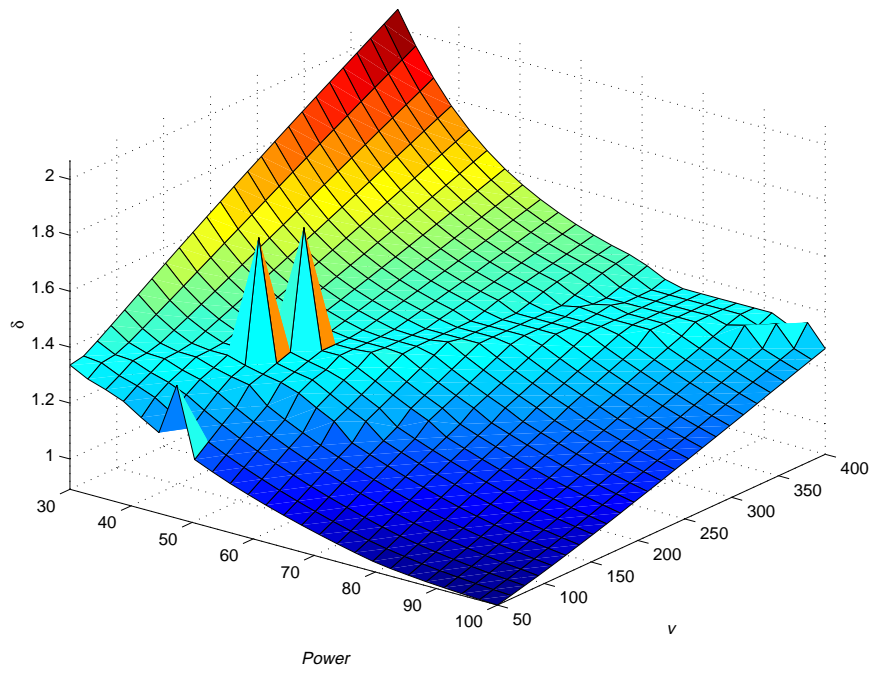


Fig. 5.42: Initial minimum resolution solution in optimal mixed card type configuration.

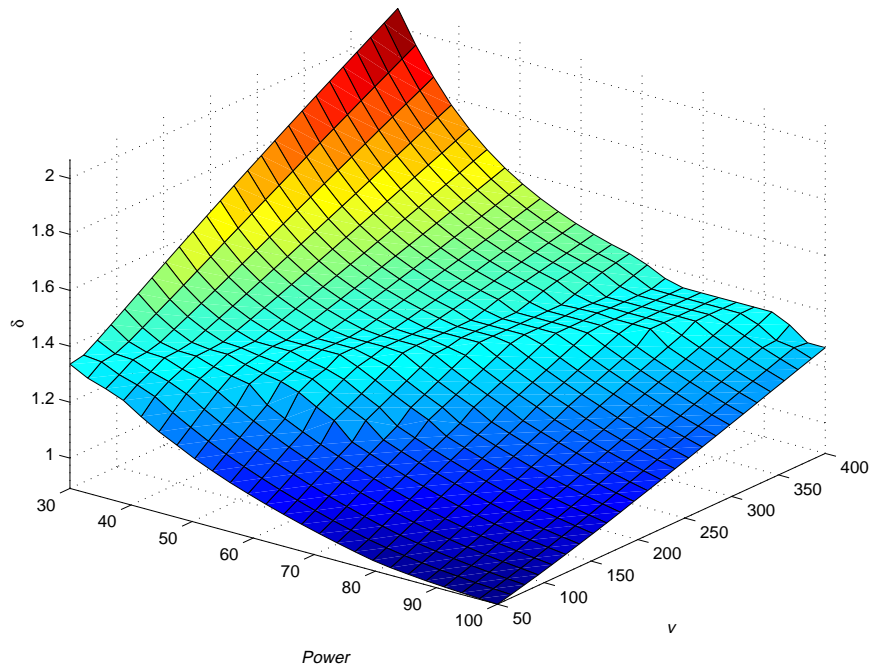


Fig. 5.43: Smoothed minimum resolution solution in optimal mixed card type configuration.

viation. Such interpolated points should provide a basis from which to calculate an optimal value in the case that the particular power-velocity coordinates are exactly the values that are required on a particular system. Confidence in the overall optimality, previously aberrant points notwithstanding, of the solution graph of Fig. 5.43 is lent both from the characteristics of the surface itself and from informal verification of values by cross checking them against the power minimization and velocity maximization graphs of Figs. 5.4 and 5.26, respectively.

Figs. 5.44, 5.45, and 5.46 illustrate the surfaces formed by the azimuth FFT size, section size, and kernel size, respectively, for optimal resolution. That is, the graphs below are based on interpolated values from the smoothed graph of Fig. 5.43. Note that in Fig. 5.44 the surface is consistent with FFT size graphs from previous problems, except for the rift toward the center of the graph. This rift is also reflected in the section size graph of Fig. 5.45, and corresponds to a portion of the level area spanning the graph of minimum resolution in Fig. 5.43. This phenomenon could result from suboptimal solutions in the lower portions of the surface, but it must be kept in mind that there are no infeasible solutions plotted in the graph. Therefore, according to the principle of necessary nonincreasing or nondecreasing functions along each dimension for optimal resolution values, as discussed above, the upper values of the surface can only be in question in that they are too high, not too low. It is assumed at this point that

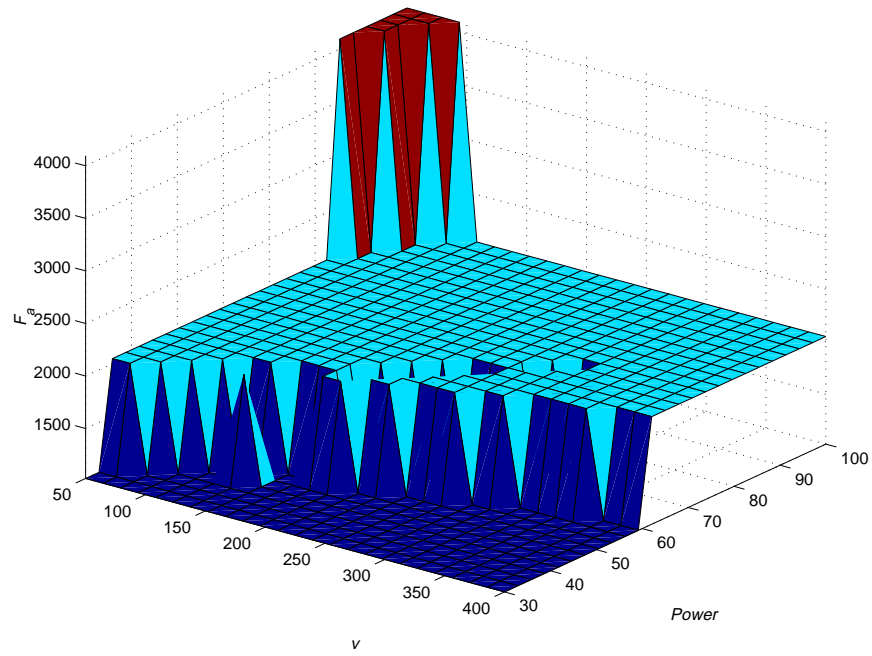


Fig. 5.44: Optimal azimuth FFT size for minimum resolution.

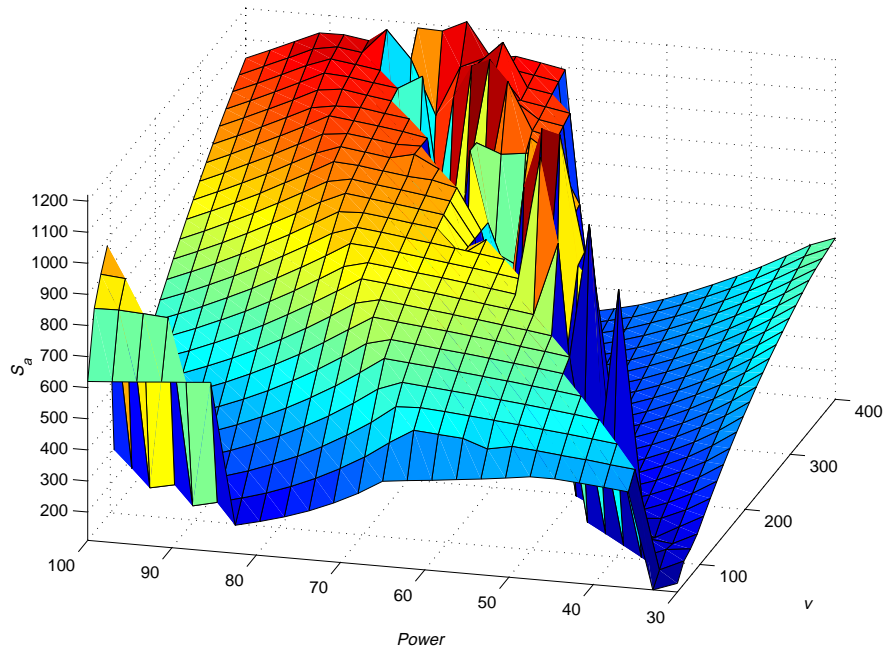


Fig. 5.45: Optimal azimuth section size for minimum resolution.

Fig. 5.43 represents a very close approximation to the optimal solution surface. Methods to scrutinize this assumption will be investigated in Chapter VII.

5.3.2 Optimal Single Card Type Configuration

Optimization of the single card type configuration for resolution minimization encountered problems. When only the S2T16B was allowed in the configuration, the initial solution surface displays several aberrant points as with the mixed card type graph in Fig. 5.43. See Figs. 5.47 and 5.48 for the initial and smoothed graph for the S2T16B-only configuration.

It would be expected that the single card type configuration using only the S1D64B would display a similar optimization graph with just several anomalies. However, Figs. 5.49 and 5.50, different views of the same graph, do not display isolated points of deviation, but deviant trends. As a result, the smoothing technique is not employed on this graph because not every aberrant point is surrounded by reasonable solution points from which to interpolate a better value. More research into the **constr** function implemented in MATLAB and the solution space of the problem is necessary to surmise why the algorithm performed so poorly for this configuration.

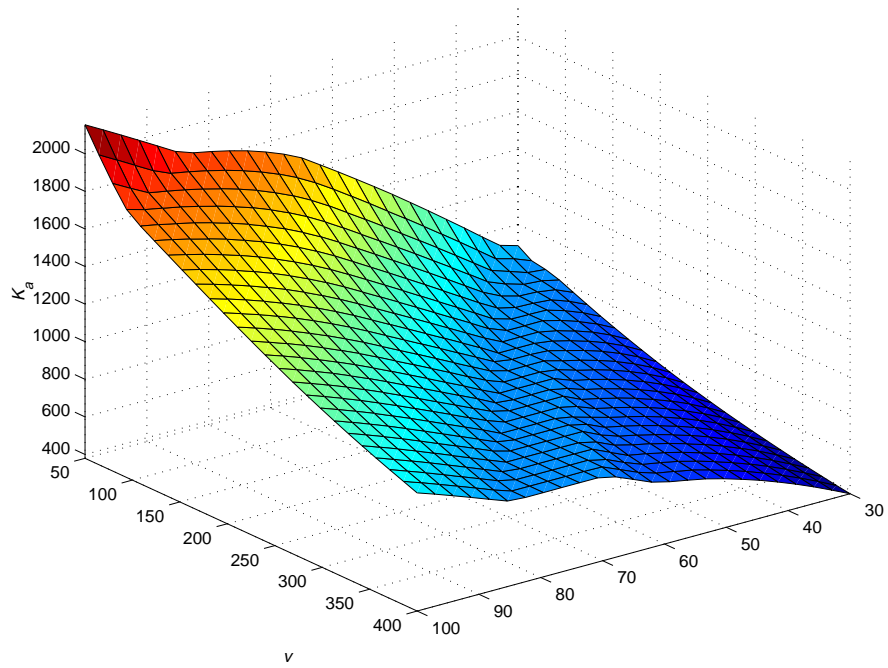


Fig. 5.46: Azimuth kernel size for minimum resolution.

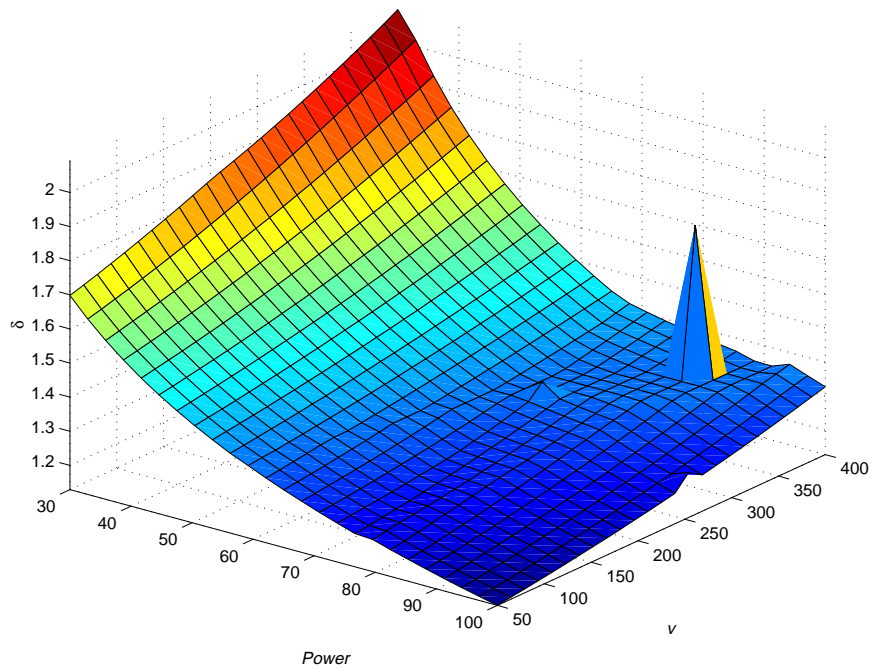


Fig. 5.47: Initial solution graph of the S2T16B-only configuration for resolution minimization.

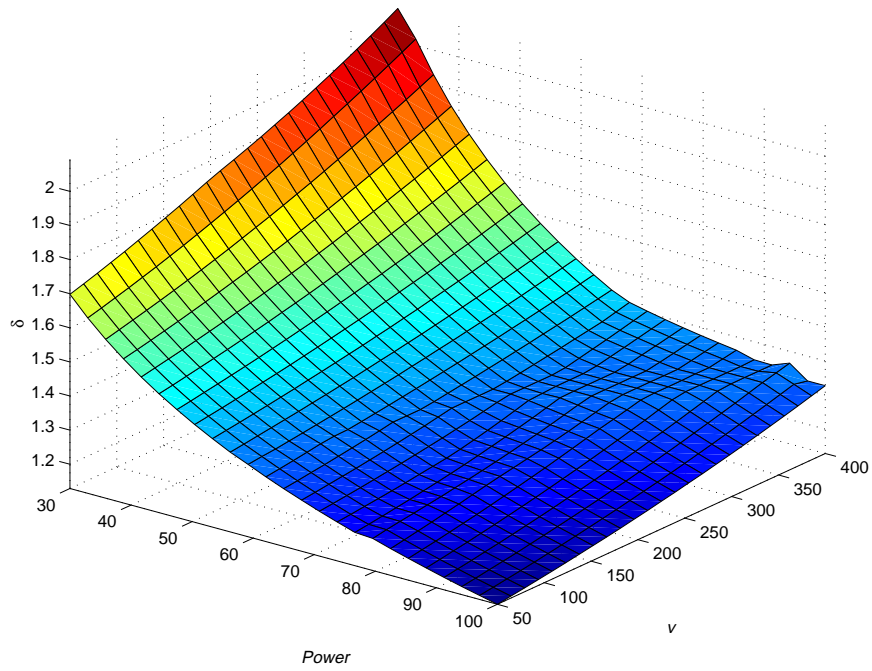


Fig. 5.48: Smoothed solution graph of the S2T16B-only configuration for resolution minimization.

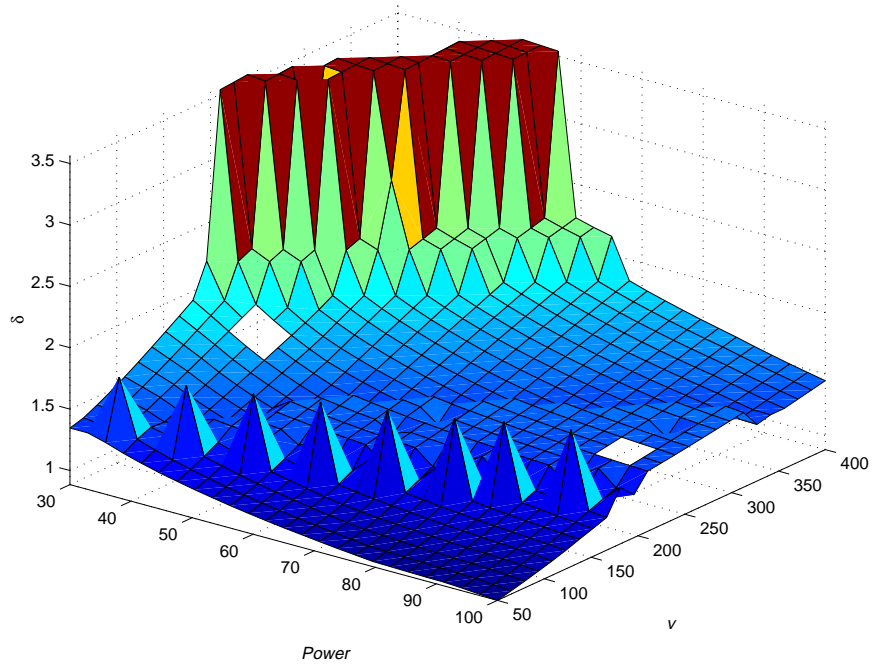


Fig. 5.49: Initial solution graph of the S1D64B-only configuration for resolution minimization.

5.4 Conclusions

Three distinct optimization objectives have been investigated in this chapter: power minimization, velocity maximization, and resolution minimization. Of the three objectives, most attention has been directed toward power minimization because power is representative of the restrictions concerned in SWAP-constrained systems, as introduced at the beginning of this work. Velocity and resolution optimizations were also investigated, with limited success in the minimization of resolution because of the computational complexity and possible lack of solution space convexity.

For each objective mentioned above, different configurations are explored. Configurations in which the section size is optimized are denoted as optimal, where configurations in which the section size is fixed as the kernel size are denoted as nominal. Both mixed and single card type configurations are investigated. In the mixed configurations, the number of each type of the two available card types are optimized, except for one scenario in the velocity maximization problem where the number of each card type is set.

One of the motivating factors in the outset of this research was to investigate the significance of the arbitrarily-set azimuth section size. It has been shown that proper selection of the section size is crucial to the performance of a system. Without optimization of this parameter, processors or memory can be wasted in a system. The optimal value of the section size is often unintuitively low,

conserving memory but causing relatively inefficient use of processors.

The ISMM provides a starting point for system design and performance evaluation. Although some significant assumptions are made in this model to simplify the optimization formulation and concomitant computation, it will be shown that this simplification provides a reasonable lower-bound for the more involved and accurate model presented in Chapter VI. Furthermore, the simplicity of the ISMM and the associated freedom granted the parameters in each scenario accentuate the interrelationships between the variables, the characteristics of which are otherwise more difficult to discern in the more realistic model. This simplification results in significantly reduced computational intensity and allows for the production of all the data presented in this chapter.

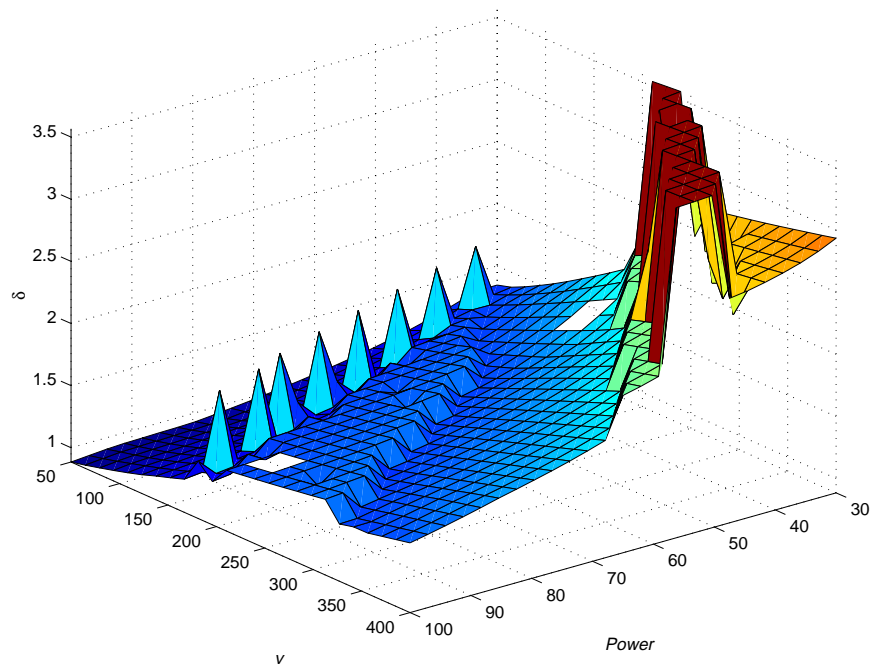


Fig. 5.50: Alternate view of the initial graph of the S1D64B-only configuration.

CHAPTER VI

CN-CONSTRAINED MODEL

Increasing the realism of the optimization model, the set of constraints is now revised to ensure that no remote memory accesses occur besides the matrix transposition operation from the range to the azimuth processors. This model is significantly more complex and necessitates the introduction of several new variables. Besides the additional constraints restricting the amount of available memory per processor, the primary difference between this model (henceforth denoted as the CNCM) and the ISMM is in the concept of the fundamental unit of system construction. The fundamental building block shifts from an ambiguously configured daughtercard to a precisely configured CN.

6.1 Formulation

The variables C_1 and C_2 , designating the first and second card types, or number of S2T16Bs and S1D64Bs employed, no longer have meaning in the present model without further refinement. Two new sets of variables, discussed in depth in the next section, are introduced to replace C_1 and C_2 , implementing these refinements. Instead of simple card type variables, the new model requires CN configuration variables. The distinction is made in that the card type is only one parameter in the configuration of a card. In addition, the configuration must

specify the number of processors dedicated to range processing and the number of processors dedicated to azimuth processing. Similarly, the configuration description must also delimit the amount of memory dedicated to range and azimuth processors on a given CN. This last detail guarantees the absence of remote memory access during range and azimuth processing. Data must still be transferred after range processing is complete from range to azimuth processors (the distributed matrix transposition).

With only two processor usages (azimuth and range processing), optimization always will require at most two different card configurations. In this chapter, a card configuration defines the number of processors on each CN type used for range and azimuth processing, and the amount of memory allocated to both types of processing per CN type. Recall that a CN consists of multiple processors sharing a common memory.

Three possible optimization scenarios are possible. The first and most simple scenario occurs when the optimization routine determines that the optimal configuration involves dividing the processors and memory on a card type such that both range and azimuth processing is executed. Furthermore, assuming that whatever division of resources is determined to be optimal, the ratio of range and azimuth processors in the given configuration is equal to the ratio of total range and azimuth processors in the system. In this case, N such configured CNs are required, providing all the required processors, and thus only one card

configuration is demanded. If this mixed CN configuration is optimal (a mixed CN configuration is one in which non-zero fractions of the resources on the CN are allocated for both range and azimuth processing), then no other configuration is necessary. That is, the addition of a second configuration will not improve the performance of the system in any way. (The only time this rule does not hold true is in the optimization of the final CN of a type, which is probably fractional according to the requirements. At this point, however, fractional CNs are permitted in the solution and further discussion of this situation is deferred until later in this chapter.)

To illustrate the first scenario, suppose ten range processors and twenty azimuth processors are optimally required. A possible configuration of the above type might be implemented with S2T16B cards, with one range and two azimuth processors assigned per CN. This configuration assumes that the sixteen megabytes of memory on the single CN is sufficient for all three processors. That is, twice the azimuth memory requirement plus the range requirement per processor must be less than or equal to sixteen megabytes. Note that the azimuth and range memory requirements per processor need not be, and most probably will not be, the same.

Consider the next optimization scenario in which the optimization algorithm determines that the best use of CNs is to dedicate all of one type of CN configuration to range processing and another to azimuth processing. In this case,

two configurations are necessary for optimality. As an example, one type of CN could be on the S2T16B and all three processors could be dedicated to range processing. Each processor would have for its own exclusive use $\frac{16}{3} = 5.33$ MB. The azimuth processing could be assigned to CNs of the S1D64B card. Both processors could be utilized, yielding $\frac{64}{2} = 32$ MB per processor.

The above example coincidentally preserves the convention of the S2T16B as the card type of the first CN configuration and the S1D64B as the card type of the second CN configuration. However, it is important to note that with the new notation, the card types associated with the two CN configurations do not necessarily correspond to the S2T16B and the S1D64B, respectively. The optimization algorithm is given freedom to determine the optimal configuration(s), and the result could be a reversal of the previously designated card types. Although this ambiguity alone could be easily forced into conformity with the earlier definition of *type*, it is important to maintain the ambiguity to allow for the possibility of only one optimal CN configuration, as in the first example, or even to allow for two different CN configurations using the same daughtercard type. The fact that there are two card types and two possible optimal CN configurations is purely coincidental. The latter is due to the presence of two possible programs, or two types of processing (range and azimuth). Even for any number of available daughtercard types, an optimal configuration still would only require at most two CN configurations.

The third possible optimization scenario resembles the first example in the mixed CN configuration, but with the exclusion of the condition that the ratio of range and azimuth processors on the CN is equivalent to the ratio of the total required range and azimuth processors. Such a situation necessarily occurs when there exists a great disparity in the required number of range and azimuth processors. In such a case, one CN configuration would be a heterogeneous assignment of range and azimuth processors to a single CN, and the second CN configuration would be a homogeneous assignment of whichever processor type was still lacking. For example, if five range processors and twenty azimuth processors were required, the CN configuration of the first example could be employed to incorporate all the range processors and ten of the azimuth processors. The remaining ten azimuth processors would be assigned in a homogeneous CN configuration, either on the same or different type card.

In each case, it is possible that there will be a portion of memory wasted on each CN. In the same way, it is possible that an entire processor is wasted on a CN. If the memory requirements hinder the utilization of all processors, then a processor must be left idle. However, in most cases the optimization algorithm decides against using such a configuration because there is usually a more efficient way of configuring the system, usually by decreasing the section size so that less memory is required and all processors are utilized. In the last example, to accommodate the remaining ten azimuth processors on the same card

type, it is probable that only two of the three processors per CN could be utilized because azimuth processors usually require more memory than range processors.

To note the distinction between the ISMM card type variables and the new CN configuration variables, let X and Y abstractly represent the two CN configurations (note that X and Y will not be used in the formulation without accompanying subscripts defining specific characteristics of each configuration). Let X_T and Y_T represent the daughtercard types of the new configuration variables of the CNCM, where the type can be either the S2T16B or the S1D64B daughtercard. Let N_X and N_Y denote the number of each CN required of the corresponding configuration. Note that the combination of the two sets of variables defined above essentially serves the same function as did C_1 and C_2 in the ISMM, with C_1 and C_2 representing the number of cards required themselves and their type implicit in their definition. In contrast, the new variables explicitly define each quantity and quality associated with them.

Two additional subscripts are necessary for the CN configuration variables to complete their description. For notational convenience, let $I \in \{X, Y\}$. To denote the number of processors dedicated to range and azimuth processing on a specifically configured CN, I_r and I_a are introduced, where the r and a refer to *range* and *azimuth*. It might seem necessary also to create a variable to define the amount of memory allocated to each processor of each type, but as shown below, this constraint can be implicitly figured by the ratio of the total amount

of memory needed per processor function (i.e., for range or azimuth processing) to the total number of processors (per function) required. Memory thus will be treated as an implicit rather than an explicit optimization variable.

The first two constraints in the formulation ensure that a sufficient number of range and azimuth processors are allocated:

$$P_r \leq N_X X_r + N_Y Y_r$$

$$P_a(S_a) \leq N_X X_a + N_Y Y_a.$$

In contrast to the ISMM, where only one constraint concerned the total number of processors required, it is necessary to separately calculate and constrain the range and azimuth processor requirements in this model. The above two constraints define the available range or azimuth processors by taking the product of the number of CNs of each type and the number of processors on that CN dedicated to the given type of processing.

The next two constraints in the formulation are the memory counterpart of the first two processor constraints. However, as mentioned earlier, the memory per processor is not an explicit optimization variable as is the processors per CN. Instead, the memory per processor is computed implicitly by the following

ratios:

$$M_{\text{CN}}(X_T) \geq X_r \frac{M_r}{P_r} + X_a \frac{M_a(S_a)}{P_a(S_a)} \quad (6.1)$$

$$M_{\text{CN}}(Y_T) \geq Y_r \frac{M_r}{P_r} + Y_a \frac{M_a(S_a)}{P_a(S_a)}. \quad (6.2)$$

Similar to the formulation in Subsection 5.1.2, M_{CN} represents the memory available per CN as a function of the configuration type. In the present case, this function is defined as follows:

$$M_{\text{CN}}(I_T) = \begin{cases} 16 & \text{if } I_T = \text{S2T16B}, \\ 64 & \text{if } I_T = \text{S1D64B}. \end{cases}$$

An additional basic constraint is necessary to ensure that the number of processors assigned to a CN is physically realizable by that CN. The following constraint ensues:

$$X_r + X_a \leq P_{\text{CN}}(X_T)$$

$$Y_r + Y_a \leq P_{\text{CN}}(Y_T),$$

where P_{CN} designates the number of processors available per CN as a function of the configuration type. Again, as in Subsection 5.1.2, such a function allows the addition of any number of daughtercard types to the hardware choice list

without changing the optimization formulation. Only the function definitions would need to be modified to incorporate the addition of daughtercard types. In the present work, the function P_{CN} is limited to the following definition:

$$P_{\text{CN}}(I_T) = \begin{cases} 3 & \text{if } I_T = \text{S2T16B}, \\ 2 & \text{if } I_T = \text{S1D64B}. \end{cases}$$

The only additional constraint involves the FFT size and is the same as in the ISMM:

$$F_a = 2^k \geq S_a + K_a, \quad k = 1, 2, \dots$$

The standard lower bounds must also be included:

$$N_I \geq 0, I_r \geq 1, I_a \geq 1, S_a \geq 1.$$

6.2 Computational Approach

The CNCM introduces additional variables that must assume only discrete values. Unlike the section size S_a , which can be computed by merely rounding its optimized value, variables I_r and I_a , respectful of I_T , must be handled in the same manner as the FFT size F_a . Consequently, many feasible combinations of processor assignments must be tried in order to ensure optimality.

The upper bound on the number of configuration combinations that must be evaluated can be calculated by examining the three optimization scenarios discussed above. In the first scenario, involving only one type of CN heterogeneously configured with both azimuth and range processors, all combinations on each daughtercard type in which the sum of the range and azimuth processors is less than or equal to the number of processors available on a given CN must be evaluated. Let it be assumed that the first configuration type variable is optimized for this heterogeneous processor assignment on a single CN configuration (i.e., $N_X \neq 0$ and $N_Y = 0$, which could be reversed in an actual solution). Let $\pi_T = P_{\text{CN}}(X_T)$, for $T \in \{1, 2, \dots, N_d\}$, where N_d is the total number of different daughtercard types available, and all daughtercard types are represented by arbitrary consecutive numbers beginning with one. Let E_{het} denote the set of different combinations that must be evaluated in the single CN heterogeneous scenario. The enumerated triples in the following equation, i.e., daughtercard type (as a number), X_r , and X_a , completely specify the set of feasible combinations in the single CN heterogeneous scenario:

$$E_{\text{het}} = \bigcup_{T=1}^{N_d} \bigcup_{X_r=1}^{\pi_T-1} \bigcup_{X_a=1}^{\pi_T-X_r} (T, X_r, X_a). \quad (6.3)$$

To sum the total number of feasible combinations that must be tried, Eqn. 6.3

is evaluated as

$$|E_{\text{het}}| = \sum_{T=1}^{N_d} \sum_{X_r=1}^{\pi_T-1} \sum_{X_a=1}^{\pi_T-X_r} \quad (1)$$

which also can be expressed by

$$\begin{aligned} |E_{\text{het}}| &= \sum_{T=1}^{N_d} \sum_{j=1}^{\pi_T-1} \binom{j}{1} \\ &= \sum_{T=1}^{N_d} \sum_{j=1}^{\pi_T-1} j \\ &= \sum_{T=1}^{N_d} \frac{[\pi_T - 1][(\pi_T - 1) + 1]}{2} \\ &= \frac{1}{2} \sum_{T=1}^{N_d} (\pi_T^2 - \pi_T). \end{aligned} \quad (6.4)$$

To illustrate, suppose that the number of available daughtercard types is $N_d = 3$ and that the number of processors for each CN associated with each daughtercard is $\pi_1 = 2$, $\pi_2 = 4$, and $\pi_3 = 3$. Then according to Eqn. 6.4, $|E_{\text{het}}| = \frac{1}{2}[(2^2 - 2) + (4^2 - 4) + (3^2 - 3)] = 10$.

In the second scenario, involving homogeneous assignments to CNs of range and azimuth processors, two CN configurations are necessary in which either $I_r = 0$ in one case and $I_a = 0$ in the other, or vice-versa. To enumerate the feasible CN configuration combinations, let π_T as used in the heterogeneous scenario above be modified to reflect the letter of the configuration variable in addition

to the daughtercard type. That is, let $\pi_{I_T} = P_{\text{CN}}(I_T)$, for $T \in \{1, 2, \dots, N_d\}$. Furthermore, let E_{hom} represent the feasible configuration combinations in the homogeneous case. Assume, without loss of generality, that $X_a = 0$ and $Y_r = 0$, effectively designating configuration set X as the range CN and configuration set Y as the azimuth CN. The set of feasible configurations in the homogeneous case is then given by the following expression:

$$E_{\text{hom}} = \bigcup_{X_T=1}^{N_d} \bigcup_{Y_T=1}^{N_d} \bigcup_{X_r=1}^{\pi_{X_T}} \bigcup_{Y_a=1}^{\pi_{Y_T}} \{(X_T, X_r, X_a = 0), (Y_T, Y_r = 0, Y_a)\}, \quad (6.5)$$

where the pair of triples is of the same convention as set in the heterogeneous formulation.

Although Eqn. 6.5 represents a large number relative to Eqn. 6.3, only a small percentage of these combinations must be actually tried for the optimal solution because the configuration of the range CN is independent of the azimuth CN configuration in the homogeneous scenario. That is, the optimal range CN is optimal regardless of the optimal azimuth CN and vice-versa, and thus one CN configuration (range or azimuth) can be optimized without evaluating every combination of the other CN (range or azimuth). Therefore, the quadruple summation can be separated into the range and azimuth CN combinations as

follows:

$$\begin{aligned} \text{range CN combinations: } & \sum_{X_T=1}^{N_d} \sum_{X_r=1}^{\pi_{X_T}} (1) \\ \text{azimuth CN combinations: } & \sum_{Y_T=1}^{N_d} \sum_{Y_a=1}^{\pi_{Y_T}} (1). \end{aligned}$$

As a result, Eqn. 6.6 can be reduced to the following:

$$E_{\text{hom}} = \bigcup_{X_T=1}^{N_d} \bigcup_{X_r=1}^{\pi_{X_T}} \{(X_T, X_r, X_a = 0)\} \cup \bigcup_{Y_T=1}^{N_d} \bigcup_{Y_a=1}^{\pi_{Y_T}} \{(Y_T, Y_r = 0, Y_a)\}. \quad (6.6)$$

If n is the number of combinations associated with Eqn. 6.5, then the number of evaluations described by Eqn. 6.6 is $2\sqrt{n}$. Eqn. 6.6 simplifies to

$$\begin{aligned} |E_{\text{hom}}| &= \sum_{X_T=1}^{N_d} \sum_{X_r=1}^{\pi_{X_T}} (1) + \sum_{Y_T=1}^{N_d} \sum_{Y_a=1}^{\pi_{Y_T}} (1) \\ &= \sum_{X_T=1}^{N_d} \pi_{X_T} + \sum_{Y_T=1}^{N_d} \pi_{Y_T} \\ &= 2 \sum_{T=1}^{N_d} \pi_T, \end{aligned}$$

where the last equation employs the notation used in the heterogeneous scenario.

The third scenario, which involves both a homogeneous and heterogeneous CN, is a combination of the first two scenarios. Let this mixed scenario be represented by $E_{\text{het,hom}}$. One heterogeneous CN configuration out of all the

feasible combinations expressed by E_{het} is necessary in this case. Because of the independence of the homogeneous range and azimuth CN configurations, exploited by the reduction of Eqn. 6.5 to Eqn. 6.6, all combinations of E_{hom} must also be applied. As a result, the following value for $E_{\text{het,hom}}$ is derived:

$$|E_{\text{het,hom}}| = |E_{\text{het}}| \cdot |E_{\text{hom}}|.$$

The upper bounds for the total number of processor assignment combinations, respective of daughtercard type, that must be considered in the CNM optimization is simply the sum of the expressions for the three scenarios already investigated. That is,

$$|E| = |E_{\text{hom}}| + |E_{\text{het}}| + |E_{\text{het,hom}}|,$$

where $|E|$ represents the total number of evaluations for all scenarios. Note that the above summation also can be expressed by the following:

$$|E| = |E_{\text{het}}| + |E_{\text{hom}}| + |E_{\text{het}}| \cdot |E_{\text{hom}}|.$$

With the S2T16B and S1D64B daughtercards exclusively as choices, E can be easily calculated for the model under investigation. Because $N_d = 2$, let the daughtercard type be the S2T16B if $T = 1$ and the S1D64B if $T = 2$, preserving

the convention of the ISMM. Thus, $\pi_1 = 3$ and $\pi_2 = 2$. With this definition, E_{het} can then be evaluated as follows:

$$\begin{aligned}
 |E_{\text{het}}| &= \frac{1}{2} \sum_{T=1}^{N_d} (\pi_T^2 - \pi_T) \\
 &= \frac{1}{2} \sum_{T=1}^2 (\pi_T^2 - \pi_T) \\
 &= \frac{1}{2} [(3^2 - 3) + (2^2 - 2)] \\
 &= 4.
 \end{aligned}$$

In the same way, $|E_{\text{hom}}|$ is evaluated:

$$\begin{aligned}
 |E_{\text{hom}}| &= 2 \sum_{T=1}^{N_d} \pi_T \\
 &= 2 \sum_{T=1}^2 \pi_T \\
 &= 2(3 + 2) \\
 &= 10.
 \end{aligned}$$

The total number of necessary evaluations is therefore

$$\begin{aligned}
 |E| &= |E_{\text{het}}| + |E_{\text{hom}}| + |E_{\text{het}}| \cdot |E_{\text{hom}}| \\
 &= 4 + 10 + (4)(10) \\
 &= 54.
 \end{aligned}$$

Up to 54 different combinations of processor assignments and card types must be evaluated to ensure optimality. For each combination, the optimization routine must be invoked and the best value, for whatever objective is chosen, of all the combinations and corresponding configuration are declared optimal.

6.3 Minimization of Power

Power minimization is the fundamental case of investigation in this work. Because of the increased computational intensity involved with the CNCM, this model is only applied to the power minimization objective. Furthermore, it is deemed sufficient to illustrate the utilization of this new model by applying it only to the two cases of optimal and nominal mixed card type configurations because the mixed configuration is the most general of all the configurations. Analysis of the solutions of both cases will be carried out, followed by investigation of the utilization of the ISMM as a lower-bounds heuristic for the CNCM.

Similar to the convention set in Subsection 5.1.2, power requirements will be represented as functions of the configuration types. Thus the objective function for the power minimization model is as follows:

$$Z = N_X \Pi_{CN}(X_T) + N_Y \Pi_{CN}(Y_T).$$

Note that with only the S2T16B and S1D64B available, the power function above

is defined as

$$\Pi_{\text{CN}}(I_T) = \begin{cases} 6.1 & \text{if } I_T = \text{S2T16B}, \\ 9.6 & \text{if } I_T = \text{S1D64B}. \end{cases}$$

The above power values reflect the power consumption per CN instead of the power per daughtercard value employed in the ISMM. Although the above function definition of Π_{CN} consists of only two cases, as with this entire model, the number of different types of daughtercards could increase by adjusting the power function without affecting the optimization formulation.

Some reductions can be made to E for the power minimization model, exploiting the fixed nature of velocity and resolution. These reductions result from the determination of either infeasible combinations, based on constraints such as memory, or inexpedient combinations, which can be proved to provide sub-optimal solutions. Note that different optimization objectives entail different methods of reducing E as an upper bound or of reducing the mean E .

In the case of the homogeneous range CN, only one combination needs to be evaluated. Because the range processor and memory requirements (P_r and M_r) are fixed with resolution and velocity, the memory per range processor is also fixed. Therefore the maximum feasible number of range processors per CN is

given by the following expression:

$$\begin{aligned} \max I_r &= \left\lceil \frac{M_{\text{CN}}(I_T)}{\frac{M_r}{P_r}} \right\rceil \\ &= \left\lceil \frac{M_{\text{CN}}(I_T)P_r}{M_r} \right\rceil. \end{aligned} \quad (6.7)$$

Eqn. 6.7 also expresses the viable minimum number of range processors per CN that should be evaluated because there is no advantage to leaving a processor idle if it can be utilized. Thus Eqn. 6.7 represents the only combination that needs to be evaluated for the homogeneously configured range CN. As a result, E_{hom} is modified accordingly:

$$\begin{aligned} |E_{\text{hom}}| &= \sum_{T=1}^{N_d} \pi_T + \sum_{T=1}^{N_d} (1) \\ &= \sum_{T=1}^{N_d} \pi_T + N_d. \end{aligned}$$

For the heterogeneous case, all combinations that do not meet the following constraint can be removed from consideration before the invocation of the optimization routine:

$$M_{\text{CN}}(I_T) \geq I_r \frac{M_r}{P_r} + I_a \frac{M_a(S_a = 1)}{P_a(S_a = 1)}. \quad (6.8)$$

The above condition is also valid for the homogeneous azimuth CN configuration,

where $I_r = 0$. No minimum azimuth processor per CN value can be computed as for the range because the expression is only a lower bound for the optimization variable S_a . A low S_a entails a very high number of processors, but this fact alone does not eliminate any configurations, whereas the memory constraint does do so. However, the upper bound on E remains unchanged because it is possible that the restriction of Eqn. 6.8 will never be active, although the mean E should be reduced.

With application of the above restrictions, the lower bound on E is zero, meaning that the application and radar parameters do not allow for a feasible configuration on the available types of daughtercards. To calculate the new upper bound of E for the power minimization model, with the two familiar daughtercards available, the new E_{hom} first must be computed:

$$\begin{aligned}
 |E_{\text{hom}}| &= \sum_{T=1}^{N_d} \pi_T + N_d \\
 &= (3 + 2) + 2 \\
 &= 7.
 \end{aligned}$$

Note that with a larger selection of daughtercard types, the effect of this reduction would be greater. Considering this change in E_{hom} , the new value of E is

found to be

$$\begin{aligned} |E| &= |E_{\text{het}}| + |E_{\text{hom}}| + |E_{\text{het}}| \cdot |E_{\text{hom}}| \\ &= 4 + 7 + (4)(7) \\ &= 39. \end{aligned}$$

This new value for E results in a 28% reduction in the upper bound of the number of evaluations necessary in the optimization routine.

6.3.1 Optimal Mixed Configuration

Results of the optimization for the optimal mixed card type configuration are given in Fig. 6.1. The smooth surface, nonincreasing in the resolution dimension and nondecreasing in the velocity dimension, is characteristic of an optimal power surface. Although comparison of the CNCM power graph to that of the ISMM will be presented later in this chapter, it is noted that the graph is almost identical in shape to that of the ISMM (see Fig. 5.4). However, important differences resulting from the additional constraints and variables introduced in the present model necessitate new approaches to analysis of the data.

The most important new information available from the new model is the configuration variables I_T , I_r and I_a . It is feasible to present all the information from all three of these variables concisely on one graph because only a subset of

the total possible permutations is found to include optimal solutions in the range of values investigated. For labeling purposes, let the following notation be used: a configuration is represented by one or two three-digit numbers. Each number is constructed digit by digit with the CN type first, then the number of range processors per CN of that type, and then the number of azimuth processors per CN of that type. Both of the two possible CN configurations are encoded in this way as part of a total system configuration. That is, a system configuration is denoted by the following: $X_T X_r X_a \quad Y_T Y_r Y_a$, where I_T may be represented by a ‘1’ or a ‘2’ in the present case, denoting the S2T16B or the S1D64B, respectively.

For example, if the optimization routine determined that the optimal system configuration consisted of assigning three range processors and no azimuth processors to the S2T16B, and two azimuth processors and no range processors to the S1D64B, the resultant representation is: 130 202. In the purely heterogeneous scenario, an optimal system configuration might assign one processor each on the S1D64B to range and azimuth processing. The representation for this case would be simply: 211.

The corresponding optimized system configurations for the power surface of Fig. 6.1 is shown in Fig. 6.2. The configuration notation explained above is employed in the legend of the graph.

The majority of system configurations are purely homogeneous CN configurations in which each CN is dedicated to either range or azimuth processing. As

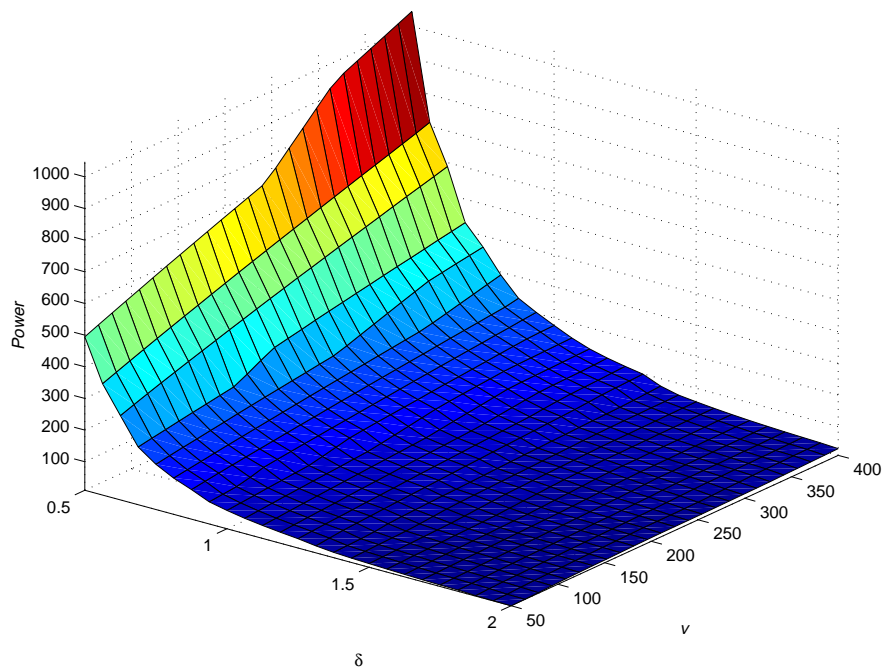


Fig. 6.1: Minimal power for optimal mixed configuration.

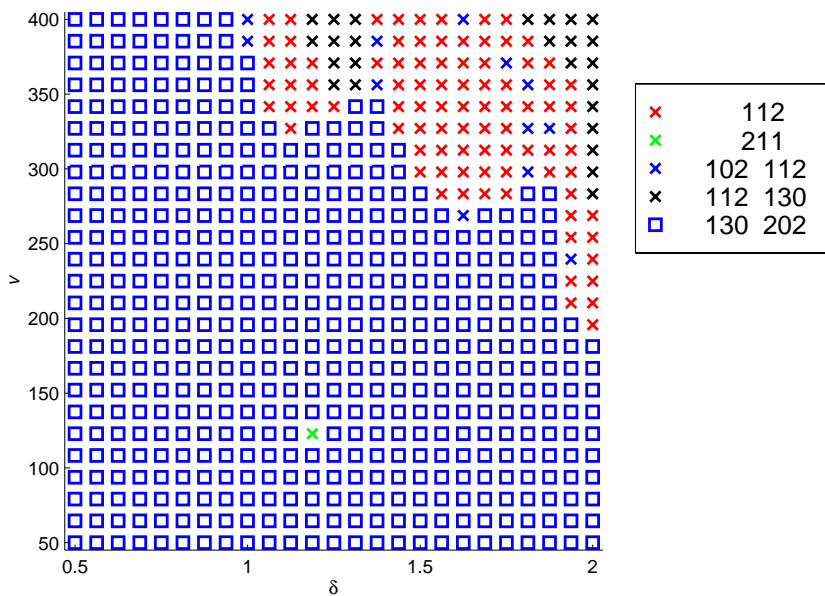


Fig. 6.2: System configurations for minimal power in optimal mixed configuration.

would be expected, for this configuration the S2T16B is used for range processing and the S1D64B for azimuth processing. As resolution becomes coarser and velocity decreases, other optimal configurations appear. Primarily motivated by decreasing memory demands resulting from coarser resolution, the heterogeneous configuration is optimal in which only the S2T16B is utilized with one range and two azimuth processors. It is expected that if power were optimized over even lower resolutions than 0.5 m, the heterogeneous configuration employing only the S1D64B would become more common.

The optimal azimuth FFT size F_a in the present model is very similar to that in the ISMM. Fig. 6.3 illustrates this resemblance. However, high velocity in conjunction with fine resolution entails a decreased azimuth memory to processor ratio relative to lower velocities at the same resolution. With no advantage in conserving memory that no other processors can use, the two processors on the S1D64B (note that the corresponding configuration at these resolution-velocity pairs is: 130 202) optimize memory usage by increasing the FFT size.

The azimuth section size S_a closely resembles that of the ISMM just as does F_a . Fig. 6.4 shows the surface of S_a . Note the corresponding additional peak in S_a as in F_a at the high performance velocity-resolution pairs. The graph of the azimuth kernel size, independent of optimization variables, is the same as in the ISMM (see Fig. 5.10).

The ratio of the azimuth kernel size to the section size (K_a/S_a) differs slightly

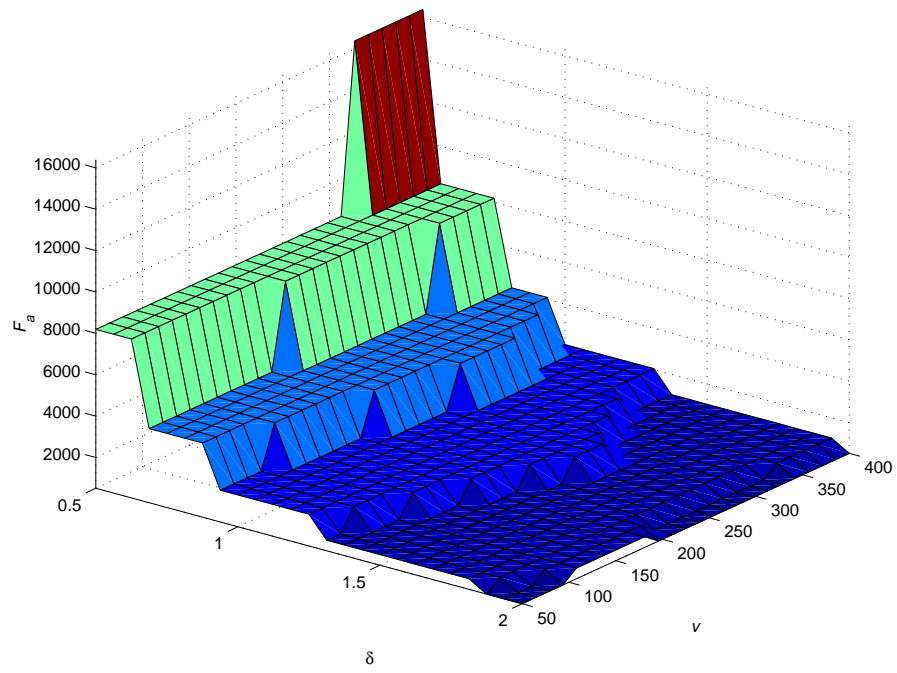


Fig. 6.3: FFT size in optimal mixed configuration.

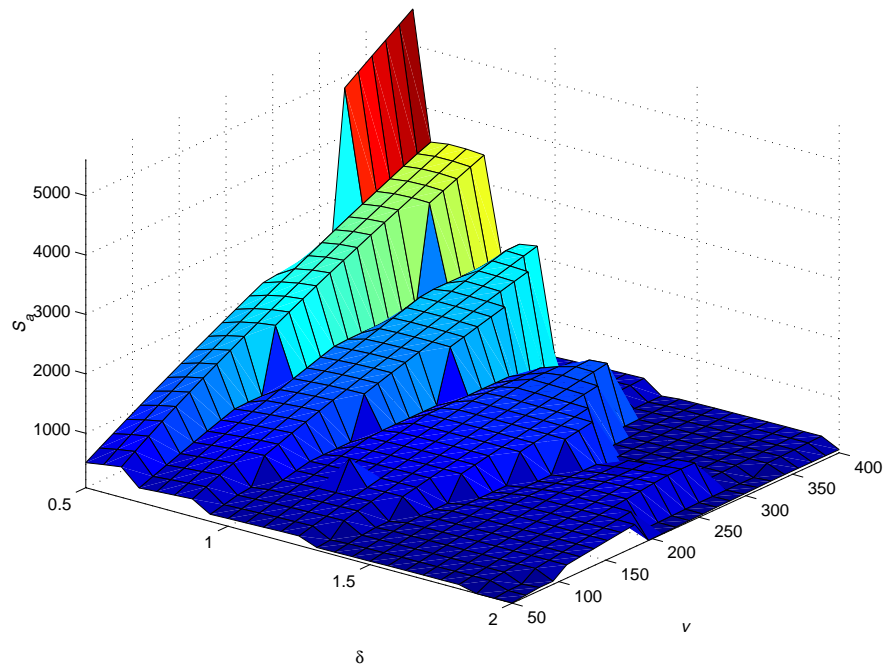


Fig. 6.4: Section size in optimal mixed configuration.

from that of the ISMM because of the motivation to not waste memory available on a CN. Fig. 5.20 shows that this ratio tends to be lower in the coarse resolution area because S_a is increased to utilize all available memory.

Azimuth memory and processor requirements are slightly different from their counterparts in the ISMM. Fig. 6.6 shows an increase in memory over the ISMM at the high performance corner of the surface, corresponding to the increase in FFT size at the same location observed in Fig. 6.3.

Azimuth processor requirements (Fig. 6.7) exhibit a premature rise and abrupt fall in the area of high velocity and coarse resolution. This seemingly anomalous feature is explained by the configuration graph (Fig. 6.2). Note that this raised area on the processor graph corresponds to the area on the configuration graph in which the memory-poor S2T16B is employed for both range and azimuth processing.

The ratios of azimuth memory and processors to range memory and processors, respectively, are shown in Figs. 6.8 and 6.9. Note that the discontinuities in the memory ratio graph is more accentuated in the CNCM than in the ISMM, while the undulations in the processor ratio graph are smoother (compare to Figs. 5.8 and 5.9). One explanation for this observation involves the optimization motivation to utilize all the available memory on a CN, combined with the restriction of using only the memory on a single CN. With the range processing and memory requirements the same for both models, only azimuth

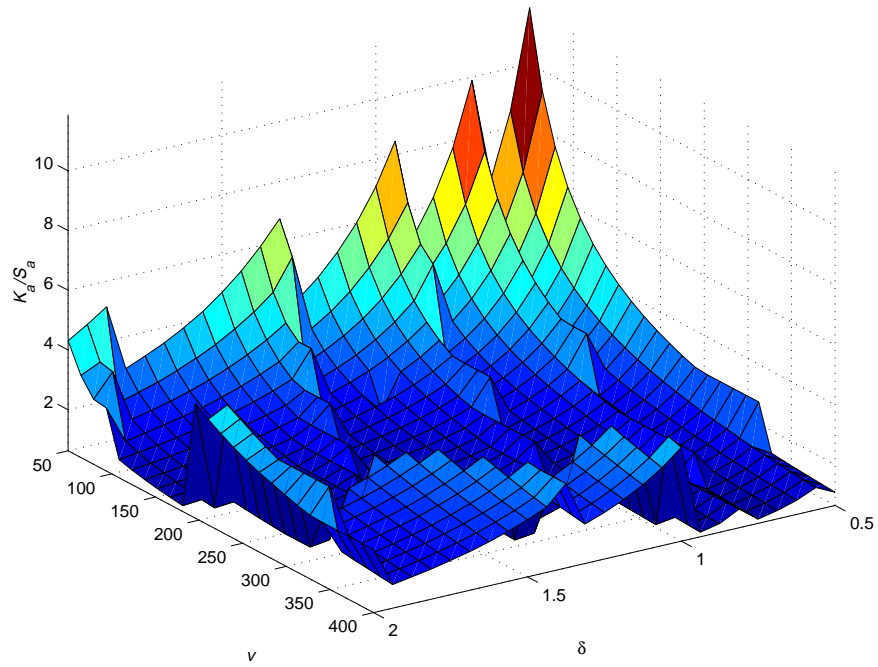


Fig. 6.5: Ratio of kernel size to section size in optimal mixed configuration.

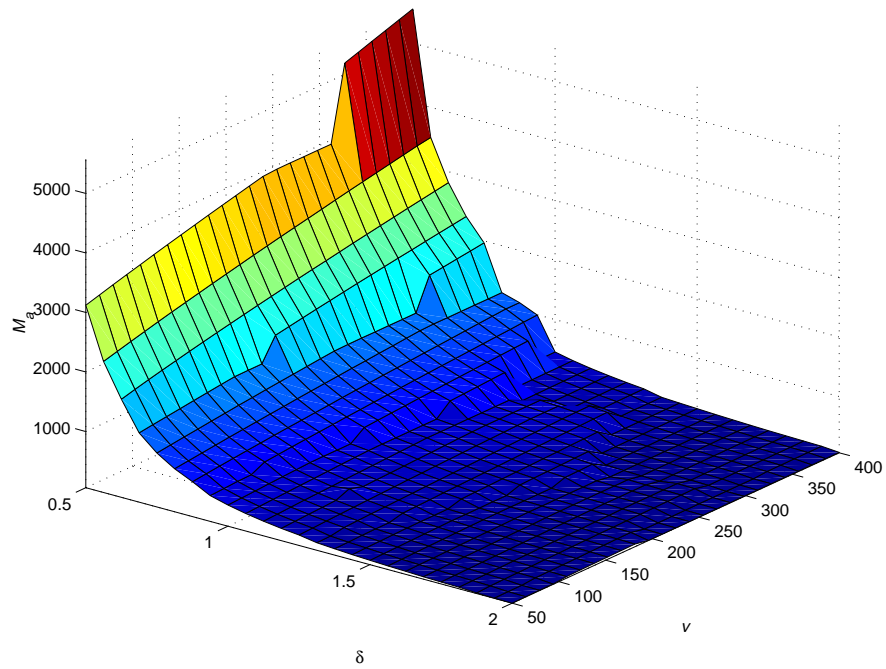


Fig. 6.6: Azimuth memory requirements in optimal mixed configuration.

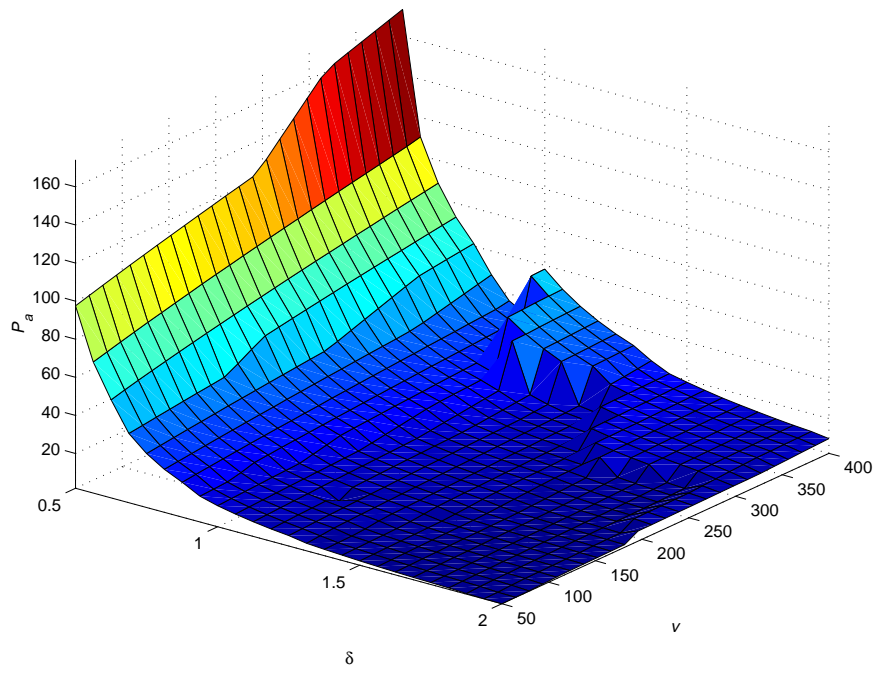


Fig. 6.7: Azimuth processor requirements in optimal mixed configuration.

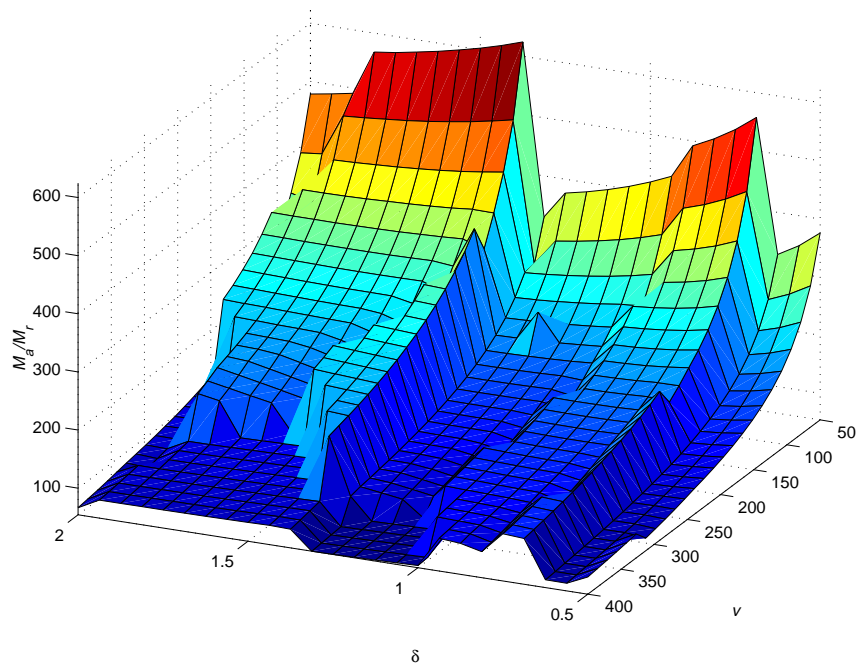


Fig. 6.8: Ratio of azimuth to range memory in optimal mixed configuration.

processing can be modified to take advantage of extra memory. As a result, the azimuth memory per processor tends to be more uniform. Fig. 6.10 illustrates this tendency, with a lone spike corresponding to the configuration in which the S1D64B was employed with one range and one azimuth processor. This uniformity in azimuth memory per processor entails greater discontinuities in the memory requirements, dependent on the FFT size and section size. The number of azimuth processors also becomes slightly more uniform because optimization of available memory on a card prevents extremely inefficient processing, as might be optimal in the ISMM.

Azimuth memory per processor is one of the most noticeable differences between the CNCM and the ISMM. In the ISMM, memory per processor was not a concern because memory is pooled. For the sake of comparison, the azimuth memory per processor for the ISMM is plotted here in Fig. 6.11. The familiar rolling effect is evident here, associated with the different FFT sizes. The surface shape of Fig. 6.11 is almost identical to that of the ratio of the section size to the FFT size, with the reversal of the velocity axis (see Fig. 5.12).

The other dramatic difference in corresponding optimization variables between the two models is the usage of the two types of daughtercards. As opposed to the undulating characteristic apparent in the ISMM (see Figs. 5.13 and 5.14), the CNCM favors the S1D64B, except in a small area of coarse resolution and high velocity, where the S2T16B is exclusively employed. Figs. 6.12 and 6.13

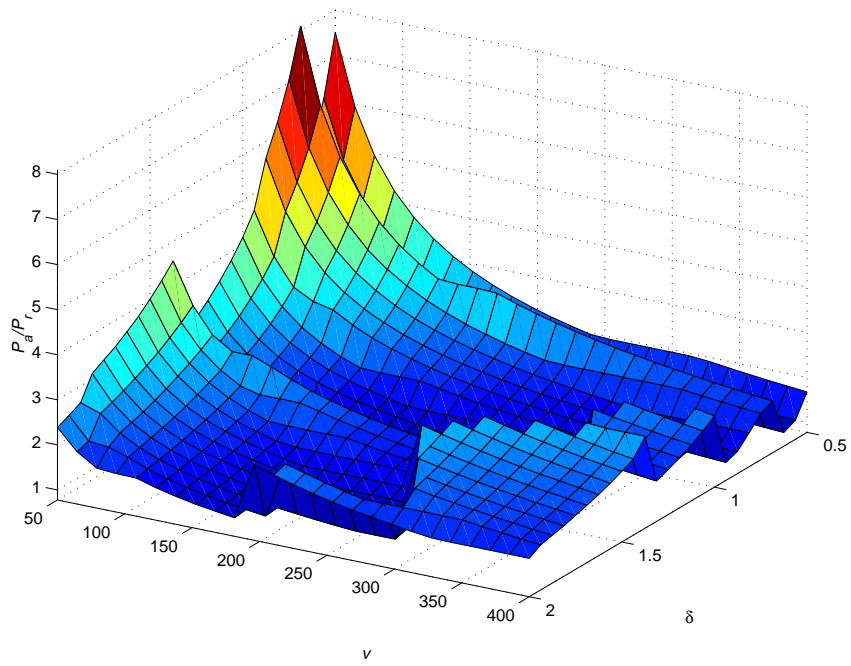


Fig. 6.9: Ratio of azimuth to range processors for optimal mixed configuration.

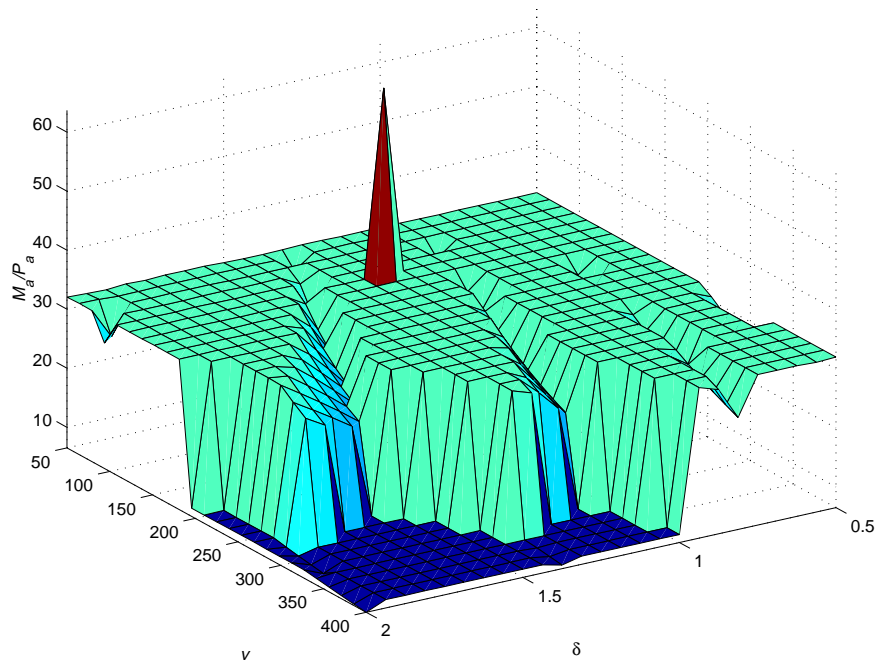


Fig. 6.10: Azimuth memory per processor for optimal mixed configuration in CNCM.

illustrate this trend. Because azimuth requirements dominate the system, homogeneous use of the S1D64B for azimuth processing results in this favoritism, although the S2T16B is homogeneously employed for range processing. In the corner of the graph where velocity is high and resolution coarse, reduced azimuth memory requirements allow implementation of azimuth processing as well on the S2T16B.

6.3.2 Nominal Mixed Configuration

As for the ISMM, comparison of the optimal configuration to the nominal configuration is used to measure the utility of optimization. However, unlike in the ISMM, in which the nominal configuration was merely a simplified and specific case of the optimal configuration, with the section size removed as an optimization variable and set equal to the kernel size, the nominal configuration in the CNCM calls for special attention and a slightly more complicated formulation for the nominal configuration to compete with the optimal. Both the simplistic approach, in which no special attention is given, and the more complicated approach will be examined. Because of this difference in formulation, the two approaches to the nominal configuration presented here will be denoted as the *naive* and *sophisticated* approaches. The general formulation of the nominal problem is the same as for the optimal configuration but with the aforementioned removal of S_a as an optimization variable.

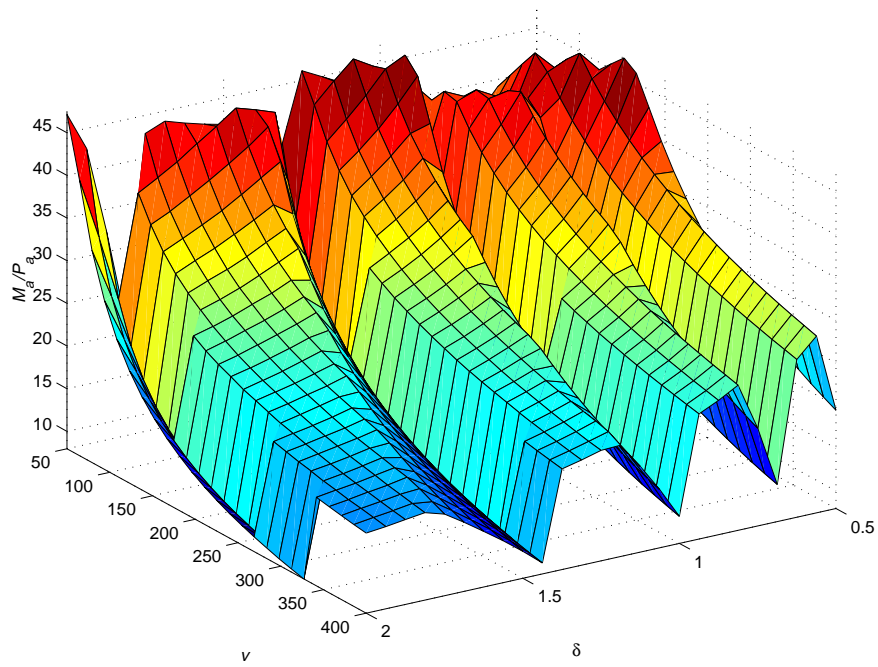


Fig. 6.11: Azimuth memory per processor for optimal mixed configuration in CNCM.

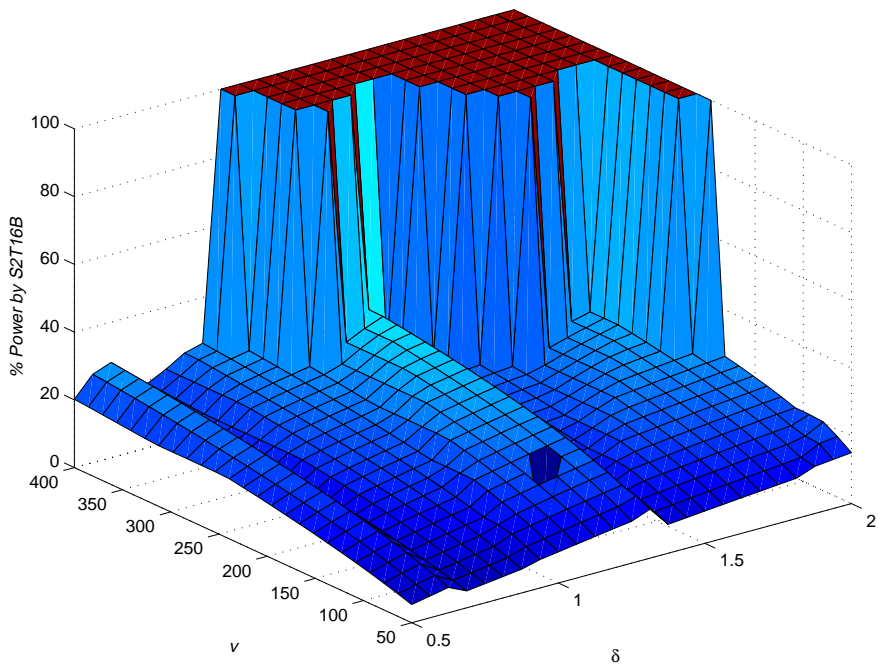


Fig. 6.12: Percentage power consumption by S2T16B for optimal mixed configuration.

The added complexity in the nominal configuration formulation results from the lack of optimizing the section size so as to correlate the employed processors with the available memory. Reliance by the present formulation on the memory per processor values to enforce the local memory access constraint (Eqns. 6.1 and 6.2) entails a serious problem for the nominal configuration. Because both azimuth processor and memory requirements are fixed, the ratio of memory to processors is also fixed. Although this fact alone does not prohibit optimization in most cases, there is a set of resolution and velocity pairs that do not permit a feasible solution regardless of how much power is allocated because the memory per azimuth processor exceeds 64 MB, the upper limit on memory per processor for the two daughtercards under consideration. As it might be surmised, in many cases even the solutions that are feasible are rather poor because there is such an inefficient use of memory. Fig. 6.14 displays the excessive power requirements of the nominal configuration, with approximately 15% of the area investigated infeasible.

This problem at first seems to be a reasonable and even expected penalty for not optimizing the section size, which has been seen to be so critical in the ISMM. However, inspection of Fig. 6.14 reveals a disturbing observation: the area of infeasibility does not occur at the highest performance corner of the graph but in the area at which resolution is fine and velocity is low. At increased velocities, solutions become feasible. This unintuitive result calls for a

new formulation. Such a formulation still may not find feasible solutions in every case, but it should not produce feasible solutions by increasing the requirements.

Fig. 6.15 depicts the optimal card and processor assignments with the nominal section size. It is noted that only one azimuth processor is allocated on the S1D64B in the feasible solution area immediately outside the infeasible solution area. This assignment infers a shortage of memory, as is expected by the nominal configuration in which the kernel size is too large to act as a section size. Fig. 6.16 confirms this suspicion that the memory per processor is greater than 64 MB at the points at which no solutions were found.

The extreme memory per processor values in Fig. 6.16 result from a high memory requirement due to the fine resolution but yet a low processor requirement due to the low velocity. In the research presented thus far, it has been accurate to assume that 100% processor utilization occurs. In the ISMM, this point is not relevant because a high memory to processor ratio never explicitly causes infeasibility. In the optimal configuration of the CNCM, the section size is always optimized such that resources are not idle. (In tests conducted after the original data was collected for the previous subsection on the optimal configuration, 100% processor utilization was shown to be optimal for every resolution and velocity value investigated. However, it is plausible that for more extreme application parameter values, allowing less than 100% processor utilization might

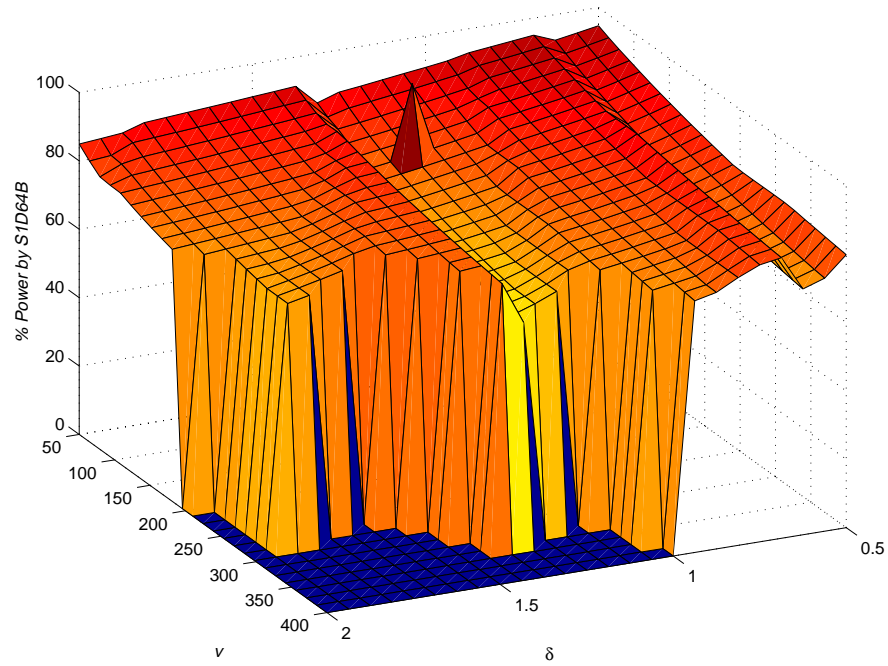


Fig. 6.13: Percentage power consumption by S1D64B for optimal mixed configuration.

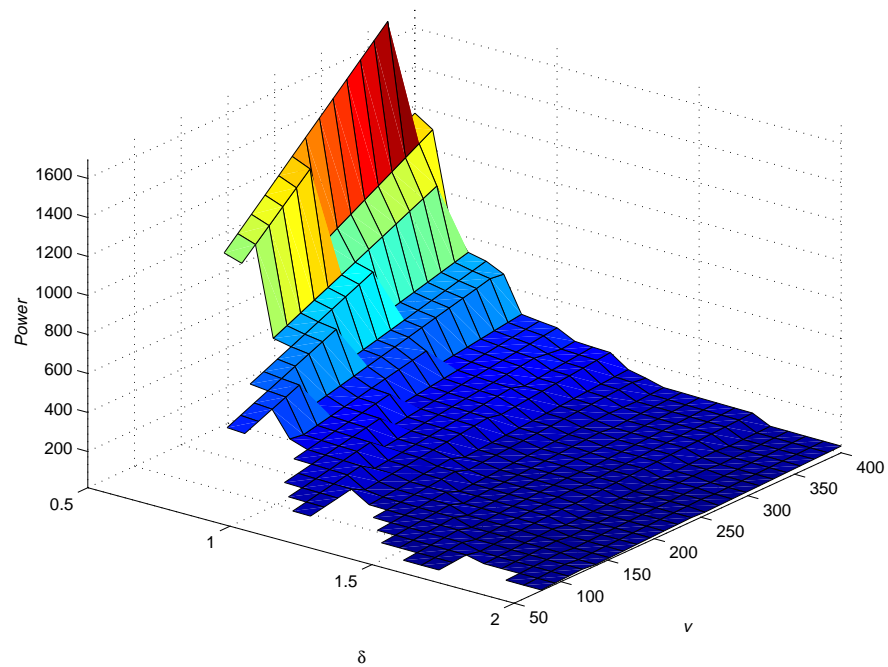


Fig. 6.14: Power consumption in naive approach to nominal mixed configuration.

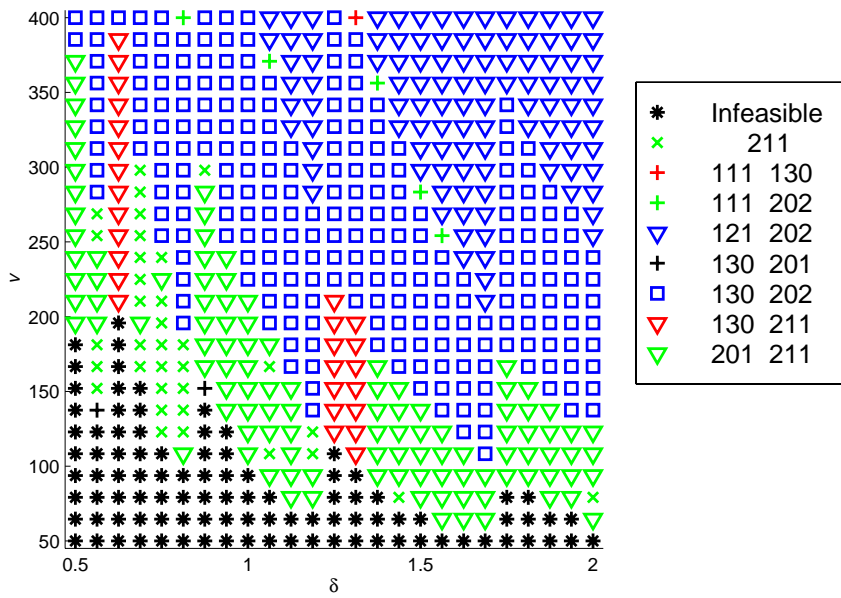


Fig. 6.15: Configurations in naive approach to nominal mixed configuration.

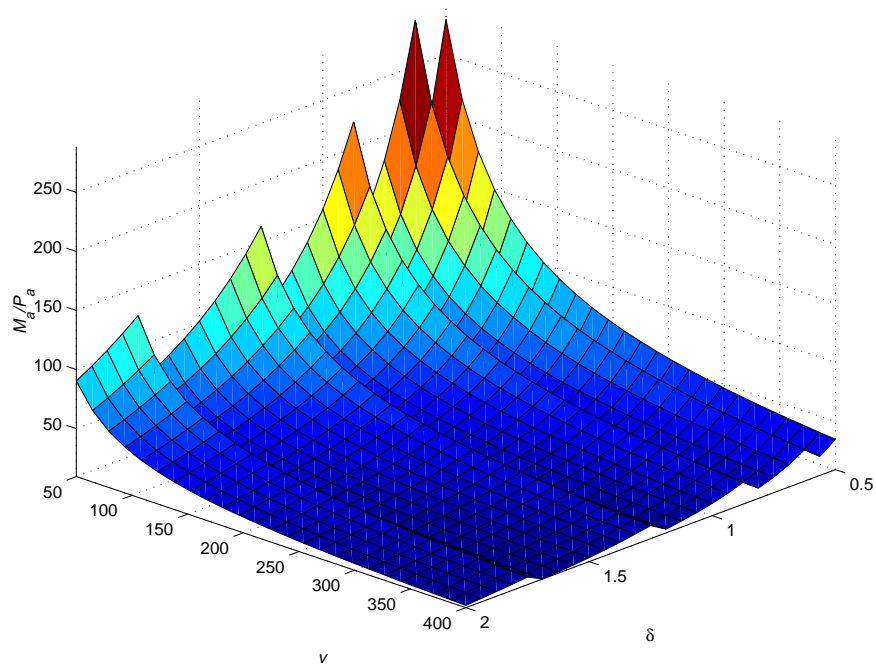


Fig. 6.16: Azimuth memory per processor in naive approach to nominal mixed configuration.

be necessary even with an optimized section size.)

To rectify an excessive memory to processor ratio, either the required memory must be decreased or the number of processors increased. Because the memory required is dependent on section size, which in the nominal case is fixed, memory requirements cannot be modified. However, the number of processors can be increased, thus reducing the individual processor utilization percentage.

One method to achieve feasible solutions in the nominal case is to calculate the actual number of processors P_{act} (as opposed to the required number of processors P_a , assuming 100% utilization) by the following equation:

$$P_{\text{act}} = \frac{P_a}{U_p}, \quad (6.9)$$

where U_p denotes the processor utilization as a ratio, and has a range (0,1].

However, the necessary U_p somehow still must be computed.

The most basic approach to calculating P_{act} is to consider the ratio of the required memory and the maximum available memory per CN. Let M_{max} represent this maximum available memory. If T is the target card type, then $M_{\text{max}} = M_{\text{CN}}(T)$. If the target card type is not previously determined, and the most versatile (in terms of feasibility) card type is desired, then the card type with the greatest amount of memory per CN should be employed. That is, $M_{\text{max}} = \max\{M_{\text{CN}}(T) : T = 1, 2, \dots, N_d\}$, where T represents the daughtercard

types denoted by consecutive integers beginning with ‘1’. Thus, one solution to the actual number of processors is

$$P_{\text{act}} = \max \left\{ P_a, \frac{M_a}{M_{\text{max}}} \right\}. \quad (6.10)$$

With the two daughtercards under consideration, M_{max} is 64.

Eqn. 6.10 implies the following expression for U_p :

$$U_p = \min \left\{ 1.0, \frac{P_a M_{\text{max}}}{M_a} \right\}. \quad (6.11)$$

With this formulation, it is assured that if a feasible solution exists with the nominal section size, the lack of a large number of required processors will not prevent finding a solution. The solution found, however, may be poor since U_p is set to its maximum possible value. In cases where all the memory is dedicated to one processor on a card, regardless of how many processors are located on the card and how little memory range processors may require, lack of consideration of less than maximum values of U_p will often entail additional wasted resources on other cards.

To compare the nominal section size with the optimal section size without this added disadvantage of the nominal formulation, either U_p or P_{act} must be optimized. Instead of only two optimization variables in the nominal configu-

ration, a third variable is introduced into the formulation. Let P_{act} be the new optimization variable so that the formulation may be modified by replacing every occurrence of P_a in the constraints with P_{act} and setting P_a as a lower bound for P_{act} .

The power requirement results of such a formulation are shown in Fig. 6.17. Notice that they more closely emulate the relationship between the optimal and nominal configurations in the ISMM, with the nominal requiring approximately 30% more power. Furthermore, solutions exist for the entire range of resolution and velocity values.

The configuration graph for the sophisticated nominal model is presented in Fig. 6.18. Comparing this graph to that of the naive approach, it is observed that where previously there were infeasible solutions and solutions that dedicated the entire S1D64B to one azimuth processor, now the S1D64B is used to accommodate one azimuth processor and one range processor. This heterogeneous use of the CN greatly reduces the overall power consumption because a range processor can be assigned to a CN on the S1D64B essentially for “free,” because the ratio of azimuth memory to range memory is so high in these areas.

It is obvious that the freedom granted to the formulation to add processors above the strict requirement as computed from the equation for P_a (see Eqn. 3.3) improves the calculated power consumption of a system. There is a definite penalty paid for this improvement, however, and the penalty is in terms of the

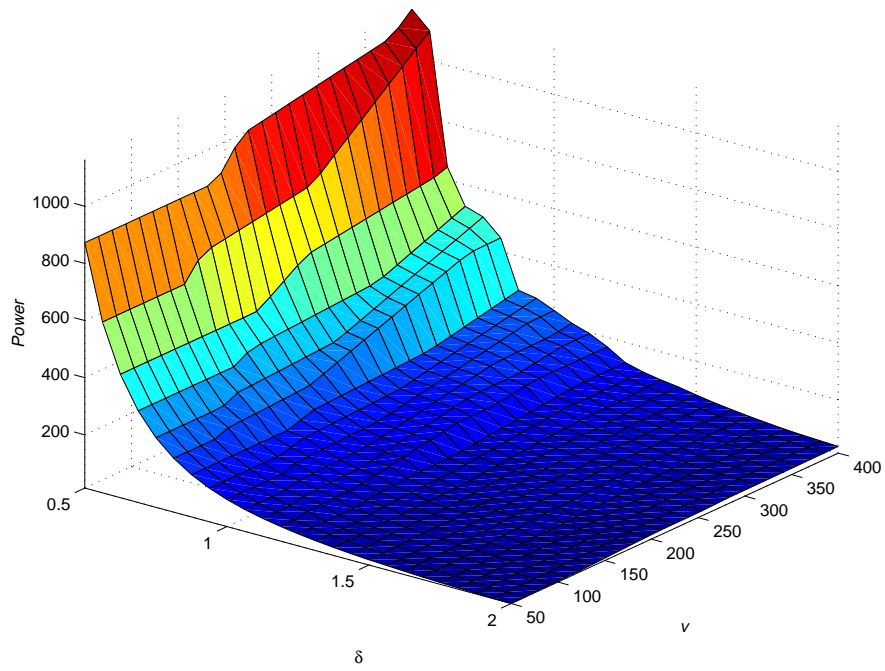


Fig. 6.17: Power requirements for the sophisticated approach to the nominal mixed configuration.

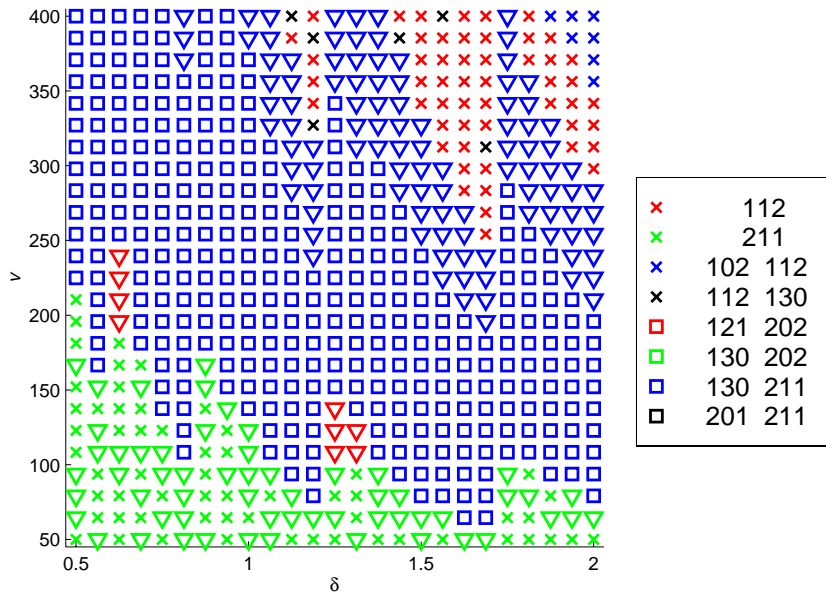


Fig. 6.18: Configurations for the sophisticated approach to the nominal mixed configuration.

extra processors added to reduce the processor utilization and thus the memory per processor. Figs. 6.19 and 6.20 show the number of added processors (i.e., $P_{\text{act}} - P_a$) and U_p , respectively. Note that as the performance requirements of the system decrease, the utilization also increases because the memory-per-processor constraint becomes inactive.

Comparing the two approaches, the ratio of the power requirements of the naive to the sophisticated approach is plotted in Fig. 6.21. Note that the area which is infeasible in the naive approach cannot be compared. The maximum ratio is 1.67 and occurs at the resolution-velocity coordinates $\{0.75 \text{ m}, 239.6 \text{ m/s}\}$. The minimum ratio is 1.00, occurring at the coordinates $\{0.94 \text{ m}, 137.5 \text{ m/s}\}$ and approaching unity in many places. The mean ratio over the graph of mutually feasible areas is 1.30. Note that this figure would be higher if the naive approach found solutions in the infeasible area because its performance in this area would be worst.

6.3.3 Comparison of Optimal and Nominal Configurations

Comparisons will be made of the optimal configuration with both the naive and sophisticated nominal configurations. It is expected that the sophisticated nominal configuration more closely resembles the nominal configuration in the ISMM in relation to the optimal configuration. Whether it is reasonable to use a sophisticated optimization approach to the nominal case, possibly violating

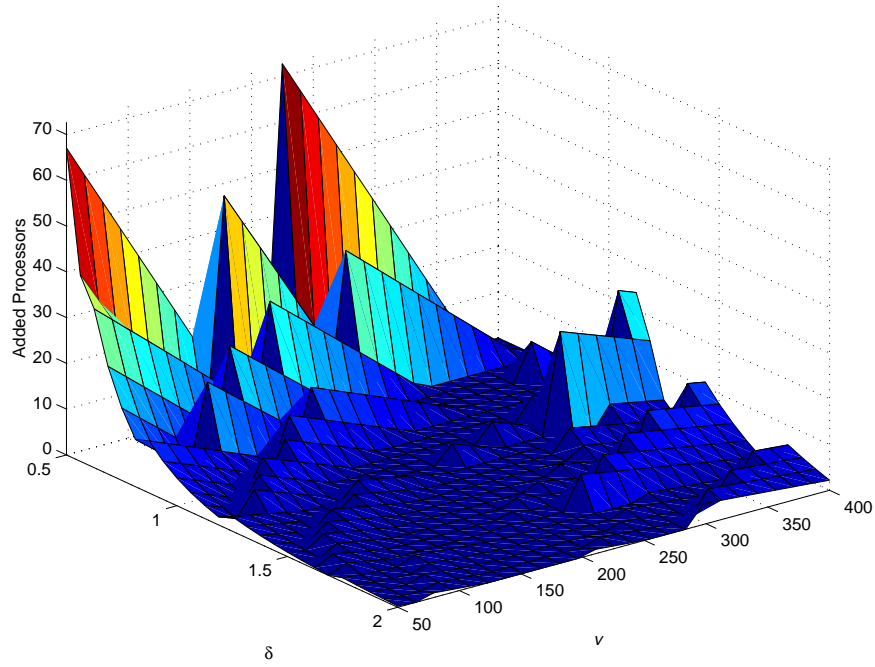


Fig. 6.19: Added processors in the sophisticated approach to the nominal mixed configuration.

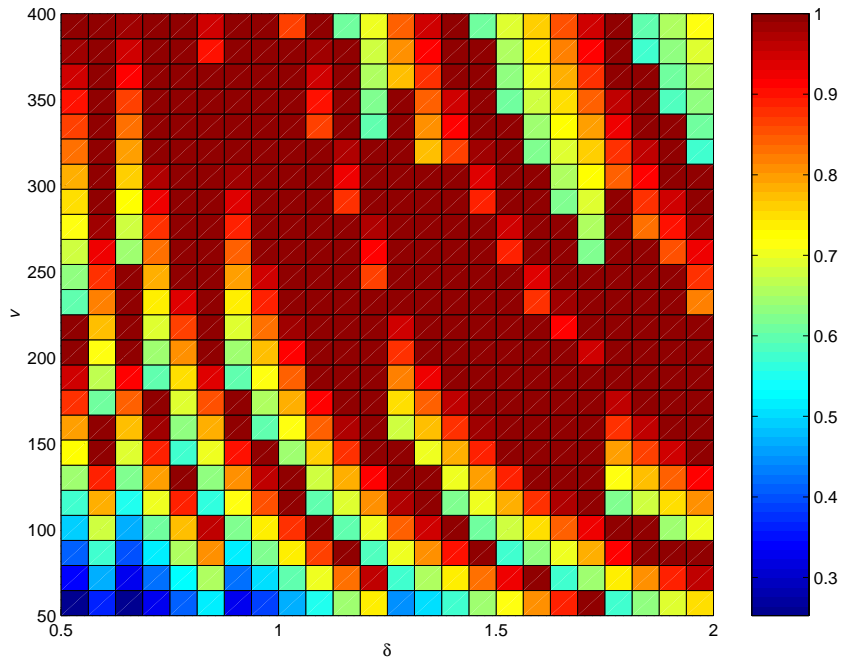


Fig. 6.20: Utilization in the sophisticated approach to the nominal mixed configuration.

the term ‘nominal’, is an issue that will be discussed in the conclusions. In this subsection only the power requirements are briefly considered.

The weakness in the naive approach to the nominal configuration is in its propensity to infeasible or extreme solutions. Fig. 6.22 displays the ratio of the naive nominal power requirement to the optimal power. The plot shows a maximum ratio of 2.15 occurring at the coordinates {0.5 m, 50 m/s}, and a minimum ratio of 1.08 occurring at {0.63 m, 195.8 m/s}. The mean ratio over all the area in which the naive approach found a solution is 1.30.

The sophisticated approach to the nominal configuration (see Fig. 6.23) avoids the extreme ratios present in the naive approach by reducing the maximum ratio to 1.76. This figure is still high but closely reflects the values computed in the ISMM. Furthermore, it must be considered that this maximum value occurs at the coordinates {1.69 m, 400 m/s}, which is in the infeasible region of the naive approach. The minimum ratio is approximately the same as in the naive approach, with a value of 1.08 at {0.94 m, 254.2 m/s}. The mean ratio of 1.33, however, is worse than in the naive approach. This aberration is easily explained, though, by taking into consideration the infeasible area of the naive approach that is not calculated in that average, but is calculated in the average of the sophisticated approach. The two approaches cannot meaningfully be compared with these statistics then because of the infeasible region of the naive approach.

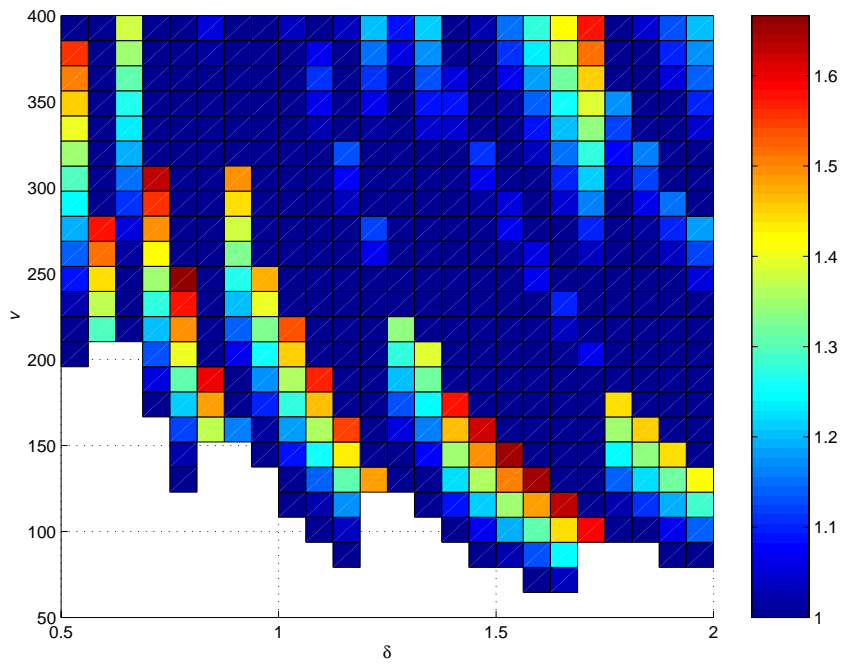


Fig. 6.21: Ratio of the power requirements of the naive to the sophisticated approach to the nominal mixed configuration.

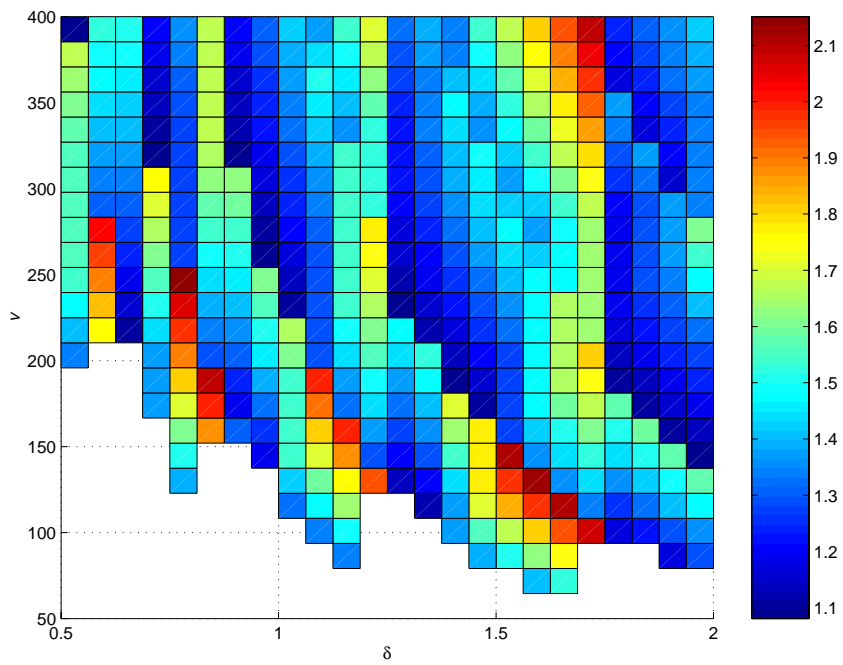


Fig. 6.22: Ratio of the optimal power requirements to those of the naive approach to the nominal mixed configuration.

6.3.4 Effects of Integer Numbers of Cards

Taking the realism of the formulation a final step farther, the effects of discretizing the number of cards N_I is explored. Up to this point, real values for the number of daughtercards have been allowed to more accurately view the effects of individual variables and variable relationships. Especially when the total number of cards required is low, forcing the number of cards to be integers greatly alters the overall results. However, to completely specify a system, it must be accepted that only integer cards are installed, even if resources are wasted to some degree.

It is assumed that the optimization routine developed in this chapter is still employed up to the point of determining the best solution out of all the configurations evaluated. Pure integer programming would provide an absolutely optimal solution to the problem, but such an optimization is infeasible in terms of time for most scenarios considered. Such a method could be applied if it is known that the total number of cards required, whatever the exact configuration, is very low. In this case an exhaustive search and optimization of all feasible integer solutions could be applied. This approach is not viable, however, for the vast number of scenarios that this research purposes to address. Therefore, the following discussion is premised on an available solution involving real values for the number of cards, the raw results of which were presented in the previous two subsections.

One approach to discretization is simply to round the numbers after optimization. Although this approach yields a good average approximation to the actual requirements, rounding off promises neither to be optimal nor even feasible. Rounding off is safe and probably optimal when card number values are near the next integer. In this case, the ceiling of a value is taken. However, in the case that rounding off calls for truncating the decimal portion of a number (i.e., rounding down), there exists the possibility of cutting resources below the required levels. Rounding down is never possible in the case of a single card type configuration or in the case in which both card number values are to be rounded down in a two-card-type (purely homogeneous) configuration.

Rounding up of values is always safe, i.e., permits a feasible value, but does not promise optimality, even in the integer sense. There exists the possibility that in a homogenous configuration the floor can be taken of one of the card number values. If this state is true, a savings of the power consumed by one daughtercard is entailed for the overall power requirement. It is therefore useful to check for this possibility, especially in systems in which the total number of cards required is low.

Disparity in the characteristics of daughtercards, as is true with the two cards under investigation in this research, often causes the probability to be low that one card can be rounded up and the other down. Greater differences in the resources supplied by each card imply lower probabilities that the rounding up

of one card can supply what is sacrificed by the rounding down of the other.

To ameliorate this problem, the card number values are fixed at the proposed rounding values and the problem is then reoptimized. The more appropriate term for this operation is *solving the problem with constraints*. Note that with the card configuration variables set (it is assumed that the processor assignments I_r and I_a remain constant) then the only remaining optimization variable is the section size S_a . The operation does not need to optimize S_a because it is insignificant whether all or part of the resources on all cards is used. Determining the absolute feasibility of the system with the fixed resources is the only goal sought with this operation.

For each real solution returned by the initial optimization routine, two permutations of rounding the card number values must be evaluated. Assuming a purely homogeneous solution with two card configuration types involved, the permutations that must be considered are the ceiling of one and the floor of the other, or the floor of the first and the ceiling of the second. It is not necessary to evaluate the floor of both card number values because this clearly violates the required resources. Similarly, neither is it necessary to evaluate the ceiling of both card number values because in this case the requirements are clearly met. Nevertheless, the addition of the two permutations that must be evaluated effectively triples the number of total optimizations that must be performed, although these two optimizations converge quickly because there is only one

optimization variable.

An additional consideration in the discretization of the number of required daughtercards is the number of CNs per card. Because the optimization routine developed in this chapter returns solutions in terms of CNs and not daughtercards, the number of CNs must be converted to daughtercards, rounded, and then converted back to CNs for the reoptimization (or solving) process. The equivalent technique implemented to collect the results presented consists of rounding the number of CNs to a multiple of the number of CNs per that type of card. In the case of the S2T16B, values are taken to the nearest multiple of two. In the case of the S1D64B, the floor or ceiling of the number of CNs constitute the same integer value in terms of daughtercards.

With all the considerations mentioned and the discussed technique applied, one card number value allowed taking its floor in approximately 15% of the solutions produced for the optimal mixed configuration in this model. The other 85% of solutions required taking the ceilings of both card number values.

Fig. 6.24 illustrates the effect on power consumption of forcing the number of cards to be integers. As expected, there is no significant effect on the overall graph (compare to Fig. 6.1), especially at the highest performance. Lower performance scenarios in which relatively few cards are present in the system suffer a more dramatic effect. Fig. 6.25 shows the ratio of power consumed by this integer version of the solutions to the real-valued requirements.

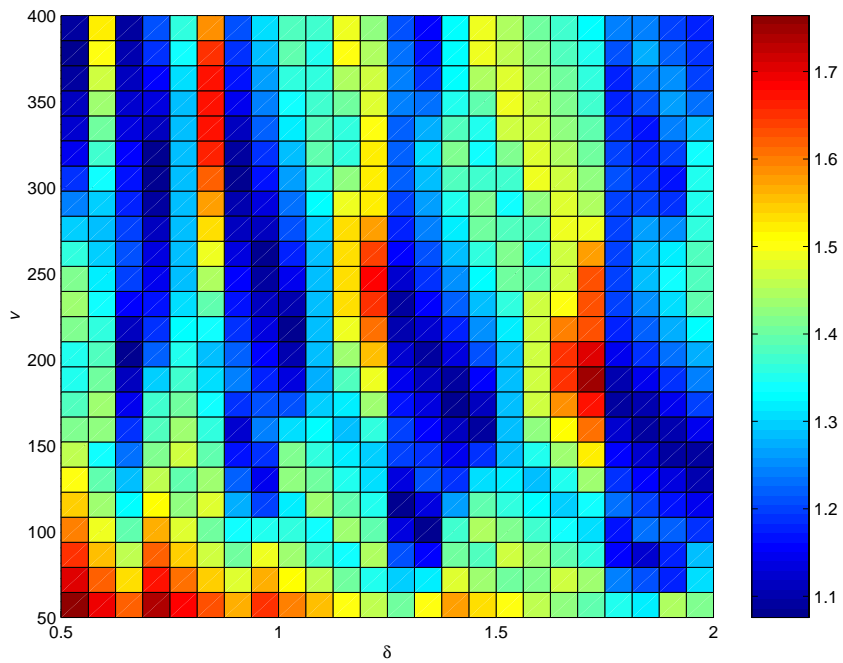


Fig. 6.23: Ratio of the optimal power requirement to those of the sophisticated approach to the nominal mixed configuration.

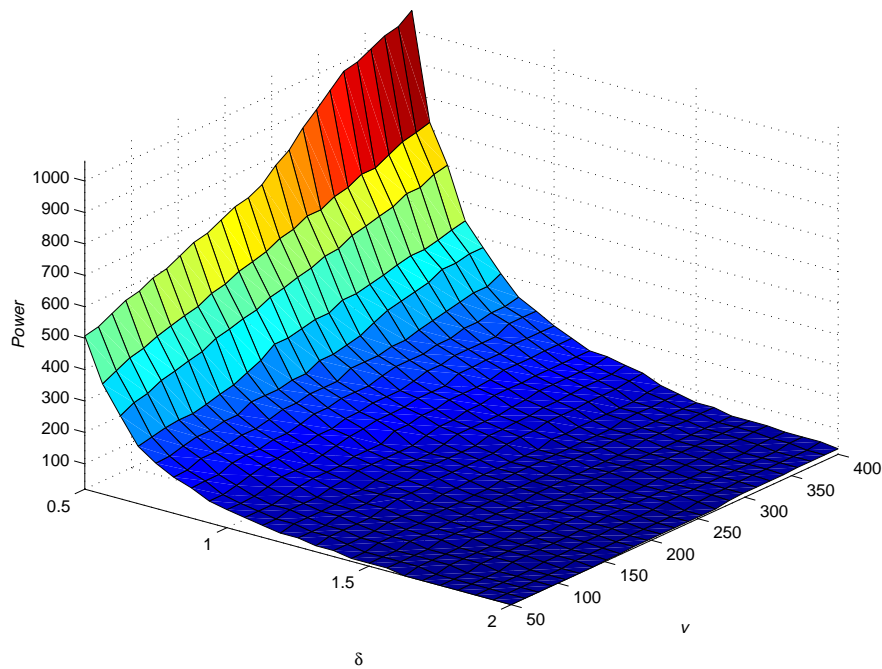


Fig. 6.24: Power requirements of the discrete card number solution to the optimal mixed configuration.

A more noticeable effect of the discretization is observed in the graph of the configuration variables (Fig. 6.26). Although the general trends remain unchanged from the original solution (Fig. 6.2), other configurations are found to be optimal where several configurations produce close power requirements in the real-valued solutions. Discretization of the values often changes the initial optimal configuration.

In addition, the intrinsic optimality of some configurations seems to be doubtful. In particular, the configuration ‘112 201’ appears inherently suboptimal. It appears illogical that two azimuth processors can share the 16 MB of memory on a S2T16B, but then only one azimuth processor is assigned to use the 64 MB of memory on the S1D64B. However, it must be noted that only one such card of the ‘201’ type is employed. The optimization routine originally configured the system to employ only the ‘112’ type and had a fractional processor left over on a card upon discretization. The routine therefore determined that one S1D64B consumed less power than one S2T16B, recalling that the discretization is at the daughtercard level and not at the CN level. In terms of CNs, the S1D64B consumes more power per CN than does the S2T16B. The optimization routine also could have determined that this one card should be of the ‘202’ type without affecting any other system values.

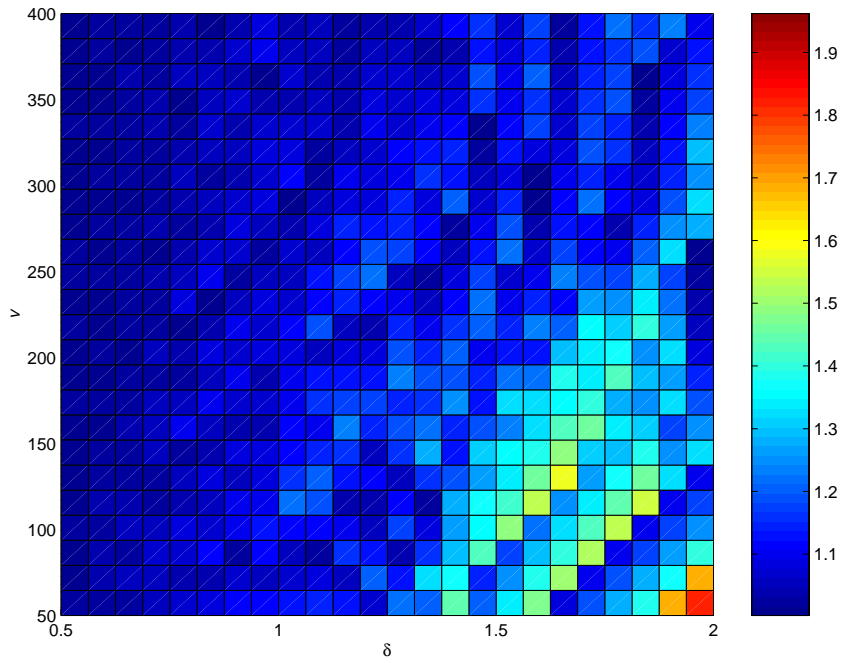


Fig. 6.25: Ratio of power by discrete card number solution to real-valued solution in optimal mixed configuration.

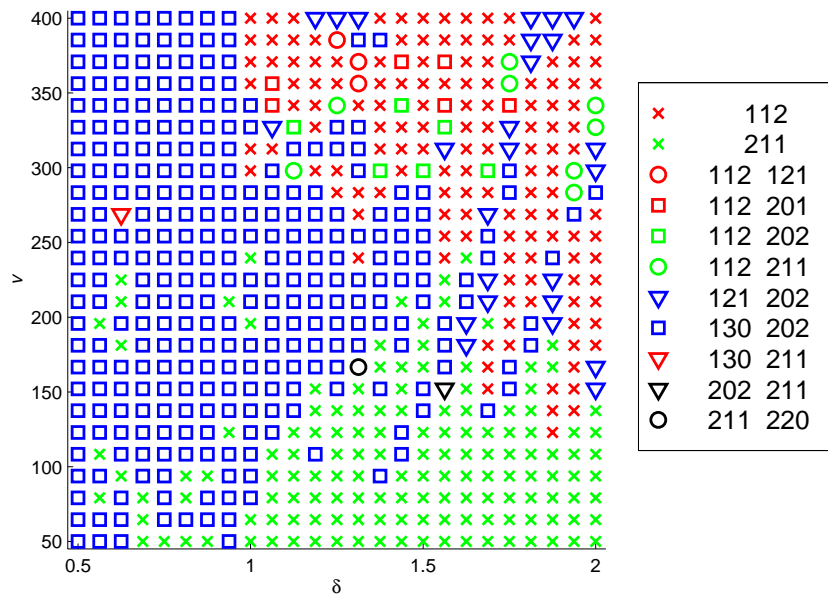


Fig. 6.26: Configurations of discrete card number solutions for the optimal mixed configuration.

6.3.5 Comparison of CNCM and ISMM

The approximation of the ISMM to the CNCM is evaluated below. If the ISMM can serve as a method of approximating the more realistic CNCM, a relatively quick approximation can be ascertained for a system before formulating the more rigorous and precise optimization. Focus is simply on the power requirements because other values have little significance in this context outside of the actual system design. That is, there probably will not be a need to accurately approximate, for instance, the section size, if the card configuration data is not yet known.

The graphs of the power requirements for both the ISMM and the CNCM for the optimal mixed configurations have been given in Figs. 5.4 and 6.1, respectively. It is observed that the general shape of the graphs is the same, although the scaling is higher for the CNCM as would be expected. Fig. 6.27 presents the ratio of the CNCM power requirements to that of the ISMM.

Peaks in the ratio graph occur along FFT size discontinuities, as is present in other graphs of the same nature. In the ISMM, azimuth memory per processor is allowed to reach values that are physically unrealizable on the available daughtercards. Hence, the CNCM explicitly limits the memory per processor and suffers from this lack of optimization that is available to the ISMM.

Overall, the approximation is deemed to be good. Statistics of the ratio graph are as follows: a maximum value of 1.31, minimum value of 1.002, mean value of

1.11, and standard deviation of 0.07. It is not feasible to add a constant to the ISMM solution to obtain a better approximation because the minimum value is so close to unity. Because the ISMM solutions by definition never exceed the CNCM solutions, the ISMM can be used as a lower bound approximation to the realistic CNCM. If a lower bound is not necessary, but rather a more accurate mean value, the solution from the ISMM can be multiplied by the mean ratio value 1.11, keeping the standard deviation in mind.

6.4 Conclusions

The CNCM involves a more sophisticated optimization formulation than does the ISMM presented in Chapter V. With this sophistication comes increased computational intensity, one to two orders of magnitude greater than in the ISMM. Benefits of the CNCM include a high degree of fidelity to plausible system realizations and the provision of information lacking in the ISMM to completely specify a system configuration.

Although not investigated in this work, the CNCM can be applied to other optimization objectives such as velocity maximization and resolution minimization for fixed power. Application to other optimization objectives entails obvious modifications to the formulation, as in the ISMM, as well as consideration of special cases that affect the upper bound on or mean number of configuration combinations to be evaluated. This same principle equally applies for other

hardware-constrained configurations such as the single card type configuration investigated for the ISMM.

Comparison of the optimal with the nominal configuration in the CNCM presents additional problems. Unlike in the ISMM, the most simplistic approach to the nominal formulation results in an area in which no feasible solutions are possible. This fact alone is not of concern, but when it is taken into consideration that increasing velocity and thus the overall requirements of the system facilitate feasible solutions, the problem calls for more attention. The problem is discovered to result from a high memory requirement and low processor requirement, thus creating an unrealizably high memory per processor value. The problem is rectified by the introduction of additional processors, thus simultaneously reducing processor utilization and memory per processor.

The added complication of the nominal formulation, resulting in three optimization variables just as in the optimal formulation, begs the question whether there is any advantage in employing the nominal section size in the CNCM. The extreme values produced by the nominal configuration probably lead to a negative response to that question. However, the primary reason for including the nominal configuration in this chapter is not to provide an alternative and clearly inferior method of determining a system configuration, but to serve as a point of reference to the advantage of optimizing the section size, which is a fundamental basis of this work.

The utility of the ISMM solution as an approximation to power requirements as determined by the more realistic CNCM has been demonstrated. A mean ratio of 1.11 (CNCM power to ISMM power) and a standard deviation of 0.07 presents the ISMM as a good approximator to the CNCM.

Although not tested in this work, a possible improvement to the ISMM, if used to approximate, is to incorporate into the formulation the same optimization variable that was created for the nominal configuration in this chapter. If the required processors P_a was used as a lower bound to the number of actual processors P_{act} , where P_{act} is an additional optimization variable, the peaks in the ratio graph might be reduced. Although the addition of an optimization variable entails greater computational complexity, overall computation time should not be greatly affected relative to the computation time of the CNCM because of the lack of integer programming.

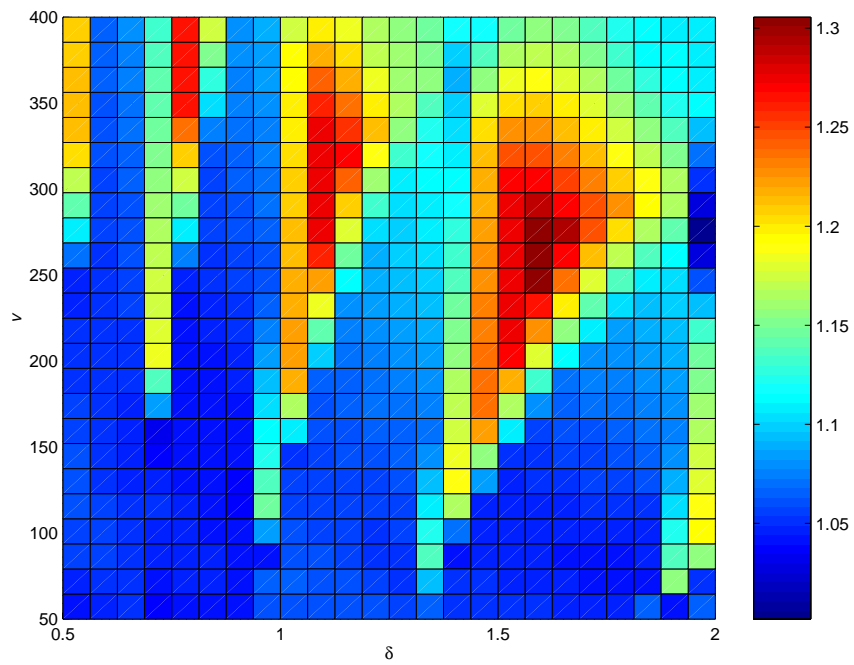


Fig. 6.27: Ratio of power requirements for the CNCM to the ISMM for the optimal mixed configuration.

CHAPTER VII

RANDOMLY-GENERATED SOLUTIONS

In this chapter the use of randomly-generated solutions is introduced to verify the results obtained in the previous two chapters. The method is applied only to the power minimization objective of the optimal mixed configuration of the ISMM, this case being the fundamental and most general configuration investigated. After application of this method of verification, the possibility of extending the use of this method as a primary means of optimization is also explored.

7.1 Solution Verification

Although it is impossible to completely verify the optimality of a given solution set without proving the convexity of the solution space, the random solutions provide a means to greatly increase the confidence invested in a solution. Random numbers are generated to form a large number of random solutions, the best of which is compared to the solution provided by the formal optimization formulation developed in Chapter V. If any random solution is superior to one obtained by the optimization formulation, implementation of the formulation cannot claim to produce truly optimal solutions. Consequently, in this case the formulation (inferring also the convexity of the problem) and/or MATLAB

routine is suspect.

To produce random solutions, several optimization variable values are randomly generated and values for the remaining variables solved based on the injected random values. Given a resolution, velocity, and proposed optimal power requirement, random values are generated for the section size S_a and number of S2T16Bs, C_1 .

The range for the section size is set at $[1, 2^{15} = 32768]$, encompassing values both much greater than and less than any found to be optimal in results collected. Such a great range increases the probability of poor solutions when a random value falls at one end of the range, but this statement presupposes the optimality of the proposed result. In random testing this assumption cannot be made. The FFT size is then calculated in the standard method based on the value of the generated S_a and K_a , which is dependent only on resolution.

C_1 is generated with a range that is dependent on the proposed optimal power. Although this dependency initially appears to corrupt the impartiality of the testing, it actually serves to ensure that every solution has at least a theoretical chance of being optimal. Let Π_{prop} be the proposed optimal power. Then an upper bound is imposed on C_1 equal to the maximum number of S2T16Bs supported by a power value of Π_{prop} . That is, if C_{max} designates this upper

bound on C_1 , then

$$\begin{aligned} C_{\max} &= \frac{\Pi_{\text{prop}}}{\Pi_d(\text{S2T16B})} \\ &= \frac{\Pi_{\text{prop}}}{12.2}, \end{aligned}$$

where Π_d is the power requirement per daughtercard expressed as a function of the daughtercard type. Although this restriction does not necessarily improve the mean solution value of the random test, it does insure that at the point of generation of C_1 , the solution is not already inferior to the proposed optimal solution.

With values for S_a and F_a , values for P_r , P_a , M_r and M_a can be calculated as usual based on Eqns. 3.2, 3.3, 3.5, and 3.6. C_2 is then calculated by heeding the more demanding of processor and memory requirements in terms of daughtercards. That is,

$$\begin{aligned} C_2 &= \max \left\{ \frac{(P_r + P_a) - \pi_d(\text{S2T16B})C_1}{\pi_d(\text{S1D64B})}, \frac{(M_r + M_a) - M_d(\text{S2T16B})C_1}{M_d(\text{S1D64B})} \right\} \\ &= \max \left\{ \frac{(P_r + P_a) - 6C_1}{2}, \frac{(M_r + M_a) - 32C_1}{64} \right\}, \end{aligned}$$

where π_d and M_d are the number of processors and amount of memory per daughtercard, respectively, as functions of the daughtercard type.

With the randomly generated and calculated values above, the power re-

quirement can be computed, as previously, as a function of C_1 and C_2 and their respective power requirements per daughtercard (Power = $C_1\Pi_d(\text{S2T16B}) + C_2\Pi_d(\text{S1D64B})$). It is clear that any given power calculation based on the random values has a very low probability of being even close to a proposed optimal solution. However, with a very large sampling of random solutions, the probability increases that if there exists a better solution than the proposed optimal power, then random testing will discover it.

In this work, MATLAB's **rand** function is used to generate random numbers. The function produces a vector of uniformly distributed floats in the interval $[2^{-53}, 1 - 2^{-53}]$, with a theoretical nonrepeating period of 2^{1492} .

The method described above was applied to the resolution and velocity values of 0.88 m and 297.9 m/s, respectively. The proposed optimal solution was 183.31 w. Table 7.1 displays the results of random testing taken over 1000000 samples. Fig. 7.1 shows the histogram of the tallied results using 100 bins. The spike at the extreme right of the graph represents the sum of the frequencies of values greater than can be displayed on the graph range.

The random tests lend a large degree of confidence to the proposed optimal solution. Considering Table 7.1, it is apparent that the best solution occurs when all parameter values are near the proposed optimal values. It is noted that random testing did not produce any solution better than the proposed solution, although it approached the proposed solution value and for all practical purposes

Table 7.1: Summary of random solutions.

Case	Power	% Inc.	F_a	K_a	S_a	C_1	C_2
Optimal	183.31	–	4096	1960	1491	5.43	12.19
Random Best	183.36	0.024	4096	1960	1455	5.63	11.94
Random Worst	94536	51471	2048	1960	1.0	0.477	9847
Random Average	839.3	357.9	–	–	–	–	–

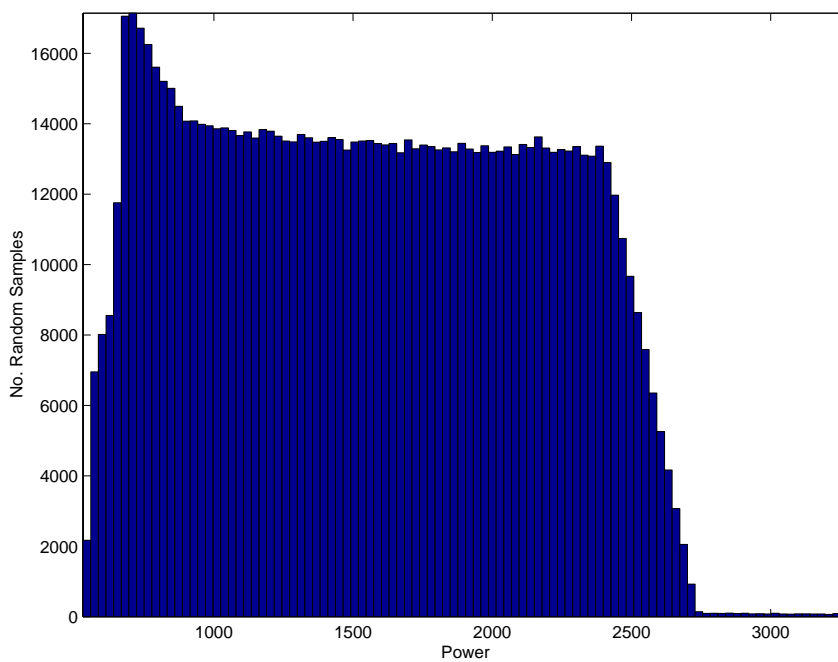


Fig. 7.1: Histogram of results from random testing on minimal power.

found a solution of equal quality, differing by only 0.024%. With one million random tests, confidence is lent to the validity of the testing method because a very good solution was indeed found.

Considering the worst case and mean of the random testing, the quality of both the proposed solution and that produced by random testing is placed in a certain perspective. Haphazard assignments of parameter values can be catastrophic.

Although the random testing has been conducted only on one set of resolution and velocity coordinates, the results are representative across the entire range of coordinates. This fact is demonstrated in the next section from a slightly different perspective.

7.2 Random Solutions as an Optimization Technique

In the previous section, random solutions are used to sample the solution space to evaluate a proposed optimal solution. This idea leads to the implementation of random sampling as a method of optimization in itself, without the premise of a proposed solution produced by a formal optimization routine.

The only necessary modification to the problem formulation as described in the previous section involves the absence of a proposed solution. Without this value to set the power range for C_1 , another value must be substituted in its place. If historical data is available to provide an upper bound, this data of

course could be utilized. However, in the absence of such data, a liberal upper bound for power must be injected into the problem. Even a poor estimate, as long as it is high enough, allows the possibility of finding the optimal solution. The greater the estimate is above the true optimal power, however, the greater the time the random sampling will take to converge to a solution approaching the optimal.

For each case investigated in this section, the power range for C_1 was set at $[0,1200]$ w. It is not completely fair to call 1200 w a “liberal” upper bound since knowledge of the solution surface presented in Fig. 5.4 suggests an upper power value of 868 w. Nevertheless, the mean power consumption of the graph is only 135 w. In practice, optimal values for power are sought for precise resolution and velocity coordinates more often than are power surfaces as has been usually presented in this work. Thus in the highest performance area, an upper bound of 1200 w (and it must be remembered that this upper bound is just for one card with no bounds on the other) is less than 50% higher than the optimal value, which might be considered a conservative and rather accurate ‘estimate’. However, for the majority of the resolution and velocity coordinates, 1200 w is a very poor and very liberal power estimate. Furthermore, an auxiliary goal of this exercise is to test all the points of the proposed optimal power surface, and too high a power range would reduce the probability of finding better solutions if they exist.

Random sampling of the solution space was done over all the coordinates evaluated by the optimization methods for power minimization as in the previous two chapters. The range of the random section size was handled slightly different than in the previous section. The range of S_a was modified as $[1, F_a - K_a]$, where K_a is dependent on the resolution and F_a is set at $2^{16} = 65536$. This modification increases the range of S_a slightly without increasing the upper bound of F_a , entailing more ‘intelligent’ S_a guesses at high values. Figs. 7.2–7.5 show the results of random sampling for sample sizes per coordinate pair of 100, 1000, 10000, and 100000 respectively. Tables 7.2 and 7.3 display the statistical results.

Table 7.3 shows a steady convergence of the random sampling solutions to the optimal solutions proposed by the formal optimization formulation. However, note that at no point does a random sampling solution discover a better solution than that of the formal optimization. This fact lends confidence to the absolute optimality of the solutions produced by the formulation presented in the ISMM and the associated convexity of the problem (at least local convexity in the range of resolution and velocity values investigated).

Evaluation of the random sampling method involves comparison with the formal optimization solution in terms of the accuracy of the results and calculation time. Using the formal optimization solution as a reference, it is seen that the accuracy of the results (see Table 7.3) depends on the number of samples

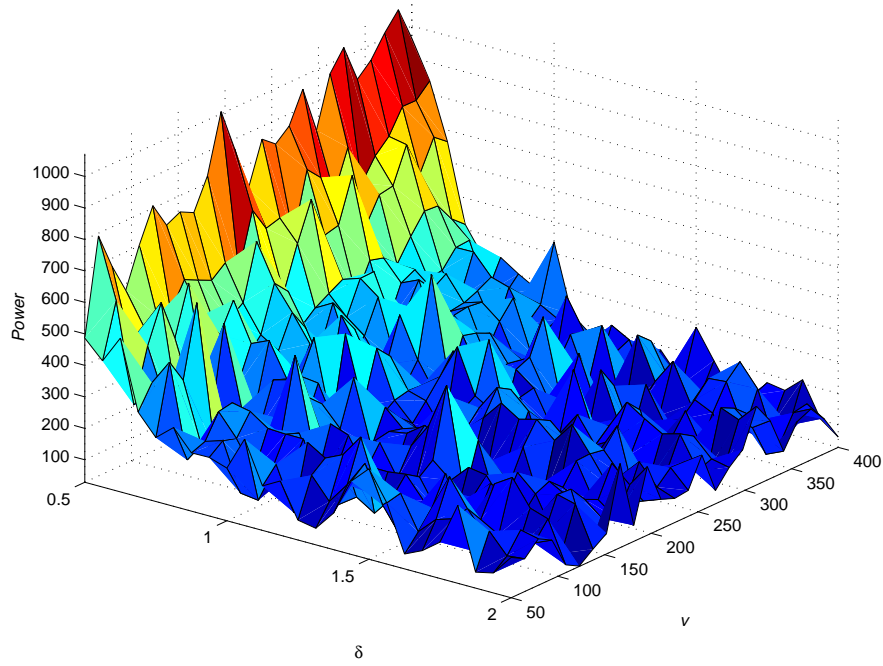


Fig. 7.2: Minimal power requirements by random sampling using sample size of 100.

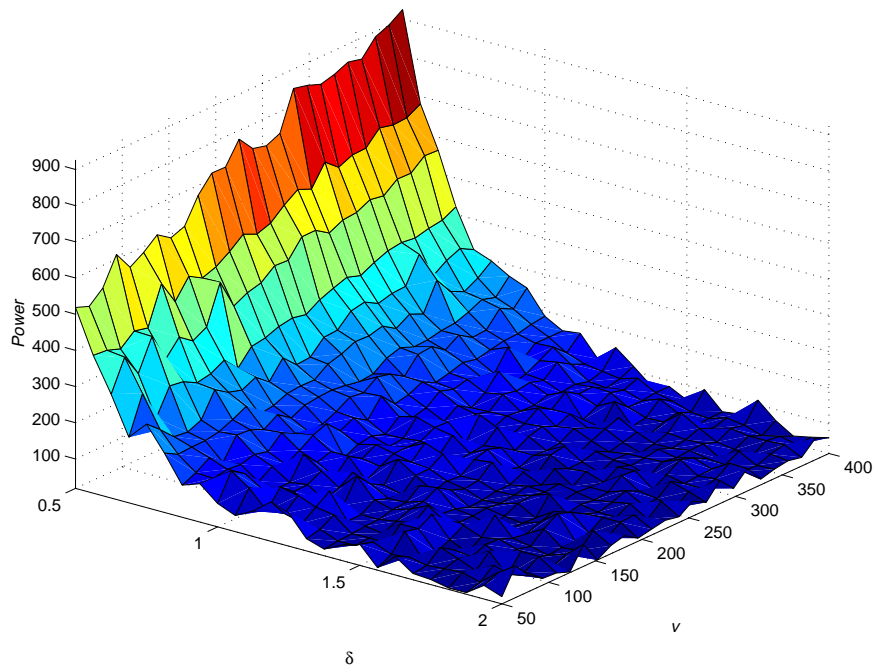


Fig. 7.3: Minimal power requirements by random sampling using sample size of 1000.

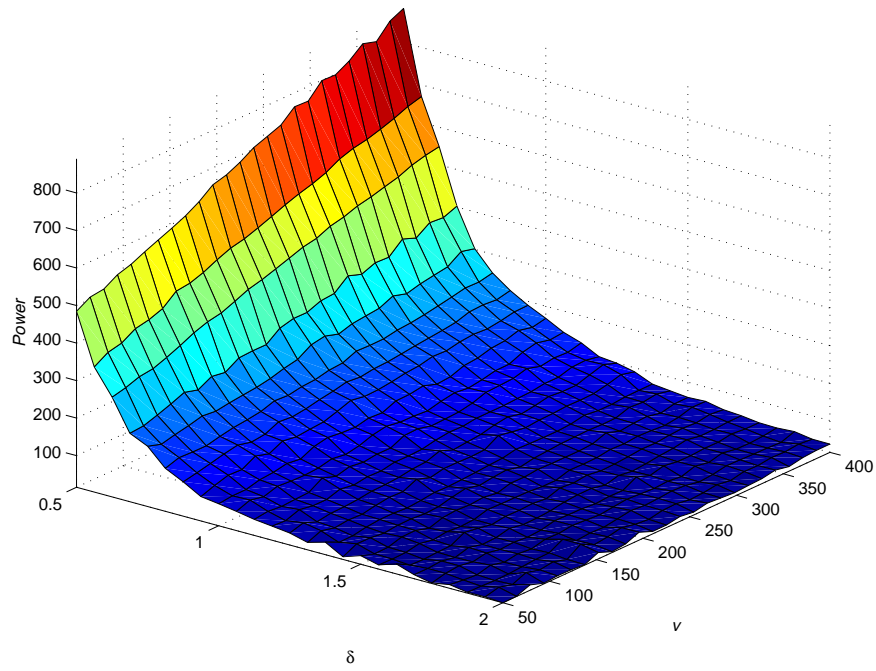


Fig. 7.4: Minimal power requirements by random sampling using sample size of 10000.

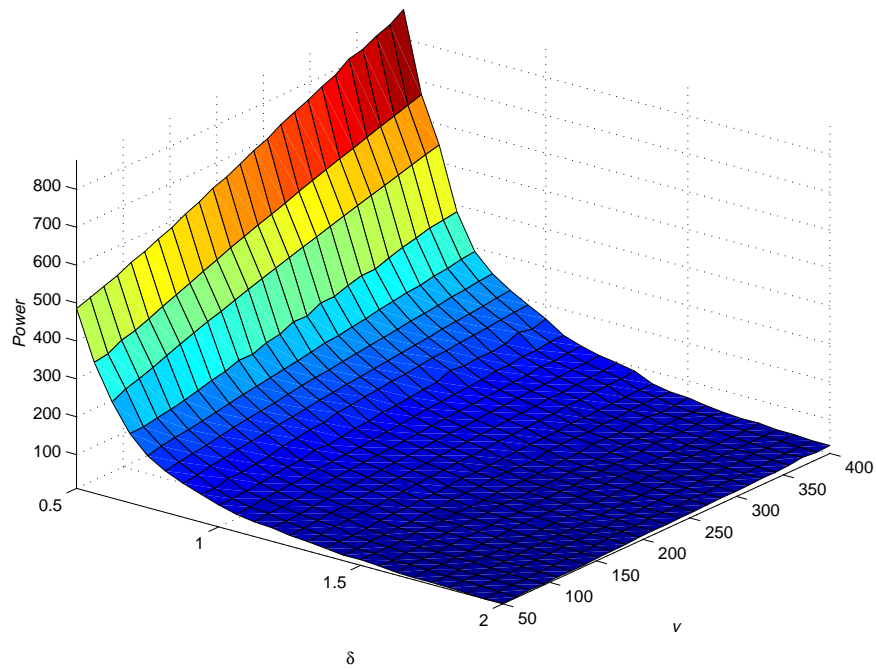


Fig. 7.5: Minimal power requirements by random sampling using sample size of 100000.

Table 7.2: Comparison of optimal and random configuration power consumption.

No. Samples	Minimum	Maximum	Mean
Optimal	9.2	867.6	135.5
100	28.6	1071.7	261.3
1000	18.5	926.1	166.7
10000	15.6	890.9	144.3
100000	10.8	877.4	137.9

Table 7.3: Ratios of random to optimal power.

No. Samples	Minimum	Maximum	Mean	Std. Dev.
100	1.0072	21.365	3.7844	3.1601
1000	1.0061	10.530	1.6493	0.8310
10000	1.0021	2.7504	1.1777	0.2203
100000	1.0010	1.3799	1.0477	0.0580

taken, although even at the 100000-sample level the accuracy is still wanting by approximately 5% and by 18% at the 10000-sample level.

Average calculation times for the random sampling method were 0.011, 0.080, 0.550, and 6.15 s per solution point for the 100-, 1000-, 10000-, and 100000-sample operations, respectively. Relative to the average 1.21 s per solution point for the formal optimization method, the times for the random sampling method are not impressive considering the quality of the solutions. The above times were all collected on the same computer, an Intel Pentium 133 MHz with 80 MB of memory running Microsoft Windows NT 4.0. There was sufficient memory to allow all computations using the 80 MB of main memory only.

It should be noted that the calculation times of the random sampling method are very consistent, since the same operation with a consistent number of iterations is performed for each resolution and velocity coordinate. This consistency is in contrast to the calculation times of the formal optimization method, which varies depending on the accuracy of the initial guess and other factors in the optimization routine such as step size, constraint and objective tolerance, etc. However, even very poor initial guesses are quickly overcome by the optimization routine and calculation times never vary by more than 300%.

To achieve comparable results with the random sampling method in regards to the formal optimization method, the number of samples would need to be increased beyond the 100000 samples appropriated in this investigation. As ob-

served in the first section of this chapter, the million-sample test provided a solution of essentially equivalent quality as the formal optimization, but this quality was accomplished with the aid of foreknowledge of the proposed optimal solution to set an upper bound on the power consumption by one card. Furthermore, the calculation time on the million-sample test was extremely high (approximately 20 minutes) in comparison to the other solutions, exaggerated by constraints in memory on the local machine performing the computation and resultant reliance on virtual memory.

Thus, in terms of time and accuracy, the random sampling method does not perform well in comparison to the formal optimization method. The random sampling method has the advantage, however, of simplicity in implementation. Even at small samples, the random sampling method produces a basic solution surface clearly similar to that of the formal optimization, so as to be useful for quick approximation.

Random sampling as a method to verify the results of a formal optimization is shown to be useful. Although conclusive verification is impossible, greater confidence in the solution is achieved with larger sample sizes and greater ranges in randomized variables.

CHAPTER VIII

CONCLUSIONS

This work focuses on modeling the processor-memory relationships of an embedded system for synthetic aperture radar (SAR) processing. The Mercury RACE multicomputer, built with commercial off-the-shelf (COTS) components, is the computing platform case study. Within the framework of the models developed in this work, optimization is performed on parameters such as the convolution section size and the choice and number of processor-memory hardware subunits (daughtercards) comprising the system.

Size, weight, and power (SWAP) constraints often motivate the maximization of performance density for a given SAR system, especially in the case of unmanned aerial vehicles (UAVs) or satellites, which often accommodate SAR systems. SAR in itself is an approach to densifying a radar system by substituting a large degree of data postprocessing for radar equipment with prohibitively high size, weight, and power characteristics. Minimization of power is the fundamental objective in this research, although with sufficient parameter guidelines size and weight could also be minimized using the same approach.

The specific mode of SAR investigated in this research is known as *stripmapping*. In stripmapping, successive radar pulses are transmitted and returned in the range dimension, which is orthogonal to the line of flight. Each received

series of pulses from an individual transmitted pulse is then convolved with a reference kernel to achieve range compression. The entire range is processed at once in this way.

To create a two-dimensional image, however, processing in the azimuth dimension is also necessary. The azimuth dimension is parallel to the line of flight and is conceptually infinite in length. Thus, processing of the entire azimuth vector, created from stacked range-processed vectors, is infeasible. To counter this problem, sectioned convolution is employed.

Sectioned convolution extracts a piece (or section) of the azimuth vector, convolves it with a reference kernel as in the range dimension, and then discards a section of the result equal to the length of the reference kernel. Successive processed azimuth sections are then overlapped (with overlaps equal to the discarded section length) to form continuous vectors in the azimuth dimension.

As is intuitive, a large azimuth section length requires more memory than a small section. Correspondingly, a small azimuth section requires more overall processing than does a large section because the percentage of new data processed that is not discarded is low, the size of the reference kernel being fixed.

One major focus of this work is the exploitation of the section size and the concomitant processor-memory tradeoff. Different daughtercards are better suited for different scenarios depending on the memory per processor ratio, the application requirement of which is largely dependent on the chosen section size.

The combination of the choices for the section size and number and types of daughtercards employed greatly affects the overall performance and associated power consumption of a system.

Two models are presented in this work that address the problem of determining the optimal values for these variables. Both methods are based on mathematical programming, which provides a method of formulating an optimization problem given an objective and set of constraints. All computation in the work was performed using MATLAB 5.1 and the associated Optimization Toolbox's **constr** function, which implements a Sequential Quadratic Programming algorithm to solve nonlinear constrained minimization problems.

The first model (introduced in [16]) is based on the assumption of an ideal shared memory system. It treats all the memory contributed by individual daughtercards as a conglomerate block, equally accessible by all processors located on all daughtercards. For the Mercury RACE system, this is an inaccurate oversimplification. However, it is useful to initially investigate the optimization of the SAR system based on such an assumption because it provides clear insight into the interrelationships between variables and the effects of perturbation of other external parameters. In addition, the simplification eases the collection of data because of the relatively low level of computational intensity.

The second model removes the assumption of shared memory and purposes to address system configuration more realistically. With this goal comes an

increase in the complexity of the optimization formulation. The constraint set is modified to ensure only local memory access by processors. To accomplish this optimization, a much higher degree of integer programming is required than in the first model, entailing higher computational intensity. The benefits of this second model include solutions that consist of a complete specification of system resources, whereas the first model only specifies which resources are to be employed.

Comparison of the two models shows the first model to be a good approximator to the second model. Furthermore, the first model in its own right is a valid representation of a system in which communication time between daughtercards is only negligibly higher than memory access time by processors to their own memory modules.

The utility of optimization of the section size is demonstrated by comparison of results produced by a heuristic used to determine section size. The heuristic defines the section size to be equal to the kernel size. This section size definition and resultant system configuration is designated as *nominal*. This work finds that the nominal section size, although relatively efficient in processing, is too large for most scenarios because of the excessive memory requirements involved. Research shows that forcing relatively inefficient processing with an associated reduction in memory requirements is optimal if power is to be minimized. Optimal section sizes thus often are found to be only a fraction of the kernel size,

entailing the processing of more old data that is to be discarded than new data.

This work also demonstrates the advantage of employing more than one type of daughtercard in a system. Different daughtercards are characterized by different power requirements, amounts of memory, number of processors, and resultant memory per processor ratios. Optimization exploits these differences and determines the optimal system configurations.

Although minimization of power is the primary objective in this work, other optimization objectives are also considered. Minimization of resolution and maximization of velocity with a fixed power or hardware configuration are also investigated.

Random sampling of the solution space is performed to verify the proposed solutions produced by the optimization formulations. Such testing demonstrated the first model to be trustworthy. Although random testing was not performed on the second model, the same principle may be applied with an associated more complex random testing formulation.

The concept of solution verification with random sampling leads to the proposal of random sampling as a method of optimization. The weakness of this method is in the quality of the solutions and computation time required. Except for very low sample sizes, the required calculation time was equal to or greater than the time required for the formal optimization algorithm. Furthermore, the quality of the solutions was uniformly worse than the formal optimization. The

strength of the random sampling method is in its simplicity of formulation. This method can be implemented without any knowledge of mathematical programming or availability to sophisticated mathematical optimization routines.

Generalization of the models developed in this work is straightforward. Especially in the second model, effort is made to avoid use of values specific to the Mercury RACE system. Instead, functions are defined that take as an argument the daughtercard type and return the number of processors, amount of memory, and power. Thus, any system that is constructed with the daughtercard concept, i.e., processor-memory nodes, can be modeled with minor modifications to the formulations presented.

REFERENCES

- [1] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, Second Edition, John Wiley & Sons, New York, NY, 1993.
- [2] M. A. Branch and A. Grace, *MATLAB: Optimization Toolbox User's Guide, Version 1.5*, The MathWorks, Inc., Natick, MA, 1996.
- [3] W. G. Carrara, R. S. Goodman, and R. M. Majewski, *Spotlight Synthetic Aperture Radar: Signal Processing Algorithms*, Artech House, Boston, MA, 1995.
- [4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1994.
- [5] J. C. Curlander and R. N. McDonough, *Synthetic Aperture Radar: Systems and Signal Processing*, John Wiley & Sons, New York, NY, 1991.
- [6] J. L. Eaves and E. K. Reedy, *Principles of Modern Radar*, Van Nostrand Reinhold Company, New York, NY, 1987.
- [7] T. Einstein, "Mercury Computer Systems' Modular Heterogeneous RACE Multicomputer," *Proceedings of the Sixth Heterogeneous Computing Workshop (HCW '97)*, sponsor: IEEE Computer Society, Geneva, Switzerland, April 1997, pp. 60-71.
- [8] T. Einstein, "Realtime Synthetic Aperture Radar Processing on the RACE Multicomputer," Application Note 203.0, Mercury Computing Systems, Inc., Chelmsford, MA, 1996.
- [9] J. Fitch, *Synthetic Aperture Radar*, Springer-Verlag, New York, NY, 1988.
- [10] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press, London, 1981.
- [11] M. Ginsberg, *Essentials of Artificial Intelligence*, Morgan Kaufmann Publishers, San Francisco, CA, 1993.
- [12] R. O. Harger, *Synthetic Aperture Radar Systems: Theory and Design*, Academic Press, New York, NY, 1970.

- [13] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, Sixth Edition, McGraw-Hill, New York, NY, 1995.
- [14] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.
- [15] B. C. Kuszmaul, "The RACE Network Architecture," *Proceedings of the 9th International Parallel Processing Symposium (IPPS '95)*, sponsor: IEEE Computer Society Technical Committee on Parallel Processing, Santa Barbara, CA, April 1995, pp. 508-513.
- [16] J.T. Muehring and J.K. Antonio, "Optimal Configuration of Parallel Embedded Systems for Synthetic Aperture Radar," *Proceedings of the 7th International Conference on Signal Processing & Applied Technology*, October 1996, pp. 1189-1194.
- [17] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
- [18] A. V. Oppenheim and R. W. Schaffer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.
- [19] *RACEway Interlink Modules*, VITA Standards Organization (VSO), 1994.
- [20] "SHARC DSP Compute Nodes (3.3-Volt)," Mercury Computing Systems, Inc., Chelmsford, MA, Sept. 1995.
- [21] M. I. Skolnik, *Introduction to Radar Systems*, McGraw-Hill, New York, NY, 1962.
- [22] M. I. Skolnik, *Radar Handbook*, Second Edition, McGraw-Hill, New York, NY, 1990.
- [23] J. S. Walker, *Fast Fourier Transforms*, Second Edition, CRC Press, Boca Raton, FL, 1996.
- [24] J. West, *Simulation of the Communication Time for a Space-Time Adaptive Processing Algorithm on a Parallel Embedded System*, M.S. Thesis, Texas Tech University, 1998.

APPENDIX

This appendix includes a representative portion of the MATLAB code that was significant in the generation and analysis of the data presented in this work. Note that the actual data was produced by many separate application files, often duplicating common portions of code. This approach was taken in favor of creating one monolithic application that required numerous input arguments or prompts and the associated internal conditional statements. Such an approach inevitably would have resulted in a more cumbersome program, in terms of both execution and maintainability. The code included in the following pages often represents only code fragments, not complete “.m” files.

Parameter Initialization

%This complete file serves as a general template for
%power minimization in both the ISMM and CNCM. In addition,
%the capability to determine integer daughtercard numbers is
%included. Execution of the complete file therefore produces
%solutions to the optimal power, mixed card-type configuration
%of the CNCM with integer card numbers. This configuration
%can be treated as the most general case with the real-value
%card number case taken as a specific instance of this program
%with the integer card-number code removed. Furthermore, the
%ISMM can similarly be treated as a specific case of the CNCM,
%where the code concerning CNS is removed.

%This code also serves as the template for the other objective
%functions, although modifications are necessary in the order
%of variable calculations (for example, in resolution
%minimization the azimuth kernel size cannot be calculated
%until a resolution is known) and to the number of parameters
%passed. This latter modification is necessary also when
%switching between the ISMM and CNCM. (Note that in the case
%of the other objective functions, changing of several
%variable names would be in order.)

%Because of the occasional lockup of the optimization routine,
%it is beneficial to allow other starting indices besides 1.
%The user can obtain the index from the last output. In
%addition, certain spots in the surface can be recalculated by
%entering the appropriate index and then aborting the program
%when the desired sequence is complete. When initiating a
%fresh run, it is advised that RESULTS are set to [] before
%invocation of the program. Otherwise they are never cleared
%in order to allow for midway restarts. The indices range
%from 1 to the total number of samples. For several reasons,
%this method proved easier than dealing with both row and
%column indices.

```
index = input('Enter starting index: ');
```

%Global variables are used to transfer data to the CONSTR
%optimization routine. Although passing of non-optimization

```

%variables is provided for in CONSTR, the large number of
%variables necessary lent global variables to be a much more
%efficient option, without causing unnecessary problems in
%program flow or maintainability. Variables are defined as
%they are introduced below.
clear delta v C1 C2 CNMem CNPow Fa Ka Mr Pr Ma Pa
clear R Rs c T alpha beta gamma lambda Meg

global delta v C1 C2 CNMem CNPow Fa Ka Mr Pr Ma Pa
global R Rs c T alpha beta gamma lambda Meg

%Parameter initializations
Meg = 10^6;           %1 million
c = 3*10^8;          %speed of light (m/s)
T = 2048*2/c;        %pulse width (s)
R = 100000;          %range (m)
Rs = 20000;          %range swath (m)
lambda = .03;        %wavelength (m)
alpha = 127/360;     %range non-fast-convolution load (MFlops)
beta = 1061/1170;    %azimuth non-fast-convolution load (MFlops)
gamma = 94;          %fast-convolution load (MFlops)

CNs = [2 1];         %CNs per card type
CNPow = [6.1 9.6];   %power per CN by card type (w)
CNMem = [16 64];     %memory per CN by card type (MB)
CNCE = [3 2];        %processors per CN by card type

lb = [1,0,0];        %lower bound for optimization parameters
options = [];         %clear options for optimization functions

x = [1500,0,0];      %Initial Guess

%convert index to row and column loop counters
i = ceil(index/25);
j = rem(index-1,25)+1;

vIter1 = 1;          %inner loop flag

%resolution sample-space vector
dvector = linspace(.5,2,25);
%velocity sample-space vector

```

```

vvector = linspace(50,400,25);

%outer loop for resolution
for delta=dvector(i:25)
    %check flag for first iteration to use INDEX value
    if ~vIter1
        %set start point of inner loop at 1 after first iteration
        j = 1;
        %clear flag
        vIter = 0;
    end
    %inner loop for velocity
    for v=vvector(j:25)

        FirstIter=1;          %flag for setting first solution value
        Reduced = 0;          %statistics of rounding integer cards
        NonReduced = 0;

        %range values can be calculated statically for each
        %new resolution-velocity pair
        %range FFT size
        Fr = 2ceil(log2(Rs/delta+c*T/(2*delta)));
        %number of range processors
        Pr = v*(alpha*Rs*gamma+10*Fr*delta*log2(Fr)...
            +6*Fr*delta)/delta2/Meg/gamma;
        %megabytes of range memory
        Mr = 16*v*Rs*(alpha*Rs*gamma+10*Fr*delta*log2(Fr)...
            +6*Fr*delta)/delta3/Meg2/gamma;

        %Although not very flexible, the execution of the
        %following two loops enumerates all the feasible
        %combinations of CN configurations in the CNCM.
        %A more versatile and generalized approach could be
        %formulated by following the derivation of the chapter
        %on the CNCM, using recursive functions or hardcoding
        %the number of loops and employing several conditionals.
        %However, for the task at hand of generating initial
        %test data, the brute force method below was efficient
        %and convenient.
        %C1 and C2 represent X and Y (as in the CNCM chapter),
        %or the first and second card configurations. Note that

```

```

%these two loops would be removed in the case of the
%ISMM.
for C1=[1,1,0; 1,2,0; 1,3,0; 2,1,0; 2,2,0; ...
        2,1,1; 1,1,1; 1,2,1; 1,1,2]'
    for C2=[1,0,1; 1,0,2; 1,0,3; 2,0,1; 2,0,2; ...
            2,1,1; 1,1,1; 1,2,1; 1,1,2]'

        %Do not test for two simultaneous heterogeneous
        %configurations. Only one is necessary.
        if ~( C1(2) & C1(3) & C2(2) & C2(3) )

            %Azimuth kernel size
            Ka = ceil(R*lambda/(2*delta^2));
            %Power of two for azimuth FFT size (Fa)
            FFTk = ceil(log2(Ka+1));

            %flag for yet-active section size constraint
            ConActive=1;

            %while section size constraint still active
            while ConActive

                %compute (next) value of Fa
                Fa = 2^FFTk;

                %x: vector of optimization variables
                %options: optimization toolbox options
                %OptFun: user-defined .m file with
                %    objective function and constraints
                [x,options]=constr('OptFun',x,options,lb);

                %get objective function value and constraints
                %values to check validity
                [f,g] = OptFun(x);

                %Check if all constraints are less than the
                %tolerance (default). CONSTR will display a
                %warning if no feasible solution is found but
                %will still return a rational value for the
                %objective function even if infeasible
                if g <= options(4)

```

```

    %Feasible flag set
    Feasible = 1;
else
    %Feasible flag cleared
    Feasible = 0;
    %objective function value set to infinity
    %for comparison's sake in subsequent
    %iterations
    options(8) = inf;
end %end if-else

%if solution is not feasible, continue
%Note that this section is only when integer
%card numbers are desired. This long
%conditional statement can be skipped if
%real-valued solutions are sought.
if Feasible

    %integer card values, to be determined
    Real = [0 0 0 inf];
    %original values returned by optimization
    Ideal = [x options(8)];

    %The below code is hardcoded for the two
    %daughtercards researched. For a general
    %formulation, the variable CNS should
    %be used.
    %Use modulus to round up to nearest card,
    %dependent on number of CNS per card.
    if C1(1) == 1      %S2T16B
        ceilx(1) = ceil(x(2))...
            + mod(ceil(x(2)),2);
        floorx(1) = floor(x(2))...
            - mod(floor(x(2)),2);
    else                %S1D64B
        ceilx(1) = ceil(x(2));
        floorx(1) = floor(x(2));
    end %end if-else

    if C2(1) == 1      %S2T16B
        ceilx(2) = ceil(x(3))...

```

```

        + mod(ceil(x(3)),2);
    floorx(2) = floor(x(3))...
        - mod(floor(x(3)),2);
else          %S1D64B
    ceilx(2) = ceil(x(3));
    floorx(2) = floor(x(3));
end %end if-else

```

```

%Check mixed card usage. Otherwise any
%rounding down is infeasible. If mixed,
%check for possible use of floor of one
%card type. Floor of both types is
%always infeasible.
if ceilx

```

```

    %Check for ceiling:floor of C1:C2 if
    %power of such a mix is >= to optimal
    %value. If not, configuration is
    %infeasible by definition of minimum
    %power.
    if( CNPow(C1(1))*ceilx(1) + ...
        CNPow(C2(1))*floorx(2) ...
        >= Ideal(4) )
        %restrict optimization to within
        %new ceiling:floor bounds by setting
        %upper bounds.
        ub=[inf,ceilx(1),floorx(2)];
        %reoptimize with new upper bounds
        [x,options]=constr('OptFun',...
            Ideal(1:3),options,lb,ub);
        %get constraints values
        [f,g] = CNHetFun(x);
        %check for validity of solution
        if all(g <= options(4))...
            & all(x <= (ub + options(4)))
            %set new card and power values
            Real = [round(x(1)) ceilx(1)...
                floorx(2)...
                CNPow(C1(1))*ceilx(1)...
                + CNPow(C2(1))*floorx(2)];
        end %if

```

```

end %if

%Check as above but floor:ceiling for
%C1:C2 instead.
if (CNPow(C1(1))*floorx(1)...
    + CNPow(C2(1))*ceilx(2)...
    >= Ideal(4) )
ub=[inf,floorx(1),ceilx(2)];
[x,options]=constr('OptFun',...
    Ideal(1:3),options,lb,ub);
[f,g] = CNHetFun(x);
if all(g <= options(4))...
    & all(x <= (ub + options(4)))
    TmpPow = CNPow(C1(1))*floorx(1)...
        + CNPow(C2(1))*ceilx(2);
    %compare present value to value
    %(if any) of ceiling:floor
    %solution and take best
    if TmpPow < Real(4)
        Real = [round(x(1)) floorx(1)...
            ceilx(2) TmpPow];
    end %if
    end %if
end %if
end %if

%check for any valid answer, else...
if Real(4) == inf
    Real = [round(x(1)) ceilx(1) ceilx(2) ...
        CNPow(C1(1))*ceilx(1)...
        + CNPow(C2(1))*ceilx(2)];
    %increment tally of ceiling:ceiling
    %solutions
    NonReduced = NonReduced + 1;
else
    %increment tally of either ceiling:floor
    %or floor:ceiling solutions
    Reduced = Reduced + 1;
end %if-else
end %if

```



```

%check if answer is feasible and is best so far
if Feasible & ( FirstIter...
    | ( Real(4) <= Best(8) ) )
    %new best solution
    Best = [delta v Fa Ka Real C1' C2'];
    %clear flag
    FirstIter = 0;
end %if

%Check for active section size constraint.
%If active, increase FFT size and try again.
%Otherwise, discontinue.
if ceil(x(1)+Ka)<Fa
    ConActive=0;
else
    FFTk=FFTk+1;
end %if-else

end %while

end %if

end %for C2
end %for C1

%RESULTS holds the best for each velocity-resolution pair.
%If first iteration flag still set, then no valid solution
%was found during all iterations. Set flags of 0 or
%infinity in values to mark infeasibility.
if FirstIter
    results(index,:) = [delta v Fa Ka 0 0 0 inf...
        0 0 0 0 0 0 0];
else
    %REDUCED and NONREDUCED hold tallies for one entire
    %resolution-velocity pair
    results(index,:) = [Best Reduced NonReduced];
end %if-else

%Print to screen results. Note that the results could
%also be printed to a file instead by including a file

```

```

%handle instead of the '1', opened at the start of the
%program.
fprintf(1, '%d\t%.2f\t%.1f\t%d\t%d\t%4d\t%.1f\t%.1f...
\t%.1f\t%d%d%d\t%d%d%d\t%d/%d\n', ...
index, Best, Reduced, NonReduced);

%Increment RESULTS array index by one.
index = index + 1;

end %for v
end %for delta

%clear global variables
clear global

```

Function File Preliminary Code

%The code fragments below are representative of the function files associated with each optimization objective and configuration. The preliminary common to each function is given below, with the unique blocks of code displayed below. Although in reality each block would be a uniquely named file, all the files are grouped here under the name %OptFun. This file is the one referenced in the previous example of the main program. Note that modification is necessary in the main program to accommodate the variously sized optimization variables vector and removal of range variable calculations where appropriate.

%Common preliminary code

%X: vector of optimization variables
%f: objective function value
%g: vector of constraints values
function [f,g] = OptFun(x)

%Global parameters

global delta v C1 C2 Fa Ka Mr Pr Ma Pa
global R Rs c T alpha beta gamma lambda Meg

ISMM Optimal Single-Card Power

%Single card-type function

%Note that there are only two optimization variables

Sa = x(1);

C1 = x(2);

Pa = v*Rs/delta^2/Meg*(beta+(10*Fa*log2(Fa)+6*Fa)/Sa/gamma);

```
Ma = 12*Rs/delta*(Sa+1/2*R*lambda/delta^2)/Meg;
```

```
%Objective Function
```

```
f = 12.2*C1;
```

```
%Constraints
```

```
g(1) = (Pr + Pa) - 6*C1;
```

```
g(2) = (Mr + Ma) - 32*C1;
```

```
g(3) = Sa - (Fa-Ka);
```

ISMM Nominal Mixed Power

```
Sa = Ka;
```

```
C1 = x(1);
```

```
C2 = x(2);
```

```
%Objective Function
```

```
f = 12.2*C1 + 9.6*C2;
```

```
%Constraints
```

```
g(1) = (Pr + Pa) - (6*C1 + 2*C2);
```

```
g(2) = (Mr + Ma) - (32*C1 + 64*C2);
```

```
g(3) = Sa - (Fa-Ka);
```

ISMM Optimal Mixed Velocity

```
v = x(1);
```

```
Sa = x(2);
```

```
C1 = x(3);
```

```

C2 = x(2);

Fr = 2^ceil(log2(Rs/delta+c*T/(2*delta)));
Pr = v*(alpha*Rs*gamma+10*Fr*delta*log2(Fr)+6*Fr*delta)/delta^2/Meg/gamma;
Mr = 16*v*Rs*(alpha*Rs*gamma+10*Fr*delta*log2(Fr)+6*Fr*delta)/delta^3/Meg^2/gamma;

Pa = v*Rs/delta^2/Meg*(beta+(10*Fa*log2(Fa)+6*Fa)/Sa/gamma);
Ma = 12*Rs/delta*(Sa+1/2*R*lambda/delta^2)/Meg;

%Objective Function
f = -v;

%Constraints
g(1) = 12.2*C1 + 9.6*C2 - P;
g(2) = Sa - (Fa-Ka);
g(3) = (Pr + Pa) - (6*C1 + 2*C2);
g(4) = (Mr + Ma) - (32*C1 + 64*C2);

```

ISMM Optimal Mixed Velocity with Set Hardware

```

v = x(1);
Sa = x(2);

Fr = 2^ceil(log2(Rs/delta+c*T/(2*delta)));
Pr = v*(alpha*Rs*gamma+10*Fr*delta*log2(Fr)+6*Fr*delta)/delta^2/Meg/gamma;
Mr = 16*v*Rs*(alpha*Rs*gamma+10*Fr*delta*log2(Fr)+6*Fr*delta)/delta^3/Meg^2/gamma;

Pa = v*Rs/delta^2/Meg*(beta+(10*Fa*log2(Fa)+6*Fa)/Sa/gamma);
Ma = 12*Rs/delta*(Sa+1/2*R*lambda/delta^2)/Meg;

%Objective Function
f = -v;

```

```

%Constraints
g(1) = Sa - (Fa-Ka);
g(2) = (Pr + Pa) - (6*C1 + 2*C2);
g(3) = (Mr + Ma) - (32*C1 + 64*C2);

```

ISMM Optimal Mixed Resolution

```

delta = x(1);
Sa = x(2);
C1 = x(3);
C2 = x(4);

Ka=R*lambda/(2*delta^2);

Fr = 2^ceil(log2(Rs/delta+c*T/(2*delta)));
Pr = v*(alpha*Rs*gamma+12*Fr*delta*log2(Fr)+6*Fr*delta)/delta^2/Meg/gamma;
Mr = 16*v*Rs*(alpha*Rs*gamma+12*Fr*delta*log2(Fr)+6*Fr*delta)/delta^3/Meg^2/gamma;

Pa = v*Rs*(beta*Sa*gamma+12*Fa*log2(Fa)+6*Fa)/delta^2/Meg/Sa/gamma;
Ma = 6*Rs*(2*Sa*delta^2+R*lambda)/delta^3/Meg;

```

```

%Objective Function
f = delta;

```

```

%Constraints
g(1) = 12.2*C1 + 9.6*C2 - P;
g(2) = Sa - (Fa-Ka);
g(3) = (Pr + Pa) - (6*C1 + 2*C2);
g(4) = (Mr + Ma) - (32*C1 + 64*C2);

```

CNCM Optimal Mixed Power

```
Sa = x(1);
NumCN1 = x(2);
NumCN2 = x(3);
PaReal = x(4);

Pa = v*Rs/delta^2/Meg*(beta+(10*Fa*log2(Fa)+6*Fa)/Sa/gamma);
Ma = 6*Rs*(2*Sa*delta^2+R*lambda)/delta^3/Meg;

%Objective Function
f = CNPow(C1(1))*NumCN1 + CNPow(C2(1))*NumCN2;

%Constraints
g(1) = Sa - (Fa-Ka);
g(2) = Pr - NumCN1*C1(2) - NumCN2*C2(2);
g(3) = PaReal - NumCN1*C1(3) - NumCN2*C2(3);
g(4) = C1(2)*Mr/Pr + C1(3)*Ma/PaReal - CNMem(C1(1));
g(5) = C2(2)*Mr/Pr + C2(3)*Ma/PaReal - CNMem(C2(1));
g(6) = Pa - PaReal;
```

CNCM Naive Nominal Mixed Power

```
NumCN1 = x(1);
NumCN2 = x(2);

%Objective Function
f = CNPow(C1(1))*NumCN1 + CNPow(C2(1))*NumCN2;

%Constraints
g(1) = Pr - NumCN1*C1(2) - NumCN2*C2(2);
g(2) = Pa - NumCN1*C1(3) - NumCN2*C2(3);
g(3) = C1(2)*Mr/Pr + C1(3)*Ma/Pa - CNMem(C1(1));
```

$$g(4) = C2(2)*Mr/Pr + C2(3)*Ma/Pa - CNMem(C2(1));$$

CNCM Sophisticated Nominal Power

$$\text{NumCN1} = x(1);$$

$$\text{NumCN2} = x(2);$$

$$\text{Pa} = x(3);$$

%Objective Function

$$f = \text{CNPow}(C1(1))*\text{NumCN1} + \text{CNPow}(C2(1))*\text{NumCN2};$$

%Constraints

$$g(1) = Pr - \text{NumCN1}*C1(2) - \text{NumCN2}*C2(2);$$

$$g(2) = Pa - \text{NumCN1}*C1(3) - \text{NumCN2}*C2(3);$$

$$g(3) = C1(2)*Mr/Pr + C1(3)*Ma/Pa - \text{CNMem}(C1(1));$$

$$g(4) = C2(2)*Mr/Pr + C2(3)*Ma/Pa - \text{CNMem}(C2(1));$$