CHAPTER 13

The Radar Signal Processor

Mark A. Richards

Chapter Outline

13.1 INTRODUCTION

The signal processor is the portion of the radar system responsible for extracting actionable information about the signal (target, clutter, and jamming) environment from raw radar signals. The signal processor is composed of two major elements: (1) the algorithms that analyze the radar data; and (2) the hardware on which those algorithms are hosted.

Historically, signal processing hardware has been constructed using a variety of analog and digital technologies. For instance, two-pulse moving target indication (MTI) has been implemented in older systems using analog mercury delay lines, and mixed analog and digital surface acoustic wave (SAW) devices have been used for pulse compression [1]. Early synthetic aperture radar (SAR) systems used remarkably elegant optical processors for image formation [2]. Many excellent radar systems have been fielded using analog signal processing techniques. The principal advantage of analog signal processing, whether performed in electronic circuits, microwave circuits, or even optical devices, is speed. Functions are computed at the speed of signal propagation, literally the speed of light or nearly so in most cases. However, analog techniques suffer from limited dynamic range, temperature sensitivity and aging, lack of flexibility, limited memory, and other disadvantages.

To counter these problems, digital signal processing (DSP) began to be applied to some radar functions in the 1960s. By the 1970s and 1980s the digital revolution enabled by Moore's Law (see Chapter 14) had progressed to the point where many radar signal

processing operations could be implemented digitally in real time for a range of important systems. Today, radar designers can capitalize on another 20 years (10+ generations of Moore's Law improvements in digital technology) to implement signal processing techniques far beyond what can currently be done with analog circuitry.

DSP offers designers several important benefits. Digital processing results are more repeatable than those obtained with analog circuits. Both the accuracy and dynamic range are controllable through the choice of digital word lengths. It is often easier in digital processors to implement controllable or adaptive parameters, making these implementations more flexible than their analog counterparts. Digital processors are inherently more compatible with other digital portions of the radar system such as displays, data links, and the data processor.

By far the most important benefit of digital processing is a direct result of it being essentially programmable. Digital processors can implement a vastly greater range of functions than can analog systems, limited only by the imagination of the algorithm developers and the real-time constraints of the application. While very high-quality analog MTI filters and linear FM pulse compression filters have been fielded, digital techniques enable the development of much more advanced processing methods. Examples include adaptive MTI filters, waveform generators and matched filters for nonlinear frequency modulations, modern spectral analysis techniques, and elaborate space-time adaptive processors. These techniques are difficult and, in many cases, effectively impossible to implement in practical analog hardware. Thus, digital processing has made it possible to undertake more advanced algorithms than is feasible with only analog processing. This increased breadth of signal processing functions, and the increased radar performance it enables, is the deciding advantage for digital signal processing, and virtually all modern radar systems use digital processing.

In this chapter, common approaches to the architecture of radar signal processors are described, along with metrics for evaluating those architectures. The role of both the hardware and the software in determining the effectiveness of a processor is described.

13.2 RADAR PROCESSOR STRUCTURE

Figure 13-1 illustrates, at a very high level, the general structure of radar signal processing algorithms from the point of view of the computational resources required. Raw radar data from the receiver arrive at the input to the signal processor. These data are sampled at rates determined by the signal bandwidth; these rates can range from several hundred thousand samples per second to several gigasamples per second. Most modern radar systems use coherent receivers, so the input data will be complex-valued. Generally, the data will be at baseband, but in systems using digital intermediate frequency (*digital IF*) or digital in-phase/quadrature (*digital I/Q*) techniques it may be on a relatively low intermediate frequency carrier.

The early stages of a radar signal processing algorithm generally involve fixed functions that are applied to all of the data and that are not varied based on the actual content of the data. This computing style is often referred to as *streaming* processing. Examples include matched filtering/pulse compression, MTI and pulse Doppler filtering, coherent and noncoherent integration, SAR image formation, space-time adaptive processing (STAP), and constant false alarm rate (CFAR) threshold detection. The purpose of such processing is generally to improve the signal-to-interference ratio. The operations required

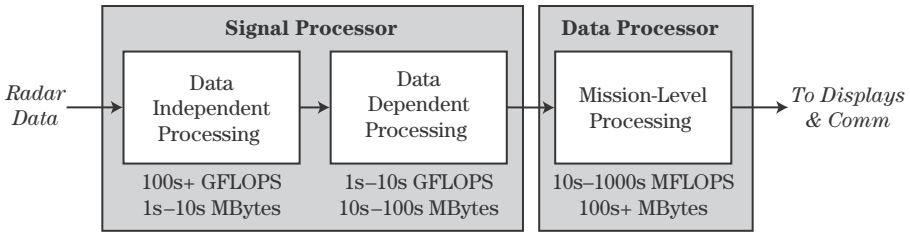

FIGURE 13-1 ■ Computational types, rates, and storage in a generic radar signal and data processor.

are primarily finite impulse response (FIR) digital filters, correlation, fast Fourier transforms (FFTs), and matrix-vector algebraic operations. Because of the high data rates and the relatively high computational complexity of the basic functions, this data-independent processing stage can require (in 2009) computational rates on the order of 1 billion to 1,000 billion operations per second (1 gigaops [GOPS] to 1 teraops [TOPS]). Because these arithmetic rates are so high, some systems use fixed-point arithmetic in these early operations due to its higher speed for a given size, weight, and space relative to a floating-point processor. Systems at the low end of the computational requirement, or those having larger-capacity signal processors, may use floating-point operations for greater arithmetic precision and ease of development. The computational rate is then measured in billions or trillions of floating-point operations per second (GFLOPS or TFLOPS).

While the signal processor stages nearest the receiver may have high computational rate requirements, their storage requirements tend to be modest, on the order of ones to tens of megabytes. This is because of the streaming nature of the calculations, which take a block or datacube of data, apply a filter or transform to it, and pass the modified data to the next operation. Intermediate results are not retained, so memory requirements at any one stage of the process are moderate.

After the initial signal conditioning operations, the data are next passed to a series of operations that do depend on the actual content of the data. Examples include single-dwell track measurements and SAR autofocus. For instance, the number of track measurements required depends on the number of targets detected by the CFAR processing. This may be only one or a few in air-to-air radars but may be thousands in some air-to-ground radars. Traditionally, the computational rates are reduced at this stage, but memory requirements may be increased.

The outputs from this data-dependent processing, such as detections ("hits") with associated position and SNR estimates, are then passed to higher-level processing that typically operates on time scales corresponding to multiple dwells. A typical example is track filtering with α-β or Kalman filters, which operates over many dwells. Processing at this level is often called *data processing*, although some systems include it in the signal processing. The computational rates at this stage may be still lower, in the tens of MFLOPS to ones of GFLOPS range, but the storage requirements may increase to hundreds of megabytes or more.

Emerging system designs in 2009 incorporate increasingly elaborate postdetection processing, for instance, for tracking, classification, and recognition of thousands of potential targets. In some newer proposed systems the computational load at the back end of the processor may actually exceed that of the front end.

Nonetheless, the typical radar signal processor reduces the computational rate but increases the storage requirements in each successive stage. The implementation technology for digital radar signal processors mimics this flow, as shown in Figure 13-2. The early,

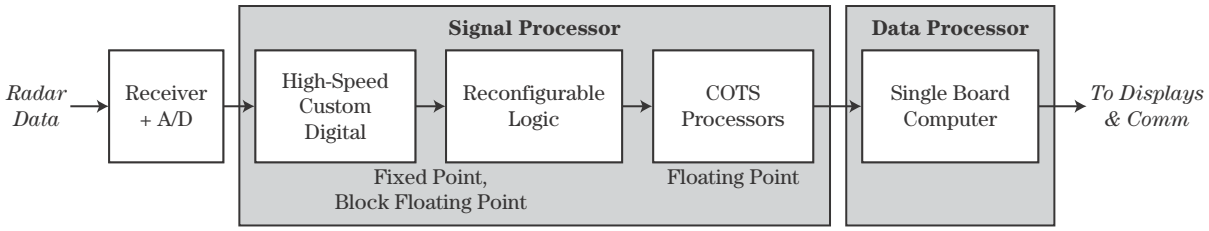

FIGURE 13-2 ■ Signal processor implementation technologies.

highest-speed data-independent operations may require the use of custom *application-specific integrated circuits* (ASICs) to obtain the required processing rates under severe size, weight, and power constraints. In many applications, this hardware may use fixed-point or block floating-point arithmetic; the latter is a compromise between the logic complexity and arithmetic precision of fixed-point and full floating-point arithmetic. The data-dependent operations can often be hosted on specialized *commercial off-the-shelf* (COTS) programmable processors designed for high-speed signal and data processing. Such processors typically use floating-point arithmetic. Finally, the relatively low computational rates, high memory requirements, and control complexity of mission-level processing can be accommodated with *single board computers* (SBCs), which are essentially equivalent to standard workstations. The various types of digital computing technology are discussed further in section 13.5.

13.3 SIGNAL PROCESSOR METRICS

The fundamental metric in evaluating a signal processor implementation is *time to solution*, which is the time required to complete all specified processing once the required data become available. In real-time radar signal processors, processing of one batch of data must be finished in time to accept the next batch of data, a so-called hard real-time requirement. If a processor does not have a hard real-time requirement, the designer may have additional flexibility to trade performance versus processor size or cost.

The time to solution depends on both the processor software and hardware. The hardware speed is determined by the type of digital processing technology being used, the amount of it available, and the speed at which it is operated. Software influences include the particular algorithms used to implement signal processing operations and the software tools such as languages, compilers, and libraries used to map those algorithms to the hardware.

13.3.1 Hardware Metrics

Signal processor metrics fall into two classes: (1) those describing the algorithms; and (2) those describing the implementation hardware. Consider hardware metrics first. The two most important are *throughput* and *latency*.

Throughput describes the rate at which a processor performs arithmetic operations. It is measured in floating-point operations per second; at radar rates, units of MFLOPS and GFLOPS are most common. If a processor uses fixed-point arithmetic, units of operations per second (MOPS, GOPS) are used.

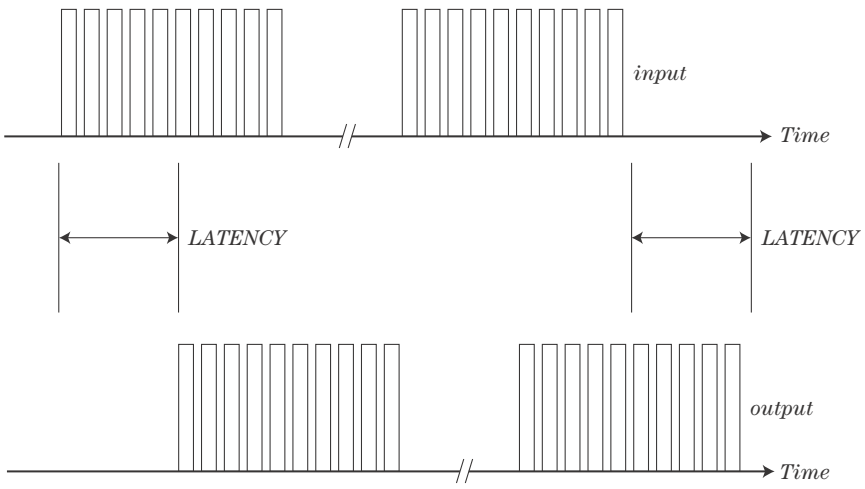

FIGURE 13-3 ■ Illustration of latency.

Another measure of processor speed is millions of instructions per second (MIPS). While MIPS is appropriate for assessing processor speed on general operations, it is not appropriate for quantifying floating-point arithmetic operations. Although some processors are designed to complete multiple floating-point additions or multiplications in a single cycle, others require many cycles, so MIPS are not a reliable indicator of floating-point arithmetic speed. However, MIPS may be appropriate for assessing arithmetic speed in fixed-point processors or custom logic.

Many signal processing operations, especially in the data-independent stages, take a finite block of data at their input and produce another block of data at their output. Examples include FFTs and FIR filters, among many others. The elapsed time from when the first element of the input block arrives at the processor to when the first element of the output block is available is called the latency of the operation.[1] This idea is illustrated in Figure 13-3. If throughput measures speed, then latency measures quickness.

High throughput is a requirement for nearly all radar signal processors. Minimizing latency is of greater importance in some systems than in others. For instance, ground MTI (GMTI) radars using adaptive processing to cancel jammers need short latencies to adapt quickly to the interference environment. Latency is less important in SAR systems, which generate imagery that may not be consumed, or even fully formed, until significantly after the data are collected. Generally, short latency requires high throughput, but the converse is not true; pipeline architectures can achieve very high throughputs but still have long latencies.

Some of the most commonly cited hardware metrics are related to peak throughput, but this value alone does not adequately characterize the technology. A processor that achieves 100 GFLOPS might be a single specialized high-performance circuit board or a large cluster of more generic workstations. Consequently, achievable throughput is most useful when normalized by the size, weight, power consumption, or cost of the processor. The corresponding metrics are GFLOPS per cubic centimeter (or cubic inch or foot or liter), kilogram (or ounce or pound), watt, or dollar.

[1]Latency can also be measured from last input to last output, or between any other start and end point definitions, as long as a consistent definition is used in comparing implementations.

Another class of hardware metrics relates to interconnection bandwidth. Examples include the available data rates in bits per second (bps, Mbps, Gbps) to and from a chip or board. Another metric is defined by dividing a processing system into two halves of equal capacity, in such a way that the interconnect bandwidth across the boundary is minimized. The *bisection bandwidth* is the rate of data transfer across that boundary. Bisection bandwidth provides a measure of the interconnect richness and speed in a multiprocessor board or system. As with throughput, interconnect metrics can be normalized with respect to size, power, weight, and cost.

Other metrics describing processor hardware relate to development costs. For example, systems using floating-point hardware typically require less software development effort than those using fixed-point arithmetic (see section 13.6)

13.3.2 Algorithm Metrics

Algorithm metrics traditionally focus on the actual number of arithmetic operations required to compute the specified functions. The quantity of floating-point operations is denoted by the acronym FLOPs (MFLOPs, GFLOPs); note the difference between this and the rate of FLOPs per second, or FLOPS. If an algorithm requires one MFLOPs to compute, and it is implemented on a processor with a throughput of one GFLOPS, the time to completion might be expected to be 1 ms. Reality is more complex, particularly in modern processors.

Another important algorithm metric is the working memory size required to hold the data being processed, any intermediate results, and any algorithm coefficients or parameters required (e.g., the impulse response coefficients in an FIR filter). Algorithms with larger working sets will require larger hardware memories and, if large enough, will suffer performance penalties whenever required data must be fetched from nonlocal memory.

Closely associated with memory size are the concepts of *spatial locality* and *temporal locality*. An algorithm with high spatial locality is one where successive data values used by the algorithm are likely to be stored in adjacent or nearby memory locations, making memory access more efficient. High temporal locality means that all accesses to the same data value tend to be clustered in time (i.e., the data are used for some period of time and then not reused later); such data values do not have to be repeatedly reaccessed from memory, again improving efficiency.

13.4 | COUNTING FLOPs: ESTIMATING ALGORITHM COMPUTATIONAL REQUIREMENTS

13.4.1 General Approach

A first estimate of the required throughput of the radar signal processor can be developed by considering each of the major modes of the radar in turn. For each mode, the major operations to be performed on the data are identified and broken down into basic signal processing functions such as linear filters and discrete Fourier transforms (DFTs). The number of arithmetic operations for each function is totaled. This total will be a function of the various parameters such as data vector lengths, impulse response lengths, and transform size. To convert this number to a computational *rate*, which is more relevant to

TABLE 13-1 ■ Number of Complex Arithmetic Operations per Output Point in Basic Signal Processing Functions[a]

Operation	Multiplications Required per Output Point	Additions Required per Output Point
FIR filter, length L	L	$L-1$
IIR filter, numerator order L, denominator order P	$L+P+1$	$L+P-1$
Autocorrelation, kernel length N	N	$N-1$
Addition of two vectors	0	1
Multiplication of two vectors	1	0
Fast Fourier transform, Cooley-Tukey radix 2	$(1/2)\log_2 N$	$\log_2 N$

[a]Complex data and coefficients assumed.

signal processor sizing, the elapsed time allotted for the operations must also be considered. The final result is a computational rate in FLOPS for that mode. The process is repeated for each mode of the particular radar, such as surveillance, stripmap imaging, spotlight imaging, GMTI, airborne moving target indication (AMTI), target tracking, or target identification. The resulting analysis of computational rates provides an initial estimate of the required signal processor throughput and also identifies the major modes driving the processor size.

As part of the same analysis, the working memory requirements can be estimated by considering the size of the data block at the input and output of each operation as well as any coefficients or auxiliary data required for that block. Provided data word lengths can be specified at each stage, the data rates into and out of each block can similarly be estimated. A spreadsheet is a common tool for building up the computational, memory, and data rate requirements because of the ease with which mode parameters can be varied to see their effect on processing requirements.

The computational requirements of the low-level signal processing operations into which the major operations are decomposed are usually easily determined from the operation formulas or from standard DSP texts such as [3]. Table 13-1 lists these operation counts for the most basic functions. In each case, the formula given is the number of operations per output point computed. For an FIR filter with an impulse response of length L (order $L-1$), computation of the complete output for an N-point input vector requires computing $L+N-1$ output points, while for an N-point FFT (N assumed to be a power of 2 in the table), the output vector is also N points. Element-wise operations on N-point vectors also return N-point results. The length of the autocorrelation of a sequence of length N is $2N-1$ lags.

Many radar signal processing operations, particularly in the earlier stages of the processing, are performed on coherent data and are therefore complex-valued. Thus, the formulas in Table 13-1 should be interpreted as complex multiplication and addition counts. To estimate hardware requirements, it is necessary to convert this to equivalent real-valued operations. For example, addition of two complex numbers requires two real additions, one for the real parts and one for the imaginary parts, a total of two real operations. Multiplication of two complex numbers using the most common algorithm requires four real multiplications and two real additions, a total of six real operations. Table 13-2 lists conversion factors for these and several other complex floating-point operations (CFLOPs) to real floating-point operations (RFLOPs). The last three rows of the tables are estimates

TABLE 13-2 ■ Conversion from Complex to Real FLOPs

Operation	Number of Real Floating-point Operations Assumed
complex-complex multiply	4 real multiplies + 2 real adds = 6 RFLOPs
complex-complex add, subtract	2 real adds + 2 real subtracts = 2 RFLOPs
magnitude squared	2 real multiplies + 1 real add = 3 RFLOPs
real-complex multiply	2 real multiplies = 2 RFLOPs
complex divided by real	1 real inverse + 1 real-complex multiply = 8 RFLOPs
complex inverse	1 conjugate (not counted) +1 magnitude-squared + 1 complex divided by real = 11 RFLOPs
complex-complex divide	1 complex inverse + 1 complex multiply = 17 RFLOPs

because the number of operations for a real inverse, which is required in each of those lines, can vary significantly on different machines, with numbers from four to eight times the number of operations for a real multiply being common [4]. In the table, a value of six RFLOPs per real inverse has been assumed.

The most compute-intensive signal processing operations, such as convolution, correlation, fast Fourier transforms, and vector-matrix operations, tend to require approximately equal numbers of additions and multiplications. As a simple example, the dot product of two complex vectors of length N requires N complex multiplications and $N - 1$ complex additions. Because complex additions require two RFLOPs and complex multiplications require six RFLOPs, a good rule of thumb for front-end signal processing algorithms is that the number of real operations is approximately four times the number of complex operations.

Figures 13-4 and 13-5 illustrate this approach for estimating computational loading. This example is for a fairly complex wideband GMTI mode incorporating reduced-dimension STAP. Each box in the flowgraph of Figure 13-4 represents a major processing operation, such as filtering the input into narrow subbands or pulse compression. Each in turn is decomposed into more basic signal processing operations. For example, the subband filtering would probably be implemented using polyphase FIR filtering, while the pulse compression is implemented using fast convolution, that is, with FFTs. Figure 13-5 is a detailed spreadsheet for the same example in which the user specifies major system parameters in the left portion of the spreadsheet. The middle portion computes a number

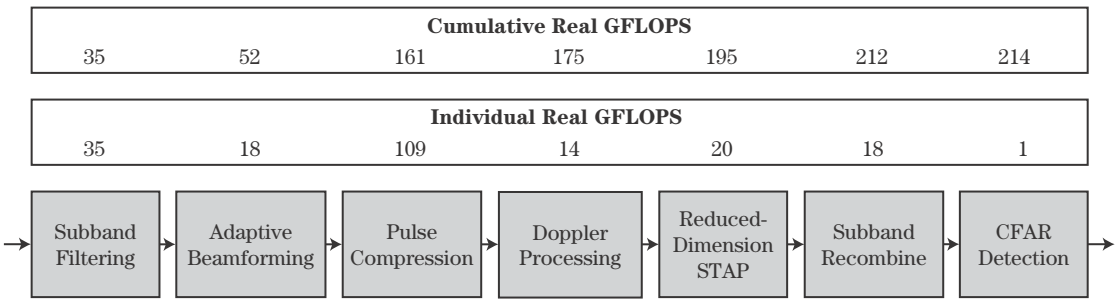

FIGURE 13-4 ■ Estimated computational load (real operations) for a wideband STAP-GMTI radar.

Input parameters:

Signal bandwidth (MHz):	50
Pulse length (us):	100
IF sampling rate (Msamp/s):	125
No. of antenna channels:	6
RMB rule averaging factor:	5
RMB rule averaging factor for STAP:	5
No. of subbands (subbanding FFT size):	16
Receiver duty factor (single pulse):	0.80
Decimation factor:	12
Subbanding prototype filter length:	240
No. of canceller beams formed:	8
No. of beams combined for STAP:	3
No. of Dopplers combined for STAP:	4
No. of STAP beams formed:	4
PRF (pps):	2000
PRI (sec):	5.00E-04
Number of pulses per CPI:	30
CPI duty cycle:	0.8
CPI (sec):	0.019
Adaptive weight update interval (s):	0.01
STAP weight update interval (s):	0.1
CFAR window size (cells):	20
CFAR guard cells:	4

Derived parameters:

Polyphpase subbanding:

Subband filter length:	15
IF oversampling factor:	2.50
Subband sampling rate (Msamp/s):	10.42
Subband bandwidth (MHz):	10.42
Subband spacing (MHz):	7.81
Subband overlap percentage:	25.00%
Number of subbands processed:	7
Suggested subband filter length:	12
Number of range bins:	50000
Number of range bins per subband:	4167
# of subband filter output samples:	4181

Adaptive Beamforming:

Number of DOF:	6
Number of snapshots averaged:	30

Pulse Compression:

Pulse samples:	1042
No. of Overlap-add segments:	5
Length of OLA segments:	837
Length of filtered segments:	1878
Pulse compression FFT size:	2048

Doppler Processing:

Doppler FFT size:	32

STAP:

Number of STAP DOF:	12
Number of snapshots averaged:	60
Number of STAP steering vectors:	128

Polyphase Recombining:

Recombine filter length:	15
Recombine FFT size:	8
Output sampling rate (Msamp/s):	83.3
Number of output range bins:	33334

Computational Load:

Processing Step	GMULTS	GADDS	Combined GFLOPS	GFLOPS	Cumulative GFLOPS
Polyphase subbanding:					
Polyphase FIR filtering:	12.0	11.2		23.3	23
Two-channel FFT:	3.2	4.8		8.0	31
Unravel two-channel FFT:		0.8		0.8	32
Subband modulation:	1.4	1.4		2.8	35
Polyphase subbanding GFLOPS:				34.9	
Adaptive Beamforming:					
SMI weight estimation:			0.26	0.3	35
Diagonal loading:		1.26E-04		0.0	35
Beam formation:			17.2	17.2	52
Adaptive Beamforming GFLOPS:				17.5	
Pulse Compression:					
Forward FFT:	20.2	30.3		50.5	103
Vector multiply:	3.7	3.7		7.3	110
Inverse FFT:	20.2	30.3		50.5	161
OLA additions:		0.7		0.7	161
Pulse Compression GFLOPS:				109.0	
Doppler Processing:					
Weighting:	0.7			0.7	162
Range curvature compensation:	1.5	1.5		3.0	165
Doppler FFT:	4.0	6.0		10.0	175
Doppler Processing GFLOPS:				13.7	
STAP:					
SMI Weight estimation:			0.7	0.7	176
Diagonal loading:		8.40E-07		0.0	176
Beam formation:			18.8	18.8	195
STAP GFLOPS:				19.5	
Polyphase Recombining:					
Subband modulation:	0.8	0.4		1.2	196
IFFT:	1.4	2.0		3.4	199
Polyphase FIR filtering:	6.9	6.4		13.2	212
Polyphase recombining GFLOPS:				17.9	
CFAR:					
Squared-magnitude detection:	0.5	0.2		0.7	213
Threshold estimation:	0.2	0.5		0.7	214
CFAR GFLOPS:				1.4	
TOTAL GFLOPS:				214.0	

FIGURE 13-5 ■ Example of spreadsheet for estimating computational load for a wideband STAP-GMTI system. Counts are for real operations.

of derived parameters, and the right-hand portion is the actual FLOPS computation. Note that the underlined subheadings in this section correspond to the stages of the signal flowgraph in Figure 13-4.

Returning to Figure 13-4, the row of numbers immediately above the operations is the spreadsheet estimate of the computational rate in real GFLOPS required for each individual operation for the particular parameters used. The upper row of numbers is the cumulative operation rate through that stage.[2] For instance, the first three operations require an estimated 35, 18, and 109 GFLOPS, respectively. The upper set of numbers reflects the total of 161 GFLOPS for those three operations.

Additional examples of this type of operation-counting analysis of complex radar signal processing modes are given in chapters 6 and 15 of [5].

13.4.2 Mode-Specific Formulas

The radar literature provides parametric formulas for rough estimates of computational loading for certain modes. Two common examples are SAR imaging and STAP.

An example of a simple formula applicable to stripmap SAR is developed in [6]. The formula begins by factoring the total computational rate F_t in FLOPS into two terms, the number of image pixels generated per second (R_p) and the number of FLOPs required per pixel (F_p)

$$F_t = F_p R_p \tag{13.1}$$

Consider the pixel rate first. This can in turn be expressed as the number of pixels generated per pulse (N_p) times the pulse repetition frequency (PRF),[3]

$$R_p = N_p PRF \tag{13.2}$$

The number of pixels per pulse is very design-specific but is at most the unambiguous range interval, which is the maximum SAR swath depth divided by the range resolution

$$N_p \leq \frac{(c \cdot PRI/2)}{(c/2B)} = B \cdot PRI = \frac{B}{PRF} \tag{13.3}$$

where B is the waveform bandwidth. This result assumes that the range sample spacing equals the range resolution, that is, one range sample per range pixel. Combining equations (13.2) and (13.3) shows that the pixel rate is bounded by a number on the order of the waveform bandwidth,

$$R_p \leq \left(\frac{B}{PRF}\right) PRF \quad \Rightarrow \quad R_p \sim O(B) = O\left(\frac{c}{2\Delta R}\right) \tag{13.4}$$

where ΔR is the range resolution of the SAR. The notation $O(x)$ means "on the order of x."

---

[2] Slight differences between the listed cumulative GFLOPS and the running sum of the individual GFLOPS are due to rounding.

[3] This assumes single-look SAR image products but is sufficient for rough processor sizing. The approach is extendable to multilook images.

F_p, the number of FLOPs per pixel, is a strong function of the image formation algorithm used. A middle-of-the-road estimate can be developed from the viewpoint that SAR imaging requires a matched filtering (correlation) operation in cross-range. The number of samples N_{CR} of the correlation kernel equals the number of cross-range samples (pulses) that contribute to a single pixel; this number is upper bounded in stripmap SAR by the cross-range spread of the antenna beam expressed in SAR pixels

$$N_{CR} = \frac{R\theta_{az}}{\Delta CR} = \frac{R\lambda}{D_{az}\Delta CR} \tag{13.5}$$

where θ_{az} and D_{az} are the antenna beamwidth and size in the cross-range (width) dimension, ΔCR is the cross-range resolution of the image, and λ is the radar wavelength. Since a correlation operation requires a multiplication and an addition for each sample of the kernel function, the number of operations per output pixel, F_p, is twice N_{CR}. The lower bound on ΔCR for a fully focused stripmap SAR is $D_{az}/2$, so that $D_{az} = 2\Delta CR$ [7]; using this result in equation (13.5) gives

$$F_p = 2N_{CR} \leq \frac{R\lambda}{(\Delta CR)^2} \quad \Rightarrow \quad F_p \sim O\left(\frac{R\lambda}{(\Delta CR)^2}\right) \tag{13.6}$$

The estimate of equation (13.6) is most appropriate for a simple range-Doppler imaging algorithm or similar. More complex algorithms, such as the ω-k (also called range migration) algorithm with Stolt interpolation may require four times as many operations [8]. On the other hand, FFTs can be used to implement the brute-force correlation operation implied by this discussion if range curvature is insignificant; in this case, F_p may be reduced significantly.

Combining equations (13.4) and (13.6) gives the final result

$$F_t \sim O\left(\frac{R\lambda c}{\Delta R(\Delta CR)^2}\right) = O\left(\frac{R\lambda c}{(\Delta CR)^3}\right) \tag{13.7}$$

where the last step assumes "square pixels," $\Delta R = \Delta CR$. Other variations on this result are given in [6–9]. Note that the computational load rises rapidly with increasingly fine resolution: Improving (reducing) the resolution by a factor of 2 produces an $8\times$ increase, nearly an order of magnitude, in the computational rate.

As an example, consider applying this formula to the Shuttle Imaging Radar-C (SIR-C). This system operates at 1.28 GHz from an orbit altitude of 225 km. At 25 m resolution, the resulting estimated computational rate is 506 (complex) MFLOPS. For a more aggressive system, consider a low Earth orbit (LEO) satellite operating at X band. Using an orbit altitude of 770 km and an RF of 10 GHz, the computational rate is estimated at 55 GFLOPS at a resolution of 5 m. This number is probably too low, since a higher-quality algorithm would be required at that resolution. Thus, the true rate is more likely to be three to four times higher, on the order of 150–200 GFLOPS.

Similar estimates have been developed for STAP, another compute-intensive signal processing operation. Derivation of this result requires analysis of the computational load of matrix algebra operations such as the Q-R decomposition, which is beyond the scope of this chapter. A typical result for the "voltage domain" approach is [7]

$$\text{RFLOPS} = \begin{cases} F_u\left[\left(8L_sR^2 - 2.67R^3\right) + 12KPR^2 + 26KPR - 4KP\right] \text{ (weight computation)} \\ (8R - 2)KPLF_{CPI} \text{ (weight application)} \end{cases} \tag{13.8}$$

where

- L_s is the number of data snapshots averaged to estimate the interference statistics;
- R is the number of sensor degrees of freedom (DOF, product of the number of slow-time pulses and number of antenna phase centers);
- K is the number of Doppler beams tested for targets;
- P is the number of angles of arrival (AOA) tested;
- F_u is the weight update rate; and
- F_{CPI} is the number of coherent processing intervals (CPIs) per second

Equation (13.8) is expressed in units of RFLOPS and includes both multiplications and additions. Note that, as with SAR, the computational load increases as the cube of a key parameter, in this case the number of degrees of freedom R of the STAP system.

As an example, consider a system with $N = 8$ phase centers and $M = 20$ pulses in the CPI, giving $R = MN = 160$ DOF. Limiting the covariance estimation matrix loss to 3 dB requires $L_s = 2R = 320$ reference cells [7]. Assume that the weights are updated at rate of $F_u = 20$ updates per second. Assume also that $K = 20$ Doppler frequencies and that $P = 12$ AOAs are tested in each range bin. Equation (13.8) then gives a computational rate of 2.6 GFLOPS to compute the adaptive weights. Assuming $F_{CPI} = 375$ CPIs per second then gives 115 GFLOPS to apply the weights and a total computational load of 117.6 GFLOPS for this system. Similar results for rough estimates of STAP processing loads are given in [10].

For both SAR and STAP-GMTI, it is important to remember that many algorithm variations exist. Some of the limitations to the SAR estimates were previously discussed. In STAP, much attention has been devoted to reducing the number of degrees of freedom to reduce both the data needed for interference statistics estimation and the computational load [11]. Thus, the formulas in this section, while useful as examples of an approach to developing rough estimates of computational loads, must be used with caution. Serious analysis of processor requirements calls for more detailed analysis specific to the particular algorithms of interest, typically starting with a spreadsheet analysis as discussed earlier.

13.4.3 Choosing Efficient Algorithms

It is frequently the case that a given signal processing operation can be implemented with more than one algorithm. A simple example is the DFT. A brute-force implementation of an N-point DFT requires N^2 complex multiplications and a similar number of complex additions. As is well known, however, the DFT can be computed much more efficiently with one of the many FFT algorithms. For the oft-used case where N is a power of 2, the Cooley-Tukey radix 2 FFT [12] computes the DFT using $(N/2)\log_2 N$ complex multiplications, a reduction by a factor of $2N/\log_2 N$. This speedup factor is illustrated in Figure 13-6, which also makes clear that, as is the case with most fast algorithms, the impact of the FFT algorithm increases with the problem order. A common benchmark is the $N = 1{,}024$ (1K) DFT, for which the speedup is a factor of just over 200.

Modern FFT algorithms of order $N\log N$ exist for almost all lengths, not just powers of 2, as well as for special cases such as real-valued inputs, or those having certain symmetries. In addition, modern FFT libraries have the capability to adaptively construct algorithms optimized for particular processors, matching the algorithm structure to the memory hierarchy and other characteristics of the host processor [13]. In fact, the

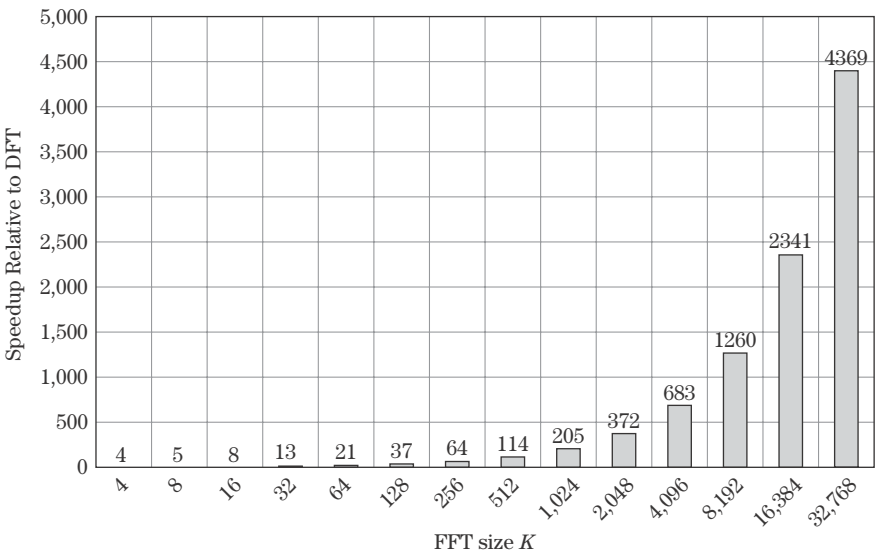

FIGURE 13-6 ■ Speedup of Cooley-Tukey radix-2 FFT relative to brute-force DFT.

complexity of modern processors makes it essential to optimize the FFT algorithm (and other algorithms as well) for the particular processors being used.

Another example of a core signal processing algorithm that can be computed using different algorithms is convolution. The brute-force algorithm for convolving an N-point data sequence $x[n]$ and an L-point filter impulse response $h[n]$ is

$$y[n] = \sum_{m=0}^{L-1} h[m]x[n-m] \tag{13.9}$$

As noted earlier, this requires L multiplications for each of the $N + L - 1$ nonzero output samples, a total of $L(N + L - 1)$ multiplications; if the data and impulse coefficients are complex, these are complex multiplications.

The same computation can be done using the multiplication property of DFTs; specifically

$$\begin{aligned} y[n] &= \text{DFT}^{-1}\{Y[k]\} = \text{DFT}^{-1}\{H[k]X[k]\} \\ &= \text{DFT}^{-1}\{H[k] \cdot \text{DFT}\{x[n]\}\} \end{aligned} \tag{13.10}$$

$H[k] = \text{DFT}\{h[n]\}$ is generally precomputed, so the DFT to obtain $H[k]$ is not shown in equation (13.10). Consequently, one forward and one inverse DFT are required to compute the convolution output. The minimum size of the DFT is $N + L - 1$ (see chapter 14). The advantage to this *fast convolution* approach is that efficient FFT algorithms can be used to compute the forward and inverse DFTs.

Figure 13-7 illustrates the number of multiplications required to compute $y[n]$ by brute-force convolution (13.9), fast convolution (13.10), or the overlap-add (OLA) method [3] using segment sizes of $L = 100$ and 500. In the latter cases, fast convolution is used for the small segment convolutions with power-of-2 FFTs. The "stair steps" in all of the results except the direct convolution occur when it is necessary to increase the FFT size to the next power of 2, for example, from 512 to 1,024.

Of the algorithms considered, the brute-force convolution is the least efficient for most, but not all, input data lengths. The most efficient choice of those shown is the

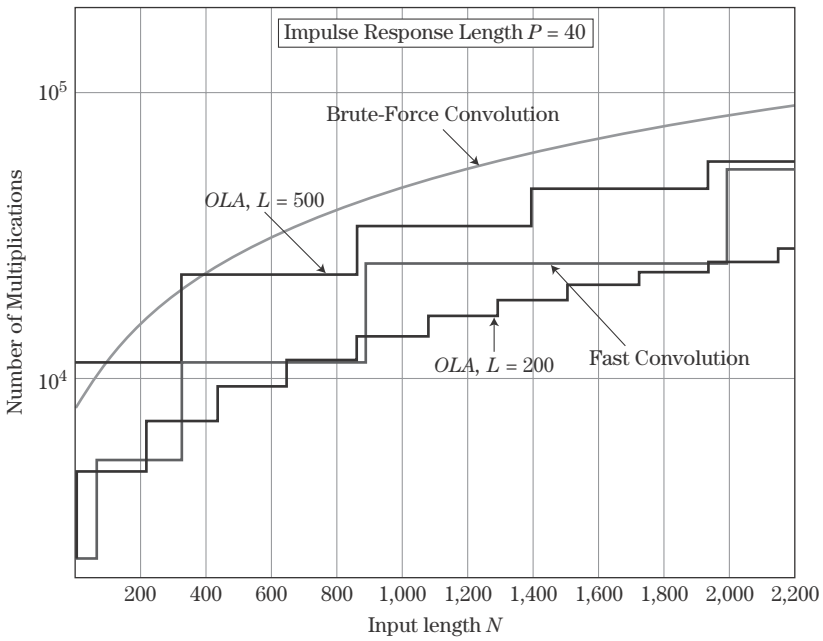

FIGURE 13-7 ■ Number of multiplications required to compute a linear convolution using various algorithms.

OLA algorithm with short $L = 100$ sample segments for most, but again not all, lengths. The difference in efficiency of the various algorithms is significant. For instance, with $N = 1{,}000$, the brute-force convolution requires 41,560 multiplications, while the OLA with $L = 100$ requires only 13,824, a reduction by a factor of three. For most input lengths, the ratio between the number of multiplications required for the least and most efficient algorithms is between 3 and 3.5. Thus, the overall processing workload can be significantly affected by the choice of the specific algorithm used for each of the major operations.

These examples show that implementing the series of operations that comprise a radar mode with other than the most efficient algorithms can result in large increases in the total number of required operations. This translates into either an increase in the time to solution or an increase in the size, weight, power, and cost of the required signal processor to compensate for the increased workload. If real-time processing is required, the time to perform the operations required on one block of data cannot exceed the block time or the processor will not be ready for the next block of data.

The previous discussion has focused on minimizing the number of operations in each algorithmic step. As noted earlier for FFTs, in modern processors the optimization of data movement and hierarchical memory is often more important than the number of arithmetic operations in determining run time. Thus, in some cases the best algorithm may not be the one with the fewest operations.

13.5 IMPLEMENTATION TECHNOLOGY

13.5.1 Analog-to-Digital Conversion

At some point in every modern radar system, the receiver output will be converted from an analog signal to a digital signal by an *analog-to-digital* (*A*/*D*) *converter*. The required sampling rate is determined through the Nyquist criterion by the instantaneous bandwidth

of the receiver output, while the number of bits in the digital sample is determined by the instantaneous dynamic range of the data and signal-to-quantization noise ratio (SQNR) requirements of the system. Nyquist sampling and the effect of the number of bits on dynamic range and SQNR are both discussed in chapter 14.

The required sampling rate is generally equal to the radar instantaneous waveform bandwidth B, perhaps increased by a safety margin of 10% to 20% to allow for the imperfect bandlimiting of real waveforms and anti-aliasing filters. In high-resolution radars the resulting sampling rates can be very high, often tens to hundreds of megasamples per second and in some cases even reaching rates of gigasamples per second. The use of digital I/Q or digital IF techniques for non-baseband sampling can increase the required rate relative to the waveform bandwidth by a factor of 2.5 to 4, further exacerbating the problem [7]. On the other hand, many very wideband systems use a specialized linear FM waveform processing method called *stretch processing* (see [7]) to reduce the analog signal bandwidth prior to A/D conversion, often by an order of magnitude or more.

Except in a few specialized situations, the minimum number of bits required is at least 6 and preferably 8, while 12 bits or more are desired in many applications [14]. Interest is increasing in the use of sigma-delta converters, which trade very small quantizer word lengths (as little as one bit) for much higher sampling rates [15].

Figure 13-8 summarizes the state of the art in A/D converters in 2006 [16]. The quantity plotted is the effective number of bits (ENOB) versus sampling rate. ENOB is inferred from measured SQNRs as described in [17], which also discusses many other A/D metrics and limiting factors. That data show that A/D converter word lengths tend to drop about two bits per decade of sampling rate. However, an ENOB of 8 bits is achievable at rates up to about 1 Gsamples/sec, and an ENOB of 12 bits is achievable at rates up to approximately 100 Msamples/sec. These are quite adequate for most radar applications. Improvement in achievable ENOB over time is somewhat slow, constrained by progress in overcoming design limitations such as sample timing jitter and fundamental circuit limitations. The data in [17] estimate that ENOB increased about 3.5 bits at a given sampling rate between 1999 and 2006 and project a further increase of about 2 bits by 2015.

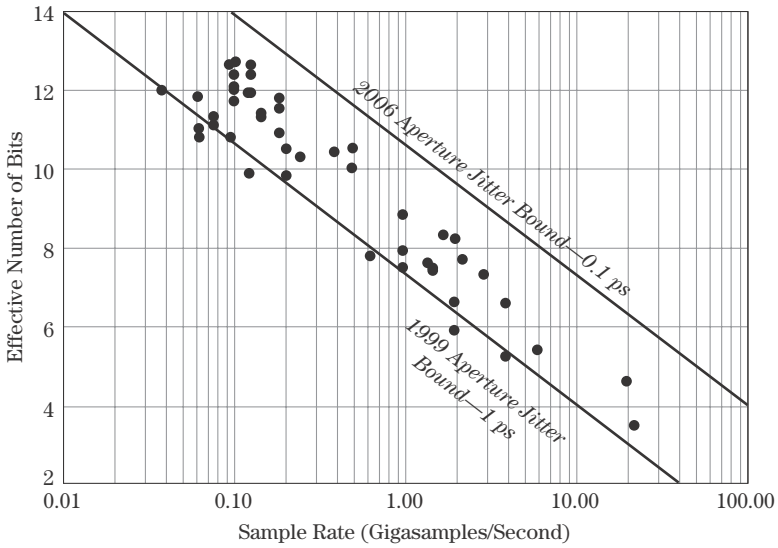

FIGURE 13-8 ■ Analog-to-digital converter performance. (Data courtesy of Dr. Robert Walden and The Aerospace Corporation. Used with permission.)

13.5.2 Processor Technologies

As was suggested in Figure 13-2, the major types of computing hardware used in implementing radar signal processors can be grouped into four broad classes: ASICs; reconfigurable modules; programmable signal processing modules; and general-purpose computing hardware. The first three classes of hardware are used for signal processing and the last for data processing.

Here, *programmable DSP module* refers to both microprocessors specialized for digital signal processing, such as those manufactured by Texas Instruments, Inc. (TI) and Analog Devices, Inc., and modular board-level products such as those manufactured by Mercury Computers, Sky Computers, CSPI, and Radstone. The board-level products typically are multiprocessors based on microprocessors such as the TI DSPs, the IBM PowerPC series, or the Analog Devices SHARC series. *General-purpose processors* refer to workstations and SBCs that are effectively workstations on a single modular circuit board, intended to be used as part of a larger system.

Reconfigurable hardware modules refers to computing elements, typically at the modular board level, usually based on *field programmable gate arrays* (FPGAs) as the core computing elements [18]. The FPGAs are manufactured by companies such as Xilinx and Altera and are integrated into board-level devices by companies such as Annapolis Microsystems and Nallatech. Classically, FPGAs provided a "sea of gates" (actually, logic cells organized into registers and multi-input look-up tables) arranged in a grid layout. An externally supplied control data stream could reconfigure the interconnection of the various registers and blocks, allowing the designer to construct semicustom complex logic functions. Newer FPGAs often add fixed-function blocks (e.g., adders or multipliers), memory, and even small programmable microprocessor cores to the basic reconfigurable logic.

Table 13-3 compares the signal processing technology classes in broad terms in several areas. In general, the performance (considered as throughput per unit volume, power, or weight) declines as the implementation technology progresses from ASICs to programmable modules. However, so does the cost, both in acquisition cost and in nonrecurring engineering effort (NRE).

Ease of functional change refers to the ability of the processing technology to adapt to an upgrade or change in processing algorithms. An ASIC can be designed to meet exactly the requirements of a particular system, but, once fabricated, it cannot be modified to meet the needs of an upgraded or different system. The programmable DSP modules are programmable in both assembly language and higher-level languages such as C or C++. As such, they can be readily adapted to new algorithms. The programmable DSP might be considered somewhat less flexible than a traditional general-purpose CPU in

TABLE 13-3 ■ Comparison of Signal Processor Implementation Techniques (after [18])

Technology	Performance	Cost	Power	Ease of Functional Change	Design Effort
ASIC	High	High	Low	Low	High
Reconfigurable Hardware Module	Medium	Medium	High	Medium	Medium
Programmable DSP Module	Low	Medium	High	High	Low

that some effort is needed to structure the source code to make sure the DSP's specialized computing resources are used effectively; simple single-thread, serial code is not generally adequate. To get the maximum efficiency from the device, the programmer must take care to identify and manage data and task parallelism so that all computing resources are kept busy, that computation is overlapped with communication, and so forth. Reconfigurable hardware is also programmable, though usually at a much lower level such as register transfer level (RTL) or through hardware description languages such as VHDL, requiring more specialized skills. It is more difficult to express signal processing algorithms at this level than in high-level languages, accounting for the greater design effort.

The arithmetic format also tends to vary between each technology type. Both fixed-point and floating-point designs exist in each of these classes. However, at least as applied to radar, it tends to be the case that reconfigurable hardware is more likely—and ASICs more likely still—to use fixed-point than programmable hardware. This is because the more specialized hardware is used for portions of the signal processing flow that require high throughput; are tightly constrained in size, weight, and power; or both. These same constraints tend also to favor the faster, smaller, and lower-power fixed-point arithmetic over floating point. Systems that combine multiple classes of hardware may thus also combine fixed-point processing in earlier stages with floating point in later stages.

The first decade of the 21st century has seen the emergence of new *multicore processor* architectures. Two examples of great interest within the radar signal processor community are the Sony/Toshiba/IBM Cell Broadband Engine (CBE) and graphical processing units (GPUs). The CBE combines a PowerPC microprocessor and on-chip random-access memory (RAM) with an array of eight signal processing elements into a single chip capable of about 200 GFLOPS in single precision [19]. The CBE was designed as the processor for the Sony Playstation 3 game machine. The primary manufacturers of GPUs are Nvidia and ATI (now part of AMD, Inc.). Designed to generate the graphics for high-end personal computers, current (2009) high-end GPUs offer performance of up to one teraflops in a single device having 240 core processors [20]. These devices hold promise of providing performance rivaling or even exceeding that of reconfigurable modules, but with development costs closer to that of more standard programmable devices because they can be programmed in standard high-level languages. For instance, an improvement of about $35\times$, compared to a standard microprocessor, in the execution time of a two-dimensional (2-D) interferometric imaging radar 2-D phase unwrapping algorithm was demonstrated in [21]. Furthermore, the CBE and GPUs potentially offer extremely good performance per dollar because both are commodity chips developed for very large markets, thus reducing the device cost.

The primary difficulty in using these devices, as well as many other emerging multicore systems and multiprocessor systems in general, is programming them to operate at peak rates. Partitioning algorithms among many processors and choreographing the data to keep all of the processing units busy is a difficult task. In addition, the developer must choose algorithms that are well matched to the architecture of the device being used [21]. Tools such as Nvidia's CUDA development environment aid in applying GPUs to more general high-performance computing applications. OpenCL [22] is an emerging open standard for parallel programming on heterogeneous, many-core architectures that may improve productivity and portability across GPUs from multiple vendors, the CBE, "traditional" dual- and quad-core processors, and other emerging devices.

A thorough discussion of the many classes of current and emerging processors that are candidates for radar signal processing is given in chapter 26 of [5].

13.5.3 COTS Technology and Modular Open Architectures

Since at least the 1980s, it has been increasingly common to implement radar signal processors whenever possible using commercial off-the-shelf components. This has been the natural result of the "leading edge" in electronics development moving from military systems to the much larger consumer products market. The benefits of COTS components include the following [23]:

- Incorporation of the rapid advances in commercial technology into radar systems more quickly than with conventional military acquisition practices.
- Reduction or avoidance of the research and development (R&D) costs for new processors.
- Cost savings associated with products developed for large consumer markets and multivendor competition.

For military systems, there are several drawbacks to COTS as well. These include the difficulty in obtaining components that:

- Meet military specifications, or in some cases reduced "ruggedization" specifications, on environmental characteristics such as temperature and vibration.
- Meet military requirements for safety, security, certification, hard real-time performance, and so forth.
- Do not become obsolete or unsupportable too quickly or fail to provide a certifiable upgrade path.

The rapid evolution of COTS processing technology is thus both a blessing and a curse. It allows rapid improvements in processor capability but makes it difficult to support deployed military systems for their typically multidecade life spans.

A small example from the author's experience illustrates some of the benefits of transitioning to COTS processor implementations. In 1991, an implementation of a particular real-time airborne SAR image formation processor required nine custom 12" $\times$ 14" 12-layer custom circuit boards. Of these nine boards, six were vector signal processors, and three were specialized "corner-turn" memories. The arithmetic was implemented in a combination of 16-bit fixed-point and 24-bit block floating-point formats; floating point could not be supported. All of the interfaces were custom designed. Six years later, the processor was reimplemented. This time, the entire system was hosted on four COTS 6U VME (6.3" $\times$ 9.2") boards, each containing four IBM PowerPC microprocessors. All of the arithmetic was 32-bit IEEE 754 floating point, and the major interfaces were the open standard versa module europe (VME) and RACEway protocols. Thus, the size of the processor was greatly reduced; the numerical quality of the algorithm implementation was improved; and the design cost was also greatly reduced. If the same processor were reimplemented today, it would require at most two COTS boards, and quite probably only one.

COTS-based signal processors are usually *open architecture* system designs as opposed to proprietary architectures [24]. Open architecture signal processors generally use a modular design approach. In a modular processor, a standard form factor chassis provides a physical enclosure, and the chassis backplane provides a control bus for the modules. The VME form factor, which uses circuit boards in a number of different sizes, is a frequent choice. Commonly used sizes are denoted (in order of increasing size) 1U, 3U, 6U,

and 9U. The computing resources required for a particular application are obtained by inserting modules of the required type and number into the backplane. The "menu" of example module types might include the following:

- Single-board general-purpose computer modules.
- Fixed- and floating-point arithmetic modules.
- General or specialized memory modules.
- High-speed interconnect modules.
- Sensor interface modules.
- Timing and control generators.
- Power supply and conditioning modules.
- System-specific interface and control-signal modules.

The reliance on open, public standards for the mechanical, electrical, and logical interfaces allows successive models of equipment from a vendor, or competing equipment from another vendor, to be inserted into a modular system to replace or upgrade existing modules with minimal redesign and cost. Open architecture approaches do not obviate the need for testing and certification, but they provide a manageable approach to maintaining equipment over a long life cycle in an environment of rapid commercial technology product cycles.

Open modular architectures also enable the development of systems that combine multiple styles of computing devices in a single chassis. For example, the FPGA-based reconfigurable computing boards discussed previously are implemented in VME and other standard form factors. They can thus be integrated in a VME chassis with conventional processors, memories, interfaces, control computers, and power supplies.

There are numerous examples of open architecture interconnection standards. The VME (IEEE P1014–1987) and VME64 (IEEE P1014 Rev D) control bus standards for modular system backplanes remain very common in commercial products. RACEway (ANSI/VITA 5–1994) and SCI (IEEE 1596–1992) provide high-speed intermodule and intramodule data interconnects for multiprocessor systems. Newer high-speed interconnect standards include Serial RapidIO, PCI Express, InfiniBand, and Gigabit Ethernet. These buses provide effective data rates in the range of 940 to 980 MB/s. A thorough discussion of interconnection fabrics and standards, as well as of the emerging VME switched serial (VXS) standard (VITA 41.2) for VME-compatible packaging of these interconnect schemes, is given in chapter 14 of [5]. Other standard bus architectures and protocols exist for test and maintenance (e.g., high-performance serial bus (FireWire), JTAG), and for input/output (Fiber Channel, MIL-STD-1553B).

Figure 13-9 illustrates a typical COTS 6U VME signal processor. This particular system from Mercury Computer Systems, current in late 2009, was said to provides 761 GFLOPS of computational capability using quad MPC7448 PowerPC modules as well as 42 GB/s sustained throughput using the Serial RapidIO interconnect fabric. The system is compliant with the VXS form factor standard.

The most important open software standards for radar signal processors are probably Vector Signal Image Processing Library (VSIPL), VSIPL++, and message passing interface (MPI) [25]. VSIPL is an application programming interface (API) for functions such as FFTs, FIR filters, and linear algebra commonly used in signal processing [26]. Manufacturers of commercial signal processing modules typically also provide signal processing libraries optimized for their products' architectures. The proprietary libraries

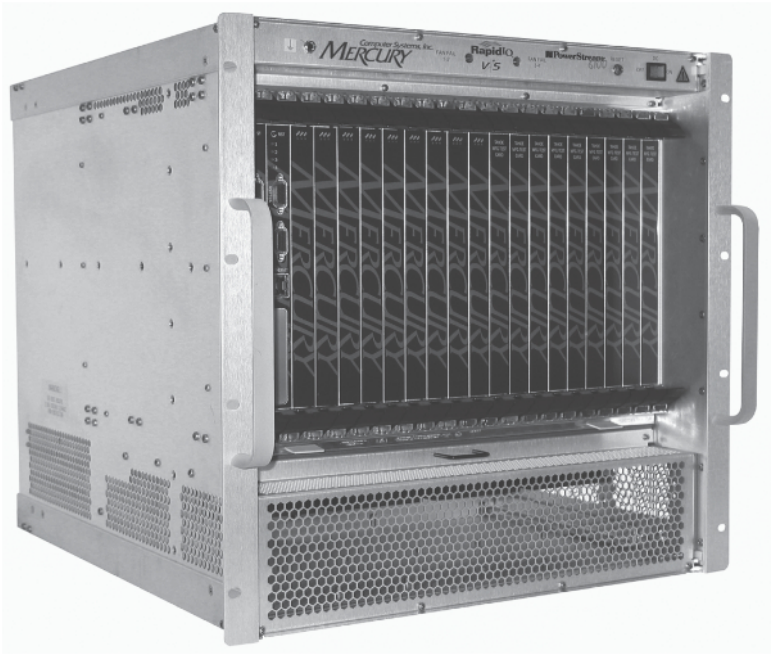

FIGURE 13-9 ■ Example of commercial off-the-shelf modular signal processor. (Copyright Mercury Computer Systems. Used with permission.)

of all such vendors would include, for example, a one-dimensional FFT function, but the specific name of the function and the calling sequence of its arguments will vary from vendor to vendor. Thus, application code written for one vendor's boards is not portable to another's, even if the hardware is interchangeable. VSIPL provides a standardized interface to these functions. Products from any vendor that supports the VSIPL API will then be more software compatible at the source code level. VSIPL++ provides a C++ binding of VSIPL with additional functionality [27].

MPI plays a similar role for interprocessor communication [28]. It provides a portable, open standard for the functions needed to support the message passing communication protocol on high-performance multiprocessor machines, especially those using distributed memory.

13.5.4 The Influence of Moore's Law

Gordon Moore of Intel first articulated what later became known as *Moore's Law* in a prescient 1965 paper [29]. Moore's Law, as it is now commonly understood, states that the number of transistors in a single integrated circuit doubles approximately every 18 to 24 months. This is evidenced by the history of Intel microprocessors, shown in Figure 13-10 [30]. Other microprocessor metrics such as clock speeds or functionality per unit cost also tend to increase exponentially. Indeed, the rapid increase in computing power per unit cost is evident to every consumer of personal computers, cell phones, and other consumer devices every time he or she visits the local electronics store. Moore's Law has a dark side as well: Power consumption, design costs, and infrastructure costs also increase exponentially. The history and implications of Moore's Law are discussed in [31], and its impact on signal processing is examined in [32].

Moore's Law describes improvements in individual integrated circuits. Obviously these carry over to complete computing systems, but the performance of complete systems

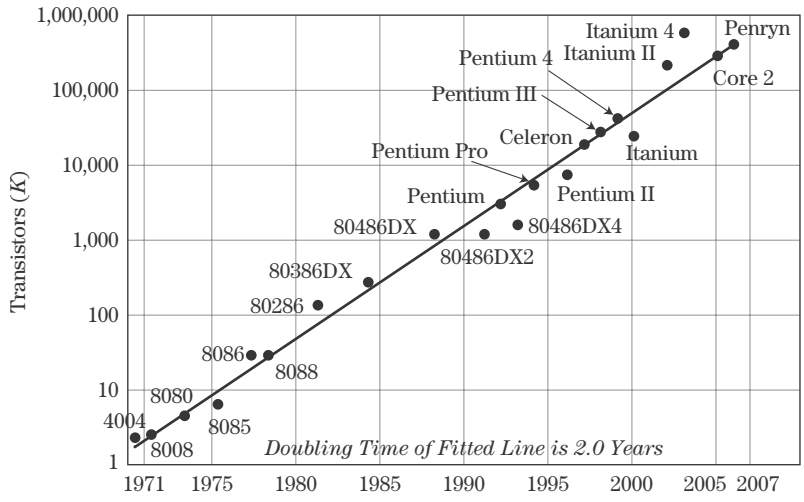

FIGURE 13-10 ■ Moore's Law, as illustrated by Intel microprocessors. (Data courtesy of Intel, Inc.)

is also impacted by the performance of other components, for example, system bus speeds, storage access speeds, and general system architecture. In fact, in recent decades the speed of computing systems as a whole has progressed more quickly than has the speed and density of individual integrated circuits. One illustration of this is the "Top 500" supercomputer rankings, which twice annually updates a list of the 500 fastest computers, based on a specific floating-point LINPACK benchmark [33]. Figure 13-11 shows the increase in speed of the fastest, 100th fastest, and 500th fastest computer over a 15-year period. The average rate of growth in performance corresponds to a doubling every 13 to 15 months, quicker than the 18- to 24-month doubling period of Moore's Law.

The fastest supercomputers are not candidates for embedded processing because they are typically building-sized systems. In mid-2008, the top-ranked system in the Top 500 list was the IBM "Roadrunner" system designed for the U.S. Department of Energy and

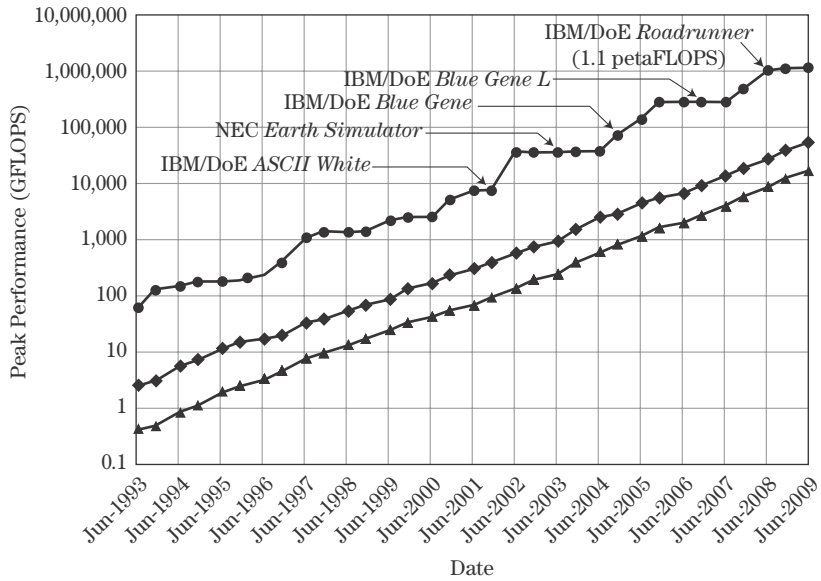

FIGURE 13-11 ■ Peak throughput of the fastest, 100th fastest, and 500th fastest computers in the Top 500 list over time.

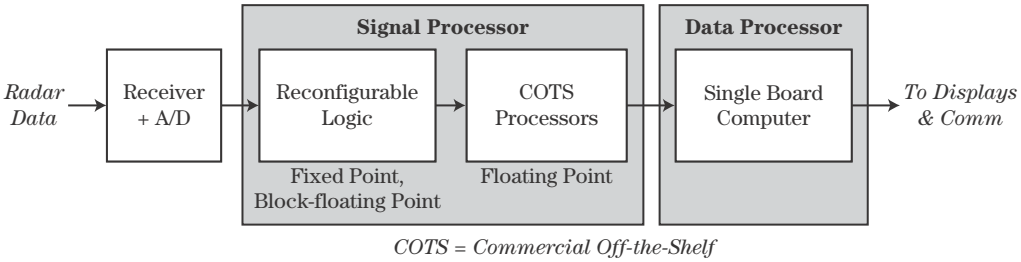

FIGURE 13-12 ■ Simplified processor architecture.

used in weapons research.[4] The machines at the #100 and #500 spots on the list are typically workstation and personal computer (PC) clusters in commercial usage. These more general-purpose machines are better indicators of the performance improvements of computer systems in general.

Moore's Law ensures that the capability of all three classes of signal processing hardware increases rapidly with time. It follows that, for a fixed processing requirement, the need for the more specialized processing technologies such as ASICs will decline over time. Specifically, as FPGAs become faster they become increasingly able to host the more demanding front-end stages of high-throughput algorithms as well as to implement floating-point instead of fixed-point arithmetic, to provide more on-chip memory, and so forth. Substituting reconfigurable hardware for ASICs allows major reductions in design cost and time as well as an increase in the ability of the processor to adapt to new algorithms. Thanks to Moore's Law, the signal processor in a wide variety of systems can be simplified from the design of Figure 13-2 to that of Figure 13-12, where only COTS reconfigurable hardware and programmable processors are needed to implement the radar signal processing algorithms [34]. Frequently, such designs can also use floating-point arithmetic throughout, even in the FPGAs. Signal processors based on reconfigurable and programmable COTS processors with floating-point arithmetic realize major savings in algorithm development cost and time as well as superior numerical performance.

Nonetheless, for the fastest, most tightly constrained processors, ASICs will remain a necessary part of the design. Space-based radars and unmanned aerial vehicle (UAV) radars with high-resolution imaging and ground moving target indication modes are most likely to require ASICs as part of the processor implementation due to high computational loads and very tight payload constraints. An example of the capability that can be obtained in an ASIC designed for radar front-end processing is given in [35]. This bit-level systolic fixed-point design achieves over 1 trillion operations per second in about one-eighth of a cubic foot, 12 kilograms, and 50 watts in a 0.25 μm CMOS technology in the year 2000.

13.6 FIXED POINT VERSUS FLOATING POINT

The choice of arithmetic format in a radar signal processor is a significant design issue, affecting numerical attributes such as accuracy, dynamic range, stability; software development costs; and hardware attributes such as speed, size, weight, and power

[4]This machine is the first to break the petaflops barrier (10^{15} FLOPS) using the Top 500 metric.

consumption. The principal choice is between fixed-point and floating-point arithmetic, though compromise formats such as block floating point [36] are used in some cases.

Digital logic for implementing fixed-point arithmetic in common word lengths may allow reductions of 30% to 50% in silicon area compared with floating-point implementations in custom digital circuits or FPGAs. Because it does not have to process exponents, fewer logic stages are needed in a fixed-point arithmetic unit, allowing it to execute computations at a faster rate. Thus, fixed-point implementations are more efficient than floating point in size, weight, and power consumption. However, when combined with memory and other functions on a complete processor chip, the difference between fixed-point and floating-point devices is reduced.

The chief disadvantages of fixed-point arithmetic are its limited dynamic range and signal-to-quantization noise ratio, both increasing at 6 dB per bit [7], and the problems of underflow and overflow of arithmetic computations. Preventing or controlling underflow and overflow requires additional scaling operations between arithmetic functions and can increase both the development cost and the implementation requirements of a fixed-point implementation of an algorithm. For instance, a 16-bit fixed-point data representation limits the dynamic range of the data to approximately 96 dB. While this may seem adequate, arithmetic operations on the data increase the dynamic range. For example, the sum of two positive numbers is larger than either alone. If two numbers that are near the maximum representable in a given format are added, the result will be twice that maximum.

To guarantee that the sum does not overflow, either the operands must be limited to a range one bit less than the maximum dynamic range or the word length must be extended by one extra bit. Similarly, the sum of N numbers has a range potentially $\log_2 N$ bits larger than that of the individual summands. The product of two operands can double the required word length. Combining these results, an operation such as a dot product of N-element vectors of M-bit numbers can require as much as many as $2M + \log_2 N$ bits. Overflow can be prevented by scaling the data down prior to an operation by enough bits to ensure that overflow cannot occur, but this approach reduces the number of bits of precision of the result. More sophisticated approaches scale the data in stages, minimizing precision loss while reducing or avoiding overflow [4,36]. Even with scaling, algorithms that explicitly or implicitly require matrix inversion, such as those common in STAP and other adaptive processing, can be very challenging to implement with adequate numerical accuracy in fixed-point arithmetic. While tools exist to aid in fixed-point algorithm design, traditionally they have been relatively limited and immature compared with the design environments available for floating-point design.

Floating-point arithmetic eliminates essentially all dynamic range and scaling issues. While a variety of formats are used, increasingly the most common is the IEEE P754 32-bit format [37]. Using a 24-bit mantissa and an 8-bit exponent in the single precision version, this 32-bit format can represent numbers over a range of about 72 orders of magnitude. The double precision version uses a 52-bit mantissa and an 11-bit exponent, increasing the representable range to about 610 orders of magnitude!

Another advantage of floating-point arithmetic is that most algorithms are developed in floating-point environments such as C or C++ source code or using simulation and analysis tools such as MATLAB, Mathematica, Maple, or MathCAD. Porting of the algorithms from the development environment to a fixed-point target requires additional development to convert the algorithm to a fixed-point equivalent while avoiding loss of accuracy or, in some cases, even numerical instability. A related advantage is that floating-point

algorithms, especially when using standardized formats such as IEEE P754, are more easily ported to new hardware platforms.

The gap between the characteristics of floating-point and fixed-point design is narrowing [38]. Improving design tools and available libraries of optimized fixed-point functions for many processors aid in efficiently developing fixed-point implementations of numerical algorithms. The increased capacity of chips in accordance with Moore's Law reduces the significance of the area and power impacts of larger floating-point logic circuits. For example, it is increasingly practical to implement FPGA arithmetic in floating point rather than fixed point. Fixed-point devices retain an advantage in very high-volume, cost-sensitive applications such as cell phones, while the numerical accuracy and ease of use of floating point remains important in numerically difficult but low-volume applications such as radar.

13.7 SIGNAL PROCESSOR SIZING

The preceding sections have discussed ways to estimate the arithmetic operation counts of radar signal processing algorithms and some of the implementation technologies available to host these algorithms. It remains to translate these operation counts into estimates of time to solution for a given processor.

13.7.1 Considerations in Estimating Timing

In the early days of DSP, it was possible to estimate algorithm run time by simply multiplying multiply and add operation counts by the time the processor of interest required to perform a single addition or multiplication. This approach, while useful for rough initial processor sizing, effectively assumes 100% use of the arithmetic unit. In many cases, users would "de-rate" the processor by a factor of 2× to 4× to obtain more realistic results.

Caution is also needed in relating FLOPS estimates to advertised capabilities of commercial processing hardware. Advertised computational rates usually represent peak theoretical values that may be difficult or impossible to sustain in practice, because they assume that all data and instructions are in level 1 (L1) caches, that pipelines are filled, and so forth. Fetching data and instructions from level 2 (L2) cache or from off-chip can significantly slow the processor while it waits for the data transfer.

This approach is not adequate in modern processors with complex memory hierarchies and multiple cores. For example, interprocessor communication, both intrachip (for multicore processors) and interchip, often causes actual throughputs to be much lower than advertised peak rates. Most processors large enough for modern radar algorithms require the use of multiprocessor architectures. Maximum throughput requires keeping the compute units busy as much as possible, but processor compute units frequently have to idle while waiting on data transfers to or from other processors. Designers must seek to partition the data flowgraph to divide the workload evenly among the compute units, a process knows as *load balancing*.

Another major constraint is memory access. In modern processors the time required for a floating-point operation is often much less than the time required to fetch the data for the operation and to store the results unless the data are in the L1 cache; a memory access to main memory can easily require 1,000 times as long as a floating-point math instruction. Thus, the designer must overlap computation and communication as much as

possible. This choreography of the data movement through levels of the memory hierarchy is frequently more important in determining throughout than operation counts. Indeed, the developers of the Fastest Fourier Transform in the West (FFTW) library stated in 2005 that "... there is no longer any clear connection between operation counts and FFT speed, thanks to the complexity of modern computers" [13].

A related issue in multiprocessor architectures is *scaling*. It is often implied that if a single processor achieves a certain throughput, then a multiprocessor composed of N such processors will achieve N times the throughput. While this is certainly the goal, such *linear speedups* are difficult to achieve, particularly as N increases, due both to *Amdahl's Law* [4], which describes the limit to the attainable speedup when only a portion of a program is parallelizable, and to the same issues of load balancing and interprocessor communication already cited.

The analysis needed to obtain time-to-solution estimates requires detailed processor architecture modeling and simulation and is too intricate for hand calculation. Many companies have internally developed tools to conduct these analyses. In addition, a number of university and commercial tools exist. An example of a university tool is the Ptolemy system developed at the University of California–Berkeley [39]. Examples of commercial tools include Simulink by The MathWorks, Inc., Signal Processing Designer by CoWare, Inc., and Gedae by Gedae, Inc., among many others. Typical capabilities of these tools include representation of algorithms using flowgraphs and user-extensible libraries of predefined functions; mapping of flowgraphs to multiprocessor architectures; simulation of flowgraphs and architectures; and visualization of processor loading, data flows, and signals. Some systems also include generation of C or C++ code for certain hardware processors.

13.7.2 Benchmarks

Another way to at least partially address the problem of estimating processor speed is to test actual hardware components using nontrivial benchmark codes that are representative of the applications of interest. Good benchmarks should include a mix of computation, communication, storage, and input/output operations. The computational operations should have a realistic mix of operation types for the application class of interest. For instance, a benchmark for evaluating general processing capability should include a wide variety of integer and floating-point operations with varied indexing and memory access patterns, while a benchmark for FFT hardware should concentrate on complex multiplications and additions and strided memory accesses.

General-purpose benchmark codes can be obtained publicly for a variety of purposes. For example, the Standard Performance Evaluation Corporation (SPEC) commercial benchmarks are widely used for evaluating general-purpose processors [40]. The SPEC suite also includes benchmarks for such areas as graphics and Web servers. Traditional arithmetic benchmarks such as the High Performance Computing (HPC) Challenge and many more are accessible through the BenchWeb site at the NETLIB repository [41]. Benchmarks for embedded processors at various scales are also available commercially. In addition, the defense community in the United States has developed several benchmark suites, including the High Performance Embedded Computing (HPEC) Challenge [42]. Table 13-4 provides information on a number of publicly available benchmarks with at least some applicability to radar signal processing. While these general-purpose benchmarks can be very useful, the best benchmarks for evaluating a processor architecture are

TABLE 13-4 ■ Selected Benchmark Suites

Name	Source	URL	Nature	Software/Platform	Comments
SPEC	Standard Performance Evaluation Corp.	www.spec.org	Broad spectrum embedded microprocessor evaluation	C	Commercial. Currently CPU2006.
EEMBC	Embedded Microprocessor Benchmark Consortium	www.eembc.org	Broad spectrum embedded microprocessor evaluation	ANSI C	Commercial.
BDTI Benchmark Suites	BDTI	www.bdti.com	DSP kernels, video, communication	C, assembly, test vectors	Commercial. Leans somewhat to communications.
HPCS Benchmarks	DARPA HPCS program	www.highproductivity.org/	HPC kernels and applications	HPC challenge requires MPI and BLAS	Not specific to signal processing.
HPC Challenge	University of Tennessee	www.hpcchallenge.org/	HPC parallel kernels	Requires MPI and BLAS	Public release.
HPEC challenge	MIT LL	www.ll.mit.edu/HPECchallenge/	Sensor signal processing	ANSI C (kernels only)	Public release
LINPACK (HPL)	University of Tennessee	www.netlib.org/benchmark/hpl/	Dense linear algebra	FORTRAN, Java, C Requires MPI and either BLAS or VSIPL	Public release. Basis of Top500 rankings. Incorporated into more comprehensive HPC challenge.

BDTI: Berkeley Design Technology, Inc.
DARPA: Defense Advanced Research Projects Agency
HPC: High Performance Computing
HPCS: High Productivity Computing Systems
HPEC: High Performance Embedded Computing
MIT LL: Massachusetts Institute of Technology Lincoln Laboratory

those that mimic the important characteristics of the actual application mix for the specific architecture of interest.

13.7.3 Software Tool Impacts

Even when an implementation of a flowgraph that minimizes operations has been developed, the actual efficiency of the executable code depends on the software tools used. The same source code, compiled to the same target machine with different compilers, is almost certain to have different execution times due to differences in compiler optimizations and in libraries used. The three-part series of articles [43–45] dramatically illustrates the effect of source code optimization, spatial and temporal locality, and compiler behavior on the execution time of a typical "inner loop" numerical computation. Execution time was reduced by a factor of over 30× compared with the original naïve code.

Radar signal processing algorithms rely heavily on certain core operations: spectral transforms such as FFTs, discrete cosine transforms (DCTs), and wavelet transforms; FIR filters; and linear algebraic calculations ranging from simple matrix-vector operations to more complex operations such as Q-R decomposition (QRD), singular value decomposition (SVD), and eigenvalue/eigenvector decomposition. Software developers rely on libraries to provide these critical functions. For instance, the BLAS – LINPACK – LAPACK – ScaLAPACK series of libraries provides versions of common linear algebra manipulations optimized for different classes of machines; information and code for all of these are available at the Netlib repository [46]. Numerous FFT libraries exist. VSIPL [26] and VSIPL++ [27] provide a portable library specification that includes a selection of the most common unary, binary, FFT, filtering, linear algebraic, and related signal processing operations.

Many vendors provide implementations of libraries optimized for their specific processors. One example is Intel's Math Kernel Library (MKL) [47], which offers the various classes of routines previously described along with statistics functions and sparse solvers in a package optimized for Intel microprocessors. A recent trend in signal processing and scientific libraries is the development of *autotuning* libraries. These are open source libraries that use deep mathematical and practical knowledge of the structure of a class of algorithms to evaluate many different implementations of a specific function on a specific target machine to experimentally find the most efficient code. The autotuning approach can provide very good performance over a wide range of target machines, often besting vendor-specific libraries. The FFTW library uses this approach for FFTs [48]. ATLAS takes a similar approach to the BLAS and some LAPACK linear algebra routines [49].

13.7.4 Data Rates

Not all radar systems need to perform all of the signal processing in an *onboard processor* colocated with the radar sensor, as has been implied so far. An alternative is to digitize the receiver outputs and transmit them to another location for subsequent processing, either using real-time communication links or by storing the data for later processing. Such *offboard* processing might be appropriate if significant latency in obtaining the final data products can be tolerated and the size of the onboard processor is severely limited by constraints of size, weight, or power. As an example, satellite systems and small unattended autonomous vehicles (UAVs) usually have very tight payload budgets. Satellite SARs that generate imagery for non-real-time uses such as cartography and earth resources

monitoring generally downlink the data to a ground station, where it can be processed in a facility with much greater resources and on whatever timeline is needed. Alternatively, the data may be cross-linked to a communications satellite for further distribution. In contrast, an airborne tactical SAR used for targeting must generate imagery and display it in the cockpit in real time. This system must perform all of the required processing onboard and with minimum latency.

The data rates at the output of a radar receiver can be very high. Consider a SAR system. Using an approach similar to that of section 13.4.2, an estimate can be developed for the data rate at the receiver output. The data rate DR_{in} in bits per second can be expressed as the product of three factors: the number of receiver samples per pulse N_{sp}; the pulse repetition frequency, PRF; and the number of bits per sample N_b:

$$DR_{in} = 2N_{sp}N_bPRF \text{ bps} \tag{13.11}$$

The factor of two results from assuming a coherent receiver so that both the in-phase and quadrature (I and Q) channels must be sampled. N_{sp} in turn is the product of the fast time sampling rate and the time duration T_f of the sampling window. The fast time sampling rate is determined by the waveform bandwidth B and will normally be about B samples/sec; thus

$$N_{sp} = T_f B \text{ samples} \tag{13.12}$$

T_f must be less than the pulse repetition interval (PRI) to avoid range ambiguities. Consequently, $T_fPRF \leq 1$, and, using $\Delta R = c/2B$,

$$DR_{in} \leq 2N_bB \quad \Rightarrow \quad DR_{in} \sim O\left(\frac{cN_b}{\Delta R}\right) \text{bps} \tag{13.13}$$

Taking the SIR-C example again with $\Delta R = 25$ m and assuming eight bits per sample produces a rate of 96 Mbps. A radar with a range resolution of 10 m would increase this to 240 Mbps; 1 m range resolution would raise it to 2.4 Gbps.

Note that equation (13.13) applies to each coherent channel. Thus, a standard three-channel monopulse antenna would require three times this rate to capture the sum and both difference channels. A phased array system used for imaging and including adaptive processing for interference rejection can be particularly stressing. For example, a system having six-bit A/D converters, 10 m range resolution, and six phase centers will have a raw data rate of 1.08 Gbps.

The data rate at the output of a SAR processor is typically lower than at the input to the processor by a modest factor. The pixel rate was found in equation (13.4); multiplying by the number of bits per pixel N_p gives the bit rate at the output of the image formation:

$$DR_{out} \sim O\left(\frac{cN_p}{2\Delta R}\right) \text{bps} \tag{13.14}$$

Equation (13.14) is of the same form as equation (13.13), just reduced by a factor of 2. This result assumes the range swath is at or near its maximum, the unambiguous range. Note that since image pixels are real, there is no need to double this rate to account for I and Q channels. On the other hand, a system doing interferometric SAR (IFSAR) does retain the pixel phase. A "one-pass" IFSAR system would have four times this output rate: one doubling for I and Q channels and another for the fact that two SAR images are required for IFSAR.

Images are often represented at eight bits per pixel in an uncompressed format. Assuming this precision, equation (13.14) becomes

$$DR_{out} \sim O\left(\frac{4c}{\Delta R}\right) \text{bps} \tag{13.15}$$

The resulting output bit rate is 120 Mbps at 10 m range resolution, 400 Mbps at 3 m, and 1.2 Gbps at 1 m.

Data compression can be applied to SAR data or imagery. A variety of techniques are used, depending in part on the stage of processing at which the compression is applied [50,51]. Block adaptive quantization is common on raw SAR data; this technique should be applicable to raw data for other wideband radar systems as well, such as STAP and GMTI systems. Vector quantization has been investigated for partially processed data. Wavelet and trellis coding have been applied to complex SAR imagery, with compression factors on the order of 4 for the phase but up to 64 on the magnitude. Standard image compression algorithms such as JPEG can be used on final detected imagery, as can wavelet and trellis coding; savings by factors of 4 to 8 have been achieved. It seems safe to say that reductions of at least 4× in data rate are achievable using data compression, sometimes much more.

In contrast to the SAR example, some radar signal processing systems have output date rates significantly lower than the input data rates. Consider again the STAP-GMTI system of Figures 13-4 and 13-5. The input consists of six channels (one for each phase center) sampled at 125 Msamples/second. These data are real, not complex, because digital I/Q formation will be used. Assuming eight bits per sample the input data rate will be 6 Gbps. After the subband filtering, there are 16 complex subband channels at a 10.42 Msample/second rate. Still assuming eight bits, this lowers the data rate to 2.7 Gbps. Subsequent stages will tend to have similar data rates unless the processing is structured to reduce the amount of data, for example, by processing only some subbands or applying a reduced-dimension STAP algorithm to a subset of the Doppler filters. In practice, the data rate at the input to the detector will be between one and two orders of magnitude below that at the radar input, in this case between 60 and 600 Mbps.

At the output of the detector the data rate will likely be lower because it will consist only of a list of threshold crossings with associated range/angle/Doppler coordinates and other metadata. The actual data rate will be scenario dependent. In a ground-to-air or air-to-air system, there may be only a few tens of targets per CPI, sometimes few or none. However, an air- or space-to-ground GMTI system may detect thousands of targets per CPI.

13.7.5 Onboard versus Offboard Processing

An example of a data link used by the space shuttle radars is the National Aeronautic and Space Administration's (NASA) Tracking and Data Relay Satellite System (TDRSS). This system provides channels with data rates of 80 to 800 Mbps. The U.S. Military's Common Data Link (CDL) provides on the order of 250 Mbps per channel but also allows multiple channels to be combined to create links approaching 1 Gbps. While optical communication links under development may raise achievable rates by one to two orders of magnitude, current technology appears limited to link rates of a few hundred Mbps to, at most, about 1 Gbps.

Figure 13-4 showed the growth of the cumulative computational load for a wideband STAP-GMTI signal processor as each successive stage was executed. If the radar platform is tightly constrained in size, weight, and power, as, for instance, on a satellite or small UAV, it may not be possible to host a processor of adequate capacity to perform all of the signal processing onboard. In systems that do not require that all signal processing be completed in real time, it may simply be unnecessary to complete the processing onboard. There may, therefore, be a need for an engineering trade-off between onboard processor capacity and data link capacity. For instance, if the platform can host a 200 GFLOPS processor, it can perform all of the processing of Figure 13-4 onboard, and only a low-rate data link will be needed to downlink the detection data. If, on the other hand, the platform can host only a 160 GFLOPS processor, then the data must be downlinked at the output of the pulse compression stage and the processing completed in a ground station or other facility. For this to be possible, a data link must be available with adequate capacity for the data rate at the pulse compression output. Assuming this data rate is on the order of 1 Gbps and that appropriate data compression can reduce it by another factor of 4, the data could be downlinked with a TDRSS or CDL data link such as already discussed.

If the data rates are such that this trade-off is not possible, the system design must be revisited until a realizable combination of onboard signal processing capacity and downlink bandwidth is found. If a larger processor can be hosted, the downlink can be deferred to a later processing stage. If a large enough processor to perform the entire processing chain can be hosted, a more narrowband data link will suffice for delivery of the final image products.

13.8 FURTHER READING

The recent edited volume by Martinez, Bond, and Vai [5] gives an excellent overview of the full range of hardware, software, and system issues in high-performance embedded computing, with an emphasis on sensing applications, including specifically advanced radar techniques.

13.9 REFERENCES

[1] Skolnik, M.I., *Introduction to Radar Systems*, 3d ed., McGraw-Hill, New York, 2001.

[2] Cutrona, L.J., Leith, E.N., Porcello, L.J., and Vivian, W.E., "On the Application of Coherent Optical Processing Techniques to Synthetic-Aperture Radar," *Proceedings of the IEEE*, vol. 54, no. 8, pp. 1026–1032, August 1966.

[3] Oppenheim, A.V., and Schafer, R.W., *Discrete-Time Signal Processing*, 3d ed., Prentice-Hall, Englewood Cliffs, NJ, 2009.

[4] Goldberg, D., "Computer Arithmetic," Appendix A in *Computer Architecture: A Quantitative Approach*, 3d ed., Ed. J.L. Hennessy and D.A. Patterson, Morgan Kaufmann, San Francisco, 2002.

[5] Martinez, D.R., Bond, R.A., and Vai, M.M., Eds., *High Performance Embedded Computing Handbook: A Systems Perspective*, CRC Press, Boca Raton, FL, 2008.

[6] Elachi, C., *Spaceborne Radar Remote Sensing: Applications and Techniques*, IEEE Press, New York, 1988.

[7] Richards, M.A., *Fundamentals of Radar Signal Processing*, McGraw-Hill, New York, 2005.

[8] Cumming, I.G., and Wong, F.H., *Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation*, Artech House, Boston, MA, 2005.

[9] Curlander, J.C., and McDonough, R.N., *Synthetic Aperture Radar: Systems and Signal Processing*, Wiley, New York, 1991.

[10] Cain, K.C., Torres, J.A., and Williams, R.T., "RT_STAP: Real-Time Space-Time Adaptive Processing Benchmark," MITRE Technical Report MTR 96B0000021, February 1997.

[11] Ward, J., "Space-Time Adaptive Processing for Airborne Radar," MIT Lincoln Laboratory Technical Report 1015, pp. 75–77, December 13, 1994.

[12] Cooley, J.W., and Tukey, J.W., "An Algorithm for the Machine Computation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, pp. 297–301, April 1965.

[13] Frigo, M., and Johnson, S.G., "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[14] Merkel, K.G., and Wilson, A.L., "A Survey of High Performance Analog-to-Digital Converters for Defense Space Applications," *Proceedings of the 2003 IEEE Aerospace Conference,* vol. 5, pp. 2415–2427, March 8–15, 2003.

[15] Aziz, P.M., Sorensen, H.V., and van der Spiegel, J., "An Overview of Sigma-Delta Converters" IEEE *Signal Processing Magazine*, vol. 13, no.1, pp. 61–84, January 1996.

[16] Walden, R.H., "Analog-to-Digital Conversion in the Early 21st Century," in *Encyclopedia of Computer Science and Engineering*, Ed. B. Wah, John Wiley & Sons, Inc., New York, 2008.

[17] Walden, R.H., "Analog-to-Digital Converter Survey and Analysis," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, pp. 539–550, April 1999.

[18] Tessier, R., and Burleson, W., "Reconfigurable Computing for Digital Signal Processing: A Survey," *Journal of VLSI Signal Processing,* vol. 28, no. 7, pp. 7–27, 2001.

[19] IBM, "Cell Broadband Engine Technology." n.d. Available at http://www.ibm.com/developerworks/power/cell/docs_articles.html.

[20] Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., and Phillips, J.C., "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[21] Karasev, P.A., Campbell, D.P., and Richards, M.A., "Obtaining a 35x Speedup in 2D Phase Unwrapping Using Commodity Graphics Processors," *Proceedings of the IEEE Radar Conference 2007,* Boston, MA, pp. 574–578, 2007.

[22] Khronos Group, "OpenCL—The Open Standard for Parallel Programming of Heterogenous Systems." August 2009. Available at http://www.khronos.org/opencl/.

[23] Alford, L.D., Jr., "The Problem with Aviation COTS," *IEEE AES Systems Magazine*, vol. 16, no. 2, pp. 33–37, February 2001.

[24] Wilcock, G., Totten, T., Gleave, A., and Wilson, R., "The Application of COTS Technology in Future Modular Avionic Systems," *Electrical & Communication Engineering Journal*, vol. 13, no. 4, pp. 183–192, August 2001.

[25] Shank, S.F., Paterson, W.J., Johansson, J., and Trevito, L.M., "An Open Architecture for an Embedded Signal Processing Subsystem," *Proceedings of the IEEE 2004 Radar Conference,* pp. 26–29, April 2004.

[26] Defense Advanced Research Projects Agency (DARPA) and the U.S. Navy, "Vector Signal Image Processing Library (VSIPL)." April 23, 2009. Available at http://www.vsipl.org.

[27] Georgia Tech Research Institute, "High Performance Embedded Computing Software Initiative (HPEC-SI)." n.d. Available at http://www.hpec-si.org.

[28] University of Illinois, "Message Passing Interface (MPI) Forum." n.d. Available at http://www.mpi-forum.org.

[29] Moore, G.E., "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, April 19, 1965.

[30] ICKnowledge, LLC, "Microprocessor Trends," 2000–2008. Available at http://www.icknowledge.com/trends/uproc.html.

[31] Schaller, R.R., "Moore's Law: Past, Present, and Future," *IEEE Spectrum*, vol. 32, no. 6, pp. 52–59, June 1997.

[32] Shaw, G.A., and Richards, M.A., "Sustaining the Exponential Growth of Embedded Digital Signal Processing Capability," *Proceedings of the 2004 High Performance Embedded Computing Workshop,* MIT Lincoln Laboratory, September 28–30, 2004. Available at http://www.ll.mit.edu/HPEC/agenda04.htm.

[33] Top 500 Supercomputer Sites, "Homepage," 2000–2009. Available at http://www.top500.org.

[34] Le, C., Chan, S., Cheng, F., Fang, W., Fischman, M., Hensley, S., et al., "Onboard FPGA-Based SAR Processing for Future Spaceborne Systems," *Proceedings of the 2004 IEEE Radar Conference,* pp. 15–20, April 26–29, 2004. Available at http://hdl.handle.net/2014/37880.

[35] Song, W.S., Baranoski, E.J., and Martinez, D.R., "One Trillion Operations per Second On-Board VLSI Signal Processor for Discoverer II Space Based Radar," *IEEE 2000 Aerospace Conference Proceedings,* vol. 5, pp. 213–218, March 18–25, 2000.

[36] Ercegovec, M., and Lang, T., *Digital Arithmetic*, Morgan-Kauffman, San Francisco, 2003.

[37] IEEE. "IEEE Standard for Binary Floating-Point Arithmetic," ANSI/IEEE Standard 754-1985, August 12, 1985.

[38] Frantz, G., and Simar, R., "Comparing Fixed- and Floating-Points DSPs," Texas Instruments White Paper SPRY061, 2004. Available at http://focus.ti.com/lit/wp/spry061/spry061.pdf.

[39] The University of California at Berkeley, Electrical Engineering and Computer Science Department, "The Ptolemy Project." n.d. Available at http://ptolemy.eecs.berkeley.edu/.

[40] Standard Performance Evaluation Corporation, "Homepage," n.d. Available at http://www.spec.org.

[41] NETLIB BenchWeb, "Homepage," n.d. Available at http://www.netlib.org/benchweb/.

[42] MIT Lincoln Laboratory, "HPEC Challenge," 2001–2006. Available at http://www.ll.mit.edu/HPECchallenge/.

[43] Pancratov, C., Kurzer, J.M., Shaw, K.A., and Trawick, M.L., "Why Computer Architecture Matters," *IEEE and American Institute of Physics Computing in Science and Engineering*, vol. 10, no. 3, pp. 59–63, May–June 2008.

[44] Pancratov, C., Kurzer, J.M., Shaw, K.A., and Trawick, M.L., "Why Computer Architecture Matters: Memory Access," *IEEE and American Institute of Physics Computing in Science and Engineering*, vol. 10, no. 4, pp. 71–75, July–August 2008.

[45] Pancratov, C., Kurzer, J.M., Shaw, K.A., and Trawick, M.L., "Why Computer Architecture Matters: Thinking Through Trade-offs in Your Code," *IEEE and American Institute of Physics Computing in Science and Engineering*, vol. 10, no. 5, pp. 74–79, September–October 2008.

[46] University of Tennessee, Computer Science Dept, "Netlib Repository." n.d. Available at http://www.netlib.org.

[47] Intel, "Intel Math Kernel Library (Intel MKL) 10.2," n.d. Available at http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm.

[48] M. Frigo, S. G. Johnson, and the Massachusetts Institute of Technology, "FFTW [Fastest Fourier Transform in the West]," n.d. Available at http://www.fftw.org.

[49] SourceForge, "Automatically Tuned Linear Algebra Software (ATLAS)," n.d. Available at http://math-atlas.sourceforge.net/.

[50] El Boustani, A., Brunham, K., and Kinsner, W., "A Review of Current Raw SAR Data Compression Techniques," *2001 Canadian Conference on Electrical and Computer Engineering,* vol. 2, pp. 925–930, May 13–16, 2001.

[51] Francaschetti, G., and Lanari, R, *Synthetic Aperture Radar Processing*, CRC Press, Boca Raton, FL, 1999.

13.10 PROBLEMS

1. Show that the number of real FLOPs required to compute all N points of a radix-2 FFT (N is assumed to be a power of 2) is $5N \log_2 N$.

2. Suppose a system convolves a 1,000-point complex-valued input sequence with a 40-point complex-valued filter impulse response. What is the length of the filter output sequence? How many complex multiplications are required to compute the complete output? Check your answer against Figure 13-7.

3. Repeat problem 2, but assume that the filtering is done using fast convolution, that is, using FFTs. Assume that the FFT length is required to be a power of 2 and that $H[k]$, the FFT of the impulse response $h[n]$, has been precomputed so that its multiplications need not be counted. Again, check your answer against Figure 13-7.

4. Now suppose the 1,000-point input vector is broken into 5 200-point segments. Each segment is filtered separately to produce a 239-point output, using fast convolutions with 256-point FFTs. Again, assume that $H[k]$ has been precomputed. What is the total number of multiplications to compute all of the required FFTs and inverse FFTs?

5. Suppose the input to a lowpass digital filter is a real-valued continuous stream of 12-bit data samples at a rate of 10 Msamples/second. The filter impulse response is 40 samples long and is also real-valued. The sample rate at the output of the filter is reduced to 5 Msamples/second due to the lowpass filtering. What is the filter input data rate in Mbps? What is the filter output data rate in Mbps if the outputs are represented as 16-bit fixed-point numbers? Repeat for the cases where the output is represented as single-precision floating point (32 bits per sample) and as double-precision floating point (64 bits per sample).

6. Consider the parametric formula for SAR resolution in section 13.4.2. If the SAR range swath is $^1/_2$ of the unambiguous range interval, how will equation (13.3) be altered?

7. In section 13.4.2, it was argued that F_p, the number of complex FLOPs per pixel, is two times the size of the cross-range data length N_{CR} (one multiply plus one add per cross-range sample). Assuming the system parameters are such that the correlation can be done using FFTs instead of a brute-force correlation calculation, calculation of the cross-range correlation would require an FFT, a vector multiply with the reference function, and an inverse FFT, all of size N_{CR}. What would be the new value of F_p in this case, and how would equation (13.6) and equation (13.7) be altered?

8. The first petaflops computer (10^{15} FLOPS), as measured by the Top 500 criterion, appeared in June 2008. The first terascale (10^{12}) computer appeared on the list in June 1997. When should we expect the first exaFLOPS (10^{18} FLOPS) computer to appear on the Top 500 list?