FY'98 Trade Study from Code Y
GSFC Earth Science Technology Office

Using COTS Components for Real-Time Processing of SAR Systems

by

Charles Le and Scott Hensley

Final Report

Interferometric Algorithms and System Analysis Group
Radar Science and Engineering Section
Jet Propulsion Laboratory

Summary of Research Activities

The research activities are divided into three stages, with each stage lasting approximately one month. In the first stage [1], we discussed the design trade-off for real-time SAR processing. The objectives are to maximize the processing density (MFLOPS/volume, MFLOPS/weight, MFLOPS/watt), to minimize the hardware cost (MFLOPS/dollar), to minimize the software development cost, and to maximize software generality. As an example, we briefly described the Mercury's RACEway multicomputer as a low cost, high-performance, embedded heterogeneous message-passing multicomputer system.

In the second stage [2], we presented a procedure to divide the SAR signal processing into pipelined parallel steps that can be performed on a parallel computer. We also estimated the throughput, memory, and I/O bandwidth requirements using the NASA/JPL GeoSAR system. Using again the Mercury's RACEway multicomputer as an example, we showed how to configure the hardware for SAR signal processing.

Parallel programming was the topic in this final stage [3]. First, we gave an overview of the standard multicomputing platforms and parallel programming models. We then compared and contrasted "scientific" versus "real-time" processing/programming. Next, we showed the hardware/software co-design approach for SAR signal processing. Then, we discussed the current issues in parallel programming. Finally, we described a typical software architecture for parallel programming of SAR signal processing.

Part I: The Design Trade-Off for Real-Time SAR Processing

1. Introduction to Parallel Processing

Multiprocessing is necessary to meet the real-time demands, and/or high I/O data rate, and/or computationally intensive algorithms, typically found in radar/sonar systems. The four key issues in large multiprocessor systems are architecture, communication efficiency, reliability, and ease of use [4]. Usually, parallel architectures are characterized by two general classes: shared-memory multiprocessors and message-passing multicomputers [5]. The main differences lie in the implementation of memory sharing and interprocessor communication. In a shared-memory multiprocessor configuration, all processors within the system have equal access to a shared-memory address space. The interprocessor communication is achieved by modifying data in the shared-memory address space. On the other hand, in the message-passing multicomputer architecture, each compute node (CN) consists of a processor and its own local memory, unshared with all other CN's. CN's are connected with each other via a common high-speed data communication fabric or interconnection network. Processors communicate with each other by passing messages through this interconnection network.

2. A Typical Example: The Mercury's RACEway Multicomputer

In recent years, Mercury Computer Systems, Inc. (MCSI) has emerged as one of the leaders in the development and manufacturing of lower cost, high-performance, embedded heterogeneous message-passing multicomputer systems. These parallel systems address complex real-time applications requiring tremendous computational throughputs (such as radar/sonar processing, medical imaging, etc.). Its main competitors include, but not limited to, Alacron, CSPI, and Sky Computers [6]. However, MCSI's hardware systems dominate the radar/sonar market in both military [7] [8] [9] and commercial [10] [11] [12] sectors. MCSI's RACE multicomputer provides a foundation for parallel systems and offers a set of building blocks that provide upward scalability [13] [14]. An example of high-level heterogeneous RACE multicomputer is shown in Fig. 1. The system consists of programmable digital signal processors (DSPs), such as the Analog Devices' SHARC chips; general-purpose reduced-instruction-set-computing (RISC) microprocessors, such as the Intel's i860, Motorola/IBM's PowerPC; application-specific-integrated-circuit devices, such as Xilinx's field-programmable-gated-array (FPGA) XC4000 series; I/O ports; and a network interface all connected via the RACEway interconnection network. The RACEway interconnection network is the framework used to provide high-performance communications among the interconnected processors and devices. Each node in the multicomputer interfaces the network through the RACE network chip (see Fig. 2). The network chip is the key to the high performance and low cost of the RACE system. The network chip is a crossbar with 6 I/O channels consisting of 32 bit datapaths, and 8 bits of control and clocking. Each channel is bi-directional. This device can handle three simultaneous transfers of 160 MB/s for a total of 489 MB/s, and can broadcast to 5 ports at 640 MB/s. The RACE

network can be configured into a wide variety of network topologies; however, the most common configuration is a fat-tree architecture [13].

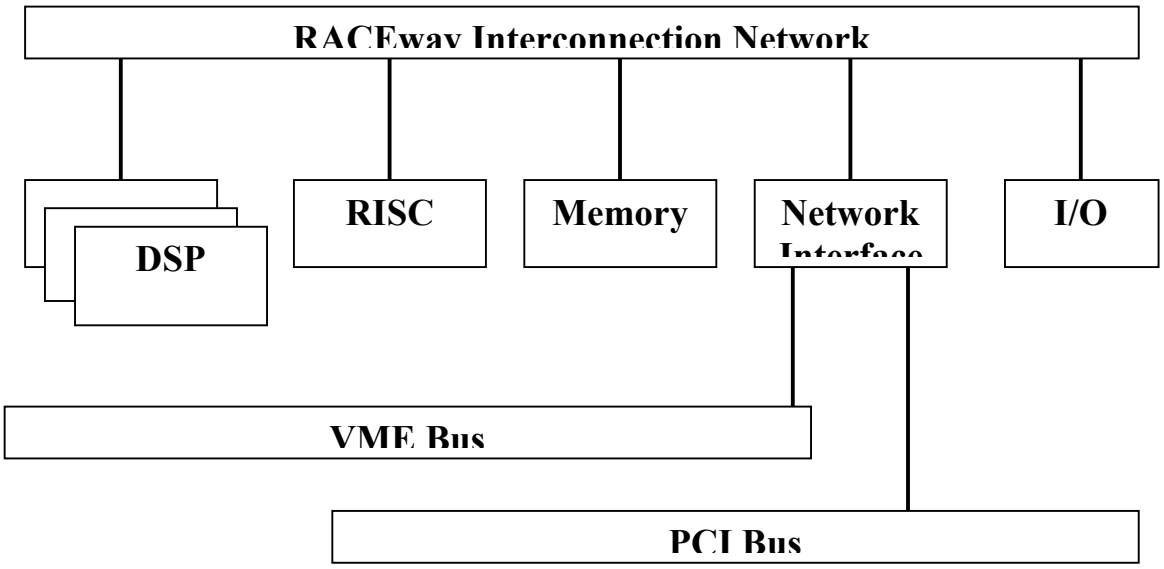

Figure 1: The RACE Multicomputer [13]

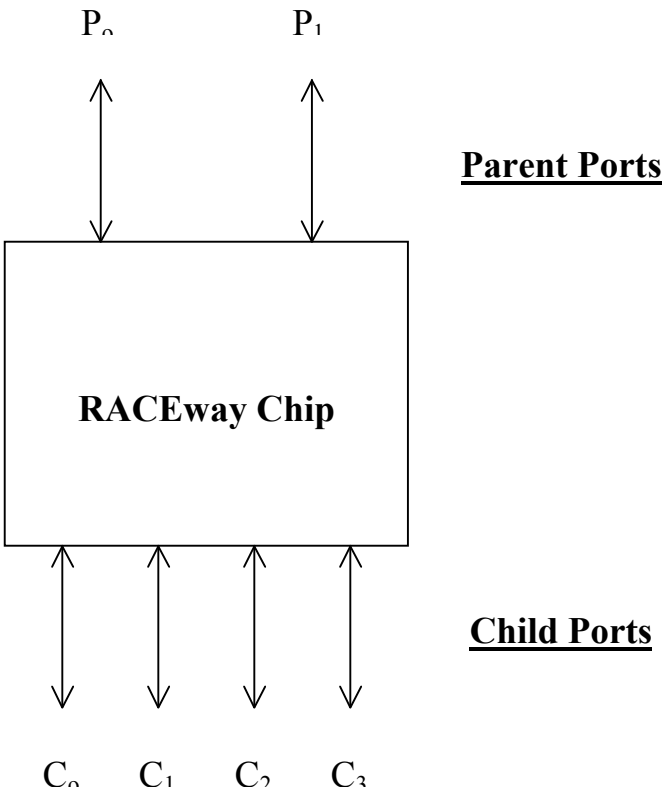

Figure 2: The RACE network chip [13]

3. Optimization Criteria for High-Performance Embedded Real-Time Systems

As stated in Section 1, the four principal design criteria are

- Maximizing processing density (MFLOPS/volume, MFLOPS/weight, and MFLOPS/watt).
- Minimizing hardware cost.
- Minimizing software development cost (ease of programming).
- Maximizing generality (easy reconfigurability and software portability).

These conditions are usually in conflict with each other, and thus cannot be all met at the same time. The system developer has to weigh each condition differently during the course of his development process. The following subsections will discuss the design trade-off imposed by these criteria.

3.1. Processing Density

No matter how fast a single processor is, typical radar/sonar systems need multiple processors working together to satisfy the throughput requirements. If the computing unit is carried in a limited enclosure (such as aircraft or spacecraft), or the application has an embedded nature, the processing density is an important consideration.

Usually, DSPs are the processors of choice for embedded *vector* or *image* processing-oriented applications (such as FFT, FIR filters, IIR filters, multirate filters, adaptive filters, pulse compression filters, etc.), where *high processing density* (i.e. MFLOPS/m^3, MFLOPS/watt) is a primary consideration, or for other similar algorithms that have a *high* data-to-computation ratio [15]. In general, the processing density of DSPs is usually 5 to 10 times that of general-purpose microprocessors. For example, up to 12 SHARCs can be mounted on one 6U VME board as compared to only 4 i860s or 4 PowerPCs. On the other hand, RISC processors are suitable for high-performance *scalar* processing applications that involve the execution of *compiled* C/FORTRAN codes with *low* data-to-computation ratio. Also, RISC processors are more applicable where ease-of-use is an important consideration. And finally, application-specific-integrated-circuits (ASIC) devices (such as FFT, data compression, or reconfigurable processors) can offer an order of magnitude throughput improvement for a specific algorithm compared to programmable processors.

Hence, heterogeneous multicomputing, where more than one kind of processor type are used, is preferred to optimally match different types of chips to different computation stages of the application. A heterogeneous configuration that can leverage DSPs, RISC microprocessors, and specialized processors optimizes the dataflow throughput and leads to fewer overall processors. Smaller systems are not only less expensive because they have fewer processors, but they also requires less hardware infrastructure since they are easier to package, power, cool, and maintain. Optimal configuration of a heterogeneous multicomputing system for minimum SWAP is the main subject of the second stage of the research project.

3.2. Hardware Cost

Driven by profit, business decision places strong emphasis on this criterion. However, it is less important during the development stage or in situation where large production volume is not an issue. In a typical research establishment, system reliability may be more relevant. Thus, the optimal system with respect to hardware cost depends only on the budget for a specific project. Fortunately, use of COTS-based components (almost) always ensure parts availability in a competitive market. Also, with a heterogeneous system, system cost can be substantially reduced by matching the processor type to the processing requirement.

3.3. Software Cost

This is the most important design criterion when developing a prototype or proof-of-concept system. In these situations, the applications require only one working prototype and a spare copy of such system. Most of the time will be spent on software developing. Minimizing software cost leads to choosing the processors that are easy to program, or have strong third-party support [16], or benefited from academia and federally funded research [17].

In the past, general-purpose microprocessors were the easiest to program due to their available high-level language (such as FORTRAN or C) compiler. Programming DSPs, on the other hand, was much harder, primarily through libraries and excruciatingly tailored hand code. However, progress in the compiler and architecture has made it possible for the DSP's applications to be written in a high-level language, and to allow the compiler to do the fine-grain scheduling necessary to achieve available instruction level parallelism [18] [19]. All of the currently popular DSPs possess at least a C compiler provided by their manufacturer. And most of DSP board vendors often include third-party high-level language compiler and real-time-operating-system (RTOS) for their boards.

If the application calls for a heterogeneous architecture, then interprocessor communication becomes important. Here, an open software infrastructure is necessary to enable seamless integration of different processor types, operating systems, and application programming interfaces, allowing rapid response to new technologies and application-specific software requirements. Development tools such as debuggers and application building tools are also of importance.

3.4. System Generality

System generality refers to the openness of the hardware architecture, and the ease with which different applications can be implemented on a given multicomputer. Open hardware architecture reduces life-cycle costs by providing a standard platform for future upgrades. Open and standard interfaces permit third-parties and end-users to add their new and unique I/O and specialized processing elements for a wide range of application requirements. COTS-based open-architecture multicomper systems can

rapidly adapt and evolve to new technology and requirements. The marketplace will provide the best technology available on an open (and standard) system, thus extending the useful life cycle of the systems for many years.

As already discussed in Section 4.3, an open software architecture is also preferred so that software modifications or upgrades can be easily ported to a given hardware system.

4. Summary

In this first part, we have presented and discussed the design trade-off one must face when designing a real-time computing platform for SAR signal processing. Past approach centered on choosing "the right processor". Advanced technology made it possible for open heterogeneous COTS-based multicomputing systems that allow several types of processors to work in parallel. The main idea is to optimally match each type of processors to the computation stages where it can provide the best throughput. Coupled with an open software infrastructure, these systems offer significant advantages and reduce overall system cost.

Part II: Pipelined Parallel Architecture for SAR Processor

1. Introduction

In this part, we present a procedure to divide the SAR signal processing into pipelined parallel steps that can be performed on a parallel computer. We also estimate the throughput, memory, and I/O bandwidth requirements using the NASA/JPL GeoSAR system [20] [21] [22]. Using again the Mercury's RACEway multicomputer [14] as an example, we show how to configure the hardware for SAR signal processing.

SAR is a radar imaging technique aiming at providing two-dimensional high-quality high-resolution images for terrain mapping at target imaging. Applications of SAR in the military consist of intelligent gathering, battle field reconnaissance, land mine detection, and weapon guidance. Civilian applications include topographic mapping, surface deformation monitoring and analysis, oil spill monitoring, ocean and sea ice characterization and tracking, agriculture and urban classification and assessment, land use monitoring, planetary exploration, etc.

One of the many challenges in SAR is the huge amount of computations required in several SAR correlation algorithms, such as the range-Doppler algorithm, step-transform processing, deramp compression processing, polar processing, ω-k algorithm, etc. [23]. This reference also gives an overview of many SAR correlation architectures that have been considered or implemented in the past. One should however note that these hardware platforms have been designed in the mid- or late eighties. Since then, dramatic advances in VLSI technology have made possible the stringent requirements that seemed insurmountable only ten years ago. Also, substantial investment in advanced reconnaissance technology from the Department of Defense (DoD) and the Defense Advanced Research Projects Agency (DARPA) through such programs as COTS [24, 25] and RASSP [17], have led to many radar processing systems [7, 11, 12, 26, 27] that provide high-performance, while offering small size, light weight, and low cost for expendable aircraft mission [9, 28]. It is our intention, in this research project, to benefit from these previous investments and to consider the resulting advanced technology for civilian use, such as in the GeoSAR program

2. General Description of SAR Signal Processing

GeoSAR [21] is a congressionally-mandated DARPA-funded project to develop a dual frequency airborne interferometric mapping radar. The overall goals of the project are to

- develop precision foliage penetration mapping technology based upon dual frequency interferometric radar
- provide military and civilian users with significant increase in mapping technology
- produce true ground surface digital elevation models suitable for military and civilian applications

The project duration is 3 years, starting November 1, 1996. The system is expected to be operational in November 1999. Table 1 gives the definitions and values of the radar parameters and Table 2 shows the processor parameters.

Symbol	Description	P-Band	X-Band
λ	Radar wavelength at bandwidth center	.86 m	.031 m
B_d	Radar bandwidth	160 MHz	160 MHz
f_s	Complex sampling frequency	180 MHz	180 MHz
τ	Pulse length	40 usec	40 usec
T_{win}	length of data collection window	110 usec	110 usec
V	Platform velociyy	215 m/s	215 m/s
L_a	Antenna length	1.5 m	1.5 m
f_p	Pulse repetition frequency	315 MHz	315 MHz

Table 1: Radar parameters

Symbol	Description	P-Band	X-Band
$N_s = T_{win}\ f_s$	Number of complex samples per range line	19800	19800
$N_{rv} = N_s - M_r$	Number of valid range samples	12600	12600
$M_r = \tau f_s$	Length of chirp reference function	7200	7200
P_r	Range FFT section length	Software parameter	Software parameter
$N_r = P_r + M_r$	Range FFT length		
Il	Interpolator length	8	8
M_a	Length of azimuth reference function	16384	1024
$P_a = M_a$	Azimuth FFT section length	16384	1024
$N_a = P_a + M_a$	Azimuth FFT length	32768	2048
$Qs = N_s\ f_p$	Data rate	6.3 Msamps	6.3 Msamps

Table 2: Processor parameters

Figure 1 shows the block diagram of the GeoSAR range-Doppler signal processing and Figure 2 gives the numbers of floating-point operations per input sample at each processing stage

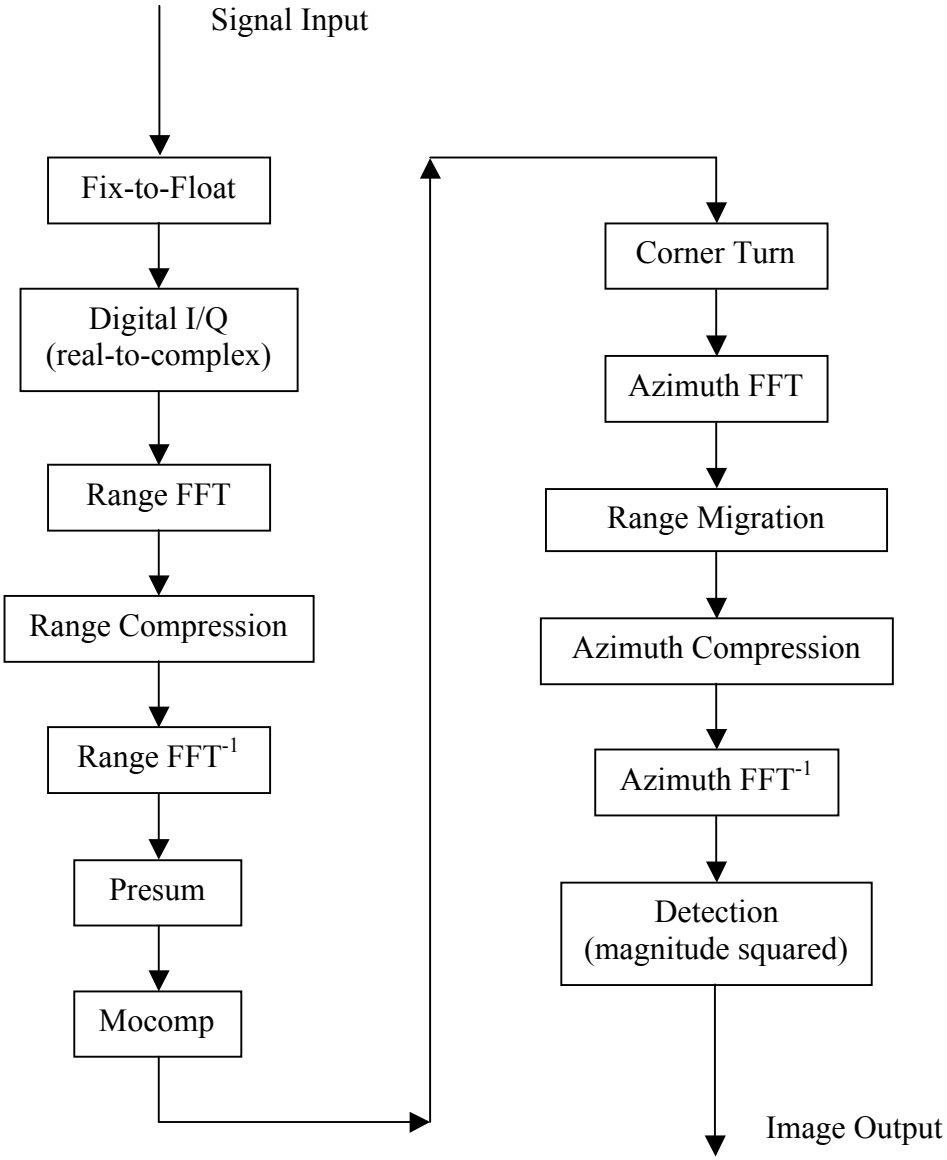

Figure 1: SAR signal processing

$$\text{Range FFT} = 5\frac{N_r}{P_r}\log_2 N_r$$

$$\text{Range Ref. Func. Mult} = 6\frac{N_r}{P_r}$$

$$\text{Range FFT}^{-1} = 5\frac{N_r}{P_r}\log_2 N_r$$

$$\text{Presum} = 6\frac{N_{rv} I_l}{N_s}$$

$$\text{Mocomp} = 6\frac{N_{rv} I_l}{N_s}$$

$$\text{Corner Turn} = 10\frac{N_{rv}}{N_s}$$

$$\text{Azimuth FFT} = \left[5\frac{N_a}{P_a}\log_2 N_a\right]\frac{N_{rv}}{N_s}$$

$$\text{Range Migration} = 6\frac{N_a}{P_a}\frac{N_{rv}}{N_s} I_l$$

$$\text{Azimuth Ref. Func. Mult.} = 6\frac{N_a}{P_a}\frac{N_{rv}}{N_s}$$

$$\text{Azimuth FFT}^{-1} = \left[5\frac{N_a}{P_a}\log_2 N_a\right]\frac{N_{rv}}{N_s}$$

$$\text{Detection} = 3\frac{N_{rv}}{N_s}$$

Figure 2: Number of FLOPs per input sample

3. Parallel of SAR Signal Processing

As shown in Figs. 1 and 2, the range-Doppler SAR algorithm can be separated into sequential stages. Hence, the processing can be pipelined to improve the throughput. In addition, multiprocessors can be employed in each stage to obtain a further throughput increase. Fig. 3 shows a possible parallel mechanism for SAR processing.

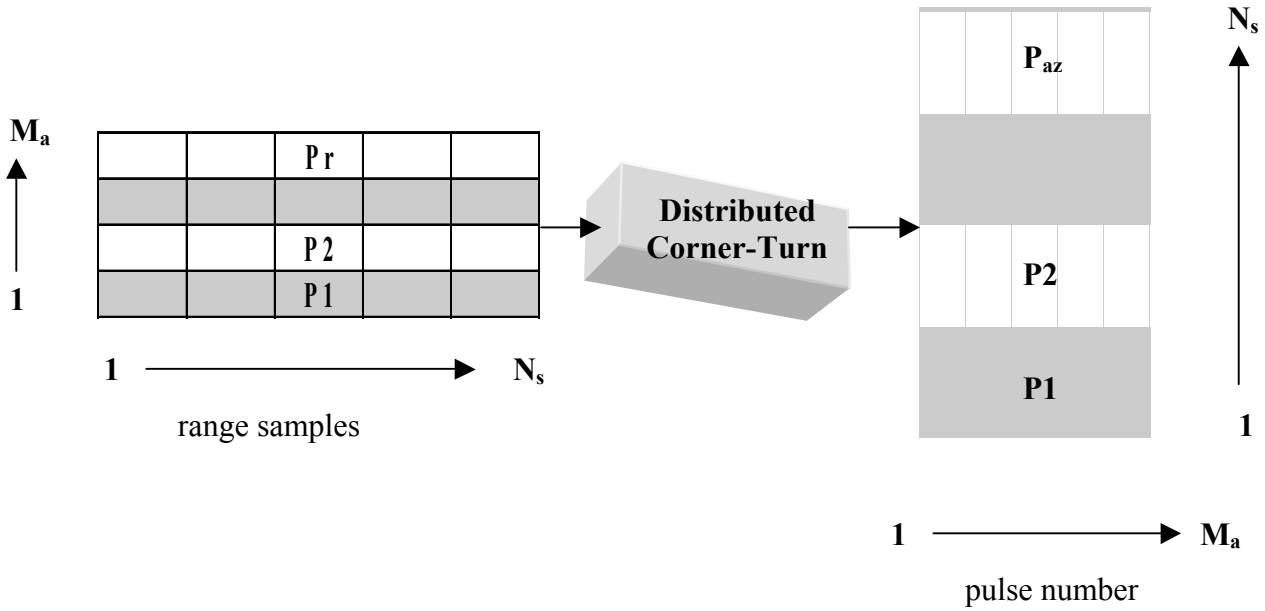

Figure 3: Parallelization of SAR Processing

The range compression can be performed independently of the pulse number. In this first stage of the pipeline chain, multiple processors (P_1, P_2, … , P_r) can work in parallel to perform the range compression, with each processor responsible for a set of pulse returns. Similarly, the azimuth compression of different range samples can be performed independently of the range cells. Hence, another set of multiple processors (P_1, P_2, …, P_{az}), distributed across the range samples, could be assigned to work in parallel. Between the two compressions in the range and cross-range directions, data needs to be "corner-turned". In this intermediate step, data are transferred from range buffer memory to azimuth buffer memory, and then (matrix) transposed.

4. Processing Requirement

4.1. Throughput and Memory Trade-Off for Overlap-Save Convolution

Because of the long data record, the convolution/correlation operations are best performed with the overlap-save technique. Given the chirp reference length M_r, there exists an optimum FFT section length P_r to minimize the FFT computational load.

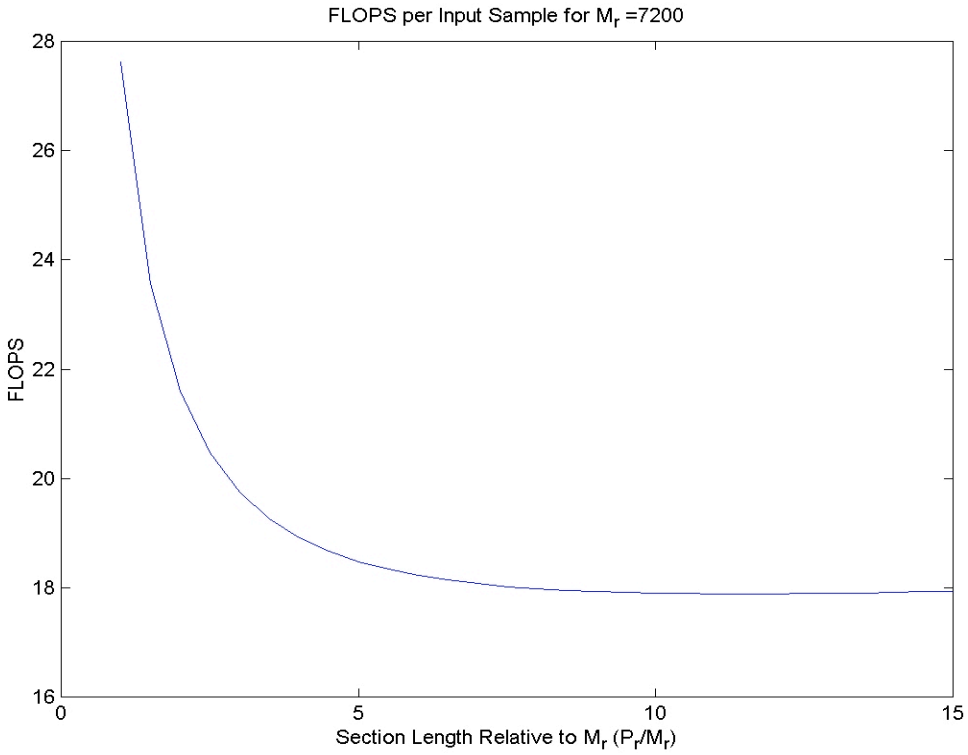

Figure 3: FFT computation load as function of section length

Figure 3 shows the FLOP per input sample as a function of the section length P_r for the FFT operation. Since the minimum region is very flat, it is better to choose the smallest section length P_r in this region to minimize the amount of memory buffer required. This is very important for azimuth processing when it is shown that the azimuth memory buffer dominates the total system required memory.

4.2. Total Computational Load

The computational load at each stage is the number of FLOP required for that stage multiplied by the corresponding sample rate. The SAR signal processing can be divided into three main stages: range compression, presumming and motion compensation, and

corner turn and azimuth compression. The sample rate during the range compression step is given by

$$Q_r = N_s f_p$$

After pulse compression, there are only N_{rv} good samples. Hence, the sample rate for the subsequent computations is

$$Q_{rv} = N_{rv} f_p$$

The computational loads for the three stages are

$$C_r = Q_r \frac{N_r}{P_r}(10\log_2 N_r + 6)$$

$$C_{mc} = 12 Q_{rv} \frac{N_{rv}}{N_s} I_l$$

$$C_{az} = Q_{rv} \frac{N_{rv}}{N_s} \left\lceil \frac{N_a}{P_a} \right\rceil (10\log_2 N_a + 6 + 6I_l) + 13 \left\lceil \frac{N_a}{P_a} \right\rceil$$

We recall from Tabs. 1 and 2 that, except for P_r which is the software parameter all the remaining parameters in the above equations are actual radar parameters, fixed at the radar system design level. As discussed in the previous section, P_r is the parameter that controls the trade-off between throughput and memory requirements. Since the lengths of the range and azimuth reference functions are long for GeoSAR system (in the order of 8K to 16K), a 50% overlap in the overlap-save convolution operation (corresponding to $P_r = M_r$ and $P_a = M_a$) is chosen to minimize the amount of memory required.

The number of samples per pulse N_s and the radar PRF f_p are inversely proportional to the range resolution Δx_r and the azimuth resolution Δx_a, respectively

$$N_s = T_{win} f_s = g_r \frac{2\Delta R}{c} B = g_r \frac{\Delta R}{\Delta x_r}$$

$$f_p = g_a B_d = g_a \frac{2V}{L_a} = g_a \frac{V}{\Delta x_a}$$

where g_r and g_a are the range and azimuth oversampling factors, respectively. Hence, the processing load, in general, increases inversely with the **square** of the image resolution.

4.3. Memory Requirements

Each range compression processing element (PE) requires both an input and output double buffer, each of length $2N_s$ complex samples. If there are P_r range processors, then the total range memory required is $4P_rN_s$ complex samples. Azimuth processing requires a corner-turning double buffer in addition to the output image buffer for a total of $(2+1)N_{rv}M_a = 3\ N_{rv}M_a$. Since the azimuth reference length M_a is on the order of 1,000 (X-band) to 20,000 (P-band), and the number of range processors is much smaller (in the order of 10 to 100), the total memory requirement are dominated by the azimuth processing. Assuming 8 bytes per complex sample, the total memory required is

$$M = 8(4N_S P_r + 3N_{rv}M_a) \approx 24N_{rv}M_a$$

Since

$$N_{rv} = N_s \cdot M_r \quad / \Delta x_r$$

$$M_a = f_p \frac{\Delta R}{L_a V} \quad / \Delta x_a$$

the total memory required is inversely proportional to the **cube** of the resolution.

5. Mapping SAR Signal Processing into a COTS-Based Parallel Computer: An Example with GeoSAR using Mercury's SHARC DSP Daughter Cards

Mapping a signal processing application into a COTS-based computer consists of determining the total throughput (GFLOPS), the number of processors, the amount of memory (MB), and inter-processor data communication bandwidth (MB).

6.1. Total Throughput and Number of Processors Required

For the GeoSAR radar and processor parameters given in Figures 1 and 2, the throughputs measured in GFLOPS (number of giga floating-point operations per second) are shown in Table 3

	Range Compression	Presum/ Mocomp	Azimuth Compression	Total Throughput
X-band	1.8	0.25	0.86	2.91
P-band	1.8	0.25	1.07	3.12

Table 3: GeoSAR signal processing throughput

The total throughput gives only an estimate of the total computations required for the application. However, not all FLOPS are created equal. This means that a given processor performs some operations faster than other operations. For example, Table 4 shows the throughput of a DSP board consisting of Analog Devices' SHARC chips, for the operations relevant to SAR signal processing [28]

Operation	Throughput (MFLOPS)
Magnitude Squared	26
Complex Multiplication	27
FIR Filter	45
Fixed-to-Float / Float-to-Fixed	80
Corner Turn	80
Fast Convolution	94

Table 4: Equivalent FLOPS for SHARC-based DSP

To determine the number of PEs required, we need to

1. determine the equivalent throughput of the selected PE type (SHARC, TMS320Cx, PowerPC, i860, …) for each operation, i.e. the processor's benchmark.
2. determine the FLOPs required for each operation in the pipelined processing.
3. divide step 2. by step 1. to get the number of PE required for each operation.
4. add the results of step 3. to get the total number of PE required.

Tables 5 and 6 show the procedure highlighted above for the GeoSAR X-band and P-band radars, respectively. The main difference between the two systems, for processing purposes, lies in the different azimuth reference lengths due to different radar center frequencies. A processing margin of 30% has been assumed for other unaccounted calculations. We note from these two tables that the range compression requires more processing elements than the azimuth compression, which also includes the corner-turn operation.

6.2. Memory Requirement

The memory requirements for the range and azimuth processors are shown in Table 7. The required memory for range compression is quite modest compared to the azimuth compression. A 50% saving in azimuth memory can be obtained if one stores the range compressed data in a fixed-point format at the expenses of dynamic range[29] and throughput, since data needs to be converted back to floating-point format after corner-turned for subsequent azimuth processing.

6.3. I/O Bandwidth Requirement

Assuming 8 bytes per complex sample, the total I/O bandwidth rate in the pipeline chain (range compression, corner turn, and azimuth compression) is just 8Q. The I/O

Operations	FLOPS required (GFLOPS)	SHARC throughput (MFLOPS)	Number of PEs
Range Processing			
Fast Convolution	1.8	94	19.15
Presum+Mocomp	0.24	27	8.99
Add	**2.04**		**28.14**
+ 30% overhead			**8.44**
Total PE_r required			**37**
Azimuth Processing			
Corner Turn	0.025	80	0.32
Range Migration	0.24	27	8.99
Fast Convolution	0.59	94	6.24
Magnitude	0.008	26	0.29
Add	**0.86**		**15.84**
+ 30 % overhead			**4.75**
Total PE_{az} required			**21**

Table 5: Number of processing elements (PEs) for X-band GeoSAR

Operations	FLOPS required (GFLOPS)	SHARC throughput (MFLOPS)	Number of PEs
Range Processing			
Fast Convolution	1.8	94	19.15
Presum+Mocomp	0.24	27	8.99
Add	**2.04**		**28.14**
+ 30% overhead			**8.44**
Total PE_r required			**37**
Azimuth Processing			
Corner Turn	0.025	80	0.32
Range Migration	0.24	27	8.99
Fast Convolution	0.79	94	8.39
Magnitude	0.008	26	0.29
Add	**0.86**		**17.99**
+ 30 % overhead			**5.40**
Total PE_{az} required			**24**

Table 6: Number of processing elements (PEs) for P-band GeoSAR

	Formula	X-band	P-band
Range Processing			
Total	32 $N_s PE_r$	18 MB	18 MB
Per Processor	32 N_s	0.64 MB	0.64 MB
Azimuth Processing			
Total	24 $N_{rv} M_a$	310 MB	4955 MB
Per Processor	24 $N_{rv} M_a$ / PE_a	20 MB	275 MB

Table 7: Memory requirements

bandwidth per processor is twice (input + output) the total I/O rate divided by the number of processing elements in each pipeline stage. Table 8 shows the results. These I/O requirements are well below today's advanced VME/PCI buses that can sustain data rate well above 160 MB/s [9, 30].

	Formula	I/O Bandwidth (MB/s)
Total	8Q	51
Per Range Processor	16Q / PE_r	1.4
Per Azimuth Processor	16Q / PE_{az}	2.4

Table 8: I/O bandwidth requirement

6. Approach for Optimal Hardware Configuration

The range compression is characterized by a low memory-to-processor ratio (18 MB divided by 37 PE_r) whereas the azimuth compression has a very high memory-to-processor ratio (310 -- 4955 MB divided by 24 PE_{az}). A custom-designed DSP board can produce any desired memory-to-processor ratio. However, this approach is time-consuming and fairly expensive. Also, custom-built board is not flexible in the sense that it is designed for specific requirements. If any of these requirements changes in future time (i.e. frequency, bandwidth, pulse width, swath width, flying altitude, etc.) leading to new memory-to-processor requirements, new boards need to be fabricated. Or the whole processing system may become inefficient, and even obsolete. These drawbacks lead to more frequent use of COTS-based products (i.e. commercially available in a competitive market), that contain a wide range of memory-to-processor ratios. The optimal hardware configuration consists of choosing the optimal mix of these COTS-based DSP boards to meet the throughput and memory requirements, and at the same time satisfying the maximum size, weight, and power (SWAP) constraints. As discussed in the first report, the host system must be standard and open to ensure easy insertion and rapid prototyping of hardware and software components. Only in this case, can hardware configuration using COTS-based components make sense and deliver the optimum solution.

6.1. Choice of Objective Function

In airborne and spaceborn missions, all SWAP constraints are important. For computation processing purposes, the power requirement is the most important parameter. The weight of a computer system is determined by the custom-built chassis. The size of the system depends on the processing density of the processing elements and on the geometry by which they are arranged. However, power consumption is by far the fundamental variable to be minimized.

6.2. Mercury's COTS -Based Daughter Cards

To quantify the hardware design, we consider as examples the following specifications for four Mercury Computer's SHARC daughter cards [31]. Each daughtercard has six SHARC processors with two independent compute nodes (CN) of three processors each. Each CN on the daughtercard has 8, 16, 32 or 64 MB of DRAM that is shared by both CN's processors.

	S2T8B-D	**S2T16B-D**	**S2T32B-D**	**S2T64B-D**
Configurations:				
Compute Nodes (CNs)	2	2	2	2
SHARCSs per CN	3	3	3	3
MB DRAM per SHARC	2.7	5.3	10.7	21.3
MB DRAM per CN	8	16	32	64
MB DRAM per card	16	32	64	128
Peak Performance:				
FFTs (MFLOPS)	720	720	720	720
Single operation functions (CPU limited)	240	240	240	240
Multiply accumulate functions (CPU limited)	480	480	480	480
Data Transfer Rates:				
DRAM (MB/s)	320	320	320	320
SHARC buses, shared (MB/s)	320	320	320	320
Per SHARC, average (MB/s)	53.3	53.3	53.3	53.3
Electrical/Mechanical Specifications:				
Power (watts)	14	14	14	14
Weight (pounds)	.4	.4	.4	.4
Dimensions	5" x 4.435"	5" x 4.435"	5" x 4.435"	5" x 4.435"

Table 9: Mercury's SHARC daughtercard specifications.

6.3. Choices of Hardware Configuration

There are three possible hardware configurations. In the first choice, only one type of daughtercards is chosen so that the available processors and memory are shared by both range and azimuth processing. From Tables 5 and 6, the number of range processors is about twice that of azimuth processors. Since each CN has three SHARCs, one SHARC can be assigned to the range processing while the remaining two perform the azimuth processing. For the X-band system, the 64 MB of memory on the S2T64B-D (21.3 MB per SHARC) satisfy the total memory required for these three range and azimuth processors. However, none of these cards meets the very high memory-to-processor requirement (85 MB per processor) for the P-band radar. In this case, some remote memory access is required, thus raising the throughput and power consumption.

Since the memory-to-processor requirements are different for the range and azimuth processors, it is possible to assign two different card types to each of the range and azimuth processing. The range (or azimuth) processing will have all the available SHARC processors (3) and memory on each CN for its own use.

Lastly, in situation where there is a large disproportion between the numbers of range and azimuth processors, we can combine the two previous approaches. The first set of CNs is shared between the range and azimuth processing. The remaining set is exclusively devoted to the processing stage that requires more processors.

The last step is to find the total power consumption as a function of the type and number of daughtercards used in the configuration. This power consumption is also a function of any software parameter in the SAR signal processing (for example, the FFT section lengths). The objective is to minimize this power consumption subject to the number of processors and memory required.

In summary, the time-consuming and expensive problem of custom-designed board is reduced to simple optimization calculation. The availability of different DSP boards in a competitive market leads to great system flexibility. Use of COTS-based components would ensure that these objectives could be met successfully.

7. Summary

In this second we have shown the steps in designing a COTS-based parallel computer for real-time SAR signal processing. Using the NASA/JPL-developed GeoSAR system as an example and the Mercury Computer System's SHARC daughtercards as a typical hardware platform, we have provided the calculations of the resource requirements (number of processors and memory), and different choices of hardware configuration for minimum power.

Part III: Parallel Programming Issues

1. Introduction

Parallel programming is the topic in this third and final part. First, we give an overview of the standard multicomputing platforms and parallel programming models. Then, we compare and contrast "scientific" versus "real-time" processing/programming. Next, we show the hardware/software co-design approach for SAR signal processing. Then, we discuss the current issues in parallel programming. Finally, we describe a typical software architecture for parallel programming of SAR signal processing.

2. Review of Standard Multicomputing Platforms

Scalable parallel computer systems are classified into five general models [32] as shown in Figure 1: the *parallel vector processors* (PVPs), the *symmetric multiprocessors* (SMPs), the *massively parallel processors* (MPPs), *the distributed shared-memory multiprocessors* (DSMs), and the *cluster of workstations* (COWs). The important characteristics of these systems consist of:

- Commodity components: using commercial-off-the-shelf (COTS) microprocessors, memory, disks, and software.
- Multiple-instruction-multiple-data (MIMD) architecture: the application program is divided into processes, each running a possibly different subprogram on different processor.
- Asynchrony: each process executes at its own pace. Synchronization is maintained through the use of semaphores, barriers, sockets, threads, blocking-mode communications, etc.
- Distributed memory: memory is physically distributed among different compute nodes (CNS), either shared or unshared. Memory access is carried out using uniform-memory-access (UMA) or non-uniform-memory-access (NUMA) models. Because of its centralized shared memory, the UMA model may limit scalability once built.

Table 1 compares the architectural attributes and performance of these five parallel architectures.

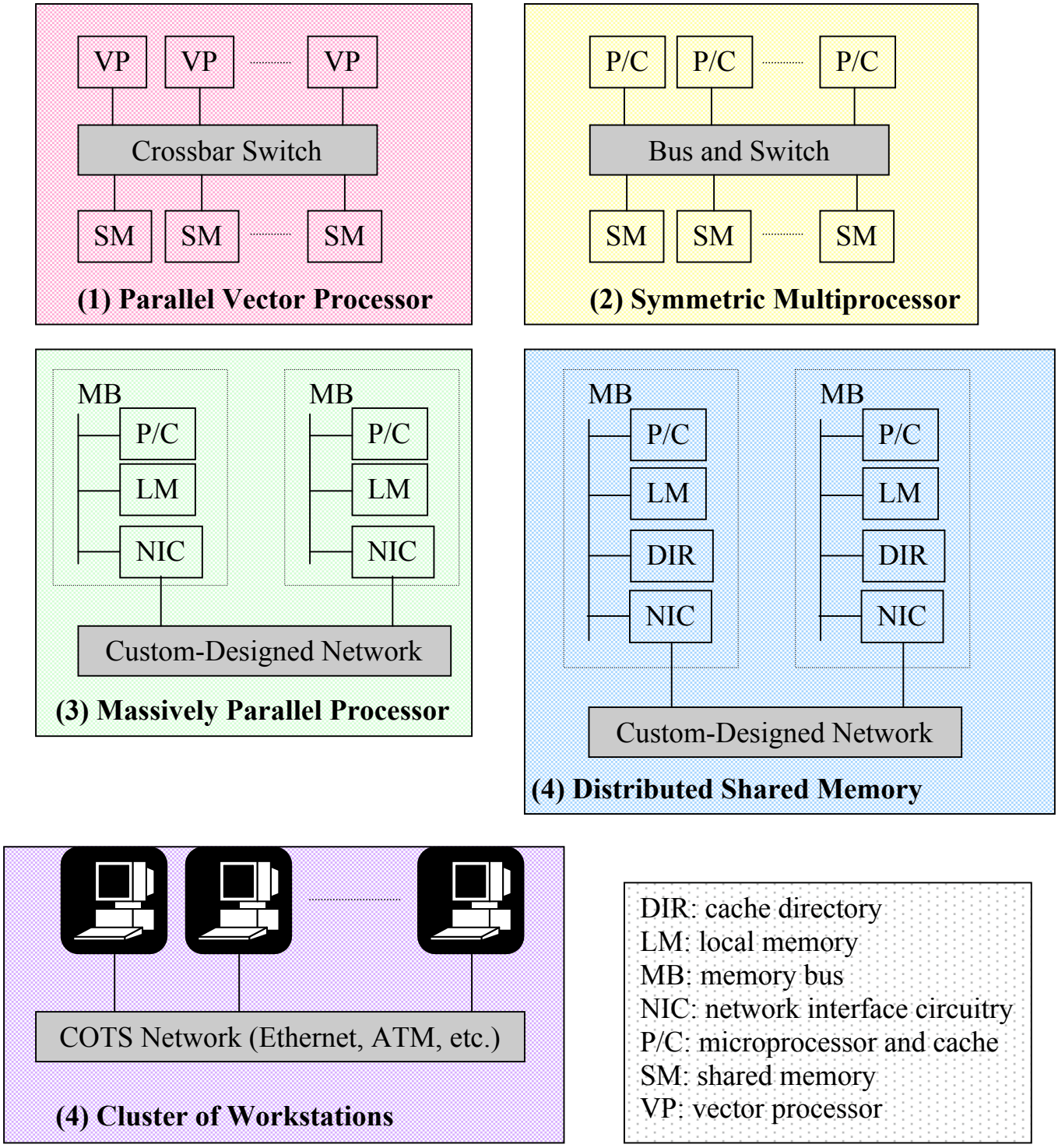

Figure 1: Computer Architectures of Scalable Parallel Computers [32]

Attribute	PVP	SMP	DSM	MPP	COW
Example systems	Cray C-90 Cray T-90 NEC SX-4	Cray CS6400 IBM R50 DEC 8400 SGI Power Challenge	Stanford DASH Cray T-30 SGI Origin 2000	Cray T3E Intel Paragon IBM SP2	Berkeley NOW IBM SP2 Alpha Farm Digital's TruCluster
Typical Usage		Database, data warehouse, on-line transaction system		Scientific computing, engineering simulation, signal processing, data warehouse	
Processor type	Custom vector-processor	COTS Microprocessor	COTS Microprocessor	COTS Microprocessor	COTS Microprocessor
Memory model	Centralize shared	Centralized shared	Distributed shared	Distributed unshared	Distributed unshared
Address space	Single	Single	Single	Multiple	Multiple
Access model	UMA	UMA	NUMA	NUMA	NUMA
Interconnect	Custom crossbar	Bus or crossbar	Custom network	Custom network	Commodity network
Scalability	Low	Low	High	High	High
Performance/ Cost Ratio	Medium			High	Low

Table 1: Architectural attributes of MIMD parallel computers

3. Review of Standard Parallel Programming Models

Table 2 lists the four parallel programming models: *implicit*, *data parallel, message-passing*, and *shared-variable* models [33]. Implicit model lets the compiler and the run-time support system exploit the program parallelism. The remaining three programming models are explicit models where the user explicitly specifies in the source code the program parallelism. This can be achieved by using three realization approaches: *library subroutines* (additional library to support parallelism and interaction operations*), new constructs* (the programming language is extended with new constructs to support parallelism and interaction), and *compiler directives* (formatted comments to help the compiler in optimization and parallelization). The table also shows existing software products together with the targeted hardware platforms. The remaining entries focus on various parallel programming issues (★★★★: most favorable, ★: weakest). *Parallelism* issues concern process creation/termination, context switching, number of processes. *Interaction* issues involve data and workload allocation, and processor synchronization and communication. *Semantic* issues affect termination, determinacy, and correctness properties. And *programmability* issues refers to code efficiency and portability.

Issues		Implicit	Data Parallel	Message Passing	Shared Variable
Platform Independent Examples		Kap, Forge	Fortran 90/95 Fortran 2001 HPF, HPF-2 PC++, Nesl	PVM, MPI, MPI-2	X3H5 POSIX Threads OpenMP C//
Platform Dependent Examples		Convex Exemplar	CM C*	IBM SP2, Intel Paragon	Cray MPP, SGI Power C
Parallel Hardware Platforms			PVP, SMP, MPP, DSM, COW	MPP, COW	PVP, SMP, MPP, DSM, COW
Realization Approach			New construct, directives	Library	Library
Parallelism issues		★★★★	★★★	★★	★★
Interaction Issues	Data Allocation	★★★★	★★	★	★★★
	Computation Allocation	★★★★	★★★★	★★	★★★
	Communication	★★★★	★★★	★	★★★
	Synchronization	★★★★	★★★★	★★	★
Semantic Issues	Termination	★★★★	★★★★	★	★
	Determinacy	★★★★	★★★★	★★	★
	Correctness	★★★★	★★★	★	★
Programmability Issues	Efficiency	★	★★	★★★★	★★★★
	Portability	★★★★	★★★	★★★	★

Table 2: Comparison of four parallel programming models [33].

Table 3 further classifies the explicit programming models based on their architectures and working principles. Further information can be found in the references [32, 33].

Feature	Data Parallel	Message Passing	Shared Variable
Control Flow (threading)	Single	Multiple	Multiple
Synchrony	Loosely synchronous	Asynchronous	Asynchronous
Address Space	Single	Multiple	Multiple
Interaction	Implicit	Explicit	Explicit
Data Allocation	Implicit or Semi-explicit	Explicit	Implicit or Semi-explicit

Table 3: Main features of explicit parallel programming models [33].

4. Comparison between "Scientific" and Real-Time" Computing

Traditional supercomputers (PVP, SMP, MPP) with general-purpose computing design philosophy have served well the scientific and business computing markets [34, 35]. In contrast, embedded systems are often found in real-time, sensor-based, signal and image processing applications. The fundamental characteristics separating the two applications

are: application flow, deterministic requirements, support for heterogeneity, and hardware utilization requirements [34].

In scientific applications data are given just once as an initial value problem and evolve according to a scientific model. Each evolution step involves different and complicate operations and is subjective to change with respect to the previous steps, depending on the data runtime value.

In real-time, sensor-based applications, the execution process is divided into two phases: SETUP and GO phases. During the SETUP phase, the software performs functions with high level of latency and low level of determinism, such as shared memory creation, processors hiring, task partitioning and scheduling, initialization of parallel libraries, and establishing of processor communication/synchronization. Thanks to the regularity of the processing during the GO phase, this SETUP procedure is performed only once and the entire subsequent processing can be completely planned ahead. That is, the processing regularity permits static task mapping and scheduling (i.e. task mapping and scheduling are determined at compile time). This is in contrast to scientific programming where dynamic task profiling and scheduling are required.

The software then gives way in the GO phase where the hardware takes over and manages the data movement and transformation. During this time, the sensor continuously delivers data, probably at a high rate, to be processed. However, the processing is relatively simple (FFT, filtering, etc.), repetitive, and predictable. That is, data value do not dictate what the application should do, as with scientific application that decides which conditional branch to process based on data values in the previous steps. Thus real-time sensor-based applications require quick and deterministic performance. What separate real-time programming from scientific programming are high data refreshing rate, its passive nature, and the processing regularity.

As mentioned previously, real-time applications require uninterrupted data collection and continuously concurrent processing of data. These requirements lead to high demand for deterministic control, high-speed processing, and efficient communication and synchronization mechanism. In addition, real-time embedded systems require extremely high hardware utilization (optimal use of resource) in terms of performance (MFLOPS) per cubic foot, per pound, per watt, and per dollar. This implies maximal hardware fine tuning and minimal software overhead. Thus, both hardware and software latencies must not only be low, but also deterministic in order to keep up with the continuous data stream.

To meet the high hardware utilization, embedded real-time applications are best served by a heterogeneous mix of processors [36]. For example, custom processors can be used for low latency I/O interfaces and FFT calculations. DSP chips are ideal for vector tasks, such as digital filtering, image formation, data reconstruction and resampling, and signal classification. And RISC can be employed to perform scalar task, such as feature extraction, inverse problem, parameter retrieval, image classification and interpretation,

large scale (global) surveying and understanding, target detection, and decision making processes.

In general, there are three types of heterogeneous computing (HC) models: mixed-machine model, mixed-mode model, and mixed-component model [37]. In a mixed-machine system, a heterogeneous mix of independent machines of different types are interconnected by a high-speed network (e.g. crossbar switch, SCI, HiPPI, Myrinet, fiber channel, gigabit Ethernet, etc. [33]). A mixed-mode system has a single parallel machine whose processors are capable of operating in either the synchronous SIMD or asynchronous MIMD mode, and can dynamically switch between these two modes. A mixed-component system consists of a machine with different components where each represents one mode of parallelism.

5. Heterogeneous Multicomputing for SAR Real-time Processing

SAR signal processing (filtering, image formation, detection, classification, etc.) usually involves different types of computation (vector and scalar processing) [2]. Consequently the corresponding signal processing architecture is best served with a mix of computing technologies [36, 38]. Homogeneous machines, where only one type of microprocessor is employed, cannot achieve their peak performance during the entire process. This is due the fact that different algorithms within the SAR signal processing have different computational requirements. And currently there is no single microprocessor architecture that can serve all computational requirements equally well [36]. To meet the high throughput and low latency requirements, real-time SAR signal processing calls for heterogeneous platform where each processor is tailored to match with the type of computation for which it was designed [38, 39, 40, 41].

Figure 2 shows the software domain (computational model) and hardware domain (hardware realization) for real-time SAR signal processing using a heterogeneous multicomputing platform. The actual parallel implementation is a parallel pipelined system where each pipeline is a collection of tasks and each task itself is parallelized. This helps in improving throughput and latency, whose definition and discussion will be given in the subsequent sections.

In the first stage, the sensors deliver analog data to be digitized by the I/O device (e.g. A/D converter). The digital data are then compressed (filtered, convolved, etc.) to form image. While this stage is highly computationally intensive, the processing can be separated (pipelined) into well-defined vector subtasks (range compression, corner turning, azimuth compression, etc.) where multiple processors can be employed in parallel. Also, these filtering operations are very regular, repetitive, and require minimum software conditional branches. Hence, DSP chips are best suited for this second stage. The third stage consists of image segmentation and feature extraction. This stage is characterized by mixed vector and scalar operations, which can be effectively performed with multiple microprocessor units running in parallel. The next stage involves data interpretation and decision making. The processing is highly scalar in

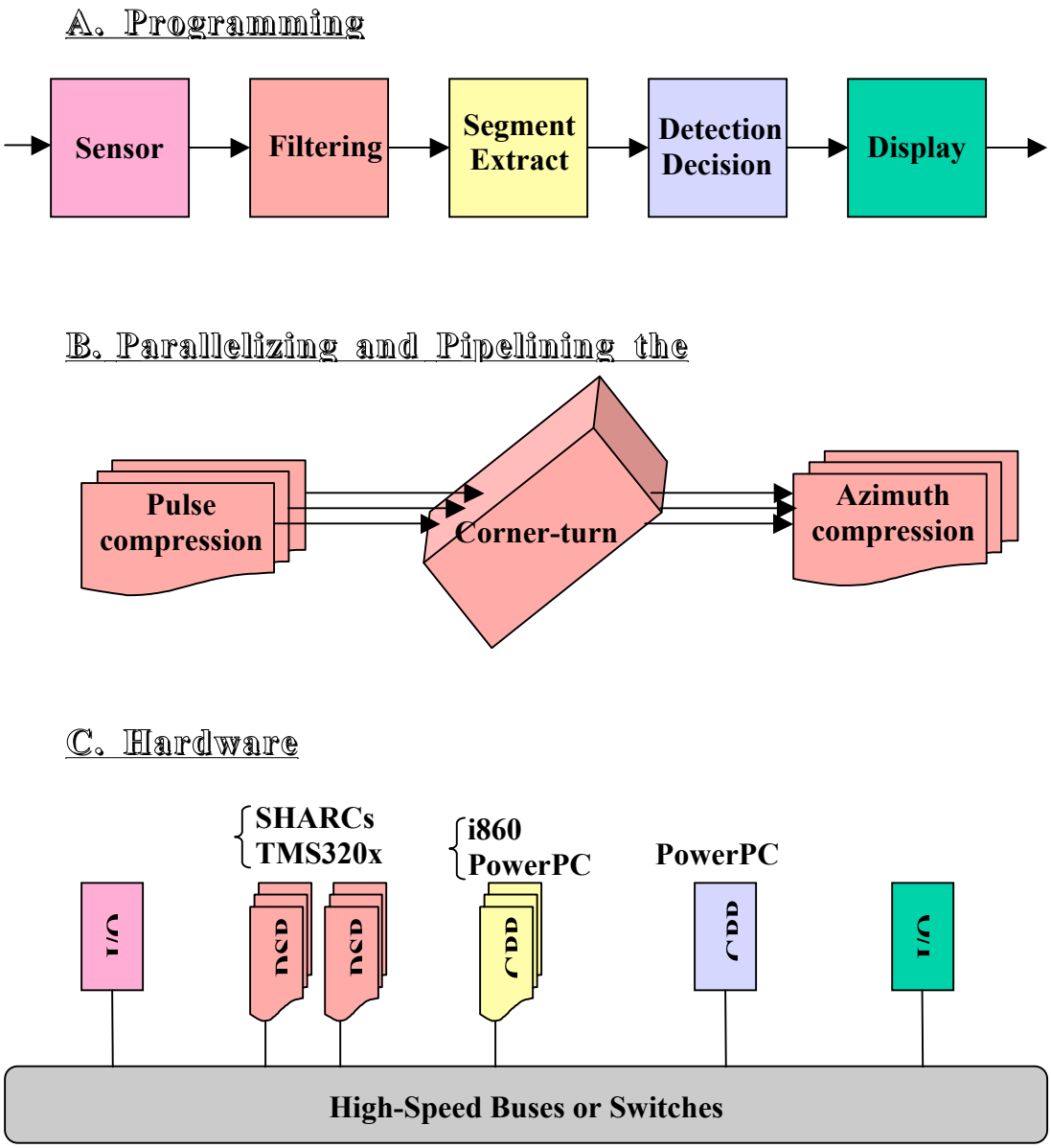

Figure 2: SAR signal processing on heterogeneous multicomputing platform.

nature, requiring complex software programming, and utilizing mostly conditionals, table look-ups, jumps, for which RISC processors present better fits than DSP chips. Finally, the processed data are stored on high-speed tapes or sent out to a display device. An I/O device is responsible for this bulk data movement.

6. Parallel Programming Issues

As described in the first two parts [1, 2], COTS components offer many advantages in terms of system scalability, programmability, flexibility, performance with respect to

size, weight, power, and cost. However, COTS components are not originally designed to guarantee optimal performance for a particular application. Hardware integration alone is not sufficient to meet the performance requirements.

To fully exploit the use of a heterogeneous mix of COTS components, software programming tools such as high-level language, efficient parallel library, task profiling and analytical benchmarking, task mapping and scheduling, supporting compiler and operating system mechanisms, should be available to the programmers. Currently, parallel programming technique is not advanced enough to permit automation of task mapping and scheduling [33, 37]. The user must explicitly decompose the application into appropriate subtasks, decide on which machine to execute each subtask, code each subtask specifically for its targeted machine, and determine the relative execution schedule for the subtasks. However, full automation is an active research topic both in academia and industry.

7.1. Communication time

In an integrated system which implements several tasks that feed data to each other, each tasks takes some amount of time that equals to the sum of [42]

- Time required to transfer the input data from the previous task.
- Time required to process the data.
- Time required transfer the output data to the next task.

Therefore, in addition to the processing time, communication time (between the processing units, between the processing unit and its on-chip memory, between the processing unit and the bulk memory storage) is also needed among the tasks. Experience over the years has shown that processor speed is increasing at a faster rate than the speed increase of memory and interconnect network [33] . In fact, communication time is the major overhead that can adversely impact scalability when mapping an embedded signal processing application onto a high-performance computing platform [39].

There are two types of data dependencies that can affect the communication time: spatial and temporal data dependency. *Spatial* data dependency is classified into intra-task and inter-task dependencies. *Intra-task* dependency occurs when subtasks are exchanging intermediate results (i.e. overlap-add and overlap-save convolution*). Inter-task* dependency results from data movement between parallel tasks (i.e. pulse compression followed by azimuth compression). *Temporal* dependency arises when previous data set is needed for the current process. Good understanding of the data flow and efficient data re-distribution and task scheduling is required in order to meet the performance requirement.

7.2. Throughput and Latency

While other parameters such as size, weight, power, and cost impose constraints on the hardware/software architecture, two of the most important parameters that need to be

optimized are *throughput* and *latency* [39, 43]. Throughput is the average rate at which output sample leaves the system. The system throughput must keep pace with the input data rate. Latency is the worst-case time between the transform of an input sample and the delivery of the processed output sample. The system latency is such that all the processing must be done within an assigned time. The throughput/latency optimization problem consists of minimizing the number of processors needed to satisfy the throughput requirement subject to the latency constraint. Numerous research studies have been conducted and several models have been proposed to help minimize latency and maximize throughput, for a given number of processors [37, 39, 43].

7. Typical Software Architecture for SAR Real-Time Signal Processing

In this section, we provide an example of a typical parallel software structure. The example shown below represents the architecture designed for Mercury's RACEway multicomputer [42]. The process breaks down into SETUP code and GO code

SETUP Phase

- *Allocate local memory*
- *Boot compute elements*
- *Spawn processes*
- *Create endpoints*
- *Create transfer requests*
- *Set up synchronization objects*
- *Set up look-up tables*

GO Phase (inner-loop functions)

while (is_data_coming?)

- *Get data in*
- *Process data*
- *Send data out*

end while

The descriptions of the activities are given below.

7.1. SETUP Code

- **Allocating local memory:** to allocate memory for function calls, to provide aligned memory for special library calls, and to allocate on-chip memory on certain DSP chips (such as the SHARCs).
- **Booting compute elements (CE):** to load an executive into the targeted CE's memory and to provide the physical address of the targeted CE in a interprocess communication database.
- **Spawning processes:** to load an executable image into the targeted CE's memory.

- **Creating endpoints:** to create endpoints which include shared memory buffer (SMB, associated with random-access memory) and stream endpoints (associated with devices).
- **Creating transfer requests:** to define the source endpoint, the transfer engine, and the destination endpoint for a particular transfer. Endpoints can be SMB or devices. Transfer engine include direct memory access controller (DMA engine) and the DMA engine for external devices.
- **Setting up synchronization objects:** Standard synchronization objects include: sockets, semaphores, and signals. *Sockets* provide full-duplex message passing between processes. *Semaphores* are used for synchronization. And *signals* provide asynchronous notification to processes.
- **Setting up look-up tables:** to create look-up tables for frequent used functions and transforms such as trigonometric functions and fast Fourier or discrete cosine tranforms.

7.2. GO Codes (inner-loop processing)

- **Data transfer:** data movement extensively used transfer requests previously created in the SETUP phase. Data transfer can be between processes, between a process and a logical device, and between a process and a raw device.
- **Processing data:** inner-loop processing is usually performed with parallel library functions that have been optimized for a targeted processor. Input data should reside in local memory to reduce the communication time from reading external memory.

8. Summary

In this final part, we reviewed the standard multicomputing platforms and parallel programming techniques. We then compared scientific programming and real-time sensor-based processing where a heterogeneous system offers many advantages. We also pointed out that parallel programming is not a mature technique and presents many challenges. The ultimate goal is to provide full automation of the hardware/software co-design. Until this automatic tool is available, the hardware designer and software programmer must explicitly specify the task partitioning and handle all the software communication/synchronization processes to ensure low latency and high throughput to meet performance specifications.

REFERENCES:

[1] C. Le and S. Hensley, "Using COTS components for real-time processing of SAR systems", First Report, Jet Propulsion Laboratory, Pasadena, CA, July 1998.

[2] C. Le and S. Hensley, "Using COTS components for real-time processing of SAR systems", Second Report, Jet Propulsion Laboratory, Pasadena, CA, August 1998.

[3] C. Le and S. Hensley, "Using COTS components for real-time processing of SAR systems", Third Report, Jet Propulsion Laboratory, Pasadena, CA, September 1998.

[4] R. Jaenicke, "Multiprocessing issues in large systems", *Computer Design*, pp. 51-53, December 1996.

[5] K. Hwang, *Advance Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., New York, NY, 1993.

[6] J. Child, "PCI, VME boards vie for image processing designs", *Computer Design*, pp. 91-97, April 1998.

[7] L. Happ, F. Le, M. Ressler, and K. Kappra, " Low-frequency ultra-wideband synthetic aperture radar: Frequency subbanding for targets obscured by the ground", in *Radar Sensor Technology*, G. S. Ustach, Editor, *Proc. SPIE*, vol. 2747, pp. 194-201, 1996.

[8] R. Dressler, D. Barnum, and M. Loiz, "COTS SAR processing software", IEEE National Radar Conference, pp. 136-141, 1996.

[9] S. Ohr, "New-generation radar processing depends on fast A/D converters:, *Computer Design*, pp. 85-90, August 1996.

[10] R. W. Bayma and E. Trujillo, "HISAR COTS-based synthetic aperture radar", *Proceedings of the 15th IAA/IEEE Digital Avionics Systems Conference*, pp. 319-325, 1996.

[11] P. G. Meisl, M. R. Ito, and I. G. Cumming, "Parallel processors for synthetic aperture radar imaging", *Proceedings of the International Conference on Parallel Processing*, 1996.

[12] TNO Physics and Electronics Laboratory, "RACEway Compatible Pulse Compression Board", http://www.tno.nl/instit/fel/div3/pulsbord.html.

[13] B. C. Kuszmaul, "The RACE network architecture", *Proceedings of the 9th International Parallel Processing Symposium (IPPS'95)*, pp. 508-513, April 1995.

[14] T. H. Einstein, "Mercury Computer Systems' modular heterogeneous RACE multicomputer", *Proceedings of the 6th Heterogeneous Computing Workshop (HCW '97*), April 1997.

[15] J. M. Rabaey, W. Gass, R. Broderson, T. Nishitani, and T. Chen, "VLSI design and implementation fuels the signal-processing revolution", *IEEE Signal Processing Magazine*, vol. 15, no. 1, pp. 22-37, January 1998.

[16] 3L Limited, "Parallel C and Multiprocessor DSP RTOS Software Solutions".

[17] J. Gadient, and G. A. Frank, Editors, *Rapid Prototyping of Application Specific Signal Processors*, M. A. Richards, A., Kluwer Academic Publishers, January 1997.

[18] K. Konstantinides, "VLIW architectures for media processing", *IEEE Signal Processing Magazine*, vol. 15, no. 2, pp. 16-19, January 1998.

[19] P. Faraboschi, G. Desoli, and J. A. Fisher, "The latest word in digital and media processing", *", IEEE Signal Processing Magazine*, vol. 15, no. 2, pp. 59-85, January 1998.

[20] Van Zyl et. al., "GeoSAR P-band radar CDR", Jet Propulsion Laboratory, Pasadena, CA, May 1997.

[21] S. Hensley et. al., "GeoSAR IFSAR processor PDR/CDR", Jet Propulsion Laboratory, Pasadena, CA, September 1997.

[22] Van Zyl et. al., "GeoSAR X-band radar CDR", Jet Propulsion Laboratory, Pasadena, CA, November 1997.

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

[24] W. Owens, "Why COTS is vital to the modern military?", *COTS' 95 Conference*, 1995

[25] R. Costello, "Commercial Products and Military Preparedness", *COTS' 95 Conference*, 1995.

[26] D. MacDonald, J. Isenman, and J. Roman, "Radar detection of hidden targets", *NAECON 1997 Proceedings of the IEEE 1997 National Aerospace and Electronics Conference*, pp. 846-855, 1997.

[27] R. W. Bayma and E. Trujillo, "HISAR COTS-based synthetic aperture radar", *Proceedings of the 15th IAA/IEEE Digital Avionics Systems Conference*, pp. 319-325, 1996.

[28] T. Einstein, "Realtime synthetic aperture radar processing on the RACE Multicomputer", Application Note 203.0, Mercury Computer Systems Inc., 1995.

[29] P. Lapsley, J. Bier, A. Shoham, and E.A. Lee, *DSP Processor Fundamentals: Architectures and Features*, IEEE Press, 1997.

[30] J. Child, "PCI, VME boards vie for image processing designs", *Computer Design*, pp. 91-97, April 1998.

[31] *Race Series SHARC Daughtercards*, Mercury Computer Systems Inc., 1998.

[32] K. Hwang and Z. Xu, "Scalable parallel computers for real-time signal processing", IEEE *Signal Processing Magazine*, pp. 50-66, July 1996.

[33] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, and Programming*, McGraw-Hill, 1998.

[34] "Embedded applications for high performance computing", Panel Session, URL: *http://www.mc.com/backgrounder_folder/embedded_panel/panel.html*

[35] "Fitting architecture to application: Choosing between SMP and RACE", Multicomputing Technology Brief, 1996.

[36] R.F. Freud and H.J. Siegel, "Heterogeneous processing", *IEEE Computer*, vol. 26, no. 6, pp. 13-17, 1993.

[37] H.J. Siegel, H.G. Dietz, and J.K. Antonio, "Software Support for Heterogeneous Computing", in *The Computer Science and Engineering Handbook*, Allen B. Tucker, Jr. editor, CRC Press, Inc., 1997.

[38] H.J. Siegel, J.K. Antonio, R.C. Metzger, M. Tan, and Y.A. Li, "Heterogeneous Computing", in *Parallel and Distributed Computing Handbook*, A.Y.H. Zomaya editor, McGraw-Hill 1996.

[39] W. Liu and V.K. Prasana, "Utilizing the power of high-performance computing", *IEEE Signal Processing Magazine*, vol. 15, no. 5, pp. 85-100, September 1998.

[40] S.C. Patterson, "Heterogeneous multicomputing", *EDN Products Edition*, July 18, 1994

[41] B. Isenstein, "VMEbus leads as fast processors stress system interconnect", *EE Times VITA Insert Supplement page*, vol. 9, April, 1996.

[42] Developer's Guide, Mercury Computer Systems, Inc., 1997.

[43] A. Choudhary, W. Liao, D. Weiner, P. Varshney, R. Linderman, and M. Linderman, "Design, implementation, and evaluation of parallel pipelined STAP on parallel computers", *1998 IPPS/SPDP*, *Proceedings of the 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing*, pp. 220-225, Orlando, Florida, 1998.