General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Produced by the NASA Center for Aerospace Information (CASI)

JPL PUBLICATION 83-54

Parallel Processing in a Host Plus Multiple Array Processor System for Radar

Budak Ziya Barkan

(NASA-CR-173100) PARALLEL PROCESSING IN A HOST PLUS MULTIPLE ARRAY PROCESSOR SYSTEM FOR RADAR (Jet Propulsion Lab.) 74 p
HC A04/MF A01 CSCL 17I

N83-35201

G3/32 Unclas
42165

September 15, 1983

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. JPL Pub. 83-54	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Parallel Processing in a Host Plus Multiple Array Processor System for Radar		5. Report Date September 15, 1983 6. Performing Organization Code
7. Author(s) Budak Ziya Barkan		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91109		10. Work Unit No 11. Contract or Grant No. NAS 7-100 13. Type of Report and Period Covered JPL Publication
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code RE 4 BP-656-44-03-03-00

15. Supplementary Notes

16. Abstract

Host plus Multiple Arroay Processor architecture is demonstrated to yiel a modular, fast, and cost-effective system for radar processing. Software methodology for programming such a system is developed. Parallel processing with pipelined data flow among the host, array processors, and discs is implemented. Theoretical analysis of performance is made and experimentally verified. The broad class of problems to which the architecture and methodology can be applied is indicated.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Computer Programming and Software Computer Systems Systems Analysis SEASAT-A Project	Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	67	

JPL PUBLICATION 83-54

Parallel Processing in a Host Plus Multiple Array Processor System for Radar

Budak Ziya Barkan

September 15, 1983

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

ACKNOWLEDGMENTS

I am grateful to my advisor Professor Walter Karplus of UCLA for his direction, sage advice, and support. I acknowledge the excellent work and dedication of Shirley Pang of JPL in all phases of the IDP and IDPMX projects. I would like to thank Dr. Chialin Wu of JPL who provided the algorithm and the environment. Without his thorough knowledge and guidance these implementations could not have been done. Beatriz Abu-Ata pleasantly prepared the manuscript.

This work was supported by research funds (as JPL Subcontract No. 955642 to UCLA) from Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS7-100, sponsored by the Information Systems Office of the National Aeronautics and Space Administration Office of Space Science and Applications.

ABSTRACT

Host plus Multiple Array Processor architecture is demonstrated to yield a modular, fast, and cost-effective system for radar processing. Software methodology for programming such a system is developed. Parallel processing with pipelined data flow among the host, array processors, and discs is implemented. Theoretical analysis of performance is made and experimentally verified. The broad class of problems to which the architecture and methodology can be applied is indicated.

TABLE OF CONTENTS

FIGURES

LISTINGS

TABLES

I. INTRODUCTION

Synthetic Aperture Radar (SAR) processing is characterized by high data volume and intensive number crunching. This double requirement makes efficient processing a challenge. The IO subsystem has to hold the large volume of data and sustain relatively high transfer rates. The arithmetic unit has to meet the demands of the necessary correlations to be done.

As the data acquisition rate by the sensor is two to three orders of magnitude faster than the processing speed, efficiency is paramount. Various attempts at system design to do SAR processing have ranged from all special-purpose hardware to minicomputer applications. There are important cost performance tradeoffs to be made in this area. Hardware attempts at real-time processing have not had widespread success, as this pushes the state of the art too much. Giant supercomputers have been utilized in medium-performance systems, but cost is prohibitive. The approach taken by us at the Jet Propulsion Laboratory (JPL) has been to use a cost-effective host plus peripheral array processor setup as an experimental and developmental system for radar.

A comprehensive background and a bibliography on SAR are in [19]. A survey of worldwide SAR digital processor development appears in [20]. The current status of the JPL system and the system's capabilities are summarized in [21].

The JPL radar algorithm on which this implementation is based is described in [1-2]. In summary, the computations involve convolutions and filtering which can more effectively be done in the frequency domain using Fast Fourier Transforms (FFT). The input data consists of sequential samples of radar echo signals. Data are range-compressed by a multiplication in the frequency domain. The ensembles of range-correlated echo lines are then corner-turned or transposed to allow access in the orthogonal direction. Azimuth correlation is done in the along-track or azimuth direction. The severe range migration effect makes the processing inherently two dimensional. The transform block sizes are large in both range and azimuth directions. So the size of the matrix involved in the cornerturn exceeds several megabytes.

The original JPL Seasat Processor is the Interim Digital Processor (IDP). It was upgraded with additional primary and secondary storage and APs. The augmented system is dubbed IDPMX, for IDP with multiple execution. In either case, the requirements were to produce 100-km x 100-km, four-look, 25-m resolution pictures. IDP was operational in April 1979, featured parallel operation with one AP, and took 10 hours per picture. IDPMX has been operational since February 1981, features parallel operation with three APs, and takes 2.5 hours per picture. Recent publications describing IDP and IDPMX have introduced some of the salient features of these systems [3-5].

The purpose of this report is to provide the necessary system design and analysis methodology as it applies to a host plus multiple array processor setup with parallel processing. In the HARDWARE ELEMENTS chapter, the characteristics of the hardware used are described to exemplify the capabilities required of the devices for such a system. The SOFTWARE TOOLS chapter describes the development necessary to utilize the hardware. SOFTWARE METHODOLOGY covers job partitioning and pipelined parallel processing at the device level. SYSTEM ORGANIZATION integrates the above hardware and software into a whole configuration. PERFORMANCE EVALUATION is done with emphasis on speed; this chapter has recommendations for the design of faster systems. In the APPENDIX there are code fragments in the various languages used to give the flavor of the programming. Also, a detailed, worked-out example illustrates the methodology of performance modeling. For completeness, two papers on cornerturn, titled "Transpose of Externally Stored Matrices", have been included in the appendix.

II. HARDWARE ELEMENTS

Discs

Discs, the host, and the array processors form the backbone of the configuration. All three elements are equally important because the resulting system balance will require about equal utilization of these subsystems.

Discs are rotating mass storage devices. Viewed in cylindrical coordinates (see Table I), each direction is partitioned into integer addresses. Herein we will assume 300-MB AMPEX discs as standard and will discuss their characteristics.

TABLE I
Disc Coordinates

angular	sector	23
vertical	surface	19
radial	cylinder	815

A sector is the smallest addressable unit on a disc. The rotation speed of 3600 rpm translates into 0.73 ms per sector. This is a natural unit to use in calculations involving discs and will be named sector-pass-time (SPT). Transfer rate, assuming a single head per track and 192 words per sector, is about 3.6 μs per word.

Crucial to the efficiency of a disc is the length of the intersector gap. This is the spacing between adjacent sectors on the disc. The sector gap is too small (10 μs) to allow time for data and command chaining without a next-sector miss. Thus multiple sequential records cannot be accessed without incurring a heavy rotational latency penalty.

The 192-word-per-sector format used as a standard is very wasteful of disc space because of fragmentation. Page sizes are powers-of-two in virtual memory systems. There are many transforms, notably the FFT, that operate on data of a power-of-two length. So a 128- or 256-word-per-sector format can be used.

Transfer control word is a designation for the number of words that are to be transferred to or from the disc. The larger the allowed maximum per transfer the better. Some records might naturally be long. Also, if the record length is a multiple of sector size, a multiple sequential record transaction can be software chained into a single large record (uneven record lengths can be padded up to multiples of sector size; though a potentially dirty method, this could be the only solution for faster IO in an existing system). Later, this can be accessed transparently and the constituent small records be available (this assumes effective sectors per cylinder can be specified to the IO routine). This does not require hardware data chaining and would be more efficient for the particular case. This is an important consideration in busy environments where congestion might cause the hardware data chaining method to miss the next sector.

Giving the controller the complete address in a single stage (at once) is preferable from logical and efficiency points of view. The cylinder and sector together form a complete address. Bookkeeping in software is diminished; all the system has to do is issue the single command, and the controller itself in hardware will determine whether a seek is needed. This effectively decouples the intelligent controller from the CPU.

A disc subsystem consists of several drives all hooked up to a single controller [6] (see Table II).

TABLE II
Desirable Characteristics of a Disc Subsystem

overlapped seek among drives
minimum latency targeting among drives
command and data chaining without next sector miss
power-of-two length sector format
wide transfer control word
single-stage address to controller

Host Computer

Systems Engineering Laboratories SEL 32/55 is a 32-bit minicomputer with an efficient bus [7]. The SEL bus is 32 bits wide in data and also has independent address and control lines. The memory on the bus is made of 4 noninterleaved 32-KW modules. Bus-to-bus memory access time is 0.6 μs, but requests to different modules can be overlapped. A memory read requires 2 transfers on the bus, a write only 1. On a cycle time of 150 ns, integer RR instructions take 0.6 μs and RX 1.2 μs. There is overlap in instruction fetch and execution.

Array Processors

Array processor AP-120B from Floating Point Systems (FPS) is a pipelined machine with a wide horizontally microcoded instruction word [8]. Independent functional units and multiple interconnecting paths are the key to its computing power. (The architecture and the programming methodologies of AP-120B have been the inspiration for the overall project. With hindsight, AP-120B microcoding provides an accessible microcosm in which one can learn and experiment with parallelism and pipelining. Also, one can make useful analogies. Data pad, say DPX, read-write is similar to APPUT-APGET, to illustrate a half-cycle operation). Working on a cycle time of .167 μs with a floating adder and a multiplier, the AP can theoretically achieve 12 million floating point operations per second (mflops). The APs have 64 KW of Main Data and 2 KW of Program Source memory each. AP-120B is hooked to the host SEL computer by a formatter and a High Speed Data (HSD) interface.

The formatter converts between the host and AP data representations on the fly. The HSD has a transfer rate of 1.2 μs per word.

The processing uses the raw power of the underlying hardware. Manual resource management is substituted for convenience to gain speed. In this light, machines which on paper have similar speed and functional capabilities can be used instead of the ones we have chosen. No nonstandard specific feature of host or array processor is necessary. To give some examples, any combination of CSPI MAP or FPS AP-120B for the array processor and DEC VAX, SEL 32, or PERKIN-ELMER 32 for the central processing unit will do. Even a 16-bit host might be sufficient (assuming intelligent array processor interface), but this would be a risky alternative in view of the overall hardware investment required for the system (half-million dollars).

III. SOFTWARE TOOLS

Disc Related

A custom disc IO routine, SARIO, was developed to efficiently handle the large volume of data that is moved around by the processing [9]. Another motivation was the necessity to define files of a size larger than allowed by the system. The key points are access modes and multiple record facility. In the processing, all of the accesses are highly regular and patterned. Most of the data are accessed in sequential mode. To facilitate corner-turning, a skipped mode was defined. In the skipped mode, a fixed distance between adjacent records is assumed. In both modes, the next record address on disc is easily calculable. Multiple record facility allows several records to be queued with a single call to the IO routine. The gains are more apparent in the sequential mode where only one latency per call is incurred. On our system, chaining was not available, so a sector is bypassed between records in order to have enough time to issue the next record transaction. The whole IO routine assumes dedicated discs. In the processing, only one file is active on a given disc at one time so the environment is virtually seek free. SARIO works asynchronously. After the subroutine call is processed and the first command initiated, control returns to the calling program. SARIO is from that point on interrupt driven to service completions and the issuing of further transactions that it was given in the original call. Later, for synchronization, SARIO is called with a wait command.

Host Computer Related

The FORTRAN compiler on SEL RTM performs reasonably for straightline code, though it is weak on loops. Lack of induction variable analysis is the main reason; also the compiler has trouble recognizing variables that are local to a loop. Thus assembly language was used to code the buffer movement loops using loop unrolling techniques. (SEL uses an encoding of the instruction type into the operand address, utilizing the redundancy in address to type correspondence, thus freeing instruction bits. Instructions are generic and can handle halfword, word, and doubleword types. This makes assembler programming quite tricky at times).

Array Processor Related

The basic software utilization of the AP-120B is through the Math Library supplied by FPS [10]. The canned routines written in APAL, the microassembler for AP-120B, are callable from the host [11]. If there is need to develop additional subroutines these can be coded in APAL and added to the Mathlib. Mathlib includes vector arithmetic, data formatting, and numerical and signal processing routines. For the bulk of the radar task the Mathlib is sufficient; nevertheless, some improvements and additions were made. These were 11-cycle 4-bit-unpack, 8-cycle 8-bit-unpack, and 3-cycle (extra +1 for highclip, +2 for lowclip) scale-bias-clip routines [12]. (Signed unpack is slower and can not be improved. After FFT, bias can be subtracted from the first coefficient. So, with the free bias adding in the clip routine, data can be kept positive if FFT is the first operation to be done).

In the development of APAL routines, the AP simulator and debugger programs, APSIM and APDEBUG, were not used. Microcode development and testing were done by comparing the end results with FORTRAN equivalents. Development and testing environments were the same. The testing philosophy was biased towards random testing. It is felt that for the type of procedures coded, this is an adequate and correct approach. Preparing deterministic test cases is inherently harder and more error prone than writing a correct program. Still, the ranges of random input values and vector lengths must be chosen with thought to make random testing effective. To check staying in loops too long, memory locations surrounding the result buffer are also examined.

The host overhead in initiating AP-calls is substantial (1.1 ms), so a software package called Vector Function Chainer (VFC) is available from FPS to efficiently utilize the AP [13]. Multiple consecutive calls are linked together to yield one call that will do the whole thing by subroutine calls internal to the AP. VFC also allows limited integer and logical operations so complete loops can be put into the AP.

AP-to-host connection is achieved by the Array Processor Executive (APEX) supplied by FPS [14]. APEX keeps tables of presently residing microcode in the AP, the commands to issue to the AP, and the HSD channel the AP is on. AP-120B also works asynchronously with respect to host CPU operation and data transfers to or from the host. The synchronization calls APWR and APWD are available in APEX to wait for AP running and AP direct memory access transfers, respectively. (The redundant wait-on-wait-type logic used in APEX was improved by us).

To make multiple array processors controllable from one program, a multiple-AP Apex was developed [15]. The modifications involved duplicating internal tables and data areas and providing facilities for indexing into them. This code also allows dynamic assignment and release of APs. It is completely upward compatible, in that programs using only one AP can run unmodified as is. The program when assigning APs establishes a logical-to-physical correspondence. This allows all code development to address logical AP numbers. The added SETAP routine sets the AP which is going to be acted on in the subsequent calls to multiple-AP Apex.

IV. SOFTWARE METHODOLOGY

Job Partitioning

Job partitioning to facilitate the efficient implementation is an important step in the planning stage. This way, proper utilization of the available computing and storage resources can be attained. Range processing does not need any big buffers. Cornerturn needs two, one for each stage. Azimuth processing needs one, a window to handle the range curvature. This way there are eight disc IO per line. Combining range and the first stage of cornerturn reduces the IO count to six. Further combining the second stage of cornerturn with azimuth would reduce the IO count to four. But this combination would require two big buffers in one program, for which we did not have space. The partitioning into two subtasks would reduce IO requirements considerably.

The overall key problem is the accommodation of the cornerturn process. Our algorithm for cornerturn is an on-line algorithm. This means all the range data need not be done before the cornerturn can start. Cornerturn can proceed to do part of its work immediately after the first range processed line arrives. In fact, cornerturn is integrated into what comes before and after the simpleminded cornerturn. This way we avoid an extra read/write of the whole range processed data. Another consideration was to implement cornerturn in a skip-write, sequential-read fashion. As data is azimuth processed with overlap, what is written once gets read multiple times depending on the degree of overlap. As the skipped mode is always the more time consuming operation, skip-write, sequential-read is the better choice than sequential-write, skip-read.

Using software disc formatting techniques, the skipped operation can be speeded up dramatically. The idea is to anticipate the locations one will be at after each record of the skipped mode whether or not a seek is involved. This way, which involves addressing complexities and also affects the reading method, latency throughout can become zero and only the seek and transfer times will show [16-17].

Another application of software cylinder formatting is real-time disc IO. For multiple records, the worst-case formula is latency + seek + latency + transfer. The first latency is random, and therefore is figured as a full revolution. The latency after seek is calculatable, and can be eliminated using appropriate DELTA in SARIO.

The total processing job is now partitioned into several subtasks called phases. Each phase represents a pass or one step of processing over a large amount of data. The limitations on the data volume for each pass are set by the available intermediate disc storage space. Three major phases of the SAR correlation are termed RACR, CRAZ, AZOL. Respectively their functions are: range correlation with initial cornerturn operation, completing cornerturn with first part of azimuth processing, and completing azimuth filtering to form single-look imagery. Another intent of this partitioning is to balance the amount of arithmetic processing that is performed by the APs over the three phases. Additionally, there are two peripheral phases for postcorrelation data handling. These are OVLY for multiple-look overlay from correlated single-look imagery and MERG for assembling smaller image blocks into the final frame size [9].

Flow of Data

In the three major functional software modules described in the previous paragraph, the flow of data is similar. Data records, called lines, are read from one disc into computer memory. Within the computer some preprocessing is applied, which consists mainly of data selection and address manipulation. The preassembled data lines are transferred to the AP where the filtering operation is performed and then returned to the computer memory for postfiltering buffer movement operation. The results are then written onto the second disc. In this data flow structure we can identify the following seven data handling functions: (1) input from disc, (2) data selection, (3) transfer to AP, (4) AP processing, (5) transfer from AP, (6) postprocessing data selection, (7) output to disc. Each data line must go through these seven functions to complete a correlation operation. Concurrent execution of these functions with pipelined data flow is possible if independent buffers are provided. (Functions 2, 6 and 3, 5 cannot be done in parallel as they utilize the same physical device, CPU and AP HSD, respectively. The CPU carries the additional load of initiating the various calls). The concept of pipelining involves doing function 1 for line n, function 2 for line n-1, up to function 7 for line n-6. In the case of multiple APs, with the number of APs equal to nap, the above scheme is imitated except now every function is done on nap lines. In the AP processing function each line is sent to one of the nap APs.

The choice of the number of APs was made based on preliminary estimates of times taken by the overlapable subfunctions to be performed. The ideal is to achieve balance in the sense that the resulting system performance will be bound by all subfunctions. This way, utilization of subsystems will be close to unity. With 3 APs the system is limited by disc IO and a reasonable balance is struck.

In general, the methodology for determining the optimum number of APs is as follows. A per-line workload is determined for each of the subsystems. Then, the total elapsed time is calculated bottom up from the formulation given in the PERFORMANCE EVALUATION section for some number of APs. At some point diminishing returns are reached as overhead to use the APs swamps the available time or the system is bounded by IO.

Another decision, which influences both the quality of the final image and the throughput, is the number of bits per pixel at intermediate stages of the computation. The big buffers made up of such data affect the computer memory size, the IO volume, and the disc storage requirements. The size chosen was 8-bit-real, 8-bit-imaginary part per complex pixel to make the problem managable within the existing hardware. To what degree this tradeoff degrades the quality of image is open to investigation, though it is felt that the choice is satisfactory.

The parallel AP methodology is used in a problem in which the independence of a sequence of lines is exploited ("time"). The methodology is also applicable to parallelism in subportions of a line ("space"). In any case uniformity, granularity, large supply, and large frame times in the AP are needed. Many signal processing and image processing problems have these characteristics.

At JPL, [21-24], the Seasat digital processor has already been adapted to two other radar tasks, namely Aircraft and Quick-look, that had peculiarities of their own. Automated preprocessing and geometric rectification have been carried out in a parallel-pipelined environment. A feasibility analysis of near-real-time processing and a proposal for the now defunct VOIR mission were submitted. The upcoming SIR-B digital correlator will utilize the tools developed and the general approach.

ORIGINAL PAGE IS
OF POOR QUALITY

V. SYSTEM ORGANIZATION

Configuration Issues

The design of the IDPMX system is based on the SEL 32/55 computer running under RTM and acting as host to three FPS AP-120B array processors running under multiple-AP Apex. Overall philosophy is one of central control and central organization around the SEL bus. Two Ampex 300-MB discs on separate channels serve as the bulk storage medium. One is the raw data disc, the other serves as the intermediate disc. A third channel houses the system disc for software and auxiliary data storage. The radar data is transferred to the raw data disc through a fiber-optic link from the playback recorder (see Figure 1).

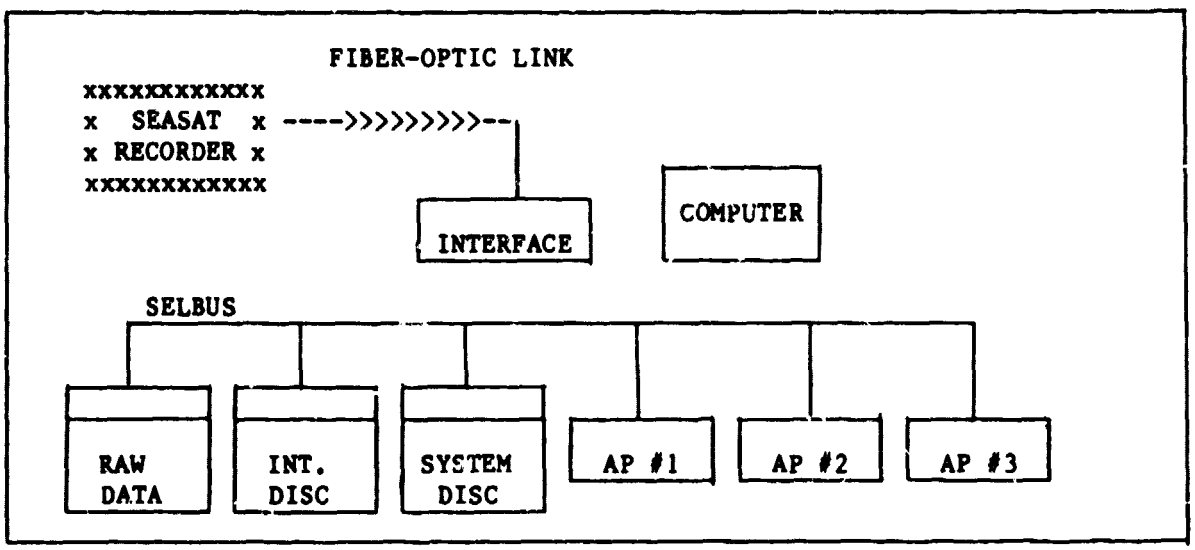


Figure 1. IDPMX Block Diagram with Key System Elements

This architecture was chosen for its conceptual simplicity and that it does not require any nonstandard interfaces. The parallel configuration involves all APs doing the same work and all of the work on different lines. Furthermore, the requirement of a common buffer accessible to all

APs rules out the direct AP-to-disc connection for the time being in the parallel setup.

The remedy seems to be the recreation of the general-purpose integer computer on the other side of the APs. Data needs to be collected in a common memory (because of algorithmic and efficiency reasons) before it is ready for IO to the disc or to the APs. This memory can be configured on the AP side. The data selection work that is now in the CPU could be done by intelligent IO controllers working on the common memory. (GPIOP by FPS is an example of such a device). The advantage of such a configuration is to off-load the host. But cost, ease of development, and flexibility would tend to favor making good use of and expanding the host facilities. As discs are the heart of the processing, the supplier of the best discs should be given weight in the decision of which common memory to configure.

The alternative is pipelining the APs so that portions of the work of one line are done serially by each AP. Each AP would perform a unique step in the pipelined setup. If the data source has limited distributive capabilities, the pipelined mode might be the only possible solution to speedup. An example would be an analog-to-digital converter whose output has to be quickly processed. If there is need for accumulation from different time periods then the last AP in the pipelined mode would automatically serve as the common buffer. Note that for very-short frame times, eliminating the host connection is desirable. This adds more to the requirements of the AP, though; essentially, you need an AP that is a self-sufficient minicomputer with operating system and interrupts.

Judging their relative merits, the parallel approach seems to be superior for our application, as shown in Table III.

TABLE III
Parallel versus Pipelined Configurations

1) Control protocol for the parallel AP processing is conceptually simple and requires no nonstandard interfaces.
2) Computational efficiency of the pipeline mode is dirty to try to optimize because the processing times of subfunctions are different.
3) Parallel AP configuration enhances the reliability and maintainability of the system.
4) Parallel AP configuration is more amenable to modular expansion.

Very-high uniformity, easy control, flexibility, and the general number of AP programmability characterize the parallel setup. A recent paper with a different viewpoint on multiple array processor configurations and related issues has come to our attention [18].

The advantages of the pipelined setup are minimizing hardware and solving the common data problem through specialization. The analogy to the tradeoffs involved in one two-stage-pipelined adder versus two nonpipelined adders is worth invoking. If the APs are specialized, then hardware is minimized but usefulness to other problems is severely limited. Common data requirements make the pipelined setup look good. An example would be interpolation done across records whereby a window slides down the data.

For the parallel setup, communication transfers are proportional to nap (the number of APs); for the pipelined setup, they are independent of nap.

Usage

The software organization of the radar job affects the general usability of the computer system. The main phases are relatively large (60-80 KW), utilize most of the hardware, and run a couple of minutes each. For IDP the executive routine was done as a main program that brought in the single level overlays. This practically tied up the computer for a whole run. For IDPMX an executive routine with a sequence of activations as a control mechanism was developed. There is a communication facility to pass parameters and to inform a phase of the executive that activated it; when a phase is done it resumes that executive. Automation, standardization, and reducing the space-time product were the goals. The implementation is a batch queue that allows cut-in by jobs in the same batch class. This way another radar run can cut in at an appropriate point when there is no loose data on the intermediate disc. Also, when there is heavy daily activity, the newly activated phase ends up in the allocation queue, thus making the system more usable to others.

As far as just the radar jobs are concerned, the system is monoprogrammed. This should not be seen as a deficiency; in fact, multiprogramming in this context is a pitfall to avoid. Other activity on the same two discs would cause thrashing and severe slowdown as a result of frequent head movement, though concurrent small jobs using the system disc present no problems. (This type of concern should influence job scheduling algorithms for mainframe operating systems).

VI. PERFORMANCE EVALUATION

Workload

The per-line workload of the phases is shown in Table IV.

TABLE IV
Per-Line Workload of the Phases

PHASES	RACR	CRAZ	AZOL	
Read	1.7	1.0	1.0	KW
Preproc	---	1.0	1.0	KW
Apput	.5	1.0	1.0	KW
Aprun	31.0	20.0	18.7	ms
Apget	.6	1.0	.3	KW
Postproc	.6	---	---	KW
Write	.6	1.0	.3	KW

Note:

1) RACR is loop within loop;
   in the outer loop is the big buffer write
2) CRAZ is loop within loop;
   in the outer loop is the big buffer read
3) AZOL is one loop;
   in the loop is a big buffer window

The programs were written with the number of APs to be used, nap, as a variable. By selectively bypassing sections of code in the main loop one can time subfunctions. Easily, by moving the wait statements around, one can simulate sequential execution. There is dramatic speed gain due to concurrent IO, CPU, and AP operation. All the phases can be modeled fairly accurately. Calculations are within 5 percent of the measured values shown in Table V (a detailed example is in the APPENDIX). The parallel 1 AP figures are faster than the original IDP due to increased disc and memory storage available on the IDPMX system.

TABLE V
Elapsed Time (in seconds) to Complete 2K Lines in each Phase on IDPMX as a Function of the Number of APs

MODES	RACR	CRAZ	AZOL
Sequential 1 AP	143.0	116.9	123.4
Parallel 1 AP	68.2	59.4	53.4
Parallel 2 AP	35.9	33.1	29.7
Parallel 3 AP	26.7	30.6	28.8

The elapsed time is not inversely proportional to nap. So the process is not AP limited. It means all sorts of additional features using AP can be added to the programs, only mildly affecting run time in the nap=3 case. As disc IO elapsed time is a step function of AP run time, the effect might even be nil.

Based on per-line workload, one calculates, as in Table VI, a percentage usage of subsystems that is normalized by the most time-consuming item.

TABLE VI
Percentage Usage with Respect to Total Time
in the 3 AP IDPMX System

PHASES	RACR	CRAZ	AZOL
Disc Input (elapsed)	93	80	100
AP IO Transfer	11	25	18
AP Processing	83	87	80
SEL CPU	61	73	95
SEL Bus Load	35	40	43
Disc Output (elapsed)	100	100	63
Disc Input (transfer)	52	55	55
Disc Output (transfer)	22	55	18

Note:

1) Disc elapsed percentages include latency and overhead.

2) CPU percentage does not include slowdown due to interference.

SEL bus load was calculated by averaging IO load and adding in a typical instruction CPU model. Assume loadword and storeword as typical instructions. This takes 2.4 μs and uses 6 bus cycles, as instructions are fetched in pairs. This results in a 10-MB/s bus load due to the CPU.

Interference

Slowdown due to IO interference applies both to the AP and the CPU. The effect of AP transfers on AP running is usually negligible. The worst case is 2 cycles per word transferred (the average case is close to the worst case). Given relatively large frame times (20 to 30 ms) and short (1 to 2 KW) transfers, ignoring this interference is justified. HSD transfer rate is quoted at 1.2 μs per word (in practice the measurements indicate 1.0 MW for APPUT and 1.4 MW for APGET). Still, on an interference per IO elapsed time basis the effect is 25 percent and could be a problem if very little processing needs to be done on the data. AP memory refresh causes a 3-percent slowdown (3 cycles every 15 μs). Host CPU slowdown due to bus and memory contention is a substantial effect. Per IO elapsed time, memory intensive CPU slowdown is 20 percent for APGET and 30 percent for APPUT (this timing was done on a 2-way interleaved SEL 32/77). For this reason we do not transfer to all the APs together in parallel. The combined load would effectively stop the CPU and this is intolerable in the case of issuing the next record transaction to the discs. The missed sector would result in a full revolution delay. As our frame times are considerably larger than transfer times to the APs, doing the transfers in sequence presents no efficiency loss. (In fact, the latest system configuration has all the APs daisy-chained on one HSD).

Disc Latency

The latency due to overlapping disc transfers with AP running is calculable because the process resynchronizes after a few repetitions. Elapsed time plotted against AP run time (fixed disc transfer size) is a step function. Assume no overhead and disc transfer of length D and an AP run of duration A in sector-pass-time (SPT) units. For concreteness 23 sectors per rotation is given. Also assume that we are on sector and all ready to proceed. For A <= D latency is 0 and elapsed time is D. For A >= D + 23 latency does not matter and elapsed time is A. In the middle range when D < A < D + 23 the step function behavior is exhibited. Define AT as the sector we are on, WAT as the sector we want to be on, LAT as the latency from AT to WAT (mod 23). ELAPS is the maximum of (A , D + LAT). WAT will increase by D from record to record. LAT is J * (D - A) mod 23, where J is the number of records already done since initial conditions. AT will increase by ELAPS = A until ELAPS = D + LAT (as 23 is prime, multiples of D - A will cycle the range). An example with D = 9 and A varied will suffice, as shown in Table VII.

TABLE VII
Disc Elapsed Time Step Function Behavior

0 <= A <= 9	avg.elaps = 9
10 <= A <= 20	41/ 2; avg.elaps = 20.5
21 <= A <= 28	85/ 3; avg.elaps = 28.3
29 <= A <= 29	233/ 8; avg.elaps = 29.12
30 <= A <= 30	361/12; avg.elaps = 30.08
31 <= A <= 31	713/23; avg.elaps = 31.00
32 <= A	; avg.elaps = A

The same calculations would not be valid for overlapping disc transfers with a CPU run. There is coupling between IO and CPU. Effective A is increased as IO slows down the CPU; this in turn affects the degree of slowdown dependent on how much of the latency coincides with the CPU run. Measurements still indicate a step function behavior. It would be interesting to try to model this. Another related question is the modeling of two independent disc IO streams from separate controllers overlapping each other and AP or CPU.

SARIO disc handler overhead is 0.5 ms for the initial call and 0.2 ms for each subsequent transaction issued as part of the software record chaining. As there is disc IO on two channels going simultaneously there is a possibility that two interrupts might appear during the same sector-pass-time (SPT = 0.7 ms). Slowdown due to other IO and additional overhead due to context switching will aggravate the situation. Though two 0.2 ms are likely to fit in an SPT, this concern highlights the importance of optimizing handler operation. Cases when one or both of the disc items to be done is the initial call can occur in our application. The possibility of doing a seek as part of the record chain, especially in the software cylinder formatted cornerturn where latency after seek is zero, complicates the assessment. There might be some asymmetry depending on relative channel interrupt priority ranking.

Future Considerations

In all phases about half the CPU time is associated with overhead. The other half is the work spent on necessary data selection buffer movements per line. The type of function "move multiple long with increment," if implemented as an instruction in the Writable Control Store (WCS), will halve the bus demand; if this type of function is implemented as IO in the Regional Processing Unit (RPU), it will also free the CPU. Data selection is best done in a RPU-based host-multiple AP interface as that boundary is crossed immediately before or after the selection in any case. This setup is desirable because the slight bus load is a good tradeoff for general utility of the host memory. If the number of lines per AP is doubled, CPU percentage usage will decrease as the overhead stays constant. But the denser environment will increase the CPU slowdown factor. Memory bandwidth can be increased to the point where it is equivalent to the bus bandwidth by additional memory bus controllers and interleaving. Another gain can result from optimizing the AP software interface.

Comparing disc elapsed to disc transfer times will show the importance of latency. By doubling the number of lines per transaction, latency can be drastically reduced. Nonfull sectors and the in-between sector can be avoided with discs that have power-of-two sector size and allow hardware record chaining. A software cylinder formatting an improved cornerturn algorithm with bigger buffers and discs can be implemented.

Disc transfer rates, using present discs, can be attacked by going to a four-channel, four-disc, split-file configuration. A file would be split across two discs, and these discs would be read simultaneously. The issues of organization, latency, and multiple-interrupts in an SPT make this a speculative attempt.

With the existing hardware, 15-percent potential improvement in the main phases is estimated. With new hardware, up to 45-percent speedup is possible (SEL 32/87 with 32-KB cache, 256-KW memory, 600-MB discs, and faster APs with 64-KW memory each. Actually that powerful a computer is unnecessary if intelligent AP interfaces can be developed as outlined above. Faster APs are not strictly required either as five present ones will do). With optimized overlay this would mean a sub-one-hour processing time per picture.

There is an ongoing activity to cut the processing time to 1.5 hours [25]. This is to be done on the existing system augmented with a fourth AP. The software effort involves the improved cornerturn, optimized APEX, twice the number of lines per disc IO, and all of the peripheral phases redone for speed.

The approach taken in implementing the IDP and IDPMX projects is software intensive. Development effort and programming difficulty are considerably more than what they would be for achieving comparable throughput on a supercomputer. But supercomputer prices are prohibitive. The operating system, sophisticated compiler, and the fast hardware of a supercomputer would automatically do partially what in our approach has to be done manually. But the full capabilities of a supercomputer would not be utilized. The parallel pipelined programming methodology can also be applied to supercomputers and their IO devices. Exploiting independent operation of subsystems by software control from one job is a valid approach for any problem where speed is needed.

VII. CONCLUSION

An efficien. SEASAT SAR processing system and the methodology of parallel data handling among the discs, CPU, and APs in a host plus peripheral array processor environment have been demonstrated. The relevant programming and efficiency issues have been analyzed. Quantitative discussion of overhead, latency, and congestion has been made. The design approach is general enough that the tools developed and implemented are applicable in different contexts.

VIII. REFERENCES

[1] C. Wu, "A Digital System to Produce Imagery from SAR Data", Proc. AIAA Systems Design Driven by Sensors Conf., Paper 76-968, New York, N.Y., October 1976.

[2] C. Wu, "A Digital Fast Correlation Approach to Produce SEASAT SAR Imagery", Proc. IEEE 1980 International Radar Conf., pp. 153-160, New York, N.Y., April 1980.

[3] C. Wu, B. Barkan, B. Huneycutt, C. Leang, S. Pang, "An Introduction to the Interim Digital SAR Processor and the Characteristics of the Associated SEASAT SAR Imagery", JPL Publication 81-26, Jet Propulsion Laboratory, Pasadena, Calif., April 1981.

[4] B. Barkan, C. Wu, W. Karplus, D. Caswell, "Application of Parallel Array Processors for SEASAT SAR Processing", Proc. IGARSS 1981, Vol. I, pp. 541-547, New York, N.Y., June 1981.

[5] C. Wu, B. Barkan, W. Karplus, D. Caswell, "SEASAT Synthetic Aperture Radar Data Reduction Using Parallel Programmable Array Processors", IEEE Trans. Geoscience and Remote Sensing, Vol. GE-20, No. 3, pp. 352-358, July 1982.

[6] Systems Engineering Laboratories, "Disc Processor Model 9024 Reference Manual", Report No. 301-329024-000, Fort Lauderdale, Fla., November 1979.

[7] Systems Engineering Laboratories, "SEL 32 Reference Manual", Report No. 324-322000-001, Fort Lauderdale, Fla., August 1975.

[8] Floating Point Systems, "FPS Processor Handbook", Report No. 860-7259-003, Portland Ore., February 1979.

[9] S. Pang, "Interim Digital SAR Data Processing (IDP) Programs Users Guide and Documentation", Jet Propulsion Laboratory, Pasadena, Calif., October 1979, (unpublished JPL document).

[10] Floating Point Systems, "AP Math Library Manual", Report No. 860-7288-004, 3 volumes, Portland, Ore., February 1979.

[11] Floating Point Systems, "Programmers Reference Manual", Report No. 860-7319-000, 2 volumes, Portland, Ore., January 1978.

[12] B. Barkan, "Fast Unpack Routines for AP-120B", Proc. 1981 ARRAY Conf., pp. 237-241, Floating Point Systems, Portland, Ore., April 1981.

[13] Floating Point Systems, "VFC The Vector Function Chainer", Report No. 860-7351-004, Portland, Ore., September 1979.

[14] Floating Point Systems, "APEX Manual", Report No. 860-7371-003, Portland, Ore., December 1979.

[15] D. Caswell, B. Barkan, "A Multiple-AP Apex", Proc. 1981 ARRAY Conf., pp. 242-244, Floating Point Systems, Portland, Ore., April 1981.

[16] B. Barkan, C. Wu, "Transpose of Externally Stored Matrices", Proc. 1982 ARRAY Conf., pp. 179-185, Floating Point Systems, Portland, Ore., March 1982.

[17] B. Barkan, S. Pang, "Transpose of Externally Stored Matrices - II", Proc. 1983 ARRAY Conf., pp. 107-114, Floating Point Systems, Portland, Ore., April 1983.

[18] J. Burns, "Greater Throughput with Multiple Array Processors", Computer Design, pp. 207-211, September 1981.

[19] C. Elachi, T. Bicknell, R. Jordan, C. Wu, "Spaceborne Synthetic-Aperture Imaging Radars: Applications, Techniques, and Technology", Proc. IEEE, Vol. 70, No. 10, pp. 1174-1209, New York, N.Y., October 1982.

[20] J. Gredel, J. Guignard, "Digital Ground Preprocessing of SAR Data", Proc. ISPRS, International Archieves of Photogrammetry, Vol. 24-II, pp. 479-490, Ottawa, Canada, September 1982.

[21] C. Wu, B. Barkan, J. Curlander, M. Jin, S. Pang, "Current Performance of the Interim Digital SAR Processor", in SIR-A Symposium Proceedings, JPL Publication 83-11, Jet Propulsion Laboratory, Pasadena, Calif. (in press).

[22] M. Jin, C. Wu, "A Digital Quick-Look SAR Processor", Proc. Inter. Society of Photogrammetry Conf., Ottawa, Canada, 1982.

[23] J. Curlander, C. Wu, A. Pang, "Automated Preprocessing of Spaceborne SAR Data", Proc. IGARSS, Vol. II, FA-1, pp. 3.1-3.6, IEEE, New York, N.Y., June 1982.

[24] J. Curlander, S. Pang, "Geometric Registration and Rectification of Spaceborne SAR Data", Proc. IGARSS, Vol. II, FA-2, pp. 5.1-5.6, IEEE, New York, N.Y., June 1982.

[25] B. Barkan, S. Pang, B. Lewis, A. Pang, "Speed Upgrade of IDPMX Digital SAR Processor", Jet Propulsion Laboratory, Pasadena, Calif. (in preparation).

APPENDIX A

Program Listings

This appendix contains program fragments to highlight the optimizations done in several languages and to illustrate the parallel programming methodology at the device level. All the examples are taken from the first phase, RACR. Listing 1 is the invocation of SETAP to establish the logical-to-physical correspondence at array processor assignment time, which is done once at the top of each phase. Listing 2 is the loop of the 4-bit unpack routine, VUP4C, coded in APAL. It shows a technique of branching that uses two inputs delivered in FA and FM after the branch. Listing 3 is an example of a Vector Function Chained list of AP calls, RACRX, that does the range processing. Listing 4 is the first-stage cornerturn data movement subroutine, RACRM, coded in SEL assembler. Listing 5 is the main loop of the phase RACR. This fragment incorporates all of the above. It also illustrates the usage of SARIO and the way first-stage cornerturn has been integrated into the processing. This main loop is preceded by a buildup and followed by a wind down in the complete code.

Listing 1

Logical-to-Physical Correspondence of APs

```
C
C     ******
C
      NAP=0
      DO 1 LAP=1,NAPDESIR
      CALL SETAP(LAP)
      CALL APINIT(0,0,IAPSTS)
      IF(IAPSTS.GT.0) NAP=NAP+1
1     CONTINUE
      IF(NAPDESIR.NE.NAP) CALL EXIT
C
C     *******
C
```

Listing 2

The Loop of 4-bit Unpack Routine

```
C
C     *******
C
                                        "1.
L:    DEC S; SETMA;                     FADD TM,DPX(CO);
                                        MUL TM,FA
                                        "2.
                                        "B2 IN FM
                                        ADD EB,E; DPY(B1)<SPFN;
                                        WRTMAN;
                                        FMUL TM,FA
                                        "3.
                                        "B2 IN FA
                                        SUB I8,D; SETMA; MI<FM;
                                        FADD DPX(F16),FA;
                                        FMUL TM,FA;
                                        BFGE P
                                        "4.M
M:    LDSPE E; DB=MD; DPX<MD;           DECMA; MI<FM;
                                        FADD ZERO,DPY(B1);
                                        FMUL TM,FA
```

```
                                        "5.M
                                        "B2 FROM FA
     FAND DPY(M8),MDPX; MOV C8,C8;      DECMA; MI<FA;
                                        FMUL;
                                        BFGE DONE
                                        "6.M
     FAND DPY(M7),MDPX; MOV C7,C7;      DECMA; MI<FA;
                                        FMUL;
                                        BR C
                                        "4.P
P:   LDSPE E; DB=MD; DPX<MD;            DECMA; MI<FM;
                                        FADD ZERO,DPY(B1);
                                        FMUL TM,FA
                                        "5.P
                                        "B2 FROM FM
     FAND DPY(M8),MDPX; MOV C8,C8;      DECMA; MI<FM;
                                        FMUL;
                                        BFGE DONE
                                        "6.P
     FAND DPY(M7),MDPX; MOV C7,C7;      DECMA; MI<FA;
                                        FMUL;
                                        BR C
```

```
                                                    "7.
C:    FAND DPY(M6),MDPX; MOV C6,C6;                 DPX(CO)<FM;
      DECMA; MI<FA
                                                    "8.
      FAND DPY(M5),MDPX; MOV C5,C5;
      DECMA; MI<FA
                                                    "9.
      FAND DPY(M4),MDPX; MOV C4,C4;
      DECMA; MI<FA
                                                    "10.
      FAND DPY(M3),MDPX; MOV C3,C3;
      DECMA; MI<FA
                                                    "11.
      FAND DPY(M2),MDPX; MOV C2,C2;
      FMUL TM,FA;
      BR L
C
C     *******
C
```

Listing 3

Range Compression in the AP

```
C
C     *******
C
      DEFINE RACRX(BL,SU,IR)
      LOCAL SU4,IR12,IR14
      SU4=4096-SU
      IR12=12288+IR
      IR14=14336+IR
      IF BL = 2 GOTO EVEN
ODD:  CALL VUP4C(0000,SU4,513)
      CALL RFFTB(4096,8192,4096,1)
      CALL RFFTSC(8192,4096,2,0)
      CALL CVMUL(IR12,2,08192,2,6144,2,1024,1)
      CALL CVMUL(IR14,2,10240,2,4096,2,1024,1)
      CALL CFFTB(4096,8192,2048,-1)
      CALL VCLIP(8192,1,4080,4081,8192,1,2560)
      CALL VPK8(8192,1,2048,1,640)
      GOTO DONE
```

```
EVEN: CALL VUP4C(1024,SU4,513)
      CALL RFFTB(4096,8192,4096,1)
      CALL RFFTSC(8192,4096,2,0)
      CALL CVMUL(IR12,2,08192,2,6144,2,1024,1)
      CALL CVMUL(IR14,2,10240,2,4096,2,1024,1)
      CALL CFFTB(4096,8192,2048,-1)
      CALL VCLIP(8192,1,4080,4081,8192,1,2560)
      CALL VPK8(8192,1,3072,1,640)
DONE: END
C
C     *******
C
```

Listing 4

Cornerturn Data Movement Routine

```
C
C     *******
C
$EXECUTE   ASSEMBLE
           PROGRAM  RACRM
           DEF      RACRM
           M.REQS
*
*          IAD,IBH,NRB,NCH . . PARAMETERS
*          ASSUMES:
*          NRB . . EQUAL TO 64
*          NCH . . DIVISIBLE BY 8
*
RACRM      TRR   R0,X1
           LEA   X2,*1W,X1
           LEA   X3,*2W,X1
NRB        EQU   64
           LNW   R1,*4W,X1
```

```
LOOP    LW    R4,0D,X2
        LW    R6,1D,X2
        STW   R4,1*NRB,X3
        STW   R5,3*NRB,X3
        STW   R6,5*NRB,X3
        STW   R7,7*NRB,X3
        SRLD  R4,16
        SRLD  R6,16
        STW   R4,0*NRB,X3
        STW   R5,2*NRB,X3
        STW   R6,4*NRB,X3
        STW   R7,6*NRB,X3
        ADI   X2,2D
        ADI   X3,8*NRB
        BID   R1,LOOP
DONE    ADI   R0,5W
        TRSW  R0
        END
C
C       *******
C
```

Listing 5

Main Loop of Phase RACR

```
C
C     *******
C
      DO 1000 L=LREDB,LREDE,NAP
C
C
      DU=BI ; BI=BL ; BL=DU
C
      CALL SARIO(R,F1,1728,NAP,IC(1,1,BL))
      DO 52 LAP=1,NAP
      CALL WSU(SU); CALL WIR(IR); CALL SETAP(LAP)
52    CALL RACRX(BL,SU,IR)
C
      DO 51 LAP=1,NAP
      CALL WSW(SW); CALL SETAP(LAP)
      CALL APPUT(IC(SW,LAP,BI),P(BI),513,0)
      FU=FU+1 ; OV=OV+1
      CALL RACRM(IAD(1,LAP,BL),IBH(FU,BBL),NRB,NCH)
      IF(FU.LT.FULL) GO TO 100
C     ASSUMES FULL DIVIDES OVLP
      CALL SARIO(D,F2,512,COMPCODE)
      IF(OV.LT.OVLP) GO TO 90
```

```
C       CONFIGURE NEW WRITING SPACE
        CALL SARIO(C,F2,512)
        F2SP=F2SP+LUFF*RECW
        CALL SARIO(O,F2,512,F2SP,WSKIP,LUFF,FF2)
        OV=0
90      CONTINUE
        DU=BBI ; BBI=BBL ; BBL=DU
        CALL SARIO(W,F2,512,RECW,IB(1,BBI))
        FU=0
100     CONTINUE
        CALL APGET(IA(1,LAP,BI),G(BI),640,0)
51      CALL APWD
C
        DO 53 LAP=1,NAP
        CALL SETAP(LAP)
53      CALL APWR
        CALL SARIO(D,F1,1728,COMPCODE)
C
C
1000    CONTINUE
C
C       *******
C
```

APPENDIX B

Performance Modeling Example

This appendix (including Tables VIII, IX, and Figure B-1) is a detailed, worked-out example of modeling the performance of the RACR phase.

TABLE VIII
Definitions of Throughput Modeling Terms

ap.ov	=	overhead per call to AP
apput.tr	=	transfer time to AP
cpu	=	cpu work overlapped with apput
cpu.el	=	cpu with slowdown
apput.el	=	elapsed time for apput
apget.tr	=	transfer time from AP
apget.el	=	elapsed time for apget
apio.el	=	elapsed time for all AP IOs
aprun	=	processing time in an AP
aprun.el	=	aprun with slowdown
aprun.av	=	available time for overlapping
ap.el	=	elapsed time for all APs
disc.tr	=	disc transfer time including extra sector
disc.el	=	disc elapsed time for overlapping with ap
total.el	=	total elapsed time in given loop

TABLE IX
Throughput Modeling Formulas

apput.el	=	max (apput.tr , cpu.el) + 2 * ap.ov
apget.el	=	apget.tr + 2 * ap.ov
apio.el	=	nap * (apput.el + apget.el)
aprun.av	=	aprun.el - (nap - 1) * ap.ov
ap.el	=	max (apio.el, aprun.av) + 2 * nap * ap.ov
disc.el	=	calculate with (in SPT units)
		D = nap * disc.tr
		A = ap.el
total.el	=	max (ap.el , disc.el)

ORIGINAL PAGE IS
OF POOR QUALITY

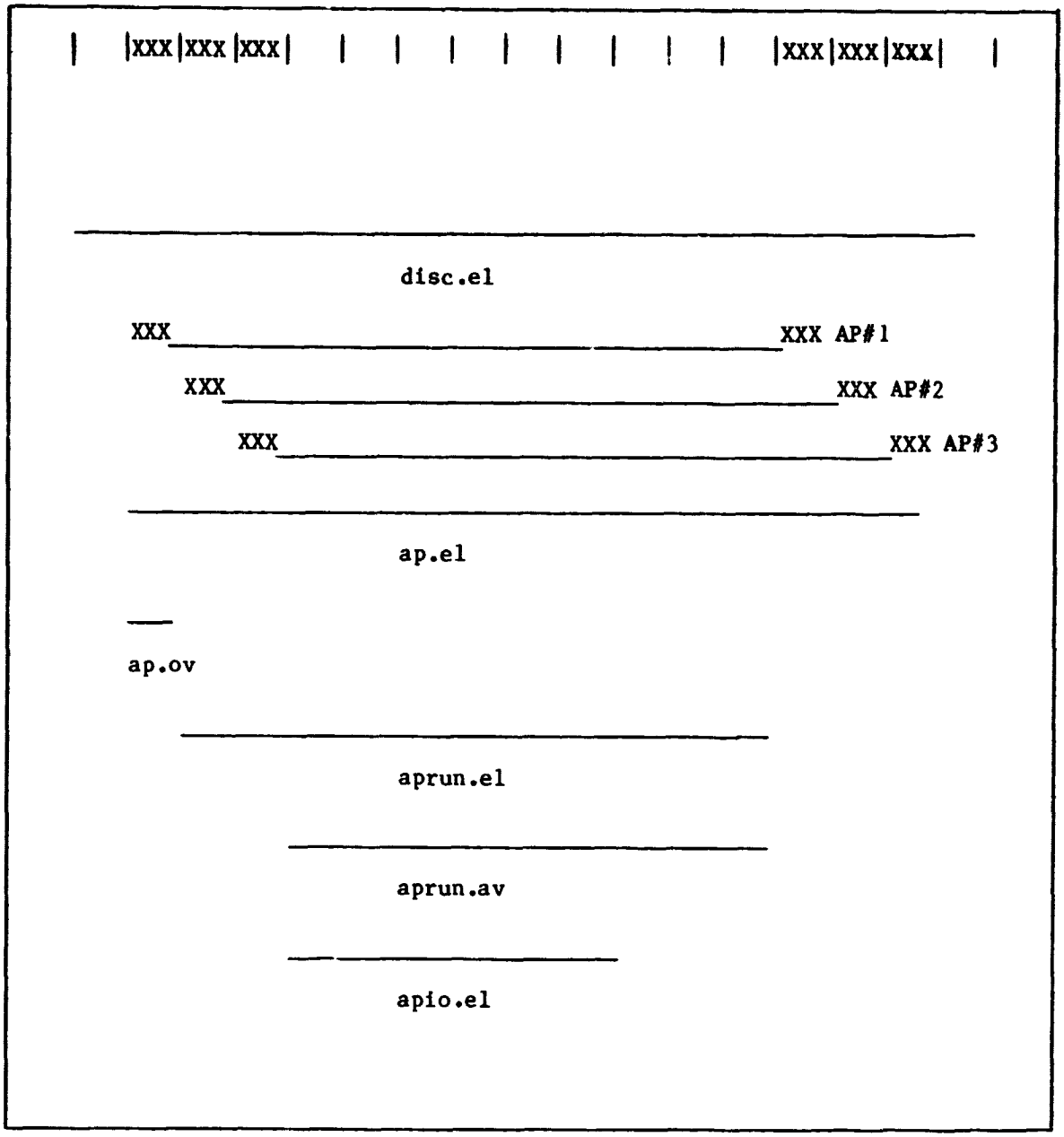


Figure B-1. Innerloop Timeline for nap = 3 in Phase RACR

Worked out Numerical Example

ap.ov = 1.1 ms (APEX per call)

apput.tr = 0.5 ms (1.0 MW per s)

cpu = 3.3 ms (5.4 μs per word, see RACRM)

apget.tr = 0.6 ms (1.0 MW per s when with contention)

aprun = 31.0 ms (see RACRX)

aprun.el = aprun + 2 AP cycles of slowdown per word transferred

= 31.0 ms + 0.3 μs per word * 1.1 KW

= 31.3 ms

(The above 31.3 ms is used in the succeeding pages. Correct aprun.el is 32.2 ms, taking into account the 3% slowdown from AP memory refresh. This inaccuracy is minor and does not cause any qualitative discrepancy).

cpu.el = portion of cpu under the APPUT and two discs (0.3 ms) + portion of cpu under just the two discs (3.0 ms)

(cpu slowdown factors are guesses)

= 0.3 ms * 1.5 + 3.0 ms * 1.1

= 3.8 ms

Assumptions:

1) Aprun and apio overheads same

2) Control code takes little time

3) SARIO overhead is hidden

Consult per-line workload in PERFORMANCE EVALUATION and the listings in the APPENDIX.

ORIGINAL PAGE IS
OF POOR QUALITY

apput.el = max (0.5 , 3.8) + 2 * 1.1

= 6.0 ms

apget.el = 0.6 + 2 * 1.1

= 2.8 ms

apio.el = 3 * (6.0 + 2.8)

= 26.4 ms

aprun.av = 31.3 - (3 - 1) * 1.1

= 29.1 ms

ap.el = max (26.4 , 29.1) + 2 * 3 * 1.1

= 35.7 ms

disc.el = calculation with (SPT units)

D = 3 * (9 + 1) = 30

A = 35.7 / 0.73 = 49

using 30 = 7, 49 = 3 (mod 23)

AT	0	3	6	9	12	15
WAT	0	7	14	21	5	12
LAT	0	4	8	12	16	20
ELAPS	49	49	49	49	49	50

ELAPS = max (LAT+D , A)

so avg.elaps = 49.2 SPT

= 35.9 ms

total.el = max (35.7 , 35.9)

= 35.9 ms (for 3 lines)

This corresponds to 24.0 s for 2K lines; as the existing cornerturn takes 26.2 s, just the cornerturn time will show. The calculated 26.2 s is in good agreement with the measured 26.7 s.

Speed Estimate for a Faster RACR

Assumptions:

1) ap.ov = 0.2 ms (software interface optimization)

2) cornerturn twice as fast (software cylinder formatting)

3) less disc input latency (twice the lines read in)

4) nap = 4 (buy fourth AP)

apput.el = 4.2 ms

apget.el = 1.0 ms

apio.el = 20.8 ms

aprun.av = 30.7 ms

ap.el = 32.3 ms

disc.el = 33.4 ms

calculate with (in SPT)

D = 80, A = 88

using 80 = 11, 88 = 19 (mod 23)

AT	0	19
WAT	0	11
LAT	0	15
ELAPS	88	95

so avg.elaps = 91.5 SPT (for 8 lines)

total.el = 33.4 ms (for 4 lines)

This corresponds to 16.7 s per 2K lines, as the improved cornerturn will take only 15.3 s and will be overlappable with the above. Note that the same first three assumptions with nap=3 give ap.el = 32.1 ms, D=60, A=88; so total.el = 32.1 ms (for 3 lines). This is 21.4 s per 2K lines. The nap=3 versus nap=4 comparison shows the fourth AP cost-effective provided our assumptions are realizable. QED

APPENDIX C

Cornerturn Papers

For completeness, two papers on cornerturn are included here.

Proceedings of the 1982 ARRAY Conference

TRANSPOSE OF EXTERNALLY STORED MATRICES*

by

Budak Barkan**, Chialin Wu

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109

ABSTRACT

This paper presents a method for transposition of arbitrary matrices that are larger than available primary storage. Advantages of the approach compared to previously reported results are flexibility and efficiency. Input matrix dimensions are unrestricted and the implementation is very fast. This transposition method should be of interest to those carrying out 2D Fast Fourier Transforms on FPS Array Processors.

INDEX TERMS

Matrix transpose, 2D transforms, Corner turning.

INTRODUCTION

In several applications, like 2D transforms, an efficient algorithm for matrix transposing is needed. Because the size of the matrix is much greater than the available high-speed primary storage, the operation requires careful buffer management and IO planning for the secondary storage device.

This is a two-stage operation; each stage requiring a big buffer in core. In the first stage data is read in line by line and put into big buffer 1. This buffer is written out in records whenever full. The above is repeated until the whole matrix has been read in and written out. In the second stage enough records are read in to fill big buffer 2. From this buffer lines are formed and written out. This is repeated until all lines have been formed.

Skip-write, sequential-read is one way of implementation. A logical but not speedwise equivalent way would be sequential-write, skip-read.

*This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS7-100, sponsored by the National Aeronautics and Space Administration.

**Part of the analysis was done while B. Barkan was a student at Computer Science Department, UCLA.

ORIGINAL PAGE IS
OF POOR QUALITY

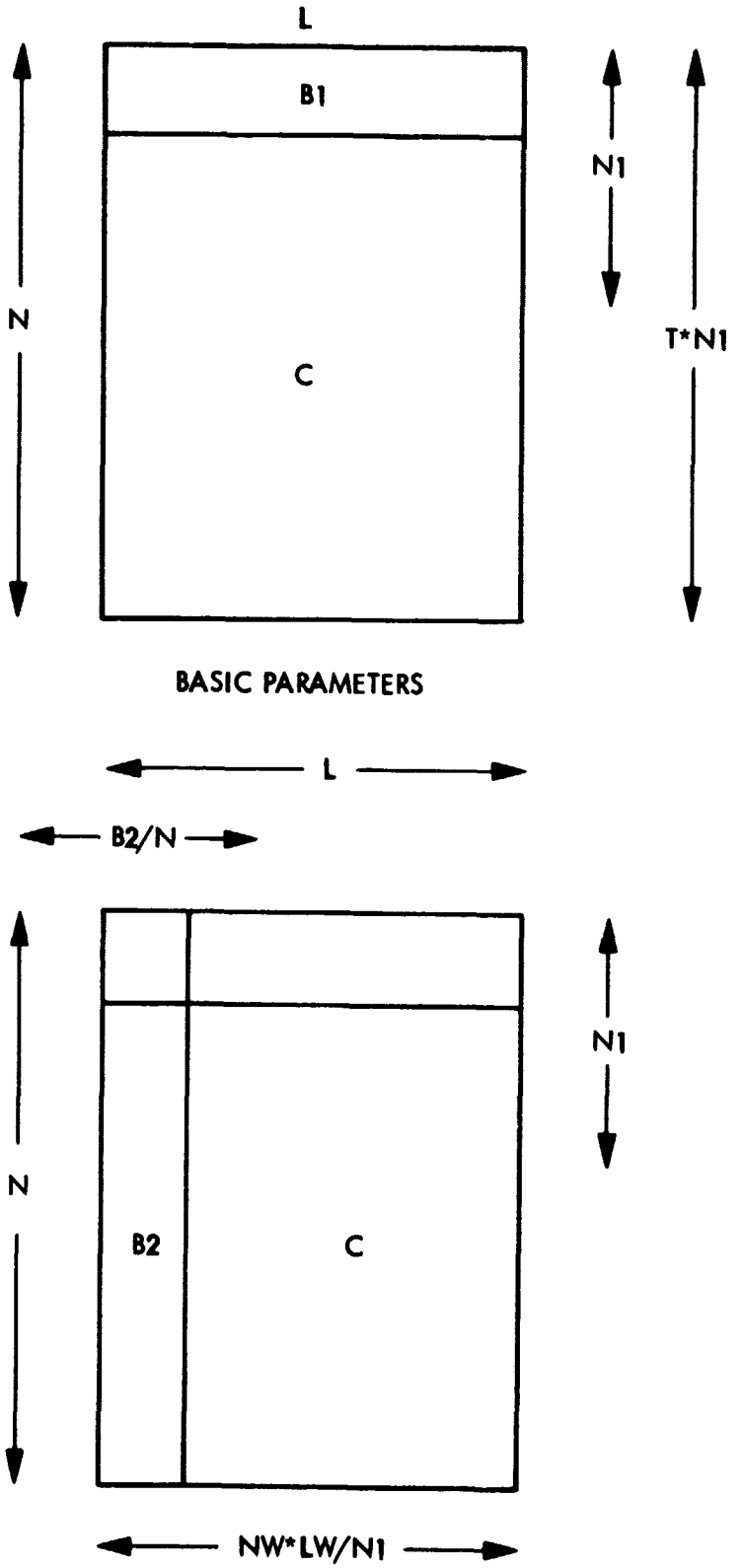


Figure C-1. Basic Parameters in the Transpose Operations

BASIC PARAMETERS (Figure C-1)

Input	N lines of length L each
C = N * L	Input matrix size
B1	First stage big buffer size
B2	Second stage big buffer size

(All below divisions should yield zero remainder)

N1 = B1 / L	Number of lines that fit into B1
T = C / B1	Number of segments in one column
LW = B2 / T	Line length in writing out B1
NW = B1 / LW	Number of lines to write out B1

The idea is to write out B1 with records as big as possible and have T of them fit B2. When B2 is full, B2 / N lines are ready to be formed.

Total write count	= Total read count
	= T * NW
	= (C*C) / (B1*B2)

PROCEDURE

Each phase is done until all "input" is exhausted:

Phase 1

A line is put into B1 point by point with spacing N1 in B1 for adjacent points in the line.

When full, B1 is written out as NW records of length LW each, with spacing T records on a disk in between.

Phase 2

T records are read in filling B2. Then complete lines are assembled, one line from T segments each of length N1.

ADVANTAGES OF THE ALGORITHM

1) Input matrix dimensions are relatively unrestricted. The matrix does not have to be square or power of two on one side. (The amount of padding required for the "odd" cases is minimal.)

2) Primary storage address calculations are of the trivial increment-fetch, add-store variety.

3) Data input and output are line by line so the transpose operation can be combined with what is being done before and after the transpose.

4) Input data is going to be processed with overlap; the first stage of the transposing operation need not be repeated.

IMPLEMENTATION

The highly regular nature of the IO operation makes it a good candidate for optimization. Asynchronous IO programs with multiple record capability for sequential and skipped modes were written for two dedicated 300-MB disks on different channels. This approach makes a seek a rare event, and latency occurs only once per multiple record. IO elapsed time becomes close to transfer times.

The implementation was completed in 1978 as part of the Interim Digital Synthetic Aperture Radar Processing System (IDP) developed at the Jet Propulsion Laboratory (Refs. C-1 through C-3). The computer was a SEL 32/55 running under RTM.

COMPARISON

The approach of Eklundh (Ref. C-4) is numerically compared to ours on similar input sizes:

	Eklundh		Ours	
Input	512 * 512	complex	1024 * 1280	halfwords
Equivalent input	512 * 1024	words	1024 * 640	words
Buffer size	16 KW, 16 KW n = 9, j = 4		20 KW, 16 KW	
IO passes	2		1	
IO count	2048	records	2560	records
Time CPU	7	s	5	s
Time IO	108	s	17	s
Total time	115	s	22	s

Our times given are the extra times just to do one transpose. (Matrix has to be read in and written out in any case.) Note that Eklundh would have to treat 1024 * 640 as 1024 * 1024.

DISK PARAMETERS

1

300 MB

SC	sectors/cylinder	437
	data surfaces	19
SS	sectors/surface	23
	rotation speed	3600 rpm
SPT	sector pass time	.73 ms
	words/sector	192
	word pass time	4 μs
	next cylinder seek	7 ms
ESC	effective sectors/cylinder	436

In this paper SPT is used as a time unit. It is a natural time unit in the present context, e.g.: the next cylinder seek is 10 SPT.

Notice that ESC (chosen for convenience as a multiple of effective record length) is a software construct and can be manipulated. (This idea is also useful in real-time IO buffering.)

ANALYSIS

RACR is our program that does the skip-write portion of cornerturn. In RACR, 40 512-word records are written with 32 record skips. A 512-word record is 3 sectors. So the start-to-end sector difference is 40*32*4 = 5120 sectors. (Notice the extra sector in effective record length is provided for reading efficiency.) This translates to 12 cylinders for the 300-MB disc.

Therefore, per 40-record skipped write:

	300 MB
no seek	28
one-cylinder seek	12
two-cylinder seek	--

Latency time calculations are tricky: the idea is to calculate which sector will be under the heads after the seek (if any) and to compare it with the next destination sector. Latency should be big enough to issue the necessary IO commands. The assumption is that 1+ sector is enough to issue a seek and a transaction.

In the following discussion, we assume the IO command is already issued, all ready to go; we are positioned just before the record.

```
    L  - present location
    R  - effective record length
   AT  - location you will be at
  WAT  - location you want to be at
 SEEK  - seek time; zero if no seek
 NYCL  - number of cylinders sought; zero if no seek
  LAT  - latency
ELAPS  - elapsed time for a record
 SKIP  - number of records to skip

   AT  = L+R+SEEK
  WAT  = L+R*SKIP+(SC-ESC)*NCYL

  LAT  = (WAT-AT) mod SS
ELAPS  = R+SEEK+LAT
```

for 300 MB:

no seek	13
one cyl. seek	14
Total:	532 = 28*13+12*14

Observed times are for a 12288-line RACR run (the scaling factor is 384*.73, because one block of 40 lines comes out of every 32 RACR lines).

	300 MB
estimated	149 s
observed	157 s

That we are underestimating is consistent because the very first record's seek time and latency time are worse (20 + 11 SPT additional).

IMPROVED ALGORITHM

An essential feature is that what is written does not get read the same way. Reading is done in 32-record sequential reads. The improved algorithm requires minor changes in the reading method.

A cylinder will be partitioned into 3*138 sectors, starting from sector zero. Writing in the first pass, the first record will go to sector 0; second record to sector 138+4; third record to 276+8. Fourth record will go to next cylinder sector 0, etc. In the second and latter passes, the records go to the "next" positions. This is equivalent to SKIP = 35.5. (In practice we implemented an approximation with SKIP = 36, as our present software does not allow fractional R skips.) Notice the resulting blocks can be read sequentially. Disk utilization efficiency will be 384/436 = 88% of usual, but overall address determination will be quite simple. This formatting scheme is shown in Figure C-2.

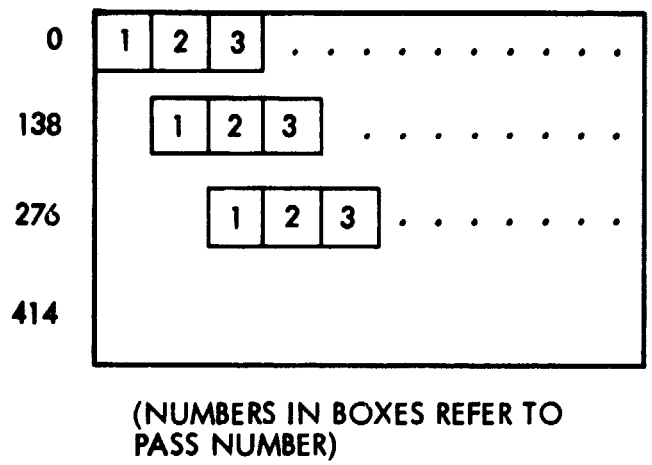


Figure C-2. Software Cylinder Formatting for Improved Algorithm

Latency time for the seek is:

```
 AT  = L+4+10
WAT  = L-8
LAT  = 1
```

Latency for no seek is:

```
LAT  = 0
```

So total time is 314 = 26*4+14*15, which compares very favorably to 532.

CONSIDERATIONS

The software cylinder formatting idea is generally applicable. If one is not lucky on the latency time for the seek, nonrepetitive addressing across cylinders can be used. Command/data chaining without next sector miss, power-of-two sector size, larger discs, and larger buffers will all facilitate faster implementations.

A further efficiency gain, at the cost of software complications, results from coupling the cornerturn with the transform operations. In that case, a direct array processor-to-disc connection (like that provided by FPS) looks particularly attractive.

CONCLUSION

With an IO routine allowing multiple-record transactions in sequential/skipped modes and software cylinder formatting, almost all latency can be masked from the cornerturn operation.

ACKNOWLEDGMENT

The authors gratefully acknowledge S. Pang for her work on the disc IO routine.

REFERENCES

(C-1) C. Wu, B. Barkan, B. Huneycutt, C. Leang, and S. Pang, "An Introduction to the Interim Digital SAR Processor and the Characteristics of the Associated SEASAT SAR Imagery," Publication 81-26, Jet Propulsion Laboratory, Pasadena, Calif., April 1981.

(C-2) B. Barkan, C. Wu, W. Karplus, and D. Caswell, "Application of Parallel Array Processors for SEASAT SAR Processing," Proc. IGARSS '81, Vol. I, pp. 541-547, IEEE, New York, N.Y., June 1981.

(C-3) C. Wu, B. Barkan, W. Karplus, and D. Caswell, "SEASAT Synthetic Aperture Radar Data Reduction Using Parallel Programmable Array Processors," submitted to IEEE Transactions on Geoscience and Remote Sensing.

(C-4) J. O. Eklundh, "A Fast Computer Method for Matrix Transposing," IEEE Trans. Comput., Vol. C-21, pp. 801-803, July 1972.

Proceedings of the 1983 ARRAY Conference

TRANSPOSE OF EXTERNALLY STORED MATRICES II*

by

Budak Barkan and Shirley Pang

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109

ABSTRACT

This paper presents a method for transposition of arbitrary matrices that are larger than available primary storage. The algorithm utilizes special purpose IO routines to facilitate the transposition operation as skip-write, sequential-read. The intermediate file structure is arranged to mask out latency. This software cylinder formatting technique is demonstrated to yield an efficient implementation. Timing results as a function of matrix and buffer sizes are tabulated.

INTRODUCTION

An algorithm for out-of-core cornerturn and its performance analysis was reported by the authors in Ref. C-5. This is a companion paper and assumes the terminology of the earlier one. Here, the emphasis is on the software cylinder formatting strategy that is used in conjunction with the skip-write. The implementation was done as part of the IDPMX Digital SAR Processor (Refs. C-6 and C-7).

MOTIVATION

Given the need to skip some fixed number of records repeatedly, one defines and optimizes a skipped access mode. The approach is software cylinder formatting to mask out latency. Additional padding, ALPHA, is appended to a block to make latency-within-cylinder zero. A relative starting sector offset, based on DELTA, is applied to cylinders to minimize latency-after-seek. The overall performance gain from these manipulations can be up to an order of magnitude, depending on the record size and skip distance. Some disc utilization efficiency is traded off for speed.

*The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology under contract with the Information Systems Office of National Aeronautics and Space Administration.

DEFINITIONS

NSB		number of sectors per block
ALPHA		additional sectors to skip
NBC		number of blocks per cylinder
DELTA		starting sector offset

NSB = T * ERL

ALPHA = ceil (NSB , SS) + ERL - NSB (mod SS)

NBC = quot (SC-SS , NSB+ALPHA)

```
if rem (SC-SS , NSB+ALPHA) >= NSB
   then NBC=NBC+1

if NBC=1
   then ALPHA=0
```

DELTA = (NBC-1) * (NSB+ALPHA) + SEEK + ERL (mod SS)

ESC = NBC * NSB

where

ceil (a,b) is least multiple of "b" greater than or equal to "a"

quot (a,b) is integer quotient of "a" divided by "b"

rem (a,b) is remainder of "a" after integer division by "b"

R-prefix refers to the quantites associated with the return from the last cylinder to the first, to start a new chain of skip-writes
these are namely RLAT, RSEEK, RELAPS, and RNCYL

RNCYL = (NW-1) / NBC

ASSUMPTIONS

1) A block is a contiguous set of T records.
2) Integral number of blocks per cylinder.
3) Last block in cylinder does not have associated ALPHA.
4) If only one block in cylinder, then ALPHA = 0.
5) Addressing is relative to disc address of last record done and uses ESC.
6) SEEK = 6 SPT for our discs.

A pictorial representation of software cylinder formatting is depicted in Fig. C-3. Elapsed time versus DELTA is plotted in Fig. C-4. Figure C-5 is a table of elapsed time for skipped-write as a function of matrix and buffer sizes.

ANALYSIS

Latency for "return seek":

AT = RNCYL * DELTA + ERL + RSEEK

WAT = ERL

RLAT = WAT - AT (MOD SS)

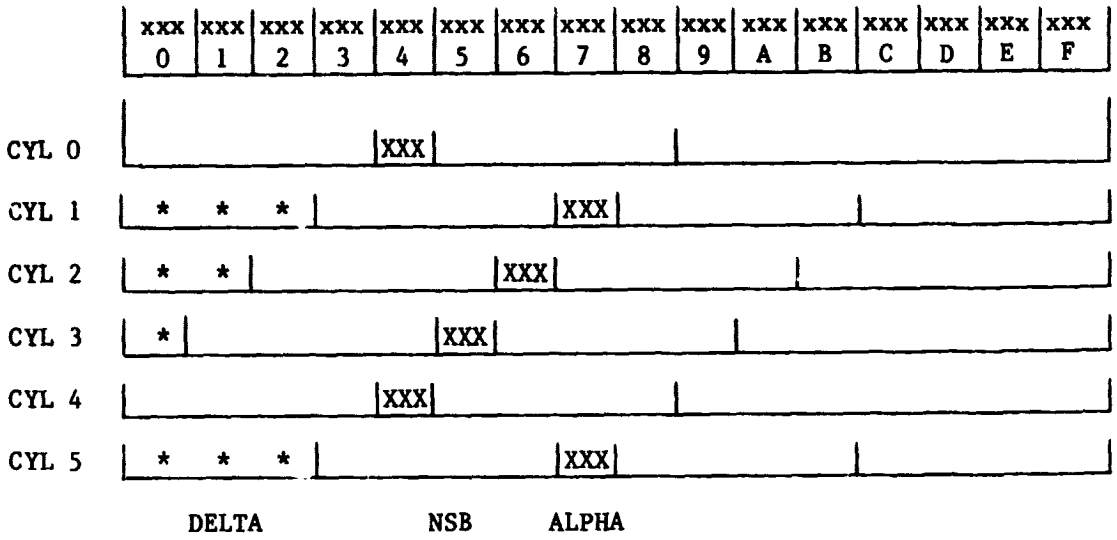
= -RNCYL * DELTA - RSEEK (MOD SS)

If there is some rough estimate of RSEEK, one can calculate the total elapsed time for "return" seek and latency exactly. With timings as a function of DELTA, one can isolate RSEEK.

Overall elapsed time for optimized ALPHA and DELTA:

TOTAL = T * [NW * ERL + RNCYL * SEEK + (RSEEK+RLAT)]

ORIGINAL PAGE IS
OF POOR QUALITY



Assumptions:

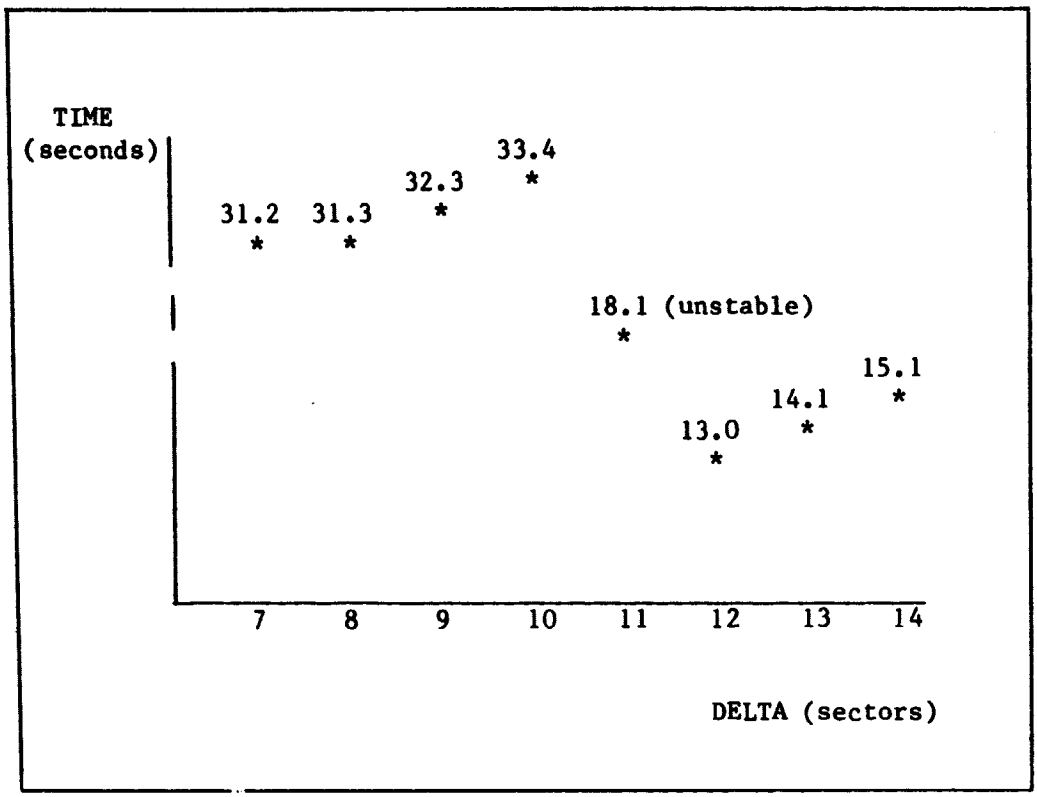
SC=16 and SS=4
ERL=1 and T=4
SEEK=1

Then by calculation:

NSB=4 and ALPHA=1
NBC=2 and DELTA=3
ESC=8

Figure C-3. Software Cylinder Formatting

ORIGINAL PAGE IS
OF POOR QUALITY



NOTE:

1) SEEK=6 for our discs as observed from similar timings
2) the time step for other points is 1.1 s (=T*RNCYL)
3) time for DELTA=7 is the same as time for DELTA=8
   (this also occurs for the DELTA=19, DELTA=20 pair)

Figure C-4. Elapsed Time as a Function of DELTA
for N=L=512 and B1=B2=4096

ORIGINAL PAGE IS
OF POOR QUALITY

MATRIX SIZE (in words, N=L)	BUFFER SIZE (in words, B1=B2) 32K	16K	8K	4K
256	+	+	.4	.8
512	+	2.0	4.4	13.0
1024	10.4	19.4	98.9	*
2048	110.0	*	*	*

Figure C-5. Elapsed Time (in seconds) in Phase 1 of Cornerturn (just skip-write)

NOTE:

1) if NSB > SC - SS
then ERROR `*`

This is when the block size is bigger than a cylinder. Our IO routine is based on an integral number of blocks per cylinder. The definition can be extended to cover the multiple cylinders per block case. The original nonblock cornerturn can handle all cases.

2) if LW >= 4096
then ERROR `+`

There is a slight increase in skip-write time for bigger buffers if record length is more than system maximum transfer count. This is because of the way a record is defined in our IO routine. A record is a contagious number of sectors followed by an extra unused sector. Changing this to software chained smaller portions with extra unused sectors in between would solve the slowdown problem for large record lengths.

3) Phase 2 cornerturn elapsed times are comparable and somewhat less. They are not given as they can be calculated rather easily as multiple sequential reads.

EXAMPLE

Calculate with N=L=512 and B1=B2=4096:

LW=64; NW=64; T=64; N1=8;

ERL=2; ESC=384; ALPHA=12; DELTA=12;

NSB=128; NBC=3.

RNCYL=21

RLAT = -21 * 12 - RSEEK (MOD 23)

= 1 - RSEEK (MOD 23)

now assume 2 =< RSEEK =< 23; then

RLAT = (1 - RSEEK) + 23

RELAPS = RLAT + RSEEK

= 24 (in SPT)

TOTAL = 64 * [64 * 2 + 21 * 6 + 24]

= 64 * [278]

= 13.1 s

The calculated 13.1 s agrees rather well with the observed 13.0 s in Fig. C-5.

Calculation of RSEEK from the elapsed time versus DELTA anomaly:

DELTA	7	8	19	20
RLAT	14-RSEEK	16-RSEEK	15-RSEEK	17-RSEEK
		+		+
EXTRA		21		21

To hide the +21, the expression above it has to be nonnegative.

Therefore, 16 =< RSEEK =< 17.

CONSIDERATIONS

Absolute addressing, where the disc address is recalculated from scratch based on record number, is expected to yield a cleaner implementation. Of some theoretical importance is the validation of an address calculation mechanism through some data structure and correctness proof type of formalism. This is particularly necessary as the addressing is nontrivial and read/write consistency needs to be maintained. A generalization to a mode where multiple records are accessed after a skip is possible. For the sake of disc utilization, the feasibility of blocks-across-cylinder-boundaries software cylinder formatting needs to be investigated.

The concept of record chaining is important for easy implementation of efficient IO. There is quite an advantage for ease of data management to preserve the independence of a record. If some generality and read/write access mode differences are to be preserved in an efficient manner, blocking is not an adequate solution. The software record chaining is an attempt to combine speed and flexibility; but it needs the extra wasted sector and involves software overhead. What is needed is intelligent controllers and the complementary disc sectoring so that hardware can chain records within the intersector gap. If the whole record chain is stored in the controller, hardware record chaining without next sector miss could be possible.

A complete cornerturn package does include the generalizations to cover the cases when the pixel size is a byte, half-word, word, or double-word.

An IO routine that incorporates access modes, record chaining, and software cylinder formatting is useful for cornerturn, real-time disc IO, and file merging; these functions frequently appear in signal processing and data-base operation.

REFERENCES

(C-5) B. Barkan, C. Wu, "Transpose of Externally Stored Matrices," Proc. 1982 ARRAY Conf., pp. 179-185, Floating Point Systems, Portland, Ore., March 1982.

(C-6) C. Wu, B. Barkan, W. Karplus, D. Caswell, "SEASAT Synthetic-Aperture Radar Data Reduction Using Parallel Programmable Array Processors," IEEE Trans. on Geoscience and Remote Sensing, Vol. GE-20, No. 3, pp. 352-358, July 1982.

(C-7) C. Wu, B. Barkan, J. Curlander, M. Jin, S. Pang, "Current Performance of the Interim Digital SAR Processor," in SIR-A Symposium Proceedings, JPL Publication 83-11, Jet Propulsion Laboratory, Pasadena, Calif. (in press).

ORIGINAL PAGE IS
OF POOR QUALITY

APPENDIX D

Cornerturn Example

A pictorial cornerturn example (Figure D-1) is given here.

Input:

11	12	13	14
21	22	23	24
31	32	33	34
41	.	.	.
51	.	.	.
61	.	.	.
71	.	.	.
81	82	83	84

Given:
N = 8
L = 4

Select:
B1 = 8
B2 = 16

Calculate:
C = 32
N1 = 2
T = 4
LW = 4
NW = 2
N2 = 2

(N2 = B2/N; note that N2 also determines the minimum overlap allowable on disc during the read phase, for multiple contiguous cornerturns in overlapped usage mode)

ORIGINAL PAGE IS
OF POOR QUALITY

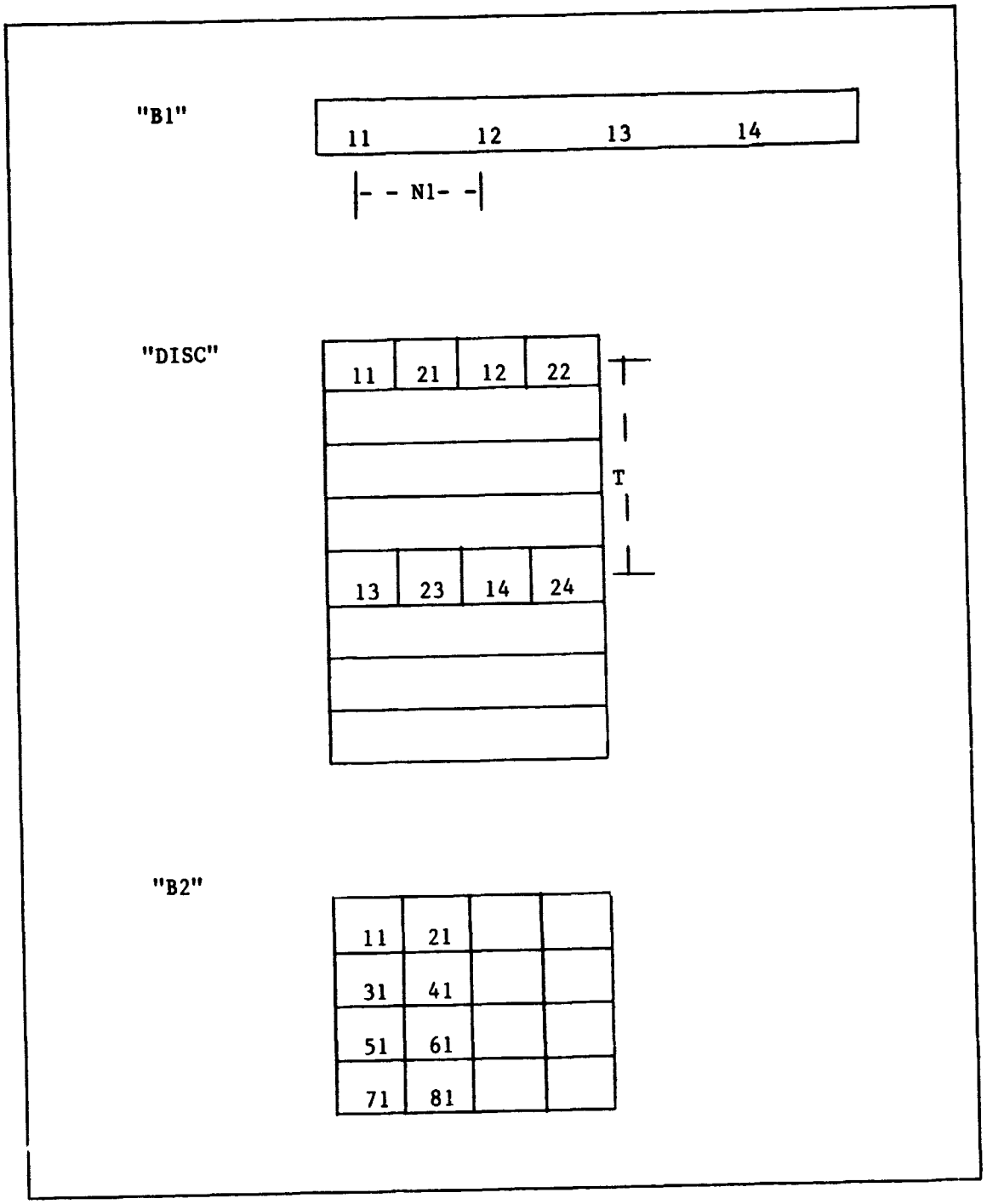


Figure D-1. Cornerturn Organization

NASA—JPL—Comi., L.A., Calif.