AD-A035 157 CONTROL DATA CORP MINNEAPOLIS MINN DIGITAL IMAGE SYS--ETC F/G 8/2
DIGITAL CARTOGRAPHIC STUDY AND BENCHMARK.(U)
SEP 76 D J PANTON, M E MURPHY DAAG53-75-C-0195

UNCLASSIFIED ETL-0092 NL

1 OF 1
ADA035157

END

DATE
FILMED
3-77

(18) ETL-0092 (19)

(2)

(6) DIGITAL CARTOGRAPHIC STUDY AND BENCHMARK, 3RD INTERIM TECHNICAL REPORT

(9) Interim technical rept. no. 3,

Prepared for:
U. S. Army Engineer Topographic Laboratories
Fort Belvoir, Virginia

(15) Contract DAAG53-75-C-0195

(10) Prepared by:
D. J. Panton
M. E. Murphy

(12) 45p.

(11) September 1976

Control Data Corporation
DIGITAL IMAGE SYSTEMS DIVISION
2800 East Old Shakopee Road
Minneapolis, Minnesota 55440

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ETL-0092	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DIGITAL CARTOGRAPHIC STUDY AND BENCHMARK 3RD INTERIM TECHNICAL REPORT		5. TYPE OF REPORT & PERIOD COVERED Contract 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) D. J. Panton M. E. Murphy		8. CONTRACT OR GRANT NUMBER(s) DAAG53-75-C-0195
9. PERFORMING ORGANIZATION NAME AND ADDRESS Control Data Corporation 2800 East Old Shakopee Road Minneapolis, Minnesota 55440		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Engineer Topographic Laboratories Fort Belvoir, Virginia		12. REPORT DATE September 1976 13. NUMBER OF PAGES 38
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved For Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Benchmark algorithm
Correlation patch
Parallel Arrays

(in AD's A035 155 and A035 156)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

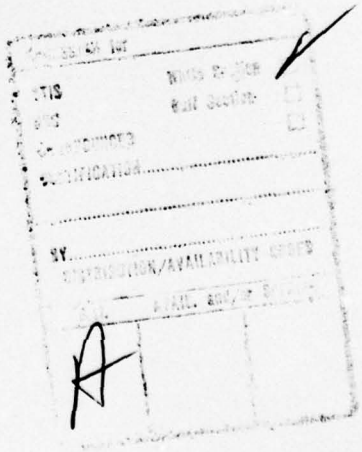
This report describes the work and findings of Phase C, ~~which~~ is to implement and test on fast, microprogrammable processors, the stereo matching algorithm that was designed and analyzed ~~under Phases A and B~~. This implementation was performed in terms of a benchmark to shed some light on the practicality of such a concept and to uncover the advantages and disadvantages of this particular kind of parallel processing application. The results stem from actually performing the benchmark rather than from a paper study outlining the approach to be taken. The benchmark was evaluated both in terms of its speed

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED (cont on p 1473-2)

(cont'd from #1473A)
and accuracy. It was found that it takes about 11 hours of CDC 6600 processing to complete the correlation and matching of a typical 9 x 9 inch frame stereo overlap area, while the benchmark implementation can process the same area in a little over one-half hour.

1473B



1473B

TABLE OF CONTENTS

		Page
1.0	INTRODUCTION	1
1.1	SUMMARY OF RESULTS	1
2.0	BENCHMARK SYSTEM COMPONENTS	3
3.0	BENCHMARK HARDWARE DESCRIPTION	5
4.0	ALGORITHM IMPLEMENTATION	9
4.1	LOGICAL MODULE DISTRIBUTION	9
4.2	COMMUNICATION AMONG PROCESSORS	14
4.3	OPERATING SYSTEM	18
5.0	OPERATIONAL FLOW	21
5.1	USER-HOST COMPUTER ENVIRONMENT	21
5.2	PARALLEL ARRAY ENVIRONMENT	23
6.0	BENCHMARK TIMING	27
6.1	INPUT-OUTPUT CONSIDERATIONS	31
7.0	LEVELS OF PARALLELISM	33
7.1	IMPACT ON ALGORITHM DEVELOPMENT	35
7.2	CONCLUSION	38

LIST OF FIGURES

FIGURE NO.	DESCRIPTION	PAGE
3-1	Internal Organization of the CDC Flexible Processor	7
3-2	Total Benchmark Configuration	8
4-1	Block Matching Conceptualization	10
4-2	Logical Module Distribution	12
4-3	Host 1700 Command Mode	15
4-4	Host 1700 Interrupt Mode	16
6-1	Timing Variable Definition	28
7-1	Sequential Implementation	35
7-2	Benchmark Parallel Implementation	36
7-3	Hypothetical Parallel Implementation	37

LIST OF TABLES

TABLE NO.	DESCRIPTION	PAGE
3-1	CDC Flexible Processor Characteristics	6
6-1	Timing Summary for a Representative Maximum Case	32

1.0 INTRODUCTION

This report describes the work and findings of Phase C of the Digital Cartographic Study and Benchmark. The primary purpose of Phace C is to implement and test on fast, microprogrammable processors the stereo matching algorithm that was designed and analyzed under Phases A and B of the program.

This implementation was performed in terms of a benchmark to shed some light on the practicality of such a concept and to uncover the advantages and disadvantages of this particular kind of parallel processing application. The results stem from actually performing the benchmark rather than from a paper study outlining the approach to be taken.

The complexity of the benchmark algorithm is representative of today's advanced digital image processing applications. That is, the algorithm is complex to the extent that production implementation on a general purpose computer system is not practical. The benchmark hardware is of a state of the art parallel processing nature with fast instruction cycle times and complete microprogramming capability. The numerical results are as accurate as the simulation results of Phase A and the benchmark speeds are well within the range predicted in Phase B.

1.1 SUMMARY OF RESULTS

The benchmark was evaluated both in terms of its speed and accuracy. The matching algorithm when implemented on a CDC 6600 computer, a large general purpose system, generates 14 match points per second for an average set of matching parameters. For the same test case, the benchmark algorithm generates 270 match points per second. Thus, the benchmark implementation runs approximately 19 times faster. Compared with the slower general purpose system, the CDC 6400, which produces about 8 match points per second, the benchmark implementation is approximately 34 times faster. Stated another way, it takes about 11 hours of CDC 6600 processing to complete the correlation and matching of a typical 9 x 9 inch frame stereo overlap area. The benchmark implementation can process the same area in a little over one half hour.

In terms of accuracy, the algorithm simulator that runs on the CDC 6600 and CDC 6400 performs arithmetic operations using the 60 bit word available on these machines. Floating point operands consist of 12 bits of exponent and 48 bits of mantissa. Computation accuracy with this word length is unquestioned; any inaccuracy here is attributed to algorithm design rather than the computing system used. However, the benchmark configuration is composed of 16 bit machines. The benchmark algorithm uses two words, or 32 bits, for most operands. But despite the smaller word length, the benchmark results were comparable in accuracy to the simulator results.

2.0 BENCHMARK SYSTEM COMPONENTS

The purpose of this section is to list the components of the benchmark system as a framework for the discussions that follow.

Hardware Components:

1) A CDC 1700 computer with the following peripherals:

   - tape controller and 3 magnetic tape units
   - disk controller and operating system disk drive
   - character display and keyboard for user interaction

2) 4 CDC Flexible Processors with associated data channels

Microcode Components:

1) Algorithm microprograms for each Flexible Processor

2) A LYNK package in one Flexible Processor

3) Special interrupt service microprograms in all FP's to handle specialized communication

Software Components:

1) COR operating system to run on the 1700 computer

2) A microcode assembler

3) A file system for creating and editing microprograms and command files

4) One overlay in the operating system to handle matching algorithm control functions.

5) A set of hardware diagnostic programs to check the integrity of Flexible Processor components

6) Utility programs to dump FP register files and to perform various debugging operations

3.0 BENCHMARK HARDWARE DESCRIPTION

The benchmark hardware configuration contains two basic computational units: a host computer that performs control functions and communicates with the external environment and an array of parallel processors that execute the image processing algorithm. The host computer is the CDC 1700, a small scale minicomputer that is typically used for production control applications . The 1700 is a 16 bit one's complement machine with an instruction cycle time of 1.1 microsecond.

The majority of computing for the benchmark is performed by the array of microprogrammable processors, each of which is a CDC Flexible Processor (FP). The FP has been designed as a modular hardware building block that can be configured in arrays to perform a wide variety of image processing applications. Each Flexible Processor is a microprogrammable computing unit that features high arithmetic computation rates and high data throughput rates. Additional FP characteristics are listed in Table 3-1.

Internally, the Flexible Processor is organized in a dual bus architecture. Figure 3-1 illustrates this organization. The register files shown in the figure are 60 nanosecond semiconductor memories that have simultaneous read/write capability. They are shown as having a word size of 32 bits, but in actual benchmark implementation they are considered as being double files of 16 bit words. For example, in the benchmark configuration each FP has 2 large files each containing 2048 16 bit words.

There are three basic types of communication channels that can be used to connect Flexible Processors together. The A/Q channel is compatible with CDC 1700 equipment and is used primarily by FP's for single word transfers of control information. The DSA channel is a data channel that is faster than the A/Q channel and is used for accessing data from and storing data into an external MOS semiconductor memory. The third type of channel is the high speed channel used for interprocessor data communication. The transfer rate on this channel is 8 megawords per second, making it ideal for transferring image data from FP to FP.

Table 3-1. ,CDC Flexible Processor Characteristics

- MICROPROGRAMMABLE-RANDOM ACCESS MICROCONTROL MEMORY
- 32-BIT OR 16-BIT WORD LENGTHS
- ARRAY HARDWARE MULTIPLIER
- 16-LEVEL HARDWARE PRIORITY INTERRUPT MECHANISM - 3 LEVEL MASK CAPABILITY
- SPECIALIZED LOGIC FOR SQUARE ROOT AND DIVIDE
- 8 MHz FILE BUFFERED WORD TRANSFER RATE-16 WORD BY 32-BIT OR 16-BIT INPUT FILE BUFFER
- 2 MHz DIRECT MEMORY ACCESS WORD TRANSFER RATE
- 1 MHz REGISTER-BUFFERED WORD TRANSFER RATES
- DUAL 16-BIT INTERNAL DATA BUS SYSTEM
- 0.125 μs CLOCK CYCLE
- 0.125 μs 32-BIT ADDITION: 0.250 μs BYTE MULTIPLICATION
- REGISTER FILE CAPACITY UP TO 4128 SIXTEEN-BIT WORDS
- HARDWARE NETWORK FOR CONDITIONAL MICROINSTRUCTION EXECUTION - 4 MASK REGISTERS AND A CONDITION HOLD REGISTER

The entire system on which the benchmark was implemented is shown in Figure 3-2. This system has been assembled in the laboratory of the Digital Image Systems Division and has been evolving for several years. It is experimental in nature and has been configured to facilitate interactive image analysis and research. As can be seen in the Figure, FP #4 is used to drive a digital scan converter which controls two color CRT displays, a color image projector, and an image hardcopy device. The scan converter is one of the resources of the total system that is currently not being used for the benchmark because the complete capability of FP #4 is needed for algorithm computations.

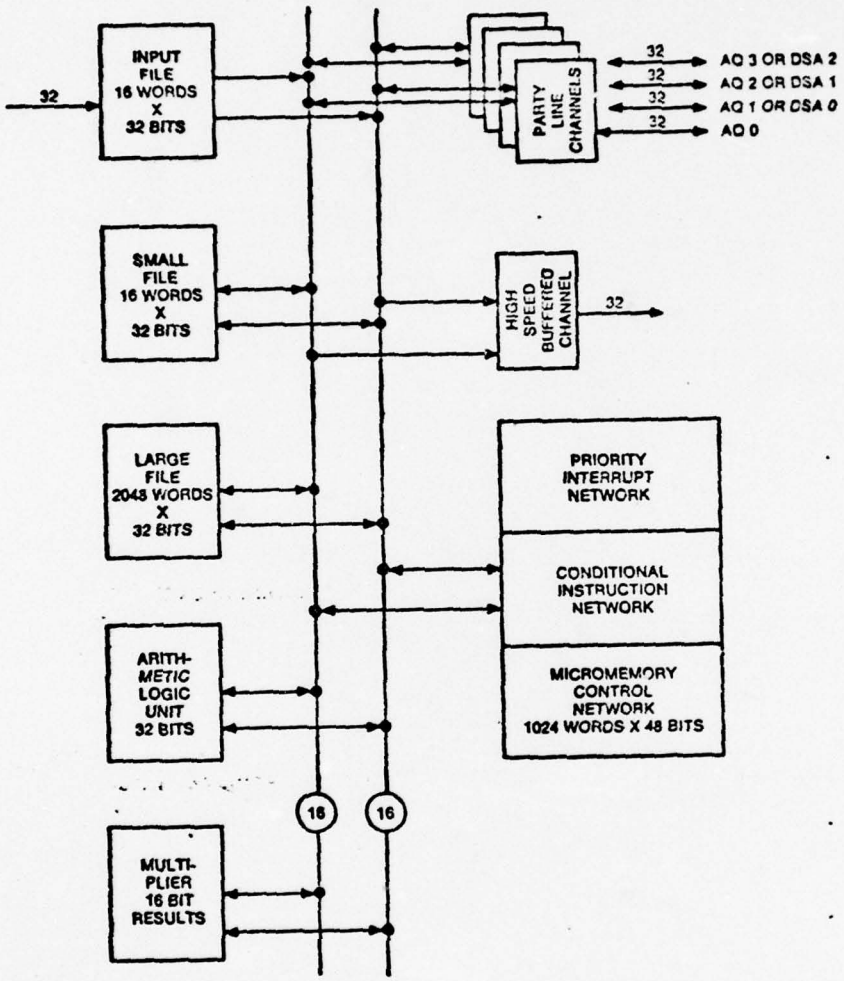


Figure 3-1. Internal Organization of the CDC Flexible Processor

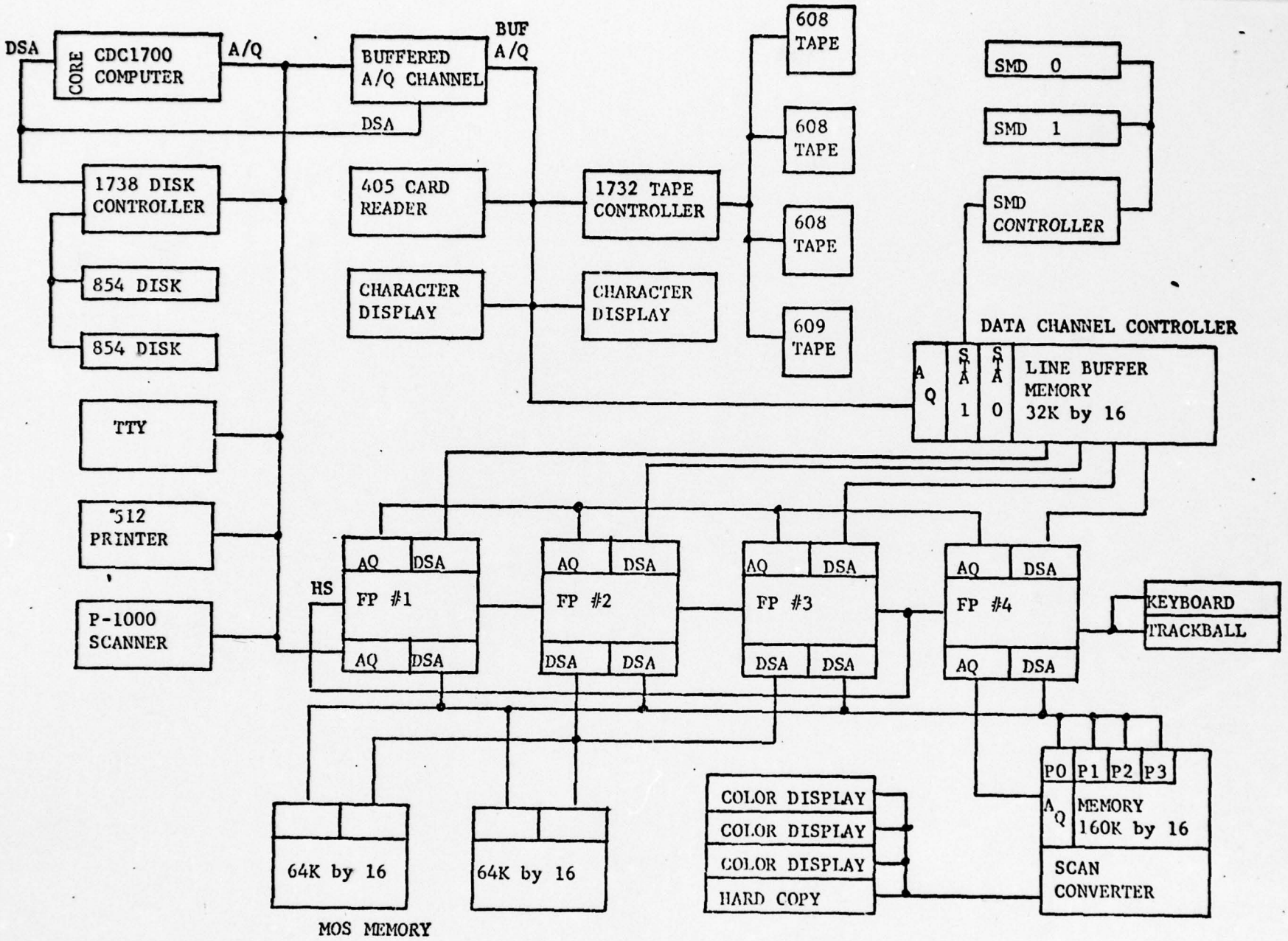


FIGURE 3-2. Total Benchmark Configuration

4.0 ALGORITHM IMPLEMENTATION

The stereo matching algorithm that was designed and benchmarked is well described in the Phase A, and Phase B, Interim Reports. The primary function of the algorithm is to correlate and match a stereo pair of digital images along a grid with arbitrary intervals for the purpose of generating digital terrain data. The images of the pair are arbitrarily denoted as the A and B images. The basic concept behind the algorithm is illustrated in Figure 4-1. A brief summary of the steps needed to generate one match point are as follows:

- Make a prediction as to the location of the next match-point on the B image using epipolar geometry and recent matching history.
- Determine the shaping parameters for the B image correlation area and shape the image data.
- Correlate the A image patch with the shaped B image area at sites on each side of the predicted location along the epipolar line.
- Determine the correlation maximum and various reliability criteria to compute the precise match-point.
- Update the prediction mechanism with the new found match-point and associated matching data.

4.1 LOGICAL MODULE DISTRIBUTION

The logical modules of the stereo matching algorithm have been named and described in the Phase B Interim Report. The problem of implementing the algorithm on the flexible processor configuration is a problem of distributing the logical modules over the available processors in a highly optimized and highly parallel fashion. The objective is to minimize the idle time of the most heavily loaded processor.

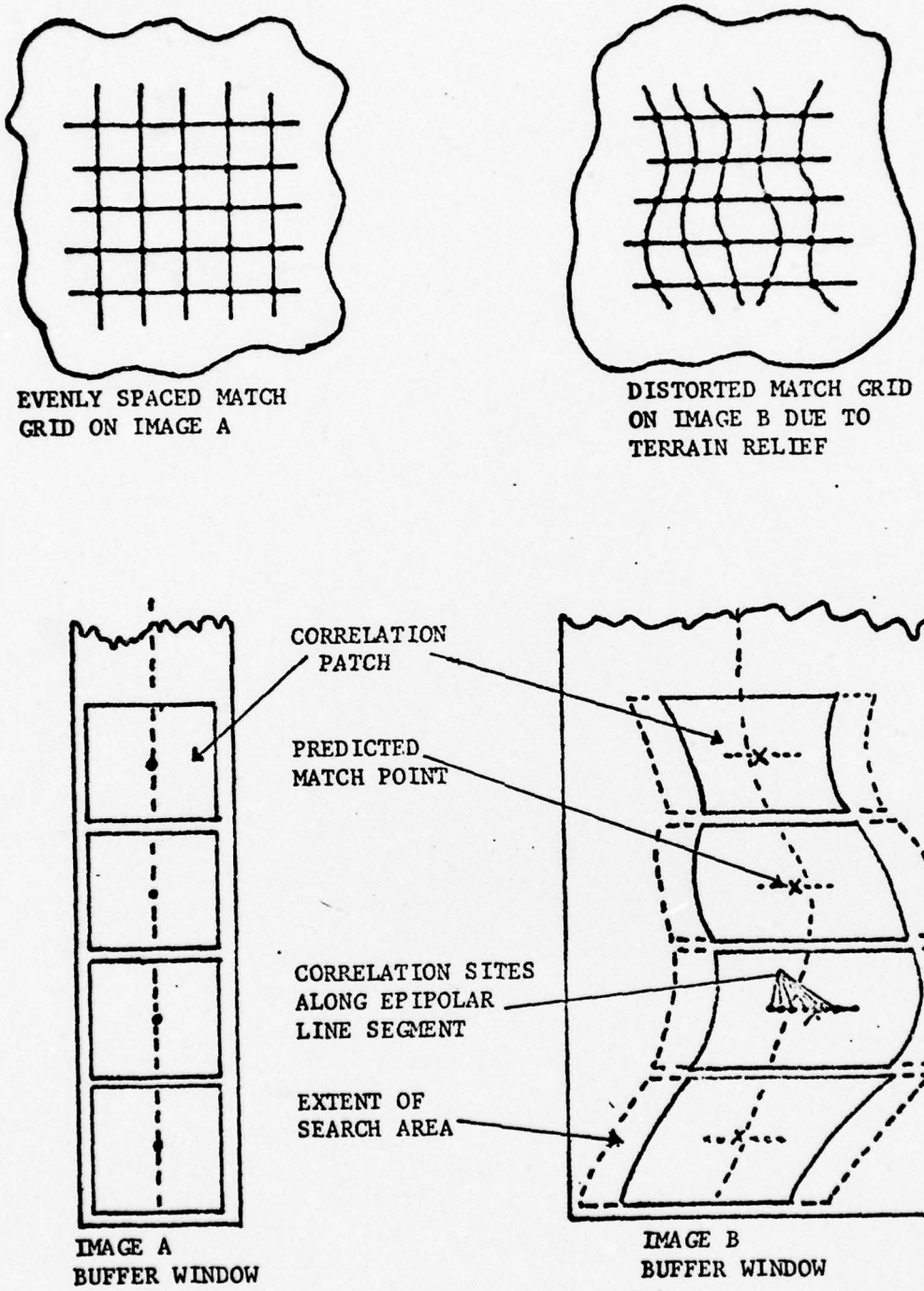


Figure 4-1. Block Matching Conceptualization

The final distribution of logical modules that was decided upon is diagramed in Figure 4-2. The primary modules of the algorithm are implemented in terms of four microprograms, ART1, IPC, CPMA, and RELY. CPMA is the largest microprogram and thus resides in FP#3, which contains 2K of micromemory. FP#1, FP#2, and FP#4, each contain 1K of micromemory and ART1, IPC, and RELY, are correspondingly smaller microprograms than CPMA.

In general, CPMA is column-oriented, ART1 and IPC are row-oriented, and RELY is match-point oriented. In the algorithm, a column is defined as a set of correlation patches or subregions whose centers all have the same X coordinate on the A image. A column can also be considered a profile. A row refers to a line of pixels extending in the X direction. For example, if correlation is to occur over a 21 by 21 pixel patch, then a row contains 21 pixels, the width of the patch, and the patch contains 21 rows along the column.

To clarify this arrangement, a general description of the function of each microprogram follows.

CPMA - This is the control processor and memory accessing processor of the array. At the start of a column of points to be matched, CPMA sets up the shaping parameters to access B image data for the entire column. Then one row at a time for every patch of the column, image data is brought from MOS memory, shaped, and sent over the high speed data line to ART1 and IPC. When match-points have been generated in RELY, they are used by CPMA to update all the control parameters for predicting succeeding columns across the image.

ART1 - This is an arithmetic processor. Its primary functions are to receive corresponding rows of A image and B image data from CPMA; send the same data to IPC; and accumulate A image gray scale sums and sums of squares and B image sums and sums of squares for each correlation site along the search segment. When all the rows for a complete patch have been accumulated, the sums are sent to RELY. One execution of the main loop of ART1 accumulates one row of pixels.

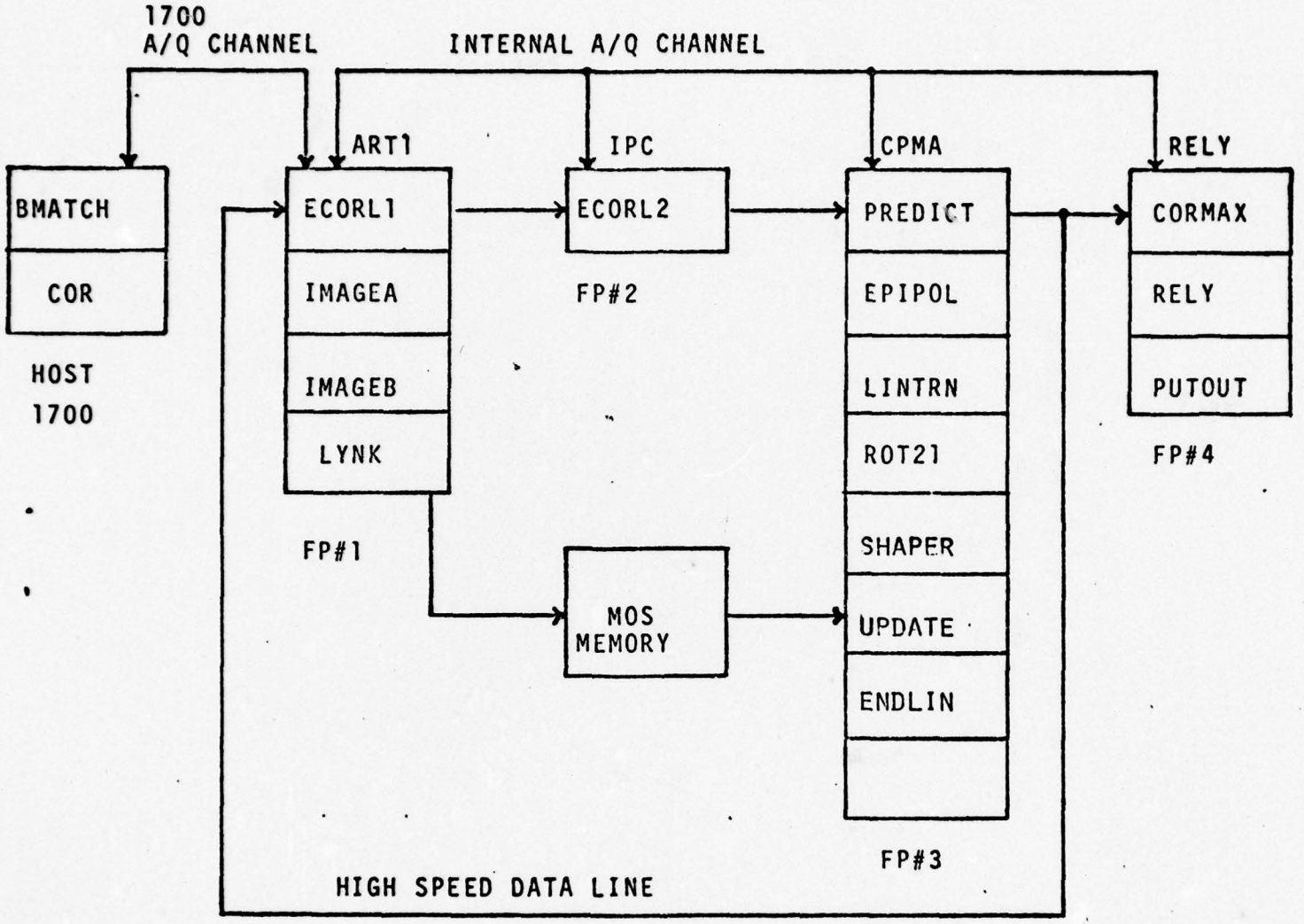


Figure 4-2 Logical Module Distribution

When not performing this correlation arithmetic, ART1 has two additional functions on an interrupt basis. The first is the transferring of image data from the 1700 to the MOS memory. The removal of this function of ART1 will be discussed later. The second is the LYNK function. LYNK is an interrupt driven microcode module that serves as the communication path between the 1700 computer and all the Flexible Processor. To the 1700 hardware, the flexible processor configuration appears as a single peripheral unit. However, the 1700 software can communicate with any FP individually. This is accomplished through LYNK. That is, the 1700 talks to LYNK over the 1700 A/Q channel in terms of command codes. This information is then routed by LYNK to the appropriate Flexible Processor over the internal A/Q channel. Likewise, when an FP wishes to interrupt the 1700, it must do so through LYNK.

IPC - The sole function of the IPC processor is to perform the inner product computations necessary for the development of correlation coefficients. A row of A image pixels and a row of shaped B image pixels is received from ART1 and cross products for the row are accumulated for each correlation site of the desired search area. When all the rows for a complete patch have been accumulated, the sums of the cross products are sent to RELY. One execution of the main loop of IPC accumulates one row of pixels.

RELY - This processor receives all of the statistical sums necessary for computing correlation coefficients from ART1 and IPC and computes a correlation coefficient for each site of the search segment. It locates the maximum correlation value along the search segment and fits a second order curve over the discrete correlation surface to determine the correlation maximum to a fraction of a pixel. A match-point is generated and the reliability factor for that point computed. RELY then sends the match-point to CPMA for prediction feedback and also places it in one of two output buffers. When one buffer is full, RELY interrupts the 1700

and goes on to fill the second buffer. Meanwhile, the 1700 empties the first buffer onto tape by interrupting RELY when necessary.

HOST 1700 - The Control Data 1700 host computer is the primary link between the operator, the peripheral I/O units and the array of parallel processors. The COR operating system running on the 1700 performs the benchmark functions in two modes: an interactive operator command mode and an interrupt-driven service mode. These two modes are outlined in Figures 4-3 and 4-4. In the command mode, the system, upon command, loads all flexible processors with their corresponding microprograms, clears and starts all FPs running, receives algorithm initialization parameters coming in as part of commands, computes additional parameters from those given, initializes all FPs with their appropriate starting parameters, and starts the algorithm running. This initialization procedure is performed once per matching run. In the interrupt service mode, the 1700 fetches more image data, sends it to the MOS bulk memory through ART1, and also empties the match-point buffers in RELY.

4.2 COMMUNICATION AMONG PROCESSORS

The backbone of the algorithm implementation on the Flexible Processor configuration is the nature of the communication between processors. Generally, the smaller the amount of communication overhead, the more efficient is the implementation. For this reason, efforts have been made to limit communication in the benchmark implementation to single-step flag settings or to transfers of just a few words of data. The exception to this, of course, is the transfer of image data from processor to processor.

Following is a breakdown for all the possible communication paths in the implementation. For each path there is an active processor ("from" column) and a passive processor ("to" column). The communication description states the function of the active processor. Generally, when communication is initiated from an active processor to a passive processor, the passive processor does respond, but this response is typically a hardware logic function internal to the communication channel. Therefore, this response is not included in the

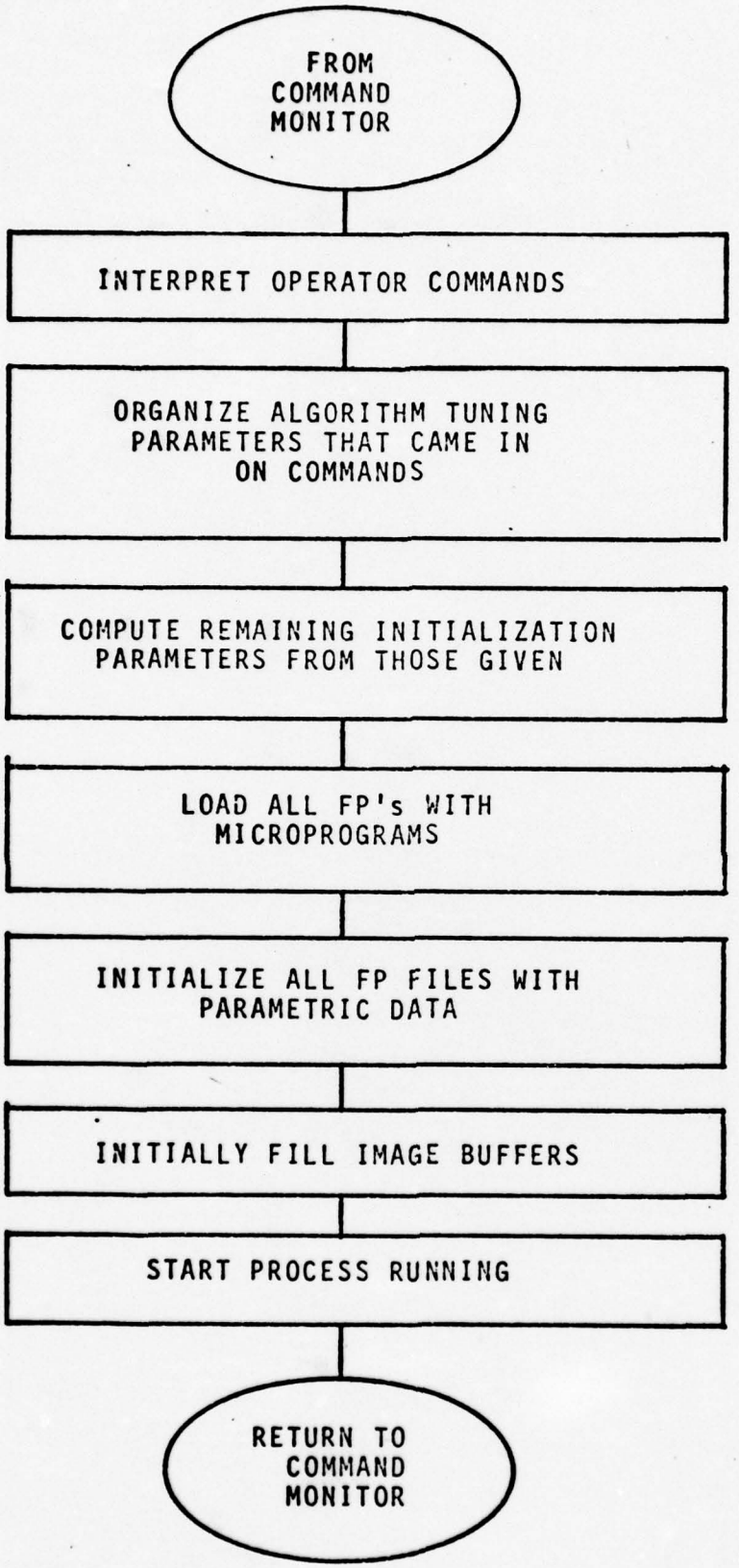


FIGURE 4-3. Host 1700 Command Mode

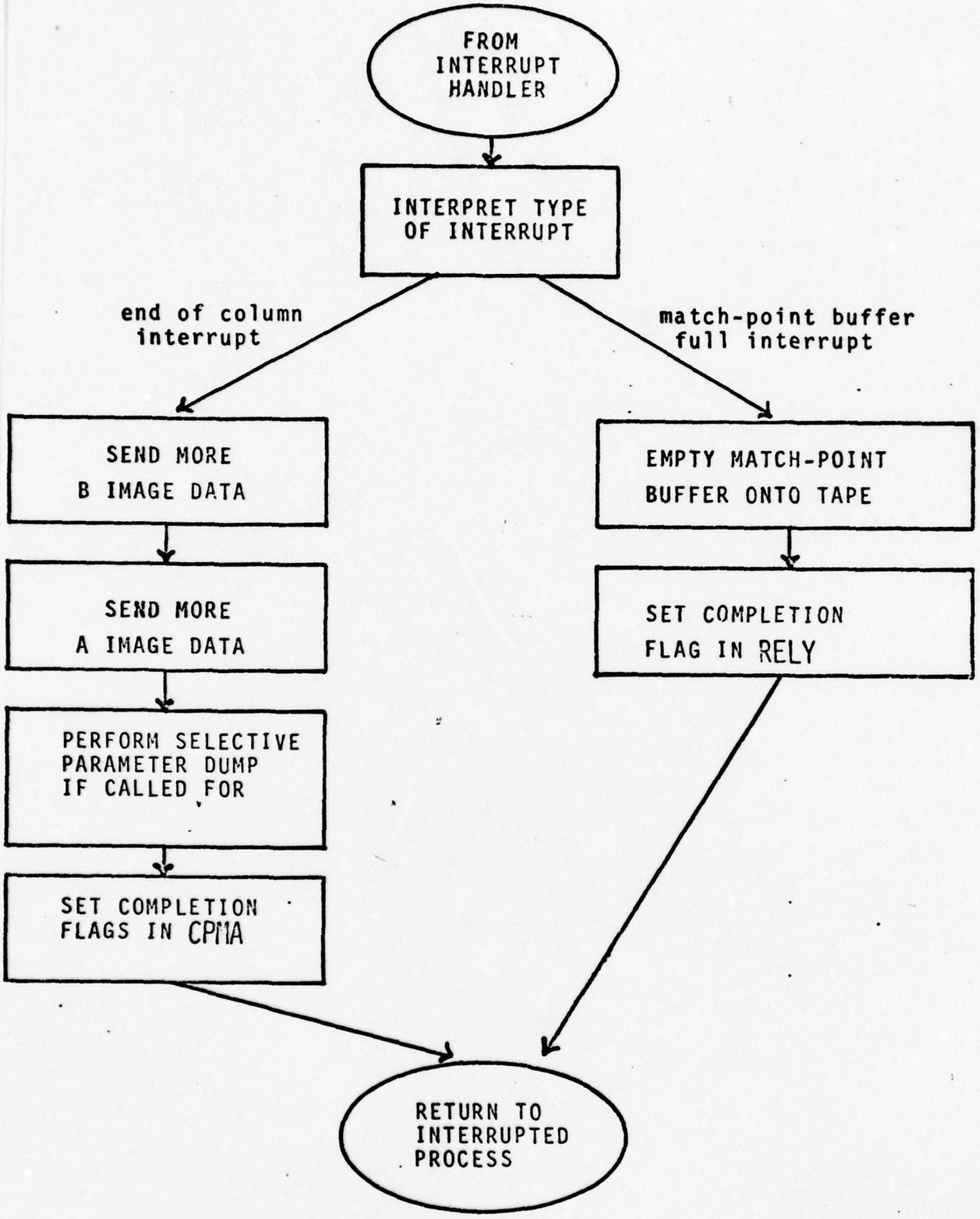
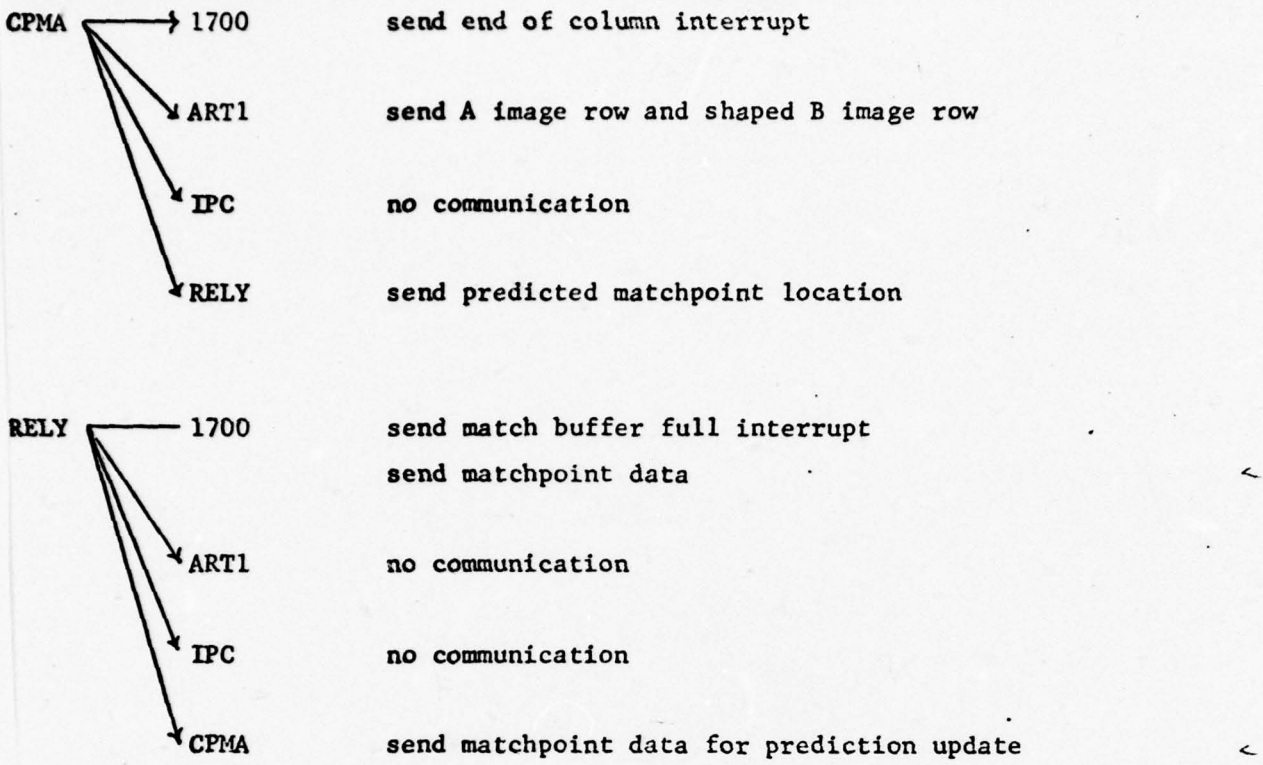


Figure 4-4. Host 1700 Interrupt Mode

breakdown because no overhead cost is incurred.

From	To	Description
1700	ART1	send data to fill A image buffer
		send data to fill B image buffer
	IPC	no communication
	CPMA	set stop/go flag
		set flag for A image transfer complete
		set flag for B image transfer complete
	RELY	set flag for buffer empty
ART1	1700	no communication
	IPC	send shaped rows of image data
	CPMA	signal row completion
	RELY	send gray-scale sums and sums of squares
IPC	1700	no communication
	ART1	no communication
	CPMA	signal row completion
	RELY	send gray-scale cross-products



.3 OPERATING SYSTEM

The operating system chosen to run on the 1700 host computer to support he benchmark algorithm is called the COR Operating System. This operating ystem is designed primarily as a vehicle for communicating with and controlling lexible Processor configurations. Some of the basic capabilities of the perating system are as follows:

- A command monitor for interpreting and executing operator commands that are entered via a CRT character device or a teletype.
- A file system for creating and editing microprograms and command files.
- A set of hardware diagnostic programs to check the integrity of Flexible Processor components.

- A set of utility programs to alter FP micromemory, to dump FP storage files, and to perform various other debugging operations.

A microcode assembler is also available, but it resides in the standard CDC 1700 MSOS Operating System.

The COR Operating System is organized in terms of a resident portion and a series of overlays. The command monitor, interrupt handlers, and FP utility programs make up the resident portion. The file editor and other file system utilities as well as the hardware diagnostic programs reside in overlays. User programs such as the benchmark control program, are also implemented as overlays.

Besides the standard set of commands that is interpretable by the command monitor, the capability exists for defining user commands that are applicable to the user's overlay. As an additional capability, commands may be grouped together in a command file; and by issuing the file name as a single command, the operating system is directed to interpret and execute each individual command in the file sequentially.

This command file capability is the vehicle for implementing various tuning parameter options for the benchmark stereo matching algorithm. A series of parametric commands have been defined for operator entry of algorithm tuning parameters, and a series of action commands defined to control the operation of the algorithm. Thus, for a given set of sensor images to be matched, the tuning parameters are determined and entered, on parametric commands, into a command file. Then, for all subsequent runs of that imagery, only the command file name is necessary to initiate the runs. This capability eliminates the need to enter parameters for each similar run.

Following is a list of the commands that have been implemented for the tuning and control of the benchmark algorithm. The basic command format is the command name followed by a sequence of integers or decimal fractions that is variable, depending on the command. All the benchmark and block matching

commands have been prefixed with BM to distinguish them from the standard COR command repertoire.

Parametric Commands:

BMBASE - relative orientation baseline vector
BMFOC - focal length of cameras
BMORM - relative orientation matrix
BMFXA - A image interior orientation transformations
BMFXB - B image interior orientation transformations
BMABUF - A image buffer size and characteristics
BMBBUF - B image buffer size and characteristics
BMGRID - matching grid specification
BMPCH - correlation patch size and search specification
BMPWGT - prediction weight functions
BMRELT - reliability cutoff values
BMKPN - number of initial matchpoints for startup
BMMTCH - initial matchpoint to start process

Action Commands:

BMLDA - load all FP's with microprograms
BMSTUP - set up initialization parameters for FP's
BMINIT - initialize all FP's with startup parameters
BMGO - start algorithm running
BMHALT - stop algorithm
BMRESM - restart algorithm after BMHALT
BMDFLT - set up parameters for A vs A autocorrelation
BMDMP - dump selected matching parameters

5.0 OPERATIONAL FLOW

To further explain the internal operation of the benchmark algorithm and system, the following sections outline the basic sequence of processing events. These events occur in two separate environments; the user-host computer environment and the parallel array environment.

5.1 USER-HOST COMPUTER ENVIRONMENT

The interactive involvement of the user and the host 1700 in the benchmark algorithm is outlined in the following steps. All of these steps occur in a matter of seconds, or as fast as the user can respond at a keyboard.

1) The user loads the resident monitor of the COR operating system from disk. The system maintains in files on the disk all microcode binaries, all command service programs, and the major overlay to run the benchmark matching algorithm.

2) Via a CRT character display, the user loads the four Flexible Processors with their microprograms and starts them running in their idle loops. This is performed by the command BMLDA.

3) The user then proceeds to execute the command file that contains the algorithm tuning parameters in the form of parametric commands. The user can also enter single parametric commands to alter the parameters entered from the command file. For example,

   BMGRID 32 144 8 16 166 10

   defines the matching grid over which to correlate; that is, find match points from image scan line 32 to line 144 in steps of 8 lines and from pixel 16 to pixel 166 in steps of 10 pixels. As these parametric commands are executed, the parameters are reformatted and placed in a storage block in 1700 memory.

4) When all parametric commands have been processed, the user issues

the BMGO command. This command performs the following functions:

- The basic tuning parameters entered above are used to compute the matching initialization values which are reformatted to be compatible with the FP's and are organized into blocks in the 1700 memory. These blocks are images of the FP large file registers. The reformatting is necessary to achieve various double precision number scalings in the FP's and because of the fact that the 1700 is a one's complement machine while the FP is a two's complement machine.

- The blocks of initialization values are transferred to the FP large files.

- The image buffers in the MOS bulk memory are initially filled.

- The 1700 interrupt system is enabled.

- The GO flag is sent to CPMA to start the block matching algorithm running.

- Control returns to the COR command monitor and the 1700 waits for interrupts.

5) The algorithm terminates when all match points over the selected grid have been generated.

6) While the algorithm is running the operator can suspend execution by issuing the BMHALT command and can continue by issuing the BMRESM command.

7) The algorithm can also be placed in a step mode by the BMDMP command such that the algorithm stops after every column of match points and a decimal dump of user selected matching values appears

on the CRT. This allows the user to monitor the internal environment of the parallel matching process.

5.2 PARALLEL ARRAY ENVIRONMENT

In this environment CPMA functions as the executive processor of the parallel array. CPMA maintains all of the prediction parameters and all of the recent correlation history in its large files. The other processors are more or less controlled by CPMA in performing tasks to generate data that is used by CPMA to keep the process running. Following is the sequence of events that occur in the four Flexible Processors. The process names, like CPMA, will be used to describe the action of the processors. It must be kept in mind that most of these events occur simultaneously in different FP's.

1) At the start of a column of match points, CPMA first determines whether the data is available in the image buffers for that column. If it is available, CPMA then sends an interrupt to the host 1700 to start filling the buffers for the next column.

2) For each correlation patch of the column, CPMA fetches the prediction data for that patch from its files and sends it to RELY. The same data is then used to compute all the shaping parameters for B image data. CPMA then accesses from MOS bulk memory one patch row of A image data and the corresponding shaped row of B image data. These rows are sent over the high speed data line to ART1. At this point CPMA goes on to access and shape the next row of data.

3) When ART1 receives the A row and shaped B row of image data, it immediately passes the same data to IPC over the high speed data line. The ART1 proceeds to accumulate the gray-scale values of the row of pixels into registers that correspond to an A patch sum, an A patch sum of squares, B patch partial sums, and B patch

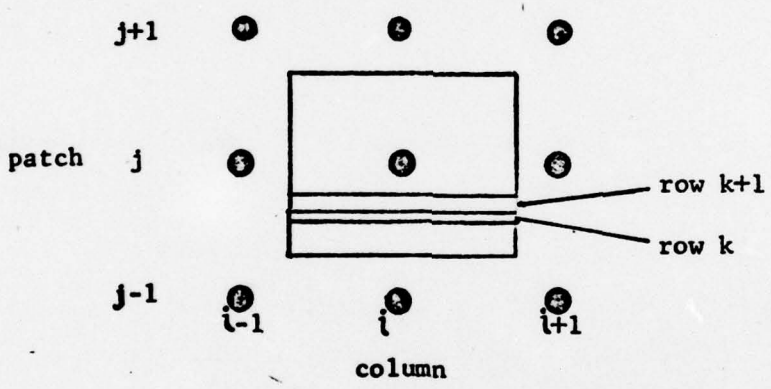
partial sums of squares. When a row is complete, ART1 signals its readiness to CPMA. When all the rows of a patch are complete, ART1 sends the resulting sums to RELY over the internal A/Q channel. While all of the above is occurring, a background process in ART1 is using the LYNK module in an interrupt mode to relay the image data for the next column from the host 1700 to MOS bulk memory.

4) When IPC receives the A row and shaped B row of image data from ART1, it starts accumulating gray-scale cross products in its registers. More than one sum of cross products is required because correlation coefficients will be formed for a number of sites on each side of the predicted site along the epipolar line. Therefore, every A image pixel in the row must be multiplied by several corresponding B image pixels. The number of multiplies here is dependent on the number of correlation sites. When a row is complete, IPC signals its readiness to CPMA. When all the rows of a patch are complete, IPC sends the sums of cross products to RELY over the internal A/Q channel.

5) When RELY receives the statistical data from ART1 and IPC, it forms a correlation coefficient for each site of the search segment surrounding the predicted site. Then RELY finds the maximum value of these correlation coefficients and fits a parabolic curve over this maximum value and the values at the two adjacent sites. In this way the correlation maximum and therefore position of best match is determined to a fraction of a pixel. RELY next computes the reliability of the match based on the preset reliability cutoff values and the statistics from the match. A correction to the match point is made if necessary. Then a final match point 5-tuple (x, y, u, v, r) is formed and deposited in a match point buffer. The u portion is sent to CPMA over the internal A/Q channel for future predictions. When one match point buffer becomes full, RELY sends an interrupt to the host 1700 to empty the buffer and subsequent match points are placed in a second buffer.

The above events comprise the majority of processing activity of the parallel array. If the algorithm were stopped arbitrarily at row k of patch j of column i, the following would be observed:

- The host 1700 is transferring image data for column i+1 into MOS bulk memory.
- CPMA is accessing and shaping row k+1 of patch j.
- ART1 is accumulating sums for row k.
- IPC is accumulating cross products for row k.
- RELY is developing a match point for patch j-1.



Additional processing events are as follows:

6) After the first patch of a column, that is, when the match point for the last patch of the preceeding column has been completed by RELY, CPMA updates its correlation history and prediction mechanism based on the entire column of new found match points.

7) In addition, CPMA corrects any patches that may have wandered in the previous column. This is done by comparing a patches position with the average of its neighbors and correcting those patches whose position deviates by more than parameter "wandering block tolerance".

8) Between the patches of a column, CPMA uses relative orientation of the exposures, the interior orientation transformations, epipolar geometry and previous matching history to define the predicted v coordinate of the next patch.

6.0 BENCHMARK TIMING

The purpose of this section is to specify the timing of each microprogram and the overall throughput rates of the benchmark algorithm. In general, the algorithm timing and throughput rates are variable, depending on certain tuning parameters such as correlation patch size and match point interval size. Below is a list of the timing variables that appear in the timing formulas.

HXA - integral half patch size in the line direction (X)

HYA - integral half patch size in the pixel direction (Y)

SX - integral half of the number of correlation search sites along the epipolar line.

For example, for a correlation patch that is 21 lines by 11 pixels, HXA = 10 and HYA = 5. If correlation is to occur at seven sites, including a predicted site and three sites on each side, SX = 3.

NSPX - number of grid intervals in the line direction (X) that fall within a single patch. Also, the number of subpatches in X that have different shaping parameters.

NSPY - number of grid intervals in the pixel direction (Y) that fall within a single patch. Also, the number of subpatches in Y that have different shaping parameters. The total number of subpatches in a patch, then, is NSPX*NSPY.

KPN - number of patches or match points per column.

Figure 6-1 illustrates the dimensionality of these variables.

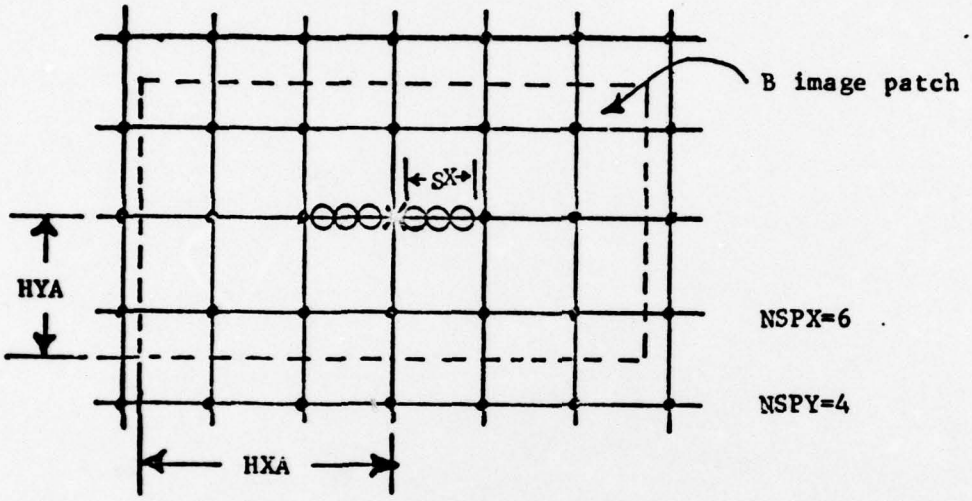


Figure 6-1. Timing Variable Definition

Following is the timing breakdown for each processor in the parallel rray. All the times are expressed in microseconds.

CPMA - Timing for one patch:

$$T_1 = 204.88 + 3.313*NSPX + 4.625*NSPY + 2*NSPX + NSPY$$

$$+ (HYA + 1) \quad \boxed{25.375 + 23.375*HXA + 19.75*SX + 3.5*NSPX}$$

- Additional time at the end of a column:

$T_2 = 18.25 + 5.438*NSPY + 15.75*KPN$

- Total time per patch including end of column:

$T_3 = T_1 + T_2/KPN$

ART1 - Timing for 1 row of patch:

$T_4 = 6.5 + 4*HXA + 2.375*SX$

- Additional time at the end of a patch:

$T_5 = 9.5 + 2.25*HXA + 11.5*SX$

IPC - Timing for 1 row of patch:

$T_6 = 2.5 + 1.625*HXA + 1.75*SX + 1.75*HXA*SX$

- Additional time at the end of a patch:

$T_7 = 2.5 + 4*SX$

RELY - Timing for 1 patch:

$T_8 = 110.125 + 106.5*SX$

.11 of these times were determined by accumulating the number of instructions n each microcode module and multiplying by the Flexible Processor instruction ycle time, .125 microseconds.

In order to determine the time per patch of the entire benchmark, the arallel interleaving of the times for individual processors must be considered. PMA is the most heavily loaded processor and thus runs the longest per atch. The critical time for this processor is the time required to access row of A image data and to access and shape a row of B image data. This

time for a single row pair is:

$$T_{1A} = \left[25.375 + 23.375*HXA + 19.75*SX + 3.5*NSPX\right] \quad /2$$

Now, the ART1 and IPX row times, T_4 and T_6 respectively, are interleaved with T_{1A}. Likewise, the ART1 and IPX end of patch times T_5 and T_7, are interleaved with the remainder of T_1. The total patch time for RELY, T_8, is also interleaved with T_1. Therefore, the total time per patch for the benchmark is determined solely by CPMA. This is time T_3.

When the benchmark was completed and analyzed, results showed that the majority of algorithm running time was spent accessing and shaping rows of image data. This was somewhat surprising. In Phase B of the program it was determined that the majority of time would be spent by the processor responsible for accumulating the statistical sums for correlation. Therefore, this task was distributed between ART1 and IPC to achieve a higher degree of parallelism and thus reduce the time spent in this task. This distribution caused the shaping portion of CPMA to emerge as the major time consumer.

In order to analyze the benchmark timing in more practical terms, an individual case of algorithm tuning will be considered. In processing the 1:48000 Phoenix stereo model, it was found that a 21 x 21 pixel patch over an eight line by ten pixel matching interval provided adequate algorithm tuning for collecting digital terrain data. This represents a rather conservative case, because the algorithm runs much faster with smaller patches. For this case the digital scan interval is 25 micrometers and the stereo overlap area is approximately nine inches by five inches. Therefore, a pair of images that are 9000 by 5000 pixels must be matched. At the specified grid interval, this involves the generation of approximately 562,500 match points. The pertinent tuning considerations for this case are summarized in Table 6-1.

6.1 INPUT-OUTPUT CONSIDERATIONS

In the preceding discussion, all of the timings refer to parallel processor times alone. The overhead time necessary to transfer image data to and from the parallel processing array and the system peripheral units was not considered. A number of options exist for this image data input; the benchmark system contains four CDC 608 tape drives, two 854 disk drives, and two storage module drives (SMD). The SMD is by far the superior I/O device having an unformatted storage capacity of 80 megabytes and the capability for transferring one 16-bit word or two 8-bit pixels in 2.7 microseconds. In the original Phase B plan for the benchmark the input of image data into the system was to be from a SMD to the Line Buffer Memory, which is shown in Figure 3-2. All the Flexible Processors can easily access this memory, and there is little host 1700 intervention in the data transfer process. However, it was learned that the SMD's, their controller, and interface would not be available in time for the benchmark. Therefore, the input implementation for the benchmark is from tape as an interim measure. At the time of this writing the SMD's are part of the system and functioning properly. Eventually, the tape input will be removed from the benchmark algorithm and the data accessing modules will be converted to receive input from the SMD's through the line buffer memory.

Currently, input image data enters the algorithm from tape at a rate of one 16-bit word or two 8-bit pixels every 533 microseconds. At this rate the data for the next column of correlation patches can be completely transferred in the time it takes the algorithm to process 1092 patches of the current column under the tuning conditions for the above sample case. In most of the test cases run to date, in which the number of patches per column was less than 1092, the algorithm ran faster than the tapes could move; the Flexible Processors spent most of their time in idle loops waiting for more data.

Under 854 disk implementation, image input occurs at the rate of one 16-bit word every 16 microseconds. At this rate, a column of image data can be transferred in the time required to process 33 patches. Using the

SMD, however, the transfer rate is one word per 2.7 microseconds, and a column can be transferred during the processing of just six correlation patches. The net result of this I/O analysis is that the benchmark algorithm is not I/O bound when using the SMD as an I/O source. Thus the parallel array spends no extra time waiting for data, and time T_3 above is an accurate prediction of the total benchmark throughput.

Table 6-1: Timing Summary for a Representative Maximum Case

Patch size	21x21 pixels
Matching grid interval	8x10 pixels
Correlation search size	5 sites
Image size	5000x9000 pixels
Match points per model	562,500
Benchmark time per match point	.0037 seconds *
Benchmark time per model	35 minutes
CDC 6400 time per model	20 hours
Speed increase of benchmark CDC 6400	34 times
CDC 6600 time per model	11 hours
Speed increase of benchmark over CDC 6600	19 times
Benchmark throughput rates	270 match points/second 119,000 pixels/second
Equivalent add operations per patch	29,600
Equivalent add operations per pixel	67

* Can this be decreased with more FPs?
what is upper limit of one channel?
Now we get 270 pts/sec. with 4 FPs and 21x21 with 5 sites

7.0 LEVELS OF PARALLELISM

Three levels of parallelism are generally considered when dealing with the implementation of algorithms on parallel processing hardware, particularly on the Flexible Processor. These are the instruction logic level, the processor level, and the systems level.

On the instruction logic level, parallelism arises from the fact that more than one operation can be performed in a single microinstruction. The Flexible Processor is organized such that there are two separate data buses that connect two input files, two temporary files, two large files, and two adders. Therefore, 16-bit operands may be routed among these components independently and in parallel over either bus. For example, a 16-bit number in a temporary file location may be sent to the adder over one bus; while, at the same time, two 8-bit pixels may be transferred from an input file location to a large file location on the other bus; while, at the same time, the hardware jump stack may be incremented or decremented. All of this occurs in one instuction cycle, the sequence of events being synchronous with the internal timing scheme of the instruction. Some of this parallelism is lost, however, when 32-bit operands are processed because both buses are required.

It is at this level that a microprogrammer can exploit all the devices of his art. To implement a fast algorithm, it is of the utmost importance to sequence the individual operations of the algorithm in such a way that the maximum parallelism is achieved at the instruction level. A truly optimized algorithm is one in which both buses are kept busy through most of the instruction cycles.

On the processor level of parallelism, how much parallelism can be achieved among the processor in a configuration is the main concern. In terms of the benchmark, much of this level has been discussed in previous sections. One additional point is that the stereo matching algorithm has been implemented on the benchmark configuration as a parallel, asynchronous

process. In this way, each Flexible Processor performs its given task in the algorithm individually and simultaneously with the other processors; when it is finished, it waits on a flag or interrupt for more data to perform its task again.

Previous Flexible Processor systems have been designed to operate in a synchronous mode. That is, the microprograms have been written with the exact number of instructions in each processor such that all processors are synchronized by the inherent instruction timing. In this way, when one processor is ready to send data to another processor, this second processor is always ready to receive it. Clearly, in this synchronous approach there is no need for the overhead required to perform handshaking between processors; one processor need not query a second processor as to his readiness and the second processor doesn't have to spend time to reply that it is ready or still busy, as in the asynchronous approach.

Thus, the synchronous approach can conceivably run faster than the asynchronous approach, but there is much more flexibility inherent in the asynchronous approach. Here, additions to the microprograms or changes to the basic tuning parameters, such as increases in patch sizes which require longer running loops, do not upset the timing of the entire algorithm. Thus, there is certain modularity in this type of parallelism; one module can be altered without significantly affecting the other modules.

The third level of parallelism, the systems level, exists if Flexible Processors are configured into separate, but duplicate, processing channels. Here, the microprograms are the same for each channel but the algorithm is run in parallel over different sections of digital image data. For example, if there are four processing channels available, then the imagery can be partitioned into four adjacent sections such that one channel processes one section simultaneously with the other channels and sections. Theoretically, then, a four channel system can realize a fourfold increase in total throughput rate over a one channel system. However, additional overhead is incurred if it is necessary to link the parallel channels together with parametric data. Due to the limited number of Flexible Processors

available for the benchmark, the block matching algorithm has been implemented in a one channel configuration.

7.1 IMPACT ON ALGORITHM DEVELOPMENT

There are some basic operational differences between the parallel implementation of the stereo matching algorithm on the Flexible Processors and the general purpose implementation of the algorithm on the CDC 6400. Most of these differences have arisen from the fact that a basically sequential algorithm has been converted for parallel implementation.

To illustrate this consider Figure 7-1 which shows the order of patch processing in the sequential algorithm. For the case illustrated, there are four patches per column to be matched. The number in each patch denotes the order in which the patch is completely processed in relation to the other patches. In other words, in the sequential algorithm, which was the predecessor of the benchmark parallel algorithm, patch j of column i is processed completely before moving on to patch j+1, j+2, j+3, and so on. This is the natural sequence of processing for a sequential machine such as the CDC 6400 and for a sequential language such as Fortran.

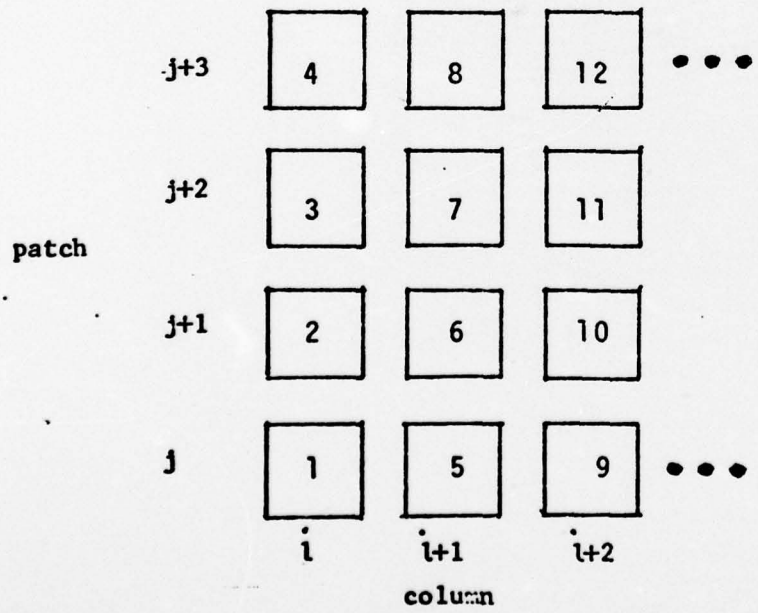


Figure 7-1. Sequential Implementation

Now, as a result of the sequential nature of the algorithm, certain correlation strategies emerge. That is, the correlation of any patch can be facilitated by the algorithm's knowledge of what occurred during the correlation of the previous patches in the sequence. In particular, the parameters for shaping and correlating patch j+1 of column i+1, number six in sequence, are predicted from the already correlated patches j+1, i and j, i+1, numbers two and five in sequence, respectively. These are the nearest patch neighbors and provide the most valid predictions, particularly when the patches overlap.

If this sequential strategy was transferred directly to hardware capable of parallel processing, such as the benchmark configuration, the increase in speed and throughput over the sequential implementation would not be great. Parallelism would occur primarily on the first level, the instruction logic level. Processors would be assigned their individual tasks, but now would perform the tasks in sequence rather than simultaneously with the other processors.

Figure 7-2 illustrates the order of patch processing for the benchmark parallel implementation.

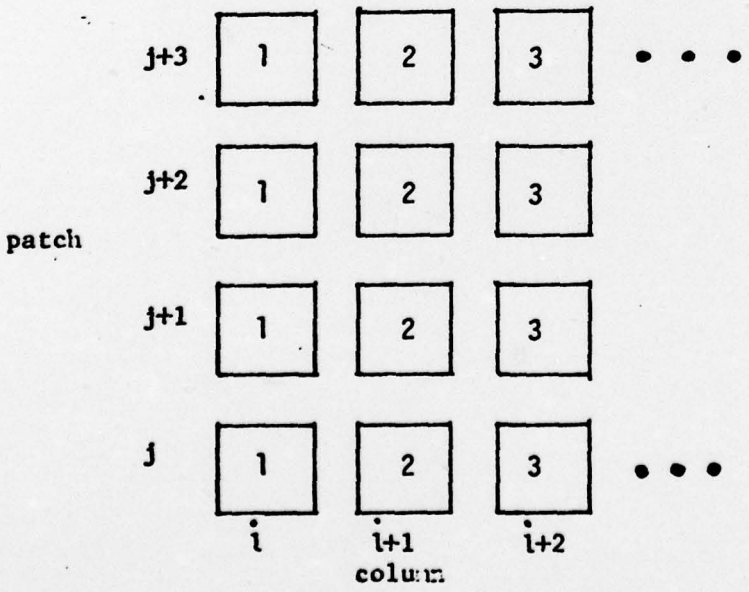


Figure 7-2. Benchmark Parallel Implementation

Here it is not possible to use patch j, i+1 in predicting parameters for patch j+1, i+1. All the patches of a column can be considered as being predicted and correlated simultaneously. Therefore, valid predictions can only be derived from the preceeding column. In this case the nearest neighbors to patch j+1, i+1 are, first, patch j+1, i, then also patches j, i and j+1, i.

Thus, a tradeoff exists in this particular algorithm case involving correlation strategy and parallel implementation. The effects of the strategy change have proven to be rather image-dependent. That is, the parallel strategy is less effective than the sequential strategy in certain hard-to-correlate areas of stereo scenes. But these areas can generally be handled by adjustments to the basic algorithm tuning parameters.

But there is a hypothetical alternative to the basic tradeoff that combines the strategy of the sequential implementation with the advantages of the parallel implementation. The order of patch processing for this alternative is illustrated in Figure 7-3.

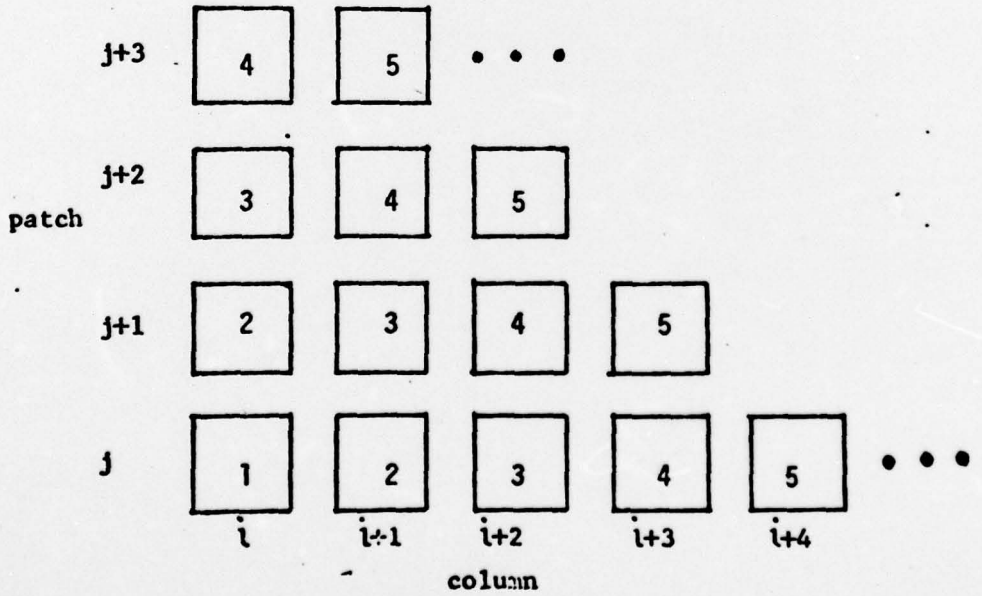


Figure 7-3. Hypothetical Parallel Implementation

Instead of a column-oriented parallelism, there arises a diagonal parallelism that spans across many columns and many scan lines of image data. One can see that as the number of patches per column increases, so does the complexity of the algorithm overhead and the data structures needed to achieve the parallelism. So, while the basic tradeoff involving correlation strategy and parallel implementation has been solved, new and possibly more serious tradeoffs emerge involving overhead complexity and time, number of parallel processors, and the size of local and bulk memory space. There are also many ramifications regarding internal algorithm design.

7.2 CONCLUSION

The above discussion has been presented as a particular algorithm example of the more general class of considerations that are involved in the design and implementation of parallel image processing algorithms. For future development, the trend in image analysis algorithms, such as the benchmark stereo matching algorithm, is toward more algorithm sophistication. This sophistication involves the use of numerous data-dependent strategies within a single algorithm and the use of dynamic decision-making procedures that determine which strategies to apply to given areas of imagery. In addition, the volume of image data to be processed and the required throughput rates are dictating a more widespread move toward parallel array processing. Yet, decision-making procedures are generally sequential in nature; parallel processors then, must wait for decision data to be generated by neighboring processors before proceeding with their tasks. What seems to be called for then, is a higher level of cleverness on the part of algorithm designers so that algorithms, from their very beginning, may be designed, tested, and developed more with parallel implementation in mind. In this way, better solutions will emerge for handling the complex trade-offs that are part of parallel processing technology.