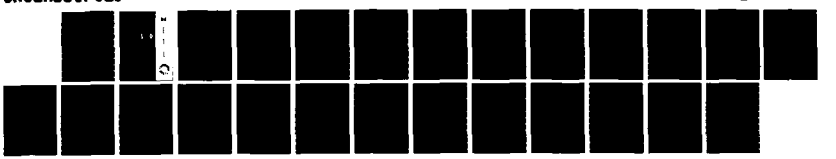
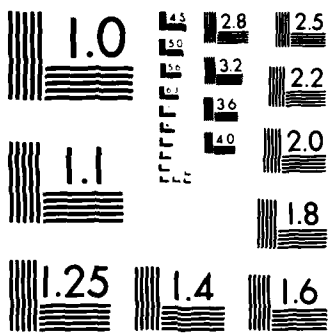
AD-A171 561 A PROGRAMMING ENVIRONMENT FOR PARALLEL VISION ALGORITHMS(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE C BROWN AUG 86 ETL-0433 DACA76-85-C-0001 1/1

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART

ETL-0433

(2)

A programming environment for parallel vision algorithms

AD-A171 561

Christopher Brown

DTIC
ELECTE
SEP 1 0 1986
S D
D

University of Rochester
Computer Science Department
Rochester, New York 14627

August 1986

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

Prepared for
U.S. ARMY CORPS OF ENGINEERS
ENGINEER TOPOGRAPHIC LABORATORIES
FORT BELVOIR, VIRGINIA 22060-5546

DTIC FILE COPY

and
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
1400 WILSON BOULEVARD
ARLINGTON, VIRGINIA 22209-2308

86 9 10 040

E T L

U.S. ARMY ENGINEER TOPOGRAPHIC LABORATORIES
ETL

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30 1986

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S) ETL-0433
6a NAME OF PERFORMING ORGANIZATION University of Rochester 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION U.S. Army Engineer Topographic Labs
6c ADDRESS (City, State, and ZIP Code) Computer Science Department Rochester, New York 14627	7b ADDRESS (City, State, and ZIP Code) Research Institute Fort Belvoir, VA 22060-5546
8a NAME OF FUNDING / SPONSORING ORGANIZATION DARPA 8b OFFICE SYMBOL (If applicable) ISTO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DACA76-85-C-0001
8c ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 22209-2308	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62301E PROJECT NO: TASK NO: WORK UNIT ACCESSION NO:
11 TITLE (Include Security Classification) A PROGRAMMING ENVIRONMENT FOR PARALLEL VISION ALGORITHMS	
12 PERSONAL AUTHOR(S) Brown, Christopher	
13a TYPE OF REPORT Annual 13b TIME COVERED FROM 85/2 TO 86/2	14 DATE OF REPORT (Year, Month, Day) 1986 August 15 PAGE COUNT 24
16 SUPPLEMENTARY NOTATION	

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
09	02		Parallel processors Butterfly computer Computer vision
17	08		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

During the first year of the award period, the Computer Science Department of the University of Rochester has pursued three main lines of work: systems support algorithms, Butterfly programming environment, and vision applications. Today's multiprocessor computer architectures are not efficiently programmed or even conceptualized with standard computer languages, and their operating systems and debugging tools are also challengingly different. The University of Rochester is doing work in the area of tools for controlling large-grain parallelism, as one finds in a distributed multiprocessor application like the Autonomous Land Vehicle, or in tightly coupled processors like the Hypercube or the Butterfly Parallel Processor.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Rosalene M. Holecheck	22b TELEPHONE (Include Area Code) (202) 355-2769 22c OFFICE SYMBOL ETL-RI

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Annual Technical Report
DARPA Contract DACA76-85-C-0001
A Programming Environment for Parallel Vision Algorithms
Christopher Brown, P.I.
University of Rochester
11 August 1986

1. Overview

During the first year of the award period, the Computer Science Department of the University of Rochester has pursued three main lines of work: systems support algorithms, Butterfly programming environment, and vision applications. The work under the Strategic Computing grant meshes nicely with the work done under our DARPA Image Understanding grant. The IU grant supports more fundamental work whose technological implications are explored under the SC grant. The SC grant is mainly concerned however with rendering today's Multiprocessor computer architectures useful. Often work is jointly funded, and the references at the end of this report includes some work done with the help of other funding. Some of this related work is described briefly below.

Today's multiprocessor computer architectures are not efficiently programmed or even conceptualized with standard computer languages, and their operating systems and debugging tools are also challengingly different. Of all the SC Vision contractors, Rochester seems to be doing the most work in the area of tools for controlling large-grain parallelism, as one finds in a distributed multiprocessor application like the ALV, or in tightly coupled processors like the Hypercube or the Butterfly Parallel Processor.

The references show two new technical report series. The *Butterfly Project Report* and *Hierarchical Process Composition Project Report* series document our work on the Butterfly Parallel Processor and in the HPC computational model.

2. The Butterfly and the Uniform System

The University of Rochester operates three Butterfly configurations with 3, 16 and 120 processors. These machines were procured with the help of an NSF Coordinated Experimental Research grant. Each processor consists of a 68000-based microcomputer connected to a bit-slice microengine. The microengine in turn is connected to local memory (one megabyte per processor), local i/o (synchronous and asynchronous serial lines, plus an optional multibus), and a high-speed switch. The microengine maps all memory references made by the 68000 into either local memory, local i/o, microengine control registers or (via the switch) some other processor's local memory. Remote memory references take two to four microseconds in the absence of contention, compared to half a microsecond for local references. Alternatively, the 68000 can direct its microengine to execute DMA transfers between local and remote memory at 4

QUALITY INSPECTED 1

y Codes
d / or
ecial
A-1

megabytes/second. The operating system supports multitasking on each node and provides a variety of microcode-assisted message passing and synchronization primitives.

For much of our Butterfly vision work we use the Uniform System, a software package that restricts the generality of the machine a great deal but simplifies coding. When a Uniform System program is initialized it starts a special process called a *task demon* on every processor on the machine. It then allocates 12 megabytes of memory (taking equal amounts from the local memory of each processor) and arranges for all processes to map that memory into the same place in their address spaces. This makes it possible for them to pass pointers to each other. The original process can call *generator* functions to place task descriptions on a global agenda, where they are picked up and executed by the task demons. Tasks can call generators recursively. The generators are typically used to simulate parallel FOR loops. For example, a task description might consist of a work function and an index; task demons would atomically read and decrement the index and call the work function with the result of the read. Uniform System applications tend to have a SIMD flavor, but the system differs from true SIMD machines in that the "single instruction" performed can be an arbitrary function call.

3. Systems Support Algorithms

The work by Sanchis [1986] and Newman-Wolfe [1985; 1986] is actually theoretical computer science brought to bear on MIMD architectures. This research has contributed algorithms for enhancing reliability and reducing contention on the butterfly switch, and on several other topologies that exist in current MIMD computers.

Liudvikas Bukys has provided vital system support and applications coding throughout the funding period. His contributions include network software, operating system installations and updates, a fast union-find algorithm useful in vision applications for region-growing and edge-linking, and general expertise with all levels of the Butterfly system.

4. Systems Research and Programming Environments

Our systems research has concentrated on three main areas: Basic building blocks, Programming environments, and Programming models. [BPR series, HPC series, and papers by LeBlanc, Scott, Finkel et al, Friedberg, Ohkami and Baldwin].

The *basic building blocks* are individual pieces of work that represent improvements to the existing Butterfly systems. They can be incorporated in later software systems or can be pursued for their own interest without the commitment to integrate them into a permanent system. Our work has included: Uniform System modifications, a parallel first fit memory allocation, measurement of Chrysalis primitives, SAR management to increase the number of available

processes, and the Modula-2 and C++ languages as natural extensions to the C environment.

A viable *parallel programming environment* is a basic goal of our SC Vision effort. To date we have worked on: a parallel file system, parallel compilation, debugging tools, and have developed and commissioned the SMP library and the LYNX programming language.

4.1. Structured Message Passing

SMP is a model of parallel programs that consists of process families whose members are created and destroyed together, and interprocess communication, within a family, based on asynchronous message-passing according to a fixed communication topology. There is a dynamic hierarchy of process families. SMP has been implemented [Leblanc, BPR-8]. The goals of SMP are: A user-level programming environment independent of the number of physical processors, Reduced coding overhead associated with creating a set of processes, Communication connections between processes according to a given interconnection pattern and naming scheme, Investigation of the utility of asynchronous message-passing as the primary form of interprocess communication within the Butterfly, and Comparison of message-passing with shared memory for various applications.

SMP offers a message-based alternative to the Uniform System package. Its heavyweight process model complements the lightweight process model of Modula-2. It gives us additional experience with a dynamic hierarchy of processes that use asynchronous message-passing.

The user interface is a handful of subroutine calls.

```
/* Create a family of communicating processes */
SMP_family_id  SMP_Spawn (topology, codef, argvf, data)

/* Destroy all processes in a family */
void SMP_Kill (family_id)

/* Send a message to a set of processes in the same family */
int  SMP_Send (family, dests, num_dest, buffer, size)

/* Receive a message from a process */
int  SMP_Receive (family, srcs, num_srcs, buffer, size, sender)
```

Additional library routines can be used to create predefined interconnection topologies or query a particular topology.

The Butterfly implementation of SMP message-passing is as follows: message buffers are implemented using shared memory objects; buffers are written by a

single process (the sender), but read by many; a count of intended recipients is associated with each message; message buffers are dynamically mapped into address spaces using a SAR cache; In our experience so far, SMP is easy to learn, its performance is very good, and the SAR cache idea is useful in practice. In the future we plan to explore new applications, to experiment with message-passing implementations, to integrate SMP with Modula-2, and to let SMP cooperate with Instant Replay.

4.2. Instant Replay

Debugging parallel programs is difficult. There are multiple threads of control, there is non-repeatable behavior due to timing variations, there is wide-ranging granularity of communication, and a lack of multiprocess debugging tools. In this work we assume: programs are communication-intensive, there is no direct microcode support for debugging, and programs use full Chrysalis capabilities. Our goal is to develop a flexible monitoring system that provides Instant Replay of parallel programs with minimal impact on program performance, minimal information collected, distributed data collection, independence from particular machine models, for both loosely-coupled and tightly-coupled domains Instant Replay has been implemented [Leblanc and Mellor-Crummey, in preparation].

The crucial observation is that data values in a shared object depend only upon: initial values in the object, the deterministic nature of the processes operating on the object, the underlying virtual machine, the inputs to processes from the external environment, and the relative order of operations on the object. Thus Instant Replay saves the relative order of events during program execution rather that the data associated with each event. Each time a shared object changes state, it is assigned a new version number. Each time a shared object is accessed, that fact is recorded on an object history tape. Each process records the version number of every shared object it accesses. For debugging, each process replays its own history tape. Each process that attempts to access a shared object must wait until the correct version of the object is available.

A prototype SMP implementation with Instant Replay has been built. The relevant actions for Send Message are: find buffer, Acquire Write Lock (buffer), copy message into buffer, set number of recipients, Release Write Lock (buffer). For Receive Message they are: Acquire Read Polling Lock (iterator), poll incoming message buffers, copy message into user area, Release Polling Lock, Acquire Write Lock (buffer), decrement number of recipients, Release Write Lock (buffer). In our experience so far, Instant Replay adds minimal overhead to the running time of a program. It is a practical tool that does not produce excessive contention. There are some locks involved, and it is their placement that the efficiency of the system resides. The rules are simple: lock concurrent structures (not shared structures), and optimize locks on idempotent (read) operations. In the future we shall work on: code efficiency improvements, proof of correctness, empirical studies with different programming models, automatic instrumentation, integration into programming environment, exploring effect on programming

cycle.

4.3. LYNX

The Butterfly implementation of LYNX is complete. A partial implementation was made publically available on 15 March. Exception handling was installed on 12 June. Pointers were installed on 14 July. The exception-handling features resemble those of Ada. The pointer facilities allow LYNX processes to share access to blocks of Butterfly memory. Exceptions can be bound to Chrysalis throw codes in such a way that throws in external C routines propagate back into LYNX programs as bona-fide exceptions.

Library packages have been written to support: storage of links in the Chrysalis name table, access to environment variables, pointer-based shared object support, screen-oriented output.

Utility programs provide access to files on the host machine and support interactive manipulation (from the Butterfly shell) of name-table links.

We have begun actively to encourage the growth of a user community. We have built a distributed game-tree searching program and are in the process of benchmarking it. We should be able to report on the results, together with further applications, in the fall of 1986.

>From an initial time of 5.9 milliseconds (best case), we have reduced the cost of message passing to 2.4 ms per remote operation. We are considering protocol optimizations that should bring the cost to well below 2 ms.

A detailed language rationale for LYNX will appear in the December 1986 issue of *IEEE Transactions of Software Engineering.* A technical note on the LYNX type-checking mechanism will appear in the same journal at a later date. A paper comparing the Butterfly version of LYNX with two earlier implementations will be presented at the 1986 International Conference on Parallel Processing. Earlier versions of the language rationale and of the implementation paper were published as Rochester Technical Reports and Butterfly Project Reports. The implementation paper will appear in the Department's 1986-87 Research Review. A LYNX reference manual was published as BPR 7.

4.4. PSYCHE

We have begun work on an operating system for the Butterfly, under the joint direction of Profs. LeBlanc and Scott. The project is known as Psyche, and has as its goal the development of systems software to support a wide variety of parallel programming paradigms.

Psyche will

(1) be more convenient than Chrysalis,

(2) support multiple users on a single Butterfly,

(3) permit experimentation with shared memory, message passing, and options in between, and

(4) permit well-structured communication between pieces of an application that use different paradigms.

We are convinced that no one model of parallelism will prove appropriate for all applications. Some algorithms are easier to implement with fully-shared memory. Others are most clearly conceived with message passing. Still others need an option, such as monitors, that falls somewhere in-between.

A major thrust of our work so far has been the comparison of solutions to common problems under various programming models (see, for example, BPR 3). We are fortunate with the Butterfly to be using hardware that lacks a built-in bias toward any one of these approaches. We hope with Psyche to exploit this lack of bias to develop an operating system that allows each application, or *part* of an application, to be written under the programming model most appropriate for its own particular needs. We expect Psyche to provide simple, well-defined mechanisms for interaction between pieces of code that employ different models.

The fundamental concept in Psyche is the **realm**. On the Butterfly, a realm will be a memory object (or set of objects) with an associated **protocol** that governs its access. There will be a many-to-many relationship between processes and realms: each realm may be shared by an arbitrary number of processes and each process may have access to an arbitrary number of realms. Psyche will provide mechanisms for creating, destroying, and managing access to realms. In effect, it will allow a program to bind access protocols and protection mechanisms to sets of memory objects.

Examples of access protocols for realms include:

(1) Pure shared memory in the style of the BBN **Uniform System**. A single, large, static realm would be shared by all processes. The access protocol would permit unrestricted reads and writes of individual memory cells.

(2) Connection-less message passing. Each message would be a separate realm. To send a message one would make the realm accessible to the receiver and (probably) inaccessible to the sender.

(3) Connection-based message passing, in the style of **LYNX**. Each communication channel (link) would be accessible to two processes and would contain buffers through which they could communicate.

(4) Monitors. Each realm would have access routines and a monitor lock. The realm protocol would insist that the access routines acquire the monitor lock before execution.

(5) Path expressions. Analogous to monitors, each realm would have access routines and an access protocol that would enforce the ordering rules described by path expressions.

One major issue in the design of Psyche will be protocol enforcement. Our goal is to make the probability of *accidental* misuse of a domain acceptably small. We have no intention of making it impossible. Protection mechanisms at our disposal include

(1) Manipulation of memory maps to make domains unaddressable.

(2) Manipulation of protection bits to make domains unreadable, unwritable, or unexecutable.

(3) Modification of compilers to make inappropriate domain operations unexpressible.

The first two options would be much more attractive in the presence of the 68020 Butterfly upgrade. In addition to the oft-cited advantages of increased speed and hardware floating point, the 68020s would permit restarting of instructions, allowing us to handle protection faults and implement virtual memory.

4.5. CONSUL

The CONSUL project is an attempt to simplify the use of multi-processor computers for general-purpose programming through automatic detection of parallelism in programs. Current programming techniques for multi-processors require programmers to worry about two related but distinct issues: how to express a solution to their problem as a program, and how to partition this program into parallel pieces. Multi-processor computers will never be as easy to program as sequential ones until programmers are freed from the need to parallelize programs manually. We believe that the best solution to this problem is to develop compilers that will automatically detect and exploit parallelism in programs that have not been explicitly parallelized by their authors. Unfortunately, automatic parallelization is a difficult problem that has so far resisted any general solution. One of the main reasons is that the source languages people are trying to parallelize are inadequate. Standard imperative languages rely on side-effects to maintain the state of a computation, a problem that is compounded by aliasing (the same piece of state information can have many distinct names). These features make imperative languages impossible to parallelize except in a few limited areas (e.g., the extremely regular code found in scientific computations). Declarative languages, which are generally free of side-effects and have a more tractable mathematical foundation (important in reasoning about both programs written in them and legal ways of compiling those programs), are more promising starting points for automatic parallelization. Our research is thus focussed on the compilation of a particular class of declarative language into a form that can be efficiently executed on multi-processors such as the Butterfly.

In late 1985 we realized that {it constraint languages} were promising ones with which to work. A constraint language is one in which programs consist of sets of relations between inputs, outputs, and (possibly) intermediate values, such that the relations hold if and only if the output values are correct for the inputs. Note

that there is a close correspondence between constraint languages and logic languages: any relation in a constraint program can be replaced by a predicate that tests whether that relation holds, and any predicate in a logic program defines a relation between its arguments. There is, however, an important qualitative distinction between our prototype constraint language and other logic languages: We provide a much richer set of primitive relations than other logic languages, in the belief that doing so makes the expression of general algorithms and their potential parallelism more natural. The cost of our richer set of primitives is a more elaborate compiler, as discussed below.

The ultimate goal of our research is to show that constraint languages are a practical tool for programming multi-processors containing several hundred relatively powerful processors. Achieving this goal requires solving two key problems. The first is to determine the features that a general purpose constraint language should have; the second is to show that such a language can make effective use of a parallel computer. During the Winter and Spring of 1986 we defined a language called CONSUL that demonstrates our solution to the first problem. We are now conducting a series of experiments intended to test a primitive dialect of CONSUL on a variety of programs and to characterize the parallelism that it makes available in each. These experiments will support our contention that CONSUL is suitable for general purpose programming, and will direct us to the richest sources of parallelism in the language. A later phase of the CONSUL project will address the second problem by developing compilers that can exploit this parallelism on a real multi-processor (the Butterfly).

The formal foundation for CONSUL is axiomatic set theory. Thus the fundamental data type is the set, and the fundamental operators are the logical connectives and quantifiers. However, a number of abstractions are built into the language to make it more palatable to programmers than raw set theory. In particular, the built-in data types include familiar ones such as sequences, integers, characters, et cetera. Each of these types can be given a set-theoretic definition, but programmers generally need not be aware of it. One consequence of the formal basis of CONSUL that can be important to programmers, however, is that relations, being sets, can be treated as data, and vice versa. This feature allows the language to include higher-order relations in a natural way. Each built-in data type is associated with built-in relations that correspond to common operations for that type. Thus CONSUL provides simple comparisons, arithmetic relations between integers, and so forth as language "primitives". Again, the fact that these operations are not really primitive to the underlying set theory is invisible to users. The built-in relations can be composed into more complex ones using the logical connectives "and", "or", and "not", with their standard meanings, and the quantifier "for all". "For all" is particularly useful as a way of mapping relations over sets in complex ways. (The existential quantifier is also available, but is mainly used just to declare variables and their scopes.)

Over the past few months (Summer 1986) we have been developing a software system that lets us estimate the parallelism available from CONSUL

programs. This system consists of a crude interpreter and a compactor. The interpreter's main purpose is to note when each relation in a CONSUL program can be satisfied and what variables are defined in the course of doing so. This information is written to a trace file, which is later compacted into a maximally parallel form by the compactor. Because the traces are taken from actual CONSUL programs in execution, the parallelism found by the compactor is "oracular" (see Nicolau and Fisher, "Using an Oracle to Measure Parallelism in Single Instruction Stream Programs", 14th ACM SIGMICRO Microprogramming Workshop, Oct. 1981) in the sense that a real compiler could fully exploit it only if it had perfect information about the object program's run-time behavior. Our results thus indicate the upper bound on the parallelism that can be derived from CONSUL programs. In the course of developing the interpreter (and programs to run on it) we have also considerably refined our notion of the proper semantics for CONSUL. For example, until early August the mathematical underpinnings of CONSUL were only intuitively defined --- the decision to unify and formalize them set-theoretically is a very recent one, whose full implications we are still evaluating.

We expect to complete the analysis of the parallelism experiments by the end of 1986. Starting in 1987, we will turn our efforts to compiling CONSUL into some form that can execute on a Butterfly (at least initially, this form will be some high-level language with its own Butterfly compiler, for example, Butterfly Common Lisp, LYNX, or C). Designing the compiler will be a very challenging project. In designing CONSUL, we have deliberately avoided the semantic compromises made to ease implementation in languages like Prolog. Thus the theoretical complexity of solving the relations in a CONSUL program is worse than in related languages. None the less, we expect that we can produce a running, parallel implementation of CONSUL. Experience with Prolog (supported by our own experiences as we begin to think about writing real CONSUL programs) suggests that people write declarative programs in fairly stylized ways that do not push the theoretical complexity limits of the solution process. Thus we do not need to be unduly concerned about the complexity of executing a CONSUL program. We believe that a "smart" compiler (in this case a "smart" compiler means one with a good symbolic algebra facility) can compile out much of the searching for solutions that current logic languages require. Finally, techniques for satisfying constraints using only local information have been demonstrated (see Steele, "The Definition and Implementation of a Computer Programming Language Based on Constraints", Ph. D. Dissertation, MIT Dept. of Electrical Engineering and Computer Science, Aug. 1980), and we hope to be able to adapt these techniques for use in CONSUL.

5. Vision

Much of our vision work is supported under the DARPA Image Understanding Program, but SC Vision is also a primary funding source for much of our vision work. Papers in the references by Aloimonos, Ballard, Bandopadhay, Brown, Cooper, Hinkelman, Hollbach, Narayanan, Sher, and Swain give details.

The work of two recently graduated Ph.D. students merits special mention. Aloimonos' work has centered on the robust and reliable computation of intrinsic images, or physical parameters of the scene. He has invented several new techniques, and his method has been to add information sources rather than to rely exclusively on apriori constraints (such as smoothness). His work has mainly been in the domains of multiple frame vision (stereo, motion) and in texture. Bandopadhay has also been working in the domain of motion. His work has been to apply clustering to the motion segmentation and egomotion problem, and to notice that proprioceptive feedback from tracking stationary points can work with vision to make the egomotion calculations easier. This tracking work is the scientific motivation for our robotic hardware, which consists of two cameras on a "robot head". With this setup we hope to investigate real-time vision with the Butterfly hardware.

5.1. BIFF: A Butterfly Vision Library

Tom Olson and Liud Bukys have constructed a parallel version of the IFF vision library written at the University of British Columbia under the direction of Prof. Havens. IFF is a file organization for images, and an associated set of image processing and vision utilities, something like SPIDER or GIPSY. IFF programs are written as UNIX filters, and the system uses UNIX pipes to concatenate operations. This is a slow way to go about things but is very modular and good for interactive use. BIFF, the parallel version, is much faster, both through capitalizing on the innately parallel nature of many low-level vision operations, and through use of the large memory on the individual butterfly nodes to achieve "in-core" files that can be passed from process to process quickly through memory mapping. We expect BIFF to be a useful tool and to expand in the future [Olson, BPR in preparation].

5.2. Segmentation on the BBN Butterfly Multiprocessor

Tom Olson has constructed an advanced program under the Uniform System to do segmentation. We are interested in the general problem of combining the outputs of low-level vision processes to produce robust interpretations of large classes of input images. In addition, we want solutions that make efficient use of large-scale parallel hardware. In order to study these issues we have chosen a particular well-studied problem (2-d segmentation) for implementation on the BBN Butterfly Multiprocessor. To date we have been more concerned with communications and systems aspects of the combination than with the mathematical aspects of cooperating constraints or evidence combination.

Two features of the Uniform System are particularly important for the design of our segmentation system. The first is that load balancing is stochastic; the machine will be used most effectively if tasks are numerous and have small execution times with low variance. The second is that critical data structures are kept in a global shared memory which ignores the distinction between remote and local memory. Since individual memories have limited bandwidth, it is advisable to scatter heavily used data structures as randomly as possible across the shared address space. Caching local copies of read-only data also helps.

5.2.1. The Segmentation Problem

The segmentation problem has a number of characteristics that make it a good vehicle for studying integration strategies. It can be approached on many levels, from low level (color, texture and gradient) to intermediate (shape) and high level (semantic checking of region labels and geometric relationships). The literature contains a large number of algorithms for segmenting based on this or that low-level feature, most of which are relatively straightforward to implement. This was important to us because we wanted to concentrate on systems and integration problems rather than on developing innovative low-level processes.

5.2.2. Our Approach

Our program works by recursively splitting regions until all regions satisfy some termination criteria. Users of the system must provide a set of functions called *experts* which take as their argument some region and generate a proposed segmentation of that region. The user also provides a *reconciling* function which integrates a set of proposed segmentations into a single proposal, which the main program then executes. Users are responsible for parallelizing their own functions. The main loop of the program can be summarized as follows :

```
agenda := original image
while (agenda is not empty)
        parallel-for (every region on agenda)
                parallel-for (every user-supplied expert)
                        apply the expert to generate a proposed segmentation
        parallel-for (every region on agenda)
                apply the reconciler to eliminate all but one proposal
                if # proposals is zero, put the region on a terminal list
                else execute the proposal and put results on agenda
        end
end
```

We claim that with minor changes a large class of currently used segmentation algorithms can be fit into this model. Among its defects are that a) there is no provision for merging and b) reasoning based on more than one region (eg based on connectivity) is forbidden. These restrictions are unfortunate, but they permit

the top-level program to avoid many locking and concurrency control problems.

5.2.3. Progress to date

Our current implementation has only one expert function, a grey-level histogram splitter loosely based on PHOENIX, the multispectral segmentor of Shafer and Kanade[1]. The process of generating a proposed segmentation breaks down into the following stages :

1) compute the grey-level histogram of the image
2) apply heuristics to partition the histogram into a set of intervals
3) back-project the intervals onto the region to generate a set of bitmaps
4) perform binary-image smoothing on each bitmap to eliminate very small or very thin regions
5) find four-connected regions in each bitmap and collect them into a list; this is the proposal.

Stages 1, 3, and 4 have been made highly parallel, essentially by performing parallel FOR loops over scan lines of the image and bitmaps. Stage 2 is serial but depends only on the grey-scale range of the image, and is in practice negligible. Stage 5 uses a serial sequential scan algorithm, applied in parallel over the set of bitmaps; it accounts for most of the running time of the algorithm.

Communication between manager, experts and reconciler is through shared memory. An expert is called on a region by the manager; it computes a proposed segmentation and stores it into a field in the region descriptor. The reconciler is called on a region and expects to find a (possibly null) list of proposals in the region descriptor. It reduces the length of the list to one or zero and returns; the manager then executes the remaining proposal, if any.

The manager and user functions are written in C under the Chrysalis operating system, using the Uniform System library to implement parallel loops. Users provide an initialized vector of pointers to the expert functions and the reconciler. They may optionally edit the region descriptor to incorporate any additional fields that they need. If the region descriptor is unchanged the manager will not need to be recompiled, but relinking is always necessary.

The manager makes use of BIFF, a locally developed image processing library for the Butterfly. In order to evaluate our parallelization efforts we use of modified version of the Uniform System that provides real-time graphic feedback on the status of the Butterfly processors (working, idle or blocked). The display portion of the status monitor runs on a SUN workstation and communicates with the Butterfly via TCP.

5.2.4. Experience

PARALLELIZATION - Our experience with the program described above has been that Amdahl's Law applies with a vengeance; that is, any non-parallel segment of the code quickly comes to dominate the running time of the system.

Many current blackboard designs view the blackboard as a server executing requests made by independent, long-lived client processes, and we agree that this model will be more manageable than ours for large systems. How should this model be parallelized? Our experience suggests that parallelization across requests will not be sufficient.

LOAD BALANCING - Sophisticated vision blackboards may offer to do non-trivial computations for their clients (eg find convex hulls of point sets, as in Stentz and Shafer[2]). At times, therefor, clients may be idle while the blackboard computes, while at other times the blackboard may be idle waiting for new requests. In a multiprocessor environment this may lead to an intolerable waste of cycles. What can be done about this?

MODELS OF COMPUTATION - Answers to the questions above depend heavily on what model of computation underlies the blackboard and its clients. The cooperating sequential process model provides a clean way to express the computation but in our opinion often fails to make efficient use of real parallel hardware. The pseudo-SIMD model provided by the Uniform System can be quite efficient (though it may not be extendable to large numbers of processors for hardware reasons), but makes writing well-structured programs difficult. Is there an intermediate solution, or can the defects of one of the approaches be remedied?

5.2.5. Future Work

The project described above has taught us quite a bit, and we intend to push it somewhat further, at least to the point of incorporating more experts and a nontrivial reconciling function. Ultimately, however, it is not the right kind of structure for a complete vision system. Segmentation, as many people have observed, is not a well-defined problem; you cannot say whether a given segmentor works until you have specified what you want to do with its output.

We envision a segmentor that is an integral part of a system for constructing intrinsic images. Segmentation could certainly use the outputs of e.g. depth maps, since depth discontinuities should signal region boundaries; but preliminary segmentations can also help identify places where the smoothness assumptions used in many intrinsic image calculations break down. Our ideal system would operate on stereo pairs of color images. Monocular processes would compute color constancy images and perform preliminary segmentation into plain and textured regions on the basis of color and brightness. The preliminary segmentation would define areas over which smoothness could be assumed; the smoothness assumption would then be used to constrain the stereo correspondance search problem and for shape from texture and shading. Reconciling the stereo, texture and shading images would give a depth map that could then be used to refine the segmentation.

References

[1] Shafer, S. and T. Kanade. Recursive Region Segmentation by Analysis of Histograms. *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, Paris, France, 1982.

[2] Stentz, A. and S. Shafer. *Module Programmers Guide to Local Map Builder for ALVan.* CMU Computer Science Department, January 1986.

6. Massively Parallel Models

The connectionist approach to programming and conceptualizing a parallel system has a powerful tool in the simulator that runs on the Butterfly [Fanty 1986]. An annotated bibliography of recent work is [Ballard , Brown, Dell, and Feldman 1985]. Some vision applications are under investigation using connectionist models (see papers by Ballard in the references).

7. Conclusion

In the period covered by this progress report the University of Rochester has gone a long way toward making the Butterfly Parallel Processor into a usable engine for complex computations. We foresee that our software will be taken up by other DARPA contractors with Butterfly or similar MIMD computers, and are anxious to bring that about. Our work involves a symbiotic merging of results, both from the literature and locally generated, from theory, artificial intelligence, and systems. At Rochester we are growing slightly, largely in the systems area, but in general are maintaining our relatively small size because we see the symbiosis is working, and letting us move ahead quickly. We are not by any means a large DARPA contractor in terms of money, but DARPA funding has been well-leveraged with other awards, and has been vital to our ability to do work that we believe is well-positioned scientifically and practically and that has a distinctive stamp.

Publications Produced under DARPA Strategic Computing Support, January 1985 - July 1986

Aloimonos, J., "Computing intrinsic images," Ph.D. thesis, Computer Science Dept., U. Rochester, September 1986.

Aloimonos, J., "Determining the illuminant direction," forthcoming TR, Computer Science Dept., U. Rochester, August 1986.

Aloimonos, J., "Shape and motion from contour, without point to point correspondence: general principles," TR 173, Computer Science Dept., U. Rochester, to appear, 1986; *Proc., IEEE Computer Vision and Pattern Recognition*, Miami, FL, June 1986.

Aloimonos, J., "Structure from motion: I) optic flow vs. discrete displacements; and II) Lower bound results," *Proc., IEEE Computer Vision and Pattern Recognition*, Miami, FL, June 1986.

Aloimonos, J. and A. Bandopadhay, "Perception of structure from motion: Lower bound results," TR 158, Computer Science Dept., U. Rochester, March 1985.

Aloimonos, J., A. Bandopadhay, and P. Chou, "On the foundations of trinocular machine vision," *Technical Digest, Topical Meeting of the Optical Society of America*, Lake Tahoe, April 1985.

Aloimonos, J. and A. Basu, "Determining the translation of a rigidly moving surface, with correspondence," TR 176, Computer Science Dept., U. Rochester, to appear, 1986; *Proc., IEEE Computer Vision and Pattern Recognition Conf.*, Miami, FL, June 1986.

Aloimonos, J., A. Basu, and C.M. Brown, "Coutour, shape and motion," *Proc., DARPA Image Understanding Workshop*, Miami, FL, December 1985.

Aloimonos, J. and P. Chou, "Detection of surface orientation and motion from texture I: The case of planes," TR 161, Computer Science Dept., U. Rochester, January 1985.

Aloimonos, J. and P. Chou, "Detection of surface orientation from texture," *Optic News*, September 1985.

Aloimonos, J. and I. Rigoutsos, "Detection of 3-D motion without correspondence: I) Planar surfaces: theory and experiments; II) Curved surfaces, theory," TR 178, Computer Science Dept., U. Rochester, December 1985.

Aloimonos, J. and I. Rigoutsos, "Determining the 3-D motion of a rigid planar patch without correspondence, under perspective projection," *Proc., Canadian Artificial Intelligence Conf.*, Montreal, 1986.

Aloimonos, J. and I. Rigoutsos, "Determining 3-D motion of a rigid planar patch, without correspondence, under perspective projection," *Proc., IEEE Workshop on Motion,* Charleston, SC, May 1986.

Aloimonos, J. and I. Rigoutsos, "Determining 3-D motion of rigid surfaces without correspondence," *Proc., AAAI 1986*, Philadelphia, PA, August 1986.

Aloimonos, J. and M.J. Swain, "Shape from texture," *Proc., 9th Int. Joint Conf. on Artificial Intelligence*, Los Angeles, CA, 926-931, August 1985.

Ballard, D.H., "Cortical connections: Structure and function," *Behavioral and Brain Sciences 9*, 1, 67-120, March 1986.

Ballard, D.H., "Form perception as transformation," TR 148, Computer Science Dept., U. Rochester, January 1986.

Ballard, D.H., "Interpolation coding: a model for the representation of metric information," TR 175, Computer Science Dept., U. Rochester, May 1986.

Ballard, D.H., "Task frames in visuo-motor coordination," *Proc., 3rd IEEE Workshop on Computer Vision*, Bellaire, MI, October 1985.

Ballard, D.H., "Transformational form perception," to appear in book to be published by Lawrence Erlbaum Assoc., 1986.

Ballard, D.H., C.M. Brown, G. Dell, and J.A. Feldman, "Rochester connectionist papers, 1979-85," TR 172, Computer Science Dept., U. Rocheter, November 1985.

Ballard, D.H. and H. Tanaka, "Transformational form perception in 3D: Constraints, algorithms, implementation," *Proc., 9th Int. Joint Conf. on Artificial Intelligence*, Los Angeles, CA, 964-968, August 1985.

Ballard, D.H., S. Tsuji, H. Tanaka, and M. Curtiss, "Parallel polyhedral form perception," *Proc., IEEE Conf. on Computer Vision and Pattern Recognition*, Spring 1985.

Bandopadhay, A., "Constraints on the computation of rigid motion parameters from retinal displacements," TR 168, Computer Science Dept., U. Rochester, October 1985.

Bandopadhay, A., "A computational study of rigid motion perception," Ph.D. thesis, Computer Science Dept., U. Rochester, September 1986.

Bandopadhay, A., "Perception of structure and motion of rigid objects," TR 169, Computer Science Dept., U. Rochester, December 1985.

Bandopadhay, A. and J. Aloimonos, "Perception of motion of rigid objects," TR 169, Computer Science Dept., U. Rochester, December 1985.

Bandopadhay, A. and J. Aloimonos, "Perception of rigid motion from spatiotemporal derivatives of optical flow," TR 157, Computer Science Dept., U. Rochester, March 1985.

Bandopadhay, A., and D.H. Ballard, "Visual navigation by tracking of environmental points," *SPIE Conf. on Artificial Intelligence*, Orlando, FL, March 1986.

Bandopadhay, A., B. Chandra, and D.H. Ballard, "Egomotion perception using active vision," *Proc., IEEE Conf. on Computer Vision Representation and Control*, Miami Beach, FL, June 1986.

Bandopadhay, A., B. Chandra, and D.H. Ballard, "Active navigation: tracking an environmental point considered beneficial," *IEEE Workshop in Motion Representation and Analysis*, Charleston, SC, May 1986.

Bandopadhay, A. and R. Dutta, "Measuring image motion in dynamic images," *IEEE Workshop on Motion, Representation and Analysis*, Charleston, SC, May 1986.

Bandopadhay A. and R. Dutta, "Measuring notion in dynamic images: a clustering approach," *6th Canadian Conf. on Artificial Intelligence*, Montreal, May 1986.

Brown, C.M., "Advances in Computer Vision," to appear in book to be published by Lawrence Erlbaum Assoc., 1986.

Brown, C.M., "Space-efficient Hough transformation for object location," in E. Wegman (Ed). *Statistical Image Processing and Graphics*. To appear, Marcel-Dekker, 1986.

Brown, C.M., J. Aloimonos, M. Swain, P. Chou, and A. Basu, "Texture, contour, shape and motion," submitted to *Pattern Recognition Letters*, September 1985.

Brown, C.M. and D.H. Ballard, "Vision: Biology challenges technology," invited article, *BYTE 10*, 44, 245-261, April 1985.

Brown, C.M., C.S. Ellis, J.A. Feldman, S.A. Friedberg, and T.J. LeBlanc, "Artificial intelligence research on the Butterfly multiprocessor," *Proc., Workshop on AI and Distributed Problem Solving*, National Academy of Sciences, Washington, DC, 109-118, May 1985.

Cooper, P.R., D.E. Friedman, and S.A. Wood, "The automatic generation of digital terrain models from satellite images by stereo," *36th Congress, Int. Astronautical Federation*, Stockholm; to appear, *Acta Astronautica, 1986*.

Fanty, M., "A connectionist simulator for the BBN Butterfly multiprocessor," Butterfly Project Report 2, Computer Science Dept., U. Rochester, January 1986.

Feldman, J.A. and C.M. Brown, "Recent progress of the Rochester image understanding project," *Proc., DARPA Image Understanding Workshop*, Miami, FL, December 1985.

Finkel, R.A., M.L. Scott, W.K. Kalsow, et al., "Experience with Charlotte: Simplicity vs. function in a distributed operating system," Computer Sciences TR 653, U. Wisconsin-Madison, July 1986; for *IEEE Workshop on Design Principles for Experimental Distributed Systems*, Purdue U., October 1986.

Friedberg, S.A., "A consistency protocol for highly available, replicated databases," submitted, *5th Symp. on Reliabiloity in Distributed Software and Database Systems*, 1986.

Friedberg, S.A., "Finding axes of skewed symmetry," *Computer Vision, Graphics, and Image Processing 34*, 138-145, 1986.

Friedberg, S.A., "Hierarchical processor composition," *Proc., 14th ACM CS Conf.*, February 1986.

Friedberg, S.A., "HPC coding style guidelines," Hierarchical Process Composition Project Report 1, Computer Science Dept., U. Rochester, May 1986.

Friedberg, S.A., "Interface structures," Hierarchical Process Composition Project Report 5, Computer Science Dept., U. Rochester, August 1986.

Friedberg, S.A., "Symmetry evaluators" (revised), TR 134, Computer Science Dept., U. Rochester, January 1986.

Friedberg, S.A., "User process--HPC interface--C language/UNIX host version," Hierarchical Process Composition Project Report 3, Computer Science Dept., U. Rochester, August 1986.

Friedberg, S.A. and G.L. Peterson, "An efficient solution to the mutual exclusion problems using weak semaphores," submitted, *Information Processing Letters*, 1986.

Friedberg, S.A. and D.H. Pitcher, "HPC IPC implementation--unmodified UNIX host version," Hierarchical Process Composition Project Report 2, Computer Science Dept., U. Rochester, June 1986.

Friedberg, S.A. and I. Rigoutsos, "Comments on Stony Brook MP," Hierarchical Process Composition Project Report 4, Computer Science Dept., U. Rochester, May 1986.

Hinkelman, E., "NET: A utility for building regular process networks on the BBN Butterfly parallel processor," Butterfly Project Report 5, Computer Science Dept., U. Rochester, February 1986.

Hinkelman, E., "Pattern: A computational approach to quantifying coating quality," Eastman Kodak Research Labs Technical Report, to appear, 1986.

Hollbach, S.C., "Tinker toy world; final report: the 2-d feature finder," forthcoming TR, Computer Science Dept., U. Rochester, 1986.

LeBlanc, T.J., "Shared memory versus message-passing in a tightly-coupled multiprocessor: A case study," Butterfly Project Report 3, Computer Science Dept., U. Rochester, January 1986; to appear, *Proc., 1986 Int. Conf. on Parallel Processing*, August 1986.

LeBlanc, T.J. and L. Bukys, "Getting started with the BBN Butterfly multiprocessor," Butterfly Project Report 1, Computer Science Dept., U. Rochester, September 1985.

LeBlanc, T.J. and R.P. Cook, "High-level broadcast communication for local area networks," *IEEE Software*, Special Issue on Experiences with Distributed Systems, 40-48, May 1985.

LeBlanc, T.J. and S.A. Friedberg," Hierarchical process composition in distributed operating systems," *Proc., 5th Int. Conf. on Distributed Computing Systems*, Denver, CO, 26-34, May 1985.

LeBlanc, T.J. and S.A. Friedberg, "HPC: A model of structure and change in distributed systems," TR 153, Computer Science Dept., U. Rochester, May 1985; *IEEE Trans. on Computers C-34*, 12, 1114-1129, 1985.

LeBlanc, T.J., N.M. Gafter, and T. Ohkami, "SMP: a message-based programming environment for the BBN Butterfly," Butterfly Project Report 8, Computer Science Dept., U. Rochester, July 1986.

Mukerjee, A. and D.H. Ballard, "Self-calibration in robot manipulators," *Proc., IEEE Workshop on Robotics*, April 1985.

Narayanan, N.H. and C.M. Brown, "Parallel stereo correspondence on the Butterfly Multiprocessor," forthcoming Butterfly Project Report, Computer Science Dept., U. Rochester, to appear, 1986.

Narayanan, N.H. and N. Viswanadham, "A methodology for knowledge acquisition and reasoning in failure analysis of systems," *Proc., Symp. on AI in Engineering*, Washington, DC, October 1985; to appear, *IEEE Trans. on Systems, Man and Cybernetics*, 1986.

Newman-Wolfe, R.E., "Communication issues in parallel processing," Ph.D. thesis, Computer Science Dept., U. Rochester, September 1986.

Newman-Wolfe, R.E. (Ed), "Proceedings of 1986 Open House," TR 184, Computer Science Dept., U. Rochester, March 1986.

Newman-Wolfe, R.E., "A set theoretic problem: a solution for the hypercube and its applications," TR 171, Computer Science Dept., U. Rochester, October 1985.

Ohkami, T. and D. Baldwin, "The SEEDS simulator: User manual and report," Computer Science Dept., U. Rochester, to appear, 1986.

Olson, T.J.,"Modula-2 on the BBN Butterfly Multiprocessor", Butterfly Project Report 4, Computer Science Dept., Univ. Rochester, January 1986.

Rigoutsos, I. and C.M. Brown, "Camera calibration," TR 186, Computer Science Dept., U. Rochester, to appear, 1986.

Sanchis, L.A., "Multiple-way network partitioning," TR 181, Computer Science Dept., U. Rochester, March 1986.

Scott, M.L., "Design and implementation of a distributed systems language," Ph.D. thesis, TR 596, U. Wisconsin-Madison, May 1985.

Scott, M.L., "The interface between distributed operating system and high-level programming language," TR 182, Computer Science Dept., U. Rochester, January 1986; Butterfly Project Report 6, Computer Science Dept., U. Rochester, March 1986; Proc., *1986 Int. Conf. on Parallel Processing*, St. Charles, IL, August 1986; *Computer Science/Engineering Research Review*, U. Rochester, September 1986.

Scott, M.L., "Language support for loosely-coupled distributed programs," TR 183, Computer Science Dept., U. Rochester, January 1986; *IEEE Transactions on Software Engineering*, Special Issue on Distributed Computing, to appear, December 1986.

Scott, M.L., "LYNX reference manual," Butterfly Project Report 7, Computer Science Dept., U. Rochester, March 1986.

Scott, M.L. and R.A. Finkel, "A simple mechanism for type security across compilation units," *IEEE Transactions on Software Engineering*, Special Issue on Distributed Computing, to appear, December 1986.

Sher, D.B., "Developing and analyzing boundary detection operators using probabilistic models," *Proc., ACM Workshop on Uncertainty and Probability in Artificial Intelligence*, August 1985.

Sher, D.B., "Evidence combination for vision, using likelihood generators," *Proc., DARPA Image Understanding Workshop*, Miami, FL, December 1985.

Sher, D.B., "Optimal likelihood generators for edge detection under Gaussian additive noise," TR 185, Computer Science Dept., U. Rochester, June 1986; *Proc., IEEE Conf. on Computer Vision and Pattern Recognition*, Miami, FL, June 1986.

Sher, D.B., "Template matching on parallel architectures," TR 156, Computer Science Dept., U. Rochester, July 1985.

Swain, M.J. and J.L. Mundy, "Experiments in using a theorem prover to prove and develop geometrical theorems in computer vision," *1986 IEEE Int. Conf. on Robotics and Automation*, San Francisco, 280-285, April, 1986.

Swain, M.J., "Algorithms," *Queen's Mathematical Communicator*, Queen's College, to appear, 1986.

Butterfly Project Reports:

1. "Getting started with the BBN Butterfly multiprocessor," by T.J. LeBlanc and L. Bukys, September 1985.

2. "A connectionist simulator for the BBN Butterfly multiprocessor," by M. Fanty, January 1986.

3. "Shared memory versus message-passing in a tightly-coupled multiprocessor: A case study," by T.J. LeBlanc, August 1986.

4. "Modula-2 on the BBN Butterfly multiprocessor," by T.J. Olson, January 1986.

5. "NET: A utility for building regular process networks on the BBN Butterfly parallel processor," by E. Hinkelman, February 1986.

6. "The interface between distributed operating system and high-level programming language," by M.L. Scott, March 1986.

7. "LYNX reference manual," by M.L. Scott, March 1986.

8. "SMP: a message-based programming environment for the BBN Butterfly," by T.J. LeBlanc, N.M. Gafter, and T. Ohkami, July 1986.

"Parallel stereo correspondence on the Butterfly multiprocessor," by N.H. Narayanan and C.M. Brown, to appear, 1986.

Hierarchical Process Composition Project Reports:

1. "HPC coding style guidelines," by S.A. Friedberg, May 1986.
2. "HPC IPC implementation--unmodified UNIX host version," by S.A. Friedberg and D.H. Pitcher, June 1986.
3. "User process--HPC interface--C language/UNIX host version," by S.A. Friedberg, August 1986.
4. "Comments on Stony Brook MP," by S.A. Friedberg and I. Rigoutsos, May 1986.
5. "Interface structures," by S.A. Friedberg, August 1986.

END

10-86

DTIC